

IBM[®] Net.Data



管理およびプログラミングの手引き OS/2 版[®]、Windows NT 版[®]、UNIX 版[®]

バージョン 7

IBM[®] Net.Data



管理およびプログラミングの手引き OS/2 版[®]、Windows NT 版[®]、UNIX 版[®]

バージョン 7

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、287ページの『特記事項』に記載する一般情報をお読みください。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

<http://www.ibm.com/jp/manuals/main/mail.html>

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

<http://www.ibm.com/jp/manuals/> の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

原 典： VDB2-NETA-01
IBM® Net.Data
for OS/2®, Windows NT®, and UNIX®
Administration and Programming Guide
Version 7

発 行： 日本アイ・ピー・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 2000.6

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

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

© Copyright International Business Machines Corporation 1997, 2000. All rights reserved.

Translation: © Copyright IBM Japan 2000

目次

まえがき	vii
Net.Data について	vii
バージョン 7 の新機能	viii
本書について	ix
本書の対象読者	ix
本書の例について	x

第1章 概要	1
Net.Data とは ?	1
Net.Data を使用する理由	2

第2章 Net.Data の構成	5
Net.Data の初期設定ファイルについて	7
任意選択のコンポーネントの Net.Data 構成ファイルについて	7
Live Connection の構成ファイル	8
キャッシュ管理プログラムの構成ファイル	8
Net.Data の初期設定、制御、およびマクロ・ファイルの共通セクション	9
Net.Data の初期設定ファイルのカスタマイズ	13
構成変数ステートメント	14
パス構成ステートメント	23
環境構成ステートメント	29
言語環境のセットアップ	32
IMS Web 言語環境のセットアップ	33
Java 言語環境のセットアップ	34
Oracle 言語環境のセットアップ	35
Live Connection の構成	38
CGI と一緒に使用するための Web サーバーの構成	44
FastCGI のための Net.Data の構成	45
Java サブレットおよび Java Beans と一緒に使用するための Net.Data の構成	49
Web サーバー API と一緒に使用するための Net.Data の構成	49
Net.Data の管理ツールを用いた Net.Data の構成	53
事前準備	53
管理ツールの開始	54
パス・ステートメントの構成	54
ポートの構成	56

クライアントの構成	57
言語環境の構成	63
構成変数の定義	68
Net.Data がアクセスするファイルへのアクセス権の授与	69

第3章 ユーザー資産を保護する	71
ファイアウォールを使用する	71
ネットワーク上のユーザーのデータを暗号化する	74
認証を使用する	74
許可を使用する	75
Net.Data のメカニズムを使用する	75
Net.Data 構成変数	75
マクロ開発技法	77

第4章 Net.Data を起動する	83
起動要求の型	83
マクロで Net.Data を呼び出す (マクロ要求)	85
マクロを使用しない Net.Data の起動 (直接要求)	90
Web サーバー API による Net.Data の起動	96
Java サブレットおよび JavaBeans とともに Net.Data を起動する	99
Net.Data サブレット	99
Net.Data JavaBeans	106

第5章 Net.Data のマクロ開発	111
Net.Data マクロの分析	112
DEFINE ブロック	114
FUNCTION ブロック	115
HTML ブロック	116
XML ブロック	118
Net.Data のマクロ変数	123
識別子の効力範囲	124
変数の定義	125
変数の参照	128
変数の型	129
Net.Data の関数	139
関数の定義	139

関数の呼び出し	145	キャッシュ管理プログラムの開始と停止	240
Net.Data 組み込み関数の呼び出し	146	Web ページのキャッシング	241
ドキュメント・マークアップの生成	151	CACHEADM コマンド	245
HTML ブロックおよび XML ブロック	151	キャッシュ・ログ	248
レポート・ブロック	153	エラー・ログ・レベルの設定	251
マクロにおける条件付き論理とループ	160	言語環境の最適化	251
条件付き論理: IF ブロック	160	REXX 言語環境	251
ループ構成体: WHILE ブロック	163	SQL 言語環境	252
		システムおよび Perl 言語環境	254
第6章 言語環境の使用	165		
Net.Data 提供の言語環境の概説	166	第8章 Net.Data のログ記録	255
言語環境の呼び出し	168	Net.Data エラー・メッセージのログを収集す る	255
エラー条件の処理	168	Net.Data エラー・ログの計画の作成	256
機密保護	168	Net.Data ログ収集レベルの制御	257
データ言語環境	169	ログ収集されない Net.Data エラー・メッ セージのタイプ	257
リレーショナル・データベース言語環境	169	Net.Data エラー・ログ・ファイルのサイズ とローテーション	257
フラット・ファイル・インターフェース言 語環境	189	Net.Data エラー・ログ・フォーマット	257
Web レジストリー言語環境	190	Live Connection のクライエントとエラー・メ ッセージのログ記録	258
プログラミング言語環境	193	Live Connection ログの計画の作成	259
Java アプレット言語環境	193	Live Connection ログ収集レベルの制御	259
Java アプリケーション言語環境	201	ログ収集されない Live Connection メッセー ジのタイプ	260
Perl 言語環境	205	Live Connection のログ・ファイル名	260
REXX 言語環境	208	Live Connection のログ・ファイル・サイズお よびローテーション	261
System 言語環境	212	Live Connection のログ・フォーマット	261
第7章 パフォーマンスを向上させる	217	付録A. 参考文献	265
Web サーバー API の使い方	217	Net.Data テクニカル・ライブラリー	265
FastCGI の使い方	218		
接続管理	218	付録B. Net.Data for AIX	267
Live Connection について	219	言語環境の共用ライブラリーをロードする	267
Live Connection の利点	220	REXX 環境のパフォーマンスの向上	268
Live Connection を使用するかどうかの判 断	221	NLS 考慮事項	268
接続管理プログラムの開始	221		
Net.Data および Live Connection のプロセ スの流れ	222	付録C. Net.Data ウィザード	271
Net.Data のキャッシング	223	事前準備	272
Web ページのキャッシングについて	224	ウィザードの実行	272
Net.Data のキャッシングについて	225		
Net.Data のキャッシュの制約事項	228	付録D. Net.Data SQL Assist を使用して SQL ステートメントを作成する	275
Net.Data のキャッシング・インターフェー ス	228	事前準備	275
キャッシュ管理プログラムのための計画	229	Net.Data SQL Assist を実行する	276
キャッシュ管理プログラムおよび Net.Data のキャッシュの構成	231		

付録E. Net.Data サブレットと NetObjects Fusion NOF プラグインを使用 する	277
NetObjects Fusion プラグインについて . . .	277
NetObjects Fusion プラグインのインストール	278
NetObjects Fusion 用 Net.Data プラグインの セットアップ	278
プラグイン・プロパティを変更する . . .	279
NOF プラグイン付きサブレットの表示	282
付録F. Net.Data サンプル・マクロ. . . .	283

特記事項	287
商標	290
用語集	293
索引	297
IBM と連絡をとる	305
製品情報	305

まえがき

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

Net.Data について

IBM の Net.Data プロダクトを用いると、DB2、IMS、および ODBC 対応のデータベース 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) の両方のデータ、ならびに Java、JavaScript、Perl、C、C++、あるいは REXX などのプログラミング言語で作成されたアプリケーションを使用して、動的な Web ページを作成することができます。

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

要求は URL (uniform resource locator) の形式で Netscape Navigator あるいは Internet Explorer などのブラウザから Web サーバーに流れ、Web サーバーはその要求を実行するために Net.Data に送ります。Net.Data はマクロを見つけて実行し、ユーザーが作成した関数に基づいてマクロがカスタマイズする Web ページを生成します。これらの関数は以下のことを行うことができます。

- ビジネス・ロジックを Perl スクリプト、C および C++ アプリケーション、あるいは REXX プログラム 内にカプセル化する。
- DB2 などのデータベースにアクセスする
- フラット・ファイルなど、他のデータ・ソースにアクセスする。

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

Net.Data は、HTTP (HyperText Transfer Protocol) および共通ゲートウェイ・インターフェース (CGI) などのインターフェースを使用するように構成されているサーバー環境で使用できます。HTTP は、ブラウザと Web サーバー間の対話のための業界標準のインターフェースです。また CGI は、Net.Data のようなゲートウェイ・アプリケーションを Web サーバーで呼び出すための業界標準のインターフェースです。これらのインターフェースでは、Net.Data を使用するのに自分の気に入ったブラウザあるいは Web サーバーを選択することができます。また Net.Data は、パフォーマンス向上のために、さまざまなアプリケーション・プログラミング・インターフェース (API) をサポートしています。Net.Data ファミリー・プロダクトは、OS/400、OS/390、Windows NT、AIX、OS/2、HP-UX、Sun Solaris、Linux、および Dynix/PTX の各オペレーティング・システムでも同様の機能を提供します。また Net.Data は複数のオペレーティング・システムで、FastCGI および主要な Web サーバーのアプリケーション・プログラミング・インターフェース (API) をサポートしています。

グラフィカル管理ツールは、AIX、Windows NT、および OS/2 のオペレーティング・システム用の、ユーザーの Net.Data 構成設定に役立ちます。管理ツールは、Live Connection を使用するデータベースへの接続の機密保護を指定することも支援します。

データベースのデータへのアクセスを容易にするために、Net.Data は、NetObjects Fusion のプラグイン、あるいは Java ベースの開発のためのウィザードなど、さまざまなツールを提供しています。これらのツールは Java 環境では Net.Data Java サブレットとともに機能し、これによってユーザーは、オペレーティング環境間での可搬性のあるアプリケーションを作成することができます。NetObjects Fusion プラグインを使用すると、NetObjects Fusion Web 開発ツールを使用して、関係データ・ソースの動的データで高度なアプリケーションを作成することができます。Net.Data のウィザードは、基本の Net.Data マクロを作成する際の手引きとなるグラフィカルなツールを提供します。

バージョン 7 の新機能

Net.Data バージョン 7 は、前リリースの Net.Data の全機能に加え、さらに多くの機能を提供しています。Net.Data OS/2 版、Windows NT 版、および UNIX 版バージョン 7 では、以下の機能が追加されています。

- Net.Data を使用して、ブラウザがサポートする XML をそのまま表示します。データは、新規の XML レポート・ブロックを使用して、XML として自動的に表示されます。Net.Data には、XSL スタイル・シートの例がいく

つか付属しています (ndTable.xml、ndObject.xml、ndRecord.xml)。これらは、Web サーバーのルート・ディレクトリにあります。

- クライアント・ブラウザを使用して、ファイルをサーバーにアップロードできます。
- Net.Data は、ストアード・プロシージャを使用して、テキスト・ファイルを DB2 にインポートできます。
- 新規の Net.Data 組み込み関数、DTW_REPLACE()、DTW_HEXTOCHAR()、DTW_CHARTOHEX()、および DTWF_READFILE() が追加されました。
- DTW_USE_DB2_PREPARE_CACHE を使用することにより、DB2 でのパフォーマンスが向上しました。DTW_USE_DB2_PREPARE_CACHE は、DB2 により作成されたステートメントとパッケージ・キャッシュを利用します。
- DTW_REMOVE_WS を使用した不要な空白の削除を改良しました。
- DTW_DEFAULT_ERROR_MESSAGE により包括的なエラー・メッセージを提供します。

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

本書では、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.ibm.com/software/data/net.data>

本書の対象読者

本書は、Net.Data アプリケーションを設計し、開発する方々を対象としています。本書で説明する概念を理解するには、Web サーバーの働きについての知

識、簡単な SQL ステートメントの理解、および HTML のフォーム・タグをはじめとする HTML のタグの知識が必要です。

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

本書の例について

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

第1章 概要

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

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

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

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

Net.Data とは ?

Net.Data マクロを使用することにより、プログラミング論理の実行、変数のアクセスおよび操作、関数の呼び出し、およびレポート作成ツールの使用が可能です。マクロとは Net.Data マクロ言語構成要素、HTML タグ、Javascript、および SQL や Perl などの言語環境ステートメントが組み込まれているテキスト・ファイルです。Net.Data はマクロを処理して、Web ブラウザーで表示可能な出力を作成することができます。マクロは、単純な HTML と、Web サーバー・プログラムの動的な機能性を結合します。この結果、live データを静的 Web ページに簡単に追加することができるようになります。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、Web サーバー、およびサポートされるデータとプログラミング言語環境の関係を示しています。

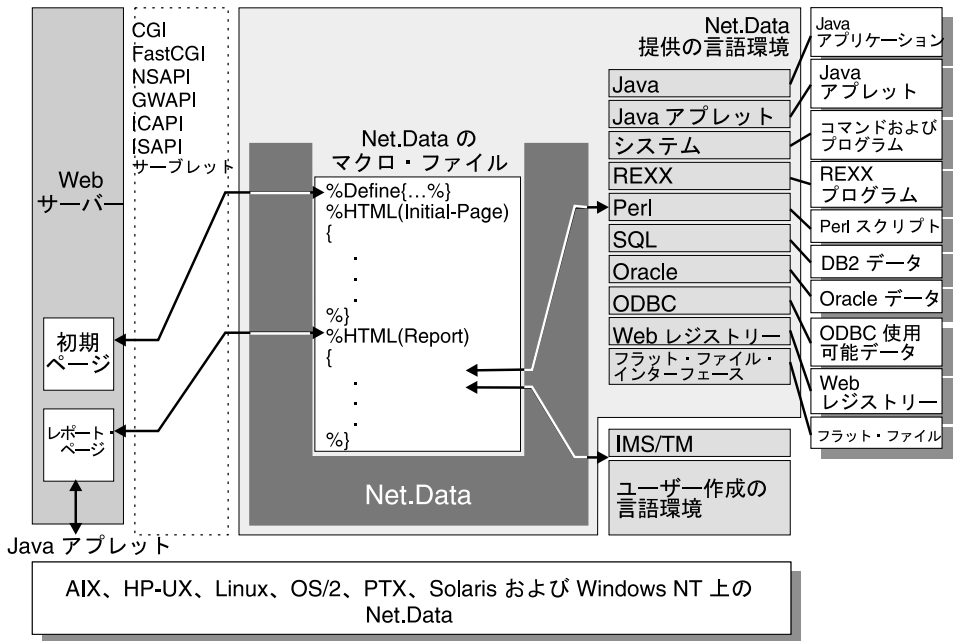


図 1. Net.Data、Web サーバ、およびサポートされるデータとプログラム・ソース間の関係

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

Net.Data を使用する理由

Net.Data を利用すれば、動的な Web ページを非常に簡単に作成できます。マクロ言語を使用すると、ユーザーが独自に Web サーバ・アプリケーションをプログラムするよりも簡単です。Net.Data では、ユーザーが現在知っている HTML、SQL、Perl、REXX、および JavaScript などの言語を使用することができます。Net.Data はまた、DB2 データベースをアクセスし、IMS Web を使用した IMS トランザクションを実行する言語環境を提供します。この言語

環境で、ユーザー・アプリケーションには、REXX、Perl、他各種言語を使用することができます。また、マクロの変更は、ブラウザで即時に表示することができます。

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

- インターネットおよびイントラネット・アプリケーションを迅速に開発することができる、簡単で強力なマクロ言語を提供します。Net.Data Web アプリケーション環境には、以下の機能があります。
- Web アプリケーション内におけるデータ生成論理と表示論理を分割します。Net.Data は、データの表示方法 (HTML または Javascript など) に制限を課しません。このような分割により、ユーザーは最新の表示方法を使用したデータ表示の変更が容易に行えます。
- 既存のスキルおよびビジネス・ロジックを使用した Web ページの作成が可能です。C、C++、REXX、Java またはその他の言語で作成されたプログラムによりインターフェースをとることができます。
- 単純なマクロ言語を使用して、複雑なインターネット・アプリケーションの開発が素早く行えます。
- DB2、および DRDA 使用可能なりモート・データベースに保管されたデータに対するパフォーマンスの高いアクセスを提供します。
- Net.Data ファミリー製品がサポートするすべてのオペレーティング・システム間でのマクロの移行が容易に行えます。

インタープリター・マクロ言語

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

直接要求

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

フリー・フォーマット

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

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、123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

使用しているオペレーティング・システムに対応した Net.Data を、製品に付属する README ファイルの説明に従ってインストールします。オペレーティング・システムによって異なりますが、大部分の構成ステップはインストール時に完了します。

ユーザーのオペレーティング・システムに対応した Net.Data をインストールした後、Net.Data を構成し、Web サーバーのための構成を変更しなければなりません。構成のステップは以下のとおりです。

- Net.Data の初期設定 (INI) ファイルのカスタマイズ
- FastCGI またはサポートされている Web サーバー API の 1 つ (任意選択) と共に使用するための Net.Data の構成
- Web サーバーの構成および環境変数ファイルのカスタマイズ
- キャッシュ・マネージャーの構成 (任意選択)
- Live Connection の構成 (任意選択)
- Net.Data の言語環境の設定
- アクセス権限の指定

以下のツールを使用して、Net.Data を構成します。

- テキスト・エディター

テキスト・エディターを使用して、すべてのオペレーティング・システム上の初期設定ファイルと、Live Connection およびキャッシュ管理プログラムの構成ファイルを編集します。テキスト・エディターを使用して、任意の Web サーバーの構成ファイルも編集します。変更を加える前に、それらのファイルのバックアップを取っておくことをお勧めします。

- Net.Data の管理ツール

管理ツールは、初期設定ファイルおよび Live Connection の構成ファイルのカスタマイズのためのグラフィカル・インターフェースを提供しています。管理ツールを使用して、OS/2、Windows NT、および AIX の各オペレーティング・システム上で Net.Data を構成することができます。

使用方法は、6ページの表1 で説明されているように、構成に必要なコンポーネントおよび Net.Data が実行されるオペレーティング・システムに依存します。構成のための作業を、ある特定の方法を使用して開始した場合は、最良の結果

を得るためには、その方法を続けて使用しなければなりません。

表 1. 構成方法と、タスクおよびオペレーティング・システムとの比較。**A** - 管理ツールあるいは手動で構成することができます **M** - 手動でしか構成することができません。

タスク

オペレーティング・システム:

	AIX	NT	OS/2	HP SUN PTX
Net.Data の INI ファイルの構成	A	A	A	M
クライアント・ポートの定義	A	A	A	M
クライアントの定義	A	A	A	M
クライアントのパスワードの暗号化をオンにする	A	N/A	N/A	N/A
エラーのログ記録をオンにする	A	A	A	M
FastCGI、CGI、および API* 対応の Web サーバーの構成	M	M	M	M
キャッシュ管理プログラムのポートの定義	M	M	N/A	N/A
キャッシュ管理プログラムの構成	M	M	N/A	N/A

***ヒント**：多くの Web サーバーは、Web サーバーの構成に使用できる管理ツールを持っています。

本章では、Net.Data の構成方法、Net.Data と一緒に使用するための Web サーバーの構成方法について説明します。さらに、任意選択のコンポーネントの構成方法についても説明しています。

- 13ページの『Net.Data の初期設定ファイルのカスタマイズ』
- 32ページの『言語環境のセットアップ』
- 38ページの『Live Connection の構成』
- 44ページの『CGI と一緒に使用するための Web サーバーの構成』
- 45ページの『FastCGI のための Net.Data の構成』
- 49ページの『Java サブレットおよび Java Beans と一緒に使用するための Net.Data の構成』
- 49ページの『Web サーバー API と一緒に使用するための Net.Data の構成』
- 53ページの『Net.Data の管理ツールを用いた Net.Data の構成』
- 69ページの『Net.Data がアクセスするファイルへのアクセス権の授与』

Net.Data の初期設定ファイルについて

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

- Unicode での文字データのコード化
- スtring関数およびワード関数が MBCS に対応しているかどうか
- データベース・データにアクセスするための DB2 インスタンスの名前
- Net.Data と言語環境、データベース、接続管理、およびキャッシングとの接続および通信の方法
- エラー・ログ記録をアクティブにするかどうか

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、REXX プログラム、および Perl のスクリプト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 初期設定ファイル db2www.ini は、Web サーバーの資料ディレクトリーにあります。詳しくは、ご使用のオペレーティング・システムの README ファイルを参照してください。

許可のためのヒント: Web サーバーの実行時のユーザー ID が、必ずこのファイルの読み取り権限を持つようにしてください。詳しくは、69ページの『Net.Data がアクセスするファイルへのアクセス権の授与』を参照してください。

任意選択のコンポーネントの Net.Data 構成ファイルについて

以下のセクションでは、Net.Data の任意選択コンポーネントの構成ファイルについて説明します。

8ページの『Live Connection の構成ファイル』

8ページの『キャッシュ管理プログラムの構成ファイル』

9ページの『Net.Data の初期設定、制御、およびマクロ・ファイルの共通セクション』

Live Connection の構成ファイル

Live Connection は、Windows NT、OS/2、AIX、および Sun Solaris のオペレーティング・システム上で接続を管理し、開始オーバーヘッドを除去することによってパフォーマンスを改善します。Net.Data Live Connection の構成ファイルには、1 つまたは複数の名前付クライアントに関する情報が含まれています。クライアントは、長期にわたって実行されるプロセスであり、データベースまたは Java アプリケーションとの接続を保守し、複数のユーザーからの Net.Data マクロの起動を許可します。クライアントの開始後、Net.Data Live Connection が終了するまで存続します。複数のクライアントが、単一のデータベースに接続することができます。

構成ファイルのクライアント情報のパーツとして、クライアント名、固有のポート、およびプロセスの最少数と最大数を指定します。データベース・クライアントの場合、各クライアント記入項目ごとに、データベース名、ログイン、およびパスワードも指定することができます。

許可のためのヒント: 接続管理プログラムを開始するユーザー ID が、必ずこのファイルの読み取り権限を持つようにしてください。詳しくは、69ページの『Net.Data がアクセスするファイルへのアクセス権の授与』を参照してください。

キャッシュ管理プログラムの構成ファイル

キャッシュ管理プログラムの構成ファイルには、キャッシュ管理プログラムと各キャッシュの定義が含まれます。Net.Data キャッシングについては、223ページの『Net.Data のキャッシング』で説明しています。キャッシュ管理プログラムの構成については、231ページの『キャッシュ管理プログラムおよび Net.Data のキャッシュの構成』で説明しています。ファイルの構造は、一連のセクション、またはスタンザです。

キャッシュ管理プログラムのスタンザ

このスタンザは、キャッシュ管理プログラム自身のパラメーターを定義し、ネットワーク情報、ログ記録状況、およびトレース状況を組み込みます。このスタンザは必須で、cache-manager のラベルが付いていなければなりません。

キャッシュ定義のスタンザ

これらのスタンザは、各キャッシュのパラメーターを定義します。キャッシュ管理プログラムによって管理されるそれぞれのキャッシュごとに、構成ファイルに1つのキャッシュ定義スタンザが存在します。このセクションは、ネットワーク情報、メモリーおよびスペースの要件、

ログ記録状況、および統計状況を含んでいます。キャッシュ定義のスタンザは、キャッシュ管理プログラムが管理する各キャッシュに必要です。

キャッシュ管理プログラムの構成ファイルは、管理ツールでは管理できません。テキスト編集プログラムで更新することができます。このファイルの定義方法については、223ページの『Net.Data のキャッシング』を参照してください。

許可のためのヒント: キャッシュ管理プログラムを開始するユーザー ID が、必ずこのファイルへのアクセス権を持つようにしてください。詳しくは、69ページの『Net.Data がアクセスするファイルへのアクセス権の授与』を参照してください。

Net.Data の初期設定、制御、およびマクロ・ファイルの共通セクション

Net.Data が完全なものとして機能するには、Net.Data 初期設定ファイル、構成ファイルおよびマクロ・ファイルの特定の部分が、そのすべてのコンポーネントについて一貫していなければなりません。以下の表は、これらの各ファイルで一致しなければならないエリアをまとめたものです。

表 2. Net.Data 構成ファイルおよびマクロの整合性要件

ファイル	共通セクション	注
Net.Data INI ファイル	環境ステートメント	Live Connection を使用する言語環境では、その環境ステートメント内のデータベース・クライアント名を指定しなければなりません。
	Live Connection 構成変数	Net.Data Live Connection を使用する場合には、Live Connection ポート DTW_CM_PORT を指定します。この可変値は、Live Connection の構成ファイルの MAIN_PORT 値と一致しなければなりません。
	キャッシュ構成変数	Net.Data のキャッシングを使用する場合は、任意選択でポート番号とマシン名変数を組み込みます。これらの値(使用する場合は、キャッシュ管理プログラムの構成ファイルで使用している値と一致しなければなりません。

表 2. Net.Data 構成ファイルおよびマクロの整合性要件 (続き)

ファイル	共通セクション	注
Live Connection の構成ファイル	クライアント定義	各クライアント定義は、INI ファイルの対応する定義と一致しなければなりません。さらに、MAIN_PORT 値は、INI ファイルの DTW_CM_PORT 可変値と一致しなければなりません。
キャッシュ管理プログラムの構成ファイル	キャッシュ管理プログラムの構成変数	Net.Data のキャッシングを使用する場合は、任意選択でポート番号とマシン名変数を組み込むことができます。これらの値 (使用する場合は、INI ファイル で使用している値と一致しなければなりません。

以下の断片は、マクロ、Net.Data の初期設定ファイル、および Live Connection の構成ファイルの関係を示しています。2 つのクライアント (DTW_SQL: SAMPLE、DTW_SQL: CELDIAL) がマクロで使用されており、それらのクライアントは、SAMPLE および CELDIAL という 2 つの DB2 データベースにアクセスします。Live Connection の構成ファイルには、クライアント名と定義が含まれます。Net.Data の初期設定ファイル内の ENVIRONMENT ステートメントは、クライアント名を参照します。LOGIN 値と PASSWORD 値は、Live Connection の構成ファイルで指定されます。

11 ページの図2 は、データベースにアクセスするためにどのクライアントを使用するかを定義する @DTW_ASSIGN ステートメントを含む、マクロの断片を示しています。

```

<3*****>
<3** This is an HTML comment **>
<3** Access the SAMPLE database using **>
<3** cliette DTW_SQL:SAMPLE **>
<3*****>
@DTW_ASSIGN (DATABASE, " SAMPLE ")
@insert_customer
(customer_name, customer_street, customer_city, customer_state,
customer_country, customer_zip, customer_credit, customer_expiry)

<3*****>
<3** This is an HTML comment **>
<3** Process the CELDIAL database using **>
<3** the cliette DTW_SQL:CELDIAL **>
<3*****>
@DTW_ASSIGN (DATABASE, " CELDIAL ")
@insert_customer
(customer_name, customer_street, customer_city, customer_state,
customer_country, customer_zip, customer_credit, customer_expiry)

```

図2. Net.Data マクロの断片

DATABASE 構成変数は初期設定ファイルの ENVIRONMENT ステートメントに代入され、クライアント名が生成されることに注意してください。これにより、同じマクロから複数のデータベースにアクセスすることができます。

12ページの図3 に、ENVIRONMENT ステートメントおよび関連するクライアント・タイプを含む Net.Data の初期設定ファイルのフラグメントを示します。初期設定ファイルには、クライアントごとに 1 つの ENVIRONMENT ステートメントがあります。それぞれのデータベース・クライアント・タイプごとに、ENVIRONMENT ステートメントでクライアント名を指定しています。その名前は、クライアント型と変数参照 \$(DATABASE) で構成され、実行時に解決されます。Live Connection を使用する各言語環境の ENVIRONMENT ステートメントには、クライアント定義がなければなりません。

```
ENVIRONMENT (DTW_SQL)
(IN DATABASE, LOGIN, PASSWORD, TRANSACTION_SCOPE, SHOWSQL,
ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
CLIETTE "DTW_SQL:${DATABASE}"
```

図 3. *Net.Data* の初期設定ファイルのフラグメント

13ページの図4 は、Live Connection の構成ファイルのフラグメントを示しています。これには、DTW_SQL:CELDIAL と DTW_JAVAPPS に対するクライアント定義が含まれています。


```

CONNECTION_MANAGER{
    MAIN_PORT=7100
    ENCRYPTION=OFF
}

#####
# This is a comment in a Live Connection configuration file.
# Comments start with a pound (hash) character.
# Comments terminate at the end of the line and do not continue to
# the next line unless another pound (hash) character is specified.
# You can include comments at the end of lines containing Live
# Connection keywords except on password lines.
# You cannot include comments anywhere on lines containing the
# password keyword.
# You cannot include spaces and pound (hash) characters within any
# name, such as cliette name or in database cliette passwords.
#####
CLIETTE DTW_SQL:CELDIAL{
    MIN_PROCESS=1
    MAX_PROCESS=5
    EXEC_NAME=./dtwcd2
    DATABASE=CELDIAL
    LOGIN=marshall
    PASSWORD=stlpwd
}
CLIETTE DTW_JAVAPPS{
    MIN_PROCESS=1
    MAX_PROCESS=5
    EXEC_NAME=./javaapp
}

```

図 4. Live Connection の構成ファイルのフラグメント

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。

- 14ページの『構成変数ステートメント』
- 23ページの『パス構成ステートメント』
- 29ページの『環境構成ステートメント』

図5 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、OS/2 と Windows NT に有効です。

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。マクロから呼び出す言語環境の初期設定ファイルごとに、ENVIRONMENT ステートメントが含まれていることを確認してください。マクロ内でファイルへのすべての参照を完全に修飾する場合は、どのパス構成ステートメントも指定する必要がありません。

```
1 DTW_CM_PORT 7128
2 DTW_INST_DIR c:¥db2www
3 DTW_LOG_DIR c:¥db2www¥logs
4 DB2INSTANCE DB2
5 DTW_DIRECT_REQUEST NO
6 DTW_SHOWSQL NO
7 DTW_UNICODE NO
8 DTW_MBMODE NO
9 MACRO_PATH c:¥DB2WWW¥Macro
10 HTML_PATH c:¥www¥html
11 INCLUDE_PATH c:¥db2www¥Macro
12 EXEC_PATH c:¥db2www¥Macro
13 FFI_PATH c:¥pub¥ffi;pub¥ffi¥data
14 ENVIRONMENT (DTW_SQL) [DLL path] [Parameter list]
15 ENVIRONMENT (DTW_ORA) [DLL path] [Parameter list]
16 ENVIRONMENT (DTW_ODBC) [DLL path] [Parameter list]
17 ENVIRONMENT (DTW_DEFAULT) [DLL path] [Parameter list]
18 ENVIRONMENT (DTW_APPLET) [DLL path] [Parameter list]
19 ENVIRONMENT (DTW_REXX) [DLL path] [Parameter list]
20 ENVIRONMENT (DTW_PERL) [DLL path] [Parameter list]
21 ENVIRONMENT (DTW_SYSTEM) [DLL path] [Parameter list]
22 ENVIRONMENT (DTW_FILE) [DLL path] [Parameter list]
23 ENVIRONMENT (DTW_WEBREG) [DLL path] [Parameter list]
24 ENVIRONMENT (DTW_JAVAPPS) [DLL path] [Parameter list]
25 ENVIRONMENT (HWS_LE) [DLL path] [Parameter list]
```

- 行 1 ～ 8 では、構成変数を定義しています。
- 行 9 ～ 13 では、マクロの処理に必要なファイルへのパスを定義しています。
- 行 14 ～ 25 では、使用可能な環境ステートメントを定義しています。

図 5. Net.Data の初期設定ファイル. DLL のパスとパラメーターのリストの詳細な説明については、db2www.ini ファイルそのものが、29ページの『環境構成ステートメント』を参照してください。

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

構成変数ステートメント

Net.Data の構成変数ステートメントは、構成変数の値を設定します。構成変数は、さまざまな目的に使用されます。変数の中には、適切な動作、あるいは代替モードでの動作のために、言語環境が必要とするものがあります。また変数によっては、文字の符号化や、構成中の Web ページの内容を制御するものも

あります。さらに、構成変数ステートメントを使用して、アプリケーションに固有の変数を定義することもできます。

使用する構成変数は、使用している言語環境およびデータベースによって異なります。また、その他、アプリケーション固有の要因によっても異なります。

構成変数ステートメントを更新するには、以下のようにします。

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

NAME [=] value-string

等号は、オプションです。大括弧で示されます。

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

- 『キャッシュ管理プログラムの構成変数』
- 17ページの『DB2INSTANCE: DB2 のインスタンス変数』
- 18ページの『DTW_CM_PORT: Live Connection のポート番号変数』
- 18ページの『DTW_DEFAULT_ERROR_MESSAGE: 総称エラー・メッセージの指定』
- 18ページの『DTW_DIRECT_REQUEST: 直接要求変数を使用可能にする』
- 19ページの『DTW_INST_DIR: Net.Data のインストール・ディレクトリー変数』
- 19ページの『DTW_LOG_DIR と DTW_LOG_LEVEL: エラー・ログ変数』
- 19ページの『DTW_LOG_LEVEL: エラーのログ・レベル変数』
- 20ページの『DTW_MBMODE: ネイティブ言語サポート変数』
- 20ページの『DTW_REMOVE_WS: 余分な空白文字を削除するための変数』
- 21ページの『DTW_SHOWSQL: SHOWSQL 構成変数を使用可能または使用不可にする』
- 21ページの『DTW_SMTP_SERVER: 電子メール SMTP サーバー変数』
- 22ページの『DTW_UNICODE: UNICODE 変数』
- 23ページの『DTW_VARIABLE_SCOPE: 変数の効力範囲変数』

キャッシュ管理プログラムの構成変数

キャッシュ管理プログラムが Net.Data のマクロを実行しているマシンとは異なるマシン上で実行される場合は、次の 2 つのオプションの構成変数が使用されます。

- DTW_CACHE_PORT は、キャッシュ管理プログラムに接続するために Net.Data が使用するポート番号を指定する。
- DTW_CACHE_HOST は、ローカル・マシンまたはリモート・マシンの TCP/IP ホスト名を指定する。

キャッシュ管理プログラムがローカル・マシンで実行されている場合、UNIX ドメインのソケットあるいは名前付きパイプは、通信のために使用され、構成は不必要です。

キャッシュ管理プログラムは、AIX マシンおよび Windows NT マシンでのみ実行することができます。Net.Data のキャッシングについてお知りになりたい場合は、223ページの『Net.Data のキャッシング』を参照してください。

DTW_CACHE_PORT: キャッシュ管理プログラムのポート変数

キャッシュ管理プログラムが listener となっている TCP/IP ポートを指定します。このポート番号は、キャッシュ管理プログラムの構成ファイルに指定されているポート番号に一致しなければなりません。そうでないと Net.Data はキャッシュ管理プログラムと通信することができません。指定しない場合には、キャッシュ管理プログラムは、デフォルトのポート 7175 を使用します。

構文：

DTW_CACHE_PORT [=] *port_number*

パラメーター：

port_number

サービス・キャッシュ要求に応えるためにキャッシュ管理プログラムに割り当てられる一意的なポート番号。デフォルト値は、7175 です。

17ページの表3 は、マシン ID および、これらの変数のポート番号を指定するオプションについて説明しています。

表 3. キャッシュ管理プログラムの構成変数：構成オプション

デフォルトのコネクション・マネージャーの値	キャッシュ・マシンが指定されている場合 ...	キャッシュ・マシンが指定されていない場合 ...
キャッシュのポートが指定されている場合 ...	Net.Data は、指定されたポートを使用して、指定されたマシン上のキャッシュ管理プログラムに接続します。	Net.Data は、指定されたポートを使用して、ローカル・マシン上のキャッシュ管理プログラムに接続します。
キャッシュ・ポートが指定されていない場合 ...	Net.Data は、デフォルトのポート 7175 を使用して、指定されたマシン上のキャッシュ管理プログラムに接続します。	Net.Data は、デフォルトのポート 7175 を使用して、ローカル・マシン上のキャッシュ管理プログラムに接続します。

DTW_CACHE_HOST: キャッシュ管理プログラムのマシン ID 変数

キャッシュ管理プログラムが常駐するマシンを指定します。指定しない場合には、Net.Data は、正しいマシンはローカルのマシンであることを前提にします。

構文：

DTW_CACHE_HOST [=] *host_name*

パラメーター：

host_name

キャッシュ管理プログラムが実行されるローカル・マシンまたはリモート・マシンの、限定された TCP/IP ホスト名。デフォルト値はローカル・マシンのホスト名です。

DB2INSTANCE: DB2 のインスタンス変数

SQL の言語環境で使用される DB2 のインスタンスを指定します。この可変値が必要なのは、Windows NT、OS/2、および UNIX のオペレーティング・システムで実行されている DB2 に Net.Data が接続するときです。

OS/2、Windows NT、および UNIX オペレーティング・システムの DB2 では、DB2INSTANCE が環境変数として定義されている必要があります。Net.Data は、DB2INSTANCE が環境変数として定義されていないことを検出した場合、DB2 への接続を試みる前に、DB2INSTANCE 環境変数に初期設定ファイルで検出した DB2INSTANCE の値を設定します。

構文：

DB2INSTANCE [=] *instance_name*

DTW_CM_PORT: Live Connection のポート番号変数

Net.Data が Live Connection のために使用する固有のポート番号を指定します。

構文 :

DTW_CM_PORT [=] *port_number*

ただし、*port_number* は Live Connection のために使用する固有のポート番号を指定します。

DTW_DEFAULT_ERROR_MESSAGE: 総称エラー・メッセージの指定

DTW_DEFAULT_ERROR_MESSAGE 構成変数は、実稼働時のアプリケーションに対して総称エラー・メッセージを指定するのに使用します。この変数は、MESSAGE ブロックでは取り込まれないエラー状態に対する総称メッセージを提供します。

Net.Data が生成する実際のエラー・メッセージを知りたいのであれば、エラー・メッセージのログ記録を使用して、エラー・メッセージを取り込んでください。エラー・ログの使用方法については、255ページの『第8章 Net.Data のログ記録』を参照してください。

構成変数を指定しない場合、Net.Data は、それ自身が提供するエラー状態のメッセージを表示します。

構文 :

DTW_DEFAULT_ERROR_MESSAGE [=] "*message*"

例: 総称メッセージが指定されています。

DTW_DEFAULT_ERROR_MESSAGE "This site is temporarily unavailable."

DTW_DIRECT_REQUEST: 直接要求変数を使用可能にする

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

直接要求方式で Net.Data を起動することにより、ユーザーは、SQL ステートメントや Perl、REXX、または C プログラムの実行を URL 内に直接指定することができます。直接要求を使用不可にした場合、ユーザーは、マクロ要求方式を使用して Net.Data を起動しなければならず、マクロの中に定義されているか、マクロの中で呼び出される SQL ステートメントと関数だけを実行することができます。DTW_DIRECT_REQUEST を使用する場合の機密保護に関連した推奨事項については、75ページの『Net.Data のメカニズムを使用する』を参照してください。

構文 :

DTW_DIRECT_REQUEST [=] YES|NO

ここで

YES Net.Data 直接要求を使用可能にします。

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

DTW_INST_DIR: Net.Data のインストール・ディレクトリー変数

Net.Data 実行中に特定のファイルを配置します。インストール時にこの変数を設定して、Net.Data がインストールされるホーム・ディレクトリー <inst_dir> を指定します。インストール後は、この値を変更しないでください。

DTW_LOG_DIR と DTW_LOG_LEVEL: エラー・ログ変数

DTW_LOG_DIR は、エラー・ログを格納するディレクトリーを指定します。ログに記録するには、この変数と DTW_LOG_LEVEL 変数の両方を設定しておかなければなりません。

これらの変数と、Net.Data でのエラー・メッセージのログ記録の詳細については、255ページの『Net.Data エラー・メッセージのログを収集する』を参照してください。

構文 :

DTW_LOG_DIR [=] %inst_dir%path

例 : 初期設定ファイルの構成

DTW_LOG_DIR %inst_dir%mylogfiles%

DTW_LOG_LEVEL: エラーのログ・レベル変数

DTW_LOG_LEVEL は、エラー・ログに記録するエラーのレベルを指定します。ログに記録するには、この変数と DTW_LOG_DIR 変数の両方を設定しておかなければなりません。

これらの変数と、Net.Data でのエラー・メッセージのログ記録の詳細については、255ページの『Net.Data エラー・メッセージのログを収集する』を参照してください。

構文 :

DTW_LOG_LEVEL [=] off|warning|error

例 : 初期設定ファイルの構成

DTW_LOG_LEVEL error

DTW_MBMODE: ネイティブ言語サポート変数

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

この構成変数は、DTW_UNICODE 構成変数と一緒に機能します。DTW_UNICODE がデフォルト値の NO を使用している場合は、DTW_MBMODE の値が使用されます。DTW_UNICODE が NO 以外の値に設定されている場合は、その値が使用されます。表4 は、この 2 つの変数の設定によって、組み込み関数がストリングを処理する方法がどのように決定されるかを示しています。

表 4. DTW_UNICODE と DTW_MBMODE の設定の関係

DTW_UNICODE の設定	DTW_MBMODE=YES の場合	DTW_MBMODE=NO の場合
NO	SBCS と混合する MBCS をサポートする	SBCS のみをサポートする
UTF8	UTF-8 をサポートする	UTF-8 をサポートする

構文 :

DTW_MBMODE [=] NO|YES

DTW_REMOVE_WS: 余分な空白文字を削除するための変数

この変数を YES に設定すると、Net.Data は HTML 出力から余分な空白を削除します。この変数は、空白文字を圧縮することにより、Web ブラウザーに送信されるデータの量を削減し、それによってパフォーマンスを改善します。デフォルトは、NO です。

この変数は、マクロで DEFINE ステートメントを使用してオーバーライドすることができます。

ヒント: DTW_PRINT_HEADER を使用して自分独自のヘッダーを生成する場合は (DTW_PRINT_HEADER "NO")、DTW_REMOVE_WS を YES に設定しないでください。

構文 :

DTW_REMOVE_WS [=] YES|NO

DTW_SHOWSQL: SHOWSQL 構成変数を使用可能または使用不可にする
Net.Data マクロ内の SHOWSQL の設定の効果をオーバーライドします。

構文 :

DTW_SHOWSQL [=] YES|NO

ここで

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

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

表5 では、特定のマクロについて SHOWSQL 変数を使用可能にするか使用不可にするかが、 Net.Data の初期設定ファイルとマクロの設定によってどのように決定されるかを示しています。

表 5. Net.Data の初期設定ファイルとマクロの SHOWSQL に関する設定の関係

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

DTW_SMTP_SERVER: 電子メール SMTP サーバー変数

SMTP サーバーを指定して、 DTW_SENDMAIL 組み込み関数を使用する電子メール・メッセージを送信します。この変数の値は、ホスト名または IP アドレスのいずれかにすることができます。この変数が設定されていない場合、Net.Data はローカル・ホストを SMTP サーバーとして使用します。

構文 :

DTW_SMTP_SERVER [=] server_name

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

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

例 :

DTW_SMTP_SERVER us.ibm.com

DTW_UNICODE: UNICODE 変数

Net.Data が Unicode を以下でサポートするかどうかを指定します。

- マクロ
- フォーム・データ
- DB2 データベースから検索されたデータ
- Net.Data 組み込み関数が処理するストリング

Net.Data は、マクロ、フォーム・データ、および組み込み関数において UTF-8 Unicode 形式をサポートしており、その出力は常に UTF-8 になります。

Net.Data は、UTF-16 データを含むデータベースにアクセスすることができ、そのデータを UTF-8 に変換することができます。

DTW_UNICODE を UTF8 に設定すると、Net.Data は Unicode 環境で実行されます。したがって、Net.Data はページを UTF-8 で生成し、入力データはすべて UTF-8 形式である必要があります (あるいは、DB2 データベースのデータの場合は、UCS-2 が受け入れられます)。入力データは、マクロ・ファイルの内容、ブラウザから送信されたフォーム・データ、および外部データ・ソースからのその他のすべてのデータです。

DB2 Unicode データベースの要件: DTW_UNICODE 変数の設定以外に、Net.Data の実行環境では、DB2 に固有の環境変数 DB2CODEPAGE を 1208 に設定します。たとえば、Apache Web サーバーの場合は、次の行を HTTPD.CONF ファイルに追加します。

```
SetEnv DB2CODEPAGE 1208
```

CGI スクリプト、Web サーバー API、Fast-CGI プログラム、あるいはサブレットに対する環境変数を設定する方法については、ご使用の Web サーバーの資料を参照してください。

Net.Data は、Unicode 環境で実行されている場合は、英語のメッセージ・カタログを使用します。

DTW_UNICODE 構成変数は、DTW_MBMODE 構成変数と共に使用します。DTW_UNICODE 構成変数の値は、ワード組み込み関数とストリング組み込み関数を処理する場合は、DTW_MBMODE 変数の設定値を一時変更します。ただし、DTW_UNICODE が NO に設定されている場合、あるいは設定されていない場合は、DTW_MBMODE の値が使用されます。20ページの表4 は、この2つの変数の設定によって、組み込み関数がストリングを処理する方法がどのように決定されるかを示しています。

構文：

DTW_UNICODE [=] NO|UTF8

ここで

NO DTW_MBMODE 変数の値に従うことを指定します。20ページの表4は、DTW_MBMODE の値に基づいた Net.Data のサポートについて説明しています。

UTF8 UTF-8 コード・ページをサポートし、DTW_MBMODE 構成変数の値を無視することを指定します。UTF-8 は、可変バイト数の文字を表し、ASCII に対して安全です。

DTW_VARIABLE_SCOPE: 変数の効力範囲変数

Net.Data がローカル変数の効力範囲をどのように扱うかを指定します。すなわち、ローカル変数がローカルのままであるか、またはローカル変数を作成した関数ブロックの外側でそのローカル変数を使用できるかどうかを指定します。この変数は、以前のバージョンの Net.Data との下位互換性のために提供されており、OS/390 または OS/400 バージョンの Net.Data では使用できません。

構文：

DTW_VARIABLE_SCOPE = LOCAL|GLOBAL

ここで

LOCAL

ローカル変数がローカルのままとなることを指定します。この動作は、Net.Data バージョン 2.0 で導入されたもので、デフォルトです。

GLOBAL

ローカル変数を作成した関数ブロックの外側で、そのローカル変数を使用できることを指定します。これは以前のバージョンの Net.Data との下位互換性のために提供されています。LOCAL に設定することをお勧めします。

パス構成ステートメント

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

- 24ページの『DTW_UPLOAD_DIR』
- 25ページの『EXEC_PATH』
- 26ページの『FFI_PATH』

- 26ページの『HTML_PATH』
- 27ページの『INCLUDE_PATH』
- 28ページの『MACRO_PATH』

これらのパス・ステートメントは、マクロ、実行可能ファイル、テキスト・ファイル、LOB 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：

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

- パス・ステートメントで指定される各ディレクトリーは、セミコロン (;) で区切ります。
- 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。ただし、HTML_PATH は除きます。HTML_PATH は、1 つのパス・ステートメントしか持つことができません。パスの検索は、指定された順で左から右に行われます。この複数パス能力により、ユーザーのファイルを複数のディレクトリーに編成することができます。たとえば、複数の Web アプリケーションをそれぞれ、それ自身のディレクトリーに配置することができます。
- 絶対パス・ステートメントを使用することをお勧めします。

以下のセクションでは、各パス・ステートメントの目的と構文を説明し、有効なパス・ステートメントの例を提供します。例は、ユーザーのオペレーティング・システムおよび構成に依存するため、ユーザーのアプリケーションとは異なることがあります。

DTW_UPLOAD_DIR

このディレクトリー構成ステートメントは、クライアントのブラウザーがファイル・タイプを含む直接要求を送信した時にアップロードされるファイルの、サーバー上の格納場所を指定します。この変数が設定されていない場合、Net.Data はファイルをアップロード・ファイルとして受け入れません。

構文：

DTW_UPLOAD_DIR [=] path

例：

DTW_UPLOAD_DIR /usr/lpp/internet/server_root/pub/upload

EXEC_PATH

このパス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って起動される外部プログラムを 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。パス・ステートメントのディレクトリーの順序は、Net.Data がディレクトリーを検索する順序を決定します。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の EXEC ステートメントを示しています。

Net.Data の初期設定ファイル：

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

Net.Data のマクロ

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

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

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

- 指定されたパスが絶対パスである場合は、Net.Data は、指定されたパスでファイルを検索します。たとえば、以下の URL を処理依頼したとします。

```
http://myserver/cgi-bin/db2www/usr/user1/prgms/myFunction.cmd
```

Net.Data は、/u/user1/prgms/myFunction.cmd ディレクトリー・パスでファイルを検索します。

- 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 URL を処理依頼したとします。

```
http://myserver/cgi-bin/db2www/myFunction.cmd/report
```

このとき、EXEC_PATH で指定されたどのディレクトリーでもファイル myFunction.cmd が検出されなかった場合、Net.Data は現行作業ディレクトリーでファイルを検索します。

FFI_PATH

このパス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 つまたは複数のディレクトリーをディレクトリーの指定順に、識別します。

構文：

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

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

Net.Data の初期設定ファイル：

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

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

FFI_PATH ステートメントは、パス・ステートメントのディレクトリーにないファイルに対する機密保護を提供するために使用されるので、検出されない FFI ファイルに対して特別に提供されるものではありません。Net.Data 解説書の FFI 組み込み関数のセクションを参照してください。

HTML_PATH

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

インストール時に、Net.Data は、HTML_PATH パス構成変数で指定されたディレクトリーの下に、tmplobs というサブディレクトリーを作成します。Net.Data は、このディレクトリーにすべての LOB ファイルを保管します。HTML_PATH の値を変更する場合は、新規のディレクトリーの下に新規のサブディレクトリーを作成します。

構文：

```
HTML_PATH [=] path
```

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

Net.Data の初期設定ファイル :

```
HTML_PATH /db2/lobs
```

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

パフォーマンスのためのヒント : LOB を使用する場合、システムの制限を考慮してください。その理由は、システムの制限は、すぐにリソースを消費してしまうからです。詳しくは、173ページの『ラージ・オブジェクトを使用する』を参照してください。

INCLUDE_PATH

このパス構成ステートメントは、Net.Data が検索する 1 つまたは複数のディレクトリーを、ディレクトリーの指定順に識別し、Net.Data のマクロの INCLUDE ステートメントで指定されたファイルを検出します。ファイルを検出すると、Net.Data は、組み込みファイル名をパス指定に追加し、修飾された組み込みファイル名を作成します。

構文 :

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

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

Net.Data の初期設定ファイル :

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

Net.Data のマクロ

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

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

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

Net.Data の初期設定ファイル :

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

Net.Data のマクロ

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

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

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

- 指定されたパスが絶対パスである場合は、Net.Data は、指定されたパスでファイルを検索します。たとえば、以下の URL を処理依頼したとします。

```
http://myserver/cgi-bin/db2www/u/user1/includes/oeheader.inc
```

Net.Data は、/u/user1/includes/oeheader.inc ディレクトリー・パスでファイルを検索します。

- 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 URL を処理依頼したとします。

```
http://myserver/cgi-bin/db2www/my.cmd/report
```

このとき、INCLUDE_PATH で指定されたどのディレクトリーでもファイル myFunction.cmd が検出されなかった場合、Net.Data は現行作業ディレクトリーでファイルを検索します。

MACRO_PATH

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

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

構文：

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

等号 (=) は、オプションで、大括弧で示されます。

Net.Data は、パス /macro/sqlm.d2w を、MACRO_PATH 構成ステートメントのパスに、Net.Data がマクロを検出するまで、またはすべてのパスを検索するまで、左から右へ、追加していきます。Net.Data のマクロの起動に関する情報については、83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、初期設定ファイル内の MACRO_PATH ステートメントと、Net.Data を起動する関連リンクを示しています。

Net.Data の初期設定ファイル :

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

HTML リンク:

```
<a href="http://server/cgi-bin/db2www/query.d2w/input">Submit another query.</a>
```

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

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

- 指定されたパスが絶対パスである場合は、Net.Data は、指定されたパスでファイルを検索します。たとえば、以下の URL を処理依頼したとします。

http://server/cgi-bin/db2www/u/user1/macros/myfile.txt/report

Net.Data は /u/user1/macros/myfile.txt ディレクトリー・パスでファイルを検索します。

- 指定されたパスが相対パスのとき、Net.Data は、ルート・ディレクトリー (/) から開始してすべてのディレクトリーでファイルを検索します。たとえば、以下の URL を処理依頼したとします。

http://server/cgi-bin/db2www/myfile.txt/report

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

環境構成ステートメント

ENVIRONMENT ステートメントは、言語環境を構成します。言語環境とは、Net.Data の構成要素です。Net.Data は、この構成要素を使用して、DB2 データベースのような データ・ソースにアクセスしたり、あるいは、REXX のような言語で書かれたプログラムを実行します。Net.Data は、言語環境のセットと、ユーザー自身の言語環境の作成を可能にするインターフェースを提供します。これらの言語環境については 165ページの『第6章 言語環境の使用』で、言語環境インターフェースについては *Net.Data* 言語環境解説書 で説明しています。

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

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

たとえば、マクロは、DTW_SQL 関数内の SQL ステートメントを実行するデータベースの名前を指定する DATABASE 変数を定義することができます。DATABASE の値は、SQL の言語環境 (DTW_SQL) に渡さなければなりません。これにより、SQL の言語環境は、指定されたデータベースに接続することができます。変数を言語環境に渡すには、DATABASE 変数を、DTW_SQL の環境ステートメントのパラメーター・リストに追加しなければなりません。

Net.Data のサンプルの初期設定ファイルは、Net.Data の環境構成ステートメントの設定のカスタマイズについて、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, ...) [CLIETTE "cliette_name"]
```

パラメーター：

- *type*

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

- *library_name*

Net.Data が呼び出す言語環境インターフェースを含む DLL または共用ライブラリーの名前。

- AIX では、共用ライブラリーの名前に拡張子 *.o* を付けて指定します。
- HP-UX では、共用ライブラリーの名前に拡張子 *.sl* を付けて指定します。

- SUN、PTX、および LINUX では、共用ライブラリーの名前に拡張子 `.so` を付けて指定します。
- OS/2 および Windows NT では、共用ライブラリー名には、拡張子 `.dll` を付けて指定します。

- *parameter_list*

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

パラメーター・リストで変数を設定して渡すには、マクロ内で変数を定義します。

言語環境で処理される関数を実行する前に、これらのパラメーターを、ユーザーのマクロで、構成変数、あるいは変数として定義しなければなりません。次の例では、変数を ENVIRONMENT ステートメントに指定します。

```
ENVIRONMENT (DTW_SQL) C:\WINNT\System32\nddb2.dll (IN
  DATABASE, TRANSACTION_SCOPE, USERID, PASSWORD)
```

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。

- *cliette_name*

クライアントの名前。 *cliette_name* は、Java アプリケーション言語環境のクライアントを参照するか、データベースのクライアントにすることができます。 *cliette_name* パラメーターは、CLIETTE キーワードとともに使用し、そのどちらも Live Connection としか使用することができません。

CLIETTE および *cliette_name* は任意選択で、データベースおよび Java アプリケーション言語環境にのみ指定することができます。

Java アプリケーション・クライアント

このクライアント名は、Java アプリケーションの言語環境を指定します。

構文：

```
CLIETTE "DTW_JAVAPPS"
```

データベース・クライアント

このクライアント名は、データベースと関連付けられているクライアントを指定します。

構文：

