



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、107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 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 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、以後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にも適用されます。

原 典： VNDT-24AD-00 (この資料番号でオーダーすることはできません。)
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 1998.11

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

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

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

Translation: © Copyright IBM Japan 1998

目次

まえがき	v
Net.Data について	v
本書について	vi
本書の対象読者	vi
本書の例について	vii
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
Net.Data プログラム・オブジェクトの CGI-BIN ライブラリーへのコピー	5
Web サーバーの構成	6
Net.Data の初期設定ファイルについて	7
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	9
パス構成ステートメント	11
環境構成ステートメント	15
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	17
第3章 ユーザー資産を保護する	19
ファイアウォールを使用する	19
ネットワーク上のユーザーのデータを暗号化する	21
認証を使用する	22
許可を使用する	23
Net.Data のメカニズムを使用する	23
第4章 Net.Data を起動する	25
マクロ・ファイルで Net.Data を呼び出す (マクロ要求)	25
HTML リンク	27
HTML フォーム	27
永続的マクロの起動	28
永続的マクロの構文	28
例	29
第5章 Net.Data のマクロ開発	31
Net.Data のマクロ・ファイルの分析	32
DEFINE ブロック	34
FUNCTION ブロック	34
HTML ブロック	35
Net.Data のマクロ変数	37
変数の効力範囲	37
変数の定義	39
変数の参照	40
変数の型	41
Net.Data の関数	48
ユーザー定義関数の定義	48
関数の呼び出し	53
Net.Data 組み込み関数の呼び出し	55

マクロでの Web ページの生成	59
HTML ブロック	59
レポート・ブロック	61
マクロ・ファイルにおける条件付き論理とループ	66
条件付き論理	66
ループ構成体	68
第6章 言語環境の使用	71
REXX 言語環境	72
REXX 言語環境を構成する	72
外部 REXX プログラムを呼び出す	72
パラメーターを渡す	72
REXX の SAY 命令を OS/400 V3R2 または V3R7 で使用する	73
機密保護	74
パフォーマンスを向上させる	74
REXX 言語環境の例	74
SQL 言語環境	75
SQL 言語環境を構成する	75
SQL 言語環境をコミットメント制御のもとで実行する	76
リモート・データベースのデータベース接続を管理する	76
ストアド・プロシージャ	77
SQL 言語環境の制約事項	82
機密保護	82
パフォーマンスを向上させる	83
SQL 言語環境の例	84
システム言語環境	86
システム言語環境を構成する	86
パラメーターを渡す	86
機密保護	87
パフォーマンスを向上させる	88
システム言語環境の例	88
第7章 永続的マクロによるトランザクション管理	89
永続的マクロの概要	89
トランザクションの定義	90
トランザクションの開始	91
トランザクションでのマクロ HTML ブロックの指定	92
トランザクションの終了	95
トランザクションでの変数の効力範囲の定義	95
トランザクションでの COMMIT および ROLLBACK の指定	96
永続的マクロの例	97
付録A. 問題分析	101
付録B. Net.Data サンプル・マクロ	103
付録C. 特記事項	107
商標	108
用語集	109
索引	111

まえがき

Net.Data バージョン 2 をお買上げいただきありがとうございます。本製品は、動的 Web ページを作成するための IBM 開発ツールです。Net.Data を使用すれば、さまざまなデータ・ソースからデータを取り込んだり、既知のプログラム言語が持つ長所を生かしたりして、動的コンテンツを含んだ Web ページを迅速に開発することができます。

Net.Data バージョン 2 は、ユーザーのインターネット・ビジネス・ソリューションの開発および活用に役立つ新規フィーチャーを提供するとともに、大幅なパフォーマンスの向上を実現しています。

Net.Data について

IBM の Net.Data 製品では、DRDA を介してアクセスできる DB2 データベースを含むリレーショナル・データベース管理システムおよび非リレーショナル・データベース管理システム (DBMS) の両方からのデータを使用するとともに、Java、JavaScript、Perl、C、C++、REXX などのプログラミング言語で作成したアプリケーションを使用して、動的 Web ページを作成することができます。

Net.Data はマクロ処理プログラムで、Web サーバー上のミドルウェアです。ユーザーは、Net.Data アプリケーション・プログラムである呼び出し先マクロを書き込むことができます。Net.Data はユーザーからの入力、ユーザーのデータベースの現在の状態、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ほかの要素に基づくカスタマイズされた内容を解釈しながら、動的 Web ページを作成します。

要求は URL (uniform resource locator) の形式で、Netscape Navigator または Internet Explorer などのブラウザから Web サーバーに流れ、Web サーバーはその要求を Net.Data に送って実行します。Net.Data はマクロを検索、実行し、ユーザーが作成した関数に基づいてカスタマイズ 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。

- C、C++、RPG、COBOL、JAVA、REXX プログラミング言語などの任意の言語で作成されたアプリケーションにビジネス論理をカプセル化する。
- DB2 などのデータベースにアクセスする。
- フラット・ファイルなどのその他のデータ・ソースにアクセスする。

Net.Data は、この Web ページを Web サーバーに渡します。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) などの業界標準のインターフェースをサポートしています。HTTP はブラウザと Web サーバー間で使用され、CGI は Web サーバーと Net.Data 間で使用されます。このサポートによって、ユーザーは使い慣れたブラウザもしくは Web サーバーを選んで Net.Data で使用することができます。Net.Data ファミリーの製品は、OS/400、OS/390、Windows NT、AIX、OS/2、HP-UX、Sun Solaris、および SCO オペレーティング・システムで同様の機能を提供します。

Net.Data バージョン 2 では、多くのパフォーマンスおよび機能も拡張されています。

- 以下の点でパフォーマンスが向上しています。
 - SQL 言語環境を介して SQL ストアード・プロシージャを呼び出す機能
 - SQL ストアード・プロシージャから、SQL 言語環境を介して 1 つまたは複数の結果セットを取り出す機能
 - 永続的な Net.Data を使用した、Net.Data の複数の呼び出しにわたるトランザクションのサポート
- マクロ言語環境は、次のものを含んでいます。
 - 注釈をどこでも追加する機能
 - ネストされた IF ブロック
 - WHILE ブロック
 - MACRO_FUNCTION ブロック
 - 整数の比較
 - 複数のレポート・ブロック

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。DB2 データベースにアクセスする Net.Data 提供の言語環境を使用する方法をはじめ、RPG や COBOLなどのプログラミング言語を使用してデータにアクセスする方法を学習します。

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。オペレーティング・システムの差、Net.Data メッセージ、およびその他の情報は、*Net.Data 解説書* で解説しています。

本書で説明する概念を理解するには、Web サーバーの仕組みに関する知識があり、単純な SQL ステートメント、HTML タグ、および HTML フォーム・タグを理解している必要があります。*Net.Data 解説書* に記載されている情報に精通しておく必要があります。

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

インターネット上の Web ページのほとんどは静的 Web ページであり、ユーザーが編集しない限り変化しません。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 について、および Net.Data を使用する理由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 マクロによって、論理、変数、関数呼び出し、およびレポート生成ツールを使用することができます。マクロとは Net.Data マクロ言語の構成要素、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・ファイルを処理して、Web ブラウザーが表示することができる出力を作成します。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 for OS/400 および Web サーバーの関係、サポートされるデータとプログラム言語環境への環境を示しています。

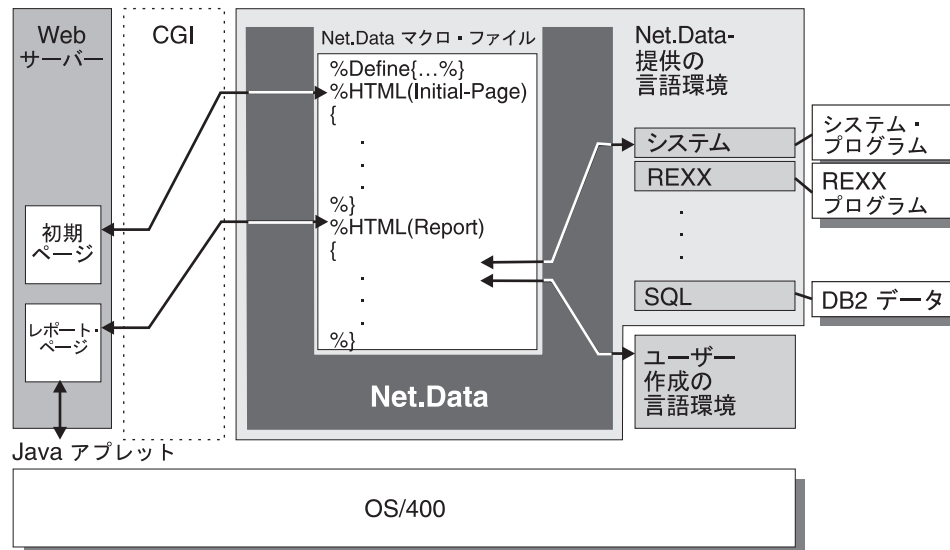


図 1. Net.Data for OS/400、Web サーバー、およびサポートされるデータとプログラム・ソース間の関係

Net.Data サービスを要求する URL を受信すると、Web サーバーは、Net.Data を CGI アプリケーションとして呼び出します。URL には、Net.Data 固有の情報が含まれます。これらの情報には、処理するマクロ・ファイルが含まれます。Net.Data は要求の処理が終了すると、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ー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。また、Net.Data は、DB2 データベースにアクセスしたり、アプリケーションに REXX や Perl などの言語を使用したりする言語環境を提供します。さらに、マクロ・ファイルに対する変更を即時にブラウザー上に表示させることができます。

Net.Data は、OS/400 の既存の広範なデータ管理機能を拡張し、インターネット情報の処理および表示の新しい形式および将来的な形式をサポートする機能を追加します。ユーザーのアプリケーションに対する利点として、Net.Data には以下の点があります。

- データの生成および表示を分離します。Net.Data では、データの表示方法 (HTML や JavaScript など) にまったく制約がありません。このため、ユーザーは、最新の表示手法を使用してデータの表示を簡単に変更することができます。
- C、C++、RPG、COBOL、REXX、Java などの言語で作成されたプログラムとのインターフェースをとる機能を提供するので、既存のスキルおよびビジネス論理を使用してデータを生成することができます。

- 特別なプログラミング・スキルが不要な、インターネット・アプリケーション開発用の機能を提供します。
- DB2 および任意のリモートの DRDA 対応データベースに保管されたハイパフォーマンスのデータへのアクセスを提供します。
- 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。

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

型を持たない変数

Net.Data はすべてのデータを文字ストリングとして取り扱います。 Net.Data は組み込み関数を使用して、有効な数字を表しているストリングを算術演算します。指数のフォーマットもサポートされます。マクロ言語の変数の詳細については、37ページの『Net.Data のマクロ変数』で解説されています。

組み込み関数

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

エラー処理

Net.Data がエラーを検出すると、説明の付いたメッセージがクライアントに戻されます。ブラウザ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

追加して購入する必要があるものではありません。ダウンロードおよびインストールに必要な Net.Data ソフトウェアはありません。

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。以下のバージョンの OS/400 オペレーティング・システムでは、以下の任意選択のソフトウェアをユーザーのシステムにインストールしなければなりません。

- 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 通信ユーティリティー/400 (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 をインストールしたら、以下の節で説明するステップを完了して Net.Data for OS/400 を構成してください。このステップを以下に示します。

- 『Net.Data プログラム・オブジェクトの CGI-BIN ライブラリーへのコピー』
- 6ページの『Web サーバーの構成』
- 7ページの『Net.Data の初期設定ファイルのカスタマイズ』
- 8ページの『初期設定ファイルの作成』
- 17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』

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

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て、オブジェクトへのアクセス権を提供しなければなりません。

Net.Data プログラム・オブジェクトをコピーする

1. オブジェクト複製 (CRTDUPOBJ) コマンドを使用して、Net.Data プログラム・オブジェクト DB2WWW を QTCP ライブラリーから CGI-BIN ライブラリーにコピーします。

OS/400 V4R3 ユーザー: ライブラリー QHTTPSVR にあるプログラム・オブジェクトを使用してください。QTCP ライブラリーにあるプログラム・オブジェクトによって、Net.Data 要求が QHTTPSVR ライブラリーに経路指定されます。

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関する詳細については、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。認証に関する詳細については、22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

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します。
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ます。
 - 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 プログラムを作成します。
 - b. CL プログラムを各ライブラリーにコピーします。

実際は、作成した CL プログラムは 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 CL 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 を異なるライブラリーにコピーする場合は、SQL 言語環境を使用すると -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 CL 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らないません。

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。

HTTP 構成ファイルをディレクティブに追加することによって Web サーバーが Net.Data を呼び出すように構成して、Net.Data が呼び出されるようにしてください。

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 に入っているとすると、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/*
```

Map ディレクティブは、形式 /cgi-bin/db2www/* を使用するエントリーを、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

の最後のアスタリスク (*) は、このストリングに続く任意のものを指します。) 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位置を指しています。

Exec ディレクティブによって、Web サーバーは、CGI ライブラリーにある任意の CGI プログラムを実行することができます。ディレクティブで、(プログラム自体ではなく) プログラムが入っているライブラリーを指定してください。

Pass ディレクティブは、Net.Data によって使用されません。URL を単純化する場合は、以下の節で説明する Net.Data 初期設定ファイルの MACRO_PATH ステートメントを使用してください。

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。構成変数の設定は、Net.Data 操作のさまざまな側面を制御します。

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、マクロやプログラムなどの 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ことによって、Net.Data マクロ・ファイル内の URL およびプログラムや組み込みファイルの参照を短縮することができます。ただし、ユーザー自身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は、サポートされている言語環境ステートメントのみで初期設定ファイルを構成したかのように Net.Data が実行されます (サポートされている言語環境については、71ページの『第6章 言語環境の使用』を参照してください)。この場合は、マクロ・ファイル内のマクロ、組み込み、および実行可能ファイルの参照は、すべて完全修飾し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。

- 9ページの『構成変数ステートメント』
- 11ページの『パス構成ステートメント』
- 15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関する詳細については、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 DTWR_CLOSE_REGISTRIES YES 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(DTW_REXX) /QSYS.LIB/QTCP.LIB/QTMHREXX.SRVPGM () 6 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) 7 ENVIRONMENT(DTW_SYSTEM) /QSYS.LIB/QTCP.LIB/QTMSYS.SRVPGM ()</pre>	<ul style="list-style-type: none">• 1 行は、構成変数の値を設定します。• 2 ～ 4 行は、Net.Data がアクセスする必要があるファイルへのパスを定義します。• 5 ～ 7 行は、使用可能な環境ステートメントを定義します。
--	---

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、見やすくするために何行かに分けて示しています。) Net.Data マクロで言語環境を呼び出さない場合は、ENVIRONMENT ステートメントは必要ありません。また、マクロ・ファイル内のファイルへの参照をすべて完全修飾する場合は、パス構成ステートメントを指定する必要はありません。

以下の節では、初期設定ファイルの作成方法およびINI ファイルで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 を使用する場合は、初期設定ファイルの作成は任意選択です。初期設定ファイルは、以下の場合に作成しなければなりません。

- 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、組み込み、実行可能プログラムのファイルに対するパス・ステートメントを定義して、これらのファイルへの参照を短縮する。
- 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

1. DB2WWW プログラム・オブジェクトが入っているライブラリーで、ソース物理ファイル作成 (CRTSRCPF) コマンドを使用して初期設定ファイルを作成します。

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 行に入っていなければならないので、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. 原始ステートメント入力ユーティリティ (SEU) またはワークステーション・エディターを使用して、サンプル・マクロおよび以下の節で説明するファイルに構成ステートメントを追加します。

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・ジョブによる最初の呼び出し中に、初期設定ファイルを一度読み取ります。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。詳細については、17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、言語環境によって異なります。また、そのほか、アプリケーションに固有な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

アプリケーションが要求する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。構成変数は、以下の構文を持ちます。

NAME[=]value-string

等号は、オプションです。大括弧で示されます。

以下のサブセクションでは、Net.Data 提供の言語環境によって使用され、初期設定ファイルで指定することができる構成変数ステートメントについて説明しています。

- 『DTW_SQL_ISOLATION: DB2 の分離変数』
- 10ページの『DTW_SQL_NAMING_MODE: SQL の表名変数』
- 11ページの『DTWR_CLOSE_REGISTRIES: Web レジストリー・オープン用変数』

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。選択されている行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。

SYS_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・オープン用変数

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

ここで、

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化することにより、(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。

パス構成ステートメント

Net.Data は、Net.Data のマクロ・ファイル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。

- 12ページの『MACRO_PATH』
- 13ページの『EXEC_PATH』
- 14ページの『INCLUDE_PATH』
- 15ページの『FFI_PATH』

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、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：

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

- 指定された各ディレクトリーは、セミコロン (;) で区切られる。
- スラッシュ (/) および円マーク (¥) 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。この複数パス能力により、ユーザーのファイルを複数のディレクトリーに編成することができます。たとえば、複数の Web アプリケーションをそれぞれ、それ自身のディレクトリーに配置することができます。
- 絶対パス・ステートメントの使用をお勧めします。

ヒント：Net.Data は、指定されたディレクトリーをすべて検索します。ただし、サブディレクトリーは検索しません。たとえば、Net.Data のマクロが以下のディレクトリーにあるとした場合、パス・ステートメントには、各サブディレクトリーを指定しなければなりません。

