Net.Data
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

SB88-7371-00

Net.Data
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

SB88-7371-00

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、 171ページの『付録E. 特記事項』に記載する一般情報をお読みください。

原　典：　VNDT-2ADM-00 (この番号でマニュアルのオーダーはできません。)
Net.Data
Administration and Programming Guide
for OS/2, Windows NT, and UNIX

発　行：　日本アイ・ビー・エム株式会社

担　当：　ナショナル・ランゲージ・サポート

第1刷 1998.9

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

目次

まえがき

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

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

Net.Data について

IBM の Net.Data プロダクトを用いると、DB2、IMS、および ODBC 対応のデータベースを含む関係データベース管理システムおよび非関係データベース管理システム (DBMS) の両方、ならびに Java、JavaScript、Perl、C、C++、および REXX などプログラミング言語で作成されたアプリケーションを使用して、動的 Web ページを作成することができます。

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

要求は URL (uniform resource locator) の形式で、 Netscape Navigator または Internet Explorer などのブラウザーから Web サーバーに流れ、 Web サーバーはその要求を Net.Data に送って実行します。 Net.Data はマクロを検索、実行し、ユーザーが作成した関数に基づいてカスタマイズ Web ページを生成します。これらの関数は以下のことを行うことができます。

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

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

Net.Data は、HTTP (HyperText Transfer Protocol) およびコモン・ゲートウェイ・インターフェース (CGI) などの業界標準のインターフェースをサポートしています。HTTP はブラウザーと Web サーバー間で使用され、CGI は Web サーバーと Net.Data 間で使用されます。このようなサポートによって、 Net.Data で使用するブラウザーまたは Web サーバーは、ユーザーが好きなものを選択することができます。製品の Net.Data ファミリーは、OS/400、OS/390、Windows NT、AIX、OS/2、 HP-UX、Sun Solaris、および SCO オペレーティング・システムと同様の機能を提供します。Net.Data は、複数のオペレーティング・システム上で、 FastCGI および主な Web サーバー API もサポートします。

さらに、Net.Data バージョン 2 では、この他にもパフォーマンスが向上し、以下のようなユーザー・アプリケーションの要件を満たしています。

- HTML ページのキャッシング
- マクロ・ファイルを使用しない Net.Data の呼び出し (直接要求)
- 生成された Web ページ内の、無駄になっている空白スペースの最小化
- 最適化された Math 関数

Net.Data バージョン 2 では、以下のようなたくさんの機能も拡張されています。

- 言語環境機能強化には、SQL またはODBC 言語環境を介して、ストアード・プロシージャーから 1 つまたは複数の結果セットを検索する機能が組み込まれています。
- マクロ言語環境は、次のものを含んでいます。
  - 注釈をどこでも追加する機能
  - ネストされた IF ブロック
  - WHILE ブロック
  - ストアード・プロシージャーから単一または複数の　結果セットを受信する機能
  - DBCS 対応のストリングおよびワード関数
  - ストアード・プロシージャー用パラメーター・リスト内の SQL 10 進数データ・タイプのサポート
  - cookie のサポート
  - 動的に生成された 電子メール・メッセージのサポート

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 コネクション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。

データベースからのデータのアクセスを容易にするために、 Net.Data には NetObjects Fusion プラグインや Java ベースの開発のためのスマートガイドなどの、さまざまなツールがあります。これらのツールは Java 環境では Net.Data Java サー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関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 プラグインによって、ユーザーは、NetObjects Fusion Web 開発ツールを使用して、関係データ・ソースからの動的なデータで、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 smartguide は図形処理ツールを提供し、ユーザーが基本 Net.Data マクロを作成するガイドをします。

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 サーブレットを使用したマクロの開発方法を学習します。ユーザーは、Net.Data が提供する言語環境の使用方法を学習します。この言語環境を利用して、DB2

データベースや IMS トランザクションを IMS Web を使用してアクセスしたり、Java、REXX、Perl、および他のプログラム言語を使用してユーザー・データをアクセスします。

本書では、告知後、まだ発売されていない製品または機能について言及する場合があります。

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

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

本書の対象読者

本書は、Net.Data アプリケーションを設計し、開発する方々を対象としています。オペレーティング・システムの差、Net.Data メッセージ、およびその他の情報は、*Net.Data 解説書* で解説しています。

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

本書の例について

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

第1章 Net.Data とは何か ？

Web ページは、HTML だけで作成することができます。したがって、ユーザーがこれらのページを編集しない限り、ページは変更されません。 Web 上に “live” データとアプリケーション (現在の営業統計など) を組み込むには、 Web サイトの開発者は通常、プログラムを作成します。このプログラムは、 Web サーバーのミドルウェアとして Web ページを動的に構築します。この種のプログラムの作成は簡単ではありません。

Net.Data を使用すると、マクロ を使用して、対話式 Web アプリケーションを簡単に作成することができます。 Net.Data マクロによって、論理、変数、関数呼び出し、およびレポート生成ツールを使用することができます。マクロとは Net.Data マクロ言語の構成要素、HTML タグ、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・ファイルを処理して HTML 出力を生成します。マクロは、単純な HTML と、Web サーバー・プログラムの動的な機能性を結合します。この結果、live データを静的 Web ページに簡単に追加することができるようになります。 live データは、ローカルのデータベースやリモートのデータベースから、さらにフラット・ファイルから抽出することができます。アプリケーションおよびシステム・サービスから生成することもできます。

1ページの図 1 は、 Net.Data および Web サーバーの関係、サポートされるデータとプログラム言語環境への環境を示しています。

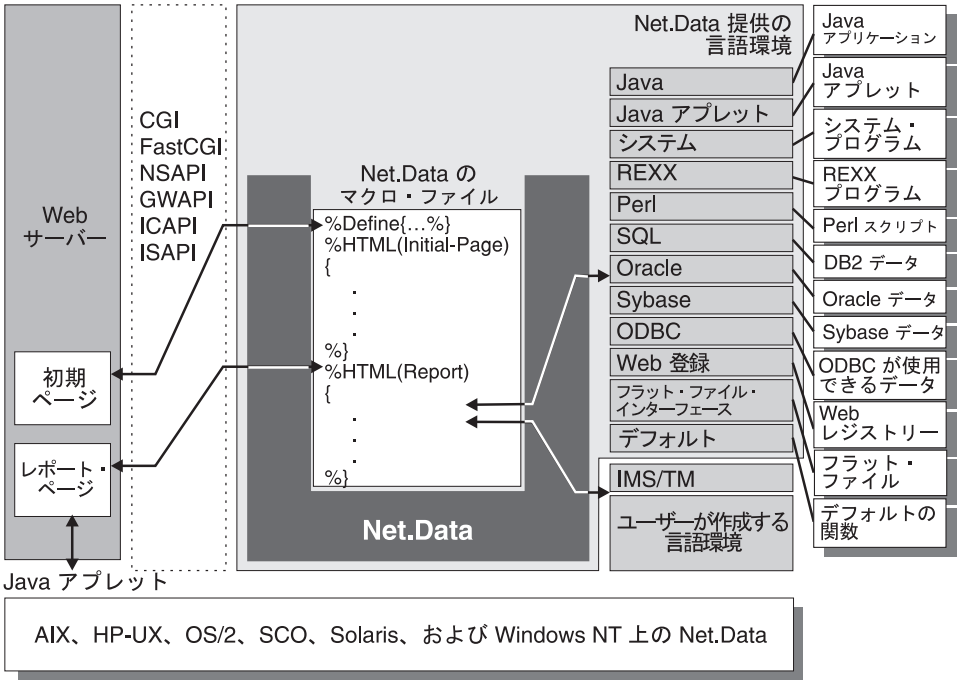


図 1. Net.Data、Web サーバーおよびサポートされるデータとプログラム・ソース間の関係

Web サーバーは、Net.Data サービスを要求する URL を受信するときに、 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って、 CGI またはFastCGI

プロセスを使用して、あるいは、 Web サーバー・アプリケーション・プログラミング・インターフェース (API) スレッドを 、 Net.Data を起動します。 URL には Net.Data 特有の情報が組み込まれています。これらの情報は、マクロ・ファイル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 は要求の処理が終了すると、結果の Web ページを Web サーバーに送信します。サーバーはそのページを Web クライアントに渡し、そのクライアントでブラウザーを使って表示します。

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

このほかに Net.Data の重要な利点としては、多数の異なるデータベース・ソースへのアクセスをサポートしていることです。これによって、Web 開発者は、DB2、IMS、Oracle、Sybase、および任意の ODBC を使用できるデータ・ソースなどのさまざまなデータベースからのデータを使用することができます。 Net.Data が提供する言語環境に関するさらに詳細な情報は、*Net.Data 言語環境解説書* を参照してください。

Net.Data Web アプリケーション環境には、以下の機能があります。

インタープリター・マクロ言語

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

直接要求

単一の SQL ステートメント、DB2 ストアード・プロシージャー、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、ブラウザーから Web サーバーまでの流れを制御します。

フリー・フォーマット

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

型を持たない変数

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

については、71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

製品に添付した README ファイルの命令を使用して、ご使用のオペレーティング・システムに対応した Net.Data をインストールすることができます。ほとんどの構成ステップは、インストール中に完了されますが、これは、オペレーティング・システムによって異なります。

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 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。構成の各ステップには、以下が組み込まれています。

- Net.Data の初期設定 (INI) ファイルのカスタマイズ
- FastCGI あるいはサポート Web サーバー API (オプション) との併用のための Net.Data の構成
- Web サーバーの構成および環境変数ファイルのカスタマイズ
- キャッシュ・マネージャーの構成 (任意選択)
- Live コネクション の構成 (任意選択)
- 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・システムの INI ファイルおよび Live コネクションの構成ファイルを編集します。テキスト・エディターをしようして、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、INI ファイルおよび Live コネクション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、5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	プラットフォーム			
	AIX	**NT**	**OS/2**	**HP|SUN|SCO**
Net.Data の INI ファイルの構成	A	A	A	M
クライエット・ポートの定義	A	A	A	M
クライエットの定義	A	A	A	M
クライエッ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

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。 (続き)*

タスク	プラットフォーム			
	AIX	**NT**	**OS/2**	**HP|SUN|SCO**
FastCGI、CGI、および APIs* 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。

- 12ページの『Net.Data の初期設定ファイルのカスタマイズ』
- 28ページの『FastCGI のための Net.Data の構成』
- 32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』
- 35ページの『Net.Data の管理ツールを用いた Net.Data の構成』
- 48ページの『Net.Data のファイルへのアクセス権の指定』

Net.Data の初期設定ファイルについて

Net.Data は、その初期設定ファイルを使用して、さまざまな構成変数の設定を確立し、言語環境と検索パスを構成します。構成変数の設定は、Net.Data 操作のさまざまな側面を制御します。たとえば、Web ページ内の文字データの符号化、ストリングおよびワード関数は、 DBCS が使用可能かどうか、およびデータベースのデータにアクセスするための DB2 インスタンス名の選択、などです。 この設定はまた、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間の接続および通信方法と、エラー・ログを起動するかどうかを指定します。

言語環境のステートメントは、使用可能な Net.Data の言語環境を定義し、言語環境に入出力する特殊な入力および出力パラメーター値を識別します。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 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 サーバーが、必ずこのファイルへのアクセス権限を持つようにしてください。詳しくは、48ページの『Net.Data のファイルへのアクセス権の指定』 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。

7ページの『Live コネクション構成ファイル』

7ページの『キャッシュ管理プログラムの構成ファイル』

8ページの『Net.Data の初期設定、制御、およびマクロ・ファイルのコモン・セクション』

Live コネクション構成ファイル

Live コネクションは、Windows NT、OS/2、および UNIX のオペレーティング・システムでの接続管理を提供し、開始オーバーヘッドを除去してパフォーマンスを改善します。 Net.Data Live コネクション構成ファイルには、 1 つまたは複数の名前付クライエットに関する情報が含まれています。クライエットとは、データベースへの接続を保守する長時間実行されるプロセス、または、複数のユーザーからの Net.Data マクロ起動を持続する Java アプリケーションです。クライエットの開始後、 Net.Data Live コネクションが終了するまで存続します。複数のクライエットが、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エット情報のパーツとして、クライエット名、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・クライエットの場合は、各クライエット記入項目のデータベース名、ログイン、およびパスワードも指定することができます。 AIX では、パスワードを暗号化することができます。

許可のためのヒント：コネクション管理プログラムを開始するユーザー ID が、確実にこのファイルにアクセスする権限を持つようにしてください。詳しくは、48ページの『Net.Data のファイルへのアクセス権の指定』 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、 119ページの『第 9章 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、 125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、一連の節または以下のスタンザです。

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、ログ状況、および状態の追跡を組み込んでいます。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。構成ファイル内にある 1 つの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存在します。このセクションには、ネットワーク

情報、メモリーとスペースの要件、ログ状況、および統計状況が含まれています。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、 119ページの『第9章 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、確実にこのファイルにアクセスする権限を持つようにしてください。詳しくは、48ページの『Net.Data のファイルへのアクセス権の指定』 を参照してください。

Net.Data の初期設定、制御、およびマクロ・ファイルのコモン・セクション

Net.Data の初期設定、構成、およびマクロ・ファイルの一定の部分は、全体として作業する Net.Data のすべてのコンポーネントとして一致していなければなりません。以下の表では、一致しなければならないこれらの各ファイルのエリアについて簡単に説明します。

表 2. Net.Data の構成ファイルおよびマクロ・ファイルの整合性の要件

ファイル	コモン・セクション	ノーツ
Net.Data INI ファイル	環境ステートメント	Live コネクション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・クライエット名を指定しなければなりません。
	Live コネクション構成変数	Net.Data の Live コネクションを使用するときには、 Live コネクション・ポート DTW_CM_PORT を指定します。この可変値は、 Live コネクション構成ファイルの 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ときには、任意選択でポートと機械変数を組み込みます。キャッシュ管理プログラム・ファイルを使用する場合には、これらの値は、このファイルで使用する値と一致しなければなりません。
Live コネクション構成ファイル	クライエット定義	各クライエット定義は、INI ファイル内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と、任意選択でポートと機械変数を組み込むことができます。 INI ファイルを使用する場合には、これらの値は、このファイルで使用する値と一致しなければなりません。

以下のフラグメントは、マクロ・ファイル、初期設定ファイル、および Live コネクション構成ファイルの間のリレーションシップを示しています。 2 つの クライエットは、マクロ・ファイル (DTW_SQL:SAMPLE、 DTW_SQL:CELDIAL) が使用し、 SAMPLE と CELDIAL という 2 つの DB2 データベースにアクセスします。 Live コネクションの構成ファイルには、クライエッ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、クライエット名を参照します。 LOGIN 値と PASSWORD 値は、Live コネクション構成ファイルで指定されます。

9ページの図 2 では、データベースにアクセスするために使用するクライエットを定義する @DTW_ASSIGN ステートメントを含むマクロ・ファイルのフラグメントを示しています。

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



<|*********************************************>
<|** This is an HTML comment                    **>
<|** Process the CELDIAL database using         **>
<|** the cliette DTW_SQL:CELDIAL  **>
<|*********************************************>
@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 のマクロ・ファイル・フラグメント

DATABASE 構成変数は INI ファイル内の ENVIRONMENT ステートメントに置換され、クライエット名を生成します。これによって、同じマクロ・ファイル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。

10ページの図 3 では、 ENVIRONMENT ステートメントおよび関連するクライエット型を含む Net.Data の初期設定ファイルのフラグメントを示しています。初期設定ファイルには、クライエットごとに 1 つの ENVIRONMENT ステートメントがあります。各データベース・クライエット型の場合には、 ENVIRONMENT ステートメントがクライエット名を指定します。その名前は、クライエット型と変数参照 $(DATABASE) で構成され、実行時の解決されます。 Live コネクションを使用する各言語環境の ENVIRONMENT ステートメントには、クライエッ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

11ページの図 4では、 Live コネクション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:SAMPLE と DTW_SQL:CELDIAL のクライエット定義が含まれます。

```
CONNECTION_MANAGER{
      MAIN_PORT=7128
      ADMIN_PORT1=7131
      ADMIN_PORT2=7133
      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:SAMPLE{
      MIN_PROCESS=1
      MAX_PROCESS=3
      START_PRIVATE_PORT=7100
      START_PUBLIC_PORT=7300
      EXEC_NAME=dtwcdb2.exe
      DATABASE=SAMPLE
            LOGIN=USER1
      PASSWORD=HAMLET
      }

      CLIETTE  DTW_SQL:CELDIAL{
      MIN_PROCESS=1
MAX_PROCESS=5
      START_PRIVATE_PORT=7500
      START_PUBLIC_PORT=7700
      EXEC_NAME=dtwcdb2.exe
      DATABASE=CELDIAL
      LOGIN=USER2
      PASSWORD=OPHELIA
      }
```

図 4. Live コネクション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。

- 12ページの『構成変数ステートメント』
- 17ページの『パス構成ステートメントのカスタマイズ』
- 20ページの『環境構成ステートメント』

12ページの図 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。

```
1  DTW_CM_PORT 7128
2  DTW_INST_DIR c:¥db2www
3  DTW_LOG_DIR c:¥db2www¥logs
4  DB2INSTANCE DB2
5  MACRO_PATH c:¥DB2WWW¥Macro
6  HTML_PATH  c:¥www¥html
7  INCLUDE_PATH c:¥db2www¥Macro
8  EXEC_PATH c:¥db2www¥Macro
9  FFI_PATH c:¥pub¥ffi;pub¥ffi¥data
10 ENVIRONMENT (DTW_SQL)          [DLL path]   [Parameter list]
11 ENVIRONMENT (DTW_SYB)          [DLL path]   [Parameter list]
12 ENVIRONMENT (DTW_ORA)          [DLL path]   [Parameter list]
13 ENVIRONMENT (DTW_ODBC)         [DLL path]   [Parameter list]
14 ENVIRONMENT (DTW_DEFAULT)      [DLL path]   [Parameter list]
15 ENVIRONMENT (DTW_APPLET)       [DLL path]   [Parameter list]
16 ENVIRONMENT (DTW_REXX)         [DLL path]   [Parameter list]
17 ENVIRONMENT (DTW_PERL)         [DLL path]   [Parameter list]
18 ENVIRONMENT (DTW_SYSTEM)       [DLL path]   [Parameter list]
19 ENVIRONMENT (DTW_FILE)         [DLL path]   [Parameter list]
20 ENVIRONMENT (DTW_WEBREG)       [DLL path]   [Parameter list]
21 ENVIRONMENT (DTW_JAVAPPS)      [Parameter list]
22 ENVIRONMENT (DTW_DLDPB)        [Parameter list]
23 ENVIRONMENT (USR_TEST)         [Parameter list]
```

- 1 ～ 4 行は、構成変数を定義します。
- 5 ～ 9 行は、マクロを処理するために必要なファイルへのパスを定義します。
- 10 ～ 23 行は、使用可能な環境ステートメントを定義します。

図 5. Net.Data の初期設定ファイル

以下の節では、INI 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、言語環境、および使用しているデータベースによって異なります。また、そのほか、アプリケーションに固有な要因によっても異なります。

構成変数ステートメントを更新するには、アプリケーションが要求する構成変数を使って初期設定ファイルをカスタマイズします。構成変数は、以下の構文を持ちます。

NAME[=]*value-string*

等号は、オプションです。大括弧で示されます。

以下のサブセクションは、初期設定ファイルで使用することができる構成変数ステートメントについて説明しています。

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。

- 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。
- CACHE_MACHINE は、ローカルあるいはリモート・マシンの TCP/IP ホスト名を指定する。

13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・マシンの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。

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 および Windows NT マシン上でのみ実行されます。Net.Data のキャッシングについてお知りになりたい場合は、119ページの『第9章 Net.Data のキャッシング』 を参照してください。

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で指定されるポート番号と一致しなければなりません。その結果、 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す。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 :

```
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。

CACHE_MACHINE: 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 :

```
CACHE_MACHINE
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あるいはリモート・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・システム上の DB2 には、環境変数として定義される が必要です。Net.Data が、DB2INSTANCE が環境変数として定義されていないことを検出すると、 DB2 への接続を試行する前に、INI ファイルで検出される DB2INSTANCE の値に、 DB2INSTANCE 環境変数を設定します。

構文 :

```
DB2INSTANCE instance_name
```

DTW_CM_PORT: Live コネクションのポート番号変数

Net.Data が Live コネクション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 コネクションのために使用する固有のポート番号を指定します。

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

Net.Data の INI ファイルの DTW_INST_DIR 変数は、Net.Data 実行中に、幾つかのファイを配置するのに使用されます。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: エラー・ログの位置変数

エラー・ログを格納するディレクトリーを、DTW_LOG_DIR 構成変数で指定します。ログ記録が、マクロ・ファイルの DTW_LOG_LEVEL 変数で使用可能になっている場合は、ログ・ファイルは、DTW_LOG_DIR 変数のパス・ステートメントで指定されるディレクトリーに格納されます。デフォルトは、¥*inst_dir*¥logs¥netdata.logs です。Net.Data によるエラー・メッセージのログ記録と DTW_LOG_LEVEL 変数については、 151ページの『第11章 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。

要件 : Net.Data がファイルのログ記録を取るためには、DTW_LOG_DIR 変数を定義しなければなりません。定義されていない場合は、たとえマクロ・ファイルで DTW_LOG_LEVEL を、ERROR あるいは WARNING に設定していたとしても、ログ記録は発生しません。

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_MBMODE: ネイティブ言語サポート変数

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

構文 :

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

例 : 各国語サポートをアクティブにします。

```
DTW_MBMODE = YES
```

マクロ・ファイルの初期設定ファイルの設定を上書きするには、以下のようにします。

- DTW_MBMODE 変数を、初期設定ファイルの DEFAULT ENVIRONMENT ステートメントの IN パラメーターとして、以下の例に示すように追加します。

```
ENVIRONMENT (DTW_DEFAULT)
DTWFUNC.DLL
   (IN DTW_MBMODE, OUT RETURN_CODE )
```

- マクロ・ファイルでは、アプリケーションに必要な値に構成変数 DTW_MBMODE を設定します。

DTW_OPTIMIZE_MATH: 数学関数変数の最適化

数学関数のパフォーマンスを最適化します。 DTW_OPTIMIZE_MATH が、YES に設定されている場合は、Net.Data は、C の数学フォーマットを使用します。これにより関数はより高速に実行されます。しかし、出力フォーマットは、この変数がない場合とは異なります。

- 特に、小数点の後ろに続くゼロは、表示されません。
- 精度は常に有効数字の桁数を意味します。

DTW_OPTIMIZE_MATH が NO に設定されている場合は、Net.Data は、REXX の数学フォーマットを使用します。関数の速度は遅くなりますが、Net.Data の前バージョンにより生成された出力と一致した出力形式を提供します。デフォルト値は、NO です。

構文 :

```
DTW_OPTIMIZE_MATH NO|YES
```

マクロ・ファイルの初期設定ファイルの設定を上書きするには、以下のようにします。

マクロ・ファイルでは、DTW_ASSIGN を使用し、アプリケーションに必要な値に構成変数 DTW_OPTIMIZE_MATH を設定します。

DTW_SMTP_SERVER: 電子メール SMTP サーバー変数

SMTP サーバーを指定して、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。この変数を設定しない場合には、 Net.Data は SMTP サーバーとしてローカル・ホストを使用します。

構文 :

```
DTW_SMTP_SERVER server_name
```

ただし、*server_name* は、電子メール・メッセージを送信するために使用する SMTP サーバーのホスト名または IP アドレス。

例:

```
DTW_SMTP_SERVER = "myserver"
```

パス構成ステートメントのカスタマイズ

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

- MACRO_PATH
- EXEC_PATH
- INCLUDE_PATH
- FFI_PATH
- HTML_PATH

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

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

- 指定された各ディレクトリーは、セミコロン (;) で区切られる。
- スラッシュ (/) および円マーク (¥) は同じものとして扱われる。
- 各パス・ステートメントは、複数のパスを指定することができる。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・ステートメントの使用をお勧めします。

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

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

MACRO_PATH ステートメントは、以下のようになります。

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

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

MACRO_PATH

MACRO_PATH 構成ステートメントは、Net.Data が Net.Data のマクロ・ファイルを検索するディレクトリーを識別します。たとえば、以下の URL を指定すると、パスおよびファイル名 macro/sqlm.d2w をもつ Net.Data のマクロが要求されます。

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

構文 :

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

等号 (=) は、オプションで、大括弧で示されます。

Net.Data は、パス macro/sqlm.d2w を、 MACRO_PATH 構成ステートメントのパスに、 Net.Data がマクロ・ファイルを検出するまで、あるいは、すべてのパスを検索するまで、左から右に追加します。 Net.Data のマクロの起動に関する情報については、55ページの『第4章 Net.Data を起動する』 を参照してください。

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

Net.Data の初期設定ファイル :

