

Net.Data



管理とプログラミングの手引き OS/390

バージョン 2 リリース 2

Net.Data



管理とプログラミングの手引き OS/390

バージョン 2 リリース 2

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、131ページの『付録D. 特記事項』に記載する一般情報をお読みください。

原 典： VNDT-2APG-01 (この資料番号ではオーダーできません)
Net.Data
Administration and Programming Guide for OS/390
Version 2 Release 2

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	vii
Net.Data について	vii
新機能	viii
バージョン 2 の新機能	viii
バージョン 2.2 の新機能	viii
本書について	ix
本書の対象読者	x
本書の例について	x
第1章 概要	1
Net.Data とは ?	1
Net.Data を使用する利点	2
第2章 Net.Data のインストールおよび構成	5
Net.Data の初期設定ファイルについて	5
Net.Data の初期設定ファイルのインストール	6
Net.Data の初期設定ファイルのカスタマイズ	6
構成変数ステートメント	8
パス構成ステートメント	15
環境構成ステートメント	18
言語環境のセットアップ	20
IMS Web の言語環境	21
SQL および ODBC の言語環境	21
DB2 との接続の管理	23
作業負荷管理の考慮事項	23
CGI と一緒に使用する Net.Data の構成	24
ICAPI あるいは GWAPI と一緒に使用するための Net.Data の構成	25
Java サブレットと一緒に使用するための Net.Data の構成	27
メッセージ・カタログを使用可能にする	28
Net.Data がアクセスするファイルおよびデータ・セットへのアクセス権の授与	29
第3章 ユーザー資産を保護する	31
ファイアウォールを使用する	31
ネットワーク上のユーザーのデータを暗号化する	32
認証を使用する	32
許可を使用する	33
Net.Data のメカニズムを使用する	33
Net.Data 構成変数	33
マクロ開発技法	34
第4章 Net.Data を起動する	39
CGI、ICAPI、または GWAPI を使用して Net.Data を起動する	39
マクロで Net.Data を呼び出す (マクロ要求)	40
マクロを使用しない Net.Data の起動 (直接要求)	43
Java Servlets によって Net.Data を起動する	48
MacroServlet を使用した Net.Data の起動	49
FunctionServletを使用して Net.Data を起動する	50
第5章 Net.Data のマクロ開発	53
Net.Data マクロの分析	54

DEFINE ブロック	55
FUNCTION ブロック	56
HTML ブロック	56
Net.Data のマクロ変数	58
識別子の効力範囲	59
変数の定義	60
変数の参照	62
変数の型	62
Net.Data の関数	70
関数の定義	70
関数の呼び出し	75
Net.Data 組み込み関数の呼び出し	75
マクロでの Web ページ生成	79
HTML ブロック	79
レポート・ブロック	80
マクロにおける条件付き論理のループ	85
条件付き論理: IF ブロック	85
ループ構成体: WHILE ブロック	87
第6章 言語環境の使用	89
Net.Data 提供の言語環境の概説	90
言語環境の呼び出し	91
データ言語環境	91
関係データベース言語環境	91
フラット・ファイル・インターフェース言語環境	101
IMS Web 言語環境	102
プログラミング言語環境	103
Java アプレット言語環境	103
Perl 言語環境	109
REXX 言語環境	112
システム言語環境	115
第7章 パフォーマンスを向上させる	117
Web サーバー API の使い方	117
Net.Data マクロ・キャッシング	117
ガイドラインおよび制約事項	117
マクロ・キャッシングの使用可能化	118
DB2 (OS/390 版) のメッセージを抑制する	119
言語環境の最適化	120
REXX 言語環境	120
SQL 言語環境	120
システムおよび Perl 言語環境	121
付録A. 参考文献	123
Net.Data テクニカル・ライブラリー	123
関連資料	123
付録B. Net.Data for OS/390 を構成し、DataJoiner をアクセスする	125
付録C. Net.Data サンプル・マクロ	127
付録D. 特記事項	131
商標	132

用語集.	133
索引	135

まえがき

Net.Data バージョン 2.2 をお買い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 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 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) の両方のデータ、ならびに Java、JavaScript、Perl、C、C++、あるいは REXX などのプログラミング言語で作成されたアプリケーションを使用して、動的な Web ページを作成することができます。

Net.Data は、Web サーバー・マシン上でミドルウェアとして実行されるマクロ処理プログラムです。ユーザーが作成する Net.Data アプリケーションはマクロと呼ばれ、Net.Data はこれを解釈して、ユーザーからの入力、データベースの現在の状態、他のデータ・ソース、既存のビジネス論理、およびマクロの設計にとり込むその他の要素に基づいてカスタマイズされた内容を持つ動的な Web ページを作成します。

要求は URL (uniform resource locator) の形式で Netscape Navigator あるいは 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実行するために Net.Data に送ります。Net.Data はマクロを見つけて実行し、ユーザーが作成した関数に基づいてマクロがカスタマイズする Web ページを生成します。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、あるいは REXX プログラム内にカプセル化する。
- DB2 などのデータベースにアクセスする

Net.Data は、この Web ページを Web サーバーに渡します。このページはネットワークを通してブラウザーに転送され、表示されます。Net.Data ファミリー・プロダクトの他のメンバーは、Windows NT、AIX、OS/2、AS/400、HP-UX、Sun Solaris、Linux、および Santa Cruz Operating System (SCO) の各オペレーティング・システムを実行するマシン上でも同様の機能を提供します。

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ように構成されているサーバー環境で使用できます。HTTP は、ブラウザーと Web サーバー間の対話のための業界標準のインターフェースです。また CGI は、Net.Data のようなゲートウェイ・アプリケーションを Web サーバーで呼び出すための業界標準のインターフェースです。これらのインターフェースでは、Net.Data を使用するのに自分の気に入ったブラウザーあるいは Web サーバーを選択することができます。また Net.Data は、パフォーマンス向上のために、さまざまなアプリケーション・プログラミング・インターフェース (API) をサポートしています。さらに、Net.Data は、サーブレットとして実行することもできます。

新機能

以下のセクションでは、Net.Data の新機能について説明します。

バージョン 2 の新機能

Net.Data for OS/390 バージョン 2 には、ユーザーのアプリケーション要件を満足する以下のようなパフォーマンスとスケーラビリティ機能が盛り込まれています。

- ICPAPI あるいは GWAPI 使用時の SQL 言語環境を介して確立された DB2 (OS/390 版) 接続の再使用
- ICAPI 環境での OS/390 用作業負荷管理プログラムとの統合
- マクロを使用しない Net.Data の呼び出し (直接要求)
- 生成された Web ページ内の、無駄になっている空白スペースの最小化
- OS/390 用 DB2 メッセージ・テキスト・ルックアップをう回する機能

Net.Data バージョン 2 ではまた、多数の機能上の強化が図られています。

- 言語環境面での機能強化では、ODBC 言語環境を使用したストアード・プロシージャの実行機能が挙げられます。
- マクロ言語環境は、次のものを含んでいます。
 - 注釈をどこでも追加する機能
 - ネストされた IF ブロック
 - WHILE ブロック
 - ストアード・プロシージャからの単一結果セットの受け取り機能
 - DBCS 対応のストリングおよびワード関数
 - ストアード・プロシージャ用パラメーター・リスト内の SQL 10 進数データ・タイプのサポート
- 他のコード・ページで符号化されたマクロからのデータで 1 コード・ページに符号化されたデータベースのデータをまとめ上げて Web ページを作成する機能。

バージョン 2.2 の新機能

Net.Data バージョン 2.2 では以下の機能強化が図られています。

- パフォーマンスとスケーラビリティの機能強化には、以下のものがあります。
 - マクロと組み込みファイルのキャッシュ機能
 - 大きな結果セットに対する表変数の処理の改良
 - ODBC 言語環境によって作成された DB2 (OS/390 版) 接続の再使用
 - Net.Data サブレットの使用時に SQL および ODBC 言語環境で作成された DB2 (OS/390 版) 接続の再使用
 - Net.Data の組み込み関数によるパフォーマンスの向上
 - 大きな結果セットを持つ表変数のスケーラビリティの向上
- マクロ言語環境では、以下の機能強化が図られています。
 - SQL および ODBC 言語環境を使用して、ストアード・プロシージャから複数の結果セットを受け取る機能

- 表処理のための新規の組み込み関数
- Net.Data アプリケーションで HTTP の Cookies を取得し、設定するための組み込み関数
- 結果セットのサイズを小さくするための START_ROW_NUM、DTW_SET_TOTAL_ROWS、および TOTAL_ROWS のサポート
- DTW_EXIT 組み込み関数を使用したマクロの即時終了機能
- DTW_SENMAIL 組み込み関数を使用した、マクロによる電子メール・メッセージの生成および送信機能
- WHILE ブロックにおける INCLUDE_URL ステートメントのサポート
- MACRO_FUNCTION 言語構造体を使用した、Net.Data のマクロ言語ステートメントだけで構成される関数のサポート
- HTML セクション名にピリオド (.) を含める機能
- Net.Data 初期設定ファイルにコメントを挿入する機能
- 関数内で作成される変数の効力範囲をローカルに変更する機能
- 関数呼び出しのパラメーター・リスト内のリテラル・ストリングにおける変数参照のサポート
- すべてのリテラル・ストリング内の 2 つの二重引用符を単一の二重引用符と解釈する機能
- ユーロ通貨記号のサポート
- Java サンプルを使用した Net.Data の呼び出し機能
- DTW_SHOWSQL 構成変数を使用して SHOWSQL 変数を使用不可にする機能 (デフォルトは、使用不可)
- DTW_DIRECT_REQUEST 構成変数を使用して Net.Data の直接要求を使用不可にする機能 (デフォルトは、使用不可)

本書について

本書は Net.Data の管理とプログラミングの概念について解説しています。Net.Data およびその構成要素の構成方法、機密保護の設計、およびパフォーマンスの向上についても解説しています。

いままでのプログラム言語やデータベースの知識を基に、ユーザーは Net.Data マクロ言語を使用したマクロの開発方法を学習します。本書では、DB2 データベースおよび IMS Web を使用する IMS トランザクションにアクセスし、Java、REXX、Perl をはじめとする、データにアクセスするためのプログラミング言語を使用する Net.Data 提供の言語環境の使用方法について学習します。

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。

サンプル Net.Data マクロ、デモ、および本書の最新バージョンの詳細な情報については、以下の World Wide Web サイトをご覧ください。(英語版)

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

本書の対象読者

本書は、Net.Data アプリケーションを設計し、開発する方々を対象としています。本書で説明する概念を理解するには、Web サーバーの働きについての知識、簡単な SQL ステートメントの理解、および HTML のフォーム・タグをはじめとする HTML のタグの知識が必要です。

SMP/E インストール情報については、*Program Directory for Net.Data for OS/390 Version 2 Release 2* を参照してください。

Net.Data マクロ言語、変数、および組み込み関数は、オペレーティング・システムの相違点と共に *Net.Data 解説書* で説明されています。

本書の例について

本書に記載されている例は、特定の概念を説明するために単純化されたものになっています。Net.Data の構成要素が使用されるすべてのケースが示されているわけではありません。例の中には、コードを追加しないと機能しない断片的なものもあります。

第1章 概要

インターネットにおけるほとんどの Web ページは静的な Web ページです。つまり、編集しない限りは変更されないページです。Web 上に “live” データとアプリケーション (現在の営業統計など) を組み込むには、Web サイトの開発者は通常、プログラムを作成します。このプログラムは、Web サーバーのミドルウェアとして Web ページを動的に構築します。この種のプログラムの作成は簡単ではありません。

Net.Data を使用すると、マクロ を使用して、対話式 Web アプリケーションを簡単に作成することができます。

本章では、Net.Data を Web アプリケーションで使用する利点について説明します。

- 『Net.Data とは ?』
- 2ページの『Net.Data を使用する利点』

Net.Data とは ?

Net.Data マクロを使用することにより、プログラミング論理の実行、変数のアクセスおよび操作、関数の呼び出し、およびレポート作成ツールの使用が可能です。マクロとは Net.Data マクロ言語構成要素、HTML タグ、Javascript、および SQL や Perl などの言語環境ステートメントが組み込まれているテキスト・ファイルです。Net.Data はマクロを処理して、Web ブラウザーで表示可能な出力を作成することができます。マクロは、単純な HTML と、Web サーバー・プログラムの動的な機能性を結合します。この結果、live データを静的 Web ページに簡単に追加することができます。live データは、ローカルのデータベースやリモートのデータベースから、さらにフラット・ファイルから抽出することができます。アプリケーションおよびシステム・サービスから生成することもできます。

2ページの図1 は、Net.Data for OS/390、Web サーバー、およびサポートされるデータとプログラミング言語環境の関係を示しています。

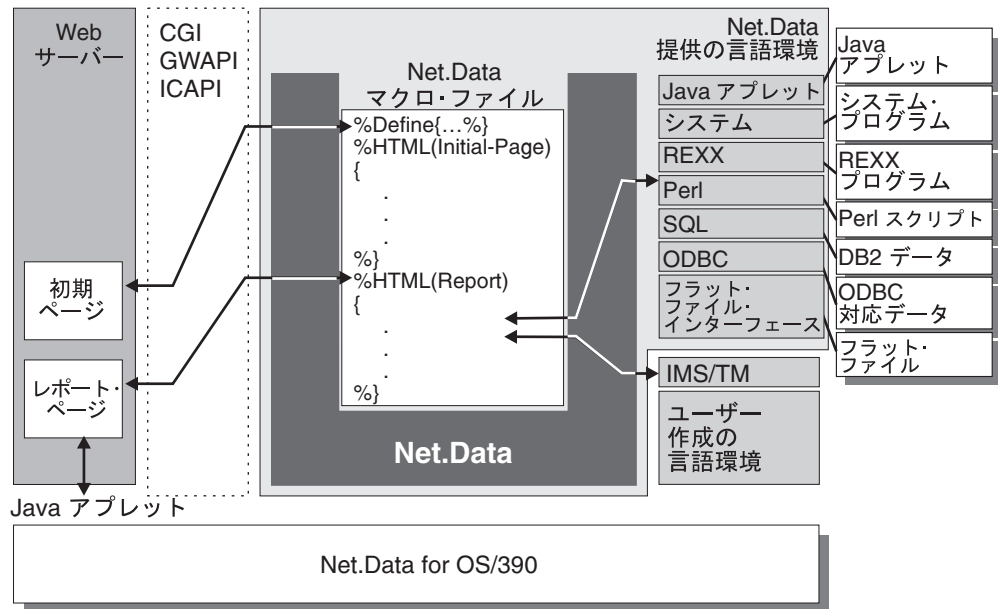


図 1. Net.Data for OS/390、Web サーバー、およびサポートされるデータとプログラム・ソース間の関係

Web サーバーは、Net.Data サービスを要求する URL を受信すると、CGI または Web サーバー・アプリケーション・プログラミング・インターフェース (API) を使用して Net.Data を起動します。URL には Net.Data 固有の情報が組み込まれています。これらの情報は、マクロとして処理されるか、SQL ステートメントまたはプログラムとして直接的に起動されます。Net.Data は要求の処理を終了すると、結果の Web ページを Web サーバーに送信します。サーバーはそのページを Web クライアントに渡し、そのクライアントでブラウザを使って表示します。

Net.Data を使用する利点

Net.Data を利用すれば、動的な Web ページを非常に簡単に作成できます。マクロ言語を使用すると、ユーザーが独自に Web サーバー・アプリケーションをプログラムするよりも簡単です。Net.Data では、ユーザーが現在知っている HTML、SQL、Perl、REXX、および JavaScript などの言語を使用することができます。また、マクロの変更は、ブラウザで即時に表示することができます。

Net.Data は、Web のデータおよび関連ビジネス・ロジックを使用可能にすることにより、OS/390 オペレーティング・システムの既存の拡張データ管理機能を補います。具体的に Net.Data は以下を行います。

- 単純でありながら強力なマクロ言語を提供します。これにより、インターネットおよびイントラネット・アプリケーションの開発が迅速に行えます。Net.Data Web アプリケーション環境には、以下の機能があります。
- Web アプリケーション内におけるデータ生成論理と表示論理を分割します。Net.Data は、データの表示方法 (HTML または Javascript など) に制限を課しません。このような分割により、ユーザーは最新の表示方法を使用したデータ表示の変更が容易に行えます。

- 既存のスキルおよびビジネス・ロジックを使用した Web ページの作成が可能です。C、C++、REXX、Java またはその他の言語で作成されたプログラムによりインターフェースをとることができます。
- 単純なマクロ言語および既存のプログラミング・スキルを使用して、複雑なインターネット・アプリケーションの開発が素早く行えます。
- ローカル DB2 サブシステム、および DRDA 使用可能なりモート・データ・ソースにより管理されるデータに対するハイパフォーマンスなアクセスを提供します。
- Net.Data ファミリー製品がサポートするすべてのオペレーティング・システム間でのマクロの移行が容易に行えます。

インタープリター・マクロ言語

Net.Data マクロ言語はインタープリター言語です。Net.Data はマクロを処理するために起動されると、各言語ステートメントを直接的にファイルの先頭から順次に解釈を開始します。このアプローチを使用することにより、ユーザーがマクロに加えた変更は、そのマクロを実行する URL をユーザーが指定すると即時に反映されます。再度コンパイルをする必要はありません。

直接要求

単一の SQL ステートメント、DB2 ストアード・プロシージャ、REXX プログラム、C や C++ プログラム、または Perl スクリプトを実行するための単純な要求では、マクロを作成する必要はありません。このような要求は直接 URL で指定して、ブラウザから Web サーバーまでの流れを制御します。

フリー・フォーマット

Net.Data マクロ言語には、2、3 のプログラミング・フォーマットがあるだけです。この単純さは、プログラマーに自由度と柔軟性をもたらします。単一の命令が複数の行に分解されたり、複数の命令が単一の行にまとめられたりします。命令はどの列からも開始することができます。スペースやすべての行をスキップすることができます。コメントはどこでも使用できます。

型を持たない変数

Net.Data はすべてのデータを文字ストリングとして取り扱います。Net.Data は組み込み関数を使用して、有効な数字を表しているストリングを算術演算します。指数のフォーマットもサポートされます。マクロ言語の変数の詳細については、58ページの『Net.Data のマクロ変数』で解説されています。

組み込み関数

Net.Data は組み込み関数を使用して、テキストや数字に関するいろいろな処理、検索、および比較演算を行います。他の組み込み関数には、フォーマット機能や算術計算があります。

エラー処理

Net.Data がエラーを検出すると、説明の付いたメッセージがクライアントに戻されます。ブラウザを使用するユーザーにエラー・メッセージが戻される前に、メッセージをカスタマイズすることができます。詳しくは、*Net.Data* 解説書を参照してください。

第2章 Net.Data のインストールおよび構成

SMP/E を使用することにより、Net.Data for OS/390 をインストールすることができます。 *Program Directory for Net.Data for OS/390 Version 2 Release 2*では、SMP/E のインストール・プロセスが説明されています。また、製品のインストール・テープが付属しています。

Net.Data を、SMP/E を使用してインストールした後、Net.Data を構成し、Web サーバーのための構成を変更しなければなりません。構成のステップは以下のとおりです。

- Net.Data の初期設定 (INI) ファイルのインストールおよびカスタマイズ
- CGI、ICAPI、GWAPI、または Net.Data サブレットとの併用のための Net.Data の構成
- Web サーバーの構成および環境変数ファイルのカスタマイズ
- Net.Data の言語環境の設定
- アクセス権限の指定
- メッセージ・カタログの使用可能化

本章では、Net.Data の構成方法、Net.Data と一緒に使用するための Web サーバーの構成方法について説明します。

- 6ページの『Net.Data の初期設定ファイルのインストール』
- 6ページの『Net.Data の初期設定ファイルのカスタマイズ』
- 20ページの『言語環境のセットアップ』
- 23ページの『DB2 との接続の管理』
- 23ページの『作業負荷管理の考慮事項』
- 24ページの『CGI と一緒に使用する Net.Data の構成』
- 25ページの『ICAPI あるいは GWAPI と一緒に使用するための Net.Data の構成』
- 27ページの『Java サブレットと一緒に使用するための Net.Data の構成』
- 28ページの『メッセージ・カタログを使用可能にする』
- 29ページの『Net.Data がアクセスするファイルおよびデータ・セットへのアクセス権の授与』

Net.Data の初期設定ファイルについて

Net.Data は、その初期設定ファイルを使用して、さまざまな構成変数の設定を確立し、言語環境と検索パスを構成します。構成変数の設定によって、以下のような Net.Data オペレーションのさまざまな局面を制御することができます。

- DB2 内での文字データのコード化
- スtring関数およびワード関数で DBCS が使用可能かどうか
- DB2 および DRDA 使用可能データにアクセスするための、デフォルト・サブシステム ID および計画名の選択

言語環境のステートメントは、使用可能な Net.Data の言語環境を定義し、言語環境に入出力する特殊な入力および出力パラメーター値を識別します。言語環境により、Net.Data は、DB2 データベースやシステム・サービスなどの異なる複数のデータ・ソースにアクセスすることができます。パス・ステートメントは、Net.Data が使用する、マクロ、REXX プログラム、および Perl のスクリプトなどの HFS ファイルのディレクトリー・パスを指定します。

Net.Data のコメントを使用して、Net.Data 初期設定ファイルのエントリーを文書化することができます。 *Net.Data* 解説書 の言語要素の章でコメント・ブロックのセクションを参照してください。

Net.Data の初期設定ファイルのインストール

SMP/E のインストール処理を行うと、db2www.ini という名前の Net.Data の初期設定ファイルのサンプルが、ディレクトリー /usr/lpp/netdata/pub に作成されます。(SMP/E のインストール処理については、*Program Directory for Net.Data for OS/390 Version 2 Release 2* で説明されています。)

Net.Data の初期設定ファイルをインストールするには、以下を行います。

1. Net.Data のサンプルの初期設定ファイルを、Web サーバーの文書のルート・ディレクトリーにコピーする。(Web サーバーの文書のルート・ディレクトリーは、Web サーバーの構成ファイル、/etc/httpd.conf において、要求テンプレート『/*』として、Pass ディレクティブにより指定されています。Web サーバーのデフォルトの文書のルート・ディレクトリーは、/usr/lpp/internet/server_root/pubです。しかし、これは、Web サーバーがインストールされた時点で変更されているかもしれません。Web サーバーの文書のルート・ディレクトリーが、internet/server_root/pub と異なっている場合、以下の指示の適切なものを選択して置き換えてください。)

Net.Data を、ディレクトリー /usr/lpp/netdata にインストールした場合は、OMVS 下で、以下のシェル・コマンドを実行して、初期設定ファイルをコピーすることができます。

```
cp /usr/lpp/netdata/pub/db2www.ini /usr/lpp/internet/server_root/pub
```

2. Net.Data の初期設定ファイルの許可番号は、必ず 644 になるようにください。

Net.Data の初期設定ファイルのカスタマイズ

初期設定ファイルに含まれている情報は、以下のセクションで説明する、3 つのタイプの構成ステートメントを使用して指定されます。

- 8ページの『構成変数ステートメント』
- 15ページの『パス構成ステートメント』
- 18ページの『環境構成ステートメント』

7ページの図2 に示されているサンプルの初期設定ファイルは、これらのステートメントの例を含んでいます。

個々の構成ステートメントのテキストは、途中で改行を入れず 1 行にしなければなりません。(ENVIRONMENT ステートメントは、読みやすくするために複数の行で示されています。) マクロから呼び出す各 LE について INI ファイルに ENVIRONMENT ステートメントが含まれていることを確認してください。

<pre> 1 %(changes: removed RETURN_CODE parm and DTW_DEFAULT ENVIRONMENT statement %) 2 DB2SSID DBNC 3 DB2PLAN DTWGAV22 4 DTW_DIRECT_REQUEST NO 5 DTW_SHOWSQL NO 6 MACRO_PATH /usr/lpp/netdata/macros; 7 EXEC_PATH /usr/lpp/netdata/testcmd; 8 FFI_PATH /usr/lpp/netdata/file-data; 9 ENVIRONMENT (DTW_SQL) dtwsql (IN LOCATION, DB2SSID, DB2PLAN, TRANSACTION SCOPE) 10 ENVIRONMENT (DTW_ODBC) odbc11 (IN LOCATION, TRANSACTION_SCOPE) 11 ENVIRONMENT (DTW_PERL) perl11 () 12 ENVIRONMENT (DTW_REXX) rexx11 () 13 ENVIRONMENT (DTW_FILE) filed11 () 14 ENVIRONMENT (DTW_APPLET) appl11 () 15 ENVIRONMENT (DTW_SYSTEM) sysd11 () </pre>	<ul style="list-style-type: none"> • 行 1 はコメントを含んでいます。 • 行 2 ～ 5 は、構成変数を定義しています。 • 行 6 ～ 8 は、HFS ファイルのパスを定義しています。 • 行 9 ～ 15 は、使用可能な環境ステートメントを定義しています。
--	--

図 2. Net.Data の初期設定ファイル

以下のセクションでは、初期設定ファイルの構成ステートメントのカスタマイズ方法について説明します。

移行のための注: 前のバージョンの Net.Data から移行する場合は、構成ステートメントの各セクションに、新しいリリースの Net.Data に移動するときに必要な変更の完全な記述が含まれていることを確認してください。

- 8ページの『構成変数ステートメント』
- 15ページの『パス構成ステートメント』

ENVIRONMENT ステートメントは、以下の変更が必要です。

- ENVIRONMENT ステートメントのパラメーター・リストから RETURN_CODE 変数を削除する。
- DTW_DEFAULT ENVIRONMENT ステートメントを削除する。

Net.Data の初期設定ファイルには新しい構成変数のデフォルト値が含まれるため、以下の変更も検討してください。

- アプリケーションで変数 SHOWSQL を使用する必要がある場合は、DTW_SHOWSQL 構成変数を YES に変更する。構文および例については、14ページの『DTW_SHOWSQL: SHOWSQL 構成変数を使用可能または使用不可にする』を参照してください。
- アプリケーションで要求を直接呼び出す必要がある場合は、DTW_DIRECT_REQUEST 構成変数を YES に変更する。構文および例については、12ページの『DTW_DIRECT_REQUEST: 直接要求変数を使用可能にする』を参照してください。

構成変数ステートメント

Net.Data の構成変数ステートメントは、構成変数の値を設定します。構成変数は、さまざまな目的に使用されます。変数の中には、適切な動作、あるいは代替モードでの動作のために、言語環境が必要とするものがあります。また変数によっては、文字の符号化や、構成中の Web ページの内容を制御するものもあります。さらに、構成変数ステートメントを使用して、アプリケーションに固有の変数を定義することもできます。

使用する構成変数は、使用している言語環境、および DB2 サブシステムに依存します。また、その他、アプリケーションに固有の要因にも依存します。

構成変数ステートメントを更新するには、以下のようになります。

アプリケーションで必要な構成変数を使って初期設定ファイルをカスタマイズする。構成変数は、以下の構文を持ちます。

```
NAME[=]value-string
```

等号は、オプションです。大括弧で示されます。

以下のサブセクションでは、初期設定ファイルで指定できる構成変数ステートメントについて説明します。

- 9ページの『DB2MSGs: DB2 のメッセージ・テキスト変数』
- 9ページの『DB2PLAN: DB2 のプラン変数』
- 10ページの『DB2SSID: DB2 のサブシステム ID 変数』
- 10ページの『DefaultDBCp: デフォルトのデータベース・コード・ページ変数』
- 11ページの『DSNAOINI: DB2 CLI の初期設定ファイル変数』
- 11ページの『DTW_CACHE_MACRO』
- 12ページの『DTW_DIRECT_REQUEST: 直接要求変数を使用可能にする』
- 12ページの『DTW_DO_NOT_CACHE_MACRO』
- 13ページの『DTW_MBMODE: ネイティブ言語サポート変数』
- 13ページの『DTW_REMOVE_WS: 余分な空白文字を削除するための変数』
- 14ページの『DTW_SHOWSQL: SHOWSQL 構成変数を使用可能または使用不可にする』
- 14ページの『DTW_SMTP_SERVER: 電子メールの SMTP サーバー変数』

構成変数の前提事項: Net.Data のサンプルの初期設定ファイルは、Net.Data の構成変数の設定のカスタマイズについて、いくつかの前提事項を設けます。これらの前提事項は、使用している環境では正しくないこともあります。

- DB2 サブシステムの ID 指定は、DBNC を使用する。ユーザーのアプリケーションの DB2SSID 構成変数を使用して、この値を置き換えます。
- DB2 プラン指定は、DTWGAV22 を使用する。ユーザーのアプリケーションの DB2PLAN 構成変数を使用して、この値を置き換えます。

- アプリケーションで変数 `SHOWSQL` を使用する必要がある場合は、`DTW_SHOWSQL` 構成変数を `YES` に変更する。構文および例については、14ページの『`DTW_SHOWSQL`: `SHOWSQL` 構成変数を使用可能または使用不可にする』を参照してください。
- アプリケーションで要求を直接呼び出す必要がある場合は、`DTW_DIRECT_REQUEST` 構成変数を `YES` に変更する。構文および例については、12ページの『`DTW_DIRECT_REQUEST`: 直接要求変数を使用可能にする』を参照してください。

DB2MSGs: DB2 のメッセージ・テキスト変数

`Net.Data` が `SQL` の言語環境を使用して `DB2` (`OS/390` 版) にアクセスする場合に、`DB2` 提供のメッセージ・テキストをロードするかどうかを指定します。

この変数は `MESSAGE` ブロックには影響しません。

構文：

`DB2MSGs [=] message_level`

ここで、`message_level` は、`Net.Data` が表示する `DB2` 提供のメッセージのレベルを示しています。 `message_level` は、以下の値に設定することができます。

NONE	<code>Net.Data</code> は、メッセージ・テキストを表示しないことを指定します。
ERRORONLY	<code>SQLCODE</code> の値が負の場合にのみ、 <code>Net.Data</code> はメッセージ・テキストを表示するよう指定します。
ALL	<code>SQLCODE</code> のすべての値の場合に、 <code>Net.Data</code> がメッセージ・テキストを表示するように指定します。これがデフォルトです。上でリストした有効な値のいずれとも異なる値が <code>DB2MSGs</code> に与えられる場合は、 <code>Net.Data</code> は、デフォルト値 <code>ALL</code> を使用します。

例： `DB2` のメッセージ・テキストのレベルを設定します。

`DB2MSGs NONE`

パフォーマンスのためのヒント： `DB2` のメッセージ・テキストを、ブラウザーで表示する必要がない場合は、`NONE` を指定すると、パフォーマンスを改善することができます。 `DB2` の警告メッセージ・テキストを、ブラウザーで表示する必要がない場合は、`ERRORONLY` を指定すると、パフォーマンスを向上することができます。

DB2PLAN: DB2 のプラン変数

`DB2` (`OS/390` 版) にアクセスする場合に、`SQL` 言語環境により使用される `DB2` のデフォルトのプランを指定します。

構文：

`DB2PLAN [=] plan_name`

例：デフォルトの `DB2` プラン名を設定します。

`DB2PLAN DTWGA22`

初期設定ファイルの設定をマクロで上書きするには、以下のようにします。

1. DB2PLAN 変数を、初期設定ファイルの DTW_SQL ENVIRONMENT ステートメントのパラメーターとして、以下の例に示すように追加する。

```
ENVIRONMENT (DTW_SQL) dtwsq1 (IN DB2PLAN)
```

2. マクロで、変数 DB2PLAN を、アプリケーションに必要な値に設定する。

DB2SSID: DB2 のサブシステム ID 変数

DB2 (OS/390 版) にアクセスする場合に、SQL 言語環境により使用されるデフォルトの DB2 のサブシステム ID を指定します。

構文：

```
DB2SSID [=] subsystem_id
```

例：デフォルトの DB2 サブシステム ID を設定します。

```
DB2SSID DBNC
```

初期設定ファイルの設定をマクロで上書きするには、以下のようにします。

1. DB2SSID 変数を、初期設定ファイルの DTW_SQL ENVIRONMENT ステートメントのパラメーターとして、以下の例に示すように追加する。

```
ENVIRONMENT (DTW_SQL) dtwsq1 (IN DB2SSID)
```

2. マクロで、変数 DB2SSID を、アプリケーションに必要な値に設定する。

DefaultDBCp: デフォルトのデータベース・コード・ページ変数

データベースのデータにアクセスするときに、Net.Data が使用するデフォルトのコード・ページ Net.Data は、この変数の設定を、以下の目的のために使用します。

- SQL ステートメント・テキストおよびストアド・プロシージャの呼び出しのための入力変数の値を、デフォルトのファイル・システムのコード・ページから、デフォルトのデータベースのコード・ページに変換する。
- ストアド・プロシージャの呼び出しおよび結果表からの出力変数の値を、デフォルトのデータベースのコード・ページから、デフォルトのファイル・システムのコード・ページに変換する。

Web サーバーの構成ファイル (/etc/httpd.conf) は、DefaultFsCp および DefaultNetCp ディレクティブを介して、デフォルトのコード・ページ環境を指定します。DefaultFsCp ディレクティブは、サーバー上のデフォルトのファイル・システムを指定します。このコード・ページは、EBCDIC のコード・ページです。Web サーバーは、この EBCDIC コード・ページで、Net.Data からテキスト・ストリームを受け取るものと予想しています。デフォルトの DefaultNetCp ディレクティブは、デフォルトのネットワークのコード・ページを指定します。このコード・ページは、ASCII のコード・ページです。この ASCII のコード・ページは、Web サーバーにより提供されるテキスト・ストリームを符号化するのに使用されます。

パフォーマンスのためのヒント：コード・ページ変数 DefaultDBCp は、アプリケーションに必要なければ、構成しないでください。この変数を定義すると、Net.Data は、特殊な変換が必要であることを前提にしています。

DefaultDBCp が、初期設定ファイル内で指定されていない場合、Net.Dataは、データベースのデータのコード・ページがデフォルトのファイル・システムのコード・ページと等価であるとみなし、変換が行われません。

構文：

DefaultDBCp [=] *code_page*

DSNAOINI: DB2 CLI の初期設定ファイル変数

DB2 CLI の初期設定ファイルの名前を指定します。この構成変数の名前は、順次データ・セットあるいは区分データ・セットのメンバーのいずれにもなることができます。

Net.Data の ODBC 言語環境を使用したい場合は、この変数を使用して、DB2 CLI の初期設定ファイルの名前を指定します。ODBC の言語環境を ICAPI と一緒に使用するつमोरの場合は、DB2 CLI の初期設定ファイルの MVSATTACHTYPE 変数を RRSAP に設定します。また、PLANNAME 変数を、DB2PLAN で指定された名前と同じプラン名に設定します。

構文：

DSNAOINI [=] *CLI_initialization_file_name*

例 1: 順次データ・セット CLI の初期設定ファイル名

DSNAOINI DBNC.DSNAOINI

例 2:

DSNAOINI DBNC.CLI(DSNAOINI)

DTW_CACHE_MACRO

Net.Data がキャッシュに入れるマクロを指定します。この変数は、DTW_DO_NOT_CACHE_MACRO 構成変数と一緒に機能します。説明、構文、および例については、12ページの『DTW_DO_NOT_CACHE_MACRO』を参照してください。

ガイドライン:

- Net.Data の初期設定ファイルにこの変数または DTW_DO_NOT_CACHE_MACRO 変数がいずれも含まれていない場合、マクロはキャッシュに入れられます。
- DTW_CACHE_MACRO と DTW_DO_NOT_CACHE_MACRO の両方が同じマクロを指定している場合、Net.Data はマクロをキャッシュに入れません。
- 指定されたマクロ・ファイルを使用する DTW_DO_NOT_CACHE_MACRO だけが Net.Data の初期設定ファイルに含まれ、サーバー上のその他すべてのマクロ・ファイルはキャッシュに入れられます。

構文：

DTW_CACHE_MACRO [=] *file_name1|path_template1;...;file_name(n)|path_template(n)*

ここで、

file_name

マクロ・ファイルの完全修飾名です。例: /u/user1/macros/custqord.d2w

path_template

マクロ・ファイルを含む 1 つ以上のディレクトリーのパス・テンプレート。パス・テンプレートは、サフィックス /* を含んでいなければなりません。Net.Data は、パス・テンプレートに一致する 1 つまたは複数のディレクトリーに含まれているすべてのマクロ・ファイルをキャッシュに入れます。

