FUJITSU

FUJITSU Software NetCOBOL V12.0

入門ガイド

Windows(64)

B1WD-3462-01Z0(00)
2017年10月

まえがき

本書の目的

本書は、製品の特長や主な機能について説明しています。また、COBOLアプリケーションを開発・運用するための流れを理解できるよう、簡単なサンプルアプリケーションを用いて開発から運用までの操作を説明しています。

製品で提供している開発環境・実行環境の使い方に基づいて説明していますので、本製品を初めて使用される方は本書をご一読ください。

本書の読者

本書は、本製品を初めて使用される方を対象としています。なお、本書を読むためには、以下の知識が必要です。

- COBOLの文法に関する基本的な知識
- 使用するOSに関する基本的な知識

本書の構成

本書は以下の構成になっています。

第1章 NetCOBOLとは

本製品の特長と主な機能、製品体系について説明しています。

第2章 NetCOBOLアプリケーション開発の基礎

簡単なファイル入出力を行うCOBOLアプリケーションを例に、開発環境NetCOBOL Studioの基本操作について説明しています。

第3章 画面帳票アプリケーションの開発

NetCOBOLシリーズのMeFt、FORMによる基本的な画面帳票アプリケーションの作成方法について説明しています。

第4章 MeFt/Webアプリケーションの構築

画面帳票アプリケーションのMeFt/Web化の方法、環境設定などについて説明しています。

第5章 効率のよいプログラムのテクニック

効率のよいCOBOLプログラムの作成テクニックについて説明しています。

第6章 サンプルプログラム

NetCOBOLが提供するサンプルプログラムについて説明しています。

第7章 COBOLファイルアクセスルーチンのサンプルプログラム

NetCOBOLが提供するCOBOLファイルアクセスルーチンのサンプルプログラムについて説明しています。

登録商標について

- Microsoft、Windows、Windows Server、Visual Basic、Visual Studioは、米国Microsoft Corporationの米国およびその他の国における登録商標または商標です。
- その他の会社名または製品名は、それぞれ各社の登録商標または商標です。
- Microsoft Corporationのガイドラインに従って画面写真を使用しています。

製品の呼び名について

本書では、各製品を次のように略記しています。あらかじめご了承ください。

正式名称	略称
Microsoft(R) Windows Server(R) 2016 Datacenter Microsoft(R) Windows Server(R) 2016 Standard Microsoft(R) Windows Server(R) 2016 Essentials	Windows Server 2016
Microsoft(R) Windows Server(R) 2012 R2 Datacenter	Windows Server 2012 R2

正式名称	略称
Microsoft(R) Windows Server(R) 2012 R2 Standard Microsoft(R) Windows Server(R) 2012 R2 Essentials Microsoft(R) Windows Server(R) 2012 R2 Foundation	
Microsoft(R) Windows Server(R) 2012 Datacenter Microsoft(R) Windows Server(R) 2012 Standard Microsoft(R) Windows Server(R) 2012 Essentials Microsoft(R) Windows Server(R) 2012 Foundation	Windows Server 2012
Microsoft(R) Windows Server(R) 2008 R2 Datacenter Microsoft(R) Windows Server(R) 2008 R2 Enterprise Microsoft(R) Windows Server(R) 2008 R2 Standard Microsoft(R) Windows Server(R) 2008 R2 Foundation	Windows Server 2008 R2
Windows(R) 10 Home Windows(R) 10 Pro Windows(R) 10 Enterprise Windows(R) 10 Education	Windows 10 または Windows 10(x64)
Windows(R) 8.1 Windows(R) 8.1 Pro Windows(R) 8.1 Enterprise	Windows 8.1 または Windows 8.1(x64)
Windows(R) 7 Home Premium Windows(R) 7 Professional Windows(R) 7 Enterprise Windows(R) 7 Ultimate	Windows 7 または Windows 7(x64)
Red Hat(R) Enterprise Linux(R) 6 (for Intel64) Red Hat(R) Enterprise Linux(R) 7 (for Intel64)	Linux または Linux(64)
Microsoft(R) Visual C++(R) development system	Visual C++
Microsoft(R) Visual Basic(R) programming system	Visual Basic
PowerSORT PowerSORT Server PowerSORT Workstation	PowerSORT

以下をすべて指す場合は、「Windows(x64)」または「Windows」と表記します。

- Windows Server 2016
- Windows Server 2012 R2
- Windows Server 2012
- Windows Server 2008 R2
- Windows 10(x64)
- Windows 8.1(x64)
- Windows 7(x64)

注意事項

- 本書に記載されている画面は、使用されているシステムにより異なる場合がありますので注意してください。
- 本書では、“COBOL文法書”で“原始プログラム”と記述されている用語を“ソースプログラム”と記述しています。
- 本書では、Windows 10環境で操作手順を説明しています。

お願い

- 本書を無断で他に転載しないようお願いします。
- 本書は予告なしに変更されることがあります。

輸出管理について

本ドキュメントを輸出または第三者へ提供する場合は、お客様が居住する国および米国輸出管理関連法規等の規制をご確認のうえ、必要な手続きをおとりください。

2017年10月

目　次

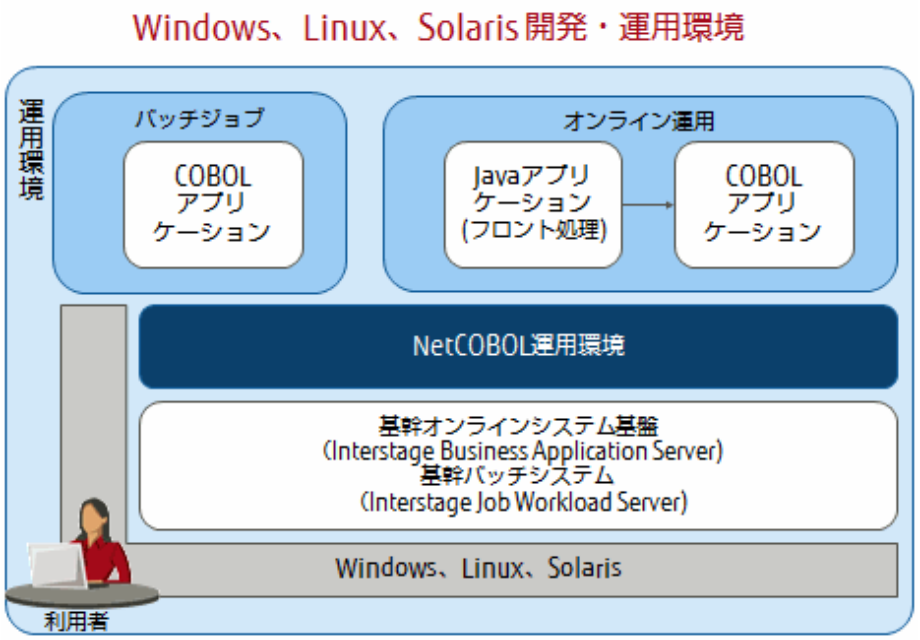
第1章 NetCOBOLとは

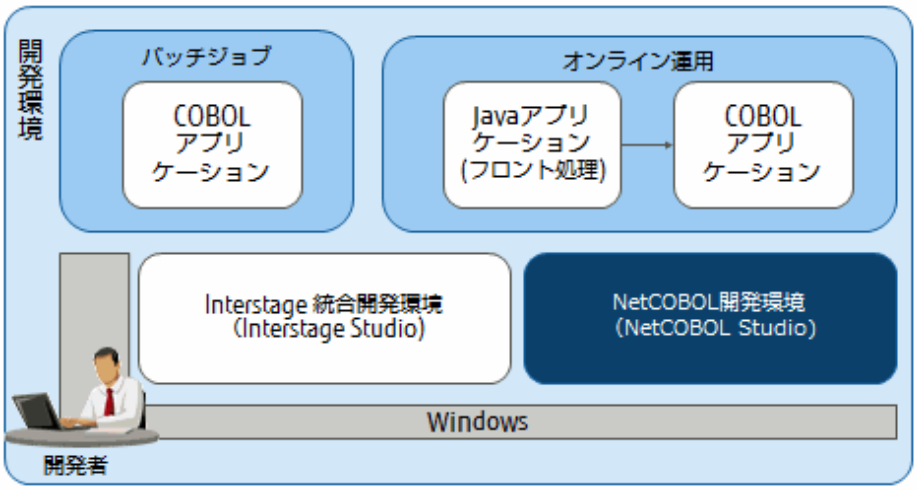
COBOL言語は、1960年に誕生して以来、ビジネスロジックの記述性や互換性などに優れている点が評価され、多くのビジネスシステムに使用され続けています。一方、ITの進展に伴い、ビジネスシステムの要件が急速に高度化・多様化しています。

COBOLは、その長い歴史において、常に最新テクノロジーや時代の要求に応えて進化してきました。

お客様の既存COBOL資産を活かし、長期に渡り安定してビジネスの成長を支援するのが「NetCOBOL」です。

クラウドやビッグデータ活用を支える富士通のソフトウェア製品と組み合わせることで、お客様のCOBOL資産の価値をさらに高めます。





ここでは、NetCOBOLシリーズの概要について述べています。ご利用のシステムによってはサポートされない機能や連携製品が含まれます。ソフトウェア説明書をご確認ください。

1.1 NetCOBOLの特長

COBOL（COmmon Business Oriented Language）は、事務処理向けに開発されたプログラム言語です。10進演算、ファイル処理、帳票作成などの事務処理に特化した仕様と英語表現に似た文法で、読み易く分かり易いプログラム記述が可能です。

NetCOBOLは、COBOLの特長を活かし、最新テクノロジー・最新環境に対応したオープンプラットフォームのCOBOL開発環境です。

NetCOBOLには、以下の特長があります。

COBOL資産を長期間、安心して利用

国際規格、業界標準仕様に対応し、上位との互換性も保証しています。将来も安心できる基幹システム運用と拡張が可能です。メインフレームやオフコンの既存COBOL資産と開発者のスキル、ノウハウも活用できます。

効率的で高生産なプログラム開発

COBOL統合開発環境により、設計・プログラム・テスト・保守まで、開発プロセス全体を効率化できます。バッチ、Webアプリケーションからクラウドアプリケーションまで、最新技術と連携したプログラム開発が可能です。

基幹システムの適用範囲拡大

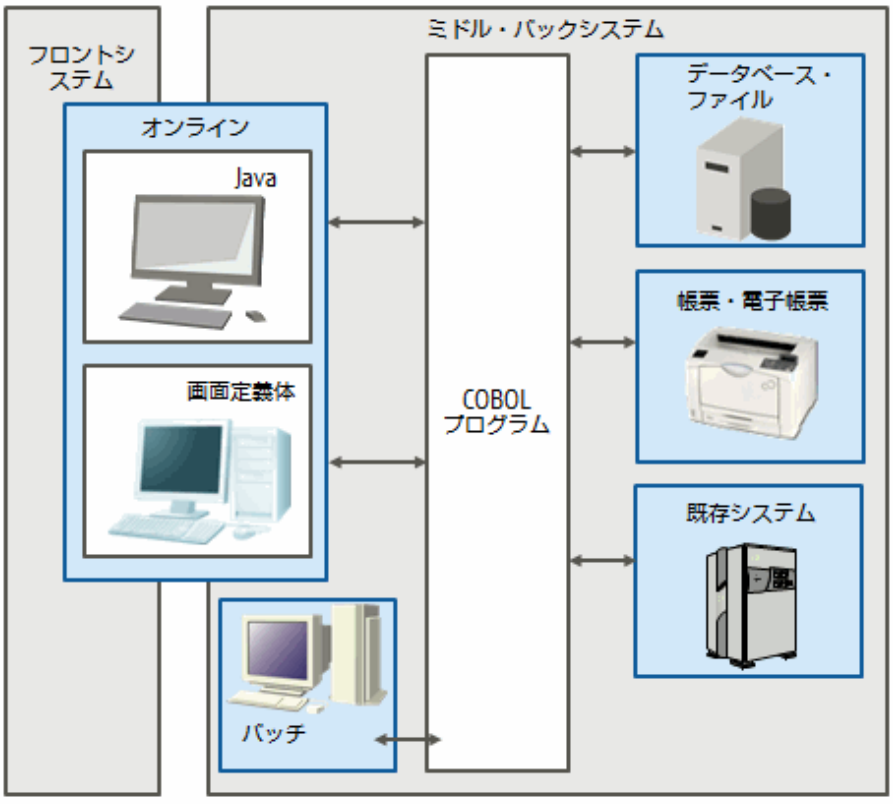
.NET、Java、クラウド連携で、基幹システムの適用範囲を拡大できます。特に基幹システムを支える富士通のソフトウェア「Interstage」との連携により、堅牢で柔軟性の高い基幹システムを構築可能です。

高い実績と安心サポート

メインフレーム、オフコンのCOBOLは50年、オープンプラットフォームのCOBOLは20年の実績を持ち、富士通の支援サービス「SupportDesk」と全国各地の営業拠点のサポート体制により、安心して利用できます。

1.2 NetCOBOLと先端技術

NetCOBOLは、COBOLの国際規格をベースに、各種RDB、画面・帳票、最新インターネット技術を組み合わせたCOBOLアプリケーションを作成することができます。NetCOBOLなら、変化の激しい今日のビジネス環境で、お客様の経営を支える基幹システムの信頼性、安定性、効率化を追及したシステム構築が可能です。



オンライン

Web技術、柔軟性の高いJavaでフロントシステムを利用し、ビジネスロジックの生産性、実行性能の高いCOBOLを、ミドル・バックシステムに利用することで、言語特性を利用し、拡張性の高い基幹システム構築が可能です。フロント、ミドル・バックシステムの間に、富士通のアプリケーションサーバ「Interstage Application Server」を配置することにより、堅牢なトランザクションシステムを構築できます。更に、「Interstage Business Application Server」を利用することで、高度な制御ロジックを実現できます。

バッチ

膨大なビジネスデータを扱うバッチ業務は、基幹システムを支える基盤です。メインフレーム・オフコンの富士通COBOLから継承された、NetCOBOLの高い実行性能、信頼性は、オープンシステムのバッチ業務に最適です。さらにバッチ処理基盤「Interstage Job Workload Server」を組み合わせることで、バッチ処理の安定稼動と運用性向上を実現することができます。また、総合運用管理「Systemwalker」との連携で、サーバの起動、終了からバッチ処理の起動、エラーリカバリーまで、24時間365日、トータルなバッチ運用を実現できます。

また、Linux(64)版 NetCOBOL Enterprise Editionでは、Apache Hadoopおよび当社の並列分散処理ソフトウェア「Interstage Big Data Parallel Processing Server」と連携し、並列分散処理によるCOBOLバッチ処理の高速化を実現できます。

データベース・ファイル

基幹システムの中核となるのは、データ集計・分析です。ビジネスの拡大に伴い、取り扱うデータも飛躍的に増加しています。NetCOBOLと各種RDBを組み合わせることにより、データの集計・分析をより早く実現できます。さらに「PowerSORT」を導入することにより、データのソート・マージが高速化され、基幹システムのパフォーマンスを向上できます。

画面定義体

入力データチェック、ファンクションキー、カーソル移動など、基幹システムの画面に求められる操作が可能です。

Windows版 NetCOBOLでは、この画面定義体をそのまま利用し、Web運用することも可能です。

メインフレーム・オフコンなど従来システムと同様の画面操作を、オープンシステムで実現できるため、システムをご利用になるお客様の教育も必要ありません。画面は、NetCOBOLの専用ツール「FORM」で作成、COBOLプログラムからは、READ文/WRITE文を利用することで、COBOLのノウハウで画面操作性の優れた基幹システムを構築できます。

帳票・電子帳票

基幹システムに求められる、きめ細かな帳票出力を実現できます。豊富な文字、罫線、図形により、きれいで見やすい帳票が作成できます。帳票製品「Interstage List Works」と連携することで、COBOLプログラムからの帳票出力をそのまま電子化できます。これにより、業務システムのコスト削減、帳票情報の共有が図れます。帳票はNetCOBOLの専用ツール「FORM」または「PowerFORM」で作成、COBOLプログラムからの帳票出力は、WRITE文を利用し、COBOLのレコード出力イメージで帳票出力が可能です。

既存システム

メインフレーム・オフコンの基幹システムをご利用ならば、オープンシステムをアドオンし、トータルなシステム構築も可能です。メインフレーム・オフコンは、長期間、高可用性が求められる業務に最適であり、オープンシステムは、即時性、定期的に見直しが可能な業務に最適です。メインフレームとは、富士通のアプリケーションサーバ「Interstage」によるアプリケーション間の連携が可能です。オフコンとは、データベースの複製・共用管理「PowerReplication」を利用したデータ連携が可能です。

1.3 製品体系

開発・運用環境製品は、複数のコンポーネントで構成されています。インストール時にカスタムインストールを選択して、必要なコンポーネントだけをインストールすることもできます。

インストールの詳細は、“インストールガイド”を参照してください。

以下の表の記号の意味は以下のとおりです。

EE ： Enterprise Edition

SE ： Standard Edition

BE ： Base Edition

○ ： 製品に同梱されるコンポーネント

× ： 製品に同梱されないコンポーネント

表1.1 開発パッケージ

コンポーネント名	機能名	EE	SE	BE
NetCOBOL	COBOL開発環境（NetCOBOL Studio)	○	○	○
	COBOLコンパイラ	○	○	○
	COBOLランタイム	○	○	○
	診断機能	○	○	○

コンポーネント名	機能名	EE	SE	BE
Jアダプタクラスジェネレータ	Java連携(注3)	○	○	×
FORM	帳票設計支援	○	○	×
MeFt	帳票の運用	○	○	×
MeFt/Web	Webアプリケーションの構築支援	○(注2)	○(注2)	×
MeFt/Web HTML変換方式	画面定義体のWebコンテンツ(HTML)変換	○	○	×
Migration CJC for INTARFRM 連携機能 (注1)	COBOL-Java(Servlet/JSP)移行支援	○	○	×
SIMPLIA/COBOL支援キット	テストデータ作成支援	○	×	×
	開発資産流用支援	○	×	×
	プログラム開発ステップ計測支援	○	×	×
	テスト効率化支援	○	×	×
富士通メインフレーム浮動小数点演算エミュレータ	富士通メインフレーム形式の浮動小数点データ	○	×	×
PowerSORT Server	高性能データ・ソートマージ	○	×	×

表1.2 運用パッケージ

コンポーネント名	機能名	サーバ運用			クライアント運用	
		EE	SE	BE	SE	BE
NetCOBOL	COBOLランタイム	○	○	○	○	○
	診断機能	○	○	○	○	○
Jアダプタクラスジェネレータ	Java連携(注3)	○	○	×	×	×
MeFt	帳票の運用	○	○	×	○	×
MeFt/Web	Webアプリケーションの構築支援	○(注2)	○(注2)	×	×	×
MeFt/Web HTML変換方式	画面定義体をWebコンテンツ(HTML)変換	○	○	×	×	×
Migration CJC for INTARFRM 連携機能 (注1)	COBOL-Java(Servlet/JSP)移行支援	○	○	×	×	×
富士通メインフレーム浮動小数点演算エミュレータ	富士通メインフレーム形式の浮動小数点データ	○	×	×	×	×
PowerSORT Server	高性能データ・ソートマージ	○	×	×	×	×

注1 : Migration CJC for INTARFRMサービスの契約が必要です。

注2 : MeFt/Webアプリケーションの利用時には、以下のソフトウェアが必須です。

- サーバ側

  IIS（Microsoft Internet Information Server）

・ クライアント側

Microsoft Internet Explorer（32bit版）

注3 : Windows(x64)版の富士通製JDK/JDKまたはOracle製JDK/JREが別途必要です。

必須ソフトウェアの詳細は、“NetCOBOL ソフトウェア説明書”を参照してください。

第2章 NetCOBOLアプリケーション開発の基礎

本章では、NetCOBOLが提供する開発環境の機能を説明するとともに、簡単なアプリケーションの作成を通じて開発環境の操作を説明します。

2.1 概要

NetCOBOLにてアプリケーション開発を行うときに使用する開発環境、および本章で作成するアプリケーションの概要について説明します。

2.1.1 NetCOBOLの開発環境

NetCOBOLでは、次のような開発環境を提供しています。

NetCOBOL Studio

NetCOBOL Studioは、オープンの世界でスタンダードなEclipse（エクリプス）をベースとしたCOBOL開発環境です。

Eclipseは、いろいろなツールをプラグインで追加していくことができるオープンソースの統合開発環境（IDE）です。

NetCOBOL Studioでは、COBOLアプリケーションの開発に必要な各種操作(プログラムの編集・翻訳・リンク・実行・デバッグ)を行うための操作ビューを持ち、プログラムの作成からデバッグまで一連の作業をサポートします。

Eclipse

Eclipse自体は部品を入れる箱のようなもので、様々な部品を追加する（プラグインする）ことで拡張可能な機構を持つ開発環境ツールのプラットフォームです。Eclipseの基本セットは、ワークベンチやワークスペースなどの部品から構成されています。これらの部品はEclipseのランタイムエンジンの上に乗っています。

NetCOBOLでは、EclipseにCOBOLの機能をプラグインしてCOBOL統合開発環境NetCOBOL Studioを提供しています。

ワークベンチ

「ワークベンチ」は、開発環境のユーザインターフェースのことで、NetCOBOL Studioを起動した際に表示される画面そのものを指します。エディタやビュー、メニューなどいろいろなGUI部品を管理します。

ワークスペース

「ワークスペース」は後述する格納場所や依存関係などプロジェクトの情報を管理します。

プロジェクト

「プロジェクト」には、以下の種類があります。

- COBOLソリューションプロジェクト

  複数のプロジェクト(COBOLプロジェクト、COBOLリソースプロジェクト）をまとめて管理する場合に使用します。共通オプションの設定やプロジェクトに対しての一括操作ができます。

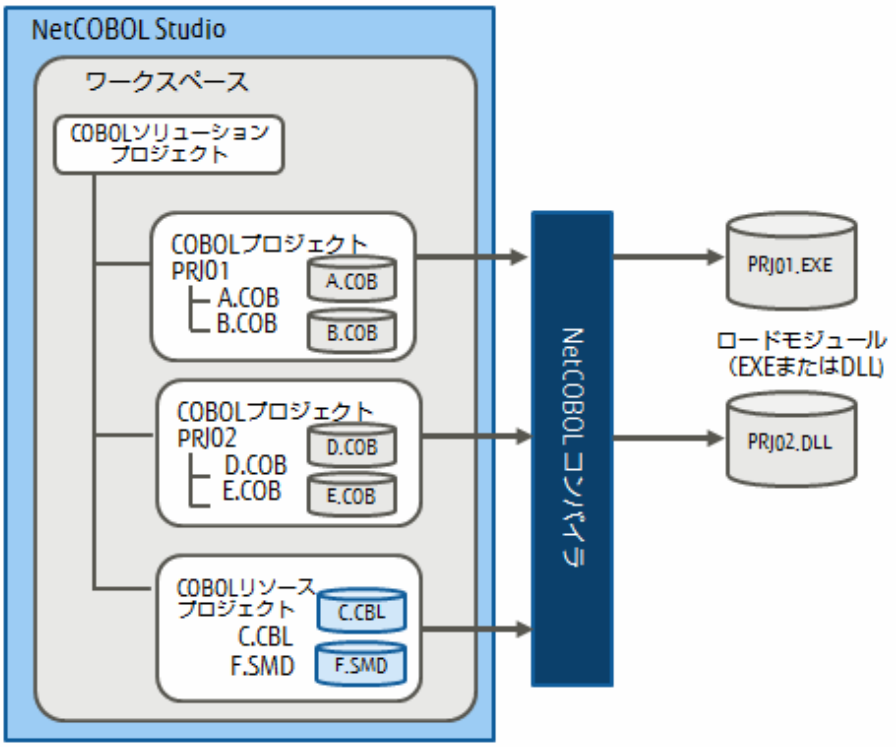
- COBOLプロジェクト

  COBOLアプリケーションを作成するために使用します。プロジェクトの作成単位は、ロードモジュール（EXEやDLL）単位になります。

- COBOLリソースプロジェクト

  COBOL資産(登録集ファイルや定義体ファイル)の保管庫として利用します。

NetCOBOL Studio上では、プロジェクト名をトップレベルにCOBOLソースプログラムや登録集がツリー構造で表示されます。



パースペクティブ

NetCOBOL Studioの画面は、複数の情報表示ビューから構成されます。このような情報表示ビューの組み合わせ(レイアウト)は「パースペクティブ」といいます。

COBOLプログラムの開発に最適な情報表示ビューの組み合わせを「COBOLパースペクティブ」といい、プログラムデバッグに最適な情報ビューの組み合わせを「デバッグパースペクティブ」といいます。作業内容に合わせ、それぞれのパースペクティブを利用することで、COBOLプログラムを効率的に開発・デバッグすることが可能です。

「COBOLパースペクティブ」と「デバッグパースペクティブ」の詳細は、“NetCOBOL Studio ユーザーズガイド”を参照してください。

本書では、NetCOBOL Studioを使用したCOBOLアプリケーションの開発方法を紹介します。

2.1.2 作成するアプリケーションについて

本章では、NetCOBOLに同梱されているサンプルの中で、行順ファイルと索引ファイルの操作を行っているサンプルを使用し、アプリケーションを作成します。

アプリケーションの概要

エディタを使って作成したデータファイル(行順ファイル)を読み込み、マスタファイル(索引ファイル)を作成するアプリケーションです。索引ファイルのレコード定義は、登録集として、COBOLのCOPY文を使って翻訳時にCOBOLプログラムに取り込みます。

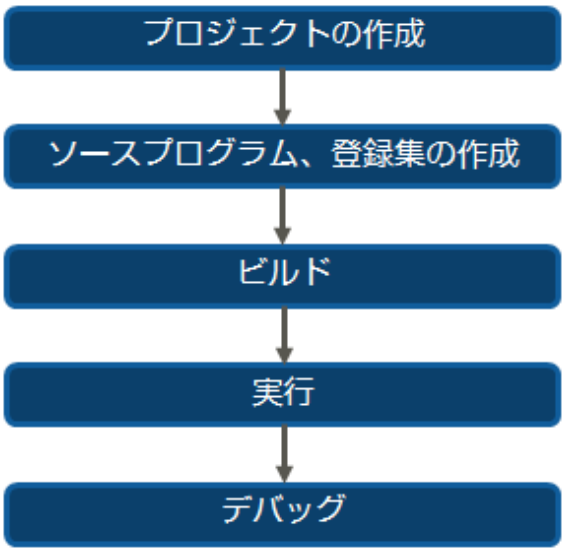
プログラムの格納場所

説明に使用するサンプルは、NetCOBOLのインストールフォルダーの「SAMPLES¥COBOL」フォルダー配下の「SAMPLE02」に格納されています。

2.1.3 アプリケーション開発の流れ

本章で説明するアプリケーションの開発の流れを次に示します。

なお、ソースプログラムおよび登録集はサンプルで提供されているものを参考に作成します。



2.2 NetCOBOL Studioの起動

1. [スタート] > お使いのNetCOBOL 製品名 > [NetCOBOL Studio(x64) - Eclipse 4.6基盤]を選択して、NetCOBOL Studioを起動します。
2. NetCOBOL Studioの起動画面が表示されたら、[起動]ボタンをクリックします。

参考

[環境設定]ボタンをクリックして表示される[動作環境の設定]ダイアログボックスからワークスペースフォルダーを切り替えることができます。

ワークスペースフォルダーのデフォルトは、以下です。

```
マイ ドキュメントフォルダー¥NetCOBOL Studio V12.0.0(x64)¥workspace4.6
```

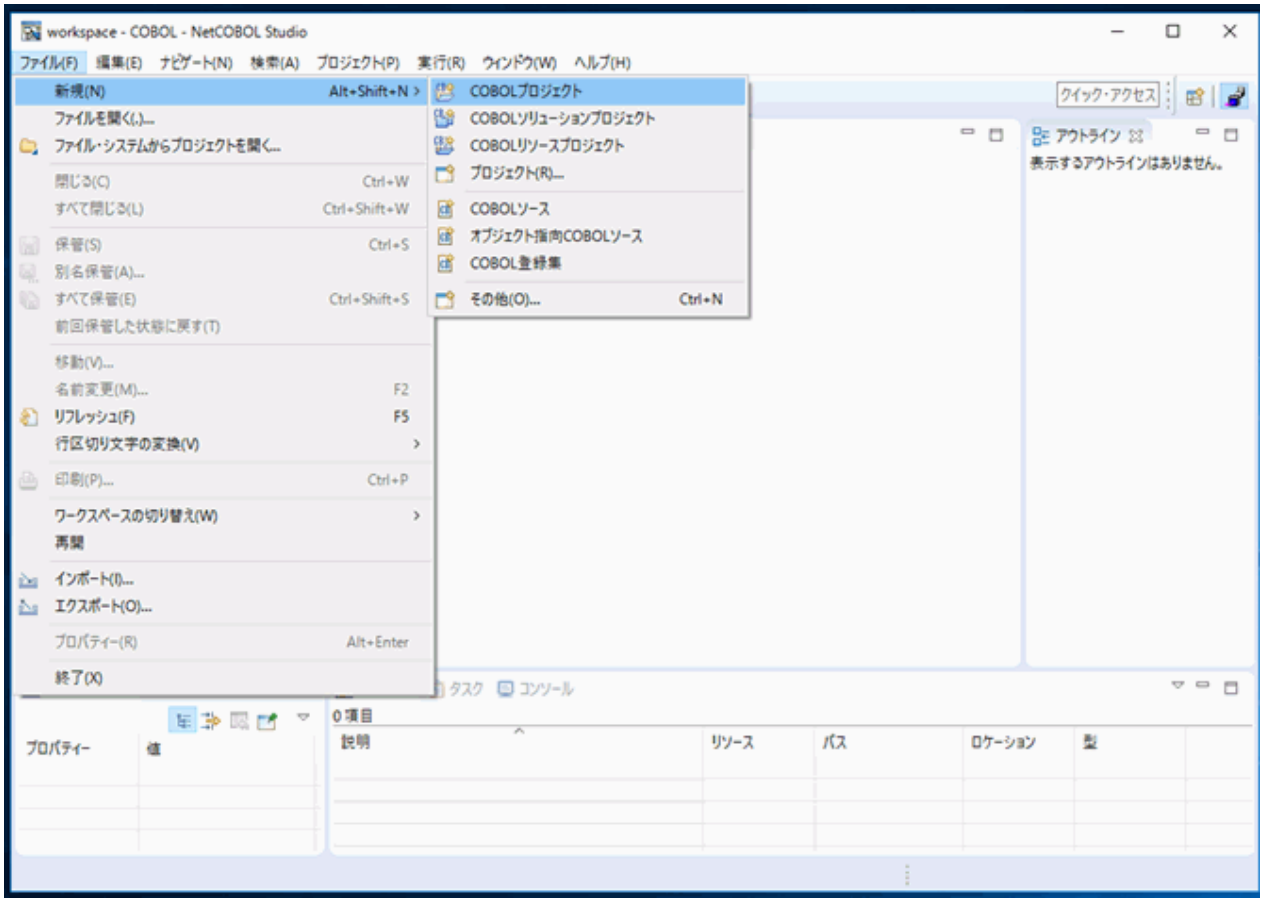
Windowsシステムには、各ユーザのデータやファイルを保存するための[マイドキュメント]フォルダーが用意されています。[マイドキュメント]フォルダーの実体は、Windowsシステムによって異なります。

3. 「ようこそ」の[ワークベンチ]アイコンをクリックします。

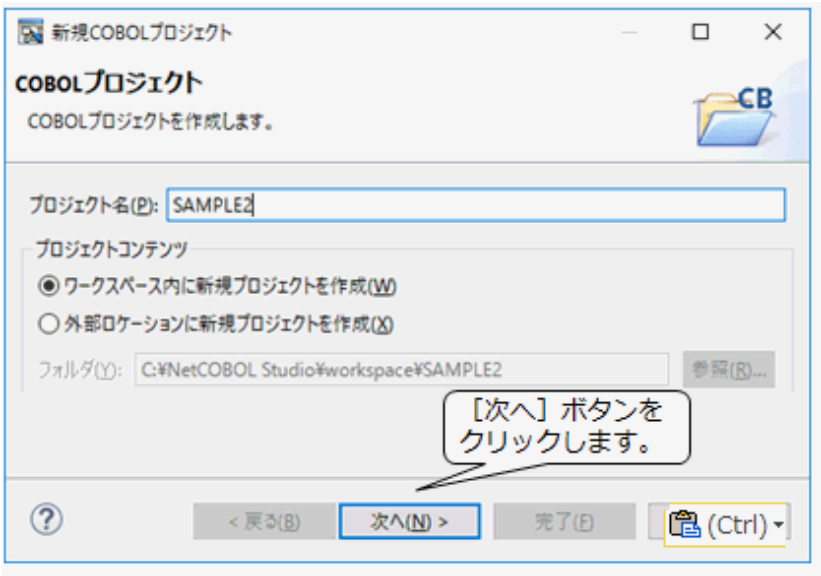
2.3 プロジェクトの作成

プロジェクトの作成からCOBOLソースのテンプレート作成まで、ウィザードの指示に従うことで簡単に作成できます。

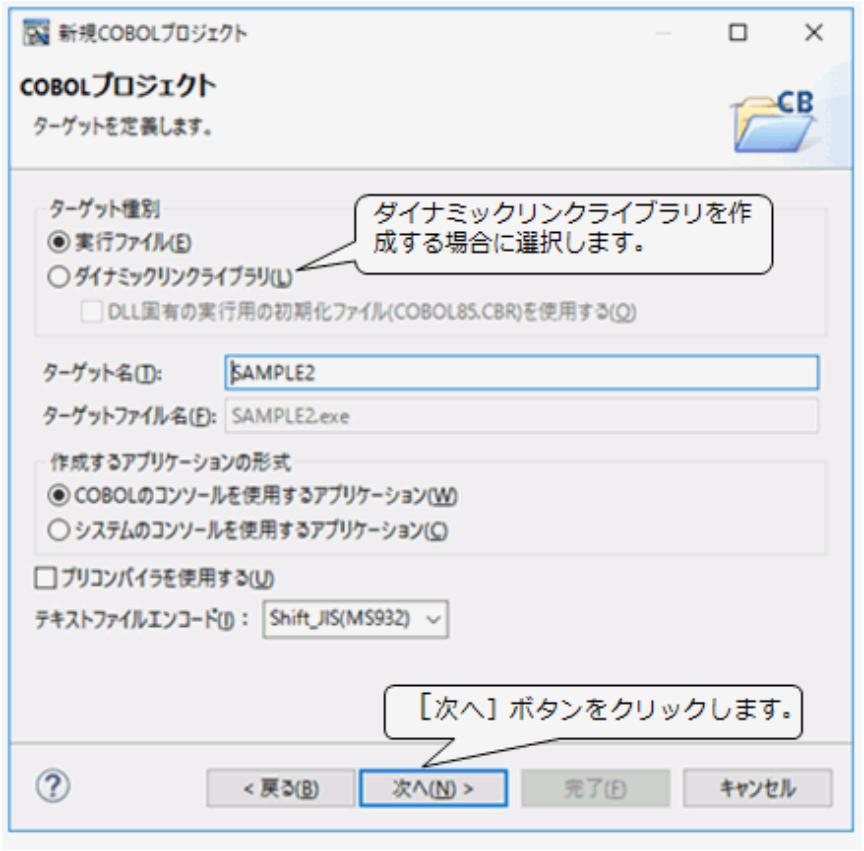
1. ［ファイル］メニューから［新規］>［COBOLプロジェクト］を選択します。



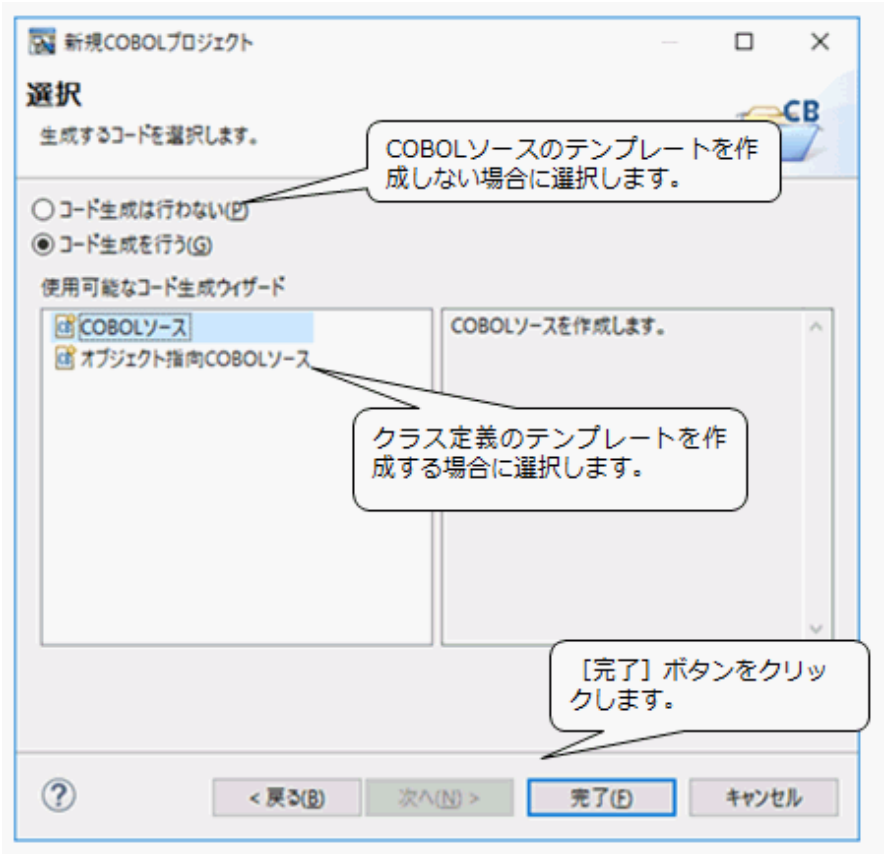
2. ［新規COBOLプロジェクト］ダイアログボックスの［プロジェクト名］に「SAMPLE2」と入力し、［次へ］ボタンをクリックします。



3. ［ターゲット種別］を「実行ファイル」とし、［ターゲット名］に「SAMPLE2」を入力します。



4. ［コード生成を行う］をチェックし、「COBOLソース」を選択し、[完了]ボタンをクリックします。



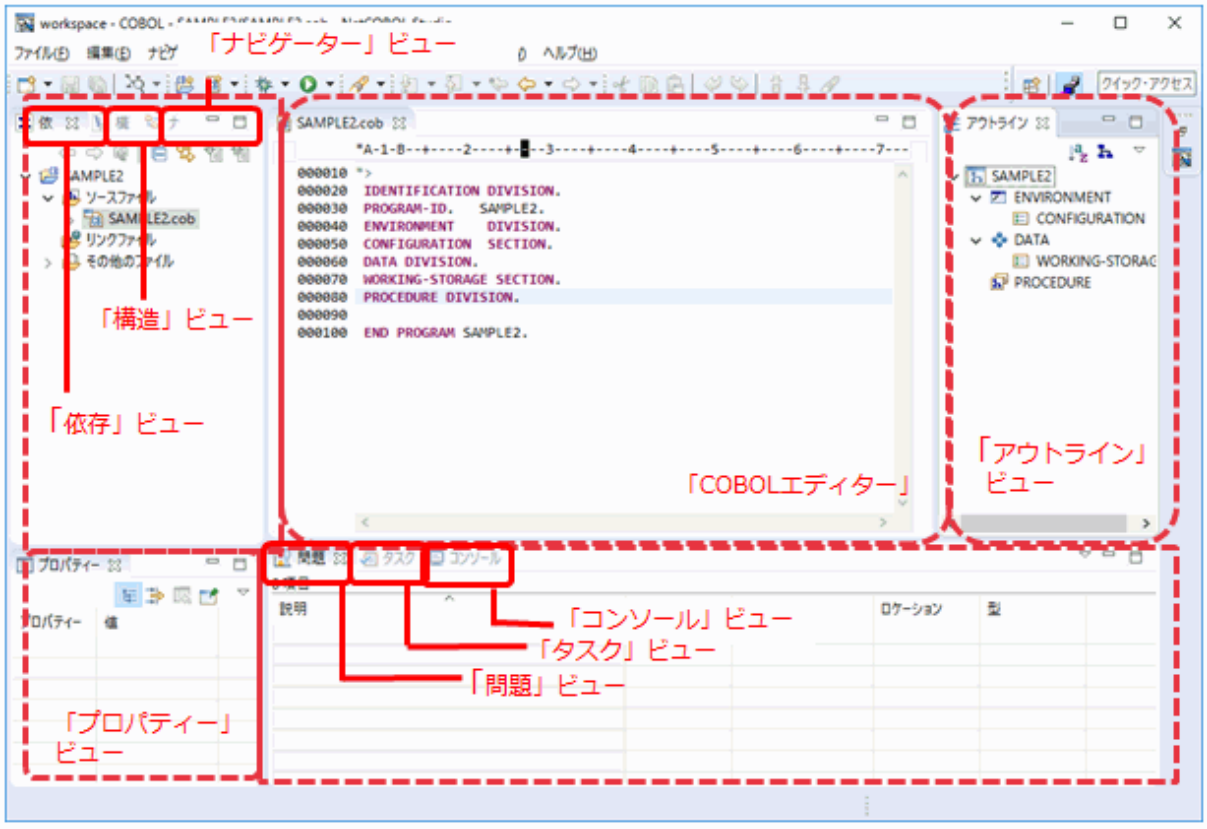
5. [関連付けられたパースペクティブを開きますか？]ダイアログボックスが表示された場合、[はい]ボタンをクリックします。

参考

COBOLプロジェクトはCOBOL開発用の「COBOLパースペクティブ」と関連付けられます。

6. COBOLソースを作成します。[ファイル名]に「SAMPLE2」と入力します。COBOLソースのプログラム名を決める[PROGRAM-ID]には自動的にファイル名と同じ文字列が入ります。異なる文字列にする場合、[PROGRAM-ID]を変更します。[ファイルコメント]には必要に応じてCOBOLソースの先頭に挿入するコメントを記述します。

7. 新規に作成されたプロジェクトが、以下のような「COBOLパースペクティブ」で表示されます。



各ビューの概要を説明します。

- 「依存」ビュー

  翻訳するCOBOLファイルと依存関係にある登録集や定義体などのファイルをツリー構造で表示します。

- 「構造」ビュー

  PROGRAM-IDや環境部、データ部などプログラムの内部構造をツリー構造で表示します。

- 「ナビゲーター」ビュー

  プロジェクト内に存在する全てのファイルを表示します。

- 「アウトライン」ビュー

  エディタに表示されている、PROGRAM-IDや環境部、データ部などのCOBOLソースの構造を表示します。

- 「プロパティー」ビュー

  プロジェクト内リソースのプロパティーを表示します。

- 「問題」ビュー

  翻訳エラーメッセージや警告情報など、翻訳時に発生した問題を表示します。

- 「タスク」ビュー

  後で検討する項目などをタスクとして記録しておく場合に使用します。

- 「コンソール」ビュー

  コンソールビューのツールバーから[コンソールを開く]を選択し、「ビルドコンソール」を選択することにより、プロジェクトのビルド結果を表示します。

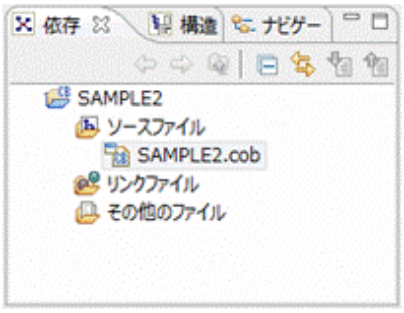
2.4 ソースプログラム、登録集の作成

実行ファイルの作成に必要なファイルをプロジェクトに登録します。作成するファイルとして、COBOLソースプログラムや登録集、画面帳票定義体などがあります。

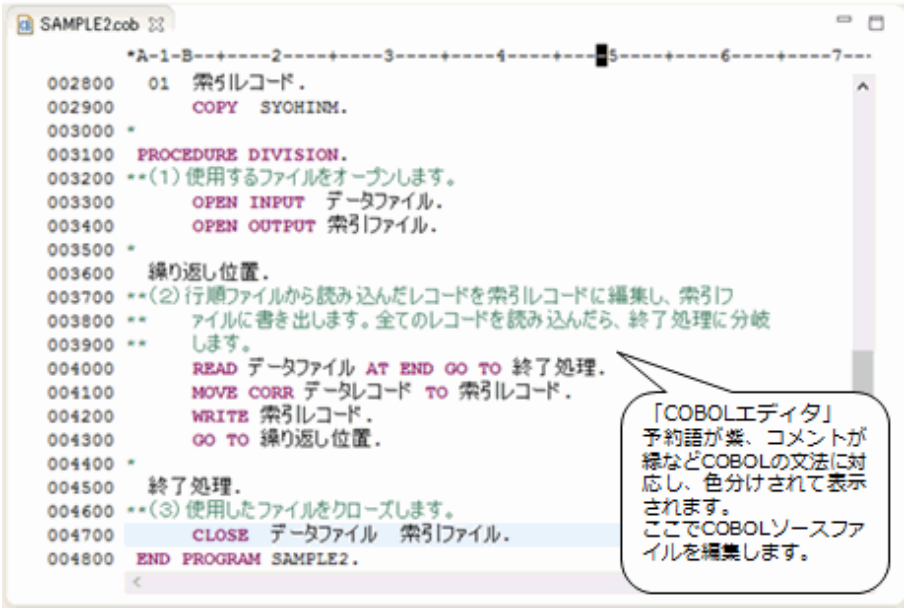
本章で作成するアプリケーションでは、COBOLソースプログラムと登録集を作成します。なお、作成するCOBOLソースプログラムと登録集の内容は、サンプルプログラムで提供されているものを参考にしてください。

COBOLソースプログラムの作成

1. メインプログラムとなるCOBOLソースプログラム「SAMPLE2.cob」はここまでの手順で既に登録されています。



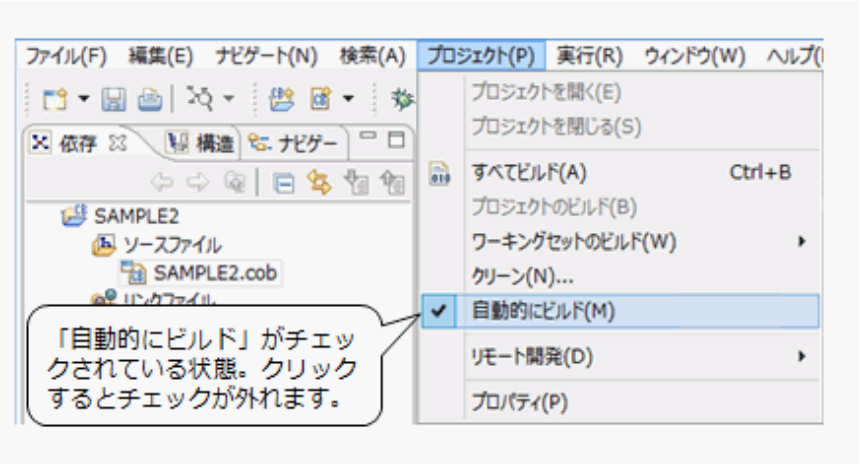
2. サンプルプログラムを参考に、エディタからCOBOLソースプログラムを編集します。



3. 編集後、[ファイル]メニューの[保存]または[Ctrl]+[S]キーを選択し、「SAMPLE2.cob」を保存します。

注意

注)[プロジェクト]メニューの「自動的にビルド」がチェックされている場合、COBOLソースや登録集などの保存のタイミングで自動的にビルドが実行されます。このとき、ビルドに必要な設定が済んでいないと、ビルドエラーとなることがあります。ここでは、説明のため、「自動的にビルド」のチェックを外してあります。



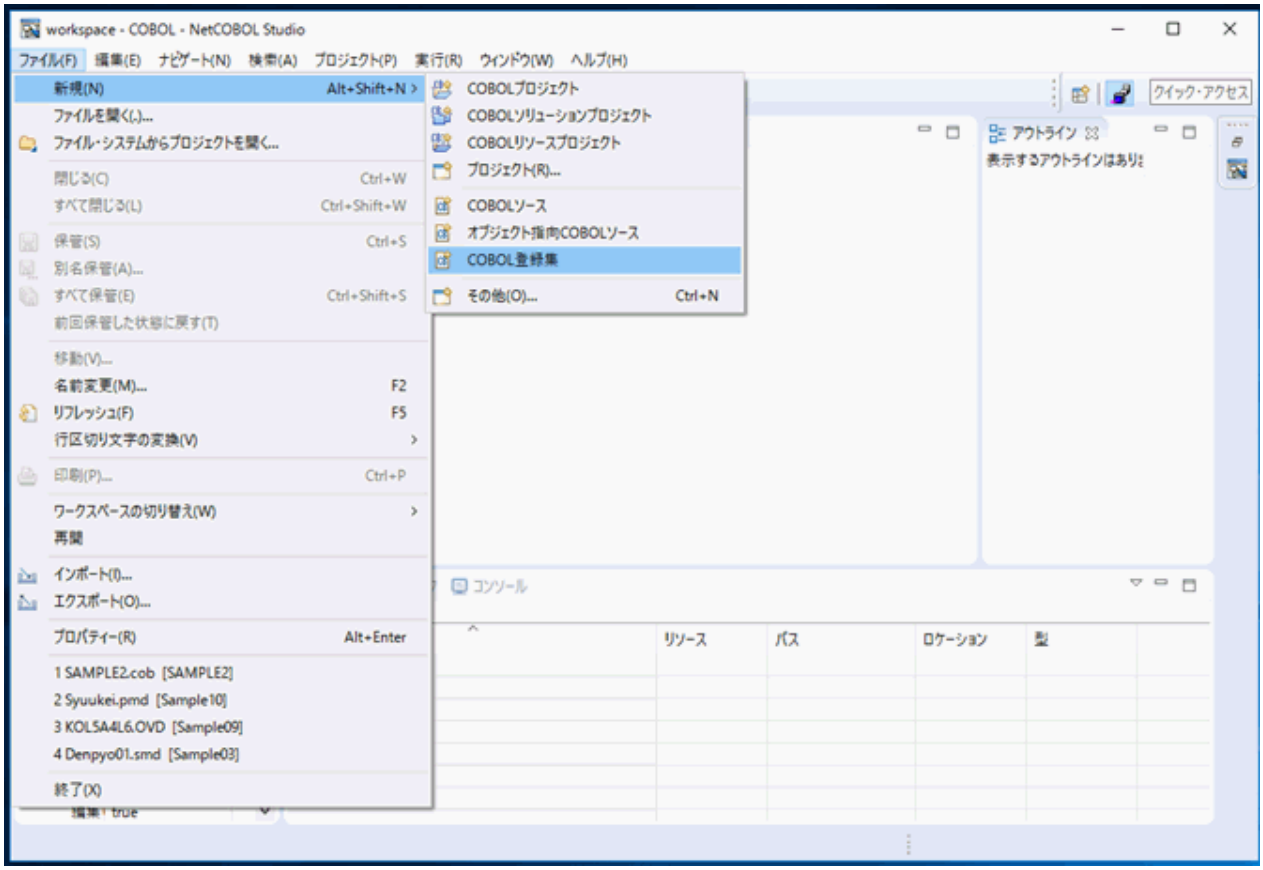
参考

- 他のCOBOLソースプログラムを新規にプロジェクトに追加するには、次の手順で行います。
  1. プロジェクトファイルに登録されているフォルダー「ソースファイル」を右クリックします。
  2. [新規]を選択し、作成するソースに合わせて[COBOLソース]または[オブジェクト指向COBOLソース]を選択します。
  3. ［COBOLソース生成ウィザード］に従って、COBOLソースを作成します。
- 既存のCOBOLソースプログラムをプロジェクトに追加するには、次の手順で行います。
  1. エクスプローラを使って既存のCOBOLソースプログラムをドラッグします。
  2. NetCOBOL Studioの[依存]ビュー（または[構造]ビュー、[ナビゲーター]ビュー）の追加するフォルダー上にドロップします。
  3. 既存のCOBOLソースプログラムがプロジェクト内に追加され、既存のCOBOLソースプログラムは物理的にプロジェクト内にコピーされます。

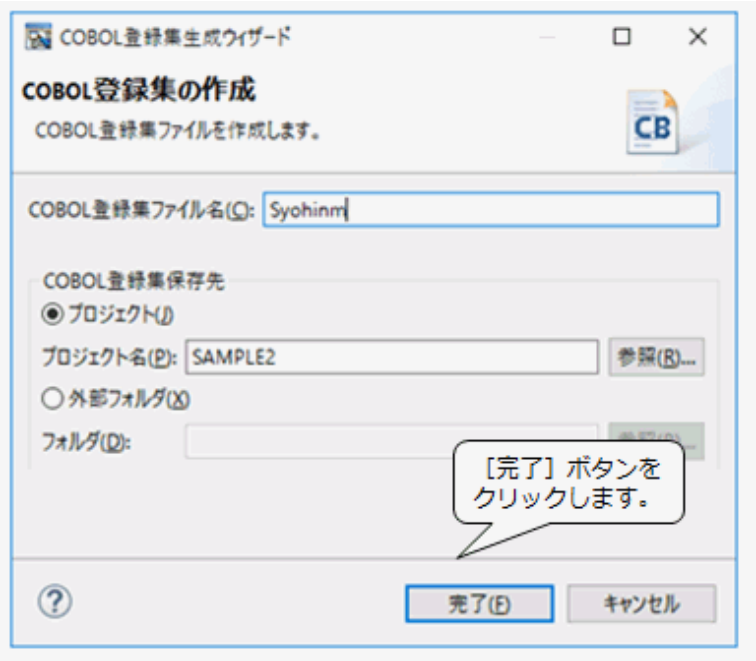
登録集の作成

COBOLソースプログラムが利用する登録集をプロジェクトに登録し、COBOLソースプログラムとの依存関係を確定します。

1. ［ファイル］メニューから［新規］>［COBOL登録集］を選択します。

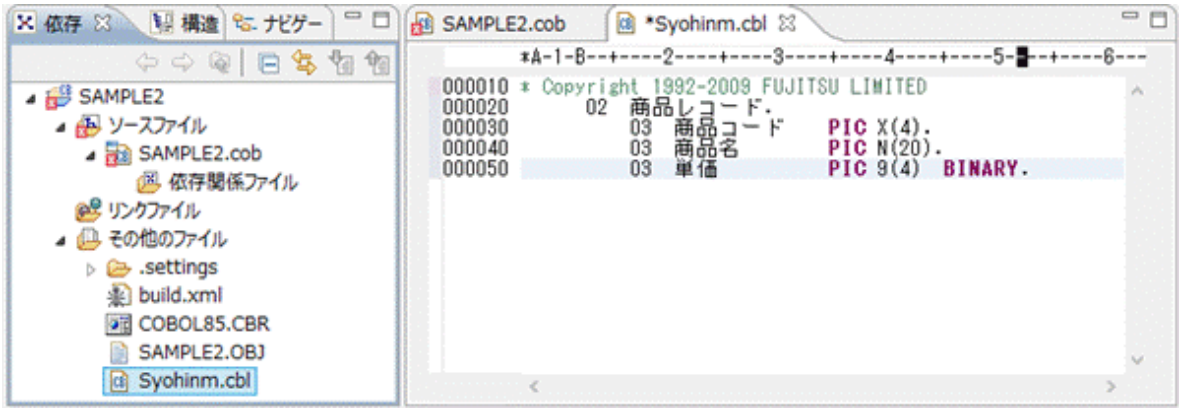


2. ［COBOL登録集ファイル名］に「Syohinm」を入力し、[完了]ボタンをクリックします。

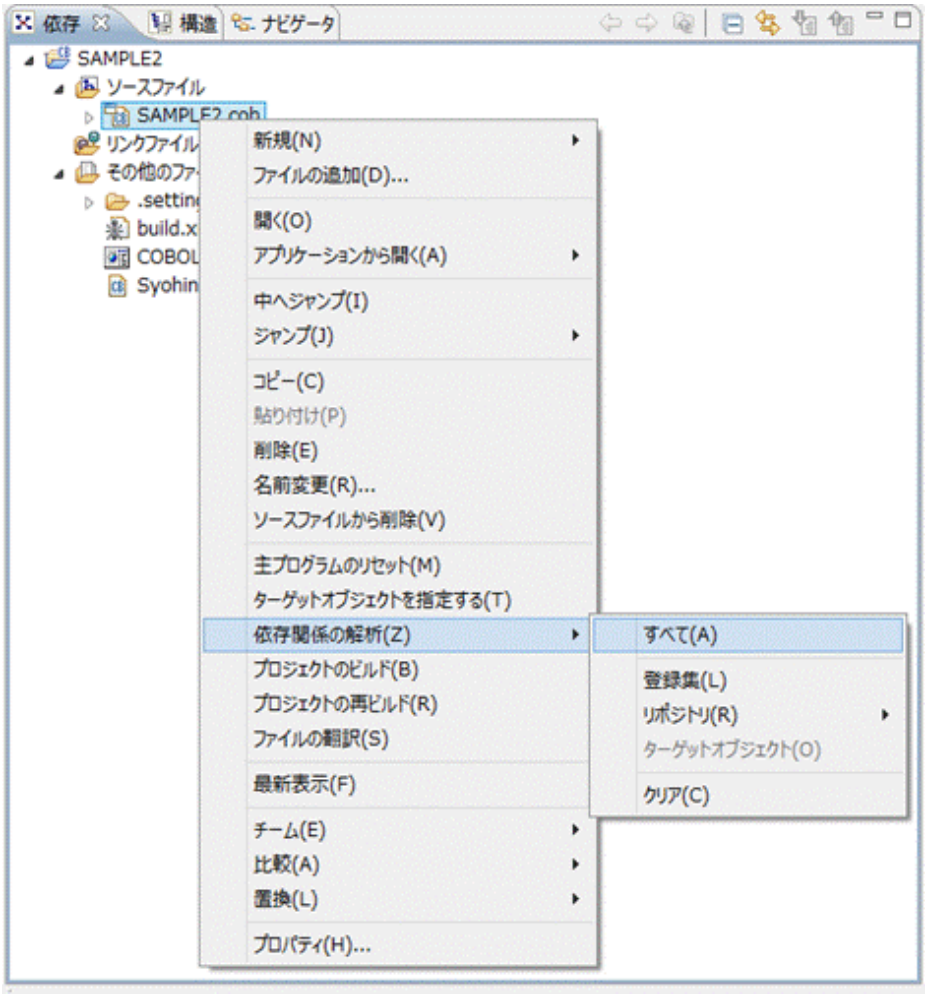


3. 登録集がプロジェクトに追加されます。

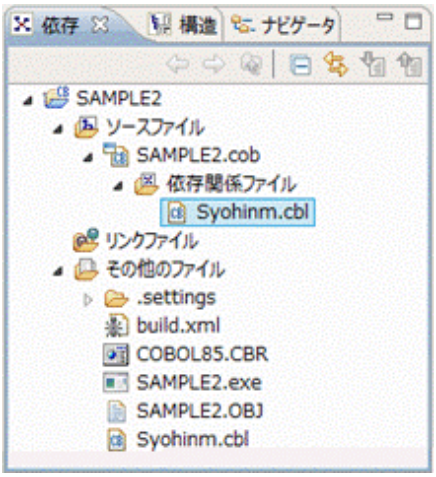
4. COBOLソースプログラムと同様、サンプルプログラムを参考にエディタを使って登録集を編集し、保存します。



5. COBOLソースと登録集を関連付けるための依存関係解析を行います。ソースファイル「SAMPLE2.cob」を右クリックし、[依存関係の解析]から「すべて」を選択します。



6. ［依存関係ファイル］フォルダーに「Syohinm.cbl」が追加されます。



参考

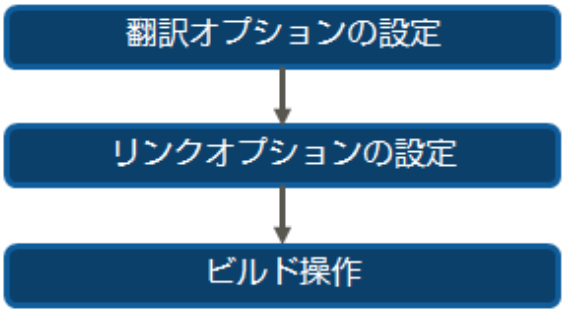
既存の登録集は、ワークスペースにあれば、自動的に[その他のファイル]フォルダーに表示されます。

ワークスペース外に登録集フォルダーがある場合は、後述する翻訳オプションLIBで指定してください。なお、ワークスペース外のファイルはNetCOBOL Studioから操作（編集・参照）することはできません。

2.5 ビルド

プロジェクトを作成してから、ビルドします。「ビルド」とは、1回の指示で翻訳およびリンクを行い、実行ファイルを作成することです。ビルドでは、COBOLソースプログラムなどの翻訳・リンクに必要なファイルのタイムスタンプを管理し、変更があったファイルのみを翻訳・リンクの対象とします。これに対して、変更の有無に係わらず、再翻訳および再リンクを行って実行ファイルを再作成することを「再ビルド」といいます。

ビルドの流れを次に示します。



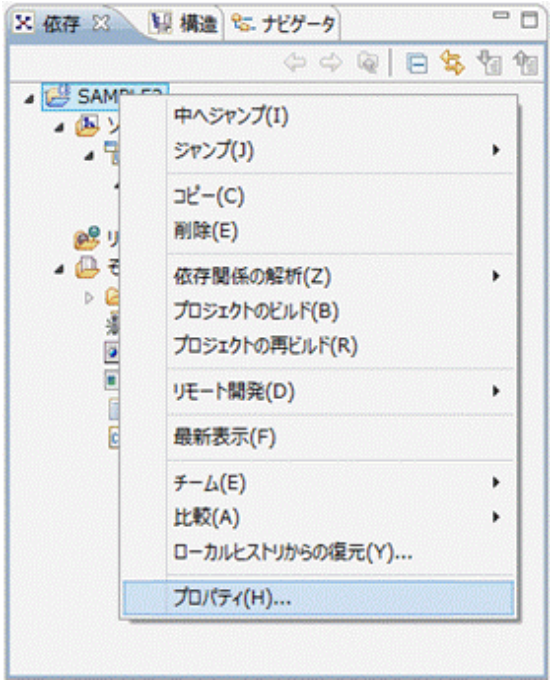
2.5.1 翻訳オプションの設定

プロジェクトで管理しているソースファイルを翻訳するときに必要になる翻訳オプションを設定します。

ここでは、例として翻訳オプションLIB（登録集ファイルのフォルダーの指定）を追加する方法を元に、翻訳オプションを設定する手順を次に示します。設定できる翻訳オプションは、“NetCOBOL ユーザーズガイド”の“付録A 翻訳オプション”を参照してください。

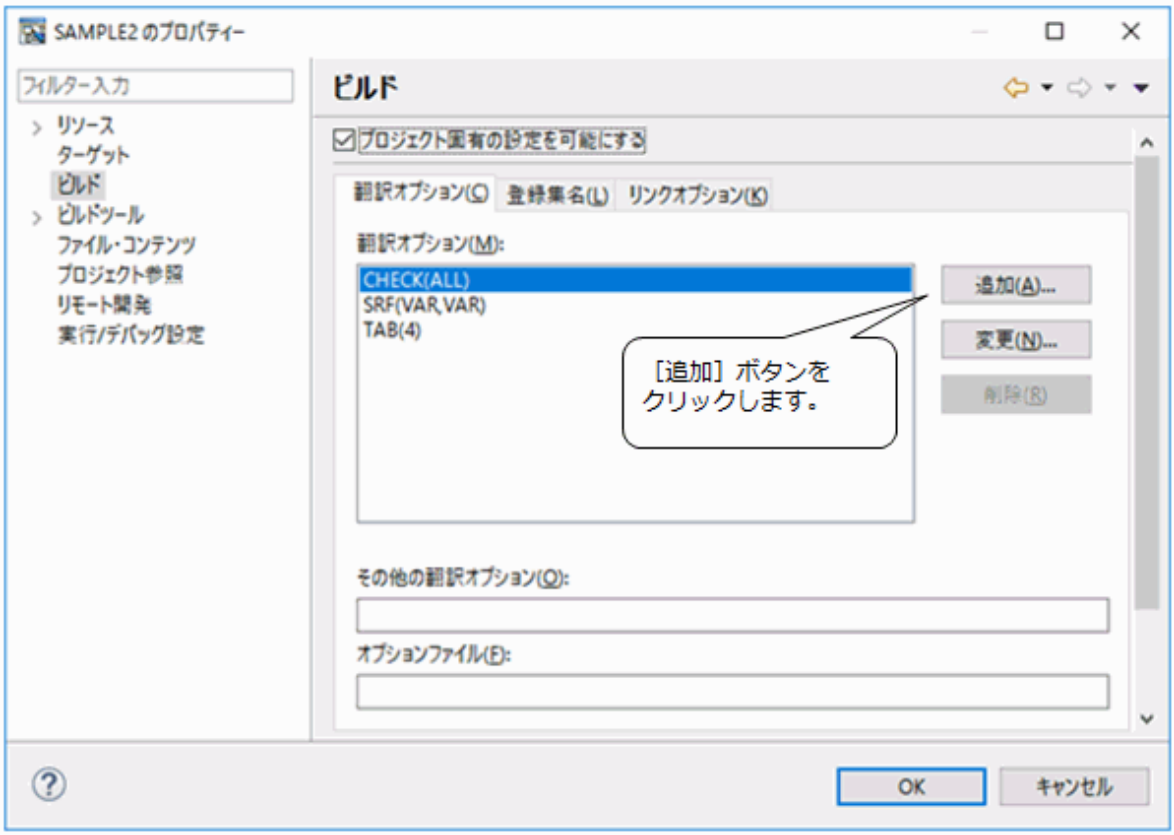
注）ここまでの手順では、COBOLソースプログラムと登録集を同じフォルダーに格納しているため、翻訳オプションLIBを指定しなくても正常にビルドできます。

1. プロジェクト「SAMPLE2」を右クリックし、[プロパティー]を選択します。



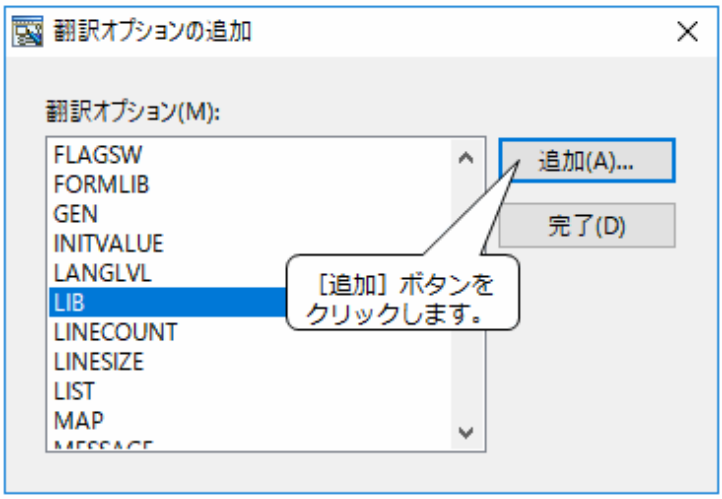
2. SAMPLE2のプロパティーの左ペインから[ビルド]を選択し、[翻訳オプション]タブの[追加]ボタンをクリックします。

   注)いくつかのオプションはデフォルトで指定されています。このうち、CHECK(ALL)はビルドモードがデバッグの場合、削除できません。



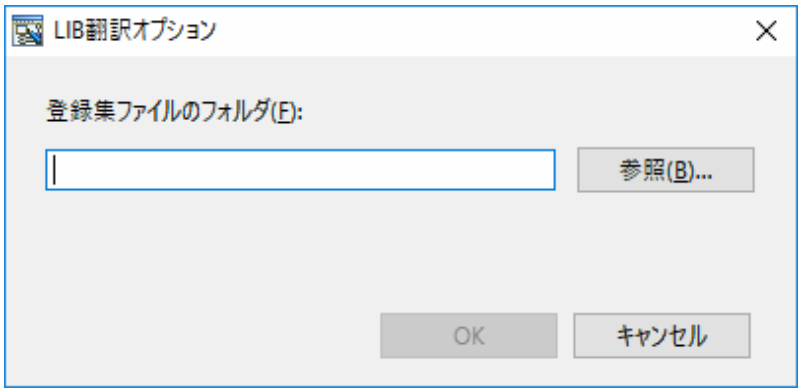
3. ［翻訳オプションの追加］ダイアログボックスから「LIB」を選択し、［追加］ボタンをクリックします。

   注）ここで[F1]キーを押すと、翻訳オプションに関するヘルプを表示させることができます。

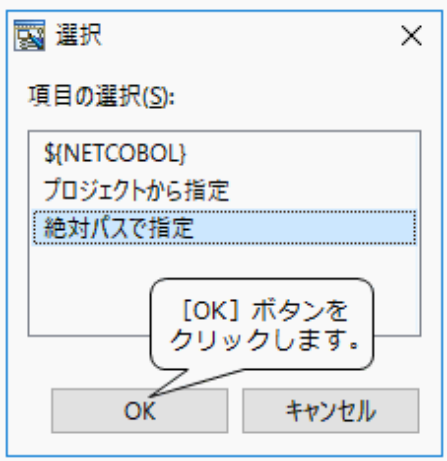


4. ［LIB翻訳オプション］ダイアログボックスに「登録集ファイルのフォルダ」を入力します。

   注）ここで[F1]キーを押すと、LIB翻訳オプションに関するヘルプを表示させることができます。



5. [参照]ボタンをクリックすると［選択］ダイアログボックスが表示され、入力を補助します。ここでは、「絶対パスで指定」を選択し、［OK］ボタンをクリックします。