```
CLIETTE "type:db_name"
```

パラメーター：

type クライエットと関連付けられているデータベース言語環境。
有効なタイプのリストについては、 67 ページを参照してください。

db_name データベースのクライエット名。この名前は、クライエットが関連付けられているデータベース、たとえば MYDBASE と同じ場合がよくあります。しかし、また別の名前にすることもできます。 *db_name* は、Oracle 言語環境を使用する場合は、任意選択です。

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

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

アプリケーション用に 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 のマクロ作成者が自分のマクロに設定あるいはオーバーライドする必要のある変数を、ENVIRONMENT ステートメントに追加します。

```
ENVIRONMENT (DTW_SQL) /net.data/lib/dtwsq1.so ( IN DATABASE, LOGIN, PASSWORD,  
TRANSACTION SCOPE, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)  
CLLETTE "DTW_SQL:MYDBASE"  
ENVIRONMENT (DTW_ORA) /net.data/lib/dtwora.so (IN DATABASE, LOGIN, PASSWORD,  
TRANSACTION SCOPE, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)  
ENVIRONMENT (DTW_ODBC) /net.data/lib/dtwodbc.so ( IN DATABASE, LOGIN, PASSWORD,  
TRANSACTION SCOPE, ALIGN, DTW_SET_TOTAL_ROWS)  
ENVIRONMENT (DTW_APPLET) /net.data/lib/dtwjava.so ( )  
ENVIRONMENT (DTW_JAVAPPS) /net.data/lib/dtwjavapps.so ( OUT RETURN_CODE ) CLLETTE "DTW_JAVAPPS"  
ENVIRONMENT (DTW_PERL) /net.data/lib/dtwperl.so ( OUT RETURN_CODE )  
ENVIRONMENT (DTW_REXX) /net.data/lib/dtwrexx.so ( OUT RETURN_CODE )  
ENVIRONMENT (DTW_SYSTEM) dtwsys.so ( OUT RETURN_CODE )  
ENVIRONMENT (HWS_LE) dtwhws.so ( OUT RETURN_CODE )
```

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

言語環境のセットアップ

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

- 33ページの『IMS Web 言語環境のセットアップ』

- 34ページの『Java 言語環境のセットアップ』
- 35ページの『Oracle 言語環境のセットアップ』

IMS Web 言語環境のセットアップ

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

1. Net.Data を実行する Web サーバーに IMS Web 実行時コンポーネントをインストールする。IMS Web 実行時コンポーネントのインストールについては、*IMS Web User's Guide* を参照してください。(英語版)
<http://www.ibm.com/software/data/ims/about/imsweb/document/>
2. トランザクション DLL を作成する。
 - a. IMS Web Studio ツールを使用したトランザクションのために、C++ コード、make ファイル (DTWproj.mak)、および Net.Data マクロ (DTWproj.d2w) ファイルを生成する。
 - b. IMS Web Studio ツールを実行するオペレーティング・システムと異なるオペレーティング・システムで Net.Data を実行する場合は、DLL ソース・ファイルをターゲット・オペレーティング・システムの IMS Web 開発マシンに移動する。たとえば、IMS Web Studio ツールを Windows NT で実行し、ターゲット・プラットフォームが AIX または OS/390 の場合は、トランザクション DLL のソースを、AIX または OS/390 を実行している IMS Web 開発マシンにそれぞれ移動します。
 - c. 生成した make ファイルを使用して、トランザクション DLL の実行可能形式を構築する。
3. トランザクション DLL ファイル (DTWproj.dll) および Net.Data のマクロ (DTWproj.d2w) を Web サーバーにコピーする。
 - a. Net.Data がマクロを検索するディレクトリーに、マクロを置く。(詳しくは、28ページの『MACRO_PATH』を参照してください。)
 - b. Web サーバーが DLL または共用ライブラリーを検索するディレクトリーに、トランザクション DLL または共用ライブラリーを置く。
4. IMS Web Studio ツールで生成したサンプル・ファイル DTWproj .htm にあるリンクを使用して、Web サーバーの HTML ツリーの HTML ファイルを変更します。その後、そのリンクを使用して Net.Data を起動し、入力 HTML フォームを表示して IMS Web 言語環境を起動します。次に、IMS Web の言語環境は、IMS トランザクション DLL を呼び出します。この DLL は、IMS Web 実行時 DLL によって検証済みのサービスを使用してトランザクションを実行し、出力を Web ブラウザーに戻します。

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

Java 言語環境のセットアップ

Java の言語環境では、マクロから関数を呼び出す前に、追加のセットアップがいくつか必要です。

1. Net.Data は直接 Java アプリケーションを開始できないので、Java アプリケーションを立ち上げるバッチ・ファイルを作成する。Net.Data はこのファイルを使用して、Java 関数を実行する Java 仮想マシンを立ち上げます。バッチ・ファイルには、必要な Java パッケージ (標準およびアプリケーション固有のパッケージ) が必ず検出できるように、java-classpath ステートメントを組み込まなければなりません。たとえば、バッチ・ファイル launchjv.bat は以下の java-classpath を含んでいます。

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

2. Java の言語環境と一緒に機能するクライアントを、Live Connection の構成ファイル dtwcm.cnf に定義する。EXEC_NAME 構成変数を使用して、クライアントおよび関連したバッチ・ファイル名に一意的なポート番号を指定してください。以下の例では、Java のクライアント名を DTW_JAVAPPS として定義し、EXEC_NAME 構成変数がバッチ・ファイルの名前である launchjv.bat に設定しています。

```
CLLETTE DTW_JAVAPPS{
MIN_PROCESS=1                <= Required: this value must be 1 because
                               the JAVAPPS cliette is multi-threaded.
MAX_PROCESS=1                <= Required: this value must be 1 because
                               the JAVAPPS cliette is multi-threaded.
EXEC_NAME=launchjv.bat      <= The name of the batch file that includes the
                               classpath statements
}
```

Net.Data 接続管理プログラムを開始すると、Net.Data は、構成ファイルに指定されている Java クライアントを開始します。開始されたクライアントは、Net.Data マクロ・アプリケーションからの Java 言語環境要求を処理するために使用可能になります。

3. Net.Data の初期設定ファイル db2www.ini 内の DTW_JAVAPPS ENVIRONMENT パス・ステートメントに、各クライアント名を追加して、そのステートメントを更新します。

4. たとえば、以下のような場合です。

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

Oracle 言語環境のセットアップ

Net.Data マクロから Oracle データベースにアクセスする場合は、以下のステップに従います。

1. 以下のように、Oracle の適切なコンポーネントがインストールされ、作動していることを確認する。
 - a. Net.Data がインストールされているマシンに SQL*Net をインストールする (まだインストールしていない場合)。詳しくは、以下の URL を参照してください。

http://www.oracle.com/products/networking/html/stnd_sqlnet.html

- b. Oracle の *tnsping* 関数が、Web サーバーと同じ機密保護許可で使用できることを検証する。これを検証するには、Web サーバーのユーザー ID とタイプを使用してログオンします。

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

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

Windows NT では、Web サーバーがシステム権限で実行されている場合、*tnsping* 関数を検証できないことがあります。その場合は、このステップをスキップしてください。

- c. Web サーバーと同じ機密保護許可で Oracle の表にアクセスできることを検証する。これを検証するには、SQL*Plus 行コマンドツールを使用して SQL SELECT ステートメントを入力し、Web サーバーの権限を持つ SQL SELECT ステートメントを使用して Oracle の表にアクセスします。たとえば、以下のような場合です。

```
SELECT * FROM tablename
```

Web サーバーが Windows NT サービスとして実行されている場合、表アクセスを検証できないことがあります。その場合は、このステップをスキップしてください。

トラブルシューティング: 上記のステップが成功しなかった場合は、先に進まないでください。いずれかのステップが失敗した場合は、Oracle の構成をチェックしてください。

2. Oracle の環境変数が Web サーバー・プロセスに正しく設定されていることを確認する。

- AIX の場合、以下の行を `/etc/environment` ファイル、または Web サーバーの実行時のユーザー ID の `.profile` ファイルに挿入します。

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

- Windows NT の場合、システムのプロパティのコントロール・パネルを使用して、以下の環境変数を追加します。

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

ヒント: 各国語サポートや 2 フェーズ・コミットなど、使用する Oracle の機能によっては、他の Oracle 環境変数に追加の行が必要な場合があります。これらの環境変数について詳しくは、Oracle の管理マニュアルを参照してください。

3. Net.Data から Oracle への接続をテストする。Net.Data マクロで、適切な値を `LOGIN` および `PASSWORD` 変数に指定します。Oracle データベースにアクセスする場合、Net.Data の `DATABASE` 変数は定義しないでください。マクロ内の接続ステートメントの例を以下に示します。

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

ローカルにある Oracle インスタンス:

ローカルにある Oracle インスタンスにだけアクセスする場合は、`remote-oracle-instance` 名をログイン・ユーザー ID の一部として指定しないでください。以下に例を示します。

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

Live Connection:

Live Connection を使用する場合は、機密保護の目的ではお勧めできませんが、`LOGIN` および `PASSWORD` を Live Connection の構成ファイルに指定することができます。たとえば、以下のような場合です。

```
CLIETTE DTW_ORA:{
MIN_PROCESS=1
MAX_PROCESS=3
EXEC_NAME=./dtwora8
DATABASE=not_used
LOGIN=userid@remote_oracle_instance_name
PASSWORD=password
}
```

ヒント: Oracle に `DATABASE` 変数は指定しないでください。

4. CGI シェル・スクリプトを実行し、Web サーバーから Oracle インスタンスにアクセスできることを確認することにより、構成をテストする。たとえば、以下のようにします。

```
#!/bin/sh
echo "content-type; text/html"
echo
echo "<html><pre>"
set
echo "</pre><p>&nbsp;</p><pre>"
tnsping oracle-instance-name
echo
```

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

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

トラブルシューティング:

検証ステップが失敗した場合は、以下の項目を検証することによって、それまでのすべてのステップが正常に終了したことを確認します。

- Oracle の構成をチェックする。
- Oracle の環境変数構文が正しいこと、および変数の欠落がないことを検証する。
- Oracle の接続をチェックし、正しいユーザー ID とパスワードを入力したことを確認する。

それでもまだ検査に合格しない場合は、IBM サービスに連絡してください。

例 :

検査のステップへのアクセスが完了した後、以下の例のように、マクロ内で関数を使用して Oracle の言語環境を呼び出すことができます。

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

Live Connection の構成

Live Connection はデータベースおよび Java アプリケーション接続を管理し、Windows NT、OS/2、AIX、および Sun Solaris のオペレーティング・システム上の Net.Data のパフォーマンスを改善します。接続管理プログラムおよびクライアント、すなわちオープン・コネクションを保守するプロセスの使用により、Live Connection は、データベースへの接続または Java 仮想マシンの開始時のオーバーヘッドを除去します。

Live Connection は、構成ファイル `dtwcm.cnf` を使用して、開始すべきクライアントを決定します。このファイルは、Live Connection とともに使用されるクライアントごとの管理情報および定義を含んでいます。Live Connection に関してさらに詳しく知りたい場合は、218ページの『接続管理』を参照してください。

39ページの図6 で示したサンプルの構成ファイルは、以下のタイプの情報を含んでいます。

- 接続管理プログラムのポート情報
- DB2 接続のための SQL のクライアント情報
- Java アプリケーションのクライアント情報

```

1 CONNECTION_MANAGER{
2   MAIN_PORT=7100
3 }
4
5 CLIETTE DTW_SQL:CELDIAL{
6   MIN_PROCESS=1
7   MAX_PROCESS=5
8   EXEC_NAME=./dtwcdb2
9   DATABASE=CELDIAL
10  LOGIN=marshall
11  PASSWORD=st1pwd
12 }
13
14 CLIETTE DTW_JAVAPPS{
15   MIN_PROCESS=1
16   MAX_PROCESS=5
17   EXEC_NAME=./javaapp
18 }

```

- 行 1 ～ 3 は、構成ファイルに必須であり、Live Connection で使用される固有のポート番号を定義します。
- 行 5 ～ 12 では、すべてのデータベース・クライアントを定義し、クライアント名、実行すべきプロセスの数、データベース名、およびクライアントの実行可能ファイルを識別します。DB2 データベースに接続のためのユーザー ID およびパスワードのような、追加情報を組み込むこともできます。
- 行 14 ～ 18 では、Java アプリケーションに対してすべてのクライアントを定義し、クライアント名、実行すべきプロセスの数、固有のポート番号、およびクライアントの実行可能ファイルを識別します。

図 6. Live Connection の構成ファイル

事前準備： Live Connection の構成をカスタマイズする前に、以下のステップの後のヒントを読んでおいてください。

Live Connection ポートの構成:

MAIN_PORT 用に選択した値が、最初に使用されるポート番号になります。Live Connection が使用するポート番号は、MAIN_PORT の設定値と各クライアントの MAX_PROCESSES の設定値を使用して計算することができます。Live Connection は、ロード時に、MAIN_PORT で指定された開始番号から MIN_PROCESSES までの番号を 1 ずつ増加させてポートに順に割り振ります。必要に応じて Live Connection は、MAX_PROCESSES になるまでポートをロードします。使用するポートの最大数は、MAX_PROCESSES の設定値の合計です。

たとえば、図6 の構成では、割り振られるポート番号は、7100、7101、および 7102 で、必要に応じて 7110 まで割り振ることができます。

重要:

- ・システム管理者に相談して、使用を計画しているポート番号を必ず使用可能にしてください。
- ・MAIN_PORT の値が、Net.Data の初期設定ファイル内の DTW_CM_PORT の値に一致していることを確認してください。

データベースのクライアントの構成

1. クライエントの環境ステートメントを入力する。

```
CLLETTE type:db_name
```

パラメーター :

type 言語環境とクライアントを関連付ける名前。有効なタイプのリストについては、67 ページを参照してください。

db_name

データベースのクライアント名。これは、クライアントが関連付けられているデータベース、たとえば MYDBASE と同じ場合があります。しかし、*db_name* は、別の名前にすることもできます。

db_name は、Oracle 言語環境を使用する場合は、任意選択です。

2. MIN_PROCESS および MAX_PROCESS の値を決定する。MIN_PROCESS は、接続管理プログラムが開始したときに、開始すべきプロセスの数を指定します。その後、同時要求の追加が必要になると、接続管理プログラムは、さらに多くのクライアントを開始し、指定された MAX_PROCESS の値に達するまで、必要に応じて 1 つずつ追加していきます。

MIN_PROCESS および MAX_PROCESS ステートメントを入力します。

```
MIN_PROCESS=min_num
MAX_PROCESS=max_num
```

パラメーター :

min_num

接続管理プログラムが開始したときに開始すべきクライアント・プロセスの数。このクライアントの数に見合うだけの使用可能な一意的なポート番号を用意しておかなければなりません。

max_num

同時に実行できるクライアントの最大数。このクライアントの数に見合うだけの使用可能な一意的なポート番号を用意しておかなければなりません。

3. クライエントの実行可能ファイルの名前を指定します。このファイル名は、以下のように指定されます。

```
EXEC_NAME=./dtwcdtypeid
```

ここで、*dbtypeid* はデータベースの型識別子です。以下の有効な実行可能ファイル名については、表6 を参照してください。

表6. クライエットの実行可能ファイル名

クライエットの説明	クライエットのタイプ	名前		使用可能なプラットフォーム					
		UNIX	Windows NT または OS/2	AIX	NT	OS/2	HP	SUN	PTX
DB2 プロセスのクライアント	DTW_SQL	dtwcdb2	dtwcdb2.exe	Y	Y	Y	Y	Y	N
ODBC プロセスのクライアント	DTW_ODBC	dtwcodbc	dtwcodbc.exe	Y	Y	N	N	N	N
Oracle プロセスのクライアント	DTW_ORA	dtwcora	dtwcora.exe	Y	Y	N	N	N	N

4. クライエットが関連付けられているデータベース名を指定する。

```
DATABASE=db_name
```

ここで、*db_name* は、クライアントが関連付けられているデータベース名(たとえば、MYDATABASE) です。

5. 任意選択: DB2 データベースに接続するために接続管理プログラムを開始したユーザー ID と同じユーザー ID を Net.Data が使用するよう、LOGIN 変数と PASSWORD 変数のデフォルト値を、*USE_DEFAULT に変更します。これらのデフォルト値を指定することにより、この情報を構成ファイルに配置するのを避けることができます。たとえば、39ページの図6 のサンプル構成ファイルの 14 ~ 15 行を以下の行に置き換えます。

```
LOGIN=*USE_DEFAULT
PASSWORD=*USE_DEFAULT
```

ヒント: 構成ファイルで複数のクライアント記入項目を定義すると、さまざまなデータベース・ログインと特定のデータベースのパスワードを指定することができます。

Java アプリケーションのクライアントの構成

1. クライエットの環境ステートメントを入力する。

```
CLIETTE DTW_JAVAPPS
```

2. MIN_PROCESS および MAX_PROCESS の値を決定する。MIN_PROCESS は、接続管理プログラムが開始したときに、開始すべきプロセスの数を指定します。その後は、同時要求が着信すると、接続管理プログラムは、指定された MAX_PROCESS の値に達するまで、必要に応じて 1 つずつクライアントを追加しながら、さらに多くのクライアントを開始します。

MIN_PROCESS および MAX_PROCESS ステートメントを入力します。

```
MIN_PROCESS=min_num  
MAX_PROCESS=max_num
```

パラメーター :

min_num

接続管理プログラムが開始したときに開始されるクライアント・プロセスの数。このクライアントの数に見合うだけの使用可能な一意的なポート番号を用意しておかなければなりません。

max_num

同時に実行可能な追加クライアントの最大数。このクライアントの数に見合うだけの使用可能な一意的なポート番号を用意しておかなければなりません。

Live Connection の構成のためのヒント :

- クライアント名は、クライアントの集合を一意的に識別するために、接続管理プログラムによって使用されます。
- データベースのクライアントの場合、アクセスするつもりデータベースごとに、1 つの名前付きのクライアント集合を持たなければなりません。ほとんどアクセスすることのないデータベースの場合は、クライアントの最小および最大数を 1 に設定することができます。別の方法としては、最小数を 0 にすることもできます。この意味は、Net.data の要求がクライアントに対してなされるまで、プロセスは開始しない、ということです。
- クライアントの名前は、初期設定ファイル内のクライアント・タイプに対する ENVIRONMENT ステートメントで参照されるクライアント名と一致していなければなりません。クライアント名には変数を含めることができ、データベース・クライアントの場合は、クライアント名には変数参照 \$(DATABASE) を含める必要があります。ENVIRONMENT ステートメントにおけるクライアント名のデフォルト値は、DTW_SQL:\$(DATABASE) です。変数参照は、初期設定ファイル内では使用できますが、Live Connection の構成ファイル内では使用できません。

DATABASE 変数は、Net.Data マクロで定義されます。マクロで SQL ステートメントが現れた場合、Net.Data の初期設定ファイルの \$(DATABASE) 変数参照は、DATABASE の現行値で置き換えられます。

この方法を使用して、複数のデータベースにアクセスすることができます。Net.Data のマクロに、アクセスするデータベースが 3 つあり (たとえば、D1、D2、および D3)、初期設定ファイルに標準の CLIETTE "DTW_SQL:\$(DATABASE)" 行がある場合は、Live Connection の構成ファイルには、次のような 3 つのセクションが必要になります。

```
CLIETTE DTW_SQL:D1{ ...}  
CLIETTE DTW_SQL:D2{....}  
CLIETTE DTW_SQL:D3{....}
```

- プロセスは、開始されますが、停止しません。プロセスの最大数を M に設定し、好きなときに M 個のプロセスを同時に使用する場合、接続管理プログラムをシャットダウンするまで、プロセスはアクティブの状態にあります。したがって、MAX_PROCESS の値を非常に大きくしてしまい、ほとんど使用することのないプロセスまで開始して、システム・リソースを使い果たしてしまうことはしたくないでしょう。

推奨：MIN_PROCESS および MAX_PROCESS には異なる値を使用するようにし、システムにとってどの作業がベストかを調べてみてください。接続管理システムが、指定した最大値よりも多くの要求を受け取る場合は、クライアントが処理を終えるまで、最後の要求は待機しています。クライアントが使用可能になると、待機中の要求が処理されます。待機している要求のこのプロセスは、アプリケーションのユーザーから見るすることができます。

- 別の名前のセクションの同じタイプのクライアントを使用することができます。たとえば、構成ファイルのすべての DB2 のデータベース・セクションは、同じクライアントのタイプを使用します。同じ名前を持つ 2 つのセクションを持つことはできません。

CGI を使用していて、いくつかのデータベースだけに Live Connection を使用させたい場合、単に、構成ファイルに、希望するデータベースをリストします。Net.Data が Net.Data のマクロを処理していて、SQL 関数を検出した場合、Net.Data は、接続管理プログラムに特定のクライアントを要求します。接続管理プログラムが、そのタイプのクライアントを持っていない場合、接続管理プログラムは、NO_CLIETTE_AVAIL メッセージで応えます。Net.Data は、その代わりに、DLL バージョンでその要求を処理します。

接続管理プログラム・サービスの自動開始のための構成

Windows NT 上では、接続管理プログラムを、コマンド行からではなく、Windows NT のサービスとして開始させるよう指定することができます。接続

管理プログラムを Windows NT のサービスとして実行すると、マシンを開始するたびに、接続管理プログラムを自動的に開始させることができます。

重要: 自動的に接続管理プログラムを開始するようにセットアップする前に、接続管理プログラムをコマンド行から開始し、Live Connection の構成ファイルが正しいことを確認します。

- Windows NT のタスクバーから、「スタート (Start) -> 設定 (Settings) -> コントロール・パネル (Control Panel) -> サービス (Services)」と選択する。
- 「Net.Data の接続管理プログラム (Net.Data Connection Manager)」を選択し、次に「開始 (Startup)」ボタンをクリックする。
- 「自動開始のタイプ (Automatic startup type)」を選択し、「OK」をクリックする。

CGI と一緒に使用するための Web サーバーの構成

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

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

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

Pass /*

Map ディレクティブ

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

Exec ディレクティブ

Exec ディレクティブにより、Web サーバーは CGI ライブラリーの

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

FastCGI のための Net.Data の構成

Net.Data は、Apache Web サーバーおよび Lotus Domino Go Webserver 上で、FastCGI プロセスとして実行することができます。FastCGI は、CGI のアプリケーション分離機能により、他の Web API プログラムと同等のパフォーマンスを提供します。FastCGI は、AIX および Sun Solaris のオペレーティング・システムでサポートされます。

FastCGI を使用する前に、Apache Web Server 1.2.0 以上、または Domino Go Webserver 4.6.2.5 以上がインストールされていることを確認してください。

Net.Data for FastCGI を構成するには、以下を行います。

1. 使用しているオペレーティング・システムの Web サーバーおよび FastCGI の構成ファイルを構成する。

Apache Web サーバーの場合：

http.conf ファイルを更新する。

- 新規アプリケーションを以下のように宣言する。

```
AppClass inst_dir
-processes proc_num
-initial-env LIBPATH=libpath
-initial-env ORACLE_HOME=oracle_path
-initial-env ORACLE_SID=oracle_instance
-initial-env DB2INSTANCE=db2_instance
-initial-env RXQUEUE_OWNER_PID=REXX_perf_var
-initial-env LANG=locale
```

- FastCGI モジュールを以下のように宣言する。

```
<location /fcgi-bin>
SetHandler fastcgi-script
</location>
```

Domino Go Webserver の場合：

httpd.conf および fcgi.conf ファイルを更新する。

- httpd.conf ファイルで、以下のようにサービス・セクションを宣言する。

```
ServerInit /u/mydir/http/fcgi-bin/fcgi.o:FCGIInit
/u/mydir/http/fcgi.conf service/fcgi-bin/*
/u/mydir/http/fcgi-bin/fcgi.o:FCGIDispatcher*ServerTerm
/u/mydir/http/fcgi-bin/fcgi.o:FCGIStop
```

- fcgi.conf ファイルで、以下のようにアプリケーションを宣言する。

```
Local {
  Exec inst_dir
  Role Responder
  URL /fcgi-bin/db2www
  BindPath /tmp/db2www.ibm
  NumProcesses proc_num
  Environ LIBPATH=libpath
  Environ ORACLE_HOME=oracle_path
  Environ ORACLE_SID=oracle_instance
  Environ DB2INSTANCE=db2_instance
  Environ RXQUEUE_OWNER_PID=REXX_perf_var
  Environ LANG=locale
  Environ NLSPATH=msg_catalog_path
  Environ MAXREQUEST=num_reqs
}
```

パラメーター：

inst_dir

Net.Data の実行可能ファイルのパスおよびディレクトリー名

Apache の場合：

AppClass /u/mydir/apache/fcgi-bin/db2www

Domino Go Webserver の場合：

Exec /u/mydir/http/fcgi-bin/db2www

Role Responder

Domino Go Webserver にのみ要求されるキーワード

URL Domino Go Webserver にのみ要求されるキーワードおよび URL アドレス。URL は、EXEC_PATH ステートメントに対して指定されたパスを指します。

BindPath

AIX 上の Domino Go Webserver のみに要求されるキーワードおよびパス・ステートメント。Net.Data および FastCGI によって使用される固有の UNIX のソケットのパス。

proc_num

同時にハンドル可能な要求数。デフォルトは 1 です。しかし、アプリケーションの要求に基づき、パフォーマンスが許すところまで大きくしても構いません。調整に関する情報については、218ページの『FastCGI の使い方』を参照してください。

Apache の場合：

-processes 7

ICS あるいは Domino Go Webserver の場合 :

NumProcesses 7

libpath Net.Data の初期設定ファイルの各 ENVIRONMENT ステートメントで宣言された LIBPATH (共有ライブラリーあるいは DLL) ステートメント。DB2 にアクセスする場合は、LIBPATH ステートメントに DB2 のライブラリー・ディレクトリーへのパスが含まれている必要があります。たとえば、以下のような場合です。

/usr/lpp/db2_05_00/lib

Apache の場合 :

-initial-env LIBPATH=/u/mydir/apache/lib:/u/mydir/apache:/usr/lib

Domino Go Webserver の場合 :

Environ LIBPATH=/u/mydir/http/lib:/u/mydir/http:/usr/lib

oracle_path

Oracle 使用時に必要です。Oracle のデータベースの実行可能ファイルのパスおよびディレクトリー。

Apache の場合 :

-initial-env ORACLE_HOME=/home.native/oracle/product/7.2

Domino Go Webserver の場合 :

Environ ORACLE_HOME=/home.native/oracle/product/7.2

oracle_instance

Oracle 使用時に必要です。Oracle のデータベースのインスタンス。Oracle には Live Connection を使用しなければなりません。

Apache の場合 :

-initial-env ORACLE_SID=mvpdb2

Domino Go Webserver の場合 :

Environ ORACLE_SID=mvpdb2

db2_instance

DB2 使用時に必要です。DB2 のデータベースのインスタンス。

Apache の場合 :

-initial-env DB2INSTANCE=wwwinst

Domino Go Webserver の場合 :

Environ DB2INSTANCE=wwwinst

REXX_perf_var

AIX で REXX を使用するときには必要です。パフォーマンス変数は、AIX オペレーティング・システム上で、FastCGI および REXX と一緒に使用します。デフォルトは 0 です。他の製品およびオペレーティング・システムの場合、Net.Data のマクロでこの変数を宣言します。この変数についての詳細は、267 ページの『付録B. Net.Data for AIX』を参照してください。

Apache の場合 :

-initial-env RXQUEUE_OWNER_PID=0

Domino Go Webserver の場合 :

Environ RXQUEUE_OWNER_PID=0

locale UNIX の locale 変数。米国英語には、En_US を使用します。

Apache の場合 :

-initial-env LANG=En_US

Domino Go Webserver の場合 :

Environ LANG=En_US

NLSPATH

メッセージ・カタログのディレクトリーのロケーションを指定します。

Apache の場合 :

-initial-env NLSPATH=/usr/lib/nls/msg/%L/%N

Domino Go Webserver の場合 :

Environ NLSPATH=/usr/lib/nls/msg/%L/%N

MAXREQUEST

Web サーバーが Net.Data の Fast-CGI プロセスをリサイクルして新規のプロセスを開始する前に、その Fast-CGI プロセスが処理する要求の数を指定します。

Apache の場合 :

-initial-env MAXREQUEST=5000

Domino Go Webserver の場合 :

Environ MAXREQUEST=5000

2. **Apache** の場合 : fgi-bin ディレクトリーを、新規のスクリプトのエイリアス `ScripAlias /fcgi-bin/ /u/mydir/apache/fci-bin` として、`srm.conf` ファイルに追加します。
3. 静的あるいは動的に生成された Web ページの任意のハイパーリンクを、CGI-BIN から FCGI-BIN へ移送します。たとえば、以下のような場合です。

```
<a href="http://server /fcgi-bin/db2www/filename.ext/block /  
[?name=val&...]">any text</a>
```

4. CGI-BIN ではなく、FCGI-BIN を使って Net.Data の URL 呼び出しのためのエンド・ユーザーの文書を変更します。たとえば、以下のような場合です。

```
http://server/fcgi-bin/db2www/filename.ext/block /[?name=val&...]
```

Java サブレットおよび Java Beans と一緒に使用するための Net.Data の構成

サブレットの命令、登録および使用については、ご使用の Web サーバー・ドキュメンテーションを参照してください。Net.Data のサブレットは、`NetDataServlets.jar` ファイルに含まれています。ご使用の Web サーバーでは、`inst_dir/servlet-lib/NetDataServlets.jar` および `inst_dir/servlet-lib` を、ユーザーの CLASSPATH に追加する必要があります。

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

Web サーバー API と一緒に使用するための Net.Data の構成

CGI ではなく、Web サーバーのアプリケーション・プログラミング・インターフェース (API) を使用することにより、Net.Data のパフォーマンスを大きく改善することができます。Net.Data は以下のサーバー API をサポートしています。

- Lotus Domino Go Webserver API (GWAPI)
- Microsoft Internet Server API (ISAPI)
- Netscape API (NSAPI)

各 API の詳細については、217ページの『Web サーバー API の使い方』および使用している Net.Data のバージョンの README ファイルを参照してください。

要件： Net.Data を GWAPI、ISAPI、または NSAPI モードで実行するには、Net.Data の DLL または共有ライブラリーをそのサービス・ディレクティブとして使用するよう、Web サーバーを再構成しなければなりません。再構成の後、Net.Data の初期設定ファイルに加えた変更が有効になるように、Web サーバーを再始動しなければなりません。デフォルトでは、Net.Data は CGI モードで実行されます。

以下のセクションでは、Web サーバー API モードを実行するよう Net.Data と Web サーバーを構成する方法について説明します。一般的なステップと例が示されていますが、それらは、ユーザーのオペレーティング・システムでは異なる場合があります。特定の指示については、ご使用のオペレーティング・システムの Net.Data の README ファイルを参照してください。

GWAPI を構成するには、以下のようになります。

1. Web サーバーを停止する。
2. GWAPI DLL あるいは共有ライブラリーが、サーバーの CGI-BIN あるいは ICAP-LIB ディレクトリーにあることを確認する。

固有のファイルおよびディレクトリー名については、ご使用のオペレーティング・システムの Net.Data の README ファイルあるいはプログラム・ディレクトリーを参照してください。

3. サービス・ステートメントを、ユーザーの Web サーバーの構成ファイル (httpd.conf あるいは httpd.cnf) に追加し、API を呼び出す。

たとえば、以下のような場合です。

```
Service /cgi-bin/db2www* /home/http/cgi-bin/dtwicapi.o:dtw_icapi*
```

固有のファイルおよびディレクトリー名については、ご使用のオペレーティング・システムの Net.Data の README ファイルを参照してください。

4. Web サーバーを再始動する。

GWAPI は、既存のアプリケーションをサポートするための完全互換性を持っています。CGI の場合に使用すると同じ方法を使用して、URL、フォーム、あるいは GWAPI とのリンクを呼び出します。CGI を使用して正常に実行されるマクロであればどのマクロも、GWAPI を使用して正常に実行されます。これらのマクロに対して変更を加える必要はありません。

ISAPI を構成するには、以下のようになります。

1. Web サーバーを停止する。
2. Net.Data に付属している ISAPI 対応の DLL を、サーバーのサブディレクトリーにコピーします。たとえば、以下のような場合です。

`/inetsrv/scripts/dtwisapi.filetype`

ここで、*filetype* は Window NT および OS/2 用の `.dll` であり、`.o` は UNIX オペレーティング・システム用です。

固有のファイルおよびディレクトリー名については、ご使用のオペレーティング・システムの Net.Data の README ファイルを参照してください。

3. ISAPI は CGI 処理をバイパスするため、フォームおよびリンクで URL の `cgi-bin/db2www/` の部分は必要ありません。代わりに、`dtwisapi.filetype` を使用します。たとえば、以下の URL は、Net.Data を CGI プログラムとして起動します。

`http://server1.stl.ibm.com/cgi-bin/db2www/test1.d2w/report`

次に、以下の URL を使用して Net.Data を ISAPI プラグインとして起動する必要があります。

`http://server1.stl.ibm.com/scripts/dtwisapi.dll/test1.d2w/report`

4. MACRO_PATH で指定されるディレクトリーまたは Web サーバーの現行ディレクトリーの下サブディレクトリー `/order/` に、ユーザーのマクロ `test1.d2w` を保管する場合には、以下の URL を使用して CGI モードの Net.Data を起動します。

`http://server1.stl.ibm.com/cgi-bin/db2www/orders/test1.d2w/report`

次に、Net.Data を ISAPI モードで起動するための同等の URL は、以下のとおりです。

`http://server1.stl.ibm.com/scripts/dtwisapi.dll/orders/test1.d2w/report`

5. Web サーバーを再起動する。

NSAPI を構成するには、以下のようにします。

1. Web サーバーを停止する。
2. Net.Data に付属している NSAPI 対応の DLL を、サーバーのディレクトリーにコピーする。たとえば、以下のような場合です。

`/netscape/server/bin/httpd/dtwnsapi.filetype`

ここで、*filetype* は Window NT および OS/2 用の `.dll` であり、`.o` は UNIX オペレーティング・システム用です。

固有のファイルおよびディレクトリー名については、ご使用のオペレーティング・システムの Net.Data の README ファイルを参照してください。

3. サーバーの構成ファイルに、以下にリストされている変更を加えます。オペレーティング・システムの違いについては、ご使用のオペレーティング・シ

システムの Net.Data の README ファイルを参照してください。

obj.conf ファイルの最初に以下を追加します。

```
Init fn="load-modules" shlib="<path>dtwnsapi.dll" funcs=dtw_nsapi
```

obj.conf Services ディレクティブに、以下を追加します。

```
Service fn="dtw_nsapi" method=(GET|HEAD|POST)
      type="magnus-internal/d2w"
```

mime.types このタイプ (d2w はマクロのデフォルト拡張子) を追加します。任意の 2 文字を組み合わせて指定できます。

```
type=magnus-internal/d2w exts=d2w
```

4. Net.Data のマクロを、netdata/macro ディレクトリーから、以下のサーバーのルート文書ディレクトリーに移動する。

```
/netscape/server/docs/
```

5. 初期設定ファイルでは、サーバーのルートの文書ディレクトリーを、MACRO_PATH ステートメントに追加します。この変更により、Net.Data は、マクロをどこで探せばよいか分かります。
6. NSAPI は CGI 処理をバイパスするため、フォームおよびリンクで URL の cgi-bin/db2www/ の部分は必要ありません。d2w ファイルの型をもつファイルは Net.Data のマクロであることを、サーバーは知っています。それは、Netscape の構成ファイルを変更したときに、それを定義したからです。たとえば、以下の URL は、Net.Data を CGI プログラムとして起動します。

```
http://server1.stl.ibm.com/cgi-bin/db2www/test1.d2w/report
```

以下の URL は、Net.Data を NSAPI プラグインとして起動します。