例: /u/user1/macros/*

例 1: ディレクトリー内のすべてのマクロ要求をキャッシュに入れる

/u/user1/macros/myapps

DTW_CACHE_MACRO /u/user1/macros/myapps/*

例 2: すべてのマクロ要求をキャッシュに入れる

DTW_CACHE_MACRO /*

DTW_DIRECT_REQUEST: 直接要求変数を使用可能にする

Net.Data の直接要求起動を使用可能にします。デフォルトでは、直接要求は使用不可になっています。

直接要求方式で Net.Data を起動することにより、ユーザーは、SQL ステートメントや Perl、REXX、または C プログラムの実行を URL 内に直接指定することができます。直接要求を使用不可にした場合、ユーザーは、マクロ要求方式を使用して Net.Data を起動しなければならず、マクロの中に定義されているか、マクロの中で呼び出される SQL ステートメントと関数だけを実行することができます。DTW_DIRECT_REQUEST を使用する場合の機密保護に関連した推奨事項については、33ページの『Net.Data のメカニズムを使用する』を参照してください。

構文 :

DTW_DIRECT_REQUEST YES|NO

ここで、

YES Net.Data 直接要求を使用可能にします。

NO Net.Data 直接要求を使用不可にします。NO がデフォルトです。

DTW_DO_NOT_CACHE_MACRO

Net.Data がキャッシュに入れないマクロを指定します。その他すべてのマクロはキャッシュに入れます。この変数は、DTW_CACHE_MACRO 構成変数と一緒に機能します。説明、構文、および例については、11ページの『DTW_CACHE_MACRO』を参照してください。

ガイドライン:

- 指定されたマクロ・ファイルを使用する DTW_DO_NOT_CACHE_MACRO 変数だけが Net.Data の初期設定ファイルに含まれ、サーバー上のその他すべてのマクロ・ファイルはキャッシュに入れます。

- Net.Data 初期設定ファイルにこの変数または DTW_CACHE_MACRO 変数がいずれも含まれていない場合、マクロはキャッシュに入れられません。
- DTW_CACHE_MACRO と DTW_DO_NOT_CACHE_MACRO の両方が同じマクロを指定している場合、Net.Data はマクロをキャッシュに入れません。

構文：

```
DTW_DO_NOT_CACHE_MACRO [=] file_name1|path_template1;...;file_name(n)|path_template(n)
```

ここで、

file_name

マクロ・ファイルの完全修飾名です。例: /u/user1/macros/custqord.d2w

path_template

マクロ・ファイルを含む 1 つ以上のディレクトリーのパス・テンプレート。パス・テンプレートは、サフィックス /* を含んでいなければなりません。Net.Data は、パス・テンプレートに一致する 1 つまたは複数のディレクトリーに含まれているすべてのマクロ・ファイルをキャッシュに入れます。

例: /u/user1/macros/*

例 1: ディレクトリー /u/user1/macros/myapps 内のマクロ要求を除くすべてのマクロ要求をキャッシュに入れる

```
DTW_DO_NOT_CACHE_MACRO /u/user1/macros/myapps/*
```

例 2: マクロ custqord.d2w を除くすべてのマクロ要求をキャッシュに入れる

```
DTW_DO_NOT_CACHE_MACRO /u/user1/macros/myapps/custqord.d2w
```

DTW_MBMODE: ネイティブ言語サポート変数

ワードおよびストリング関数の各国語サポートをアクティブにします。この変数の値が YES の場合、すべてのストリングおよびワード関数は、ストリングを混合データとして (すなわち、1 バイト文字集合および 2 バイト文字集合の両者の文字を含む可能性のあるストリングとして) 扱うことにより、ストリング内の DBCS 文字を正しく処理します。デフォルト値は、NO です。Net.Data のマクロで DTW_MBMODE 変数を設定することにより、初期設定ファイルで設定されている値を上書きすることができます。

構文：

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

例： 各国語サポートをアクティブにします。

```
DTW_MBMODE YES
```

この変数は、マクロで DEFINE ステートメントを使用して上書きすることができます。

DTW_REMOVE_WS: 余分な空白文字を削除するための変数

タブ、ブランク、および改行文字で構成される不必要なスペースを削除して、動的に生成された Web ページのサイズを削減します。この変数が YES に設定されていると、Net.Data は、2 つ以上の連続した空白文字を 1 つの改行文字に圧縮し、より短

くした HTML のページを生成します。この変数は、空白文字を圧縮することにより、Web ブラウザーに送信されるデータの量を削減し、それによってパフォーマンスを改善します。デフォルトは、NO です。

この変数は、マクロで DEFINE ステートメントを使用して上書きすることができます。

ヒント: この変数を YES と定義すると、表示される空白文字の量とタイプに影響します。ほとんどの場合ブラウザーは余分な空白文字を無視しますが、<PRE></PRE> タグを使用するマクロまたは DTW_PRINT_HEADER を NO に設定したマクロでは、ページ表示が異なることがあるので、この変数を NO に設定する (または未定義のままにする) ことをお勧めします。

構文 :

DTW_REMOVE_WS [=] YES|NO

例: 空白文字の圧縮

DTW_REMOVE_WS YES

DTW_SHOWSQL: SHOWSQL 構成変数を使用可能または使用不可にする

Net.Data マクロ内の SHOWSQL の設定の効果を上書きします。

構文 :

DTW_SHOWSQL YES|NO

ここで、

YES SHOWSQL の値を YES に設定したマクロにおいて SHOWSQL を使用可能にします。

NO 変数 SHOWSQL が YES に設定されていたとしても、マクロ内の SHOWSQL を使用不可にします。NO がデフォルトです。

表1 では、特定のマクロについて SHOWSQL 変数を使用可能にするか使用不可にするかが、Net.Data の初期設定ファイルとマクロの設定によってどのように決定されるかを示しています。

表1. Net.Data の初期設定ファイルとマクロの SHOWSQL に関する設定の関係

DTW_SHOWSQL の設定	SHOWSQL の設定	SQL ステートメントの表示
NO	NO	NO
NO	YES	NO
YES	NO	NO
YES	YES	YES

DTW_SMTP_SERVER: 電子メールの SMTP サーバー変数

電子メール・メッセージの送信に使用する SMTP サーバーを指定します。この変数の値は、SMTP サーバーの名前 (SMTP サーバーがローカル・システム上にある場合)、またはノードと SMTP サーバーの名前 (SMTP サーバーがリモート・システム上にあ

る場合) にすることができます。この変数が設定されていない場合、Net.Data は値 SMTP をサーバーの名前として使用し、サーバーがローカル・システム上にあることを前提にします。

構文：

```
DTW_SMTP_SERVER server_name
```

ここで、*server_name* は、以下の値の 1 つです。

name ローカル・システム上で実行されている SMTP サーバーの名前を指定します。デフォルト名は SMTP です。

node.name

SMTP サーバーが実行されているノードと SMTP サーバーの名前を指定します。

例:

```
DTW_SMTP_SERVER mynode.myserver
```

パス構成ステートメント

Net.Data は、Net.Data のマクロが使用するファイルと実行可能プログラムの位置を、パス構成ステートメントの設定から決定します。パス・ステートメントは以下の通りです。

- 16ページの『MACRO_PATH』
- 17ページの『EXEC_PATH』
- 17ページの『INCLUDE_PATH』
- 18ページの『FFI_PATH』

これらのパス・ステートメントは、マクロ、実行可能ファイル、HFS ファイル、および組み込みファイルを見つけようとするときに、Net.Data が検索する 1 つまたは複数のディレクトリーを識別します。必要とするパス・ステートメントは、マクロが使用する Net.Data の能力に依存します。

Net.Data のサンプルの初期設定ファイルは、Net.Data の検索パスの設定のカスタマイズについて、幾つかの前提事項を設けます。これらの前提事項は、使用環境によっては正しくないこともあり、次のようにパス構成ステートメントの変更を要求することもあります。

- Net.Data のマクロ・ディレクトリーのパスが、`/usr/lpp/netdata/macros` と異なる場合は、そのパスを、MACRO_PATH ステートメントのマクロ・ディレクトリーのパスに置き換える。

Net.Data の `/usr/lpp/netdata/macros` ディレクトリーに含まれるファイルは、SMP/E の制御下にあり、変更することはできません。これらのファイルのどれか 1 つでも変更した場合は、作成するディレクトリーに格納されるファイルのコピーに変更を加えてください。SMP/E が作成したディレクトリーの検索前に、Net.Data に、ユーザーの専用ディレクトリー内のこれらのファイルを検索するように指示しなければなりません。これを行うには、`db2www.ini` ファイルにおいて、SMP/E が作成したディレクトリーの前に、ユーザー専用のディレクトリーを追加します。た

例えば、SMP/E のインストール中に提供されたマクロをカスタマイズし、そのマクロを、ディレクトリー /u/user1/macros に配置し、デフォルトの MACRO_PATH ステートメントを、以下のステートメントに置き換えます。

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

- Net.Data の外部プログラムのディレクトリーのパスが、/usr/lpp/netdata/testcmd と異なる場合は、そのパスを、EXEC_PATH ステートメントのユーザーの外部プログラムのディレクトリーのパスに置き換える。
- Net.Data のフラット・ファイルのディレクトリーのパスが、/usr/lpp/netdata/file-data と異なる場合は、そのパスを、FFI_PATH ステートメントのユーザーのフラット・ファイルのディレクトリーのパスに置き換えます。

更新のガイドライン：

幾つかの一般的なガイドラインは、すべてのパス・ステートメントに適用されます。

- 指定された各ディレクトリーは、セミコロン (;) で終了する。
- 各パス・ステートメントは、複数のパスを指定することができる。パスの検索は、指定された順で左から右に行われます。この複数パス能力により、ユーザーのファイルを複数のディレクトリーに編成することができます。たとえば、複数の Web アプリケーションをそれぞれ、それ自身のディレクトリーに配置することができます。
- 絶対パス・ステートメントを使用することをお勧めします。

以下のセクションでは、各パス・ステートメントの目的と構文を説明し、有効なパス・ステートメントの例を提供します。

MACRO_PATH

MACRO_PATH 構成ステートメントは、Net.Data が Net.Data マクロを検索するディレクトリーを識別します。たとえば、以下の URL 指定は、パスおよびファイル名 /macro/sqlm.d2w を持つ Net.Data のマクロを要求します。

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

構文：

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

等号 (=) は、オプションで、大括弧で示されます。

Net.Data は、パス /macro/sqlm.d2w を、MACRO_PATH 構成ステートメントのパスに、Net.Data がマクロを検出するまで、またはすべてのパスを検索するまで、左から右へ、追加していきます。Net.Data のマクロの起動に関する情報については、39ページの『第4章 Net.Data を起動する』を参照してください。

例：以下の例は、初期設定ファイルの MACRO_PATH ステートメントと、Net.Data を起動する関連リンクを示しています。

Net.Data の初期設定ファイル：

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

HTML リンク

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

ファイル *query.d2w* がディレクトリー */u/user1/macros* で検出された場合、完全修飾のパスは */u/user1/macros/query.d2w* です。

EXEC_PATH

EXEC_PATH 構成ステートメントは、1 つ以上のディレクトリーを識別します。これは、EXEC ステートメントまたは実行可能変数により起動される外部プログラムを Net.Data が検索するディレクトリーです。プログラムが検出されれば、外部プログラム名がパス指定に追加され、実行のために言語環境に渡される完全修飾ファイル名になります。

構文：

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

例: 以下の例は、初期設定ファイルの EXEC_PATH ステートメントと、外部プログラムを起動するマクロの EXEC ステートメントを示しています。

Net.Data の初期設定ファイル：

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

Net.Data のマクロ

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

ファイル *myFunction.cmd* が */usr/lpp/netdata/prgms* ディレクトリーで検出された場合、プログラムの修飾名は */usr/lpp/netdata/prgms/myFunction.cmd* です。

INCLUDE_PATH

INCLUDE_PATH 構成ステートメントは、Net.Data が検索する 1 つ以上のディレクトリーを識別し、Net.Data のマクロの INCLUDE ステートメントで指定されたファイルを検出します。ファイルを検出すると、Net.Data は、組み込みファイル名を、パス指定に追加し、修飾された組み込みファイル名を作成します。

構文：

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

ヒント: ローカルな Web サーバーから HTML ファイルを組み込む場合、*Net.Data* 解説書の INCLUDE_URL のローカル Web サーバーの例に示されているように、INCLUDE_URL 構成要素を使用します。例示された構文を使用することにより、INCLUDE_PATH を更新して、Web サーバーにとっては既知のディレクトリーを指定する必要がなくなります。

例 1: 以下の例は、初期設定ファイルの INCLUDE_PATH ステートメントと、組み込みファイルを指定する INCLUDE ステートメントの両方を示しています。

Net.Data の初期設定ファイル :

```
INCLUDE_PATH /u/user1/includes;/usr/lpp/netdata/includes;
```

Net.Data のマクロ

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

ファイル *myInclude.txt* が /u/user1/includes ディレクトリーで検出された場合、組み込みファイルの完全修飾名は /u/user1/includes/myInclude.txt です。

例 2: 以下の例は、INCLUDE_PATH ステートメントと、サブディレクトリー名を指定した INCLUDE ファイルを示しています。

Net.Data の初期設定ファイル :

```
INCLUDE_PATH /u/user1/includes;/usr/lpp/netdata/includes;
```

Net.Data のマクロ

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

組み込みファイルは、ディレクトリー /u/user1/includes/OE と /usr/lpp/netdata/includes/OE で検索されます。ファイルが、/usr/lpp/netdata/includes/OE で検出される場合、組み込みファイルの完全修飾名は、/usr/lpp/netdata/includes/OE/oeheader.inc になります。

FFI_PATH

FFI_PATH 構成ステートメントは、Net.Data がフラット・ファイル・インターフェース (FFI) 関数で参照される HFS ファイルを検索する 1 つ以上のディレクトリーを識別します。

構文 :

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

例: 以下の例は、初期設定ファイルの FFI_PATH ステートメントを示しています。

Net.Data の初期設定ファイル :

```
FFI_PATH /u/user1/ffi;/usr/lpp/netdata/ffi;
```

FFI 言語環境が呼び出されると、Net.Data は、FFI_PATH ステートメントで指定されたパス内を調べます。

FFI_PATH ステートメントは、パス・ステートメントのディレクトリーに含まれていないファイルに対して機密保護を提供するために使用されるため、検出されなかった FFI ファイルに対する特別な機能があります。 *Net.Data* 解説書の FFI 組み込み関数のセクションを参照してください。

環境構成ステートメント

ENVIRONMENT ステートメントは、言語環境を構成します。言語環境とは、Net.Data の構成要素です。Net.Data は、この構成要素を使用して、DB2 データベースのようなデータ・ソースにアクセスしたり、あるいは、REXX のような言語で書かれたプログ

ラムを実行します。Net.Data は、言語環境のセットと、ユーザー自身の言語環境の作成を可能にするインターフェースを提供します。これらの言語環境については 89ページの『第6章 言語環境の使用』で、言語環境インターフェースについては *Net.Data* 言語環境解説書 で説明しています。

Net.Data では、特定の言語環境を起動する前に、その言語環境のための ENVIRONMENT ステートメントが存在していることが必要です。

Net.Data は、Net.Data 言語環境が行う、FUNCTION ブロックで定義された関数の呼び出しの解釈方法に影響を与える幾つかの変数を指定します。これらの変数の設定が有効になるためには、言語環境に渡されなければなりません。

たとえば、マクロは、DTW_SQL 関数内の SQL ステートメントを実行するリモートの DBMS の場所名を指定する LOCATION 変数を定義することができます。LOCATION の値は、SQL の言語環境 (DTW_SQL) に渡さなければなりません。これにより、SQL の言語環境は、指定されたリモートの DBMS に接続することができます。変数を言語環境に渡すには、LOCATION 編集を、DTW_SQL の環境ステートメントのパラメーター・リストに追加しなければなりません。

また、構成変数として初期設定ファイル内に設定し、マクロで上書きすることができる変数もあります。たとえば、SQL 言語環境が呼び出されたときに、DB2PLAN および DB2SSID 変数のデフォルトの設定をマクロに上書きさせたい場合は、それらの変数を DTW_SQL の ENVIRONMENT ステートメントに組み込みます。

ENVIRONMENT ステートメントの変更: 前のバージョンの Net.Data から移行する場合は、ENVIRONMENT ステートメント・セクションで以下の変更を行ってください。

- ENVIRONMENT ステートメントのパラメーター・リストから RETURN_CODE 変数を削除する。
- DTW_DEFAULT ENVIRONMENT ステートメントを削除する。
- DB2 (OS/390 版) V6 をアプリケーションに使用する場合は、DTW_SQL の ENVIRONMENT ステートメントを、ENVIRONMENT (DTW_SQL) dtwsq1 () から ENVIRONMENT (DTW_SQL) dtwsq1v6 () に変更する。

Net.Data のサンプルの初期設定ファイルは、Net.Data の環境構成ステートメントの設定のカスタマイズについて、幾つかの前提事項を設けます。これらの前提事項は、使用する環境では正しくないこともあります。ステートメントを使用環境に合わせて適切に変更します。

ENVIRONMENT の追加と更新を行うには、以下のようにします。

ENVIRONMENT ステートメントは以下の構文を持っています。

ENVIRONMENT(*type*) *library_name* (*parameter_list*, ...)

パラメーター :

- *type*

Net.Data が、Net.Data のマクロで定義された FUNCTION ブロックと、この言語環境とを関連付けるための名前です。FUNCTION ブロックの定義で、言語環境のタイプを指定し、Net.Data が関数を実行するために使用しなければならない言語環境を識別しなければなりません。

- *library_name*

Net.Data が呼び出す言語環境インターフェースを含む DLL の名前。

DLL 名は、拡張子 *.dll* を付けずに指定します。

- *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。

パラメーター・リストで変数を設定して渡すには、マクロ内で変数を定義します。

言語環境で処理される関数を実行する前に、これらのパラメーターを、ユーザーのマクロで、構成変数、あるいは変数として定義しなければなりません。関数が、その出力パラメーターのどれかを変更すると、パラメーターは、関数が完了後も、それらパラメーターの値を保持します。以下のリストは、ENVIRONMENT ステートメントが渡すことのできる変数を指定しています。

DTW_SQL: TRANSACTION_SCOPE, LOCATION, DB2SSID, DB2PLAN

DTW_ODBC: TRANSACTION_SCOPE, LOCATION

Net.Data が初期設定ファイル进行处理する場合、Net.Data は、言語環境の DLL をロードしません。Net.Data が言語環境の DLL をロードするのは、その言語環境を識別する関数を最初に実行するときです。DLL は、Net.Data がロードされている限り、ロードされた状態を続けます。

例: Net.Data 提供の言語環境の ENVIRONMENT ステートメント

アプリケーションの ENVIRONMENT ステートメントをカスタマイズする場合、ユーザーの初期設定ファイルから言語環境に渡す必要のある変数、あるいは Net.Data のマクロ記述者が、自分のマクロを設定したり上書きしたりする必要のある変数を、ENVIRONMENT ステートメントに追加します。

```
ENVIRONMENT (DTW_SQL)      dtwsq1      (IN LOCATION, DB2SSID, DB2PLAN,
TRANSACTION_SCOPE)
ENVIRONMENT (DTW_ODBC)     odbcd11     (IN LOCATION, TRANSACTION_SCOPE)
ENVIRONMENT (DTW_APPLET)   appld11     ()
ENVIRONMENT (DTW_PERL)     perld11     ()
ENVIRONMENT (DTW_FILE)     filed11     ()
ENVIRONMENT (DTW_REXX)     rexxd11     ()
ENVIRONMENT (DTW_SYSTEM)   sysd11      ()
```

必要事項: 各 ENVIRONMENT ステートメントは、途中で改行を入れず単一の行にしなければなりません。

言語環境のセットアップ

Net.Data の言語環境の構成変数および ENVIRONMENT 構成ステートメントを変更した後、以下の言語環境が適切に機能するようになるには、さらにいくつかのセットアップが必要です。以下のセクションでは、言語環境をセットアップするために必要なステップについて説明します。

- 『IMS Web の言語環境』
- 『SQL および ODBC の言語環境』

IMS Web の言語環境

IMS Web の言語環境を使用するには、以下のステップを完了しなければなりません。

1. Net.Data を実行する Web サーバーに IMS Web 実行時コンポーネントをインストールする。IMS Web 実行時コンポーネントのインストールについては、*IMS Web User's Guide* を参照してください。(英語版)

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

2. ホスト・システムに IMS TCP/IP OTMA 接続 (IMS TOC) をインストールする。IMS TCP/IP OTMA 接続のインストールおよび使用については、以下を参照してください。(英語版)

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

3. トランザクション DLL を作成する。
 - a. IMS Web Studio ツールを使用したトランザクションのために、C++ コード、make ファイル (DTWproj.mak)、および Net.Data マクロ (DTWproj.d2w) ファイルを、HFS ソースから生成する。
 - b. 生成した make ファイルを使用して、トランザクション DLL の実行可能形式を構築する。
4. トランザクション DLL ファイル (DTWproj.dll) および Net.Data のマクロ・ファイル (DTWproj.d2w) を Web サーバーにコピーする。
 - a. Net.Data がマクロを検索するディレクトリーに、マクロを置く。(詳しくは、16ページの『MACRO_PATH』を参照してください。)
 - b. Web サーバーが DLL を検索するディレクトリーに、トランザクション DLL または共用ライブラリーを置く。

5. IMS Web Studio ツールで生成したサンプル・ファイル DTWproj.htm にあるリンクを使用して、Web サーバーの HTML ツリーの HTML ファイルを変更します。その後、そのリンクを使用して Net.Data を起動し、入力 HTML フォームを表示して IMS Web 言語環境を起動します。次に、IMS Web の言語環境は、IMS トランザクション DLL を呼び出します。この DLL は、IMS Web 実行時 DLL によって検証済みのサービスを使用してトランザクションを実行し、出力を Web ブラウザーに戻します。

IMS Web 実行時 DLL は、要求メッセージをフォーマットし、IMS TOC から OTMA へ送信します。それによって、適切なトランザクションが待ち行列に入れます。トランザクションの出力は、OTMA によって IMS TOC を介して IMS Web に戻されます。その後、トランザクションは IMS Web の言語環境を通して Net.Data に戻され、Web ブラウザーに表示されます。

SQL および ODBC の言語環境

SQL の言語環境 (DTW_SQL) および ODBC の言語環境 (DTW_ODBC) は、DB2 のロード・モジュール・ライブラリー SDSNLOAD を使用します。Net.Data の SQL および ODBC の言語環境は、このライブラリーが、LINKLIST に常駐すること、あるいは、このライブラリーが、Web サーバーの始動プロシージャの STEPLIB DD ス

ステートメントで指定されていることを要求します。Web サーバーの開始プロシージャの名前と位置は、使用しているシステムの構成に依存します。

必要事項:

- Net.Data の SQL および ODBC の言語環境を使用する前に、Net.Data のプランを作成し、ストアード・プロシージャを呼び出すか、他のタイプの SQL ステートメントを実行します。このプランを作成するのに必要なバインドは、使用を計画している言語環境および使用している DB2 のバージョンに依存します。
- ICAPI、GWAPI、および Net.Data サブレットと一緒に Net.Data を使用する場合、SQL および ODBC の言語環境は、RRS 接続機能を必要とします。RRS 接続機能が DB2 用にインストールされており、OS/390 RRS がインストールされて適切に構成されていることを確認してください。

以下のアプローチの 1 つを使用して、Net.Data の DBRM をパッケージにバインドします。

- サンプルの JCL を使用して Net.Data をバインドします。サンプルは、DBRM をパッケージにバインドし、SQL 言語環境の使用をサポートする Net.Data プランを作成し、そのプランに対する EXECUTE 権限を PUBLIC に授与します。サンプル JCL は、以下のジョブの 1 つに入っています。

DTWBIND DB2 V5 で SQL 言語環境を使用する場合

DTWBIND6 DB2 V6 で SQL 言語環境を使用する場合

DTWOBIND DB2 V5 で、ODBC 言語環境、または ODBC 言語環境と SQL 言語環境の両方を使用する場合

DTWOBND6 DB2 V6 で、ODBC 言語環境、または ODBC 言語環境と SQL 言語環境の両方を使用する場合

- SQL および ODBC 言語環境の両方を使用する計画なら、DB2 CLI 対応の DBRM を Net.Data の DBRM と同じプランにバインドする。Net.Data の DBRM および DB2 CLI の DBRM のパッケージへのバインド、SQL および ODBC の言語環境の使用をサポートする Net.Data プランの作成、およびプランに対する EXECUTE 権限の PUBLIC への授与を行うためのサンプルの JCL は、DTW220.SDTWBASE(DTWOBIND) にあります。

使用している環境内で JCL の実行を成功させるためには、サンプルの JCL に幾つかの小さな変更を加える必要があるかもしれません。

- STEPLIB DD ステートメントで指定される SDSNEXIT および SDSNLOAD データ・セット名の接頭部は、使用している DB2 のバージョンに依存し、インストールには誤っている可能性があります。
- DSN コマンドの SYSTEM オプションおよび RUN コマンドの PLAN オプションで指定される値も、インストールには誤っている可能性があります。DSN コマンドの SYSTEM オプションは、DB2 サブシステムの名前を指定し、アプリケーションのサブシステム ID と同じでなければなりません。
- RUN コマンドの PLAN オプションは、DSNTIAD プログラムのためのアプリケーション・プランの名前を指定します。ストアード・プロシージャを使用する計画であるのならば、ストアード・プロシージャのためのパッケージを、Net.Data のプランにバインドする必要があるかもしれません。

適切な変更をすべて行い、JCL を処理依頼します。

DB2 との接続の管理

Net.Data のようなアプリケーション・プログラムは、DB2 により管理されるデータにアクセスしたり、DB2 のストアド・プロシージャを実行するには、DB2 (OS/390 版) に接続しなければなりません。、ICAPI、GWAPI または Net.Data サブレットを使用する場合、Net.Data は、DB2 製品の一部として提供される、リソース回復サービス接続機能 (RRSAF) を使用して、この目的を達成します。DB2 のサブシステムへの接続を確立すると、大きなオーバーヘッドが発生するため、既存の接続の再利用が、ユーザーの要求ごとに新規に接続を生成し直す方法に代わる魅力的な代替案になります。

Net.Data は、それが ICAPI、GWAPI、または Net.Data サブレットとの使用のために構成されるときに、SQL および ODBC の言語環境が使用する接続の再利用をサポートします。Web サーバーのスレッドが、DB2 へのアクセスを必要とする Net.Data のユーザー要求を処理する場合、Net.Data は DB2 に接続し、DB2 スレッドを作成します。DB2 スレッドは、Web サーバーが実行されている限り、残っています。Web サーバーがこの Web サーバー・スレッドに後続の要求を割り当て、DB2 へのアクセスが必要な場合、Net.Data は既存の DB2 スレッドを再利用します。Net.Data は、必要に応じて、DB2 のプラン名およびユーザー ID を変更し、新規サブシステム ID に切り替えて、要求の要件を満たします。作成される DB2 スレッドの数は、DB2 スレッドの数と Web サーバースレッドの数と同じになるまで、増加します。この時点で、サーバーの定常状態オペレーションに達します。Net.Data は、既存の DB2 スレッドを再利用し、新規の DB2 スレッドは作成されません。

接続管理機能の利用には、Net.Data の構成は必要ありません。しかし、作業負荷管理プログラム (WLM) を使用して、Net.Data の要求を処理する Web サーバーのアドレス空間を管理したいのであれば、何らかの WLM 構成を追加する必要があります。

作業負荷管理の考慮事項

作業負荷管理プログラム (WLM) は、OS/390 オペレーティング・システムの構成要素で、ビジネスの目標に向かってのシステム性能の定義、実装、およびモニターのための機能を提供してくれます。WLM は、処理作業のリソースを割り当てます。そのために、ユーザーのアプリケーションのパフォーマンスおよびスケーラビリティが、確実にユーザーの要求を満たすように、ユーザーが定義する方針を使用します。

Net.Data を ICAPI あるいは GWAPI とともに使用するために構成する場合、IBM Internet Connection Server あるいは Lotus Domino Go Webserver いずれかにより、WLM を使用して、ユーザーの Net.Data 作業負荷を管理するための方針を確立します。ユーザーは、与えられたテンプレートに一致する URL 要求の処理のためのアプリケーション環境 および WLM トランザクション・クラスにより、これらの方針を確立することができます。

WLM を使用することにより得られる利点は、Net.Data の初期設定ファイル (db2www.ini) に変更を加え、WLM の REFRESH あるいは WLM の QUIESCE コマンドを使用することにより、Web サーバーを停止したり、再始動しなくても、その変更を有効にすることができる、ということです。

WLM の詳細については、“OS/390 MVS 計画: ワークロード管理サービス V2、GC880-6582-04”を参照してください。

WLM を用いた IBM Internet Connection Server および Lotus Domino Go Webserver の使用の詳細については、以下を参照してください。

- *ICSS OS/390:Webmaster の手引き V2R、GC88-7534-00*
- *Go Webserver for OS/390 Webmaster の手引き リリース 4.6.1、SD88-7826-00*

CGI と一緒に使用する Net.Data の構成

コモン・ゲートウェイ・インターフェース (CGI) は、Net.Data のようなアプリケーション・プログラムを、Web サーバーから起動することができるようにする業界標準のインターフェースです。Net.Data の CGI サポートにより、Net.Data をお気に入りの Web サーバーと一緒に使用することができます。

Net.Data のための階層ファイル・システム (HFS) のディレクトリを作成したときに、ディレクトリの構造や名前を変更していない限り、SMP/E のインストール・プロセスは、Net.Data の実行可能ファイルおよび DLL を、ディレクトリ `/usr/lpp/netdata/cgi-bin` にインストールしています。`/usr/lpp/netdata` は、ユーザーの Web サーバーのルート・ディレクトリではないので、Web サーバーは、ユーザーが Web サーバーの構成に変更を加えていなければ、クライアントの Net.Data 要求を処理することはできません。

Web サーバーを変更するには、以下を行います。

1. Web サーバーを停止する。
2. 以下のアプローチのいずれかを使用して、実行可能ファイルと DLL のインストールを完了する。

- **Net.Data のディレクトリの使用**

- a. Net.Data の要求を `/usr/lpp/netdata/cgi-bin` ディレクトリにリダイレクトする Web サーバーの構成ファイル `/etc/httpd.conf` に、Exec ディレクティブを追加する。たとえば、以下のようになります。

```
Exec /netdata-cgi/* /usr/lpp/netdata/cgi-bin/*
```

- b. ユーザーの Net.Data の `cgi-bin` ディレクトリを、Web サーバーの環境変数ファイル `/etc/httpd.envvars` の `LIBPATH` ステートメントに追加します。ユーザーの Net.Data の `cgi-bin` ディレクトリが、`/usr/lpp/netdata/cgi-bin` の場合、`LIBPATH` ステートメントは、以下のステートメントと類似するはずです。

```
LIBPATH=/usr/lpp/internet/bin:/usr/lpp/netdata/cgi-bin
```

- **Web サーバーのディレクトリの使用**

- a. 実行可能ファイルと DLL (`appldll`、`db2www`、`dtwle`、`dtwlei`、`dtwsqll`、`dtwsqllv6`、`filedll`、`odbc.dll`、`perl.dll`、`rexcdll`、`sysdll`) を、Web サーバーの `cgi-bin` ディレクトリに移動する。Web サーバーのデフォルトの `cgi-bin` ディレクトリは、`/usr/lpp/internet/server_root/cgi-bin` です。Web サーバーのデフォルトのルート・ディレクトリは、Web サーバーの構成ファイル `/etc/httpd.conf` の `ServerRoot` ディレクティブによって指定されますが、Web サーバーのインストール時に、変更された可能性もあり

ます。Web サーバーのデフォルトの cgi-bin ディレクトリーは、Web サーバーの構成ファイルの Exec ディレクティブで指定されますが、Web サーバーのインストール時に、変更された可能性もあります。Web サーバーのルート・ディレクトリーが /usr/lpp/internet/server_root と異なっていたり、Web サーバーの cgi-bin ディレクトリーが、/usr/lpp/internet/server_root/cgi-bin と異なっている場合は、以下の指示では、適宜ユーザーの選択ディレクトリーに置き換えてください。

- b. Web サーバーの cgi-bin ディレクトリーを、Web サーバーの環境変数ファイル /etc/httpd.envvars の LIBPATH ステートメントに追加する。ユーザーの Web サーバーの cgi-bin ディレクトリーが、/usr/lpp/internet/server_root/cgi-bin の場合、ユーザーの LIBPATH ステートメントは、以下のステートメントと類似するはずです。

```
LIBPATH=/usr/lpp/internet/bin:/usr/lpp/internet/server_root/cgi-bin
```

3. Net.Data の実行可能ファイルと DLL、および実行可能なファイルと DLL へのパスの各ディレクトリーに対する許可番号が、確実に 755 になるようにする。
4. Web サーバーを再始動する。

制約事項: Web サーバーの環境変数ファイルの LIBPATH ステートメントに、以下のディレクトリーを 2 つ以上指定しないでください。

- cgi-bin
- icapi-lib
- servlet-lib

Web サーバーのインストールと Web サーバーの構成ファイルのディレクティブの詳細については、以下の資料を参照してください。

- *ICSS for OS/390 概説およびインストール V2R2*, GC88-7949-00
- *ICSS OS/390:Webmaster の手引き V2R2*, GC88-7534-00
- *Go Webserver 概説およびインストール V4R6*, SD88-7825-00
- *Go Webserver for OS/390 Webmaster の手引き リリース 4.6.1*, SD88-7826-00

ICAPI あるいは GWAPI と一緒に使用するための Net.Data の構成

CGI ではなく、Web サーバーのアプリケーション・プログラミング・インターフェース (API) を使用することにより、Net.Data のパフォーマンスを大きく改善することができます。これらの API を使用して、Net.Data は DB2 への接続を再利用します。Net.Data は、DB2 スレッドを作成し、プロセスが存続している限り、作成したスレッドを保持します。

CGI を使用して実行に成功するマクロならどれでも、ICAPI あるいは GWAPI の実行に成功します。これらのマクロに対して変更を加える必要はありません。

Net.Data のための HFS ディレクトリーの作成時に、ディレクトリー構造や名前を変更していなければ、SMP/E のインストール・プロセスは、Net.Data の実行可能ファイルおよび DLL を、ディレクトリー /usr/lpp/netdata/icapi-lib にインストールしています。/usr/lpp/netdata は、ユーザーの Web サーバーのルート・ディレクト

リーではないので、Web サーバーの構成に何らかの追加変更をしていない限り、Web サーバーは、Net.Data に対するクライアントの要求には応えることができません。

Web サーバーを変更するには、以下を行います。

1. Web サーバーを停止する。
2. 以下のアプローチのいずれかを使用して、実行可能ファイルと DLL のインストールを完了する。

- **Net.Data のディレクトリーの使用**

- a. Web サーバーの構成ファイル /etc/httpd.conf に、ServerInit ディレクティブを追加し、Web サーバーがその初期化ルーチンを実行するときに、Net.Data に固有の初期化を実行するよう、Web サーバーに指示する。想定される ServerInit ディレクティブを 1 つ挙げると、以下のようになります。

```
ServerInit /usr/lpp/netdata/icapi-lib/db2www:dtw_init
```

- b. Web サーバーの構成ファイル/etc/httpd.conf に Service ディレクティブを追加し、/usr/lpp/netdata/icapi-lib ディレクトリーに Net.Data の要求をリダイレクトする。想定される Service ディレクティブを 1 つ挙げると、以下のようになります。

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

- c. ユーザーの Net.Data の icapi-lib ディレクトリーを、Web サーバーの環境変数ファイル /etc/httpd.envvars の LIBPATH ステートメントに追加する。ユーザーの Net.Data の icapi-lib ディレクトリーが、/usr/lpp/netdata/icapi-lib の場合、LIBPATH ステートメントは、以下と類似するはずです。

```
LIBPATH=/usr/lpp/internet/bin:/usr/lpp/netdata/icapi-lib
```

- **Web サーバーのディレクトリーの使用**

- a. 実行可能ファイルと DLL (appld11、db2www、dtwle、dtwlei、dtwsql、dtwsqlv6、filed11、odbcd11、perl11、rex11、sysd11) を、Web サーバーの cgi-bin ディレクトリーに移動する。Web サーバーのデフォルトの cgi-bin ディレクトリーは、/usr/lpp/internet/server_root/cgi-bin です。

- b. Web サーバーの構成ファイル /etc/httpd.conf に、ServerInit ディレクティブを追加し、Web サーバーがその初期化ルーチンを実行するときに、Net.Data に固有の初期化を実行するよう、Web サーバーに指示する。想定される ServerInit ディレクティブを 1 つ挙げると、以下のようになります。

```
ServerInit /usr/lpp/internet/server_root/cgi-bin/db2www:dtw_init
```

- c. Web サーバーの構成ファイル /etc/httpd.conf に Service ディレクティブを追加し、Net.Data の要求を /usr/lpp/internet/server_root/cgi-bin ディレクトリーにリダイレクトする。想定される Service ディレクティブを 1 つ挙げると、以下のようになります。

```
Service /cgi-bin/db2www* /usr/lpp/internet/server_root/cgi-bin/db2www:dtw_icapi*
```