```
/usr/test/client  
/usr/test/assoc  
/usr/test/partner
```

MACRO_PATH ステートメントは、以下のようになります。

```
MACRO_PATH [=] /usr/test/client;usr/test/assoc;usr/test/partner
```

以下の節では、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 のマクロ・ファイル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 を指定すると、パスおよびファイル名 macro/sqlm.d2w をもつ Net.Data のマクロが要求されます。

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

構文：

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

等号 (=) は、オプションで、大括弧で示されます。

Net.Data は、パス macro/sqlm.d2w を、MACRO_PATH 構成ステートメントのパスに、Net.Data がマクロ・ファイルを検出するまで、あるいは、すべてのパスを検索するまで、左から右に追加します。Net.Data のマクロの起動に関する情報については、25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* で検出される場合には、完全修飾パスは、*/u/user1/macros/query.d2w* になります。

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は、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、

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

上記の URL が処理依頼されて、ファイル *myfile.txt* が MACRO_PATH で指定されたいずれのディレクトリーでも検出されなかった場合は、Net.Data は、ルート (/) ディレクトリーでファイルを検索しようとします。

```
/myfile.txt
```

EXEC_PATH

EXEC_PATH 構成ステートメントは、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 1 つ以上のディレクトリーを識別します。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・ファイルの 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/rexxprgms.file* ディレクトリーで検出される場合には、プログラムの修飾名は */qsys.lib/rexx.lib/rexxpgms.file/myFunction.mbr* になります。

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、以下が行われます。

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

```
%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 ステートメントが指定された場合は、

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

Net.Data は、現行作業ディレクトリーでファイル rpg1.pgm を検索しようとします。

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る場合は、組み込みファイルの完全修飾名は、/u/user1/includes/myInclude.txt になります。

例 2: 以下の例は、INCLUDE_PATH ステートメントおよび サブディレクトリー名で完全修飾された 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、以下が行われます。

- 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、

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


上記の INCLUDE ステートメントが指定されている場合は、Net.Data は、
/u/user1/includes ディレクトリー内でファイル 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、

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

以下の INCLUDE ステートメントが指定される場合は、Net.Data は、現行作業ディレクトリー内でファイル oeheader.inc を検索しようとします。

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 組み込み関数についての節を参照してください。

環境構成ステートメント

ENVIRONMENT ステートメントは、言語環境を構成します。言語環境とは、Net.Data の構成要素です。Net.Data は、この構成要素を使用して、DB2 データベースのようなデータ・ソースにアクセスしたり、あるいは、REXX のような言語で書かれたプログラムを実行します。Net.Data は、言語環境のセットと、ユーザー自身の言語環境の作成を可能にしてくれるインターフェースを提供してくれます。これらの言語環境については 71 ページの『第6章 言語環境の使用』で説明しています。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 は、ENVIRONMENT ステートメントが、言語環境のための Net.Data の初期設定ファイルに存在することを要求します。それによって初めて、言語環境を起動することができるのです。

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

たとえば、マクロは DATABASE 変数の場所名を指定することができます。DATABASE の値は、SQL 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、幾つかの前提事項を設けます。これらの前提事項は、使用する環境では正しくないこともあります。ステートメントを使用環境に合わせて適切に変更します。

ENVIRONMENT の追加と更新を行うには、以下のようにします。

ENVIRONMENT ステートメントは以下の構文を持っています。

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

パラメーター :

- *type*

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

- *library_name*

Net.Data が呼び出す言語環境インターフェースを含む サービス・プログラムの名前。OS/400 では、サービス・プログラム名に拡張子 *.SRVPGM* を付けて指定します。

- *parameter_list*

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

これらのパラメーターは、FUNCTION ブロック定義で指定されたパラメーターに続けて、*dtw_lei* 構造の *parm_data_array* フィールドで渡されます。(これらの構造に関する情報については、*Net.Data* 言語環境解説書 を参照してください。)

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 INI 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することはありません。Net.Data が言語環境 サービス・プログラムをロードするのは、それが最初に、言語環境を識別する関数を実行するときです。サービス・プログラムは、Net.Data がロードされている限り、ロードされた状態が続けます。

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。

```
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/QTMHSQL.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/QTMHSYS.SRVPGM ( )
```

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。

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 に、Net.Data マクロで参照されるオブジェクトへのアクセス権、および URL が参照するマクロへの適切なアクセス権を与えてください。

さらに具体的にいえば、Net.Data が実行されるユーザー 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およびサービス・プログラムの実行、ならびに実行可能ファイルおよびサービス・プログラムに対するパスにおけるディレクトリー（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

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。以下の方法によって、QTMHHTP1 ユーザー・プロファイル権限が与えられます (V3R2 および V3R7 では、Internet Connection for AS/400 は QTMHHTP1 ユーザー・プロファイルでしか CGI プログラムを実行しません)。

- ルート・ファイル・システムでは、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(QTMHHTP1) DTAAUT(*RX)
CHGAUT OBJ('/WWW/macro') USER(QTMHHTP1) DTAAUT(*RX)
CHGAUT OBJ('/WWW/macro/*') USER(QTMHHTP1) 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(QTMHHTP1) AUT(*USE)
GRTOBJAUT OBJ(WWW/MACRO) OBJTYPE(*FILE) USER(QTMHHTP1) 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。

CL コマンド CHGAUT 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権限を与えることもできます。

```
CHGAUT OBJ('/QSYS.LIB/WWW.LIB') USER(QTMHHTP1) DTAAUT(*RX)
CHGAUT OBJ('/QSYS.LIB/WWW.LIB/MACRO.FILE') USER(QTMHHTP1) DTAAUT(*RX)
```

言語環境固有の権限に関する考慮事項については、71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節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。

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

- 『ファイアウォールを使用する』
- 21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』
- 22ページの『認証を使用する』
- 23ページの『許可を使用する』
- 23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、以下を行います。

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

Net.Data は、ファイアウォール製品、または ユーザーの環境で実行する同等のファイアウォール製品とともに使用されます。

以下の可能な構成では、ご使用の Net.Data アプリケーションの機密保護の管理に対して勧告を提供します。これらの構成は、ハイレベルの情報を提供し、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を構成していることを前提としています。組織のセキュリティー・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**

この構成によって、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 と Web サーバーを分離するサブネットワークが作成されます。ファイアウォール・ソフトウェアを使用して、Web サーバーとパブリック・

インターネットの間にファイアウォールを、DB2 サーバーを含む Web サーバーとセキュア・イントラネットの間に別のファイアウォールを作成します。この構成を、図3 に示します。

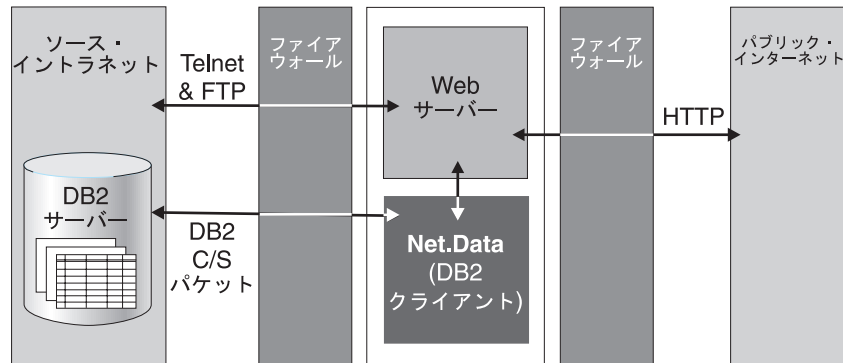


図3. 高度な機密保護構成

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・イントラネットの間の FTP および Telnet アクセスの許可。1つのメソッドは、Web サーバー・マシンに 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することができます。また、出力 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どんなホストにも進むことができることを指定します。
- **中間機密保護構成**

この構成では、ファイアウォール・ソフトウェアが、DB2 サーバーで確保されたイントラネットをパブリック・インターネットから分離します。Net.Data および Web サーバーは、ワークステーション・プラットフォーム上のファイアウォールの外側にあります。この構成は、最初の構成より簡単ですが、それでも、データベース保護が可能です。21ページの図4 は、この構成を示しています。

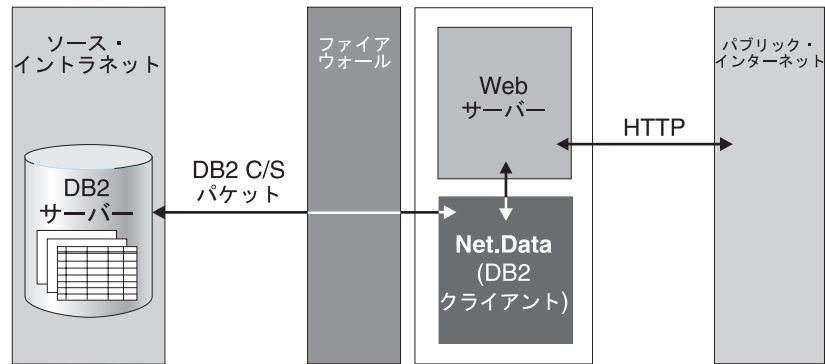


図4. 中間機密保護構成

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、ファイアウォールを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と確保されたイントラネットの外側にインストールされます。これは、外部からの不正なアクセスから保護されません。ファイアウォールは、この構成にはパケット・フィルタ一規則を必要としません。図5 は、この構成を示しています。

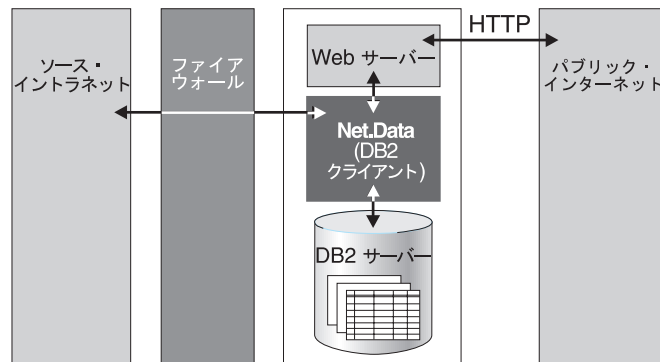


図5. 単純な機密保護構成

ネットワーク上のユーザーのデータを暗号化する

セキュア・ソケット・レイヤー (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、アプリケーション内のデータへのアクセスおよび更新を、Net.Data 要求を行うユーザー ID に許可するために使用します。認証は、要求が有効なユーザー ID から出されているかどうかを検査するために、ユーザー ID とパスワードを突き合わせる処理です。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 は関連付けされるようになります。

クライアントを基にした認証

ユーザーは、ローカル OS/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されているため、ユーザー ID またはパスワードを入力するようなユーザーへのプロンプト指示はされません。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) のもとで CGI プログラムを実行します。しかし、UserID ディレクティブが有効であるか、UserID サブディレクティブが指定され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づけられています。この認証タイプには、ユーザーのグループまたは要求のクラスに適切なアクセス権限をもつ代理ユーザー ID の作成が必要です。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・ファイルを保護するには、以下を行ってください。

1. Net.Data プログラム・オブジェクトに対する Web サーバーの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、17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

許可は、オブジェクト、資源、または関数への完全なアクセスおよび制限付きのアクセスをユーザーに提供します。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したユーザー ID が、22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としています。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

上記の方法のほかに、パス・ステートメントおよび隠し変数をはじめ、HTML フォームまたは SQL ステートメントを使用する方法などの Net.Data 提供のメカニズムを使用することができます。

パス・ステートメント

Net.Data は、パス構成ステートメントの設定から、Net.Data マクロ・ファイル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、フラット・ファイル、または組み込みファイルを探すときに Net.Data が検索する1つまたは複数のディレクトリーを識別します。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。

隠し変数

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、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。詳しくは43ページの『隠し変数』を参照してください。

以下のメソッドを使用して、保護方式をセットアップすることもできます。

HTML フォーム

Net.Data を使用して、ユーザー独自の保護方式を作成します。たとえば、HTML フォームを通してユーザーから妥当性検査情報を要求し、データベースのデータを使用して、または Net.Data から呼び出される外部プログラムを通して、この情報の妥当性検査を行います。

SQL ステートメント

SQL ステートメントによって、他のユーザーがデータベースに送信することができる資産を保護します。たとえば、SELECT ステートメントを 2 つのテーブルに制限します。

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

Net.Data for OS/400 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・ファイル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、永続的マクロや、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、89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。

Net.Data マクロ・ファイルの名前、および Net.Data マクロ内で実行される HTML ブロックの名前は、リンク、フォーム、または URL 内で指定されます。

本章では、マクロ・ファイルでの Net.Data の呼び出しを説明します。

- 『マクロ・ファイルで Net.Data を呼び出す (マクロ要求)』
- 28ページの『永続的マクロの起動』

マクロ・ファイルで Net.Data を呼び出す (マクロ要求)

