Net Express

ソリューション ガイド

Micro Focus NetExpress™

ソリューション ガイド

Micro Focus®

第 3 版
1998 年 10 月

Micro Focus は、このマニュアルの内容が公正かつ正確であるよう万全を期しておりますが、このマニュアルの内容は予告なしに随時変更されることがあります。

このマニュアルに述べられているソフトウェアはライセンスに基づいて提供され、その使用および複写は、ライセンス契約に基づいてのみ許可されます。特に、Micro Focus 社製品のいかなる用途への適合性も明示的に本契約から除外されており、Micro Focus はいかなる必然的損害に対しても一切責任を負いません。

Micro Focus® は、Micro Focus Limited の登録商標です。Object COBOL™ 、NetExpress™ 、および WebSync™ は、Micro Focus Limited の商標です。

WinCIM® と CompuServe® は、CompuServe Incorporated の登録商標です。

Alta Vista™ は、Digital Equipment Corporation の商標です。

Microsoft® 、Windows® 、および MS-DOS® は、Microsoft Corporation の登録商標です。Windows NT™ および AutoPlay™ は Microsoft Corporation の商標です。

Netscape™ 、Netscape Navigator™ は Netscape Communications Corporation の商標です。

Yahoo!™ は Yahoo! Incorporated の商標です。

目次

第1章 はじめに

この章では、このマニュアルの目的、対象読者、NetExpress の他のマニュアルとの使用方法について説明します。また、このマニュアルの他の章で使用する仮想企業のシナリオ例についても説明します。

1.1 このマニュアルの目的

このマニュアルは、完全なクライアント サーバー ソリューションを開発するプログラマや、NetExpress で実際の COBOL アプリケーションを作成する場合の技術や問題について具体的に理解する必要があるプログラマを対象としています。

このマニュアルでは、NetExpress でサポートされている主要な技術の重要な概念や実例を中心に説明します。同時に、現場に強いクライアント サーバー COBOL アプリケーションを設計、プログラミング、実装する上での一般的な課題も説明します。このマニュアルの目的は、考えられるソリューションの選択肢を紹介することです。そのために、重要な技術的情報と広く普及している詳細情報を紹介し、十分な情報をもとに意思決定ができるようにします。また、説明されている技法については、適宜、実例を示します。

このマニュアルは、NetExpress の機能の使用方法を詳細に説明する他のオンライン マニュアルや、包括的なリファレンスと手順ごとの作業情報を提供する NetExpress オンライン ヘルプ システムと併用するように制作されています。テキストで表示された参照箇所は、オンライン マニュアルやヘルプ システムで参照する特定のトピックを指します。

このマニュアルでは、常に NetExpress で使用できる最新のテクニックを紹介したいと思います。今後ここで紹介する内容についてご意見やご提言をお持ちの方、または、このマニュアルより優れた技法を見つけられた方は、弊社にご連絡ください。その成果を他の COBOL プログラマと共有させていただければ幸いです。連絡先については、オンライン ヘルプの目次から [*Micro Focus 社 連絡先*] を選択し、参照してください。

1.2 シナリオ例

このマニュアルで紹介する多くのソリューションでは、架空のシナリオについて設計されたアプリケーションのコードを使用して、実際の状況でアプリケーションを説明します。これらのシナリオについては、コードが複雑すぎてわかりにくくならないように、特定のアプリケーションをできるだけ現実的に説明するように設計されています。詳細は、後述します。たとえば、これらのシナリオには、実際のアプリケーションに必須の機能 (データ検証など) を組み込んでいますが、必ずしも現実の環境でこれらを実装するとはかぎりません (たとえば、データベースとトランザクション処理システムを併用する場合など)。そのため、概要を読み、シナリオの雰囲気を理解してから、このマニュアルの後半で必要に応じて特定の段階に戻って参照してください。

1.2.1 Gas Services

Gas Services Incorporated は、国内の顧客にガスを供給する会社です。現在は、国内の東部にだけガスを供給してい

ますが、州全体に供給を広げるために、競合会社である WestGas Corporation と、合併を視野に入れて交渉しています。サービス地域の各世帯には、ガスメーターが備え付けられ、課金期間のガスの請求書を発行する直前に Gas Services の従業員が定期的にこのメーターを読み取ります。従業員が何らかの理由でメーターを読み取ることができない場合、その期間のガス消費量を予測し、予測した数値を使用して請求書を発行します。顧客が予測した数値に同意しない場合、自分自身でメーターを読み取り、数値を知らせると、この数値を使用して請求書を作成します。

1.2.1.1 第 1 段階

Gas Services 社の顧客は、現在、自分で読み取ったメーターの数値を Gas Services 社の電話相談を使用して報告することができます。ただし、顧客が WWW のページに接続し、Web アプリケーションを使用してメーターの読み取り値を更新するという条件があります。サーバー側のプログラムは、Gas Services 社の既存の Windows NT Web サーバーで実行されます。

1.2.1.2 第 2 段階

Gas Services 社には、顧客の照会を処理する電話相談オペレータをサポートするためのクライアント サーバー システムが必要です。顧客は、電話相談に電話し、読み取ったメーターの数値の報告、選択したガス課金方法の変更、請求書に関する照会、一般的な照会などを行うことができます。電話相談オペレータは、操作できるデータベース システムから必要な顧客データを検索し、データへの変更を承認するためのシステムを必要とします。電話相談のスーパーバイザは、さらに、ガス供給の課金方法の変更を申し込んだ顧客に対して各営業日の終業時に回答を作成するシステムを必要とします。

システムは、電話相談センタの Windows NT サーバーに導入されます。Gas Services 社のデータは、UNIX サーバーの Oracle リレーショナル データベースに保存されています。電話相談のアプリケーション サーバーは、広域ネットワークによりデータベースにリンクされています。電話相談オペレータは、ローカル エリア ネットワークに接続されている Windows 95 マシンを使用します。各オペレータは、同時に 4 件までの顧客の照会を処理する必要があります。

1.2.1.3 第 3 段階

WestGas 社との合併が完了しました。

Gas Services 社と WestGas 社は、合併前、それぞれ電話相談を運営し、独立したデータベースを持っていました。以前の WestGas 社のデータベースは、メインフレームで COBOL ファイルとして保存されています。合併以来、電話相談は 1 つの電話相談 センタに統合されましたが、それぞれのデータベースはまだ別々のままです。

1.2.1.4 第 4 段階

Gas Services 社は、標準的なガス課金方法として 12 種類の方法を提供しています。しかし、市場の状態が厳しいため、同社は顧客の個々の状況に合わせることができる特別な課金方法を導入しました。電話相談オペレータは、過去 12 ヶ月の間に顧客が使用したガスの量を調べ、これを使用して、通常の年間消費量を求め、課金方法の変数を調節した場合に料金がどの程度変化するかを調べることができます。

第2章 クライアント　サーバー アプリケーションの開発

この章では、クライアント サーバー アプリケーションの基本的な原則を紹介し、伝統的なモノリシック アーキテクチャよりも優れている点について説明します。また、アプリケーションをモジュールに分割し、クライアント サーバー アーキテクチャを最も有効に利用する方法を紹介し、NetExpress アプリケーションで使用するプラットフォームの概要を説明します。

2.1 クライアント サーバー アーキテクチャ

クライアント サーバー アプリケーションが急速に普及している一方で、「クライアント サーバー」という用語は、さまざまな概念の説明に使用されるため、不明瞭な印象を与えます。

2.1.1 クライアント サーバー アプリケーションの定義

原則的に、クライアント サーバー アプリケーションは、*サーバー プログラム*が提供するサービスを使用する*クライアント プログラム*で構成されます。クライアントは、サーバー アプリケーションの関数を呼び出すことによって、サーバーからのサービスを要求します。分散コンピューティング環境では、クライアント プログラムとサーバー プログラムは別のマシンで実行され、それぞれのプラットフォームが異なる場合があります。このような環境では、クライアントとサーバーは *ミドルウェア* と呼ばれる通信層を通して通信します。

図 2-1 基本的なクライアント サーバー アーキテクチャ

アプリケーションを別のマシンで実行している場合、これらのマシンを何らかの方法 (通常は、LAN、WAN または インターネットなどのネットワーク) で物理的に接続する必要があります。この場合、ネットワークのアーキテクチャとクライアント サーバー アプリケーションのアーキテクチャを区別することが重要です。クライアント アプリケーションは、ネットワーク クライアントか、ネットワーク サーバーで実行します。クライアント アプリケーションとサーバー アプリケーションは、同じマシンで実行することができます。このマシンは、ネットワーク クライアントとネットワーク サーバーのどちらでもかまいません。また、そのどちらでもないこともあります。クライアント サーバー アプリケーションは、そのアーキテクチャの性質上、ネットワークでの実装方法を参照せずに単独で記述します。たとえば、多くの UNIX システムのグラフィック用フロント エンドとして使用される X システムは、クライアント サーバー アプリケーションです。この場合、アプリケーションのサーバー部分をネットワーク クライアント マシンで実行し、アプリケーションのクライアント部分をネットワーク サーバーで実行することがよくあ

ります。アプリケーションのクライアント部分を最も簡単に識別するには、クライアントが常にサービス リクエスタであることを覚えておいてください。

次に、クライアント サーバー アプリケーションの一般的な特徴を示します。

- クライアント プログラムは、複数のサーバー プログラムにサービスを要求することができる。
- クライアント プログラムが、実際にサービスを提供するサブプログラムを認識する必要はない。
- 複数のサブプログラムは、連動してサービスを提供することができる。
- 複数のクライアント プログラムが 1 つのサーバー プログラムからサービスを要求することができる。
- サーバー プログラムは、複数のサービスを提供することができる。
- 通常、サーバー プログラムは、クライアント プログラムを実行しているマシンに対してリモートのマシンで実行される。

COBOL アプリケーションは、CALL 文を使用してサービスを要求します。サービス要求は、実際には、プロシージャで実行される関数呼び出しです。CALL 文には、通常、ローカル関数、つまり、呼び出し側のプログラムと同じマシンで実行されるプロシージャを関連付けます。ただし、CALL 文には、別のマシンで実行されるリモート関数も同様に関連付けることができます。このように CALL 文を使用する場合、*リモート プロシージャ コール*、または RPC と呼ばれます。クライアント サーバー アプリケーションが急速に発達した主な理由は、リモート プロシージャ コールを使用中のネットワーク プロトコルと別に処理できることです。これにより、基礎となるネットワークの処理よりも、アプリケーションのコード作成に集中することができるようになります。NetExpress では、クライアント サーバー結合と呼ばれる簡単な RPC 機構を採用しています。これは、クライアント プログラムとサーバー プログラムの間にネットワークに依存しない直接的な通信層を提供します。

2.1.2 クライアント サーバーの利点

企業向けのアプリケーションにクライアント サーバー アーキテクチャを適用する最大の利点は、柔軟に実装でき、メンテナンスが比較的簡単であるということです。たとえば、クライアント サーバー アーキテクチャを使用すると、通常、次のような利点があります。

- ビジネス ロジックに既存のレガシー コードを再使用できる。
- タスクに最も適したプラットフォームでアプリケーションの各関数を実行できる。
- 処理とネットワークの負荷を分散できる。
- クライアント プログラムやユーザー インターフェイスを変更せずに、ビジネス ロジック プロシージャを素早く、簡単に変更できる。
- アプリケーションや更新をエンドユーザーに簡便に配信できる。

- 同じサーバー側のプログラムに別のクライアント ユーザー インターフェイスを提供できる。
- 互いに機能するように設計された複数の開発ツールを使用できる。

クライアント サーバー アーキテクチャを最も有効に利用するには、いくつかの基本的な設計方針にしたがう必要があります。次にその概要を説明します。

2.2 アプローチ方法の提案

2.2.1 プログラム ロジックの分離

クライアント サーバー アーキテクチャで新しいアプリケーションを作成する場合、または、既存のアプリケーションを更新して変換する場合に、このアーキテクチャを最も有効に利用するには、アプリケーションの機能を論理的に (物理的にも) 分離し、それぞれを区別できるようにすることが重要です。通常のアプローチとしては、アプリケーションの論理的な機能を次の 3 つに分割します。

- ユーザー インターフェイス ロジック (画面処理)
- ビジネス ロジック (データ処理)
- データ アクセス ロジック (ファイル処理またはデータベース処理)

概念的には、これらの 3 つの機能または層は、それぞれ別のプログラムで処理されます。ユーザー インターフェイス ロジックは、常にクライアント アプリケーションにより処理されます。クライアント アプリケーションがユーザー インターフェイス ロジックだけを処理する場合には、*Thin Client* と呼ばれます。一部またはすべてのビジネス ロジックがクライアント アプリケーションにより処理されることもあります。この場合、*Thick Client* と呼ばれます。

クライアント サーバー アプリケーションを作成する場合に、実際のプログラム コードにこのような概念的な機能分割を適用すると便利です。この結果、物理的に分割したプログラムを作成し、3 つの各層を処理することができます。分散コンピューティング環境では、これらの各プログラムは別のマシンで実行されている場合があります。この場合も、プログラムは、同じマシンですべてのプログラムが実行されている場合と同様に、正常に動作します。

Web アプリケーションは、究極の Thin Client です。ユーザー インターフェイスは、ユーザーの Web ブラウザだけで処理されます。インターフェイスの定義は、Web サーバーに保存されている HTML 形式で提供されますが、Web ブラウザの制御により一時的にダウンロードされます。

2.2.2 サポートされているプラットフォーム

NetExpress は、次のプラットフォームで、32 ビットの Web ベースおよびネットワーク ベースのクライアント サーバー アプリケーションを作成できるように設計されています。サポートされているプラットフォームの詳細については NetExpress Readme ヘルプファイルをご覧下さい。

第3章 Web のクライアント サーバー

この章では、Web アプリケーションの基本概念について紹介し、フォームとサーバー側プログラムの役割について説明します。また、使用できる Web ブラウザに応じたフォーム作成方法や、フォームにイベント ハンドラを使用する状況についても説明します。さらに、サーバー側のさまざまなプログラムを紹介し、それらの違いについて説明します。

