Micro Focus Net ExpressR

入門書

第 5 版
2003 年 4 月

このマニュアルでは、Net Express の使用方法について説明します。第 1 部の各章では、Net Express の概要について簡単に説明します。第 2 部の各章は、Net Express の主な機能を紹介するためのチュートリアルです。各チュートリアルは、10 ～ 30 分ほどの短いセッションです。

第 1 章 : ようこそ

Net Express へようこそ。Net Express は、分散型アプリケーションの開発に最適なソフトウェアです。 Net Express には、企業内の COBOL コンポーネントやビジネスアプリケーションを短時間で構築、近代化して、クライアント / サーバプラットフォームや Web で使用可能にするための優れた環境が用意されています。

すべてのマニュアルは、オンラインで参照できます。 オンラインマニュアルを参照するには、Windows の [**スタート**] メニューまたは Net Express の [**ヘルプ**] メニューを開きます。
Net Express のインストール方法は、インストールユーティリティ「Setup」によって表示されます。

『***入門書***』マニュアルでは、Net Express の主なツールに関するチュートリアルを紹介します。入門書の補足情報は、『***入門書 - その他のトピック***』オンラインマニュアルで説明します。 オンラインヘルプでは、より高度なトピックに関するチュートリアルを数多く紹介しています。

Net Express の概要

Net Express は、COBOL アプリケーションを開発するためのグラフィック環境です。 この統合開発環境では、COBOL コードの編集、コンパイル、およびデバッグが可能です。

また、Net Express では、COBOL Web サービスなどの多数の機能がサポートされます。 COBOL Web サービスは、サードパーティーのソフトウェアを使用しないで Web サービスを作成、使用、および、展開するための機能です。 プログラムを追加しないで COBOL 構成要素を Web サービスに変換できるので、多様なプラットフォームや言語に対応したアプリケーションやシステムで既存のビジネスロジックを使用または再利用できます。 Net Express で作成した COBOL Web サービスは、Micro Focus Enterprise Server でディプロイし、Microsoft .NET クライアント、Java 環境、および Net Express と Server Express でサポートされる Web サービスクライアントを使用した COBOL によって企業内で使用できます。

Net Express を使用すると、Windows アプリケーション (C# や Visual Basic .NET のような Microsoft .NET 言語を使用したアプリケーションなど) と相互運用できる COBOL COM オブジェクトをビルドできます。Net Express では、COBOL と Java の複合ソリューションを構成できる Java クラスまたは EJB (Enterprise Java Bean) クラスとして、既存の COBOL インターフェイスを使用できます。 Micro Focus Enterprise Server と Micro Focus J2EE コネクタをディプロイすると、IBM WebSphere、BEA Weblogic などの Java Application Server から J2EE に準拠した方法で COBOL アプリケーションを使用できます。

Net Express の主な機能

Net Express には、上記の機能以外にも多数の機能が組み込まれています。ここでは、Net Express の主な機能を簡単に説明します。『*チュートリアルの概要*』の章では、これらの主な機能を数多く使用したチュートリアルを紹介しています。 その他の主な機能は、『***入門書 - その他のトピック***』マニュアルを参照してください。

- **開発環境**

  プルダウンメニューがあるメインウィンドウ。メインウィンドウでは、コードの編集、コンパイル、およびデバッグを実行できます。また、Net Express のほとんどのツールにアクセスできます。通常、このメインウィンドウを IDE (Integrated Development Environment; 統合開発環境) と呼びます。

- **プロジェクト**

  アプリケーションに含まれるすべてのファイルとそれらのファイルをコンパイルしてリンクする方法を記述した 1 つのファイルです。プロジェクトは簡単に作成できます。マウスを 1 回クリックするのみでアプリケーション全体をコンパイルしてリンクできます。簡単なアプリケーションも含め、すべてのアプリケーションに対してプロジェクトを作成してください。

- **データツール**

  アプリケーションで使用するデータファイルを変換、参照、編集、および、作成するためのツールセット。データファイルがアプリケーションでどのように更新されるかを確認したり、アプリケーションにテストデータを提供するためにファイルを作成および編集したりできます。ファイルは、どの COBOL 形式でもかまいません。ファイルは、レコードレベルとフィールドレベルで表示できます。

- **Interface Mapping Toolkit**

  COBOL アプリケーションをサービス (Web サービス、EJB、または、COM オブジェクト) に変え、別に作成されてロードされたアプリケーションから独立して起動できるようにするためのツールセット。このツールキットには、次の機能が含まれています。

  - **マッピングウィザード**

    新しいサービスを定義するためのツール。このウィザードで基本的な詳細情報 (マッピングファイルの位置と名前など) を指定します。

  - **インターフェイスマッパー**

    呼び出し側アプリケーションとサービスの間のデータを受け渡しを定義し、そのデータを COBOL アプリケーションのパラメータにマップするためのツール

  - **ディプロイツール**

    Enterprise Server でサービスをディプロイするためのツール

- **XML 対応 COBOL**

COBOL プログラムにコードとして入力できる XML 構文の拡張機能のセット、および、これらの拡張機能を自動生成する CBL2XML ウィザードやコマンド行ユーティリティ。構文の拡張機能を使用すると、COBOL プログラムで XML インスタンスドキュメントに入出力できます。索引ファイルなどの従来のファイル方法を使用する必要はありません。

- Enterprise Server

  COBOL で作成されたサービスをディプロイする環境。Enterprise Server は、クライアントアプリケーションからの要求を受け取ってサービスを起動します。また、スケーラビリティ機能などによってプログラミング作業を減らすことができます。Net Express を開発用マシンにインストールすると、Enterprise Server のコピーもテスト用にインストールされます (Enterprise Server をディプロイしてマシンで運用するためのライセンスについては、『*ライセンスとディプロイ*』の章を参照してください)。

- Dialog System

  Windows ベースのアプリケーションで使用するウィンドウとダイアログボックスを設計するためのグラフィックスクリーンペインタ。イベントを処理する手続きも作成できます。

- UNIX オプション

  UNIX サーバに Net Express アプリケーションをディプロイするためのツール。Net Express を使用して UNIX アプリケーションを完全に開発するためのツールも含まれます。

- OpenESQL アシスタント

  OpenESQL アシスタントは、COBOL プログラムに埋め込む SQL 文を作成するためのウィザードです。

- インターネットアプリケーションウィザード

  CGI ベースの Web アプリケーションを作成するためのツール。HTML ページの作成、および、これらのページを処理するための COBOL のコード化を行います。アプリケーションは、次の 3 通りの方法で作成できます。

  - HTML ページウィザードを使用して作成した HTML フォームから作成
  - 既存の COBOL プログラムの連絡節から作成
  - 既存のデータベース (ODBC ドライバを使用できるデータベース) 構造から作成

- HTML ページウィザード

  CGI ベースの Web アプリケーションに使用するユーザインターフェイスを作成するためのツール。Web フォームを指定すると、必要なファイルが生成されます。

- **Form Designer**

  CGI ベースの Web アプリケーションに使用するフォームを設計するための WYSIWYG フォームエディタ。

- **クラスウィザードとメソッドウィザード**

  COBOL クラスと Java ラッパークラスを作成して、COBOL アプリケーションを COM オブジェクトまたは JavaBean としてディプロイするためのツールセット。

- **テストカバレッジ**

  プログラムのどの範囲がテストされたかを記録する機能。テキスト形式または HTML 形式のレポートが作成されるので、プログラムのどの文がテストで実行され、どの部分がテストされていないのかを開発チームで把握できます。レポートには、プログラムコードの各ブロックがテストされたことを証明する記録も表示できます。

その他のソフトウェアのライセンス

Net Express CD-ROM または付属の CD-ROM には、Micro Focus 製以外のソフトウェアがライセンス付きで収録されています。

- **Microsoft Internet Explorer**

  Microsoft Internet Explorer は、一般的な Web ブラウザです。Form Designer (『*入門書 - その他のトピック*』を参照) でフォームの設計時に Internet Explorer の機能を使用するのでこのソフトウェアが必要です。また、Net Express のマニュアルを表示して Web アプリケーションをテストする場合にも必要です。Internet Explorer がインストールされていない場合、または、Form Designer に必要な Internet Explorer のバージョンがインストールされていない場合は、設定時にインストールできます。

  Samba

  Samba を使用すると、PC から UNIX サーバのドライブを表示できます。Samba は、UNIX マシンでインストールします。

第 2 章 : ライセンスとディプロイ

ここでは、Net Express のライセンスについて、および運用環境にアプリケーションをディプロイする場合のライセンスについて説明します。

ライセンスの概要

Micro Focus Net Express のライセンスで開発して実行できるアプリケーションは、使用許諾対象者が使用するアプリケーションのみです。 開発したアプリケーションを運用環境でエンドユーザに配布できません。

Net Express で開発したアプリケーションを実行するには、実行時環境が必要です。 Micro Focus では、次の 3 つの実行時環境を提供しています。

- Enterprise Server for Windows

  Enterprise Server は、COBOL サービスや COBOL/J2EE のディプロイ、Windows、LINUX、および UNIX での COBOL Web サービスのホスティングなどを実現するスケーラブルな管理処理環境を提供します。

- Application Server for Net Express

  Application Server は、Net Express で作成した COBOL アプリケーションを Windows、LINUX、および UNIX でディプロイするための「高性能である」、「障害が少ない」、「信頼性が高い」、「移植できる」プラットフォームを提供します。 Application Server は、Net Express を使用してディプロイされたアプリケーション用のランタイムサポートファイルで構成されています。

- Run Time System for Net Express

  Runtime System for Net Express は、Net Express を使用して作成されたクライアントアプリケーション (Dialog System も含まれます) を実行するためのソフトウェアとライセンスを含んでいます。 ここで、クライアントアプリケーションとは、実行するコンピュータで稼動している Windows に、これと同じコンピュータからログインしているユーザのオペレーションによって起動されるアプリケーションを言います。ミドルウェアやシステムソフトウェアによって自動起動されるアプリケーションや、ネットワークを介したユーザのオペレーションによって直接･間接に起動されるアプリケーションは、これに該当しません。

Enterprise Server for Windows には、Application Server for Net Express が含まれています。

Application Server for Net Express と、Run Time System for Net Express はソフトウェアとしては同一です。ライセンスの形態のみが異なります。以下、本書で Application Server について述べる場合には Run Time System も含めて参照しているものとします。

アプリケーションを UNIX でディプロイする場合は、Enterprise Server for UNIX のライセンス、Application Server for Server Express のライセンスが必要です。製品の使用許諾契約の内容を確認してください。

Net Express のライセンスの設定

Net Express を実行するには、有効なライセンスをコンピュータに設定する必要があります。デフォルトでは、30 日のライセンスが設定されています。完全なライセンスを取得するには、要求キーを作成して、弊社に送付してください。弊社から応答キーを返信します。応答キーを使用すると、製品を使用するためのライセンスが設定されます。

要求キーと応答キーの作成方法は、次のとおりです。

- 弊社 Web サイトから申請する方法。ライセンスがない場合 (または、ライセンスの期限が近い場合) は、Net Express の起動時に警告が表示されます。弊社の Web サイト (http://www.microfocus.co.jp/) を開き、**[サポート]** > **[ライセンスキーの申請]** へ進んでください。このサイトで要求キーの申請が行えます。
- 電子メールまたはファクスで処理する方法。インターネットから Web サイトに接続できない場合は、電子メールまたはファクスで弊社に要求キーを送信してください。ライセンスに関する警告が表示された場合は、[**電子メール用フォーム**] をクリックします。オンラインでこのフォームに入力して電子メールで送信するか、またはフォームを印刷して記入し、ファクスで送信してください。

  詳細については、『*ライセンスの期限が切れた場合にフルライセンスを取得する方法*』を参照してください。

詳細については、『***ライセンス管理システムユーザガイド***』を参照してください。

Enterprise Server、Application Server、Run Time System のライセンスが必要な場合

Net Express で作成したアプリケーションを運用環境で実行するには、あらかじめ Run Time System、Application Server または Enterprise Server のライセンスを購入しておく必要があります。アプリケーションを社内的に使用する場合も販売する場合も同様です。

Enterprise Server のライセンスが必要な場合は、次のような場合です。

- アプリケーションを Enterprise Server でディプロイする場合
- COBOL Web サービスを Enterprise Server でホスティングする場合
- Enterprise Server と COBOL プログラムの間を J2EE で接続し、その接続に Micro Focus 製の J2EE コネクタ (リソースアダプタ) を使用する場合

他の方法でディプロイする場合は、Application Server または Run Time System のライセンスが必要です。ライセンスがまったく必要ない場合は、次の場合のみです。

- ユーザが、取得済みの Net Express の開発用ライセンスを使用して Net Express でアプリケーションを開発し、そのマシンでそのアプリケーションを実行する場合。別のマシンの Net Express から Application Server をテストするには、開発者用ライセンスが必要です。このライセンスは、Net Express から提供され、使用できる開発者の数が制限されます。詳細については、『***実行ライセンスガイド***』にある『*開発者用ライセンス*』を参照してください。
- Net Express を使用するマシンで、ユーザが他のアプリケーションについて Application Server または Enterprise Server の使用ライセンスを取得している場合。購入したライセンスによって異なる場合があります。

ライセンス条件については、readme ファイルの『*エンドユーザ向けソフトウェア使用許諾書*』を参照してください。

Enterprise Server と Application Server のライセンスの設定

Enterprise Server と Application Server をインストールする場合は、Micro Focus のセットアッププログラムを使用することをお勧めします。このプログラムでは、ライセンスデータベースの場所やライセンスキーなどを指定できます。セットアッププログラムを使用しないで、別のファイルでデータベースの場所を指定し、**aslmpcsilent.exe** スクリプトを使用してライセンスキーをインストールすることもできます。Enterprise Server または Application Server の ***Readme*** で『*インストール上の注意*』を参照してください。これらの ***Readme*** ファイルは、Enterprise Server と Application Server を収録した CD に含まれています。

独自のセットアッププログラムを作成する場合は、そのプログラムにライセンスのインストールを組み込むこともできます。

また、Enterprise Server や Application Server のインストール中にライセンスのインストールを中止することもできます。その場合は、後で AppTrack License Administration ユーティリティを使用してライセンスをインストールする必要があります。

ライセンスの詳細については、『***実行ライセンスガイド***』を参照してください。

他のソフトウェアのライセンス

Micro Focus Net Express の開発システムには、他のベンダの製品も含まれています。これらのコンポーネントをアプリケーションに組み込んで配布するには、これらの製品の使用許諾書に基づき、別のライセンスが必要になることがあります。

Net Express に含まれる Microsoft SDK などのサードパーティのソフトウェアについては、『*エンドユーザ向けソフトウェア使用許諾書*』にある『*サードパーティのソフトウェアに関する契約*』を参照するか、または各ベンダにお問い合わせください。

第 3 章 : チュートリアルの概要

Net Express の「入門書」チュートリアルへようこそ!

概要

ヘルプの [*ここからスタート*] をクリックしてこのマニュアルを開いた場合は、ここまでの章が飛ばされて表示されます。ここは、第 1 部の最後の章です。ここから各部のチュートリアルを直接表示できます。このマニュアルの各部は、Net Express の主な分野ごとにまとめられています。 各部は、チュートリアルを説明した 1 つまたは複数の章で構成されます。

第 1 部の第 1〜2 章を必ず参照してください。これらの章では、Net Express の機能の概要と、ライセンスに関する重要情報を説明しています。

チュートリアルは、COBOL と Windows オペレーティングシステムに精通している一方で、Net Express などの Micro Focus 製品を使用したことがない読者を対象に作成されています。

このマニュアル内で使用する「Windows」という用語は、Net Express を実行できるすべての Windows のバージョンを表します。

大文字と小文字は特に区別する必要はありません。 大文字と小文字を区別する必要がある場合は、明記されています。

プロジェクトのリロード

関連があるチュートリアルでは、同じプロジェクトを使用することがあります。 セッションごとに Net Express を閉じる場合は、プロジェクトをリロードすると便利です。

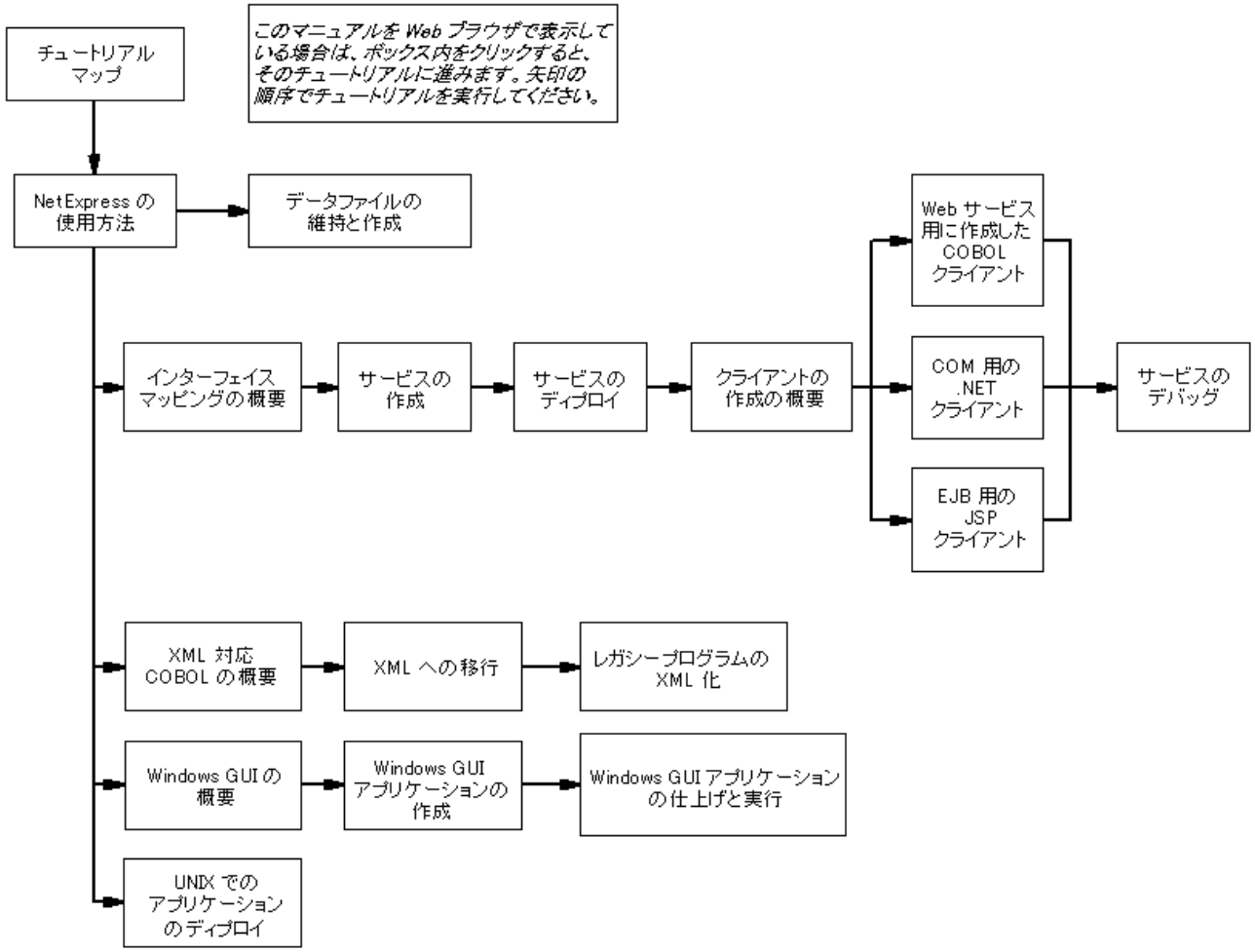
前回 Net Express を閉じたときに開いていたプロジェクトを Net Express の起動時にロードするオプションがあります。このオプションが設定されていることを確認するには、[**オプション > IDE のカスタマイズ**] をクリックし、[**ワークスペース**] タブをクリックして、「**起動時にファイルと前回のプロジェクトを再読み込みする**」がチェックされていることを確認します。[**OK**] をクリックして、ダイアログボックスを閉じます。

また、[**ファイル**] メニューの [**開く**] または [**最近のプロジェクト**] を使用して、既存のプロジェクトを開くこともできます。

チュートリアルマップ

各チュートリアルセッションは、章にまとめられています。このマニュアルのオンラインバージョンでは、次の図のボックス内をクリックすると、各章が表示されます。次の表は、チュートリアルの相関関係を表しています。複数のチュートリアルが矢印で結合されている場合は、矢印の順序に従ってチュートリアルを実行してください。

「入門書」のチュートリアルは、一般的な作業を実行しながら、入門書で説明する各機能を簡単に紹介するために作成されています。 各機能の詳細は、Windows の [スタート] メニューからアクセスできるヘルプを参照してください。



第 4 章 : Net Express の使用方法

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

概要

IDE には、COBOL アプリケーションの編集、コンパイル、およびデバッグ (アニメート) に必要なすべてのツールが統合されています。 また、ツールと COBOL 言語に関するオンラインマニュアルも用意されています。

IDE の使用方法を早く理解するには、簡単な作業をいくつか実行することをお勧めします。このセッションでは、プロジェクトのロード、コンパイル、実行、および、アニメート (デバッグ) について説明します。アニメートとは、ソースコードをステップ実行することです。

プロジェクトとプロジェクトフォルダ

プロジェクトは、アプリケーションに含まれるすべてのファイルとそのコンパイル方法を記述した 1 つのファイルです。 プロジェクトは、簡単に作成でき、コンパイル作業を非常に簡素化します。 簡単なアプリケーションも含め、すべてのアプリケーションに対してプロジェクトを作成してください。 プロジェクトの拡張子は、**.app** です。ディスクでプロジェクトを検索する場合は、ファイル名の末尾にある **.app** 拡張子で識別できます。 ただし、Net Express では、通常、IDE でプロジェクトを作成して維持するので、プロジェクトを直接検索する必要はありません。

アプリケーションのプロジェクトを保存するフォルダをプロジェクトフォルダと呼びます。 アプリケーションで使用する他のファイルの情報がプロジェクト内に記述されるので、これらの他のファイルを別の場所に保存することもできます。ただし、アプリケーションで使用するすべてのファイルをプロジェクトフォルダ内に保存しておく方が便利です。

Net Express をインストールすると、Net Express を含むシステムフォルダ内に、**Net Express\Base\Workarea** フォルダ (デフォルト) がセットアッププログラムによって作成されます。 このフォルダは、作業領域として使用されます。すべてのプロジェクトフォルダをこの作業領域内に保存することをお勧めします。

デモンストレーションアプリケーション

Net Express には、多数のデモアプリケーションが組み込まれています。 一部のデモアプリケーションは、この『***入門書***』マニュアルのチュートリアルでも使用します。 その他のデモアプリケーションは、オンラインヘルプから表示できる高度なチュートリアルで使用します。 デモンストレーションアプリケーションは、**Net Express\Base\Demo** に保存されています。ただし、オプションの拡張機能に関するデモンストレーションアプリケーションは、他のフォルダに保存されています。たとえば、Dialog System のデモは、**Net Express\DialogSystem\Demo** に保存されています。

このマニュアルでは、Net Express が **Net Express** フォルダにインストールされていることを前提にしています。 Net Express を別のフォルダにインストールした場合は、パスを実際のインストールディレクトリに変更してください。

このセッションで使用するデモンストレーションアプリケーションは、Locking です。 このデモンストレーションアプリケーションにはプロジェクトファイルが含まれているので、このセッションでは、プロジェクトファイルをロードしてアプリケーションを実行できます。

IDE の起動

インストール手順の最後に [**実行**] ボタンをクリックして IDE を開いた場合は、インストールユーティリティのメイン画面が表示されたままになることがあります。 ×ボタンをクリックすると、このメイン画面をいつでも閉じることができます。

その他の場合は、次のように IDE を起動します。

1. Windows では、[**スタート > プログラム > Micro Focus Net Express 4.0J > Net Express**] をクリックします。

   このマニュアルでは、メニュー項目に表示されるバージョン番号を省略します。

   数日以内にライセンスの期限が切れることを警告するダイアログボックスが表示されることがあります。これらのセッションでは、この警告を無視します。後で Net Express を起動し、画面の [**ヘルプ**] ボタンをクリックして、完全なライセンスの取得方法を参照してください。

2. ライセンスの警告が表示された場合は、[**OK**] をクリックして無視します。

   IDE をロードすると、IDE ウィンドウと「Micro Focus Net Express へようこそ」画面が表示されます。「Micro Focus Net Express へようこそ」画面には、この画面を毎回開くかどうかを指定するためのチェックボックスが表示されます。 ヘルプファイルを開くか、または、直接 Net Express を開くかを指定できます。

3. 「Micro Focus Net Express へようこそ」画面の [**続ける**] をクリックします。

   「Micro Focus Net Express へようこそ」画面が閉じます。画面の IDE は、図 7-1 のように表示されます。.

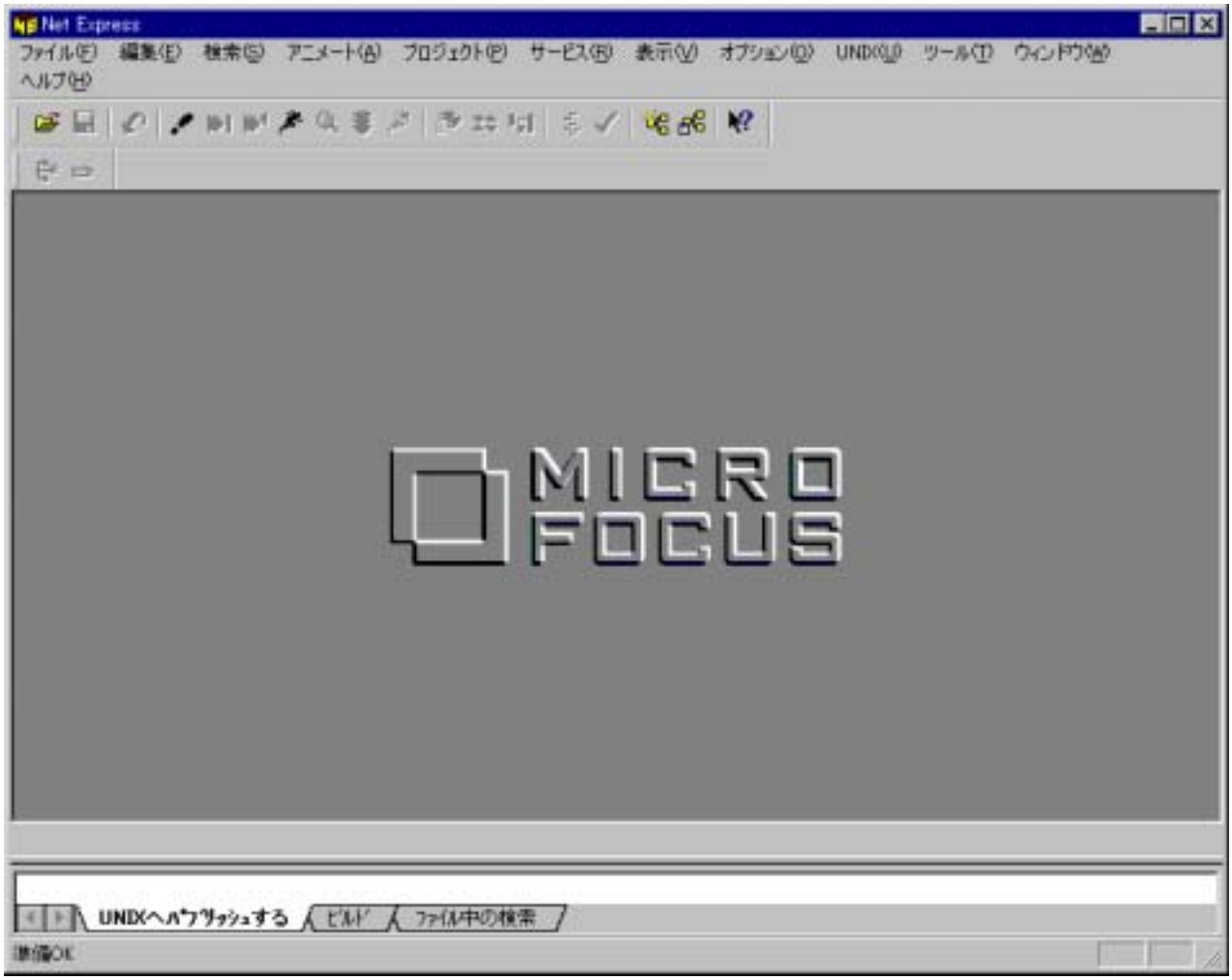


図 4-1: IDE (Integrated Development Environment; 統合開発環境)

大きいペインでは、「プロジェクト」ウィンドウや「編集」ウィンドウなどのさまざまなウィンドウを開きます。その下のペインは、「出力」ウィンドウです。ここには、IDE のメッセージとコンパイラメッセージが表示されます。「出力」ウィンドウには、最もよく使用する [**ビルド**] タブなどのタブが表示されます。[**ビルド**] タブがハイライト (白色表示) されている場合は、このウィンドウにビルドの進行状況を表すメッセージが表示されます。

IDE のサイズと形を変更するには、IDE の端や隅をドラッグします。一部のペインは、切り離して、IDE 内の別の位置や IDE の外側に移動できます。この操作をドッキングの解除と呼びます。一方、ペインを結合させることをドッキングと呼びます (ペイン内を右クリックして「**ドッキングを許可する**」オプションを見つけてください)。

このマニュアルのセッションを実行するコンピュータで他のユーザがすでに Net Express を実行している場合は、ペインが標準の位置以外に動かされていることがあるので注意してください。

プロジェクトのロード

Locking プロジェクトをロードするには、次の手順を実行します。

1. [**ファイル > 開く**] をクリックします。

   「開く」ダイアログボックスが開きます。インストール後に初めて Net Express を起動する場合は、**Net Express\Base\Demo** 内のファイルがこのダイアログボックスに表示されます。このディレクトリ以外のファイルが表示された場合は、ディレクトリを **Net Express\Base\Demo** に変更します。

2. **Locking** サブディレクトリを開き、**locking.app** プロジェクトをクリックして、[**開く**] をクリックします。(標準では、プロジェクトファイルに対して、拡張子 **.app** と  アイコンが表示されます。拡張子を表示しないように Windows を設定している場合は、アイコンによってプロジェクトファイルを識別できます。)

   IDE に「プロジェクト」ウィンドウが開き、プロジェクト内のファイルが表示されます。左側のペインは、ファイルの相関関係を表すツリービューです。右側のペインは、ファイルリストです。 ウィンドウの境界をドラッグすると、ウィンドウのサイズを変更できます。たとえば、右側のペインの幅を広げて、ファイルの情報をより多く表示することができます。

3. 右側のペインを右クリックして、ポップアップメニューを表示します。このメニューの [**ソースファイルだけを表示**] がチェックされていないことを確認します。チェックされている場合は、このメニューをクリックしてチェックマークを外し、コンパイル可能なファイルのみでなく、すべてのファイルがこのウィンドウに表示されるようにします。

   .cbl ファイルは、COBOL ソースファイルです。

4. いずれかの **.cbl** ファイルをダブルクリックします。

   ソースコードを表示するテキストウィンドウが開きます。

5. [**ファイル > 閉じる**] をクリックします。

   テキストウィンドウが閉じます。「プロジェクト」ウィンドウは開いたままです。

.int ファイルは、中間コードの実行可能ファイルです (Micro Focus の実行形式)。

IDE では、業界標準の **.exe** ファイルや **.dll** ファイルも作成できます。ただし、**.int** を作成すると、リンクする必要がないので、新しいアプリケーションをデバッグするときに各プログラムをすぐに再コンパイルできます。

存在しないファイルは、プロジェクト内に表示されません。プロジェクトは、ビルドするための makefile やバッチファイルと似ています。この段階では、プロジェクトをまだビルドしていないので、.int ファイルが存在しません。

プロジェクトのビルド

プロジェクトをビルドすると、ファイルが実行形式にコンパイルされます。作成されるファイルの形式は、プロジェクトタイプに対して選択されたオプションとビルドタイプによって異なります。各プロジェクトには、2 種類の標準のビルドタイプ (デバッグとリリース) のどちらかを設定できます。また、独自のビルドタイプを設定することもできます。

プロジェクトをリビルドするには、次の手順を実行します。

1. [**プロジェクト > リビルド**] をクリックするか、または、ツールバーの  ボタンをクリックします。

   前回プロジェクトをビルドした後に変更されたすべてのファイルがリビルドされます。この場合は、プロジェクトをまだビルドしていないので、すべてのファイルがリビルドされます。同じメニューの [**すべてをリビルド**] をクリックすると、プロジェクト内のすべてのファイル (リビルドが不要なファイルも含む) がリビルドされます。

この機能のみで全体をビルドできます。各ソースファイルに対してコンパイラまたはトランスレータが正しく呼び出されます。Net Express では、通常、コンパイル、翻訳、変換、および、前処理を表すために「コンパイル」という用語を使用します。また、「ソースファイル」という用語は、コンパイル時に入力するすべてのファイルを指します。

ツールバーの  ボタンに注目してください。[**プロジェクト**] メニューの [**リビルド**] にもこの記号が表示されます。メニューに表示される機能のうち、頻繁に使用される機能は、ツールバーのボタンとしても表示されます。このマニュアルのセッションでは、必要に応じてメニューとツールバーのどちらかを使用します。ツールバーのボタンにマウスポインタを合わせると、ボタンに関する簡単な説明が表示されます。このような説明は、ツールチップと呼ばれます。

「出力」ウィンドウには、ビルドの進行状況を表すメッセージが表示されます。ビルドが完了すると、「リビルド完了」というメッセージが表示されます。

アプリケーションの実行

アプリケーションの実行とデバッグの両方に [**アニメート**] メニューを使用します。アプリケーションをデバッグしないで実行するには、次の手順を実行します。

1. [**アニメート > 実行**] をクリックします。

IDE のオプションによって、アプリケーションがすぐに起動する場合と「アニメーションの起動」ダイアログボックスが開く場合があります。このダイアログボックスでアニメートの開始場所を指定します。

このダイアログボックスは、アプリケーションの実行とデバッグの両方に対して表示されます。IDE では、実行とデバッグに同じ機構を使用します。IDE でアプリケーションを実行する場合は、デバッグ機能を無効にしたデバッグエンジンが使用されます。そのため、アプリケーションの実行も [**アニメート**] メニューで制御され、アプリケーションのデバッグと実行は、ほぼ同様に処理されます。

ツールバーの  ボタンは、メニュー項目の [**デバッグ > 実行**] に相当します。

2. 「アニメーションの起動」ダイアログボックスの表示後、[**OK**] をクリックして、Locking プログラムを起動します。

   Locking プログラムは、文字ベースのアプリケーションなので、「アプリケーション出力」ウィンドウという別のウィンドウが開きます。このウィンドウは、図 4-2 のように、文字を表示するアプリケーションに対して開きます。

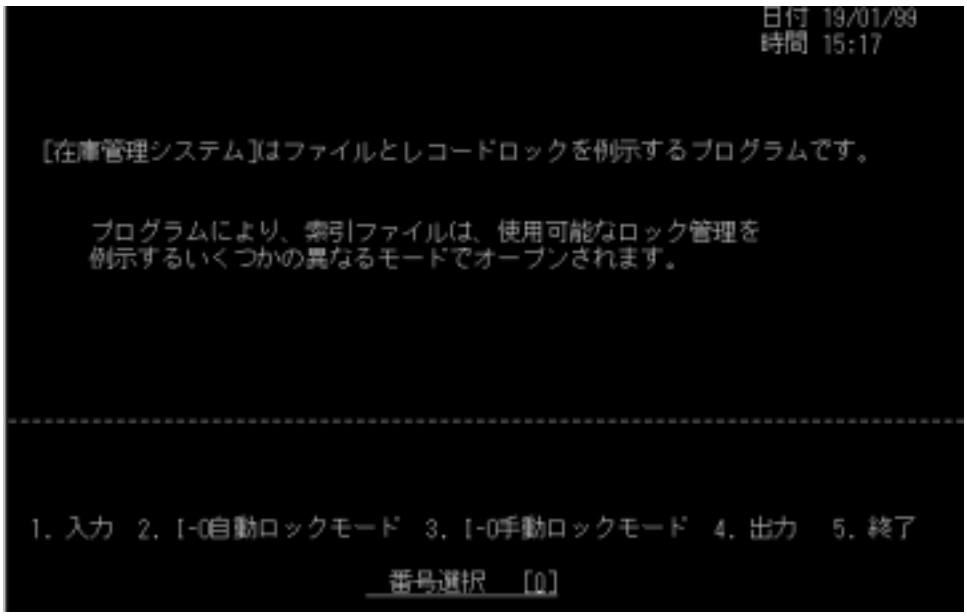


図 4-2：「アプリケーション出力」ウィンドウ

このチュートリアルでは、アプリケーションをビルドして実行する方法のみについて説明します。そのため、アプリケーションの詳細は紹介しません。

3. 「アプリケーション出力」ウィンドウをクリックして選択し、「5」と入力して **Enter** キーを押します。

「5」は、このアプリケーションを終了するためのコードです。アプリケーションの最終的な状態を表示するために「アプリケーション出力」ウィンドウは開いたままになります。

アプリケーションが終了していることを確認するには、[**アニメート**] メニューをクリックします。[**アニメート開始**] と [**実行**] は、アプリケーションの実行中と異なり、淡色表示されません。メニューを再度閉じるには、画面の他の部分をクリックします。

この Locking アプリケーションは、オンラインヘルプの高度なチュートリアルでも使用します。そのチュートリアルでは、別のファイルロック方法を紹介します。

4. 「アプリケーション出力」ウィンドウ内を右クリックし、表示されるポップアップメニューで [**非表示**] をクリックします。

   「アプリケーション出力」ウィンドウが閉じます。

アプリケーションのデバッグ

IDE では、コードの実行状態を視覚的にわかりやすくトレースできます。IDE には、多くのデバッグ (「アニメート」と呼ばれる) 機能が組み込まれています。ここでは、このアプリケーションをアニメートして、これらの機能の一部を紹介します。

1. [**アニメート > アニメート開始**] をクリックします。

   「アニメーションの起動」ダイアログボックスが表示されます。

2. [**OK**] をクリックして、Locking プログラムをデバッグすることを確認します。

   IDE 内にテキストウィンドウが開き、**locking.cbl** のソースコードが表示されます。最初の文は、ハイライトされています。これは、実行の準備が整っていることを表します。

3. [**アニメート > ステップ実行**] をクリックするか、またはツールバーの  ボタンをクリックします。

   ステップ実行すると、次の文が実行されます。この文によって、テキストコンソールに空白文字が表示されます。そのため、このアプリケーションに対して「アプリケーション出力」ウィンドウが開きます。

4. 次の文もステップ実行します (DISPLAY LOCKING01-00)。
5. [**アニメート > 指定範囲を実行**] をクリックします。

   [指定範囲を実行] を選択すると、PERFORM 文または CALL 文のコードが一度にすべて実行されます。

6. EVALUATE CHOICE 文 (RE-ENTER-CHOICE 段落の 2 番目の文) を右クリックして、ポップアップメニューの [**カーソルまで実行**] をクリックします (必要な場合は、テキストウィンドウのサイズを大きくしてソースの表示範囲を広げてください)。

   [カーソルまで実行] を選択すると、ブレークポイントを設定しなくても、簡単に特定の文まで実行できます。 コードは、まだ EVALUATE 文を実行していません。ACCEPT 文まで実行され、入力を受け付けるまでアプリケーションが一時停止します。

7. 「アプリケーション出力」ウィンドウをクリックして選択し、「5」と入力して **Enter** キーを押します。

   EVALUATE CHOICE 文は、ハイライトされています。これは、実行の準備が整っていることを表します。

8. データ項目 CHOICE の値を参照するために、データ項目 CHOICE をダブルクリックします。

   「確認リスト」が開き、値「5」が表示されます。このダイアログボックスでは、各データ項目に対して個別にモニタウィンドウを作成できます。また、データ項目の値を変更したり、「ウォッチリスト」 (画面の下部に「ウォッチリスト」ウィンドウが開きます) にデータ項目を追加したりできます。

9. 「確認リスト」の右隅にある ボタンをクリックします。

IDE オプションの設定

IDE の動作の設定方法について説明します。

1. [**オプション > アニメート**] をクリックします。
2. ダイアログボックスの「**ツールチップでデータ内容を表示**」がチェックされていることを確認し、[**OK**] をクリックします。
3. データ項目 CHOICE にマウスポインタを合わせて、しばらく待ちます。クリックしないでください。

   このオプションを設定して、データ項目にマウスポインタを合わせると、項目の値が表示されます。 この方法の方が「確認リスト」を使用するより便利です。ただし、マウスポインタをデータ項目に合わせるたびに値が表示されるのが不便な場合もあります。このオプションは、必要に応じて設定してください。

4. ソースの行にマウスポインタを合わせ、マウスボタンを押したまま行に沿って少しドラッグします。

   この操作によって行が選択されます。 この方法は、編集時によく使用します。 ただし、行全体を選択する必要がない場合もあります。

5. 行の選択を解除するには、ソースをクリックします。
6. [**オプション > 編集**] をクリックします。
7. ダイアログボックスで [**ブロック / クリップボード**] タブをクリックし、「**プリフィックス領域以外でマウスをドラッグして列範囲を選択**」を選択して、[**OK**] をクリックします。
8. ソースの行にマウスポインタを合わせ、マウスボタンを押したまま、行に沿って少し横方向と下方向に数行分ドラッグします。

   この操作によって長方形のブロックが選択されます。このオプションは、必要に応じて設定してください。

   他にも設定できるオプションが多数あります。これらのオプションについては、IDE の使用方法に慣れた後で学習してください。

アニメートの終了

1.  ボタンをクリックして、アプリケーションの残りの部分を実行します。

   アプリケーションの終わりにある STOP RUN 文に到達すると、コードの実行が終了します。

2. 「STOP RUN 文に達しました」というメッセージの表示後、[**OK**] をクリックします。次に [**アニメート > アニメート停止**] をクリックします。

コンテキストヘルプの使用方法

Net Express でのヘルプの表示方法について説明します。

1. ツールバーの  ボタンをクリックして、「プロジェクト」ウィンドウ内をクリックします。

   「プロジェクト」ウィンドウの説明がポップアップ表示されます。通常、コンテキストヘルプボタン ( または ) を使用すると、このように状況に応じたヘルプをすぐに表示できます。また、多くの画面には、メインヘルプを開くための [**ヘルプ**] メニューまたは [**ヘルプ**] ボタンが表示されます。

2. ポップアップを表示しないようにするには、どれかのキーを押すか、または、マウスで画面をクリックします。

次に進む前に

プロジェクト内をクリックして選択し、[**ファイル > 閉じる**] をクリックするか、または、  ボタンをクリックして、プロジェクトを閉じます。「プロジェクト」ウィンドウを閉じると、そのウィンドウに依存するすべてのウィンドウ (このセッションの場合は、ソースが表示されていたテキストウィンドウ) も閉じます。

「アプリケーション出力」ウィンドウのような標準のウィンドウは、[**ファイル > 閉じる**] を使用して閉じることができません。また、これらのウィンドウがドッキングしている場合は、 ボタンが表示されません。 そのため、「アプリケーション出力」ウィンドウ内を右クリックし、表示されるポップアップメニューで [**非表示**] をクリックします。 また、[**表示 > ドッキングできるウィンドウ**] をクリックし、ダイアログボックスの「**アプリケーション出力**」チェックボックスをクリックしてチェックを外します。次に、[**閉じる**] をクリックして、ダイアログボックスを閉じます。 この操作では、ウィンドウを閉じるのではなく、表示しません。ウィンドウを再度開くと、前回表示されていた内容がそのまま表示されます。

