B1WW-9591-01Z0(00)

Microsoft Windows 2000
Microsoft Windows XP
Microsoft Windows Server 2003

NetCOBOL V9.0

OSIV分散開発の手引き

NetCOBOL

FUJITSU

まえがき

NetCOBOLでは、以下に示すシステムでOSIV系システムで動作するアプリケーションを開発するための開発環境を提供します。

- Microsoft(R) Windows(R) 2000 Professional operating system
- Microsoft(R) Windows(R) 2000 Server operating system
- Microsoft(R) Windows(R) 2000 Advanced Server operating system
- Microsoft(R) Windows(R) XP Professional operating system
- Microsoft(R) Windows(R) XP Home Edition operating system
- Microsoft(R) Windows Server(R) 2003, Standard Edition
- Microsoft(R) Windows Server(R) 2003, Enterprise Edition

製品の呼び名について

本書では以下の製品の名称を、『Windows系システム』と略して表記します。

- 「Microsoft(R) Windows(R) 2000 Professional operating system」
- 「Microsoft(R) Windows(R) 2000 Server operating system」
- 「Microsoft(R) Windows(R) 2000 Advanced Server operating system」
- 「Microsoft(R) Windows(R) XP Professional operating system」
- 「Microsoft(R) Windows(R) XP Home Edition operating system」
- 「Microsoft(R) Windows Server(R) 2003, Standard Edition」
- 「Microsoft(R) Windows Server(R) 2003, Enterprise Edition」

本書では以下の製品の名称を、『Solaris』と略して表記します。

- 「Solaris™ 8 オペレーティングシステム」
- 「Solaris™ 9 オペレーティングシステム」
- 「Solaris™ 10 オペレーティングシステム」

本書の目的

本書は、NetCOBOLを利用して、OSIV系システムで動作するCOBOLプログラムを分散開発する方法について説明しています。
COBOLの文法規則については、“COBOL文法書”をお読みください。

本書の対象読者

本書は、NetCOBOLを利用してOSIV系システムの動作するCOBOLプログラムを開発される方を対象としています。

前提知識

本書を読むにあたって、以下の知識が必要です。

- COBOLの文法に関する基本的な知識
- OSIVでのCOBOLプログラム開発に関する基本的な知識
- Windows系システムに関する基本的な知識

本書の構成

本書の構成と内容は、以下のとおりです。

第1章 分散開発の概要

NetCOBOLを使用した分散開発における作業の流れとその適用範囲について説明しています。

第2章 分散開発環境の構築

NetCOBOLを使用した分散開発における開発環境構築の考え方と環境設定の方法を説明しています。

第3章 開発作業(プログラミング)

NetCOBOLのプログラミング作業の手順を説明しています。また、OSIV系システムからプログラミング資産をWindows系システムに移行する手順についてもここで説明します。

第4章 単体テスト

NetCOBOLで開発したOSIV系システム用のCOBOLプログラムを、プログラム単位でデバッグする方法について説明しています。

第5章 サーバ連携機能

OSIV系システムと連携し、NetCOBOLからOSIV系システムへの資産の登録、OSIV系システムでのプログラムの翻訳・リンクを実行する方法について説明しています。

第6章 CORBAアプリケーションの分散開発

OSIV系システムで動作するCORBAアプリケーションに固有な分散開発の手順を説明しています。

第7章 トラブルシューティング

分散開発時に起こりやすい問題とその回避方法について説明しています。

付録A OSIV系COBOLとオープン系COBOLの相違点

OSIV系のCOBOL(COBOL85)とオープン系のCOBOL(NetCOBOL)の言語の機能の違い等について説明しています。

付録B 定義体移行時の留意点

ソース・登録集以外のプログラミング資産である各種定義体移行上の留意点について説明しています。

付録C COBOL85非互換指摘機能

分散開発を支援するためのCOBOL85非互換指摘機能について説明しています。

付録D NetCOBOL JEFオプション

Windows系システム上で、EBCDIC/JEFのデータを操作するためのNetCOBOL JEFオプションについて製品の概要と留意事項を説明しています。

付録E GETSSCH診断メッセージ一覧

分散開発支援用にNetCOBOLで提供しているツールGETSSCHの診断メッセージについて説明しています。

付録F 文字コード系

文字コード系の基礎的な知識とコード系の違いがCOBOLプログラミングに与える影響について説明しています。

本書の位置付け

NetCOBOLおよび関連製品のマニュアルには、本書のほかに以下のマニュアルがあります。

マニュアル名称	内　　容
COBOL文法書	COBOL の文法規則の詳細な説明（オープン系）

NetCOBOL使用手引書	NetCOBOLを利用したCOBOLプログラムの作成、実行およびデバッグの方法の説明
COBOL 文法書 V12用	OSIV系システムのCOBOL の文法規則の詳細な説明
FUJITSU COBOL85 文法書　システム拡張編	OSIV系システムのCOBOL85 固有機能の説明
OSIV COBOL85 メッセージ説明書	OSIV系システムで出力されるメッセージの説明

本書の説明に使用している関連製品について、詳細な情報を知りたい方は、以下のマニュアルまたはヘルプをお読みください。

マニュアル名称（注1）	製品名	使用目的
FORM V9.0 説明書 FORM V9.0 ヘルプ（注2） PowerFORM V9.0 ヘルプ（注2）	FORM V9.0 FORMオーバレイオプション（オプション製品）	画面帳票定義体およびフォームオーバレイパターンの作成
MeFt V9.0 説明書(注2)	MeFt V9.0	画面帳票定義体を使用したプログラムの実行
PowerSORT Workstation V5.0 使用手引書 PowerSORT Server V5.0 使用手引書	PowerSORT Workstation V5.0 PowerSORT Server V5.0	PowerSORTを使用した整列併合を行うプログラムの実行

注1

マニュアル名称は、製品の適応機種およびバージョンレベルによって異なります。なお、本文中では、バージョンレベルは記載されていません。

注2

ヘルプまたはオンラインマニュアルは、各製品の中にあります。

注意

“ソフトウェア説明書”で組み合わせが可能とされている旧バージョンレベル製品については、旧バージョンレベルのマニュアルをお読みください。

本書で使用する書体と記号

書体および記号	意　　味
［参照］	参照先を示します。
→	操作結果を示します。
あいうえお	プログラム例中で、可変文字列を示します。可変文字列は、実際には他の文字列に置き換えます。 例： PROGRAM-ID.　プログラム名. 　　→　PROGRAM-ID.　SAMPLE1.
{あい うえお} または {あい｜うえお}	{ } で囲まれた文字列の１つを選択することを示します。省略した場合、“＿”（アンダーライン）の文字列が選択されたものとして扱われます。
[あいうえお]	[ ] で囲まれた文字列は省略できることを示します。

その他の注意事項

- 本書に記載されている画面は、Windows 2000で採取したものです。
- 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。
- OSIV/MSP、OSIV/XSPなどのOSIV系システムを総称して、“OSIV系システム”と記述しています。
- SystemWalker/CharsetMGRおよびInterstage Charset Managerを“CharsetMGR”と記述しています。

登録商標について

本書に記載されている登録商標を、以下に示します。

Microsoft、Windowsは、米国Microsoft Corporationの米国およびその他の国における商標または登録商標です。

Sun、Sun Microsystems、Sunロゴ、SolarisおよびすべてのSolarisに関連する商標及びロゴは、米国およびその他の国における米国Sun Microsystems, Inc.の商標または登録商標です。

2006年12月

目次

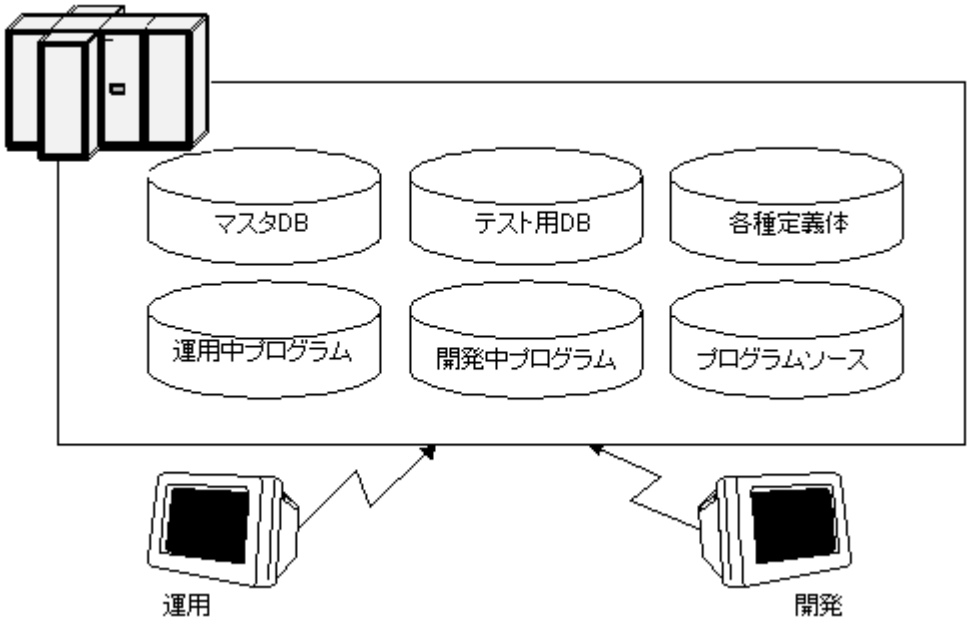
第1章 分散開発の概要

NetCOBOLは、SIA文法（SIA文法については、“FUJITSU COBOL文法書”参照)を採用しています。したがって、SIAの仕様にそってプログラムを記述することにより、各種システムで動作するプログラムをWindows系システムで分散開発することができます。
本章では、特にOSIV系システムのアプリケーションの開発をWindows系システム上で行う場合の分散開発についての概要を説明します。OSIV系システムのアプリケーションの開発にあたっては、本書とあわせて“FUJITSU COBOL 文法書”および“OSIV COBOL85 使用手引書”をお読みください。

1.1 分散開発とは？

かつて、システムやアプリケーションは、それを最終的に運用するマシン/オペレーティングシステムの上で開発することが普通でした。そして、ソースや各種定義体などのプログラム資産も同じマシン上で、一極集中で管理されていました。

図1-1 一極集中型の開発形態



このように、同じマシン/オペレーティングシステム上に運用環境が存在することから、開発作業に制約が加わることも少なくありませんでした。それでも、このような一極集中型の開発形態が一般的であったのは、半ば慣習的な面もありますが、技術上の制約から他に選択のしようがないと言う面もありました。

しかし、ネットワーク技術や各種の開発技術の進歩は、従来の技術上の制約を取り払い、開発に特化した環境を運用環境と別に構築し、なおかつ、それを容易に運用環境に持ち込むことを可能としました。このようにして、生まれた開発手法を分散開発と呼びます。

分散開発は、このように広い意味を持つ言葉であるため、その目的や構成する技術によって、さまざまな形態を取りますが、このマニュアルでは、NetCOBOLを使用して、OSIV系システムで動作するプログラム(以降、OSIV系システムで動作するプログラムのことをOSIV系プログラムといいます)をPC上で開発する開発形態のみを扱います。

用語の定義

OSIV系システムについてなじみのない方に簡単に用語の定義を示します。

OSIV系システム:
OSIV/MSP、OSIV/XSPなどグローバルサーバまたはPRIMEFORCEで動作するOSIV系のシステムの総称

OSIV系プログラム:
OSIV系システムで動作するプログラム

OSIV系システム固有機能:
OSIV系システムでのみ利用できる機能

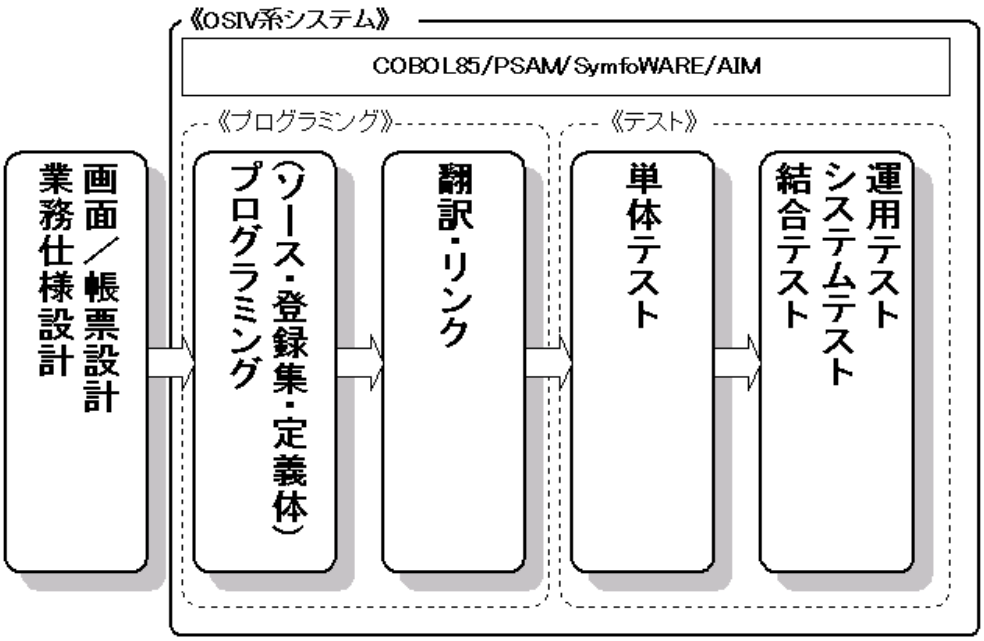
また、Windows系システムとは、MicrosoftのWindowsオペレーティングシステム全般を指す場合に用います。

1.2 OSIV系プログラムの分散開発の全体像

1.2.1 分散開発の作業の流れ

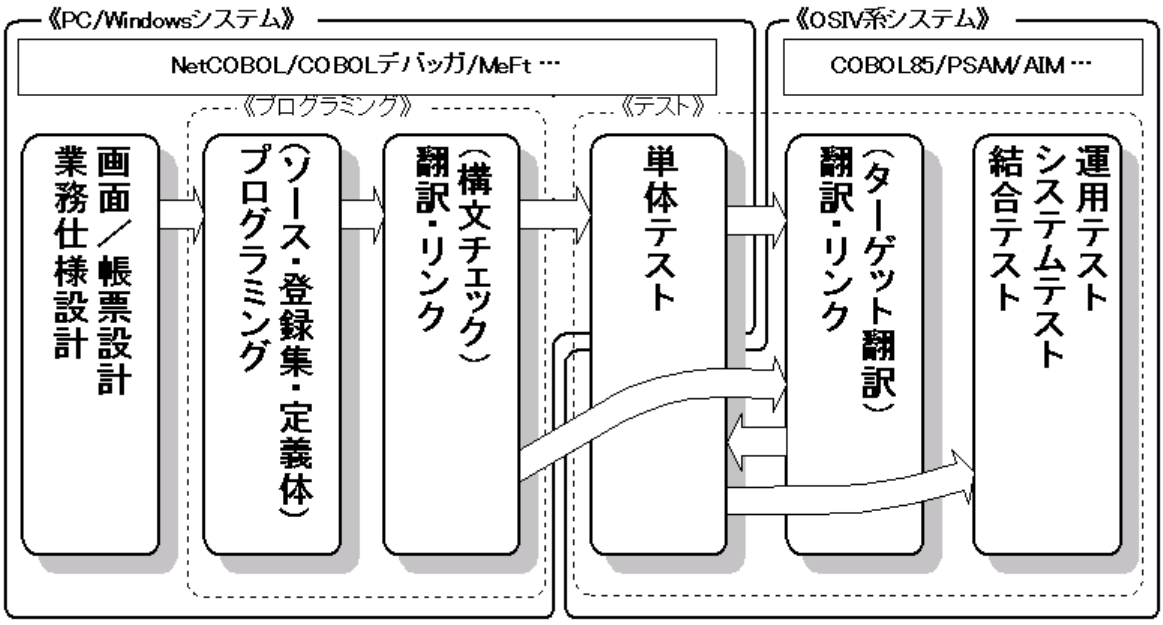
これまでのOSIV系プログラムの開発は、“図1-2 従来のOSIV系プログラムの開発作業の流れ”で示すように、開発からテストに至るすべての作業をOSIV系システム上で行っていました。

図1-2 従来のOSIV系プログラムの開発作業の流れ



一方、分散開発では、“図1-3 OSIV系プログラムの分散開発作業の流れ”に示すように、Windows系システムで実施する作業とOSIV系システムで実施する作業の2つに大きくわかれます。

図1-3 OSIV系プログラムの分散開発作業の流れ



このようにプログラミングから単体テストまでをWindows系システムで実施し、OSIV系システムでターゲット翻訳以降の作業を実施するというのが、分散開発の基本的な作業です。

- プログラミング(ソース、登録集、定義体)
  COBOLソースを始めとする各種プログラム資産をPC上で作成・更新します。
  - COBOLソースプログラム
  - COBOL登録集原文(COPY句)
  - 画面帳票定義体
  - オーバレイ定義体

  開発の目的が、新規開発ではなく、既存のプログラムのUP開発である場合、OSIVシステムから、これらの資産をWindows系システムに移行して、それを修正することになります。

- 翻訳・リンク(構文チェック)
  NetCOBOLを使用して、Windows系システムで作成・更新したプログラム資源を翻訳・リンクします。この作業の第一の目的は、作成したプログラム資源に誤りや矛盾がないことをまず確認することです。

- 単体テスト
  Windows系システムで翻訳・リンクしたプログラムを使用して、そのプログラムに閉じた範囲の機能をテストします。

- 翻訳・リンク(ターゲット翻訳)
  Windows系システムで翻訳･リンクしたプログラムは､OSIV系システムでは動作しません。このため、Windows系システムで作成・更新したプログラム資産をOSIV系システムに移行して、OSIV系システムのCOBOL85を用いて、改めて翻訳・リンクします。

しかし、実際の分散開発でどの作業工程までをWindows系システムで実施できるかは、開発対象のOSIV系プログラムに依存します。たとえば、ビジネスロジックの実装に特化したプログラムのように、OSIV系システムのシステム固有機能を使用しないプログラムの場合、そのプログラム単体の動作確認までをWindows系システム上で行うことができます。一方、データ入出力処理や通信処理などでOSIV系システムのシステム固有機能を使用しているプログラムの場合、Windows系システム上では構文チェック程度の確認しかできない場合もあります。
個々の機能の注意事項や制限については、“付録A OSIV系COBOLとオープン系COBOLの相違点”を

参照してください。

1.2.2 分散開発のメリットとデメリット

OSIV系プログラムをWindows系システムのNetCOBOLを使用しての分散開発には、いくつかの強力なメリットがある反面、多くの細々としたデメリットもあります。
このメリットとデメリットを理解し、そのメリットを最大限に活かすことができたなら、高い生産性と品質を得ることが可能になります。

メリット

- 安価に開発環境を構築、維持
  ハードウェア/ソフトウェアの購入から維持・管理に至るまでのあらゆる費用的な側面でOSIV系システムと、Windows系システムでは比較になりません。既存のOSIV系システムとは独立の開発に特化して環境を構築するのであれば、Windows系システムでの構築が容易かつ安価です。
  また、OSIV系システムでは、システムを維持・運用していくために、開発要員とは別個に要員が必要となる場合が多いですが、Windows系システムを用いての開発では、そのような要員は不要となります。
- プログラム毎に独立したテスト環境の構築が容易
  OSIV系システムを使用しての一極集中型の開発では、AIM環境とその配下のデータベース等の資源を共有するため、これらの資源を使用する複数のプログラムのテストを同時に行うことが困難となります。作業グループ毎にテスト環境を分割する、あるいはスケジュールを調整して対応してゆくなどのことも可能ですが、個々の開発者に独立したテスト環境を準備するようなことは現実的でありません。
  これに対して、分散開発では各作業者の使用するPC毎に独立したテスト環境を容易に構築できるので、真の作業の並列化が可能となります。これは開発効率の向上に役立ちます。
  また、テスト期間中のアクシデント(たとえば、品質の十分でないプログラムが引き起こしたテストデータの破壊など)を局所化する働きもします。
- 優れたデバッグ・テスト機能
  OSIV系システムは、作成したプログラムのデバッグという点については貧弱な機能しか提供してきませんでした。
  これに対して、Windows系システムのNetCOBOLでは、プログラムのデバッグ・テストの機能が充実しています。特にCOBOLデバッガは実行中のプログラムのソースを表示しながら、対話的にプログラムのデバッグを可能とするため、プログラムの品質を早期に向上させることができます。

デメリット

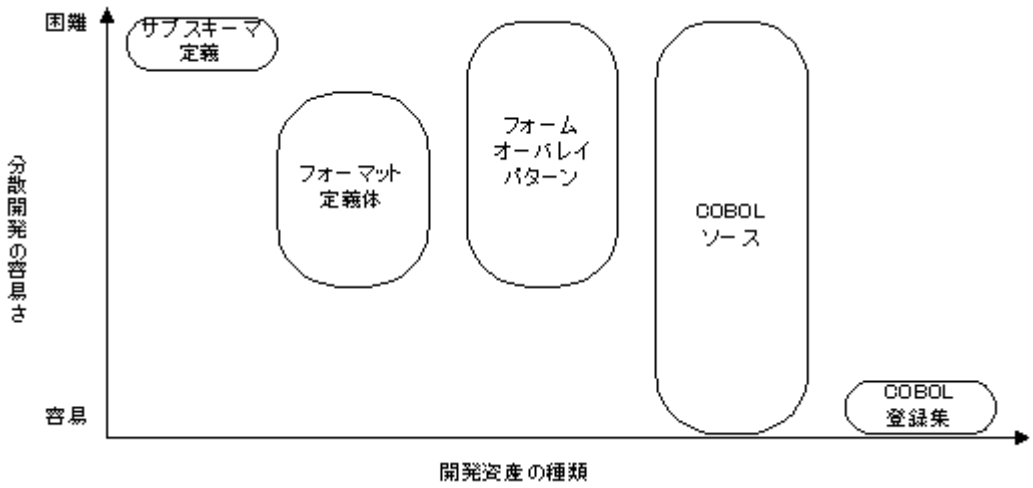
- 開発プラットフォームの違い
  OSIV系システムとWindows系システムでは、その基礎的な概念からオペレーティングシステムとしての機能まで、実にさまざまな範囲で細かな違いがあります。
  NetCOBOLはこれらの違いを意識しないための仕組みを幾つか用意していますが、すべてをカバーできるものではありません。
  この種の問題で最大のものが、文字型のデータを表現するための形式(コード系)です。
- 開発系製品の持つ機能差
  “1.2.3.2 COBOLの機能範囲から見た分散開発の適用範囲”で説明する通り、OSIV系のCOBOL85とNetCOBOLでは機能差が存在します。また、COBOL以外の開発製品(たとえば、画面や帳票をデザインする機能を提供するPSAMとFORM/PowerFORM)にもいくつもの機能差が存在します。分散開発を行う場合、この機能差を理解することが不可欠となります。
- AIMに相当する製品の不在
  OSIV系プログラムのかなりの割合を占めるオンライン系のプログラムに必須であるAIMに相当する製品は、Windows系システム上には存在しません。分散開発を行う上では、このAIMに相当する製品の不在をどのように埋め合わせするかが、大きな鍵となっています。
- 資産の管理・流通の複雑化

OSIV系システムという単一のシステムの上でのみ、開発資産を流通・管理させてゆけばよかった、一極集中型の開発に比べて、分散開発では開発した各種資源の流通・管理が一層複雑になります。

1.2.3 分散開発の適用範囲

既に説明したような分散開発のメリットを最大限に活かし、そのデメリットを最小限にするためには、分散開発を適用する範囲の選択が重要です。
開発対象であるOSIV系プログラムを構築するすべての開発資産をWindows系システムで開発し、単体テストまで完了させるということは分散開発の理想です。しかし、開発対象のOSIV系プログラムによっては、それが不可能であったり、あるいは生産性を却って悪くするものであったりします。
開発資産の種類によって、分散開発の難易度は異なります。以下に、開発資産の種類と分散開発の簡易度の関係を図示します。

図1-4 開発資産と分散開発の適用の難易度



更にCOBOLソースプログラムの分散開発の難易度は、ソース中で使用しているCOBOLの言語の機能に強く依存します。
このため、あらゆるCOBOLの機能を詰め込んだたった1つのソースプログラムからなるOSIV系プログラムの開発を分散開発することは困難になりますが、機能単位などで分割された複数のソースプログラムからなるOSIV系プログラムの分散開発はそれより容易です。
また、この難易度は同じプログラムであっても作業の工程によっても違ってきます。そこで、開発対象とするOSIV系プログラムを構成する資産を次のように分類しておくことが必要になります。

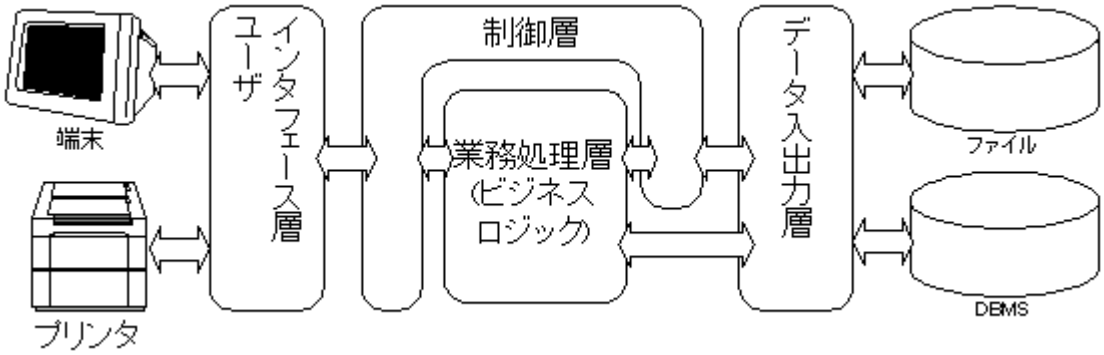
- 分散開発を適用する資産
  - 作成・更新から翻訳チェックまで適用
  - 作成・更新から単体テストまで適用
- 分散開発の対象外とする資産

以下、COBOLソースについて分散開発の適用範囲を決めるための考え方を示します。その他の資産については、付録で示すOSIV系システムとWindows系システムの機能差などを参考にしてください。

1.2.3.1 アプリケーションの構成からみた分散開発の適用範囲

以下、“図1-5 OSIV系アプリレーションの基本的な構成”に従って、各機能範囲に対する分散開発を適用が適切かどうかを説明します。

図1-5 OSIV系アプリレーションの基本的な構成



注意

上の図では、説明の都合からアプリケーションを4つの部分に分けていますが、実際は1つのプログラムである場合もあります。

ユーザインタフェース層

PFDの対話管理機能を用いるようなプログラムを除き、分散開発を適用することが可能です。ただし、画面入出力機能も印刷機能もOSIV系のプログラムとWindows系のプログラムでは詳細な機能仕様が異なります。このため、Windows系システムで動作を完全に確認したプログラムでも、OSIV系システムで動作させると期待した結果が得られない場合が少なくありません。Windows系システム上で単体テストまで実施する場合、最終的な入出力結果を確認するというのではなく、それを作成するまでのプログラムのロジックを確認するなどの割り切りが必要です。

制御層

AIM等のOSIV系のミドルウェアと連携を必要としないものであれば、分散開発も適用可能です。他のミドルウェアとの連携が必要なものである場合、基本的に翻訳チェックまでしか分散開発では行うことはできません。

業務処理層

分散開発の適用にもっとも適した部分です。通常、Windows系システムで単体テストまで可能です。

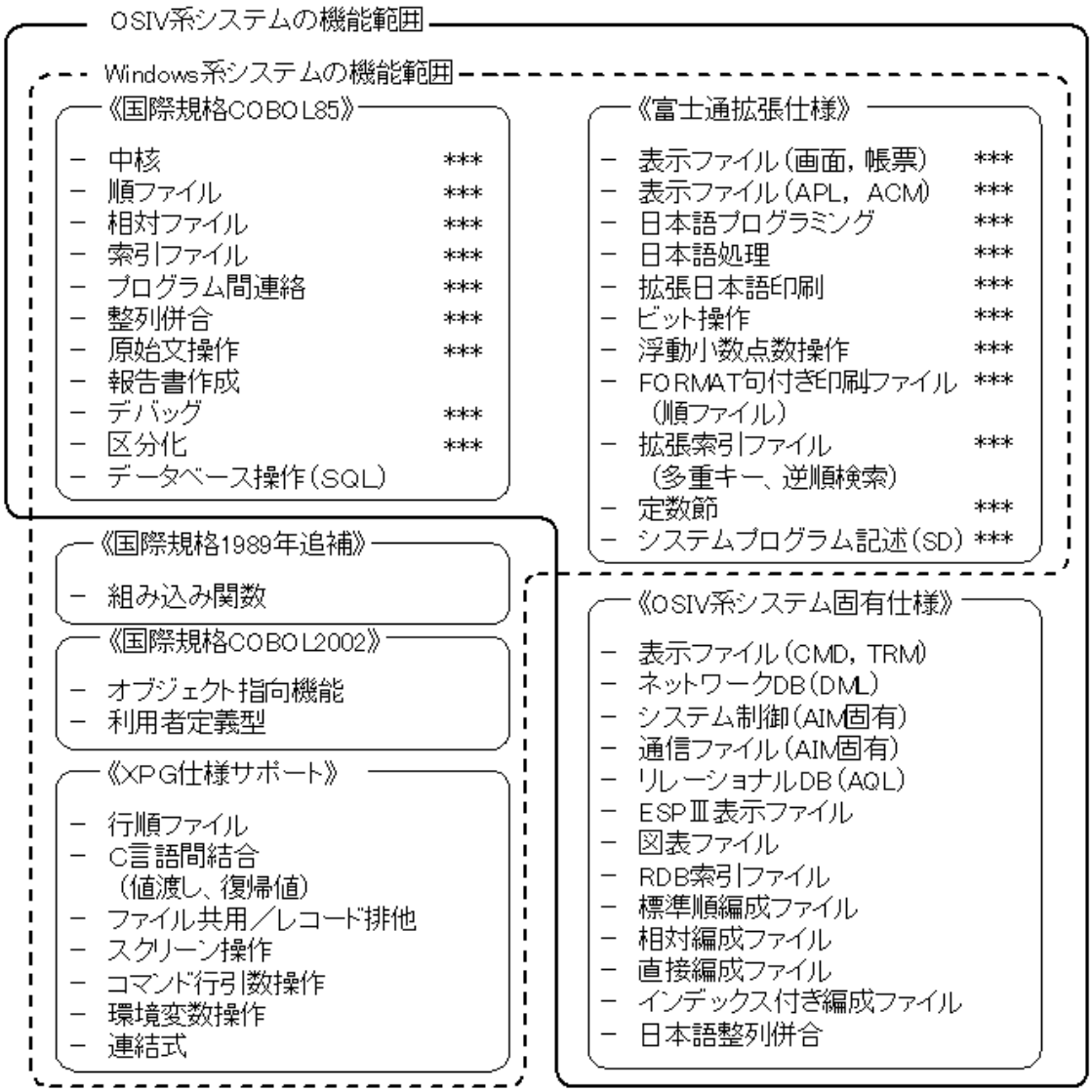
データ入出力層

使用する機能により、分散開発の適用可能かどうか、大きく異なります。共通範囲に含まれているファイル入出力機能を指定しているのであれば、Windows系システム上で、プログラムの動作の確認まで可能ですが、一部のOSIV系システム固有のファイル入出力機能を使用している場合、翻訳チェックさえも正しく行うことができません。

1.2.3.2 COBOLの機能範囲から見た分散開発の適用範囲

OSIV系システムとWindows系システムのNetCOBOLの機能範囲を“図1-6 OSIV系システムとWindows系システムの機能範囲”に示します。

図1-6 OSIV系システムとWindows系システムの機能範囲



***: 共通仕様範囲

以下、“図1-6 OSIV系システムとWindows系システムの機能範囲”に従って、各機能範囲に対する分散開発を適用可能か説明します。

共通仕様範囲の機能

“図1-6 OSIV系システムとWindows系システムの機能範囲”で、共通仕様範囲の機能については、分散開発が適用可能です。通常のWindows系システム上のプログラム開発と同じ手順で翻訳・リンク・実行を行って、プログラム単位で動作を確認するところまで可能です。
ただし、共通仕様範囲の機能であっても、次のような細かな仕様の違いから完全にOSIV系システムでの動作と違いがある場合があります。

- システムに依存して処理が異なる。
- 同じ記述の解釈が異なる。
- OSIV系システムでしか意味を持たない記述。

これらの詳細については“付録A OSIV系COBOLとオープン系COBOLの相違点”を参照してください。

OSIV系システム固有仕様の機能

“図1-6 OSIV系システムとWindows系システムの機能範囲”で、OSIV系システム固有仕様である機能の中で、次の機能については、分散開発が適用可能です。

- 表示ファイル機能
- ネットワークデータベース機能

NetCOBOLの持つ分散開発を支援する機能を使用することによって翻訳・リンクすることができます。実行形式ファイルは、COBOLデバッガ上でのみ動作可能で、これにより制限付きながらWindows系システム上で動作確認することもできます。

その他の機能については、Windows系のNetCOBOLでの分散開発はある程度の翻訳チェックまでしかできません。

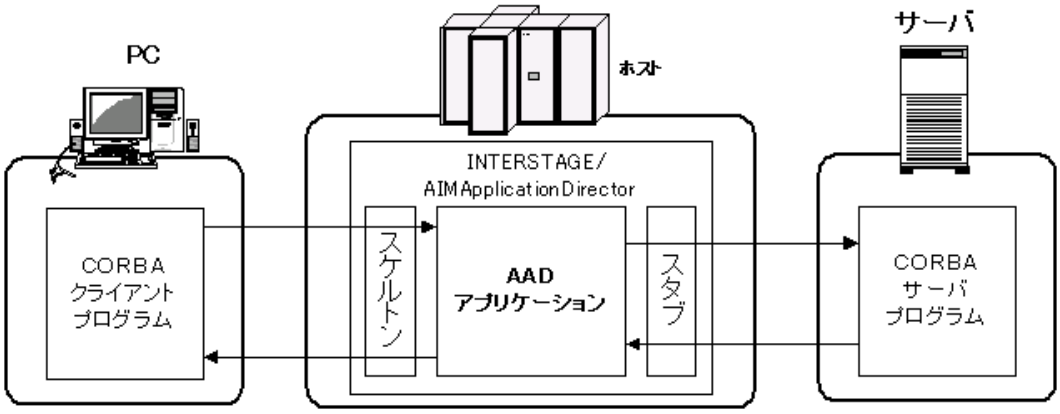
OSIV系のアプリケーションには、目的や運用方法などから種類があり、さまざまな構成を持ちますが、機能的には3つまたは4つの部分からなります。

1.2.3.3 CORBAアプリケーションへの適用

OSIV系システムで動作するCORBAアプリケーションは開発スタイルの違いから大きく2つに分かれます。

AIMApplicationDirectorアプリケーション(以降、AADアプリケーション)は、Interstageスタイルで開発するOSIV系システムで動作するCORBAアプリケーションです。このAADアプリケーションの開発には通常分散開発が適用されます。

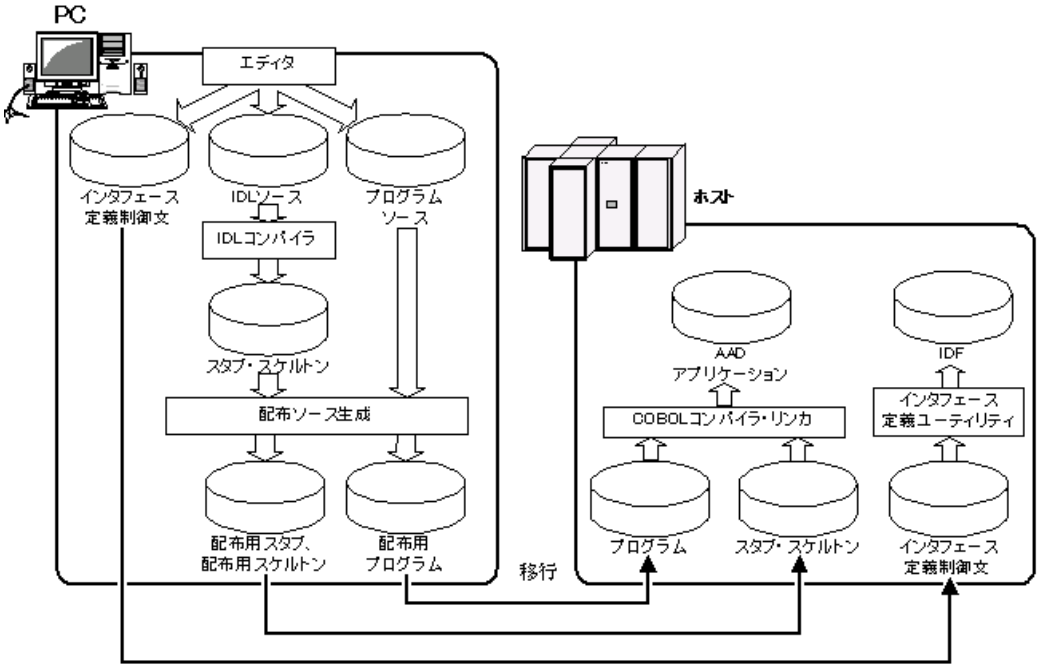
図1-7 AADアプリケーションの概要



AADアプリケーションの開発では、必要なスタブ・スケルトンファイルはIDLコンパイラによってその雛型を生成します。

しかし、IDLコンパイラが生成した雛型ファイルはファイル名の形式の違い等からそのままではOSIV系システムに移行することができません。このため、スタブ・スケルトンファイルとそれを参照するプログラムをOSIV系システムに移行可能な形式に変換する機能がNetCOBOLには用意されています。

図1-8 AADアプリケーションの開発の流れ



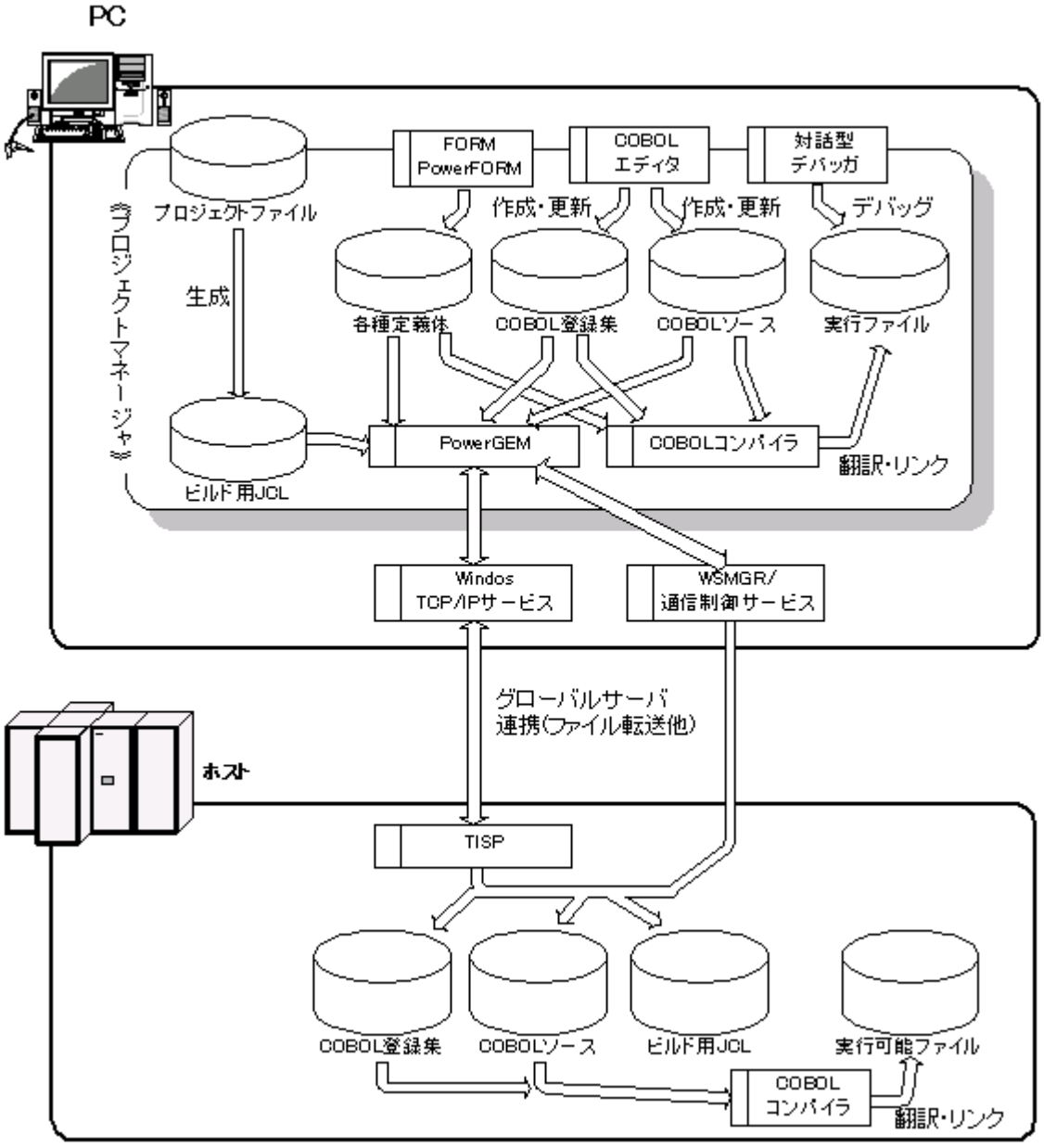
もう一方のOSIV系システムで動作するCORBAアプリケーションは、従来のAIMアプリケーションと同じ開発スタイルで開発するものです。この場合、分散開発は必須ではありませんが、PCあるいはUNIXサーバ側のCORBAプログラムとの通信インタフェースの作成にNetCOBOLの機能を用います。このため、AIMスタイルのCORBAアプリケーションの開発でも分散開発を適用するという選択肢も考えられます。

1.3 OSIV系アプリケーションの分散開発環境概要

1.3.1 分散開発環境の基本的なシステム構成

以下の分散開発を行う場合の、開発環境の大まかなシステム構成を示します。

図1-9 分散開発環境のシステム構成



1.3.2 分散開発に必要なソフトウェア製品・コンポーネント

OSIV系プログラムをWindows系システム上で分散開発するために、OSIV系システム上および

Windows系システム上に必要となるソフトウェア製品・コンポーネントとの一覧を“表1-1 分散開発環境に必要なソフトウェア製品・コンポーネント”に示します。

表1-1 分散開発環境に必要なソフトウェア製品・コンポーネント

目的	OSIV系システム	Windows系システム
分散開発基盤		
文字コード変換	ADJUST	SystemWaker/CharsetMGR　または Interstage Charset Manager
グローバルサーバ連携*1 (TCP/IP連携)	TISP + DTS	Windows TCP/IPサービス*2
グローバルサーバ連携*1 (WSMGR連携)	-	通信制御サービス + WSMGR + WSMGR APIオプション
開発系		
開発言語(COBOL)	COBOL85	NetCOBOL *3 (+ JEFオプション)
開発環境	PFD/APDF	プログラムマネージャ *3
ソース等の編集	PFDエディタ/GEMエディタ	COBOLエディタ *3
プログラムのデバッグ	TESTCOB/TESTコマンド	COBOLデバッガ *3
画面、帳票などのデザイン	PSAM(IFD/FMTGEN制御文)	FORM/PowerFORM *3
オーバレイパターンの作成	ADJUST (JRQNOVL)	FORM/PowerFORM
画面、帳票表示、出力	PSAM	MeFt *3
資産管理	GEM	PowerGEM Plus *3 (+ PowerGEM Plus Administrator)
実行基盤		
データベース	SymfoWARE Server	SymfoWARE Server他
トランザクション管理、 通信制御	AIM	なし
ネットワークデータベース	AIM/NDB	なし
文字コード処理		JEF拡張漢字サポート*4

*1 グローバルサーバ連携機能は、TCP/IP連携とWSMGR連携のいずれかを選択して使用します。 必要製品の詳細なVL/PTFのについては、“PowerGEM PLUS ソフトウェア説明書”を参照してください。
*2 オペレーティングシステムが提供するサービスで、Windowsネットワークの設定でTCP/IPネットワークを設定します。
*3 NetCOBOL Professional Edition 開発パッケージに含まれます。
*4 JEFオプションを使用する際に必要になります。

Windows系システム上の分散開発の環境は、NetCOBOL製品に含まれるコンポーネントだけで構築可能ですが、以下のソフトウェア製品と組み合わせで、より分散開発に適した環境を構築できます。

- NetCOBOL JEFオプション
- PowerGEM Plus Administrator

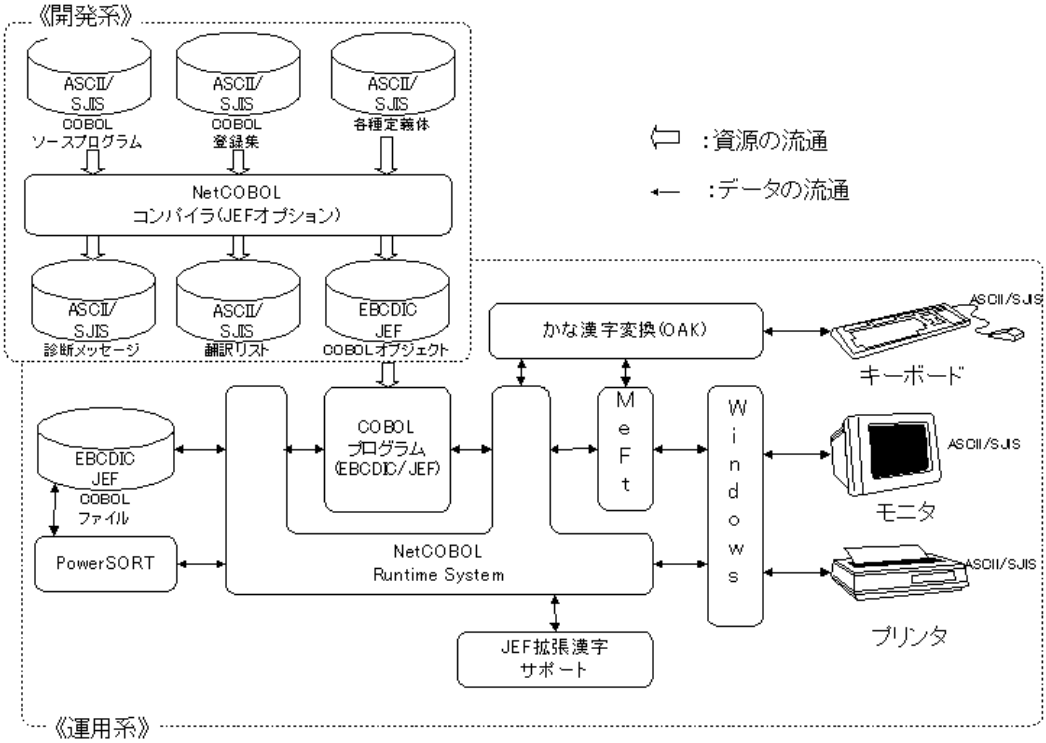
1.3.2.1 NetCOBOL JEFオプション

NetCOBOL JEFオプションは、NetCOBOLに次の機能を追加するためのオプション製品です。

- Windows環境で動作し、EBCDIC(カナ)/JEFコードでデータを扱うCOBOLアプリケーションを開発する機能。

OSIV系のプログラムをWindows系システムのNetCOBOLで分散開発する場合、それぞれのオペレーティングシステムの採用している文字コードの違いが問題となる場合が少なくありません。NetCOBOL JEFオプションを使用する場合、コード系の違いが原因の問題のほとんどが自然に解決されます。

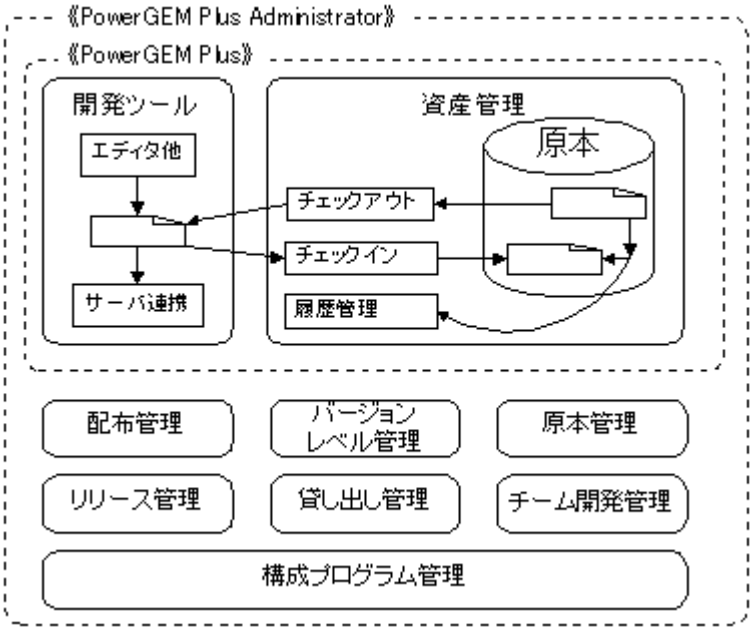
図1-10 NetCOBOL JEFオプション使用時の各種資源とシステム構成



1.3.2.2 PowerGEM Plus Administrator

NetCOBOL製品に含まれるPowerGEM Plusが開発ツール群と資産管理の機能を提供するに対して、PowerGEM Plus Administratorは、ソフトウェア構成管理(SCM：Software Configuration Management)の考え方を適用して、開発プロセスと開発資産を統合的に管理し、ソフトウェア開発の生産性を高める機能を提供します。

図1-11 PowerGEM PlusとPowerGEM Plus Administratorの機能差



チーム開発管理

ソフトウェアの開発を“チーム開発”と言う論理的な単位で管理します。チーム開発は階層構造を持つ次の3つの要素で定義し、各要素を同一レベルに配置して、それら全体をチーム開発という単位で定義して管理します。

- ソフトウェア構成
  ソフトウェア全体のプログラム構成を表します。コンポーネントによる階層構造を持ちます。
- プロジェクト構成
  開発プロジェクトの組織構成を表します。システム管理者、チーム開発管理者および開発者による階層構造を持ちます。
- 資産構成
  ソフトウェア全体の開発資産の構成を表します。コンポーネントの構成に対応した階層構造を持ちます。

PowerGEM Plus Administratorを利用するユーザは、いずれかのチーム開発に属し、ユーザ単位に設定された“アクセス権”に従って、チーム開発に登録されている資産を操作することができます。

原本管理

Windows系システムを含めた各種プラットフォーム上に配置し、各種資産を格納・管理する資産格納庫を“原本”として定義して、これWindows系システムから一括管理します。
原本として定義した資産格納庫は、配置先や物理ファイル名などを意識することなく論理的な原本名を使用して、操作することができます。また、原本管理では安易な資産の修正や誤操作による削除などから開発資産を保全する機能を提供します。

貸出管理

貸出管理では、ソフトウェア開発における開発の手続きを開発資産と作業手順を結びつけて統合的に管理します。このことにより管理者は開発資産について次の点を確実に把握・管理することができます。

- どの資産(WHAT)
- いつ(WHEN)
- だれに(WHO)

- 何のために(WHY)

また、開発資産の二重修正防止や正規の手続き以外での資産の修正防止など、開発資産の確実な保全を計ることができます。このため、開発作業の効率向上や品質記録の保持を図ることができます。

進捗管理

貸出管理の情報を利用して開発作業の進捗状況を管理する機能です。
進捗状況を定量的に集計してリアルタイムに表示することができます。作業の完了時には、作業全体の開発規模をあらわす各種情報を集計して出力することができます。

構成プログラム管理

チーム開発に登録してあるプロジェクトファイルおよびメイクファイルで定義しているプログラムの依存関係をもとに、資産の構成情報や修正を行った資産が他の資産に与える影響などを確認することができます。

リリース管理

提供するソフトウェアのバージョンを構成するソースファイルなどの資産を原本からWindows系システム上に取り出します。プロジェクトファイルおよびメイクファイルに定義している依存関係情報をもとに、資産を取り出すことができます。また、過去に提供したソフトウェアのバージョンを構成する資産を、確実かつ容易に復元することができます。

配布管理

原本で管理されている資産やWindows系システム上の資産を、他のWindows系システムや他プラットフォーム(グローバルサーバ、UNIXサーバなど)に送付します。次のような場面で有効に利用することができます。

- 生成した実行形式ファイルをテスト環境に送付する場合
- 資産を翻訳するために、目的とするプラットフォーム上に送付する場合

送付先や送付資産の組み合わせを“配布パターン”として登録しておき、資産の送付作業では、登録した配布パターンを選択するだけで資産を確実に送付することができます。

バージョンレベル管理

Windows系システム上の原本(GEMライブラリ)に格納している各資産の世代を統一的に管理します。ソフトウェアを構成する資産の最新レベルに論理的な名称(バージョンレベル)を設定します。なお、このバージョンレベルは、リリース管理でも設定することができます。

第2章 分散開発環境の構築

本章では、OSIV系システムのアプリケーションの開発をWindows系システム上で行う場合の開発環境構築の考え方と、実際の開発作業に先立って必要な環境設定の方法について、説明します。

2.1 分散開発環境の構築

ここではより具体的に分散開発の手順を説明するとともに、そのための開発計画、開発環境の立案の仕方について、説明します。

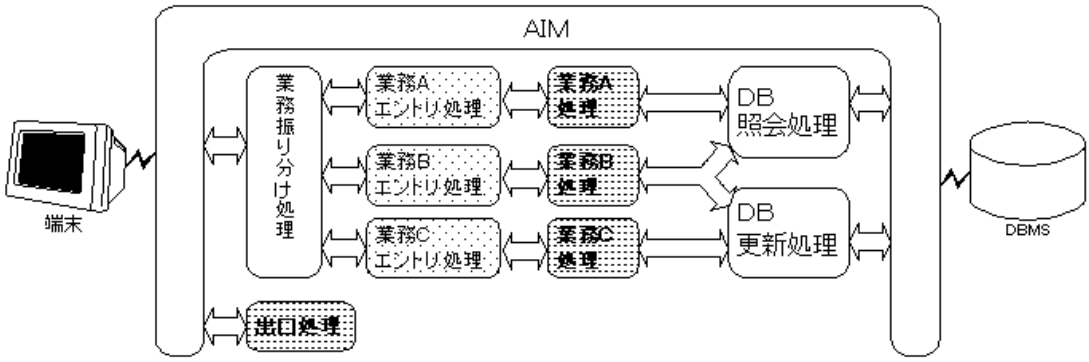
2.1.1 分散開発の開発計画の立案

分散開発の開発計画の立案する場合、開発対象となるOSIV系プログラムに対して、次のようなことを検討しておく必要があります。

- 分散開発の適用範囲
- 開発対象と方法
- テスト方法
- 資産の管理方法と配置

以下では、次のような構成のオンライン形態で動作するOSIVプログラムの分散開発を行う場合を例として考えます。

図2-1 分散開発の対象プログラムの構成例



2.1.1.1 分散開発の適用範囲の決定

このOSIV系プログラムを構成する個々の処理が次のように分類できるとします。

表2-1 分散開発の難易度による処理の分類

No	分散開発の適用の難易度	処理内容
1	分散開発が困難な制御プログラム	業務振り分け処理
2	分散開発可能であるが、一部OSIVと動作の異なるプログラム	業務Ａ～Ｃエントリ処理
3	分散開発に適したプログラム	業務Ａ～Ｃ処理、出口処理
4	分散開発が困難な共通プログラム	ＤＢ照会処理、ＤＢ更新処理

このようなOSIV系プログラムを分散開発する場合、その適用範囲の決め方は次のような幾つかの案があります。

- 案1:分散開発をしない。
  対象プログラムが処理毎に別々のCOBOLソースプログラムに分割されていない1つのCOBOLソースプログラムからなっているような場合は、分散開発のメリットをほとんど活かす事ができません。
  新規開発であれば、翻訳チェックのみ行うという選択もありますが、UP開発の場合、OSIV系システムからWindows系システムにCOBOLソース、登録集等の資源を移行する手間がかか

る分、分散開発をするメリットはまったくありません。

- 案2:翻訳チェックのみ行う
  分散開発が適用しづらい処理であっても、翻訳チェックまでは適切にできる場合が多くあります。次のような条件に該当するなら、分散開発を適用する価値があります。
  - 開発するプログラムの量が多い(総ステップ数よりも、分割されたプログラム数)。
  - 開発要員のOSIV系システムでの開発経験が乏しい。
- 案3:個々の機能・処理に対する分散開発の難易度に依存して、どの工程までやるか決める。
  機能・処理など毎に開発の単位を細分化し、それぞれについてどの工程まで分散開発を適用するか決定します。
  例えば、上記の例に対して、次のように分散開発でどの工程まで行うか決めます。

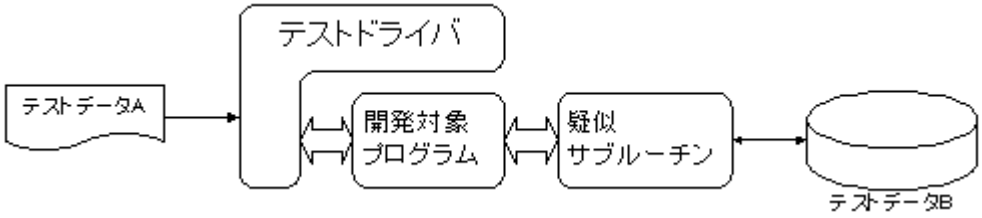
表2-2 処理毎の分散開発の適用範囲

処理内容	分散開発を適用する工程
業務振り分け処理	分散開発しない
業務Ａ～Ｃエントリ処理	プログラミング、翻訳チェック、単体テスト
業務Ａ～Ｃ処理、出口処理	プログラミング、翻訳チェック、単体テスト
ＤＢ照会処理、ＤＢ更新処理	プログラミング、翻訳チェック