3.1 WWW の概要

WWW (略して "Web") は、グローバルなインターネットや社内イントラネットを通じて通信するための機構です。Web は、多くの相互リンクされたドキュメントで構成されており、Microsoft インターネット エクスプローラや Netscape ナビゲータなどの Web ブラウザを使用してアクセスし、表示することができます。これらのほとんどのドキュメントは、NetExpress のヘルプシステムのように、ホットスポットのあるテキストやグラフィックスを含むページで構成されています。ホットスポットをクリックすると別のページに移動し、さらに、そのページで興味のある話題に合わせてホットスポットをクリックすることができます。各ホットスポットは *uniform resource locator* (URL) と関連付けられているため、Web ブラウザは、ホットスポットのリンク先を判別することができます。URL は、配信機構 (*方式*と呼ばれる) と標準形式の文書のアドレスを簡単に指定する方法です。たとえば、`http://www.microfocus.com/Compilations/index.htm` のように記述します。
この例では、方式は `http:` で、アドレスは `//www.microfocus.com/Compilations/index.htm` です。

技術情報: インターネットとイントラネット

インターネットは、世界規模のコンピュータ ネットワークです。インターネットには、400 万台以上のコンピュータがさまざまなオペレーティング システムを使用して接続されていると推測されます。インターネットは、特定の組織や国が所有するものではありません。ところが、接続している各コンピュータが同じ通信プロトコル TCP/IP を使用して他のすべてのコンピュータと通信しているため、インターネットは機能することができます。TCP/IP は、Windows 95 と Windows NT の両方でサポートされています。

インターネットでは、情報は、宛先アドレスを持つ自己格納式の小さなチャンクやパケットで渡さます。各コンピュータは、パケットの宛先アドレスにしたがってパケットを転送します。その結果、パケットは、1 つのコンピュータから別のコンピュータに「ホップ」しながら移動します。コンピュータまたはネットワークからインターネットへの最初の「ホップ」は、インターネット サービス プロバイダを通して行われます。インターネット サービス プロバイダは、専用回線やダイアルアップ回線接続を使用してサービスを提供します。

インターネットに接続するには、ネットワーク通信に TCP/IP プロトコルを使用できることが前提になります。これは、社内ネットワークがインターネットと同様に機能し、それに接続されているコンピュータの間で情報のパケットを受け渡しすることを意味します。ネットワークをこのように設定し、標準的なインターネット アプリケー

ション (Web ブラウザなど) を社内ネットワークで使用できるようにした場合、このネットワークをイントラネットと呼びます。

ネットワークをグローバルなインターネットにアクセスできるように設定した場合、自動的にイントラネットとして機能します。つまり、何もせずにイントラネット機能を使用できるわけです。これは、イントラネットが企業の通信手段として重要になった理由の 1 つです。

Web ドキュメントは、テキストやグラフィックスで構成する必要はありません。ダウンロードできるファイルのディレクトリ、アニメーション、ムービー、オーディオの断片でもかまいません。各リンクをクリックすると、選択したドキュメントに該当するファイルがコンピュータの Web ブラウザにダウンロードされ、そのファイルが表示されるか、ファイルを実行したり、再生するためのヘルパー アプリケーションが呼び出されます。

ドキュメントはすべて標準言語である *Hypertext Markup Language* (HTML) を使用してコード化されているため、Web ブラウザは多くの種類のドキュメントの処理方法を認識することができます。HTML は、簡単なテキスト マークアップ言語で、ファイルのレイアウト上必要な箇所で標準テキストの先頭と末尾をマークアップ タグで囲みます。HTML ファイル例とこのファイルを Web ブラウザで開いたときに表示される結果を次に示します。

```
<html>
<head>
<title>Web ページ例</title>
</head>
<body>
<h1>HTML 例</h1>
<img src="sgfrntlo.gif">
<p>
これは、グラフィックとテキストを含む簡単な Web ページの例です。
</p>
</body>
</html>
```

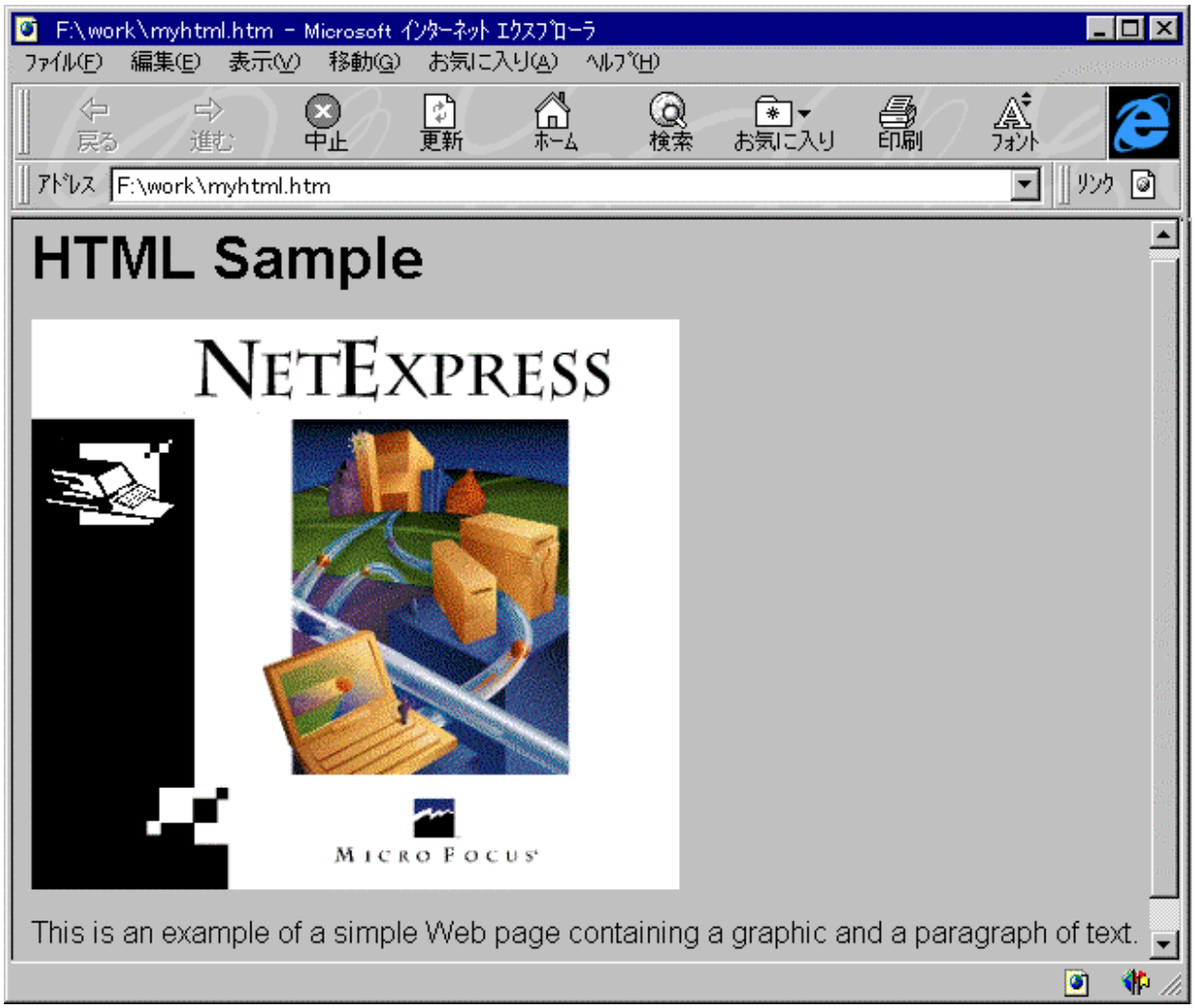

図 3-1 Web ブラウザによる HTML ページの表示例

3.2 Web アプリケーションの構成

Web ドキュメントには、フォームと呼ばれる形式があります。これは、Web ブラウザで表示できる標準的な HTML ページです。フォームは、グラフィカル ユーザー インタフェースで使用するコントロールを含めることができるため、テキストとグラフィックスで構成されたページとは異なります。たとえば、入力フィールド、オプション ボタン、チェック ボックスなどを使用することができます。フォームは、ユーザーとの対話手段として、ユーザーからのデータ入力を受け付けたり、選択内容を Web サーバーで実行されているアプリケーションに渡したりすることができます。HTML フォームの例を図 3-2 に示します。

インターネットやイントラネットで実行される Web アプリケーションのユーザー側からは、Web アプリケーションは Web ブラウザで表示された HTML ページとフォームの組み合せに見えます。ページのリンクをクリックすると、Web サーバーのプログラムが起動します。プログラムは、ユーザーが対話に使用する HTML フォームをダウンロードします。

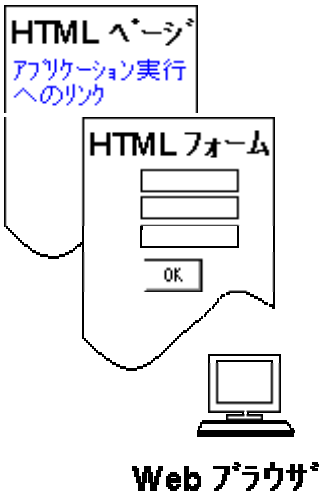

図 3-2 Web ブラウザによる Web アプリケーションの表示例

Web アプリケーションは次の構成要素で構成されています。

- サーバー側のプログラムを起動する URL へのリンクをもつ HTML ページ
- エンドユーザーが照会や更新を入力し、データを表示するフォーム
- COBOL サーバー側のプログラム

複雑なアプリケーションは、 HTML フォームの URL により互いにリンクされた、多くの小さなアプリケーションで構成されています。

3.2.1 実行フロー

通常の Web アプリケーションは、次のように実行されます。

1. ユーザーは、Web ブラウザを使用して Web ページにアクセスし、ホットスポットをクリックして Web サーバーにアプリケーションを要求します。
2. アプリケーションは、サーバーで実行を開始し、アプリケーションの HTML フォームを返します。
3. ユーザーは、フォームに記入し、[送信] ボタンをクリックします。
4. サーバー側のプログラムは、ファイルやデータベースからデータを取り込むなどしてデータ処理を実行し、結果をユーザーのブラウザに返します。結果は、新しいフォームまたは HTML ページとして、入力または出力を行った同じフォームに表示することができます。

Web アプリケーションの大きな利点の 1 つは、プログラミング時にミドルウェアや通信プロトコルを気にする必要がないことです。ミドルウェアや通信プロトコルは、すでに既存のインターネット基盤に組み込まれています。その

ため、Web アプリケーションは、比較的早く、簡単に開発や実装を行えます。

3.3 課題: ユーザー インターフェイスの作成

上記で説明するとおり、Web アプリケーションのユーザー インターフェイスは、HTML フォームで構成されており、Windows アプリケーションで使用されているコントロールに類似したコントロールを含みます。

Web ブラウザは、フォームのコントロールを実際に機能させる方法を指定します。たとえば、各オプション ボタン セットでは、一度に 1 つのオプション ボタンしか選択できないように制限されます。また、入力したデータをリストボックスに自動的に表示します。ユーザーが必要なデータをすべてフォームに入力し、[送信] ボタンをクリックすると、入力された情報が Web サーバーに返されます。

サーバー側のプログラムがフォームで送信されたデータを処理すると、Web ブラウザでサポートされているどのタイプのドキュメントでも返すことができます。たとえば、結果を含む通常の Web ページ、または、さらに情報を要求する別の HTML を返すこともできます。CICS と IMS のようなシステムを使用したことのある方は、これらのシステムと Web アプリケーションの類似性にすぐに気付かれることでしょう。CICS と IMS のようなシステムと Web アプリケーションでは、すべて、バッチ処理システム、または、トランザクション処理システムを使用しており、データは個別のチャンクに分けられてユーザー インターフェイスと送受信されます。データのチャンクを受け渡すとき以外は、クライアントとサーバーの間で通信は行われません。

Web アプリケーションのユーザー インターフェイスは、非常に簡単に作成することができます。Form Designer で必要なフォームをペイントするだけです。Form Designer は、フォームをマークアップする HTML ファイルと、コピー ファイルを作成します。このコピー ファイルには、フォームからデータへアクセスする場合にプログラムで必要となるデータ定義が含まれます。既存のアプリケーションがある場合には、インターネット アプリケーションウィザードを使用して簡単な HTML フォームを作成してから、Form Designer を使用してフォームを希望の形式に変更してください。スクラッチから Web アプリケーションを作成する場合には、Form Designer を直接使用して新しいフォームを作成します。

HTML フォームには、HTML フォーム コントロールだけでなく、ActiveX コントロールや Java アプレットも追加することができます。ActiveX コントロールや Java アプレットは、より対話型の要素をフォームに加えるための方法です。詳細については、『*インターネット アプリケーション*』オンライン マニュアルの 「フォームと HTML 」の章を参照してください。

Form Designer を使用すると、フォームに対して異なる 2 種類の出力を作成することができます。

- クロスプラットフォーム HTML
- ダイナミック HTML

クロスプラットフォーム HTML のフォームは、より多くの種類の Web ブラウザで機能します。 クロスプラットフォーム HTML を使用すると、コントロールとラベルの順序やおおよその位置を指定することができますが、フォームの正確な表示形式を指定することはできません。これは Web ブラウザで制御されます。一方、ダイナミック

HTML は、より正確な位置を指定できますが、インターネット エクスプローラ 4.0 以降 でしかサポートされていません。

3.3.1 ソリューション: クロスプラットフォーム出力またはダイナミック HTML を使用した HTML フォーム

Form Designer を使用すると、Web アプリケーションのグラフィカル ユーザー インターフェイスをペイントしたり、サーバー側のプログラムに対応するスケルトン コードを自動生成することができます。インターフェイスをペイントすると、イベント ハンドラを追加し、必要なクライアント側の処理を実行するすることができます。(詳細については、「イベント ハンドラの追加」を参照してください)。