```
http://server1.stl.ibm.com/test1.d2w/report
```

7. Web サーバーを再始動する。

Net.Data のマクロをいくつかのディレクトリーに保持している場合は、最後の 3 つのステップ が次のように変わります。

1. そのディレクトリーと、そこに含まれている Net.Data のマクロと一緒に、サーバーのルート文書ディレクトリーに移動する。
2. 初期設定ファイル内の MACRO_PATH 変数を更新して、マクロが格納されているすべてのディレクトリーおよびサブディレクトリーを含める。
3. これらの Net.Data のマクロを指すリンクおよびフォームを変更し、そのディレクトリー名を保持する。たとえば、CGI モードで実行している場合、以下の URL は、/orders/ ディレクトリーに保管されている Net.Data のマクロを呼び出します。

<http://server1.stl.ibm.com/cgi-bin/db2www/orders/test1.d2w/report>

更新された URL は、NSAPI モードで Net.Data を起動するために使用され、以下のように短くなりますが、ディレクトリー名は保持します。

<http://server1.stl.ibm.com/orders/test1.d2w/report>

Net.Data の管理ツールを用いた Net.Data の構成

Net.Data の管理ツールは、Net.Data の初期設定ファイル (DB2WWW.INI)、および Windows NT、AIX、そして OS/2 オペレーティング・システム上の Live Connection (dtwcm.cnf) の構成ファイルの構成および管理を支援します。このツールを使用すると、以下のタスクを完成させることができます。

- 54ページの『管理ツールの開始』
- 54ページの『パス・ステートメントの構成』
- 56ページの『ポートの構成』
- 57ページの『クライアントの構成』
- 63ページの『言語環境の構成』
- 68ページの『構成変数の定義』

管理ツールのセットアップとソフトウェア前提条件を満たしていることの確認については、『事前準備』を参照してください。

事前準備

1. Net.Data の言語環境、データベース、クライアント、ポート、および構成変数の構成のプランを立てる。
2. CD-ROM から Net.Data をインストールする。
3. Java の実行時ライブラリー (オペレーティング・システムごとの JDK 1.1 およびそれ以降のバージョン) をインストールする。詳しくは、ご使用のオペレーティング・システムの Net.Data の README ファイルをチェックしてください。

JDK をインストールした後、CLASSPATH に classes.zip があることを確認してください。

4. DB2 Universal Database と一緒にパッケージングされている IBM JDBC ドライバーをインストールした場合は、Java の CLASSPATH ステートメントに、そのドライバーを追加し、DB2 のログイン・テストを使用可能にする。
5. Net.Data の管理ツールのプログラムが格納されているディレクトリーに変更する。

OS/2 および Windows NT の場合：

inst_dir¥connect¥*admin_directory*。ここで、*inst_dir* はインストール時に Net.Data に対して指定したディレクトリーで、*admin_directory* は管理ツールのファイルが存在するディレクトリーです。

AIX の場合：

/usr/lpp/internet/db2www/db2.v2/*admin_directory*。ここで *admin_directory* は、管理ツールのファイルが存在するディレクトリーです。

管理ツールの開始

使用するオペレーティング・システムによって、管理ツールの開始方法が異なります。

OS/2 および Windows NT の場合:

IBM Net.Data バージョン 2 のフォルダーから、「**Net.Data の管理ツール (Net.Data Admin Tool)**」アイコンを選択します。

AIX の場合:

Net.Data のインストール・ディレクトリー (*inst_dir*) に変更します。コマンド行から、*ndadmin* を入力し、ツールを開始します。

管理ツールが立ち上がり、「Net.Data の管理 (Net.Data Administration)」ノートブックが表示されます。

パス・ステートメントの構成

「**パス (Path)**」ページを使用して、Net.Data が、Net.Data のマクロの処理に必要なファイルを見つけるためのパス・ステートメントを追加、変更、あるいは削除します。これらのステートメントは、23ページの『パス構成ステートメント』で説明されています。55ページの図7 は、「**パス (Path)**」ページを示しています。

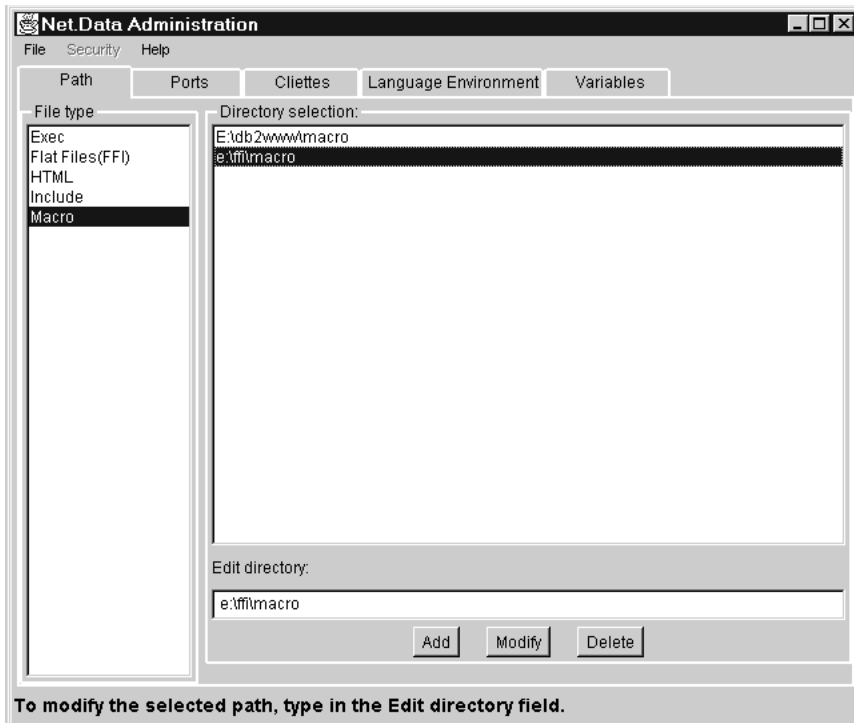


図7. Net.Data の管理ツールの「パス (Path)」ページ. このページを使用して、パス・ステートメントの追加、変更、あるいは削除を行います。

構成のヒント： HTML ファイルのタイプは、1 つのパスしか持つことができません。

パス・ステートメントを追加するには、以下を行います。

1. 管理ツールを開始する。
2. 「パス (Path)」ページから、ファイル・タイプを選択し、「ファイル・タイプ (File type)」から、たとえば、「実行 (Exec)」を選択する。
3. 「ディレクトリーの編集 (Edit directory)」フィールドで、新規のパスを入力し、「追加 (Add)」ボタンをクリックする。

指定されたパスが存在しない場合は、警告ウィンドウがオープンします。ディレクトリーが選択されていない場合は、新規のディレクトリーが、リストの最後の項目として追加されます。

4. 管理ツールをクローズするか、別のタブをクリックして、追加の構成タスクを完了する。

パス・ステートメントを変更するには、以下を行います。

1. 管理ツールを開始する。
2. 「パス (Path)」ページで、「ファイル・タイプ (File type)」リストから変更したいファイル・タイプを選択する。
3. 「ディレクトリーの選択 (Directory selection)」リストで、変更したいパスを選択する。選択されたパスが、「ディレクトリーの編集 (Edit directory)」フィールドでオープンします。
4. 「ディレクトリーの編集 (Edit directory)」フィールドでパスを変更し、「変更 (Modify)」ボタンをクリックする。入力したパスが存在しない場合は、警告ウィンドウがオープンします。
5. 管理ツールをクローズするか、別のタブをクリックして、追加の構成タスクを完了する。

パス・ステートメントを削除するには、以下を行います。

1. 管理ツールを開始する。
2. 「パス (Path)」ページで「ファイル・タイプ (File type)」リストから削除したいファイル・タイプを選択する。
3. 「ディレクトリーの選択 (Directory selection)」フィールドで、削除したいパスを選択する。選択されたパスが、「ディレクトリーの編集 (Edit directory)」フィールドでオープンします。
4. 「削除 (Delete)」ボタンをクリックする。
5. 管理ツールをクローズするか、別のタブをクリックして、追加の構成タスクを完了する。

ポートの構成

「ポート (Port)」ページを使用して、Net.Data が使用する TCP/IP のポート番号を指定します。 57ページの図8 は、「ポート (Port)」ページを示しています。

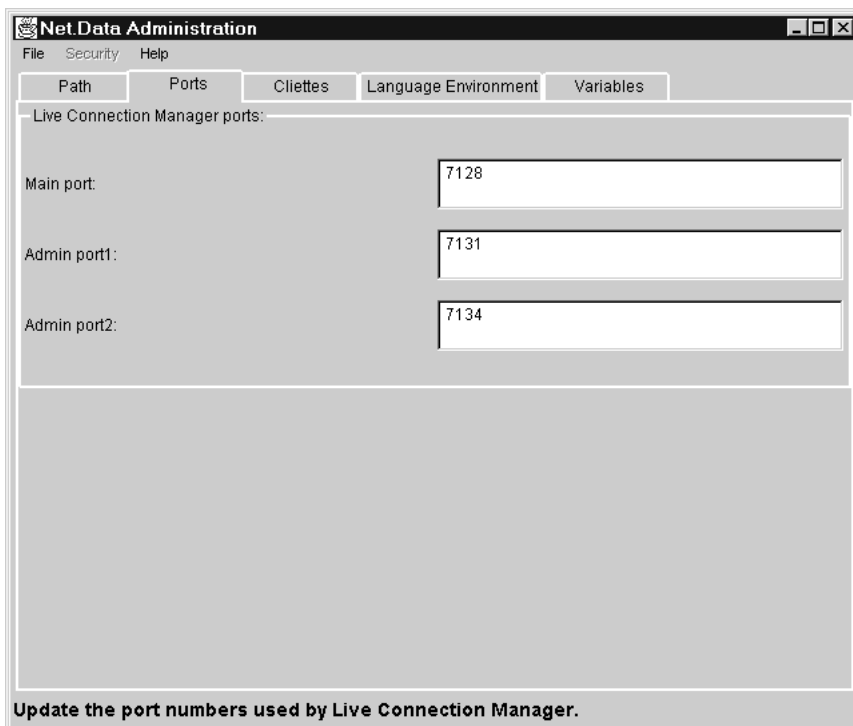


図 8. Net.Data の管理ツールの「ポート (Port)」ページ。このページを使用して、ポートを指定します。

TCP/IP のポート番号を指定するには、以下のようにします。

1. 管理ツールを開始する。
2. 「**ポート (Port)**」ページで、ポート・フィールドごとに一意のポート番号を入力します。管理ツールは、タブで次のフィールドに移動したときに、各フィールドに入力したポート番号を検査します。
3. 管理ツールをクローズするか、別のタブをクリックして、追加の構成タスクを完了する。

クライアントの構成

「**クライアント (Cliette)**」ページを使用して、Live Connection のデータベースのクライアントを追加、変更、あるいは削除します。また、データベースおよびデータベースのクライアントの管理者のユーザー ID とパスワードを管理することもできます。クライアントについての詳細は、218ページの『接続管理』で提供されています。58ページの図9 は、「**クライアント (Cliette)**」ページを示しています。

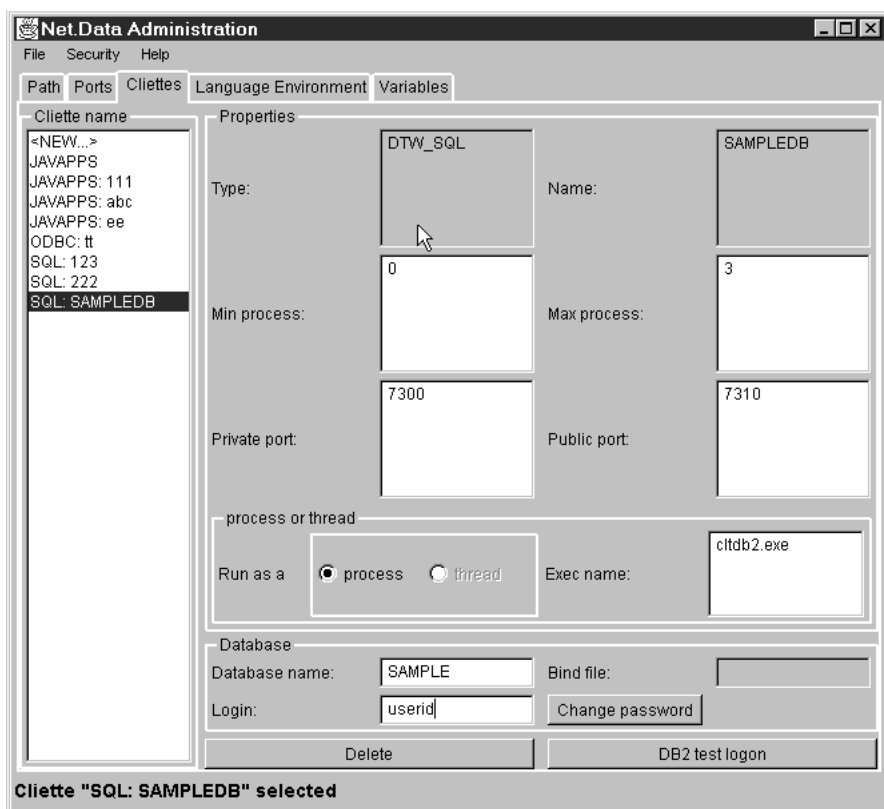


図9. Net.Data の管理ツールの「クライアント (Cliette)」ページ. このページを使用して、クライアントの追加、変更、あるいは削除を行います。

クライアントを追加するには、以下のように行います。

1. 管理ツールを開始する。
2. 「クライアント (Cliette)」ページで、「クライアント名 (Cliette name)」リストから「<新規...> (<new...>)」を選択する。「クライアントの追加 (Add a cliette)」ウィンドウがオープンします。

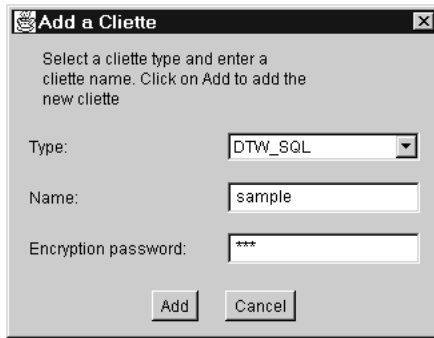


図 10. Net.Data の管理ツールの「クライエットの追加 (Add a Clette)」ウィンドウ. このページを使用して、クライエットを追加します。

- 暗号化が使用可能になっている場合には、初めてクライエットを作成したり変更したときに、暗号化パスワードの入力をプロンプト指示されます。このパスワードは保管されるので、その後は再入力する必要はありません。
3. 「**タイプ (Type)**」リストから、クライエットのタイプを選択する。
 4. 「**名前 (Name)**」フィールドに、新規のクライエット名を入力する。この名前は、データベースの名前、あるいは別の一意のクライエット名にすることができます。たとえば、MYCLLETTE です。
 5. 「**暗号パスワード (Encryption password)**」フィールドが使用可能になっていれば、暗号パスワードを入力する。管理ツールがパスワードを保管してくれるため、パスワードを再入力する必要はありません。
 6. 「**追加 (Add)**」ボタンをクリックします。

新規クライエットが作成され、クライエット・リストの一番下に追加されます。さらに、新規の名前が強調表示され、そのクライエットのデフォルトのプロパティが、「**プロパティ (Properties)**」グループ・ボックスに表示されます。これらの値を変更して、ユーザーの構成に適合させることができます。

7. 管理ツールをクローズするか、別のタブをクリックして、追加の構成タスクを完了する。

クライエットを変更するには、以下のように行います。

1. 管理ツールを開始する。
2. 「**クライエット (Clette)**」ページで、「**クライエット名 (Clette name)**」リストから変更したいクライエット名を選択する。クライエットのプロパティが、「**プロパティ (Properties)**」グループ・ボックスに表示されます。

3. 必要に応じて、「**プロパティ (Properties)**」グループ・ボックスからプロパティを変更する。
- a. 「**タイプ (Type)**」フィールドは、定義中のクライアントのタイプを表示し、言語環境のタイプ名に対応します。Net.Data は、ユーザーが新規のクライアントを追加すると、このフィールドに追加し、選択項目が、「クライアントの追加 (Add a Client)」ウィンドウの「**クライアント・タイプ (Client type)**」リストで定義されます。
 - b. 「**名前 (Name)**」フィールドは、クライアント名を表示します。普通この名前は、データベースの名前です。Net.Data は、新規のクライアントを追加すると、このフィールドに追加します。
 - c. 接続管理プログラムが起動したときに開始することができるクライアント・プロセスの数を、「**最小プロセス (Min process)**」フィールドに入力します。プロセスごとに、一意のポート・アドレスが必要になります。最小プロセス値についての詳細は、38ページの『Live Connection の構成』を参照してください。
 - d. 接続管理プログラムを開始したときに開始したプロセスに加えて、同時に実行することができるクライアント・プロセスの数を、「**最大プロセス (Max process)**」フィールドに入力する。プロセスごとに、一意のポート・アドレスが必要になります。最大プロセス値の詳細については、38ページの『Live Connection の構成』を参照してください。
 - e. 一意のポート番号を「**プライベート・ポート (Private port)**」フィールドに入力し、接続管理プログラムと共に開始するクライアント・プロセスと一緒に使用するための開始ポート番号を指定する。「**最小プロセス (MIN Process)**」値で指定された処理ごとに、追加ポート番号が使用されます。たとえば、「**プライベート・ポート (Private port)**」にポート番号 7012、「**最小プロセス (Min process)**」に 5 を指定すると、ポート番号 7012 ～ 7016 が使用されます。この番号は、システムでは、他のポート割り当てと競合してはいけません。
 - f. 一意のポート番号を、「**パブリック・ポート (Public port)**」フィールドに入力し、追加プロセスが開始されたときに開始されるクライアント・プロセスと一緒に使用される開始ポート番号を、「**最大プロセス (Max process)**」フィールドで指定された番号まで指定する。追加ポート番号が、プロセスごとに使用されます。たとえば、「**パブリック・ポート (Public port)**」にポート番号 7020 を指定し、「**最大プロセス (Max process)**」に値 5 を指定すると、ポート番号 7020 ～ 7024 が使用されます。この番号は、システムでは、他のポート割り当てと競合してはいけません。

- g. 「**実行ファイル名 (Exec name)**」フィールドには、クライアントの実行可能ファイルの名前が表示されます。
- 4. クライエントをデータベースと一緒に使用している場合は、必要に応じて、「**データベース (Database)**」の値を変更する。
 - a. クライエントが関連付けられているデータベースのデータベース名、たとえば MYDBASE、を「**データベース名 (Database name)**」フィールドで指定する。
 - b. 「**バインド・ファイル (Bind file)**」フィールドは、使用中のクライアントのタイプのバインド・ファイルの名前とパスを含んでいます。
 - c. 「**ログイン (Login)**」フィールドは、データベースに接続するためのログイン・ユーザー ID を指定します。
 - d. 「**パスワードの変更 (Change password)**」押しボタンは、「データベースのパスワードの変更 (Change Database Password)」ウィンドウをオープンする。暗号化パスワードと新規のパスワードを、2 回入力します。「**機密保護 (Security)**」プルダウン・メニューで指定される暗号化関数を使用して、データベースのパスワードを暗号化することができます。
- 5. 「**ファイル (File)**」を選択し、次に「**保管 (Save)**」を選択して、変更を保管する。
- 6. 管理ツールをクローズするか、別のタブをクリックして、追加の構成タスクを完了する。

DB2 データベースのログオンと接続をテストするには、以下を行います。

- 1. 管理ツールの「**クライアント (Cliette)**」ページで、「**DB2 テストのログオン (DB2 test logon)**」押しボタンをクリックする。テストが完了すると確認ウィンドウがオープンし、接続テストの状況が表示されます。
- 2. ウィンドウをクローズし、構成を続行する、あるいは管理ツールをクローズする。

クライアントを削除するには、以下を行います。

- 1. 管理ツールを開始する。
- 2. 「**クライアント (Cliette)**」ページで、「**クライアント名 (Cliette name)**」リストから、削除したいクライアントの名前を選択する。
- 3. 「**削除 (Delete)**」ボタンをクリックする。
- 4. 管理ツールをクローズするか、別のタブをクリックして、追加の構成タスクを完了する。

クライアントのユーザー ID およびパスワードの暗号化をオンにするには、以下を行います。

暗号化は、データベースとクライアントとの接続のための機密保護を提供します。暗号化をオンにすると、Live Connection の構成ファイルのデータベースのすべてのパスワードが暗号化され、アクセスと復号のための暗号が必要になります。

要件： 暗号化を使用するには、Net.Data バージョン 2 の Live Connection の構成ファイルを使用しなければなりません。

1. **重要：** 使用している Live Connection の構成ファイル <path>dtwcm.cnf のコピーをバックアップしてください。暗号化パスワードを紛失したり、データベースのパスワードを復号してパスワードを復元したい場合に、このファイルが必要になります。
2. 管理ツールの「**クライアント (Cliette)**」ページで、「**機密保護 (Security)**」 -> 「**暗号化をオン (Turn encryption on)**」としてプルダウン・メニューのオプションを選択する。「暗号化をオン (Turn Encryption On)」確認ウィンドウがオープンします。
3. 「**はい (Yes)**」を押して、続行する。「暗号化パスワード (Encryption Password)」ウィンドウがオープンします。
4. 暗号化されたパスワードを持つクライアントを操作するための許可を得るために、パスワードを 2 回入力する。
5. 「**OK**」をクリックして、新規のパスワードを定義し、ユーザーのクライアントのデータベースのパスワードをすべて暗号化する。

クライアントのユーザー ID およびパスワードをオフにするには、以下を行います。

1. 管理ツールの「**クライアント (Cliette)**」ページで、「**機密保護 (Security)**」 -> 「**暗号化をオフ (Turn encryption off)**」としてプルダウン・メニューのオプションを選択する。「暗号化をオフ (Turn Encryption Off)」確認ウィンドウがオープンします。
2. 「**はい (Yes)**」を押して、続行する。すべてのパスワードが、機密保護上の理由で、*USE_DEFAULT に設定されます。パスワードは、Live Connection ファイル <path>dtwcm.cnf のバックアップ・コピーから復元することができます。

暗号化のためのパスワードを変更するには、以下を行います。

1. 管理ツールの「**クライアント (Cliette)**」ページで、「**機密保護 (Security)**」 -> 「**暗号化パスワードの変更 (Change Encryption**

Password)」としてプルダウン・メニューのオプションを選択する。「暗号化パスワードの変更 (Change Encryption Password)」確認ウィンドウがオープンします。

2. 「はい (Yes)」を押して、続行する。「暗号化パスワードの変更 (Change Encryption Password)」ウィンドウがオープンします。
3. 古い暗号化パスワードを 1 回入力し、新規のパスワードを 2 回入力する。
4. 「OK」をクリックして、暗号化パスワードを変更する。

データベースのパスワードを変更するには、以下を行います。

1. 管理ツールの「クライアント (Client)」ページで、「パスワードの変更 (Change Password)」押しボタンをクリックする。「データベースのパスワードの変更 (Change Database Password)」ウィンドウがオープンします。
2. 暗号化パスワードを 1 回入力し、新規のデータベースのパスワードを、2 回入力する。
3. 「OK」をクリックして、パスワードを変更し、ウィンドウをクローズします。暗号化をオンにしておれば、変更されたデータベースのパスワードは、暗号化されます。

言語環境の構成

「言語環境 (Language Environment)」ページを使用して、Net.Data 言語環境の追加、変更、あるいは削除を行うことができます。言語環境については、29ページの『環境構成ステートメント』で議論されています。64ページの図11は、「言語環境 (Language Environment)」ページを示しています。

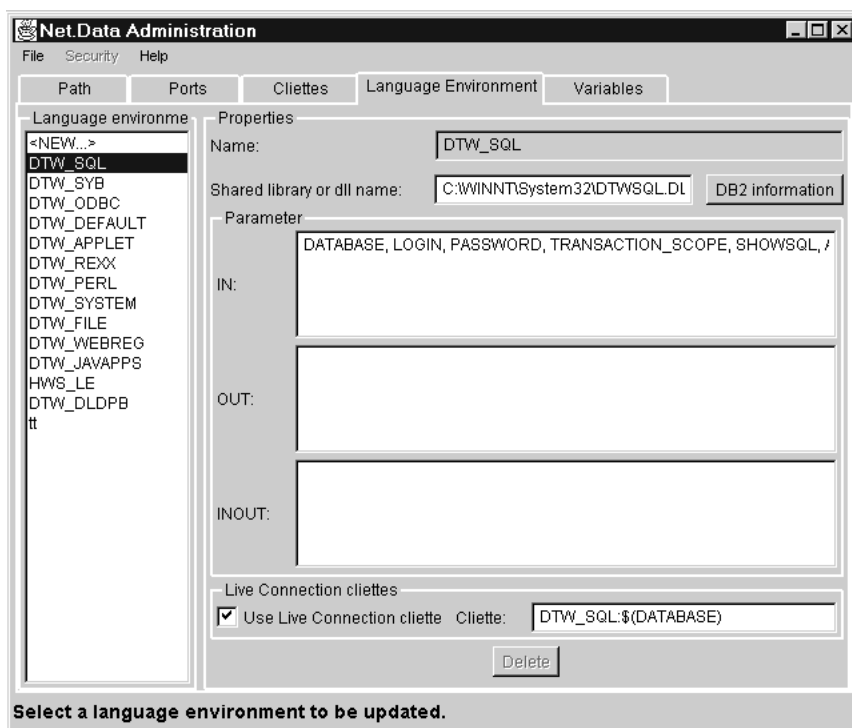


図 11. Net.Data 管理ツールの「言語環境 (Language Environment)」ページ. このページを使用して、言語環境を指定します。

言語環境を追加するには、以下を行います。

1. 管理ツールを開始する。
2. 「言語環境 (Language Environment)」ページで、「言語環境 (Language environment)」リストから「<新規...> (<NEW...>)」を選択する。「言語環境の新規追加 (Add a new language environment)」ウィンドウがオープンします。
3. フィールドに言語環境名を入力し、「追加 (Add)」ボタンをクリックする。「言語環境の追加 (Add a Language Environment)」ウィンドウがオープンします。

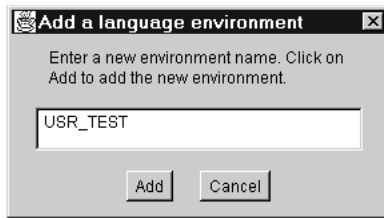


図 12. Net.Data の管理ツールの「言語環境の追加 (Add a Language Environment)」ウィンドウ. このページを使用して、新規の言語環境を指定します。

新規の言語環境が作成され、その名前が、言語環境のリストの一番下に追加されます。さらに、新規の名前が強調表示され、その言語環境のデフォルトのプロパティが、「**プロパティ (Properties)**」グループ・ボックスに表示されます。これらの値を変更して、ユーザーの構成に適合させることができます。

4. 管理ツールをクローズするか、別のタブをクリックして、追加の構成タスクを完了する。

言語環境を変更するには、以下を行います。

1. 管理ツールを開始する。
2. 「**言語環境 (Language Environment)**」ページで、「**言語環境 (Language environment)**」リストから、変更したい言語環境の名前を選択する。クライエットのプロパティが、「**プロパティ (Properties)**」グループ・ボックスに表示されます。
3. 必要に応じて、図12 で示されている「**プロパティ (Properties)**」グループ・ボックスでプロパティを変更する。
 - a. 「**名前 (Name)**」フィールドで、言語環境の名前を指定する。この名前は、クライエットを定義するのに使用した言語環境のタイプに対応します。この値を変更するには、「**言語環境 (Language environment)**」リストで、別の名前をダブルクリックする。言語環境のタイプの詳細については、29ページの『環境構成ステートメント』を参照してください。
 - b. 「**共有ライブラリーあるいは DLL 名 (Shared library or dll name)**」フィールドで、共有ライブラリーあるいは DLL のプログラム名パスを指定する。
 - c. 「**DB2 の情報 (DB2 information)**」押しボタンを選択し、66ページの図13 に示されているような、DB2 情報ウィンドウを表示する。

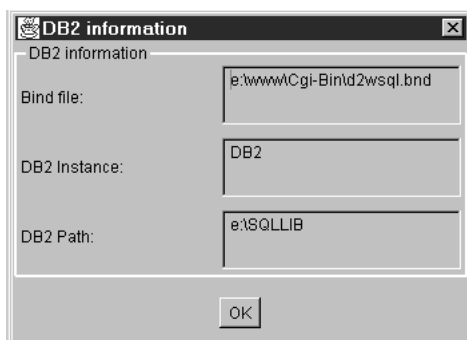


図 13. Net.Data の管理ツールの「DB2 の情報 (DB2 Information)」ウィンドウ. このページを使用して、DB2 データベースだけの情報を指定します。

DB2 の環境変数の値を以下のようにして指定します。

- 1) 「**バインド・ファイル (Bind file)**」フィールドで、バインド・ファイルのパスとファイル名を入力する。
 - 2) SQL 言語環境を使用する場合に、関連付けられているデータベースの DB2INSTANCE 値を、「**DB2 のインスタンス (DB2 Instance)**」フィールドに指定する。
 - 3) 通常、¥SQLLIB となっている、DB2 の製品の実行可能ファイルのパス・ディレクトリー名を、「**DB2 のパス (DB2 Path)**」フィールドに指定する。
 - 4) 「**OK**」をクリックして、変更を保管し、ウィンドウをクローズする。
- d. 言語環境が呼び出されるたびに言語環境に渡される、あるいは言語環境から渡される入出力パラメーターを、「**パラメーター (Parameters)**」グループ・ボックスで指定する。
- ヒント：** ユーザー自身の言語環境の定義中でない場合は、これらのフィールドを更新しないでください。
- e. クライエットの使用の有無、および言語環境と関連付けるべきクライエットを、「**Live Connection のクライエット (Live Connection cliettes)**」グループ・ボックスで指定する。
- 1) 「**Live Connection クライエットの使用 (Use Live Connection cliette)**」チェック・ボックスをチェックして、言語環境のクライエットをアクティブにするかどうかを指定する。言語環境を呼び出すときに、「**クライエット (Cliette)**」フィールドで指定されたクライエットを使用したい場合は、このチェック・ボックスを選択します。

- 2) 「**クライアント (Clette)**」フィールドで定義されている言語環境で実行させようとするクライアントの名前を指定する。名前の構文は、データベースを構成しているのか、あるいは Java アプリケーションの言語環境を定義しているのかに依存します。デフォルトは、DTW_SQL:\$(DATABASE) です。

データベースの構文：

type:name

ここで

type クライエントに対する言語環境のタイプ以下の値のうちの1 つを取ることができます。

Windows NT の場合：

DTW_ODBC, DTW_ORA, DTW_SQL, DTW_JAVAPPS

OS/2 の場合：

DTW_SQL, DTW_JAVAPPS

AIX の場合：

DTW_ODBC, DTW_ORA, DTW_SQL, DTW_JAVAPPS

name 「**クライアント (Clette)**」ページで定義されたのと同じクライアント名。デフォルトは、\$(DATABASE) です。

Java アプリケーションの構文：

DTW_JAVAPPS

4. 「**ファイル (File)**」を選択し、次に「**保管 (Save)**」を選択して、変更を保管する。
5. 管理ツールをクローズするか、別のタブをクリックして、追加の構成タスクを完了する。

言語環境を削除するには、以下を行います。

制約事項: 削除できるのは、ユーザーが定義した言語環境だけです。Net.Data に備わっている言語環境は削除できません。

1. 管理ツールを開始する。
2. 「**言語環境 (Language Environment)**」ページで、「**言語環境 (Language environment)**」リストから削除したい言語環境名を選択する。
3. 「**削除 (Delete)**」ボタンをクリックする。
4. 管理ツールをクローズするか、別のタブをクリックして、追加の構成タスクを完了する。

構成変数の定義

「変数 (Variables)」ページを使用して、Net.Data のホーム・ディレクトリーを指定し、エラー・メッセージのログ記録のレベルを選択します。

図14 は、「変数 (Variables)」ページを示しています。



図 14. Net.Data 管理ツールの「変数 (Variables)」ページ. このページを使用して、初期設定を指定します。

Net.Data のホーム・ディレクトリーを指定するには、以下を行います。

この変数は、初期設定ディレクトリー変数としても知られています。

1. 管理ツールを開始する。
2. 「変数 (Variables)」ページで、ログ・ファイルを格納したいディレクトリーのパスを、「インストール・ディレクトリー (Installation directory)」フィールドに入力する。デフォルトは、`¥inst_dir¥logs¥` です。たとえば、`e:¥db2www` となります。
3. 管理ツールをクローズするか、別のタブをクリックして、追加の構成タスクを完了する。

Net.Data のエラー・メッセージのログ記録のレベルを指定するには、以下のように行います。

1. 管理ツールを開始する。
2. 「変数 (Variables)」ページで、以下のエラーのログ記録のレベルを、「エラーのログ記録 (Error logging)」グループ・ボックスから選択する。
 - オフ
 - エラーのみ
 - 警告とエラー
3. 管理ツールをクローズするか、別のタブをクリックして、追加の構成タスクを完了する。

Net.Data がアクセスするファイルへのアクセス権の授与

Net.Data を使用する前に、Net.Data が実行されるユーザー ID が、必ず、Net.Data のマクロで参照されるファイルと、URL が参照するマクロへの適切なアクセス権を持つようにしてください。この意味は、これらのファイルは、ユーザー ID が明示的なアクセス権を持つ ディレクトリーあるいは Web サーバーが接続可能なライブラリーになければならない、ということです。

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

- Net.Data の初期設定ファイル、db2www.ini の読み取り
- Net.Data の実行可能ファイルと DLLの実行、および実行可能なファイルと DLLへのパスにあるディレクトリーの検索
- 適切な Net.Data のマクロの読み取り、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
- Live Connection の構成ファイル、dtwcm.cnf の読み取り
- キャッシュ管理プログラムの構成ファイル、CACHEMGR.CNF の読み取り
- 言語環境により参照される、外部の Perl および REXX の実行可能ファイルの読み取り

これらのファイルにアクセスを許可するための方法は、Net.Data が実行されるオペレーティング・システムに依存します。

第3章 ユーザー資産を保護する

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

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

以下の節には、ユーザーの資産保護のガイドラインが含まれています。ここで解説する機密保護のメカニズムは、次のとおりです。

- 『ファイアウォールを使用する』
- 74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』
- 74ページの『認証を使用する』
- 75ページの『許可を使用する』
- 75ページの『Net.Data のメカニズムを使用する』

ファイアウォールを使用する

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

ファイアウォールは、

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

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

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

この構成は、セキュア・イントラネットとパブリック・インターネットの両方から、Net.Data および Web サーバーを分離する、サブネットワークを作成します。このファイアウォール・ソフトウェアは、Web サーバーとパブリック・インターネット間のファイアウォール、および、Web サーバーとセキュア・イントラネット間のもう 1 つのファイアウォールを、作成するために使用されます。ここには、DB2 サーバーが含まれます。この構成を、図15 に示します。

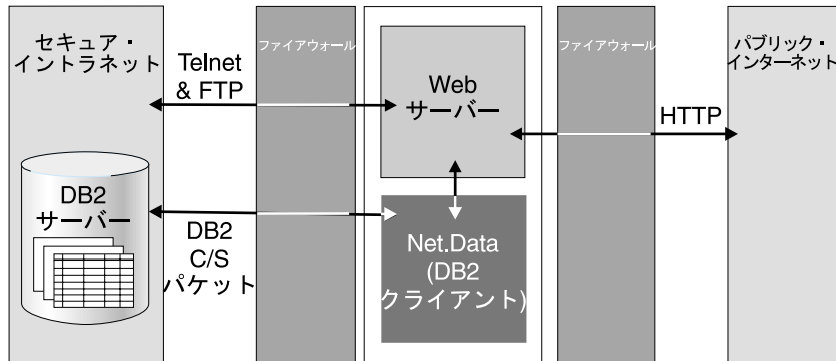


図 15. 高度な機密保護構成

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、以下によって、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。
 - Web サーバー・マシンに Client Application Enabler (CAE) をインストールする。
 - ファイアウォールが DB2 トラフィックを通すように、そのファイアウォールを構成する。1 つの方法は、パケット・フィルター規則を追加することで、Net.Data からの DB2 クライアント要求および、DB2 サーバーから Net.Data への確認パケットを許可することが可能になります。

- Web サーバーとセキュア・イントラネット間で、FTP アクセスおよび Telnet アクセスを許可します。1 つの方法は、Web サーバー・マシン上に socks サーバーをインストールすることです。
- ファイアウォール・ソフトウェアのパケット・フィルター構成ファイルでは、標準 HTTP ポートからの着信 TCP パケットが、Web サーバーにアクセスできるように指定します。また、発信 TCP 確認パケットが、Web サーバーからパブリック・インターネット上のすべてのホストに着信するように指定します。

• 中間機密保護構成

この構成では、ファイアウォール・ソフトウェアは、パブリック・インターネットから、DB2 サーバーを使用するセキュア・イントラネットを分離します。Net.Data および Web サーバーは、ファイアウォール以外はワークステーション・プラットフォーム上にあります。この構成は、最初の構成より単純ですが、それでもデータベースの保護が可能です。図16 は、この構成を示しています。

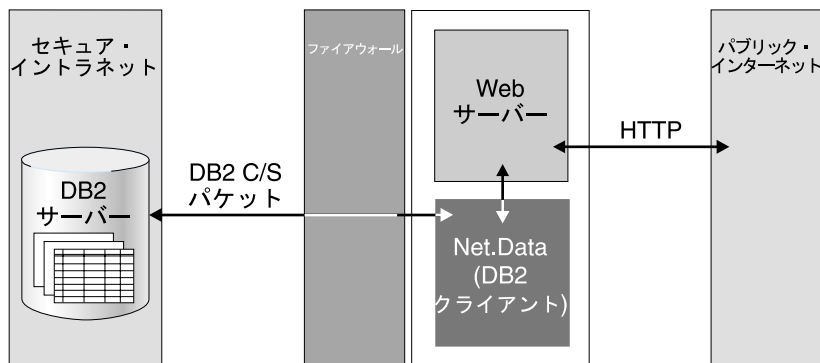


図 16. 中間機密保護構成

Web サーバーに CAE をインストールして、Net.Data が DB2 サーバーと通信することができるようにする必要があります。ファイアウォールは、DB2 クライアント要求が Net.Data から DB2 に流れ、確認パケットが DB2 から Net.Data に流れるように構成しなければなりません。

• 単純な機密保護構成

この構成では、DB2 サーバーおよび Net.Data は、ファイアウォールおよびセキュア・イントラネットの外側にインストールされます。これらの DB2 サーバーおよび Net.Data は、外部ハッキングからは保護されません。ファ

ファイアウォールには、この構成ではパケット・フィルタ規則は必要ありません。図17 は、この構成を示しています。

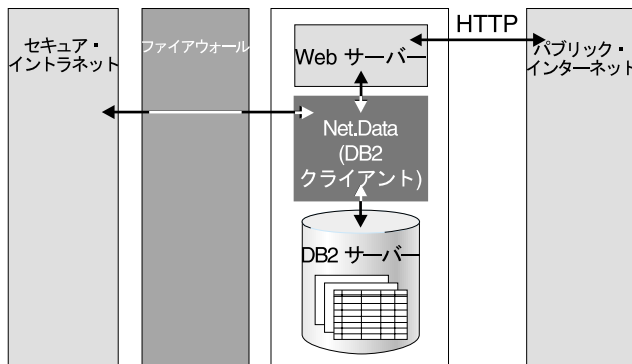


図 17. 単純な機密保護構成

ネットワーク上のユーザーのデータを暗号化する

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

認証を使用する

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

次の 2 つのタイプの認証を使用することができます。1 つはサーバーの特定のディレクトリーの保護で、もう 1 つはデータベースの保護です。

- ほとんどの Web サーバーは、保護のためにサーバー上にディレクトリーを作成できるようになっています。さらに指定したディレクトリーのファイル

をアクセスするために、ユーザー ID およびパスワードをシステムが要求するようにすることもできます。Web サーバーのシステム機能の決定に関しては、*Administrator's Guide* を参照してください。

- DB2 は、特定のユーザーだけにテーブルや列のアクセスを制限するための、データベース・アクセスの認証システムを持っています。LOGIN および PASSWORD などの Net.Data の特別な変数を使用して、DB2 認証ルーチンにリンクすることができます。

ヒント: Net.Data マクロを保護するには、以下のようにします。

1. Net.Data プログラム・オブジェクトの Web サーバー構成ファイルに、保護ディレクティブを追加する。
2. Net.Data を実行するためのユーザー ID が、マクロへのアクセス権を持っていることを確認する。アクセス権の授与に関する詳細については、69ページの『Net.Data がアクセスするファイルへのアクセス権の授与』を参照してください。

許可を使用する

許可によって、ユーザーに、オブジェクト、資源、および関数への完全なアクセス権または制限付きアクセス権が与えられます。DB2 のようなデータ・ソースは、独自に許可のメカニズムを備えていて、データ・ソースが管理する情報を保護しています。このような許可のメカニズムは、Net.Data 要求を実行しているプロセスに関連付けられているユーザー ID が、74ページの『認証を使用する』で説明されているように正しく認証されていることを前提にしています。これらのデータ・ソースに対する既存のアクセス制御メカニズムは、次に、認証されたユーザー ID によって保持されている認証に基づいて、アクセスを許可または拒否します。

Net.Data のメカニズムを使用する

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

Net.Data 構成変数

Net.Data は、エンド・ユーザーの活動を制限したり、あるいはデータベースの設計を隠したりするために使用できる、いくつかの構成変数を提供します。

パス・ステートメントでファイル・アクセスを制御する

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

また、75ページの『許可を使用する』で説明されている許可検査を使用して、77ページの『マクロ開発技法』で説明されているように、INCLUDE ステートメントのファイル名が変更できないことを確認します。

実動システムの SHOWSQL を使用不可にする

SHOWSQL 変数によって、Net.Data 関数内に指定された SQL ステートメントを、Net.Data が Web ブラウザーに表示するように指定できます。この変数は、アプリケーション内の SQL の開発およびテストで主に使用され、実動システムで使用するものではありません。

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

- Net.Data のバージョン 2.0.7 以降を使用している場合は、Net.Data の初期設定ファイルで DTW_SHOWSQL 構成変数を使用して、Net.Data マクロ内の SHOWSQL 設定の結果をオーバーライドします。構文および追加情報については、21ページの『DTW_SHOWSQL: SHOWSQL 構成変数を使用可能または使用不可にする』を参照してください。
- Net.Data のバージョン 2.0.5 以前のユーザーは、77ページの『マクロ開発技法』で説明されている、DTW_ASSIGN() 関数を使用することができます。

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

実稼働環境の直接要求を使用可能にすることが適切であるかどうかを考慮する直接要求方式で Net.Data を起動することにより、ユーザーは、SQL ステートメントや Perl、REXX、または C プログラムの実行を、URL から直接指定することができます。マクロ要求メソッドによって、ユー

ザーは、1つのマクロで定義または呼び出されたこれらの SQL ステートメントおよび関数だけを、実行することができます。

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

DTW_DIRECT_REQUEST 構成変数を使用して、直接要求を使用不可能にすることができます。構文および追加情報については、18ページの『DTW_DIRECT_REQUEST: 直接要求変数を使用可能にする』を参照してください。

マクロ開発技法

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

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

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

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

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

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

- DTW_SHOWSQL 構成変数をサポートしている Net.Data のバージョンを使用している場合、Net.Data 初期設定ファイルでこの変数を使用して、Net.Data マクロ内の SHOWSQL 設定をオーバーライドし

ます。構文および追加情報については、21ページの

『DTW_SHOWSQL: SHOWSQL 構成変数を使用可能または使用不可にする』を参照してください。

- 上記例の説明に従って DTW_ASSIGN() 関数を使用して、SHOWSQL の値を割り当ててオーバーライドされないようにします。

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

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

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

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

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

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

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

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

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

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

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

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

例 2: DTW_TRANSLATE() を使用する

アプリケーションが、入力変数 num_orders で指定された値が整数かどうかを検査する必要があるとします。これを行うための 1 つの方法は、数字 0-9 を除くすべてのキーボード文字を含む変換テーブル、translation_table を作成し、以下のように DTW_TRANSLATE および DTW_POS ストリング関数を使用して、その入力の妥当性検査を行うことです。

```
@DTW_TRANSLATE(num_orders, "x", trans_table, "x", string_out)  
  
@DTW_POS("x", string_out, result)  
  
    %IF (result = "0")  
  
        %{ continue with normal processing %}  
  
%ELSE
```

```
%{ perform some sort of error processing %}
```

```
%ENDIF
```

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

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

Net.Data 変数を使用して INCLUDE ステートメントでファイル名の値を指定すると、インクルードされるファイルは、INCLUDE ファイルが実行されるまで決定されません。マクロ内でこの変数の値をセットして、さらに、ユーザーがブラウザーでマクロが指定する値をオーバーライドできないようにする場合には、DEFINE の代わりに DTW_ASSIGN を使用して変数の値を設定する必要があります。ユーザーがブラウザーでファイル名の値を提供できるようにする場合には、ユーザーのマクロは、提供された値の妥当性検査を行う必要があります。

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

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

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

```
%INCLUDE "$(filename)"
```

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

Net.Data ストリング関数は、指定されたファイル名がそのアプリケーションに適切であることを検証するために、使用することができます。たとえば、以下のロジックを使用して、ファイル名の変数と関連付けられた入力値に、ストリング "." が含まれないことが確認できます。

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

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

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

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

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

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

- 特定の行の中の順序情報と関連付けられた顧客を識別する、順序情報のテーブルに、列を追加します。
- SQL SELECT ステートメントを変更して、ブラウザーでユーザーが提供した認証された顧客 ID によって、SELECT が限定されていることを確認します。

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

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

Net.Data 隠し変数を使用する

Web ブラウザーで HTML ソースを見ることがあるユーザーから、`Net.Data` マクロのいろいろな特性を隠すために、`Net.Data` 隠し変数を

使用することができます。たとえば、データベースの内部構造を隠すことができます。隠し変数の詳細については、133ページの『隠し変数』を参照してください。

ユーザーからの検証情報を要求する

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

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

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

第4章 Net.Data を起動する

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

- 共通ゲートウェイ・インターフェース (CGI)
- FastCGI
- Lotus Domino Go Web server (GWAPI)
- Netscape Server (NSAPI)
- Microsoft Internet Server (ISAPI)
- Java サブレット

これらのインターフェースに対する Net.Data の構成については、5ページの『第2章 Net.Data の構成』を参照してください。デフォルトでは、Web サーバーは Net.Data を CGI プログラムとして起動し、個々の Net.Data 要求を別々の新規プロセスで実行します。Web サーバーの構成時に Net.Data を起動する方法は、ユーザーが決定します。

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

- 『起動要求の型』
- 96ページの『Web サーバー API による Net.Data の起動』
- 99ページの『Java サブレットおよび JavaBeans とともに Net.Data を起動する』

起動要求の型

Net.Data を起動するときに使用するメソッドとは関係なく、指定可能な要求は2種類あります。

マクロ要求

Net.Data が指定されたマクロを実行するように、指定します。

直接要求

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

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

図18 は、マクロ要求と直接要求の違いを示しています。マクロ要求は通常、その要求の URL 内のマクロを指定します。フォーム・データを使用することもできます。直接要求では URL 内のマクロは指定されませんが、フォーム・データを引き続き使用できます。

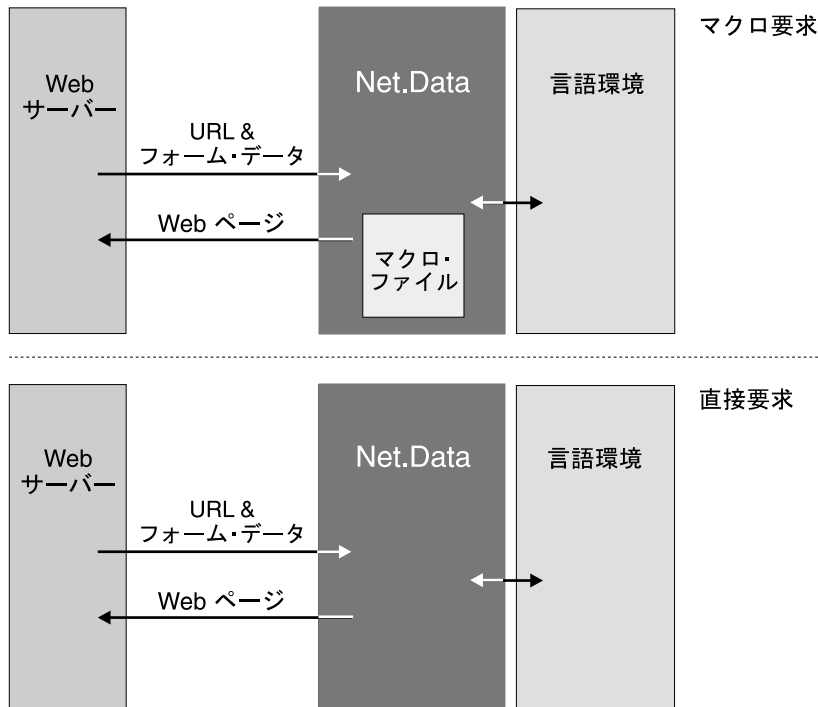


図 18. マクロ要求対直接要求

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

以下のセクションでは、これらの起動要求を詳細に説明しています。

- 『マクロで `Net.Data` を呼び出す (マクロ要求)』
- 90ページの『マクロを使用しない `Net.Data` の起動 (直接要求)』

この例では、`CGI` を使用して、`Net.Data` を起動するときに使用する構文を指定していますが、この概念は、`Net.Data` の起動に使用されるすべてのインターフェースに適用されます。インターフェースのそれぞれのタイプに必要なとされる確な構文については、それぞれのタイプに特定のセクションを参照してください。

- 96ページの『`Web` サーバー `API` による `Net.Data` の起動』
- 99ページの『`Java` サブレットおよび `JavaBeans` とともに `Net.Data` を起動する』

マクロで `Net.Data` を呼び出す (マクロ要求)

クライアントのブラウザーは、`URL` の形式で要求を送信して `Net.Data` を呼び出します。このセクションでは、`URL` 要求でマクロを指定して `Net.Data` を呼び出す方法を示します。

`Net.Data` に送信された要求は、以下の形式となります。

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

パラメーター :

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

Net.Data_invocation_path

`Net.Data` の 実行可能ファイル、サブレット・クラス、`DLL`、または共用ライブラリーのパスおよびファイル名です。たとえば、`/cgi-bin/db2www/` のようになります。

filename

`Net.Data` マクロ・ファイルの名前を指定します。 `Net.Data` は、このファイル名を検索して、`MACRO_PATH` 初期設定パス変数で定義された

パス・ステートメントとの突き合わせを試行します。詳細については、28ページの『MACRO_PATH』を参照してください。

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

?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。

この URL は、ブラウザで直接指定するか、または、URL を HTML のリンクで使用したり、以下のようなフォームを使用して作成することができます。

- HTML リンク:

```
<a href="URL">any text</a>
```

- HTML フォーム :

```
<form method=method ACTION="URL">any text</form>
```

パラメーター :

method フォームで使用される HTML メソッドを指定します。

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

例

以下の例は、Net.Data のさまざまな呼び出し方法を示しています。

例 1: HTML リンクを使用して Net.Data を呼び出す

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

例 2: フォームを使用して Net.Data を呼び出す

```
<form method=post
  action="http://server/cgi-bin/db2www/myMacro.d2w/report">
.
.
.
</form>
```

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

- 『HTML リンク』
- 『HTML フォーム』

HTML リンク

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

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

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

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

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

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

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

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

この照会は、EQPTABLE 表の中に記述された各モニターに関する型式番号、コスト、および記述情報を持つ表を戻します。この例は、照会の結果をデフォルトのレポートで表示します。REPORT ブロックを使用してレポートをカスタマイズする方法に関しては、153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、87ページの『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?</p>
<menu>
<li><input type="radio" name="hardware" value="mon" checked /> Monitors</li>
<li><input type="radio" name="hardware" value="pnt" /> Pointing devices</li>
<li><input type="radio" name="hardware" value="prt" /> Printers</li>
<li><input type="radio" name="hardware" value="scn" /> Scanners</li>
</menu>

<input type="submit" value="submit" />
</form>
```

ブラウザで選択し、「Submit (実行)」ボタンをクリックすると、Web ブラウザーは FORM タグの ACTION パラメーターを処理し、Net.Data が起動します。次に Net.Data は、equip1st.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説明については、*Net.Data* 解説書を参照してください。

Net.Data により特別処理される入力タイプが FILE 入力タイプです。この入力タイプを使用すると、ユーザーはファイルをサーバーにアップロードすることができます。アップロードされたファイルは、サーバー上の Net.Data または別のアプリケーションによってさらに処理することができます。

Net.Data は、アップロードされたファイルに対しては変換処理を行いません。それらのファイルはバイナリー・データとして扱われます。アップロードされたファイルは、DTW_UPLOAD_DIR で指定されたディレクトリーに格納され、以下の規則を使用して決定された固有の名前が付けられます。

構文：

MacroFileName + '.' + *FormVarName* + '.' + *UniqueIdentifier* + '.' + *FormFileName*

MacroFileName

要求を処理するマクロの名前（フォーム内で呼び出されるマクロ名）。ファイル名のみが使用され、完全パスは使用されません。

FormVarName

フォーム内でファイルを識別するのに使用される変数の名前。

UniqueIdentifier

一意性を持たせるために使用されるストリング。このストリングの構成は、Net.Data のプラットフォームにより異なります。たとえば、OS/400 では、以下の方法が使用されます。

YYYYMMDDHHMMSSCCC + '-' + PID + '-' + TID

ここで

YYYY = year
MM = month
DD = day
HH = hour (00-23)
SS = seconds
CCC = milliseconds
PID = process ID
TID = thread ID

例：

最初に、以下のようにして DTW_UPLOAD_DIR を Net.Data の初期設定ファイルに設定します。

DTW_UPLOAD_DIR /home/http/pub/upload

次に、以下のようにしてマクロを起動し、ファイル名とファイルをパラメーターとして渡すフォームを作成します。

```
<form method="post" enctype="multipart/form-data"
      action="/netdatadev/form.dtw/report">
  <input type="text" name="name" value="john doe" />Enter name
```

```



```

ユーザーがデフォルト値を受け入れてフォームを送信した場合、作成されたファイルは、サーバー上では次のようになります。

```
/home/http/pub/upload/form.dtw.resume.20000313112341275-6245-021.myresume.txt
```

マクロを使用しない Net.Data の起動 (直接要求)

このセクションでは、**直接要求** を使用して、Net.Data を起動する方法を示します。直接要求を使用するときには、URL にマクロの名前を指定しないでください。その代わりに、Net.Data によって定義された構文を使用して、Net.Data 言語環境、SQL ステートメントまたは実行するプログラム、および追加の必須パラメーター値を、URL で指定します。直接要求を使用可能または使用不可にする方法については、18ページの『DTW_DIRECT_REQUEST: 直接要求変数を使用可能にする』を参照してください。

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

直接要求は、URL またはフォーム・データの照会ストリングで、パラメーターを渡すことによって、Net.Data を起動します。次の例は、ユーザーが直接要求を指定するコンテキストを図示しています。

```
<a href="http://server/cgi-bin/db2www/?direct_request">any text</a>
```

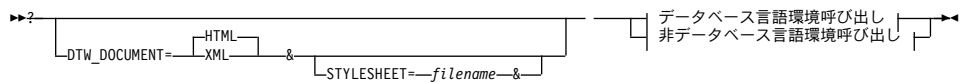
ここで *direct_request* は、直接要求の構文を表しています。たとえば、以下の HTML リンクには、直接要求が含まれています。

```
<a href="http://server /cgi-bin/db2www/?LANGENV=DTW_PERL&FUNC=my_perl(hi)">
any text</a>
```

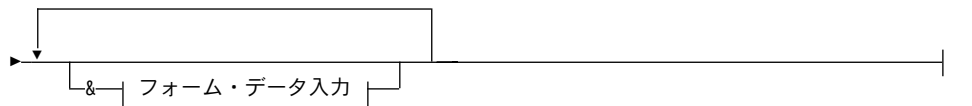
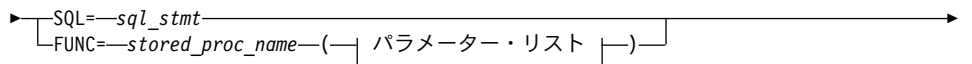
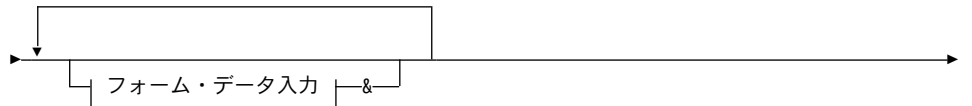
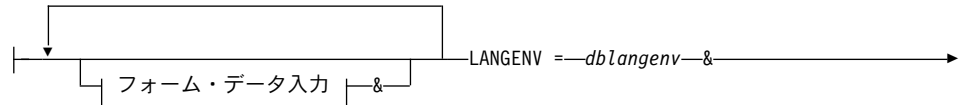
直接要求の構文

直接要求で Net.Data を起動する構文には、データベースや非データベース言語環境への呼び出しが含まれています。

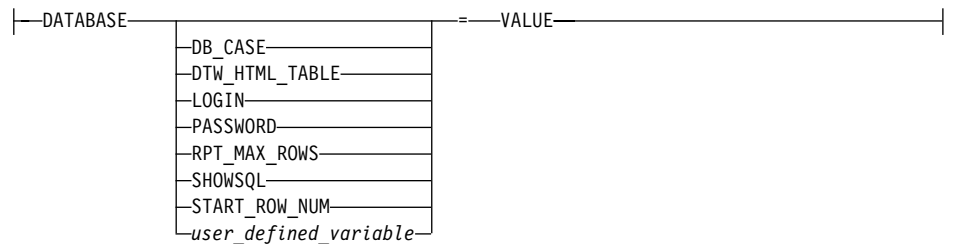
構文



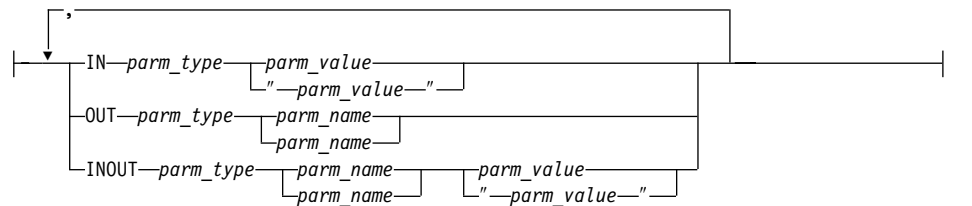
データベース言語環境呼び出し:




フォーム・データ入力:



パラメーター・リスト:



|—LANGENV=—*lang_env*—&—FUNC=—*program_name*—(——)

LOGIN

データベースのユーザー ID を指定します。

PASSWORD

データベースのパスワードを指定します。

RPT_MAX_ROWS

関数がレポートに戻すテーブルの最大行数を指定します。

SHOWSQL

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

START_ROW_NUM

関数がレポートを開始する行番号を指定します。

user_defined_variable

Net.Data に渡され、必要な情報を提供するか、あるいは Net.Data の振る舞いに影響を与える変数。ユーザー定義の変数は、ユーザーがアプリケーション用に定義する変数です。

VALUE

Net.Data 変数の値を指定します。

LANGENV

SQL ステートメントまたはストアド・プロシージャ呼び出しの、ターゲット言語環境を指定します。言語環境が、データベース言語環境の 1 つになっている場合には、そのデータベースの名前も指定されていなければなりません。

dblangenv

以下のデータベース言語環境の名前。

- DTW_SQL
- DTW_ODBC
- DTW_ORA

SQL

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

sql_stmt

動的 SQL を使用して実行される、すべての有効な SQL ステートメントを含むストリングを指定します。

FUNC

直接要求が、ストアド・プロシージャの実行を指定していることを示しています。

stored_proc_name

有効な DB2 ストアド・プロシージャを指定します。

parm_type

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

parm_name

有効なパラメーター名を指定します。

parm_value

DB2 ストアド・プロシージャに対する有効なパラメーター値を指定します。

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

INOUT

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

OUT

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

非データベース言語環境呼び出し

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

LANGENV

その関数を実行するターゲット言語環境を指定します。

lang_env

以下の非データベース言語環境の名前を指定します。

- DTW_PERL
- DTW_REXX
- DTW_SYSTEM

FUNC

直接要求がプログラムの実行を指定していることを示しています。

program_name

実行される関数を含んでいるプログラムを指定します。

parm_value

その関数に有効なパラメーター値を指定します。

直接要求の例

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

HTML リンク: 以下の例では、リンクによって `Net.Data` を起動するために、直接要求を使用しています。

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

```
<a href="http://server /cgi-bin/db2www/?LANGENV=DTW_PERL&FUNC=my_perl(hi)">
  any text</a>
```

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

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

ヒント: URL では、スペースや二重引用符などの特定の文字は符号化しなければなりません。この例では、パラメーター値の中の二重引用符とスペースは、それぞれ `%22` および `+` 文字に符号化されています。ユーザーは、組み込み関数 `DTW_URLESCSEQ` を使用して、URL で符号化の必要があるテキストをすべて符号化することができます。

`DTW_URLESCSEQ` 関数についての詳細は、*Net.Data* 解説書の説明を参照してください。

HTML フォーム: 以下の例では、フォームによって `Net.Data` を起動するために、直接要求が使用されています。

例 1: SQL 言語環境を使用して SQL 照会の実行、`CELDIAL` データベースへの接続、テーブルの照会を行う HTML フォーム

```
<form method="post"
  action="http://server /cgi-bin/db2www/">
<input type=hidden name="langenv" value="dtw_sql" />
<input type=hidden name="database" value="ceTdia1" />
  <input type=hidden name="sql"
    value="select * from table1 where col1=$(inputname)" />
Enter Customer name:
<input type=text name="inputname" value="john" />
<input type=submit />
</form>
```

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

URL: 以下の例では、URL によって Net.Data を起動するために、直接要求が使用されています。

例 1: SQL 言語環境を使用する SQL 照会を実行する URL

```
http://server /cgi-bin/db2www/?LANGENV=DTW_SQL&DATABASE=CELDIAL
&SQL=select+*+from+customer
```

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

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

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

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

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

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

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

```
http://server /cgi-bin/db2www/?LANGENV=DTW_SQL&FUNC=MY_STORED_PROC
(IN+CHAR(30)+Salaries)&DATABASE=CELDIAL
```

Web サーバー API による Net.Data の起動

Net.Data は、ご使用のオペレーティング・システムによって、以下のリストの Web API をサポートします。

GWAPI プラグイン

Lotus Domino Go Webserver API プラグイン

ISAPI プラグイン

Microsoft Internet Server API プラグイン

NSAPI プラグイン

Netscape Server API プラグイン

ご使用のオペレーティング・システムのためにサポートされる Web サーバー API を決定するには、*Net.Data* 解説書のオペレーティング・システムの参照付録を参照してください。API を使用するための Net.Data と Web サーバー

の構成方法を理解するには、49ページの『Web サーバー API と一緒に使用するための Net.Data の構成』を参照してください。

要件：

- Net.Data を GWAPI、ISAPI または NSAPI モードで実行している場合には、Web サーバーを再始動します。これによって、Web サーバーは、Net.Data を再ロードして、1 つのプロセスとして実行することができます。
- Web サーバーが API モードで Net.Data を起動した後に、初期設定ファイルを変更する場合は、Web サーバーを再始動する必要があります。Net.Data 初期設定ファイル (db2www.ini) を変更しても、効果はありません。API モードでは、Net.Data は、パフォーマンスのオーバーヘッドを削減するために、初期設定ファイルの読み取りを 1 度しか行いません。
- API モードでの実行中は、Oracle および ODBC 言語環境には Live Connection が必要です。

Web サーバー API を起動するには、以下のようにします。

GWAPI の場合:

構文：

`http://server/CGI-BIN/db2www/macro_name/block[?name=val&...]`

パラメーター：

server

サーバー名

macro_name

MACRO_PATH で指定されたディレクトリーにあるマクロの相対パス名

block

処理されるマクロの HTML ブロック名または XML ブロック名

?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：

`http://myserver/CGI-BIN/db2www/mymacro.d2w/report`

ISAPI の場合：

構文：

`http://server/server_HTML_root_directory/dll_name/macro_name/
block[?name=val&...]`

パラメーター :

server_name

サーバー名

server_HTML_root_directory

Web サーバーの HTML ルート・ディレクトリー名

dll_name

Net.Data の ISAPI .dll ファイル名、dtwisapi.dll。

macro_name

MACRO_PATH で指定されたディレクトリーにあるマクロの相対パス名

block

処理されるマクロの HTML ブロック名または XML ブロック名

?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 :

`http://myserver/scripts/dtwisapi.dll/mymacro.d2w/report`

NSAPI の場合 :

構文 :

`http://server/macro_name/block[?name=val&...]`

パラメーター :

server

サーバー名

macro_name

MACRO_PATH で指定されたディレクトリーにあるマクロの相対パス名。 .d2w などのマクロの拡張子は、 Web サーバー構成ファイルで定義する必要があります。詳しくは、49ページの『Web サーバー API と一緒に使用するための Net.Data の構成』を参照してください。

block

処理されるマクロの HTML ブロック名または XML ブロック名

`?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：

`http://myserver/mymacro.d2w/report`

Java サブレットおよび JavaBeans とともに Net.Data を起動する

サブレットは、CGI プログラムまたは Web サーバーの API プラグインに似た役割を果たす、Java クラスです。Web サーバーは、サブレットを使用して、HTML ページを動的に作成します。サブレットは独自のグラフィカル・ユーザー・インターフェースを持っていませんが、サブレットのクラスは、ローカルで、またはネットワークを通じて、動的にロードされます。また、URL アドレスを使用して (リモート)、またはクラス名によって (ローカル)、呼び出すことができます。

Net.Data は、Net.Data がサポートする Java 使用可能オペレーティング・システムの中で、Net.Data マクロの起動、Net.Data 関数の実行、または Net.Data による SQL ステートメントの実行のために、使用可能なサブレットを提供します。

本章では、次の概念と操作を説明しています。

『Net.Data サブレット』

106ページの『Net.Data JavaBeans』

Net.Data サブレット

Net.Data は、サブレットおよび NetObjects Fusion プラグインを、Java 環境で利用できる Net.Data とともに提供します。サブレットとプラグインを使用して、次のことが行えます。

- URL から、または Server-Side-Include (SSI) として、Net.Data マクロを実行する。
- URL から SSI としても、Net.Data 関数を実行する。
- NetObjects Fusion (NOF) を使用して、マクロを管理する。これによって、Web サイト・マネージメントと優れた統合性を持ち、単純なマクロを開発するための簡単に使用できるグラフィカル・ユーザー・インターフェースが提供されます。NOF プラグインの使用方法を理解するには、277ページの『付録E. Net.Data サブレットと NetObjects Fusion NOF プラグインを使用する』を参照してください。

このセクションでは、次のサーブレットのトピックを説明しています。

- 『Net.Data サーブレットについて』
- 101ページの『Net.Data サーブレットを実行する』

Net.Data サーブレットについて

Net.Data にはサーブレットがあり、Java 環境を利用したマクロの開発および管理に役立ちます。サーブレットは、CGI プログラムまたは Web サーバーの API プラグインに似た役割を果たす、Java クラスです。Java サーブレットが使用可能な Web サーバーがサーブレットを使用し、HTML ページを作成します。サーブレットは独自のグラフィカル・ユーザー・インターフェースを持っていません。ただしサーブレットのクラスは、ローカルまたはネットワークを通じて動的にロードできるため、URL アドレスを使用 (リモート)、またはクラス名によって (ローカル) 呼び出すことができます。サーブレットは、Windows NT および AIX オペレーティング・システムで使用可能です。

Net.Data サーブレットは Java ベースのラッパーで、ネイティブ DLL ファイルを使用して、Net.Data バージョン 2 のマクロまたは直接要求を実行します。これらのサーブレットによって、既存のマクロ (MacroServlet) または単一の関数 (FunctionServlet) を実行できます。Net.Data バージョン 2 は、これらのサーブレットを使用するために必要です。Net.Data サーブレットはデータベースまたは非データベースの言語環境プログラムを実行できるほか、Live Connection と一緒に実行することもできます。

サーブレットには、アプリケーションとともに使用する API が付いています。サーブレット API は Net.Data で文書化されます。API ドキュメンテーションについては、<inst_dir>/servlets/NetDataServlets.jar ファイルを参照してください。

Net.Data には次の 2 つのサーブレットがあります。

マクロ・サーブレット (com.ibm.netdata.servlets.MacroServlet)

既存の Net.Data マクロを実行します。

マクロ・サーブレットを使用することは、Java サーブレットを通じてマクロを実行する場合を除いて、CGI-BIN インターフェースを通じて Net.Data マクロを実行することに似ています。マクロ・サーブレットには、Net.Data バージョン 2、またはそれ以降のバージョンがインストールされている必要があります。

マクロ・サーブレットを使用すると、以下のような利点があります。

- SQL 照会が ODBC を介して、Live Connection Manager を使用して実行され、パフォーマンスが向上する。

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

さらに マクロ・サーブレットによって、Perl、REXX、Java のような種々の言語環境プログラムに加えて、DB2、Oracle のような異機種のデータベースに、ネイティブ・アクセスできるようになります。

関数サーブレット (com.ibm.netdata.servlets.FunctionServlet)

%FUNCTION DTW_SQL() などのサーブレット・インターフェースを介して、Net.Data 関数または SQL ステートメントを実行します。詳しくは、90ページの『マクロを使用しない Net.Data の起動 (直接要求)』を参照してください。

関数サーブレットには、Net.Data バージョン 2 がインストールされている必要があります。

Net.Data サーブレットを実行する

Net.Data サーブレットは、URL から、または HTML ファイルの SSI として、実行可能です。NetObjects Fusion プラグインを使用して、Net.Data サーブレットを NOF サイトに取り込むことができます。以下のセクションでは、サーブレットの構文に従って入力し、サーブレットを変更および実行する方法を説明します。NetObjects Fusion を用いたサーブレットの変更および実行の方法を理解するには、277ページの『付録E. Net.Data サーブレットと NetObjects Fusion NOF プラグインを使用する』を参照してください。

- 『マクロ・サーブレットを実行する』
- 103ページの『関数サーブレットを実行する』

マクロ・サーブレットを実行する: HTML ファイル内から、次の構文オプションの 1 つを使用して、サーブレット・パラメーターを入力します。

1. URL:

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

例 :

```
http://myserver/servlet/com.ibm.netdata.servlets.MacroServlet
?MACRO=my_macro&BLOCK=my_block&field1=custno
```

2. SSI:

```
<servlet code="com.ibm.netdata.servlets.MacroServlet">
  <param name="MACRO" value="macro_value">
  <param name="BLOCK" value="block_value">
  <param name="parmnn" value="valuenn">
</servlet>
```

例 :

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

パラメーター :

macro_value

既存の Net.Data マクロへの完全修飾パス

block_value

指定された Net.Data 実行マクロ内の、HTML ブロックの名前。デフォルトは report (オプション) です。

parmn

マクロが必要とする追加のパラメーター。たとえば、

```
<param name="field1" ...
```

valuenn

マクロが必要とする追加の値。たとえば、

```
... value="custnum"
```

HTMLPATH パラメーター : 指定されていない HTMLPATH パラメーターを参照するエラー・メッセージが出力された場合には、サーブレット呼び出しコマンドに HTMLPATH パラメーターを追加します。

- URL:

```
http://myserver/servlet/com.ibm.netdata.servlets.MacroServlet
?MACRO=macro_name&BLOCK=block_value&htmlpath=html_path&parmn=valuenn
```

例 :

```
http://myserver/servlet/com.ibm.netdata.servlets.MacroServlet?MACRO=my_macro
&BLOCK=my_blockhtmlpath=e:¥html&field1=custno
```

- SSI:

```
<servlet code="com.ibm.netdata.servlets.MacroServlet">
  <param name="MACRO" value="macro_value">
  <param name="BLOCK" value="block_value">
  <param name="htmlpath" value="html_path">
  <param name="parmn" value="valuenn">
</servlet>
```

例 :

```

<servlet code="com.ibm.netdata.servlets.MacroServlet">
<param name="MACRO" value="my_macro">
<param name="BLOCK" value="my_block">
<param name="htmlpath" value="e:%html">
<param name="field1" value="custno">
</servlet>

```

OUTBUFLN パラメーター : ユーザーのマクロの結果が 32 KB を超える場合は、OUTBUFLN パラメーターを指定します。これらのパラメーターが必須の場合、それを指定しないと、予期しない結果が発生することがあります。

- URL:

```

http://myserver/servlet/com.ibm.netdata.servlets.MacroServlet
?MACRO=macro_name&BLOCK=block_value
&OUTBUFLN=output_buffer_size&parmnn=valuenn

```

例 :

```

http://myserver/servlet/com.ibm.netdata.servlets.MacroServlet?MACRO=my_macro
&BLOCK=my_block&OUTBUFLN=48&field1=custno

```

- SSI:

```

<servlet code="com.ibm.netdata.servlets.MacroServlet">
  <param name="MACRO" value="macro_value">
  <param name="BLOCK" value="block_value">
  <param name="OUTBUFLN" value="output_buffer_size">
  <param name="parmnn" value="valuenn">
</servlet>

```

例 :

```

<servlet code="com.ibm.netdata.servlets.MacroServlet">
<param name="MACRO" value="my_macro">
<param name="BLOCK" value="my_block">
<param name="OUTBUFLN" value="48">
<param name="field1" value="custno">
</servlet>

```

関数サーブレットを実行する: 関数サーブレットは、直接要求を使用して Net.Data を起動し、関数 (REXX 関数など) または SQL ステートメントのいずれかを実行する場合があります。サーブレット用に指定するパラメーターは、ユーザーが関数または SQL ステートメントのどちらを実行するかによって異なります。HTML ファイル内から、次の構文オプションの 1 つを使用して、サーブレット・パラメーターを入力します。

1. URL の場合 :

- 関数を起動するには :

```

http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
?LANGENV=lang_env_name&FUNC=program_name&parmnn=valuenn