別のセッションに進む場合は、Net Express を開いたままにしてください。 その他の場合は、[**ファイル**] メニューの [**終了**] をクリックするか、または、IDE の ボタンをクリックします。

別のチュートリアルに進むには、『*チュートリアルの概要*』の章にある*チュートリアルマップ*を参照してください。

第 5 章 : データファイルの維持と作成

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

概要

アプリケーションで使用するデータファイルを変換、参照、編集、および作成するには、Net Express のデータツールを使用します。

データツールを使用すると、データファイルがアプリケーションでどのように更新されるかを検証したり、アプリケーションにテストデータを提供するためにファイルを作成および編集したりできます。ファイルは、どの COBOL 形式でもかまいません。ファイルは、レコードレベルとフィールドレベルで表示できます。形式と文字集合を変換できます。

このセッションで使用するデモアプリケーションは、データファイルとデータファイルを維持する COBOL 原始プログラムで構成されます。このデータファイルは、可変長の順ファイルで、職員の詳細情報が 3 種類 (従業員、管理職、経営者) のレコードとして格納されています。ここでは、データファイルエディタを使用して、このデータファイルを可変長の索引順ファイルに変換し、2 つの方法 (フォーマット表示と非フォーマット表示) で表示します。

準備

Net Express を閉じている場合は、再度開きます。「プロジェクト」ウィンドウなどのウィンドウが開いている場合は、閉じます。

プロジェクトを作成する手間を省くために、既存のプロジェクトを使用します。[**ファイル**] メニューの [**開く**] をクリックして、**Net Express\Base\Demo\Dtoldemo\dtoldemo.app** を開きます。「プロジェクト」ウィンドウ に **.cbl** ファイルと **.dat** ファイルが表示されます。

ファイルの変換

組み込みの順ファイルを索引順ファイルとしてコピーするには、次の手順を実行します。

1. [**ツール > データツール > 変換**] をクリックします。
2. 入力ファイルの詳細を表示するには、[**参照**] ボタンをクリックし、「**開く**」ダイアログボックスで **Net Express\Base\Demo\Dtoldemo** フォルダの **datavseq.dat** を選択します。

   入力ファイルと出力ファイルの詳細は、**datavseq.dat** のファイルヘッダーから読み取られた情報で初期化されます。

3. ダイアログボックスの下部で出力ファイルの次の詳細を入力します (他のフィールドは変更しません)。

ファイル名	**Net Express\Base\Demo\Dtoldemo\staff.dat**
形式	Micro Focus
編成	索引順

4.
5. [**キーの定義**] をクリックし、[**キーの挿入**] をクリックします。

   レコードの先頭には、キーとして使用するフィールドが配置されます。このフィールドの長さは、7 バイトです。

6. 「**キーオフセット**」を「**0**」に、「**キー長**」を「**7**」に設定して、[**OK**] をクリックします。
7. [**変換**] をクリックします。
8. データファイルの変換が完了したことを表すメッセージに対して [**OK**] をクリックします。

   「データファイルの変換」ダイアログボックスが再度表示されるので、他のファイルを変換することもできます。

9. [**キャンセル**] をクリックします。
10. 「プロジェクト」ウィンドウ内のこのプロジェクトに新しいファイル **staff.dat** が追加されています。

データファイルの非フォーマット表示

作成したファイルを表示するには、次の手順を実行します。

1. 「プロジェクト」ウィンドウで、**staff.dat** ファイルをダブルクリックします。

   設定したオプションに応じて、「データファイルエディタ」ウィンドウがすぐに表示される場合と、すべての編集内容が索引ファイルと相対ファイルに適用されることを表すメッセージボックスが表示される場合があります。

2. 表示されたこのメッセージを再度表示しない場合には、「**このメッセージは今後表示しない**」というプロンプトをチェックして、[**OK**] をクリックします。

   「データファイルエディタ」ウィンドウが開き、図 5-1 のように **staff.dat** の内容が表示されます。

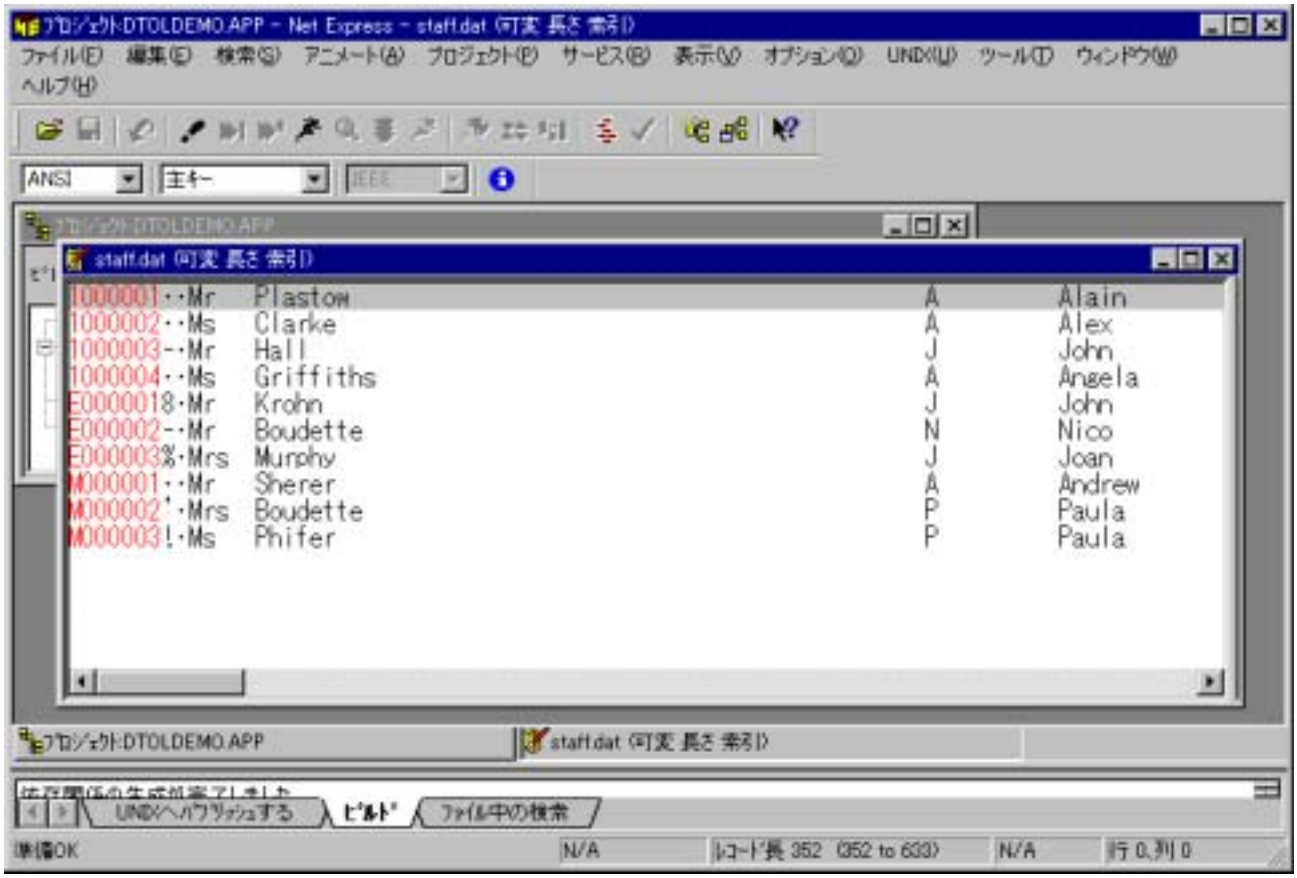


図 5-1: IDE に表示された「データファイルエディタ」ウィンドウ

各レコードは、1 行に表示されます。 データファイルエディタでは、レコードのフィールド長が認識されないため、各行は、フォーマットされていない状態で表示されます (すべてのテキストが 1 行に表示されます)。 COMP 項目のような文字以外のフィールドは (印刷可能な文字に対応していない場合)、通常、□のように、標準文字以外の文字として表示されます。

ファイル内の既存のデータを上書きして編集できます。 編集時には、多数の機能を使用できます。 たとえば、[**検索**] メニューの「**データツール**」オプションには、データファイルの検索方法が 4 通りあります。

- **データファイルの検索と置換**
- **現在のキーで検索** - このオプションは、索引ファイルのみに対して選択できます。「データファイルエディタ」ウィンドウが開いている場合は、現在の参照キーに応じて、このオプション名が「**主キーの検索**」または「**副キー番号 *n* の検索**」に変わることがあります。
- **レコードへの移動** - 現在のレコード内のフィールドを検索するためのオプション。このオプションは、データファイルをフォーマット表示できる場合のみに選択できます。このチュートリアルでも説明するとおり、データファイルをフォーマット表示する場合は、レコードレイアウトファイルをロードします。

IDE の下部にあるステータス行には、ファイルの詳細に応じてさまざまな統計が表示されます。

- 相対レコード番号 (相対ファイルのみ)
- 選択されたレコードの長さ。可変長レコードの場合は、「(最小長、最大長)」がレコード長の後に表示されます。
- レコード番号 (順ファイルのみ)
- カーソル位置 - 行番号と列の位置。どちらも 0 から始まります。

「データファイルエディタ」ウィンドウ内の右側に「このウィンドウを使用するには、先にレコードレイアウトファイルを読み込む必要があります。」というメッセージを表示したペインが開く場合があります。 このウィンドウについては、後で説明します。

ファイルの属性をすべて表示できます。

1. 「データファイルエディタ」ウィンドウ内を右クリックし、ポップアップメニューの [**ファイル情報**] をクリックします。

   「ファイル情報」ウィンドウにファイル形式や編成などの情報が表示されます。

2. ファイルについてより多くの情報を表示するには、[**一般**] タブや [**キー**] タブをクリックします。
3. 完了後、[**キャンセル**] をクリックして、「ファイル情報」ウィンドウを閉じます。
4. このチュートリアルで、まだ「データファイルエディタ」ウィンドウを使用するので、「データファイルエディタ」ウィンドウを開いたままにしておいてください。

レコードレイアウトファイルの作成

データファイルエディタでフィールドのレイアウトが認識される場合は、ファイル内容をフォーマット表示できます。 レコードレイアウトファイルを使用する COBOL プログラムのデータ部からレコードレイアウトファイルを作成します。 必要な情報を作成するには、あらかじめプログラムをコンパイルしておく必要があります。

データファイル **staff.dat** には、3 種類のレコード (従業員、管理職、経営者) が格納されています。 各タイプのレコードのレイアウトと種類の識別方法をデータファイルエディタに認識させます。 どれかの種類をデフォルトとして定義します。 その他のタイプを条件付きレコードレイアウトと呼びます。

レコードレイアウトファイルを生成するには、次の手順を実行します。

1. 「プロジェクト」ウィンドウを選択します。
2. 「プロジェクト」ウィンドウ内の左側のペインで **dfdstaff.cbl** を右クリックし、ポップアップメニューの [**コンパイル**] をクリックします。
3. **dfdstaff.cbl** を再度右クリックし、ポップアップメニューの [**レコードレイアウトの生成**] をクリックします。

「レコードレイアウトエディタ」ウィンドウが開きます。このウィンドウには、**dfdstaff.cbl** のデータ部のツリービューが表示されます。

省略時のレコードレイアウトの作成

省略時のレコードレイアウトを作成するには、次の手順を実行します。

1. **FD MF-FILE** の前にある「+」記号をクリックして、このフォルダを展開します。
2. **01 EMPLOYEE-REC** を右クリックし、ポップアップメニューの [**レイアウトの新規作成**] をクリックします。

   省略時のレイアウトを定義する前なので、「**省略時のレイアウト**」が自動的に設定されます。

3. 設定されていない場合は、[**省略時のレイアウト**] をクリックして **EMPLOYEE-REC** を省略時のレコードレイアウトとして設定します。
4. [**次へ**] をクリックし、さらに [**完了**] をクリックします。

   右側のペインに **EMPLOYEE-REC-DEFAULT** のフォルダが表示されます。

条件付きレコードレイアウトの作成

条件付きレコードレイアウトを作成するには、次の手順を実行します。

1. **01 MANAGER-REC** を右クリックし、ポップアップメニューの [**レイアウトの新規作成**] をクリックします。

   デフォルトのレイアウトをすでに定義したので、「**条件付きレイアウト**」が自動的に設定されます。

2. [**次へ**] をクリックし、さらに [**完了**] をクリックして、**MANAGER-REC** を条件付きレイアウトとして追加します。

   右側のペインに **MANAGER-REC** のフォルダが表示されます。

   ここで、レコードタイプを識別するフィールドと条件を指定します。

3. 右側のペインの **MANAGER-REC** フォルダの「+」記号をクリックして、このフォルダを展開します。次に **01 MANAGER-REC** の「+」記号をクリックして、このエントリを展開します。さらに、**02 MN-CODE** の「+」記号をクリックして、このエントリを展開します。
4. 右側のペインで **03 MN-POSITION** を右クリックし、ポップアップメニューの [**属性**] をクリックします。

   「**フィールドの属性**」ダイアログボックスが開きます。

5. 「**条件**」プルダウンリストから「**IS = TO**」を選択して設定します。
6. 左側の「**条件**」フィールドに **M** (大文字) を入力します。

   M は、「管理職 (Manager)」レコードタイプを表します。 点線の下に表示される 2 つの文字は、16 進数に相当し、縦に並べられます。 ANSI の M は、16 進数の 4D なので、これらの文字は、「4」と「D」になります。

7. [**OK**] をクリックします。

   右側のペインの **03 MN-POSITION** に小さく赤い文字で **IF** と表示されます。

8. 「経営者」レコードタイプに対して条件付きレコードレイアウトを作成するには、**MANAGER** を **EXECUTIVE** に、**MN-POSITION** を **EX-POSITION** に、**M** を **E** に変更して同じ操作を実行します。

レコードレイアウトファイルの保存

レコードレイアウトファイルを保存するには、次の手順を実行します。

1. [**ファイル > 上書き保存**] をクリックします。

   「名前を付けて保存」ダイアログボックスが表示されます。 デフォルトで表示されるフォルダは、**Net Express\Base\Demo\Dtoldemo** です。

   データファイルエディタでデータファイルを開こうとすると、データファイルと同じディレクトリにある同じ基本名のレコードレイアウトファイル (拡張子は **.str**) が検索されます。 ここで使用するデータファイルは、**staff.dat** です (データファイルコンバータを使用して **datavseq.dat** を **staff.dat** に変換したため)。

2. 「**ファイル名**」フィールドを **staff.str** レコードレイアウトファイルに変更します。 [**保存**] ボタンをクリックします。
3. [**ファイル**] メニューの [**閉じる**] ボタンをクリックして、「レコードレイアウトエディタ」ウィンドウを閉じます。

データファイルのフォーマット表示

拡張子以外が同じ名前のレコードレイアウトファイルがフォルダ内に存在する場合にデータファイルを開くと、このレコードレイアウトファイルが自動的に使用されます。 すでに開いているデータファイル、または、同じ名前のレコードレイアウトファイルがないデータファイルに、レコードレイアウトファイルを適用するには、そのレコードレイアウトファイルを指定してロードする必要があります。 この操作を関連付けの作成と呼びます。 データファイルにレコードレイアウトファイルを関連付けると、そのデータファイルと同じ基本名をもつレコードレイアウトファイルが存在する場合でも、そのデータファイルに対しては、関連付けたレコードレイアウトファイルが使用されます。

レコードレイアウトファイルをロードするには、次の手順を実行します。

1. 前に開いておいた「データファイルエディタ」ウィンドウをクリックして、前面に表示します。
2. [**ファイル > データツール > レコードレイアウトのロード**] をクリックします。
3. **staff.str** を選択し、[**開く**] をクリックします。

   右側にペインが新しく開きます。左側のペインには、前と同様に、複数の非フォーマット表示レコードが表示されます。各レコードは、1 行ごとに表示されます。

   右側のペインには、データファイルエディタによってレコードレイアウトファイルから抽出したフォーマット情報が適用されています。このペインには、1 つのフォーマット表示レコードが表示されます。レコードレイアウト名は、ペインの最上部に表示されます。上部の右側には、2 つのナビゲーションコントロールが表示されます。小さい上向きの矢印をクリックすると、前のレコードに移動し、小さい下向きの矢印をクリックすると、次のレコードに移動します。

   「**値**」列をクリックし、既存のデータを上書きして編集できます。非フォーマット表示レコードについて前に説明した編集機能は、フォーマット表示レコードにも適用できます。

4. 左側のペインをクリックし、非フォーマットレコード内でカーソルを動かします。

   フォーマット表示レコードが、ファイル内のレコードのタイプに合わせて変化することに注目してください。

レコードレイアウトファイルの情報検索

データファイルに関連付けられたレコードレイアウトファイルを検索して、さまざまな条件付レイアウトに対して設定した条件を見直すことができます。

1. 「データファイルエディタ」ウィンドウ内を右クリックし、ポップアップメニューの [**ファイル情報**] をクリックします。

   「ファイル情報」ウィンドウが開きます。レコードレイアウトファイルを **staff.dat** に関連付けたので、「ファイル情報」ウィンドウには、[**レコードレイアウト**] タブが表示されます。

2. [**レコードレイアウト**] タブをクリックします。

   staff.dat に関連付けた **staff.str** レコードレイアウトファイルの情報がこのウィンドウに表示されます。このウィンドウの上部には、レコードレイアウトファイルの名前と位置が表示されます。

3. 「**レイアウト名**」の下に表示される **MANAGER-REC** をクリックします。

下の「条件」ボックスに、「管理職」レコードの **MN-POSITION** と **M** は同じであることが表示されます。

4. 「**レイアウト名**」の下に表示される **EXECUTIVE-REC** をクリックします。

   下の「条件」ボックスに、「経営者」レコードの **EX-POSITION** と **E** は同じであることが表示されます。

5. [**キャンセル**] をクリックして、「ファイル情報」ウィンドウを閉じます。

ファイルの 16 進表示

データファイルエディタでは、データを 16 進数として表示して編集できます。

1. 「データファイルエディタ」ウィンドウ (どちらかの子ペイン) 内を右クリックし、ポップアップメニューの [**16 進表示**] をクリックします。

   ウィンドウの下部に別のペインが開きます。 左側のペインには、左上のペインで選択したレコードが表示されます。その下にこのレコードが 16 進数として表示されます。 基本的なフィールド項目のみについては、右上のペインで選択したフィールドが右側のペインに表示されます。その下にこのフィールドが 16 進数として表示されます。 その他の場合は、右側のペインに「**グループ項目**」などのテキストの説明が表示されます。

   16 進数の各組みは、縦に並べて表示されます。 たとえば、ANSI データを編集する場合は、**M** が「4」と「D」として表示されます。「D」は「4」の下に表示されます。 16 進数の行の一部が表示されない場合は、ウィンドウの下の隅を下方にドラッグして、ウィンドウのサイズを大きくしてください。

2. 「データファイルエディタ」ウィンドウ内を右クリックし、ポップアップメニューの [**16 進表示**] をクリックします。

   このメニュー項目のチェックが外れ、16 進数のペインが閉じます。

データファイルの印刷

データファイルエディタの印刷機能を使用すると、データファイルを印刷できます。また、印刷プレビュー機能を使用すると、データファイルを印刷時と同様に画面に表示できます。 データファイルをフォーマット表示するか、または、非フォーマット表示するかを選択できます。 印刷するレコードを、現在のレコード、すべてのレコード、または一部のレコードから選択できます。

データファイルの非フォーマット表示を印刷プレビューするには、次の手順を実行します。

1. 「データファイルエディタ」ウィンドウ内の左側のペインをクリックし、[**ファイル > 印刷設定**] をクリックします。

2. 「**印刷の向き**」を「**横**」に設定して、[**OK**] をクリックします。
3. [**ファイル > 印刷プレビュー**] を選択します。
4. 「**ヘッダ (テキスト)**」の次にあるチェックボックスをチェックし、見出しのテキストを「**STAFF.DAT データファイルの印刷**」と指定します。
5. 次のチェックボックスをチェックします。
    - ページ番号
    - ルーラー
    - レコード番号
    - 16 進の値

    次の項目を選択します。

    - すべてのレコード
    - レコード全体
6. [**プレビュー**] をクリックします。

    「プレビュー」ウィンドウが表示されます。ツールバーのボタンを使用して、拡大と縮小、ウィンドウ内の移動、および、印刷を実行します。

7. 「プレビュー」ウィンドウのボタンバーの [**閉じる**] をクリックします。

複数のデータファイルの編集

Net Express の他のエディタと同様に、データファイルエディタを使用して、複数のファイルを一度に編集できます。IDE 内のウィンドウはさまざまな方法で配置できます。

2 つのデータファイルの一方を他方の上に表示して編集するには、次の手順を実行します。

1. 「プロジェクト」ウィンドウで、**datavseq.dat** ファイルをダブルクリックします。

    datavseq.dat 用に「データファイルエディタ」ウィンドウが新しく開きます。このファイルにはレコードレイアウトが関連付けられていないので、非フォーマットペインのみが表示されます。

2. ボタンをクリックして「プロジェクト」ウィンドウを最小化します。
3. [**Window > Tile Horizontally**] をクリックします。

    datavseq.dat と **staff.dat** の両方に対して「データファイルエディタ」ウィンドウが表示されます。

4. Net Express ウィンドウの下部にある **datavseq.dat** と **staff.dat** のタグをクリックすると、ウィンドウを切り替えることができます。
5. 両方の「データファイルエディタ」ウィンドウを閉じます。 ボタンをクリックして「プロジェクト」ウィンドウを元のサイズに戻します。

新しいデータファイルの作成

固定長の順ファイルを生成するには、次の手順を実行します。

1. [**ファイル > 新規作成**] をクリックし、「**データファイル**」を選択して、[**OK**] をクリックします。

   「**ファイルの作成**」ダイアログボックスが開きます。

2. 「**ファイル名**」フィールドに **Net Express\Base\Demo\Dtoldemo\newfile.dat** を指定して、「**最大の長さ**」フィールドに「**20**」を指定します。その他のフィールドについては、デフォルト設定を使用して、[**作成**] をクリックします。

   固定長の順ファイルにはファイルヘッダーがないので、プロファイルファイルに入力したファイルヘッダー情報を保存するようにプロンプトが表示されます。 プロファイルを作成すると、次にデータファイルを開いたときにファイルヘッダー情報を入力する手間が省けます。

3. [**はい**] をクリックします。

   データファイルと同じディレクトリのプロファイルファイルにプロファイルが保存されます。 プロファイルファイルの基本名は、データファイルと同じですが、拡張子が **.pro** です。 そのため、この新しいデータファイルのプロファイルファイルの名前は、**newfile.pro** になります。

   「データファイルエディタ」ウィンドウが開き、「ファイルが空です。」が表示されます。

   次の手順で、レコードをいくつかの追加および削除します。

4. 「データファイルエディタ」ウィンドウ内を右クリックし、ポップアップメニューの [**後ろにレコードを挿入する**] をクリックします。

   カーソルは、ファイルの先頭にあります。

5. **abc** と入力します。
6. 「データファイルエディタ」ウィンドウ内を右クリックし、ポップアップメニューの [**レコードの繰り返し**] をクリックします。

   上記のレコードのコピーがファイルに追加されます。

7. 「データファイルエディタ」ウィンドウ内を右クリックし、ポップアップメニューの [**レコードの削除**] をクリックして、削除の警告メッセージボックスの [**はい**] をクリックします。

   新しいレコードがファイルから削除され、カーソルが前のレコードに移動します。

8. [**ファイル**] メニューの [**上書き保存**] をクリックします。
9. 「データファイルエディタ」ウィンドウを閉じます。

ファイルの文字集合の変換

組み込みの順ファイルを EBCDIC ファイルとしてコピーするには、次の手順を実行します。

1. [**ツール > データツール > 変換**] をクリックします。
2. 入力ファイルの詳細を表示するには、[**参照**] ボタンをクリックし、「**開く**」ダイアログボックスを使用して、**datavseq.dat** ファイルを選択します。

   入力ファイルと出力ファイルの詳細は、**datavseq.dat** のファイルヘッダーから読み取られた情報で初期化されます。

3. 「**文字集合の変換**」チェックボックスをチェックします。

   ダイアログボックスの下部に表示される出力ファイルの文字集合が ANSI から EBCDIC に変更されます。

4. 「**非テキストデータ項目を含むレコード**」チェックボックスをチェックします。

   テキスト以外のデータ項目 (COMP フィールドなど) を変換しないために、コンバータにこのようなデータ項目の場所を認識させる必要があります。そのため、レコードレイアウトファイルを指定する必要があります。このチュートリアルでは、すでにこのような情報を記述した **datavseq.dat** を作成済みです。

5. [**変換用レイアウトの選択**] をクリックします。「**変換用レイアウトの選択**」ダイアログボックスの [**参照**] ボタンを使用して **staff.str** を選択し、[**OK**] をクリックします。
6. ダイアログボックスの下部で、新しいファイルの名前として **ebcvseq.dat** を入力し、[**変換**] をクリックします。
7. 変換が完了したことを表すメッセージに対して [**OK**] をクリックしてから、[**キャンセル**] をクリックします。

   新しいファイル **ebcvseq.dat** がプロジェクトに追加されます。

EBCDIC ファイルの表示

データファイルエディタが開いている場合は、IDE にツールバーが追加されています。このツールバーには、「データファイルエディタ」ウィンドウを ANSI と EBCDIC の間で切り替えるためのドロップダウンリストが含まれています。作成した EBCDIC ファイルを表示するには、次の手順を実行します。

1. 「プロジェクト」ウィンドウの **ebcvseq.dat** をダブルクリックします。

データファイルエディタでは ANSI 文字を予期しているために、文字が化けて表示されます。

2. ツールバーのドロップダウンリストを使用して文字集合を ANSI から EBCDIC に変更します。

   ファイルが正しく表示されます。

   ツールバーのこのフィールドは、現在表示されている「データファイルエディタ」ウィンドウのみに適用されます。

   このファイルをフォーマット表示する場合に、このチュートリアルで作成したレコードレイアウトファイル使用することはできません。これは、レコードタイプが ANSI 文字として認識され、比較判断されるためです。 データファイルエディタの文字集合を EBCDIC に変更し、新しいレコードレイアウトファイルを作成する必要があります。

3. [**オプション > データツール**] をクリックし、[**一般**] タブをクリックします。
4. ドロップダウンリストを使用して、「**文字セット**」フィールドを EBCDIC に変更します。[**OK**] をクリックします。

   データファイルエディタが読み取られ、すべてのファイルが EBCDIC で書き込まれます。 新しいレコードレイアウトファイルを作成して使用する場合は、『*レコードレイアウトファイルの作成*』～『*データファイルのフォーマット表示*』で説明した方法と同じ方法です。

5. 作業の終了後、「データファイルエディタ」ウィンドウを閉じます。
6. [**オプション > データツール**] をクリックし、文字集合を ANSI に戻して [OK] をクリックします。

   ANSI 文字集合を通常使用する場合は、このオプションを ANSI に設定しておきます。

次に進む前に

プロジェクトを閉じます。

別のセッションに進む場合は、Net Express を開いたままにしてください。

一連のチュートリアルは、ここまでです。 別のチュートリアルに進むには、『*チュートリアルの概要*』の章にある*チュートリアルマップ*を参照してください。

第 6 章 : インターフェイスマッピングの概要

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

COBOL を別のインターフェイスにマッピングする方法のチュートリアルに進む前に、ここでは、Interface Mapping Toolkit の概要を説明します。

Interface Mapping Toolkit

Interface Mapping Toolkit を使用すると、Web サービス、EJB (Enterprise JavaBeans)、COM (Component Object Model) などを介して、既存の COBOL プログラムを COBOL 以外のクライアントで実行できます。 その場合は、COBOL プログラムのエントリポイントとデータ項目を外部インターフェイスにマッピングします。

Web サービスや EJB を介して COBOL 以外のクライアントから COBOL プログラムを実行する場合は、この COBOL プログラムをエンタープライズサーバ (COBOL 用の柔軟なアプリケーションサーバ) で実行します。 エンタープライズサーバを作成するには、Enterprise Server を使用します。 エンタープライズサーバは、マップしたインターフェイス経由でクライアントから送信される要求を処理します。 さらに、エンタープライズサーバは、要求を COBOL のランタイムシステムに渡し、COBOL のランタイムシステムで処理された要求の結果をインターフェイス経由でクライアントに返します。

COM オブジェクトを介して COBOL 以外のクライアントから COBOL プログラムを実行する場合は、この COBOL プログラムを Application Server (Net Express で作成したアプリケーション用にランタイムシステムをサポート) で実行します。 Enterprise Server は使用しません。

このマニュアルの一連のチュートリアルでは、Interface Mapping Toolkit でツールを使用する方法を説明します。 使用するツールは、次のとおりです。

- マッピングウィザード - COBOL プログラムから COBOL の連絡節とエントリポイントを抽出して、すぐに操作できるようにします。
- インターフェイスマッパー - ユーザインターフェイスをドラッグアンドドロップし、必要な COBOL フィールドとエントリポイントを新しいインターフェイスにマップします。
- ディプロイツール - エンタープライズサーバに定義したマッピング情報をディプロイします。 また、必要なインターフェイス (WSDL ファイル、EJB オブジェクトまたは COM オブジェクト) も生成します。 WDSL ファイルでは、Web サービスを定義します。 EJB は、Java アーカイブファイル (.jar) にパッケージされます。 COM オブジェクトは、すべてのマッピング情報をビルドした .dll ファイルです。

Enterprise Server および Application Server のランタイムサポートは、テスト用に Net Express に組み込まれています。 運用マシンで Net Express のアプリケーションをディプロイして実行するには、Micro Focus 社が提供する Enterprise Server または Application

Server のライセンスが必要です。詳細については、『*ライセンスとディプロイ*』の章を参照してください。

Net Express には、COBOL 以外の技術とインターフェイスするための機能や方法が組み込まれています (例 C、Java、COM、Web サーバ側インターフェイスの CGI など)。詳細については、『*入門書 - その他のトピック*』のチュートリアルを参照してください。

次の作業

次の章『*サービスの作成*』に進んでください。

第 7 章 : サービスの作成

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

概要

ここでは、Interface Mapping Toolkit を使用して、既存の (組み込みの) COBOL プログラムからサービスを作成します。つまり、この COBOL アプリケーションをサービスとして活用します。次の章では、このサービスをディプロイします。Web サービス、COM インターフェイス、および、EJB の作成方法について、それぞれの相違点を含めて説明します。以後のチュートリアル「クライアントに関するチュートリアル」では、多様なクライアントアプリケーションからサービスを呼び出す方法について説明します。

サンプルプログラム

ここでは、Net Express に組み込まれたサンプルプログラム book.cbl を使用します。このサンプルプログラムは、索引ファイル (書店で使用する在庫ファイルなど) を維持するための簡単なプログラムです。このセッションを実行するには、このプログラムのロジックの概要を理解する必要があります。

この説明の参照中に、COBOL ソースを表示することもできます。この COBOL ソースは、Net Express\Base\Demo\Mapdemo に格納されており、Net Express またはテキストエディタで表示できます。

Mapdemo フォルダ内には、Clients というサブフォルダが含まれています。ここでは、このフォルダを無視します。このフォルダ内のファイルは、このチュートリアルの後で説明するクライアントのチュートリアルで使用します。

通常、サービスに合わせて調整するアプリケーションは、レガシーアプリケーション (すでに運用中のアプリケーション) です。このようなアプリケーションに最新のインターフェイス (Web インターフェイスなど) を追加して、サービスに使用できるようにします (このマニュアルでも、インターフェイスとして使用するクライアントアプリケーションを作成します)。

Interface Mapper Toolkit は、副プログラムとして設計されたプログラム (連絡節が記述されたプログラム) に対して使用します。このツールキットでは、インターフェイスファイルを作成し、クライアントから受け取ったデータを、指定したパラメータにマップします。レガシーアプリケーションで最上位にある元のプログラムは、使用しません。

book.cbl と同じフォルダ内にある booktest.cbl ファイルは、このような最上位にあるプログラムの簡単な例です。このプログラムには、画面インターフェイスからの入力を要求して、受け取ったデータを処理する副プログラムを呼び出すメインループが指定されています。このセッションでは、まず、レガシーアプリケーションをコンパイルして実行し、レガシーアプリケーションを理解します。

book.cbl には、4 つの機能 (レコードの読み取り、レコードの追加、レコードの削除、および、次のレコードの取得) が含まれています。また、3 つのパラメータ (必要な機能、処理データ、および、ファイル状態を返すデータ項目) も含まれています。ソースには、lnk-function パラメータをテストする EVALUATE 文に必要な機能が指定されています。

「レコードの読み取り」の場合は、呼び出し側プログラムによって、書籍の在庫参照番号、タイトル、または、作者が lnk-b-details 内の対応するフィールドに読み込まれます。このプログラムでは、索引ファイルが検索され、lnk-b-details に完全な詳細情報が返されます。他の機能については、以後で説明します。

インターフェイスマッパー

サービスとクライアントの間でデータを受け渡すフィールドと、そのデータをマップする COBOL プログラムの連絡節のパラメータを定義するには、インターフェイスマッパーを使用します。

COBOL では、PICTURE 文字列を使用してデータ項目のデータ形式を詳細に指定します。また、XML (クライアントと Web サービスの間でのデータ送信に使用)、Java、および、COM では、それぞれ独自のデータ型セットが定義されています。インターフェイスマッパーは、既存の COBOL プログラムのデータ形式を XML、Java、または COM のデータ型にマップするための機能です。

Net Express では、レガシープログラムのインターフェイスを反映するのみのデフォルトのマッピングを生成できます。この例では、元の最上位のプログラムと同様に、クライアントプログラムから lnk-function、lnk-b-details、および、lnk-file-status のデータを送信する必要があります。lnk-file-status は、入力時には必ず空白になっています。機能によっては、lnk-b-details 内の一部のフィールドも入力時に空白になっています。プログラムの EVALUATE 文によって、lnk-function に格納された値に応じて必要な処理が選択され、更新されたデータが lnk-b-details と lnk-file-status に返されます。

ここでは、プログラムの 4 つの操作 (レコードの読み取り、レコードの追加、レコードの削除、および、次のレコードの取得) に個別の名前を付け、各操作 (オペレーション) に必要なデータのみをクライアントから受け取って個別に処理することにより、クライアントに合理的なサービスを提供します。このチュートリアルでは、インターフェイスマッパーを使用して合理的なインターフェイスを定義します。

プログラムの制約

サービスに使用するプログラムの構造には、制約があります。これらの制約は、サービスの使用方法に応じた通常の範囲の制約です。たとえば、プログラム内に ACCEPT/DISPLAY 文を指定しても、このプログラムをリモートで起動できず、入力待ち状態の場合にアプリケーションサーバ全体が一時停止します。これらの制約の詳細については、『*分散コンピューティング*』マニュアルを参照してください。

準備

Net Express を閉じている場合は、再度開きます。「プロジェクト」ウィンドウなどのウィンドウが開いている場合は、閉じます。

プロジェクトを作成する手間を省くために、既存のプロジェクトを使用します。[ファイル > 開く] をクリックして、Net Express\Base\Demo\Mapdemo\mapdemo.app を開きます。「プロジェクト」ウィンドウに book.cbl などのファイルが表示されます。

レガシーアプリケーションのビルドと実行

サービスを作成するためにレガシーアプリケーションをビルドして実行する必要はありません。ただし、ここでは、レガシーアプリケーションを理解するためにこの作業を実行します。

1. [リビルド]  をクリックします。
2. 「プロジェクト」ウィンドウで booktest.int を選択し、ポップアップメニューの [実行] をクリックします。

   画面の一番上の行に、このプログラムでは実行可能な 4 つのオペレーションに数字コードを使用することが表示されます。

3. 機能として 1 (読み取り) と入力し、「在庫番号」フィールドに 1111 を入力して、Enter キーを押します。
4. 機能として 1 と入力し、「在庫番号」フィールドに 2222 を入力して、Enter キーを押します。
5. 機能として 1 と入力し、「在庫番号」フィールドに 3333 を入力して、Enter キーを押します。

   アスタリスクとファイル状態 23 が返される場合は、その在庫番号のレコードがないことを表します。 組み込みの索引ファイルには、レコードが 2 つしかありません。

6. 機能として 2 (追加) と入力し、「在庫番号」フィールドに 4444 を入力します。アスタリスクを削除して、別の書籍の情報を入力します。 たとえば、次のとおりです。

タイトル:	高慢と偏見
タイプ:	フィクション
作者:	オースティン
小売:	08.00
在庫:	4000
販売数:	3000

7. Enter キーを押します。
8. サービスに使用するアプリケーションの概要は、ここまでです。
9. 機能として 9 と入力し、Enter キーを押します。  ボタンを使用して「アプリケーション出力」ウィンドウを閉じます。

前の作業で使用したユーザインターフェイスは、Booktest.cbl から提供されています。このインターフェイスは、後の作業で廃棄されます。ビジネスロジックとファイルアクセスは、Book.cbl によって実行され、サービスが提供されます。新しいインターフェイスを提供するクライアントアプリケーションの作成方法については、後で説明します。

このチュートリアルの構成

Web サービス、EJB、および COM のマッピングの作成手順は類似していますが、それぞれ機構が異なるため、やや相違点があります。

これらの相違点に基づき、この章の残りの範囲では、COM、EJB、および Web サービスに関する項に分割して説明します。また、最初の項で基本的な概念を説明し、残りの項でより高度な機能を説明します。そのため、COM、EJB、および Web サービスのどれを使用する場合でも、これらすべての項を参照してください。

COM

新しいマッピングの作成

新しいマッピングを作成するには、次の手順を実行します。

1. [ファイル > 新規作成] をクリックし、「サービスインターフェイス」を選択して、[OK] をクリックします。

   マッピングウィザードの最初のページが表示されます。このページの後に「サービスインターフェイス：Mapdemo」というタイトルのウィンドウが表示されます。マッピングウィザードと「サービスインターフェイス」ウィンドウは、Interface Mapping Toolkit の一部です。

   マッピングの作成時にプロジェクトを開いておく必要はありません。プロジェクトが 1 つも開いていない場合は、「サービスインターフェイス」グループを作成して新しくマッピングを追加するためのダイアログボックスが表示されます。ただし、通常は、プロジェクトを使用した方が便利です。

2. ウィザードのこのページで、作成するサービスのタイプを選択します。

   [COBOL を COM インターフェイスとしてマップ] をクリックします。[次へ] をクリックします。

3. ここでは、他のプロジェクトを指定したり、マッパーの後のページに情報を入力したりしないで、現在開いているプロジェクトからマッパーによって情報を取得します。

   「現在の Net Express プロジェクトを使用する」が選択されていることを確認して、[次へ] をクリックします。

4. 連絡節を含むプログラムの名前をマッパーに指定する必要があります。このプログラムが、サービスで使用する COBOL アプリケーションの最上位のプログラムになります。

   book.cbl (リストの 2 番目の項目) をクリックして選択し、[次へ] をクリックします。

   IDE で「出力」ウィンドウの [ビルド] タブを選択すると、プロジェクトが自動的にビルドされていることがわかります。これは、ビルドによって作成されたファイル (book.int 実行形式ファイルとデバッグ中に必要な book.idy ファイル) がサービスのディプロイに使用できる状態であることを表します。 デフォルトのビルドタイプが「一般デバッグビルド」なので、これらのファイルは、Mapdemo フォルダ内の Debug サブフォルダに格納されます。

5. このサービスを外部に認識させ、クライアントが起動できるようにするために、サービスの名前を定義します。

   cmapserv と入力します。[次へ] をクリックします。

6. サービスとクライアントの間で受け渡すデータを COBOL プログラムの連絡節のパラメータにマップする方法を定義します。 この操作をマッピングと呼びます。 マッピングウィザードでは、デフォルトのマッピングを作成し、必要に応じて後で編集できます。

   「デフォルトのマッピング」が選択されていることを確認して、[次へ] をクリックします。

7. [終了] をクリックします。

   2 つの新しいメニュー ([オペレーション] と [フィールド]) がメニューバーに追加されます。「サービスインターフェイス」ウィンドウが更新され、インターフェイスマッパーが開きます。次の項では、このウィンドウとインターフェイスマッパーについて説明します。

「サービスインターフェイス」ウィンドウ

「サービスインターフェイス」ウィンドウが 図 7-1 のように表示されます。 この図では、Mapdemo プロジェクトに関連付けられた 3 種類のサービスが表示されています。この段階で作成されているサービスは、1 つです。このサービスが格納されているフォルダが、サービスの種類を表しています。このサービスには、Book という 1 つのオペレーションが含まれています。このチュートリアルでは、さらにオペレーションを追加します。

「サービスインターフェイス」グループは、Mapdemo プロジェクトのサービスインターフェイスセットです。 これらのインターフェイスは、mapdemo.mpr というリポジトリに保存されています。通常は、「サービスインターフェイス」グループは、「サービスインターフェイス」ウィンドウで表示できるので、[ファイル] メニューからこのウィンドウを再度開く場合を除いて、ファイル名を記憶しておく必要はありません。

図 7-1: COM インターフェイスが 1 つ表示された「サービスインターフェイス」ウィンドウ

インターフェイスマッパー

インターフェイスマッパーは、図 7-2 のように表示されます。インターフェイスマッパーは、ウィザードの [終了] をクリックすると、自動的に開きます。インターフェイスマッパーが開いていない場合は、「サービスインターフェイス」ウィンドウでサービス名をダブルクリックするか、またはサービス名を選択して [サービス > インターフェイスの編集] をクリックして、インターフェイスマッパーを開きます。

この図では、インターフェイスマッパーのツリービューがすべて展開されています。 作成したデフォルトのマッピングも表示されています。 ここでは、まず、このマッピングの意味を説明し、このマッピングを使用するかどうかを検討します。

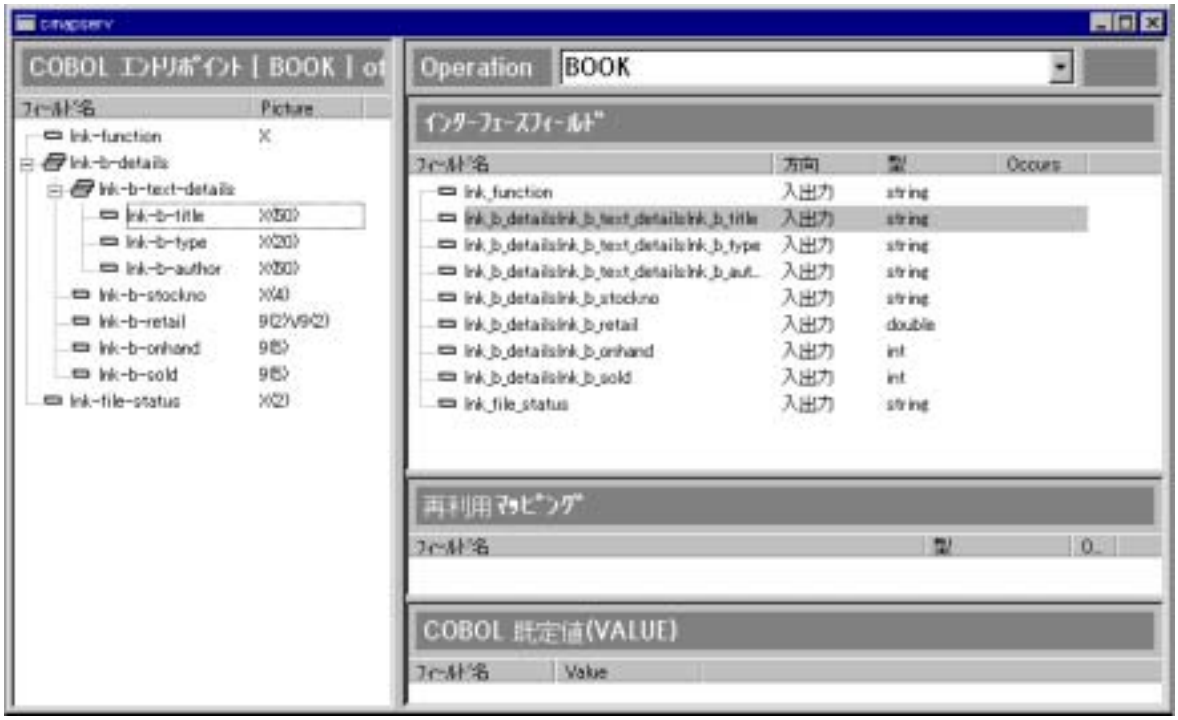


図 7-2: cmapserv のデフォルトのマッピング

デフォルトのマッピングでは、新しいサービスで提供する機能 (オペレーション。ここでは「Book」オペレーション) が定義されています。クライアントがこのオペレーションを起動すると、Book というエントリポイントで COBOL プログラムに制御が移ります。この簡単なプログラムでは、このエントリポイントが手続き部の開始点になります。左側のペインには、プログラムの連絡節が表示されています。

デフォルトのマッピングでは、インターフェイスのフィールド (サービスとクライアントの間でデータを受け渡すフィールド) が定義されています。右上のペインには、これらのフィールドが表示されています。 データは、これらのインターフェイスのフィールドからプログラムのパラメータにマップされます。フィールドは、連絡節の各項目に対して宣言されており、マッピングが自動的に設定されます。 各インターフェイスフィールドには、対応する連絡節の項目の PICTURE 文字列に最も近い COM データ型が設定されています。

これらのフィールドの名前は、対応する連絡節の項目に基づいています。名前の競合を防ぐには、連絡節の項目名とその項目を含む集団項目の名前を続けて指定します。 ハイフンは、COM の変数名に使用できないので、アンダスコアに置換されます。 インターフェイスのフィールド名を指定する方法については、後で説明します。

インターフェイスのフィールドは、COM で使用できるように、すべて個別のフィールドとして作成されます。 インターフェイスのフィールド名に COBOL のデータ構造を反映した階層構造を指定する方法については、後で説明します。

COM のパラメータは、入力、出力、入出力、または、出力リターンに使用できます。 デフォルトのマッピングでは、すべてのインターフェイスフィールドが入出力フィールドです。

不要なオペレーションの削除

この後の項では、『*サンプルプログラム*』の項で説明したサンプルプログラム「Book」のソースについて何度か説明します。

インターフェイスマッパーの説明では、デフォルトのマッピングを使用しましたが、ここでは使用しません。デフォルトのマッピングの場合は、クライアントからインターフェイスのフィールドにデータが渡され、インターフェイスフィールド lnk_function で受け取ったデータに応じてプログラムの EVALUATE 文で処理が選択されて、データ項目 lnk-function にマップされます。lnk-b-details と lnk-file-status の更新データがインターフェイスのフィールドに返されます。

ここでは、プログラムの 4 つの操作 (レコードの読み取り、レコードの追加、レコードの削除、および、次のレコードの取得) に個別の名前を付け、各操作 (オペレーション) に必要なデータのみをクライアントから受け取って個別に処理することにより、クライアントに合理的なサービスを提供します。

そのため、ここでは、デフォルトのマッピングで作成されたオペレーションを削除し、後の項で必要なオペレーションを作成します。

1. IDE のプルダウンメニューで [オペレーション > 削除] をクリックし、[はい] をクリックします。

インターフェイスマッパー内のすべての項目が表示されなくなります。

「レコードの追加」オペレーションの定義

「レコードの追加」オペレーションを定義するには、次の手順を実行します。

1. IDE のプルダウンメニューで [オペレーション > 新規作成] をクリックします。
2. オペレーション名として「Add」を入力し、[OK] をクリックします。

インターフェイスフィールドの定義

ソース book.cbl を確認すると、ファイルにレコードを追加した場合のファイル状態が lnk-file-status に返されます。

1. lnk-file-status を「インターフェイスフィールド」ペインにドラッグします。

   lnk_file_status というインターフェイスフィールドが作成されます。 新しいフィールドと、そのフィールドの作成に使用した連絡節の項目の間のマッピングは自動作成されます。

2. lnk_file_status インターフェイスフィールドにある「方向」フィールドをダブルクリックし、ドロップダウンリストで方向を「出力」に変更します。

   その結果、出力フィールドが作成され、クライアントから受け取る入力メッセージではなく、クライアントに返す応答メッセージに含まれます。

   COM の場合は、フィールドを Input、Output、I/O、または Output Return に指定できるので、フィールドを通常の出力パラメータとしてではなく、戻り値として使用できます。.