クロスプラットフォーム HTML を指定すると、Form Designer は最近の Web ブラウザで表示できる HTML フォームを作成します。ダイナミック HTML を指定すると、Form Designer はインターネット エクスプローラ 4.0 でのみ表示できる HTML フォームを作成します。

手順の概要は、次のとおりです。

- フォームベースのプロジェクトを作成します。
- HTML ページウィザードを完了します。
- フォームをデザインします。コントロールとラベルを追加し、プロパティを変更します。
- フォームを保存します。
- コントロールに必要なイベント ハンドラをセットアップします。
- フォームを保存します。
- Form Designer を閉じます。
- [インターネット アプリケーション ウィザード] を使用して、サーバー側のスケルトン プログラムを生成します。
- 適宜、プロジェクトの設定を変更します。
- サーバー側のプログラムにビジネス ロジックを追加します。
- デバッグします。

3.3.1.1 例

シナリオ「Gas Services 社」の第 1 段階 (詳細については、「*はじめに*」の章の「シナリオ例」の項を参照) につい

て、サーバー側の CGI プログラムを使用する Web アプリケーション例を示します。このアプリケーションは、WWW を使用してユーザーがアクセスできるフォームから構成されています。フォームには、ユーザーのアカウント番号と新しいメーターの読み取り値を入力するためのフィールドがあります。ユーザーが [送信] ボタンをクリックすると、メーターの読み取り値が Gas Services 社のデータベースで更新され、確認用のページがユーザーに返されます。このアプリケーションの入力フォームのプロトタイプは、次のとおりです。

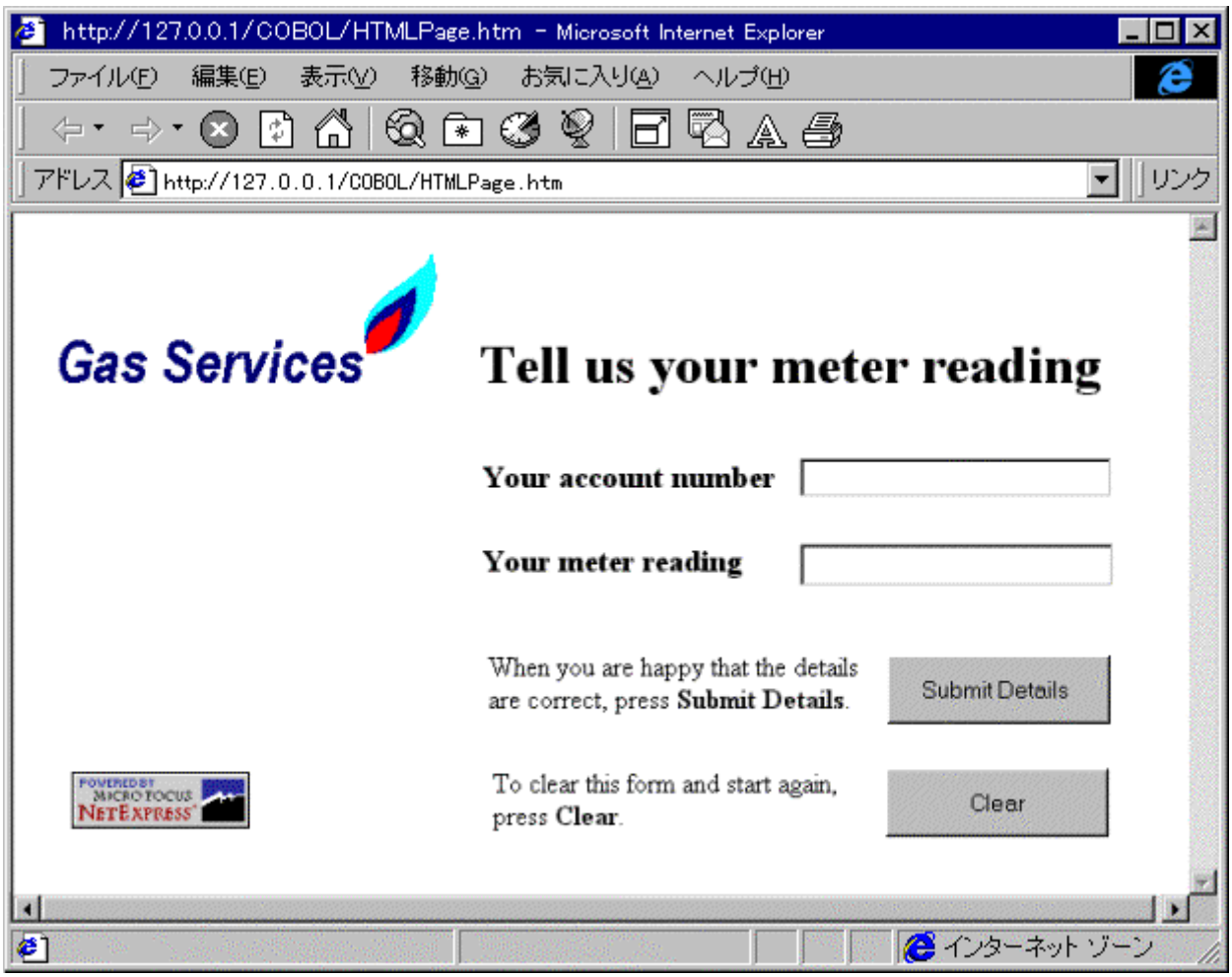

図 3-3 Gas Services 社の Web アプリケーションで使用する入力フォームのプロトタイプ

上記のフォームを Form Designer で作成する方法は、次のとおりです。

1. 新規プロジェクトを作成します。「プロジェクトの新規作成」ダイアログ ボックスで、空プロジェクトを選択します。

2. 新規 HTML フォームを作成します。 この場合、ダイナミック HTML フォームではなく、クロスプラットフォーム (表出力) を選択します。このアプリケーションは、Gas Services 社の顧客が使用することを前提としており、顧客がインターネット エクスプローラ 4.0 を持っていない可能性があるためです。

3. Form Designer で、入力フォームをデザインします。各コントロールを追加するには、最上部のツールバー

で該当するアイコンをクリックしてから、フォームでコントロールを配置する位置をクリックします。次に、右下のペインでコントロールのプロパティを作成します。この操作結果が Form Designer でどのように表示されるかを次に示します。

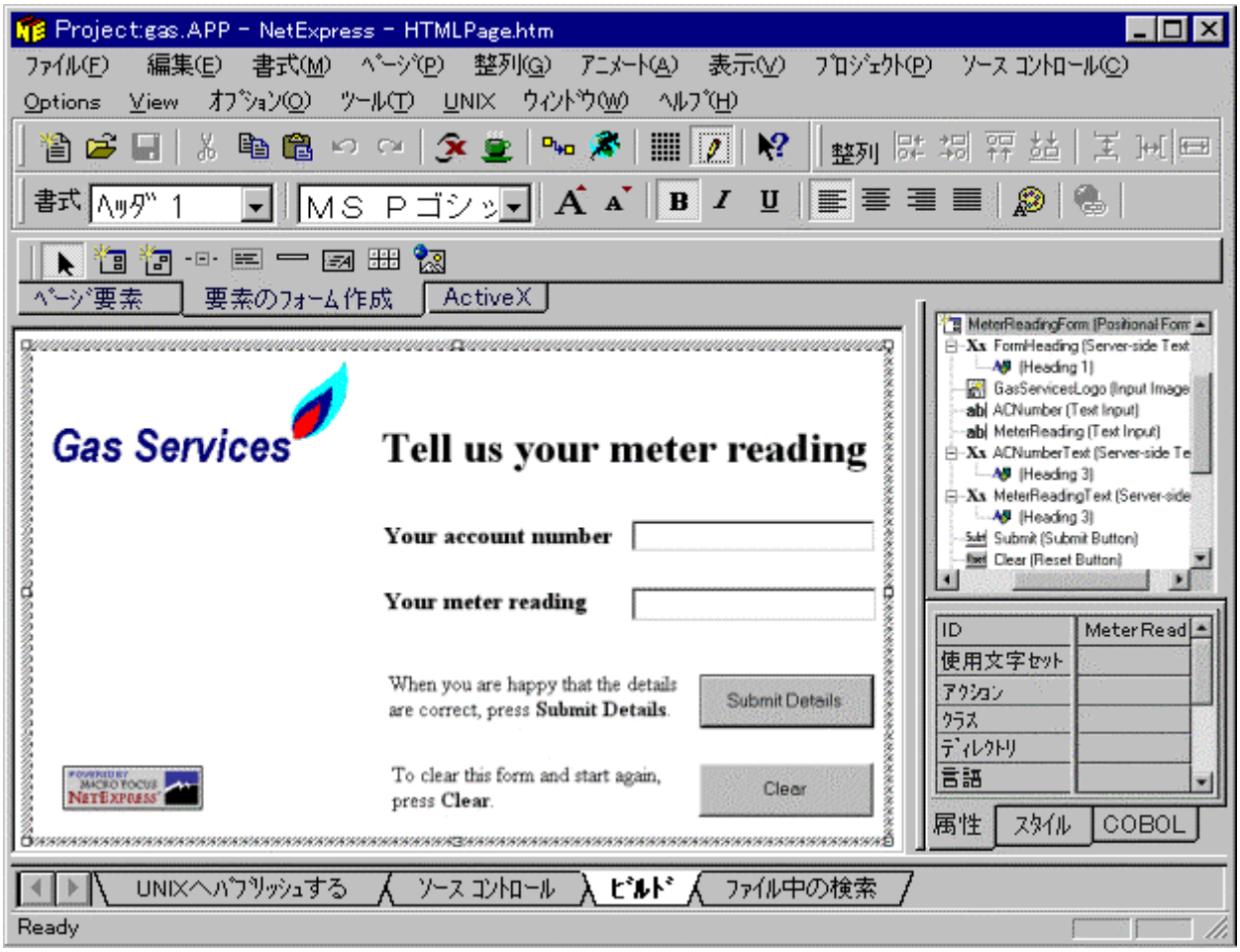

図 3-4 Form Designer で作成した Gas Services 社の入力フォーム

4. フォームを保存します。[保存] をクリックすると、Form Designer がプロジェクトのフォームを更新します。

3.3.2 イベント ハンドラの追加

イベント ハンドラは、クライアントで実行され、フォームのイベント (マウスによるクリックなど) を処理するプログラム コードです。イベント ハンドラは、Win32 API アプリケーションのメッセージ ループ、または、Object COBOL アプリケーションや Panels V2 アプリケーションのイベント マネージャに似たタスクを実行します。NetExpress アプリケーションでは、フォームのイベント ハンドラは JavaScript で作成されています。Form Designer には、多くの簡単な条件に対してコードを生成することができる Script アシスタント が添付されているため、必ずしも JavaScript を学習する必要はありません。ただし、より高度なクライアント側の処理を行うためには JavaScript も一部学習することが必要になります。

質問: イベント ハンドラは必要ですか。
Web アプリケーションでは、データのチャンク (フォーム全体または HTML ドキュメントから構成されている) を受け渡しするとき以外は、クライアントとサーバーの間で通信が行われることはありません。フォームを 1 つの単位として処理する標準的なフォームベースのアプリケーションを作成する場合、イベント ハンドラは必要ありません。このようなアプリケーションとしては、埋め込み HTML を使用するインターネット アプリケーション ウィザードで作成されたアプリケーション、手動でコード化され、埋め込み HTML、拡張 ACCEPT 構文と拡張 DISPLAY 構文を使用できる Web アプリケーションなどが挙げられます。 HTML フォームでフォーム自体の一部を処理する場合は、イベント ハンドラが必要です。このような処理は、クライアント側でローカルに行う必要があります。たとえば、フォームにデータを入力するときにそのデータを検証する場合 (たとえば、編集フィールドの数字が特定の範囲内であることをチェックするため)、この処理はイベント ハンドラで行う必要があります。同様に、ユーザーが特定のチェック ボックスを選択したときにフォームの一部を灰色で表示するには、イベント ハンドラをセットアップする必要があります。イベント ハンドラを使用すると、このような処理に必要な対話操作が可能になります。

コントロールにイベント ハンドラを追加するには、Form Designer の Script アシスタント を使用します。Script アシスタント では、フォームで特定のイベントが発生したとき (プッシュ ボタンがクリックされるなど) に、どのようなアクションを実行するかを指定できます。Script アシスタントの使用方法の詳細については、『*インターネット アプリケーション*』オンライン マニュアルの「フォーム プログラミング」の章を参照してください。

3.4 課題: 既存のユーザー インターフェイスの移行または更新

3.4.1 ソリューション: インターネット アプリケーション ウィザード

インターネット アプリケーション ウィザード」を使用して、元のアプリケーションの変更やリコンパイルを行わずに、既存のレガシー アプリケーションから Web アプリケーションを作成します。インターネット アプリケーション ウィザードを使用すると、データ定義を連絡節に正しく記述したアプリケーションについて、必要なフォームを作成することができます。つまり、事実上、ユーザー インターフェイスからビジネス ロジックを分離した、ほとんどの既存のクライアント サーバー アプリケーションに対して、フォームを作成できることになります。たとえば、CICS と IMS プログラム、スクリーン節を使用するほとんどのプログラム、Visula Basic ユーザー インターフェイスをもつアプリケーションなどが対象となります。

この方法では、元のアプリケーションが変更されないため、移行期には、ユーザーはどちらのインターフェイスでも選択することができます。そのため、新しいインターフェイスを徐々に普及させることができます。または、ライブ プロダクション アプリケーションで小さなユーザー グループと新しいインターフェイスを試すことができます。この

場合、元のインターフェイスを引き続き使用している他のユーザーに影響を与えることはありません。

手順の概要を次に示します。

- プロジェクトを作成します。
- レガシー アプリケーションにプロジェクトをビルドします。
- インターネット アプリケーション ウィザードを開きます。
- フォームを生成する.cbl ファイルを選択します。
- コントロールのプロパティ (データ フローのラベル、コントロールの種類、方向) を設定します。
- フォームを生成します。