- d. Web サーバーの cgi-bin ディレクトリーを、Web サーバーの環境変数ファイル /etc/httpd.envvars の LIBPATH ステートメントに追加する。ユーザーの Web サーバーの cgi-bin ディレクトリーが、/usr/lpp/internet/server_root/cgi-bin の場合、ユーザーの LIBPATH ステートメントは、以下と類似するはずです。

```
LIBPATH=/usr/lpp/internet/bin:/usr/lpp/internet/server_root/cgi-bin
```

制約事項: Web サーバーの環境変数ファイルの LIBPATH ステートメントに、以下のディレクトリーを 2 つ以上指定しないでください。

- cgi-bin
- icapi-lib
- servlet-lib

3. Net.Data の実行可能ファイルと DLL、および実行可能なファイルと DLL へのパスの各ディレクトリーに対する許可番号が、必ず 755 になるようにしてください。
4. Web サーバーを再始動する。

Web サーバーのインストールと Web サーバーの構成ファイルのディレクティブの詳細については、以下の資料を参照してください。

- *ICSS for OS/390 概説およびインストール V2R2*、GC88-7949-00
- *ICSS OS/390:Webmaster の手引き V2R*、GC88-7534-00
- *Go Webserver 概説およびインストール V4R6*、SD88-7825-00
- *Go Webserver for OS/390 Webmaster の手引き リリース 4.6.1*、SD88-7826-00

Java サブレットと一緒に使用するための Net.Data の構成

サブレットは、CGI プログラムまたは Web サーバーの API プラグインに似た役割を果たす Java のクラスです。サブレットは Java サブレットに対応した Web サーバー上で実行され、ブラウザ上で実行される Java アプレットがブラウザの機能を拡張するのと同じように、サーバーの機能を拡張します。以下のステップに従って、Net.Data をこの Java サブレット・インターフェースを介して起動するための環境を構成します。

Net.Data のための HFS ディレクトリーを作成する時にディレクトリー構造や名前を変更していなければ、SMP/E のインストール・プロセスは、Net.Data の DLL および NetDataServlets.jar ファイルを、ディレクトリー /usr/lpp/netdata/servlet-lib にインストールしています。/usr/lpp/netdata は、ユーザーの Web サーバーのルート・ディレクトリーではないので、Web サーバーの構成に何らかの追加変更をしていない限り、Web サーバーは、Net.Data に対するクライアントの要求には応えることができません。

Web サーバーを変更するには、以下を行います。

1. Web がサブレットを実行できるようにする。(サブレットの登録および使用については、Web サーバーの資料を参照してください。)
2. 以下のアプローチのいずれかを使用して、実行可能ファイルと DLL のインストールを完了する。

- **Net.Data のディレクトリーの使用**

- a. Net.Data の servlet-lib ディレクトリーを、Web サーバーの環境変数ファイル /etc/httpd.envvars の LIBPATH ステートメントに追加する。Net.Data の servlet-lib ディレクトリーが /usr/lpp/netdata/servlet-lib の場合、LIBPATH ステートメントは、以下のようになります。

LIBPATH=/usr/lpp/internet/bin:/usr/lpp/netdata/servlet-lib

- b. NetDataServlets.jar ファイルを、Web サーバーの環境変数ファイル /etc/httpd.envvars の CLASSPATH ステートメントに追加する。Net.Data の servlet-lib ディレクトリーが /usr/lpp/netdata/servlet-lib の場合、CLASSPATH ステートメントは、以下のステートメントに類似したものになるはずです。

```
CLASSPATH=/usr/lpp/JDK1.1/lib/classes.zip:/usr/lpp/netdata/servlet-lib/  
NetDataServlets.jar
```

- **Web サーバーのディレクトリーの使用**

- a. Net.Data の DLL (libdtwndapi.so、appldll、dtwle、dtwlei、dtwsql、dtwsqlv6、filedll、odbc.dll、perl.dll、rex.dll、sys.dll) および NetDataServlets.jar ファイルを、Web サーバーの cgi-bin ディレクトリーに移動する。Web サーバーのデフォルトの cgi-bin ディレクトリーは、/usr/lpp/internet/server_root/cgi-bin です。

- b. NetDataServlets.jar ファイルを、Web サーバーの環境変数ファイル /etc/httpd.envvars の CLASSPATH ステートメントに追加する。CLASSPATH ステートメントは、以下のステートメントに類似したものになるはずです。

```
CLASSPATH=/usr/lpp/JDK1.1/lib/classes.zip:/usr/lpp/internet/server_root/cgi-bin/  
NetDataServlets.jar
```

- c. Web サーバーの cgi-bin ディレクトリーを、Web サーバーの環境変数ファイル /etc/httpd.envvars の LIBPATH ステートメントに追加する。ユーザーの Web サーバーの cgi-bin ディレクトリーが、/usr/lpp/internet/server_root/cgi-bin の場合、ユーザーの LIBPATH ステートメントは、以下と類似するはずです。

```
LIBPATH=/usr/lpp/internet/bin:/usr/lpp/internet/server_root/cgi-bin
```

- 3. Net.Data の実行可能ファイルと DLL、および実行可能なファイルと DLL へのパスの各ディレクトリーに対する許可番号が、必ず 755 になるようにしてください。
- 4. Web サーバーを再始動する。

制約事項: Web サーバーの環境変数ファイルの LIBPATH ステートメントに、以下のディレクトリーを 2 つ以上指定しないでください。

- cgi-bin
- icapi-lib
- servlet-lib

Web サーバーのインストールと Web サーバーの構成ファイルのディレクティブの詳細については、以下の資料を参照してください。

- Lotus Domino Go Webserver Planning for Installation Version 5.0 for OS/390, SC31-8690-00
- Lotus Domino Go Webserver Webmaster's Guide Version 5.0 for OS/390, SC31-8691-00

メッセージ・カタログを使用可能にする

Net.Data for OS/390 は、英語、日本語、および韓国語のメッセージ・カタログを提供します。これらのメッセージ・カタログは、Web サーバーの環境変数ファイルで使用可能にし、指定します。

Net.Dataのための階層ファイル・システム (HFS) のディレクトリーを作成したときに、そのディレクトリー構造や名前を変更していなければ、Net.Data の英語、日本語、および韓国語のメッセージ・カタログは、すでに、それぞれ
/usr/lpp/netdata/C/d2w.cat、/usr/lpp/netdata/Ja_JP/d2w.cat、および
/usr/lpp/netdata/Ko_KR/d2w.cat にインストールされています。

ディレクトリーの構造あるいは名前を変更した場合、以下のステップでは、
/usr/lpp/netdata を、ユーザーの選択ディレクトリーに置き換えてください。

1. Net.Data のメッセージ・カタログを使用可能にするには、/usr/lpp/netdata/%L/%N を、Web サーバーの環境変数ファイルの NLSPATH ステートメントに追加します。NLSPATH ステートメントは、以下と類似のものになるはずです。

NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lpp/internet/%L/%N:/usr/lpp/netdata/%L/%N

2. Net.Data が使用する特定のカタログを選択するには、Web サーバーの環境変数ファイル/etc/httpd.envvars で、LANG ステートメントの値を指定します。ステートメントの構文は、次のようになります。

LANG = *locale*

表2 を使用して、*locale* の正しい値を指定します。

表2. LANG ステートメント値

	英語	日本語	韓国語
LANG =	C	Ja_JP	Ko_KR

Net.Data がアクセスするファイルおよびデータ・セットへのアクセス権の授与

Net.Data を使用する前に、Net.Data が実行されるユーザー ID が、必ず、Net.Data のマクロで参照されているファイルおよびデータ・セットと、URL が参照するマクロへの適切なアクセス権を持つようにしてください。この意味は、これらのファイルは、ユーザー ID が明示的なアクセス権を持つ MVS データ・セットあるいは HFS ファイル、およびディレクトリーになければならない、ということです。

さらに具体的には、Net.Data を実行するユーザー ID が、必ず以下の許可を持つようにしてください。

- DSNAOINI 構成変数で指定される DB2 CLI の初期設定ファイルの読み取り
- Net.Data の初期設定ファイル、db2www.ini の読み取り
- Net.Data の実行可能ファイルと DLLの実行、および実行可能なファイルと DLLへのパスにあるディレクトリーの検索
- 適切な Net.Data のマクロ・ファイルの読み取り、および MACRO_PATH パス構成ステートメントで識別される適切なディレクトリーの検索
- 適切なファイルの実行、および EXEC_PATH パス構成ステートメントで識別される適切なディレクトリーの検索
- 適切なファイルの読み取り、および INCLUDE_PATH パス構成ステートメントで識別される適切なディレクトリーの検索
- 適切なファイルの読み取りと書き込み、および FFI_PATH パス構成ステートメントで識別される適切なディレクトリーの検索

- /tmp HFS ディレクトリーのファイルの読み込み、書き込み、および実行

第3章 ユーザー資産を保護する

OS/390 環境のインターネット機密保護は、ファイアウォール・テクノロジー、オペレーティング・システム機能、Web サーバー機能、Net.Data メカニズム、およびデータ・ソースの一部であるアクセス制御メカニズムの組み合わせで提供されます。

ユーザーの資産には、機密保護の適切なレベルを決定する必要があります。本章では、ユーザー資産を保護するために使用できるメソッドを説明し、Web サイトの機密保護のプランをたてるために使用できるその他のリソースのリファレンスも提供します。

以下のセクションには、ユーザーの資産保護のガイドラインが含まれています。ここで解説する機密保護のメカニズムは、次の通りです。

- 『ファイアウォールを使用する』
- 32ページの『ネットワーク上のユーザーのデータを暗号化する』
- 32ページの『認証を使用する』
- 33ページの『許可を使用する』
- 33ページの『Net.Data のメカニズムを使用する』

ファイアウォールを使用する

ファイアウォール は、ハードウェア、ソフトウェア、および、ネットワーク環境のリソースへのアクセスを制限するように設計されたポリシーのコレクションです。

ファイアウォールは、

- 侵入または割り込みから、内部のネットワークを保護します。
- 内部ユーザーが持ち込むデータとプログラムから、内部のネットワークを保護します。
- 外部データへの内部ユーザーのアクセスを制限します。
- ファイアウォールが侵害された場合に起こる損傷を、限定的な部分に制限します。

Net.Data は、OS/390の環境で実行する、OS/390 ファイアウォール・テクノロジーまたは同等のファイアウォール製品と組み合わせて使用されます。

OS/390 ファイアウォール・テクノロジーは、いろいろなアーキテクチャーおよびストラテジーを実装するために使用するツールキットです。このキットには、以下のツールがあります。

- IP フィルター
- プロキシ・サーバー
- Socks サーバー
- 定義域名サービス (DNS)
- 仮想私設ネットワーク

保護方法に関するファイアウォールのインストールおよび構成方法について詳しくは、OS/390 ファイアウォール技術解説書 V2R5、SD88-7094 を参照してください。

ネットワーク上のユーザーのデータを暗号化する

Secured Sockets Layer (SSL) をサポートする Web サーバーを使用しているときに、クライアント・システムと Web サーバー間で送信されるすべてのデータを暗号化できます。この機密保護の基準は、ログイン ID、パスワード、およびクライアント・システムから Web サーバーに HTML フォームで転送されるすべてのデータと、Web サーバーからクライアント・システムに送信されるすべてのデータの、暗号化をサポートしています。

認証を使用する

認証は、Net.Data 要求を作成するユーザー ID が、アプリケーション内のデータをアクセスし、更新する許可を持っているということを、確認するために使用されます。認証は、ユーザー ID とパスワードをマッチングする処理で、要求が有効なユーザー ID から発行されたものかどうかの、妥当性検査を行います。Web サーバーは、サーバーが処理するそれぞれの Net.Data 要求と、ユーザー ID を関連付けます。次に、その要求を処理しているプロセスまたはスレッドは、ユーザー ID が許可されているすべてのリソースにアクセスすることができます。

OS/390 環境では、次の 3 つのうちの 1 つの方法で Net.Data 要求を処理しているスレッドまたはプロセスと、ユーザー ID は関連付けされるようになります。

クライアントを基にした認証

ユーザーは、ローカル OS/390 ユーザー ID とパスワードをクライアント側で入力するようにプロンプト指示されます。次に Web サーバーは、ローカルの機密保護サブシステム (RACF など) を起動してユーザーを認証します。正常に認証された場合には、提供されたユーザー ID は要求と関連付けられます。特別な Web サーバー %%CLIENT%% アクセス制御ユーザー ID を使用することによって、このタイプの認証を使用可能にすることができます。

サーバーを基にした認証

Web サーバーのユーザー ID はそれぞれの要求と関連付けされているため、ユーザー ID またはパスワードを入力するようなユーザーへのプロンプト指示はされません。この選択は、Web サーバーのユーザー ID と通常関連付けされる権限レベルであるため、お勧めできません。特別な Web サーバー %%SERVER%% アクセス制御ユーザー ID を使用することによって、このタイプの認証を使用可能にすることができます。

代理の認証

いくつかの事前に定義されたコレクションのリソースにアクセスする権限を持つ代理ユーザー ID は、クライアント要求に関連づけられています。この認証タイプには、ユーザーのグループまたは要求のクラスに適切なアクセス権限をもつ代理ユーザー ID の作成が必要です。

Web サーバーがクライアント要求とユーザー ID を関連付けるために使用するアプローチは、Web サーバーの構成時に指定します。アクセス制御ユーザー ID、Web サー

パーのインストール、および Web サーバーを構成するための Protect、Protection、DefProt、UserId ディレクティブの使用について詳しくは、以下を参照してください。

- *ICSS for OS/390 概説およびインストール V2R2*, GC88-7949-00
- *ICSS OS/390:Webmaster の手引き V2R2*, GC88-7534-00
- *Go Webserver 概説およびインストール V4R6*, SD88-7825-00
- *Go Webserver for OS/390 Webmaster の手引き リリース 4.6.1*, SD88-7826-00

許可を使用する

許可によって、ユーザーに、オブジェクト、資源、および関数への完全なアクセス権または制限付きアクセス権が与えられます。DB2 および HFS のようなデータ・ソースは、独自に許可のメカニズムを備えていて、データ・ソースが管理する情報を保護しています。このような許可のメカニズムは、Net.Data 要求を実行しているプロセスに関連付けられているユーザー ID が、32 ページの『認証を使用する』で説明されているように正しく認証されていることを前提にしています。これらのデータ・ソースに対する既存のアクセス制御メカニズムは、次に、認証されたユーザー ID によって保持されている認証に基づいて、アクセスを許可または拒否します。

Net.Data のメカニズムを使用する

上記で説明したメソッドに加えて、Net.Data 構成変数またはマクロ開発技法を使用して、エンド・ユーザーの活動を制限し、データベースの設計など共通の資産を隠し、実稼働環境のユーザーが指定した入力データの値の妥当性検査を行うことができます。

Net.Data 構成変数

Net.Data は、エンド・ユーザーの活動を制限する、またはデータベースの設計を隠すために使用できる、いくつかの構成変数を提供します。

パス・ステートメントでファイル・アクセスを制御する

Net.Data は、パス構成ステートメントの設定から、Net.Data マクロが使用するファイルと実行可能プログラムのロケーションを判別します。これらのパス・ステートメントは、マクロ・ファイル、実行可能ファイル、組み込みファイル、またはその他の HFS ファイルを見つけようとするとき、Net.Data が検索する 1 つまたは複数のディレクトリーを示します。このパス・ステートメントにディレクトリーを選択して組み込むことによって、ブラウザーで明示的にユーザーがアクセスできるファイルを制御することができます。パス・ステートメントの詳細については 5 ページの『第2章 Net.Data のインストールおよび構成』を参照してください。

また、『許可を使用する』で説明されている許可検査を使用して、34 ページの『マクロ開発技法』の説明通りに、INCLUDE ステートメントのファイル名が変更できないことを確認します。

実動システムでは SHOWSQL を使用不可にする

SHOWSQL 変数によって、Net.Data 関数内に指定された SQL ステートメントを、Net.Data が Web ブラウザーに表示することを指定できます。この変数

は、アプリケーション内の SQL の開発およびテストで主に使用され、実動システムで使用するものではありません。

以下のメソッドのいずれかを使用して、実稼働環境の SQL ステートメントの表示を使用不可にすることができます。

- DTW_SHOWSQL 構成変数をサポートしている Net.Data のバージョンを使用している場合、Net.Data 初期設定ファイルでこの変数を使用して、Net.Data マクロ内の SHOWSQL 設定をオーバーライドします。構文および追加情報については、14ページの『DTW_SHOWSQL: SHOWSQL 構成変数を使用可能または使用不可にする』を参照してください。

- 『マクロ開発技法』の説明に従って、DTW_ASSIGN() 関数を使用します。

SHOWSQL Net.Data 変数の構文および例については、Net.Data 解説書 の変数の章の、SHOWSQL を参照してください。

実稼働環境の直接要求を使用可能にすることが適切であるかどうかを考慮する

Net.Data を起動する直接要求メソッドによって、ユーザーは、SQL ステートメントの実行や、Perl、REXX、または C プログラムを URL で直接指定することができます。マクロ要求メソッドによってユーザーは、1つのマクロで定義または呼び出されたこれらの SQL ステートメントおよび関数だけを、実行することができます。

直接要求によってユーザーは非常に多くの関数を実行することができるため、直接要求の使用を許可するかどうかは細心の注意が必要です。このメソッドの起動を使用可能にするときには、Net.Data 要求を処理する際に使用されるユーザー ID が、適切な許可レベルを持っていることを確認します。

DTW_DIRECT_REQUEST 構成変数を使用して、直接要求を使用不可にすることができます。構文および追加情報については、12ページの『DTW_DIRECT_REQUEST: 直接要求変数を使用可能にする』を参照してください。

マクロ開発技法

Net.Data は、ユーザーが入力変数に値を指定できるようにするための、いくつかのメカニズムを提供します。マクロが本来の機能通りに実行していることを確認するには、そのマクロがこれらの入力変数を妥当性検査する必要があります。また、ユーザーのデータベースおよびアプリケーションは、ユーザーが読み込みを許可されているデータへのアクセスに、アクセスを制限するような設計も行う必要があります。

Net.Data マクロをコーディングするときには、以下の開発技法を使用します。これらの技法は、アプリケーションを目的どおりに実行し、データへのアクセスを正式に許可されたユーザーだけに確実に制限するために役立ちます。

Net.Data 変数がある URL で、まったくオーバーライドできなくする

URL 内の Net.Data 変数のユーザー設定は、マクロが変数の初期化に使用する DEFINE ステートメントの効果をオーバーライドします。これによって、マクロを実行する方法が変わることがあります。これを避けるためには、DTW_ASSIGN() 関数を使用して Net.Data 変数を初期化します。

例: Net.Data SHOWSQL 変数の定義は %DEFINE SHOWSQL="NO" を使用せずに、@DTW_ASSIGN(SHOWSQL,"NO") を使用して行います。また、SHOWSQL=YES などの照会ストリング割り当ては、マクロ設定をオーバーライドしません。

以下のメソッドのいずれかを使用して、実稼働環境の SQL ステートメントの表示を使用不可にすることができます。

- DTW_SHOWSQL 構成変数をサポートしている Net.Data のバージョンを使用している場合、Net.Data 初期設定ファイルでこの変数を使用して、Net.Data マクロ内の SHOWSQL 設定をオーバーライドします。構文および追加情報については、14ページの『DTW_SHOWSQL: SHOWSQL 構成変数を使用可能または使用不可にする』を参照してください。
- 上記例の説明に従って DTW_ASSIGN() 関数を使用して、SHOWSQL の値を割り当ててオーバーライドされないようにします。

SHOWSQL Net.Data 変数の構文および例については、*Net.Data 解説書* の変数の章の、SHOWSQL を参照してください。

また、DTW_ASSIGN を使用して、RPT_MAX_ROWS または START_ROW_NUM などの他の Net.Data 変数が、オーバーライドされないようにすることもできます。これらの変数について詳しくは、*Net.Data 解説書* の変数の章を参照してください。

SQL ステートメントに対して、ユーザー・アプリケーションの本来の振る舞いを変えてしまうような変更ができなくなっていることの、妥当性検査をする

マクロ内の SQL ステートメントに Net.Data 変数を追加することによって、ユーザーは、SQL ステートメントを実行する前に、その SQL ステートメントを動的に変更することができます。ユーザーが指定した入力値の妥当性検査、および変数リファレンスを含んでいる SQL ステートメントの、予期しない方法による変更の防止は、マクロ開発者が行う必要があります。ユーザーの Net.Data アプリケーションは、URL でユーザーが指定した入力値の妥当性検査を行います。これによって、Net.Data アプリケーションは無効な入力を拒否できます。検証設計プロセスは、以下のステップを行う必要があります。

1. 入力された構文の有効性を識別します。たとえば、顧客 ID は文字で始まり、英数字だけで構成されます。
2. 入力の誤り、意図的な害のある入力、または Net.Data アプリケーションの内部資産にアクセスするために入力された入力データを取り込むことによって、発生する可能性のある障害は何かを判別します。
3. アプリケーションの要件に合う入力検査ステートメントを、マクロに組み込みます。この検査は、入力データの構文およびその構文の使用方法によって異なります。簡単な検査を行う場合には、入力データの無効な内容を検査するか、入力データ・タイプの検査をする Net.Data を起動するだけで十分です。入力の構文がさらに複雑な場合には、マクロ開発者は、その入力の有効であるかを検査するために、その入力を部分的な解析、または完全な解析が必要となる場合があります。

例 1: SQL ステートメントを検査するために DTW_POS() ストリング関数を使用する

```
%FUNCTION(DTW_SQL) query1() {
    select * from shopper where shlogid = '${shlogid}'
}%
```

shlogid 変数の値が、shopper ID になります。この目的は、SELECT ステートメントから戻される行を、shopper ID が識別された shopper に関する情報を含む行だけに、制限することです。ただし、ストリング 『smith' または shlogid<>'smith』 が、変数 shlogid の値として渡された場合は、照会は以下のようになります。

```
select * from shopper where shlogid = 'smith' or shlogid<>'smith'
```

オリジナル SQL SELECT ステートメントを変更したこのユーザー・バージョンは、shopper テーブル全体を戻します。

Net.Data ストリング関数は、ユーザーが不適切な方法で SQL ステートメントを変更していないことを確認するために使用することができます。たとえば、以下のロジックは、shlogid 変数と関連付けられた入力値が、単一の shopper ID からなることを確認するために使用することができます。

```
@DTW_POS(" ", ${shlogid}, result)
%IF (result == "0")
    @query1()
%ELSE
    %{ perform some sort of error processing %}
%ENDIF
```

例 2: DTW_TRANSLATE() の使用

ユーザーのアプリケーションが、入力変数 number_of_orders で指定された値が整数かどうかを検査する必要があるとします。これを行う方法の 1 つは、数字 0-9 を除くすべてのキーボード文字を含む変換テーブル

input_translation_table を作成して、DTW_TRANSLATE および DTW_POS ストリング関数を使用してその入力の妥当性検査を行うことです。

```
@DTW_TRANSLATE(number_of_orders, "x", input_translation_table, "x", string_out)

    @DTW_POS("x", string_out, result)
    %IF (result = "0")
    %{ continue with normal processing %}
%ELSE
    %{ perform some sort of error processing %}
%ENDIF
```

ストアード・プロシージャ内の SQL ステートメントは、Web ブラウザーからユーザーが修正できないこと、およびユーザー提供の入力パラメーター値は、入力パラメーターと関連付けられた SQL データ型によって制約されないことに注意してください。Net.Data ストリング関数を使用して、ユーザーの入力値の妥当性検査を行うことができない状態では、ストアード・プロシージャを使用することができます。

INCLUDE ステートメントのファイル名に対して、ユーザー・アプリケーションの本来の振る舞いを変えてしまうような変更をできなくする

Net.Data 変数を使用して INCLUDE ステートメントでファイル名の値を指定すると、インクルードされるファイルは、INCLUDE ファイルが実行されるまで決定されません。マクロ内でこの変数の値をセットして、さらに、ユーザーがブラウザーでマクロが指定する値をオーバーライドできないようにする

場合には、`DEFINE` の代わりに `DTW_ASSIGN` を使用して変数の値を設定する必要があります。ユーザーがブラウザーでファイル名の値を提供できるようにする場合には、ユーザーのマクロは、提供された値の妥当性検査を行う必要があります。

例: `filename="../../x` のような照会ストリング割り当てによって、`INCLUDE_PATH` 構成ステートメントで通常指定されないディレクトリーから、ファイルはインクルードされます。`Net.Data` 初期設定ファイルは、以下のパス構成ステートメントを含んでいるとします。

```
INCLUDE_PATH /usr/lpp/netdata/include
```

また `Net.Data` マクロは、以下の `INCLUDE` ステートメントを含んでいるとします。

```
%INCLUDE "${filename}"
```

`filename="../../x` の照会ストリング割り当ては、ファイル `/usr/lpp/x` をインクルードします。このファイルは、`INCLUDE_PATH` 構成ステートメント指定ではインクルードされません。

`Net.Data` ストリング関数は、指定されたファイル名がそのアプリケーションに適切であることを検証するために使用することができます。たとえば、以下のロジックは、ファイル名の変数と関連付けられた入力値がストリング `".."` を含まないことを確認するために使用されます。

```
@DTW_POS(".", ${filename}, result)
%IF (result > "0")
  %{ perform some sort of error processing %}
%ELSE
  %{ continue with normal processing %}
%ENDIF
```

ユーザー要求が他のユーザーに関する機密データにアクセスしないように、データベースと照会を設計する

一部のデータベース設計には、単一のテーブルにユーザーの機密データを収集するものがあります。`SQL SELECT` 要求が、いくつかの方法で限定されていない限り、このアプローチによってすべての機密データは、Web ブラウザーですべてのユーザーが使用可能になります。

例: 以下の `SQL` ステートメントは、変数 `order_rn` で識別された順序に関する順序情報を戻します。

```
select setsstatcode, setsfailtype, mestname
from   merchant, setstatus
where  merfnbr   = setsmenbr
and    setsornbr = $(order_rn)
```

このメソッドは、ブラウザーのユーザーが、ランダムな順番を指定して、可能であれば、他の顧客の順序に関する機密情報を取得することを許可します。この種の公開タイプにおける保護の方法としては、以下のような変更をする方法があります。

- 特定の行の中の順序情報と関連付けられた顧客を識別する、順序情報のテーブルに、列を追加します。

- SQL SELECT ステートメントを変更して、ブラウザでユーザーが提供した認証された顧客 ID によって、SELECT が限定されていることを確認します。

たとえば、shlogid が、順序と関連付けられた顧客 ID を含む列で、SESSION_ID が、ブラウザのユーザーの認証された ID を含む Net.Data 変数である場合、以下のステートメントで直前の SELECT ステートメントを置き換えることができます。

```
select setsstatcode, setsfailtype, mestname
  from merchant, setstatus
 where merfnbr   = setsmenbr
 and   setsornbr = $(order_rn)
 and   shlogid   = $(SESSION_ID)
```

Net.Data 隠蔽変数を使用する

Web ブラウザーで HTML ソースを見ることがあるユーザーから、Net.Data マクロのいろいろな特性を隠すために、Net.Data 隠蔽変数を使用することができます。たとえば、データベースの内部構造を隠すことができます。隠蔽変数の詳細については、65ページの『隠し変数』を参照してください。

第4章 Net.Data を起動する

本章では、いろいろな Web サーバー・インターフェースを使用した、Net.Data の起動方法について説明しています。起動メソッドのいずれかを使用する前に、Net.Data は、指定されたインターフェースに対して、最初に構成されていなければなりません。Net.Data を構成することで、以下の Web サーバー・インターフェースが使用できます。

- Common Gateway Interface (CGI)
- Lotus Domino Go Web server (GWAPI)
- Internet Connection Server (ICAPI)
- Java サブレット

これらのインターフェースに対する Net.Data の構成については、5ページの『第2章 Net.Data のインストールおよび構成』を参照してください。Web サーバーを構成するとき、Net.Data がどのように起動されるかを決定します。

以下のセクションでは、Net.Data が受け入れる要求の種類、およびいろいろな API およびサブレットを使用して Net.Data を起動するために使用できるメソッドを説明しています。

- 『CGI、ICAPI、または GWAPI を使用して Net.Data を起動する』
- 48ページの『Java Servlets によって Net.Data を起動する』

CGI、ICAPI、または GWAPI を使用して Net.Data を起動する

Net.Data を起動するときに使用するメソッドとは関係なく、指定可能な要求は 2 種類あります。これは、マクロを実行するかどうか、または単一の SQL ステートメント、ストアド・プロシージャ、または関数を実行するかどうかに依存します。

マクロ要求

Net.Data が指定されたマクロを実行するように、指定します。

直接要求

Net.Data が SQL ステートメント、ストアド・プロシージャ、または関数を実行するように、指定します。この要求は、以下を指定します。

- 言語環境の名前
- 関数の呼び出しに必要なすべてのパラメーター値と、SQL ステートメントまたは関数の名前。
- SQL ステートメントまたは関数の呼び出しに必要なフォーム・データ

単一の SQL 照会を作成したり、または DB2 ストアド・プロシージャ、REXX プログラム、または Perl 関数などの単一の関数の呼び出しをする Web 開発者は、データベースに直接要求を発行することができます。直接要求は、Net.Data マクロを必要とする複雑な Net.Data アプリケーション・ロジックではありません。したがって、Net.Data マクロ処理プログラムをバイパスします。直接要求パラメーターは、パフォーマンス良く処理を行うために、適切な言語環境に渡されます。

図3 は、マクロ要求と直接要求の違いを示しています。マクロ要求は通常、その要求の URL 内のマクロを指定します。フォーム・データを使用することもできます。直接要求では URL 内のマクロは指定されませんが、フォーム・データを引き続き使用できます。

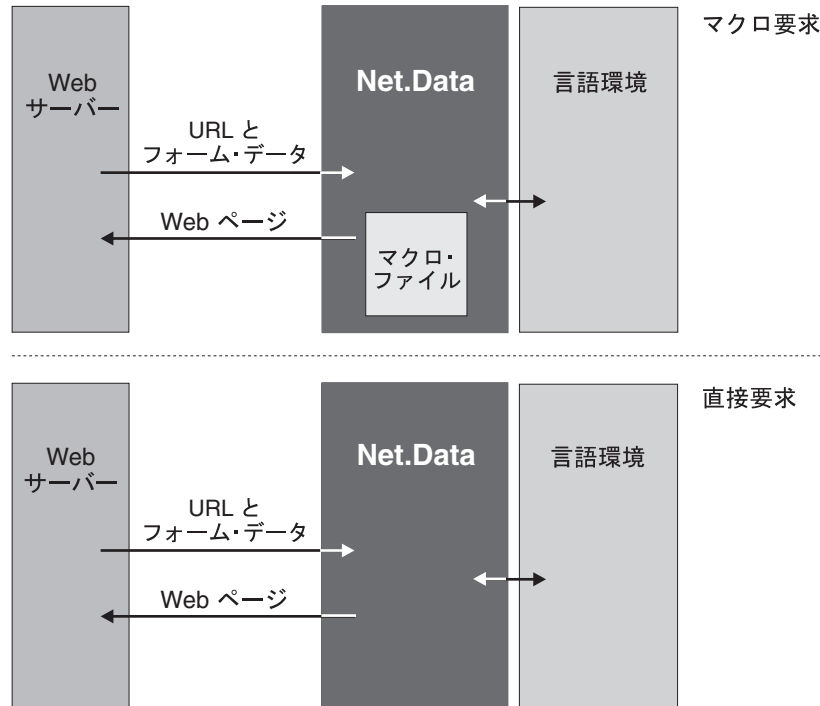


図3. マクロ要求対直接要求

Net.Data を ICAPI または GWAPI とともに使用するような構成にするために、Net.Data を起動するための構文は、CGI とともに使用するような構成をするために Net.Data を起動する構文と同じです。マクロおよび直接要求では、どちらの場合も Net.Data は、URL を使用して起動されます。この URL は、ユーザーが直接入力できます。また、HTML リンクまたは HTML 形式として、HTML ページにコーディング可能です。Web サーバーは、CGI、ICAPI、または GWAPI を使用して、Net.Data を起動します。

マクロ要求の場合、Net.Data マクロの名前、および Net.Data マクロ内で実行される HTML ブロックの名前を、URL 内で指定します。直接要求の場合、Net.Data 言語環境の名前、SQL ステートメントまたは関数の名前、および追加の必須パラメーター値を、URL 内で指定します。これらの値は、Net.Data によって定義された構文を使用して、指定します。

以下のセクションでは、これらの起動要求を詳細に説明しています。

- 『マクロで Net.Data を呼び出す (マクロ要求)』
- 43ページの『マクロを使用しない Net.Data の起動 (直接要求)』

マクロで Net.Data を呼び出す (マクロ要求)

このセクションでは、マクロを指定した Net.Data の起動の方法を示します。

以下の構文ステートメントに Net.Data の起動方法を示します。この例では、すでに 24ページの『CGI と一緒に使用する Net.Data の構成』および 25ページの『ICAPI あるいは GWAPI と一緒に使用するための Net.Data の構成』で記述してあるように、Net.Data が Net.Data ディレクトリーを使用して構成済みであることを前提としています。

- URL:

`http://server/Net.Data_invocation_path/filename/block[?name=val&...]`

パラメーター:

server Web サーバーの名前およびパスを指定します。サーバーがローカル・サーバーであれば、このサーバー名を省略し、相対 URL を使用することができます。

Net.Data_invocation_path

Net.Data ロード・モジュールのパスおよびファイル名。たとえば、`/netdata-cgi/db2www/` のようになります。

filename

Net.Data マクロ・ファイルの名前を指定します。Net.Data はこのファイル名を検索し、MACRO_PATH 初期設定パス変数に定義されたパス・ステートメントに一致させるよう試みます。詳しくは、16ページの『MACRO_PATH』を参照してください。

block 参照される Net.Data マクロの HTML ブロックの名前を指定します。

method フォームで使用される HTML メソッドを指定します。

?name=val&...