```

例 :

```
http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
?LANGENV=DTW_REXX&FUNC=my_rexx&field1=custno
```

- **SQL ステートメントを起動するには :**

```
http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
?LANGENV=database_lang_env_name&SQL=SQL_statement
&DATABASE=database_name&parmnn=valuenn
```

例 :

```
http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
?LANGENV=DTW_SQL&SQL=select+++from+myTable&DATABASE=CELDIAL
```

2. SSI:

- **関数を起動するには :**

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
  <param name="LANGENV" value="lang_env_name">
  <param name="FUNC" value="program_name">
  <param name="parmnn" value="valuenn">
</servlet>
```

例 :

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
<param name="LANGENV" value="DTW_REXX">
<param name="FUNC" value="myREXX">
<param name="field1" value="custno">
</servlet>
```

- **SQL ステートメントを起動するには :**

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
  <param name="LANGENV" value="lang_env_name">
  <param name="SQL" value="SQL_stmt_name">
  <param name="DATABASE" value="database_name">
  <param name="parmnn" value="valuenn">
</servlet>
```

例 :

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
<param name="LANGENV" value="DTW_SQL">
<param name="SQL" value="select * from employee">
<param name="DATABASE" value="CELDIAL">
</servlet>
```

パラメーター :

lang_env_name

関数、SQL ステートメント、またはストアード・プロシージャを処

理するために呼び出される、 Net.Data 言語環境 (DTW_SQL, DTW_REXX など)。このパラメーターには FUNC または SQL を使用する必要があります。

program_name

実行する関数を含むプログラムの名前。たとえば、my_rexx。ここで my_rexx は REXX 実行可能の名前です。 *parmn* および *valuenn* パラメーターで、関数の入力パラメーターを指定します。

database_name

DATABASE パラメーターに関連するデータベースの名前。 The specified

SQL_stmt_name

データベースをアクセスする SQL ステートメントまたはストアド・プロシージャ。たとえば、"select * from employee"。 *parmn* および *valuenn* パラメーターで、SQL ステートメントまたはストアド・プロシージャの入力パラメーターを指定します。

parmn

その他ユーザーのマクロに必要とされるパラメーター。たとえば、以下のようになります。 <param name="field1" ...

valuenn

その他ユーザーのマクロが必要とする値。たとえば、以下のようになります。 ... value="custnum"

HTMLPATH パラメーター：指定されていない HTMLPATH パラメーターを参照するエラー・メッセージが出力された場合には、サブレット呼び出しコマンドに HTMLPATH パラメーターを追加します。

• URL:

```
http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
?LANGENV=lang_env_name&FUNC=program_name&htmlpath=html_path
&parmn=valuenn
```

例：

```
http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
?LANGENV=DTW_REXX&FUNC=my_rexx&htmlpath=e:¥html&field1=custno
```

• SSI:

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
  <param name="LANGENV" value="lang_env_name">
  <param name="SQL" value="SQL_stmt_name">
  <param name="htmlpath" value="html_path">
  <param name="parmn" value="valuenn">
</servlet>
```

例 :

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
  <param name="LANGENV" value="DTW_SQL">
  <param name="SQL" value="select * from employee">
  <param name="htmlpath" value="e:¥html">
  <param name="field1" value="custno">
  <param name="DATABASE" value="SAMPLE">
</servlet>
```

ここで *html_path* は、Web サーバー・ルートの HTML ディレクトリーへのパスを指定します。例 : *htmlpath=e:¥html*

OUTBUFLEN パラメーター : ユーザーのマクロの結果が 32 KB を超える場合は、OUTBUFLEN パラメーターを指定します。これらのパラメーターが必須の場合、それを指定しないと、予期しない結果が発生することがあります。

- URL:

```
http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
?LANGENV=lang_env_name&FUNC=program_name&OUTBUFLEN=output_buffer_size
&parmn=valuenn
```

例 :

```
http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
?LANGENV=DTW_REXX&FUNC=my_rexx&OUTBUFLEN=48K&field1=custno
```

- SSI:

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
  <param name="LANGENV" value="lang_env_name">
  <param name="FUNC" value="program_name">
  <param name="OUTBUFLEN" value="output_buffer_size">
  <param name="parmn" value="valuenn">
</servlet>
```

例 :

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
  <param name="LANGENV" value="DTW_REXX">
  <param name="FUNC" value="my_rexx">
  <param name="OUTBUFLEN" value="48">
  <param name="field1" value="custno">
</servlet>
```

Net.Data JavaBeans

Net.Data は JavaBeans を提供します。この JavaBeans は、Web サーバーを実行しなくても、Java 環境で使用することができます。JavaBean は、オブジェクト指向プログラミング・インターフェースです。これによって、再利用可能

なアプリケーション、またはプログラム組み立てブロックを作成することができます。これらのオブジェクトは、Java 対応オペレーティング・システムのネットワークで使用されます。

JavaBean は、ネイティブ Net.Data DLL を使用して Net.Data を起動し、リターン・コードおよび Net.Data 出力 (結果) を含むストリングを生成します。JavaBeans はネイティブ DLL を使用するので、Net.Data 関数を使用するために Web サーバーを実行させる必要はありません。

設計のヒント：Net.Data JavaBeans が戻す結果は、ユーザーのマクロまたは関数が戻す内容に一致します。通常、これは HTML です。結果は HTML と互換性を持つ JavaBean に渡すことを推奨します。HTML 互換の JavaBean は、HTML を解釈し、結果を表示することができます。

JavaBeans を使用すれば、次のことが可能です。

- Net.Data マクロを実行する。
- Net.Data を通して SQL ステートメントを実行する。

このセクションでは、以下の JavaBean のトピックについて説明します。

- 『Net.Data JavaBeans について』
- 108ページの『Net.Data JavaBeans のセットアップおよび実行』

Net.Data JavaBeans について

Net.Data には JavaBeans があります。Java 環境を利用することで、マクロの開発および管理に役立ちます。JavaBeans は Java のオブジェクトで、次のインターフェースを提供しています。

- JavaBean 開発環境 (Lotus BeanMachine など) で使用する場合には、提供されているカスタマイザーを必要な構成要素と一緒に組み合わせて、マクロまたは SQL ステートメントの結果の処理と表示を行い、Java アプレットを生成します。
- API を使用する場合には、JavaBeans を使用して、ユーザー独自の Java アプレットまたはアプリケーションに、Net.Data の機能性を提供します。API ドキュメンテーションは、<inst_dir>/beans/NetDataBeans.jar にあります。

Net.Data には次の 2 つの型の JavaBeans があります。

Net.Data マクロ JavaBean

Net.Data を通して既存の Net.Data マクロを実行するために、Java ベースのインターフェースを提供します。

Net.Data SQL JavaBean

Net.Data を通して既存の SQL ステートメントを実行するために、Java ベースのインターフェースを提供します。

Net.Data JavaBeans は、ネイティブ DLL ファイルを使用して Net.Data を通して実行する、Java ベースのラッパーです。両方とも Net.Data バージョン 2 またはそれ以降のバージョン、および JDK バージョン 1.1 またはそれ以降のバージョンが、インストールされている必要があります。

Net.Data JavaBeans のセットアップおよび実行

このセクションでは、Bean Machine などの JavaBean 開発ツールを使用した、Net.Data JavaBeans のセットアップおよび実行の方法を説明しています。開発ツールを使用するためのステップは、特に決まった規則はありません。したがってツールはユーザーが選択することができます。

- 『マクロ Bean』
- 109ページの『SQL Bean』

マクロ Bean: Net.Data Macro bean、com.ibm.netdata.beans.NetDataMacro によって、Java を使用することで、既存のマクロを実行できるようになります。この Bean を使用するには、Bean の Net.Data プロパティを指定する必要があります。指定するとマクロが使用できるようになります。

JavaBean 開発ツールとともに、Net.Data マクロ JavaBean をセットアップするには:

1. <inst_dir>/beans/NetDataBeans.jar ファイルを、JavaBean 開発ツールに追加またはインポートする。
2. 開発ツールのカスタマイザー・インターフェースを使用して、次の入力プロパティを設定する。

マクロ 実行する既存のマクロの名前を指定します。たとえば、MyMacro.mac。

ブロック

実行する HTML ブロック・セクションの名前を指定します。デフォルトは report です。

HTML パス

Net.Data db2www.ini ファイルへのパスを指定します。

パラメーター

マクロの実行時に使用するパラメーターの名前と値を指定します。

構文 :

name1=value1&nameN=valueN

JavaBean 開発ツールで Net.Data マクロ JavaBean を実行するには:

1. 「実行 (run)」を選択するか、JavaBean 開発ツールによって提供されているアクションを実行して、マクロを実行します。
2. マクロが実行されると、ユーザーは次の出力プロパティを参照できます。

RC Net.Data から戻されるリターン・コードを指定します。

結果 Net.Data マクロの実行によって戻されたデータを指定します。

SQL Bean: Net.Data SQL bean、com.ibm.netdata.beans.NetDataSQL によって、Java を使用し、Net.Data を介して SQL ステートメントを実行できるようになります。この Bean を使用するには、Bean の Net.Data プロパティを指定する必要があります。指定するとマクロが使用できるようになります。

JavaBean 開発ツールとともに、Net.Data SQL JavaBean をセットアップするには

1. NetDataBeans.jar ファイルを、ユーザーの JavaBean 開発ツールに追加またはインポートする。
2. 開発ツールのカスタマイザー・インターフェースを使用して、次の入力プロパティを設定する。

言語環境

使用する言語環境を指定します。デフォルトは、DTW_SQL です。

SQL 実行する SQL ステートメントを指定します。デフォルトは select * from employee です。

DATABASE

使用するデータベースを指定します。デフォルトは、SAMPLE です。

HTML パス

Net.Data db2www.ini ファイルへのパスを指定します。

パラメーター

SQL ステートメントの実行時に使用するパラメーターの名前と値を指定します。

構文 :

name1=value1&nameN=valueN

JavaBean 開発ツールで、Net.Data SQL JavaBean を実行するには :

1. 「実行 (run)」を選択するか、JavaBean 開発ツールによって提供されているアクションを実行して、マクロを実行します。

2. SQL ステートメントが実行されると、ユーザーは次の出力プロパティを参照できます。

RC Net.Data から戻されるリターン・コードを指定します。

結果 SQL ステートメントから戻されるデータを指定します。

第5章 Net.Data のマクロ開発

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

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

Net.Data のマクロには、図19 に示されているように、宣言パーツと表示パーツという 2 つの構成部分があります。

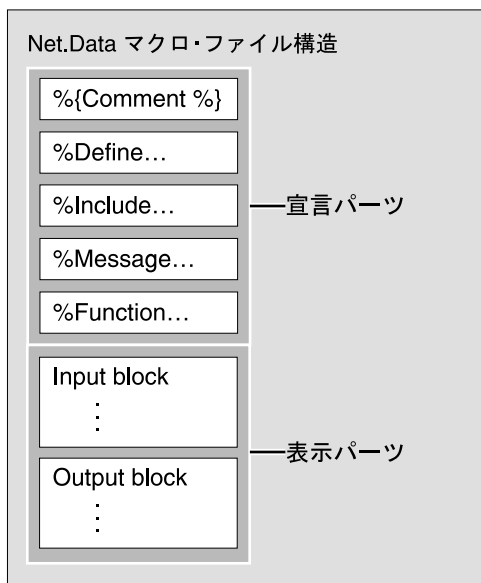


図 19. マクロ構造

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

これらのパーツは、複数回、任意の順序で使用することができます。マクロのパーツ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: *Net.Data* が実行するもとのユーザー ID がこのファイルを読み取る権限を持つようにしてください。詳しくは、69ページの『*Net.Data* がアクセスするファイルへのアクセス権の授与』を参照してください。

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

- 『*Net.Data* マクロの分析』
- 123ページの『*Net.Data* のマクロ変数』
- 139ページの『*Net.Data* の関数』
- 151ページの『ドキュメント・マークアップの生成』
- 160ページの『マクロにおける条件付き論理とループ』

Net.Data マクロの分析

マクロは、以下の 2 つのパーツで構成されています。

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

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

許可のためのヒント: *Net.Data* が実行するもとのユーザー ID が、EXEC ステートメントによって参照されるファイルを読み取って実行する権限、および、INCLUDE ステートメントによって参照されるファイルを読み取る権限を持つようにしてください。詳しくは、69ページの『*Net.Data* がアクセスするファイルへのアクセス権の授与』を参照してください。

- 表示パーツは、マクロからのエン트리・ポイント、およびエグジット・ポイントとして使用される HTML ブロックを使用して、Web ページのレイアウト、参照変数、および呼び出し関数を定義します。*Net.Data* を起動する際に、HTML ブロック名をマクロ処理のエン트리・ポイントに指定します。HTML ブロックは、116ページの『HTML ブロック』で説明されています。

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

まず最初に、マクロ全体を見渡し、次に各ブロックを詳細に見てみましょう。

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

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

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

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

<form method="post" action="output">
REXX プログラムに渡すデータを入力します。
<input name="input_data" type="text" size="30" />
<p>
<input type="submit" value="enter" />
</p>
</form>

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

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

```

<h1>Output Page</h1>
<p>@rex1(input_data)
<p><hr>
<p>[<a href="/">Home page</a> |
<a href="input">Previous page</a>]
</body></html>
%}

```

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

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

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

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

DEFINE ブロック

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

```

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

```

1 行目はコメントです。コメントは、%{ と %} で囲まれた任意のテキストです。コメントは、マクロの任意の場所に置くことができます。その次のステートメントが、DEFINE ブロックの始まりです。1 つの定義ブロックに、複数の変数を定義することができます。この例では、page_title という 1 つの変数だけが定義されています。変数の定義を行うと、\$(page_title) という構文を

使用して、この変数をマクロの任意の場所で参照することができます。変数を使用すると、後でマクロを全体にわたって変更することが容易にできるようになります。このブロックの最後の行 `%` は、`DEFINE` ブロックの終了を確認するものです。

FUNCTION ブロック

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

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

```
%{ ***** FUNCTION Block *****%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result) { <-- この関数は 1 つの
                                                         パラメーターを受け取り、
                                                         外部 REXX プログラムで
                                                         割り当てられた変数
                                                         'result' を返します。
                                                         %EXEC{ompsamp.cmd %} <-- 関数は "ompsamp.cmd" という外部 REXX プログラム
                                                         を実行します。
%}

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

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

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

この関数のコードは、その関数に渡されたデータをそのまま返します。この結果のテキストは自分の好きなようにフォーマットすることができます。それには、`@rexx1()` 関数呼び出し要求を、通常のマークアップ・スタイルのタグでくくります (たとえば、`` 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す。

2 番目の関数ブロック `today` も、REXX プログラムを参照します。しかし、この場合の REXX プログラム全体は、関数宣言それ自身に含まれています。外部プログラムは必要ありません。インライン・プログラムは、REXX および Perl の関数では許可されています。その理由は、これらのプログラムは、動的な解析および実行ができる、インタープリター言語だからです。インライン・プログラムは、管理すべき独立したプログラムの必要がないので、単純であるという優位点を持っています。最初の REXX の関数は、インラインとして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` マクロ要求内に指定され、それぞれのマクロは少なくとも 1 つの HTML ブロックを持つ必要があります。

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

```
%{ *****                                HTML Block: Input      *****%}
%HTML (INPUT) {                                <--- この HTML ブロックの名前を識別します。
<html>
<head>
<title>$(page_title)</title>  <--- 変換を置換します。
</head><body>
<h1>Input Form</h1>
Today is @today()                <--- この行では関数の呼び出しをします。

<form method="post" action="output">  <--- このフォームがサブミットされると、
                                         "OUTPUT" HTML ブロックが呼び出され
                                         ます。<p>

REXX プログラムに渡すデータを入力します。
<input name="input_data"              <--- "input_data" はフォームがサブミットされる
TYPE="text" SIZE="30" />              ときに定義され、このマクロ内ならどこでも参照
                                         できます。これは、入力フィールドに入れられ
                                         たユーザー・タイプが何であっても、そのタイ
                                         プに初期化されます。

</p>
<input type="submit" value="enter" />

<hr>
<p>
[
```



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

<--- HTML ブロックを閉じます。

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

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

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

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

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

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

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

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

```
%{ ***** HTML Block: Output *****%}
%HTML(OUTPUT) {
<html>
<head>
<title>$(page_title)</title> <--- 再度、置換しています。

</head><body>
<h1>Output Page</h1>
<p>@rex1(input_data) <--- この行では、関数 rex1 を呼び出し、
                           引数 "input_data" を渡します。

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

HTML(INPUT) ブロックと同じように、このブロックも、変数および関数呼び出しを置換するための Net.Data マクロ・ステートメントを持つ、標準の HTML です。再度、page_title 変数はこの title ステートメントに置換されます。そして、前と同じように、このブロックは関数呼び出しを含みます。この場合、このブロックは、関数 rex1 を呼び出し、変数 input_data の内容をこの関数に渡します。この変数は、このブロックが Input ブロックで定義されたフォームから受け取ったものです。任意の数の変数を関数に渡したり、関数から受け取ったりすることができます。関数定義は、渡される変数の数および使用法を指定します。

XML ブロック

その他の処理アプリケーションまたは Microsoft Internet Explorer バージョン 5 などのクライアント・ブラウザへ XML を送達したい場合は、XML コンテンツを送達する XML ブロック構造を使用することができます。

XML ブロックは、HTML ブロックと同じように機能します。つまり、マクロのエントリー・ポイントまたはエグジット・ポイントです。ブロック内で XML タグを直接入力し、変数を参照し、関数呼び出しを行うことができます。XML ブロック内で関数呼び出しを行うと、Net.Data は、HTML の代わりに XML タグを使用して結果を表示します。

生成された XML ドキュメントは、ユーザーのニーズに合わせてカスタマイズできるため、XML ブロックは prolog タグを生成しません。自分の企業に独自の prolog 情報を入力し、選択したスタイルシートを組み込んでください。使用できる XSL スタイルシートは、Net.Data とともに 3 枚含まれています。これらのスタイルシートには、Net.Data によって生成されたすべての XML および HTML のエレメントの変形が含まれています (生成された HTML エレメン

トは新規の XHTML 規格に準じます)。ただし、これらのスタイルシートはサンプルであり、これらを拡張してユーザー独自のものを作成することをお勧めします。

```
%DEFINE DATABASE = "SAMPLE"

%FUNCTION(DTW_SQL) NewManager(){
select * from staff where job = 'Mgr' and years <= 5
%}

%XML(report){
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="ndTable.xsl" ?>

<XMLBlock>
  <h1>List of New Managers</h1>
  @NewManager()
</XMLBlock>
%}
```

図 20. XML レポート・ブロックを含むマクロ

Net.Data は、小さなエレメントのセットを使用して XML ブロック内に情報を生成します。以下にそのドキュメント記述を示します。

```
<!------->
<!-- The root element used in all XML output generated -->
<!-- by Net.Data is the XMLBlock element. -->
<!------->
<!ELEMENT XMLBlock (RowSet|ShowSQL|Message)*>
<!-- ATTLIST XMLBlock name CDATA #IMPLIED>

<!------->
<!-- The default presentation format for tables uses -->
<!-- the RowSet, Row, and Column elements. -->
<!------->
<!ELEMENT RowSet (Row)*>
<!-- ATTLIST RowSet name CDATA #IMPLIED>
<!ELEMENT Row (Column)*>
<!-- ATTLIST Row name CDATA #IMPLIED
number CDATA #IMPLIED>
<!ELEMENT Column (#PCDATA)>

<!------->
<!-- SQL statements resulting from setting the SHOWSQL -->
<!-- variable are presented with the ShowSQL element. -->
<!------->
<!ELEMENT ShowSQL (#PCDATA)>

<!------->
```

```

<!-- SQL statements resulting from setting the SHOWSQL -->
<!-- variable are presented with the ShowSQL element. -->
<!------->
<!ELEMENT Message (#PCDATA)>

```

各エレメントの定義は、以下のとおりです。

XMLBlock

XML ブロックのデータ表示部分を含みます。このタグは、手動で入力しなければなりません。

RowSet

Net.Data によって生成されます。結果セットの行を含みます。RowSet の名前属性は、以下のように決定されます。

- テーブル変数の参照には、テーブル変数の名前が使用されます。
- SQL 照会を実行する関数への呼び出しから入手された結果セットには、関数の名前が使用されます。
- ストアード・プロシージャを実行する関数への呼び出しから入手された結果セットには、結果セットの名前が使用されます。結果セットが 1 つだけ戻され、結果セット・パラメーターがない場合は、関数名が使用されます。

例:

以下の関数が XML ブロックで呼び出された場合、その関連する RowSets が以下のように表示されます。

```

%FUNCTION(DTW_SQL) call_sp() {
CALL stored_proc
%}

%XML(report){
...
<RowSet name="call_sp" number="1">...</RowSet>
<RowSet name="call_sp" number="2">...</RowSet>
...
%}

```

Row Net.Data によって生成されます。1 つの行には複数の列が含まれており、識別のために番号が付けられています。

Column

Net.Data によって生成されます。特定の行のデータ値および、名前が付けられるもとなった列が含まれます。

以上のエレメントを使用して、Net.Data は、119ページの図20 にリストされたマクロから以下の出力を生成します。

Content-type: text/xml

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="ndTable.xsl" ?>
<XMLBlock>
  <h1>List of New Managers</h1>
  <RowSet name="NewManager">
    <Row number="1">
      <Column name="ID">30</Column>
      <Column name="NAME">Marenghi</Column>
      <Column name="DEPT">38</Column>
      <Column name="JOB">Mgr</Column>
      <Column name="YEARS">5</Column>
      <Column name="SALARY">17506.75</Column>
      <Column name="COMM"></Column>
    </Row>
    <Row number="2">
      <Column name="ID">240</Column>
      <Column name="NAME">Daniels</Column>
      <Column name="DEPT">10</Column>
      <Column name="JOB">Mgr</Column>
      <Column name="YEARS">5</Column>
      <Column name="SALARY">19260.25</Column>
      <Column name="COMM"></Column>
    </Row>
  </RowSet>
</XMLBlock>
```

122ページの図21 と 123ページの図22 では、Net.Data とともに提供された 2 つのスタイルシート ndTable.xsl および ndRecord.xsl を使用して、上記のデータが、ブラウザ内にどのように表示されるかを示しています。

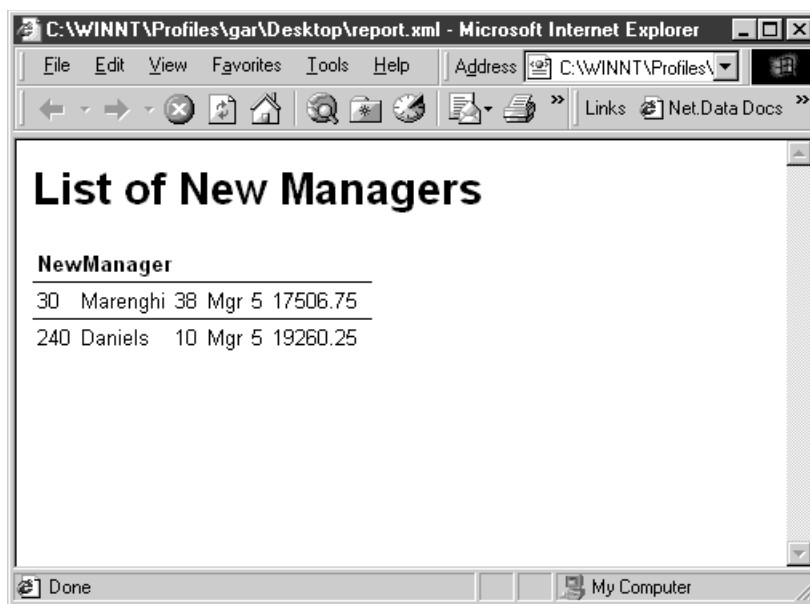


図 21. ndTable.xml スタイルシートを使用して表示された XML

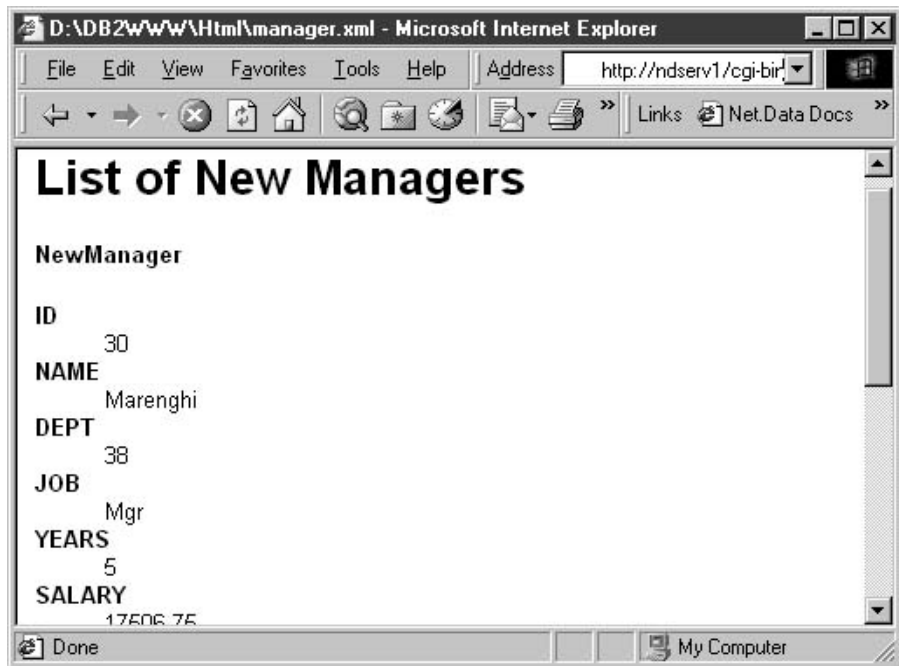


図 22. ndRecord.xml スタイルシートを使用して表示された XML

Net.Data のマクロ変数

Net.Data により、Net.Data マクロの変数を定義したり、参照したりすることができます。さらに、これらの変数を、マクロから言語環境に渡したり、言語環境から受け取ったりすることができます。渡される変数名、値、およびリテラル・ストリングは、トークンと呼ばれます。Net.Data では、トークンのサイズに制限を設けないため、システムのメモリーが処理できるすべてのトークンを渡します。ただし、言語環境プログラム環境ではトークンのサイズに制限があります。

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

- DEFINE ブロックで DEFINE ステートメントを使用して、明示的に定義された変数。
- 事前定義の変数。これは、Net.Data により使用可能になり、ある値が設定されます。通常、この値は変更できません。
- 暗黙的に定義された変数。これには、以下の 4 つのタイプがあります。

- 明示的には定義されていないが、最初に値を割り当てる際にインスタンス化される変数。
- FUNCTION ブロック定義の一部で、FUNCTION ブロック内でしか参照できないパラメーター変数。
- Net.Data によりインスタンス化され、フォーム・データまたは照会ストリング・データに対応する変数。
- Net.Data の表と関連付けられ、ROW ブロックあるいは REPORT ブロック内でしか参照することができない変数。

以後のセクションでは、以下について説明します。

- 『識別子の効力範囲』
- 125ページの『変数の定義』
- 128ページの『変数の参照』
- 129ページの『変数の型』

識別子の効力範囲

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

- グローバル

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

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

- マクロ

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

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

以下の場合、識別子には、関数ブロックの効力範囲があります。

- 識別子が関数定義のパラメーター・リストで宣言される場合。
同じ名前を持つ識別子が、関数定義の外にすでに存在する場合は、
Net.Data は、その関数ブロック内の関数パラメーター・リストの識別子を使用します。
- 識別子が関数ブロック内でインスタンス化され、関数呼び出しの前において宣言またはインスタンス化されない場合。

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

- **REPORT ブロック**

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

- **ROW ブロック**

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

変数の定義

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

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

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

- **DEFINE ステートメントまたはブロック**

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

```
%DEFINE variable_name ="variable value"

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

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

variable_name は、ユーザーが変数に与える名前です。変数名の初めは、必ず文字または下線にします。変数名に使用できるのは、英数字、下線、ピリオド、あるいはハッシュ (#) です。すべての変数名は、大文字小文字の区別をします。ただし、表変数の、 *N_columnName* 、および *V_columnName* は除きます。

たとえば、以下のようにします。

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

変数 *reply* は、値 *hello* をとります。

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

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

変数 *empty* は、空ストリングをとります。

変数に行の終わりなどの特殊文字が含まれる場合、値を中括弧で囲みます。

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

ストリングに引用符を組み込むためには、2 つの引用符を続けて使用します。

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

ブロック中括弧を使用することにより、引用が拡張できます。

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

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

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

- **HTML フォーム・タグ : SELECT、INPUT、および TEXTAREA**

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

```
<input name="variable_name" TYPE=... />
```

あるいは

```
<select name="variable_name">
    <option>value one
    <option>value two
</select>
```

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

```
<textarea name="variable_name" ROWS="4">
Please type the multi-line value
of your variable here.
</textarea>
```

variable_name は、その変数に与えた名前です。そして、その変数の値は、フォームで受け取った入力から決定されます。Net.Data のマクロにおけるこのような変数定義のタイプの使用法の例については、87ページの『HTML フォーム』を参照してください。

- **照会ストリング・データ**

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

```
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 が照会ストリングから受け取った追加データを指定します。Net.Data は、そのデータをフォーム・データからのデータと同様に受け取り処理します。

変数の参照

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

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

`Net.Data` が変数参照を検出すると、`Net.Data` は、その変数参照を変数の値で置き換えます。変数参照は、ストリング、変数参照、および関数呼び出しを含むことができます。

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

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

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

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

例 1: リンクにおける変数参照

変数 `homeURL` を定義した場合:

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

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

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

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

例 2: 動的に生成される変数参照

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

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

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

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

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

例 3: ネストされた変数参照および関数呼び出しを持つ変数参照

```
%define my = "my"
%define u = "lower"
%define myLOWERvar = "hey"

$($(my)@dtw_ruppercase(u)var)
```

変数参照は、値 hey を戻します。

変数の型

マクロでは以下の型の変数が使用できます。

- 130ページの『条件変数』
- 130ページの『環境変数』
- 131ページの『実行可能な変数』
- 133ページの『隠し変数』
- 134ページの『リスト変数』

- 135ページの『表変数』
- 136ページの『各種変数』
- 136ページの『表処理変数』
- 137ページの『レポート変数』
- 138ページの『言語環境変数』

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

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

条件変数

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

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

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

```
%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、134ページの『リスト変数』を参照してください。

環境変数

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

環境変数を定義するための構文は以下のとおりです。

```
%DEFINE var=ENVVAR
```

`var` は、定義される環境変数の名前です。

たとえば、変数 `SERVER_NAME` は、以下の環境変数として定義することができます。

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

そして、参照は以下のように行います。

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

出力は以下のようになります。

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

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

実行可能な変数

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

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

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

`runit` は、実行可能な変数になります。

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

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

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

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

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

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

1. Net.Data の初期設定ファイルの EXEC_PATH で指定されるディレクトリーを検索する。詳細については、25ページの『EXEC_PATH』を参照してください。
2. プログラムが見つからない場合は、システムの PATH 環境変数、あるいはライブラリー・リストで定義されるディレクトリーを、システムが検索する。実行可能なプログラムが見つければ、Net.Data は、そのプログラムを実行します。

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

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

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

この次の例では、システム日付をレポートの一部として戻します。

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

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

%HTML(report){
<h1>Report made: $(tstamp) </h1>
@myQuery()
%}
```

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

隠し変数

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

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

リスト変数

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

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

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

```
%HTML(INPUT) {  
<form method="post" action="/cgi-bin/db2www/example2.d2w/report">  
<h2>Select one or more cities:</h2>  
<input type="checkbox" name="conditions" value="$(cond1)" />Sao Paolo<br />  
<input type="checkbox" name="conditions" value="$(cond2)" />Seattle<br />  
<input type="checkbox" name="conditions" value="$(cond3)" />Shanghai<br />  
<input type="submit" value="submit query" />  
</form>  
%}  
  
%DEFINE{  
DATABASE="custcity"  
%LIST " OR " conditions  
cond1="cond1='Sao Paolo'"  
cond2="cond2='Seattle'"  
cond3="cond3='Shanghai'"  
whereClause= ? "WHERE $(conditions)"  
%}  
  
%FUNCTION(DTW_SQL) mySelect() {  
SELECT name, city FROM citylist  
$(whereClause)  
%}  
  
%HTML(REPORT) {  
@mySelect()  
%}
```

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

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

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

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

表変数

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

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

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

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

表を定義すると、表はゼロの行とゼロの列をとります。表に値を代入するには、表を `OUT` または `INOUT` パラメーターとして関数に渡すか、`Net.Data` によって提供される組み込み表関数を使用する以外に方法はありません。

`DTW_SQL` 言語環境は、`SELECT` ステートメントの結果を表に自動的に書き込みます。

`DTW_REXX` または `DTW_PERL` など、非データベース言語環境の場合、言語環境もまた表の値を設定する責任を負います。ただし、言語環境スクリプトまたはプログラムは、表の値をセルごとに定義します。言語環境による表変数の使用方法に関する詳細については、165ページの『第6章 言語環境の使用』を参照してください。

表変数の名前を参照することにより、関数間で表を渡すことができます。表の個々の要素は、関数の `REPORT` ブロックで、あるいは `Net.Data` 表関数の使用によって、参照することができます。`REPORT` ブロック内の表の各要素へのアクセスに関しては136ページの『表処理変数』を、表関数を使用して表の各要素にアクセスすることについては150ページの『表関数』を、参照してください。表変数は、通常、`SQL` 関数の値が代入され、次に、`SQL` 関数あるいは別の関数のいずれかにパラメーターとして渡された後、その関数におけるレポートへの入力として使用されます。表変数を、`IN`、`OUT`、あるいは `INOUT` パラメーターとして、任意の非 `SQL` 関数に渡すことができます。表は、`SQL` 関数には、`OUT` パラメーターとしてのみ渡すことができます。

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

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

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

各種変数

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

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

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

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

```
%REPORT {
  <p>This file is <i>$(DTW_CURRENT_FILENAME)</i>.</p>
}
```

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

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

表処理変数

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

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

行、またはフィールドで参照することができます。また、処理されている行の数に関する情報、あるいはすべての列名のリストにアクセスすることができます。

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

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

```
%REPORT {  
  <p>Column 1 is <i>$(N1)</i>.</p>  
}
```

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

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

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

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

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

レポート変数

Net.Data は、マクロにより生成された Web ページ出力を、デフォルトのレポート・フォーマットで表示します。HTML ブロックでは、デフォルトのレポート・フォーマットは <pre> </pre> タグを使用してテーブルを表示します。XML ブロックでは、<RowSet>、<Row>、および <Column> のタグが使用されます。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・フォーマットをオーバーライドし、表出力を HTML 形式または固定幅文字で表示するかどうかを指定して、その他の表示機能を指定します。たとえば、ALIGN 変数を使用すると、表処理変数に対して前後のスペースを制御することができます。以下の例では、ALIGN 変数を使用して、照会により戻されるリストの各列名をスペースで区切ります。

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

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

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

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

```
%DEFINE DTW_HTML_TABLE="YES"

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

言語環境変数

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

これらの変数を使用することにより、データベースへの接続の確立、Java アプレットの代替テキストの提供、NLS サポートの使用可能化、および、SQL ステートメントが正常に実行されたか判別するなどのタスクが実行できます。

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

```
%FUNCTION (DTW_SQL) val1() {
  select * from customer
%REPORT {
...
%ROW {
```

```
...
%}
SQLSTATE=$(SQL_STATE)
%}
```

この次の例では、アクセスすべきデータベースを定義する方法を示しています。

```
%DEFINE DATABASE="CELDIAL"
```

Net.Data の関数

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

ユーザー定義の関数

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

Net.Data 組み込み関数

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

次のセクションでは、以下のトピックについて説明します。

- 『関数の定義』
- 145ページの『関数の呼び出し』
- 146ページの『Net.Data 組み込み関数の呼び出し』

関数の定義

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

FUNCTION ブロック

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

MACRO_FUNCTION ブロック

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

処理されるサブルーチンを定義します。 **MACRO_FUNCTION** ブロックには、 **HTML** ブロックで使用可能ないずれのステートメントも含むことができます。

構文：関数を定義するには、以下の構文を使用してください。

FUNCTION ブロック:

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

MACRO_ FUNCTION ブロック :

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

ここで、

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

function-name

FUNCTION または **MACRO_FUNCTION** ブロックの名前を指定します。関数呼び出しは、@ 記号を前に付けた *function-name* を指定します。詳細については、145ページの『関数の呼び出し』を参照してください。

複数の **FUNCTION** または **MACRO_FUNCTION** ブロックを同じ名前で定義し、それらの同時に処理することができます。各ブロックは、すべて、同じパラメーター・リストを持たなければなりません。

Net.Data が関数を呼び出すと、同じ名前を持つすべての **FUNCTION** ブロック、または同じ名前を持つ **MACRO_FUNCTION** ブロックは、**Net.Data** のマクロにおける定義順に実行されます。

usage パラメーターが、入力 (IN) パラメーターか、出力 (OUT) パラメーターか、それとも両方のタイプ (INOUT) かを指定します。この指定は、

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

datatype

パラメーターのデータ型。言語環境プログラムの中には、渡されるパラメーターのデータ型を予想するものがあります。たとえば SQL 言語環境は、ストアード・プロシージャを呼び出す際に、それらを予想します。使用中の言語環境がサポートするデータ型についての詳細は、165ページの『第6章 言語環境の使用』を参照してください。

parameter

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

return-var

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

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

executable-statements

変数の置換および関数処理の後で、処理のための指定された言語環境に渡される言語ステートメントのセット。 *executable-statements* には、`Net.Data` の変数参照および `Net.Data` の関数呼び出しを含むことができます。 *executable-statements* には、`HTML` ブロックで許可される実行可能ステートメントが組み込まれます。

`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、131ページの『実行可能な変数』を参照してください。

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

report-block

FUNCTION ブロックまたは **MACRO_FUNCTION** ブロックの出力処理のため、1 つ以上の **REPORT** ブロックを定義します 153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。143ページの『メッセージ・ブロック』を参照してください。

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

関数での特殊文字の使用

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

たとえば、**Perl** 関数は **COMMENT** ブロック区切り文字 **%{** を使用するかもしれませんが、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持たずに、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});
    }
    ...
}%}
```

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

- %EXEC ステートメントを使用する。

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

- 変数参照を使用して、`%{` 文字を指定する。

```
%define percent_openbrace = "%{"

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

メッセージ・ブロック

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

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

MESSAGE ブロックは、グローバルあるいはローカルな効力範囲を持つことができます。MESSAGE ブロックが、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。ただし、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 (グローバル・メッセージ・ブロックによってエラーが処理される Net.Data 組み込み関数を除く)。

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

1. ローカルな MESSAGE ブロックを、RETURN_CODE または SQL_STATE の値の完全一致で検査する。指定に応じて、抜けるか、続行します。
2. その値が 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、値の符号に依存し、指定に応じて、抜け出るか、続行します。
3. 値が 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、RETURN_CODE または SQL_STATE の値の完全一致で検査する。指定に応じて、抜けるか、続行します。
5. その値が 0 でない場合は、+default または -default に対して、グローバルな MESSAGE ブロックを検査する。これは、値の符号に依存し、指定に応じて、抜け出るか、続行します。
6. 値が 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. 値が 0 でない場合、Net.Data 内部デフォルト・メッセージを発行して終了します。

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

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

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

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

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

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

Total:                    $994.37
```

関数の呼び出し

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。@ 文字に続けて関数名、またはマクロ関数名を使用します。

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

function_name

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

argument

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

引き数に使用する引用符付き文字列には、変数参照および関数呼び出しを含むことができます。

例 1: テキスト・ストリング引き数による関数呼び出し

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

例 2: 変数および関数呼び出し引き数による関数呼び出し

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

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

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

Net.Data 組み込み関数の呼び出し

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

147ページの図23 に、Net.Data 組み込み関数およびマクロがどのように相互作用しているかを示します。

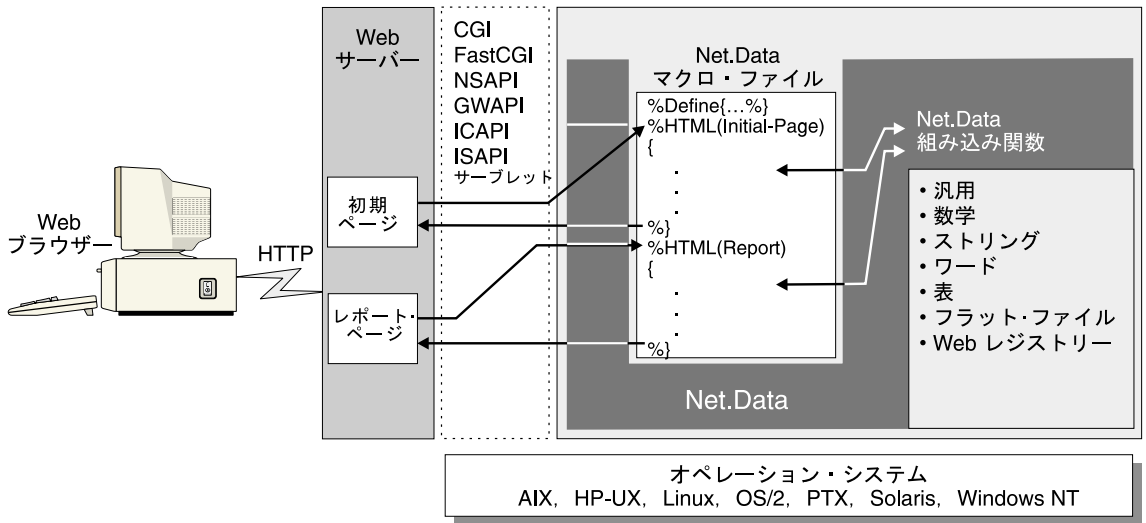


図 23. Net.Data の組み込み関数

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

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

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

以下のセクションでは、Net.Data の組み込み関数について高度な概説を提供します。以下の関数を使用して、汎用、数学、スティング、ワード、あるいは表操作の各関数を実行します。各関数とその構文および例の説明については、Net.Data 解説書を参照してください。これらの関数の中には、使用前に変数を設定する必要があるものや、特定のコンテキストで使用しなければならないものがあります。オペレーティング・システムがすべて、個々の組み込み関数

をサポートしているわけではありません。ご使用のオペレーティング・システムがサポートしている関数を判別するには、 *Net.Data* 解説書 を参照してください。

- 『汎用関数』
- 149ページの『数学関数』
- 149ページの『ストリング関数』
- 149ページの『ワード関数』
- 150ページの『表関数』
- 150ページの『フラット・ファイル関数』
- 150ページの『Web レジストリー関数』

汎用関数

この関数セットは、データを変更したり、システム・サービスを利用することにより、 Web ページの開発に役に立ちます。これらの関数を使用して、メール送信、HTTP Cookie の処理、HTML エスケープ・コードの生成、およびシステムからのその他有益情報の取得を行うことができます。

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

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


もう 1 つ、役に立つのは、DTW_URLESCSEQ 関数です。これは、URL では許可されていない文字をエスケープ値に置換します。たとえば、入力変数 string1 が "Guys & Dolls" に等しい場合は、DTW_URLESCSEQ は、出力変数に値 "Guys%20%26%20Dolls" を割り当てます。

数学関数

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

たとえば、関数 DTW_POWER は、第 1 パラメーターの値の第 2 パラメーター乗を求め、その結果を戻します。以下に例を示します。

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

DTW_POWER は、変数 result に ".125" を戻します。

ストリング関数

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

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

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

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

ワード関数

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

たとえば、DTW_DELWORD を使用して、ストリングから指定した数だけワードを削除します。

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

DTW_DELWORD は、ストリング "Now time" を戻します。

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

表関数

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

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

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

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

フラット・ファイル関数

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

たとえば、DTWF_APPEND は表変数の内容をファイルの最後に書き込み、DTWF_DELETE はレコードをファイルから削除します。

さらに、FFI 関数では、DTWF_CLOSE および DTWF_OPEN で、ファイルをロックすることができます。DTWF_OPEN はファイルをロックして、他の要求によりファイルの読み取りまたは更新を行えないようにします。DTWF_CLOSE は、Net.Data が処理を完了するとファイルをリリースし、他の要求によりファイルがアクセスできるようにします。

Web レジストリー関数

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

ドキュメント・マークアップの生成

Net.Data は、Web ブラウザーなどのクライアント・アプリケーションが使用する HTML または XML ドキュメントを動的に生成します。以下のセクションでは、Net.Data マクロを使用してドキュメントをフォーマットするために使用できるさまざまな構成について説明します。それぞれの特定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 ブロックおよび XML ブロック

クライアント・アプリケーションは、マクロの名前、およびマクロのエントリー・ポイントのうちの 1 つの名前を指定して、Net.Data を起動します。マクロへのエントリー・ポイントは、HTML ブロックまたは XML ブロックです。これらのブロックには、ほとんどのドキュメント・マークアップが含まれており、ここで関数への呼び出しが行われます。これらは、マクロの“主な”プログラム・ブロックです。

エントリー・ポイント・ブロックがマクロの実行を行うため、マクロごとに少なくともエントリー・ポイント・ブロックが 1 つなければなりません。HTML ブロックまたは XML ブロックが複数ある場合もありますが、クライアント要求ごとに 1 つしか実行されません。また、要求ごとに、1 つのドキュメントがクライアントに戻されます。多くのクライアント・ドキュメントから構成されるアプリケーションを作成するには、Net.Data を複数回起動することで、リンクやフォームなどの標準ナビゲーション手法を使用したさまざまな HTML ブロックまたは XML ブロックを処理することができます。

テキスト表示ステートメントは、クライアントに有効であれば、HTML ブロックまたは XML ブロックに表示されます。たとえば、HTML ブロックは HTML または JavaScript を含むことができます。JavaScript は Net.Data によって実行されませんが、残りの HTML ブロック出力とともにクライアントへ送られて実行および表示されます。HTML ブロックまたは XML ブロックでは、関数呼び出し、変数参照、および INCLUDE ステートメントも組み込むことができます。以下に、Net.Data マクロにおける HTML ブロックの一般的な使い方の例を示します。

```
%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()
%}

```

以下の HTML リンクから、Net.Data マクロを起動することができます。

```

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

```

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

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

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

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

レポート・ブロック

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

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

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

例：

```
%REPORT{
<h2>Query Results</h2>
<p>Select a name for details.
<table border=1>
  <tr>
    <td>Name</td>
    <td>Location</td></tr>
  %ROW{
    <tr>
      <td>
<a href="/cgi-bin/db2www/name.d2w/details?name=$(V1)&loc;=$(V2)">$(V1)</a>
      </td>
      <td>$(V2)</td>
```

```

    </tr>
    %}
</table>
%}

```

REPORT ブロックのガイドライン

REPORT ブロックの作成時には、以下のガイドラインを使用してください。

- ROW ブロックからの表出力をどれも表示しないようにするには、ROW ブロックを空にしておくか、ROW ブロックを完全に省略します。
- Net.Data により提供される REPORT ブロック内の変数を使用して、Net.Data のマクロの結果表のデータにアクセスします。これらの変数は、136ページの『表処理変数』で説明されています。これ以上の詳細については、*Net.Data* 解説書 のレポート変数の節を参照してください。
- ヘッダーおよびフッター情報を提供するには、ROW ブロックの前後にテキストを入力します。Net.Data は、ROW ブロックの前で検出したものはすべてヘッダー情報として処理します。Net.Data は、ROW ブロックの後で検出したものはすべてフッター情報として処理します。HTML ブロックの場合と同様、Net.Data は、マクロ言語構成要素として認識できない、ヘッダー、ROW およびフッター・ブロックにおけるすべてのものを、テキスト表示ステートメントとして扱い、これらのステートメントを、ブラウザに送信します。
- REPORT ブロックでは関数および参照変数を呼び出すことができます。
- Net.Data に、フォーマット済みのテキストを使用して、デフォルトのレポートをプリントさせるには、マクロ・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、関数が HTML ブロックで呼び出されるときの、デフォルトのレポート・フォーマットを示しています。

```

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

```

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

```
%REPORT{%}
```

例：レポートのカスタマイズ

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

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

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

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

Net.Data は、以下を行い、レポートを生成しました。

1. *Phone Query Results:* を、レポートの最初に 1 回プリントする。区切り線の付いたこのテキストは、REPORT ブロックのヘッダー部分です。

2. 検索時に各行ごとに、変数 V1、V2、および V3 を、それぞれ Name、Phone、および Fax の値で置換する。
3. スtring *Total records retrieved:* 、および TOTAL_ROWS の値を、レポートの最後に一度プリントする。(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。

一般的に、複数の REPORT ブロックは、DTW_SQL 言語環境でストアード・プロシージャを呼び出す関数に対して使用します。この場合、複数の結果セットが戻ります (177ページの『ストアード・プロシージャ』を参照)。ただし、複数の REPORT ブロックは、複数のレポートを生成させるために、任意の言語環境とともに使用することができます。

複数の REPORT ブロックを使用するには、それぞれの結果セットについて、ストアード・プロシージャの CALL に結果セット名を配置します。指定したレポート・ブロックの数より多い結果セットが、ストアード・プロシージャから戻される場合、そしてまた、Net.Data 組み込み関数 DTW_DEFAULT_REPORT = "MULTIPLE" である場合は、レポート・ブロックと関連がない各表に対して、デフォルトのレポートが生成されます。レポート・ブロックが指定されていない場合、および DTW_DEFAULT_REPORT = "YES" である場合は、デフォルトのレポートは 1 つ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 () {  
    CALL myproc (table1, table2) %}
```

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

例 2: MACRO_FUNCTION ブロック この例では、2 つの表が MACRO_FUNCTION ブロックに渡されます。

DTW_DEFAULT_REPORT="MULTIPLE" が指定されている場合は、Net.Data は両方の表に対するレポートを生成します。

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

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

例 3: DTW_REXX 言語環境

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

この例では、2 つの表が REXX 関数 multReport に渡されます。DTW_DEFAULT_REPORT="YES" が指定されているため、Net.Data は最初の表にのみデフォルトのレポートを表示します。

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

例 1: 名前付き REPORT ブロック

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

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

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

この例では、REPORT ブロックが、FUNCTION ブロック・パラメーター・リストに渡された両方の表に対して指定されています。これらの表は、REPORT ブロックで指定された順序で、最初に table2 に、次に table1 について表示

されます。 **REPORT** ブロックで表の名前を指定することによって、レポートが表示される順序を制御することができます。

例 2: 名前のない **REPORT** ブロック

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

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

デフォルトのレポート、および *REPORT* ブロックの組み合わせを使用して、複数のレポートを表示させる

例 : デフォルトのレポートと **REPORT** ブロックの組み合わせ

```
%DEFINE DTW_DEFAULT_REPORT = "MULTIPLE"  
%FUNCTION(dtw_system) editTables (INOUT table1, table2, table3) {  
    %REPORT(table2) {  
        ...  
        %ROW { .... %}  
        ...  
    %}  
%}
```

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

複数の REPORT ブロックに関するガイドラインおよび制約事項:

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

ガイドライン :

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

制約事項 :

- 関数内のすべてのレポート変数の値は、その関数内のすべての REPORT ブロックに適用されます。個々の REPORT ブロックのレポート変数の値を変更することはできません。
- MESSAGE ブロック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、関数に渡す前に TABLE ステートメント内で定義する必要があります。
- 最初のレポート・ブロックに表の名前が指定されていると、すべてのレポート・ブロックが表の名前を指定しなくてはなりません。
- 最初のレポート・ブロックが表の名前を指定していなければ、レポート・ブロックは表の名前を指定することはできません。
- 1 つのストアード・プロシージャの表の最大数は 32 です。

マクロにおける条件付き論理とループ

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

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

以下のセクションでは、条件付き論理およびループを説明しています。

- 『条件付き論理：IF ブロック』
- 163ページの『ループ構成体：WHILE ブロック』

条件付き論理：IF ブロック

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

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

IF ブロックの規則：IF ブロック構文の規則は、マクロ内のブロックの位置によって決まります。IF ブロックの実行可能なステートメント・ブロックに許される要素は、IF ブロック自身の位置に依存します。

- IF ブロックを含むブロック内で有効な要素ならどれでも、その IF ブロックで有効です。たとえば、IF ブロックを HTML ブロック内で指定する場合、HTML ブロックに許可される要素はどれでも、INCLUDE ステートメントおよび WHILE ブロックなどの IF ブロックで許可されます。

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

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

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

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

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

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

例外： IF ブロックで ROW ブロックを指定しないでください。

IF ブロックのSTRING比較

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

有効な整数ストリングの例：

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

無効な整数ストリングの例：

```
- 20      (ブランク文字を含んでいる)
234,000   (コンマを含んでいる)
57.987    (小数点を含んでいる)
```

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

IF ブロックの例：他のブロック内に IF ブロックを含むマクロ

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

%IF (platform == "AS400")
  %IF (version == "V3R2")
    %INCLUDE "as400v3r2_def.hti"
  %ELIF (version == "V3R7")
    %INCLUDE "as400v3r7_def.hti"
  %ELIF (version == "V4R1")
    %INCLUDE "as400v4r1_def.hti"
  %ENDIF
%ELSE
  %INCLUDE "default_def.hti"
%ENDIF

%MACRO_FUNCTION numericCompare(IN term1, term2, OUT result) {
  %IF (term1 < term2)
    @dtw_assign(result, "-1")
  %ELIF (term1 > term2)
    @dtw_assign(result, "1")
  %ELSE
    @dtw_assign(result, "0")
  %ENDIF
}%

%HTML(report){
  %WHILE (a < "10") {
    outer while loop #$(a)<br />
```

```

%IF (@dtw_rdivrem(a,"2") == "0")
    this is an even number loop<br />
%ENDIF
@DTW_ADD(a, "1", a)
%}
%}

```

ループ構成体：WHILE ブロック

WHILE ブロックを使用して、Net.Data のマクロでループを実行します。IF ブロックと同様、WHILE ブロックにより、1 つ以上の条件をテストし、次に、条件テストの結果に基づいてステートメントのブロックを実行することができます。IF ブロックと異なり、ステートメントのブロックは、条件テスト結果に基づき、何回でも実行することができます。

WHILE ブロックを HTML ブロック、REPORT ブロック、ROW ブロック、MACRO_FUNCTION ブロック、および IF ブロック内で指定し、それらをネストすることができます。WHILE ブロックの構文は、*Net.Data* 解説書の言語構成要素の章に示されています。

Net.Data は、WHILE を、IF ブロックを処理するのと全く同じ方法で処理しますが、ブロックを実行するごとに条件を再評価します。そして、どの条件付きループ構成要素の場合も同じですが、条件のコード化に誤りがある場合は、処理は無限ループに陥ることがあります。

例：WHILE ブロックを持つマクロ

```

%DEFINE loopCounter = "1"

%HTML(build_table) {
    %WHILE (loopCounter <= "100") {
        %{ generate table tag and column headings %}
        %IF (loopCounter == "1")
            <table border>
            <tr>
            <th>Item #
            <th>Description
        %ENDIF