インターフェイスフィールドセットの定義

ソースを参照すると、「レコードの追加」オペレーションについては、索引ファイルに新しいレコードを追加するためのすべてのフィールドをクライアントプログラムから提供する必要があることがわかります。これらのフィールドの値は、Book プログラムの lnk-b-details データ項目に渡されます。

1. lnk-b-details を「インターフェイスフィールド」ペインにドラッグします。

   各データ項目のインターフェイスフィールドが作成されます。 ここでは、デフォルトの方向「入力」を使用します。これらのフィールドでは、lnk-b-details デフォルト項目に渡すデータを受け取ります。

COBOL 規定値の定義

必要なオペレーションを通知するパラメータは、lnk-function です。ただし、受け取るメッセージには、対応するパラメータではなく、オペレーション名「Add」が含まれています。ここでは、Add の要求を受け取ったときに、lnk-function に格納される値によって、プログラムで「レコードの追加」オペレーションが実行されるように指定する必要があります。ソースでは、この値が 2 となっています。lnk-function フィールドのようなフィールドを COBOL 規定値と呼びます。

1. 左側のペインから lnk-function を「COBOL 規定値 (VALUE)」ペインにドラッグします。
2. 値として 2 を入力し、[OK] をクリックします。

完了した「レコードの追加」オペレーション

図 7-3 は、完了した Add オペレーションを表しています。

[ファイル > 上書き保存] をクリックするか、または、Ctrl+S キーを押すと、編集内容をいつでも保存できます。

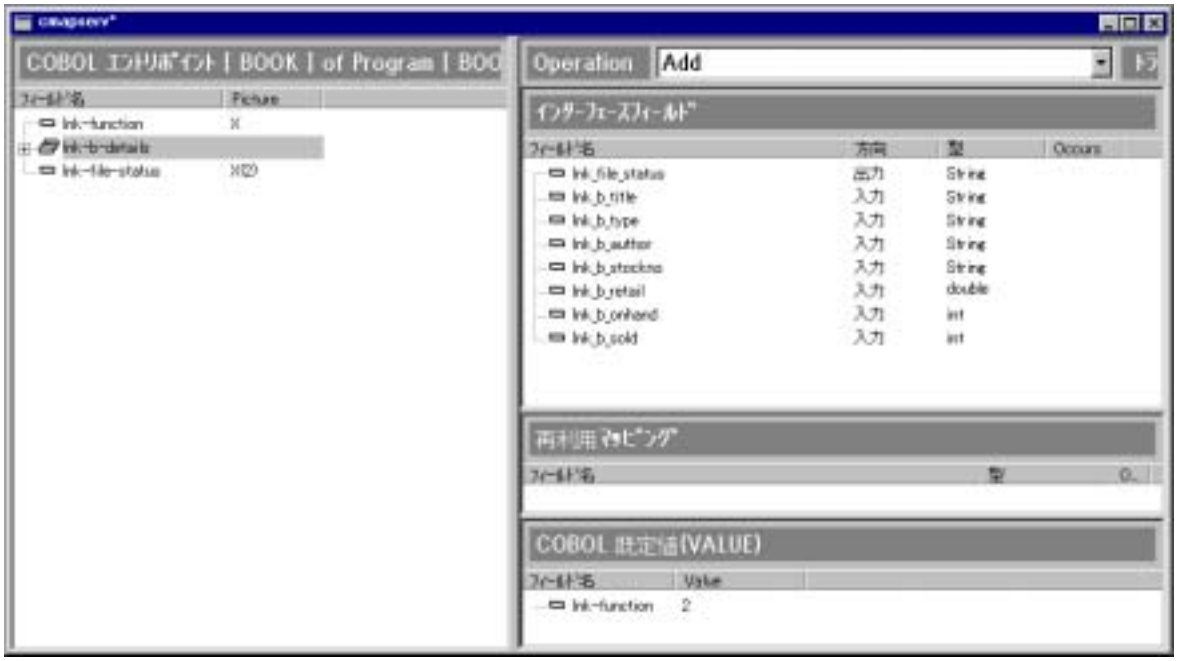


図 7-3: Add オペレーション (COM)

詳細を説明します。

連絡節の項目とインターフェイスフィールドの対応関係は、ドラッグアンドドロップ操作の実行時に設定されます。インターフェイスフィールドを右クリックし、[マッピング] をクリックすると、ダイアログボックスが開き、関連付けられた連絡節の項目が表示されます。この対応関係は、インターフェイスフィールドの名前や順序とは関係ありません。このダイアログボックスで、マッピングを変更できます。

サービスとクライアントの間で渡されるメッセージでは、インターフェイスフィールドはキーワードで識別されます。キーワードには、「インターフェイスフィールド」ペイン内の名前が使用されます。「インターフェイスフィールド」ペインに表示されるインターフェイスフィールドの名前は、クライアントでも同じ順序で表示されるので、この順序を記憶しておくと便利です。フィールドの順序を変更するには、フィールドをドラッグします。

「次のレコードの取得」オペレーションの定義

ここで使用する索引ファイルの主キーは、在庫番号です。プログラムによる「次のレコードの取得」オペレーションでは、在庫番号を入力として受け付け、その在庫番号をもつレコードを検索して、そのレコードを返します。

このプログラムでは、このオペレーションを実行するために、連絡節の lnk-b-details 項目を入出力パラメータとして使用します。ただし、入力時には、lnk-b-stockno は、実際に使用される lnk-b-details の一部にすぎません。

1. タイトルバーの下にある「オペレーション」を右クリックし、表示されるポップアップメニューで [新規作成] をクリックします。 オペレーション名として「Next」を入力し、[OK] をクリックします。

   ポップアップメニューは、前に使用したプルダウンメニューのかわりに使用できます。

   新規作成したオペレーションの「インターフェイスフィールド」ペインと「COBOL 規定値 (VALUE)」ペインは、空白なので、インターフェイスフィールドと規定値を設定します。連絡節は表示されたままなので、連絡節にアクセスしてフィールドと規定値を作成できます。

2. 上記の『*COBOL 規定値の定義*』の項に従って、COBOL 規定値を定義します。 ソースを参照すると、lnk-function の値が 4 であることがわかります。
3. 上記の『*インターフェイスフィールドの定義*』の項に従って、lnk_file_status 出力パラメータを定義します。 必ず、方向を「出力」に変更してください。
4. 「次のレコードの取得」オペレーションの分岐では、ファイルから必要なレコードを取得してクライアントに返します。 そのため、このレコードのフィールドを返す出力フィールドのセットを定義する必要があります。

   左側のペインから lnk-b-details を「インターフェイスフィールド」ペインにドラッグします。

5. 手順 4 で作成した各フィールドの方向を「出力」に変更します。
6. lnk_b_stockno フィールドは、出力フィールド、および、必要なレコードを識別するキーを入力するためのフィールドとして機能します。

   lnk_b_stockno インターフェイスフィールドの方向を「入出力」に変更します。

完了した「次のレコードを取得」オペレーション

図 7-4 は、完了した Next オペレーションを表しています。

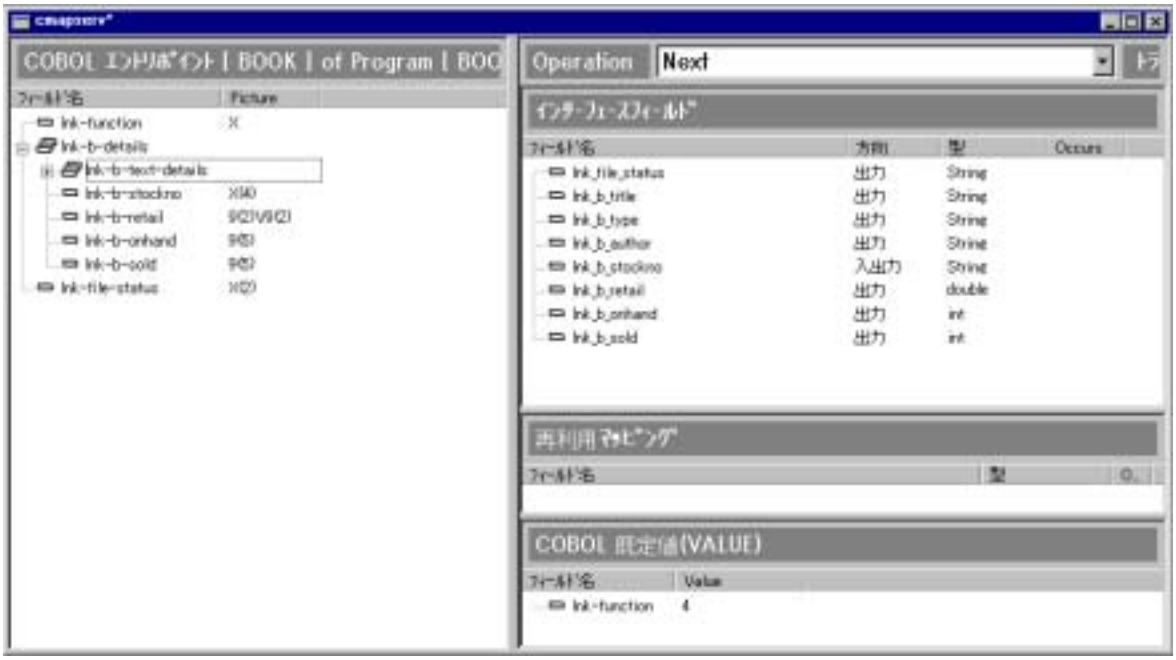


図 7-4: Next オペレーション (COM)

「レコードの読み取り」オペレーションの定義

プログラムによる「レコードの読み取り」オペレーションでは、lnk-b-details を入出力パラメータとして使用します。このフィールドには、在庫番号、タイトル、または、作者を入力します。lnk-b-details の他のデータ項目は無視されます。また、lnk-b-details には、入力された在庫番号、タイトル、または、作者を示すレコードが出力されます。

1. [オペレーション > 新規作成] をクリックし、オペレーション名として Read を入力し、[OK] をクリックします。
2. 前の説明に従って、COBOL 規定値を定義します。ソースを参照すると、lnk-function の値が 1 であることがわかります。
3. 前の説明に従って、lnk_file_status 出力パラメータを定義します。 必ず、方向を「出力」に変更してください。
4. 「レコードの読み取り」オペレーションの分岐では、ファイルから必要なレコードを取得してクライアントに返します。 そのため、このレコードのフィールドを返す出力フィールドのセットを定義する必要があります。

   左側のペインから lnk-b-details を「インターフェイスフィールド」ペインにドラッグします。

5. 手順 4 で作成した各フィールドの方向を「出力」に変更します。 lnk_b_title、lnk_b_author、および lnk_b_stockno は変更しません。

6. インターフェイスフィールドの lnk_b_title、lnk_b_author、および lnk_b_stockno の方向を「入出力」に変更します。

完了した「レコードの読み取り」オペレーション

図 7-5 は、完了した Read オペレーションを表しています。

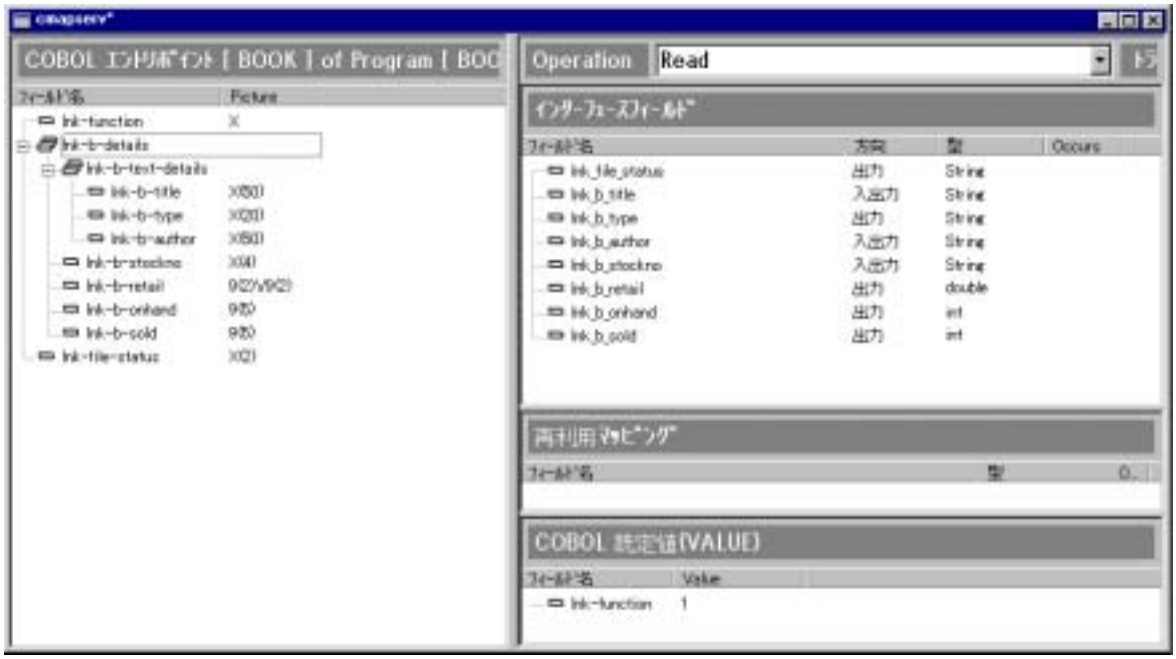


図 7-5: Read オペレーション (COM)

「レコードの削除」オペレーションの定義

「レコードの削除」オペレーションでは、「レコードの読み取り」オペレーションと同様に、在庫番号、タイトル、または作者を入力として受け付けます。lnk-b-details の他のデータ項目は無視されます。このオペレーションでは、在庫番号、タイトル、または、作者を示すレコードを削除します。

「レコードの削除」オペレーションを定義するには、次の手順を実行します。

1. [オペレーション > 新規作成] をクリックし、オペレーション名として Delete を入力し、[OK] をクリックします。
2. 前の説明に従って、COBOL 規定値を定義します。ソースを参照すると、lnk-function の値が 3 であることがわかります。
3. 前の説明に従って、lnk_file_status 出力パラメータを定義します。必ず、方向を「出力」に変更してください。

   ファイル状態を除き、出力フィールドは必要ありません。

4. 左側のペインで、lnk-b-details と lnk-b-text-details のツリーを展開します。
5. 左側のペインから lnk-b-stockno、lnk-b-title、および lnk-b-author を「インターフェイスフィールド」ペインにドラッグします。

完了した「レコードの削除」オペレーション

図 7-6 は、完了した Delete オペレーションを表しています。

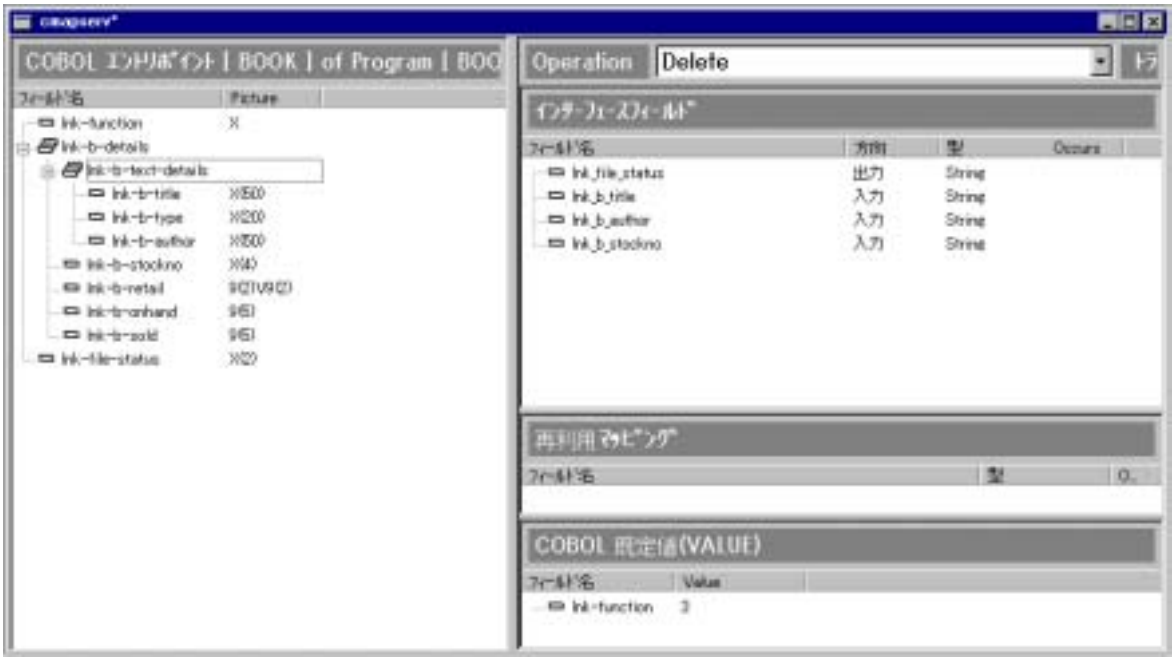


図 7-6：Delete オペレーション (COM)

定義したオペレーションの確認

COM インターフェイスでオペレーションを正しく定義できているかどうか再度確認するには、次の表と照合してください。タイトルバーのオペレーション名の次にあるドロップダウンリストを使用して、オペレーションを切り替えて表示できます。

	Add	Next	Read	Delete
機能コード	2	4	1	3
入力	詳細レコードのすべてのフィールド			タイトル、作者、在庫番号
出力	ファイル状態	詳細レコードの他のすべてのフィールド、ファイル状態	詳細レコードの他のすべてのフィールド、ファイル状態	ファイル状態

入出力	在庫番号	タイトル、作者、在庫番号
出力リターン		

更新された「サービスインターフェイス」ウィンドウ

ここまでで、このサービスで提供する 4 つのオペレーションを定義しました。

1. ボタンをクリックして、インターフェイスマッパーを閉じます。 保存するかどうかを確認するメッセージに対して [はい] をクリックします。
2. 「サービスインターフェイス」ウィンドウで、サービスのツリーを展開します。

   図 7-7 は、JMapServ のツリービューを表しています。定義した 4 つのオペレーションが表示されています。

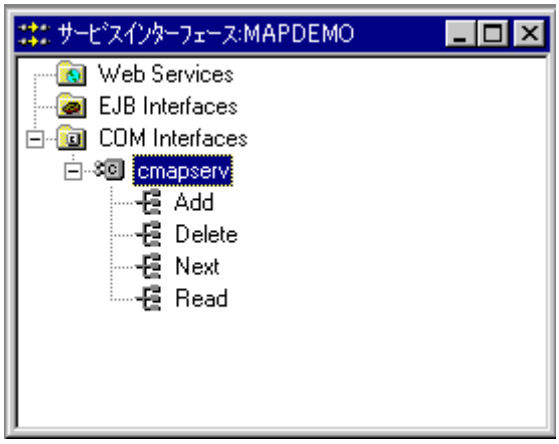


図 7-7: 更新された「サービスインターフェイス」ウィンドウ

作成されたファイル

作成されたファイルを表示するには、Net Express\Base\Demo\Mapdemo のサブフォルダを開きます。

マッピングウィザードでプロジェクトを自動的にビルドしたので、Debug サブフォルダに、book.int ファイルと book.idy ファイルが作成されています。 これらのファイルは、通常、ビルドによって作成されます。

Mapdemo\Repos (...Demo\Mapdemo\Mapdemo\Repos) サブフォルダ内には、次のファイルが作成されています。

book.xml	レガシープログラムが記述されています。
cmapserv.xml	サービスが記述されています。

EJB

『*COM*』の項で説明したように、ここでも、「Book」アプリケーションのサービスを作成します。ただし、このサービスは、EJB として作成します。 前の項で説明した作業をインターフェイスマッパーの高度な機能を使用して実行します。

新しいマッピングの作成

新しいマッピングを作成するには、次の手順を実行します。

1. [ファイル > 新規作成] をクリックし、「サービスインターフェイス」を選択して、[OK] をクリックします。

   マッピングウィザードの最初のページが表示されます。このページの後に「サービスインターフェイス：Mapdemo」というタイトルのウィンドウが表示されます。マッピングウィザードと「サービスインターフェイス」ウィンドウは、Interface Mapping Toolkit の一部です。

   マッピングの作成時にプロジェクトを開いておく必要はありません。プロジェクトが 1 つも開いていない場合は、「サービスインターフェイス」グループを作成して新しくマッピングを追加するためのダイアログボックスが表示されます。 ただし、通常は、プロジェクトを使用した方が便利です。

2. ウィザードのこのページで、作成するサービスのタイプを選択します。

   [COBOL を EJB としてマップ] をクリックします。[次へ] をクリックします。

3. ここでは、他のプロジェクトを指定したり、マッパーの後のページに情報を入力したりしないで、現在開いているプロジェクトからマッパーによって情報を取得します。

   「現在の Net Express プロジェクトを使用する」が選択されていることを確認して、[次へ] をクリックします。

4. 連絡節を含むプログラムの名前をマッパーに指定する必要があります。このプログラムが、サービスで使用する COBOL アプリケーションの最上位のプログラムになります。

   book.cbl (リストの 2 番目の項目) をクリックして選択し、[次へ] をクリックします。

   IDE で「出力」ウィンドウの [ビルド] タブを選択すると、プロジェクトが自動的にビルドされていることがわかります。これは、ビルドによって作成されたファイル (book.int 実行形式ファイルとデバッグ中に必要な book.idy ファイル) がサービスのディプロイに使用できる状態であることを表します。 デフォルトのビルドタイプが「一般デバッグビルド」なので、これらのファイルは、Mapdemo フォルダ内の Debug サブフォルダに格納されます。

5. このサービスを外部に認識させ、クライアントが起動できるようにするために、サービスの名前を定義します。

   JMapServ と入力します。大文字と小文字を区別して正しく入力してください。[次へ] をクリックします。

6. サービスとクライアントの間で受け渡すデータを COBOL プログラムの連絡節のパラメータにマップする方法を定義します。この操作をマッピングと呼びます。マッピングウィザードでは、デフォルトのマッピングを作成し、必要に応じて後で編集できます。

   「デフォルトのマッピング」が選択されていることを確認して、[次へ] をクリックします。

7. [終了] をクリックします。

   Book が更新されていることを警告するメッセージが表示されることがあります。このメッセージは、マッピングウィザードの使用中にプログラムが再コンパイルされたことを表します。プログラム自体は、変更されていません。

   2 つの新しいメニュー ([オペレーション] と [フィールド]) がメニューバーに追加されます。「サービスインターフェイス」ウィンドウが更新され、インターフェイスマッパーが開きます。次の項では、このウィンドウとインターフェイスマッパーについて説明します。

「サービスインターフェイス」ウィンドウ

「サービスインターフェイス」ウィンドウが 図 7-8 のように表示されます。この図では、Mapdemo プロジェクトに関連付けられた 3 種類のサービスが表示されています。この段階で作成されているサービスは、1 つです。このサービスが格納されているフォルダが、サービスの種類を表しています。このサービスには、Book という 1 つのオペレーションが含まれています。このチュートリアルでは、さらにオペレーションを追加します。

「サービスインターフェイス」グループは、Mapdemo プロジェクトのサービスインターフェイスセットです。これらのインターフェイスは、mapdemo.mpr というリポジトリに保存されています。通常は、「サービスインターフェイス」グループは、「サービスインターフェイス」ウィンドウで表示できるので、[ファイル] メニューからこのウィンドウを再度開く場合を除いて、ファイル名を記憶しておく必要はありません。

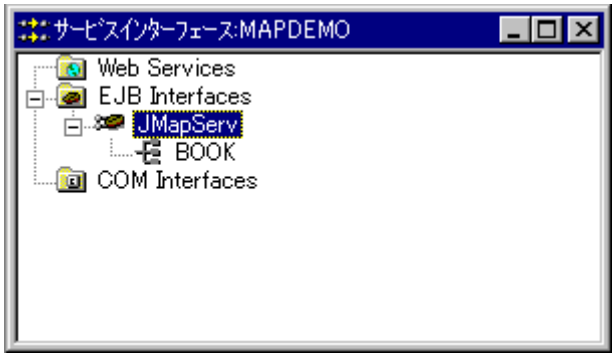


図 7-8: EJB が 1 つ表示された「サービスインターフェイス」ウィンドウ

インターフェイスマッパー

インターフェイスマッパーは、図 7-9 のように表示されます。インターフェイスマッパーは、ウィザードの [終了] をクリックすると、自動的に開きます。インターフェイスマッパーが開いていない場合は、「サービスインターフェイス」ウィンドウでサービス名をダブルクリックするか、またはサービス名を選択して [サービス > インターフェイスの編集] をクリックして、インターフェイスマッパーを開きます。

この図では、インターフェイスマッパーのツリービューがすべて展開されています。 作成したデフォルトのマッピングも表示されています。 ここでは、まず、このマッピングの意味を説明し、このマッピングを使用するかどうかを検討します。

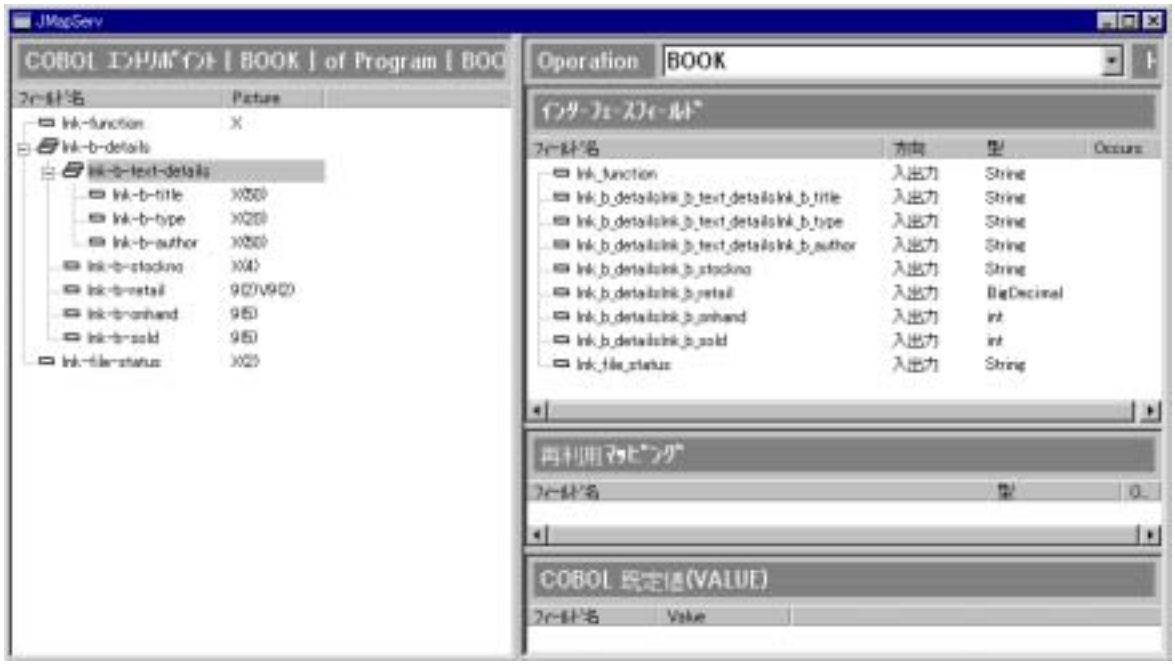


図 7-9: JMapServ のデフォルトのマッピング

デフォルトのマッピングでは、新しいサービスで提供する機能 (オペレーション。ここでは「Book」オペレーション) が定義されています。クライアントがこのオペレーションを起動すると、Book というエントリポイントで COBOL プログラムに制御が移ります。この簡単なプログラムでは、このエントリポイントが手続き部の開始点になります。左側のペインには、プログラムの連絡節が表示されています。

デフォルトのマッピングでは、インターフェイスのフィールド (サービスとクライアントの間でデータを受け渡すフィールド) が定義されています。右上のペインには、これらのフィールドが表示されています。 データは、これらのインターフェイスのフィールドからプログラムのパラメータにマップされます。フィールドは、連絡節の各項目に対して宣言されており、マッピングが自動的に設定されます。 各インターフェイスフィールドには、対応する連絡節の項目の PICTURE 文字列に最も近い Java または COM データ型が設定されています。

これらのフィールドの名前は、対応する連絡節の項目に基づいています。名前の競合を防ぐには、連絡節の項目名とその項目を含む集団項目の名前を続けて指定します。 ハイフンは、Java の変数名に使用できないので、アンダスコアに置換されます。 インターフェイスのフィールド名を指定する方法については、後で説明します。

インターフェイスのフィールドは、EJB で Bean 形式のインターフェイスを提供するための個別のフィールドです。 インターフェイスのフィールド名に COBOL のデータ構造を反映した階層構造を指定する方法については、後で説明します。

Java のパラメータは、入力、出力、または、入出力に使用できます。デフォルトのマッピングでは、すべてのフィールドが「入出力」に設定されます。

図 7-9 のタイトルバーの右側にある「**サポートされていないトランザクション**」は、コンテナで管理するトランザクションの EJB トランザクション属性です。 ここで選択した内容は、EJB ディプロイ記述ファイルの内容に影響します。 これは、デフォルト設定です。 このチュートリアルでは、説明を省略します。

不要なオペレーションの削除

この後の項では、『*サンプルプログラム*』の項で説明したサンプルプログラム「Book」のソースについて何度か説明します。

インターフェイスマッパーの説明では、デフォルトのマッピングを使用しましたが、ここでは使用しません。デフォルトのマッピングの場合は、クライアントからインターフェイスのフィールドにデータが渡され、インターフェイスフィールド lnk_function で受け取ったデータに応じてプログラムの EVALUATE 文で処理が選択されて、データ項目 lnk-function にマップされます。lnk-b-details と lnk-file-status の更新データがインターフェイスのフィールドに返されます。

ここでは、プログラムの 4 つの操作 (レコードの読み取り、レコードの追加、レコードの削除、および、次のレコードの取得) に個別の名前を付け、各操作 (オペレーション) に必要なデータのみをクライアントから受け取って個別に処理することにより、クライアントに合理的なサービスを提供します。

そのため、ここでは、デフォルトのマッピングで作成されたオペレーションを削除し、後の項で必要なオペレーションを作成します。

1. IDE のプルダウンメニューで [オペレーション > 削除] をクリックし、[はい] をクリックします。

   インターフェイスマッパー内のすべての項目が表示されなくなります。

「レコードの追加」オペレーションの定義

「レコードの追加」オペレーションを定義するには、次の手順を実行します。

1. IDE のプルダウンメニューで [オペレーション > 新規作成] をクリックします。
2. オペレーション名として「Add」を入力し、[OK] をクリックします。

インターフェイスフィールドの定義

ソース book.cbl を確認すると、ファイルにレコードを追加した場合のファイル状態が lnk-file-status に返されます。

1. lnk-file-status を「インターフェイスフィールド」ペインにドラッグします。

   lnk_file_status というインターフェイスフィールドが作成されます。 新しいフィールドと、そのフィールドの作成に使用した連絡節の項目の間のマッピングは自動作成されます。

2. lnk_file_status インターフェイスフィールドにある「方向」フィールドをダブルクリックし、ドロップダウンリストで方向を「出力」に変更します。

   その結果、出力フィールドが作成され、クライアントから受け取る入力メッセージではなく、クライアントに返す応答メッセージに含まれます。

   EJB の場合、指定できる方向は、入力、出力、および入出力です。

インターフェイスフィールドセットの定義

ソースを参照すると、「レコードの追加」オペレーションについては、索引ファイルに新しいレコードを追加するためのすべてのフィールドをクライアントプログラムから提供する必要があることがわかります。これらのフィールドの値は、Book プログラムの lnk-b-details データ項目に渡されます。

1. lnk-b-details を「インターフェイスフィールド」ペインにドラッグします。

各データ項目のインターフェイスフィールドが作成されます。ここでは、デフォルトの方向「入力」を使用します。これらのフィールドでは、lnk-b-details デフォルト項目に渡すデータを受け取ります。

型の変更

アプリケーションの内容と使用方法を理解しておくと、より高度なインターフェイスを作成できます。たとえば、小売価格フィールド lnk-b-retail が「99V99」として定義されているために、lnk_b_retail が BigDecimal にデフォルト設定されていることに注目してください。もし、すべての価格が整数単位のドルであることがわかっている場合は、lnk_b_retail を int に設定することができます。ここでは、このような状況を想定してみます。

1. 「インターフェイスフィールド」ペインの BigDecimal をダブルクリックします。
2. ドロップダウンリストから int を選択します。

COBOL 規定値の定義

必要なオペレーションを通知するパラメータは、lnk-function です。ただし、受け取るメッセージには、対応するパラメータではなく、オペレーション名「Add」が含まれています。ここでは、Add の要求を受け取ったときに、lnk-function に格納される値によって、プログラムで「レコードの追加」オペレーションが実行されるように指定する必要があります。ソースでは、この値が 2 となっています。lnk-function フィールドのようなフィールドを COBOL 規定値と呼びます。

1. 左側のペインから lnk-function を「COBOL 規定値 (VALUE)」ペインにドラッグします。
2. 値として 2 を入力し、[OK] をクリックします。

完了した「レコードの追加」オペレーション

図 7-10 は、完了した Add オペレーションを表しています。

[ファイル > 上書き保存] をクリックするか、または、Ctrl+S キーを押すと、編集内容をいつでも保存できます。

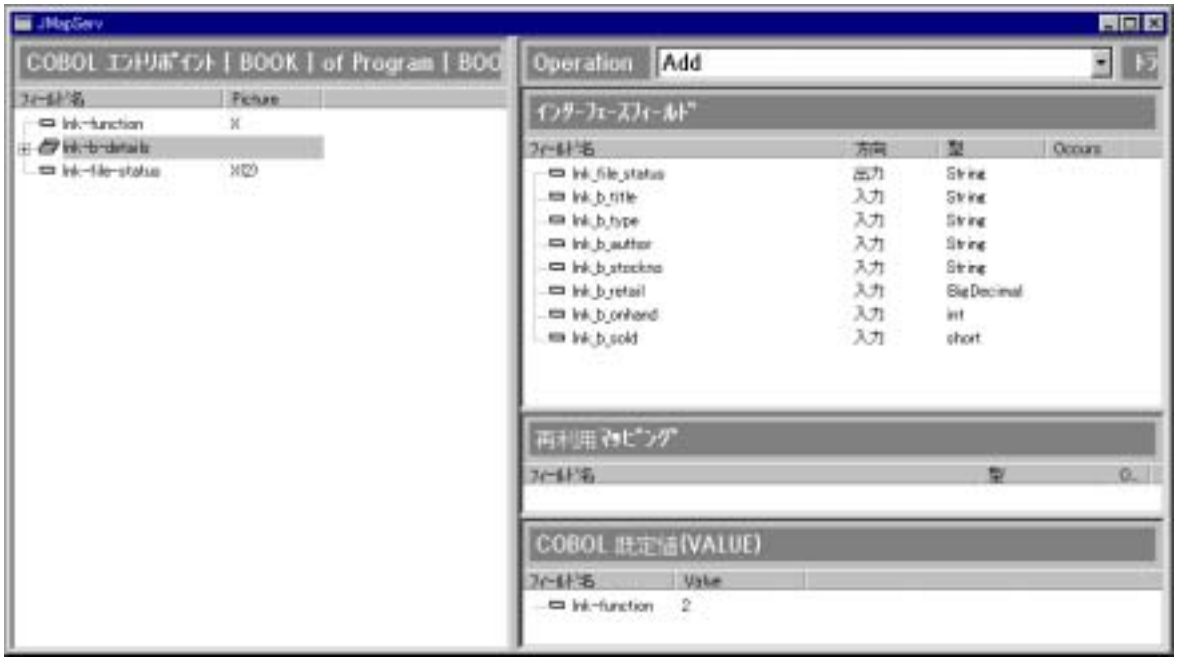


図 7-10: Add オペレーション (EJB)

詳細を説明します。

連絡節の項目とインターフェイスフィールドの対応関係は、ドラッグアンドドロップ操作の実行時に設定されます。インターフェイスフィールドを右クリックし、[マッピング] をクリックすると、ダイアログボックスが開き、関連付けられた連絡節の項目が表示されます。この対応関係は、インターフェイスフィールドの名前や順序とは関係ありません。このダイアログボックスで、マッピングを変更できます。

サービスとクライアントの間で渡されるメッセージでは、インターフェイスフィールドはキーワードで識別されます。キーワードには、「インターフェイスフィールド」ペイン内の名前が使用されます。「インターフェイスフィールド」ペインに表示されるインターフェイスフィールドの名前は、クライアントでも同じ順序で表示されるので、この順序を記憶しておくと便利です。フィールドの順序を変更するには、フィールドをドラッグします。

「次のレコードの取得」オペレーションの定義

ここで使用する索引ファイルの主キーは、在庫番号です。プログラムによる「次のレコードの取得」オペレーションでは、在庫番号を入力として受け付け、その在庫番号をもつレコードを検索して、そのレコードを返します。

このプログラムでは、このオペレーションを実行するために、連絡節の lnk-b-details 項目を入出力パラメータとして使用します。ただし、入力時には、lnk-b-stockno は、実際に使用される lnk-b-details の一部にすぎません。

1. タイトルバーの下にある「オペレーション」を右クリックし、表示されるポップアップメニューで [新規作成] をクリックします。 オペレーション名として「Next」を入力し、[OK] をクリックします。

   ポップアップメニューは、前に使用したプルダウンメニューのかわりに使用できます。

   新規作成したオペレーションの「インターフェイスフィールド」ペインと「COBOL 規定値 (VALUE)」ペインは、空白なので、インターフェイスフィールドと規定値を設定します。 連絡節は表示されたままなので、連絡節にアクセスしてフィールドと規定値を作成できます。

2. 『*COBOL 規定値の定義*』の項の説明に従って、COBOL 規定値を定義します。 ソースを参照すると、lnk-function の値が 4 であることがわかります。
3. 上記の『*インターフェイスフィールドの定義*』の項に従って、lnk_file_status 出力パラメータを定義します。 必ず、方向を「出力」に変更してください。
4. 「次のレコードの取得」オペレーションの分岐では、ファイルから必要なレコードを取得してクライアントに返します。 そのため、このレコードのフィールドを返す出力フィールドのセットを定義する必要があります。

   左側のペインから lnk-b-details を「インターフェイスフィールド」ペインにドラッグします。

5. 上記の『*型の変更*』の項に従い、lnk_b_retail を int に変更します。
6. 手順 4 で作成した各フィールドの方向を「出力」に変更します。
7. lnk_b_stockno フィールドは、出力フィールド、および、必要なレコードを識別するキーを入力するためのフィールドとして機能します。

   lnk_b_stockno を「I/O」に変更するだけでもかまいませんが、ここでは、インターフェイスマッパーの別の機能を説明するために、別の方法で入力フィールドを作成します。