6. [LIB翻訳オプション］ダイアログボックスに［登録集ファイルのフォルダー］を指定し、［OK］ボタンをクリックします。
7. ［翻訳オプションの追加］ダイアログボックスで［完了］ボタンをクリックします。設定した翻訳オプションは、プロパティーの翻訳オプション一覧に表示されます。プロパティーの［OK］ボタンをクリックします。

8. [確認]ダイアログボックスに「ビルド設定が変更されました。変更を有効にするには、プロジェクトのクリーンが必要です。ここでプロジェクトのクリーンを行いますか？」が表示された場合、[はい]ボタンをクリックします。プロジェクトのクリーンが行われ、自動的にビルドが実行されます。

参考

翻訳オプションの設定内容を変更するには、次の手順で行います。

1. プロジェクトのプロパティーから[ビルド]を選択し、翻訳オプションから設定内容を変更したい翻訳オプションを選択します。
2. [変更]ボタンをクリックし、各オプションに合わせて変更します。

また、翻訳オプションを削除するには、次の手順で行います。

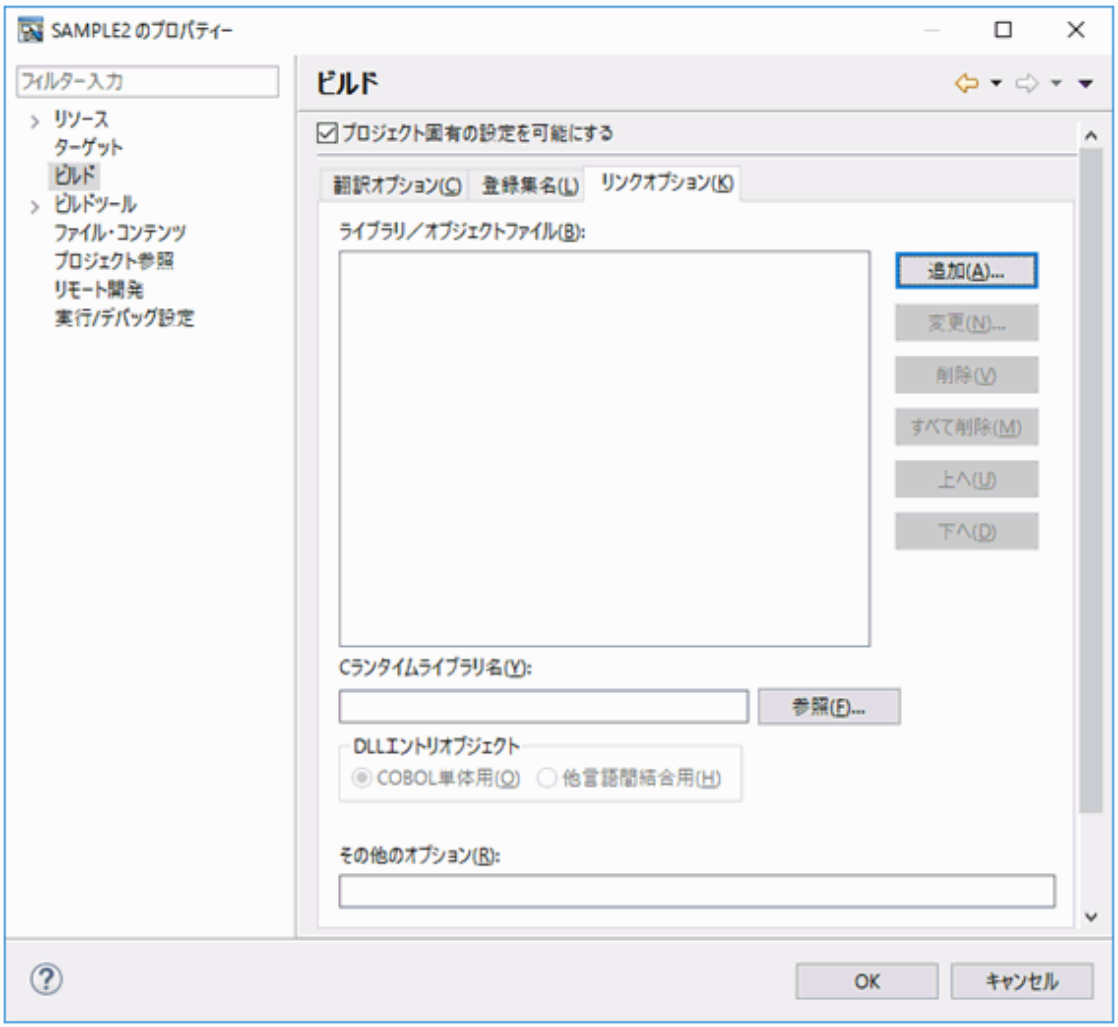
1. 翻訳オプションから削除したい翻訳オプションを選択し、[削除]ボタンをクリックします。

2.5.2 リンクオプションの設定

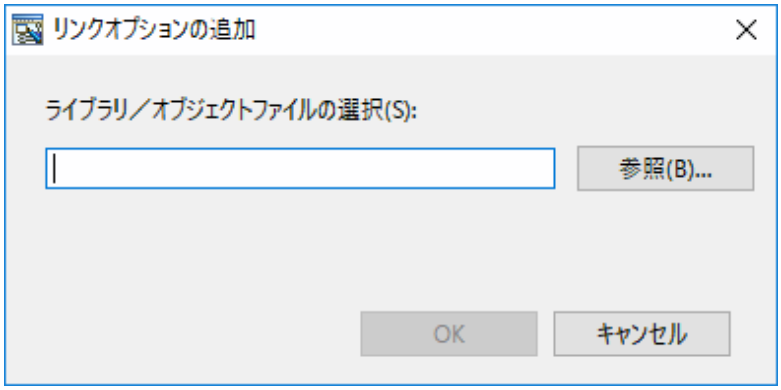
プロジェクトで管理している実行可能ファイルまたはDLLをリンクするときに有効になるリンクオプションを設定します。リンクオプションは、C言語で作成されたライブラリを結合したいときなどに設定しますが、本章で作成するアプリケーションでは、リンクオプションの設定は不要です。

ここでは、リンクオプションの設定画面の表示方法について説明します。

1. SAMPLE2のプロパティーの左ペインから[ビルド]を選択し、[リンクオプション]タブを選択します。



2. [追加]ボタンをクリックし、[リンクオプションの追加]ダイアログボックスに必要に応じて各項目の設定を行います。

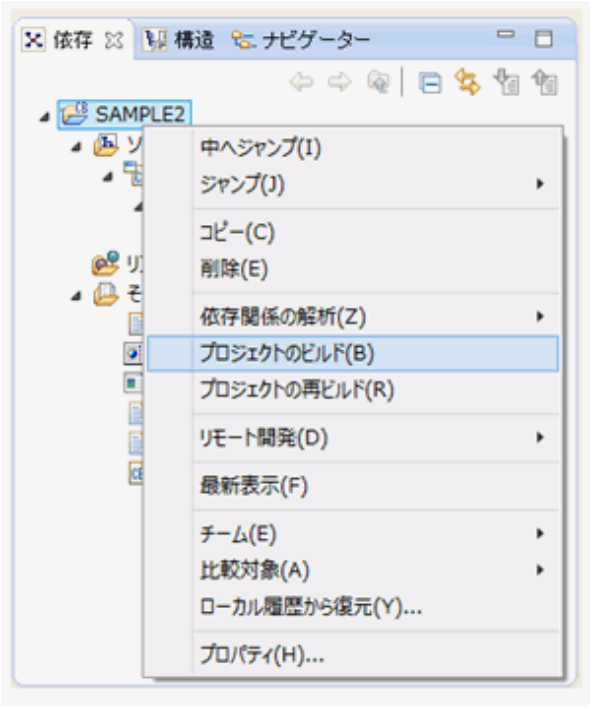


2.5.3 ビルド操作

プロジェクト「SAMPLE2」を右クリックし、[プロジェクトのビルド]を選択します。再ビルドの場合は、[プロジェクトの再ビルド]を選択します。

注)［翻訳オプションの設定］の操作で、ビルドは自動的に実行されていますので、変更がなければ[プロジェクトのビルド]を選択しても翻訳・リンクは実行されません。

エラーがなければ、ビルドが終了し、プロジェクトに登録した実行ファイルが生成されます。

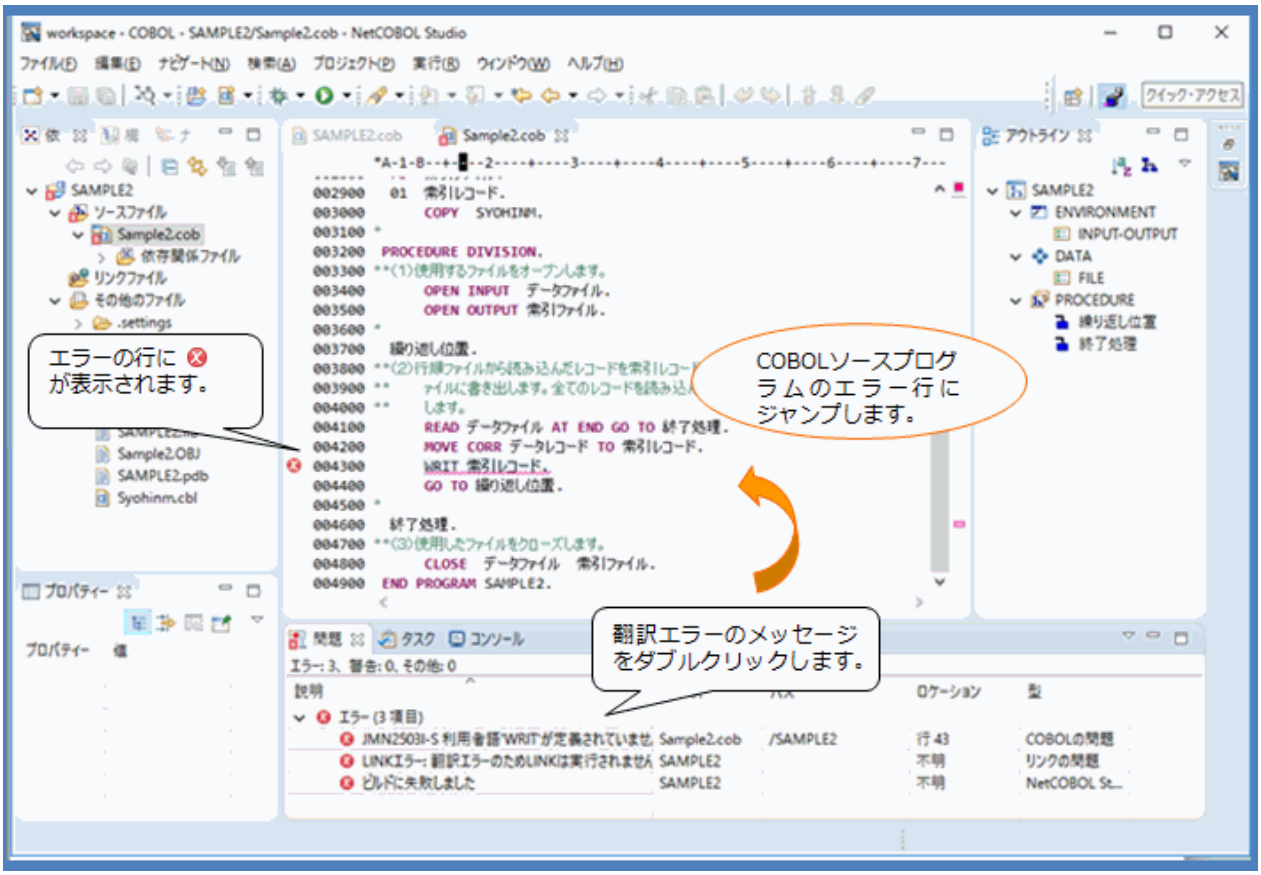


翻訳エラーの修正

ビルドにより翻訳エラーが検出された場合、翻訳終了時に[問題]ビューにエラーメッセージが表示されます。

[問題]ビューのエラーメッセージをダブルクリックするとCOBOLエディタ上の翻訳エラーが検出されたCOBOLソースプログラムの行にジャンプします。また、COBOLエディタ上でもエラー行に ⊗ がつきます。

このように翻訳エラーの発生した文の検出を簡単に行うことができ、プログラムの修正作業が効率よく行えます。



2.6 実行

ビルドされたCOBOLプログラムを実行します。

2.6.1 実行環境情報の設定

実行環境情報と初期化ファイルについて

COBOLプログラムを実行するには、実行環境情報を設定する必要があります。

NetCOBOLでは、COBOLプログラムを実行するために割り当てる資源や情報のことを実行環境情報といいます。本章では、ファイルの入出力を行うプログラムを作成しましたので、入出力するファイルを実行環境情報として指定します。

実行環境情報は、実行用の初期化ファイルに格納します。COBOLプログラムは、実行時に実行用の初期化ファイルから情報を取り出して実行します。通常、実行可能プログラム(EXE)が格納されているフォルダーの「COBOL85.CBR」を実行用の初期化ファイルとして扱います。

実行環境設定ツールによる実行用の初期化ファイルの作成

NetCOBOLでは、実行用の初期化ファイルの内容を編集し、実行環境情報を設定するツールとして「実行環境設定ツール」があります。実行環境設定ツールを使用して実行用の初期化ファイルを作成する方法を以下に示します。

1. 実行環境設定ツールを起動します。実行環境設定ツールは、以下のいずれかの方法で起動します。

   ― [スタート]> お使いのNetCOBOL製品 > [実行環境設定ツール]を選択します。

   ― [NetCOBOLコマンドプロンプト]から、実行環境設定ツール(COBENVUT.EXE)を起動します。

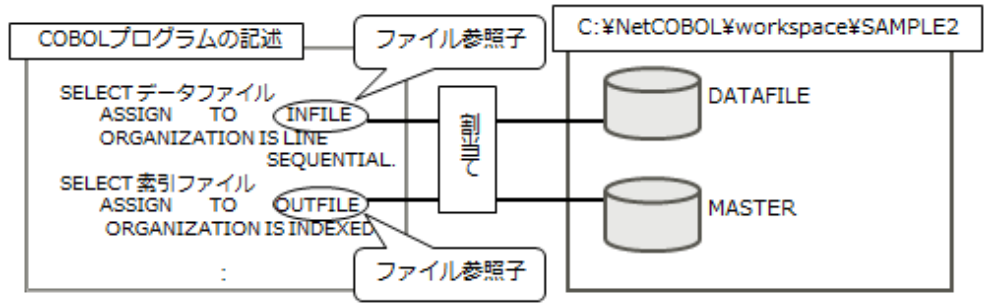
2. [実行環境設定ツール]のメニューバーから、[ファイル] > [開く]を選択します。

   → [実行用の初期化ファイルの指定]ウィンドウが表示されます。

3. 実行可能プログラム(EXE)が存在するフォルダーのCOBOL85.CBRを選択し、[開く]ボタンをクリックします。

ファイル識別名とファイルの関連付け

実行環境設定として、COBOLプログラムとファイルの実体との関連付けを行います。関連付けとして、本章で作成するアプリケーションでは、COBOLプログラムのASSIGN句に定義されたファイル参照子に実際のファイルを割り当てます。

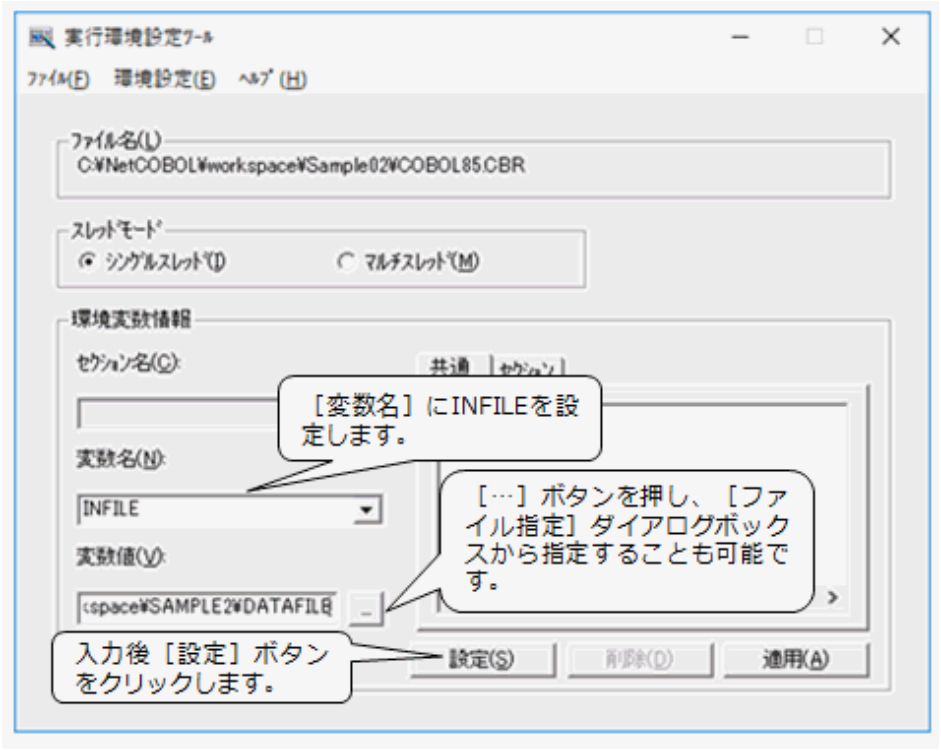


実行環境設定ツールでは、[変数名]にファイル参照子を指定し、[変数値]には実際のファイルを指定します。

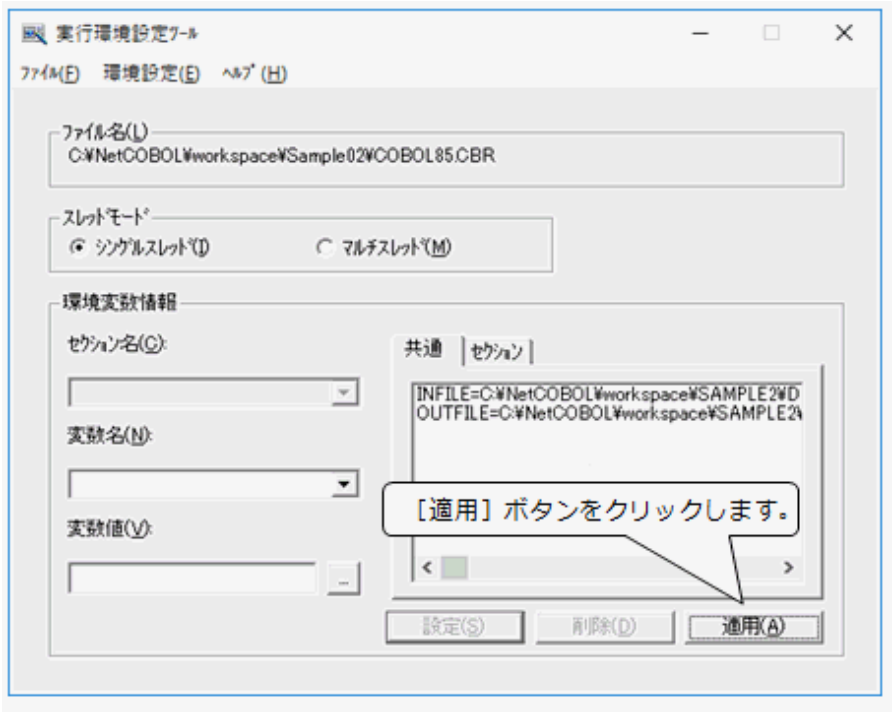
ここでは、以下を指定します。

変数名	変数値
INFILE	C:¥NetCOBOL¥workspace¥SAMPLE2¥DATAFILE
OUTFILE	C:¥NetCOBOL¥workspace¥SAMPLE2¥MASTER

1. [実行環境設定ツール]から[変数名]と[変数値]を設定します。



2. 同様に、OUTFILEに対する[変数名]と[変数値]を設定し、[適用]ボタンをクリックします。



3. 実行環境情報の設定が終了したら、[ファイル]メニューから[終了]を選択し、実行環境設定ツールを終了します。

参考

実行環境情報の設定方法は、ここで紹介した実行環境設定ツールによる設定のほかに、[実行]メニューから[実行構成]を選択し、[実行構成]ダイアログボックスで実行環境変数を指定することも可能です。

2.6.2 プログラムの実行

プログラムを実行するには、プロジェクトを選択した状態で、[実行]メニューから[実行(S)] > [COBOLアプリケーション]を選択します。

作成されたプログラムでは、実行の終了メッセージが画面に表示されません。実行が終了すると、索引ファイル「MASTER」が作成されます。

参考

引数を指定してプログラムを実行するには、[実行]メニューから[実行構成(N)]を選択し、[実行構成]ダイアログボックスからプログラム引数を指定して実行します。

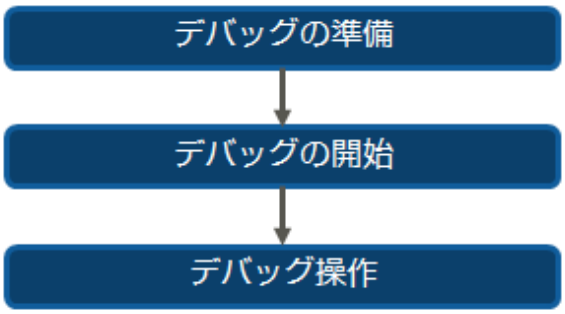
2.7 デバッグ

NetCOBOL Studio上でのプログラムのデバッグについて説明します。

デバッグ機能では、実行可能プログラムをそのままデバッグの対象とし、プログラムの論理的な誤りを、プログラムを動作させながら検出することができます。

デバッグ作業は、画面に表示したCOBOLソースプログラムに対する直接的で簡単な操作で行うことができます。キーボードやマウスを使い、メニュー内のコマンドやツールバーに表示されたボタンを操作することによって、デバッグ作業を行います。

デバッグの流れを次に示します。

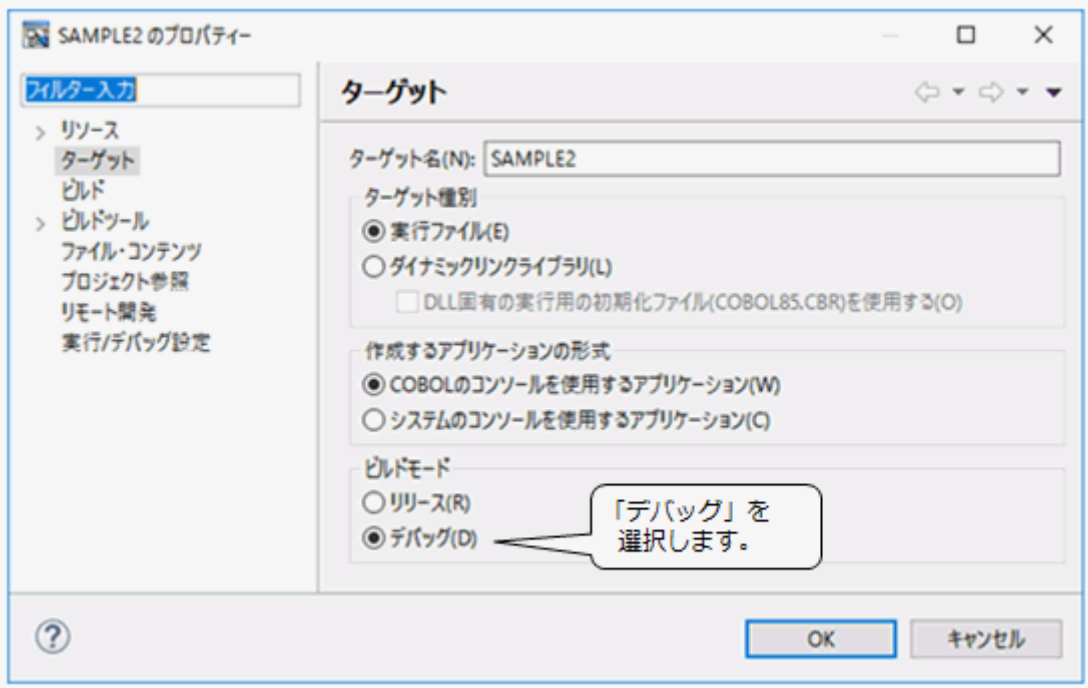


2.7.1 デバッグの準備

デバッグの準備として以下の操作を行います。

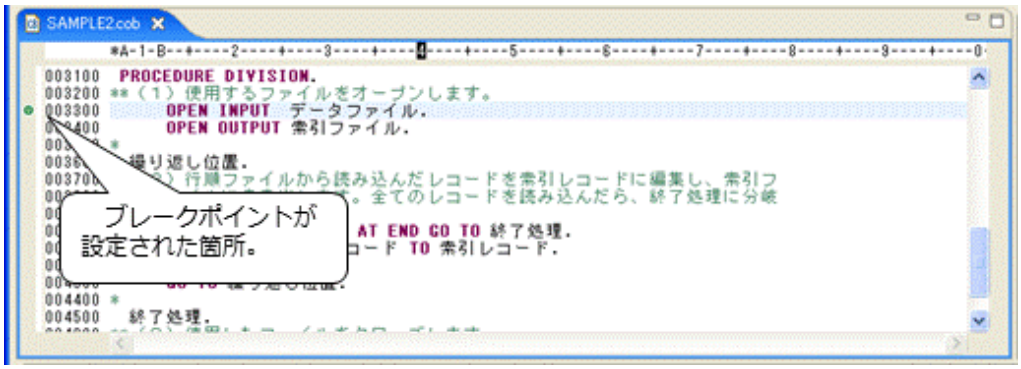
1. NetCOBOL Studio上で、デバッグ作業を行うプログラムのプロジェクトを右クリックし、プロパティーを開きます。
2. プロジェクトのプロパティーの[ターゲット]から、ビルドモード「デバッグ」を選択します。

   注）デフォルトはデバッグモードになっています。



3. [OK]ボタンをクリックします。
4. 「ビルド設定が変更されました。変更を有効にするには、プロジェクトのクリーンが必要です。ここでプロジェクトのクリーンを行いますか？」という[確認]メッセージボックスが表示されたら、[はい]ボタンをクリックします。
5. プロジェクトをビルド（または再ビルド）します。

6. [COBOLパースペクティブ]のエディタ上で、手続きの先頭の行(OPEN文)の左端を右クリックし、コンテキストメニューから[ブレークポイントの追加]を選択します。

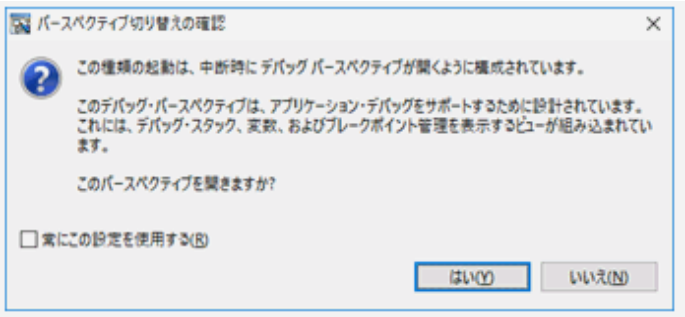


2.7.2 デバッグの開始

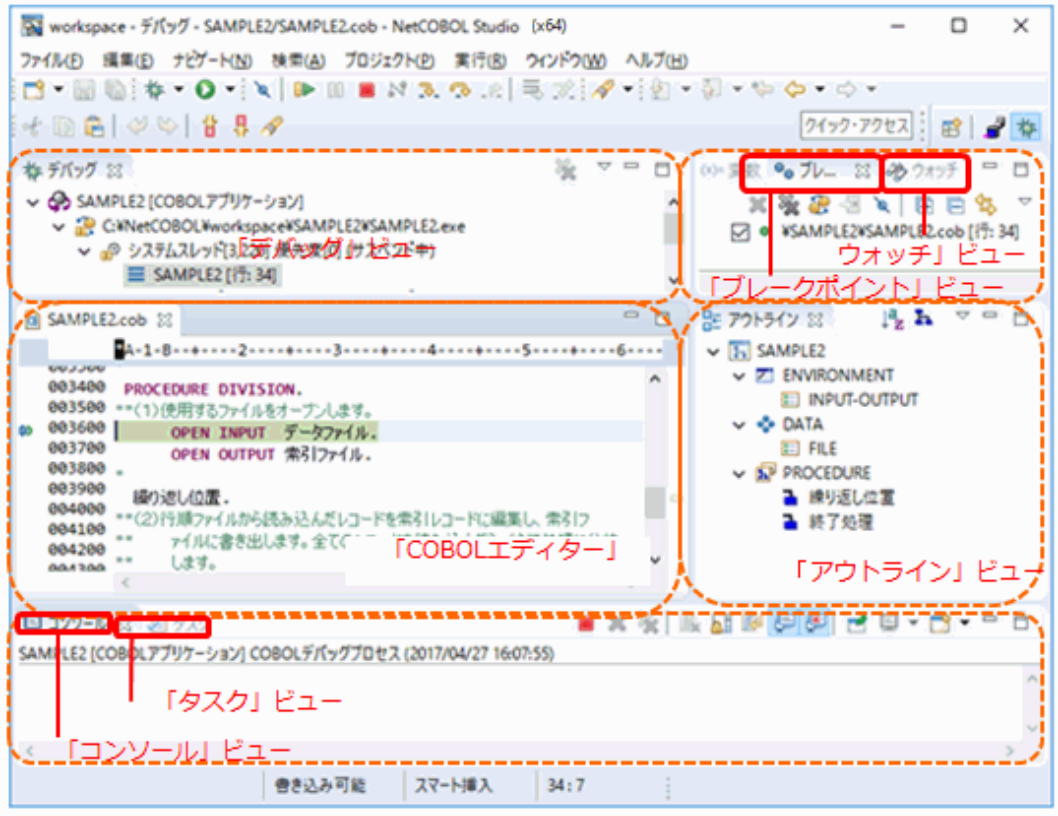
NetCOBOL Studioでのデバッグは、デバッグパースペクティブで行います。デバッグパースペクティブはデバッグに適したビューで構成されています。

デバッグパースペクティブは、以下の操作で表示します。

1. [実行]メニューから、[デバッグ(G)] > [COBOLアプリケーション]を選択します。
2. [パースペクティブ切り替えの確認]ダイアログボックスが表示された場合、[はい]ボタンをクリックします。



3. 「COBOLパースペクティブ」から「デバッグパースペクティブ」に切り替わります。



各ビューの概要を説明します。

- 「デバッグ」ビュー

  プロジェクト名、実行中のプログラム名などがツリー表示され、プログラムの実行状態や呼び出し経路などを確認することができます。

- 「ブレークポイント」ビュー

  プロジェクトで設定したブレークポイントを表示します。

- 「ウォッチ」ビュー

  ウォッチ対象のデータ項目を表示します。データ項目が保持する値などを確認することができます。

- 「アウトライン」ビュー

  エディタに表示されているCOBOLソースの構造（PROGRAM-IDや環境部、データ部など）を表示します。

- 「タスク」ビュー

  後で検討する項目などをタスクとして記録しておきたい場合に使用します。

- 「コンソール」ビュー

  コンソール出力結果を表示します。

参考

「デバッグパースペクティブ」から「COBOLパースペクティブ」へ切り替えるには、以下の方法があります。

- 右上の[ ]ボタンをクリックし、メニューから「その他」を選択します。[パースペクティブの選択]ダイアログボックスからCOBOLを選択します。

- 「COBOLパースペクティブ」を閉じていない場合、ショートカットバーの[COBOL]アイコンをクリックしてパースペクティブを切り替えます（ショートカットバーに[COBOL]アイコンが表示されていない場合は、右上端の[»]ボタンをクリックするか、ショートカットバーの表示域を広げると[COBOL]アイコンが表示されます）。

2.7.3 デバッグ操作

ここでは、次に示すようなデバッグ操作について説明します。

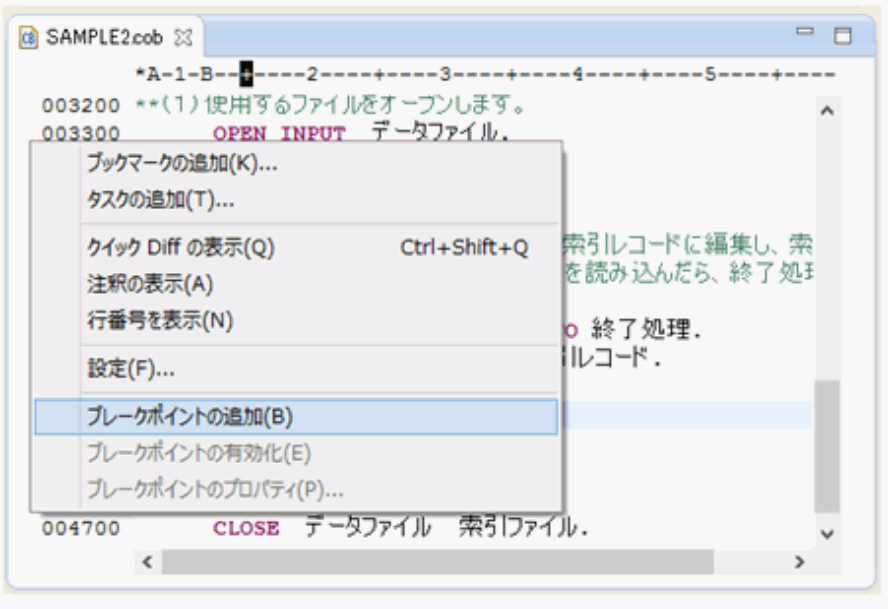
- ある文に到達したら実行を中断する
- 1文だけ実行したら実行を中断する
- あるデータの値を確認する
- データの値が変更されたら実行を中断する

2.7.3.1 ある文に達したら実行を中断する

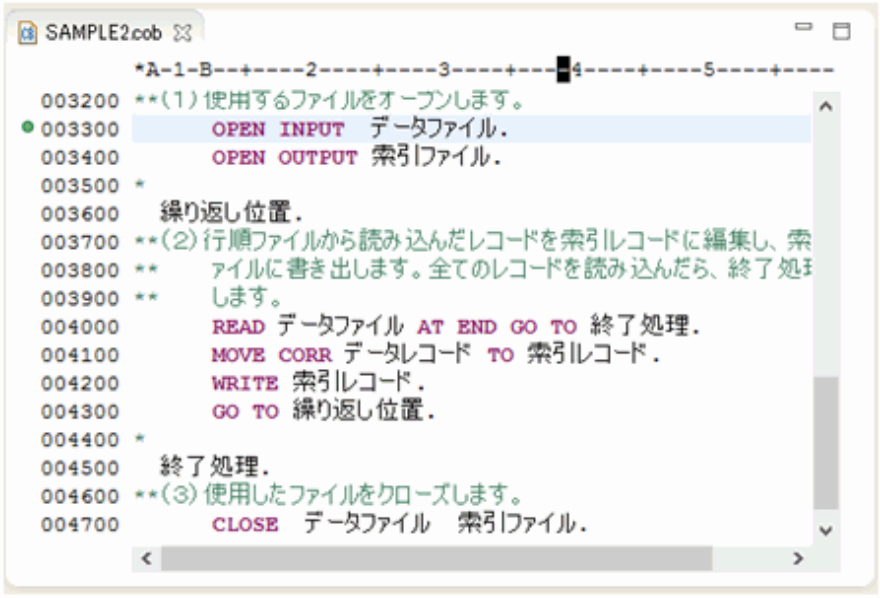
ソースプログラム中のある文に達したら、プログラムの実行を中断してデバッグ操作を可能にするには、ブレークポイントを設定します。ブレークポイントを設定すると、ブレークポイントを設定した前の文の処理でプログラムが中断されます。

ブレークポイントを設定するデバッグ操作の手順を説明します。

1. ブレークポイントを設定するには、デバッグパースペクティブのCOBOLソースプログラムが表示されているビューで、ブレークポイントを設定する行の左端にカーソルを置きます。
2. マウスを右クリックし、[ブレークポイントの追加]を選択します。

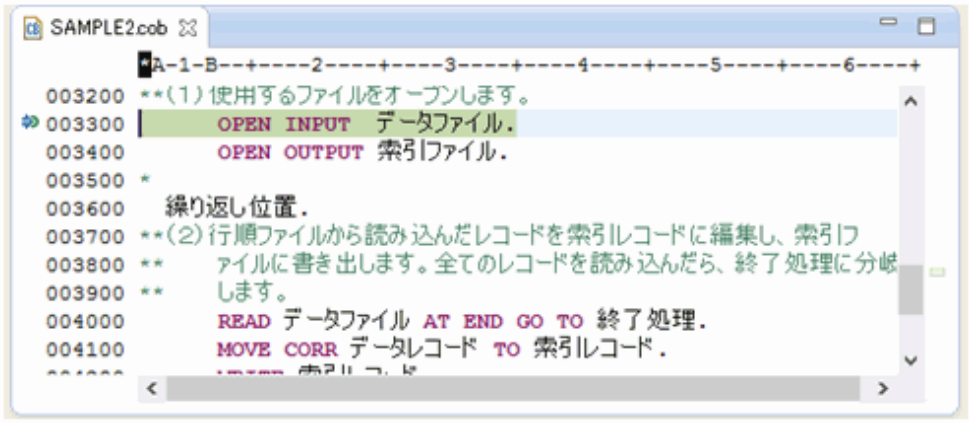


3. ブレークポイントを設定すると、行の左端に ● が表示され、行に色がつきます。



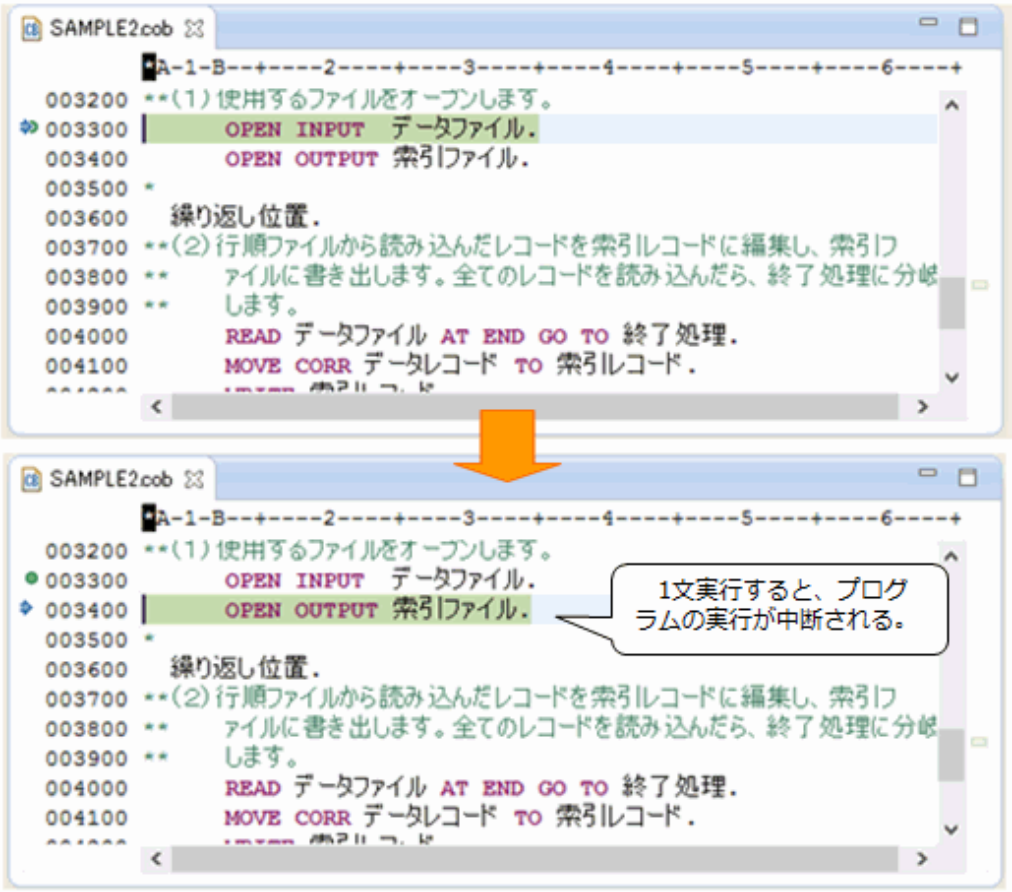
4. [ブレークポイント]ビューに設定したブレークポイントが表示されます。

5. ブレークポイントを設定した文の前の処理でプログラムが中断されます。ブレークポイントを設定した行に現在の命令ポインタを示す が表示され、行の色が変わります。



2.7.3.2 1文だけ実行して実行を中断する

ソースプログラムの1文だけ実行したら実行を中断するには、[デバッグ]ビューのツールバーボタン または[F5]キーをクリックします。



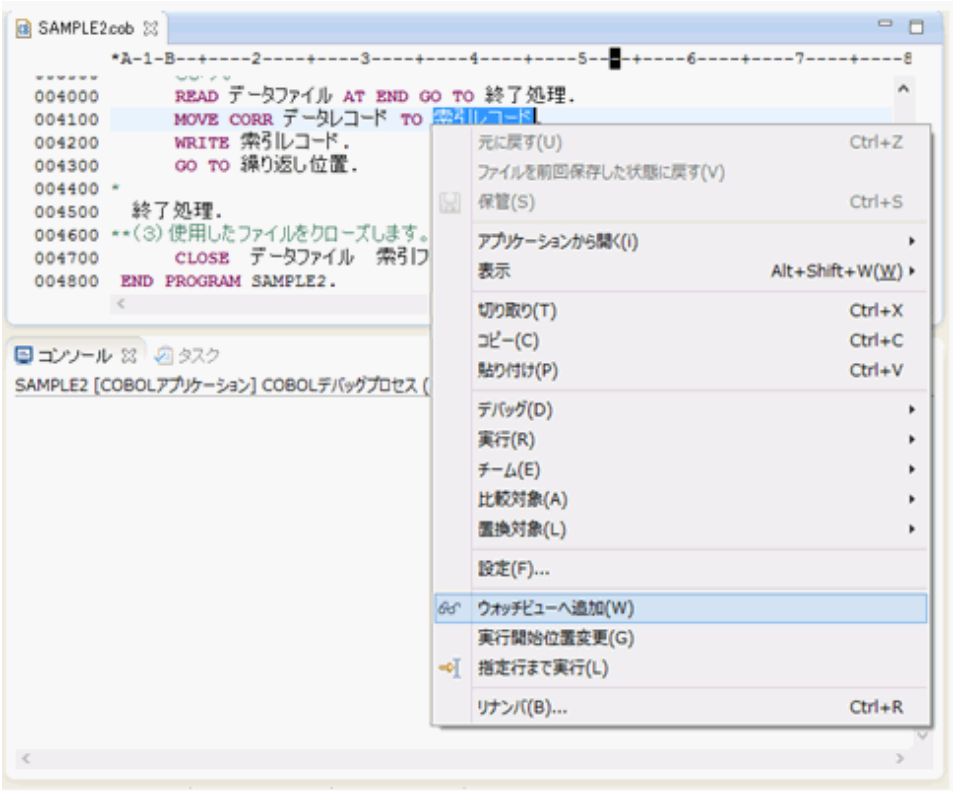
2.7.3.3 データの値を確認する

データの現在の値を確認するには、[ウォッチ]ビューを使用します。

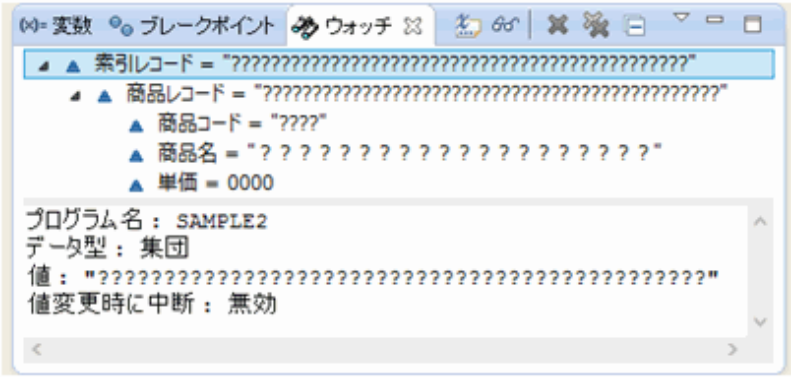
ここでは、データ名「索引レコード」の値を確認する例を元に、データの値を確認する手順を説明します。

1. エディタ上で値を確認するデータを範囲選択し、右クリックします。

2. コンテキストメニューから[ウォッチビューへ追加]を選択します。

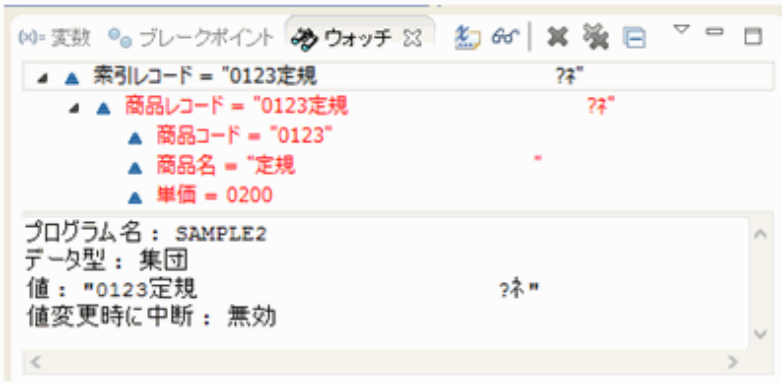


→[ウォッチ]ビューに[索引レコード]が追加されます。[索引レコード]を展開すると、従属する項目がツリー状に表示され、集団項目全てを確認できます。



3. プログラムを実行しながら、データの値が変化するのを監視します。

   このサンプルでは、41行目のMOVE文を実行すると「索引レコード」が変化します。

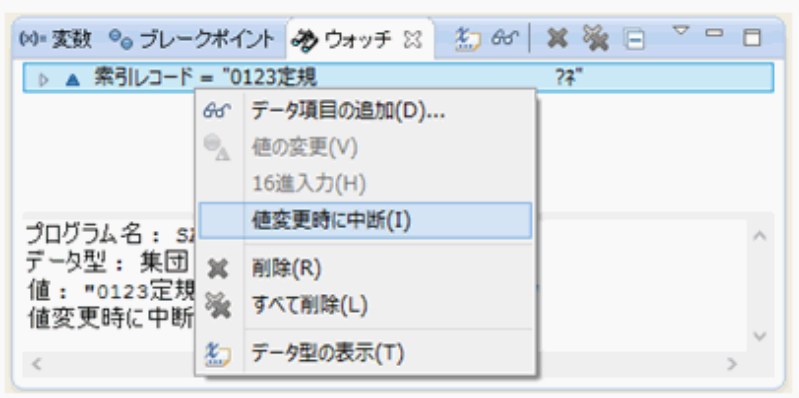


2.7.3.4 データの値が変更されたら実行を中断する

データの値が変更された場合に実行を中断することができます。

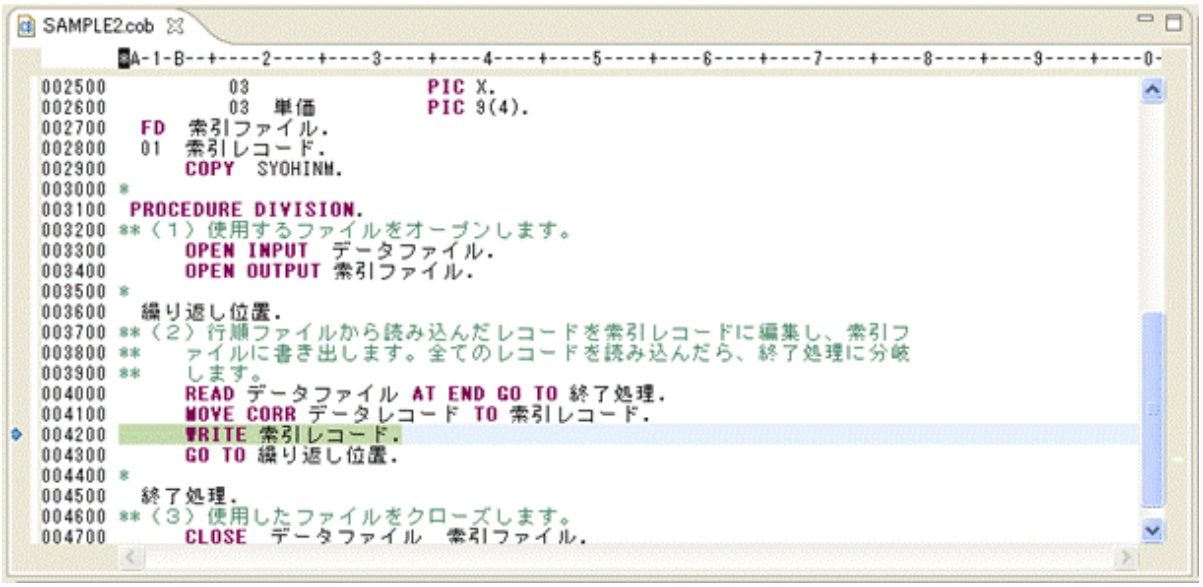
ここでは、データ名「索引レコード」の値が変更される度に実行が中断する例を元に、手順を説明します。

1. 「索引レコード」を[ウォッチ]ビューに追加します。
2. 「索引レコード」を右クリックし、[値変更時に中断]を選択します。



3. プログラムを実行すると、「索引レコード」が変更される度に実行が中断します。

   このサンプルでは、41行目のMOVE文により「索引レコード」が変更されるので、42行目のWRITE文で中断します。



2.7.4 デバッグの終了

［実行］メニューから[終了]を選択し、デバッグを終了します。

2.8 NetCOBOL Studioの終了

NetCOBOL Studioの[ファイル]メニューから[終了]を選択して、NetCOBOL Studioを終了します。

第3章 画面帳票アプリケーションの開発

本章では、画面帳票定義体を使用した画面帳票アプリケーションについて説明します。また、基本的な画面帳票アプリケーションを作成する方法について説明します。

3.1 概要

NetCOBOLシリーズで作成できる画面帳票アプリケーション、および本章で作成する画面帳票アプリケーションの概要について説明します。

3.1.1 画面帳票アプリケーションの概要

NetCOBOLシリーズでは、「画面定義体」と呼ばれる画面フォーマット、「帳票定義体」と呼ばれる帳票フォーマットおよび帳票定義体に重ねて使用する「オーバレイ定義体」を用いることにより、COBOLによるきめ細かい画面帳票アプリケーションを作成することができます。

画面定義体および帳票定義体には、項目の位置、項目の種別(どのような種類のデータを扱う項目か、固定的な項目かなど)、項目の属性(文字の大きさ、色など)、罫線や網がけといった装飾などを定義します。また、オーバレイ定義体には固定的な文字や図形などを定義します。

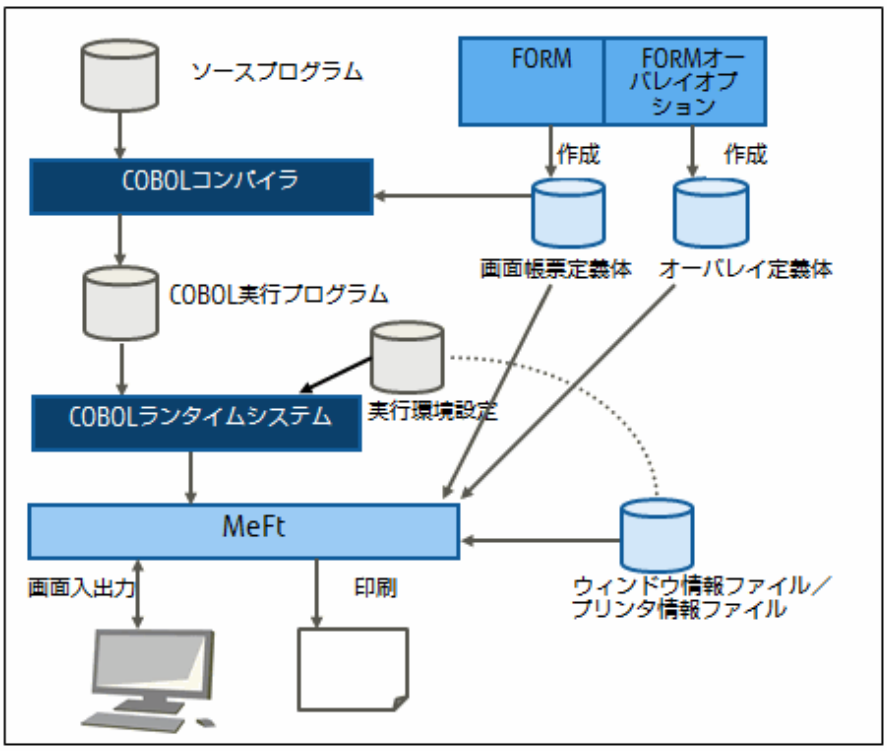
画面定義体、帳票定義体およびオーバレイ定義体は、COBOLプログラムから独立しているため、作成や変更が容易です。なお、画面定義体と帳票定義体を総称して、「画面帳票定義体」と呼びます。

画面帳票定義体およびオーバレイ定義体を使用したプログラムの開発および実行には、NetCOBOLのほかに次のツールも使用します。

名称	説明
FORM	画面帳票定義体を画面イメージで設計するツール。
FORMオーバレイオプション	オーバレイ定義体を画面イメージで設計するFORMのオプション製品。
PowerFORM	帳票定義体を画面イメージで作成する帳票設計ツール。FORMに含まれます。
MeFt	画面帳票定義体およびオーバレイ定義体を元に、画面表示および帳票印刷を行うライブラリ。

これらの製品はNetCOBOLシリーズのStandard Editionおよび Enterprise Editionに含まれています。

COBOL、FORM、FORMオーバレイオプション、MeFtの関連図を示します。



COBOLプログラムからFORM、MeFtを使用して画面入出力を行う場合、表示ファイルによるアプリケーションを作成します。表示ファイルによる画面帳票入出力では、通常のファイルを扱うのと同じようにWRITE文やREAD文を使用します。つまり、WRITE文で画面への出力を行い、READ文で画面から入力します。

プログラムは、画面およびプリンターとのデータの受渡し手段としてレコードを使用します。画面帳票定義体に定義されたデータ項目のレコードは、COBOLのCOPY文を使って、翻訳時にプログラムに取り込むことができます。そのため、画面帳票の入出力のためのレコードの定義をCOBOLプログラムに記述する必要はありません。

なお、データ項目のウィンドウ内での位置や印刷位置など、ウィンドウやプリンターの制御はMeFtが行うため、COBOLプログラムでは意識する必要がありません。

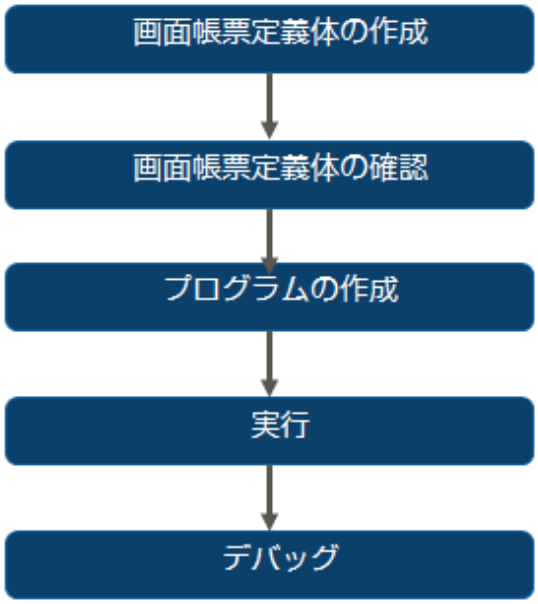
3.1.2 作成するアプリケーションについて

この章で作成するアプリケーションでは、1つの画面定義体と1つの帳票定義体を使用します。画面定義体を使用してデータを入力し、入力されたデータは帳票定義体を使用して印刷します。

なお、画面帳票定義体の作成では、定義体は「C:¥EDUCATION」に格納することとします。

3.1.3 アプリケーション開発の流れ

画面帳票定義体を使用した画面帳票アプリケーションの開発の流れを次に示します。



3.2 表示ファイルのプログラミング

表示ファイル機能を使って画面入出力を行うときのプログラム記述について、COBOLの各部ごとに説明します。

[ADDR.cob]

COBOLソースプログラムは以下を使用します。

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.  ADDR.
*
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.

   FILE-CONTROL.
     SELECT ディスプレイファイル ASSIGN  TO  GS-DSPFILE
         SYMBOLIC DESTINATION IS  "DSP"
         FORMAT               IS DSP-FORMAT
         GROUP                IS DSP-GROUP
         PROCESSING MODE      IS DSP-MODE
         UNIT CONTROL         IS DSP-CONTROL
         SELECTED FUNCTION    IS DSP-ATTN
         FILE STATUS          IS DSP-STATUS1 DSP-STATUS2.
     SELECT プリンタファイル ASSIGN  TO  GS-PRTFILE
         SYMBOLIC DESTINATION IS  "PRT"
         FORMAT               IS PRT-FORMAT
         GROUP                IS PRT-GROUP
         PROCESSING MODE      IS PRT-MODE
         UNIT CONTROL         IS PRT-CONTROL
         FILE STATUS          IS PRT-STATUS1 PRT-STATUS2.
*
```

```
 DATA DIVISION.
 FILE SECTION.
 FD ディスプレイファイル.
  COPY ADDRDSP OF XMDLIB.
 FD プリンタファイル.
  COPY ADDRPRT OF XMDLIB.
*
```

```
 WORKING-STORAGE SECTION.
 01  DSP-FORMAT        PIC X(08).
 01  DSP-GROUP         PIC X(08).
 01  DSP-MODE          PIC X(02).
 01  DSP-CONTROL       PIC X(06).
 01  DSP-ATTN          PIC X(04).
 01  DSP-STATUS1       PIC X(02).
 01  DSP-STATUS2       PIC X(04).
*
 01  PRT-FORMAT        PIC X(08).
 01  PRT-GROUP         PIC X(08).
 01  PRT-MODE          PIC X(02).
 01  PRT-CONTROL       PIC X(06).
 01  PRT-STATUS1       PIC X(02).
 01  PRT-STATUS2       PIC X(04).
*
 PROCEDURE DIVISION.
*
     PERFORM 画面オープン.
     INITIALIZE 住所録入力画面.
     PERFORM NO LIMIT
         PERFORM 画面出力
         PERFORM 画面入力
         EVALUATE DSP-ATTN
             WHEN "PRT "
                 PERFORM 印刷オープン
                 INITIALIZE 住所録印刷帳票
                 PERFORM 印刷データ設定
                 PERFORM 印刷処理
                 PERFORM 印刷クローズ
             WHEN "END "
                 CLOSE ディスプレイファイル
                 GO TO 終了処理
         END-EVALUATE
     END-PERFORM.
*
*=========================================================
 画面オープン.
     OPEN I-O ディスプレイファイル.
     IF DSP-STATUS2 NOT = "0000" THEN
         PERFORM 終了処理
     END-IF.
*=========================================================
 画面出力.
     MOVE "ADDRDSP" TO DSP-FORMAT.
     MOVE "@ALLF"   TO DSP-GROUP.
     MOVE "  "      TO DSP-MODE.
     WRITE 住所録入力画面.
     IF DSP-STATUS2 NOT = "0000" THEN
         CLOSE ディスプレイファイル
         GO TO 終了処理
     END-IF.
*=========================================================
 画面入力.
     MOVE "@ALLF"   TO DSP-GROUP.
     MOVE "NE"      TO DSP-MODE.
     READ ディスプレイファイル.
     IF DSP-STATUS2 NOT = "0000" THEN
         CLOSE ディスプレイファイル
         GO TO 終了処理
     END-IF.
*=========================================================
 印刷オープン.
```

```
     OPEN OUTPUT プリンタファイル.
     IF PRT-STATUS2 NOT = "0000" THEN
         CLOSE ディスプレイファイル
         GO TO 終了処理
     END-IF.
*=========================================================
 印刷データ設定.
     MOVE 名前     OF 住所録入力画面 TO 名前     OF 住所データ OF 住所録印刷帳票(1).
     MOVE 住所     OF 住所録入力画面 TO 住所     OF 住所データ OF 住所録印刷帳票(1).
     MOVE 電話番号 OF 住所録入力画面 TO 電話番号 OF 住所データ OF 住所録印刷帳票(1).
     MOVE メール   OF 住所録入力画面 TO メール   OF 住所データ OF 住所録印刷帳票(1).
     MOVE 生年月日 OF 住所録入力画面 TO 生年月日 OF 住所データ OF 住所録印刷帳票(1).
*=========================================================
 印刷処理.
     MOVE "ADDRPRT" TO PRT-FORMAT.
     MOVE "@ALLF"   TO PRT-GROUP.
     MOVE "  "      TO PRT-MODE.
     WRITE 住所録印刷帳票.
     IF PRT-STATUS2 NOT = "0000" THEN
         CLOSE ディスプレイファイル
         CLOSE プリンタファイル
         GO TO 終了処理
     END-IF.
*=========================================================
 印刷クローズ.
     CLOSE プリンタファイル.
*=========================================================
 終了処理.
 END PROGRAM ADDR.
*=========================================================
```

3.2.1 環境部(ENVIRONMENT DIVISION)

表示ファイルを定義します。表示ファイルは、通常のファイルを定義するときと同様に、入出力節のファイル管理段落にファイル管理記述項を記述します。

以下に、COBOLプログラムの記述例とファイル管理記述項に指定できる内容を示します。

COBOLプログラムの記述例

```
ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
   FILE-CONTROL.
     SELECT  ディスプレイファイル ASSIGN TO  GS-DSPFILE  *>--+
       SYMBOLIC DESTINATION IS  "DSP"                    *>  |
       FORMAT               IS  DSP-FORMAT               *>  |表示ファイル（画面機能）の定義
       GROUP                IS  DSP-GROUP                *>  |
       PROCESSING MODE      IS  DSP-MODE                 *>  |
       UNIT CONTROL         IS  DSP-CONTROL              *>  |
       SELECTED FUNCTION    IS  DSP-ATTN                 *>  |
       FILE  STATUS         IS  DSP-STATUS1 DSP-STATUS2. *>--+
     SELECT  プリンタファイル  ASSIGN TO  GS-PRTFILE     *>--+
       SYMBOLIC DESTINATION IS  "PRT"                    *>  |
       FORMAT               IS  PRT-FORMAT               *>  |表示ファイル（帳票機能）の定義
       GROUP                IS  PRT-GROUP                *>  |
       PROCESSING MODE      IS  PRT-MODE                 *>  |
       UNIT CONTROL         IS  PRT-CONTROL              *>  |
       FILE  STATUS         IS  PRT-STATUS1 PRT-STATUS2. *>--+
```

ファイル管理記述項に指定できる内容

指定場所	情報の種類	指定する内容	必須／任意
SELECT句	ファイル名	COBOL プログラム中で使用するファイル名を指定します。	必須
ASSIGN句	ファイル参照子	「GS-ファイル識別名」の形式で指定します。このファイル識別名は、実行時に使用するウィンドウ情報ファイルまたはプリンタ情報ファイルのファイル名を設定する環境変数情報になります。	必須
SYMBOLIC DESTINATION句	あて先種別	データの入出力のあて先を指定します。画面機能では「DSP」(または省略)、帳票機能では「PRT」を指定します。	任意(画面) 必須(帳票)
FORMAT句	データ名	作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名には、画面帳票定義体名を設定します。	必須
GROUP 句	データ名	作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名には、入出力の対象となる項目群名を設定します。	必須
PROCESSING MODE句	データ名	作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、入出力の処理種別を設定します。	任意
UNIT CONTROL句	データ名	作業場所節または連絡節で、6桁の英数字項目として定義したデータ名を指定します。このデータ名には、制御情報を設定します。	任意
SELECTED FUNCTION句	データ名	作業場所節または連絡節で、4桁の英数字項目として定義したデータ名を指定します。このデータ名には、READ文完了時にアテンション情報が通知されます。画面機能で指定します。	任意
FILE STATUS句	データ名	作業場所節または連絡節で、2桁および4桁の英数字項目として定義したデータ名を指定します。このデータ名には、入出力処理の実行結果が設定されます。なお、4桁のデータ名の領域には、実行結果の詳細情報が設定されます。	任意

注)詳細は、“NetCOBOLユーザーズガイド”の“表示ファイル(帳票印刷)の使い方”の“プログラムの記述”(帳票機能の場合)および“表示ファイル(画面入出力)の使い方”の“プログラムの記述”(画面機能の場合)をご参照ください。

3.2.2 データ部(DATA DIVISION)

データ部には、表示ファイルのレコード定義およびファイル管理記述項に指定したデータの定義を記述します。表示ファイルのレコードは、XMDLIBを指定したCOPY文を使って画面帳票定義体から取り込むことができます。

以下にCOBOLプログラムの記述例を示します。

```
DATA DIVISION.
FILE SECTION.
FD ディスプレイファイル.      *>--+
  COPY ADDRDSP OF XMDLIB.     *>  |各表示ファイルのレコードを画面帳票定義体からCOPY文で取り込み
FD プリンタファイル.          *>  |
  COPY ADDRPRT OF XMDLIB.     *>--+
WORKING-STORAGE SECTION.
 01  DSP-FORMAT        PIC X(08).    *>--+
 01  DSP-GROUP         PIC X(08).    *>  |
 01  DSP-MODE          PIC X(02).    *>  |FILE-CONTROLで定義した表示ファイル（画面機能）の
 01  DSP-CONTROL       PIC X(06).    *>  |各データ項目の定義
 01  DSP-ATTN          PIC X(04).    *>  |
 01  DSP-STATUS1       PIC X(02).    *>  |
 01  DSP-STATUS2       PIC X(04).    *>--+

 01  PRT-FORMAT        PIC X(08).    *>--+
```

```
01  PRT-GROUP          PIC X(08).     *>  |
01  PRT-MODE           PIC X(02).     *>  | FILE-CONTROLで定義した表示ファイル（帳票機能）の
01  PRT-CONTROL        PIC X(06).     *>  | 各データ項目の定義
01  PRT-STATUS1        PIC X(02).     *>  |
01  PRT-STATUS2        PIC X(04).     *>--+
```

3.2.3 手続き部(PROCEDURE DIVISION)

画面入出力および帳票出力の開始にはOPEN文を、終了にはCLOSE文を使用します。また、画面および帳票の入出力には、通常のファイル処理を行うときと同様に、READ文およびWRITE文を使用します。

手続き部について、画面機能と帳票機能に分けて説明します。

3.2.3.1 画面機能

COBOLプログラムの記述例

```
OPEN I-O ディスプレイファイル.
MOVE "ADDRDSP" TO DSP-FORMAT.
MOVE "@ALLF"   TO DSP-GROUP.
MOVE "  "      TO DSP-MODE.
WRITE 住所録入力画面.
MOVE "@ALLF"   TO DSP-GROUP.
MOVE "NE"      TO DSP-MODE.
READ ディスプレイファイル.
EVALUATE DSP-ATTN
    WHEN "PRT "
        :
    WHEN "END "
        :
END-EVALUATE.
CLOSE ディスプレイファイル.
```

画面機能のREAD文およびWRITE文について

画面を表示するときには表示ファイルのレコード名を指定したWRITE文を、画面からデータを読み込むときには表示ファイルを指定したREAD文を使います。

WRITE文を実行する前には、画面出力に使用する画面定義体の名前をFORMAT句に指定したデータ名に設定し、出力の対象となる画面定義体の項目群名をGROUP句に指定したデータ名に設定します。

また、画面出力の処理種別(モード)をPROCESSING MODE句に指定したデータ名に設定します。

入力の対象となる画面定義体の項目群名と入力の処理種別(モード)が出力時と異なるときは、READ文を実行する前に、画面定義体の項目群名をGROUP句に指定したデータ名に設定し、処理種別(モード)をPROCESSING MODE句に指定したデータ名に設定します。

画面定義入力中にファンクションキーなどが押されると、READ文が完了してCOBOLプログラムに入力結果の情報が通知されます。そのとき、SELECTED FUNCTION句に指定したデータ名にアテンション情報が通知されます。

3.2.3.2 帳票機能

COBOLプログラムの記述例