ただし、このように作業を進める場合、単体テストまで実施する部分に対して、通常は次のようなプログラムを用意する必要があります。

- テスト用ドライバ
- 疑似サブルーチン

図2-2 分散開発時の単体テスト時の構成



この場合、テストドライバ/疑似サブルーチンの作成が負担になるため、分散開発で高い生産性を期待するためには、次の式のｎが大きな値にならなければなりません。

テスト対象のプログラム数　＝　テストドライバ/疑似サブルーチン　×　ｎ

従って、次の条件を満足するようなOSIV系プログラムを開発するのであれば、分散開発による効果を多く期待できると言えます。

- 開発する機能・処理が多い。
- 機能・処理毎に適切にプログラムが分割されている。
- 機能・処理階層毎のインタフェースが共通化されている。

2.1.1.2 開発対象と方法等の決定

これまでに何度か述べたとおり、最終的にOSIV系プログラムを構成するプログラム資産のすべてを分散開発の対象とすることは不可能であるか、あるいは効率的でない場合が少なくありません。基本的には、分散開発の開発対象となりうるのは、次のような資産に限定されます。

- COBOLソースプログラム
- COBOL登録集
- メッセージ定義体
- フォームオーバレイパターン


注意

メッセージ定義体、フォームオーバレイパターンは、Windows系システムで作成したものとOSIV系システムで最終的に利用可能なものとで機能差を機械的にチェックする仕組みはWindows系システムには存在しません。このため、作成時の注意が必要です。

なお、次のような資産は分散開発の対象にはなりえません。

- アセンブラ等のCOBOL以外の言語で記述されたプログラム
- JCL/CLIST
- データベース定義


注意

データベース定義は、OSIV系システムでSymfoWARE Serverのリレーショナルデータベース機能を使用している場合は部分的に再利用可能です。

また、分散開発時にJEFオプションを使用するかどうかも予め決めておかなければなりません。JEFオプションを使用する場合、オペレーティングシステムの採用している文字コードの違いが原因となる問題の多くを意識する必要がなくなるというメリットがあります。
しかし、単体テストまで実施する場合、テスト用データはEBCDIC/JEFコード系のものを用意する必要があります。また、いくつかJEFオプション固有の制限事項(“D.2 JEFオプションの機能上の特徴と制約”を参照)もあります。

2.1.1.3 テスト方法

作成したプログラムをそれぞれテストする際、処理ロジックの確認と処理結果の確認という2つの観点が存在します。NetCOBOLでは、それを支援するためにそれぞれ次のような機能を提供します。

- プログラムの処理ロジックの確認
  - COBOLデバッガ
    ソースコードを見ながらの対話型デバッグを可能とします。また、対話的に実行したデバッグの過程を記録することによって、同じデバッグ操作を半自動的に繰り返す事ができます。
  - COUNTオプション + SIMPLIA-EXCOUNTER
    翻訳オプションCOUNTを指定して翻訳したプログラムを実行することで、プログラム中のどの命令が何回実行されたかを記録することができます。SIMPLIA-EXCOUNTERはこの情報を蓄積し、各テストケースに対する命令網羅率などの帳票を作成します。
- プログラムの出力の確認
  - COBOLファイルユーティリティ
    COBOLプログラムを使用せず、COBOLファイルを操作する機能を提供します。次のような操作が可能です。
    - 各種テキストエディタを使って作成したデータからCOBOLファイルを作成する。
    - COBOLファイルの複写/移動/削除、ファイル構造の変換
    - 索引ファイルの再編成/復旧/属性の表示
    - COBOLファイルのレコードの操作(表示/編集/整列など)を行う。
  - SIMPLIA-FILECOMP
    2つのファイルの内容をレコード単位、フィールド単位で比較します。ファイルレコードの構造はCOBOL登録集を入力・解析して比較条件の設定時に参照することが

できます。

注意

NetCOBOLでは、以下の機能を使用するプログラムをそのまま実行できません。これらの機能を使用するプログラムを実行する場合にはCOBOLデバッガの使用が不可欠です。

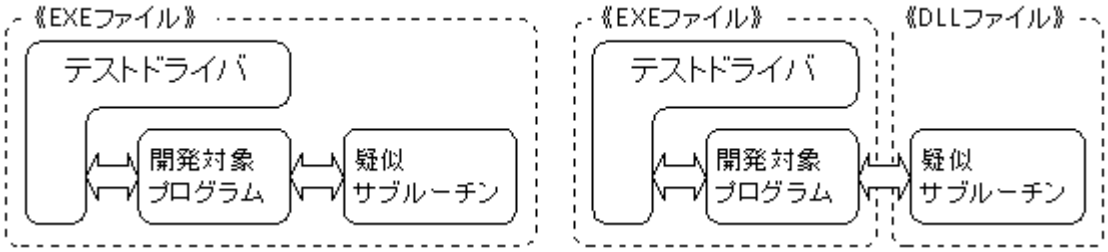
- ネットワークデータベース機能
- OSIV系システム固有の宛先を使用する表示ファイル機能

また、分散開発で単体テストを実施する場合は先に説明したようにテスト対象のプログラムだけではなく、テストドライバ/疑似サブルーチンが必要となります。これらをどう組み合わせて実行形式ファイルを作成するか、いくつか方法があるので説明します。

- テストドライバとテスト対象プログラムを静的にリンクする。
  次のような理由から、テスト対象プログラム毎にテストドライバを用意するような場合は、この形態を用いるべきです。
  - テスト対象プログラムの呼び出しインタフェースが共通化されていない。
  - テスト対象プログラムの個数が少ない。

  疑似サブルーチンを含めて静的にリンクするかどうかにより、次のどちらかの形態になります。

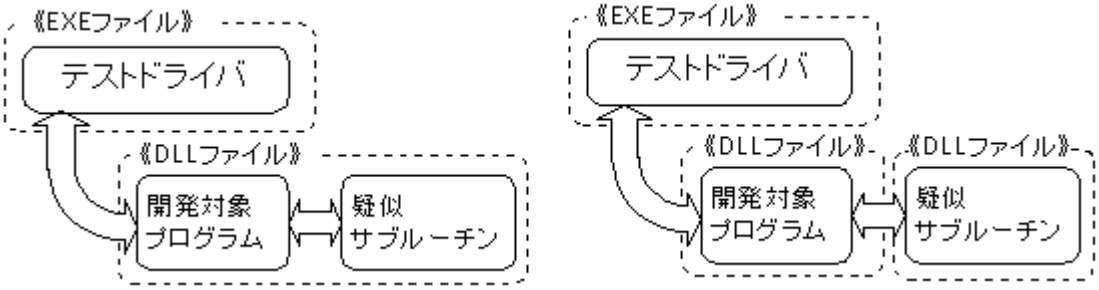
図2-3 分散開発時のテスト用の補助プログラムと開発対象プログラムの関係例1



- テスト対象プログラムを動的に呼び出すようにする。
  次のような理由から、複数のテスト対象プログラムに共通のテストドライバを用意するような場合は、この形態をとるべきです。
  - テスト対象プログラムの呼び出しインタフェースが共通化されている。
  - テスト対象プログラムの個数が多い。

  疑似サブルーチンを含めて静的にリンクするかどうかにより、次のどちらかの形態になります。

図2-4 分散開発時のテスト用の補助プログラムと開発対象プログラムの関係例2

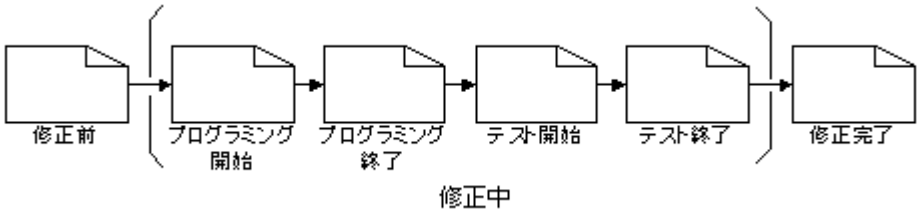


どちらの方法を採用するかで、プログラミング・テスト作業の一部が異なります。

2.1.1.4 資産管理についての考え方

どのようなプログラム資産を開発する場合であっても、各開発資産はさまざまな状態を持ち、それを管理してゆくことが要求されます。

図2-5 開発工程における資産の状態



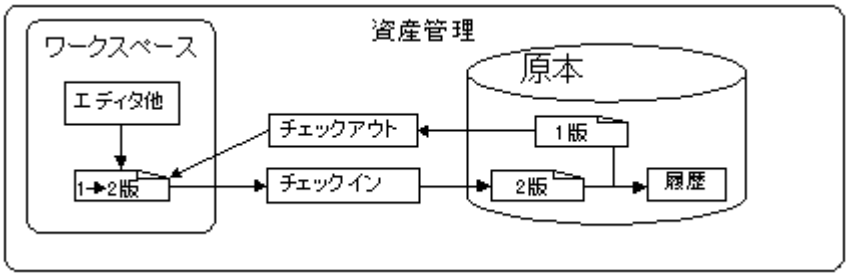
分散開発を行う場合、更に次のような状態も管理してゆく必要があります。

- 資産はどこにあるか？
- コード系は？
- 資産はどこで作成・修正するべきか？
- 資産の転送はいつおこなわれたか？
- 資産は修正の対象か、参照専用か？
- 形式変換処理の必要があるか？必要なら、それは済んでいるか？

その上、最終的にOSIV系プログラムを構成するプログラム資産のすべてを分散開発の対象とすることは不可能であるか、あるいは効率的でない場合が少なくないという事情から、資産の種類毎にあつかいを変える必要もあり、管理作業がより複雑になります。このため、OSIV系プログラムの分散開発を行う場合は、開発作業の流れのなかでどのように資産を管理するか予め明確にしておく必要があります。

NetCOBOL製品に含まれるPowerGEM Plusでは、この開発資産の状態と履歴を管理するために次のような機能(資産管理機能)を持っています。

図2-6 PowerGEM Plusの資産管理機能の概要



用語の定義

PowerGEM Plusの固有の用語についてその定義を示します。

原本:

開発資産の格納庫です。概念的にはOSIV系システムのGEMのGEMライブラリに相当しますが、次の点で異なります。

- OSIV系のGEMライブラリと異なり、直接操作することができません。
- 物理的な階層構造を持つ事ができます。
- Windows系システムのPowerGEMの原本をOSIV系システムやSolarisなど他システム上におけます。

ワークスペース:
参照、更新その他の操作のために、原本から取り出した開発資産を格納する場所です。PowerGEM Plusで開発資産を操作する場合、かならず原本に対してワークスペースを割り当てる必要があります。

チェックアウト:
開発資産を参照、更新するために原本からワークスペースに取り出す操作です。

チェックイン:
参照、更新が済んだ開発資産をワークスペースから原本に戻す操作です。

履歴:
チェックアウトからチェックインまでの間に開発資産が修正されたなら、ファイルの属性に依存した方法で修正の前後の差分がとられ、履歴として管理されます。

PowerGEMの資産管理機能を使用して分散開発を実施する場合、開発環境の構築前に次のことを決めておく必要があります。

- 管理対象ファイル(COBOLソース、COBOL登録集、…)
- 原本の格納場所
- 原本の階層構造
- ワークスペースの格納場所
- ワークスペースの階層構造

なお、PowerGEM Plusを使用しての資産管理の詳細については、“PowerGEM Plus説明書”および“PowerGEM Plus 資産管理オペレーションガイド”を参照してください。

2.1.2 分散開発環境の構築

分散開発環境は、その開発計画の立て方によってさまざまな構成を持つもので、それをすべて説明することは困難です。ここでは、ある開発計画をモデルとして、それに合わせた分散開発環境の例を説明します。

2.1.2.1 開発計画例

“2.1.1 分散開発の開発計画の立案”で示したようなOSIV系プログラムに対して、次のような開発計画を立案します。

- 分散開発の適用範囲
  このOSIV系プログラムは、各機能・処理毎にプログラムが分割されているものとします。また、いくつか特徴的な機能が使用されているものとします。

No	分散開発の適用範囲	処理内容	備考
1	分散開発の対象外	業務振り分け処理	表示ファイル機能(AIM固有)使用
2	単体テストまで実施 ただし、一部資産はOSIV系システムで開発	業務Ａ～Ｃエントリ処理	表示ファイル機能(DSP)を使用
3	単体テストまで実施	業務Ａ～Ｃ処理、出口処理	
4	翻訳チェックまで実施 ただし、一部資産はOSIV系システムで開発	ＤＢ照会処理、ＤＢ更新処理	ネットワークデータベース機能を使用


注意

上記は、1つのモデルケースであり、表示ファイル機能(AIM固有)やネットワークデータベ

ース機能を使用するプログラムが分散開発の対象とならないことを示すものではありません。

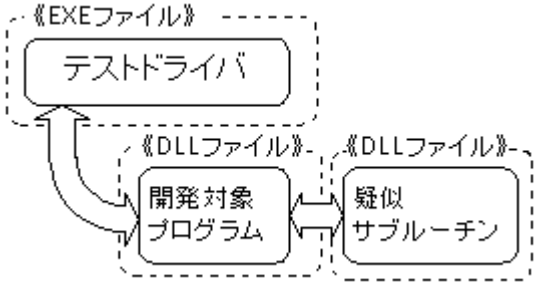
● 開発対象と方法
どの資産をOSIV系システム/Windows系システムで参照するか、また分散開発時にどのように利用するかを決めます。

処理内容	開発資産	開発方法他
業務振り分け処理	COBOLソース	分散開発の対象とせず、OSIV系システムで開発
	COBOL登録集	分散開発。
業務Ａ～Ｃ エントリ処理	COBOLソース	分散開発。
	COBOL登録集	
	メッセージ定義体	OSIV系システムで開発。画面帳票定義体(SMD形式)に変換するツールを使用して、Windows系システムに移行して、参照のみ行う。
業務Ａ～Ｃ処理、 出口処理	COBOLソース	分散開発。
	COBOL登録集	
ＤＢ照会処理、 ＤＢ更新処理	COBOLソース	分散開発。ただし、翻訳チェック後にOSIV系システムに登録、以降はOSIV系システムで開発
	COBOL登録集	分散開発。
	サブスキーマ定義	OSIV系システムで開発。登録集ファイル化するツールを使用して、Windows系システムに移行して、参照のみ行う。

なお、COBOL登録集は、次の2つのカテゴリに分かれると考えます。
― すべての開発者が共通で参照するもの。
― 各担当者が開発するもの。

● テスト方法
各業務プログラムの呼び出しインタフェースを統一して、共通のテストドライバから呼び出す形で単体テストを行うこととします。また、疑似サブルーチンも静的にリンクせず、DLLとして作成しておき、動的に呼び出します。

図2-7 テストドライバ/疑似サブルーチンと開発対象プログラムの関係



開発グループ内にテストドライバや疑似サブルーチンの開発に専任する要員を用意し、他の開発要員はテストドライバや疑似サブルーチンの作成には関わりません。

● 資産の管理方法と配置
開発資産を次の分類で配置します。
― 開発の開始時点の資産および開発・テストが完了したもの

- すべての開発生産物

— 開発担当者が共通で参照するもの
  - テストドライバ
  - 疑似サブルーチン
  - COBOL登録集(一部)
  - メッセージ定義体
  - サブスキーマ定義体

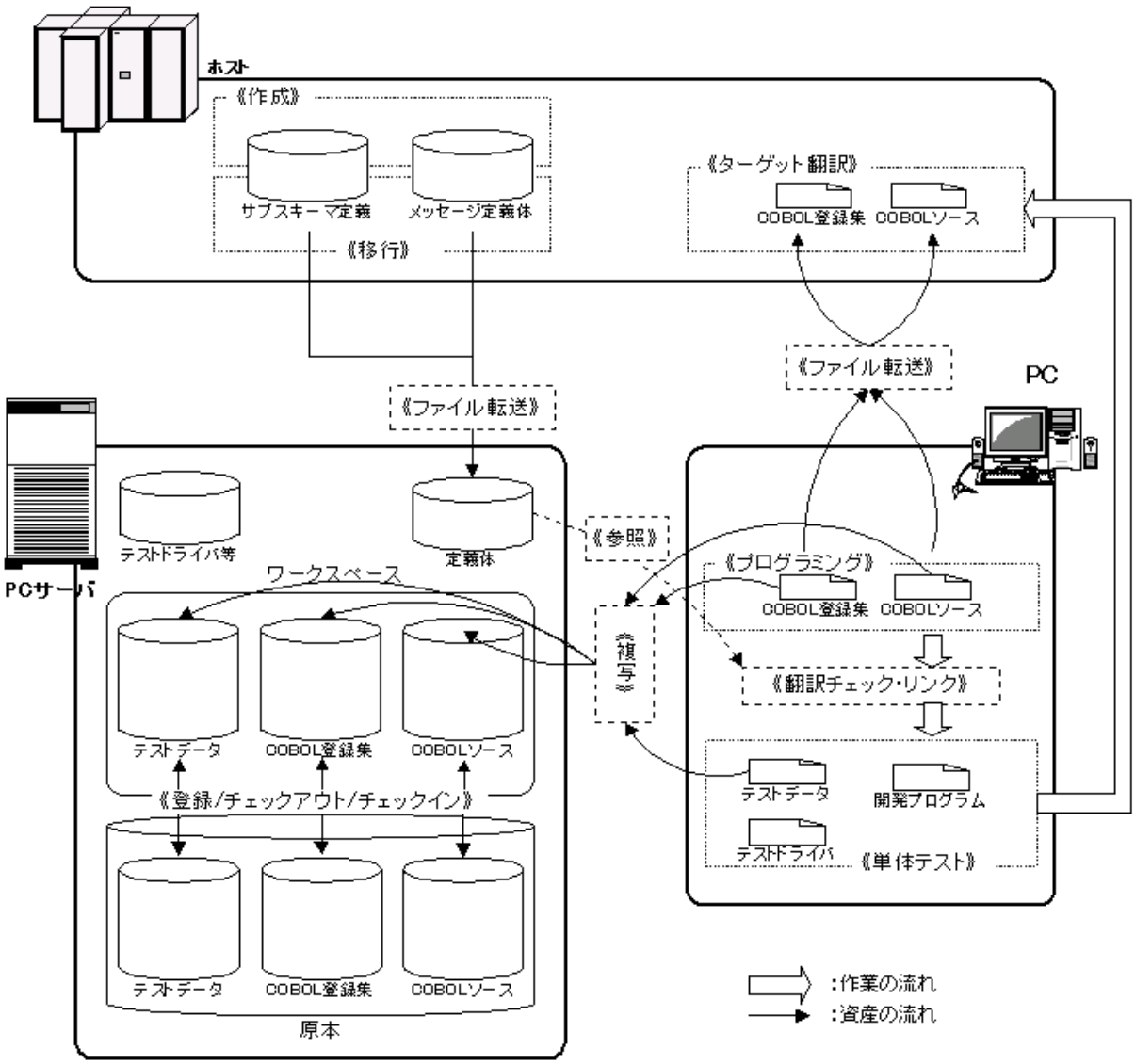
— 開発担当者個々人が開発するもの
  - COBOLソースプログラム
  - COBOL登録集
  - テストデータ等

これらの資産をPowerGEM Plusの資産管理機能を使用して管理するものとします。開発担当者が共通で参照するものは、PCサーバに置き、ネットワーク経由で参照するものとします。

2.1.2.2 開発環境構築例

前述の開発計画を例として、開発環境を構築した例を示します。

図2-8 分散開発環境の全体構成(例)



2.2 分散開発のための環境設定

Windows系システムで、OSIV系プログラムの分散開発を実施する場合、OSIV系システムとの連携が必要となります。ここでは、そのための環境設定の方法を説明します。

2.2.1 サーバ連携方法の選択

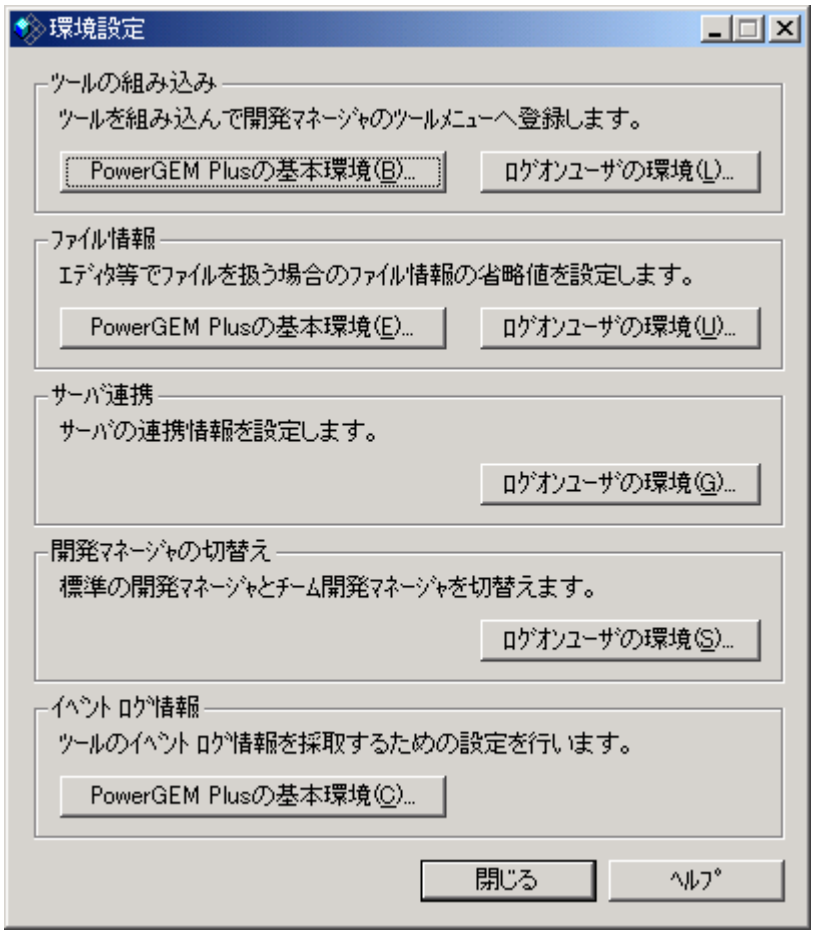
OSIV系システムとWindows系システムで連携を行う場合、次の2つの方法が存在します。

- TCP/IP接続
- WSMGR接続

サーバ連携情報の設定に先立って、まずどちらの方法で連携を行うか選択する必要があります。なお、この設定は分散開発の他の操作と異なり、COBOLプロジェクトマネージャから行うことはできませんので、注意してください。
以下、サーバ連携方法の選択方法を示します。

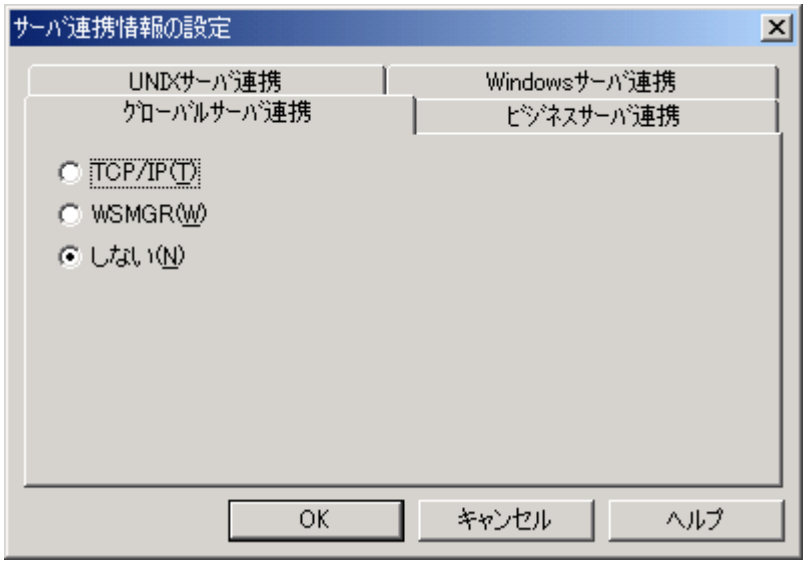
1. スタートメニューから、PowerGEM Plusを起動します。
2. 〔オプション〕メニューから“環境設定”を選択すると、〔環境設定〕ダイアログが表示されます。

図2-9 環境設定ダイアログ



3. 〔サーバ連携〕の “ログオンユーザの環境”ボタンをクリックして、〔サーバ連携情報の設定〕ダイアログを開きます。

図2-10 サーバ連携情報の設定ダイアログ



4. 〔グローバルサーバ連携〕のタブで使用する連携方法を選択します。
5. 〔ＯＫ〕ボタンをクリックして、選択を保存します。
6. PowerGEM Plusを終了します。

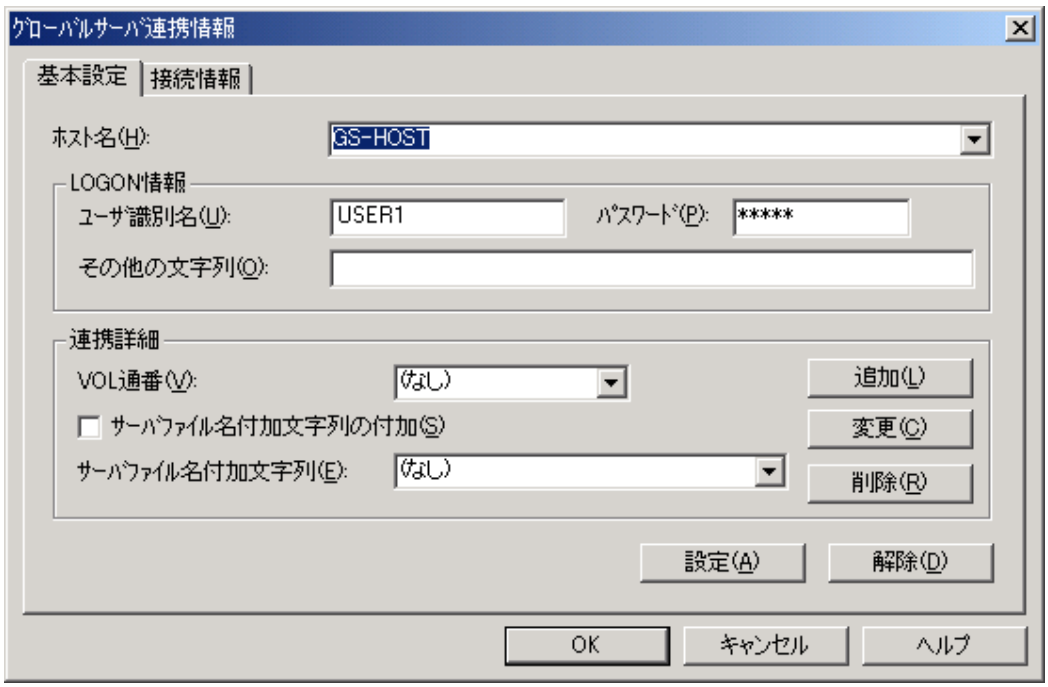
2.2.2 サーバ連携情報の設定

サーバ連携情報は、PowerGEM PlusあるいはCOBOLプロジェクトマネージャのどちらからでも設定できます。ここでは、COBOLプロジェクトマネージャから設定する方法を説明します。

1. スタートメニューから、COBOLプロジェクトマネージャを起動します。
2. 〔プロジェクト〕－〔分散開発〕メニューから“サーバ連携情報”を選択すると、〔グローバルサーバ連携情報〕ダイアログが表示されます。〔グローバルサーバ連携情報〕ダイアログで設定する必要のある情報は連携方法として、“TCP/IP”を選択した場合と“WSMGR”を選択した場合で異なります。
3. 接続方法として“TCP/IP”を選択した場合の設定方法について、説明します。
   — 〔基本設定〕ページ
   以下の情報を設定します。
   - ホスト名:
     ネットワーク上のグローバルサーバを識別する名前あるいはＩＰアドレスを直接指定します。
   - ユーザ識別名:
     グローバルサーバとの接続を開設するユーザ識別名を指定します。英字で始まる7文字以内の文字列でなければなりません。
   - パスワード:
     ユーザ識別名と対応するパスワードを指定します。入力結果は各文字が“*”に置き換えられて表示されます。
   - その他の文字列:
     グローバルサーバとの接続を開設するLOGONコマンドのオペランドを指定します。LOGON時にアカウント名の指定が必要な場合などは、ここで指定します。

〔連携詳細〕はグローバルサーバのファイルを操作するダイアログボックスに対するデフォルト時の操作ビューの設定です。通常は設定の必要はありません。

図2-11 〔グローバルサーバ連携情報〕ダイアログ(TCP/IP接続時)の〔基本設定〕ページ

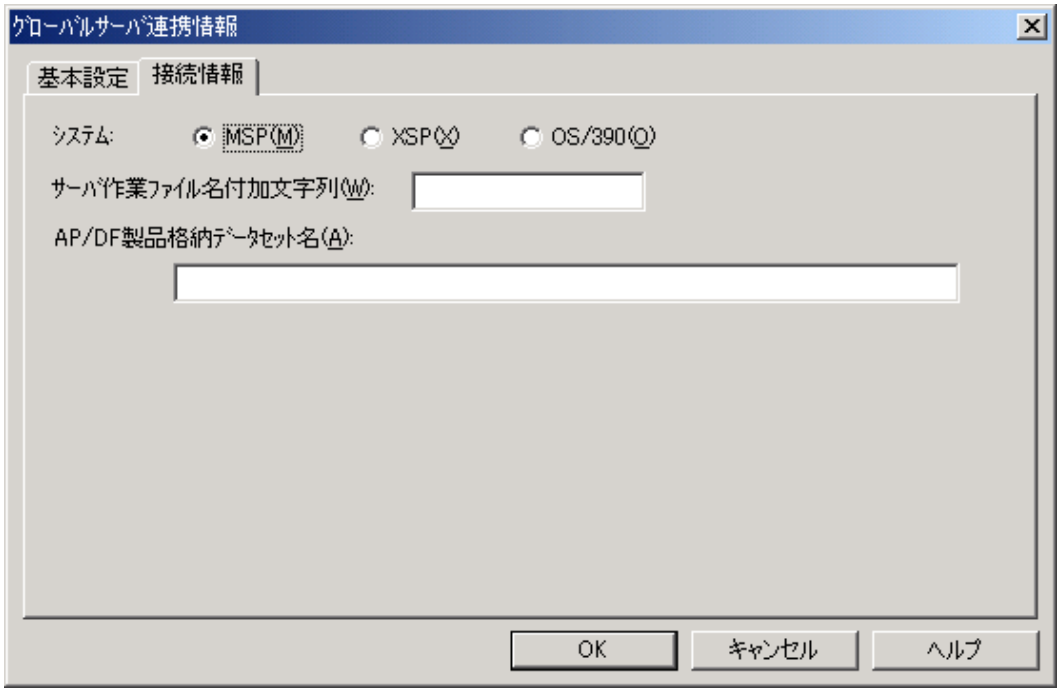


— 〔接続情報〕ページ

以下の情報を設定します。

- システム:
  連携するグローバルサーバのシステム名を選択します。
- AP/DF製品格納データセット名:
  連携するグローバルサーバのシステムがXSPの場合、AP/DFがインストールされているロードモジュールライブラリを、引用符で囲んで完全修飾名で指定します。省略した場合は、リンクリストに連結されているロードモジュールライブラリを検索します。

図2-12 〔グローバルサーバ連携情報〕ダイアログ(TCP/IP接続時)の〔接続情報〕ページ

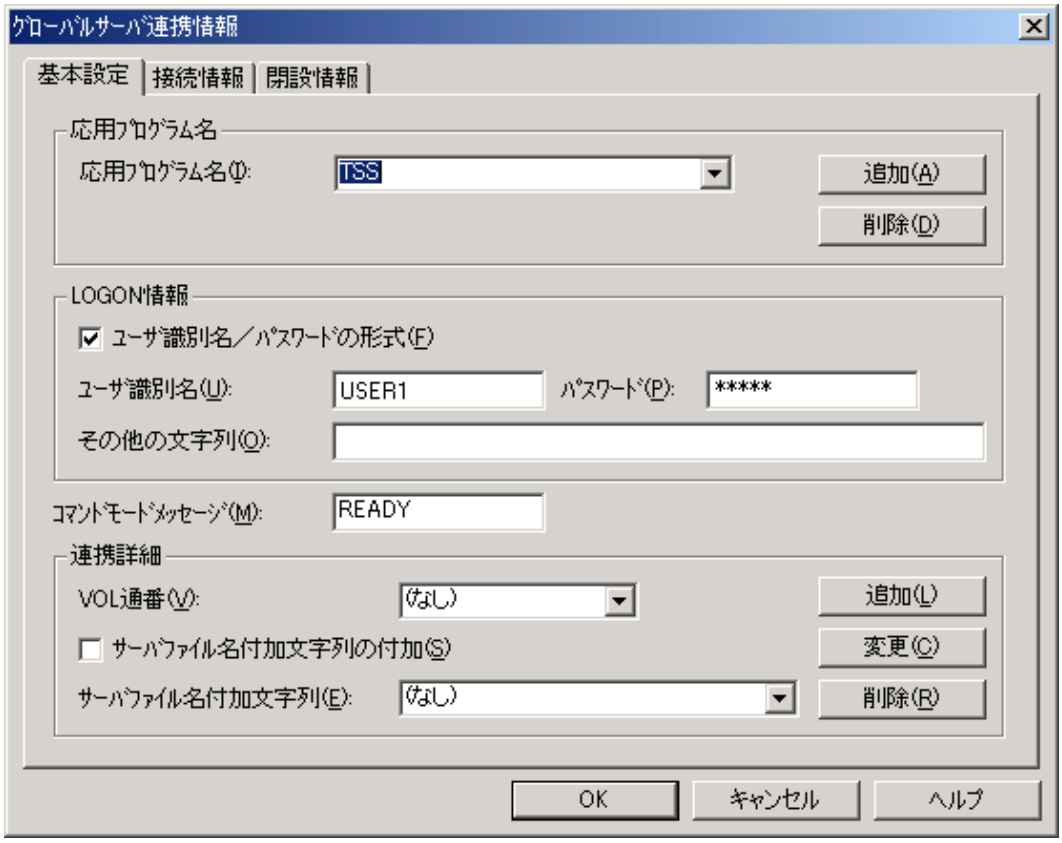


4. “WSMGR”を選択した場合、次のような〔グローバルサーバ連携情報〕ダイアログが表示されます。

   — 〔基本設定〕ページ
   以下の情報を設定します。
   - 応用プログラム名:
     グローバルサーバ連携を開設する応用プログラム名を英字で始まる８文字以内の英数字で指定します。
   - ユーザ識別名:
     グローバルサーバとの接続を開設するユーザ識別名を指定します。英字で始まる7文字以内の文字列でなければなりません。
   - パスワード:
     ユーザ識別名と対応するパスワードを指定します。入力結果は各文字が“*”に置き換えられて表示されます。
   - その他の文字列:
     グローバルサーバとの接続を開設するLOGONコマンドのオペランドを指定します。LOGON時にアカウント名の指定が必要な場合などは、ここで指定します。

   連携詳細はグローバルサーバのファイルを操作するダイアログボックスに対するデフォルト時の操作ビューの設定です。通常は特に設定の必要はありません。

図2-13 〔グローバルサーバ連携情報〕ダイアログ(WSMGR接続時)の〔基本設定〕ページ



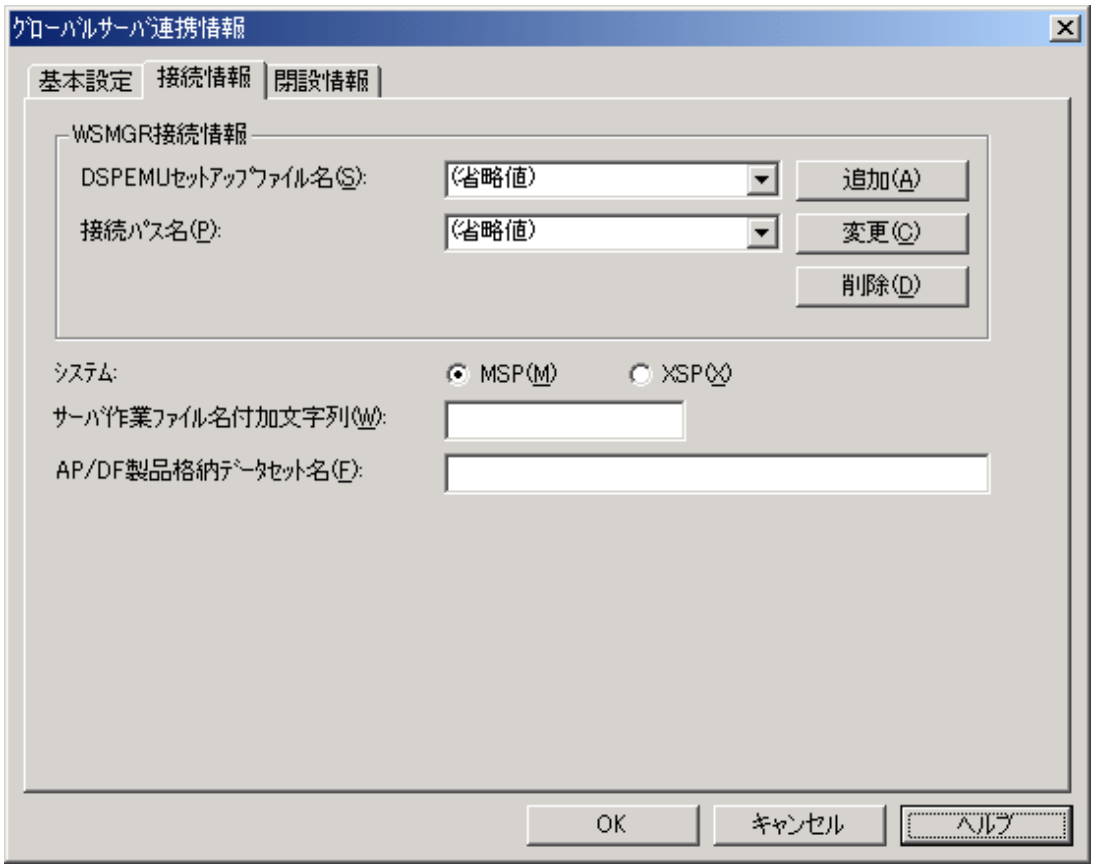
— 〔接続情報〕ページ

以下の情報を設定します。

- システム:
  連携するグローバルサーバのシステム名を選択します。
- AP/DF製品格納データセット名:
  連携するグローバルサーバのシステムがXSPの場合、AP/DFがインストールされているロードモジュールライブラリを、引用符で囲んで完全修飾名で指定します。省略した場合は、リンクリストに連結されているロードモジュールライブラリを検索します。

その他については、通常は設定する必要はありません。

図2-14 〔グローバルサーバ連携情報〕ダイアログの〔接続情報〕ページ



第3章 開発作業(プログラミング)

OSIV系プログラムを分散開発する場合でも、COBOLソースを始めとするプログラム資産の開発作業はWindows系システムで動作するプログラムを開発する場合と基本的に違いはありません。このような点について、このマニュアルでは概要レベルの説明にとどめますので、その詳細については、“NetCOBOL 使用手引書”を参照してください。
ここでは、従来のOSIV系のプログラミングとの違いや特に分散開発のために用意された機能について説明します。

3.1 開発作業の概要

Windows系システム上のNetCOBOLで、OSIV系のプログラムのプログラミングを行う場合、以下の作業が必要になります。

- プロジェクトファイルの作成
  NetCOBOLでプログラムを翻訳・リンクするためにCOBOLプロジェクトマネージャのプロジェクト管理機能を使用します。この機能を使用して作成するプロジェクトファイルは、OSIV系システムでの開発でプログラムの翻訳・リンクするために使用するJCLに相当します。
- 開発資産の移行
  分散開発の目的が、新規開発ではなく、既存のプログラムのUP開発である場合、OSIV系システムから、これらの資産をWindows系システムに移行する作業が必要になります。
  - COBOLソースプログラム
  - COBOL登録集（COPY句）
  - 画面帳票定義体
  - オーバレイ定義体
- プログラミング
  COBOLエディタ等を使って、ソースファイル、COBOL登録集（COPY句）等を作成・更新します。
- 翻訳チェック
  プロジェクトのビルド機能を使用して、プロジェクトに含まれるCOBOLプログラムを翻訳リンクします。翻訳エラーがあった場合、エラージャンプ機能を使用して、翻訳エラーが発生したプログラムソース、COBOL登録集（COPY句）を開き、修正します。

3.2 プロジェクトの作成

NetCOBOLのプロジェクト管理機能は、プログラムの開発・保守を支援するさまざまな機能を含みます。OSIV系プログラムの分散開発では、そのうち、次のような機能を使用します。

- プログラムを構成するソースプログラム・登録集などのファイルの編集・管理
- NetCOBOLでのプログラムの翻訳・リンク
- OSIV系システムとのプログラム資源の移出入
- OSIV系システムでのプログラムの翻訳・リンク

これらの機能を使用するためには、プロジェクトファイルを作成して、次のような情報を設定する必要があります。

- 翻訳オプションおよびリンクオプション
- ソースプログラム名
- 登録集ファイル名
- 分散開発時固有の設定

3.2.1 基本的なプロジェクトファイルの作成

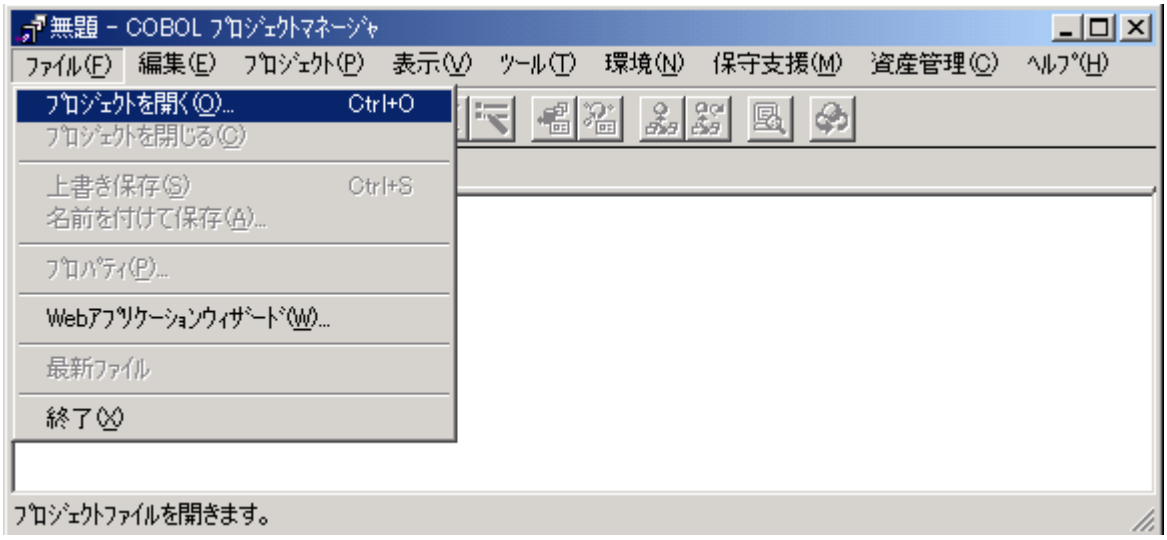
基本的なプロジェクトファイルの作成手順を説明します。

プロジェクトファイルの作成

プロジェクトファイルは、プロジェクト管理を行うための情報を登録するファイルで、1つのプロジェクトについて1つ必要です。以下に、プロジェクトファイルを作成する方法について説明します。

1. COBOLプロジェクトマネージャを起動して、〔ファイル〕メニューから“プロジェクトを開く”を選択します。

図3-1 プロジェクトファイルの作成



2. 〔ファイルを開く〕ダイアログが表示されますので、プロジェクトファイルを格納するフォルダに移動して、作成するプロジェクトファイルの名前を、ファイル名のエディットボックスに入力します。
3. その後、〔開く〕ボタンをクリックすると、新しい空のプロジェクトファイルが作成されます。

図3-2 作成されたプロジェクトファイル

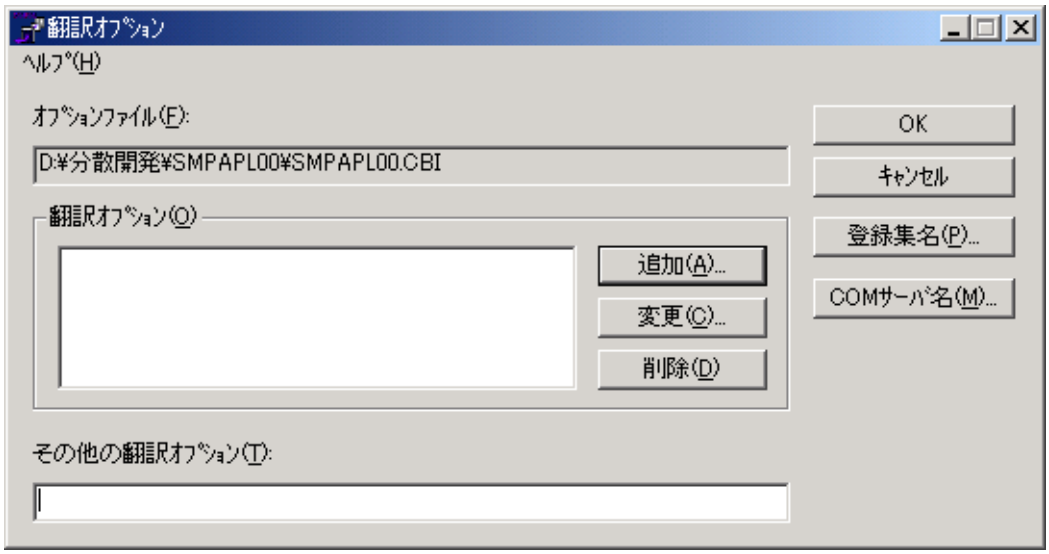
4. 作成されたプロジェクトは、そのままではCOBOLデバッガでデバッグ可能なプログラムを作成するようになっていません。〔プロジェクト〕－〔オプション〕メニューから “デバッグモジュール作成”を選択すると、メニューのこの項目がチェックされて、プロジェクトマネージャのツールバー上のモード表示が“リリース”から“デバッグ”に変わります。

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有効になる、翻訳オプションを指定します。指定した翻訳オプションは、翻訳オプションファイル(プロジェクト名.CBI)に格納され、ビルド制御文生成機能を使用して生成したJCL/CLISTに反映されます。
以下に、翻訳オプションの指定方法を示します。

1. 〔プロジェクト〕－〔オプション〕メニューから “翻訳オプション”を選択すると、〔翻訳オプション〕ダイアログが表示されます。

図3-3 翻訳オプションダイアログ



2. 〔追加〕ボタンをクリックすると、〔翻訳オプションの追加〕ダイアログが表示されます。
3. 〔翻訳オプションの追加〕ダイアログで必要な翻訳オプションの指定を追加します。必要となる翻訳オプションは次のものです。
   — OSIV系システムで翻訳する際に必要な翻訳オプション

— プログラム以外の資源の存在するパスを指定する翻訳オプション
  - LIBオプション:登録集ファイルの格納パスを指定します。
  - AIMLIBオプション:ネットワークデータベース機能を使用するプログラムで参照するサブスキーマ定義ファイルの格納パスを指定します。

4. 必要な翻訳オプションの追加が済んだら、〔翻訳オプションの追加〕ダイアログを閉じます。
5. 〔翻訳オプション〕ダイアログの〔OK〕ボタンをクリックして、翻訳オプションの設定は終了です。

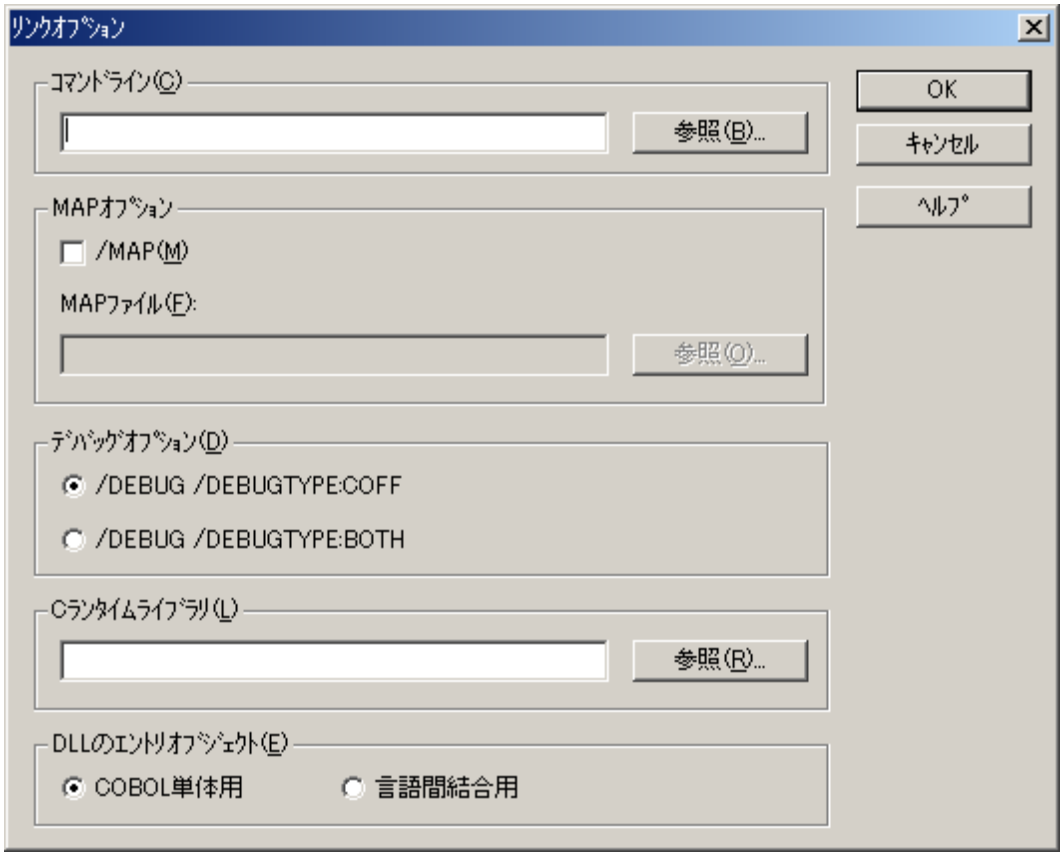
なお、翻訳オプションの指定は後からでも変更が可能です。

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指定します。指定したリンクオプションは、リンクオプションファイル(プロジェクト名.LNI)に格納されますが、ビルド制御文生成機能を使用して生成したJCL/CLISTに反映されません。翻訳チェックまでしか、NetCOBOLで行わない場合は特に指定する必要はありません。

1. 〔プロジェクト〕ー〔オプション〕メニューから “リンクオプション”を選択すると、〔リンクオプション〕ダイアログが表示されます。

図3-4 リンクオプションダイアログ



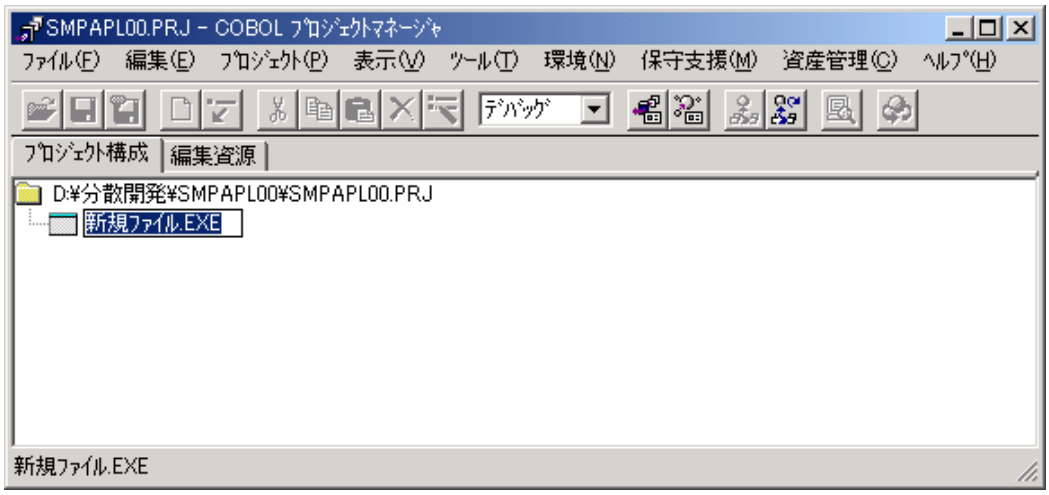
2. 図:〔リンクオプション〕ダイアログに示すように、設定されていることを確認します。

最終ターゲットファイルの追加

プロジェクトには最終ターゲットファイルとして、Windows系システムの実行形式ファイルかDLLを登録する必要があります。最終ターゲットファイルは複数指定可能ですが、分散開発を行う場合は、通常は実行形式ファイル1つにしてください。

1. COBOLプロジェクトマネージャの〔プロジェクト構成〕タブのツリービューで、プロジェクトファイルを選択して、〔編集〕メニューから“新規作成”を選択します。
2. ツリービューのプロジェクトファイルの配下にエディットコントロールが追加されるので、このエディットコントロールに最終ターゲットファイル名を入力します。

図3-5 追加された“最終ターゲットファイル”



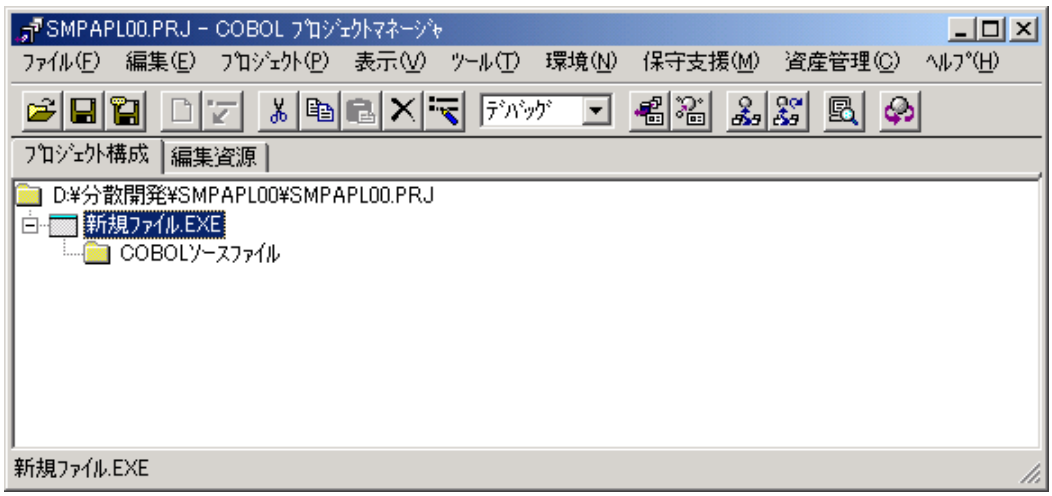
なお、最終ターゲットファイル名は、これをツリービュー上で選択して、〔編集〕メニューから“名前の変更”を選択することで、いつでも変更可能です。

COBOLソースファイルフォルダの作成とCOBOLソースファイルの追加

開発の対象となるCOBOLソースをプロジェクトに登録します。COBOLソースファイルを登録するためには、まず以下の手順で、“COBOLソースファイル”フォルダを作成します。

1. COBOLプロジェクトマネージャの〔プロジェクト構成〕タブのツリービューで、最終ターゲットファイルを選択します。
2. 〔編集〕ー〔フォルダ作成〕メニューから“COBOLソースファイル”を選択するとツリービューの最終ターゲットの配下に“COBOLソースファイル”という名前のフォルダが追加されます。

図3-6 追加された“COBOLソースファイル”フォルダ

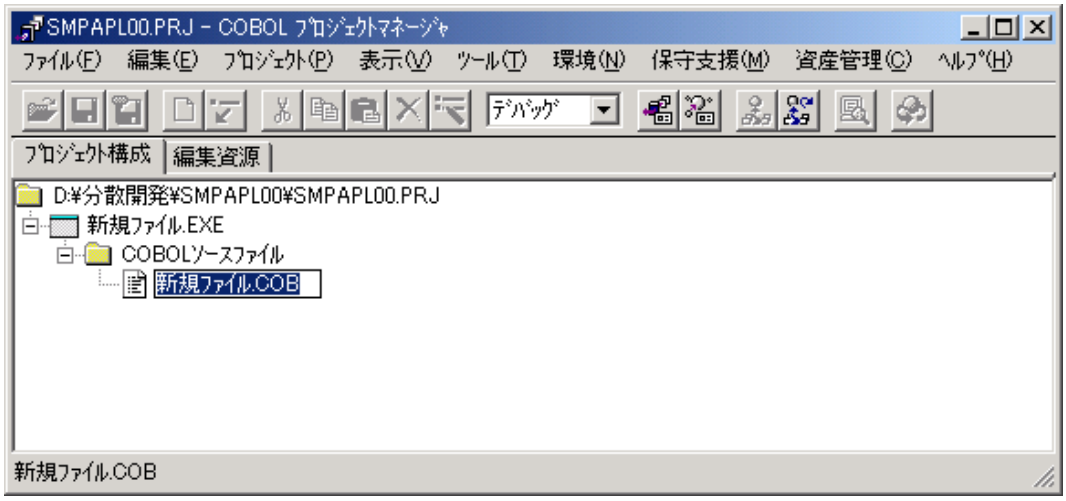


作成した“COBOLソースファイル”フォルダに以下の手順でCOBOLソースファイルを登録します。

3. この“COBOLソースファイル”フォルダを選択します。
4. 〔編集〕メニューから“新規作成”を選択するとツリービューのプロジェクトファイルの配下にエディットコントロールが追加されます。このエディットコントロールに開発の対

象となるCOBOLソースファイル名を入力します。

図3-7 新しいCOBOLソースファイルの登録



5. ファイルが既に存在するものなら、〔編集〕メニューから“追加”を選択して、〔ファイルの参照〕ダイアログを開いて、ダイアログで選択したファイルを登録することもできます。
6. COBOLソースファイル名は、これをツリービュー上で選択して、〔編集〕メニューから“名前の変更”を選択することで、いつでも変更可能です。

