

Net.Data



管理およびプログラミングの手引き OS/400 版

Net.Data



管理およびプログラミングの手引き OS/400 版

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C. 特記事項』に記載する一般情報をお読みください。

本書は以下に適用されます。

- IBM オペレーティング・システム/400 (プログラム番号 5763-SS1) バージョン 3 リリース 2 モディフィケーション・レベル 0
- IBM オペレーティング・システム/400 (プログラム番号 5716-SS1) バージョン 3 リリース 7 モディフィケーション・レベル 0
- IBM TCP/IP 接続ユーティリティー AS/400 用 (プログラム番号 5763-TC1) バージョン 3 リリース 2 モディフィケーション・レベル 0
- IBM TCP/IP 接続ユーティリティー AS/400 用 (プログラム番号 5716-TC1) バージョン 3 リリース 7 モディフィケーション・レベル 0
- IBM HTTP Server for AS/400 (プログラム番号 5769-DG1) バージョン 4 リリース 3 モディフィケーション・レベル 0

また、改訂版などで特に断らない限り、以降のすべてのバージョン、リリース、およびモディフィケーション・レベルにも適用されます。

原 典： Net.Data
Administration and Programming Guide for OS/400

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 1999.6

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

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

© Copyright International Business Machines Corporation 1997, 1999. All rights reserved.

Translation: © Copyright IBM Japan 1999

目次

まえがき	vii
Net.Data について	vii
このリリースで改訂された内容	viii
本書について	viii
本書の対象読者	ix
本書の例について	ix
第1章 概要	1
Net.Data とは ?	1
Net.Data を使用する理由	2
第2章 Net.Data の構成	5
CGI-BIN ライブラリーへの Net.Data プログラム・オブジェクトのコピー	5
Net.Data の初期設定ファイルについて	6
Net.Data の初期設定ファイルのカスタマイズ	7
初期設定ファイルの作成	8
構成変数ステートメント	8
パス構成ステートメント	14
環境構成ステートメント	19
言語環境のセットアップ	21
Java アプリケーション言語環境のセットアップ	21
SQL 言語環境のセットアップ	21
Web サーバーの構成	22
Net.Data がアクセスするオブジェクトへのアクセス権の授与	23
第3章 ユーザー資産を保護する	25
ファイアウォールを使用する	25
ネットワーク上のユーザーのデータを暗号化する	27
認証を使用する	28
許可を使用する	29
Net.Data のメカニズムを使用する	29
Net.Data 構成変数	29
マクロ開発技法	30
第4章 Net.Data を起動する	35
マクロで Net.Data を呼び出す (マクロ要求)	35
HTML リンク	37
HTML フォーム	37
永続的マクロの起動	38
永続的マクロの構文	38
例	39
第5章 Net.Data のマクロ開発	41
Net.Data マクロの分析	42
DEFINE ブロック	43
FUNCTION ブロック	44
HTML ブロック	45
Net.Data のマクロ変数	47
識別子の効力範囲	47
変数の定義	48

変数の参照	50
変数の型	52
Net.Data の関数	59
関数の定義	60
関数の呼び出し	64
Net.Data 組み込み関数の呼び出し	65
マクロでの Web ページの生成	70
HTML ブロック	70
レポート・ブロック	71
マクロにおける条件付き論理とループ	76
条件付き論理：IF ブロック	77
ループ構成体：WHILE ブロック	79
第6章 言語環境の使用	81
Net.Data 提供の言語環境の概説	82
言語環境の呼び出し	82
エラー条件の処理	82
機密保護	83
直接呼び出し言語環境	83
プログラムの呼び出し	83
パラメーターをプログラムに渡す	84
プログラムからの戻り値	86
直接呼び出し言語環境の例	87
Java アプリケーション言語環境	87
java プログラムの呼び出し	88
Java プログラムにパラメーターを渡す	88
Java アプリケーション言語環境の例	88
REXX 言語環境	89
REXX プログラムの実行	89
パラメーターを REXX プログラムに渡す	90
REXX 言語環境の例	91
SQL 言語環境	91
SQL ステートメントの実行	91
データ型の考慮事項	94
Net.Data アプリケーションにおけるトランザクション管理	98
複数のデータベース接続の管理	99
ストアード・プロシージャ	99
SQL 言語環境の例	104
システム言語環境	106
コマンドの発行およびプログラムの呼び出し	106
プログラムにパラメーターを渡す	106
システム言語環境の例	107
第7章 永続的マクロによるトランザクション管理	109
永続的マクロについて	109
トランザクションの定義	110
トランザクションの開始	111
トランザクションでのマクロ HTML ブロックの指定	112
トランザクションの終了	115
トランザクションでの変数の効力範囲の定義	115
トランザクションでの COMMIT および ROLLBACK の指定	116
永続的マクロの例	117

第8章 パフォーマンスを向上させる	121
Net.Data マクロ・キャッシング	121
言語環境の最適化	121
REXX 言語環境	121
SQL 言語環境	122
システム言語環境	123
付録A. 参考文献	125
Net.Data テクニカル・ライブラリー	125
関連資料	125
付録B. Net.Data サンプル・マクロ	127
付録C. 特記事項	131
商標	132
用語集	135
索引	137

まえがき

Net.Data® をお買い上げいただきありがとうございます。本品は、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、DB2DRDA を介してアクセスできる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 C、C++、RPG、COBOL、Java、あるいは REXX プログラミング言語で作成された (必ずしもこれらに限定されるわけではありません) アプリケーション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
- フラット・ファイルなど、他のデータ・ソースにアクセスする。

Net.Data は、この Web ページを Web サーバーに渡します。このページはネットワークを通してブラウザに転送され、表示されます。

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。Net.Data ファミリー・プロダクトは、OS/400、OS/390、Windows NT、AIX、OS/2、HP-UX、Sun Solaris、Linux、および Santa Cruz Operating System (SCO) の各オペレーティング・システムでも同様の機能を提供します。

このリリースで改訂された内容

OS/400 版 Net.Data のこのリリースでは、以下の新規機能を備えています。

- マクロと組み込みファイルのキャッシュ機能を使ったパフォーマンスの改善
- 言語環境の機能強化には、以下のものがあります。
 - 以下の 2 つの新規言語環境
 - Java アプリケーション
 - 直接呼び出し
 - SQL 言語環境でのラージ・オブジェクト (LOB) のサポート
- マクロ言語環境は、次のものを含んでいます。
 - DTW_SENDMAIL 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
 - DTW_SETCOOKIE および DTW_GETCOOKIE 組み込み関数で、HTTP の Cookies を取得し、設定する機能
 - スtring を DTW_REPLACE 関数で置き換える機能
 - DTW_TIME 関数におけるミリ秒のサポート
 - 変数参照の動的作成のサポート
 - DTW_SELECT() の OPTION 要素で VALUE 属性を設定する機能
 - MACRO_FUNCTION 言語構造体の RETURNS キーワードのサポート
 - 変数名のハッシュ (#) 文字のサポート
- 構成の機能強化には、以下のものがあります。
 - Net.Data 提供の言語環境の Net.Data 初期設定ファイルでは、ENVIRONMENT ステートメントはもう必要ありません。
 - 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)

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 データベースにアクセスし、RPG、COBOL 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、発表されていても、まだ利用可能でない製品または機能について言及する場合があります。

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。

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

<http://www.as400.ibm.com/netdata>

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。

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 を使用する理由』

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 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るようになります。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 OS/400 版、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。

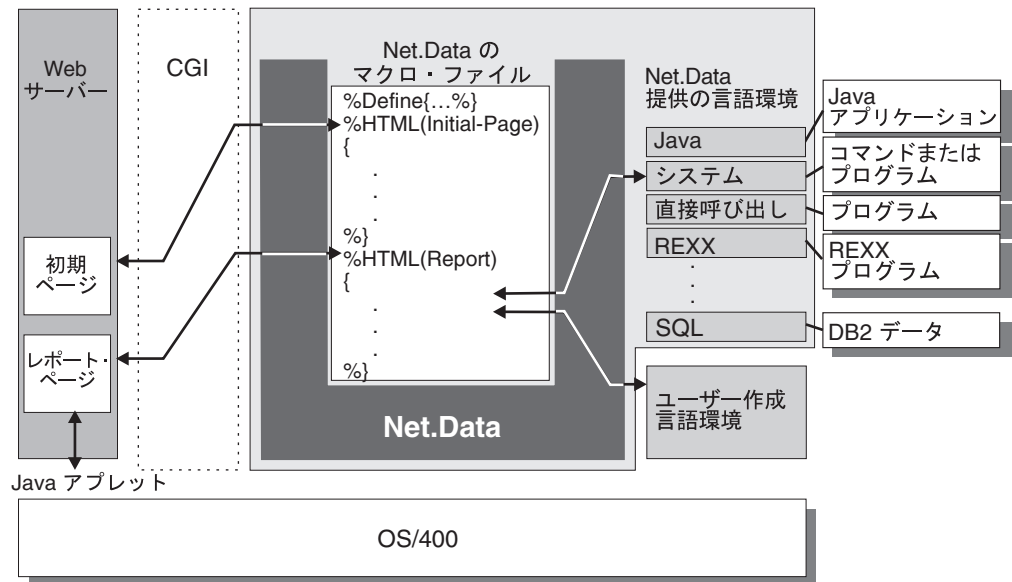


図 1. Net.Data for OS/400、Web サーバー、およびサポートされるデータとプログラム・ソース間の関係

Web サーバーは、Net.Data サービスを要求する URL を受信すると、Net.Data を CGI アプリケーションとして起動します。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 を使用する理由

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、REXX、および JavaScript などの言語を使用することができます。Net.Data はまた、DB2 データベースをアクセスする言語環境を提供します。この言語環境で、ユーザー・アプリケーションには、REXX、Perl、他各種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、ユーザーのオペレーティング・システムの既存の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。

- インターネットおよびイントラネット・アプリケーションを迅速に開発することができる、簡単で強力なマクロ言語を提供します。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++、RPG、COBOL、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。
- DB2、および DRDA 使用可能なリモート・データベースに保管されたデータに対するパフォーマンスの高い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。

フリー・フォーマット

Net.Data マクロ言語には、プログラミング・フォーマットに関して 2、3 の規則があるだけです。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。コメントはどこにでも使用できます。

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。指数形式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、47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、説明付きのメッセージがクライアントに戻されます。ブラウザー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 の構成

Net.Data for OS/400 は、標準の部分として以下の通り納入されます。

- IBM TCP/IP 通信ユーティリティー/400 V3R2、V3R7、V4R1、および V4R2
- IBM HTTP Server for AS/400 V4R3 および以降のリリース

追加購入が必要なものはありません。ダウンロードおよびインストールが必要な Net.Data ソフトウェアもありません。

必要な AS/400 TCP/IP および HTTP Server のソフトウェアは、OS/400 では標準で添付されていますが、インストールは任意選択です。OS/400 オペレーティング・システムのバージョンに応じてインストールする任意選択のソフトウェアを、以下に示します。

- IBM OS/400 オペレーティング・システム バージョン 3 リリース 2、バージョン 3 リリース 7、およびそれ以降のバージョンおよびリリース (57xx-SS1) の場合:
 - IBM TCP/IP 接続ユーティリティー AS/400 用 (57xx-TC1)
- IBM OS/400 オペレーティング・システム バージョン 4 リリース 3、およびそれ以降のバージョンおよびリリース (57xx-SS1) の場合:
 - IBM HTTP Server for AS/400 (57xx-DG1)

Net.Data のインストール後、以下の節で説明するステップを完了して Net.Data for OS/400 を構成します。このステップは以下のとおりです。

- 『CGI-BIN ライブラリーへの Net.Data プログラム・オブジェクトのコピー』
- 8ページの『初期設定ファイルの作成』
- 7ページの『Net.Data の初期設定ファイルのカスタマイズ』
- 21ページの『言語環境のセットアップ』
- 22ページの『Web サーバーの構成』
- 23ページの『Net.Data がアクセスするオブジェクトへのアクセス権の授与』

CGI-BIN ライブラリーへの Net.Data プログラム・オブジェクトのコピー

Net.Data を使用する前に、Net.Data プログラム・オブジェクトを CGI-BIN ライブラリーにコピーし、そのオブジェクトにアクセス権を与える必要があります。

Net.Data プログラム・オブジェクトをコピーするには、以下のようにします。

1. 重複オブジェクトの作成 (CRTDUPOBJ) コマンドを使用して、QTCP ライブラリーから CGI-BIN ライブラリーに Net.Data プログラム・オブジェクト DB2WWW をコピーする。

OS/400 V4R3 ユーザーの場合: ライブラリー QHTTPSVR のプログラム・オブジェクトを使用してください。QTCP ライブラリーのプログラム・オブジェクトは、Net.Data 要求を QHTTPSVR ライブラリーに経路指定します。

2. CGI-BIN ディレクトリーにある DB2WWW プログラム・オブジェクトを変更して、CGI プログラムが実行されるユーザー・プロファイルにプログラム・オブジェクトへのアクセス権を与えます。

デフォルトでは、*PUBLIC ユーザーに対する DB2WWW プログラム・オブジェクトの権限は *EXCLUDE に設定されます。プログラム・オブジェクトへのアクセス権を提供するには、*PUBLIC ユーザーに対するプログラム・オブジェクトの権限を *USE に変更するか、特に、ユーザー・プロファイルのアクセス権を DB2WWW プログラム・オブジェクトに提供してください。

Net.Data プログラム・オブジェクトを、異なるアプリケーションの複数のライブラリーにコピーすることができます。これによって、複数のバージョンの Net.Data 初期設定ファイルまたは複数の保護方式を持つことができます。Net.Data 初期設定ファイルについての詳細は、7ページの『Net.Data の初期設定ファイルのカスタマイズ』を参照してください。認証については、28ページの『認証を使用する』を参照してください。

Net.Data プログラム・オブジェクトを複数のライブラリーにコピーするには、以下のようになります。

1. 前述のステップを使用して、Net.Data プログラム・オブジェクト DB2WWW をライブラリーにコピーする。
2. Net.Data プログラム・オブジェクトを、各ライブラリーの CL プログラムに関連付ける。
 - a. ステップ 1 で指定したライブラリーに入っている Net.Data プログラム・オブジェクトを呼び出す CL プログラムを作成する。
 - b. 制御言語プログラムを各ライブラリーにコピーする。

実際には、作成した制御言語プログラムが Net.Data プログラム・オブジェクトになります。プログラム・オブジェクトを制御言語プログラムに関連付けずに Net.Data プログラム・オブジェクト DB2WWW を異なる複数のライブラリーにコピーすると、SQL 言語環境を使用するときに -901 SQL コードを受け取ります。

以下の節では、CL プログラムを作成して Net.Data を呼び出すことを選択した場合は、作成した制御言語プログラムを Net.Data プログラム・オブジェクトとして扱わなければなりません。

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。

- 電子メールを送信するための SMTP サーバーと文字セットの指定
- SQL 言語環境変数 SHOWSQL の使用可能化

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。言語環境によって、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、マクロとプログラムなどのファイルのディレクトリー・パスを指定します。

Net.Data for OS/400 では、Net.Data 初期設定ファイルの作成は任意選択です。初期設定ファイルを使用すると、Net.Data のマクロ・ファイル内で、短い URL とプログラムおよび組み込みファイルへの短い参照を使用することができます。ただし、ユーザー独自の言語環境を作成する場合は、初期設定ファイルが必要です。

初期設定ファイルを作成しない場合、Net.Data は、サポートされている言語環境ステートメントだけを使用して初期設定ファイルを構成したかのように実行されます (サポートされている言語環境について理解するには、81ページの『第6章 言語環境の使用』を参照してください)。この場合、マクロ、組み込みファイル、および実行可能ファイルのマクロ内での参照は、すべて完全修飾でなければなりません。

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、以下の節で説明する、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』
- 14ページの『パス構成ステートメント』
- 19ページの『環境構成ステートメント』

初期設定ファイルの作成方法については、8ページの『初期設定ファイルの作成』を参照してください。

図2 に示されている例の初期設定ファイルには、以下のステートメントの例が含まれ、

- | | |
|---|--|
| <pre>1 DTW_SMTP_SERVER 9.5.5.78 2 MACRO_PATH /WWW/MACRO;/QSYS.LIB/WWW.LIB/MACRO.FILE 3 INCLUDE_PATH /WWW/MACRO;/QSYS.LIB/WWW.LIB/MACRO.FILE 4 EXEC_PATH /QSYS.LIB;/QSYS.LIB/WWW.LIB 5 ENVIRONMENT(MYLE1) /QSYS.LIB/LELIB.LIB/MYLE1.SRVPGM (IN VAR1, OUT VAR2)</pre> | <ul style="list-style-type: none">• 行 1 は、構成変数の値を設定しています。• 行 2 ~ 4 は、Net.Data がアクセスする必要のあるファイルのパスを定義しています。• 行 5 は、ユーザー定義の ENVIRONMENT ステートメントを指定しています。 |
|---|--|

図2. Net.Data の初期設定ファイル

個々の構成ステートメントのテキストは、すべて 1 行に入ってい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ユーザー定義の言語環境の初期設定ファイルごとに、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。マクロ内でファイルへのすべての参照を完全に修飾する場合は、どのパス構成ステートメントも指定する必要がありません。

以下のセクションでは、初期設定ファイルの作成方法と、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。

初期設定ファイルの作成

Net.Data for OS/400 を使用する場合、初期設定ファイルの作成は任意選択です。以下の場合には、初期設定ファイルを作成する必要があります。

- Net.Data 構成変数のいずれかをデフォルトでない値に設定する。
- マクロ・ファイル、組み込みファイル、実行可能プログラム・ファイルの参照を短くするために、これらのファイルのパス・ステートメントを定義する。
- Net.Data が提供しない言語環境を使用している。

初期設定ファイルを作成するには、以下のようにします。

1. DB2WWW プログラム・オブジェクトが常駐するライブラリーで、ソース物理ファイルの作成 (CRTSRCPF) コマンドを使用して初期設定ファイルを作成する。

ファイル名: INI

メンバー名: DB2WWW

構成ステートメントのテキストはすべて 1 行に入っていないといけないので、初期設定ファイルはレコード長を 240 として作成することをお勧めします。

2. サンプル・マクロおよび以下のセクションに示されているように、原始ステートメント入力ユーティリティー (SEU) またはワークステーション・エディターを使用して、構成ステートメントをファイルに追加する。

初期設定ファイルを作成してから更新する場合は、変更内容を有効にするために Web サーバーを終了または再始動する必要はありません。Net.Data は、HTTP サーバー・ジョブによる初期起動時に初期設定ファイルを 1 度読み取ります。構成データは保管されるので、次の Net.Data 呼び出し時でも、Net.Data は初期設定ファイルを読み取る必要はありません。ただし、初期設定ファイルを変更した場合は、Net.Data は初期設定ファイルに対する変更を検出して初期設定ファイルを再度読み取ります。

許可に関するヒント: Net.Data が実行されるユーザー ID に、このファイルに対する適切なアクセス権を与えてください。詳しくは、23 ページの『Net.Data がアクセスするオブジェクトへのアクセス権の授与』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によって異なります。また、その他、アプリケーション固有の要因によっても異なります。

構成変数ステートメントを更新するには、以下のようにします。

アプリケーションに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します。構成変数の構文は、以下のようになります。

`NAME[=]value-string`

等号は、大括弧で示されるように、オプションです。

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更ステートメントについて説明します。

- 『DTW_MACRO_CACHE_SIZE: マクロ・キャッシュ・サイズ変数』
- 『DTW_PAD_PGM_PARMS: パラメーター埋め込み構成変数』
- 10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』
- 11ページの『DTW_SMTP_CCSSID: 電子メール SMTP CCSSID 変数』
- 11ページの『DTW_SMTP_CHARSET: 電子メールの SMTP 文字セット変数』
- 12ページの『DTW_SMTP_SERVER: 電子メール SMTP サーバー変数』
- 12ページの『DTW_SQL_ISOLATION: DB2 の分離変数』
- 13ページの『DTW_SQL_NAMING_MODE: SQL の表名変数』
- 14ページの『DTWR_CLOSE_REGISTRIES: Web レジストリー・オープン変数』

DTW_MACRO_CACHE_SIZE: マクロ・キャッシュ・サイズ変数

マクロをキャッシングするときに、`Net.Data` が使用しなければならないメモリーをメガバイトで示します。キャッシュ・サイズを超えると、`Net.Data` は古いキャッシュ済みのマクロを除去し、キャッシュ内に空間を作ります。`Net.Data` は、最近最も使用度の低いマクロを除去します。

構文：

`DTW_MACRO_CACHE_SIZE [=] size`

変数:

`size` キャッシュ・メモリーのサイズをメガバイト数で指定します。デフォルトは 5 MB で、キャッシングはいつでも使用可能です。`size` が 0 の場合は、マクロはキャッシュされません。`size` が 1 から 4 の場合は、デフォルトの 5 を使用します。

例：16 MB のキャッシュ・サイズを指定する

`DTW_MACRO_CACHE_SIZE 16`

DTW_PAD_PGM_PARMS: パラメーター埋め込み構成変数

プログラムまたはストアド・プロシージャに渡される文字パラメーターを、ブランクで埋め込むかどうかを言語環境に示します。文字パラメーターのデータ型は、`CHARACTER` または `CHAR` です。

`IN` パラメーターまたは `INOUT` パラメーターの場合、パラメーター値の長さが指定された精度より小さいと、パラメーター値の長さが精度と同じになるまで、パラメーター値の右方にブランクが埋め込まれます。

OUT パラメーターの場合、パラメーター値は *precision* ブランクに設定されます。

プログラムまたはストアード・プロシージャへの呼び出し後、すべての後書きブランクは、OUT と INOUT のパラメーター値から除去されます。

この変数を Net.Data 初期設定ファイルで設定し、すべてのマクロに値を指定します。マクロでこの変数を定義して、その値を上書きすることができます。

DTW_PAD_PGM_PARMS がマクロで定義されていない場合は、初期設定ファイルの値を使用します。

DTW_PAD_PGM_PARMS は、Direct Call および SQL の言語環境でサポートされます。

構文：

DTW_PAD_PGM_PARMS [=] YES|NO

変数:

YES すべての IN および INOUT の文字パラメーター値を、パラメーターの定義済み精度に応じて、左寄せしブランクで埋め込んだままにしておいてから、そのパラメーターをプログラムまたはストアード・プロシージャに渡すことを指定します。後書きブランクは、プログラムまたはストアード・プロシージャへの呼び出し後除去されます。

NO パラメーターをプログラムまたはストアード・プロシージャに渡すときに、文字パラメーター値に埋め込みを追加しないことを指定します (値は NULL で終了します)。後書きブランクは、プログラムまたはストアード・プロシージャへの呼び出し後に除去されます。これはデフォルトです。

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

変数:

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。

11ページの表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_CCSID: 電子メール SMTP CCSID 変数

DTW SMTP_CHARSET で指定されるマルチパーパス・インターネット・メール・エクステンション (MIME) 文字セットに関連した、ASCII エンコード文字セット識別子 (CCSID)を指定します。DTW_SENDMAIL 関数で指定されたデータを EBCDIC から ASCII に変換するときに、CCSID を使用します。