```
OPEN OUTPUT プリンタファイル.
MOVE "ADDRPRT" TO PRT-FORMAT.
MOVE "@ALLF"   TO PRT-GROUP.
MOVE "  "      TO PRT-MODE.
WRITE 住所録印刷帳票.
CLOSE プリンタファイル.
```

帳票機能のWRITE文について

帳票を出力するときには、表示ファイルのレコード名を指定したWRITE文を実行します。

WRITE文を実行する前には、印刷に使用する帳票定義体の名前をFORMAT句に指定したデータ名に設定し、印刷の対象となる帳票定義体の項目群名をGROUP句に指定したデータ名に設定します。

また、帳票出力の処理種別(モード)をPROCESSING MODE句に指定したデータ名に設定します。

3.2.4 エラー処理

表示ファイルの各命令(OPEN文、WRITE文、READ文、CLOSE文)の実行結果は、FILE STATUS句に指定したデータ名に通知されます。FILE STATUS句に指定したデータ名のうち2桁のデータ名の領域には、成功時は「00」、「04」が通知され、不成功時は「90」、「99」、「9E」のいずれかが通知されます。また、4桁のデータ名の領域には、詳細結果として上記の4種類のエラーにMeFtの通知コードが付加されたものが通知されます。例えば、MeFtの通知コードが「22」であった場合、FILE STATUS句に指定した4桁のデータ領域には「9022」が通知されます。

入出力文の後に、このデータ名の内容をチェックする文を記述することによって、プログラムで入出力文の結果に応じた処理手続きを実行することができます。

FILE STATUS句の使用例を次に示します。

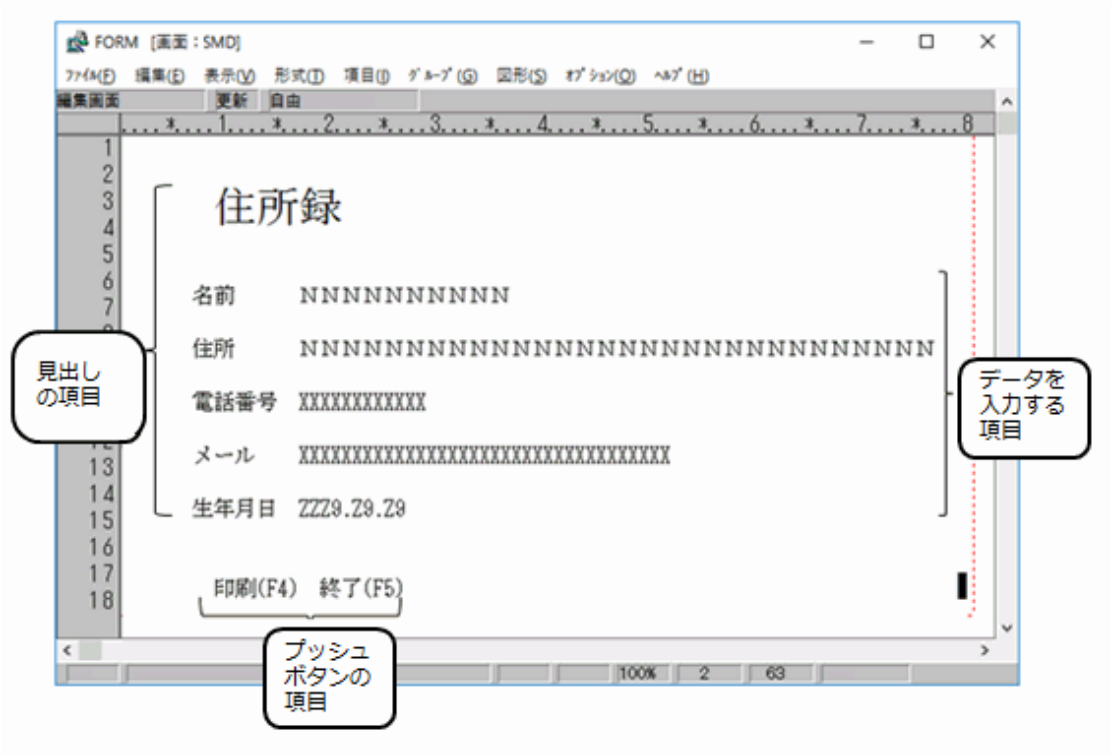
```
SELECT  ディスプレイファイル ASSIGN TO  GS-DSPFILE
    :
  FILE  STATUS         IS  DSP-STATUS1 DSP-STATUS2.
    :
WORKING-STORAGE SECTION.
01  DSP-STATUS1       PIC X(02).
01  DSP-STATUS2       PIC X(04).
    :
PROCEDURE DIVISION.
OPEN ディスプレイファイル.
    :
WRITE 住所録入力画面.
IF DSP-STATUS2 NOT = "0000" THEN
    CLOSE ディスプレイファイル
END-IF.
```

3.3 画面帳票定義体の作成

画面帳票定義体の作成について、基本的な操作を説明します。操作の詳細な説明については、FORMのマニュアルをご参照ください。

3.3.1 画面定義体の作成

ここでは、次に示すような画面定義体を作成する方法を説明します。



次に示す手順で画面定義体を作成していきます。

1. 画面定義体の属性設定

   画面定義体の定義体サイズ（縦幅／横幅）など、画面定義体全体に対する属性を設定します。

2. 項目の配置と属性設定

   データを入力する項目の配置など、画面定義体のレイアウトを作成します。また、配置した項目に対する属性を設定します。

3. アテンション情報の設定

   キーボードで押されたファンクションキーをプログラムに通知する情報（アテンション情報）を設定します。

4. 項目のボタン化

   画面定義体に配置された項目のうち、一部の項目をボタンとして使用する指定を行います。

3.3.1.1 FORMの起動

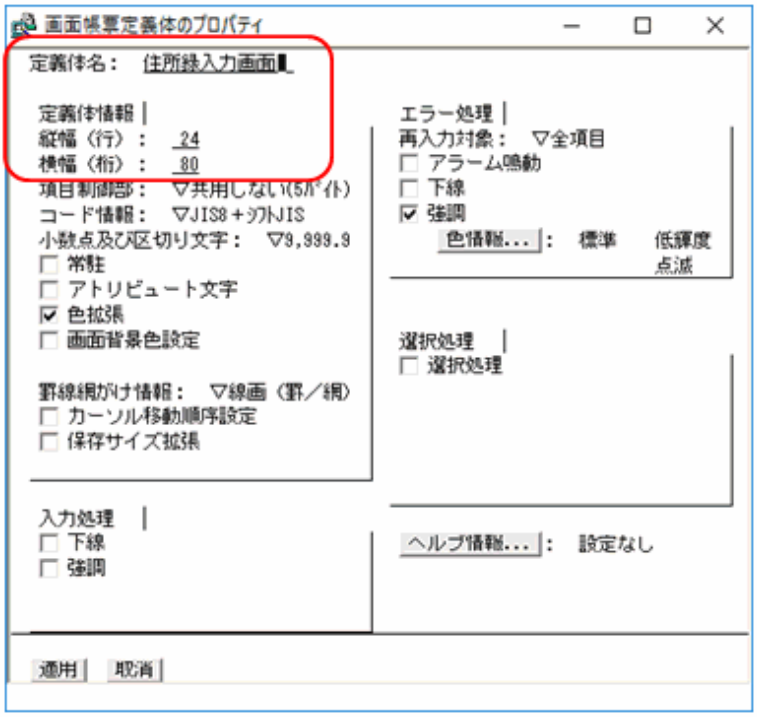
[スタート] > お使いのNetCOBOL製品名 > [FORM]を選択して、FORMを起動します。

FORMの編集画面が表示されます。

3.3.1.2 画面定義体の属性設定

1. [ファイル]メニューから、[新規作成] > [画面定義体]を選択します。
2. [ファイル]メニューから、[プロパティ] > [画面帳票定義体]を選択すると、画面定義体のプロパティ画面が表示されます。
3. プロパティの画面で、以下に示す内容を設定します。

設定箇所	設定内容
定義体名	住所録入力画面
縦幅(行)	20
横幅(桁)	80



4. [適用]ボタンをクリックすると、編集画面に戻ります。

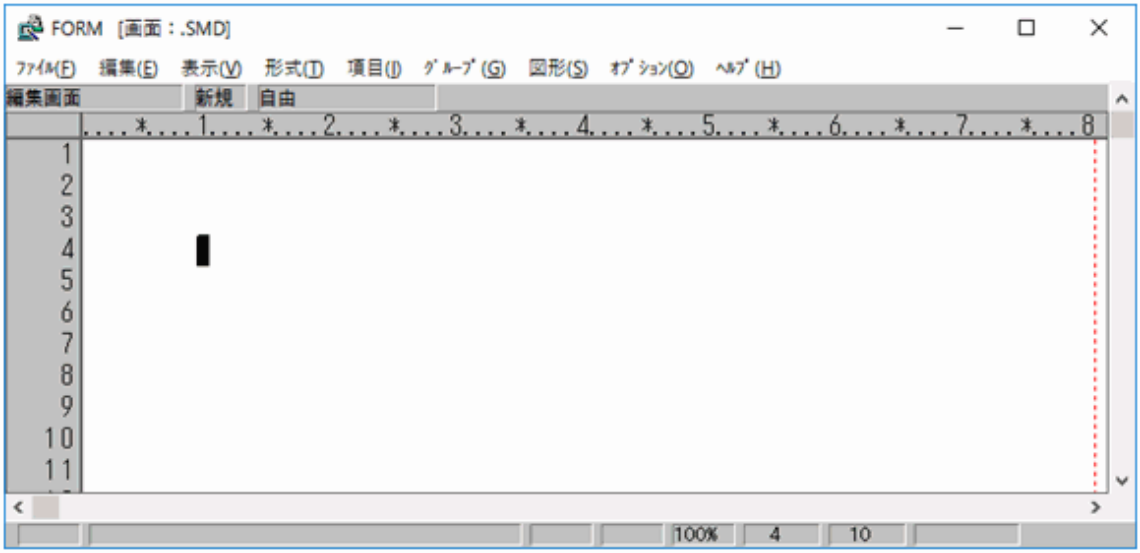
3.3.1.3 項目の配置と属性設定

項目を配置し、その属性を設定します。

項目を配置するには、FORMの［項目］メニューから配置したい項目の種類を選択し、表示されるプロパティの画面で属性を設定します。

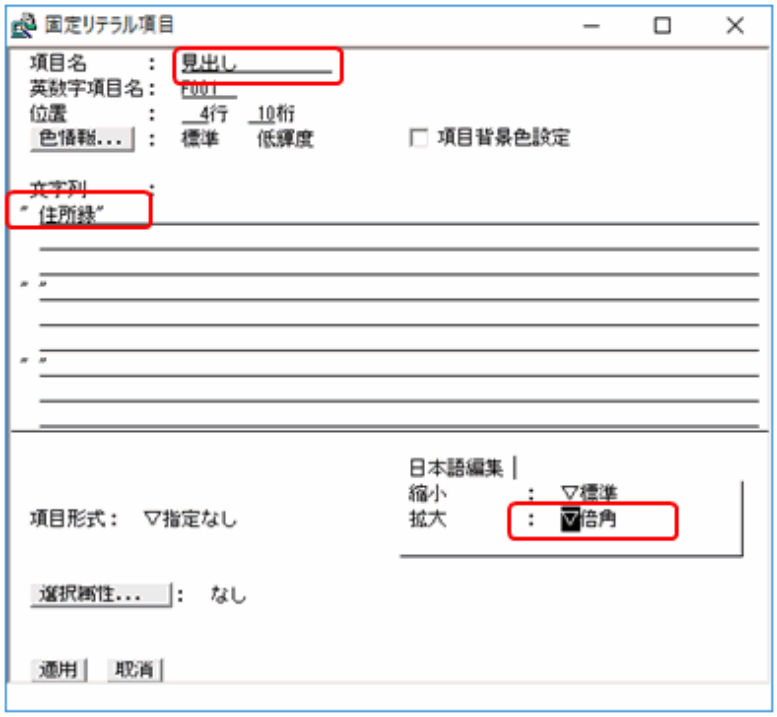
ここでは、例として、見出しの「住所録」という文字列の項目を配置し、属性を設定する方法を説明します。

1. 編集画面で、見出しを配置したい位置にカーソルを移動します。

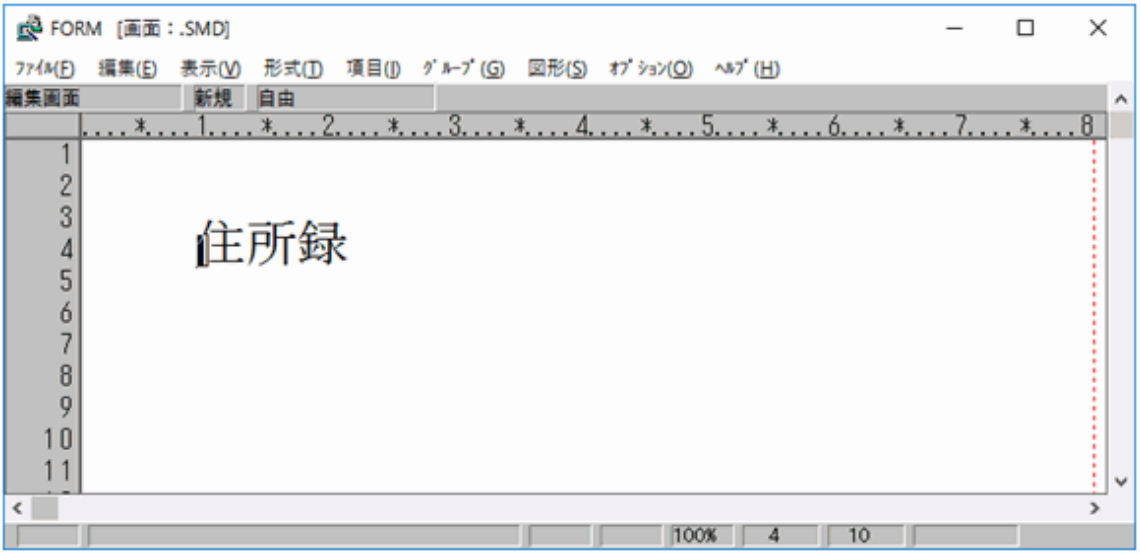


2. 見出しのように表示する文字が固定されている項目を「固定リテラル項目」といいます。[項目]メニューから「固定リテラル」を選択します。
3. 固定リテラル項目のプロパティ画面が表示されますので、次に示す内容を設定します。

設定箇所	設定内容
項目名	見出し
文字列	住所録
日本語編集の「拡大」	倍角



4. [適用]ボタンを選択すると、プロパティ画面が終了し、見出しの項目が編集画面上に配置されます。

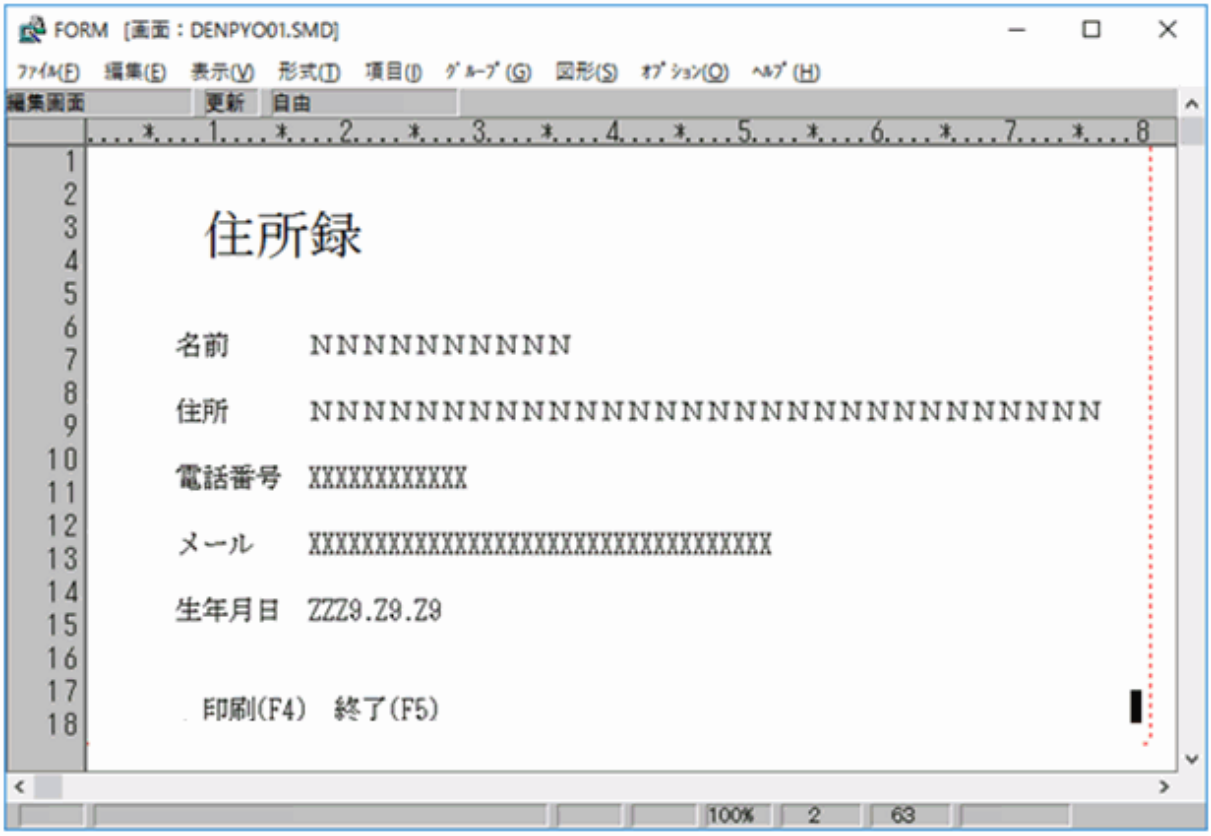


5. その他の項目は、次に示す内容で設定します。

[項目]メニューでの項目の種類	項目名	文字列	横幅(文字数)	桁数	編集形式
固定リテラル	名前見出し	名前			
日本語	名前		10		
固定リテラル	住所見出し	住所			
日本語	住所		30		
固定リテラル	電話番号見出し	電話番号			
英数字	電話番号		12		
固定リテラル	メール見出し	メール			

[項目]メニューでの項目の種類	項目名	文字列	横幅(文字数)	桁数	編集形式
英数字	メール		35		
固定リテラル	生年月日見出し	生年月日			
数字	生年月日			8	ZZZ9.Z9.Z9、全ゼロサプレスあり
固定リテラル	印刷ボタン	印刷(F4)			
固定リテラル	終了ボタン	終了(F5)			

全ての項目の配置、属性設定が終了すると、以下のような状態になります。



3.3.1.4 アテンション情報の設定

アテンション情報とは

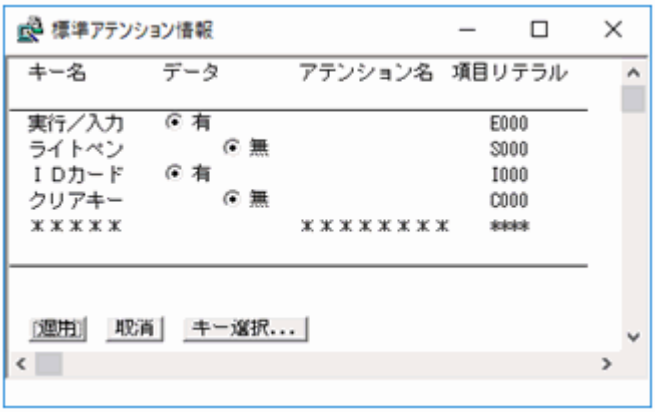
アテンション情報とは、キーボードで押されたファンクションキーをプログラムに通知する情報です。

画面定義体では、各ファンクションキーに対応するアテンション情報を設定します。プログラムでは、通知されるアテンション情報を参照すると、押されたファンクションキーを知ることができるため、アテンション情報によって処理を分けます。

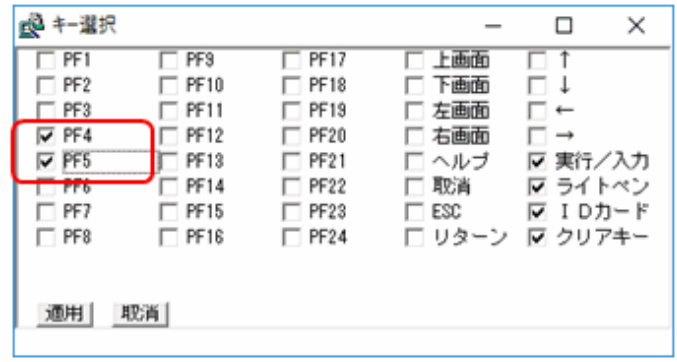
利用者が設定するアテンション情報には、標準アテンション情報と拡張アテンション情報がありますが、ここでは、標準アテンション情報の設定方法を説明します。

標準アテンション情報の設定

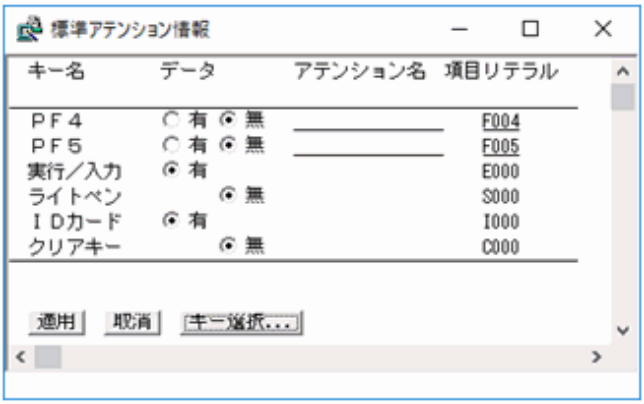
1. [形式]メニューから[標準アテンション情報]を選択し、標準アテンション情報の設定画面を表示します。



2. 標準アテンション情報の設定画面の[キー選択]ボタンを選択し、キー選択の画面を表示します。
3. キー選択の設定画面で、「PF4」と「PF5」をチェックします。

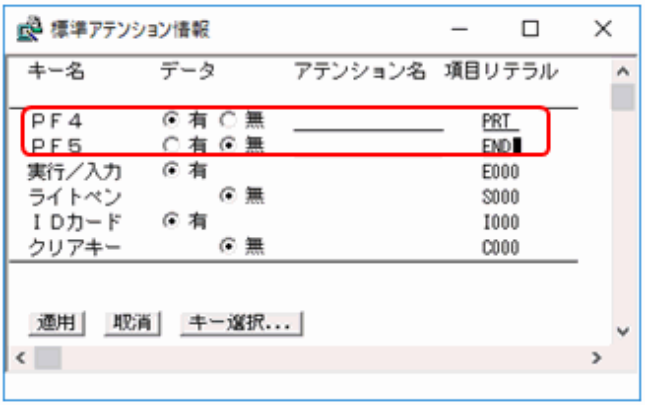


4. ［適用］ボタンを選択すると、キー選択の設定画面でチェックしたキーが、標準アテンション情報の設定画面に表示されます



5. 標準アテンション情報の設定画面では次に示す内容を設定し、［適用］ボタンを選択します。

キー名	データ	項目リテラル
PF4	有	PRT
PF5	無	END



データの有無について

ファンクションキーが押されたとき、画面に入力されたデータを有効にするか、無効にするかを指定します。

「有」の場合、入力されたデータを有効とし、押されたファンクションキーのアテンション情報と共にプログラムに通知します。

「無」の場合、押されたファンクションキーのアテンション情報だけをプログラムに通知し、入力データがあっても通知しません。

項目リテラルについて

ファンクションキーを識別する名前です。プログラムでは、READ文のあとに通知される項目リテラルを判断し、押されたファンクションキーを判断できます。

3.3.1.5 項目のボタン化

画面定義体に定義された項目をボタン化します。なお、画面定義体では、ボタンとしてプッシュボタンのほかにチェックボックス、ラジオボタンが設定できます。

本章で作成する画面定義体では、プッシュボタンを設定します。

プッシュボタンは、次に示す手順で設定します。

- 定義体でのボタン利用の指定
- 項目をボタンとして使用する指定
- 使用するボタンの種類や属性の指定

定義体でのボタン利用の指定

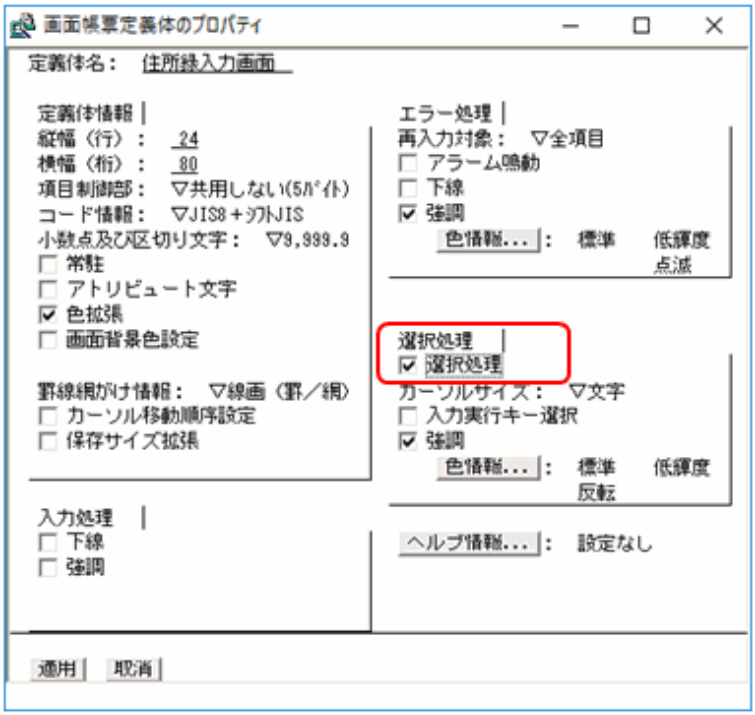
画面定義体でボタンを使用するには、画面定義体に対して選択処理を行うかどうかを設定する必要があります。

選択処理とは、画面内の項目に対する操作の通知をすることです。例えば、ボタンを押すといった項目の操作が選択処理にあたります。

選択処理を行うかどうかは、画面定義体のプロパティで指定します。

1. [ファイル]メニューから[プロパティ] > [画面帳票定義体]を選択し、画面定義体のプロパティ画面を表示します。

2. 表示されたプロパティ画面の「選択処理」をチェックします。



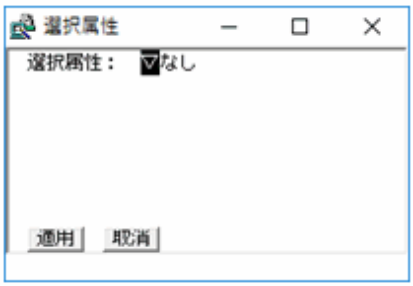
3. ［適用］ボタンをクリックし、画面定義体のプロパティ画面を終了します。

項目をボタンとして使用する指定

プッシュボタンにする項目に選択属性を設定し、ボタン化する宣言をします。

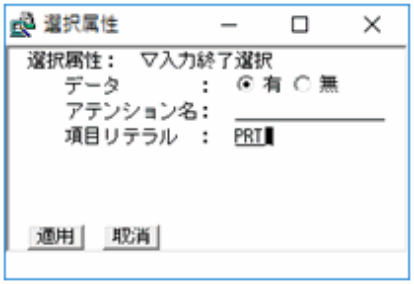
本章で作成する画面定義体では、「印刷(F4)」と「終了(F5)」という文字列の2つの項目をプッシュボタンにします。

1. 画面定義体の編集画面で「印刷(F4)」を選択し、右クリックして表示されたコンテキストメニューから[プロパティ]を選択します。
2. 表示されたプロパティ画面の［選択属性］ボタンを選択し、選択属性の設定画面を表示します。



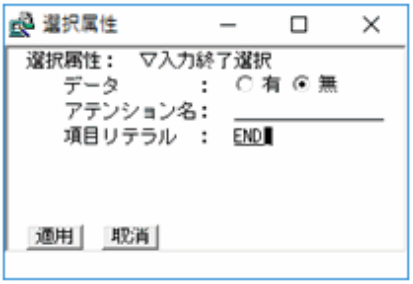
3. 選択属性の設定画面では、そのボタンが押されたときに通知される情報を設定します。ここでは、ボタンが押されたときに通知される項目リテラルを設定するため、次に示す内容を設定します。

設定箇所	設定内容
選択属性	入力終了選択
データ	有
項目リテラル	PRT



4. ［適用］ボタンを選択し、選択属性の設定画面を終了します。
5. 同様に、「終了(F5)］」項目は、次に示す内容で選択属性を設定します。

設定箇所	設定内容
選択属性	入力終了選択
データ	無
項目リテラル	END



使用するボタンの種類や属性の指定

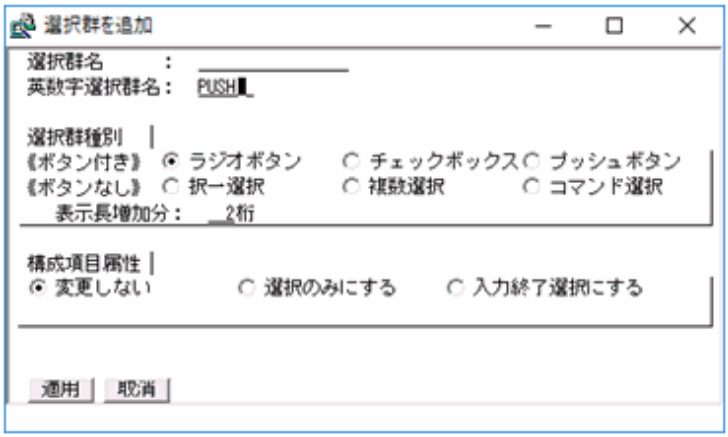
選択項目（選択属性を持った項目）は選択群を作成してグループ単位で利用します。

ボタンの種類や属性も、選択群に対してグループ単位で設定します。また、選択群を設定した後、選択群に属する項目を指定します。

選択群の追加

1. [グループ]メニューから[選択群を追加]を選択し、選択群の追加画面を表示します。
2. 選択群の追加画面では、次に示す内容を設定します。

設定箇所	設定内容
英数字選択群名	PUSH
選択群種別	プッシュボタン
構成項目属性	変更しない

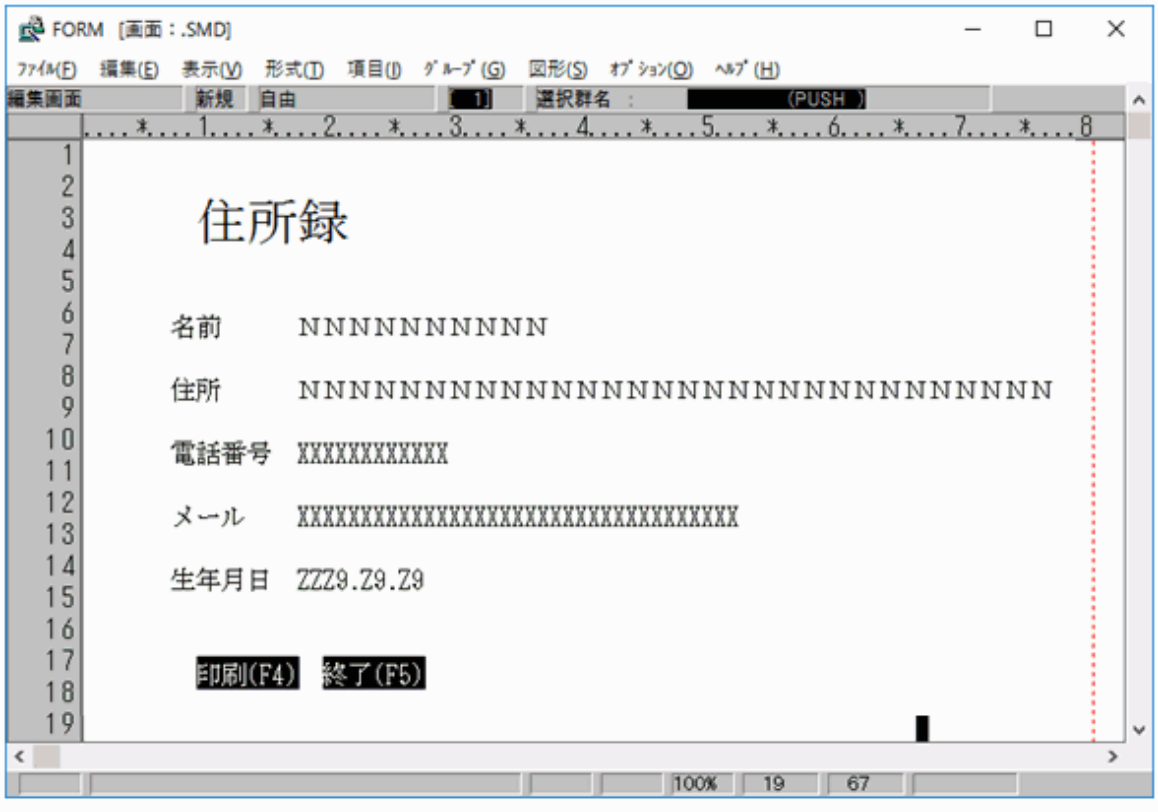


3. ［適用］ボタンを選択します。選択群の追加画面が終了し、編集画面に戻ります。

項目の追加

1. 編集画面は、選択群を構成する項目を表示する状態（構成項目表示）になっています。項目を選択群に追加するには、「印刷(F4)」の項目を選択し、右クリックして表示されるコンテキストメニューから[項目を追加]を選択します。
   → 選択群に含まれる項目は黒く反転します。

2. プッシュボタンにする項目「終了(F5)」も、同じように選択群に追加します。

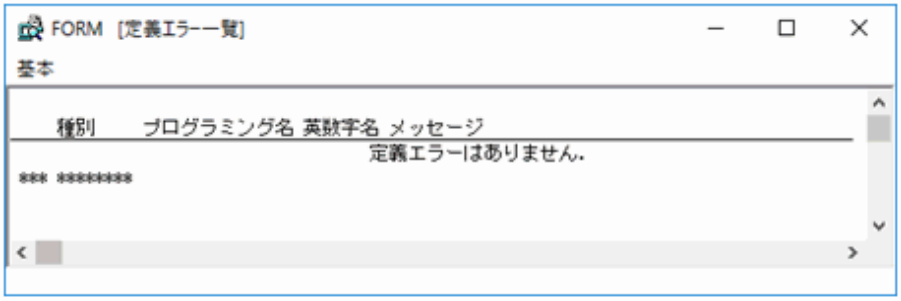


3. [表示]メニューから[構成項目表示を解除]を選択し、選択群の構成項目の表示状態を解除します。

3.3.1.6 定義の正当性の確認

定義した内容にエラーがないか、定義エラー一覧画面で確認します。

定義エラー一覧画面は、[オプション]メニューから[定義エラー一覧]を選択することで表示されます。



3.3.1.7 画面定義体の保存

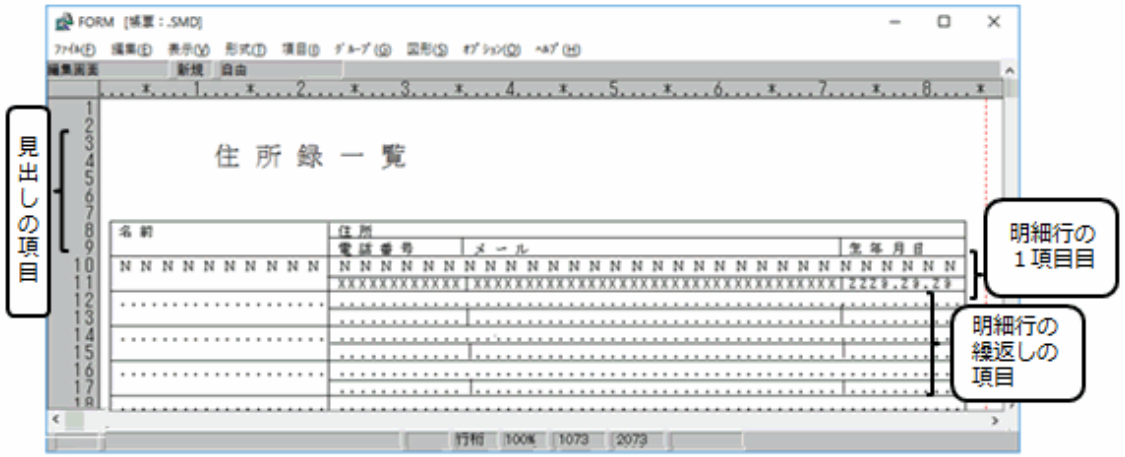
[ファイル]メニューから[閉じる] > [画面帳票定義体]を選択すると、[名前を付けて保存]ウィンドウが表示されます。

この例では、「C:¥EDUCATION」を選択し、「ADDRDSP.SMD」という名前を付けて画面定義体を保存します。

3.3.2 帳票定義体の作成

ここでは、次に示すような帳票定義体を作成する方法を説明します。

なお、画面定義体の作成と同じ方法の事項については、説明を省略しています。



次に示す手順で帳票定義体を作成していきます。

1. 帳票定義体の属性設定

   帳票定義体の大きさ(縦幅／横幅)など、帳票定義体全体に対する属性を設定します。

2. 項目の配置と属性設定

   データを出力する項目の配置など、帳票定義体のレイアウトを作成します。また、配置した項目に対する属性を設定します。

3. 繰返しの設定

   同じ属性を持つ項目を複数設定します。

4. 罫線の定義

   帳票定義体に罫線を定義します。

5. 罫線のオーバレイ定義

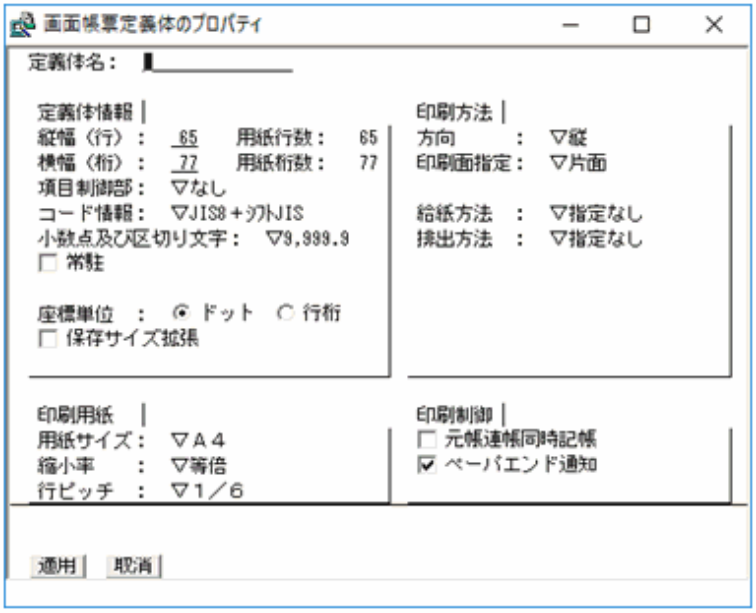
   帳票定義体に定義した罫線を、オーバレイ定義体の罫線にします。

3.3.2.1 FORMの起動

画面定義体の作成時と同じ方法で起動します。

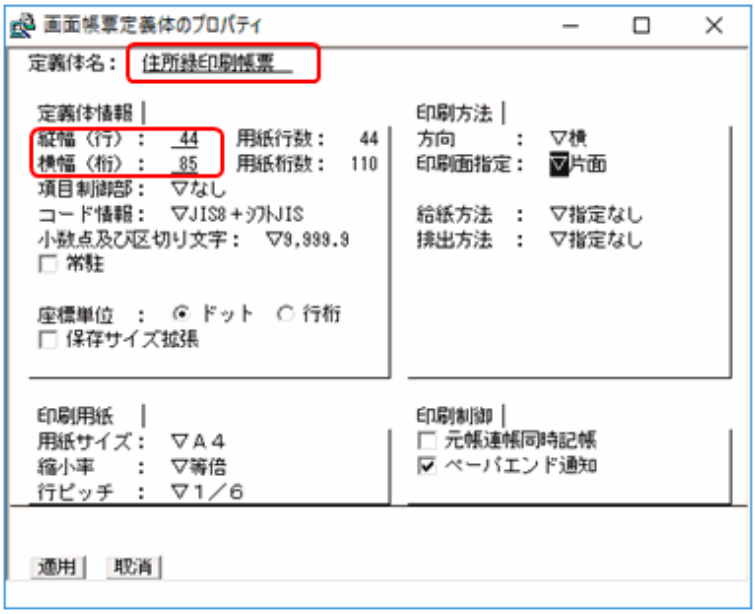
3.3.2.2 帳票定義体の属性設定

1. [ファイル]メニューから、[新規作成] > [帳票定義体]を選択します。
2. [ファイル]メニューから、[プロパティ] > 「画面帳票定義体」を選択します。
   → 帳票定義体のプロパティ画面が表示されます。



3. プロパティの画面で、以下に示す内容を設定します。

設定箇所	設定内容
定義体名	住所録印刷帳票
縦幅(行)	44
横幅(桁)	85
方向	横



4. [適用]ボタンをクリックすると、編集画面に戻ります。

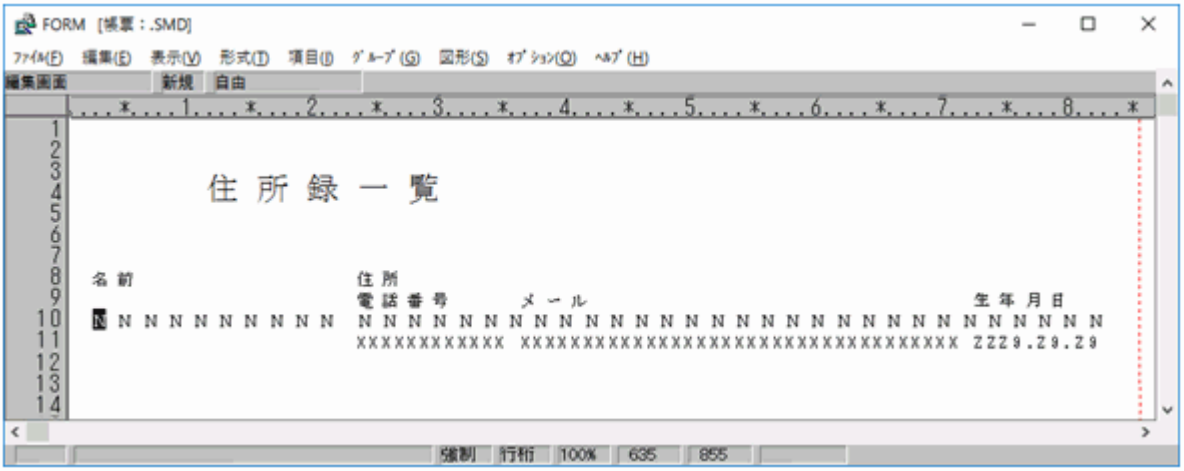
3.3.2.3 項目の配置と属性設定

項目の配置と属性設定は、画面定義体の作成と同じくFORMの[項目]メニューから行います。

各項目は、次に示す内容で設定します。

「項目」メニューでの項目の種類	項目名	文字列	文字数	桁数	和文書体の拡大	編集形式
固定リテラル	見出し	住所録一覧			倍角	
固定リテラル	名前見出し	名前				
固定リテラル	住所見出し	住所				
固定リテラル	電話番号見出し	電話番号				
固定リテラル	メール見出し	メール				
固定リテラル	生年月日見出し	生年月日				
日本語	名前		10			
日本語	住所		30			
英数字	電話番号		12			
英数字	メール		35			
数字	生年月日			8		ZZZ9.Z9.Z9、全ゼロサプレスあり

ここまでの項目の配置、属性設定が終了すると、以下のような状態になります。

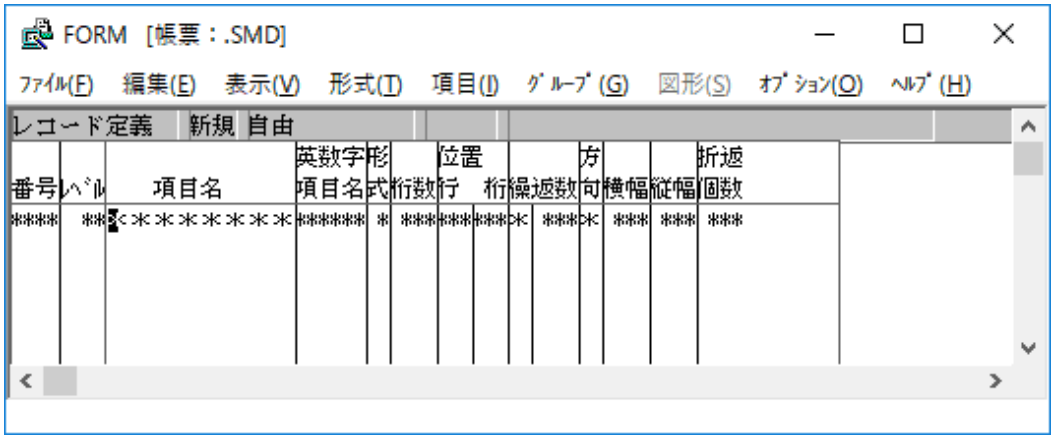


3.3.2.4 繰返しの設定

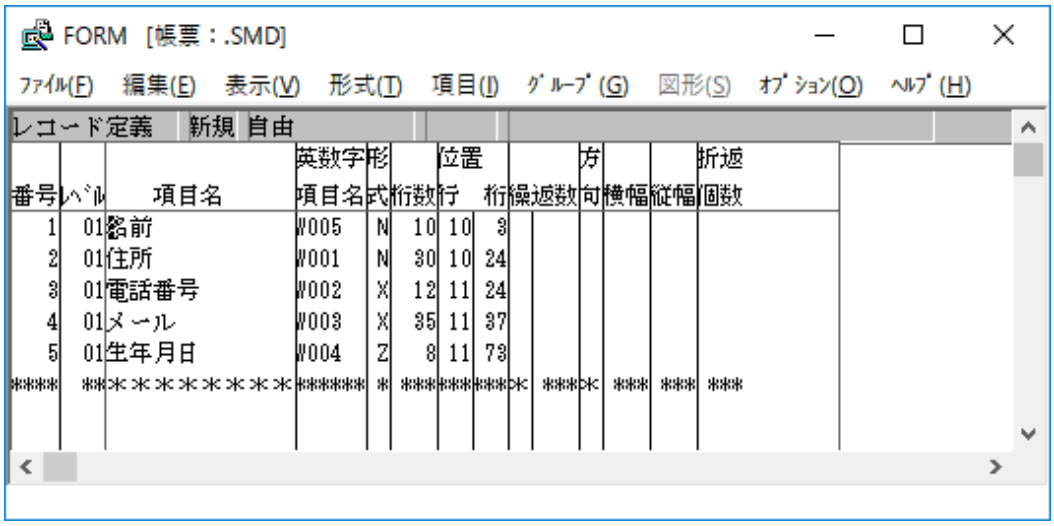
繰返しの設定とは、同じ属性を持つ項目を複数設定することをいいます。繰返しは、単一の項目にも、複数の項目から形成される集団項目にも行うことができます。本章で作成する帳票定義体では、プログラムからデータ出力を行う全ての項目を繰返し定義します。

繰返しの設定方法を説明します。なお、繰返しの設定は、帳票定義体だけではなく画面定義体でも行うことができます。

1. FORMの[表示]メニューから「レコード定義へ」を選択し、レコード定義画面を表示します。

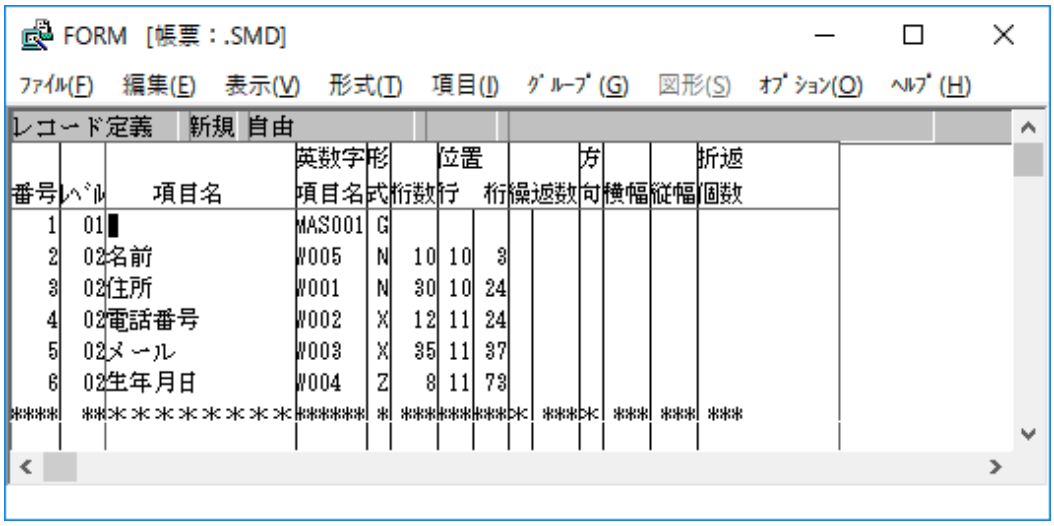


2. [オプション]メニューから[基本レコードを生成]を選択します。
   → 帳票定義体に定義したデータ出力用の項目がレコード定義画面に表示されます。



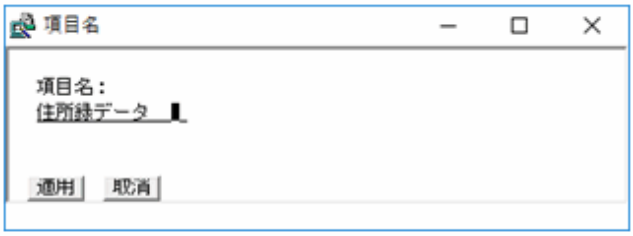
3. [編集]メニューから[全てを選択]を選択します。
   → 全ての項目が選択されて反転します。

4. [項目]メニューから[集団項目定義]を選択します。
   → レコード定義画面の先頭にレベル01の項目が挿入され、3.で選択した項目のレベルが02に変更されています。



5. レベル01の項目を選択し、[項目]メニューから「項目名」を選択します。

6. 項目名の入力画面が表示されるので、項目名として「住所データ」を入力します。この名前が集団項目の名前になります。

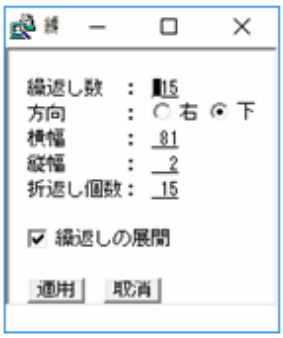


7. 項目名の入力画面の[適用]ボタンをクリックして項目名の入力画面を閉じます。
   → レコード定義画面では、6. で入力した集団項目の項目名が表示されています。

8. 集団項目の項目名を選択し、[項目]メニューから[繰返し定義]を選択します。

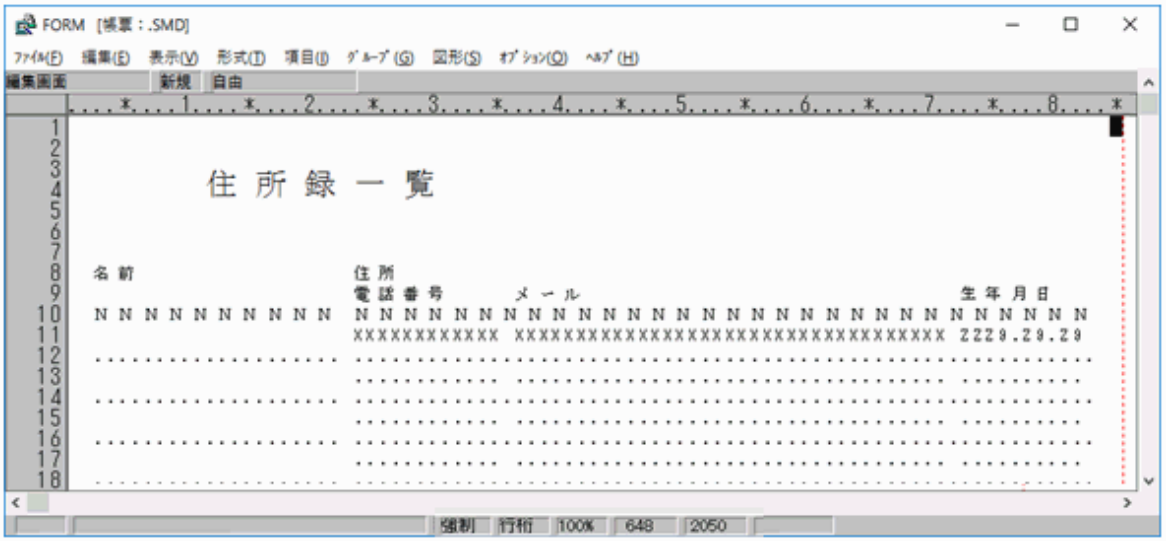
9. 繰返し定義の設定画面が表示されるので、次に示す内容を設定します。

設定箇所	設定内容
繰返し数	15
方向	下
折返し個数	15
繰返しの展開のチェック	チェックあり



10. 繰返し定義の設定画面の[適用]ボタンをクリックして、繰返し定義の設定画面を閉じます。
    → レコード定義画面では、9.で入力した繰返し定義の設定が表示されています。

11. [表示]メニューから[編集画面へ戻る]を選択します。
    → 編集画面に、繰返しで設定した項目が[・・]で表示されます。



3.3.2.5 罫線の定義

罫線を定義する方法を説明します。

罫線は、帳票定義体に定義する方法とオーバレイ定義体に定義する方法があります。ここでは、帳票定義体に定義する方法を説明します。

なお、画面定義体にも同じ方法で罫線が定義できます。

1. FORM編集画面の左側にある[シンボル]メニューで図形を選択します。

本章で作成する帳票では枠、水平線および垂直線を定義します。

枠を定義する際はシンボルメニューで「□」の図形を選択します。水平線は「—」、垂直線は「｜」のシンボルメニューの図形を選択します。

2. 作図状態（マウスポインタがクロスポインタに変わります）になるので、作図範囲を開始位置から終了位置までドラッギングします。

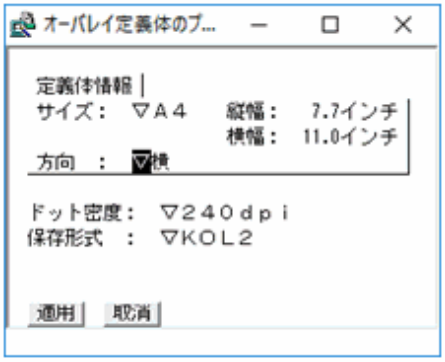
   全ての罫線を作図し終わると、以下のような状態になります。

3.3.2.6 オーバレイ定義体の作成（罫線のオーバレイ定義）

帳票定義体にオーバレイ定義体を重畳して新規作成することにより、帳票定義体に定義された罫線をオーバレイ定義体の罫線にできます。

帳票定義体にオーバレイ定義体を重畳して作成する方法を説明します。

1. オーバレイ定義体を重畳して作成するには、[ファイル]メニューから[新規作成] > [オーバレイ定義体]を選択します。
   → FORM編集画面のタイトルバーにオーバレイも表示されます。
2. [ファイル]メニューから[プロパティ] > 「オーバレイ定義体」を選択します。
   → オーバレイ定義体のプロパティ画面を表示します。
3. 帳票定義体では用紙の指定をA4横にしているので、オーバレイ定義体もプロパティ画面でA4横を指定します。



4. [適用]ボタンをクリックして、オーバレイ定義体のプロパティ画面を閉じます。

次に、帳票定義体の印刷時に重ねて使用するオーバレイ定義体名を帳票定義体に設定します。

1. [形式]メニューから[オーバレイ名の指定]を選択します。
2. オーバレイ名の指定画面が表示されるので、格納時に命名を予定しているオーバレイ定義体の名前「ADDRPRT.OVD」から拡張子を除いた名称を指定します。

3. [適用]ボタンをクリックして、オーバレイ名の指定画面を閉じます。

3.3.2.7 定義の正当性の確認

画面定義体の作成時と同じ方法で定義エラーの確認を行います。

3.3.2.8 帳票定義体の保存

[ファイル]メニューから[閉じる] > [画面帳票定義体]を選択します。

[名前を付けて保存]ウィンドウで「C:¥EDUCATION」を選択し、「ADDRPRT.SMD」という名前を付けて帳票定義体を保存します。

3.3.2.9 オーバレイ定義体の保存

[ファイル]メニューから[閉じる] > [オーバレイ定義体]を選択します。

[名前を付けて保存]ウィンドウで、「C:¥EDUCATION」を選択し、「ADDRPRT.OVD」という名前を付けてオーバレイ定義体を保存します。

参考

FORMでは、帳票定義体を作成せずにオーバレイ定義体だけを編集することが可能です。

3.4 プロジェクトの作成

NetCOBOL Studioによる、プロジェクト管理機能を使用した画面帳票アプリケーションの作成方法について説明します。

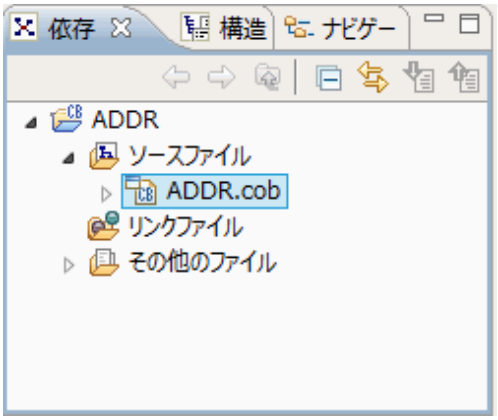
なお、画面帳票アプリケーションの作成方法は、“第2章 NetCOBOLアプリケーション開発の基礎”で作成したアプリケーションとほぼ同じです。

以下の手順でプロジェクトを作成します。[参考]“2.3 プロジェクトの作成”

- NetCOBOL Studioの起動
- プロジェクトの作成
- 実行ファイルの登録

画面帳票アプリケーション固有の設定はありません。

本章で作成するアプリケーションでは、COBOLプロジェクトを「ADDR」、ソースファイルを「ADDR.cob」、実行ファイルを「ADDR.exe」として作成します。



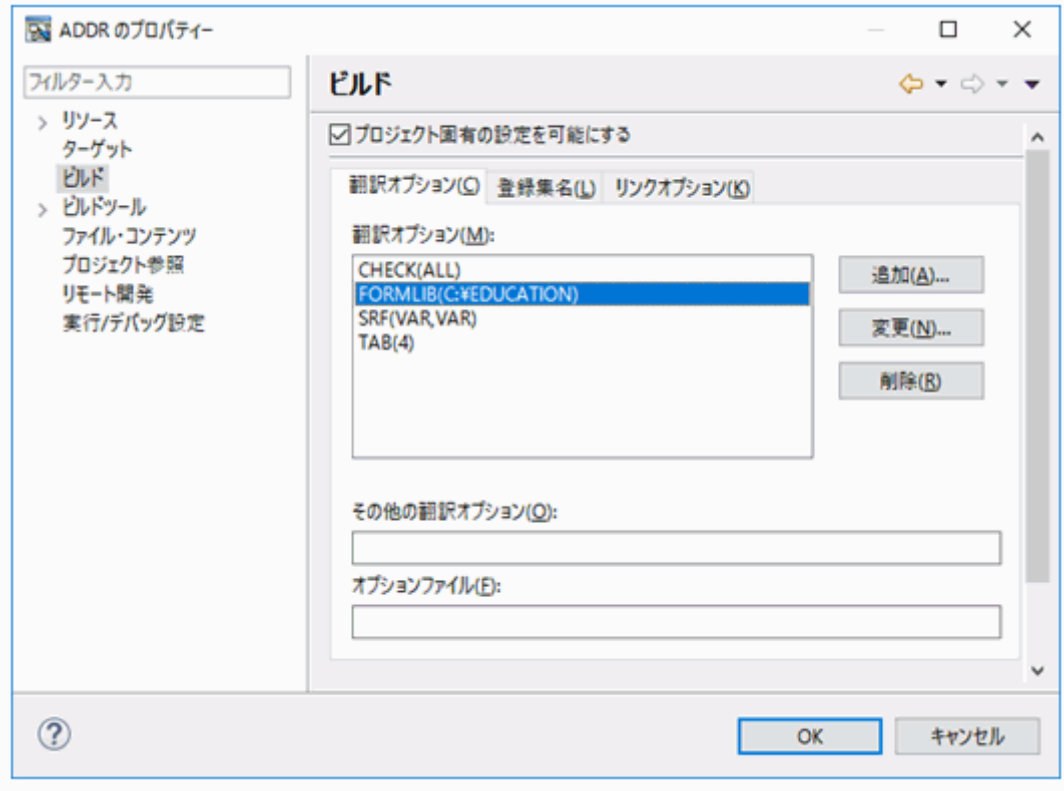
3.4.1 各種ファイルの登録

実行ファイルを作成するのに必要となるファイルとして、ソースファイルと画面帳票定義体をプロジェクトに登録します。

ソースファイルの登録は、画面帳票アプリケーション固有の設定はありません。一方、画面帳票定義体の登録は画面帳票アプリケーション固有です。

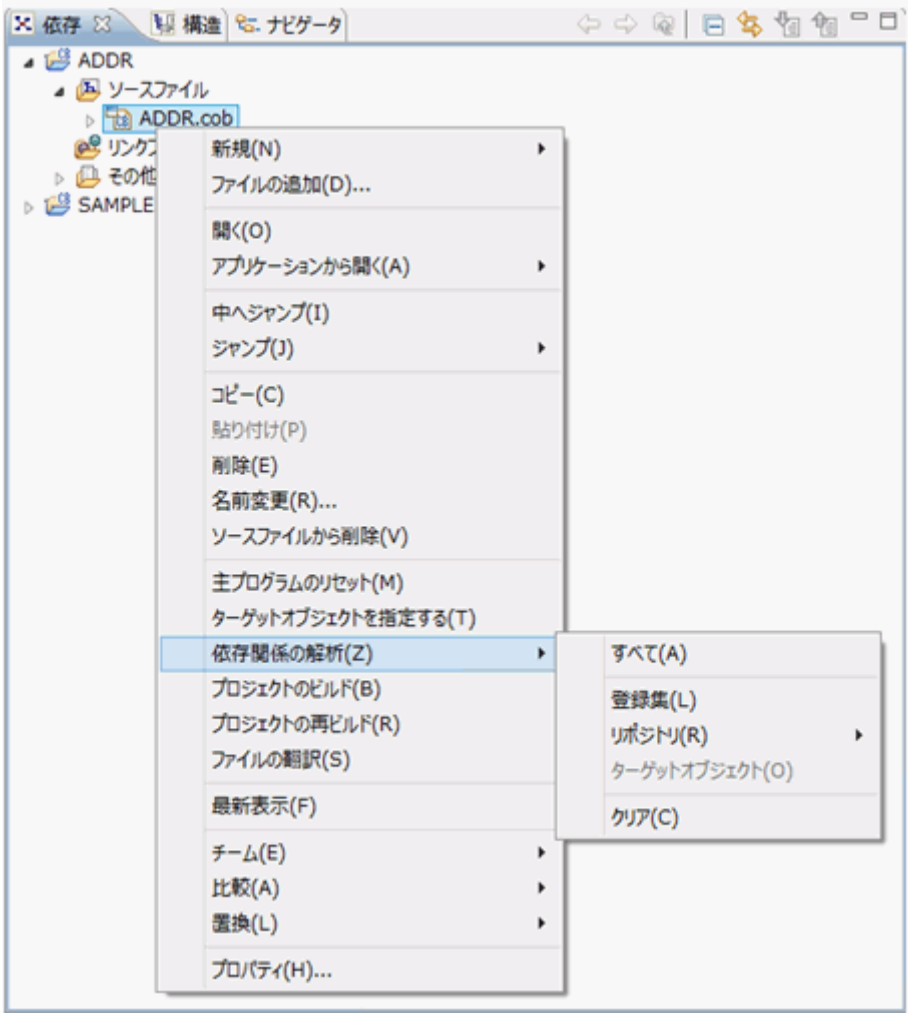
以下に、画面帳票定義体を登録する方法を説明します。

1. プロジェクトのプロパティから、翻訳オプションFORMLIBを追加し、画面定義体のフォルダーに「C:¥EDUCATION」を指定します。

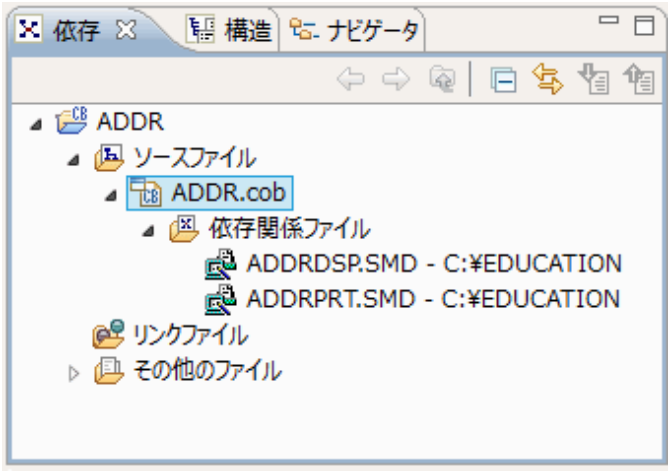


2. [OK]ボタンをクリックします。
   → プロジェクトのクリーンの[確認]メッセージが表示されるため、[はい]をクリックします。

3. ［ADDR.cob］を右クリックし、［依存関係の解析］-［すべて］を選択します。



4. ［依存関係フォルダー］に定義体が追加されます。



参考

この場合、画面定義体はワークスペース外にあるため、NetCOBOL Studioから操作（編集・参照）することはできません。

画面定義体をNetCOBOL Studioから操作（編集・参照）する必要がある場合は、定義体をワークスペース配下に格納してください。

3.5 ビルド

- 翻訳オプションの設定
- リンクオプションの設定

画面帳票アプリケーション固有の設定はありません。

なお、画面帳票定義体に関する翻訳オプションとして次のようなものがあり、必要に応じて設定します。本章で作成するアプリケーションでは、設定する必要はありません。

- FORMEXT（画面帳票定義体ファイルの拡張子）
- FORMLIB（画面帳票定義体ファイルのフォルダー）

3.5.1 ビルド操作

画面帳票アプリケーション固有の設定はありません。

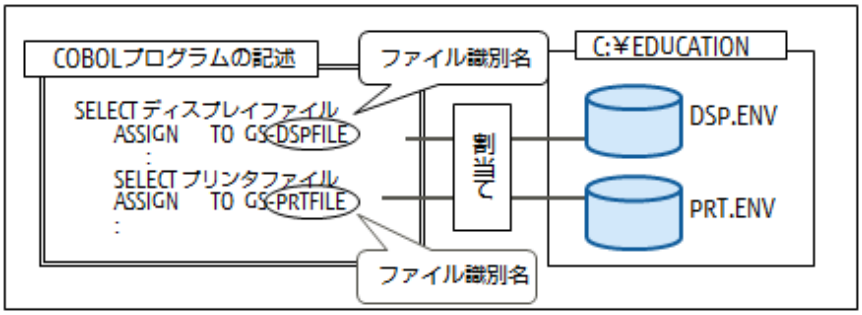
“第2章 NetCOBOLアプリケーション開発の基礎”で作成したアプリケーションと同じようにビルド操作を行うと、「ADDR.EXE」が生成されます。

3.6 実行

3.6.1 実行環境情報の設定

画面帳票アプリケーションを実行する場合、実行環境情報として、画面入出力用のファイルのASSIGN句で定義されたファイル識別名にウィンドウ情報ファイル名を設定し、帳票印刷用のファイルのASSIGN句で定義されたファイル識別名にプリンタ情報ファイル名を設定します。

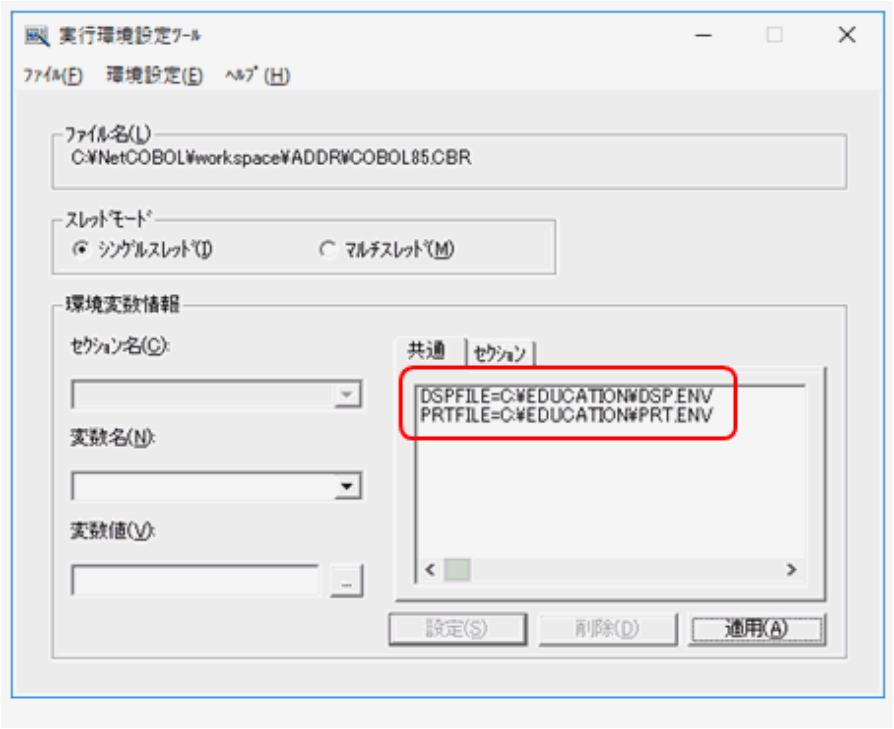
本章のアプリケーションでは、画面入出力の表示ファイルのファイル識別名「DSPFILE」と「C:¥EDUCATION」に格納されているウィンドウ情報ファイル「DSP.ENV」を割り当て、帳票印刷の表示ファイルのファイル識別名「PRTFILE」と「C:¥EDUCATION」に格納されているプリンタ情報ファイル「PRT.ENV」を割り当てます。



実行環境設定ツールの起動や操作には、画面帳票アプリケーション固有の部分はありません。

“第2章 NetCOBOLアプリケーション開発の基礎”で作成したアプリケーションと同じ手順で「C:¥EDUCATION」に「COBOL85.CBR」を作成し、実行環境設定ツールで次に示す指定を行います。

変数名	変数値
DSPFILE	C:¥EDUCATION¥DSP.ENV
PRTFILE	C:¥EDUCATION¥PRT.ENV



3.6.2 プログラムの実行

プログラムの実行についても、画面帳票アプリケーション固有の部分はありません。

プログラムを実行し、画面の入出力および帳票出力を確認します。

3.7 デバッグ

NetCOBOL Studioからの画面帳票アプリケーションのデバッグについて説明します。

3.7.1 デバッグの準備

画面帳票アプリケーション固有の設定はありません。

“第2章 NetCOBOLアプリケーション開発の基礎”で作成したアプリケーションと同じ方法で、プロジェクトをビルド（または再ビルド）します。

3.7.2 デバッグの開始

画面帳票アプリケーション固有の設定はありません。

3.7.3 デバッグ操作

画面帳票アプリケーション固有の設定はありません。

画面帳票アプリケーションでも、ブレークポイントを設定してのデバッグなどの“第2章 NetCOBOLアプリケーション開発の基礎”で説明したデバッグ操作が行えます。

3.7.4 デバッグの終了

デバッグパースペクティブの［実行］メニューから「終了」を選択し、デバッグを終了します。

第4章 MeFt/Webアプリケーションの構築

本章では、画面帳票アプリケーションをWeb環境で動作させることができるMeFt/Webを説明します。また、画面帳票アプリケーションをMeFt/Webアプリケーションとして構築する方法を説明します。

4.1 概要

MeFt/Webの概要および本章で構築するMeFt/Webアプリケーションを説明します。

4.1.1 MeFt/Webの概要

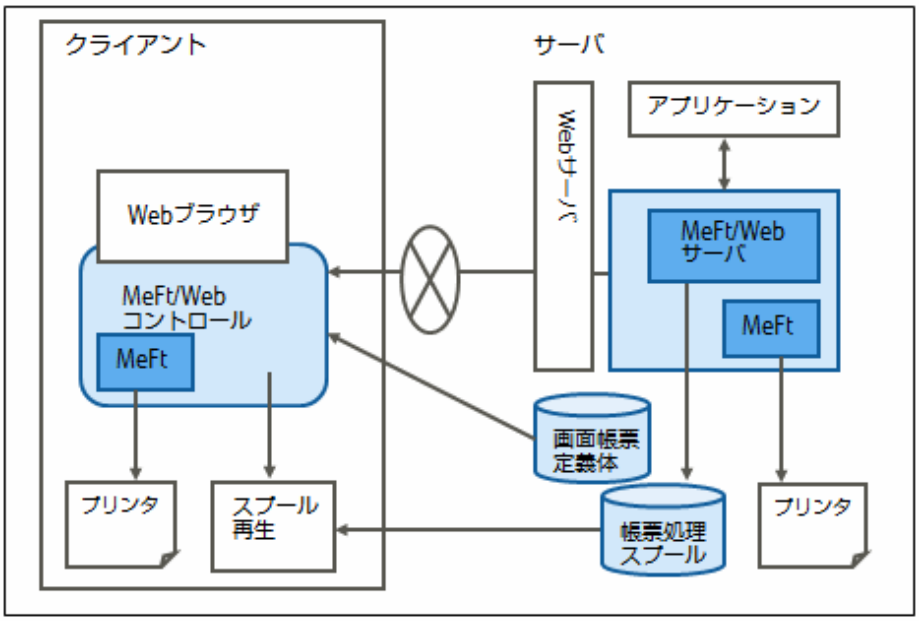
MeFt/Webとは、Webサーバ上で動作するアプリケーションのディスプレイ装置またはプリンタ装置への入出力を、Webブラウザ上で行うことができる通信プログラムです。

MeFt/Webを使用することにより、画面帳票定義体の入出力を行うプログラムをWeb環境で動作させることができるようになります。

MeFt/Webは、サーバ上で動作するWebサーバ連携プログラム（以降、MeFt/Webサーバ）と、クライアント側で動作するActiveXコントロール（以降、MeFt/Webコントロール）から構成されています。

MeFt/Webサーバは、Webサーバを介して、アプリケーションからMeFtに要求された入出力要求を、クライアント側のMeFt/Webコントロールに渡すなどの処理を行っています。

MeFt/Webコントロールは、MeFt/Webサーバからの入出力要求をWebブラウザやプリンタ装置に対して行います。また、MeFt/Webコントロールは、MeFt/Webサーバとの通信処理やMeFt機能がActiveXコントロール化されたものであり、必要時にサーバ上からダウンロードされます。MeFt/Webコントロールからサーバ上のアプリケーションを実行し、画面帳票の入出力をWebブラウザで行うことを「リモート実行」といいます。



MeFt/Webには、次のような機能があります。

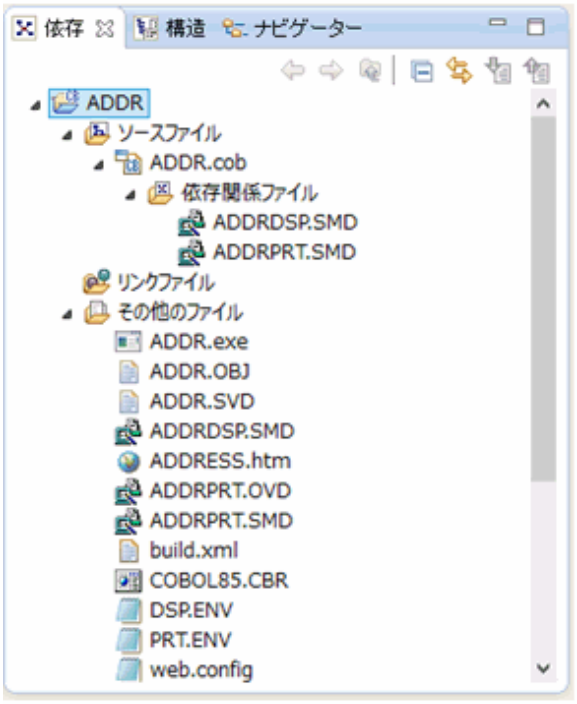
機能名		説明
画面機能	画面処理	リモート実行した利用者プログラムからの画面入出力をWebブラウザ上で行います。
	ハイパーリンク	項目にURLを設定することができます。
印刷機能	プレビューおよびクライアント印刷機能	印刷イメージをWebブラウザ上に表示したり、クライアントに接続されているプリンタ装置に印刷します。
	サーバ印刷機能	サーバに接続されているプリンタ装置に印刷します。

機能名		説明
	スプール機能およびスプール再生機能	利用者プログラムからの印刷要求をサーバ上にスプールしたり、その帳票結果をWebブラウザ上で再生(プレビュー)します。
サービスマネージャー機能		サーバ上の利用者プログラムを起動したり、起動しているプログラムの一覧表示などを行います。

4.1.2 構築するアプリケーションについて

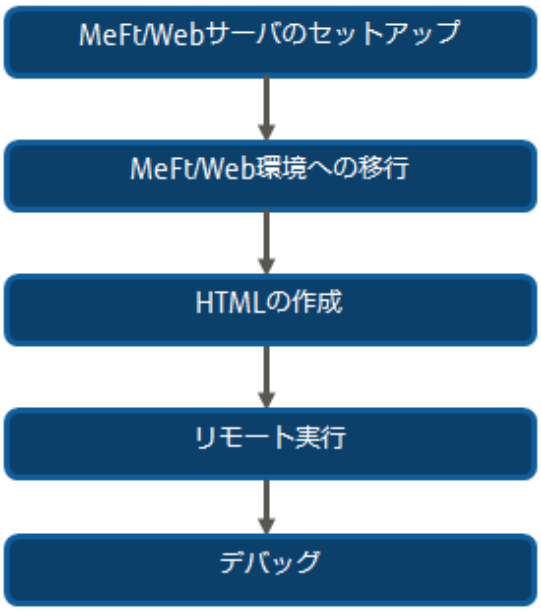
“第3章 画面帳票アプリケーションの開発”で作成した画面帳票アプリケーションをWebサーバに移行して、MeFt/Webアプリケーションを構築します。

なお、Webサーバのホスト名は「SampleSvr」とし、アプリケーションの実行に必要な資産は、サーバ上の「ADDR」プロジェクトに登録します。
[参照]“4.3.2.1 アプリケーション資産の配置”



4.1.3 構築作業の流れ

MeFt/Webアプリケーションの構築の流れを次に示します。

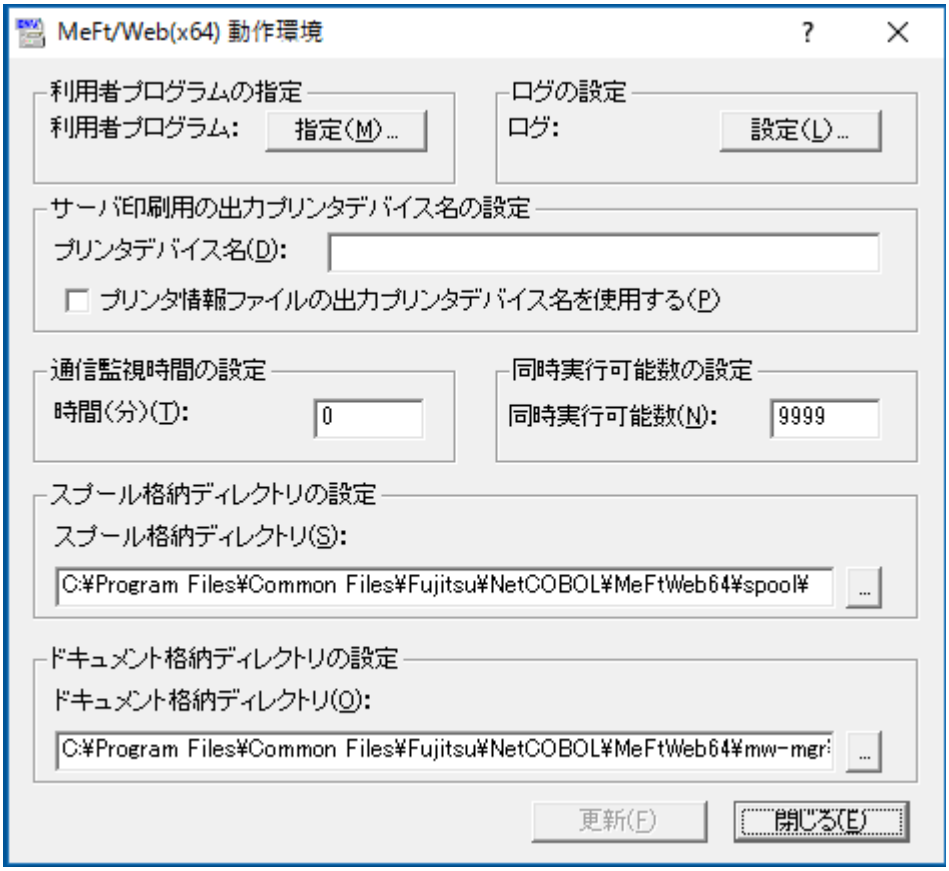


4.2 MeFt/Webサーバのセットアップ

MeFt/Webサーバのセットアップ方法を説明します。セットアップとして、MeFt/Web動作環境の設定と権限の設定を行います。

4.2.1 MeFt/Web動作環境の設定

[スタート] > お使いのNetCOBOL製品名 > [MeFt/Web 動作環境]を選択して、MeFt/Web動作環境の設定画面を表示します。



動作環境設定の各項目の概要を次に示します。詳細は、“MeFt/Webユーザーズガイド”の“MeFt/Webの動作環境を設定する”をご参照ください。

項目名	説明
利用者プログラム	起動を許可する利用者プログラムおよび参照を許可するユーザ資源を指定します。
ログ	MeFt/Webサーバで採取するトレースログ環境を指定します。
サーバ印刷用の出力プリンタデバイス名	サーバ印刷で使用するプリンタデバイス名を指定します。
プリンタ情報ファイルの出力プリンタデバイス名を使用する	MeFt/Web動作環境とプリンタ情報ファイルの両方に出力プリンタデバイス名が指定された場合、どちらの指定のプリンタに印刷するか指定します。
通信監視時間	Webブラウザからの無応答を監視する時間を指定します。
同時実行可能数	アプリケーションの同時実行可能数を指定します。
スプール格納ディレクトリ	スプール機能を実行した際に印刷データを格納するフォルダーを指定します。
ドキュメント格納ディレクトリ	MeFt/Webドキュメントを格納するフォルダーを指定します。

動作環境設定の各項目のうち、サーバ印刷の出力プリンタデバイス名、通信監視時間および同時実行可能数を説明します。

4.2.1.1 サーバ印刷の出力プリンタデバイス名

サーバ印刷で使用するプリンタデバイス名は、MeFt/Webの動作環境とサーバ印刷用のプリンタ情報ファイル2ヵ所で設定することができます。両方で指定があった場合、「プリンタ情報ファイルの出力プリンタデバイス名」の指定に従います。

なお、出力プリンタデバイス名は、[デバイスとプリンタの表示]を選択して表示される一覧を参照し、プリンタ名を""で囲んで指定します。ただし、ローカルプリンターとネットワークプリンターとで指定方法が違います。

参考

[デバイスとプリンタの表示]を表示させるには、[コントロールパネル]を開き、[表示方法]が「カテゴリ」になっていることを確認して、「デバイスとプリンタの表示」をクリックします。

ローカルプリンターの場合

一覧から得られる名前で指定します。

例えば、ローカルプリンター「FUJITSU VSP4530B」の場合、この名前を指定します。

ネットワークプリンターの場合

「¥¥サーバ名¥プリンタ名」という形で指定します。

例えば、ネットワークプリンター「FUJITSU VSP4530B（COBPRTSV)」の場合、COBPRTSVというサーバに接続されたFUJITSU VSP4530Bというネットワークプリンターなので、「"¥¥COBPRTSV¥FUJITSU VSP4530B"」と指定します。

4.2.1.2 通信監視時間

ネットワークの異常やクライアントマシンの強制終了などによってWebブラウザからの応答をサーバのアプリケーションに返すことができなくなった場合、アプリケーションは応答待ちのままになってしまいます。この問題に対処するため、通信監視時間を設定します。

MeFt/Webサーバでは、一定の時間（通信監視時間）を超えてアプリケーションに応答することができなかった場合、MeFtの通知コードとして「N7」を通知します。COBOLアプリケーションでは、表示ファイルの各命令後にFILE STATUS句に指定した4桁のデータ領域が「90N7」であるか判定することで、通信監視時間を超えて応答がなかったことを知ることができます。

4.2.1.3 同時実行可能数

MeFt/Webでは、リモート実行されるアプリケーションの同時実行可能数を指定することができます。サーバマシンの性能などを考慮し、同時に実行するアプリケーションの数を設定します。

ここで指定された同時実行可能数を越えてプログラムをリモート実行しようとすると、Webブラウザに「P2006プログラムを処理できませんでした。同時実行可能数を超えました。」というMeFt/Webコントロールのエラーが表示されます。

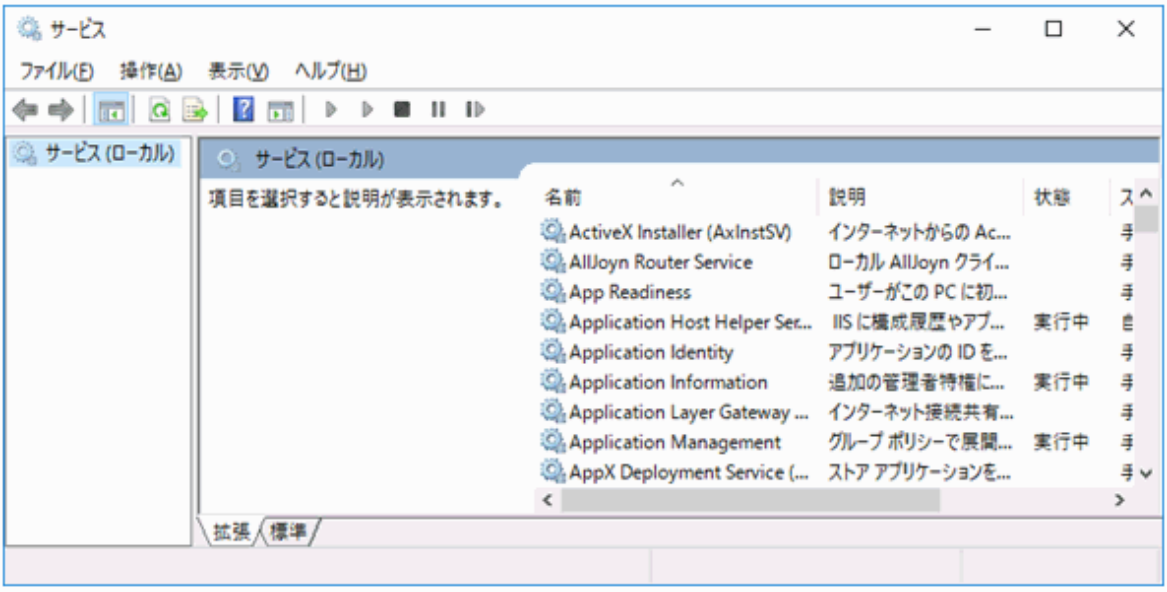
なお、同時実行可能数のチェックの対象となるのは、単一のアプリケーションではなく、リモート実行される全アプリケーションです。また、同時に実行するクライアントの台数ではなく、実行されるアプリケーションの数が対象となります。

4.2.2 権限の設定

MeFt/Webで起動されるCOBOLプログラムの権限を設定します。COBOLプログラムが扱う資源（フォルダー、プリンターなど）に応じて、アカウントを設定します。

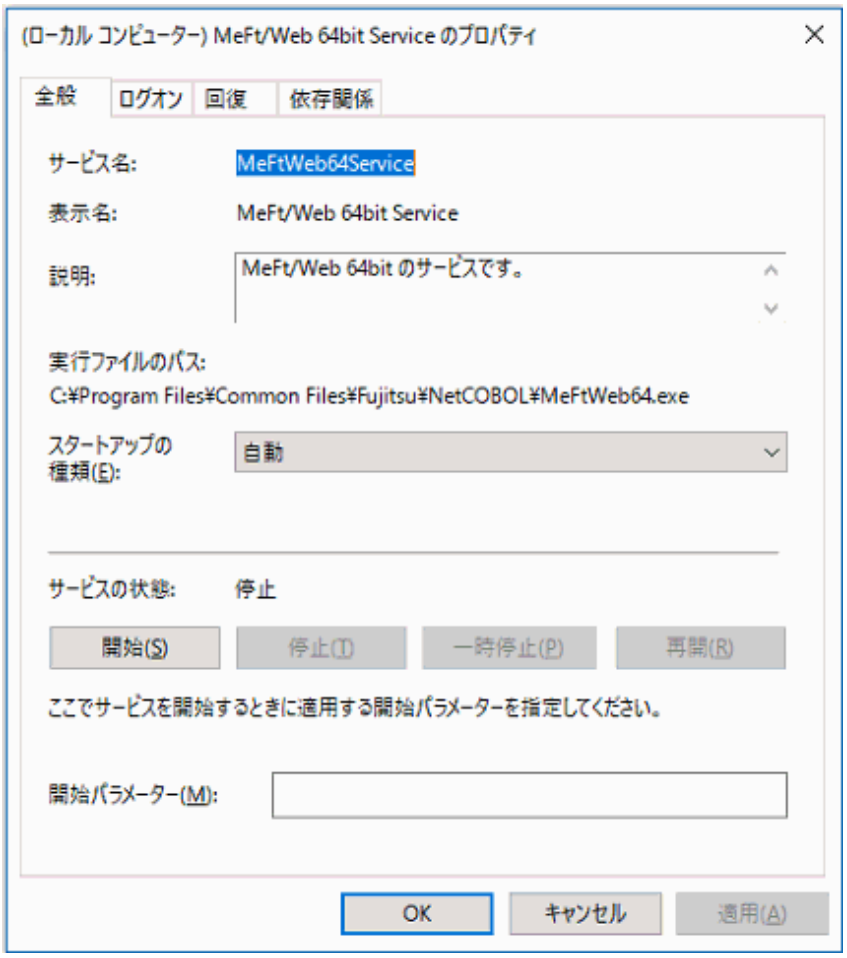
プログラムの権限を設定する方法を説明します。

1. MeFt/Webがインストールされたマシンで、[コントロールパネル]を開き、[表示方法]を「小さいアイコン」にして、[管理ツール]を選択します。
2. [サービス]をダブルクリックします。
3. [サービス]画面が表示されます。

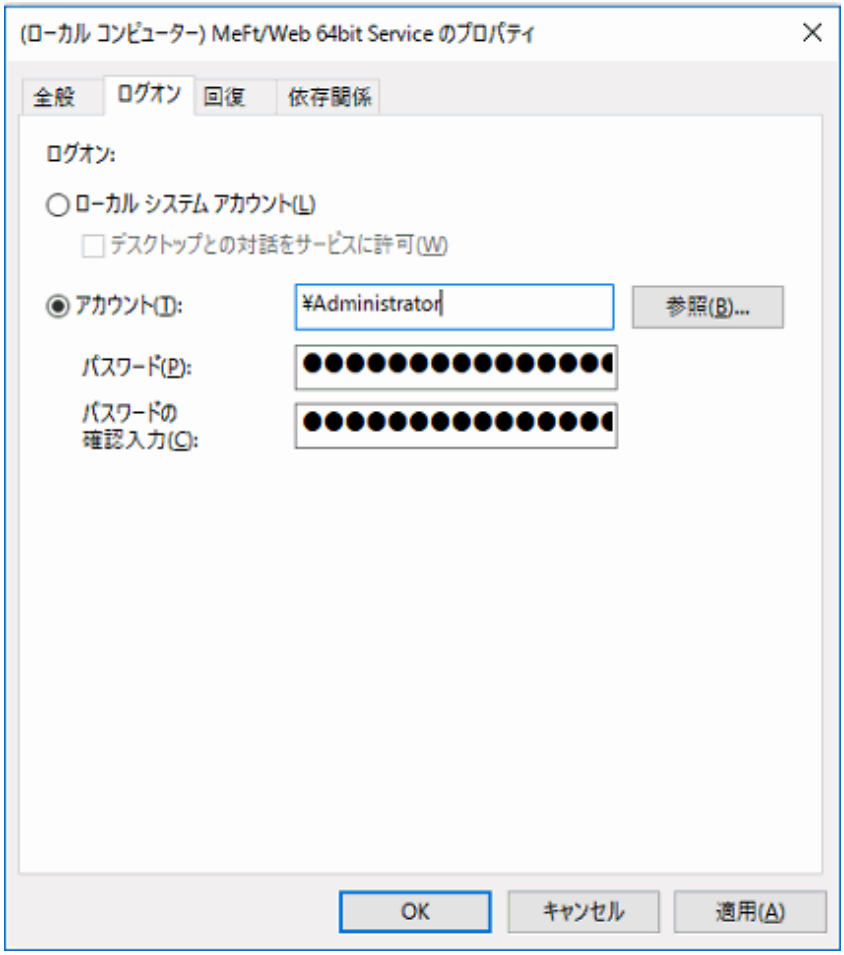


4. サービスの画面から「MeFt/Web 64bit Service」を選択します。

   [操作]メニューから「プロパティ」を選択して、MeFt/Web 64bit Serviceのプロパティ画面を表示します。



5. [ログオン]タブで「アカウント」を選択し、使用するユーザアカウントとそのパスワードを設定します。次に示す例では、ユーザアカウントとして「Administrator」を設定しています。



6. [OK]ボタンを押して、MeFt/Web 64git Serviceのプロパティ画面を閉じます。

参考

MeFt/Web 64bit Serviceは、インストール時はシステムアカウントになっていますが、システムアカウントを指定するとプロセスを強制終了できないなどの不都合が発生するため、システムアカウント以外にします。

また、システムアカウントのままだと、イベントビューアの「アプリケーション ログ」に次に示すイベントが通知されます。

項目	内容
ソース	MeFt/Web Service
イベントID	122
説明	ユーザレジストリのロードに失敗しました。

4.3 MeFt/Web環境への移行

スタンドアロンで動作していた画面帳票アプリケーションを、MeFt/Web環境へ移行する方法を説明します。

4.3.1 MeFt/Web移行時のアプリケーションの対応

スタンドアロンの画面帳票アプリケーションをMeFt/Web環境へ移行するにあたり、対応が必要となる点を説明します。

4.3.1.1 表示ファイル以外の画面

MeFt/Webでリモート実行したプログラムで表示される画面のうち、表示ファイルを使用した画面以外はサーバ上で処理されます。サーバ上で処理される画面には、次のようなものがあります。

- エラーメッセージ
- コンソール画面
- 診断機能メッセージ
- ウィンドウクローズメッセージ

しかし、通常、Webサーバを介して起動されたプログラムの画面は表示されないため、その画面への応答ができずに入力待ち状態になってしまいます。その結果、クライアントのWebブラウザが無応答となってしまいます。

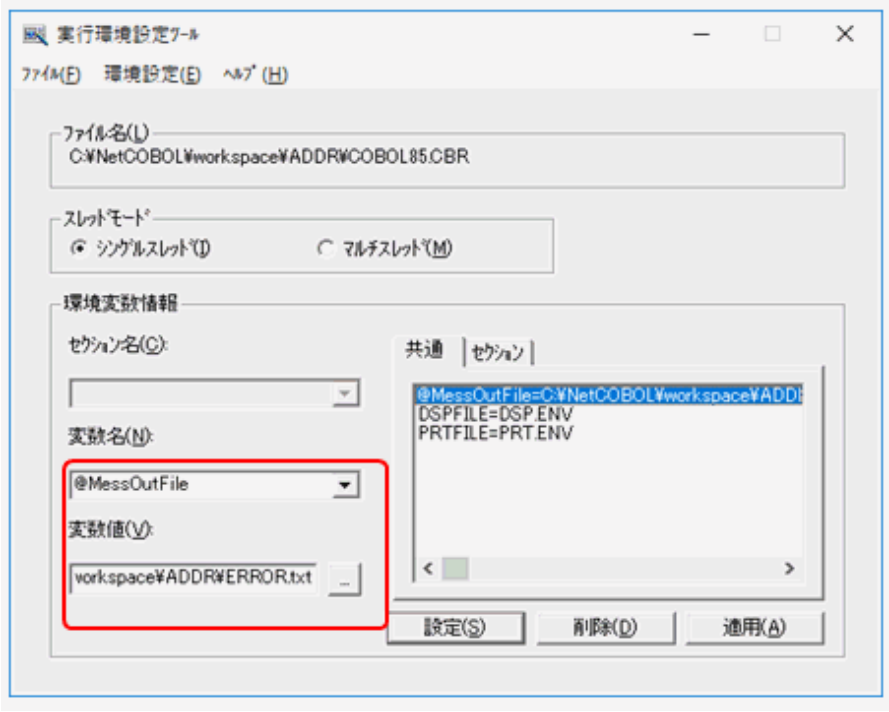
この問題への対応を画面ごとに説明します。

エラーメッセージ

COBOLプログラムの実行時にエラーが発生すると、エラーのメッセージボックスが表示され、[OK]ボタンが押されるのを待つ状態になります。しかし、MeFt/Web環境ではメッセージボックスが表示されても応答できないため、必ずエラーメッセージをファイルに出力する指定をします。

エラーメッセージをファイルに出力するには、実行環境情報で、COBOLが用意する環境変数情報「@MessOutFile」に出力するファイル名を割り当てます。なお、実行環境設定ツールでは、COBOLが用意する環境変数情報を[変数名]のリストから選択することができます。

次の例では、メッセージを出力するファイルとして「C:¥NetCOBOL¥workspace¥ADDR¥ERROR.TXT」を割り当てています。



コンソール画面

コンソール画面に対するACCEPT文およびDISPLAY文を使用したデータの入出力はできません。コンソール画面への出力は、ファイルに出力するよう変更します。コンソール画面への出力をファイルにするには、翻訳オプション「SSOUT」で任意の環境変数情報名を指定し、実行環境情報でその環境変数情報名とファイル名を対応づけます

以下に、翻訳オプションと実行環境情報の設定例を示します。