複数のCOBOLソースプログラムを登録するのであれば、3～5の手順を繰り返します。

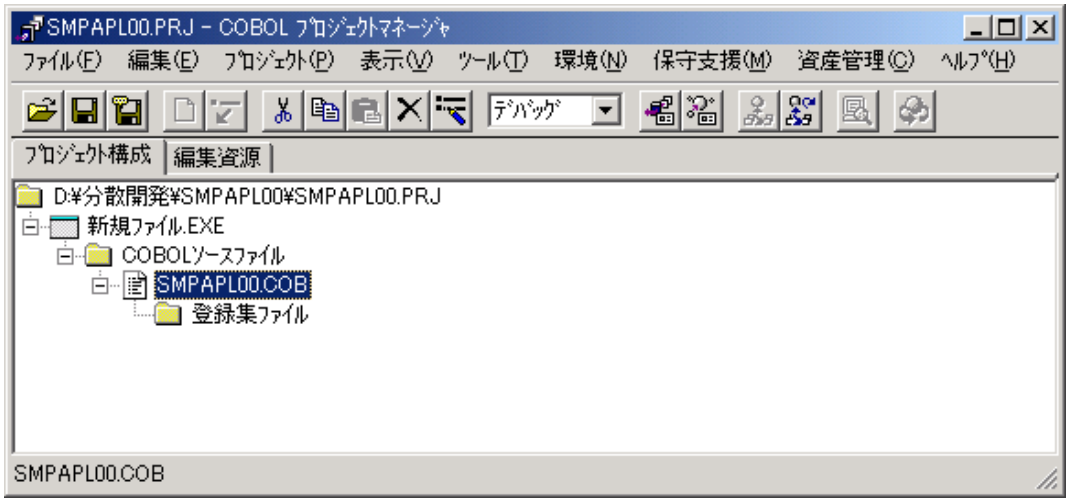
登録集ファイルフォルダの作成と登録集(COPY句)ファイルの追加

開発の対象となるCOBOLソースに依存関係を持つ登録集ファイル（COPY句）をプロジェクトに追加します。

まず、以下の手順で“登録集ファイル” フォルダを作成します。

1. COBOLプロジェクトマネージャの〔プロジェクト構成〕タブのツリービューで、COBOLソースファイルを選択します。
2. 〔編集〕－〔フォルダ作成〕メニューから“登録集ファイル”を選択するとツリービューのCOBOLソースファイルの配下に“登録集ファイル”という名前のフォルダが追加されます。

図3-8 追加された“登録集ファイル”フォルダ

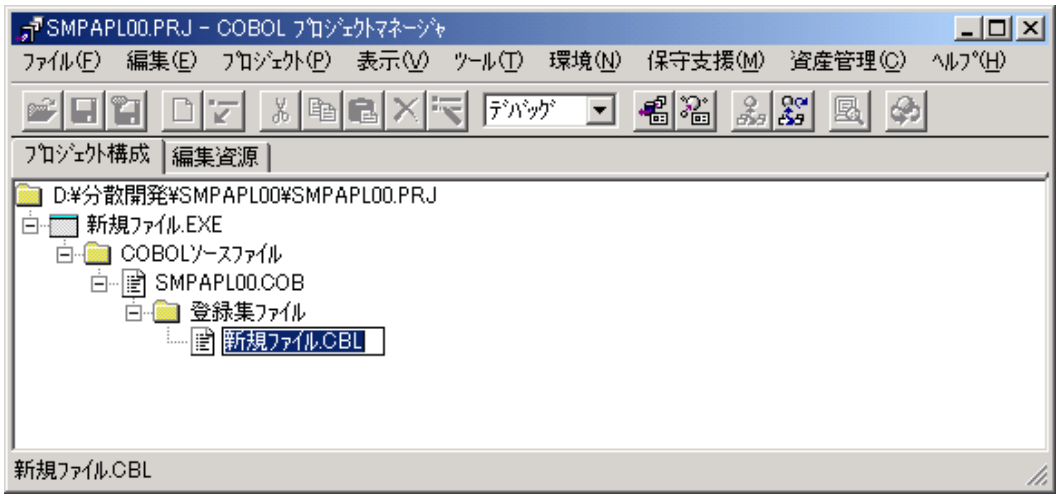


作成した“登録集ファイル”フォルダに以下の手順で登録集ファイルを登録します。

3. この“登録集ファイル”フォルダを選択します。

4. 〔編集〕メニューから“新規作成”を選択するとツリービューのプロジェクトファイルの配下にエディットコントロールが追加されます。このエディットコントロールに登録集ファイル名を入力します。

図3-9 新しい登録集ファイルの登録



5. ファイルが既に存在するものなら、〔編集〕メニューから“追加”を選択して、〔ファイルの参照〕ダイアログを開いて、ダイアログで選択したファイルを登録することもできます。
6. 登録集ファイル名は、これをツリービュー上で選択して、〔編集〕メニューから“名前の変更”を選択することで、いつでも変更可能です。

複数の登録集原文(COPY句)ファイル名を登録するのであれば、3～5の処理を繰り返します。

注意

ここでの登録集ファイルの登録はCOBOLプロジェクトマネージャの“ビルド／リビルド”機能に関係して必要となるCOBOLソースと登録集ファイルの依存関係を定義するために行います。このため、通常は新規作成・更新を行う登録集ファイルのみ登録します。

また、次の点に注意してください。

- 登録集の格納パス名の指定は翻訳オプション“LIB”を使用して、別途指定する必要があります。
- OSIVシステムとWindows系システムでファイルの受信・送信をするサーバ連携機能の使用とは無関係です。“登録集ファイル”フォルダを作成して、登録集ファイルを登録しておかなくとも、サーバ連携機能で登録集ファイルの受信・送信をすることは可能です。
- 開発中のCOBOLソースで参照するが、修正の必要のない登録集ファイルは、ここでの登録を行わず、翻訳オプション“LIB”で格納パスを指定するだけで十分です。

その他の資源の追加

その他に次のようなプログラム資産があれば、同じような操作でプロジェクトに登録することができます。

- 画面帳票定義体ファイル
  ソースファイル配下に“定義体ファイル”フォルダを作成し、登録します。
- サブスキーマ定義ファイル
  専用のフォルダはありません。必要なら“登録集ファイル”フォルダに登録してください。

主プログラムの設定

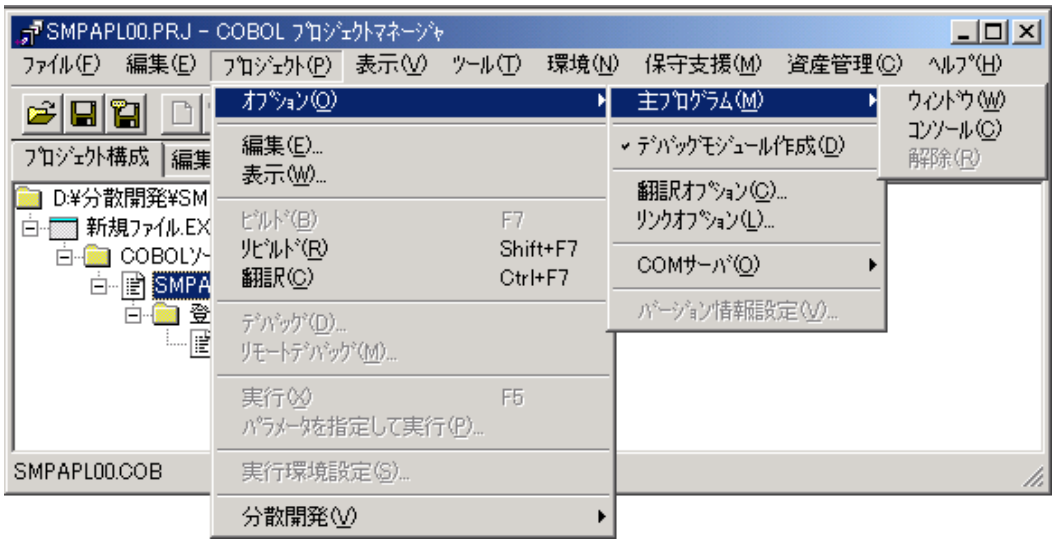
Windows系システムで単体テストを実施する場合で、次の条件に該当する場合は主プログラムの指定が必要です。

- OSIV系システムでJCL/CLISTで直接起動するプログラム
- 他のプログラムから呼び出されるプログラムのテスト用のドライバプログラム

以下の手順で、主プログラムの指定を実施します。

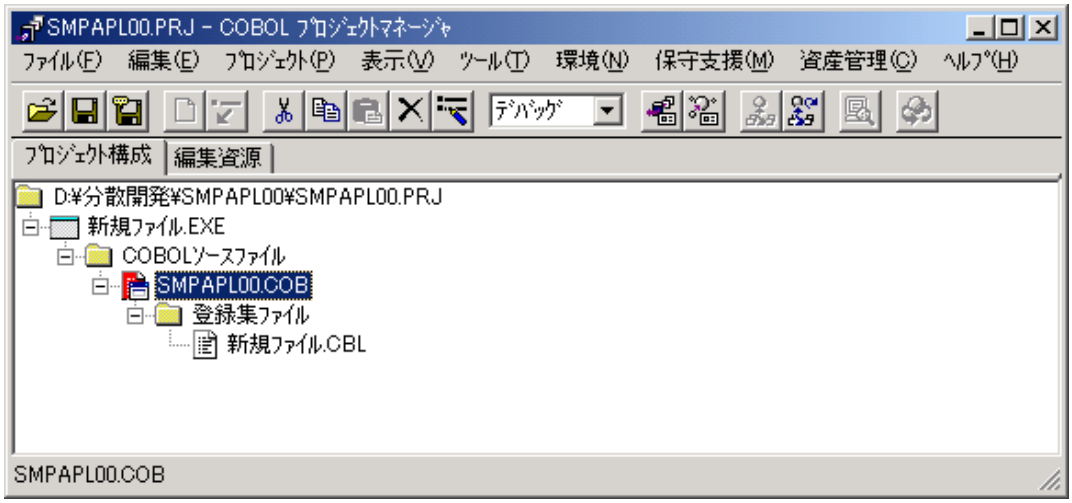
1. 主プログラムに設定するソースプログラムを選択します。
2. 〔プロジェクト〕－〔オプション〕－〔主プログラム〕メニューで“ウィンドウ”を選択します。

図3-10 主プログラムの指定



3. 主プログラムに指定されたCOBOLソースファイルのアイコンの色が変わります。

図3-11 設定された主プログラム



3.2.2 分散開発時固有の設定

NetCOBOLを用いて、OSIV系プログラムの分散開発を行う場合、さらにプロジェクトのプロパティで分散開発の設定をします。これにより次のような機能が使用可能となります。

- OSIV系システムからWindows系システムへのファイルの受信
- OSIV系システムへのWindows系システムからのファイルの送信
- ビルド制御文生成(OSIV系システムでの翻訳・リンク用JCLの雛型生成)
- ターゲットビルド(OSIV系システムでの翻訳・リンク)

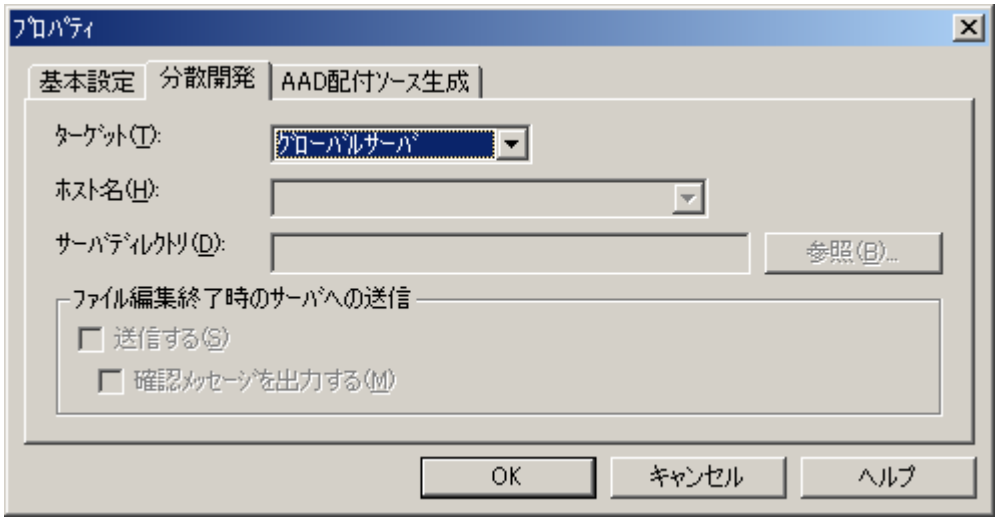
- AAD配布ソース生成機能（AADアプリケーションの開発時のみ使用）

なお、AADアプリケーションの開発に関する設定については、ここでは説明しません。それについては“第6章 CORBAアプリケーションの分散開発”を参照してください。

以下、プロジェクトのプロパティに分散開発のための情報を設定する方法について説明します。

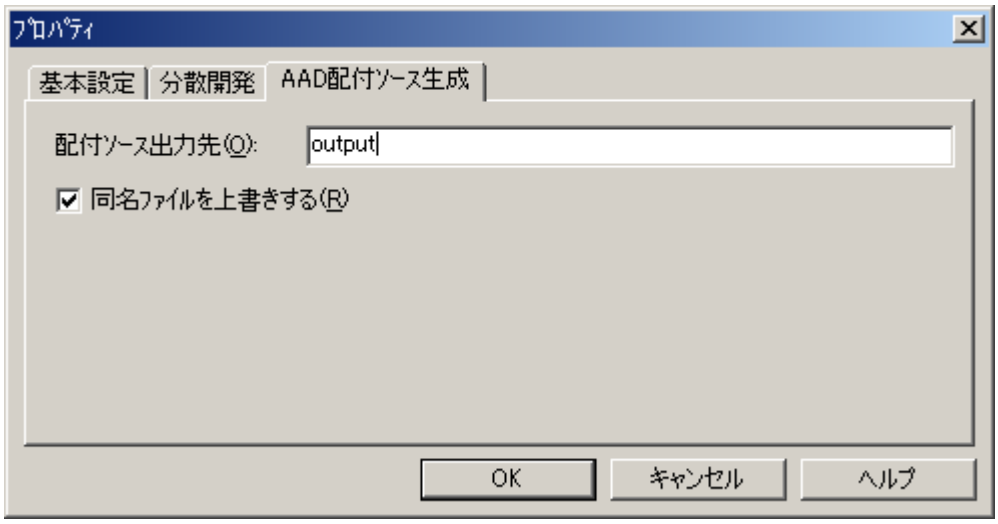
1. COBOLプロジェクトマネージャの〔プロジェクト構成〕ページのツリービューで、プロジェクトファイルを選択して、〔ファイル〕メニューから“プロパティ”を選択するとプロジェクトの“プロパティ”ダイアログボックスが表示されます。
2. 〔基本設定〕のページでプロジェクトのプロパティについて次の項目を設定します。
   - 分散開発:
     チェックボックスで“グローバルサーバ”を選択します。使用されるホストは“サーバ連携情報”で設定したホストが自動的に選択されます。その他の項目は設定する必要はありません。
   - 実行時のコード系:
     通常は“シフトJIS”を選択します。
     NetCOBOL JEFオプションがインストール済みなら、チェックボックスの“JEF”が選択可能になります。JEFオプションを使用するのであれば“JEF”を選択してください。

図3-12 〔プロパティ〕ダイアログの〔基本設定〕ページ設定例



3. AADアプリケーションの開発を行うのであれば、〔AAD配布ソース生成〕のページの設定をします。デフォルトでは、“図:〔プロパティ〕ダイアログの〔AAD配布ソース生成〕ページの初期状態”のようになっています。必要なら、次の設定を変更してください。
   - 配布ソース出力先:
     グローバルサーバに配付するためのソースファイルの出力先フォルダ名を指定します。フォルダはプロジェクトのサブフォルダ名を指定します。
     省略することはできません。また、指定した名前のフォルダが存在しないなら、新しいフォルダが作成されます。
   - 同名ファイルを上書きする:
     配布ソースファイル生成時に同名のファイルを上書きするかどうか指定します。

図3-13 〔プロパティ〕ダイアログの〔AAD配布ソース生成〕ページの初期状態



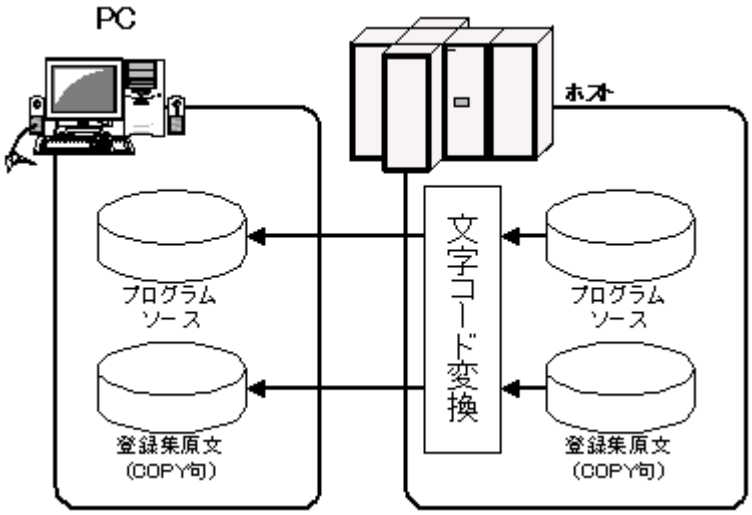
3.3 プログラム資産のＰＣへの移行

OSIV系システム上に存在するプログラム資産を基に分散開発を行う場合、これらの資産をWindows系システムに移行する作業が必要になります。OSIV系システムからWindows系システムにファイルを転送する方法はいくつかありますが、ここではCOBOLプロジェクトマネージャの“受信”機能を使用するやり方を説明します。

3.3.1 COBOLソース・登録集原文の移行

COBOLソース・登録集原文(COPY句)は、OSIV系システムでもWindows系システムでもテキスト形式のファイルであるため、比較的容易に移行できます。
システムで採用する文字コード系は、OSIV系システムではEBCDIC/JEF、Windows系ではASCII/SJISと異なりますが、ファイルを転送する過程で文字コードの変換も自動的に行われます。

図3-14 OSIV系システムとWindows系システムにおけるソース・登録集ファイル



3.3.1.1 OSIV系システムでの処理

COBOLソース・登録集原文をWindows系システムに移行するために、次の場合を除きOSIV系システムで準備が必要になることはありません。

注意

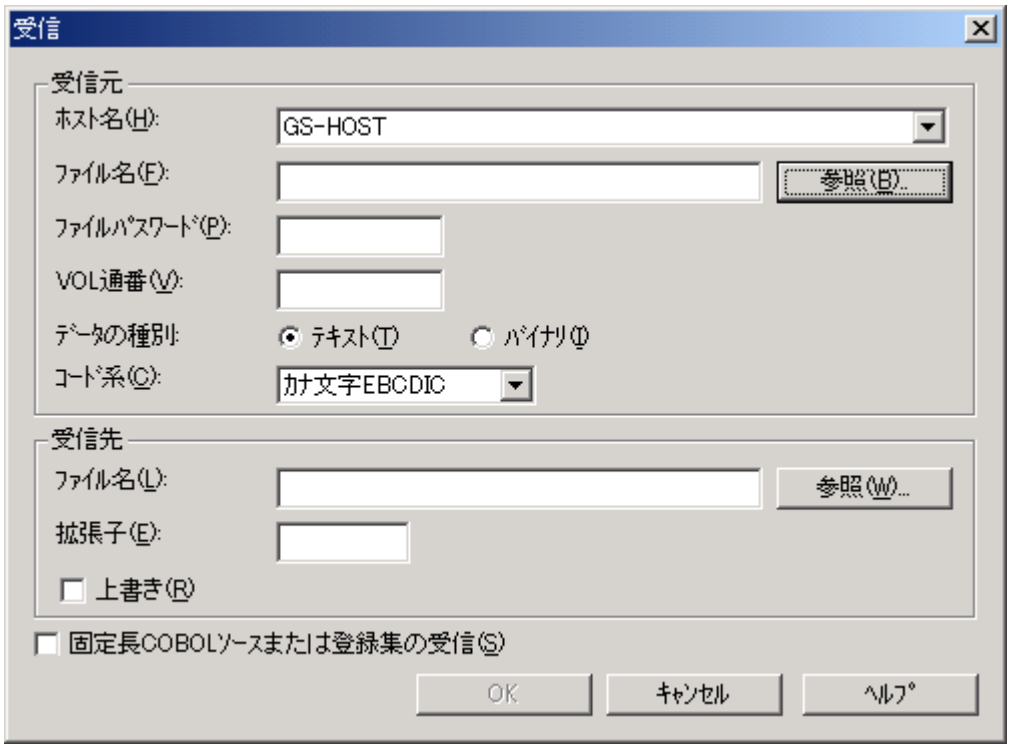
ソース・登録集原文の格納データセットがF形式の場合、〔受信〕機能で扱うことができません。FB形式のデータセットに複製しておいてください。

3.3.1.2 Windows系システムでの処理

COBOLプロジェクトマネージャの分散開発支援機能の“受信”機能を使用して、COBOLソース・登録集原文をOSIV系システムのデータセットからWindows系システムのファイルに受信します。
以下、その手順を説明します。

1. COBOLプロジェクトマネージャの〔プロジェクト〕－〔分散開発〕メニューから、“受信”を選択します。

図3-15 〔受信〕ダイアログ

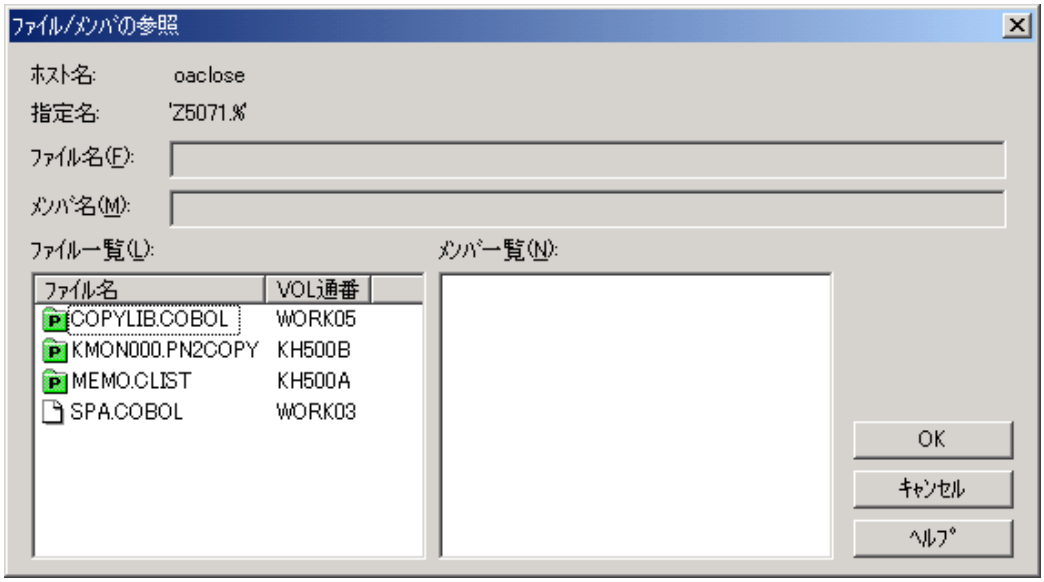


2. “受信”ダイアログが表示されるので、〔受信元〕および〔受信先〕に必要な情報を設定します。
3. 〔受信元〕については少なくとも以下の情報を設定します。
   - ファイル名:
     格納データセットを指定します。順編成データセット(PS形式)あるいは区分編成データセット(PO)形式が指定できます。区分編成データセットの場合は次のいずれかの方法で指定できます。
     - メンバ指定なし
       区分データセット内の全メンバが転送の対象となります。
     - メンバ指定あり
       ファイル名に続いてメンバ名を()で囲んで指定します。また、メンバ名を複数指定する場合は、メンバ名をブランクで区切って指定します。

     例:SRCWK.COBOL(SMPAPL00)
     CPYLIB.COBOL(SPA MENUREC HOKKAIIN HONSHUIN SHIKOKIN KYUSHUIN)

     “参照”ボタンを押すと“ファイル／メンバの参照”ダイアログが開きますので、ここから対象となるファイルを選択することもできます。

図3-16 〔ファイル／メンバの参照〕ダイアログ



— データの種別:
COBOLソース・登録集原文の場合は、必ずテキストを選択します。
— コード系:
受信元のファイルのコードを設定します。

以下の情報については、必要な場合のみ指定してください。

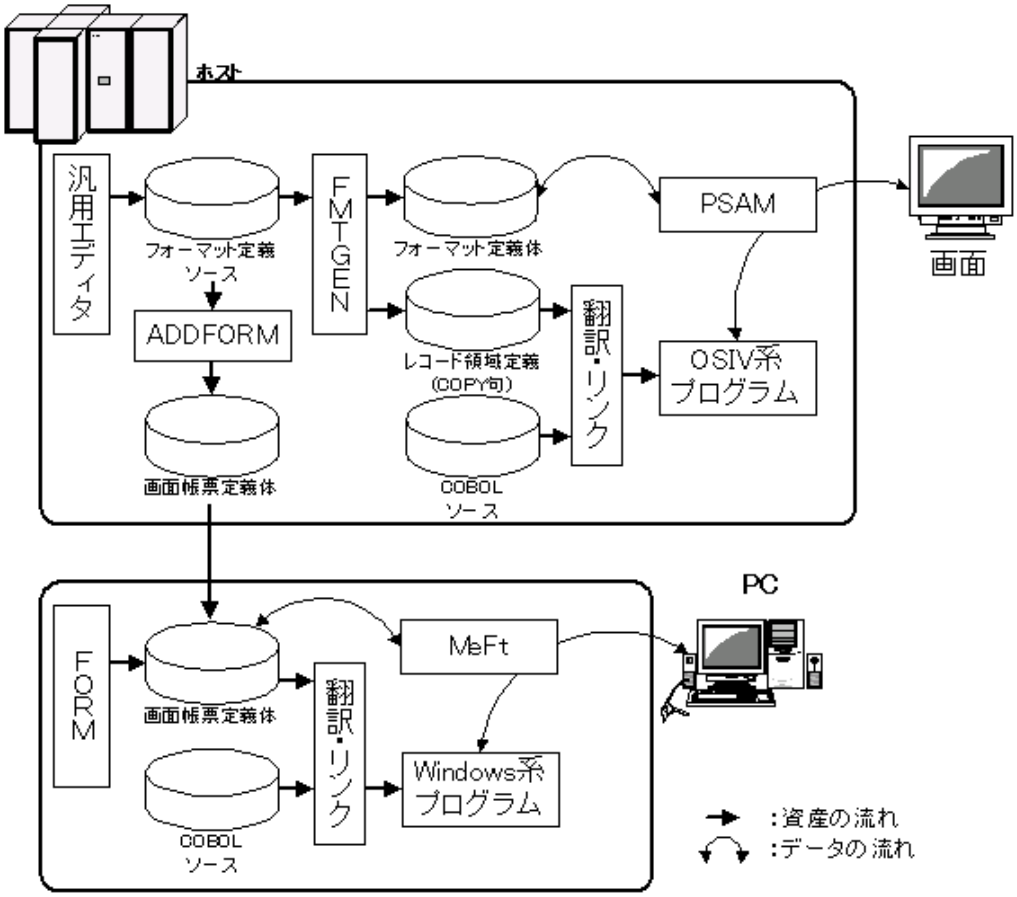
— ファイルパスワード:
受信元のファイルがパスワード保護されている場合、そのファイルパスワードを指定します。
— VOL通番:
受信元のファイルがカタログされていない場合、そのボリューム通し番号を指定します。
〔受信先〕については少なくとも以下の情報を設定します。
— ファイル名:
Windows系システム上での格納ファイル名を指定します。
— 拡張子:
NetCOBOLはデフォルトでは、拡張子COBのファイルをCOBOLソースファイル、拡張子CBLのファイルを登録集原文ファイルと見なします。COBOLソースならCOBと、登録集原文ならCBLと指定してください。

4. 対象となる受信元ファイルが固定形式である場合、“固定長のCOBOLソースまたは登録集の受信”を必ずチェックしてください。
5. 以上の設定が済んだら、“OK”ボタンを押して、ファイルの受信を開始します。

3.3.2 フォーマット定義体の移行

画面あるいは帳票を宛先とする表示ファイル機能では、画面または帳票のフォーマットを定義する定義体を必要とします。この定義体の形式と使用法は、OSIV系システム用のもの(フォーマット定義体)とWindows系用のもの(画面帳票定義体)では異なります。

図3-17 OSIV系システム:フォーマット定義体とWindows系システム:画面帳票定義体の流通



OSIV系システムでは、テキストファイルとして作成したフォーマット定義体ソースから、プログラムの翻訳時に参照するレコード領域定義(COPY句)とプログラムの実行時に参照するフォーマット定義体を生成して使用します。
Windows系システムでは、FORM/PowerFORMで作成して、翻訳時にも実行時にも参照する資源となる画面帳票定義体を使用します。
OSIV系システムのフォーマット定義体を移行する場合、フォーマット定義体ソースから移出機能(ADDFORM)を使用して、画面帳票定義体に変換してから、それを移行する必要があります。

注意

PSAMとMeFtの機能差があります。このため、フォーマット定義体ソースの移出機能によって生成した画面帳票定義体は、同じソースから生成されるフォーマット定義体と完全には同じものにならない場合があります。

3.3.2.1 OSIV系システムでの処理

フォーマット定義体ソースから移出機能(ADDFORM)を使用して、画面帳票定義体への変換を行います。
以下、移出機能を使用するために必要なデータセットおよびADDFORM制御文について、説明します。

フォーマット定義体の移出機能の使用するデータセット

フォーマット定義体の移出機能を使用する場合の入出力データとそのファイル属性を示します。

図3-18 フォーマット定義体の移出機能における入出力の流れ

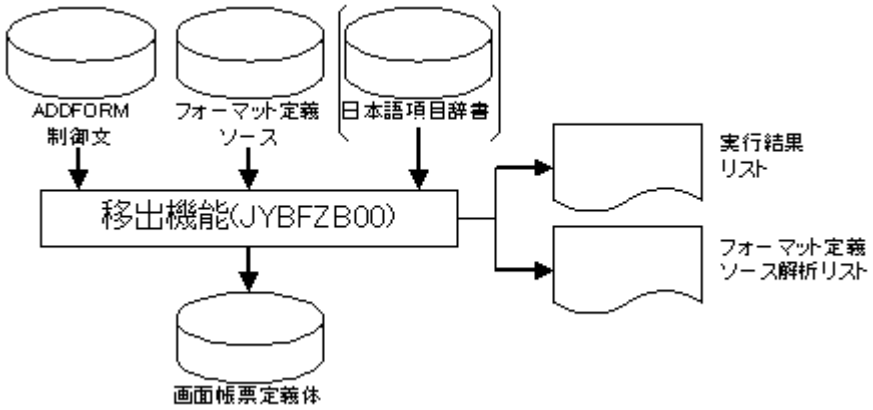


表3-1 フォーマット定義体の移出機能におけるファイルの属性

ファイル種別	指定方法（DD名またはアクセス名）	ファイル属性			
		編成	レコード形式	レコード長（バイト）	ブロック長（バイト）
フォーマット定義ソースライブラリ	ＳＯＣＬＩＢ	区分編成	Ｆ、ＦＢ	80～255	レコード長×ｎ
			Ｖ、ＶＢ	84～255	レコード長＋４以上
画面帳票定義体ライブラリ	任意	順編成 区分編成	Ｆ、ＦＢ	1～32760	レコード長×ｎ
			Ｖ、ＶＢ	5～32760	レコード長＋４以上
リスト出力先（実行結果リスト）	ＳＹＳＰＲＩＮＴ	順編成 区分編成	ＦＡ ＦＢＡ	255以上	レコード長×ｎ
			ＶＡ ＶＢＡ	255以上	レコード長＋４以上
リスト出力先（フォーマット定義ソース解析リスト）	ＳＹＳＬＩＳＴ	順編成 区分編成	ＦＡ ＦＢＡ	255以上	レコード長×ｎ
			ＶＡ ＶＢＡ	255以上	レコード長＋４以上
日本語項目辞書	ＪＩＭＬＩＢｎ（ｎ＝１～５）	ＩＳＡＭ ＶＳＡＭ 順編成	※OS毎のADJUSTの使用手引書を参照してください		

日本語項目辞書は通常は必要ありません。フォーマット定義ソース中に日本語を日本語項目定数として記述した場合にのみ指定してください。

ADDFORM制御文の記述形式

ADDFORM制御文の記述形式を示します。

表3-2 ADDFORM制御文の記述形式

制御	命令	オペランド

文字		
－	ＡＤＤＦＯＲＭ	ＭＥＭＢＥＲ＝メンバ名 ＯＵＴＦＩＬＥ＝ＤＤ名 〔ＵＳＥＲ＝利用者名 ＰＡＳＳＷＯＲＤ＝パスワード〕 〔｛ＲＥＰ ｝〕 ＮＯＲＥＰ 〔ＲＥＮＡＭＥ＝｛ＹＥＳ ｝〕 ＮＯ 〔ＥＱＵ〕

各オペランドの意味と指定方法は次の通りです。

1. ＭＥＭＢＥＲ＝メンバ名
   フォーマット定義ソースのメンバ名を以下のいずれかの形式で指定します。
   - メンバ名:
     指定したメンバだけを処理の対象とします。
   - 文字列＋:
     フォーマット定義ソースライブラリ内に存在するメンバのうち、メンバ名の先頭から指定文字列と一致するメンバを処理の対象とし、選択されたメンバを連続的に処理します。
   - ＋:
     フォーマット定義ソースライブラリ内に存在するすべてのメンバを処理の対象とします。
     複数メンバを処理する指定をしても、画面帳票定義体の出力先が順編成データセットの場合、対象となる一連のメンバの先頭しか処理されません。また、複数メンバ処理中にエラーが発生した場合は、そのメンバの処理を打ち切り、次のメンバを処理します。
2. ＯＵＴＦＩＬＥ＝ＤＤ名
   画面帳票定義体の出力先であるデータセットのDD名を指定します。画面帳票定義体が区分編成データセットである場合、メンバ名は、フォーマット定義ソースのフォーマットIDと同じになります。
3. ＵＳＥＲ＝利用者名
   DSMサブシステムの利用者管理簿に登録されている利用者のユーザ識別名を指定します。
4. ＰＡＳＳＷＯＲＤ＝パスワード
   RACFが導入されているシステムではRACFに登録されたパスワードを、RACFが導入されていないシステムではDSMサブシステムの利用者管理簿に登録されている利用者のユーザ識別名に対応するパスワードを指定します。
5. ｛ＲＥＰ｜ＮＯＲＥＰ｝
   出力先のキャビネットに、同一オブジェクトが存在したとき、または出力先の区分編成データセットに、同一メンバが存在したときの処理を指定します。
   - ＲＥＰ:
     置き換えます。
   - ＮＯＲＥＰ:
     置き換えません。

   画面帳票定義体の出力先が、順編成データセットの場合、このオペランドの指定に関係なく、置き換えます。
6. ＲＥＮＡＭＥ＝｛ＹＥＳ｜ＮＯ｝

画面帳票定義体の項目名および項目群名の扱いを指定します。

— ＹＥＳ：
システムが自動的に6文字の名前を作成します。

— ＮＯ：
DATA文、FIELD文およびGROUP文で指定した名札とします。ただし、名札が6文字以上の場合、システムが自動的に6文字の名前を作成します。

7. ＥＱＵ
フォーマット定義ソース解析リストにEQU文を含めるかどうかを指定します。本オペランドを省略した場合、EQU文は印刷されません。

より詳細な情報は“PSAM使用手引書 V20L10 付録I ホスト資産の流用”を参照してください。

ADDFORMの起動パラメタ

ADDFORMの起動パラメタは、MSPではEXEC文のPARMパラメタで指定し、XSPではPARA文で指定します。起動パラメタには以下のものがあります。

表3-3 ADDFORMの起動パラメタ

指定形式	説明
ＷＫＳＺ＝｛３００｜ｎ｝	移出機能が使用する作業領域の大きさをキロバイト単位で指定します(1≦n≦16000)。
ＬＩＮＥＣＴ＝｛６０｜ｎ｝	実行結果リストおよびフォーマット定義ソース解析リストの1ページあたりの印刷行数を指定します(10≦n≦999)。
ＭＥＤＳＩＺＥ＝｛６４｜ｎ｝	移出機能が生成する画面帳票定義体の大きさを指定します(1≦n≦320)。本パラメタで指定した値を超える画面帳票定義体は作成できません。

ADDFORMの起動ジョブのJCL

ADDFORMを起動するジョブのJCLの例を示します。

図3-19 MSPでのADDFORM起動ジョブのJCL例

```
//USER1ADF  JOB ,…
//ADDFORM   EXEC PGM=JYBFZB00,REGION=4096K
//SOCLIB    DD DSN=USER1.FMTGEN.DATA,DISP=SHR
//MEDLIB    DD DSN=USER1.MEDLIB.DATA,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSLIST   DD SYSOUT=*
//SYSIN     DD *
-ADDFORM MEMBER=+,OUTFILE=MEDLIB,REP
/*
```

図3-20 XSPでのADDFORM起動ジョブのJCL例

```
¥ JOB USER1ADF
¥ EX JYBFZB00,RSIZE=4096
¥ FD SOCLIB=DA,FILE=USER1.FMTGEN.DATA
¥ FD MEDLIB=DA,FILE=USER1.MEDLIB.DATA
¥ FD SYSPRINT=DA,VOL=WORK,BLK=(13030*50,50),SOUT=A
¥ FD SYSLIST=DA,VOL=WORK,BLK=(13030*50,50),SOUT=A
¥ FD SYSIN=*
-ADDFORM MEMBER=+,OUTFILE=MEDLIB,REP
 ¥ JEND
```

ADDFORMの復帰コード

ADDFORMの復帰コードを示します。

表3-4 ADDFORMの起動パラメタ

状態	復帰値		説明
	ＭＳＰ	ＸＳＰ	
正常	０	１０	画面帳票体を正常に作成し、データセットに出力した。
警告	４	２０	軽度のエラーがあったが、画面帳票体を作成し、データセットに出力した。
異常	８	３０	重大なエラーが発生したため、画面帳票体を作成できなかった。またはデータセットに出力できなかった。

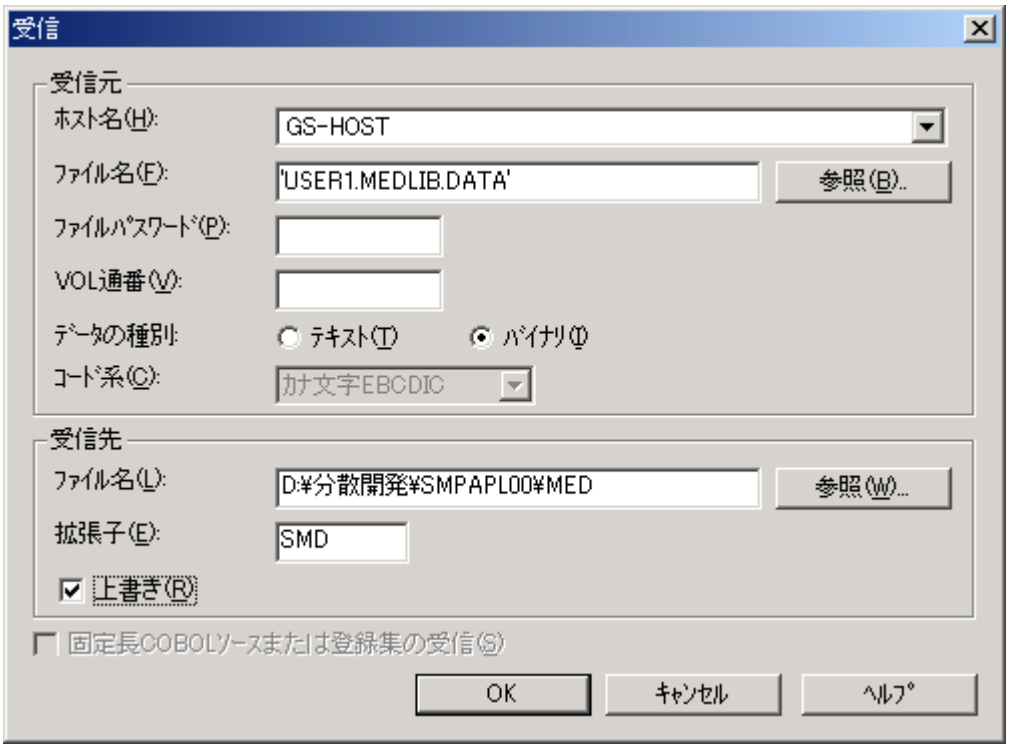
3.3.2.2 Windows系システムでの処理

COBOLプロジェクトマネージャの分散開発支援機能の“受信”機能を使用して、フォーマット定義体の移出機能で作成した画面帳票定義体をOSIV系システムのデータセットからWindows系システムのファイルに受信します。

基本的な手順は、COBOLソース・登録集原文を移行する場合と同じです。ここでは、異なる設定が必要となる項目のみ説明します。

1. 〔受信元〕は次のように設定してください。
   - データの種別:
     画面帳票定義体の受信には、必ず“バイナリ”を選択します。
2. 〔受信先〕は次のように設定してください。
   - 拡張子:
     NetCOBOLはデフォルトでは、拡張子SMDのファイルを画面帳票定義体ファイルと見なします。ここは、SMDと指定してください。

図3-21 画面帳票定義体の移行時の〔受信〕ダイアログの設定例



3. 〔OK〕ボタンをクリックすると受信処理を開始します。

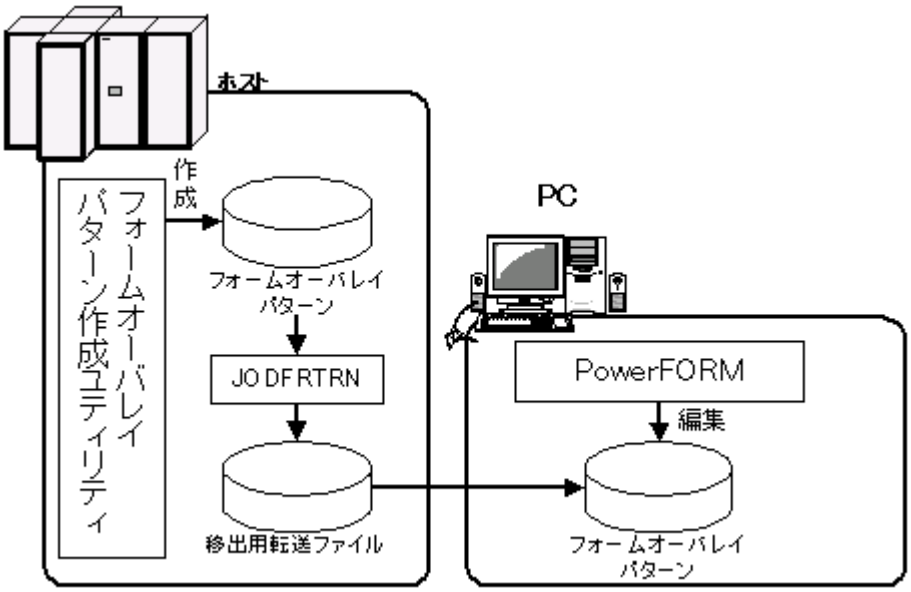
3.3.3 オーバレイ定義体の移行

帳票印刷機能使用時に、罫線や見出し文字などの帳票の固定的な部分を、予め定義しておき他の印刷物(可変部分)と重合わせて印刷するためにオーバレイ定義体(フォームオーバレイパターン)が使用されます。OSIV系システムで使用するオーバレイ定義体は、次の2種類があります。

- NLP用オーバレイパターン
- JEF/AP用オーバレイパターン

これらとWindows系システムで利用可能なオーバレイ定義体とでは、機能の一部や形式に相違があります。このため、OSIV系システムで開発・運用してきたフォームオーバレイパターンをWindows系システムに移行するためには、ツールを使用して形式を変換する必要があります。

図3-22 OSIV系システムとWindows系システムのオーバレイ定義体の流通



3.3.3.1 OSIV系システムでの処理

フォームオーバレイパターンから印刷資源流通用のユーティリティ(JODFRTRN)を使用して、移出転送用のファイルへの変換を行います。
以下、このユーティリティを使用するために必要なデータセットおよびJODFRTRN制御文について、説明します。

印刷資源流通用のユーティリティの使用するデータセット

印刷資源流通用のユーティリティを使用する場合の入出力データとそのファイル属性を示します。

図3-23 印刷資源流通用ユーティリティにおける入出力の流れ

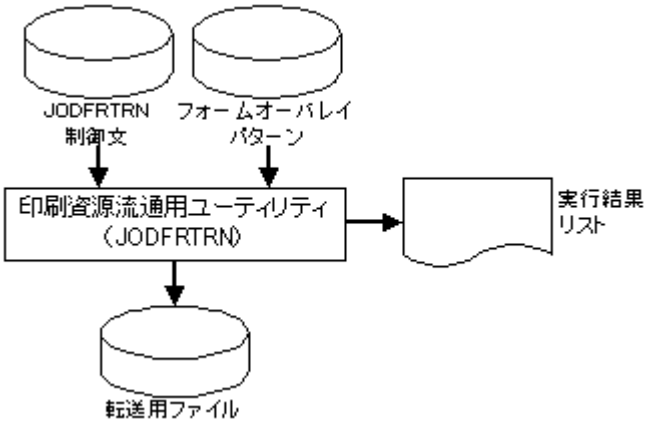


表3-5 印刷資源流通用ユーティリティにおけるファイルの属性

ファイル種別	指定方法（ＤＤ名またはアクセス名）	ファイル属性			
		編成	レコード形式	レコード長（バイト）	ブロック長
JODFRTRN制御文	ＳＹＳＩＮ	順編成	Ｆ、ＦＢ	80	－
イメージライブラリ（フォームオーバレイパターン格納）	ＳＹＳＩＭＡＧＥ	区分編成	Ｕ	－	1024以上
リスト出力先（実行結果リスト）	ＳＹＳＰＲＩＮＴ	順編成	ＶＢ	80	－
転送用ファイル	任意	順編成	Ｆ、Ｖ	80	

JODFRTRN制御文の記述形式

JODFRTRN制御文の記述形式を示します。制御文は移行するフォームオーバレイパターンがNLP用のものか、JEF/AP用のものかで異なります。

NLP用フォームオーバレイパターンのJODFRTRNの制御文

表3-6 JODFRTRN制御文(NLP用)の記述形式

命令	オペランド
ＥＸＰＴＯＶＬＮ	ＴＦＩＬＥ(転送用ファイルＤＤ／ＦＤ名) ＮＡＭＥ(ＮＬＰ用オーバレイパターン名)

各オペランドの意味と指定方法は次の通りです。

1. ＴＦＩＬＥ(転送用ファイルＤＤ／ＦＤ名)
   転送ファイルに割り当てられたDD/FD名(MSP系:DD名、XSP:FD名)を指定します。
   転送ファイルには、オーバレイパターンまたはオーバレイパターングループが格納されます。
   なお、本オペランドを省略することはできません。
2. ＮＡＭＥ(ＮＬＰ用オーバレイパターン名)
   イメージライブラリに格納したオーバレイパターンまたはオーバレイパターングループの名前を指定します。本オペランドを省略することはできません。
   これは次のように指定します。

— NAME(KOL1xxxx):
xxxxには4文字以内の英数字(オーバレイ識別子)を指定します。

JEF/AP用フォームオーバレイパターンのJODFRTRNの制御文

表3-7 JODFRTRN制御文(JEF/AP用)の記述形式

命令	オペランド
ＥＸＰＴＯＶＬＥ	ＴＦＩＬＥ(転送用ファイルＤＤ／ＦＤ名) ＮＡＭＥ(ＪＥＦ／ＡＰ用オーバレイパターン名)

各オペランドの意味と指定方法は次の通りです。

1. ＴＦＩＬＥ(転送用ファイルＤＤ／ＦＤ名)
   転送ファイルに割り当てられたDD/FD名(MSP系:DD名、XSP:FD名)を指定します。
   転送ファイルには、オーバレイパターンまたはオーバレイパターングループが格納されます。
   なお、本オペランドを省略することはできません。
2. ＮＡＭＥ(ＪＥＦ用オーバレイパターン名)
   イメージライブラリに格納したオーバレイパターンまたはオーバレイパターングループの名前を指定します。本オペランドを省略することはできません。
   これは次のように指定します。
   — NAME(KOL5xxxx):
   xxxxには4文字以内の英数字(オーバレイ識別子)を指定します。

より詳細な情報は“ADJUST使用手引書 印刷資源連携編 V12用”を参照してください。

JODFRTRNの起動パラメタ

JODFRTRNに起動パラメタは、特にありません。ただし、実行時のリージョンサイズとして、1024Kb以上を割り当ててください。

JODFRTRNの起動ジョブのJCL

JODFRTRNを起動するジョブのJCLの例を示します。

図3-24 MSPでのJODFRTRN起動ジョブのJCL例

```
//USER1ADF  JOB ,…
//OVLCNV  EXEC PGM=JODFRTRN,REGION=4096K
//SYSIMAGE  DD DSN=USER1.OVLIMAGE.LIB,DISP=SHR
//OVLDATA   DD DSN=USER1.KOL50021.DATA,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
EXTOVLE TFILE(OVLDATA),NAME(TOL50021)
/*
```

図3-25 XSPでのJODFRTRN起動ジョブのJCL例

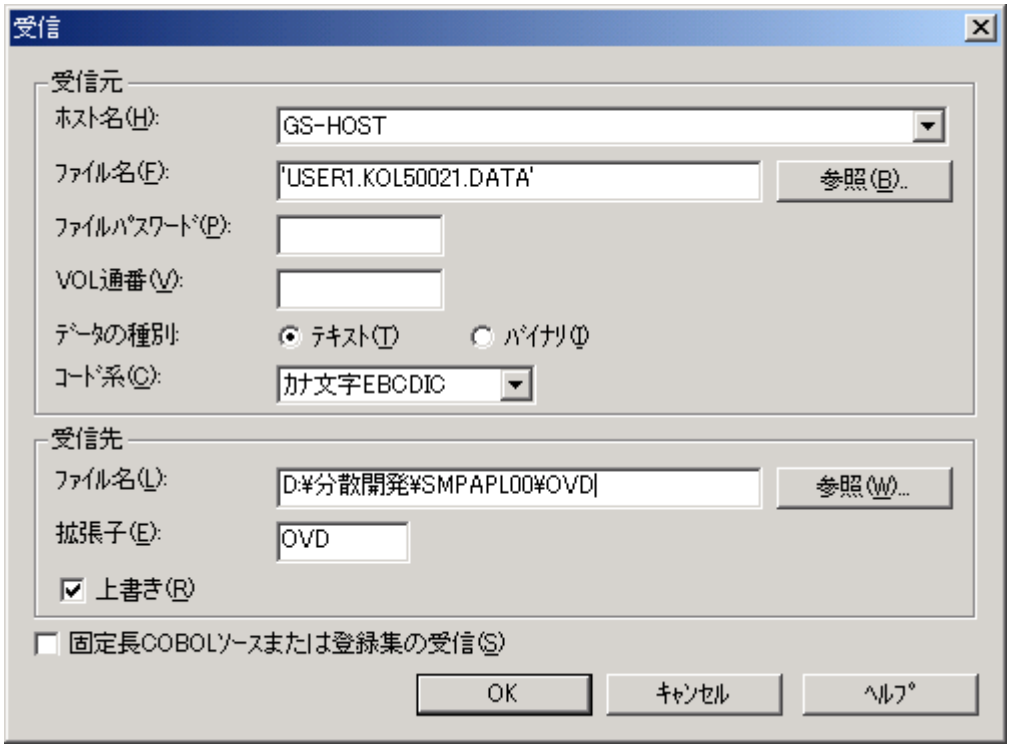
```
¥ JOB USER1ADF
¥ EX JODFRTRN,RSIZE=4096
¥ FD SYSIMAGE=DA,FILE=USER1.OVLIMAGE.LIB
¥ FD OVLDATA=DA,FILE=USER1.KOL50021.DATA
¥ FD SYSPRINT=DA,VOL=WORK,TRK(1,1),SOUT=A
¥ FD SYSIN=*
EXTOVLE TFILE(OVLDATA),NAME(TOL50021)
¥ JEND
```

3.3.3.2 Windows系システムでの処理

COBOLプロジェクトマネージャの分散開発支援機能の“受信”機能を使用して、転送用ファイルをOSIV系システムのデータセットからWindows系システムのファイルに受信します。
基本的な手順は、フォーマット定義体を移行する場合と同じです。ここでは、異なる設定が必要となる項目のみ説明します。

1. 〔受信元〕は次のように設定してください。
   - データの種別:
     オーバレイ定義体の受信には、必ず“バイナリ”を選択します。
2. 〔受信先〕は次のように設定してください。
   - 拡張子:
     NetCOBOLはデフォルトでは、拡張子OVDのファイルをオーバレイ定義体ファイルと見なします。ここは、OVDと指定してください。

図3-26 オーバレイ定義体の移行時の〔受信〕ダイアログの設定例

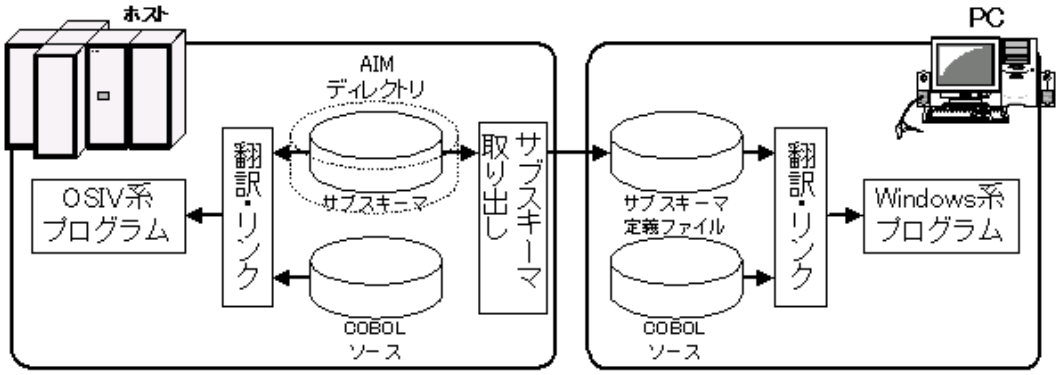


3. 〔OK〕ボタンをクリックすると受信処理を開始します。

3.3.4 サブスキーマの移行

ネットワークデータベース機能では、プログラムによって操作可能なデータベースの論理構造を必要とします。OSIV系システムでは、SUBSCHEMAコマンドで定義し、AIMディレクトリに格納したサブスキーマを参照することで、これが可能になります。プログラムの翻訳時にAIMを介して、サブスキーマの情報は取り出され、AIMとの連絡領域(FCOM)とデータベースとのデータのやり取りをするレコード(UWA)として、プログラム内に展開(AIM展開レコード)されます。

図3-27 OSIV系システムのサブスキーマとWindows系システムへの流通



Windows系システムにはAIMは存在しないため、AIMディレクトリもサブスキーマもそのままでは移行することはできません。そこで、OSIV系システム上でサブスキーマ取り出しツール(GETSSCH)を用いてサブスキーマから情報を取り出し、それをCOBOLのレコード定義に展開したもの(以降、サブスキーマ定義ファイルと呼びます)を用います。

3.3.4.1 OSIV系システムでの処理

AIMディレクトリからサブスキーマを取り出すツール(GETSSCH)を使用して、サブスキーマ定義ファイルへの変換を行います。
以下、サブスキーマ取り出しツール(GETSSCH)の準備、データセットや使用法について説明します。

GETSSCHの準備

サブスキーマ取り出しツール(GETSSCH)は、OSIV系システムで動作するものですが、NetCOBOLの製品の一部として提供されています。まず、それをOSIV系システムに転送し、実行可能なものを準備する必要があります。
以下、その手順を説明します。

1. NetCOBOLの製品媒体のGETSSCH格納ディレクトリ配下に“表3-8 GETSSCH関連の提供ファイル一覧”で示すファイルが存在します。使用するOSIV系システムで必要とされるファイルをOSIV系システムに転送してください。

表3-8 GETSSCH関連の提供ファイル一覧

ファイル名	説明	転送の有無		転送先データセット名(UIDはユーザ識別名)
		MSP	XSP	
GETSSCH.OBJ	OSIV系プログラムのオブジェクト	○*1	○*1	‘UID.GETSSCH.OBJ’
GTSLINK.CLM	MSPでのリンク用CLIST	○*2	×	‘UID.GTSLINK.CLIST’
GTSLINK.CLX	XSPでのリンク用CLIST	×	○*2	‘UID.GTSLINK.CLIST’
GTSLINK.DAT	MSPのリンケージエディタ制御文	○*2	×	‘UID.GTSLINK.DATA’
GETSSCH.CL	GETSSCHの実行用CLIST	△	△	‘UID.GETSSCH.CLIST’

○: *1はバイナリモード、*2はテキストモード(EBCDIC-ASCII変換)で転送。
×:転送不要、△:任意

転送にはDUET転送(OSIV系システムのFEXPORT)などを使用して、転送先のデータセットは固定長80バイトの順編成ファイルとします。

2. 転送したリンク用のCLISTを実行して、実行可能なGETSSCHを作成します。なお、このリンク用CLISTは、リンケージエディタの格納データセットが以下に示すものとして記述され

ています。必要に応じて、修正してください。
- MSP:’SYS1.LINKLIB’
- XSP:’SYS.XSP.LINKLIB’

3. 作成されたGETSSCHのロードモジュールは以下に示すものになります。
   - ‘UID.GETSSCH.LOAD(JMNGTS)’

GETSSCHの使用するデータセット

以下にサブスキーマ取り出しツール(GETSSCH)を使用する場合の入出力データとそのファイル属性を示します。

図3-28 サブスキーマ取り出しツール(GETSSCH)における入出力の流れ

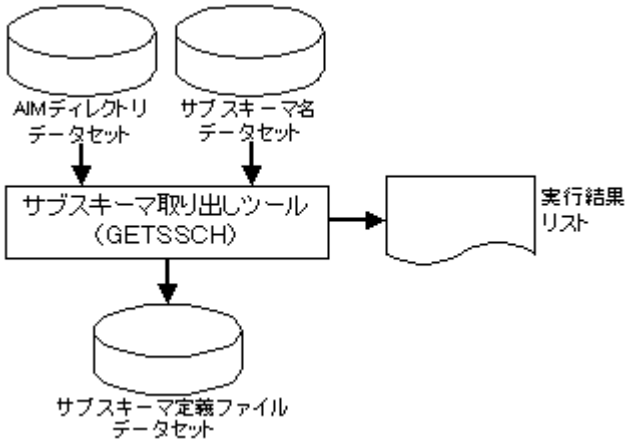


表3-9 サブスキーマ取り出しツール(GETSSCH)におけるファイルの属性

ファイル種別	指定方法（ＤＤ名またはアクセス名）	ファイル属性			
		編成	レコード形式	レコード長（バイト）	ブロック長（バイト）
ＡＩＭディレクトリデータセット	ＡＩＭＬＩＢ				
サブスキーマ名データセット	ＳＹＳＩＮ	順編成	ＦＢ	80	任意
サブスキーマ定義ファイルデータセット	ＳＳＣＨＬＩＢ	区分編成	ＦＢ	80	任意
リスト出力先(実行結果リスト)	ＳＹＳＰＲＩＮＴ	順編成	ＶＢＡ	255	任意