DTW SMTP_CCSID を指定する場合は、DTW SMTP_CHARSET も指定しなければなりません。CCSID を指定するときには、それが、DTW SMTP_CHARSET で指定された MIME 文字セットに適していること、およびシステムで CCSID がサポートされていることを確認してください。12ページの表2 には、共通 MIME 文字セットおよび関連した ASCII CCSID がリストされています。DTW SMTP_CCSID が設定されていない場合は、*Net.Data* は、MIME 文字セット ISO-8859-1 に関連した CCSID を使用します。その CCSID は 819 です。

構文：

```
DTW SMTP_CCSID [=] ascii_ccsid
```

ここで、*ascii_ccsid* は ASCII CCSID (1 から 65534 までの数字) で、EBCDIC から ASCII に変換するときに使用します。

例：

```
DTW SMTP_CCSID 912
```

この ASCII CCSID は、MIME 文字セット ISO-8859-2 に対応します。

DTW SMTP_CHARSET: 電子メールの SMTP 文字セット変数

マルチパーパス・インターネット・メール・エクステンション (MIME) 文字セットを指定します。この文字セットは、DTW_SENDMAIL 関数が電子メール・メッセージで使用します。DTW SMTP_CHARSET を指定する場合は、DTW SMTP_CCSID も指定しなければなりません。MIME 文字セットを指定するときには、文字セットが有効であることを確認してください。それは、*Net.Data* は、この変数で指定された値の妥当性検査を行わないからです。DTW SMTP_CHARSET が設定されていない場合は、*Net.Data* は、関連した 819 の CCSID とともに MIME 文字セット ISO-8859-1 を使用します。

12ページの表2 には、共通 MIME 文字セットおよび関連した ASCII CCSID がリストされています。

表 2. *Net.Data* がサポートする文字セット

MIME 標準文字セット	ASCII CCSID	説明
米国 ASCII	367	米国英語
ISO-2022-JP	5052	日本 MBCS
ISO-8859-1	819	ラテン 1
ISO-8859-2	912	ラテン 2
ISO-8859-5	915	キリル文字
ISO-8859-6	1089	アラビア語
ISO-8859-7	813	ギリシャ語
ISO-8859-8	916	ヘブライ語
ISO-8859-9	920	ラテン 5

構文：

`DTW_SMTP_CHARSET character_set`

ここで、*character_set* は、使用する MIME 文字セットです。

例：

`DTW_SMTP_CHARSET iso-8859-2`

この MIME 文字セットは、912 ASCII CCSID に対応します。

DTW_SMTP_SERVER: 電子メール SMTP サーバー変数

SMTP サーバーを指定して、`DTW_SENDMAIL` 組み込み関数を使用する電子メール・メッセージを送信します。この変数の値は、ホスト名または IP アドレスのいずれか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* はローカル・ホストを SMTP サーバーとして使用します。

構文：

`DTW_SMTP_SERVER server_name`

ここで、*server_name* は、電子メール・メッセージの送信に使用する SMTP サーバーのホスト名または IP アドレスです。

パフォーマンスのためにヒント: この値に IP アドレスを指定して、指定された SMTP サーバーの IP アドレスを検索するときに、*Net.Data* がドメイン名サーバーに接続しないようにします。

例：

`DTW_SMTP_SERVER 9.5.34.5`

DTW_SQL_ISOLATION: DB2 の分離変数

`DTW_SQL` 言語環境は、`DTW_SQL_ISOLATION` 構成ステートメントを使用して、`DTW_SQL` 言語環境により実行されているデータベースの操作を、同時に実行されているプロセスからどの程度分離するかを決定します。

構文：

DTW_SQL_ISOLATION *locking_method*

ここで、*locking_method* は、以下の値の 1 つです。

DTW_SQL_NO_COMMIT

コミットメント制御を使用しないよう指定します。OS/400 オペレーティング・システムの場合、リレーショナル・データベースがリレーショナル・データベースのディレクトリーで指定されていたり、リレーショナル・データベースが非 OS/400 オペレーティング・システムにある場合は、この値を指定しないでください。

DTW_SQL_READ_UNCOMMITTED

SQL の ALTER、COMMENT ON、CREATE、DROP、GRANT、LABEL ON、および REVOKE ステートメントと、更新、削除および挿入された列で参照されているオブジェクトのロックングを指定します。オブジェクトは、作業単位 (トランザクション) の終了までロックされます。他の処理でのコミットされていない変更は、見ることはできません。

DTW_SQL_READ_COMMITTED

SQL の ALTER、COMMENT ON、CREATE、DROP、GRANT、LABEL ON、および REVOKE ステートメントと、更新、削除および挿入された列で参照されているオブジェクトのロックングを指定します。オブジェクトは、作業単位 (トランザクション) の終了までロックされます。選択されてはいるが更新されていない行は、次の行が選択されるまでロックされます。他の処理でのコミットされていない変更は、見ることはできません。

DTW_SQL_REPEATABLE_READ

SQL の ALTER、COMMENT ON、CREATE、DROP、GRANT、LABEL ON、および REVOKE ステートメントと、選択、更新、削除および挿入された列で参照されているオブジェクトのロックングを指定します。オブジェクトは、作業単位 (トランザクション) の終了までロックされます。他の処理でのコミットされていない変更は、見ることはできません。

DTW_SQL_SERIALIZABLE

SQL の ALTER、COMMENT ON、CREATE、DROP、GRANT、LABEL ON、および REVOKE ステートメントと、選択、更新、削除および挿入された列で参照されているオブジェクトのロックングを指定します。オブジェクトは、作業単位 (トランザクション) の終了までロックされます。他の処理でのコミットされていない変更は、見ることはできません。

SELECT、UPDATE、DELETE、および INSERT ステートメントで参照されるすべての表は、作業単位 (トランザクション) の終了まで、排他的にロックされます。

DTW_SQL_NAMING_MODE: SQL の表名変数

DTW_SQL_NAMING_MODE 構成ステートメントは、SQL ステートメントにおける表名の指定方法を指定します。

構文：

DTW_SQL_NAMING_MODE *mode*

ここで、*mode* は、以下の値の 1 つです。

SQL_NAMING

次の形式の集合名で表が修飾されるよう指定します。

collection.table

ここで、*collection* は、集合の名前で、*table* は表名です。デフォルトの修飾子は、SQL ステートメントを実行するプロセスを実行しているユーザー ID です。表名が明示的に修飾されておらず、デフォルトの集合名が指定されていない場合に使用されます。SQL_NAMING は、デフォルトの表名です。

SYSTEM_NAMING

以下の形式のライブラリー名でファイルを修飾するよう指定します。

library/file

ここで、*library* はライブラリー名で、*file* は表名です。デフォルトの検索パスは、表名 (file) が明示的に修飾されておらず、デフォルトの集合名 (library) が指定されていければ、非修飾表名の場合は、ライブラリー・リスト (*LIBL) になります。

DTWR_CLOSE_REGISTRIES: Web レジストリー・オープン変数

Web レジストリーをクローズするか、オープンしておくかを指定します。この変数は、Web レジストリーをオープンしておき、同じレジストリーにアクセスする Net.Data のマクロが連続して起動しても、そのレジストリーを再オープンしないようにします。

構文：

DTWR_CLOSE_REGISTRIES YES|NO

変数:

YES Net.Data のマクロが処理された後で、オープンしているすべての Web レジストリーを閉じるよう指定します。

NO Net.Data のマクロが処理された後でも、オープンしているすべての Web レジストリーをそのままオープンにしておくことを指定します。NO がデフォルトです。

パフォーマンスのためのヒント：DTWR_CLOSE_REGISTRIES 構成ステートメントを使用して、レジストリーのオープンおよびクローズを最小限にすることにより、(Web レジストリーの組み込み関数を使用して) Web レジストリーのアクセス性能を改善することができます。同時に多重プロセスでレジストリーにアクセスできる場合 (同時のブラウザー要求の場合) は、DTWR_CLOSE_REGISTRIES を YES に設定します。

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。

- 15ページの『MACRO_PATH』
- 16ページの『EXEC_PATH』

- 17ページの『INCLUDE_PATH』
- 18ページの『FFI_PATH』
- 18ページの『HTML_PATH』
- 19ページの『DTW_JAVA_CLASSPATH』

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、テキスト・ファイル、および組み込みファイルを配置しようとしたときに、Net.Data が検索する 1 つ以上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。

更新のガイドライン：

いくつかの一般的なガイドラインは、パス・ステートメントに適用されます。例外は、各パス・ステートメントの説明で注釈が付けられています。

- 指定された各ディレクトリーは、セミコロン (;) で終了する。
- スラッシュ (/) および円記号 (¥) は同じものとして扱われる。
- 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。この複数パスの指定が可能であることにより、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/cgi-bin/db2www/macro/sqlm.d2w/report
```

構文：

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

等号 (=) は、大括弧で示されているようにオプションです。

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、35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/cgi-bin/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。

MACRO_PATH ステートメントで指定されたディレクトリーでファイルが検出されない場合、*Net.Data* はルート・ディレクトリー (*/*) でファイルを検索します。たとえば、以下の URL を実行依頼したとします。

```
http://myserver/cgi-bin/db2www/myfile.txt/report
```

MACRO_PATH に指定されたどのディレクトリーでもファイル *myfile.txt* が検出されなかった場合、*Net.Data* はルート・ディレクトリー (*/*) でファイルを検索します。

```
/myfile.txt
```

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 /qsys.lib/programs.lib;/qsys.lib/rexx.lib/rexxpgms.file;
```

Net.Data のマクロ

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

ファイル *myFunction.mbr* が */qsys.lib/rexx.lib/rexxpgms.file* 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 */qsys.lib/rexx.lib/rexxpgms.file/myFunction.mbr* です。

EXEC_PATH ステートメントで指定されたディレクトリーでファイルが検出されない場合は、以下のように処理されます。

- 指定されたパスが絶対パスである場合は、*Net.Data* は、指定されたパスでファイルを検索します。たとえば、以下の EXEC ステートメントを指定したとします。

```
%EXEC{/qsys.lib/programs.lib/rpg1.pgm %}
```

Net.Data は、*/qsys.lib/programs.lib* ディレクトリーでファイル *rpg1.pgm* を検索します。

- 指定されたパスが相対パスである場合は、Net.Data は現行作業ディレクトリーを検索します。たとえば、以下の EXEC ステートメントを指定したとします。

```
%EXEC { rpg1.pgm %}
```

すると、Net.Data は現行作業ディレクトリーでファイル rpg1.pgm を検索します。

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
```

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、および /usr/lpp/netdata/includes/OE で検出されます。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。

INCLUDE_PATH ステートメントで指定されたディレクトリーでファイルが検出されない場合は、以下のように処理されます。

- 指定されたパスが絶対パスである場合は、Net.Data は、指定されたパスでファイルを検索します。たとえば、以下の INCLUDE ステートメントを指定したとします。

```
%INCLUDE "/u/user1/includes/oeheader.inc"
```

すると、Net.Data は、/u/user1/includes ディレクトリーでファイル oeheader.inc を検索します。

- 指定されたパスが相対パスである場合は、`Net.Data` は現行作業ディレクトリーを検索します。たとえば、以下の `INCLUDE` ステートメントを指定したとします。

```
%INCLUDE "oeheader.inc"
```

すると、`Net.Data` は現行作業ディレクトリーでファイル `oeheader.inc` を検索します。

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 フラット・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ないファイルに対する機密保護を提供するために使用されるので、検出されない `FFI` ファイルに対して特別に提供されるものではありません。 *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。

HTML_PATH

`HTML_PATH` 構成ステートメントは、`Net.Data` がラージ・オブジェクト (LOB) を書き込むディレクトリーを指定します。このパス・ステートメントは、1 つのディレクトリー・パスしか受け取りません。

`HTML_PATH` は、`QSYS.LIB` ファイル・システムにない `IFS` ディレクトリーを指定しなければなりません。

構文：

```
HTML_PATH [=] path
```

例：以下の例は、初期設定ファイルの `HTML_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：

```
HTML_PATH /db2/lobs
```

照会が `LOB` を戻すと、`Net.Data` はそれを、`HTML_PATH` 構成ステートメントで指定されたディレクトリーに格納します。

パフォーマンスのためのヒント：LOB を使用する場合、システムの制限を考慮してください。その理由は、システムの制限は、すぐにリソースを消費してしまうからです。詳しくは、94ページの『ラージ・オブジェクトを使用する』を参照してください。

DTW_JAVA_CLASSPATH

DTW_JAVA_CLASSPATH 構成ステートメントは、Java クラスを位置指定するために使用するパスを指定します。ディレクトリーはコロンで区切ります。

構文：

```
DTW_JAVA_CLASSPATH [=] path
```

例：以下の例は、初期設定ファイルの DTW_JAVA_CLASSPATH 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：

```
DTW_JAVA_CLASSPATH /directory1/directory2:/QIBM/ProdData/Java400
```

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 81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。

OS/400 版の Net.Data には、Net.Data とともに出荷される言語環境の ENVIRONMENT ステートメントは必要ありません。ただし、言語環境ステートメントがある場合は、Net.Data が使用するデフォルトをオーバーライドします。言語環境を提供された Net.Data の ENVIRONMENT ステートメントは、Net.Data 構成ファイルに追加されないことを推奨します。

ENVIRONMENT ステートメントのパラメーターとして変数を指定することによって、その変数を言語環境と関連付けることができます。Net.Data は、ENVIRONMENT ステートメントで指定されるパラメーターを、暗黙的にマクロ変数として言語環境に渡します。マクロの ENVIRONMENT ステートメントで指定されるパラメーターの値を変更するには、DTW_ASSIGN() 関数を使用する変数に値を割り当てるか、あるいは、DEFINE セクションで変数を定義します。

重要：マクロ変数がマクロでは定義されているが、ENVIRONMENT ステートメントでは指定されていない場合は、そのマクロ変数は言語環境には渡されません。

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データベースの名前を指定する DATABASE 変数を定義することができます。DATABASE 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、指定されたデータベースに接続することができます。変数を言語環境に渡すには、 `DATABASE` 変数を、 `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。

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、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、以下のようになります。

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 サービス・プログラムの名前。

サービス・プログラム名は、拡張子 `.SRVPGM` を付けて指定します。

- *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。

Net.Data が初期設定ファイルを処理する場合、Net.Data は、言語環境の サービス・プログラム をロードしません。Net.Data が言語環境の サービス・プログラムをロードするのは、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。サービス・プログラム 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。

OS/400 では、ENVIRONMENT ステートメントは Net.Data の言語環境には必要なく、お勧めできません。ただし、この例では、Net.Data が使用する デフォルトの ENVIRONMENT ステートメントのいくつかを示しています。

```
1 MACRO_PATH      /WWW/MACRO;/QSYS.LIB/WWW.LIB/MACRO.FILE
2 INCLUDE_PATH    /WWW/MACRO;/QSYS.LIB/WWW.LIB/MACRO.FILE
3 EXEC_PATH       /QSYS.LIB;/QSYS.LIB/WWW.LIB

4 ENVIRONMENT(DTW_REXX) /QSYS.LIB/QTCP.LIB/QTMHREXX.SRVPGM ( )
5 ENVIRONMENT(DTW_SQL) /QSYS.LIB/QTCP.LIB/QTMSQL.SRVPGM (IN DATABASE,
  LOGIN, PASSWORD, TRANSACTION_SCOPE, SHOWSQL, DB_CASE,
  RPT_MAX_ROWS, START_ROW_NUM, DTW_SET_TOTAL_ROWS,
  OUT DTWTABLE, SQL_CODE, TOTAL_ROWS)
6 ENVIRONMENT(DTW_SYSTEM) /QSYS.LIB/QTCP.LIB/QTMSYS.SRVPGM ( )
```

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。

- 『Java アプリケーション言語環境のセットアップ』
- 『SQL 言語環境のセットアップ』

Java アプリケーション言語環境のセットアップ

最初に OS/400 V4R4 に導入されるJava アプリケーション言語環境を使用する前に、以下のステップを完了します。

1. AS/400 Java 開発キット ライセンス・プログラム (製品識別コード 5769JV1) をインストールします。AS/400 Java 開発キットをインストールし、AS/400 上で Java アプリケーションを実行しなければなりません。
2. Net.Data 初期設定ファイルで DTW_JAVA_CLASSPATH パス構成変数を設定し、Java が Java アプリケーション・クラスを検出できるようにします。このパス構成ステートメントの詳細については、19ページの『DTW_JAVA_CLASSPATH』を参照してください。

Java アプリケーション言語環境をセットアップ後、Java アプリケーション言語環境の使用方法については、87ページの『Java アプリケーション言語環境』を参照してください。

SQL 言語環境のセットアップ

SQL 言語環境を使用する前に、以下のステップを完了します。

1. SQL 言語環境がアクセスする必要があるリモート・データベースの他に、ローカル・データベースのディレクトリー項目 (リモート・ロケーション *LOCAL を持つディレクトリー項目) を関係データベースのディレクトリーに作成する。

リレーショナル・データベース登録簿項目追加 (ADDRDBDIRE) コマンドを使用して項目を追加します。

リモート・データベースにアクセスしている場合は、ローカル・システムとリモート・システム間の通信のセットアップなど、追加の構成のためのステップを完了します。分散データベース・サポートの詳細については、 *OS/400 Distributed Database Programming* を参照してください。

2. DataLinks を使用している場合は、使用するすべてのシステムで TCP/IP が構成されていること、および、リンクするオブジェクトを含むすべてのシステムで、DataLink ファイル・マネージャーが開始され構成されていることを確認します。DataLinks の詳細については、*DB2 AS/400 用 SQL プログラミング* を参照してください。
3. SQL 言語環境でラージ・オブジェクト (LOB) が戻される場合には、HTML_PATH 構成変数を設定します。この構成変数の詳細については、18 ページの『HTML_PATH』を参照してください。
4. 構成変数を追加または更新する。SQL 言語環境では、Net.Data 初期設定ファイルで指定できる以下の構成変数をサポートします。

DTW_SQL_ISOLATION

SQL 言語環境で実行されるデータベース操作を、同時に実行されているプロセスからどの程度分離するかを決定します。

DTW_SQL_NAMING_MODE

SQL ステートメントにおける表名の指定方法を決定します。

DTW_SHOWSQL

マクロ変数 SHOWSQL を使用できるようにします。

Net.Data 構成変数ステートメントについては、8 ページの『構成変数ステートメント』を参照してください。

SQL 言語環境のセットアップ後は、SQL 言語環境の使用方法については、91 ページの『SQL 言語環境』を参照してください。

Web サーバーの構成

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 を使い慣れた Web サーバーと一緒に使用することができます。

Net.Data が起動されるように Map、Exec、および Pass のディレクティブを HTTP 構成ファイルに追加することによって、Net.Data を起動するように Web サーバーを構成します。

たとえば、Net.Data プログラム・オブジェクトがライブラリー CGI に常駐すると想定した場合、以下のディレクティブは Net.Data 要求を /QSYS.LIB/CGI.LIB/DB2WWW.PGM にリダイレクトします。

```
Map /cgi-bin/db2www/* /QSYS.LIB/CGI.LIB/DB2WWW.PGM/*
Map /CGI-BIN/DB2WWW/* /QSYS.LIB/CGI.LIB/DB2WWW.PGM/*
Exec /QSYS.LIB/CGI.LIB/*
```

勧告： HTTP 構成ファイル内でディレクティブを Map、Exec、Pass の順序で編成し、ディレクティブが無視されないようにします。たとえば、以下の Pass ディレクティブが Map または Exec のディレクティブより先になると、Map と Exec のディレクティブは無視されます。

```
Pass /*
```

Map ディレクティブ

Map ディレクティブは、形式 `/cgi-bin/db2www/*` を使用する記入項目を、システム上の Net.Data プログラムが入っているライブラリーにマップします。(ストリングの終わりのアスタリスク (*) は、ストリングの後に続く任意の項目を参照します。) ディレクティブでは大文字と小文字が区別されるので、大文字と小文字の両方のマップ・ステートメントが含まれます。この例では、両方の Map ステートメントが同じロケーションを指します。

Exec ディレクティブ

Exec ディレクティブにより、Web サーバーは CGI ライブラリーの CGI プログラムを実行することができます。プログラムが常駐するライブラリー (プログラム自体ではなく) をディレクティブに指定します。

Pass ディレクティブ

SQL 言語環境でラージ・オブジェクト (LOB) を使用したい場合は、Pass ディレクティブを作成し、SQL 言語環境が LOB ファイルを保管するディレクトリーを指定します。たとえば、以下のような場合です。

```
Pass /tmplobs/* /html_path/*
```

ここで、`html_path` は、LOB を保管するデフォルトのディレクトリーを指定する HTML_PATH 構成変数で指定されるディレクトリー名です。詳しくは、18ページの『HTML_PATH』を参照してください。

Net.Data は Pass ディレクティブを使用しません。URL を単純化したい場合は、15ページの『MACRO_PATH』で説明するように、Net.Data 初期設定ファイルの MACRO_PATH ステートメントを使用します。

Net.Data がアクセスするオブジェク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オブジェクトと、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。

さらに具体的にいえば、Net.Data を実行するユーザー ID に以下の許可を与えてください。

- Net.Data の初期設定ファイル、INI.FILE/DB2WWW.MBR の読み取り
- Net.Data の実行可能ファイルとサービス・プログラムの実行、および実行可能なファイルとサービス・プログラムへのパスにあるディレクトリー (ライブラリー) 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
- 言語環境のターゲットが参照する可能性がある任意のオブジェクトへのアクセス。たとえば、SQL 言語環境では SQL ステートメントが実行され、SQL ステートメントはデータ・ファイルにアクセスします。したがって、Net.Data が実行されているユーザー ID には、データベース・ファイルに対する権限がなければなりません。

例:

Net.Data マクロの保管用に選択したファイル・システムに応じて、Net.Data CGI プログラムを実行するユーザー・プロファイルに Net.Data マクロを許可する必要があります。以下のメソッドでは、QTMHHTTP1 ユーザー・プロファイルに権限を与えています。(V3R2 および V3R7では、Internet Connection for AS/400 は CGI プログラムを QTMHHTTP1 ユーザー・プロファイルでのみ実行していました。)

- ルート・ファイル・システムで、権限変更 (CHGAUT) CL コマンドを使用して、ユーザー・プロファイルに権限を与えます。

```
CHGAUT OBJ('/WWW') USER(QTMHHTTP1) DTAAUT(*RX)
CHGAUT OBJ('/WWW/macro') USER(QTMHHTTP1) DTAAUT(*RX)
CHGAUT OBJ('/WWW/macro/*') USER(QTMHHTTP1) DTAAUT(*RX)
```

パス内のすべてのオブジェクトに権限を与える必要があります。

- ライブラリー・ファイル・システム (QSYS.LIB) では、オブジェクト権限認可 (GRTOBJAUT) CL コマンドを使用してユーザー・プロファイルに権限を与えます。