```
＜翻訳オプションの指定内容＞
SSOUT(OUTFILE)
＜実行環境情報の指定内容＞
```

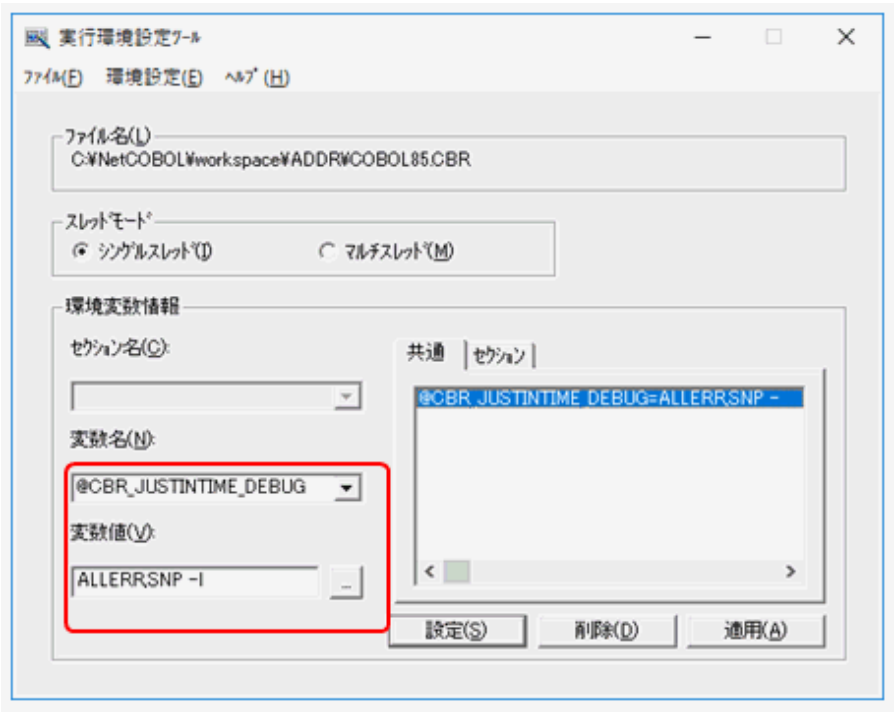
```
OUTFILE=C:¥NetCOBOL¥workspace¥ADDR¥SSOUT.DAT
```

診断機能メッセージ

COBOLプログラムの実行時にアプリケーションエラーなどが発生すると、COBOLの診断機能によりエラーの発生箇所などを記載した診断レポートファイルが出力され、そのことがメッセージボックスに表示されます。MeFt/Web環境では、そのメッセージを表示しないようにするか、診断機能を起動しないようにする指定を必ず行います。

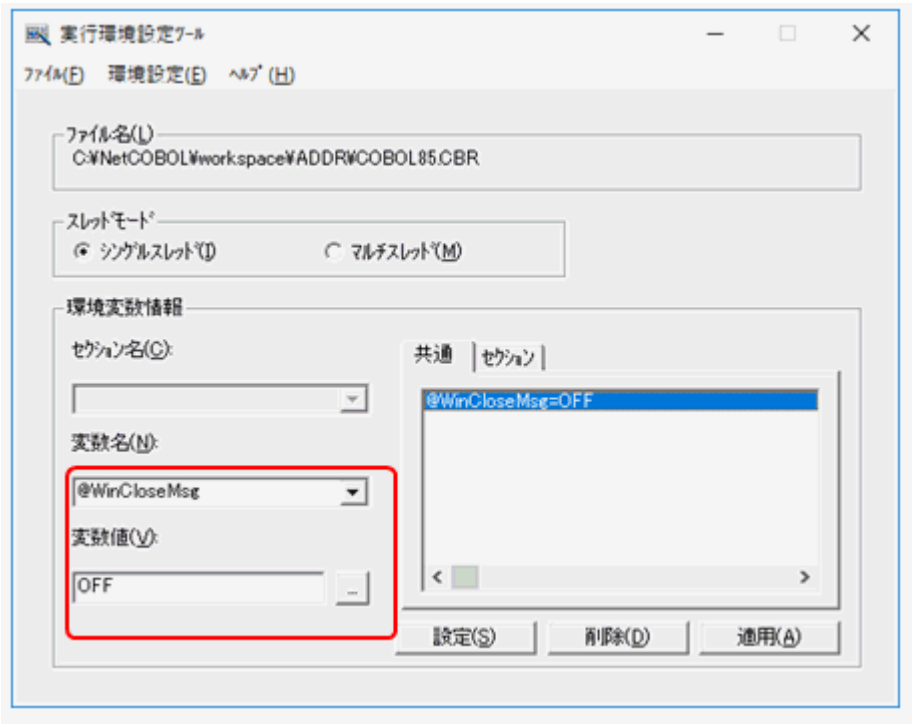
この指定は、実行環境情報で実行環境変数「@CBR_JUSTINTIME_DEBUG」にて行います。診断機能のメッセージを表示させないようにするには、「@CBR_JUSTINTIME_DEBUG」の変数値を「ALLERR,SNAP -l」とします。また、診断機能を起動しないようにするには、「@CBR_JUSTINTIME_DEBUG」の変数値を「NO」とします。

次の例では、診断機能のメッセージを表示させないように「ALLERR,SNAP -l」を指定しています。



ウィンドウクローズメッセージ

ウィンドウを閉じる時の確認メッセージであるウィンドウクローズメッセージを非表示にするため、実行環境情報の実行環境変数「@WinCloseMsg=OFF」を指定する必要があります。



4.3.1.2 プロセス型プログラムとスレッド型プログラム

MeFt/Webでは、次の2種類のCOBOLプログラムを起動できます。

プログラムの種類	説明
プロセス型プログラム	実行可能なモジュール形式(EXE)のプログラムです。
スレッド型プログラム	ダイナミックリンクライブラリ形式(DLL)のプログラムです。

プロセス型プログラムとスレッド型プログラムの比較

プロセス型プログラムとスレッド型プログラムの違いを下表に示します。

項目	プロセス型プログラム	スレッド型プログラム
アプリケーションの形式	主プログラム（EXE）	副プログラム（DLL）
実行単位	プロセス	スレッド
起動性能	スレッド型プログラムと比べ低速。	スタートアップのオーバヘッドがないため高速。
サーバの資源消費	大	小
既存資産の活用性	ソース修正および再翻訳・再リンクとも不要。	再翻訳・再リンクが必要。場合によっては若干のソース修正が必要。
アプリ異常終了時の影響範囲	異常が発生したプログラム以外には影響が及ばない。	同じプロセスで動作する他のスレッド型プログラムも異常終了してしまう。

プロセス型プログラムをスレッド型プログラムに移行する場合の修正点

既存のプロセス型プログラムをスレッド型プログラムに移行する際、プログラム修正が必要となる部分を説明します。

- 環境変数操作

  スレッド型プログラムでは、1つのプロセスで複数のスレッドが動作するため、環境変数の内容が変更されると、他のプログラムに影響を及ぼす可能性があります。したがって、スレッド型プログラムでは環境変数操作を行うことはできません。環境変数操作を行っている場合は、環境変数操作は行わないように修正します。

- 引数の受渡し方法

  プロセス型プログラムでは、起動時に指定された引数を受け取るには、コマンド行引数の操作機能を使用します。コマンド行引数の操作機能は、“NetCOBOL ユーザーズガイド”の「コマンド行引数の取出し」をご参照ください。一方、スレッド型プログラムはC呼出し規約に従って呼び出されるため、起動時に指定された引数を受け取るには、手続き部(PROCEDURE DIVISION)の見出しのUSING指定にデータ名を記述します。したがって、引数の受渡しを行っている場合は、受取り方法を変更します。C呼出し規約による呼出しの詳細は、“NetCOBOL ユーザーズガイド”の「CプログラムからCOBOLプログラムを呼び出す方法」をご参照ください。

  参考

  MeFt/Webアプリケーションでは、起動時に指定する引数はMeFt/Webコントロールのargumentプロパティに指定します。argumentプロパティは、“MeFt/Webユーザーズガイド”の「利用者プログラムの指定方法(pathname/argument/environment/funcname)」を参照してください。

- プログラムの終了

  スレッド型プログラムでは、プログラムの終了にSTOP RUNを使用することはできません。STOP RUNを使用している場合は、EXIT PROGRAMを使用するように修正します。

4.3.1.3 MeFt/Web運用時の追加エラーコード

MeFt/Webの運用時には、スタンドアロンでのエラーコードに加えて、次のエラーコードが通知されます。

エラーコードを判定して処理を分けているプログラムをMeFt/Webで運用する場合は、次のコードも考慮した判定を行うようにします。エラーコードの通知は、“3.2 表示ファイルのプログラミング”の「エラー処理」に記載があります。

通知コード	FILE STATUS句(4桁)に通知される内容	エラー内容
N1	90N1	Webサーバが正常に通信を行うことができなかったため、リモート実行処理を続行できなくなりました。または、クライアントマシンかサーバマシンでメモリ不足が発生しました。
N7	90N7	MeFt/Webサーバで通信監視時間のタイムアウトが発生しました。
N8	90N8	MeFt/WebコントロールのQuitメソッドが実行されました。

なお、本章で構築するアプリケーションでは、画面機能および帳票機能の各ファイルFILE STATUS句に指定した4桁のデータ名の領域の値が「0000」であるかを判定して処理を分けています。そのため、MeFt/Webの運用時に追加されるエラーコードに対応するための修正は特に行いません。

4.3.2 アプリケーション資産の配置と環境設定

MeFt/Webでリモート実行するアプリケーションの資産をWebサーバに配置し、資産に関する環境設定を行います。資産に関する環境設定としては、仮想フォルダーの指定と画面帳票資産の格納先の設定があります。

4.3.2.1 アプリケーション資産の配置

“第3章 画面帳票アプリケーションの開発”で作成した画面帳票アプリケーションの資産をWebサーバに配置します。本章で構築するMeFt/Webアプリケーションでは、次に示す資産をWebサーバの「ADDR」プロジェクト(C:¥NetCOBOL¥workspace¥ADDR)に配置します。

資産の種類	ファイル名
実行可能ファイル	ADDR.exe
画面帳票定義体	ADDRDSP.SMD
	ADDRPRT.SMD
オーバレイ定義体	ADDRPRT.OVD
実行用の初期化ファイル	COBOL85.CBR
ウィンドウ情報ファイル	DSP.ENV

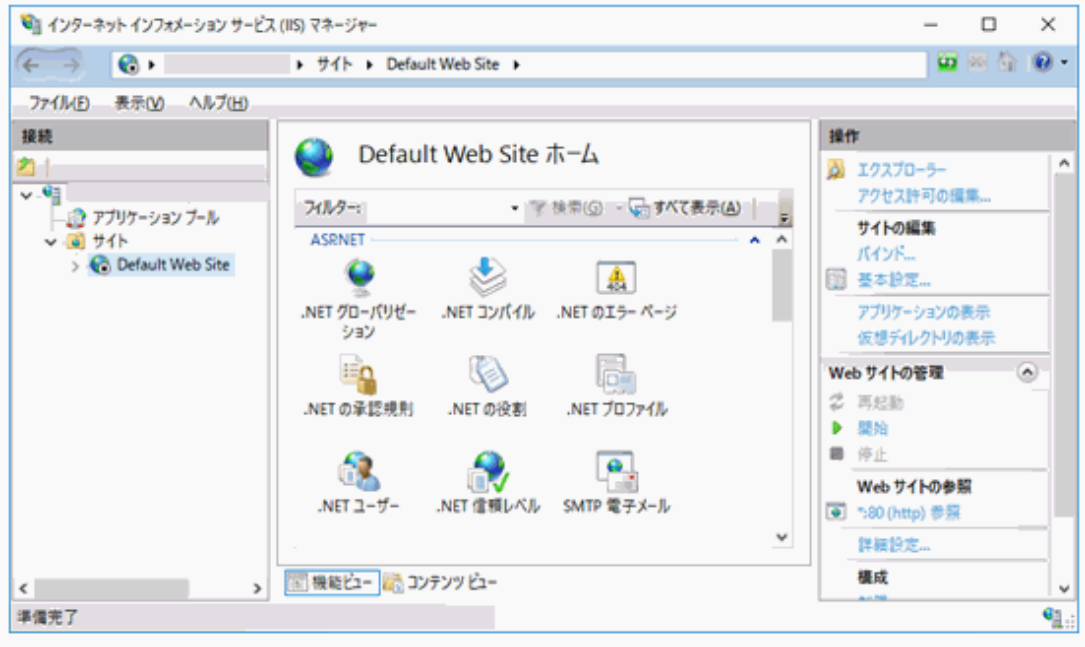
資産の種類	ファイル名
プリンタ情報ファイル	PRT.ENV

4.3.2.2 仮想ディレクトリの設定

配置されたアプリケーションの資産をWebサーバから参照できるようにするため、アプリケーションの資産が格納されているフォルダーを仮想ディレクトリとしてWebサーバに設定します。

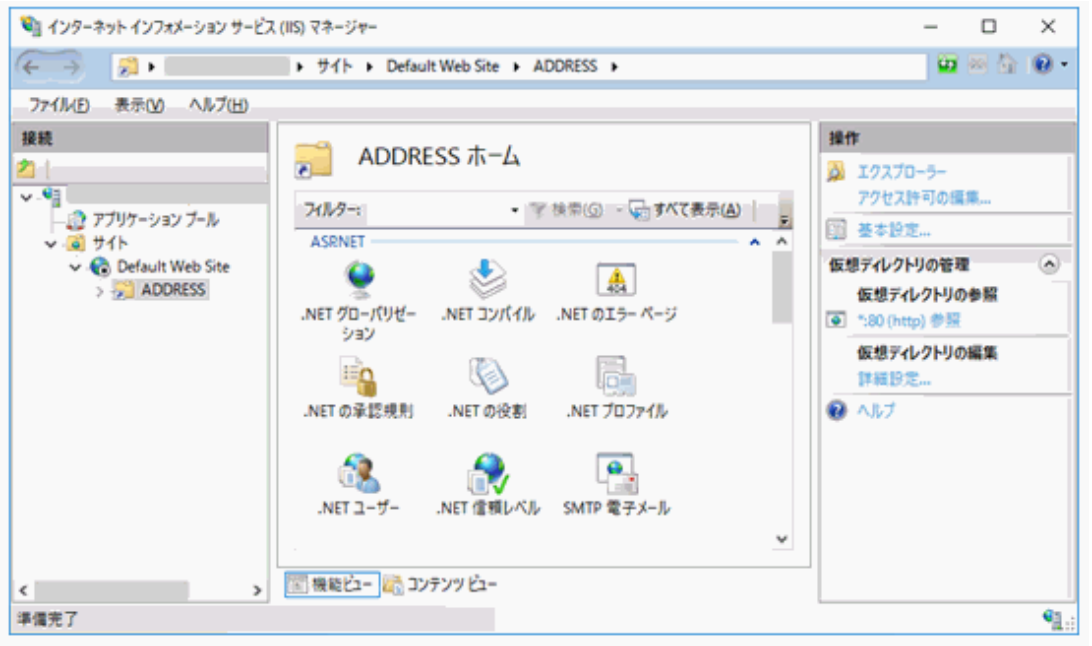
仮想ディレクトリを設定する方法を説明します。下記の説明は、IIS 8.5での設定方法です。

1. [コントロールパネル]を開き、[表示方法]を「小さいアイコン」にして、[管理ツール]を選択します。
2. [インターネット インフォメーション サービス(IIS) マネージャー]をダブルクリックします。
   → IISの画面が表示されます。



3. ホスト名の「サイト」から「Default Web Site」を選択し、右クリックして表示されるコンテキストメニューから[仮想ディレクトリの追加]を選択します。
   → [仮想ディレクトリの追加ウィザード]が表示されます。
4. 次に示す内容を設定して、[OK]ボタンをクリックします。

設定箇所	設定内容
エイリアス	ADDRESS
物理パス	C:¥NetCOBOL¥workspace¥ADDR



参考

Windows Server 2008 R2のIISを利用する場合には、以下を参照してください。

NetCOBOLの技術情報のトラブルシューティング

http://software.fujitsu.com/jp/cobol/technical/trouble_sum.html

「エラー」→「MeFt/Web」→

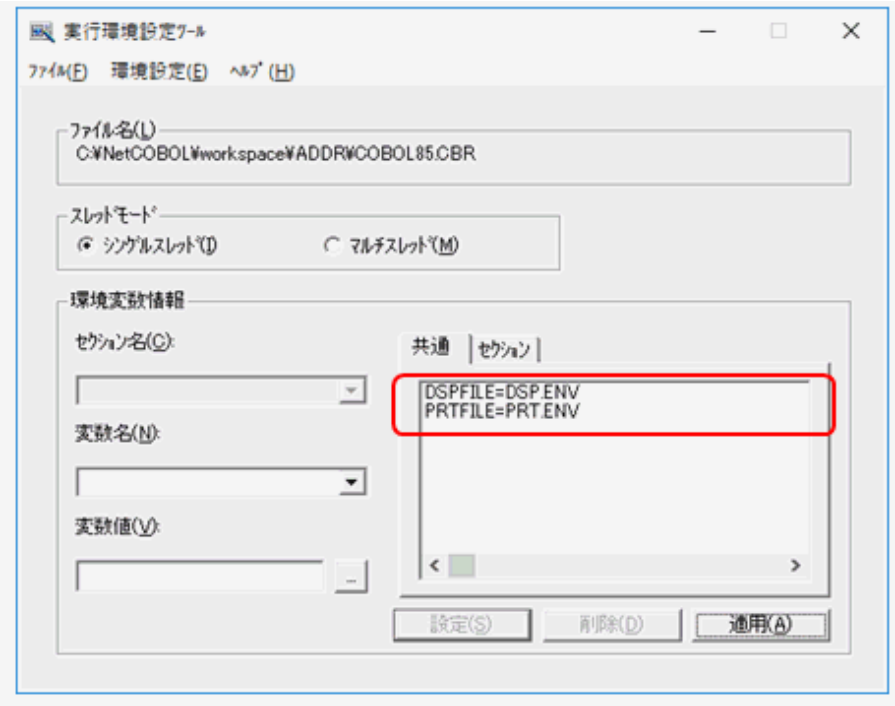
- 「Q.Windows Server 2008でIIS環境設定コマンドを実行すると、「IISの環境設定に失敗しました。詳細コード:500」のエラーとなります。」

4.3.2.3 画面帳票資産の格納先の設定

MeFt/Webアプリケーションが使用する、画面帳票定義体やウィンドウ情報ファイルといったMeFt資産の格納先を指定します。MeFt/Webアプリケーションでは、資産の格納先の指定方法として、URL指定とサーバのローカルパス指定がありますが、ここではURLで指定します。

ウィンドウ情報ファイルおよびプリンタ情報ファイルの格納先の指定

“第3章 画面帳票アプリケーションの開発”で作成したアプリケーションでは、COBOL初期化ファイル(COBOL85.CBR)で、ファイル識別名に対応するウィンドウ情報ファイルおよびプリンタ情報ファイルをフルパスで指定しました。しかし、MeFt/Web環境で運用する場合、ウィンドウ情報ファイルおよびプリンタ情報ファイルの指定は、ファイル名のみを推奨します。



ウィンドウ情報ファイルの格納フォルダーは、アプリケーション実行時の環境変数MEFTWEBDIRに指定します。

一方、プリンタ情報ファイルは、サーバに接続されているプリンタとクライアントに接続されているプリンタが異なるため、サーバ印刷用のプリンタ情報ファイルとクライアント印刷用のプリンタ情報ファイルを用意しますが、サーバ印刷用のプリンタ情報ファイルとクライアント印刷用のプリンタ情報ファイルとでは、格納フォルダーの指定が異なります。

サーバ印刷用のプリンタ情報ファイルは、アプリケーション実行時の環境変数MEFTDIRに指定します。それに対し、クライアント印刷用のプリンタ情報ファイルの格納フォルダーは、アプリケーション実行時の環境変数MEFTWEBDIRに指定します。

なお、サーバ印刷用のプリンタ情報ファイルのファイル名とクライアント印刷用のプリンタ情報ファイルのファイル名は同一である必要があります。

参考

アプリケーション実行時の環境変数を設定するには、以下のような方法があります。

- 実行用の初期化ファイルで設定する
- MeFt/Webをリモート実行するHTMLで、MeFt/Webコントロールのenvironmentプロパティで設定する
- システム環境変数で設定する
- SETコマンドで設定する

MeFt/Webコントロールのenvironmentプロパティでの設定方法は、“4.4 HTMLの作成”および“MeFt/Webユーザーズガイド”の“利用者プログラムの指定方法(pathname/argument/environment/funcname)”を参照してください。

システム環境変数およびSETコマンドで設定する方法は、“NetCOBOL ユーザーズガイド”の“実行環境情報の設定方法”を参照してください。

画面帳票定義体の格納先の指定

“第3章 画面帳票アプリケーションの開発”で作成したアプリケーションにおいては、画面帳票定義体の格納フォルダーは、ウィンドウ情報ファイルおよびプリンタ情報ファイルのMEDDIRキーワードにてローカルパスで指定しましたが、ここでは次に示すようにURLで指定します。

```
MEDDIR http://SampleSvr/ADDRESS
```

オーバレイ定義体の格納先の指定

“第3章 画面帳票アプリケーションの開発”で作成したアプリケーションにおいては、オーバレイ定義体の格納フォルダーは、プリンタ情報ファイルのOVLPDIRキーワードにてローカルパスで指定しましたが、ここでは次に示すようにURLで指定します。

```
OVLPDIR http://SampleSvr/ADDRESS
```

画像ファイルの指定

画面帳票定義体には、画像ファイルを出力するための項目として「組み込みメディア項目」があり、プログラムの実行時にビットマップファイルなどを画面帳票定義体に出力することができます。組み込みメディア項目に使用するビットマップファイルの格納フォルダーは、ウィンドウ情報ファイルおよびプリンタ情報ファイルのMEDIADIRキーワードで指定します。

また、画面定義体の背景にビットマップファイルを表示することもできます。画面定義体の背景に表示する画像ファイルを「背景メディア」といいます。背景メディアに使用するビットマップファイルの格納フォルダーは、ウィンドウ情報ファイルのBACKMEDIAキーワードで指定します。

スタンドアロンで組み込みメディア項目および背景メディアを使用していたアプリケーションをMeFt/Web環境へ移行する場合は、MEDIADIRキーワードおよびBACKMEDIAキーワードを使用して、ビットマップファイルの格納フォルダーをURLで指定します。

“第3章 画面帳票アプリケーションの開発”で作成したアプリケーションでは、組み込みメディア項目および背景メディアの定義を行わなかったため、ここでは、MEDIADIRキーワードおよびBACKMEDIAキーワードの指定は行いません。

参考

URLで指定する資産は、上記で説明したもののみです。初期化ファイルで指定するデータファイルなどは、フルパスで指定します。

4.3.2.4 利用者プログラムの指定

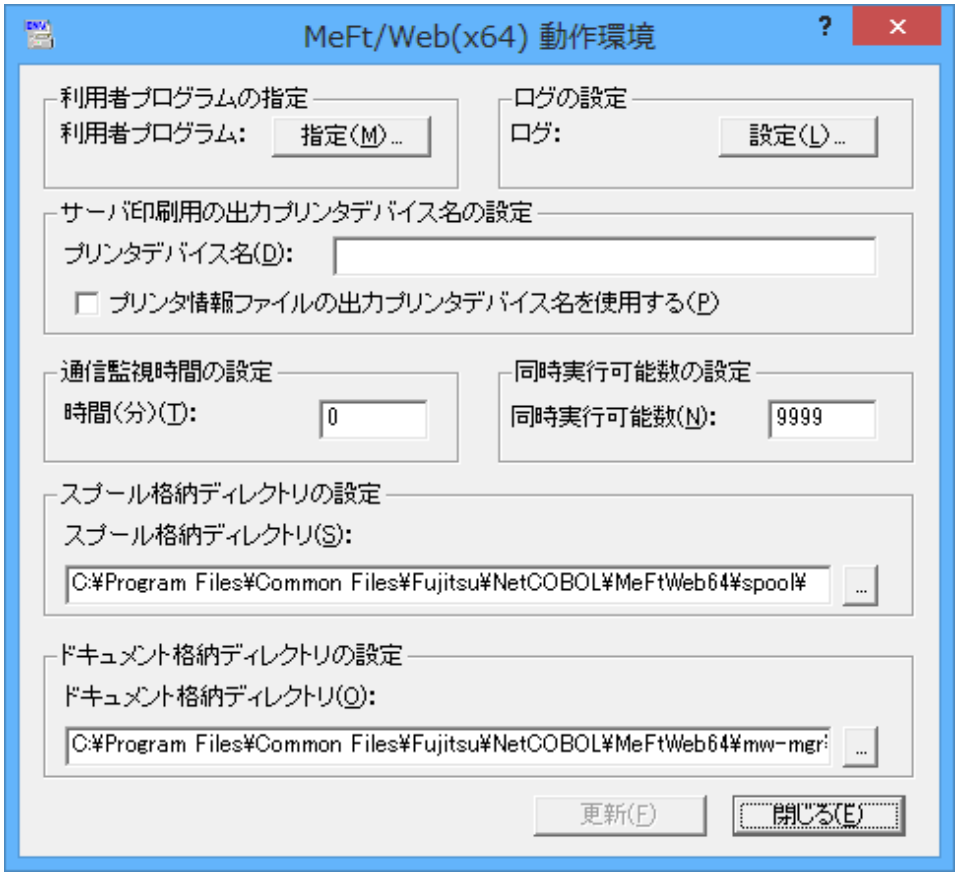
「利用者プログラム指定ファイル」にリモート実行で起動する利用者プログラムを限定し、実行できるプログラムを制限します。

「利用者プログラム指定ファイル」に記述されていない利用者プログラムが指定された場合、「P2016プログラムの起動に失敗しました」のエラーメッセージがクライアントに表示され、処理が停止されます。

詳細は、“MeFt/Webユーザーズガイド”の“利用者プログラムの指定”をご参照ください。

利用者プログラム指定ファイルの設定

1. [スタート] > お使いのNetCOBOL製品名 > [MeFt/Web 動作環境]を選択します。
   → [MeFt/Web動作環境]の設定画面が表示されます。



2. 利用者プログラムの[指定]ボタンを押します。

   → メモ帳で利用者プログラム指定ファイルがオープンされます。

3. [programs] セクションに、リモート実行機能で起動する利用者プログラムを指定します。

```
***  MeFt/Web 利用者プログラム指定ファイル  ***
[programs]
* 以下にMeFt/Webサーバで実行を許可する利用者プログラムの
* ファイル名またはフォルダー名を記述してください。
C:¥NetCOBOL¥workspace¥ADDR¥ADDR.EXE
```

4. [ファイル]メニューから「上書き保存」を選択します。保存後メモ帳を終了します。

4.4 HTMLの作成

Webサーバ上のプログラムをリモート実行するには、HTMLを作成する必要があります。

HTMLでは、次に示すような内容を記述します。

- MeFt/Webコントロールの指定
- アプリケーションの起動方法
- Webサーバの指定
- リモート実行するアプリケーションの指定
- リモート実行時の動作に関する指定

- ページ移動時の動作に関する指定

ここでは、MeFt/Webのサンプルプログラムで提供されているHTMLをコピーし、必要な部分を修正します。

サンプルプログラムのHTMLは次に示す場所にある「denpyou1.htm」です。

```
C:¥Program Files¥NetCOBOL¥Samples¥MeFtWeb¥Sample.web¥
```

修正したHTMLを以下に示します。下線付き赤字の部分が修正した箇所です。

なお、このHTMLに「address.htm」というファイル名をつけ、「C:¥NetCOBOL¥workspace¥ADDR」に格納することとします。

```
<HTML>
<HEAD>
<TITLE>住所録</TITLE>
</HEAD>
<BODY>
<INPUT TYPE=BUTTON VALUE="GO!" NAME="GO"><BR>
<OBJECT
ID="MeFtWeb1"
CLASSID="CLSID:61F12C43-5357-11D0-9EA0-00000E4A0F56"
WIDTH="670" HEIGHT="470"
CODEBASE="http://SampleSvr/MeFtWeb64/meftweb.cab#version=12,0,0,2">
</OBJECT>
<SCRIPT LANGUAGE="VBScript">
Sub GO_onClick()
MeFtWeb1.hostname = "SampleSvr"
MeFtWeb1.pathname = "C:¥ADDRESS¥ADDR.EXE"
MeFtWeb1.environment = "MEFTWEBDIR=http://SampleSvr/ADDRESS"
MeFtWeb1.gatewaypathname = "MeFtWeb64"
MeFtWeb1.message = TRUE
MeFtWeb1.usedcgi = FALSE
MeFtWeb1.ssl = FALSE
MeFtWeb1.displaywindow = 0
MeFtWeb1.hyperlink = 0
MeFtWeb1.printmode = 0
MeFtWeb1.previewwindow = 0
MeFtWeb1.previewdrawpos = 0
MeFtWeb1.previewdc = 0
MeFtWeb1.previewrate = 0
MeFtWeb1.submit()
end sub
Sub Window_onUnload()
MeFtWeb1.Quit()
end sub
</SCRIPT>
</BODY>
</HTML>
```

このHTMLを元に、記述する内容を説明します。

MeFt/Webコントロールの指定

MeFt/Webの画面を表示するため、MeFt/Webコントロールの指定を行います。

MeFt/Webコントロールの指定は、HTMLのOBJECTタグに記述します。CODEBASEで指定するMeFt/Webコントロールの格納先には、Webサーバのホスト名を記述します。

本章で作成するHTMLでは、以下の記述でMeFt/Webコントロールを指定しています。

```
OBJECT
ID="MeFtWeb1"
CLASSID="CLSID:61F12C43-5357-11D0-9EA0-00000E4A0F56"
WIDTH="670" HEIGHT="470"
```

```
CODEBASE="http://SampleSvr/MeFtWeb64/meftweb.cab#version=12,0,0,2">
</OBJECT>
```

アプリケーションの起動方法

どのような操作によってアプリケーションを起動するのかを指定します。アプリケーションの起動には、MeFt/Webコントロールのsubmitメソッドを使用します。

本章で作成するHTMLでは、INPUTタグで指定されたボタンが押されたら起動するようにしています。

```
<INPUT TYPE=BUTTON VALUE="GO!" NAME="GO">
   :
<SCRIPT LANGUAGE="VBScript">
Sub GO_onClick()
   :
MeFtWeb1.submit()
end sub
   :
</SCRIPT>
```

参考

HTMLの表示と同時にアプリケーションを起動するには、以下のように記述します。

```
<SCRIPT LANGUAGE="VBScript">
Sub Window_onload()
   :
MeFtWeb1.submit()
end sub
   :
</SCRIPT>
```

Webサーバの指定

アプリケーションが格納されているWebサーバのホスト名を指定します。

Webサーバのホスト名は、MeFt/Webコントロールのhostnameプロパティで指定します。Webサーバ名を省略することはできません。なお、サーバとクライアントが異なるドメインに所属する場合は、hostnameプロパティで指定するホスト名をフルドメイン形式で指定します。

本章で作成したHTMLでは、以下の記述でWebサーバを指定しています。

```
MeFtWeb1.hostname = "SampleSvr"
```

リモート実行するアプリケーションの指定

どのアプリケーションをリモート実行するのかを指定します。

リモート実行するアプリケーション名は、MeFt/Webコントロールのpathnameプロパティにサーバのローカルパスで指定します。アプリケーション名を省略することはできません。

また、アプリケーション実行時の環境変数は、MeFt/Webコントロールのenvironmentプロパティで指定することができます。

本章で作成したHTMLでは、以下の記述でアプリケーション名と環境変数を指定しています。

```
MeFtWeb1.pathname = "C:¥ADDRESS¥ADDR.EXE"
MeFtWeb1.environment = "MEFTWEBDIR=http://SampleSvr/ADDRESS"
```

参考

ここでは、環境変数としてMEFTWEBDIRを指定しています。環境変数MEFTWEBDIRは、“4.3.2 アプリケーション資産の配置と環境設定”を参照してください。

また、サーバ印刷用のプリンタ情報ファイルの格納フォルダーを指定する環境変数MEFTDIRは指定していません。その場合、サーバ印刷時の出力プリンタは、MeFt/Web動作環境の「サーバ印刷用の出力プリンタデバイス名」の設定に従います。

リモート実行時の動作に関する指定

画面の表示形式や印刷先の指定など、リモート実行時の動作に関する指定は、MeFt/Webコントロールの各プロパティで指定します。

本章で作成したHTMLでは、以下の記述でリモート実行時の動作に関する指定を行っています。MeFt/Webコントロールの各プロパティの説明は、“MeFt/Webユーザーズガイド”の“プロパティ”を参照してください。

```
MeFtWeb1.message = TRUE
MeFtWeb1.gatewaypathname = "MeFtWeb64"
MeFtWeb1.usedcgi = FALSE
MeFtWeb1.ssl = FALSE
MeFtWeb1.displaywindow = 0
MeFtWeb1.hyperlink = 0
MeFtWeb1.printmode = 0
MeFtWeb1.previewwindow = 0
MeFtWeb1.previewdrawpos = 0
MeFtWeb1.previewdc = 0
MeFtWeb1.previewrate = 0
```

ページ移動時の動作に関する指定

リモート実行中にページが移動された場合の動作は、ページ移動時に発生するWindow_onUnloadイベントの処理として指定します。リモート実行中にページが移動された場合、そのことをアプリケーションに通知するため、MeFt/WebコントロールのQuitメソッドを記述します。

本章で作成したHTMLでは、以下の記述でページ移動時の動作に関する指定を行っています。詳細は、“4.7 通信が切断されるパターンについて”を参照してください。