サブスキーマ名データセットには、ＡＩＭディレクトリから取り出したいサブスキーマの名前を指定します。

```
----+----1----+----2----+----3----+----4----+----5----+---～-8
COB85SB
CB85NRSB
```

サブスキーマ名を複数指定すれば、一度の起動で複数のサブスキーマを取り出すことができます。

GETSSCHの起動パラメタ

GETSSCHの起動パラメタは、MSPではEXEC文のPARMパラメタで指定し、XSPではPARA文で指定します。起動パラメタには以下のものがあります。

表3-10 GETSSCHの起動パラメタ

指定形式	説明
ＵＷＡ({A｜Ｎ})	ＡＩＭディレクトリより取り出すサブスキーマのＵＷＡの言語種別を指定します。 ＵＷＡ(Ａ):英数字(省略値) ＵＷＡ(Ｎ):日本語

GETSSCHの起動ジョブのJCL

GETSSCHを起動するジョブのJCLの例を示します。

図3-29 MSPでのGETSSCH起動ジョブのJCL例

```
//USER1SCH JOB …
//GETSSCH  EXEC PGM=JMNGTS,REGION=256K,PARM='UWA(A)'
//STEPLIB  DD DSN=USER1.GETSSCH.LOAD,DISP=SHR
//SYSIN    DD DSN=USER1.SSNAME.DATA,DISP=SHR
//AIMLIB   DD DSN=AIMPP2.V12L30.DRCTLIB,DISP=SHR
//SSCHLIB  DD DSN=USER1.SSCH.LIB,DISP=(NEW,CATLG),
//            UNIT=SYSDA,SPACE=(TRK,(10,10,10)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD SYSOUT=*
/*
```

図3-30 XSPでのGETSSCH起動ジョブのJCL例

```
¥ JOB  EXAMPLE
¥ EX   JMNGTS,RSIZE=256
¥ PARA UWA(A)
¥ FD   PRGLIB=DA,FILE=USER1.GETSSCH.LOAD
¥ FD   SYSIN=DA,FILE=USER1.SSNAME.DATA
¥ FD   AIMLIB=DA,FILE=AIMPP2.V12L30.DRCTLIB
¥ FD   SSCHLIB=DA,FILE=USER1.SSCH.LIB,DISP=CAT,
       TRK=(10,10),DRTY=(10,BLK),VOL=WORK,
       FCB=(LRECL=80,RECFM=FB,BLKSIZE=3120)
¥ FD   SYSPRINT=DA,VOL=WORK,TRK=(5,1),SOUT=A
¥ JEND
```

GETSSCHの復帰コード

GETSSCHの復帰コードを示します。

表3-11 GETSSCHの起動パラメタ

状態	復帰値		説明
	ＭＳＰ	ＸＳＰ	
正常	０	１０	サブスキーマ名の取り出しが正常に終了し、サブスキーマ定義ファイルに登録集原文(COPY句)を出力した。
エラー	８	３０	

重度のエラー	１２	４０	エラーによりサブスキーマ名の取り出しが失敗した。
致命的エラー	１６	５０	

CLISTによるGETSSCHの実行

ここまで、JCLによるGETSSCHの実行方法を説明してきましたが、NetCOBOLで提供しているCLISTを使用して、GETSSCHを実行することもできます。
このCLISTは以下のパラメタを持ちます。

1. SYSIN()
   サブスキーマ名データセットを指定します。
   SSCHNAME()と同時に指定することはできません。
2. SSCHNAME ()
   サブスキーマ名を指定します。このパラメタはサブスキーマ名データセットを用意してないときに使用するもので、直接サブスキーマ名を指定(複数は不可)できます。
   SYSIN()と同時に指定することはできません。
3. AIMLIB ()
   ＡＩＭディレクトリデータセット名を指定します。必ず指定する必要があります。
4. SSCHLIB ()
   サブスキーマ定義ファイルのデータセット名を指定します。必ず指定する必要があります。
5. SYSPRINT ()
   診断メッセージの出力先を指定します。データセットに出力したい場合にはデータセット名を、画面に出力したい場合には*を、プリンタへ直接出力したい場合はプリンタクラスを指定します。
   省略可能で、省略時は画面に出力します。
6. OPT ()
   GETSSCHの起動パラメタを指定します。省略可能で、省略時は省略値を使用します。

以下にその実行例を示します。

図3-31 SYSIN()を指定してのCLISTでの実行例

```
READY
EX GETSSCH 'SYSIN(USER1.SSNAME.DATA),AIMLIB(AIMPP2.V12L30.DRCTLIB),
SCHLIB(USER1.SSCH.LIB)'
```

図3-32 SSCHNAME ()を指定してのCLISTでの実行例

```
READY
EX GETSSCH 'SSCHNAME(COB85SB),AIMLIB(AIMPP2.V12L30.DRCTLIB),
  SSCHLIB(USER1.SSCH.LIB),SYSPRINT(USER1.SSCH.LIST)'
```

図3-33 プリンタクラスを指定してのCLISTでの実行例

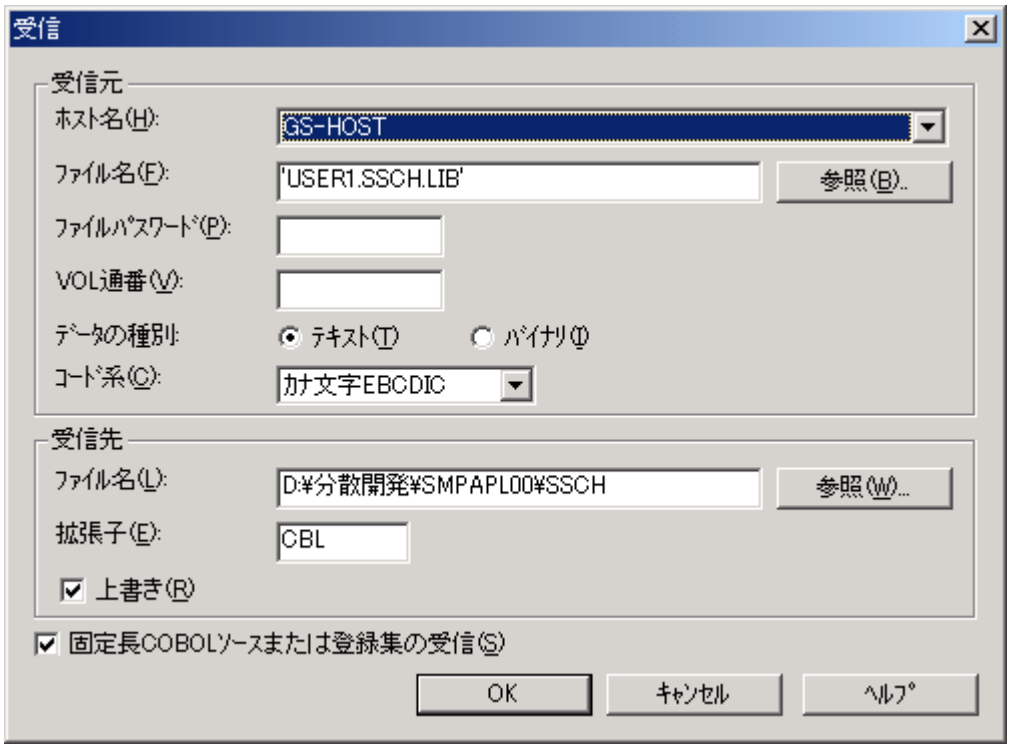
```
READY
EX GETSSCH 'SSCHNAME(COB85SB),AIMLIB(AIMPP2.V12L30.DRCTLIB),
  SSCHLIB(USER1.SSCH.LIB),SYSPRINT(A)'
```

3.3.4.2 Windows系システムでの処理

COBOLプロジェクトマネージャの分散開発支援機能の“受信”機能を使用して、サブスキーマ取り出しツール(GETSSCH)で作成したサブスキーマ定義ファイルをOSIV系システムのデータセットからWindows系システムのファイルに受信します。
基本的な手順は、COBOLソース・登録集原文を移行する場合と同じです。ここでは、異なる設定が必要となる項目のみ説明します。

1. 〔受信先〕は次のように設定してください。
   — 拡張子:
     サブスキーマ定義ファイルは、NetCOBOLでは登録集原文に相当するものとして扱います。ここは、CBLと指定してください。
2. サブスキーマ定義ファイルは固定形式です。“固定形式COBOLソースまたは登録集の受信”をチェックしてください。

図3-34 サブスキーマ定義ファイルの移行時の〔受信〕ダイアログの設定例



3. 〔OK〕ボタンをクリックすると受信処理を開始します。

3.4 プログラミング作業

3.4.1 ソース・登録集原文の作成、修正

ここでは、ソース・登録集原文の作成、編集方法について説明します。なお、対象となるソース・登録集原文は、既に次の作業を実施済みのものとして説明を続けます。

- プロジェクトファイルを作成し、ソース・登録集ファイルをプロジェクトに登録している。
- UP開発の場合は、OSIV系システムからソース・登録集原文がWindows系システムに転送し、プロジェクトに登録したパスに格納されている。

COBOLプロジェクトマネージャは次の2つの操作ビューを持ちますが、ソース・登録集原文の作成、編集方法はこのどちらの操作ビューからも可能です。

図3-35 プロジェクトマネージャの〔プロジェクト構成〕 ページ

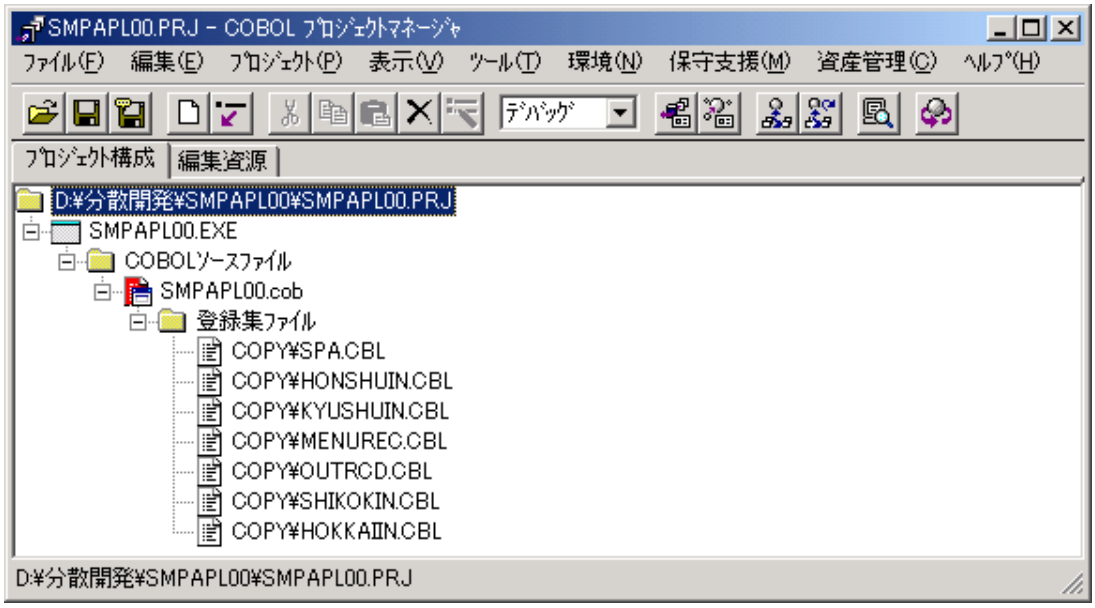


図3-36 プロジェクトマネージャの〔編集資源〕 ページ

作成、編集対象のソース・登録集原文を選択して、〔プロジェクト〕メニューから“編集”を選ぶか、選択したファイル名をダブルクリックすることで、エディタが起動して、選択したファイ

ルの編集ができるようになります。
通常は、エディタとしてNetCOBOLの製品に組み込みのCOBOLエディタが用いられます。以下、このエディタの固有の操作とプロジェクトマネージャで使用するエディタをカスタマイズする方法について説明します。

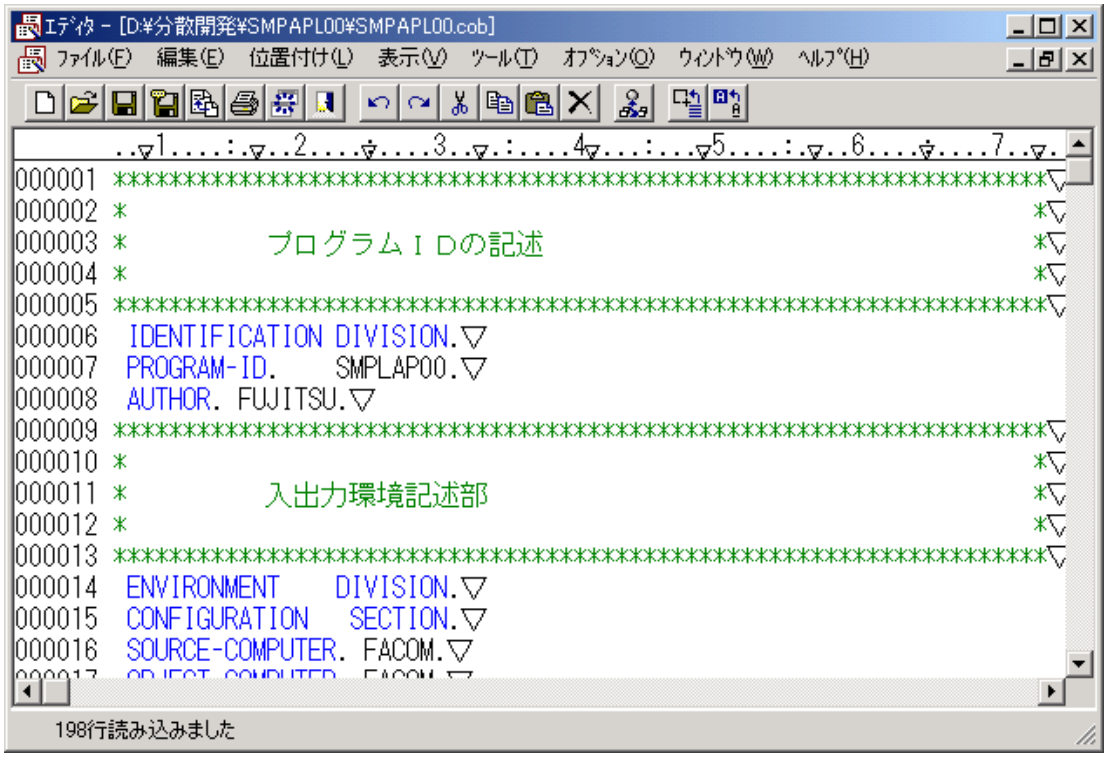
3.4.1.1 COBOLエディタ

NetCOBOLに組み込みのエディタは、通常のテキストエディタとしての機能の他にCOBOLソースの効率的な編集のため、次のような機能を持っています。

- 一連番号領域の操作の自動化・制限
- カラー構文表示
- テンプレート展開機能
- 簡易翻訳モード
- メッセージ連携機能

ここではこれらの機能についてのみ説明します。一般的なエディタの機能については、エディタのヘルプを参照してください。

図3-37 NetCOBOL組み込みのエディタの外観(行番号付きテキストとして開いた例)

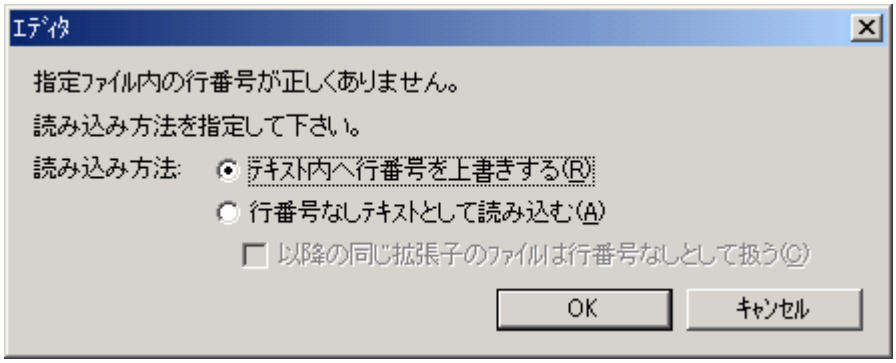


なお、このエディタの編集対象ファイルの管理方式は次の2つの形式があります。

- 行番号付きテキスト
- 行番号なしテキスト

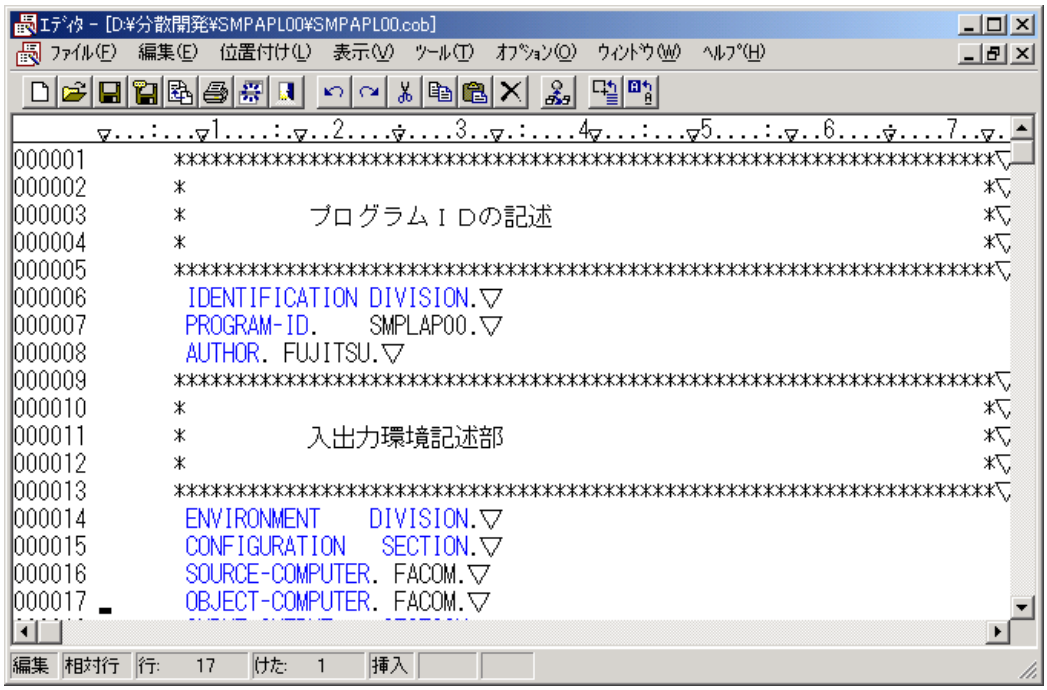
COBOLソースの正書法の形式で一連番号領域(1～6カラム)が正しく昇順の行番号となっている場合は、自動的に行番号付きテキストとして読み込みます。一連番号領域に空白や行番号として認識できない文字、あるいは昇順でない行番号が含まれる場合、ファイルのオープン時に次のダイアログボックスが表示されます。

図3-38 エディタの入力ファイル確認のダイアログ



そのまま〔OK〕ボタンをクリックした場合、一連番号領域に新しい昇順に生成した行番号を上書きした上で、行番号付きテキストとしてファイルを開きます。
“行番号なしテキストとして読み込む”を選択してから、〔OK〕ボタンをクリックした場合、次のように行番号なしテキストとしてファイルを開きます。
行番号なしテキストとしてファイルを開いた場合、編集操作の一部の機能でふるまいが変わってくるので注意してください。

図3-39 NetCOBOL組み込みのエディタの外観(行番号なしテキストとして開いた例)



一連番号領域の操作の自動化・制限

ファイルを行番号つきテキストとして読み込んだ場合、一連番号領域(1～6カラム目)をエディタが認識して、次のような操作の自動化と制限が行われます。

- 行の追加・挿入時、行番号を自動生成／自動リナンバ
- 編集やソース行の“右シフト”、“左シフト”の対象になりません。

カラー構文表示

編集中のソース・登録集の構文を認識して以下のカテゴリで色分けして表示します。

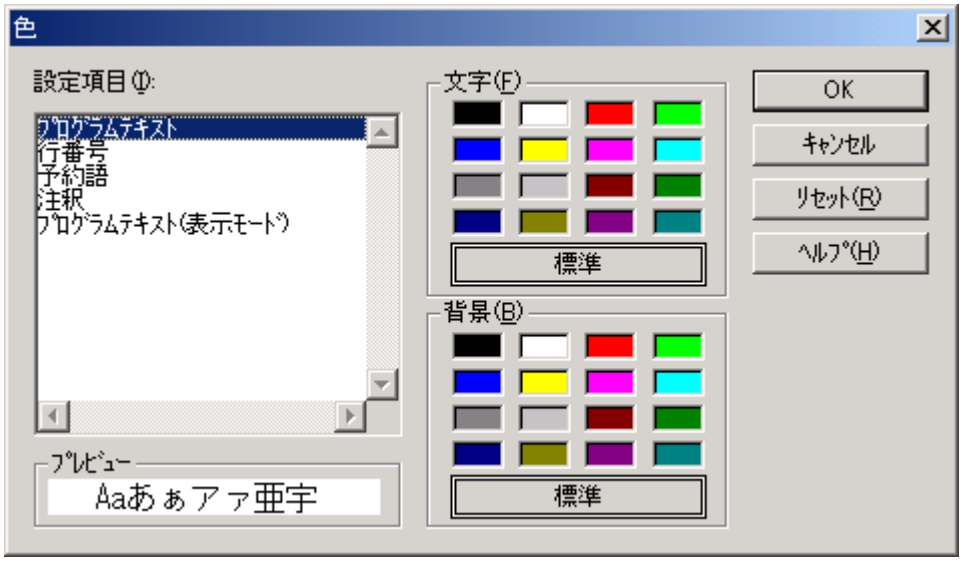
- 行番号
- 注釈／行内注記
- 予約語
- プログラムテキスト(利用者語／定数)

注意

NetCOBOLコンパイラは、翻訳オプションRSVの指定で使用する予約語セットを選択して、どの語を予約語と見なすか切り換えることができます。しかし、エディタのカラー構文表示で使用される予約語セットの選択を行うことはできません。

エディタの〔表示〕メニューから“色”を選択すると、次のダイアログを表示します。このダイアログから、個々の表示色のカスタマイズが可能です。

図3-40 カラー構文表示のカスタマイズ用のダイアログ



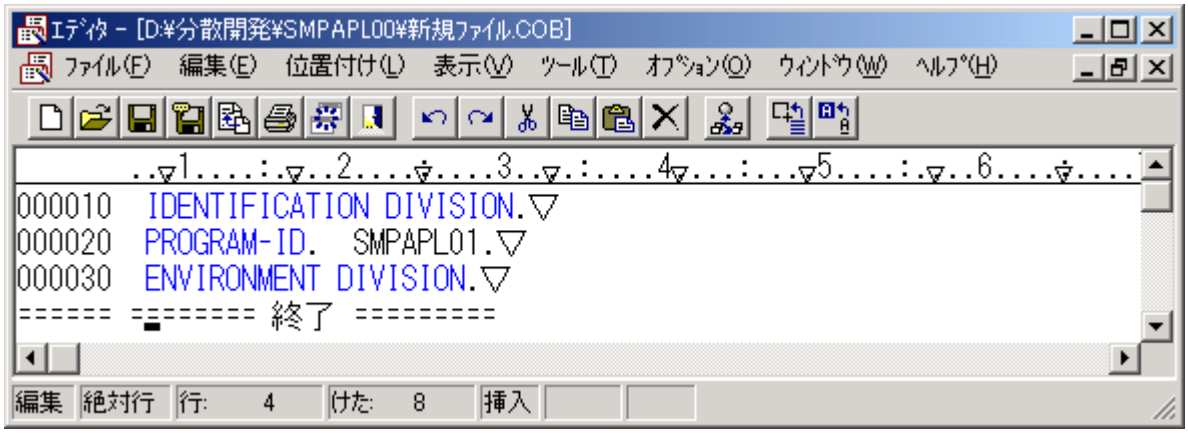
テンプレート展開機能

COBOLの基本的な構文についてのテンプレートが用意されており、編集中のソース・登録集原文の任意の位置に展開することができます。

以下、テンプレート展開機能の使用法を説明します。

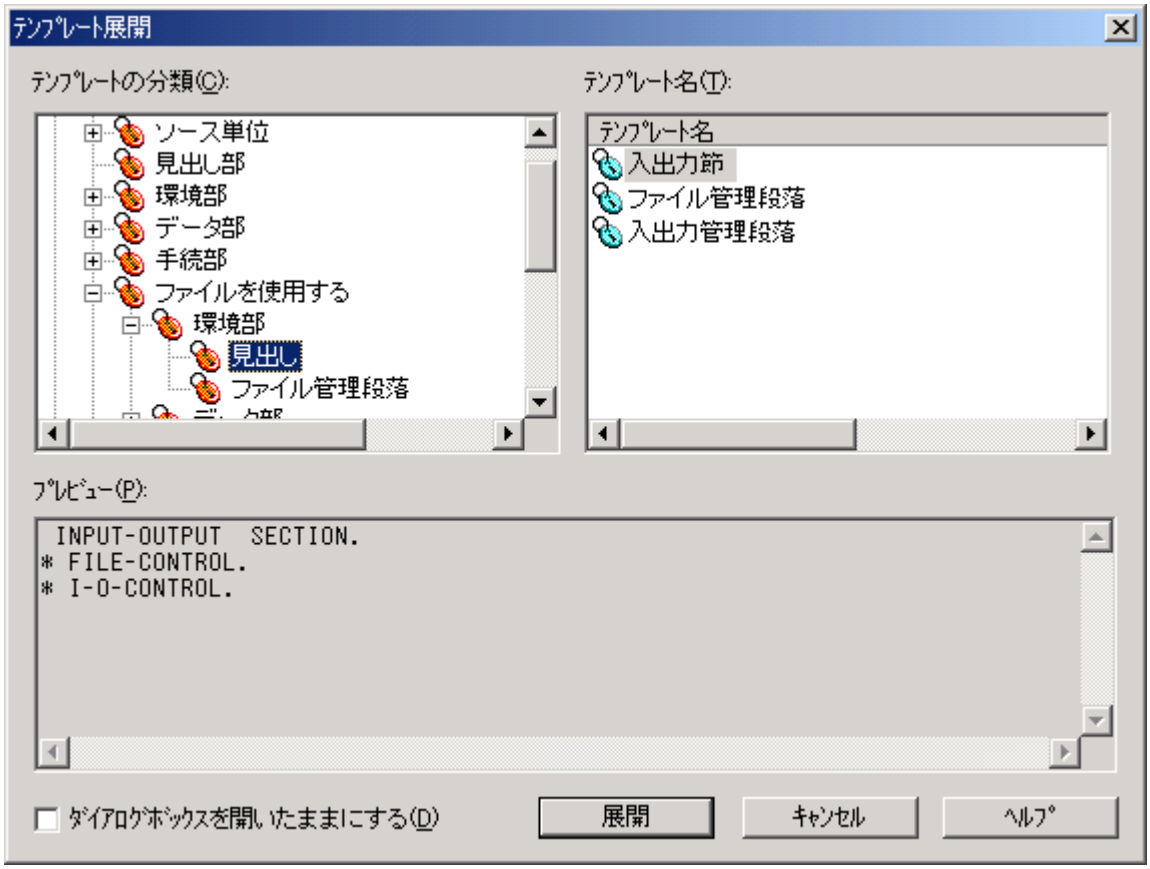
1. テンプレートを展開しようとする行にカーソルを合わせます。

図3-41 テンプレート展開前の状態



2. エディタの〔表示〕メニューから “テンプレート展開”を選択します。
3. 〔テンプレート展開〕ダイアログから、“テンプレートの分類”、“テンプレート名”および“プレビュー”などを参考に展開するテンプレートを選択します。

図3-42 〔テンプレート展開〕ダイアログ



4. 展開するテンプレートが決まったら、そのテンプレート名を選択して、〔選択〕ボタンをクリックします。
5. エディタのカーソル行から、テンプレートが挿入されます。この例では、入出力節の見出しを展開しています。

図3-43 テンプレート展開後の状態

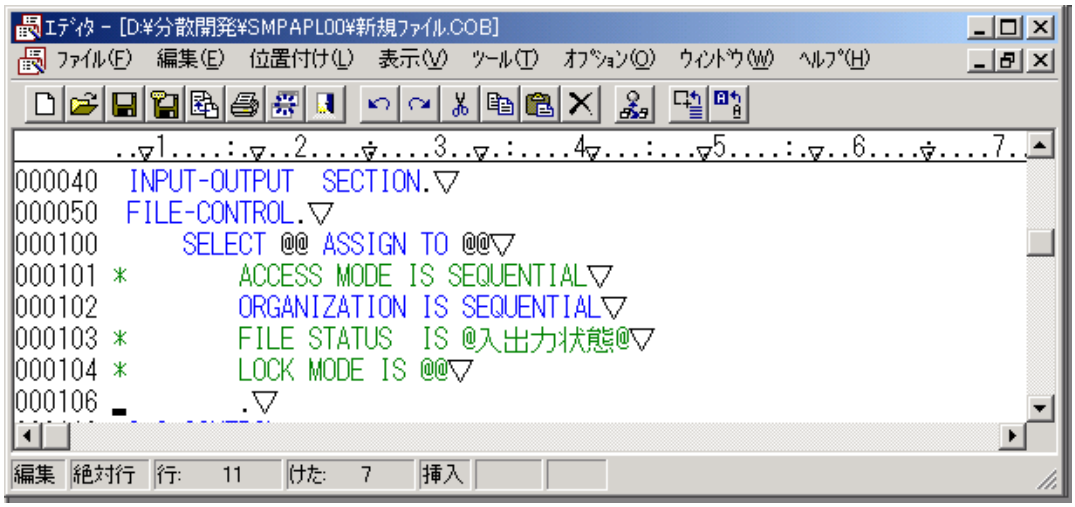
テンプレートはいくつかのカテゴリがありますが、最も基本的なカテゴリである“COBOLの基本機能”では次のようなテンプレートを用意しています。

- ソース単位

- 見出し部
- 環境部
- 手続き部
- ファイルを使用する
- 印刷機能を使用する
- 整列併合用機能を使用する入出力機能を使用する

なお、テンプレート展開で挿入するソース中で、データ名や定数などはその位置を示すだけの記号で表現します。適切な形に修正してください。

図3-44 テンプレート展開後の状態(ファイル記述項の展開直後)



簡易翻訳処理

編集中のソースをエディタの上から一時的に翻訳してみることができます。

注意

プロジェクトの翻訳オプションの設定を引き継ぎません。別途設定が必要なので注意してください。

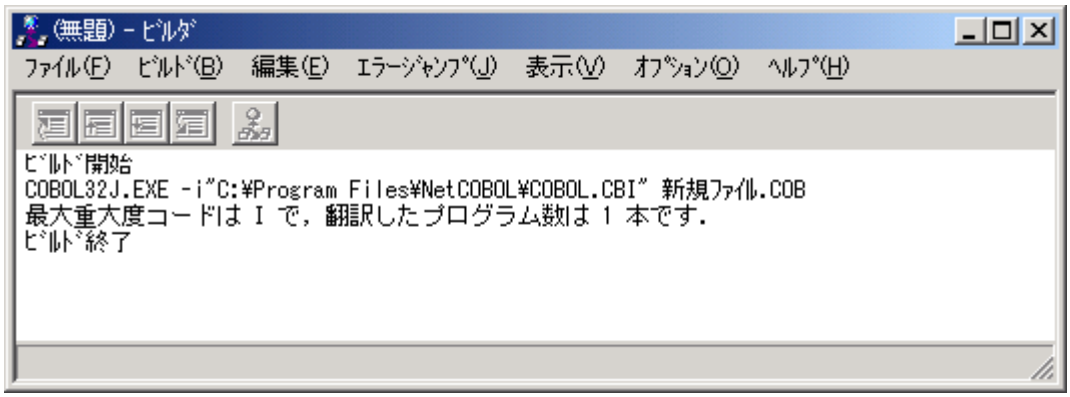
以下、その手順を説明します。

1. エディタの〔ツール〕メニューから“翻訳”を選択します。

図3-45 簡易翻訳処理の実行

2. 〔ビルダ〕ウィンドウが現れ、翻訳処理を実行します。
3. 翻訳が終了すると翻訳結果を〔ビルダ〕ウィンドウ内に表示します。

図3-46 簡易翻訳処理の実行結果



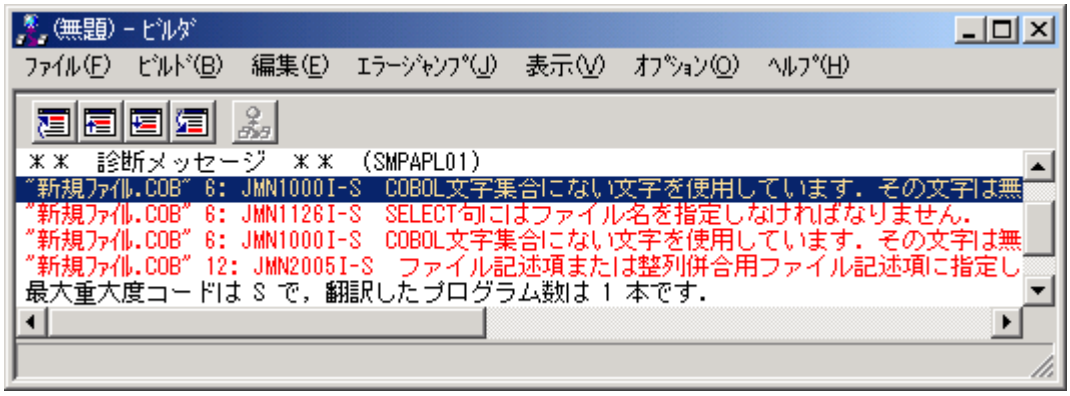
メッセージ連携機能

エディタの簡易翻訳機能あるいはプロジェクトマネージャのビルド機能によってプログラムを翻訳した結果、翻訳エラーが検出された場合、〔ビルダ〕ウィンドウに表示した診断メッセージからエディタ上でエラーの発生箇所に移動します。

以下、手順を説明します。

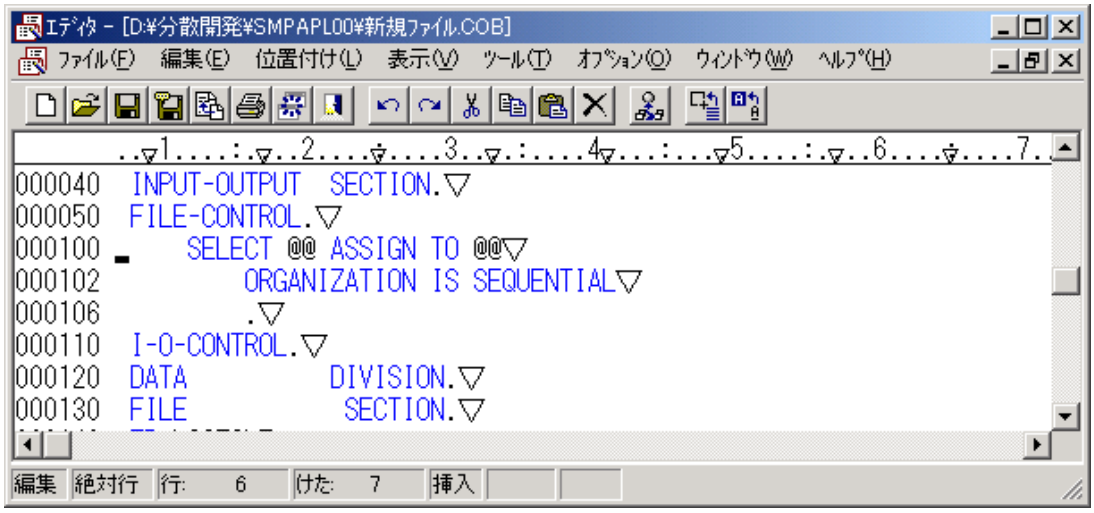
1. 〔ビルダ〕ウィンドウ上で、修正しようとしている翻訳エラーに対する診断メッセージを選択します。

図3-47 修正対象の診断メッセージの選択



2. 選択した診断メッセージをダブルクリックすると、エディタにフォーカスが移動し、エラーの発生した行の先頭にカーソルが移動します。翻訳エラーの発生したソースがエディタで開かれていない場合、自動的にエディタが起動します。

図3-48 メッセージ連携機能の実行結果



3.4.1.2 エディタのカスタマイズ

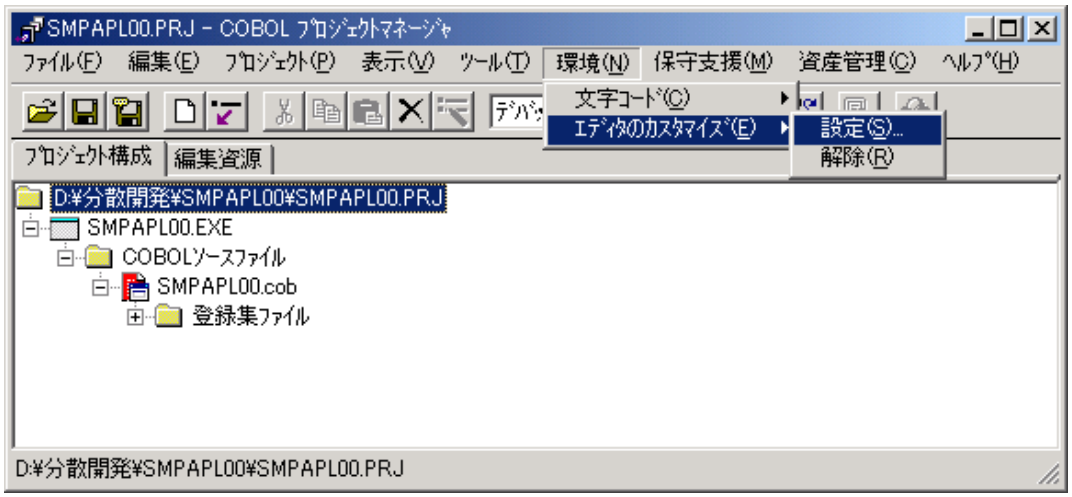
プログラムマネージャから起動するエディタは、初期状態ではNetCOBOLに組み込まれているエディタです。普段使い慣れているエディタが他に存在する場合、プログラムマネージャから起動するエディタをそのエディタで置き換えることもできます。これを“エディタのカスタマイズ”と呼びます。

以下、“エディタのカスタマイズ”の設定方法と解除方法を説明します。

エディタのカスタマイズの設定

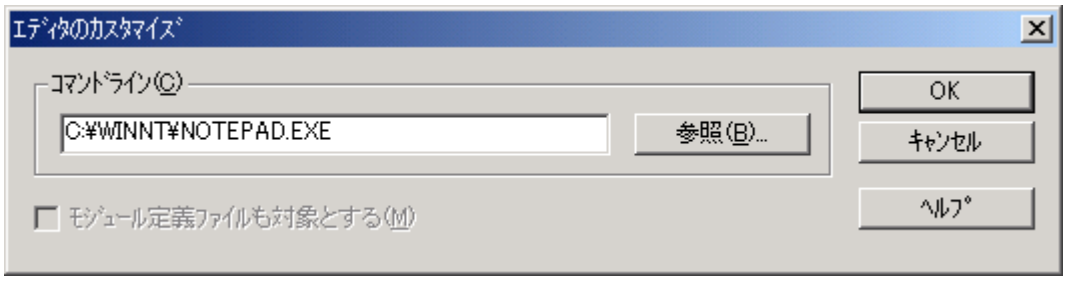
1. プロジェクトマネージャの〔環境〕－〔エディタのカスタマイズ〕メニューから、“設定”を選択します。

図3-49 エディタのカスタマイズの設定



2. 〔エディタのカスタマイズ〕ダイアログが表示されるので、エディタを起動するためのコマンドラインを直接テキストボックスに入力するか、〔参照〕ボタンをクリックして、〔ファイルを開く〕ダイアログでエディタの実行形式ファイルを選択ください。

図3-50 エディタのカスタマイズ例(NOTEPADを選択)



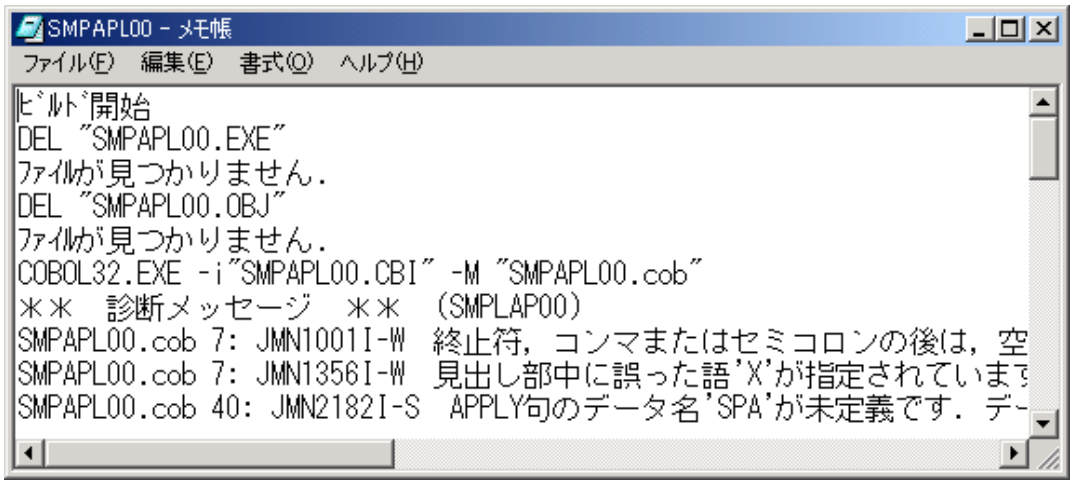
3. 〔OK〕ボタンをクリックすれば、これでエディタのカスタマイズの設定は完了です。

エディタのカスタマイズ後の動作

エディタをカスタマイズした場合、COBOLプロジェクトマネージャからの操作が次のように代わります。

- COBOLソース・登録集ファイルの編集
  COBOLソース・登録集ファイルの編集の為にファイルを開くエディタが“カスタマイズ”で指定したエディタに変更されます。編集に使用できる機能は“カスタマイズ”で指定したエディタの機能に依存します。
- 翻訳結果の表示
  COBOLプロジェクトマネージャでCOBOLソースの翻訳処理を行った場合、その翻訳結果が一時的なファイルに出力され、カスタマイズしたエディタで開かれます。“カスタマイズ”によりCOBOLエディタのメッセージ連携機能は使用できなくなりますが、カスタマイズ後のエディタが“タグジャンプ”機能などを持つ場合、同等のことが可能です。

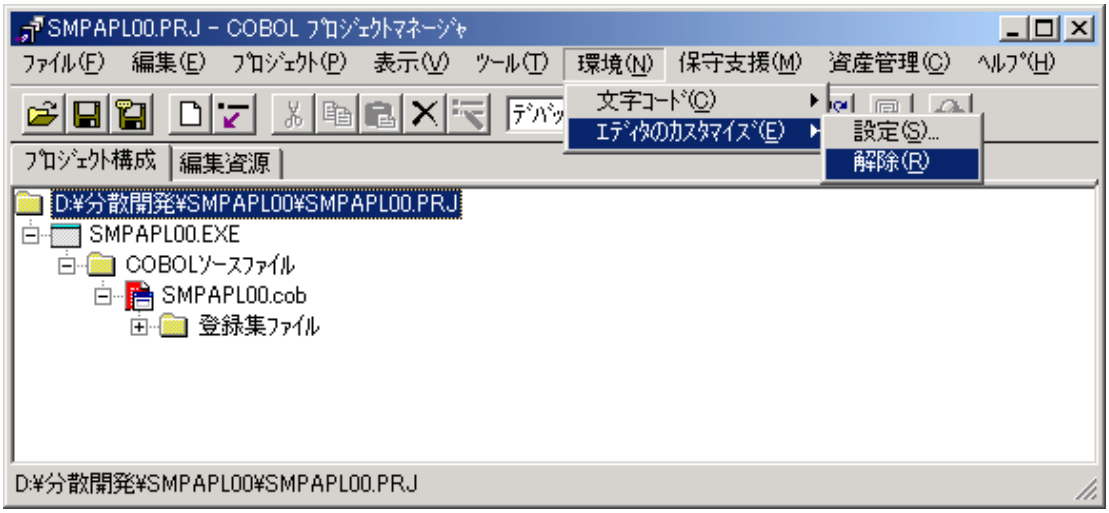
図3-51 カスタマイズしたエディタで翻訳結果を開いた場合の例



エディタのカスタマイズの設定解除

1. 設定したエディタのカスタマイズを解除するには、プロジェクトマネージャの〔環境〕－〔エディタのカスタマイズ〕メニューから、“解除”を選択します。

図3-52 エディタのカスタマイズの設定の解除



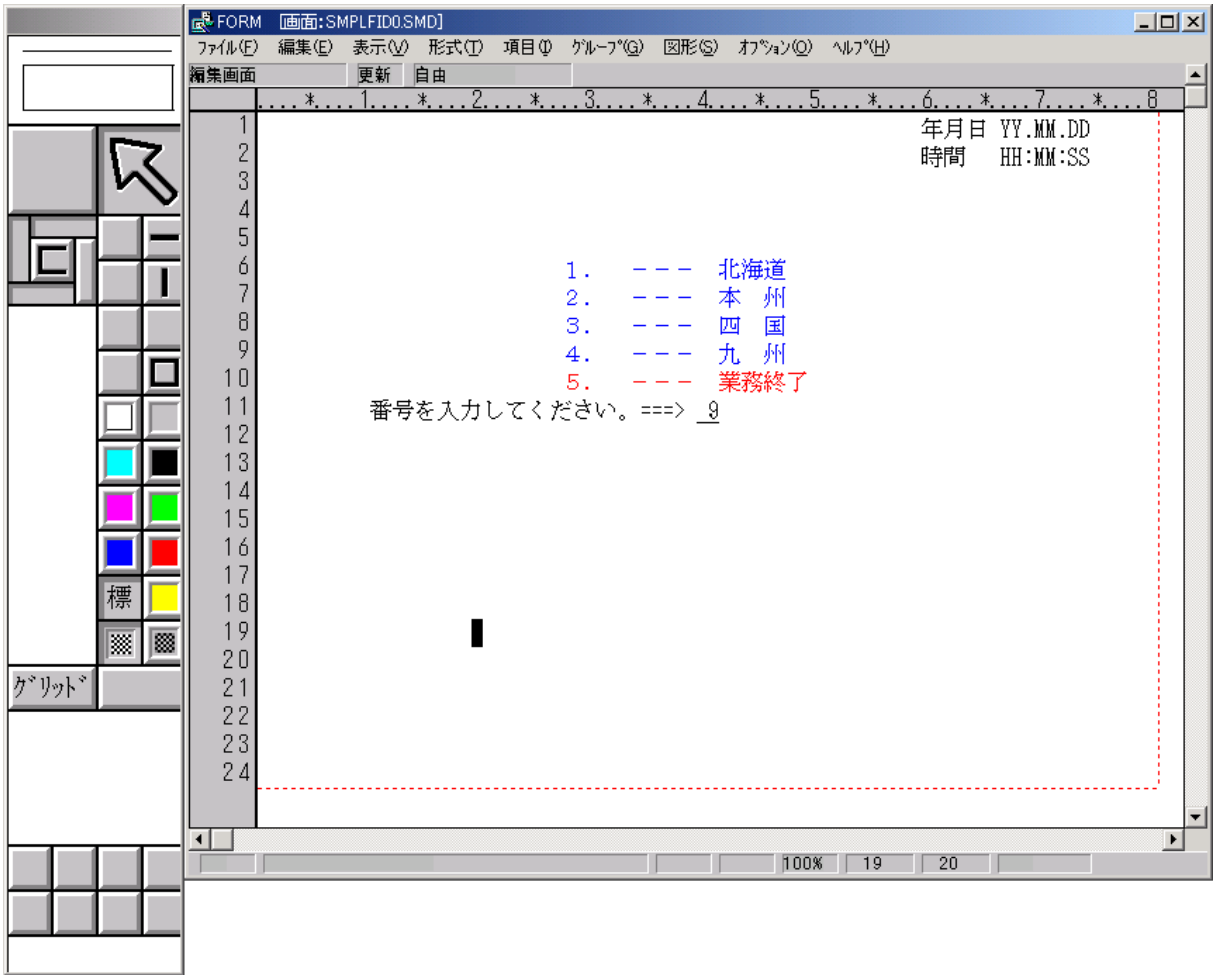
3.4.2 各種定義体の作成、修正

OSIV系プログラムで使用される以下の資産は、Windows系システム上の開発ツールを使用して作成・修正することができます。

- フォーマット定義体
- フォームオーバレイパターン

もちろん、OSIV系プログラムで使用可能なものの間には形式や機能差がありますから、Windows系システム上で作成・修正したもののすべてがOSIV系システムでそのまま使用できると言うわけではありませんが、しかし、Windows系システムでの開発ツールは、画面イメージを直接表示しながら、それを直接編集してゆく方法をとるため、より短時間で望む定義体を作成することが可能となります。

図3-53 FORMによる定義体の編集例



ここでは、OSIV系プログラムでも使用できる定義体をWindows系システム上で作成するための設定のみを説明します。開発ツールの使用法についての詳細は、それぞれのヘルプを参照してください。

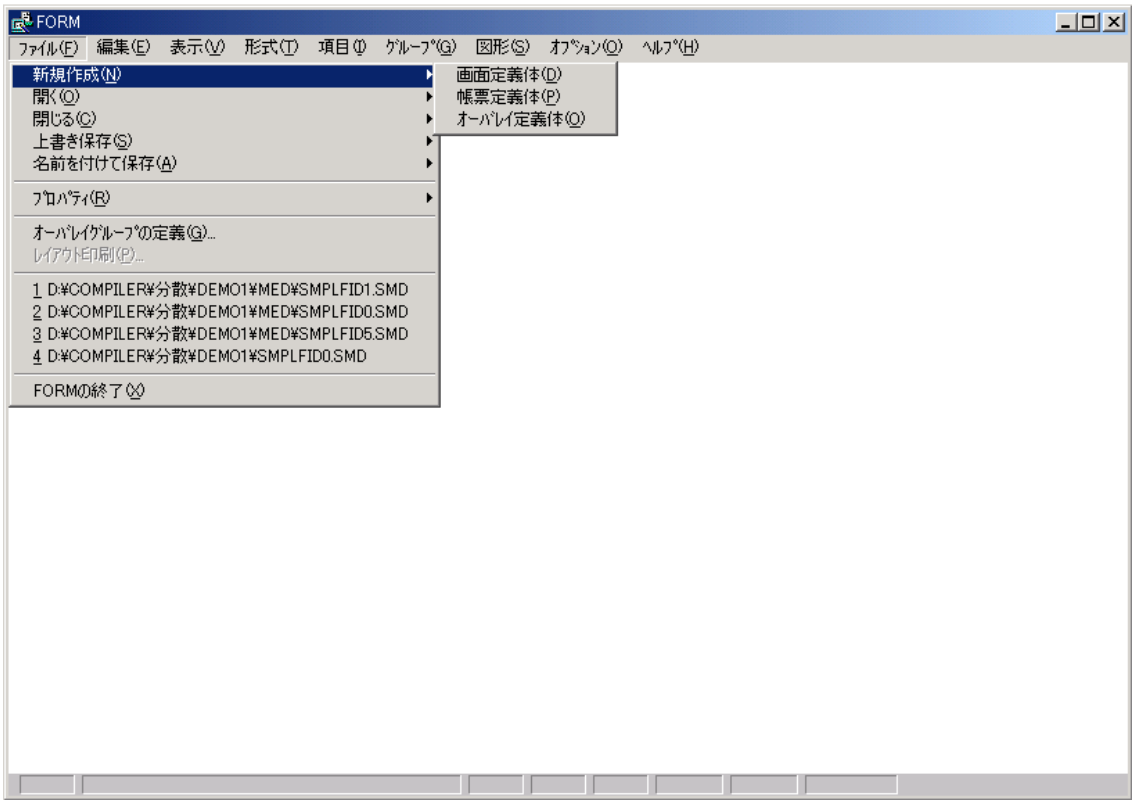
3.4.2.1 フォーマット定義体の作成・編集

OSIV系システムで使用するフォーマット定義体をWindows系システムで作成する場合、FORMを使用します。FORMで作成できる定義体(画面／帳票定義体)とフォーマット定義体とは形式が異なるため、OSIV系システム上で形式を変換する必要があります。

FORMにより画面帳票定義体の作成を開始する手順を説明します。

1. FORMを起動します。
2. 〔ファイル〕－〔新規作成〕メニューから、作成する定義体の種別を選択します。
   - 宛先“DSP”の表示ファイル用のフォーマット定義体が必要な場合、“画面定義体”を選択します。
   - 宛先“PRT”の表示ファイル用のフォーマット定義体が必要な場合、“帳票定義体”を選択します。

図3-54 FORMの新規作成画面



OSIV系システムで作成し、移出機能を用いてOSIV系システムから移行した定義体を編集する場合は、定義体そのものが、画面用か帳票用かの情報を持っているため、特にその種別を意識する必要はありません。

3.4.2.2 フォームオーバレイパターンの作成・編集

Windows系システムでは、FORMまたはPowerFORMを使用して次のフォームオーバレイパターンを作成することができます。

- NLP用フォームオーバレイパターン(KOL2形式)
- JEF/AP用フォームオーバレイパターン(KOL5形式)

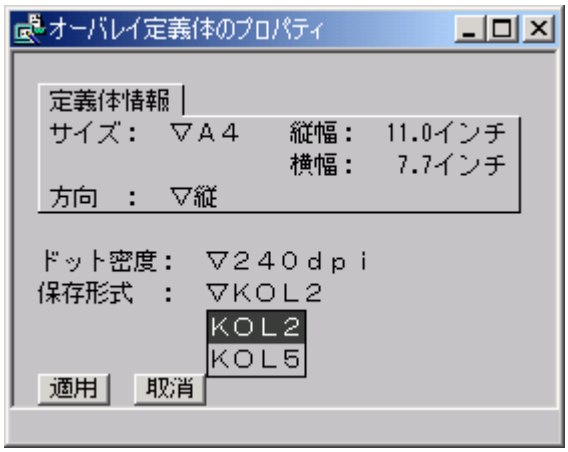
ただし、FORM/PowerFORMで作成可能なNLP用フォームオーバレイパターン(KOL2形式)は、OSIV系システムで一般に使用するNLP用フォームオーバレイパターン(KOL1形式)と形式が異なります。これについては、AP/DFのフォームオーバレイパターン移入機能を使用して、形式を変換することができます。

フォームオーバレイパターンには、複数の形式があるため、FORMまたはPowerFORMを使用して、フォームオーバレイパターンを作成・編集する際は、明示的にその形式を指定する必要があります。以下、その指定方法について説明します。

FORMでオーバレイ定義体を作成する場合

1. FORMを起動します。
2. 〔ファイル〕－〔新規作成〕メニューから、作成する定義体として“オーバレイ定義体”を選択します。
3. 〔ファイル〕－〔プロパティ〕メニューから“オーバレイ定義体”を選択します。
4. “オーバレイ定義体のプロパティ”ダイアログが表示されるので、ここで保存形式を選択します。デフォルトでは“KOL2”がFORMで作成するオーバレイ定義体の保存形式です。

図3-55 “オーバレイ定義体のプロパティ”ダイアログでの保存形式の選択



FORMでオーバレイ定義体を編集する場合

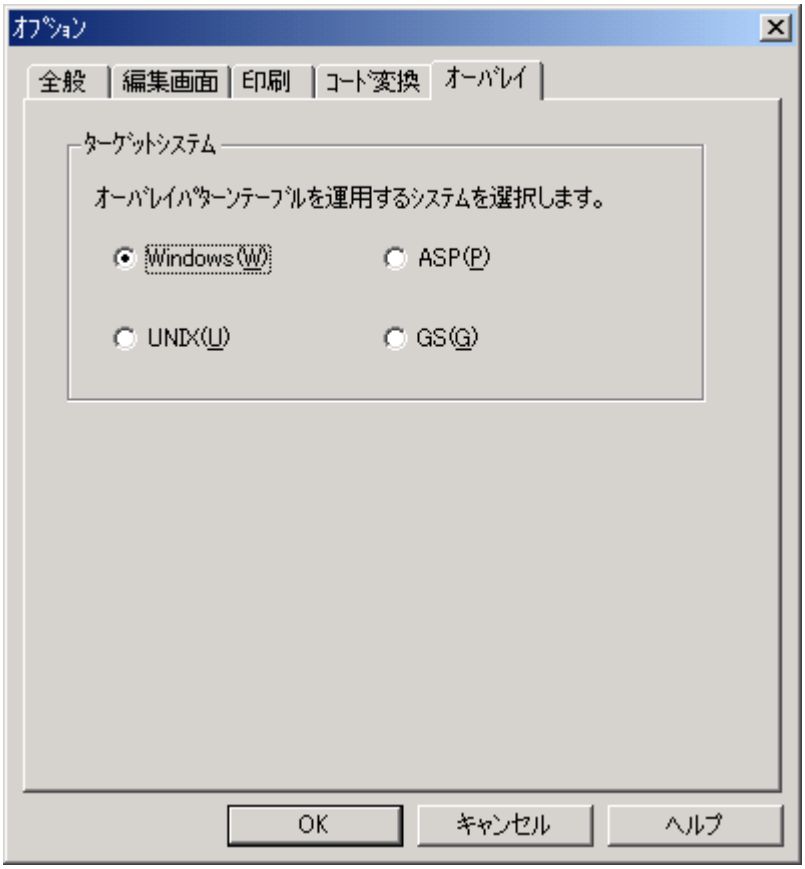
KOL2またはKOL5形式のオーバレイ定義体であれば、編集可能です。それ以外の形式のオーバレイ定義体は編集することはできません。

PowerFORMでオーバレイ定義体を作成・編集する場合

PowerFORMで、デフォルトで作成・編集可能なオーバレイ定義体はKOL6形式のみです。OSIV系プログラムで使用可能なオーバレイ定義体を作成するためにはターゲットシステムの設定を変更する必要があります。以下、その手順を説明します。