        %{ generate individual rows %}
        <tr>
        <td>$(loopCounter)
        <td>@getDescription(loopCounter)

        %{ generate end table tag %}
    }
}

```

```
%IF (loopCounter == "100")
%ENDIF

%{ increment loop counter %}
@DTW_ADD(loopCounter, "1", loopCounter)
%}
%}
```

第6章 言語環境の使用

Net.Data はデータ・ソースへのアクセスと、ビジネス・ロジックを持つアプリケーション・プログラムの実行のための、言語環境を提供します。たとえば、SQL 言語環境によって、SQL ステートメントを DB2 データベースに渡すことが可能になり、REXX 言語環境によって、REXX プログラムを起動できるようになります。また、SYSTEM 言語環境を使用することにより、プログラムを実行したり、コマンドを発行したりすることができます。

Net.Data を使うことにより、ユーザー作成の言語環境をプラグイン方式で追加することができます。ユーザー作成言語環境は、それぞれ Net.Data によって定義された、標準的なインターフェースのセットをサポートしていなければなりません。さらに、ダイナミック・リンク・ライブラリー (DLL)、または共用ライブラリーとして実装されていなければなりません。Net.Data 提供の言語環境およびユーザー作成の言語環境の作成方法の詳細については、*Net.Data 言語環境解説書* を参照してください。

166ページの図24 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。

• 193ページの『プログラミング言語環境』

表7 では、各言語環境を簡単に説明しています。どのオペレーティング・システムが、どの言語環境をサポートしているかを知るには、*Net.Data* 解説書のオペレーティング・システムの付録を参照してください。

表 7. *Net.Data* 言語環境

言語環境	環境名	説明
フラット・ファイル・インターフェース	DTW_FILE	フラット・ファイル・インターフェース (FFI) は、データ・ソースとしてのテキスト・ファイルをサポートする関数をサポートしています。
IMS Web	HWS_LE	IMS Web 言語環境では、IMS Web を使用して IMS トランザクションを処理依頼し、Web ブラウザーでそのトランザクションの出力を受け取ることができます。
Java アプレット	DTW_APPLET	Java アプレット言語環境では、 <i>Net.Data</i> アプリケーションで、Java アプレットを使用することができます。applet タグを生成するには、applet タグの限定子とアプレットのパラメーター・リストを指定しなければなりません。
Java アプリケーション	DTW_JAVAPPS	<i>Net.Data</i> は、Java 言語環境により既存の Java アプリケーションをサポートします。
ODBC	DTW_ODBC	ODBC 言語環境は、複数のデータベース管理システムにアクセスするための、ODBC インターフェースにより SQL を実行します。
Oracle	DTW_ORA	Oracle 言語環境では、Oracle データに直接アクセスすることができます。
Perl	DTW_PERL	Perl 言語環境は、 <i>Net.Data</i> の FUNCTION ブロックで指定された、内部 Perl スクリプトを解釈したり、別のファイルに保管されている外部 Perl スクリプトを実行したりします。
REXX	DTW_REXX	REXX 言語環境は、 <i>Net.Data</i> の FUNCTION ブロックで指定された、内部 REXX プログラムを解釈したり、別のファイルに保管されている外部 REXX プログラムを実行したりします。
SQL	DTW_SQL	SQL 言語環境は、DB2 を介して SQL ステートメントを実行します。SQL ステートメントの結果は、表変数に格納して戻すことができます。
システム	DTW_SYSTEM	システム言語環境は、コマンドの実行と外部プログラムの呼び出しをサポートします。

表 7. *Net.Data* 言語環境 (続き)

言語環境	環境名	説明
Web レジストリー	DTW_WEBREG	Web レジストリー言語環境は、アプリケーション関連の永続的記憶域のための関数を提供します。

言語環境の呼び出し

言語環境を呼び出すには、以下のことを行います。

- FUNCTION ステートメントを使用して、言語環境を呼び出す関数を定義する。
- 言語環境への関数呼び出しを使用する。

たとえば、以下のような場合です。

```
%FUNCTION(DTW_SQL) custinfo() {
  select CUSTNAME, CUSTNO from ibmuser.customer
  %}
...
%HTML(REPORT) {
  @custinfo()
  %}
```

エラー条件の処理

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

以下のリソースを使用して、エラー条件を処理することができます。

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

機密保護

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

データ言語環境

Net.Data が提供するデータ言語環境では、リレーショナル・データベースおよび階層データベースからデータにアクセスできます。また、その他のデータ・ソースには Net.Data のマクロからアクセスできます。以下のセクションでは、Net.Data が提供するデータ言語環境と、 Net.Data マクロにおけるその使用方法について説明します。

- 『リレーショナル・データベース言語環境』
- 189ページの『フラット・ファイル・インターフェース言語環境』
- 190ページの『Web レジストリー言語環境』

リレーショナル・データベース言語環境

Net.Data は、関係データ・ソースにアクセスするのに役立つリレーショナル・データベース言語環境を提供します。関係データにアクセスするために提供する SQL ステートメントは、動的 SQL として実行されます。動的 SQL についての詳細は、DB2 資料を参照してください。

以下のセクションでは、言語環境とその使用方法について解説しています。

- 『ODBC 言語環境』
- 170ページの『Oracle の言語環境』
- 171ページの『SQL 言語環境』
- 172ページの『Net.Data アプリケーションにおけるトランザクション管理』
- 173ページの『ラージ・オブジェクトを使用する』
- 177ページの『ストアド・プロシージャ』
- 184ページの『結果セットでの DataLink URL のコード化』
- 186ページの『リレーショナル・データベース言語環境の例』

ODBC 言語環境

オープン・データベース・コネクティビィティ (ODBC) 言語環境は、ODBC インターフェースを介して SQL ステートメントを実行します。ODBC は X/Open SQL CAE 仕様にに基づいています。この仕様では、単一のアプリケーションから多数のデータベース管理システムにアクセスできます。

ODBC 言語環境の使用方法 :

ODBC 言語環境を使用するには、まず ODBC ドライバーおよびドライバー・マネージャーを入手してインストールします。ODBC ドライバーの資料には、ODBC 環境のインストールとその構成方法が説明されています。

以下の構成ステートメントが、Net.Data の初期設定ファイルに 1 行で記述されていることを確認してください。

```
ENVIRONMENT (DTW_ODBC) d:/net.data/lib/dtwodbc.dll ( IN DATABASE, LOGIN, PASSWORD,  
TRANSACTION_SCOPE, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
```

制約事項：

- ODBC 言語環境は、DB2 に接続するときのみ、ストアード・プロシージャをサポートします。
- DATABASE 変数を指定する場合には、ODBC 初期設定ファイルのデータ・ソースと同じデータベースを指定しなければなりません。
- インラインのステートメント・ブロックの、SQL ステートメントの最大サイズは 64KB です。DB2 ユニバーサル・データベースには、以下の制約事項があります。
 - バージョン 6 以降：64KB
 - バージョン 5 リリース 2 以前：32KB

ご使用のデータベースによっては別の制限がある場合があります。そのような制限があるかどうかを知るには、ご使用のデータベースの資料を参照してください。

Oracle の言語環境

Oracle 言語環境では、Oracle データへのネイティブなアクセスが提供されます。CGI、FastCGI、NSAPI、ISAPI、あるいは GWAPI を使用して、Net.Data から Oracle のデータベースにアクセスできます。この言語環境では、Oracle 7.2、7.3、および 8.0 がサポートされています。

Oracle 言語環境を使用するには、以下の構成ステートメントが、初期設定ファイルに 1 行で記述されていることを確認してください。

```
ENVIRONMENT (DTW_ORA) /net.data/lib/dtwora.so (IN DATABASE, LOGIN, PASSWORD,  
TRANSACTION_SCOPE, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
```

Oracle 言語環境のセットアップ方法については、35ページの『Oracle 言語環境のセットアップ』を参照してください。

制約事項：

- DATABASE 変数を使用して Oracle データベースにアクセスできない。
- LOGIN 変数には、Oracle データベースのインスタンス名が格納されていなければならない。たとえば、ora73 は、次の LOGIN 変数では定義済みのインスタンス名です。

```
LOGON=admin@ora73
```

- CGI 以外のインターフェースを使用する場合は、Live Connection を使用しなければならない。
- ストアード・プロシージャはサポートされない。

SQL 言語環境

SQL 言語環境では、DB2 データベースにアクセスできます。DB2 へのアクセス時に最適のパフォーマンスを得るには、この言語環境を使用してください。

SQL 言語環境を使用するには、以下の構成ステートメントが、初期設定ファイルに 1 行で記述されていることを確認してください。

```
ENVIRONMENT (DTW_SQL) d:/net.data/lib/dtwsq1.d11 (IN DATABASE, LOGIN, PASSWORD,
TRANSACTION_SCOPE, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
```

ネストされた SQL ステートメント

SQL 関数は、別の SQL 関数内から呼び出すことができます。テーブルを渡す場合は、各関数ごとに固有のテーブル名を使用するようにしてください。そうしないと、予期しない結果が生じることがあります。

例：別の SQL 関数の ROW ブロックから SQL 関数を呼び出す場合

```
%define mytable1 = %TABLE
%define mytable2 = %TABLE

%FUNCTION(DTW_SQL) sq12 (IN p1, OUT t2) {
    select * from NETDATA.STAFFINF where projno='${p1}'
    %REPORT {
        %ROW { $(N1) is $(V1) %}
    %}
%}

%FUNCTION(DTW_SQL) sq11 (OUT t1) {
    select * from NETDATA.STAFFINF
    %REPORT {
        %ROW { @sq12(V1, mytable2) %}
    %}
%}

%HTML(netcall1) { @sq11(mytable1) %}
```

制約事項：

インラインのステートメント・ブロックの、SQL ステートメントの最大サイズは 64KB です。DB2 ユニバーサル・データベースには、以下の制約事項があります。

- バージョン 6 以降：64KB

- バージョン 5 リリース 2 以前 : 32KB

ご使用のデータベースによっては別の制限がある場合があります。そのような制限があるかどうかを判別するには、データベースの資料を参照してください。

ネスト SQL は、SQL または ODBC の言語環境でのみ使用でき、Live Connection と連係して使用することはできません。

SQL ステートメントにネストする際、結果セットの最大数は 32 です。たとえば、3 つのレベルにネストすると、それぞれ 10 の結果セットを戻すことができます。あるいは、32 レベルにネストして、結果セットを 1 つずつ戻すこともできます。

Net.Data アプリケーションにおけるトランザクション管理

挿入、削除、または更新ステートメントを用いてデータベースの内容を変更する場合、その変更は、データベースが Net.Data からコミット・ステートメントを受け取るまでは永続的なものとはなりません。エラーが発生すると、Net.Data はデータベースにロールバック・ステートメントを送り、前回のコミットからの修正をすべてリバースにします。

Net.Data がコミットと、場合によってはロールバックを送信する方法は、TRANSACTION_SCOPE の設定値、およびマクロにコミット・ステートメントが明示的に指定されているかどうかによって異なります。

TRANSACTION_SCOPE の値は MULTIPLE および SINGLE です。

SINGLE

各 SQL ステートメントが正常に終了した後に Net.Data がコミット・ステートメントを発行するよう指定します。SQL ステートメントがエラーを戻すと、ロールバック・ステートメントが発行されます。単一トランザクション効力範囲によりデータベースの即時変更が確実となりますが、この効力範囲では後でロールバック・ステートメントを使用して変更を取り消すことはできません。

このコミット・メソッドを活動化するには、TRANSACTION_SCOPE を SINGLE に設定します。たとえば、以下のような場合です。

```
@DTW_ASSIGN(TRANSACTION_SCOPE,"SINGLE")
```

MULTIPLE

コミットおよび場合によってはロールバック・ステートメントが発行される前に、Net.Data がすべての SQL ステートメントを実行するかを指定します。Net.Data は要求の最後にコミットを送信し、各 SQL ステ

ートメントが正常に発行されると、コミットによりデータベースのすべての変更が永続的なものとなります。ステートメントのいずれかがエラーを戻すと、Net.Data はロールバック・ステートメントを発行し、データベースの設定は元の状態に戻ります。TRANSACTION_SCOPE が設定されていなければ、MULTIPLE がデフォルトです。

コミット・ステートメントは、COMMIT SQL ステートメントを使用することによりマクロ内の任意の SQL ステートメントの最後で発行することができます。アプリケーション開発者は、TRANSACTION_SCOPE の設定を MULTIPLE にして、トランザクションとみなしたステートメント・グループの最後にコミット・ステートメントを発行することにより、アプリケーションにおけるコミットおよびロールバックの振る舞いを完全に管理することができます。たとえば、マクロ中のそれぞれの更新の後にコミット・ステートメントを発行することにより、データの保全性が保証できます。

SQL コミット・ステートメントを発行するには、HTML ブロックの任意のポイントに呼び出し可能な関数を定義します。

```
%FUNCTION(DTW_SQL) user_commit() {  
    commit  
}%  
  
...  
%HTML {  
    ...  
    @user_commit()  
    ...  
}%
```

制約事項：

データベースに接続後は、TRANSACTION_SCOPE の設定値は変更できません。したがって、マクロ内のすべての SQL トランザクションの効力範囲は、同じになります。

Net.Commerce の一部として Net.Data を使用する場合、Net.Commerce は独自のトランザクション処理を持ち、Net.data のトランザクション処理を使用不可にするので注意してください。

ラージ・オブジェクトを使用する

ラージ・オブジェクト・ファイル (LOB) は、DB2 データベースに格納することができ、Net.Data SQL または ODBC 言語環境を使用して、動的 Web ページに組み込むことができます。

言語環境は、LOB を戻す SQL SELECT ステートメントまたはストアド・プロシージャを実行する場合、V(n) テーブル処理変数または Net.Data テーブル・フィールドにオブジェクトの割り当てを行いません。その代わりに言語環境は、Net.Data が作成するファイルに LOB を格納し、V(n) テーブル処理変数または Net.Data テーブル・フィールド内のファイルの名前のみを戻します。Net.Data マクロでは、その名前を使用して LOB ファイルを参照することができます。たとえば、ハイパーテキスト参照を持つ HTML のアンカー要素、あるいは LOB ファイルの URL を含むイメージ要素を作成することができます。Net.Data は、LOB を含むファイルを、Net.Data の初期設定ファイル (db2www.ini) 内の HTML_PATH パス・ステートメントで指定したディレクトリに配置します。LOB ファイルへの書き込みアクセスは、その LOB を取得した Net.Data 要求に関連付けられているユーザー ID に制限されます。

LOB のファイル名は動的に構成され、以下の形式をしています。

name[*.extension*]

ここで

name ラージ・オブジェクトを識別する、動的に生成された固有のストリング。

extension

オブジェクトの型を識別するストリング。CLOB と DBCLOB の場合、拡張子は .txt です。BLOB の場合、SQL 環境は、LOB ファイルの先頭の数バイトにあるシグニチャーを探し、拡張子を決定します。

表8に、SQL 言語環境で使用される LOB の拡張子を示します。

表 8. SQL 言語環境で使用される LOB の拡張子

拡張子	オブジェクト・タイプ
.bmp	ビットマップ・イメージ
.gif	グラフィック・イメージ形式
.jpg	Joint Photographic Experts Group (JPEG) イメージ
.tif	Tagged Image File Format
.ps	postscript
.mid	MIDI オーディオ
.aif	AIFF オーディオ
.avi	Audio Visual Interleave オーディオ
.au	基本オーディオ
.ra	リアル・オーディオ
.wav	Windows オーディオ・ビジュアル
.pdf	PDF (portable document format)
.rmi	MIDI シーケンス

BLOB のオブジェクト・タイプが認識されると、ファイル名に拡張子が付加されます。

Net.Data は、LOB を含むファイルの名前を戻す際に、次の構文を使用して、そのファイル名にストリング /tmplobs/ という接頭部を付けます。

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

この接頭部により、Web サーバーの文書ルート・ディレクトリー以外のディレクトリー内の LOB ディレクトリーを見つけることができます。

LOB ファイルへの参照が正確に解決されるように、次の Pass ディレクティブを、Web サーバーの構成ファイルに追加してください。

```
Pass    /tmplobs/*    <html_path>/tmplobs/*
```

<html_path> は、Net.Data の初期設定ファイル内の HTML_PATH パス・ステートメントに指定された値です。

計画のためのヒント： 照会が LOB を戻すと、HTML_PATH パス構成変数で指定されたディレクトリーにファイルが作成されます。LOB を使用する場合、リソースがすぐに消費されるため、システムの制限を考慮してください。定期的にディレクトリーを整理する必要があります。あるいは dtwclean デーモンを実行しても構いません。詳しくは、176ページの『一時的 LOBS の管理』を参照してください。DataLinks を使用することをお勧めします。これによって、SQL 言語環境でディレクトリーにファイルを保管する必要性がなくなり、パフォーマンスが向上して、システム資源の使用量が少なくなります。

例： 以下に示したアプリケーションでは、MPEG オーディオ (.mpa) ファイルが使用されています。SQL 言語環境ではこのファイル・タイプは認識されないため、EXEC 変数を使用して、.mpa 拡張子をファイル名に追加します。このアプリケーションのユーザーは、ファイル名をクリックして、MPEG オーディオのファイル・ビューアーを起動しなければなりません。

```
%DEFINE{
docroot="/usr/lpp/internet/server_root/html"
myFile=%EXEC "rename $(docroot){filename} $(docroot){filename}.mpa"
%}
%{ where rename is the command on your operating system to rename files %}
%FUNCTION(DTW_SQL) queryData() {
    SELECT Name, IDPhoto, Voice FROM RepProfile
    %REPORT{
        <p>Here is the information you selected:</p>
        %ROW{
            @DTW_ASSIGN(filename, @DTW_rSUBSTR(V3, @DTW_rLASTPOS("/", V3)))
            $(myFile)
            $(V1) 
```

```

        <a href="$(V3).mpa">Voice sample</a><p>
    %}
    %}
%}

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

```

RepProfile テーブルに Kinson Yamamoto と Merilee Lau に関する情報が含まれている場合、REPORT ブロックを実行すると、以下に示した HTML ファイルが、生成される Web ページに追加されます。

```

<p>Here is the information you selected:</p>
Kinson Yamamoto 
<a href="/tmplobs/p2345n2.mpa">Voice sample</a><p>
Merilee Lau 
<a href="/tmplobs/p2345n4.mpa">Voice sample</a><p>

```

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

- V1 の値は個人名で、文字データです。
- V2 の値は、その人の写真を含む GIF ファイルの名前です。このイメージは、生成された Web ページ内ではインラインで表示されます。
- V3 の値は、その人の音声のサンプルを含む MPA ファイルの名前です。Net.Data は MPA ファイル形式を認識しないので、HTML_PATH で指定されたディレクトリーに LOB のファイルを作成しても、ファイルに拡張子を追加しません。この例は、EXEC 変数を使用して .mpa 拡張子をファイル名に追加するための方法を示したものです。音声サンプルは、ユーザーが「音声サンプル (Voice Sample)」というテキストをクリックすると再生されます。このテキストは、ハイパーテキストになっています。

LOB に対するアクセス権限:

LOB のデフォルトの tmplobs ディレクトリーは、出荷時の Net.Data の初期設定ファイル内の HTML_PATH で指定されたディレクトリーの下にあります。どんなユーザー ID でも、アクセス可能です。HTML_PATH 値が変更されている場合は、Web サーバーが実行されているユーザー ID が、HTML_PATH で指定されたディレクトリーに書き込みアクセスできることを確認してください (詳しくは、26ページの『HTML_PATH』を参照してください)。

一時的 LOB の管理:

Net.Data は、HTML_PATH パス構成変数で指定されたディレクトリーの下にある、tmplobs というサブディレクトリーに一時的 LOB を保管します。これら

のファイルは大きくても構いませんが、定期的に整理し、受け入れ可能なパフォーマンスを保守する必要があります。

Net.Data には、dtwclean というデーモンが備えられています。これは、tmplobs ディレクトリーを定期的に管理する際に役立ちます。 dtwclean は、ポート 7127 を使用します。

dtwcleanデーモンを実行するには、以下のようにします。 コマンド行ウィンドウで、以下のコマンドを入力します。

```
dtwclean [-t xx] [-d|-l]
```

ここで

- t** dtwclean がディレクトリーを整理する間隔を指定するフラグ
- xx** dtwclean がファイルを消去する前に、そのファイルがディレクトリー内にとどまる間隔 (秒)。この値には、制限がありません。デフォルトは、3600 秒です。
- d** デバッグ・モードを指定するフラグ。トレース情報はコマンド・ウィンドウに表示されます。
- l** ログ・モードを指定するフラグ。トレース情報はログ・ファイルに表示されます。

ストアード・プロシージャ

ストアード・プロシージャは DB2 に保管されたコンパイル済みのプログラムで、SQL ステートメントを実行することができます。Net.Data では、ストアード・プロシージャは、CALL ステートメントを使用して、Net.Data の関数から呼び出されます。ストアード・プロシージャのパラメーターは、Net.Data の関数仮引き数リストから渡されます。ストアード・プロシージャを使用すると、コンパイル済みの SQL ステートメントを、データベース・サーバーと一緒に保管することにより、パフォーマンスと保全性を改良することができます。Net.Data は、SQL および ODBC 言語環境での DB2 によるストアード・プロシージャ使用をサポートします。

このセクションでは、以下のトピックについて説明します。

- 178ページの『ストアード・プロシージャの構文』
- 179ページの『ストアード・プロシージャの呼び出し』
- 180ページの『パラメーターを渡す』
- 180ページの『結果セットの処理』

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

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

%}
```

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 関数名

stored_procedure

ストアード・プロシージャの名前

datatype

Net.Data がサポートするデータベース・データ型の 1 つ (表9 を参照)。パラメーター・リストで指定されるデータ型は、ストアード・プロシージャ内のデータ型と一致しなければなりません。これらのデータ型に関するさらに詳しい情報については、データベースの文書を参照してください。

tablename

結果セットを保管する Net.Data テーブルの名前 (結果セットを Net.Data テーブルに保管する場合のみ使用)。指定した場合、パラメーター名は *resultsetname* の関連パラメーター名に一致する必要があります。

resultsetname

ストアード・プロシージャから戻された結果を REPORT ブロックまたは関数仮引き数リスト (あるいはその両方) に関連付ける名前。REPORT ブロックの *resultsetname* は、CALL ステートメントの結果セットに一致する必要があります。

表 9. サポートされているストアード・プロシージャのデータ・タイプ

BIGINT	DOUBLEPRECISION	SMALLINT
CHAR	FLOAT	TIME
CLOB ¹	INTEGER	TIMESTAMP
DATE	GRAPHIC	VARCHAR
DECIMAL	LONGVARCHAR	VARGRAPHIC
DOUBLE	LONGVARGRAPHIC	

表9. サポートされているストアード・プロシージャのデータ・タイプ (続き)

¹ CLOB は、OUT および INOUT パラメーターとしてだけ使用されます。Net.Data はサイズをバイトで解釈します。たとえば、変数を OUT CLOB(20000) と指定すると、20K のサイズの CLOB が出力パラメーターとして使用されます。

ストアード・プロシージャの呼び出し:

1. ストアード・プロシージャへの呼び出しを開始する関数を定義します。

```
%FUNCTION (DTW_SQL) function_name()
```

2. オプションで任意の IN、INOUT、または OUT パラメーターをストアード・プロシージャに指定します。これには、結果セットを Net.Data テーブルに保管するためにテーブル変数名 (結果セットを Net.Data テーブルに保管する場合は、Net.Data テーブルのみを指定) が含まれます。他のストアード・プロシージャからのテーブル名または結果セットとして (IN または INOUT パラメーターとして) 指定することもできます。

```
%FUNCTION (DTW_SQL) function_name (IN datatype
arg1, INOUT datatype arg2,
      OUT tablename...)
```

3. CALL ステートメントを使用して、ストアード・プロシージャ名を識別します。

```
CALL stored_procedure
```

4. ストアード・プロシージャが 1 つの結果セットを生成する場合は、オプションで REPORT ブロックを指定して Net.Data による結果セットの表示方法を定義します。

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

例 :

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

5. ストアード・プロシージャが複数の結果セットを生成する場合には以下のようになります。

- CALL ステートメントに結果セット名を指定します。

```
CALL stored_procedure (resultsetname1, resultsetname2, ...)
```

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

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

例 :

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

パラメーターを渡す: ストアド・プロシージャにパラメーターを渡すことができます。また、ストアド・プロシージャがパラメーターを更新するようにして、新規値が Net.Data マクロに渡されるようにできます。関数仮引き数リストのパラメーターの数および型は、ストアド・プロシージャに定義した数および型に一致する必要があります。たとえば、ストアド・プロシージャに定義したパラメーター・リストのパラメーターが INOUT の場合、関数仮引き数リストにある対応パラメーターは INOUT でなくてはなりません。ストアド・プロシージャに定義したリストのパラメーターが CHAR(30) の場合、関数仮引き数リストの対応パラメーターは CHAR(30) でなくてはなりません。

例 1: ストアド・プロシージャにパラメーター値を渡す

```
%FUNCTION (DTW_SQL) mystoredproc (IN CHAR(30) valuein) {
    CALL myproc
...
}
```

例 2: ストアド・プロシージャから値を戻す

```
%FUNCTION (DTW_SQL) mystoredproc (OUT VARCHAR(9) retvalue) {
    CALL myproc
...
}
```

結果セットの処理: SQL または ODBC 言語環境を使用することによりストアド・プロシージャから 1 つ以上の結果セットを戻すことができます。結果セットは、Net.Data テーブルに保管してさらにマクロで処理をするか、または REPORT ブロックを使用して処理することができます。ストアド・プロシージャが複数の結果セットを生成する場合、ストアド・プロシージャの生成した結果セットにそれぞれ名前を関連付ける必要があります。これは、CALL ステートメントにパラメーターを指定して行います。結果セットに指定

する名前を **REPORT** ブロックまたは **Net.Data** 表と関連付け、**Net.Data** が各結果セットを処理する方法を決めることができます。以下を行うことができます。

- 結果セットにレポート・ブロックを定義せずに、結果を **Net.Data** のデフォルトのレポート・スタイルで処理します。
- 結果セットを **REPORT** ブロックに関連付け、自身のレポート・スタイルを適用します。 **REPORT** ブロックでは、**Net.Data** 変数、 **HTML** または **JavaScript** などのテキスト処理ステートメント、またはその他の関数を使用して、ブラウザのレポート・データ表示方法が指定できます。
- 後に **Net.Data** マクロでデータを使用する場合は、結果セットを **Net.Data** テーブルに保管します。たとえば、**Net.Data** 表をほかの関数に渡し、その関数が計算のためにデータを使用し、その計算に基づいた結果を表示することができます。

複数のレポート・ブロックを使用する場合は、159ページの『複数の **REPORT** ブロックに関するガイドラインおよび制約事項』のガイドラインおよび制限を参照してください。

単一の結果セットを戻してデフォルトのレポートを使用する

以下の構文を使用します。

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

たとえば、以下のような場合です。

```
%FUNCTION (DTW_SQL) mystoredproc () {  
    CALL myproc  
%}
```

単一の結果セットを戻して **REPORT** ブロックを指定する

以下の構文を使用します。

```
%FUNCTION (DTW_SQL) function_name () {  
    CALL stored_procedure [(resultsetname)]  
    %REPORT [(resultsetname)] {  
        ...  
    %}  
%}
```

たとえば、以下のような場合です。

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

代わりに、以下の構文を使用することもできます。

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

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

たとえば、以下のような場合です。

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

処理を続行するために、**Net.Data** 表に単一の結果セットを保管するには、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 (resultsetname)
}
```

たとえば、以下のような場合です。

```
%DEFINE DTW_DEFAULT_REPORT = "NO"

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

DTW_DEFAULT_REPORT が NO に設定され、デフォルトのレポートが結果セットのために生成されないことに注意してください。

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 () {  
    CALL stored_procedure [(resultsetname1, resultsetname2, ...)]  
%}
```

ただし、レポート・ブロックは指定されていません。

たとえば、以下のような場合です。

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

複数の結果セットを戻し、処理を続行するために **Net.Data** 表にその結果セットを保管するには、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 (resultsetname1, resultsetname2, ...)  
%}
```

たとえば、以下のような場合です。

```
%DEFINE DTW_DEFAULT_REPORT = "NO"  
  
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable1, mytable2) {  
    CALL myproc (mytable1, mytable2)  
%}
```

DTW_DEFAULT_REPORT が NO に設定され、デフォルトのレポートが結果セットのために生成されないことに注意してください。

複数の結果セットを戻し、表示処理のために **REPORT** ブロックを指定するには、以下のようにします。

それぞれの結果セットは、1 つまたは複数の REPORT ブロックに関連付けられています。以下の構文を使用します。

```
%FUNCTION (DTW_SQL) function_name (, ...) {  
    CALL stored_procedure (resultsetname1, resultsetname2, ...)  
    %REPORT (tablename1)  
    ...  
    %ROW { ... %}  
    ...  
%}  
%REPORT (tablename2)  
    ...  
    %ROW { ... %}  
    ...
```

```
%}

...
%}
```

たとえば、以下のような場合です。

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

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

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

複数の結果セットを戻し、各結果セットに異なる表示または処理オプションを指定するには、以下のようにします。

固有のパラメーター名を使用して、各結果セットに異なる処理オプションを指定することができます。たとえば、以下のような場合です。

```
%FUNCTION (DTW_SQL) mystoredproc (OUT mytable2) {
    CALL myproc (mytable1, mytable2, mytable3)

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

結果セット mytable1 は、対応する REPORT ブロックにより処理され、マクロ書き込みプロセスの指定どおりに表示されます。結果セット mytable2 は、Net.Data 表 mytable2 に保管され、ほかの関数に渡すなどの処理を続行するために使用することができます。結果セット mytable3 は、Net.Data のデフォルトのレポート形式設定を使用して表示することができます。これは、この結果セットに対して、REPORT ブロックを指定しなかったからです。

結果セットでの DataLink URL のコード化

DataLink データ型は、データベース・ファイルに保管できるデータの型を拡張するための、基本作成ブロックです。DataLink の場合、列に保管される実際のデータは、ファイルを指すポインターだけです。このファイルは、イメー

ジ・ファイル、音声記録方式、またはテキスト・ファイルのいずれの型のファイルでも構いません。DataLinks は URL を保管して、ファイルのロケーションを解決します。

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

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

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

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

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

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

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

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

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

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

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

リレーショナル・データベース言語環境の例

以下の例では、マクロからリレーショナル・データベース言語環境を呼び出す
方法を示しています。

ODBC

次の例は、ODBC 言語環境に対して複数の関数を定義し、それらを呼
び出しています。

```
%DEFINE {
  DATABASE="qesql1"
  SHOWSQL="YES"
  table="int_null"
  LOGIN="netdata1"
  PASSWORD="ibmdb2"%}

%function(dtw_odbc) sql1() {
  create table int_null (int1 int, int2 int)
%}

%function(dtw_odbc) sql2() {
  insert into $(table) (int1) values (111)
%}

%function(dtw_odbc) sql3() {
  insert into $(table) (int2) values (222)
%}

%function(dtw_odbc) sql4() {
  select * from $(table)
```

```
%}

%function(dtw_odbc) sql5() {
drop table $(table)
%}

%HTML(REPORT) {
@sql1()
@sql2()
@sql3()
@sql4()
%}
```

Oracle

以下の例は、Oracle のデータベース `udatabase` を照会する、関数定義が `DTW_ORA` のマクロを表すもので、変数参照を使用して、照会すべきデータベース・テーブルを決定します。FUNCTION ブロックには、エラー条件を処理する MESSAGE ブロックも含まれます。Net.Data がマクロを処理する場合は、デフォルトのレポートがブラウザーに表示されます。

```
%DEFINE {
    LOGIN="ulogin"
    PASSWORD="upassword"
    DATABASE=""
    table= "utable"
%}

%FUNCTION(DTW_ORA) myQuery(){
select ename,job,empno,hiredate,sal,deptno from $(table) order by ename
%}
%MESSAGE{
100 : "<b>WARNING</b>: No employee were found that met your search criteria.<p>"
      : continue
%}

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

SQL

以下の例は、DTW_SQL 関数定義を持つ、SQL ストアード・プロシージャを呼び出すマクロを示しています。この例は、データ型が異なる 3 つのパラメーターを持っています。DTW_SQL 言語環境は各パラメーターを、そのデータ型に従ってストアード・プロシージャに渡します。ストアード・プロシージャが処理を完了すると、出力パラメーターが戻され、Net.Data はそれに応じて変数を更新します。

```
%{ *****
                                DEFINE BLOCK
*****%}
%DEFINE {
    MACRO_NAME      = "TEST ALL TYPES"
    DTW_HTML_TABLE  = "YES"
```

```

Procedure      = "TESTTYPE"
parm1          = "1"                %{SMALLINT      %}
parm2          = "11"               %{INT           %}
parm3          = "1.1"              %{DECIMAL (2,1) %}
%}

%FUNCTION(DTW_SQL) myProc
  (INOUT SMALLINT parm1,
   INOUT INT      parm2,
   INOUT DECIMAL(2,1) parm3){
CALL $(Procedure)
%}
%HTML(REPORT) {
<head>
<title>Net.Data : SQL Stored Procedure: Example '$(MACRO_NAME)'. </title>
</head>
<body bgcolor="#bbffff" text="#000000" link="#000000">
<p><p>
Calling the function to create the stored procedure.
<p><p>
@CRTPROC()
<hr>
<h2>
Values of the INOUT parameters
prior to calling the stored procedure:<p>
</h2>
<b>parm1 (SMALLINT)</b><br />
$(parm1)<p>
<b>parm2 (INT)</b><br />
$(parm2)<p>
<b>parm3 (DECIMAL)</b><br />
$(parm3)<p>
<p>
<hr>
<h2>
Calling the function that executes the stored procedure.
</h2>
<p><p>
@myProc(parm1,parm2,parm3)
<hr>
<h2>
Values of the INOUT parameters after
calling the stored procedure:<p>
</h2>
<b>parm1 (SMALLINT)</b><br />
$(parm1)<p>
<b>parm2 (INT)</b><br />
$(parm2)<p>
<b>parm3 (DECIMAL)</b><br />
$(parm3)<p>
</body>
%}

```


フラット・ファイル・インターフェース言語環境

データ・ソースとしてフラット・ファイル (すなわち平文ファイル) を使用する場合は、フラット・ファイル・インターフェース (FFI) と、それに関連付けられている Web サーバー上のファイルを、オープン、クローズ、読み取り、書き込み、そして削除するための関数を使用します。ファイル言語サポートは、ブラウザから Web クライアントの要求が発生すると、FFI 関数を使用して Web サーバー上のファイルに対して読み取り、または書き込みを行います。FFI はファイルをレコード・ファイルとして表示します。レコードは Net.Data マクロ表変数の行と等価であり、レコードの値は、Net.Data マクロ表変数のフィールドの値と等価です。FFI はファイルからレコードを Net.Data マクロ表の行に読み込み、行を表からレコードに書き込みます。

FFI 組み込み関数の説明と構文については、*Net.Data* 解説書 を参照してください。

FFI 言語環境の構成

以下の構成ステートメントが、初期設定ファイルに 1 行で記述されていることを確認します。

```
ENVIRONMENT (DTW_FILE) DTWFILE ( OUT RETURN_CODE )
```

Net.Data 初期設定ファイル、および言語環境の ENVIRONMENT ステートメントの詳細は、29ページの『環境構成ステートメント』を参照してください。

FFI 組み込み関数の呼び出し

FFI 関数の呼び出しは、他の関数の呼び出しと同じです。DEFINE ステートメントを使用して、渡したいパラメーターを変数として定義します。たとえば次のように定義します。

```
%DEFINE {  
    myFile = "c:/private/myfile"  
    myTable = %TABLE  
    myWait = "1500"  
    myRows = "2"  
%}
```

次に関数呼び出しステートメントを使用して関数を呼び出します。たとえば次のようにします。

```
@DTWF_UPDATE(myFile, "Delimited", "|", myTable, myWait, myRows)
```

例

この例では、Net.Data は ffi001.dat の内容を Net.Data の表に読み込み、この表の内容を tmp.dat ファイルに書き込みます。最後に、Net.Data は tmp.dat ファイルを削除します。

```

%DEFINE {
mytable = %TABLE(ALL)
myfile = "/usr/lpp/netdata/ffi//ffi001.dat"
tmpfile = "/usr/lpp/netdata/ffi/tmp.dat"
%}
%HTML(report){
@DTWF_READ(myfile, "ASCIITEXT", " ", mytable)
@DTW_TB_TABLE(mytable)

@DTWF_WRITE(tmpfile, "ASCIITEXT", " ", mytable)
@DTW_TB_TABLE(mytable)

@DTWF_REMOVE(tmpfile)
%}

```

Web レジストリー言語環境

Net.Data の Web レジストリーは、アプリケーションに関連するデータ用の永続的な記憶域を提供します。 Web レジスターには、実行時に Web ベースのアプリケーションが動的にアクセスできる構成情報、およびその他のデータが保管されます。 Web レジストリーには、Net.Data と Web レジストリーの組み込みサポートを使用する Net.Data マクロを介するか、この目的のために作成された CGI プログラムからしかアクセスできません。 Web レジストリーは、オペレーティング・システムのサブセットで使用できます。 Web レジストリーの組み込み関数の説明と構文、およびこの言語環境をサポートするオペレーティング・システムのリストについては、*Net.Data 解説書* を参照してください。

標準的な Web ページの開発では、URL は直接 Web ページの HTML ソースに挿入しなければなりません。このため、リンクの変更が難しくなります。また、静的な性格のため、Web ページに簡単に挿入できるリンクのタイプも制限されます。 Web レジストリーを使用して、アプリケーションに関連するデータ（たとえば URL）を保管しておけば、動的に設定されたリンクを持つ HTML のページを作成するのに役に立ちます。

レジストリーへの情報の保管およびその保守は、レジストリーへの書き込みアクセス権を持つアプリケーション開発者、あるいは Web 管理者が行うことができます。アプリケーションは、それに関連付けられているレジストリーから実行時に情報を取得します。これにより、柔軟なアプリケーションを設計でき、アプリケーションとサーバーの移動も可能になります。Net.Data のマクロを使用すると、動的に設定されたリンクを持つ HTML のページを作成できます。

情報は Web レジストリーにレジストリー項目の形で保管されます。各レジストリー項目は、RegistryVariable スtringと、それに対応する RegistryData

ストリングという、ペアの文字ストリングで構成されます。ペアのストリングで表示できる情報はすべて、レジストリー項目として保管できます。 `Net.Data` は、この変数ストリングを、レジストリーの特定の項目を見つけ、取得するための検索キーとして使用します。

表10 は、Web レジストリーのサンプルを表示したものです。

表 10. Web レジストリーのサンプル

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

Web レジストリーの使用を考慮すべき理由：

- Web レジストリーを使用すると、サーバーおよび URL の別名を保管できます。そのため、アプリケーションとサーバーの再配置が可能になります。
- アプリケーション開発者は、URL などのデータをレジストリーにあらかじめ定義して、Web ベースのアプリケーションを出荷できます。エンド・ユーザーはそのレジストリー・データを変更し、アプリケーションの振る舞いを変更することができます。
- Web レジストリーを使用すると、プロダクト名、各言語、製造者などに基づき、URL 検索を行うことができます。

Web レジストリーの索引付け項目は、`Index` ストリングが付加された `RegistryVariable` ストリングを持つ項目で、次の構文を使用します。

`RegistryVariable/Index`

ユーザーは、個々のパラメーターの索引ストリングの値を、索引付け項目を操作するように設計された組み込み関数に与えます。複数の索引付けレジストリー項目は、同じ `RegistryVariable` ストリング値を持つことができますが、異なる `Index` ストリング値を持つことにより、各項目を一意的に識別することができます。

表 11. 索引付け Web レジストリーのサンプル

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

上の 2 つの索引付け項目は、同じ Smith という RegistryVariable スtring 値を持っていますが、それぞれ異なる Index String を持っています。これらの項目は、Web レジストリー関数では異なる 2 つの項目として処理されます。

Web レジストリー言語環境の構成

以下の構成ステートメントが、初期設定ファイルに 1 行で記述されていることを確認します。

```
ENVIRONMENT (DTW_WEBREG) DTWWEB (OUT RETURN_CODE )
```

Net.Data 初期設定ファイル、および言語環境の ENVIRONMENT ステートメントの詳細は、29ページの『環境構成ステートメント』を参照してください。

Web レジストリーの組み込み関数の呼び出し

Web レジストリー関数を、他の関数を呼び出すのと同じように呼び出します。DEFINE ステートメントを使用して、渡したいパラメーターを変数として定義します。たとえば、以下のような場合です。

```
%DEFINE {  
    name = "smith"  
%}
```

次に関数呼び出しステートメントを使用して関数を呼び出します。たとえば次のようにします。

```
@DTWR_ADDENTRY("URLLIST", name, "http://www.ibm.com/software/",  
    "WORK_URL")
```

例

次の例では Web レジストリーを作成し、記入項目を追加しています。その後、記入項目を含むレポートを表示します。

```
%DEFINE {  
    RegTable = %TABLE(ALL)  
%}  
%MESSAGE {  
    default:"<p>Function Error: Return code: $(RETURN_CODE)." :continue  
%}  
  
%FUNCTION(DTW_WEBREG) ListTable(INOUT RegTable) {  
%}  
  
%HTML(report){  
    @DTWR_CREATEREG("MYREG")  
    @DTWR_ADDENTRY("MYREG", "Dept. 1", "Payroll")  
    @DTWR_ADDENTRY("MYREG", "Dept. 2", "Technical Support")  
    @DTWR_ADDENTRY("MYREG", "Dept. 3", "Research")  
    @DTWR_LISTREG("MYREG", RegTable)
```

```
<p>Report:<br />
@ListTable(RegTable)

%}
```

プログラミング言語環境

Net.Data は、外部プログラムを呼び出す際に使用するための、以下の言語環境を提供しています。

- 『Java アプレット言語環境』
- 201ページの『Java アプリケーション言語環境』
- 205ページの『Perl 言語環境』
- 208ページの『REXX 言語環境』
- 212ページの『System 言語環境』

アクセス権：Net.Data を実行するときのユーザー ID が、プログラムがアクセスする可能性のあるオブジェクトを含めて、プログラムを実行するためのアクセス権を持っていることを確認してください。詳しくは、69ページの『Net.Data がアクセスするファイルへのアクセス権の授与』を参照してください。

Java アプレット言語環境

Java アプレット言語環境では、Net.Data アプリケーションに、Java アプレット用の HTML タグを簡単に生成することができます。Java アプレット言語環境を呼び出す際に、アプレット名を指定し、そのアプレットが必要とするパラメーターを渡します。Java アプレット言語環境は、マクロを処理し、HTML アプレット・タグを生成します。Web ブラウザーはこのタグを使用して、アプレットを実行します。

さらに Net.Data は、アプレットが表パラメーターにアクセスするのに使用可能な、インターフェースのセットを提供します。これらのインターフェースは DTW_Applet.class というクラスに含まれています。

以下のセクションでは、Java アプレット言語環境を使用して、Java アプレットを実行する方法について説明しています。

Java アプレット言語環境の構成

以下の構成ステートメントが、初期設定ファイルに 1 行で記述されていることを確認します。

```
ENVIRONMENT (DTW_APPLET) DTWJAVA ( OUT RETURN_CODE )
```

Net.Data 初期設定ファイル、および言語環境の ENVIRONMENT ステートメントの詳細は、29ページの『環境構成ステートメント』を参照してください。

Java アプレットの作成

Net.Data の Java アプレット言語環境を使用する前に、どのアプレットを使用するのか、あるいはどのアプレットを作成する必要があるのかを決定する必要があります。アプレットの作成の詳細については、Java の資料を参照してください。

アプレット・タグの生成

アプレット言語環境は、Net.Data の関数呼び出しを使って呼び出します。関数呼び出しには宣言は不要です。関数呼び出しの構文を以下に示します。

```
@DTWA_AppletName(param1, param2, ..., paramN)
```

- DTWA_ は、アプレット言語環境への関数呼び出しを識別します。
- AppletName は、タグが生成されるアプレットの名前です。
- param1 から paramN は、PARAM タグを生成するのに使用されるパラメータです。

アプレット・タグを生成するマクロの作成方法：

1. アプレットに必要なパラメータをすべて、マクロの DEFINE セクションに定義する。これらのパラメータには、アプレットの入力として必要な、アプレット・タグの属性、Net.Data の変数、および Net.Data の表パラメータをすべて含めます。たとえば、以下のような場合です。

```
%define{
DATABASE = "celdial"                <=Name of the database
MyGraph.codebase = "/netdata-java/" <=Required applet attribute
MyGraph.height = "200"              <=Required applet attribute
MyGraph.width = "400"               <=Required applet attribute
MyTitle = "Celdial results"         <=Name of the Web page
MyTable = %TABLE(all)               <=Table to store query results
%}
```

2. オプション：アプレットの入力としての結果セットを生成するための、データベースへの照会を指定する。これは、図あるいは表を生成するアプレットを使用する場合に役に立ちます。たとえば、以下のような場合です。

```
%FUNCTION(DTW_SQL) mySQL(OUT table){
select name, ages from ibmuser.guests
%}
```

3. Java アプレット言語環境を呼び出し、アプレットを起動するための関数呼び出しを Net.Data マクロに指定する。関数呼び出しは、アプレットの名前と、言語環境に渡したいパラメータを指定します。これらのパラメータ

には、アプレットの入力として必要な Net.Data 変数、および Net.Data の表または列パラメーターをすべて含めます。

たとえば、以下のような場合です。

```
%HTML(report){
@mysql(MyTable)                                <=A call to mysql
@DTWA_MyGraph(MyTitle, DTW_COLUMN(ages) MyTable) <=Applet function call
%}
```

アプレット・タグの属性: アプレット・タグの属性は Net.Data マクロの任意の位置に指定できます。Net.Data は、*AppletName.attribute* という形式を持つすべての変数を、属性としてアプレット・タグに置き換えます。属性をアプレット・タグに定義するための構文を以下に示します。

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

以下に示した属性はすべてのアプレットで必須です。

- *codebase*: アプレットの場所で、URL で識別されます。
- *height*: アプレットの高さをピクセルで指定します。
- *width*: アプレットの幅をピクセルで指定します。

以下の属性はオプションです。

- *align*: アプレットの配置を指定します。
- *alt*: ブラウザーが APPLET タグを理解できても、Java アプレットを実行できない場合に表示するテキストを指定します。
- *archive*: クラスおよびその他のリソースを含む、アーカイブを指定します。
- *hspace*: アプレットの両側をピクセル数で指定します。
- *name*: アプレット・インスタンスの名前を指定します。
- *object*: アプレットのシリアライズ表示を含む、ファイルの名前を指定します。
- *vspace*: アプレットの上下のピクセル数を指定します。

たとえば、MyGraph という名前のアプレットに対しては、これら必須の属性を以下のように定義できます。

```
%DEFINE{
MyGraph.codebase = "/netdata-java/"
MyGraph.height   = "200"
MyGraph.width    = "400"
%}
```

実際の指定は `DEFINE` セクションにある必要はありません。値は `DTW_ASSIGN` 関数で設定できます。 `AppletName.code` 変数に変数を定義しない場合は、 `Net.Data` はデフォルトの `code` パラメーターを、アプレット・タグに追加します。 `code` パラメーターの値は `AppletName.class` です。ここで、 `AppletName` はアプレットの名前です。

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

- `Net.Data` 変数 (`LIST` 変数を含む)
- `Net.Data` 表
- `Net.Data` 表の列

パラメーターを渡すと、 `Net.Data` は、パラメーターに割り当てた名前と値を持つ Java アプレットの `PARAM` タグを、 `HTML` 出力に作成します。文字列リテラルまたは関数呼び出しの結果は渡すことができません。

***Net.Data* の変数パラメーター:**

`Net.Data` 変数はパラメーターとして使用することができます。変数をマクロの `DEFINE` ブロックで定義し、その変数の値を `DTWA_AppletName` 関数呼び出しで渡すと、 `Net.Data` は、その変数と同じ名前と値を持つ `PARAM` タグを生成します。たとえば、次のマクロ・ステートメントが与えられたとします。

```
%define{  
  
...  
  
MyTitle = "This is my Title"  
%}  
  
%HTML(report){  
  @DTWA_MyGraph( MyTitle, ...)  
%}
```

`Net.Data` は次のアプレット・タグ `PARAM` を作成します。

```
<param name = 'MyTitle' value = "This is my Title" />
```

***Net.Data* の表パラメーター:**

`Net.Data` は、Java アプレット言語環境が呼び出されるたびに、 `DTW_NUMBER_OF_TABLES` という名前の `PARAM` タグを生成し、関数呼び

出しが何らかの表変数を渡したかどうかを指定します。値は `Net.Data` が関数で使用する表変数の数です。関数呼び出しに表変数が指定されていない場合は、以下のタグが生成されます。

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

関数呼び出しでは、1 つ以上の `Net.Data` の表変数を渡すことができます。`Net.Data` の表変数を `DTWA_AppletName` 関数呼び出しで指定すると、`Net.Data` は以下の `PARAM` タグを生成します。

表名パラメーター・タグ：

このタグは渡す表の名前を指定します。このタグの構文は、以下のとおりです。

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

ここで、`i` は関数呼び出しの順序に基づく表の番号です。また、`tname` は表の名前です。

行および列の仕様パラメーター・タグ：

`PARAM` タグは特定の表の行と列の数を指定するために生成されます。このタグの構文は、以下のとおりです。

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

ここで、表の名前は `tname` 、 `rows` は表の行の数、そして `cols` は表の列の数です。このタグのペアは、関数呼び出しで指定された固有の表ごとに生成されます。

列値パラメーター・タグ：

この `PARAM` タグは、特定の列の列名を指定します。このタグの構文は、以下のとおりです。

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

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

行値パラメーター・タグ：

この `PARAM` タグは、特定の行と列にある値を指定します。このタグの構文は、以下のとおりです。

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

ここで、表名は `tname` 、 `cname` は列名、`k` は行番号、そして `val` は対応する行と列の値に一致する値です。

表列パラメーター: 表列を関数呼び出し時にパラメーターとして渡すことにより、特定の列のタグを生成することができます。Net.Data は、指定された列に対してのみ対応するアプレット・タグを生成します。表列パラメーターは次の構文を使用します。

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

ここで、*x* は表の列の名前または番号です。

表列パラメーターは、表パラメーターに対して定義された同じアプレット・タグを使用します。

Java に非対応のブラウザでのアプレット・タグの代替テキスト: 変数 DTW_APPLET_ALTTEXT は、Java をサポートしていないブラウザ、あるいは Java のサポートをオフにしているブラウザに表示するテキストを指定します。たとえば、次の変数定義

```
%define DTW_APPLET_ALTTEXT = "<p>Sorry, your browser is not Java-enabled."</p>
```

は、以下の HTML タグとテキストを作成します。

```
<p>Sorry, your browser is not Java-enabled.</p><
```

この変数が定義されていない場合は、代替テキストは表示されません。

Java アプレットの例

次の例は、Java アプレット言語環境を呼び出す Net.Data マクロと、呼び出された Java アプレット言語環境によって生成されたアプレット・タグを示しています。

Net.Data マクロには、Java アプレット言語環境への次の関数呼び出しが含まれています。

```
%define{
DATABASE = "celdial"
DTW_APPLET_ALTTEXT = "<p>Sorry, your browser is not Java-enabled."</p>
DTW_DEFAULT_REPORT = "no"
MyGraph.codebase = "/netdata-java/"
MyGraph.height   = "200"
MyGraph.width    = "400"
MyTitle = "This is my Title"
%}
%FUNCTION(DTW_SQL) mySQL(OUT table){
select name, ages from ibmuser.guests
%}
%HTML(report){
@mySQL(MyTable)
@DTWA_MyGraph(MyTitle, DTW_COLUMN(ages) MyTable)
%}
```

DEFINE セクションの Net.Data マクロの行は、アプレット・タブの属性を指定しています。

```
MyGraph.codebase = "/netdata-java/"
MyGraph.height   = "200"
MyGraph.width    = "400"
```

この言語環境は、次の限定子を持つアプレットを生成します。

```
<applet code='MyGraph.class'
        codebase='/netdata-java/'
width='400'
height='200'
>
```

Net.Data は、Net.Data マクロの SQL セクションから、SQL 照会結果を出力表 MyTable に戻します。この表は次のように DEFINE セクションで指定されています。

```
MyTable = %TABLE(all)
```

マクロでのアプレットの呼び出しは、次のように HTML セクションで指定されます。

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

関数呼び出しのパラメーターに基づき、Net.Data は、結果表に関する情報（たとえば、戻される列と行の数、結果行など）を含む完全なアプレット・タグを生成します。Net.Data は、次の例のように、結果表のセルごとに 1 つのパラメーター・タグを生成します。

```
<param name = 'DTW_MyTable_ages_VALUE_1' value = "35" />
```

パラメーター名 *DTW_MyTable_ages_VALUE_1* は、表 MyTable の表セル（行 1、列 ages）を指定します。このセルの値は 4 です。アプレットへの関数呼び出しの際のキーワード DTW_COLUMN は、必要なのは結果表 MyTable の列 ages だけであることを指定します。

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

以下の出力は、この例に対して Net.Data が生成する完全なアプレット・タグを示しています。

```
<applet code='MyGraph.class' codebase='/netdata-java/'
        width='400'           height='200'
>
<param name = 'MyTitle' value = "This is my Title" />
<param name = 'DTW_NUMBER_OF_TABLES' value = "1" />
<param name = 'DTW_TABLE_1_NAME' value = "MyTable" />
<param name = 'DTW_MyTable_NUMBER_OF_ROWS' value = "5" />
<param name = 'DTW_MyTable_NUMBER_OF_COLUMNS' value = "1" />
```

```

<param name = 'DTW_MyTable_COLUMN_NAME_1' value = "ages" />
<param name = 'DTW_MyTable_ages_VALUE_1' value = "35" />
<param name = 'DTW_MyTable_ages_VALUE_2' value = "32" />
<param name = 'DTW_MyTable_ages_VALUE_3' value = "31" />
<param name = 'DTW_MyTable_ages_VALUE_4' value = "28" />
<param name = 'DTW_MyTable_ages_VALUE_5' value = "40" />
<p>Sorry, your browser is not Java-enabled.</p>
</applet>

```

Net.Data の Java アプレット・インターフェースの使い方

Net.Data は、DTW_Applet.class と呼ばれるインターフェース・セットを提供しています。これを Java アプレットで使用すると、表変数に対して生成される PARAM タグを処理するのに役に立ちます。このインターフェースを拡張するアプレットを作成して、アプレットからそのルーチンを呼び出すことができます。

Net.Data が提供するインターフェースには以下のものがあります。

- **int GetNumberOfTables()** は、アプレット・タグで検出された表の数を返します。
- **String [] GetTableNames()** は、アプレット・タグで検出された表名のリストを返します。
- **int GetNumberOfColumns(String table_name)** は、表 table_name の列の数を返します。
- **int GetNumberOfRows(String table_name)** は、表 table_name の行の数を返します。
- **String[] GetColumnNames(String table_name)** は、表 table_name の列の名前を返します。
- **String[][] GetTable(String table_name)** は、表の行と列の値を格納するストリングの 2 次元配列を返します。