```
<SCRIPT LANGUAGE="VBScript">
   :
Sub Window_onUnload()
MeFtWeb1.Quit()
end sub
   :
</SCRIPT>
```

作成したHTMLをWebブラウザで表示すると、次のようになります。

4.5 リモート実行

Webサーバマシン上に配置したCOBOLアプリケーションをWebブラウザからリモート実行する方法を説明します。

4.5.1 HTMLの表示

Webブラウザで、MeFt/Webをリモート実行するHTMLを表示するために、次のようなURLを指定します。

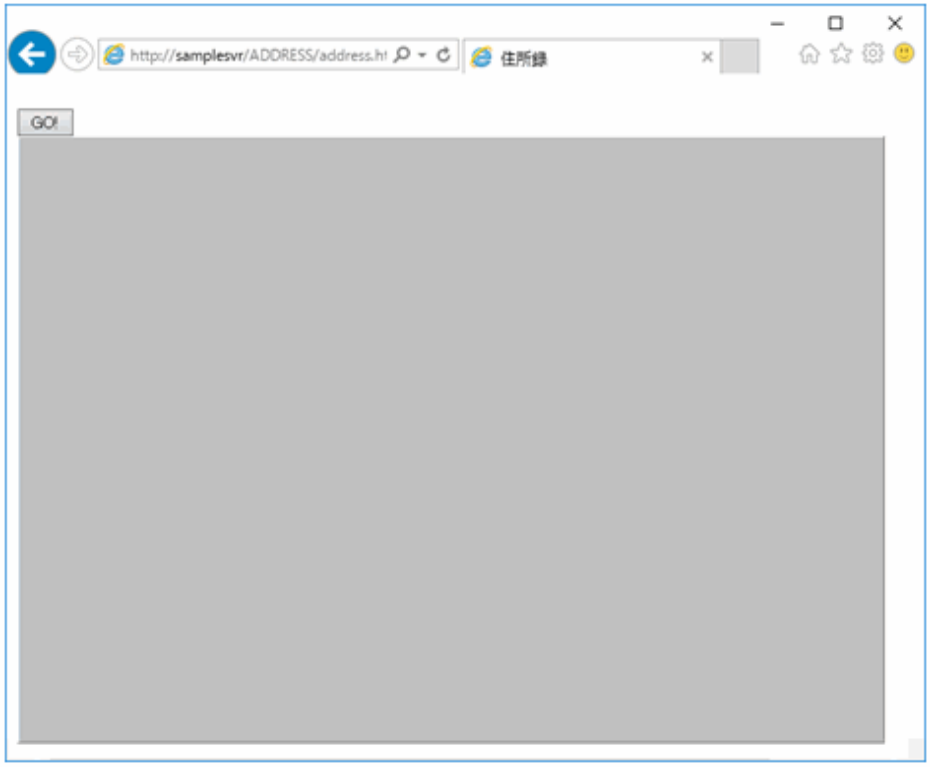
```
http://SampleSvr/ADDRESS/address.htm
```

リモート実行を行うには、クライアントにMeFt/Webコントロールをダウンロードする必要がありますが、Webブラウザでリモート実行するためのHTMLを表示すると、サーバマシンからMeFt/Webコントロールが自動的にダウンロードされます。

MeFt/Webコントロールのダウンロード時、ActiveXコントロールの認証を行うダイアログボックスが表示されますので、[インストールする]を押してダウンロードします。

なお、MeFt/Webコントロールがダウンロードされるのは、MeFt/Webサーバへの初回の接続時のみです。2回目以降は、ダウンロードされているMeFt/Webコントロールが使用されます。

MeFt/Webコントロールがダウンロードされると、次のような画面が表示されます。



4.5.2 プログラムの実行

HTMLの[GO!]ボタンを押すと、プログラムが起動し、次のような画面がWebブラウザに表示されます。

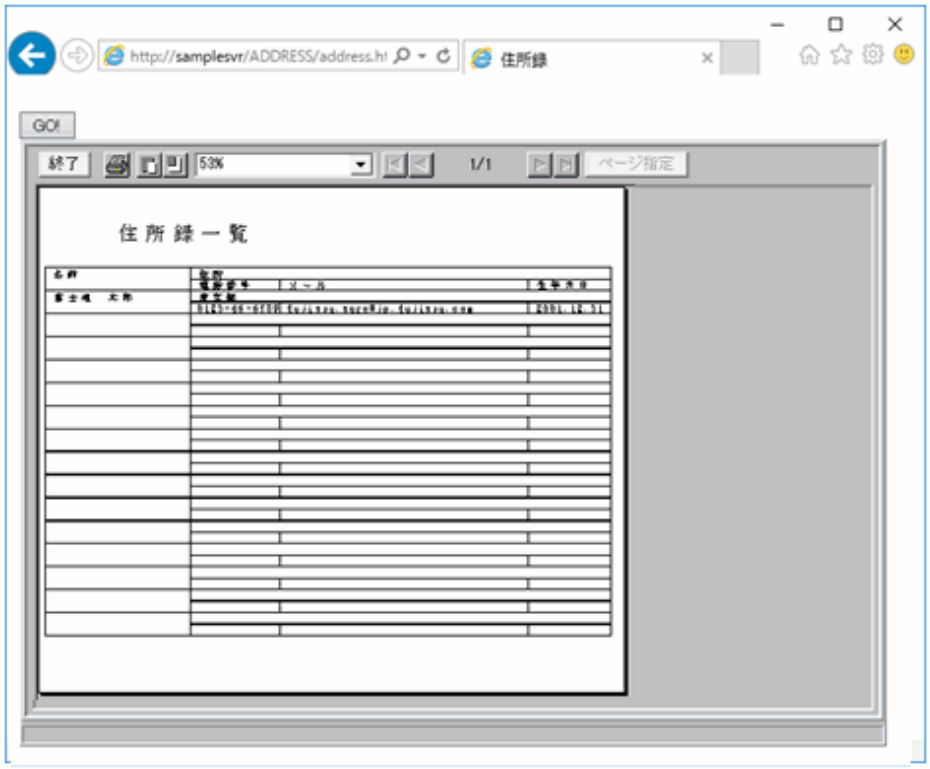
表示された画面に必要な情報を入力し、[F4]キーまたは画面上の[印刷(F4)]というボタンを押すと、サーバのアプリケーションは印刷処理を行います。

MeFt/Webの印刷機能には次の4つがあります。

- プレビュー
- クライアント印刷
- サーバ印刷
- スプール

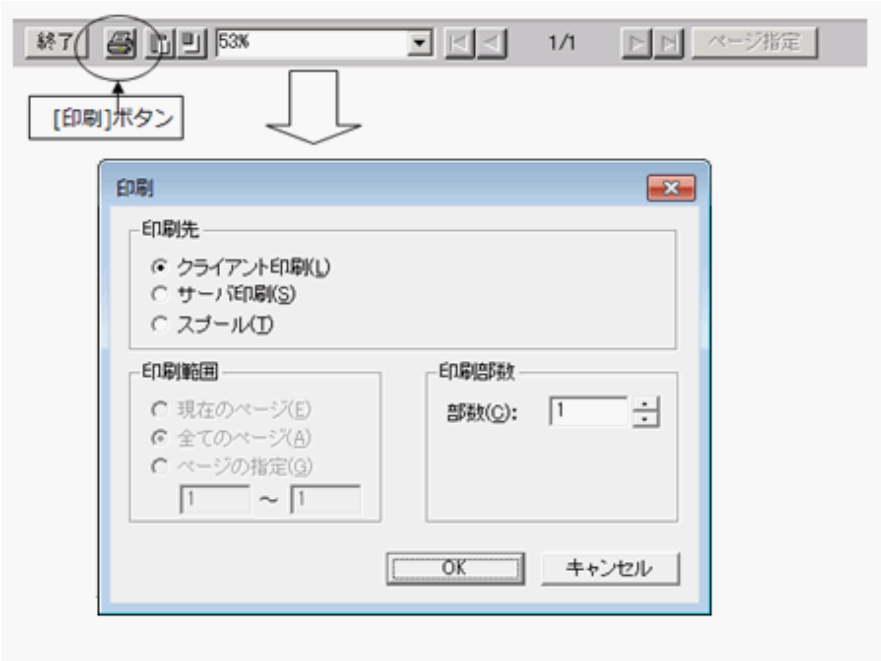
アプリケーションが印刷処理を行った場合、どのような印刷を行うかはリモート実行するためのHTMLで定義しますが、今回使用するHTMLでは、省略値であるプレビューが設定されています。

以下は、帳票印刷のプレビュー表示の例です。



プレビュー表示からの印刷

印刷プレビューの画面で[印刷]ボタンを押すと、印刷先などを設定する画面が表示されます。この画面で印刷先を設定すると、印刷プレビューの画面から指定した印刷先に印刷を行うことができます。



プレビューの終了

プレビュー表示を終了するには、プレビュー画面の[終了]ボタンを押します。

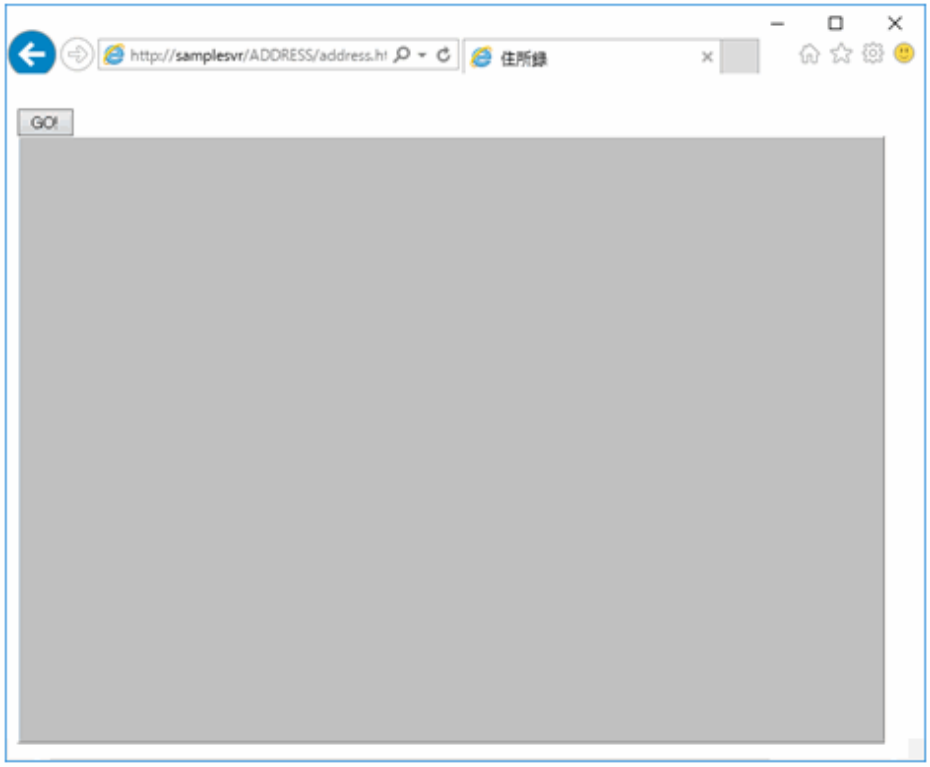
参考

帳票定義体を使用した印刷処理では、プレビューやクライアント印刷などのMeFt/Webの印刷機能が使用できます。

一方、帳票定義体を使用しない印刷処理では、MeFt/Webの印刷機能を使用しないサーバアプリケーションの印刷として処理されます。

プログラムの終了

ここでリモート実行されたアプリケーションでは、入力画面で[F5]キーまたは画面上の[終了(F5)]というボタンを押すと、プログラムが終了します。プログラムが終了すると、Webブラウザは、リモート実行前と同じように表示されます。



4.6 デバッグ

MeFt/Web環境のアプリケーションのデバッグ手順や方法を説明します。

4.6.1 MeFt/Webアプリケーションのデバッグについて

MeFt/Web環境のCOBOLアプリケーションをデバッグするには、NetCOBOL Studioのデバッグ機能を使用します。

4.6.2 デバッグの準備

デバッグの準備として、次の2点の作業を行います。

- デバッグモジュールの作成とデバッグ資産の配置
- 実行環境情報の設定

4.6.2.1 デバッグモジュールの作成とデバッグ資産の配置

デバッグモジュールを作成し、デバッグに必要となる資産をWebサーバマシンに配置します。

デバッグモジュールの作成には、MeFt/Webアプリケーション固有の部分はありません。“第3章 画面帳票アプリケーションの開発”で作成したアプリケーションと同じ方法で作成します。

デバッグモジュールを作成するためのビルド、またはリビルドが終了したあと、次の資産が登録されていることを確認します。

資産の種類	ファイル名
デバッグモジュール	ADDR.EXE
デバッグ情報ファイル	ADDR.SVD
COBOLソースファイル	ADDR.COB

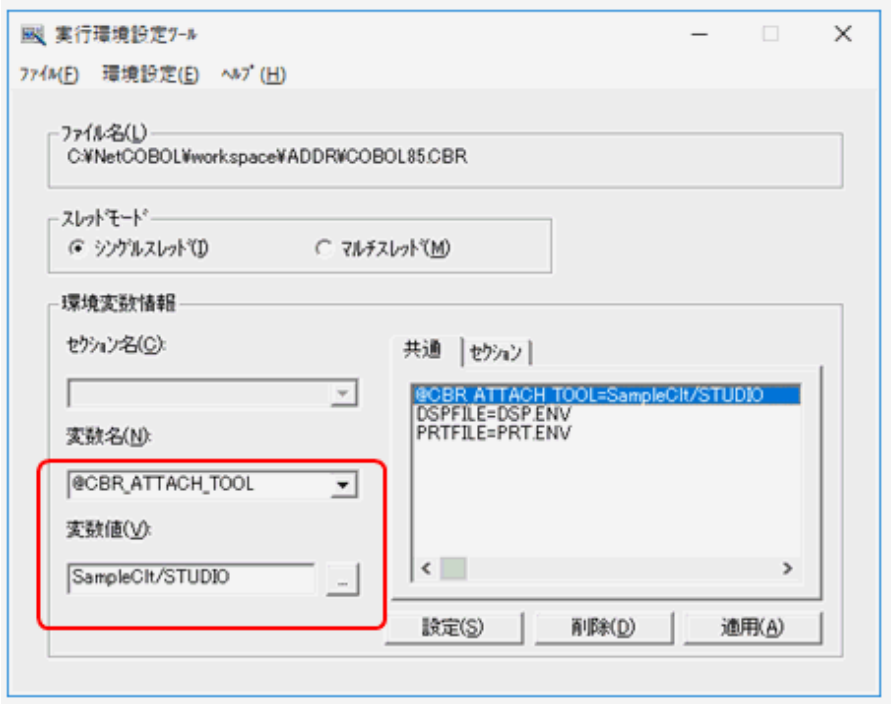
4.6.2.2 実行環境情報の設定

COBOLの実行環境情報で、デバッグしたいプログラムからデバッガを起動する設定を行います。

デバッグしたいプログラムからデバッガを起動する設定は、実行環境変数「@CBR_ATTACH_TOOL」を使用します。

実行環境設定ツールを使用して、変数名「@CBR_ATTACH_TOOL」に変数値を“クライアントのIPアドレスまたはホスト名(*1)/STUDIO”の形式で指定します。「STUDIO」に続けてデバッガ起動時のパラメータ(デバッグ情報ファイルの格納フォルダーなど)を指定できますが、本章で構築するアプリケーションでは必要ありません。

*1:ここではクライアントのホスト名を「SampleClt」とします。



4.6.3 デバッグの開始

プログラムを通常に実行するときと同じように、WebブラウザからWebサーバのCOBOLプログラムをリモート実行します。このとき、Webブラウザはデバッグするプログラムが実行されるサーバマシンでなくても構いません。

COBOLプログラムのデバッグは、NetCOBOL Studioを使用します。デバッグするCOBOLプログラムが実行されるアプリケーションを起動する前に、サーバ上のNetCOBOL Studioの[実行]メニューから、[デバッグ]>[リモートCOBOLアプリケーション]を選択して、デバッガを待ち受け状態にします。

デバッガが待ち受け状態になったことを確認した後、デバッグするCOBOLプログラムが実行されるアプリケーションを起動します。

デバッグするCOBOOLプログラムが起動されたとき、NetCOBOL Studioと接続されてデバッグ操作を開始できます。

このアプリケーションの場合は、Webブラウザでプログラムを実行すると、サーバ上のCOBOLプログラムが起動され、デバッグも開始されます。

4.6.4 デバッグ操作

デバッグ操作には、MeFt/Webアプリケーション固有の部分はありません。“第3章 画面帳票アプリケーションの開発”で作成したアプリケーションと同じデバッグ操作ができます。

4.6.5 デバッグの終了

[実行]メニューから「終了」を選択し、デバッグを終了します。

4.7 通信が切断されるパターンについて

MeFt/Webアプリケーションの運用中に、サーバとクライアントとの間の通信が切断されるパターンとして、次の2つがあります。

- Webブラウザで[戻る]ボタンが押されたり、Webブラウザのアドレスバーで別のURLが指定されたとき
- クライアントマシンの電源が切断されたり、ネットワークが不通になったとき

サーバとクライアントとの間の通信が切断された場合、サーバのアプリケーションはそのことを認識できず、クライアントからの入力を待ちつづける状態となります。しかし、通信切断によりクライアントから応答を返すことができないため、正常にプログラムを終了することができなくなります。プログラムを終了するには強制終了するしかありませんが、ファイルやデータベースが正常にクローズされないため、データに影響が出る可能性があります。

MeFt/Webアプリケーションを構築するにあたっては、このような通信切断時への対応を考慮する必要があります。以下に、とるべき対応を説明します。

Webブラウザで[戻る]ボタンが押されたり、Webブラウザのアドレスバーで別のURLが指定されたとき

Webブラウザで[戻る]ボタンが押されたり、Webブラウザのアドレスバーで別のURLが指定されたりすると、他のページに移動します。その結果、それまで入出力を行っていたページとサーバとの通信が切断されてしまいます。この状況に対応するには、MeFt/Webコントロールの Quitメソッドを使用します。Quitメソッドを使用すると、Webブラウザでイベントが発生したことをアプリケーションに通知することができます。

リモート実行に使用するHTMLで、Webブラウザのページが遷移したときに発生するWindow_onUnloadイベントの処理としてMeFt/Web コントロールのQuitメソッドを実行するように記述すると、Webブラウザで[戻る]ボタンが押されるなどしてページが移動した場合、Quitメソッドが実行されてアプリケーションにコードが通知されます。

HTMLでのQuitメソッドの記述例を以下に示します。

```
<HTML>
  :
<OBJECT
ID="MeFtWeb1"
  :
</OBJECT>
<SCRIPT type="text/javascript">
function Window_onUnload(){
MeFtWeb1.Quit();
}  :
</SCRIPT>
</HTML>
```

COBOLアプリケーションには、表示ファイルのFILE STATUS句に指定された4桁のデータ名の領域に「90N8」で通知されます。それを判定することによってページが移動されたことを知ることができるので、ファイルのクローズやデータベースの切断などの後処理を行うようにします。

MeFt/WebコントロールのQuitメソッドは、“4.4 HTMLの作成”および、“MeFt/Webユーザーズガイド”の“利用者プログラムの中断(Quit)”を参照してください。

クライアントマシンの電源が切断されたり、ネットワークが不通になったとき

クライアントマシンの電源が切断されたり、ネットワークが不通になったりして、サーバとクライアントとの間の通信が切断された状況に対応するには、MeFt/Webサーバの通信監視時間の機能を使用します。

MeFt/Webサーバに通信監視時間を設定すると、設定された通信時間を越えてクライアントからレスポンスがなかった場合、MeFt/Webサーバからアプリケーションにコードを通知します。COBOLアプリケーションには、表示ファイルのFILE STATUS句に指定された4桁のデータ名の領域に「90N7」で通知されます。それを判定することによって通信監視時間を超えて応答がなかったことを知ることができるので、ファイルのクローズやデータベースの切断などの後処理を行うようにします。

MeFt/Webの通信監視時間の設定は、“4.2.1 MeFt/Web動作環境の設定”、および“MeFt/Webユーザーズガイド”の“MeFt/Webの動作環境を設定する”を参照してください。

参考

設定される通信監視時間が長すぎると、サーバとクライアントとの間の通信が切断されているにもかかわらずアプリケーションは起動し続けることになり、サーバの負荷が高まります。

一方、設定される通信監視時間が短すぎると、画面での作業に時間がかかった場合、突然アプリケーションが終了することになってしまいます。

通信監視時間は、業務の内容に応じて適切な値を設定してください。

第5章 効率のよいプログラムのテクニック

ここでは、プログラムの実行時間を短縮するための細かい注意点を述べます。

プログラムは、効率が良いことの他に、読みやすく、拡張しやすいことも必要です。しかし、これから述べる項目の中には、読みやすさに逆行するものもあります。

プログラム作成時には、以下の知識を念頭において、プログラムを読みやすく作り、実行時命令統計機能を使ってボトルネックを発見し、ボトルネックになっている一連の文に対して再度効率向上のための修正を加える、という方法をおすすめします。

また、細かいコーディング技術を駆使するより、プログラムのアルゴリズムを検討する方が、効率を向上させる程度が大きいことがしばしばあります。細部の検討に入る前に、まず、アルゴリズムを改善できないかを考えることも重要です。

5.1 一般的なテクニック

5.1.1 作業場所節の項目

- レコードを設計する際は、よく使われる項目や関連のある項目を一か所に集めて定義するようにします。よく使われる項目が数キロバイト以上の大きな項目にはさまれるような設計は避けてください。
- 転記しても参照されないまま再転記されるような、不要な転記は避けてください。
  例えば、集団項目全体に空白文字を転記した後で、改めて個々の項目に別の値を転記するのでなく、必要な項目だけに空白詰めを行ってください。
- 作業場所節にある項目で、プログラム実行中に値を変更する必要のないものは、VALUE句で初期値を設定してください。

5.1.2 ループの最適化

ループの中では、特に、COBOLの文では見かけ上わからない添字計算や、データの属性の変換など、目的コードの生成を極力抑えるような配慮が必要です。

- ループの中で実行しなくてもいい文はループの外に出してください。
- ループの中では、データ名による添字付けを避け、指標名による添字付けを用いてください。
- ループの中で数字転記や数字比較などに用いる項目は、ループの外であらかじめ最適な属性の項目に移してください。

5.1.3 複合条件の判定順序

ANDだけ、またはORだけで結ばれた複合条件は、括弧がない限り左から右へ順に評価されます。以下の様に書くと、平均実行時間を短縮することができます。

- ORで結ばれている場合は、真になりやすい条件を先に書く
- ANDで結ばれている場合は、偽になりやすい条件を先に書く

5.2 データ項目の属性を理解して使う

5.2.1 英数字項目と数字項目

英数字項目が使用できるところは、数字項目でなく、英数字項目を使ってください。

数字項目は、その中に入っている数値が意味を持っています。
例えば、PIC S9 DISPLAYの項目のビットパターンがX'39'でもX'49'でも、数値としては等しく、共に+9を示すものとみなされます。このため、比較などの目的コードは、英数字項目に比べて遅くなります。

5.2.2 USAGE DISPLAYの数字項目(外部10進項目)

- 各文字位置には、文字"0"～"9"(16進表記でX”30”～X”39”)が入ります。最後の文字の先頭4ビットは符号を表し、数値が正の場合はX"4"、負の場合はX"5"が入ります。

・ 印字表示用として使用してください。演算や比較に使用したときの処理速度は、数字項目の中で最も遅く、使用領域も最も大きくなります。

5.2.3 USAGE PACKED-DECIMALの数字項目(内部10進項目)

・ 4ビットで1桁の数値を表し、最後の4ビットは符号を表します。数値が正の場合はX"C"、負の場合はX"D"、符号なしの場合はX”F”が入ります。

・ 演算や比較に使用したときの処理速度は、外部10進項目よりは速く、2進項目よりは遅くなります。

5.2.4 USAGE BINARY/COMP/COMP-5の数字項目(2進項目)

・ COMP-5の内部表現形式はシステムのエンディアンに従います。BINARYとCOMPは同義で、システムのエンディアンに従わず、常にビッグエンディアンの内部表現形式になります。

・ 表示を目的としない演算、添字に適しています。演算や比較は外部10進項目および内部10進項目より速く、リトルエンディアン・システムではBINARYよりCOMP-5が速くなります。

5.2.5 数字項目の符号

数字項目には、その項目に値を転記する時に絶対値をとる必要がある場合を除き、PICTURE句でSを指定してください。符号をつける場合、SIGN LEADINGやSIGN SEPARATEは指定しないでください。

・ Sの指定がないと、転記する時に絶対値をとるための目的コードが生成されます。

・ SIGN LEADINGやSIGN SEPARATEの指定をすると、符号処理のための余分な命令が生成されます。

5.3 数字転記・数字比較・算術演算の処理時間を短くする

5.3.1 属性

・ 数字転記、数字比較、算術演算では、作用対象のUSAGE句を統一してください。

・ 数字転記、数字比較、加減算では、作用対象の小数部桁数を一致させてください。乗除算では、中間結果の小数部桁数と受取り側項目の小数部桁数を一致させてください。

  ー 一致していないと、演算や比較のたびに、一致させるための変換や桁合せのための目的コードが生成されます。

  ー 乗算C=B*AではdC=db+daを、除算C=B/AではdC=db-daを満たすようにすると、桁合せは発生しません。(da,db,dcはそれぞれA,B,Cの小数部桁数を表します)

・ 算術式では、属性が同じもの同士の演算が多くなるような順に、演算を行ってください。

5.3.2 桁数

桁数は、必要以上に大きくとらないでください。

一般的に、演算や比較の時間は、桁数が多いほど長くなります。

5.3.3 べき乗の指数

べき乗の指数は、整数の定数が最も適しています。次に適しているのは、整数項目です。

整数でない指数が指定されると、COBOLランタイムシステムによる浮動小数点演算となるので、効率は極めて悪くなります。

5.3.4 ROUNDED指定

ROUNDED指定の使用は、必要最小限にしてください。

ROUNDED指定をすると、演算結果が1桁余分に求められ、正負の判定と四捨五入を行う目的コードが生成されます。

5.3.5 ON SIZE ERROR指定

ON SIZE ERROR指定の使用は、必要最小限にしてください。

ON SIZE ERROR指定すると、桁あふれを判定するために以下のような目的コードが生成されます。

- 演算結果が2進で求まる場合

  絶対値をとるなどして受取り側項目に収まる最大値との比較

- 演算結果が内部10進で求まる場合

  受取り側項目の文字位置を超える部分とゼロとの比較

5.3.6 TRUNCオプション

- TRUNCオプションの使用が必要最小限になるように、プログラムを設計してください。
- TRUNCオプションを指定した場合、2進項目間の転記における切り捨ては、10**n(nは受取り側項目の桁数)で除算し、剰余を求めて行っています。したがって、2進項目を多用するプログラムでは、TRUNCオプションを指定すると、大幅に効率が悪くなります。
- NOTRUNCオプションを指定する場合、受取り側項目に文字位置を超える値が入らないようにプログラムを設計しなければなりません。入力データによってそのような問題が起こる可能性がある場合は、不当な入力データを除外するプログラムに変更した上で、NOTRUNCオプションを指定してください。

5.4 英数字転記・英数字比較を効率よく行う

5.4.1 境界合せ

機種によって異なりますが、英数字項目も左端を4バイトまたは8バイト境界に合わせると、一般に効率がよくなります。ただし、英数字項目に対して指定されたSYNCHRONIZED句は注釈とみなされるので、使用しない項目を定義して境界を合わせるようしてください。
境界合せによって全体の使用領域は大きくなります。境界合せは、よく使われる項目を対象にしてください。

5.4.2 項目長

- 英数字転記では、送出し側項目長が受取り側項目長より大きいか等しい時、効率よく実行できます。英数字比較では、両方の項目長が等しい時、効率よく実行できます。
  一方が定数の時は、その長さを他方の項目長に合わせると、効率よく実行できます。
- 上記は、大きな項目(数百バイト以上)の場合、あてはまりません。

5.4.3 転記の統合

複数個のMOVE文が、それらの項目を含む集団項目のMOVE文にまとめられるときは、1個の集団項目のMOVE文にしてください。

5.5 入出力におけるテクニック

5.5.1 SAME RECORD AREA句

SAME RECORD AREA句は、2つ以上のファイルでレコード領域の内容を共有したい場合や、WRITE文の実行後もレコードを使用する必要がある場合に限って指定してください。

SAME RECORD AREA句が指定されたファイルのREAD文およびWRITE文は、レコード領域とバッファ領域の間でレコードの転送を行うため、効率が悪くなります。

5.5.2 ACCEPT文、DISPLAY文

ACCEPT文(DATE、DAY、TIME指定を除く)およびDISPLAY文は、少量のデータの入出力のみに用いてください。

これらの文は、READ文およびWRITE文よりも一般的に効率が悪くなります。

5.5.3 OPEN文、CLOSE文

OPEN文およびCLOSE文は、非常に複雑な内部処理を伴う文であるため、1つのファイルに対するOPEN文およびCLOSE文の実行回数は、必要最小限に抑えてください。

5.6 プログラム間連絡におけるテクニック

5.6.1 副プログラムの分割の基準

1つのシステムを多数のプログラムから構成する場合は、必要以上に小さい副プログラムに分割しないことをおすすめします。

- 副プログラムの呼出しから復帰までに、静的構造の場合でも最低数10ステップの機械文が実行されます。したがって,小さい副プログラムでは、このステップ数が相対的に大きな比重を占めることになり、効率を悪化させてしまいます。副プログラムの手続き部が数百行以上あれば、効率は悪化しません。
- 目的プログラムは初期化・終了ルーチンや作業領域などを必ず持っているので、小さい副プログラムに分割すると全体の領域が多く必要になります。

5.6.2 動的プログラム構造と動的リンク構造

動的プログラム構造(CALL一意名、またはDLOADオプションを指定して翻訳したCALL定数を用いるプログラム構造)は、非常に大きなシステムで、仮想記憶を節約するために仮想記憶上から副プログラムを削除する必要のある場合以外は使用しないでください。
動的リンク構造で済むときは、動的リンク構造を使うことをおすすめします。

- 動的プログラム構造では、副プログラムがローディングされた後も、副プログラムが既にローディングされているかどうかを調べるサーチ処理や、プログラム名のチェックが、呼出しのたびに行われます。そのため、オーバヘッドは、静的プログラム構造より大きくなってしまいます。
- 動的リンク構造では、副プログラムがローディングされた後は、呼出しは直接行われます。そのため、オーバヘッドは、静的リンク構造の場合よりわずかに多い程度です。

5.6.3 CANCEL文

動的プログラム構造を使う場合、CANCEL文は必要最小限に抑えてください。

5.6.4 パラメタの個数

CALL文のUSING指定、およびENTRY文またはPROCEDURE DIVISIONのUSING指定にパラメタを記述すると、個々のパラメタについてアドレスの設定が行われます。したがって、パラメタは可能な限り集団項目にまとめ、USING指定での個数を少なくする方が、効率がよくなります。

5.7 デバッグ機能を使用する

プログラムの誤りを発見する手段として、TRACE機能、CHECK機能、COUNT機能を提供しています。これらの機能をプログラムの運用前に使用することでトラブルの発生防止につながります。なお、運用時には以下の点にご注意ください。

- TRACE機能では、トレース情報の採取など、COBOLプログラムで記述した以外の処理を行います。そのため、TRACE機能使用時には、プログラムのサイズが大きくなり、実行速度も遅くなります。TRACE機能は、デバッグ時にだけ使用し、デバッグ終了後には、翻訳オプションNOTRACEを指定して再翻訳してください。
- COUNT機能では、COUNT情報の採取など、COBOLプログラムで記述した以外の処理を行います。そのため、COUNT機能使用時には、プログラムのサイズが大きくなり、実行速度も遅くなります。COUNT機能は、デバッグ時にだけ使用し、デバッグ終了後には、翻訳オプションNOCOUNTを指定して再翻訳してください。
- CHECKオプションの指定によって、実行時間が2倍以上遅くなることがあります。運用時にCHECK(NUMERIC)を有効にする場合は、可能な限り10進項目を2進項目に変更すると、CHECKオプションによる性能劣化を防ぐことができます。
- CHECK機能およびTRACE機能は、実行時オプションを指定することで、機能を抑制することができます。デバッグ時、一時的にこれらの機能を無効にしたい場合に有用です。

5.8 数字項目の標準規則

ここでは、COBOLプログラムで数字データを扱う上での標準的な規則を述べます。

5.8.1 10進項目

10進項目の入力

- 入力レコード中に10進項目が含まれている場合、誤った内容表現のデータが入らないように注意してください。
  正しいビットパターンが入っているかどうかを確かめるには、字類条件(IF … IS NUMERIC)を使います。コンパイラは、READ文の実行時に、10進項目のビットパターンを検査しません。
- 10進項目を含む集団項目へCORRESPONDING指定のない転記を行う場合、誤ったビットパターンが入らないよう注意してください。この場合も、コンパイラは検査を行いません。

10進項目に誤ったビットパターンが入った場合

- 外部10進項目のゾーン部(符号を持つバイトを除く)が16進数の3でない場合、誤りです。
- 10進項目の数字部が許されるビットパターン(16進数の0～9)でない場合、この項目を転記、演算または比較などに使用すると、結果は規定されません。

誤ったプログラム例

英数字項目から数字項目の転記は数字転記になり、英数字項目を符号なし10進項目にみなして転記します。よって、(a)のSND-DATAはPIC 9(4)とみなし翻訳されます。空白が入っている場合などの不正な値は誤りとなり、結果は予測できません。(a)のMOVE文が予測できないため、(b)の比較結果も予測できません。

```
01 SND-DATA PIC X(4).
01 RSV-DATA PIC 9(4).
...
   MOVE  SPACE  TO  SND-DATA
   ...
   MOVE  SND-DATA  TO  RSV-DATA  …(a)
   IF  RSV-DATA  =  SPACE  THEN  …(b)
```

正しいプログラム例

10進項目に不正な値が設定される可能性がある場合は、(c)のように字類条件(IS NUMERIC)を使用し、正しい値が格納されていることを確認してから使用します。

```
01 SND-DATA PIC X(4).
01 RSV-DATA PIC 9(4).
...
   MOVE  SPACE  TO  SND-DATA
   MOVE   0     TO  RSV-DATA
   ...
   IF SND-DATA IS NUMERIC THEN     …(c)
     MOVE  SND-DATA  TO  RSV-DATA
   ELSE
     DISPLAY NC"データ異常" SND-DATA
   END-IF
```

5.8.2 2進項目

2進項目の値の範囲

2進項目は、一般に、PICTURE句で示された値の範囲より大きい絶対値をもつ値を含むことができます。
NOTRUNC指定の場合、2進項目への転記の際にPICTURE句に合わせた切り捨てが行われなかった結果、正の値が負の値になることもあります。
[参照]“NetCOBOLユーザーズガイド”の“TRUNCオプション”

2進項目のけた数の扱い

2進項目は、PICTURE句より大きい値を含むことができますが、これをDISPLAY文で参照した時は、PICTURE句で示された桁数だけ表示されます。
ON SIZE ERROR指定、またはNOT ON SIZE ERROR指定が記述された演算文では、PICTURE句の指定を超えた値を格納しようとしているかどうかの判定が行われます。
一般に、コンパイラは、2進項目の値はPICTURE句で示された範囲内にあることを前提としてコンパイルを行うため、PICTURE句より大きい値を持つ2進項目を演算などに使用すると、場合によっては異常終了を起こすことがあります。

5.8.3 浮動小数点項目

固定小数点への変換

演算の結果が浮動小数点で求まり、これを固定小数点項目に格納する場合、変換の誤差が最小になるように格納されます。同様に、浮動小数点項目から固定小数点項目へ転記する場合も、変換の誤差が最小になるような値が格納されます。

5.8.4 乗除算の混合時の小数部桁数

次のプログラムの[1]と[2]を比較します。

```
77  X  PIC  S99  VALUE 10.
77  Y  PIC  S9   VALUE 3.
77  Z  PIC  S999V99.
   COMPUTE  Z = X / Y * 100.     [1]
   COMPUTE  Z = X * 100 / Y.     [2]
```

[1]の答はZ=333.00、[2]の答はZ=333.33となります。
この違いは、除算を行うタイミングによって発生します。
どちらも、除算の結果はXとZの小数部桁数の大きい方、すなわち小数第2位まで求められます。[1]では、X/Yが3.33と求められ、これが100倍されます。[2]ではX*100の中間結果である1000がYで割られ、333.33が求まります。
よって精度のよい結果を求めるには、[2]のように、乗算を先に除算を後に行ってください。

5.8.5 絶対値がとられる転記

- 受取り側項目が符号なし数字項目である転記の場合、送出し側項目の絶対値が、受取り側項目に格納されます。
- 符号付き数字項目から英数字項目への転記の場合、送出し側項目の絶対値が、受取り側項目に格納されます。

5.9 注意事項

外部ブール項目のビットパターン

- 外部ブール項目の内容は、16進数で30または31です。それ以外は許されません。
- 0～6ビットに不適当な値を持つ外部ブール項目を比較や演算に使用したときの結果は、規定されません。
- 不適当な値を持つ可能性がある場合は、外部ブール項目を字類条件により検査してください。

レコード領域の参照

- レコード領域は、OPEN文の実行後、またはCLOSE文実行前のみ参照することができます。
- 整列併合用ファイルのレコードは、入力手続きまたは出力手続き内でのみ参照することができます。

第6章 サンプルプログラム

NetCOBOLでは、以下のプログラムをサンプルとして提供しています。

- 6.2 標準入出力を使ったデータ処理(Sample01)
- 6.3 行順ファイルと索引ファイルの操作(Sample02)
- 6.4 表示ファイル機能を使ったプログラム(Sample03)
- 6.5 スクリーン操作機能を使った画面入出力(Sample04)
- 6.6 COBOLプログラム間の呼出し(Sample05)
- 6.7 コマンド行引数の受取り方(Sample06)
- 6.8 環境変数の操作(Sample07)
- 6.9 印刷ファイルを使ったプログラム(Sample08)
- 6.10 印刷ファイルを使ったプログラム(応用編)(Sample09)
- 6.11 FORMAT句付き印刷ファイルを使ったプログラム(Sample10)
- 6.12 データベース機能を使ったプログラム(Sample11)
- 6.13 データベース機能を使ったプログラム(応用編)(Sample12)
- 6.14 Visual Basicからの呼出し(Sample13)
- 6.15 Visual Basicを使った簡易ATM端末処理機能(Sample14)
- 6.16 オブジェクト指向プログラム(初級編)(Sample15)
- 6.17 コレクションクラス(クラスライブラリ)(Sample16)
- 6.18 Unicodeを使用するプログラム(Sample30)
- 6.19 メッセージボックスの出力(Sample31)
- 6.20 他のプログラムの起動(Sample32)
- 6.21 エンコード方式を使用するプログラム(Sample33)

各サンプルでは、サンプルプログラムを動作させる方法として、以下の方法を説明しています。

- NetCOBOL Studioを利用する方法
- MAKEファイルを利用する方法

NetCOBOL Studioを利用してSampleプログラムを動作させる場合は、NetCOBOL Studioでサンプルを利用するための事前準備を行ってください。

6.1 NetCOBOL Studioでサンプルを利用するための事前準備

6.1.1 NetCOBOL Studioの基本概念を理解する

NetCOBOL Studioを使用する上で必要な基本概念(ワークスペース、パースペクティブなど)を理解するために、“2.1.1 NetCOBOLの開発環境”を一読してください。

自動ビルド

自動ビルドは、NetCOBOL Studioのメニューバーから[プロジェクト] > [自動的にビルド]をチェックした場合に設定されます。既定では自動ビルドに設定されています。自動ビルドの詳細は、“NetCOBOL Studioユーザーズガイド”の“自動ビルド”を参照してください。

プロジェクトフォルダー

プロジェクト資産が格納されたフォルダーです。ワークスペースにプロジェクトをインポートした場合は、ワークスペースフォルダー配下に作成されます。

例:ワークスペースフォルダーをC:¥NetCOBOL Studio¥workspaceとした場合のSample01のプロジェクトフォルダー

```
C:¥NetCOBOL Studio¥workspace¥Sample01
```

6.1.2 サンプルを利用するための事前準備

NetCOBOL Studioを使用して、サンプルプログラムをビルド、実行およびデバッグするためには、ワークスペースと呼ばれるフォルダーにサンプルプログラムのプロジェクトを作成する必要があります。

ここでは、以下の順で説明します。

1. ワークスペースを準備する
2. サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする

ワークスペースを準備する

「ワークスペース」とは、NetCOBOL Studioで作成したプロジェクト等の各種リソースを格納するフォルダーのことです。

サンプルプログラムのプロジェクトを格納するワークスペースは、サンプル用のワークスペースとして新規に作成します。ワークスペースの作成方法は、“NetCOBOL Studioユーザーズガイド”の“ワークスペースの設定と切り替え方法”を参照してください。

本マニュアルでは、作成したワークスペースをC:¥NetCOBOL Studio¥workspaceとして説明しています。

サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする

以下の手順で、提供するサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

注意

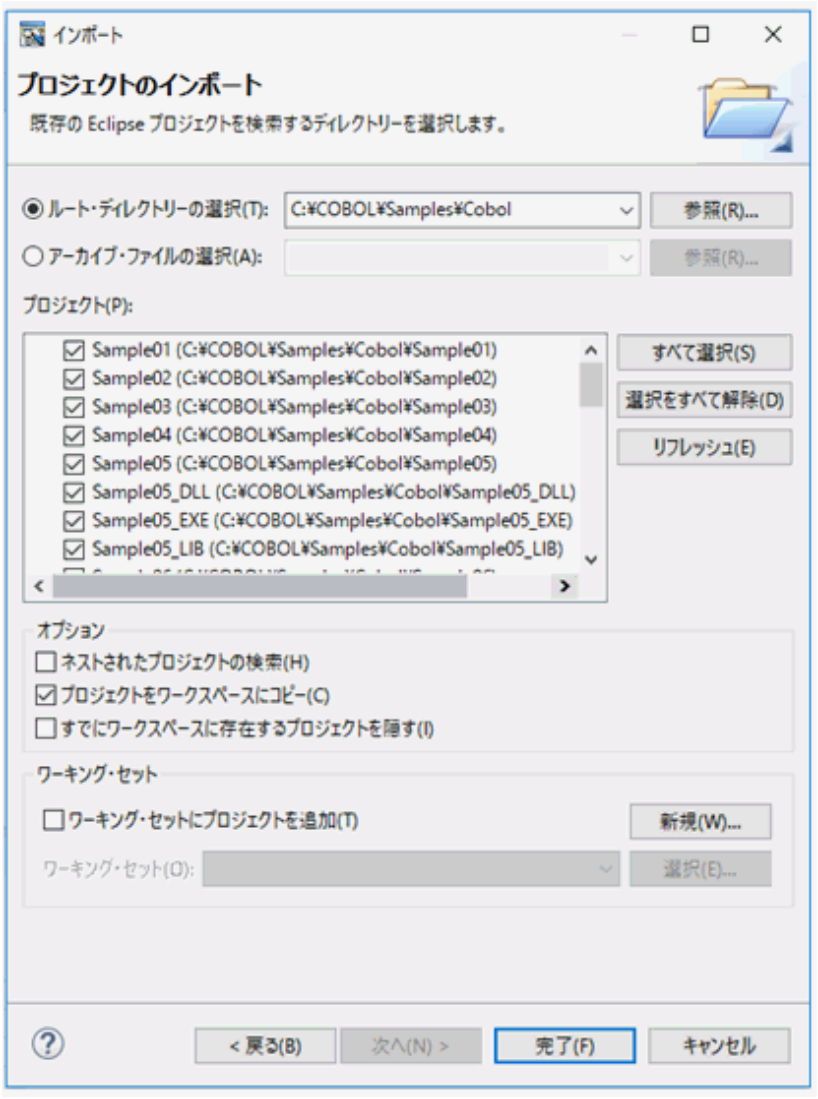
以降では、NetCOBOLのインストール先フォルダーをC:¥COBOLとして説明しています。フォルダー名がC:¥COBOLになっているところは、NetCOBOLをインストールしたフォルダーに変更してください。

1. [スタート] > お使いのNetCOBOL製品名 > [NetCOBOL Studio(x64) - Eclipce 4.6基盤]を選択し、NetCOBOL Studioを起動します。
2. メニューバーから[ファイル] > [インポート]を選択します。

   →[インポート]ウィザードが起動されます。
3. [一般] > [既存プロジェクトをワークスペースへ]を選択して、[次へ]ボタンをクリックします。
4. [ルート・ディレクトリーの選択]を選択し、[参照]ボタンをクリックします。

   →[フォルダーの参照]ダイアログボックスが表示されます。
5. プロジェクトを含んでいるフォルダー(COBOLサンプルプログラムの格納先(ここでは、C:¥COBOL¥Samples¥COBOL))を選択し、[OK]ボタンをクリックします。

6. [プロジェクト]ペインにCOBOLサンプルプログラムのプロジェクトが表示されていることを確認し、[すべてを選択]ボタンをクリックします。
   [プロジェクトをワークスペースにコピー]をチェックして、[終了]ボタンをクリックします。



→NetCOBOL Studioのワークスペースにサンプルプログラムのプロジェクトがインポートされます。

6.1.3 サンプルを利用する上での注意事項

各サンプルプログラムでは、以下のNetCOBOL Studioのプロジェクト関連ファイルを提供しています。これらのプロジェクト関連ファイルは、編集しないでください。これらのファイルを変更するとアプリケーションが正しく動作しません。

- .Settings¥org.eclipse.core.resources.prefs
- .CobolOptions
- .project

6.2 標準入出力を使ったデータ処理(Sample01)

ここでは、本製品で提供するサンプルプログラム-Sample01-について説明します。

Sample01は、COBOLの小入出力機能を使って、コンソールウィンドウからデータを入力したり、コンソールウィンドウにデータを出力したりするプログラムの例を示します。小入出力機能の使い方の詳細は、“NetCOBOL ユーザーズガイド”の“小入出力機能”を参照してください。

概要

コンソールウィンドウからアルファベット1文字(小文字)を入力し、入力したアルファベットで始まる単語をコンソールウィンドウに出力します。

提供プログラム

- Sample1.cob（COBOLソースプログラム）
- Makefile（メイクファイル）
- COBOL85.CBR（実行用の初期化ファイル）

使用しているCOBOLの機能

- 小入出力機能(コンソールウィンドウ)

使用しているCOBOLの文

- ACCEPT文
- DISPLAY文
- EXIT文
- IF文
- PERFORM文

6.2.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

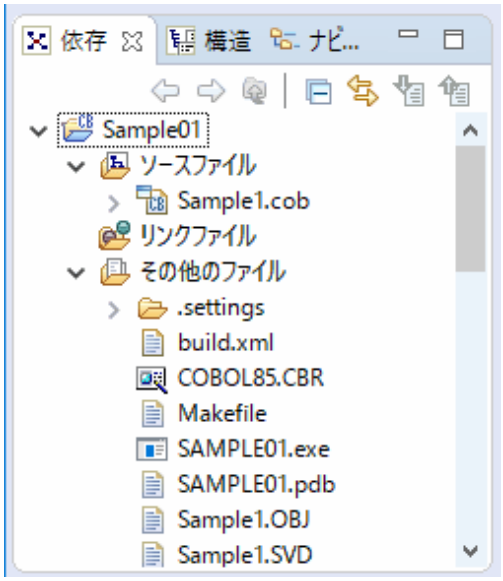
   “ワークスペースを準備する”

2. ［依存］ビューを確認し、Sample01プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample01プロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. [その他のファイル]にSample1.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、Sample1.exeが作成されます。

デバッグ

NetCOBOL Studioのデバッグ機能を使用したSample01のデバッグ手順は、“NetCOBOL Studio ユーザーズガイド”の“チュートリアル”を参照してください。

プログラムの実行

[依存]ビューからSample01プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択します。

→アルファベット1文字を入力するとその文字が先頭である英単語が表示されます。

6.2.2 COBOLコマンドとリンクコマンドを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample01>COBOL.exe -M Sample1.cob
C:¥COBOL¥Samples¥COBOL¥Sample01>LINK /OUT:Sample1.exe Sample1.obj F4AGCIMP.lib MSVCRT.lib
```

プログラムの実行

コマンドプロンプトまたはエクスプローラからSample1.exeを実行します。

→ アルファベット1文字を入力するとその文字が先頭である英単語が表示されます。

6.2.3 MAKEファイルを利用する場合

プログラムの翻訳・リンク

サンプルプログラムとして提供されているMakefileを利用してプログラムを翻訳・リンクするには、Makefileが格納されているフォルダーでnmakeコマンドを実行します。

```
C:¥COBOL¥Samples¥COBOL¥Sample01>nmake
```

翻訳およびリンク終了後、Sample1.exeが作成されていることを確認してください

プログラムの実行

COBOLコマンドとリンクコマンドを利用する場合と同じです。

6.3 行順ファイルと索引ファイルの操作(Sample02)

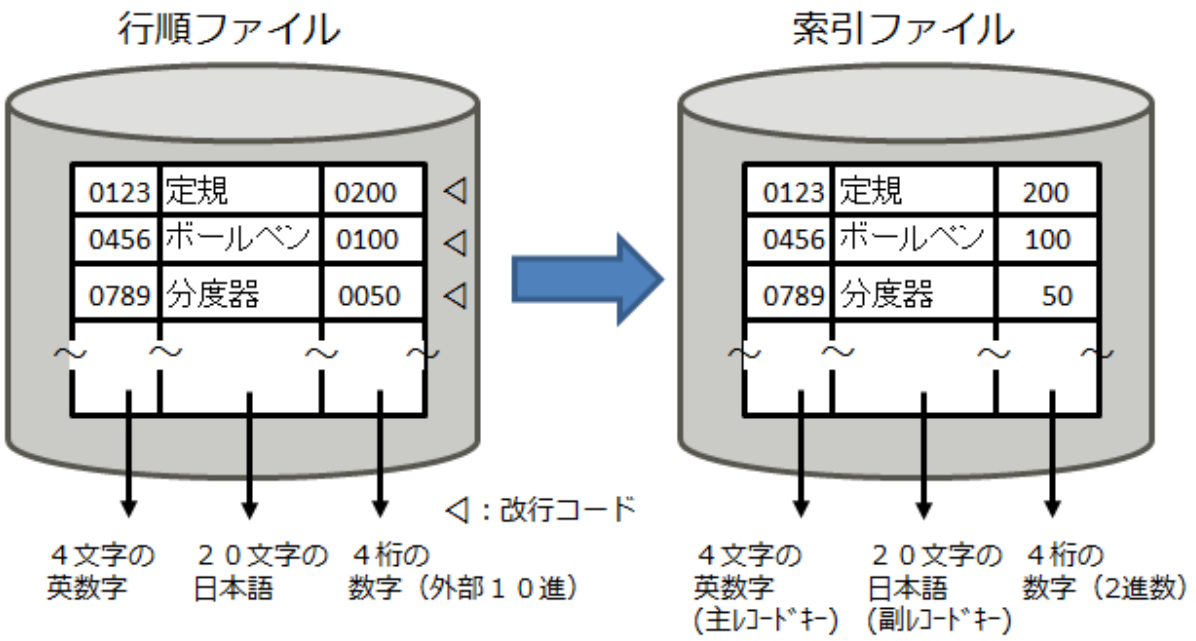
ここでは、本製品で提供するサンプルプログラム-Sample02-について説明します。

エディタを使って作成したデータファイル(行順ファイル)を読み込み、マスタファイル(索引ファイル)を作成するプログラムの例を示します。行順ファイルおよび索引ファイルの使い方の詳細は、“NetCOBOL ユーザーズガイド”の“ファイルの処理”を参照してください。

Sample02で作成したマスタファイル(索引ファイル)は、Sample05、Sample07およびSample10で入力ファイルとして使用します。

概要

商品コード、商品名および単価が入力されているデータファイル(行順ファイル)を読み込み、商品コードを主レコードキー、商品名を副レコードキーとする索引ファイルを作成します。



提供プログラム

- Sample2.cob（COBOLソースプログラム）
- Syohinm.cbl（登録集原文）
- Datafile（データファイル）
- Makefile（メイクファイル）
- COBOL85.CBR（実行用の初期化ファイル）

使用しているCOBOLの機能

- 行順ファイル(参照)

- 索引ファイル(創成)

使用しているCOBOLの文

- CLOSE文
- EXIT文
- GO TO文
- MOVE文
- OPEN文
- READ文
- WRITE文
- COPY文

6.3.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

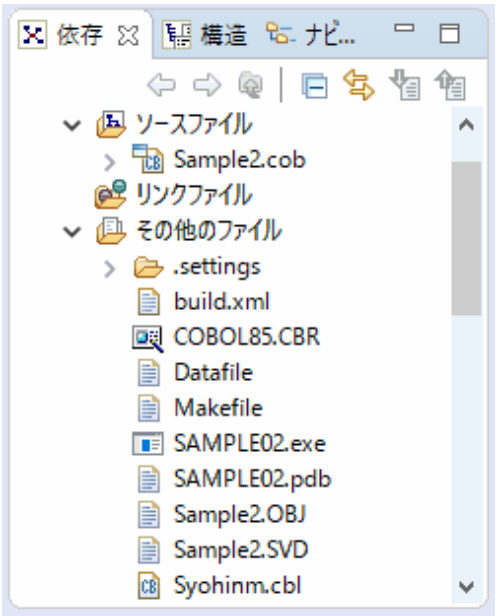
   “ワークスペースを準備する”

2. [依存]ビューを確認し、Sample02プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample02プロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. [その他のファイル]にSample2.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、Sample2.exeが作成されます。

実行環境情報の設定

1. [実行環境設定]ツールを起動します。

   [実行環境設定]ツールを起動する方法には以下があります。

   - [スタート]> お使いのNetCOBOL製品 > [実行環境設定ツール]を選択します。
   - [NetCOBOLコマンドプロンプト]から、実行環境設定ツール(COBENVUT.exe)を起動します。

2. [実行環境設定ツール]のメニューバーから、[ファイル] > [開く]を選択します。

   → [実行用の初期化ファイルの指定]ウィンドウが表示されます。

3. 実行可能プログラム(Sample2.exe)が存在するフォルダーのCOBOL85.CBRを選択し、[開く]ボタンをクリックします。
   NetCOBOL Studioからビルドした場合、実行可能プログラムは、プロジェクトフォルダーに作成されています。

   → 実行用の初期化ファイルの内容が表示されます。

4. [共通]タブを選択し、以下の設定を確認します。

   - ファイル識別名INFILEに、データファイル(行順ファイル)のファイル名(DATAFILE)が指定されている。
   - ファイル識別名OUTFILEに、マスタファイル(索引ファイル)のファイル名(MASTER)が指定されている。

   ```
   INFILE=.¥DATAFILE
   OUTFILE=.¥MASTER
   ```

   相対パスでファイルを指定する場合、カレントフォルダーからの相対パスになります。
   NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択して実行する場合、カレントフォルダーはプロジェクトフォルダーです。

5. [適用]ボタンをクリックします。

   → 設定した内容が実行用の初期化ファイルに保存されます。

6. [ファイル]メニューの[終了]を選択し、実行環境設定ツールを終了します。

参考

ファイル識別名のINFILEおよびOUTFILEは、COBOLソースプログラムのASSIGN句に指定されているファイル参照子です。ファイル参照子は、COBOLプログラムおよび実際の媒体(ファイル)を対応付けるために使用します。

プログラムの実行

[依存]ビューからSample02プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択します。

実行結果

実行の終了メッセージは、ウィンドウに表示されません。実行終了後、商品コードをキーとする索引ファイル(MASTER)が作成されます。

索引ファイル(MASTER)が正しく作成されたことを確認する場合は、COBOLファイルユーティリティを使用してください。COBOLファイルユーティリティの[表示]機能では、索引ファイルのレコードを表示することができます。使い方の詳細は、“NetCOBOL ユーザーズガイド”の“COBOLファイルユーティリティ”を参照してください。

6.3.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample02>nmake
```

翻訳およびリンク終了後、Sample2.exeが作成されていることを確認してください

実行環境情報の設定

NetCOBOL Studioを利用する場合と同じです。

プログラムの実行

コマンドプロンプトまたはエクスプローラからSample2.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.4 表示ファイル機能を使ったプログラム(Sample03)

ここでは、本製品で提供するサンプルプログラム-Sample03-について説明します。

Sample03では、表示ファイル機能を使って、画面から入力したデータを画面に出力し、さらにデータを印刷装置に出力するプログラムの例を示します。画面入出力を行う場合の、表示ファイル機能の使い方は“NetCOBOL ユーザーズガイド”の“表示ファイル(画面入出力)の使い方”を、印刷を行う場合の表示ファイル機能の使い方は、“NetCOBOL ユーザーズガイド”の“表示ファイル(帳票印刷)の使い方”を参照してください。なお、このプログラムを実行するには、MeFtが必要です。

概要

ディスプレイ装置に表示された入力画面に商品コードおよび個数を入力すると、商品コードをキーにマスタファイルを検索し、商品名、単価および金額を求め、さらに合計金額を計算し、ディスプレイ装置に表示します。また、帳票を印刷装置に出力します。

Sample03で入力したデータは、売上げファイル(索引ファイル)に格納します。売上げファイルは、Sample10で入力ファイルとして使用します。

提供プログラム

- Sample3.cob(COBOLソースプログラム)
- Denpyou.cob(COBOLソースプログラム)
- Uriage.cbl(登録集原文)
- Denpyo01.smd(画面定義体)
- Denpyo02.smd(帳票定義体)
- Mefwrc(ウィンドウ情報ファイル)
- Mefprc(プリンタ情報ファイル)
- Makefile（メイクファイル）

Syohinm.cbl(登録集原文)は、Sample02の提供ファイルを使用します。

使用しているCOBOLの機能

- 表示ファイル機能(画面入出力)

- 表示ファイル機能(帳票印刷)
- 索引ファイル(参照/創成)
- 登録集の取込み
- 小入出力機能(メッセージボックス)
- プロジェクト管理機能

使用しているCOBOLの文

- OPEN文
- READ文
- WRITE文
- CLOSE文
- PERFORM文
- DISPLAY文
- IF文
- MOVE文
- GO TO文
- EXIT文
- COPY文

プログラムを実行する前に

- MeFtのセットアップを行い、使用できる状態にしておいてください。
- Sample02で作成されるマスタファイルを使用します。
  “6.3 行順ファイルと索引ファイルの操作(Sample02)”を実行しておきます。
- Sample03では、Sample02で作成されるマスタファイルにある商品コードを入力しないと入力エラーとなります。
  “6.6 COBOLプログラム間の呼出し(Sample05)”を実行し、入力データを印刷しておくと便利です。

6.4.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

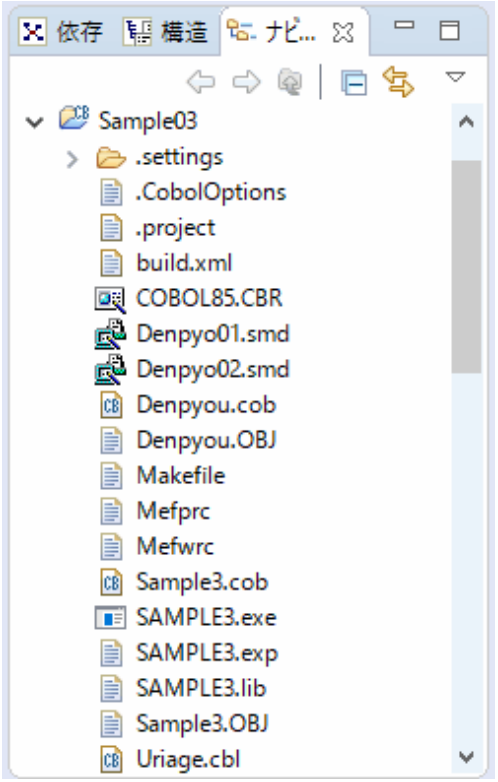
   “ワークスペースを準備する”

2. [依存]ビューを確認し、Sample03プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample03プロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. [その他のファイル]にSample3.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、Sample3.exeが作成されます。

ウィンドウ情報ファイル・プリンタ情報ファイルの設定

表示ファイルを実行するために必要な情報を設定します。

実行を行う前に、ウィンドウ情報ファイル(MEFWRC)およびプリンタ情報ファイル(MEFPRC)の下線部の情報を、エディタを使って変更してください。

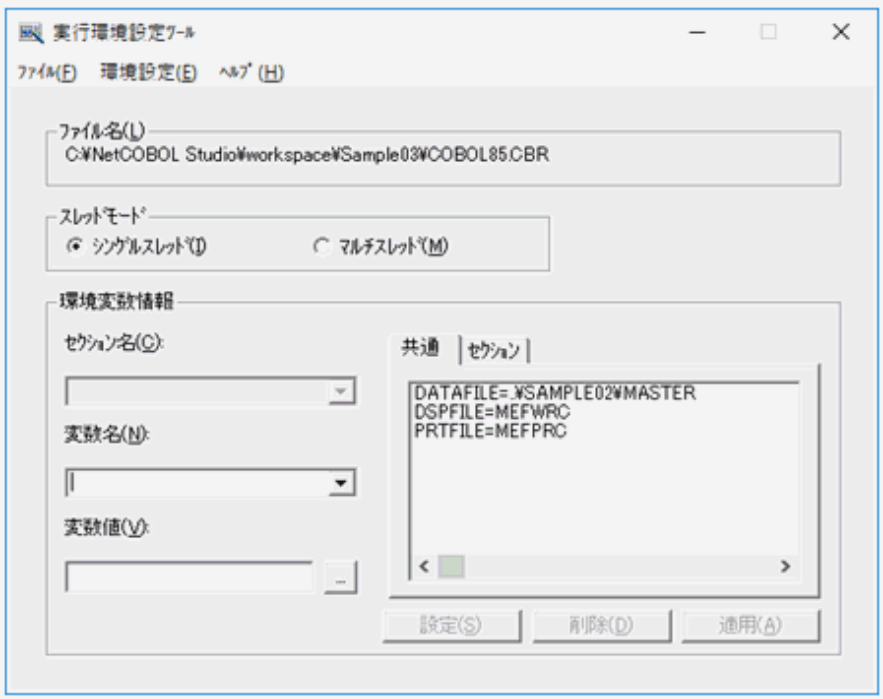
```
MEDDIR C:¥COBOL¥SampleS¥COBOL¥Sample03
```

- MEDDIR:画面帳票定義体を格納したフォルダーのパス名

実行環境情報の設定

1. ［実行環境設定］ツールを起動するには、COBOL85.CBRをダブルクリックします。

   → 実行用の初期化ファイルの内容が表示されます。



2. [共通]タブを選択し、以下を設定します。

   ー ファイル識別名DATAFILEに、Sample02で作成したマスタファイル(MASTER)を指定します。

   ー ファイル識別名DSPFILEに、ウィンドウ情報ファイル(MEFWRC)を指定します。

   ー ファイル識別名PRTFILEに、プリンタ情報ファイル(MEFPRC)を指定します。

3. [適用]ボタンをクリックします。

   → 設定した内容が実行用の初期化ファイルに保存されます。

4. [ファイル]メニューの“終了”を選択し、実行環境設定ツールを終了します。

参考

ファイル識別名のDATAFILE、DSPFILEおよびPRTFILEは、COBOLソースプログラムのASSIGN句に指定されているファイル参照子です。表示ファイルを使用する場合、ファイル識別名DSPFILEにはウィンドウ情報ファイル、PRTFILEにはプリンタ情報ファイルのパス名を設定します。

プログラムの実行

1. ［依存］ビューからSample03プロジェクトを選択し、NetCOBOL Studioのメニューバーから［実行(R)］>［実行(S)］>［COBOLアプリケーション］を選択します。

   → ディスプレイ装置に以下に示す画面が表示されます。

2. 商品コードおよび数量を入力します。次の項目に移動するには、ENTERキーまたはタブキーを押します。

   Sample03では、Sample02で作成されたマスタファイルにある商品コードを入力しないと入力エラーとなります。商品コードの値は、Sample02のDATAFILEまたはSample05を実行し、マスタファイルの内容を参照してから入力してください。

   入力を終了し、計算を行うには、F1キーを押します。

→ 商品名、単価、金額、合計金額が表示されます。エラーメッセージが表示された場合、メッセージの内容を確認し、再度入力してください。

3. 続けて処理を行う場合は、F2キーを押してください。
4. 帳票の出力を行う場合は、F4キーを押してください。

   → 表示されているデータの内容が通常使うプリンターに出力されます。
5. 処理を終了する場合は、F3キーを押してください。

   → 実行終了後、入力データを格納した索引ファイル(SALES)が、Sample03のフォルダーに作成されます。

注意

画面帳票定義体(DENPYO01.SMD、DENPYO02.SMD)をテキストエディタなどで開いて保存すると、ファイルの内容が破壊されることがあります。画面帳票定義体を更新する場合には、FORMのエディタ以外は使用しないでください。

6.4.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample03>nmake
```

翻訳およびリンク終了後、Sample3.exeが作成されていることを確認してください

実行環境情報の設定

NetCOBOL Studioを利用する場合と同じです。

プログラムの実行

コマンドプロンプトまたはエクスプローラからSample3.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.5 スクリーン操作機能を使った画面入出力(Sample04)

ここでは、本製品で提供するサンプルプログラム-Sample04-について説明します。

Sample04では、スクリーン操作機能を使って、画面からデータを入力するプログラムの例を示します。スクリーン操作機能の使い方の詳細は、“NetCOBOL ユーザーズガイド”の“スクリーン操作機能の使い方”を参照してください。

概要

ディスプレイ画面から従業員番号および氏名を入力し、従業員番号を主レコードキー、氏名を副レコードキーとする索引ファイルを作成します。

提供プログラム

- Sample4.cob（COBOLソースプログラム）
- Makefile（メイクファイル）
- Sample4.KBD（キー定義ファイル）
- COBOL85.CBR（実行用の初期化ファイル）

使用しているCOBOLの機能

- スクリーン操作機能
- 索引ファイル(参照)

使用しているCOBOLの文

- ACCEPT文
- CLOSE文
- DISPLAY文
- EXIT文
- GO TO文
- IF文
- MOVE文
- OPEN文
- WRITE文

6.5.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

参考

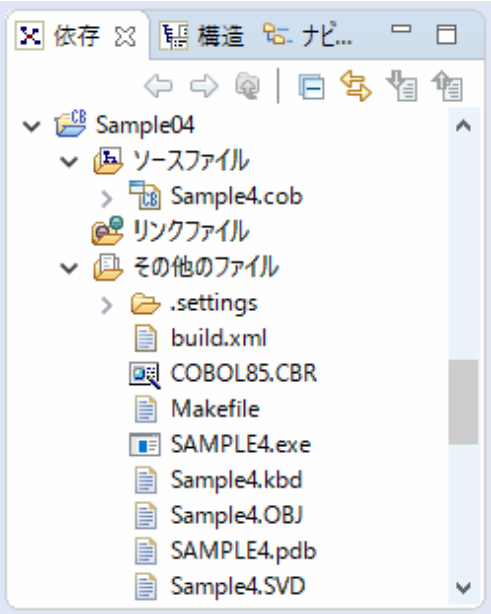
“ワークスペースを準備する”

2. [依存]ビューを確認し、Sample04プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

参考

"サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする"

3. [依存]ビューからSample04プロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. [その他のファイル]にSample4.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、Sample4.exeが作成されます。

実行環境情報の設定

1. [実行環境設定]ツールを起動します。

   [実行環境設定]ツールを起動する方法には以下があります。

   ー [スタート]> お使いのNetCOBOL製品 > [実行環境設定ツール]を選択します。

   ー [NetCOBOLコマンドプロンプト]から、実行環境設定ツール(COBENVUT.exe)を起動します。

2. [実行環境設定ツール]のメニューバーから、[ファイル] > [開く]を選択します。

   → [実行用の初期化ファイルの指定]ウィンドウが表示されます。

3. 実行可能プログラム(Sample4.exe)が存在するフォルダーのCOBOL85.CBRを選択し、[開く]ボタンをクリックします。
   NetCOBOL Studioからビルドした場合、実行可能プログラムは、プロジェクトフォルダーに作成されています。

   → 実行用の初期化ファイルの内容が表示されます。

4. [共通]タブを選択し、以下の設定を確認します。

   ー ファイル識別名OUTFILEに、マスタファイル名(MASTER)が指定されている。

ー F2キーの入力だけが有効となるように設定されたキー定義ファイル(Sample4.KBD)が、環境変数情報@CBR_SCR_KEYDEFFILEに設定されている。

```
OUTFILE=.¥MASTER
@CBR_SCR_KEYDEFFILE=.¥Sample4.KBD
```

相対パスでファイルを指定する場合、カレントフォルダーからの相対パスになります。

NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択して実行する場合、カレントフォルダーはプロジェクトフォルダーです。

5. [適用]ボタンをクリックします。

   → 設定した内容が実行用の初期化ファイルに保存されます。

6. [ファイル]メニューの[終了]を選択し、実行環境設定ツールを終了します。

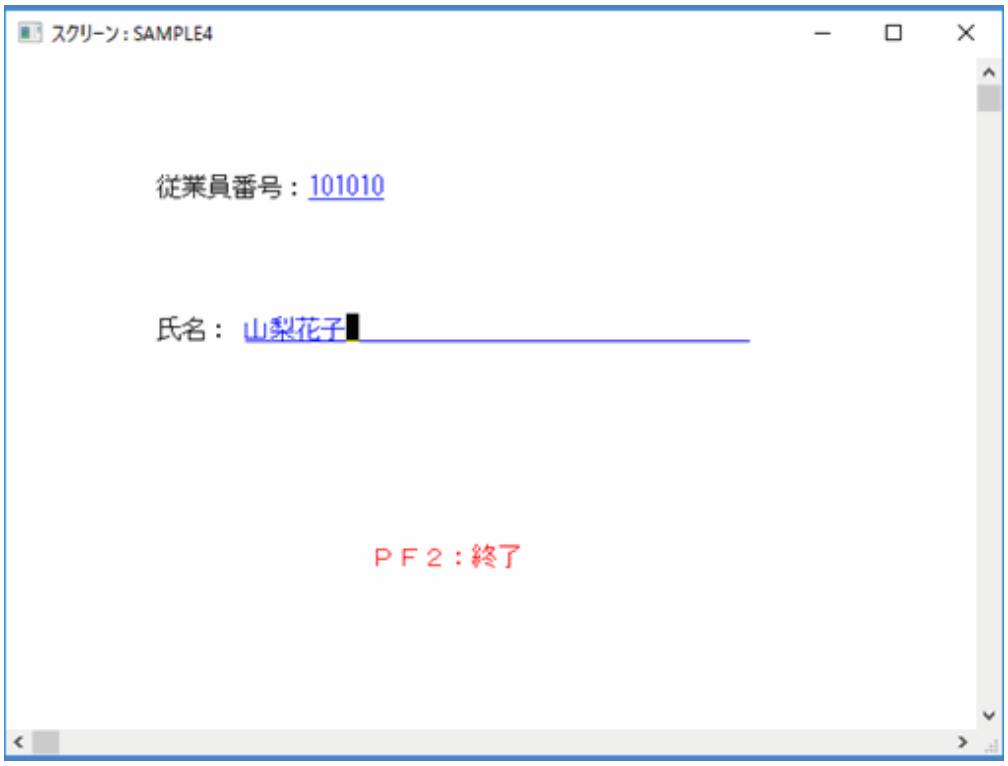
プログラムの実行

[依存]ビューからSample04プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択します。

実行結果

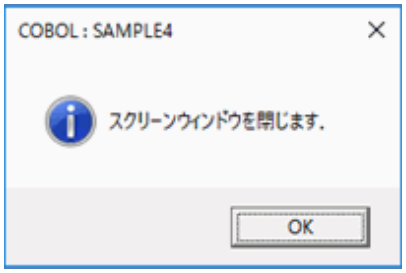
1. ディスプレイ装置に従業員番号および氏名を入力するための画面が表示されます。

2. 登録する従業員データとして、従業員番号(6桁の数字)および氏名(20文字以内の日本語文字)を入力します。ただし、従業員番号は昇順に入力してください。入力後、ENTERキーを押してください。



→ 設定した内容がマスタファイルに登録され、次の情報を入力するために画面がクリアされます。

3. 処理を終了する場合は、F2キーを押してください。



→ 実行終了後、従業員番号を主レコードキー、氏名を副レコードキーとする索引ファイル(MASTER)が、Sample04のフォルダーに作成されます。

6.5.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample04>nmake
```

翻訳およびリンク終了後、Sample4.exeが作成されていることを確認してください

実行環境情報の設定

NetCOBOL Studioを利用する場合と同じです。

プログラムの実行

コマンドプロンプトまたはエクスプローラからSample4.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.6 COBOLプログラム間の呼出し(Sample05)

ここでは、本製品で提供するサンプルプログラム-Sample05-について説明します。

Sample05では、主プログラムから、副プログラムを呼び出すプログラムの例を示します。なお、Sample05では、正書法の自由形式を使用して記述しています。

概要

商品コード、商品名および単価が格納されているマスタファイル(Sample02で作成した索引ファイル)の内容を印刷可能文字に変換して作業用のテキストファイル(*.TMP)に格納し、印刷します。なお、作業用のファイルの名前は、プログラム実行時にパラメタで指定します。

提供プログラム

- Sample05_EXE¥Sample5.cob
- Sample05_EXE¥COBOL85.CBR（実行用の初期化ファイル）
- Sample05_DLL¥Insatsu.cob（COBOLソースプログラム）
- Sample05_LIB¥S_rec.cbl（登録集原文）
- Sample05¥Makefile(メイクファイル)

使用しているCOBOLの機能

- COBOLソリューションプロジェクト
- COBOLリソースプロジェクト
- プログラム間連絡機能
- 登録集の取込み
- 小入出力機能(メッセージボックス)
- 印刷ファイル
- 索引ファイル(参照)
- 行順ファイル(創成)
- 実行時パラメタの受渡し
- 正書法の自由形式

使用しているCOBOLの文

- CALL文
- DISPLAY文
- EXIT文
- GO TO文
- MOVE文
- OPEN文
- READ文

・ WRITE文

プログラムの内容

ソースの記述(自由形式)

正書法の自由形式を使用してCOBOLソースプログラムを記述する例を以下に示します。(C:¥COBOL¥Samples¥COBOL¥Sample05¥Sample5.cob)