1. PowerFORMを起動します。この状態で、開かれている定義体があるなら、一度それを閉じます。
2. 〔ツール〕メニューから、“オプション”を選択します。
3. “オプション”ダイアログが表示されるので、〔オーバレイ〕ページでターゲットシステムを選択します。

図3-56 〔オーバレイ〕ページでのターゲットシステムの選択



4. OSIV系プログラムで使用可能なフォーマット定義体を作成・編集する場合、“GS”または“ASP”を選択してください。

表3-12 PowerFORMにおけるターゲットシステムの指定とオーバレイ定義体の形式

ターゲットシステムの選択	読み込み可能な形式	保存形式
GS	KOL2形式	KOL2形式
ASP	ADJUSTで作成したKOL5形式 PowerFORMで作成したGS形式	GS形式

3.5 翻訳チェックとリンク

COBOLプログラムマネージャの機能を使用して、作成／修正したOSIV系プログラムの翻訳およびリンクを行います。
Windows系システムで、分散開発のどの過程まで完了するかに関わらず、ここで翻訳エラーやリンクエラーが出力されなくなるようにしておくことが重要です。

3.5.1 OSIV系プログラムの翻訳

NetCOBOLにおいて、Windows系のプログラムの開発時もOSIV系プログラムの分散開発時も、プログラムの翻訳は基本的に同じ操作で可能です。いくつか方法はありますが、ここではCOBOLプロジェクトマネージャからの操作のみ説明します。

1. プロジェクトマネージャで翻訳対象となるソースプログラムを選択します。〔プロジェクト構成〕ビュー、〔編集資源〕ビューのどちらからでも可能です。
2. 〔プロジェクト〕メニューから“翻訳”を選択します。〔ビルダ〕ウィンドウが現れて、翻訳処理を実行します。

図3-57 プロジェクトマネージャの〔プロジェクト構成〕ビューからの翻訳

3. 翻訳が終了すると翻訳結果を〔ビルダ〕ウィンドウ内に表示します。

3.5.1.1 翻訳チェックに有効な機能／製品

新規作成／修正したOSIV系プログラムの翻訳チェックでは、OSIV系のCOBOL85と同じ結果となることが理想です。しかしOSIV系COBOL85/NetCOBOLではサポートする言語の機能範囲が異なるため、しばしば次のような問題が発生します。

- NetCOBOLでは翻訳エラーが出力されなくなったプログラムをOSIV系システムのCOBOL85で翻訳すると翻訳エラーが残っている。
- NetCOBOLでは翻訳エラーが出力されているが、実はOSIV系システムのCOBOL85で翻訳エラーが出力されないプログラムが完成している。

このような問題に対応するため、次のような機能／製品が用意されています。

- 予約語セットの変更
- COBOL85非互換項目の指摘
- 使用コード系の変更

NetCOBOLを使用して、分散開発を効率的に行うためにはこれらの機能を組み合わせて使用する必要があります。

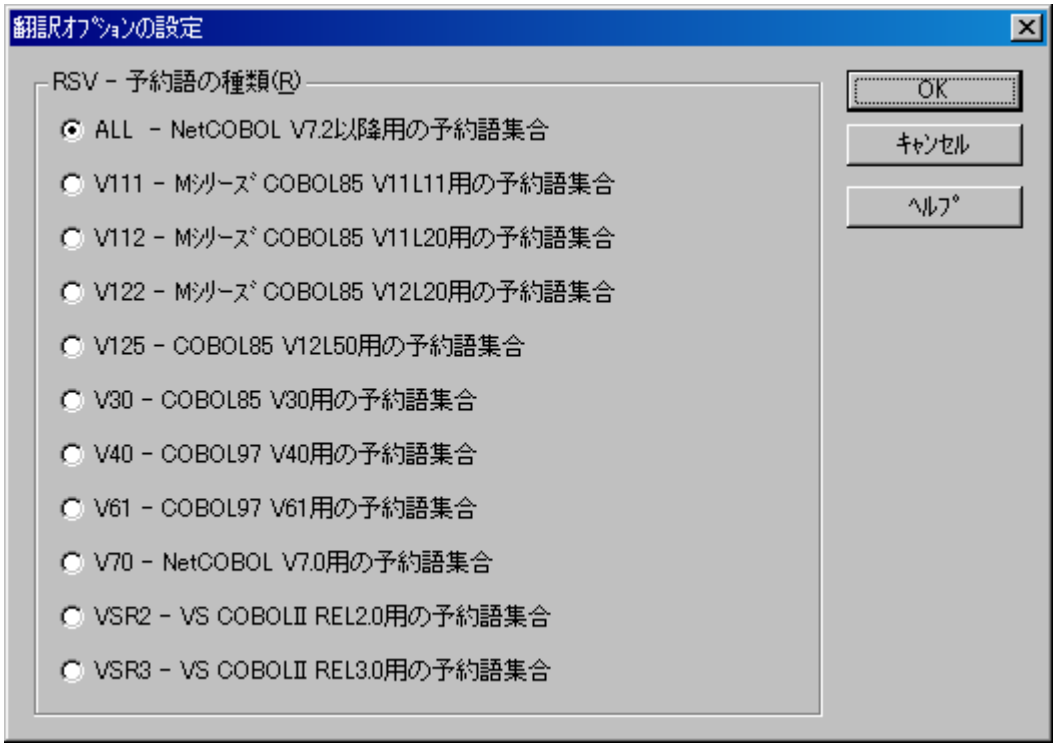
予約語セットの変更

翻訳オプションRSVの指定により、翻訳時に使用するCOBOLの予約語セットを切り換えます。
NetCOBOLが通常使用する予約語セットに対する部分集合となる予約語セットを使用することで、次のような効果を得る事ができます。

- 特定の予約語に依存する言語の機能を使用できなくします。
- 特定の予約語と同じ綴りを持つ名前をデータ名、ファイル名などの利用者語として使用可能とします。

OSIV系プログラムを分散開発する場合、通常は翻訳オプションRSV(V122)を指定します。

図3-58 翻訳オプションRSVの指定画面



なお、予約語セットの詳細な内容については“A.4 予約語”を参照してください。

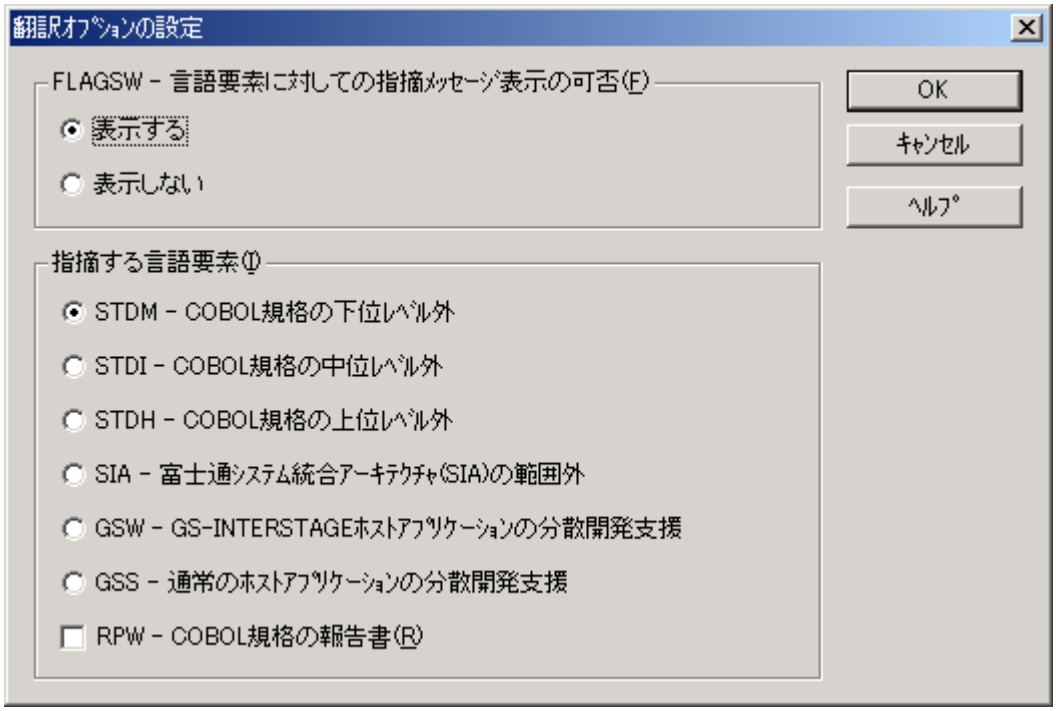
COBOL85非互換項目指摘機能

COBOL85/NetCOBOLの言語の機能差は予約語セットの切り換えただけでは、十分チェックすることができません。作成／修正したOSIV系プログラムが次のような記述を含む場合、翻訳オプションRSV(V122)を指定していても、正しく翻訳チェックを行うことができません。

- COBOL85の予約語セットの新しい組み合わせからなる新しい構文。
- 同じ記述に対する解釈がCOBOL85とNetCOBOLで解釈が異なる。

このような記述をただしくチェックするためには、翻訳オプションFLAGSWの指定で“COBOL85非互換項目”を指定します。

図3-59 翻訳オプションFLAGSWの指定画面



翻訳オプションFLAGSWでCOBOL85の非互換項目を指摘する指定は2種類あります。これは以下のように使い分けます。

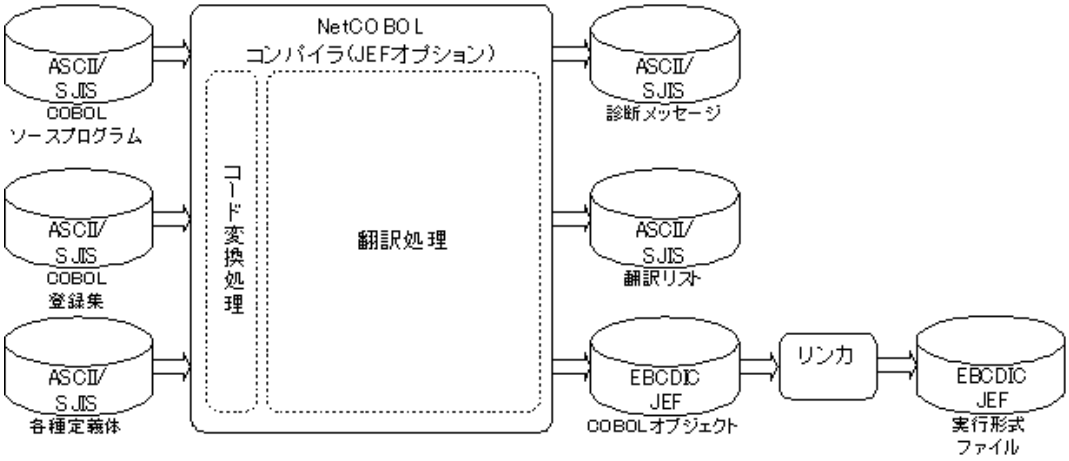
- FLAGSW(GSW):
  COBOL85の非互換項目の一部だけを指摘します。
  AADアプリケーション用の分散開発支援機能(配布ソース生成)と組み合わせて使用し、この機能で処理できない項目のみを指摘します。
- FLAGSW(GSS):
  COBOL85の非互換項目のすべてを指摘します。通常のOSIVプログラムの分散開発時に指定します。

なお、FLAGSW(GSS/GSW)で指摘される項目の一覧については“付録C COBOL85非互換指摘機能”を参照してください。

使用する文字コード系の変更

分散開発対象のOSIV系プログラムの翻訳チェックを通常のNetCOBOL製品で行う場合、文字コードの違いが原因で起こる翻訳結果の違いの問題は解決することはできません。
このような問題に対しては、NetCOBOL JEFオプションの使用が有効です。JEFオプションは使用する文字コード系としてOSIV系システムと同じEBCDIC/JEFを使用するものとして、プログラムを翻訳・リンクし、実行します。

図3-60 JEFオプション使用時の翻訳処理概要



文字コードの違いが原因で発生する問題の詳細については“第7章 トラブルシューティング”および“付録F 文字コード系”を参照してください。

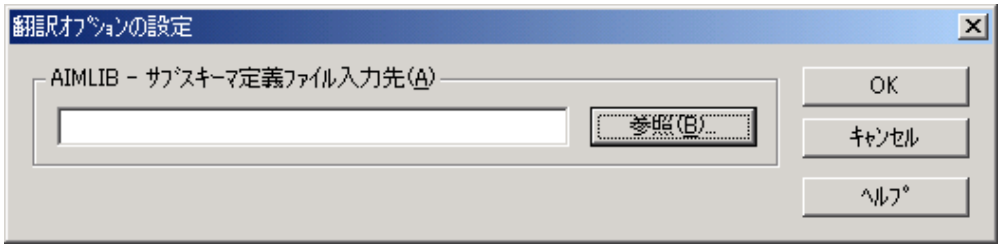
3.5.1.2 ネットワークデータベース機能を使用するプログラムの翻訳

ネットワークデータベース(NDB)機能を使用するOSIV系プログラムを翻訳するときには、サブスキーマの定義情報を取り込み、AIMとの連絡領域(FCOM)とデータベースとのデータのやり取りをするレコード(UWA)をプログラム内に展開する必要があります。この点が他のOSIV系プログラムを翻訳する場合と異なります。
Windows系システムでは、サブスキーマの定義情報はプログラムのサブスキーマ名段落に記述されているサブスキーマ名に拡張子CBLを付加した名前を持つファイルで、OSIV系システム上のAIMディレクトリからツールによって取り出します(“3.3.4 サブスキーマの移行”を参照)。

ネットワークデータベース(NDB)機能を使用するOSIV系プログラムの翻訳時には次の翻訳オプションを指定してください。

- AIMLIB(フォルダ):
  サブスキーマ定義ファイルを格納したフォルダを指定します。
  COBOLプロジェクトマネージャの〔翻訳オプションの追加〕ダイアログで、“AIMLIB”を選択すると次のダイヤログが表示されます。エディットボックスに直接、サブスキーマ定義ファイルを格納したフォルダ名を入力するか、“参照”ボタンをクリックして“フォルダの参照”ダイアログでフォルダを選択してください。

図3-61 翻訳オプションAIMLIBの設定ダイアログ



サブスキーマ定義ファイルが複数のフォルダに分けて格納去れている場合、セミコロンで区切って複数指定します。“参照”ボタンによるフォルダ名の参照を繰り返す事で同じ結果が得られます。なお、フォルダを複数指定した場合、指定された順序でフォルダが検索

されます。

- GEN/NOGEN:
  翻訳リストを出力するときに、翻訳リスト上にFCOMおよびUWAの展開を出力する(GEN)／しない(NOGEN)を指定します。
  翻訳オプションPRINT(翻訳リストの出力)およびSOURCE(ソースリストの出力)がともに指定されていないと有効となりません。

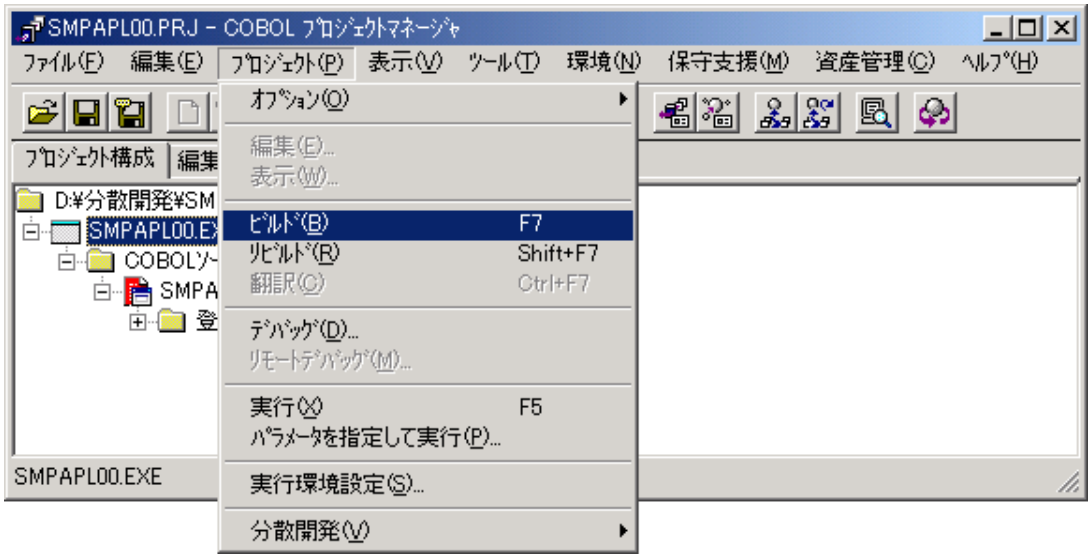
3.5.1.3 翻訳エラーの修正

翻訳チェックの結果、翻訳エラーが出力された場合、その診断メッセージに従って、COBOLソース・登録集ファイル等を修正します。
多くの診断メッセージは、OSIV系システムで翻訳をする場合と同じです。
しかし、OSIV系システムのCOBOL85とWindows系システムのNetCOBOLの機能差などから、まったく予想もしなかったエラーを告げられる場合もあります。その種の翻訳エラーで、よく見受けられるものについては“第7章 トラブルシューティング”を参照してください。

3.5.2 OSIV用プログラムのリンク

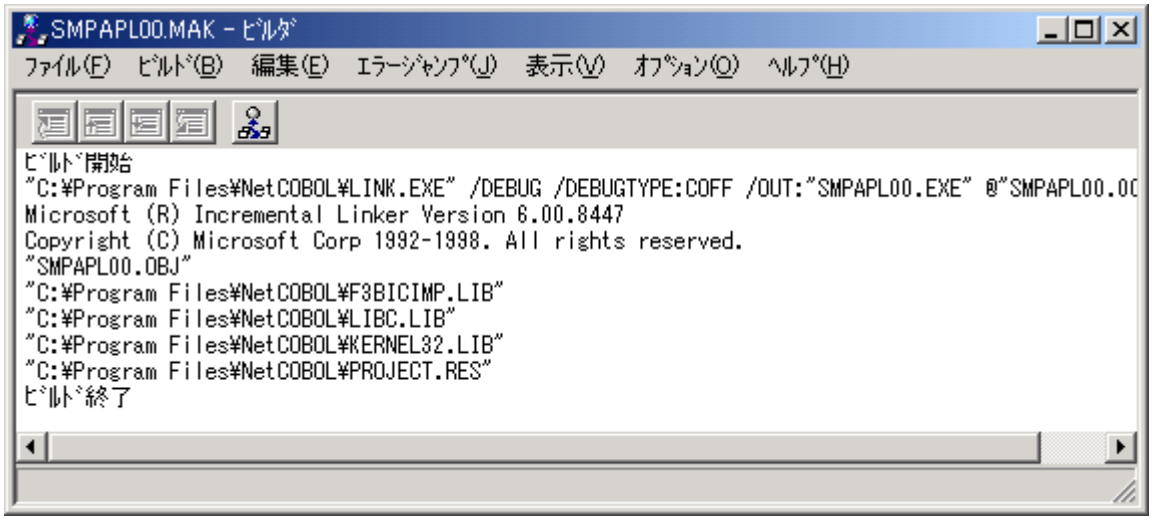
分散開発でのOSIV系プログラムのリンクは、COBOLプロジェクトマネージャの“ビルド”機能を使用して行います。
COBOLプロジェクトマネージャで“最終ターゲットファイル”を選択します。
〔プロジェクト〕メニューから“ビルド”を選択します(“Ｆ７”キーでも可)。

図3-62 プログラムのリンク処理(“ビルド”機能を使用)



〔ビルダ〕ウィンドウが開かれ、プログラムのリンクが行われます。なお、翻訳チェックの済んでいないCOBOLソースがあるなら、その翻訳が先に行われます。

図3-63 リンク結果の例



第4章 単体テスト

グローバルサーバ上においても、Windows系システム上においても単体テストで確認すべき点には違いはありません。
NetCOBOLは、GUIデバッガをはじめとして、これらを効率よく確認するための機能を備えています。しかし、その反面、さまざまな原因からなるプログラムの動作の非互換にまどわされる可能性も否定できません。
ここでは次のような分散開発の単体テストをWindowsシステム上で行う場合について、次のことを説明します。

- 単体テストをWindows系システムで行なう場合のメリットとデメリット
- OSVI系プログラムのWindows系システムでの実行の手順
- NetCOBOLの備える単体テストのための機能

4.1 Windows系システムでの単体テストについて

OSIV系プログラムの分散開発の場合であっても単体テストとは、開発したプログラム単位に以下の点を確認する作業です。

- プログラムの入出力の確認
- プログラム内の処理ロジックの確認

しかし、その作業手順などは、グローバルサーバで実施する場合とWindows系システムで実施する場合では、大きく異なります。ここでは、その違いからくるメリットとデメリットを説明します。

4.1.1 Windows系システムでの単体テスト実施のメリット

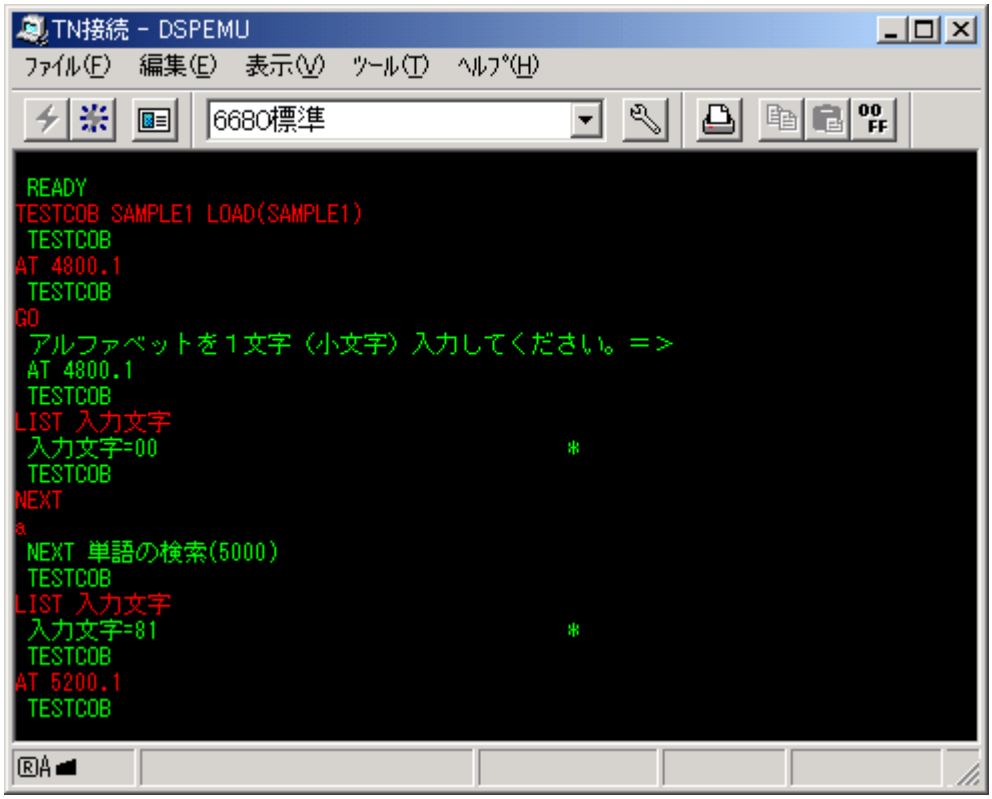
OSIV系プログラムの分散開発において、単体テストまでWindows系システムで実施することのメリットは大きく次の2つにわかれます。

- 使い易いデバッグツールの存在
- 独立したテスト環境構築の容易さ

使い易いデバッグツールの存在

OSIV系プログラムの分散開発を、単体テストまでWindows系システムで実施する場合に得られる最大のメリットは使い勝手が良く、機能の豊富なGUIデバッガが存在することです。使い易いデバッガの存在は単体テストに必須というわけではありませんが、短時間でプログラムの品質を向上させるために役立ちます。
グローバルサーバでCOBOLプログラムを対話的にデバッグするには、TESTCOBコマンドを使用します。

図4-1 TESTCOBコマンドによるCOBOLプログラムのデバッグ



TESTCOBコマンドを使用してデバッグする場合でも、PFD内での実行とPFDの画面分割オプションを併用することで、ソースプログラムを表示させながらデバッグを行うことができます。
しかし、対話的操作とその使い易さ／わかり易さという点では、複雑なGUIを使用できるWindows系システムのデバッグツールの方が優れています。

図4-2 PFD内からのTESTCOBコマンド呼び出しによるCOBOLプログラムのデバッグ

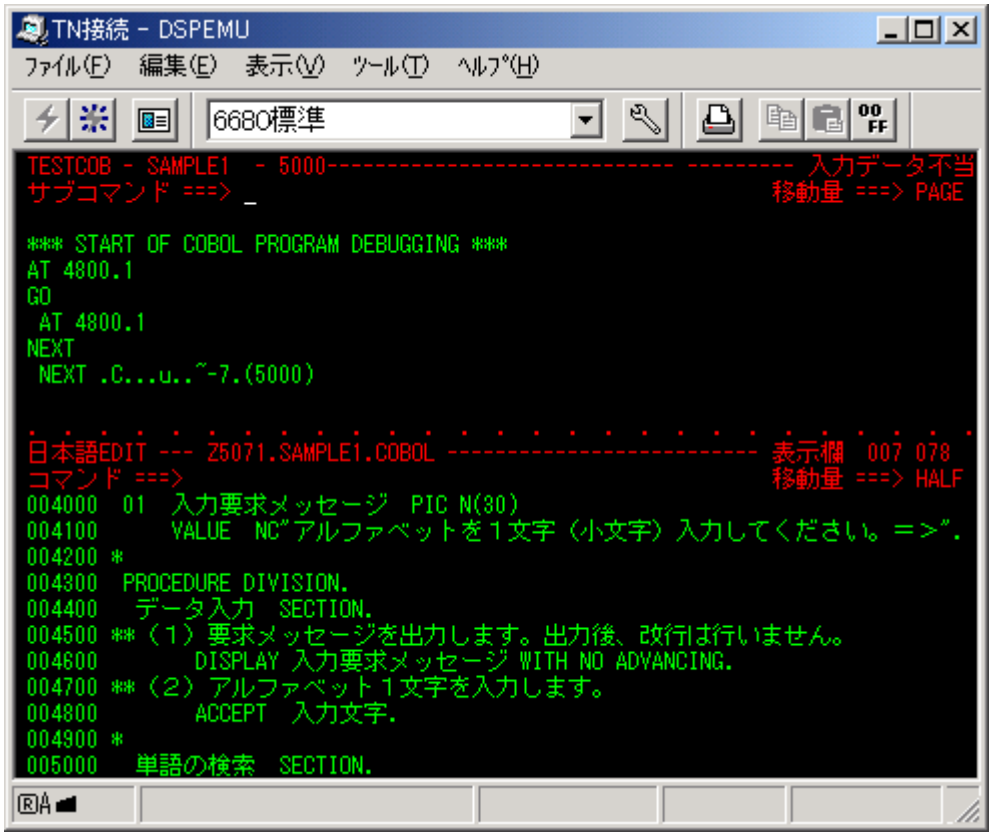
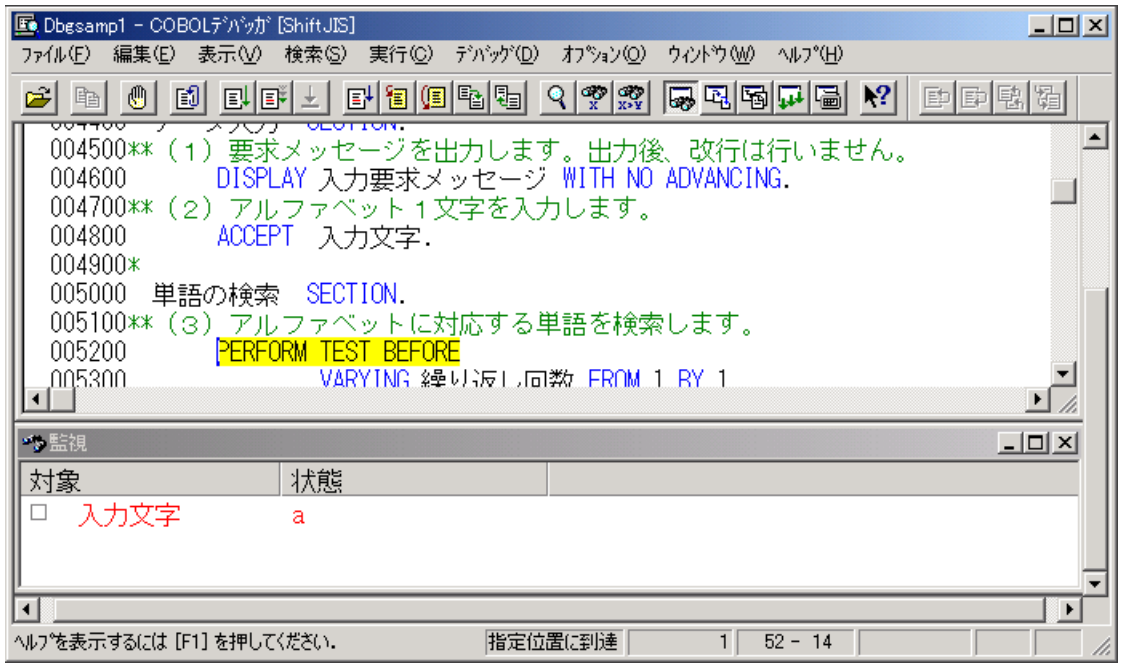


図4-3 Windows系システムでのCOBOLデバッガによるCOBOLプログラムのデバッグ



独立したテスト環境

グローバルサーバでは、ユーザの見えないところで、多くの資源や環境を共有しています。例えば、AIM環境とその配下のデータベース等の資源を共有するなら、データベースの更新処理をともなう複数のプログラムを同時にテストすることはできません。作業グループ毎にテスト環境を分割したり、スケジュールを調整して対応することは可能ですが、個々の開発者用に独立したテスト環境を準備するようなことは現実的でありません。

これに対して、Windows系システムでの分散開発では各作業者の使用するPC毎に独立したテスト環境を容易に構築できるので、真の作業の並列化が可能となります。これは開発効率の向上に役立ちます。また、テスト期間中のアクシデント(例えば、品質の十分でないプログラムが引き起こしたテストデータの破壊など)を局所化する働きもします。

4.1.2 Windows系システムでの単体テスト実施のデメリット

OSIV系プログラムの分散開発において、単体テストまでWindows系システムで実施することは必ずしも容易なことではありません。それは次の難しさがあるためです。

- 技術的な難しさ
- 作業の標準化に関する難しさ

技術的な難しさ

NetCOBOLは、グローバルサーバのCOBOL85をベースとして開発されたものですが、まったく同じ機能を持つものではありません。また、次のような点も同じプログラムについて、グローバルサーバと異なる結果を与える可能性があります。

- オペレーティングシステムの機能差
  文字コード系の違いやファイルシステムの持つ機能の違い等。
- サブシステムの機能差
  グローバルサーバでも、Windows系システムでも画面表示、帳票出力、通信、データベース管理などの機能はCOBOL以外の製品との連携によって実現されるものである。これらの提供する機能がグローバルサーバとWindows系システムでは異なる。

これらの詳細については“付録A OSIV系COBOLとオープン系COBOLの相違点”を参照してください。

作業の標準化に関する難しさ

OSIV系プログラムの単体テストをWindowsシステム上で実施するには、前述のような技術的な難しさが存在します。しかし、実際にOSIV系プログラムを分散開発する際、多くのプログラムでは単体テストが容易に実施可能であり、上記のような問題が発生するプログラムは全体の極一部であるということも少なくありません。

これは技術的な難しさとは別の次元の難しさをもたらします。

OSIV系プログラムはしばしば大量のプログラムを組み合わせたものとなります。その開発は大量の要員と時間を投入したものとなり、開発作業の標準化は不可欠となります。先に述べたように一部のプログラムだけ異なる作業手順が必要になるような状況は、作業の標準化をさまたげます。また、Windowsシステム上の単体テストの難易度に応じて、複数の標準を設けるという方法もありますが、どのプログラムがどの標準に合致するか機械的に判定する術がないことが、それを難しくしています。

4.2 OSIV系プログラムの実行

翻訳・リンクしたCOBOLプログラムを実行する際は、資源の割り当てや実行時オプションの指定などが必要になります。グローバルサーバでは、これらはプログラムを起動するJCLやCLISTで指定していました。
Windows系システムでは、これは次のようにして設定する必要があります。

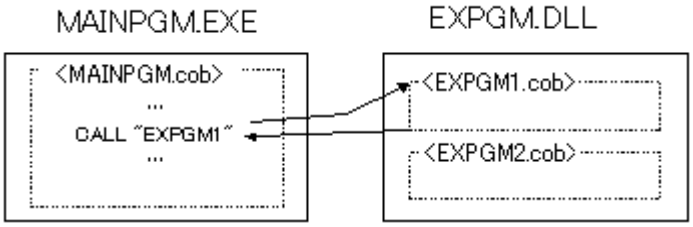
設定する情報	グローバルサーバでの指定方法		Windows系システムでの指定方法
	JCL	CLIST	
呼び出すロードモジュールやライブラリ等の格納場所	DD文/FD文(STEPLIB)	LIBコマンド	環境変数PATH
動的プログラム構造時の呼び出すプログラムを含むロードモジュール名	不要		エントリ情報(実行環境情報)
プログラムが必要とする資源(ファイル等)の割り当て	DD文/FD文(任意)	ALLOCATE文(任意)	実行環境情報
COBOLプログラムの実行に影響を与える実行時パラメタ	EXEC 文 の PARM パラメタ	CALL文のパラメタ	実行環境情報

以下、その設定方法について説明します。

4.2.1 環境変数PATHの設定

Windows系システムでは、実行可能ファイルが呼び出すロードモジュールやライブラリの格納場所は、環境変数PATHに設定しておく必要があります。例えば、実行しようとするプログラムが次のような構成であった場合、

図4-4 ダイナミックリンクライブラリを含むプログラム構成例



ダイナミックリンクライブラリ“EXPGM.DLL”が“MAINPGM.EXE”と異なるフォルダに有り、かつ、その格納場所が環境変数PATHに含まれていなかった場合、EXPGM1.DLLが見つからない場合があります。

図4-5 ダイナミックリンクライブラリが見つからない場合のエラーメッセージ

環境変数PATHの変更の方法は、Windows系システムの種類によって以下のように異なります。より詳細については、使用しているWindows系システムのヘルプを参照してください。

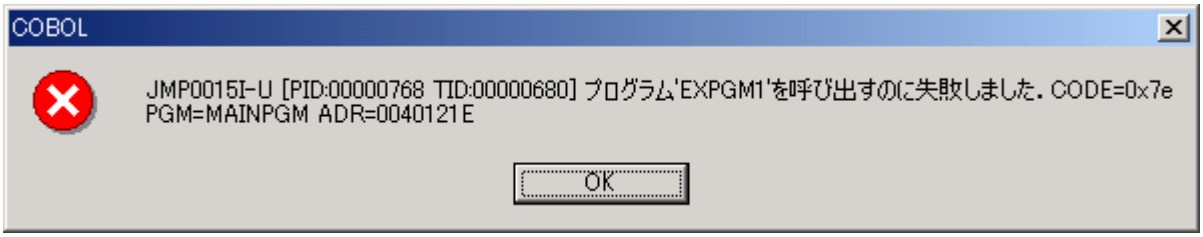
- Windows 2000 およびWindows XP
  プログラムの実行前に、コントロールパネルのシステムで環境変数を変更します。

なお、NetCOBOLのランタイムライブラリについては、NetCOBOL製品インストール時、インストールフォルダ名が環境変数PATHに追加されるので、この問題は発生しません。

4.2.2 エントリ情報の設定

NetCOBOLで作成した実行可能プログラムの構造が動的プログラム構造の場合、呼び出すプログラムが格納されているダイナミックリンクライブラリ (DLL)を特定するために、エントリ情報の指定が必要になります。

図4-6 動的プログラム構造でエントリ情報がない場合のエラーメッセージ



エントリ情報ファイルの指定方法

エントリ情報には次の2種類の指定が存在します。

- 副プログラムの格納されているDLL名を指定する。
- 二次入口点(ENTRY文で指定した入口点)を含む副プログラム名を指定する。

これらを以下の形式のファイル(エントリ情報ファイル)に保存し、そのファイル名を実行時オプション@CBR_ENTRYFILEに指定しておきます。

図4-7 エントリ情報ファイルの内部形式

```
[ENTRY]                                    [1]
エントリ情報                               [2]
   :
```

〔図の説明〕

[1]セクション名“[ENTRY]”を記述します。このセクションは、副プログラムに対するエントリ情報の定義の開始を意味し、エントリ情報ファイルに1つしか記述できません。

[2]エントリ情報を指定します。エントリ情報の指定形式については、以下に示します。

表4-1 エントリ情報の指定形式

エントリ情報の種別	エントリ情報の指定形式
副プログラム名の指定	副プログラム名=ダイナミックリンクライブラリ名
二次入口点の指定	二次入口点名=副プログラム名

エントリ情報ファイルの指定例

次のような構造のプログラムを例にエントリ情報ファイルの設定例を示します。

図4-8 エントリ情報ファイルが必要なプログラムの構成例

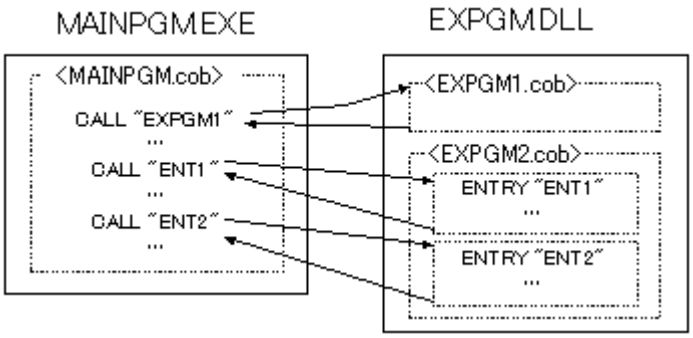


図4-9 エントリ情報ファイルの例

```
[ENTRY]
EXPGM1=EXPGM.DLL
EXPGM2=EXPGM.DLL
ENT1=EXPGM2
ENT2=EXPGM2
```

4.2.3 COBOL実行環境情報の設定

NetCOBOLで指定したプログラムを実行する際に必要な次の情報をCOBOL実行環境情報として設定します。

- プログラムが必要とする資源(ファイル等)の割り当て
- COBOLプログラムの実行に影響を与える実行時パラメタの指定

4.2.3.1 プログラムが必要とする資源の割り当て

プログラムが必要とするファイルなどの資源の割り当ては、Windows系では次の構文で行います。

```
環境変数名=割り当て資源名(ファイル/フォルダ名)
```

〔構文の説明〕

1. 環境変数名はグローバルサーバにおけるDD名(MSP)/アクセス名(XSP)にあたります。
2. 割り当て資源名はグローバルサーバにおけるDA名にあたります。
3. グローバルサーバのDD文/FD文における割り当て資源の扱いに関する各種の指定を指定する方法はありません。

以下、いくつか個別の例をあげて、説明します。

通常の入出力ファイル（順／相対／索引ファイル）

〔ソース記述例〕

```
          :
    SELECT SQFILE ASSIGN TO DA-DATFILE1
       ORGRANIZATION IS SEQUENTIAL
          :
```

〔設定例〕

```
DATFILE1=D:¥分散開発¥SMPAPL02¥DATFILE1
```

ＳＹＳＩＮ／ＳＹＳＯＵＴ

通常は特に割り当ての必要がありません。通常は実行したプログラムのCOBOLコンソールウィンドウが自動的に割り当てられます。

SYSIN/SYSOUTをファイルに割り当てる場合は、次のようにする必要があります。

1. 翻訳オプションSSINまたはSSOUTに入出力先となるファイルを割り当てる環境変数を指定してプログラムを翻訳・リンクします。

翻訳オプション名	機能	指定方法
SSIN	SYSINの入力先を指定	SSIN({SYSIN ｜ 環境変数名})
SSOUT	SYSOUTの出力先を指定	SSOUT({SYSOUT ｜環境変数名})

2. 翻訳オプションSSINまたはSSOUTに指定した環境変数名に対して、ファイルを割り当てます。
   例えば、SSIN(INF)およびSSOUT(OUTF)を指定して、プログラムを翻訳・リンクしている場合、割り当ては次のように行います。

```
INF=D:¥分散開発¥SMPAPL02¥INF.TXT
OUTF= D:¥分散開発¥SMPAPL02¥OUTF.TXT
```

印刷ファイル（OSIV系の物理順ファイル）

LINAGE句やADVAINCING付きWRITE文などを使用するファイルは、印字用デバイスとして、以下のいずれかを割り当てます。

- プリンタ名
- ローカルプリンタポート名(”LPTn:”)
- シリアルポート名(”COMn:”)

印刷ファイルに通常のファイルを割り当てることが可能ですが、その場合、印字用の情報が正しく処理されません。行送り／改頁などが期待したとおりの結果を得られない場合があります。

整列併合機能使用時に必要な資源

整列併合機能を使用する場合、グローバルサーバでは以下の資源の割り当てが必要となりますが、NetCOBOLでは特に割り当ての必要はありません。

- 整列併合プログラムの出力するメッセージの出力先SYSOUT
  →通常はメッセージボックスで出力されます。
- 整列併合ファイル

整列併合プログラムの出力するメッセージをファイルに出力する場合は実行時オプション@MessOutFileを使用します。

宛先ＤＳＰの表示ファイル使用時に必要な資源

フォーマット定義体とPSAMを使用するOSIV系の表示ファイルと画面帳票定義体(SMD)とMeFtを使用するWindows系の表示ファイルでは必要とする資源が異なります。
以下、例を元に説明します。

〔ソース記述例〕

```
          :
     SELECT SQFILE ASSIGN TO GS-DSPFILE
        SYMBOLIC DESTINATION IS “DSP”
          :
```

〔設定例〕

```
DSPFILE=D:¥分散開発¥SMPAPL03¥MEFWRC
```

割り当てるファイルはウィンドウ情報ファイルでなければなりません。

〔ウィンドウ情報ファイル設定例〕

```
TITLE ″ SMPAPL03″                          ←ウィンドウタイトル
WINSIZECX 80                               ←ウィンドウX方向サイズ
WINSIZECY 18                               ←ウィンドウX方向サイズ
MEDDIR D:¥分散開発¥SMPAPL03¥SMD            ←定義体格納フォルダ:必須
MEDSUF SMD                                 ←定義体拡張子
```

ウィンドウ情報ファイルの設定項目の詳細については“MeFt説明書”を参照してください。

宛先ＰＲＴの表示ファイル使用時に必要な資源

フォーマット定義体とPSAMを使用するOSIV系の表示ファイルと画面帳票定義体(SMD)とMeFtを使用するWindows系の表示ファイルでは必要とする資源が異なります。
以下、例を元に説明します。

〔ソース記述例〕

```
          :
     SELECT SQFILE ASSIGN TO GS-PRTPFILE
        SYMBOLIC DESTINATION IS “PRT”
          :
```

〔設定例〕

```
PRTFILE=D:¥分散開発¥SMPAPL03¥MEFWRC
```

割り当てるファイルはプリンタ情報ファイルでなければなりません。

〔プリンタ情報ファイル設定例〕

```
PRTID    ″ SMPAPL03″                       ←印刷名
PRTDRV ″FUJITSU FMLBP224″                  ←プリンタデバイス名
MEDDIR  D:¥分散開発¥SMPAPL03¥SMD           ←定義体格納フォルダ:必須
MEDSUF   SMD                               ←定義体拡張子
```

プリンタ情報ファイルの設定項目の詳細については“MeFt説明書”を参照してください。

その他の宛先の表示ファイル

分散開発対象のOSIV系プログラムで宛先が“DSP”および“PRT”以外の表示ファイルを使用している場合、そのプログラムは単体で実行することはできず、実行は対話型デバッガ上での擬似的なものとなります。
このため、それらの表示ファイルのために特に資源を割り当てる必要はありません。

ネットワークデータベース機能使用時に必要な資源

分散開発対象のOSIV系プログラムでネットワークデータベース機能を使用している場合、そのプログラムは単体で実行することはできず、実行は対話型デバッガ上での擬似的なものとなります。このため、それらのネットワークデータベースのために特に資源を割り当てる必要はありません。

4.2.3.2 実行時オプション

NetCOBOLでは、実行時のふるまいを指定するためにさまざまな実行時オプションが指定可能ですが、ここではOSIV系の分散開発を実施する際に重要となるもののみ説明します。その他の実行時オプションについては “NetCOBOL使用手引書”を参照してください。

エントリ情報ファイルの指定

プログラムの実行にエントリ情報が必要である場合、それを含むエントリ情報ファイルを指定します。

```
@CBR_ENTRYFILE=エントリ情報ファイル名
```

エントリ情報ファイル名には、絶対パスと相対パスを指定できます。相対パスが指定された場合は、実行している実行可能ファイルが存在するフォルダからの相対パスになります。

実行時メッセージの出力先の指定

プログラムの実行時に、COBOLランタイムから出力されるメッセージは通常、メッセージボックスに出力されます。このメッセージをファイルに出力したい場合に指定します。

```
@MessOutFile=出力先ファイル名
```

出力先ファイル名には、絶対パスと相対パスを指定できます。相対パスが指定された場合は、実行している実行可能ファイルが存在するフォルダからの相対パスになります。

OSIV形式の実行時パラメタの指定

```
@MGPRM=”実行時パラメタの文字列”
```

プログラムに渡す文字列を二重引用符(”)で囲んで指定します。指定した文字列は、グローバルサーバでプログラムを実行させたときと同様の形式で、プログラムに渡されます。
実行時パラメタの文字列は、最大で100バイトまで指定できます。

図4-10 プログラムの記述例

```
      :
LINKAGE SECTION.
01 パラメタ.
  03 パラメタ長 PIC 9(4) BINARY.
  03 パラメタ文字列.
    05 文字 PIC X OCCURS 1 TO 100 TIMES
                DEPENDING ON パラメタ長.
PROCEDURE DIVISION USING パラメタ.
      :
```

COBOLコンソールウィンドウを閉じる際のメッセージの出力

ACCEPT/DISPLAYの入出力機能で使用するCOBOLコンソールウィンドウを閉じる際に確認のメッセージを出力するかどうかを指定します。

```
@WinCloseMsg= {ON | OFF}
```

明示的に指定しない場合、ONが指定されているものと見なします。

トレース情報の出力先

TRACE機能を使用する際、トレース情報を出力するファイル名を指定します。

```
@CBR_TRACEFILE=トレース情報の出力先ファイル名
```

出力先ファイル名には、絶対パスと相対パスを指定できます。相対パスが指定された場合は、実行している実行可能ファイルが存在するフォルダからの相対パスになります。
なお、ファイル名に拡張子を指定しても、その拡張子は無視され、TRCおよびTROに置き換わります。

表示ファイルのファイルから接続する製品名

表示ファイルの接続製品名をデフォルトのものから変更するために指定します。宛先APLの表示ファイルの単体テストをPC上で実施する場合に必要になります(他の宛先については不要)。
次のように指定します。

```
@CBR_PSFILE_APL=DEBUG
```

4.2.3.3 COBOL実行環境の設定方法

NetCOBOLで作成したCOBOLプログラムの実行に際し、必要な情報をまとめてCOBOL実行環境情報と呼びます。実行環境情報の設定方法は次の方法があります。

a. システムの環境変数に指定する
b. 実行用の初期化ファイルに環境変数情報を設定する
c. コマンドラインで設定する(実行時オプション、実行時パラメタ(OSIV系形式だけ))

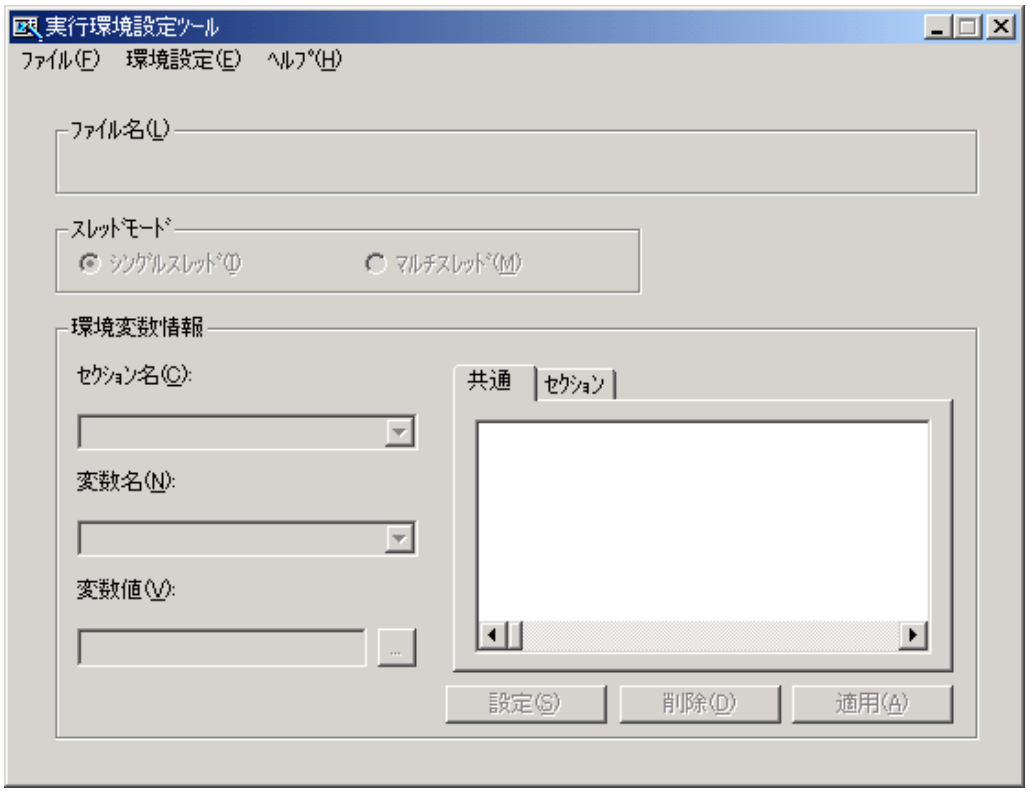
OSIV系プログラムの分散開発などの場合はb.の初期化ファイルに設定する方法が最も適切です。ここではb.の方法のみ説明します。その他の方法については“NetCOBOL使用手引書”を参照してください。

実行用の初期化ファイルの作成／オープン

新しく初期化ファイルを作成する場合、あるいは既存の初期化ファイルを開く場合、次の操作を行います。

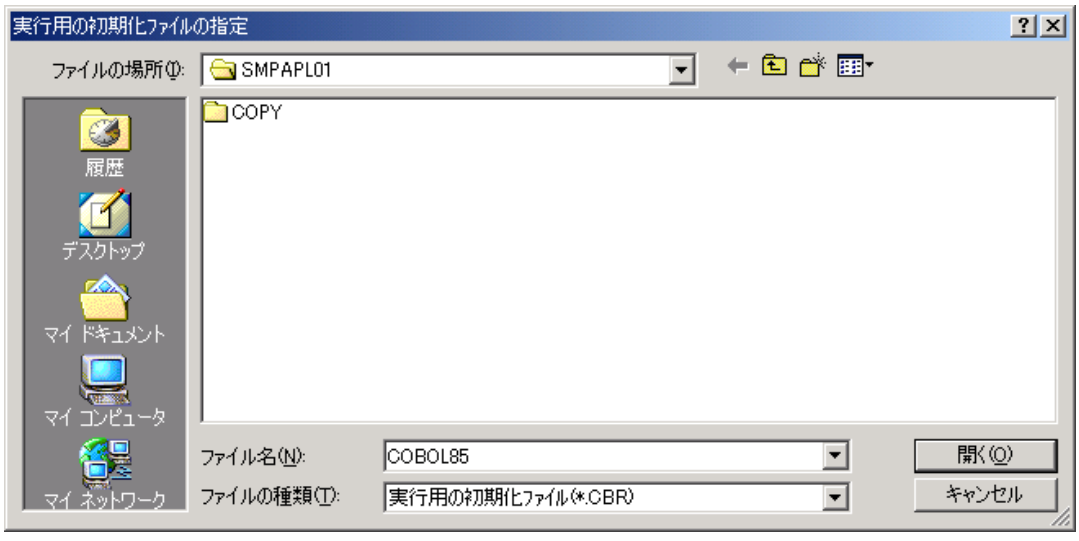
1. 開発対象プログラムのプロジェクトをCOBOLプロジェクトマネージャで開きます。
2. 〔ツール〕メニューから“実行環境設定ツール”を選択すると、実行環境設定ツールが起動されます。

図4-11 実行環境設定ツール(初期状態)



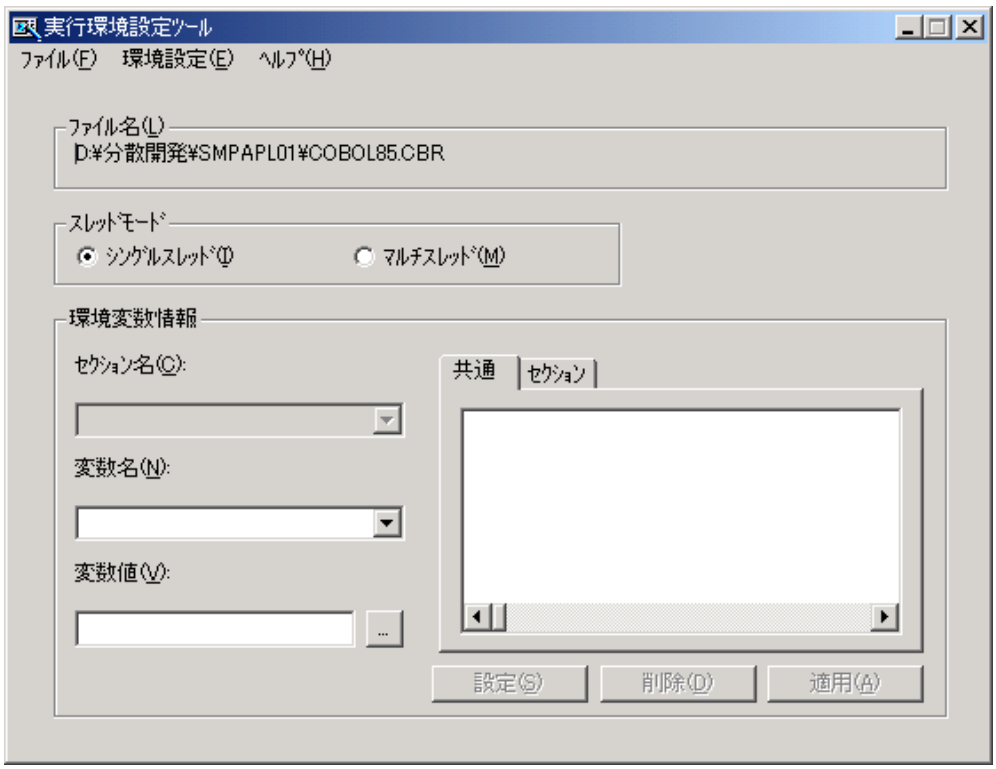
3. 実行環境設定ツールの〔ファイル〕メニューの“開く”を選択すると、実行するプログラムの存在するフォルダの〔実行用の初期化ファイルの指定〕ダイアログが表示されます。新規に実行用の初期化ファイルを作成する場合は、そのファイル名としてエディットボックスに“COBOL85”と入力してください。既存のファイルを開く場合は、ファイルを選択してください。その上で〔開く〕ボタンをクリックします。

図4-12 〔実行用の初期化ファイルの指定〕ダイアログ



4. COBOL85.CBRの設定内容が実行環境設定ツール内のリストボックスに表示されます。
OSIV系プログラムの分散開発の場合、〔スレッドモード〕は“シングルスレッド”として、〔セクション〕は使用しませんので、次の状態になります。

図4-13 実行環境設定ツール(初期化ファイルを開いた状態)

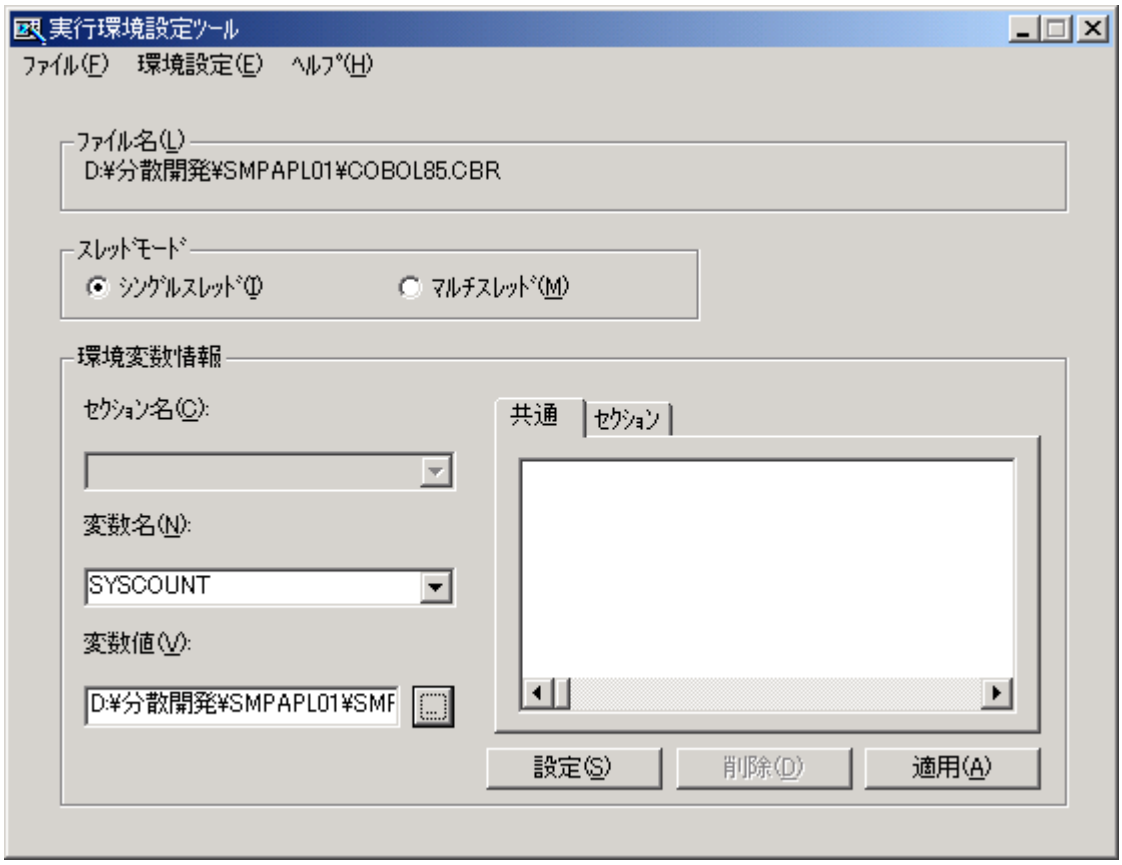


環境変数情報の追加

新しく環境変数情報を追加する場合、以下の操作を行います。

1. 〔変数名〕コンボボックスに環境変数名を直接入力するか、コンボボックスのリストから選択します（実行時オプションなどが選択可能です)。
2. 〔変数値〕に情報を設定します。ファイル名を指定する場合は、〔ファイルの指定〕ダイアログを開いて、そこから選択することもできます。
3. 〔設定〕ボタンをクリックすると、リストボックスに環境変数情報が追加され、変更を保存するための〔適用〕ボタンが有効になります。