Net.Data に渡される 1 つまたは複数のオプション・パラメーターを指定します。

その上でブラウザで URL を直接指定する、または以下のように HTML リンクまたはフォームで URL を使用することができます。

- HTML リンク:

`any text`

- HTML フォーム:

`<FORM METHOD=method
ACTION="URL">any text</FORM>`

パラメーター:

method フォームで使用される HTML メソッドを指定します。

URL Net.Data マクロの実行に使用する URL を指定します。パラメーターは上記のとおりです。

例

以下の例は、Net.Data の各種起動方法を示しています。

例 1: HTML リンクを使用した Net.Data の起動

```
<A HREF="http://server/netdata-cgi/db2www/myMacro.d2w/report">
.
.
.
</A>
```

例 2: フォームを使用した Net.Data の起動

```
<FORM METHOD=POST
  ACTION="http://server/netdata-cgi/db2www/myMacro.d2w/report">
.
.
.
</FORM>
```

以下のセクションでは、HTML リンクおよびフォームについて、およびこれらを使用した Net.Data 起動方法について説明します。

- 『HTML リンク』
- 43ページの『HTML フォーム』

HTML リンク

Web ページのオーサリングを行う場合、HTML リンクを作成し、HTML ブロックの実行することができます。ブラウザでユーザーが HTML リンクに定義されたテキストやイメージをクリックすると、Net.Data はそのマクロ内の HTML ブロックを実行します。

HTML リンクを作成するには、HTML `<a>` タグを使用します。Net.Data マクロへのハイパーリンクに使用するテキストまたはグラフィックを決定して、これを `<a>` および `` タグで囲みます。`<a>` タグの `HREF` 属性にマクロおよび HTML ブロックを指定します。

次の例は、ユーザーが Web ページ上でテキスト「モニターをすべてリストする (List all monitors)」を選択したときに実行される SQL 照会とのリンクを示しています。

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

リンクをクリックすると、listA.d2w 名のマクロが呼び出されます。これには以下の例のように "report" という HTML ブロックがあります。

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

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

この照会は、EQPTABLE 表の中に記述された各モニターに関する型式番号、コスト、および記述情報を持つ表を戻します。この例は、照会の結果をデフォルトのレポートで表示します。REPORT ブロックを使用してレポートをカスタマイズする方法に関しては、80ページの『レポート・ブロック』を参照してください。

HTML フォーム

HTML フォームを使用して、Net.Data マクロの実行を動的にカスタマイズすることができます。フォームによって値を入力することが可能になり、マクロの実行と Net.Data が生成する Web ページの内容を変更することができます。

次の例は、42ページの『HTML リンク』でのモニター・リストの例を基本としたもので、ユーザーはブラウザで、簡単な HTML フォームを使用して製品タイプを選択し、その情報を表示することができます。

```
<H1>Hardware Query Form</H1>
<HR>
<FORM METHOD=POST ACTION="/netdata-cgi/db2www/equip1st.d2w/report">
<P>What type of hardware do you want to see?
<MENU>
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="MON" checked> Monitors
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="PNT"> Pointing devices
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="PRT"> Printers
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="SCN"> Scanners
</MENU>

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

ブラウザで選択し、「実行 (Submit)」ボタンをクリックすると、Web ブラウザーは FORM タグの ACTION パラメーターを処理し、Net.Data が起動します。次に Net.Data は、equip1st.d2w マクロの中の HTML レポート・ブロックを実行します。

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

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

上記の例では、SQL ステートメント中の TYPE=\${hdware} の値が HTML フォーム入力から取得されます。

ROW ブロックで使用される変数の詳細な記述に関しては、“*Net.Data 解説書*”を参照してください。

マクロを使用しない Net.Data の起動 (直接要求)

このセクションでは、直接要求を使用して、Net.Data を起動する方法を示します。直接要求を使用するときには、URL にマクロの名前を指定しないでください。その代わりに、Net.Data によって定義された構文を使用して、Net.Data 言語環境、SQL ステートメントまたは実行するプログラム、および追加の必須パラメーター値を、URL で

指定します。直接要求を使用可能にする、および使用不可にする方法については、12ページの『DTW_DIRECT_REQUEST: 直接要求変数を使用可能にする』を参照してください。

SQL ステートメントまたはプログラム、およびその他指定されたパラメーターは、直接指定された言語環境に渡され、処理されます。直接要求によって、Net.Data がマクロを読み込んで処理する必要がなくなるため、パフォーマンスが向上します。SQL、ODBC、System、Perl、および REXX Net.Data によって提供される言語環境は、直接要求をサポートしています。ユーザーは URL、HTML フォーム、またはリンクを使用して Net.Data を呼び出すことができます。

直接要求は、URL またはフォーム・データの照会ストリングでパラメーターを渡すことによって、Net.Data を起動します。次の例は、ユーザーが直接要求を指定するコンテキストを図示しています。ここでは、24ページの『CGI と一緒に使用する Net.Data の構成』 および 25ページの『ICAPI あるいは GWAPI と一緒に使用するための Net.Data の構成』です。すでに説明されているように、Net.Data ディレクトリーを使用して Net.Data が構成されていることを前提とし、ユーザーが Perl 言語環境に対して直接要求を指定するコンテキストを図示しています。

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

ここで *direct_request* は、直接要求の構文を表しています。たとえば、以下の HTML リンクは、直接要求を含んでいます。

```
<A HREF="http://server/netdata-cgi/db2www/?LANGENV=DTW_PERL&FUNC=my_perl(hi)">すべてのテキスト</A>
```

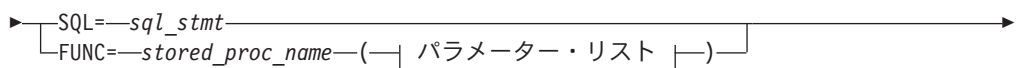
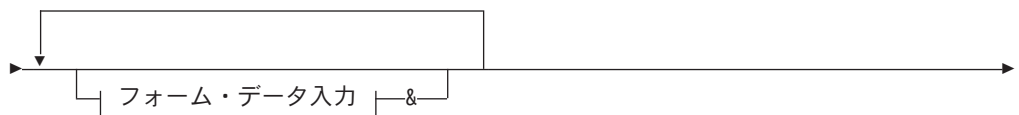
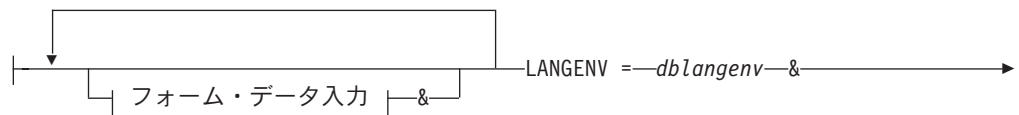
直接要求の構文

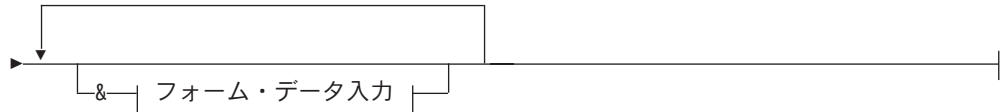
直接要求で Net.Data を起動する構文は、データベースや非データベース言語環境への呼び出しを含んでいます。

構文

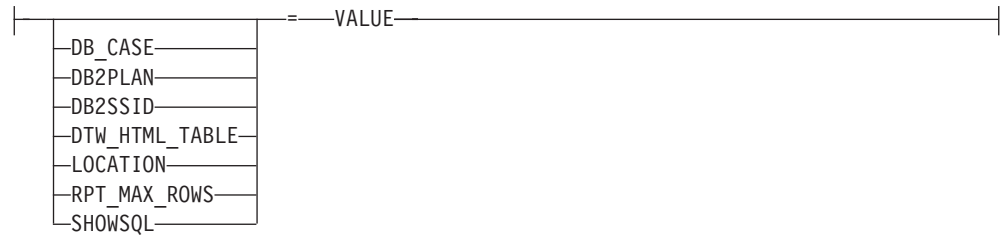


データベース言語環境呼び出し:

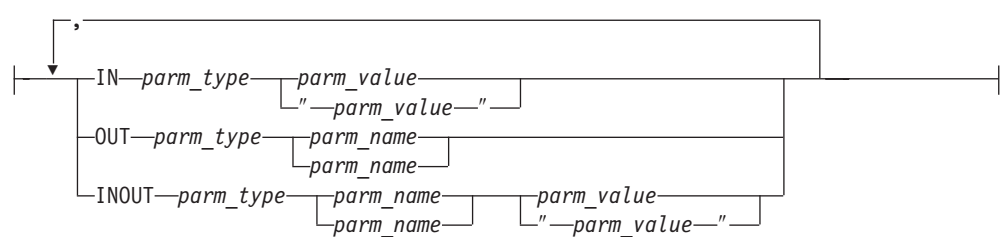




フォーム・データ入力:



パラメーター・リスト:



非データベース言語環境呼び出し:



パラメーター

データベース言語環境呼び出し

データベース言語環境を起動する Net.Data への直接要求を指定します。

フォーム・データ入力

パラメーター。SQL 変数の設定を指定する、または単純な HTML フォーマットを要求することができます。これらの変数について詳しくは、*Net.Data* 解説書の変数の章を参照してください。

DB_CASE

SQL ステートメントの文字ケース (大文字または小文字) を指定します。

DB2PLAN

ローカルの DB2 サブシステムをアクセスするときに使用する DB2 プランを指定します。

DB2SSID

ローカルの DB2 サブシステムをアクセスするときに使用する DB2 サブシステム ID を指定します。

DTW_HTML_TABLE

Net.Data が、HTML テーブル、または事前にフォーマットされたテキスト・テーブルの、どちらを戻すべきかどうかを指定します。

LOCATION

ローカルの DB2 サブシステムが SQL 要求を渡すリモート・サーバーの名前を指定します。

RPT_MAX_ROWS

関数がレポートに戻すテーブルの最大行数を指定します。

SHOWSQL

Net.Data が実行される SQL ステートメントを隠すか表示するかを指定します。

START_ROW_NUM

レポートの開始に使用する関数のテーブルの行数を指定します。

VALUE

Net.Data 変数の値を指定します。

LANGENV

SQL ステートメントまたはストアド・プロシージャ呼び出しの、ターゲット言語環境を指定します。

dblangenv

データベース言語環境の名前:

- DTW_SQL
- DTW_ODBC

SQL

直接要求がインライン SQL ステートメントの実行を指定していることを示しています。

sql_stmt

動的 SQL を使用して実行されるすべての有効な SQL ステートメントを含む文字列を指定します。

FUNC

直接要求が、ストアド・プロシージャの実行を指定していることを示しています。

stored_proc_name

有効な DB2 ストアド・プロシージャを指定します。

parm_type

DB2 ストアド・プロシージャに対する有効なパラメーター・タイプを指定します。

parm_name

有効なパラメーター名を指定します。

parm_value

DB2 ストアド・プロシージャに対する有効なパラメーターの値を指定します。

IN Net.Data がパラメーターを使用して入力データをストアド・プロシージャに渡すことを指定します。

INOUT

Net.Data がパラメーターを使用して、入力データをストアド・プロシージャに渡し、さらに言語環境からの出力データを戻すことを指定します。

OUT

言語環境がパラメーターを使用して、ストアド・プロシージャからの出力データを戻すことを指定します。

非データベース言語環境呼び出し

非データベース言語環境を起動する Net.Data への直接要求を指定します。

LANGENV

その関数を実行するターゲット言語環境を指定します。

lang_env

以下の非データベース言語環境の名前を指定します。

- DTW_PERL
- DTW_REXX
- DTW_SYSTEM

FUNC

直接要求がプログラムの実行を指定していることを示しています。

program_name

実行される関数を含んでいるプログラムを指定します。

parm_value

その関数に有効なパラメーターの値を指定します。

直接要求の例

次の例は、直接要求メソッドを使用する場合に、Net.Data を起動するいろいろな方法を示しています。この例では、すでに

24ページの『CGI と一緒に使用する Net.Data の構成』 および 25ページの『ICAPI あるいは GWAPI と一緒に使用するための Net.Data の構成』 で記述してあるように、Net.Data は Net.Data ディレクトリーを使用して構成されていることを前提にしています。

HTML リンク: 例 1: Perl 言語環境を起動して、Net.Data 初期設定ファイルの EXEC パス・ステートメントの Perl スクリプトを呼び出すリンク

```
<A HREF="http://server/netdata-cgi/db2www/?LANGENV=DTW_PERL&FUNC=my_perl(hi)">
すべてのテキスト</A>
```

例 2: 直前の例のように、Perl 言語環境を起動しますが、二重引用符およびスペース文字に対して、URL 形式で符号化された値でストリングを渡します。

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

ヒント: URL では、スペースや二重引用符などの特定の文字は符号化しなければなりません。この例では、パラメーター値の中の二重引用符やスペースは、 %22 および + 文字に符号化します。ユーザーは、組み込み関数 DTW_URLESCSEQ を使用して、URL で符号化されなければならないテキストをすべて符号化することができます。DTW_URLESCSEQ 関数について詳しくは、*Net.Data* 解説書の説明を参照してください。

HTML フォーム: 例 1: SQL 言語環境を使用する SQL 照会を実行する HTML フォーム。

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

これは、SQL ステートメント内の変数置換の例で、WHERE 文節は動的になります。

URL: 例 1: SQL 言語環境を使用する SQL 照会を実行する URL

http://server/netdata-cgi/db2www/?LANGENV=DTW_SQL&SQL=select+++from+customer

例 2: Perl 言語環境を起動して、Net.Data 初期設定ファイルの EXEC パス・ステートメントにない実行可能ファイルを呼び出す URL

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

例 3: システム言語環境を起動し、外部 Perl スクリプトを呼び出す URL

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

例 4: REXX 言語環境を起動し、プログラムにパラメーターを渡す URL

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

例 5: ストアード・プロシージャを呼び出し、SQL 言語環境にパラメーターを渡す URL

http://server/netdata-cgi/db2www/?LANGENV=DTW_SQL&FUNC=MY_STORED_PROC
(IN+CHAR(30)+Salaries)&DTW_HTML_TABLE=YES

Java Servlets によって Net.Data を起動する

CGI プログラムまたは Web サーバー API プラグインに類似した役割を果たすサーブレットおよび Java クラス。サーブレットは、CGI 的な関数を実行するために、Java サーブレットが使用可能な Web サーバーによって使用されます。サーブレットは独自のグラフィカル・ユーザー・インターフェースを持っていません。ただしサーブレットのクラスは、ローカルで、またはネットワークから動的にロードされ、URL アドレスを使用して (リモート) またはクラス名によって (ローカル) 呼び出されます。

Net.Data は、Net.Data マクロ、単一 SQL ステートメント、ストアード・プロシージャ、および OS/390 の関数を起動するために使用できるサーブレットを提供します。

このサーブレットは、URL から、または Server-Side-Include (SSI) として、実行されます。 Net.Data には次の 2 つのサーブレットがあります。

MacroServlet (com.ibm.netdata.servlets.MacroServlet)

Net.Data マクロを実行します。

Server-Side-Includes (SSI) によってマクロを実行し、HTML ファイルに複数のマクロを組み込むことができます。

FunctionServlet (com.ibm.netdata.servlets.FunctionServlet)

以下を指定することによって、マクロを使用せずに Net.Data を起動します。

- 言語環境の名前。
- 関数の呼び出しに必要なすべてのパラメーター値と、SQL ステートメントまたは関数の名前。
- SQL ステートメントまたは関数の呼び出しに必要なフォーム・データ。

関数サーブレットは直接要求の機能を提供しますが、Java インターフェースを使用しています。詳しくは、43ページの『マクロを使用しない Net.Data の起動 (直接要求)』を参照してください。

MacroServlet を使用した Net.Data の起動

URL または HTML ファイルの SSI から、このサーブレットを呼び出すことができます。

構文および例

- URL:

```
http://server/servlet/com.ibm.netdata.servlets.MacroServlet?MACRO=macro_value&BLOCK=block_value&parmn=valuenn
```

たとえば、以下のようにします。

```
http://server/servlet/com.ibm.netdata.servlets.MacroServlet?MACRO=companies.d2w&BLOCK=gatherinfo
```

- SSI:

```
<servlet code="com.ibm.netdata.servlets.MacroServlet">
  <param name="MACRO" value="my_macro">
  <param name="BLOCK" value="my_block">
  <param name="parmn" value="valuenn">
</servlet>
```

たとえば、以下のようにします。

```
<servlet code="com.ibm.netdata.servlets.MacroServlet">
  <param name="MACRO" value="companies.d2w">
  <param name="field1" value="custno">
</servlet>
```

パラメーター

MACRO

必須 既存の Net.Data マクロへの完全修飾パス。

BLOCK

指定された Net.Data マクロ内の実行される HTML ブロックの名前を指定します。デフォルトのブロックは、reportです。

parmn

その他のユーザーのマクロに必要とされるパラメーターを指定します。

FunctionServletを使用して Net.Data を起動する

URL または HTML ファイルの SSI から、このサーブレットを呼び出すことができます。このサーブレットを呼び出すと、関数、SQL ステートメント、またはストアド・プロシージャを起動することが可能です。

構文および例

- URL:

- 関数の起動:

```
http://server/servlet/com.ibm.netdata.srvlets.  
FunctionServlet?LANGENV=language&  
FUNC=function_name&parmn=valuenn
```

たとえば、以下のようにします。

```
http://server/servlet/com.ibm.netdata.srvlets.  
FunctionServlet?LANGENV=DTW_REXX&  
FUNC=custinput.cmd&field1=custno
```

- SQL ステートメントの起動:

```
http://server/servlet/com.ibm.netdata.srvlets.  
FunctionServlet?LANGENV=database_lang&  
SQL=SQL_statement&parmn=valuenn
```

たとえば、以下のようにします。

```
http://server/servlet/com.ibm.netdata.srvlets.  
FunctionServlet?LANGENV=DTW_SQL&  
SQL=select+lastname+from+customers
```

- ストアド・プロシージャの起動:

```
http://server/servlet/com.ibm.netdata.srvlets.FunctionServlet  
?LANGENV=DTW_SQL&FUNC=stored_procedure_name(parameter_list)
```

たとえば、以下のようにします。

```
http://server/servlet/com.ibm.netdata.srvlets.FunctionServlet  
?LANGENV=DTW_SQL&FUNC=myStoredProc(IN+CHAR(20)+"inval")
```

- SSI:

- 関数の起動:

```
<servlet code="com.ibm.netdata.srvlets.MacroServlet">  
  <param name="LANGENV" value="language">  
  <param name="FUNC" value="function_name">  
  <param name="parmn" value="valuenn">  
</servlet>
```

- SQL ステートメントの起動:

```
<servlet code="com.ibm.netdata.srvlets.MacroServlet">  
  <param name="LANGENV" value="language">  
  <param name="SQL" value="SQL_statement">  
  <param name="parmn" value="valuenn">  
</servlet>
```

- ストアド・プロシーチャーの起動:

```
<servlet code="com.ibm.netdata.servlets.MacroServlet">
  <param name="LANGENV" value="language">
  <param name="FUNC" value="stored_procedure">
  <param name="parmn" value="valuen">
</servlet>
```

たとえば、以下のようにします。

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
  <param name="LANGENV" value="DTW_SQL">
  <param name="FUNC" value="myStoredProc(IN CHAR(20) invalue)">
</servlet>
```

パラメーター

LANGENV

関数 (たとえば、DTW_SQL または DTW_REXX) を処理するために呼び出される Net.Data 言語環境を指定します。

FUNC

実行する関数を含むプログラムの名前、またはストアド・プロシーチャーの場合には、ストアド・プロシーチャー名とパラメーターを指定します。たとえば、my_rexx。ここで my_rexx は、実行可能 REXX ファイルの名前です。 **parmn** キーワードを使用して、関数への入力パラメーターを指定します。

SQL

データベースをアクセスする SQL ステートメントまたはストアド・プロシーチャーを指定します。たとえば、"select * from employee" です。

parmn

その他の関数に必要とされるパラメーターを指定します。

第5章 Net.Data のマクロ開発

Net.Data のマクロは、Net.Data のマクロ言語の連続した構成要素で構成されるテキスト・ファイルで、以下を行います。

- Web ページのレイアウトを指定する
- 変数と関数を定義する
- Net.Data の組み込み関数またはマクロに定義された関数の呼び出し
- 処理出力をフォーマットし、Web ブラウザーに戻す

Net.Data のマクロには、図4 に示されているように、宣言部分および表示部分という 2 つの構成部分があります。

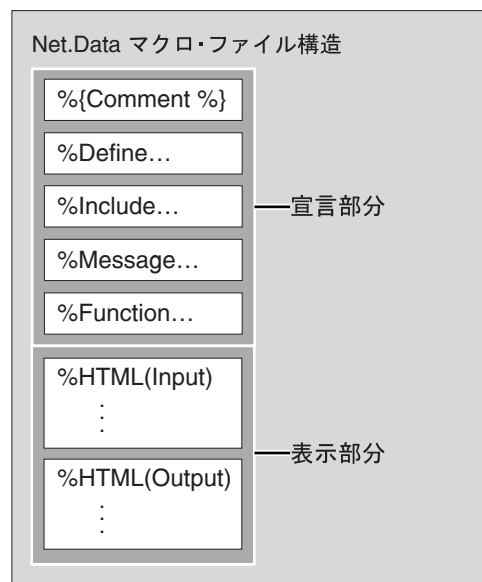


図4. マクロ構造

- 宣言部分 には、マクロにおける変数および関数の定義が含まれます。
- 表示部分 には、Web ページのレイアウトを指定する HTML ブロックが含まれます。HTML ブロックは、HTML および JavaScript などユーザーの Web ブラウザーがサポートするテキスト表示ステートメントから構成されます。

これらの部分は、複数回、任意の順序で使用することができます。マクロの部分および構成要素の構文については、*Net.Data* 解説書 を参照してください。

本章では、Net.Data のマクロを構成しているさまざまなブロックと、マクロ・ファイルを記述するために使用できる方法について考察します。

- 54ページの『Net.Data マクロの分析』
- 58ページの『Net.Data のマクロ変数』
- 70ページの『Net.Data の関数』
- 79ページの『マクロでの Web ページ生成』
- 85ページの『マクロにおける条件付き論理のループ』

Net.Data マクロの分析

マクロは、以下の 2 つの部分から構成されます。

- 宣言部分。これは、表示部分で使用される定義を含みます。宣言部分は、以下の 2 つの主要なオプション・ブロックを使用します。

- DEFINE ブロック
- FUNCTION ブロック

宣言部分には、他の言語構成要素およびステートメント (EXEC ステートメント、IF ブロック、INCLUDE ステートメント、および MESSAGE ブロックなど) を含むこともできます。言語構成要素についての詳細は、*Net.Data* 解説書の言語構成要素に関する章を参照してください。

- 表示部分は、Web ページ・レイアウトの定義、変数の参照、および HTML ブロックをマクロの入り口および出口点に使用した関数の呼び出しを行います。Net.Data を起動する際に、HTML ブロック名をマクロ処理の入り口点に指定します。HTML ブロックは、56ページの『HTML ブロック』で説明されています。

本節では、簡単な Net.Data のマクロを使って、マクロ言語の要素を例示します。このマクロの例は、REXX プログラムに渡す情報をプロンプト指示するフォームを提供します。このマクロは、この情報を ompsamp.cmd と呼ばれる REXX プログラムに渡します。このプログラムは、ユーザーが入力したデータをそのまま返します。次に、この結果が、HTML の 2 ページに表示されます。

まず最初に、マクロ全体を見渡し、次に各ブロックを詳細に見てみましょう。

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

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

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

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

<FORM METHOD="post" ACTION="OUTPUT">
Type some data to pass to a REXX program:
<INPUT NAME="input_data" TYPE="text" SIZE="30">
<p>
<INPUT TYPE="submit" VALUE="Enter">

</form>

< hr>
```

```

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

%{ ***** HTML Block: Output *****%}
%HTML (OUTPUT) {
<html>
<head>
<title>$(page_title)</title>
</head><body>
<h1>Output Page</h1>
<p>@rex1(input_data)
<p><hr>
<p>[<a href="/">Home page</a> |
<a href="input">Previous page</a>]
</body></html>
%}

```

サンプルのマクロは、DEFINE、FUNCTION、および 2 つの HTML ブロック、という 4 つの主要なブロックで構成されます。1 つの Net.Data のマクロに、複数の DEFINE、FUNCTION、および HTML ブロックを持つことができます。

2 つの HTML ブロックは、HTML などのテキスト表示ステートメントを含みます。これにより Web マクロの作成が簡単になります。HTML について詳しくれば、マクロの作成は、単に、サーバーで動的に処理されるマクロ・ステートメントと、データベースに送信する SQL ステートメントの追加を行うだけになります。

マクロは HTML 文書に類似しているように見えますが、Web サーバーは、Net.Data から CGI、Web サーバー API、または Java サーブレットを使用してマクロにアクセスします。マクロを起動するには、Net.Data は、処理すべきマクロの名前、およびそのマクロの表示すべき HTML ブロックの 2 つのパラメーターが必要です。

マクロが呼び出されると、Net.Data はそのマクロ・ファイルを最初から処理していきます。以下の節では、Net.Data がマクロ・ファイルを処理していくとどうなるかを見てみます。

DEFINE ブロック

DEFINE ブロックには、後で HTML のブロックで使用する DEFINE 言語構成要素および変数の定義が含まれます。以下の例は、1 つの変数定義を持つ DEFINE ブロックを示しています。

```

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

```

1 行目はコメントです。コメントは、%{ と %} で囲まれた任意のテキストです。コメントは、マクロの任意の場所に置くことができます。その次のステートメントが、DEFINE ブロックの始まりです。1 つの定義ブロックに、複数の変数を定義することができます。この例では、page_title という 1 つの変数だけが定義されています。変数の定義を行うと、\$(page_title) という構文を使用して、この変数をマクロの任意の場所で参照することができます。変数を使用すると、後でマクロを全体にわたって変更することが容易にできるようになります。このブロックの最後の行 %} は、DEFINE ブロックの終了を識別します。

FUNCTION ブロック

FUNCTION ブロックには、HTML ブロックで呼び出される関数の関数宣言が含まれます。関数は、言語環境によって処理され、プログラム、SQL 照会、あるいは ストアード・プロシージャを実行することができます。

以下の例では、2 つの FUNCTION ブロックを示します。1 つは外部 REXX プログラムへの呼び出しを定義し、もう 1 つはインライン REXX ステートメントを含みます。

```
%{ ***** FUNCTION Block *****%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result) { <-- This function accepts
                                                         one parameter and returns the
                                                         variable 'result', which is
                                                         assigned by the external REXX
                                                         program
                                                         %EXEC{ompsamp.cmd %} <-- The function executes an external REXX program
                                                         called "ompsamp.cmd"
                                                         %}
%FUNCTION(DTW_REXX) today () RETURNS(result) {
    result = date() <-- The single source statement for this function is
                      contained inline.
%}
```

最初の関数ブロック rexx1 は、REXX 関数宣言です。この宣言は、次に ompsamp.cmd と呼ばれる外部の REXX プログラムを実行します。1 つの入力変数 input は、この関数によって受け取られ、自動的に外部の REXX コマンドに渡されます。REXX コマンドはまた、result と呼ばれる 1 つの変数を戻します。REXX コマンドにおける result 変数の内容は、OUTPUT ブロックに含まれる @rexx1() 関数呼び出しの起動に置き換わります。ompsamp.cmd のソース・コードに示されるように、REXX のプログラムにより、変数 input および result に直接アクセスすることができます。

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

この関数のコードは、その関数に渡されたデータをそのまま返します。この結果のテキストは自分の好きなようにフォーマットすることができます。それには、@rexx1() 関数呼び出し要求を、通常のマークアップ・スタイルのタグでくくります(たとえば または のように)。result 変数を使用しなくても、REXX のプログラムは、REXX の SAY ステートメントを使用すれば、HTML のタグを標準出力に書き込むことができたはずです。

2 番目の関数ブロックも、REXX プログラム today を参照しています。しかし、この場合の REXX プログラム全体は、関数宣言それ自身に含まれています。外部プログラムは必要ありません。インライン・プログラムは、REXX および Perl の関数では許可されています。その理由は、これらのプログラムは、動的な解析および実行ができるインタープリター言語だからです。インライン・プログラムは、管理すべき独立したプログラムの必要がないので、単純であるという優位点を持っています。最初の REXX の関数は、インラインとしてハンドルすることも可能です。

HTML ブロック

HTML ブロックは、Web ページ・レイアウトの定義、変数の参照、および関数の呼び出しを行います。HTML ブロックはマクロの入り口点および出口点に使用されま

す。HTML ブロックは常に Net.Data マクロ要求内に指定され、それぞれのマクロは少なくとも 1 つの HTML ブロックを持つ必要があります。

マクロの例にある最初の HTML ブロックには INPUT と名前が付けられています。HTML(INPUT) には、1 つの入力フィールドを持つ簡単なフォームの HTML が含まれます。

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

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

<p>
<INPUT TYPE="submit" VALUE="Enter">

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

ブロック全体は、HTMLのブロック識別子、%HTML (INPUT) {...}% で囲まれます。INPUT は、このブロックの名前を識別します。名前は、任意の英数字、アンダースコア、またはピリオドを含むことができます。HTML の <title> タグは、変数置換の例を含んでいます。変数 page_title の値が、フォームの表題に取って代わります。

このブロックはまた、関数呼び出しを持っています。式 @today() は、関数 today への呼び出しです。この関数は、上記の FUNCTION ブロックで定義されます。Net.Data は、today 関数の結果、すなわち現在日付を、@today() 式が配置されているのと同じ場所に挿入します。

FORM ステートメントの ACTION パラメーターは、HTML ブロック間あるいはマクロ間のナビゲーションの例を提供してくれています。ACTION パラメーターの別のブロックの名前を指定すると、フォームが処理依頼されたときに、そのブロックにアクセスします。HTML フォームからの入力データはどれも、暗黙の変数としてブロックに渡されます。これは、このフォーム上で定義される単一の入力フィールドにもあてはまります。フォームが処理依頼されると、このフォームに入力されたデータは、変数 input_data に入れられ、HTML(OUTPUT) ブロックに渡されます。

マクロが同じ Web サーバー上にある場合、相対参照を持つ他のマクロの HTML ブロックにアクセスすることができます。たとえば、ACTION パラメーター ACTION="../../othermacro.d2w/main" は、マクロ othermacro.d2w の main と呼ばれる HTML ブロックにアクセスすることができます。フォームに入力されたデータはどれも、もう一度変数 input_data に入れられて、このマクロに渡されます。

Net.Data を起動する場合、変数を URL の一部として渡します。たとえば、以下のようになります。

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

マクロのフォーム・データのアクセスまたは操作は、フォーム内に指定された変数名を参照して行います。

この例における次の HTML ブロックは、HTML(OUTPUT) ブロックです。これには、HTML のタグ付け、および HTML(INPUT) の要求により処理された出力を定義する Net.Data のマクロ・ステートメントが含まれます。

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

</head><body>
<h1>Output Page</h1>
<p>@rexx1(input_data) <--- This line contains a call to function rexx1
                        passing the argument "input_data".

<p>
< hr>
<p>
[
<a href="/">Home page</a> |
<a href="input">Previous page</a>]
%}
```

HTML(INPUT) ブロックと同じように、このブロックも、変数および関数呼び出しを置換するための Net.Data マクロ・ステートメントを持つ標準の HTML です。再度、page_title 変数はこの title ステートメントに置換されます。そして、前と同じように、このブロックは関数呼び出しを含みます。この場合、このブロックは、関数 rexx1 を呼び出し、変数 input_data の内容をこの関数に渡します。この変数は、このブロックが Input ブロックで定義されたフォームから受け取ったものです。任意の数の変数を関数に渡したり、関数から受け取ったりすることができます。関数定義は、渡される変数の数および使用法を指定します。

Net.Data のマクロ変数

Net.Data により、Net.Data でマクロの変数を定義したり、参照することができますさらに、これらの変数を、マクロから言語環境に渡したり、また言語環境から受け取ったりすることができます。変数名と値、およびリテラル・ストリングなどの Net.Data のトークンは、256 KB までのデータを含むことができます。

Net.Data 変数は、変数の型、および事前定義値をとるかどうかに応じて定義ができます。これらの変数は、その定義方法に基づき以下の型に分類できます。

- **DEFINE ブロックで DEFINE ステートメントを使用して、明示的に定義された変数。**
- **定義済みの変数。**これは、Net.Data により使用可能になり、ある値が設定されます。通常、この値は変更できません。
- **暗黙的に定義された変数。**これには、以下の 4 つのタイプがあります。
 - 明示的には定義されていないが、最初に値を割り当てる際にインスタンス化される変数。

- FUNCTION ブロック定義の一部で、FUNCTION ブロック内でしか参照できないパラメーター変数。
- Net.Data によりインスタンス化され、フォーム・データまたは照会ストリング・データに対応する変数。
- Net.Data の表と関連付けられ、ROW ブロックあるいは REPORT ブロック内でしか参照することができない変数。

次のセクションでは、以下を説明します。

- 『識別子の効力範囲』
- 60ページの『変数の定義』
- 62ページの『変数の参照』
- 62ページの『変数の型』

識別子の効力範囲

識別子は、変数あるいは関数呼び出しで、可視 になります。すなわち、識別子が宣言されたり初期化されたときに参照することができる、という意味です。識別子が可視になっている領域は、その効力範囲 と呼ばれます。効力範囲には、次の 5 つのタイプがあります。

- グローバル

識別子は、マクロ内の任意の場所で参照できる場合、グローバルな効力範囲を持ちます。グローバルな効力範囲を持つ識別子には、以下のようなものがあります。

- Net.Data の組み込み関数
- フォームのデータ
- 照会ストリング・データ
- HTML ブロック内からインスタンス化された変数

- マクロ

識別子は、その宣言がブロックの外に現れる場合、グローバルな効力範囲を持ちます。ブロックは、左大括弧 ({) で始まり、パーセント記号と大括弧 (% }) で終わります。(DEFINE ブロックは、この定義から除外され、独立した DEFINE ステートメントとして扱わなければなりません。) グローバルな効力範囲を持つ識別子と異なり、マクロ効力範囲を持つ識別子への参照は、識別子の宣言以降のマクロ内の項目によってのみ行えます。

- FUNCTION ブロックまたは MACRO_FUNCTION ブロック

識別子は、以下の場合に関数ブロックの効力範囲を持ちます。

- 識別子に関数定義のパラメーター・リスト内で宣言された場合。
同じ名前を持つ識別子が、関数定義の外に存在する場合は、Net.Data は、その関数ブロック内の関数仮引き数リストの識別子を使用します。
- 識別子は関数ブロック内でインスタンス化され、関数呼び出しの前において宣言またはインスタンス化されることはありません。

識別子は、それが関数の外で宣言されたり、初期化されている場合、および関数のパラメーター・リストで宣言されてもいない場合は、関数ブロックの効力範囲を持ちません。関数ブロック内の識別子の値は、関数により更新されない限りは、変更されることはありません。

- **REPORT ブロック**

識別子は、それが **REPORT** ブロック内からだけしか参照できない場合は、レポート・ブロックの効力範囲を持ちます (たとえば、表の列の名前 *N1*、*N2*、...、*Nn*)。Net.Data が、その表処理の一部として暗黙的に定義する変数だけが、レポート・ブロックの効力範囲を持つことができます。インスタンス化されるそれ以外の変数はどれも、関数ブロックの効力範囲を持ちます。

- **ROW ブロック**

識別子は、それが **ROW** ブロック内からしか参照できない場合、行ブロックの効力範囲を持ちます (たとえば、表値の名前 *V1*、*V2*、...、*Vn*)。Net.Data が、その表処理の一部として暗黙的に定義する変数だけが、行ブロックの効力範囲を持つことができます。インスタンス化されるそれ以外の変数はどれも、関数ブロックの効力範囲を持ちます。

変数の定義

Net.Data マクロにおける変数の定義方法には、以下の 3 つの方法があります。

- **Define** ステートメントまたはブロック
- **HTML** フォーム・タグ
- 照会ストリング・データ

フォームまたは照会ストリング・データから受け取った変数の値は、Net.Data マクロにおいて **DEFINE** ステートメントにより設定された変数の値を上書きします。

- **DEFINE** ステートメントまたはブロック

Net.Data のマクロで使用するための最も簡単な変数定義の方法は、以下のように **DEFINE** ステートメントを使うことです。構文は以下のようになります。

```
%DEFINE variable_name="variable value"

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

%DEFINE {
    variable_name1="variable value 1"
    variable_name2="variable value 2"
}%
```

variable_name は、ユーザーが変数に与える名前です。変数名は、文字またはアンダースコアで始めなければなりません。そして、任意の英数字、アンダースコア、またはピリオドを含むことができます。すべての変数名は、大文字小文字の区別をします。ただし、表変数の、*N_columnName* および *V_columnName* は除きます。

たとえば、以下のようにします。

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

変数 *reply* は、値 *hello* をとります。

連続する引用符が 2 つだけの場合は、空ストリングに等しくなります。たとえば、以下のようにします。

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

変数 `empty` は、空ストリングをとります。

変数に行の終わりなどの特殊文字が含まれる場合、値を中括弧で囲みます。

```
%DEFINE introduction={  
Hello,  
My name is John.  
%}
```

ストリングに引用符を組み込むためには、2 つの引用符を続けて使用します。

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

ブロック中括弧を使用することにより、引用が拡張できます。

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

幾つかの変数を、`DEFINE` ステートメントを使用して定義するには、`DEFINE` ブロックを以下のように使用します。

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

• HTML フォーム・タグ: `SELECT`、`INPUT`、および `TEXTAREA`

HTML FORM タグを使用して変数に値を割り当てることができます。これには `SELECT`、`INPUT`、および `TEXTAREA` タグがあります。以下の例では、標準の HTML フォーム・タグを使用して、`Net.Data` 変数を定義しています。

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

あるいは

```
<SELECT NAME="variable_name">  
    <OPTION>value one  
    <OPTION>value two  
</SELECT>
```

複数行におよぶ変数、または引用符などの特殊文字を含む変数を割り当てるには、`TEXTAREA` タグを使用します。

```
<TEXTAREA NAME="variable_name" ROWS="4">  
Please type the multiline value  
of your variable here.  
</TEXTAREA>
```

`variable_name` は、その変数に与えた名前です。そして、その変数の値は、フォームで受け取った入力から決定されます。`Net.Data` のマクロにおける、このような変数定義のタイプの使用方法の例については、43ページの『HTML フォーム』を参照してください。

• 照会ストリング・データ

照会ストリングにより `Net.Data` に変数を渡すことができます。たとえば、以下のようになります。

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

上の例では、変数名 `field` とその変数値 `custno` は、`Net.Data` が 照会ストリングから受け取った追加データを指定します。`Net.Data` は、そのデータをフォーム・データからのデータと同様に受け取り処理します。

変数の参照

定義済みの変数を参照して、その値を戻すことができます。`Net.Data` のマクロにおいて変数を参照するには、その変数名を `$(および)` 内に指定します。たとえば、以下のようにします。

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

`Net.Data` が変数参照を検出すると、`Net.Data` は、その変数参照を変数の値で置き換えます。

変数をテキスト表示ステートメントの一部として使用するには、マクロの `HTML` ブロックでそれらの変数を参照します。

例 1: リンクにおける変数参照

変数 `homeURL` を定義した場合:

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

ホーム・ページは `$(homeURL)` として参照でき、以下のようにリンクを作成します。

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

変数の参照は `Net.Data` マクロの多くの部分で行うことができます。本章における言語構成要素をチェックして、マクロのどの部分で変数参照が使用できるか決定してください。変数とその参照時に未定義である場合、`Net.Data` は空ストリングを戻します。変数参照は単独では変数を定義しません。

変数の型

マクロでは以下の型の変数が使用できます。

- 63ページの『条件変数』
- 63ページの『環境変数』
- 64ページの『実行可能な変数』
- 65ページの『隠し変数』
- 66ページの『リスト変数』
- 67ページの『表変数』
- 67ページの『各種変数』
- 68ページの『表処理変数』
- 68ページの『レポート変数』
- 69ページの『言語環境変数』

ENVVAR、LIST、条件リスト変数など、Net.Data の手順により定義された変数に文字列を割り当てると、変数は定義どおりの振る舞いをしなくなります。つまり、変数はストリングを含む単純変数となります。

各変数の構文および例については、*Net.Data* 解説書 を参照してください。

条件変数

条件変数を使用すると、IF、THEN 構成要素と類似の方法を使用して、変数に対して条件値を定義することができます。条件変数を定義する場合は、変数を取ることができる 2 つの値を指定することができます。参照する最初の変数が存在する場合は、条件変数は最初の値を取得します。そうでない場合は、条件変数は 2 番目の値を取得します。条件変数の構文は、次のようになります。

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

varB が定義される場合は、varA="value_1"、そうでない場合は、varA="value_2" となります。これは、次の例のように、IF ブロックを使用するのと同じこととなります。

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

条件変数をリスト変数と一緒に使用する例については、66ページの『リスト変数』を参照してください。

環境変数

Net.Data 要求の処理プロセスまたはスレッドに対し Web サーバーが使用可能とする環境変数を参照することができます。ENVVAR 変数を参照する場合、Net.Data は環境変数の現行値を同じ名前で戻します。

環境変数を定義する構文は以下のとおりです。

```
%DEFINE var=%ENVVAR
```

var は、定義される環境変数の名前です。

たとえば、変数は SERVER_NAME は、以下のようにして環境変数として定義することができます。

```
%DEFINE SERVER_NAME=%ENVVAR
```

そして、参照は以下のように行います。

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

出力は以下のように見えます。

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

ENVVAR ステートメントについての詳細は、"*Net.Data* 解説書" を参照してください。

実行可能な変数

実行可能な変数を使用すると、変数参照から他のプログラムを呼び出すことができます。

DEFINE ブロックのEXEC 言語構成要素を使用して、Net.Data のマクロに実行可能な変数を定義します。EXEC 言語要素のさらに詳しい情報については、"*Net.Data* 解説書"の、言語構成要素の章を参照してください。以下の例では、変数 `runit` が定義され、実行可能なプログラム `testProg` が実行されます。

```
%DEFINE runit=%EXEC "testProg"
```

`runit` は、実行可能な変数になります。

Net.Data は、Net.Data のマクロ内で、有効な変数参照に出会うと、実行可能なプログラムを実行します。たとえば、有効な変数参照が、Net.Data のマクロの変数 `runit` に対して行われると、プログラム `testProg` が実行されます。

簡単な方法は、別の変数定義から、実行可能な変数を参照することです。以下の例は、この方法の一例を示しています。変数 `date` は、実行可能な変数として定義され、`dateRpt` は、実行可能な変数への参照が含まれます。

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

`$(dateRpt)` が Net.Data のマクロに現れるごとに、Net.Data は、実行可能なプログラム `date` を検索し、それを見つけると、戻ります。

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

Net.Data がマクロ内で実行可能な変数に出会うと、Net.Data は、以下の方法によって、参照されている実行可能なプログラムを探します。

1. Net.Data 初期設定ファイルの EXEC_PATH で指定されたディレクトリを検索します。詳細については、17ページの『EXEC_PATH』を参照してください。
2. Net.Data がプログラムを見つけなかった場合、システムは、システムの PATH 環境変数、またはライブラリー・リストで定義されているディレクトリを検索します。実行可能なプログラムが見つければ、Net.Data は、そのプログラムを実行します。

制約事項： 実行可能な変数を、それが呼び出す実行可能なプログラムの出力値に設定しないでください。前の例では、変数 `date` の値は NULL です。DTW_ASSIGN 関数呼び出しでこの変数を使用し、その値を別の変数に割り当てると、割り当て後の新規の変数の値も NULL になります。実行可能な変数の目的はただ 1 つ、その変数が定義するプログラムを呼び出すことです。

実行するプログラムにパラメーターを渡すことができます。それには、変数定義の際に、そのプログラム名と一緒にそのパラメーターを指定します。次の例では、距離と時間の値が、プログラム `calcMPH` に渡されています。

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

この次の例では、システムの日付をレポートの一部として戻します。

```
%DEFINE tstamp=%EXEC "date"

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

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

各レポートは、追跡がしやすくなるように日付を表示します。この例ではまた、顧客番号と名前を、別の Net.Data のマクロのリンクに書き込んでいます。レポートの任意の顧客をクリックすると、exmp.d2w という Net.Data のマクロが呼び出され、顧客番号と名前を Net.Data のマクロに渡します。

隠し変数

隠し変数を使用すると、変数の実際の名前を、Web ブラウザーを使用して Web のソースを見ているアプリケーション・ユーザーから隠すことができます。隠し変数を定義するには、以下のようにします。

1. HTML ブロックにおける最後の変数参照の後に、隠したいストリングごとに変数を指定する。変数は、HTML ブロックで使用された後、以下の例のように、常に DEFINE 言語要素を使用して定義されます。\$(variable) 変数が参照され、次に定義されます。
2. 変数が参照されている HTML ブロックで、1 つのドル記号ではなく、2 つのドル記号を使用して、変数を参照します。たとえば、\$(X)ではなく、\$(X) とします。

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

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

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

...
```

Web ブラウザーが HTML のフォームを表示すると、\$(name) および \$(addr) は、\$(name) および \$(addr) にそれぞれ置き換えられます。その結果、実際の表と列名は、HTML のフォームに表示されることはありません。アプリケーションのユーザーは、真の変数名が隠されているのかどうかを知ることができません。

ユーザーがフォームを処理依頼すると、HTML(REPORT) ブロックが呼び出されます。 @mySelect() が FUNCTION ブロックを呼び出すと、\$(Field) は、SQL ステートメントにおいて、SQL 照会の customer.name あるいは customer.addr で置き換えられます。

リスト変数

リスト変数を使用して、値が区切られたストリングを作成します。リスト変数は、WHERE あるいは HAVING 節に見られるような複数項目を持つ SQL 照会を構成するのに特に役に立ちます。リスト変数の構文は、次のようになります。

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

推奨：ブランクは重要です。ほとんどの場合、値の区切り文字の前後にスペースを挿入します。ほとんどの照会は、値の区切り文字に、ブールあるいは数学演算子を使用します (たとえば、AND、OR、あるいは >)。次の例は、条件、隠し、およびリスト変数の使用例を示したものです。

```
%HTML(INPUT){
<FORM METHOD="POST" ACTION="/netdata-cgi/db2www/example2.d2w/report">
<H2>Select one or more cities:</H2>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$(cond1)">Sao Paolo<BR>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$(cond2)">Seattle<BR>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$(cond3)">Shanghai<BR>
<INPUT TYPE="submit" VALUE="Submit Query">
</FORM>
%}

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

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

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

HTML のフォームでは、ボックスがチェックされていない場合、conditions は NULL となり、照会では whereClause もまた NULL になります。そうでない場合は、whereClause は、OR で区切られた選択値を持ちます。たとえば、3 都市がすべて選択される場合は、SQL は以下ようになります。

```
SELECT name, city FROM citylist
WHERE cond1='Sao Paolo' OR cond2='Seattle' OR cond3='Shanghai'
```

次の例は、Seattle が選択されると、その結果、次のような SQL 照会になることを示しています、

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


表変数

表変数は、関係しあうデータの集合を定義します。これには、列見出しの行を含めた行および列のセットが含まれます。表は、Net.Data のマクロでは、以下のステートメントのようにして定義されます。

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

%TABLE の後続く数字は、この表変数が含むことができる行数の制限です。行数に制限のない表を指定するには、次の例のように、デフォルトを使用するか、ALL を指定します。

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

表を定義すると、表はゼロの行とゼロの列をとります。表に値を代入するには、表を OUT または INOUT パラメーターとして関数に渡すか、Net.Data が提供する組み込みの表関数を使用する以外には方法がありません。DTW_SQL 言語環境は、SELECT ステートメントの結果を表に自動的に書き込みます。

DTW_REXX または DTW_PERL など、非データベース言語環境の場合、言語環境もまた表の値を設定する責任を負います。ただし、言語環境スクリプトまたはプログラムは、表の値をセルごとに定義します。言語環境による表変数の使用方法についての詳細は、89ページの『第6章 言語環境の使用』を参照してください。

表変数の名前を参照することにより、関数間で表を渡すことができます。表の個々の要素は、関数の REPORT ブロックで参照することができます。詳細については、68ページの『表処理変数』を参照してください。表変数は、通常、SQL 関数の値が代入され、次に、SQL 関数あるいは別の関数のいずれかにパラメーターとして渡された後、その関数におけるレポートへの入力として使用されます。表変数を、IN、OUT、あるいは INOUT パラメーターとして、任意の非 SQL 関数に渡すことができます。表は、SQL関数には、OUT パラメーターとしてのみ渡すことができます。

各種変数

これらの変数は、Net.Data で定義された変数で、以下の目的のために使用することができます。

- Net.Data の処理に影響を与える
- 関数呼び出しの状態を検出する
- データベース照会の結果セットに関する情報を取得する
- ファイル場所と日付に関する情報を決定する

各種変数は、Net.Data が決定する定義済みの値、あるいはユーザーが設定する値を持つことができます。たとえば、Net.Data は、Net.Data が処理中の現行ファイルに基づいた DTW_CURRENT_FILENAME 変数の値を決定します。これに対して、Net.Data は、タブや改行文字で作られた余分なスペースを削除するかどうかを指定することができます。

定義済みの変数は、マクロ内では変数参照として使用され、ファイルの現在の状態、日付、あるいは関数呼び出しの状態に関する情報を提供します。たとえば、現行ファイルの名前を検索するには、以下を使用することができます。

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

変更可能な変数値は、一般には、**DEFINE** ステートメント、あるいは **@DTW_ASSIGN()** 関数を使用して設定され、Net.Data のマクロの処理方法に影響を与えます。たとえば、空白文字を削除するかどうかを指定するには、以下の **DEFINE** ステートメントを使用することができます。

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

表処理変数

Net.Data は、**REPORT** および **ROW** ブロックで使用するための表処理変数を定義します。これらの変数を使用して、**SQL** 照会および関数呼び出しからの値を参照します。

表処理変数は、Net.Data が決定する事前定義値をとります。これらの変数により、**SQL** 照会または関数呼び出しの結果セットからの値を、処理中の列、行、またはフィールドで参照することができます。また、処理されている行の数に関する情報、あるいはすべての列名のリストにアクセスすることができます。

たとえば、Net.Data は、**SQL** 照会からの結果セットを処理しながら、現行の列名ごとに、変数 **Nn** の値を、**N1** を最初の列に、**N2** を 2 番目の列に、というようにして割り当てます。Web ページ出力の現行列名を参照することができます。

表処理変数を、変数参照としてマクロ内で使用します。たとえば、処理中の現行列の名前を検索するには、以下を使用することができます。

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

表処理変数はまた、照会の結果に関する情報を提供します。マクロにおいて変数 **TOTAL_ROWS** を参照して、以下の例のように、**SQL** 参照からどれだけの行が戻されたかを表示することができます。

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

表処理変数の中には、他の変数あるいは組み込み関数により影響されるものがあります。たとえば、**TOTAL_ROWS** は、**DTW_SET_TOTAL_ROWS** という **SQL** の言語環境変数をアクティブにして、以下の例のように、**SQL** 照会あるいは関数呼び出しからの結果を処理するときに、Net.Data が **TOTAL_ROWS** の値を割り当てるように、要求します。

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

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

レポート変数

Net.Data は、マクロにより生成された Web ページ出力を、デフォルトのレポート・フォーマットで表示します。デフォルトのレポート・フォーマットは、**<PRE>** **</PRE>** タグを使用した表形式で表示をします。出力の表示命令を持つ **REPORT** ブロックを定義、またはデフォルトのレポート生成を行わないようにするレポート変数を使用することにより、デフォルトのレポートがオーバーライドできます。

レポート変数は、デフォルトのレポートおよび Net.Data テーブルによる Web ページ出力の表示および使用方法をカスタマイズするのに役に立ちます。レポート変数は、使用する前に、DEFINE ステートメント、あるいは @DTW_ASSIGN() で定義されなければなりません。

レポート変数は、スペーシングを指定し、デフォルトのレポート・フォーマットを上書きし、デフォルトの表出力に対する HTML の表出力、および他の表示機能を指定します。たとえば、ALIGN 変数を使用すると、表処理変数に対して前後のスペースを制御することができます。以下の例では、ALIGN 変数を使用して、照会により戻されるリストの各列名をスペースで区切ります。

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

START_ROW_NUM レポート変数により、どの行で、照会の結果の表示を始めるかを決定することができます。たとえば、以下の変数は、Net.Data が、照会の結果の表示を 3 番目の行から始めるように指定しています。

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

また、Net.Data がデフォルトのフォーマットに HTML のタグを使用するかどうかを決定することもできます。DTW_HTML_TABLE を YES に設定すると、テキスト・フォーマットの表ではなく、HTML の表が作成されます。

```
%DEFINE DTW_HTML_TABLE="YES"

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

言語環境変数

言語環境変数は、言語環境と共に使用され、言語環境による要求処理の方法に影響を与えます。

これらの変数を使用することにより、DB2 サブシステムへの接続の確立、NLS サポートの使用可能化、および SQL ステートメントが正常に実行されたか判別するなどのタスクが実行できます。

たとえば、SQL_STATE 変数を使用すれば、データベースから戻される SQL の状態値にアクセスしたり、表示することができます。

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

Net.Data の関数

Net.Data には、ユーザーのアプリケーションで使用するための組み込み関数が提供されています。これにはワードおよびストリング処理関数、または表変数関数の検索および設定をする関数があります。また、たとえば外部プログラムやストアード・プロシージャを呼び出すため、アプリケーションで使用する関数を定義することができます。

ユーザー定義の関数

たとえば、外部プログラムやストアード・プロシージャを呼び出すため、アプリケーションで使用するために定義する関数。

Net.Data の組み込み関数

ユーザーのアプリケーションで使用するため Net.Data が提供する関数。ワードおよびストリングを操作する関数、および表変数の設定をする関数などがあります。

次のセクションでは、以下のトピックについて説明します。

- 『関数の定義』
- 75ページの『関数の呼び出し』
- 75ページの『Net.Data 組み込み関数の呼び出し』

関数の定義

ユーザー自身の関数をマクロに定義するには、FUNCTION ブロック、または MACRO_FUNCTION ブロックを使用します。

FUNCTION ブロック

Net.Data マクロから呼び出され、言語環境で処理されるサブルーチンを定義します。FUNCTION ブロックには、言語ステートメント、または外部プログラムの呼び出しが含まれなくてはなりません。

MACRO_FUNCTION ブロック

Net.Data マクロから呼び出され、言語環境ではなく Net.Data により処理されるサブルーチンを定義します。MACRO_FUNCTION ブロックには、HTML ブロックで使用可能ないずれのステートメントも含むことができます。

構文: 関数の定義には以下の構文を使用します。

FUNCTION ブロック:

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

MACRO_FUNCTION ブロック :

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

ここで、

type 初期設定ファイルにおいて構成されている言語環境を識別します。言語環境は、特定の言語プロセッサ（これは、実行可能なステートメントを処理します）および *Net.Data* と言語プロセッサとの標準インターフェースを提供します。

function-name

FUNCTION または MACRO_FUNCTION ブロックの名前を指定します。関数呼び出しは、@ 記号を前に付けた *function-name* を指定します。詳細については、75ページの『関数の呼び出し』を参照してください。

複数の FUNCTION または MACRO_FUNCTION ブロックを同じ名前で定義し、それらの同時に処理することができます。各ブロックは、すべて、同じパラメーター・リストを持たなければなりません。*Net.Data* が関数を呼び出すと、同じ名前を持つすべての FUNCTION ブロック、または同じ名前を持つ MACRO_FUNCTION ブロックは、*Net.Data* のマクロにおける定義順に実行されます。

usage パラメーターが、入力 (IN) パラメーターか、出力 (OUT) パラメーターか、それとも両方のタイプ (INOUT) かを指定します。この指定は、FUNCTION ブロックまたは MACRO_FUNCTION (あるいはその両方) にパラメーターが渡されるのか、またはそこから受け取られるのかを示しています。*usage* のタイプは、別の *usage* のタイプにより変更されるまで、パラメーター・リストのその後に続くパラメーターすべてに適用されます。デフォルトのタイプは IN です。

datatype

ストアド・プロシージャを呼び出すためのパラメーターのデータ型。ストアド・プロシージャを呼び出す場合、パラメーターのデータ型は、ストアド・プロシージャの対応するパラメーターのデータ型に一致する必要があります。ストアド・プロシージャでサポートされるデータ型のリストについては、*Net.Data* 解説書のオペレーティング・システムの付録を参照してください。

parameter

関数呼び出し時に指定される対応する引き数の値で置き換えられる、ローカルな効力範囲を持つ変数の名前。実行可能ステートメント、あるいは REPORT ブロック内の、たとえば \$(*parm1*) というパラメーター参照は、そのパラメーターの実際の値で置き換えられます。さらに、パラメーターは、言語環境に渡されて、その言語の自然構文を使用する実行可能ステートメントから、あるいは環境変数として、アクセス可能となります。パラメーター変数変数の参照は、FUNCTION または MACRO_FUNCTION ブロックの外では無効です。

return-var

このパラメーターを、RETURNS キーワードの後に指定して、特殊な OUT パラメーターを識別します。戻り変数の値は関数ブロックに割り当てられており、その値は、マクロ内で関数が呼び出された位置に戻されます。たとえば次の文の <p>My name is @my_name(). において、@my_name() は戻り変数の値で置き換えられます。RETURNS 節を指定しなければ、関数呼び出しの値は以下ようになります。

- 呼び出しから言語環境への戻りコードがゼロの場合、NULL。
- 戻りコードがゼロ以外の場合、戻りコードの値。

executable-statements

変数の置換および関数処理の後で、処理のための指定された言語環境に渡される言語ステートメントのセット。*executable-statements* には、Net.Data 変数参照および Net.Data 関数呼び出しを含むことができます。

FUNCTION ブロックの場合、Net.Data は、すべての変数参照を変数の値に置き換え、すべての関数呼び出しを実行し、関数呼び出しをその結果値に置き換えます。その後で、実行可能なステートメントが言語環境に渡されます。各言語環境は、ステートメントを異なる方法で処理します。実行可能なステートメントあるいは実行可能なプログラムの呼び出しについての詳細は、64ページの『実行可能な変数』を参照してください。

MACRO_FUNCTION ブロックの場合は、実行可能なステートメントは、テキストと Net.Data マクロ言語構成要素との組み合わせです。この場合、言語環境は全く関与しません。Net.Data が言語プロセッサとして働き、実行可能なステートメントを実行するためです。

report-block

FUNCTIONの出力処理のため 1 つ以上の REPORT ブロックを定義します。80ページの『レポート・ブロック』を参照してください。(複数のレポート・ブロックは、SQL および ODBC 言語環境内で FUNCTION ブロックにおいてのみ使用可能で、MACRO_FUNCTION ブロックでは使用できません。)

message-block

MESSAGE ブロックを定義します。このブロックは、FUNCTION ブロックによって戻された任意のメッセージをハンドルします。73ページの『メッセージ・ブロック』を参照してください。

最外部のブロックにあり、Net.Data マクロで呼び出される前の関数を定義します。

関数における特殊文字の使用

関数ブロックの言語ステートメント・セクション中、Net.Data 言語構成要素の構文に一致した文字を構文的に有効な組み込みプログラム・コード (REXX または Perl など) の部分に使用した場合、これは Net.Data の言語構成要素として誤って解釈され、マクロ中、エラーまたは予期せぬ結果の原因となります。

たとえば、Perl 関数では COMMENT ブロックの区切り文字に %{ を使用することがあります。マクロを実行すると、%{ 文字は COMMENT ブロックの開始と解釈されます。すると、Net.Data は COMMENT ブロックの終了を検索します。Net.Data は関数ブロックの終了まで読み込むと見つかるものと判断します。Net.Data は関数ブロックの終了まで検索を進め、検出ができないとエラーを出します。

以下のいずれかの方法で COMMENT ブロックの区切り文字を使用する、または他の Net.Data 特殊文字を組み込みプログラム・コードの部分に使用することにより、Net.Data がこれを特殊文字と解釈しないようにできます。

- コードをインラインに書き込まずに、EXEC ステートメントを使用してプログラム・コードを呼び出します。

- 変数参照を使用して特殊文字を指定します。

たとえば、以下の Perl 関数には COMMENT ブロックの区切り文字 `%{` が Perl 言語ステートメントの部分として含まれています。

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

Net.Data が `%{` 文字を Net.Data の COMMENT ブロック区切り文字ではなく Perl ソース・コードとして解釈するように、以下のいずれかのように関数を再度記述します。

- %EXEC ステートメントを使用:

```
%FUNCTION(DTW_PERL) func() {
    %EXEC{ func.pr1 %}
}%}
```

- 変数参照を使用して `%{` 文字を指定する:

```
%define percent_openbrace = "%{"

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

メッセージ・ブロック

MESSAGE ブロックにより、関数呼び出しの成功あるいは失敗を基にして、関数呼び出し後の進め方を決定することができ、関数の呼び出し側に情報を表示することができます。メッセージを処理する際、Net.Data は、言語環境変数 RETURN_CODE を FUNCTION ブロックへの関数呼び出しごとに設定します。RETURN_CODE は、関数呼び出し時には、MACRO_FUNCTION ブロックには設定されません。

MESSAGE ブロックは、連続したメッセージ・ステートメントで構成され、各メッセージ・ステートメントは、戻りコード値、メッセージ・テキスト、および取るべきアクションを指定します。MESSAGE ブロックの構文は、Net.Data 解説書の言語構成要素の章に示されています。

MESSAGE ブロックは、グローバルあるいはローカルな効力範囲を持つことができます。MESSAGE ブロックが、FUNCTION ブロックで定義されている場合は、その効力範囲は、その FUNCTION ブロックに対してはローカルです。MESSAGE ブロックが、最外部のマクロ・レイヤーで指定されている場合は、MESSAGE ブロックは、グローバルな効力範囲を持ち、Net.Data のマクロで実行されるすべての関数呼び出しに対してアクティブになります。2 つ以上のグローバルな MESSAGE ブロックを定義している場合は、最後に定義されたブロックがアクティブになります。

Net.Data は、以下のルールを使用し、関数呼び出しからの RETURN_CODE 変数の値を処理します。

1. ローカルな MESSAGE ブロックを 完全一致で検査する。指定に応じて、抜け出るか、続行します。
2. RETURN_CODE が 0 でない場合は、ローカルな MESSAGE ブロックを +default あるいは -default で検査する。RETURN_CODE の符号に依存し、指定に応じて、抜け出るか、続行します。
3. RETURN_CODE が 0 でない場合、ローカルな MESSAGE ブロックを、default で検査する。指定に応じて、抜け出るか、続行します。
4. グローバルな MESSAGE ブロックを 完全一致で検査する。指定に応じて、抜け出るか、続行します。
5. RETURN_CODE が 0 でない場合は、グローバルな MESSAGE ブロックを、+default あるいは -default で検査する。これは、RETURN_CODE の符号に依存し、指定に応じて、抜け出るか、続行します。
6. RETURN_CODE が 0 でない場合、グローバルな MESSAGE ブロックを、default で検査する。指定に応じて、抜け出るか、続行します。
7. RETURN_CODE が 0 でない場合、Net.Data の内部デフォルト・メッセージを発行し、抜け出ます。

以下の例は、グローバルな MESSAGE ブロックと、関数の MESSAGE ブロックを持つ Net.Data のマクロの部分を表示しています。

```
%{ global message block %}
%MESSAGE {
    -100      : "Return code -100 message"    : exit
    100       : "Return code 100 message"     : continue
    +default : {
This is a long message that spans more
than one line. You can use HTML tags, including
links and forms, in this message. %}    : continue
%}

%{ local message block inside a FUNCTION block %}
%FUNCTION(DTW_REXX) my_function() {
    %EXEC { my_command.cmd %}
%MESSAGE {
    -100      : "Return code -100 message"    : exit
    100       : "Return code 100 message"     : continue
    -default : {
This is a long message that spans more
than one line. You can use HTML tags, including
links and forms, in this message. %}    : exit
%}
}
```

my_function() が RETURN_CODE 値 50 で戻れば、Net.Data は次の順にエラーを処理します。

1. ローカルな MESSAGE ブロックを、完全一致で検査する。
2. ローカルな MESSAGE ブロックを +default で検査する。
3. ローカルな MESSAGE ブロックを default で検査する。
4. グローバルな MESSAGE ブロックを、完全一致で検査する。
5. グローバルな MESSAGE ブロックを +default で検査する。

Net.Data が一致を検出した場合、Net.Data はメッセージ・テキストを Web ブラウザーに送信し、要求されたアクションを検査します。

continue を指定した場合は、Net.Data は、メッセージ・テキストをプリントしてから、Net.Data マクロの処理を継続します。たとえば、マクロが my_functions() を 5 回呼び出し、エラー 100 が、上の例の MESSAGE ブロックの処理中に検出された場合、プログラムからの出力は、次のようになります。

```
.
.
.
11 May 1997                $245.45
13 May 1997                $623.23
19 May 1997                $ 83.02
return code 100 message
22 May 1997                $ 42.67

Total:                    $994.37
```

関数の呼び出し

Net.Data 関数呼び出しを使用することにより、ユーザー定義関数および組み込み関数の両方が呼び出せます。@ 文字に続けて関数名、またはマクロ関数名を使用します。

@function_name([argument,...])

function_name

起動する関数またはマクロ関数の名前です。関数は、それが組み込み関数でなければ、Net.Data のマクロであらかじめ定義されていなければなりません。

argument

これは、変数、引用符付き文字列、変数参照、または関数呼び出しの名前です。関数呼び出し時の引き数は、関数またはマクロ関数仮引き数リストのパラメーターにより突き合わせられます。各パラメーターには、関数またはマクロ関数の処理中に、対応する引き数の値が割り当てられます。引き数は、対応するパラメーターと同じ数および型でなければなりません。

Net.Data 組み込み関数の呼び出し

Net.Data は、Web ページ開発を容易にするための大きな組み込み関数のセットを提供します。これらの関数は、すでに Net.Data により定義されているので、定義する必要はありません。他の関数と同様に呼び出すことができます。

76ページの図5 に、Net.Data 組み込み関数およびマクロがどのように相互作用しているかを示します。

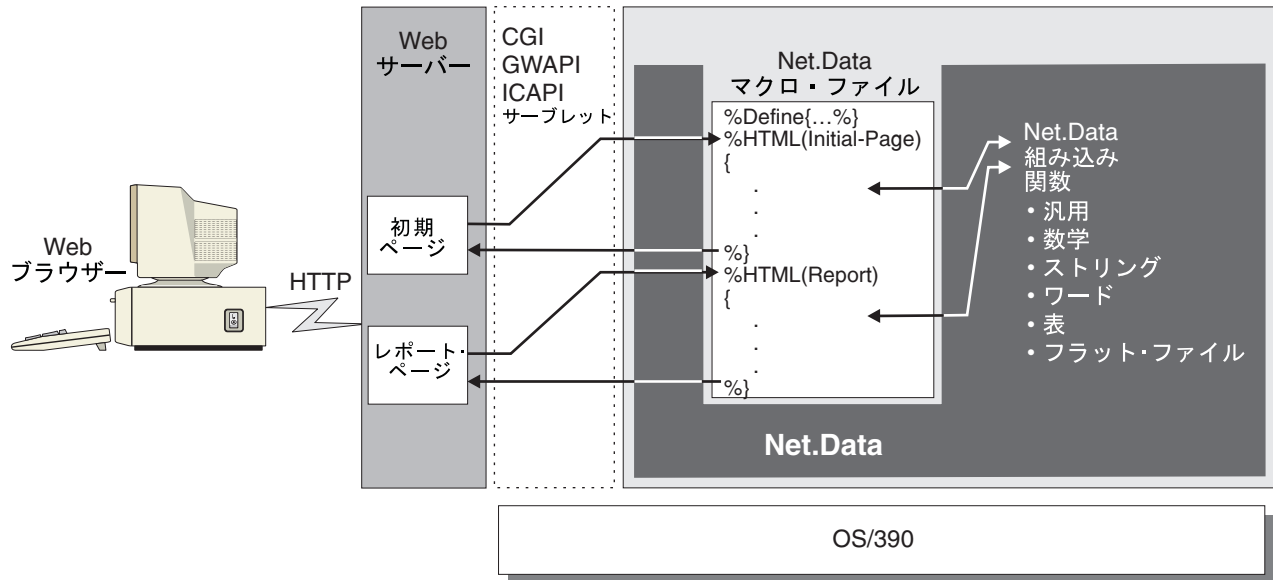


図 5. Net.Data の組み込み関数

組み込み関数は、その接頭部に応じて 3 の方法でその結果を戻すことができます。

- **DTW_ and DTWF_:** 呼び出し結果は、出力パラメーターで戻されます。あるいは、結果は戻されません。(DTWF_ は、フラット・ファイル関数に対応する接頭部です。)
- **DTW_r および DTWF_r:** マクロにおける関数呼び出しは、関数呼び出しの結果に置き換えられます。その方法は、RETURNS キーワードを指定したユーザー関数の関数呼び出しが、RETURNS キーワードの値で置き換えられるのと同じです。
- **DTW_m:** 複数の結果が、関数に渡されたそれぞれのパラメーターで戻されます。

組み込み関数によっては、タイプを持たないものがあります。特定の組み込み関数の型を判別するには、Net.Data 解説書の Net.Data 組み込み関数の章を参照してください。

以下の節では、Net.Data 組み込み関数について高度な概説を行います。これらの関数を使用して、汎用、数学、ストリング、ワード、または表操作の各関数を実行します。各関数の構文および例の説明については、"Net.Data 解説書" を参照してください。これらの関数には、使用する前に変数を設定する必要があるもの、または特定のコンテキストで使用しなくてはならないものがあります。

- 77ページの『汎用関数』
- 77ページの『数学関数』
- 77ページの『ストリング関数』
- 78ページの『ワード関数』
- 78ページの『表関数』
- 78ページの『フラット・ファイル関数』

汎用関数

この関数セットは、データを変更したり、システム・サービスを利用することにより、Web ページの開発に役に立ちます。これらの関数を使用して、照会、環境変数の設定、HTML のエスケープ・コードの使用、およびシステムからの有益な情報の取得、を行うことができます。

たとえば、特定の条件が発生した場合に Net.Data がマクロの残り部分を処理せず終了するように指定するには、DTW_EXIT 関数を使用します。

```
%HTML(cache_example) {  
  
<html>  
<head>  
  <title>This is the page title</title>  
</head>  
<body>  
  <center>  
    <h3>This is the Main Heading</h3>  
    <!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>  
    <! Joe Smith sees a very short page                               !>  
    <!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>  
    %IF (customer == "Joe Smith")  
  </body>  
</html>  
  
@DTW_EXIT()  
  
  %ENDIF  
  
...  
  