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

HTML リンク

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

ファイル *query.d2w* がディレクトリー /u/SYSADM/macros で検出される場合には、完全修飾パスは、/u/SYSADM/macros/query.d2w になります。

EXEC_PATH

EXEC_PATH 構成ステートメントは、 EXEC ステートメントまたは実行可能変数により起動される外部プログラムを、 Net.Data が検索する 1 つ以上のディレクトリーを識別します。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・ファイルの EXEC ステートメントを示しています。

Net.Data の初期設定ファイル :

```
EXEC_PATH = /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る場合には、プログラムの修飾名は /usr/lpp/netdata/prgms/myFunction.cmd になります。

INCLUDE_PATH

INCLUDE_PATH 構成ステートメントは、Net.Data が検索する 1 つ以上のディレクトリーを、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;...;path<i>n</i>
```

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

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

Net.Data の初期設定ファイル :

```
INCLUDE_PATH = /u/SYSADM/includes;/usr/lpp/netdata/includes;
```

Net.Data のマクロ

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

ファイル *myInclude.txt* が /u/SYSADM/includes ディレクトリーで検出される場合には、組み込みファイルの完全修飾名は、/u/SYSADM/includes/myInclude.txt になります。

例 2: 以下の例は、 INCLUDE_PATH ステートメントおよび サブディレクトリー名で完全修飾された INCLUDE ファイルを示しています。

Net.Data の初期設定ファイル :

```
INCLUDE_PATH = /u/SYSADM/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/SYSADM/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。

FFI_PATH

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

構文 :

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

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

Net.Data の初期設定ファイル :

```
FFI_PATH = /u/SYSADM/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。

HTML_PATH

HTML_PATH 構成ステートメントは、 Net.Data がラージ・オブジェクト (LOB) を書き込むディレクトリーを指定します。この値は、初期設定ファイルを構成することによって変更することができます。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。

構文 :

```
HTML_PATH [=] path
```

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

Net.Data の初期設定ファイル :

```
HTML_PATH = /pub/htm
```

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、103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。

環境構成ステートメント

ENVIRONMENT ステートメントは、言語環境を構成します。言語環境とは、Net.Data の構成要素です。Net.Data は、この構成要素を使用して、DB2 データベースのようなデータ・ソースにアクセスしたり、あるいは、REXX のような言語で書かれたプログラムを実行します。 Net.Data は、言語環境のセットと、ユーザー自身の言語環境の作成を可能にしてくれるインターフェースを提供してくれます。これらの言語環境および言語環境インターフェースは、 *Net.Data の言語環境解説書*で説明されています。

Net.Data は、ENVIRONMENT ステートメントが、言語環境のための Net.Data の初期設定ファイルに存在することを要求します。それによって初めて、言語環境を起動することができるのです。

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

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

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

ENVIRONMENT の追加と更新を行うには、以下のようにします。

ENVIRONMENT ステートメントは以下の構文を持っています。

```
ENVIRONMENT(type) library_name (parameter_list, ...) [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 あるいは共有ライブラリーの名前。 OS/2 では、DLL 名には、拡張子 *.dll* を付けずに指定します。AIX では、共有オブジェクトの名前に、拡張子 *.o* を付けて指定します。

- *parameter_list*

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

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

  言語環境で処理される関数を実行する前に、これらのパラメーターを、ユーザーのマクロで、構成変数、あるいは変数として定義しなければなりません。関数が、その出力パラメーターのどれかを変更すると、パラメーターは、関数が完了後も、それらパラメーターの値を保持します。

- *cliette_name*

  クライエットの名前。 *cliette_ name* は、 java アプリケーション言語環境のクライエットを参照したり、データベースのクライエットになることができます。

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 コネクション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、データベースと Java アプリケーション言語環境にだけと、これらのためにクライエットに書き込ませるユーザー定義の言語環境に指定することができます。

Java アプリケーション・クライエット

このクライエット名は、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・クライエット

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

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、 46ページの 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 INI ファイルを処理する場合、Net.Data は、言語環境の DLL あるいは、共有ライブラリー をロードすることはありません。 Net.Data が言語環境 DLL をロードするのは、それが最初に、言語環境を識別する関数を実行するときです。 DLL は、Net.Data がロードされている限り、ロードされた状態を続けます。

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

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

```
ENVIRONMENT (DTW_SQL)      DTWSQL    ( IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, SHOWSQL, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
  CLIETTE "DTW_SQL:MYDBASE"
ENVIRONMENT (DTW_SYB)      DTWSYB    ( IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, SHOWSQL, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
ENVIRONMENT (DTW_ORA)      DTWORA    ( IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, SHOWSQL, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
ENVIRONMENT (DTW_ODBC)     DTWODBC   ( IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, SHOWSQL, ALIGN, DTW_SET_TOTAL_ROWS)
ENVIRONMENT (DTW_APPLET)   DTWJAVA   ( )
ENVIRONMENT (DTW_JAVAPPS)   ( OUT RETURN_CODE ) CLIETTE "DTW_JAVAPPS"
ENVIRONMENT (DTW_PERL)     DTWPERL   ( OUT RETURN_CODE )
ENVIRONMENT (DTW_REXX)     DTWREXX   ( OUT RETURN_CODE )
ENVIRONMENT (DTW_SYSTEM)   DTWSYS    ( OUT RETURN_CODE )
ENVIRONMENT (HWS_LE)       DTWHWS    ( OUT RETURN_CODE )
```

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

Live コネクションの構成

Live コネクション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、および UNIX 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エット、すなわち、オープン・コネクションを保持するプロセスの使用により、 Live コネクションは、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 コネクションは、構成ファイル dtwcm.cnf を使用して、開始すべきクライエットを決定します。このファイルは、Live コネクションとともに使用されるクライエットごとの管理情報および定義を含んでいます。 Live コネクションに関してさらに詳しく知りたい場合は、144ページの『接続管理によって、パフォーマンスを向上させる』 を参照してください。

23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 のクライエット情報
- Java アプリケーションのクライエット情報

```
 1  CONNECTION_MANAGER{
 2  MAIN_PORT=7100
 3  ADMIN_PORT1=7101
 4  ADMIN_PORT2=7102
 5  }
 6
 7  CLIETTE DTW_SQL:CELDIAL{
 8  MIN_PROCESS=1
 9  MAX_PROCESS=5
10  START_PRIVATE_PORT=7200
11  START_PUBLIC_PORT=7210
12  EXEC_NAME=./dtwcdb2
13  DATABASE=CELDIAL
14  LOGIN=marshall
15  PASSWORD=stlpwd
16  }
17
18  CLIETTE DTW_JAVAPPS{
19  MIN_PROCESS=1
20  MAX_PROCESS=5
21  START_PRIVATE_PORT=7300
22  START_PUBLIC_PORT=7310
23  EXEC_NAME=./javaapp
24  }
```

- 行 1 ～ 5 は、構成ファイルに必要であり、Live コネクションとともに使用される一意的なポート番号を定義します。
- 行 7 ～ 16 は、すべてのデータベースのクライエットを定義し、クライエット名、実行すべきプロセスの数、データベース名、ポート番号、およびクライエット実行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。これらの追加値は、行 13 ～ 15 に示されています。
- 行 19 ～ 25 は、Java アプリケーションのすべてのクライエットを定義し、クライエット名、実行すべきプロセスの数、固有のポート番号、およびクライエット実行ファイルを識別します。

図 6. Live コネクションの構成ファイル

事前準備 : Live コネクション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 コネクションのポートの構成 :

1. エディターで構成ファイル dtwcm.cnf を開く。
2. Live コネクションの以下の 3 つのポート番号を構成する。
   - MAIN_PORT
   - ADMIN_PORT1
   - ADMIN_PORT2

   23ページの図 6 は、デフォルトのポート番号を表示しています。これらの番号が一意的でない場合、番号を一意的なポート番号に変更しなければなりません。
3. **重要 :** MAIN_PORT の値が、確実に、 Net.Data の初期設定ファイルにある DTW_CM_PORT の値と一致する用にしてください。

データベースのクライエットの構成するには、以下を行います。

1. クライエットの環境ステートメントを入力する。

   ```
   CLIETTE type:db_name
   ```

   パラメーター :

   type 言語環境とクライエットを関連付ける名前。有効なタイプのリストについては、46ページの 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。これは、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 を同じ場合がよくあります。しかし、*db_name* は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。
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エットを開始し、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。使用する値は、パフォーマンスに影響を与えますが、後で変更することができます。

   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エット・プロセスの数。このクライエッ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エットの最大数。このクライエッ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. ユーザー・システム上でデータベースのクライエットに使用するポート番号を決定する。これらの番号は、キャッシュ管理プログラムあるいは他のアプリケーションに使用されるポート番号と競合しないように、一意的でなければなりません。各クライエットは、2 つのポートを使用します。ポートの集合を指定する場合、使用するポート番号の範囲を指定しなければなりません。最初の 2 つの値は、START_PUBLIC_PORT とSTART_PRIVATE_PORT です。もう 1 つは MAX_PROCESS で、これはクライエットの最大数を示しています。以下の例は、どのポート番号を使用すべきかを示しています。

```
START_PUBLIC_PORT=1000
START_PRIVATE_PORT=1010
MAX_PROCESS=5
```

この例は、以下のポートを使用しています。

1000	1010
1001	1011
1002	1012
1003	1013
1004	1014

よくある誤りは、使用するポート番号を 2 組のクライエットの間でオーバーラップさせたり、キャッシュ管理プログラムのポート番号とオーバーラップさせることです。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。 ご使用のオペレーティング・システムの README ファイルには、使用しているオペレーティング・システムで有効なポート番号はどれかについての一般的なガイドラインがあります。

4. クライエット実行ファイルの名前を指定する。このファイル名は、以下のように指定されます。

```
EXEC_NAME=./dtwc xxx
```

ここで、*xxx* 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、25ページの表 4 を参照してください。

表 4. 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	SCO
DB2 プロセスのクライエット	DTW_SQL	dtwcdb2	dtwcdb2.exe	Y	Y	Y	Y	Y	N
ODBC プロセスのクライエット	DTW_ODBC	dtwcodbc	dtwcodbc.exe	Y	Y	N	N	N	N
Sybase プロセスのクライエット	DTW_SYB	dtwcsyb	dtwcsyb.exe	Y	Y	N	N	N	N
Oracle プロセスのクライエット	DTW_ORA	dtwcora	dtwcora.exe	Y	Y	N	N	N	N

5. 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、クライエットが関連付けられているデータベース名 (たとえば、MYDBASE) です。

6. オプション：LOGIN および PASSWORD 変数のデフォルト値を変更します。これにより、Net.Data は、DB2 のデータベースに接続するのと同じユーザー ID を使用し、接続管理プログラムを開始します。 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、23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エッ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エットを構成するには、以下のようにします。

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、同時処理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エットを開始し、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。使用する値は、パフォーマンスに影響を与えますが、後で変更することができます。

   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されるクライエット・プロセスの数。このクライエッ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エットの最大数。このクライエッ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. ユーザー・システム上でデータベースのクライエットに使用するポート番号を決定する。これらの番号は、キャッシュ管理プログラムあるいは他のアプリケーションに使用されるポート番号と競合しないように、一意的でなければなりません。各クライエットは、2 つのポートを使用します。ポートの集合を指定する場合、使用するポート番号の範囲を指定しなければなりません。最初の 2 つの値は、START_PUBLIC_PORT とSTART_PRIVATE_PORT です。もう 1 つは MAX_PROCESS で、これはクライエットの最大数を示しています。以下の例は、どのポート番号を使用すべきかを示しています。

```
START_PUBLIC_PORT=1000
START_PRIVATE_PORT=1010
MAX_PROCESS=5
```

この例は、以下のポートを使用しています。

1000	1010
1001	1011
1002	1012
1003	1013
1004	1014

よくある誤りは、使用するポート番号を 2 組のクライエットの間でオーバーラップさせたり、キャッシュ管理プログラムのポート番号とオーバーラップさせることです。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。ご使用のオペレーティング・システムの README ファイルには、使用しているオペレーティング・システムで有効なポート番号はどれかについての一般的なガイドラインがあります。

Live コネクションの構成のためのヒント :

- クライエット名は、クライエッ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エットの場合、アクセスするつもりのデータベースごとに、1 つの名前付きのクライエッ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、クライエッ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 の要求がクライエッ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。
- クライエットの名前は、初期設定ファイルにおけるクライエットのタイプのための ENVIRONMENT ステートメントで参照されるクライエット名と一致していなければなりません。 クライエット名には、変数を含むことができ、DB2 クライエットの場合、クライエット名には、変数参照 $(DATABASE) を含まれていなければなりません。 ENVIRONMENT ステートメントにおけるクライエット名のデフォルト値は、DTW_SQL:$(DATABASE) です。 INI ファイルの変数参照を使用することができますが、 Live コネクション構成ファイルの変数参照を使用することはできません。

  DATABASE 変数は、Net.Data のマクロ・ファイルで定義されます。マクロ・ファイルで 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、アクセスを希望していた 3 つのデータベース (たとえば、D1、D2、および D3) があり、初期設定ファイルに標準の CLIETTE ″DTW_SQL:$(DATABASE)″ 行がある場合は、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、クライエットが処理を終えるまで、最後の要求は待機しています。クライエッ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エッ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、同じクライエッ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、幾つかのデータベースだけに Live コネクション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 セクションに出会った場合、Net.Data は、接続管理プログラムに、特定のクライエットを要求します。接続管理プログラムが、そのタイプのクライエッ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 コネクション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。

FastCGI のための Net.Data の構成

FastCGI により、Apache Web サーバー、および IBM Internet Connection Secure Server (ICSS) に続く製品である、 Lotus Domino Go Webserver において、FastCGI モードで Net.Data を実行することができます。 FastCGI モードは、CGI-BIN プログラムの信頼度で、他の Web API プログラムと同様のパフォーマンスを提供します (独立したメモリー空間)。

事前準備 :

FastCGI を使用する前に、前提条件となっている以下の製品のインストールを必ず完了しているようにしてください。

- **Apache の場合 :** Apache Web Server 1.2.0 あるいはそれ以上を、ダウンロードし、インストールする。
- **Lotus Domino Go Webserver の場合 :** AIX 形式の Lotus Domino Go Webserver をダウンロードし、インストールする。

  `http://www.ics.raleigh.ibm.com/dominowebserver`

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 SYBASE=sybase_path
     -initial-env DSQUERY=sybase_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>
     ```

   AIX 上の Lotus 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 SYBASE=sybase_path
     Environ DSQUERY=sybase_instance
     ```

```
Environ DB2INSTANCE=db2_instance
Environ RXQUEUE_OWNER_PID=REXX_perf_var
Environ LANG=locale
}
```

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かまいません。調整に関する情報については、143ページの『FastCGI によるパフォーマンスの向上』 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 INI 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。

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 の Live コネクション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
```

sybase_path

Sybase の使用時に必要です。Sybase 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 SYBASE=/home.native/sybase/product
```

Domino Go Webserver の場合 :

```
Environ SYBASE=/home.native/sybase/product
```

sybase_instance

Sybase の使用時に必要です。Sybase のデータベースのインスタンス。Sybase の Live コネクションを使用しなければなりません。

Apache の場合 :

```
-initial-env DSQUERY=SybaseAIX
```

Domino Go Webserver の場合 :

```
Environ DSQUERY=SybaseAIX
```

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・ファイルでこの変数を宣言します。 この変数の詳細については、*Net.Data 解説書* の付録 『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
```

ロケール

UNIX のロケール変数。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
```

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、以下のようにします。

   ```
   <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、以下のようにします。

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

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。

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

各 API についての詳細は、 141ページの『Web Server API を使用してパフォーマンスを向上させる』 および Net.Data のご使用のバージョンの README ファイルを参照してください。

要件 : ICAPI、GWAPI、ISAPI、あるいは NSAPI モードで Net.Data を実行するには、Web サーバーを再構成して、その保守宣言として Net.Data DLL または共用ライブラリーを使用しなければなりません。再構成後、ご使用の Web サーバーをリスタートして、 Net.Data 初期設定ファイルに加えたすべての変更が有効になるように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、Net.Data の構成方法と、 Web サーバー API モードを実行するための Web サーバーの構成方法について説明します。一般的なステップと例を提供しますが、それらはご使用のオペレーティング・システムと異なる可能性があります。 特定の命令については、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。

ICAPI および GWAPI を構成するには :

Lotus Domino Go Webserverは、IBM Internet Connection Secure Server に続く製品です。アップグレードしている場合には、新しい Lotus Domino Go Webserver を使用したいかもしれません。 GWAPI と ICAPI は同じ製品で、使用している Web サーバーを確認するためにリネームしただけであることに注意してください。

1. Web サーバーを停止する。
2. ICAPI あるいは GWAPI DLL あるいは共有ライブラリーは、必ずサーバーの CGI-BIN あるいは ICAPI-LIB ディレクトリーにあるようにしてください。

   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、以下のようにします。

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

ICAPI および GWAPI は、既存のアプリケーションをサポートする完全互換性を持っています。 CGI に対して使用するのと同じ方法を使用して、URL、フォーム、あるいは、ICAPI や GWAPI とのリンクを呼び出します。 CGI を使用して実行に成功するマクロならどれでも、ICAPI または GWAPI、あるいはその両方を使用しても実行に成功します。これらのマクロに対して、変更を行う必要はありません。

ISAPI 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 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、以下のようにします。

   ```
   /inetsrv/scripts/dtwisapi.filetype
   ```

   ただし、*filetype* は、 Window NT と OS/2 のオペレーティング・システム場合には .dll に、 UNIX オペレーティング・システムの場合には .o になります。

   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 プロセッシングをバイパスするため、フォームとリンクに 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 が CGI プログラムとして Net.Data を起動する場合です。

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

   次に、以下の URL で、ISAPI プラグインとして Net.Data を起動しなければなりません。

   ```
   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の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 モードで起動する等価 URL は、以下の通りです。

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

5. Web サーバーを再始動する。

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、以下のようにします。

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

   ただし、*filetype* は、 Window NT と OS/2 のオペレーティング・システム場合には .dll に、 UNIX オペレーティング・システムの場合には .o になります。

   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	以下の型を追加します。ただし、*d2w* は、マクロ・ファイルのデフォルトの拡張子です。任意の 3 文字を組み合わせて指定することができます。 `type=magnum-internal/d2w exts=d2w`

4. Net.Data のマクロ・ファイルを、 netdata/macro ディレクトリーから、以下のサーバーのルートの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、どこでマクロ・ファイルを探せばよいかが分かります。
6. NSAPI は CGI プロセッシングをバイパスするため、フォームとリンクに URL の cgi-bin/db2www/ の部分は必要ありません。サーバーは、d2w ファイルのタイプをもつファイルは、 Net.Data のマクロであることを知っています。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
   ```

   たとえば、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 のマクロをいくつも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 のマクロを一緒に、サーバーのルートの文書ディレクトリーに移動する。
2. 初期設定ファイルの MACRO_PATH 変数を更新して、マクロ・ファイルが位置す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/` の保管されている 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 コネクション (DTWCM.CNF) の構成ファイルの構成および管理を支援してくれます。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。

- 36ページの『管理ツールの開始』
- 36ページの『パス・ステートメントの構成』
- 38ページの『ポートの構成』
- 39ページの『クライエットの構成』
- 43ページの『言語環境の構成』
- 47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が正しいかどうかの確認についてを理解するには、 35ページの『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、クライエット、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 のインストール後、ユーザーの CLASSPATH に、 `classes.zip` があるかどうか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、インストール中に 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、管理ツールを開始する方法が決まります。

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げられ、「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、17ページの『パス構成ステートメントのカスタマイズ』 で説明されています。37ページの図 7 は、「**パス (Path)**」ページを示しています。

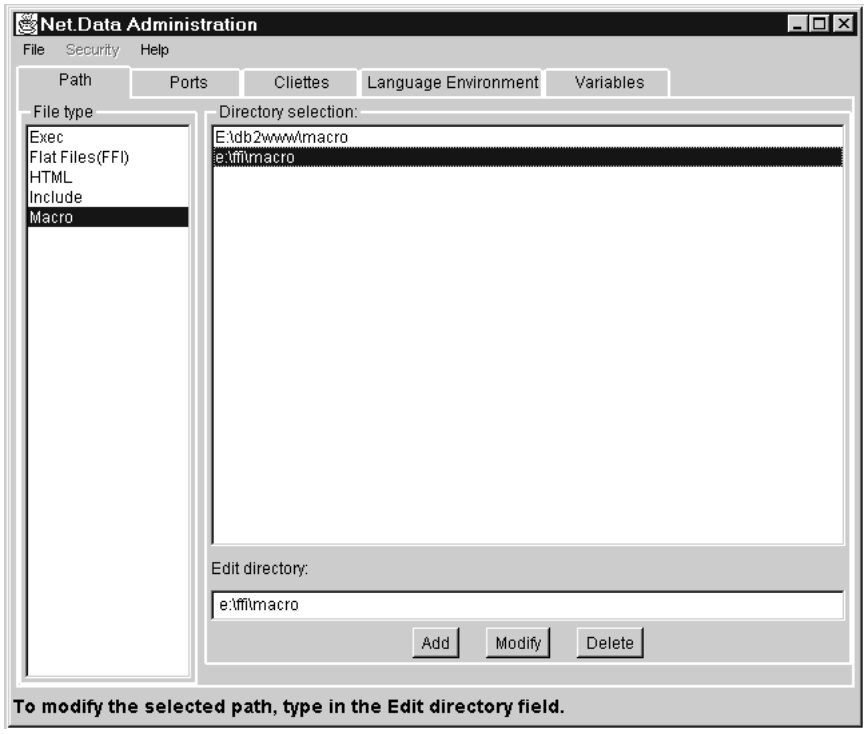


図 7. Net.Data の管理ツールの「パス (Path)」ページ. このページを使用して、パス・ステートメントの追加、変更、あるいは削除を行います。

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。 38ページの図 8 は、「**ポート (Port)**」ページを示しています。

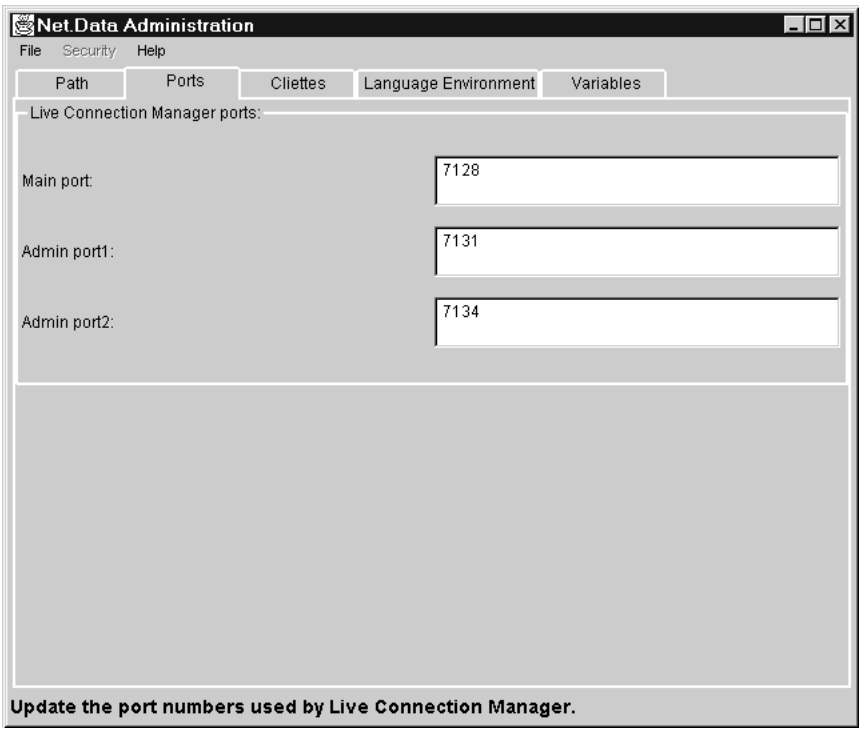


図 8. Net.Data の管理ツールのポート・ページ. このページを使用して、ポートを指定します。

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。

クライエットの構成

「**クライエット (Cliette)**」ページを使用して、 Live コネクションのデータベースのクライエットを追加、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エッ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。クライエットについての詳細は、144ページの『接続管理によって、パフォーマンスを向上させる』 で提供されています。39ページの図 9 は、「**クライエット (Cliette)**」ページを示しています。