図4-14 環境変数情報の追加

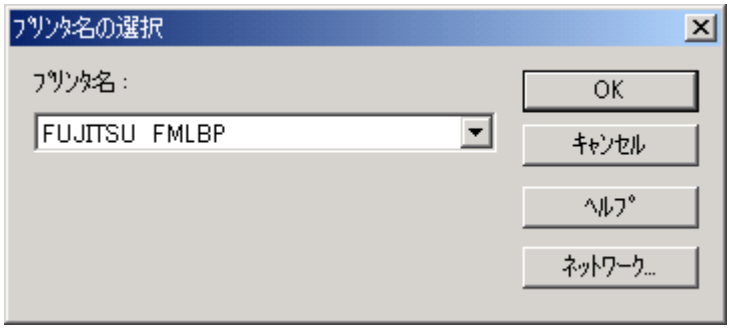


プリンタ名の設定

新しく環境変数情報を追加する際に、プリンタ名(印字用デバイス)を指定する場合、以下の操作を行います。

1. 〔変数名〕コンボボックスに環境変数名を直接入力します。
2. 〔環境変数〕メニューから“プリンタ名”を選択すると、〔プリンタ名の選択〕ダイヤログボックスが表示されます。

図4-15 〔プリンタ名〕の選択ダイアログ



3. 〔プリンタ名〕コンボボックスから使用可能なプリンタ名を選択すると、選択したプリンタ名が〔変数値〕に設定されるので、〔設定〕ボタンをクリックします。変更を保存するための〔適用〕ボタンが有効になります。

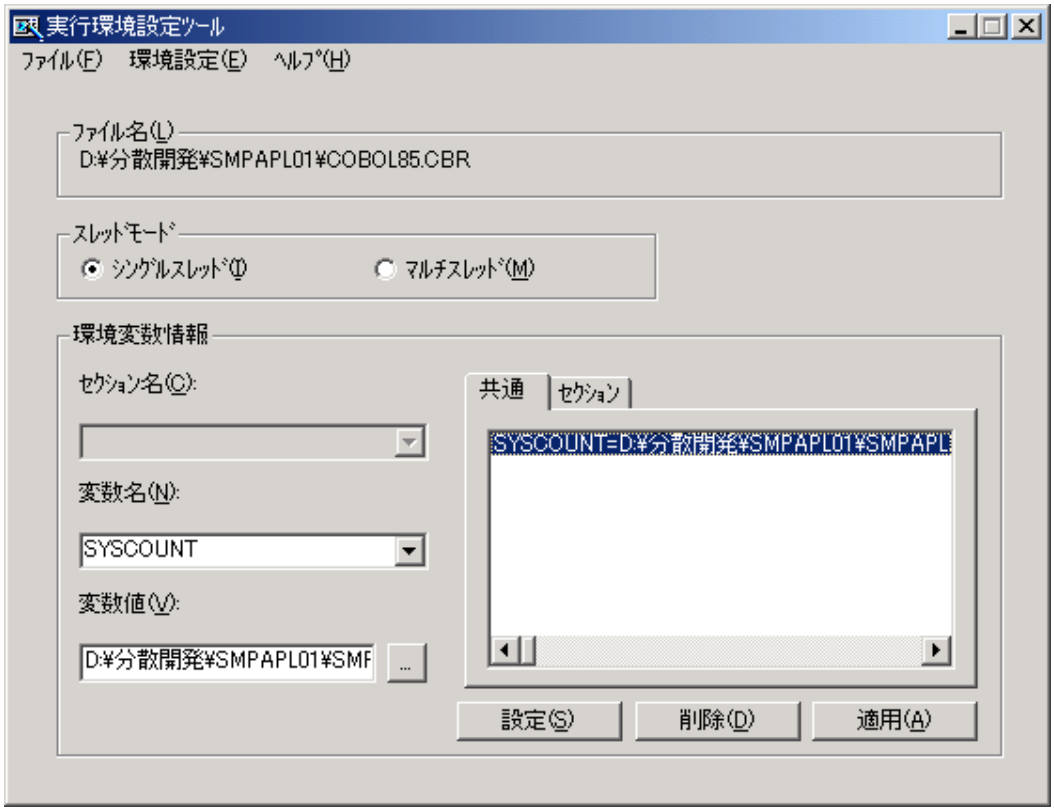
環境変数情報の変更

リストボックスに表示されている環境変数情報を変更する場合、以下の操作を行います。

1. リストボックスから変更する環境変数情報を選択します。選択した情報は、変数名および変数値のエディットボックスに表示されます。

2. 指定内容を変更します。
3. 〔設定〕ボタンをクリックすると、変更した環境変数情報がリストボックスに反映され、変更を保存するための〔適用〕ボタンが有効になります。

図4-16 変更する環境変数情報の選択



環境変数情報の削除

リストボックスに表示されている環境変数情報を削除する場合、以下の操作を行います。

1. リストボックスから変更する環境変数情報を選択します。選択した情報は、変数名および変数値のエディットボックスに表示されます。
2. 〔削除〕ボタンをクリックすると、削除した環境変数情報がリストボックスからなくなり、変更を保存するための〔適用〕ボタンが有効になります。

環境変数情報の初期化ファイルへの保存

リストボックスに表示されている環境変数情報を初期化ファイルに保存するためには、以下の操作を行います。

1. 〔適用〕ボタンをクリックすると、リストボックスに表示されている環境変数情報が初期化ファイルに保存されます。情報がファイルに保存されると、〔適用〕ボタンは無効になります。

4.3 COBOLのデバッグ機能

作成したCOBOLプログラムをデバッグする手段として、対話型デバッガを使用するのではなく、NetCOBOL自身の持つ次のデバッグ機能を使用することもできます。

- 誤った領域参照など代表的な誤りをチェックする機能(CHECK機能)
- 実行したCOBOLの文のトレース(TRACE機能)
- 実行したCOBOLの文ごと、文種別ごとの実行回数とその比率を出力(COUNT機能)

これらのデバッグ機能は、非対話的に実行されて、出力する結果がある場合、それらをファイルに出力します。このため、対話型デバッガを使用してのデバッグでは見つけづらい次のような問題を発見するのに力を発揮します。

- 多数の繰り返し処理の実施後に発生する問題
- 他のプログラムから多数回呼び出した末に発生する問題
- 問題の発生する条件が確定できない問題

4.3.1 CHECK機能

CHECK機能は、プログラム実行時に以下の誤り検査を行います。

- データ例外(属性形式に合った値が数字項目に入っているかおよび除数がゼロでないか)
- 添字・指標および部分参照の範囲外検査
- INVOKE文のパラメタと呼び出すメソッドの仮パラメタの適合検査
- CALL文によるプログラム呼び出し時のリンケージ規約の検査
- CALL文によるプログラム呼び出し時の引数、返却項目の検査

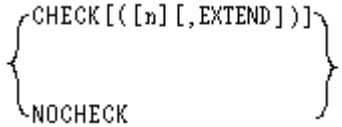
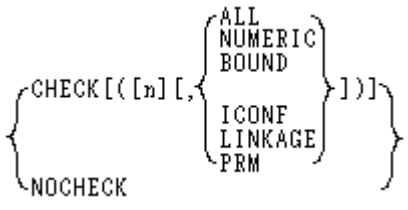
異常を検出するとエラーメッセージを出力して、プログラムの実行を強制的に終了させます(指定した回数の異常が検出されるまで、続行するように指定することもできます)。

4.3.1.1 OSIV系COBOL85との相違点

機能の概要と指定する翻訳オプション名は同じですが、翻訳オプション形式や検査項目の詳細は単純な対応がとれるものではありません。

翻訳オプション形式の違い

表4-2 CHECKオプションの形式の相違

OSIV COBOL85	NetCOBOL
	

検査項目の相違

表4-3 CHECKオプションによる検査項目の相違

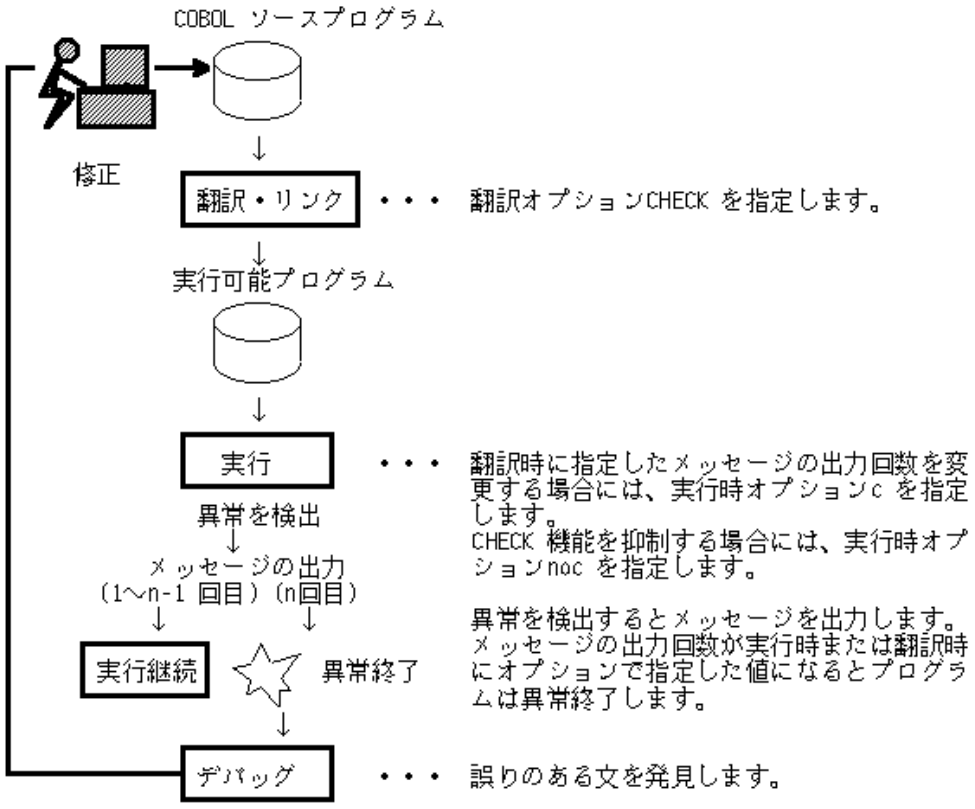
検査項目	OSIV COBOL85の指定	NetCOBOLの指定	備考
数字のデータ例外検査	−	CHECK(NUMERIC)	

除数０検査	－		OSIV系では項目属性によりハードウェア例外が発生する場合もあるが、基本的にチェックされない
添字・指標の範囲検査	CHECK	CHECK(BOUND)	
部分参照長の範囲検査			
OCCURS DEPENDING ON 句の目的語検査			
メソッド呼び出し時の引数検査	－	CHECK(ICONF)	
プログラムの呼び出し規約検査	－	CHECK(LINKAGE)	
プログラム呼び出し時の引数検査	CHECK(EXTEND)	CHECK(PRM)	NetCOBOL V7.2以前は制限付き
けたあふれ検査		－	
ファイルのレコード領域のアクセス検査			

4.3.1.2 CHECK機能を使用したデバッグの手順

以下にCHECK機能を使ったデバッグ作業の流れを示します。

図4-17 CHECK機能を使用したデバッグ作業の流れ



4.3.1.3 検査項目の詳細

ここでは、NetCOBOLのCHECK機能によって、チェックされる検査項目の詳細を示します。

数字のデータ例外検査

CHECK(NUMERIC)またはCHECK(ALL)を指定して翻訳したプログラムについて、参照した数字項目に属性形式と異なるデータが入っていないかを、そのプログラムの実行時にチェックします。

ソース記述例

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.    PGM1.
            :
000050  01  文字   PIC X(4)   VALUE "ABCD".
000060  01  外部１０進  REDEFINES   文字   PIC S9(4).
000070  01  数字   PIC S9(4).
            :
000150     MOVE   外部１０進   TO   数字.
            :
```

実行結果

MOVE文を実行するときに、外部10進に対して以下のメッセージが出力されます。

```
JMP0828I-E/U 〔PID:xxxxxxxx TID:xxxxxxxx〕  属性と異なる形式のデータが格納されています.
                                          PGM=PGM1. LINE=150. OPD=外部10進.
```

除数のゼロ検査

CHECK(NUMERIC)またはCHECK(ALL)を指定して翻訳したプログラムについて、除算の除数が０でないかを、そのプログラムの実行時にチェックします。

ソース記述例

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.    PGM1.
            :
000060  01  被除数  PIC S9(8)  BINARY  VALUE  1234.
000070  01  除数    PIC S9(4)  BINARY  VALUE  0.
000080  01  結果    PIC S9(4)  BINARY  VALUE  0.
            :
000150     COMPUTE  結果 =  被除数 /  除数.
            :
```

実行結果

COMPUTE文を実行するときに、除数に対して以下のメッセージが出力されます。

```
JMP0829I-E/U 〔PID:xxxxxxxx TID:xxxxxxxx〕  除数にゼロが指定されています.
                                          PGM=PGM1. LINE=150. OPD=除数
```

添字および指標検査

CHECK(BOUND)またはCHECK(ALL)を指定して翻訳したプログラムについて、表の参照時、添字・指標が表の範囲外を示すものでないかを、そのプログラムの実行時にチェックします。

ソース記述例

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.    PGM1.
            :
000500  77  添字   PIC S9(4).
000600  01  表.
000700      02  表１  OCCURS 10 TIMES INDEXED BY  指標１.
000800          03  要素１  PIC 9(5).
            :
001100     MOVE  15     TO  添字.
001200     ADD    1     TO  要素１ (添字).
001300     SET  指標１  TO  0.
001400     SUBTRACT  1  FROM  要素１ (指標１).
            :
```

実行結果

ADD/SUBTRACT文を実行するときに以下のメッセージが出力されます。

```
JMP0820I-E/U 〔PID:xxxxxxxx TID:xxxxxxxx〕  添字または指標の値が範囲外を指しています.
                                          PGM=PGM1. LINE=1200.1. OPD=要素１
JMP0820I-E/U 〔PID:xxxxxxxx TID:xxxxxxxx〕  添字または指標の値が範囲外を指しています.
                                          PGM=PGM1. LINE=1400.1. OPD=要素１
```

部分参照検査

CHECK(BOUND)またはCHECK(ALL)を指定して翻訳したプログラムについて、部分参照時、その参照位置がデータの領域外でないかを、そのプログラムの実行時にチェックします。

ソース記述例

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.    PGM1.
            :
000500  77  データ１        PIC X(12).
000600  77  データ２        PIC X(12).
000700  77  参照する長さ    PIC 9(4) BINARY.
            :
001100     MOVE  10   TO  参照する長さ.
001200     MOVE   データ１ (1:参照する長さ) TO  データ２ (4:参照する長さ).
            :
```

実行結果

1200行のMOVE文を実行するときに、データ２に対して以下のメッセージが出力されます。

```
JMP0821I-E/U 〔PID:xxxxxxxx TID:xxxxxxxx〕  参照可能範囲外の部分参照を行っています.
                                          PGM=PGM1. LINE=1200.1. OPD= データ２.
```

OCCURS DEPENDING ON句の目的語検査

CHECK(BOUND)またはCHECK(ALL)を指定して翻訳したプログラムについて、OCCURS DEPENDING ON句を含む項目の参照時、その目的語の値がOCCURS句の最大繰り返し数を越えない範囲にあるかを、そのプログラムの実行時にチェックします。

ソース記述例

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.    PGM1.
            :
000500  77  添字   PIC S9(4).
000600  77  個数   PIC S9(4).
000700  01  表.
000800      02  表１  OCCURS 1 TO  10 TIMES DEPENDING ON  個数.
000900          03  要素    PIC X(5).
            :
001100     MOVE   5   TO  添字.
001200     MOVE  25   TO  個数.
001300     MOVE  "ABCDE"  TO  要素 (添字).
            :
```

実行結果

1200行のMOVE文を実行するときに、個数に対して以下のメッセージが出力されます。

```
JMP0822I-E/U 〔PID:xxxxxxxx TID:xxxxxxxx〕 ODO句の目的語の値が許容範囲を超えています.
                                          PGM=PGM1. LINE=120.1.  OPD=要素. ODO=個数.
```

メソッド呼出しのパラメタの検査

CHECK(ICONF)またはCHECK(ALL)を指定して翻訳したプログラムについて、メソッド呼び出し時に指定した実パラメタが実際に呼び出されるメソッドの仮パラメタに適合するかを、そのプログラムの実行時にチェックします。

OSIV系プログラムの分散開発では使用しないため、説明の詳細は省略します。

プログラム呼び出し規約検査

CHECK(LINKAGE)またはCHECK(ALL)を指定して翻訳したプログラムについて、プログラムの呼び出し時、CALL文に指定したプログラム呼び出し規約が実際に呼び出すプログラムのそれと一致するかを、そのプログラムの実行時にチェックします。

OSIV系プログラムの分散開発では使用しないため、説明の詳細は省略します。

プログラム呼出しのパラメタの検査

CHECK(PRM)またはCHECK(ALL)を指定して翻訳したプログラムについて、プログラムの呼び出し時、CALL文に指定した実パラメタの個数と長さが実際に呼び出すプログラムの仮パラメタとそれと一致するかを、チェックします。

呼び出されるプログラムが内部プログラムの場合は翻訳時に、外部プログラムの場合は実行時にチェックされます。

ソース記述例1

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. PGM1.
            :
000700 DATA DIVISION.
000800 WORKING-STORAGE SECTION.
000900 01 P1 PIC X(20).
```

```
001000 01 P2 PIC X(10).
001100 01 P3 USAGE OBJECT REFERENCE FJBASE.
001200 PROCEDURE DIVISION.
001300     CALL "SUB1" USING P1 P2           *> JMN3333I-S
001400     CALL "SUB2"                       *> JMN3414I-S
001500     CALL "SUB1" USING P1 RETURNING P2  *> JMN3508I-S
001600     CALL "SUB1" USING P2              *> JMN3335I-S
001700     CALL "SUB3" USING P3              *> JMN3334I-S
            :

002000 PROGRAM-ID. SUB1.
002100 DATA DIVISION.
002200 LINKAGE SECTION.
002300 01 L1 PIC X(20).
002400 PROCEDURE DIVISION USING L1.
002500 END PROGRAM SUB1.
002600*
002700 PROGRAM-ID. SUB2.
002800 DATA DIVISION.
002900 LINKAGE SECTION.
003000 01 RET PIC X(10).
003100 PROCEDURE DIVISION RETURNING RET.
003200 END PROGRAM SUB2.
003300*
003400 PROGRAM-ID. SUB3.
003500 DATA DIVISION.
003600 LINKAGE SECTION.
003700 01 L-OR1 USAGE OBJECT REFERENCE.
003800 PROCEDURE DIVISION USING L-OR1.
003900 END PROGRAM SUB3.
004000 END PROGRAM PGM1.
```

翻訳結果1

このプログラムをCHECK(PRM)またはCHECK(ALL)を指定して、翻訳した場合、次の診断メッセージが出力されます。

```
13: JMN3333I-S  CALL文のUSING指定に記述したパラメタの個数は，PROCEDURE DIVISIONのUSING
                指定に記述したパラメタの個数と一致していなければなりません．
14: JMN3414I-S  'SUB2'を呼ぶCALL文にはRETURNING指定を記述しなければなりません．プログラ
                ム'SUB2'のPROCEDURE DIVISIONにRETURNING指定があります．
15: JMN3508I-S  'SUB1'を呼ぶCALL文にはRETURNING指定を記述することはできません．プログラ
                ム'SUB1'のPROCEDURE DIVISIONにRETURNING指定がありません．
16: JMN3335I-S  CALL文のUSING指定またはRETURNING指定に記述したパラメタ'P2'の長さは，プ
                ログラム'SUB1'のPROCEDURE DIVISIONのUSING指定またはRETURNING指定に記述
                したパラメタ'L1'の長さと一致していなければなりません．
17: JMN3334I-S  CALL文のUSING指定またはRETURNING指定に記述したパラメタ'P3'の型は，プロ
                グラム'SUB3'のPROCEDURE DIVISIONのUSING指定またはRETURNING指定に記述し
                たパラメタ'L-OR1'の型と一致していなければなりません．
```

ソース記述例2

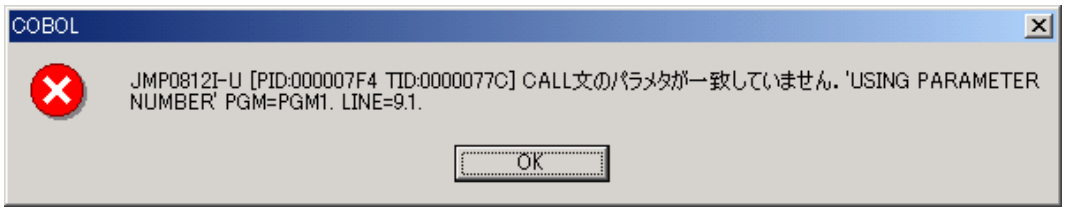
```
[PGM1.cob]
```

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.    PGM1.
000300 ENVIRONMENT    DIVISION.
000400 DATA           DIVISION.
000500 WORKING-STORAGE SECTION.
000600 01 WK1        PIC X(2).
000700 01 WK2        PIC X(2).
           …
002800 PROCEDURE      DIVISION.
           …
005900     CALL "EXSUB1" USING WK1 WK2.
           …
007000 END PROGRAM    PGM1.
[EXSUB1.cob]
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.    EXSUB1.
000300 ENVIRONMENT    DIVISION.
000400 DATA           DIVISION.
000500 LINKAGE         SECTION.
000000 01 LK1        PIC X(2).
000000 01 LK2        PIC X(2).
000000 01 LK3        PIC X(4).
000600 PROCEDURE      DIVISION USING LK1 LK2 LK3.
           …
002700 END PROGRAM    EXSUB1.
```

実行結果2

このプログラムをCHECK(PRM)またはCHECK(ALL)を指定して、翻訳・リンクしたプログラムを実行した場合、次の診断メッセージが出力されます。

図4-18 外部プログラム呼び出し時のパラメタチェックによるエラーメッセージ



4.3.2 TRACE機能

TRACE機能は次のような目的で使用されるデバッグ機能ですが、まったく同じ機能はOSIV COBOL85には存在しません(NetCOBOL固有の機能)。

- どの文で異常終了したのかを知りたい場合
- 異常終了までに実行した文の経路を知りたい場合
- 実行の途中で出力されたメッセージを確認したい場合

OSIV系のCOBOL85で同じことをする場合、STATEMENT機能およびFLOW機能を使用します（COBOL85固有機能)。

4.3.2.1 OSIV系COBOL85との相違点

TRACE機能とSTATEMENT機能/FLOW機能では、使用する目的は似ていますが、その機能、出力の条件などが異なります。以下、その詳細を説明します。

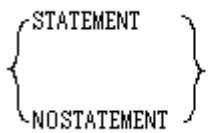
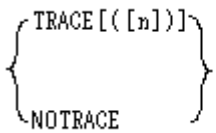
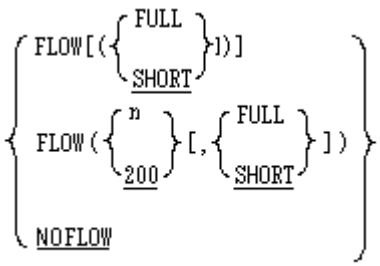
機能の相違

表4-4 STATEMENT/FLOW機能とTRACE機能の相違

OSIV COBOL85			NetCOBOL		
機能	出力条件	出力内容	機能	出力条件	出力内容
STATEMENT	実行時 エラー発生時	最後の実行文の行番号と動詞番号	TRACE	各文の実行	実行した文の行番号、動詞番号
FLOW	実行時 エラー発生時	最後の実行文から逆トレースを行って、その文の実行までの過程にある手続き名および文の行番号と動詞番号			実行した文を含むプログラム名とプログラム属性情報 実行中に出力されたメッセージ

翻訳オプション形式の違い

表4-5　STATEMENT/FLOW機能とTRACE機能のオプション形式の相違

OSIV COBOL85	NetCOBOL
	
	

出力先の相違

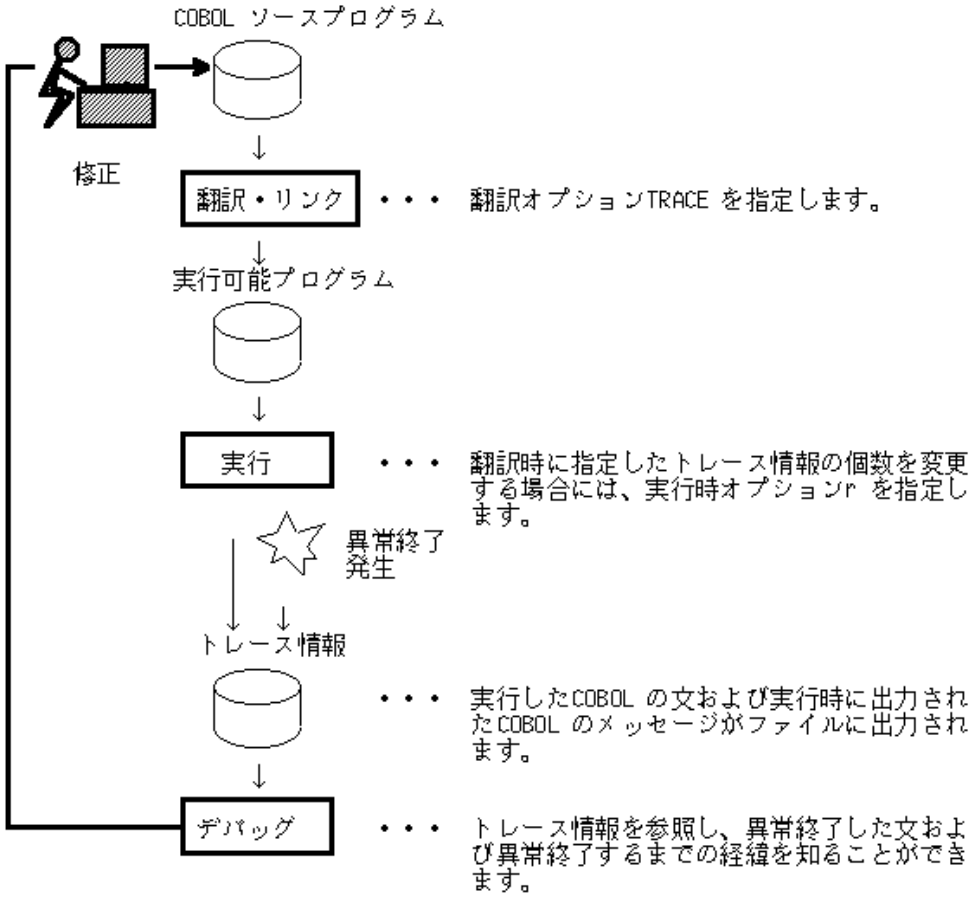
表4-6 OSIV STATEMENT/FLOWとTRACEの出力先の相違

OSIV COBOL85		NetCOBOL	
STATEMENT FLOW	DD名SYSDBOUT(MSP)のデータセット or アクセス名SYSDBOUT(XSP)のデータセット	TRACE	環境変数情報@CBR_TRACE_FILEに指定したファイルに格納します。 この環境変数が未設定の場合、実行形式ファイル名に拡張子TRCを付けたファイルに格納します。 トレース情報が翻訳時、または実行時の指定個数になる毎に拡張子TRCのファイルの内容は、拡張子TROのファイルに移されます。

4.3.2.2 TRACE機能を使用したデバッグの手順

以下にTRACE機能を使ったデバッグ作業の流れを示します。

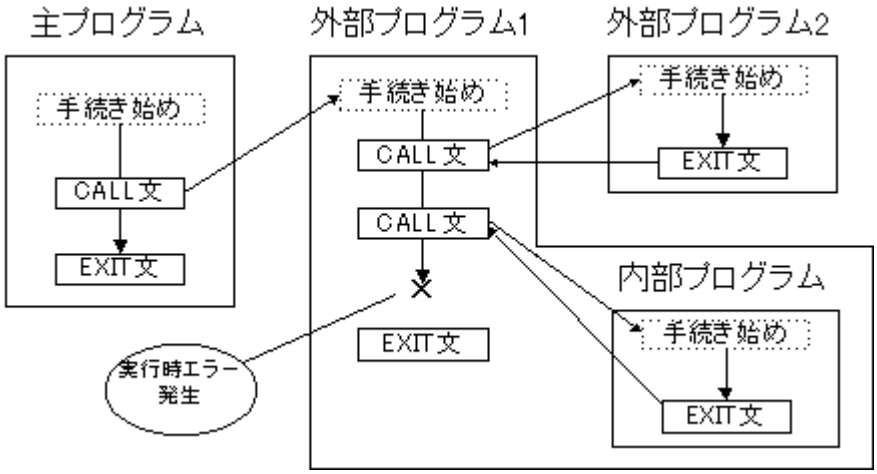
図4-19 TRACE機能を使用したデバッグ作業の流れ



4.3.2.3 TRACE情報の出力例

次のような構成のプログラムのトレース情報を採取したものとして、その出力例を示します。

図4-20 TRACE情報を採取するプログラムの構成図



トレース情報の出力例

```
NetCOBOL DEBUG INFORMATION                          DATE 2003-08-27  TIME 10:46:52
                                                                     PID=000006C0     〔1〕

TRACE INFORMATION
         〔2〕  〔3〕                              〔4〕              〔5〕
         1  MAINPGM    DATE 2003-08-27  TIME 10:46:44  TID=000006C0
         2             700.1 TID=000006C0                                          〔6〕
         3             800.1 TID=000006C0
         4  EXPGM1     DATE 2003-08-27  TIME 10:46:27  TID=000006C0
         5             1000.1 TID=000006C0
         6             1100.1 TID=000006C0
         7  EXPGM2     DATE 2003-08-27  TIME 10:46:28  TID=000006C0
         8             2000.1 TID=000006C0
         9             2100.1 TID=000006C0
        10  EXPGM1 TID=000006C0
        11             1200.1 TID=000006C0
        12  EXPGM1(INSUBPGM) TID=000006C0                                          〔7〕
        13             2400.1 TID=000006C0
        14             2500.1 TID=000006C0
        15  EXPGM1 TID=000006C0
        16             1300.1 TID=000006C0
        17             1400.1 TID=000006C0
        18             1500.1 TID=000006C0
        19    THE INTERRUPTION WAS OCCURRED.PID=00000728,TID=000006C0              〔8〕
```

〔図の説明〕

〔1〕 プロセスID(16進数表記 8桁)
プログラムを実行したとき、オペレーティングシステムにより割り当てられたプロセスを識別する番号が出力されます。

〔2〕 トレース情報の通番(10進数表記 10桁)
トレース情報を出力するたびにカウントアップされた値が表示されます。トレース情報は2つのファイルに交互に上書きされていくため、この値によりプログラムの開始から何番目の情報であるかがわかります。

〔3〕 プログラム名
トレース情報の出力対象のプログラム名が出力されます。内部プログラム中の場合は、次の形式で表示されます(〔7〕参照)。
外部プログラム名(内部プログラム名)

〔4〕 翻訳日付
外部プログラムが動作する場合、動作するプログラムの翻訳日時を出力します。

〔5〕 スレッドID(16進数表記 8桁)
プログラムを実行したとき、オペレーティングシステムにより割り当てられたスレッドを識別する番号が出力されます。

〔6〕 実行した文、手続き名/段落名
実行した文、手続き名または段落名の行番号を出力します。
行番号の出力形式は翻訳オプションNUMBERの指定に依存します。この例は、翻訳オプションNUMBERを指定した場合の例を示しています(NetCOBOLではNONUMBERがデフォルトです)。

〔7〕 内部プログラム呼び出し時
内部プログラム名が出力されます。

〔8〕 実行時エラー発生箇所
発生した実行時エラーにより、以下のいずれかが出力されます。

— 実行時メッセージ
プログラムの実行中にCOBOLランタイムシステムからメッセージが出力された場合、そのメッセージを出力します。詳細については、“NetCOBOL 使用手引書”の“F.3 実行時メッセージ”を参照してください。

— 例外通知メッセージ
オペレーティングシステムから例外(不当なアドレスを参照した場合など)が通知された場合、このメッセージを出力します。

4.3.3 COUNT機能

COUNT機能は次のような目的で使用されるデバッグ機能です。

- プログラムの実行した全ルートの走行を確認したい場合
- プログラムの効率化を図りたい場合

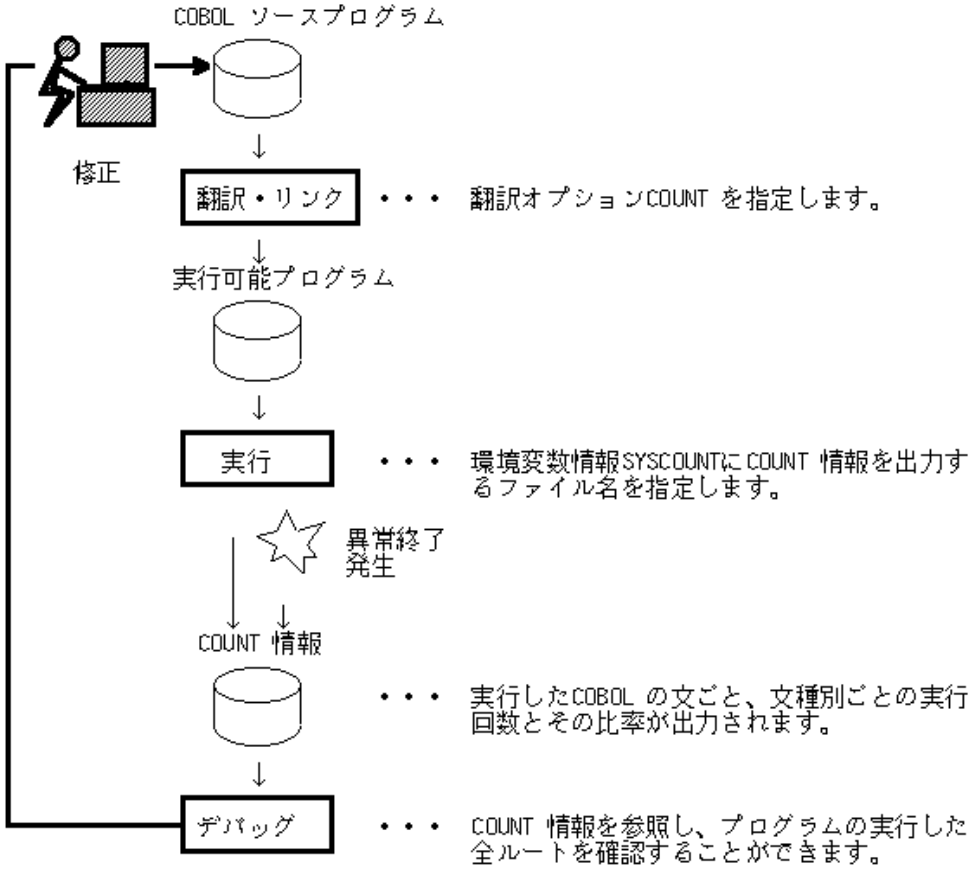
OSIV系のCOBOL85の持つCOUNT機能と違いはありません。以下のような情報を環境変数SYSCOUNTに対応付けたファイルに出力します。

- プログラム上の各文の実行回数および全文の全実行回数に対する各文の実行比率
- プログラム上の文種別ごとの実行回数とその比率

4.3.3.1 COUNT機能を使用したデバッグの手順

以下に、COUNT機能を使ったデバッグ作業の流れを示します。

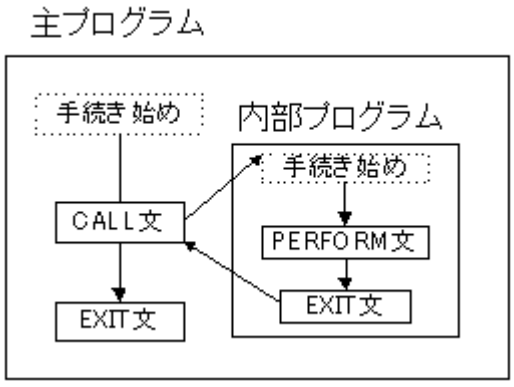
図4-21 COUNT機能を使用したデバッグ作業の流れ



4.3.3.2 COUNT情報の出力例

次のような構成のプログラムのCOUNT情報を採取したものとして、その出力例を示します。

図4-22 COUNT情報を採取するプログラムの概要



ＣＯＵＮＴ情報の出力例

COUNT情報は、大きく３つの部分に分けられるので、それぞれの部分ごとに説明します。

ソースプログラムイメージ実行回数リスト

```
 [1]
NetCOBOL COUNT INFORMATION(END OF RUN UNIT)          DATE 2003-08-30   TIME 11:28:59
                                                    PID=00000638  TID=00000438
[2]
  STATEMENT EXECUTION COUNT    PROGRAM-NAME : MAINPGM

     [3]          [4]                          [5]          [6]
     STATEMENT                                 EXECUTION    PERCENTAGE
       NUMBER       PROCEDURE-NAME/VERB-ID       COUNT          (%)
   ------------  ----------------------------  --------------  --------
            11  PROCEDURE DIVISION   MAINPGM
            12  データ入力
            14      DISPLAY                              1    3.1250
            16      ACCEPT                               1    3.1250
            19  単語の検索
            20      CALL                                 1    3.1250
            22  単語の表示
            24      DISPLAY                              1    3.1250
            26      EXIT PROGRAM                         1    3.1250
            69  PROCEDURE DIVISION   SUBPGM
            70      PERFORM                              1    3.1250
            73      IF                                  23   71.8750
            74      MOVE                                 1    3.1250
            75      EXIT PERFORM                         1    3.1250
            78      EXIT PROGRAM                         1    3.1250
   ------------  ----------------------------  --------------  --------
                                                        32
```

〔図の説明〕

〔1〕 COUNT情報ヘッダ

COUNT機能の出力ファイルであることを表し、()内は出力時期を表示します。出力時期には、次の4種類があります。

- END OF RUN UNIT
  COBOLの実行単位の終了時(STOP RUN文または主プログラムのEXIT PROGRAM文の実行時)に出力されます。
- ABNORMAL END
  異常終了時に出力されます。
- END OF INITIAL PROGRAM
  INITIAL属性を持つプログラムの終了時に出力されます。ただし、内部プログラム終了時には出力されません。
- CANCEL PROGRAM
  翻訳オプションCOUNTが有効なプログラムがCANCEL文によりキャンセルされた時点で出力されます。ただし、内部プログラムの終了時には出力されません。

〔2〕 ソースプログラムイメージ実行回数リスト

以降の出力がソースプログラムイメージ実行回数リストであることを示します。
この情報は、ソースプログラムの翻訳単位で出力します。翻訳単位がプログラムの場合、PROGRAM-NAMEには外部プログラム名を表示します。

〔3〕 ステートメント番号

ステートメント番号を次の形式で表示します。

[COPY修飾値-]行番号

1行に複数の文が存在する場合、2番目以降の文については、行番号を同じ値で表示します。

〔4〕 手続き名および文

手続き名および文を表示します。手続き部の始まりには、“PROCEDURE DIVISION”の文字列の後にプログラム名を表示します。

〔5〕 実行回数

文の実行回数を表示します。最後に実行回数の総数を表示します。

〔6〕 実行比率

その文の総実行回数に対する比率を表示します。

文別の実行回数リスト

```
[7]
 VERB EXECUTION COUNT          PROGRAM-NAME : MAINPGM

       [8]              [9]             [10]        [11]        [12]           [13]
                                                  PERCENTAGE                 PERCENTAGE
    VERB-ID          ACTIVE VERB     TOTAL VERB         (%)  EXECUTION COUNT      (%)
------------------  -------------- -------------- --------  -------------- --------
ACCEPT                          1              1 100.0000              1   20.0000
CALL                            1              1 100.0000              1   20.0000
DISPLAY                         2              2 100.0000              2   40.0000
EXIT PROGRAM                    1              1 100.0000              1   20.0000
------------------  -------------- -------------- --------  -------------- --------
                                5              5 100.0000              5
[7]
  VERB EXECUTION COUNT          PROGRAM-NAME : MAINPGM
                                               (SUBPGM)

       [8]              [9]             [10]        [11]        [12]           [13]
                                                  PERCENTAGE                 PERCENTAGE
    VERB-ID          ACTIVE VERB     TOTAL VERB         (%)  EXECUTION COUNT      (%)
------------------  -------------- -------------- --------  -------------- --------
EXIT PERFORM                    1              1 100.0000              1    3.7037
EXIT PROGRAM                    1              1 100.0000              1    3.7037
IF                              1              1 100.0000             23   85.1852
MOVE                            1              1 100.0000              1    3.7037
PERFORM                         1              1 100.0000              1    3.7037
------------------  -------------- -------------- --------  -------------- --------
                                5              5 100.0000             27
```

〔図の説明〕

〔7〕 文別の実行回数リストヘッダ

以降の出力が文別の実行回数リストであることを示します。この情報は、プログラム単位に出力します。したがって、内部プログラムを持つプログラムでは、複数の文別の実行回数リストを出力します。PROGRAM-NAMEには、プログラム名を次の形式で表示します。

PROGRAM-NAME:プログラム名

[(呼ばれる内部プログラム名)]

〔8〕 文の名前

文の名前をアルファベット順に出力します。出力の対象となる文は、対応するソースプログラム上に記述されている文です。

〔9〕 実行された文の個数
ソースプログラム上に書かれている各文のうち、実際に実行した文の数を表示します。
〔10〕文の個数
ソースプログラム上に書かれている各文の数を表示します。
〔11〕文の実行比率
ソースプログラム上に書かれている各文の実行比率を表示します。計算式は、[9]÷[10]×100です。
〔12〕各文の実行回数
各文の実行回数を表示します。最後に実行回数の総数を表示します。
〔13〕各文の実行比率
各文の全体に対する実行回数の比率を表示します。各文の実行回数÷実行回数×100で求めます。

プログラム別実行回数リスト