インターネット アプリケーション ウィザードを使用してフォームを作成すると、このフォームを Form Designer にインポートして、表示状態を最終調整することができます。

このようなインターネット アプリケーション ウィザードの使用方法の詳細については、『*インターネット アプリケーション*』オンライン マニュアルの「レガシー コードの再使用」を参照してください。

3.5 課題: Web サーバー側のプログラム

ほとんどの処理は、Web アプリケーションのサーバー側プログラムで行われます。通常、サーバー アプリケーションはクライアントからのデータ処理をすべて処理し、COBOL ファイルまたはリレーショナル データベースなどのデータ記憶域とのインターフェイスとなるだけでなく、Web サーバーと通信することができます。COBOL プログラミングを経験している場合、ビジネス ロジックの処理とデータとのインターフェイスは、おそらくなじみ深いテーマでしょう。むしろ、Web サーバーとのインターフェイスの方が、新しいテーマかもしれません。Web サーバーとのインターフェイスは、次の 3 つの主要なインターフェイス定義のどれかを使用すると簡単に作成することができます。

- CGI (Common Gateway Interface)
- ISAPI (Internet Server Application Programming Interface)
- NSAPI (Netscape Server Application Programming Interface)

これら 3 つのインターフェイス規格については、この章と『*インターネット アプリケーション*』オンライン マニュアルの「CGI、ISAPI、および、NSAPI プログラム」の章で詳しく説明します。3 規格の使用結果はすべて似ていますが、処理過程がそれぞれ異なります。NetExpress を使用すると、ビルド オプションをいくつか変更するだけで、これらのインターフェイス規格に準拠したサーバー側のプログラムを簡単に作成できます。

簡単な COBOL Web アプリケーションには、次の 3 つの構成要素があります。

- HTML 入力フォーム

  入力フィールドやデータ入力のためのコントロール。[送信] ボタンをクリックすると、データが Web サーバーに送られ、サーバー側プログラムを起動します。

- COBOL サーバー側プログラム

  ACCEPT 文を使用してフォームからデータを受信し、DISPLAY 文を使用して出力ページを送信します。2 つの新しい句を使用すると、グループ データ項目を HTML フォームと HTML ページに、また、基礎データ項目を HTML フォームと HTML ページにある名前付きパラメータに結合できます。

- 出力ページ

  サーバー側のプログラムからのデータに対するプレースホルダとして置換可能なパラメータをもつ HTML ページ。

ほとんどの Web アプリケーションは、エンドユーザーが HTML フォームでコントロールをクリックすると起動します。コントロールは、[OK] というラベルのプッシュ ボタンや、HTML ハイパーリンクなどです。Web アプリケーションがどのように表示されるかわからない場合には、『始動マニュアル』のセッション例を実行してください。

エンドユーザーがフォームを送信すると、フォームに入力されたすべての情報は 1 つのストリームにパッケージ化され、サーバー側のプログラムを実行するための命令とともに Web サーバーに返信されます。サーバー側のプログラムは、ストリームのパッケージを解除し、ユーザー入力を処理してから、結果を返します。次の図は、Web アプリケーションの実行方法を示します。

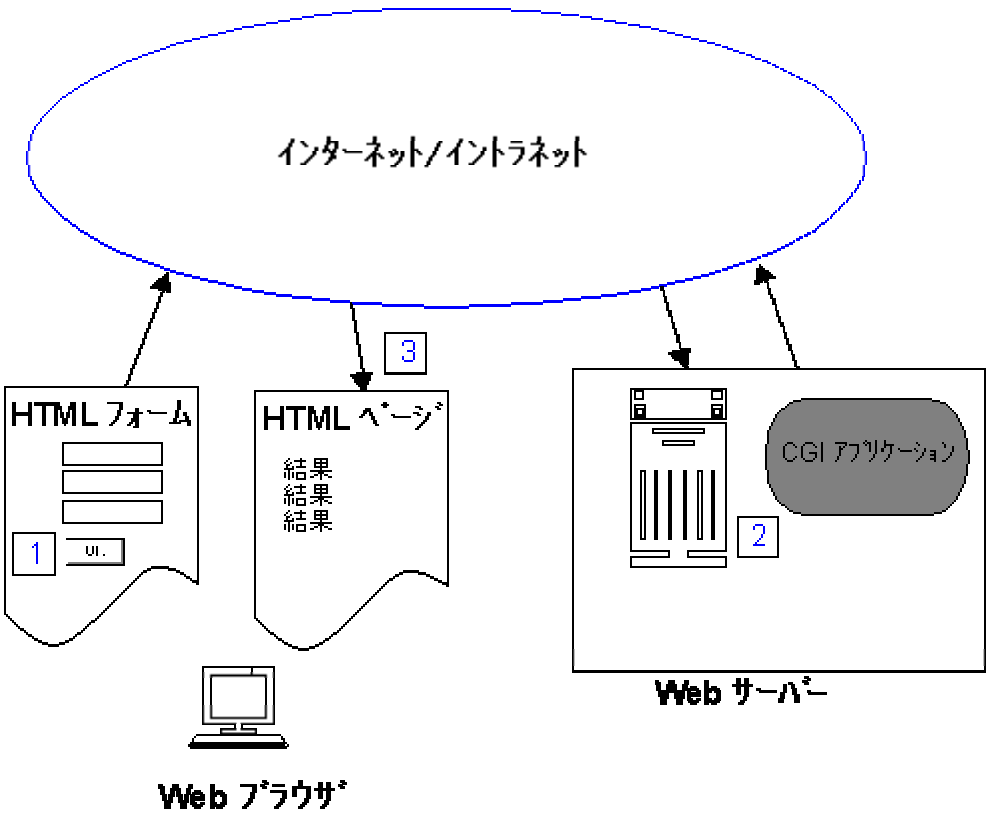

図 3-5 Web アプリケーションのサーバー側プログラムの役割

1. エンドユーザーは、HTML フォームに記入し、[フォームを送信] ボタンをクリックします。フォームのフィールドに入力された情報は、1 つのデータ ストリームにまとめられ、Web サーバーに送られます。データ ストリームは、ここで、サーバー側プログラムを実行するための要求として解釈されます。

2. Web サーバーは、要求されたサーバー側プログラムを起動し、データ ストリームに返します。

3. サーバー側プログラムは、実行され、HTML ページを Web ブラウザに返すと、終了します。

ほとんどの言語では、サーバー側プログラムでデータ ストリームを解析するためのコードを作成する (または検索する) 必要があります。NetExpress には、よく使用される ACCEPT 文と DISPLAY 文によるサーバー側のサポートが組み込まれています。NetExpress COBOL のサーバー側プログラムは、入力フォームを ACCEPT し、出力フォームを DISPLAY することができます。

3.5.1 ソリューション 1: CGI プログラム

Common Gateway Interface (CGI) は、National Center for Supercomputing Applications (NCSA) によって考案された、サーバー側のプログラムと Web サーバーの間で使用するインターフェイス規格です。CGI では、Web サーバーを使用して、クライアントの Web ブラウザの Web ページとデータを送受信できるサーバー側プログラムをビルドすることができます。また、COBOL データ ファイルやリレーショナル データベースなどの外部データ記憶域と通信す

ることもできます。CGI インターフェイスは、今日、Web のサーバー プログラムで最も一般的に使用されているインターフェイスです。

技術情報: CGI の実装

ユーザーが Web ページのホットスポットをクリックすると、関連付けられた URL 要求が Web サーバーに送信されます。要求された URL が特別な構成済みのディレクトリ (通常は、cgi-bin と呼ばれる) のファイルを指す場合、Web サーバーは、この URL を、サーバー側プログラム (この場合は、CGI プログラム) としてファイルを実行するコマンドとして解釈します。CGI プログラムは、コマンド行から実行されたかのように、CGI プログラム自体のプロセスで実行されます。Web サーバーは、CGI プログラムのプロセスの環境でセットアップする環境変数を使用して、CGI プログラムにデータを渡します。CGI プログラムの出力は、標準的な出力ストリーム stdout を使用して、HTTP データ ストリーム形式で Web サーバーに返されます (HTTP は、Web ブラウザが認識する標準的な符号化方式です)。Web サーバーは、データ ストリームをユーザーのブラウザに返します。返されたデータ ストリームは、HTML ドキュメントとして表示されます。

詳細については、次の URL を参照してください。
http://www.microfocus.com/NetExpress/nxbooks/links.htm#cgi1

3.5.1.1 長所と短所

長所	短所
すべての Web サーバーでサポートされる。	サーバーのリソース使用が非効率的である。
保護違反が発生しても Web サーバーに影響がない。	Web との通信が制限される。
汎用性がある。	各インスタンスは、クライアント要求を 1 つしか処理できない。

3.5.1.2 プログラミングに際しての留意点

CGI によるサーバー側プログラム作成の詳細については、『*インターネット アプリケーション*』オンライン マニュアルの「サーバー側プログラミング」の章を参照してください。

3.5.1.3 例

シナリオ「Gas Services 社」の第 1 段階について (詳細については、「シナリオ例」を参照してください)、サーバ

ー側の CGI プログラムを使用する Web アプリケーション例を示します。Form Designer を使用して入力フォームを作成する方法の詳細については、「HTML フォーム」の項を参照してください。

ユーザーがフォームを送信すると、データベースが更新され、確認ページがユーザーに返されます。確認ページは、次のように表示されます。

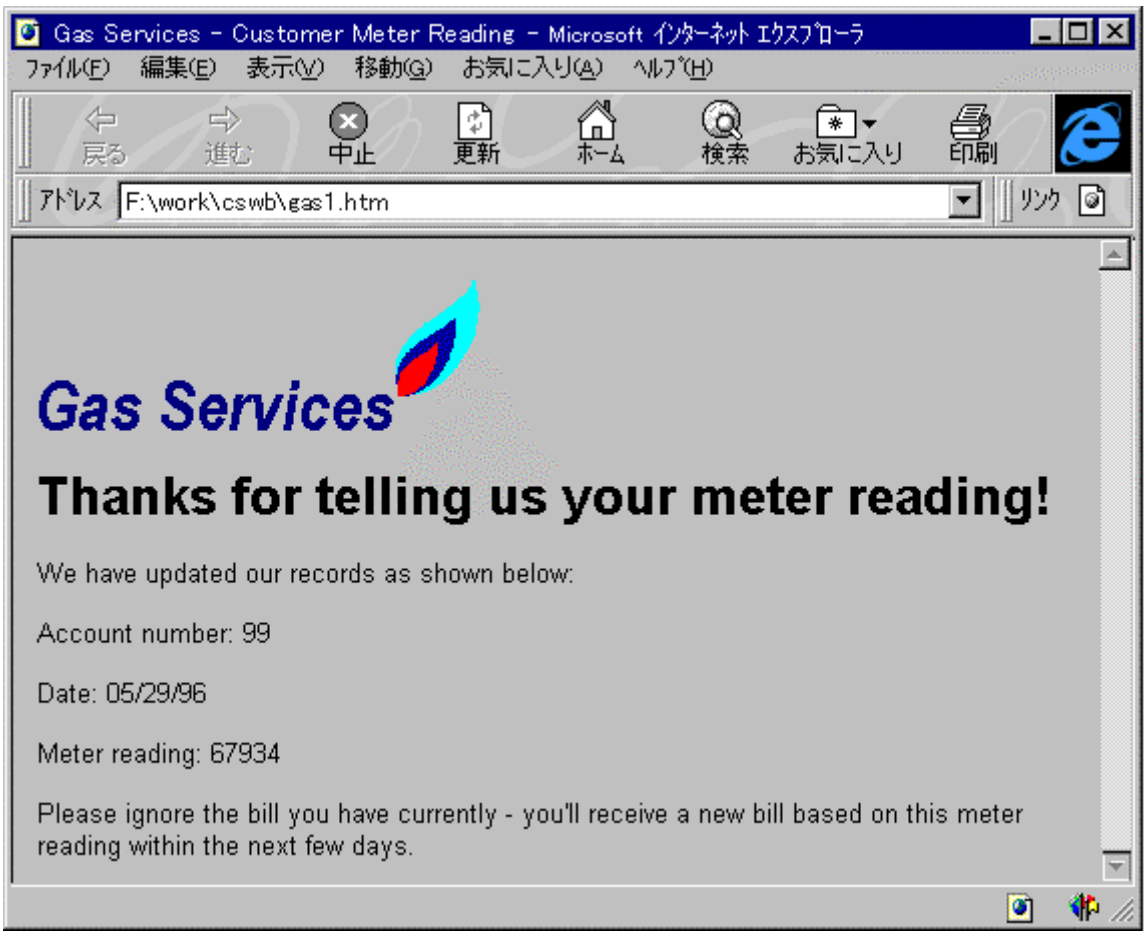

図 3-6 Gas Services 社の Web アプリケーションにより表示された HTML 出力ページのプロトタイプ

アプリケーションから返されるフォームが、入力に使用されるフォームと同じではない場合、非対称アプリケーションと呼びます。このアプリケーションは、非対称アプリケーションです。実際には、入力フォームは、フォームではなく通常の HTML ページです。(対称アプリケーションと非対称アプリケーションの詳細については、『*インターネット アプリケーション*』オンライン マニュアルの「より複雑なアプリケーション」を参照してください)。コード作成を開始する前に、対称アプリケーションと非対称アプリケーションのどちらを作成するか決めることが重要です。インターネット アプリケーションウィザードでは、アプリケーションの種類によって選択内容が異なります。インターネット アプリケーション ウィザードが作成するスケルトンに追加するコードもアプリケーションの種類によって異なります。