```
GRTOBJAUT OBJ(WWW) OBJTYPE(*LIB) USER(QTMHHTTP1) AUT(*USE)
GRTOBJAUT OBJ(WWW/MACRO) OBJTYPE(*FILE) USER(QTMHHTTP1) AUT(*USE)
```

ライブラリーおよびソース物理ファイルのみに権限を与える必要があります。

CHGAUT CL コマンドを使用して、以下のように、QSYS.LIB ファイル・システムのオブジェクトに対する権限を与えることもできます。

```
CHGAUT OBJ('/QSYS.LIB/WWW.LIB') USER(QTMHHTTP1) DTAAUT(*RX)
CHGAUT OBJ('/QSYS.LIB/WWW.LIB/MACRO.FILE') USER(QTMHHTTP1) DTAAUT(*RX)
```

言語環境固有の権限に関する考慮事項については、81ページの『第6章 言語環境の使用』の各言語環境についてのセクションを参照してください。

第3章 ユーザー資産を保護する

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。

以下の節には、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、次のとおりです。

- 『ファイアウォールを使用する』
- 27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』
- 28ページの『認証を使用する』
- 29ページの『許可を使用する』
- 29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制限します。

Net.Data は、ユーザーの環境で実行する、ファイアウォール製品と組み合わせて使用されます。

以下の構成が、Net.Data アプリケーションの機密保護の推奨される管理方法です。これらの構成情報は高水準なもので、パブリック・インターネットからユーザーのセキュア・イントラネットを分離するファイアウォールが、構成されていることを前提としています。組織のセキュリティ・ポリシーと合わせて、これらの構成を注意深く考慮してください。

- **高度な機密保護構成**

この構成は、セキュア・イントラネットとパブリック・インターネットの両方から、Net.Data および Web サーバーを分離する、サブネットワークを作成します。このファイアウォール・ソフトウェアは、Web サーバーとパブリック・インターネット間のファイアウォール、および、Web サーバーとセキュア・イントラネット間

のもう 1 つのファイアウォールを、作成するために使用されます。ここには、DB2 サーバーが含まれます。この構成を、図3 に示します。

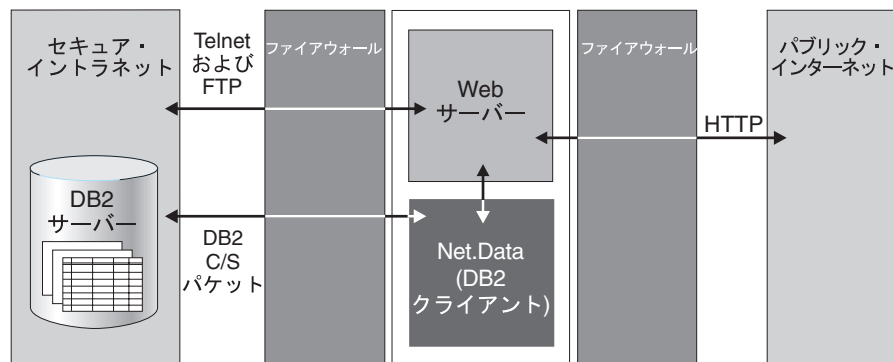


図3. 高度な機密保護構成

この構成をセットアップするには、以下のようにします。

- Net.Data を Web サーバー・マシンにインストールし、ファイアウォールを介した DB2 通信を可能にするようにファイアウォールを構成することによって、イントラネット内部の DB2 サーバーに確実にアクセスできるようにします。1 つのメソッドは、パケット・フィルタ規則を追加して、Net.Data からの DB2 クライアント要求を可能にし、DB2 サーバーから Net.Data へのパケットを確認することです。
- Web サーバーとセキュア・イントラネット間で、FTP アクセスおよび Telnet アクセスを許可します。1 つの方法は、Web サーバー・マシン上に socks サーバーをインストールすることです。
- ファイアウォール・ソフトウェアのパケット・フィルタ構成ファイルでは、標準 HTTP ポートからの着信 TCP パケットが、Web サーバーにアクセスできるように指定します。また、発信 TCP 確認パケットが、Web サーバーからパブリック・インターネット上のすべてのホストに着信するように指定します。

• 中間機密保護構成

この構成では、ファイアウォール・ソフトウェアは、パブリック・インターネットから、DB2 サーバーを使用するセキュア・イントラネットを分離します。Net.Data および Web サーバーは、ファイアウォール以外はワークステーション・プラットフォーム上にあります。この構成は、最初の構成より単純ですが、それでもデータベースの保護が可能です。27ページの図4 は、この構成を示しています。

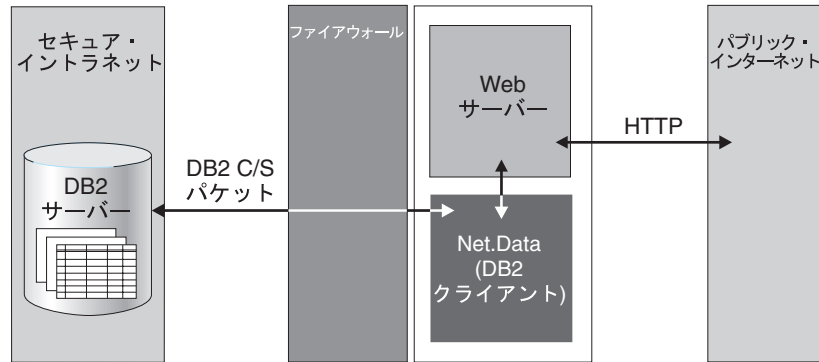


図4. 中間機密保護構成

ファイアウォールは、DB2 クライアント要求が Net.Data から DB2 に流れ、確認パケットが DB2 から Net.Data に流れるように構成しなければなりません。

- 単純な機密保護構成

この構成では、DB2 サーバーおよび Net.Data は、ファイアウォールおよびセキュア・イントラネットの外側にインストールされます。これらの DB2 サーバーおよび Net.Data は、外部ハッキングからは保護されません。ファイアウォールには、この構成ではパケット・フィルタ規則は必要ありません。図5 は、この構成を示しています。

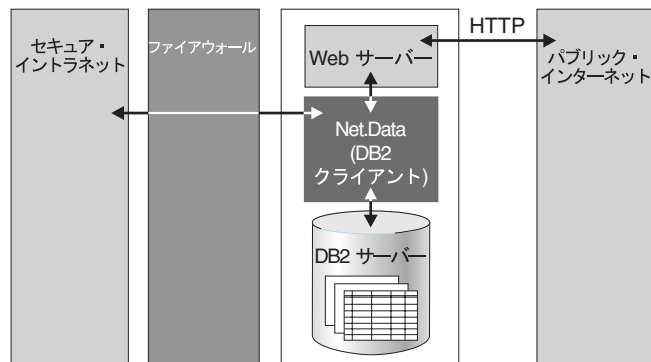


図5. 単純な機密保護構成

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/400 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 を関連付けることができます。

クライアントを基にした認証

ユーザーは、ローカル OS/400 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、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。

クライアントを基にした認証は、OS/400 V4R1 以降の IBM の HTTP サーバーでサポートされています。

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られているため、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。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。

デフォルトでは、IBM の HTTP サーバーは、QTMHHTTP1 ユーザー ID (ユーザー・プロファイル) で CGI プログラムを実行します。ただし、UserID ディレクティブが有効であるか、あるいは UserID サブディレクティブを指定した保護セットアップ内にある場合は、このプログラムは指定されたユーザー ID のもとで実行されます。

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付けられています。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持つ、代理ユーザー ID の作成が必要です。代理ユーザー ID による認証では、通常、V4R1 で最初に導入された妥当性検査リスト・オブジェクトを使用します。詳細および例については、OS/400 システム API 解説書を参照してください。

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 サーバーのインストール、および Protect、Protection、DefProt、UserId ディレクティブを使用した Web サーバーの構成に関する詳細については、HTTP サーバーの資料を参照してください。

ヒント: Net.Data マクロを保護するには、以下のようにします。

1. Net.Data プログラム・オブジェクトの Web サーバー構成ファイルに、保護ディレクティブを追加する。
2. Net.Data が実行されるユーザー ID に、マクロ・ファイルへのアクセス権を与える。アクセス権の授与に関する詳細については、23ページの『Net.Data がアクセスするオブジェクトへのアクセス権の授与』を参照してください。

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 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、28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

上記で説明した方法に加えて、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したり、あるいはデータベースの設計を隠したりするために使用できる、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、またはその他のフラット・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、30ページの『マクロ開発技法』で説明されているように、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。

実動システムの 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ように指定できます。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。

以下の方法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する Net.Data のバージョンを使用している場合は、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 設定の結果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、10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』で説明されている、DTW_ASSIGN() 関数を使用してください。

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。

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。マクロが意図された方法で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 変数を、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 変数の設定は、%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、10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の詳細は、*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、その入力が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、または完全な解析を必要とする場合があります。

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 になるようにします。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" or 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() を使用する

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、DTW_TRANSLATE および DTW_POS スtring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tring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、組み込まれるファイルは、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String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、ファイルは組み込まれます。 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 を組み込みます。このファイルは、INCLUDE_PATH 構成ステートメント指定では組み込まれません。

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、以下のロジックを使用して、ファイル名の変数と関連付けられた入力値に、ストリング ".." が含まれないことが確認できます。

```
@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、データベースおよび照会を設計する

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を、すべてのユーザーが Web ブラウザーで使えるようになります。

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)
```

この方法は、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、ブラウザー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 がブラウザー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 隠し変数を使用する

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 隠し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。隠し変数の詳細については、54ページの『隠し変数』を参照してください。

ユーザーからの検証情報を要求する

ユーザー提供の入力データを基に独自の保護体系を作成することができます。たとえば、HTML フォームでユーザーに妥当性検査情報を要求し、Net.Data マクロがデータベースから検索するデータを使用するか、または Net.Data マクロで定義された関数から外部プログラムを呼び出すことによって、この情報の妥当性検査を行うことができます。

資産の保護に関する詳細な情報は、以下の Web サイトの、インターネット機密保護に関する頻繁に問い合わせのある質問リスト (FAQ) を参照してください。

<http://www.w3.org/Security/Faq>

第4章 Net.Data を起動する

OS/400 用 Net.Data は、コモン・ゲートウェイ・インターフェース (CGI) およびマクロを使用して起動されます。このタイプの呼び出し方式は、マクロ要求と呼ばれます。さらに、永続的マクロ、またはトランザクション境界を指定する関数を含むマクロを、起動することができます。永続的マクロに関する詳細については、109ページの『第7章 永続的マクロによるトランザクション管理』を参照してください。

本章では、マクロによる Net.Data の起動について説明します。

- 『マクロで Net.Data を呼び出す (マクロ要求)』
- 38ページの『永続的マクロの起動』

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。

- 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 実行可能ファイルのパスおよびファイル名。たとえば、`/cgi-bin/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て、MACRO_PATH 初期設定パス変数で定義されたパス・ステートメントとの突き合わせを試行します。詳しくは、15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 つまたは複数の任意指定パラメーターを指定します。

その上でブラウザーで 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 のさまざまな呼び出し方法を示しています。

例 1: HTML リンクを使用して Net.Data を呼び出す

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

例 2: フォームを使用して Net.Data を呼び出す

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

例 3: HTML リンクを使用して、qsys.lib ファイル・システム内の Net.Data マクロを呼び出す

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

例 4: フォームを使用して、qsys.lib ファイル・システム内の Net.Data マクロを起動する

```
<FORM METHOD=POST  
  ACTION="http://server/cgi-bin/db2www/  
  qsys.lib/mylib.lib/myfile.file/myMacro.mbr/report">  
.  
.  
.  
</FORM>
```

以下の節では、HTML リンクおよびフォームと、これらを使用した Net.Data の呼び出し方法の詳細について説明します。

- 37ページの『HTML リンク』
- 37ページの『HTML フォーム』

HTML リンク

Web ページを作成する場合、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「List all monitors (モニターをすべてリストする)」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/cgi-bin/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。

```
%DEFINE DATABASE="MNS97"
```

```
%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、71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。

以下の例は、『HTML リンク』でのモニター・リストの例に基づいていますが、ユーザーはブラウザで、簡単な HTML 形式を使用して製品型を選択し、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="/cgi-bin/db2www/equip1st.d2w/report">
<P>What type of hardware do you want to see?
<MENU>
<LI><INPUT TYPE="RADIO" NAME="hardware" VALUE="MON" checked> Monitors
<LI><INPUT TYPE="RADIO" NAME="hardware" VALUE="PNT"> Pointing devices
<LI><INPUT TYPE="RADIO" NAME="hardware" VALUE="PRT"> Printers
<LI><INPUT TYPE="RADIO" NAME="hardware" VALUE="SCN"> Scanners
</MENU>
```

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

ブラウザで選択し、「Submit (実行)」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、equiplst.d2w マクロの中の HTML レポート・ブロックを実行します。

```
%DEFINE DATABASE="MNS97"

%FUNCTION(DTW_SQL) myQuery(){
SELECT MODNO, COST, DESCRIP FROM EQPTABLE
WHERE TYPE='$(hardware)'
%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 ステートメント内の TYPE=\$(hardware) の値は、HTML 形式入力データから利用することができます。

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。

永続的マクロの起動

このセクションでは、永続的マクロの起動方法について説明します。これらのマクロには、トランザクション処理に使用される関数が含まれます。これらのマクロの起動は、一般のマクロ要求に似ています。ここで、サーバー、マクロ、および HTML ブロックを指定します。永続的マクロの場合は、トランザクション・ハンドルも指定し、HTML ブロックをトランザクションの一部として識別します。

永続的マクロおよびトランザクション処理に関する詳細については、109ページの『第7章 永続的マクロによるトランザクション管理』を参照してください。

永続的マクロの構文

永続的マクロを起動するには、以下の構文を使用します。

- HTML リンク :

```
<A HREF="http://server/Net.Data_invocation_path/transaction_handle/filename/
block/[?name=val&...]">any text</A>
```

- HTML フォーム:

```
<FORM METHOD=method ACTION="http://server/Net.Data_invocation_path/
transaction_handle/filename/block/
[?name=val&...]">any text</FORM>
```

- URL:

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

パラメーター:

server Web サーバーの名前を指定します。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 実行可能ファイルのパスおよびファイル名。たとえば、
/cgi-bin/db2www/ のようになります。

transaction_handle

どの URL が、Net.Data マクロが開始されるトランザクションの一部であるかを指定します。この識別子は、DTW_RTVHANDLE 組込み関数を呼び出して取得し、場所は *Net.Data_invocation_path* の後ろでなければなりません。

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、
15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。

例

以下の例では、永続的マクロの起動方法を示しています。

例 1: マクロの URL :

```
http://www.mycompany.com/cgi-bin/db2www/$(handle)/mymacro.mac/report1
```

例 2: 同一のトランザクションで実行される他のマクロの起動にリンクする、一般的な HTML ブロック

```
@DTW_STATIC()
...
%define handle = ""
@DTW_RTVHANDLE(handle)

%html (report) {
@DTW_ACCEPT(handle)
...
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/
pcgil.mbr/report2">continue</a><br>
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/
pcgil.mbr/quit">quit</a><br>
%}
```


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、以下のことを行います。

- 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6 に示されているように、宣言パートと表示パートという2つの構成部分があります。

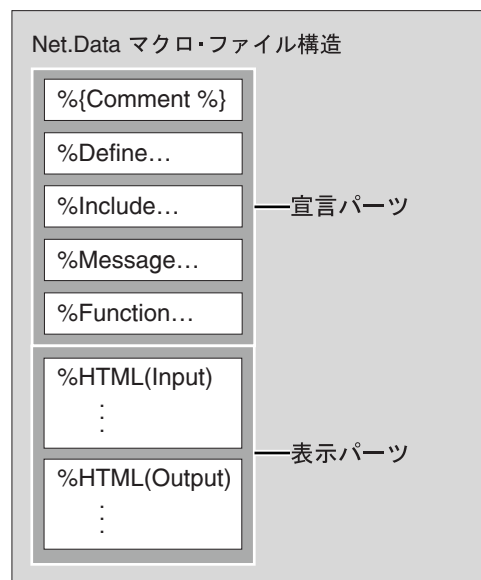


図6. マクロ構造

- 宣言パート 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。
- 表示パート 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。

これらのパートは、複数回、任意の順序で使用することができます。マクロのパート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。

許可のためのヒント： Web サーバーが、確実に、このファイルへのアクセス権限を持つようにします。詳しくは、23ページの『Net.Data がアクセスするオブジェクトへのアクセス権の授与』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。

- 42ページの『Net.Data マクロの分析』
- 47ページの『Net.Data のマクロ変数』

- 59ページの『Net.Data の関数』
- 70ページの『マクロでの Web ページの生成』
- 76ページの『マクロにおける条件付き論理とループ』

Net.Data マクロの分析

マクロは、以下の 2 つのパーツで構成されています。

- 宣言パーツ。これは、表示パーツで使用される定義を含みます。宣言パーツは、以下の 2 つの主要な任意選択のブロックを使用します。
 - DEFINE ブロック
 - FUNCTION ブロック

宣言パーツには、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、他の言語構成要素およびステートメントが含まれる場合もあります。言語構成要素に関する詳細は、*Net.Data* 解説書の言語構成要素の章を参照してください。

許可のためのヒント： Web サーバーが、確実に、EXEC および INCLUDE ステートメントが参照するファイルへのアクセス権限を持つようにします。詳しくは、23ページの『Net.Data がアクセスするオブジェクトへのアクセス権の授与』を参照してください。

- 表示パーツは、マクロからのエン트리・ポイント、およびエグジット・ポイントとして使用される HTML ブロックを使用して、Web ページのレイアウト、参照変数、および呼び出し関数を定義します。Net.Data を起動する際に、HTML ブロック名をマクロ処理のエン트리・ポイントに指定します。HTML ブロックは、45ページの『HTML ブロック』で説明されています。

このセクションでは、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.mbr 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.mbr %}
}%

%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>@rexx1(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。

この 2 つの HTML ブロックには、HTML などのテキスト表示ステートメントを入れます。これにより、Web の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、という 2 つのパラメーターを必要とします。

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。以下のセクションでは、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えます。

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` ブロックの終了を確認するものです。

FUNCTION ブロック

`FUNCTION` ブロックには、`HTML` ブロックが呼び出す関数の宣言が含まれます。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) { <-- この関数は 1 つの
                                                         パラメーターを受け取り、
                                                         外部 REXX プログラムで
                                                         割り当てられた変数
                                                         'result' を返します。
    %EXEC{ompsamp.mbr %} <-- 関数は "ompsamp.mbr" という
                                                         外部 REXX プログラムを実行します。
}%

%FUNCTION(DTW_REXX) today () RETURNS(result) {
    result = date() <-- この関数のただ 1 つのソース・ステートメントは、
                                                         インラインに含まれています。
}%
```

最初の関数ブロック `rexx1` は、`REXX` 関数宣言です。この宣言が次に、`ompsamp .mbr` 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 つの変数を返します。`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 .mbr` 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、`` あるいは `` 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。

2 番目の関数ブロック today も、REXX プログラムを参照します。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、インラインとして扱うことも可能です。

HTML ブロック

HTML ブロックは、Web ページのレイアウトを定義し、変数を参照し、関数を呼び出します。HTML ブロックは、マクロからのエントリ・ポイント、およびエグジット・ポイントとして使用されます。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) { <--- この HTML ブロックの名前を識別します。
<html>
<head>
<title>$(page_title)</title> <--- 変数を置換します。
</head><body>
<h1>Input Form</h1>
Today is @today() <--- この行では関数の呼び出しをします。

<FORM METHOD="post" ACTION="OUTPUT"> <--- このフォームが実行されると
                                     "OUTPUT" HTML ブロックが呼び出されます。
Type some data to pass to a REXX program:
<INPUT NAME="input_data" <--- "input_data" はフォームが実行された時に定義され、
TYPE="text" SIZE="30">      このマクロ内のどこでも参照可能になります。
                             これは、入力フィールドに入れられたユーザー・タイプが
                             何であっても、そのタイプに初期化されます。

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

<hr>
<p>
[
<a href="/">Home page</a>]
</body><html>
%} <--- HTML ブロックを閉じます。
```

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。名前には、任意の英数字、下線、またはピリオドを使用することができます。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、前述の 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="/cgi-bin/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> <--- 再度、置換しています。
</head><body>
<h1>Output Page</h1>
<p>@rex1(input_data) <--- この行では、関数 rex1 を呼び出し、
                           引き数 "input_data" を渡します。

<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 `rex1` 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 マクロの変数を定義したり、参照したり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、および文字列リテラルなどの 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。OS/400 の場合、トークンの最大のサイズは、オペレーティング・システムで決まります。個々の言語環境は、値のサイズに、制限を追加することができます。

変数の型によって、または事前定義値があるかどうかによって、Net.Data の変数を定義することができます。定義方法に基づき、これらの変数を以下の型に分類することができます。

- 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。
- 事前定義の変数。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。

以後のセクションでは、以下について説明します。

- 『識別子の効力範囲』
- 48ページの『変数の定義』
- 50ページの『変数の参照』
- 52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・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

以下の場合、識別子には、関数ブロックの効力範囲があります。

- 識別子が関数定義のパラメーター・リストで宣言される場合。
同じ名前を持つ識別子が、関数定義の外にすでに存在する場合は、Net.Data は、その関数ブロック内の関数パラメーター・リストの識別子を使用します。
- 識別子が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ない場合。

識別子が関数の外で宣言されたり、初期化されたりしている場合、および関数のパラメーター・リストで宣言されていない場合は、その識別子には、関数ブロックの効力範囲はありません。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、表の列名を表す 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にはすべて、関数ブロックの効力範囲があります。

- 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にはすべて、関数ブロックの効力範囲があります。

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オーバーライドします。

- **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。変数名の初めは、必ず文字または下線にします。変数名に使用できるのは、英数字、下線、ピリオド、あるいはハッシュ (#) です。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" %}
```

いくつかの変数を、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>
```

複数行に及ぶ変数、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 multi-line 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、37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/cgi-bin/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 \$(と) の内側に指定します。たとえば、以下のようになります。

```
$(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。変数参照は、ストリング、変数参照、および関数呼び出しを含むことができます。

変数名は動的に生成することができます。リストの番号が拡張機能で決定できない場合には、この手法で、ループを使用して、実行時に作成されるリスト用の、さまざまなサイズの表や入力データを処理することができます。たとえば、HTML 形式要素のリストが生成できます。この要素は、SQL 照会から戻されたレコードに基づいて生成されます。

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。

無効な変数参照：無効な変数参照は、空ストリングに分解されます。たとえば、変数参照に感嘆符 (!) などの無効文字が含まれる場合、参照は空ストリングに分解されます。

有効な変数名は、必ず英数字または下線で始まっている必要があります。変数名に使用できるのは、英数字（ピリオドを含む）、下線、およびハッシュです。

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を判別してください。変数が、それが参照されたときに未定義である場合、Net.Data は空ストリングを返します。変数参照は単独では変数を定義しません。

例 2: 動的に生成される変数参照

任意の数の要素を持つ SQL SELECT ステートメントを実行場合を考えてください。以下の ROW ブロックを使用して、入力フィールドを持つ HTML 形式を作成することができます。