インターフェースにアクセスするには、EXTENDS キーワードをアプレットのコードで使用して、アプレットを DTW_APPLET クラスからサブクラス化します。その例を、以下に示します。

```

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

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

        if (GetNumberOfTables() > 0)
        {

```

```

        String [] tables = GetTableNames();
        printTables(tables);
    }

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

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

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

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

        String [] col_names = GetColumnNames(table_name);

        System.out.println("-----");

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

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

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

```

Java アプリケーション言語環境

Net.Data は、Java 言語環境により既存の Java アプリケーションをサポートします。Java アプレットおよび Java メソッド (またはアプリケーション) をサポートすることで、Java データベース・コネクティビティ (JDBC**) API から DB2 へのアクセスが可能です。

JDBC に関する詳細は、以下の Web サイトで入手できます。

- IBM ソフトウェアには JDK 1.1 以降が含まれます。これは Net.Data で JDBC を使用するにあたり必須です。

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

- JavaSoft には追加 JDBC ドライバー、JDBC API 資料、および最新の JDBC アップデートが含まれます。

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

Java 言語環境の構成

Java 言語環境を使用するには、Net.Data 初期設定を検証し、言語環境を設定する必要があります。

以下の構成ステートメントが、初期設定ファイルに 1 行で記述されていることを確認します。

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

Net.Data 初期設定ファイル、および言語環境の ENVIRONMENT ステートメントの詳細は、29ページの『環境構成ステートメント』を参照してください。

重要：Java 言語環境の設定方法については、34ページの『Java 言語環境のセットアップ』を参照してください。

Java 関数の呼び出し

Java 言語環境には、リモート・プロシーチャー呼び出し (RPC) と同様のインターフェースが提供されています。Net.Data スtring をパラメーターに持つ Net.Data マクロから、Java 関数呼び出しが発行でき、起動した Java 関数から String が戻されます。Java 言語環境を使用する際は、Net.Data Live Connection を使用する必要があります (Live Connection に関する詳細は、218ページの『接続管理』を参照)。

Java 関数の呼び出し方法

1. Java 関数を記述する。
2. すべての Java 関数について、Net.Data クライエントを作成する (Net.Data クライエントは、Java 仮想マシンを立ち上げて Java 関数を実行します)。
3. Live Connection 構成ファイルの Java ENVIRONMENT ステートメントに、クライエントを定義する。

新規の Java 関数を追加する度に、Java クライエントを再度作成する必要があります。

4. 接続管理プログラムを開始する。
5. Net.Data マクロを実行して、Java 言語環境を起動する。

Java 関数の作成: Java 関数サンプル・ファイル UserFunctions.java を変更するか、または以下の myfile.java という名前のサンプル・ファイルをモデルに、新規ファイルを作成します。

```
=====myfile.java=====
import mypackage.*
public String myfctcall(...parameters from macro...)
{
    return ( mypackage.mymethod(...parameters...));
}
public String lowlevelcall(...parameters...)
{
    string result;
    .....code using many functions of your package...
    return(result)
}
```

Java 言語環境ファイル構造: Net.Data は、Net.Data のインストール中に複数のディレクトリーを作成します。これらのディレクトリーには、以下のよう
な、Java 関数の作成、クライエットの定義、および Java 言語環境でのマクロ
の実行に必要なファイルが含まれています。

- UserFunctions.java という名前のサンプル Java 関数。
- makeClas という名前のサンプル・ファイル。このファイルを実行すると、Java 関数に対する Net.Data クライエット・クラスが作成されます。
- launchjv という名前のサンプル・ファイル。 Net.Data クライエットはこのファイルを使用して、 Java 仮想マシンを立ち上げ Java 関数を実行しま
す。

表12 には、ご使用のオペレーティング・システムにおけるファイルの、ディレ
クトリーおよびファイル名が記述されています。

表 12. Java 関数作成用ファイル

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

Java 言語環境クライアントの定義: サンプル・ファイル `makeClas.bat` を変更するか、または新規の `.bat` ファイルを作成し、すべての Java 関数について、`dtw_samp.class` という名前の `Net.Data` クライエント・クラスを生成します。以下の例に、バッチ・ファイル `CreateServer` による 3 つの Java 関数の処理方法を示します。

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

バッチ・ファイルは、`Net.Data` に提供される `Stub.java` という名前のスタブ・ファイルに加えて、以下のファイルを処理して `dtw_samp.class` を作成します。

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

JDBC アプリケーションまたはアプレットの作成は、DB2 CLI または ODBC を使用した、データベース・アクセス用 C アプリケーションの作成に類似しています。アプリケーションとアプレットの主な違いとして、アプリケーションは DB2 との通信に特別なソフトウェア（たとえば、DB2 クライアント・アプリケーション・イネーブラー）を必要とする場合があります。アプレットは Java を使用可能な Web ブラウザーに依存し、DB2 コードがクライアントにインストールされている必要はありません。

JDBC を使用する前に、システムの構成を行う必要があります。これらの考慮事項については、DB2 JDBC アプリケーションおよびアプレット・サポートの Web サイトで説明されています。

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

Java 言語環境の例

Java 関数の作成、クライアント・クラスの定義、および `Net.Data` の構成が終了したら、Java 関数への参照を含むマクロを実行することができます。

重要： `Net.Data` マクロを起動する前に、接続管理プログラムを開始してください。

以下の例では、関数呼び出しの `myfctcall` が、クライアント `DTW_JAVAPPS` を使用して、`Net.Data` に提供されるサンプル関数を呼び出します。

```
%FUNCTION (DTW_JAVAPPS) myfctcall( ....parameters from macro ....)
```

```
%{ to call the sample provided with Net.Data %}
%FUNCTION (DTW_JAVAPPS) reverse_line(str);
```



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

Perl 言語環境

Perl 言語環境は、Net.Data マクロの FUNCTION ブロックで指定した、インラインの Perl スクリプトを解釈することができます。あるいは、サーバー上の別ファイルに保管されている外部の Perl スクリプトを処理することができます。

Perl 言語環境の構成

以下の構成ステートメントが、Net.Data の初期設定ファイルに 1 行で記述されていることを確認してください。

```
ENVIRONMENT (DTW_PERL)      DTWPERL    ( OUT RETURN_CODE )
```

Net.Data 初期設定ファイル、および言語環境の ENVIRONMENT ステートメントの詳細は、29ページの『環境構成ステートメント』を参照してください。

日本語ユーザー: 日本語 SJIS 文字セットで表示される文字の中には、制御文字として Perl が誤って解釈しているものがあります。この問題を解決するには jperl というオープン・ソース・パッケージを使用します。パッケージをダウンロードしてインストールし、Perl スクリプトのヘッダーに `use I18N::Japanese.pm` というステートメントを組み込んでください。

外部 Perl スクリプトの呼び出し

外部 Perl スクリプトの呼び出しは、EXEC ステートメントにより FUNCTION ブロックで識別されます。使用する構文は以下のとおりです。

```
%EXEC{ perl_script_name [optional parameters] %}
```

必須: Perl のスクリプト名 *perl_script_name* が、Net.Data の初期設定ファイルの EXEC_PATH 構成変数に対して指定されているパスに、リストされていることを確認してください。

```
%FUNCTION(DTW_PERL) perl1() {
%EXEC{MyPerl.pl %}
%}
```

パラメーターを渡す

Perl によって呼び出されるプログラムに情報を渡すには、2 つの方法があります (DTW_PERL 言語環境では、直接渡しと間接渡し)。

直接 Perl スクリプトの呼び出し時にパラメーターを直接渡します。たとえば、以下のような場合です。

```
%DEFINE INPARAM1 = "SWITCH1"

%FUNCTION(DTW_PERL) sys1() {
    %EXEC{
        MyPerl.pl $(INPARAM1) "literal string"
    }
}
```

Net.Data 変数 INPARAM1 が参照され、Perl スクリプトに渡されます。パラメーターを Perl スクリプトに渡す方法は、Perl スクリプトがコマンド行から呼び出された場合に、Perl スクリプトにパラメーターを渡すのと同じです。この方法で渡されるパラメーターは、入力タイプのパラメーターと考えられます (Perl スクリプトに渡されるパラメーターは、Perl スクリプトで使い、操作することができますが、パラメーターの変更は Net.Data には反映されません)。

間接

以下に示す方法のいずれかを使用して、Perl スクリプトの呼び出し時に間接的にパラメーターを渡します。

- Net.Data が入力パラメーターを環境変数として、Perl スクリプトに渡すようにする。これにより、Perl スクリプトは環境変数からパラメーターを取得することができます。
- Perl スクリプトが出力パラメーターを言語環境に戻すようにする。そのためには、Net.Data が環境変数 DTWPIPE に格納して渡す名前を持つ、名前付きパイプにデータを書き込みます。以下の構文を使用して、名前付きパイプにデータを書き込みます。

```
name="value"
```

データ項目が複数ある場合は、各項目を改行あるいはブランク文字で区切ります。

変数名が出力パラメーターと同じ名前上記の構文を使用する場合、現行値は新規値により置き換えられます。変数名が出力パラメーターに対応しない場合は、Net.Data はその変数を無視します。

以下の例では、Net.Data がマクロの変数を渡す方法を示しています。

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

```
%}
%HTML(INPUT) {
    @today()
%}
```

Perl スクリプトが `today.pl` と呼ばれる外部ファイルにある場合は、次の例にある関数と同じ関数を作成することができます。

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

Net.Data の表は、Perl 言語環境で呼び出される Perl スクリプトに渡すことができます。Perl スクリプトは、Net.Data 名によって、Net.Data のマクロの表パラメーターの値にアクセスします。列見出しおよびフィールド値は、表名および列番号により識別される変数に含まれます。たとえばテーブル `myTable` では、列見出しは `myTable_N_j` で、フィールド値は `myTable_V_i_j` です。ここで、*i* は行番号、*j* は列番号です。テーブルの行および列番号は、`myTable_ROWS` および `myTable_COLS` です。

FUNCTION ブロックの REPORT ブロックと MESSAGE ブロック

REPORT および MESSAGE ブロックは、どの FUNCTION セクションにおいても同じように使用することができます。これらのブロックは Net.Data によって処理され、言語環境では処理されません。ただし、Perl スクリプトはテキストを標準出力ストリームに書き込み、Web ページの一部として組み込むことはできます。

Perl 言語環境の例

次の例は、Net.Data が外部 Perl スクリプトを実行して、どのように表を生成するかを示しています。

```
%define {
    c = %TABLE(20)
    rows = "5"
    columns = "5" %}
%function(DTW_PERL) genTable(in rows, in columns, out table) {
    %exec{ perl.pl
%}

%message{
    default: "genTable: Unexpected Error"
%}
%}

%HTML(REPORT) {
    @genTable(rows, columns, c)
    return code is $(RETURN_CODE)
```

```
%}
The Perl script (perl.pl):

open(D2W,"> $ENV{DTWPIPE}");
print "genTable begins ...

";
$r = $ENV{ROWS};
$c = $ENV{COLUMNS};
print D2W "table_ROWS=¥"$r¥" ";
print D2W "table_COLS=¥"$c¥" ";
print "rows: $r
";
print "columns: $c";
for ($j=1; $j<=$c; $j++)
{
print D2W "table_N_$j=¥"COL$j¥" ";
}
for ($i=1; $i<=$r; $i++)
{
for ($j=1; $j<=$c; $j++)
{
print D2W "table_V_$i","_", "$j=¥" " $i $j "¥" ";
}
}
close(D2W);
```

結果 : genTable は以下を生成します。

```
rows: 5 columns: 5
COL1 | COL2 | COL3 | COL4 | COL5 |
-----
[ 1 1 ] | [ 1 2 ] | [ 1 3 ] | [ 1 4 ] | [ 1 5 ] |
-----
[ 2 1 ] | [ 2 2 ] | [ 2 3 ] | [ 2 4 ] | [ 2 5 ] |
-----
[ 3 1 ] | [ 3 2 ] | [ 3 3 ] | [ 3 4 ] | [ 3 5 ] |
-----
[ 4 1 ] | [ 4 2 ] | [ 4 3 ] | [ 4 4 ] | [ 4 5 ] |
-----
[ 5 1 ] | [ 5 2 ] | [ 5 3 ] | [ 5 4 ] | [ 5 5 ] |
-----
return code is 0
```

REXX 言語環境

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

REXX 言語環境の構成

REXX 言語環境を使用するには、Net.Data 初期設定を検証し、言語環境を設定する必要があります。

以下の構成ステートメントが、初期設定ファイルに 1 行で記述されていることを確認してください。

```
ENVIRONMENT (DTW_REXX)      DTWREXX    ( OUT RETURN_CODE )
```

Net.Data の初期設定ファイルと言語環境の ENVIRONMENT ステートメントの詳細については、29ページの『環境構成ステートメント』を参照してください。

REXX プログラムの実行

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

インライン REXX プログラムを実行するには、以下のようにします。

REXX (DTW_REXX) 言語環境を使用する関数で、関数の言語環境が実行可能なセクション内に REXX コードを含む関数を定義します。

例：インライン REXX プログラムを含む関数

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

外部 REXX プログラムを実行するには、以下のようにします。

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

例：外部プログラムを指す EXEC ステートメントを含む関数

```
%function(DTW_REXX) externalHelloWorld() {  
%EXEC{ helloworld.exe%}  
%}
```

必須：REXX ファイル名は、Net.Data の初期設定ファイルにある EXEC_PATH 構成変数に対して指定されているパスにリストされていることを確認してください。EXEC_PATH 構成変数の定義方法を理解するには、25ページの『EXEC_PATH』を参照してください。

パラメーターを REXX プログラムに渡す

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

直接渡し

%EXEC ステートメントを使用して、外部 REXX プログラムに直接パラメーターを渡します。たとえば、以下のような場合です。

```
%FUNCTION(DTW_REXX) rexx1() {  
  %EXEC{  
    CALL1.CMD $(INPARM) "literal string"  %}  
  %}
```

Net.Data 変数 INPARM1 が参照解除され、外部 REXX プログラムに渡されます。REXX プログラムは、REXX PARSE ARG 命令を使用して変数を参照することができます。このメソッドを使用してプログラムに渡されるパラメーターは、入力型パラメーターとみなされます(プログラムに渡されるパラメーターは、そのプログラムにより使用して操作できますが、パラメーターに対する変更は Net.Data に反映されません)。

間接渡し

REXX プログラムの 変数プール 経由で、パラメーターを間接的に渡します。REXX プログラムが開始すると、REXX インタープリターによりすべての変数に関する情報を含むスペースが作成され保守されます。このスペースは、変数プールと呼ばれます。

REXX 言語環境 (DTW_REXX) 関数が呼び出されると、REXX プログラムの実行の前に、入力 (IN) または入出力 (INOUT) となる関数仮引き数が REXX 言語環境により変数プールに保管されます。REXX プログラムは、起動されると、これらの変数に直接アクセスすることができます。REXX プログラムが正常に終了すると、DTW_REXX 言語環境は、OUT または INOUT 関数仮引き数があるかどうかを判別します。ある場合は、言語環境はこの関数パラメーターに対応する値を変数プールから検索し、この新しい値で関数パラメーター値を更新します。Net.Data は、制御を受け取ると、REXX 言語環境から取得した新しい値ですべての OUT または INOUT パラメーターを更新します。たとえば、以下のような場合です。

```
%DEFINE a = "3"  
%DEFINE b = "0"  
%FUNCTION(DTW_REXX) double_func(IN inpl, OUT outpl){  
  outpl = 2*inpl  
  %}  
  
%HTML(REPORT) {  
  Value of b is $(b), @double_func(a, b) Value of b is $(b)  
  %}
```

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

Net.Data の表を REXX プログラムに渡すことができます。REXX プログラムでは、Net.Data マクロの表パラメーターの値には REXX の stem 変数としてアクセスします。REXX プログラムでは、列見出しおよびフィールド値は、表名および列番号で識別される変数に含まれます。たとえば、表 `myTable` では、列見出しは `myTable_N.j` であり、フィールド値は `myTable_N.i.j` です。ここで、`i` は行番号、`j` は列番号です。この表の行数は `myTable_ROWS` であり、行数は `myTable_COLS` です。

AIX オペレーティング・システムのパフォーマンス向上

AIX システムで REXX 言語環境への呼び出しが多くある場合は、RXQUEUE_OWNER_PID 環境変数を 0 に設定してください。REXX 言語環境に多くの呼び出しを行うマクロは、多くのプロセスを spawn しやすくなり、システム資源が消費されます。

以下の 3 つの方法のいずれかで環境変数を設定します。

- DTW_SETENV 組み込み関数を使用して、マクロで設定する方法
`@DTW_rSETENV("RXQUEUE_OWNER_PID", "0")`
- AIX システム環境ファイルに以下のステートメントを挿入する方法
`/etc/environment: RXQUEUE_OWNER_PID = 0`

この方法は、マシン全体の REXX の振る舞いに影響を与えます。

- たとえば HTTP Web サーバー環境ファイル中、Domino Go Webserver 用に以下のステートメントを挿入します。

```
InheritEnv RXQUEUE_OWNER_PID = 0
```

この方法は、Web サーバーの REXX の振る舞いに影響を与えます。

REXX 言語環境の例

以下の例では、REXX 関数を呼び出して 2 つの列および 3 つの行を持つ Net.Data テーブルを生成するマクロを示しています。REXX 関数への呼び出しに続いて組み込み関数 `DTW_TB_TABLE()` が呼び出され、HTML 表を生成してブラウザーに戻します。

```
%DEFINE myTable = %TABLE
%DEFINE DTW_DEFAULT_REPORT = "NO"
```

```
%FUNCTION(DTW_REXX) genTable(out out_table) {
    out_table_ROWS = 3
    out_table_COLS = 2

    /* Set Column Headings */
    do j=1 to out_table_COLS
        out_table_N.j = 'COL'j
    end

    /* Set the fields in the row */
    do i = 1 to out_table_ROWS
        do j = 1 to out_table_COLS
            out_table_V.i.j = '[' i j ']'
        end
    end
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]

System 言語環境

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

System 言語環境の構成

以下の構成ステートメントを初期設定ファイルに 1 行で追加します。

```
ENVIRONMENT (DTW_SYSTEM) DTWSYS ( OUT RETURN_CODE )
```

Net.Data の初期設定ファイルと言語環境の ENVIRONMENT ステートメントの詳細については、29ページの『環境構成ステートメント』を参照してください。

コマンドの発行およびプログラムの呼び出し

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

```
%FUNCTION(DTW_SYSTEM) sys1() {
    %EXEC { ADDLIB.CMD %}
%}
```


EXEC_PATH 構成変数を使用して、オブジェクト (コマンドおよびプログラムなど) を含むディレクトリへのパスを定義すると、実行可能なオブジェクトへのパスを短縮することができます。 EXEC_PATH 構成変数の定義方法を理解するには、25ページの『EXEC_PATH』を参照してください。

例 1: コマンドを発行する

```
%FUNCTION(DTW_SYSTEM) sys2() {  
    %EXEC {MYPGM %}  
%}
```

例 2: プログラムの呼び出し

```
%FUNCTION(DTW_SYSTEM) sys3() {  
    %EXEC {MYPGM.EXE %}  
%}
```

プログラムにパラメーターを渡す

System (DTW_SYSTEM) 言語環境により起動するプログラムに情報を渡すには、直接および間接の 2 とおりの方法があります。

直接渡し

プログラムへの呼び出し時にパラメーターを直接渡す。たとえば、以下のような場合です。

```
%DEFINE INPARAM1 = "SWITCH1"  
  
%FUNCTION(DTW_SYSTEM) sys1() {  
    %EXEC{  
        CALL1.CMD $(INPARAM1) "literal string"  
    %}  
%}
```

Net.Data 変数 INPARAM1 が参照され、プログラムに渡されます。パラメーターはプログラムに渡されます。これはコマンド行からプログラムを呼び出す場合と同様に行われます。このメソッドを使用してプログラムに渡されるパラメーターは、入力型パラメーターとみなされます (プログラムに渡されるパラメーターは、そのプログラムにより使用して操作できますが、パラメーターに対する変更は Net.Data に反映されません)。

間接渡し

System 言語環境は Net.Data 変数を直接に渡すまたは検索することができないため、以下のようにしてプログラムに対し変数を使用可能にします。

- Net.Data は入力パラメーターを環境変数としてプログラムに渡します。するとプログラムは環境変数を通じてパラメーターを検索します。
- プログラムは、名前付きパイプ (名前は Net.Data が環境変数 DTWPIPE で渡します) に書き込みを行い、出力パラメーターを言語環境に渡します。以下の構文を使用して、名前付きパイプにデータを書き込みます。

```
name="value"
```

複数のデータ項目の場合、各項目を改行あるいはブランク文字で区切ります。

変数名が出力パラメーターと同じ名前であり、上記の構文を使用する場合、現行値は新規値により置き換えられます。変数名が出力パラメーターに対応しない場合は、Net.Data はその変数を無視します。

以下の例では、Net.Data がマクロから変数を渡す方法を示しています。

```
%FUNCTION(DTW_SYSTEM) sys1 (IN P1, OUT P2, P3) {
    %EXEC {
        UPDPGM
    %}
%}
```

Net.Data 表を、System 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、フィールド値は myTable_V_i_j です。ここで、i は行番号、j は列番号です。この表の行と列の数は myTable_ROWS と myTable_COLS です。

プロセスに対する環境変数の数には制限があるため、行が多い表を渡すことはお勧めできません。

System 言語環境の例

以下の例では、3 つのパラメーター P1、P2、および P3 を持つ関数定義を含むマクロを示しています。P1 は入力 (IN) パラメーター、P2 および P3 は出力 (OUT) パラメーターです。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 ブロックのステートメントを処理する前に、 DTW_SYSTEM 言語環境は P1 および対応する値を環境スペースに保管します。

```
%DEFINE {
    MYPARAM2 = "ValueOfParm2"
    MYPARAM3 = "ValueOfParm3"
}%
%FUNCTION(DTW_SYSTEM) sys1 (IN P1, OUT P2, P3) {
    %EXEC {
        UPDPGM
    }
}%

%HTML(upd1) {
<p>
    Passing data to a program. The current value
    of MYPARAM2 is "${MYPARAM2}", and the current value of MYPARAM3 is
    "${MYPARAM3}". Now we invoke the Web macro function.

    @sys1("ValueOfParm1", MYPARAM2, MYPARAM3)

<p>
    After the function call, the value of MYPARAM2 is "${MYPARAM2}",
    and the value of MYPARAM3 is "${MYPARAM3}".
}%
```

第7章 パフォーマンスを向上させる

パフォーマンスを向上させることは、システム・チューニングの重要な部分です。本章では、Net.Data のパフォーマンスを向上させるための戦略について説明します。本章で説明するトピックは、以下の通りです。

- 『Web サーバー API の使い方』
- 218ページの『FastCGI の使い方』
- 218ページの『接続管理』
- 223ページの『Net.Data のキャッシング』
- 251ページの『エラー・ログ・レベルの設定』
- 251ページの『言語環境の最適化』

さらに、Web サーバーが正しく調整されていることを確認してください。Web サーバーのパフォーマンスは、Net.Data がマクロや直接要求を処理する速度には関係なく、応答時間に直接影響を与えます。

Web サーバー API の使い方

CGI の代わりに、GWAPI などの Web サーバー API を使用して Net.Data を起動すると、パフォーマンスを向上させることができます。Net.Data が Web サーバー API を使用して実行されている場合は、Net.Data は Web サーバーのプロセス内でスレッドとして実行されます。Web サーバーの処理はマルチスレッド化されているので、複数の Net.Data 要求を同一のアドレス空間内で同時に処理することができます。そのため、Net.Data を CGI プロセスとして起動する場合のオーバーヘッドがなくなります。

考慮事項: Web サーバー API を使用すると、アプリケーションを分離せずにパフォーマンスを向上させることができます。Net.Data はマルチスレッド環境で実行されるため、ユーザー作成の言語環境、不適切な呼び出し、またはデータベースの故障などによるエラーは、Web サーバーでの問題発生の原因となり、サーバーをダウンさせることもあります。Web サーバー API の 1 つを使用するかどうかを決定する場合は、アプリケーションがパフォーマンスを重視するか、アプリケーションの分離を重視するかによって決定します。

FastCGI の使い方

FastCGI を使用してアプリケーションを CGI から独立させることにより、パフォーマンスが向上します。Net.Data は、FastCGI をサポートするすべての Web サーバー上で FastCGI と共に使用できます。FastCGI の構成方法を理解するには、45ページの『FastCGI のための Net.Data の構成』を参照してください。

着信要求の数を処理するためのプロセスを適切な量だけ実行するよう、プロセス構成パラメーターで FastCGI を調整することができます。たとえば、1 秒あたり 100 件の要求を処理する必要がある、各要求の処理に 0.5 秒かかるとします。この場合、FastCGI の構成ファイルで NumProcesses ディレクティブを 50 に設定することができます。

FastCGI はすべての LE でサポートされていますが、Oracle および ODBC の場合は、Live Connection が必要です。

同時プロセスの数の調整方法:

1. プロセスの構成パラメーターが定義されている構成ファイルをオープンする。
 - Apache の場合、このファイルは httpd.conf です。
 - Lotus Domino Go の場合、このファイルは lgw_fcgi.conf です。
 2. プロセスの数を指定する構成パラメーター値を次のようにして変更する。
 - Apache の場合: Process=num
 - ISC の場合: NumProcess=num
- ここで、num はプロセス数です。

接続管理

Net.Data は、データベースと Java 仮想マシン接続を管理するための Live Connection と呼ばれるコンポーネントを提供しています。Live Connection はパフォーマンスを向上させるための永続的接続を保持します。Net.Data アクションには、開始に時間がかかるものがあります。たとえばデータベース照会、発行前に、このプロセスの DBMS に対する識別と、データベースへの接続を行わなければなりません。これは多くの場合、Net.Data マクロがデータベースをアクセスするために必要な処理時間の、大半を占めてしまいます。開始時間がかかるその他の例としては、Java 仮想マシンです。このマシンは Java アプリケーション (Java アプレットではありません) を実行するものです。CGI プログラムはその動作上、Web サーバーへの各要求ごとに相当の開始時

間がかかります。Net.Data は、OS/2、Windows NT、および UNIX オペレーティング・システム上の Live Connection 提供し、永続的接続を保持します。

Live Connection について

Live Connection は、開始時のオーバーヘッドをなくすことにより、パフォーマンスを劇的に向上させることができます。これは、開始関数を実行する 1 つ以上のプロセスを連続して実行するからです。したがってこれらのプロセスは要求が処理されるまで待機します。Net.Data を CGI プログラムまたは FastCGI プログラムとして、あるいは Web サーバー API のプラグインとして使用する場合は Live Connection を使用することができます。

Live Connection は接続管理プログラムとクライアントで構成されています。クライアント は、接続管理プログラムが開始するプログラムで、サーバーが実行中は活動状態に保たれます。クライアントはデータを処理し、初期設定ファイルにおいてキーワード CLINETTE で指定した Net.Data 言語環境と通信します。クライアントの各タイプごとに、特定の言語環境関数を処理します。たとえば DB2 クライアントは、DB2 データベースに接続し、Net.Data のマクロが Net.Data によって処理される前に、SQL 呼び出しを実行するための操作をセットアップします。実行可能ファイルは、Live Connection の構成ファイル dtwcm.cnf で指定されています。220ページの図25 は Live Connection、マクロ、および言語環境の間での対話を示しています。

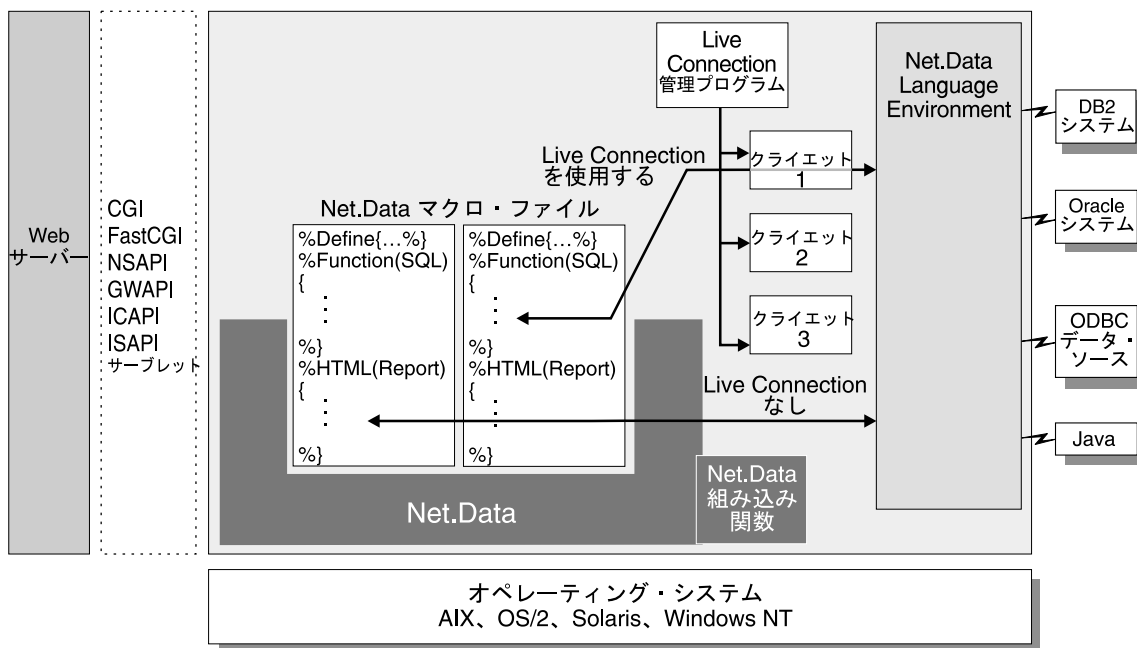


図 25. クライエントを持つ Live Connection

以下のセクションでは、Live Connection を詳細に説明します。Live Connection の構成方法を理解するには、38ページの『Live Connection の構成』を参照してください。

Live Connection の利点

Live Connection を使用した場合の主な利点は次のとおりです。

- パフォーマンスの向上

接続を再使用すると、新規に接続を作成するよりも効率的になります。一般に、小さな SQL ステートメントを要求したり (たとえば、100000 より少ない行のデータベースに対する簡単な照会)、あるいはデータベース接続が難しい場合 (たとえばリモート・サーバー)、接続時間はかなり長くなります。

- 複数のデータベース・アクセス

Live Connection では、1 つの Net.Data マクロを同時に複数のデータベースに接続できます。これが可能なのは、各データベースは固有のクライアントを持っており、したがって Net.Data は単に複数のクライアントと通信するだけだからです。

Live Connection を使用するかどうかの判断

Live Connection を CGI、FastCGI、あるいは API モードで使用すると、データベースと通信することができます。さらに、アプリケーションが複数のデータベースのデータを要求する場合には、Live Connection の恩恵を受けることができます。

Live Connection を API プラグインと共に使用すると、多くのシステムでパフォーマンスが向上します。ただし、これはシステムの負荷と構成によって異なります。使用しているシステムで試行し、最適の構成を見つけだしてください。

多くのアプリケーションでは、Live Connection を使用しなくても、ACTIVATE DATABASE コマンドを使用してデータベース接続の確立時間を節約することにより、パフォーマンスを向上させることができます。データベースが使用するコマンドの詳細は、データベースの資料を参照してください。また、オペレーティング・システムの資料も参照して、パフォーマンスの向上に役に立つその他のステップがないか調べてみてください。

要件: FastCGI モードおよび API モードで実行する場合には、ODBC および Oracle 言語環境は Live Connection を必要とします。

接続管理プログラムの開始

接続管理プログラムは、Net.Data に付属している独立した実行可能ファイルで、dtwcm という名前が付けられています。Web サーバーの開始時に接続管理プログラムを開始してください。

接続管理プログラムを開始すると、このプログラムは構成ファイルを読み取り、プロセスのグループを開始します。各プロセスでは、接続管理プログラムは特定のクライアントの実行を開始します。Live Connection の構成方法を理解するには、38ページの『Live Connection の構成』を参照してください。

Windows NT および OS/2 での接続プログラムの開始方法:

1. コマンド行から `<inst_dir>%connect%` ディレクトリーにディレクトリーを変更する。
2. dtwcm を入力する。

ここで、`<inst_dir>` は、Net.Data のインストール・ディレクトリーです。

AIX での接続管理プログラムの開始

1. コマンド行から、`/usr/lpp/internet/db2www/db2/` ディレクトリーにディレクトリーを変更する。

2. dtwcm を入力する。

メッセージ・オプションを指定して接続管理プログラムを開始する方法:

デフォルトでは、接続管理プログラムのメッセージは、表示されないようになっています。接続管理プログラムのメッセージを表示したい場合は、-d オプションを指定して接続管理プログラムを開始します。

コマンド行から、dtwcm -d を入力します。

-d オプションを指定した場合、再度メッセージを表示させないようにするには、接続管理プログラムを再始動しなければなりません。

接続管理プログラムを、Windows NT のサービスとして、自動的に開始するには、以下のようにします。

Windows NT 上では、接続管理プログラムを、コマンド行からではなく、Windows NT のサービスとして開始させるよう指定することができます。接続管理プログラムを Windows NT のサービスとして実行すると、マシンを開始するたびに、接続管理プログラムを自動的に開始させることができます。

ヒント：自動的に接続管理プログラムを開始するようにセットアップする前に、接続管理プログラムをコマンド行から開始し、Live Connection の構成ファイルが正しいことを確認します。

1. Windows NT のタスクバーから、「**スタート (Start) -> 設定 (Settings) -> コントロール・パネル (Control Panel) -> サービス (Services)**」と選択する。
2. 「**Net.Data の接続管理プログラム (Net.Data Connection Manager)**」を選択し、次に「**開始 (Startup)**」ボタンをクリックする。
3. 「**自動開始のタイプ (Automatic startup type)**」を選択し、「**OK**」をクリックする。

Net.Data および Live Connection のプロセスの流れ

構成が完了し、データベース、Web サーバー、および接続管理プログラムを開始してからの Net.Data のプロセスは、Live Connection が使用可能な場合には、一般に次のようなステップを取ります。

1. Web サーバーが要求を受け取り、FastCGI、CGI または API プロセスのいずれかを開始して Net.Data を実行する。
2. Net.Data が Net.Data マクロの処理を開始する。

3. Net.Data が Live Connection を使用する関数呼び出しを検出すると Net.Data は、初期設定ファイルから必要なクライアントのタイプを決定する。DB2 の場合、クライアントのタイプは、多くの場合、DB2 データベースの名前に基づいた名前になります (たとえば、DTW_SQL:CELDIAL)。
4. Net.Data が接続管理プログラムにそのタイプのクライアントを要求する。
5. 接続管理プログラムは、そのタイプの使用可能なクライアントを探す。使用可能なクライアントがない場合、接続管理プログラムはその要求を待ち行列に挿入し、正しいクライアントのタイプが使用可能になるとそれを処理します。
6. クライアントが使用可能になると、接続管理プログラムは Net.Data に対して、クライアントとの通信方法を指示する。
7. Net.Data はクライアントに対して関数の処理を依頼する。
8. Net.Data マクロの処理が完了するまで、このプロセスをステップ 3 から繰り返す。
9. すべてのクライアントを解放する。

初期設定ファイルでクライアントが指定されており、接続管理プログラムが実行されていない場合は、Net.Data は DLL をロードしてマクロを処理します。API を使用すると、おそらくエラーが発生します。その場合は、接続管理プログラムを開始しなければなりません。

Net.Data のキャッシング

キャッシングは、アプリケーション・ユーザーにとっては、応答時間の改善に役に立ちます。Net.Data は、高速な検索のために、Web サーバーに要求した結果を、情報のリフレッシュのときまで、ローカルに保管しておきます。本章では、Net.Data のキャッシングの概念、タスク、および制約事項について説明します。

- 224ページの『Web ページのキャッシングについて』
- 225ページの『Net.Data のキャッシングについて』
- 225ページの『Net.Data のキャッシングの用語』
- 226ページの『Net.Data のキャッシュの概念』
- 228ページの『Net.Data のキャッシュの制約事項』
- 228ページの『Net.Data のキャッシング・インターフェース』
- 229ページの『キャッシュ管理プログラムのための計画』
- 230ページの『キャッシュ識別子』

- 231ページの『キャッシュ管理プログラムおよび Net.Data のキャッシュの構成』
- 240ページの『キャッシュ管理プログラムの開始と停止』
- 241ページの『Web ページのキャッシング』
- 245ページの『CACHEADM コマンド』
- 248ページの『キャッシュ・ログ』

Web ページのキャッシングについて

多くのソフトウェア・コンポーネントは、Web アプリケーションに対してキャッシングを実行します。以下にキャッシング・アプリケーションの例を幾つか示します。

- Web ブラウザーは、Web ページ、およびイメージとオーディオ・ファイル、それに Java のアプレットなどの関連するオブジェクトを、ローカルにメモリーやディスクに保管し、Web ユーザーが繰り返し同じページにアクセスするときのネットワーク時間を節約する。
- Web プロキシ・サーバー・キャッシュは、Web ページおよび関連付けられているオブジェクトを、ユーザー・グループの近くのローカル・サーバーに保管し、リモートの Web サーバーへのアクセス時間を削減します。たとえば、Web サーバーが要求した項目を検索する回数を削減します。Web プロキシ・サーバー・キャッシュはまた、複数のユーザー間で共通にアクセスされる共有ページを効率的に共有できるようにします。
- Web サーバーは、検索される回数が多いページおよび関連付けられているオブジェクトをメモリーにキャッシュし、ユーザーが繰り返し同じページを検索するときのディスク・アクセス時間を節約する。
- データベース管理システムは、通常ディスクに保持されているデータ項目をメモリーにキャッシュし、繰り返し同じデータ項目を検索するときのディスク・アクセス時間を節約する。

これらのコンポーネントはすべて、独立してキャッシングを実行しますが、全体的な結果としては、ユーザーにとっては応答時間が改善されています。キャッシュされた項目をリフレッシュする時期を決定するために、Web コンポーネント (ブラウザー、プロキシ・サーバー、および Web サーバー) は、通常、以下のようなさまざまなオプションを考慮に入れます。

- ブラウザーおよびサーバーの構成オプション
- Web サーバーから Web ページおよび関連付けられている項目と一緒に戻される HTTP のヘッダーの内容、特に有効期限情報

Net.Data のキャッシングについて

Net.Data それ自身は、それ自身のキャッシング関数を、Net.Data のマクロにより生成された何度もアクセスされるページおよび関連付けられたデータ項目に提供します。 Net.Data のキャッシュからページを送達することにより、Net.Data のマクロの実行、およびページ作成のためのデータへのアクセスに必要な時間を節約します。

サーバーごとに 1 つのキャッシュ管理プログラムが使用できます。**推奨:** 複数の Net.Data インスタンスに対して 1 つのキャッシュ管理プログラムを使用し、キャッシュ管理プログラムごとに複数のキャッシュを使用します。

図26 は、Net.Data でキャッシュ管理プログラムを使用して、マクロからの HTML 出力のキャッシュの管理を示しています。この出力は、データベースからのデータを組み込むことができます。

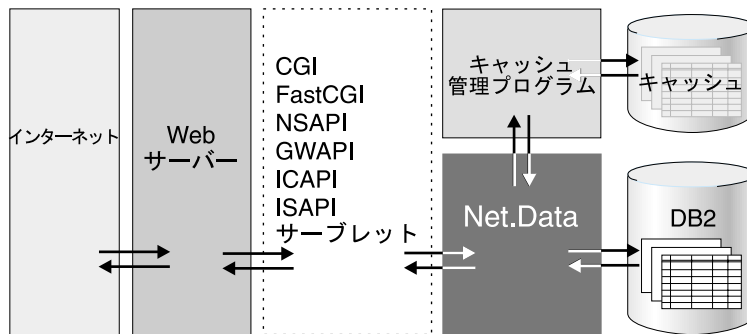


図 26. Net.Data のキャッシング

Net.Data のキャッシングの用語

Net.Data の文書は、以下の用語を使用して、Net.Data のキャッシングを説明します。

キャッシュ

最近アクセスされたデータが入るメモリの型。同一データに続けてアクセスする場合の速度を上げることが目的。キャッシュは、ネットワークを介してアクセス可能な、使用頻度の高いデータのローカル・コピーを保留するのに使用される場合が多い。 Net.Data では、Net.Data のマクロが再使用するために、Net.Data により生成される HTML の Web ページを含むローカル・メモリー。ページをキャッシュに格納することにより、Net.Data はキャッシュ内の情報を再生成する必要がありません。各キャッシュは、キャッシュ管理プログラムにより管理されます。

キャッシュ管理プログラムは、複数のキャッシュの管理をまかせることができ、Net.Data のサーバーの複数インスタンスを提供することができます。

キャッシュ ID

特定のキャッシュを識別するストリング。

キャッシュ管理プログラム

1 台のマシンのキャッシュを管理するプログラム。複数のキャッシュを管理できる。

構成、キャッシュ管理プログラムファイル

ログ記録、トレース、キャッシュ・サイズ、および他のオプションの設定を決定するために、Net.Data によって使用される設定を含むファイル。キャッシュ管理プログラムの設定および特定のキャッシュ管理プログラムにより管理されるすべてのキャッシュ・ファイルを含みます。ファイル名は、Net.Data と一緒にパックされている場合、cachemgr.cnf になります。

Net.Data のキャッシュの概念

システム上の HTTP の数、および HTTP サーバーが、Net.Data のそれ自身のコピーを (別々の Net.Data の構成ファイルを使用して) 実行させるかどうかにもよりますが、Net.Data のすべてのコピーを、1 つのキャッシュ管理プログラムあるいは複数のキャッシュ管理プログラムと関連付けることができます。

1 つのキャッシュ管理プログラムは、メモリー内の多くのキャッシュをサポートことができ、各キャッシュは、キャッシュ *ID* と呼ばれるキャッシュ識別子を持っています。227ページの図27 は、1 つのキャッシュ管理プログラムが複数のマクロを使用して、2 つのキャッシュの管理を示しています。

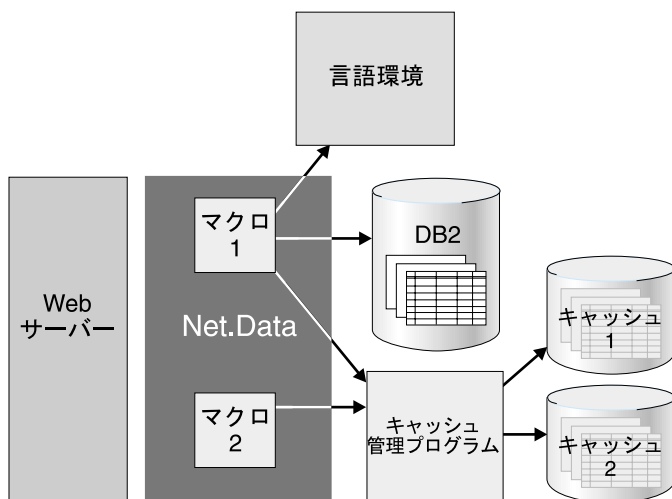


図 27. 複数のマクロおよびキャッシュを扱うキャッシュ管理プログラム

任意の数の項目は、キャッシュ・ページとして知られており、キャッシュ内に配置することができます。キャッシュ・ページはそれぞれ、一意の識別子を持っており、これにはたとえば URL (Uniform Resource Locator) があります。ページとは、完全な HTML のページのセグメント、あるいは完全な HTML のページです。

Net.Data がキャッシュ・データへの要求を受け取ると (たとえば、組み込み関数 DTW_CACHE_PAGE から)、以下のステップが行われます。

1. Net.Data が、キャッシュ管理プログラムに接続する。
2. Net.Data が、そのデータはキャッシュされているかどうかをチェックする。
 - データがキャッシュされており有効期限が切れていなければ、Net.Data はキャッシュ管理プログラムにページを要求、ブラウザーに送信し、マクロの実行を停止します。
 - データがキャッシュされていなければ、Net.Data はマクロの処理を継続し、生成された HTML ページを Web ブラウザーおよびキャッシュ管理プログラムに送信します。ここでページがキャッシュされます。
3. Net.Data がキャッシュ管理プログラムから切断される。

キャッシュ管理プログラムにより、マクロ・ファイルの処理が問題なく終了すると、HTML 出力がキャッシュされ、これにより正常に生成された Web ページのみがキャッシュされます。データはブラウザーに送信されてからキャッシュされ、ユーザーに表示されるデータはキャッシュされるデータと同じものになります。

Net.Data にエラーが発生、またはマクロが途中で終了した場合、キャッシュ管理プログラムは以下を行います。

- 部分的または不足するページは受け入れません。
- 既存のページをキャッシュに保存します。

Net.Data のキャッシュの制約事項

Net.Data のキャッシングには、以下の制約事項があります。

機密保護

機密保護は、キャッシュ管理プログラムからは提供されません。たとえば、あるデータベース・ユーザーがマクロを実行し、何ページものデータベースの結果をキャッシュした場合です。別のデータベース・ユーザーは、そのキャッシュ・ページを取得することができます。

直接要求

Net.Data の直接要求起動は、Net.Data のキャッシングを使用できません。

Net.Data のキャッシング・インターフェース

Net.Data は、ユーザー・アプリケーションのためのキャッシングの構成とセットアップを行うときに使用する柔軟なインターフェースのセットを提供してくれます。表13 は、Net.Data のキャッシング機能を使用するためのさまざまなオプション、およびこれらの機能の説明箇所について説明しています。

表 13. Net.Data のキャッシュ・インターフェース

インターフェース	説明	参照 ...
キャッシュ管理プログラムの構成オプション	ログ記録とトレースなど、キャッシュ管理プログラムのための多くのオプションは、キャッシュ管理プログラムの構成ファイルの、キャッシュ管理プログラムのスタンザで指定することができます。	231ページの『キャッシュ管理プログラムの定義』

表 13. *Net.Data* のキャッシュ・インターフェース (続き)

インターフェース	説明	参照 ...
キャッシュ構成のオプション	<i>Net.Data</i> キャッシュ管理プログラムの単一インスタンスにおいては、多くのキャッシュを定義して、キャッシュ項目を保持することができます。各キャッシュは、サイズと位置、そしてキャッシュ ID といったキャッシュ自身の特性の集合を持っています。これらの特性は、キャッシュ管理プログラムの構成ファイルのキャッシュ・スタンザで定義されます。各スタンザは、キャッシュ ID で識別されます。	234ページの『キャッシュの定義』
<i>Net.Data</i> の初期設定オプション	<i>Net.Data</i> およびそれに対応するキャッシュ管理プログラムを別々のシステムで実行する場合は、キャッシュ管理プログラムのシステムとポート番号を、 <i>Net.Data</i> の初期設定ファイルで指定します。	15ページの『キャッシュ管理プログラムの構成変数』
<i>Net.Data</i> のキャッシュの組み込み関数	<i>Net.Data</i> のキャッシュの組み込み関数を使用して、 <i>Net.Data</i> のキャッシュの内容を操作することができます。適切なマクロ関数にキャッシュ ID を指定し、最適な特性を持つキャッシュを選択します。	<i>Net.Data</i> 解説書 の組み込み関数の章を参照してください。

キャッシュ管理プログラムのための計画

Net.Data のキャッシュ関数を使用するための計画を立てる場合は、以下の点を考慮しなければなりません。

- キャッシングの恩恵を受けるページ数および得られるパフォーマンスの改善度
- 項目をキャッシュする時期

- キャッシュ内の項目をリフレッシュする時期、および使用するリフレッシュ方法

Net.Data のキャッシングを使用するには、以下のステップを完了しておく必要があります。そのためには、どのようにキャッシングを使用したいのかを知っておかなければなりません。

推奨：キャッシングを使用する大きなアプリケーションに取りかかる前に、アプリケーションの実働に先立って、アプリケーションの計画を立て、プロトタイプを作成することを強くお勧めします。

- Net.Data をインストールする。これには、キャッシング関数が組み込まれます。
- キャッシュ管理プログラムを構成する。231ページの『キャッシュ管理プログラムおよび Net.Data のキャッシュの構成』を参照してください。
- Net.Data のアプリケーションをどのようにして実働に移すのかを決定する。
- さまざまな Net.Data のキャッシュ・ログをチェックして、キャッシュの使用上およびその構成方法の改善を行うべきかどうかを決定する。

キャッシュ・エラー

Net.Data に内部エラーが発生し、マクロ・ファイルの処理が完了する前に終了された場合、キャッシュ管理プログラムでは Web ページがキャッシュされません。キャッシュ管理プログラムは、不完全または Net.Data エラーを含むページをキャッシュしません。このようなエラーには、マクロの構文エラーおよび SQL エラーがあります。

エラーを含むページは、以下の場合にキャッシュされます。

- Net.Data にエラーが発生し、メッセージ・ブロックの CONTINUE 構成変数により Net.Data のマクロの実行が継続され正常に終了した場合。
- データベースのロールバックなど、Net.Data のエラー判別効力範囲の外部でエラーが発生した場合。

キャッシュ識別子

アプリケーションのためのキャッシングの設計時に、2 つのタイプの識別子のプランを立てる必要があります。

- **キャッシュのための識別子：**この識別子は、キャッシュ ID であり、キャッシュを定義する構成ファイルのスタンザでその名前を指定します。キャッシュの分類および名前付けには、さまざまな方法を使用することができます。たとえば、キャッシュにはアプリケーションごとに名前を付けることができ

ます。Net.Data アプリケーションごとにキャッシュが作られ、キャッシュがサービスを提供する Net.Data マクロから派生した名前を、各キャッシュに与えます。

- **キャッシュ・ページの識別子**：この識別子は、キャッシュ・ページの ID で、キャッシュされるページの名前を指定します。キャッシュ・ページ ID は、URL アドレスなどの任意のストリングとします。ID は DTW_CACHE_PAGE() 組み込み関数で指定します。構文および例については、*Net.Data 解説書* の組み込み関数の章を参照してください。

キャッシュ管理プログラムおよび Net.Data のキャッシュの構成

キャッシュ管理プログラムは、ユーザー・システムにおける 1 つ以上のキャッシュを管理します。これらのキャッシュはそれぞれ、動的に生成された HTML ページの内容を含んでいます。キャッシュ管理プログラムおよび、各キャッシュを構成するには、キャッシュ管理プログラムの構成ファイル `cachemgr.cnf` におけるキーワードの値を更新します。

キャッシュ管理プログラムの構成ファイルには 2 種類のスタンザが含まれます。キャッシュ管理プログラム・スタンザとキャッシュ定義スタンザです。以下のステップでは、ユーザー・アプリケーションのためのこれら 2 種類のスタンザのカスタマイズ方法について説明します。

キャッシュ管理プログラムの定義

許可済みのキーワードの値を指定して、キャッシュ管理プログラムのスタンザを定義します。キーワードはすべてオプションです。デフォルト値を受け入れたくない場合以外は、キーワードを指定する必要があります。

キャッシュ管理プログラムを定義するには、以下を行います。

1. キャッシュ管理プログラムのログ・ファイルの名前を指定する。ログは、すべてのキャッシュに対して行うすべてのトランザクション活動を表示し、デバッグおよび問題解析のために提供されます。

デフォルトでは、メッセージをコンソールに書き込みます。

構文：

`log=path`

ここで、`path` は、キャッシュ・ファイルのパスおよびファイル名です。

ヒント：キャッシュごとにログ・ファイルを指定するには、キャッシュ定義スタンザの **tran-log** キーワードを使用します。

2. 着信要求に対してキャッシュ管理プログラムが使用する TCP/IP のポート番号を指定する。このポート番号は、リモート・マシンからキャッシュ管理プログラムに接続する場合のみ使用します。

この値は、Net.Data の初期設定ファイルの `DTW_CACHE_PORT` 構成変数で指定されるポート番号と一致していなければなりません。デフォルト値は、以下のようにして決定されます。

- a. キャッシュ管理プログラムは、`ibm-cachemgrd` という名前に関連付けられている値がないか、パス `/etc/services` をチェックする。この値が検出されれば、キャッシュ管理プログラムは、その値を使用します。値が見つからなければ、キャッシュ管理プログラムは、次の方法を使用します。
- b. キャッシュ管理プログラムは、デフォルトのポート、7175 を使用する。

構文：

```
port=port_number
```

ここで、`port_number` は、一意の TCP/IP ポート番号です。

3. キャッシュ管理プログラムが、保留読み取りをアクティブにしておかなければならない最大の時間の長さを、秒で指定する。この時間を過ぎると、キャッシュ管理プログラムは接続を除去します。

デフォルトは、30 秒です。

構文：

```
connection-timeout=seconds
```

ここで、`seconds` は、保留読み取りをアクティブにしておかなければならない時間の長さに使用する秒数です。

4. メッセージのログ記録を取るかどうかを指定する。

デフォルトは、**no** あるいは **off** です。

構文：

```
logging=yes|on|no|off
```

ここで、

yes|on

ログ記録が必要であることを示します。

no|off ログ記録を実行しないことを示します。

5. ログを循環するかどうかを指定する。

デフォルトは、 **no** です。**yes** と指定された場合は、ログのサイズ (下の **log-size** を参照) が最大になったときに、現行ログが閉じられ、そのファイルは、.old というファイル・タイプを持ち、新規のログがオープンします。ログ・ファイルは、1 世代分しか保守されません (既存の .old ファイルは上書きされます)。

構文 :

```
wrap-log=yes|no
```

ここで、

yes ログを循環するよう指定します。

no ログを循環しないよう指定します。

6. 最大サイズをバイトで指定します。wrap-log が指定されている場合は、ログは、その最大サイズまで大きくなることができます。

デフォルトは、64000 バイトです。

構文 :

```
log-size=bytes
```

ここで、bytes は、最大サイズのバイト数です。

7. ログに書き込むメッセージのレベルを指定する。以下の値は、*trace_flag_definitions* リストに組み込まれたときに設定されますが、設定項目というものはありません。

デフォルトでは、キャッシュ管理プログラムの開始およびシャットダウンのメッセージだけがログ記録されます。

構文 :

```
trace-flags=trace_flag_definitions
```

ここで、

D_ALL

すべてのトレース・フラグを使用可能にします。

D_NONE

すべてのトレース・フラグを使用不可にします。

例: すべてのトレース・フラグを使用可能に指定するトレース・フラグ。