上記のアプリケーションのプロトタイプは、次のように作成します。

1. プロジェクトを開いたまま、空フォームとクロスプラットフォーム (表出力) を選択して、gas1 という HTML ページを新規に作成します。

2. Form Designer で、出力ページを設計します。テキストを入力し、CGI プログラムからのデータ出力を表示する場所にテキスト入力コントロールを作成します (これらは、エンドユーザーの入力ではなく CGI プログラムにより設定されます)。使用する「変数名」プロパティと COBOLPicture プロパティは、次のとおりです。

変数名	COBOLPicture
ACNumber	X(4)
Outputdate	X(10)
MeterReading	X(5)

   もう一つの方法は、HTML エディタでページを設計してから、Form Designer でテキスト コントロールを追加する方法です。

3. [ファイル] メニューで [新規作成] を選択してから、[インターネット アプリケーション] をクリックします。

4. サーバー プログラム」を選択して、既に作成した HTML ページから新しい CGI プログラムを作成します。

5. サーバー側プログラムの名前 (mycgi.cbl) を入力し、先に設計しておいたフォームを入力フォームとして指定します。また、ここで設計したページを出力フォームとして指定します。

   ウィザードで [完了] をクリックすると、CGIプログラムが生成され、プログラムに追加されます。

6. インターネット アプリケーション ウィザードが生成したサーバー側のスケルトン プログラム mycgi.cbl を開きます。

7. フォームの [送信] ボタンが押されたかどうかを確認するコードからコメント文字を削除します。

```
*>-----------------------------------------------------------------
*> CGI が入出力に同じフォームを使用する場合、
*> 「初期読み込み」を処理する必要があります。
*> 最も簡単にこの処理を行う方法は、次のとおりです。
*> より優れた方法としては、非表示フィールドを使用します。
*>
*> CGI が実行されているかどうかを最初に確認するには、
```

```
*> 空白文字を使用して［送信］ボタン（デフォルト名は ssubmit）
*> をテストします。ブラウザによりフォームが返されると、
*> ［送信］ボタンはそのキャプションを返します。
*> 必要に応じてコメントを削除してください。
*>

    perform process-form-input-data
    perform convert-input
    if ssubmit = spaces
*> コードを追加し、「空」フォームを初期化します。
      perform HTMLForm-cvt
      perform HTMLForm-out
    else
      perform process-business-logic
      perform HTMLForm-cvt
      perform HTMLForm-out
    end-if
    exit program
    stop run.
```

コメント削除が必要な理由は、Gas Services 社の顧客がフォームを使用するたびに CGI プログラムが 2 回実行されるためです。1 回目に CGI プログラムが実行されるのは、Web サーバーの /cgi-bin/ ディレクトリへのリンクをユーザーがクリックした場合です。この段階で CGI プログラムが行う処理は、入力フォームの初期化と、ブラウザへの入力フォームの送信だけです。次に、ユーザーは、フォームに記入し、[送信] ボタンをクリックします。2 回目に CGI プログラムが実行されるのは、この時点です。この場合、CGI プログラムは、データ処理を行ってから、出力ページをブラウザに返す必要があります。

ユーザーが [送信] ボタンをクリックすると、ブラウザは、ssubmit と呼ばれるデータ項目を返します。この ssubmit には、[送信] ボタンに表示されるキャプションのテキストが含まれます。このデータ項目をテストしたときに、データ項目がすべて空白文字である場合、ユーザーが [送信] ボタンをクリックしていないことになります。この場合は、プログラムが Web サーバーで起動されているはずなので、入力フォームを表示する必要があります。ssubmit がボタン キャプションのテキストを含む場合 (空白文字以外がある場合)、ユーザーが [送信] ボタンをクリックしたと考えられるため、データベースに問い合わせ、出力フォームを返す必要があります。

8. 出力フォームが表示されるプログラムの節を編集して、出力ページを表示します。

```
perform process-business-logic
perform HTMLForm-cvt
exec HTML
  copy "gas1.htm".
end-exec
end-if
exit program
stop run.
```

9. ビジネス ロジックを含むモジュールの呼び出しを追加します。

この呼び出しは、ユーザーが入力した情報を表に反映させて更新する SQL 文で構成されます。SQL 文の後には、情報をf-ACNumber、Outputdate、f-MeterReading と呼ばれるデータ項目としてプログラムに返す問い合わせが続きます。出力 HTML ページのテキスト入力フィールドにこれらの名前を使用しているため、データをそこへ移動する必要はありません。(ここで使用されるビジネス ロジックの詳細については、「データアクセス」の章を参照してください)。

```
*>-----------------------------------------------------------------
 process-business-logic Section

*> アプリケーションのビジネス ロジックをここに追加します。
*> ビジネス ロジックは別のモジュールに含め、この CGI から
*> 次のような方法で呼び出すことをお勧めします。
*>   call "metersql" using FormFields
*> その場合、ビジネス ロジック モジュールの連絡節で
*> この CGI に対して 'cpy' コピーブックを使用します。
*>
*> これには次のような利点があります。
*>  - ビジネス処理と表現ロジックを別々に維持できる。
*>  - そのため、これらを独立して機能させることができる。
*>  - アプリケーションが Unix サーバーに移動された場合、
*>    ビジネス ロジックをアニメートすることができる。
 call "metersql" using FormFields
```

```
exit.
```

10. [プロジェクト] メニューで [すべてをリビルド]をクリックします。

3.5.2 ソリューション 2: ISAPI プログラム

Internet Server Application Programming Interface (ISAPI) は、Microsoft が考案した、サーバー側のプログラムと Microsoft Internet Server for Windows NT の間の呼び出しインターフェイス規格です。ISAPI を使用すると、動的な読み込みが可能で Web サーバーを効果的にカスタマイズするプログラムをビルドすることができます。そのため、HTML クライアントからの要求に対話型で応答し、ログ処理またはセキュリティ システム、データベースまたはトランザクション処理システムのような他のアプリケーションと通信することができます。また、WWW アプリケーションで使用される HTTP データ ストリームのフィルタや暗号化システムを作成するためにも使用できます。

技術情報: ISAPI の実装
ユーザーが Web ページのホットスポットをクリックすると、関連付けられた URL 要求が Web サーバーに送信されます。要求された URL が特別な構成済みのディレクトリ (通常は、cgi-bin と呼ばれる) のファイルを指す場合、Web サーバーは、この URL を、サーバー側プログラム (この場合は、ISAPI プログラム) としてファイルを実行するコマンドとして解釈します。ISAPI プログラムは、ダイナミック リンク ライブラリとしてビルドされます。ISAPI プログラムの実行要求を受け取ると、Web サーバーは DLL をメモリに読み込み、標準のエントリ ポイントでその DLL を呼び出します。ISAPI プログラムでは、必要な関数をどれでも実行することができます。その場合、拡張制御ブロック (ECB) と呼ばれる標準化されたデータ構造体とコールバック関数を使用して、Web サーバーからデータを取得したり、データを返したりします。ISAPI プログラムは、ダイナミック リンク ライブラリなので、Web サーバーと同じプロセスで実行され、メモリにコピーが 1 つあるだけで、複数の同時並行のクライアント要求に対して、件数に関係なくサービスすることができます。
ISAPI アプリケーションは、ダイナミック リンク ライブラリとして実装されるため、再入力可能またはスレッドセーフになるように作成する必要があります。NetExpress では、単純なプログラミング方法とビルド方法を使用して、この作業を簡単に行うことができます。 詳細については、次の URL を参照してください。 http://www.microfocus.com/NetExpress/nxbooks/links.htm#isapi1

3.5.2.1 長所と短所

長所	短所
リソース効率が高い。	Windows NT の MS インターネット サーバーでしかサポートされていない。
応答時間が短い。	保護違反を避ける必要がある。
大量のデータ読み込みが可能である。	CGI アプリケーションより汎用性が低い。
CGI アプリケーションより柔軟性が高い。	

3.5.2.2 プログラミングに際しての留意点

別のアプリケーション (この場合は Microsoft Internet Server) が制御するダイナミック リンク ライブラリをデバッグするのは困難です。そのため、サーバー側のプログラムを CGI プログラムとして作成し、デバッグしてから、ビルド設定を変更して、最終的に ISAPI ダイナミック リンク ライブラリを作成することをお勧めします。必要なビルド変更の詳細については、『*インターネット アプリケーション*』オンライン マニュアルの「CGI、ISAPI、および、NSAPI プログラム」の章を参照してください。

3.5.3 ソリューション 3: NSAPI プログラム

Netscape Server Application Programming Interface (NSAPI) は、Netscape が考案した、サーバー側プログラムと Netscape web サーバーの間の呼出しインターフェイス規格です。NSAPI を使用すると、動的な読み込みが可能なプログラム「プラグイン」を作成することができます。プラグインでは、Web サーバー動作のカスタマイズ、ブラウザからのクライアント要求に対する対話型応答、データベース、ログ処理、セキュリティ システムとのインターフェイス作成などが可能です。ISAPI プログラムと異なり、NSAPI プログラムは、Web サーバーの関数を呼び出すだけではなく、逆に Web サーバーから呼び出されることもあります。Web サーバーの基本的な動作を変更する場合に、これは便利です。

技術情報: NSAPI の実装

ユーザーが Web ページのホットスポットをクリックすると、関連付けられた URL 要求が Web サーバーに送信されます。要求された URL が特別な構成済みのディレクトリ (通常は、cgi-bin と呼ばれる) のファイルを指す場合、Web サーバーは、この URL を、サーバー側プログラム (この場合は、NSAPI プログラム) としてファイルを実行するコマンドとして解釈します。NSAPIプログラムは、ダイナミック リンク ライブラリとしてビルドされます。NSAPI プログラムの実行要求を受け取ると、Web サーバーは DLL をメモリに読み込み、標準のエントリ ポイントでその DLL を呼び出します。NSAPI プログラムは、必要な関数はどれでも実行することができます。そ

の場合、標準化された呼び出しインターフェイス (API) を使用して、Web サーバーからデータを取得したり、データを返したりします。NSAPI プログラムは、ダイナミック リンク ライブラリなので、Web サーバーと同じプロセスで実行され、メモリにコピーが 1 つあるだけで、複数の同時並行のクライアント要求に対して、件数に関係なくサービスすることができます。

NSAPI アプリケーションは、ダイナミック リンク ライブラリとして実装されるため、再入力可能またはスレッド セーフになるように作成する必要があります。NetExpress では、単純なプログラミング方法とビルド方法を使用して、この作業を簡単に行うことができます。

詳細については、次の URL を参照してください。http://www.microfocus.com/NetExpress/nxbooks/links.htm#nsapi1

3.5.3.1 長所と短所

長所	短所
リソース効率が高い。	Netscape 準拠の Web サーバーでしかサポートされない。
応答時間が短い。	保護違反やメモリ リークを避ける必要がある。
大量のデータ読み込みが可能である。	CGI アプリケーションより汎用性が低い。
CGI や ISAPI アプリケーションより柔軟性が高い。	
Web サーバー動作の変更に使用できる。	
ISAPI より多くのプラットフォームでサポートされている。	

3.5.3.2 プログラミングに際しての留意点

別のアプリケーション (この場合は Netscape Web サーバー) が制御するダイナミック リンク ライブラリをデバッグするのは困難です。そのため、サーバー側のプログラムを CGI プログラムとして作成し、デバッグしてから、ビルド設定を変更して、最終的に NSAPI ダイナミック リンク ライブラリを作成することをお勧めします。必要なビルド変更の詳細については、『*インターネット アプリケーション*』オンライン マニュアルの「CGI、ISAPI、および、NSAPI プログラム」の章を参照してください

サーバーで DLL の関数を呼び出す場合は、Netsite サーバーにインポート ライブラリを作成する必要があります。サーバーの関数にアクセスする場合は、DLL にサーバーのインポート ライブラリをリンクする必要があります。

第4章 データ アクセス

通常、COBOL で作成したアプリケーションの動作は、データ量の影響を受けやすく、膨大なファイルの結合処理や RDBMS (Relational Database Management System : リレーショナル データベース管理システム) へのアクセスに大きく左右されます。この章では、NetExpress が提供するデータ アクセス拡張機能の使用方法について説明します。

4.1 課題: RDBMS へのアクセス

クライアント サーバー アプリケーション アーキテクチャで COBOL プログラムからリレーショナル データベースにアクセスするのは、困難です。最も直接的で柔軟性の高い方法は、データベース ベンダにより提供される Open Database Connectivity (ODBC) ドライバを使用する方法でしょう。ただし、ODBC ドライバを直接呼び出す場合、問い合わせを API 呼び出しに変換する方法、データ形式を設定する方法、また使用する呼び出し規則に準拠させる方法を理解している必要があります。幸い、NetExpress の強力なデータ アクセス機能を使用すると、これらの問題を解決することができます。

COBOL プログラムからリレーショナル データへアクセスするには、Open Embedded SQL を使用することをお勧めします。Open Embedded SQL を使用すると、プログラムの EXEC 文と END-EXEC 文の間に、直接、構造化問合せ言語 (SQL) の文を埋め込むことができます。埋め込まれた文は、SQL プリプロセッサであらかじめ処理され、Open Database Connectivity (ODBC) デバイス ドライバの呼び出しに変換されます。

4.1.1 ソリューション: OpenESQL

Open Embedded SQL は、標準的な SQL 文を作成し、COBOL プログラムで EXEC 文と END-EXEC 文の間に作成した SQL 文を埋め込むために NetExpress で使用する機構です。プログラムとデータベースの間の通信は、ODBC を使用して確立されます。ODBC は、業界標準の呼び出しインターフェイスで、Microsoft 社がデータベース用に開発し、普及させています。プログラムは、ODBC ドライバとだけ通信するため、データベースの性質を認識する必要はありません。ODBC ドライバを通して接続されたデータベースは、通常、ODBC データ ソースと呼ばれます。