```
...
%ROW {
<input type=text name=@dtw_rconcat("I", ROW_NUM) size=10 maxlength=10>
%}
...
```

入力フィールドを作成したからには、おそらく、その形式を処理用のマクロに実行依頼するときに入力した値に、アクセスしたくなるでしょう。次のようにループをコーディングして、可変長リストの値を検索することができます。

```
<PRE>
...
@dtw_assign(rowIndex, "1")
%while (rowIndex <= rowCount) {
The value entered for row $(rowIndex) is: $(I$(rowIndex))
@dtw_add(rowIndex, "1", rowIndex) %}
...
</PRE>
```

Net.Data はまず、 I(rowIndex)$ 参照を使用して、変数名を生成します。たとえば、最初の変数名は *I1* となります。次に、Net.Data はその値を使用して、変数の値を決定します。

例 3: ネストされた変数参照および関数呼び出しを持つ変数参照

```
%define my = "my"
%define u = "lower"
%define myLOWERvar = "hey"
$\$(my)@dtw_ruppercase(u)var)
```

変数参照は、値 *hey* を返します。

変数の型

マクロでは以下の型の変数が使用できます。

- 『条件変数』
- 『環境変数』
- 53ページの『実行可能な変数』
- 54ページの『隠し変数』
- 55ページの『リスト変数』
- 56ページの『表変数』
- 57ページの『各種変数』
- 57ページの『表処理変数』
- 58ページの『レポート変数』
- 59ページの『言語環境変数』

Net.Data によって特定の方法を定義される変数に、ストリングを割り当てると、ENVVAR、LIST、条件リストなどの変数は、定義された方法ではもはや動作しません。すなわち、このような変数は、ストリングを含む単純変数になります。

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、55ページの『リスト変数』を参照してください。

環境変数

Net.Data 要求を処理している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ための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 `SERVER_NAME` は、以下の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なります。

```
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* を参照にしてください。

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にしてください。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` には、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、以下のように戻します。

```
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` の初期設定ファイルの `EXEC_PATH` で指定されるディレクトリーを検索する。詳細については、16ページの『`EXEC_PATH`』を参照してください。

2. プログラムが見つからない場合は、システムの PATH 環境変数、あるいはライブラリー・リストで定義されるディレクトリーを、システムが検索する。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、システム日付をレポートの一部として戻します。

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

%FUNCTION(DTW_SQL) myQuery() {
SELECT CUSTNO, CUSTNAME from dist1.customer
%REPORT{
%ROW{
<A HREF="/cgi-bin/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 ページ・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、条件変数、隠し変数、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="/cgi-bin/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{
DATABASE="custcity"
%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 は以下のようにになります。

```
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` の後に続く数字は、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` によって提供される組み込み表関数を使用する以外に方法はありません。`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関する詳細については、81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、あるいは `Net.Data` 表関数の使用によって、参照することができます。`REPORT` ブロック内の表の各要素へのアクセスに関しては57ページの『表処理変数』を、表関数を使用して表の各要素にアクセスすることについては68ページの『表関数』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。

表変数を参照すると、表の内容は、DTW_HTML_TABLE 変数の設定に基づき表示およびフォーマットされます。以下の例では、myTable の内容が表示されます。

```
%HTML (output) {  
    $(myTable)  
}
```

表の列名とフィールドの値は、1 を原点に持つ配列要素としてアドレス指定されます。

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。

```
%REPORT {  
<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、空白スペース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。

表処理変数には、Net.Data が判別する事前定義値があります。これらの変数によって、SQL 照会または関数呼び出しの結果セットから値を、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。

```
%REPORT {  
<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>` タグを使用して、表形式で表示されます。出力を表示するための命令をもつ `REPORT` ブロックを定義して、あるいは、デフォルトのレポートが生成されるのを防ぐためのレポート変数の 1 つを使用して、デフォルトのレポートを取り消すことができます。

レポート変数は、Web ページ出力の表示方法、およびデフォルトのレポートと `Net.Data` 表と一緒に使用する方法を、カスタマイズするのに役立ちます。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・フォーマットを取り消し、HTML の表の出力対デフォルトの表出力を指定して、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 がデフォルトの形式設定に 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、データベースへの接続の確立、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 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。

- 60ページの『関数の定義』
- 64ページの『関数の呼び出し』
- 65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： 関数を定義するには、以下の構文を使用してください。

FUNCTION ブロック:

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

MACRO_FUNCTION ブロック :

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

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、64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

パラメーターのデータ型。言語環境プログラムの中には、渡されるパラメーターのデータ型を予想するものがあります。たとえば `SQL` 言語環境は、ストアド・プロシージャを呼び出す際に、それらを予想します。直接呼び出し言語環境が、プログラムを呼び出す際に予想するのと同様です。使用中の言語環境がサポートするデータ型についての詳細は、81ページの『第6章 言語環境の使用』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のようになります。

- 言語環境への呼び出しからの戻りコードがゼロの場合は、`NULL`
- 戻りコードがゼロ以外の場合は、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` の変数参照および `Net.Data` の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、53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 ブロックまたは MACRO_FUNCTION ブロックの出力処理のため、1 つ以上の REPORT ブロックを定義します。71ページの『レポート・ブロック』を参照してください。

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。63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。

関数での特殊文字の使用

構文上有効な組み込みプログラム・コード (REXX または Perl など)として、Net.Data の言語構成要素構文と一致する文字を、関数ブロックの言語ステートメント・セクションで使用すると、これらの文字は、Net.Data の言語構成要素として、間違って解釈され、マクロ内で予測不能な結果が発生する可能性があります。

たとえば、Perl 関数は COMMENT ブロック区切り文字 %{ を使用するかもしれません。マクロが実行されると、文字 %{ は、COMMENT ブロックの先頭と解釈されます。Net.Data は、次に、COMMENT ブロックの終わりを検索し、関数ブロックの終わりを読み取ったときに、検出したと判断します。Net.Data は、次に、関数ブロックの終わりを検索し始めます。そして、関数の終わりを検出できないと、エラーを発行します。

以下の方法の 1 つを使用して、Net.Data が特殊文字として解釈する区切り文字ではなく、COMMENT ブロックの区切り文字を使用するか、あるいはユーザーの組み込みプログラム・コードの一部として Net.Data の特殊文字を使用します。

- コードをインラインに置くのではなく、EXEC ステートメントを使用してプログラム・コードを呼び出す。
- 変数参照を使用して、特殊文字を指定する。

たとえば、以下の Perl 関数では、COMMENT ブロックの区切りを表す文字 %{ が、Perl 言語ステートメントの一部として含まれます。

```
%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する。

```
%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、`+default` または `-default` に対して、ローカルな MESSAGE ブロックを検査する。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。
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、抜け出る。

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 のマクロのパーツを示しています。

```
%{ 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.mbr %}
    %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・テキストを印刷してから、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

ユーザー定義関数および組み込み関数の両方を呼び出すには、Net.Data 関数呼び出しステートメントを使用します。単価記号 (@) 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。

引き数に使用する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。

例 1: テキスト・ストリング引き数による関数呼び出し

```
@myFunction("abc")
```

例 2: 変数および関数呼び出し引き数による関数呼び出し

```
@myFunction(myvar, @DTW_rADD("2","3"))
```

例 3: 変数参照および関数呼び出しを含む、テキスト・ストリング引き数による関数呼び出し

```
@myFunction("abc$(myvar)def@DTW_rADD("2","3")ghi")
```

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。

66ページの図7 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。

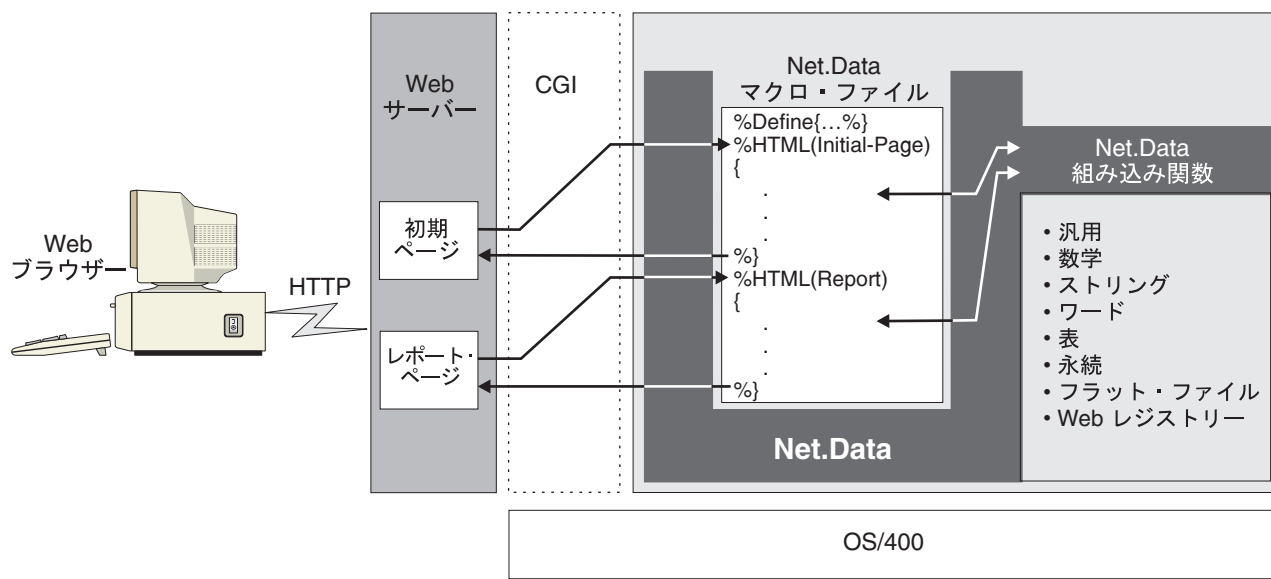


図 7. Net.Data の組み込み関数

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

- **DTW_、DTWF_、および DTWR_:** 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。DTWR_ は、Web のレジストリー関数に対応する接頭部です。)
- **DTW_r、DTWF_r、および DTWR_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。

以下のセクションでは、Net.Data の組み込み関数について高度な概説を提供します。以下の関数を使用して、汎用、数学、String、ワード、あるいは表操作の各関数を実行します。さらに、トランザクション処理に永続関数を使用することができます。各関数とそ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。これらの関数の中には、使用前に変数を設定する必要があるものや、特定のコンテキストで使用しなければならないものがあります。オペレーティング・システムがすべて、個々の組み込み関数をサポートしているわけではありません。ご使用のオペレーティング・システムがサポートしている関数を判別するには、Net.Data 解説書を参照してください。

- 67ページの『汎用関数』
- 67ページの『数学関数』
- 68ページの『String関数』

- 68ページの『ワード関数』
- 68ページの『表関数』
- 69ページの『フラット・ファイル関数』
- 69ページの『Web レジストリー関数』
- 69ページの『永続関数』

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、メール送信、HTTP Cookie の処理、HTML エスケープ・コードの生成、およびシステムからのその他役立つ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>
%}
```

もう 1 つ、役に立つのは、DTW_URLESCSEQ 関数です。これは、URL では許可されていない文字をエスケープ値に置換します。たとえば、入力変数 string1 が "Guys & Dolls" に等しい場合は、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、関数 DTW_POWER は、第 1 パラメーターの値の第 2 パラメーター乗を求め、その結果を戻します。以下に例を示します。


```
@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、その他の有用な関数を実行することができます。

たとえば、DTW_ASSIGN を使用して、入力変数の値を出力変数に割り当てることができます。この関数を使用して、マクロで変数を変更することもできます。以下の例では、変数 RC はゼロに割り当てられます。

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

他のストリング関数としては、ストリングを連結する DTW_CONCAT、特定の位置にストリングを挿入する DTW_INSERT などの多くのものがあります。

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 を使用して、ストリングから指定した数だけワードを削除します。

```
@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 行を追加します。

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

さらに、DTW_TB_DUMP は、表の各行が異なる行に表示されるように、`<PRE></PRE>` タグで囲んでマクロ表変数の内容を戻します。また、DTW_TB_CHECKBOX は、マクロ表変数から、1 つまたは複数の HTML チェック・ボックスの入力タグを戻します。

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の最後に書き込み、DTWF_DELETE はレコードをファイルから削除します。

さらに、FFI 関数では、DTWF_CLOSE および DTWF_OPEN で、ファイルをロックすることができます。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。

Web レジストリー関数

Web レジストリー関数を使用することにより、レジストリーおよびそれに含まれるエントリーを保守することができます。Web のレジストリーとは、*Net.Data* により保守されるキーを持つファイルで、エントリーの追加、検索、および削除を簡単にできるようにしてくれます。

たとえば、DTWR_ADDENTRY はエントリーを追加して、DTWR_DELEENTRY はエントリーを削除します。DTWR_LISTSUB は、OUT 表パラメーターにレジストリー・エントリーに関する情報を戻し、DTWR_UPDATEENTRY は、指定したレジストリー・エントリーの既存の値を新しい値で置換します。

永続関数

永続的なマクロ関数は、単一のトランザクション内における、永続的なマクロ・ブロックの定義を援助することによって、*Net.Data* でのトランザクション処理をサポートします。これらの関数を使用して、トランザクションの開始と終了、トランザクション全体にわたって永続的な HTML ブロック、トランザクションでの変数の有効範囲、および、トランザクションで変更をコミットするかロールバックするかについて定義してください。

たとえば、DTW_ACCEPT はトランザクション・ハンドルを識別し、DTW_TERMINATE はトランザクションの最後の HTML ブロックを識別します。DTW_RTVHANDLE は、トランザクションのブロックに、固有のトランザクション・ハンドルを生成します。DTW_COMMIT および DTW_ROLLBACK を使用して、トランザクション中にコミットを開始してロールバックすることができます。

詳しくは、109ページの『第7章 永続のマクロによるトランザクション管理』を参照してください。有効な永続関数のリストおよびその構文と例については、*Net.Data* 解説書の組み込み関数の章も参照してください。

マクロでの Web ページの生成

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。以下のセクションでは、マクロの HTML および REPORT ブロックについて説明し、Net.Data のマクロにおける 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。各 HTML ブロックは、ブラウザーで単一の 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 や JavaScript などの任意の有効なテキスト表示ステートメントを、HTML ブロックに記述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い方を示しています。

```
%DEFINE DATABASE="MNS96"

%HTML(INPUT){
<H1>Hardware Query Form</H1>
<HR>
<FORM METHOD="POST" ACTION="/cgi-bin/db2www/equip1st.d2w/report">
<d1>
<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
</d1>
<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/cgi-bin/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、「Submit (処理依頼)」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、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。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 ブロックには、次の 3 つの部分があり、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 ブロックの作成時には、以下のガイドラインを使用してください。

- ROW ブロックからの表出力をどれも表示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 により提供される REPORT ブロック内の変数を使用して、Net.Data の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、57ページの『表処理変数』で説明されています。これ以上の詳細については、*Net.Data* 解説書のレポート変数の節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 ブロックの前で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 ブロックの後で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、マクロ言語構成要素として認識できない、ヘッダー、ROW およびフッター・ブロックにおけるすべてのものを、テキスト表示ステートメントとして扱い、これらのステートメントを、ブラウザーに送信します。
- 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印刷させるには、マクロ・ファイルに 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印刷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。

例：

```
%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、Name、Phone、および Fax 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 回印刷する。区切り線の付いたこのテキストは、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 の値で置換する。
3. スtring *Total records retrieved:*、および TOTAL_ROWS の値を、レポートの最後に一度印刷する。(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

単一の FUNCTION または MACRO FUNCTION ブロックで複数の REPORT ブロックを指定して、1 回の関数呼び出しで複数のレポートを生成させることができます。

一般的に、複数の REPORT ブロックは、DTW_SQL 言語環境でストアード・プロシージャーを呼び出す関数に対して使用します。この場合、複数の結果セットが戻ります (99ページの『ストアード・プロシージャー』を参照)。ただし、複数の REPORT ブロックは、複数のレポートを生成させるために、任意の言語環境とともに使用することができます。

複数の REPORT ブロックを使用するには、関数仮引き数リストに Net.Data 表変数を配置します。指定したレポート・ブロックの数より多い結果セットが、ストアード・プロシージャから戻される場合、そしてまた、Net.Data 組み込み関数 DTW_DEFAULT_REPORT = "MULTIPLE" である場合は、レポート・ブロックと関連がない各表に対して、デフォルトのレポートが生成されます。レポート・ブロックが指定されていない場合、および DTW_DEFAULT_REPORT = "YES" である場合は、デフォルトのレポートは 1 つのみ生成されます。SQL 言語環境の場合に限り、DTW_DEFAULT_REPORT 値の YES は、値 MULTIPLE と同等であることに注意してください。

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のフォーマットを使用して、複数のレポートを表示させる

例 1: DTW_SQL 言語環境

```
%DEFINE DTW_DEFAULT_REPORT = "MULTIPLE"
%FUNCTION(dtw_sql) myStoredProc (OUT table1, table2) {
    CALL myproc %}
```

この例では、ストアード・プロシージャ myproc は、2 つの結果セットを戻します。これらは、table1 および table2 に入ります。REPORT ブロックが指定されていないので、デフォルトのレポートは両方の表 (最初に table1、次に table2) について表示されます。

例 2: MACRO_FUNCTION ブロック この例では、2 つの表が MACRO_FUNCTION ブロックに渡されます。DTW_DEFAULT_REPORT="MULTIPLE" が指定されているため、Net.Data は、最初の表に対してのみデフォルトのレポートを表示します。MULTIPLE が指定されている場合は、どちらの表に対してもデフォルトのレポートが生成されます。

```
%DEFINE DTW_DEFAULT_REPORT = "MULTIPLE"
%MACRO_FUNCTION multReport (INOUT tablename1, tablename2) {
    %}
```

この例では、2 つの表が MACRO_FUNCTION multReport に渡されます。ここでも Net.Data は、MACRO FUNCTION ブロック・パラメーター・リストに示されている順序で (最初に table1 に、次に table2 について)、デフォルトのレポートを表示します。

例 3: DTW_REXX 言語環境

```
%DEFINE DTW_DEFAULT_REPORT = "YES"
%FUNCTION (dtw_rexx) multReport (INOUT table1, table2) {
    SAY 'Generating multiple default reports...<BR>'
    %}
```

この例では、2 つの表が REXX 関数 multReport に渡されます。

DTW_DEFAULT_REPORT="YES" が指定されているため、Net.Data は最初の表にのみデフォルトのレポートを表示します。

表示処理用に REPORT ブロックを指定することによって、複数のレポートを表示させる

例 1: 名前付き REPORT ブロック


```

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

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

    %REPORT (table1) {
        ...
        %row { .... %}
        ...
    %}
%}

```

この例では、REPORT ブロックが、FUNCTION ブロック・パラメーター・リストに渡された両方の表に対して指定されています。これらの表は、REPORT ブロックで指定された順序で、最初に table2 に、次に table1 について表示されます。REPORT ブロックで表の名前を指定することによって、レポートが表示される順序を制御することができます。

例 2: 名前のない REPORT ブロック

```

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

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

```

この例では、REPORT ブロックが、FUNCTION ブロック・パラメーター・リストに渡された両方の表に対して指定されています。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、および 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させる

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 = "MULTIPLE"
%FUNCTION(dtw_system) editTables (INOUT table1, table2, table3) {
    %EXEC{ /qsys.lib/mylib.lib/mypgm.pgm %}
    %REPORT(table2) {
        ...
        %ROW { .... %}
        ...
    %}
%}

```

この例では、REPORT ブロックが 1 つしか指定されておらず、表名 table2 が指定されているので、この表を使用してレポートを表示します。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。最初に table1 に対するデフォルトのレポートが、続いて table3 に対するデフォルトのレポートが表示されます。

複数の REPORT ブロックに関するガイドラインおよび制約事項: FUNCTION または MACRO_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 :

- 1 つの結果セットまたは表名につき、1 つ以上の REPORT ブロックが指定できます。REPORT ブロックに指定された名前は、FUNCTION ブロックのパラメーター・リストにおける、対応する結果セットあるいは表名パラメーターと、一致しなければなりません。
- 表を処理したい順序で、複数の表に REPORT ブロックを指定します。
- 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 = "MULTIPLE" を定義します。Net.Data が Web ページを作成すると、そのページには、REPORT ブロックをもつ表のレポートを表示後、表のデフォルトのレポートが表示されます。REPORT ブロックが指定されていない場合に、DTW_DEFAULT_REPORT = "YES" を設定すると、1 つの表に対してのみデフォルトのレポートが生成されます。例外は SQL 言語環境の場合であり、値 YES を指定すると、MULTIPLE と同じ処理が行われます。
- Net.Data が、REPORT ブロックをもたない表を表示しないようにするには、DTW_DEFAULT_REPORT = "NO" を設定します。
- 複数の表を戻す関数とともに DTW_SAVE_TABLE_IN 変数を使用する場合は、関数から戻される最初の表が DTW_SAVE_TABLE_IN 表に割り当てられます。
- いずれの言語環境でも、複数のレポート・ブロックが使用できます。

制約事項 :

- 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 ブロックの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 ブロックの間ではなく、REPORT ブロックのリストの前または後のどちらかに置く必要があります。
- 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。

マクロにおける条件付き論理とループ

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 つまたは複数の条件をテストし、次にその条件リ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』
- 79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 ブロックの規則：IF ブロック構文の規則は、マクロ内のブロックの位置によって決まります。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 ブロック内で指定する場合、HTML ブロックに許可される要素はどれでも、`INCLUDE` ステートメントおよび `WHILE` ブロックなどの IF ブロックで許可されます。

```
%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 ブロックは、宣言パーツ内の他のブロックの外側にある IF ブロック内でネストされる場合には、外側のブロックが使用できる要素ならすべて使用することができます。また、IF ブロック内にある別のブロック内でネストされる場合には、内側にあるブロックの構文規則を採用します。