   左側のペインから lnk-b-stockno を「インターフェイスフィールド」ペインにドラッグします。

   新しいインターフェイスフィールドの名前は、lnk_b_stockno1 と表示されます。 既存のフィールド名と競合しないように、フィールド名に「1」が自動的に追加されています。

完了した「次のレコードを取得」オペレーション

図 7-11 は、完了した Next オペレーションを表しています。

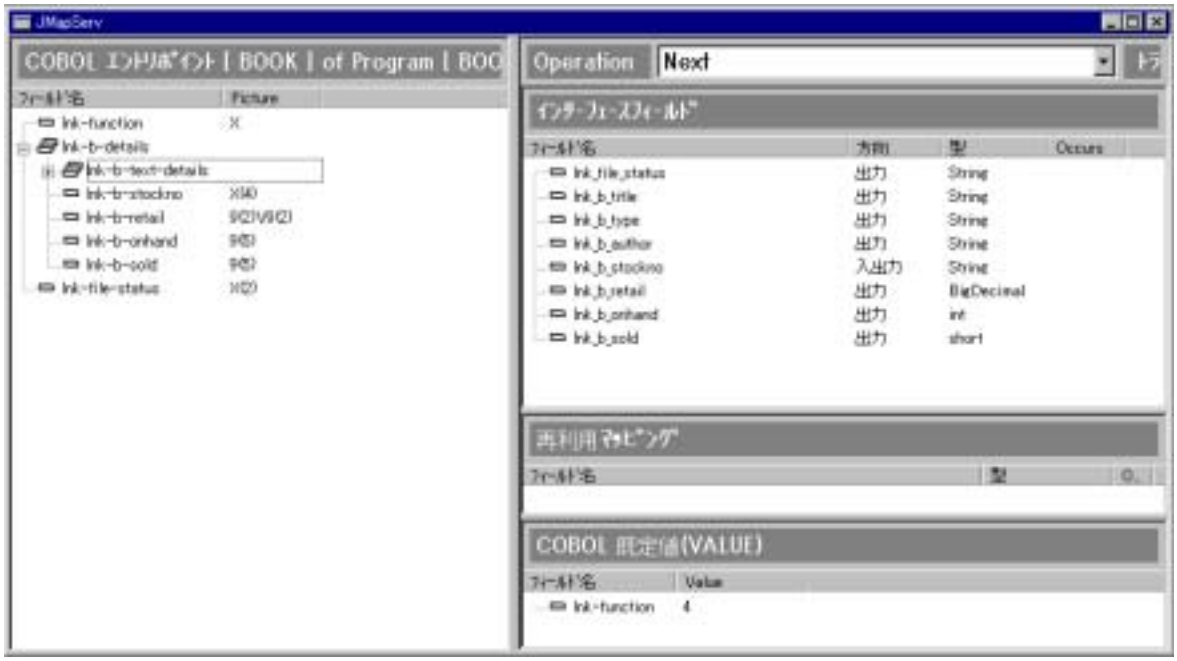


図 7-11: Next オペレーション (EJB)

「レコードの読み取り」オペレーションの定義

プログラムによる「レコードの読み取り」オペレーションでは、lnk-b-details を入出力パラメータとして使用します。このフィールドには、在庫番号、タイトル、または、作者を入力します。lnk-b-details の他のデータ項目は無視されます。また、lnk-b-details には、入力された在庫番号、タイトル、または、作者を示すレコードが出力されます。

1. [オペレーション > 新規作成] をクリックし、オペレーション名として Read を入力し、[OK] をクリックします。
2. 前の説明に従って、COBOL 規定値を定義します。ソースを参照すると、lnk-function の値が 1 であることがわかります。
3. 前の説明に従って、lnk_file_status 出力パラメータを定義します。 必ず、方向を「出力」に変更してください。
4. 「レコードの読み取り」オペレーションの分岐では、ファイルから必要なレコードを取得してクライアントに返します。 そのため、このレコードのフィールドを返す出力フィールドのセットを定義する必要があります。

   左側のペインから lnk-b-details を「インターフェイスフィールド」ペインにドラッグします。

5. 上記のように、lnk_b_retail を int に変更します。
6. 手順 4 で作成した各フィールドの方向を「出力」に変更します。

再利用マッピングの定義

インターフェイスフィールド lnk_b_title、lnk_b_author、および lnk_b_stockno を「I/O」に変更するだけでもかまいませんが、ここでは、インターフェイスマッパーの別の機能を説明するために、別の方法で入力フィールドを作成します。

1. lnk-b-details を「再利用マッピング」ペインにドラッグします。

   再利用マッピングとは、Java 用語でカスタムレコードを表します。

2. 「再利用マッピング」ペインの lnk_b_details 以下のツリーをすべて展開します。
3. 「再利用マッピング」ペインで、lnk_b_title、lnk_b_author、および lnk_b_stockno を除き、基本的な各フィールドを削除します。 フィールドを削除するには、フィールドをクリックして Delete キーを押すか、フィールドを右クリックして [削除] をクリックします。
4. 「再利用マッピング」ペインで、lnk_b_text_details を右クリックし、[グループ解除] をクリックします。

   ツリービューが閉じた場合は、再度開いて、作成したフィールドを確認します。 ここでは、図 7-12 に示すような再利用マッピングが定義されているはずです。

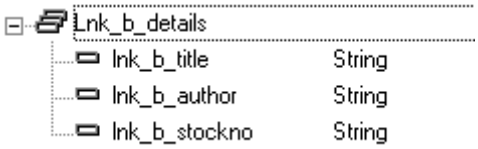


図 7-12: EJB に対して作成された再利用マッピング

5. 「再利用マッピング」ペインの lnk_b_details を「インターフェイスフィールド」ペインにドラッグします。 このフィールドは、「インターフェイスフィールド」ペインのどこにドラッグしても、一番下に追加されます。

   lnk_b_details タイプのインターフェイスフィールドが作成されます。 デフォルトのフィールド名も lnk_b_details です。 この名前を変更することもできますが、ここでは、デフォルト値を使用します。

COM についても同様の方法を適用できます。EJB の場合は、再利用マッピングを定義すると、Sun 社の CCI (Common Client Interface) で定義された CustomRecord インターフェイスに基づいてカスタムレコードが作成されます。 COM の場合は、新しい複合データ形式を作成できます。集団構造内の項目は、属性として機能します。どちらの場合も「再利用マッピング」ペインの機能と使用方法は同じです。

完了した「レコードの読み取り」オペレーション

図 7-13 は、完了した Read オペレーションを表しています。

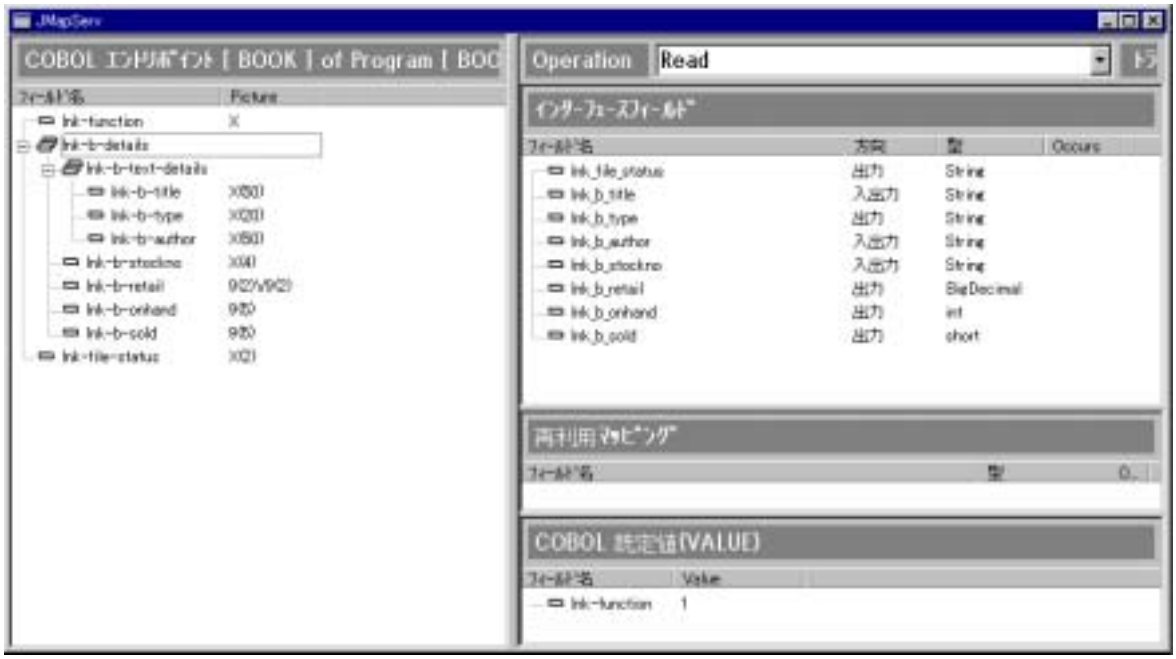


図 7-13: Read オペレーション (EJB)

「レコードの削除」オペレーションの定義

「レコードの削除」オペレーションでは、「レコードの読み取り」オペレーションと同様に、在庫番号、タイトル、または作者を入力として受け付けます。lnk-b-details の他のデータ項目は無視されます。このオペレーションでは、在庫番号、タイトル、または、作者を示すレコードを削除します。

「レコードの削除」オペレーションを定義するには、次の手順を実行します。

1. [オペレーション > 新規作成] をクリックし、オペレーション名として Delete を入力し、[OK] をクリックします。

   連絡節と同様に、現在表示されているオペレーションに関係なく、定義した再利用マッピングが表示されます。

2. 前の説明に従って、COBOL 規定値を定義します。ソースを参照すると、lnk-function の値が 3 であることがわかります。
3. 前の説明に従って、lnk_file_status 出力パラメータを定義します。必ず、方向を「出力」に変更してください。

   ファイル状態を除き、出力フィールドは必要ありません。

4. 「再利用マッピング」ペインの lnk_b_details を「インターフェイスフィールド」ペインにドラッグします。

完了した「レコードの削除」オペレーション

図 7-14 は、完了した Delete オペレーションを表しています。

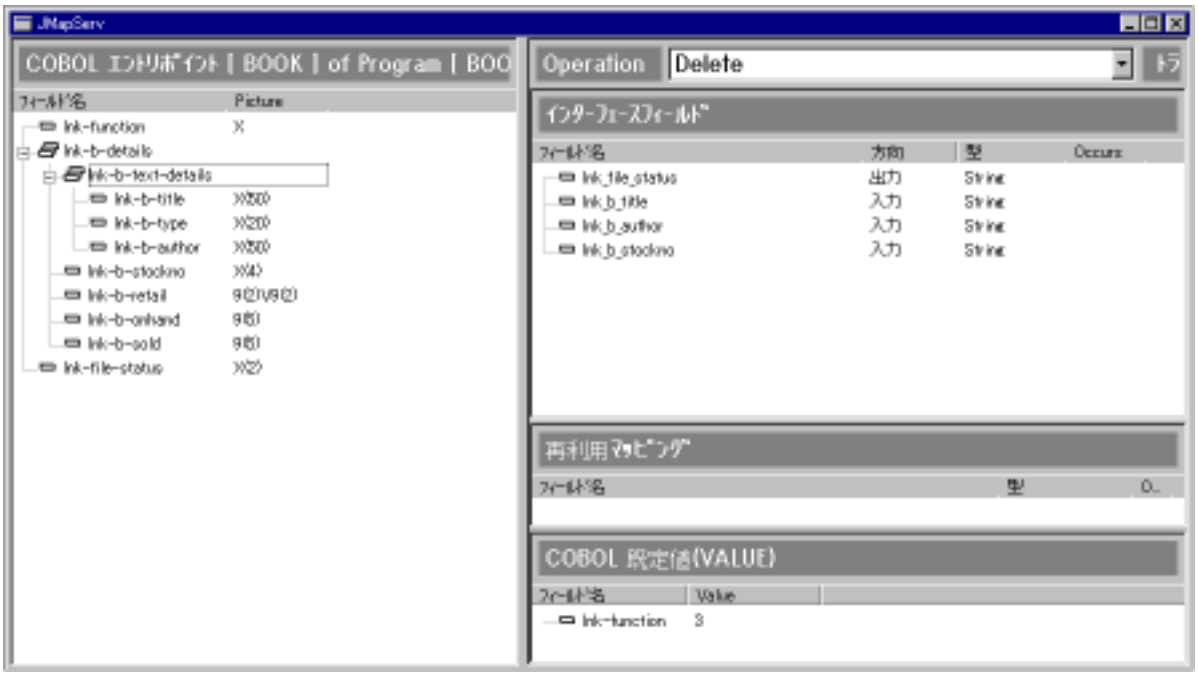


図 7-14：Delete オペレーション (EJB)

定義したオペレーションの確認

EJB でオペレーションを正しく定義できているかどうか再度確認するには、次の表と照合してください。 タイトルバーのオペレーション名の次にあるドロップダウンリストを使用して、オペレーションを切り替え表示できます。

	Add	Next	Read	Delete
機能コード	2	4	1	3
入力	詳細レコードのすべてのフィールド	在庫番号	タイトル、作者、在庫番号	タイトル、作者、在庫番号
出力	ファイル状態	詳細レコードの他のすべてのフィールド、ファイル状態	詳細レコードの他のすべてのフィールド、ファイル状態	ファイル状態

入
出
力

更新された「サービスインターフェイス」ウィンドウ

ここまでで、このサービスで提供する 4 つのオペレーションを定義しました。

1. ボタンをクリックして、インターフェイスマッパーを閉じます。 保存するかどうかを確認するメッセージに対して [はい] をクリックします。
2. 「サービスインターフェイス」ウィンドウで、サービスのツリーを展開します。

   図 7-15 は、JMapServ のツリービューを表しています。 定義した 4 つのオペレーションが表示されています。

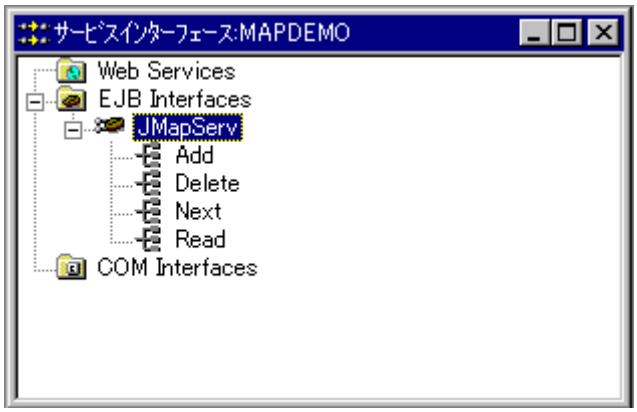


図 7-15: 更新された「サービスインターフェイス」ウィンドウ

作成されたファイル

作成されたファイルを表示するには、Net Express\Base\Demo\Mapdemo のサブフォルダを開きます。

マッピングウィザードでプロジェクトを自動的にビルドしたので、Debug サブフォルダに、book.int ファイルと book.idy ファイルが作成されています。 これらのファイルは、通常、ビルドによって作成されます。

Mapdemo\Repos (...Demo\Mapdemo\Mapdemo\Repos) サブフォルダ内には、次のファイルが作成されています。

book.xml	レガシープログラムが記述されています。
JMapServ.xml	サービスが記述されています。

Web サービス

『*COM*』と『*EJB*』の項で説明したように、ここでも、「Book」アプリケーションのサービスを作成します。ただし、このサービスは、Web サービスとして作成します。前の項で説明した作業をインターフェイスマッパーの高度な機能を使用して実行します。

マッピングの新規作成

新しいマッピングを作成するには、次の手順を実行します。

1. [ファイル > 新規作成] をクリックし、「サービスインターフェイス」を選択して、[OK] をクリックします。
2. [COBOL を Web サービスとしてマップ] をクリックして、[次へ] をクリックします。
3. 「現在の Net Express プロジェクトを使用する」が選択されていることを確認して、[次へ] をクリックします。
4. book.cbl を選択して、[次へ] をクリックします。
5. サービス名として「wmapserv」をクリックして、[次へ] をクリックします。
6. 前と同様に、このチュートリアルでは、マッピングウィザードで作成されるデフォルトのマッピングを使用して説明します。

   「デフォルトのマッピング」が選択されていることを確認して、[次へ] をクリックします。

7. [終了] をクリックします。

   Book が更新されていることを警告するメッセージが表示されることがあります。このメッセージは、マッピングウィザードの使用中にプログラムが再コンパイルされたことを表します。プログラム自体は、変更されていません。

8. [OK] をクリックします。

   前と同様にインターフェイスマッパーが開き、「サービスインターフェイス」ウィンドウが更新されて、新しいサービスが表示されます。

インターフェイスマッパー

図 7-16 は、インターフェイスマッパーと、作成されたデフォルトのマッピングを表しています。この図では、ツリービューがすべて展開されています。

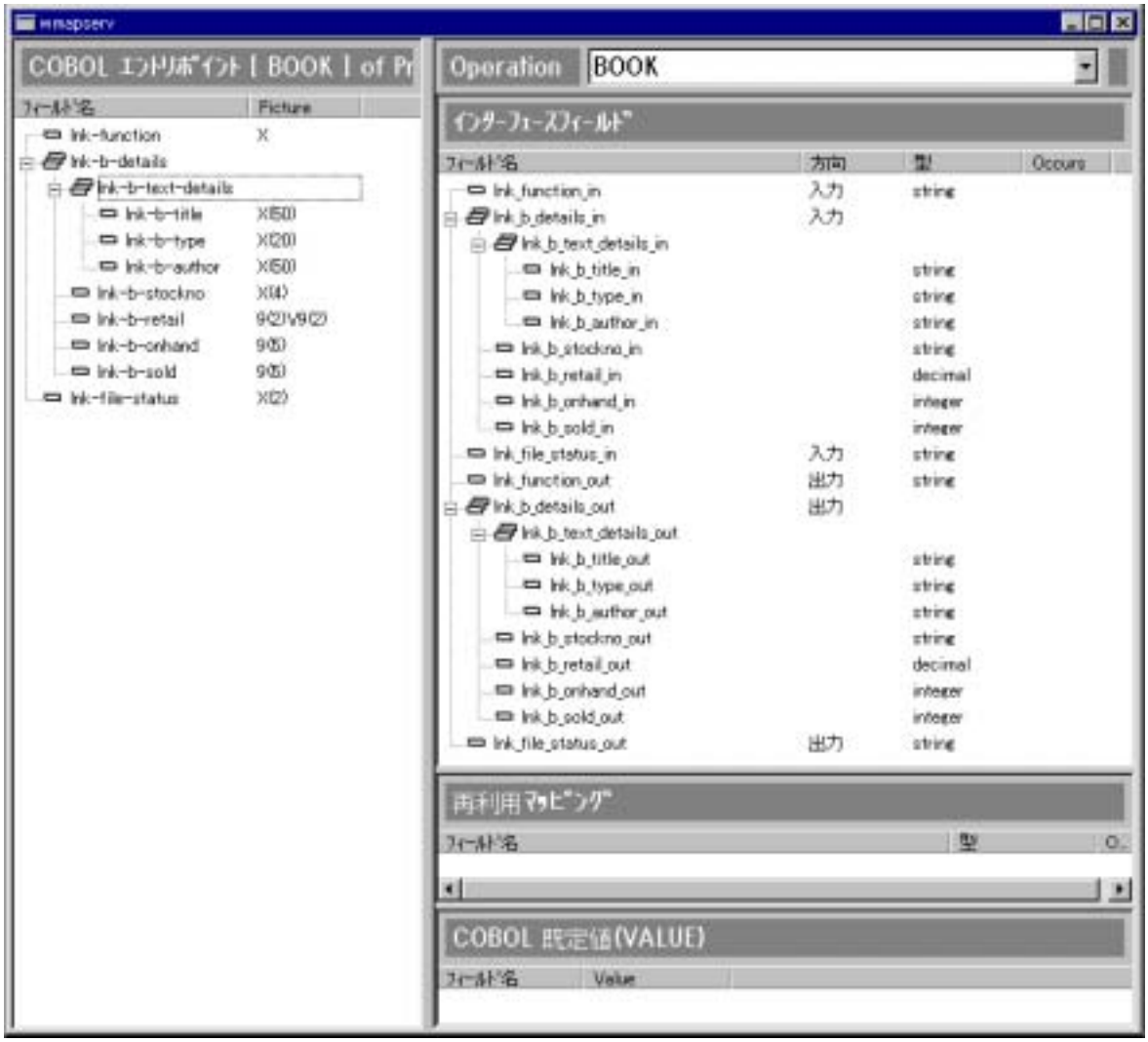


図 7-16: Web サービス wmapserv のデフォルトのマッピング

前と同様に、デフォルトのマッピングによって Book というオペレーションが 1 つ作成されています。

Web サービスの場合は、インターフェイスフィールドは、入力フィールドまたは出力フィールドであり、入力メッセージまたは応答メッセージに使用されます。 そのため、このデフォルトのマッピングでは、2 つのインターフェイスフィールド (入力フィールドと出力フィールド) が各連絡節に対して作成されます。

このインターフェイスフィールドは、連絡節の項目の階層構造に対応した階層構造になっています。 この分類は、XML メッセージに反映されます。

インターフェイスフィールドには、XML のデータ型が適用されます。また、これらのフィールドの名前は、対応する連絡節の項目に基づいています。 ハイフンは、アンダスコアに置換され

ます。連絡節の項目として 2 つのフィールドが宣言されているので、これらのフィールドに -in と -out が追加されています。

サービスとクライアントの間で渡されるメッセージでは、インターフェイスフィールドはキーワードで識別されます。キーワードには、「インターフェイスフィールド」ペイン内の名前が使用されます。「インターフェイスフィールド」ペインに表示されるインターフェイスフィールドの名前は、クライアントでも同じ順序で表示されるので、この順序を記憶しておくと便利です。

また、.NET を使用して Web サービス向けのクライアントを作成する場合は、.NET の規則として、WSDL ファイルの最初の出力パラメータは、出力戻りパラメータとしてクライアントプログラムに渡されることに注意してください。特定のフィールドを最初に指定するには、そのフィールドを「インターフェイスフィールド」ペインの最上部にドラッグします。その場合は、現在最上部にある項目の真上ではなく、その項目よりも前にドロップしてください。現在最上部にある項目が集団である場合はハイライトされ、フィールドをドロップすると、フィールドは集団の前でなく、集団内に配置されます。

「レコードの追加」オペレーションの定義

前の説明に従って、デフォルトのマッピングを削除し、独自のオペレーションを定義します。

1. [オペレーション > 削除] をクリックし、[はい] をクリックします。
2. [オペレーション > 新規作成] をクリックし、オペレーション名として Add を入力し、[OK] をクリックします。
3. COBOL 規定値 lnk-function を、前の説明に従って、2 として定義します。

インターフェイスフィールドの定義

前の説明に従って、ファイル状態を返すための出力フィールドを作成します。作成過程で、インターフェイスマッパーについても補足説明します。

1. lnk-file-status を「インターフェイスフィールド」ペインにドラッグします。
2. lnk_file_status インターフェイスフィールドにある「方向」フィールドをダブルクリックし、ドロップダウンリストで方向を「出力」に変更します。

   Web サービスの XML メッセージは一方向にしか送信できないので、EJB と COM に設定できた「入出力」を Web サービスに適用できません。

3. Net Express で Web サービスに対して作成される重要なファイルの 1 つに WSDL ファイル (サービスを正式に記述したファイル) があります。このファイルは、クライアントの作成に使用されます。自動生成されたフィールド名をそのまま使用することもできますが、WSDL ファイルやそのファイルから生成したクライアントコードを参照した場合に、フィールドの意味がわかりにくいので、ここでは、新しく名前を付けます。

   「インターフェイスフィールド」ペインの lnk_file_status をダブルクリックし、FileStatus という名前を上書き入力します。

EJB や COM 向けのクライアントを作成する場合も、インターフェイスフィールドを参照するので、EJB や COM についても、必要に応じてこの作業を実行してください。

再利用マッピングの定義

前の説明と同様に、インターフェイスフィールドセットを作成し、ファイルに追加する新しいレコードのデータを受け取ります。 ただし、Web サービスの場合は、EJB や COM と異なる問題があります。

生成する WSDL ファイルでは、基本的な COBOL 項目に対応する各フィールドがプリミティブな XML データ型で定義されています。 各集団項目には、複合形式が定義されています。 lnk-b-details は、集団項目であり、lnk-b-text-details 集団項目を含んでいます。

WSDL ファイルでは、各複合形式を一意の名前で宣言する必要があります。 ただし、lnk-b-details は、4 つのオペレーション中 3 つのオペレーションで使用されています (Delete オペレーションでは、レコードを受け取ったり、レコードを返したりすることはありません)。 3 つの各オペレーションについて lnk-b-details を「インターフェイスフィールド」ペインにドラッグすると、インターフェイスマッパーによって複合形式 lnk_b_details および lnk_b_text_details がそれぞれ (計 3 回) 宣言され、WSDL ファイルが無効になります。

この現象を防ぐには、インターフェイスマッパーを使用しないで、必要な複合形式を宣言します。

「再利用マッピング」ペインについては、『*EJB*』の項で説明しました。 このペインは、EJB、COM、および Web サービスのどの場合でも、同様に使用できます。EJB の場合は、再利用マッピングを定義すると、Sun 社の CCI (Common Client Interface) で定義された CustomRecord インターフェイスに基づいてカスタムレコードが作成されます。 COM の場合は、新しい複合データ形式を作成できます。集団構造内の項目は、属性として機能します。 Web サービスの場合は、新しい複合データ型が作成されます。 どの場合も効果は同じです。

1. lnk-b-details を「再利用マッピング」ペインにドラッグします。
2. 「再利用マッピング」ペインの lnk_b_details 以下のツリーをすべて展開します。
3. 上記の説明のように、再利用マッピングに適切な名前を付けることもできます。 この作業は省いてもかまいません。ただし、再利用マッピングに適切な名前を付けておくと、WSDL ファイルやその WSDL ファイルから生成されたクライアントを参照する場合に便利です。

   「再利用マッピング」ペインの各フィールド名をダブルクリックし、そのフィールド名の lnk_b_ を削除して、残った名前の先頭の文字を大文字にします。 たとえば、lnk_b_details を Details のように変更します。

4. Details を「インターフェイスフィールド」ペインにドラッグします。

   Details タイプのインターフェイスフィールドが作成されます。 デフォルトのフィールド名も Details です。 この名前を変更することもできますが、ここでは、デフォルト値を使用します。

完了した「レコードの追加」オペレーション

図 7-17 は、完了した Add オペレーションを表しています。

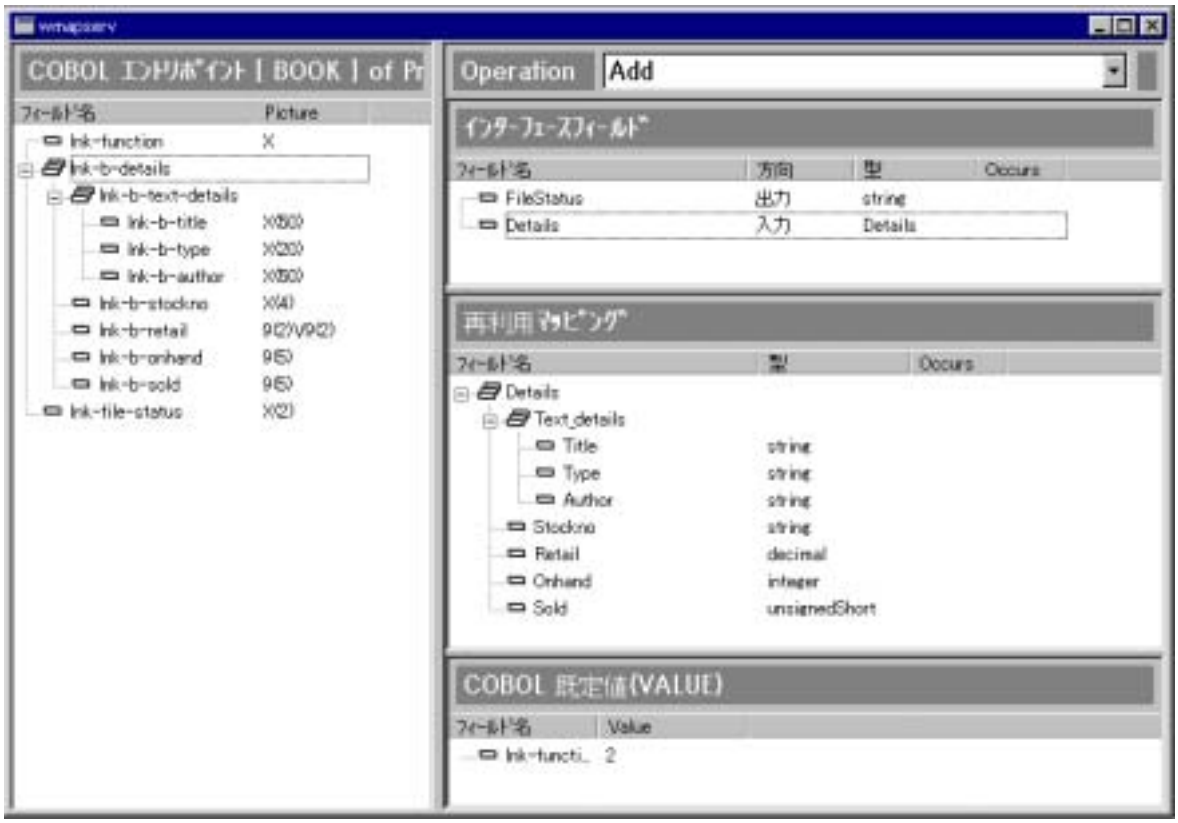


図 7-17: Add オペレーション (Web サービス)

「次のレコードの取得」オペレーションの定義

プログラムによる「次のレコードの取得」オペレーションでは、連絡節の項目 lnk-b-details 内の要求したレコードの値が返るように処理します。

1. [オペレーション > 新規作成] をクリックし、オペレーション名として Next を入力し、[OK] をクリックします。
2. COBOL 規定値 lnk-function を、前の説明に従って、4 として定義します。
3. 前の説明に従って、FileStatus インターフェイスフィールドを定義します。このフィールド名を lnk_file_status から変更し、方向を「出力」に設定します。
4. 前の説明に従って、Details を「インターフェイスフィールド」ペインにドラッグし、作成したインターフェイスフィールドの方向を「出力」に変更します。

再使用可能なマッピングの編集

このプログラムでは、このオペレーションを実行するために、lnk-b-details を入出力に使用します。ただし、入力時には、lnk-b-stockno は、実際に使用される lnk-b-details の一部にすぎません。

Web サービスでは、1 つのインターフェイスフィールドを入力と出力の両方に使用できません。そのため、lnk-b-stockno にマップする入力フィールドを別に作成する必要があります。

プログラムのソースを再度参照すると、残りの 2 つのオペレーション (Read と Delete) では lnk-b-stockno から入力を受け付けることがわかります。 同様に lnk-b-title と lnk-b-author からも入力を受け付けます。

オペレーションを定義する場合に、そのオペレーションに必要な各フィールドを「インターフェイスフィールド」ペインにドラッグし、必要に応じて名前を変更することもできます。 ただし、ここでは、「再利用マッピング」ペインを使用して、作業を簡略化します。

1. 左側のペインで、lnk-b-details のツリーを展開します。
2. lnk-b-stockno を「再利用マッピング」ペインにドラッグします。

   このときは、Details に従属するフィールドの間にドロップしないように注意してください。これらのフィールド間にドロップすると、lnk-b-stockno が Details に従属します。

3. 作成した再利用マッピングの名前を BookStockno に変更します。

   インターフェイスマッパーのペインに Stockno というフィールドがすでに存在するので、インターフェイスマッパーでこのマッピングに Stockno という名前を付けることができません。

4. lnk-b-title と lnk-b-author をドラッグすることもできますが、ここでは、別の機能を使用します。

   lnk-b-text-details (lnk-b-details の従属項目) を「再利用マッピング」ペインにドラッグします。

   「再利用マッピング」ペインのツリービューは、自動的に圧縮されます。

5. 作成した再利用マッピング lnk_b_text_details を右クリックし、[グループ解除] をクリックします。
6. 再利用マッピング lnk_b_type を選択し、Delete キーを押します。 確認メッセージに対して [はい] をクリックします。
7. lnk_b_title の名前を BookTitle に変更し、lnk_b_author の名前を BookAuthor に変更します。

   これらの機能は、「再利用マッピング」ペインの編集機能です。「インターフェイスマッピング」ペインでも同様の機能を使用できます。

これらの 3 つの再利用マッピングを作成することによって、作業が簡略化されます。1 つのマッピングの名前を変更して、合計で 3 つのマッピングを作成したので、残りの 2 つのオペレーションを作成するたびに、インターフェイスフィールドの名前を変更する必要がありません。 FileStatus も同様に処理できます。

8. BookStockno を「インターフェイスフィールド」ペインにドラッグします。

完了した「次のレコードを取得」オペレーション

図 7-18 は、完了した Next オペレーションを表しています。

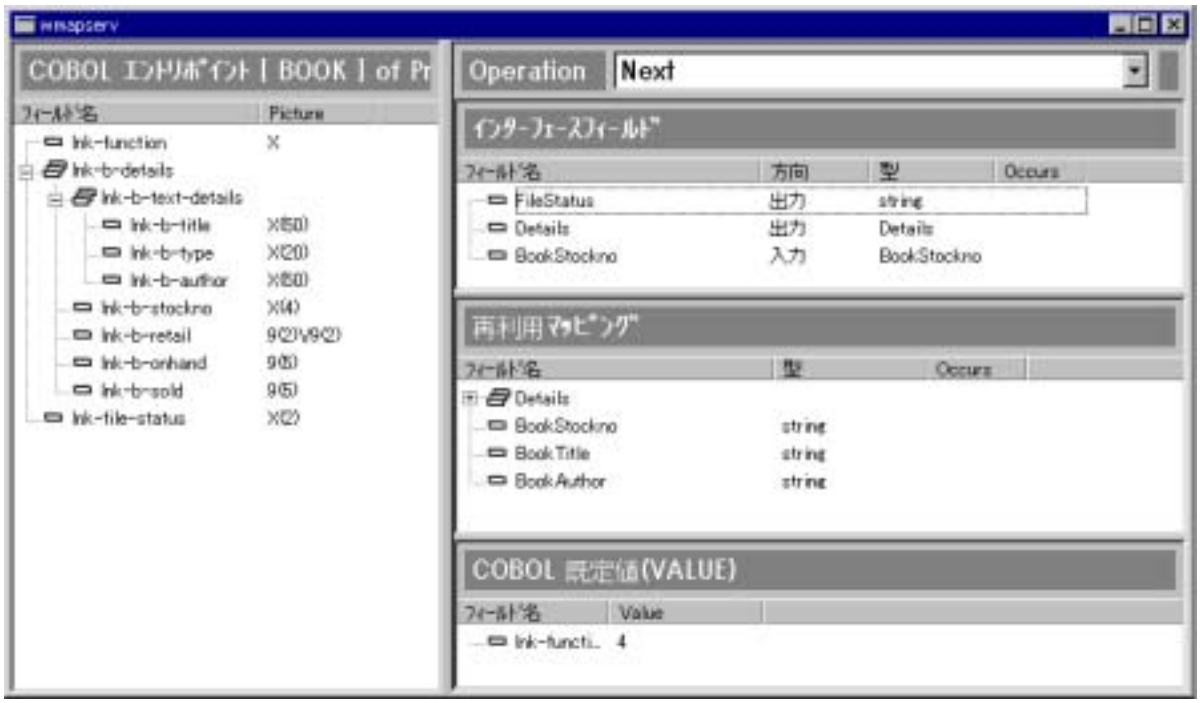


図 7-18: Next オペレーション (Web サービス)

「レコードの読み取り」オペレーションの定義

このオペレーションは、ここまでの作業と同様に簡単に定義できます。

1. [オペレーション > 新規作成] をクリックし、オペレーション名として Read を入力し、[OK] をクリックします。
2. COBOL 規定値 lnk-function を、前の説明に従って、1 として定義します。
3. 前の説明に従って、FileStatus インターフェイスフィールドを定義します。 このフィールド名を lnk_file_status から変更し、方向を「出力」に設定します。
4. 前の説明に従って、Details を「インターフェイスフィールド」ペインにドラッグします。作成したインターフェイスフィールドの方向を「出力」に変更します。

5. BookStockno、BookTitle、および BookAuthor を「インターフェイスフィールド」ペインにドラッグします。

完了した「レコードの読み取り」オペレーション

図 7-19 は、完了した Read オペレーションを表しています。

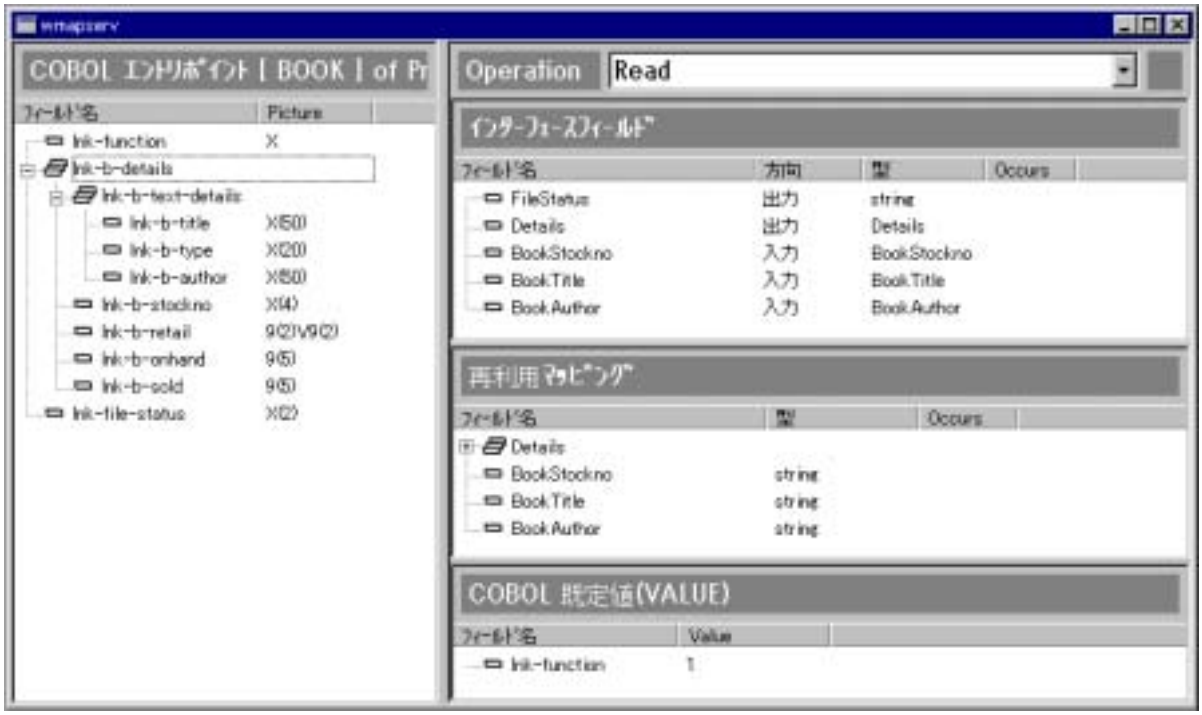


図 7-19: Read オペレーション (Web サービス)

「レコードの削除」オペレーションの定義

このオペレーションも簡単に作成できます。

1. [オペレーション > 新規作成] をクリックし、オペレーション名として Delete を入力し、[OK] をクリックします。
2. COBOL 規定値 lnk-function を、前の説明に従って、3 として定義します。
3. 前の説明に従って、FileStatus インターフェイスフィールドを定義します。このフィールド名を lnk_file_status から変更し、方向を「出力」に設定します。
4. BookStockno、BookTitle、および BookAuthor を「インターフェイスフィールド」ペインにドラッグします。

完了した「レコードの削除」オペレーション

図 7-20 は、完了した Delete オペレーションを表しています。

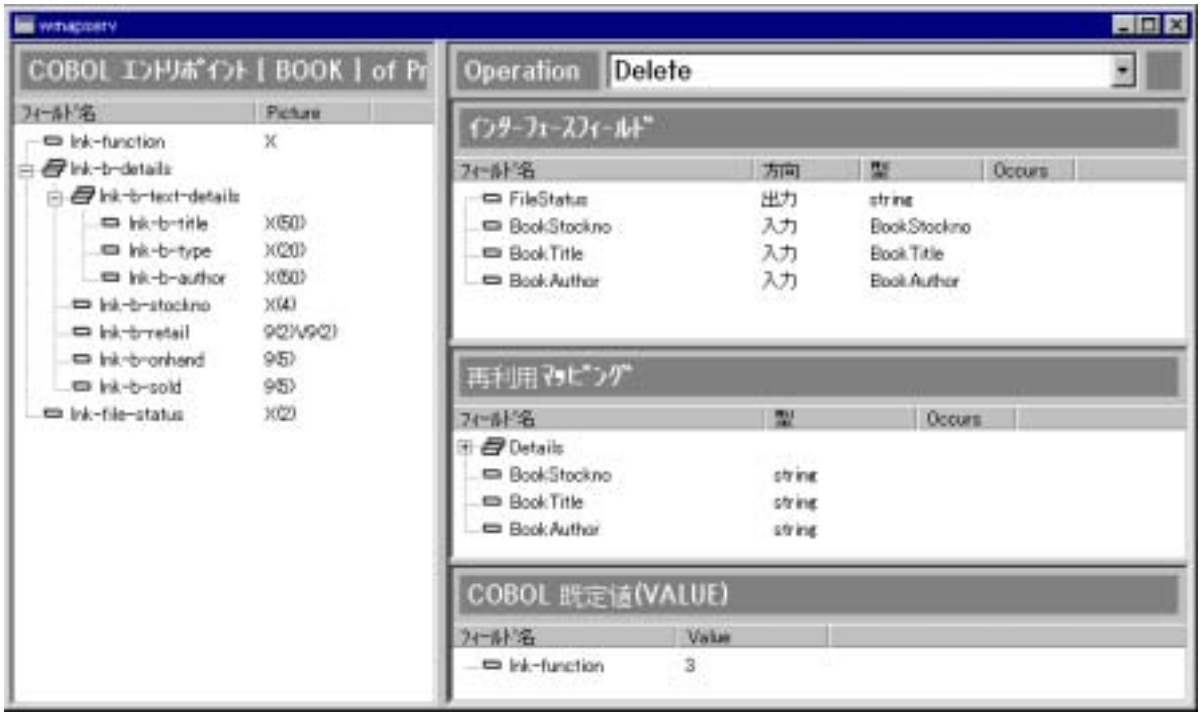


図 7-20：Delete オペレーション (Web サービス)

定義したオペレーションの確認

Web サービスでオペレーションを正しく定義できているかどうか再度確認するには、次の表と照合してください。

	Add	Next	Read	Delete
機能コード	2	4	1	3
入力	詳細レコード	在庫番号	タイトル、作者、在庫番号	タイトル、作者、在庫番号
出力	ファイル状態	詳細レコード、ファイル状態	詳細レコード、ファイル状態	ファイル状態

作成されたファイル

EJB と COM については、同じファイルを作成しました。『*サービスのディプロイ*』の章でサービスをディプロイすると、WSDL ファイルが作成されます。

次に進む前に

インターフェイスマッパーを閉じます。保存するかどうかを確認するメッセージに対して [はい] をクリックします。

サービスのディプロイ方法については、『*サービスのディプロイ*』の章を参照してください。

作業を中断する場合は、Net Express やプロジェクトを閉じてかまいません。その場合は、次の章に進んだときに、Net Express やプロジェクトを再度開いてください。