技術情報: OpenESQL の実装

プログラムで EXEC SQL 文と END-EXEC 文の間に埋め込まれた SQL 文は、OpenESQL プリプロセッサにより処理されます。プリプロセッサは、データ項目 (ホスト変数) を渡すための処理を行い、SQL 文を対応する ODBC インターフェイス呼び出しに変換します。ODBC インターフェイスは、ODBC デバイス ドライバによって実装されます。データベースの ODBC ドライバは、通常、データベースと一緒に RDBMS ベンダから提供されます。ただし、サード パーティのドライバも使用できます。ドライバは、インタプリタとして動作し、標準の ODBC 呼び出しをデータベースで必要なネイティブの API 呼び出しに変換します。ドライバは、ベンダが作成するため、

標準の ODBC を完全に実装していないことがあります。また、標準の ODBC 構文に加えて、独自の拡張機能を使用している場合もあります。ODBC API の準拠レベル と ODBC SQL 文法の準拠レベルは、それぞれ 3 段階ありますが、すべてのドライバで高度な SQL 機能がサポートされているわけではありません。特に、デスクトップ データ ソース向けのドライバでは、高度な SQL 機能がサポートされていないものがあります。また、サポートされているデータ型とその名前も、ドライバごとに異なります。すべての ODBC ドライバは一定水準以上のサブセットを実装することになっていますが、SQL プログラムのコードを作成する前に、使用する ODBC ドライバのマニュアルを確認する必要があります。

4.1.1.1 プログラミングに際しての留意点

OpenESQL を使用すると、RDBMS と簡単で効果的に通信できます。しかし、実際にアプリケーションを効果的に機能させるためには、注意すべき点が数多くあります。

必要な SQL 文の作成

NetExpress の OpenESQL 機能を使用するために、SQL を理解する必要はありません。NetExpress では、OpenESQL アシスタント とインターネット アプリケーション ウィザードのどちらかを使用して、多くのアプリケーションで必要な SQL を作成することができます。データベース参照用、または、更新用のアプリケーションのフレームワークを COBOL コードで自動作成するには、インターネット アプリケーション ウィザードを使用します。ユーザーが作成した COBOL プログラムに SQL 文を追加するには、OpenESQL アシスタント を使用します。OpenESQL アシスタント の詳細については、『*データベース アクセス*』オンライン マニュアルの「OpenESQL アシスタント」の章を参照してください。

SQL 文を手動で作成したり、OpenESQL アシスタント で作成した SQL 文を修正することもできます。ほとんどのベンダは、ODBC ドライバ ソフトウェアに SQL リファレンス マニュアルを添付しています。これらのマニュアルには、完全な構文も含め、サポートされている文に関する詳細が記載されています。また、Web にも、SQL 情報を提供する便利な情報源が数多くあります。Yahoo! や Alta Vista などの好きな Web 検索エンジンで、検索文字列として *SQL* と入力すると、これらの情報源を検出できます。まず初めに、
http://www.microfocus.com/NetExpress/nxbooks/links.htm#sql1 を参照してください。

1 組の EXEC SQL 文と END-EXEC 文の間で記述する SQL 文は、1 文だけにします。たとえば、カーソルを定義して、データベースから複数行を取り出す場合は、次のように複数の EXEC SQL 文を使用します。

```
EXEC SQL
DECLARE mycursor CURSOR
FOR SELECT mycolumn_NAME FROM mytable
```

```
END-EXEC

EXEC SQL
OPEN mycursor
END-EXEC

EXEC SQL
FETCH mycursor INTO :myfirstname
END-EXEC

EXEC SQL
FETCH mycursor INTO :mysecondname
END-EXEC

EXEC SQL
CLOSE mycursor
END-EXEC
```

ODBC ドライバの中には、この方法を適用するために、1 組の EXEC SQL 文と END-EXEC 文の間に複数の SQL 文が記述されている API 呼び出しを受け付けないものもあります。

データベースへの接続

プログラムから ODBC データ ソースのデータへアクセスするには、データベースへ接続する必要があります。
データベースへの接続方法には、2 通りあります。

- *暗黙的接続*

  この方法は、プログラムを常に単独のデータベースに同じユーザー名で接続する場合のような、単純な接続に使用します。プログラムのビルド時には、ODBC データ ソース名、ユーザー名、パスワードを確認しておきます。これらのパラメータは、SQL コンパイラ指令を使用して指定することができます。次に例を示します。

  ```
  SQL(DBMAN=ODBC,INIT,DB=gasops,PASS=public.meters)
  ```

または、.cbl プログラム ファイルの [ビルド設定] で [コンパイル] タブの [詳細] プッシュ ボタンをクリックすると、ダイアログ ボックスにパラメータを入力することができます。

- *明示的接続*

  プログラムを複数のデータベースに接続したり、複数のユーザー名で接続する場合に使用します。明示的に接続するには、次のように ESQL 内に CONNECT 文を記述します。

```
EXEC SQL
CONNECT TO "gasops" USER "public.meters"
END-EXEC
```

データベース処理の終了後には、プログラムをデータベースから切断します。暗黙的接続で接続したプログラムは、プログラムの終了時に OpenESQL によって自動的にデータ ソースから切断されます。一方、明示的接続で接続したプログラムを切断するには、ESQL 内で DISCONNECT 文を記述してください。

ホスト変数

ホスト変数は、COBOL プログラムで定義されたデータ項目であり、プログラムの ESQL 節の変数パラメータとして、ODBC データ ソースとの値の受け渡しに使用されます。ホスト変数は、COBOL プログラムのファイル節 (File Section)、作業場所節 (Working-Storage Section)、ローカル記憶域節 (Local-Storage Section)、連絡節 (Linkage Section) で定義することができ、1 から 48 までのレベル番号を割り当てます。レベル 49 は、SQL データ型 *varchar* のデータ項目用に予約されています。ESQL 文内でホスト変数を使用する場合は、データ項目名の先頭にコロン (:) を付けます。コンパイラは、このコロンによって、データ項目と同じ名前の表や列を識別します。ODBC でパラメータ マーカーを使用できる位置には、ESQL 文のホスト変数を使用できます。パラメータ マーカーは、ODBC ドライバのマニュアルでは疑問符 (?) で表されます。ODBC ドライバによっては、ホスト変数のデータ名が 26 文字を超えると正しく処理されない場合があるので、できる限り 26 文字以下のデータ名を使用してください。

注記: Form Designer を使用して入力フォームをデザインし、インターネット アプリケーション ウィザードを使用してスケルトン CGI プログラムを生成した場合、ホスト変数は、自動的に作成され、フォームの入力フィールドからのデータを格納します。ホスト変数名は、Form Designer でコントロールのツリーで表示される名前と同じです。適切なコントロールの COBOLPicture プロパティを設定すると、ホスト変数のデータ型を制御することができます。

日付と時刻の形式

COBOL では、SQL データベースで通常使用される日時やタイムスタンプなどのデータ型に相当する日付または時刻のデータ型があらかじめ定義されていません。そのため、OpenESQL は、これらの形式を pic x(n) データ項目を使用して、OpenESQL の文字表現に変換します。ODBC 規格では、日付を yyyy-mm-dd、時間を hh:mm:ss として、文字形式で定義しています。これらの形式は、使用するデータ ソースのネイティブな日付形式または時間形式と対

応しているとはかぎりません。

COBOL では、編集されたフィールドで文字 "-" と ":" を使用できないため、日時をグループ データ項目で定義する必要があります。一方、ODBC 規格では、グループ データ項目をデータベースに渡す場合、構成部分に分割するため、グループ項目を再定義し、単一のデータ項目を作成する必要があります。たとえば、次のようになります。

```
01 mydate.
   03   myyear   pic x(4).
   03   filler   pic x value "-".
   03   mymonth  pic x(2).
   03   filler   pic x value "-".
   03   myday    pic x(2).
01 SQLmydate redefines mydate pic x(10).
```

この章で後述する Gas Services 社の例では、日付を ACCEPT し、正しい形式でデータベースへ渡す方法について説明します。

Varchar データ型

SQL の varchar データ型は、可変長の文字列で構成されます。COBOL で varchar データ型に相当するデータ型を定義する方法は、2 通りあります。

- 固定長文字列のデータ項目 (`pic x(`*n*`)`) を定義します。 この場合、*n* は文字列の最大長です。
- comp または comp-5 を使用してグループ項目を定義します。この場合、最初の項目は文字列の長さを、2 番目の項目は文字列自体を示します。どちらの項目にも、49 のレベルがあります。たとえば、次のように表現します。

```
01 varchar1.
   49 varchar1-len     pic 9(4) comp-5.
   49 varchar1-data    pic x(200).


01 Longvarchar1.
   49 Longvarchar1-len  pic 9(4) comp.
   49 Longvarchar1-data pic x(30000).
```

SQL のデータ型 char、varchar、または、long varchar データ型にコピーされたデータが、定義された長さよりも長い場合、データの右端が切り捨てられ、sqlca データ構造体に sqlwarn1 フラグが設定されます。データが定義された長さよりも短い場合、渡された char データ型は空白文字で埋められます。

エラー処理

SQL 文が正しく実行されない場合、プログラムにエラー コードが返されます。SQL 文を実行するたびに SQL エラーを確認する必要があります。ODBC ドライバは、SQL 通信領域または sqlca と呼ばれるデータ ブロックにエラー情報を返します。sqlca には、2 つの変数 (sqlcode と sqlstate) と最後に実行された SQL 文でエラーが発生したかどうかを示す警告フラグが数多く格納されています。SQL 通信領域 (sqlca) と sqlstate 変数の詳細については、ヘルプの [目次] で「*sqlcode*」と「*sqlstate*」を選択し、参照してください。

通常、埋め込み SQL 文が正常に実行されたかどうかを確認するには、この sqlcode の値をチェックします。sqlcode の値は、次のとおりです。

値	意味
0	この文は、正常に実行されました。
1	文は実行されましたが、警告が生成されました。エラーの種類については、sqlwarn フラグの値を確認してください。
100	問い合わせに一致するデータが見つからないか、または結果集合の終わりに到達しました。処理された行はありません。
< 0 (負)	アプリケーション、データベース、またはネットワークのエラーのため、文が実行されませんでした。

COBOL プログラムで SQLCA データブロックを使用するには、データ部に `EXEC SQL INCLUDE SQLCA END-EXEC` 文を記述します。次に例を示します。

```
working-storage section.
...
EXEC SQL INCLUDE SQLCA END-EXEC

procedure division.
EXEC SQL
  SELECT company, city INTO :pcompany, :pcity
                   FROM customer
                   WHERE custid = :pcustid
END-EXEC
if sqlcode not = 0
  if sqlcode = 100
    display "顧客が見つかりませんでした。"
```

```
  else
    display sqlcode
    display sqlerrmc
  end-if
else
  display pcustid " の企業は " pcompany
  display pcustid " の都市は " pcity
end-if
```

埋め込み SQL 文を実行するたびに sqlcode または sqlstate の値を確認するには、非常に多くのコードを記述する必要があります。この問題を回避するには、プログラムに WHENEVER 文を記述し、SQL 文の実行結果を確認します。WHENEVER 文は実行文ではありません。ただし、この文を記述すると、埋め込み SQL 文を実行するたびに、コンパイラによってエラー処理コードが自動生成されるため、コードを記述する手間を省略できます。

FSQLMESSAGETEXT データ項目を宣言すると、さらにエラー処理を簡略化できます。sqlcode が 0 以外の値をとるたびに、このデータ項目には、更新された例外状態を説明する情報が格納されます。MFSQLMESSAGETEXT は、文字列データ項目として、pic x(*n*) 形式で宣言します 。*n* には、有効値を入力します。ODBC エラーメッセージは、70 バイトの SQLCA メッセージ フィールドを超えることがあるので、MFSQLMESSAGETEXT を使用すると便利です。なお、MFSQLMESSAGETEXT のほか、SQLCA、SQLCODE、および SQLSTATE は、どれもホスト変数として宣言する必要がありません。

```
working-storage section.
01 MFSQLMESSAGETEXT    pic x(512).
EXEC SQL INCLUDE SQLCA END-EXEC

procedure division.
EXEC SQL WHENEVER SQLERROR PERFORM ERROR-PROC END-EXEC

EXEC SQL
...
END-EXEC

stop run.

error-proc.
```

```
display "SQL エラー"
display "SQLCODE = " sqlcode
display "SQLERRMC = " sqlerrmc
display "MFSQLMESSAGETEXT = "
display mfsqlmessagetext.
```

UNIX への移植

UNIX サーバーに移植するプログラムを開発する場合、まず、NetExpress on Windows を使用してコード作成とデバッグを行い、完成したプログラムを UNIX でリコンパイルします。リコンパイルには、RDBMS ベンダが提供するターゲット データベース用の COBOL ESQL プリコンパイラを使用します。この場合、ターゲット データベースでサポートされている SQL 文とデータ型だけを使用するようにします。ほとんどのプリコンパイラは、ANSI SQL92 構文をサポートしていますが、特に次の文や関数に注意する必要があります。

- CONNECT 文と DISCONNECT 文
- COMMIT 文と ROLLBACK 文
- MAX または AVERAGE などの関数

また、UNIX に移植できる COBOL データ型を使用するように注意します。特に、comp-5 を使用した COBOL データ型には注意してください。2 つのプラットフォームでは、デフォルトのバイト順位が異なるため、これらのデータ項目が必ずしも予想どおり移植できないこともあります。その場合、comp-x (または display) を使用してください。

4.1.1.2 例

シナリオ「Gas Services 社」の第 1 段階の Web アプリケーション (詳細については、「シナリオ例」を参照してください) について、サーバー側の COBOL プログラムに埋め込まれている OpenESQL 問い合わせの簡単な例を紹介します。この場合、アプリケーションは、HTML フォームでユーザーが入力したデータを使用して SQL データベースを更新します。さらに、アプリケーションは、ユーザーへ更新情報を返すようにデータベースに問い合わせます。問い合わせにより、必ずしも情報を取得する必要はありませんが、データベースが正しく更新されたことを確認します。

gasops という ODBC データ ソース名を持つデータベースは、transact という表に行を追加すると更新できます。この表では、各行が 1 件のガスのトランザクションを示し、各列は、次の内容を表します。