```
trace=flags=D_ALL
```

スタンザの例 : 有効な構成マネージャーのスタンザは、以下のようになります。

```
cache-manager {
port = 7175
connection-timeout = 60
logging = off
log = /local/netdata/cachemgr/logs/cachemgr.log
wrap-log = yes
log-size = 32KB
}
```

キャッシュの定義

許可済みのキーワードの値を指定して、キャッシュ定義スタanzasを定義します。ほとんどのキーワードはオプションで、デフォルト値を使用したくない場合以外は、指定する必要はありません。

キャッシュを定義するには、以下を行います。

1. キャッシュ・ページを保持するパスおよびディレクトリー名。始動時には、このディレクトリーを含むファイル・システムは、少なくとも、**fssize** (以下を参照) の値と同じ大きさでなければなりません。そうでないと、キャッシュは開始されません。以下の値は、絶対パス名として、あるいはキャッシュ管理プログラムが開始したパスに対応する相対パス名として指定することができます。

必須項目です。

構文：

root=path_name

ここで、

path_name

キャッシュ・ページが保管されているパスおよびディレクトリーの絶対あるいは相対名です。

2. キャッシュ管理プログラムが始動したときに、現行キャッシュをアクティブにするかどうかを指定します。

必須項目ではありません。デフォルトは、**yes** です。**no** に設定されている場合は、キャッシュは、キャッシュ管理プログラムで定義されますが、アクティブにはなりません。**cacheadm** コマンドを使えば、後でアクティブにすることができます。

構文：

caching=yes|no

ここで、

yes キャッシュ管理プログラムが開始したとき、キャッシュをアクティブにするように指定します。

no キャッシュ管理プログラムが開始したとき、キャッシュをアクティブにしないように指定します。

3. 現行キャッシュ内のページが、ファイル・システムで使用する最大のスペースを指定します。スペースの最大量を超えた場合、キャッシュ管理プログラムは、キャッシュが占有するスペースの合計を制限内に持つていくのに十分なページを、最も古いものから順に削除していきます。この値を大きな値に設定すると、実効的に、エントリーの自動パージを使用不可能にすることができます。しかし、ファイル・システムの物理的なスペースを超えてしまうと、キャッシュに新規のスペースを追加することはできません。

必須項目ではありません。デフォルトは 0 です (ディスクにはキャッシュしません)。

構文：

`fssize=nnB|nnKB|nnM`

ここで、

nnB バイト数です。たとえば、5000B とします。

nnKB K バイト数です。たとえば、640KB とします。

nnMB M バイト数です。たとえば、30MB とします。

4. このキャッシュ内のすべてのページが使用するメモリの最大量を指定する。メモリの最大量を超えた場合、キャッシュ管理プログラムは、キャッシュが占有するメモリの合計を制限内に持つていくのに十分なページを、最も古いものから順に削除していきます。これを大きな値に設定すると、実効的に、ページの自動パージを使用不可能にすることができます。しかし **cachemgrd** プロセスがメモリを消費しすぎると、オペレーティング・システムは、プロセスを終了することがあります。

必須項目ではありません。デフォルトは、1MB です。

構文：

`mem-size=nnB|nnKB|nnMB`

ここで、

nnB バイト数です。たとえば、5000B とします。

nnKB K バイト数です。たとえば、640KB とします。

nnMB M バイト数です。たとえば、30MB とします。

5. キャッシュにページを保留しておくことができる最大の時間の長さ。この値を超えると、キャッシュ管理プログラムは、そのページを有効期限切れとしてマークします。しかし、**fssize** (ページがディスクにキャッシュされている場合) あるいは **memsize** (ページがメモリーにキャッシュされている場合) の制限に達しない限り、そのページは削除されません。キャッシュ管理プログラムは、**memsize** あるいは **fssize** の制限に達した場合は、他のすべてのページに先だって、有効期限切れとしてマークされたページを削除します。**lifetime** のチェックを、**check_expiration** キーワードを使って使用不可にすることができます。

必須項目ではありません。デフォルトは、5 分です。

構文：

```
lifetime=time_length
```

ここで、

nnS 秒数です。たとえば 600S とします。

nnM 分数です。たとえば 20M とします。

nnH 時間数です。たとえば 30H とします。

6. キャッシュ・ページを有効期限切れとしてマークし、存続時間の検査を実行するかどうかを指定する。

必須項目ではありません。デフォルトは、**yes** で、デフォルトの存続時間の長さは 60 秒です。次の値は、時間の長さに設定することもできます。**yes** という値を指示し、項目をキャッシュに保留しておくことができる最大の時間の長さを宣言します。**no** に設定すると、キャッシュ・ページは、有効期限切れとマークされることはありません。そして、存続時間検査は実行されません。

構文：

```
check-expiration=yes|nnS|nnM|nnH|no
```

ここで、

yes キャッシュ管理プログラムは存続時間検査を実行し、キャッシュ・ページを有効期限切れとしてマークするように指定します。

nnS 秒数です。たとえば 600S とします。

nnM 分数です。たとえば 20M とします。

nnH 時間数です。たとえば 30H とします。

no キャッシュ管理プログラムは存続時間検査を実行しないように、そしてキャッシュ・ページを有効期限切れとしてマークしないように指定します。

7. キャッシュ・ページが、メモリー・キャッシュ内に占有することができる最大のスペース量を指定する。ページがメモリーにとって大きすぎる場合は、ファイル・キャッシュがチェックされます。十分なスペースが存在する場合は、キャッシュ管理プログラムは、そのかわりに、そのキャッシュ・ページをファイル・キャッシュに格納します。ページがファイル・キャッシュに収まりきれない場合は、キャッシングの実行は失敗します。ページは、`datum_memory_limit` 値 (`cacheobj-memory-limit`) よりも小さいけれども、キャッシュには十分なスペースがない場合、最も古いキャッシュ・スペースがメモリー・キャッシュから削除され、新規のページが収容されます。

必須項目ではありません。デフォルトは、1KB です。

構文：

```
datum-memory-limit (cacheobj-memory-limit)=nnB|nnKB|nnMB
```

ここで、

nnB バイト数です。たとえば、5000B とします。

nnKB K バイト数です。たとえば、640KB とします。

nnMB M バイト数です。たとえば、30MB とします。

8. キャッシュ・ページが、ファイル・キャッシュ内に占有することができる最大のスペース量を指定します。ページは `datum_disk_limit` 値 よりも小さいけれども、キャッシュにはスペースが残っていない場合、最も古いキャッシュ・ページがファイル・キャッシュから削除され、新規のページが収容されます。

必須項目ではありません。デフォルトは、1KB です。

構文：

```
datum-disk-limit (cacheobj-space-limit)=nnB|nnKB|nnMB
```

ここで、

nnB バイト数です。たとえば、5000B とします。

nnKB K バイト数です。たとえば、640KB とします。

nnMB M バイト数です。たとえば、30MB とします。

9. 統計レコードの作成間隔の時間を指定する。0 に設定すると、統計レコードは書き込まれません。

必須項目ではありません。デフォルトは、0 です (統計はありません)。

構文 :

```
stat-interval = nnS|nnM|nnH
```

ここで、

nnS 秒数です。たとえば 600S とします。

nnM 分数です。たとえば 1M とします。

nnH 時間数です。たとえば 3H とします。

10. 現行キャッシュの統計のログを記録するために使用されるパスおよびファイルの名前を指定する。

stat-interval の値が、0 よりも大きい場合に必須です。

構文 :

```
stat-files=filename
```

ここで、*filename* は、ログを記録する統計ファイルのパス名です。

11. 統計がログ・ファイルに書き込まれるごとに、統計のカウンターを 0 にリセットするかどうかを指定する。

必須項目ではありません。デフォルトは、**yes** です。

構文 :

```
reset-stat-counters=yes|no
```

ここで、

yes 統計カウンターをリセットします。

no 統計カウンターをリセットしません。

12. キャッシュごとにトランザクションのログを保持するパスおよびファイル名を定義する。キャッシュのトランザクション・ログのファイルは、キャッシュ管理プログラムのログ・ファイルとは別のもので、後者は、キャッシュ管理プログラムの活動全体のログを記録するのに使用されます。

必須項目です。特に指定されていなければ、キャッシュのトランザクション・ログは作成されません。

構文 :

```
tran-log=filename
```

ここで、*filename* は、キャッシュごとのトランザクション・ログのパスおよびファイル名です。

13. キャッシュ管理プログラムが最初に開始したときに、キャッシュのトランザクション・ログをオンにするかどうかを指定する。このパラメーターは、**tran-log** パラメーターを介して有効なトランザクション・ログのファイルが指定されていなければ、無視されます。有効な **tran-log** 値が、キャッシュ管理プログラムの構成ファイルで指定されていれば、キャッシュ管理プログラムのデーモンが実行されている間は、**cacheadm** コマンドを使用して、トランザクションのログ記録をアクティブにしておくことができます。

必須項目ではありません。デフォルトは、**no** です。

構文：

```
tran-logging=yes|on|no|off
```

ここで、

yes|on

ログが必要であることを示します。

no|off ログ記録を実行しないことを示します。

14. トランザクション・ログを循環するかどうかを指定する。

必須項目ではありません。デフォルトは、**yes** です。**yes** として指定されている場合は、現行ログは、最大サイズ (**tran-log-size** を参照) に達したときに閉じられ、**.old** というファイル・タイプを持ちます。そして、新規のログがオープンします。ログは、1 世代分しか保守されません (既存の **.old** ファイルは上書きされます)。

構文：

```
wrap-tran-log=yes|no
```

ここで、

yes ログを循環することを示します。

no ログを循環しないことを示します。

15. **wrap-tran-log** が指定されている場合は、トランザクション・ログを大きくすることができる最大サイズをバイトで指定します。

必須項目ではありません。デフォルトは、64000 です。

構文：

```
tran-log-size=bytes
```

ここで、**bytes** は、最大サイズのバイト数です。

スタンザの例：キャッシュに対する有効なキャッシュ定義スタンザ

```

cache0
{
root = /locale/netdata/cachemgr/caches/cache0
caching = on
mem-size = 10MB
fs-size = 1MB
datum-memory-limit = 200KB
datum-disk-limit = 1MB
lifetime = 6000000
check-expiration = 999999
tran-logging = no
tran-log-size = 10000
wrap-tran-log = yes
tran-log = /ocale/netdata/cachemgr/logs/tran.log
}

```

キャッシュ管理プログラムの開始と停止

以下のセクションでは、キャッシュ管理プログラムの開始および停止方法について説明します。

- 『キャッシュ管理プログラムの開始』
- 241ページの『キャッシュ管理プログラムの停止』

キャッシュ管理プログラムの開始

cachemgrd コマンドを使用して、キャッシュ管理プログラムのデーモンを開始します。

構文：

➡cachemgrd—c—*config_file*—————➡◀

パラメーター：

cachemgrd

コマンドのキーワード。

config_file

キャッシュ管理プログラムおよびキャッシュ管理プログラムにより管理される各キャッシュが定義されるファイルの名前を指定します。

Net.Data の製品と一緒に出荷された構成ファイルは、cachemgr.cnf です。

例：

```
cachemgrd -c myconfig.cfg
```

キャッシュ管理プログラムの停止

キャッシュ管理プログラムを停止するには、`cacheadm` コマンドを使用します。

構文：

```
➡—cacheadm—┬──hostname—hostname—┬──port—port_num—┬──terminate—➡
```

パラメーター：

cacheadm

コマンドのキーワード。

hostname

キャッシュが実行されているマシンが、`cacheadm` コマンドが発行されるマシンと異なっている場合、そのマシンの名前を指定します。

port_num

キャッシュのポート番号が、デフォルト (7175) と異なっている場合は、キャッシュのポート番号を指定します。

terminate

キャッシュ管理プログラムの停止を指定します。

例：

```
cacheadm hostname host1 port 7178 terminate
```

Web ページのキャッシング

`DTW_CACHE_PAGE` 組み込み関数を使用して、Web ページをキャッシュすることができます。Net.Data は、マクロの `DTW_CACHE_PAGE` 関数を見つけると、キャッシュ管理プログラムに接続し、マクロの HTML 出力をメモリーにキャッシュし始めます。Net.Data がマクロの処理を正常に終了すると、HTML 出力がブラウザーに送られ、キャッシュ管理プログラムは 1 トランザクションの出力をキャッシュします (242ページの図28 を参照)。

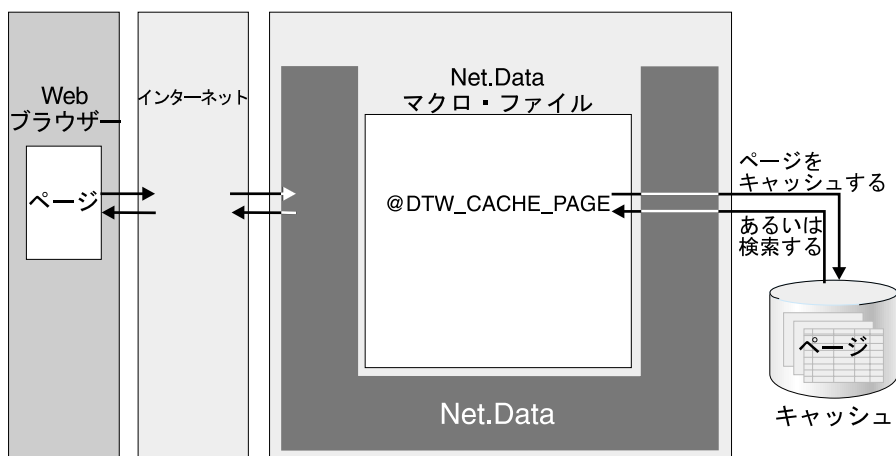


図 28. キャッシングを開始する DTW_CACHE_PAGE 関数

ページのキャッシング

Net.Data の DTW_CACHE_PAGE() 組み込み関数を使用して、Net.Data によって生成されたページを、キャッシュに書き込むように指定します。

DTW_CACHE_PAGE() 関数は、ページがすでにキャッシュに存在しない、または有効期限が切れていると判断すると、関数ステートメントに続くマクロ・ファイルからの出力をすべてキャッシュします。ページがキャッシュに存在していなかったり、指定された経過時間よりも古い場合は、Net.Data は出力ページをブラウザに送り戻し、マクロ実行からの新規の出力ページを生成し、そのページをキャッシュに格納します。

キャッシュ管理プログラムがキャッシュされたページを検出し、ページがまだ現行のものであれば、キャッシュ内容が表示され、Net.Data はマクロを終了します。このような振る舞いにより、Web ページがキャッシュから検索された後に必要のない処理が行われないようにしています。

パフォーマンスのためのヒント: DTW_CACHE_PAGE() をマクロの最初のステートメント、または最初のステートメントの 1 つとして配置し、マクロ・ファイルの実行時負荷を最小にします。

ページをキャッシュするには、以下を行います。

1. マクロ・ファイルの HTML ブロックでは、HTML のコーディングに先立ち、以下の関数ステートメントが挿入されます。

```
@DTW_CACHE_PAGE("cache_id", cached_page_id, "age", status)
```

この関数を使用して、Net.Data がこのステートメント以降のマクロからのすべての HTML 出力をキャッシュするように指定します。すべての HTML 出力をキャッシュする場合は、このステートメントをマクロの冒頭部分に置きます。

パラメーター :

cache_id

ページが配置されるキャッシュを識別するストリング。キャッシュ ID をマクロ、あるいはマクロのグループに関連付けることができます。

cached_page_id

後に @DTW_CACHE_PAGE により行うキャッシュ要求時に、キャッシュ内のページを識別するのに使用される識別子を含むストリング。たとえばページの URL。

age

ページが古くなったと考えられるときを指定する、秒で示される時間の長さを含むストリング変数。要求されたページが、*age* の値よりも長くキャッシュ内にある場合は、Net.Data は、マクロを実行し、ページを再生成し、そして、生成されたページをキャッシュして、古くなったページを置き換えます。要求されたページが、*age* の値よりも短いと同じ時間キャッシュ内にある場合は、Net.Data は、キャッシュからページを取得し、そのページをブラウザーに送信します。この場合、Net.Data は即時にマクロの実行を終了します。

status ページのキャッシュに成功したかそうでないかを示すために、Net.Data によって戻されるストリング変数。

例 :

```
%HTML(cache_example) {
  %IF (customer == "Joe Smith")
    @DTW_CACHE_PAGE("mymacro.d2w", "http://www.mypage.org", "-1", status)
  %ENDIF
  ...
<html>

<head>
  <:title>This is the page title</title>
</head>

<body>
  <center>
    <h3>This is the Main Heading</h3>
    <p>It is $(time). Have a nice day!
```

```
</body>

</html>
%}
```

高度なキャッシング: キャッシュの動的な判別

DTW_CACHE_PAGE() 関数は、マクロ内のその位置からキャッシングを開始します。通常は、関数をマクロの先頭に配置することによりパフォーマンスが向上し、すべての HTML 出力をキャッシュすることができます。

高度なキャッシング・アプリケーションでは、マクロの先頭ではなく処理途中の特定のポイントでキャッシュを行う必要がある場合、DTW_CACHE_PAGE() 関数を HTML 出力セクションに配置することができます。たとえば、照会または関数呼び出しから戻される行数に基づきキャッシングを行うか判断する場合があります。

例: 予想される HTML 出力のサイズに応じてキャッシュの決定を行うため、関数は HTML ブロックに配置します。

```
% DEFINE { ...%}

...

%FUNCTION(DTW_SQL) count_rows(){
    select count(*) from customer
%REPORT{
    %ROW{
        @DTW_ASSIGN(ALL_ROWS, V1)
    %}
    %}
    %}

%FUNCTION(DTW_SQL) all_customers(){
    select * from customer
    %}

%HTML(OUTPUT) {
<html>
<head>
<title>This is the customer list
</head>
<body>

@count_rows()

    %IF ($(ALL_ROWS) > "100")
        @DTW_CACHE_PAGE("mymacro.d2w", "http://www.mypage.org", "-1", status)
    %ENDIF

    @all_customers()
```



```

</body>
</html>
%}

```

この例では、ページは予想される HTML 出力サイズに基づきキャッシュまたは検索されます。データベース・テーブルの行が 100 以上の場合のみ、HTML 出力ページはキャッシュの対象となります。Net.Data は常に、マクロの実行後に OUTPUT ブロックのテキスト `This is the customer list` をブラウザに送信します。テキストはキャッシュされません。関数呼び出しに続く行の `@count_rows()` は、IF ブロックの条件が満たされるとキャッシュまたは検索されます。2 つの部分により完全な Net.Data 出力ページが形成されます。

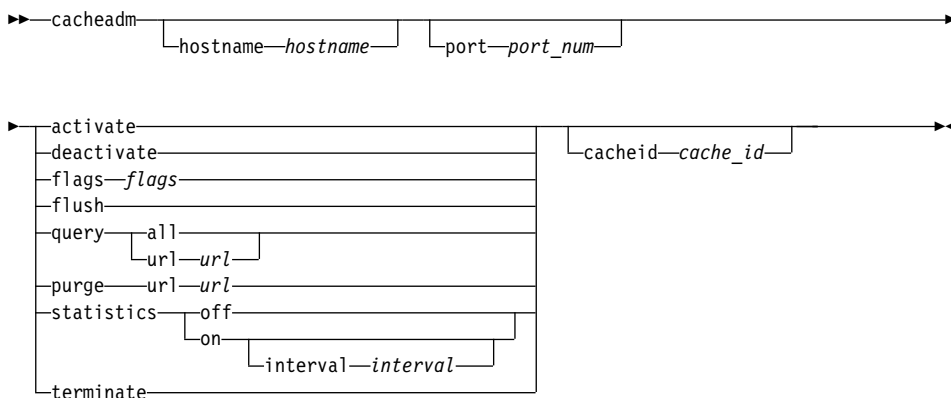
CACHEADM コマンド

以下のタスクに CACHEADM コマンドを使用します。

- キャッシュ管理プログラムの停止
- 特定のキャッシュのフラッシュ
- 照会、特定のキャッシュの
- ログ記録の使用可あるいは使用不可
- フラグのログ記録
- 統計収集の開始および停止

すべてのパラメーターは、省略して、最小の固有の文字集合にすることができます。

構文：



パラメーター：

activate

指定キャッシュをアクティブにします。キャッシュがすでにアクティブの場合は、キャッシュ管理プログラムは何もしません。

cache_id

ページが配置されるキャッシュを識別するストリング変数。たとえば、cache1 となります。

deactivate

指定キャッシュを非アクティブにします。キャッシュがすでに非アクティブの場合は、キャッシュ管理プログラムは何もしません。すべての保留操作は完了し、新規の操作は受け付けられません。最後の操作が完了すると、キャッシュ管理プログラムは、キャッシュを非アクティブとしてマークします。

flags

リストされたフラグを、オンあるいはオフに切り替えるかどうかを指定します。

D_ALL

すべてのトレース・フラグを、オンにします。

D_NONE

すべてのトレース・フラグを、オフにします。

flush

cache_id パラメーターで指定されるキャッシュをフラッシュします。この *flush* パラメーターには、このパラメーターが必要です。このパラメーターにより、指定のキャッシュからすべての項目が無条件に削除されます。

hostname

キャッシュが実行されているマシンが、*cacheadm* コマンドが発行されるマシンと異なっている場合、そのマシンの名前を指定します。たとえば、*myhost* とします。

port_num

キャッシュのポート番号が、デフォルト (7175) と異なっている場合は、キャッシュのポート番号を指定します。この番号は、システム内では一意的でなければなりません。

purge

キャッシュからパージする特定のページを指定します。 *url* が指定されている場合は、キャッシュ管理プログラムは、*url* と一致するキーを持つページをパージします。従属関係が指定されている場合は、キャッシュ管理プ

ログラムは、関連付けられている従属関係を持つすべての項目をページし、各項目のキーを、標準出力ストリームである `stdout` に書き込みます。

query

ユーザーが指定する以下のパラメーターに応じて、キャッシング・データを戻します。

- キャッシュ ID のみが指定されている場合は、キャッシュに関する情報を戻す。
- `url` が指定されている場合は、特定のキャッシュ・ページに関する情報を戻す。
- `all` が指定されている場合は、すべてのページに関する情報を戻す。

他のプログラムは、`all` オプションを使用して、結果をフォーマットあるいは解釈します。各行には、以下の情報が含まれます。

- ページ・キー
- ページの経過時間
- ページの長さ
- ページの作成日
- ページの有効期限
- ページが最後に参照された日付

すべての日付は、UNIX の標準整数時刻形式です。

パフォーマンスのためのヒント： オプション `cache query all` は、パフォーマンスに大きな影響を与えるので、あまり使わないようにします。

statistics

特定のキャッシュの統計収集のログ記録を使用可能にするか使用不可にします。これは、`cache_id` パラメーターを必要とします。インターバルが、`statistics` パラメーターを `on` にして指定されている場合は、`Net.Data` は、更新インターバルを、指定された秒数に設定あるいはリセットします。

terminate

キャッシュ管理プログラムの停止を指定します。

tranlogging

特定のキャッシュのトランザクション・ログの記録を使用化するか使用不可にします。これは、`cache_id` パラメーターを必要とします。このパラメーターが有効になるのは、キャッシュの有効なトランザクション・ログ

が、`tran-log` パラメーターと一緒に、キャッシュ管理プログラムの構成ファイルで指定されている場合だけです。

url Web サーバー上のファイルの位置を指定する URL (Universal Relative Location) アドレス。たとえば、`http:www.ibm.com/mydir/page1` となります。

キャッシュ・ログ

内部操作に関する各種統計が保持され、オプションでキャッシュ・ログに書き込むことができます。それぞれのキャッシュごとにログを別個に保持するか、またはすべての統計を同一のログに書き込むことができます。このセクションでは、以下のキャッシュ・ログに関するトピックを説明します。

- 『ログの構成』
- 『キャッシュ・ログ・フォーマット』

ログの構成

統計のログを作成するには、キャッシュ管理プログラムの構成ファイルを構成する必要があります。

ログの構成方法

キャッシュ管理プログラム構成ファイルのキャッシュ・スタンザに、`stat-files` および `stat-interval` キーワードを指定します。

統計設定の変更は、キャッシュ管理プログラムを停止、再構成、または再始動しなくても行うことができます。

統計収集設定の変更方法

`cacheadm statistics` コマンドを指定します。ただし、`cacheadm statistics` コマンドによる変更は、キャッシュ管理プログラムの再始動時に保存されません。

キャッシュ・ログ・フォーマット

統計ログは ASCII プレーン・ファイルで、表計算またはデータベース・プログラムにより処理またはインポートができます。以下の 3 種類のレコードが書き込まれます。

- 初期化レコードには、特定のキャッシュに関する統計収集の開始が記録されます。これらレコードは、以下のフォーマットをとります。

`mm/dd/yy hh:mm:ss id Initialization: interval n seconds`

ここで、

mm/dd/yy 統計収集を開始した月、日、および年。
hh:mm:ss 統計収集を開始した時間、分、および秒。
id レコードに関連したキャッシュの名前。
n 収集間隔

- 終了レコードには、特定のキャッシュに関する統計収集の終了が記録されます。これらレコードは、以下のフォーマットをとります。

mm/dd/yy hh:mm:ss id Termination

ここで、

mm/dd/yy 統計収集が終了した月、日、および年。
hh:mm:ss 統計収集が終了した時間、分、および秒。
id レコードに関連したキャッシュの名前。

- 統計レコードは、キャッシュ内のアクティビティーを示す、ブランクで区切られた数字です。これらレコードは、以下のフォーマットをとります。

mm/dd/yy hh:mm:ss id statistics

ここで、

mm/dd/yy 統計収集が作成された月、日、および年。
hh:mm:ss 統計収集が作成された時間、分、および秒。
id レコードに関連したキャッシュの名前。
<statistics> このキャッシュについて収集した統計に関するブランクで区切られたリスト。指定は 表14 を参照。

表 14. 統計リスト

フィールド番号	内容	説明	カウンターをゼロに再度初期化
1	読み取り	キャッシュに対して実行された読み取り操作の回数	はい
2	書き込み	キャッシュに対して実行された書き込み操作の回数	はい
3	クローズ	キャッシュ内のオブジェクトに対して実行されたクローズ操作の回数	はい

表 14. 統計リスト (続き)

フィールド番号	内容	説明	カウンターを ゼロに再度初 期化
4	オープン読み 取り	キャッシュ内のオブジェクトに対して実行 されたオープン読み取り操作の回数	はい
5	オープン書き 込み	キャッシュ内のオブジェクトに対して実行 されたオープン書き込み操作の回数	はい
6	オープン書き 込み照会	キャッシュ内のオブジェクトに対して実行 されたオープン書き込み照会操作の回数	はい
7	読み取りヒッ ト	キャッシュ内のオブジェクトに対する読み 取りヒットの回数	はい
8	書き込みヒッ ト	キャッシュ内のオブジェクトに対する書き 込みヒットの回数	はい
9	書き込み照会 ヒット	キャッシュ内のオブジェクトに対する書き 込み照会ヒットの回数	はい
10	初期化	このキャッシュで確立された新規セッシ ョンの数	はい
11	終了	このキャッシュで終了したセッションの数	はい
12	ページ	このキャッシュから削除されたオブジェク トの数	いいえ
13	使用メモリー	キャッシュのメモリー部分のオブジェクト が使用するメモリー量	いいえ
14	使用ディスク	キャッシュのディスク部分のオブジェクト が使用するディスク容量	いいえ
15	使用可能メモ リー	キャッシュのメモリー部分のオブジェクト が使用可能なメモリー量	いいえ
16	使用可能ディ スク	キャッシュのディスク部分のオブジェクト が使用可能なディスク・スペース容量	いいえ
17	メモリー・オ ブジェクト・ カウント	キャッシュのメモリー部分のオブジェクト 数	いいえ
18	ファイル・オ ブジェクト・ カウント	キャッシュのディスク部分のオブジェクト 数	いいえ
19	セッション・ カウント	キャッシュに対して現在活動状態のセッシ ョン数	いいえ

エラー・ログ・レベルの設定

Net.Data はエラー・ログを提供してくれるので、Net.Data システムでのエラー問題あるいはパフォーマンス上の問題をモニターすることができます。

Net.Data のエラー・ログを使用している場合、多数のメッセージがエラー・ログに書き込まれていると、システムのパフォーマンスに影響がでることに気が付くかもしれません。たとえば、Net.Data が検出できないマクロにユーザーがアクセスするたびに、Net.Data は出力としてメッセージをエラー・ログに渡します。

パフォーマンスへの影響を小さくするために、DTW_LOG_LEVEL キーワードを使用して Net.Data マクロに設定されているエラー・ログのログ記録レベルをチェックしてみてください。このレベルが WARNING に設定されている場合は、そのレベルを ERROR に下げてください。パフォーマンスが若干向上します。あるいは OFF にしてみてください。この場合はパフォーマンスが大きく向上します。

言語環境の最適化

以下のセクションでは、Net.Data 提供の言語環境の使用時にパフォーマンスを向上させるための手法について説明します。

- 『REXX 言語環境』
- 252ページの『SQL 言語環境』
- 254ページの『システムおよび Perl 言語環境』

REXX 言語環境

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

- REXX プログラムを可能な限り結合する。サイズの大きなプログラムでもその数が少ないほど、小さなプログラムが数多くある場合よりもパフォーマンスは向上します。それは、REXX language 関数がマクロで呼び出されるたびに、REXX インタープリターが初期化されるからです。
- REXX プログラムは、Net.Data のマクロにインラインで組み込まないで、外部ファイルに保管する。
- 外部 REXX プログラムの場合は、%EXEC ステートメントのコマンド行ではグローバル変数を参照する。

- 入力専用パラメーターは、グローバルな Net.Data 変数を定義し、その変数を参照することにより、直接 REXX プログラムに渡す。インライン REXX プログラムの場合は、グローバル変数を直接 REXX ソースで参照する。

SQL 言語環境

以下のセクションでは、データベースと SQL 言語環境に関するパフォーマンス手法をいくつか説明します。DB2 のパフォーマンスの考慮事項については、Web 上で <http://review.ibm.com/software/data/db2/performance> にアクセスしてください。

データベース手法

以下に示した要約は、データベース・アクセスを改善する最も簡単な手法をいくつか概説したものです。

- データベースを活動化する。コマンド `db2 activate database databaseName` を発行すると、データベースへの接続時間は大幅に短縮されます。DB2 のデータベース活動化コマンドの詳細は、*DB2 管理の手引き*を参照してください。
- 数値変換をしない。列値とリテラル値を比較する場合、同じデータ型と属性を指定するようにしてください。DB2 は、リテラル値の精度が列の精度よりも高ければ、名前付き列に索引を使用しません。比較される 2 つの項目が異なるデータ型の場合、DB2 はその値のどちらか一方を変換しなければなりません。(マシンの精度限界のため) これにより精度が落ちます。

たとえば、EDUCLVL をハーフワードの整数値 (SMALLINT) とします。次のように指定します。

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

次のような指定はしないでください。

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

- 文字ストリングを埋め込まない。固定長の文字ストリングの列値をリテラル値と比較する場合、同じデータ長を使用するようにしてください。DB2 は、リテラル値が列長よりも長い場合、索引を使用しません。

たとえば、EMPNO が CHAR(6) で、DEPTNO が CHAR(3) とします。次のように指定します。

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

次のような指定はしないでください。

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


- % あるいは _ 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 の述部のパターンで使用する場合は、選択したい行の列値に類似した文字สตริงを指定してください。これらを文字สตริงの真ん中、あるいは最後の文字を示すのに使用すると、LIKE パターンは索引を利用することができます。たとえば、以下のような場合です。

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

ただし、文字สตริงの先頭で使用すると、LIKE パターンは、走査する行数を制限するために LASTNAME 列で定義されている索引を DB2 が使用できなくなります。たとえば、以下のような場合です。

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

これらの記号は文字สตริงの先頭では使用しないでください。特に、大きな表にアクセスする場合には使用しないでください。

SQL 言語環境の手法

- 結果セットに大量の行が含まれている場合は、START_ROW_NUM および RPT_MAX_ROWS を使用して、ブラウザーに戻される結果セットのサブセットを指定することができます。START_ROW_NUM は、戻りサブセットの最初の行を、RPT_MAX_ROWS は、ページへの戻り行数を指定します。START_ROW_NUM をリンクで使用して、結果の次のページを表示することができます。

カーソルが要求間で保持されないため、Net.Data はページごとに照会を再発行します。

- 静的な SQL を使用するストアード・プロシージャの呼び出しを考慮してみる。動的 SQL が実行時に作成されるのに対して、静的 SQL はプリコンパイル段階で作成されます。SQL 言語環境では動的 SQL が使用されます。動的 SQL の使用により、プログラムの実行時に SQL ステートメントが実行されます。ステートメントの準備にはその処理時間がさらに必要になるため、静的な SQL の方がより効率的でしょう。

- SQL 言語環境にパラメーターを渡す際には、DB2 の準備キャッシュを利用してください。最初に、キャッシング・パッケージと合わせてステートメントを作成できるように、DB2 の設定を行います。次に、マクロ内の Net.Data 変数 DTW_USE_DB2_PREPARE_CACHE を YES に設定します。この設定により、パラメーター・マーカーを使用して SQL ステートメントが再書き込みされるので、DB2 準備キャッシュを利用することができます。これにより、繰り返しの結果セットを検索する必要はなくなり、パフォーマンスが向上します。

Net.Data では、照会の最適化に変数置換を使用するため、SQL ステートメントにおいて単一引用符は正しく使用するようになっています。たとえば、列が文字列を戻す場合、DTW_ADDQUOTE() 関数を使用して、文字列の単一引用符を拡張します。

システムおよび Perl 言語環境

入力専用パラメーターは、システムまたは Perl 言語環境が呼び出すプログラムに直接渡してください。それには、グローバルな Net.Data 変数を定義し、それを参照します。外部のプログラムおよび Perl スクリプトの場合、%EXEC ステートメントのコマンド行でその変数を参照します。インラインの Perl スクリプトの場合は、その変数を Perl のソースで直接参照します。

第8章 Net.Data のログ記録

Net.Data は、Net.Data のパフォーマンスのモニターとエラーのトラブルシューティングを行う際に使用するいくつかのログを提供しています。Net.Data のログには以下のものがあります。

Net.Data のエラー・メッセージ・ログ

すべての Net.Data のエラー・メッセージを含むログ。

Live Connection のログ

Live Connection のエラー・メッセージと Net.Data、Live Connection および DB2 データベース間の通信を含むログ。ログ記録は、DB2 データベース用の DTW_SQL および DTW_ODBC 言語環境で使用可能です。

- 『Net.Data エラー・メッセージのログを収集する』
- 258ページの『Live Connection のクライアントとエラー・メッセージのログ記録』

Net.Data エラー・メッセージのログを収集する

Net.Data はエラー・メッセージを Net.Data エラー・ログ・ファイル `netdata.log` に書き込みます。エラー・ログの最大サイズは、Net.Data によって 500 KB に固定されています。これはおよそ 3000 ログ・エントリーです。

エラー・ログ・ファイルやアーカイブされたコピーは、Net.Data システムの問題の発生を判別するために、定期的にブラウズすることができます。

Net.Data のエラー・ログの活動化方法:

- Net.Data のログ記録の構成変数 `DTW_LOG_DIR` を設定する。

`DTW_LOG_DIR path`

ここで、*path* はエラー・ログ・ファイルを保管するディレクトリです。

- Net.Data のログ記録変数 `DTW_LOG_LEVEL` をマクロに設定します。

`@DTW_ASSIGN(DTW_LOG_LEVEL, "level")`

ここで、*level* はログ記録のレベルです。指定できる値は以下のとおりです。

オフ Net.Data はエラー・ログを収集しません。これがデフォルトです。

エラー Net.Data はエラー・メッセージのログを収集します。

このセクションでは、以下のログ記録のトピックについて説明します。

- 『Net.Data エラー・ログの計画の作成』
- 257ページの『Net.Data ログ収集レベルの制御』
- 257ページの『ログ収集されない Net.Data エラー・メッセージのタイプ』
- 257ページの『Net.Data エラー・ログ・ファイルのサイズとローテーション』
- 257ページの『Net.Data エラー・ログ・フォーマット』

Net.Data エラー・ログの計画の作成

エラーをログ記録する場合には、以下の項目について計画を立てる必要があります。

- ディスク・スペースの決定。
エラー・ログの収集をする場合には、エラー・ログ用のディスク・スペースが必要です。
- Net.Data を構成する。
Net.Data システム全体に対してエラー・ログを制御する場合には、Net.Data 初期設定ファイルの構成変数の 1 つ DTW_LOG_DIR を設定します。
この変数は、DTW_LOG_LEVEL 変数がマクロの中でエラーや警告に設定されている場合でも、エラー・ログを収集するために必要です。19ページの『DTW_LOG_DIR と DTW_LOG_LEVEL: エラー・ログ変数』を参照して、初期設定ファイルの更新方法を理解してください。
- Net.Data マクロを作成します。
マクロで DTW_LOG_LEVEL キーワードを使用して、ログ記録のレベルを設定します。
- Net.Data を実行します。
エラーのログ記録を行っている場合には、Net.Data システムにエラーがないか、エラー・ログとアーカイブ・ファイルをチェックすることができます。
- 調整。
ログの収集はパフォーマンスに影響するため注意してください。パフォーマンスに関する情報は、251ページの『エラー・ログ・レベルの設定』を参照してください。

Net.Data ログ収集レベルの制御

DTW_LOG_LEVEL 変数を使用してログ記録のレベルを指定できます。このキーワードを Net.Data マクロで定義します。この変数には以下の 3 つの設定値があります。

オフ Net.Data はエラー・ログを収集しません。これがデフォルトです。

エラー Net.Data はエラー・メッセージのログを収集します。

警告 Net.Data は、エラー・メッセージだけでなく、警告のログも収集します。

ログ収集されない Net.Data エラー・メッセージのタイプ

Net.Data は、マクロの MESSAGE セクションで明示的に処理されるエラーはログに記録しません。

Net.Data エラー・ログ・ファイルのサイズとローテーション

ログ・ファイルの最大サイズは 500 KB です。このサイズで、およそ 3000 エントリーのログが保持できます。

ログ・ファイルが最大サイズに到達すると、ファイルは netdata.logMMMDDYYYY_nn にアーカイブされます。

ここで、

MMM 月 (Jan ~ Dec)

DD 日付

YYYY 年

nn 01 から 99 までの数字。これによって各アーカイブ・ファイルを 1 日の中で一意的に識別します。

ログの収集はオリジナルのファイルで継続されます。

Net.Data エラー・ログ・フォーマット

ログ・ファイルのエントリーのフォーマットは以下の通りです。

[DD/MMM/YYYY:HH:MM:SS] [MACRO] [BLOCK] [PID#] [TID#] error_message

パラメーター :

DD 日付

MMM 月 (Jan ~ Dec)

YYYY 年

HH 時間 (00 ～ 23)

MM 分 (00 ～ 59)

SS 秒数 (00 ～ 59)

MACRO

エラー・メッセージを生成したマクロ

BLOCK

エラー・メッセージを生成した HTML ブロックの名前

PID# エラー・メッセージを生成したプロセスのプロセス ID 番号。この ID は、複数の Net.Data プロセスがログ・ファイルに書き込みを行うために必要です。

TID# エラー・メッセージを生成したスレッドのスレッド ID 番号。この ID は、同一の Net.Data プロセスから複数のスレッドがログ・ファイルに書き込みを行うために必要です。

error_message

エラー・メッセージのテキスト

Live Connection のクライアントとエラー・メッセージのログ記録

Live Connection は、メッセージと、Live Connection、Net.Data、および DB2 データベースの間での通信を Live Connection のログ・ファイルに記録します。ログ・ファイルの最大サイズは、Net.Data により、1 MB、およそ 1200 のログ項目に固定されています。

ログ・ファイルあるいはアーカイブされたログ・ファイルのコピーは、クライアントあるいは DB2 データベースに問題がないかを判別するために定期的にブラウズすることができます。

Live Connection のログを活動化する方法:

次のように -l 属性を指定して接続管理プログラムを開始します。

```
dtwcm -l [level]
```

ここで、*level* はログ記録のレベルです。指定できる値は以下のとおりです。

normal

Live Connection はクライアントの活動、関連する DB2 SQL ステートメントと状況メッセージ、および Live Connection のエラー・メッセージをすべてログに記録します。

minimal

Live Connection はデータベース照会や、結果セット内の行数など、基本的な情報だけをログに記録します。

このセクションでは、以下のログ記録のトピックについて説明します。

- 『Live Connection ログの計画の作成』
- 『Live Connection ログ収集レベルの制御』
- 260ページの『ログ収集されない Live Connection メッセージのタイプ』
- 260ページの『Live Connection のログ・ファイル名』
- 261ページの『Live Connection のログ・ファイル・サイズおよびローテーション』
- 261ページの『Live Connection のログ・フォーマット』

Live Connection ログの計画の作成

メッセージをログに記録する場合は、以下の項目について計画を立てる必要があります。

- ディスク・スペースの決定。
エラーをログに記録するのであれば、ログ・ファイル用のディスク・スペースを確保しておく必要があります。
- 接続管理プログラムの実行
ログ記録を活動化するには、dtwcm コマンドで属性を入力します。その構文については、258ページの『Live Connection のクライエットとエラー・メッセージのログ記録』を参照してください。
ログ記録を取っている場合は、クライエットにエラーがないかログおよびアーカイブ・ファイルをチェックできます。
- 調整。
ログの収集はパフォーマンスに影響するため注意してください。パフォーマンス上の問題と『Live Connection ログ収集レベルの制御』については251ページの『エラー・ログ・レベルの設定』を参照してください。

Live Connection ログ収集レベルの制御

接続管理プログラムの起動中に、dtwcm コマンドでログ記録のレベルを指定できます。dtwcm コマンドの -l 属性には 2 つの設定値があります。

normal

Live Connection はクライアントの活動、関連する DB2 SQL ステートメントと状況メッセージ、および Live Connection のエラー・メッセージをすべてログに記録します。

minimal

Live Connection は基本的なメッセージだけをログに記録します。このオプションを指定するとログに記録されるメッセージが少なくなります。

ログ収集されない Live Connection メッセージのタイプ

Live Connection は、Net.Data のエラー、あるいはマクロの MESSAGE セクションで明示的に処理されるエラーはログに記録しません。

Live Connection のログ・ファイル名

Live Connection は、接続管理プログラムと各クライアント用のログ・ファイルを作成します。以下に示したリストは、名前の形式を説明したものです。

接続管理プログラム・ファイル

形式:

`conman-process_id-DDMMYYYYHHMMSS.log`

パラメーター :

process_id

接続管理プログラムのプロセスの識別子

DD

日付

MMM

月 (Jan ~ Dec)

YYYY

年

HH

24 時形式の時間

MM

分

SS 秒

例:

conman-513-01Feb1999095639.log

クライアント・ファイル

形式:

cliett-process_id-DDMMYYYYHHMMSS.log

パラメーター :

process_id

クライアント・スレッドの識別子

DD

日付

MMM

月 (Jan ~ Dec)

YYYY

年

HH

24 時形式の時間

MM

分

SS 秒

例:

cliett-592-01Feb1999095647.log

Live Connection のログ・ファイル・サイズおよびローテーション

ログ・ファイルの最大サイズは 1 MB です。このサイズで、およそ 6000 のログ項目を収容できます。ログ・ファイルが最大サイズに達すると、プロセスはオリジナル・ログ・ファイルをクローズし、新規ログ・ファイルを作成し、その新規ファイルにログ収集を続けます。

ログ・ファイルは、dtwcm および dtwcdb2 と同じディレクトリーに配置されます。

Live Connection のログ・フォーマット

ログ・ファイルのエントリーのフォーマットは以下の通りです。

--process_type-DD/MMM/YYYY:HH:MM:SS-PID#--
message_text

パラメーター :

process_type

dtwcm または cliet のいずれか。これは、メッセージをログ収集した
のが接続管理プログラムか、クライアントかによります。

DD 日付

MMM 月 (Jan ~ Dec)

YYYY 年

HH 時間 (00 ~ 23)

MM 分 (00 ~ 59)

SS 秒数 (00 ~ 59)

PID# メッセージを生成したプロセスのプロセス ID 番号。

message_text

メッセージのテキスト。

例 1: 接続管理プログラムのログ記入項目。

```
--dtwcm-02/Mar/1999:13:43:07-330--  
Creating connection manager ...successfully  
Reading configuration info ...  
Completing initialization ...  
Initializing cm server ... successfully  
Initializing NLS environment ... successfully  
Detecting cliette ./dtwcdb2 for DTW_SQL:CELDIAL:  
    Min process(es) = 1,  
    Priv Port = 7100.  
Starting 1 cliettes for DTW SQL:CELDIAL.  
Started: ./dtwcdb2 7128 7100 7200 DTW_SQL:CELDIAL LOG_MAX , pid: 213  
1 cliettes for DTW_SQL:CELDIAL started.  
...
```

例 2: クライエットのログ記入項目。

```
--cliet-02/Mar/1999:13:43:08-335--  
Cliette starting ...  
Cliette: DTW_SQL:SAMPLE, database: SAMPLE, user: *USE_DEFAULT  
Making a new connection to database: SAMPLE, user: *USE_DEFAULT.  
Calling SQLAllocHandle for environment ...  
Calling SQLAllocHandle for connection ...  
Calling SQLSetConnectAttr ...  
Calling SQLConnect ...
```

```
Connecting to database: SAMPLE sucessfully.  
Telling CM the cliette is ready ...  
Ready and waiting for command from CM ...
```

付録A. 参考文献

Net.Data テクニカル・ライブラリー

Net.Data テクニカル・ライブラリーは、以下の Net.Data Web サイトで入手できます。

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

文書	説明
<ul style="list-style-type: none">• <i>Net.Data</i> 管理およびプログラミングの手引き OS/390 版• <i>Net.Data</i> 管理およびプログラミングの手引き OS/2、Windows NT、および UNIX 版• <i>Net.Data</i> 管理およびプログラミングの手引き OS/400 版	Net.Data のインストール、構成、および起動に関する概念およびタスクについての情報が含まれています。また、Net.Data マクロの作成、Net.Data パフォーマンス向上の手法、Net.Data 言語環境の使用、接続管理、および Net.Data ログおよびトレースを使用した問題解決およびパフォーマンス・チューニングに関する説明がされています。
<i>Net.Data</i> 解説書	Net.Data マクロ言語、変数、および組み込み関数について説明します。
<i>Net.Data</i> 言語環境解説書	Net.Data 言語環境インターフェースについて説明します。
<i>Net.Data</i> メッセージおよびコード	Net.Data エラー・メッセージおよび戻りコードのリストです。

付録B. Net.Data for AIX

AIX の詳細は、Net.Data とともに出荷される README ファイルに記述されています。README ファイルは、以下の情報を含んでいます。

- 要件
- インストール
- 構成
- アンインストール

言語環境の共用ライブラリーをロードする

AIX プラットフォームで言語環境を作成するときには、共用ライブラリーをロードする必要があります。AIX では、言語環境が、Net.Data に呼び出されるルーチンを提供するために必要で、`dtw_initialize()` や `dtw_execute()` などの言語環境インターフェース・ルーチンのアドレスを戻します。

Net.Data は、AIX の言語環境から言語環境インターフェース・ルーチンへのポインターを検索するために、`dtw_fp` 構造を使用します。この構造は、以下のような形式となります。

```
typedef struct dtw_fp {  
    int (* dtw_initialize_fp)(); /* dtw_initialize function pointer */  
    int (* dtw_execute_fp)();   /* dtw_execute function pointer */  
    int (* dtw_cleanup_fp)();   /* dtw_cleanup function pointer */  
} dtw_fp_t;
```

この構造は、共用ライブラリーがロードされると、Net.Data によって `dtw_getFp()` ルーチンの中でパラメーターとして、言語環境に渡されます。

`dtw_fp` 構造は、唯一のパラメーターとして渡されます。この構造は、各サポートされるインターフェースに対して、1 つのフィールドを含みます。これらのフィールドを設定するのは、言語環境の役割です。言語環境が、指定されたインターフェースを提供している場合、言語環境は、そのフィールドにそのインターフェースの関数ポインターをセットします。言語環境が、指定されたインターフェースを提供していない場合、言語環境はそのフィールドを `NULL` にセットします。プログラム・テンプレートの `dtw_getFp()` ルーチンは、このルーチンの正確なインプリメンテーションを示しています。

共用ライブラリーがロードされる際に、Net.Data がこのルーチンへのポインターを取得するためには、`dtw_getFp` ルーチンが、共用ライブラリーのエクスポ

ート・ファイルで指定された最初のエントリー・ポイントでなければなりません。すべての使用可能な言語環境インターフェース・ルーチンをサポートする、dtwsampshr.o という名前のライブラリーのサンプル・エクスポート・ファイルは、以下のようになっています。

```
#!dtwsampshr.o
dtw_getFp
dtw_initialize
dtw_execute
dtw_cleanup
```

REXX 環境のパフォーマンスの向上

AIX システムで REXX 言語環境への呼び出しが多数ある場合、RXQUEUE_OWNER_PID 環境変数を 0 にセットすることを考慮してください。REXX 言語環境への呼び出しを多数行うマクロは、簡単に多くのプロセスを生成し、システム資源を消費します。

以下の 3 つの方法のいずれかで環境変数を設定します。

- DTW_SETENV 組み込み関数を使用して、マクロで設定する場合

```
@DTW_rSETENV("RXQUEUE_OWNER_PID", "0")
```

- AIX システム環境ファイルで、設定する場合

```
/etc/environment: RXQUEUE_OWNER_PID = 0
```

この方法は、マシン全体の REXX の動作に影響を与えます。

- HTTP Web サーバー環境ファイルで、設定する場合。たとえば、Domino Go Webserver の場合、次のステートメントを挿入します。

```
InheritEnv RXQUEUE_OWNER_PID = 0
```

この方法は、Web サーバーの REXX の動作に影響を与えます。

NLS 考慮事項

Net.Data は、Web サーバーと同じコード・ページを使用して実行します。ロケールに対して Net.Data が正しいコード・ページを使用するためには、Web サーバーが正しいコード・ページを使用している必要があります。たとえば、IBM Internet Connection Server を使用していて、韓国語のコード・ページを使用したい場合、サーバーを停止して、韓国語ロケールを使用してそのサーバーを再始動します。

```
stopsrc httpd
startsrc -s httpd -e "LC_ALL=ko_KR"
```


マクロが UTF-8 文字を含んでいるか、あるいはユーザーが Unicode データを含んでいる DB2 データベースに接続している場合は、INI ファイルの DTW_UNICODE 構成変数を UTF8 に設定してください。Net.Data は現在、マクロ内の UTF-8 文字をサポートしていますが、UTF-16 文字はサポートしていません。Net.Data は、UTF-8 または UCS-2 のいずれかでエンコードされた DB2 データベース・データを処理することができます。Net.Data の出力は常に UTF-8 です。

パフォーマンスのヒント：ユーザーが 2 バイトのロケールで実行しており、string または work の組み込み関数が常に単一バイトの文字ストリングをプロセスする場合は、DTW_MBMODE を NO に設定して不必要な型変換を避けてください。

付録C. Net.Data ウィザード

Net.Data ウィザードは、カスタマイズされた Net.Data アプリケーションを作成する、迅速で、簡単な方法をユーザーに提供するように設計されています。まずウィザードを選択して、わずかな質問に答えると、Net.Data はカスタマイズされたアプリケーションをそのユーザー用に作成します。

Net.Data は、マクロの作成方法および Net.Data 機能の使用方法の学習中に使用する、以下の SmartGuides を提供します。

ドリルダウン

この SmartGuide は、ユーザーの既存データベース・テーブルを取り込み、Web で使用可能なドリルダウン・アプリケーションを作成するため、ユーザーは詳細データをいろいろなレベルから詳細にアクセスできるようになります。オプションで、ドリルダウン SmartGuide は、データベースに接続して、データベースの情報を収集することができます。5 つの Web レポートまでカスタマイズすることが可能です。生成されたマクロは、Web レポートを自動的にリンクするために、データベースに保管されたプライマリー・キーの情報と外部キーの情報を使用します。

ストアード・プロシージャ

この SmartGuide は、ユーザーのデータベースに接続して、データベースに登録されているすべてのストアード・プロシージャのリストを検索します。ストアード・プロシージャを選択します。SmartGuide は、ユーザーのストアード・プロシージャを呼び出す Net.Data マクロを生成します。マクロが生成されたら、そのマクロを変更したり、既存の Net.Data アプリケーションに統合したりすることができます。

連絡先 この SmartGuide は、名前、住所、電話番号、およびその他の重要な連絡先情報を保管するための、Web で使用可能な住所録を生成します。この住所録は、即時に連絡先にアクセスするための検索関数が備えられています。生成される連絡先アプリケーションには、概要報告書または明細報告書のいずれかを組み込むことができます。さらに、カスタマイズされたレポートを組み込みこともできます。

Net.Data SmartGuide パッケージの最新バージョンについては、Net.Data Web サイト (<http://www.ibm.com/software/data/net.data>) を参照してください。

この付録では、以下のトピックを説明しています。

- 『事前準備』
- 『ウィザードの実行』

事前準備

SmartGuides を実行して、Net.Data マクロを生成するには、以下のソフトウェアがインストールされていなければなりません。

- Java 開発キット (JDK) または Java 実行時環境 (JRE) 1.1.x
- Net.Data バージョン 2 またはそれ以降
- IBM ユニバーサル・データベース (UDB) 5.0 またはそれ以降
- REXX (ドリルダウン SmartGuide に必須)

ウィザードの実行

Net.Data ウィザードは、コマンド行から開始します。これは、ファイル NetDataSmartGuides.jar に含まれています。

Java 開発キット (JDK) を使用してウィザードを開始するには

1. 以下の行を CLASSPATH 環境変数に追加します。

```
[Path]NetDataSmartGuides.jar
```

ここで [Path] は、NetDataSmartGuides.jar ファイルへのパスで、オプションです。

2. ドリルダウン、およびストアード・プロシーチャーのウィザードのデータベース接続機能を使用する場合は、UDB JDBC ドライバーを CLASSPATH 環境変数に追加します。

Windows NT および OS/2 オペレーティング・システムの場合は、CLASSPATH 環境変数に次の行を追加します。

```
[Path]NetDataSmartGuides.jar;[UDBInstallationPath]¥java¥db2java.zip
```

UNIX オペレーティング・システムの場合は、CLASSPATH 環境変数に次の行を追加します。

```
[Path]NetDataSmartGuides.jar:[UDBInstallationPath]¥java¥db2java.zip
```

ここで、[Path] は NetDataSmartGuides.jar ファイルへのパスで、オプションです。[UDBInstallationPath] は、C:¥SQLLIB などの、ユーザー UDB インストールへのパスです。

3. ウィザードを開始します。

- UNIX オペレーティング・システムの場合、次のコマンドを入力します。

```
java TaskGuide LaunchPad.class
```

- Windows NT および OS/2 オペレーティング・システムの場合、次のファイルを実行します。