第 8 章 : サービスのディプロイ

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

概要

『*サービスの作成*』の章では、COBOL プログラムからサービスを作成しました。

ここでは、このサービスをディプロイします。

- 作成したサービスが Web サービスの場合は、Micro Focus Enterprise Server にディプロイします。
- 作成したサービスが EJB の場合は、Micro Focus Enterprise Server でマッピング情報をディプロイし、IBM WebSphere、BEA WebLogic などの Java アプリケーションサーバで EJB をディプロイします。
- 作成したサービスが COM インターフェイスの場合は、Net Express でディプロイし、Windows で登録します。

これらのすべての場合について、ここで説明します。

運用環境では、通常、Micro Focus Enterprise Server 製品を専用のサーバマシンで設定し、サイト管理者が管理します。 ただし、Net Express のインストール時に、オプションでユーザのマシンにもインストールできます。 このチュートリアルでは、サービスをディプロイするための情報を説明します。

準備

- COM

  Net Express でプロジェクト Mapdemo と「サービスインターフェイス」ウィンドウが開いていない場合は、開きます。「サービスインターフェイス」ウィンドウを開くには、[ファイル > サービスインターフェイスを開く] をクリックします。
  Net Express\Base\Demo\Mapdemo から mapdemo.mpr を選択し、[開く] をクリックします。

- Web サービス

  ここでは、Net Express と Mapdemo プロジェクトが必要です。 この段階では、Net Express と Mapdemo プロジェクトを開いておく必要はありません。

- EJB

ここでは、Net Express と Mapdemo プロジェクトが必要です。この段階では、Net Express と Mapdemo プロジェクトを開いておく必要はありません。

JDK (Java Development Kit) 1.3.x 以上をインストールし、PATH 環境変数に JDK bin ディレクトリを指定する必要があります。PATH 環境変数は、次のように設定します。

```
set path=jdk-installation\bin;%path%
```

COM インターフェイスのディプロイ

1. 「サービスインターフェイス」ウィンドウで cmapserv を右クリックして、[ディプロイ] をクリックします。

   進捗状況を表すメッセージボックスが表示されます。その他のメッセージは、IDE の下部の「出力」領域に表示されます。数秒後に正しくディプロイされたことを表すメッセージが表示されます。[OK] をクリックします。

   .dll ファイルがビルドされ、このサービスインターフェイスグループに使用するディプロイフォルダに格納されます。このフォルダに移動し、COM に .dll ファイルを登録します。

2. Windows のコマンドプロンプトで、ディレクトリを Net Express\Base\DEMO\Mapdemo\Mapdemo\REPOS\cmapserv.deploy に変更します。
3. 同じプロンプトで、次のコマンドを入力します。

   ```
   regsvr32 cmapserv.dll
   ```

   COM インターフェイスを再ディプロイする場合 (更新する場合やこのチュートリアルを再度実行する場合など) は、.dll ファイルを再登録する必要があります。これは、更新された .dll ファイルに新しい GUID (Globally Unique Identifier) が含まれているためです。GUID は、COM が各オブジェクトを識別するためのキーです。

4. .dll ファイルには、新しいインターフェイスが含まれていますが、元の COBOL アプリケーションは含まれていません。COBOL アプリケーションで .dll ファイルを呼び出せるようにするには、このアプリケーションの実行形式ファイルを、COBDIR 環境変数で指定したフォルダ内に格納する必要があります。

   [コントロールパネル > システム > 環境] をクリックし、Net Express\Base\DEMO\Mapdemo\Debug フォルダを COBDIR に追加します (COBDIR が存在しない場合は作成します)。

   マッピングウィザードでプロジェクトを自動的にビルドしたときに、このフォルダに book.int ファイルと book.idy ファイルが作成されています。

また、COBOL アプリケーションでデータファイルを検索できるようにする必要があります。詳細については、このマニュアルでクライアントに関するチュートリアルを参照してください。

COM インターフェイスのディプロイは、ここまでです。『*次に進む前に*』へ進んでください。

Enterprise Server の起動

エンタープライズサーバは、サービスを実行するための環境です。Enterprise Server 製品をインストールすると、マシンにエンタープライズサーバが作成されます。エンタープライズサーバは、複数作成できます。

エンタープライズサーバを管理するために、Enterprise Server 製品では、Web インターフェイスを含む Enterprise Server Administration という機能がインストールされます。そのため、エンタープライズサーバを Web のどこからでも管理できます。この場合は、エンタープライズサーバをローカルマシンから管理すると便利です。

1. [スタート > プログラム > Micro Focus Net Express > 構成 > Enterprise Server Administration] をクリックします。

   図 8-1 のように、「Enterprise Server Administration」のホームページが表示されます。バージョン番号のような詳細とステータスログは、マシンによって異なることがあります。

   Enterprise Server Administration を起動する別の方法として、Net Express IDE の [ツール > Enterprise Server Administration] をクリックするか、または、Web ブラウザを起動して、URL「http://localhost:86」を入力します。

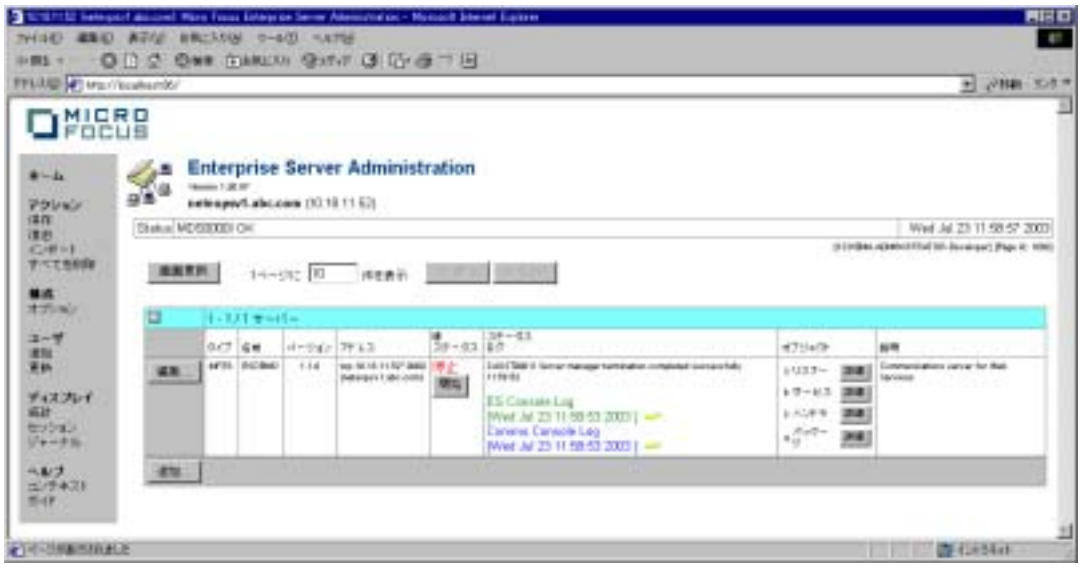


図 8-1: Server Administration のホームページ

この Web ページには、主に定義済みのエンタープライズサーバが一覧表示されます。エンタープライズサーバを定義する場合は、その構成を指定します。1 つのエンタープライズサーバは、デフォルトで定義されます。他のエンタープライズサーバを作成することもできますが、デフォルトのエンタープライズサーバを使用することをお勧めします。デフォルトのエンタープライズサーバの名前は、ESDEMO です。図 8-1 では、リスト内に ESDEMO のみが表示されています。このセッションでは、ESDEMO を使用します。

2. エンタープライズサーバ ESDEMO を起動するには、「現ステータス」列の [開始] をクリックします。

   構成オプションに応じて、「エンタープライズサーバコンソールデーモン」というコンソールウィンドウが表示される場合があります。このウィンドウには、進捗メッセージとエラーメッセージが表示されます。

3. コンソールウィンドウが表示される場合は、メッセージ「通信インターフェイスの初期化が完了しました。」が表示されるまで待機します。その他の場合は、数秒待機します。「現ステータス」列が「開始」に変化するまで、この Web ページの左側にある [画面更新] ボタン (ブラウザの [更新] ボタンではありません) をクリックします。

   コンソールウィンドウを表示するかどうかを構成するには、[編集] をクリックして、「ローカルコンソールを表示」をチェックします。[適用] をクリックします。この変更は、ESDEMO を次に起動したときに適用されます。Web ブラウザでコンソールメッセージを表示するには、「現ステータス」列の [詳細] をクリックし、次に [ES コンソール] をクリックします。

ディプロイ設定

ディプロイ前に詳細情報を指定する必要があります。この作業は、IDE で実行します。

Net Express でプロジェクト Mapdemo と「サービスインターフェイス」ウィンドウが開いていない場合は、開きます。「サービスインターフェイス」ウィンドウを開くには、[ファイル > サービスインターフェイスを開く] をクリックします。Net Express\Base\Demo\Mapdemo から mapdemo.mpr を選択し、[開く] をクリックします。

1. 「サービスインターフェイス」ウィンドウでサービス (wmapserv または JMapServ) を右クリックして、[設定] をクリックします。
2. Web サービスを作成したことがないコンピュータで初めて Web サービスを作成する場合は、図 8-2 のようなダイアログボックスが表示されます。

図 8-2:「サービスの名前空間」ダイアログボックス

このダイアログボックスでは、名前空間を指定できます。サービスに指定した名前と名前空間を合わせて、クライアントがサービスを起動するための名前を作成します。会社によっては標準の名前空間が指定されていることがあります。 指定されていない場合は、デフォルトのまま、http://tempuri.org/ を使用します。 このダイアログボックスは、再度表示されません。後で設定を変更するには、次のダイアログボックスの [詳細] ボタンを使用します。

[OK] をクリックします。

図 8-3 に示すようなダイアログボックスが表示されます。 [ディプロイメントサーバー] タブが選択されていることを確認します。

図 8-3:「マッピング属性」ダイアログボックス

このダイアログボックスについては、次の点に注意してください。

- **「Enterprise Server 名」フィールドには、ディプロイ時にサービスを受信するエンタープライズサーバを指定します。 サービスを実行するエンタープライズサーバを指定するわけではありません。 このエンタープライズサーバを指定するには、Net Express\Base\DEPLOY\.MFDEPLOY ファイルの「MFUS_SERVER=」行を編集します。 ここでは、ESDEMO を両方に使用します。ESDEMO は、Enterprise Server 製品のインストール時に自動作成されたエンタープライズサーバです。 組み込みのファイルでは、ESDEMO が指定されているので、このファイルを編集する必要はありません。 このフィールドの設定は、すぐに終わります。**
- **「Enterprise Server 実行時環境の使用」チェックボックスとボタンを選択すると、サービスの実行環境を設定するためのダイアログボックスが表示されます。 このダイアログボックスでは、スイッチ、チューナ、環境変数などを設定できます。 このチュートリアルでは、このチェックボックスを使用しません。**

- 「サービス名」フィールドには、クライアントがサービスにアクセスするための名前を指定します。Web サービスの場合は、サービスの作成時に指定した名前の後に、上の説明で指定した名前空間を続けて指定します。EJB の場合は、COBOL プログラム名を指定します。ここでは、変更する必要はありません。

3. 「Enterprise Server 名」フィールドの次の [変更] ボタンをクリックします。
4. [ESDEMO] をクリックして選択し、[OK] をクリックします。
5. [アプリケーションファイル] タブをクリックし、「レガシーアプリケーションをディプロイする」をクリックします。
6. [ファイルを追加] ボタンを使用して、Mapdemo フォルダの bookfile.dat ファイルと bookfile.idx ファイル、および、Mapdemo\Debug フォルダの book.int ファイルと book.idy ファイルを追加します。このダイアログボックスは、図 8-4 のように表示されます。

図 8-4：ディプロイするアプリケーションファイル

ここに表示されるファイルは、ディプロイ時にエンタープライズサーバにコピーされます。.int ファイルは、実行形式ファイルです。.idy は、サービスのデバッグに必要です。これらの .dat ファイルと .idx ファイルは、前の章『*サービスの作成*』の冒頭で 1 つのレコードを追加したデモに含まれている索引ファイルのデータと指標です。

デフォルトでは、サービスを実行するための現在のディレクトリは、サービスの実行を開始するディレクトリに設定されます。 そのため、サービスでは、Mapdemo フォルダの元のファイルではなく、そのサービスに対してディプロイされたデータファイルが選択されます。

7. EJB のみ
    1. [EJB の生成] タブをクリックします。
    2. 「クラスパス」フィールドに EJB コネクタクラスへのパスを指定します。 WebSphere の場合は、c:\Program Files\Websphere\AppServer\lib\j2ee.jar のようなパスを指定します。 WebLogic の場合は、c:\bea\weblogic700\server\lib\weblogic.jar のようなパスを指定します。
    3. 他のフィールドについては、デフォルト値を使用します。
8. [OK] をクリックします。

サービスのディプロイ

詳細を入力したので、次にサービスをディプロイします。

1. 「サービスインターフェイス」ウィンドウで、ディプロイするサービス (wmapserv または JMapServ) が選択されていることを確認し、[サービス > ディプロイ] をクリックします。

    ディプロイの進捗状況を表すログが表示されます。 数秒後に正しくディプロイされたことを表すメッセージが表示されます。

    ディプロイが失敗した場合は、理由が「ディプロイの進捗状況」ウィンドウに表示されます。 このエラーメッセージには、ログファイルのパスも表示されるので、ブラウザでこのパスを指定してログファイルを開くこともできます。

2. [OK] をクリックします。

    「エンタープライズサーバコンソールデーモン」ウィンドウにもディプロイが成功したことを表すメッセージが表示されます。

    この段階で IDE を最小化してもかまいません。

3. Web ブラウザで Enterprise Server Administration のホームページを表示して、この Web ページの左側にある [画面更新] ボタン (ブラウザの [更新] ボタンではありません) をクリックします。

    「オブジェクト」列に 6 つのサービスと 4 つのパッケージが表示されていることに注意してください。更新前に表示されていたのは、2 つのサービスのみで、パッケージは表示されていませんでした。

4. 「6 サービス」の次にある [詳細] をクリックします。

ESDEMO エンタープライズサーバから使用できるサービスが Web ページに表示されます。ディプロイした 4 つのサービスの他に、デフォルトの 2 つのシステムサービスもあります。新しいサービスは、「使用可能」と表示され、クライアントから起動できることを表します。このページは、図 8-5 に示されています。

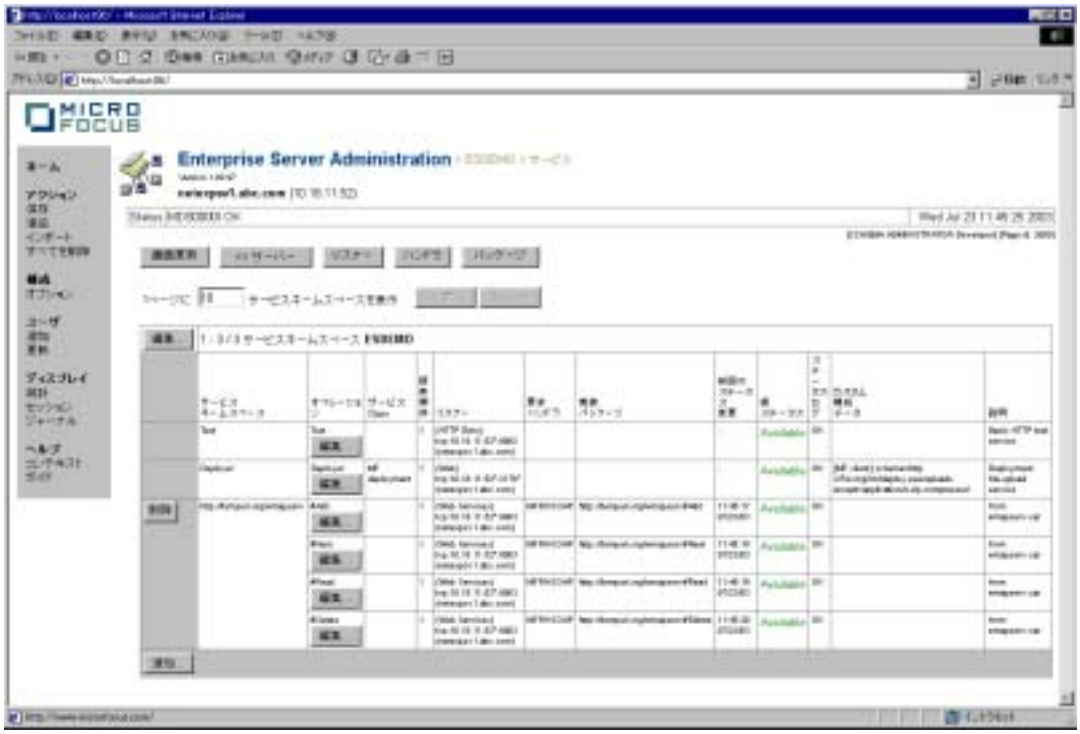


図 8-5: ディプロイしたサービス

5. リストの確認が終了した後で、Web ページの最上部にある [<< サーバ] リンクをクリックし、Enterprise Server Administration のホームページに戻ります。

作成されたファイル

ディプロイ用に作成されたファイルを表示するには、Net Express\Base\Demo\Mapdemo のサブフォルダを開きます。

Mapdemo\Repos (...Demo\Mapdemo\Mapdemo\Repos) サブフォルダ内には、次のファイルが作成されています (servicename は、wmapserv、JMapServ、または cmapserv です)。

servicename.properties　　指定したディプロイ設定の定義。

Mapdemo\Repos\servicename.deploy サブフォルダには、次のファイルを含め、いくつかのファイルが作成されています。

servicename.car	サービスの COBOL アーカイブファイル。マッピング情報、COBOL ファイル、データファイルなど、ディプロイする必要があるすべてのファイルが格納されています。内容を表示するには、Winzip を使用します。
servicename.idt	マッピング情報が含まれています。このファイルは、先に説明した .car ファイルにパッケージ化されています。
wmapserv.wsdl (Web サービスのみ)	サービスとのインターフェイスを定義する XML ファイル。このファイルは、クライアントアプリケーションの作成者に配布します。
cmapserv.dll (COM のみ)	ビルドサービスを含む動的リンクライブラリ。このファイルを Windows に登録し、Debug\book.int とともにクライアントで使用できるようにします。
JMapServ.jar (EJB のみ)	生成された EJB に含まれる Java アーカイブファイル。Java システムをインストールしている場合は、jar コマンドを実行すると、このファイルの内容を表示できます。
com\mypackage\JMapServ (EJB のみ。次のファイルを含む。)	次のソースコード
• JMapServ.java • JMapServBean.java • JMapServ.Homejava • Lnk_b_details.java • *.class	• • JMapServ Bean クラス • ホームインターフェイス • リモートインターフェイス • カスタムレコード • コンパイルされたクラス

削除と再ディプロイ

Web サービスまたは EJB を再ディプロイする必要がある場合には (更新する場合やこのチュートリアルを再度実行する場合)、まず、Web サービスまたは EJB をエンタープライズサーバから削除してください。

このチュートリアルを再度実行する場合は、前回に作成したサービスも削除します。Web サービスまたは EJB の削除と再ディプロイが不要な場合は、次の手順を実行しないでください。次の章を実行するために、サービスをディプロイしたままにしておく必要があります。

サービスを削除するには、次の手順を実行します。

1. 「Enterprise Server Administration」ホームページで「オブジェクト」列の「サービス」の行にある [詳細] をクリックします。
2. 左側の列で wmapserv の行の [削除] をクリックします。
3. 「関連付けられたパッケージの削除」にチェックマークが設定されていることを確認します。
4. メッセージ「良いですか？」の下の [削除] をクリックします。

5. [サーバ] をクリックして、ホームページに戻ります。

   サービス、および、関連付けられたパッケージが削除されたことを表すメッセージがコンソールウィンドウ内に表示されます。 ホームページに戻ると、このサービスは表示されなくなっています。

サービスを再ディプロイするには、次の手順を実行します。

1. IDE の「サービスインターフェイス」ウィンドウで、サービス (wmapserv または JMapServ) を右クリックして、[ディプロイ] をクリックします。

Web サービスのディプロイの完了

Web サービスをディプロイする場合は、この段階でディプロイ作業が完了しているので、『*次に進む前に*』へ進んでください。

EJB をディプロイする場合は、作業が残っているので、次へ進んでください。

EJB とリソースアダプタのディプロイ

EJB をディプロイする場合は、2 つの作業が必要です。COBOL 側をエンタープライズサーバにディプロイし、Java 側を J2EE アプリケーションサーバにディプロイします。 このチュートリアルの『*サービスのディプロイ*』では、COBOL 側をエンタープライズサーバにディプロイし、EJB を生成して、Java アーカイブファイル JMapServ.jar にパッケージ化しました。

ここでは、次の Java を J2EE プラットフォームにディプロイする必要があります。

- 組み込みのリソースアダプタ - mfcobol-notx.rar。 リソースアダプタを使用すると、EJB が J2EE アプリケーションサーバ経由でエンタープライズサーバと通信できます。リソースアダプタは、J2EE コネクタとも呼ばれ、リソースアーカイブファイル (.rar) にパッケージ化されています。
- アプリケーションのファイルをすべて含むエンタープライズアーカイブファイル - MapDemo.ear。 MapDemo.ear のバージョンがいくつか組み込まれています (サードパーティのアプリケーションサーバごとに 1 つのバージョン)。 各 MapDemo.ear には、次のファイルが格納されています。
  - MapDemo.war - この Web アーカイブファイルにはクライアントソフトウェア (JSP、servlet、Bean クラスなど) が格納されています。
  - JMapServ.jar - 既製の EJB などを含むアーカイブファイル。 この既製の JMapServ.jar ファイルを、生成された JMapServ.jar ファイルに置き換える必要があります。
  - mfejblib.jar - CustomRecord と RuntimeProperties の Micro Focus サポートを含むアーカイブファイル。 .ear ファイルのルートに格納する必要があります。詳細については、『*ディプロイガイド*』を参照してください。

- さまざまなディプロイ記述ファイルとマニフェストファイル - 一部のディプロイ記述ファイルは、サードパーティのアプリケーションサーバ専用なので、アプリケーションサーバに固有の .ear を選択する必要があります。

このチュートリアルでは、2 つのサードパーティの J2EE アプリケーションサーバ (IBM WebSphere 5.0 と BEA WebLogic 7.0) のディプロイ方法を説明します。これらのアプリケーションサーバのどちらかをインストールして構成します。このチュートリアルでは、WebLogic 7.0 と WebSphere 5.0 Advanced Edition について説明します。WebLogic または WebSphere の別のバージョンを使用している場合は、このチュートリアルとやや異なることがあります。ただし、基本的には、同じ方法を使用します。

WebLogic へのディプロイ

WebLogic は、サーバコンソールに付属しており、EJB とリソースアダプタのディプロイに使用します。サーバコンソールの使用方法は、コンソールからオンラインで表示できます。コンソールを使用して、左側のペイン内のツリーを移動し、左側の項目をクリックして、必要なページを表示します。

WebLogic Server の設定と起動

WebLogic の設定には、ディプロイする EJB とリソースアダプタを格納するドメインの設定と WebLogic の実行環境の設定が含まれます。

WebLogic Server は、次のように設定して起動します。

1. ドメイン構成ウィザードを使用してドメインを設定します。
    1. [スタート > プログラム > BEA WebLogic Platform 7.0 > ドメイン構成ウィザード] をクリックします。
    2. [WLS ドメイン] をクリックします。
    3. 次のように、ウィザードの質問に答えながら進みます。

必要な詳細情報	選択または指定する詳細情報
ドメインテンプレート名	WLS ドメイン
ドメイン名	MicroFocus
サーバタイプ	単独のサーバ (スタンドアロンサーバ)
サーバ名	MFserver
インストールディレクトリ	bea_installation\user_projects
受信アドレス	このフィールドは、空白でもかまいません。空白にした場合は、localhost が指定されます。
受信ポート	7001
SSL 受信ポート	7002
Windows サービス	はい

Windows スタートメニュー はい

ユーザ名 どのユーザ名でも入力できます。

パスワード **********

2. 次のように、起動コマンドファイルを編集して環境を設定します。
    1. テキストエディタで bea\user_projects\MicroFocus\StartWebLogic.cmd を開きます。
    2. ファイルの末尾に進みます。
    3. 行「@rem Call WebLogic Server」の上に、次の行を追加します。
        - set MF_HOME=NX\base\bin (NX は、Net Express のインストールディレクトリです)
        - set JAVA_HOME=bea\jdk (bea\jdk は、WebLogic 用にインストールした JDK の位置です)。WebLogic 用以外にインストールした JDK を指定しないでください。
        - set classpath=%JAVA_HOME%\lib\tools.jar;%MF_HOME%\mfconnector.jar;%MF_HOME%\mfcobolpure.jar
    4. ファイルを保存します。
3. [スタート > プログラム > BEA WebLogic Platform 7.0 > ユーザプロジェクト > MicroFocus > サーバの起動] をクリックして、WebLogic Server を起動します。編集した StartWebLogic.cmd が実行されます。
4. サーバが正しく起動するまで待ちます。
5. WebLogic のサーバコンソールを次のように起動します。
    1. ブラウザを開いて、URL「http://localhost:7001/console」を入力します。
    2. WebLogic ドメインの構成時に指定したユーザ名とパスワードを入力します。

WebLogic へのリソースアダプタのディプロイ

リソースアダプタをディプロイするには、WebLogic のサーバコンソールに移動し、次の手順を実行します。

1. 左側のペインで [MicroFocus > ディプロイ > コネクタ] の順に展開します。左側の階層に、Micro Focus のリソースアダプタがディプロイされていないことが表示されます。
2. [新しいコネクタコンポーネントの構成] をクリックして、このページの手順を実行します。リソースアダプタファイルをアップロードするようにメッセージが表示された後、NX\base\bin\mfcobol-notx.rar からアップロードします (NX は、Net Express のインストールディレクトリです)。
3. リソースアダプタがディプロイを完了するまで待ちます。このページの右側にあるディプロイアクティビティテーブルを調べ、リソースアダプタが正しくディプロイされたことを確認します。リソースアダプタは、左側の階層にも表示されます。

接続ファクトリを作成したり、リソースアダプタの JNDI 名を指定したりする必要はありません。これらは、接続ファクトリのディプロイ記述ファイルですでに指定されています。ディプロイ記述ファイル weblogic-ra.xml の該当する行は、次のとおりです。

```
<connection-factory-name>CCIMFCobol_v1.0</connection-factory-name>
<jndi-name>eis/MFCobol_v1.0</jndi-name>
```

このファイルを表示するには、上記の位置で mfcobol-notx.rar を検索し、weblogic-ra.xml を抽出します。

WebLogic への EJB のディプロイ

生成した EJB をディプロイする前に、その EJB と他のアプリケーションファイルをエンタープライズアーカイブファイル (.ear) としてパッケージ化する必要があります。 Net Express には、WebLogic 向けの MapDemo アプリケーションに使用するアプリケーションアーカイブファイル (MapDemo-WL.ear) が組み込まれています。

EJB をディプロイするには、次の手順を実行します。

1. 次のように、MapDemo アプリケーションに使用する組み込みの .ear ファイルに JMapServ.jar を追加します。
    1. NX\base\demo\MapDemo\Client\Java\lib ディレクトリ (NX は、Net Express のインストールディレクトリ) に移動し、MapDemo-WL.ear ファイルを同じディレクトリの MapDemo.ear にコピーします。

       アーカイブファイル名はそのアーカイブファイル内にビルドされているので、通常は、アーカイブファイル名を変更すると、そのファイルが機能しません。 その場合は、このファイルをコピーするときに、ファイルの元の名前 MapDemo.ear に変更します。

    2. MapDemo.ear と同じディレクトリに、NXbase\demo\MapDemo\MapDemo\repos\JMapServ.deploy ディレクトリから JMapServ.jar をコピーします。
    3. 次のコマンドを使用して、JMapServ.jar を MapDemo.ear に追加します。

       jar - uvf MapDemo.ear JMapServ.jar

2. WebLogic のサーバコンソールで [MicroFocus > ディプロイ > アプリケーション] の順に展開します。
3. 右側の [新しいアプリケーションの構成] をクリックして、手順を実行します。 デフォルト値を使用して、次のように指定します。

プロンプト	指定内容
ear ファイルへのパス	NX\base\demo\MapDemo\Client\Java\lib\MapDemo.ear NX は、Net Express のインストールディレクトリです。
ターゲットサーバ:	MFServer

4. アプリケーションのディプロイが完了するまで待ちます。このページの右側にあるディプロイアクティビティテーブルを調べ、アプリケーションが正しくディプロイされたことを確認します。アプリケーションは、左側の階層にも表示されます。

EJB の JNDI 名を指定したり、リソースアダプタの JNDI 名をマップしたりする必要はありません。これらは、EJB のディプロイ記述ファイルですでに指定されています。ディプロイ記述ファイル weblogic-ejb-jar.xml の該当する行は、次のとおりです。

```
<resource-description>
    <res-ref-name>CCIMFCobol_v1.0</res-ref-name>
        <jndi-name>eis/MFCobol_v1.0</jndi-name>
</resource-description>
...
<jndi-name>ejb/JMapServEJB</jndi-name>
```

このファイルを表示するには、上記の位置で MapDemo-WL.ear を検索し、JMapServ.jar を抽出します。次に JMapServ.jar から weblogic-ejb-jar.xml を抽出します。

WebLogic Server の停止と再起動

BEA WebLogic のマニュアルでは、サーバにディプロイしたモジュールを変更した場合に、サーバを停止してから再起動することを推奨しています。この操作手順は、次のとおりです。

1. WebLogic のサーバコンソールの左側のペインで [MicroFocus] を右クリックし、[サーバ > MFServer] の順に選択します。[サーバの起動と停止] をクリックすると、次のメッセージが表示されます。

   WebLogic のサーバコンソールは、サーバを再起動するまで使用できません。

2. 前回と同じ手順で WebLogic Server を再起動します。
3. サーバが再起動するまで待ちます。再起動後に、サーバコンソールの [更新] をクリックします。

注: マシンをリブートする場合は、WebLogic サーバを再起動する必要があります。WebLogic サーバは、[スタート] メニューから起動してください。コントロールパネルの「サービス」を使用して WebLogic サーバを起動したり、マシンのリブート時に WebLogic サーバを自動起動するようにコントロールパネルで設定することはできません。

WebSphere へのディプロイ

WebSphere は、管理コンソールに付属しており、EJB とリソースアダプタのディプロイに使用します。このコンソールでは、表示される手順とその詳細説明に従って、ディプロイ作業を進めることができます。コンソールの左側のペインにあるツリーを移動し、左側の項目をクリックして、必要なページを右側に表示できます。

WebSphere サーバの起動と管理コンソール

WebSphere サーバと管理コンソールを次のように起動します。

1. [スタート] メニューを使用するか、または次のようなコマンドを入力して WebSphere サーバを起動します。

   websphere_installation\bin\startServer

2. WebSphere サーバが起動するまで待ち、[スタート] メニューを使用するか、または、ブラウザを開いて URL「http://localhost:9090/admin」を指定し、WebSphere の管理コンソールを起動します。
3. ID としてユーザ名を入力します。

WebSphere へのリソースアダプタのディプロイ

リソースアダプタをディプロイするには、WebSphere の管理コンソールに移動し、次の手順を実行します。

1. 左側のペインで [リソース] を展開し、[リソースアダプタ] をクリックします。右側のページの下部で、Micro Focus のリソースアダプタがディプロイされていないことに注目します。
2. [Install RAR] をクリックし、画面の説明に従って操作します。次の値以外は、デフォルト値を使用します。

プロンプト	指定内容
リソースアダプタへのパス	NX\base\bin\mfcobol-notx.rar を指定します。 NX は、Net Express のインストールディレクトリです。
リソースアダプタの名前	「Micro Focus のリソースアダプタ」などのリソースアダプタ名を自由に指定します。

3. 新しいリソースアダプタが右側に表示されます。このリソースアダプタをクリックして属性を表示します。
4. リソースアダプタの接続ファクトリを追加します。末尾の「その他の属性」までスクロールして、[J2C 接続ファクトリ] をクリックします。[新規作成] をクリックし、画面の説明に従って操作します。
   - 「MF アダプタ接続」などの接続ファクトリ名を指定します。
   - エンタープライズサーバの JNDI 名 (eis/MFCobol_v1.0) を指定します。
5. リソースアダプタの新しい接続が右側に表示されていることを確認します。

WebSphere への EJB のディプロイ

生成した EJB をディプロイする前に、その EJB と他のアプリケーションファイルをエンタープライズアーカイブファイル (.ear) としてパッケージ化する必要があります。Net Express には、WebSphere 向けの MapDemo アプリケーションに使用するアプリケーションアーカイブファイル MapDemo-WS.ear が組み込まれています。

EJB をディプロイするには、次の手順を実行します。

1. 次のように、MapDemo アプリケーションに使用する組み込みの .ear ファイルに JMapServ.jar を追加します。
    1. NX\base\demo\MapDemo\Client\Java\lib ディレクトリ (NX は、Net Express のインストールディレクトリ) に移動し、MapDemo-WS.ear ファイルを同じディレクトリの MapDemo.ear にコピーします。

        通常、アーカイブファイル名はそのアーカイブファイル内にビルドされているので、アーカイブファイル名を変更すると、そのファイルが機能しません。その場合は、このファイルをコピーするときに、ファイルの元の名前 MapDemo.ear に変更します。

    2. MapDemo.ear と同じディレクトリに、NX\base\demo\MapDemo\MapDemo\repos\JMapServ.deploy ディレクトリから JMapServ.jar をコピーします。
    3. 次のコマンドを使用して、JMapServ.jar を MapDemo.ear に追加します。

        jar - uvf MapDemo.ear JMapServ.jar.

2. WebSphere の管理コンソールで、左側の「アプリケーション」の階層を展開します。
3. [新しいアプリケーションのインストール] をクリックし、画面の説明に従って操作します。次の値以外は、デフォルト値を使用します。

プロンプト	指定内容
ear ファイルへのパス	NX\base\demo\MapDemo\Client\Java\lib\MapDemo.ear を指定します。 NX は、Net Express のインストールディレクトリです。
JSP のプリコンパイル	チェックします。
EJB のディプロイ	チェックします。
EJB の JNDI	JNDI 名として「MyMapEJB」を入力します。

名

EJB 参照を Bean にマップ　JNDI 名として「MyMapEJB」を入力します。

リソース参照をリソースにマップ　JNDI 名として「MyMapEJB」を入力します。

モジュールをアプリケーションサーバにマップ　両方のモジュールをチェックします。

4. ディプロイが完了するまで待ち、画面のメッセージで、正しくディプロイされたことを確認します。 画面の説明に従って、新しい構成を保存します。
5. 左側のエンタープライズアプリケーションをクリックして、インストールしたアプリケーションを表示します。 MapDemo アプリケーションは、右側のページのように表示されます。
6. MapDemo アプリケーションを選択し、[起動] をクリックして、起動します。

WebSphere サーバの停止と再起動

IBM WebSphere のマニュアルでは、サーバにディプロイしたモジュールを変更した場合に、サーバを停止してから再起動することを推奨しています。

サーバを停止してから再起動するには、WebSphere の管理コンソールに移動し、次の手順を実行します。

1. 右側のペインの上部にある [保存] ボタンをクリックして、構成を保存します。

2. [スタート] メニューを使用するか、または、次のコマンドを入力して、WebSphere サーバを停止します。

   websphere_installation\bin\stopServer

3. 前回と同じ手順でサーバを再起動します。
4. サーバが再起動するまで待ちます。再起動後に、管理コンソールの [更新] をクリックします。

次に進む前に

サービスを使用するためのクライアントアプリケーションの作成方法については、『*クライアント作成の概要*』の章を参照してください。

作業を中断する場合は、Net Express、プロジェクト、または、リポジトリを閉じてかまいません。後の章に進んだときに、Net Express とプロジェクトを再度開いてください。

第 9 章 : クライアントの作成の概要

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

さまざまなクライアントの実行方法とサービスとのインターフェイス方法を説明したチュートリアルの章に進む前に、ここでは、サービス用クライアントの概要を説明します。

クライアントのタイプ

作成してディプロイしたサービスは、別途作成したクライアントで起動できます。EJB を作成した場合は、他の EJB、JSP などの J2EE コンポーネントから呼び出せます。また、J2EE 環境以外の管理されていない (非 J2EE) 接続からサービスを起動することもできます。COM インターフェイスを使用する場合は、通常の COM 機構と .NET (たとえば GUI クライアントには VB.NET、Web クライアントには ASP.NET を使用) で呼び出すことができます。Web サービスの場合は、Web サービスをリモートから起動できるすべてのクライアントで起動できます。

Web サービスは、WSDL (Web Services Description Language) の規則に従って記述します。Net Express で Web サービスを作成する場合は、Net Express によって WSDL ファイルが作成されます。クライアントアプリケーションを作成するユーザは、WSDL ファイルから作業すると、サービスの起動方法やサービスとの間で受け渡すデータを理解できます。たとえば、Java または .NET のワークベンチでこの WSDL ファイルを使用して Java クライアントまたは .NET クライアントを作成できます。

Interface Mapping Toolkit には、クライアントの生成機能が含まれています。この機能を使用すると、Web サービスに使用できる簡単な COBOL クライアントアプリケーションが生成されます。

このマニュアルのこの部分では、さまざまなタイプのクライアントの選択方法について説明します。各章では、組み込まれたクライアントアプリケーションを実行し、サービスとのインターフェイス方法について説明します。Web サービス用 COBOL クライアントに関する章では、作成済みのクライアントではなく、クライアントの生成機能を使用します。

実際のサービスに使用するクライアントを作成する場合は、このマニュアルのこの部分で使用するクライアントを参考にしてください。

次の作業

次のいずれかの章に進んでください。

- Web サービス用に生成した COBOL クライアント
- COM 用の .NET クライアント
- EJB 用の JSP クライアント

第 10 章 : Web サービス用に生成した COBOL クライアント

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

概要

ここでは、Net Express でクライアントの生成機能を使用して、特定の Web サービスのクライアントとして使用できる COBOL プロキシプログラムを生成します。 生成したプロキシは、同時に作成した COBOL サンプルアプリケーションや独自に作成したクライアントアプリケーションから呼び出せます。 このプロキシでは、呼び出し側アプリケーションからデータを受け取り、Web サービスを起動して、アプリケーションにデータを返します。

クライアントは Web 上で Web サービスと通信しますが、クライアントが Web 型のインターフェイスである必要はありません。 Net Express でプロキシとともに生成したクライアントアプリケーションは、ユーザインターフェイスに ACCEPT/DISPLAY 文を使用する簡単な文字モードのアプリケーションです。

『*サービスのディプロイ*』の章では、wmapserv という Web サービスをディプロイしました。 ここでは、wmapserv を使用するクライアントアプリケーションを作成して実行します。

準備

Web ブラウザとコンソールデーモンを最小化します。 Net Express の IDE が最小化されている場合は、元に戻します。

Net Express でプロジェクト Mapdemo と「サービスインターフェイス」ウィンドウが開いていない場合は、開きます。「サービスインターフェイス」ウィンドウを開くには、[ファイル > サービスインターフェイスを開く] をクリックします。Net Express\Base\Demo\Mapdemo から mapdemo.mpr を選択し、[開く] をクリックします。

『*サービスのディプロイ*』の章で説明するとおり、ESDEMO エンタープライズサーバが起動しており、wmapserv サービスがディプロイされていることを確認します。

クライアントの生成

Net Express で Web サービス用の COBOL クライアントを生成する方法は、2 通りあります。 一般的な方法は、Web サービスの WSDL ファイルからクライアントを直接生成する方法です。 この方法は、Interface Mapping Toolkit で作成した Web サービスに限らず、WSDL ファイルを使用するすべての Web サービスに適用できます。 そのため、公開されている Web サービスについて Web から WSDL ファイルを取得して、Net Express でクライアントを生成できます。

ただし、Interface Mapping Toolkit を使用してサービスを作成してディプロイした場合は、WSDL を参照しないで、サービスから直接クライアントを生成できます。

このチュートリアルでは、2 通りの方法を説明します。クライアントを 2 回作成しても問題はありません。このチュートリアルを 2 回実行して、2 通りの方法を試すこともできます。

マッピングからクライアントを生成する手順は、次のとおりです。

1. 「サービスインターフェイス」ウィンドウで wmapserv が選択されていることを確認し、[サービス > クライアントの生成 > マッピングを使用] をクリックします。

   または、wmapserv を右クリックして、[クライアントの生成] をクリックします。この場合は、WSDL を使用するための選択が表示されず、直接マッピングが使用されます。

   ダイアログボックスが開きます。ただし、このチュートリアルをすでに実行しており、wmapserv を作成している場合は、このダイアログボックスが表示されません。その場合は、手順 3 に進んでください。

2. ここでは、クライアントのコードを、現在開いているプロジェクトの一部にするので、このダイアログボックスで「現在の Net Express プロジェクトを使用する」が選択されていることを確認し、[OK] をクリックします。
3. プロキシファイル wmapserv-proxy.cbl、サンプルアプリケーションファイル wmapserv-app.cbl、および、コピーブックファイル wmapserv-copy.cpy が生成され、関連付けられたビルドオブジェクトとともにプロジェクトに追加されています

   クライアントの生成が終了したことを表すメッセージに対して [OK] をクリックします。

WSDL ファイルからクライアントを作成する手順は、次のとおりです。

1. [サービス > クライアントの生成 > WSDL の使用] をクリックします。

   ウィザードが開きます。

2. [参照] をクリックし、Mapdemo\Mapdemo\REPOS\wmapserv.deploy フォルダ内の wmapserv.wsdl を選択して、[開く] をクリックします。次に [次へ] をクリックします。

   このファイルが、wmapserv の作成時にともに作成した WSDL ファイルです。この機能を使用すると、Net Express で作成した Web サービスに限らず、どの Web サービスに対して、この WSDL ファイルを使用してクライアントを作成できます。

3. 「現在の Net Express プロジェクトを使用する」が選択されていることを確認して、[次へ] をクリックします。
4. [終了] をクリックします。
5. マッピングを使用した場合と同じファイルが生成されます。

   クライアントの生成が終了したことを表すメッセージに対して [OK] をクリックします。

クライアントのビルドと実行

1. [リビルド]  をクリックします。
2. 「プロジェクト」ウィンドウで wmapserv-app.int を選択し、ポップアップメニューの [実行] をクリックします。
3. Net Express で生成するクライアントアプリケーションでは、各操作に数字コードを使用します。

   3 (読み取り操作) と入力し、Enter キーを押します。

   この読み取り用コードは、book.cbl とそのサービスで使用するコードとは異なります。クライアントは、サービスの内部的な処理を認識しません。

4. 『*サービスの作成*』の章の読み取り操作では、タイトル、作者、在庫番号のどれか 1 つしか必要でなかったことを思い出してください。

   プロンプトに対して 1111 (在庫番号) を入力し、タイトルと作者のフィールドを空白にします。 それぞれに Enter キーを押します。

   しばらくすると、要求したレコードの各フィールドの名前と内容がクライアントに表示されます。 その場合は、Web サービスの呼び出しに成功しています。ファイル内のレコードが検索され、データが返されています。

   このチュートリアルを 2 回実行して (マッピングからと WSDL ファイルから) クライアントを生成すると、生成されたクライアントの動作が異なることがわかります。 マッピングからクライアントを生成すると、クライアントの PICTURE 文字列が元のレガシープログラムから直接作成されます。 WDSL ファイルからクライアントを生成すると、このように直接リンクせず、クライアントの PICTURE 文字列が WSDL のデータ型に基づいて作成されます。