たとえば、ネストされた IF ブロックは、HTML ブロック内にあるときに使用する規則に従わなければなりません。

```
%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。

IF ブロックのストリング比較

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、整数以外の数字の数値比較はサポートしません。

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      (ブランク文字を含んでいる)
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、それを実行するときなのです。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。

IF ブロックの例： 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 == "AS400")
  %IF (version == "V3R2")
    %INCLUDE "as400v3r2_def.hti"
  %ELIF (version == "V3R7")
    %INCLUDE "as400v3r7_def.hti"
  %ELIF (version == "V4R1")
    %INCLUDE "as400v4r1_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、MACRO_FUNCTION ブロック、および IF ブロック内で指定し、それらを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、条件のコーディング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、SQL ステートメントを DB2 データベースに渡すことが可能になり、REXX 言語環境によって、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、プログラムを実行したり、コマンドを発行したり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。ユーザー作成言語環境は、それぞれ 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、サービス・プログラム として実装されていなければなりません。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8 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。

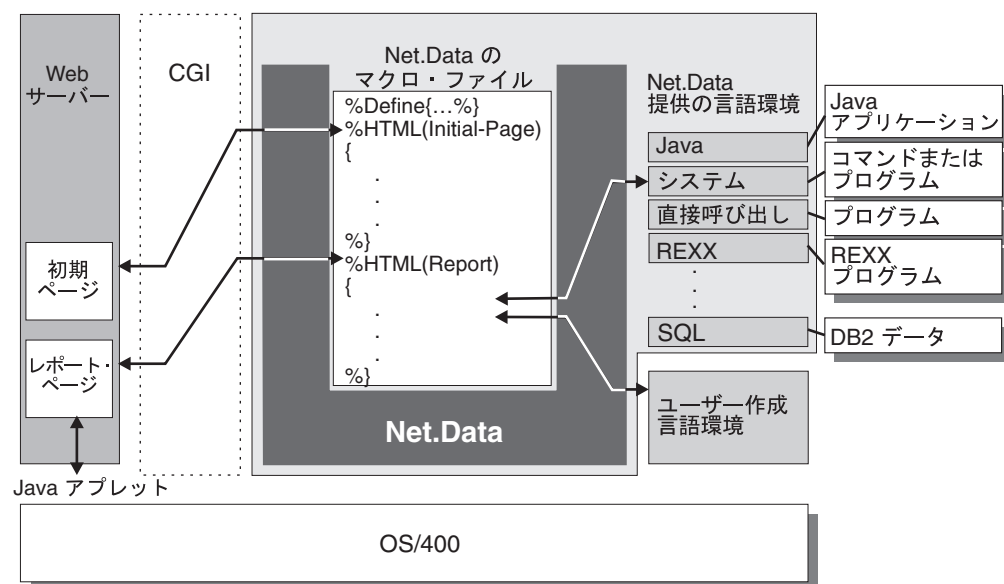


図 8. Net.Data 言語環境

以下のセクションでは、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。

- 82ページの『Net.Data 提供の言語環境の概説』
- 82ページの『言語環境の呼び出し』
- 83ページの『直接呼び出し言語環境』
- 87ページの『Java アプリケーション言語環境』
- 89ページの『REXX 言語環境』
- 91ページの『SQL 言語環境』

- 106ページの『システム言語環境』

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、121ページの『言語環境の最適化』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。

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
直接呼び出し	DTW_DIRECTCALL	直接呼び出し言語環境は、RPG、COBOL、あるいは C/C++ などの高級プログラミング言語を使用して作成された、外部プログラムの呼び出しをサポートしています。
Java アプリケーション	DTW_JAVAPPS	Net.Data は、Java 言語環境により既存の Java アプリケーションをサポートします。
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たりします。
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。
システム	DTW_SYSTEM	システム言語環境は、コマンドの実行と外部プログラムの呼び出しをサポートします。

言語環境の呼び出し

言語環境を呼び出すには、以下のことを行います。

- 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。

例：

```
%FUNCTION(DTW_SQL) custinfo() {
  select customer, custno from customer.data
  %}
...
%HTML(REPORT) {
  @custinfo()
  %}
```

エラー条件の処理

言語環境関数にエラーが検出されると、言語環境はエラー・コードを含む Net.Data の RETURN_CODE 変数を設定します。

以下のリソースを使用して、エラー条件を処理することができます。

- Net.Data が提供する言語環境は、*Net.Data* メッセージに文書化されているエラー・コードを戻します。
- データベース言語環境 (SQL 言語環境など) は、SQLCODE と名付けたデータベース管理システム (DBMS) が戻すエラー・コードに対し、RETURN_CODE を設定します。DBMS が使用する SQLCODE について詳しく知るためには、DBMS のメッセージとコードに関する資料を参照してください。

機密保護

Net.Data を実行しているユーザー ID が、言語環境ステートメントのターゲットが参照できる任意のオブジェクトにアクセスする、正当な権限を持っていることを、確認してください。たとえば、SQL 言語環境が SQL ステートメントを実行し、SQL ステートメントがデータベース・ファイルにアクセスします。だから、Net.Data を実行するユーザー ID は、そのデータベース・ファイルを利用する権限を持っている必要があります。

直接呼び出し言語環境

直接呼び出し言語環境によって、C、RPG、COBOL、CL などの高水準言語を使用して作成されたプログラムを呼び出すことができます。パラメーターをプログラムに渡すことも、パラメーター値をプログラムから受け取ることもできます。また、既存のプログラムと Net.Data との統合が容易にでき、ユーザーが既存のプログラミング・スキルを使用して、複雑なビジネス・ロジックをコーディングすることもできます。

プログラムの呼び出し

プログラムを呼び出すには、直接呼び出し (DTW_DIRECTCALL) 言語環境を使用する関数で、EXEC ステートメントで呼び出されるプログラムへのパスを含む関数を定義します。例：

```
%function(DTW_DIRECTCALL) dc1() {  
    %EXEC { /QSYS.LIB/NETDATA.LIB/MYPGM.PGM %}  
%}
```

EXEC_PATH 構成変数を使用して、プログラムを格納しているディレクトリーへのパスを定義すれば、プログラムへのパスを短縮することができます。EXEC_PATH 構成変数の定義方法を理解するには、16ページの『EXEC_PATH』を参照してください。

サポートされる言語環境変数

直接呼び出し言語環境は DTW_PAD_PGM_PARMS 変数をサポートします。この変数は、プログラムに渡されるパラメーターに、指定される精度までブランクを埋め込むかどうかを指示するものです。この変数の説明、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。パラメーターをプログラムに渡すことに関する詳細は、84ページの『パラメーターをプログラムに渡す』を参照してください。

パラメーターをプログラムに渡す

パラメーターをプログラムに渡すには、関数定義で、パラメーターのデータ型と、渡すパラメーターが入力専用 (IN) なのか、出力専用 (OUT) なのか、それとも入出力 (INOUT) なのかを指定します。

例：

```
%function(DTW_DIRECTCALL) dc2(IN CHAR(3) p1,  
                                INOUT INTEGER p2,  
                                OUT DECIMAL(7,2) p3) {  
    %EXEC { /QSYS.LIB/NETDATA.LIB/MYPGM.PGM %}  
}%
```

上の例では、直接呼び出し言語環境は、文字型変数、整数、およびパック 10 進型変数の 3 つのパラメーターを、プログラム MYPGM に渡しています。呼び出し先プログラムには、50 個までのパラメーターを渡すことができます。プログラムにはデータ型を指定したパラメーターしか渡せません。直接呼び出し言語環境は、パラメーターに対応するストリングを、そのデータ型の内部表記に変換します。次に言語環境は、関数定義で指定された順に、変数の内部表記へのポインターを呼び出し先プログラムに渡します。

変数へのポインターはプログラムに渡されるので、プログラムは変数の値を変更することができます。ただし、プログラムによって変更される OUT または INOUT 変数は、言語環境を呼び出したマクロに反映されます。

サポートされているデータ型

表4 に、直接呼び出し言語環境がサポートしているデータ型をリストします。個々の高水準言語が、すべてのデータ型をサポートしているわけではありません。

表4. 直接呼び出しデータ型

データ型	使用上の注意
CHAR(<i>n</i>) CHARACTER(<i>n</i>) CHARACTER	文字ストリング。 <i>n</i> が指定されている場合、それはゼロより大きい数字でなければなりません。ストリングが指定されていない場合は、1 文字と考えられます。直接呼び出し言語環境から渡されるストリングはすべて NULL で終わるので、言語環境は <i>n</i> +1 バイト (NULL 終止符に 1 バイト) を割り当てます。 <i>n</i> を超えるストリングは切り捨てられます。
VARCHAR(<i>n</i>)	可変長文字ストリング。ここで <i>n</i> は、ゼロより大きく、32740 以下の数となります。ストリングは NULL で終わり、言語環境は <i>n</i> +2+1 バイト (ストリング長の保管に 2 バイト、NULL 終止符に 1 バイト) を割り当てます。 <i>n</i> を超えるストリングは切り捨てられます。ストリングの最初の 2 バイトには、ストリング長 (2 進数値) が含まれています。パラメーターが OUT (出力のみ) と定義されている場合は、ストリング長は、呼び出されるプログラムに変数が渡される前に、ゼロに設定されます。
INTEGER INT	符号付き2 進整数、長さ 4 バイト
SMALLINT	符号付き2 進整数、長さ 2 バイト

表 4. 直接呼び出しデータ型 (続き)

データ型	使用上の注意
FLOAT(p,s)	単精度または倍精度、浮動小数点数単精度の場合、 p は 0 より大きく、25 より小さくなければなりません。倍精度の場合、 p は 24 より大きく、54 より小さくなければなりません。データをストリングなどの表示可能な形式に変換する場合にのみ、精度 (p) と位取り (s) が使用されます。
REAL(p,s)	単精度浮動小数点数 p は 0 より大きく、25 より小さくなければなりません。データをストリングなどの表示可能な形式に変換する場合にのみ、精度 (p) と位取り (s) が使用されます。
DOUBLE (p,s) DOUBLEPRECISION(p,s)	倍精度浮動小数点数 p は 0 より大きく、53 より小さくなければなりません。データをストリングなどの表示可能な形式に変換する場合にのみ、精度 (p) と位取り (s) が使用されます。
NUMERIC(p,s)	精度 p と位取り s を含むゾーン 10 進数。 p の値は 0 より大きく、32 より小さくなければなりません。
DEC(p,s) DECIMAL(p,s)	精度 p と位取り s を含むパック 10 進数。 p の値は 0 より大きく、32 より小さくなければなりません。
DTWTABLE	Net.Data の表を呼び出し先プログラムに渡すために使用される特別なデータ型。直接呼び出し言語環境はポインターを表に渡します。 Net.Data 言語環境インターフェースの表関数を使用して、この表を操作することができます。

数値として定義されるパラメーターには、通貨記号と 3 桁ごとの区切り記号を含めることができます。直接呼び出し言語環境は、数値変数をストリング形式からその内部形式に変換する際に、通貨記号と 3 桁ごとの区切り文字を削除してから、変数をプログラムに渡します。 Net.Data は、それが実行されているプロセスのプロセス属性から、通貨記号、10 進形式、および 3 桁ごとの区切り記号を取得します。

NULL 文字で終了するストリングのパラメーター

DTW_PAD_PGM_PARMS が、構成ファイルあるいはマクロ内で NO に設定されている場合は、直接呼び出し言語環境は、NULL 終止符文字 (値 `x'00'`) を使用して、ユーザーのプログラムにストリング値を渡します。このためユーザーは、コードを作成してストリングを処理する必要があります (ただし、C あるいは C++ のような、NULL で終了するストリングを予期する言語を使用している場合は別です)。

たとえば、パラメーター・フィールドを CHAR(10) と定義しながら、長さが 5 バイトのストリング値を渡す場合、Net.Data は、5 バイト目の文字の次に NULL 終止符を入れます。値 "12345" を CHAR(10) フィールドのストリングとして渡すと、結果は次のようになります。

```
x'F1F2F3F4F500.....'
```

NULL 終止符に続くバイトは未定義です (NULL なのかブランクなのか、断定できません)。

このストリングは NULL で終わり、NULL 終止符の後に、初期化されていないバイトが含まれているので、RPG プログラムや COBOL プログラムでは使用できません。たとえば、比較演算でこのストリングを使用しても、演算は有効な結果を出し

ません。プログラムがストリングに NULL 終止符が含まれることを予期せず、ストリングの末尾に空白が埋め込まれることを予期しているからです。

ユーザーはプログラム内で文字列操作関数を使用して、ストリング値を抽出したり、VARCHAR データ型を使用したりすることができます。この方法では、最初の 2 バイトにストリングの長さが与えられます。

DTW_PAD_PGM_PARMS が、構成ファイルあるいはマクロ内で YES に設定されている場合は、直接呼び出し言語環境は、精度長に達するまで右端に空白を埋め込んで、ユーザーのプログラムにストリング値を渡します。前述と同じ例を使用し、DTW_PAD_PGM_PARMS を YES に設定して、値 "12345" を CHAR(10) フィールドのストリングとして渡すと、次のような結果が得られます。

```
x'F1F2F3F4F5404040404000'
```

ストリングの長さが 5 で、指定精度に達しないので、精度長に達するまで、値の後に空白が挿入されます。RPG などの言語で書かれたプログラムは、今では、NULL で終了するストリングの処理を必要とせず、パラメーターを使用することができます。

パラメーターを渡すときの一般的エラー

以下に示したリストでは、直接呼び出し言語環境を使用して、プログラムを呼び出す際および、プログラムにパラメーターを渡す際に発生する可能性のあるエラーについて説明しています。これらのエラーを避けるためのヒントも提供しています。

パラメーターのミスマッチ・エラー

パラメーターの数と順序が、呼び出し先プログラムのパラメーター・リストにおける数と順序に一致しているか、確認してください。

データ型エラー

指定されたパラメーターのデータ型が、呼び出し先プログラムが予期しているデータ型と一致するか確認してください。直接呼び出し言語環境でサポートされているデータ型が、呼び出し先プログラムを作成するのに使用した高水準プログラミング言語では、サポートされていない場合があります。

長さエラー

パラメーターに対して定義されている長さが正しく、かつ呼び出し先プログラムで指定されている長さと一致していることを確認してください。宣言されている呼び出し先プログラムの長さよりも短い長さを指定すると、記憶域が破壊され、Net.Data が正しく動作しなくなることがあります。

プログラムからの戻り値

C など高水準プログラミング言語には、プログラム呼び出しで整数を戻すことができるものがあります。整数は、関数定義で RETURNS キーワードを指定することによって取得できます。たとえば次のように指定します。

```
%function(DTW_DIRECTCALL) dc3(IN CHAR(3) p1) RETURNS(retval) {  
  %EXEC { /QSYS.LIB/NETDATA.LIB/MYPGM.PGM %}  
  %}
```

関数呼び出しが正常に完了すると、パラメーター *retval* には、プログラムによって戻された値が格納されます。

直接呼び出し言語環境の例

この例では、マクロがプログラムを呼び出し、いくつかのパラメーターを渡します。マクロに続いてプログラムのソースが、RPG および CL で書かれます。呼び出されるプログラムは、2 つの整数パラメーターを受け入れます。そして最初のパラメーター (入力パラメーター) を、2 番目のパラメーター (出力パラメーター) にコピーします。

マクロ :

```
%define ilepgm = "/QSYS.LIB/NETDATADEV.LIB/TDCCLI01.PGM"
%define out1 = "0"

%FUNCTION(DTW_DIRECTCALL) dcFunction(IN INT inp1,
                                      OUT INT outp2)

{ %EXEC { $(ilepgm) %} %}

%HTML(REPORT) {
  @dcFunction("123", out1)
  The value of out1 is: "${out1}"
  %}
```

ILE RPG プログラム :

```
DINP1          S          10I00
DOUTP2         S          10I00
C*
C      *ENTRY      PLIST
C              PARM          INP1
C              PARM          OUTP2
C*
C              Z-ADD      INP1      OUTP2
C*
C              SETON                      LR
```

CL プログラム :

```
PGM PARM(&INP1; &OUTP2;)
DCL VAR(&INP1;) TYPE(*CHAR) LEN(4)
DCL VAR(&OUTP2;) TYPE(*CHAR) LEN(4)
CHGVAR VAR(&OUTP2;) VALUE(&INP1;)
ENDPGM
```

Java アプリケーション言語環境

Java アプリケーション言語環境によって、Java プログラムを呼び出し、Java アプリケーションと Net.Data を容易に統合することができます。Java アプリケーション言語環境は、OS/400 V4R4 で初めて導入されました。

Java アプリケーション言語環境を使用するためには、21ページの『Java アプリケーション言語環境のセットアップ』で説明されている、構成ステップを完了してください。

java プログラムの呼び出し

Java プログラムを呼び出すには、Java アプリケーション (DTW_JAVAPPS) 言語環境を使用する関数を定義します。Java プログラムのクラス名を示す関数名を指定してください。

例：Java プログラム helloWorld.java の呼び出し

```
%function(DTW_JAVAPPS) helloWorld() { %}
```

Java アプリケーション言語環境は、Java プログラムに、Java プログラムで実行される最初のメソッドである 'main' の、メソッド識別子が含まれていると予期しています。言語環境がアプリケーションを呼び出す場合、そのアプリケーションは stdin および stdout へのアクセス権を持っています。stdin には形式データがありません。Net.Data がすでにデータを読み取っているためです。

重要：Java アプリケーションを呼び出す前に、Java クラスが検出できるように、DTW_JAVA_CLASSPATH パス構成変数を設定してください。この変数の構文については、19ページの『DTW_JAVA_CLASSPATH』を参照してください。

Java プログラムにパラメーターを渡す

Java プログラムにパラメーターを渡すには、関数定義上で、渡されるパラメーターを指定します。入力のみ (IN)、あるいは入力または出力 (INOUT) の、ストリング・パラメーターのみを指定してください。

例：機能呼び出し上で IN パラメーター p1 を渡す

```
%function(DTW_JAVAPPS) jv1(IN p1) { %}
```

Java アプリケーション言語環境は、変数を更新する Java プログラムはサポートしません。更新値をマクロに戻すことができないためです。

Java アプリケーション言語環境の例

この例では、Net.Data マクロが Java プログラム echoString を呼び出します。このマクロは、2 つのストリング・パラメーターを Java 言語環境に渡します。最初のストリングは Java プログラムに、2 番目のパラメーター (テキスト・ストリング) の強調表示にイタリックを使用するか、太字を使用するかを指示した後、2 番目のパラメーターを標準出力 (stdout) で印刷します。イタリックの場合、プログラムは "I" を渡すので、Web サーバーは、*Hello World* をイタリックでブラウザーに表示します。マクロに続いて、Java プログラムのソースが表示されます。

マクロ：

```
%FUNCTION(DTW_JAVAPPS) echoString(textAttribute, text){ %}  
  %HTML(runjava){  
    @echoString("I","Hello World")  
  %}
```

Java プログラム：

```
class echoString {  
  public static void main (String args[]) {  
    if (args[0].equals("I"))
```



```

        System.out.println("<I>" + args[1] + "</I>");
    else
        System.out.println("<B>" + args[1] + "</B>");
    }
}

```

REXX 言語環境

REXX 言語環境を使用すると、REXX プログラムを実行することができます。

REXX プログラムの実行

REXX 言語環境を使用すると、インライン REXX プログラムまたは外部 REXX プログラムの両方を実行することができます。インライン REXX プログラムは、マクロ内に REXX プログラムのソースがある REXX プログラムです。外部 REXX プログラムでは、外部ファイルに REXX プログラムのソースがあります。

インライン REXX プログラムを実行するには、以下のようにします。

REXX (DTW_REXX) 言語環境を使用する関数で、関数内に REXX コードを含む関数を定義します。

例：インライン REXX プログラムを含む関数

```

%function(DTW_REXX) helloWorld() {
    SAY 'Hello World'
%}

```

外部 REXX プログラムを実行するには、以下のようにします。

REXX (DTW_REXX) 言語環境を使用する関数で、REXX ステートメントで実行される EXEC プログラムを含む関数を定義します。

例：外部プログラムを指す EXEC ステートメントを含む関数

```

%function(DTW_REXX) externalHelloWorld() {
%EXEC{ /QSYS.LIB/REXX.LIB/REXXSRC.FILE/HELLOWORLD.MBR%}
%}

```

EXEC_PATH 構成変数を使用して、プログラムを格納しているディレクトリへのパスを定義すれば、プログラムへのパスを短縮することができます。EXEC_PATH 構成変数の定義方法を理解するには、16ページの『EXEC_PATH』を参照してください。

制約事項：OS/400 V3R2 または V3R7 を実行しており、REXX プログラムでは SAY REXX 命令を使用して stdout に書き込みを行う場合、ストリングの開始部分に 12 のブランクを挿入します。たとえば、以下のような場合です。

```

SAY '          STARTOFDATA'

```

12 のブランクは無視されますが、これを挿入しないと、予期せぬ結果が起こる場合があります。

パラメーターを REXX プログラムに渡す

REXX (DTW_REXX) 言語環境によって呼び出される REXX プログラムに情報を渡すには、直接的な方法と間接的な方法の 2 つがあります。

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、以下のような場合です。

```
%FUNCTION(DTW_REXX) rexx1() {  
  %EXEC{  
    /QSYS.LIB/NETDATA.LIB/QREXSRG.FILE/CALL1.MBR $(INPARM1) %}  
}
```

Net.Data 変数 INPARM1 が参照解除され、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。この方法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 プログラムは、起動されると、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。ある場合は、言語環境はこの関数パラメーターに対応する値を変数プールから検索し、この新しい値で関数パラメーター値を更新します。Net.Data は、制御を受け取ると、REXX 言語環境から取得した新しい値ですべての OUT または INOUT パラメーターを更新します。たとえば、以下のような場合です。

```
%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、@double_func の呼び出しによって、a および b の 2 つのパラメーターが渡されます。REXX 関数 double_func は、第 1 パラメーターを 2 倍にし、その結果を第 2 パラメーターに保管します。Net.Data がマクロを呼び出すと、b の値は 6 になります。

Net.Data の表を REXX プログラムに渡すことができます。REXX プログラムでは、Net.Data マクロの表パラメーターの値には REXX の stem 変数としてアクセスします。REXX プログラムでは、列見出しおよびフィールド値

は、表名および列番号で識別される変数に含まれます。たとえば、表 myTable では、列見出しは myTable_N.j であり、フィールド値は myTable_N.i.j です。ここで、i は行番号、j は列番号です。この表の行数は myTable_ROWS であり、列数は 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いて組み込み関数 DTW_TB_TABLE() が呼び出され、HTML 表を生成してブラウザーに戻します。

```
%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)
}%
```

結果は以下のとおりです。

COL1	COL2
[1 1]	[1 2]
[2 1]	[2 2]
[3 1]	[3 2]

SQL 言語環境

SQL 言語環境によって、SQL ステートメントをデータベース管理システム (DBMS) に送信し、実行することが可能になります。

SQL 言語環境を使用するには、必ず、21ページの『言語環境のセットアップ』で説明されている構成ステップに従ってください。

SQL ステートメントの実行

ユーザーは、動的 SQL がサポートする、任意の SQL ステートメントを実行することができます。

SQL ステートメントを実行するには、SQL (DTW_SQL) 言語環境を使用し、言語環境実行可能セクションに SQL ステートメントを含む関数を定義します。

例：SQL SELECT ステートメントを実行する SQL 関数

```
%function(DTW_SQL) getOrders() {  
    SELECT cust, custid, custorder FROM mylibrary.customers  
%}
```

コミットメント制御

SQL 言語環境は、デフォルトではコミットメント制御のもとで実行され、コミットメント制御に適用されるすべての規則に従います。

- SQL ステートメントが SELECT の場合を除き、DTW_SQL を介してアクセスするファイルおよび表をすべてジャーナル処理する。
- Net.Data 初期設定ファイルで DTW_SQL_ISOLATION を指定することによって、コミットメントのレベルを任意選択で変更する。SQL 言語環境がサポートしている分離レベルの詳細については、12ページの『DTW_SQL_ISOLATION: DB2 の分離変数』を参照してください。