```
カラム
位置
1 --|-- 10 --|-- 20 --|-- 30 --|-- 40 --|-- 50 --|-- 60 --|-- 70
------------------------------------------------------------------------
IDENTIFICATION DIVISION.
 PROGRAM-ID. Sample5.
*>  このサンプルプログラムは自由形式の正書法で記述されています。
*>  翻訳時には翻訳オプションSRFを使用して、正書法の形式として
*>  自由形式(FREE)を指定してください。
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
  SPECIAL-NAMES.
    SYSERR IS メッセージ出力先.
                              :
DATA DIVISION.
 FILE SECTION.
 FD マスタファイル.
 01 マスタレコード.
     02 商品レコード.
         03 商品コード      PIC X(4).
         03 商品名          PIC N(20).
         03 単価            PIC 9(4)  BINARY.
                              :
PROCEDURE DIVISION USING  パラメタ.
*>（１）作業ファイル名を決定します。
     IF パラメタ長 = 0
       DISPLAY NC"パラメタが指定されていません。"-
     "パラメタを指定してください。"
          UPON  メッセージ出力先
       GO TO  処理終了.
                              :
処理終了.
    EXIT PROGRAM.
END PROGRAM Sample5.
------------------------------------------------------------------------
```

自由形式では、COBOLの文および注記は、行の任意の文字位置から書くことができます。行の最初の空白でない文字の並びが“*>”である場合、その行は注記行とみなされます。また、上記の例のような方法で文字定数や日本語文字定数などを複数の行に分けて書くことができます。

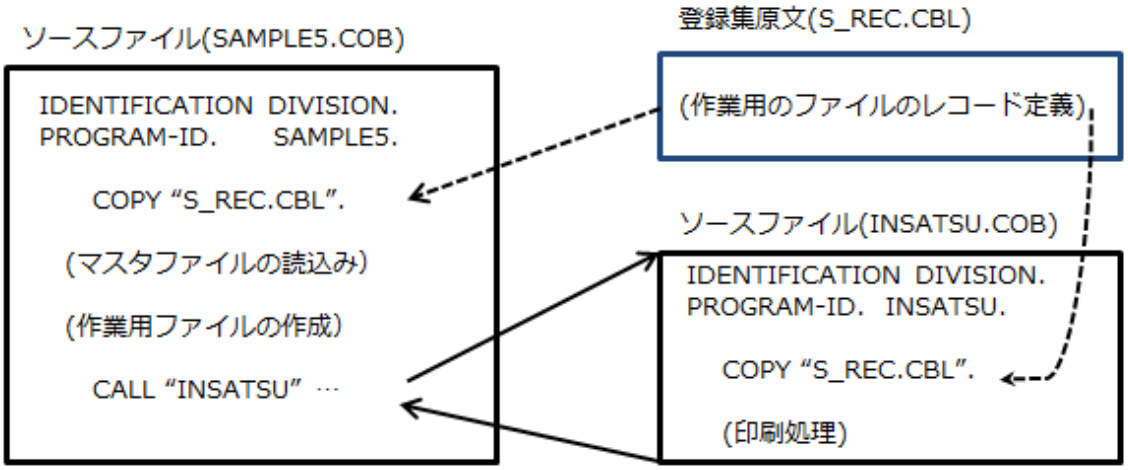
正書法の自由形式の詳細は、“COBOL文法書”を参照してください。

参考

翻訳時、COBOLソースプログラムおよび登録集の正書法の形式は、翻訳オプションSRFを用いて、それぞれ指定します。このため、1つのCOBOLソースプログラムに正書法の形式の異なる複数の登録集を取り込むことはできません。

また、画面帳票定義体を自由形式のCOBOLソースプログラムに取り込んで使用する場合、登録集の正書法の形式として可変形式を指定してください。[参照]“NetCOBOL ユーザーズガイド”の“SRF (正書法の種類)”

ファイルの依存関係



プログラムを実行する前に

Sample02で作成されるマスタファイルを使用します。
“6.3 行順ファイルと索引ファイルの操作(Sample02)”を実行しておきます。

6.6.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

   “ワークスペースを準備する”

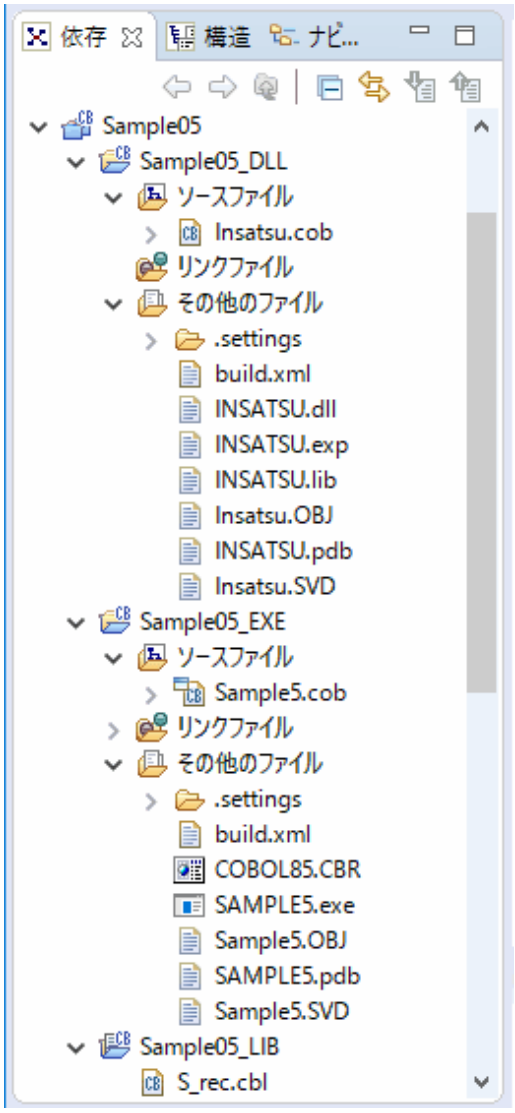
2. [依存]ビューを確認して、以下のプロジェクトがなければ、サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   - Sample5(COBOLソリューションプロジェクト)
   - Sample05_EXE(COBOLプロジェクト)
   - Sample05_DLL(COBOLプロジェクト)
   - Sample05_LIB(COBOLリソースプロジェクト)

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからプロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. **[ソリューションプロジェクトのビルド設定]**

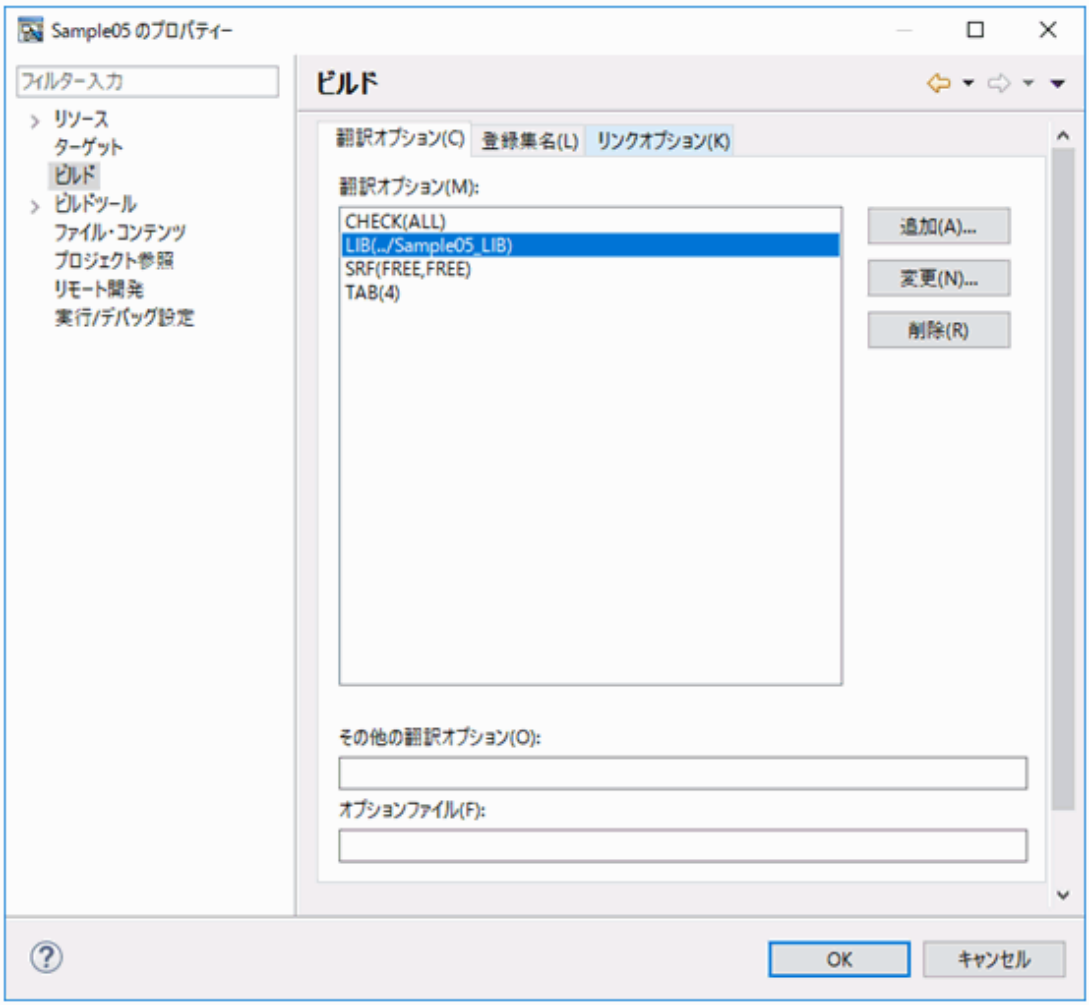
Sample05ソリューションプロジェクトの[ビルド]ページには、プロジェクトに共通なオプションを設定します。

共通オプションとして翻訳オプションLIBを設定します。

a. 翻訳オプションの設定は、NetCOBOL Studioの[依存]ビューからSample05プロジェクトを選択し、コンテキストメニューから[プロパティ]を選択します。

→ [プロパティ]ダイアログボックスが表示されます。

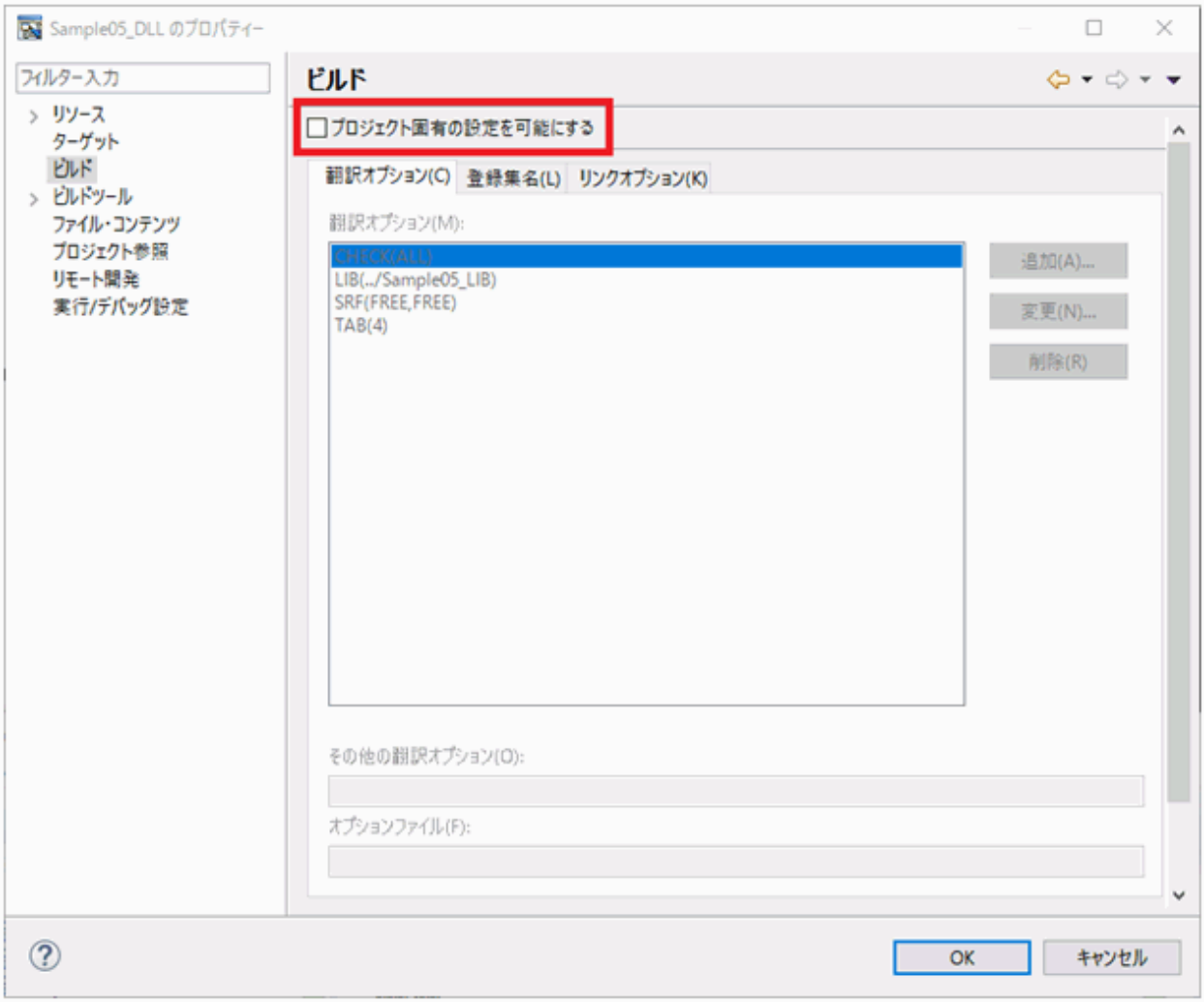
b. 左ペインから[ビルド]を選択すると、[ビルド]ページが表示されます。[翻訳オプション]タブを選択して、設定オプションの内容を確認します。



ここでは、翻訳オプションLIBにS_REC.cblの格納フォルダー（Sample05_LIBプロジェクトのフォルダー:"../Sample05_LIB"）が指定されていることと、翻訳オプションSRF(FREE,FREE)が指定されていることを確認して、[OK]ボタンをクリックします。

5. ［副プログラム／主プログラムのビルド設定］

上と同じ方法で、Sample05_DLLプロジェクトおよびSample05_EXEプロジェクトの［ビルド］ページを表示します。

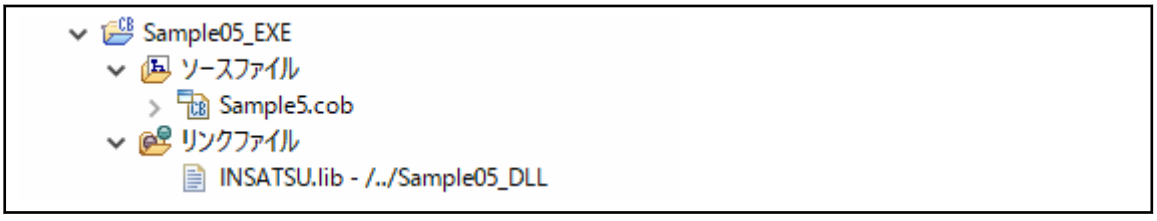


［プロジェクト固有の設定を可能にする］がチェックされていないことを確認します。チェックされている場合、ソリューションプロジェクトのビルド設定が有効にならないため、チェックを外してください。

6. ［主プログラムにおけるライブラリ参照］

主プログラム(Sample05.exe)はSample05_DLLプロジェクトが出力するライブラリファイル（INSATU.lib）をリンクします。

このリンクファイルは、［依存］ビューから、Sample05_EXEプロジェクトの［リンクファイル］に“INSATU.lib”が追加されていることを確認してください。追加されていない場合は、［リンクファイル］のコンテキストメニューから［ファイルの追加］を選択して、Sample05_DLLプロジェクト配下のINSATU.libを指定してください。



7. ［ソリューションプロジェクトのビルド］

Sample05_EXEプロジェクトの［その他のファイル］にSample5.exeが作成されていない場合（自動ビルドが実行されていない場合）、Sample05プロジェクトを選択して、コンテキストメニューから［プロジェクトの再ビルド］を選択します。

→ ソリューションプロジェクト配下のすべてのプロジェクトのビルドが行われ、Sample5.exeが作成されます。

実行環境情報の設定

1. [実行環境設定]ツールを起動します。

   [実行環境設定]ツールを起動する方法には以下があります。

   ― [スタート]> お使いのNetCOBOL製品 > [実行環境設定ツール]を選択します。

   ― [NetCOBOLコマンドプロンプト]から、実行環境設定ツール(COBENVUT.exe)を起動します。

2. [実行環境設定ツール]のメニューバーから、[ファイル] > [開く]を選択します。

   → [実行用の初期化ファイルの指定]ウィンドウが表示されます。

3. 実行可能プログラム(Sample5.exe)が存在するフォルダーのCOBOL85.CBRを選択し、[開く]ボタンをクリックします。
   NetCOBOL Studioからビルドした場合、実行可能プログラムは、Sample05_EXEプロジェクトフォルダーに作成されています。

   → 実行用の初期化ファイルの内容が表示されます。

4. [共通]タブを選択し、以下の設定を確認します。

   ― 環境変数情報@MGPRM(実行時パラメタの指定)に、作業用ファイルのベース名(sample5)が指定されている(英数字8文字以内)こと。
   ここで指定した名前に拡張子TMPを付加した名前のファイルが作業用ファイルとして作成されます。

   ― ファイル識別名INFILEに、Sample02で作成したマスタファイル(MASTER)が指定されていること。

   ```
   INFILE=..¥Sample02¥MASTER
   ```

   相対パスでファイルを指定する場合、カレントフォルダーからの相対パスになります。

   NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択して実行する場合、カレントフォルダーはプロジェクトフォルダーです。

5. [適用]ボタンをクリックします。

   → 設定した内容が実行用の初期化ファイルに保存されます。

6. [ファイル]メニューの[終了]を選択し、実行環境設定ツールを終了します。

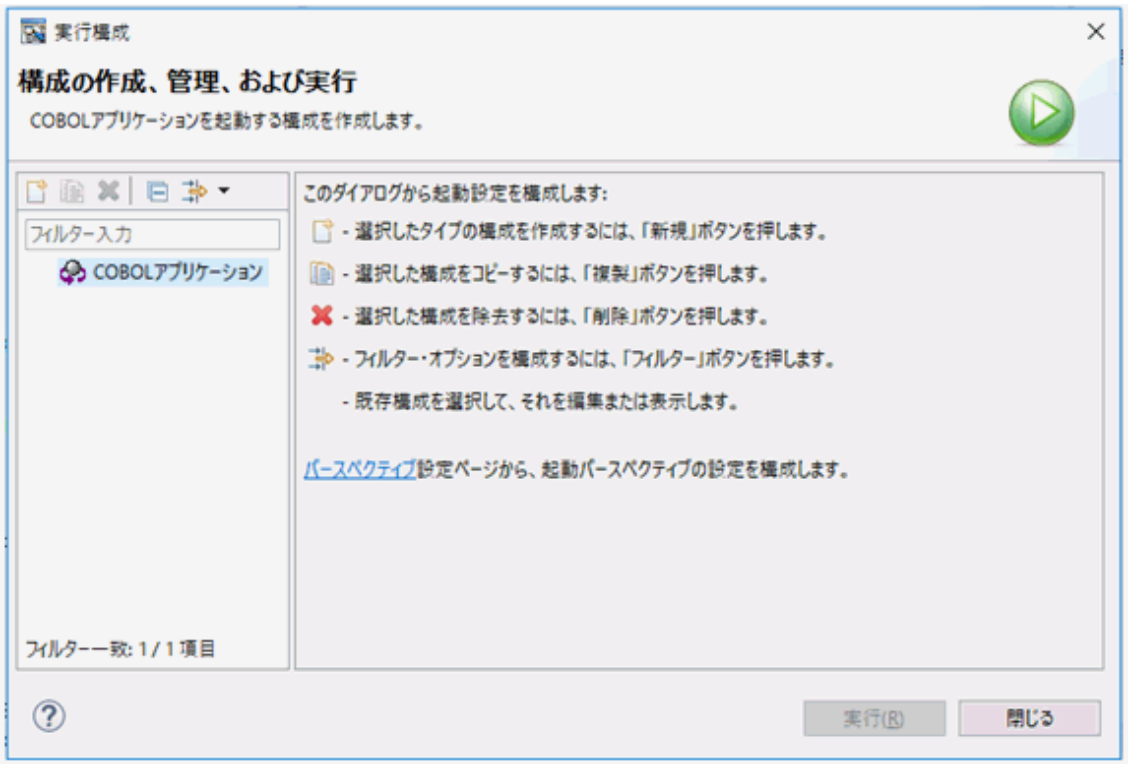
プログラムの実行

このサンプルプログラムは、動的リンク構造で実行可能ファイルを作成しています。副プログラムのダイナミックリンクライブラリ(以降はDLLといいます)はシステムのダイナミックリンカによってローディングされるため、実行可能ファイル(.exe)と同じフォルダーにDLLが存在しない場合、環境変数PATHにDLLの格納フォルダーを追加する必要があります。

NetCOBOL Studioで環境変数情報を設定する方法を説明します。

1. ［依存］ビューからSample05プロジェクトを選択し、NetCOBOL Studioのメニューバーから［実行(R)］>［実行構成(N)］を選択します。

   → ［実行構成］ダイアログボックスが表示されます。

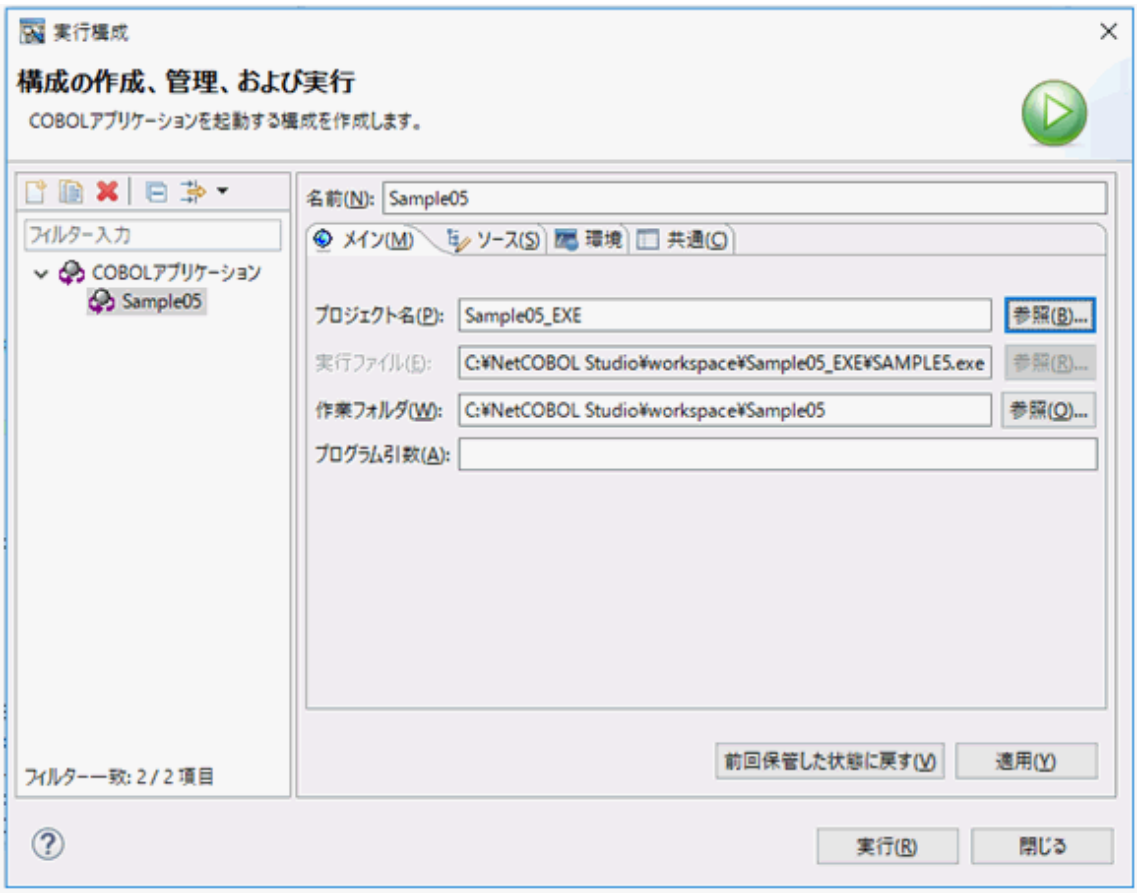


2. 左ペインから［COBOLアプリケーション］を選択し、［新規］ボタン( )をクリックします。

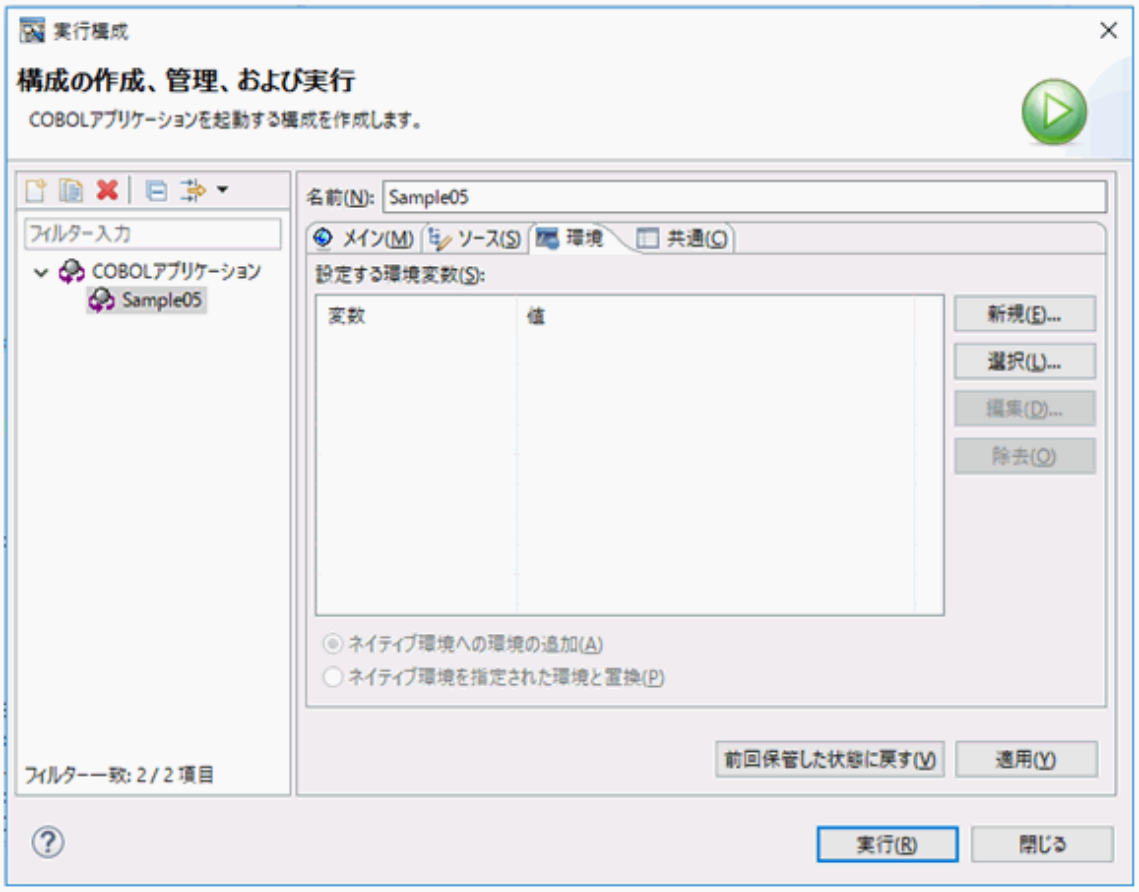
   → 右ペインの［名前］に"Sample05"が表示され、実行時の構成情報が表示されます。

3. ［プロジェクト名］は、［参照］ボタンをクリックして、表示されたプロジェクト一覧から"Sample05_EXE"を選択します。

→ [実行ファイル名]に実行可能ファイル名(.exe)が表示されます。



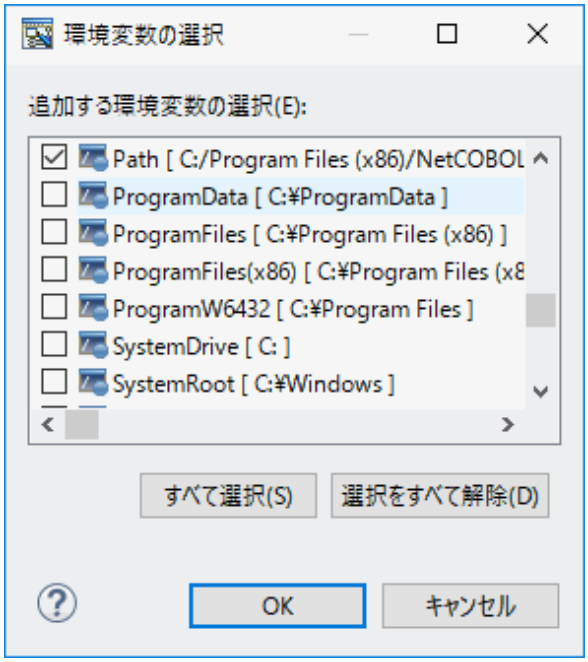
4. 右ペインから[環境]タブを選択します。



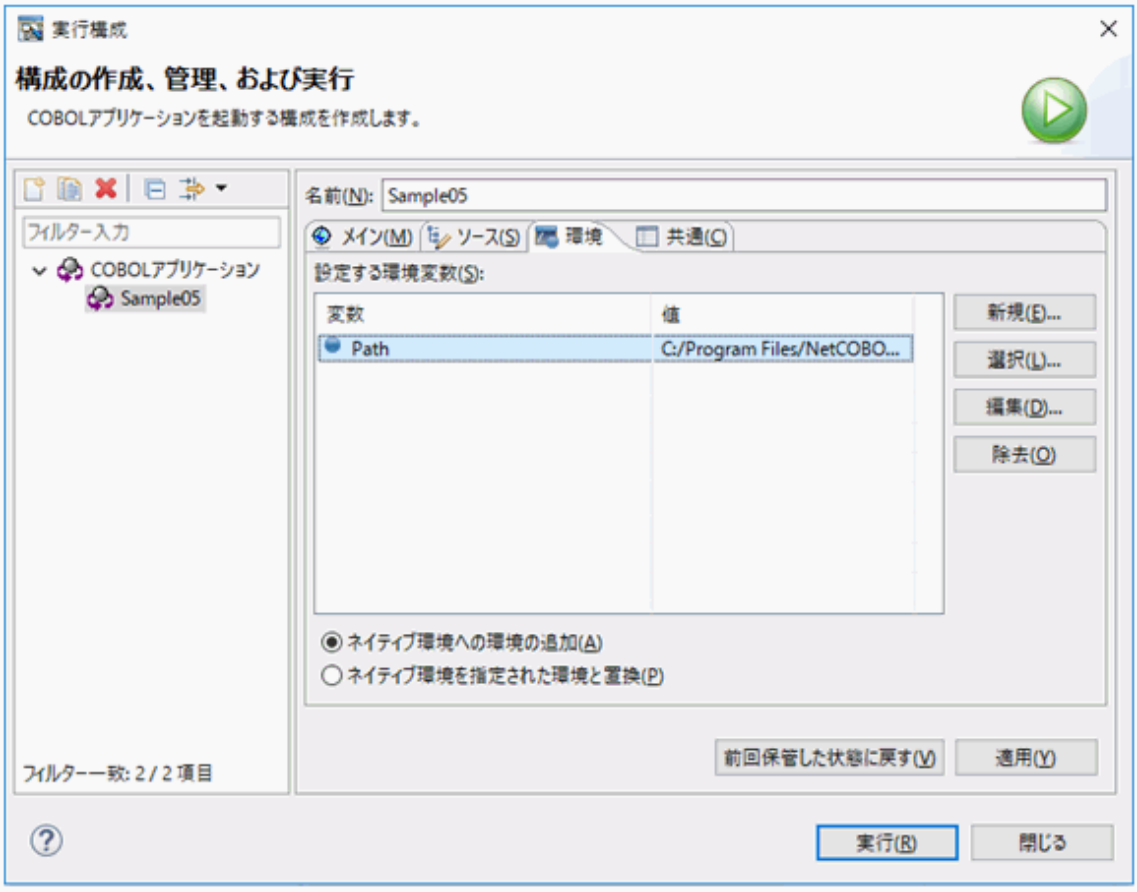
5. ここでは、環境変数PathにINSATSU.dllの格納フォルダーを追加します。まず、[選択]ボタンをクリックします。

   → [環境変数の選択]ダイアログボックスが表示されます。

6. “Path”をチェックし、[OK]ボタンをクリックします。



→ [設定する環境変数]に“Path”が追加されます。



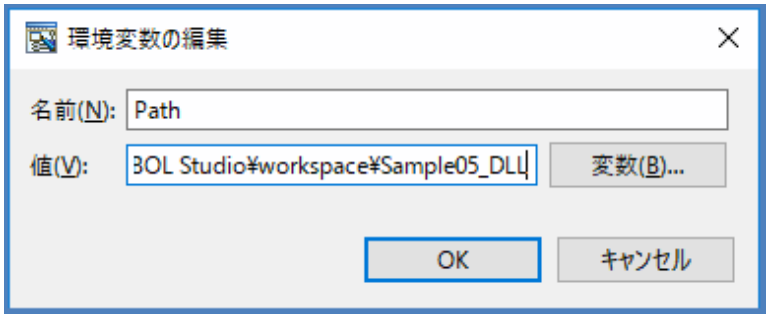
7. [ネイティブ環境への環境の追加]がチェックされていることを確認します。

注意

[ネイティブ環境を指定された環境と置換]をチェックして以降の手順を進めた場合、アプリケーションが正しく実行できません。必ず、[ネイティブ環境への環境の追加]をチェックしてください。

8. [設定する環境変数]タブから“Path”を選択し、[編集]ボタンをクリックします。

   → [環境変数の編集]ダイアログボックスが表示されます。



   [値]にSample05_DLLプロジェクトの格納フォルダーを追加し、[OK]ボタンをクリックします。

9. [環境]タブの[適用]ボタンをクリックします。これで、実行時の環境設定は完了です。
10. [実行]ボタンをクリックします。

    → Sample5.exeが実行されます。

実行結果

“作業ファイル(Sample5.TMP)を作成します”というメッセージが表示されます。内容を確認したら、[OK]ボタンをクリックしてメッセージボックスを閉じてください。
プログラムの実行が終了すると、マスタファイルの内容が“通常使うプリンター”として設定されている印刷装置に出力されます。

6.6.2 MAKEファイルを利用する

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample05>nmake
```

翻訳およびリンク終了後、Sample5.exeとINSATSU.dllが作成されていることを確認してください。

実行環境情報の設定

NetCOBOL Studioを利用する場合と同じです。

プログラムの実行

INSATSU.dllファイルが、カレントフォルダーまたは環境変数PATHに設定したフォルダーにあることを確認してください。コマンドプロンプトまたはエクスプローラからSample5.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.7 コマンド行引数の受取り方(Sample06)

ここでは、本製品で提供するサンプルプログラム-Sample06-について説明します。

Sample06では、コマンド行引数の操作機能を使って、COBOLプログラムの実行時に指定された引数を受け取るプログラムの例を示します。コマンド行引数の操作機能の使い方は、“NetCOBOL ユーザーズガイド”の“コマンド行引数の取出し”を参照してください。また、Sample06では、内部プログラムの呼出しも行います。

概要

開始年月日から終了年月日までの日数を求めます。開始年月日および終了年月日は、コマンドの引数として次の形式で指定されます。

```
コマンド名　開始年月日　終了年月日
```

開始年月日および終了年月日は、1900年1月1日～2172年12月31日までの日付をYYYYMMDDで指定します。西暦については4桁で指定します。

提供プログラム

- Sample6.cob(COBOLソースプログラム)
- Makefile(メイクファイル)
- COBOL85.CBR(実行用の初期化ファイル)

使用しているCOBOLの機能

- コマンド行引数の取出し
- コンソール型のアプリケーションの作成
- 内部プログラム

使用しているCOBOLの文

- ACCEPT文
- CALL文
- COMPUTE文
- COPY文
- DISPLAY文
- DIVIDE文
- EXIT文
- GO TO文
- IF文
- MOVE文
- PERFORM文

6.7.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

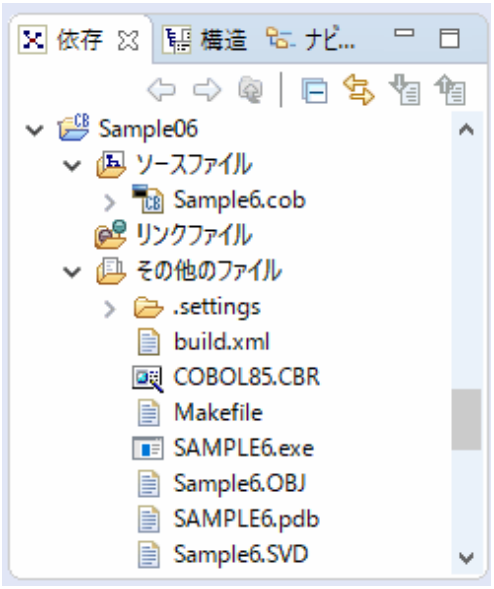
   “ワークスペースを準備する”

2. [依存]ビューを確認し、Sample06プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample06プロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. [その他のファイル]にSample6.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、Sample6.exeが作成されます。

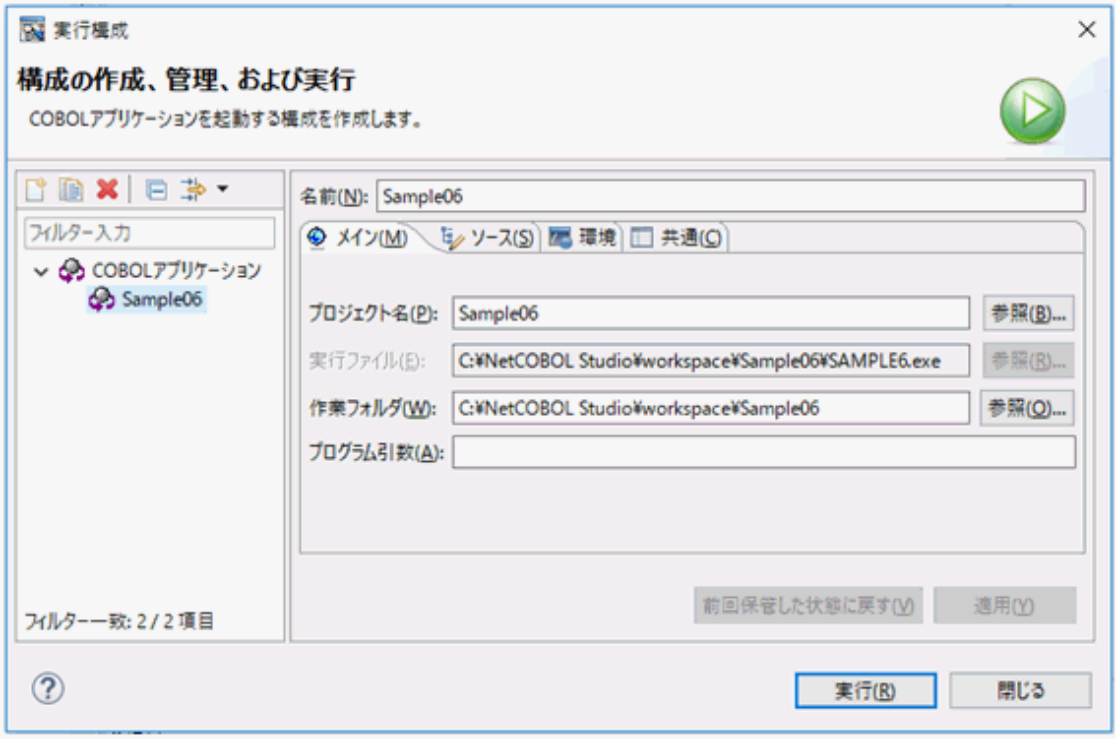
プログラムの実行

1. [依存]ビューからSample06プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行構成(N)]を選択します。

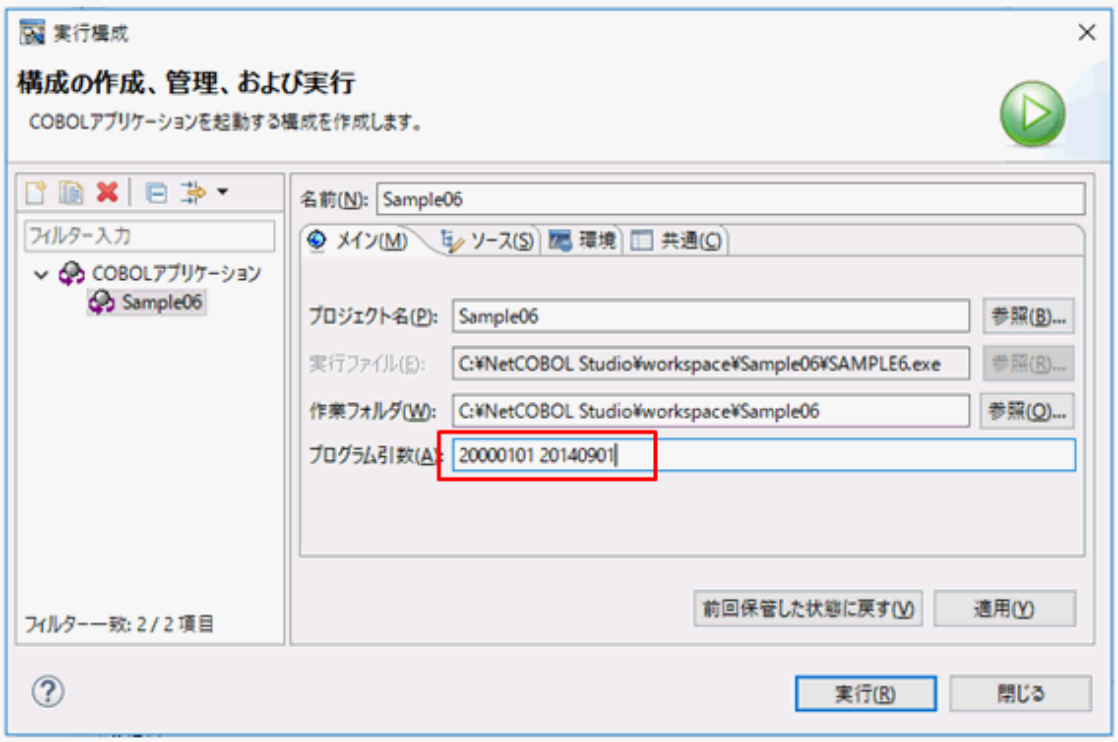
   → [実行構成]ダイアログボックスが表示されます。

2. 左ペインから[COBOLアプリケーション]を選択し、[新規]ボタンをクリックします。

   → 右ペインの[名前]に"Sample06"が表示され、実行時の構成情報が表示されます。



3. [メイン]タブを選択し、[プログラム引数]に開始年月日および終了年月日を入力します。



プログラム引数の入力例:

```
20000101 20140901
```

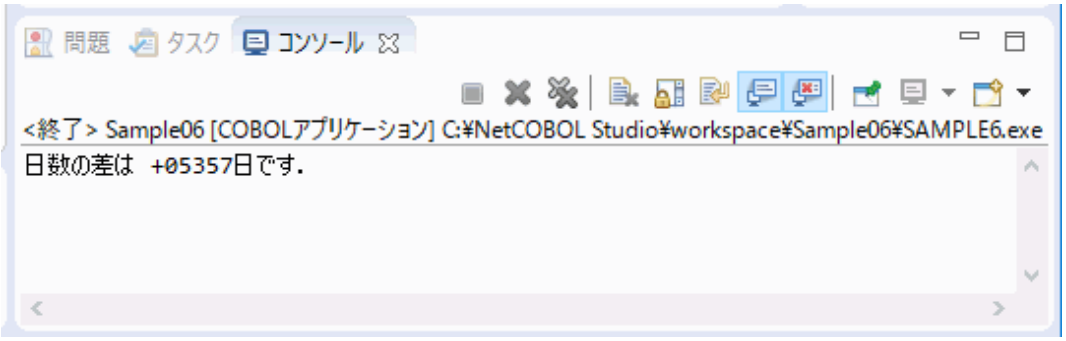
4. [適用]ボタンをクリックし、続けて[実行]ボタンをクリックします。

   → Sample6.exeが実行されます。

実行結果

このSampleの場合、DISPLAY文の出力先はシステムコンソールです。

次のように、2000年1月1日から2014年9月1日までの日数が表示されます。



参考

DISPLAY文の出力先をCOBOLコンソールにしたい場合は、“NetCOBOL Studio ユーザーズガイド”の“ターゲットの定義”を参照してください。

6.7.2 MAKEファイルを利用する

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample06>nmake
```

翻訳およびリンク終了後、Sample6.exeが作成されていることを確認してください。

プログラムの実行

1. Sample6.exeの指定に続けて開始年月日および終了年月日を入力します。

```
C:¥COBOL¥Samples¥COBOL¥Sample06>Sample6.exe 20000101 20140901
```

実行結果

コンソール型のアプリケーションでは、DISPLAY文による出力はCOBOLのコンソールウィンドウではなく、システムのコンソール(コマンドプロンプト)に出力されます。

次のように、2000年1月1日から2014年9月1日までの日数が表示されます。

```
C:¥COBOL¥Samples¥COBOL¥Sample06>Sample6.exe 20000101 20140901
日数の差は   +5357日です.

C:¥COBOL¥Samples¥COBOL¥Sample06>
```

6.8 環境変数の操作(Sample07)

ここでは、本製品で提供するサンプルプログラム-Sample07-について説明します。

Sample07では、環境変数の操作機能を使って、COBOLプログラム実行中に環境変数の値を変更するプログラムの例を示します。環境変数の操作機能の使い方は、“NetCOBOL ユーザーズガイド”の“環境変数の操作機能”を参照してください。

概要

商品コード、商品名および単価が格納されているマスタファイル(Sample02で作成した索引ファイル)中のデータを、商品コードの値によって2つのマスタファイルに分割します。分割方法および新規に作成する2つのマスタファイルのファイル名を以下に示します。

商品コードの値	ファイル名
先頭が"0"	マスタファイル名.a
先頭が"0"以外	マスタファイル名.b

提供プログラム

- Sample7.cob(COBOLソースプログラム)
- Makefile(メイクファイル)
- COBOL85.CBR(実行用の初期化ファイル)

注意

Syohinm.cbl(登録集原文)は、Sample02の提供ファイルを使用します。

使用しているCOBOLの機能

- 環境変数の操作機能

- 索引ファイル

使用しているCOBOLの文

- ACCEPT文
- CLOSE文
- DISPLAY文
- EXIT文
- GO TO文
- IF文
- OPEN文
- READ文
- STRING文
- WRITE文

プログラムを実行する前に

Sample02で作成されるマスタファイルを使用します。

“6.3 行順ファイルと索引ファイルの操作(Sample02)”を実行しておきます。

6.8.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

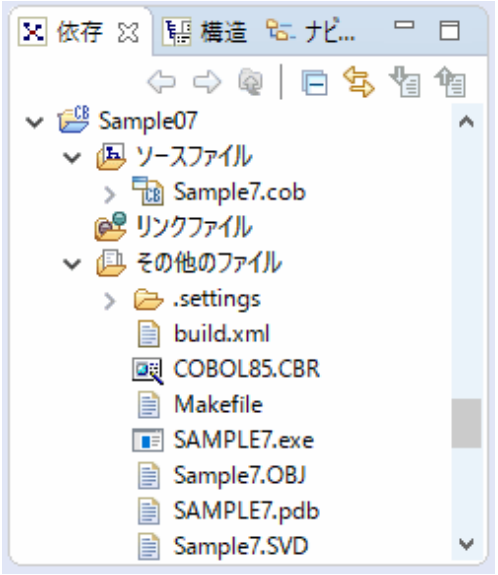
   “ワークスペースを準備する”

2. [依存]ビューを確認し、Sample07プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. ［依存］ビューからSample07プロジェクトを選択し、以下の構成になっていることを確認します。

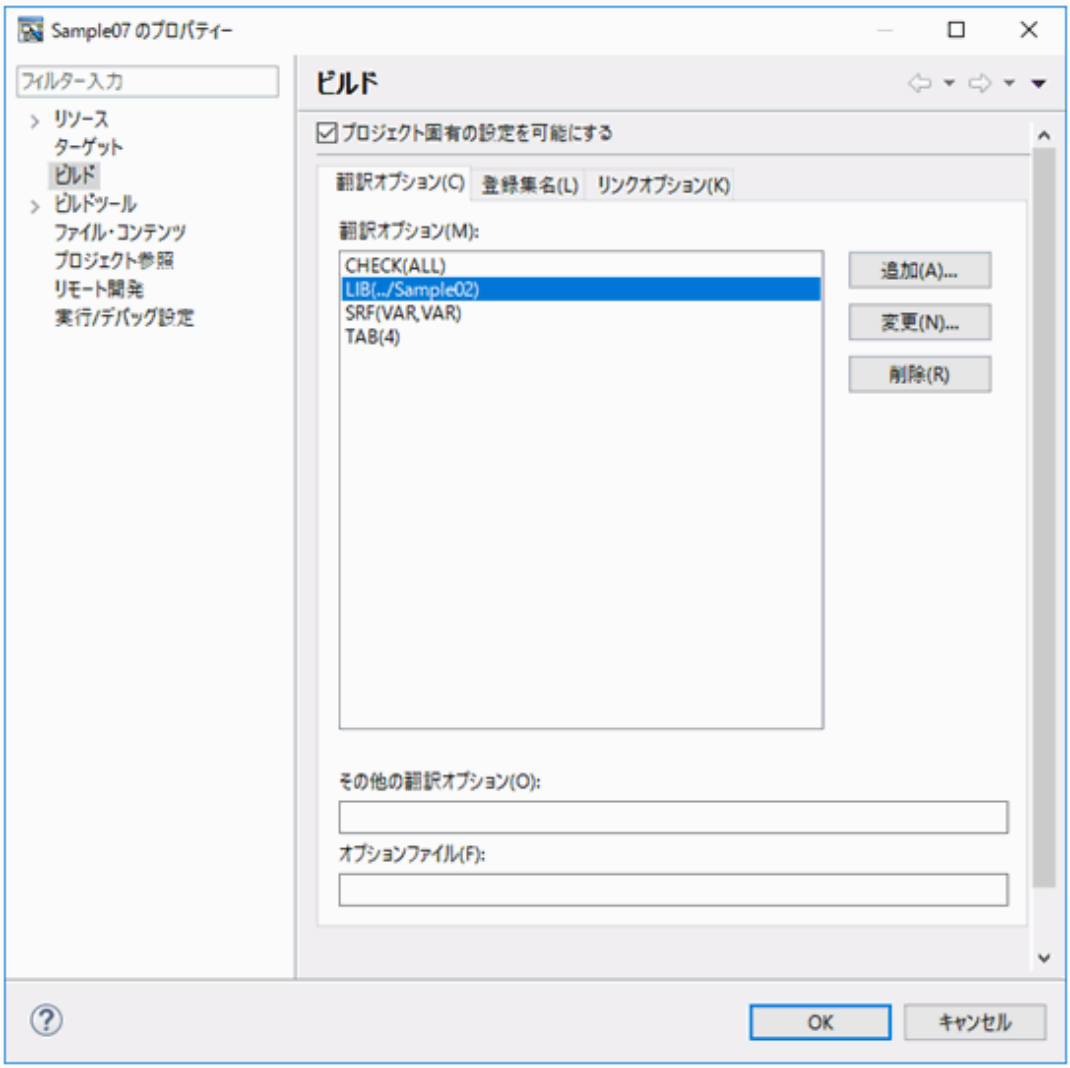


自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、［その他のファイル］には、ビルド後に生成されるファイル（.exeや.objなど）が表示されます。既定では自動ビルドに設定されています。

4. Sample07プロジェクトの翻訳オプションLIBに、Syohinm.cblが格納されたフォルダー（Sample02のフォルダー）が指定されていることを確認します。
翻訳オプションの確認は、NetCOBOL Studioの［依存］ビューからSample07プロジェクトを選択し、コンテキストメニューから［プロパティ］を選択します。

→ ［プロパティ］ダイアログボックスが表示されます。

5. 左ペインから[ビルド]を選択すると、[ビルド]ページが表示されます。[翻訳オプション]タブを選択して、設定オプションの内容を確認します。



ここでは、翻訳オプションLIBに、Syohinm.cblの格納フォルダー(Sample02プロジェクトのフォルダー:"..¥Sample02")が指定されていることを確認して、[OK]ボタンをクリックします。

NetCOBOL Studioからビルドする場合、カレントフォルダーはプロジェクトフォルダーです。

参考

翻訳オプションを設定するための詳細な手順は、“NetCOBOL Studio ユーザーズガイド”の“翻訳オプションの設定”を参照してください。

6. [その他のファイル]にSample7.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、Sample7.exeが作成されます。

実行環境情報の設定

1. [実行環境設定]ツールを起動します。

   [実行環境設定]ツールを起動する方法には以下があります。

   ー [スタート]> お使いのNetCOBOL製品 > [実行環境設定ツール]を選択します。

ー ［NetCOBOLコマンドプロンプト］から、実行環境設定ツール（COBENVUT.exe）を起動します。

2. ［実行環境設定ツール］のメニューバーから、［ファイル］＞［開く］を選択します。

   → ［実行用の初期化ファイルの指定］ウィンドウが表示されます。

3. 実行可能プログラム（Sample7.exe）が存在するフォルダーのCOBOL85.CBRを選択し、［開く］ボタンをクリックします。
   NetCOBOL Studioからビルドした場合、実行可能プログラムは、プロジェクトフォルダーに作成されています。

   → 実行用の初期化ファイルの内容が表示されます。

4. [共通]タブを選択し、以下の設定を確認します。

   ー ファイル識別名INFILEに、Sample02で作成したマスタファイル（MASTER）が指定されている。

   ```
   INFILE=..¥Sample02¥MASTER
   ```

   相対パスでファイルを指定する場合、カレントフォルダーからの相対パスになります。

   NetCOBOL Studioのメニューバーから［実行(R)］＞［実行(S)］＞［COBOLアプリケーション］を選択して実行する場合、カレントフォルダーはプロジェクトフォルダーです。

5. ［適用］ボタンをクリックします。

   → 設定した内容が実行用の初期化ファイルに保存されます。

6. ［ファイル］メニューの［終了］を選択し、実行環境設定ツールを終了します。

プログラムの実行

［依存］ビューからSample07プロジェクトを選択し、NetCOBOL Studioのメニューバーから［実行(R)］＞［実行(S)］＞［COBOLアプリケーション］を選択します。

実行結果

マスタファイルと同じフォルダー（Sample02のフォルダー）に次の2つのファイルが作成されます。

- MASTER.a（商品コードの先頭が“0”の商品のデータを格納したマスタファイル）
- MASTER.b（商品コードの先頭が“0”以外の商品のデータを格納したマスタファイル）

参考

作成したマスタファイル（MASTER.aおよびMASTER.b）の内容は、Sample02で作成したマスタファイルと同様に、“例題5のサンプルプログラム”を使って内容を確認することができます。この場合、実行環境情報の設定で、INFILEに作成したマスタファイルを指定する必要があります。

6.8.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample07>nmake
```

翻訳およびリンク終了後、Sample7.exeが作成されていることを確認してください

実行環境情報の設定

NetCOBOL Studioを利用する場合と同じです。

プログラムの実行

コマンドプロンプトまたはエクスプローラからSample7.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.9 印刷ファイルを使ったプログラム(Sample08)

ここでは、本製品で提供するサンプルプログラム-Sample08-について説明します。

Sample08では、印刷ファイルを使って、コンソールウィンドウから入力したデータを印刷装置に出力するプログラムの例を示します。印刷ファイルの使い方の詳細は、“NetCOBOL ユーザーズガイド”の“行単位のデータを印刷する方法”を参照してください。

概要

コンソールウィンドウから英数字のデータ40文字を入力し、印刷装置に出力します。

提供プログラム

- Sample8.cob（COBOLソースプログラム）
- Makefile（メイクファイル）
- COBOL85.CBR（実行用の初期化ファイル）

使用しているCOBOLの機能

- 印刷ファイル
- 小入出力機能(コンソールウィンドウ)

使用しているCOBOLの文

- ACCEPT文
- CLOSE文
- EXIT文
- GO TO文
- IF文
- OPEN文
- WRITE文

6.9.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

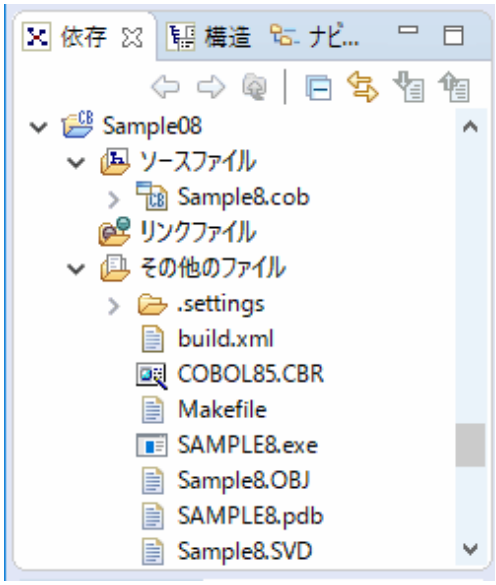
   “ワークスペースを準備する”

2. ［依存］ビューを確認し、Sample08プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample08プロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. [その他のファイル]にSample8.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、Sample8.exeが作成されます。

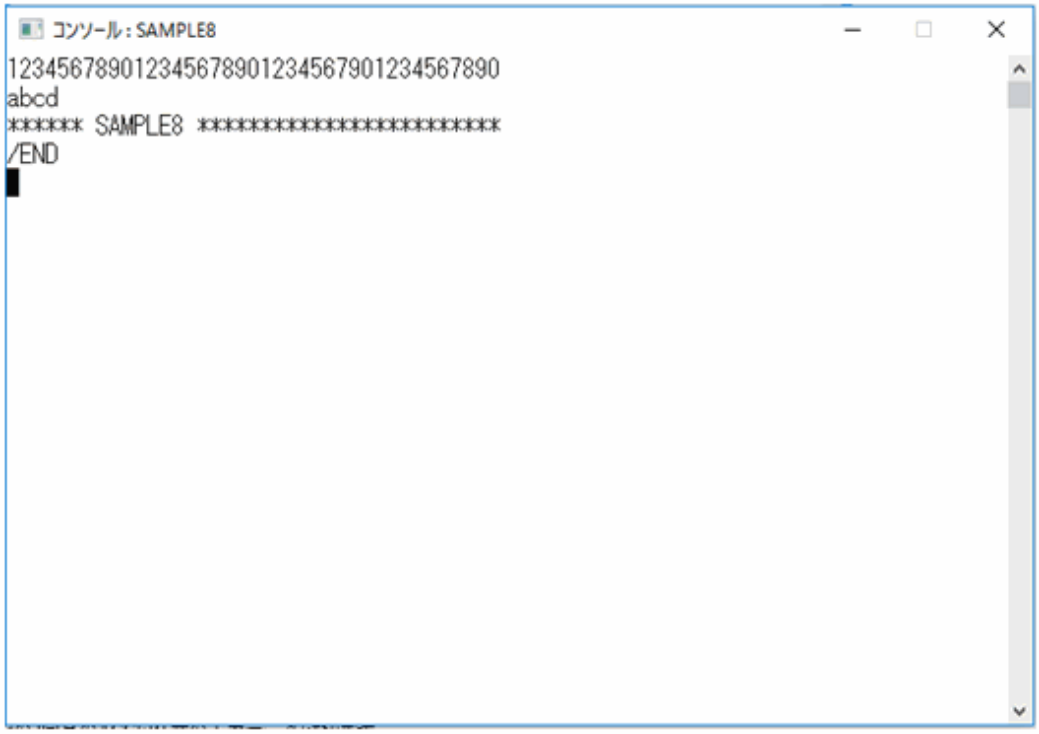
プログラムの実行

[依存]ビューからSample08プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択します。

実行結果

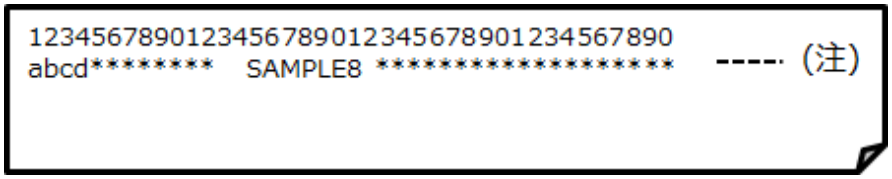
1. コンソールウィンドウが表示されます。

2. コンソールウィンドウから、印刷するデータを入力します。1回のデータの入力は40文字以内です。たとえば、以下のようにデータを入力します。



3. データの入力を終了する場合、“/END”に続けて空白を36文字入力し、ENTERキーを押します。

   → プログラムが終了すると、入力したデータがプリンターに印刷されます。



(注)コンソールウィンドウでの2回目の入力が40文字未満なので、2回目の入力データと3回目の入力データを合わせたデータが、プログラムでの2回目のACCEPT文の入力データとなります。

6.9.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample08>nmake
```

翻訳およびリンク終了後、Sample8.exeが作成されていることを確認してください

プログラムの実行

コマンドプロンプトまたはエクスプローラからSample8.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.10 印刷ファイルを使ったプログラム（応用編）(Sample09)

ここでは、本製品で提供するサンプルプログラム-Sample09-について説明します。

Sample09では、FORMAT句なし印刷ファイルを使って、I制御レコードを使用したページ形式の設定/変更と、CHARACTER TYPE句やPRINTING POSITION句を使用して印字したい文字の修飾および配置(行/桁)を意識して印刷装置に出力するプログラムの例を示します。FORMAT句なし印刷ファイルの使い方の詳細は、“NetCOBOL ユーザーズガイド”の“行単位のデータを印刷する方法”および“フォームオーバレイおよびFCBを使う方法”を参照してください。

概要

FORMAT句なし印刷ファイルを使用して帳票印刷を行う場合、主に利用される機能を想定し、以下の項目について印刷デモを行います。

FCBを使用した6LPI、8LPIでの帳票印刷

FCBを利用した任意の行間隔(6/8LPI)で帳票印刷を行うことを想定し、I制御レコードによるFCB(LPI)の切り替えを行います。ソースプログラムには、CHARACTER TYPE句やPRINTING POSITION句を記述して、行間隔(LPI)や文字間隔(CPI)などの行・桁を意識して帳票の体裁を整えます。

以下の帳票印刷を行います。

- A4用紙を横向きに使用し、1ページすべての行間隔を6LPIとした場合の帳票をイメージし、6LPI/10CPIフォーマットのスペーシングチャート形式のフォームオーバレイと重畳印刷します。
- A4用紙を横向きに使用し、1ページすべての行間隔を8LPIとした場合の帳票をイメージし、8LPI/10CPIフォーマットのスペーシングチャート形式のフォームオーバレイと重畳印刷します。

CHARACTER TYPE句で指定する各種文字属性での印刷

I制御レコードを使用し、用紙サイズをA4/横向きからB4/横向きに変更し、これにあわせてFCBもA4/横向き用からB4/横向き用に変更します。

以下の各種文字属性の印字サンプルを印刷装置に出力します。

1. 文字サイズ

   1文字ずつ3ポ、7.2ポ、9ポ、12ポ、18ポ、24ポ、36ポ、50ポ、72ポ、100ポ、200ポ、300ポの文字サイズを印字します。

   参考

   ここでは、文字ピッチ指定を省略することにより、文字サイズに合わせた最適な文字ピッチをCOBOLランタイムシステムに自動算出させます。

2. 文字ピッチ

   文字ピッチ1CPIで1文字、2CPIで2文字、3CPIで3文字、5CPIで5文字、6CPIで6文字、7.5CPIで15文字、20CPIで20文字、24CPIで24文字指定します。

   参考

   ここでは、文字サイズ指定を省略することにより、文字ピッチに合わせた最適な文字サイズをCOBOLランタイムシステムに自動算出させます。

3. 文字書体

   ゴシック、ゴシック半角(文字形態半角)、明朝、明朝半角(文字形態半角)を10文字ずつ2回繰り返し印字します。

   参考

   ここで指定した書体名は、以下の実行環境情報に関連付けられています。

```
@PrinterFontName=(FONT-NAME1,FONT-NAME2)
```

この指定により、“MINCHOU”、“MINCHOU-HANKAKU”を指定したデータ項目は、“FONT-NAME1”に指定されたフォントで印字され、“GOTHIC”、“GOTHIC-HANKAKU”を指定したデータ項目は、“FONT-NAME2”に指定されたフォントで印字されます。

なお、このサンプルプログラムでは、実行用の初期化ファイル(COBOL85.CBR)に“@PrinterFontName=(MS 明朝,MS ゴシック)”を指定しています。

4. 文字回転

   縦書き(反時計回りに90度回転)、横書きを10文字ずつ繰り返し印字します。

5. 文字形態

   全角、半角、全角平体、半角平体、全角長体、半角長体、全角倍角、半各倍角の文字形態指定を9文字ずつ印刷します。

6. 上記5つの文字属性を組み合わせた印刷を行います。

提供プログラム

- Sample9.cob（COBOLソースプログラム）
- KOL5A4L6.OVD（フォームオーバレイパターン）
- KOL5A4L8.OVD（フォームオーバレイパターン）
- KOL5B4OV.OVD（フォームオーバレイパターン）
- COBOL85.CBR（実行用の初期化ファイル）
- Makefile（メイクファイル）

使用しているCOBOLの機能

- 印刷ファイル
- 小入出力機能(コンソールウィンドウ)

使用しているCOBOLの文

- ADD文
- CLOSE文
- DISPLAY文
- IF文
- MOVE文
- OPEN文
- PERFORM文
- STOP文
- WRITE文

6.10.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

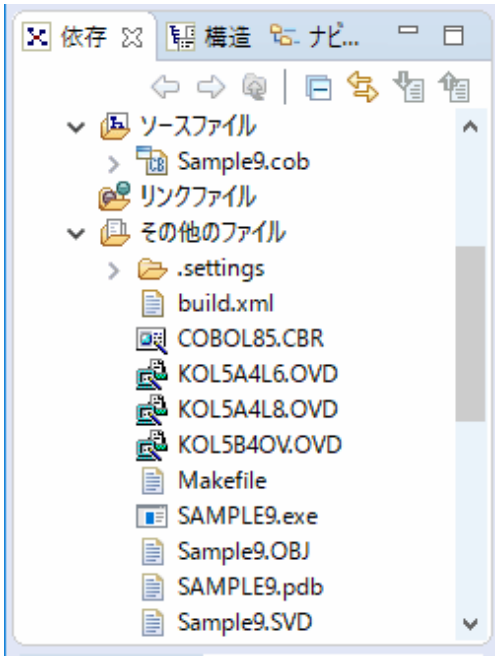
   “ワークスペースを準備する”

2. [依存]ビューを確認し、Sample09プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

参考

"サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする"

3. [依存]ビューからSample09プロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. [その他のファイル]にSample9.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、Sample9.exeが作成されます。

実行環境情報の設定

1. 実行環境設定]ツールを起動します。

   [実行環境設定]ツールを起動する方法には以下があります。

   - [スタート] > お使いのNetCOBOL製品 > [実行環境設定ツール]を選択します。
   - [NetCOBOLコマンドプロンプト]から、実行環境設定ツール(COBENVUT.exe)を起動します。

2. [実行環境設定ツール]のメニューバーから、[ファイル] > [開く]を選択します。

   → [実行用の初期化ファイルの指定]ウィンドウが表示されます。

3. 実行可能プログラム(Sample9.exe)が存在するフォルダーのCOBOL85.CBRを選択し、[開く]ボタンをクリックします。
   NetCOBOL Studioからビルドした場合、実行可能プログラムは、プロジェクトフォルダーに作成されています。

   → 実行用の初期化ファイルの内容が表示されます。

4. Sample09で必要な実行環境情報は、あらかじめCOBOL85.CBRに設定されています。以下の実行環境情報だけを[共通]タブから追加してください。

   ー 環境変数情報FOVLDIR(フォームオーバレイパターンのフォルダーの指定)に、実行可能プログラム(Sample9.exe)が存在するフォルダーを指定します。

   ```
   FOVLDIR=.¥
   ```

   相対パスでファイルを指定する場合、カレントフォルダーからの相対パスになります。
   NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択して実行する場合、カレントフォルダーはプロジェクトフォルダーです。

5. [適用]ボタンをクリックします。

   → 設定した内容が実行用の初期化ファイルに保存されます。

6. [ファイル]メニューの[終了]を選択し、実行環境設定ツールを終了します。

プログラムの実行

[依存]ビューからSample09プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択します。

実行結果

実行は、特に応答・操作する必要はなく自動的に終了します。実行が終了すると、サンプル帳票が、“通常使うプリンター”として設定されている印刷装置に出力されます。

6.10.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample09>nmake
```

翻訳およびリンク終了後、Sample9.exeが作成されていることを確認してください

実行環境情報の設定

NetCOBOL Studioを利用する場合と同じです。

プログラムの実行

コマンドプロンプトまたはエクスプローラからSample9.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.11 FORMAT句付き印刷ファイルを使ったプログラム(Sample10)

ここでは、本製品で提供するサンプルプログラム-Sample10-について説明します。

Sample10では、FORMAT句付き印刷ファイルを使って、集計表を印刷装置に出力するプログラムの例を示します。FORMAT句付き印刷ファイルの使い方は、“NetCOBOL ユーザーズガイド”の“帳票定義体を使う印刷ファイルの使い方”を参照してください。なお、このプログラムを実行するには、MeFtが必要です。

概要

商品コード、商品名および単価が格納されているマスタファイル(Sample02で作成した索引ファイル)と受注日、数量および売上げ金額が格納されている売上げファイル(Sample03で作成した索引ファイル)を入力して、売上集計表を印刷装置に出力します。

提供プログラム

- Sample10.cob（COBOLソースプログラム）
- Syuukei.pmd（帳票定義体）
- Mefprc（プリンタ情報ファイル）
- Uriage.cbl（登録集原文）
- Makefile（メイクファイル）
- COBOL85.CBR（実行用の初期化ファイル）

Syohinm.cbl(登録集原文)は、Sample02で提供されたものを使用します。Uriage.cbl(登録集原文)は、Sample03で提供されたものを使用します。