   他の操作も試してみます。『*サービスの作成*』の章で、組み込みのファイルにレコード 1111 と 2222 が含まれていたことを思い出してください。このデータ例を使用した場合は、番号 4444 も追加してみます。

5. 作業の終了後、「アプリケーション出力」ウィンドウを閉じます。

次に進む前に

このサービスのデバッグ方法については、『*サービスのデバッグ*』の章を参照してください。

作業を中断する場合は、Net Express やプロジェクトを閉じてかまいません。

第 11 章 : COM 用の .NET クライアント

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

このセッションを実行するには、Microsoft Visual Studio .NET をインストールする必要があります。ここでは、ユーザが Microsoft Visual Studio .NET に精通していることを前提に説明します。

概要

ここでは、組み込みの C# クライアントを使用して、『*サービスのディプロイ*』の章でディプロイした COM オブジェクトを呼び出します。 ここでは、Visual Studio を使用して、COM オブジェクトに対するクライアントのインターフェイス方法を説明します。

COM オブジェクトを Visual Studio にインポートします。 Visual Studio では、COM オブジェクトを起動するプロキシプログラムを生成します。 このプロキシをクライアントアプリケーションで呼び出します。

準備

『*サービスのディプロイ*』の章で説明するとおり、cmapserv サービスがディプロイされ、Windows 上で登録されていることを確認します。

Visual Studio を起動し、組み込みのサンプルプログラム Net Express\Base\Demo\Mapdemo\Client\Microsoft.NET\COMClient\CMapClient.sln のソースをロードします。 表示される cmapclient.cs ファイルは、クライアントアプリケーションのソースコードです。

クライアントのソース

ここでは、クライアントから COM オブジェクトを起動する方法について説明します。 組み込みのクライアントソースコードはすでに完成しているので、ここでソースコードを作成する必要はありません。クライアントアプリケーションを独自に作成する場合には、次の作業を実行してください。

1. [プロジェクト > 参照の追加] をクリックして、COM オブジェクトを Visual Studio にインポートします。
2. [COM] タブをクリックします。
3. リストボックスから cmapserv を選択するか、または、Net Express\Base\Demo\Mapdemo\REPOS\cmapserv.deploy\cmapserv.dll を参照します。
4. [OK] をクリックします。

COM オブジェクトを再ディプロイする場合は、新しい登録キーが選択されるように、COM オブジェクトを Visual Studio からいったん削除して再インポートします。

5. cmapclient.cs をダブルクリックしてアプリケーションのソースコードを表示します。

   組み込みのアプリケーションでは、次の行によって COM オブジェクトを起動します。

   ```
   cmapserv comproxy = new comproxy();
   ```

   また、次の行によって次の操作を呼び出します。

   ```
   comproxy.Next(out title,
                 out type,
                 out author,
                 ref stockno,
                 out retail,
                 out onhand,
                 out sold,
                 out fileStatus
                 );
   ```

クライアントの実行

クライアントを実行するには、次の操作を実行します。

1. Visual Studio で [デバッグ > 実行] をクリックします。
2. 1111 (在庫番号) と入力し、他のフィールドを空白にしたまま、[読み取り] をクリックします。

   しばらくすると、要求したレコードのデータがクライアントに表示されます。その場合は、Web サービスの呼び出しに成功しています。ファイル内のレコードが検索され、データが返されています。

   他の操作も試してみます。『*サービスの作成*』の章で、組み込みのファイルにレコード 1111 と 2222 が含まれていたことを思い出してください。このデータ例を使用した場合は、番号 4444 も追加してみます。

3. 作業の終了後、  ボタンをクリックして、アプリケーションを終了します。

次に進む前に

このサービスのデバッグ方法については、『*サービスのデバッグ*』の章を参照してください。

第 12 章 : EJB 用の JSP クライアント

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

概要

ここでは、組み込みの JSP クライアントを使用して、『*サービスのディプロイ*』の章でディプロイした EJB を起動します。

JSP は、エンドユーザから要求を受け付け、生成した EJB にその要求を渡します。EJB は、エンタープライズサーバ ESDEMO に、この要求を渡します。このエンタープライズサーバでは、元の COBOL プログラムが実行され、要求が処理されて、JSP に結果が返されます。

クライアントのソース

クライアント JSP とサーブレットのソースコードを表示するには、NX\base\demo\MapDemo\Client\Java\src\com\mypackage\JMapServ (NX は Net Express のインストールディレクトリ) を開きます。次のファイルが表示されます。

- MapDemoServlet.java - このファイルは、servlet です。JSP からの要求を処理し、状態を設定しない EJB のインスタンスを起動して、応答を JSP に返します。
- MapservJspBean.java - このファイルでは、getter メソッドと setter メソッドを使用して、servlet と JSP の間で情報を移動します。
- SessionMonitor.java - 状態を設定するセッション Bean の EJB インスタンスを格納するヘルパークラスです。
- MapDemo.jsp - JSP です。エンドユーザと servlet の間でデータを送受信します。

servlet から EJB を起動する方法を調べるには、Java 開発ツールやテキストエディタで MapDemoServlet.java ファイルを開き、次の行を確認してください。

```
InitialContext initCtx = new InitialContext();
Object objRef = initCtx.lookup(
                "java:comp/env/ejb/JMapServEJB"
                );
mHome = (JMapServHome)
        javax.rmi.PortableRemoteObject.narrow(
        objRef,
        com.mypackage.JMapServ.JMapServHome.class
        );
```

これらの行によって、EJB が最初に必要になったときに JNDI 内で EJB が検索されます。

- `mr = mHome.create();`
- `SessionMonitor.setClientRequest(session, mr);`

  これらの行によって、EJB のインスタンスが作成され、要求を処理するためのセッションが開始されます。

- `mr = SessionMonitor.getClientRequest(session);`

  この行によって、次のオカレンスで EJB が取得されます。

準備

『*サービスのディプロイ*』の章で説明するとおり、ESDEMO エンタープライズサーバが起動しており、JMapServ サービスがディプロイされていることを確認します。

前のチュートリアル『*サービスのディプロイ*』では、アプリケーションサーバに MapDemo.ear をディプロイしました。この .ear ファイルには、JSP クライアントが含まれています (Web アーカイブファイル MapDemo.war にパッケージ化されています)。そのため、JSP クライアントもアプリケーションサーバにディプロイされており、すぐに使用できます。

JSP クライアントの実行

組み込みの JSP を実行するには、次の操作を実行します。

1. 前回と同様に [スタート] メニューから WebLogic Server または WebSphere Server を起動します。WebLogic Server Console または WebSphere Administrative Console を起動する必要はありません。
2. 両方のサーバが正しく起動した後、別のブラウザを開きます。
3. WebLogic については、URL「http://localhost:7001/MapDemo/MapDemo.jsp」を入力します。

   WebSphere については、URL「http://localhost:8000/MapDemo/MapDemo.jsp」を入力します。

4. Map Demo の Web ページで book データベースを表示します。

   1111 (在庫番号) と入力し、[読み取り] を選択して、[送信] をクリックします。

   しばらくすると、要求したレコードの各フィールドの名前と内容がクライアントに表示されます。これらの内容は、EJB インターフェイスからリソースアダプタ経由でエンタープライズサーバに渡された要求を表しています。また、要求を処理し、ファイル内のレコードを検索して、応答を JSP に返す COBOL サービス BOOK.Read も表しています。

他の操作も試してみます。『*サービスの作成*』の章で、組み込みのファイルにレコード 1111 と 2222 が含まれていたことを思い出してください。このデータ例を使用した場合は、番号 4444 も追加してみます。

次に進む前に

一連のチュートリアルは、ここまでです。別のチュートリアルに進むには、『*チュートリアルの概要*』の章にあるチュートリアルマップを参照してください。

第 13 章 : サービスのデバッグ

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

概要

作成したサービスのタイプやそのサービスを起動するクライアントに関係なく、Net Express のデバッグ機能を使用して、 COBOL プログラムをサービスとして実行しながらデバッグできます。

その場合は、CBL_DEBUGBREAK を使用します。CBL_DEBUGBREAK は、通常のように IDE メニューから選択して起動することができない状況でデバッグを選択するために、プログラムを一時停止させる呼び出しです。

準備

Net Express とプロジェクト Mapdemo が開いていない場合は、開きます。

『*サービスのディプロイ*』の章で説明するとおり、ESDEMO エンタープライズサーバが起動していることを確認します。

サービスの変更と再ディプロイ

1. プロジェクトツリービューで、編集する book.cbl をダブルクリックします。
2. 手続き部の先頭で main 節見出しの後の行に次の文を挿入します。

   ```
   call "cbl_debugbreak"
   ```

3. book.cbl の「編集」ウィンドウを閉じ、  をクリックします。
4. 『*サービスのディプロイ*』の章にある『*削除と再ディプロイ*』で説明するとおり、エンタープライズサーバからサービスを削除して再ディプロイします。
5. 次の生成した Web サービスクライアントを使用しない場合は、Net Express を閉じてかまいません。

サービスのデバッグ

クライアントをデバッグする場合も、また、単に実行する場合も、サービスをデバッグできます。ここでは、サービスとクライアントの両方をデバッグします。

1. このマニュアルの一連のチュートリアルで使用したクライアントアプリケーションのどれかを実行します。

たとえば、Web サービス用に作成した COBOL クライアントを使用する場合は、プロジェクトツリービューで wmapserv-app.int を選択して右クリックし、[実行] をクリックします (ステップ実行する場合は、[アニメート] をクリックします)。

2. 必要なデータを入力します。 たとえば、読み取り操作を選択し、タイトルと作者のフィールドを空白にして、在庫番号として 1111 を入力します。
3. call "cbl_debugbreak" の呼び出しが検出されたことを表すメッセージに対して [はい] をクリックします。

   IDE が開き、すぐにデバッグできる book.cbl が表示されます。 生成した Web サービスクライアントを使用している場合は、IDE のコピーが 2 つ開きます。

4. これで、book.cbl をデバッグできます。 最後までまとめて実行します。
5. デバッグが exit program 文に到達し、行がハイライトされていない場合は、サービスが終了し、クライアントに戻っています。

   book.cbl が表示されている IDE を閉じます。 アニメートを停止するかどうかを確認するメッセージに対して [はい] をクリックします。次に、待機を続けるかどうかを確認するメッセージ (数秒後に表示されます) に対して [いいえ] をクリックします。

   タイムアウトを表すエラー 0015 が Web サービスから返されます。このエラーは、デバッグに時間がかかりすぎた場合に返されます。 この問題が発生した場合は、タイムアウト間隔を変更します。「Enterprise Server Administration」ホームページの左側にあるリストから [オプション] をクリックし、「クライアントプログラムのタイムアウト」フィールドを再設定して、[適用] をクリックします。

6. クライアントの実行を完了します。 たとえば、生成した Web サービスクライアントを使用している場合は、ステップ実行を継続するか、または  をクリックします。

デバッグ後

サービスをデバッグする必要がなくなった場合は、book.cbl を編集して cbl_debugbreak の呼び出しを削除し、リビルドします。 前と同様に、エンタープライズサーバからサービスを削除して再ディプロイします。

デバッグの繰り返し

book.cbl に CBL_DEBUGBREAK 呼び出しを指定した場合でも、呼び出しが検出されたことを表すメッセージに対して [いいえ] をクリックすると、デバッグしません。 ただし、この操作を実行すると、次回以降にアプリケーションを実行したときに CBL_DEBUGBREAK が無視されます。 そのため、エンタープライズサーバをいったん停止させてから再起動して、CBL_DEBUGBREAK を機能させる必要があります。 この操作方法は、次のとおりです。

1. 「Enterprise Server Administration」ホームページで「現ステータス」の下にある [詳細] をクリックします。
2. [サーバの停止] をクリックします。

   ホームページに戻り、コンソールウィンドウを閉じます。

3. 数秒待機してから、このページの [画面更新] ボタン (ブラウザの [更新] ボタンではありません) をクリックして、「現ステータス」が「停止処理中」から「停止済み」に更新されていることを確認します。
4. 「現ステータス」の下にある [開始] をクリックして、エンタープライズサーバを再起動します。

次に進む前に

一連のチュートリアルは、ここまでです。 別のチュートリアルに進むには、『*チュートリアルの概要*』の章にあるチュートリアルマップを参照してください。

第 14 章 : XML 対応 COBOL の概要

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

ここでは、Net Express での XML 対応 COBOL の処理方法について簡単に説明します。XML 対応 COBOL の処理方法を説明するチュートリアルに進む前に、この説明を参照すると、チュートリアルの処理を早く理解できます。

Net Express と XML

Net Express には、COBOL プログラムにコードとして入力できる XML 構文の拡張機能が多数用意されています。これらの機能は、手入力することも、CBL2XML ウィザードやコマンド行ユーティリティを使用して自動的にコード化することもできます。 構文の拡張機能を使用すると、COBOL プログラムで XML インスタンスドキュメントに入出力できます。索引ファイルなどの従来のファイル方法を使用する必要はありません。

XML のチュートリアルで使用する XML 対応 COBOL アプリケーションは、次のように処理されます。

- XML 構文の拡張機能を使用して COBOL コードを変更します。

  これらのチュートリアルでは、COBOL SELECT 文を変更して XML インスタンスドキュメントと通信し、XML 構文の拡張機能を使用して FD 文を XD 文 (ファイルレコードを含む) に置換します。 これらの構文の拡張機能は、SELECT 文で指定した XML インスタンスドキュメントと通信できるように設計されています。 チュートリアルのサンプルプログラムでも、COBOL 動詞を必要に応じて変更し、XML と通信します。

- XML 構文を使用して COBOL データレコードを生成します。

  これらのチュートリアルでは、CBL2XML ウィザードを使用してこの操作を実行します。

- Net Express のコードをコンパイルして XML プリプロセッサを起動します。
- アプリケーションをディプロイします。

COBOL の変更方法は、XML ドキュメントの構造によって異なり、XML スキーマによって規定されます。

CBL2XML ウィザードとコマンド行ユーティリティでは、処理を自動化できます。 これらの機能を使用して、COBOL レコードを指定し、このレコードに基づいて XML 対応のコピーファイルとスキーマを自動的に生成できます。

Net Express の XML サポートについては、『*分散コンピューティング*』マニュアルにある『*COBOL と XML の併用*』の章を参照してください。

次の作業

XML に関する最初のチュートリアル『*XML への移行*』に進んでください。

第 15 章 : XML への移行

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

概要

このセッションでは、XML 構文の拡張機能と CBL2XML ウィザードを使用して、COBOL の索引ファイルを XML インスタンスドキュメントに移行します。

ここでは、索引ファイルからの読み取りと削除、および、索引ファイルへの書き込みを実行する COBOL プログラムを紹介します。 また、このセッションと次のセッションでは、XML インスタンスドキュメントからの読み取りと削除、および、XML インスタンスドキュメントへの書き込みを実行するプログラムを変更します。

サンプルプログラム

このセッションでは、組み込みの 2 つのサンプルプログラム (customerdb.cbl と migrate.cbl) を使用します。

customerdb.cbl

顧客データベースの索引ファイル (customers.dat) からの読み取りと削除、および、この索引ファイルへの書き込みを実行する簡単なレガシープログラムです。 ファイル記述レコードを格納した customer.cpy コピーファイルが含まれています。 XML と通信するために変更するのは、索引ファイルではなく、このレコードです。

migrate.cbl

また、migrate.cbl というプログラムも使用します。このプログラムは、customers.dat ファイルを受け付けて XML インスタンスドキュメント customers.xml に移行できます。

このプログラムを使用する理由は、次のとおりです。

- 既存のレコードを変更して、XML 対応プログラムを作成する手順を紹介するため
- 索引ファイルから XML インスタンスドキュメントを生成する機構があるため (次のセッションで XML インスタンスドキュメントを使用)

これらのプログラムの COBOL ソースは、Net Express\Base\Demo\XMLdemos\migrate に格納されており、Net Express またはテキストエディタで表示することができます。

準備

Net Express を閉じている場合は、再度開きます。「プロジェクト」ウィンドウなどのウィンドウが開いている場合は、閉じます。

プロジェクトを作成する手間を省くために、既存のプロジェクトを使用します。[ファイル > 開く] をクリックして、Net Express\Base\Demo\XMLdemos\migrate\migrate.app を開きます。customerdb.cbl と migrate.cbl の両方が表示されます。

レガシーアプリケーションのビルドと実行

customerdb レガシーアプリケーションを理解するためにこの作業を実行します。この作業は、省いてもかまいません。

1. 「プロジェクト」ウィンドウで customerdb.cbl を右クリックします。
2. [コンパイル] をクリックします。
3. [実行]  をクリックします。プロジェクトをリビルドするかどうかを確認するためのダイアログが表示されます。この段階ではプロジェクトをリビルドしないので、[いいえ] をクリックします。選択したプログラムのみがリビルドされます。
4. DEBUG\customerdb が指定されていることを確認し、[OK] をクリックします。
5. 「選択」フィールドに 2 を入力します。
6. 「顧客 ID の入力」フィールドに 00001 を入力し、Enter キーを 2 回押します。データベースのレコードが表示されます。
7. Enter キーを押します。
8. 「選択」フィールドに 1 を入力します。
9. Enter キーを押してフィールド間を移動し、別の顧客に関する詳細を入力します。たとえば次のとおりです。

顧客 ID	00005
顧客名	James Monroe
住所 1	Ash Lawn-Highland
住所 2	1000 James Monroe Parkway
市町村名	Charlottesville
都道府県名	VA
郵便番号	22902
電話番号	434-293-9539

10. Enter キーを押します。
11. Y を入力して、レコードを追加します。
12. 「選択」フィールドに 9 を入力してアプリケーションを終了します。[閉じる] ×ボタンをクリックして「アプリケーション出力」ウィンドウを閉じます。

アプリケーションの概要は、ここまでです。

XML 対応のコピーファイルとスキーマの生成

ここでは、CBL2XML ウィザードを使用して、既存のコピーファイル customer.cpy を XML に対応させます。次では、生成されたコピーファイルを使用して、索引ファイルを XML インスタンスドキュメントに移行します。

元のコピーファイルの表示

customer.cpy コピーファイルに含まれるファイルレコードを XML 対応のコピーファイルに変換する前に、元のコピーファイルを表示します。

1. 「プロジェクト」ペインで customerdb.cbl の次の ⊞(展開) をクリックします。customer.cpy コピーファイルが表示されます。
2. customer.cpy をダブルクリックします。

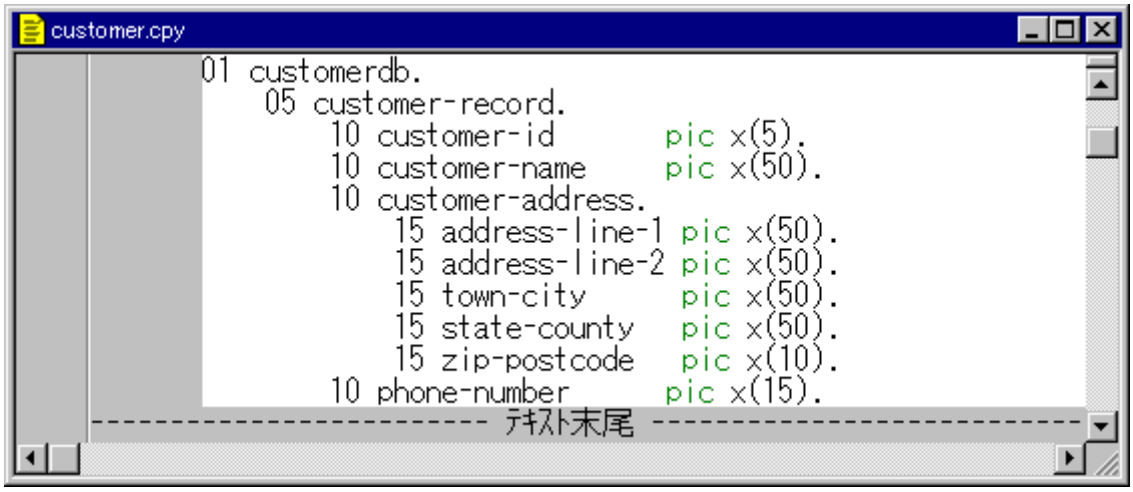


図 15-1: 元の customer.cpy コピーファイル

このコピーファイルには、索引を設定したデータベースにアクセスするためのファイル定義が記述されています。

[閉じる] ×ボタンをクリックして customer.cpy を閉じます。

ファイルレコードの XML 変換

索引ファイルを XML に移行するための第 1 段階では、customer.cpy コピーファイルに含まれるレコードを XML に変換します。

1. [ツール > CBL2XML ウィザード] をクリックします。
2. [COBOL から XML 構文を使用した COBOL に変換する] をクリックして、[次へ] をクリックします。

3. [参照] をクリックし、Net Express\Base\Demo\XMLdemos\migrate フォルダに移動します。customer.cpy をクリックして、[開く] をクリックします。 この操作によって、「XML スキーマに変換する COBOL プログラムまたはコピーファイルの指定」フィールドに入力されます。 [次へ] をクリックします。
4. 「既存の COBOL データ名の先頭に追加するプリフィックスの指定」フィールドに xml- と入力し、[次へ] をクリックします。

   生成された XML 対応のコピーファイルを migrate.cbl 内に指定するので、プリフィックスが必要です。migrate.cbl 内には、customer.cpy ファイルも指定します。 各データ名の先頭にプリフィックスを指定すると、データ名が競合しません。

5. 「生成される XML 拡張構文 COBOL が出力されるコピーファイルを指定」フィールドの次にある [参照] ボタンをクリックし、Net Express\Base\Demo\XMLdemos\migrate フォルダに移動します。「ファイル名」フィールドに customer-xml-migrate.cpy を入力します。 [開く] をクリックします。
6. 「XML スキーマへの出力ファイルの指定」フィールドの次にある [参照] ボタンをクリックし、Net Express\Base\Demo\XMLdemos\migrate フォルダに移動します。「ファイル名」フィールドに customer.xsd を入力します。 [開く] をクリックします。
7. [次へ] をクリックし、さらに [終了] をクリックします。

CBL2XML によって、プロジェクトと同じディレクトリにコピーファイル customer-xml-migrate.cpy と XML スキーマ customer.xsd が生成されます。

注：CBL2XML では、元のコピーファイルは変更されません。元のコピーファイルが完全な状態で残ったまま、新しいファイルが生成されます。

生成されたコピーファイルの表示

生成された customer-xml-migrate.cpy コピーファイルは、移行プログラムに依存します。ただし、migrate.cbl との依存関係を更新するまでは、「プロジェクト」ウィンドウには表示されません。依存関係を更新するには、migrate.cbl をクリックし、[プロジェクト > すべての従属関係の更新] をクリックします。 コピーファイルを表示するには、customer-xml-migrate.cpy をダブルクリックします。

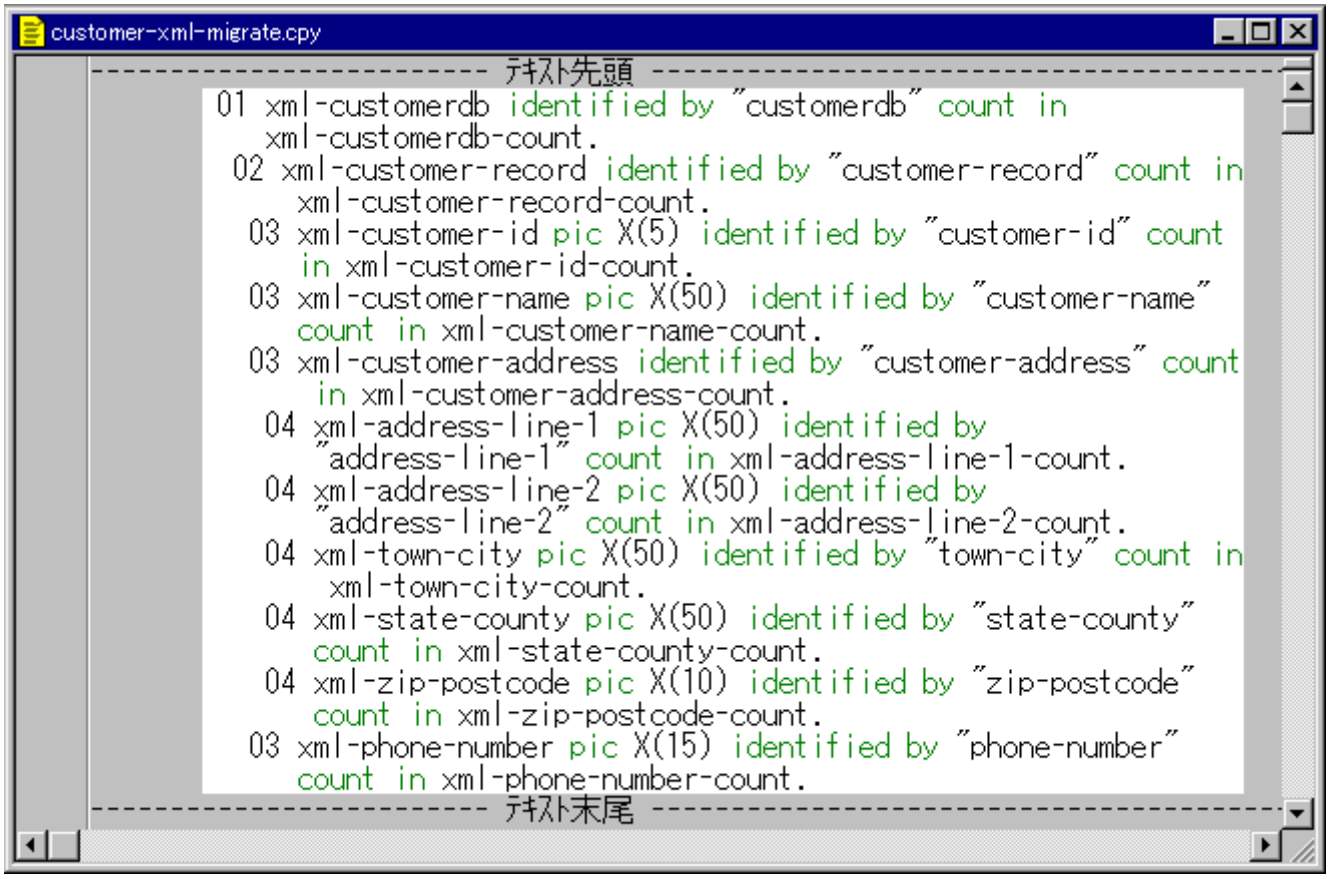


図 15-2: customer-xml-migrate.cpy コピーファイルの作成

ここで使用するすべてのデータ名は、customer.cpy 内のデータ名と同じです。 ただし、customer-xml-migrate.cpy 内では、このファイルの変換時に指定したとおり、各データ名の先頭にプリフィックス「xml-」が追加されています。 各データ名には、XML 構文の拡張機能も生成されています。 レコード構造体は、元のファイルと同じです。

[閉じる] ボタンをクリックして、customer-xml-migrate.cpy を閉じます。

生成されたスキーマの表示

この処理では、XML スキーマ customer.xsd も生成します。このスキーマによって、XML 対応の COBOL レコードとインターフェイスする XML インスタンスドキュメントの構造を規定します。スキーマは、プロジェクトと同じディレクトリに格納されますが、「プロジェクト」ウィンドウには表示されません。 スキーマは、次のようなファイルです。

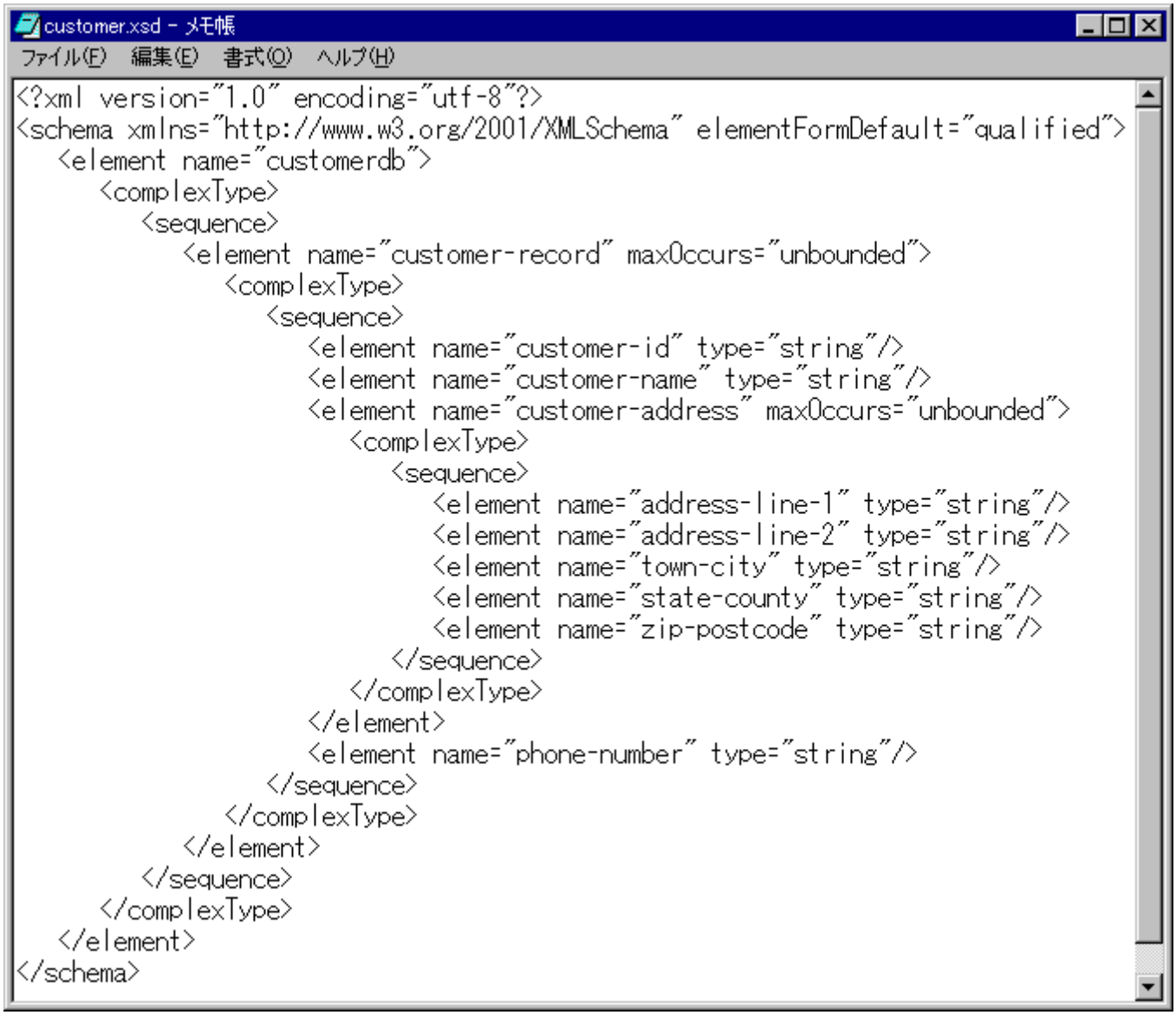


図 15-3: 生成されたスキーマ customer.xsd

これらの XML 要素から、このスキーマが customer.cpy に含まれる COBOL レコードに基づいていることがわかります。migrate.cbl プログラムでは、このスキーマに準拠した XML インスタンスドキュメントが生成されます。

migrate.cbl

組み込まれた migrate.cbl プログラムは、customer.cpy 内の元のレコードと customer-xml-migrate.cpy 内の新しいレコードを使用して、customers.dat 内のデータを customers.xml という XML インスタンスドキュメントに移行します。

移行プログラムの表示

移行プログラムには、COBOL で XML インスタンスドキュメントを読み書きするための特殊な XML 構文が含まれています。次の図は、プログラムの一部を表しています。この部分では、SELECT 文と XD 文が変更されています。これらの文の詳細については、『*分散コンピューティング*』マニュアルにある『*XML 構文の拡張機能*』の章を参照してください。

```
 $set p(prexml) endp
 identification division.
 program-id. Migrate.

*****************************************************************
*Assign for existing COBOL indexed file.
*****************************************************************
 select customer-file assign to "customers.dat"
                      organization is indexed
                      record key is customer-id
                      alternate key is customer-name
                      access mode is dynamic.

*****************************************************************
*Assign for new XML file "customer.xml".  This is the target for
*the migration to XML.
*****************************************************************
 select xml-customer-file assign to "customers.xml"
                          organization is xml
                          document-type is "customer.xsd"
                          file status is file-status-spec.

*****************************************************************
*File definition for the existing customer database
*****************************************************************
 fd customer-file.
 copy "customer.cpy".

*****************************************************************
*File definition for the new XML customer database.  Note the
*usage of "xd" to denote an XML format.
*****************************************************************
 xd xml-customer-file.
 copy "customer-xml-migrate.cpy".
```

図 15-4: 移行プログラムの一部

migrate.cbl プログラムでは、レコードのコピー (索引ファイル customers.dat を記述する元のレコード) と新しく生成されたレコード (XML インスタンスドキュメント customers.xml を記述するレコード) の両方を使用します。

migrate.cbl のビルドと実行

migrate.cbl は、一度に実行できるバッチプログラムなので、プログラムの実行中に対話する必要はありません。 このプログラムによって XML インスタンスドキュメントが生成されますが、customerdb.cbl レガシープログラムを XML に完全に対応させる必要があります。

1. 「プロジェクト」ウィンドウで migrate.cbl をクリックします。
2. [リビルド] をクリックします。
3. [実行] をクリックし、DEBUG\migrate が指定されていることを確認して、[OK] をクリックします。

migrate.cbl プログラムによって customers.xml ファイル (XML インスタンスドキュメント) が生成されます。

customers.xml の表示

次のコードは、customers.xml の一部です。 最後の 2 つのレコードは、customers.dat から取得したレコードです。 最後のレコードは、このセッションで、すでに追加したレコードです。

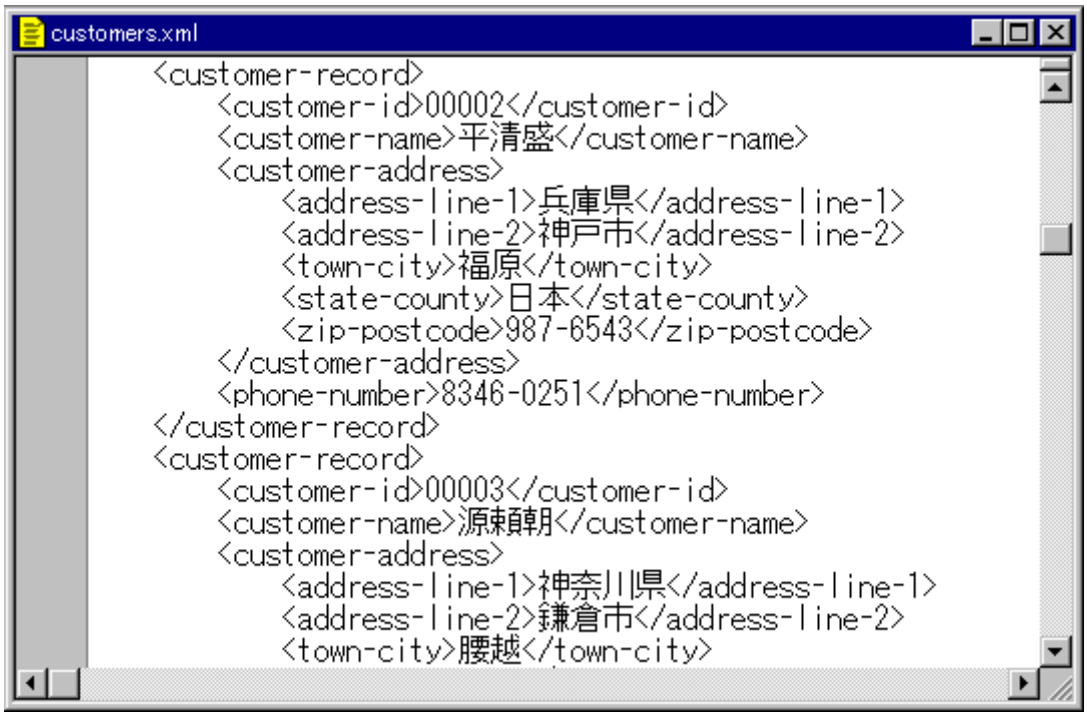


図 15-5: 生成された customers.xml ファイル

Net Express で customers.xml 全体を表示するには、次の操作を実行します。

1. [ファイル > 開く] をクリックします。
2. 「ファイルの種類」フィールドでスクロールして XML ファイル (*.xml) をクリックします。
3. このファイルリストで customers をダブルクリックします。

次に進む前に

生成された XML インスタンスドキュメントとスキーマを使用して customerdb.cbl を XML に対応させる方法については、『*レガシープログラムの XML 化*』の章を参照してください。

作業を中断する場合は、Net Express やプロジェクトを閉じてかまいません。次の章を参照するときに再度 Net Express を開いてください。

第 16 章 : レガシープログラムの XML 化

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

概要

前の章『*XML への移行*』では、レガシーアプリケーションを XML に対応させるための XML インスタンスドキュメントとスキーマを生成しました。

ここでは、前のセッションと同じレガシーアプリケーションを変更し、索引ファイルを使用せずに、XML インスタンスドキュメントからの読み取りと削除、および、このドキュメントへの書き込みをできるようにします。

このセッションでは、CBL2XML ウィザードと XML 構文の拡張機能を使用して、元の customerdb.cbl レガシープログラムの XML バージョンを作成します。

サンプルプログラム

前のセッションと同様に、サンプルプログラム (customerdb.cbl) を使用します。このプログラムは、索引ファイルを使用する簡単なレガシープログラムです。

また、customerdbxml.cbl サンプルプログラムも使用します。 このプログラムをこの状態のままでコンパイルまたは実行することはありません。 このプログラムの SELECT 文と FD を変更し、前のセッションで作成した XML インスタンスドキュメント (customers.xml) からの読み取りと削除、および、このドキュメントへの書き込みをこのプログラムで実行できるようにします。このプログラムの残りの部分には、すでに必要な XML 構文が含まれています。これらの構文についてもこのセッションで説明します。

準備

Net Express を閉じている場合は、再度開きます。「プロジェクト」ウィンドウやテキストウィンドウが開いている場合は、閉じます。

プロジェクトを作成する手間を省くために、既存のプロジェクトを使用します。[ファイル > 開く] をクリックして、Net Express\Base\Demo\XMLdemos\enableXML\enableXML.app を開きます。customerdb.cbl と customerdbxml.cbl が表示されます。

XML 対応のコピーファイルとスキーマの生成

前のセッションでは、各データ名にプリフィックスを設定し、XML 対応のコピーファイルとスキーマを作成しました。 プログラムで新しいコピーファイルと元のコピーファイルの両方をインク

ルードファイルとして使用するので、この操作が必要でした。プリフィックスを指定しないと、データ名が競合するためです。

このセッションでは、元のコピーファイルと同じデータ名を使用する XML 対応のコピーファイルを作成します。customerdbxml.cbl プログラムでは、元のコピーファイルと XML 対応のコピーファイルを置換するので、プリフィックスを指定する必要がありません。

この処理で生成するスキーマは、前のセッションで作成したスキーマとまったく同じです。

1. [ツール > CBL2XML ウィザード] をクリックします。
2. [COBOL から XML 構文を使用した COBOL に変換する] をクリックして、[次へ] をクリックします。
3. [参照] をクリックし、Net Express\Base\Demo\XMLdemos\enableXML フォルダに移動します。customer.cpy をクリックして、[開く] をクリックします。この操作によって、「XML スキーマに変換する COBOL プログラムまたはコピーファイルの指定」フィールドに入力されます。[次へ] をクリックします。
4. このページでは何も指定しないので、[次へ] をクリックして、次に進みます。
5. 「XML 構文の拡張機能を使用して生成された COBOL の出力を受け取るコピーファイルの指定」フィールドの次にある [参照] ボタンをクリックし、C:\Program Files\Micro Focus\Net Express 4.0\Base\DEMO\XMLDEMOS\EnableXML フォルダに移動します。「ファイル名」フィールドに customer-xml.cpy を入力します。[開く] をクリックします。
6. 「XML スキーマへの出力ファイルの指定」フィールドの次にある [参照] ボタンをクリックし、C:\Program Files\Micro Focus\Net Express 4.0\Base\DEMO\XMLDEMOS\EnableXML フォルダに移動します。「ファイル名」フィールドに customer.xsd を入力します。[開く] をクリックします。
7. [次へ] をクリックし、さらに [終了] をクリックします。

生成されたコピーファイルの表示

生成されたコピーファイルは、後でこのプロジェクトに追加しますが、このプロジェクトの一部ではありません。ファイルを表示するには、次の操作を実行します。

1. [ファイル > 開く] をクリックします。
2. base\demo\XMLdemos\enableXML フォルダに移動します。
3. 「ファイルの種類」フィールドのドロップダウンメニューで共通ファイル (*.cbl, *.cpy, *.cob, *.app) をクリックします。
4. customer-xml をダブルクリックします。

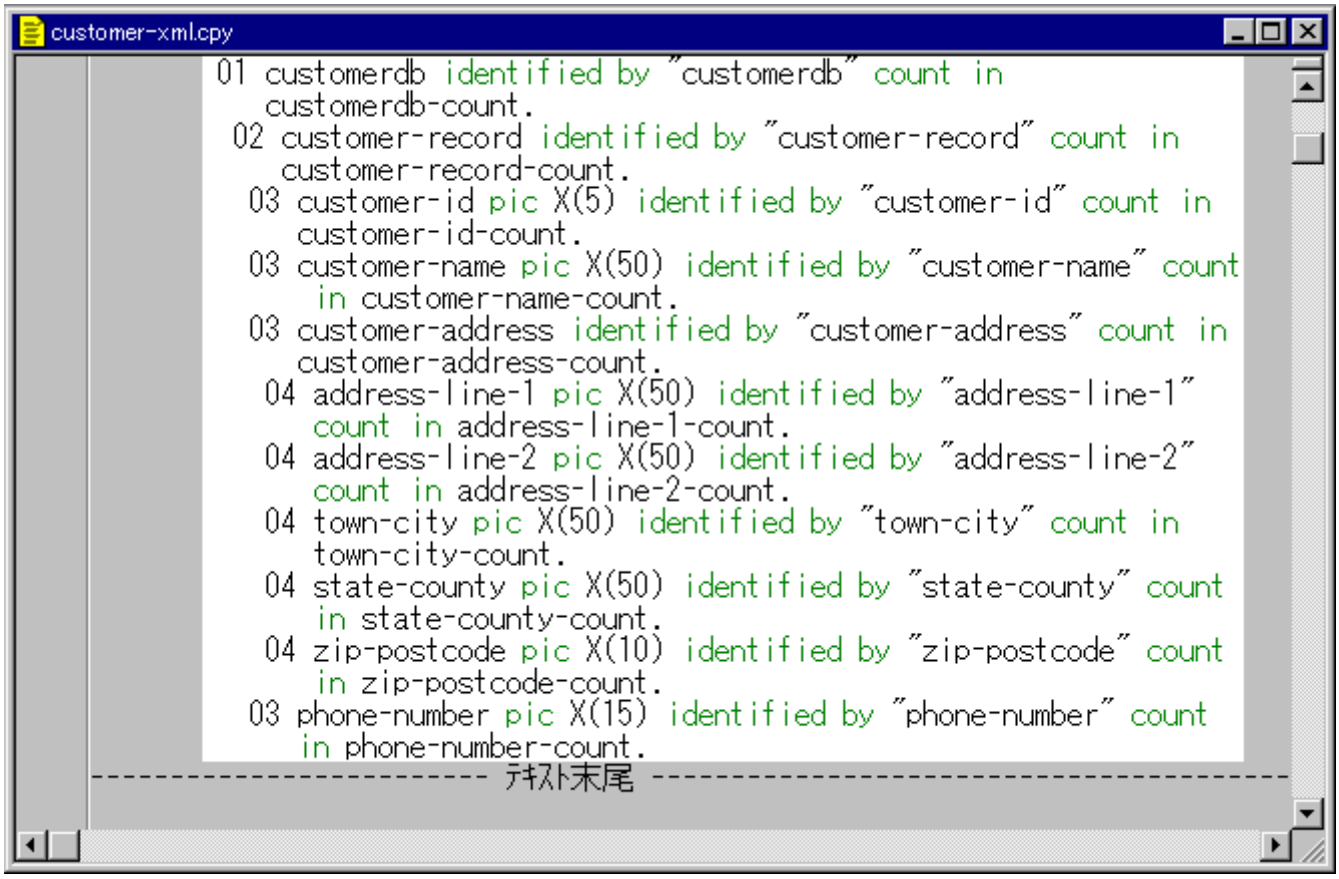


図 16-1: CBL2XML で生成された customer-xml.cpy

この場合は、データ名は変更されていません。CBL2XML では、XML 構文の拡張機能によって各レコードが変更され、プログラムが customers.xml ファイルと対話できるようになります。

[閉じる] ボタンをクリックして、customer-xml.cpy ファイルを閉じます。

customerdbxml.cbl の編集

XML 構文の拡張機能を使用して customerdbxml.cbl ファイルを変更し、XML に対応させます。SELECT 文と FD 文を変更して、コピーファイルの指定を変更します。

customerdbxml.cbl の SELECT 文と FD 文は、元のレガシープログラム customerdb.cbl の SELECT 文と FD 文と同じです。

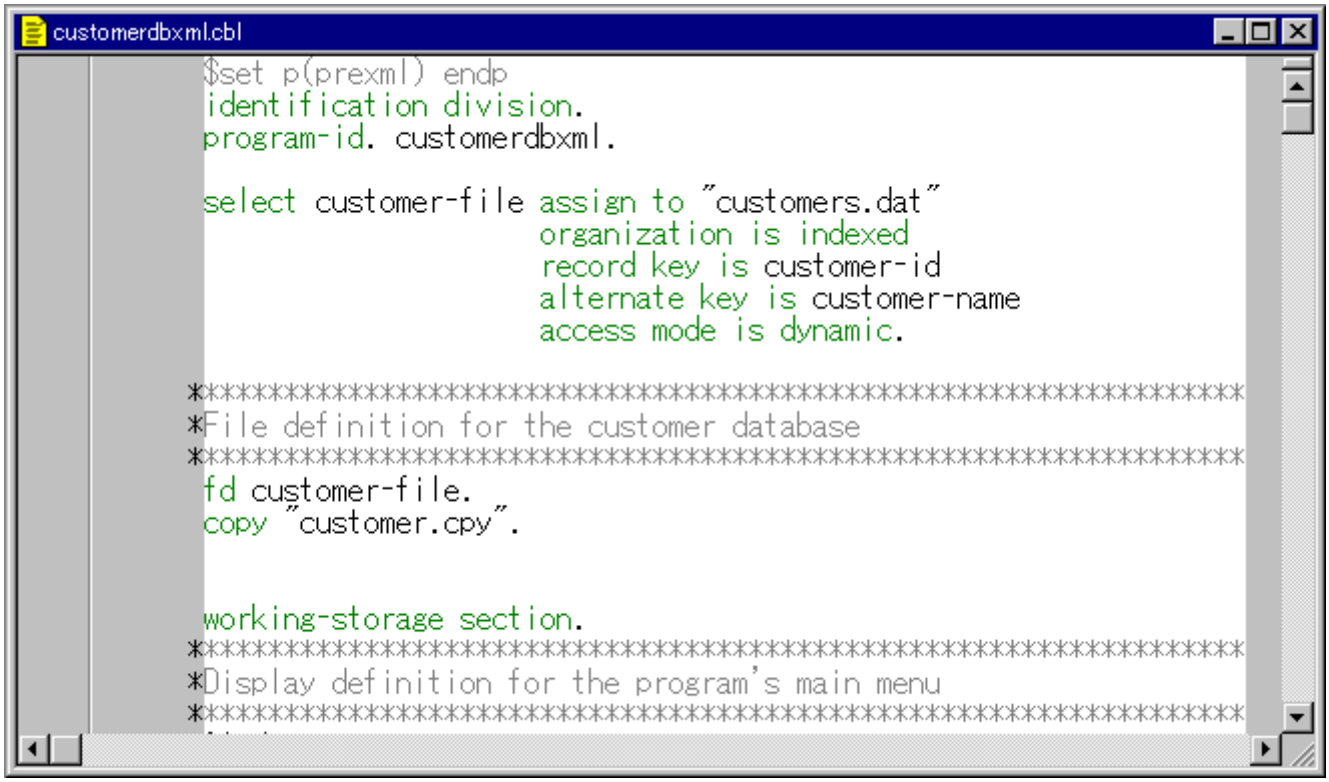


図 16-2: customerdbxml.cbl の SELECT 文と FD 文

注:このプログラムの最初の行にある「$set p(prexml) endp」構文は、コンパイル前に XML プリプロセッサを起動するために必要になります。XML 対応の COBOL のコンパイルについては、『*分散コンピューティング*』マニュアルにある『*XML 構文の拡張機能*』の章の『*XML 対応 COBOL のコンパイル*』を参照してください。

customerdbxml.cbl を編集するには、次の操作を実行します。

1. **「プロジェクト」ウィンドウで customerdbxml.cbl をダブルクリックします。**
2. **SELECT 文を XML に対応させるには、コードを次のように変更します。**

変更前	変更後
assign to "customers.dat"	assign to "customers.xml"
organization is indexed	organization is xml
record key is customer-id alternate key is customer-name access mode is dynamic.	document-type is "customer.xsd" file status is file-status-spec.