このセクションでは、マクロ・ファイルを指定した Net.Data の起動の方法を示します。マクロ要求スタイルによる起動に対しては、HTML リンク、HTML フォーム、または URL を使用して Net.Data を呼び出すことができます。

以下の構文ステートメントに、Net.Data の異なる起動方法を示します。

- HTML リンク:

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

- HTML フォーム:

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

- 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/` です。

filename

Net.Data マクロ・ファイルの名前を指定する。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。詳細については、12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・ファイルの中の 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 つまたは複数のオプション・パラメーターを指定します。

例

以下の例では、さまざまな方法で Net.Data を呼び出します。

例 1: HTML リンクを使用して Net.Data を呼び出す

```
<A HREF="http://MyServer/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://MyServer/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 マクロを起動する

```
<A
HREF="http://MyServer/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://MyServer/cgi-bin/db2www/
  qsys.lib/mylib.lib/myfile.file/myMacro.mbr/report">
.
.
.
</FORM>
```

ご注意: URL は、実際のコードでは 1 行の連続したストリングでなければなりません。ここでは、読みやすくするために 2 行に分割してあります。

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

- 『HTML リンク』
- 『HTML フォーム』

HTML リンク

マクロ・ファイルで HTML `<a>` タグを使用することによって、HTML ブロックを実行する Web ページの中のリンクを作成します。マクロおよび HTML ブロックを指定する HREF 属性を使用することによって、さらにいくつかのテキストまたはリンク・タグ内のイメージまでも含むことによって、ユーザーはこのリンクを作成することができます。このメソッドは、Web ページがブラウザに表示される時に、『hot spot』としてテキストまたはイメージを識別します。ブラウザでユーザーがテキストやイメージをクリックする場合には、Net.Data はそのマクロ内の HTML ブロックを実行します。

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

```
<a href="http://server/cgi-bin/db2www/listA.d2w/report">
「モニターをすべてリストする (List all monitors)」</a>
```

このリンクは、以下のマクロを呼び出します。

```
%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、61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="hdware" VALUE="MON" checked> Monitors
<LI><INPUT TYPE="RADIO" NAME="hdware" 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、「実行 (Submit)」ボタンをクリックすると、Web ブラウザーは FORM タグの ACTION パラメーターを処理し、Net.Data が起動します。次に Net.Data は、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

本節では、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する関数が含まれます。これらのマクロの起動は通常のマクロ要求と同様であり、サーバー、マクロ・ファイル、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、89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/ です。

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 を指定する。
この識別子は、DTW_RTVHANDLE 組み込み関数を呼び出すことによって取得します。これは、*Net.Data_invocation_path* の後ろに続け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する。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。詳細については、12 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・ファイルの中の 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 つまたは複数のオプション・パラメーターを指定します。

例

以下の例に、永続的マクロの起動方法を示します。

例 1: マクロ・ファイル内の URL

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

例 2 同じ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 に組み込まれる、またはマクロ・ファイルで定義される関数を呼び出す。
- 処理出力をフォーマットして、表示用に Web ブラウザーに戻す。

Net.Data のマクロには、図6 に示すように、宣言パートおよび表示パートの 2 つの編成パートが含まれます。

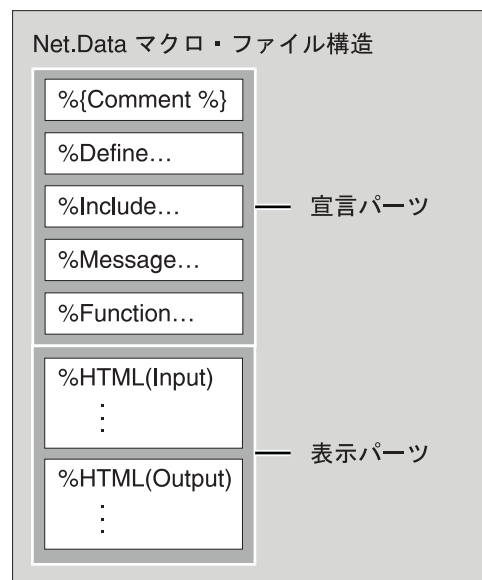


図6. マクロ・ファイル構造

- 宣言パート には、マクロ・ファイルにおける変数および関数の定義が含まれる。
- 表示パート には、Web ページのレイアウトを指定する HTML のブロックが含まれる。HTML ブロックは、HTML や JavaScript などの Web ブラウザーによってサポートされ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・ファイル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、17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・ファイル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。

- 32ページの『*Net.Data* のマクロ・ファイルの分析』

- 37ページの『Net.Data のマクロ変数』
- 48ページの『Net.Data の関数』
- 59ページの『マクロでの Web ページの生成』
- 66ページの『マクロ・ファイルにおける条件付き論理とループ』

Net.Data のマクロ・ファイルの分析

マクロ・ファイルは、次の 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 ブロックなどの、ほかの言語構成要素とステートメントが含まれます。言語構成要素についての詳細は、*Net.Data* 解説書の言語構成要素に関する章を参照してください。

許可のためのヒント： Web サーバーが、確実に、EXEC および INCLUDE ステートメントによって参照されるファイルへのアクセス権を持つようにします。詳しくは、17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、マクロからのエンタリー・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 を起動するときには、マクロ・ファイルの処理のためにエンタリー・ポイントとして、HTML ブロック名を指定します。HTML ブロックは、35ページの『HTML ブロック』で説明されています。

本節では、簡単な Net.Data のマクロを使って、マクロ言語の要素を例示します。このマクロの例は、REXX プログラムに渡す情報をプロンプト指示するフォームを提供します。このマクロは、この情報を OMPSAMP.MBR と呼ばれる REXX の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。これによって、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、CGI を介してマクロにアクセスします。Net.Data には、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 つのパラメーターが必要です。

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

DEFINE ブロック

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

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

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

FUNCTION ブロック

FUNCTION ブロックには、HTML ブロックで呼び出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。

以下の例は、外部の REXX プログラムの関数呼び出し、およびマクロ・ファイルに含まれる関数の関数呼び出しを定義している 2 つの FUNCTION ブロックを示しています。

```
%{ ***** FUNCTION Block *****%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result) { <-- This function accepts
                                                         one parameter and returns a
                                                         result which is substituted
                                                         for the associated function
                                                         call
    %EXEC{ompsamp.mbr %} <-- The function executes an external REXX program
                           called "ompsamp.mbr"
}%

%FUNCTION(DTW_REXX) today () RETURNS(result) {
    result = date() <-- The single source statement for this function is
                       contained inline.
}%
```

最初の関数ブロック `rexx1` は、REXX 関数の宣言です。この宣言は、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 つの変数を戻します。REXX コマンドにおける `result` 変数の内容は、Output ブロックに含まれる `@rexx1()` 関数呼び出しの起動に置き換わります。`ompsamp.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、@rex1() 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 のプログラム全体 (1 行全部) は、関数宣言それ自身に含まれています。外部プログラムは必要ありません。インライン・プログラムは、REXX および Perl の関数では許可されています。その理由は、これらのプログラムは、動的な解析および実行ができるインタープリター言語だからです。インライン・プログラムは、管理すべき独立したプログラムの必要がないので、単純であるという優位点を持っています。最初の REXX の関数は、インラインとしてハンドルすることも可能です。

HTML ブロック

HTML ブロックは、Web ページ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ー・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 呼び出し要求で指定され、マクロごとに少なくとも 1 つの HTML ブロックがなければなりません。

例であるマクロ・ファイルの最初の HTML ブロックは、INPUT という名前が付いています。INPUT ブロックには、1 つの入力フィールドを持つ簡単な形式の HTML が含まれます。

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

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

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

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

ブロック全体は、HTMLのブロック識別子、%HTML (INPUT) {...%} で囲まれます。INPUT は、このブロックの名前を識別します。これに名前を付けることができます。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、このフィールドに入力されたデータは、変数 input_data に入れられ、HTML 出力ブロックに渡されます。

マクロ・ファイルが同じ Web サーバー上にある場合、相対参照を持つ他のマクロ・ファイルの 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・ファイル 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>
```

ほとんどの CGI プログラムの場合と同じように、入力データを受け取るために、環境変数を定義する必要はありません。Net.Data は、ユーザーに代わって環境変数をハンドルしてくれます。ユーザーに要求されるのは、単に変数の名前を参照することだけです。

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

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

  </head><body>
  <h1>Output Page</h1>
  <p>@rex1(input_data) <--- This line contains a call to function rex1
                                passing the argument "input_data".

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

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 のマクロ・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、渡される変数の数と型を決定します。

Net.Data のマクロ変数

`Net.Data` により、`Net.Data` でマクロの変数を定義したり、参照することができます。さらに、これらの変数を、マクロから言語環境に渡したり、また言語環境から受け取ったりすることができます。変数名と値、および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カテゴリー化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、この変数を変更することはできません。
- 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、最初の参照時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・データあるいは URL のデータの名前と値のペアに対応する変数。
 - `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。

以後の節では、以下について説明します。

- 『変数の効力範囲』
- 39ページの『変数の定義』
- 40ページの『変数の参照』
- 41ページの『変数の型』

変数の効力範囲

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・ファイル内の任意の場所でそれを参照できる場合、グローバルな効力範囲を持ちます。グローバルな効力範囲を持つ識別子には、以下のよう
なものがあります。

- Net.Data の組み込み関数
- フォームのデータ
- URL 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・ファイル

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。) マクロ・ファイルの効力範囲を持つ識
別子は、それが宣言された点から、マクロ・ファイルの終わりまでが可視になり
ます。

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

識別子は、以下の場合、関数ブロックの効力範囲を持ちます。

- 識別子が関数定義のパラメーター・リストで宣言される。

同じ名前を持つ識別子が、関数定義の外に存在する場合は、Net.Data は、その関
数ブロック内の関数パラメーター・リストの識別子を使用します。

- 識別子が関数ブロックでインスタンス化されるが、関数呼び出し前に宣言また
は初期化されていない。

識別子は、それが関数の外で宣言されたり、初期化されている場合、および関数
のパラメーター・リストで宣言されてもいない場合は、関数ブロックの効力範囲
を持ちます。識別子が、関数ブロックの内部で使用される場合、関数呼び出しの
前に割り当てられていた値を保持します。関数ブロック内で更新された場合、識別
子は、関数呼び出しの後では新規の値を保持します。

- REPORT ブロック

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

- ROW ブロック

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

識別子が参照されると、識別子は、その識別子の値で置き換えられます変数への参
照が、それに関連付けられている値を持たない場合、あるいは関数呼び出しが戻り
値を持たない場合、その参照は空ストリングで置き換えられます。

変数の定義

Net.Data の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**

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。この構文は、Net.Data に固有のもので

```
%DEFINE variable_name="variable value"
```

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

variable_name は、ユーザーが変数に与える名前です。変数名は、文字あるいは下線で始まらなければなりません。そして、任意の英数字および下線を含むことができます。すべての変数名は、大文字小文字の区別をします。ただし、表変数の、*N_columnName* および *V_columnName* は除きます。

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。連続する引用符が 2 つだけの場合は、ヌル・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 HI="say ""hello"""
```

変数 HI は、say "hello" を表示します。

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

変数 reply は、hello を表示します。

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

変数 empty は、ヌルです。

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

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

- **HTML のフォームの SELECT および INPUT タグ**

HTML のフォームに使用される SELECT および INPUT タグを使用することができます。以下の例では、標準の HTML フォームのタグを使用して、変数を定義しています。

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

あるいは

```
<SELECT NAME="variable_name">
```

`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、27ページの『HTML フォーム』を参照してください。

INPUT あるいは SELECT タグから受け取った変数の値は、Net.Data のマクロにおいて DEFINE ステートメント により設定された変数の値を上書きします。

• URL のデータ

Net.Data のマクロを URL 要求として呼び出し、ユーザー ID のような変数を URL に組み込み、Net.Data に送信することができます。たとえば、以下のようにします。

```
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 が入力ステートメントから受け取った追加データを指定します。Net.Data は、データを受け取り、そのデータを Net.Data がデータを形成するように処理します。

変数の参照

定義済みの変数を参照して、その値を戻すことができます。

Net.Data のマクロにおいて変数を参照するには、その変数名を `$(および)` 内に指定します。たとえば、以下のようにします。

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

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

変数をテキスト表示ステートメントの一部として使用するには、HTML ブロックでそれらの変数を参照します。たとえば、すでに変数 `homeURL` を定義している場合、以下のようにします。

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

ホームページは `$(homeURL)` として指定することができ、以下のようにリンクを作成します。

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

変数は、Net.Data のマクロの任意のパーツで参照することができます。変数が、それが参照されたときに未定義である場合、Net.Data は空ストリングを戻します。Net.Data は変数を定義しません。

制約事項：循環参照（あるいは、循環）は許可されていません。たとえば、以下の DEFINE ステートメントは、値が参照され、最終値が評価されたときに、エラーとなります。

```
%DEFINE a="$(b)"  
%DEFINE b="$(a)"
```

変数の型

マクロ・ファイルでは、以下の変数参照のタイプを使用することができます。

- 『条件変数』
- 『環境変数』
- 42ページの『実行可能な変数』
- 43ページの『隠し変数』
- 44ページの『リスト変数』
- 45ページの『表変数』
- 46ページの『各種変数』
- 46ページの『表処理変数』
- 47ページの『レポート変数』
- 48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う動作しません。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、44ページの『リスト変数』 を参照してください。

環境変数

Net.Data が実行されている処理に存在する Net.Data の環境変数を参照することができます。

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
```

ただし、`var` は定義される変数の名前です。

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見えます。

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

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

実行可能な変数

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

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

```
%DEFINE runit=%exec "testProg"
```

`runit` は、実行可能な変数になります。

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

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

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

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

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

Net.Data がマクロ・ファイル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* は、*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を検索する。詳細については、13ページの『`EXEC_PATH`』を参照してください。

2. Net.Data がプログラムを見つけなかった場合は、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、システムが検索します。実行可能なプログラムが見つければ、Net.Data は、そのプログラムを実行します。

制約事項： 実行可能変数を、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` の値はヌルです。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、ヌルになります。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、隠し、およびリスト変数の使用例を示したものです。

```

%HTML(INPUT){
<FORM METHOD="POST" ACTION="/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 はヌルとなり、照会では、whereClause もまたヌルになります。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。表変数は、同一のレコード、すなわち行の配列と、各行のフィールドを説明する列名の配列を含みます。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)
```

表を定義するときに、表がゼロの行とゼロの列を持つと、記憶域が割り当てられません。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 あるいは DTW_PERL など、他のすべての環境の場合は、言語環境が自動的に表の値を設定することはありません。そのかわり、表の個々の要素は、ネイティブの言語のインタープリターから出力パラメーターとして利用できるようにすることができます。そのためには、表の個々の要素は、スクリプトあるいはプログラムを実行して設定されなければなりません。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、71ページの『第6章 言語環境の使用』を参照してください。