トランザクション管理の詳細については、98ページの『Net.Data アプリケーションにおけるトランザクション管理』を参照してください。

OUT および INOUT の表

関数定義上で OUT または INOUT の Net.Data の表を指定する場合、および SQL ステートメントが結果セットを戻す場合は、SQL 言語環境は、個々の結果セットを指定された表に保管します。そのようにしておく、ユーザーはあとでその表をマクロに使用することができます。OUT の表が指定されていない場合は、SQL 言語環境はデフォルトの表を使用します。

ネストされた SQL ステートメント

別の SQL 関数の ROW ブロック内から、他の SQL 関数を呼び出すことができます。SQL 関数では、それぞれ固有の Net.Data の表名を使用してください。使用しない場合には、予測不能な結果が発生することがあります。

例：別の SQL 関数の ROW ブロックから SQL 関数を呼び出す場合

```
%define mytable1 = %TABLE  
%define mytable2 = %TABLE  
%FUNCTION(DTW_SQL) sql2 (IN p1, OUT t2) {  
    select * from NETDATA.STAFFINF where projno='$(p1)'  
%REPORT {  
    %ROW { $(N1) is $(V1) %}  
    %}  
%}  
%FUNCTION(DTW_SQL) sql1 (OUT t1) {  
    select * from NETDATA.STAFFINF  
%REPORT {  
    %ROW { @sql2(V1, mytable2) %}  
    %}  
%}  
%HTML(netcall1) { @sql1(mytable1) %}
```

サポートされる言語環境変数

SQL 言語環境は、DB2 をサポートするために設計された変数をサポートします。たとえば DATABASE 変数は、SQL 言語環境が SQL ステートメントを実行する場合に接続するデータ・ソースを指定します。以下のリストは、SQL 言語環境がサポートする変数を指定しています。これらの変数の説明、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。

- DATABASE
- DB_CASE
- DTW_EDIT_CODES
- DTW_PAD_PGM_PARMS
- DTW_SET_TOTAL_ROWS
- LOGIN
- NULL_RPT_FIELD
- PASSWORD
- SHOWSQL
- SQL_STATE
- TRANSACTION_SCOPE

サポートされているデータ型

SQL 言語環境は、表5 にリストされているデータ型をサポートします。

表 5. データ型

BLOB(1)	DOUBLE	SMALLINT
CHAR	DOUBLEPRECISION	TIME
CLOB(1)	FLOAT	TIMESTAMP
DATE	GRAPHIC	VARCHAR
DBCLOB(1)	INTEGER	VARGRAPHIC
DECIMAL	REAL	

(1) これらのデータ型を、パラメーターとしてストアド・プロシージャ呼び出しに渡すことはできません。ストアド・プロシージャがサポートしているデータ型については、100ページの『ストアド・プロシージャの構文』を参照してください。

LOB および DATALINK データ型に特有の考慮事項を知るには、94ページの『データ型の考慮事項』を参照してください。

SQL 言語環境の制約事項

環境を計画する際には、以下の制約事項を考慮してください。

- 以下の条件の少なくとも 1 つに当てはまる場合は、SQL 言語環境を使用しないでください。
 - データベースのアクセス・クラス・ライブラリー、または SQL コール・レベル・インターフェース (CLI) を使用する、ユーザー定義の言語環境を作成し、その言語環境をマクロで参照する。
 - SQL CLI を使用するアプリケーションを、Net.Data と同じプロセスで実行する。

- 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。
- ローカルまたはリモートのデータベース接続を、最大 50 まで使用することができます。多重接続を使用する場合は、次の制約事項を考慮に入れてください。
 - Net.Data では、同一のリモート・データベースに同時に接続することはできません。
 - TRANSACTION_SCOPE=MULTIPLE (デフォルト) の場合は、リモート・データベースへのアクセス後にログイン ID を変更することはできません。98ページの『Net.Data アプリケーションにおけるトランザクション管理』を参照してください。

これらの制約事項についての詳細は、99ページの『複数のデータベース接続の管理』を参照してください。

データ型の考慮事項

SQL 言語環境がサポートする以下のデータ型には、特別な考慮事項が必要です。

- 『ラージ・オブジェクトを使用する』
- 96ページの『結果セットでの DataLink URL のエンコード』

ラージ・オブジェクトを使用する

ラージ・オブジェクト・ファイル (LOB) を DB2 データベースに保管し、ユーザーの Web アプリケーションの SQL 言語環境を使用してそれらのファイルにアクセスすることができます。

SQL 言語環境では、SQL 照会が結果セットで LOB を戻す場合は、変数 (V1 または V2 など) を処理する Net.Data 表または Net.Data 表フィールドに、ラージ・オブジェクトを保管しません。代わりに、Net.Data に LOB がある場合は、Net.Data で作成されるファイルにその LOB を保管します。このファイルは、HTML_PATH パス構成変数で指定されるディレクトリにあります。変数を処理する Net.Data 表フィールドおよび表の値は、そのファイルのパスに設定されます。そのファイルにアクセスできるのは、Net.Data を実行しているユーザー ID に限定されます。

LOB を保管するファイルの名前は動的に構成され、その形式は以下のようになります。

name[*.extension*]

変数:

name ラージ・オブジェクトを識別する固有のストリング

extension

オブジェクトの型を識別するストリング。CLOB および DBCLOB の場合、拡張子は 'txt' です。BLOB の場合、SQL 言語環境では、LOB が表す内容を示す LOB ファイルの最初の数バイトのシグニチャーを捜して、拡張子を判別しようとします。SQL 言語環境では、以下の型のデータ (括弧内は使用する拡張子) を認識します。

- Bitmap イメージ (.bmp)

- GIF (.gif)
- JPEG イメージ・ファイル (.jpg)
- Tagged image file format (.tif)
- Postscript (.ps)
- MIDI オーディオ・ファイル (.mid)
- AIFF オーディオ・ファイル (.aif)
- AVI オーディオ・ファイル (.avi)
- Basic オーディオ・ファイル (.au)
- Real オーディオ・ファイル (.ra)
- Windows オーディオビジュアル・ファイル (.wav)
- PDF (.pdf)
- Midi 順次ファイル (.rmi)

BLOB のオブジェクトの型が認識されないと、ファイル名には拡張子が付加されません。

LOB がマクロ・ファイル内で参照されている場合、SQL 言語環境では、以下の構文を使用して、LOB ファイル名の前に /tmplobs/ のストリングが付いたファイル名を戻します。

```
/tmplobs/name.[extension]
```

計画のためのヒント：照会が LOB を戻すと、HTML_PATH パス構成変数で指定されたディレクトリーにファイルが作成されます。LOB を使用する場合、リソースがすぐに消費されるため、システムの制限を考慮してください。定期的にディレクトリーを整理するバッチ・プログラムを作成するか DataLinks を使用することをお勧めします。これによって、SQL 言語環境でディレクトリーにファイルを保管する必要性がなくなり、パフォーマンスが向上して、システム資源の使用量が少なくなります。

例：アプリケーションでは MPEG オーディオ (.MPA) ファイルを使用するため、アプリケーション・ユーザーはファイル名をクリックしてビューアーを起動する必要があります。SQL 言語環境ではこのファイル型を認識しないため、EXEC 変数を使用してファイルに拡張子を追加します。

```
%DEFINE{
lobpath = "@DTW_RGETINIDATA("HTML_PATH")"
filename = "@DTW_RREPLACE($(V3), "/tmplobs/", "", "1", "F")"
myFile=%EXEC "REN '${lobpath}/${filename}' '${filename}.mpa'"
}%

%{ where rename is the rename command on your operating system %}
%FUNCTION(DTW_SQL) queryData() {
SELECT Name, IDPhoto, Voice FROM RepProfile
%REPORT{
<P>Here is the information you selected:<P>
%ROW{
$(myFile)
$(V1) Voice sample <IMG SRC="$(V2)">
<A HREF="$(V3).mpa">Voice sample</A><P>
}%
}%
%}
```



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

queryData 関数は以下の HTML 出力を戻します。

```
<P>Here are the images you selected:<P>
Kinson Yamamoto
<IMG SRC="/tmplobs/p2345n1.gif">
<A HREF="/tmplobs/p2345n2.mpa">Voice sample</A><P>
Merilee Lau
<IMG SRC="/tmplobs/p2345n3.gif">
<A HREF="/tmplobs/p2345n4.mpa">Voice sample</A><P>
```

前述の例の REPORT ブロックでは、暗黙のテーブル変数 V1、V2、および V3 を使用しています。

- V1 は個人名で、平文です。
- V2 は .GIF ファイル形式の個人の写真です。イメージはインライン表示されます。SQL 言語環境により、/tmplobs/ の接頭部と .GIF の拡張子が自動的に組み込まれます。
- V3 は .MPA ファイル形式の個人の音声サンプルです。SQL 言語環境に MPA ファイルなどの認識できない形式がある場合、そのファイルを、HTML_PATH 構成変数で指定されたディレクトリーに拡張子なしで書き込みます。この例では、EXEC 変数を追加して拡張子を追加することにより、このファイル・タイプを処理する方法を示しています。変数 \$(V3) が解決すると、ファイル名の前にパス /tmplobs/ が追加されます。たとえば、/tmplobs/sound2a のようになります。この例では、EXEC 変数が REN コマンドを使用してファイル名を変更し、拡張子 .mpa をファイルに追加しています。ファイル名を変更できるようにする前に、DTW_RGETINIDATA 関数を使用して、HTML_PATH で指定されたパスを検索することによって、/tmplobs/ がファイル名から除去され、変更するファイルへのフル・パス名が検索されます。音声サンプルは、アプリケーション・ユーザーが音声サンプルをクリックすると再生されます。

LOB のためのアクセス権 : Web サーバーが実行されているユーザー ID が、HTML_PATH で指定されたディレクトリーに書き込みアクセスできることを確認してください。

結果セットでの DataLink URL のエンコード

DataLink データ型は、データベース・ファイルに保管できるデータの型を拡張するための、基本作成ブロックです。DataLink の場合、列に保管される実際のデータは、ファイルを指すポインターだけです。このファイルは、イメージ・ファイル、音声記録方式、またはテキスト・ファイルのいずれの型のファイルでも構いません。DataLinks は URL を保管して、ファイルのロケーションを解決します。

DATALINK データ型については、DataLink ファイル・マネージャーを使用する必要があります。DataLink ファイル・マネージャーの詳細については、使用しているオペレーティング・システムの DataLinks の資料を参照してください。DATALINK データ型を使用する前に、Web サーバーが、DB2 ファイル・マネージャー・サーバーの管理するファイル・システムにアクセスできることを確認します。

SQL 照会が DataLinks を使って結果セットを戻し、その DataLink 列が、READ PERMISSION DB DataLink オプションをもつ FILE LINK CONTROL を使って作成されている場合は、DataLink 列のファイル・パスにはアクセス・トークンが含まれています。DB2 はそのアクセス・トークンを使用して、ファイルへのアクセスを認証します。このアクセス・トークンがない場合は、ファイルにアクセスしようとしても、権限違反のために、すべて失敗します。ただし、アクセス・トークンには、ブラウザーに戻される URL では使用できない文字 (セミコロン (;) 文字など) が組み込まれている可能性があります。たとえば、以下のような場合です。

```
/datalink/pics/UN1B;0YPVKG346KEBE;baibien.jpg
```

URL にはセミコロン (;) が含まれているため、これは無効です。有効な URL にするには、Net.Data の組み込み関数 DTW_URLESCSEQ を使用してセミコロンをエンコードしなくてはなりません。ただし、この関数は (/) もエンコードするため、関数を適用する前に行う必要のあるストリング処理もあります。

Net.Data MACRO_FUNCTION を記述してストリング処理を自動化し、DTW_URLESCSEQ 関数を使用することができます。DATALINK データ型の列からデータを検索するマクロについてそれぞれこの方法を用います。

例 1: DB2 UDB から戻された URL のエンコードを自動化する MACRO_FUNCTION

```
%{ TO DO: Apply DTW_URLESCSEQ to a DATALINK URL to make it a valid URL.
  IN: DATALINK URL from DB2 File Manager column.
  RETURN: The URL with token portion is URL encoded
%}
%MACRO_FUNCTION encodeDataLink(in DLURL) {
  @DTW_rCONCAT( @DTW_rDELSTR( DLURL,
    @DTW_rADD(@DTW_rLASTPOS("/", DLURL), "1" ) ),
    @DTW_rURLESCSEQ( @DTW_rSUBSTR(DLURL,
    @DTW_rADD( @DTW_rLASTPOS("/", DLURL), "1" ) ) ) )
%}
```

この MACRO_FUNCTION を使用すると、URL は正しくエンコードされ、DATALINK 列に指定されたファイルはいずれの Web ブラウザーにおいても参照可能となります。

例 2: DATALINK URL を戻す SQL 照会を指定する Net.Data マクロ

```
%FUNCTION(DTW_SQL) myQuery() {
  select name, DLURLCOMPLETE(picture) from myTable where name like '%river%'
  %REPORT{
    %ROW{
      <p> $(V1) <br>
      Before Encoding: $(V2) <br>
      After Encoding: @encodeDataLink($(V2)) <br>
      Make HREF: <a href="@encodeDataLink($(V2))"> click here </a> <br> <p>
    %}
  %}
%}
```

DataLink ファイル・マネージャーの関数が使用されることに注意してください。関数 dlurlcomplete は完全な URL を戻します。

Net.Data アプリケーションにおけるトランザクション管理

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戻します。

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、以下のような場合です。

```
@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、以下のような場合です。

```
@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()
    ...
}%

```

複数のデータベース接続の管理

ローカル・データベースまたはリモート・データベースは、一度に 50 個まで接続することができます。SQL 言語環境は、Net.data が実行されている Web サーバー・プロセスの存続期間中は、常に接続を活動状態にしています。接続を活動状態にしておくことにより、データベースへの初期接続後のデータベース・アクセスを高速に行うことができます。以下の事項を考慮に入れると、エラーを回避することができます。

- Net.Data では、同一のリモート・データベースに同時に接続することはできません。あるユーザー ID (LOGIN SQL 言語環境パラメーター) を使用した、リモート・データベースへの接続が存在する場合に、別のユーザー ID を使用して、同じリモート・データベースへの接続を再び要求すると、SQL 言語環境は、最初に既存の接続を切断して (コミットメント制御を使用している場合には) コミットを行い、「新しい」ユーザー ID およびパスワードを使用して接続を再確立しなければなりません。接続が切れた場合には、マクロ内で後でエラーが起こった場合にロールバックを行う方法がないため、コミットする必要があります。
- TRANSACTION_SCOPE=SINGLE の場合には、リモート・データベースへのアクセス後にログイン ID を変更することができます。SQL 言語環境は、既存の接続を切断してコミットを行ってから、新しいユーザー ID およびパスワードを使用して接続を再確立します。
- TRANSACTION_SCOPE=MULTIPLE (デフォルト) の場合は、リモート・データベースへのアクセス後にログイン ID を変更してはなりません。SQL 言語環境は自動的にロールバックを行い、SQL_CODE の -752 を戻します。このコードは、接続を変更できなかったことを示します。

ストアド・プロシージャ

ストアド・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、以下のトピックについて説明します。

- 100ページの『ストアド・プロシージャの構文』

- ・ 『ストアード・プロシージャの呼び出し』
- ・ 102ページの『パラメーターを渡す』
- ・ 102ページの『結果セットの処理』

ストアード・プロシージャの構文

ストアード・プロシージャの構文は FUNCTION ステートメント、CALL ステートメント、および REPORT ブロック (任意選択) を使用します。

```
%FUNCTION function_name ([IN datatype arg1, INOUT datatype arg2,
    OUT tablename, ...]) {
    CALL stored_procedure
[%REPORT [(resultsetname)] { %}]
..
[%REPORT [(resultsetname)] { %}]
[%MESSAGE %]}

%}
```

変数:

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 (表6を参照)。パラメーター・リストで指定されるデータ型は、ストアード・プロシージャ内のデータ型と一致しなければなりません。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、データベースの資料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
 ストアード・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、関数仮引き数リストの *tablename* に一致する必要があります。

表 6. ストアード・プロシージャのデータ型

CHAR	FLOAT	SMALLINT
DATE	GRAPHIC	TIME
DECIMAL	INTEGER	TIMESTAMP
DOUBLE	REAL	VARCHAR
DOUBLEPRECISION		VARGRAPHIC

ストアード・プロシージャの呼び出し

1. ストアード・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ード・プロシージャに指定します。これには、ストアード・プロシージャから戻される結果セットの結果セット名が含まれます。他のストアード・プロシージャからのテーブル名または結果セットとして (IN または INOUT パラメーターとして) 指定することもできます。

```
%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、ストアード・プロシージャ名を識別します。

```
CALL stored_procedure
```

4. ストアード・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 {  
..  
%}
```

例 :

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) arg1 OUT mytable) {  
    CALL myproc  
    %REPORT {  
        ...  
        %ROW { ... %}  
        ...  
    %}  
%}
```

5. ストアード・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なります。

- 結果セットを FUNCTION ステートメントの OUT パラメーターに指定します。結果セットはローカル表として保管されます。

```
%FUNCTION (DTW_SQL) function_name (OUT tablename, ...)
```

- 任意選択で 1 つ以上の REPORT ブロックを指定し、Net.Data が結果セットを表示する方法を定義します。

```
%REPORT(resultsetname1) {  
..  
%}
```

例 :

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) arg1, OUT table1, table2) {  
    CALL myproc  
    %REPORT (table1) {  
        ...  
        %ROW { ... %}  
        ...  
    %}  
    %REPORT (table1) {  
        ...  
        %ROW { ... %}  
        ...  
    %}  
%}
```


パラメーターを渡す

ストアード・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、ストアード・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、ストアード・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、ストアード・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。ストアード・プロシージャに定義したリストのパラメーターが `CHAR(30)` 型の場合、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: ストアード・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: ストアード・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

ストアード・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。ストアード・プロシージャが複数の結果セットを生成する場合、ストアード・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、`FUNCTION` ステートメントにパラメーターを指定して行います。結果セットに指定する名前を `REPORT` ブロックまたは `Net.Data` 表と関連付け、`Net.Data` が各結果セットを処理する方法を決めることができます。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。

結果セットは常にローカル・テーブルに保管され、マクロの他の関数からもデータにアクセスできるようにされます。たとえば、`Net.Data` 表を別の関数に渡して、そのデータを計算に使用し、その計算に基づいて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、76ページの『複数の `REPORT` ブロックに関する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 (OUT tablename) {
    CALL stored_procedure
%}
```

たとえば、以下のような場合です。

```
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1) {
    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 (OUT tablename) {
    CALL stored_procedure [(resultsetname)]
    %REPORT [(resultsetname)] {
        ...
    %}
%}
```

たとえば、以下のような場合です。

```
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1) {
    CALL myproc
    %REPORT {
        ...
        %ROW { ... %}
        ...
    %}
%}
```

代わりに、以下の構文を使用することもできます。

```
%FUNCTION (DTW_SQL) function_name () {
    CALL stored_procedure

    %REPORT () {
        ...
    %}
%}
```

たとえば、以下のような場合です。

```
%FUNCTION (DTW_SQL) mystoredproc () {
    CALL myproc
    %REPORT {
        ...
        %ROW { ... %}
        ...
    %}
%}
```

複数の結果セットを戻し、それをデフォルトのレポート形式設定を使用して表示するには、以下のようにします。

以下の構文を使用します。

```
%FUNCTION (DTW_SQL) function_name (OUT tablename1, tablename2) {
    CALL stored_procedure
%}
```

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、以下のような場合です。

```
%DEFINE DTW_DEFAULT_REPORT = "YES"
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1, mytable2) {
    CALL myproc
%}
```

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には、以下のようにします。

それぞれの結果セットは、1 つまたは複数の **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 (OUT tablename1, tablename2, ...) {
    CALL stored_procedure
    %REPORT (tablename1)
    ...
    %ROW { ... }
    ...
%}
%REPORT (tablename2)
    ...
    %ROW { ... }
    ...
%}
..
%}
```

たとえば、以下のような場合です。

```
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1, mytable2) {
    CALL myproc

    %REPORT(mytable1) {
        ...
        %ROW { ... }
        ...
    %}

    %REPORT(mytable2) {
        ...
        %ROW { ... }
        ...
    %}
%}
```

SQL 言語環境の例

以下の例は、DTW_SQL 関数定義を持つ、SQL ストアード・プロシージャを呼び出すマクロを示しています。この例は、データ型が異なる 3 つのパラメーターを持っています。DTW_SQL 言語環境は、各パラメーターの文字ストリングの値を正しい内部形式に変換して、参照によって各パラメーターを SQL ストアード・プロシージャに渡します。SQL ストアード・プロシージャが処理を完了すると、更新された内部表記が文字ストリングに変換され、対応するパラメーターに格納されます。

```
%{*****
*****%}
DEFINE {
    MACRO_NAME      = "TEST ALL TYPES"
    DTW_HTML_TABLE  = "YES"
    Procedure       = "NDLIB.TESTTYPE"
    parm1           = "1"           %{SMALLINT      %}
    parm2           = "11"          %{INT          %}
    parm3           = "1.1"         %{DECIMAL (2,1) %}
```

```

%}

%FUNCTION(DTW_SQL) CRTPROC(){
  CREATE PROCEDURE $(Procedure)
  ( INOUT SMALLINT,
    INOUT INT,
    INOUT DECIMAL(2,1))
  EXTERNAL NAME $(Procedure) LANGUAGE C SIMPLE CALL
  %MESSAGE{
    default : "$(DTW_DEFAULT_MESSAGE) : continuing.<br>": continue
  %}
%}

%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>
%}

```

システム言語環境

システム言語環境では、コマンドの実行と外部プログラムの呼び出しをサポートしています。

コマンドの発行およびプログラムの呼び出し

コマンドを発行するには、EXEC ステートメントで発行されるコマンドへのパスを含むシステム (DTW_SYSTEM) 言語環境を使用する関数を定義します。たとえば、以下のような場合です。

```
%FUNCTION(DTW_SYSTEM) sys1() {  
    %EXEC { /QSYS.LIB/ADDLIB.LIB LIB(MYLIBRARY) %}  
    %}
```

EXEC_PATH 構成変数を使用して、オブジェクト (コマンドおよびプログラムなど) を含むディレクトリへのパスを定義すると、実行可能なオブジェクトへのパスを短縮することができます。EXEC_PATH 構成変数の定義方法を理解するには、16ページの『EXEC_PATH』を参照してください。

例 1: コマンドを発行する

```
%FUNCTION(DTW_SYSTEM) sys2() {  
    %EXEC { /QSYS.LIB/CALL.COM MYLIB/MYPRG %}  
    %}
```

例 2: プログラムの呼び出し

```
%FUNCTION(DTW_SYSTEM) sys3() {  
    %EXEC { /QSYS.LIB/MYLIB.LIB/MYPRG.PGM %}  
    %}
```

ヒント: プログラムを呼び出すときには、直接呼び出し言語環境を使用します。この方が、より効果的で使用しやすいからです。

プログラムに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時にパラメーターを直接渡す。たとえば、以下のような場合です。