```
SmartGuides.cmd
```

図29 に示すように、ランチパッドが開き、使用可能なウィザードのリストが表示されます。

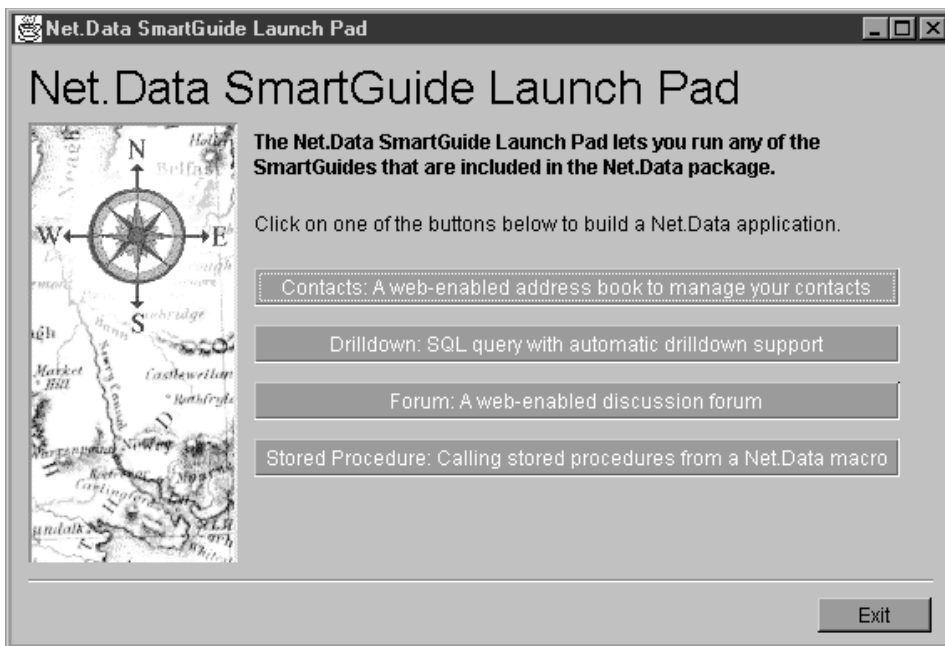


図29. ウィザードランチパッド

4. 実行するウィザードの名前をクリックします。

Java 実行時環境 (JRE) を使用してウィザードを開始するには

1. Windows NT オペレーティング・システムの場合は、「開始 (Start) -> プログラム (Programs) -> Net.Data -> SmartGuides」を選択します。これにより、SMARTGUIDES.BAT と呼ばれるバッチ・ファイルを実行します。その他のオペレーティング・システムの場合は、以下のコマンドを入力してウィザードを実行します。

```
jre -cp [Path]NetDataSmartGuides.jar TaskGuide LaunchPad.class
```

ここで *[Path]* は、NetDataSmartGuides.jar ファイルへのパスで、オプションです。

273ページの図29 に示すように、ランチパッドが開き、使用可能なウィザードのリストが表示されます。

2. ドリルダウンおよびストアード・プロシージャウィザードのデータベース接続機能を使用する場合は、以下のコマンドを入力します。

Windows NT および OS/2 オペレーティング・システムの場合:

```
jre -cp [Path]NetDataSmartGuides.jar;[UDBInstallationPath]
¥java¥db2java.zip TaskGuide LaunchPad.class
```

UNIX オペレーティング・システムの場合:

```
jre -cp [Path]NetDataSmartGuides.jar:[UDBInstallationPath]
¥java¥db2java.zip TaskGuide LaunchPad.class
```

ここで、*[Path]* は NetDataSmartGuides.jar ファイルへのパスで、オプションです。 *[UDBInstallationPath]* は、C:¥SQLLIB などの、ユーザー UDB インストールへのパスです。

3. 実行するウィザードの名前をクリックします。

付録D. Net.Data SQL Assist を使用して SQL ステートメントを作成する

Net.Data SQL Assist は、Java ベースの SQL ステートメント・ビルダーで、SQL ステートメント作成のプロセスを解説する使いやすい GUI を、ユーザーに提供します。Net.Data SQL Assist を使用すると、以下を行うことができます。

- SELECT、INSERT、UPDATE、および DELETE ステートメント (SELECT DISTINCT を含む) を作成する
- 値の検索、AND または OR、および型依存入力フィールドを使用して、複数の条件を作成する
- テーブル JOINS (内部、右外部、および左外部) を定義する
- 表示する列を選択する
- ソート順を選択する
- 条件、値、およびソートで使用するユーザー定義変数を入力する

SQL ステートメントを作成すると、以下を行うことができます。

- SQL ステートメントをファイルに保管する
- SQL ステートメントを含むマクロを生成および保管する
- SQL ステートメントまたはマクロをクリップボードにコピーする

この付録では、以下のトピックを説明しています。

- 『事前準備』
- 276ページの『Net.Data SQL Assist を実行する』

事前準備

Net.Data SQL Assist を実行するには、以下のソフトウェアがインストールされていなければなりません。

- Java 開発キット (JDK) または Java 実行時環境 (JRE) 1.1.x
- JDBC が使用可能なデータベース

JDBC およびその他必要なサーバー・スタートアップでのデータ・ソースのアクセスに関する詳細については、データベースのドキュメンテーションを参照してください。たとえば、リモートから DB2 UDB v5.0 データ・ソースをア

クセスしているときには、データベース・サーバーで JDBC サーバー (db2jstrt) が実行中でなければなりません。

Net.Data SQL Assist を実行する

Net.Data SQL Assist は、コマンド行で開始するもので、ファイル `{inst_dir}/assist/NetDataAssist.jar` に含まれています。

Java 開発キット (JDK) を使用して **Net.Data Assist** を開始するには、以下のようになります。

Net.Data Assist を開始するには、次のコマンドを入力します。

```
java -classpath %CLASSPATH%;{inst_dir}/assist/NetDataAssist.jar NetDataAssist
```

Java 実行時環境 (JRE) を使用して **Net.Data Assist** を開始するには、以下のようになります。

Net.Data Assist を開始するには、次のコマンドを入力します。

```
jre -cp %CLASSPATH%;{inst_dir}/assist/NetDataAssist.jar NetDataAssist
```

「**次へ (Next)**」 ボタンをクリックしながらログオンに使用されるウィンドウをナビゲートし、SQL ステートメントを構成して、Net.Data マクロを生成します。

付録E. Net.Data サブレットと NetObjects Fusion NOF プラグインを使用する

Net.Data は、Net.Data のサブレット用の NetObjects Fusion プラグインを提供しています。Net.Data サブレットは、99ページの『Net.Data サブレット』に記述されています。

NetObjects Fusion (NOF) を使用すると、既存の Net.Data マクロを統合できます。これによって、Web サイト管理機能と使いやすいグラフィカル・ユーザー・インターフェースとを持つ優れた統合化を実現できます。

この付録では、以下のトピックを説明しています。

- 『NetObjects Fusion プラグインについて』
- 278ページの『NetObjects Fusion プラグインのインストール』
- 278ページの『NetObjects Fusion 用 Net.Data プラグインのセットアップ』
- 282ページの『NOF プラグイン付きサブレットの表示』

NetObjects Fusion プラグインについて

NetDataServlet.NFX プラグインは Net.Data サブレット と一緒に機能します。この NOF プラグインは、既存の Net.Data マクロ、あるいは単一の Net.Data 関数の呼び出しをサポートします。このプラグインは、Net.Data をサブレットまたはサーバー・サイド・インクルード (SSI) として起動するための HTML を生成します。Web サーバーが Net.Data を起動すると、Net.Data マクロあるいは関数が実行されます。Net.Data サブレット プラグインを使用して、マクロおよび関数サブレット のどちらかを NOF 管理の Web サイトに組み込みます。これについては、278ページの図30 で説明しています。このプラグインは、マクロの構築を自動化するために選択されたデフォルト値を、多数の基本マクロあるいは関数のパラメーターに提供します。プラグインを使用するには、NOF とプラグイン・ファイルをインストールして構成しなければなりません。

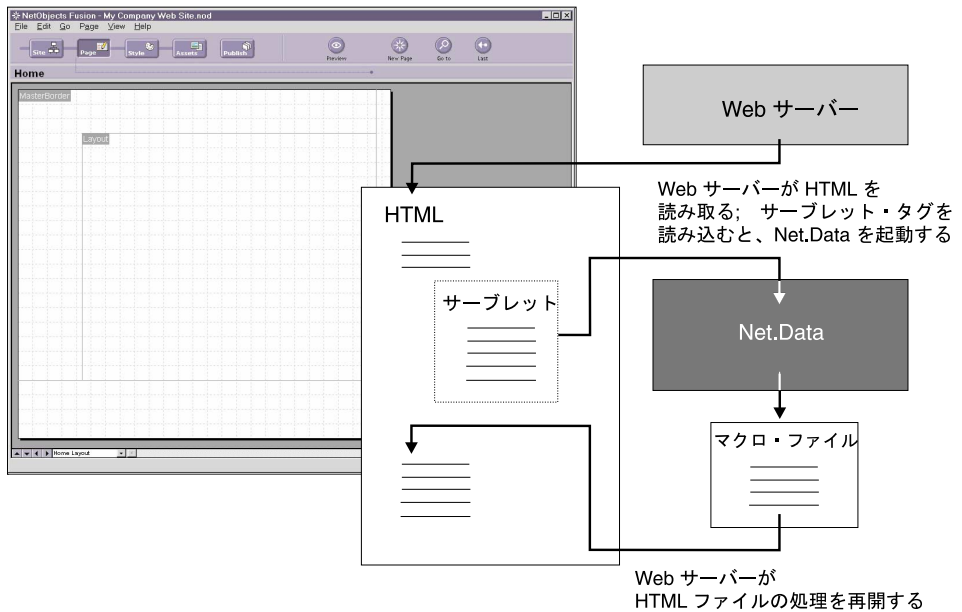


図 30. Net.Data サブレット・プラグイン

NetObjects Fusion プラグインのインストール

ソフトウェアおよびハードウェア要件:

NOF プラグインは、NetObjects Fusion Version 2.0 またはそれ以降のバージョンが必要です。

インストール方法: <inst_dir>%fx%NetDataServlet.nfx および <inst_dir>%fx%NetDataServlet.gif を <NetObjects fusion>%components% にコピーします。

NetObjects Fusion 用 Net.Data プラグインのセットアップ

NOF を使用して操作しているサーブレットのプロパティーを変更することができます。

1. NetObjects Fusion をオープンします。
2. 以下のように、NetObjects Fusion (NOF) の「**ツール (Tools)**」パレットから、「**NetObjects 構成要素 (NetObjects Components)**」ボタンを選択し

ます。「プラグイン (plug-in)」ボタン  が、「**ツール (Tools)**」パレットの下部に表示されます。

3. これら 6 つの「ツール (Tools)」パレットのボタンから、以下の

「NetObjects 構成要素 (NetObjects Components)」ボタン  を選択します。

4. NOF キャンバス上に、エリアをマークして、選択したプラグインを置く場所を指定します。この場所にサーブレットの結果が表示されます。「インストールされている構成要素 (Installed Components)」ウィンドウが開き、プラグインの選択肢リストが表示されます。サーブレットのプラグインがリストにない場合は、パスおよびファイル名フィールドを使用して、マクロまたは関数サーブレットと一緒に使用するプラグイン・ファイル名 `NetDataServlet.NFX` を指定します。
5. リストからサーブレット・プラグインを選択し、「了解 (OK)」押しボタンをクリックします。
プラグインは NOF キャンバス上のオブジェクトになります。

プラグイン・プロパティを変更する

Net.Data サーブレットのプラグインを使用して、マクロおよび関数サーブレットを変更することができます。

NOF で Net.Data サーブレット を変更するには、

1. NOF キャンバス上に、エリアをマークして、選択したプラグインを置く場所を指定します。この場所にサーブレットの結果が表示されます。「インストールされている構成要素 (Installed Components)」ウィンドウが開き、プラグインの選択肢リストが表示されます。サーブレット・プラグインがリストされていなければ、パスおよびファイル名フィールドを使用して、プラグイン・ファイル名、`NetDataServlet.NFX` を指定します。このファイルは、マクロまたは関数サーブレットと一緒に使用します。
2. リストから Net.Data サーブレット プラグインを選択し、「了解 (OK)」押しボタンをクリックします。
プラグインは NOF キャンバス上のオブジェクトになります。
3. Net.Data サーブレット プラグインのプロパティを、以下のように選択し、カスタマイズします。
 - a. NOF キャンバス上の Net.Data サーブレット プラグインを選択します。
「プロパティ (Properties)」パレットがオープンし、280ページの図31に示すようにプラグイン・プロパティが表示されます。

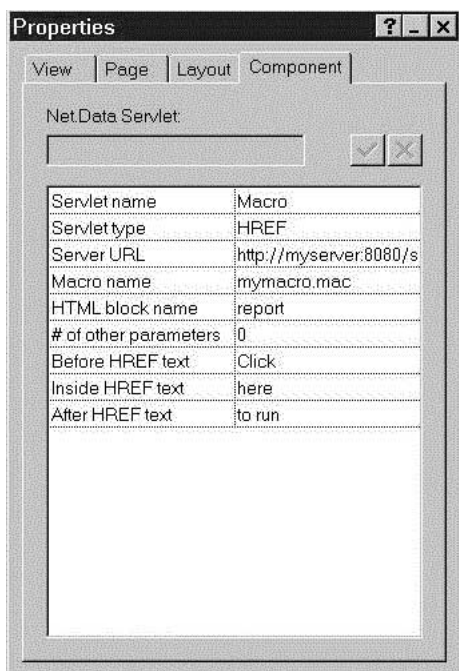


図 31. Net.Data サーブレット プロパティ・パレット

Net.Data サーブレット 用のプロパティ:

カスタマイズできるのは、以下のプロパティです。

サーブレット名

起動するサーブレット名、Function または Macro を選択します。どちらのサーブレット名を選択したかによって、異なるプロパティが表示されます。

サーブレット・タイプ

希望するサーブレット・タイプ、「SSI 実行 (SSI Submit)」、「HREF 実行 (HREF Submit)」、または「FORM 実行 (FORM Submit)」ボタンを選択します。どちらのサーブレット・タイプを希望するかによって、異なるプロパティが表示されます。

実行依頼ラベル

FORM 実行ボタン・タイプを選択する場合には、実行依頼ラベルのテキストを指定します。選択しない場合は、このプロパティは表示されません。

サーバー URL

HREF サーブレット・タイプを選択する場合には、サーバー URL を、サーブレット対応の Web サーバーに指定します。SSI を選択する場合には、このプロパティは表示されません。

マクロ名

マクロ・サーブレット 名を選択する場合には、既存の Net.Data マクロを指定して実行します。選択しない場合は、このプロパティは表示されません。

HTML ブロック名

マクロ・サーブレット 名を選択する場合には、Net.Data マクロで HTML ブロックの名前を指定し、実行します。選択しない場合は、このプロパティは表示されません。

関数タイプ

関数サーブレット 名を選択する場合には、関数タイプ、Function または SQL を選択して実行します。選択しない場合は、このプロパティは表示されません。

言語環境

関数サーブレット 名を選択する場合には、Net.Data 言語環境を指定して使用します。選択しない場合は、このプロパティは表示されません。

ステートメント

関数サーブレット 名を選択する場合には、ステートメントを指定して実行します。選択しない場合は、このプロパティは表示されません。

データベース

関数サーブレット名を選択する場合は、使用するデータベースの名前を指定します。選択しない場合は、このプロパティは表示されません。

その他のパラメーターの数

Net.Data へ渡すその他のパラメーターの数を指定します (最大 25)。それぞれのパラメーターに対して、パラメーターの名前とその値 (オプション) を入力します。

HREF 前テキスト

HREF サーブレット・タイプを選択する場合には、<a href> HTML タグの前のテキストが表示される前に表示されるテキストを指定します。この指定をしない場合には、このプロパティは表示されません。(オプション)。

HREF 内部テキスト

HREF サブレット・タイプを選択する場合には、`<a href>` HTML タグの内部に表示するテキストを指定します。この指定をしない場合には、このプロパティは表示されません。(オプション)。

HREF 後テキスト

HREF サブレット・タイプを選択する場合には、`<a href>` HTML タグの後ろのテキストが表示された後で表示されるテキストを指定します。この指定をしない場合には、このプロパティは表示されません。(オプション)。

SQL 確認事項!

HREF サブレット・タイプを選択して SQL 関数タイプを指定する場合には、HREF SQL ステートメントではスペース () 文字の代わりにプラス (+) 文字を使用します、という確認メッセージが表示されます。このテキストは変更できないか、ページが生成された後に表示されることはありません。選択しない場合は、このプロパティは表示されません。

- b. ユーザーのページにプロパティを定義した後で、「**パブリッシュ (Publish)**」押しボタンをクリックすると、Net.Data サブレット NOF プラグインを使用して、Web ページが生成され、表示されます。

注: SSI サブレット・タイプを選択すると、Web ページ・ファイルの拡張子は .shtml でなければなりません。NOF の「**プロパティ (Properties)**」パレットから、デフォルトのページを設定することができます。このページから、**Page** ノートブック・タブを選択し、「**カスタム名 (Custom names)**」押しボタンをクリックし、**Extension Type** フィールドに .shtml と入力します。

NOF プラグイン付きサブレットの表示

ユーザーのページにプロパティを設定した後で、「**パブリッシュ (Publish)**」押しボタンをクリックすると、プラグインを使用して、Web ページが生成され、表示されます。

付録F. 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。

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

284ページの図32 は、サンプル・マクロを示しています。 287ページの図33 は、組み込みファイルを示しています。

```

%{***** Sample Macro *****}
*   FileName = sqlsamp1.d2w                                     *
*   Description:                                               *
*       This Net.Data macro queries...                           *
*       - The EMPLOYEE table to create a selection list of      *
*       employees for display at a browser                       *
*       - The EMPLOYEE table to obtain additional information    *
*       about an individual employee                             *
*                                                                 *
*****%}
%{*****}
*   Include for global DEFINES -                                *
*****%}
%INCLUDE "sqlsamp1.hti"
%{*****}
*   Function: queryDB                                           Language Environment: SQL      *
*   Description: Queries the table designated by the variable myTable and *
*   creates a selection list from the result. The value of the variable *
*   myTable is specified in the include file sqlsamp1.hti.      *
*****%}
%FUNCTION(DTW SQL) queryDB() {
  SELECT FIRSTNME FROM $(myTable)
%MESSAGE {
  -204: {<p><b>ERROR -204: Table $(myTable) not found. </b>
        <p>Be sure the correct include file is being used.</b>
        %} : exit
  +default: "WARNING $(RETURN_CODE)" : continue
  -default: "Unexpected ERROR $(RETURN_CODE)" : exit
%}
%REPORT {
<select name=emp_name>
%ROW{
<option>$(V1)
%}
</select>
%}
%}
%{*****}
*   Function: fname                                           Language Environment: SQL      *
*   Description: Queries the table designated by the variable myTable for *
*   additional information about the employee identified by the *
*   variable emp_name.                                         *
*****%}
%FUNCTION(DTW SQL) fname(){
  SELECT FIRSTNME, PHONENO, JOB FROM $(myTable) WHERE FIRSTNME='$(emp_name)'
%MESSAGE {
  -204: "Error -204: Table not found "
  -104: "Error -104: Syntax error"
  100: "Warning 100: No records" : continue
  +default: "Warning $(RETURN_CODE)" : continue
  -default: "Unexpected SQL error" : exit
%}
%}
%}

```

図 32. サンプル・マクロ (1/3)


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

図 32. サンプル・マクロ (2/3)

```
%{*****
*   HTML block:    REPORT                               *
*   Description: Queries the EMPLOYEE table to obtain additional information *
*                  about an individual employee                               *
*****%}
%HTML(REPORT) {
<html>
<head>
<title>Obtain Employee Information</title>
</head>
<body>
<h3>You selected employee name = $(emp_name)</h3>
<p>Here is the information for that employee:
<pre>
@fname()
</pre>
<hr><a href="input">Return to previous page</a>
</body>
</html>
%}
%{      End of Net.Data macro 1 %}
```

図 32. サンプル・マクロ (3/3)


```

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

%{      End of include file  %}

```

図 33. 組み込みファイル

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な他社のプログラム、製品またはサービスを使用することができます。ただし、IBM によって明示的に指示されたものを除き、これらのプログラムまたは製品に関する稼働の評価および検査はお客様の責任で行っていただきます。

IBM および他社は、本書で説明する主題に関する特許権（特許出願を含む）商標権、または著作権を所有している場合があります。本書は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用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 は本書を“現存のまま”提供するものであり、明示または黙示にかかわらず、非侵害性、商業的な使用可能性、または特定の目的に対する適合性に関する黙示の保証を含み、かつそれには限定されない、いかなる保証も行いません。国によっては特定の商取引における明示または黙示の保証の放棄は認められていません。したがってこの記述はお客様に適用されない場合があります。

本書の情報は定期的に変更されており、それらの変更は本書の新しい版で取り入れられる予定です。IBM は本書に記載された製品およびプログラムを予告なく改良または変更する場合があります。

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。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

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 Canada Limited
Office of the Lab Director
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

本プログラムに関する上記の情報は、適切な条件の下で 사용할 수 있지만、有償の場合もあります。

本書において解説されているライセンス・プログラムおよびそのライセンス・プログラム資料は、「IBM 使用契約書」または「IBM プログラムのご使用条件」の契約条件に基づいて弊社から提供されるものです。

本書に含まれるパフォーマンス・データは、制御された環境下でのものです。したがって、他のオペレーティング環境で得られる結果は大幅に異なる場合があります。開発レベル・システムでいくつかの測定方法が存在する場合があります、それらの測定方法が一般的に使用可能なシステム上ですべて同じであるという保証はありません。さらに、測定値の中には統計での補外 (extrapolation) によって得られたものが含まれていることがあります。実際の結果は違っていることがあります。本書をご使用になる際には、特定の環境にデータが適用できるかどうかを確かめる必要があります。

IBM 以外の製品に関する情報は、それらの製品の提供元、その出版物、またはその他の公開された情報源から入手したものです。IBM はそれらの製品のテストは行っておらず、パフォーマンスの正確性、互換性、または IBM 以外の製品に関連するその他のいかなる主張も確認できません。IBM 以外の製品の機能についての質問は、それらの製品の提供元にご照会ください。

IBM の今後の方向性および予定に関する記述は、目標や目的を表明したものであり、予告なく変更または取り消しされることがあります。

本書には、日常の常務処理で用いられるデータや報告書の例が含まれています。これは説明に具体性を与えるために記載されたものであり、それらの例には、個人、企業、ブランドあるいは製品の名前が含まれている場合があります。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。

著作権のライセンス

本書には、IBM が説明するための一例として提供している簡単なプログラムが含まれています。これらの例は必ずしもすべての場合について完全にテストされたものではありません。IBM は、これらのプログラムの信頼性、可用性、および機能について法律上の瑕疵担保責任を含むいかなる明示または黙示の保証責任も負いません。本書中に含まれているプログラムは「現存するままの状態」で提供されます。IBM はプログラムの商業的な使用可能性および特定の目的に対する適合性については、いかなる保証も行いません。

それぞれの複製物、サンプル・プログラムのすべての部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (年).このコードの一部は、IBM Corp. のサンプル・プログラムの派生物です。 © Copyright IBM Corp. _年を入力してください_. All rights reserved.

商標

以下の用語は、IBM Corp. の米国およびその他の国におけるの商標です。

ACF/VTAM	IBM
AISPO	IMS
AIX	IMS/ESA
AIX/6000	LAN DistanceMVS
AIXwindows	MVS/ESA
AnyNet	MVS/XA
APPN	Net.Data
AS/400	OS/2
BookManager	OS/390
CICS	OS/400
C Set++	PowerPC
C/370	QBIC
DATABASE 2	QMF
DataHub	RACF
DataJoiner	RISC System/6000
DataPropagator	RS/6000
DataRefresher	S/370
DB2	SP
DB2 Connect	SQL/DS
DB2 Extenders	SQL/400
DB2 OLAP Server	System/370
DB2 Universal Database	System/390
Distributed Relational Database Architecture	SystemView
DRDA	VisualAge
eNetwork	VM/ESA
Extended Services	VSE/ESA
FFST	VTAM
First Failure Support Technology	WebExplorer
	WIN-OS/2

以下の用語は、他社の商標または登録商標です。

Microsoft、Windows、Windows NT、および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

Tivoli および NetView は、Tivoli Systems Inc. の米国およびその他の国における商標です。

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

2 個のアスタリスク (**) で示されている他の会社名、製品名およびサービス名等はそれぞれ各社の商標または登録商標です。

用語集

絶対パス (absolute path). オブジェクトのフルパス名。絶対パス名は最上位のレベル、すなわち「ルート」ディレクトリー (これはスラッシュ (/) あるいは円記号 (¥) で識別される) で始まる。

API. アプリケーション・プログラミング・インターフェース (application programming interface)。Net.Data は、CGI プロセスよりもパフォーマンスを向上させるために 3 つの Web サーバー API をサポートしている。

アプレット (applet). HTML ページに組み込まれる Java プログラム。アプレットは Netscape Navigator などの Java 対応ブラウザで動作し、HTML ページが処理される際にロードされる。

アプリケーション・プログラミング・インターフェース (application programming interface (API)). オペレーティング・システムまたは別途発注可能なライセンス・プログラムによって提供される機能インターフェース。これにより、高水準言語で書かれたアプリケーション・プログラムが、特定のデータもしくは、オペレーティング・システムまたはライセンス・プログラムを使用できるようにする。Net.Data は、CGI プロセスよりもパフォーマンスを向上させるために、GWAPI、ISAPI、および NSAPI という Web サーバーのプロプラエタリー API をサポートしている。

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

キャッシュ (cache). 最近アクセスされたデータを格納するメモリーまたはディスクの一部。同一データへの連続アクセスの速度を高速にするように設計されている。キャッシュは、ネットワーク

を介してアクセス可能な、使用頻度の高いデータのローカル・コピーを保留するのに使用される場合が多い。

キャッシング (caching). ローカルに Web サーバーに要求した結果得られる使用頻度の高いデータを高速に取り出すために、情報を最新表示するときまで保管するプロセス。

キャッシュ管理プログラム (Cache Manager). 1 つのマシン用のキャッシュを管理するプログラム。複数のキャッシュを管理できる。

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

CLIETTE. Web サーバーからの要求に長期にわたって対応する Net.Data Live Connection における処理。この接続管理プログラムは、これらの要求に対応する CLIETTE 処理のスケジュールを行う。

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

コミットメント制御 (commitment control). Net.Data が実行されているプロセスで、資源に対する操作が作業単位の一部であるプロセス内の境界の確立。

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

接続管理プログラム (Connection Manager). Live コネクションのサポートに必要とされる Net.Data 内の、実行可能ファイル、dtwcm。

cookie. HTTP サーバーによって Web ブラウザーに送信され、次にブラウザーがそのサーバーに

アクセスするつど、送り返す情報のバケット。クッキーには、サーバーが選択する任意の情報を含めることができ、特に国籍をうたわない HTTP トランザクション間の状態を保持するのに使用される。 *Free Online Dictionary of Computing* より。

現行作業ディレクトリー (current working directory). すべての相対パス名を解決するための基準となる、プロセスのデフォルト・ディレクトリー。

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

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

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

データ型 (data type). 列およびリテラルの属性。

DBCLOB. 2 バイト文字ラージ・オブジェクト。

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

Domino Go Web サーバー (Domino Go Web server). 標準接続およびセキュア接続の両方を提供する、Lotus Corp. と IBM が提供する Web サーバー。GWAPI は、サーバーに提供されるインターフェース。

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

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

GWAPI. Go Web サーバー API。

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

HTTP. Hypertext transfer protocol (HTTP)。

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

hypertext transfer protocol (HTTP). Web サーバーとブラウザー間で使用する通信プロトコル。

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

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

ISAPI. Microsoft のインターネット・サーバー API。

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

言語環境 (language environment). Net.Data マクロから、DB2 などの外部データ・ソースや Perl などのプログラミング言語へのアクセスを行うモジュール。

Live コネクション (Live Connection). 接続管理プログラムと複数のクライアントで構成される Net.Data のコンポーネント。Live Connection は、データベースと Java 仮想マシン接続の再使用を管理する。

LOB. ラージ・オブジェクト (Large object)。

ミドルウェア (middleware). アプリケーション・プログラムとネットワークの中間にあるソフトウェア。これは、ネットワークを介した、クライアントのアプリケーション・プログラムとサーバー間の対話を管理する。

NSAPI. Netscape API。

ヌル (null). 情報が無いことを示す特殊な値。

パス (path). ファイルを探すのに使用する検索経路。

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

Perl. 解釈済みプログラミング言語。

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

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

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

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めると想定する。

TCP/IP. 伝送制御プロトコル / インターネット・プロトコル (Transmission Control Protocol / Internet Protocol)。

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

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

URL. Uniform resource locator (URL)。

uniform resource locator (URL). HTTP サーバー、および任意選択でディレクトリーとファイル名を指定するアドレス。たとえば、
`http://www.ibm.com/software/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, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

アクセス、DB2 の 171
アクセス、ODBC データベースの 169
アクセス、Oracle データベースの 170
アクセス権
 言語環境の 168
アクセス権限
 Net.Data のファイルのための 69
アップロード、ファイルの 24, 88
暗号化
 データベースのクライアントのパスワード 62
暗号化、ネットワークの 74
一時的 LOB の管理 176
インストール・ディレクトリー構成変数
 管理ツールでの構成 68
 構成、初期設定ファイル内の 19
隠蔽変数
 変数名を隠す 133
エラー条件、言語環境の 168
エラーのログ記録
 ログ記録のレベル
 指定 69
 変数 69
 ログ・ファイル
 位置の指定 19
 位置変数 19
 レベル変数 19
 DTW_LOG_DIR 19
 DTW_LOG_LEVEL 69

エラー・ログの収集
 計画 256, 259
 説明 255, 258
 パフォーマンスの考慮 251
 ログ収集のレベル
 起動属性 259
 指定 257, 259
 パフォーマンスへの影響 251
 変数 257
 ログ・ファイル
 活動化 256, 259
 サイズ 255, 258
 フォーマット 257, 261
 DTW_LOG_DIR 256
 DTW_LOG_LEVEL 257
 Live Connection のファイル名 260

[カ行]

開始、Net.Data の 83
隠し変数
 資産の保護 75
各種変数 136
型、変数の 129
環境変数 130
関数 177
 数学 149
 ストアド・プロシーチャーの呼び出し 177
 ストリング 149
 説明 139
 定義 139
 汎用 148
 表 150
 フラット・ファイル 150
 ユーザー定義 139
 呼び出し 145
 ワード 149
 FUNCTION ブロックの構文 139

関数 177 (続き)
 MACRO_FUNCTION ブロックの構文 140
 Web レジストリー 150
関数呼び出し
 組み込み 146
 構文 145
管理ツール
 データベースのクライアントのパスワードの暗号化、クライアント 62
 ENVIRONMENT ステートメント 63
 Java の実行時ライブラリーのインスツール 53
 Net.Data の構成
 概説 53
 クライアント 57
 構成変数ステートメント 68
 事前準備 53
 パス・ステートメント 54
 Live Connection ポート 56
起動、Net.Data の
 FastCGI 49
 HTML ブロック 151
 Web サーバー API の使用 96
機密保護
 アクセス権の指定 69, 168
 概説 71
 キャッシング 228
 許可 75
 言語環境 168
 データベースのクライアントのパスワードの暗号化 62
 認証 74
 ネットワーク暗号化 74
 ファイアウォール 71
 Net.Data のメカニズム 75
キャッシュ
 アクティブ化、現行の 234
 識別子 226, 227, 230

- キャッシュ (続き)
 - 指定、メモリーの 235
 - スタンザ、構成 234
 - 定義 225
 - パス 234
 - ページの経過時間の指定 236
 - ページのスペースの指定 235
- キャッシュ ID
 - 定義 226, 227
 - プランの作成 230
- キャッシュ管理プログラム
 - 開始 240
 - 構成ファイル 8, 226, 231
 - 構成変数 15
 - スタンザ、構成 231
 - 接続タイムアウト 232
 - 定義 226, 231
 - 定義、キャッシュの 234
 - 停止 241
 - ポート 231
 - ログ・ファイル
 - 活動化 232
 - キャッシュごと 238
 - トレース・フラグ 233
 - 命名 231
- キャッシュのトランザクション・ログのファイル 238
- キャッシング
 - インターフェース 228
 - 決定、構成の 226
 - サンプル・アプリケーション 224
 - 収集、統計の 245
 - 紹介 225
 - 照会、特定のキャッシュの 245
 - 制約事項 228
 - 停止 245
 - フラグ 245
 - フラッシュ 245
 - プランの作成 229
 - ページ 242
 - 用語 225
 - ログ 232, 245
 - cacheadm コマンド 245
- 共用ライブラリー
 - AIX での言語環境のロード 267
- 許可
 - 機密保護 75
 - Net.Data のファイルへのアクセス権の指定 69
- 空白文字、不要文字を削除するための変数 20
- クライエント
 - 管理ツールでの構成
 - 削除 61
 - 追加 58
 - データベース情報 61
 - データベースのパスワードの暗号化 62
 - 変更 59
 - DB2 データベースのログオン
 - のテスト 61
 - 実行可能ファイル名 40, 41
 - 説明 219
 - Java 言語環境 204
- グローバルな識別子の効力範囲 124
- 結果セット 180, 181, 182
 - 処理、ストアード・プロシージャ 180
 - 単一 181
 - 複数の 182
 - ガイドラインおよび制約事項 159
 - デフォルトのレポート 182
- 結果セットでの DataLink URL のコード化 184
- 結果セットの処理、ストアード・プロシージャ 180
- 言語環境 208, 212
 - エラー条件の処理 168
 - 管理ツールでの構成
 - 削除 67
 - 追加 64
 - 変更 65
 - 機密保護 168
 - 構成、初期設定ファイル内の 29
 - サポートされた 166
 - セットアップ 32
 - フラット・ファイル・インターフェース 189
 - 変数 138
 - 呼び出し 168
- 言語環境 208, 212 (続き)
 - 例 29
 - AIX での共用ライブラリーのロード 267
 - ENVIRONMENT ステートメントの構成 29, 63
 - Java アプリケーション 201
 - Java アプレット 193
 - ODBC 169
 - Oracle 170
 - Perl 205
 - REXX 208
 - SQL 171
 - System 212
 - Web レジストリー 190
- コード・ページ 268
- 向上、パフォーマンスの 217
- 構成、キャッシュ管理プログラム 231, 234
- 構成、Net.Data の
 - キャッシュ管理プログラム構成ファイル
 - スタンザ 231, 234
- 構成変数ステートメント
 - 構成
 - 管理ツールでの 68
 - 構成、初期設定ファイル内の 14
 - 説明 14
 - ホーム・ディレクトリー (inst_dir) 19
 - DB2INSTANCE 17
 - DTW_CACHE_HOST 15
 - DTW_CACHE_PORT 15
 - DTW_CM_PORT 18
 - DTW_DEFAULT_ERROR_MESSAGE 18
 - DTW_DIRECT_REQUEST 18
 - DTW_INST_DIR 19, 68
 - DTW_LOG_DIR 19
 - DTW_LOG_LEVEL 19, 69
 - DTW_MBMODE 20
 - DTW_REMOVE_WS 20
 - DTW_SHOWSQL 21
 - DTW_SMTP_SERVER 21
 - DTW_UNICODE 22
 - DTW_VARIABLE_SCOPE 23

効力範囲、識別子の
グローバルな 124
マクロ 124

FUNCTION ブロック 124

REPORT ブロック 125

ROW ブロック 125

コマンドの実行 212

[サ行]

サブレット

実行 101

説明 100

API 文書 100

NetObjects Fusion プラグイン
277

Net.Data

関数 101

プラグインでのプロパティ
の変更 282

プラグインのセットアップ
278

マクロ 100

Net.Data のための構成 49

NOF プラグイン 277

NOF プラグインで表示 282

参照、変数の 128

サンプル・マクロ 282

識別子の効力範囲 124

実行、SQL ステートメントの 169,
170, 171

実行可能な変数 131

使用、Web サーバー API の

Net.Data の起動 96

使用可能にする、直接要求を

(DTW_DIRECT_REQUEST) 18

条件付き

変数 130

論理、IF ブロック 160

初期設定ファイル

更新 13

構成変数ステートメント 14

サンプル 11

説明 7

パス・ステートメント 23

フォーマット 14

初期設定ファイル (続き)

ENVIRONMENT ステートメント
29

数学関数 149

スタンザ

キャッシュ、構成 234

キャッシュ管理プログラム、構成
231

ストアード・プロシージャ 177,
178, 179, 180, 181, 182, 183

結果セットの処理 180

ステップ 179

単一の結果セット 181

デフォルトのレポート 181, 182

パラメーターを渡す 180

複数の結果セット 182

マクロからの呼び出し 177

有効なデータ型 178

Net.Data テーブル 182, 183

REPORT ブロック 181, 183

ストリング関数 149

生成、Java アプレットの 193

接続管理

構成 38

パフォーマンス 218

接続管理プログラム

開始

メッセージ・オプション付き
222

AIX 221

OS/2 および Windows

NT 221

説明 219

Live Connection のログ記録の活
動化 259

接続タイムアウト、キャッシュ管理
プログラムの 232

宣言部分、マクロ構造の 111

[タ行]

直接要求

キャッシングの制約事項 228

構文 90

説明 83

例 95

データ型 173, 178, 184

データ型 173, 178, 184 (続き)

ストアード・プロシージャの
178

DATALINK 184

LOB 173

データ言語環境 169

データ出力のフォーマット 153

データベース

クライエット、構成 57

定義、変数の

照会ストリング・データ 127

DEFINE ステートメントまたはプ
ロック 125

HTML フォーム

SELECT、INPUT、および

TEXTAREA タグ 127

デフォルトのレポート 181, 182

ストアード・プロシージャに指
定 181, 182

プリント 154

トークンのサイズ 123

動的な生成、変数名の 128

トレース・フラグ、キャッシュ管理
プログラムの 233

[ナ行]

ナビゲーション、マクロ内およびマ
クロ間の 117

認証、機密保護の 74

ネイティブ言語サポート、関数のた
めの 20

[ハ行]

パスワードおよびログイン、クライ
エットの構成の 41

パス・ステートメント

管理ツールでの構成

削除 56

追加 55

変更 55

更新のガイドライン 24

構成、初期設定ファイル内の 23

資産の保護 75

DTW_UPLOAD_DIR 24

EXEC_PATH 25

パス・ステートメント (続き)

FFI_PATH 26
HTML_PATH 26
INCLUDE_PATH 27
MACRO_PATH 28

パフォーマンス 211

エラー・ログの収集 251
最適化、言語環境の 251
システム言語環境 254
cache query all 247
FastCGI 218
Live Connection 218
Perl 言語環境 254
REXX 環境 211, 268
REXX 言語環境 251
SQL 言語環境 252
Web サーバー API 217

パラメーターを渡す 180, 209, 213

ストアド・プロシージャ
180
REXX プログラム 209
System 言語環境 213

汎用関数 148

表関数 150

表示、NOF プラグインでサブプレッ
トを 282

表処理変数 136

表変数 135

ファイアウォール 71

ファイル 88

アップロード 24, 88
Net.Data へのアクセス権の指定
69

フォーム 86, 87, 88

FILE 入力タイプの使用 88
Net.Data の起動 86, 95
Net.Data を起動するための Web
ページ内の 87

複数のレポート・ブロック 156

フッター情報、REPORT ブロック
154

部分、マクロの

宣言 111
表示 111

プラグイン、NetObjects Fusion 277

フラット・ファイル関数 150

フラット・ファイル・インターフェ
ース言語環境

概説 189

フラット・ファイル・データ・ソー
ス 189

ブランク、不要文字を削除するた
めの変数 20

プリント、デフォルトのレポートで
は使用禁止の 154

ブロック、マクロの 114

ヘッダー情報、REPORT ブロック
154

変数

隠し 133
各種 136
型 123, 129

環境 130

言語環境 138

構成、ステートメントの

インストール・ディレクトリ
ー (DTW_INST_DIR) 19,
68

エラーのログ・レベル
(DTW_LOG_LEVEL) 19, 69

エラー・ログの位置
(DTW_LOG_DIR) 19

管理ツール 68

キャッシュ管理プログラムの
ポート

(DTW_CACHE_PORT) 15

キャッシュ・マシン名
(DTW_CACHE_HOST) 15

削除、不要なブランクの
(DTW_REMOVE_WS) 20

初期設定ファイル 14

説明 14

直接要求を使用可能にする
(DTW_DIRECT_REQUEST) 18

デフォルトのエラー・メッセ
ージを使用可能にする

(DTW_DEFAULT_ERROR_
MESSAGE) 18

電子メール SMTP サーバー
(DTW_SMTP_SERVER) 21

ネイティブ言語サポート

(DTW_MBMODE) 20

変数 (続き)

構成、ステートメントの (続き)

編集マスク

(DTW_CM_PORT) 18

変数の効力範囲変数

(DTW_VARIABLE_SCOPE) 23

ホーム・ディレクトリー 19,
68

DB2 のインスタンス

(DB2INSTANCE) 17

SHOWSQL を使用可能にする
(DTW_SHOWSQL) 21

SMTP サーバー

(DTW_SMTP_SERVER) 21

UNICODE 変数

(DTW_UNICODE) 22

効力範囲 124

参照 128

実行可能 131

条件付き 130

説明 123

定義 125

トークンのサイズ 123

動的に生成された参照 128

名前の動的な生成 128

表 135

表の処理 136

リスト 134

レポート 137

ポート

キャッシュ管理プログラム 15,
231

Live Connection

管理ツールでの構成 56

構成ファイル 39

ホーム・ディレクトリー

管理ツールでの構成 68

構成、初期設定ファイル内の 19,
53

保護、資産の 71

[マ行]

マクロ

開発 111

関数 139

サンプル 10, 113

マクロ (続き)

- 識別子の効力範囲 124
- 条件付き論理 160
- 説明 1
- 宣言部分 111
- 内部およびその間のナビゲーション 117
- 表示部分 111
- ブロック 114
- 分析 112
- 変数 123
- ループ 163
- DEFINE ブロック 114
- FUNCTION ブロック 115
- HTML の生成 151
- HTML ブロック 116
- IF ブロック 160
- NOF プラグイン 277
- WHILE ブロック 163

マクロ要求 85

- 構文 85
- 説明 83
- 例 85

文字セット 20, 22

[ヤ行]

ユーザー定義関数 139

用語集 291

呼び出し 177, 179, 208, 209, 212

- 関数 145
- 言語環境 168
- ストアド・プロシージャ 177, 179
- プログラム、System 212
- FFI 組み込み関数 189
- Java アプリケーション 201, 202
- Perl スクリプト 205
- REXX プログラム 208, 209
- Web レジストリー組み込み関数 192

呼び出し、アプレットの 194

[ラ行]

ラージ・オブジェクト (LOB) 173, 174, 175, 176

ラージ・オブジェクト (LOB) 173, 174, 175, 176 (続き)

- 一時的、管理 176
- サポートされているデータ型 174
- 説明 173
- 有効な形式 175

リスト変数 134

リレーショナル・データベース言語環境 169

リンク 86, 87

- Net.Data の起動 86, 95
- Net.Data を起動するための Web ページ内の 87

ループ、WHILE ブロックの 163

レジストリー 190

レポート

- デフォルト 156
- 1 つの関数呼び出しによる複数生成 156

レポートのフォーマットのカスタマイズ 155

レポート変数 137

ログインおよびパスワード、クライアントの構成の 41

ログ・ファイル

- 活動化 19, 256, 259
- キャッシュ管理プログラム 231, 232, 233
- キャッシュごと 238
- 最大サイズ 255, 257, 258, 261
- フォーマット 257, 261
- レベルの制御 256, 259
- ローテーション 257, 261
- Live Connection、名前 260

ロケール 268

[ワ行]

ワード関数 149

渡す、パラメーターを

- Perl スクリプト 205

[数字]

2 バイト文字セット 269

A

AIX、付録: Net.Data for 267

Apache Web サーバーのインストール 45

B

Beans

- Net.Data のための構成 49

BLOB 173

C

cacheadm

- 構文 245
- 停止、キャッシュ管理プログラム 241

CLOB 173

COMMIT 172

D

DATALINK データ型 184, 185

- DataLink ファイル・マネージャー 184
- URL のコード化 185

DB2INSTANCE 17

DBCLOB 173

DBCS 269

DEFINE ブロック

- 説明 114
- 変数の定義 125

Domino Go Webserver のインストール 45

dtwclean デーモン、一時的 LOB の管理 176

dtwcm コマンド 221

DTW_APPLET 193

DTW_CACHE_HOST 15

DTW_CACHE_PAGE 242

DTW_CACHE_PORT 15

DTW_CM_PORT 18

DTW_DEFAULT_ERROR_MESSAGE 18

DTW_DEFAULT_REPORT 156

DTW_DIRECT_REQUEST 18

DTW_FFI 189

DTW_INST_DIR 19, 68
DTW_JAVAPPS 201
DTW_LOG_DIR 19
DTW_LOG_LEVEL 19, 69, 251, 257
DTW_MBMODE 20, 269
DTW_ODBC 169
DTW_ORA 170
DTW_PERL 205
DTW_REMOVE_WS 20
DTW_REXX 208
DTW_SHOWSQL 21
DTW_SMTP_SERVER 21
DTW_SQL 171
DTW_SYSTEM 212
DTW_UNICODE 22, 269
DTW_UPLOAD_DIR 24, 88
DTW_VARIABLE_SCOPE 23
DTW_WEBREG 190

E

ENVIRONMENT ステートメント
 クライエント名 31
 言語環境のタイプ 30
 構成、初期設定ファイル内の 29, 30
 構文 30
 説明 29, 63
 パラメーター・リスト 31
 例 32
 DLL あるいはライブラリー名 30
EXEC_PATH 25, 54

F

FastCGI
 サポートされる言語環境 45, 218
 同時処理の決定 218
 パフォーマンスの考慮 218
 Net.Data の構成 45
 Net.Data のための構成
 Apache Web サーバーのイン
 ストール 45
 Domino Go Webserver のイン
 ストール 45
FFI 言語環境
 組み込み関数の呼び出し 189

FFI_PATH 26, 54
FUNCTION ブロック
 関数の呼び出し 145
 識別子の効力範囲 124
 出力のフォーマット 153
 説明 115
FunctionServlet
 実行 103
 説明 101
 NOF プラグイン 277

G

GWAPI
 Net.Data の起動 97
 Net.Data のための構成 50

H

HTML 86, 87
 認識されていないデータ 152
 表のタグ 154
 フォーム 86, 87
 について 87
 変数を定義する
 SELECT、INPUT、および
 TEXTAREA タグ 127
 Net.Data の起動 86, 95
 ブロック
 処理 152
 説明 116
 例 151
 Net.Data の起動 151
 マクロ内に生成 151
 リンク 86, 87
 について 87
 Net.Data の起動 86, 95
 FORM の処理依頼ボタン 152
 URL、Net.Data の起動 96
HTML_PATH 26, 54

I

IF ブロック 160
IMS Web 言語環境
 セットアップ 33
INCLUDE_PATH 27, 54

inst_dir 53
ISAPI
 Net.Data の起動 97
 Net.Data のための構成 50

J

Java Beans
 Net.Data のための構成 49
Java アプリケーション言語環境
 概説 201
 セットアップ 34
Java アプレット
 起動 194
 クラス 200
 作成 194
 タグの生成 194
Java アプレット言語環境
 言語環境 193
Java 言語環境
 関数の作成 203
 関数の呼び出し 202
 起動 204
 クライエントの作成 204
 ファイル構造 203
Java サーブレット
 Net.Data のための構成 49
Java のクライエント、構成 41

L

Live Connection
 クライエント
 管理ツールでの構成 57
 構成ファイル 9
構成ファイル
 更新 38
 サンプル 12
 説明 8
 データベースのクライエント
 40
 データベース名 41
 名前 39
 フォーマット 38
 プロセスの数 40, 42
 プロセスのタイプ 40

Live Connection (続き)
 ログインおよびパスワード
 41
 Java のクライアント 41
 使用するかどうかを決定する
 221
 接続管理プログラムの開始 221
 パフォーマンスの向上 218
 プロセスの流れ 222
 ポート
 管理ツールでの構成 56
 構成、初期設定ファイル内の
 39
 利点 220
Live Connection のログ記録
 活動化 259
 計画 259
 説明 258
 ファイル名 260
 レベルの制御 259
 ログ収集のレベル
 起動属性 259
 指定 259
 ログ・ファイル
 サイズ 258
 フォーマット 261
LOB 173
Lotus Domino Go Webserver
 (GWAPI)
 Net.Data のための構成 50

M

MacroServlet
 実行 101
 説明 100
 NOF プラグイン 277
MACRO_FUNCTION ブロック
 関数の呼び出し 145
 構文 140
MACRO_PATH 28, 54
MAX_PROCESS 40, 42, 60
MBCS サポート、関数のための 20
MESSAGE ブロック
 構文 143
 効力範囲 143

MESSAGE ブロック (続き)
 処理 143
 説明 143
 例 144
MIN_PROCESS 40, 42, 60

N

NetObjects Fusion (NOF) プラグイン
 インストール 278
 サブレットのプロパティの変更 282
 セットアップ 278
 説明 277
 ハードウェアおよびソフトウェア
 要件 278
 表示 282
 マクロおよび関数サブレット用
 277
Net.Data
 概説 1
 起動 83
 機密保護のメカニズム 75
 構成 5
 ファイル、アクセス権 69
 マクロ、開発 111
Net.Data サブレット
 FunctionServlet 101
 MacroServlet 100
 NOF プラグイン
 サブレットの表示 282
 セットアップ 278
 説明 277
 プロパティの変更 282
Net.Data の起動 85, 86
 概説 83
 構文 85
 直接要求 83
 フォーム 86, 95
 マクロによる 85
 マクロ要求 83
 マクロを使用しない 90
 リンク 86, 95
 CGI の使用 83
 GWAPI 97
 ISAPI 97
 NSAPI 98

Net.Data の起動 85, 86 (続き)
 URL 86, 96
Net.Data の構成
 概説 5
 管理ツール
 概説 53
 クライアント 57
 事前準備 53
 パス・ステートメント 54
 変数ステートメントの構成
 68
 ポート 56
 ENVIRONMENT ステートメン
 ト 63
 Java の実行時ライブラリーの
 インストール 53
 キャッシュ管理プログラムの構成
 ファイル
 説明 8
 ポート 15
 言語環境のセットアップ 32
 手動による方法と管理ツールによ
 る方法 5
 初期設定ファイル
 更新 13
 構成変数ステートメント 14
 説明 7
 パス・ステートメント 23
 ENVIRONMENT ステートメン
 ト 29
 制御ファイル比較 9
 方法の比較 6
FastCGI 45
Java Beans と一緒に使用するため
 の 49
Java サブレットと一緒に使用す
 るための 49
Live Connection の構成ファイル
 39
 更新 38
 説明 8
Net.Data のファイルへのアクセス
 権 69
Web サーバー API と一緒に使用
 するための 49

Net.Data 表、ストアード・プロシ
ジャー 182, 183
Net.Data マクロ。マクロを参照。 1
NOF (NetObjects Fusion) プラグイン
277
NSAPI
Net.Data の起動 98
Net.Data のための構成 51

O

ODBC 言語環境
概説 169
制約事項 170
変数 170
Oracle 言語環境
概説 170
制約事項 170
セットアップ 35

P

Perl 言語環境
概説 205
組み込み関数の呼び出し 205
パラメーターを渡す 205
REPORT および MESSAGE ブロ
ック 207

R

REPORT および MESSAGE ブロッ
ク
Perl スクリプト 207
REPORT ブロック 181, 183
効力範囲 125
ストアード・プロシジャー
181, 183
制約事項 159
説明 153
データ出力のフォーマット 153
デフォルトのレポート 156
複数の 156
複数のためのガイドライン 159
ヘッダーおよびフッター情報
154

REPORT ブロック 181, 183 (続き)
例 156
RETURN_CODE 変数 143, 168
REXX 言語環境 208, 209, 211
概説 208
パラメーターを渡す 209
プログラムの呼び出し 209
AIX 上でのパフォーマンス 211
REXX、パフォーマンスの向上 268
ROW ブロック、識別子の効力範囲の
125

S

SQL 言語環境
概説 171
制約事項 171
変数 171
SQLCODE 168
System 言語環境 212, 213
概説 212
コマンドの発行 212
パラメーターを渡す 213
プログラムの呼び出し 212

T

TRANSACTION_SCOPE 172

U

UNICODE 269
UNICODE 変数
DTW_MBMODE による 20, 22
URL 86
変数の定義 127
Net.Data の起動 86, 96
UTF-8 269

W

Web サーバー
FastCGI のための構成 45
Web サーバー API のための構成
49
Web サーバー API
考慮事項 97

Web サーバー API (続き)
説明 96
パフォーマンスの向上 217
パフォーマンスの考慮事項 217
Net.Data の起動
GWAPI 97
ISAPI 97
NSAPI 98
Net.Data のための構成
説明 49
GWAPI 50
ISAPI 50
NSAPI 51
Web ページのキャッシング 242
Web レジストリー関数 150
Web レジストリー言語環境
概説 190
組み込み関数の呼び出し 192
WHILE ブロック 163

IBM と連絡をとる

技術上の問題がある場合は、時間をとって問題判別の手引き に定義されている処置を検討し、それらの提案を実行した後で、DB2 顧客サービスに連絡をとってください。この資料には、DB2 顧客サービスがお客さまを支援するために必要とする情報が説明されています。

製品情報

以下の情報は英語で提供されます。内容は英語版製品に関する情報です。

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

DB2 World Wide Web ページには、ニュース、製品説明、研修スケジュールなどの DB2 に関する最新情報が提供されています。ただし、提供されている情報は英語です。

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

「DB2 Product and Service Technical Library」では、よくされる質問 (FAQ)、修正内容、資料、および最新の DB2 技術情報などの情報へのアクセスが提供されています。

注: この情報のご提供は英語のみとなりますのでご注意ください。

<http://www.elink.ibm.com/pbl/pbl/>

「International Publications」注文用 Web サイトでは、マニュアルの注文方法についての情報を提供しています。ただし、提供されている情報は英語です。

<http://www.ibm.com/education/certify/>

IBM の「Professional Certification Program」Web サイトでは、DB2 を含むさまざまな IBM 製品の認証テストの情報を提供しています。ただし、提供されている情報は英語です。

<ftp.software.ibm.com>

匿名でログインしてください。ディレクトリー /ps/products/db2 には、DB2 および多数の他製品に関連したデモ、修正プログラム、情報、およびツールがあります。ただし、提供されている情報は英語です。

comp.databases.ibm-db2、bit.listserv.db2-l

これらのインターネット・ニュースグループは、ユーザーが DB2 製品に関する自分の経験について話し合うために利用できます。ただし、提供されている情報は英語です。

Compuserve: GO IBMDB2

このコマンドを入力すると、IBM DB2 Family forum にアクセスできます。すべての DB2 製品が、このフォーラムでサポートされています。ただし、提供されている情報は英語です。

米国以外の国で IBM に連絡する方法については、*IBM Software Support Handbook* の Appendix A を参照してください。この資料にアクセスするには、Web ページ <http://www.ibm.com/support/> にアクセスし、ページの最下部にある「IBM Software Support Handbook」リンク・ボタンを選択します。

注: 国によっては、IBM が承認している販売業者が、IBM サポート・センターの代わりにそれら販売業者のサポート・センターに連絡する場合があります。



Printed in Japan

SB88-7412-01



日本アイ・ビー・エム株式会社

〒106-8711 東京都港区六本木3-2-12