</body>  
</html>  
%}
```

DTW_URLESCSEQ 関数は、URL で使用できない文字をその拡張値に置換します。たとえば、入力値 string1 が "Guys & Dolls" に等しい場合、DTW_URLESCSEQ は、出力変数に値 "Guys%20%26%20Dolls" を割り当てます。

数学関数

これらの関数は、数学操作を実行し、数値データの計算あるいは変更を行うことができます。標準的な数学操作の他にも、法による除算の実行、演算結果の精度の指定、科学表記の使用、などを行うことができます。

たとえば DTW_POWER は、その最初のパラメーターの値を 2 番目のパラメーターで累乗し、結果を戻します。以下の例のようになります。

```
@DTW_POWER("2", "-3", result)
```

DTW_POWER は、変数 result に ".125" を戻します。

ストリング関数

これらの関数により、ストリング内の文字を操作することができます。ストリングの大文字小文字の変更、文字の挿入あるいは削除、別の変数へのストリング値の割り当てを行うことができます。さらに、その他の役に立つ関数を実行することができます。

たとえば、DTW_ASSIGN を使用することにより、入力変数の値を出力変数に割り当てることができます。また、この関数を使用してマクロ内の変数が変更できます。以下の例では、変数 RC がゼロに割り当てられています。

```
@DTW_ASSIGN(RC, "0")
```

他にもストリング関数には、ストリングを連結する DTW_CONCAT、および特定の位置にストリングを挿入する DTW_INSERT など、多くのストリング処理関数があります。

ワード関数

これらの関数により、ストリング内のワードを操作することができます。これらの関数のほとんどは、ストリング関数と同じ働きをしますが、ワード全体に対して働きます。たとえば、これらの関数を使用して、ストリング内のワード数のカウント、ワードの削除、ストリングからのワードの取得、などを実行することができます。

たとえば、DTW_DELWORD を使用すると、ストリングから指定番号のワードを削除することができます。

```
@DTW_DELWORD("Now is the time", "2", "2", result)
```

DTW_DELWORD は、ストリング "Now time" を戻します。

他のワード関数には、ワードの文字数を戻す DTW_WORDLENGTH、およびストリング内の文字位置を戻す DTW_WORDPOS があります。

表関数

これらの関数を使用して、Net.Data の表変数のデータを使い、レポートやフォームを生成することができます。また、これらの関数を使用して Net.Data 表の作成、およびその表の値の操作および検索が行えます。表変数には、値のセット、およびこれに関連付けられた列の名前が含まれます。これらの関数は、値のグループを関数に渡すのに都合の良い方法を提供してくれます。

たとえば、DTW_TB_APPENDROW は表に行を追加します。以下の例では、Net.Data は表 myTable に 10 の列を追加します。

```
@DTW_TB_APPENDROW(myTable, "10")
```

この他にも、DTW_TB_DUMP はマクロ表変数の内容を <PRE></PRE> タグ付きで戻します。表の各行は、改行して表示されます。DTW_TB_CHECKBOX は、マクロ表変数から 1 つ以上の HTML チェック・ボックス入カタグを戻します。

フラット・ファイル関数

フラット・ファイル・インターフェース (FFI) 関数を使用して、フラット・ファイル内の保管データだけでなく、フラット・ファイルのソース (テキスト・ファイル) のデータのオープン、読み取り、および操作をすることができます。

たとえば、DTWF_APPEND は、表変数の内容をファイル末尾に書き込み、DTWF_DELETE はファイルからレコードを削除します。

この他にも、FFI 関数を使用することにより DTWF_CLOSE および DTWF_OPEN によるファイル・ロックが可能となります。DTWF_OPEN はファイルをロックして、他の要求によりファイルの読み取りまたは更新を行えないようにします。DTWF_CLOSE は、Net.Data が処理を完了するとファイルをリリースし、他の要求によりファイルがアクセスできるようにします。

マクロでの Web ページ生成

Net.Data により、標準 Web ページをアプリケーション・ユーザーのブラウザーに簡単に提供することができます。以下の節では、マクロの HTML および REPORT ブロックについて説明し、Net.Data マクロにおける Web ページのフォーマット方法を示します。これらのブロックの構文情報については、“*Net.Data 解説書*”の言語構成要素の章を参照してください。

HTML ブロック

Net.Data には Web ブラウザーに対して HTML などのテキスト表示ステートメントを生成する HTML ブロックが含まれます。マクロ内には、少なくとも 1 つの HTML を指定しなくてはなりません。これは必要な数だけ指定できます。それぞれの HTML ブロックは、ブラウザー上での単一の Web ページを生成します。Net.Data は、起動ごとに HTML ブロックを 1 つだけ処理します。多くの Web ページから構成されるアプリケーションを作成するには、Net.Data を複数回起動して、リンクおよびフォームなど通常のナビゲーション手法を使用した HTML ブロックの処理を行います。

HTML または JavaScript などの有効なテキスト表示ステートメントを HTML ブロックに使用することができます。さらに、INCLUDE ステートメント、関数呼び出し、および HTML ブロックの変数参照を使用することができます。以下の例は、Net.Data のマクロにおける HTML ブロックの一般的な使われ方を示しています。

```
%DEFINE DATABASE="MNS96"

%HTML(INPUT){
<H1>Hardware Query Form</H1>
<HR>
<FORM METHOD="POST" ACTION="/netdata-cgi/db2www/equip1st.d2w/report">
<dl>
<dt>What hardware do you want to list?
<dd><input type="radio" name="hardware" value="MON" checked>Monitors
<dd><input type="radio" name="hardware" value="PNT">Pointing devices
<dd><input type="radio" name="hardware" value="PRT">Printers
<dd><input type="radio" name="hardware" value="SCN">Scanners
</dl>
<HR>
<input type="submit" value="Submit">
</FORM>
%}

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

```
%}  
%}
```

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

Net.Data のマクロを、以下の例のように、HTML のリンクから起動することができます。

```
<a href="http://www.ibm.com/netdata-cgi/db2www/equip1st.d2w/input">  
List of hardware</a>
```

アプリケーション・ユーザーがこのリンクをクリックすると、Web ブラウザーは Net.Data を起動し、Net.Data はマクロを解析します。Net.Data が起動に指定された HTML ブロック、この場合は HTML (INPUT) ブロック、の処理を開始すると、Net.Data は、そのブロック内のテキストの処理を開始します。Net.Data は、Net.Data のマクロ言語構成要素として認識できないものはどれも、ブラウザーに送信して表示します。

ユーザーが選択を行い、「処理依頼 (Submit)」ボタンを押すと、Net.Data は HTML の FORM 要素の ACTION 部分を実行します。この部分は、Net.Data のマクロの HTML(OUTPUT) ブロックへの呼び出しを指定します。次に Net.Data は、HTML(INPUT) ブロックの場合と同様に、HTML(OUTPUT) ブロックを処理します。

Net.Data は次に、myQuery() 関数呼び出しを処理します。この関数呼び出しは、次に SQL FUNCTION ブロックを起動します。SQL ステートメントの \$(hardware) 変数参照を、入力フォームで戻された値と置き換えた後、Net.Data は照会を実行します。この時点で、Net.Data はレポート処理を再開し、REPORT ブロックで指定されたテキスト表示ステートメントに従って照会結果を表示します。

Net.Data は REPORT ブロックの処理を完了した後で、HTML(OUTPUT) ブロックに戻り、処理を終了します。

レポート・ブロック

REPORT ブロックの言語構成要素を使用して、FUNCTION ブロックからのデータ出力をフォーマットし、表示することができます。この出力は、基本的には表データです。ただし、テキスト、マクロ変数参照、および関数呼び出しの有効な組み合わせを指定することはできません。表の名前は、オプションで REPORT ブロックで指定することができます。SQL および ODBC 言語環境以外では、表の名前を指定しない場合は、Net.Data は、FUNCTION パラメーター・リストの最初の出力表の表データを使用します。

REPORT ブロックは、次の 3 つの部分を持ち、各部分はオプションです。

- ヘッダー情報。含まれるテキストは、テーブル行データの前に一度表示されます。
- ROW ブロック。結果表の行ごとに一度だけ表示されるテキストおよび表変数が含まれます。
- フッター情報。含まれるテキストは、テーブル行データの後に一度表示されます。

例:

```
%REPORT{
<H2>Query Results</H2>
<P>Select a name for details.
<TABLE BORDER=1>
  <TR>
    <TD>Name</TD>
    <TD>Location</TD></TR>
%ROW{
  <TR>
    <TD>
      <a href="/cgi-bin/db2www/name.d2w/details?name=$(V1)&location;=$(V2)">$(V1)</a>
    </TD>
    <TD>$(V2)</TD>
  </TR>
%}
</TABLE>
%}
```

REPORT ブロックのガイドライン

REPORT ブロックを作成する際は、以下のガイドラインに従います。

- ROW ブロックからの表出力はどれも表示をしないようにするには、ROW ブロックを空にしておくか、ROW ブロックを完全に省略します。
- Net.Data が提供する REPORT ブロック内の変数を使用して、Net.Data マクロの結果表のデータにアクセスします。これらの変数は、68ページの『表処理変数』で説明されています。追加詳細については、“*Net.Data 解説*”のレポート変数のセクションを参照してください。
- ヘッダーおよびフッター情報を提供するには、ROW ブロックの前後にテキストを入力します。Net.Data は、ROW ブロックの前に検出したものはすべてヘッダー情報として処理します。Net.Data は、ROW ブロックの後に検出したものはすべてフッター情報として処理します。HTML ブロックの場合と同様、Net.Data は、ヘッダー、ROW、およびフッター・ブロックにおいてマクロ言語構成要素として認識できないものをすべてテキスト表示ステートメントとして扱い、これらのデータをブラウザに送信します。
- REPORT ブロックでは関数および参照変数を呼び出すことができます。
- Net.Data に、フォーマット済みのテキストを使用してデフォルトのレポートをプリントさせるには、マクロに REPORT ブロックを組み込まないようにします。以下の例は、デフォルトのレポート・フォーマットを示しています。

SHIPDATE	RECDATE	SHIPNO
25/05/1997	30/05/1997	1495194B
25/05/1997	28/05/1997	2942821G

- HTML のタグをフォーマット済みテキストの代わりに使用するには、DTW_HTML_TABLE を YES に設定します。
- デフォルトのレポートのプリントを使用禁止にするには、DTW_DEFAULT_REPORT を NO に設定するか、空の REPORT ブロックを指定します。たとえば、以下のようにします。

```
%REPORT{%
```

例: レポートのカスタマイズ

以下の例は、特殊変数と HTML のタグを使用した、レポート・フォーマットのカスタマイズ方法を示しています。この例では、CustomerTbl の表から、名前、電話番号、および FAX 番号を表示しています。

```
%DEFINE SET_TOTAL_ROWS="YES"
...
%FUNCTION(DTW_SQL) custlist() {
    SELECT Name, Phone, Fax FROM CustomerTbl
%REPORT{
<I>Phone Query Results:</I>
<BR>
=====
<BR>
%ROW{
    Name: <B>$(V1)</B>
<BR>
    Phone: $(V2)
<BR>
    Fax: $(V3)
<BR>
    -----
<BR>
    %}
    Total records retrieved: $(TOTAL_ROWS)
    %}
%}
```

この結果作成されるレポートは、Web ブラウザーでは、次のように表示されます。

```
Phone Query Results:
=====
Name: Doen, David
Phone: 422-245-1293
Fax: 422-245-7383
-----
Name: Ramirez, Paolo
Phone: 955-768-3489
Fax: 955-768-3974
-----
Name: Wu, Jianli
Phone: 525-472-1234
Fax: 525-472-1234
-----
Total records retrieved: 3
```

Net.Data は、以下を行い、レポートを生成しました。

1. *Phone Query Results:* を、レポートの最初に 1 回プリントする。このテキストは、区切り線を含めて REPORT ブロックのヘッダー部分です。
2. 検索時に各行ごとに、変数 V1、V2、および V3 を、それぞれ Name、Phone、および Fax の値で置換します。
3. スtring *Total records retrieved:* および TOTAL_ROWS の値を、レポートの最後に一度プリントします。(このテキストは、REPORT ブロックのフッター部分です。)

複数の REPORT ブロック

DTW_SQL 言語環境または DTW_ODBC 言語環境で、複数の結果セットを戻すストアド・プロシージャを関数が呼び出す場合、複数の REPORT ブロックが使用できます。94ページの『ストアド・プロシージャ』を参照してください。

複数の REPORT ブロックを使用するには、それぞれの結果セットについてストアド・プロシージャの CALL ステートメントに結果セット名を配置します。指定した REPORT ブロックの数より多くの結果セットがストアド・プロシージャから戻される場合は、REPORT ブロックとは関係のない結果セットについてデフォルトのレポートが生成されます。この場合、DTW_DEFAULT_REPORT = "NO" によりデフォルトのレポート処理を使用不可にしていることを前提としています。

例: 以下の例では、複数のレポート・ブロックを使用する方法を示します。

デフォルトのレポート・フォーマットを使用した複数レポートの表示

例 1: DTW_SQL 言語環境

```
%FUNCTION (dtw_sql) myStoredProc () {  
    CALL myproc (table1, table2) %}
```

表示処理に対し **REPORT** ブロックを指定して複数のレポートを表示する

例 1: 名前付き REPORT ブロック

```
%FUNCTION(dtw_sql) myStoredProc  
( ) {  
    CALL myproc (table1, table2)  
  
    %REPORT(table2) {  
        ...  
        %ROW { .... %}  
        ...  
    %}  
  
    %REPORT (table1) {  
        ...  
        %row { .... %}  
        ...  
    %}  
%}
```

この例では、FUNCTION ブロック・パラメーター・リストで渡される両方の表について REPORT ブロックが指定されています。表は、REPORT ブロックで指定された順 (最初に table2、続いて table1) で表示されます。REPORT ブロックで表の名前を指定することにより、レポートが表示される順序が制御できます。

例 2: 名前なし REPORT ブロック

```
%FUNCTION(dtw_sql) myStoredProc  
( ) {  
    CALL myproc  
  
    %REPORT {  
        ...  
        %ROW { .... %}  
        ...  
    %}  
    %REPORT {  
        ...
```

```

        %ROW { .... %}
        ...
    %}
%}

```

この例では、FUNCTION ブロック・パラメーター・リストで渡される両方の表について REPORT ブロックが指定されています。REPORT ブロックには表の名前が指定されていないため、レポートは、2 つの表についてストアード・プロシージャから戻される順序で表示されます。

デフォルトのレポートと **REPORT** ブロックの組み合わせを使用して複数のレポートを表示

例: デフォルトのレポートと REPORT ブロックの組み合わせ

```

%DEFINE DTW_DEFAULT_REPORT = "YES"
%FUNCTION(dtw_sql) myStoredProc (OUT table1) {
    CALL myproc (table1, table2, table3)

    %REPORT(table2) {
        ...
        %ROW { .... %}
        ...
    %}

    %}

```

この例では、1 つの REPORT ブロックのみが指定されています。ブロックが table2 を指定しており、table2 は CALL ステートメントにリストされている 2 番目の結果セットであるため、2 番目の結果セットがレポート表示に使用されます。指定された REPORT ブロックがストアード・プロシージャから戻される結果セットの数より少ないため、余分な結果セットについてはデフォルトのレポートが表示されます。最初の結果セット table1 に対するデフォルトのレポートが最初に、続いて 3 番目の結果セット table3 にデフォルトのレポートが表示されます。出力表が 1 つ指定されていますが (table1)、これはマクロ・ファイル中、後の処理に使用することができます。

複数の REPORT ブロックのガイドラインおよび制約事項: FUNCTION ブロックで複数の REPORT ブロックを指定する際は、以下のガイドラインおよび制約事項に従います。

ガイドライン:

- 複数の表に対する REPORT ブロックの指定は、これら进行处理する順序で行います。
- 表に REPORT ブロックが指定されていない場合にデフォルトの処理を指定するには、DTW_DEFAULT_REPORT = "YES" を定義します。Net.Data は、Web ページを作成する際に、REPORT ブロックを持つ表のレポートを表示した後に表のデフォルト・レポートを表示します。
- Net.Data が、REPORT ブロックを持たない表を表示しないようにするには、DTW_DEFAULT_REPORT = "NO" に設定します。
- 複数の表を戻す関数で DTW_SAVE_TABLE_IN 変数を使用する場合、関数から戻される最初の表は、DTW_SAVE_TABLE_IN の表に割り当てられます。

制約事項:

- 複数の REPORT ブロックは、DTW_SQL または DTW_ODBC 言語環境を使用する関数で、複数の結果セットを戻すストアード・プロシージャを関数が呼び出す場合にのみ使用できます。
- 関数内のすべてのレポート変数の値は、その関数内のすべての REPORT ブロックに適用されます。個々の REPORT ブロックについて、レポート変数の値を変更することはできません。
- MESSAGE ブロックは、REPORT ブロックのリストの前後いずれかに配置する必要があり、REPORT ブロックの間には配置できません。
- 表変数は、関数に渡す前に TABLE ステートメント内で定義する必要があります。
- 最初のレポート・ブロックに表の名前が指定されていると、すべてのレポート・ブロックが表の名前を指定しなくてはなりません。
- 最初のレポート・ブロックが表の名前を指定していなければ、レポート・ブロックは表の名前を指定することはできません。
- 複数の REPORT ブロックは、同一の表に対して指定できません。

マクロにおける条件付き論理のループ

Net.Data により、IF および WHILE ブロックを使用して、条件論理およびループを Net.Data のマクロに取り込むことができます。

IF および WHILE ブロックは 条件リスト を使用します。これにより、1 つ以上の条件をテストし、次に条件テストの結果に基づいてステートメントのブロックを実行することができます。条件リストには、= および <+ などの論理演算子および用語が含まれ、これらは引用符付き文字列、変数、変数参照および関数呼び出しから構成されます。引用符付き文字列には、変数参照および関数呼び出しを含むことができます。条件リストはネストできます。

以下のセクションでは、条件付き論理およびループを説明しています。

- 『条件付き論理: IF ブロック』
- 87ページの『ループ構成体: WHILE ブロック』

条件付き論理: IF ブロック

IF ブロックを使用して、Net.Data マクロで条件付き処理を行います。IF ブロックは、ほとんどの高級言語の IF ステートメントに類似しています。その理由は、この IF ブロックは、1 つ以上の条件をテストし、次に条件テストの結果に基づき、ステートメントのブロックを実行することができるからです。

IF ブロックは、マクロ内のほとんどどこにでも指定することができ、それらをネストすることができます。IF ブロックの構文は、*Net.Data 解説書* の言語構成要素の章に示されています。

IF ブロックの構文規則は、マクロ内のブロックの位置により決定します。IF ブロックの実行可能なステートメント・ブロックに許される要素は、IF ブロック自身の位置に依存します。IF ブロックを含むブロック内で有効な要素ならどれでも、その IF ブロックで有効です。たとえば、IF ブロックを HTML ブロック内部に指定する場合、

HTML ブロックに許される要素はどれでも IF ブロックに許されます (INCLUDE ステートメントおよび WHILE ブロックなど)。

```
%HTML block
...
  %IF block
...
  %INCLUDE
...
  %WHILE
...
  %ENDIF
%}
```

同様に、IF ブロックを、Net.Data のマクロの宣言文の他のブロックの外で指定する場合は、その他のブロック (たとえば、DEFINE ブロック、あるいは FUNCTION ブロック) の外で許される要素のみが、IF ブロック内で許されます。

```
%IF
...
  %DEFINE
...
  %FUNCTION
...
  %ENDIF
```

IF ブロックが、宣言部分のその他のブロックの外にある IF ブロック内でネストされている場合、外側のブロックが使用可能ないずれの要素も使用することができます。IF ブロックが、IF ブロック内の他のブロック内でネストされている場合は、その存在するブロックの構文規則に従います。

たとえば、ネストされた IF ブロックは、HTML ブロック内にある場合に使用される規則に従います。

```
%IF
...
  %HTML {
...
  %IF
...
    %ENDIF
  %}
...
  %ENDIF
```

例外: IF ブロックで ROW ブロックを指定しないでください。

Net.Data は、IF ブロック条件リストを、条件を構成している項の内容に基づき、2 つの方法のうちのいずれか 1 つで処理します。デフォルトのアクションは、すべての項を、ストリングとして処理し、条件で指定されたストリング比較を実行します。ただし、比較が整数を表す 2 つのストリングで行われる場合、比較は数値となります。Net.Data は、ストリングが数字のみ (オプションで前に '+' または '-' 文字) を含む場合、ストリングが数値であると想定します。ストリングは、'+' あるいは '-' 以外の非数字文字を含むことができません。Net.Data は、非整数の数字の数値比較をサポートしません。

有効な整数ストリングの例:

```
+1234567890
-47
000812
92000
```

無効な整数ストリングの例:

```
- 20      (空白文字を含んでいる)
234,000   (コンマを含んでいる)
57.987    (小数点を含んでいる)
```

Net.Data は、IF 条件を、そのブロックを実行したときに評価します。これは Net.Data によって最初に読み取られるときとは異なる場合があります。たとえば、IF ブロックを REPORT ブロックに指定すると、Net.Data は、REPORT ブロックを含む FUNCTION ブロック定義を読み取るときに、IF ブロックに関連付けられた条件リストを評価しません。これを行うのは、関数を呼び出して実行するときです。これは、IF ブロックの条件リストの部分および実行されるステートメントのブロックの両方に対してもあてはまります。

例: 他のブロック内に IF ブロックを含むマクロ

```
%{ This macro is called from another macro, passing the operating system
   and version variables in the form data.
%}

%IF (platform == "OS390")
  %IF (version == "1.3")
    %INCLUDE "os390v1r3_def.hti"
  %ELIF (version == "2.0")
    %INCLUDE "os390v2r1_def.hti"
  %ELIF (version == "2.2")
    %INCLUDE "os390v2r2_def.hti"
  %ENDIF
%ELSE
  %INCLUDE "default_def.hti"
%ENDIF
%MACRO_FUNCTION numericCompare(IN term1, term2, OUT result) {
  %IF (term1 < term2)
    @dtw_assign(result, "-1")
  %ELIF (term1 > term2)
    @dtw_assign(result, "1")
  %ELSE
    @dtw_assign(result, "0")
  %ENDIF
%}

%HTML(report){
  %WHILE (a < "10") {
    outer while loop #$(a)<BR>
    %IF (@dtw_rdivrem(a,"2") == "0")
      this is an even number loop<BR>
    %ENDIF
    @DTW_ADD(a, "1", a)
  %}
%}
```

ループ構成体: WHILE ブロック

WHILE ブロックを使用して、Net.Data のマクロでループを実行します。IF ブロックと同様、WHILE ブロックにより、1 つ以上の条件をテストし、次に、条件テスト

の結果に基づいてステートメントのブロックを実行することができます。IF ブロックと異なり、ステートメントのブロックは、条件テスト結果に基づき、何回でも実行することができます。

WHILE ブロックを HTML ブロック、REPORT ブロック、ROW ブロック、および IF ブロック内で指定し、ネストすることができます。WHILE ブロックの構文は、*Net.Data* 解説書 の言語構成要素の章に示されています。

Net.Data は、WHILE を、IF ブロックを処理するのと全く同じ方法で処理しますが、ブロックを実行するごとに条件を再評価します。そして、どの条件付きループ構成要素の場合も同じですが、条件のコード化に誤りがある場合は、処理は無限ループに陥ることがあります。

例: WHILE ブロックを持つマクロ

```
%DEFINE loopCounter = "1"

%HTML(build_table) {
  %WHILE (loopCounter <= "100") {
    %{ generate table tag and column headings %}
    %IF (loopCounter == "1")
      <TABLE BORDER>
      <TR>
      <TH>Item #
      <TH>Description
    %ENDIF

    %{ generate individual rows %}
    <TR>
    <TD>$(loopCounter)
    <TD>@getDescription(loopCounter)

    %{ generate end table tag %}
    %IF (loopCounter == "100")
    %ENDIF

    %{ increment loop counter %}
    @DTW_ADD(loopCounter, "1", loopCounter)
  %}
%}
```

第6章 言語環境の使用

Net.Data はデータ・ソースへのアクセスと、ビジネス・ロジックを持つアプリケーション・プログラムの実行のための言語環境を提供します。たとえば、SQL 言語環境によって、ユーザーは SQL ステートメントを DB2 サブシステムに渡すことが可能になり、REXX 言語環境によって、ユーザーは REXX プログラムを起動できるようになります。また、SYSTEM 言語環境を使用することにより、たとえば、CICS プログラムを実行するための外部 CICS インターフェース (EXCI) を使用する外部プログラムを実行することができます。

Net.Data を使うことにより、ユーザー作成の言語環境をプラグイン方式で追加することができます。それぞれのユーザー作成言語環境は、Net.Data によって定義された標準的なインターフェースのセットをサポートしていなければなりません。さらに、ダイナミック・リンク・ライブラリー (DLL)。Net.Data 提供の言語環境およびユーザー作成の言語環境の作成方法の詳細については、*Net.Data 言語環境解説書* を参照してください。

図6 では、Web サーバー、Net.Data、および Net.Data 言語環境間の関連を表示しています。

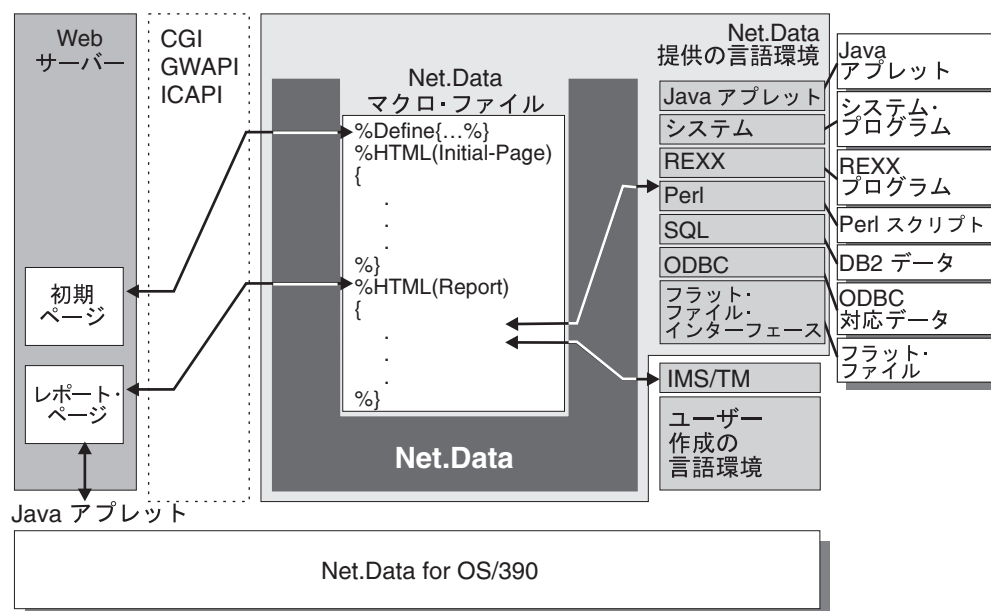


図6. Net.Data 言語環境

以下に示したセクションでは、Net.Data 言語環境と、それをマクロで使用方法について解説しています。

- 90ページの『Net.Data 提供の言語環境の概説』
- 91ページの『言語環境の呼び出し』
- 91ページの『データ言語環境』
- 103ページの『プログラミング言語環境』

Net.Data 提供の言語環境の構成情報については、20ページの『言語環境のセットアップ』を参照してください。

言語環境を使用した場合のパフォーマンスの向上については、117ページの『第7章 パフォーマンスを向上させる』を参照してください。

Net.Data 提供の言語環境の概説

Net.Data は、アプリケーション用のデータおよびプログラミング資源へのアクセスを可能にしてくれる言語環境を提供します。

Net.Data は次の 2 つの型の言語環境を提供します。

- 91ページの『データ言語環境』
- 103ページの『プログラミング言語環境』

表3 では各言語環境を簡単に説明しています。

表3. Net.Data 言語環境

言語環境	環境名	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、データ・ソースとしてのテキスト・ファイルをサポートする関数をサポートしています。
IMS Web	HWS_LE	IMS Web 言語環境では、IMS Web を使用して IMS トランザクションを処理依頼し、Web ブラウザーでそのトランザクションの出力を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では、Net.Data アプリケーションで Java アプレットを使用することができます。applet タグ を生成するには、applet タグの限定子とアプレットのパラメーター・リストを提供しなければなりません。
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 インターフェースにより SQL を実行します。
Perl	DTW_PERL	Perl 言語環境は、Net.Data の FUNCTION ブロックで指定された内部 Perl スクリプトを解釈したり、別のファイルに保管されている外部 Perl スクリプトを実行します。
REXX	DTW_REXX	REXX 言語環境は、Net.Data の FUNCTION ブロックで指定された内部 REXX プログラムを解釈したり、別のファイルに保管されている外部 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、表変数に格納して戻すことができます。

表 3. *Net.Data* 言語環境 (続き)

言語環境	環境名	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、外部プログラムの呼び出しをサポートします。システム言語環境は、プログラムを実行するために、プログラム名とそのパラメータをオペレーティング・システムに渡すことにより、EXEC ステートメントを解釈します。

言語環境の呼び出し

言語環境を呼び出すには以下のようにします。

- FUNCTION ステートメントを使用して、言語環境を呼び出す関数を定義する。
- 言語環境への関数呼び出しを使用する。

たとえば、以下のようにします。

```
...
%function(dtw_odbc) myStoredProc() {
create table int_null (int1 int, int2 int)
%}
...
%HTML(REPORT) {
@myStoredProc()
%}
```

データ言語環境

Net.Data が提供するデータ言語環境では、関係データベースおよび階層データベースからデータにアクセスできます。また、その他のデータ・ソースには *Net.Data* のマクロからアクセスできます。以下のセクションでは、*Net.Data* が提供するデータ言語環境と、*Net.Data* マクロにおけるその使用方法について説明します。

- 『関係データベース言語環境』
- 101ページの『フラット・ファイル・インターフェース言語環境』
- 102ページの『IMS Web 言語環境』

アクセス権: *Net.Data* を実行するときのユーザー ID が、データベース照会の実行のための、あるいはトランザクションを完了するのに必要なファイルへのアクセスのためのアクセス権を持っていることを確認してください。詳しくは、29ページの『*Net.Data* がアクセスするファイルおよびデータ・セットへのアクセス権の授与』を参照してください。

関係データベース言語環境

Net.Data は、関係データ・ソースにアクセスするのに役立つ関係データベース言語環境を提供します。 *Net.Data* は以下の関係データベース言語環境を提供します。

ODBC 言語環境

オープン・データベース・コネクティビティ (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

X/Open SQL CAE 仕様に基づいています。この仕様では、単一のアプリケーションから多数のデータベース管理システムにアクセスできます。

ODBC 言語環境の使用方法:

CLI 初期設定ファイルの場所が、構成変数 DSNAOINI で指定されていることを確認します。DSNAOINI 構成変数の設定方法については、11ページの『DSNAOINI: DB2 CLI の初期設定ファイル変数』を参照してください。

以下の構成ステートメントが、初期設定ファイルに 1 行で記述されていることを確認してください。

```
ENVIRONMENT (DTW_ODBC) odbcd11  
( )
```

ENVIRONMENT ステートメントで指定できる変数は、TRANSACTION_SCOPE、LOCATION です。

制約事項

- インラインのステートメント・ブロックの SQL ステートメントの最大サイズは 32KB です。

SQL 言語環境

SQL 言語環境では、DB2 データベースにアクセスできます。DB2 へのアクセス時に最適のパフォーマンスを得るには、この言語環境を使用してください。

SQL 言語環境の使用方法:

以下の構成ステートメントが、初期設定ファイルに 1 行で記述されていることを確認してください。

```
ENVIRONMENT (DTW_SQL) dtwsq1 ( )
```

ENVIRONMENT ステートメントに指定できる変数は、TRANSACTION_SCOPE、LOCATION、DB2SSID、DB2PLAN です。

重要:SQL 言語環境のセットアップ方法については、21ページの『SQL および ODBC の言語環境』を参照してください。

制約事項: インラインのステートメント・ブロックの SQL ステートメントの最大サイズは 32KB です。

以下のセクションでは、これらの言語環境の使用方法について説明しています。

- 『Net.Data アプリケーションにおけるトランザクション管理』
- 94ページの『ストアド・プロシージャ』
- 99ページの『例』

Net.Data アプリケーションにおけるトランザクション管理

挿入、削除、または更新ステートメントを用いてデータベースの内容を変更する場合、その変更は、データベースが Net.Data からコミット・ステートメントを受け取る

までは永続的なものとはなりません。エラーが発生すると、Net.Data はデータベースにロールバック・ステートメントを送り、前回のコミットからの修正をすべてリバースにします。

Net.Data によるコミットおよび場合によってはロールバックの送信は、TRANSACTION_SCOPE の設定方法、およびマクロにコミットが明示的に指定されているかどうかにより異なります。 TRANSACTION_SCOPE の値は MULTIPLE および SINGLE です。

MULTIPLE

コミットおよび場合によってはロールバック・ステートメントが発行される前に、Net.Data がすべての SQL ステートメントを実行するかを指定します。Net.Data は要求の最後にコミットを送信し、各 SQL ステートメントが正常に発行されると、コミットによりデータベースのすべての変更が永続的なものとなります。ステートメントのいずれかがエラーを戻すと、Net.Data はロールバック・ステートメントを発行し、データベースの設定は元の状態に戻ります。TRANSACTION_SCOPE が設定されていなければ、MULTIPLE がデフォルトです。

このコミット・メソッドを活動化するには、TRANSACTION_SCOPE を MULTIPLE に設定します。

たとえば、以下のようにします。

```
@DTW_ASSIGN(TRANSACTION_SCOPE,"MULTIPLE")
```

SINGLE

各 SQL ステートメントが正常に終了した後に Net.Data がコミット・ステートメントを発行するよう指定します。SQL ステートメントがエラーを戻すと、ロールバック・ステートメントが発行されます。単一トランザクション効力範囲によりデータベースの即時変更が確実となりますが、この効力範囲では後でロールバック・ステートメントを使用して変更を取り消すことはできません。

このコミット・メソッドを活動化するには、TRANSACTION_SCOPE を SINGLE に設定します。たとえば、以下のようにします。

```
@DTW_ASSIGN(TRANSACTION_SCOPE,"SINGLE")
```

コミット・ステートメントは、COMMIT SQL ステートメントを使用することによりマクロ内の任意の SQL ステートメントの最後で発行することができます。アプリケーション開発者は、TRANSACTION_SCOPE の設定を MULTIPLE にして、トランザクションとみなしたステートメント・グループの最後にコミット・ステートメントを発行することにより、アプリケーションにおけるコミットおよびロールバックの振る舞いを完全に管理することができます。

SQL コミット・ステートメントを発行するには、HTML ブロックの任意のポイントに呼び出し可能な関数を定義します。

```
%FUNCTION(DTW_SQL) user_commit() {  
    commit  
}%  
...  
%HTML {
```

```

...
@user_commit()
...
%}

```

ストアード・プロシージャ

ストアード・プロシージャは DB2 に保管されたコンパイル済みのプログラムで、SQL ステートメントを実行することができます。Net.Data では、ストアード・プロシージャは、CALL ステートメントを使用して、Net.Data の関数から呼び出されます。ストアード・プロシージャのパラメーターは、Net.Data の関数仮引き数リストから渡されます。ストアード・プロシージャを使用すると、コンパイル済みの SQL ステートメントを、データベース・サーバーと一緒に保管することにより、パフォーマンスと保全性を改良することができます。Net.Data は、SQL および ODBC 言語環境での DB2 によるストアード・プロシージャ使用をサポートします。

このセクションでは以下のトピックを説明します。

- 『ストアード・プロシージャ構文』
- 95ページの『ストアード・プロシージャの呼び出し』
- 96ページの『パラメーターを渡す』
- 96ページの『結果セットの処理』

ストアード・プロシージャ構文: ストアード・プロシージャ構文は FUNCTION ステートメント、CALL ステートメント、および REPORT ブロック (オプション) を使用します。

```

%FUNCTION (DTW_lang_env) function_name ([IN datatype arg1, INOUT datatype arg2,
    OUT tablename, ...]) {
    CALL stored_procedure [(resultsetname, ...)]
[%REPORT [(resultsetname)] { %}]
...
[%REPORT [(resultsetname)] { %}]
[%MESSAGE %}]
%}

```

ここで、

lang_env

呼び出される言語環境の名前。DTW_SQL または DTW_ODBC とすることができます。

function_name

ストアード・プロシージャの呼び出しを開始する Net.Data 関数名。

stored_procedure

ストアード・プロシージャの名前。

datatype

Net.Data がサポートするデータベース・データ型の 1 つ (95ページの表4を参照)。パラメーター・リストに指定したデータ型は、ストアード・プロシージャ内のデータ型に一致する必要があります。これらのデータ型に関するさらに詳しい情報については、データベースの文書を参照してください。

tablename

結果セットを保管する Net.Data テーブルの名前 (結果セットを Net.Data テーブ

ルに保管する場合のみ使用)。指定した場合、パラメーター名は *resultsetname* の関連パラメーター名に一致する必要があります。

resultsetname

ストアド・プロシージャから戻された結果を **REPORT** ブロックまたは関数仮引き数リスト (あるいはその両方) に関連付ける名前。**REPORT** ブロックの *resultsetname* は、**CALL** ステートメントの結果セットに一致する必要があります。

表 4. ストアド・プロシージャのデータ型

CHAR	FLOAT	SMALLINT
DECIMAL	INTEGER	VARCHAR
DOUBLE	GRAPHIC	VARGRAPHIC
DOUBLEPRECISION		

ストアド・プロシージャの呼び出し:

1. ストアド・プロシージャへの呼び出しを開始する関数を定義します。

```
%FUNCTION (DTW_SQL) function_name()
```

2. オプションで任意の **IN**、**INOUT**、または **OUT** パラメーターをストアド・プロシージャに指定します。これには、結果セットを **Net.Data** テーブルに保管するためにテーブル変数名 (結果セットを **Net.Data** テーブルに保管する場合は、**Net.Data** テーブルのみを指定) が含まれます。

```
%FUNCTION (DTW_SQL) function_name (IN datatype  
arg1, INOUT datatype arg2,  
OUT tablename...)
```

3. **CALL** ステートメントを使用して、ストアド・プロシージャ名を識別します。

```
CALL stored_procedure
```

4. ストアド・プロシージャが 1 つの結果セットを生成する場合は、オプションで **REPORT** ブロックを指定して **Net.Data** による結果セットの表示方法を定義します。

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

例:

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30)  
arg1) {  
    CALL myproc  
    %REPORT (mytable){  
        ...  
        %ROW { ...    %}  
        ...  
    %}  
%}
```

5. ストアド・プロシージャが複数の結果セットを生成する場合には以下のようにします。

- **CALL** ステートメントに結果セット名を指定します。

```
CALL stored_procedure (resultsetname1, resultsetname2, ...)
```

- オプションで 1 つ以上の **REPORT** ブロックを指定して、**Net.Data** による結果セットの表示方法を定義します。

```
%REPORT(resultsetname1) {
...
%}
```

例:

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) arg1, OUT table1) {
  CALL myproc (table1, table2)
  %REPORT(table2) {
    ...
    %ROW { ... %}
    ...
  %}
  %REPORT (table1) {
    ...
    %ROW { ... %}
    ...
  %}
  %}
%}
```

パラメーターを渡す: ストアド・プロシージャにパラメーターを渡すことができます。また、ストアド・プロシージャがパラメーターを更新するようにして、新規値が `Net.Data` マクロに渡されるようにできます。関数仮引き数リストのパラメーターの数および型は、ストアド・プロシージャに定義した数および型に一致する必要があります。たとえば、ストアド・プロシージャに定義したパラメーター・リストのパラメーターが `INOUT` の場合、関数仮引き数リストにある対応パラメーターは `INOUT` でなくてはなりません。ストアド・プロシージャに定義したりストのパラメーターが `CHAR(30)` の場合、関数仮引き数リストの対応パラメーターは `CHAR(30)` でなくてはなりません。

例 1: ストアド・プロシージャにパラメーター値を渡す

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) valuein) {
  CALL myproc
  ...
}
```

例 2: ストアド・プロシージャから値を戻す

```
%FUNCTION (DTW_SQL) mystoredproc (OUT VARCHAR(9) retvalue) {
  CALL myproc
  ...
}
```

結果セットの処理: SQL または ODBC 言語環境を使用することによりストアド・プロシージャから 1 つ以上の結果セットを戻すことができます。結果セットは、`Net.Data` テーブルに保管してさらにマクロで処理をするか、または `REPORT` ブロックを使用して処理することができます。ストアド・プロシージャが複数の結果セットを生成する場合、ストアド・プロシージャの生成した結果セットにそれぞれ名前を関連付ける必要があります。これは、`CALL` ステートメントにパラメーターを指定して行います。これにより、結果セットに指定した名前は `REPORT` ブロック、または `Net.Data` テーブルに関連付けることができ、`Net.Data` によるそれぞれの結果セットの処理方法が判別できます。以下を行うことができます。

- 結果セットにレポート・ブロックを定義せずに、結果を `Net.Data` のデフォルトのレポート・スタイルで処理します。
- 結果セットを `REPORT` ブロックに関連付け、自身のレポート・スタイルを適用します。 `REPORT` ブロックでは、`Net.Data` 変数、 `HTML` または `JavaScript` などのテキスト処理ステートメント、またはその他の関数を使用して、ブラウザーのレポート・データ表示方法が指定できます。

- 後に Net.Data マクロでデータを使用する場合は、結果セットを Net.Data テーブルに保管します。たとえば、Net.Data テーブルを他の関数に渡して、データを計算に使用し、その計算に基づいた結果を表示することができます。

複数のレポート・ブロックを使用する場合は、84ページの『複数の REPORT ブロックのガイドラインおよび制約事項』のガイドラインおよび制限を参照してください。

単一の結果セットを戻してデフォルトのレポートを使用する

以下の構文を使用します。