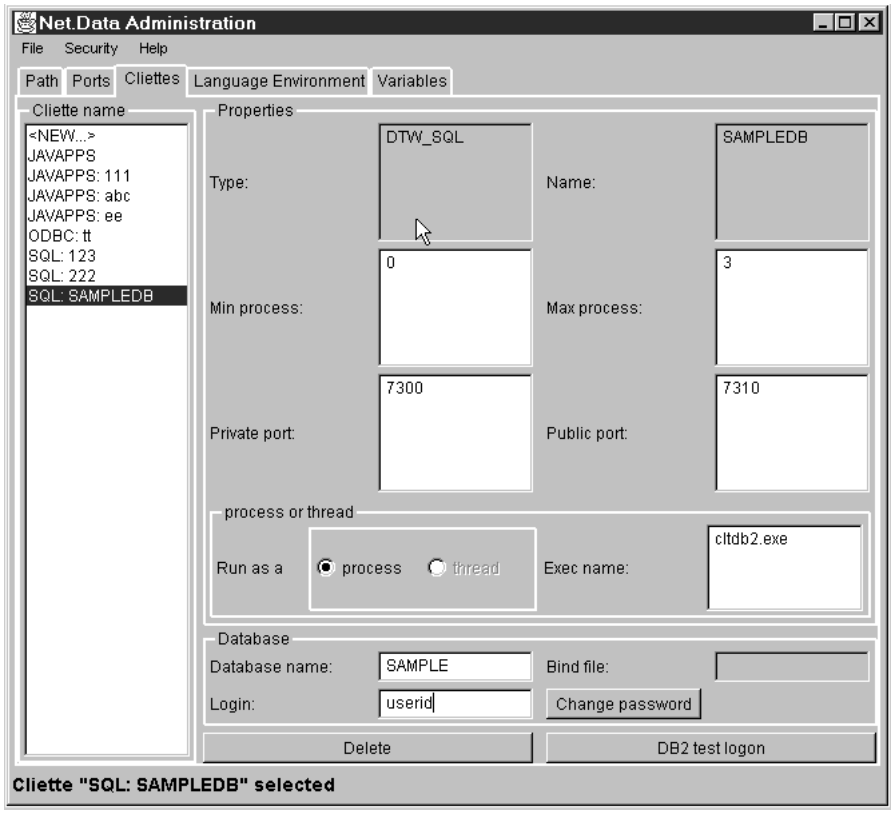


図 9. Net.Data の管理ツールのクライエットのページ. このページを使用して、クライエットの追加、変更、あるいは削除を行います。

クライエットを追加するには、以下のように行います。

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

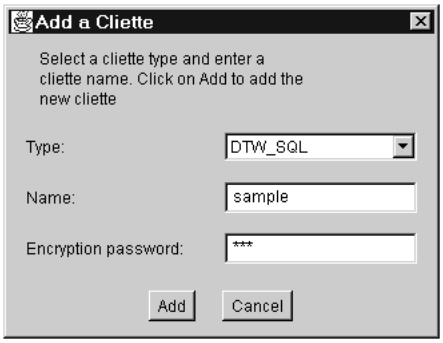


図 10. Net.Data の管理ツールの「クライエットの追加 (Add a Cliette)」ウィンドウ. このページを使用して、クライエットを追加します。

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、MYCLIETTE 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ーが、「**プロパティー (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 (Cliette)**」ページで、「**クライエット名 (Cliette 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ーが、「**プロパティー (Properties)**」グループ・ボックスに表示されます。
3. 必要に応じて、「**プロパティー (Properties)**」グループ・ボックスからプロパティーを変更する。
   a. 「**タイプ (Type)**」フィールドは、定義中のクライエッ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エットを追加すると、このフィールドに追加し、選択項目が、「クライエットの追加 (Add a Cliette)」ウィンドウの「**クライエット・タイプ (Cliette type)**」リストで定義されます。
   b. 「**名前 (Name)**」フィールドは、クライエット名を表示します。普通この名前は、データベースの名前です。 Net.Data は、新規のクライエットを追加すると、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エット・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、23ページの『Live コネクションの構成』 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エット・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、23ページの『Live コネクションの構成』 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エット・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エット・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、クライエットの実行可能ファイルの名前が表示されます。

4. クライエッ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. クライエッ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、使用中のクライエッ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「**クライエット (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。

クライエットを削除するには、以下を行います。

1. 管理ツールを開始する。
2. 「**クライエット (Cliette)**」ページで、「**クライエット名 (Cliette name)**」リストから、削除したいクライエッ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。

クライエットのユーザー 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エッ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 コネクション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 コネクション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 コネクションの構成ファイル `<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「**クライエット (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エッ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、ユーザーのクライエットのデータベースのパスワードをすべて暗号化する。

クライエットのユーザー 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「**クライエット (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 コネクションのファイル <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「**クライエット (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「**クライエット (Cliette)**」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、20ページの『環境構成ステートメント』 で議論されています。44ページの図 11 は、「**言語環境 (Language Environment)**」ページを示しています。

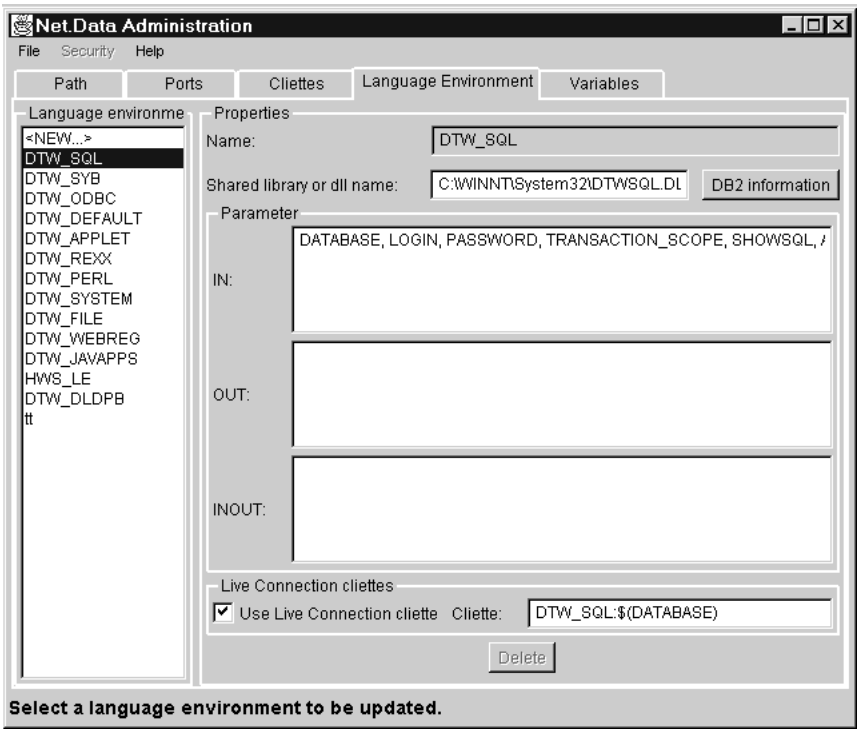


図 11. Net.Data 管理ツールの「言語環境 (Language Environment)」ページ. このページを使用して、言語環境を指定します。

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ーが、「**プロパティー (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ーが、「**プロパティー (Properties)**」グループ・ボックスに表示されます。
3. 必要に応じて、44ページの図 12 で示されている「**プロパティー (Properties)**」グループ・ボックスでプロパティーを変更する。

   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、20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、45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。

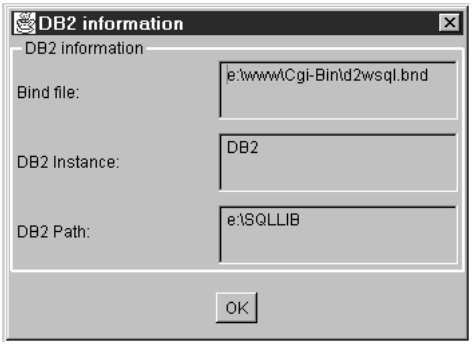


図 13. Net.Data の管理ツールの「DB2 の情報 (DB2 Information)」ウィンドウ. このページを使用して、DB2 データベースだけの情報を指定します。

   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 コネクションのクライエット (Live Connection cliettes)**」グループ・ボックスで指定する。

1) 「**Live コネクション・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) 「**クライエット (Cliette)**」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エッ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 クライエッ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_SYB, DTW_SQL, DTW_JAVAPPS

OS/2 の場合 :
DTW_SQL, DTW_JAVAPPS

AIX の場合 :
DTW_ODBC, DTW_ORA, DTW_SYB, DTW_SQL, DTW_JAVAPPS

name 「**クライエット (Cliette)**」ページで定義されたのと同じクライエット名。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 : ユーザーが定義した言語環境しか削除できません。Net.Data に添付されているデフォルトの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。 47ページの図 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 のファイルへのアクセス権の指定

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 が使用するファイルへの適切なアクセス権を持つようにする必要があります。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 が実行するときのユーザー ID は、以下の許可を持っていなければなりません。

- 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および DLL の実行、および実行可能ファイルおよびDLLへのパスにあるディレクトリーの検索
- 適切な Net.Data のマクロ・ファイルの読み取り、および MACRO_PATH パス構成ステートメントで識別される適切なディレクトリーの検索
- 適切なファイルの実行、および EXEC_PATH パス構成ステートメントで識別される適切なディレクトリーの検索
- 適切なファイルの読み取り、および INCLUDE_PATH パス構成ステートメントで識別される適切なディレクトリーの検索
- 適切なファイルの読み取りと書き込み、および FFI_PATH パス構成ステートメントで識別される適切なディレクトリーの検索
- Live コネクション構成ファイル DTWCN.CNF の読み取り
- キャッシュ管理プログラム構成ファイルの読み取り
- 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

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

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

- 49ページの『ファイアウォールを使用する』
- 51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』
- 52ページの『認証を使用する』
- 52ページの『許可を使用する』
- 52ページの『Net.Data のメカニズムを使用する』

さらに、Net.Data ではデータベース・クライエット・パスワードの暗号化を提供します。詳しくは 39ページの『クライエットの構成』 を参照してください。

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

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

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

以下の可能な構成では、ご使用の Net.Data アプリケーションの機密保護の管理に対して勧告を提供します。これらの構成は、ハイレベルの情報を提供し、公用インターネットからユーザーのセキュア・イントラネットを分離するファイアウォールを構成していることを前提としています。組織のセキュリティー・ポリシーと合わせて、これらの構成を注意深く考慮してください。

- **高度な安全保護構成 :**

  この構成によって、セキュア・イントラネットと公用インターネットの両方から、 Net.Data と Web サーバーを分離するサブネットワークが作成されます。ファイアウォール・ソフトウェアを使用して、 Web サーバーと公用インターネットの間にファイアウォールを、 DB2 サーバーを含む Web サーバーとセキュア・イン

トラネットの間に別のファイアウォールを作成します。この構成を、50ページの図 15 に示します。

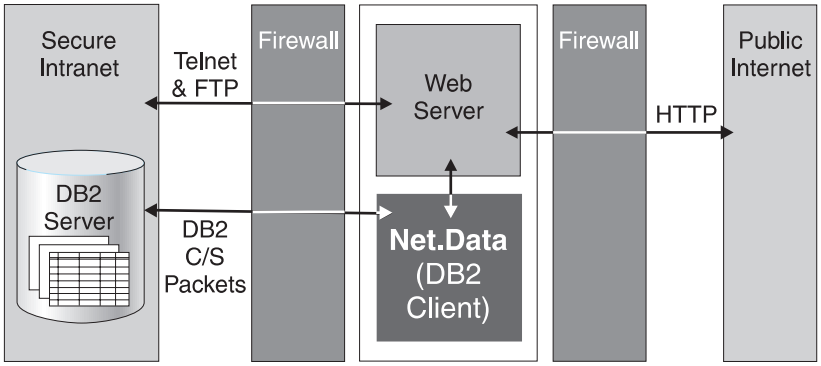


図 15. 高度な安全保護構成

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・マシンでのクライアント・アプリケーション・イネーブラー (CAE) のインストール
  - ファイアウォールを通して DB2 通信を可能にするファイアウォールの構成。1 つのメソッドは、パケット・フィルター規則を追加して、 Net.Data からの DB2 クライアント要求を可能にし、 DB2 サーバーから Net.Data へのパケットを確認することです。
- Web サーバーとセキュア・イントラネットの間の FTP および Telnet アクセスの許可。 1 つのメソッドは、Web サーバー・マシンに socks サーバーをインストールすることです。
- ファイアウォール・ソフトウェアのパケット・フィルタ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することができます。また、出力 TCP 確認パケットが、 Web サーバーから公用インターネット上のどんなホストにも進むことができることを指定します。

• **中間安全保護構成 :**

この構成では、ファイアウォール・ソフトウェアが、DB2 サーバーで確保されたイントラネットを公用インターネットから分離します。 Net.Data および Web サーバーは、ワークステーション・プラットフォーム上のファイアウォールの外側にあります。この構成は、最初の構成より簡単ですが、それでも、データベース保護が可能です。 51ページの図 16 は、この構成を示しています。

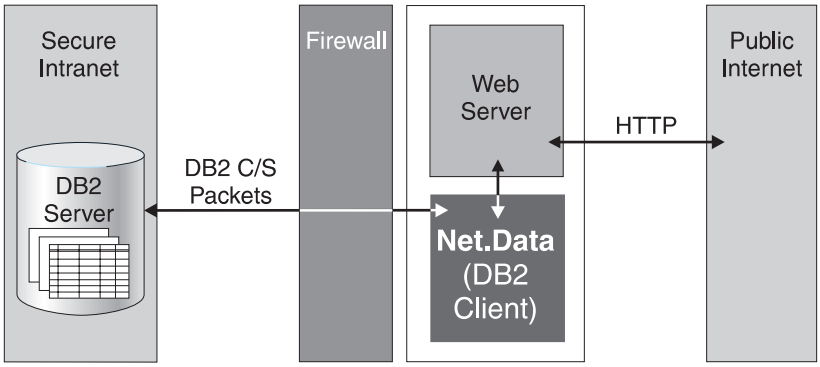


図 16. 中間安全保護構成

Web サーバーに CAE をインストールして、 Net.Data が DB2 サーバーと通信できるようにしなければなりません。 DB2 クライアント要求が Net.Data から DB2 に流れ、確認パケットが DB2 から Net.Data に流れるように、ファイアウォールを構成しなければなりません。

- **単純な安全保護構成**

  この構成では、 DB2 サーバーおよび Net.Data はファイアウォールと確保されたイントラネットの外側にインストールされます。 DB2 と Net.Data は、外部の攻撃から保護されません。ファイアウォールは、この構成にはパケット・フィルター規則を必要としません。 51ページの図 17 は、この構成を示しています。

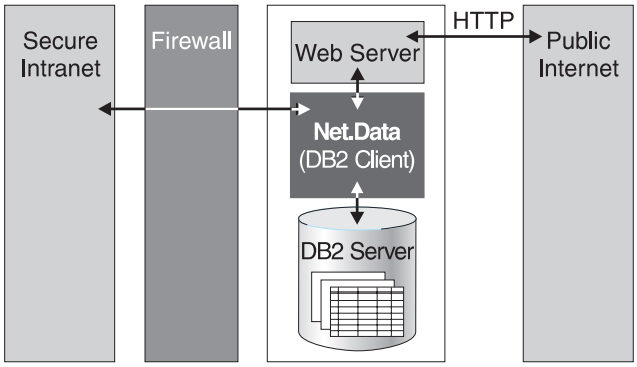


図 17. 単純な安全保護構成

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、 Internet Connection Secure Server、バージョン 2 リリース 2 またはそれ以降、および Lotus Domino Go Webserver 4.6.1 またはそれ以降などの SSL をサポートします。

認証を使用する

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。

Net.Data は、次の 2 つのタイプの認証をサポートしています。 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 サーバーのユーザー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。

許可を使用する

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。このようなメカニズムは、 Net.Data 要求を実行しているプロセスに関連したユーザー ID が、 52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のほかに、 HTML 形式または SQL ステートメントを使用するメソッドと同様に、パス・ステートメントおよび隠蔽変数などの Net.Data 提供のメカニズムを使用することができます。

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

Web ブラウザーで HTML ソースを見ることがあるユーザーから、Net.Data マクロのいろいろな特性を隠すために、隠蔽変数を使用することができます。たとえば、データベースの内部構造を隠すことができます。詳しくは 77ページの『隠し変数』 を参照してください。

以下のメソッドを使用して、保護方式をセットアップすることもできます。

- Net.Data を使用して、ユーザー独自の保護方式を作成します。たとえば、 HTML フォームを通してユーザーから妥当性検査情報を要求し、データベースのデータを使用して、または Net.Data から呼び出される外部プログラムを通して、この情報の妥当性検査を行います。
- SQL ステートメントによって、他のユーザーがデータベースに送信することができる資産を保護します。たとえば、SELECT ステートメントを 2 つのテーブルに制限します。

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

`http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.html`

第4章 Net.Data を起動する

ユーザーは Net.Data を、コモン・ゲートウェイ・インターフェース (CGI) や FastCGI を、または Lotus Domino Go Webserver (GWAPI) 、 Internet Connection Server (ICAPI)、 Netscape Server (NSAPI)、および Microsoft Internet Server (ISAPI) などの Web サーバー APIとともに使用するように、構成することができます。Net.Data を起動するために使用される構文は、Net.Data の構成のされ方によって変わってきます。Net.Data の構成を理解するには、5ページの『第2章 Net.Data の構成』 を参照してください。

本章では、CGI と構成された Net.Data の起動について説明します。 API モードので Net.Data を起動する方法について理解するには、 141ページの『Web Server API を使用してパフォーマンスを向上させる』 を参照してください。

さらにNet.Data が、マクロ・ファイルやただ 1 つの SQL ステートメント、ストアード・プロシージャー、または関数を実行するようにするかどうかも指定できます。このようなタイプの呼び出しは、それぞれマクロ要求および直接要求と呼ばれます。

マクロ要求

Net.Data マクロ言語で書かれているマクロ・ファイルを指定することによって Net.Data を起動します。

直接要求

次のように指定して Net.Data を起動します:

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

シングル SQL 照会を書き込みたい Web 開発者、または DB2 ストアード・プロシージャー、 REXX プログラム、または Perl 関数などのシングル関数を呼び出したい Web 開発者は、データベースへのダイレクト要求を発行することができます。直接要求には、Net.Data マクロ・ファイルを必要とするような複雑な Net.Data アプリケーション論理はありません。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、改善されたパフォーマンスを目指して処理するために適した言語環境に渡されます。

56ページの図 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・ファイルを指定します。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・ファイル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。

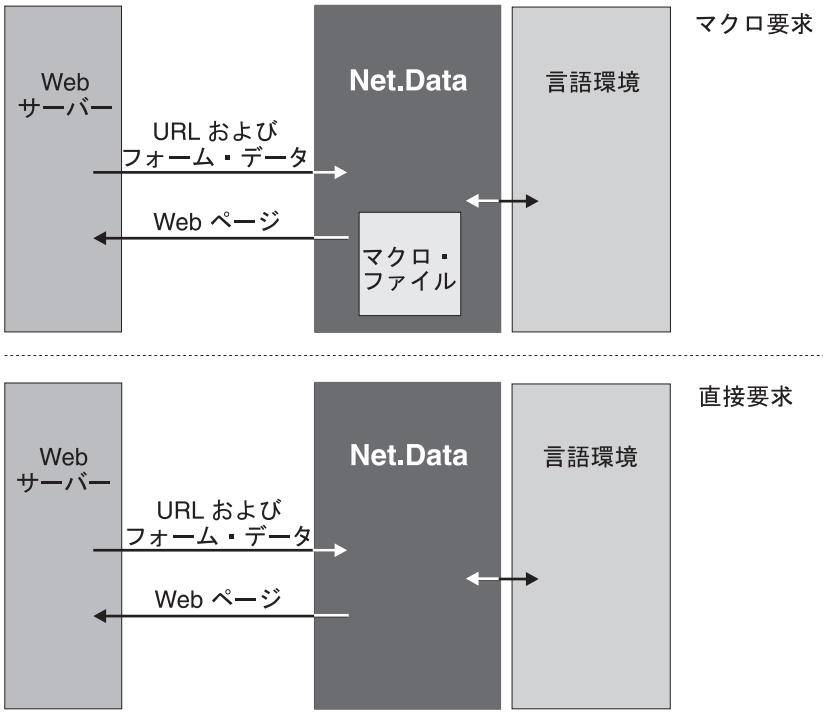


図 18. マクロ要求対直接要求

Net.Data を起動する構文は、Net.Data の構成と、作成する要求のタイプによって異なります。マクロ要求と直接要求の場合は共に、Net.Data は HTML リンク、HTML フォーム、または URL によって Web ブラウザーから起動されます。 Web サーバーは、CGI、FastCGI、または Web サーバー API の 1 つを使用して Net.Data を起動します。

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

直接要求の場合、Net.Data 言語環境の名前、SQL ステートメントまたはプログラムの名前、および追加の必須パラメーター値が、Net.Data によって定義された構文を使用して URL 内で指定されます。

本章では、両方の呼び出し方式を説明します:

- 56ページの『マクロ・ファイルで Net.Data を呼び出す (マクロ要求)』
- 59ページの『マクロ・ファイルを使用しない Net.Data の起動 (直接要求)』

マクロ・ファイルで Net.Data を呼び出す (マクロ要求)

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

次の例は、Net.Data の異なる起動の方法を示しています。

- HTML リンク:

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

- HTML フォーム:

  ```
  <FORM METHOD=method
  ACTION="http://server/cgi-bin/db2www/
  filename/block/[?name=val&...]">any text</FORM>
  ```

- URL:

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

パラメーター:

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

filename Net.Data のマクロ・ファイルの名前および拡張子を指定します。ここで、ファイル名は、MACRO_PATH で指定されるディレクトリーの下の相対パスです。

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

method フォームで使用される HTML メソッドを指定します。METHOD=POST を推奨します。

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

HTML リンク

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

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

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

このリンクは、以下のマクロを呼び出します。

```
%DEFINE DATABASE="MNS97"

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

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

この照会は、EQPTABLE 表の中に記述された各モニターに関する型式番号、コスト、および記述情報を持つ表を戻します。この例は、照会の結果をデフォルトのレポートで表示します。 REPORT ブロックを使用してレポートをカスタマイズする方法に関しては、 98ページの『レポート・ブロック』 を参照してください。

一般的に、Net.Data の各ブロックは、%block_name{ で始まり、%} で終わります。Net.Data マクロ言語のその他詳細な構文に関しては、*Net.Data 解説書* 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、 57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ーで、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/equiplst.d2w/report">
<P>What type of hardware do you want to see?
<MENU>
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="MON" checked> Monitors
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="PNT"> Pointing devices
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="PRT"> Printers
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="SCN"> Scanners
</MENU>

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

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

```
%DEFINE DATABASE="MNS97"

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

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

上記の例では、SQL ステートメント内の TYPE=$(hdware) の値は、 HTML 形式入力データから利用することができます。

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

マクロ・ファイルを使用しない Net.Data の起動 (直接要求)

このセクションでは、特定のマクロ・ファイルを指定しない Net.Data の起動の方法を示します。直接要求のスタイルによる呼び出しでは、ユーザーは Net.Data によって定義された構文に従って、 Net.Data 言語環境名、実行される SQL ステートメントまたはプログラム名、および URL 内のその他必要とされるパラメーターの値を指定します。