表変数の名前を参照することにより、関数間で表を渡すことができます。表の個々の要素は、関数の REPORT ブロックで参照することができます。詳細については、46ページの『表処理変数』を参照してください。表変数は、通常、SQL 関数の値が代入され、次に、SQL 関数あるいは別の関数のいずれかにパラメーターとして渡された後、その関数におけるレポートへの入力として使用されます。表変数を、IN、OUT、あるいは INOUT パラメーターとして、任意の非 SQL 関数に渡すことができます。表は、SQL 関数には、OUT パラメーターとしてのみ渡すことができます。

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。配列 は 0 から開始する、という標準 C および C++ の言語規則とは異なります。

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。

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

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

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

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

表処理変数

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

表処理変数には、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・ファイル内で使用します。たとえば、処理中の現行列の名前を検索するには、以下を使用することができます。

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


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

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

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

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

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

レポート変数

`Net.Data` は、デフォルトのレポート形式のマクロ・ファイルから生成される Web ページ出力を表示します。デフォルトのレポート形式は、`<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。言語環境変数は、それらを参照する前に、`DEFINE` ステートメント、あるいは `@DTW_ASSIGN()` 関数で定義されなければなりません。適切な `Net.Data` のマクロ・ブロックにおいて、言語環境変数を設定あるいは参照します。

言語環境変数を使用すると、データベースとの接続の確立、それに `NLS` サポートを使用可能にする、などのタスクを実行することができます。

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` マクロでは、以下の 2 つのタイプの関数を使用することができます。

ユーザー定義関数 (UDF)

外部プログラムまたはストアド・プロシージャの呼び出しなど、アプリケーションとともに使用するためにユーザーが定義する関数。

Net.Data 組み込み関数

ワードおよびストリングを操作する関数や表変数の取得や設定を行う関数など、アプリケーションで使用するために `Net.Data` が提供する関数。

本節では、以下のトピックについて説明します。

- 『ユーザー定義関数の定義』
- 53ページの『関数の呼び出し』
- 55ページの『`Net.Data` 組み込み関数の呼び出し』

ユーザー定義関数の定義

ユーザー自身の関数をマクロ・ファイルで定義するには、`FUNCTION` ブロック または `MACRO_FUNCTION` を使用します。

FUNCTION ブロック

`Net.Data` のマクロから呼び出され、言語環境で処理されるか、あるいは外部プログラムを呼び出すサブルーチンを定義します。

MACRO_FUNCTION ブロック

`Net.Data` のマクロから呼び出され、別の言語環境ではなく、`Net.Data` により

処理されなければならないサブルーチンを定義します。ブロック内のステートメントは、Net.Data のマクロ言語のソース・ステートメントでなければなりません。

MACRO_FUNCTION ブロックは、パフォーマンスを改善できる FUNCTION ブロックの代替です。MACRO_FUNCTION ブロックは Net.Data によってのみ処理され、言語環境を呼び出しません。これらの 2 つの構成に関する詳細については、*Net.Data* 解説書 を参照してください。

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 parameter, ...) [RETURNS(return-var)] {  
    executable-statements  
    report-block  
    ...  
    report-block  
  
    message-block  
%}
```

MACRO_FUNCTION ブロック :

```
%MACRO_FUNCTION function-name(usage parameter, ...) {  
    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。

幾つかのデフォルトの言語環境が Net.Data で提供されています。

function-name

FUNCTION あるいは MACRO_FUNCTION ブロックの名前を指定します。FUNCTION ブロックあるいは MACRO_FUNCTION ブロック を、関数呼び出しと共に、マクロ・ファイルのどこか別の場所で実行します。関数呼び出しは、@ 記号を前に付けた *function-name* を参照します。詳細については、53ページの『関数の呼び出し』 を参照してください。

複数の FUNCTION あるいは MACRO_FUNCTION ブロックを、同じ名前で定義し、それらを同時に実行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 ブロックあるいは同じ名前を持つ 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、パラメーターが、FUNCTION ブロック、MACRO_FUNCTION ブロック、あるいはその両方に渡されるのか、あるいはそこから受け取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。

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。パラメーター変数の参照は、FUNCTION ブロック あるいは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、関数呼び出しに割り当てられ、Net.Data のマクロ処理時に関数呼び出しが、この値に置き換えられます。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
- 戻りコードが非ゼロの場合は、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。実行可能ステートメントを言語環境に渡す前に、Net.Data は、これらの変数参照または実際の値をもつ関数呼び出しを取り替えます。

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、42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、HTML ステートメントと Net.Data の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。その理由は、Net.Data は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評価し、それを実行するからです。

report-block

FUNCTION ブロックの出力を処理するために、1 つまたは複数の REPORT ブロックを定義してください。 61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。 52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 Net.Data のマクロ・レイヤーと、Net.Data の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。

以下のメソッドの 1 つを使用して、COMMENT ブロックの区切り文字を使用するか、あるいは、Net.Data が特殊文字として解釈する区切り文字をもたずに、ユーザーの組み込みプログラム・コードとしての Net.Data の特殊文字を使用します。

- インラインでコードをプットするのではなく、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});  
    }  
    ...  
    %}
```

確実に、Net.Data が `%{` 文字を Net.Data の COMMENT ブロックの区切りとしてではなく、Perl のソース・コードとして解釈できるようにするには、以下の方法のいずれかで、関数を再書き込みします。

- %EXEC ステートメントを使用する。

```
%function(DTW_PERL) func() {  
    %EXEC{ func.prl %}  
    %}
```

- 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。Net.Data は、以下のメッセージ・ブロック処理を使用します。

1. Net.Data は、FUNCTION ブロックへの関数呼び出しごとに、言語環境変数 RETURN_CODE を設定する。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。
2. 言語環境が、戻りコード値を Net.Data に渡すと、Net.Data は、RETURN_CODE の値を、戻りコード値に設定する。
3. 関数呼び出しが完了すると、MESSAGE ブロックは、RETURN_CODE の値を使用して、進め方を決定する。

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、抜け出ます。

以下の例は、グローバルな MESSAGE ブロックと、関数の MESSAGE ブロックを持つ Net.Data のマクロの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・テキストをプリントしてから、`Net.Data` マクロの処理を続けます。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。FUNCTION ブロック名あるいは MACRO_FUNCTION ブロック名の前に付いている @ 文字を使用してください。

```
@function_name([ argument,... ])
```

```
function_name
```

これは、呼び出す FUNCTION ブロックまたは MACRO_FUNCTION ブロッ

クの名前です。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、定義済みの変数、リテラル文字ストリング、変数参照、あるいは関数呼び出しの名前です。関数呼び出し時の引き数は、FUNCTION ブロック、あるいは MACRO_FUNCTION ブロックのパラメーターと一致しており、各パラメーターは、FUNCTION ブロックあるいは MACRO_FUNCTION ブロックの処理中に、対応する引き数の値が割り当てられます。引き数は、対応するパラメーターと同じ数および型でなければなりません。

Net.Data は、FUNCTION ブロック、MACRO_FUNCTION ブロック、あるいは関数呼び出しに関連付けられている組み込み関数を、次の順で処理します。

1. Net.Data は、FUNCTION ブロックの実行可能ステートメントのセクションにおける変数参照と関数呼び出しを処理する。Net.Data は、すべての変数参照を、変数の現行値で置き換え、すべての関数呼び出しを実行し、すべての関数呼び出しを関数呼び出しの戻り値と置き換えます。変数参照と関数呼び出しは、それらが指定された順に処理されます。このステップ中は、Net.Data は、組み込み関数あるいは MACRO_FUNCTION ブロックを処理しません。

2. ネイティブの言語プロセッサは、実行可能ステートメントのセクションを処理する。FUNCTION ブロックの場合、プロセッサは、SQL、REXX、あるいは Perl などの FUNCTION ブロックで指定された言語環境に対応します。MACRO_FUNCTION ブロックの場合、Net.Data は、言語プロセッサとして働き、実行可能ステートメントを実行します。組み込み関数は、実行可能ステートメントを持ちません。Net.Data は、関数名で組み込み関数を処理します。

Net.Data は、関数のパラメーターを、ネイティブの言語プロセッサに渡します。Net.Data は、IN および INOUT の場合にのみ、ネイティブ言語プロセッサに値を渡し、OUT および INOUT の場合にのみ、ネイティブの言語プロセッサから戻り値を受け取ります。

3. Net.Data は、言語プロセッサからの戻りコードと言語プロセッサから戻されたメッセージに基づき、暗黙の RETURN_CODE および DTW_DEFAULT_MESSAGE 変数を設定する。Net.Data は、これらの変数を、MACRO_FUNCTION ブロックに対しては設定しません。

4. FUNCTION ブロックまたは MACRO_FUNCTION ブロックが 1 つまたは複数の REPORT ブロックを含む場合、またはデフォルト・レポートの生成を指定する場合は、Net.Data は、すべての参照出力パラメーターについて新しい値を使用してレポートを処理します。Net.Data は、組み込み関数に対してはレポートを生成しません。

5. FUNCTION ブロックが、ローカルな MESSAGE ブロックを含む場合、Net.Data は、その MESSAGE ブロックを処理する。Net.Data は、以下の条件のどれか 1 つが発生した場合、グローバルな MESSAGE ブロックを処理します。

- グローバルな MESSAGE ブロックは指定されるが、戻りコードは、ローカルな MESSAGE ブロックではハンドルされない。
- 組み込み関数が呼び出される。

Net.Data は、MACRO_FUNCTION ブロックに対して MESSAGE ブロックを処理しません。

6. Net.Data は、関数呼び出しを、関数の戻り値で置き換える。FUNCTION ブロックの場合、この値は以下のどれか 1 つになります。

RETURNS パラメーター値

RETURNS キーワードを持つ FUNCTION ブロックに置き換えられます。

空ストリング ("")

RETURN_CODE がゼロの場合、RETURNS キーワードを持たない FUNCTION ブロックに置き換えられます。

RETURN_CODE

RETURN_CODE がゼロでない 場合、RETURNS キーワードを持たない FUNCTION ブロックで置き換えられます。

MACRO_FUNCTION ブロックの場合、関数呼び出しは、実行可能ステートメントのセクションの処理結果に置き換えられます。

組み込み関数の場合、その値は、組み込み関数のフォーマットに依存します。

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、FUNCTION ブロックではそれらの関数を定義する必要はありません。ユーザー定義の関数を呼び出すことができるのなら、マクロのどこからでもこれらの関数を簡単に呼び出すことができます。

ユーザー定義関数を呼び出すために使用する組み込み関数を呼び出す同じ方法、つまり Net.Data 関数呼び出しを使用します。図7は、Net.Data 関数およびマクロ・ファイルがどのようにして対話しているかを示しています。

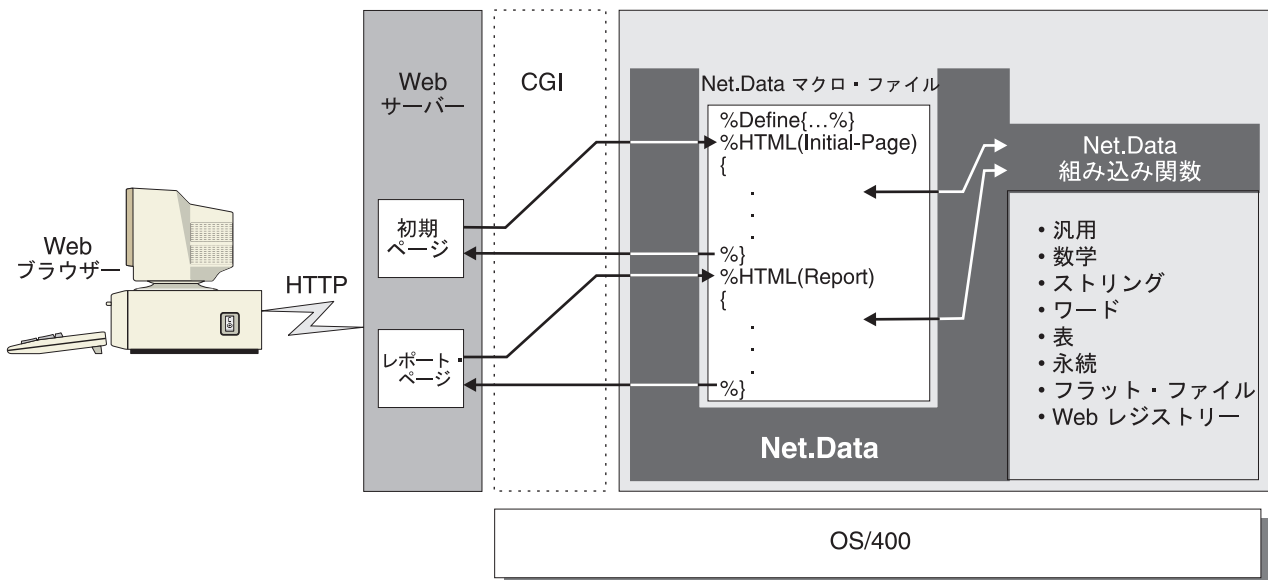


図 7. Net.Data の組み込み関数

HTML ブロック内部から、`Net.Data` は、組み込み関数に対する関数呼び出しを処理します。`Net.Data` は、関数を処理して、マクロファイルの HTML ブロックにすべての結果を戻します。

組み込み関数は、3 つの方法で、その結果を戻すことができます。各関数とその結果をどのように戻すかを接頭部により示すことができます。

- **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タイプを判別するには、*Net.Data* 解説書 の `Net.Data` 組み込み関数についての章を参照してください。

以下の節では、`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、ストリング、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。

- 『汎用関数』
- 57ページの『数学関数』
- 57ページの『ストリング関数』
- 58ページの『ワード関数』
- 58ページの『表関数』
- 58ページの『フラット・ファイル関数』
- 59ページの『Web レジストリー関数』
- 59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、照会、環境変数の設定、HTML のエスケープ・コードの使用、およびシステムからの有益な情報の取得、を行うことができます。

たとえば、特定の条件が起こった場合に、`Net.Data` がマクロ・ファイルの残りの部分を処理せずにマクロを終了するように指定するには、`DTW_EXIT` 関数を使用します。

```
%HTML(cache_example) {

    <html>
    <head>
    <title>This is the page title</title>
    </head>
    <body>
    <center>
    <h3>This is the Main Heading</h3>
    <!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
    <! Joe Smith sees a very short page                !>
    <!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
    %IF (customer == "Joe Smith")
    </body>
    </html>

@DTW_EXIT()

%ENDIF

...