```
%DEFINE INPARAM1 = "SWITCH1"  
  
%FUNCTION(DTW_SYSTEM) sys1() {  
    %EXEC {  
        /QSYS.LIB/NETDATA.LIB/RPGCALL1.PGM ('$(INPARAM1)' 'literalstring')  
    %}  
    %}
```

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。この方法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

環境変数 を使用してパラメーターを間接的に渡します。環境変数は "name=value" 形式の文字ストリングで、プログラム外部の環境スペースに保管されます。ストリングは、プロセスに関連した一時スペースに保管されます。

Net.Data が DTW_SYSTEM 言語環境の関数を呼び出すと、言語環境は、入力 (IN) または入出力 (INOUT) のすべての関数パラメーターを環境スペースに保管してから、 %EXEC ブロック内のステートメントを実行します。ステートメントが正常に終了すると、DTW_SYSTEM 言語環境は、出力 (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環境スペースから検索し、関数仮引き数の値を新規値に更新します。 Net.Data は、制御を得ると、DTW_SYSTEM 言語環境から取得した新しい値で、すべての OUT または INOUT パラメーターを更新します。

環境変数の設定および取得には、表7に記載する API を使用します。

表 7. 環境変数 API

ILE プログラム言語	検索に使用するもの	設定に使用するもの
C, C++	getenv()	putenv()
CL(1), RPG, COBOL	QtmhGetEnv()(2)	QtmhPutEnv()(3)

1. OS/400 V3R7 以降では、CHGENVVAR および ADDENVVAR CL コマンドを使用して環境変数を設定することもできます。
2. QtmhGetEnv() は、IBM TCP/IP 接続ユーティリティ AS/400 用の一部として出荷されています。
3. QtmhPutEnv() は、もともと IBM TCP/IP 接続ユーティリティ AS/400 用の V3R2 および V3R7 の一部として出荷されていたものではありません。これは、後から追加され、V3R2 PTF 5763TC1-SF40953 または V3R7 PTF 5716TC1-SF40954 を介して入手することができます。

Net.Data 表を、システム言語環境によって呼び出されるプログラムに渡すことができます。プログラムでは、Net.Dataマクロの表パラメーターの値に、Net.Data 名でアクセスします。列見出しおよびフィールド値は、表名および列番号で識別される変数に含まれます。たとえば、表 myTable では、列見出しは myTable_N_j であり、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。表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、行が多い表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、システム言語環境を使用し、すべてのワークステーション・メッセージ待ち行列に Send Break Message (SNDBRKMSG) コマンドを発行するマクロを示しています。送信するメッセージのテキストは、形式データ (msgToSend) で構成されます。

```

%FUNCTION(DTW_SYSTEM) sndbrkmsg () {
  %EXEC { /QSYS.LIB/SNDBRKMSG.CMD MSG('$(msgToSend)') TOMSGQ(*ALLWS) %}
%}
%HTML(sndbrkmsg) {
  @sndbrkmsg()
%}

```

第7章 永続的マクロによるトランザクション管理

Net.Data は、永続的マクロによるトランザクション処理のサポートを提供します。永続的マクロは、マクロを Web サーバーで永続的 CGI プロセスの一部として実行できるようにする、組み込み関数を含むマクロです。つまり、1 つのマクロの複数ブロック、または複数マクロの複数ブロックを、単一の論理トランザクションとして実行できるようになるということです。

非永続的マクロの場合、Net.Data は、各マクロの起動を 1 つの完全なトランザクションとして扱います。つまり、各応答がブラウザーに送信されると、データベースがコミットされ、資源が開放されて、すべてが初期状態に設定されるということです。次に同じマクロを起動すると、フォーム・データからマクロに渡された情報、またはマクロ自体にある情報に基づいて、アプリケーションの状態が再確立されます。複数の起動にわたってマクロ変数を保管したり、変更されたデータベースの内容を明示的に元に戻さずにその変更をロールバックしたり、複数のブラウザー・セッションにわたるデータベースの変更を、1 つの完全なトランザクションとして取り扱ったりする機能はありません。

永続的マクロの場合、ユーザーはアプリケーション開発者として、トランザクション・レベルでアプリケーションを作成して、永続的な接続を保守しながら、1 つまたは複数のマクロを起動することができます。つまり、変数データは複数の起動にわたって永続的であるため、ユーザー・ログイン ID などの情報を隠し変数として、マクロの起動ごとに渡す必要はなくなります。この情報には Net.Data 表変数も含まれます。この変数は、非永続的マクロでは複数の起動にわたって渡すことができません。最も重要なことは、トランザクションの途中でユーザーが取り消しを行っても、アプリケーションは行われた作業をすべてロールバックできるということです。

永続的マクロの起動については、38ページの『永続的マクロの起動』を参照してください。

本章では、以下のトピックについて説明します。

- 『永続的マクロについて』
- 110ページの『トランザクションの定義』
- 117ページの『永続的マクロの例』

永続的マクロについて

永続的マクロを使用する場合、Net.Data は、Web サーバーの特別な永続的 CGI プロセスで実行され、標準入力および環境変数を介して入力を受け取り、標準出力を介してデータを提供します。ただし、出力が Web サーバーに戻った後でも、Web サーバーは、Net.Data のプロセスを終了する必要はありません。代わりに、プロセスは活動状態のままで、Web ブラウザーを介したユーザーからの応答を待ちます。プロセスが終了しないため、Net.Data はマクロの状態情報を保持して、トランザクションをオープンしたままにしておくことが可能です。

Net.Data は、サーバーに新規の HTTP ヘッダーを送信することによって、永続的 CGI プロセスで実行させたい Web サーバーと通信します。AS/400 HTTP Server バージョン 4 リリース 3 (V4R3) で、新規ヘッダー『Accept-HTSession』がサポートされるようになりました。HTTP ヘッダーは出力より先に送信する必要があるため、Net.Data は、最初の出力を送信する際に、サーバーに送信する HTTP ヘッダーを決定します。したがって、永続的マクロの開発では、以下の点に注意する必要があります。

- 最初の出力がマクロから生成されるときに、Net.Data は、これが永続的マクロになるかどうかを認識していなければならない。
- 新規の永続的マクロの組み込み関数を使用して、出力が生成される前に、そのマクロが永続的であると指定しなければならない。

これらの制約事項については、これ以降で説明します。

永続的 Net.Data プロセスの特性は、標準的な Net.Data プロセスの特性に非常によく似ていますが、以下の点が異なります。

- 疑似接続指向環境で実行される。Net.Data と Web サーバー間の接続は永続的ですが、ブラウザと Web サーバー間には接続がありません。
- 長時間トランザクションを実行できる。単一の Net.Data プロセスは、複数のブラウザ要求に対応することができるので、トランザクションをオープンしてコミットしたままにしたり、後続のブラウザの要求またはエラー条件に基づいて、適切にロールバックしたりすることができます。
- 永続的 Net.Data プロセスは、非常に長時間活動状態を維持することができるため、より多くのシステム資源を消費する可能性がある。これらの資源は注意して管理する必要があります。
- Web サーバーは永続性をサポートする必要があるため、移行性が低下する。

トランザクションの定義

1 つのトランザクションは、1 つの HTML ブロック、複数の HTML ブロック、または複数のマクロにわたって実行することができます。トランザクション内でマクロが永続的になるように指定する場合は、トランザクションの開始と終了を定義するだけでなく、そのトランザクションに組み込む HTML ブロックも定義する必要があります。Net.Data は、以下の永続的マクロのタスクの完了を支援する組み込み関数を提供します。

- 111ページの『トランザクションの開始』
- 112ページの『トランザクションでのマクロ HTML ブロックの指定』
- 115ページの『トランザクションの終了』
- 115ページの『トランザクションでの変数の効力範囲の定義』
- 116ページの『トランザクションでの COMMIT および ROLLBACK の指定』

トランザクションの開始

トランザクションを開始するには、ブラウザに出力が送信される前に、Net.Data に対して、マクロ内でマクロが永続的であることを示します。Net.Data は、その後、特別な HTTP ヘッダーを Web サーバーに送信して、マクロに永続的な CGI サポートが必要であることを通知します。

トランザクションを開始する

何らかの出力が Web ブラウザーに送信される前に、マクロで以下の方法のいずれかを使用します。

- DTW_STATIC() 組み込み関数を呼び出す。

DTW_STATIC() 関数は、現在のマクロが永続的であることを Net.Data に通知します。

構文 : @DTW_STATIC (["*timeout*"])

ここで、*timeout* は任意指定パラメーターであり、トランザクションの終了前に、Web サーバーがブラウザからの応答を待機する秒数を指定します。

例 :

```
@DTW_STATIC("60")
%DEFINE {
    var1 = "val1"
    var2 = "val2"
}%
...

%HTML(input){
    ...
}%

%HTML(report){
    ...
}%
```

このトランザクションでは、タイムアウト値が 60 秒に指定されています。60 秒以内にブラウザから応答を受信しない場合、Web サーバーはトランザクションを終了します。これによって、ブラウザの現行のページが影響を受けることはありません。ただし、次のページはこのトランザクションの一部になっているはずですが、新しいトランザクションの一部になります。

- STATIC 属性で変数を定義する。

構文 : %DEFINE(STATIC) var1 = "val1"

例 :

```
%DEFINE(STATIC) var1 = "val1"
%DEFINE var2 = "val2"
...
%HTML(input){
    ...
}%
%HTML(report){
    ...
}%
```

静的に定義された変数の値は、トランザクション全体にわたって値を保持するため、複数の Net.Data の呼び出しにわたって使用することができます。

トランザクションでのマクロ HTML ブロックの指定

HTML ブロックを起動する URL 要求において、トランザクション・ハンドル という識別子を使用して、トランザクションの一部にする HTML ブロックを定義します。トランザクション・ハンドルを定義して使用するには、以下の 3 つのステップを踏みます。

1. トランザクション・ハンドルをマクロで定義する。
2. DTW_ACCEPT 組み込み関数を呼び出して、Net.Data と Web サーバーにハンドル名を渡す。
3. URL 要求でハンドルを指定して、次の HTML ブロックを呼び出す。

トランザクション・ハンドルを定義する

1. DEFINE セクションでトランザクション・ハンドルの変数を定義する。たとえば、以下のようにします。

```
%DEFINE handle=""
```

2. DTW_RTVHANDLE() 組み込み関数をDEFINE セクションで指定して、固有のトランザクション・ハンドルをオプションで生成する。

構文 : @DTW_RTVHANDLE(*handle_name*)

例 :

```
@DTW_STATIC()
```

```
%DEFINE handle = ""  
@DTW_RTVHANDLE(handle)
```

トランザクション・ハンドルは、任意の有効な文字ストリングにすることができます。ただし、DTW_RTVHANDLE() 関数は、固有のトランザクション・ハンドルを生成することによって機密保護の手段を提供し、他のユーザーがこのトランザクションで実行されるマクロを起動できないようにします。

Net.Data に対するトランザクション・ハンドルを指定する

DTW_ACCEPT() 組み込み関数で、Net.Data に対するトランザクション・ハンドルの値を指定します。このハンドルは、サーバーに送信される HTTP ヘッダーに含まれる情報の一部であるため、マクロが出力を生成する前に、DTW_ACCEPT() 関数を呼び出さなければなりません。通常は、HTML ブロックの最初の要素になります。

構文 : @DTW_ACCEPT(*handle_name*, ["*timeout*"])

ここで、*timeout* は任意指定パラメーターであり、トランザクションの終了前に、Web サーバーがブラウザからの応答を待機する秒数を指定します。

DTW_ACCEPT() は、HTML ブロックの内部でも、任意の HTML ブロックの外部でも、呼び出すことができます。この関数が任意の HTML ブロックの外部で呼び出されると、トランザクション・ハンドルおよびオプションのタイムアウト値は、マクロ内のすべての HTML ブロックに適用されます。

例 1: このトランザクションで実行される後続の URL 要求に、トランザクション・ハンドルを指定する

```
@DTW_STATIC()

%DEFINE handle = ""
@DTW_RTVHANDLE(handle)

%HTML(Block1){
@DTW_ACCEPT(handle)
...
%}
```

重要： HTML ブロックの最初の要素として、DTW_ACCEPT() を呼び出す場合は、%HTML ステートメントが指定されている行と、DTW_ACCEPT() 呼び出し自体の間に空白文字がないことを確認してください。Net.Data は、空白文字をテキストとみなしてブラウザに送信しますが、データがブラウザに送信される前に DTW_ACCEPT() 呼び出しが検出されないため、エラーを発行します。

例 2: すべての HTML ブロックに適用されるトランザクション・ハンドルを、マクロで指定する

```
@DTW_STATIC()

%DEFINE handle = ""
@DTW_RTVHANDLE(handle)

@DTW_ACCEPT(handle)

%HTML(Block1){
...
%}

%HTML(Block2){
...
%}
```

トランザクションで実行させる HTML ブロックの呼び出し時に、ハンドルを指定する

トランザクション・ハンドルを生成し、DTW_ACCEPT() 関数を呼び出した後は、そのトランザクション・ハンドルを持つ URL だけが、ユーザーのトランザクションで実行することができます。トランザクション・ハンドルは、URL では CGI プログラムの直後になければなりません。

ご注意: コードにステートメントを入力する場合、URL はスペースを入れずに 1 行で入力しなければなりません、ここでは表示上、2 行に分割してあります。

- HTML リンク :

```
<A HREF="http://server/Net.Data_invocation_path/transaction_handle/
filename/block/[?name=val&...]">any text</A>
```

- HTML フォーム :

```
<FORM METHOD=method
ACTION="http://server/Net.Data_invocation_path/transaction_handle/
filename/block/[?name=val&...]">any text</FORM>
```

- URL:

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

パラメーター :

server Web サーバーの名前を指定します。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 実行可能ファイルのパスおよびファイル名。たとえば、
/cgi-bin/db2www/ のようになります。

transaction_handle

Net.Data マクロが開始したトランザクションの一部である URL を指定します。この識別子は、DTW_RTVHANDLE 組み込み関数を呼び出して取得します。これは、*Net.Data_invocation_path* の後になければなりません。

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・ステートメントとの突き合わせを試行します。詳しくは、
15ページの『MACRO_PATH』を参照してください。

block 参照される Net.Data マクロの中の 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。METHOD=POST を推奨します。

?name=val&...

Net.Data に渡される 1 つまたは複数の任意指定パラメーターを指定します。

通常は、これらの URL への HTML リンクを提供するか、マクロでフォーム・アクション・タグに URL を指定します。

例 1: 同じトランザクションで実行される、他のマクロ呼び出しにリンクする一般的な HTML ブロック

```
@DTW_STATIC()
...
%define handle = ""
@DTW_RTVHANDLE(handle)

%html (report) {
@DTW_ACCEPT(handle)
...
<a href="/cgi-bin/db2www/${handle}/qsys.lib/mylib.lib/
macros.file/pcgil.mbr/report2">continue</a><br>
<a href="/cgi-bin/db2www/${handle}/qsys.lib/mylib.lib/
macros.file/pcgil.mbr/quit">quit</a><br>
%}
```

例 2: 他のマクロへの FORM ACTION リンクを持つ、一般的な HTML ブロック

```
@DTW_STATIC()
...
%define handle = ""
@DTW_RTVHANDLE(handle)

%html(input) {
@DTW_ACCEPT(handle)
...
<form method=post action="/cgi-bin/db2www/${handle}/qsys.lib/
mylib.lib/macros.file/pcgil.mbr/report2">
<p>What type of hardware do you want to see?
```

```

<menu>
<li><input type="radio" name="hardware" value="MON" checked>Monitors
<li><input type="radio" name="hardware" value="PNT">Pointing devices
<li><input type="radio" name="hardware" value="PRT">Printers
<li><input type="radio" name="hardware" value="SCN">Scanners
</menu>
</form>
%}

```

トランザクションの終了

トランザクションを終了するには、マクロをこれ以降永続にしないよう、Net.Data に指示します。

トランザクションを終了する

トランザクションの終了を指定するには、DTW_TERMINATE() 組み込み関数を使用します。DTW_ACCEPT() 関数と同様に、この関数は、マクロが出力を生成する前に呼び出さなければならず、通常は HTML ブロックの最初の要素として指定します。DTW_TERMINATE は、この呼び出しが現行のトランザクションの最後の呼び出しであることを、Net.Data に通知します。

構文： @DTW_TERMINATE()

この関数は、パラメーターを一切受け入れません。

例：

```

%html(quit) {
@DTW_TERMINATE()
...
%}

```

トランザクションでの変数の効力範囲の定義

%DEFINE ステートメントの属性として効力範囲を指定することによって、トランザクションでの変数の有効範囲を決定することができます。指定できるのは、以下の通りです。

トランザクション有効範囲

トランザクション全体に対する変数の効力範囲。

単一呼び出し有効範囲

単一の Net.Data 呼び出しに対する変数の効力範囲。

変数のトランザクション有効範囲を指定する

STATIC 属性を指定して、変数がトランザクション有効範囲を持つように、つまり、変数の値がトランザクション内のすべての呼び出しにわたって保管されるように、指定します。STATIC は、永続的マクロの場合のデフォルトです。たとえば、以下のようになります。

```

@dtw_static()
%define(static) var1 = "val1"

```

変数に対して単一呼び出し有効範囲を指定する

TRANSIENT 属性を指定して、変数が単一呼び出し有効範囲を持つように、つまり、呼び出されるたびに変数の値が再初期設定されるように、指定します。 TRANSIENT は、非永続的マクロの場合のデフォルトです。たとえば、以下のようにします。

```
@dtw_static()
%define(transient) var1 = "val1"
```

永続的マクロでは以下ようになります。

- DTW_STATIC() 呼び出し以降の変数は、 TRANSIENT として明示的に定義されていない限りすべて STATIC です。
- DTW_STATIC() 呼び出し以前の変数は、 STATIC として明示的に定義されていない限りすべて TRANSIENT です。

トランザクションでの COMMIT および ROLLBACK の指定

非永続的マクロでは、コミットおよびロールバックは、呼び出しが成功したか失敗したかに基づいて、マクロ呼び出しの終了時に Net.Data によって暗黙的に実行されます。永続的マクロでは、コミットまたはロールバックは、トランザクションの終了時に実行されるようになっていきます。しかし、トランザクションは多数のマクロを呼び出すことができるため、トランザクション内の変更を段階的にコミットまたはロールバックしたい場合があります。

トランザクション中に保留中の変更をコミットする

DTW_COMMIT() 組み込み関数を指定します。

この関数はパラメーターを一切受け入れず、トランザクション内で保留中になっている変更をすべて実行します。

たとえば、以下のようにします。

```
%html (report) {
@dtw_accept(handle)
...
%IF (action="Enter")
  @dtw_commit()
  %ENDIF
%}
```

トランザクションで保留中の変更をロールバックする

DTW_ROLLBACK() 組み込み関数を指定します。

この関数はパラメーターを一切受け入れず、トランザクション内で保留中になっている変更をすべてバックアウトします。

たとえば、以下のようにします。

```
%html (report) {
@dtw_accept(handle)
...
%IF (action="Cancel")
  @dtw_rollback()
  %ENDIF
%}
```

永続的マクロの例

以下の単純なマクロには、単一のトランザクションで実行される複数の HTML ブロックが含まれています。

```
@dtw_static()
%define a = "0"
%define(transient) b = "0"
%define handle = ""
@dtw_rtvhandle(handle)

%html (report) {
@dtw_accept(handle)
a = $(a)<br>
b = $(b)<br>
@dtw_add(a, "2", a)
@dtw_add(b, "2", b)
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/report2">
click here to continue</a><br>
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/quit">
click here to quit</a><br>
%}

%html(report2) {
@dtw_accept(handle)
a = $(a)<br>
b = $(b)<br>
@dtw_add(a, "2", a)
@dtw_add(b, "2", b)
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/report3">
Click here to continue</a><br>
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/quit">
Click here to quit</a><br>
%}

%html(report3) {
@dtw_accept(handle)
a = $(a)<br>
b = $(b)<br>
@dtw_add(a, "2", a)
@dtw_add(b, "2", b)
<a href="/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/quit">
Click here to quit</a><br>
%}

%html(quit) {
@dtw_terminate()
a = $(a)<br>
b = $(b)<br>
done
%}
```

最初の呼び出しが HTML ブロックの `report` に対する呼び出しであるとする、`Net.Data` は以下を行います。

1. `DTW_STATIC()` 関数を呼び出して、このマクロが永続的であることを示す。
2. 永続的マクロのデフォルトが `STATIC` であるため、変数 `a` を `STATIC` 変数として作成する。
3. `TRANSIENT` 属性で明示的に定義されているため、変数 `b` を `TRANSIENT` 変数として作成する。
4. `DTW_RTVHANDLE()` を呼び出し、トランザクション・ハンドルを生成して変数 `handle` に入れる。
5. HTML ブロック `report` の処理を開始し、`DTW_ACCEPT()` を呼び出して、このトランザクション用のトランザクション・ハンドルを `Net.Data` に通知する。
6. 出力を検索してブラウザに送信する。これによって、`Net.Data` は HTTP ヘッダーを Web サーバーに送信し、トランザクションの開始を指示します。

7. HTML ページを表示する。変数 a および b の値は両方とも 0 です。

先頭ページの出力がブラウザに送信されると、ユーザーは、トランザクションを継続するか終了するかを選択することができます。継続を選択した場合は、Web サーバーによって以下の URL が呼び出されます。

```
/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/report2
```

Web サーバーは、トランザクション・ハンドルを、Net.Data によって HTTP ヘッダーで指定されたものとして認識します。そして、Net.Data を永続的な CGI プログラムとして呼び出します。これは、マクロの起動が現行トランザクションの一部であることを意味します。

HTML ブロック report2 が呼び出されると、Net.Data は以下を行います。

1. DTW_STATIC() 関数を呼び出して、このマクロが永続であることを示す。
2. 変数 a が STATIC 変数であることを認識し、0 に再初期設定するのではなく、現行の値を保持する。
3. 変数 b が TRANSIENT 変数であることを認識し、変数の新しいインスタンスを作成して 0 に初期設定する。
4. DTW_RTVHANDLE() を呼び出し、トランザクション・ハンドルを生成して変数 handle に入れる。
5. HTML ブロック report2 の処理を開始し、DTW_ACCEPT() を呼び出して、このトランザクション用のトランザクション・ハンドルを Net.Data に通知する。
6. 出力を検索してブラウザに送信する。これによって、Net.Data は HTTP ヘッダーをサーバーに送信し、トランザクションを続行するよう指示します。
7. HTML ページを表示する。変数 a の値は 2 に、変数 b の値は 0 になります。変数 a は静的変数であるため、直前の呼び出しからの値が保管されています。変数 b の値は 0 にリセットされます。

次のページがブラウザに送信されると、ユーザーは、トランザクションを継続するか終了するかを選択することができます。終了を選択した場合は、Web サーバーによって以下の URL が呼び出されます。