SQL ステートメントまたはプログラム名、およびそのほか指定されたパラメーターは、直接指定された言語環境に渡され、処理されます。直接要求によって、Net.Data がマクロ・ファイル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、Sybase、Java、 System、Perl、および REXX Net.Data によって提供される言語環境は、直接要求呼び出し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・データの照会文字列にある (NAME=VALUE) ペア・セクションの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>
```

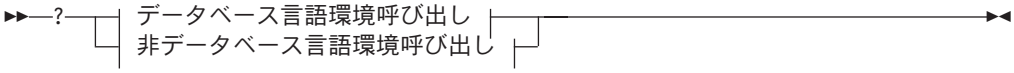
ここで、*direct_request* は、直接要求の構文を示しています。たとえば、以下の HTML リンクには、直接要求が含まれています。

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

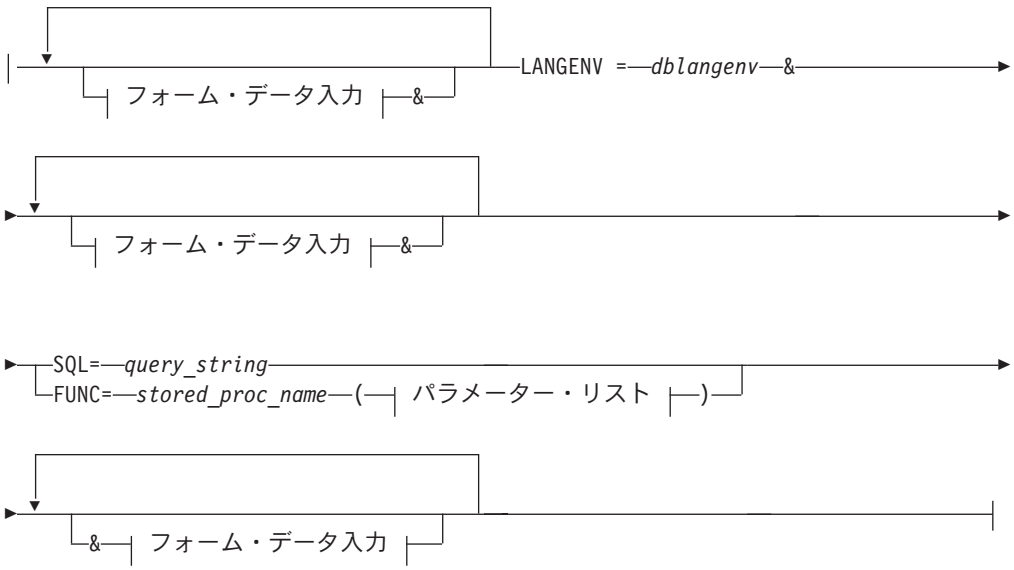
直接要求の構文

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

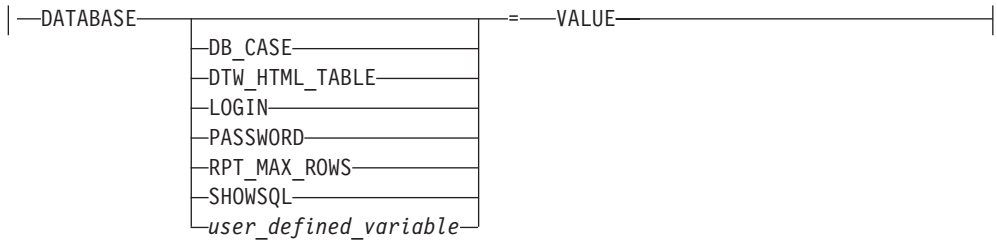
構文



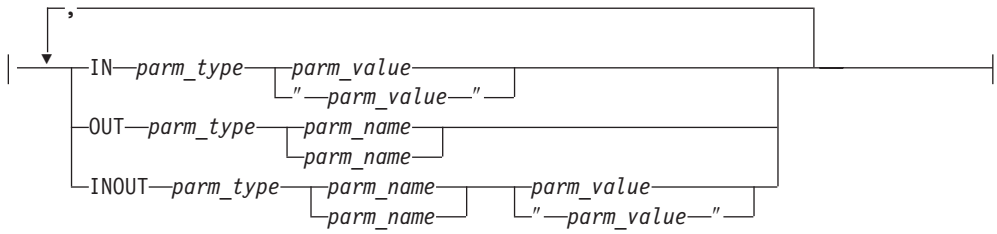
データベース言語環境呼び出し



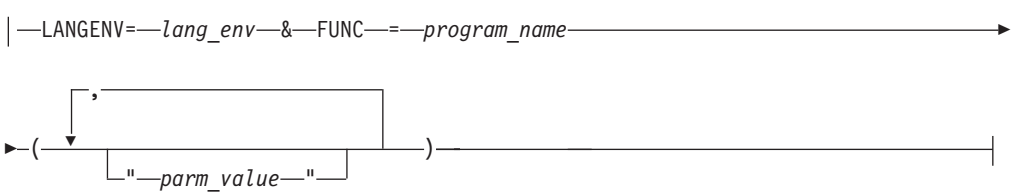
フォーム・データ入力



パラメーター・リスト



非データベース言語環境呼び出し



パラメーター

データベース言語環境呼び出し

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

フォーム・データ入力

SQL 変数の設定を指定できる、または単純な HTML 形式設定を要求するこ

とができるパラメーター。これらの変数について詳しく理解するには、*Net.Data 解説書* の変数の章を参照してください。

DATABASE
Net.Data が SQL 要求を渡すデータベースを指定します。このパラメーターは必須です。

DB_CASE
SQL ステートメントの文字ケース (大文字または小文字) を指定します。

DTW_HTML_TABLE
Net.Data が HTML テーブルを戻すべきかどうかを指定します。

LOGIN
データベース・ユーザー ID を指定します。

PASSWORD
データベース・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。

user_defined_variable
Net.Data に渡され、必要な情報または効果的な Net.Data の振る舞いを提供する変数。ユーザー定義の変数は、ユーザーがアプリケーションのため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ード・プロシージャー呼び出しの、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
データベース言語環境の名前:

- DTW_SQL
- DTW_ODBC
- DTW_ORA
- DTW_SYB

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。

照会ストリング
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、ストアード・プロシージャー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 ストアード・プロシージャーを指定します。

parm_type
DB2 ストアード・プロシージャー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 ストアード・プロシージャーに対する有効なパラメーターの値を指定します。

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

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

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

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

LANGENV
その関数を実行するターゲット言語環境を指定します。

lang_env
以下の非データベース言語環境の名前を指定します。

- DTW_PERL
- DTW_REXX
- DTW_SYSTEM

FUNC
直接要求がプログラムの実行を指定していることを示しています。

program_name
実行される関数を含んでいるプログラムを指定します。

parm_value
その関数に有効なパラメーターの値を指定します。

直接要求の例

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

HTML リンク

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

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

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

```
<A HREF="http://server/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やスペースは、 %22 および + 文字に符号化します。URL で符号化しなければならないすべての文字のリストに関しては、 *Net.Data 解説書* の DTW_URLESCSEQ 関数の記述を参照してください。

HTML フォーム

例 1: SQL 言語環境を使用する SQL 照会を実行する HTML フォーム。 CELDIAL データベースへ接続し、テーブルを照会します。

```
<FORM METHOD="POST"
 ACTION="http://server/cgi-bin/db2www/">
<INPUT TYPE=hidden NAME="LANGENV" VALUE="DTW_SQL">
<INPUT TYPE=hidden NAME="DATABASE" VALUE="CELDIAL"
 <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>
```

ヒント: Net.Data マクロで作成される HTML フォームを使用する直接要求呼び出しに、いろいろな変数置換を使用することができます。

URL

例 1: SQL 言語環境を使用する SQL 照会を実行する URL

```
http://server/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
```

ヒント: EXEC パス構成ステートメントで指定されていないパスでファイル名を参照するには、 *function_name* の値としてファイル名と完全な修飾パスを指定してください。

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・プロシージャーを呼び出し、 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
```

第5章 Net.Data のマクロ開発

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

- Web ページのレイアウトを指定する
- 変数と関数を定義する
- Net.Data に組み込まれる、またはマクロ・ファイルで定義される関数を呼び出す。
- 処理出力の HTML 形式へのフォーマットし、それを Web ブラウザーに戻す

Net.Data のマクロには、65ページの図 19 に示されているように、宣言パーツおよび HTML パーツという 2 つのパーツが含まれます。

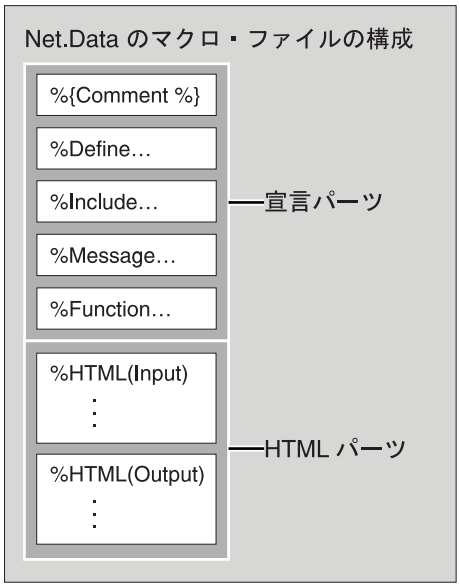


図 19. マクロ・ファイル構造

- *宣言パーツ* には、マクロ・ファイルにおける変数および関数の定義が含まれる。
- *HTML パーツ* には、Web ページのレイアウトを指定するHTML のステートメントで構成される HTML のブロックが含まれる。

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・ファイル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： Web サーバーが、確実に、このファイルへのアクセス権限を持つようにします。詳しくは、48ページの『Net.Data のファイルへのアクセス権の指定』 を参照してください。

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

- 66ページの『Net.Data のマクロ・ファイルの分析』
- 71ページの『Net.Data のマクロ変数』

- 82ページの『Net.Data の関数』
- 96ページの『マクロに HTML を作成』
- 100ページの『マクロ・ファイルにおける条件付き論理とループ』
- 103ページの『ラージ・オブジェクトを使用する』

Net.Data のマクロ・ファイルの分析

マクロ・ファイルは、次の 2 つのパーツで構成されています。

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

  宣言パーツには、 EXEC ステートメント、IF ブロック、INCLUDE ステートメント、および MESSAGE ブロックなどの、ほかの言語構成要素とステートメントが含まれます。言語構成要素についての詳細は、*Net.Data 解説書* の言語構成要素に関する章を参照してください。

  許可のためのヒント：Web サーバーが、確実に、EXEC および INCLUDE ステートメントによって参照されるファイルへのアクセス権限を持つようにします。 詳しくは、48ページの『Net.Data のファイルへのアクセス権の指定』 を参照してください。

- HTML パーツは、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るHTML ブロックを使用して、 Web ページのレイアウト、参照変数、および呼び出し関数を定義します。 Net.Data を起動するときには、マクロ・ファイルの処理のためにエントリー・ポイントとして、 HTML ブロック名を指定します。 HTML ブロックは、69ページの『HTML ブロック』 で説明されています。

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

まず最初に、マクロ全体を見渡し、次に各ブロックを詳細に見てみましょう。

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

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

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

%{ ***********************      HTML Block: Input    ************************%}
%HTML(INPUT) {
```

```
 <html>
<head>
<title>$(page_title)</title>
</head><body>
<h1>Input Form</h1>
Today is @today()

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

</form>

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

%{ **********************    HTML Block: Output     ************************%}
%HTML (OUTPUT) {
 <html>
<head>
<title>$(page_title)</title>
</head><body>
<h1>Output Page</h1>
<p>@rexx1(input_data)
<p><hr>
<p>[<a href="/">Home page</a> |
<a href="input">Previous page</a>]
</body></html>
%}
```

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

2 つの HTML ブロックは、おなじみの HTML のタグを含みます。このため、Web のマクロを書くのが簡単になります。HTML について詳しければ、マクロの作成は、単に、サーバーで動的に処理されるマクロ・ステートメントと、データベースに送信する SQL ステートメントの追加を行うだけになります。

マクロは HTML 文書に類似しているように見えますが、Web サーバーは、CGI あるいは Web サーバーの API を介してマクロにアクセスします。 Net.Data には、処理すべきマクロの名前、およびそのマクロの表示すべき HTML ブロック、という 2 つのパラメーターが必要です。

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

DEFINE ブロック

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

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

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

FUNCTION ブロック

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

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

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

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

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

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

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

ば、<b> 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 番目のブロック today も、REXX プログラムを参照します。しかし、この場合の REXX のプログラム全体 (1 行全部) は、関数宣言それ自身に含まれています。 外部プログラムは必要ありません。 インライン・プログラムは、REXX および Perl の関数では許可されています。その理由は、これらのプログラムは、動的な解析および実行ができるインタープリター言語だからです。 インライン・プログラムは、管理すべき独立したプログラムの必要がないので、単純であるという優位点を持っています。最初の REXX の関数は、インラインとしてハンドルすることも可能です。

HTML ブロック

HTML ブロックは、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。 HTML は常に Net.Data 呼び出し要求で指定され、マクロごとに少なくても 1 つの HTML ブロックがなければなりません。

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

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

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

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

ブロック全体は、HTMLのブロック識別子、%HTML (INPUT) {...%} で囲まれます。INPUT は、このブロックの名前を識別します。これに名前を付けることができます。HTML の <title>タグは、変数置換の例を含んでいます。変数 page_title の値が、フォームの表題に取って代わります。

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

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

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

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

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

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

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

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

</head><body>
<h1>Output Page</h1>
<p>@rexx1(input_data)  <--- This line contains a call to function rexx1
                           passing the argument "input_data".
<p>
< hr>
<p>
[
<a href="/">Home page</a> |
<a href="input">Previous page</a>]
%}
```

INPUT ブロックと同じように、このブロックも、変数および関数呼び出しを置換するための Net.Data のマクロ・ステートメントを持つ標準の HTMLです。再度、page_title 変数はこの title ステートメントに置換されます。そして、前と同じように、このブロックは関数呼び出しを含みます。この場合、このブロックは、関数 rexx1

を呼び出し、変数 input_data の内容をこの関数に渡します。この変数は、このブロックが Input ブロックで定義されたフォームから受け取ったものです。任意の数の変数を関数に渡したり、関数から受け取ったりすることができます。関数定義は、渡される変数の数と型を決定します。

Net.Data のマクロ変数

Net.Data により、Net.Data でマクロの変数を定義したり、参照することができますさらに、これらの変数を、マクロから言語環境に渡したり、また言語環境から受け取ったりすることができます。

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

- DEFINE ブロックで DEFINE ステートメントを使用して、明示的に定義された変数。
- 定義済みの変数。これは、Net.Data により使用可能になり、ある値が設定されます。通常、この変数を変更することはできません。
- 暗黙的に定義された変数。これには、以下の 4 つのタイプがあります。
  - 明示的には定義されていないが、最初の参照時にインスタンス化される変数。
  - FUNCTION ブロック定義の一部で、FUNCTION ブロック内でしか参照できないパラメーター変数。
  - Net.Data によりインスタンス化され、フォーム・データあるいは URL のデータの名前と値のペアに対応する変数。
  - Net.Data の表と関連付けられ、ROW ブロックあるいは REPORT ブロック内でしか参照することができない変数。

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

- グローバル

  識別子は、マクロ・ファイル内の任意の場所でそれを参照できる場合、グローバルな効力範囲を持ちます。グローバルな効力範囲を持つ識別子には、以下のようなものがあります。
  - Net.Data の組み込み関数
  - フォームのデータ
  - URL データ
  - HTML ブロック内からインスタンス化された変数
- マクロ・ファイル

  識別子は、その宣言がブロックの外に現れる場合、グローバルな効力範囲を持ちます。ブロックは、左大括弧 ({) で始まり、パーセント記号と大括弧 (%}) で終わります。(DEFINE ブロックは、この定義から除外され、独立した DEFINE ステー

トメントとして扱わなければなりません。) マクロ・ファイルの効力範囲を持つ識別子は、それが宣言された点から、マクロ・ファイルの終わりまでが可視になります。

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

  識別子は、関数定義のパラメーター・リスト内で宣言された場合、関数ブロックの効力範囲を持ちます。同じ名前を持つ識別子が、関数定義の外に存在する場合は、Net.Data は、その関数ブロック内の関数パラメーター・リストの識別子を使用します。 Net.Data は、関数ブロックの外部からの識別子あるいはその値を使用したり、変更したりすることはありません。

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

  識別子は、それがパラメーター・リストで宣言されておらず、関数呼び出しの前に宣言されたり初期化されたりしていない場合は、関数のブロックの効力範囲を持ちます。 そのような識別子は、関数ブロックの外では参照することができません。

- REPORT ブロック

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

- ROW ブロック

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

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

以下の節では、変数の定義およびその参照方法について説明します。また、さまざまな変数の型とその使用方法についても説明します。

変数の定義

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

- **DEFINE ステートメントまたはブロック**

  Net.Data のマクロで使用するための最も簡単な変数定義の方法は、以下のように DEFINE ステートメントを使うことです。この構文は、Net.Data に固有のものです。

```
%DEFINE variable_name="variable value"

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

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

ストリングに引用符を組み込むためには、2 つの引用符を続けて使用します。連続する引用符が 2 つだけの場合は、ヌル・ストリングに等しくなります。たとえば、以下のようにします。

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

変数 HI は、say "hello" を表示します。

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

変数 reply は、hello を表示します。

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

変数 empty は、ヌルです。

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

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

- **HTML のフォームの SELECT および INPUT タグ**

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

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

  あるいは

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

  variable_name は、その変数に与えた名前です。そして、その変数の値は、フォームで受け取った入力から決定されます。 Net.Data のマクロにおける、このような変数定義のタイプの使用方法の例については、58ページの『HTML フォーム』 を参照してください。

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

- **URL のデータ**

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

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

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

変数の参照

定義済みの変数を参照して、その値を戻すことができます。

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

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

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

変数を HTMLの一部として使用するには、HTML ブロックでそれらの変数を参照します。たとえば、すでに変数 homeURL を定義している場合、以下のようにします。

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

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

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

変数は、Net.Data のマクロの任意のパーツで参照することができます。変数が、それが参照されたときに未定義である場合、Net.Data は、変数を定義し、その変数に初期値としてヌルを与えます。 変数が、それが参照されたときに未定義である場合、Net.Data は空ストリングを戻します。 Net.Data は変数を定義しません。

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

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

変数の型

マクロ・ファイルでは、以下の変数参照のタイプを使用することができます。

- 75ページの『条件変数』
- 75ページの『環境変数』
- 76ページの『実行可能な変数』
- 77ページの『隠し変数』
- 78ページの『リスト変数』

- 79ページの『表変数』
- 79ページの『各種変数』
- 80ページの『表処理変数』
- 81ページの『レポート変数』
- 82ページの『言語環境変数』

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

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、変数が取ることができる 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、78ページの『リスト変数』 を参照してください。

環境変数

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

環境変数を定義するための構文は以下のとおりです。

```
%define var=%ENVVAR
```

ただし、*var* は定義される変数の名前です。

たとえば、変数 SERVER_NAME は、以下の環境変数として定義することができます。

```
%define SERVER_NAME=%ENVVAR
```

そして、参照は以下のように行います。

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

出力は以下のように見えます。

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

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

実行可能な変数

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

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

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

`runit` は、実行可能な変数になります。

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

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

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

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

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

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

1. Net.Data は、 Net.Data の初期設定ファイルの EXEC_PATH で指定されるディレクトリーを検索する。詳細については、18ページの『EXEC_PATH』 を参照してください。
2. Net.Data がプログラムを見つけなかった場合は、システムの PATH 環境変数、あるいはライブラリー・リストで定義されるディレクトリーを、システムが検索します。実行可能なプログラムが見つかれば、Net.Data は、そのプログラムを実行します。

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

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

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

この次の例では、システム日付を HTML リ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 HTML のソースを見ているアプリケーション・ユーザーから隠すことができます。隠し変数を定義するには、以下のようにします。

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

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

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

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

.
.
.

Web ブラウザーが HTML のフォームを表示すると、$$(name) および $$(addr) は、$(name) および $(addr) にそれぞれ置き換えられます。その結果、実際の表と列名は、HTML のフォームに表示されることはありません。アプリケーションのユーザーは、真の変数名が隠されているのかどうかを知ることができません。ユーザーがフォームを処理依頼すると、HTML(REPORT) ブロックが呼び出されます。 @mySelect() が FUNCTION ブロックを呼び出すと、$(Field) は、SQL ステートメントにおいて、SQL 照会の customer.name あるいは customer.addr で置き換えられます。

リスト変数

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

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

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

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

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

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

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

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

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

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

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

表変数

表変数は、関係しあうデータの集合を定義します。表変数は、同一のレコード、すなわち行の配列と、各行のフィールドを説明する列名の配列を含みます。表は、Net.Data のマクロでは、以下のステートメントのようにして定義されます。

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

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

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

表を定義するときに、表がゼロの行とゼロの列を持つと、記憶域が割り当てられません。表に値を代入するには、表を OUT あるいは INOUT パラメーターとして、関数に渡す以外に方法はありません。 DTW_SQL 言語環境は、SELECT ステートメントの結果を表に自動的に書き込みます。

DTW_REXX あるいは DTW_PERL など、他のすべての環境の場合は、言語環境が自動的に表の値を設定することはありません。そのかわり、表の個々の要素は、ネイティブの言語のインタープリターから出力パラメーターとして利用できるようにすることができます。そのためには、表の個々の要素は、スクリプトあるいはプログラムを実行して設定されなければなりません。詳細については、*言語環境解説書* の言語環境の説明を参照してください。

表変数の名前を参照することにより、関数間で表を渡すことができます。表の個々の要素は、関数の REPORT ブロックで参照することができます。詳細については、80ページの『表処理変数』 を参照してください。表変数は、通常、SQL 関数の値が代入され、次に、SQL 関数あるいは別の関数のいずれかにパラメーターとして渡された後、その関数におけるレポートへの入力として使用されます。表変数を、IN、OUT、あるいは INOUTパラメーターとして、任意の非 SQL 関数に渡すことができます。表は、SQL関数には、OUT パラメーターとしてのみ渡すことができます。

表の列名とフィールドの値は、1 を原点に持つ配列要素としてアドレス指定されます。配列 は 0 から開始する、という標準 C および C++ の言語規則とは異なります。

各種変数

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

- Net.Data の処理に影響を与える
- 関数呼び出しの状態を検出する

- データベース照会の結果セットに関する情報を取得する
- ファイル場所と日付に関する情報を決定する

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

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

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

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

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

有効な各種変数のリストおよびその構文と例については、*Net.Data 解説書* を参照してください。

表処理変数

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

表処理変数は、Net.Data が決定する事前定義値を持ち、これにより、SQL 照会あるいは関数呼び出しの結果セットからの値を、処理中の列、行、あるい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。 HTML 出力の現行列名を参照することができます。

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

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

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

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

表処理変数の中には、他の変数あるいは組み込み関数により影響されるものがあります。たとえば、TOTAL_ROWS は、DTW_SET_TOTAL_ROWS という SQL の言語

環境変数をアクティブにして、以下の例のように、SQL 照会あるいは関数呼び出しからの結果を処理するときに、Net.Data が TOTAL_ROWS の値を割り当てるように、要求します。

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

Names found: $(TOTAL_ROWS)
```

有効な表処理変数のリストおよびその構文と例については、*Net.Data 解説書* を参照してください。

レポート変数

Net.Data は、デフォルトのレポート形式のマクロ・ファイルから生成される HTML 出力を表示します。デフォルトのレポート形式は、<PRE> </PRE> タグを使用して、表形式で表示されます。出力を表示するための命令をもつ REPORT ブロックを定義して、あるいは、デフォルトのレポートが生成されるのを防ぐためのレポート変数の 1 つを使用して、デフォルトのレポートを取り消すことができます。

レポート変数は、HTML 出力の表示方法、およびデフォルトのレポートと Net.Data 表と一緒に使用する方法をカスタマイズするのに役立ちます。レポート変数は、使用する前に、DEFINE ステートメント、あるいは @DTW_ASSIGN() で定義されなければなりません。

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

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

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

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

また、Net.Data が、デフォルトの景色設定に 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)
%}
```

有効なレポート変数のリストおよびその構文と例については、*Net.Data 解説書* を参照してください。

言語環境変数

言語環境変数は、言語環境と共に使用され、言語環境による要求処理の方法に影響を与えます。言語環境変数は、それらを参照する前に、DEFINE ステートメント、あるいは @DTW_ASSIGN() 関数で定義されなければなりません。適切なNet.Data のマクロ・ブロックにおいて、言語環境変数を設定あるいは参照します。

言語環境変数を使用すると、データベースとの接続の確立、Java アプレットの代替えテキストの供給、それに NLS サポートを使用可能にする、などのタスクを実行することができます。