    </body>
    </html>
    %}
}
```

DTW_URLESCSEQ 関数は、URL で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_rPOWER("2", "-3", result)
```

DTW_POWER は ".125" を戻します。

ストリング関数

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

たとえば、DTW_ASSIGN を使用して、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。ただし、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。

その他のワード関数には、ストリング内の文字の数を戻す DTW_WORDLENGTH や、ストリング内のワードの位置を戻す 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。これらの関数を使用して、Net.Data 表変数の値を入手して設定することもできます。表変数には、値の配列、およびそれらに関連付けられている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するので、他のプロセスはファイルを読み込んだり更新したりすることはできません。Net.Data が DTWF_CLOSE で終了すると、DTWF_CLOSE はファイルを解放するので、他のプロセスがファイルにアクセスできるようになります。

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行え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_DELENTY 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。

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。以下の節では、マクロの HTML および REPORT ブロックについて説明し、Net.Data 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 のマクロ・ファイルには、HTML などのテキスト表示ステートメントを Web ブラウザーに対して生成する HTML ブロックおよび HTML ブロックの構成が含まれます。マクロ・ファイルでは、最低でも 1 つの HTML ブロックを指定しなければなりませんが、任意の数のブロックを指定することができます。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、呼び出されるたびに 1 つの HTML ブロックしか処理せず、HTML ブロックの内容は、それ以降の Net.Data の呼び出しによって制御されます。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ためには、Net.Data を複数回呼び出して、リンクやフォームなどの標準 HTML のナビゲーション手法を使用して HTML ブロックを処理することができます。

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">
<dl>
<dt>What hardware do you want to list?
<dd><input type="radio" name="hardware" value="MON" checked>Monitors
<dd><input type="radio" name="hardware" value="PNT">Pointing devices
<dd><input type="radio" name="hardware" value="PRT">Printers
<dd><input type="radio" name="hardware" value="SCN">Scanners
</dl>
<HR>
<input type="submit" value="Submit">
</FORM>
%}

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

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

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

```
<a href="http://www.ibm.com/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・ファイルを解析します。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、HTML ステートメントとみなし、表示するためにブラウザーに送信します。

ユーザーが、選択を行い、「処理依頼 (Submit)」ボタンを押すと、Net.Data は、HTML の FORM 要素の ACTION パーツを実行します。このパーツは、Net.Data のマクロの HTML(REPORT) 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(REPORT) 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 は再度 HTML レポートのブラウザへの送信を開始し、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 が REPORT ブロックの処理を完了した後で、Net.Data は、HTML(REPORT) 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、HTML タグ、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す。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 ブロックのパラメーター・リストの最初の出力表の表データを使用します。FUNCTION ブロックで表を指定しない場合は、Net.Data は、デフォルトの表データを使用します。

REPORT ブロックは、次の 3 つのパーツを持ち、各パーツはオプションです。

- 表の行データの前に一度だけ表示される HTML データを含むヘッダー情報。
- 結果表の行ごとに一度だけ表示される HTML および表変数を含む ROW ブロック。
- 表の行データの後に一度だけ表示されるデータを含むフッター情報。

例:

```
%REPORT{
<H2>Query Results</H2>
<P>Select a name for details.
<TABLE BORDER=1>
<TR><TD>Name</TD><TD>Location</TD>
%ROW{
<TR>
<TD>
<a href="/cgi-bin/db2www/name.mac/details?name=$(V1)&location;=$(V2)">$(V1)</a></TD>
<TD>$(V2)</TD>
%}
</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はどれも表示をしないようにするには、ROW ブロックを空にしておくか、ROW ブロックを完全に省略します。
- Net.Data 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、46ページの『表処理変数』で説明されています。追加詳細については、*Net.Data 解説* のレポート変数のセクションを参照してください。
- ヘッダーおよびフッター情報を提供するには、ROW ブロックの前後にテキストを提供します。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

口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に送信します。

- REPORT ブロックのユーザー定義関数および変数を使用することができます。
- Net.Data に、フォーマット済みのテキストを使用してデフォルトのレポートをプリントさせるには、マクロ・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プリントを使用禁止にするには、DTW_DEFAULT_REPORT を NO に設定するか、空の REPORT ブロックを指定します。たとえば、以下のようになります。

```
%REPORT{%}
```

例: レポートのカスタマイズ

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

```
%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: $(NUM_ROWS)
    %}
    %}
```

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

```
Phone Query Results:
=====
Name: Doen, David
Phone: 422-245-1293
Fax: 422-245-7383
-----
Name: Ramirez, Paolo
Phone: 955-768-3489
Fax: 955-768-3974
-----
```


Name: Wu, Jianli
Phone: 525-472-1234
Fax: 525-472-1234

Total records retrieved: 3

Net.Data は、以下を行い、レポートを生成しました。

1. *Phone Query Results:* を、レポートの最初に 1 回プリントする。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. 変数 V1、V2、そして V3 に、Name、Phone、および Fax の値をそれぞれ、検索時に各行ごとに与える。このテキストは、REPORT ブロックのフッター部分です。
3. 各取得行の後ろに罫線を引き、読みやすくする。
4. スtring *Total records retrieved:* および NUM_ROWS の値を、レポートの最後に 1 回だけプリントする。(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。

一般に、DTW_SQL 言語環境で複数の REPORT ブロックを使用し、関数からストアド・プロシージャを呼び出すと、複数の結果セットが戻されます (77ページの『ストアド・プロシージャ』を参照してください)。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、関数パラメーター・リストに Net.Data 表変数を渡さなければなりません。指定したレポート・ブロックの数より多い表をパラメーター・リストに渡す場合に、DTW_DEFAULT_REPORT = "MULTIPLE" である場合は、デフォルトのレポートがレポート・ブロックと関連がない各表に対して生成されます。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 つしか生成されません。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: 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" が指定されているので、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。

例 3: MACRO_FUNCTION ブロック

```
%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。

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

    %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: 無名 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 ブロックでテーブルの名前が指定されていないので、レポートは、FUNCTION ブロック・パラメーター・リストに示された順序 (最初に **table1**、次に **table2**) で 2 つの表について表示されます。

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および 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。FUNCTION パラメーター・リストに渡された表の数よりも指定された REPORT ブロックの数が少ないので、デフォルトのレポートが、FUNCTION ブロック・パラメーター・リストに示されている順序 (最初に table1 に対するデフォルトのレポート、次に table3 に対するデフォルトのレポート) で残りの表について表示されます。

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場合は、以下のガイドラインおよび制約事項を使用してください。

ガイドライン:

- 表ごとに 1 つの REPORT ブロックを指定する。REPORT ブロックに指定された表名パラメーターは、FUNCTION ブロック・パラメーター・リストの対応する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 ブロックを指定する。
- 表に REPORT ブロックが指定されていないときにデフォルトの処理を指定するには、DTW_DEFAULT_REPORT = "MULTIPLE" を定義する。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 つの表に対してしかデフォルトのレポートは生成されません。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" を設定する。
- 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。

- 複数のレポート・ブロックを、Net.Data for OS/400 がサポートする任意の言語環境とともに使用することができます。

制約事項：

- レポート変数は、全関数に設定され、すべての REPORT ブロックの処理とその変数が処理する表に影響を及ぼします。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 ブロックを突き止めなければならないのは、REPORT ブロックのリストの前または後のどちらかであって、REPORT ブロックの間ではありません。
- 表変数は、関数で使用する前に、TABLE ステートメントで定義しなければなりません。

マクロ・ファイル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。

- 『条件付き論理』
- 68ページの『ループ構成体』

条件付き論理

IF ブロックを使用して、Net.Data のマクロで、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 ブロック構文の規則は、マクロ・ファイルでのブロックの位置によって決定します。IF ブロックの実行可能なステートメント・ブロックに許される要素は、IF ブロック自身の位置に依存します。IF ブロックを含むブロック内で有効な要素ならどれでも、その IF ブロックで有効です。たとえば、IF ブロックを HTML ブロック内で指定する場合、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
```

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
```

IF ブロックは、宣言パーツ内のほか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どんな要素でも使用することができます。IF block は、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らないかもしれません。

```
%IF
...
%HTML block
...
%IF block
```

例外： IF ブロックが REPORT ブロック内にある場合は、IF ブロック内に、ROW ブロックを指定しないでください。

Net.Data は、IF ブロック条件リストを、条件を構成している項の内容に基づき、2 つの方法のうちのいずれか 1 つで処理します。デフォルトのアクションは、すべての項を、ストリングとして処理し、条件で指定されたストリング比較を実行します。しかし、以下の 2 つの条件が満足される場合は、Net.Data は、数値比較を実行します。

- 条件がバイナリー操作 (<, >, <=, >=, !=, ==) の場合。
- 条件に含まれる項が共に整数を表している場合。この意味は、項は数字からなるストリングで、オプションとして、その前に、'+' あるいは '-' の文字がくるといことです。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。

有効なストリングの例：

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

無効なストリングの例：

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

Net.Data は、IF ブロックを、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。

例： 他のブロックの内部に IF ブロックを含むマクロ・ファイル

```
%{ 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)
  %}
%}
```

制約事項: Net.Data は、非整数の数字値の比較をサポートしません。

ループ構成体

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 ブロックを、IF ブロックを処理するのと全く同じ方法で処理します。しかし、ループを 1 回完了するごとに、条件リストを再評価します。そして、どの条件付きループ構成要素の場合も同じですが、条件のコード化に誤りがある場合は、処理は無限ループに陥ることがあります。

例: WHILE ブロックを含むマクロ・ファイル

```
%DEFINE loopCounter = "1"
```

```

%HTML(build_table) {
%WHILE (loopCounter <= "100") {
  %{ generate table tag and column headings %}
  %IF (loopCounter == "1")
    <TABLE BORDER>
    <TR>
    <TH>Item #
    <TH>説明
    %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 は言語環境を提供します。これによって、ユーザーはデータ・ソースをアクセスし、ビジネス論理を持つアプリケーション・プログラムを実行します。たとえば、SQL 言語環境によって、ユーザーは SQL ステートメントを DB2 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、ユーザーは REXX プログラムを起動できるようになります。ユーザーは、さらに SYSTEM 言語環境を使用して、プログラムを実行するか、あるいはコマンドを発行します。

Net.Data によって、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。ENVIRONMENT ステートメントを Net.Data 初期設定ファイルに追加して、サービス・プログラムを、ユーザーが作成する言語環境に関連付けなければなりません。Net.Data は、言語環境名を指定する FUNCTION ブロックへの関数呼び出しを最初に検出すると、サービス・プログラムのロードおよび実行を 1 回行います。同じ言語環境名を指定する FUNCTION ブロックへのこれ以降の関数呼び出しでは、単にロード済みのサービス・プログラムが Net.Data によって実行されるだけです。

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。

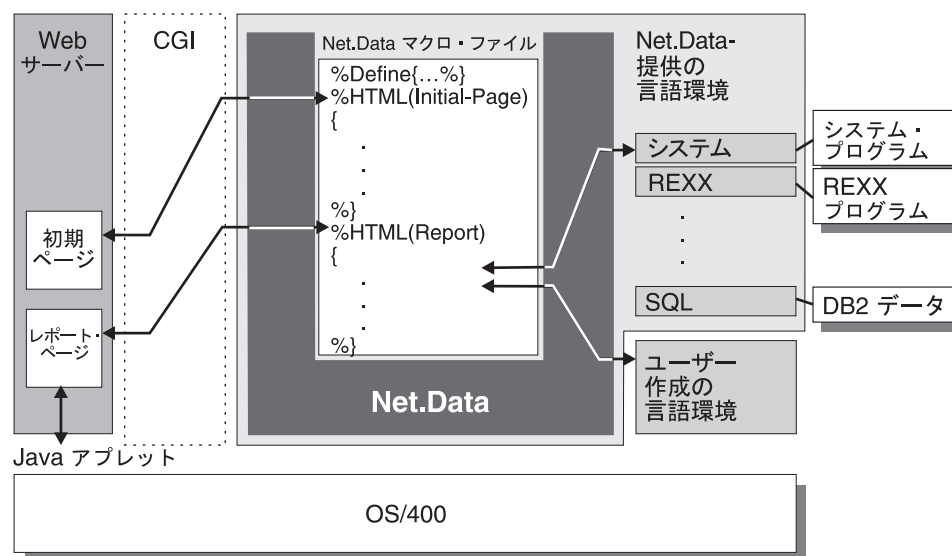


図 8. Net.Data 言語環境

ユーザーによる言語環境の作成方法に関する詳細については、*Net.Data 言語環境解説書* を参照してください。

以下の節では、言語環境の構成方法や使用方法を含め、Net.Data に組み込まれている言語環境についてそれぞれ説明します。

- 72ページの『REXX 言語環境』

- 75ページの『SQL 言語環境』
- 86ページの『システム言語環境』

REXX 言語環境

REXX 言語環境では、内部 REXX プログラム (Net.Data マクロの FUNCTION ブロックで指定します) を解釈したり、別個のファイルに保管された外部 REXX プログラムを実行したりすることができます。

REXX 言語環境を構成する

以下のステップを使用して、REXX 言語環境を構成します。 Net.Data 初期設定ファイルを作成しないように選択した場合は、REXX 言語環境がデフォルトで使用可能になります。この場合、追加の構成を行う必要はありません。

1. 初期設定ファイルを作成しており、かつ REXX 言語環境を使用したい場合は、以下の構成ステートメントを初期設定ファイルに追加する。

```
ENVIRONMENT(DTW_REXX)/QSYS.LIB/QTCP.LIB/QTMHREXX.SRVPGM ( )
```

環境構成ステートメントなどの Net.Data の初期設定ファイルに関する詳細については、5ページの『第2章 Net.Data の構成』を参照してください。

2. 外部 REXX プログラムを QSYS.LIB ファイル・システムに入れる。

外部 REXX プログラムを呼び出す

外部 REXX プログラムを呼び出すには、FUNCTION ブロックを以下の形式で使します。

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

たとえば、以下のようにします。

```
%FUNCTION(DTW_REXX) rexx1() {
%EXEC{ /QSYS.LIB/REXX.LIB/REXXSRC.FILE/TREXX.MBR %}
%}
```

パラメーターを渡す

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、パラメーターを外部 REXX プログラムに直接渡します。たとえば、以下のようにします。

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

Net.Data 変数 INPARM1 への参照が解決され、外部 REXX プログラムに渡されます。REXX プログラムは、REXX PARSE ARG 命令を使用してこの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た

パラメーターは、入力タイプのパラメーターと見なされます (プログラムに渡されたパラメーターは、プログラムで使用および操作することができますが、これらのパラメーターに対する変更は Net.Data 側には反映されません)。

間接

パラメーターを、REXX プログラムの変数プール を介して間接的に渡します。REXX プログラムが開始されると、すべての変数に関する情報が入ったスペースが作成され、REXX インタープリターによって保守されます。このスペースを変数プールと呼びます。

REXX 言語環境 (DTW_REXX) の関数を呼び出すと、入力 (IN) または入出力 (INOUT) のすべての関数パラメーターが REXX 言語環境によって変数プールに保管されてから REXX プログラムが実行されます。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完了した時点で、DTW_REXX 言語環境は、出力 (OUT) または INOUT 関数パラメーター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取り出し、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、以下のようになります。

```
%define a = "3"
%define b = "0"
%function(DTW_REXX) double_func(IN inpl, OUT outpl){
    outpl = 2*inpl
}%

%HTML(REPORT){
Value of b is $(b), @double_func(a, b) Value of b is $(b)
%}
```

上の例では、@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 倍にして、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、行数は myTable_COLS です。

REXX の SAY 命令を OS/400 V3R2 または V3R7 で使用する

OS/400 V3R2 または V3R7 を実行しており、かつ REXX プログラムが REXX の SAY 命令を使用して stdout に書き込む場合は、ストリングの先頭に 12 個の空白を挿入します。