```
/cgi-bin/db2www/$(handle)/qsys.lib/mylib.lib/macros.file/pcgil.mbr/quit
```

Web サーバーは、トランザクション・ハンドルを、Net.Data によって HTTP ヘッダーで指定されたものとして認識し、Net.Data を永続的な CGI プログラムとして呼び出します。これは、マクロの起動が現行トランザクションの一部であることを意味します。

HTML ブロック quit が呼び出されると、Net.Data は以下を行います。

1. DTW_STATIC() 関数を呼び出して、このマクロが永続であることを示す。
2. 変数 a が STATIC 変数であることを認識し、0 に再初期設定するのではなく、現行の値を保持する。
3. 変数 b が TRANSIENT 変数であることを認識し、変数の新しいインスタンスを作成して 0 に初期設定する。
4. DTW_RTVHANDLE() を呼び出し、トランザクション・ハンドルを生成して変数 handle に入れる。

5. HTML ブロック quit の処理を開始し、DTW_TERMINATE() を呼び出して、このトランザクションでの最後の呼び出しであることを Net.Data に通知する。
6. 出力を検索してブラウザーに送信する。これによって、Net.Data は HTTP ヘッダーをサーバーに送信し、トランザクションの終了を指示します。
7. HTML ページを表示する。変数 a の値は 4 で、変数 b の値は 0 です。
8. DTW_TERMINATE() 呼び出しが実行されたため、トランザクション・レベルの効力範囲を持つ変数およびその他の資源すべてについて、終結処理を行う。

第8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。

- 『Net.Data マクロ・キャッシング』
- 『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。

Net.Data マクロ・キャッシング

Net.Data (OS/400 版) では、マクロ・キャッシングはデフォルトで使用可能となり、スループットを向上させ、CPU の使用率を下げるために使用されます。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。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、キャッシング構成変数を使用してコントロールできます。この変数の使用方法の詳細については、9ページの『DTW_MACRO_CACHE_SIZE: マクロ・キャッシュ・サイズ変数』を参照してください。

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』
- 122ページの『SQL 言語環境』
- 123ページの『システム言語環境』

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 言語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。
- REXX プログラムは、Net.Data のマクロにインラインに組み込まないで、外部ファイルに保管する。
- 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

DB2 のパフォーマンスの考慮について理解するには、*DB2 AS/400 用 SQL プログラミング*を参照してください。同書には、効率的な SQL 索引の使い方、結合照会のパフォーマンスの向上、3 つ以上の表からデータを選択する場合のパフォーマンスの向上、など豊富な情報が含まれています。

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。

- データベースに再接続しないようにするために、データベースに接続するユーザー ID の数を減らす。SQL 言語環境はユーザー・プロファイルとパスワードを、SQL 言語環境が確立するデータベースとのリモート接続に関連付けます。LOGIN と PASSWORD 変数が、開かれた接続に関連付けられているユーザー・プロファイルとパスワードに一致しない場合、その接続は閉じられ、再確立後、LOGIN および PASSWORD の値が再度開かれた接続と関連付けられます。
- 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。SELECT SQL ステートメントによる結果セットに数百のレコードが含まれている場合は、その結果セットのサブセットをブラウザーに戻します。それには、START_ROW_NUM をスクロール可能なカーソルのよう使用し、RPT_MAX_ROWS を使って、戻されるレコード数を制限します。Net.Data は、状態が不明になるとそのたびに照会を発行することを頭に入れておいてください。ただし、Net.Data の永続マクロのサポートを使用して、トランザクションの寿命期間の間永続する結果セットを Net.Data の表に保管することはできます。永続的な Net.Data マクロを理解するには、109ページの『第7章 永続的マクロによるトランザクション管理』を参照してください。
- 静的な SQL を使用するストアード・プロシーチャー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 の方が効率的である場合があります。

OS/400 V4R2 で開始した場合は、SQL エンジンが準備済みステートメント・キャッシュを持っています。キャッシュを使用すると、SQL エンジンが準備済みステートメントに関する情報を保管し、この情報をシステム全体の記憶域に保持します。したがって、同じステートメントが再度実行される場合には、そのステートメントが別のユーザーおよび別のジョブによるステートメントであっても、その

ステートメントはより高速に実行されます。システム全体の準備済みステートメント・キャッシュは、標準の SQL 処理の一部であり、それを構成したり、使用可能にするのにユーザーがアクションを取る必要はありません。このキャッシュは、静的 SQL が動的 SQL に対して持っているパフォーマンス上の利点を小さくしてしまうこともあります。

システム言語環境

グローバル Net.Data 変数を定義してその変数を参照することによって、システム言語環境が呼び出すプログラムに入力専用パラメーターを直接渡します。

Net.Data (OS/400 版) は、直接呼び出しと呼ばれる新しい言語環境を導入しました。これにより、プログラムを呼び出すのに、より容易でより効率的なインターフェースを提供しています。コマンドを発行するには、システム言語環境を使用し、プログラムを呼び出すには、直接呼び出し言語を使用してください。

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>

資料	説明
• <i>Net.Data</i> 管理およびプログラミングの手引き OS/390 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> 管理およびプログラミングの手引き OS/2、Windows NT、UNIX 版	
• <i>Net.Data</i> 管理およびプログラミングの手引き OS/400 版	
<i>Net.Data</i> 解説書	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・メッセージおよび戻りコードのリストです。

関連資料

Net.Data および関連製品を使用するにあたり、以下の資料が利用できます。

- *DB2 AS/400 用 SQL プログラミング*, SD88-5034
- *OS/400 Distributed Database Programming*, SC41-5702

さらに、OS/400 の資料とレッドブックが、DB2 に関する資料も含めて以下の URL で入手できます。

<http://publib.boulder.ibm.com/html/as400/infocenter.html>

付録B. 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、社員の名前リストから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。

マクロ・ファイルは、マクロ用の DEFINE ブロックを含んでいる組み込みファイルを使用します。

128ページの図9 は、サンプル・マクロを示しています。 130ページの図10 は、組み込みファイルを示しています。

```

%{***** 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 FIRSTNME 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>$(V1)
%}
</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 FIRSTNME, PHONENO, JOB FROM $(myTable) WHERE FIRSTNME='$(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
%}
%}

```

図9. サンプル・マクロ (1/3)

```
%{ *****
* 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>
%}
```

図9. サンプル・マクロ (2/3)

```
%{ *****
* 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 %}
```

図9. サンプル・マクロ (3/3)

```

=====
%{***** Include File *****)
*   FileName = sqlsamp1.hti                               *
*   Description:                                           *
*       This include file provides global DEFINES for the sqlsamp1.d2w *
*       Net.Data macro.                                   *
*****%}
%define {
    emp_name    = ""
    reposition  = sign
    exampleTitle = "Sample Macro"
    myTable     = "EMPLOYEE"
    DATABASE    = "sample"
%}

%{      End of include file  %}

```

図 10. 組み込みファイル

付録C. 特記事項

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミング、またはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミングまたはサービスを、日本で発表する意図があることを必ずしも示すものではありません。本書で、IBM ライセンス・プログラムまたは他の IBM 製品に言及している部分があっても、このことは当該プログラムまたは製品のみが使用可能であることを意味するものではありません。これらのプログラムまたは製品に代えて、IBM の知的所有権を侵害することのない機能的に同等のプログラムまたは製品を使用することができます。ただし、IBM によって明示的に指示されたものを除き、これらのプログラムまたは製品に関する稼働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 および他社は、本書で説明する主題に関する特許権 (特許出願を含む) 商標権、または著作権を所有している場合があります。本書は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用权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、使用权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

IBM は法律上の瑕疵担保責任を含むいかなる明示または黙示の保証責任も負いません。本書中に含まれているすべてのプログラムは "現存するままの状態" で提供されます。IBM はプログラムの商業的な使用可能性および特定の目的に対する適合性については、いかなる保証も行いません。国によっては、特定の取り引きに関する明示または黙示の保証責任を拒否することができないため、この文章は適用されません。

この資料の情報に関する変更は随時行われています。それらの変更は、本書の改訂版で反映されます。IBM は、本書で説明されている製品やプログラムを、お客様に通知することなく改善または変更する場合があります。

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。

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

IBM Corporation
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 プログラムのご使用条件」の契約条件に基づいて弊社から提供されるものです。

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

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

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

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

Java および すべての Java ベースの商標とロゴは、米国ならびに他の国における Sun Microsystems, Inc. の商標です。

UNIX は、X/Open Company Limited がライセンスしている米国ならびに他の国における登録商標です。

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 (/) あるいは円記号 (¥) 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。

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

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

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

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 (Common Gateway Interface (CGI)). Web サーバーがアプリケーション・プログラムに制御権を渡し、反対にデータを受け取る、標準的な方法。

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。

DATALINK. DB2 の 1 つのデータ型。データベースから、データベースの外側に格納されたファイルへの論理参照を可能にする。

データ型 (data type). 列およびリテラルの属性。

DBCLOB. 2 バイト文字ラージ・オブジェクト。

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ー。

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

GWAPI. Go Web サーバー API。

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

HTTP. Hypertext transfer protocol (HTTP)。

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

hypertext transfer protocol (HTTP). 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中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 (path name). オブジェクトの検索方法をシステムに通知する。パス名は、一連のディレクトリー名にオブジェクトの名前を続けたものとして表される。個々のディレクトリーおよびオブジェクト名は、通常のスラッシュ (/) または円記号 (¥) 文字によって区切られる。

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ついて割り当てられた値が保持されている状態。1 つのトランザクションは、複数の Net.Data 呼び出しにわたることができる。永続的にすることができるのは変数のみである。さらに、コミットメント制御によって影響を受けるリソースの操作は、明示的なコミットまたはロールバックが行われるかトランザクションが完了するまでアクティブのまま保持される。

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

レジストリー (registry). スtringを保管および取得することができるリポジトリ。

相対パス名 (relative path name). 最高レベル (つまり「ルート」ディレクトリー) で始まらないパス名。システムは、このパス名をプロセスの現行作業ディレクトリーで始めると想定する。

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). 1 回の Net.Data の呼び出し。永続的な Net.Data を使用する場合は、1 つのトランザクションが複数の Net.Data 呼び出しにわたることができる。

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

URL. Uniform resource locator (URL)。

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

作業単位 (unit of work). アトミック操作として処理される操作の回復可能なシーケンス。あらゆる作業単位内の操作は、複数の操作が単一の操作であるかのように完了 (コミット) や取り消し (ロールバック) を行うことができる。コミットメント制御によって影響を受けるリソースの操作のみをコミットまたはロールバックすることができる。

Web サーバー (Web server). Internet Connection などの HTTP サーバー・ソフトウェアを実行するコンピューター。

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、DB2 の 91

アクセス権

言語環境の 83

アクセス権限

Net.Data のファイルのための 23

暗号化、ネットワークの 27

一般的エラー、パラメーターを渡すときの 86, 99

印刷、デフォルトのレポートでは使用禁止の 72

永続関数 69

永続的マクロ 109

エラー条件、言語環境の 82

[カ行]

開始、Net.Data の 35

隠し変数

資産の保護 29

変数名を隠す 54

各種変数 57

型、変数の 52

環境変数 52

関数 99

永続的 69

数学 67

ストリング 68

説明 59

定義 60

汎用 67

表 68

フラット・ファイル 69

ユーザー定義 60

呼び出し 64

呼び出し、ストアード・プロシーチャーの 99

ワード 68

FUNCTION ブロックの構文 60

MACRO_FUNCTION ブロックの構文 60

Web レジストリー 69

関数呼び出し

組み込み 65

構文 64

起動、Net.Data の 35, 36

概説 35

フォーム 35, 38, 113

起動、Net.Data の 35, 36 (続き)

マクロによる 35

リンク 35, 38, 113

CGI の使用 35

HTML ブロック 70

URL 36, 39, 113

機密保護

アクセス権の指定 23, 83

概説 25

許可 29

言語環境 83

認証 27

ネットワーク暗号化 27

ファイアウォール 25

Net.Data のメカニズム 29

許可

機密保護 29

Net.Data のファイルへのアクセス権の指定 23

グローバルな識別子の効力範囲 47

結果セット 102, 103

処理、ストアード・プロシーチャー 102

単一 102

複数の 103

ガイドラインおよび制約事項 76

デフォルトのレポート 103

結果セットでの DataLink URL のエンコード 96

結果セットの処理、ストアード・プロシーチャー 102

言語環境 89, 106

エラー条件の処理 82

機密保護 83

構成、初期設定ファイル内の 19

サポートされた 82

システム 106

セットアップ 21

直接呼び出し 83

変数 59

呼び出し 82

例 19

ENVIRONMENT ステートメントの構成 19

Java アプリケーション 87

REXX 89

SQL 91

向上、パフォーマンスの 121

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

DTWR_CLOSE_REGISTRIES 14

DTW_MACRO_CACHE_SIZE 9

DTW_PAD_PGM_PARMS 9

構成変数ステートメント (続き)

DTW_SHOWSQL 8
DTW_SMTP_CCSSID 11
DTW_SMTP_CHARSET 11
DTW_SMTP_SERVER 12
DTW_SQL_ISOLATION 12
DTW_SQL_NAMING_MODE 13

効力範囲、識別子の

グローバルな 47
マクロ 48
FUNCTION ブロック 48
REPORT ブロック 48
ROW ブロック 48

コピー、Net.Data プログラム・オブジェクトの
複数のライブラリーへの 6

CGI-BIN ライブラリーへの 5

コマンドの実行 106

[サ行]

作成、初期設定ファイルの 8

参照、変数の 50

サンプル・マクロ 127

識別子の効力範囲 47

システム言語環境 106

概説 106
コマンドの発行 106
パラメーターを渡す 106
プログラムの呼び出し 106

実行、SQL ステートメントの 91

実行可能な変数 53

条件付き

変数 52
論理、IF ブロック 77

初期設定ファイル

更新 7
構成変数ステートメント 8
作成 7, 8
説明 6
パス・ステートメント 14
フォーマット 7
ENVIRONMENT ステートメント 19

数学関数 67

ストアド・プロシージャー 99, 100, 102, 103, 104

結果セットの処理 102
ステップ 100
単一の結果セット 102
デフォルトのレポート 102, 103
パラメーターを渡す 102
複数の結果セット 103
マクロからの呼び出し 99
有効なデータ型 100
REPORT ブロック 103, 104

ストリング関数 68

宣言部分、マクロ構造の 41

[タ行]

直接呼び出し言語環境

概説 83
サポートされているデータ型 84
パラメーターを渡す 84
パラメーターを渡すときの一般的エラー 86
プログラムからの戻り値 86
プログラムの呼び出し 83

データ型 94, 96, 100

ストアド・プロシージャーの 100

直接呼び出しの 84

DATALINK 96

LOB 94

定義、変数の

照会ストリング・データ 50
DEFINE ステートメントまたはブロック 48
HTML フォーム SELECT、INPUT、および
TEXTAREA タグ 49

デフォルトのレポート 102, 103

印刷 72
ストアド・プロシージャーに指定 102, 103

トークンのサイズ 47

動的な生成、変数名の 50

特記事項 131

トランザクション処理 109

[ナ行]

ナビゲーション、マクロ内およびマクロ間の 45

認証、機密保護の 27

[ハ行]

パス・ステートメント

更新のガイドライン 15
構成、初期設定ファイル内の 14
資産の保護 29

DTW_JAVA_CLASSPATH 19

EXEC_PATH 16

FFI_PATH 18

HTML_PATH 18

INCLUDE_PATH 17

MACRO_PATH 15

パフォーマンス 91

最適化、言語環境の 121

システム言語環境 123

REXX 環境 91

REXX 言語環境 121

パフォーマンス 121 (続き)
SQL 言語環境 121
パラメーターを渡す 90, 102, 106
システム言語環境 106
ストアード・プロシージャー 102
REXX プログラム 90
汎用関数 67
表関数 68
表処理変数 57
表変数 56
ファイアウォール 25
ファイル、Net.Dataへのアクセス権の指定 23
フォーマット、データ出力の 71
フォーム 35, 37
起動、Net.Data の 113
Net.Data の起動 35, 38
Net.Data を起動するための Web ページ内の 37
複数のレポート・ブロック 73
フッター情報、REPORT ブロック 72
部分、マクロの
宣言 41
表示 41
フラット・ファイル関数 69
ブロック、マクロの 43
ヘッダー情報、REPORT ブロック 72
変数
隠し 54
各種 57
型 47, 52
環境 52
言語環境 59
構成、ステートメントの
初期設定ファイル 8
説明 8
電子メール SMTP CCSID
(DTW_SMTP_CCSSID) 11
電子メール SMTP サーバー
(DTW_SMTP_SERVER) 12
電子メールの SMTP 文字セット
(DTW_SMTP_CHARSET) 11
ブランクがある埋め込みパラメーター
(DTW_PAD_PGM_PARMS) 9
マクロ・キャッシュ・サイズ
(DTW_MACRO_CACHE_SIZE) 9
SHOWSQL を使用可能にする
(DTW_SHOWSQL) 10
SHOWSQL を使用不可にする
(DTW_SHOWSQL) 10
SMTP サーバー (DTW_SMTP_SERVER) 12
SMTP 文字セット (DTW_SMTP_CHARSET) 11
SQL の名前モード
(DTW_SQL_NAMING_MODE) 13

変数 (続き)
構成、ステートメントの (続き)
SQL の分離 (DTW_SQL_ISOLATION) 8
Web レジストリーのクローズ
(DTWR_CLOSE_REGISTRIES) 14
効力範囲 47
参照 50
実行可能 53
条件付き 52
説明 47
定義 48
トークンのサイズ 47
動的に生成された参照 50
名前の動的な生成 50
表 56
表の処理 57
リスト 55
レポート 58
保護、資産の 25

[マ行]

マクロ
永続的 109
開発 41
関数 59
サンプル 42
識別子の効力範囲 48
条件付き論理 77
説明 1
宣言部分 41
内部およびその間のナビゲーション 45
表示部分 41
ブロック 43
分析 42
変数 47
ループ 79
DEFINE ブロック 43
FUNCTION ブロック 44
HTML の生成 70
HTML ブロック 45
IF ブロック 77
WHILE ブロック 79
マクロのキャッシング、キャッシュ・サイズ 9
マクロ要求 35
構文 35
例 35
戻り値、プログラムからの 86

[ヤ行]

ユーザー定義関数 60
ユーザー定義の言語環境、ENVIRONMENT ステートメント 7
用語集 133
呼び出し 89, 99, 100, 106
関数 64
言語環境 82
ストアード・プロシージャ 99, 100
プログラム、システム 106
プログラム、直接呼び出し 83
Java アプリケーション 88
REXX プログラム 89

[ラ行]

ラージ・オブジェクト (LOB) 94, 95
説明 94
有効な形式 95
リスト変数 55
リンク 35, 37
起動、Net.Data の 35, 113
Net.Data の起動 38
Net.Data を起動するための Web ページ内の 37
ループ、WHILE ブロックの 79
レポート
デフォルト 73
1 つの関数呼び出しによる複数生成 73
レポートのフォーマットのカスタマイズ 72
レポート変数 58

[ワ行]

ワード関数 68
渡す、パラメーターを
直接呼び出し言語環境 84
Java アプリケーション言語環境 88

B

BLOB 94

C

CGI-BIN ライブラリー、Net.Data プログラム・オブジェクトのコピー 5
CLOB 94

D

DATALINK データ型 96
DataLink ファイル・マネージャー 96

DATALINK データ型 96 (続き)
URL のエンコード 96
DBCLOB 94
DEFINE ブロック
説明 43
変数の定義 48
DTWR_CLOSE_REGISTRIES 14
DTW_DEFAULT_REPORT 73
DTW_DIRECTCALL 83
DTW_JAVAPPS 87
DTW_JAVA_CLASSPATH 19
DTW_MACRO_CACHE_SIZE 9
DTW_PAD_PGM_PARMS 9
DTW_REXX 89
DTW_SHOWSQL 10
DTW_SMTP_CCSID 11
DTW_SMTP_CHARSET 11
DTW_SMTP_SERVER 12
DTW_SQL 91
DTW_SQL_ISOLATION 12
DTW_SQL_NAMING_MODE 13
DTW_SYSTEM 106

E

ENVIRONMENT ステートメント
言語環境のタイプ 20
構成、初期設定ファイル内の 19, 20
構文 20
サービス・プログラム 20
説明 19
パラメーター・リスト 20
ユーザー定義の言語環境の 7
例 20

F

FFI_PATH 18
FUNCTION ブロック
関数の呼び出し 64
識別子の効力範囲 48
出力のフォーマット 71
説明 44

H

HTML 35, 37
認識されていないデータ 71
表のタグ 72
フォーム 35, 37
起動、Net.Data の 113
について 37

HTML 113, 37 (続き)
変数を定義する SELECT、INPUT、および
TEXTAREA タグ 71
Net.Data の起動 35, 38
ブロック
処理 71
説明 45
例 70
Net.Data の起動 70
マクロ内に生成 70
リンク 35, 37
起動、Net.Data の 113
について 37
Net.Data の起動 35, 38
FORM の処理依頼ボタン 71
HTML_PATH 18

I

IF ブロック 77
INCLUDE_PATH 17

J

Java アプリケーション言語環境
概説 87
セットアップ 21
パラメーターを渡す 88
プログラムの呼び出し 88

L

LOB (ラージ・オブジェクト) 94
サポートされているデータ型 94
SQL および ODBC 言語環境を使った 94

M

MACRO_FUNCTION ブロック
関数の呼び出し 64
構文 60
MACRO_PATH 15
MESSAGE ブロック
構文 63
効力範囲 63
処理 63
説明 63
例 64

N

Net.Data
概説 1

Net.Data (続き)
起動 1
機密保護のメカニズム 29
構成 5
ファイル、アクセス権 23
マクロ、開発 41
Net.Data の構成
概説 5
言語環境のセットアップ 21
初期設定ファイル
更新 7
構成変数ステートメント 8
作成 7
説明 6
パス・ステートメント 14
ENVIRONMENT ステートメント 19
Net.Dataのファイルへのアクセス権 23
Net.Data プログラム・オブジェクト
複数のライブラリーへのコピー 6
CGI-BIN ライブラリーへのコピー 5
Net.Data マクロ。マクロを参照。 1

R

REPORT ブロック 103, 104
効力範囲 48
ストアド・プロシージャ 103, 104
制約事項 76
説明 71
データ出力のフォーマット 71
デフォルトのレポート 73
複数の 73
複数のためのガイドライン 76
ヘッダーおよびフッター情報 72
例 74
RETURN_CODE 変数 63, 82
REXX 言語環境 89, 90, 91
概説 89
パラメーターを渡す 90
プログラムの呼び出し 89
AIX 上でのパフォーマンス 91
ROW ブロック、識別子の効力範囲の 48

S

SQL
名前付けモード構成変数 13
分離構成変数 12
SQL 言語環境
概説 91
セットアップ 21
パラメーターを渡すときの一般的エラー 99
SQL ステートメントの実行 91

SQL ステートメントの実行 91

SQLCODE 82, 83

U

URL 36

起動、Net.Data の 113

変数の定義 50

Net.Data の起動 36, 39

W

Web レジストリー、クローズ変数 14

Web レジストリー関数 69

WHILE ブロック 79



Printed in Japan

SB88-7422-00



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