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

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

この次の例では、アクセスする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"
```

有効な言語環境変数のリストおよびその構文と例については、*Net.Data 解説書* を参照してください。

Net.Data の関数

Net.Data は、Web アプリケーションで有益だと思われる多くの関数を提供します。ユーザー自身が関数を書くことも簡単です。83ページの図 20は、Net.Data 関数およびマクロ・ファイルがどのようにして対話しているかを示しています。

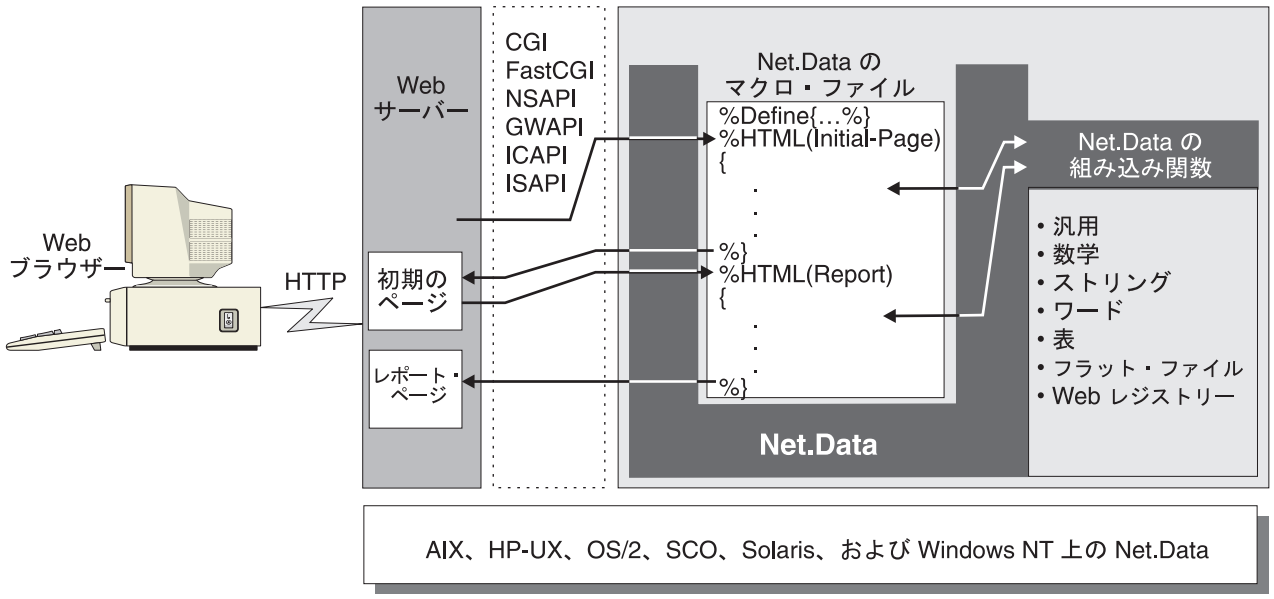


図 20. Net.Data の組み込み関数

関数の定義

ユーザー自身の関数を定義したり、Net.Data の組み込み関数を使用することができます。ユーザー自身の関数をマクロ・ファイル (ユーザー定義の関数という) に定義するには、 FUNCTION ブロック または MACRO_FUNCTION を使用します。 block

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

MACRO_FUNCTION ブロック
Net.Data のマクロから呼び出され、別の言語環境ではなく、Net.Data により処理されなければならないサブルーチンを定義します。ブロック内のステートメントは、Net.Data のマクロ言語のソース・ステートメントでなければなりません。

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

構文を以下に示します。

FUNCTION ブロック:

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

MACRO_ FUNCTION ブロック :

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

ここで、

type

初期設定ファイルにおいて構成されている言語環境を識別します。言語環境は、特定の言語プロセッサー (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 と言語プロセッサーとの標準インターフェースを提供します。

幾つかのデフォルトの言語環境が Net.Data で提供されています。

function-name

FUNCTION あるいは MACRO_FUNCTION ブロックの名前を指定します。FUNCTION ブロックあるいは MACRO_FUNCTION ブロック を、関数呼び出しと共に、マクロ・ファイルのどこか別の場所で実行します。関数呼び出しは、@ 記号を前に付けた *function-name* を参照します。詳細については、86ページの『関数の呼び出し』 を参照してください。

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

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

parameter

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

return-var

このパラメーターを、RETURNS キーワードの後に指定して、特殊な OUT パラメーターを識別します。戻り変数の値は、関数呼び出しに割り当てられ、Net.Data のマクロ処理時に関数呼び出しが、この値に置き換えられます。RETURNS 節を指定しなければ、関数呼び出しの値は以下のようになります。

- 言語環境への呼び出しからの戻りコードがゼロの場合は、NULL
- 戻りコードが非ゼロの場合は、戻りコードの値

executable-statements

変数の置換および関数処理の後で、処理のための指定された言語環境に渡される言語ステートメントのセット。 *executable-statements* には、Net.Data の変数参照および Net.Data の関数呼び出しを含むことができます。実行可能ステートメントを言語環境に渡す前に、 Net.Data は、これらの変数参照または実際の値をもつ関数呼び出しを取り替えます。 *executable-statement* 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、76ページの『実行可能な変数』 を参照してください。

MACRO_FUNCTION ブロックの場合は、実行可能なステートメントは、HTML ステートメントと Net.Data のマクロ言語構成要素との組み合わせです。この場合、言語環境は全く関与しません。その理由は、Net.Data は言語プロセッサーとして働き、実行可能なステートメントを評価し、それを実行するからです。

report-block

FUNCTION ブロックの出力をハンドルするための REPORT ブロックを定義します。 98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。 94ページの『メッセージ・ブロック』 を参照してください。

最外部の Net.Data のマクロ・レイヤーと、Net.Data のマクロで呼び出される前の関数を定義します。

言語ステートメントでの特殊文字の使用

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

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

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

- インラインでコードをプットするのではなく、 EXEC ステートメントを使用してプログラム・コードを呼び出す。
- 変数参照を使用して、特殊文字を指定する。

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

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

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

- o %EXEC ステートメントを使用する。

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

- o 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 bynumber keys $(percent_openbrace) $Rtitles{$num} }) {
  &make_links($Rtitles{$num}{$num_words});
 }
 ...
%}
```

関数の呼び出し

FUNCTION ブロック名あるいは MACRO_FUNCTION ブロック名の前に付いている @ 文字を使用して、Net.Data のマクロから関数を呼び出します。

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

function_name

これは、呼び出される FUNCTION ブロック、

あるいは、MACRO_FUNCTION ブロックの名前です。関数は、それが組み込み関数でなければ、Net.Data のマクロであらかじめ定義されていなければなりません。

argument

これは、定義済みの変数、リテラル文字ストリング、変数参照、あるいは関数呼び出しの名前です。関数呼び出し時の引き数は、FUNCTION ブロック、あるいは MACRO_FUNCTION ブロックのパラメーターと一致しており、各パラメーターは、FUNCTION ブロックあるいは MACRO_FUNCTION ブロッ

クの処理中に、対応する引き数の値が割り当てられます。引き数は、対応するパラメーターと同じ数および型でなければなりません。

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

1. Net.Data は、FUNCTION ブロックの実行可能ステートメントのセクションにおける変数参照と関数呼び出しを処理する。 Net.Data は、すべての変数参照を、変数の現行値で置き換え、すべての関数呼び出しを実行し、すべての関数呼び出しを関数呼び出しの戻り値と置き換えます。変数参照と関数呼び出しは、それらが指定された順に処理されます。 このステップ中は、Net.Data は、組み込み関数あるいは MACRO_FUNCTION ブロックを処理しません。
2. ネイティブの言語プロセッサーは、実行可能ステートメントのセクションを処理する。 FUNCTION ブロックの場合、プロセッサーは、SQL、REXX、あるいは Perl などの FUNCTION ブロックで指定された言語環境に対応します。 MACRO_FUNCTION ブロックの場合、Net.Data は、言語プロセッサーとして働き、実行可能ステートメントを実行します。 組み込み関数は、実行可能ステートメントを持ちません。 Net.Data は、関数名で組み込み関数を処理します。

   Net.Data は、関数のパラメーターを、ネイティブの言語プロセッサーに渡します。 Net.Data は、IN あるいは INOUT の場合にのみ、ネイティブ言語プロセッサーに値を渡し、OUT あるいは INOUT の場合にのみ、ネイティブの言語プロセッサーから戻り値を受け取ります。
3. Net.Data は、言語プロセッサーからの戻りコードと言語プロセッサーから戻されたメッセージに基づき、暗黙の RETURN_CODE および DTW_DEFAULT_MESSAGE 変数を設定する。 Net.Data は、これらの変数を、MACRO_FUNCTION ブロックに対しては設定しません。
4. FUNCTION ブロックは、 REPORT ブロックを含む場合、あるいはデフォルトのレポートを生成するように指定する場合、Net.Data は、任意の参照出力パラメーターを使用して、レポート・セクションを処理する。 Net.Data は、組み込み関数に対してはレポートを生成しません。
5. FUNCTION ブロックが、ローカルな MESSAGE ブロックを含む場合、Net.Data は、その MESSAGE ブロックを処理する。Net.Data は、以下の条件のどれか 1 つが発生した場合、グローバルな MESSAGE ブロックを処理します。
   - グローバルな MESSAGE ブロックは指定されるが、戻りコードは、ローカルな MESSAGE ブロックではハンドルされない。
   - 組み込み関数が呼び出される。

   Net.Data は、MESSAGE ブロックあるいは MACRO_FUNCTION ブロックを処理しません。
6. Net.Data は、関数呼び出しを、関数の戻り値で置き換える。 FUNCTION ブロックの場合、この値は以下のどれか 1 つになります。

   RETURNS パラメーター値
   RETURNS キーワードを持つ FUNCTION ブロックに置き換えられます。

   空ストリング ("")
   RETURN_CODE がゼロの場合、RETURNS キーワードを持たない FUNCTION ブロックに置き換えられます。

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

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

組み込み関数の場合、その値は、組み込み関数のフォーマットに依存します。

ストアード・プロシージャーの呼び出し

ストアード・プロシージャーは、コンパイル済みのプログラムで、DB2 のローカルあるいはリモート・サーバーに格納され、SQL ステートメントを実行することができます。 Net.Data では、ストアード・プロシージャーは、 CALL SQL ステートメントを使用して、Net.Data の関数から呼び出されます。 ストアード・プロシージャーのパラメーターは、Net.Data の関数パラメーター・リストから渡されます。ストアード・プロシージャーを使用すると、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。

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。

- 88ページの『ストアード・プロシージャーの構文』
- 89ページの『ストアード・プロシージャーの呼び出し』
- 90ページの『パラメーターを渡す』
- 90ページの『結果セットの処理』

ストアード・プロシージャーの構文

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

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

ここで、

function_name
ストアード・プロシージャーの呼び出しを開始する Net.Data 関数の名前

stored_procedure
ストアード・プロシージャーの名前

datatype
89ページの表 5 で示すような、 Net.Data がサポートするデータベースのデータ型。パラメーター・リスト内で指定されるデータ型は、ストアード・プロシージャー内のデータ型と一致しなければなりません。 これらのデータ型に関するさらに詳しい情報については、データベースの文書を参照してください。

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

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

表 5. ストアード・プロシージャーのデータ型

BIGINT	FLOAT	SMALLINT
CHAR	INTEGER	TIME
DATE	GRAPHIC	TIMESTAMP
DECIMAL	LONGVARCHAR	VARCHAR
DOUBLE	LONGVARGRAPHIC	VARGRAPHIC
DOUBLEPRECISION		

ストアード・プロシージャーの呼び出し

ストアード・プロシージャーを呼び出すには、以下のようにします。

1. ストアード・プロシージャーへの呼び出しを開始する関数を定義する。

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

2. 任意選択で、Net.Data 表内に結果セットを保管するための表名を含む、ストアード・プロシージャーに対して、IN、INOUT、または OUT パラメーターを指定する (Net.Data 表を指定する必要があるのは、結果セットを Net.Data 表に保管したい場合だけです)。

   ```
   %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・プロシージャー名を確認します。

   ```
   CALL stored_procedure
   ```

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

   ```
   %REPORT
   ```

   例 :

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

5. ストアード・プロシージャーが 1 つ以上の結果セットを生成中である場合には、以下のようにします。
   - CALL ステートメントで、1 つ以上の結果セット名を指定する。

     ```
     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する。

  ```
  %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

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

例 1 : ストアード・プロシージャーにパラメーター値を渡す

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

例 2: ストアード・プロシージャーから値を戻す

```
%function (dtw_sql) mystoredproc (OUT VARCHAR(9) retvalue)  {
  CALL myproc
...
```

結果セットの処理

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

- 結果セットを、結果セットのレポート・ブロックを定義しないで、 Net.Data のデフォルトのレポート・スタイルで表示する。
- 結果セットを REPORT ブロックに関連付け、Net.Data がレポートに結果セットを表示するようにする。次に、Net.Data 変数、HTML、またはほかの関数を使用して、レポート・データをブラウザーで表示する方法を指定することができます。

- 後で、Net.Data がマクロ・ファイル内のデータを使用するようにしたい場合には、Net.Data 表に結果セットを保管する。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。

ストアード・プロシージャーが複数の結果セットを生成する場合には、 CALL SQL ステートメントの各結果セットにパラメーターを指定しなければなりません。

単一の結果セットを戻し、それをデフォルトのレポートを使用して表示するには、以下のようにします。

以下の構文を使用します。

```
%function (dtw_sql) function_name () {
    CALL stored_procedure ()
%}
```

たとえば、以下のようにします。

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

単一の結果セットを戻し、表示処理に対して REPORT ブロックを指定するには、以下のようにします。

以下の構文を使用します。

```
%function (dtw_sql) function_name () {
     CALL stored_procedure
 %REPORT (resultsetname) {
  ...
 %}
%}
```

たとえば、以下のようにします。

```
%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、以下のようにします。

```
%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 (tablename)
%}
```

たとえば、以下のようにします。

```
%define DTW_DEFAULT_REPORT = "NO"

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

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 (tablename1, tablename2)
%}
```

たとえば、以下のようにします。

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

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ようにするには、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、以下のようにします。

```
%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。

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

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

たとえば、以下のようにします。

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

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

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

複数の結果セットを戻し、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、以下のようにします。

```
%function (dtw_sql) mystoredproc (OUT mytable2) {
    CALL myproc (mytable1, mytable2, mytable3)

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

結果セット mytable1 は、それに対応する REPORT ブロックが処理し、マクロ・ライターが指定するように表示されます。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。

複数の結果セットを戻すためのガイドラインおよび制約事項

ストアード・プロシージャーから複数に結果セットを戻す場合には、以下のガイドラインおよび制約事項を使用します。

ガイドライン：

- 結果セットごとに 1 つの REPORT ブロックを指定する。 REPORT ブロックに指定された表名パラメーターは、 CALL ステートメントの対応する表名パラメーターと一致しなければなりません。
- 結果セットを処理したい順序で、複数の結果セットに REPORT ブロックを指定する。
- 結果セットに REPORT ブロックが指定されていないときにデフォルトの処理を指定するには、 DTW_DEFAULT_REPORT = ″YES″ を定義する。 Net.Data が Web ページを作成すると、そのページには、 REPORT ブロックをもつ結果セットのリポートを表示後、結果セットのデフォルトのリポートが表示されます。
- Net.Data が、REPORT ブロックをもたない結果セットを表示しないようにするには、 DTW_DEFAULT_REPORT = ″NO″ を設定する。
- ストアード・プロシージャーで DTW_SET_TABLE_IN 変数を使用するときには、ストアード・プロシージャーから戻される最初の結果セットを、DTW_SET_TABLE_IN 表に割り当てる。

制約事項：

- DTW_SQL DB2 言語環境のストアード・プロシージャーにおいてのみ、複数の REPORT ブロックを使用できます。
- リポート変数は、全関数に設定され、すべての REPORT ブロックの処理とその変数が処理する結果セットに影響を及ぼします。個々の REPORT ブロックのリ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 ブロックの間でではありません。
- 表変数は、ストアード・プロシージャーで使用する前に、 TABLE ステートメントで定義しなければなりません。
- 複数の REPORT ブロックを同じ結果セットに指定することはできません。
- 1 つのストアード・プロシージャーの結果セットの最大数は 32 です。

メッセージ・ブロック

MESSAGE ブロックにより、関数呼び出しの成功あるいは失敗を基にして、関数呼び出し後の進め方を決定することができ、関数の呼び出し側に情報を表示することができます。 Net.Data は、以下のメッセージ・ブロック処理を使用します。

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

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

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

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

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

以下の例は、グローバルな MESSAGE ブロックと、関数の MESSAGE ブロックを持つ Net.Data のマクロのパーツを示しています。

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

%{ local message block inside a FUNCTION block %}
%FUNCTION(DTW_REXX) my_function() {
  %EXEC { my_command.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
```

マクロに HTML を作成

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

HTML ブロック

Net.Data のマクロ・ファイルには、HTML ブロックおよび Web ブラウザーへの HTML 出力を生成する、HTML ブロックの関数が含まれます。マクロ・ファイルでは、最低でも 1 つの HTML ブロックを指定しなければなりません。HTML ブロックの内容は、それ以降の Net.Data の起動を制御します。

有効な HTML ステートメントならどれでも、HTML ブロックに表示することができます。さらに、INCLUDE ステートメント、関数呼び出し、および HTML ブロックの変数参照を使用することができます。 以下の例は、Net.Data のマクロにおける HTML ブロックの一般的な使われ方を示しています 。

```
%DEFINE DATABASE="MNS96"

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

```
<dd><input type="radio" name="hdware" value="SCN">Scanners
</dl>
<HR>
<input type="submit" value="Submit">
</FORM>
%}

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

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

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

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

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

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

Net.Data は次に、myQuery() 関数呼び出しを処理します。この関数呼び出しは、次に SQL FUNCTION ブロックを起動します。 SQL ステートメントの $(hdware) 変数参照を、入力フォームで戻された値と置き換えた後、Net.Data は照会を実行します。この時点で、Net.Data は再度 HTML レポートのブラウザーへの送信を開始し、 REPORT ブロックで指定された HTML ステートメントに従って、照会結果を表示します。

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

Net.Data は、起動されるごとに、1 つのHTMLだけを処理します。しかし、HTML のリンクとフォームを使用することにより、他の人が別の HTML ブロックで、簡単にもう 1 つの Net.Data の起動を開始することができます。すべては、ユーザーにより制御されます。

レポート・ブロック

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

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

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

ROW ブロックからの表出力はどれも表示をしないようにするには、ROW ブロックを空にしておくか、ROW ブロックを完全に省略します。

Net.Data により提供される REPORT ブロック内の幾つかの変数を使用して、Net.Data のマクロの結果表のデータにアクセスすることができます。これらの変数は、80ページの『表処理変数』 で説明されています。追加詳細については、*Net.Data 解説* のレポート変数のセクションを参照してください。

ヘッダーおよびフッター情報を提供するには、ROW ブロックの前後にテキストを提供します。 Net.Data は、ROW ブロックの前で検出したものはすべて処理し、それをヘッダー情報として扱います。そして、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扱います。 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、HTML として扱い、HTML データをブラウザーに送信します。

REPORT ブロックの関数および変数を使用することもできます。

Net.Data に、フォーマット済みのテキストを使用してデフォルトのレポートをプリントさせるには、マクロ・ファイルに REPORT ブロックを組み込まないようにします。以下の例は、デフォルトのレポート・フォーマットを示しています。

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

HTML のタグをフォーマット済みテキストの代わりに使用するには、DTW_HTML_TABLE を YES に設定します。

デフォルトのレポートのプリントを使用禁止にするには、DTW_DEFAULT_REPORT を NO に設定するか、空の REPORT ブロックを指定します。たとえば、以下のようにします。

```
%REPORT{%}
```

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

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

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

Phone Query Results:
====================
Name: **Doen, David**
Phone: 422-245-1293
Fax: 422-245-7383

Name: **Ramirez, Paolo**
Phone: 955-768-3489
Fax: 955-768-3974

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

Total records retrieved: 3

Net.Data は、以下を行い、レポートを生成しました。

1. *Phone Query Results:* を、レポートの最初に 1 回プリントする。
2. 変数 V1、V2、そして V3 に、Name、Phone、および Fax の値をそれぞれ、検索時に各行ごとに与える。
3. 各取得行の後ろに罫線を引き、読みやすくする。
4. ストリング *Total records retrieved:* および NUM_ROWS の値を、レポートの最後に 1 回だけプリントする。

ストアード・プロシージャーに複数のリポート・ブロックを使用して、複数の結果セットを戻すことができます。 DTW_SQL 言語環境を使用するときに、複数のリポート・ブロックをサポートするのは、ストアード・プロシージャーに対してだけです。ストアード・プロシージャーから複数の結果セットを戻す方法については、 88ページの『ストアード・プロシージャーの呼び出し』 を参照してください。

マクロ・ファイルにおける条件付き論理とループ

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

- 100ページの『条件付き論理』
- 102ページの『ループ構成体』

条件付き論理

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

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

IF ブロック構文の規則は、マクロ・ファイルでのブロックの位置によって決定します。 IF ブロックの実行可能なステートメント・ブロックに許される要素は、IF ブロック自身の位置に依存します。 IF ブロックを含むブロック内で有効な要素ならどれでも、その IF ブロックで有効です。 たとえば、IF ブロックを HTML ブロック内で指定する場合、 HTML ブロックに許可される要素はどれでも、 INCLUDE ステートメントおよび WHILE ブロックなどの IF ブロックで許可されます。

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

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

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

IF ブロックは、宣言パーツ内のほかのブロックの外側にある IF ブロック内でネストされる場合には、外側のブロックが使用できるどんな要素でも使用することができます。 IF block は、IF ブロック内にある別のブロック内でネストされる場合には、内側にあるブロックの構文規則を採用します。

以下の例では、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 block
...
     %IF block
```

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

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

- 条件がバイナリー操作 (<、>、<=、>=、!=、==) の場合。
- 条件に含まれる項が共に整数を表している場合。この意味は、項は数字からなるストリングで、オプションとして、その前に、'+' あるいは '-' の文字がくるということです。ストリングは、'+' あるいは '-'以外の非数字文字を含むことができません。

  有効なストリングの例 :

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

  無効なストリングの例 :

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

制約事項 : Net.Data は、非整数の数字値の比較をサポートしません。

例 : 次の例は、他のブロックに、IF ブロックを含むマクロ・ファイルを示しています。

```
%{ 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 HTML の IF ブロック内で指定し、それらをネストすることができます。 WHILE ブロックの構文は、*Net.Data 解説書* の言語構成要素の章に示されています。

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

例 : 次の例は、WHILE ブロックを持つマクロ・ファイルを示しています。

```
%DEFINE loopCounter = "1"

%HTML(build_table) {
%WHILE (loopCounter <= "100") {
  %{ generate table tag and column headings %}
  %IF (loopCounter == "1")
    <TABLE BORDER>
    <TR>
    <TH>Item #
    <TH>説明
%ENDIF

  %{ generate individual rows %}
  <TR>
  <TD>$(loopCounter)
```

```
  <TD>@getDescription(loopCounter)

  %{ generate end table tag %}
  %IF (loopCounter == "100")
%ENDIF

  %{ increment loop counter %}
  @dtw_add(loopCounter, "1", loopCounter)
%}
%}
```

ラージ・オブジェクトを使用する

ラージ・マルチメディア・オブジェクトおよび文書オブジェクトは、表示または印刷に使用する BMP、TIF、GIF、および PostScript ファイルなどのバイナリー・ファイルです。 これらのファイルを、ほとんどのオペレーティング・システムの DB2 データベースに保管し、その後、ご使用の Web アプリケーションの SQL 言語環境を通してそのファイルにアクセスすることができます。

Net.Data は以下の LOB の以下の型のオブジェクトをサポートします。

- バイナリー・ラージ・オブジェクト (BLOB)
- 文字ラージ・オブジェクト (CLOB)
- 2 バイト・ラージ・オブジェクト (DBLOB)

一般的に、LOB の最初の数バイトにはファイル識別記号が含まれ、そのファイルに含まれる情報の型を示しています。 Net.Data は LOB を認知すると、一時ファイルと、その名前を表す Net.Data マクロ変数に拡張子を追加します。マクロ・ファイル内に REPORT ブロックがない場合には、 Net.Data は .txt 拡張子を CLOB ファイルに追加します。 Net.Data は以下の LOB 形式を認知します。

- ビットマップ (.bmp)
- 図形による画像形式 (.gif)
- Tagged Image File Format (.tif)
- ポストスクリプト (.ps) これらは CLOB ではなく BLOB として保管しなければなりません。

制約事項 : Net.Data はほかのファイル型を認知せず、サポートしません。 さらに、Net.Data は、どんなラージ・オブジェクトの SQL ステートメントの UPDATE および INSERT も行いません。

プランのためのヒント : 照会が LOB を戻すと、Net.Data はそれを、 HTML_PATH 構成変数で指定されたディレクトリーに保管します。 LOB を使用する場合、システム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。音声ファイルのように、LOB の中には、特定のハードウェアとソフトウェアを必要とするものがあります。

例 1: ピクチャー・インラインを表示します。

```
<IMG SRC="/tmplobs/filename">
<A HREF="/tmplobs/filename">filename</A>
```