```
[14]
  PROGRAM EXECUTION COUNT      PROGRAM-NAME : MAINPGM
[15]                      [16]          [17]        [18]          [19]          [20]
                                                    PERCENTAGE                PERCENTAGE
PROGRAM-NAME            ACTIVE VERB    TOTAL VERB        (%)  EXECUTION COUNT      (%)
--------------------  --------------- --------------- --------  --------------- --------
MAINPGM                           5               5  100.0000               5  15.6250
SUBPGM                            5               5  100.0000              27  84.3750
--------------------  --------------- --------------- --------  --------------- --------
                                 10              10 100.0000               32
```

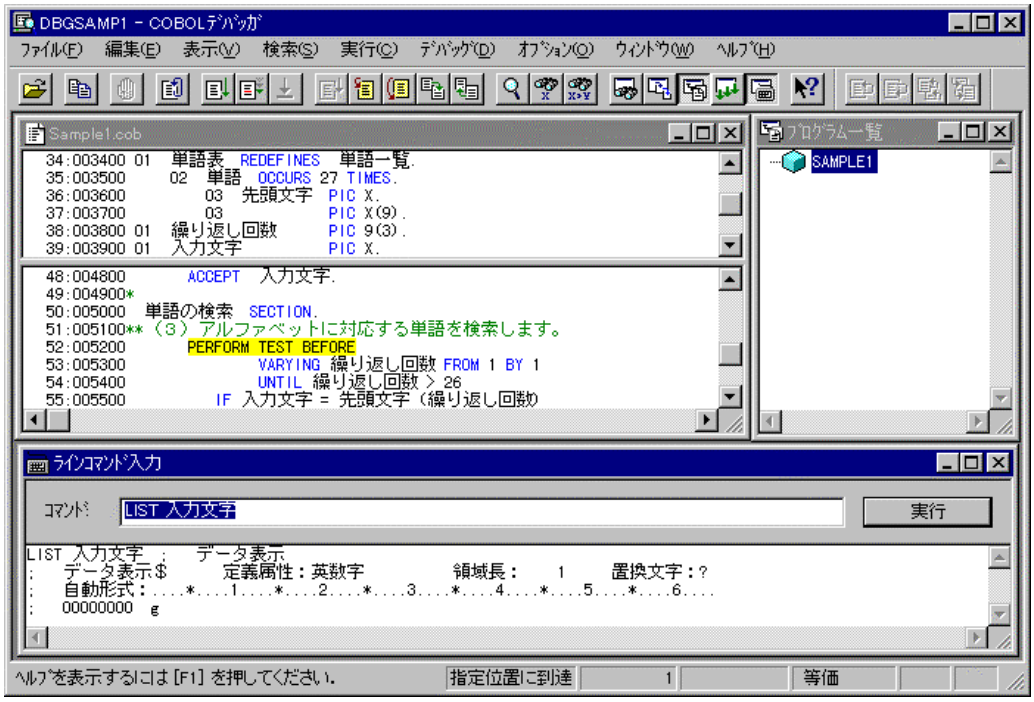
〔図の説明〕

〔14〕プログラム別実行回数リストヘッダ
以降の出力がプログラム別の実行回数リストであることを示します。このリストは、内部プログラムを持つプログラムの場合に出力します。各文の全体に対する実行回数の比率を表示します。各文の実行回数÷実行回数×100で求めます。
〔15〕プログラム名
プログラム名をソースプログラム上の出現順に出力します。
〔16〕実行した文の個数
ソースプログラム上に書かれている文のうち、実際に実行した文の数を表示します。
〔17〕文の個数
ソースプログラム上に書かれている文の数を表示します。
〔18〕文の実行比率
ソースプログラム上に書かれた文の全体に対する比率を表示します。計算式は、[16]÷[17]×100です。
〔19〕各プログラム中の文の実行回数
各プログラム中の文の実行回数を表示します。最後に合計を表示します。
〔20〕各プログラム中の文の実行比率
各プログラムの全体に対する実行文数の比率を表示します。
各プログラムの文実行回数÷全プログラムの文実行回数合計×100で求めます。

4.4 対話型デバッガによるデバッグ

NetCOBOL製品は、COBOLプログラムのデバッグに特化した対話型デバッガを含みます。
ここでは、その概要を述べ、OSIV系プログラムの分散開発に役立つ機能を中心に説明します。それ以外の対話型デバッガの詳細な機能や操作方法については、“NetCOBOL 使用手引書”またはデバッガ自身のヘルプを参照してください。

図4-23 対話型デバッガによるデバッグ例



4.4.1 対話型デバッガの特徴

NetCOBOLの提供する対話型デバッガは次のような特徴を持っています。

各種情報をリアルタイムに表示しながら、対話的にデバッグ可能

デバッグ対象プログラムのCOBOLソースプログラムを画面に表示させ、これをキーボード・マウスで直接操作する形で、デバッガとして基本的な機能である、次のような操作が可能です。

- プログラムの各種実行（ステップイン／ステップオーバ／ステップアウト／…）
- 中断点の設定／解除
- データの表示／変更／監視

また、必要に応じて、次のような情報を表示させ、操作することも可能です。

- 監視中のデータ項目の情報
- プログラムの呼び出し経路
- プログラム一覧

データ変更での中断

データの内容が変更されたときにプログラムの実行を中断し、デバッグ操作を可能にします。
実行監視条件での中断指定した条件式が成立したときにプログラムの実行を中断し、デバッグ操作を可能にします。

特定実行条件での中断

動詞の種類やファイルを指定して、該当する文に到達したときにプログラムの実行を中断、そこからのデバッグ操作を可能にします。

バッチデバッグやデバッグ操作の再現

ファイルからコマンドを入力することにより、バッチデバッグを行うことができます。
また、操作の履歴をファイルに保存して、これを入力として使用することにより、デバッグ操作を再現させることができます。

連絡節データ獲得

呼び出されたプログラムで、呼び出し元のプログラムから渡されたパラメタとは別に、連絡節で定義されたデータの領域を確保することができます。

分散開発のためにWindows系システム上で実行不可能なプログラムを疑似的に実行

次の機能は、実行にあたってOSIV固有のサブシステムを必要とします。

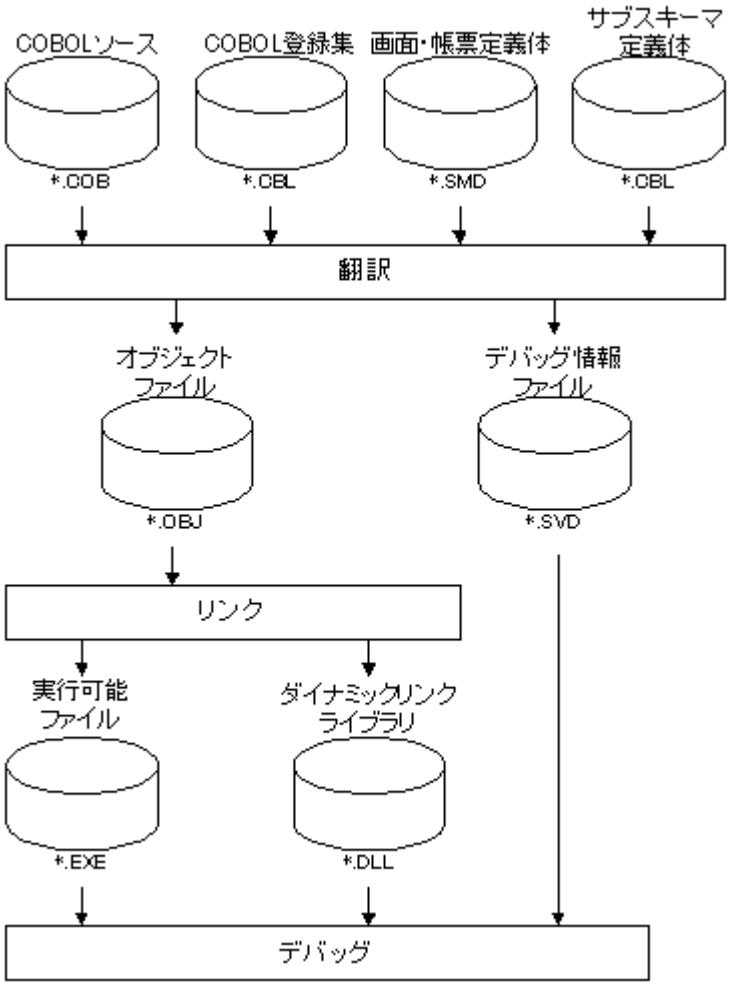
- AIM表示ファイル機能（宛先ACM/APL/TRM/CMD/WST）
- ネットワークデータベース機能

このため、これらの機能を使用するプログラムは、単独ではWindows系システム上で動作させることはできません。しかし、対話型デバッガ上では、これらの機能を含むプログラムを疑似的に実行することができます。

4.4.2 対話型デバッグのための準備

対話型デバッガは、実行可能プログラム(EXE)および実行可能プログラムから呼び出されるダイナミックリンクライブラリ(DLL)をデバッグすることができます。これらの実行可能プログラムおよびダイナミックリンクライブラリをデバッグ対象プログラムと呼びます。
デバッグ対象プログラムは、通常、複数個のCOBOLソースプログラムで構成されます。このうち、デバッガの機能を利用してデバッグの操作が行えるCOBOLソースプログラムを被デバッグプログラムと呼びます。
デバッガに、被デバッグプログラムと認識させるには、COBOLコンパイラにより作成されるデバッグ情報ファイルが必要です。

図4-24 対話型デバッグに必要となる資源



以降、OSIV系プログラムにおけるデバッグ対象プログラムの準備と、そのデバッグを開始するまでの手順を説明します。

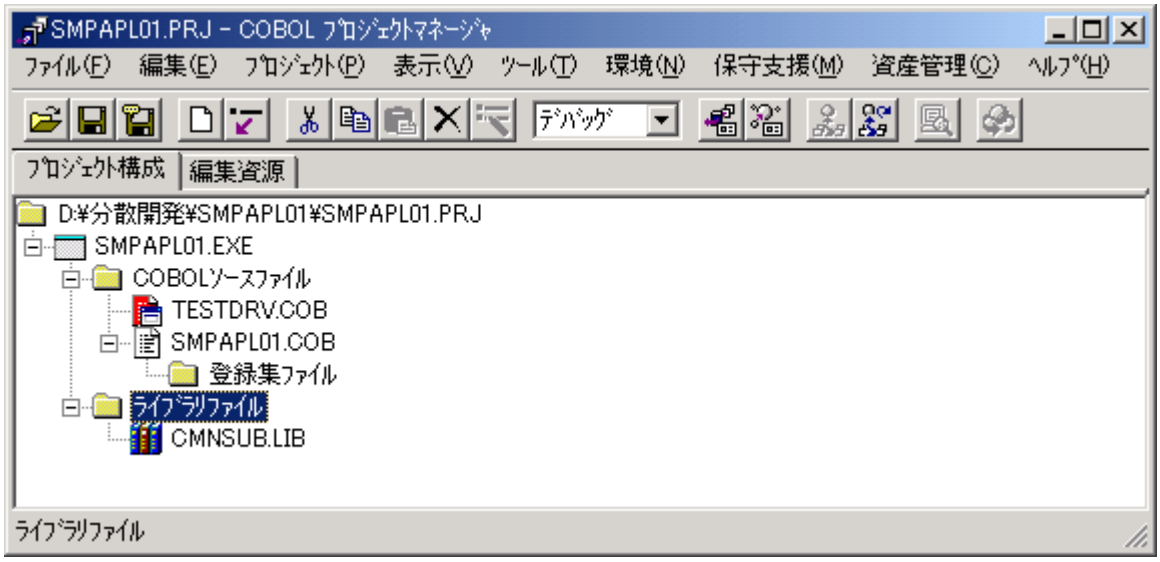
4.4.2.1 デバッグ対象プログラム／被デバッグプログラムの準備

デバッグ対象プログラムが実行可能ファイルであっても、ダイナミックリンクライブラリであっても、その中に含まれるプログラムが次の条件を満たせば、被デバッグプログラムとなります。

- 翻訳時:翻訳オプションTESTを指定して翻訳した。
- リンク時:リンクオプション“/DEBUG /DEBUGTYPE:COFF”を指定して、リンクした。

第3章で説明したNetCOBOLのプロジェクト管理機能の使用手順(“3.2.1 基本的なプロジェクトファイルの作成”を参照)に従って、プロジェクトを作成し、〔プロジェクト〕－〔オプション〕メニューから “デバッグモジュール作成”を選択した状態で最終ターゲットファイルのビルドが行われたなら、この条件は自然に満たされます。

図4-25 “デバッグモジュール作成”が選択されているプロジェクト

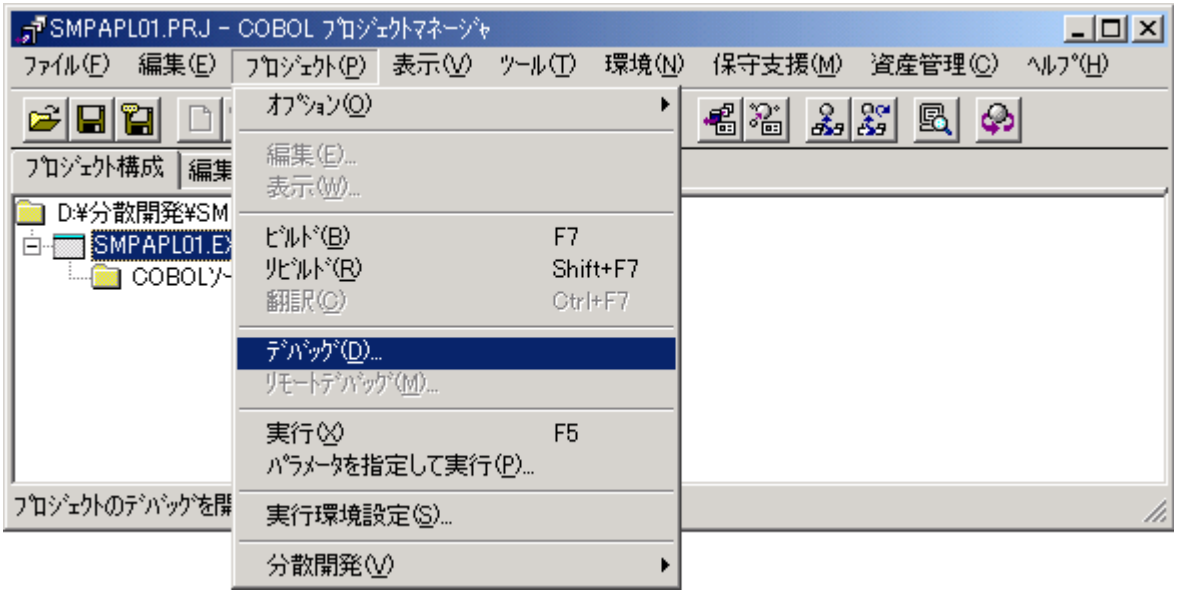


4.4.2.2 デバッグの開始

以下、対話型デバッガによるデバッグの開始手順を説明します。

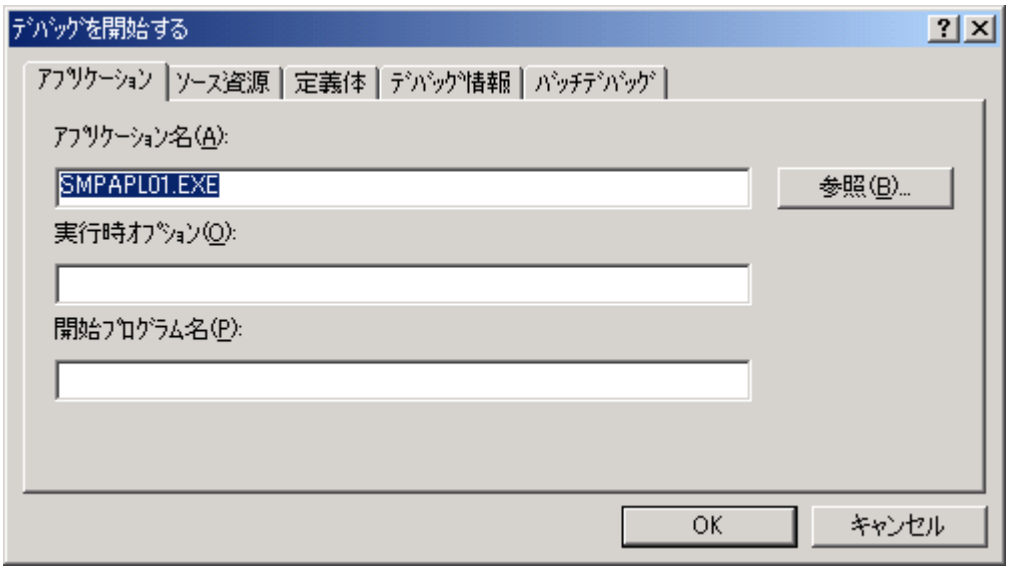
1. COBOLプロジェクトマネージャを起動します。プロジェクトの最終ターゲットファイルを選択します。
2. 〔プロジェクト〕メニューの“デバッグ”を選択します。

図4-26 COBOLプロジェクトマネージャからデバッグを開始する



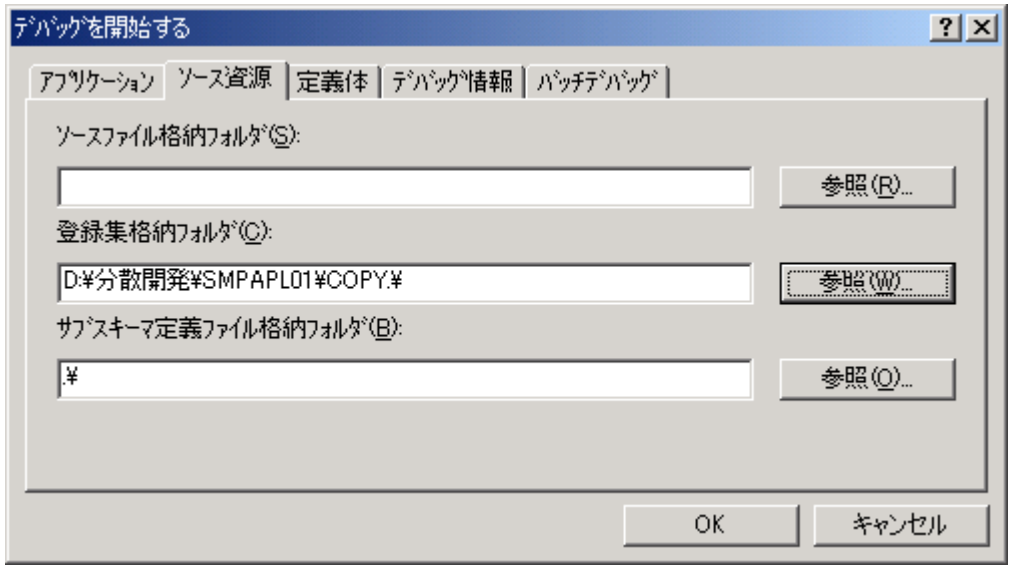
3. 対話型デバッガが起動して、“デバッグを開始する”ダイアログが表示されます。
4. プロジェクトの最終ターゲットが実行可能ファイルである場合、〔デバッグを開始する〕ダイアログの〔アプリケーション名〕に実行可能ファイル名が指定されています。
   プロジェクトの最終ターゲットがダイナミックリンクライブラリである場合、〔アプリケーション名〕は空白になっているため、テスト対象プログラム(ダイナミックリンクライブラリ)を呼び出す実行可能ファイル(テストドライバなど)を指定します。

図4-27 〔デバッグを開始する〕ダイアログ



5. 〔アプリケーション名〕に指定した実行可能ファイル名に含まれる最初に実行されるプログラムが被デバッグプログラムでない場合(テストドライバなど)、〔開始プログラム名〕に被デバッグプログラムの名前を指定します。
6. この状態でデバッグに必要な資源は、すべてデバッグ対象プログラムと同じフォルダにあるものと見なされています。他のフォルダに格納されて資源(例えば、COBOL登録集)があるなら、正しい格納フォルダを指定してください。

図4-28 〔デバッグを開始する〕ダイアログ



7. 〔OK〕ボタンをクリックするとデバッグが開始されます。

4.4.3 分散開発のための対話型デバッガの機能

ここでは、OSIV系プログラムの分散開発のために、デバッガで提供している機能を説明します。

4.4.3.1 ネットワークデータベース操作文

ネットワークデータベース(NDB)機能はグローバルサーバ固有の機能です。この機能を使用する

プログラムを実行するためにはネットワークデータベースシステム（グローバルサーバではAIM/DB）が必要ですが、Windows系システムにはこれを代替するシステムが存在しません。このため、ネットワークデータベース(NDB)機能を使用するプログラムをWindows系システム上で単独に実行することはできません。実行した場合、次のようなエラーが発生します。

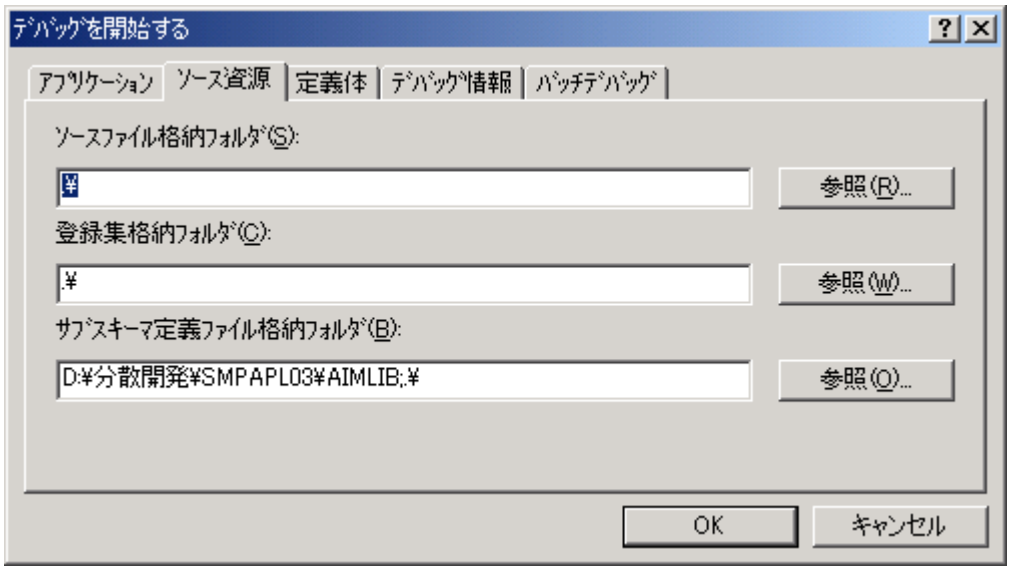
図4-29 ネットワークデータベース機能を使用するプログラムを実行時のエラー

ネットワークデータベース機能を使用するプログラムのデバッグの手順

対話型デバッガ上では、ネットワークデータベース(NDB)機能を使用するプログラムを疑似的に実行し、デバッグすることが可能です。以下、その詳しい手順について、説明します。

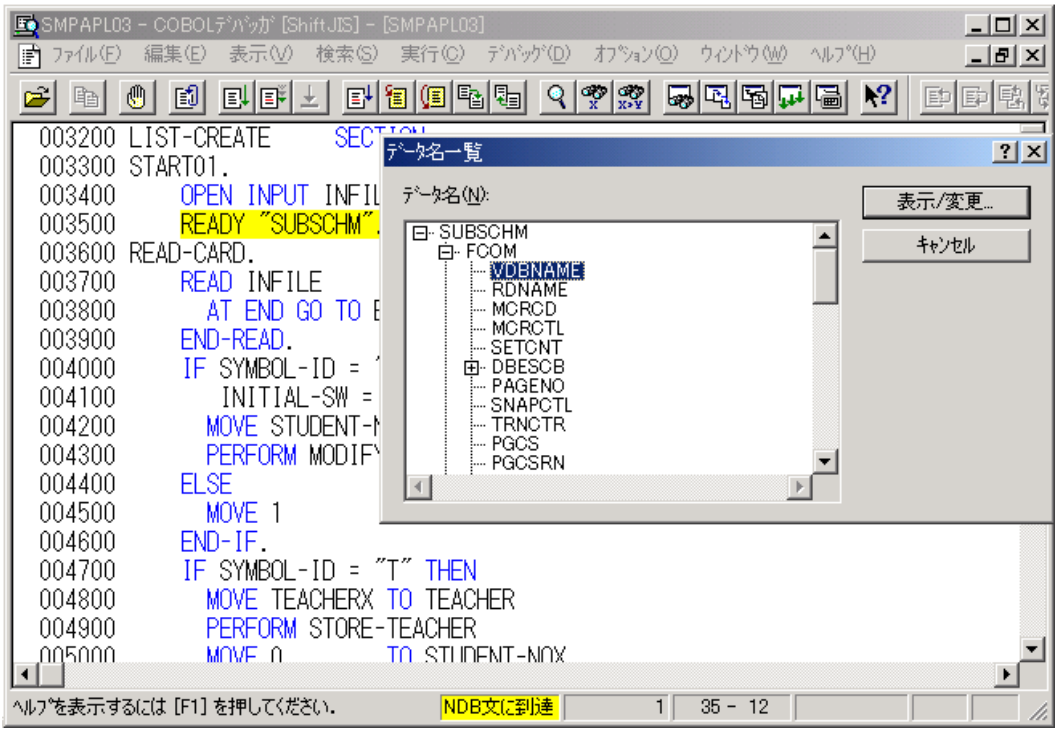
1. ネットワークデータベース機能を使用する被デバッグプログラムを含むプロジェクトを開き、デバッガを起動します。
2. 〔デバッグを開始する〕ダイアログで〔サブスキーマ定義ファイル格納フォルダ〕の設定に、翻訳オプションAIMLIBで指定したフォルダが含まれていることを確認します。含まれていない場合、〔参照〕ボタンを押して、翻訳オプションAIMLIBで指定したフォルダを選択し、追加します。

図4-30 〔サブスキーマ定義ファイル格納フォルダ〕の設定の確認



3. 〔OK〕ボタンを押して、デバッグを実行します。
4. プログラムを実行すると、ネットワークデータベース機能を使用した文を実行するときに、デバッガは自動的にプログラムを中断し、ネットワークデータベース機能に関連したデータ名を選択する〔データ名一覧〕ダイアログを表示します。ここで、データ名を選択し、〔表示/変更〕ボタンをクリックすると、〔データの表示/変更〕ダイアログが開き、データの内容の確認または変更を行うことができます。

図4-31 対話型デバッガ上でのネットワークデータベースを操作する文の疑似実行



ネットワークデータベース機能を使用するプログラムをデバッグする際の注意事項

ここでは、分散開発支援機能使用時の注意事項について説明します。

READY 文について

- READY文の記述がなく、サブスキーマ名段落にEXTERNAL指定もないプログラムでは、FCOM/UWAは項目名の定義だけであり、領域は確保されません。そのため、このようなプログラムをそのままの状態(領域未割当て)で実行した場合は、その動作は保証されません。このようなプログラムを実行する場合は、最初にFCOM/UWAの各01レベルの項目に対して、デバッガの機能である“連絡節獲得”を行います。“連絡節獲得”を行うと、デバッガがFCOM/UWAの領域を確保し、READY文を記述したときと同様に実行することができます。

USE 文の実行

- USE手続きは、アプリケーションから呼ばれるネットワークデータベース処理システム側で異常が発生した場合に制御を分岐させるためのものです。対話型デバッガ上での疑似実行時はネットワークデータベース機能を処理するシステムが存在しないため、実行時にUSE手続きへ分岐することはできません。
- USE手続きを実行したい場合には、中断点の設定により実行を中断し、デバッガの“実行開始位置をカーソル位置へ変更”によってUSE手続きの最初の命令へ実行開始点を移動させ、実行を再開します。

4.4.3.2 宛先DSP、PRT以外の表示ファイルの入出力文

表示ファイル機能の実行には、COBOLだけではなく各宛先種別に応じた表示ファイル機能を実現する他のシステムが必要です。宛先が“DSP”および“PRT”の場合、Windows系システムではMeFtがグローバルサーバのPSAMの機能を代替します(ただし、完全に同じ動作は期待できません)。その他の宛先の場合は、次のような状況です。

- MeFt以外のサブシステムで実現する表示ファイル機能
  宛先“ACM”および“APL”が該当します。代替機能を提供するシステムはWindows系システムには存在しますが、OSIV系プログラムの分散開発には使えません。

注意

宛先APLの表示ファイルを以下の手順で単体テストする場合、実行時オプション@CBR_PSFILE_APLの指定が必要です。詳細については“4.2.3.2実行時オプション”を参照してください。

- OSIVシステム固有の表示ファイル機能
  宛先“TRM”，“WST”および“CMD”が該当します。代替機能を提供するシステムはWindows系システムには存在しません。

このため、上記の宛先の表示ファイル機能を使用するプログラムをWindows系システム上で単独に実行することはできません。実行した場合、次のようなエラーが発生します。

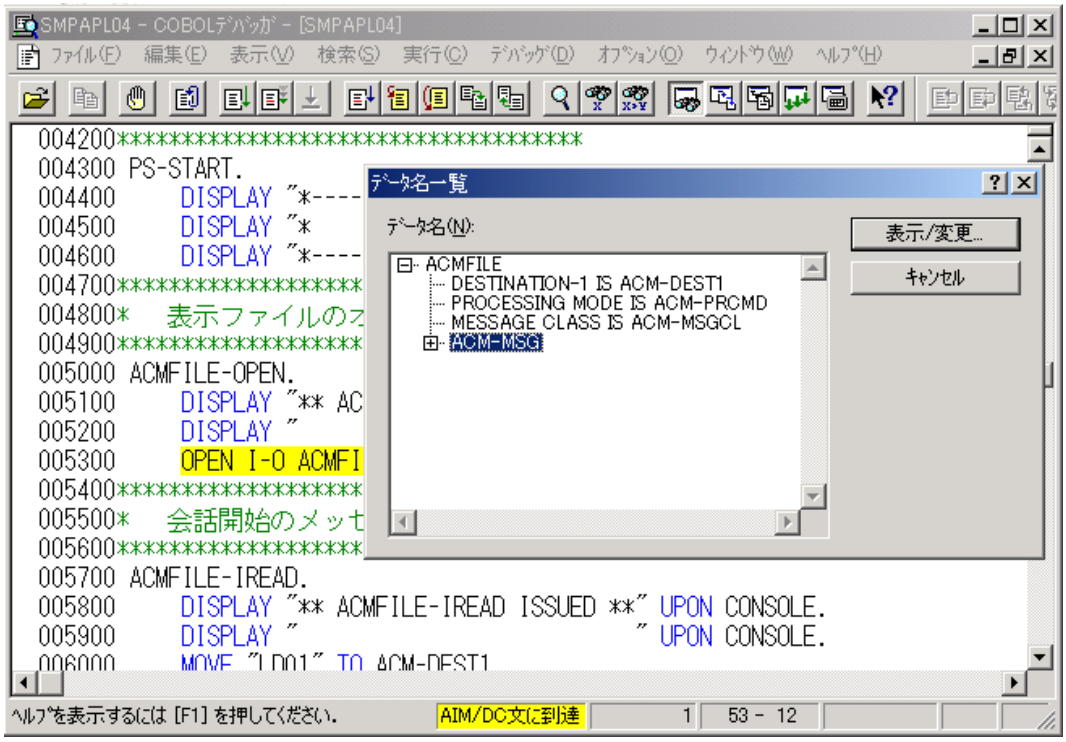
図4-32 宛先DSP、PRT以外の表示ファイル機能を使用するプログラムを実行時のエラー

宛先DSP、PRT以外の表示ファイル機能を使用するプログラムのデバッグの手順

対話型デバッガ上では、宛先DSP、PRT以外の表示ファイル機能を使用するプログラムを疑似的に実行し、デバッグすることが可能です。以下、その詳しい手順について、説明します。

1. 実行環境情報の初期化ファイルで、宛先DSP、PRT以外の表示ファイルにダミーのファイル(実際は存在しないファイル)を割り当てます。
2. 宛先DSP、PRT以外の表示ファイル機能を使用する被デバッグプログラムを含むプロジェクトを開き、通常の手順どおりにデバッグを開始します。
3. 〔OK〕ボタンを押して、デバッグを実行します。
4. プログラムを実行すると、宛先DSP、PRT以外の表示ファイルの入出力文を実行するときに、デバッガは自動的にプログラムを中断し、表示ファイル機能に関連したデータ名を選択する〔データ名一覧〕ダイアログを表示します。ここで、データ名を選択し、〔表示/変更〕ボタンをクリックすると、〔データの表示/変更〕ダイアログが開き、データの内容の確認または変更を行うことができます。

図4-33 対話型デバッガ上での宛先DSP、PRT以外の表示ファイルの入出力文の疑似実行



宛先DSP、PRT以外の表示ファイル機能を使用するプログラムをデバッグする際の注意事項

ここでは、分散開発支援機能使用時の注意事項について説明します。

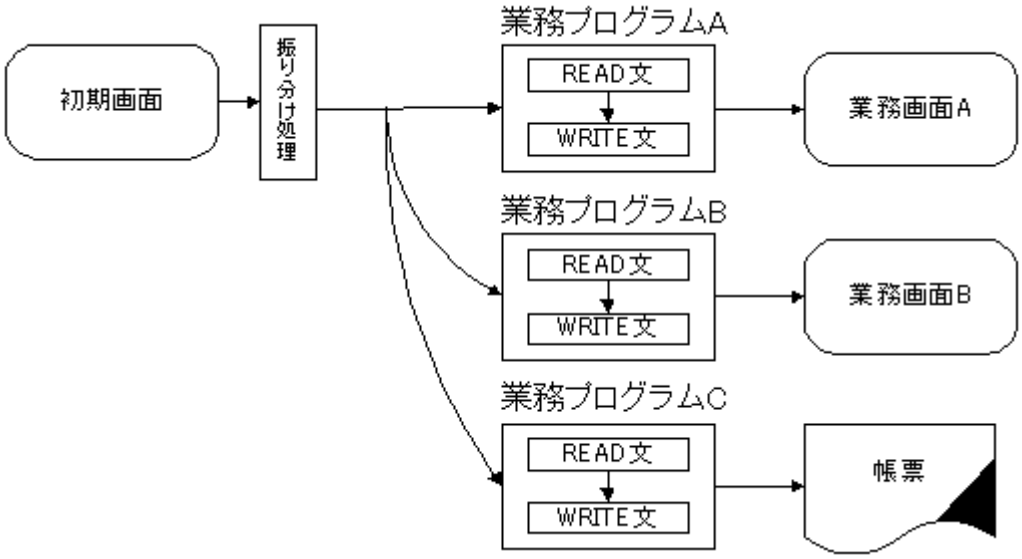
USE 文の実行

- USE手続きは、アプリケーションから呼ばれる表示ファイルの処理システム側で異常が発生した場合に制御を分岐させるためのものです。対話型デバッガ上での疑似実行時は宛先DSP、PRT以外の表示ファイル機能を処理するシステムが存在しないため、実行時にUSE手続きへ分岐することはできません。
- USE手続きを実行したい場合には、中断点の設定により実行を中断し、デバッガの“実行開始位置をカーソル位置へ変更”によってUSE手続きの最初の命令へ実行開始点を移動させ、実行を再開します。

4.4.3.3 単体テスト支援機能

グローバルサーバで表示ファイルを使用して対話処理を行うプログラムに、AP/EFの画面振り分け機能を用いて次のような形態で動作するものがあります(詳細については、“OSIV AP/EF使用手引書 業務プログラム編”を参照してください)。

図4-34 表示ファイル機能を使用して対話処理をするプログラム

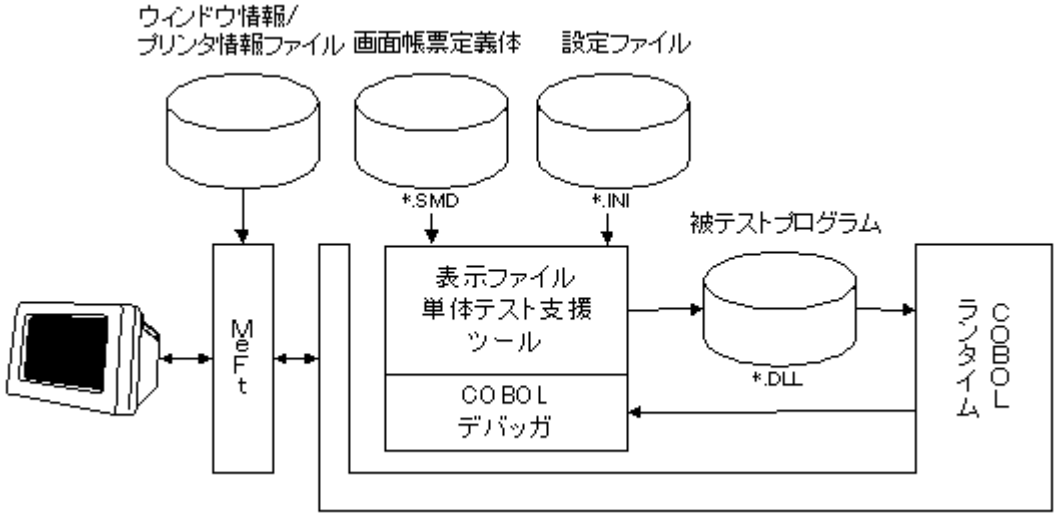


表示ファイル単体テスト支援ツール(COBPRTST.EXE)は、このようなOSIV系プログラムを単体テストするために用意されています。

表示ファイル単体テスト支援ツールの概要

以下に、表示ファイル単体テスト支援ツールを使用した単体テストの概要を示します。

図4-35 表示ファイル単体テスト支援ツールを使用したテストの概要



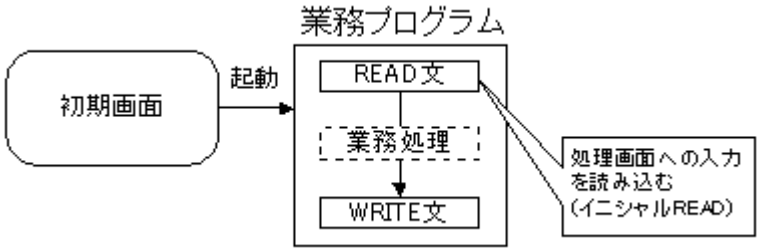
対話型デバッガは被デバッグプログラムが宛先DSPまたはPRT以外の表示ファイル機能を使用している場合は必須です。

初期画面の画面定義体の準備

表示ファイル単体テスト支援ツールが対象とする被デバッグプログラムは、次のような特徴を持つため、初期画面用の画面定義体が必要です。

- 初期画面から振り分け処理によって起動される。
- 初期画面に入力された項目の値を読み込んで動作を開始する。

図4-36 AP/EFの画面振り分け機能により起動されるプログラム



初期画面用の画面定義体を準備する方法としては、次の2つのどちらであってもかまいません。

- グローバルサーバ上に存在する初期画面用のフォーマット定義体をWindows系システム用の画面帳票定義体に変換し、移行する(詳細は“3.3.2 フォーマット定義体の移行”を参照してください)。
- Windows系システム上でFORMを使用して、新規に初期画面用の画面定義体を作成し、その定義体に振り分け手順を設定する。

また、この画面帳票定義体用にウィンドウ情報ファイルを用意する必要があります。なお、ウィンドウ情報ファイルについては“4.2.3.1 プログラムが必要とする資源の割り当て”を参照してください。

被デバッグプログラムの準備

このツールを使用して単体テストを行う場合でも、翻訳・リンクなどの手順に大きな違いはありません。ただし、被デバッグプログラムはダイナミックリンクライブラリ(DLLファイル)として作成しておく必要があります。

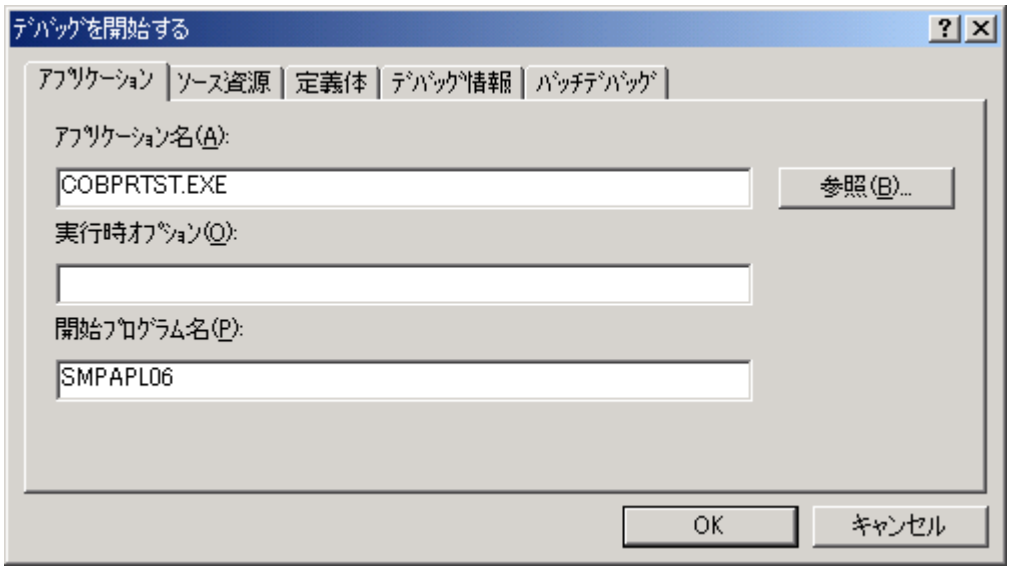
また、被デバッグプログラムの実行に必要な実行環境情報を格納したファイル(COBOL85.CBR)を被デバッグプログラム(DLLファイル)のあるフォルダに用意しておく必要があります。なお、実行環境情報の詳細については、“4.2.3 COBOL実行環境の設定”を参照してください。

表示ファイル単体テスト支援ツールの起動方法

以下の手順で表示ファイル単体テストツールを起動します。

1. 対話型デバッガを起動し、〔ファイル〕メニューの“デバッグを開始する”を選択します。
2. 〔デバッグを開始する〕ダイアログの〔アプリケーション名〕に“COBPRTST.exe”と、〔開始プログラム名〕に被デバッグプログラムの名前を入力します。
3. 〔デバッグを開始する〕ダイアログの〔OK〕ボタンをクリックすると、ツールが起動して、〔表示ファイル単体テスト〕ダイアログが表示されます。

図4-37 対話型デバッガからの表示ファイル単体テスト支援ツールの起動



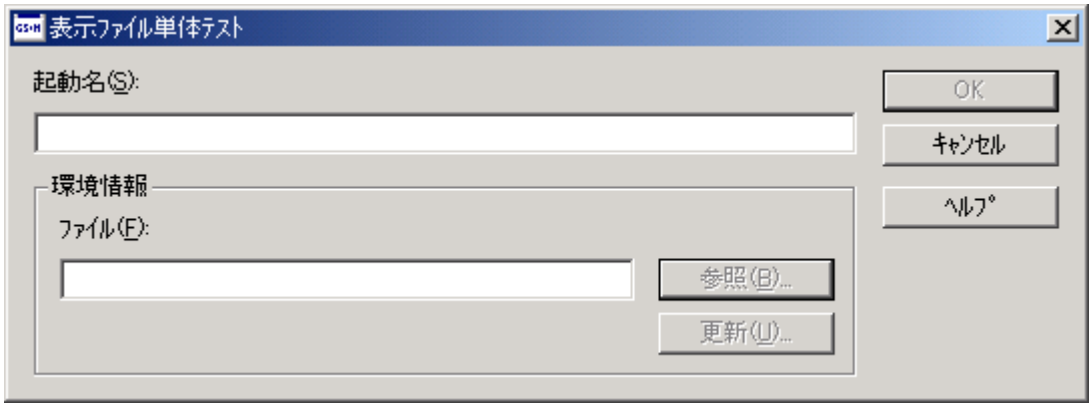
注意

被デバッグプログラムが、宛先DSPまたはPRT以外の宛先を使用していない場合、対話型デバッガからではなく、直接、ツール（NetCOBOLインストールフォルダのCOBPRTST.exe）を起動して、単体テストを実施することもできます。

表示ファイル単体テストツールの使用法

表示ファイル単体テストツールを起動すると、以下のダイアログが現れます。

図4-38 〔表示ファイル単体テスト〕ダイアログ



以下、このツールを使用して、単体テストを行う手順について説明します。

1. 〔起動名〕に初期画面の画面定義体の名前を指定します。
2. 〔環境情報〕に環境ファイルの名前を指定し、〔更新〕ボタンをクリックします。指定したファイルが存在しないなら、新規に環境ファイルが作成されます。
3. 〔実行環境変更〕ダイアログが表示されるので、ツールの実行環境情報を設定します。
   設定の方法は、〔オプション一覧〕から変更するオプションを選択し、〔参照〕ボタンでファイル名を選択・指定するか、〔オプション内容〕に変更内容を指定して〔設定〕ボタンをクリックします。
   各オプションは次の意味を持ちます。
   — CBRNAME（COBOLの実行用の初期化ファイル名）:

被デバッグプログラム用の実行用の初期化ファイル名(COBOL85.CBR)を指定します。

— ICONNAME（アイコンDLL名):
表示ファイル単体テスト機能のアイコンを変更する場合に指定します。アイコン用DLLモジュールのファイル名をフルパスで指定してください。

— ICONID（アイコンリソースID):
アイコン用DLLモジュールに格納されているアイコンリソースの識別用のIDを数値で指定してください。省略値は、“1”となります。

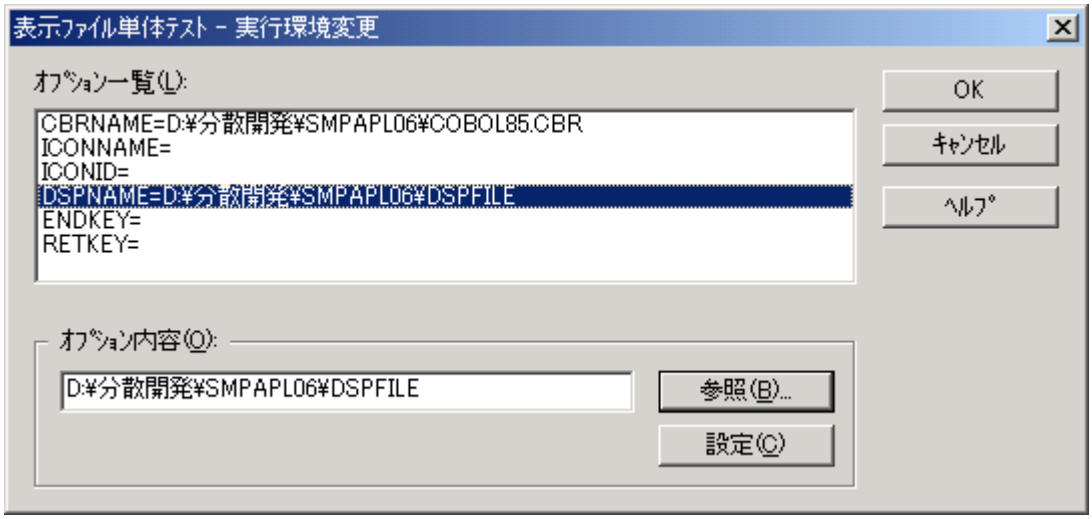
— DSPNAME（ウィンドウ情報ファイル名):
画面定義体の環境情報を変更したい場合は、ウィンドウ情報ファイル名を指定してください。

— ENDKEY（ENDキー値):
ENDキー値を変更したい場合は、画面定義体のアテンション定義で設定した項目リテラル値を設定してください。省略値は、“F002”(F2キー)となります。

— RETKEY（RETURNキー値):
RETURNキー値を変更したい場合は、画面定義体のアテンション定義で設定した項目リテラル値を設定してください。省略値は、“F003”(F3キー)となります。

“CBRNAME”と“DSPNAME”は必ず指定してください。

図4-39 〔実行環境変更〕ダイアログでの設定例



4. 〔実行環境変更〕ダイアログの〔OK〕ボタンをクリックすると変更の内容が環境ファイルに保存され、〔表示ファイル単体テスト〕のダイアログに戻ります。
5. 〔表示ファイル単体テスト〕のダイアログの〔OK〕ボタンをクリックすると、初期画面が表示されて、単体テストが開始されます。

4.4.4 分散開発時に有効な対話型デバッガの機能

ここでは、対話型デバッガの機能の標準的な機能の中から、OSIVプログラムの分散開発に特に有効な機能について、説明します。
その他の対話型デバッガの一般的な機能については、“NetCOBOL使用手引書”および“COBOLデバッガ ヘルプ”を参照してください。

4.4.4.1 連絡節獲得

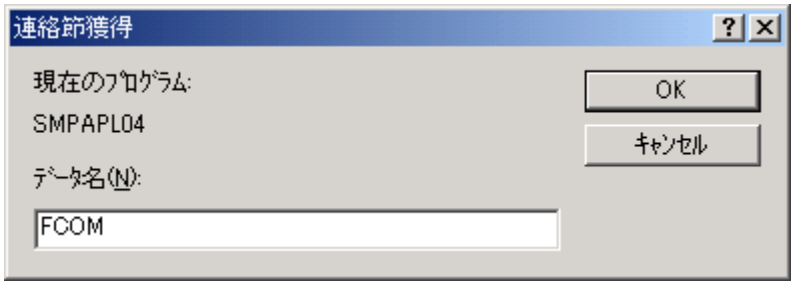
対話型デバッガでは、副プログラムのデバッグ時に、呼出し元のプログラムから渡される連絡節のデータ領域を獲得できます。
ネットワークデータベース機能を使用するプログラムのデバッグ時、次の条件に合致する場合はFCOM/UWAの領域は未割当ての状態にあるため、プログラムの動作は保証されません。

- READY文の記述がなく、かつ、サブスキーマ名段落にEXTERNAL指定がない。

そのような場合、この機能を使用し領域を確保することで、正しくプログラムを動作させることができます。
以下、その手順について説明します。

1. プログラムのデバッグを開始し、被デバッグプログラムの入口で停止します。
2. 〔デバッグ〕メニューから“連絡節獲得”を選択すると、〔連絡節獲得〕ダイアログが表示されます。
3. 〔データ名〕に獲得する領域のデータ名を入力し、〔OK〕ボタンをクリックします。

図4-40 連絡節獲得機能によるFCOMの獲得



この機能によって、獲得した領域は、被デバッグプログラムからその呼び出し元のプログラムに制御が戻るときにデバッガによって解放されます。

呼出し元のプログラムと呼び出されたプログラムのパラメタの個数や形式が異なるような場合であっても、この機能を使用することによって呼出し元のプログラムの修正や再翻訳なしに呼び出されたプログラムのデバッグが可能です。
OSIV系プログラムの分散開発では、しばしば開発対象のプログラム毎にテスト用のドライバプログラムを必要とします。しかし、この機能をうまく利用することによって、異なる属性や数のパラメタを持つ複数のプログラムを１つのドライバプログラムでテストするなどという使い方も考えられます。

4.4.4.2 実行開始位置の変更

対話型デバッガを使用することで、Windows系システムでは使用できない次のような機能を含むプログラムを擬似的に実行し、デバッグすることができます。

- ネットワークデータベース機能
- 宛先DSP、PRT以外の表示ファイル機能

しかし、このようなデバッガ上の擬似的な実行では、実行時にUSE手続きへ分岐することはありません。USE手続きは、被デバッグプログラムから呼ばれるネットワークデータベース処理システムや表示ファイル処理システムで異常が発生した場合に制御を分岐させるためのものですが、対話型デバッガ上での疑似実行時はそれらが存在しないためです。
このような場合、USE手続きをデバッグするためにデバッガの“実行開始位置の変更”機能を使用します。
以下、その手順を説明します。

1. 実行開始位置を変更してデバッグする手続き（ここではUSE手続き）の直前に実行される

文でプログラムの実行を一時中断します。

2. 新たな実行の開始位置とする文（USE手続きの場合、USE文の次の文）にカーソルをあわせます。
3. 〔実行〕メニューから“実行開始位置をカーソル位置に変更”を選択すると、実行開始位置が変更され、“現在中断している文”がカーソルを位置づけた文に変わります。後は、通常の操作で以降の手続きのデバッグ作業が行えます。

通常、USE手続きの実行後、入出力状態が重大誤りでなければ、USE手続き実行の契機となった誤りの発生した文の次に位置する実行文に制御が移ります。しかし、デバッガで実行開始位置を変更した場合、このようなプログラムの制御の移動は起こりません。再び、実行を再開したい文にカーソルを合わせ、“実行開始位置をカーソル位置に変更”することが必要です。

4.4.4.3 デバッグコマンドによるデバッグの自動化

通常、対話型デバッガによるデバッグは、表示された被デバッグプログラムのCOBOLソースの表示を見ながら、キーボード・マウスでメニューやツールバー上のデバッグ操作を選択して行います。このようなGUIを使用した操作は、きめの細かい作業を可能とする反面、同じ作業を繰り返し行わなければならないような場合、効率的とはいえません。
同じ作業を繰り返し行う必要があるような場合、デバッグコマンドとコマンドファイルを用いたバッチデバッグが有効です。

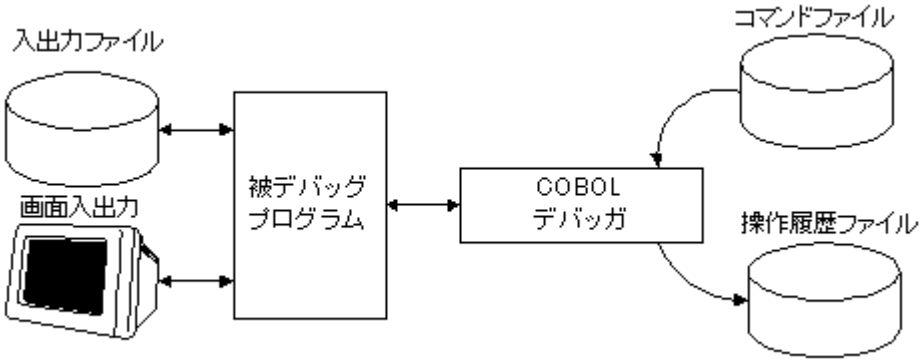
デバッグコマンドとバッチデバッグ

デバッグコマンドは、デバッガに対する操作をコマンド形式で指定するものです。通常はラインコマンド入力ウィンドウで使用します。

図4-41 ラインコマンド入力ウィンドウでのデバッグコマンドの使用例

しかし、デバッグ操作を再現したり、同一のデバッグ処理手順を頻繁に行うときには、一連のデバッグコマンドを格納したコマンドファイルを使ってデバッグ作業を自動化することができます。以下、バッチデバッグについての概要を示します。

図4-42 バッチデバッグの概要

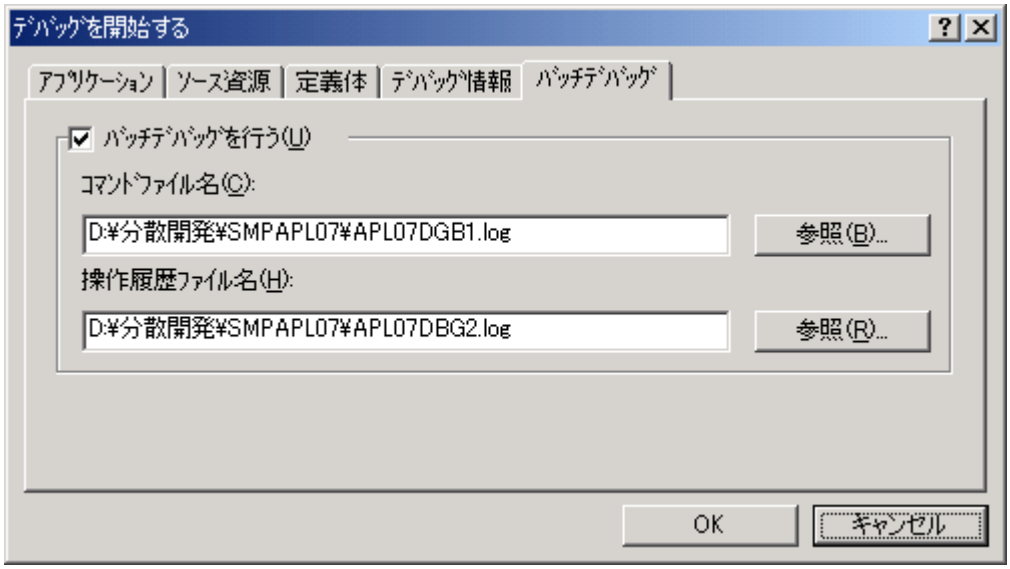


バッチデバッグの手順

以下、バッチデバッグの手順を説明します。

1. 対話型デバッガを起動し、通常の操作でデバッグを行う場合と同様に、〔アプリケーション名〕その他を〔デバッグを開始する〕ダイアログに指定します。
2. このダイアログの〔バッチデバッグ〕ページを選択し、以下のように指定します。
   - ― バッチデバッグを行う:
     チェックします。
   - ― コマンドファイル名:
     一連のデバッグ作業に対するデバッグコマンドを格納したファイルを指定します。
   - ― 操作履歴ファイル名:
     デバッグの結果の出力先を指定します。

図4-43 〔バッチデバッグ〕ページの入力例



3. 〔OK〕ボタンをクリックするとデバッグが開始されます。COBOLコンソールに対するACCEPT文などの実行で入力待ちとなるような場合を除けば、プログラムが最後まで自動的に実行されます。
4. プログラムの実行結果は、〔操作履歴ファイル名〕として指定したファイルに保存されます。以下、履歴ファイルへの出力例を示します。

図4-44 履歴ファイルへの出力例