3. **FD 文を XML に対応させるには、XML 構文の XD に置換します。コードを次のように変更します。**

変更前	変更後
fd	xd
copy "customer.cpy"	copy "customer-xml.cpy"

4. 変更を保存するには、[閉じる] ×をクリックし、[はい] をクリックします。

XML に対応した SELECT 文と XD 文については、『*分散コンピューティング*』マニュアルにある『*XML 構文の拡張機能*』の章を参照してください。

変更後の customerdbxml.cbl は、次のようになります。

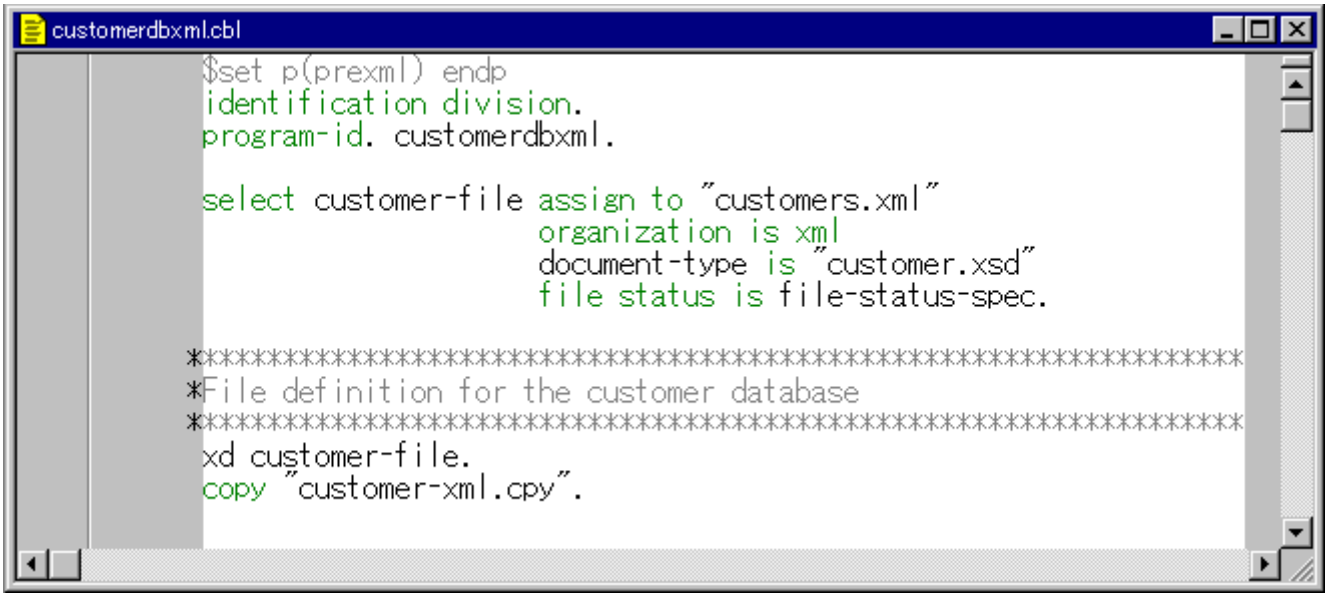


図 16-3: customerdbxml.cbl の SELECT 文と XD 文

指定するコピーファイルを customer.cpy から customer-xml.cpy に変更すると、customer-xml.cpy がプロジェクトに自動的に追加されます。プロジェクトに追加されたこのファイルを参照するには、customerdbxml.cbl をクリックし、[プロジェクト > すべての従属関係の更新] をクリックします。customerdbxml.cbl の次の ⊞(展開) をクリックします。

手続き部のコードを理解するために、customerdbxml.cbl プログラムの残りの部分も表示するには、Net Express またはテキストエディタを使用します。

customers.xml の再確認

customers.xml ファイル内のデータを確認するために、このファイルを表示します。

1. [ファイル > 開く] をクリックします。
2. 「ファイルの種類」フィールドでスクロールして XML ファイル (*.xml) をクリックします。

3. このファイルリストで customers をダブルクリックします。ファイル内に含まれる最後の 3 名の顧客情報は、次のとおりです。

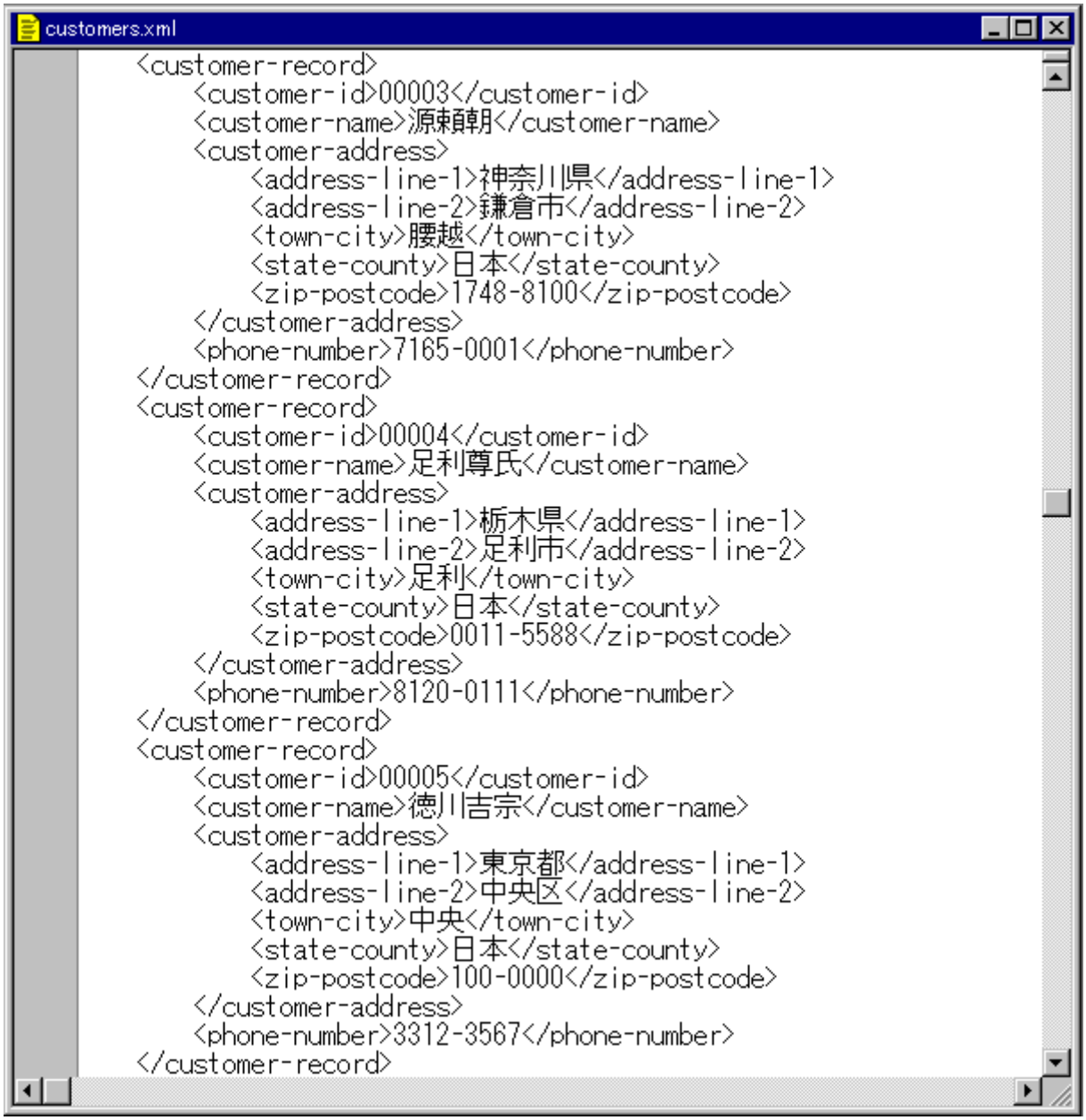


図 16-4: customers.xml 内の最後の 3 名の顧客情報

customerdbxml.cbl のビルドと実行

新しく作成した XML 対応アプリケーションをビルドおよび実行し、レコードを削除します。このアプリケーションの元のバージョンでは、索引ファイルからの読み取りや削除、および、索引フ

ァイルへの書き込みが可能でした。ここでも同じ処理を実行しますが、XML インスタンスドキュメントを使用します。

1. 「プロジェクト」ウィンドウで customerdbxml.cbl をクリックします。
2. [リビルド]  をクリックします。
3. [実行]  をクリックし、DEBUG\customerdbxml が指定されていることを確認して、[OK] をクリックします。
4. 「選択」フィールドに 7 を入力します。
5. 「顧客 ID の入力」フィールドに 00004 を入力します。
6. Y を入力して、レコードを削除します。
7. 「選択」フィールドに 9 を入力してアプリケーションを終了します。[閉じる] ボタンをクリックして「アプリケーション出力」ウィンドウを閉じます。

XML インスタンスドキュメントの表示

customers.xml を再度表示し、レコードが実際に削除されていることを確認します。

1. [ファイル > 開く] をクリックします。
2. 「ファイルの種類」フィールドでスクロールして XML ファイル (*.xml) をクリックします。
3. このファイルリストで customers をダブルクリックします。

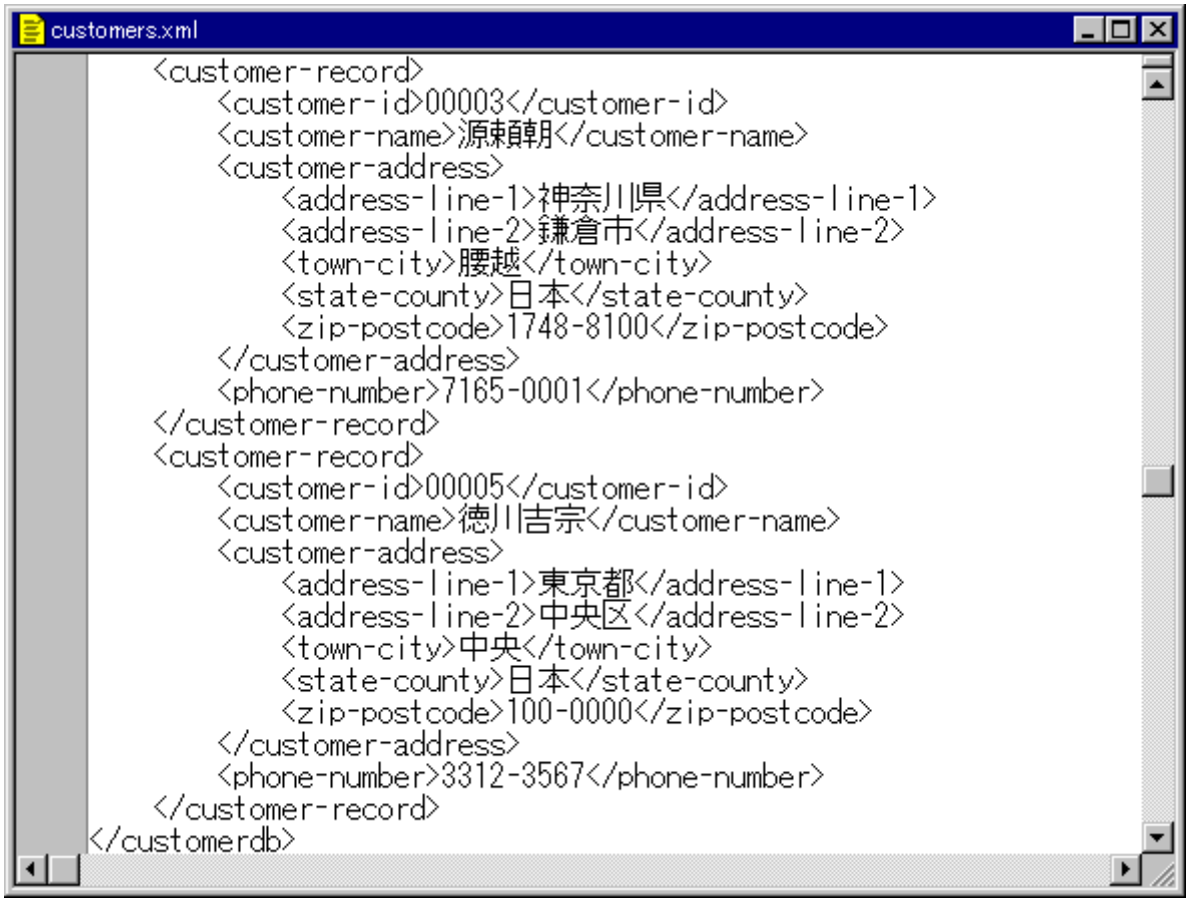


図 16-5: customers.xml 内で削除された顧客 00004

次に進む前に

一連のチュートリアルは、ここまでです。別のチュートリアルに進むには、『*チュートリアルの概要*』の章にあるチュートリアルマップを参照してください。

第 17 章 : Windows GUI の概要

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

グラフィックユーザインターフェイス

「GUI (Graphical User Interface; グラフィックユーザインターフェイス)」または「グラフィックインターフェイス」という用語は、マウス、アイコン、および、ウィンドウを使用するユーザインターフェイスを指します。この用語は、当初、コマンド行インターフェイスと全画面の文字インターフェイスに対するインターフェイスとして紹介されました。

「Windows GUI」という用語は、Windows オペレーティングシステムで提供される GUI を指し、他のシステム (Web ブラウザの HTML の GUI など) と対比して使用します。「Windows GUI」は、「GUI」と略すことがあります。

Net Express では、Windows GUI を作成するための技術がサポートされています。中心的なツールは、Dialog System です。『入門書』マニュアルのこの部分では、Dialog System の使用方法に関するチュートリアルを説明します。Net Express では、下位互換性を確保したり、特殊な用途に使用したりするために、別のツールも用意されています。詳細については、ヘルプの『*概要 - ユーザインターフェイスを作成する*』を参照してください。

Net Express で作成されたソフトウェアに対し、他のシステムを使用して GUI フロントエンドを作成できます。このマニュアルで説明するクライアントのチュートリアルでは、Net Express で作成されたサービスに対してフロントエンドを作成する方法も紹介します。

Dialog System の詳細については、『*Dialog System*』マニュアルを参照してください。

Dialog System のチュートリアルについては、次の章『*Windows GUI アプリケーションの作成*』以降を参照してください。

第 18 章 : Windows GUI アプリケーションの作成

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

このセッションを実行するには、Dialog System をインストールする必要があります。

概要

このセッションと次のセッションでは、Windows のグラフィカルアプリケーションを作成します。

ここでは、Windows GUI アプリケーションの新規作成ウィザードと Dialog System を使用して、データを処理するためのユーザインターフェイスとプログラムを作成します。 インターフェイスを表示するには、IDE でアプリケーションを実行します。

次のセッション『*Windows GUI アプリケーションの仕上げと実行*』では、このプログラムにビジネスロジックを追加して、完全なアプリケーションを実行します。

Windows GUI アプリケーションの新規作成ウィザードを使用して、Windows システムで表示するウィンドウとダイアログボックスを作成します。 これらがアプリケーションのユーザインターフェイスになります。 ウィザードから Dialog System に移動し、ユーザインターフェイスのレイアウトの設計や編集を実行できます。 設計したウィンドウとダイアログボックスのセットをスクリーンセットと呼びます。 Dialog System では、スクリーンセットファイルというファイルを作成して、設計したウィンドウとダイアログボックスを定義します。

また、Dialog System を使用して、スクリーンセットを処理する COBOL プログラムを生成します。 このプログラムを関連付けられたプログラムと呼びます。

スクリーンセットの作成

スクリーンセットを作成するには、次の 4 つの手順を実行します。

1. ウィンドウとダイアログボックス、および、それらの属性 (ステータスバー、ツールバーなど) を定義します。
2. ユーザインターフェイスと COBOL プログラムの間で受け渡すデータを格納するデータ項目を定義します。
3. ウィンドウやダイアログボックスに表示するコントロール (入力フィールド、プッシュボタンなど) を定義します。
4. インターフェイスでのユーザの操作 (プッシュボタンのクリックなど) に対する動作を定義するダイアログを作成します。

手順 4 で作成するダイアログは、Dialog System 独自のスクリプト言語で作成します。

Dialog System のマニュアルでは、通常、コントロールを表す場合に「*グラフィックオブジェクト*」または「*オブジェクト*」という用語を使用します。

準備

Net Express を閉じている場合は、再度開きます。「プロジェクト」ウィンドウなどのウィンドウが開いている場合は、閉じます。

プロジェクトとスクリーンセットの作成

『*Net Express の使用方法*』の章で説明した方法でプロジェクトを作成することもできますが、すぐにスクリーンセットを作成する場合は、Net Express でプロジェクトを作成できます。スクリーンセットを作成するには、次の操作を実行します。

1. [**ファイル > 新規作成**] をクリックし、「新規作成」ダイアログボックスで「**Dialog System スクリーンセット**」を選択し、[**OK**] をクリックします。
2. プロジェクトを作成するかどうかを確認するメッセージに対して [**はい**] をクリックします。
3. 「**Windows GUI プロジェクト**」を選択します。プロジェクト名として「**Welcome**」を入力します。プロジェクトを保存するフォルダとして「**\Program files\Net Express\Base\Demo\Welcome**」を指定します。ここで、**\Program files\Net Express** は Net Express をインストールしたディレクトリ名です。この段階では、このフォルダが存在しないので、このパス全体を手入力するか、または [**参照**] ボタンをクリックしてこのフィールドに「**\Program files\Net Express\Base\Demo**」を表示し、末尾に「**\Welcome**」を手入力します。[**作成**] をクリックします。
4. ディレクトリを作成するかどうかを確認するメッセージに対して [**はい**] をクリックします。(このセッションを実行したことがある場合は、既存のプロジェクトを上書きするかどうかを確認するメッセージが表示されます。[**はい**] をクリックします。)

   このプロジェクトの「プロジェクト」ウィンドウが表示されます。Windows GUI アプリケーションの新規作成ウィザードの最初のページも表示されます。

5. デフォルトのスクリーンセット名 **NewSet.gs** を **Welcome.gs** に変更し、[**次へ**] をクリックします。(このセッションを実行したことがある場合は、既存の **Welcome.gs** を上書きするかどうかを確認するメッセージが表示されます。[**はい**] をクリックします。)
6. MDI (Multiple Document Interface) アプリケーションを作成するために [**はい**] を選択して、子 MDI の数として「**1**」を入力します。[**次へ**] をクリックします。

   SDI (Single Document Interface) は、1 つのウィンドウのみで構成されます。MDI は、1 つのウィンドウを親として、多数のウィンドウとダイアログボックスで構成されます。

   MDI の親ウィンドウは、その内側に子ウィンドウを表示するための背景なので、オブジェクトを追加できません。ユーザが実際に使用するのは、子ウィンドウです。

7. [**ステータスバー**] と [**ツールバー**] を選択し、[**次へ**] をクリックします。

   この操作によってメインウィンドウにステータスバー、ツールバー、および、メニューバーが作成されます。ツールバーとメニューバーは同時に作成されます。

8. [**次へ**] を 2 回クリックします。

   スキップしたこれら 2 つのダイアログボックスは、Dialog System の拡張機能を使用するためのダイアログボックスと実行時の動作を変更するためのダイアログボックスです。 ここでは、デフォルト設定を使用します。

9. 「**スケルトン COBOL プログラムを生成**」がチェックされ、生成する COBOL プログラムの名前が **Welcome.cbl** であることを確認して、[**次へ**] をクリックします。

   次のダイアログボックスには、選択内容の概要が表示されます。

10. [**完了**] をクリックします。

    Windows GUI アプリケーションの新規作成ウィザードによってスクリーンセットとプロジェクトが生成され、終了します。 生成されたファイルの名前がプロジェクトに追加され、プロジェクトの依存関係が検索されます。

    Dialog System がロードされます。

Dialog System のウィンドウ

Dialog System のウィンドウは、図 18-1 のように表示されます。 各ウィンドウの位置は、異なる場合があります。必要な場合は、Net Express IDE を最小化して、これらの新しいウィンドウを大きく表示してください。

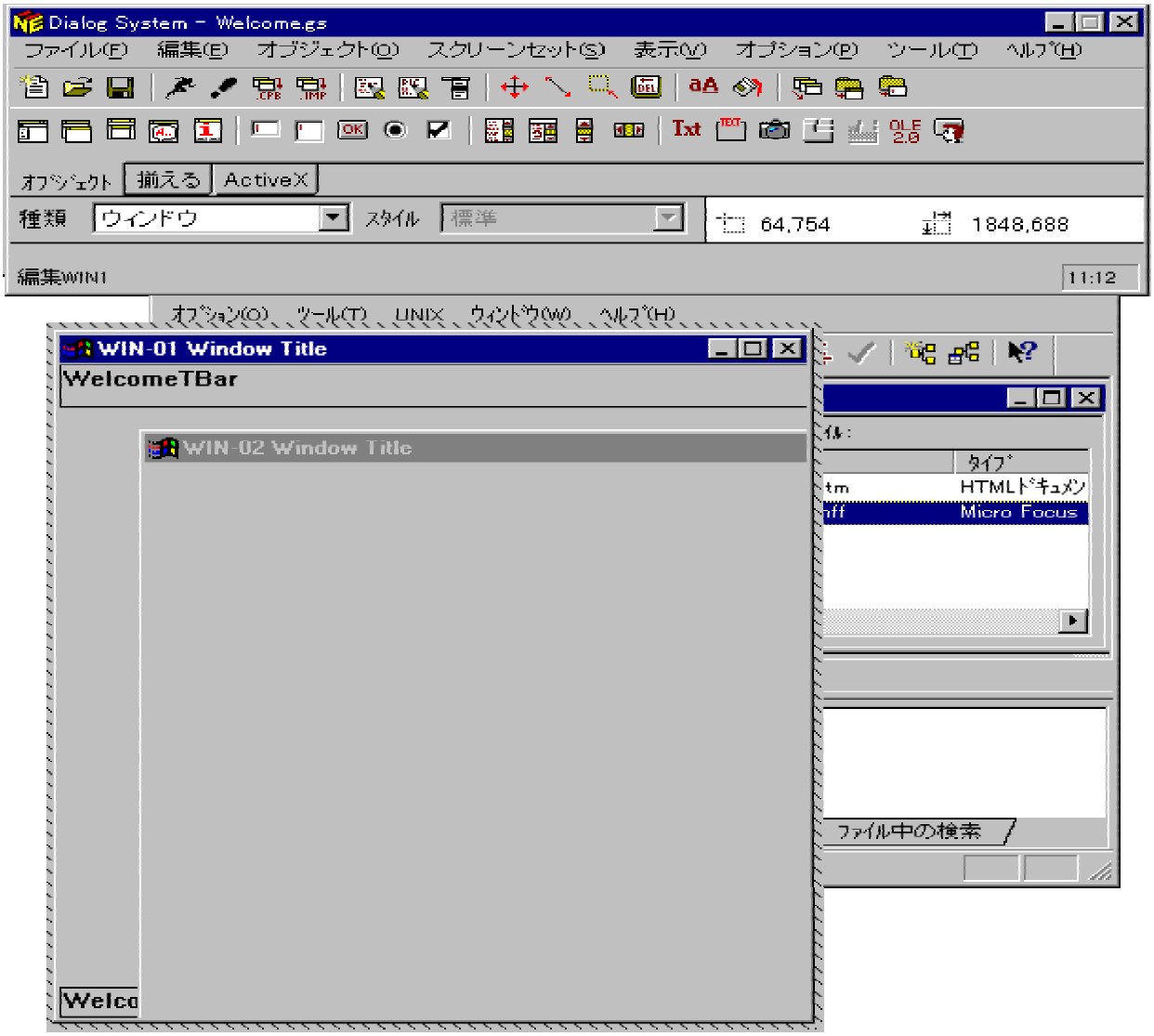


図 18-1: Dialog System のウィンドウ

新しい 2 つのウィンドウは、次のとおりです。

- WIN-01

  「WIN-01 ウィンドウタイトル」というタイトルのウィンドウでスクリーンセットを設計します。この「WIN-01」ウィンドウは、スクリーンセットのメインウィンドウです。アプリケーションを MDI として作成するように指定したので、子ウィンドウ「WIN-02」が「WIN-01」の内側に表示されます。

- Dialog System 独自のウィンドウ

このウィンドウは、メニューバー、ツールバー、および、ステータスバーで構成され、上部に「Dialog System - Welcome.gs」、下部に「編集 WIN-02」というタイトルが表示されます。 初めは、画面上で「WIN-01」と重なっていることがあります。

また、IDE でプロジェクトの変更内容を確認します。

スクリーンセット用データの定義

データを格納する画面上のオブジェクト (入力フィールドなど) は、データ項目と関連付ける必要があります。 データは、データ項目と画面上のオブジェクトの間で受け渡されます。

このチュートリアルでは、ここでデータ項目を定義し、以後でオブジェクトを定義します。 この順序で定義する必要はありません。 後で説明するとおり、データを使用しないでスクリーンセットの表示状態と動作をテストできます。ただし、ユーザとの間で受け渡す情報を明確に把握するために、データを先に定義することをお勧めします。 スクリーンセットの表示状態を変更する場合でもデータ定義を変更する必要はありません。

定義したデータ項目を「*データブロック*」と呼びます。 生成されたプログラムによって Dialog System のランタイムサポートモジュール DSGRUN を呼び出すときに、データブロックがパラメータとして渡されます。 COBOL プログラムとスクリーンセットの間では、このようにしてデータが受け渡されます。

データ項目を定義するには、Dialog System のウィンドウを使用します。 ここでは、「WIN-01」を無視してください。

Dialog System には、データフィールド定義の入力モードが 2 種類あります。 プロンプトモードでは、完成させる必要がある選択リストとダイアログボックスのプロンプトが表示されます。この機能は、Dialog System のデータ宣言の構文に詳しくない場合に便利です。 非プロンプトモード (エディタモード) では、宣言を直接入力します。ここでは、非プロンプトモードを使用します。

1. Dialog System のウィンドウで [**スクリーンセット > データブロック**] をクリックします。

   「データの定義」のメインウィンドウが表示されます。 Dialog System では、このような専用の編集ウィンドウを使用してさまざまな情報を入力します。 これらのウィンドウには、独自のメニューがあります。次のいくつかの手順では、このメニューを使用します。

   既存のデータ宣言が表示されます。 これらのデータ宣言は、スクリーンセットの作成時に自動生成されています。 これらのデータ宣言は、MDI、ツールバー、メニューバー、および、ステータスバーの維持に使用されるのみなので、無視してかまいません。

2. 「データの定義」ウィンドウの [**オプション**] メニューで [**プロンプトモード**] がチェックされていないことを確認します (チェックされている場合は、このメニュー項目をクリックしてチェックを外します)。

3. 「データの定義」ウィンドウをクリックして選択し、既存の項目の末尾までページダウンして **Enter** キーを押します。
4. **I-NAME X 12.0** と入力し、**Enter** キーを押します。

   この行は、自動的にフォーマットされます。小文字で入力してもかまいません。小文字は、大文字に自動的に変更されます。これで最初のデータ項目 I-NAME の定義が完了しました。X は英数字を表し、12.0 は、データ項目の長さが 12 文字で、小数点以下を含まないことを表します。フィールドが英数字でも「.0」は必要です。ただし、省略することができます。その場合は、自動的に挿入されます。

   これらのデータ項目には、生成された COBOL プログラムと Dialog System の両方からアクセスできるので、COBOL の予約語は使用しないでください (COBOL の予約語を使用した場合に警告が表示されません)。そのため、ここでは、NAME ではなく I-NAME と呼んでいます。

5. **GREETING X 22.0** と入力し、**Enter** キーを押します。

   これが 2 つ目のデータ項目 GREETING になります。このデータ項目は、22 文字の英数字です。

   「データの定義」ウィンドウは、図 18-2 のようになります。

図 18-2:「データの定義」ウィンドウ内のデータ宣言

6. 「データの定義」ウィンドウで [**編集 > データの定義の終了**] をクリックします。

   データ定義は画面に表示されなくなりますが、メモリにロードされたままです。データ定義をディスクに保存するには、スクリーンセットを保存するための [**上書き保存**] を実行する必要があります。

Dialog System でスクリーンセットと COBOL プログラムを作成したときに、データブロックを COBOL で定義したコピーファイル (**.cpb**) が生成されます。このコピーファイルは、生成されたプログラムの作業場所節 (ここに実際のデータが格納されます) で使用されます。データブロックを変更した場合は、このコピーファイルを再生成する必要があります。

7. Dialog System のウィンドウで、[**ファイル > 生成 > データブロック COPY ファイル**] をクリックし、デフォルトの **Welcome.cpb** を コピーファイル名として使用します。[**保存**] をクリックします。[**はい**] をクリックして、既存のコピーファイルを上書きすることを確認します。

   データブロックを変更するたびにこのコピーファイルを再生成してください。

   次に進む前に、ここまでの作業を確定させるためにスクリーンセットを保存します。

8. Dialog System のウィンドウで [**ファイル > 保存**] をクリックします。

スクリーンセットへのコントロールの追加

ユーザインターフェイスの表示状態を指定します。ステータスバー、ツールバー、および、メニューバーは、スクリーンセットの作成時に追加済みなので、ここでは、テキストフィールドと入力フィールドを追加します。Dialog System では、このようなコントロールを、通常、「オブジェクト」と呼びます。

画面に表示されている「WIN-01」ウィンドウは、スクリーンセットの親 (メインウィンドウ) です。「WIN-01」ウィンドウの内側に、実際に使用する「WIN-02」子ウィンドウが表示されます。

[**オブジェクト**] メニューまたはオブジェクトツールバーを使用してオブジェクトを追加します。マウスユーザは、オブジェクトツールバーを使用すると便利です。キーボードユーザは、メニューを使用してください。これらの機能は、まったく同じです。このセッションでは、オブジェクトツールバーを使用します。

Dialog System で表示されるウィンドウの初期サイズは必ず同じです。まず、ウィンドウのサイズを変更し、「WIN-01」内に「WIN-02」全体を表示します。

1. ウィンドウの隅をドラッグして、「WIN-01」と「WIN-02」のサイズを変更し、「WIN-02」全体が「WIN-01」ウィンドウに表示されるようにします。
2. 「WIN-02」内をクリックして選択します。Dialog System のウィンドウで [**編集 > 属性**] をクリックします (または、「WIN-02」内を右クリックして、ポップアップメニューの [**属性**] をクリックするか、または、「WIN-02」内をダブルクリックします)。

   「WIN-02」の属性を編集するための「ウィンドウの属性」ダイアログボックスが表示されます。

3. デフォルトのタイトル「**WIN-02 ウィンドウタイトル**」を「**Welcome**」に変更します。残りのデフォルト設定は変更しないで、[**OK**] をクリックします。
4. オブジェクトツールバーの [入力フィールド] ボタン  をクリックして、「WIN-02」ウィンドウ内をクリックします。

   クリックした場所に入力フィールドが作成されます。エンドユーザは、入力フィールドにデータを入力したり、表示したりします。いったん配置した後でもマウスでドラッグして移動できます。入力フィールドをウィンドウの右側に配置します。

5. 入力フィールドをダブルクリックします。

   入力フィールドの属性を定義するための「入力フィールドの属性」ダイアログボックスが表示されます。

6. デフォルトの名前 **EF1** を **I-NAME-DISP** に変更します。

   入力フィールドの名前が I-NAME-DISP になります。

7. [**マスターフィールド**] をクリックします。

   「マスターフィールド」ダイアログボックスが表示されます。このダイアログボックスには、データブロックのフィールドがリストされるので、データフィールドを入力フィールドと関連付けることができます。

8. 「マスターフィールド」ダイアログボックスでページダウンして **I-NAME** フィールドをクリックし、[**OK**] をクリックします。

   「マスターフィールド」には、I-NAME が表示され、「PICTURE 文字列」フィールドが自動的に X(12) に設定されます。これは、I-NAME-DISP 入力フィールドは 12 文字の英数字であり、この入力フィールドにデータ項目 I-NAME 内のデータが表示されることを表しています。

9. 残りの属性に関するデフォルト設定は変更しないで、[**OK**] をクリックします。
10. オブジェクトツールバーの [テキスト] ボタン  をクリックし、入力フィールドの左側をクリックして、入力フィールドのラベルを作成します。
11. ラベルをダブルクリックして、「テキストの属性」ダイアログボックスを開きます。
12. デフォルトのテキスト「**Text**」を「**名前**」に変更し、[**OK**] をクリックします。
13. 同様に 2 つ目の入力フィールドとラベルを追加します。フィールド名に「GREETING-DISP」を使用し、マスターフィールドとして GREETING を選択します。ラベルを「あいさつ」として設定します。
14. オブジェクトツールバーの [OK] ボタン  をクリックし、「WIN-02」ウィンドウ内をクリックして、このボタンを配置します。このボタンは、ウィンドウの下部に配置してください。
15. ボタンをダブルクリックします。

ボタンの属性を定義するための「プッシュボタンの属性」ダイアログボックスが表示されます。

16. デフォルトの名前 **PB1** を **START-BUTTON** に変更します。次に、デフォルトのテキスト「**~Push Button**」を [**開始**] に変更します。残りの属性に関するデフォルト設定は変更しないで、[**OK**] をクリックします。

    図 18-3 のようなスクリーンセットが表示されます。

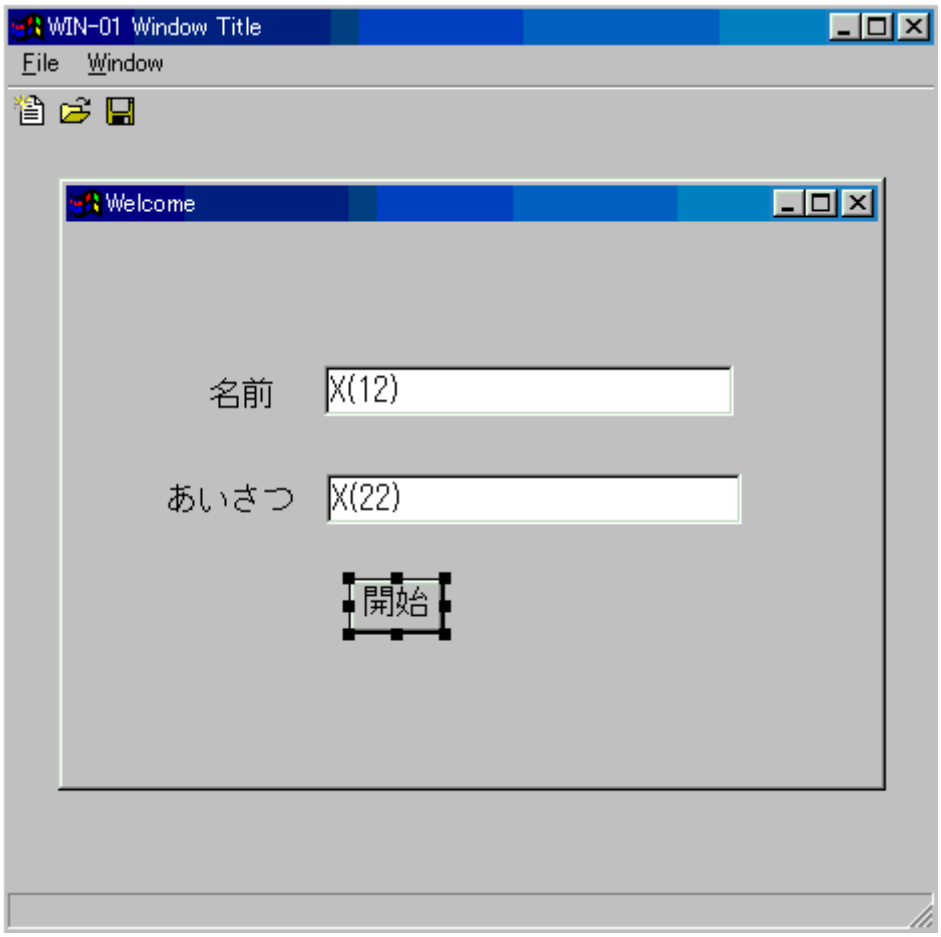


図 18-3: 入力フィールド、テキストフィールド、および、プッシュボタンを設定したスクリーンセット

次に進む前に、ここまでの作業を確定させるためにスクリーンセットを保存します。

17. Dialog System のウィンドウで [**ファイル > 上書き保存**] をクリックします。

スクリーンセットへのダイアログの追加

ここまででウィンドウの表示状態の定義が終わりました。また、ウィンドウのフィールドから受け取ったデータやウィンドウのフィールドに表示するデータを格納するデータ項目も定義しまし

た。ただし、この段階では、ユーザがウィンドウを使用しても ([**開始**] のクリックなど) 何も起こりません。*ダイアログスクリプト* (通常は*ダイアログ*と呼ぶ) を作成してユーザの操作に対する動作を定義する必要があります。

キーを押す、マウスボタンのクリックなどのユーザの操作によって「*イベント*」が発生します。各イベントに対して実行する手続きを作成する必要があります。

ダイアログを特定のオブジェクト (コントロールまたはウィンドウ) に設定したり、グローバルに使用したりできます。これをダイアログの*範囲*と呼びます。オブジェクトに設定されたダイアログをローカルダイアログと呼びます。オブジェクトに対してイベントが発生すると、まずオブジェクトに設定されたダイアログが Dialog System によって検索されます。設定されたダイアログがない場合は、オブジェクトのあるウィンドウに設定されたダイアログが検索されます。まだ、ダイアログが検出されない場合は、グローバルダイアログが検索されます。

ここでは、ユーザが [**開始**] ボタンをクリックした場合に DSGRUN から呼び出し側プログラムに戻るようにダイアログを作成します。ここでは、ユーザが入力したデータをまずデータブロックのフィールドに転記します。ただし、前の操作で、マスターフィールドと関連付けてあるため、この処理は自動的に実行されます。この処理に関するダイアログを作成する必要はありません。