たとえば、以下のようになります。

```
SAY '          STARTOFDATA'
```

この 12 個のブランクは無視されますが、挿入しないと予期しない結果が起こる場合があります。

機密保護

Net.Data を実行するユーザー ID には、REXX プログラムが使用するすべてのオブジェクトに加えて、外部 REXX プログラムを含むファイルにも読み取りアクセス権および書き込みアクセス権を与えてください。

パフォーマンスを向上させる

Net.Data アプリケーションのパフォーマンスを向上させるには、以下のヒントを使用してください。

- REXX プログラムを結合させることによって、呼び出す REXX プログラムの数を制限する。大きなプログラムが少しだけある方が、小さなプログラムが多数ある場合よりもパフォーマンスが向上します。これは、REXX インタープリターが常に新しい活動化グループで実行されるためです。インタープリターは、Net.Data がインタープリターを呼び出して REXX プログラムを実行するたびにアクティブになります。
- REXX プログラムを、Net.Data マクロにインラインとして含めるのではなく、外部ファイルに保管する。
- 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、入力専用パラメーターを REXX プログラムに直接渡す。インライン REXX プログラムについては、REXX ソースでグローバル変数を直接参照する。

REXX 言語環境の例

以下の例に、REXX 関数を呼び出して 2 列 3 行の Net.Data 表を生成するマクロを示します。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]

Web のマクロをライブラリー NETDATA、ファイル REXXMAC、およびメンバー REXX1 に保管するとすると、このマクロは、以下の URL をブラウザからロードすることによって参照します。

```
http://hostname/cgi-bin/db2www/qsys.lib/netdata.lib/rexxmac.file/  
rex1.mbr/report
```

SQL 言語環境

DB2 を使用して SQL ステートメントを実行するには、SQL 言語環境を使用します。有効な DB2 for OS/400 コマンドである限り、任意の有効な SQL ステートメントを SQL 言語環境に渡すことができます。

SQL 言語環境を構成する

SQL 言語環境を構成するには、以下のステップを使用します。

1. SQL 言語環境がアクセスするすべてのリモート・データベースに加えて、リレーショナル・データベース登録簿にあるローカル・データベースの登録簿項目（つまり、リモート・ロケーションが *LOCAL の登録簿項目）を作成する。項目は、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追加します。

Net.Data 初期設定ファイルを作成しないように選択した場合は、SQL 言語環境がデフォルトで使用可能になります。この場合、追加の構成は不要です。

2. Net.Data 初期設定ファイルを変更する。
 - a. 初期設定ファイルを作成しており、かつ SQL 言語環境を使用したい場合は、初期設定ファイルに以下の構成ステートメントを追加する。この環境ステートメントのテキストは、初期設定ファイル内ではすべて 1 行に納めなければなりません。ここでは、読みやすくするために数行に分けてあります。

```
ENVIRONMENT(DTW_SQL) /QSYS.LIB/QTCP.LIB/QTMSQL.SRVPGM ( )  
(IN DATABASE, LOGIN, PASSWORD, TRANSACTION_SCOPE, SHOWSQL,  
DTW_SET_TOTAL_ROWS, DB_CASE, START_ROW_NUM, RPT_MAX_ROWS,  
OUT DTWTABLE, SQL_CODE, TOTAL_ROWS)
```

環境ステートメントには、上記で指定された変数を必ずしもすべて指定する必要はありません。DTW_SET_TOTAL_ROWS や TOTAL_ROWS などの変数を使用しない場合は、環境ステートメントから削除することができます。これらの変数を SQL 言語環境に渡す別の方法として、マクロの Net.Data 関数呼び出しで変数を渡す方法があります。言語環境変数に関する詳細については、*Net.Data 解説書* を参照してください。

- b. 構成変数を追加または更新する。SQL 言語環境は、以下の構成変数をサポートしています。これらの変数は、76 ページの表 1 に示すように Net.Data の初期設定ファイルで指定することができます。

表 1. SQL 言語環境の構成変数

構成変数	説明
DTW_SQL_ISOLATION	SQL 言語環境によって実行されるデータベースの操作が、同時に実行されているプロセスからどの程度分離されるかを指定します。指定可能な値は以下のとおりです。 DTW_SQL_NO_COMMIT DTW_SQL_READ_UNCOMMITTED DTW_SQL_READ_COMMITTED DTW_SQL_REPEATABLE_READ DTW_SQL_SERIALIZABLE デフォルトは DTW_SQL_READ_UNCOMMITTED です。
DTW_SQL_NAMING_MODE	SQL ステートメントにおける表名の指定方法を指定します。指定可能な値は以下のとおりです。 SQL_NAMING SYS_NAMING デフォルトは SQL_NAMING です。

環境構成ステートメントや構成変数ステートメントなどの Net.Data 初期設定ファイルに関する詳細については、5ページの『第2章 Net.Data の構成』を参照してください。

SQL 言語環境をコミットメント制御のもとで実行する

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。

集合内の SQL 表にアクセスしている場合は、AS/400 のネイティブ SQL サポートを介して自動的にジャーナル処理されるため、明示的なアクションは不要です。

- 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に関する詳細については、75ページの『SQL 言語環境を構成する』を参照してください。

リモート・データベースのデータベース接続を管理する

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 HTTP サーバー・プロセス・ジョブの存続期間については接続をアクティブに保ちます。接続がアクティブに保たれるので、データベースへの初期接続後のデータベースへのアクセスが高速になります。以下の事項を考慮に入れると、エラーを回避してパフォーマンスを向上させることができます。

- データベース接続は、トランザクションの効力範囲の設定に基づいて計画する。

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。以下の規則に基づいて接続を計画してください。

- 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る。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 を変更してはならない。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。
- パフォーマンスを向上させるために、同一のリモート・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、リモート・システムへの接続を確立すると、ユーザー ID をその接続に関連付けます。以後の Net.Data 照会では、このユーザー ID が接続に関連付けられたユーザー ID と一致しないと、この接続は終了し、データベースへの新しい接続が確立されます (これが起こるのは、トランザクションの有効範囲が SINGLE の場合だけです)。

パフォーマンスを向上させるには、SQL ステートメントをリモート・データベースに発行する場合に、ユーザー ID をハードコーディングするか、同じユーザー ID を使用します。ローカル・データベースへのアクセスの場合は、ユーザー ID およびパスワードは無視されます。

ストアード・プロシージャ

ストアード・プロシージャは、コンパイル済みのプログラムであり、DB2 のローカル・サーバーまたはリモート・サーバーに保管され、SQL ステートメントを実行することができます。Net.Data では、ストアード・プロシージャは、CALL SQL ステートメントを使用して、Net.Data の関数から呼び出されます。ストアード・プロシージャのパラメーターは、Net.Data の関数パラメーター・リストから渡されます。ストアード・プロシージャを使用すると、コンパイル済みの SQL ステートメントをデータベース・サーバーと一緒に保管することにより、パフォーマンスと保全性を向上させることができます。

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。

- 78ページの『ストアード・プロシージャの構文』
- 78ページの『ストアード・プロシージャの呼び出し』
- 79ページの『パラメーターを渡す』
- 80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、および任意選択で REPORT ブロックを使用します。

```
%function (dtw_sql) function_name ([IN datatype arg1, INOUT datatype arg2,  
    OUT tablename, ...])  
    CALL stored_procedure  
[%REPORT(resultsetname...)]
```

ここで、

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 関数の名前

stored_procedure

ストアード・プロシージャの名前

datatype

表2 で示すような、Net.Data がサポートするデータベースのデータ型。パラメーター・リスト内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、データベースの文書を参照してください。

tablename

結果セットが保管されるべき Net.Data の表の名前 (結果セットを Net.Data 表に保管するときだけ使用する)。これを指定する場合には、このパラメーター名は、resultsetname に関連するパラメーター名と一致しなければなりません。

resultsetname

レポート・ブロックをもつストアード・プロシージャから戻される結果セットに関連する名前。これを指定する場合には、このパラメーター名は、tablename に関連するパラメーター名と一致しなければなりません。

表2. ストアード・プロシージャのデータ型

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

ストアード・プロシージャの呼び出し

ストアード・プロシージャを呼び出すには、以下のようにします。

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する。

```
%function (dtw_sql) function_name()
```

2. 任意選択で、ストアード・プロシージャから戻される任意の結果セットの結果セット名を含むストアード・プロシージャに対して、IN、INOUT、または OUT パラメーターを指定する。

```
%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・プロシージャ名を確認します。

```
CALL stored_procedure
```

4. ストアド・プロシージャが 1 つの結果セットを生成中である場合には、任意選択で、Net.Data が結果セットを表示する方法を定義する REPORT ブロックを指定します。

```
%report {  
...  
%}
```

例 :

```
%function (dtw_sql) mystoredproc (IN CHAR(30)  
arg1 OUT mytable) {  
    CALL myproc  
    %report {  
        ...  
        %row { ... %}  
        ...  
    %}  
%}
```

5. ストアド・プロシージャが 1 つ以上の結果セットを生成中である場合には、以下のようにします。

- 結果セットを OUT パラメーターとして FUNCTION ステートメントで指定する。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する。

```
%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

パラメーターをストアド・プロシージャに渡し、ストアド・プロシージャにパラメーター値を更新させて、新規値を Net.Data のマクロに戻すことができるようにします。IN キーワードでパラメーター名を指定して、ストアド・プロシージャ内にパラメーターを渡します。ストアド・プロシージャがパラメーターの更新中である場合には、INOUT または OUT キーワードで戻される値にパラメーターを渡さなければなりません。パラメーターに対して指定されるデータ型は、ストアド・プロシージャが予測する型と一致しなければなりません。

例 1 : ストアド・プロシージャにパラメーター値を渡す

```
%function (dtw_sql) mystoredproc (IN CHAR(30) valuein) {
  CALL myproc
...
}
```

例 2: ストアード・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

1 つ以上の値を、ストアード・プロシージャから戻すことができます。ユーザーのマクロでさらに処理を続けるために、Net.Data 表に結果セットを保管、あるいは、REPORT ブロックを使用して表示することができます。名前を、ストアード・プロシージャが生成する結果セットと関連付けなければなりません。これを行うには、FUNCTION ステートメントでパラメーターを指定します。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。ユーザーは以下を行うことができます。

- 結果セットを、結果セットのレポート・ブロックを定義しないで、Net.Data のデフォルトのレポート・スタイルで表示する。
- 結果セットを REPORT ブロックに関連付け、Net.Data がレポートの結果セットを表示するようにする。次に、Net.Data 変数、HTML や JavaScript のようなテキスト処理ステートメント、または他の関数を使用して、レポート・データをブラウザで表示する方法を指定することができます。

結果セットは必ずローカル表に保管されるため、別の関数がマクロ・ファイル内で後でこのデータを使用することができます。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のガイドラインおよび制約事項については、65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』を参照してください。

単一の結果セットを戻し、それをデフォルトのレポート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 tablename) {
  CALL stored_procedure
%}
```

たとえば、以下のようにします。

```
%function (dtw_sql) mystoredproc(OUT mytable1) {
  CALL myproc
%}
```

単一の結果セットを戻し、表示処理に対して REPORT ブロックを指定するには、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 tablename) {
    CALL stored_procedure
    %REPORT (resultsetname) {
        ...
    %}
%}
```

たとえば、以下のようにします。

```
%function (dtw_sql) mystoredproc (OUT mytable1) {
    CALL myproc
    %REPORT (mytable1) {
        ...
        %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、以下のようにします。

```
%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
%}
```

ここにはレポート・ブロックを指定しません。

たとえば、以下のようにします。

```
%define DTW_DEFAULT_REPORT = "MULTIPLE"
%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。

各結果セットを、固有の **REPORT** ブロックと関連付けます。以下の構文を使用します。

```
%function (dtw_sql) function_name (OUT tablename1, tablename2) {
  CALL stored_procedure
  %REPORT (resultsetname1)
  ...
  %row { ... %}
  ...
%}
%REPORT (resultsetname2)
...
%row { ... %}
...
%}
%}
```

たとえば、以下のようにします。

```
%function (dtw_sql) mystoredproc (OUT mytable1, mytable2) {
  CALL myproc

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

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

SQL 言語環境の制約事項

環境を計画するには、以下の制約事項を考慮してください。

- 以下の場合には SQL 言語環境を使用してはなりません。
 - データベース・アクセス・クラス・ライブラリーまたは SQL コール・レベル・インターフェースを使用するユーザー定義の言語環境を作成する。
 - さらに、ユーザー定義の言語環境をマクロで参照する。
- EXEC ステートメントで SQL 言語環境に SQL ステートメントを渡してはなりません。

機密保護

Net.Data が実行されるユーザー ID に、HTTP サーバーが常駐するマシンにあるデータベース (データベース・ファイルやジャーナルも含む) へのアクセス権を与えてください (コミットメント制御がアクティブな場合)。

リモート・データベースにアクセスしている場合は、Net.Data 変数の LOGIN (ユーザー ID) および PASSWORD を使用して、アクセス可能なデータベース・リソースが判別されます。これらの変数は、HTTP サーバーが常駐するマシンにあるデータベースにアクセスする場合には無視されます。これは、オブジェクトへのアクセスが、Net.Data が実行されているユーザー・プロファイルによって判別されるためです。

パフォーマンスを向上させる

DB2 のパフォーマンスに関する考慮事項の詳細については、 *DB2 for AS/400 SQL Programming Guide* を参照してください。この資料には、SQL 索引の効果的な使用、結合照会のパフォーマンスの改善、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改善など、多くの情報が記載されています。以下の節では、データベースおよび SQL 言語環境固有の技法について要約します。

データベースの技法

以下の要約では、データベース・アクセスを向上させることができる最も簡単なデータベース技法の概要について説明します。

- 数値変換を避ける。列値とリテラル値を比較する場合には、なるべく同じデータ型と属性を指定するようにします。リテラル値の精度が列の精度より高い場合には、DB2 AS/400 用は指定された列の索引を使用しません。比較する 2 つの項目のデータ型が異なる場合は、DB2 AS/400 用は、それらの値の一方を変換しなければなりません。この場合、マシンの精度に限度があるため、値が不正確になる場合があります。

たとえば、EDUCLVL がハーフワードの整数値 (SMALLINT) であるとしします。この場合は、以下のように指定します。

```
... WHERE EDUCLVL < 11 AND EDUCLVL >= 2
```

以下のようにはしません。

```
... WHERE EDUCLVL < 1.1E1 AND EDUCLVL > 1.3
```

- 文字ストリングの埋め込みを行わない。固定長の文字ストリングの列値とリテラル値を比較する場合は、なるべく同じデータ長を使用するようにします。リテラル値が列の長さより長い場合には、DB2 AS/400 用は索引を使用しません。

たとえば、EMPNO が CHAR(6) で、DEPTNO が CHAR(3) であるとしします。この場合は、以下のように指定します。