例 2: 以下の例では、アプリケーション・ユーザーはファイル名をクリックして、ビューアーを呼び出さなければなりません。その理由は、アプリケーションは .WAV ファイルを使用するからです。 Net.Data はこのファイル型をサポートしません。そのため、EXEC 変数を使用して、そのファイルに拡張子を追加します。

```
%DEFINE{
docroot="/usr/lpp/internet/server_root/html"
rename=%EXEC "rename $(docroot)$(V3) $(docroot)$(V3).wav"
%}

%FUNCTION(DTW_SQL) queryData() {
SELECT Name, IDPhoto, Voice FROM RepProfile
%REPORT{
<P>Here are the images you selected:<P>
%ROW{
$(rename)
$(V1) Voice sample <IMG SRC="$(V2)">
<A HREF="$(V3.wav")>Voice sample</A><P>
%}
%}
%}

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

queryData 関数は以下の HTML を戻します。

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

直前の例の REPORT ブロックは、暗黙の表変数 V1、V2、および V3 を使用します。

- V1 は人物の名前で、平文テキストです。
- V2 は .GIF ファイル内の人物の写真です。画像は、インラインで表示されます。Net.Data には、一時的 LOB ディレクトリーと .GIF 拡張子を自動的に組み込まれます。
- V3 は .WAV ファイル内の人物のボイスの例です。 Net.Data は、.WAV ファイルなどの未認知のファイル形式に出会うと、そのファイルを、一時的 LOB ディレクトリーに拡張子なしで書き込みます。この例は、EXEC 変数を使用して拡張子を追加することによって、このファイル型を処理する方法を示しています。変数 *$(V3)* が解決されると、ファイル名の前にパス /tmplobs/ が追加されます。たとえば、/tmplobs/sound2a です。このようなファイルを処理する 1 つの方法は、 REXX プログラムを書き込み、斜線を円記号に変更して、ファイルをリネームすることです。アプリケーション・ユーザーがボイスの例をクリックすると、ボイスの例が再生されます。

すべての Web ブラウザーが図形処理および音をサポートする訳ではありません。ここで説明した機能をサポートするために、音カードやドライバーなどの特定のハードウェアとソフトウェアが必要となる可能性があります。

第6章 組み込み関数の使用

Net.Data は、Web ページ開発を容易にするための大きな組み込み関数のセットを提供します。これらの関数は、すでに Net.Data により定義されているので、FUNCTION ブロックではそれらの関数を定義する必要はありません。ユーザー定義の関数を呼び出すことができるところなら、マクロのどこからでもこれらの関数を簡単に呼び出すことができます。

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

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

組み込み関数によっては、タイプを持たないものがあります。詳細は、*Net.Data 解説書* を参照してください。

以下のリストでは、Net.Data の組み込み関数について高度な概説を提供します。以下の関数を使用して、汎用、数学、ストリング、ワード、あるいは表操作の各関数を実行します。 各関数の構文および例の説明については、*Net.Data 解説書* を参照してください。

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たちます。これらの関数を使用して、照会、環境変数の設定、HTML のエスケープ・コードの使用、およびシステムからの有益な情報の取得、を行うことができます。

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。

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

ワード関数
これらの関数により、ストリング内のワードを操作することができます。これらの関数のほとんどは、ストリング関数と同じ働きをします。ただし、ワード全体に対して働きます。たとえば、これらの関数

を使用して、ストリング内のワード数のカウント、ワードの削除、ストリングからのワードの取得、などを実行することができます。

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。表変数には、値の配列、およびそれらに関連付けられている列の名前が含まれます。これらの関数は、値のグループを関数に渡すのに都合の良い方法を提供してくれます。

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。

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。

第7章 言語環境の使用

Net.Data は言語環境を提供します。これによって、ユーザーはデータ・ソースをアクセスし、ビジネス論理を持つアプリケーション・プログラムを実行します。たとえば、SQL 言語環境によって、ユーザーは SQL ステートメントを DB2 データベースに渡すことが可能になり、 REXX 言語環境によって、ユーザーは REXX プログラムを起動できるようになります。 ユーザーは、さらに SYSTEM 言語環境を使用して、プログラムを実行するか、あるいはコマンドを発行します。

Net.Data によって、ユーザー作成の言語環境をプラグイン方式で追加することができます。 それぞれのユーザー作成言語環境は、 Net.Data によって定義された標準的なインターフェースのセットを支援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 として設定されていなければなりません。 ENVIRONMENT ステートメントを Net.Data 初期設定ファイルに追加して、 DLL をユーザーが作成する言語環境に関連付けなければなりません。Net.Data は、FUNCTION ブロックへの関数呼び出しが最初に検出されると、 DLL をロードして実行します。FUNCTION ブロックは言語環境名を指定します。 同じ言語環境名を指定する FUNCTION ブロックへのこれ以降の関数呼び出しでは、 DLL のロードは 1 度しか行われないため、Net.Data は DLL を実行するだけになります。

107ページの図 21 では、Web サーバー、Net.Data、および Net.Data 言語環境間の関連を表示しています。

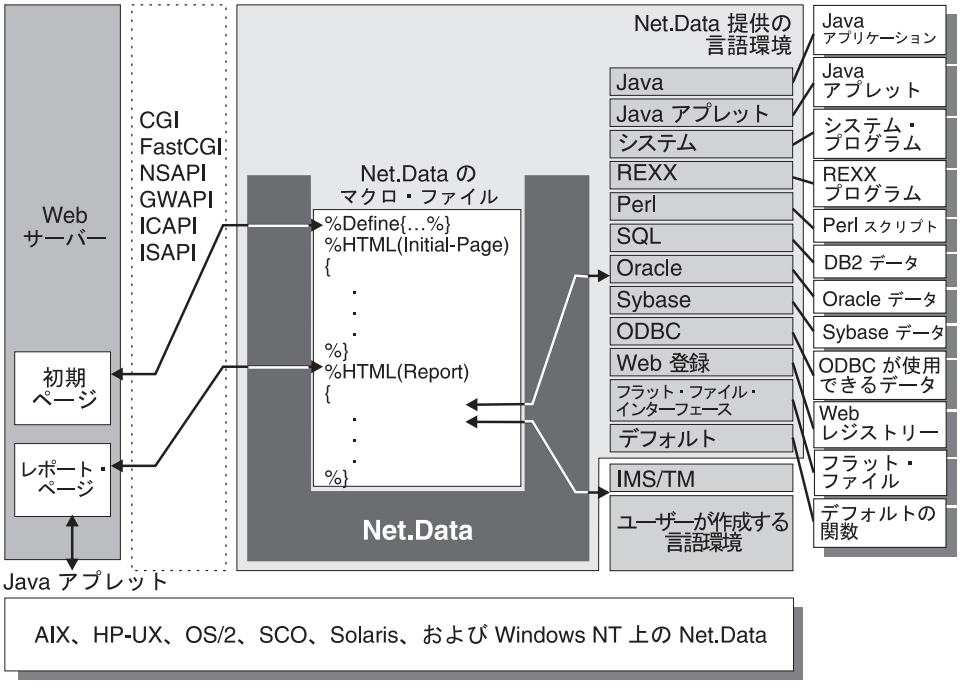


図 21. Net.Data 言語環境

Net.Data 提供の言語環境および言語環境のユーザー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。

第8章 Java Servlets および JavaBeans とともに Net.Data を起動する

Net.Data には Java ベースのクラスがあり、このクラスを使用して Net.Data マクロ・ファイルの起動、Net.Data 関数の実行、または Net.Data が支援する Java が使用可能なオペレーティング・システム内の Net.Data を通した、SQL ステートメントの実行を行います。

この章では、次の概念と操作を説明しています。

Net.Data サーブレット

Net.Data は、サーブレットおよび NetObjects Fusion プラグインを、 Java 環境で使用できる Net.Data とともに提供します。サー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・ファイルを実行します。
- URL からかつ SSI として、Net.Data 関数を実行します。
- NetObjects Fusion (NOF) を使用して、マクロ・ファイルを管理します。これによって、 Web サイト・マネージメントとすぐれた統合性を持ち、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、161ページの『付録C. 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

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 API プラグインと同じような役割を実行する Java クラスです。 Java のサーブレットが使用できる 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。サーブレットには独自の GUI がありません。ただし、サーブレットのクラスは、ローカルで、または、ネットワークから動的にロードするか、あるいは、 URL アドレスを使用して (リモート) 、またはクラス名で (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・ファイル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・ファイル (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データベースでない言語環境プログラムを実行できるほか、 Live コネクションと一緒に実行できます。

サーブレットには、ユーザーのアプリケーションで使用する 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 のマクロ・ファイルを実行します。

マクロ・サーブレット を使用することは、 Java サーブレット を通してマクロ・ファイルを実行する場合を除いて、 CGI-BIN インターフェースを通して Net.Data マクロ・ファイル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 照会は、パフォーマンスの向上のために、Live コネクション管理プログラムを使用する ODBC を介して実行される。
- Server-Side-Includes (SSI) を介してマクロ・ファイルを実行し、複数のマクロをユーザーの HTML ファイルに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、および Java のようないろいろな言語環境プログラムに加えて、 DB2、Oracle、および Sybase 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 の関数または SQL ステートメントを実行します。詳しくは、59ページの『マクロ・ファイルを使用しない 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。

サーブレットをセットアップする

サーブレットの命令、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 のサーブレットは、 Net.Data の *<inst_dir>*/servlets/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>*/servlets/NetDataServlets.jar を、ユーザーの CLASSPATH 環境変数に追加する必要があるかもしれません。

Net.Data サーブレットを実行する

Net.Data のサーブレットは、URL からか、または HTML ファイル内の SSI としての、いずれかで実行することができます。 NetObjects Fusion プラグインを使用して、Net.Data サーブレットをご使用の NOF サイトにと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で入力し、サーブレットを変更および実行する方法について説明します。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、 161ページの『付録C. 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。

- 111ページの『マクロ・サーブレットを実行する』
- 113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
```

   たとえば、以下のようにします。

```
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>
```

   たとえば、以下のようにします。

```
<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・ファイルへの完全修飾パス

block_value
指定された Net.Data マクロ・ファイル内の実行される HTML ブロックの名前。デフォルトは `report` (オプション) です。

parmnn
ご使用のマクロ・ファイルに必要な追加のパラメーターで、形式は以下のようになります。

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

valuenn

ご使用のマクロ・ファイルに必要な追加の値で、形式は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&parmnn=valuenn
```

たとえば、以下のようにします。

```
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="parmnn" value="valuenn">
 </servlet>
```

たとえば、以下のようにします。

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

INBUFLEN および OUTBUFLEN パラメーター : マクロ・ファイルへのユーザーの入力データが 1 KB より大きい場合には、 INBUFLEN パラメーターを指定しなければなりません。ユーザーのマクロ・ファイルの結果が 32 KB より大きい場合には、OUTBUFLEN パラメーターを指定しなければなりません。必要なパラメーターの指定に失敗すると、予想外の結果になります。

- URL:

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

たとえば、以下のようにします。

```
http://myserver/servlet/com.ibm.netdata.servlets.MacroServlet?MACRO=my_macro
  &BLOCK=my_blockINBUFLEN=3K&OUTBUFLEN=48K&field1=custno
```

- SSI:

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

たとえば、以下のようにします。

```
<servlet code="com.ibm.netdata.servlets.MacroServlet">
<param name="MACRO" value="my_macro">
<param name="BLOCK" value="my_block">
<param name="INBUFLEN" value="3K">
<param name="OUTBUFLEN" value="48K">
<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、関数 (REXX 関数など) または SQL ステートメントのどちらかを実行する直接要求を使用して、 Net.Data を呼び出すことができます。ユーザーがサーブレット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 :
   - ***関数を呼び出すには :***

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

     たとえば、以下のようにします。

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

   - ***SQL ステートメントを呼び出すには :***

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

     たとえば、以下のようにします。

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

2. SSI:
   - ***関数を呼び出すには :***

     ```
     <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>
     ```

     たとえば、以下のようにします。

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

   - ***SQL ステートメントを呼び出すには :***

     ```
     <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>
     ```

     たとえば、以下のようにします。

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
<param name="LANGENV" value="DTW_SQL">
<param name="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・プロシージャー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。 *parmnn* 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ード・プロシージャー。たとえば、"select * from employee"。*parmnn* 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ード・プロシージャーの入力パラメーターを指定します。

parmnn
その他ユーザーのマクロ・ファイルに必要とされるパラメーター。形式は、<param name="field1" ... となります。

valuenn
その他ユーザーのマクロ・ファイルに必要とされる値。形式は、 ... value="custnum" となります。

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.FunctionServlet
?LANGENV=lang_env_name&FUNC=program_name&htmlpath=html_path
&parmnn=valuenn
```

たとえば、以下のようにします。

```
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="parmnn" value="valuenn">
 </servlet>
```

たとえば、以下のようにします。

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
<param name="LANGENV" value="DTW_SQL">
<param name="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。たとえば `htmlpath=e:¥html` 。

INBUFLEN および OUTBUFLEN パラメーター： マクロ・ファイルへのユーザーの入力データが 1 KB より大きい場合には、 INBUFLEN パラメーターを指定しなければなりません。ユーザーのマクロ・ファイルの結果が 32 KB より大きい場合には、OUTBUFLEN パラメーターを指定しなければなりません。必要なパラメーターの指定に失敗すると、予想外の結果になります。

- URL:

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

  たとえば、以下のようにします。

  ```
  http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet?LANGENV=DTW_REXX
    &FUNC=my_rexxINBUFLEN=3K&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="INBUFLEN" value="input_buffer_size">
  <param name="OUTBUFLEN" value="output_buffer_size">
  <param name="parmnn" value="valuenn">
   </servlet>
  ```

  たとえば、以下のようにします。

  ```
  <servlet code="com.ibm.netdata.servlets.FunctionServlet">
  <param name="LANGENV" value="DTW_REXX">
  <param name="FUNC" value="my_rexx">
  <param name="INBUFLEN" value="3K">
  <param name="OUTBUFLEN" value="48K">
  <param name="field1" value="custno">
   </servlet>
  ```

Net.Data JavaBeans

Net.Data 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る JavaBeans を提供します。 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。

ネイティブ Net.Data DLL を使用して、JavaBean は Net.Data を起動して、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はネイティブを使用するため、 Net.Data の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。

設計のためのヒント： Net.Data の JavaBeans が戻す結果は、ユーザーのマクロ・ファイルまたは関数が何を戻しても、一般的には 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 では、以下のことを行うことができます。

- Net.Data マクロ・ファイルを実行する
- 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。

- 116ページの『Net.Data JavaBeans について』
- 116ページの『Net.Data JavaBean のセットアップと実行』

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・ファイルまたは 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 JavaBean のセットアップと実行

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。

- 117ページの『Macro Bean をセットアップする』
- 117ページの『マクロ Bean を実行する』

- 117ページの『SQL Bean をセットアップする』
- 118ページの『SQL Bean を実行する』

Macro Bean をセットアップする

Net.Data Macro bean、com.ibm.netdata.beans.NetDataMacro によって、 Java を使用して、既存のマクロ・ファイル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 プロパティーを指定する必要があります。指定するとマクロ・ファイル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、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ーを設定する。

   マクロ 実行する既存のマクロ・ファイル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
   ```

マクロ Bean を実行する

JavaBean 開発ツールとともに Net.Data マクロ JavaBean を実行するには:

- 「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。
- マクロが実行されると、ユーザーは次の出力プロパティーを参照できます。

  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・ファイルの実行によって戻されたデータを指定します。

SQL Bean をセットアップする

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 プロパティーを指定する必要があります。指定するとマクロ・ファイル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. 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します。
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ーを設定する。

   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
   ```

SQL Bean を実行する

JavaBean 開発ツールとともに Net.Data マクロ JavaBean を実行するには:

- 「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。
- 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ーを参照できます。

  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たデータを指定します。

第9章 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たちます。 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。

Web 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 のプロキシー・サーバーのキャッシュは、 Web ページと関連するオブジェクトを、ユーザーのグループの近くにあるローカル・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する。たとえば、Web サーバーが要求した項目を検索する時間を削減します。 Web のプロキシー・サーバーはまた、複数のユーザーの間で、普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てくれます。
- 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、プロキシー・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を使用することができます。**勧告 :** Net.Data のたくさんのインスタンスには 1 つのキャッシュ管理プログラムを、また、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てください。

120ページの図 22 は、Net.Data がキャッシュ管理プログラムを使用して、マクロ・ファイルからの HTML 出力のキャッシュを管理することを示しています。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。

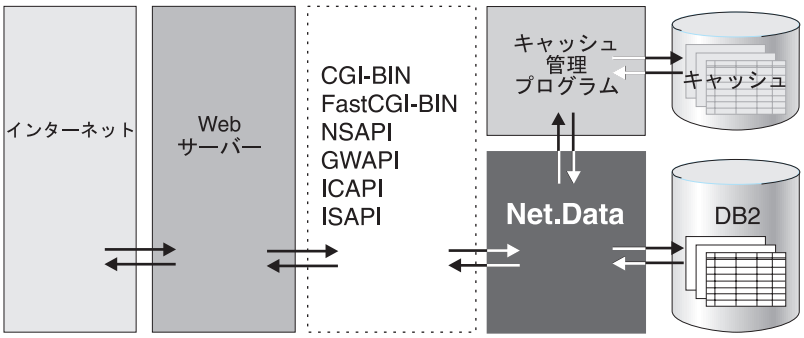


図 22. Net.Data のキャッシング

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ーの型。同一データに続けてアクセスする場合の速度を上げることが目的。キャッシュは、ネットワークを介してアクセス可能な、使用頻度の高いデータのローカル・コピーを保留するのに使用される場合が多い。 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することができ、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。 121ページの図 23 は、1 つのキャッシュ管理プログラムが複数のマクロ・ファイルを取り扱い、2 つのキャッシュを管理しているところを示しています。

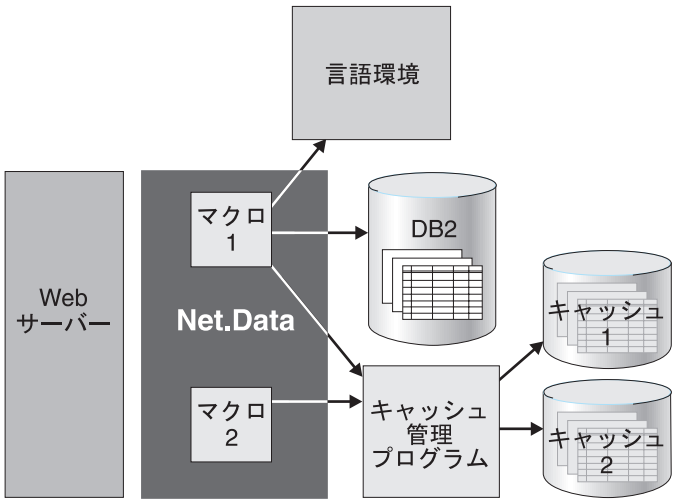


図 23. 複数のマクロ・ファイルおよびキャッシュを扱うキャッシュ管理プログラム

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。それぞれのキャッシュ・ページには、一意的な識別子、たとえば URL 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 (たとえば、組み込み関数 DTW_CACHE_PAGE) を受信すると、以下のステップを実行します。

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いるが満了していない場合には、 Net.Data は、キャッシュ管理プログラムからのページを要求し、そのページをブラウザーに送信し、マクロの実行を停止する。
   - データがキャッシュされていない場合には、 Net.Data はマクロの処理を続行し、生成済みの HTML ページを Web ブラウザーと、データがキャッシュされるキャッシュ管理プログラムに送信します。
3. Net.Data がキャッシュ管理プログラムから切断される。

マクロ・ファイルが正常に処理を完了すると、キャッシュ管理プログラムは、 HTML 出力をキャッシュし、生成済みの Web ページがキャッシュされていることを確認します。 データがキャッシュされるのは、そのデータがブラウザーに送信されてからで、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です。

Net.Data でエラーが発生するか、あるいは、マクロを早々と終了すると、キャッシュ管理プログラムは、

- 一部分のページまたは不完全なページを受け入れません。
- キャッシュ内に既存のページを保持します。

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。 123 ページの表 6 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 6. *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	行き先 ...
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。	125ページの『キャッシュ管理プログラムの定義』
キャッシュ構成のオプション	Net.Data のキャッシュ管理プログラムの単一インスタンス内では、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。	127ページの『定義、キャッシュの』
Net.Data の初期設定オプション	Net.Data 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、Net.Data の初期設定ファイルで指定します。	13ページの『キャッシュ管理プログラムの構成変数』
Net.Data のキャッシュの組み込み関数	Net.Data のキャッシュの組み込み関数を使用して、Net.Data 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。	*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のためのプラン

Net.Data のキャッシュ関数を使用するためのプランを立てる場合は、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制作前に、アプリケーションのプランを立て、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。125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生産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。

キャッシュのエラー

Net.Data で内部エラーが発生すると、キャッシュ管理プログラムは Web ページをキャッシュしません。この内部エラーが原因で、処理が完了する前に、Net.Data がマクロ・ファイルと終了します。キャッシュ管理プログラムは、不完全なページをキャッシュしないか、あるいは、キャッシュ管理プログラムに Net.Data のエラーが含まれます。このような型のエラーには、マクロの構文エラーおよび SQL エラーが組み込まれています。

エラーがあるページは、以下の場合にキャッシュされます。

- Net.Data でエラーが発生し、メッセージ・ブロック内の CONTINUE 宣言のために、Net.Data がマクロ・ファイルの処理を続行し、正常に終了する。
- データベースのロールバックのように、エラーが、Net.Data のエラー判別効力範囲外で発生する。

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、アプリケーションでキャッシュに名前を付けることができます。Net.Data の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 の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。キャッシュされたページの識別コードは、 URL アドレスなどの任意のストリングでも構いません。
  DTW_CACHE_PAGE() 組み込み関数で識別子を指定します。 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、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の、 2 つの型のスタンザが含まれます。以下のステップでは、ユーザー・アプリケーションのために、これらの 2 つのスタンザをカスタマイズする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はありません。

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する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. やってくる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。 このポート番号を使用するのは、相手側機械からキャッシュ管理プログラムに接続する場合だけです。

   この値は、Net.Data の初期設定ファイルの 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
すべての追跡フラグを使用不可にします。

例 : すべての追跡フラグを指定する追跡フラグが使用可能です。

```
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 {
log=/u/cached/logs/cached.log
port=7177
connection-timeout=0
logging=yes
wrap-log=yes
log-size=64000
trace-flags=D_ALL
}
```

定義、キャッシュの

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ンザを定義します。ほとんどのキーワードは任意選択で、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

   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。

   不要 ; 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っ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越えてしまうと、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。

   不要 ; 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、

   *nn***B** バイト数です。たとえば、5000B とします。

   *nn***KB** Kバイト数です。たとえば、640KB とします。

   *nn***MB** 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ーの最大量を指定する。メモリー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ーの合計を制限内に持っ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にすることができます。しかし **cachemgrd** プロセスがメモリーを消費しすぎると、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。

   不要 ; デフォルトは、1MB です。

   構文 :

   ```
   mem-size=nnB|nnKB|nnMB
   ```

   ここで、

   *nn***B** バイト数です。たとえば、5000B とします。

   *nn***KB** Kバイト数です。たとえば、640KB とします。

   *nn***MB** 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。

   必要 : No; デフォルトは、5 分です。

   構文 :

```
lifetime=time_length
```

ここで、

*nn***S** 秒数です。たとえば 600S とします。

*nn***M** 分数です。たとえば 20M とします。

*nn***H** 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。

   不要 ; 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。

   *nn***S** 秒数です。たとえば 600S とします。

   *nn***M** 分数です。たとえば 20M とします。

   *nn***H** 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。

   不要 ; デフォルトは、1KB です。

   構文 :

   ```
   datum-memory-limit (cacheobj-memory-limit)=nnB|nnKB|nnMB
   ```

   ここで、

   *nn***B** バイト数です。たとえば、5000B とします。

   *nn***KB** Kバイト数です。たとえば、640KB とします。

   *nn***MB** 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。

   不要 ; デフォルトは、1KB です。

   構文 :

   ```
   datum-disk-limit (cacheobj-space-limit)=nnB|nnKB|nnMB
   ```

   ここで、

   ***nn*B** バイト数です。たとえば、5000B とします。

   ***nn*KB** Kバイト数です。たとえば、640KB とします。

   ***nn*MB** 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。

   不要 ; 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、

   ***nn*S** 秒数です。たとえば 600S とします。

   ***nn*M** 分数です。たとえば 1M とします。

   ***nn*H** 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要です。

    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。

    不要 ; 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；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おれば、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。

    不要；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。

    不要；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。

    不要；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

```
test1
{
caching=on
fssize=5MB
mem-size=10MB
```

```
lifetime=6000000
check-expiration=150
datum-memory-limit=5KB
datum-disk-limit=500KB
stat-interval=60
reset-stat-counters=no
root=/u/cached/chaches/cache0
stat-files=/u/cached/logs/chache0.stats
tran-log=/u/cached/logs/cache0.log
tran-logging=yes
wrap-tran-log=yes
tran-log-size=100k
}
```

キャッシュ管理プログラムの開始と停止

以下の節では、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解説します。

- 132ページの『キャッシュ管理プログラムの開始』
- 132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 :

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 :

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 がマクロ・ファイルで DTW_CACHE_PAGE 関数を見つけると、Net.Data はキャッシュ管理プログラムと交信し、マクロ・ファイルの HTML 出力をメモリーに保管しはじめます。 Net.Data が正常にマクロを処理した後、 HTML 出力がブラウザーに送信され、キャッシュ管理プログラムが、133ページの図 24 に示すように、 1 つの取引の出力をキャッシュします。

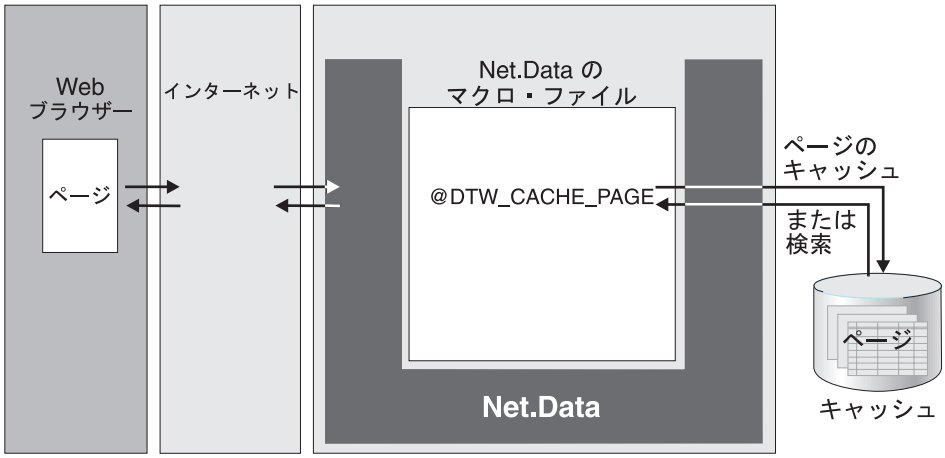


図 24. キャッシングを開始する DTW_CACHE_PAGE 関数

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、あるいは古くなっ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ーに送り戻し、マクロ実行から新規の出力ページを生成し、そのページをキャッシュに保管します。