使用しているCOBOLの機能

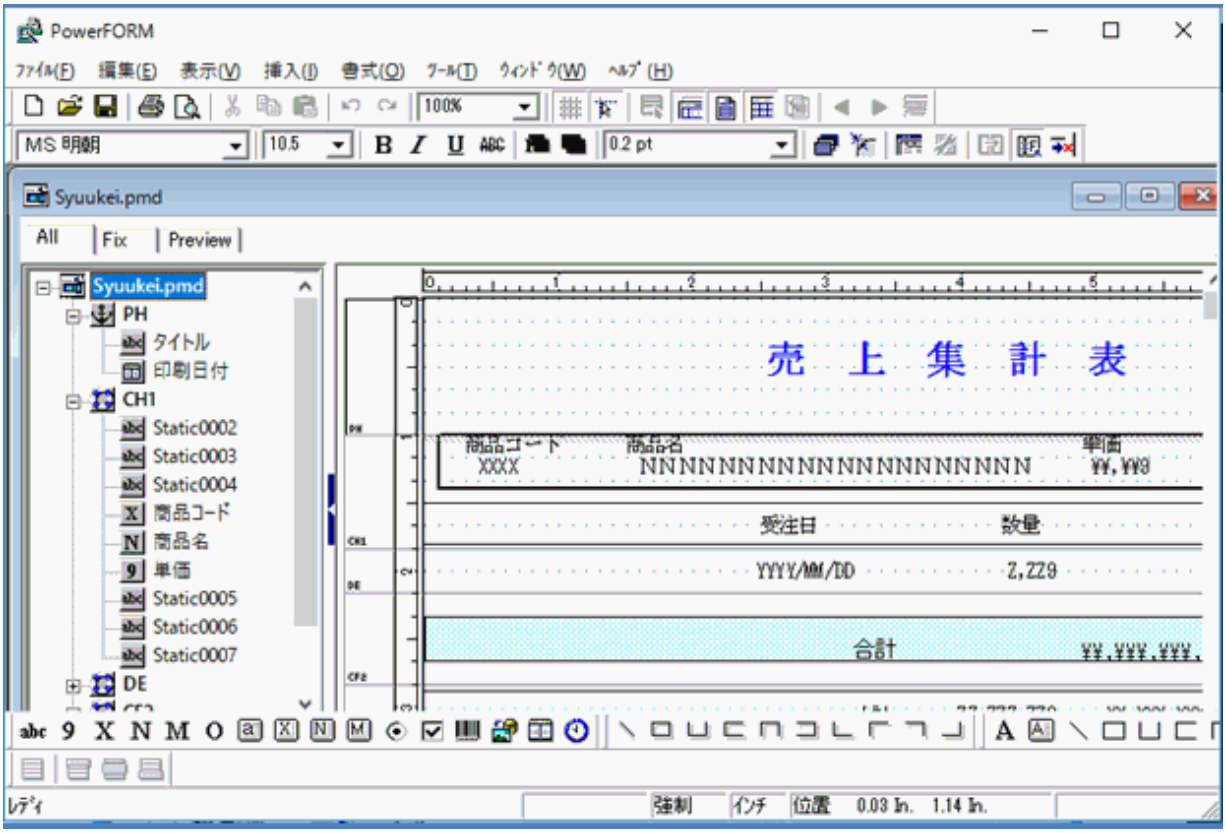
- FORMAT句付き印刷ファイル
- 索引ファイル(参照)
- 登録集の取込み
- 小入出力機能(メッセージボックス)

使用しているCOBOLの文

- OPEN文
- READ文
- WRITE文
- START文
- CLOSE文
- PERFORM文
- DISPLAY文
- IF文
- MOVE文
- SET文
- GO TO文
- EXIT文
- COPY文
- ADD文

使用している帳票定義体

売上集計表(Syuukei.pmd)



形式

集計表形式

用紙サイズ

A4

用紙方向

縦

行ピッチ

1/6インチ

パーティション

- PH(ページ頭書き)
  ［固定パーティション、印刷開始位置:0インチ(1行目)、縦幅:1インチ(6行)］
- CH1(制御頭書き)
  ［浮動パーティション、縦幅:0.83インチ(5行)］
- DE(明細)
  ［浮動パーティション、縦幅:0.33インチ(2行)］
- CF1(制御脚書き)
  ［浮動パーティション、縦幅:0.83インチ(5行)］
- CF2(制御脚書き)
  ［浮動パーティション、縦幅:0.67インチ(4行)］
- PF(ページ脚書き)
  ［固定パーティション、印刷開始位置:10.48インチ(63行目)、縦幅:0.49インチ(3行)］

プログラムを作成する上でのポイント

- PHおよびPFは、固定パーティション(固定の印刷位置情報を持っている)なので、パーティションを出力すると、必ず、パーティションに定義されている印刷開始位置に出力されます。
- CH1、DE、CF1およびCF2は、浮動パーティション(固定の印刷位置情報を持たない)なので、自由な位置に出力することができる反面、パーティション出力時に印刷位置を制御する必要があります。
- 各パーティションに定義された出力項目は、翻訳時にCOPY文で帳票定義体からレコードに展開されます。このとき、定義した出力項目の項目名がデータ名になります。

プログラムを実行する前に

- MeFtのセットアップを行い、使用できる状態にしておいてください。
- Sample02で作成されるマスタファイルを使用します。
  “6.3 行順ファイルと索引ファイルの操作(Sample02)”を実行しておきます。

6.11.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

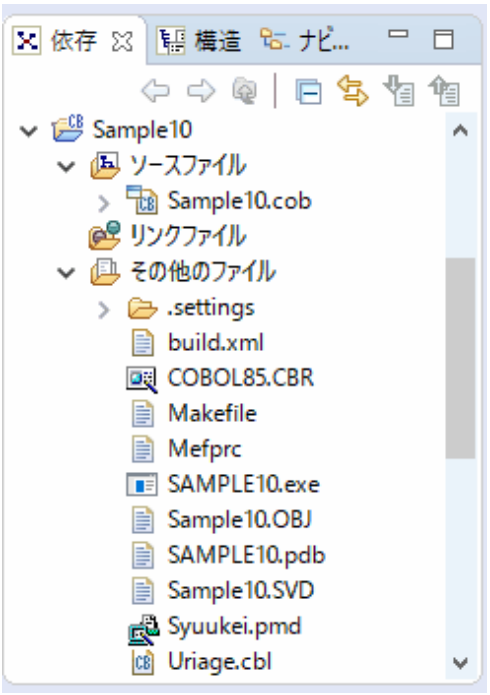
   “ワークスペースを準備する”

2. [依存]ビューを確認し、Sample10プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample10プロジェクトを選択し、以下の構成になっていることを確認します。

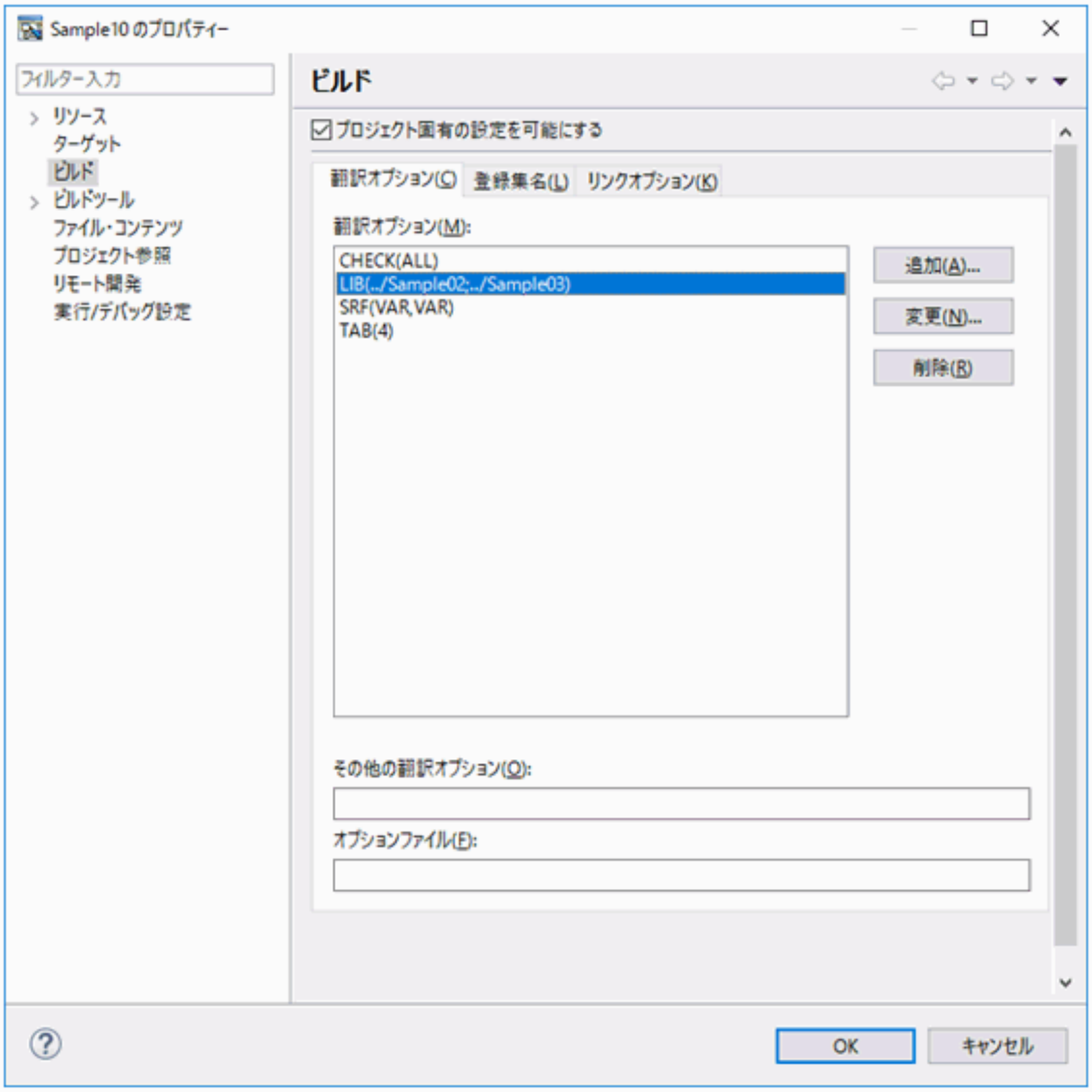


自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. Sample10プロジェクトの翻訳オプションLIBに、Syohinm.cblが格納されたフォルダー(Sample02のフォルダー)およびUriage.cblが格納されたフォルダー(Sample03のフォルダー)指定されていることを確認します。
   翻訳オプションの確認は、NetCOBOL Studioの[依存]ビューからSample10プロジェクトを選択し、コンテキストメニューから[プロパティ]を選択します。

   → [プロパティ]ダイアログボックスが表示されます。

5. 左ペインから[ビルド]を選択すると、[ビルド]ページが表示されます。[翻訳オプション]タブを選択して、設定オプションの内容を確認します。



ここでは、翻訳オプションLIBに、Syohinm.cblの格納フォルダー(Sample02プロジェクトのフォルダー:..¥Sample02")とUriage.cblの格納フォルダー(Sample03プロジェクトのフォルダー:..¥Sample03")が指定されていることを確認して、[OK]ボタンをクリックします。

NetCOBOL Studioからビルドする場合、カレントフォルダーはプロジェクトフォルダーです。

参考

翻訳オプションを設定するための詳細な手順は、“NetCOBOL Studio ユーザーズガイド”の“翻訳オプションの設定”を参照してください。

6. [その他のファイル]にSample10.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、Sample10.exeが作成されます。

プリンタ情報ファイルの設定

実行する前に、プリンタ情報ファイル(MEFPRC)の下線部(注)の情報を、エディタを使って変更しておきます。

```
MEDDIR  C:¥NetCOBOL Studio¥workspace¥Sample10
```

注:帳票定義体(Syuukei.pmd)を格納したフォルダーのパス名。帳票定義体(Syuukei.pmd)は、プロジェクトフォルダーに格納されます。

実行環境情報の設定

1. [実行環境設定]ツールを起動します。

   [実行環境設定]ツールを起動する方法には以下があります。

   - [スタート]> お使いのNetCOBOL製品 > [実行環境設定ツール]を選択します。
   - [NetCOBOLコマンドプロンプト]から、実行環境設定ツール(COBENVUT.exe)を起動します。

2. [実行環境設定ツール]のメニューバーから、[ファイル] > [開く]を選択します。

   → [実行用の初期化ファイルの指定]ウィンドウが表示されます。

3. 実行可能プログラム(Sample10.exe)が存在するフォルダーのCOBOL85.CBRを選択し、[開く]ボタンをクリックします。
   NetCOBOL Studioからビルドした場合、実行可能プログラムは、プロジェクトフォルダーに作成されています。

   → 実行用の初期化ファイルの内容が表示されます。

4. [共通]タブを選択し、以下の設定を確認します。

   - ファイル識別名DATAFILEに、Sample02で作成したマスタファイル(MASTER)が指定されている。
   - ファイル識別名PRTFILEに、プリンタ情報ファイル(MEFPRC)が指定されている。
   - ファイル識別名SALEFILEに、Sample03で作成した売上げファイル(SALES)が指定されている。

     ```
     DATAFILE=..¥Sample02¥MASTER
     PRTFILE=.¥MEFPRC
     SALEFLE=..¥Sample03¥SALES
     ```

   相対パスでファイルを指定する場合、カレントフォルダーはCOBOL85.CBRが格納されているフォルダーです。

   NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択して実行する場合、カレントフォルダーはプロジェクトフォルダーです。

5. [適用]ボタンをクリックします。

   → 設定した内容が実行用の初期化ファイルに保存されます。

6. [ファイル]メニューの[終了]を選択し、実行環境設定ツールを終了します。

参考

ファイル識別名のDATAFILE、SALEFILEおよびPRTFILEは、COBOLソースプログラムのASSIGN句に指定されているファイル参照子です。FORMAT句付き印刷ファイルを使用する場合、ファイル識別名PRTFILEにはプリンタ情報ファイルのパス名を指定します。

プログラムの実行

[依存]ビューからSample10プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択します。

実行結果

通常使うプリンターに設定されている印刷装置に集計表が出力されます。

6.11.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample10>nmake
```

翻訳およびリンク終了後、Sample10.exeが作成されていることを確認してください。

プリンタ情報ファイルの設定

NetCOBOL Studioを利用する場合と同じです。

実行環境情報の設定

NetCOBOL Studioを利用する場合と同じです。

プログラムの実行

コマンドプロンプトまたはエクスプローラからSample10.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.12 データベース機能を使ったプログラム(Sample11)

ここでは、本製品で提供するサンプルプログラム-Sample11-について説明します。

Sample11では、データベース(SQL)機能を使ってデータベースからデータを取り出しホスト変数に格納する例を示します。

データベースはサーバ上に存在し、クライアント側からこれにアクセスします。

データベースのアクセスは、ODBCドライバを経由して行います。ODBCドライバを使用するデータベースアクセスについては、“NetCOBOL ユーザーズガイド”の“リモートデータベースアクセス”を参照してください。

このプログラムを動作させるためには、以下の製品が必要です。

クライアント側

- ODBCドライバマネージャ
- ODBCドライバ
- ODBCドライバの必要とする製品

サーバ側

- データベース
- データベースにODBCでアクセスするために必要な製品

概要

サーバのデータベースにアクセスし、データベース上のテーブル“STOCK”に格納されている全データをディスプレイ装置に表示します。データをすべて参照し終わると、データベースとの接続を切断します。

提供プログラム

- Sample11.cob（COBOLソースプログラム）

- Makefile（メイクファイル）
- COBOL85.CBR（実行用の初期化ファイル）

使用しているCOBOLの機能

- リモートデータベースアクセス

使用しているCOBOLの文

- DISPLAY文
- GO TO文
- IF文
- PERFORM文
- 埋込みSQL文(埋込み例外宣言、CONNECT文、カーソル宣言、OPEN文、FETCH文、CLOSE文、ROLLBACK文、DISCONNECT文)

プログラムを実行する前に

- ODBCドライバを経由してサーバのデータベースへアクセスできる環境を構築しておいてください。
- デフォルトで接続するサーバを設定し、そのサーバのデータベース上に“STOCK”という名前でテーブルを作成しておいてください。

  “STOCK”テーブルは、以下の形式で作成してください。

GNO	GOODS	QOH	WHNO
2進整数 4桁	固定長文字 20バイト	2進整数 9桁	2進整数 4桁

“STOCK”テーブルに格納しておくデータは任意です。以下に例を示します。

GNO	GOODS	QOH	WHNO
110	TELEVISION	85	2
111	TELEVISION	90	2
123	REFRIGERATOR	60	1
124	REFRIGERATOR	75	1
137	RADIO	150	2
138	RADIO	200	2
140	CASSETTE DECK	120	2
141	CASSETTE DECK	80	2
200	AIR CONDITIONER	04	1
201	AIR CONDITIONER	15	1
212	TELEVISION	10	2
215	VIDEO	05	2
226	REFRIGERATOR	08	1
227	REFRIGERATOR	15	1
240	CASSETTE DECK	25	2
243	CASSETTE DECK	14	2

GNO	GOODS	QOH	WHNO
351	CASSETTE TAPE	2500	2
380	SHAVER	870	2
390	DRIER	540	2

- ODBC情報ファイル設定ツール(SQLODBCS.exe)を使用して、ODBC情報ファイル(ここではDBMSACS.INFとします)を作成してください。

6.12.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

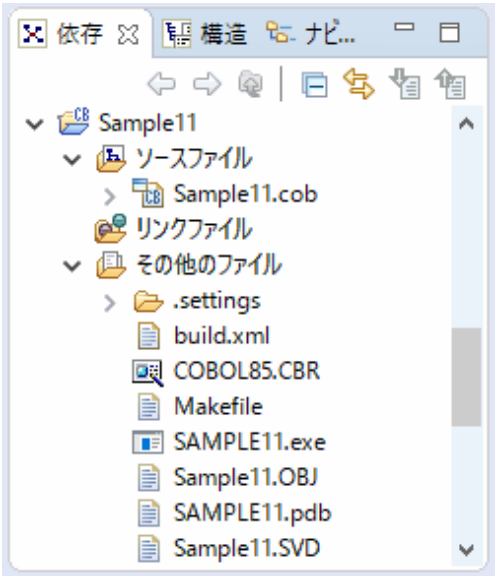
   “ワークスペースを準備する”

2. [依存]ビューを確認し、Sample11プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample11プロジェクトを選択し、以下の構成になっていることを確認します。

   自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. [その他のファイル]にSample11.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、Sample11.exeが作成されます。

実行環境情報の設定

1. [実行環境設定]ツールを起動します。

   [実行環境設定]ツールを起動する方法には以下があります。

   ー [スタート]> お使いのNetCOBOL製品 > [実行環境設定ツール]を選択します。

   ー [NetCOBOLコマンドプロンプト]から、実行環境設定ツール(COBENVUT.exe)を起動します。

2. [実行環境設定ツール]のメニューバーから、[ファイル] > [開く]を選択します。

   → [実行用の初期化ファイルの指定]ウィンドウが表示されます。

3. 実行可能プログラム(Sample11.exe)が存在するフォルダーのCOBOL85.CBRを選択し、[開く]ボタンをクリックします。
   NetCOBOL Studioからビルドした場合、実行可能プログラムは、プロジェクトフォルダーに作成されています。

   → 実行用の初期化ファイルの内容が表示されます。

4. [共通]タブを選択し、以下の設定を確認します。

   ー 環境変数情報@ODBC_Inf(ODBC情報ファイルの指定)に、ODBC情報ファイル名が指定されている。

   ```
   @ODBC_Inf=.¥DBMSACS.INF
   ```

   相対パスでファイルを指定する場合、カレントフォルダーからの相対パスになります。

   NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択して実行する場合、カレントフォルダーはプロジェクトフォルダーです。

5. [適用]ボタンをクリックします。

   → 設定した内容が実行用の初期化ファイルに保存されます。

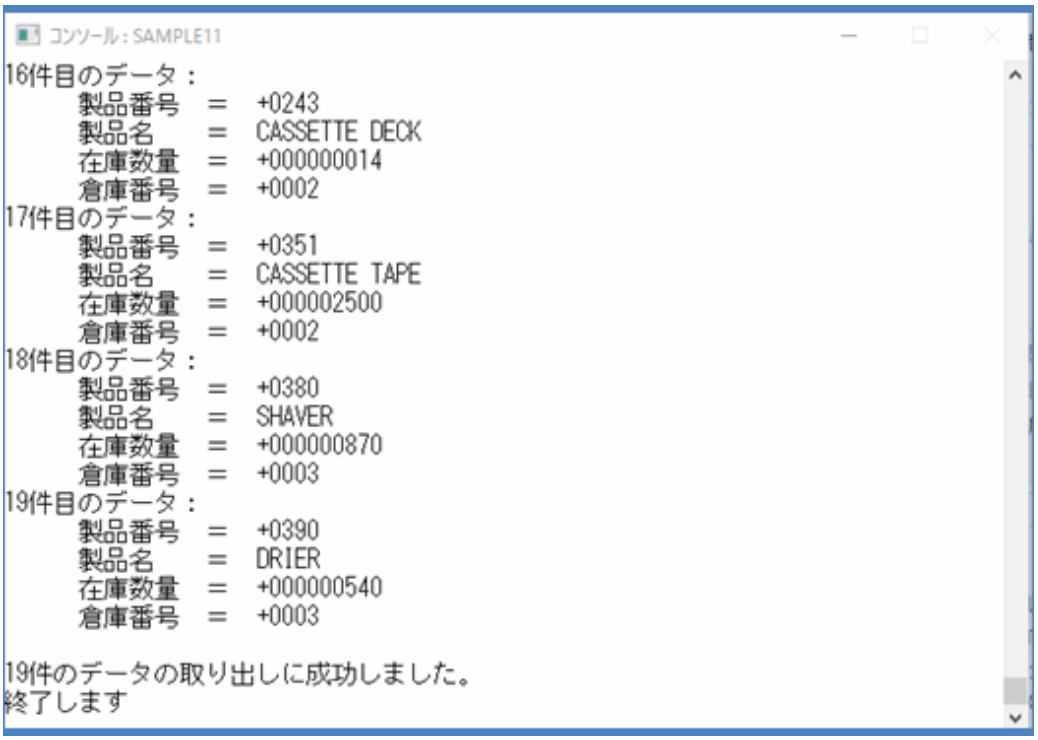
6. [ファイル]メニューの[終了]を選択し、実行環境設定ツールを終了します。

プログラムの実行

[依存]ビューからSample11プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択します。

実行結果

COBOLのコンソールウィンドウに、以下が表示されます。



6.12.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample11>nmake
```

翻訳およびリンク終了後、Sample11.exeが作成されていることを確認してください

実行環境情報の設定

NetCOBOL Studioを利用する場合と同じです。

プログラムの実行

コマンドプロンプトまたはエクスプローラからSample11.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.13 データベース機能を使ったプログラム（応用編）(Sample12)

ここでは、本製品で提供するサンプルプログラム-Sample12-について説明します。

Sample12では、データベース(SQL)機能のより進んだ使い方として、データベースの複数の行を1つの埋込みSQL文で操作する例を示します。

データベースはサーバ上に存在し、クライアント側からこれにアクセスします。

データベースのアクセスは、ODBCドライバを経由して行います。ODBCドライバを使用するデータベースアクセスについては、“NetCOBOL ユーザーズガイド”の“リモートデータベースアクセス”を参照してください。

このプログラムを動作させるためには、以下の製品が必要です。

クライアント側

- ODBCドライバマネージャ
- ODBCドライバ
- ODBCドライバの必要とする製品

サーバ側

- データベース
- データベースにODBCでアクセスするために必要な製品

概要

Sample11と同じデータベースにアクセスし、次の操作を行ってからデータベースとの接続を切断します。

- データベース全データの表示
- GOODSの値が“TELEVISION”である行のGNOの値の取り出し
- 取り出したGNOを持つ行のQOHの更新
- GOODSの値が“RADIO”、“SHAVER”、“DRIER”の行の削除
- データベースの全データの再表示

なお、出力結果は翻訳オプションSSOUTを使用して、一部をファイルに出力します。

提供プログラム

- Sample12.cob（COBOLソースプログラム）
- Makefile（メイクファイル）
- COBOL85.CBR（実行用の初期化ファイル）

使用しているCOBOLの機能

- リモートデータベースアクセス

使用しているCOBOLの文

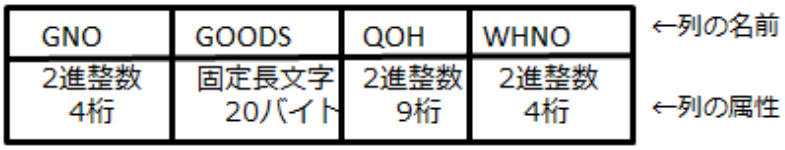
- CALL文
- DISPLAY文
- GO TO文
- IF文
- PERFORM文
- 埋込みSQL文 (複数行指定ホスト変数、表指定ホスト変数、埋込み例外宣言、CONNECT文、カーソル宣言、OPEN文、FETCH文、SELECT文、DELETE文、UPDATE文、CLOSE文、COMMIT文、ROLLBACK文、DISCONNECT文)

プログラムを実行する前に

- ODBCドライバを経由してサーバのデータベースへアクセスできる環境を構築しておいてください。

・ デフォルトで接続するサーバを設定し、そのサーバのデータベース上に“STOCK”という名前でテーブルを作成しておいてください。

“STOCK”テーブルは、以下の形式で作成してください。



“STOCK”テーブルに次のデータを格納しておいてください。

GNO	GOODS	QOH	WHNO
110	TELEVISION	85	2
111	TELEVISION	90	2
123	REFRIGERATOR	60	1
124	REFRIGERATOR	75	1
137	RADIO	150	2
138	RADIO	200	2
140	CASSETTE DECK	120	2
141	CASSETTE DECK	80	2
200	AIR CONDITIONER	04	1
201	AIR CONDITIONER	15	1
212	TELEVISION	10	2
215	VIDEO	05	2
226	REFRIGERATOR	08	1
227	REFRIGERATOR	15	1
240	CASSETTE DECK	25	2
243	CASSETTE DECK	14	2
351	CASSETTE TAPE	2500	2
380	SHAVER	870	3
390	DRIER	540	3

・ ODBC情報ファイル設定ツール(SQLODBCS.exe)を使用して、ODBC情報ファイル(ここではDBMSACS.INFとします)を作成してください。

6.13.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

参考

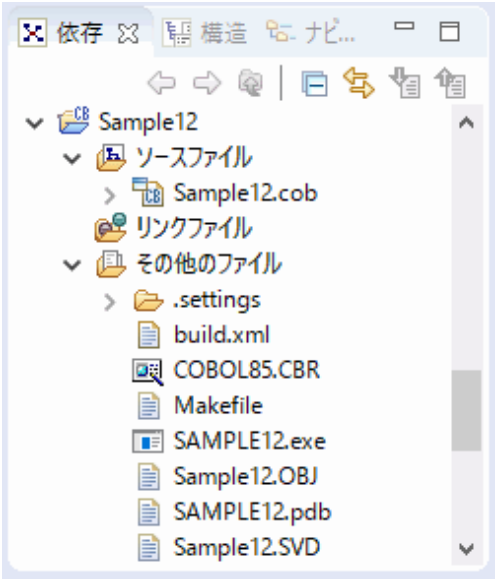
“ワークスペースを準備する”

2. [依存]ビューを確認し、Sample12プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

参考

“サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample12プロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. [その他のファイル]にSample12.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

→ プロジェクトのビルドが行われ、Sample12.exeが作成されます。

実行環境情報の設定

1. [実行環境設定]ツールを起動します。

[実行環境設定]ツールを起動する方法には以下があります。

- [スタート]> お使いのNetCOBOL製品 > [実行環境設定ツール]を選択します。
- [NetCOBOLコマンドプロンプト]から、実行環境設定ツール(COBENVUT.exe)を起動します。

2. [実行環境設定ツール]のメニューバーから、[ファイル] > [開く]を選択します。

→ [実行用の初期化ファイルの指定]ウィンドウが表示されます。

3. 実行可能プログラム(Sample12.exe)が存在するフォルダーのCOBOL85.CBRを選択し、[開く]ボタンをクリックします。
NetCOBOL Studioからビルドした場合、実行可能プログラムは、プロジェクトフォルダーに作成されています。

→ 実行用の初期化ファイルの内容が表示されます。

4. [共通]タブを選択し、以下の設定を確認します。

- 環境変数情報@ODBC_Inf(ODBC情報ファイルの指定)に、ODBC情報ファイル名が指定されている。
- 環境変数RESULTに、DISPLAY文の出力結果を保存するファイルが指定されている。

相対パスでファイルを指定する場合、カレントフォルダーからの相対パスになります。

NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択して実行する場合、カレントフォルダーはプロジェクトフォルダーです。

5. [適用]ボタンをクリックします。

→ 設定した内容が実行用の初期化ファイルに保存されます。

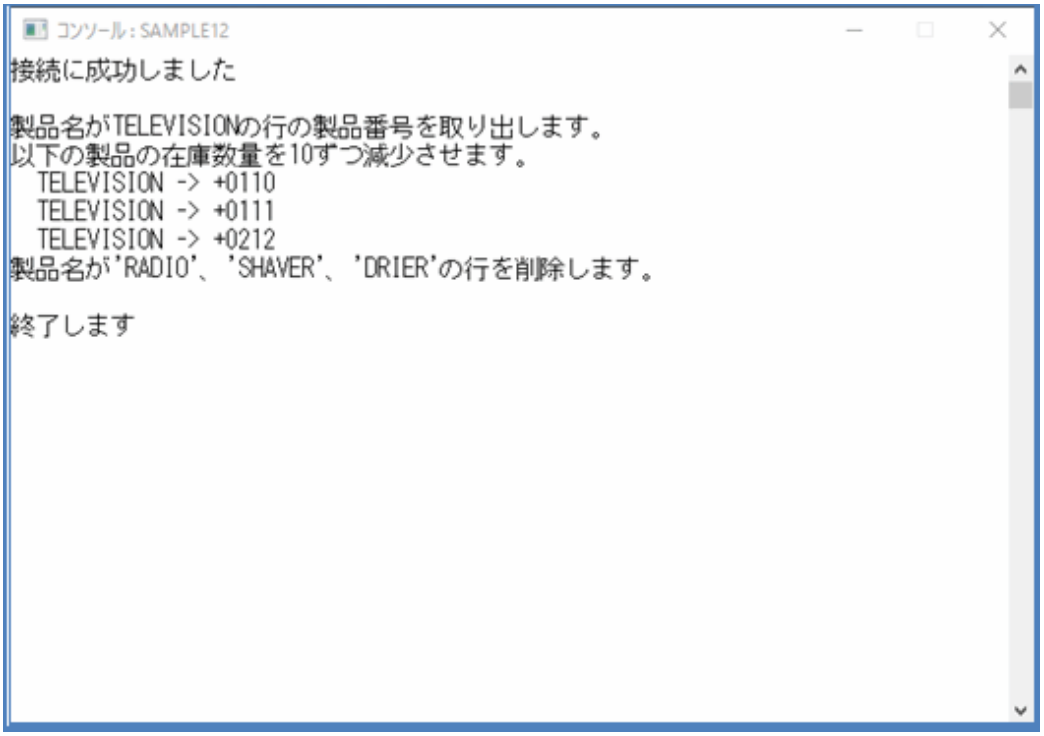
6. [ファイル]メニューの[終了]を選択し、実行環境設定ツールを終了します。

プログラムの実行

[依存]ビューからSample12プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択します。

実行結果

COBOLのコンソールウィンドウに、以下が表示されます。



環境変数RESULTに割り当てたファイルには、次の形式で操作の前後でのSTOCKテーブルの内容が出力されます。

```
処理前のテーブルの内容

01件目のデータ：
        製品番号   ＝  +0110
        製品名     ＝  TELEVISION
        在庫数量   ＝  +000000085
        倉庫番号   ＝  +0002
                    ・
                    ・
19件目のデータ：
        製品番号   ＝  +0390
        製品名     ＝  DRIER
        在庫数量   ＝  +000000540
        倉庫番号   ＝  +0003

全データ件数は19件です

処理後のテーブルの内容

01件目のデータ：
        製品番号   ＝  +0110
        製品名     ＝  TELEVISION
        在庫数量   ＝  +000000075
        倉庫番号   ＝  +0002
```

```
                    .
                    .
15件目のデータ：
        製品番号   ＝  +0351
        製品名     ＝  CASSETTE TAPE
        在庫数量   ＝  +000002500
        倉庫番号   ＝  +0002

全データ件数は15件です
```

6.13.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample12>nmake
```

翻訳およびリンク終了後、Sample12.exeが作成されていることを確認してください

実行環境情報の設定

NetCOBOL Studioを利用する場合と同じです。

プログラムの実行

コマンドプロンプトまたはエクスプローラからSample12.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.14 Visual Basicからの呼出し(Sample13)

ここでは、本製品で提供するサンプルプログラム-Sample13-について説明します。

Sample13では、Visual Basicアプリケーションから、NetCOBOLで作成したDLLを呼び出すプログラムの例を示します。

なお、このプログラムをビルドおよび動作させるためには、Microsoft .NET Framework 4.0以降が必要です。

概要

Visual Basicアプリケーションを起動し、フォームがロードされたときにCOBOLプログラムの初期化手続きを行うサブルーチンJMPCINT2を呼び出します。

4桁以下の2つの数値をVisual Basicアプリケーションのテキストボックスから入力し、[=](イコール)ボタンを押すと、その2つの数値がCOBOLアプリケーションに渡されます。

COBOLアプリケーションは、2つの数値を乗算し、結果を文字に編集してVisual Basicアプリケーションに返却します。Visual Basicアプリケーションでは、返却された編集結果の文字をテキストボックスに出力します。

Visual Basicアプリケーションを終了し、フォームがアンロードされたときに、COBOLプログラムの終了手続きを行うサブルーチンJMPCINT3を呼び出します。

提供プログラム

- Sample13.cob（COBOLソースプログラム）
- Makefile_VB（Visual Basicプロジェクトをビルドするメイクファイル）
- Makefile_COBOL（NetCOBOLプロジェクトをビルドするメイクファイル）
- VBProj¥AssemblyInfo.vb（Visual Basic アセンブリ情報ファイル）
- VBProj¥Sample13.sln（Visual Basic ソリューションファイル）

- VBProj¥Sample13.vbproj（Visual Basic プロジェクトファイル）
- VBProj¥Sample13.vb（Visual Basic ソースコードファイル）
- VBProj¥Sample13.resX（Visual Basic XMリソースファイル）
- VBProj¥Sample13.Designer.vb（Visual Basic デザイナコードファイル）
- COBOL85.CBR（実行用の初期化ファイル）

使用しているCOBOLの機能

- Visual Basicからの呼出し
- COBOLランタイムシステムのサブルーチン(JMPCINT2、JMPCINT3)の使用

6.14.1 NetCOBOL Studioを利用する場合

Visual Basicプログラムの翻訳・リンク

コマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行い、実行可能ファイルを作成します。

```
C:¥COBOL¥Samples¥COBOL¥Sample13>nmake -f MakeFile_VB
```

翻訳およびリンク終了後、実行可能プログラムSample13.exeが作成されていることを確認してください。
上記の例の場合、Sample13.exeは以下に作成されます。

```
C:¥COBOL¥Samples¥COBOL¥Sample13¥VBProj¥bin¥x64¥Release¥Sample13.exe
```

注意

Microsoft .NET Framework v4.0.30319を使用しています。Frameworkのバージョンが異なる場合は、Makefile_VBのNETFRAMEWORKPATHに正しい値を設定してください。

COBOLプログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

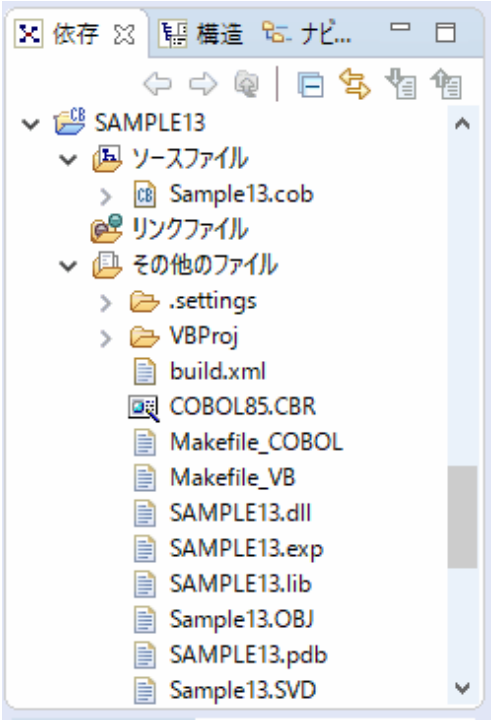
   “ワークスペースを準備する”

2. [依存]ビューを確認し、Sample13プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample13プロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.dllや.objなど)が表示されます。既定では自動ビルドに設定されています。

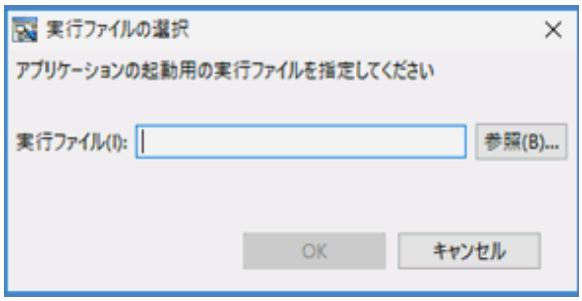
4. [その他のファイル]にSample13.dllが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、Sample13.dllが作成されます。

プログラムの実行

1. [依存]ビューからSample13プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択します。

   → [実行ファイルの選択]ダイアログボックスが表示されます。



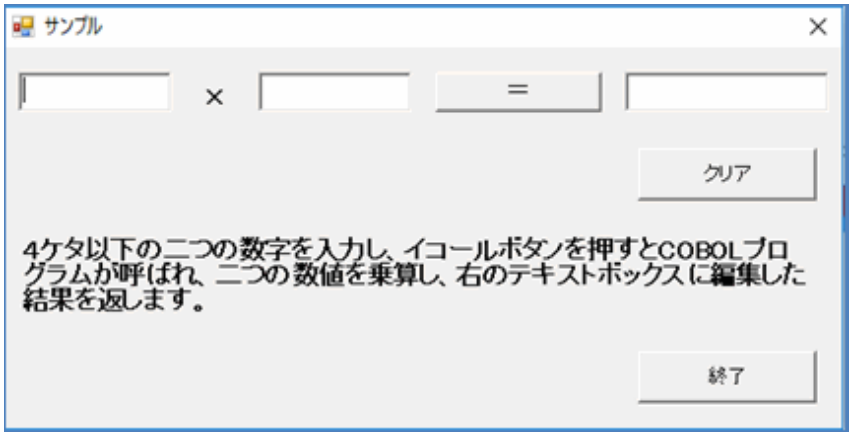
2. [実行ファイル]に以下のファイルを指定し、[OK]ボタンをクリックします。

```
C:¥COBOL¥Samples¥COBOL¥Sample13¥VBProj¥bin¥x64¥Release¥Sample13.exe
```

実行結果

1. プログラムを実行すると、Visual Basicで作成したフォームが表示されます。

   フォームには、左辺を示す2つの入力用のテキストボックスおよび右辺を示す1つの出力用テキストボックスがあります。



2. 左辺を示すテキストボックスに数値を入力します。
3. [=](イコール)ボタンを押すと、COBOLで作成したアプリケーションに左辺で指定した数値が渡されます。

   COBOLアプリケーションでは、2つの数値を乗算し、編集項目に格納し、これをVisual Basicアプリケーションに返却します。
4. Visual Basicアプリケーションは、返却された文字を右辺に示すテキストボックスに表示します。
5. 各テキストボックスの値をクリアするには、[クリア]ボタンを押します。

6.14.2 MAKEファイルを利用する場合

Visual Basicプログラムの翻訳・リンク

NetCOBOL Studioを利用する場合と同じです。

COBOLプログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample13>nmake -f Makefile_COBOL
```

翻訳およびリンク終了後、Sample13.dllが作成されていることを確認してください。

プログラムの実行

Sample13.dllファイルが、カレントフォルダーまたは環境変数PATHに設定したフォルダーにあることを確認してください。確認後、コマンドプロンプトまたはエクスプローラからSample13.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.15 Visual Basicを使った簡易ATM端末処理機能(Sample14)

ここでは、本製品で提供するサンプルプログラム-Sample14-について説明します。

Sample14では、Visual Basicアプリケーションから、COBOLで作成したDLLの呼出し方法を簡易ATM端末処理機能のプログラム例で示します。

なお、このプログラムをビルドおよび動作させるためには、Microsoft .NET Framework 4.0以降が必要です。

概要

サンプルプログラムは、以下のATM端末の機能を実現しています。

- 新規口座の作成
- 入金処理
- 出金処理

口座のデータ(口座番号、暗証番号、氏名および貯金額)は、索引ファイルに保存します。

索引ファイルの構造は、以下に示すとおりです。

```
口座番号    9(5) 主レコードキー
暗証番号    9(4)
氏名        N(6)
貯金額      9(6)
```

ATM端末から要求された機能により、索引ファイル内の任意の口座のレコードのデータを更新します。

提供プログラム

- K_ken.cob（COBOLソースプログラム）：口座番号を検索する
- K_sin.cob（COBOLソースプログラム）：新規口座を作成する
- K_nyu.cob（COBOLソースプログラム）：口座に入金を行う
- K_syu.cob（COBOLソースプログラム）：口座に出金を行う
- Makefile_VB（Visual Basicプロジェクトをビルドするメイクファイル）
- Makefile_COBOL（NetCOBOLプロジェクトをビルドするメイクファイル）
- VBProj¥AssemblyInfo.vb（Visual Basic アセンブリ情報ファイル）
- VBProj¥Sample14.sln（Visual Basic ソリューションファイル）
- VBProj¥Sample14.vbproj（Visual Basic プロジェクトファイル）
- VBProj¥Sample14_bas.vb（Visual Basic標準モジュール）
- VBProj¥Sample14_frm.vb（Visual Basic ソースコードファイル）：簡易ATM 端末処理のメイン処理を行う
- VBProj¥Sample14_frm.resX（Visual Basic XMLリソースファイル）：簡易ATM 端末処理のメイン処理を行う
- VBProj¥Sample14_frm.Designer.vb（Visual Basic デザイナコードファイル）：簡易ATM 端末処理のメイン処理を行う
- VBProj¥Sinki.vb（Visual Basic ソースコードファイル）：新規口座を開く
- VBProj¥Sinki.resX（Visual Basic XMLリソースファイル）：新規口座を開く
- VBProj¥Sinki.Designer.vb（Visual Basic デザイナコードファイル）：新規口座を開く
- VBProj¥Sinkichk.vb（Visual Basic ソースコードファイル）：新規口座について口座番号を確認する
- VBProj¥Sinkichk.resX（Visual Basic XMLリソースファイル）：新規口座について口座番号を確認する
- VBProj¥Sinkichk.Designer.vb（Visual Basic デザイナコードファイル）：新規口座について口座番号を確認する
- VBProj¥Sele.vb（Visual Basic ソースコードファイル）：入金/出金選択を行う
- VBProj¥Sele.resX（Visual Basic XMLリソースファイル）：入金/出金選択を行う
- VBProj¥Sele.Designer.vb（Visual Basic デザイナコードファイル）：入金/出金選択を行う
- VBProj¥Nyukin.vb（Visual Basic ソースコードファイル）：入金処理を行う
- VBProj¥Nyukin.resX（Visual Basic XMKリソースファイル）：入金処理を行う
- VBProj¥Nyukin.Designer.vb（Visual Basic デザイナコードファイル）：入金処理を行う

- VBProj¥Syukin.vb（Visual Basic ソースコードファイル）：出金処理を行う
- VBProj¥Syukin.resX（Visual Basic XMLリソースファイル）：出金処理を行う
- VBProj¥Syukin.Designer.vb（Visual Basic デザイナコードファイル）：出金処理を行う
- VBProj¥Error_h.vb（Visual Basic ソースコードファイル）：エラーメッセージを表示する
- VBProj¥Error_h.resX（Visual Basic XMLリソースファイル）：エラーメッセージを表示する
- VBProj¥Error_h.Designer.vb（Visual Basic デザイナコードファイル）：エラーメッセージを表示する
- COBOL85.CBR（実行用の初期化ファイル）

使用しているCOBOLの機能

- Visual Basicからの呼出し
- COBOLランタイムシステムのサブルーチン(JMPCINT2、JMPCINT3)の使用

使用しているCOBOLの文

- MOVE文
- IF文
- PERFORM文
- COMPUTE文
- OPEN文
- READ文
- WRITE文
- REWRITE文
- CLOSE文
- EXIT文

プログラムの内容

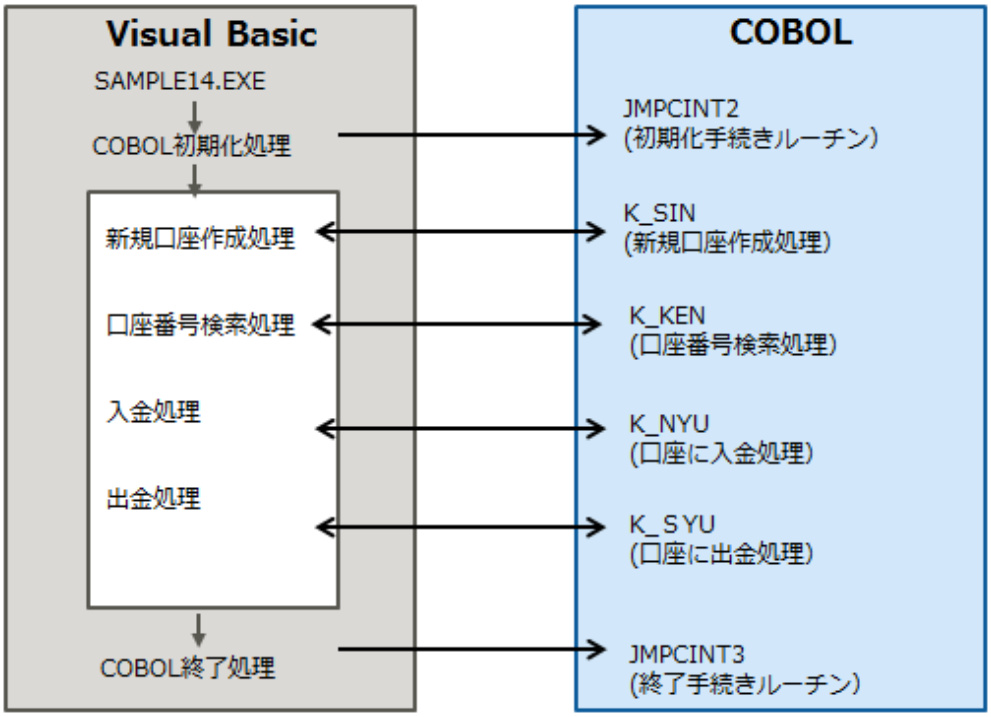
Visual Basicアプリケーションを起動し、フォームがロードされたときにCOBOLプログラムの初期化手続きを行うサブルーチンJMPCINT2を呼び出します。

[新規口座]ボタンを押すと、入力用フォームが開き、各項目を入力して[OK]ボタンを押すと、入力した内容がCOBOLアプリケーションに渡されます。

COBOLアプリケーションは、入力した内容をレコードに書き込み、数値をVisual Basicアプリケーションに返却します。

Visual Basicアプリケーションでは、返却された数値をテキストボックスに出力します。

Visual Basicアプリケーションを終了し、フォームがアンロードされたときに、COBOLプログラムの終了手続きを行うサブルーチンJMPCINT3を呼び出します。



6.15.1 NetCOBOL Studioを利用する場合

Visual Basicプログラムの翻訳・リンク

コマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行い、実行可能ファイルを作成します。

```
C:¥COBOL¥Samples¥COBOL¥Sample14>nmake -f MakeFile_VB
```

翻訳およびリンク終了後、実行可能プログラムSample14.exeが作成されていることを確認してください。
上記の例の場合、Sample14.exeは以下に作成されます。

```
C:¥COBOL¥Samples¥COBOL¥Sample14¥VBProj¥bin¥x64¥Release¥Sample14.exe
```

注意

Microsoft .NET Framework v4.0.30319を使用しています。Frameworkのバージョンが異なる場合は、Makefile_VBのNETFRAMEWORKPATHに正しい値を設定してください。

COBOLプログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

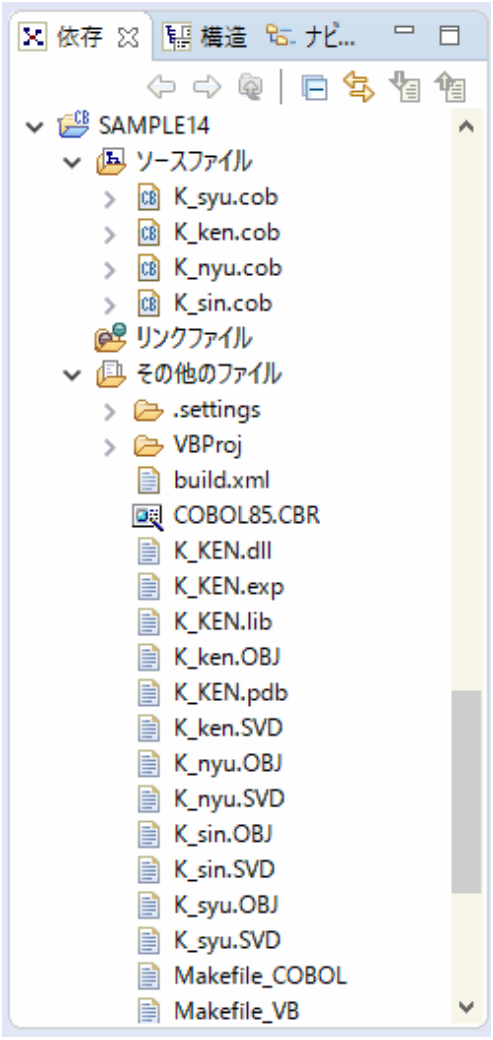
   “ワークスペースを準備する”

2. [依存]ビューを確認し、Sample14プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

参考

“サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample14プロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.dllや.objなど)が表示されます。既定では自動ビルドに設定されています。

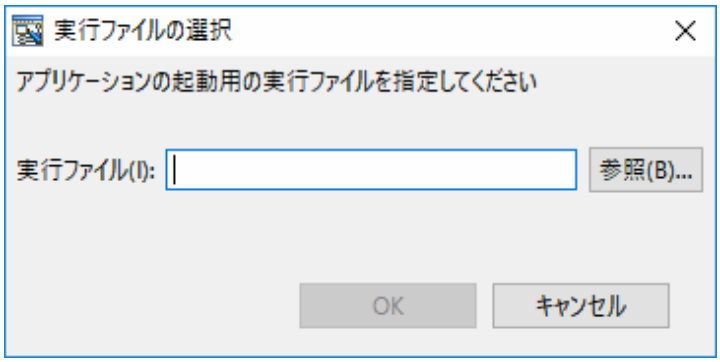
4. [その他のファイル]にK_KEN.dllが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

→ プロジェクトのビルドが行われ、K_KEN.dllが作成されます。

プログラムの実行

1. [依存]ビューからSample14プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択します。

   → [実行ファイルの選択]ダイアログボックスが表示されます。

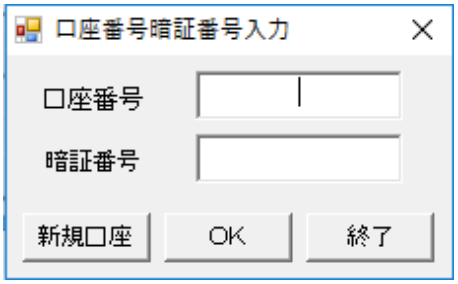


2. [実行ファイル]に以下のファイルを指定し、[OK]ボタンをクリックします。

```
C:¥COBOL¥Samples¥COBOL¥Sample14¥VBProj¥bin¥x64¥Release¥Sample14.exe
```

実行結果

[口座番号暗証番号入力]ダイアログ



1. [新規口座]ボタンをクリックします。

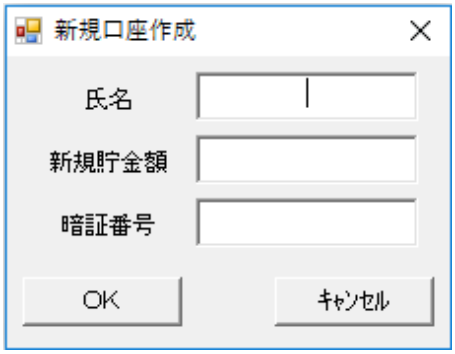
   → [新規口座作成]ダイアログが表示されます。ここで、新規口座を作成し、口座番号と暗証番号を取得します。

2. 取得した口座番号および暗証番号を入力し、[OK]ボタンをクリックします。

   → 該当する口座番号の[口座番号、氏名、貯金額表示]ダイアログが表示されます。エラーが発生した場合は、[エラー画面]ダイアログが表示されます。

3. プログラムを終了するときには、[終了]ボタンをクリックします。

[新規口座作成]ダイアログ



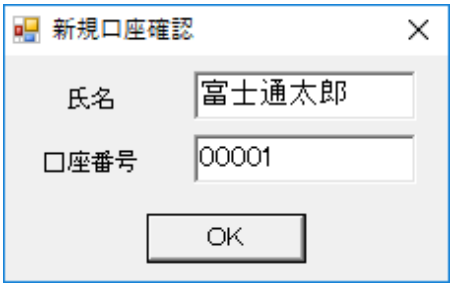
1. 氏名、貯金額、暗証番号を入力し、[OK]ボタンをクリックします。

   → 新規口座作成処理が行われ、[新規口座確認]ダイアログが表示されます。エラーが発生した場合は、[エラー画面]ダイアログが表示されます。

2. 新規口座作成を取りやめる場合は、[キャンセル]ボタンをクリックします。

   →［口座番号暗証番号入力]ダイアログに戻ります。

[新規口座確認]ダイアログ



1. 口座番号を確認して[OK]ボタンをクリックします。

   →［口座番号暗証番号入力]ダイアログに戻ります。

[口座番号、氏名、貯金額表示]ダイアログ

1. 出金の場合は、[出金]ボタンをクリックします。

   →［出金]ダイアログが表示されます。

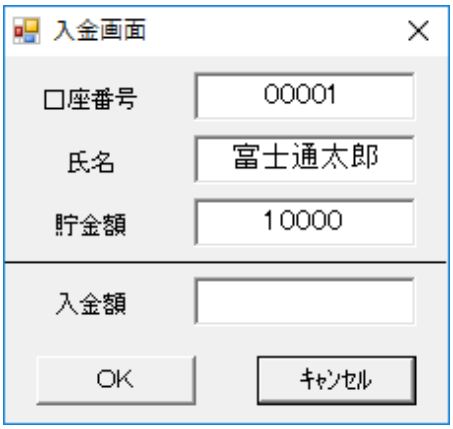
2. 入金の場合は、[入金]ボタンをクリックします。

   →［入金]ダイアログが表示されます。

3. 処理を中断する場合は、[キャンセル]ボタンをクリックします。

   →［口座番号暗証番号入力]ダイアログに戻ります。

[入金画面]ダイアログ



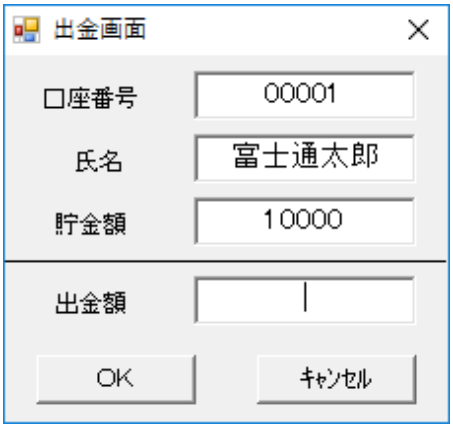
1. 入金額を入力し、[OK]ボタンをクリックします。

   → 入金処理が行われ、[口座番号、氏名、貯金額表示]ダイアログが表示されます。エラーが発生した場合は、[エラー画面]ダイアログが表示されます。

2. 処理を中断する場合は、[キャンセル]ボタンをクリックします。

   → [口座番号、氏名、貯金額表示]ダイアログに戻ります。

[出金画面]ダイアログ



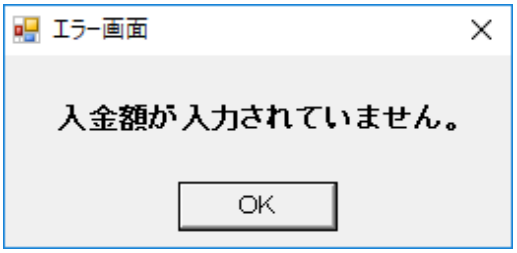
1. 出金額を入力し、[OK]ボタンをクリックします。

   → 出金処理が行われ、[口座番号、氏名、貯金額表示]ダイアログが表示されます。エラーが発生した場合は、[エラー画面]ダイアログが表示されます。

2. 処理を中断する場合は、[キャンセル]ボタンをクリックします。

   → [口座番号、氏名、貯金額表示]ダイアログに戻ります。

[エラー画面]ダイアログ



1. エラーメッセージを確認し、[OK]ボタンをクリックします。

   → エラー元の画面に戻ります。

6.15.2 MAKEファイルを利用する場合

Visual Basicプログラムの翻訳・リンク

NetCOBOL Studioを利用する場合と同じです。

COBOLプログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample14>nmake -f Makefile_COBOL
```

翻訳およびリンク終了後、K_KEN.dllが作成されていることを確認してください。

プログラムの実行

K_KEN.dllファイルが、カレントフォルダーまたは環境変数PATHに設定したフォルダーにあることを確認してください。コマンドプロンプトまたはエクスプローラからSample14.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.16 オブジェクト指向プログラム（初級編）(Sample15)

ここでは、本製品で提供するサンプルプログラム-Sample15-について説明します。

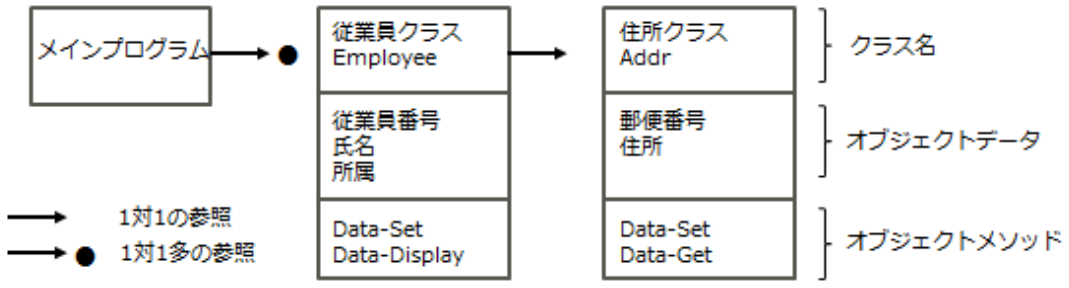
Sample15では、オブジェクト指向プログラミング機能を使ったプログラムの例を示します。このプログラムでは、カプセル化、オブジェクトの生成、メソッド呼出しといった、オブジェクト指向の基本的な機能だけを使用しています。

概要

最初に従業員オブジェクトを3つ生成しています。“NEW”メソッドでオブジェクトを生成した後、“Data-Set”を呼び出してデータを設定しています。それぞれの従業員オブジェクトはすべて同じ形をしていますが、持っているデータ(従業員番号、氏名、所属、住所情報)は異なります。また、住所情報もオブジェクトであり、郵便番号と住所を持っています。

画面から従業員番号を入力すると、該当する従業員オブジェクトに対して“Data-Display”メソッドを呼び出し、そのオブジェクトが持っている従業員情報を画面に表示します。このとき、従業員オブジェクトは、住所の情報を得るために、従業員オブジェクトが指している住所オブジェクトに対し、“Data-Get”メソッドを呼び出します。

従業員オブジェクトは、3つのデータと住所オブジェクトから構成されています。しかし、これを使う側(この場合はメインプログラム)はオブジェクトの構造を知っている必要はありません。“Data-Set”と“Data-Display”の2つのメソッドだけを知っていれば十分です。つまり、データとアクセス手段を1つにまとめる(カプセル化)ことにより、オブジェクト内部の情報を完全に隠蔽しているわけです。



提供プログラム

- Main.cob（COBOLソースプログラム）
- Member.cob（COBOLソースプログラム）
- Address.cob（COBOLソースプログラム）

- Makefile（メイクファイル）
- COBOL85.CBR（実行用の初期化ファイル）

使用しているCOBOLの機能

- オブジェクト指向プログラミング機能
  - クラスの定義(カプセル化)
  - オブジェクトの生成
  - メソッド呼出し

使用しているオブジェクト指向の文/段落/定義

- INVOKE文、SET文
- リポジトリ段落
- クラス定義、オブジェクト定義、メソッド定義

6.16.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

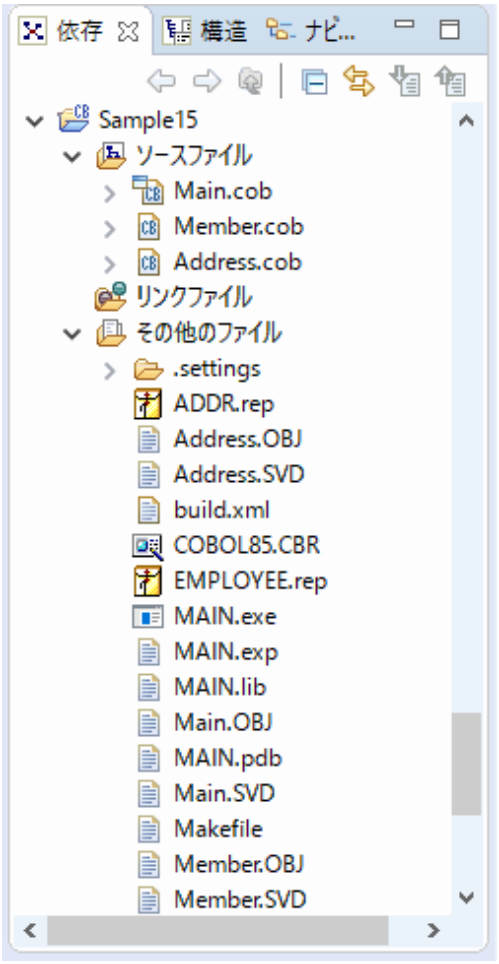
   “ワークスペースを準備する”

2. ［依存］ビューを確認し、Sample15プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample15プロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. [その他のファイル]MAIN.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

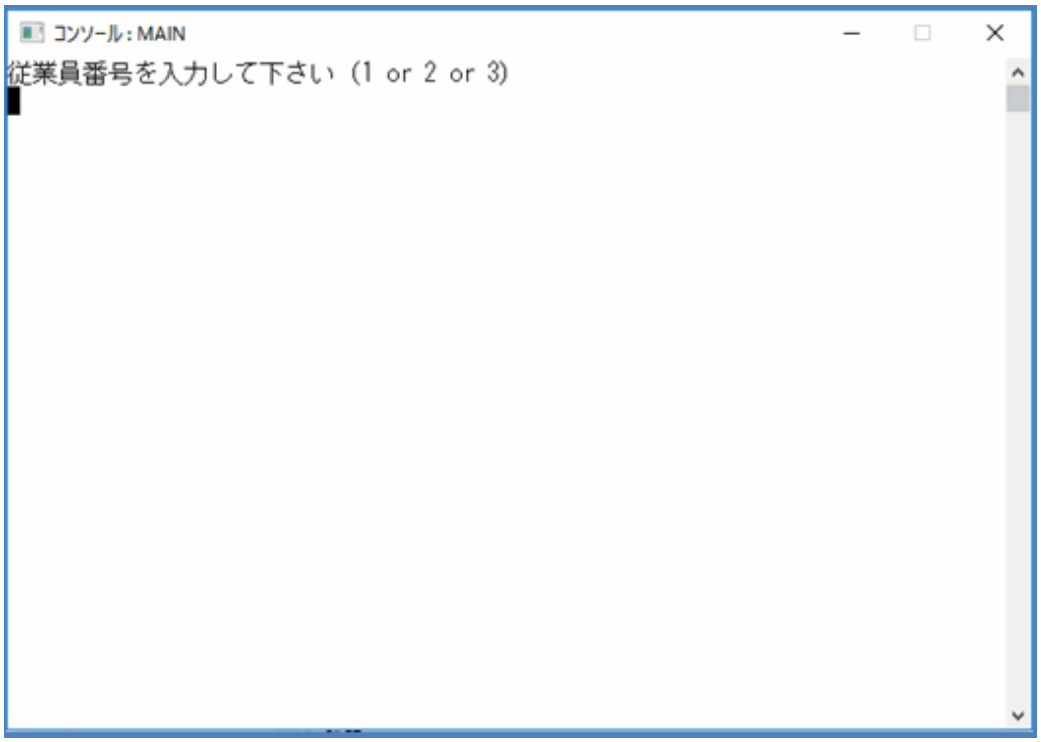
   → プロジェクトのビルドが行われ、MAIN.exeが作成されます。

プログラムの実行

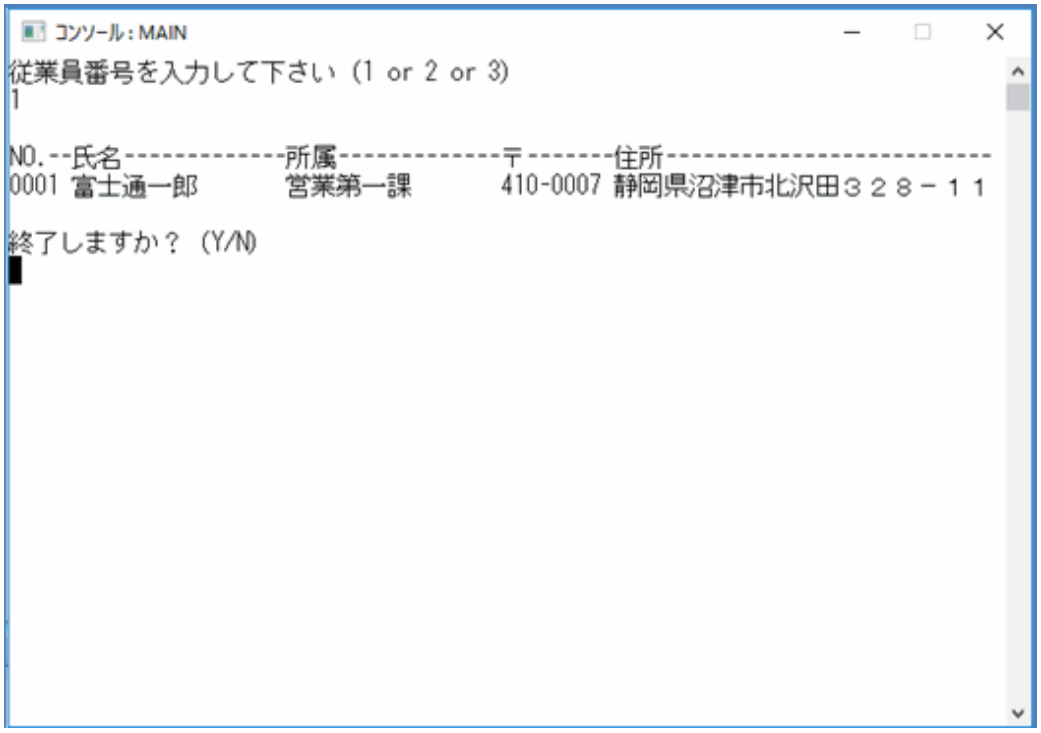
[依存]ビューからSample15プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択します。

実行結果

1. 従業員番号を入力するためのCOBOLコンソールが表示されます。



2. 従業員番号(1～3)を入力すると、従業員情報が表示されます。



6.16.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample15>nmake
```

翻訳およびリンク終了後、MAIN.exeが作成されていることを確認してください。

プログラムの実行

コマンドプロンプトまたはエクスプローラからMAIN.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.17 コレクションクラス(クラスライブラリ)(Sample16)

ここでは、本製品で提供するサンプルプログラム-Sample16-について説明します。

Sample16では、クラスライブラリの作成方法の例として、コレクションクラスの例を示します。

このサンプルは、クラスライブラリの作成方法の例として使用するだけでなく、実際にプログラムに組み込んで使用することができます。また、このサンプルには基本的な操作しか含まれていませんが、改造・変更することにより、さらに使いやすいクラスライブラリにすることができます。

注意

翻訳およびリンク以降では、NetCOBOLのインストール先フォルダーをC:¥COBOLとして説明しています。フォルダー名がC:¥COBOLになっているところは、NetCOBOLをインストールしたフォルダーに変更してください。