```
... WHERE EMPNO > '000300' AND DEPTNO < 'E20'
```

以下のようにはしません。

```
... WHERE EMPNO > '000300 ' AND DEPTNO < 'E20 '
```

- % または _ で始まる LIKE パターンを使用しない。パーセント記号 (%) および下線 () を LIKE の述部のパターンに使用すると、選択したい行の列値に似た文字ストリングが指定されます。文字ストリングの途中や末尾の文字を示すために使用する場合は、LIKE パターンは索引を利用することができます。たとえば、以下のようにします。

```
... WHERE LASTNAME LIKE 'J%SON%'
```

しかし、文字ストリングの先頭に使用した場合は、スキャンされる行数を制限する LASTNAME 列上に定義されるはずのいずれの索引についても、LIKE パターンは、DB2 AS/400 用がそれらを使用するのを妨げる場合があります。例を示します。

```
... WHERE LASTNAME LIKE '%SON'
```

大きな表にアクセスする場合には、特にこれらのシンボルを文字ストリングの先頭に使用しないようにしてください。

SQL 言語環境の技法

パフォーマンスを向上させるには、以下の SQL 言語環境を使用します。

- データベースに接続するユーザー ID の数を減らして、データベースへの再接続を避ける。SQL 言語環境は、ユーザー・プロファイルおよびパスワードを、この環境が確立するすべてのデータベースへのリモート接続に関連付けます。LOGIN および PASSWORD 変数がオープン済みの接続に関連付けられたユーザー・プロファイルおよびパスワードに一致しない場合は、この接続はクローズしてから再確立され、再度オープンされた接続に LOGIN および PASSWORD 変数が関連付けられます。
- START_ROW_NUM および RPT_MAX_ROWS の各 Net.Data 変数を使用して、戻される表のサイズを削減する。SELECT SQL ステートメントにおいて、結果セットに非常に多くのレコードが含まれる場合は、START_ROW_NUM を (スクロール可能なカーソルのように) 使用するとともに RPT_MAX_ROWS を使用して結果セットのサブセットをブラウザーに戻し、戻されるレコードの数を制限します。Net.Data が状態を認識できないので毎回照会を再発行することに注意してください。ただし、永続的マクロに Net.Data サポートを使用して、トランザクションの存続期間にわたって存続する Net.Data 表に結果セットを保管することができます。永続的 Net.Data マクロに関する詳細については、89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 SQL を使用するストアード・プロシージャを呼び出すように考慮する。動的 SQL は実行時に準備されますが、静的 SQL はプリコンパイル段階で準備されます。SQL 言語環境は、動的 SQL を使用して、プログラムの実行時に SQL ステートメントを実行することができます。ステートメントの準備には処理時間が余計に必要なため、静的 SQL の方が効率が高くなる場合があります。

OS/400 V4R2 で開始する場合は、SQL エンジンに準備済みのステートメント・キャッシュがあるので注意してください。このキャッシュを使用すると、SQL エンジンに準備済みの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。その後、同じステートメントを再び実行する際には、異なるユーザーおよび異なるジョブによる場合であっても、このステートメントは以前より高速に実行されます。システム全体にわたる準備済みステートメント・キャッシュは通常の SQL 処理の一部であり、キャッシュを構成したり使用可能にしたりするためのユーザーのアクションは不要です。このキャッシュによって、動的 SQL に対する静的 SQL のパフォーマンス上の優位性が低下する場合があります。

SQL 言語環境の例

以下の例に、SQL ストアード・プロシージャを呼び出す DTW_SQL 関数定義でのマクロを示します。この例には、データ型の異なる 3 つのパラメーターがあります。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 BLOCK  
/*****%}  
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>
%}

```

Web のマクロをライブラリー NETDATA、ファイル SQLMAC、およびメンバー SQL1 に保管するとすると、このマクロは、以下の URL をブラウザからロードすることによって参照します。

```
http://hostname/cgi-bin/db2www/qsys.lib/netdata.lib/sqlmac.file/  
sql1.mbr/report
```

システム言語環境

システム言語環境は、RPG、COBOL、C などの外部プログラムの呼び出しをはじめ、FUNCTION ブロックの EXEC ステートメントで識別される CL コマンドの実行をサポートしています。システム言語環境は、指定されたプログラム名かコマンド、およびパラメーターをオペレーティング・システムに渡し、C 言語の system() 関数呼び出しを使用して実行することによって、EXEC ステートメントを解釈します。

システム言語環境を構成する

Net.Data 初期設定ファイルを作成しないように選択した場合は、システム言語環境がデフォルトで使用可能になります。この場合、追加の構成は不要です。

初期設定ファイルを作成しており、かつシステム言語環境を使用したい場合は、以下の構成ステートメントを初期設定ファイルに追加します。

```
ENVIRONMENT(DTW_SYSTEM) /QSYS.LIB/QTCP.LIB/QTMSYS.SRVPGM ( )
```

Net.Data の初期設定ファイルおよび環境構成ステートメントに関する詳細については、5ページの『第2章 Net.Data の構成』を参照してください。

パラメーターを渡す

システム (DTW_SYSTEM) 言語環境によって呼び出される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 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、以下のようになります。

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

Net.Data 変数 INPARAM1 への参照が解決され、プログラムに渡されます。このパラメーターは、「コマンド入力 (Command Entry)」画面からプログラムを呼び出すときにパラメーターがプログラムに渡される方法と同じ方法でプログラムに渡されます。この方法を使用してプログラムに渡されたパラメーターは、入力タイプのパラメーターと見なされます (プログラムに渡されたパラメーターは、このプログラムで使用および操作することができますが、このパラメーターに対する変更は Net.Data 側には反映されません)。

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" の形式の文字ストリングであり、プログラムの外部の環境スペースに保管されます。このストリングは、プロセスに関連する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完了すると、DTW_SYSTEM 言語環境は、出力 (OUT または INOUT) 関数パラメーター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環境スペースから取り出し、この新しい値で関数パラメーターの値を更新します。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3に記載する API を使用します。

表 3. 環境変数の API

ILE プログラミング言語	取得に使用するもの	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 以降の場合は、環境変数の設定に CHGENVVAR コマンドおよび ADDENVVAR の各 CL コマンドを使用することもできます。
2. QtmhGetEnv() は、IBM TCP/IP 通信ユーティリティ/400 の一部として出荷されています。
3. QtmhPutEnv() は、もともと IBM TCP/IP 通信ユーティリティ/400 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、後になって最終段階で追加されたものであり、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。この表の行数は myTable_ROWS であり、列数は myTable_COLS です。

行が多い表は渡さないようにしてください。これは、プロセス用の環境変数の数に制限があるためです。

機密保護

Net.Data を実行するユーザー ID には、プログラム (プログラムがアクセスする可能性があるすべてのオブジェクトを含む) を実行するためのアクセス権を与えてください。

パフォーマンスを向上させる

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。

システム言語環境の例

以下の例に、P1、P2、および P3 の 3 つのパラメーターを持つ関数定義を含むマクロを示します。P1 は入力 (IN) パラメーターであり、P2 と P3 は出力 (OUT) パラメーターです。この関数はプログラム UPDPGM を呼び出して、パラメーター P2 を P1 の値で更新し、P3 を文字ストリングに設定します。 %EXEC ブロック内のステートメントを処理する前に、DTW_SYSTEM 言語環境は、P1 およびその対応する値を環境スペースに保管します。

```
%DEFINE {
    MYPARM2 = "ValueOfParm2"
    MYPARM3 = "ValueOfParm3"
}%
%FUNCTION(DTW_SYSTEM) sys1 (IN P1, OUT P2, P3) {
    %EXEC {
        /QSYS.LIB/NETDATA.LIB/UPDPGM.PGM
    }
}%

%HTML(upd1) {
    <P>
    Passing data to a program. The current value
    of MYPARM2 is "$(MYPARM2)", and the current value of MYPARM3 is
    "$(MYPARM3)". Now we invoke the Web macro function.

    @sys1("ValueOfParm1", MYPARM2, MYPARM3)

    <P>
    After the function call, the value of MYPARM2 is "$(MYPARM2)",
    and the value of MYPARM3 is "$(MYPARM3)".
}%
```

Web のマクロをライブラリー NETDATA、ファイル SYSMAC、およびメンバー SYS1 に保管するとすると、このマクロは、以下の URL をブラウザからロードすることによって参照します。

```
http://hostname/cgi-bin/db2www/qsys.lib/netdata.lib/sysmac.file/
sys1.mbr/upd1
```

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 つのマクロの複数のブロックあるいは複数のマクロを単一の論理トランザクションの一部として実行できるようになります。

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戻されると、データベースがコミットされ、資源が解放されて、すべて初期状態に設定されます。次に同じマクロを起動すると、フォーム・データとして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ておいたり、行った変更を明示的に取り消さずにデータベース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扱ったりする機能はありません。

永続的マクロの場合は、アプリケーション開発者と同様に、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。重要なのは、トランザクションの途中でユーザーが作業を取り消した場合に、アプリケーションがすべての作業をロールバックできることです。

永続的マクロの起動に関する詳細については、28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。

- 『永続的マクロの概要』
- 90ページの『トランザクションの定義』
- 97ページの『永続的マクロの例』

永続的マクロの概要

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 サーバーが Net.Data のプロセスを終了する必要はありません。代わりに、プロセスはアクティブのままになり、Web ブラウザーを介したユーザーからの応答を待ちます。プロセスは終了しないため、Net.Data はマクロの状態情報を保守し、トランザクションをオープンしたままにすることができます。

Net.Data は、サーバーに新しい 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。新しいヘッダー『Accept-HTSession』のサポートは、AS/400 HTTP Server のバージョン 4 リリース 3 (V4R3) に追加されています。HTTP ヘッダーは出力より先に送信しなければならないため、Net.Data は、最初の出力を送信するときに、サーバーに送信する HTTP ヘッダーを決定します。つまり、永続的マクロの開発時には、以下の点に注意しなければなりません。

- 最初に出力が生成されるときに、Net.Data は、マクロ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。
- 新しい永続的マクロの組み込み関数を使用して、出力の生成前に、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 サーバーの間の接続は永続的ですが、ブラウザと Web サーバーの間の接続では接続が保守されません。
- トランザクションを長時間実行できる。単一の Net.Data プロセスを複数のブラウザ要求にわたって実行することができるため、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 プロセスは長時間アクティブのままになる可能性があるため、より多くのシステム・リソースを消費する可能性がある。これらの資源の管理には注意しなければなりません。
- Web サーバーに永続性のサポートを含めなければならないため、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、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。

- 91ページの『トランザクションの開始』
- 92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』
- 95ページの『トランザクションの終了』
- 95ページの『トランザクションでの変数の効力範囲の定義』
- 96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、出力がブラウザーに送信される前に、マクロ内でマクロが永続的であることを `Net.Data` に示します。 `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 サポートが必要なことを通知します。

トランザクションを開始する

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、現行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ー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には、60 秒のタイムアウト値が指定されています。60 秒以内にブラウザーから応答を受信しなかった場合、Web サーバーはトランザクションを終了します。この処理は、ブラウザー上の現行のページには影響しません。しかし、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 ブロックを呼び出す 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際には、以下の 3 つのステップを行います。

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、ハンドル名を Net.Data および Web サーバーに渡す。
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. トランザクション・ハンドルの変数を DEFINE セクションで定義する。たとえば、以下のようにします。

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

2. 任意選択で、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・ハンドルを生成させる。

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。一方、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ます。これによって、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することができなくなります。

Net.Data に対するトランザクション・ハンドルを指定する

Net.Data に対するトランザクション・ハンドルの値を DTW_ACCEPT() 組み込み関数で指定します。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。任意の HTML ブロックの外部でこの関数を呼び出した場合、トランザクション・ハンドルおよび任意指定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 ブロック内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() 呼び出しそのものの間に空白文字を置かないでください。Net.Data は空白文字をテキストと見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ので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・ハンドルを生成させて 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 しかトランザクションで実行することはできません。トランザクション・ハンドルは、URL では CGI プログラム名の直後になければなりません。

ご注意: 以下の URL は、それぞれ実際のコードでは 1 行の連続したストリングとして指定しなければなりません。ただし、ここでは、読みやすくするため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/` です。

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 を指定する。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する。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。詳細については、12 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・ファイルの中の 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: 別のマクロへの 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ことを 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。以下を指定することができます。

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。

変数に対して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こと、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ことを指定します。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こと、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ことを指定します。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ます。しかし、トランザクションは多くのマクロ呼び出しにわたって実行することができるため、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こととすると、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たら、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/pcgi1.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指示する。
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ことを指示します。
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たら、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/pcgi1.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、このマクロが永続的であることを指示する。
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。

付録A. 問題分析

本節では、Net.Data のデバッグ時に使用できる問題分析の技法について説明します。

以下の問題では、Net.Data CGI-BIN プログラム・オブジェクト DB2WWW が、CGI-BIN プログラムが常駐する WWWCGI という名前のライブラリー名に移動されていると想定しています。

- 徴候: エラー 500:

```
Bad script request -- '/QSYS.LIB/WWWCGI.LIB/DB2WWW.PGM/  
QSYS.LIB' not executable
```

原因: Exec 規則が誤っています。

```
Exec /QSYS.LIB/WWWCGI.LIB/DB2WWW.PGM/*  
Exec /qsys.lib/wwwcgi.lib/db2www.pgm/*
```

解決法: DB2WWW プログラムにパスだけを提供するように Exec 規則を指定します。たとえば、以下のようにします。

```
Exec /QSYS.LIB/WWWCGI.LIB/*  
Exec /qsys.lib/wwwcgi.lib/*
```

- 徴候: エラー 404:

```
Not found - file doesn't exist or is read protected  
Even tried multi"
```

原因: Exec 規則がありません。

解決法: DB2WWW プログラムに対するパスを提供する Exec 規則を大文字と小文字の両方で指定します。たとえば、以下のようにします。

```
Exec /QSYS.LIB/WWWCGI.LIB/*  
Exec /qsys.lib/wwwcgi.lib/*
```

- 徴候: エラー 403:

```
Forbidden - by rule
```

原因: Map または Exec 規則がないか、誤っています。

解決法: DB2WWW プログラムの Map および Exec 規則を大文字と小文字の両方で指定します。たとえば、以下のようにします。

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

- 徴候: Web サーバーの構成ファイルにある構成ステートメントは正しいが、Net.Data がマクロ・ファイルを正常に処理しない。この場合は、いくつかの原因と解決法が考えられます。

- **原因:** Map、Exec、または Pass 規則が正しい順序になっていません。

解決法: Map、Exec、または Pass 規則について URL を評価する際には、最初に一致した規則に基づいて処理されます。評価対象のステートメントが所要の規則に到達する前に、それらの再マップまたは変更が行われていないことを確認してください。また、構成ファイルで Pass /* ステートメントを指定していないことも確認してください。

- **原因:** ユーザー・プロファイル QTMHHTTP1 に、 Net.Data マクロにアクセスするための適切な権限がありません。

解決法: すべての CGI-BIN プログラムを、ユーザー・プロファイル QTMHHTTP1 のもとで実行してください。 Net.Data マクロの処理中に Net.Data がアクセスするすべてのオブジェクトに対する権限を QTMHHTTP1 ユーザー・プロファイルに与えなければなりません。