列名	データ型	目的
ACC_TRANS_NO	integer	各トランザクションを識別する一意の数字

ACCOUNT_NO	integer	トランザクションに使用する顧客アカウント番号
TRANS_TYPE	integer	トランザクションの種類を示すコード
TRANS_TEXT	char(80)	トランザクションのテキスト記述
TRANS_UNITS	decimal(11,2)	トランザクションに使用する項目の数
TRANS_RATE	decimal(8,2)	トランザクション単位ごとの値 (該当する場合)
TRANS_VALUE	decimal(8,2)	トランザクションの合計値
TRANS_TAX_RATE	char(1)	適用される税率を表すコード
TRANS_POST_DATE	datetime	トランザクションが入力された日付と時刻
TRANS_EFFECT_DATE	datetime	トランザクションが有効になる日付と時刻

この例の目的に合わせて、データベースを次のように仮定しています。

- 表の各行に対する固有のトランザクション番号は COBOL プログラムで計算する必要がある (データベースがこの番号を自動的に作成するわけではない)。
- 日付の列を現在の日付に設定しない。

指定したアカウント番号でメーターの最新読み取り値を表に問い合わせると、ユーザーはエントリを確認できます。

4.1.1.2.1 プログラム構造体

ユーザー インターフェイス ロジックと (CGI プログラムに含まれる) ビジネス ロジックを分離させるために、別のモジュールを作成してデータベースとの対話を処理します。このモジュールが実行するタスクを次に示します。

1. データベースに接続します。
2. 新しいトランザクション番号を取得します。
3. 顧客の新しいメーター読み取り値でデータベースを更新します。
4. 更新が正しく行われたことを確認するために、データベースを読み取ります。
5. データベースから切断します。

この点に注意して、新しいプログラムのスケルトン metersql.cbl を次のように作成します。

```
data division.
working-storage section.
```

```
linkage section.
copy "mycgi.cpy".
01 Outputdate   pic x(10).

procedure division using FormFields,
                     Outputdate.
perform dbconnect
perform dbgettrans
perform dbupdate
perform dbread
perform dbdisconnect
exit program.
stop run.

dbconnect section.

dbgettrans section.

dbupdate section.

dbread section.

dbdisconnect section.
```

このプログラムは、CGI プログラムから呼び出され、CGI プログラムに結果を返します。CGI プログラムが最初に生成されたときに認識している唯一のデータ項目は、入力フォームのデータ項目です。これらのデータ項目は、フォームの作成時に自動生成されるコピー ファイル mycgi.cpy で定義されます。新しいプログラムの連絡節にこのコピー ファイルを記述するだけで、これらのデータ項目へのアクセスすることができます。ただし、現在の日付のデータ項目は、フォームで表示されていないため、HTML 出力ページに返す必要があります。CGI プログラムの作業場所節で現在の日付を定義し、metersql.cbl の呼び出しパラメータとして使用すると、CGI プログラムに現在の日付を渡すことができます。現在の日付は、metersql.cbl の連絡節で宣言します。

SQL 文の追加

OpenESQL アシスタント を使用して、スケルトン プログラムに必要な SQL 文を追加します。

まず、ESQL 文を追加します。どの ESQL 文でも、手順は基本的に同じです。たとえば、INSERT 文を追加する場合、次のような手順にしたがいます。

1. プログラムで行 `dbupdate section` の下をクリックします。
2. NetExpress の [表示] メニューで [OpenESQL アシスタント] をクリックします。
3. 左側のペインの [ODBC データ ソース] をダブルクリックします。
4. 表示された ODBC 用の「データ ソースの選択」ダイアログ ボックスで、正しいデータ ソース名を選択します。
5. データ ソースへのアクセス権を取得するために、「SQL サーバー ログイン」ダイアログ ボックスにユーザー ID とパスワードを入力します。
6. transact という表をダブルクリックします。
7. 「問い合わせ型の選択」ダイアログ ボックスで [挿入] を選択し、データベースに新規行を追加します。(「問い合わせ型の選択」ダイアログ ボックスでは、各文をクリックすると、それぞれの処理内容が表示されます)。
8. 左側のペインの表 transact を右クリックします。次に、ポップアップ メニューで [列をすべて選択] をクリックします。
9.  アイコンをクリックし、プログラムに SQL 文を追加します。

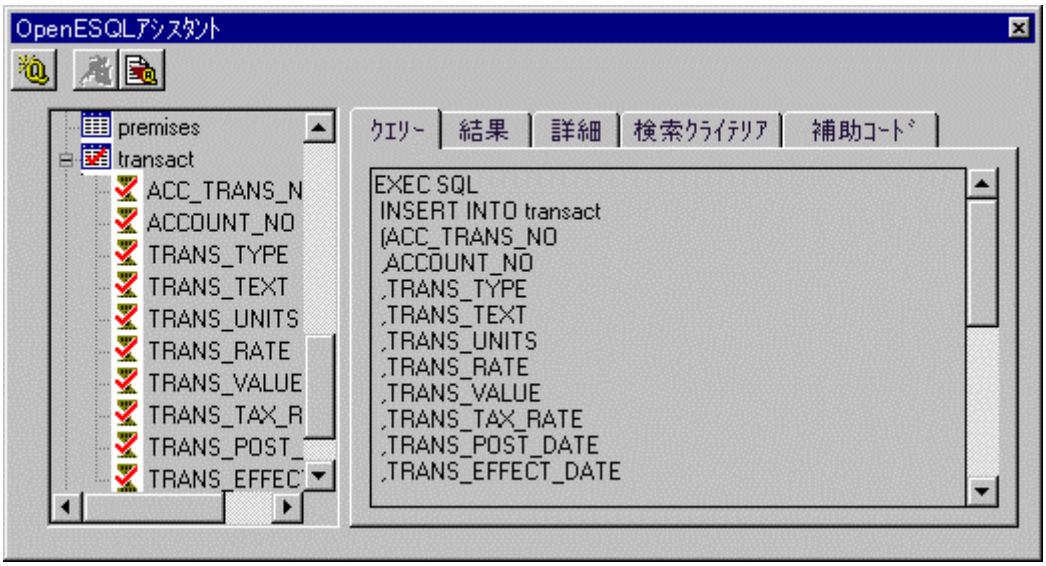

図 4-1 OpenESQL アシスタント を使用した INSERT 文の追加

また、プログラムに SELECT 文を 2 つ追加します。1 つは、最新のトランザクション番号を検索し、もう 1 つは、プログラムがデータベースに追加したデータを読み返します。

最初の SELECT 文は、dbgettrans 節に追加します。

1. SELECT (シングルトン) 問い合わせ型を選択し、1 行分のデータを検索します。
2. 左側のペインで列 A.ACC_TRANS_NO をダブルクリックします。
3.  アイコンをクリックし、プログラムに SQL 文を追加します。

生成された SELECT 文は、データベース テーブルからすべてのトランザクション番号を検索します。ただし、必要なのは、最新のトランザクション番号だけです。最新番号は最も大きな番号なので、次のように SQL 文を修正します。

```
EXEC SQL
 SELECT
    MAX(A.ACC_TRANS_NO)
 INTO
    :transact-ACC-TRANS-NO
  FROM transact A
END-EXEC
```

ほとんどのデータ ソースは、MAX 関数をサポートしていますが、使用する構文は RDBMS によって異なります。上記の構文は、ターゲット システムである Microsoft SQLServer に使用できる正しい構文です。

2 番目に必要な SELECT 文は、プログラムの dbread 節に記述します。

1. SELECT (シングルトン) 問い合わせ型を選択し、1 行分のデータを検索します。
2. 左側のペインの A.TRANS_UNITS をダブルクリックします。
3. 返された行数を絞り込む必要があるので、[検索条件] タブをクリックします。
4. 「列」リストで A.ACCOUNT_NO を選択します。
5. 「ターゲット値」入力フィールドで :Input1 と入力します。
6. [>] ボタンをクリックして、この式を「検索条件」リスト ボックスに追加します。
7. 「列」リストで、[A.TRANS_TYPE] を選択します。
8. 「ターゲット型」を「リテラル」に変更します。

9. 「ターゲット値」では、数字の 9 を入力します (9 は、Gas Services 社のシステムで、顧客が読み取ったメーター値のトランザクション型コードを表します)。

10. [>] ボタンをクリックして、リスト ボックスにこの条件を追加します。

11. [問い合わせ] タブをクリックしてから、  アイコンをクリックし、プログラムに SQL 文を追加します。

上記の手順の結果、次のような SQL 文が追加されます。

```
EXEC SQL
 SELECT
     A.TRANS_UNITS
 INTO
     :transact-TRANS-UNITS
   FROM transact A
  WHERE ( A.ACCOUNT_NO = :Input1 )
   AND  ( A.TRANS_TYPE = 9 )
END-EXEC
```

この問い合わせは、transact テーブルで、ユーザーのアカウント番号と一致し、トランザクション型コードが 9 (顧客が読み取ったメーター値) である行を検索します。実際には、顧客が読み取ったメーター値 (すべて別の機会に読み取った値) がデータベースで数行にわたることもあります。そのため、この時点では、結果集合に複数行が含まれている場合があります。返された結果を必要な 1 行だけに絞り込むには、トランザクション番号を降順 (データの新しい順) に並べます。この段階でも、結果集合は順番に並んだ複数行である可能性がありますが、SELECT 文は、結果集合の 1 行目しか返しません。この行が、最も最近顧客が読み取ったメーター値に対応する行です。このような処理を行うために、生成された上記の SQL 文を次のように編集します。

```
 EXEC SQL
  SELECT
      A.TRANS_UNITS
  INTO
      :transact-TRANS-UNITS
    FROM transact A
   WHERE ( A.ACCOUNT_NO = :Input1 )
    AND  ( A.TRANS_TYPE = 9 )
  ORDER BY A.ACC_TRANS_NO DESC
 END-EXEC
```

ホスト変数とフォーム変数

OpenESQL アシスタント では、正しいホスト変数を簡単にプログラムに追加し、データベース テーブルの各列の COBOL データ型を自動的に正しく調整することができます。

ホスト変数をプログラムに追加するには、次の手順にしたがいます。

1. プログラムで、行 `working-storage section` の下をクリックします。
2. OpenESQL アシスタント で [補助コード] タブをクリックします。
3. 「ホスト変数宣言」オプション ボタンをクリックします。
4.  アイコンをクリックし、プログラムに SQL 文を追加します。

OpenESQL アシスタント は、表の各列のホスト変数宣言を記述したコピーファイル transact.cpy をプログラムに追加します。

データベースの列に対応するホスト変数は、列と同じ名前で (ただし、アンダスコアはハイフンに変換されています)、先頭にテーブル名が付いています。たとえば、列 ACC_TRANS_NO に対応するホスト変数は、transact-ACC-TRANS-NO です。先頭のテーブル名は、オプションであり、NetExpress の [オプション] メニューで [埋め込みSQL] をクリックすると変更できます。

この例で使用する他の変数は、Input1、Input2、および、Outputdate です。これらはすべて、データベース アクセス用ではなく、CGI プログラムの入力フォームと出力ページに関する変数です。これらの変数は、2 つのプログラム間でデータを渡すために使用するので、データ型に互換性があることが必要です。

Outputdate は、metersql.cbl の連絡節と mycgi.cbl の作業場所節で定義されるデータ項目です。現在の日付を CGI プログラムに渡すために使用されます。SQLtoday と同様に、単に `pic x(10)` と定義します。

Input1 は、入力フォームの最初の入力フィールドで入力されたデータを格納します。具体的には、顧客アカウント番号を格納し、データベースの ACCOUNT_NO 列に対応しています。Input2 は、入力フォームの 2 番目の入力フィールドで入力されたデータを格納します。具体的には、顧客が読み取ったメーター値を格納し、列 TRANS_UNITS に対応しています。

transact テーブルの列に関するデータ宣言は、コピーファイル transact.cpy に記述されています。このデータ宣言は 、OpenESQL アシスタント により作成されます。

```
**********************************************************************
     * COBOL DECLARATION FOR TABLE transact *
     **********************************************************************
```

```
01  DCLtransact.
    03 transact-ACC-TRANS-NO          PIC S9(09)  COMP-5.
    03 transact-ACCOUNT-NO            PIC S9(09)  COMP-5.
    03 transact-TRANS-TYPE            PIC S9(09)  COMP-5.
    03 transact-TRANS-TEXT            PIC X(80).
    03 transact-TRANS-UNITS           PIC S9(09)V9(02) COMP-3.
    03 transact-TRANS-RATE            PIC S9(06)V9(02) COMP-3.
    03 transact-TRANS-VALUE           PIC S9(06)V9(02) COMP-3.
    03 transact-TRANS-TAX-RATE         PIC X(1).
    03 transact-TRANS-POST-DATE        PIC X(26).
    03 transact-TRANS-EFFECT-DATE      PIC X(26).
```

上記の例から、Input1 を `pic s9(9) comp-5` 、Input2 を `pic s9(9)v9(2) comp-3` として定義する必要があることがわかります。

変数 Input1 と Input2 は、入力フォームをペイントするときに Form Designer で定義されます。これらの定義は、次のようにコピーファイル gascgi.cpy の COBOL プログラムに追加されます。

```
01 FormFields.
     03 Input1              PIC X(15).
     03 Input2              PIC X(15).
```

このコピーファイルのデータ定義には、Form Designer のフォームで変更した内容を上書きするので、手動で変更しないでください。このデータ定義を変更するには、Form Designer でフォームを開き、SQL データ型と対応するようにコントロールの COBOLPicture プロパティを変更します。たとえば、「Input2」編集フィールドの COBOLPicture プロパティは、次のように変更します。