```
%FUNCTION (DTW_SQL) function_name
() {
    CALL stored_procedure
%}
```

たとえば、以下のようにします。

```
%FUNCTION (DTW_SQL) mystoredproc() {
    CALL myproc
%}
```

単一の結果セットを戻して **REPORT** ブロックを指定する

以下の構文を使用します。

```
%FUNCTION (DTW_SQL) function_name
() {
    CALL stored_procedure [(resultsetname)]
    %REPORT [(resultsetname)] {
        ...
    %}
%}
```

たとえば、以下のようにします。

```
%FUNCTION (DTW_SQL) mystoredproc () {
    CALL myproc
    %REPORT {
        ...
        %ROW { ' ... ' }
        ...
    %}
%}
```

または、以下の構文が使用できます。

```
%FUNCTION (DTW_SQL) function_name () {
    CALL stored_procedure (resultsetname)

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

たとえば、以下のようにします。

```
%FUNCTION (DTW_SQL) mystoredproc () {
    CALL myproc (mytable1)
    %REPORT (mytable1) {
        ...
    %}
%}
```

```

%ROW { ... %}
...
%}
%}

```

さらに処理を行うために **Net.Data** テーブルに単一の結果セットを保管する

以下の構文を使用します。

```

%FUNCTION (DTW_SQL) function_name (OUT tablename) {
    CALL stored_procedure (resultsetname)
%}

```

たとえば、以下のようにします。

```

%DEFINE DTW_DEFAULT_REPORT = "NO"

%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1) {
    CALL myproc (mytable1)
%}

```

結果セットにデフォルトのレポートを生成しないよう DTW_DEFAULT_REPORT は NO に設定されています。

複数の結果セットを戻し、これをデフォルトのレポート・フォーマットで表示する

以下の構文を使用します。

```

%FUNCTION (DTW_SQL) function_name
() {
    CALL stored_procedure [(resultsetname1, resultsetname2, ...)]
%}

```

ただし、レポート・ブロックは指定されていません。

たとえば、以下のようにします。

```

%DEFINE DTW_DEFAULT_REPORT = "YES"
%FUNCTION (DTW_SQL) mystoredproc () {
    CALL myproc
%}

```

複数の結果セットを戻し、さらに処理を行うために結果セットを **Net.Data** テーブルに保管する

以下の構文を使用します。

```

%FUNCTION (DTW_SQL) function_name (OUT tablename1, tablename2, ...) {
    CALL stored_procedure (resultsetname1, resultsetname2, ...)
%}

```

たとえば、以下のようにします。

```

%DEFINE DTW_DEFAULT_REPORT = "NO"

%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1, mytable2) {
    CALL myproc (mytable1, mytable2)
%}

```

結果セットにデフォルトのレポートを生成しないよう DTW_DEFAULT_REPORT は NO に設定されています。

複数の結果セットを戻し、表示処理のために **REPORT** ブロックを指定する

それぞれの結果セットは、1 つ REPORT ブロックに関連付けられています。以下の構文を使用します。

```
%FUNCTION (DTW_SQL) function_name (, ...) {  
    CALL stored_procedure (resultsetname1, resultsetname2, ...)  
    %REPORT (tablename1)  
    ...  
    %ROW { ... %}  
    ...  
}%  
%REPORT (tablename2)  
    ...  
    %ROW { ... %}  
    ...  
}%  
...  
}%
```

たとえば、以下のようにします。

```
%FUNCTION (DTW_SQL) mystoredproc () {  
    CALL myproc (mytable1, mytable2)  
  
    %REPORT(mytable1) {  
        ...  
        %ROW { ... %}  
        ...  
    }  
  
    %REPORT(mytable2) {  
        ...  
        %ROW { ... %}  
        ...  
    }  
}%
```

複数の結果セットを戻し、それぞれの結果セットに異なる表示または処理オプションを指定する

固有のパラメーター名を使用することにより、それぞれの結果セットに異なる処理オプションが指定できます。たとえば、以下のようにします。

```
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable2) {  
    CALL myproc (mytable1, mytable2, mytable3)  
  
    %REPORT(mytable1)  
    ...  
    %ROW { ... %}  
    ...  
}%
```

結果セット mytable1 は、対応する REPORT ブロックにより処理され、マクロ書き込みプロセスの指定どおりに表示されます。結果セット mytable2 は、Net.Data テーブル mytable2 に保管され、さらに処理を行う (他の関数に渡すなど) ために使用することができます。結果セット mytable3 には REPORT ブロックが指定されていないので、Net.Data のデフォルト・レポート・フォーマットを使用して表示されます。

例

以下の例では、マクロから関係データベース言語環境を呼び出す方法を示しています。

SQL および ODBC

以下の例は、DTW_SQL 関数定義を持ち、SQL ストアド・プロシージャを呼び出すマクロを示しています。ODBC 言語環境の場合は、DTW_SQL をすべて DTW_ODBC に置き換えます。このマクロは、データ型の異なる 3 つのパラメーターを持っています。DTW_SQL 言語環境は各パラメーターを、そのデータ型に従ってストアド・プロシージャに渡します。ストアド・プロシージャが処理を完了すると、出力パラメーターが戻され、Net.Data はそれに応じて変数を更新します。

```
%{*****
                                DEFINE BLOCK
*****%}
%DEFINE {
  MACRO_NAME      = "TEST ALL TYPES"
  DTW_HTML_TABLE  = "YES"
  Procedure       = "TESTTYPE"
  parm1           = "1"                %{SMALLINT      %}
  parm2           = "11"               %{INT           %}
  parm3           = "1.1"              %{DECIMAL (2,1) %}
  %}
  %FUNCTION(DTW_SQL) myProc
    (INOUT SMALLINT parm1,
     INOUT INT      parm2,
     INOUT DECIMAL(2,1) parm3){
  CALL $(Procedure)
  %}
  %HTML(REPORT) {
  <HEAD>
  <TITLE>Net.Data : SQL Stored Procedure: Example '$(MACRO_NAME)'. </TITLE>
  </HEAD>
  <BODY BGCOLOR="#BBFFFF" TEXT="#000000" LINK="#000000">
  <p><p>
  Calling the function to create the stored procedure.
  <p><p>
  @CRTPROC()
  < hr>
  <h2>
  Values of the INOUT parameters
  prior to calling the stored procedure:<p>
  </h2>
  <b>parm1 (SMALLINT)</b><br>
  $(parm1)<p>
  <b>parm2 (INT)</b><br>
  $(parm2)<p>
  <b>parm3 (DECIMAL)</b><br>
  $(parm3)<p>
  <p>
  < hr>
  <h2>
  Calling the function that executes the stored procedure.
  </h2>
  <p><p>
  @myProc(parm1,parm2,parm3)
  < hr>
  <h2>
  Values of the INOUT parameters after
  calling the stored procedure:<p>
  </h2>
  <b>parm1 (SMALLINT)</b><br>
  $(parm1)<p>
  <b>parm2 (INT)</b><br>
  $(parm2)<p>
```

```
<b>parm3 (DECIMAL)</b><br>
$(parm3)<p>
</body>
%}
```

フラット・ファイル・インターフェース言語環境

データ・ソースとしてフラット・ファイル (すなわち平文ファイル) を使用する場合は、フラット・ファイル・インターフェース (FFI) とそれに関連付けられている、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、そして削除するための関数を使用します。ファイル言語サポートは、ブラウザから Web クライアントの要求が発生すると、FFI 関数を使用して Web サーバー上のファイルに対して読み取り、または書き込みを行います。FFI はファイルをレコード・ファイルとして表示します。レコードは Net.Data マクロ表変数の行と等価であり、レコードの値は、Net.Data マクロ表変数のフィールドの値と等価です。FFI はファイルからレコードを Net.Data マクロ表の行に読み込み、行を表からレコードに書き込みます。

FFI 組み込み関数の説明と構文については、*Net.Data* 解説書 を参照してください。

FFI 言語環境の構成

以下の構成ステートメントが、初期設定ファイルに 1 行で記述されていることを確認してください。

```
ENVIRONMENT (DTW_FILE)   filed11   ()
```

Net.Data の初期設定ファイルと言語環境の ENVIRONMENT ステートメントの詳細については、18ページの『環境構成ステートメント』 を参照してください。

FFI 組み込み関数の呼び出し

FFI 関数の呼び出しは、他の関数の呼び出しと同じです。DEFINE ステートメントを使用して、渡したいパラメーターを変数として定義します。たとえば次のように定義します。

```
%DEFINE {
    myFile = "c:/private/myfile"
    myTable = %TABLE
    myWait = "1500"
    myRows = "2"
%}
```

次に関数呼び出しステートメントを使用して関数を呼び出します。たとえば次のようにします。

```
@DTWF_UPDATE(myFile, "Delimited", "|", myTable, myWait, myRows)
```

例

この例では、Net.Data は ffi001.dat の内容を Net.Data の表に読み込み、この表の内容を tmp.dat ファイルに書き込みます。最後に、Net.Data は tmp.dat ファイルを削除します。

```
%DEFINE {
mytable = %TABLE(ALL)
myfile = "/usr/lpp/netdata/ffi//ffi001.dat"
tmpfile = "/usr/lpp/netdata/ffi/tmp.dat"
```



```
%}
%HTML(report){
@DTWF_READ(myfile, "ASCIITEXT", " ", mytable)
@DTW_TB_TABLE(mytable)
@DTWF_WRITE(tmpfile, "ASCIITEXT", " ", mytable)
@DTW_TB_TABLE(mytable)
@DTWF_REMOVE(tmpfile)
%}
```

IMS Web 言語環境

IMS Web 言語環境は、Net.Data を使用して WWW 環境で IMS トランザクションを実行するための完全な終端間解決のパーツです。IMS Web 言語環境は、以下を提供します。

- 以下を持つ Net.Data マクロ
 - トランザクションの入力データを入力するのに使用する HTML
 - IBM 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 言語環境から呼び出される トランザクションの DLL または共有ライブラリー

IMS Web Studio ツールは、DLL およびマクロのためのコードと、DLL 実行可能ファイル を構築するための make ファイルを、トランザクションのメッセージ形式サービス (MFS) のソース、および IMS Web の Net.Data アプリケーションのための HTML ページのサンプルから生成します。実行可能な形式の DLL を構築後、ユーザーはその DLL とマクロを Net.Data を実行している Web サーバーに移動します。トランザクションは、Web 環境で作動可能になります。

IMS Web は Web サーバーと IMS 環境との間で通信を行うために IMS TCP/IP オープン・トランザクション・マネージャー・アクセス (OTMA) 接続を使用します。

IMS Web の使い方の詳細は、IMS Web のホーム・ページを参照してください。(英語版)

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

IMS Web 言語環境の構成

IMS Web 言語環境を使用するには、Net.Data 初期設定ファイルを検査し、言語環境をセットアップしなければなりません。

以下の構成ステートメントが、初期設定ファイルに 1 行で記述されていることを確認してください。

```
ENVIRONMENT (HWS_LE)      hwsd11      ()
```

Net.Data の初期設定ファイルと言語環境の ENVIRONMENT ステートメントの詳細については、18ページの『環境構成ステートメント』を参照してください。

重要: IMS 言語環境のセットアップ方法については、21ページの『IMS Web の言語環境』を参照してください。

制約事項

Net.Data の Web 言語環境は、Net.Data が CGI アプリケーションとして実行される場合しかサポートされません。

プログラミング言語環境

Net.Data は、外部プログラムを呼び出す際に使用するための以下の言語環境を提供しています。

- 『Java アプレット言語環境』
- 109ページの『Perl 言語環境』
- 112ページの『REXX 言語環境』
- 115ページの『システム言語環境』

アクセス権: Net.Data を実行するときのユーザー ID が、プログラムがアクセスする可能性のあるオブジェクトを含めて、プログラムを実行するためのアクセス権を持っていることを確認してください。詳しくは、29ページの『Net.Data がアクセスするファイルおよびデータ・セットへのアクセス権の授与』を参照してください。

Java アプレット言語環境

Java アプレット言語環境では、Net.Data アプリケーションに Java アプレット用の HTML タグを簡単に生成することができます。Java アプレット言語環境を呼び出す際に、アプレット名を指定し、そのアプレットが必要とするパラメーターを渡します。Java アプレット言語環境は、マクロを処理し、HTML アプレット・タグを生成します。Web ブラウザーはこのタグを使用して、アプレットを実行します。

さらにNet.Data は、アプレットが表パラメーターにアクセスするのに使用可能なインターフェースのセットを提供します。これらのインターフェースは DTW_Applet.class というクラスに含まれています。

以下のセクションでは、Java アプレット言語環境を使用して Java アプレットを実行する方法について説明しています。

Java アプレット言語環境の構成

以下の構成ステートメントが、初期設定ファイルに 1 行で記述されていることを確認してください。

```
ENVIRONMENT (DTW_APPLET)    app1dl1    ()
```

Net.Data の初期設定ファイルと言語環境の ENVIRONMENT ステートメントの詳細については、18ページの『環境構成ステートメント』を参照してください。

Java アプレットの作成

Net.Data の Java アプレット言語環境を使用する前に、どのアプレットを使用するか、あるいはどのアプレットを作成する必要があるのかを決定する必要があります。アプレットの作成の詳細については、Java のドキュメンテーションを参照してください。

アプレット・タグの生成

アプレット言語環境は、Net.Data の関数呼び出しを使って呼び出します。関数呼び出しには宣言は不要です。関数呼び出しの構文を以下に示します。

```
@DTWA_AppletName(parm1, parm2, ..., parmN)
```

- DTWA_ は、アプレット言語環境への関数呼び出しを識別します。
- AppletName は、タグが生成されるアプレットの名前です。 generated.
- parm1 から parmN は、PARAM タグを生成するのに使用されるパラメーターです。

アプレット・タグを生成するマクロの作成方法:

1. アプレットに必要なパラメーターをすべてマクロの DEFINE セクションに定義する。これらのパラメーターには、アプレットの入力として必要な、アプレット・タグの属性、Net.Data の変数、および Net.Data の表パラメーターをすべて含めます。たとえば、以下のようにします。

```
%define{
DATABASE = "celdial"                <=Net.Data variable: name of the database
MyGraph.codebase = "/netdata-java/" <=Required applet attribute
MyGraph.height = "200"              <=Required applet attribute
MyGraph.width = "400"               <=Required applet attribute
MyTitle = "This is my Title"        <=Net.Data variable: name of the Web page
MyTable = %TABLE(all)               <=Table to store query results
%}
```

2. オプション: アプレットの入力としての結果セットを生成するためのデータベースへの照会を指定する。これは、図あるいは表を生成するアプレットを使用する場合に役に立ちます。たとえば、以下のようにします。

```
%FUNCTION(DTW_SQL) mySQL(OUT table){
select name, ages from ibmuser.guests
%}
```

3. Java アプレット言語環境を呼び出し、アプレットを起動するための関数呼び出しを Net.Data マクロに指定する。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、アプレットの入力として必要な Net.Data 変数、および Net.Data の表または列パラメーターをすべて含めます。

たとえば、以下のようにします。

```
%HTML(report){                                <=The start of the HTML block
@mySQL(MyTable)                                <=A call to the SQL function
                                                mySQL
@DTWA_MyGraph(MyTitle, DTW_COLUMN(ages) MyTable) <=Applet function call
%}
```

アプレット・タグの属性: アプレット・タグの属性は Net.Data マクロの任意の位置に指定できます。Net.Data は、*AppletName.attribute* という形式を持つすべての変数を属性としてアプレット・タグに置き換えます。属性をアプレット・タグに定義するための構文を以下に示します。

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

以下に示した属性はすべてのアプレットで必須です。

- *codebase*: アプレットの場所で、URL で識別されます。
- *height*: アプレットの高さをピクセルで指定します。
- *width*: アプレットの幅をピクセルで指定します。

以下の属性はオプションです。

- *align*: アプレットの配置を指定します。
- *alt*: ブラウザーが APPLET タグを理解できても、Java アプレットを実行できない場合に表示するテキストを指定します。
- *archive*: クラスおよびその他のリソースを含むアーカイブを指定します。
- *hspace*: アプレットの両側をピクセル数で指定します。
- *name*: アプレット・インスタンスの名前を指定します。
- *object*: アプレットのシリアル化表示を含んでいるファイルの名前を指定します。
- *vspace*: アプレットの上下のピクセル数を指定します。

たとえば、MyGraph という名前のアプレットに対しては、これら必須の属性を以下のように定義できます。

```
%DEFINE{  
MyGraph.codebase = "/netdata-java/"  
MyGraph.height = "200"  
MyGraph.width = "400"  
%}
```

実際の指定は DEFINE セクションにある必要はありません。値は DTW_ASSIGN 関数で設定できます。AppletName.code 変数に変数を定義しない場合は、Net.Data はデフォルトの code パラメーターをアプレット・タグに追加します。code 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・タグのパラメーター: 関数呼び出しの際に 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。以下を含むパラメーターを渡すことができます。

- Net.Data 変数 (LIST 変数を含む)
- Net.Data 表
- Net.Data 表の列

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を持つ Java アプレットの PARAM タグを HTML 出力に作成します。文字列リテラルまたは関数呼び出しの結果は渡すことができません。

Net.Data の変数パラメーター:

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。変数をマクロの DEFINE ブロックで定義し、その変数の値を DTWA_AppletName 関数呼び出しで渡すと、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、次のマクロ・ステートメントが与えられたとします。

```
%define{
...
MyTitle = "This is my Title"
%}
%HTML(report){
@DTWA_MyGraph( MyTitle, ...)
%}
```

Net.Data は次のアプレット・タグ PARAM を作成します。

```
<param name = 'MyTitle' value = "This is my Title" >
```

Net.Data の表パラメーター:

Net.Data は、Java アプレット言語環境が呼び出されるたびに、DTW_NUMBER_OF_TABLES という名前の PARAM タグを生成し、関数呼び出しが何らかの表変数を渡したかどうかを指定します。値は Net.Data が関数で使用する表変数の数です。関数呼び出しに表変数が指定されていない場合は、以下のタグが生成されます。

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

関数呼び出しでは、1 つ以上の Net.Data の表変数を渡すことができます。Net.Data の表変数を DTWA_AppletName 関数呼び出しで指定すると、Net.Data は以下の PARAM タグを生成します。

表名パラメーター・タグ

このタグは渡す表の名前を指定します。このタグの構文は、以下のとおりです。

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

ここで、i は関数呼び出しの順序に基づく表の番号です。また、tname は表の名前です。

行および列の仕様パラメーター・タグ:

PARAM タグは特定の表の行と列の数を指定するために生成されます。このタグの構文は、以下のとおりです。

```
<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 は表の行の数、そしてcols は表の列の数です。このタグのペアは、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:

この PARAM タグは、特定の列の列名を指定します。このタグの構文は、以下のとおりです。

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

ここで、表名は tname、j は列番号、cname は表の列の名前です。

行値パラメーター・タグ:

この PARAM タグは、特定の行と列にある値を指定します。このタグの構文は、以下のとおりです。

```
<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* は対応する行と列の値に一致する値です。

表列パラメーター: 表列を関数呼び出し時にパラメーターとして渡すことにより、特定の列のタグを生成することができます。Net.Data は、指定された列に対してのみ対応するアプレット・タグを生成します。表列パラメーターは次の構文を使用します。

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

ここで、*x* は表の列の名前または番号です。

表列パラメーターは、表パラメーターに対して定義された同じアプレット・タグを使用します。

Java に非対応のブラウザでのアプレット・タグの代替テキスト: 変数

DTW_APPLET_ALTTEXT は、Java をサポートしていないブラウザ、あるいは Java のサポートをオフにしているブラウザに表示するテキストを指定します。たとえば、次の変数定義

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

は、以下の HTML タグとテキストを作成します。

```
<P>このブラウザは、Java に対応していません。<BR>
```

この変数が定義されていない場合は、代替テキストは表示されません。

Java アプレットの例

次の例は、Java アプレット言語環境を呼び出す Net.Data マクロと、呼び出された Java アプレット言語環境によって生成されたアプレット・タグを示しています。

Net.Data マクロには、Java アプレット言語環境への次の関数呼び出しが含まれています。

```
%define{
DATABASE = "celdial"
DTW_APPLET_ALTTEXT = "<P>Sorry, your browser is not Java-enabled."
DTW_DEFAULT_REPORT = "no"
MyGraph.codebase = "/netdata-java/"
MyGraph.height = "200"
MyGraph.width = "400"
MyTitle = "This is my Title"
%}
%FUNCTION(DTW_SQL) mySQL(OUT table){
select name, ages from ibmuser.guests
%}
%HTML(report){
@mySQL(MyTable)
@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・タブの属性を指定しています。

```
MyGraph.codebase = "/netdata-java/"
MyGraph.height = "200"
MyGraph.width = "400"
```


この言語環境は、次の限定子を持つアプレットを生成します。

```
<applet code = 'MyGraph.class' codebase = '/netdata-java/' width = '400' height = '200' >
```

Net.Data は、Net.Data マクロの SQL セクションから SQL 照会結果を出力表 MyTable に戻します。この表は次のように DEFINE セクションで指定されています。

```
MyTable = %TABLE(all)
```

マクロでのアプレットの呼び出しは、次のように HTML 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、結果表に関する情報（たとえば、戻される列と行の数、結果行など）を含む完全なアプレット・タグを生成します。Net.Data は、次の例のように、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 = "35">
```

パラメーター名 *DTW_MyTable_ages_VALUE_1* は、表 MyTable の表セル（行 1、列 ages）を指定します。このセルの値は 4 です。アプレットへの関数呼び出しの際のキーワード DTW_COLUMN は、必要なのは結果表 MyTable の列 ages だけであることを指定します。

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

以下の出力は、この例に対して Net.Data が生成する完全なアプレット・タグを示しています。

```
<applet code = 'MyGraph.class' codebase = '/netdata-java/' 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 = "35">
<param name = 'DTW_MyTable_ages_VALUE_2' value = "32">
<param name = 'DTW_MyTable_ages_VALUE_3' value = "31" >
<param name = 'DTW_MyTable_ages_VALUE_4' value = "28" >
<param name = 'DTW_MyTable_ages_VALUE_5' value = "40" >
<P>Sorry, your browser is not Java-enabled.<BR>
</applet>
```

Net.Data の Java アプレット・インターフェースの使い方

Net.Data は、DTW_Applet.class と呼ばれるインターフェース・セットを提供しています。これを Java アプレットで使用すると、表変数に対して生成される PARAM タグを処理するのに役に立ちます。このインターフェースを拡張するアプレットを作成して、アプレットからそのルーチンを呼び出すことができます。

Net.Data が提供するインターフェースには以下のものがあります。

- **int GetNumberOfTables()** は、アプレット・タグで検出された表の数を戻します。
- **String [] GetTableNames()** は、アプレット・タグで検出された表名のリストを戻します。
- **int GetNumberOfColumns(String table_name)** は、表 table_name の列の数を戻します。
- **int GetNumberOfRows(String table_name)** は、表 table_name の行の数を戻します。

- **String[] GetColumnNames(String table_name)** は、表 table_name の列の名前を返します。
- **String[][] GetTable(String table_name)** は、表の行と列の値を格納するストリングの 2 次元配列を返します。

インターフェースにアクセスするには、EXTENDS キーワードをアプレットのコードで使用して、アプレットを DTW_APPLET クラスからサブクラス化します。その例を、以下に示します。

```
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");
        }
    }
}
```

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の Perl スクリプトを解釈することができます。あるいは、サーバー上の別ファイルに保管されている外部の Perl スクリプトを処理することができます。

Perl 言語環境の構成

以下の構成ステートメントが、Net.Data の初期設定ファイルに 1 行で記述されていることを確認してください。

```
ENVIRONMENT (DTW_PERL)      perl.dll    ()
```

Net.Data の初期設定ファイルと言語環境の ENVIRONMENT ステートメントの詳細については、18ページの『環境構成ステートメント』を参照してください。

外部 Perl スクリプトの呼び出し

外部 Perl スクリプトの呼び出しは、EXEC ステートメントにより FUNCTION ブロックで識別されます。使用する構文は以下のとおりです。

```
%EXEC{ perl_script_name [optional parameters] %}
```

必須: Perl のスクリプト名 *perl_script_name* は、Net.Data の初期設定ファイルの EXEC_PATH 構成変数に対して指定されているパスにリストされていることを確認してください。

```
%FUNCTION(DTW_PERL) rexx1() {  
%EXEC{MyPerl.pl %}  
%}
```

パラメーターの渡し方

Perl によって呼び出されるプログラムに情報を渡すには、2 つの方法があります (DTW_PERL 言語環境では、直接呼び渡しと間接渡し)。

直接渡し

Perl スクリプトの呼び出し時にパラメーターを直接渡します。たとえば、以下のようになります。

```
%DEFINE INPARAM1 = "SWITCH1"  
%FUNCTION(DTW_PERL) sys1() {  
%EXEC{  
    MyPerl.pl $(INPARAM1) "literal string"  
%}  
%}
```

Net.Data 変数 INPARAM1 が参照され、Perl スクリプトに渡されます。パラメーターを Perl スクリプトに渡す方法は、Perl スクリプトがコマンド行から呼び出された場合に Perl スクリプトにパラメーターを渡すのと同じです。この方法で渡されるパラメーターは、入力タイプのパラメーターと考えられます (Perl スクリプトに渡されるパラメーターは、Perl スクリプトで使用し、操作することができますが、パラメーターの変更は Net.Data には反映されません)。

間接渡し

以下に示す方法のいずれかを使用して、Perl スクリプトの呼び出し時に間接的にパラメーターを渡します。

- Net.Data が入力パラメーターを環境変数として Perl スクリプトに渡すようにする。これにより、Perl スクリプトは環境変数からパラメーターを取得することができます。
- Perl スクリプトが出力パラメーターを言語環境に戻すようにする。そのためには、Net.Data が環境変数 DTWPIPE に格納して渡す名前を持つ、名前付きパイプにデータを書き込みます。次の構文を使用してデータを名前付きパイプに書き込みます。

```
name="value"
```

複数のデータ項目の場合、各項目を改行あるいはブランク文字で区切ります。

変数名が出力パラメーターと同じ名前を持っており、上記の構文を使用する場合は、現行値は新規の値で置き換えられます。変数名が出力パラメーターに対応しない場合は、Net.Data はその変数を無視します。

次の例は、Net.Data が変数をマクロからどのように渡すかを示しています。

```
%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.pl と呼ばれる外部ファイルにある場合は、次の例にある関数と同じ関数を作成することができます。

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

Net.Data の表は、Perl 言語環境で呼び出される Perl スクリプトに渡すことができます。Perl スクリプトは、Net.Data 名によって Net.Data のマクロの表パラメーターの値にアクセスします。列見出しとフィールド値は、表名と列番号で識別される変数に格納されます。たとえば、テーブル myTable では、列見出しは myTable_N_j で、フィールド値は myTable_V_i_j です。ここで、i は行番号、j は列番号です。この表の行と列の数は myTable_ROWS と myTable_COLS です。

FUNCTION セクションの REPORT ブロックと MESSAGE ブロック

REPORT および MESSAGE ブロックは、どの FUNCTION セクションにおいても同じように使用することができます。これらのブロックは Net.Data によって処理され、言語環境では処理されません。ただし、Perl スクリプトはテキストを標準出力ストリームに書き込み、Web ページの一部として組み込むことはできます。

Perl 言語環境の例

次の例は、Net.Data が外部 Perl スクリプトを実行してどのように表を生成するかを示しています。

```
%define {
    c = %TABLE(20)
    rows = "5"
    columns = "5" %}
%function(DTW_PERL) genTable(in rows, in columns, out table) {
    %exec{ perl.pl
}%
%message{
    default: "genTable: Unexpected Error"
}%
}%
%HTML(REPORT) {
```

```

@genTable(rows, columns, c)
return code is $(RETURN_CODE)
%}
The Perl script (perl.pl):
open(D2W,"> $ENV{DTWPIPE}");
print "genTable begins ...
";
$r = $ENV{ROWS};
$c = $ENV{COLUMNS};
print D2W "table_ROWS=\"$r\" ";
print D2W "table_COLS=\"$c\" ";
print "rows: $r
";
print "columns: $c";
for ($j=1; $j<=$c; $j++)
{
print D2W "table_N_$j=\"COL$j\" ";
}
for ($i=1; $i<=$r; $i++)
{
for ($j=1; $j<=$c; $j++)
{
print D2W "table_V_$i","_","$j=\"| $i $j |\" ";
}
}
close(D2W);

```

結果: genTable は以下を生成します。

```

rows: 5 columns: 5
COL1 | COL2 | COL3 | COL4 | COL5 |
-----
[ 1 1 ] | [ 1 2 ] | [ 1 3 ] | [ 1 4 ] | [ 1 5 ] |
-----
[ 2 1 ] | [ 2 2 ] | [ 2 3 ] | [ 2 4 ] | [ 2 5 ] |
-----
[ 3 1 ] | [ 3 2 ] | [ 3 3 ] | [ 3 4 ] | [ 3 5 ] |
-----
[ 4 1 ] | [ 4 2 ] | [ 4 3 ] | [ 4 4 ] | [ 4 5 ] |
-----
[ 5 1 ] | [ 5 2 ] | [ 5 3 ] | [ 5 4 ] | [ 5 5 ] |
-----
return code is 0

```

REXX 言語環境

REXX 言語環境は、インライン REXX プログラム (Net.Data マクロの FUNCTION ブロックに指定) をインタープリット、または別個のファイルに保管された外部 REXX プログラムを実行することができます。

REXX 言語環境の構成

REXX 言語環境を使用するには、Net.Data 初期設定を検証し、言語環境を設定する必要があります。

以下の構成ステートメントが、初期設定ファイルに 1 行で記述されていることを確認します。

```
ENVIRONMENT (DTW_REXX)      rexxd11  ()
```

Net.Data 初期設定ファイル、および言語環境の ENVIRONMENT ステートメントの詳細は、18ページの『環境構成ステートメント』を参照してください。

外部 REXX プログラムの呼び出し

外部 REXX プログラムを呼び出すには、以下のフォーマットで FUNCTION ブロックを使用します。

```
%EXEC{ REXX_file_name [optional  
parameters] %}
```

必要: *REXX_script_name* (REXX ファイル名) が、Net.Data 初期設定ファイルの EXEC_PATH 構成変数に指定したパスにリストされていることを確認します。

たとえば、以下のようにします。

```
%FUNCTION(DTW_REXX) rexx1() {  
%EXEC{REXX.EXE %}  
%}
```

パラメーターを渡す

REXX (DTW_REXX) 言語環境により起動する REXX プログラムに情報を渡すには、直接および間接の 2 とおりの方法があります。

直接 %EXEC ステートメントを使用して、外部 REXX プログラムに直接パラメーターを渡します。たとえば、以下のようにします。

```
%FUNCTION(DTW_REXX) rexx1() {  
%EXEC{  
CALL1.CMD $(INPARM) "literal string" %}  
%}
```

Net.Data 変数 INPARM1 がデリファレンスされ、外部 REXX プログラムに渡されます。REXX プログラムは REXX PARSE ARG 命令を使用して変数を参照することができます。このメソッドを使用してプログラムに渡されるパラメーターは、入力型パラメーターとみなされます (プログラムに渡されるパラメーターは、そのプログラムにより使用して操作できますが、パラメーターに対する変更は Net.Data に反映されません)。

間接

REXX プログラムの 変数プール 経由でパラメーターを間接的に渡します。REXX プログラムが開始すると、REXX インタープリターによりすべての変数に関する情報を含むスペースが作成され保守されます。このスペースは、変数プールと呼ばれます。

REXX 言語環境 (DTW_REXX) 関数が呼び出されると、REXX プログラムの実行の前に、入力 (IN) または入出力 (INOUT) となる関数仮引き数が REXX 言語環境により変数プールに保管されます。REXX プログラムが起動すると、これはこれらの変数に直接アクセスすることができます。REXX プログラムが正常に終了すると、DTW_REXX 言語環境は、OUT または INOUT 関数仮引き数があるかどうかを判別します。ある場合には、言語環境は関数仮引き数に対応する値を変数プールから検索し、関数仮引き数の値を新規値に更新します。Net.Data がコントロールを受け取ると、すべての OUT または INOUT パラメーターが REXX 言語環境から取得された新規値で更新されます。たとえば、以下のようにします。

```
%DEFINE a = "3"  
%DEFINE b = "0"  
%FUNCTION(DTW_REXX) double_func(IN inp1, OUT outp1){
```

```

        outp1 = 2*inp1
    %}

%HTML(REPORT) {
Value of b is $(b), @double_func(a, b) Value of b is $(b)
%}

```

上記の例では、呼び出し `@double_func` は 2 つのパラメーター *a* および *b* を渡します。REXX 関数 `double_func` は最初のパラメーターを 2 倍し、その結果を 2 番目のパラメーターに保管します。Net.Data がマクロを起動すると、*b* は値 6 をとります。

Net.Data テーブルを REXX プログラムに渡すことができます。REXX プログラムは、Net.Data マクロ・テーブル・パラメーターの値を REXX ステム変数としてアクセスします。REXX プログラムに対しては、列見出しおよびフィールド値は、テーブル名および列番号により識別される変数に含まれます。たとえば、テーブル `myTable` 中、列見出しは `myTable_N.j`、フィールド値は `myTable_N.i.j` で、*i* が行番号、*j* が列番号です。テーブル中の行番号は `myTable_ROWS`、列番号は `myTable_COLS` です。

REXX 言語環境の例

以下の例では、REXX 関数を呼び出して 2 つの列および 3 つの行を持つ Net.Data テーブルを生成するマクロを示しています。REXX 関数への呼び出しに続き、ブラウザーに送り返す HTML テーブルを生成するための組み込み関数 `DTW_TB_TABLE()` が呼び出されます。

```

%DEFINE myTable = %TABLE
%DEFINE DTW_DEFAULT_REPORT = "NO"

%FUNCTION(DTW REXX) genTable(out out_table) {
    out_table_ROWS = 3
    out_table_COLS = 2

    /* Set Column Headings */
    do j=1 to out_table_COLS
        out_table_N.j = 'COL'j
    end

    /* Set the fields in the row */
    do i = 1 to out_table_ROWS
        do j = 1 to out_table_COLS
            out_table_V.i.j = '[' i j ']'
        end
    end
%}

%HTML(REPORT) {
    @genTable(myTable)
    @DTW_TB_TABLE(myTable)
%}

```

結果

COL1	COL2
[1 1]	[1 2]
[2 1]	[2 2]
[3 1]	[3 2]

システム言語環境

システム言語環境は、C/C++ および COBOL などの外部プログラムへの呼び出しをサポートします。これらは FUNCTION ブロックの EXEC ステートメントで識別されます。システム言語環境は、指定されたプログラム名、またはコマンドおよびパラメーターを実行するようオペレーティング・システムに渡すことにより EXEC ステートメントをインタープリットします。

システム言語環境の構成

以下の構成ステートメントを初期設定ファイルに 1 行で追加します。

```
ENVIRONMENT (DTW_SYSTEM) sysd11  ()
```

Net.Data 初期設定ファイル、および言語環境の ENVIRONMENT ステートメントの詳細は、18ページの『環境構成ステートメント』を参照してください。

パラメーターを渡す

システム (DTW_SYSTEM) 言語環境により起動するプログラムに情報を渡すには、直接および間接の 2 とおりの方法があります。

直接 プログラムへの呼び出しの際、パラメーターが直接渡されます。たとえば、以下のようにします。

```
%DEFINE INPARAM1 = "SWITCH1"

%FUNCTION(DTW_SYSTEM) sys1() {
  %EXEC{
    CALL1.CMD $(INPARAM1) "literal string"
  %}
%}
```

Net.Data 変数 INPARAM1 が参照され、プログラムに渡されます。パラメーターはプログラムに渡されます。これはコマンド行からプログラムを呼び出す場合と同様に行われます。このメソッドを使用してプログラムに渡されるパラメーターは、入力型パラメーターとみなされます (プログラムに渡されるパラメーターは、そのプログラムにより使用して操作できますが、パラメーターに対する変更は Net.Data に反映されません)。

間接

システム言語環境は Net.Data 変数を直接に渡すまたは検索することができないため、以下のようにしてプログラムに対し変数を使用可能にします。

- Net.Data は入力パラメーターを環境変数としてプログラムに渡します。するとプログラムは環境変数を通じてパラメーターを検索します。
- プログラムは、名前付きパイプ (名前は Net.Data が環境変数 DTWPIPE で渡します) に書き込みを行い、出力パラメーターを言語環境に渡します。以下の構文を使用して、名前付きパイプにデータを書き込みます。

```
name="value"
```

複数のデータ項目については、項目を改行またはブランク文字により区切ります。

変数名が出力パラメーターと同じ名前上記の構文を使用する場合、現行値は新規値により置き換えられます。変数名が出力パラメーターに対応しなければ、Net.Data はこれを無視します。

以下の例では、Net.Data がマクロの変数を渡す方法を示しています。

```
%FUNCTION(DTW_SYSTEM) sys1 (IN P1, OUT P2, P3) {  
  %EXEC {  
    UPDPGM  
  }  
}
```

Net.Data テーブルをシステム言語環境が呼び出すプログラムに渡すことができます。プログラムは、Net.Data マクロ・テーブル・パラメーターの値をその Net.Data 名によりアクセスします。列見出しおよびフィールド値は、テーブル名および列番号により識別される変数に含まれます。たとえば、テーブル myTable 中、列見出しは myTable_N_j、フィールド値は myTable_V_i_j で、i が行番号、j が列番号です。テーブルの行および列番号は、myTable_ROWS および myTable_COLS です。

プロセスに対する環境変数の数には制限があるため、多くの行を持つテーブルを渡すことはお勧めできません。

システム言語環境の例

以下の例では、3つのパラメーター P1、P2、および P3 を持つ関数定義を含むマクロを示しています。P1 は入力 (IN) パラメーター、P2 および P3 は出力 (OUT) パラメーターです。関数はプログラム UPDPGM を起動し、これによりパラメーター P2 が P1 の値で更新され、P3 を文字ストリングに設定します。%EXEC ブロックのステートメントを処理する前に、DTW_SYSTEM 言語環境は P1 および対応する値を環境スペースに保管します。

```
%DEFINE {  
  MYPARM2 = "ValueOfParm2"  
  MYPARM3 = "ValueOfParm3"  
}  
%FUNCTION(DTW_SYSTEM) sys1 (IN P1, OUT P2, P3) {  
  %EXEC {  
    UPDPGM  
  }  
}  
  
%HTML(upd1) {  
<P>  
Passing data to a program. The current value  
of MYPARM2 is "$(MYPARM2)", and the current value of MYPARM3 is  
"$ (MYPARM3)". Now we invoke the Web macro function.  
  