概要

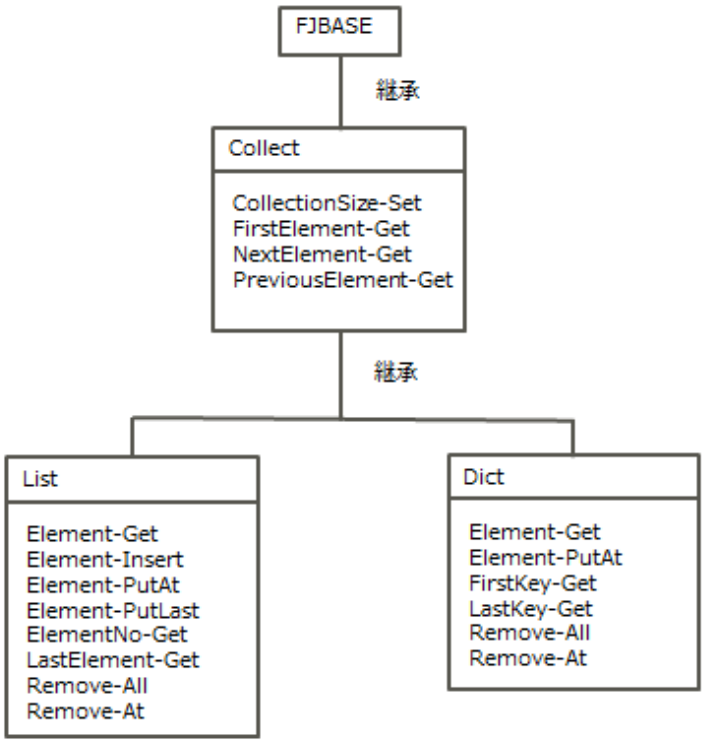
“コレクションクラス”とは、集合を扱うクラスの総称です。つまり、たくさんのオブジェクトをまとめて扱ったり、格納したりするためのクラスです。サンプルには、以下のクラスが含まれています。

- Collect(Collection:収集)
- Dict(Dictionary:辞書)
- List(リスト)

プログラムの内容

クラス階層

Sample16のクラスの階層関係を下図に示します。



参考

サンプルクラスには、上記の他にBinaryTree-Class、DictionaryNode-ClassおよびListNode-Classが含まれています。これらのクラスは、ListクラスおよびDictクラス内部で使用しているクラスで、コレクションクラス使用者からは見えなくなっています。そのため、ここではこれらのクラスの説明は省略します。

Collectクラス

コレクションクラスの最上位のクラスです。すべてのコレクションクラスはこのクラスを継承しています。Collectは抽象クラスであり、オブジェクトを作成しません。

このクラスは、FJBASEクラスを継承しているので、FJBASEクラスで定義されているメソッドもすべて使用することができます。

定義

```
CLASS-ID.  Collect  INHERITS  FJBASE.
 ENVIRONMENT  DIVISION.
 CONFIGURATION  SECTION.
   REPOSITORY.
     CLASS  FJBASE.
   OBJECT.
   PROCEDURE  DIVISION.
   METHOD-ID.  CollectionSize-Get.
   METHOD-ID.  FirstElement-Get.
   METHOD-ID.  NextElement-Get.
   METHOD-ID.  PreviousElement-Get.
   END  OBJECT.
 END  CLASS  Collect.
```

CollectionSize-Getメソッド

集合の要素数を調べます。

パラメタ

なし

復帰値

PIC 9(8) BINARY ： 集合の要素数を返します。

FirstElement-Getメソッド

集合の先頭の要素を取り出します。

パラメタ

なし

復帰値

USAGE OBJECT REFERENCE ： 集合の先頭の要素を返します。要素がない場合は、NULLを返します。

NextElement-Getメソッド

現在指している要素の次の要素を取り出します。

パラメタ

なし

復帰値

USAGE OBJECT REFERENCE ： 現在指している要素の次の要素を返します。次の要素がない場合は、NULLを返します。

PreviousElement-Getメソッド

現在指している要素の直前の要素を取り出します。

パラメタ

なし

復帰値

USAGE OBJECT REFERENCE ： 直前の要素を返します。直前の要素がない場合は、NULLを返します。

Dictクラス

以下の特徴を持つ集合クラスです。

－ 各要素はキーを持っています。

－ キーの値は一意です。

－ キーによる検索ができます。

－ キーにより順序付けされています。

このクラスは、Collectクラスを継承しているので、Collectクラスで定義されているメソッドもすべて使用することができます。

定義

```
CLASS-ID.  Dict  INHERITS  Collect.
 ENVIRONMENT  DIVISION.
 CONFIGURATION  SECTION.
   REPOSITORY.
     CLASS  Collect.
   OBJECT.
   PROCEDURE  DIVISION.
   METHOD-ID.  Element-Get.
   METHOD-ID.  Element-PutAt.
   METHOD-ID.  FirstKey-Get.
   METHOD-ID.  LastKey-Get.
   METHOD-ID.  Remove-All.
   METHOD-ID.  Remove-At.
   END  OBJECT.
 END  CLASS  Dict.
```

Element-Getメソッド

指定されたキーの要素を取り出します。

パラメタ

PIC X ANY LENGTH ： 取り出す要素のキー値を指定します。

復帰値

USAGE OBJECT REFERENCE ： 指定されたキーの要素が見つかった場合はその要素を、見つからなかった場合はNULLを返します。

Element-PutAtメソッド

指定されたキーの要素を追加します。すでに同じキーを持つ要素が存在している場合は、新しい要素で置き換えます。

パラメタ

PIC X ANY LENGTH ： 追加・置換する要素のキー値を指定します。

USAGE OBJECT REFERENCE ： 追加・置換する要素を指定します。

復帰値

なし

FirstKey-Getメソッド

先頭の要素のキー値を求めます。

パラメタ

なし

復帰値

PIC X ANY LENGTH ： 先頭の要素のキー値を返します。要素数が0の場合または先頭の要素のキーが空白の場合、空白を返します。

LastKey-Getメソッド

最後の要素のキー値を求めます。

パラメタ

なし

復帰値

PIC X ANY LENGTH ： 最後の要素のキー値を返します。要素数が0の場合または最後の要素のキーが空白の場合、空白を返します。

Remove-Allメソッド

集合に含まれるすべての要素を削除します。

パラメタ

なし

復帰値

なし

Remove-Atメソッド

指定されたキーの要素を削除します。

パラメタ

PIC X ANY LENGTH ： 削除する要素のキー値を指定します。

復帰値

なし

Listクラス

以下の特徴を持つ集合クラスです。

ー 要素間に順序があります。

ー 同一の要素を複数含むことができます。

このクラスは、Collectクラスを継承しているので、Collectクラスで定義されているメソッドもすべて使用することができます。

定義

```
CLASS-ID.  List  INHERITS  Collect.
 ENVIRONMENT  DIVISION.
 CONFIGURATION  SECTION.
   REPOSITORY.
     CLASS  Collect.
   OBJECT.
   PROCEDURE  DIVISION.
   METHOD-ID.  Element-Get.
   METHOD-ID.  Element-Insert.
   METHOD-ID.  Element-PutAt.
   METHOD-ID.  Element-PutLast.
   METHOD-ID.  ElementNo-Get.
   METHOD-ID.  LastElement-Get.
   METHOD-ID.  Remove-All.
   METHOD-ID.  Remove-At.
   END  OBJECT.
 END  CLASS  List.
```

Element-Getメソッド

指定された位置(インデックス)の要素を取り出します。

パラメタ

PIC 9(8) BINARY ： 取り出す要素の位置を、先頭を1とした整数で指定します。

復帰値

USAGE OBJECT REFERENCE ： 取り出した要素を返します。指定した位置の要素がなかった場合は、NULLを返します。

Element-Insertメソッド

指定された位置(インデックス)に要素を追加します。

パラメタ

PIC 9(8) BINARY ： 要素を追加する位置を、先頭を1とした整数で指定します。なお、要素数+1より大きい数を指定した場合は、要素は追加されません。

USAGE OBJECT REFERENCE ： 追加する要素を指定します。

復帰値

PIC 9(8) BINARY ： 要素を追加した位置を、先頭を1とした整数で返します。要素が追加されなかった場合は、0を返します。

Element-PutAtメソッド

指定された位置(インデックス)の要素を置き換えます。

パラメタ

PIC 9(8) BINARY：置き換える要素の位置を、先頭を1とした整数で指定します。なお、要素数より大きい数を指定した場合は、置き換えられません。

USAGE OBJECT REFERENCE ： 置き換える要素を指定します。

復帰値

PIC 9(4) BINARY ： 要素を置き換えた位置を、先頭を1とした整数で返します。要素が置き換えられなかった場合は、0を返します。

Element-PutLastメソッド

最後の要素の後に、要素を追加します。

パラメタ

USAGE OBJECT REFERENCE ： 追加する要素を指定します。

復帰値

なし

ElementNo-Getメソッド

指定した要素の位置(インデックス)を調べます。

パラメタ

USAGE OBJECT REFERENCE ： 位置を調べる要素を指定します。

復帰値

PIC 9(8) BINARY ： 要素の位置を、先頭を1とした整数で返します。指定した要素が見つからなかった場合は0を返します。同じ要素が複数存在する場合は、最初に見つかった位置を返します。

LastElement-Getメソッド

最後の要素を取り出します。

パラメタ

なし

復帰値

USAGE OBJECT REFERENCE ： 最後の要素を返します。要素数が0の場合は、NULLを返します。

Remove-Allメソッド

集合に含まれるすべての要素を削除します。

パラメタ

なし

復帰値

なし

Remove-Atメソッド

指定された位置(インデックス)の要素を削除します。

パラメタ

PIC 9(8) BINARY ： 削除する要素の位置を、先頭を1とした整数で指定します。なお、要素数より大きい数を指定した場合は、削除されません。

復帰値

PIC 9(4) BINARY ： 要素を削除した位置を、先頭を1とした整数で返します。削除されなかった場合は、0を返します。

提供プログラム

- Collect.cob（COBOLソースプログラム）
- Dict.cob（COBOLソースプログラム）
- List.cob（COBOLソースプログラム）
- Bin_Tree.cob（COBOLソースプログラム）
- D_Node.cob（COBOLソースプログラム）
- L_Node.cob（COBOLソースプログラム）
- Makefile（メイクファイル）

使用しているCOBOLの機能

- オブジェクト指向プログラミング機能
  - ー クラスの定義(カプセル化)

- 継承
- オブジェクトの生成
- メソッド呼出し

使用しているオブジェクト指向の文/段落/定義

- INVOKE文、SET文
- オブジェクトプロパティ
- メソッドの行内呼出し
- リポジトリ段落
- クラス定義、オブジェクト定義、メソッド定義

6.17.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

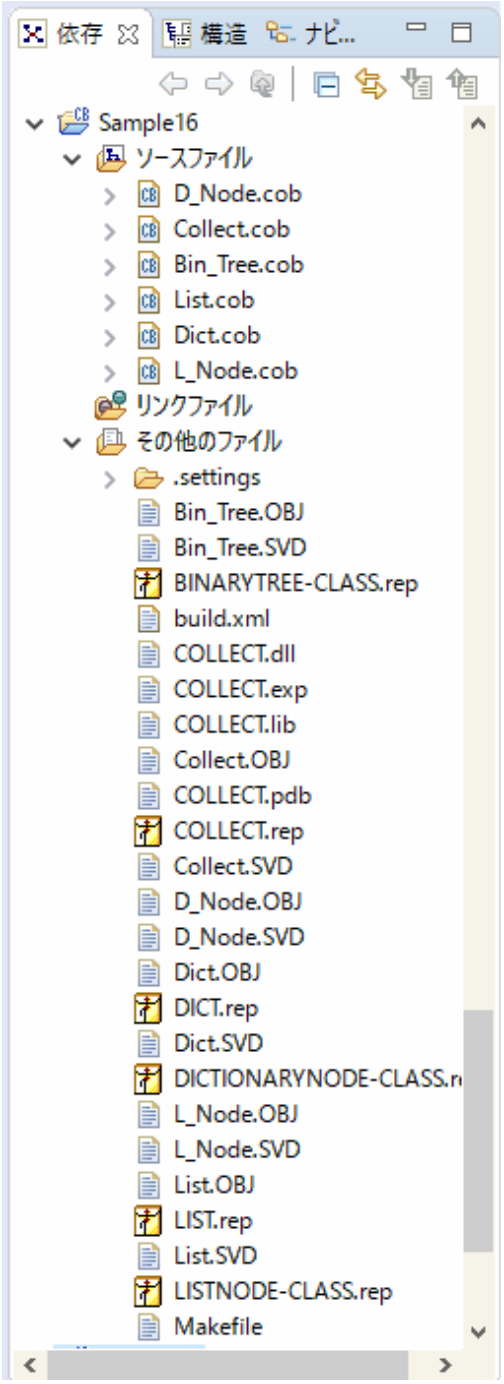
   “ワークスペースを準備する”

2. [依存]ビューを確認し、Sample16プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample16プロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.dllや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. [その他のファイル]に以下のファイルが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

- COLLECT.dll
- COLLECT.lib
- COLLECT.REP
- DICT.REP
- LIST.REP

参考

上記以外にも作成されるファイルがありますが、それらはクラスライブラリ使用時には必要ありません。

クラスライブラリの利用

サンプルクラスライブラリをプログラムに組み込んで使用する場合、以下のファイルが必要です。

翻訳時およびリンク時

- COLLECT.lib（インポートライブラリ）
- COLLECT.REP（リポジトリファイル）
- DICT.REP（リポジトリファイル）
- LIST.REP（リポジトリファイル）

これらのファイルを、クラスライブラリを使用するプロジェクトに組み込んで使用します。

実行時

- COLLECT.dll（ダイナミックリンクライブラリ）

6.17.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample16>nmake
```

翻訳およびリンク終了後、以下のファイルが作成されていることを確認してください。

- COLLECT.dll
- COLLECT.lib
- COLLECT.REP
- DICT.REP
- LIST.REP

6.18 Unicodeを使用するプログラム(Sample30)

ここでは、本製品で提供するサンプルプログラム-Sample30-について説明します。

Sample30では、UTF-16の行順ファイルを入力し、それらを表示・印刷するプログラムの例を示します。

概要

Unicode固有の漢字および英語の発音記号が格納されているファイル(UTF-16の行順ファイル)のレコードを読み出し、そのデータを出力します。画面には、UTF-16のデータを表示します。印刷ファイルには、UTF-16のデータの他に、レコード件数を示す数字をUTF-8で出力します。

提供プログラム

- Sample30.cob（COBOLソースプログラム）
- Makefile（メイクファイル）
- INDATA（入力ファイル）
- COBOL85.CBR（実行用の初期化ファイル）

使用しているCOBOLの機能

- プログラム間連絡機能
- 組込み関数機能
- 小入出力機能(コンソールウィンドウ)
- 印刷ファイル
- 行順ファイル(参照)
- 内部プログラム

使用しているCOBOLの文

- CALL文
- ACCEPT文
- DISPLAY文
- PERFORM文
- IF文
- EVALUATE文
- GO TO文
- MOVE文
- COMPUTE文
- OPEN文
- CLOSE文
- READ文
- WRITE文
- EXIT文

プログラムを実行する前に

印刷ファイルの内容が通常使うプリンターに出力されます。“通常使うプリンター”の設定を確認してください。

6.18.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

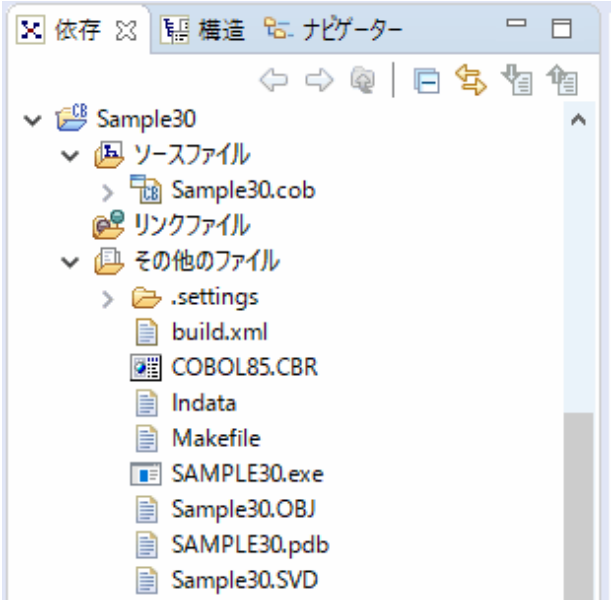
   “ワークスペースを準備する”

2. [依存]ビューを確認し、Sample30プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. ［依存］ビューからSample30プロジェクトを選択し、以下の構成になっていることを確認します。

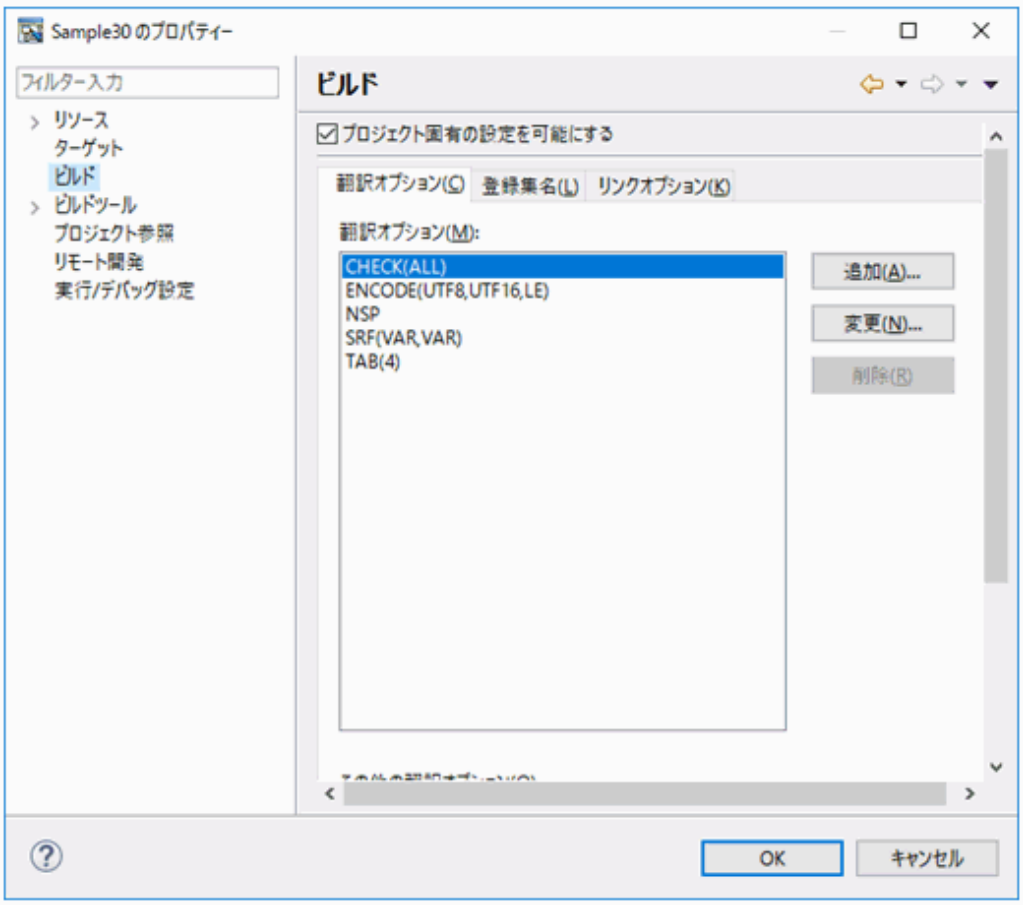


自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、［その他のファイル］には、ビルド後に生成されるファイル（.exeや.objなど）が表示されます。既定では自動ビルドに設定されています。

4. Sample30プロジェクトに翻訳オプションENCODE(UTF8,UTF16,LE)が指定されていることを確認します。
   翻訳オプションの設定は、NetCOBOL Studioの［依存］ビューからSample30プロジェクトを選択し、コンテキストメニューから［プロパティ］を選択します。

   → ［プロパティ］ダイアログボックスが表示されます。

5. 左ペインから[ビルド]を選択すると、[ビルド]ページが表示されます。[翻訳オプション]タブを選択して、設定オプションの内容を確認します。



翻訳オプションENCODE(UTF8,UTF16,LE)が指定されていることを確認して、[OK]ボタンをクリックします。

参考

翻訳オプションを設定するための詳細な手順は、“NetCOBOL Studio ユーザーズガイド”の“翻訳オプションの設定”を参照してください。

6. [その他のファイル]にSample30.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、Sample30.exeが作成されます。

実行環境情報の設定

1. [実行環境設定]ツールを起動します。

   [実行環境設定]ツールを起動する方法には以下があります。

   ― [スタート] > お使いのNetCOBOL製品 > [実行環境設定ツール]を選択します。

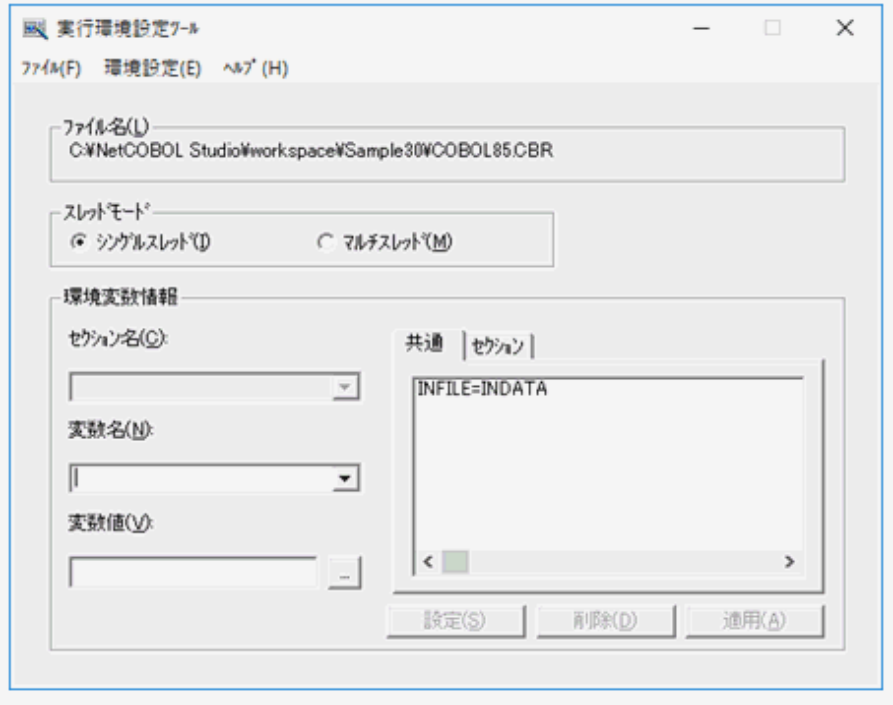
   ― [NetCOBOLコマンドプロンプト]から、実行環境設定ツール(COBENVUT.exe)を起動します。

2. [実行環境設定ツール]のメニューバーから、[ファイル] > [開く]を選択します。

   → [実行用の初期化ファイルの指定]ウィンドウが表示されます。

3. 実行可能プログラム(Sample30.exe)が存在するフォルダーのCOBOL85.CBRを選択し、[開く]ボタンをクリックします。
   NetCOBOL Studioからビルドした場合、実行可能プログラムは、プロジェクトフォルダーに作成されています。

   → 実行用の初期化ファイルの内容が表示されます。



4. [共通]タブを選択し、以下の設定を確認します。

   ー ファイル識別名INFILEにデータファイル(行順ファイル)のファイル名(INDATA)が指定されている。

   ```
   INFILE=.¥INDATA
   ```

   相対パスでファイルを指定する場合、カレントフォルダーからの相対パスになります。

   NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択して実行する場合、カレントフォルダーはプロジェクトフォルダーです。

5. [適用]ボタンをクリックします。

   → 設定した内容が実行用の初期化ファイルに保存されます。

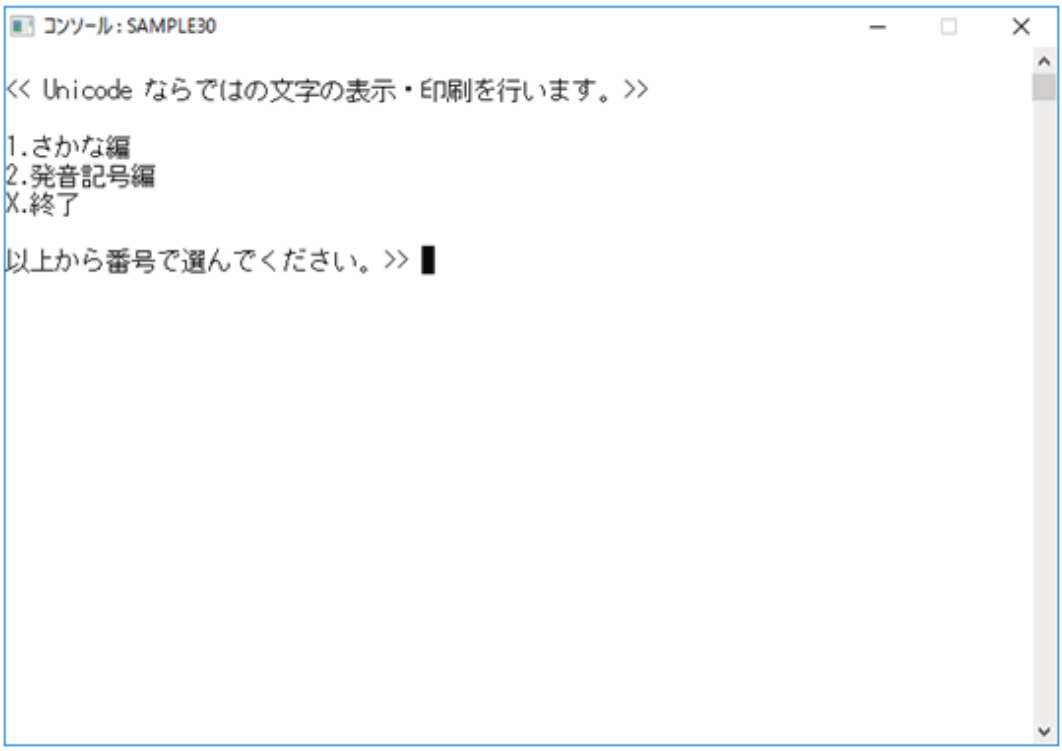
6. [ファイル]メニューの[終了]を選択し、実行環境設定ツールを終了します。

プログラムの実行

[依存]ビューからSample30プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択します。

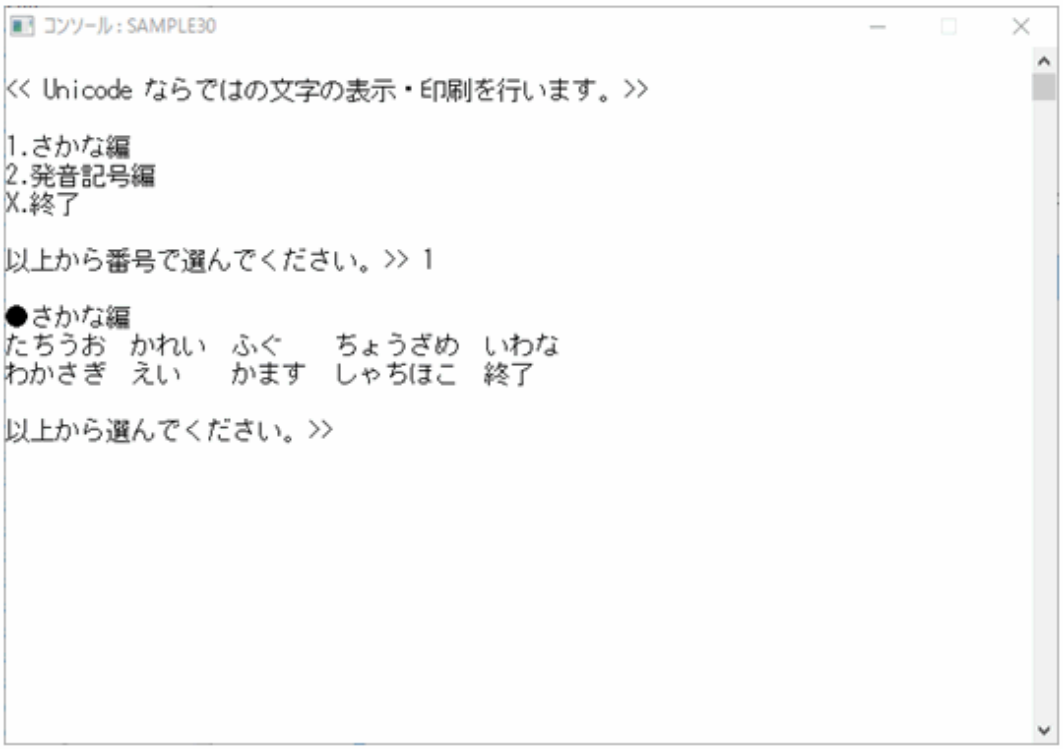
実行結果

1. COBOLコンソール画面が開き、以下のメッセージを表示して入力待ちになります。



2. 表示する文字列の種類を選択してください。

   “1”を入力し、ENTERキーを押すと、日本語文字列がいくつかひらがなで提示されます。それらのうちからひとつを選んで入力すると、対応する漢字が画面に表示されます。



3. 終了する場合、“X”を入力します。

   → 実行が終了すると、印刷ファイルの内容が通常使うプリンターに出力されます。

6.18.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample30>nmake
```

翻訳およびリンク終了後、Sample30.exeが作成されていることを確認してください

実行環境情報の設定

NetCOBOL Studioを利用する場合と同じです。

プログラムの実行

コマンドプロンプトまたはエクスプローラからSample30.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.19 メッセージボックスの出力(Sample31)

ここでは、本製品で提供するサンプルプログラム-Sample31-について説明します。

Sample31では、プログラム間連絡機能を使って、Windowsシステム関数を呼び出し、メッセージボックスを出力するプログラムの例を示します。プログラム間連絡機能の詳細は、“NetCOBOL ユーザーズガイド”の“サブプログラムを呼び出す～プログラム間連絡機能～”を参照してください。

なお、このSampleではCOBOLプログラムの復帰値をバッチファイルで参照します。

概要

CALL文でWindowsシステム関数の“MessageBoxA”(末尾に“A”が付いていることに注意してください)を呼び出します。[はい]/[いいえ]/[キャンセル]ボタンを持つメッセージボックスを出力し、メッセージボックスからの復帰値をCALL文のRETURNING指定で受け取ります。さらにその復帰値に対応する値をCOBOLプログラムからの復帰値として、バッチファイルで参照します。

COBOLプログラムの復帰値はバッチファイルではERRORLEVELという名前で参照することができます。以下は“Sample31.BAT”の一部です。

```
:START
    start /w MsgBox.exe
    @rem 復帰値が9999以上ならCOBOLプログラムを再度呼び出す
    if errorlevel 9999 goto START
    @rem 復帰値が9以上なら『いいえ』が押された。
    if errorlevel 9 goto NG
    echo 『はい』が押されました。
```

提供プログラム

- Msgbox.cob（COBOLソースプログラム）
- Makefile（メイクファイル）
- Sample31.bat（起動用バッチファイル）
- COBOL85.CBR（実行用の初期化ファイル）

使用しているCOBOLの機能

- COBOLプログラムからCプログラムを呼び出す方法
- BY VALUEでのパラメタの受渡し
- CALL文のRETURNING指定
- 特殊レジスタPROGRAM-STATUS

注意

- メッセージ本文とメッセージタイトルの文字列の末尾には、X"00"またはLOW-VALUEを格納しなければなりません。文字列を部分参照して、末尾にX"00"またはLOW-VALUEを格納する必要があります。
- メッセージボックスを表示する関数名は、"MessageBoxA"です。小文字を有効にするため、翻訳オプションNOALPHALまたはALPHAL(WORD)を指定する必要があります。

6.19.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

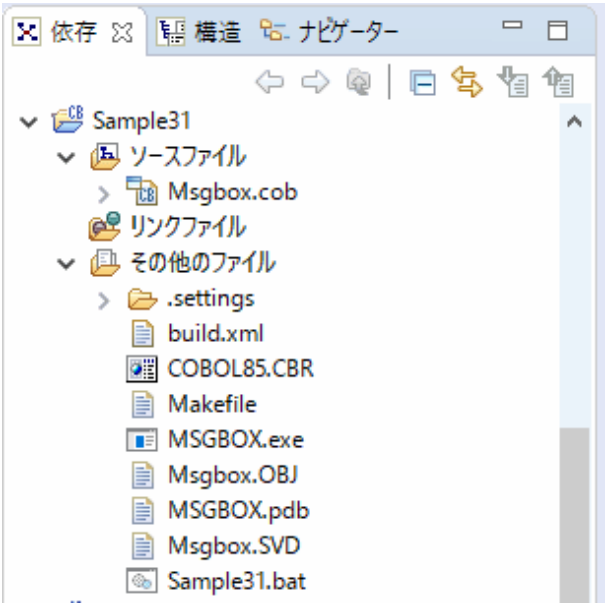
   "ワークスペースを準備する"

2. [依存]ビューを確認し、Sample31プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   "サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする"

3. [依存]ビューからSample31プロジェクトを選択し、以下の構成になっていることを確認します。



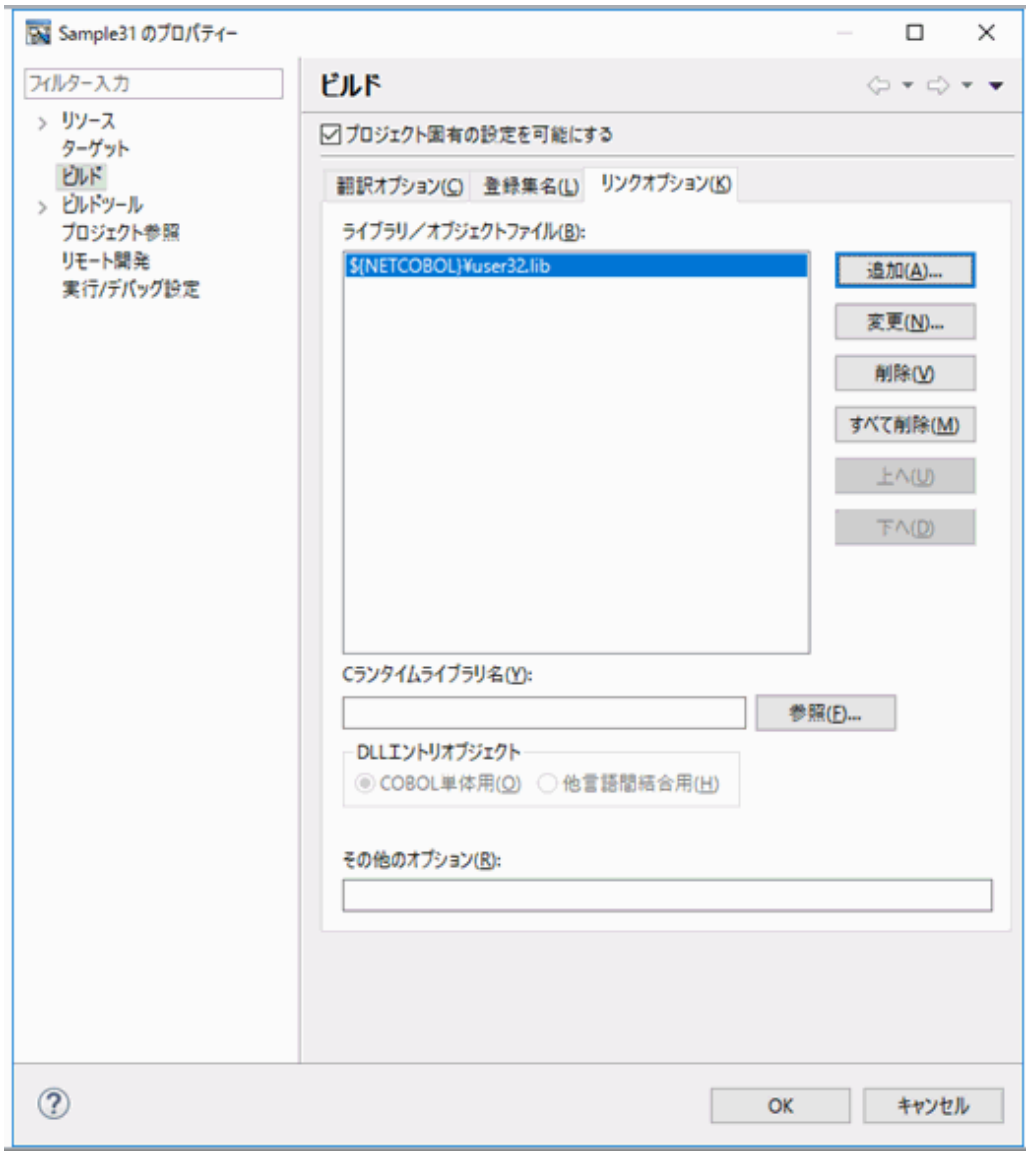
自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. このSampleでは、Windowsシステム関数の“MessageBoxA”を使用するため、USER32.libをリンクします。

   リンクするライブラリを確認するには、NetCOBOL Studioの[依存]ビューからSample31プロジェクトを選択し、コンテキストメニューから[プロパティ]を選択します。

   → [プロパティ]ダイアログボックスが表示されます。

5. 左ペインから[ビルド]を選択すると、[ビルド]ページが表示されます。[リンクオプション]タブを選択して、リンクするライブラリファイル(USER32.lib)を確認します。



USER32.libの格納場所は、以下のように設定しています。Windows SDKのインストール環境に合わせて、設定を変更してください。

```
${NETCOBOL}¥USER32.lib
```

6. [その他のファイル]にMSGBOX.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、MSGBOX.exeが作成されます。

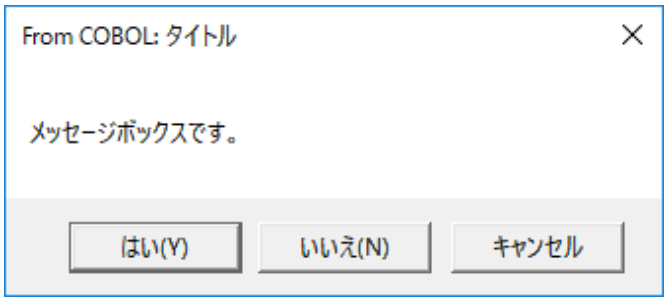
プログラムの実行

コマンドプロンプトを開き、実行可能プログラムと同じフォルダーにあるバッチファイル“Sample31.BAT”を実行します。

```
C:¥NetCOBOL Studio¥workspace¥Sample31>Sample31.BAT
```

実行結果

1. 以下のメッセージボックスが表示されます。[はい]、[いいえ]または[キャンセル]ボタンをクリックします。



2. [はい]または[いいえ]のボタンをクリックすると、どちらのボタンが押されたかがコマンドプロンプトに表示されます。[はい]ボタンをクリックすると、次のように表示されます。

```
C:¥NetCOBOL Studio¥workspace¥Sample31>Sample31.BAT
MessageBox関数の戻り値に従って,値を返します。
『はい』が押されました。

C:¥NetCOBOL Studio¥workspace¥Sample31>
```

3. [キャンセル]ボタンをクリックした場合、実行可能プログラムが再度実行されます。

6.19.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample31>nmake
```

翻訳およびリンク終了後、MSGBOX.exeが作成されていることを確認してください

プログラムの実行

NetCOBOL Studioを利用する場合と同じです。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.20 他のプログラムの起動(Sample32)

ここでは、本製品で提供するサンプルプログラム-Sample32-について説明します。

Sample32では、プログラム間連絡機能を使って、Windowsシステム関数を呼び出し、他のプログラムあるいはバッチファイルを起動して、その終了コードを受け取るプログラムの例を示します。プログラム間連絡機能の詳細は、“NetCOBOL ユーザーズガイド”の“サブプログラムを呼び出す～プログラム間連絡機能～”を参照してください。

注意

以降では、NetCOBOLのインストール先フォルダーをC:¥COBOLとして説明しています。フォルダー名がC:¥COBOLになっているところは、NetCOBOLをインストールしたフォルダーに変更してください。

概要

起動するプログラムあるいはバッチファイルのパス名と、必要ならコマンド文字列を入力します。これを引数に指定してWindowsシステム関数を呼び出し、指定したプログラムあるいはバッチファイルを起動します。また、起動に成功した場合、その実行が終了するまで待って、終了コードを受け取ります。

提供プログラム

- Sample32.cob（COBOLソースプログラム）
- Makefile（メイクファイル）
- COBOL85.CBR（実行用の初期化ファイル）

使用しているCOBOLの機能

- COBOLプログラムからCプログラムを呼び出す方法
- BY VALUEでのパラメタの受渡し
- CALL文のRETURNING指定
- STORED-CHAR-LENGTH関数

プログラムを実行する前に

起動するプログラムとして、以下のSampleプログラムの実行可能ファイルを使用します。

- Sample6のSample6.exe
- Sample31のMSGBOX.exe

6.20.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

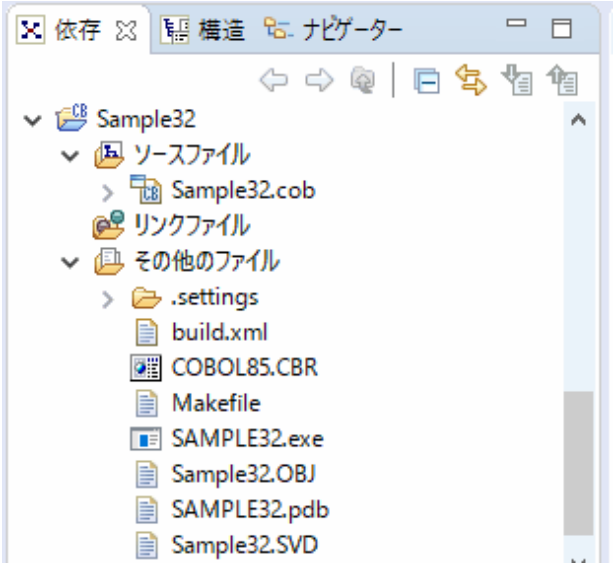
   “ワークスペースを準備する”

2. [依存]ビューを確認し、Sample32プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample32プロジェクトを選択し、以下の構成になっていることを確認します。



自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. [その他のファイル]にSample32.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

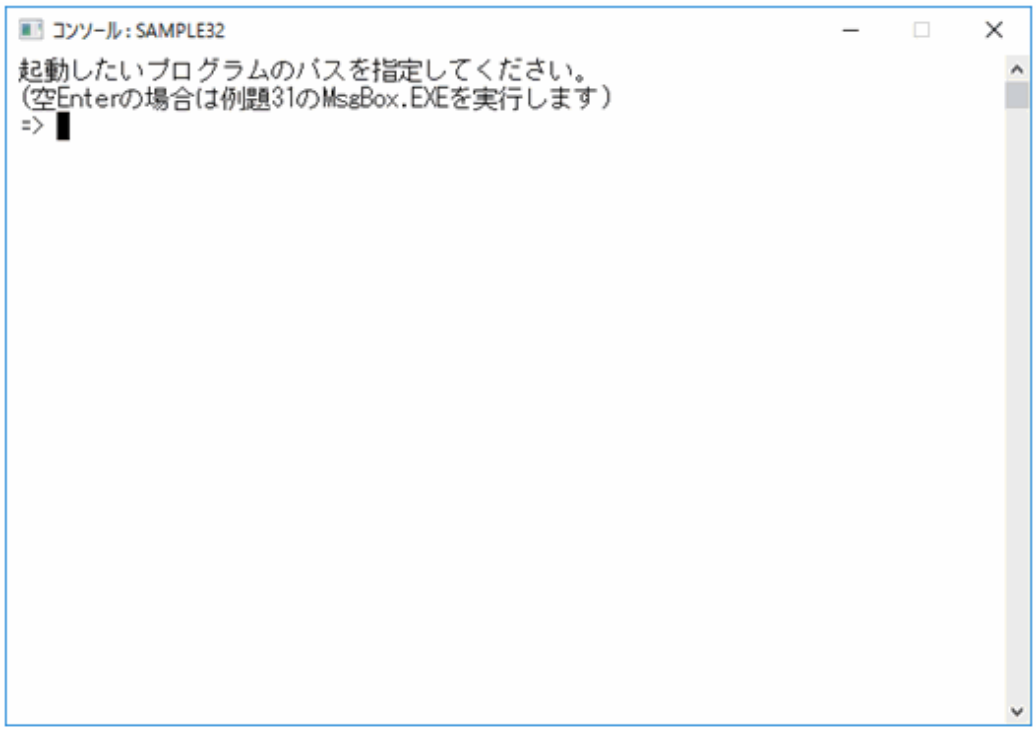
   → プロジェクトのビルドが行われ、Sample32.exeが作成されます。

プログラムの実行

[依存]ビューからSample32プロジェクトを選択し、NetCOBOL Studioのメニューバーから[実行(R)] > [実行(S)] > [COBOLアプリケーション]を選択します。

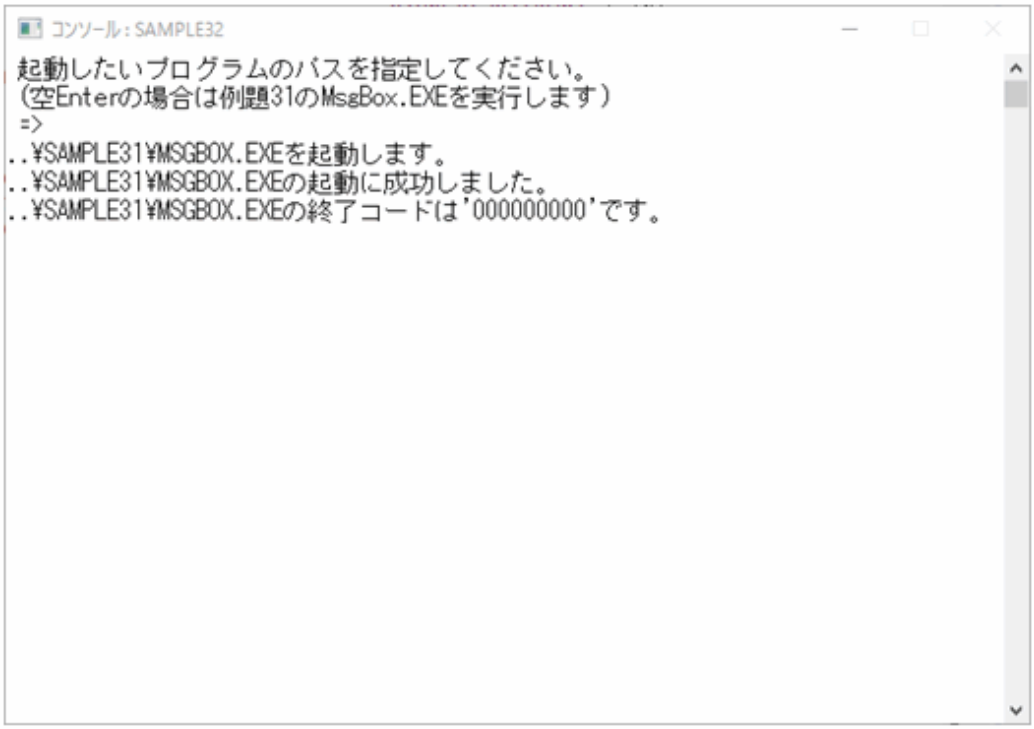
実行結果

1. 次の表示が現れて入力待ちになります。



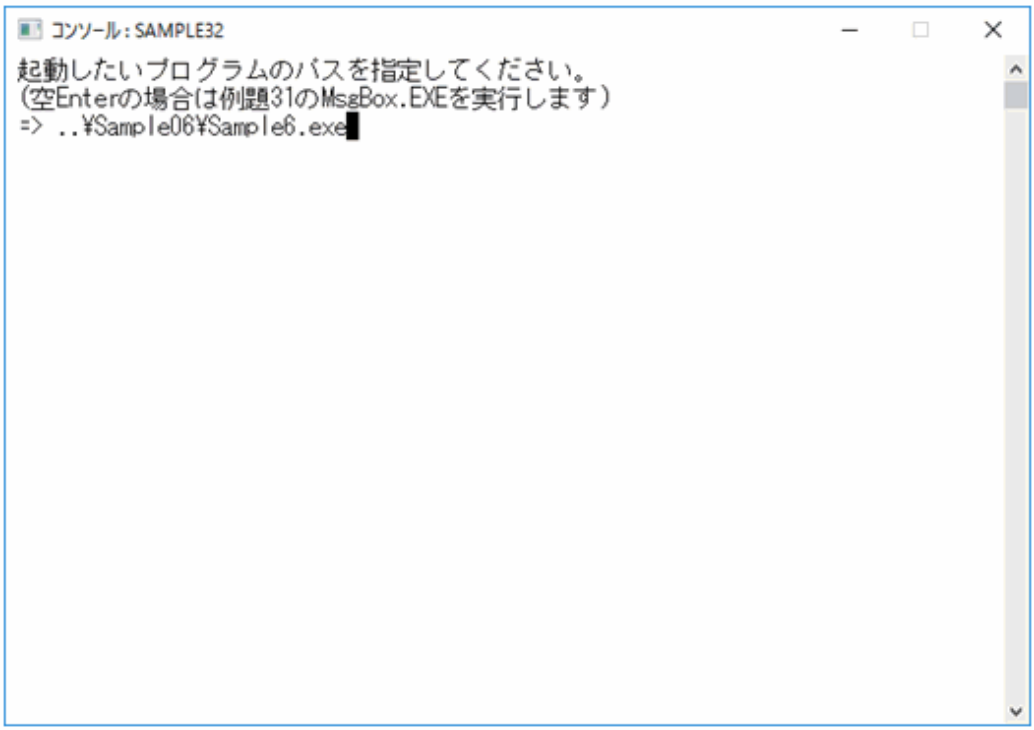
2. 起動するプログラムあるいはバッチファイルのパス名を入力します。ここでは環境変数PATHの指定は無効であるため、絶対パスまたは、Sample32.exeを実行したフォルダーからの相対パスを指定する必要があります。
3. 何も入力しないでENTERキーを押すと、Sample31のMSGBOX.exeを実行します。

4. MSGBOX.exeを起動する旨のメッセージが表示され、Sample31と同様のメッセージボックスが表示されます。メッセージボックスのボタンのどれかをクリックすると、Sample31のMSGBOX.exeの終了コードが示されて、プログラムが終了します。[いいえ]ボタンをクリックした場合、次のように表示されます。

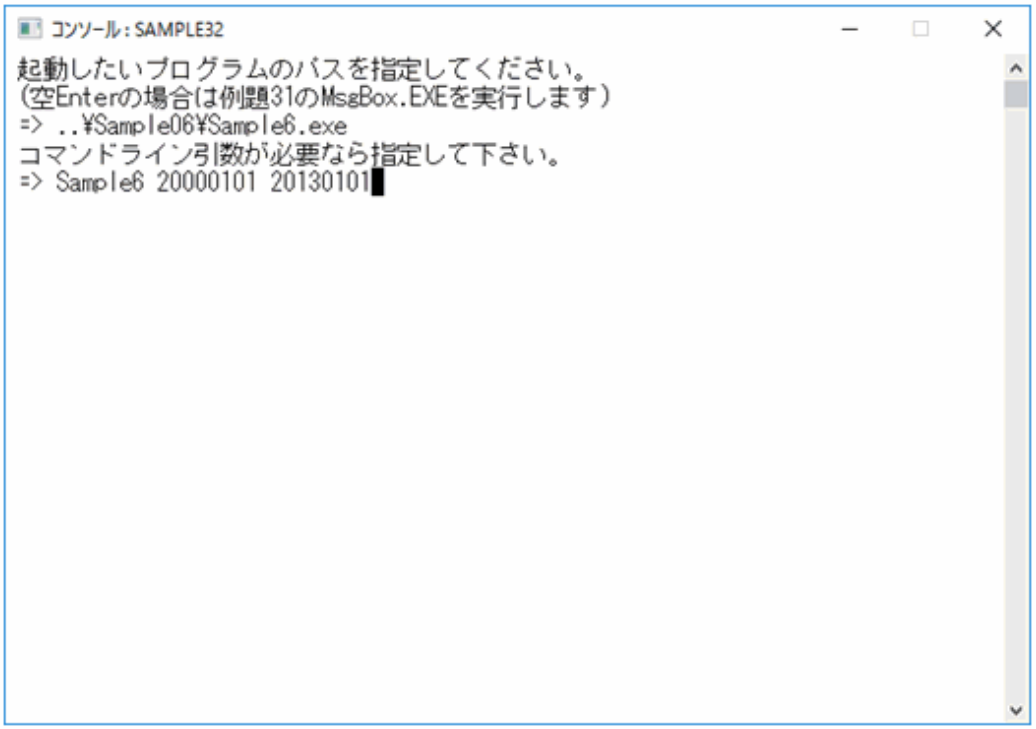


5. 起動するプログラムやバッチファイルのパス名を明示的に指定した場合、コマンド行引数の入力を促すメッセージが表示されて、入力待ちになります。コマンド行引数が必要ならここで入力します。不要な場合は、何も入力しないでENTERキーを押してください。

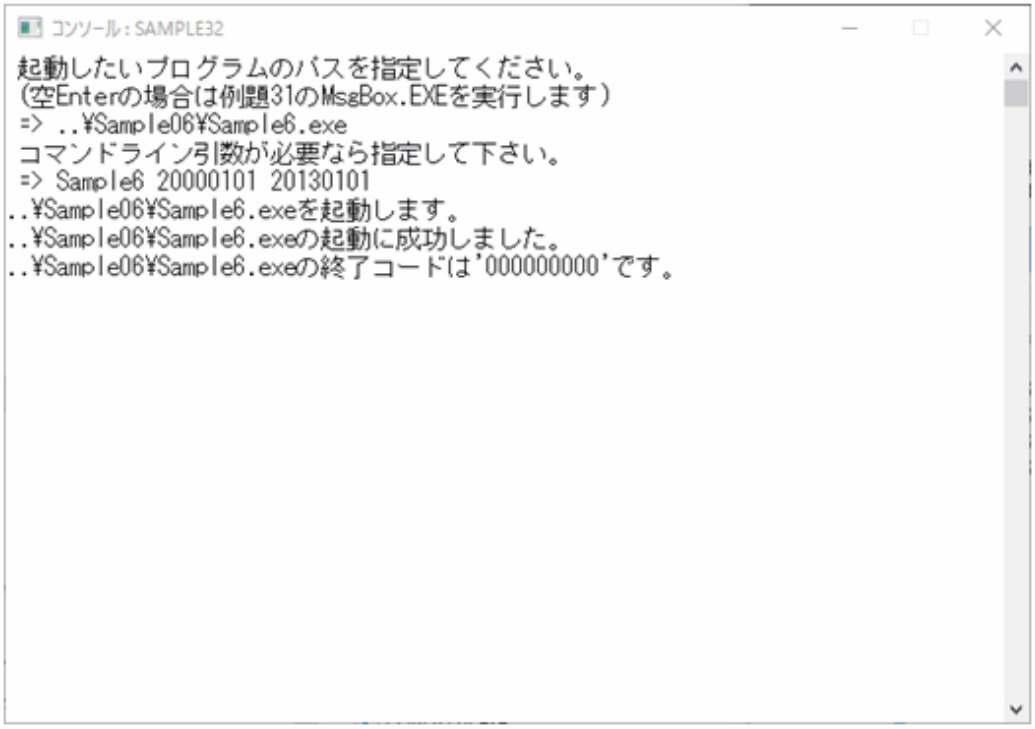
   ここでは、Sample6のSample6.exeを実行します。



6. Sample6のSample6.exeは2つのコマンド行引数が必要なため、ここで指定します。コマンド行引数をプログラム名に続けて指定することに注意してください。



7. Sample6.exeを起動する旨のメッセージが表示され、Sample6.exeが実行されます(システムのコンソールが開かれて実行結果が出力されます)。実行が終了すると、Sample6.exeの終了コードが示されます。



6.20.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample32>nmake
```

翻訳およびリンク終了後、Sample32.exeが作成されていることを確認してください

プログラムの実行

コマンドプロンプトまたはエクスプローラからSample32.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

6.21 エンコード方式を使用するプログラム(Sample33)

ここでは、本製品で提供されているサンプルプログラム-Sample33-について説明します。

Sample33では、UTF-16のファイルレコードを入力し、それらを表示・印刷するプログラムの例を示します。

概要

サロゲートペアの文字が格納されているファイル(UTF-16形式の行順ファイル) のレコードを読み出し、エンコード方式がUTF-16とUTF-32のデータに格納します。

画面には、格納したデータと文字数、バイト数を表示します。また、標準プリンターに、画面に表示したデータの他に、レコード件数を示す数字を出力します。

提供プログラム

- Sample33.cob(COBOLソースプログラム)
- Makefile（メイクファイル）
- INDATA(入力ファイル)
- COBOL85.CBR(実行用の初期化ファイル)

使用しているCOBOLの機能

- プログラム間連絡機能
- 組込み関数機能
- 小入出力機能(コンソールウィンドウ)
- 印刷ファイル
- 行順ファイル(参照)
- 内部プログラム

使用しているCOBOLの文

- CALL文
- ACCEPT文
- DISPLAY文
- PERFORM文
- IF文
- EVALUATE文

- GO TO文
- MOVE文
- COMPUTE文
- OPEN文
- CLOSE文
- READ文
- WRITE文
- EXIT文

プログラムを実行する前に

“通常使うプリンター”の設定を確認してください。

6.21.1 NetCOBOL Studioを利用する場合

プログラムの翻訳・リンク

1. サンプル用に作成したワークスペースを指定して、NetCOBOL Studioを起動します。

   参考

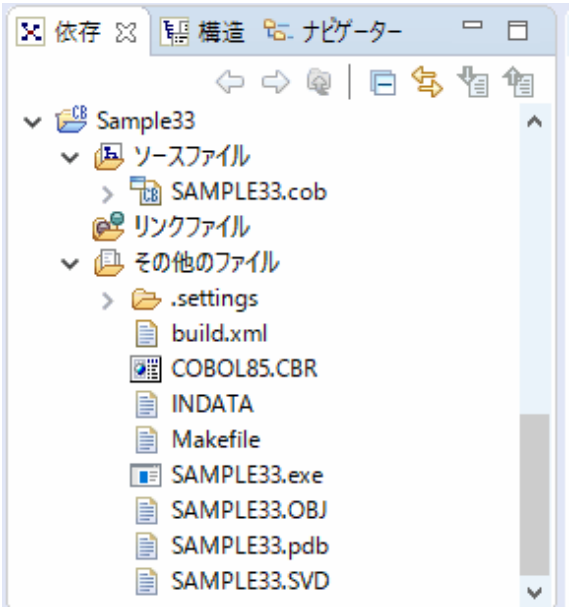
   “ワークスペースを準備する”

2. [依存]ビューを確認し、Sample33プロジェクトがなければ、以下を参考にサンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートします。

   参考

   “サンプルプログラムのプロジェクトをNetCOBOL Studioのワークスペースにインポートする”

3. [依存]ビューからSample33プロジェクトを選択し、以下の構成になっていることを確認します。

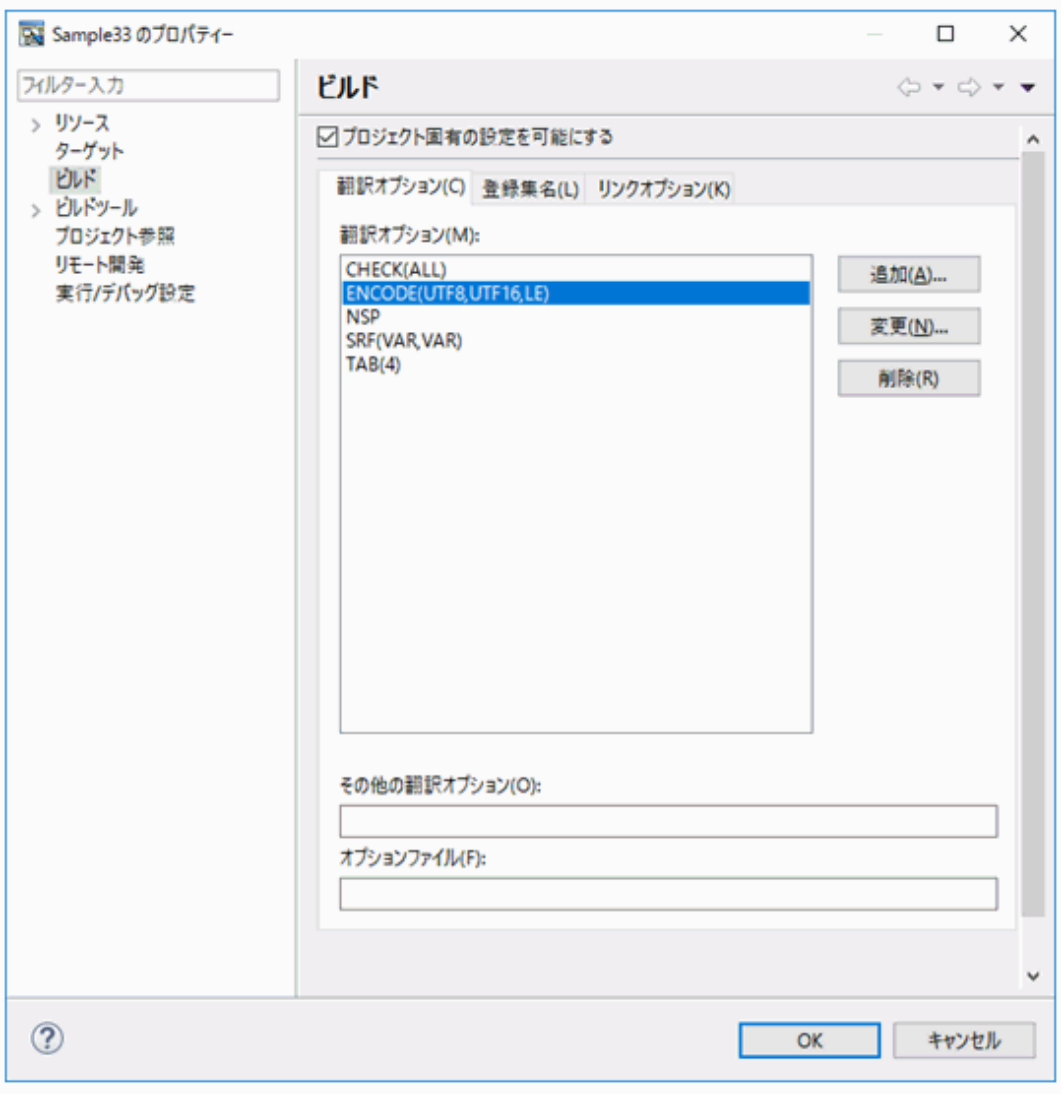


自動ビルドが設定されている場合、プロジェクトをワークスペースにインポートした直後にビルドが実行されます。この場合、[その他のファイル]には、ビルド後に生成されるファイル(.exeや.objなど)が表示されます。既定では自動ビルドに設定されています。

4. Sample33プロジェクトに翻訳オプションENCODE(UTF8,UTF16,LE)が指定されていることを確認します。
   翻訳オプションの設定は、NetCOBOL Studioの[依存]ビューからSample33プロジェクトを選択し、コンテキストメニューから[プロパティ]を選択します。

   → [プロパティ]ダイアログボックスが表示されます。

5. 左ペインから[ビルド]を選択すると、[ビルド]ページが表示されます。[翻訳オプション]タブを選択して、設定オプションの内容を確認します。



   翻訳オプションENCODE(UTF8,UTF16,LE)が指定されていることを確認して、[OK]ボタンをクリックします。

参考

翻訳オプションを設定するための詳細な手順は、“NetCOBOL Studio ユーザーズガイド”の“翻訳オプションの設定”を参照してください。

6. [その他のファイル]にSample33.exeが作成されていない場合(自動ビルドが実行されていない場合)、NetCOBOL Studioのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

   → プロジェクトのビルドが行われ、Sample33.exeが作成されます。

プログラムの実行

［依存］ビューからSample33プロジェクトを選択し、NetCOBOL Studioのメニューバーから［実行(R)］>［実行(S)］>［COBOLアプリケーション］を選択します。

実行結果

画面には、格納したデータと文字数、バイト数が表示されます。また、標準プリンターに、画面に表示したデータの他に、レコード件数を示す数字が出力されます。

6.21.2 MAKEファイルを利用する場合

プログラムの翻訳・リンク

NetCOBOLコマンドプロンプトから以下のコマンドを実行し、翻訳およびリンクを行います。

```
C:¥COBOL¥Samples¥COBOL¥Sample33>nmake
```

翻訳およびリンク終了後、Sample33.exeが作成されていることを確認してください

プログラムの実行

コマンドプロンプトまたはエクスプローラからSample33.exeを実行します。

実行結果

NetCOBOL Studioを利用する場合と同じです。

第7章 COBOLファイルアクセスルーチンのサンプルプログラム

NetCOBOLでは、以下のプログラムをファイルアクセスルーチンのサンプルとして提供しています。

- 7.1 行順ファイルの読込み(FCFA01)
- 7.2 行順ファイルの読込みと索引ファイルの書出し(FCFA02)
- 7.3 索引ファイルの情報の取得(FCFA03)

7.1 行順ファイルの読込み(FCFA01)

概要

このサンプルプログラムは、ファイルアクセスルーチンを使用します。指定したファイルを行順ファイルとしてINPUTモードでオープンし、読み込んだレコードの内容を表示します。

提供プログラム

- fcfa01.c (Cソースプログラム)
- fcfa01.mak (MAKEファイル)
- fcfa01.txt (プログラム説明書)

使用しているCOBOLファイルアクセスルーチンの関数

- cobfa_open()関数
- cobfa_rdnext()関数
- cobfa_stat()関数
- cobfa_errno()関数
- cobfa_reclen()関数
- cobfa_close()関数

プログラムの翻訳とリンク

MAKEファイルの“CDIR =”と書かれている行の右側の内容が、Cコンパイラのインストールフォルダー名となるように修正してください。

また、“COBDIR =”と書かれている行の右側の内容が、NetCOBOLランタイムシステムのインストールフォルダー名となるように修正してください。

MAKEファイルを修正した後、以下のコマンドを入力します。

```
> nmake -f fcfa01.mak
```

プログラムの実行

適当なテキストファイルをコマンドライン引数にしてプログラムを実行します。ここではfcfa01自身のソースプログラムを入力します。

```
> fcfa01 fcfa01.c
```

格納フォルダー

C:¥COBOL¥SampleS¥FCFA01

(*C:¥COBOL*は、NetCOBOLをインストールしたフォルダーです)

7.2 行順ファイルの読込みと索引ファイルの書出し(FCFA02)

概要

このサンプルプログラムは、ファイルアクセスルーチンを使用します。

特定の行順ファイル(fcfa02.inp)をINPUTモードでオープンし、そのレコードの内容を索引ファイル(fcfa02.idx)のレコードとして書き出します。最後に、その索引ファイルをINPUTモードでオープンし、主キーの順で画面に表示します。

提供プログラム

- fcfa02.c (Cソースプログラム)
- fcfa02.inp (入力用行順ファイル)
- fcfa02.mak (MAKEファイル)
- fcfa02.txt (プログラム説明書)

使用しているCOBOLファイルアクセスルーチンの関数

- cobfa_open()関数
- cobfa_rdnext()関数
- cobfa_wrkey()関数
- cobfa_stkey()関数
- cobfa_close()関数

プログラムの翻訳とリンク

MAKEファイルの“CDIR =”と書かれている行の右側の内容が、Cコンパイラのインストールフォルダー名となるように修正してください。

また、“COBDIR =”と書かれている行の右側の内容が、NetCOBOLランタイムシステムのインストールフォルダー名となるように修正してください。

MAKEファイルを修正した後、以下のコマンドを入力します。

```
> nmake -f fcfa02.mak
```

プログラムの実行

コマンドライン引数を付けずに実行します。

```
> fcfa02
```

格納フォルダー

C:¥COBOL¥SampleS¥FCFA02

(*C:¥COBOL*は、NetCOBOLをインストールしたフォルダーです)

7.3 索引ファイルの情報の取得(FCFA03)

概要

このサンプルプログラムは、ファイルアクセスルーチンを使用します。指定したファイルを索引ファイルとしてINPUTモードでオープンし、ファイル自体の属性と、レコードキーの各構成を表示します。

提供プログラム

- fcfa03.c (Cソースプログラム)
- fcfa03.mak (MAKEファイル)
- fcfa03.txt (プログラム説明書)

使用しているCOBOLファイルアクセスルーチンの関数

- cobfa_open()関数
- cobfa_indexinfo()関数
- cobfa_stat()関数
- cobfa_errno()関数
- cobfa_close()関数

プログラムの翻訳とリンク

MAKEファイルの“CDIR =”と書かれている行の右側の内容が、Cコンパイラのインストールフォルダー名となるように修正してください。

また、“COBDIR =”と書かれている行の右側の内容が、NetCOBOLランタイムシステムのインストールフォルダー名となるように修正してください。

MAKEファイルを修正した後、以下のコマンドを入力します。

```
> nmake -f fcfa03.mak
```

プログラムの実行

適当な索引ファイルをコマンドライン引数にしてプログラムを実行します。ここではfcfa02を実行して生成した索引ファイルを指定します。

```
> fcfa03 ..¥fcfa02¥fcfa02.idx
```

格納フォルダー

C:¥COBOL¥SampleS¥FCFA03

(*C:¥COBOL*は、NetCOBOLをインストールしたフォルダーです)

索　引

[は]

[ま]

[ら]

[わ]