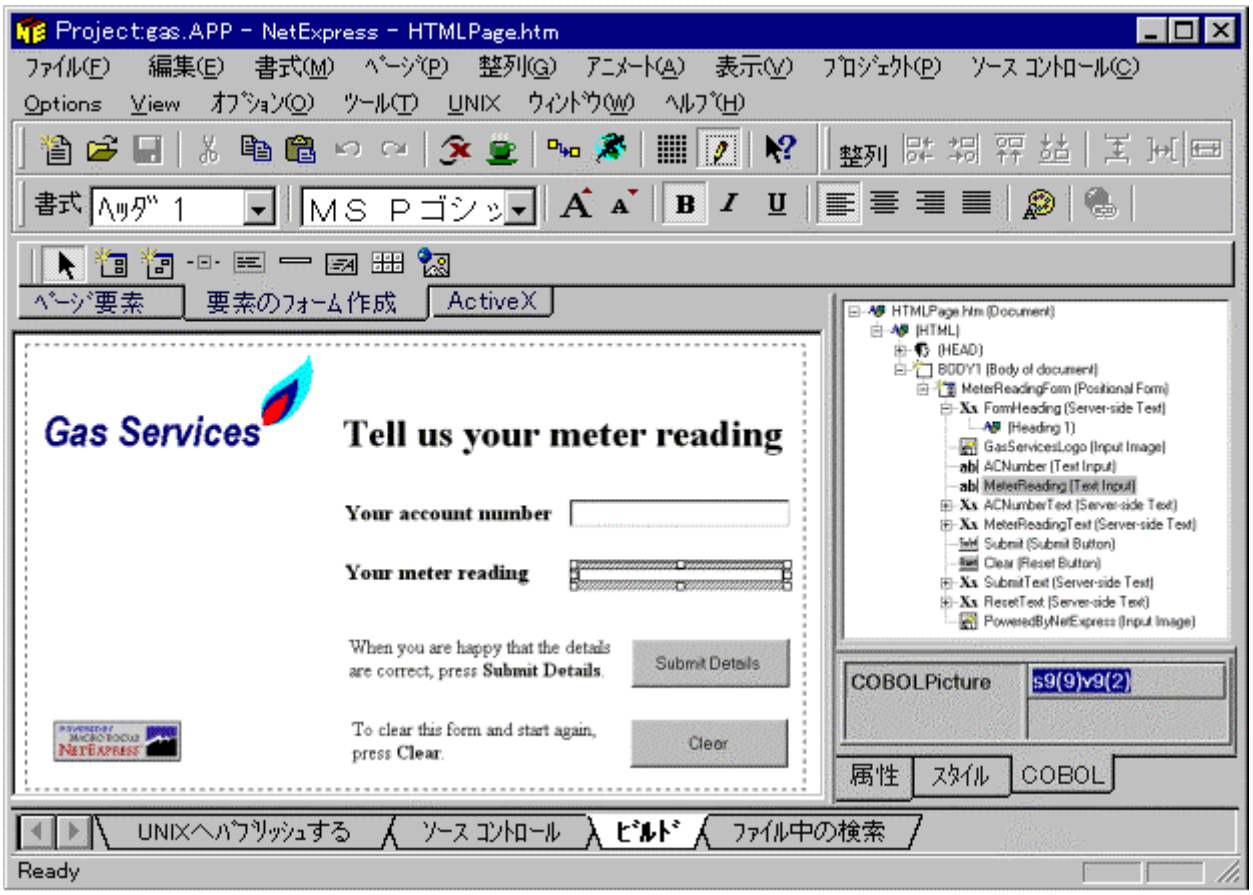

図 4-2 Form Designer による COBOLPicture プロパティの変更

同様に、ACNumber「編集」フィールドの COBOLPicture プロパティを `pic s9(9) comp-5` に変更します。

COBOLPicture を指定する各コントロールには、フォームと通信するための対応するデータ項目があります。このデータ項目は、スケルトン CGI プログラムで Form Designer により自動的に作成されます。このデータ項目は、プログラムで使用するデータ型に関係なく常に pic x 項目で、f- に続けてコントロール名を記述するとこのデータ項目名になります。たとえば、Gas Services 社の入力フォームにある 2 番目の入力フィールドには、次のようなスケルトン CGI プログラムで定義された 2 つのデータ項目が関係します。

- `01 MeterReading pic s9(9)v9(2) comp-3.` (コピーファイル gascgi.cpy で定義されます。)
- `01 f-MeterReading pic x(13)` (コピーファイル gascgi.cpf で定義されます。)

このデータ項目のサイズは、COBOLPicture プロパティから計算されます。この例では、ACNumberは、pic x(10) と、MeterReading は、pic x(13) データ項目と同じサイズです。

HTML は、標準テキストで作成されるため、pic x データ項目しかブラウザで表示できません。入力フォームが送信されるときに、スケルトン CGI プログラムのデータ変換ルーチンにより、データが f-MeterReading から MeterReadingに移動します。また、出力フォームが表示されるときは、データは MeterReading から f-MeterReading に移動します。Form Designer 以外で作成したフォームがある場合 (このマニュアルの説明で、フォームのかわりに

HTML ページを出力するために使用した方法)、HTML のプレースホルダとして正しいデータ名 (f- を前に付けた名前) を使用するように注意してください。

最終プログラム

完成したプロトタイプ用プログラムは、次のとおりです。一部、説明のためのコメントが追加されています。

```
 data division.
 working-storage section.
 EXEC SQL INCLUDE transact END-EXEC

 01 transno        pic x(4) comp-x.
 01 today.
    03   thisyear   pic x(4).
    03   filler     pic x value "-".
    03   thismonth  pic x(2).
    03   filler     pic x value "-".
    03   thisday    pic x(2).
 01 SQLtoday redefines today pic x(10).
 01 acceptdate      pic x(8).
 01 dateparts redefines acceptdate.
    03 yearpart     pic x(4).
    03 monthpart    pic x(2).
    03 daypart      pic x(2).
 01 textnote.
    03 fixedtext    pic x(19) value "Customer reading - ".
    03 mreading     pic x(9).
*>SQLtextnote は、リテラル テキスト (ESQL では単一引用符
*>で囲む必要がある) とホスト変数データ (変数なので引用符
*>で囲めない) を連結するために必要です。
 01 SQLtextnote redefines textnote pic x(28).
 01 tempint        pic 9(9).

 linkage section.
```

```
*>これは、入力フォームを作成したときに生成された
*>コピーファイルです。
 copy "mycgi.cpy".
 01 Outputdate    pic x(10).

 procedure division using FormFields, Outputdate.
 accept acceptdate from date yyyymmdd
 move yearpart to thisyear
 move monthpart to thismonth
 move daypart to thisday
 perform dbconnect
 perform dbgettrans
 perform dbupdate
 perform dbread
 perform dbdisconnect
 move SQLtoday to Outputdate
 exit program.
 stop run.

 dbconnect section.
 EXEC SQL
   CONNECT TO 'boxtest' USER 'rjh.rtfm'
 END-EXEC
 .
 dbgettrans section.
 EXEC SQL
   SELECT
     MAX(A.ACC_TRANS_NO)
   INTO
     :transact-ACC-TRANS-NO
   FROM transact A
```

```
END-EXEC
add 1 to transact-ACC-TRANS-NO
move function integer-part(Input2) to tempint
move tempint to mreading
.
dbupdate section.
move  ACNumber to transact-ACCOUNT-NO
move 9 to transact-TRANS-TYPE
move SQLtextnote to transact-TRANS-TEXT
move MeterReading to transact-TRANS-UNITS
move SQLtoday to transact-TRANS-POST-DATE,
               transact-TRANS-EFFECT-DATE
```

***>このプログラムでは、次の文に含まれる**

***>3 つのホスト変数が常に同じであるため、**

***>これらをリテラルで置換しました。**

```
EXEC SQL
  INSERT INTO transact
  (ACC_TRANS_NO
  ,ACCOUNT_NO
  ,TRANS_TYPE
  ,TRANS_TEXT
  ,TRANS_UNITS
  ,TRANS_RATE
  ,TRANS_VALUE
  ,TRANS_TAX_RATE
  ,TRANS_POST_DATE
  ,TRANS_EFFECT_DATE
  ) VALUES
  (:transact-ACC-TRANS-NO
  ,:transact-ACCOUNT-NO
  ,:transact-TRANS-TYPE
```

```
  ,:transact-TRANS-TEXT
  ,:transact-TRANS-UNITS
  ,0
  ,0
  ,'N'
  ,:transact-TRANS-POST-DATE
  ,:transact-TRANS-EFFECT-DATE
  )
END-EXEC
.
dbread section.
initialize Input2
EXEC SQL
  SELECT
    A.TRANS_UNITS
  INTO
    :transact-TRANS-UNITS
  FROM transact A
  WHERE ( A.ACCOUNT_NO = :Input1 )
  AND  ( A.TRANS_TYPE = 9 )
  ORDER BY A.ACC_TRANS_NO DESC
END-EXEC
move transact-TRANS-UNITS to Input2
.
dbdisconnect section.
EXEC SQL
  DISCONNECT CURRENT
END-EXEC
.
```

アプリケーションのビルド

アプリケーションをビルドする前に、SQL コンパイラ指令を設定し、データ処理モジュールに対して OpenESQL サ

ポートを有効化します。そのためには、metersql.cbl の [ビルド設定] で [コンパイル] タブの [詳細] プッシュ ボタンをクリックします。ダイアログ ボックスでは、次のように選択します。

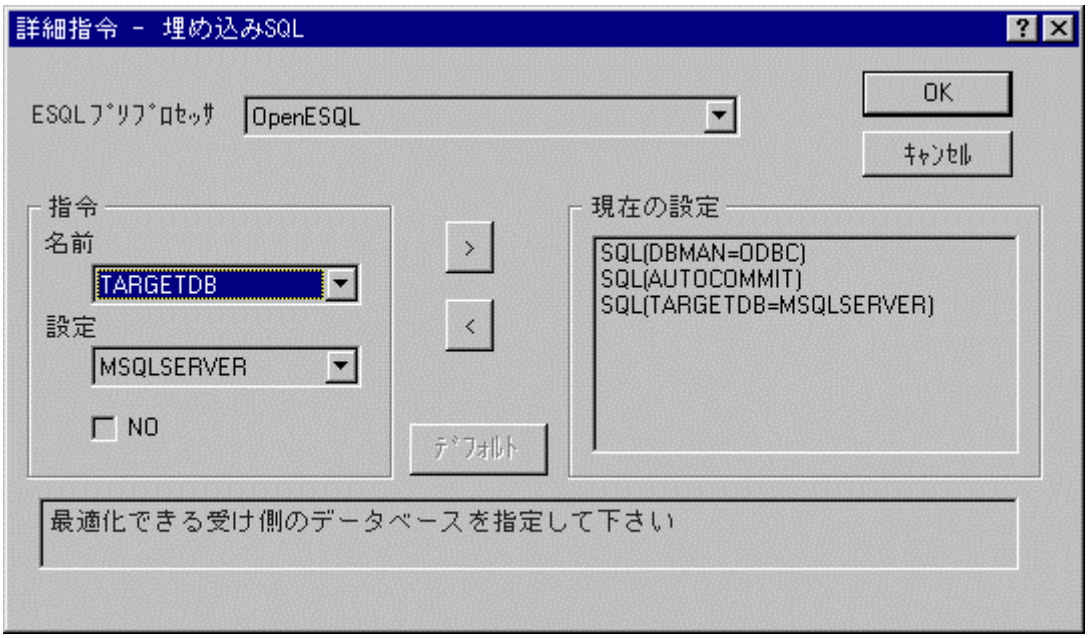

図 4-3 「詳細指令」ダイアログ ボックスを使用した SQL コンパイラ指令の追加

[OK] をクリックすると、SQL コンパイラ指令 (適切なパラメータを設定したもの) が [ビルド設定] の [コンパイル] タブにあるリストに追加されます。これらの設定は、この COBOL ファイルにだけ適用されます。別の方法としては、プログラムに次のコードを 1 行分追加します。

```
$set SQL(DBMAN=ODBC,AUTOCOMMIT,TARGETDB=MSSQLSERVER)
```

4.1.2 ソリューション:インターネット アプリケーションウィザード

インターネット アプリケーションウィザードは、ODBC データ ソースの検索と更新に使用するアプリケーションを簡単に自動生成します。インターネット アプリケーション ウイザードでは、順を追って条件設定用の質問に回答していくと、アプリケーションを実行するために必要な HTML フォームと CGI プログラムを生成することができます。

技術情報:インターネット アプリケーションウィザードの実装
インターネット アプリケーションウィザードは、ODBC データ ソースに読み書きできる CGI プログラムを生成します。CGI プログラムでは、データ ソースと通信するために埋め込み SQL を使用するので、ウィザードの一手順では、必要な SQL 文を生成するために OpenESQL アシスタント を使用します。生成された CGI コードと

HTML フォームは、アプリケーションに合わせて修正することができます。詳細については、オンライン ヘルプの[目次]で [*CGI コード生成*] を選択し、参照してください。

4.1.2.1プログラミングに際しての留意点

各アプリケーションを作成する場合、インターネット アプリケーション ウィザードでプログラムを生成する方法と、独自にプログラムを作成し、OpenESQL アシスタント を使用して SQL を生成する方法のどちらを使用すべきか検討する必要があります。通常、インターネット アプリケーション ウィザードは、次のようなタイプのアプリケーションに適しています。

- 対称的な、フォームベースのアプリケーション
- 完全なレコードを参照し、更新するためのアプリケーション
- データ ソース全体にアクセスできるアプリケーション

OpenESQL アシスタント は、次のようなタイプのアプリケーションに適しています。

- 非対称的な、または、フォームベース以外のアプリケーション
- 問い合わせの結果を操作することが主な目的であるアプリケーション
- フィルタ処理したデータ ソースだけにアクセスできるアプリケーション

インターネット アプリケーション ウィザードの詳細については、『*インターネット アプリケーション*』オンライン マニュアルの「データ アクセス プログラム」の章を参照してください。または、オンライン ヘルプの [目次] で [*インターネット アプリケーション ウィザード*] を選択し、参照してください。

索引