- **原因:** Net.Data 初期設定ファイルの Path ステートメントが誤っています。

解決法: オブジェクトの参照が完全修飾されているか、 Net.Data 初期設定ファイルに適切なパス・ステートメントがあることを確認してください。初期設定ファイルにパス・ステートメントがある場合は、 Net.Data はこれを使用して、処理中の Net.Data マクロにある Net.Data マクロまたは実行可能プログラムの参照に解決します。オブジェクトの参照が完全修飾されていない場合および初期設定ファイルのパス・ステートメントが誤っている場合は、参照されているオブジェクトが検出されないことが Net.Data によって示されます。

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、リストで社員の名前を選択して個々の社員の追加情報を得られるアプリケーションから、社員名のリストを表示します。このマクロは、SQL 言語環境を使用して、社員名および特定の社員に関する情報の EMPLOYEE 表を照会します。

```

%{***** Sample Macro *****)
*   FileName = sqlsamp1.d2w
*   Description:
*       This Net.Data macro file queries...
*       - The EMPLOYEE table to create a selection list of
*         employees for display at a browser
*       - The EMPLOYEE table to obtain additional information
*         about an individual employee
*
*****%}
%{*****
*   Include for global DEFINES -
*****%}
%INCLUDE "sqlsamp1.hti"
%{*****
*   Function: queryDB           Language Environment: SQL
*   Description: Queries the table designated by the variable myTable and
*   creates a selection list from the result. The value of the variable
*   myTable is specified in the include file sqlsamp1.hti.
*****%}
%FUNCTION(DTW_SQL) queryDB() {
    SELECT * FROM $(myTable)
%MESSAGE {
    -204: {<p><b>ERROR -204: Table $(myTable) not found. </b>
        <p>Be sure the correct include file is being used.</b>
        %} : exit
    +default: "WARNING $(RETURN_CODE)" : continue
    -default: "Unexpected ERROR $(RETURN_CODE)" : exit
%}

%REPORT {
<select name=emp_name>
%ROW{
<option>$(V2)
%}
</select>
%}
%}

%{*****
*   Function: fname           Language Environment: SQL
*   Description: Queries the table designated by the variable myTable for
*   additional information about the employee identified by the
*   variable emp_name.
*****%}
%FUNCTION(DTW_SQL) fname(){
SELECT EMPNME, PHONENO, JOB FROM $(myTable) WHERE EMPNME='$(emp_name)'
%MESSAGE {
    -204: "Error -204: Table not found "
    -104: "Error -104: Syntax error"
    100: "Warning 100: No records" : continue
    +default: "Warning $(RETURN_CODE)" : continue
    -default: "Unexpected SQL error" : exit
%}
%}

```



```
%{ *****
* HTML block: INPUT Title: Dynamic Query Selection *
*
* Description: Queries the EMPLOYEE table to create a selection list of
* the employees for display at the browser *
*****%}
%HTML(INPUT){
<html>
<head>
<title>Generate Employee Selection List</title>
</head>
<body>
<h3>$(exampleTitle)</h3>
<p>This example queries a table and uses the result to create
a selection list using a <em>%REPORT</em> block.
< hr>
<form method="post" action="report">
@queryDB(<input type="submit" value="Select Employee">
</form>
< hr>
</body>
</html>
%}
```

```
%{ *****
* HTML block: REPORT *
* Description: Queries the EMPLOYEE table to obtain additional information *
* about an individual employee *
*****%}
%HTML(REPORT){
<html>
<head>
<title>Obtain Employee Information</title>
</head>
<body>
<h3>You selected employee name = $(emp_name)</h3>
<p>Here is the information for that employee:
<PRE>
@fname()
</PRE>
<hr><a href="input">Return to previous page</a>
</body>
</html>
%}
```

```
%{ End of Net.Data macro 1 %}
=====
%{ ***** Include File *****
* FileName = sqlsamp1.hti *
* Description: *
* This include file provides global DEFINES for the sqlsamp1.d2w *
* Net.Data macro. *
*****%}
%define {
emp_name = ""
reposition = sign
exampleTitle = "Sample Macro"
myTable = "MRZ.EMPLOYEE"
%}

%{ End of include file %}
```


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連する動作の評価および検査はお客様の責任で行っていただきます。

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

〒106-0032 東京都港区六本木3丁目2-31

AP事業所

IBM World Trade Asia Corporation

Intellectual Property Law & Licensing

IBM は、この Web サイトよりアクセスできるその他の Web サイトに関していかなる保証もしません。お客様が IBM 以外の Web サイトにアクセスされた場合、これらの Web サイトは、IBM から独立して運営されており、IBM は、当該 Web サイトの内容に関していかなる責任も負わないことをご了承ください。さらに、IBM 以外の Web サイトにリンクがはられていることにより IBM が推奨するものでもなく、当該 Web サイトの内容もしくは使用について責任を負うものではありません。

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」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 の今後の方針や意図に関する記述は、通知なしに変更されたり撤回されたりすることがあります。それらは単に目標を示しているにすぎません。

本書には、日常の業務処理で用いられるデータの報告書の例が含まれていますが、それらの名前はすべて架空のものであり、また名称や住所が類似する企業が実在しても、それは偶然に過ぎません。

商標

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

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

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

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

UNIX は、X/Open Company Ltd. により独占的にライセンスを受けている米国ならびにその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、米国ならびにその他の国における Microsoft Corporation の商標あるいは登録商標です。

2 個のアスタリスク (**) で示されている他社名、製品名およびサービス名は、他社の商標またはサービス・マークです。

用語集

絶対パス (absolute path). オブジェクトのフルパス名。絶対パス名は、最高レベル (つまり「ルート」ディレクトリー) で始まる。このレベルは、通常のスラッシュ (/) または円記号 (¥) 文字によって識別される。

API. アプリケーション・プログラミング・インターフェース (Application programming interface)。Net.Data は、3 つの所有権を主張できる API をサポートし、CGI 処理のパフォーマンスを向上させる。

アプレット (applet). HTML ページに組み込まれる Java プログラム。アプレットは、Netscape などの Java 対応のブラウザを処理し、HTML ページのロードの際にロードされる。

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

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

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内の境界の確立。リソースの操作は作業単位の一部となる。

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

現行作業ディレクトリー (current working directory). すべての相対パス名が解決される、プロセスのデフォルト・ディレクトリー。

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

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

データ型 (data type). 列およびリテラルの属性。

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

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

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

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

HTTP. Hypertext transfer protocol (HTTP)。

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

hypertext transfer protocol (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)。

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

ICSS. Internet Connection Secure Server。

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

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

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

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

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

言語環境 (language environment). Net.Data マクロから、DB2 などの外部データ・ソースや Perl などのプログラミング言語へのアクセスを行うモジュール。言語環境によっては、REXX、Perl、および Oracle などの Net.Data と一緒に提供されるものもある。また、独自の言語環境を作成することもできる。

LOB. ラージ・オブジェクト (Large object)。

ミドルウェア (middleware). アプリケーション・プログラムとネットワークの中間にあるソフトウェア。ネットワークを介したクライアント・アプリケーション・プログラムとサーバー間の相互作用を管理します。

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レジストリー。

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まると想定する。

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]

アクセス権、Net.Data のファイルへの指定 17
暗号化、ネットワーク 21
永続的マクロ・ファイル 89

[カ行]

開始、Net.Data の 25
隠し変数
 資産の保護 23
 変数名を隠す 43
各種変数 46
型、変数の 41
環境変数 41
関数
 ストアド・プロシーチャーの呼び出し 77
 説明 48
 定義 48
 ユーザー定義 48
 呼び出し 53
 FUNCTION ブロックの構文 48
 MACRO_FUNCTION ブロックの構文 49
関数呼び出し
 構文 53
 処理順序 54
起動、Net.Data の
 概説 25
 構文 25
 直接要求 25
 フォーム 25, 28, 93
 マクロ要求 25
 マクロ・ファイルでの 25
 リンク 25, 28, 93
 CGI の使用 25
 HTML ブロック 59
 URL 25, 29, 93
機密保護
 アクセス権の指定 17
 概説 19
 許可 23
 認証 21
 ネットワーク暗号化 21
 ファイアウォール 19
 Net.Data のメカニズム 23

許可
 機密保護 23
 Net.Data のファイルへのアクセス権の指定 17
グローバルな識別子の効力範囲 37
結果セット
 簡単な 80
 ストアド・プロシーチャーの処理 80
 複数の 81
 複数の、ガイドラインおよび制約事項 65
言語環境 71
 構成、初期設定ファイル内の 15
 変数 48
 例 15
 ENVIRONMENT ステートメントの構成 15
構成、Net.Data の
 概説 5
 初期設定ファイル
 更新 7
 構成変数ステートメント 9
 作成 7
 説明 7
 パス・ステートメント 11
 ENVIRONMENT ステートメント 15
 Net.Data のファイルへのアクセス権 17
構成変数ステートメント
 構成、初期設定ファイル内の 9
 説明 9
 DTWR_CLOSE_REGISTRIES 11
 DTW_SQL_ISOLATION 9
 DTW_SQL_NAMING_MODE 10
効力範囲
 変数 37
 REPORT ブロック 38
効力範囲、識別子の
 グローバルな 37
 マクロ・ファイル 38
 FUNCTION ブロック 38
 ROW ブロック 38
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

[サ行]

作成、初期設定ファイルの 8
参照、変数の 40
サンプル・マクロ 103
実行可能な変数 42

処理、ストアード・プロシージャでの結果セットの
80

初期設定ファイル

更新 7

構成変数ステートメント 9

作成 7, 8

説明 7

パス・ステートメント 11

フォーマット 8

ENVIRONMENT ステートメント 15

条件付き

変数 41

論理、IF ブロック 66

ストアード・プロシージャ

簡単な結果セット 80

結果セットの処理 80

ステップ 78

デフォルトのレポート 80, 81

パラメーターの引き渡し 79

複数の結果セット 81

マクロ・ファイルからの呼び出し 77

有効なデータ型 78

REPORT ブロック 80, 81

宣言パーツ、マクロ・ファイル構造の 31

[タ行]

データ型、ストアード・プロシージャに対して有効な
78

定義、変数の

DEFINE ステートメントまたはブロック 39

HTML のフォームの SELECT および INPUT タグ
39

URL データ 40

デフォルトのレポート

ストアード・プロシージャに対する指定 80, 81

プリント 62

トークンのサイズ 37

特記事項 107

トランザクション処理 89

[ナ行]

ナビゲーション、マクロ内およびマクロ間の 36

認証、機密保護 21

[ハ行]

パーツ、マクロ・ファイルの

宣言 31

表示 31

パス・ステートメント

更新のガイドライン 12

パス・ステートメント (続き)

構成、初期設定ファイル内の 11

資産の保護 23

EXEC_PATH 13

FFI_PATH 15

INCLUDE_PATH 14

MACRO_PATH 12

引き渡し、ストアード・プロシージャへのパラメータ
ーの 79

表処理変数 46

表変数 45

ファイアウォール 19

ファイル、Net.Data へのアクセス権の指定 17

フォーマット、データ出力の 61

フォーム

Net.Data の起動 25, 28, 93

Net.Data を起動するための Web ページ内の 27

フッター情報、REPORT ブロックの 61

プリントの使用禁止、デフォルトのレポートの 62

ブロック、マクロ・ファイルの 33

ヘッダー情報、REPORT ブロックの 61

変数

隠し 43

各種 46

型 37, 41

環境 41

言語環境 48

効力範囲 37

参照 40

実行可能 42

条件付き 41

ステートメントの構成

初期設定ファイル 9

説明 9

SQL の名前モード (DTW_SQL_NAMING_MODE)
10

SQL の分離 (DTW_SQL_ISOLATION) 9

Web レジストリーのクローズ

(DTWR_CLOSE_REGISTRIES) 11

説明 37

定義 39

トークンのサイズ 37

表 45

表の処理 46

リスト 44

レポート 47

変数参照の処理順序 54

保護、資産の 19

[マ行]

マクロ要求

構文 25

マクロ要求 (続き)

説明 25

例 25

マクロ・ファイル

永続的 89

開発 31

関数 48

サンプル 32

識別子の効力範囲 38

条件付き論理 66

説明 1

宣言パーツ 31

内部およびその間のナビゲーション 36

表示パーツ 31

ブロック 33

分析 32

変数 37

ループ 68

DEFINE ブロック 34

FUNCTION ブロック 34

HTML の生成 59

HTML ブロック 35

IF ブロック 66

WHILE ブロック 68

[ヤ行]

ユーザー定義関数 48

用語集 108

呼び出し、関数の 53

呼び出し、ストアード・プロシーチャーの 77, 78

[ラ行]

リスト変数 44

リンク

Net.Data の起動 25, 28, 93

Net.Data を起動するための Web ページ内の 27

ループ、WHILE ブロックの 68

レポートのフォーマットのカスタマイズ 62

レポート変数 47

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

D

DEFINE ブロック

説明 34

変数の定義 39

DTWR_CLOSE_REGISTRIES 11

DTW_SQL_ISOLATION 9

DTW_SQL_NAMING_MODE 10

E

ENVIRONMENT ステートメント

言語環境のタイプ 16

構成、初期設定ファイル内の 15, 16

構文 16

サービス・プログラム 16

説明 15

パラメーター・リスト 16

例 16

F

FUNCTION ブロック

関数の呼び出し 53

識別子の効力範囲 38

出力のフォーマット 61

処理、関数呼び出しの 54

説明 34

変数参照の処理 54

H

HTML

認識されていないデータ 60

表のタグ 62

フォーム

説明 27

Net.Data の起動 25, 28, 93

SELECT および INPUT タグ、変数定義 39

ブロック

起動、Net.Data の 59

処理 60

説明 35

例 59

マクロ・ファイルでの生成 59

リンク

説明 27

Net.Data の起動 25, 28, 93

FORM の処理依頼ボタン 60

I

IF ブロック 66

IN パラメーター 54

INOUT パラメーター 54

M

MACRO_FUNCTION ブロック

- 関数の呼び出し 53
- 構文 49
- 処理、関数呼び出しの 54
- 変数参照の処理 54

MESSAGE ブロック

- 構文 52
- 効力範囲 52
- 処理 52
- 説明 52
- 例 52

N

Net.Data

- 概説 1
- 起動 25
- 機密保護のメカニズム 23
- 構成 5
- ファイル、アクセス権 17
- マクロ・ファイル、開発 31

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。参照: マクロ・ファイル 1

O

OUT パラメーター 54

R

REPORT ブロック

- ガイドライン 65
- 効力範囲 38
- ストアード・プロシージャー 80, 81
- 制約事項 65
- 説明 61
- データ出力のフォーマット 61
- ヘッダーおよびフッター情報 61

RETURNS パラメーター 55

RETURN_CODE 変数 52, 54

ROW ブロック、識別子の効力範囲 38

S

SQL

- 名前付けモード構成変数 10
- 分離構成変数 9

U

URL

- 変数の定義 40

URL (続き)

- Net.Data の起動 25, 29, 93

W

Web レジストリー・クローズ用変数 11

WHILE ブロック 68



Printed in Japan

SB88-7395-00



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