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、それがまだ現行のページである場合には、キャッシュの内容が表示され、 Net.Data がマクロの外で終了します。この振る舞いによって、Web ページをキャッシュから検索した後に、不必要な処理が行なわれていないことを確認します。

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あるいは、最初のステートメントの 1 つとして配置し、マクロ・ファイルの実行コストを最小にします。

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 の符号化に先立ち、以下の関数ステートメントの 1 つを挿入します。

   ```
   @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の後のマクロから出されるすべての HTML 出力をキャッシュするように指定します。 すべての HTML 出力をキャッシュしたい場合には、このステートメントを、マクロ・ファイル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
   : たとえば、ページの URL のような、連続して発生した @DTW_CACHE_PAGE キャッシュ要求のキャッシュで、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。

   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 の値よりも短いか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 はマクロの実行を停止します。

   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>
 %}
```

拡張機能のキャッシング : キャッシュするかどうかを動的に判断する

DTW_CACHE_PAGE() 関数は、マクロ・ファイルでの位置からキャッシングを開始します。通常、パフォーマンスを向上させ、すべての HTML 出力がキャッシュされていることを確認するために、マクロ・ファイルの先頭に関数を置きます。

拡張機能のキャッシング・アプリケーションの場合、処理中の特定の点でキャッシュすることを決定する必要があるときには、マクロ・ファイルの先頭にではなく、HTML 出力セクションに DTW_CACHE_PAGE() 関数を置くことができます。たとえば、照会または関数呼び出しから戻される行数によって、キャッシングを決定する必要があるかもしれません。

例 : HTML 出力のよそうサイズに基づいてキャッシュを決定したため、 HTML ブロックに関数を置きます。

```
% 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、HTML 出力の予想サイズに基づいて、そのページをキャッシュするか、検索します。 HTML 出力ページがキャッシュに適していると判断されるのは、デー

タベース表に含まれる行が 100 以上であるときだけです。 Net.Data は常に OUTPUT ブロックのテキスト `This is the customer list` を、マクロの実行後、ブラウザーに送信します。テキストがキャッシュされることはありません。関数呼び出し `@count_rows()` の後の行は、 IF ブロックの条件が満たされているときに、キャッシュまたは検索されます。同時に、どちらのパーツも、完全な Net.Data 出力ページを形成します。

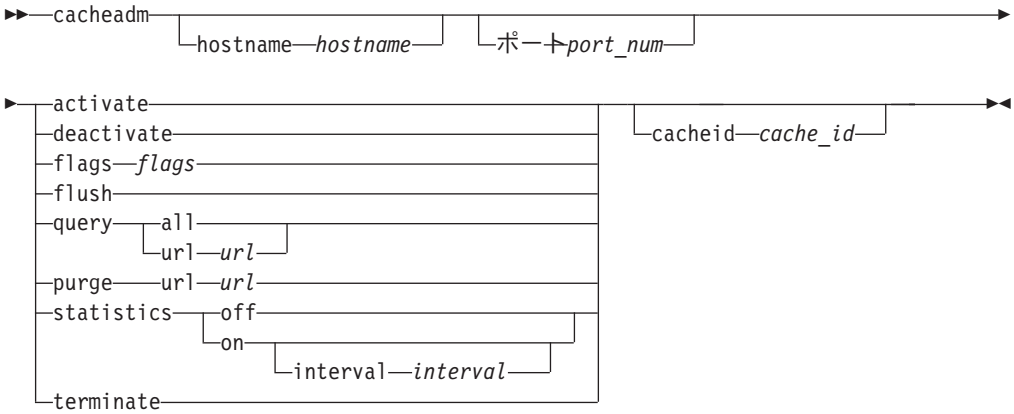
CACHEADM コマンド

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。このパラメーターは、指定されるキャッシュからのすべての項目を、無条件で削除します。

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パージし、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。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

内部操作に関する統計のいくつかの型を保持し、任意選択でキャッシュ・ログに書き込みます。各キャッシュの別のログを保守するか、あるいはすべての統計を同じログに書き込んでも構いません。このセクションでは、以下のキャッシュ・ログのトピックについて説明します。

- 138ページの『ログの構成』
- 139ページの『キャッシュ・ログの形式設定』

ログの構成

統計を記録するためには、キャッシュ管理プログラムの構成ファイルを構成しなければなりません。

ログを構成するには :

キャッシュ管理プログラムの構成ファイルで、 stat-files と stat-interval キーワードを指定します。

キャッシュ管理プログラムを停止、再構成、およびリスタートすることなく、統計の設定を変更することができます。

統計収集の設定を変更するには :

cacheadm statistics コマンドを指定します。 ただし、cacheadm statistics コマンドを使用して行なわれた変更は、キャッシュ管理プログラムをリスタートするときに、保管されないことに注意してください。

キャッシュ・ログの形式設定

統計ログは、表計算またはデータベース・プログラムによって処理またはインポートできる、分かりやすい ASCII ファイルです。 以下の 3 つの型のレコードが書き込まれます。

- 特定のキャッシュのために統計の収集の始動を文書化する初期設定レコード。これらのレコードの形式は、以下のとおりです。

  mm/dd/yy hh:mm:ss id Initialization: interval *n* seconds

  ここで、

mm/dd/yy	統計の収集を開始する月、日、および西暦
hh:mm:ss	統計の収集を開始する時間、分、および秒
id	レコードに関連するキャッシュの名前
n	収集の間隔

- 特定のキャッシュのための統計収集の終端を文書化する終端レコード。これらのレコードの形式は、以下のとおりです。

  mm/dd/yy hh:mm:ss id Termination

  ここで、

mm/dd/yy	統計の収集を停止する月、日、および西暦
hh:mm:ss	統計の収集を停止する時間、分、および秒
id	レコードに関連するキャッシュの名前

- 統計レコードは、ブランクで区切られた数字の集合で、キャッシュ内の活動記録を表示します。これらのレコードの形式は、以下のとおりです。

  mm/dd/yy hh:mm:ss id statistics

  ここで、

mm/dd/yy	統計の収集を作成する月、日、および西暦
hh:mm:ss	統計の収集を作成する時間、分、および秒
id	レコードに関連するキャッシュの名前
<*statistics***>**	139ページの表 7 で、このキャッシュのために収集された統計のブランクで区切られたリストです。

表 7. 統計のリスト

フィールド番号	目次	説明	ゼロに再初期化されたカウンター
1	読み取り	キャッシュに対して実行する読み取り操作の番号	はい
2	書き込み	キャッシュに対して実行する書き込み操作の番号	はい
3	クローズ	キャッシュ内のオブジェクトで実行するクローズ操作の番号	はい

表 7. 統計のリスト (続き)

フィールド番号	目次	説明	ゼロに再初期化されたカウンター
4	オープン読み取り	キャッシュ内のオブジェクトで実行するオープン読み取り操作の番号	はい
5	オープン書き込み	キャッシュ内のオブジェクトで実行するオープン書き込み操作の番号	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で実行するオープン書き込み照会操作の番号	はい
7	読み取り瞬断	キャッシュ内のオブジェクトでの読み取り瞬断の番号	はい
8	書き込み瞬断	キャッシュ内のオブジェクトでの書き込み瞬断の番号	はい
9	書き込み照会瞬断	キャッシュ内のオブジェクトでの書き込み照会瞬断の番号	はい
10	初期設定	このキャッシュで明確化する新規セッションの番号	はい
11	終端	このキャッシュで終端するセッションの番号	はい
12	パージ	このキャッシュから削除するオブジェクトの番号	いいえ
13	使用する記憶域	キャッシュの記憶域部分でオブジェクトが使用する記憶域の量	いいえ
14	使用するディスク	キャッシュの記憶域部分でオブジェクトが使用するディスク空間の量	いいえ
15	使用可能な記憶域	キャッシュの記憶域部分でオブジェクトが使用するためのまだ使用可能な記憶域の量	いいえ
16	使用可能なディスク	キャッシュの記憶域部分でオブジェクトが使用するためのまだ使用可能なディスク空間の量	いいえ
17	メモリー・オブジェクトのカウント	キャッシュの記憶域部分にあるオブジェクトの番号	いいえ
18	ファイル・オブジェクトのカウント	キャッシュのディスク部分にあるオブジェクトの番号	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にあるセッションの番号	いいえ

第10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、パフォーマンスを向上させるためのストラテジーに関する説明をします。本章で説明するトピックは、以下の通りです。

Web Server API を使用してパフォーマンスを向上させる

CGI の代わりに Web サーバー API で 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 サーバーの API モードで実行されている場合には、 Net.Data は Web サーバーのプロセスでスレッドとして実行され、 CGI プロセスとして Net.Data を起動するオーバーヘッドがなくなります。 Web サーバー API を使用して、 Net.Data はサーバーのプロセス内で複数のスレッドとして実行されます。

デフォルトでは、 Web サーバーは CGI プログラムとして Net.Data を起動します。起動された Net.Data は、分離されたプロセスで実行されるプロセスとなります。パフォーマンスを向上させるために、Net.Data には Web サーバー API 用の構成オプションがあります。

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 プラグインおよび ICAPI プラグイン
: IBM Internet Connection Secure Sever プラグインへの後継としての 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。 system. 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、 32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。

考慮事項: Web サーバー API を使用すると、アプリケーションの分離をせずにパフォーマンスを向上することができます。 Net.Data はマルチ・スレッド・モードで実行

します。このため、ユーザー開発言語環境、不適切な呼び出し、またはデータベースの故障によるエラーは、 Web サーバーに問題を発生させ、サーバーをダウンさせることがあります。 CGI または FastCGI に対して Web サーバー API の内の 1 つを使用するかどうかは、ユーザーの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。

要件 :

- Net.Data を GWAPI、ICAPI、ISAPI または NSAPI モードで実行する場合には、Web サーバーをリスタートして、Web サーバーが Net.Data を再ロードし、プロセスとして実行できるようにしなければなりません。
- Web サーバーが Net.Data を API モードで呼び出した後に、初期設定ファイルを変更する場合には、 Web サーバーをリスタートしなければなりません。 Net.Data の初期設定ファイル (db2www.ini) に対して行った変更は無効です。 API モードで、Net.Data は初期設定ファイルを一度だけ読み取り、パフォーマンスのオーバーヘッドを削減します。
- API モードで実行されているときには、Oracle および Sybase の言語環境には、Live コネクションが必要です。

Web サーバー API を呼び出すには:

ICAPI および GWAPI の場合

構文:

```
http://server_name/CGI-BIN/db2www/macro_name/html_block
```

パラメーター:

server_name
サーバーの名前

macro_name
MACRO_PATH で指定するディレクトリーの下にある、ユーザーのマクロ・ファイルの相対パス名

html_block
処理されるマクロ・ファイル内の HTML ブロックの名前

例:

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

ISAPI の場合 :

構文:

```
http://server_name/server_HTML_root_directory/dll_name/macro_name/
  html_block
```

パラメーター:

server_name
サーバーの名前

server_HTML_root_directory
Web サーバー 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 で指定するディレクトリーの下にある、ユーザーのマクロ・ファイルの相対パス名

html_block
処理されるマクロ・ファイル内の HTML ブロックの名前

例:

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

NSAPI の場合 :

構文:

```
http://server_name/macro_name/html_block
```

パラメーター:

server_name
サーバーの名前

macro_name
MACRO_PATH で指定するディレクトリーの下にある、ユーザーのマクロ・ファイルの相対パス名 マクロ・ファイルの拡張子、たとえば、.d2w は、 Web サーバーの構成ファイルで定義しなければなりません。 詳しくは、32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。

html_block
処理されるマクロ・ファイル内の HTML ブロックの名前

例:

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

FastCGI によるパフォーマンスの向上

FastCGI によって、CGI-BIN の信頼性とともにパフォーマンスが改善されます。 FastCGI を使用すると、別々の記憶域スペースを使用するより信頼性の高いメソッドを使い、 API サーバーの感度でマクロを実行することができます。 Net.Data 起動のデフォルトは、CGI で実行できます。

FastCGI をサポートするすべてのサーバー上で、 FastCGI とともに Net.Data を使用することができます。

FastCGI の構成方法を理解するには、28ページの『FastCGI のための Net.Data の構成』を参照してください。

FastCGI を調整して、プロセス構成パラメーターをもつ着信要求の番号を処理するプロセスの適切な量を実行することができます。たとえば、1 人のお客さまが、秒当り

平均して 100 の要求を出し、そのそれぞれの要求を処理するのに 1/2 秒かかる場合です。着信要求は、プロセス・パラメーターを 50 に設定します。

FastCGI は、以下の言語環境でサポートされます。

- SQL
- Oracle 7.2, 7.3, 8.0
- ODBC
- Sybase
- REXX
- Perl

要件 : FastCGI モードで実行されているときには、 Oracle および Sybase の言語環境には、Live コネクションが必要です。

同時処理の数を調整するには、以下のようにします。

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 の場合の構成ファイルは、`httpd.conf` ファイルです。
   - ICS または Lotus Domino Go Webserver の場合の構成ファイルは、`lgw_fcgi.conf` ファイルです。
2. プロセスの番号を指定する構成パラメーター値を変更する。
   - Apache の場合 : `Process=`*num*.
   - ISC の場合 : `NumProcess=`*num*.

   ただし、*num* はプロセスの番号。

接続管理によって、パフォーマンスを向上させる

Net.Data は、Live コネクションを呼び出されたコンポーネントを提供し、データベースおよび Java 仮想マシン接続を管理します。 Live コネクションは、持続接続を保持し、パフォーマンスを改善します。 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は、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、Web サーバーへの各要求ごとに相当の開始時間がかかるようになっています。 Net.Data は、OS/2、Windows NT、および UNIX のオペレーティング・システムで、 Live コネクションを提供し、永続的な接続を保守します。

以下のセクションでは、Live コネクションについて説明します。

- 145ページの『Live コネクションについて』
- 146ページの『Live コネクションの利点』
- 146ページの『Live コネクションを使用すべきでしょうか ?』
- 146ページの『接続管理プログラムの開始』
- 147ページの『Net.Data および Live コネクション・プロセスの流れ』

Live コネクションについて

Live コネクションは、開始のオーバーヘッドを除去することによって、パフォーマンスを動的に改善することができます。始動関数を実行する 1 つまたは複数のプロセスを絶え間なく実行することによって、保管が発生します。これらのプロセスは、次に、サービス要求を待ち受けます。 Net.Data を CGI または FastCGI プログラムとして、あるいは、 Web サーバー API プラグインとして使用すると、Live コネクションを実行することができます。

Live コネクションは、接続管理プログラムおよびクライエットで構成されます。クライエット は、接続管理プログラムが介しするプロセスで、サーバーが実行されている間は、活動状態に保たれます。クライエットはデータを処理し、キーワード CLIETTE を使用して初期設定ファイルで指定する Net.Data 言語環境と通信します。各型のクライエットは、DB2 クライエットなどの特定の言語環境関数を処理します。この DB2 クライエットは DB2 データベースに接続し、 Net.Data が Net.Data マクロを処理する前に、 SQL 呼び出しを実行する命令をセットアップします。実行可能ファイルは、Live コネクションの構成ファイル `dtwcm.cnf` で命名されます。 145ページの図 25 は、 Live コネクション、マクロ・ファイル、および言語環境の間の対話を示しています。

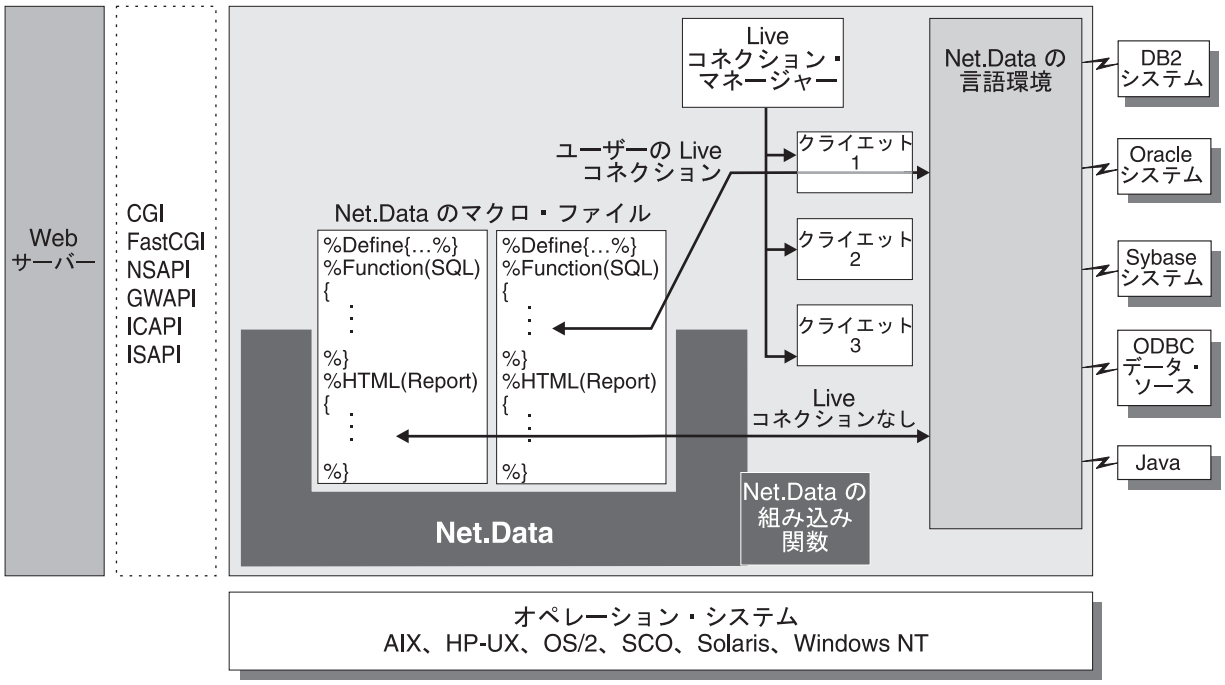


図 25. クライエットによる Live コネクション

以下のセクションでは、Live コネクションについて詳細に説明します。 Live コネクションの構成方法を理解するには、 23ページの『Live コネクションの構成』 を参照してください。

Live コネクションの利点

Live コネクションの使用には、以下の主な利点があります。

- **パフォーマンスの改善**

  新規接続を作成するより、再使用する方が効果的です。一般的に、短精度の SQL ステートメントを要求する場合 (たとえば、100000 未満の行のデータベースで行う簡単な照会)、あるいは、ご使用のデータベース接続が困難な場合 (たとえば、相手側サーバー)には、接続時間が重要です。

- **多重データベースのアクセス**

  Live コネクションは、1 つの Net.Data マクロを、同時に複数のデータベースに接続させることができます。これが可能なのは、各データベースに固有のクライエットがあるためで、その結果、Net.Data は複数のクライエットと簡単に通信することができます。

Live コネクションを使用すべきでしょうか ?

CGI、FastCGI、または API モードで Live コネクションを使用し、ユーザーのデータベースと通信することができます。さらに、ユーザーのアプリケーションが複数のデータベースからのデータを要求する場合には、 Live コネクションから利益を得るかもしれません。

API プラグインとともに使用する Live コネクションは、多くのシステムのパフォーマンスを改善しますが、システムの負荷および構成によって異なります。ユーザー独自のシステムで体験して、ユーザーのために最善の作業をする構成を決定すべきです。

多くのアプリケーションでは、 Live コネクションを使用しなくてもパフォーマンスを改善することができます。その方法は、ACTIVATE DATABASE コマンドまたは START DATABASE コマンドを使用して、データベース接続を確立する時間を省くことができます。ユーザーのデータベースで使用するコマンドの詳細については、ご使用の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。また、ご使用のオペレーティング・システムのドキュメンテーションを検査して、パフォーマンスの改善に役立つ追加のステップがあるかどうか見てください。

要件 : CGI、FastCGI、および API モードで実行されているときには、 Oracle および Sybase の言語環境には、Live コネクションが必要です。

接続管理プログラムの開始

接続管理プログラムは別々の実行可能ファイルで、 Net.Data とともに出荷され、`dtwcm` と命名されます。 Web サーバーを開始するときに、接続管理プログラムを開始します。

接続管理プログラムを開始すると、この管理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。それぞれのプロセスでは、接続管理プログラムが特定のクライエットの実行を開始します。 Live コネクションの構成方法を理解するには、 23ページの『Live コネクションの構成』 を参照してください。

Windows NT および OS/2 を使用して、接続管理プログラムを開始するには :

1. コマンド行から、<*inst_dir*>¥connect¥ ディレクトリーに変更する。
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。

AIX を使用して、接続管理プログラムを開始するには :

1. コマンド行から、/usr/lpp/internet/db2www/db2/ ディレクトリーに変更する。
2. dtwcm を入力する。

メッセージ・オプションを使用して、接続管理プログラムを開始するには :

デフォルトでは、接続管理プログラムのメッセージは抑止されます。接続管理プログラムのメッセージを表示したい場合には、接続管理プログラムを開始するときに、-d オプションを使用します。

コマンド行から、dtwcm -d を入力します。

-d オプションを使用後、メッセージを再度抑止するために、接続管理プログラムをリスタートする必要があります。

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 コネクション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 コネクション・プロセスの流れ

データベース、Web サーバー、および接続管理プログラムを構成し開始した後、 Live コネクションが使用可能である場合には、一般的に、 Net.Data プロセッシングには以下のステップが含まれます。

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 のマクロ処理を開始する。
3. Net.Data で、Live コネクションを使用する関数呼び出しが発生すると、 Net.Data は、必要なクライエットの型を初期設定ファイルから判別する。 DB2 の場合は、クライエット型は、DTW_SQL:CELDIAL などの、 DB2 データベース名に基づいた名前になることがよくあります。
4. Net.Data では、その型のクライエットに、接続管理プログラムを要求する。

5. 接続管理プログラムは、その型で使用可能なクライエットを探す。使用可能なクライエットがない場合には、接続管理プログラムはその要求をキューに入れ、正しいクライエットの型が使用可能になったときに、待機中の要求を処理します。
6. クライエットが使用可能になると、接続管理プログラムは、そのクライエットで通信する方法を、Net.Data に知らせる。
7. Net.Data はクライエットにその関数を処理するように要求する。
8. このプロセスは、マクロ・プロセッシングが完了するまで、ステップ 3 から繰り返されます。
9. すべてのクライエットが解放される。

クライエットを初期設定ファイルで指定しても、接続管理プログラムが実行されない場合には、 Net.Data が DLL をロードしてマクロを処理します。 API を使用する場合には、エラーを受け取る可能性があり、接続管理プログラムを開始しなければなりません。

Net.Data のキャッシングを使用したパフォーマンスの改善

頻繁に要求される Net.Data の Web ページをキャッシュ内でキャッシングして、ご使用のシステムのパフォーマンスを改善することができます。キャッシュからページを検索して、使用するプロセッサー時間を削減することができます。それは、Net.Data マクロがインタープリットされることも、また、 Net.Data の言語環境呼び出しが実行されることもないからです。 Net.Data のページがデータベースまたはファイル内に保留されている情報から生成される場合には、ディスク I/O アクセスが削減される可能性もあります。このような省力の結果、スループットおよび応答時間が削減されていることも発見するかもしれません。

Net.Data のキャッシングによって、シングル・プロセス内で複数のキャッシュをマルチ・スレッドで管理することができます。キャッシングは、複数の機械の多重プロセス間でシングル・キャッシュを共用するネットワーク環境で、十分に作業します。 Net.Data のキャッシングでは、キャッシュ管理プログラム部品を使用して、Net.Data のキャッシュを管理します。

ユーザーが定義するキャッシュのサイズとユーザーの既存のシステムによって、ページ送りを避けるために、追加の記憶域が必要となる可能性があります。

Net.Data のキャッシングのセットアップおよび使用の方法を理解するには、 119ページの『第9章 Net.Data のキャッシング』 を参照してください。

Net.Data のエラー・ログ : パフォーマンスの考慮

Net.Data はエラー・ログを提供し、ユーザーの Net.Data システムでのエラーまたはパフォーマンス上の問題を、ユーザーがモニターすることができるようにします。

Net.Data のエラー・ログを使用するときに、たくさんのメッセージがエラー・ログに書き込まれている場合には、ご使用のシステムのパフォーマンスへの影響に気付く可能性があります。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 はエラー・ログに出力のメッセージを渡します。

パフォーマンスへの影響を削減するには、DTW_LOG_LEVEL キーワードを使用して、Net.Data のマクロで設定されるエラー・ログのログ段階を検査します。その段階が WARNING に設定されている場合には、段階を、短精度のパフォーマンス向上のための ERROR に、あるいは、より大きなパフォーマンス向上のための OFF に削減することを考えてください。

Math 関数を最適化する

DTW_OPTIMIZE_MATH 構成変数を使用して math 関数のパフォーマンスを向上することができます。この変数によって数学関数の C 言語スタイル・フォーマットを利用することが可能になり、これによって関数の処理がより速くなります。 YES に設定すると、Net.Data は C スタイル・フォーマットを使用します。 NO がデフォルトです。この変数の構成方法を理解するには、16ページの『DTW_OPTIMIZE_MATH: 数学関数変数の最適化』 を参照してください。

構文:

```
DTW_OPTIMIZE_MATH=NO|YES
```

考慮事項: Net.Data バージョン 1 は、数学関数に REXX スタイル・フォーマット を使用していました。出力は、DTW_OPTIMIZE_MATH 構成変数を使用する場合とは異なります。後書きブランクは表示されません。出力が Net.Data バージョン 1 マクロ・ファイルと整合性を持つことが、ユーザー・アプリケーションに重要であれば、パフォーマンスの他に整合性のメリットも考慮してください。

第11章 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。

本章では、以下のログ・トピックについて説明します。

ログのモニターをプランする

エラーを記録する場合には、以下の項目に関するプランを作成する必要があります。

- ディスク空間を決定します。

  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。 15ページの『DTW_LOG_DIR: エラー・ログの位置変数』 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・ファイルの中で、DTW_LOG_LEVEL キーワードでログのレベルを設定します。

- Net.Data を実行します。

  エラー・ログを使用している場合には、ご使用の Net.Data システム内のエラーを、エラー・ログおよび保存ファイルで検査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。パフォーマンスの項目については、148ページの『Net.Data のエラー・ログ : パフォーマンスの考慮』 を参照してください。

ログ・ファイル内のログの容量を制御する

DTW_LOG_LEVEL キーワードで、ログ収集のレベルを指定することができます。Net.Data マクロ・ファイルでこのキーワードを定義します。このキーワードには 3 つの設定があります。

オフ Net.Data はエラー・ログを収集しません。 これはデフォルトです。

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。

ログの収集がされないメッセージの種類

Net.Data は、マクロ・ファイルの MESSAGE セクションで処理が明示されているエラーのログは収集しません。

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.log*MMMDDYYYY_nn* にアーカイブされます。

変数:

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。

ログ・ファイルのフォーマット

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・ファイル

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

付録A. AIX 用の Net.Data

AIX についての詳細は、Net.Data とともに出荷される README ファイルに組み込まれています。 README ファイルには、以下の情報が組み込まれています。

- 要件
- インストール
- 構成
- インストール解除

言語環境のための共用ライブラリーのロード

AIX の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、言語環境は、Net.Data が呼び出すルーチンを提供するために必要で、 dtw_initialize() および 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 dtw_fp 構成を使用して、 AIX の言語環境からの言語環境インターフェース・ルーチンを指すポインターを検索し、structure to retrieve pointers to the language 以下の形式を取ります。

```
typedef struct dtw_fp {
    int (* dtw_initialize_fp)();   /* dtw_initialize function pointer */
    int (* dtw_execute_fp)();      /* dtw_execute function pointer    */
    int (* dtw_getNextRow_fp)();   /* dtw_getNextRow function pointer */
    int (* dtw_cleanup_fp)();      /* dtw_cleanup function pointer    */
} dtw_fp_t;
```

この構成は、共用ライブラリーをロードするときに、 Net.Data が dtw_getFp() ルーチンとして言語環境に渡します。

dtw_fp 構成はパラメーターとしてのみ渡されます。この構成は、各サポート済みインターフェースのフィールドを含み、これらのフィールドを設定することは、言語環境の応答性です。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は、言語環境がそのインターフェースの関数ポインターにフィールドを設定します。指定されたインターフェースを提供していない場合には、言語環境は、そのフィールドを空白に設定します。プログラム・テンプレート内の dtw_getFp() ルーチンは、このルーチンの正しい導入を示しています。

共用ライブラリーをロードするときに、 Net.Data がこのルーチンを指すポインターを取得するために、 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る最初のエントリー・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 を呼び出したライブラリーのエクスポート・ファイルの例は、以下のようになります。

```
#!dtwsampshr.o
dtw_getFp
dtw_initialize
dtw_execute
dtw_getNextRow
dtw_cleanup
```

REXX 環境でのパフォーマンスの向上

ご使用の AIX で、REXX 言語環境を数多く呼び出している場合には、RXQUEUE_OWNER_PID 環境変数を 0 に設定することを考えてください。 REXX 言語環境を数多く呼び出したマクロは、簡単にたくさんのプロセスを作成し、システム・リソースを開拓します。

環境変数を、以下の 3 つの方法のいずれかで設定することができます。

- 以下の DTW_SETENV 組み込み関数を使用して、マクロ・ファイル内で設定する。

  ```
  @DTW_rSETENV("RXQUEUE_OWNER_PID", "0")
  ```

- 以下の AIX のシステム環境ファイル内で設定する。

  ```
  /etc/environment: RXQUEUE_OWNER_PID = 0
  ```

  このメソッドは、マシン全体の REXX の性質に影響を及ぼします。

- HTTP Web サーバー環境ファイル、たとえば、Lotus Domino Go Webserver の場合には、以下のステートメントを挿入します。

  ```
  InheritEnv RXQUEUE_OWNER_PID = 0
  ```

  このメソッドは、Web サーバーに対する REXX の性質に影響を及ぼします。

付録B. Net.Data の SmartGuides

Net.Data の SmartGuides は、カスタマイズされた Net.Data のアプリケーションを作成するための、速くて簡単な方法をユーザーに提供するために設計されています。ただ、SmartGuide を選択し、2、3 の質問に応答すると、 Net.Data がユーザーのために、カスタマイズされたアプリケーションを作成します。

Net.Data は、以下の SmartGuide を提供します。ユーザーは、マクロの作成方法と Net.Data のフィーチャーの使用方法を学習する間、この SmartGuide を使用します。

Drilldown

この SmartGuide は、ユーザーの既存のデータベース表を採用し、 Web で使用できるドリルダウンのアプリケーションを作成します。このアプリケーションを使用すると、異なるレベルのデータの詳細にアクセスすることができます。任意選択で、Drilldown SmartGuide はユーザーのデータベースに接続し、ユーザーのデータベース情報を収集することができます。最高 5 つの Web レポートをカスタマイズできます。生成済みのマクロ・ファイルは、ユーザーのデータベースに保管されるプライマリー・キー情報と外部キー情報を使用して、ユーザーの Web レポートを自動的にリンクします。

Forum

この SmartGuide は、メッセージの郵送、表示、およびメッセージへの応答のために Web が使用できる討論フォーラムを作成します。フォーラムのカテゴリーと討論のトピックをカスタマイズすることができます。生成済みのフォーラム・アプリケーションには関数が組み込まれ、キーワードを郵送するフォーラムを検索します。任意選択で、ユーザーのマクロ・ファイル内で Cookie 関数を使用可能にして、フォーラムがユーザーを 『記憶する』 ことができるようにします。

Stored Procedure

この SmartGuide はユーザーのデータベースに接続し、ユーザーのデータベースとともに登録されるすべてのストアード・プロシージャーのリストを検索します。ストアード・プロシージャーを選択すると、 SmartGuide が、ユーザーのストアード・プロシージャーを呼び出す、ユーザーの Net.Data のマクロ・ファイルを生成します。次に、生成済みのマクロ・ファイルを変更するか、あるいは、ユーザーの既存のアプリケーション内で統合します。

接点 この SmartGuide は、名前、アドレス、電話番号、およびほかの重要な接点の情報を保管するために、 Web が使用できる住所録を生成します。この住所録は、ユーザーの接点に迅速にアクセスできる検索関数に付属しています。生成済みの接点アプリケーションは、簡単なレポートまたは詳細レポートのいずれかに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も組み込むことができます。

Net.Data の SmartGuide パッケージの最新版の、
`http://www.software.ibm.com/data/net.data` で、Net.Data の Web サイトをチェックします。

事前準備

SmartGuides を実行し、Net.Data のマクロ・ファイルを生成するには、以下のソフトウェアをインストールし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 (Drilldown SmartGuide の場合には必須)

SmartGuide の実行

Net.Data の SmartGuides はコマンド行から開始され、
ファイル NetDataSmartGuides.jar に含まれます。

Java 開発キット (JDK) で SmartGuide を開始するには :

1. 以下の行をユーザーの CLASSPATH 環境変数に追加する。

   *[Path]*NetDataSmartGuides.jar

   ここで、*[Path]* は、 NetDataSmartGuides.jar ファイルへの任意選択のパス。

2. Drilldown および Stored Procedure の SmartGuide のデータベース接続フィーチャーを使用したい場合には、 UDB JDBC ドライバーをユーザーの CLASSPATH 環境変数に追加します。

   Windows NT および OS/2 のオペレーティング・システムの場合には、以下の行をユーザーの CLASSPATH 環境変数に追加します。

   *[Path]*NetDataSmartGuides.jar;*[UDBInstallationPath]*¥jav¥db2java.zip

   UNIX オペレーティング・システムの場合には、以下の行をユーザーの CLASSPATH 環境変数に追加します。

   *[Path]*NetDataSmartGuides.jar:*[UDBInstallationPath]*¥java¥db2java.zip

   ここで、*[Path]* は、NetDataSmartGuides.jar ファイルへの任意選択のパスで、*[UDBInstallationPath]* は、ユーザーの UDB インストール、たとえば、 C:¥SQLLIB へのパスです。

3. SmartGuide を開始する。
   - UNIX オペレーティング・システムの場合には、以下のコマンドを入力する。

     ```
     java TaskGuide LaunchPad.class
     ```

   - Windows NT および OS/2 のオペレーティング・システムの場合には、以下のファイルを実行する。

     ```
     SmartGuides.cmd
     ```

   159ページの図 26 のように、使用可能な SmartGuide のリストと一緒に、ランチパッドがオープンします。

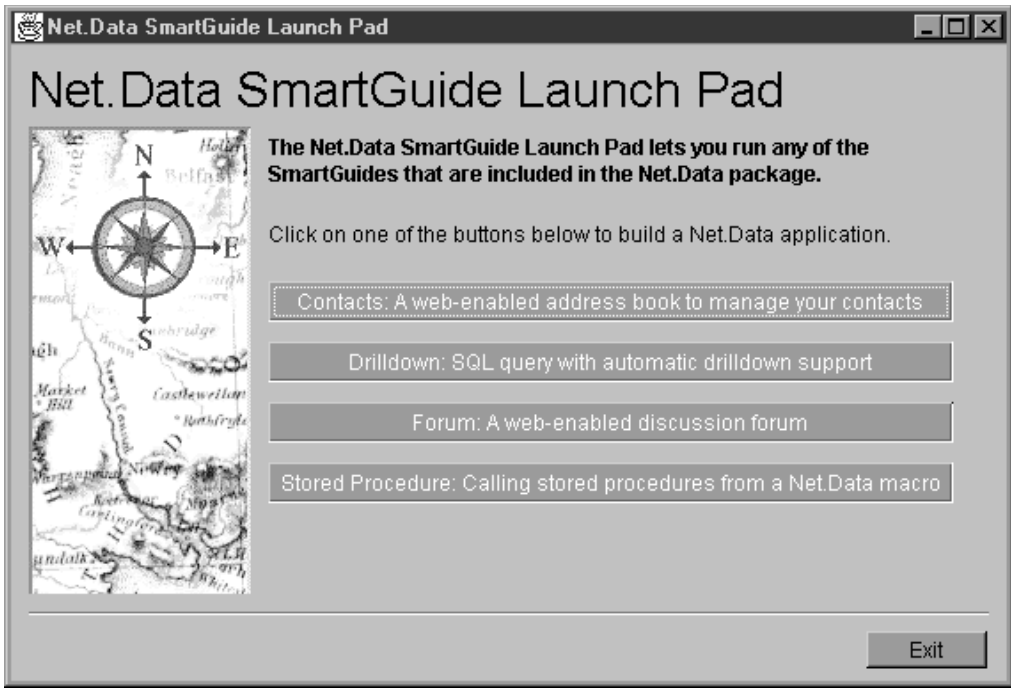


図 26. SmartGuide のランチパッド

4. 実行したい SmartGuide の名前をクリックする。

Java 実行時環境 (JRE)で SmartGuide を開始するには :

1. 以下のコマンドを入力して、SmartGuide を実行する。

   ```
   jre -cp [Path]NetDataSmartGuides.jar TaskGuide LaunchPad.class
   ```

   ここで、*[Path]* は、 NetDataSmartGuides.jar ファイルへの任意選択のパス。

   159ページの図 26 のように、使用可能な SmartGuide のリストと一緒に、ランチパッドがオープンします。

2. Drilldown および Stored Procedure の SmartGuide のデータベース接続フィーチャーを使用したい場合には、以下のコマンドを入力する。

   Windows NT および OS/2 のオペレーティング・システムの場合は :

   ```
   jre -cp [Path]NetDataSmartGuides.jar;[UDBInstallationPath]
     ¥java¥db2java.zip TaskGuide LaunchPad.class
   ```

   UNIX オペレーティング・システムの場合は :

   ```
   jre -cp [Path]NetDataSmartGuides.jar:[UDBInstallationPath]
     ¥java¥db2java.zip TaskGuide LaunchPad.class
   ```

   ここで、*[Path]* は、NetDataSmartGuides.jar ファイルへの任意選択のパスで、*[UDBInstallationPath]* は、ユーザーの UDB インストール、たとえば、 C:¥SQLLIB へのパスです。

3. 実行したい SmartGuide の名前をクリックする。

付録C. Net.Data サーブレットと NetObjects Fusion NOF プラグインを使用する

Net.Data は、Net.Data サーブレット用に NetObjects Fusion プラグインを提供します。Net.Data サーブレットは、109ページの『Net.Data サーブレット』 に記述されています。

NetObjects Fusion (NOF) を使用して、ユーザーの既存の Net.Data のマクロ・ファイルを統合します。これによって、Web サイト・マネージメントとすぐれた統合性をもち、簡単に使用できる GUI が提供されます。

NetObjects Fusion プラグインについて

NetDataServlet.NFX プラグインは Net.Data サーブレット 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・ファイルの呼び出しや、単一の Net.Data 関数の呼び出しをサポートします。 プラグインは HTML を生成し、サーブレットまたは server-side-include (SSI) として Net.Data を起動します。 Web サーバーが Net.Data を起動する時に、 Net.Data マクロ・ファイルか関数が実行されます。 Net.Data サーブレット プラグインを使用して、マクロおよび関数サーブレット 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、161ページの図 27 で説明しています。プラグインは、マクロ・ファイルの構築を自動化するために選択されたデフォルト値で、多数の基本マクロ・ファイルや関数パラメーターを提供します。 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。

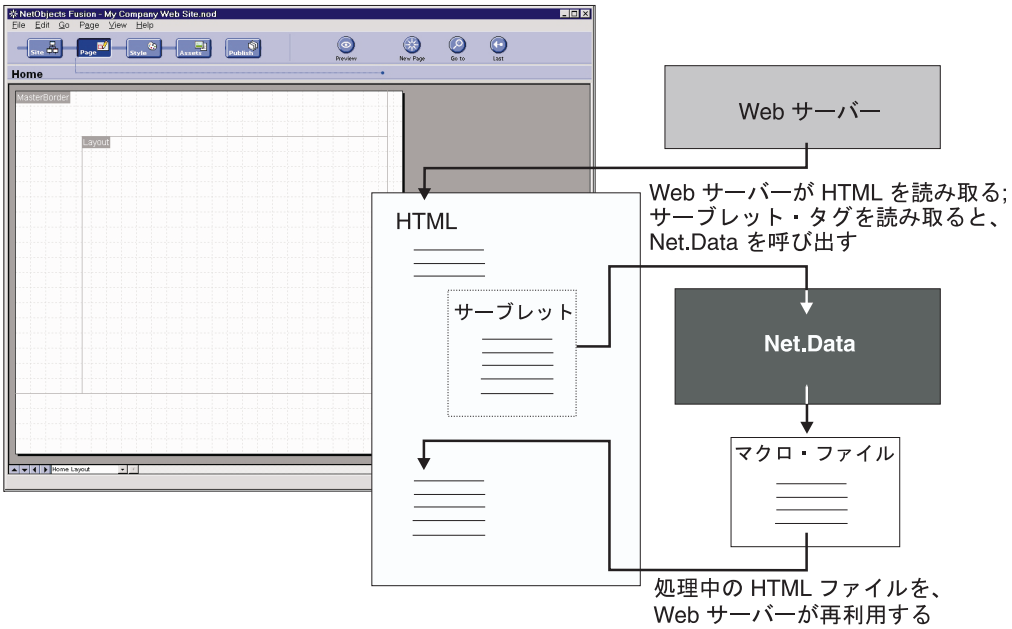


図 27. Net.Data サーブレット・プラグイン

NetObjects Fusion プラグインの導入

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。

インストールするには : <inst_dir>¥fx¥NetDataServlet.nfx および <inst_dir>¥fx¥NetDataServlet.gif をユーザーの <NetObjects Fusion>¥components¥ ディレクトリーにコピーします。

NetObjects Fusion の Net.Data プラグインのセットアップ

NOF を使用して作業中の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・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ともに使用します。
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・プロパティーを変更する

Net.Data サーブレット・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ーを、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。 NOF の「**プロパティー (Properties)**」パレットがオープンし、 163ページの図 28 に示すように、プラグイン・プロパティーが表示されます。

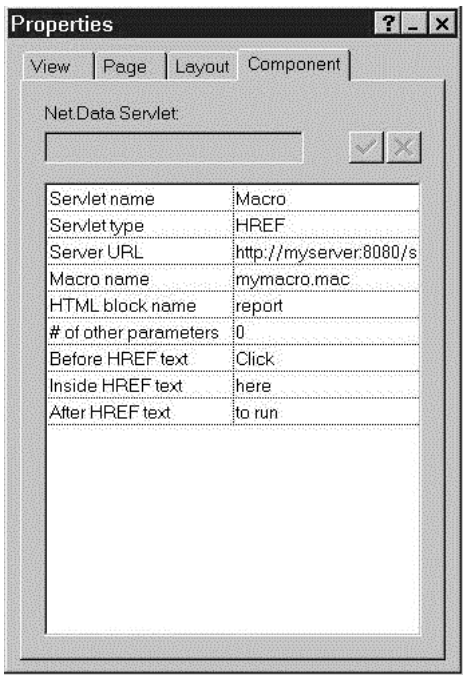


図 28. Net.Data サーブレット プロパティー・パレット

Net.Data サーブレット 用のプロパティー:

カスタマイズできるのは、以下のプロパティーです。

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、異なるプロパティーが表示されます。

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、異なるプロパティーが表示されます。

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。指定しない場合には、このプロパティーは表示されません。

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、このプロパティーは表示されません。

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 マクロ名を指定して実行します。指定しない場合には、このプロパティーは表示されません。

HTML ブロック名

マクロ・サーブレット 名を指定する場合には、Net.Data マクロ・ファイル内の HTML ブロックの名前を指定して、実行します。指定しない場合には、このプロパティーは表示されません。

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。指定しない場合には、このプロパティーは表示されません。

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。これ以外の場合には、このプロパティーは表示されません。

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。これ以外の場合には、このプロパティーは表示されません。

データベース

関数のサーブレット名を選択する場合には、使用するデータベースの名前を指定します。指定しない場合には、このプロパティーは表示されません。

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、このプロパティーは表示されません。(オプション)。

HREF 内部テキスト

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、このプロパティーは表示されません。(オプション)。

HREF 後テキスト

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、このプロパティーは表示されません。(オプション)。

SQL 確認事項!

HREF サーブレット・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。指定しない場合には、このプロパティーは表示されません。

b. ユーザーのページにプロパティーを定義した後で、「**パブリッシュ (Publish)**」押しボタンをクリックすると、Net.Data サーブレット 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 サーブレット・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「**プロパティー (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 プラグインを使用してサーブレットを表示する

ユーザーのページにプロパティーを設定した後で、「**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D. Net.Data サンプル・マクロ

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

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

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

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

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


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

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

%{     End of include file  %}
```

付録E. 特記事項

以下の情報は、米国で提供される製品とサービスに関するものです。本書で説明されている製品、サービス、または機能でも、米国以外の国では提供されていないことがあります。お客様の地域で現在使用可能な製品およびサービスに関する情報については、その地域の IBM 担当員にお尋ねください。本書で、IBM ライセンス・プログラムまたは他の IBM 製品に言及している部分があっても、このことは当該プログラムまたは製品のみが使用可能であることを意味するものではありません。これらのプログラムまたは製品に代えて、 IBM の知的所有権を侵害することのない機能的に同等のプログラムまたは製品を使用することができます。ただし、IBM によって明示的に指示されたものを除き、これらのプログラムまたは製品に関連する動作の評価および検査はお客様の責任で行っていただきます。

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

〒106-0032 東京都港区六本木3丁目2-31
AP事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

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

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

IBM Corporation
_W92/H3_
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
_U.S.A.

本プログラムに関する上記の情報は、適切な条件の下で、使用することができますが、有償の場合もあります。

本書において解説されているライセンス・プログラムおよびそのライセンス・プログラム資料は、「IBM 使用契約書」の契約条件に基づいて弊社から提供されるものです。

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

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

本書には、日常の業務処理で用いられるデータや報告書の例が含まれていますが、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、ブランドの、あるいは製品など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いても、それは偶然に過ぎません。

商標

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

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

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

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

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

Lotus および Domino Go Webserver は、米国ならびにその他の国における Lotus Development Corporation の商標です。

Microsoft、Windows、Windows NT、および Windows のロゴは、米国ならびにその他の国における Microsoft Corporation の商標あるいは登録商標です。

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

用語集

API. アプリケーション・プログラミング・インターフェース (Application programming interface)。

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

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

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

キャッシュ (cache). 最近アクセスされたデータが入るメモリーの一部またはディスク空間。同一データに続けてアクセスする場合の速度を上げることが目的。キャッシュは、ネットワークを介してアクセス可能な、使用頻度の高いデータのローカル・コピーを保留するのに使用される場合が多い。

キャッシング (caching). ローカルに Web サーバーに要求した結果得られる使用頻度の高いデータを高速で取り出すために、情報を最新表示するときまで保管するプロセス。

キャッシュ管理プログラム (Cache Manager). 1 つのマシン用のキャッシュを管理するプログラム。複数のキャッシュを管理できる。

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

CLIETTE. Web サーバーからの要求に、Net.Data の Live コネクションで長期にわたって対応する処理。 この接続管理プログラムは、これらの要求に対応する 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)。

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

接続管理プログラム (Connection Manager). Live コネクションのサポートに必要とされる Net.Data 内の、実行可能ファイル、`dtwcm`。

クッキー (cookie). HTTP サーバーによって Web ブラウザーに送信され、次にブラウザーがそのサーバーにアクセスするつど、送り返す情報のパケット。クッキーには、サーバーが選択する任意の情報を含めることができ、特に国籍をうたわない HTTPトランザクション間の状態を保持するのに使用される。 *Free Online Dictionary of Computing* より。

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

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

データ型 (data type). 列およびリテラルの属性。

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

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

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

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

HTTP. Hypertext transfer protocol (HTTP)。

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

hypertext transfer protocol (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。

ICAPI. インターネット接続 API (Internet Connection API)。

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

ICSS. Internet Connection Secure Server。

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

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

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

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

ISAPI. Microsoft Internet Server API。

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

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

Live コネクション (Live Connection). 接続管理プログラムおよびクライエットで構成される Net.Data コンポーネント。 Live コネクションは、データベースの再使用および Java 仮想マシン・コネクションを管理します。

LOB. ラージ・オブジェクト (Large object)。

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

NSAPI. Netscape API。

ヌル (null). 情報がないことを示す特殊な値。

パス (path). ファイルを探すのに使用する検索経路。

Perl. 解釈済みプログラミング言語。

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

TCP/IP. 伝送制御プロトコル/インターネット・プロトコル (Transmission Control Protocol / Internet Protocol)。

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

URL. Uniform resource locator (URL)。

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

Web サーバー (Web server). (1) Internet Connection などの HTTPmiddleware (2) サーバー・ソフトウェアを実行する計算機。

索引

日本語, 英字, 数字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

［ナ行］

［ハ行］

［マ行］

［ヤ行］

［ラ行］

A

B

C

D

E

F

G

H

I

J

L

M

N

O

R

U

W

Printed in Japan

SB88-7371-00