Dialog System では、データを定義する場合にプロンプトモードと非プロンプトモードを選択できます。ここでは、非プロンプトモードを使用します。

グローバルダイアログ

この例では、グローバルダイアログを作成する必要はありません。自動的に生成されたいくつかのグローバルダイアログについて説明します。

1. Dialog System のウィンドウで [**スクリーンセット > グローバルダイアログ**] をクリックします。

   「ダイアログの定義」ウィンドウが表示されます。このウィンドウ内に既存のダイアログが表示されます。これらのダイアログは、スクリーンセットの作成時に、ウィザードで選択した実行時のデフォルト動作を実装するために、自動的に作成されたダイアログです。

   最初の手続きは、ESC イベントと 2 つの関数 MOVE と RETC で構成されます。この手続きでは、ESC イベントが発生すると、DSGRUN によって EXIT-FLAG データ項目に 1 が転記され、呼び出し側プログラムに戻ります (ESC イベントは、ユーザが **Esc** キーを押すと発生します)。

   このダイアログはグローバルダイアログなので、ユーザが **Esc** キーを押すと、現在フォーカスがあるオブジェクトに関係なく、必ずこの手続きが実行されます。ただし、現在フォーカスがあるオブジェクトやそのオブジェクトが設定されたウィンドウに、ESC に対する独自のダイアログが設定されている場合を除きます。

2 つ目の手続きでは、CLOSED-WINDOW イベント (ユーザがウィンドウを閉じるときに発生) に対して同じ動作を設定します。

このダイアログの残りの部分では、MDI、ツールバー、メニューバー、および、ステータスバーの維持に使用されるのみなので、無視してかまいません。

Dialog System には、ダイアログテーブルという手続きがあります。ダイアログテーブルには、イベント、および、そのイベントが発生した場合に実行する関数を指定します。または、手続き名、および、その手続きが別の手続きによって明示的に実行される場合に実行する関数を指定します。Dialog System のイベントと関数の完全なリストについては、ヘルプを参照してください。

COBOL と異なり、手続きを指定する順序は関係ありません。手続きの末尾から次の手続きに制御が移ることはありません。DSGRUN は、各手続きの末尾で停止し、次のイベントを検索します (ただし、最後の関数が RETC である場合を除きます。RETC が実行されると制御が呼び出し側プログラムに戻ります)。

2. 「ダイアログの定義」ウィンドウで [**編集 > ダイアログの定義の終了**] をクリックします。

ローカルダイアログ

ここでは、[**開始**] ボタンを機能させるためのダイアログを作成します。このダイアログは、このオブジェクトのみに適用するので、ローカルダイアログとして作成し、このオブジェクトに設定します。

1. 「Welcome」ウィンドウ内で、[**開始**] ボタンを右クリックし、表示されるポップアップメニューの [**ダイアログ**] をクリックします。

   「ダイアログの定義」ウィンドウが表示されます。Dialog System によって、このタイプのオブジェクトに対してダイアログを作成すると思われるイベントの名前が挿入され、ダイアログが起動します。ここでは、BUTTON-SELECTED (ユーザがボタンをクリックした場合に発生) が挿入されます。

   このイベントの下の行がすでにハイライトされているので、ここが次の入力行です。

2. *** [開始] ボタンから呼び出し側プログラムに戻る**と入力して、**Enter** キーを押します。

   行にアスタリスク (*) を指定すると、そのアスタリスク以降の文字は注釈として処理されます。

3. **RETC** と入力し、**Enter** キーを押します。

   この行は、自動的にフォーマットされます。小文字で入力してもかまいません。小文字は、大文字に自動的に変更されます。Dialog System では、RETC は関数名として認識され、インデントの後に配置されます。

4. **REFRESH-OBJECT GREETING-DISP** と入力し、**Enter** キーを押します。

   ダイアログは、次のようになります。

```
 BUTTON-SELECTED
* [開始] ボタンから呼び出し側プログラムに戻る
    RETC
    REFRESH-OBJECT GREETING-DISP
```

   RETC によって DSGRUN から COBOL プログラムに制御が戻ります。DSGRUN を次に呼び出すと、前回に DSGRUN から制御が移された位置から処理が継続されます。そのため、再度 DSGRUN に制御が戻ったときに、DSGRUN がその手続きが完了していることを認識して次のイベントまで待機できるように、RETC は、通常、手続き内の最後の関数として指定します。ただし、ここでは、DSGRUN の再入時に DSGRUN による処理を継続します (GREETING-DISP を、COBOL プログラムによってマスターフィールド「GREETING」に格納する更新データに書き換える必要があります)。この処理は、REFRESH-OBJECT 関数で実行します。

   関数の構文については、Dialog System のヘルプトピックの『*ダイアログ文：ファンクション*』を参照してください。

5. 「ダイアログの定義」ウィンドウで [**編集 > ダイアログの定義の終了**] をクリックします。
6. Dialog System のウィンドウで [**ファイル > 上書き保存**] をクリックします。

生成されたファイル

ここまでの作業では、ユーザインターフェイスを作成しました。ウィザードと Dialog System によって作成され、プロジェクトに追加されたファイルは、次のとおりです (これらのファイルをすべて表示するには、「プロジェクト」ウィンドウのツリービューで「+」記号をクリックして、ツリービューを展開する必要があります)。

- **welcome.gs** - スクリーンセット
- **welcome.cbl** - 関連付けられたプログラム
- **welcome.cpb** - COBOL のデータブロックを定義するコピーファイル
- **welcomeTBar.cbl** - ツールバーとメニューを処理するプログラム
- **welcomeSBar.cbl** - ステータスバーを処理するプログラム

プロジェクトのビルド時に作成された、次のオブジェクトファイルもプロジェクトに追加されています。

- **welcome.int** - コンパイル済みの関連付けられたプログラム
- **welcomeTBar.int** - コンパイル済みのツールバー / メニュープログラム
- **welcomeSBar.int** - コンパイル済みのステータスバープログラム

また、複数のコピーファイル (拡張子 **.cp***) も作成されています。

上記のファイル名は、小文字で表記されていますが、プロジェクト内では、大文字の場合があります。 ファイル名については、大文字小文字が区別されません。

スクリーンセットを更新する必要がある場合は、「プロジェクト」ウィンドウの **Welcome.gs** をダブルクリックし、Dialog System を開きます。 更新したスクリーンセットを保存すると、コピーファイルが再生成されるので、コピーファイルを直接編集しないでください。ウィザードで生成した **.cbl** プログラムは、編集できます。

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には、次の手順を実行します。

1. Net Express の IDE で [**プロジェクト > すべてをリビルド**] をクリックします。

   Net Express でプログラムがコンパイルされ、実行形式ファイルがビルドされます。「出力」ウィンドウに「リビルドの完了」というメッセージが表示された後で、次の作業に進みます。

スクリーンセットのテスト

Dialog System で、関連付けられたプログラムを使用しないで、スクリーンセットを実行できます。 そのため、スクリーンセットの動作をテストして評価し、必要に応じてスクリーンセットの定義に戻って修正できます。 この機能は、インターフェイスのプロトタイプ化に便利です。

テスト中にツールバー、メニューバー、および、ステータスバーが実行されるので、アプリケーションをあらかじめビルドする必要があります。

1. Dialog System のウィンドウで [**ファイル > 実行**] をクリックします。

   Dialog System のウィンドウと「WIN-01」ウィンドウが表示されなくなります。次に、「WIN-01」が実行時と同様に表示されます。 ウィザードで指定したとおり、ツールバー、メニューバー、およびステータスバーが配置されていることに注目してください。「WIN-02」は、「WIN-01」内に表示されています。

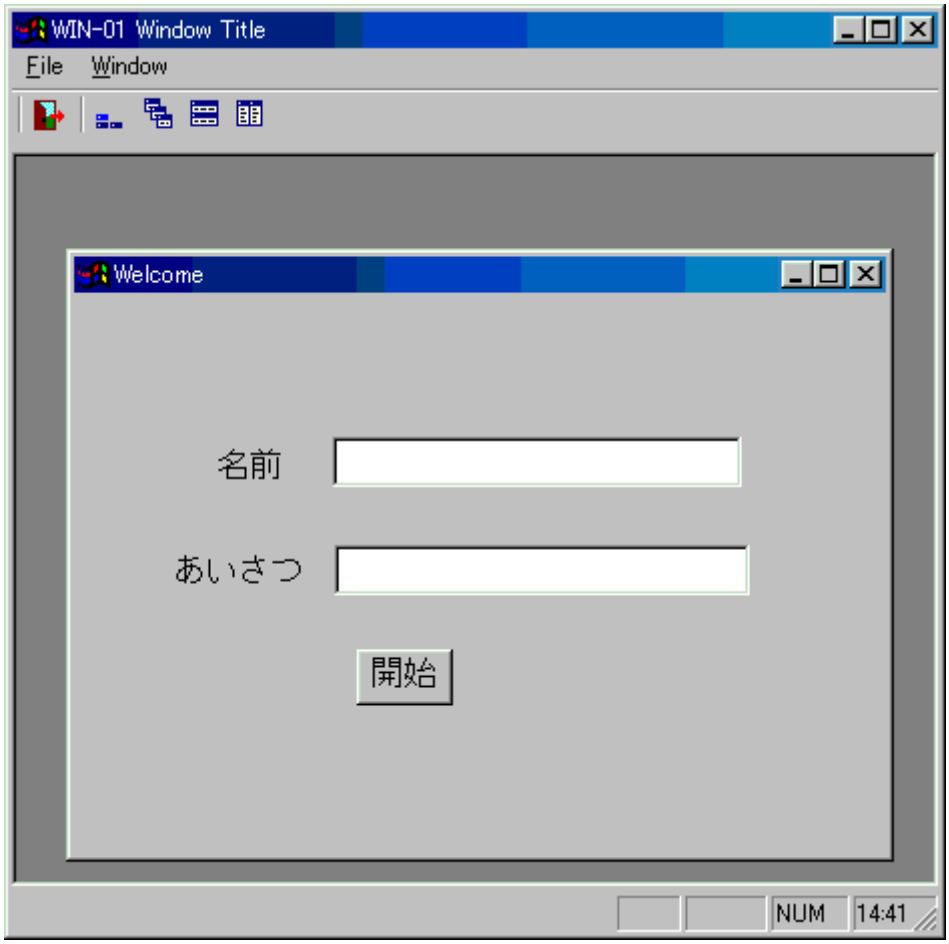


図 18-4: 実行機能によって表示されるスクリーンセット

ウィザードで作成されたツールバーとメニューバーは、生成されたコントロールプログラムによって作成されます。このときは、Net Express のクラスライブラリが使用されます。ウィザードで作成されたツールバーとメニューバーの修正方法などの詳細については、『***Dialog System ユーザガイド***』を参照してください。ウィンドウにメニューを作成する場合は、メニューを作成するための Dialog System 独自の統合システムも使用できます。詳細については、『***Dialog System ユーザガイド***』にある『*ウィンドウとオブジェクト*』の章を参照してください。

画面の右下に小さいダイアログボックスが表示されます。このダイアログボックスには、[**デバッグ**] ボタンと [**中止**] ボタンが配置されています。Dialog System でこのスクリーンセットテスト機能を使用する場合は、スクリーンセットの実行が Screenset Animator によって制御されます。Screenset Animator は、ダイアログのデバッガで、概念上は IDE の COBOL デバッグ機能に類似しています。このダイアログボックスでは、次の操作を実行できます。

- **デバッグ** - バッチ処理によるステップ実行などの Screenset Animator の機能を使用するためのダイアログが表示されます。
- **中止** - スクリーンセットのテストを停止し、Dialog System に戻ります。

2. 定義した [**開始**] ボタンをクリックします。

   図 18-5 のようなダイアログボックスが表示されます。

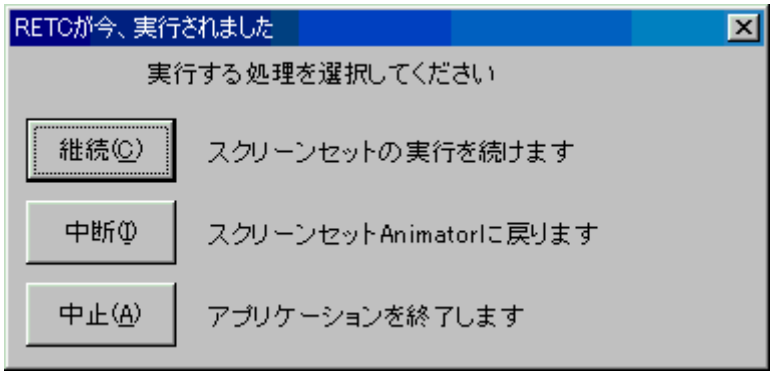


   図 18-5：「RETC」ダイアログボックス

   [**開始**] をクリックすると、このボタンに対して作成したダイアログ (RETC コマンドを含むダイアログ) が実行されます。 RETC が実行されると、必ず Screenset Animator がこのダイアログボックスに切り替わるので、次の操作を選択する必要があります。

   - 継続 - アニメートしないでスクリーンセットのテストを継続します。
   - 中断 - バッチ処理によるステップ実行などの Screenset Animator の機能を使用するためのダイアログが表示されます。
   - 中止 - スクリーンセットのテストを停止し、Dialog System に戻ります。

   アプリケーション内のスクリーンセットを実行するときにダイアログをデバッグするには、DSGRUN パラメータを使用して Screenset Animator を起動します。 Screenset Animator の詳細については、Dialog System のヘルプを参照してください。

3. [**中止**] をクリックします。
4. Dialog System のウィンドウで [**ファイル > 終了**] をクリックします。

次に進む前に

次のセッション『*Windows GUI アプリケーションの仕上げと実行*』では、このアプリケーションを完成させて実行します。

作業を中断する場合は、プロジェクトを閉じてかまいません。次のセッションを始めるときに再度開いてください。 また、プロジェクトを閉じずに、Net Express 全体を閉じてもかまいません。

第 19 章 : Windows GUI アプリケーションの仕上げと実行

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

このセッションを実行するには、Dialog System をインストールする必要があります。

概要

前のセッションでは、Windows GUI アプリケーションの新規作成ウィザードを使用して、スクリーンセットを設計し、COBOL プログラムを生成しました。 ここでは、生成された COBOL プログラムを編集して、ビジネスロジックを追加します。 その後で、IDE を使用して、完成したアプリケーションを実行します。

Windows GUI アプリケーションの新規作成ウィザードでは、**.cbl** ファイル内に COBOL 原始プログラムを生成します。この原始プログラムには、スクリーンセットを入出力するためのコードが含まれています。 この **.cbl** ファイルを編集し、スクリーンセットから入力されたデータを処理して出力データを作成するためのビジネスロジックを追加します。このスクリーンセット処理プログラムとビジネスロジックを完全に切り離すために、CALL 文のみを追加して他のプログラムを呼び出すこともできます。

関連付けられたプログラムの構造

関連付けられたプログラムには、通常、次のようなメインループが指定されます。

1. DSGRUN の呼び出し
2. DSGRUN から返された状態コードのテスト
3. 挿入したビジネスロジックの実行
4. 出力を表示してさらに入力を受け付けるための DSGRUN 呼び出しの繰り返し

Windows GUI アプリケーションの新規作成ウィザードによって生成されたプログラムは、このような構造をもっています。別の例として、**Net Express\DialogSystem\Demo\Customer** ディレクトリの Customer デモも後で参照します。

準備

Net Express と Welcome プロジェクトが開いていない場合は、開きます。

COBOL プログラムの編集

ビジネスロジックを COBOL プログラムに追加する場合は、どのテキストエディタでも使用できますが、ここでは、IDE を使用します。

1. 「プロジェクト」ウィンドウの **Welcome.cbl** をダブルクリックします。

   テキストウィンドウが開き、プログラムのソースが表示されます。必要な場合は、ソースを見やすいようにウィンドウのサイズを変更してください。[**表示**] メニューの [**コピーファイルをすべて非表示**] の次にあるボタンのようなアイコンが押されていることを確認します。押されていない場合は、このメニュー項目をクリックします。

   プログラムの動作を確認するには、数分かかります。

2. 作業場所節の **COPY "Welcome.CPB"** の行にカーソルを合わせ、[**ファイル > コピーファイル > 表示**] をクリックします。

   welcome.cpb ファイルは、データブロックです。ファイルの末尾にデータ項目 I-NAME と GREETING が表示されます。

   もう 1 つの COPY ファイル **ds-cntrl.mf** を制御ブロックと呼びます。このファイルは、Net Express で提供され、すべての Dialog System アプリケーションに必要な標準の定義が記述されています。

3. Program-Body 節で、**PERFORM Call-Dialog-System** 文とその後のピリオドの間に、次の行を挿入します。

   ```
   string "Hello " I-Name delimited by size into Greeting
   ```

   実際のアプリケーションでは、通常、ここで副プログラムを呼び出し、スクリーンセットからのデータを処理して、表示データを計算または検索します。EVALUATE 文を使用して、DSGRUN からの戻り値に応じて操作を実行することもできます。その場合は、通常、WHEN EXIT-FLAG-TRUE CONTINUE 分岐を指定します。

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には、次の手順を実行します。

1. [**プロジェクト > すべてをリビルド**] をクリックします。

   Net Express でプログラムが保存およびコンパイルされ、実行可能ファイルがリビルドされます。「出力」ウィンドウに「リビルドの完了」というメッセージが表示された後で、次の作業に進みます。

アプリケーションの実行

アプリケーションを実行するには、次の手順を実行します。

1. [**アニメート > 実行**] をクリックします。

   「アニメーションの起動」ダイアログボックスが表示されます。

2. 「アプリケーションの起動」ダイアログボックスの [**OK**] をクリックします。

   プログラムが実行され、スクリーンセットが表示されます。

3. 「**名前**」フィールドにユーザ名を入力して [**開始**] をクリックします。I-NAME-DISP の長さを 12 文字と指定したので、入力できる名前は 12 文字までです。

   「**あいさつ**」フィールドの「Hello」の後にこの名前が追加され、スクリーンセットが再表示されます。

4. 「**名前**」フィールドをクリックしてユーザ名を削除し、別の名前を入力して [**開始**] をクリックします。

   プログラムでメインループが再度実行され、「Hello」の後に新しい名前が表示されます。

5. **Esc** キーを押してウィンドウを閉じ、プログラムを終了します。

次に進む前に

プロジェクトを閉じます。

別のセッションに進む場合は、Net Express を開いたままにしてください。

一連のチュートリアルは、ここまでです。別のチュートリアルに進むには、『*チュートリアルの概要*』の章にある*チュートリアル マップ*を参照してください。

第 20 章 : UNIX でのアプリケーションのディプロイ

チュートリアルは、『*チュートリアルの概要*』の章にある『*チュートリアルマップ*』に表示されている矢印の順に読み進んでください。

このセッションを実行するには、UNIX オプションをインストールする必要があります。

概要

UNIX オプションを使用して、Net Express で作成したアプリケーションを、Micro Focus COBOL for UNIX V3.1 以上がインストールされている UNIX システムにディプロイします。

このサンプルセッションでは、既存の COBOL アプリケーションを使用します。 UNIX オプションを使用して UNIX システムにアプリケーションをディプロイすると、UNIX 上でアプリケーションがリビルドされ、実行されます。 サンプルアプリケーションは、1 つの COBOL プログラムのみで構成されています。このプログラムでは、1 つの出力ファイルが作成され、そのファイル内に 1 つのレコードのみが格納されます。

UNIX 上に Web アプリケーションをディプロイする方法については、このマニュアルを読み終えた後で、『***分散コンピューティング***』オンラインマニュアルを参照してください。

UNIX では、Dialog System で作成された Windows GUI アプリケーションをディプロイすることはできません。 ただし、UNIX オプションには、キャラクタユーザインターフェイスを作成するための文字バージョンの Dialog System が含まれています。このバージョンの Dialog System については、『***Dialog System 文字モードユーザガイド***』を参照してください。

準備

このセッションを実行する前に、UNIX システムに SCP (Server Control Program; サーバコントロールプログラム) をインストールする必要があります。 UNIX オプションで**パブリッシュ**機能を実行するためには、SCP が必要です。 SCP のインストール方法については、『***UNIX オプションユーザガイド***』の付録『*SCP と Samba のインストール*』を参照してください。

また、UNIX システムに接続できる必要があります。 UNIX システムにログインし、パスワードなしで透過的にログインできるように **.rhosts** ファイルを設定します。 詳細については、UNIX のマニュアルを参照してください。

Net Express が開いていない場合は、開きます。「プロジェクト」ウィンドウやテキストウィンドウが開いている場合は、閉じます。

プロジェクトの作成

アプリケーション用のプロジェクトを作成するには、次の手順を実行します。

1. [**ファイル > 新規作成**] をクリックし、「新規作成」ダイアログボックスで「**プロジェクト**」を選択し、[**OK**] をクリックします。
2. 「新規作成」ダイアログボックスの [**既存アプリケーションから作成するプロジェクト**] をクリックします。
3. プロジェクト名として「**Unixo**」を入力し、プロジェクトを格納するフォルダとして **Net Express\Base\Demo\Unixo** を入力して、[**作成**] をクリックします。
4. このチュートリアルを実行したことがある場合は、既存のプロジェクトを上書きするかどうかを確認するメッセージが表示されます。[**はい**] をクリックします。
5. [**ファイルの追加**] をクリックします。
6. **Unixo.cbl** を選択して [**追加**] をクリックします。
7. [**次へ**] をクリックします。
8. 次のダイアログボックスで [**次へ**] をクリックします。
9. [**完了**] をクリックして、プロジェクトを作成します。

   このプロジェクトの「プロジェクト」ウィンドウが表示され、プロジェクトの依存関係が検索されます。この段階でプロジェクトに含まれているのは、アプリケーションのビルドに使用する COBOL プログラムのみです。

アプリケーションのビルド

このアプリケーションをデバッグするには、UNIX に移植できない機能がある場合に警告を表示するオプションを設定する必要があります。

1. [**プロジェクト > プロパティ**] をクリックします。
2. 「**プロジェクト指令**」フィールドの末尾にあるセミコロン (;) の前に **WARNINGS"2"** と入力します (**ENSUITE"3"**の後は空白にします)。[**OK**] をクリックします。
3. [**プロジェクト > すべてをリビルド**] をクリックして、プログラムをコンパイルします。

   「出力」ウィンドウに、中間コードを移植できないことを警告するメッセージが表示されます。必要な場合は、メッセージを見やすいように「出力」ウィンドウのサイズを変更してください。ファイルを編集して、移植できない構文を変更します。

4. 「出力」ウィンドウの警告メッセージをダブルクリックします。

   「編集」ウィンドウが開き、問題のある行の記述部に赤色でマークがつけられます。

5. この行を編集して、OUT FILE を OUTFILE に置換します。

   コンパイラによってこの行にフラグが設定されたのは、ファイル名に空白文字が含まれていたためです。Windows では、ファイル名に空白文字を使用できますが、UNIX では使用できません。

6. [**プロジェクト > リビルド**] をクリックします。

警告が表示されずにプログラムがコンパイルされます。次に、このアプリケーションを UNIX システムにディプロイします。

7. **Unixo.cbl** が表示されているテキストウィンドウを閉じます。

UNIX 端末としての PC

Net Express には、PowerTerm というターミナルエミュレーションソフトウェアが含まれているので、PC を UNIX 端末として使用できます。このチュートリアルの実行中に、PowerTerm を使用して UNIX ホストシステムの端末として PC で別途ウィンドウを開くと便利です。他のターミナルエミュレーションソフトウェアや UNIX 端末を使用しているためにこの操作が不要な場合は、ここを飛ばしてください。

PowerTerm は日本語版の Net Express には含まれていません。

1. [**UNIX > ターミナル**] をクリックします。
2. PowerTerm で [**通信 > 接続**] をクリックします。
3. 「接続」ダイアログボックスの「セッションの種類」が、使用中のネットワークと互換性がある種類に設定されていることを確認します。「ホスト名」フィールドに、使用する UNIX ホストマシンの名前を入力します。[**接続**] をクリックします。
4. 通常どおりに、UNIX マシンにログインします。

パブリッシュ用オプションの設定

UNIX オプションでは、ディプロイを「*パブリッシュ*」と呼びます。パブリッシュ操作では、ファイルを UNIX システムのディレクトリに正しくコピーし、UNIX 上でアプリケーションをビルドします。パブリッシャは、この処理を実行するツールです。Net Express のメニューからアクセスできます。

まず、必要なパブリッシュ操作の詳細をパブリッシャに指定します。

1. [**UNIX > セットアップ**] をクリックします。
2. 「Welcome to Micro Focus Net Express」画面に表示される準備は、『*準備*』で完了しているので、この画面の [**OK**] をクリックします。

   図 20-1 に示すようなフォームが表示されます。

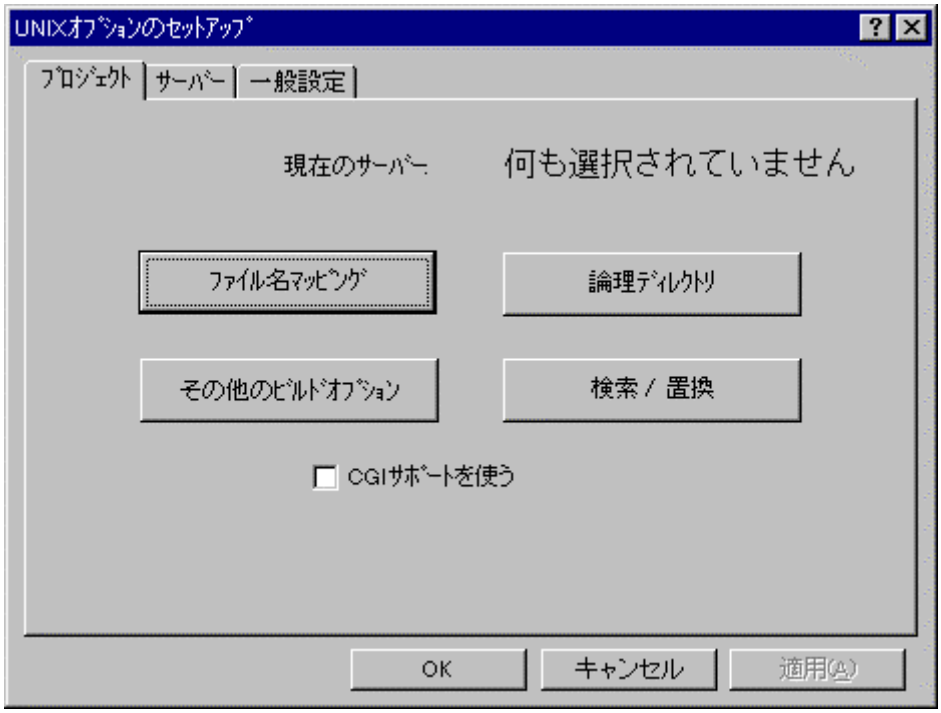


図 20-1:「セットアップ」ダイアログボックス

3. [サーバ] タブをクリックします。図 20-2 に示すようなフォームが表示されます。

図 20-2:「サーバ」ダイアログボックス

現在定義されているサーバがリストボックスに表示されます。これらのサーバの 1 つを選択した場合は、このタブのボタンを使用して行った変更が、選択したサーバに適用されます。「**新しいサーバ**」を選択すると新しいサーバを定義できます。構成済みのサーバがない場合は、デフォルトで「**新しいサーバ**」が選択され、このダイアログの [**設定**] 以外のボタンがすべて非アクティブになります (上の図を参照)。

4. [**設定**] をクリックします。新しいサーバ名を指定するためのダイアログが表示されます。

図 20-3: サーバの指定

5. 有効なサーバ名を入力して [**OK**] をクリックします。次のダイアログボックスが表示されます。

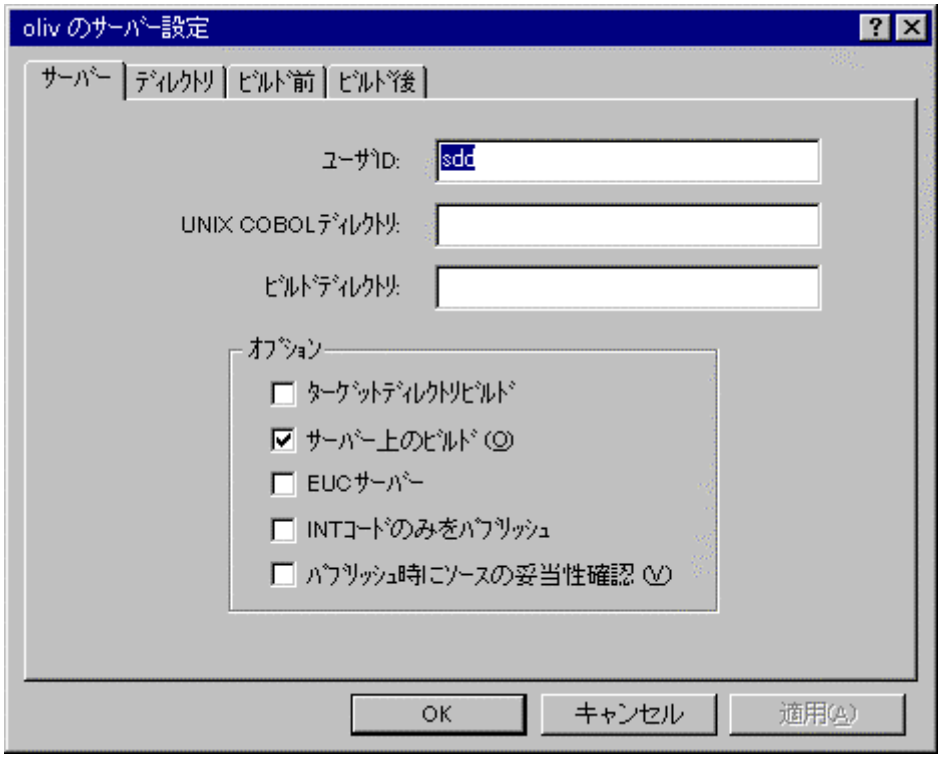


図 20-4: サーバ設定

6. 「**ユーザ ID**」フィールドに、UNIX システムへのログインに使用するユーザ ID を入力します。
7. 「UNIX COBOL ディレクトリ」フィールドに、UNIX システムの COBOL システムディレクトリの名前を入力します。このディレクトリを検索するには、UNIX システムで **echo $COBDIR** と入力します。このフィールドを空白のままにした場合は、デフォルトの **/usr/lib/cobol** が使用されます。
8. アプリケーションをパブリッシュする UNIX のディレクトリを決定し、「ビルドディレクトリ」フィールドにそのディレクトリ名を入力します。
9. [**OK**] をクリックします。
10. [**OK**] を再度クリックします。

アプリケーションのパブリッシュ

アプリケーションをパブリッシュするには、次の手順を実行します。

1. 「セットアップ」ダイアログボックスの「ビルドディレクトリ」フィールドに入力したディレクトリが、使用する UNIX システムに存在することを確認します。
2. [**UNIX > パブリッシュ**] をクリックします。

   PC でプロジェクトがビルドされ、**Makefile** (UNIX での動作も同じです) というファイルが作成されます。プロジェクトファイルが UNIX システムにコピーされ、**Makefile** が UNIX で実行されます。

問題が発生した場合は、設定を調べ ([**UNIX > セットアップ**] を再度クリック)、サーバ名、UNIX ログイン名、および、ディレクトリ名が正しく指定されていることを確認します。

まだ問題が解決しない場合は、『***UNIX オプションユーザガイド***』にある『*ヒントとトラブルシューティング*』の章を参照してください。

3. UNIX マシンでリストユーティリティを使用してログファイル **Make.log** を表示します。

   ログファイルにエラーが表示されている場合、または、『*作成されたファイル*』で説明するファイルが UNIX のビルドディレクトリにない場合は、設定を見直して、再試行してください。

   [**パブリッシュ**] を選択すると、前回プロジェクトをパブリッシュした後に変更されたファイルのみがパブリッシュされます。[**すべてをパブリッシュ**] を選択すると、無条件にすべてのファイルがパブリッシュされます。このチュートリアルを前に実行したことがある場合でも、UNIX のビルドディレクトリからファイルを削除しないでください。[**パブリッシュ**] を選択した場合は、ファイルが削除されていることが検出されず、新しいファイルが作成されません。UNIX ディレクトリからファイルを削除してしまった場合は、[**すべてをパブリッシュ**] を使用してください。

作成されたファイル

この段階では、UNIX オプションによって PC のプロジェクトディレクトリに次のファイルが作成され、UNIX のビルドディレクトリにこのファイルがコピーされています (PC ではこのファイルを使用しないので、プロジェクトには追加されていません)。

- **Makefile** - UNIX でのビルドを実行するスクリプトファイル

また、PC のプロジェクトディレクトリ内の次のファイルも UNIX のビルドディレクトリにコピーされています。

- **Unixo.cbl** - COBOL プログラム

ビルドディレクトリには、UNIX でのビルド過程で作成された次のファイルも保存されています。

- **Make.log** - UNIX でのビルド結果を表すログ
- ビルド用に作成されたいくつかのファイル

UNIX では、大文字と小文字が区別されるので注意してください。

アプリケーションの実行

アプリケーションを実行するには、次の手順を実行します。

1. UNIX システムで、UNIX COBOL アプリケーションを通常実行する場合と同じ方法でアプリケーションを実行します。たとえば、次のように実行します。

   ```
   cobrun Unixo.int
   ```

2. ビルドディレクトリ (**ls** コマンドなどを使用) を表示して、**OUTPUT** ファイルが存在するかどうかを確認します。リストユーティリティを使用して、「12345abcde」という語句が含まれたレコードが 1 つのみ存在することを確認します。

   Unixo.cbl では、このファイルが作成されるのみです。このファイルが存在する場合は、アプリケーションが正しくビルドされ、実行されたことを表します。

次に進む前に

プロジェクトを閉じます。

PowerTerm を使用している場合は、通常の方法で UNIX からログオフし、PowerTerm の [**ファイル**] メニューの [**終了**] をクリックします。

別のセッションに進む場合は、Net Express を開いたままにしてください。

一連のチュートリアルは、ここまでです。別のチュートリアルに進むには、『*チュートリアルの概要*』の章にある*チュートリアルマップ*を参照してください。

付録 A 章 : 他の機能

この付録では、『*ようこそ*』の章で紹介した主な機能以外の機能を説明します。 ただし、Net Express のすべての機能が網羅されているわけではありません。

主に他の COBOL システムから移行するための機能は、この付録の末尾にある『A.2』で説明します。

機能説明

機能には、開発ツールとアプリケーションで使用する機能があります。

機能リスト

機能はアルファベット順に説明します。

Btrieve インターフェイス

Novell 社の Btrieve ファイル処理システムに対して、この Btrieve インターフェイスを使用すると、COBOL ファイルの入出力構文を使用して Btrieve ファイルにアクセスできます。 そのため、Btrieve ファイルと Net Express ネイティブの形式のファイルを同じプログラム内で使用できます。

呼び出し可能ファイルハンドラ

呼び出し可能ファイルハンドラは、Net Express のファイルハンドラとのインターフェイスです。プログラムで CALL 文を使用して呼び出すことができます。 すべての COBOL 編成のファイルを下位レベルで制御できるので、ファイルやデータベースを処理するための高度なプログラムを作成できます。 C またはアセンブラなどの他の言語から COBOL 形式のファイルにアクセスする場合にも使用できます。

呼び出し可能リビルド

呼び出し可能 Rebuild は、Rebuild ユーティリティとのインターフェイスです。プログラムで CALL 文を使用して呼び出すことができます (『*リビルド*』を参照)。

呼び出し可能ソート

呼び出し可能ソートモジュールは、データファイルのソートと並べ替えを可能にするスタンドアロンのソートルーチンです。デフォルトの COBOL ソート機構を使用するより高速です。この呼び出しインターフェイスを使用すると、データを柔軟にソートしたり、別のソートモジュールのかわりに使用したりできます。

Cbllink

Cbllink は、システムリンカーに対する高度なコマンド行インターフェイスです。通常は、アプリケーションのプロジェクトによって自動的に呼び出されます。

クライアント / サーバ結合

クライアント / サーバ結合は、ネットワーク経由で多種のプロトコルを使用して通信するための標準的な機構です。クライアント / サーバ結合を使用できるように作成されたプログラムは、モジュールを交換するのみで、サポートされるどのプロトコルでも通信できます。

COBOL システムライブラリルーチン

COBOL システムライブラリルーチンは、プログラムから呼び出すことができるルーチンのセットです。このルーチンを使用すると、COBOL 言語のみでは実行できない操作を実行できます。

コマンドプロンプト

Net Express のコマンドプロンプトは、Net Express 環境を設定した場合に使用できるコマンド行セッションです。コマンドプロンプトから Net Express の一部の機能を実行できます。Windows では、[**スタート > プログラム > Micro Focus Net Express > Net Express コマンドプロンプト**] をクリックします。

CGI (Common Gateway Interface) サポート

CGI サポートでは、CGI または ISAPI 規格に準拠したプログラムをリモートの Web ブラウザから起動し、Web サーバソフトウェアで制御しながら実行できます。

埋め込み HTML

埋め込み HTML は、プログラムの手続き部で HTML を使用して、Web ページを作成および表示するための COBOL の拡張機能です。

Fileshare

Fileshare は、入出力要求をパケットとして圧縮し、ネットワークを介して送信することで、ネットワーク入出力を高速にします。そのため、ファイル処理はファイルが存在するサーバ上で実行されます。 論理データベース内のいくつかのファイルとリンクできます。 データベースのアクセス中に回復ログを設定でき、データの高度な完全性を提供します。 Fileshare には、トランザクションをログして、すべての情報が完結するまでファイルの変更を遅延させる機能があります。 COMMIT で変更を保証し、ROLLBACK で変更をキャンセルできます。

統合プリプロセッササポート

統合プリプロセッササポートは、コンパイラの拡張機能です。 この機能を使用すると、ユーザが定義した言語のプロセッサをコンパイラで起動し、COBOL 以外の構文を COBOL 構文に変換できます。 デバッグ時には、プリプロセッサで文が変換される前の元のソースが表示されます。

プリプロセッサの作成と使用方法については、[**ヘルプ > ヘルプトピック**] をクリックしてください。 ヘルプの表示後、[**索引**] タブをクリックして「**プリプロセッサ**」と入力します。[**表示**] をクリックして、トピックを選択します。

マルチスレッド

マルチスレッド機能を使用すると、同一のアプリケーションに対して複数の実行を並列処理できます。

各国語サポート

NLS (National Locale Support; 各国語サポート) を使用すると、プログラムの実行時に、文字集合、通貨記号、および、編集記号がユーザの国に合わせて自動的に変換されます。 この機能では、各国語の文字 (アクセント付き文字など) が正しく照合および変換され 、対応する言語のメッセージをメッセージファイルから取得するためのライブラリルーチンが提供されます。

各国語サポートの詳細については、次の操作を実行してヘルプトピックを参照してください。まず、[**ヘルプ > ヘルプトピック**] をクリックします。 ヘルプが表示されたら、[**索引**] タブをクリックして「**各国語サポート**」または「**NLS**」と入力します。[**表示**] をクリックして、トピックを選択します。

オブジェクト指向 COBOL 構文

オブジェクト指向構文は、OO (object-oriented; オブジェクト指向) プログラミングを使用するための COBOL の拡張機能セットです。 OO クラスライブラリ (OO プログラムで使用する定義済みの機能セット) によってサポートされます。

OpenESQL

OpenESQL は、Net Express に組み込まれたプリプロセッサです。ソースコードに SQL 文を埋め込んで、COBOL プログラムから ODBC データソースにアクセスできます。

OpenESQL アシスタント

OpenESQL アシスタントは、COBOL プログラムに埋め込む SQL 文を作成するためのウィザードです。

リビルド

リビルドユーティリティは、索引ファイルを維持するためのツールです。このツールを使用すると、キーとデータを再編成してパフォーマンスを向上させることができます。また、既存のファイルに対する新しいキー構造の設定、破損した指標の再作成、および、順ファイルと相対ファイルの索引形式への変換も可能です。64 ビットのアドレスを使用するので、大きいファイルも処理できます。

ソートユーティリティ

ソートユーティリティは、オペレーティングシステムのプロンプトから呼び出すことができるファイルソート機能です。 呼び出し可能ソートモジュールを使用します。

ソースコードコントロールシステムのサポート

SCCS のサポート機能によって、Net Express IDE でプロジェクトを開いている場合に、その IDE のメニューから SCCS にアクセスできます。SCCS は、PVCS、MS Visual SourceSafe、および VisualAge TeamConnection と併用できます。

Win32 にネイティブの API プログラミング

下位レベルな制御を使用したい場合は、Windows オペレーティングシステムのルーチンを呼び出すことができます。

旧製品からの互換性の機能

次の機能は、主に Net Express に移行するための機能です。『*移行ガイド*』では、新しいアプリケーションに対して代替機能を使用するようにお勧めしています。

機能リスト

機能はアルファベット順に説明します。

COBSQL

COBSQL は、リレーショナルデータベースのベンダが提供する、COBOL プリコンパイラで機能するように設計された統合プリプロセッサです。Oracle Pro*COBOL や Sybase Open Client Embedded SQL/COBOL と使用できるように設計されています。すでに上記のプリコンパイラを Micro Focus COBOL の旧製品とともに使用しており、作成済みのアプリケーションを Net Express に移行する場合、または、UNIX にディプロイして Oracle または Sybase のリレーショナルデータベースにアクセスするアプリケーションを開発している場合に、COBSQL を使用することをお勧めします。

それ以外の埋め込み SQL アプリケーションの開発には、OpenESQL を使用することをお勧めします。

Dialog Editor

Dialog Editor は、ウィンドウとダイアログボックスを作成するためのスクリーンペインタです。同じツールが Micro Focus の Visual Object COBOL V1.0 にも含まれています。Visual

Object COBOL V1.0 を使用して作成したアプリケーションを Net Express の Dialog Editor で実行および開発することもできます。

オンラインヘルプシステム

オンラインヘルプシステム (Hyhelp と Ohbld) は、文字モードのオンラインマニュアルを作成して画面表示するための機能です。オンラインヘルプでは、多くの情報を検索できます。最小限のプログラムを追加するのみでオンラインヘルプを呼び出してヘルプを表示できます。文字とグラフィックの両方を使用したヘルプと Windows ネイティブのヘルプを作成できます。

Panels

Panels は、アプリケーションで呼び出して文字モードの画面でウィンドウを使用するための API (Application Programming Interface; アプリケーションプログラミングインターフェイス) です。ウィンドウを重ねて表示したり、テキストと属性を分離または結合したりできます。また、ウィンドウ内でのテキストのスクロール、ポップアップメニューやプルダウンメニューの表示なども可能です。

画面とキーボードのハンドラ (文字モード)

画面とキーボードのハンドラ (ADIS) は、Accept/Display 構文の拡張機能を実行時にサポートするためのモジュールです。データ部の画面節によって全画面文字モードで ACCEPT 文と DISPLAY 文を使用できるようにします (Web ページを処理するための拡張 ACCEPT/DISPLAY 文と混同しないでください)。

画面とキーボードの構成ツール

画面とキーボードの構成ツール (Adiscf と Keybcf) は、文字モードでの画面とキーボードの処理を環境と条件 (他の COBOL システムとの互換性を確保する場合など) に合わせて構成するための 2 つのユーティリティです。

画面節

画面節は、データ部の節です。文字モードの画面で表示するためのフォームを定義します。

ウィンドウサポート

ウィンドウサポートは、文字モードの画面で線やボックスを表示したり、物理的な画面で仮想ウィンドウを作成したりするための COBOL 構文で構成されています。この構文では、基礎となる表示を保存して復元することもできます。ACU COBOL と互換性があります。