@sys1("ValueOfParm1", MYPARM2, MYPARM3)  
  
<P>  
After the function call, the value of MYPARM2 is "$(MYPARM2)",  
and the value of MYPARM3 is "$(MYPARM3)".  
}
```

第7章 パフォーマンスを向上させる

パフォーマンスを向上させることは、システム・チューニングの重要な部分です。本章では、Net.Data のパフォーマンスを向上させるための戦略について説明します。本章で説明するトピックは、以下の通りです。

- 『Web サーバー API の使い方』
- 『Net.Data マクロ・キャッシング』
- 119ページの『DB2 (OS/390 版) のメッセージを抑制する』
- 120ページの『言語環境の最適化』

さらに、Web サーバーが正しく調整されていることを確認してください。Web サーバーのパフォーマンスは、Net.Data がマクロや直接要求を処理する速度には関係なく、応答時間に直接影響を与えます。

Web サーバー API の使い方

CGI を使わず、ICAPI あるいは GWAPI などの Web サーバー API を使って Net.Data を起動すると、パフォーマンスを向上させることができます。Net.Data が Web サーバー API を使用して実行されている場合は、Net.Data は Web サーバーのプロセス内でスレッドとして実行されます。Web サーバーの処理はマルチスレッド化されているので、複数の Net.Data 要求を同一のアドレス空間内で同時に処理することができます。そのため、Net.Data を CGI プロセスとして起動する場合のオーバーヘッドがなくなります。

考慮事項: Web サーバー API を使用すると、アプリケーションを分離せずにパフォーマンスを向上させることができます。Net.Data はマルチスレッド環境で実行されるため、ユーザー作成の言語環境、不適切な呼び出し、またはデータベースの故障などによるエラーは、Web サーバーでの問題発生の原因となり、サーバーをダウンさせることもあります。Web サーバー API の 1 つを使用するかどうかを決定する場合は、アプリケーションがパフォーマンスを重視するか、アプリケーションの分離を重視するかによって決定します。

Net.Data マクロ・キャッシング

マクロ・キャッシングを使用することによりスループットの向上および CPU 使用率の減少が図れます。マクロ・キャッシングを使用可能にすると、最初にマクロを起動する際、プリプロセスされたマクロがメモリーにキャッシュされます。これらのプリプロセスされたバージョンは再利用が可能となり、これにより HFS からのマクロの読み取り、および要求ごとに行われる処理にかかる負担が取り除かれます。

ガイドラインおよび制約事項

マクロ・キャッシングに関連して以下の項目に注意してください。

- キャッシングは ICAPI、GWAPI、または Net.Data サブレットを使用する際に使用可能となります。

- キャッシュされたマクロのバージョンは、マクロを含むファイルへの読み取り許可を持つ要求発行者に対して使用可能となります。
- プリプロセスされたマクロのバージョンが使用するメモリー量は、マクロ・ファイルのおよそ倍のサイズを必要とします。
- マクロ・キャッシングに使用するメモリー量は、キャッシング・コンフィギュレーション変数を使用してコントロールできます。これらのコンフィギュレーション変数により、キャッシュするマクロを指定します。

マクロ・キャッシングの使用可能化

マクロ・キャッシングは、キャッシング・コンフィギュレーション変数を `Net.Data` 初期設定ファイル (`db2www.ini`) に追加して使用可能にします。`DTW_CACHE_MACRO` 変数、または `DTW_DO_NOT_CACHE_MACRO` 変数 (あるいはその両方) を `Net.Data` 初期設定ファイルに追加すると、キャッシングが使用可能になります。いずれの変数も追加しない場合は、マクロはキャッシュされません。

`DTW_CACHE_MACRO` および `DTW_DO_NOT_CACHE` コンフィギュレーション指定の両方が同じマクロを指定している場合は、`Net.Data` はマクロをキャッシュしません。

キャッシュするマクロの定義

`DTW_MACRO_CACHE` コンフィギュレーション変数は、キャッシュするマクロを指定します。

このコンフィギュレーション変数を `Net.Data` 初期設定ファイルに設定します。

構文：

```
DTW_CACHE_MACRO [=] filename_or_pathtemplate;...
```

ここで `filename_or_pathtemplate` は以下のいずれかです。

- マクロ・ファイルの完全修飾名。
- パス・テンプレート。ディレクトリー・パスに `/*` を続けます。パス・テンプレートを使用する場合、ディレクトリーおよびそのサブディレクトリー内のすべてのマクロがキャッシュされます。

例 1: `/u/user1/macros` およびそのサブディレクトリー内のすべてのマクロをキャッシュするには、以下のようにコンフィギュレーション変数を設定します。

```
DTW_CACHE_MACRO /u/user1/macros/*
```

例 2: `DIR1` および `DIR2` ディレクトリー内のすべてのマクロ、および別個のマクロ `sql.dtw` をキャッシュするには、`DTW_CACHE_MACRO` パスは以下ようになります。

```
DTW_CACHE_MACRO /u/user1/macros/DIR1/*;/u/user2/macros/sql.dtw;/u/user2/macros/DIR2/*
```

キャッシュしないマクロの定義

`DTW_DO_NOT_CACHE_MACRO` コンフィギュレーション変数は、キャッシュしないマクロを指定します。

このコンフィギュレーション変数を Net.Data 初期設定ファイルに設定します。Net.Data 初期設定ファイルにこの変数が含まれ、DTW_CACHE_MACRO 変数が含まれない場合は、DTW_DO_NOT_CACHE_MACRO 変数にリストされたマクロを除くすべてのマクロがキャッシュされます。

構文：

```
DTW_DO_NOT_CACHE_MACRO [=] filename_or_pathtemplate;...
```

ここで *filename_or_pathtemplate* は以下のいずれかです。

- マクロ・ファイルの完全修飾名。
- パス・テンプレート。ディレクトリー・パスに /* を続けます。パス・テンプレートを使用する場合、ディレクトリーまたはそのサブディレクトリー内のマクロはキャッシュされません。

例 1: adminset.d2w マクロを除くすべてのマクロをキャッシュするには、以下のよう
にコンフィギュレーション変数を設定します。

```
DTW_DO_NOT_CACHE_MACRO /u/user1/macros/adminset.d2w
```

例 2: 両方のコンフィギュレーション変数を初期設定ファイルに設定した場合は、
DTW_DO_NOT_CACHE_MACRO が優先されます。たとえば、変数の設定が以下の場
合を考えてみます。

```
DTW_CACHE_MACRO /u/user1/user_macros/*;/u/user1/admin_macros/*  
DTW_DO_NOT_CACHE_MACRO /u/user1/admin_macros/adminset.d2w
```

user_macros および admin_macros ディレクトリー内のマクロは、マクロ adminset.d2w
を除いてキャッシュされます。このマクロは admin_macros ディレクトリーにありま
すが、キャッシュされません。DTW_DO_NOT_CACHE_MACRO の設定が
DTW_CACHE_MACRO の設定をオーバーライドするためです。

DB2 (OS/390 版) のメッセージを抑制する

SQL 言語環境を使用している場合は、DB2 のメッセージを非ゼロ SQLCODE コード
から抑制することにより、Net.Data for OS/390 のパフォーマンスを向上することがで
きます。DB2MSGs 構成変数を使用して、アプリケーションに必要なメッセージ・レ
ベルを指定します。実稼働環境では、DB2MSGs を NONE に設定して、DB2 のメッ
セージ検索をバイパスすることができます。DB2MSGs が NONE または
ERRORONLY に設定されている場合は、MESSAGE ブロックを使用して、非ゼロの
SQLCODE をマクロ内でキャッチすることができます。ユーザー・マクロで
MESSAGE ブロックを使用する方法を理解するには、*Net.Data 解説書* を参照してく
ださい。

メッセージ・レベルを指定するには、Net.Data の初期設定ファイルの DB2MSGs 構
成変数を使用します。

指定可能な値:

```
DB2MSGs [=] message_level
```

message_level は Net.Data が提供する DB2 メッセージのレベルを指示します。次の
ように指定することができます。

NONE	Net.Data はメッセージを出力しないことを指定します
ERRORONLY	Net.Data が負の SQLCODE 値に対するメッセージだけを出力するように指定します。
ALL	Net.Data がすべての SQLCODE 値に対するメッセージを出力するように指定します。これはデフォルトです。上記にリストされた有効な値以外を DB2MSGSG に指定した場合には、Net.Data はデフォルトで ALL を使用します。

言語環境の最適化

以下のセクションでは、Net.Data 提供の言語環境の使用時にパフォーマンスを向上させるための手法について説明します。

- 『REXX 言語環境』
- 『SQL 言語環境』
- 121ページの『システムおよび Perl 言語環境』

REXX 言語環境

以下のヒントを使用して Net.Data アプリケーションのパフォーマンスを向上させてください。

- REXX プログラムを可能な限り結合する。サイズの大きなプログラムでもその数が少ないほど、小さなプログラムが数多くある場合よりもパフォーマンスは向上します。それは、REXX LE 関数がマクロで呼び出されるたびに、REXX インタープリターが初期化されるからです。
- 外部 REXX プログラムの場合は、%EXEC ステートメントのコマンド行ではグローバル変数を参照する。
- 入力専用パラメーターは、グローバルな Net.Data 変数を定義し、その変数を参照することにより、直接 REXX プログラムに渡す。インライン REXX プログラムの場合は、グローバル変数を直接 REXX ソースで参照する。

SQL 言語環境

以下のセクションでは、SQL 言語環境に関するパフォーマンス手法をいくつか説明します。DB2 のパフォーマンスの考慮事項について理解するには、Web サイト <http://review.software.ibm.com/data/db2/performance> にアクセスしてください。(英語版)

SQL 言語環境の手法

パフォーマンスを向上させるには、以下の SQL 言語環境手法を使用します。

- START_ROW_NUM および RPT_MAX_ROWS Net.Data 変数を使用して、戻される表のサイズを小さくする。結果セットに大量の行が含まれている場合は、START_ROW_NUM および RPT_MAX_ROWS を使用して、ブラウザーに戻される結果セットのサブセットを指定することができます。START_ROW_NUM は、戻す最初の行の行番号を指定します。RPT_MAX_ROWS は戻す行数を指定します。

重要: カーソル位置が要求間で保持されないため、Net.Data は要求ごとに照会を再発行します。

- 静的な SQL を使用するストアド・プロシージャの呼び出しを考慮してみる。動的SQL は実行時に作成されるのに対して、静的 SQL はプリコンパイル段階で作成されます。SQL 言語環境は動的 SQL を使用します。これにより、プログラムの実行時に SQL ステートメントが実行されます。ステートメントの準備にはその処理時間がさらに必要になるため、静的な SQL の方がより効率的です。

システムおよび Perl 言語環境

入力専用パラメーターは、システムまたは Perl 言語環境が呼び出すプログラムに直接渡してください。それには、グローバルな Net.Data 変数を定義し、それを参照します。外部のプログラムおよび Perl スクリプトの場合、%EXEC ステートメントのコマンド行でその変数を参照します。インラインの Perl スクリプトの場合は、その変数を Perl のソースで直接参照します。

付録A. 参考文献

本セクションでは、本書中の参考文献をリストしています。

- 『Net.Data テクニカル・ライブラリー』
- 『関連資料』

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、以下の Net.Data Web サイトで入手できます。

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

文書	説明
<ul style="list-style-type: none">• <i>Net.Data</i> 管理およびプログラミングの手引き OS/390 版• <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	Net.Data のインストール、構成、および起動に関する概念およびタスクについての情報が含まれています。また、Net.Data マクロの作成、Net.Data パフォーマンス向上の手法、Net.Data 言語環境の使用、接続管理、および Net.Data ログおよびトレースを使用した問題解決およびパフォーマンス・チューニングに関する説明がされています。
<i>Net.Data</i> 解説書	Net.Data マクロ言語、変数、および組み込み関数について説明します。
<i>Net.Data</i> 言語環境解説書	Net.Data 言語環境インターフェースについて説明します。
<i>Net.Data</i> メッセージ	Net.Data エラー・メッセージおよびリターン・コードのリストです。
<i>Program Directory for Net.Data for OS/390 Version 2 Release 2</i>	SMP/E インストールおよび Net.Data for OS/390 の構成について説明します。

関連資料

Net.Data および関連製品を使用するにあたり、以下の文書が利用できます。

- *Accessing DB2 for OS/390 Data from the World Wide Web*, SG24-5273, Maria Sueli Almeida, Charles E. Lewis, Uwe Sager, Pilar Sandoval
- *ICSS for OS/390 概説およびインストール V2R2*, GC88-7949
- *ICSS OS/390: Webmaster の手引き V2R2*, GC88-7534
- *Go Webserver 概説およびインストール V4R6*, SC88-7825
- *Go Webserver for OS/390 Webmaster の手引き リリース 4.6.1*, SD88-7826
- *OS/390 MVS 計画: ワークロード管理サービス*, GC88-6582

付録B. Net.Data for OS/390 を構成し、DataJoiner をアクセスする

Net.Data for OS/390 と DataJoiner を使用して、DB2/6000、Oracle、および Sybase などのリモート・データベースをアクセスすることができます。このセクションでは、DataJoiner for AIX バージョン 1.2 PTF U447593 または DataJoiner for HP-UX バージョン 1.1 を使用して、システムの構成方法を説明しています。

構成の各ステップ:

1. DataJoiner との遠隔通信に、通信データベース (CDB) で必要な情報を入力します。CDB 内の情報は、*DB2 Installation Guide* に記述されています。
2. DataJoiner がインストールされているリモート・ロケーションに、BIND PACKAGE コマンドを使用して、Net.Data DBRM をバインドします。
3. BIND PLAN コマンドを使用して、Net.Data DBRM を DB2 にバインドします。PKLIST オプションを使用して、リモート・ロケーションで作成されたパッケージを組み込みます。
4. Net.Data 初期設定ファイルを変更し、SQL 関数への入力変数として LOCATION 変数を指定します。このファイルは Web サーバーのドキュメント・ルート・ディレクトリにあります。新規の DTW_SQL 環境ステートメントは以下のようになります。

```
ENVIRONMENT (DTW_SQL) dtwsq1 (IN LOCATION)
```

DataJoiner を使用してリモート・データをアクセスする Net.Data マクロは、LOCATION に値を指定する必要があります。次の Net.Data マクロの例では、DataJoiner を使用してリモート・データベースを照会します。

```
%{ ***** Define Block ***** %}
%DEFINE {
    DB2SSID="NDA1"
    LOCATION="QMFDJ00"
    DTW_DEFAULT_REPORT="YES"
}%

%{ ***** Function Definition Block ***** %}
%FUNCTION(DTW_SQL) selectall() {
    SELECT * FROM $(tabnam)
}%

%{ ***** HTML Block: Table_Input ***** %}
%HTML(Table_Input) {
<Title>DJ Test #1</title>
<Body>
<h1 align=center>Table Selection</h1>
<br>
<form method="post" action="Column_Output">
<p>Enter Table Name: <input type="text" name="tabnam"></p>
<p><input type="submit"></p>
</form>
</Body>
}%

%{ ***** HTML Block: Column_Output ***** %}
%HTML(Column_Output) {
<Title>DJ Test #1</title>
```

```
<Body>  
@selectall()  
</Body>  
%}
```

付録C. Net.Data サンプル・マクロ

このサンプル・マクロ・アプリケーションでは、リストで社員の名前を選択して個々の社員の追加情報を得られるアプリケーションから、社員名のリストを表示します。このマクロは、SQL 言語環境を使用して、社員名および特定の社員に関する情報の EMPLOYEE 表を照会します。

```

%{***** Sample Macro *****)
*   FileName = sqlsamp1.d2w
*   Description:
*       This Net.Data macro queries...
*       - The EMPLOYEE table to create a selection list of
*         employees for display at a browser
*       - The EMPLOYEE table to obtain additional information
*         about an individual employee
*
*****%}
%{*****
*   Include for global DEFINES -
*****%}
%INCLUDE "sqlsamp1.hti"
%{*****
*   Function: queryDB           Language Environment: SQL
*   Description: Queries the table designated by the variable myTable and
*   creates a selection list from the result. The value of the variable
*   myTable is specified in the include file sqlsamp1.hti.
*****%}
%FUNCTION(DTW_SQL) queryDB() {
    SELECT * FROM $(myTable)
%MESSAGE {
    -204: {<p><b>ERROR -204: Table $(myTable) not found. </b>
        <p>Be sure the correct include file is being used.</b>
        %} : exit
    +default: "WARNING $(RETURN_CODE)" : continue
    -default: "Unexpected ERROR $(RETURN_CODE)" : exit
%}

%REPORT {
<select name=emp_name>
%ROW{
<option>$(V2)
%}
</select>
%}
%}

%{*****
*   Function: fname           Language Environment: SQL
*   Description: Queries the table designated by the variable myTable for
*   additional information about the employee identified by the
*   variable emp_name.
*****%}
%FUNCTION(DTW_SQL) fname(){
SELECT EMPNME, PHONENO, JOB FROM $(myTable) WHERE EMPNME='$(emp_name)'
%MESSAGE {
    -204: "Error -204: Table not found "
    -104: "Error -104: Syntax error"
    100: "Warning 100: No records" : continue
    +default: "Warning $(RETURN_CODE)" : continue
    -default: "Unexpected SQL error" : exit
%}
%}

```

```
%{ *****
* HTML block: INPUT Title: Dynamic Query Selection *
*
* Description: Queries the EMPLOYEE table to create a selection list of
* the employees for display at the browser *
*****%}
%HTML(INPUT){
<html>
<head>
<title>Generate Employee Selection List</title>
</head>
<body>
<h3>$(exampleTitle)</h3>
<p>This example queries a table and uses the result to create
a selection list using a <em>%REPORT</em> block.
< hr>
<form method="post" action="report">
@queryDB(<input type="submit" value="Select Employee">
</form>
< hr>
</body>
</html>
%}
```

```
%{ *****
* HTML block: REPORT *
* Description: Queries the EMPLOYEE table to obtain additional information *
* about an individual employee *
*****%}
%HTML(REPORT) {
<html>
<head>
<title>Obtain Employee Information</title>
</head>
<body>
<h3>You selected employee name = $(emp_name)</h3>
<p>Here is the information for that employee:
<PRE>
@fname()
</PRE>
<hr><a href="input">Return to previous page</a>
</body>
</html>
%}
```

```
%{ End of Net.Data macro 1 %}
=====
%{ ***** Include File *****
* FileName = sqlsamp1.hti *
* Description: *
* This include file provides global DEFINES for the sqlsamp1.d2w *
* Net.Data macro. *
*****%}
%define {
emp_name = ""
exampleTitle = "Sample Macro"
myTable = "MRZ.EMPLOYEE"
%}

%{ End of include file %}
```


付録D. 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

本書で参照される IBM 以外の Web サイトは、参考として掲載するものです。これらの Web サイトは IBM 社が認定しているものではありません。これら Web サイトの資料は、本 IBM 製品の資料の一部ではありません。これらの Web サイトの利用は、ユーザーの責任で行ってください。

(i) 独自に作成されたプログラムおよび (本プログラムを含む) その他のプログラムの間の情報交換、および (ii) 交換された情報の相互利用、を目的に、本プログラムの情報の入手を希望するライセンス所有者は、次の窓口までご連絡ください。

IBM Corporation

W92/H3

555 Bailey Avenue

P.O. Box 49023

San Jose, CA 95161-9023

U.S.A.

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」または「IBM 海外使用契約書」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。

IBM の今後の方向性および予定に関する記述は、目標や目的を表明したものであり、予告なく変更または取り消しされることがあります。

この情報は、今後の計画の参考としてのみお取り扱いください。ここで記述された製品に関する情報は、製品化の前に変更されることがあります。

ここでの情報は、日々の業務で使用されるデータやレポートの例を含んでいます。これらの例は、なるべく実情に合うように、個人名、会社名、ブランド名、および製品名が使用されています。これらの名前はすべてフィクションであり、実際のビジネスの世界で使用されている名前と、偶発的に類似していることがあります。

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

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

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

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

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

2 個のアスタリスク (**) で示されている他社名、製品名およびサービス名は、他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のレベル、すなわち「ルート」ディレクトリー (これはスラッシュ (/) あるいは逆スラッシュ (\) で識別される) 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、CGI プロセスよりもパフォーマンスを向上させるために 3 つの Web サーバー API をサポートしている。

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 対応ブラウザで動作し、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 CGI プロセスよりもパフォーマンスを向上させるために、ICAPI および GWAPI という Web サーバーのプロプラエタリー API をサポートしている。

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

コミットメント制御 (commitment control). Net.Data が実行されているプロセスで、資源に対する操作が作業単位の一部であるプロセス内の境界の確立。

コモン・ゲートウェイ・インターフェース (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。

現行作業ディレクトリー (current working directory). すべての相対パス名を解決するための基準となる、プロセスのデフォルト・ディレクトリー。

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

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

データ型 (data type). 列およびリテラルの属性。

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

Domino Go Web サーバー (Domino Go Web server). 標準接続およびセキュア接続の両方を提供する、Lotus Corp. と IBM が提供する Web サーバー。ICAPI および GWAPI は、このサーバーで提供されるインターフェースである。

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

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。

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)。を参照してください。

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。

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。

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。これ

は、ネットワークを介した、クライアントのアプリケーション・プログラムとサーバー間の対話を管理する。

ヌル (null). 情報が無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. システムに対して、オブジェクトの検索経路を指示する。パス名は、連続したディレクトリー名に続けてオブジェクト名を指定して表す。個々のディレクトリー名とオブジェクト名は、スラッシュ (/) または円記号 (¥) 文字で区切る。

Perl. 解釈済みプログラミング言語。

永続性 (persistence). トランザクション全体に対して、割り当てられた値を保持している状態。この場合トランザクションは複数の Net.Data 呼び出しにまたがる。永続性を持つのは変数だけである。さらに、コミットメント制御によって影響を受けた資源に対する操作は、コミットまたはロールバックを明示的に行う、あるいはトランザクションが完了するまで活動状態に保たれる。

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). スtringの保管および取得ができるリポジトリー。

相対パス名 (relative path name). 最上位レベル、すなわち「ルート」ディレクトリーから開始しないパス名。システムは、パス名はプロセスの現行作業ディレクトリーから開始するものと想定する。

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). Net.Data の 一度の起動。永続的 Net.Data が使用される場合は、トランザクションは複数の Net.Data 起動にまたがることできる。

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

URL. Uniform resource locator (URL)。

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

作業単位 (unit of work). 1 つのアトミック操作として扱われる回復可能な連続したの操作。作業単位内のすべての操作は、操作があたかも単一の操作であるかのように完了する (コミットする)、あるいは取り消す (ロールバックする) ことができる。コミットメント制御によって影響を受ける資源に対する操作だけが、コミットあるいはロールバック可能である。

Web サーバー (Web server). インターネット接続 (Internet Connection) などの HTTP サーバーの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]

アクセス権、Net.Data のファイルへの指定 29
暗号化、ネットワーク 32
インストール
 Net.Data 5
隠蔽変数
 資産の保護 33

[カ行]

開始、Net.Data の 39
隠し変数
 変数名を隠す 65
各種変数 67
型、変数 62
環境変数 63
関数 94
 説明 70
 定義 70
 ユーザー定義 70
 呼び出し 75
 呼び出し、ストアード・プロシージャの 94
 FUNCTION ブロックの構文 70
 MACRO_FUNCTION ブロックの構文 70
関数の呼び出し 75
起動、Net.Data の
 概説 39
 構文 40
 直接要求 39
 マクロ要求 39
 マクロを使用しない 43
 リンク 47
 CGI の使用 39
 HTML ブロック 79
 URL 48
機密保護
 概説 31
 許可 33
 指定、アクセス権の 29
 認証 32
 ネットワーク暗号化 32
 ファイアウォール 31
 Net.Data のメカニズム 33

許可
 機密保護 33
 指定、Net.Data のファイルへのアクセス権の 29
空白文字、変数、不要文字を削除するための 13
グローバルな識別子の効力範囲 59
結果セット 96, 97, 98
 処理、ストアード・プロシージャ 96
 単一 97
 複数、ガイドラインおよび制約事項 84
 複数の 98
結果セットの処理、ストアード・プロシージャ 96
言語環境 89
 構成、初期設定ファイル内の 18
 セットアップ 20
 フラット・ファイル・インターフェース 101
 変数 69
 要約 90
 例 18
 ENVIRONMENT ステートメントの構成 18
 IMS Web 102
 Java アプレット 103
 ODBC 91
 Perl 109
向上、パフォーマンスの 117
構成、DataJoiner 用 125
構成、Net.Data の
 概説 5
 言語環境のセットアップ 20
 作業負荷管理プログラム (WLM) 23
 使用、ICAPI および GWAPI と 25
 初期設定ファイル
 更新 6
 構成変数ステートメント 8
 説明 5
 パス・ステートメント 15
 ENVIRONMENT ステートメント 18
接続管理 23
メッセージ・カタログ 28
CGI のための 24
Java サブレットと一緒に使用するための 27
Net.Dataのファイルおよびデータ・セットへのアクセス権 29
構成変数ステートメント
 構成、初期設定ファイル内の 8
 説明 8
 DB2MSG 9
 DB2PLAN 9
 DB2SSID 10
 DefaultDBCp 10

構成変数ステートメント (続き)

DSNAOINI 8

DTW_DIRECT_REQUEST 12

DTW_MBMODE 13

DTW_REMOVE_WS 13

DTW_SHOWSQL 14

効力範囲

識別子 59

REPORT ブロック 60

効力範囲、識別子の

グローバルな 59

マクロ 59

FUNCTION ブロック 59

ROW ブロック 60

コモン・ゲートウェイ・インターフェース (Common Gateway Interface)。CGI を参照 24

[サ行]

サブレット

構成、Net.Data の 27

サンプル・マクロ 127

識別子

効力範囲 59

実行可能な変数 64

使用可能にする、直接要求変数

(DTW_DIRECT_REQUEST) 12

条件付き

変数 63

論理、IF ブロック 85

使用不可にする、直接要求変数

(DTW_DIRECT_REQUEST) 12

初期設定ファイル

更新 6

構成変数ステートメント 8

説明 5

パス・ステートメント 15

フォーマット 6

ENVIRONMENT ステートメント 18

ストアード・プロシージャ 94, 95, 96, 97, 98

結果セットの処理 96

ステップ 95

単一の結果セット 97

デフォルトのレポート 97, 98

パラメーターを渡す 96

複数の結果セット 98

マクロからの呼び出し 94

有効なデータ型 95

Net.Data テーブル 98

REPORT ブロック 97, 98

接続管理

構成 23

作業負荷管理プログラムの考慮事項 23

宣言部分、マクロ構造の 53

[タ行]

直接要求

構文 44

説明 39

例 47

データ型、ストアード・プロシージャに対して有効な 95

データ・セット、アクセス権 29

定義、変数の

照会ストリング・データ 61

DEFINE ステートメントまたはブロック 60

HTML フォーム SELECT、INPUT、および TEXTAREA タグ 61

デフォルトのレポート 97, 98

印刷 81

ストアード・プロシージャに指定 97, 98

トークンのサイズ 58

特記事項 131

[ナ行]

ナビゲーション、マクロ内およびマクロ間の 57

認証、機密保護 32

ネイティブ言語サポート、関数のための 13

[ハ行]

パス・ステートメント

更新のガイドライン 16

構成、初期設定ファイル内の 15

資産の保護 33

EXEC_PATH 17

FFL_PATH 18

INCLUDE_PATH 17

MACRO_PATH 16

パフォーマンス 117

SQLCODE メッセージ 119

Web サーバー API 117

パラメーターを渡す、ストアード・プロシージャ 96

表処理変数 68

表変数 67

ファイアウォール 31

ファイル、Net.Dataへのアクセス権の指定 29

フォーマット、データ出力の 80

フォーム 41, 43

起動、Net.Data の 48

Net.Data の起動 41

Net.Data を起動するための Web ページ内の 43

複数のレポート・ブロック 83

フッター情報、REPORT ブロック 81
部分、マクロの
 宣言 53
 表示 53
フラット・ファイル・インターフェース
 言語環境 101
ブランク、変数、不要文字を削除するための 13
プリント、デフォルトのレポートの禁止 81
プログラム資料説明書、OS/390 125
ブロック、マクロ 55
ヘッダー情報、REPORT ブロック 81
変数
 隠し 65
 各種 67
 型 58, 62
 環境 63
 言語環境 69
 構成、ステートメントの
 削除、不要なブランクの
 (DTW_REMOVE_WS) 13
 使用可能にする、直接要求
 (DTW_DIRECT_REQUEST) 12
 初期設定ファイル 8
 説明 8
 データベースのコード・ページ変数
 (DefaultNetCp) 10
 ネイティブ言語サポート (DTW_MBMODE) 13
 DB2 CLI の初期設定ファイル変数
 (DSNAOINI) 11
 DB2 Subsystem ID (DB2SSID) 10
 DB2 のプラン変数 (DB2PLAN) 9
 DB2 のメッセージのパフォーマンス変数
 (DB2MSGs) 9
 SHOWSQL を使用可能にする
 (DTW_SHOWSQL) 14
 SHOWSQL を使用不可にする
 (DTW_SHOWSQL) 14
 効力範囲 59
 参照 62
 実行可能 64
 条件付き 63
 説明 58
 定義 60
 トークンのサイズ 58
 名前の動的な生成 62
 ネストされた参照 62
 表の処理 68
 リスト 66
 レポート 68
 table 67
変数の参照 62
変数名の動的な生成 62

保護、資産の 31

[マ行]

マクロ
 開発 53
 関数 70
 サンプル 54
 識別子の効力範囲 59
 条件付き論理 85
 生成、HTML の 79
 説明 1
 宣言部分 53
 内部およびその間のナビゲーション 57
 表示部分 53
 ブロック 55
 分析 54
 変数 58
 ループ 87
 DEFINE ブロック 55
 FUNCTION ブロック 56
 HTML ブロック 56
 IF ブロック 85
 WHILE ブロック 87
マクロ要求 40
 構文 40
 説明 39
 例 40
メッセージ・カタログ、使用可能にする 28
文字セット 13

[ヤ行]

ユーザー定義関数 70
用語集 132
呼び出し、アプレットの 104
呼び出し、関数の
 構文 75
呼び出し、ストアード・プロシージャの 94, 95

[ラ行]

リスト変数 66
リンク 41, 42
 Net.Data の起動 41, 47
 Net.Data を起動するための Web ページ内の 42
ループ、WHILE ブロックの 87
レポート
 複数生成、1 つの関数呼び出しによる 83
 レポートのフォーマット、カスタマイズ 82
 レポート変数 68

C

CGI、Net.Data for OS/390 の構成 24

D

DB2MSG 9, 119
DB2PLAN 9
DB2SSID 10
DBCS サポート、関数のための 13
DEFINE ブロック
 説明 55
 定義、変数の 60
DTW_APPLET 103
DTW_DIRECT_REQUEST 12
DTW_FFI 101
DTW_MBMODE 13
DTW_ODBC 91
DTW_PERL 109
DTW_REMOVE_WS 13
DTW_SHOWSQL 14

E

ENVIRONMENT ステートメント
 言語環境のタイプ 19
 構成、初期設定ファイル内の 18, 19
 構文 19
 説明 18
 パラメーター・リスト 20
 例 20
 DLL あるいはライブラリー名 20
EXEC_PATH
 構成、初期設定ファイル内の 17

F

FFI_PATH
 構成、初期設定ファイル内の 18
FUNCTION ブロック
 関数の呼び出し 75
 識別子の効力範囲 59
 説明 56
 フォーマット、出力の 80

G

GWAPI
 構成、Net.Data の 25

H

HTML 41, 42, 43
 生成、マクロ内に 79
 タグ、表の 81
 認識されていないデータ 80
 フォーム 41, 43

HTML 43, 42, 43 (続き)
 について 79
 Net.Data の起動 41, 48
 SELECT、INPUT、および TEXTAREA タグ、変数の定義 61
 ブロック
 起動、Net.Data の 79
 処理 80
 説明 56
 例 79
 リンク 41, 42
 について 42
 Net.Data の起動 41, 47
 FORM の処理依頼ボタン 80
 URL、Net.Data の起動 48
HWS_LE 102

I

ICAPI
 および Domino Go Webserver (GWAPI) 25
 構成、Net.Data の 25
IF ブロック 85
IMS Web
 言語環境 102
 Studio ツール 102
INCLUDE_PATH
 構成、初期設定ファイル内の 17

J

Java アプレット
 起動 104
 クラス 108
 言語環境 103
 作成 104
 タグの生成 104
 例 107
Java サブレット
 構成、Net.Data の 27

M

MACRO_FUNCTION ブロック
 関数の呼び出し 75
 構文 70
MACRO_PATH
 構成、初期設定ファイル内の 16
MBCS サポート、関数のための 13
MESSAGE ブロック
 構文 73
 効力範囲 73

MESSAGE ブロック (続き)

処理 73

説明 73

例 74

N

Net.Data

インストール 5

概説 1

起動 39

機密保護のメカニズム 33

構成 5

ファイル、アクセス権 29

マクロ、開発 53

OS/390 のインストール 125

Net.Data テーブル、ストアード・プロシージャ 98

Net.Data の起動 40, 41

フォーム 41, 48

マクロによる 40

リンク 41

URL 41

Net.Data マクロ。マクロを参照。 1

O

ODBC

言語環境 91

ODBC、言語環境のセットアップ 21

OS/390、Net.Data for 125

P

Perl 115

言語環境 109

スクリプトの Net.Data 変数 110, 115

R

REPORT ブロック 97, 98

ガイドライン、複数のための 84

効力範囲 60

ストアード・プロシージャ 97, 98

制約事項 84

説明 80

フォーマット、データ出力の 80

複数の 83

ヘッダーおよびフッター情報 81

例 83

RETURN_CODE 変数 73

ROW ブロック、識別子の効力範囲の 60

S

SQLCODE メッセージ、取り消し 119

SQL、言語環境のセットアップ 21

U

UNICODE 変数

DTW_MBMODE による 13

URL 41

定義、変数の 61

Net.Data の起動 41, 48

W

Web サーバー

構成、ICAPI および GWAPI の 25

設定、メッセージ・カタログのための環境変数の 28

CGI のための構成 24

Web サーバー API

によるパフォーマンスの向上 117

パフォーマンスの考慮事項 117

Web サーバーの API

構成、Net.Data の

GWAPI 25

ICAPI 25

WHILE ブロック 87



Printed in Japan

SB88-7379-01



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