```
ENV DBTR(C:\DEBUGDEMO\SAMPLE\DEBUGDEMO1.log)  ;   環境変更
```

```
;   コマンドファイル内のDBTR/NODBTRは無視されます.
;   環境変更$
LIST (入力日付) IN(INTEGER-OF-DATE) FORMAT(A)  ;   データ表示
;   データ表示$       定義属性:集団            領域長:    8      置換文字:?
;   自動形式:....*....1....*....2....*....3....*....4....*....5....*....6....
;   00000000  19991230
BREAK 34 IN(INTEGER-OF-DATE)  ;   中断点設定
;   中断点設定$                                                  34, 1
;   中断点位置:  INTEGER-OF-DATE
;   条件式:なし
;   中断点通過回数指定:  なし
BREAK 40 IN(INTEGER-OF-DATE)  ;   中断点設定
;   中断点設定$                                                  40, 1
;   中断点位置:  INTEGER-OF-DATE
;   条件式:なし
;   中断点通過回数指定:  なし
BREAK 43 IN(INTEGER-OF-DATE)  ;   中断点設定
;   中断点設定$                                                  43, 1
;   中断点位置:  INTEGER-OF-DATE
;   条件式:なし
;   中断点通過回数指定:  なし
BREAK 48 IN(INTEGER-OF-DATE)  ;   中断点設定
;   中断点設定$                                                  48, 1
;   中断点位置:  INTEGER-OF-DATE
;   条件式:なし
;   中断点通過回数指定:  なし
  …
```

コマンドファイルの作成

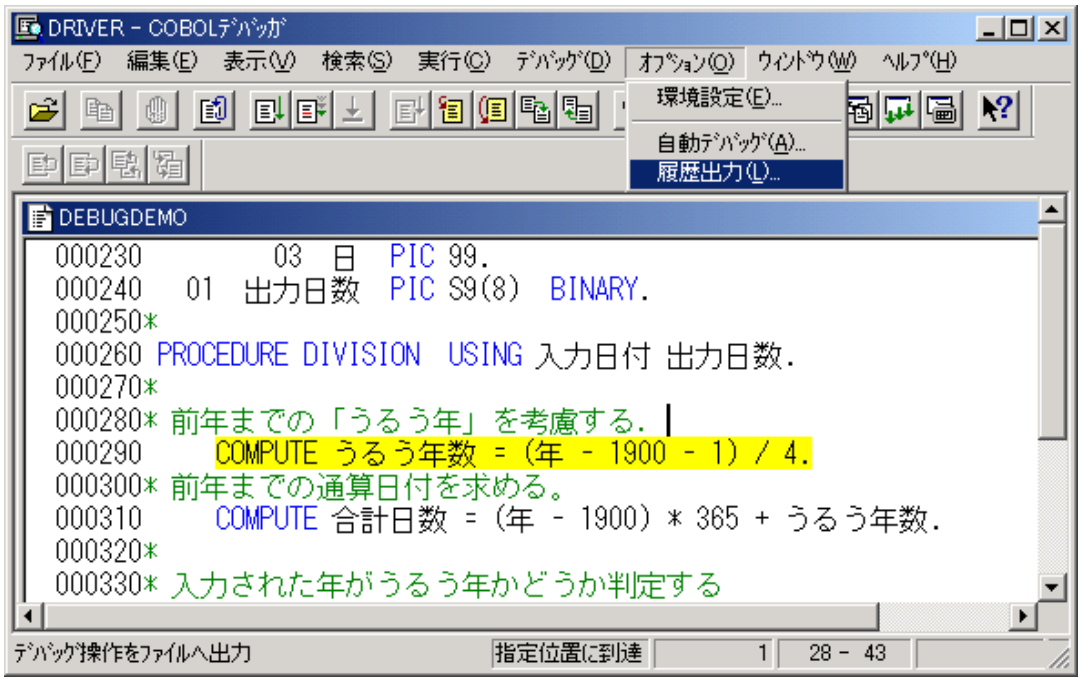
コマンドファイルはデバッグ操作の順にデバッグコマンドを記述したファイルなので、テキストエディタを使用して作成することができます。

しかし、最初からテキストエディタでコマンドファイルを作成するよりも、自動化したいデバッグ手続きの操作履歴をファイル出力したものをそのまま、あるいは修正して使用するほうが容易です。

以下、操作履歴をファイルに出力する手順を説明します。

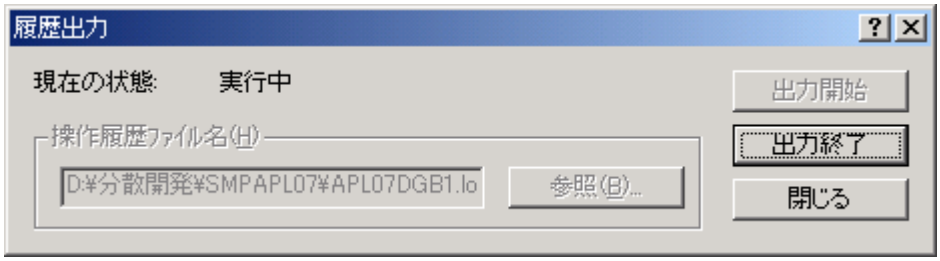
1. 通常の操作でデバッグを開始し、被デバッグプログラムの先頭に停止した所で、〔オプション〕メニューの“履歴出力”を選択します。

図4-45 履歴出力の選択



2. 〔履歴出力〕ダイアログが現れるので、履歴を出力するファイル名を指定し、〔出力開始〕ボタンをクリックします。

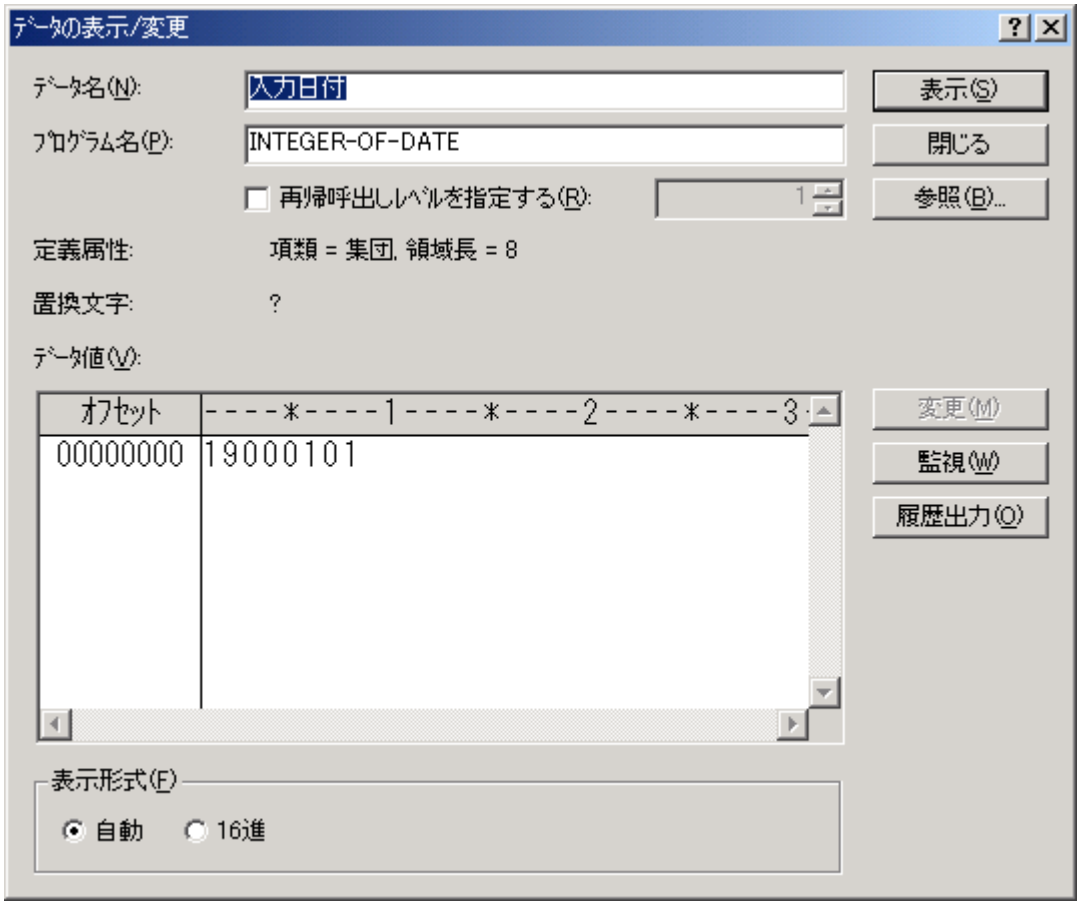
図4-46 〔履歴出力〕ダイアログ(履歴出力中)



3. 〔閉じる〕ボタンをクリックして〔履歴出力〕ダイアログを閉じ、対話的にデバッグを実施します。
4. デバッグを終了すると、それまでに実施したデバッグの手順が〔操作履歴ファイル名〕に指定したファイルに記録されます。

データの内容の表示操作は明示的にその操作を履歴に出力することを指定する必要があります。履歴を出力するには〔データの内容の表示／変更〕ダイアログの履歴出力のボタンを押して下さい。

図4-47 〔データの内容の表示／変更〕ダイアログ



デバッグコマンドにより可能な操作

対話型デバッガはデバッグコマンドにより、次のような操作が可能です。

表4-7 デバッガコマンドの一覧

コマンドの種別	コマンド名	機能
実行	CONTINUE	プログラムを現在の中断位置から実行します。
	RUNTO	実行が中断している位置から指定された任意の位置までプログラムを実行します。
	SKIP	プログラムの中断している位置を変更します。
	RERUN	デバッグしているプログラムを強制的に終了させて、もう一度最初からデバッグを行います。
中断点	BREAK	指定された位置に中断点を設定します。
	DELETE	指定した位置に設定された中断点を解除します。
通過点カウント	COUNT	指定された位置に通過カウント点を設定します。
	DELCOUNT	指定した位置に設定された通過カウント点を解除します。
データ	LIST	指定されたデータの値を表示します。
	SET	データの内容を更新します。
	LINKAGE	現在中断しているプログラムの連絡節で定義されているデータの領域を獲得します。
	ASSIGNDATA	レコードとデータファイルを関連付けたファイル識別名を割り当てます。
	OPENDATA	データファイルをオープンします。

	READDATA	レコードのデータ値をデータファイルから 1 レコード分読み込みます。
	WRITEDATA	レコードのデータ値をデータファイルに 1 レコード分書き込みます。
	CLOSEDATA	データファイルをクローズします。
データの監視	DTRACE	データ監視を設定します。
	DELDTR	データ監視を解除します。
	MONITOR	変更時中断の属性を持つデータ監視を設定します。
	DELMON	変更時中断の属性を持つデータ監視を解除します。
条件式の監視	DATACHK	条件監視を設定します。
	DELCHK	現在行っている条件監視を解除します。
状態	ENV	オペランドを指定すると、指定されたオペランドに対応するデバッグ環境を変更します。
	STATUS	中断点、通過カウント点、データ監視、条件監視、暗黙プログラム修飾、ファイル識別名設定、および暗黙スレッド状態の各デバッグ機能について、現在の状況を表示します。
	SCOPE	暗黙プログラム修飾を変更します。
	WHERE	現在中断している位置を通知します。
	CALLS	プログラムの呼び出しの経路を表示します。
	THREADLIST	現在動作しているスレッドおよびスレッド終了事象を通知したスレッドの状態を表示します。
スレッドの操作	ALTERTHREAD	スレッドのサスペンド状態、およびデバッグ事象を通知するかどうかを設定します。
	CURRENTTHREAD	暗黙スレッドを切り替えます。
操作の再現	AUTORUN	自動デバッグを開始します。
終了	QUIT	デバッガを終了します。

各コマンドの機能の詳細および構文の詳細については、“COBOLデバッガ”のヘルプを参照してください。

第5章 サーバ連携機能

NetCOBOLは、分散開発を効率よく行うために、サーバ連携機能を提供しています。
この機能を使用することで、Windows系システム上で開発したプログラム資産をOSIV系システムに配付・登録して、ロードモジュールの作成までを行う事ができます。
本章では、そのサーバ連携機能の説明します。

5.1 OSIV系システムへのプログラム資産の登録

OSIV系プログラムの分散開発では、Windowsシステム上で開発した各種のプログラム資産をOSIV系システムに登録する必要があります。
このためのNetCOBOLのサーバ連携機能として用意されている次の機能を使用します。

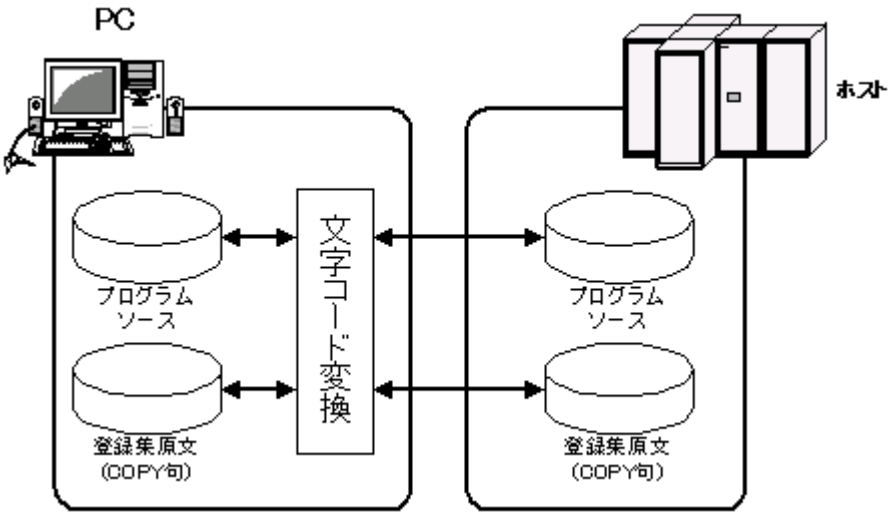
- OSIV系システムのファイル送信

NetCOBOLではサーバ連携機能の一部としてOSIV系システムからのファイル受信の機能も用意されています。この機能については、“3.3 プログラム資産のＰＣへの移行”で説明していますので、必要であればそちらを参照してください。

5.1.1 COBOLソース・登録集の登録

COBOLソース・登録集原文(COPY句)は、OSIV系システムでもWindows系システムでもテキスト形式のファイルであるため、比較的容易に移行できます。
システムで採用する文字コード系は、OSIV系システムではEBCDIC/JEF、Windows系ではASCII/SJISと異なりますが、ファイルを転送する過程で文字コードの変換も自動的に行われます。

図5-1 OSIV系システム～Windows系システム間のソース・登録集ファイルの流通

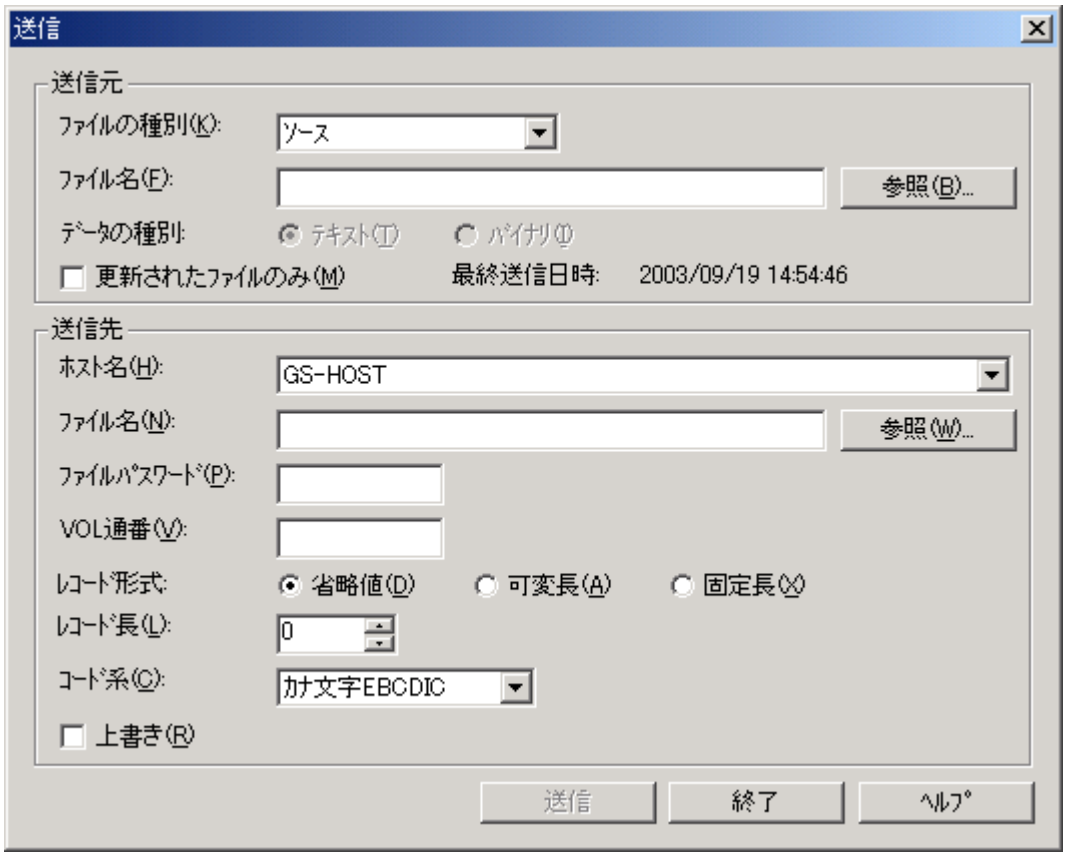


5.1.1.1 Windows系システムでの処理

COBOLプロジェクトマネージャの分散開発支援機能の“送信”機能を使用して、COBOLソース・登録集原文をWindows系システムのファイルから、OSVI系システム上の区分編成ファイルのメンバとして送信します。
以下、その手順を説明します。

1. COBOLプロジェクトマネージャの〔プロジェクト〕－〔分散開発〕メニューから、“送信”を選択します。
2. “送信”ダイアログが表示されるので、〔送信元〕および〔送信先〕に必要な情報を設定します。

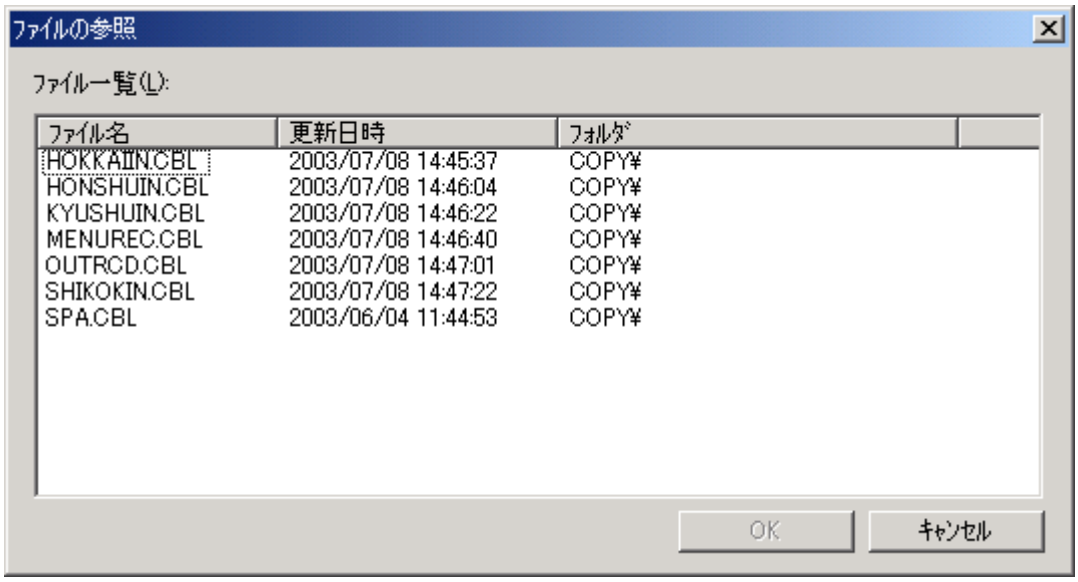
図5-2 送信ダイアログ



3. 〔送信元〕については少なくとも以下の情報を設定します。
   - ファイルの種別:
     送信するファイルの種別を指定します。ここでは送信するファイルの種別に従って“ソース”または“登録集”を選択します。
   - ファイル名:
     送信するファイルの名前を指定します。ここで指定された名前からパス名と拡張子を除いた名前が送信先の区分編成ファイルの名前になります。そのため、ファイル名は区分編成ファイルのメンバ名として有効なものだけが指定できます。
     次のいずれかの方法で指定できます。
     - ワイルドカード指定
       一文字の“*”のみが指定できます。プロジェクトに登録しているファイルのうち、〔ファイルの種別〕で選択している種別に一致するすべてのファイルが対象になります。
     - ファイル名指定
       相対パス指定または絶対パス指定で送信するファイル名を指定します。複数指定する場合、ファイル名毎に引用符文字(“)で囲んで、空白文字を挟んで続けます。
       例: "COPY¥HOKKAIIN.CBL" "COPY¥HONSHUIN.CBL" "COPY¥KYUSHUIN.CBL"

     〔参照〕ボタンをクリックすることで、〔ファイルの種別〕で選択している種別に一致するファイルの一覧から選択することもできます。

図5-3 送信ファイル用の〔ファイルの選択〕ダイアログ



その他情報については、必要な場合のみ指定してください。

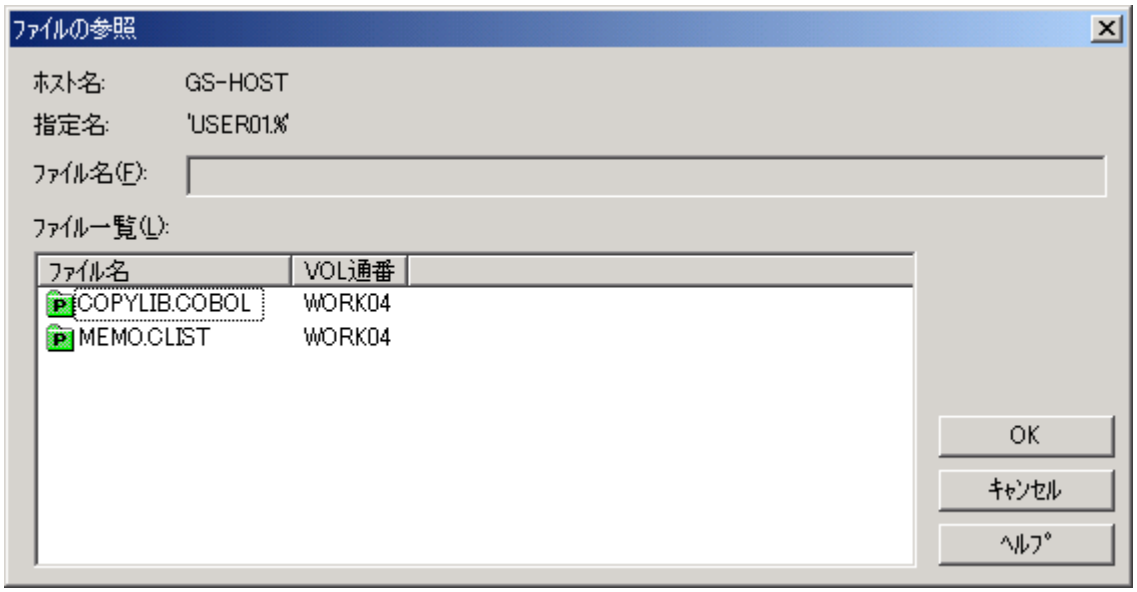
— 更新されたファイルのみ:
〔最終送信日時〕に表示される日時以降に更新されたファイルのみ送信する場合にチェックします。〔最終送信日時〕は、〔ファイルの種別〕で選択している種別毎に日時を保持します。

4. 〔送信先〕については少なくとも以下の情報を設定します。

— ホスト名:
サーバのホスト名を選択します。〔グローバルサーバ連携情報〕ダイアログボックスで指定したホスト名の一覧から選択できます。

— ファイル名:
送信先の区分編成ファイルの名前を完全修飾名で1つだけ指定します。メンバ名を指定すること、ファイル名を複数指定することはできません。
〔参照〕ボタンをクリックすると“ファイルの参照”ダイアログが開きますので、ここから対象となるファイルを選択することもできます。

図5-4 送信先ファイル用の〔ファイルの参照〕ダイアログ



なお、指定した名前の区分編成ファイルがグローバルサーバ上に存在しない場合、指定した名前の区分編成ファイルが作成されます。

— コード系:
送信先のグローバルサーバのコード系を以下の種類から選択します。
- カナ文字EBCDIC
- 英小文字EBCDIC
- EBCDIC(ASCII)

その他の情報については、必要な場合のみ指定してください。

— ファイルパスワード:
送信先のファイルがパスワードで保護されている場合、ファイルパスワードを指定します。

— VOL通番:
送信先のファイルがカタログされていない場合、ボリューム通し番号を指定します。

— レコード形式:
送信先ファイルのレコード形式を、“省略値”、“可変長”、“固定長”から選択します。“省略値”を選択した場合は、〔データの種別〕が“テキスト”の場合は可変長、“バイナリ”の場合は固定長になります。

— レコード長:
送信先ファイルのレコード長を指定します。0を指定した場合は、省略値になります。詳細は、“表:指定値とOSIVシステムのファイル形式との関係”を参照してください。

表5-1 指定値とOSIVシステムのファイル形式との関係

ファイルの種別	データの種別	送信先ファイル	
		レコード形式	レコード長
ソース	テキスト	固定長	80
		可変長(省略値)	最大255(省略値255)
登録集	テキスト	固定長	80
		可変長(省略値)	最大255(省略値255)
画面帳票定義体	バイナリ	固定長(変更不可)	256
その他	テキスト バイナリ	固定長	最大3176(省略値256)
		可変長	最大3180(省略値256)

— 上書き:
送信先の既存のファイルまたはメンバを上書きする場合に指定します。〔更新されたファイルのみ〕を指定していない場合に、指定可能になります。

5. 以上の設定が済んだら、〔送信〕ボタンをクリックして、ファイルの送信処理を開始します。

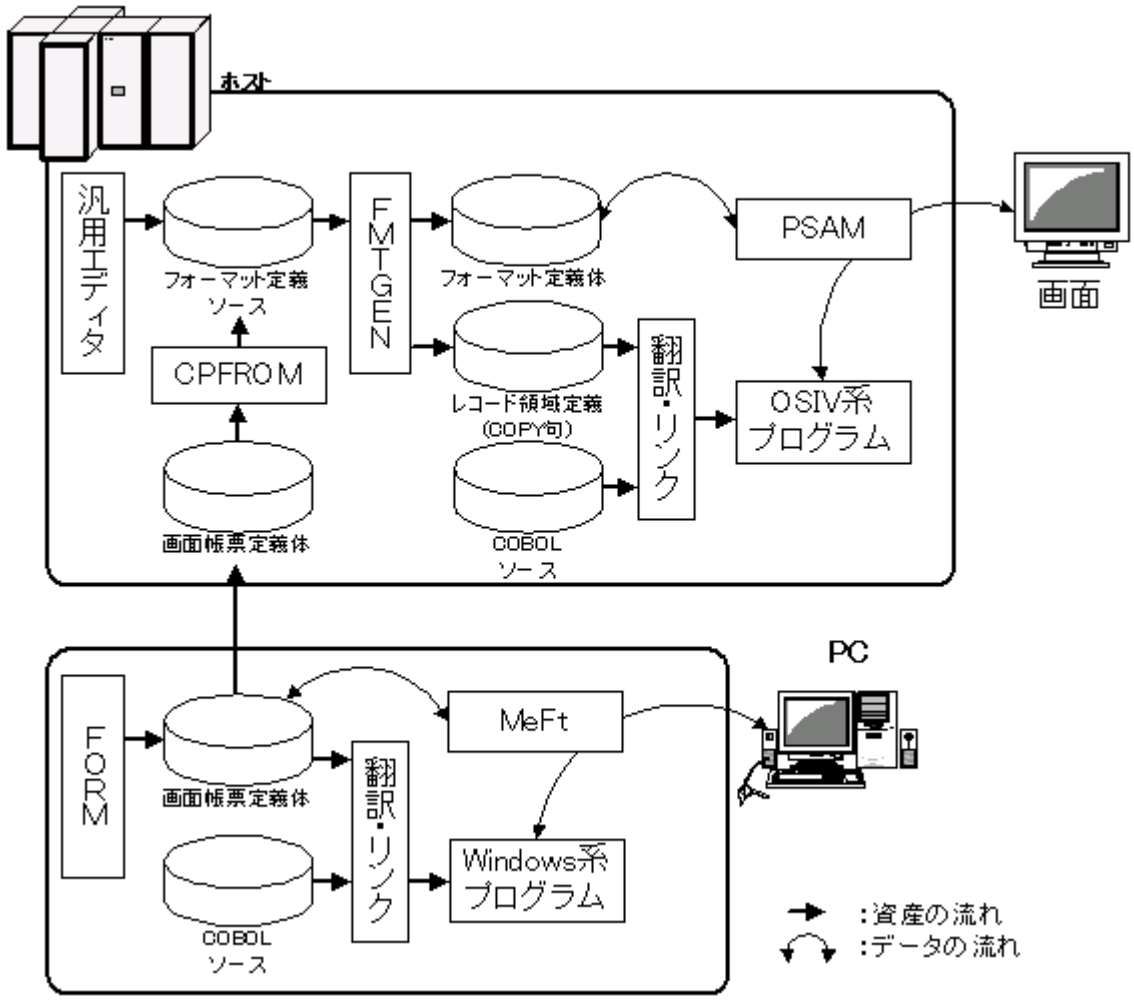
5.1.1.2 OSIV系システムでの処理

COBOLソース・登録集原文のOSIV系システムへの登録は、ファイルの送信以外の処理が必要となることはありません。

5.1.2 画面帳票定義体の登録

Windows系システム上のFORM/PowerFORMを使用して作成した画面帳票定義体は、そのままではOSIV系システムでは使用できません。ツールを用いて、OSIV系システムに送信した画面帳票定義体をフォーマット定義体ソースに変換する必要があります。
以下にその概要を示します。

図5-5　画面帳票定義体→フォーマット定義体の流通

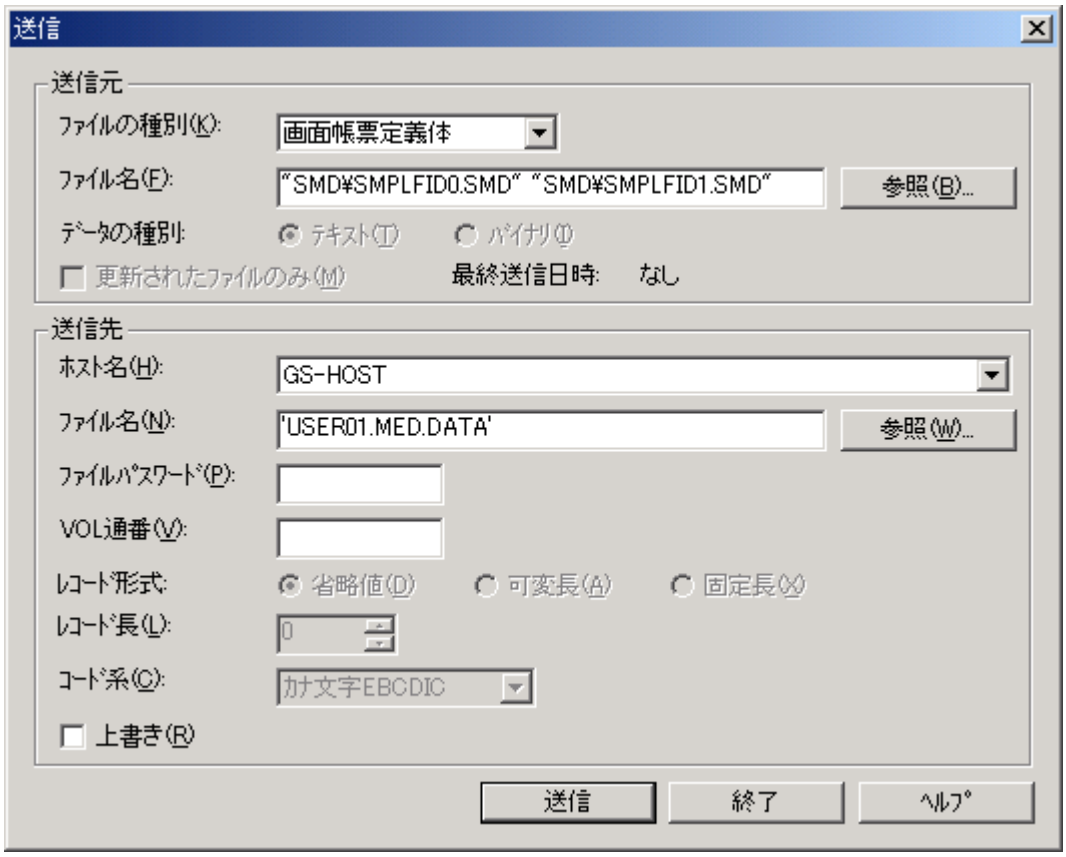


5.1.2.1 Windows系システムでの処理

COBOLプロジェクトマネージャの分散開発支援機能の“送信”機能を使用して、画面帳票定義体を、OSVI系システム上の区分編成ファイルのメンバとして送信します。
基本的な手順は、COBOLソース・登録集原文を送信する場合と同じです。ここでは、異なる設定が必要となる項目のみ説明します。

1. 〔送信元〕は次のように指定してください。
   - ファイルの種別:
     必ず“画面帳票定義体”を選択します。
2. 後は必要に応じて、指定してください。

図5-6 画面帳票定義体送信時の〔送信〕ダイアログの設定例



3. 以上の設定が済んだら、〔送信〕ボタンをクリックして、ファイルの送信処理を開始します。

5.1.2.2 OSIV系システムでの処理

画面帳票定義体から移入機能(CPFORM)を使用し、フォーマット定義体ソースへの変換を行います。以下、移入機能を使用するために必要なデータセットおよびCPFORMの起動パラメタ等について説明します。より詳細な情報は“OSIV PSAM使用手引書 付録Kホスト・ワークステーション連携”を参照してください。

フォーマット定義体の移入機能の使用するデータセット

フォーマット定義体の移入機能を使用する場合の入出力データとそのファイル属性を示します。

図5-7 フォーマット定義体の移入機能における入出力の流れ

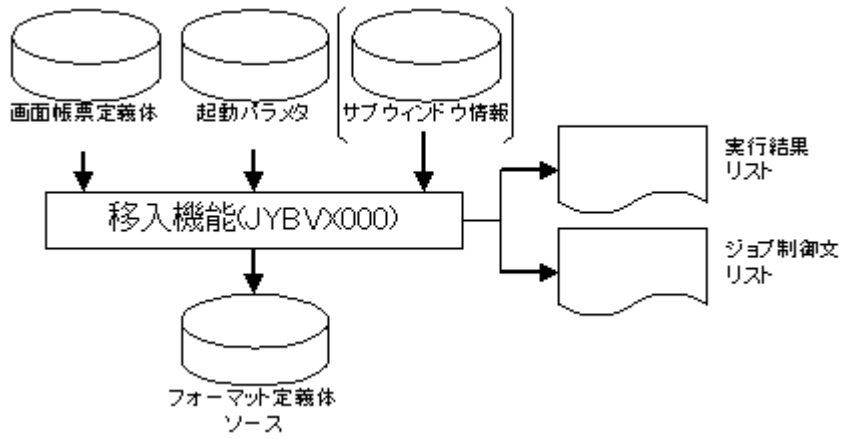


表5-2 フォーマット定義体の移入機能におけるファイルの属性

ファイル種別	指定方法（DD名またはアクセス名）	ファイル属性			
		編成	レコード形式	レコード長（バイト）	ブロック長（バイト）
画面帳票定義体ライブラリ	ＩＮＦＬＩＢ	順編成 区分編成	Ｆ、ＦＢ	256	レコード長×ｎ
フォーマット定義ソースライブラリ	ＦＭＴＳＯＣ	順編成 区分編成			
サブウィンドウ情報格納ファイル	ＣＰＩＮＦ	区分編成	Ｆ、ＦＢ	80	レコード長×ｎ
起動パラメタ格納ファイル	ＣＰＰＡＲＭ	順編成	Ｆ、ＦＢ	80	レコード長×ｎ
リスト出力先（実行結果リスト）	ＳＹＳＰＲＩＮＴ	順編成 区分編成	ＦＡ ＦＢＡ	255以上	レコード長×ｎ
			ＶＡ ＶＢＡ	255以上	レコード長＋４以上
リスト出力先（ジョブ制御文リスト）	ＳＹＳＬＩＳＴ	順編成 区分編成	ＦＡ ＦＢＡ	255以上	レコード長×ｎ
			ＶＡ ＶＢＡ	255以上	レコード長＋４以上

サブウィンドウ情報格納ファイルは通常は必要ありません。サブウィンドウの変換を行う場合にのみ指定してください。

CPFORMの起動パラメタ

CPFORMの起動パラメタは次の2つの指定方法があります。

- JCLの起動パラメタで指定する方法
  OSIV/MSPではEXEC文のPARMパラメタで、OSIV/XSPではPARA文で指定します。
- 起動パラメタ格納ファイルに指定する方法
  起動パラメタ格納ファイルは、80欄カードイメージで記述したファイルです。
  この起動パラメタ格納ファイルは、EXEC文のPARMパラメタ、またはPARA文で記述した起動パラメタを補うものであり、同時に指定が可能です。

起動パラメタ格納ファイルの形式

以下に、起動パラメタ格納ファイル内部の形式を示します。起動パラメタ格納ファイルの属性については、“フォーマット定義体の移入機能におけるファイルの属性”を参照してください。

図5-8 CPFORM起動パラメタ格納ファイルの形式

欄	1～71	72	73～80
内容	パラメタ欄	継続欄	任意

〔内容の説明〕

[1] パラメタ欄

第1欄目から各起動パラメタを記述します。
起動パラメタと起動パラメタの間はコンマで区切らなければなりません。1つの起動パラメタは複数のカードに跨がることはできません。1つの起動パラメタは必ず1カード内に記述してください。

[2] 継続欄

継続欄は起動パラメタがそのカードに入りきらない場合、次のカードに継続させるために使用します。継続欄はカードイメージの72欄目です.次のカードに継続させる場合には、その継続欄に空白以外の文字を記入します。

起動パラメタ格納ファイル使用時の注意事項

起動パラメタを起動パラメタ格納ファイルで指定する場合には以下の注意が必要です。

- 起動パラメタ格納ファイル内では、起動パラメタの省略時解釈(下線部の採用)は行われません。ただし、EXEC文のPARMパラメタまたはPARA文が指定されない場合に限り省略時解釈が行われます。
- EXEC文のPARMパラメタまたはPARA文で指定された起動パラメタと起動パラメタ格納ファイルに同一の起動パラメタが指定された場合は、起動パラメタ格納ファイルに記述されたものが有効になります。
- 起動パラメタ格納ファイルには最大100カードまで指定可能です。これを超えた場合はエラーメッセージを出力し、変換処理を中断します。

起動パラメタ

以下に、CPFORMの起動パラメタの意味を示します。

表5-3 CPFORMの起動パラメタ

指定形式	説明
ＭＥＭ＝｛＋｜メンバ名｝	変換する画面帳票定義体のメンバ名および登録するフォーマット定義体ソースのメンバ名を指定します。 ＦＴＹＰＥパラメタでＰＳを指定した場合、本パラメタを省略することはできません。 ＋　　:全メンバを変換対象とします。 ＦＴＹＰＥパラメタでＰＳを指定した場合は指定できません メンバ名:指定したメンバのみを変換対象とします。 ＦＴＹＰＥパラメタでＰＳを指定した場合は登録するフォーマット定義ソースのメンバ名を指定します。
ＦＴＹＰＥ＝｛ＰＯ｜ＰＳ｝	画面帳票定義体のファイル編成を指定します。 ＰＯ:区分編成ファイル ＰＳ:順編成ファイル
｛ＲＥＰ｜ＮＯＲＥＰ｝	フォーマット定義ソースを登録するライブラリに同名のメンバが存在する場合、置き換えるか否かを指定します。 ＲＥＰ　:置き換えます。 ＮＯＲＥＰ:置き換えません。
ＷＫＳＺ＝ｎ	移入機能が使用する作業領域の大きさをキロバイト単位で指定します(1≦n≦16000)。
ＬＩＮＥＣＴ＝ｎ	制御文リストの１ページあたりの印刷行数を指定します。 (10≦n≦999)
ＬＡＹＯＵＴ	画面定義体をレイアウト記述のフォーマット定義体ソースに変換することを指定します。
ＤＡＴＡ＝ （ ＤＣＡ | ＣＵＲ | ＡＩＤＰ ）	ＦＯＲＭで定義不可能なデータ項目の生成を指定します。なお、レコード定義がない場合は無視されます。 ＤＣＡ　:装置制御情報指定領域のデータ項目 ＣＵＲ　:カーソル位置指定領域のデータ項目 ＡＩＤＰ:アテンションコードおよびカーソル位置指定領域のデータ項目 なお、ＣＵＲおよびＡＩＤＰは画面定義体にのみ有効です。

WMODE＝ {ＥＲＡＳＥ　｜ ＰＡＲＴＩＡＬ}	画面表示の基本動作を指定します。画面定義体の場合のみ有効です。 ＥＲＡＳＥ　:前画面消去表示 ＰＡＲＴＩＡＬ:上書き表示
ＤＰＩ＝{１６０｜２４０}	プリンタの解像度を指定します。バーコード項目は指定された解像度に合わせて変換されます。 １６０:１６０ＤＰＩ ２４０:２４０ＤＰＩ
ＨＳＫＩＰ	水平スキップ機能付きプリンタ装置用のフォーマット定義体ソースに変換することを指定します。なお、基準文字間隔１．５ピッチの日本語項目を定義している場合は指定できません。
ＦＩＧＩＤＭ＝ｎ	ＦＯＲＭで定義した実データ組み込みメディア項目を修飾付き図形識別子指定領域として変換することを指定します。省略時は、図形識別子指定領域に変換します。なお、実データのイメージ項目はイメージ識別子指定領域に変換します。 （ｎは図形データ領域の大きさ:1≦n≦32000）
ＭＤＴ	ＦＯＲＭで定義不可能なフィールド属性（ＭＤＴ）の生成を指定します。画面定義体のときのみ有効です。
ＤＥＶ＝ＤＰ２０	ウィンドウ型ディスプレイ装置用のフォーマット定義体ソースに変換することを指定します。画面定義体のときのみ有効です。
ＭＥＤＳＩＺＥ＝ {１７０｜ｎ}	画面帳票定義体読み込み領域のサイズを１０２４バイト単位で指定します。(170≦n≦1500)
ＭＳＧ＝{Ｅ｜Ｎ}	実行結果メッセージの出力言語を指定します。 Ｅ:英語メッセージ Ｎ:日本語メッセージ
ＥＤＩＴＰＴＮ＝ {ＯＬＤ｜ＮＥＷ}	数字項目の編集形式を指定します。 ＯＬＤ:従来どおり。 ＮＥＷ:拡張形式 以下、その対応を示す。 ＦＯＲＭでの指定 ／ ＥＤＩＴＰＴＮパラメタ値 ＋表示 ／ ＯＬＤ ／ ＮＥＷ -¥¥¥,¥¥9 ／ なし ／ ¥¥¥,¥¥9-　(P24) ／ -¥¥¥,¥¥9　(P51) -¥¥¥,¥¥9 ／ あり ／ ¥¥¥,¥¥9+　(P47) ／ +¥¥¥,¥¥9　(P52) ZZZZZZZ- ／ なし ／ ZZZZZZ9-　(P16) ／ ZZZZZZZ-　(P53) ZZZZZZZ- ／ あり ／ ZZZZZZ9+　(P39) ／ ZZZZZZZ+　(P54) -ZZZZZZZ ／ なし ／ -ZZZZZZ9　(P22) ／ -ZZZZZZZ　(P55) +ZZZZZZZ ／ あり ／ +ZZZZZZ9　(P45) ／ +ZZZZZZZ　(P56) Z9.Z9.Z9 ／ - ／ 99.99.99　(P29) ／ Z9.Z9.Z9　(P61) なお、括弧内はフォーマット定義ソースのEDITオペランド値です。

SUBWINDOW制御文

SUBWINDOW制御文は、サブウィンドウとして変換する画面定義体の名前、位置およびサイズを指定する制御文です。SUBWINDOW制御文は、サブウィンドウ情報格納ファイルに、親ウィンドウ名をメンバ名とするメンバ内に記述、格納しておきます。

サブウィンドウ情報格納ファイルの形式

以下に、サブウィンドウ情報格納ファイル内部の形式を示します。サブウィンドウ情報格納ファイルの属性については、“フォーマット定義体の移入機能におけるファイルの属性”を参照して

ください.

図5-9　サブウィンドウ情報格納ファイルの形式

欄	1	2～	～71	72	73～80
内容	制御文字欄	命令欄	オペランド欄	継続欄	任意

〔内容の説明〕

1. 制御文字欄
   第1欄で、制御文の開始を示す“-”を指定します.
2. 命令欄
   サブウィンドウ変換を示す命令“SUBWINDOW”を制御文字に続く第2欄から記述します。
3. オペランド欄
   サブウィンドウ変換の各オペランドを記述します。命令欄とは1個以上の空白を置かなければなりません。
4. 継続欄
   継続欄は制御文のオペランドがそのカードに入りきらない場合、次のカードに継続させるために使用します。継続欄はカードイメージの72欄目です。次のカードに継続させる場合には、その継続欄に空白以外の文字を記入します。

記述形式

以下のSUBWINDOW制御文の記述形式とその記述例を示します。

図5-10 SUBWINDOW制御文の記述形式

制御文字	命令	オペランド
－	ＳＵＢＷＩＮＤＯＷ	メンバ名 ＰＯＳ＝（行数，列数） ＳＩＺＥ＝（行数，列数）

〔図の説明〕

1. メンバ名
   サブウィンドウに変換する画面定義体のメンバ名を指定します。
2. POS=(行数,列数)
   サブウィンドウの左上角位置を行列数の形式で指定します。
3. SIZE=(行数,列数)
   サブウィンドウのサイズを行列数の形式で指定します。

SUBWINDOW制御文の記述例

以下に、SUBWINDOW制御文の記述例を示します.

図5-11 SUBWINDOW制御文の記述例

```
-SUBWINDOW S1SCRN  POS=(2,3) SIZE=(22,76)
-SUBWINDOW S2SCRN  POS=(5,6) SIZE=(20,60)
```

この例は親ウィンドウM1SCRNに対して、サブウィンドウS1SCRN、S2SCRNを結合するものです。

SUBWINDOW制御文の注意事項

- MEMオペランドで指定したメンバ名と同一のメンバ名に格納されたサブウィンドウ情報を元にサブウィンドウの変換を行います。このため、MEMオペランドで指定したメンバ名と異なるメンバ名でサブウィンドウ情報を作成した場合は、サブウィンドウの変換は行われません。
- SUBWINDOW制御文は、1メンバにつき20個まで指定できます。20個を超えた分については無視されます。
- MEMオペランドに+を指定した場合は、サブウィンドウ用に作成した画面定義体についても

フォーマット定義ソースが生成されます。

- 画面帳票定義体が順編成ファイルの場合、サブウィンドウの変換は行われません。

CPFORMの起動ジョブのJCL

CPFORMを起動するジョブのJCLの例を示します。

図5-12 MSPでのCPFORM起動ジョブのJCL例

```
//USER1CPF JOB,…
//CPFORM   EXEC PGM=JYBVX000,REGION=1536,PARM=('MEM=+')
//INFLIB   DD DSN=USER1.INFLIB.DATA,DISP=SHR
//FMTSOC   DD DSN=USER1.FMTSRC.DATA,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLIST  DD SYSOUT=*
//CPINF    DD DSN=USER1.CPINF.DATA,DISP=SHR
//CPPARM   DD DSN=USER1.CPPARM.DATA,DISP=SHR
/*
```

図5-13 XSPでのCPFORM起動ジョブのJCL例

```
¥ JOB USER1CPF
¥ EX JYBVX000,RSIZE=1536
¥ PARA MEM=+
¥ FD INFLIB=DA,FILE=USER1.INFLIB.DATA
¥ FD FMTSOC=DA,FILE=USER1.FMTSRC.DATA
¥ FD SYSPRINT=DA,VOL=WORK,BLK=(13030*50,50),SOUT=A
¥ FD SYSLIST=DA,VOL=WORK,BLK=(13030*50,50),SOUT=A
¥ FD CPINF=DA,FILE=USER1.CPINF.DATA
¥ FD CPPARM=DA,FILE=USER1.CPPARM.DATA
¥/
```

CPFORMの復帰コード

CPFORMの復帰コードを示します。

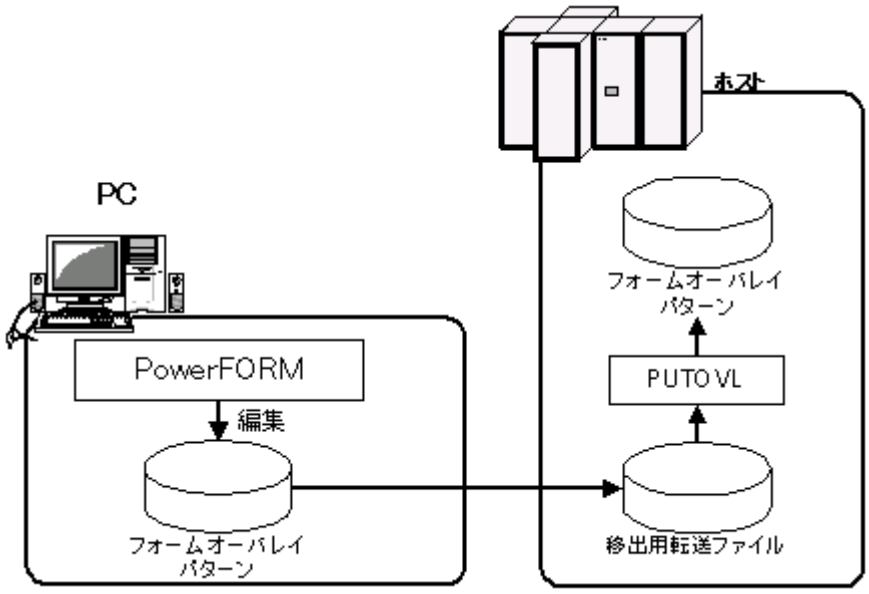
表5-4 CPFORMの起動パラメタ

状態	復帰値		説明
	ＭＳＰ	ＸＳＰ	
正常	０	１０	フォーマット定義体ソースを正常に作成し、データセットに出力した。
警告	４	２０	軽度のエラーがあったが、フォーマット定義体ソースを作成し、データセットに出力した。
異常	８	３０	重大なエラーが発生したため、フォーマット定義体ソースを作成できなかった。またはデータセットに出力できなかった。

5.1.3 オーバレイ定義体の送信

Windows系システム上のFORM/PowerFORMを使用して作成したオーバレイ定義体は、そのままではOSIV系システムでは使用できません。ツールを用いて、OSIV系システムに送信したオーバレイ定義体をフォームオーバレイパターンに変換する必要があります。
以下にその概要を示します。

図5-14　オーバレイ定義体の流通

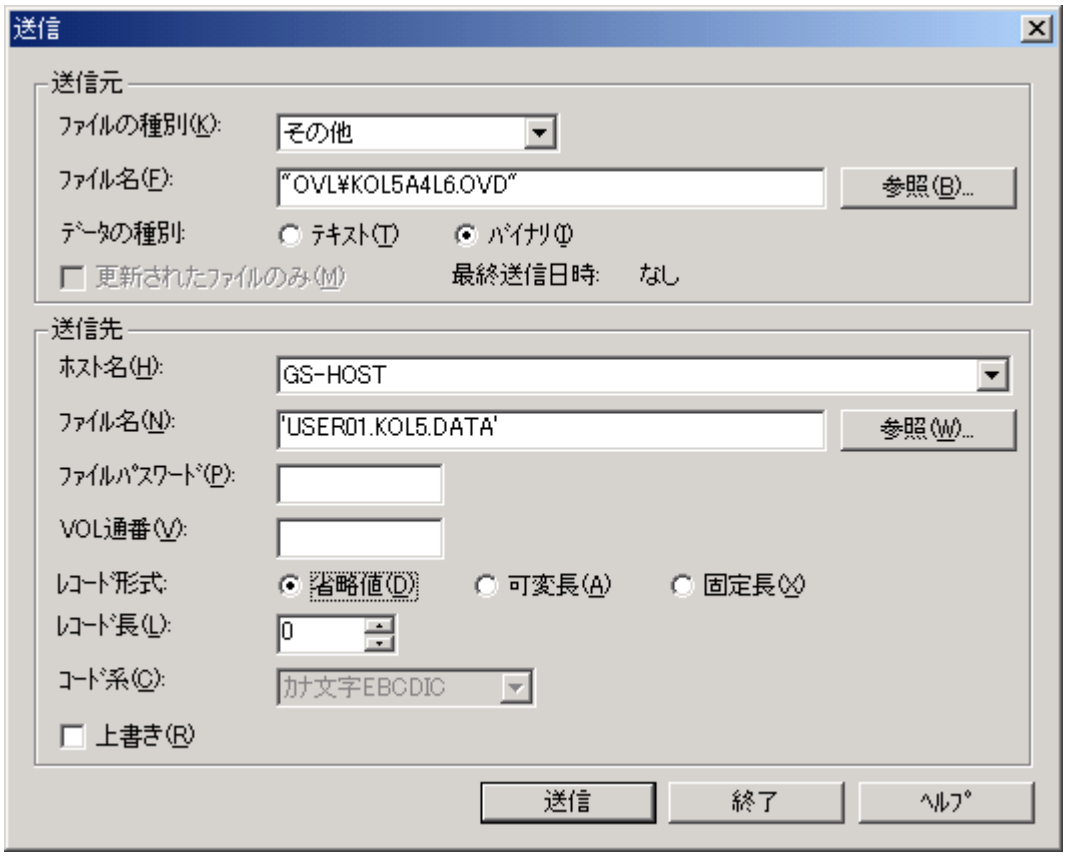


5.1.3.1 Windows系システムでの操作

COBOLプロジェクトマネージャの分散開発支援機能の“送信”機能を使用して、オーバレイ定義体を、OSVI系システム上の順編成ファイルとして送信します。
基本的な手順は、COBOLソース・登録集原文を送信する場合と同じです。ここでは、異なる設定が必要となる項目のみ説明します。

1. 〔送信元〕は次のように指定してください。
   - ファイルの種別:
     必ず“その他”を選択します。
   - データの種別:
     必ず“バイナリ”を選択します。
2. 後は必要に応じて、指定してください。

図5-15 オーバレイ定義体送信時の〔送信〕ダイアログの設定例



3. 以上の設定が済んだら、〔送信〕ボタンをクリックして、ファイルの送信処理を開始します。

5.1.3.2 OSIV系システムでの操作

オーバレイ定義体からAP/DFの移入機能(PUTOVL)を使用し、フォームオーバレイパターンをイメージライブラリまたはトランスライブラリに取り出します。
以下、移入機能に使用するコマンドとそのオペランドについて説明します。より詳細な情報は“OSIV AP/DF説明書 V20L10”を参照してください。

フォーマット定義体の移入機能のコマンド形式

フォーマット定義体の移入には、AP/DFのPUTOVLコマンドを使用します。PUTOVLコマンドの実行はフルスクリーンモードとラインモードの2種類の形態がありますが、順編成データセットに格納したオーバレイ定義体を入力とする場合、ラインモードを使用する必要があります。

表5-5 フォーマット定義体の移入機能のコマンド形式

命令	オペランド
ＰＵＴＯＶＬ	ＰＳＤＤ（アクセス名）　ＯＶＬＮＭ（オーバレイ名） ＦＩＬＥ（ファイル名） 〔ＴＹＰＥ＝{1, 2, 3, 4, 5}　〕

〔{ REP / NOREP }〕

〔**KF**ＩＬＥ（文字パターンマスタファイル)〕
〔**LE**T〕

〔KLANG={ K / J / C / T }〕

〔ＰＴＮＳＩＺＥ（パターンサイズの上限値)〕
〔ＧＲＰＳＩＺＥ（グループサイズの上限値)〕
〔ＤＥＶＴＹＰＥ（６)〕

〔{ ＰＲＩＮＴ / ＮＯＰＲＩＮＴ / ＳＹＳＰＲＩＮＴ }〕

各オペランドの意味と指定方法は次の通りです。

1. ＰＳＤＤ（アクセス名）
   オーバレイ定義体を順編成ファイルから取り出して処理を行うことを指示します。アクセス名には、オーバレイ定義体が格納されている順編成ファイルに対して割り当てられたアクセス名を指定します。

2. ＯＶＬＮＭ（オーバレイ名）
   格納するフォームオーバレイパターンのパターン名(英数字4文字以内)を指定します。
   本オペランドは、PSDDオペランドが指定された場合は省略不可です。
   移入時のフォームオーバレイパターン名は、TYPEオペランドまたはDEVTYPE(6)オペランドの指定値により、以下のように決まります。

表5-6 TYPEオペランドの指定とOVLNMオペランドに指定するオーバレイ名

TYPEオペランドの指定	オーバレイ名
TYPE(1)が指定された場合(NLP/CLP)	KOL1xxxx
TYPE(2)が指定された場合(OPR/CFC3)	KOL1xxxx
TYPE(3)が指定された場合(CFC2)	KOL3xxxx
TYPE(4)が指定された場合(VIP)	KOL2xxxx
TYPE(5)が指定された場合(APP)	KOL5xxxx

3. ＦＩＬＥ（ファイル名）
   フォームオーバレイパターンを格納するイメージライブラリ名またはトランスライブラリ名(MSPは44文字以内、XSPは26文字以内)を指定します。
   指定するファイルの種類は、TYPEオペランドの指定値により以下のように異なります。

表5-7 TYPEオペランドの指定とFILEオペランドに指定するファイルの種類

TYPEオペランドの指定	ファイルの種類
TYPE(1)が指定された場合(NLP/CLP)	イメージライブラリ名
TYPE(2)が指定された場合(OPR/CFC3)	イメージライブラリ名
TYPE(3)が指定された場合(CFC2)	イメージライブラリ名
TYPE(4)が指定された場合(VIP)	トランスライブラリ名

TYPE(5)が指定された場合(APP)	イメージライブラリ名

利用者識別修飾子付きのファイル名を指定する場合は、ファイル名の前後にそれぞれ3つの引用符をつけなければなりません。
本オペランドは省略不可です。

4. ＴＹＰＥ（{１｜２｜３｜４｜５}）
フォームオーバレイパターンの出力対象となる装置を、以下の番号で指定します。
— １：
NLPまたはCLPに出力する場合、またはページプリンタ装置にラインプリンタモードで出力する場合に指定します。
— ２：
OPRまたはCFC3に出力する場合に指定します。
— ３：
CFC2に出力する場合に指定します。
— ４：
VIPに出力する場合に指定します。
— ５：
ページプリンタ装置にページプリンタモードで出力する場合に指定します。
なお、本オペランドが省略された場合は1を指定したと見なします。

5. {ＮＯＲＥＰ｜ＲＥＰ}
出力先のイメージライブラリまたはトランスライブラリに同一オーバレイパターン名が存在した時に置き換えるかどうかを指定します。
— ＮＯＲＥＰ:
置き換えない場合に指定します。
— ＲＥＰ:
置き換える場合に指定します。
なお、本オペランドが省略された場合はNOREPを指定したと見なします。

6. ＫＦＩＬＥ（文字パターンマスタファイル名）
文字パターンマスタファイル名(MSPは44文字、XSPは26文字以内)を指定します。ただし、このオペランドが有効となるのは、TYPEオペランドに1(NLP/CLP)、2(OPR/CFC3)、3(CFC2)が指定された場合のみです。

利用者識別修飾子付きのファイル名を指定します場合は、ファイル名の前後にそれぞれ3つの引用符をつけなければなりません。
なお、本オペランドが省略された場合は、KLANGオペランドで指定されたコード系のシステム標準文字パターンマスタファイルが割り当てられる。

7. ＬＥＴ
オーバレイパターンの作成時、実行状態コードが8以上の構文エラーが発生しても、オーバレイパターンの作成を行う場合に指定します。

8. ＴＹＰＥ（{Ｊ｜Ｋ｜Ｃ｜Ｔ}）
フォームオーバレイパターンを出力する際の文字のコード系を以下の文字で指定します。
— Ｊ:
JEFコード系で出力する。
— Ｋ:
KEFコード系で出力する。
— Ｃ:

CEFコード系で出力する。

― T:
TEFコード系で出力する。

KFILEオペランドが指定された場合は、本オペランドは無視されるので注意が必要です。
なお、本オペランドが省略された場合は、端末言語種別と同じコード系を指定したと見なします。

9. ＰＴＮＳＩＺＥ(パターンサイズの上限値)
フォームオーバレイパターンサイズの上限値をKB単位で指定します。なお、指定できる上限値の最大値は、TYPEオペランドの指定値により以下のように異なります。

表5-8 TYPEオペランドの指定とPTNSIZEの最大値

TYPEオペランドの指定	ファイルの種類
TYPE(1)が指定された場合(NLP/CLP)	10240KB以内(省略値:512KB)
TYPE(2)が指定された場合(OPR/CFC3)	512KB以内(省略値:512KB)
TYPE(3)が指定された場合(CFC2)	256KB以内(省略値:256KB)
TYPE(4)が指定された場合(VIP)	96KB以内(省略値:64KB)
TYPE(5)が指定された場合(APP)	上限なし

10. ＧＲＰＳＩＺＥ(グループサイズの上限値)
グループサイズの上限値(96KB以内)をKB単位で指定します。省略値は64KBと見なされます。
なお、本オペランドが有効となるのは、TYPEオペランドに4(VIP)が指定された場合のみです。
また、本オペランドを指定した場合は、TYPEオペランドは無視されるので、注意が必要です。

11. {ＰＲＩＮＴ|ＮＯＰＲＩＮＴ|ＳＹＳＰＲＩＮＴ}
ADJUSTにおける処理結果リストの出力先を指定します。

― ＰＲＩＮＴ（ファイル名）
結果リストをファイルに出力する場合に指定します。ファイル名には、結果リストを出力するファイル名(MSPは44文字以内、XSPは26文字以内)を指定します。なお、割り当てられていないファイルは新規に割り当てられる。
利用者識別修飾子付きのファイル名を指定します場合は、ファイル名の前後にそれぞれ3つの引用符をつけなければなりません。

― ＮＯＰＲＩＮＴ
結果リストを出力しない場合に指定します。

― ＳＹＳＰＲＩＮＴ（クラス名）
結果リストをクラスに出力する場合に指定します。クラス名には結果リストを出力するクラス名を指定します。
なお、本オペランドの全てが省略された場合は、結果リストは端末に出力されます。

5.2 ビルド制御文生成機能

ビルド制御文生成機能は、OSIV系システムに転送したソースプログラム、登録集原文などの資産を使用してロードモジュールを作成するための、ビルド制御文(JCLまたはCLIST)の雛型を生成し、指定したファイルに出力する機能です。
ビルド制御文の雛型は、プロジェクトに登録しているCOBOLソースファイル名、翻訳オプションおよびファイルの送信情報を使用して生成します。

5.2.1 ビルド制御文雛型の生成時の規則

ビルド制御文の雛型は、プロジェクトに登録しているCOBOLソースファイル名、翻訳オプションおよびファイルの送信情報を使用して生成します。
以下、プロジェクトファイルに登録している情報とファイルの送信情報から、どのような規則でビルド制御文の雛型を生するか説明します。

- 翻訳対象ファイル
  プロジェクトに登録しているソースプログラムファイルのうち、次のものを翻訳対象ファイルとする翻訳処理をビルド制御文として展開します。
  - ツリー上で〔COBOLソースファイル〕フォルダに登録されているCOBOLソースファイル。ただし、〔プリコンパイラ〕フォルダ、〔INSDBINF〕フォルダ、〔AAD配付ソースファイル〕フォルダが下位階層に定義されているものは対象になりません。
  - 〔プリコンパイラ〕フォルダの最下位の階層に登録したソースファイル。
  - 〔AAD配付ソースファイル〕フォルダに登録された配付ソースファイル。

  翻訳対象ファイルの名前としては、プロジェクトに登録しているファイル名から、パス名および拡張子を除いた名前を使用します。この名前はOSIV系システムの区分編成ファイルのメンバ名として正しい名前でなければなりません。
- ソース格納データセット名、登録集格納データセット名
  “送信”機能でCOBOLソースファイルおよび登録集を送信した際の送信情報を元にデータセット名、ファイルパスワードおよびボリューム通し番号が制御文に展開します。
  送信を行っていない場合、データセット名は“SAMPLE.COBOL”の名前で展開します。実際のデータセット名に従って、ビルド制御文の雛型を修正します。
- 翻訳オプション
  プロジェクトに設定している翻訳オプションをOSIV系のCOBOL85の翻訳オプション形式に変換したものを制御文に展開します。設定した翻訳オプションのうち、OSIV系のCOBOL85と互換を持つオプションのみが展開されます。
  展開するオプションの一覧を以下に示します。

表5-9 ビルト制御文に展開する翻訳オプションの一覧

オプション名	説明	備考
CONF	規格の違いによるメッセージの出力の可否	-
COPY	登録集原文の表示	-
DLOAD	プログラム構造の指定	-
EQAULS	ソート文での同一キーデータの処理方法	-
FLAG	診断メッセージのレベル	-
FLAGSW	COBOL言語要素に対しての指摘メッセージ表示	-
LANGLVL	ANSI COBOL規格の指定	-

LIB	登録集ファイル格納場所の指定	登録集データセット名は [送信]機能を用いて登録集原文をサーバへ登録した際の情報を元に決定します。 送信を行っていない場合は、次の名前が用います。 ‘SAMPLE.COBOL’
LINECOUNT	翻訳リスト1ページ当たりの行数	-
LINESIZE	翻訳リスト1行当たりの文字数	-
LIST	目的プログラムリスト出力可否	-
MESSAGE	オプション情報リスト、翻訳統計情報の出力可否	-
NCW	日本語利用者語の文字集合指定	-
NUMBER	ソースプログラムの一連番号領域の指定	-
OBJECT	目的プログラム出力の可否	●OBJECT指定時 オブジェクト出力先/リンクエディット制御文を展開する。 “OBJECT”を翻訳オプションには展開しない。 ●“NOOBJECT”指定時 オブジェクト出力先/リンクエディット制御文を展開しない。 “OBJECT”を翻訳オプションには展開しない。
OPTIMIZE	広域最適化の扱い	-
QUOTE/APOST	表意定数QUOTEの扱い (引用符文字の指定)	-
SDS	符号付き10進項目の符号整形の可否	-
SOURCE	ソースプログラムリストの出力可否	-
TRUNC	桁落とし処理の可否	-
XREF	相互参照リスト出力の可否	-
ZWB	符号付き外部10進項目と英数字項目の比較	-

- リンクオプション
  リンクオプションは設定してあっても、ビルド制御文の雛型には展開しません。
- オブジェクト格納データセット名
  ビルド制御文の種類として、CLISTを選択した場合、格納データセット名として“SAMPLE.OBJ”を展開します。使用可能なデータセット名に修正してください。
- ロードモジュール名
  ロードモジュール名は、プロジェクトの最終ターゲットのファイル名から、パス名および拡張子を除いた名前を使用します。この名前はOSIV系システムの区分編成ファイルのメンバ名として正しい名前でなければなりません。
  プロジェクトに複数の最終ターゲットファイルを登録している場合、最終ターゲットファイル毎にビルド制御文の雛型が作成し、1つのファイルに出力します。
- ロードモジュールデータセット名
  格納データセット名として“SAMPLE.LOAD”を展開します。使用可能なデータセット名に修正してください。

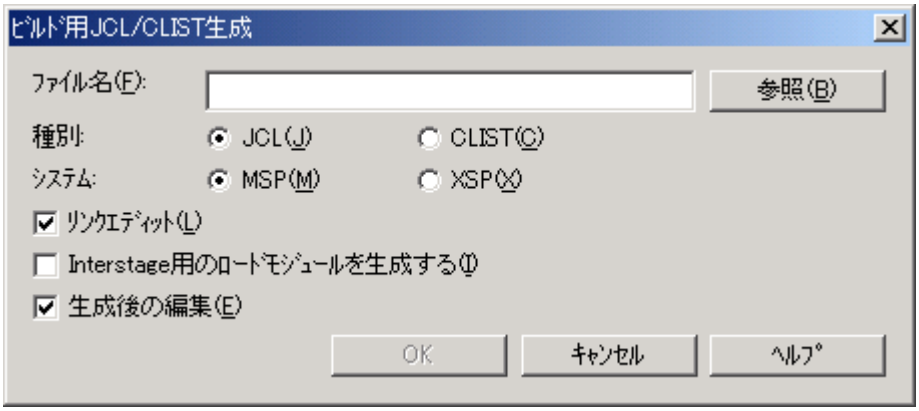
- ロードモジュールの入口名
  ロードモジュールの入口名として“ENTRY1”を展開します。実際の入口名に合わせて修正してください。
- AADアプリケーション固有の展開
  AADアプリケーション用のビルド制御文の雛型を生成する場合、以下の点が通常のOSIV系プログラムに対して生成するビルド制御文の雛型と異なります。
  - Interstageシステム提供のCOBOL登録集の指定“CORBA DD文(XSPの場合、CORBA FD文)”を追加します。
  - リンクエディット処理の生成を指定した場合、リンクオプションに“DYNAMIC”、“NCAL(XSPの場合はNOCALL)”を展開します。
  - スケルトンおよびオペレーション処理プログラムのロードモジュールを生成する場合、翻訳オプション“PSF”を追加します。また、プロジェクトにIDLソースファイルを登録している場合、スケルトンのオブジェクトモジュールが初めに結合されます。なお、IDLソースファイルを登録していない場合は、スケルトンのオブジェクトモジュールが最初に結合されるようにビルド制御文の雛型を修正する必要があります。
  - Interstageの出口プログラムのロードモジュールと、スタブのロードモジュールを生成する場合、翻訳オプション“PSF”および“NAME”を追加します。

5.2.2 ビルド制御文雛型の生成手順

ビルド制御文の雛型の生成は次の手順で行います。

1. 〔プロジェクト〕-〔分散開発〕メニューの“ビルド制御文生成”を選択します。“〔ビルド用JCL/CLIST生成〕ダイアログ”が表示されます。

図5-16 [ビルド用JCL/CLIST生成] ダイアログ



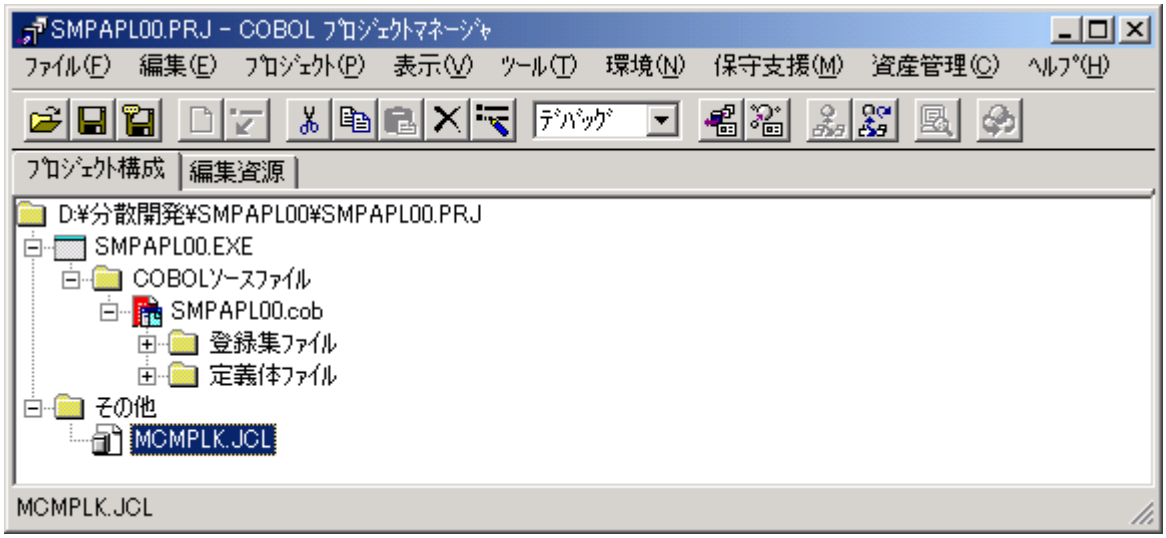
2. このダイアログで次の情報を設定します。
   - ファイル名:
     生成したビルド制御文の雛型を出力するファイル名を指定します。
     [参照]ボタンをクリックすると、[ファイルを開く]ダイアログが開いて、そこから出力先のフォルダ名、ファイル名を選択することもできます。
   - システム:
     生成したビルド制御文を実行するOSIV系システムの種類を選択します。
   - リンクエディット:
     リンクエディット制御文を生成するかどうかを指定します。翻訳オプションとして“NOOBJECT”を指定していない場合に限り、選択可能となります。
   - Interstage用のロードモジュールを生成する:
     AADアプリケーションを作成する場合に指定します。指定した場合、最終ターゲットファイル配下のファイルの登録状況によって、以下の制御文が生成されます。

- スケルトンファイルの配付ソースファイルを登録している場合、スケルトンおよびオペレーション処理プログラムのロードモジュールを作成する制御文を生成します。
- スタブファイルの配付ソースファイルを登録している場合、スタブのロードモジュールを作成する制御文を生成します。
- スケルトンファイルの配付ソースファイルおよびスタブファイルの配付ソースファイルが登録されていない場合、出口プログラムのロードモジュールを作成する制御文を生成します。

— 生成後の編集:
ビルド制御文の雛型生成後、自動的にエディタを起動して編集する場合に指定します。

3. 〔OK〕ボタンをクリックすると、雛型ファイルを生成します。生成に成功した場合、ツリーの〔その他〕フォルダにビルド制御文ファイルが登録されます。
4. “生成後の編集”をチェックしていた場合、直ちにエディタが起動されます。

図5-17 プロジェクトに登録されたビルド制御文(JCL)



5.2.3 生成したビルド制御文雛型とその修正

生成するビルド制御文の例とその修正方法を説明します。

5.2.3.1 JCLの雛型とその修正

以下にOSIV MSPおよびOSIV XSPそれぞれのシステムに対して、生成するJCLの雛型を示します。

図5-18 OSIV MSP用に生成されるJCLの雛型

```
//JOB1 JOB                                          …[1]
//COMP1  EXEC PGM=JMN000,REGION=2048K,
//          PARM=(翻訳オプション…)                   …[2]
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DSN=&&SYSUT1,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2   DD  DSN=&&SYSUT2,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3   DD  DSN=&&SYSUT3,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4   DD  DSN=&&SYSUT4,UNIT=SYSDA,SPACE=(460,(700,100))
```

```
//SYSLIN    DD  DSN=&&LOADSET(ソースファイル名),UNIT=SYSDA,DISP=(NEW,PASS),
//                      SPACE=(80,(500,100,30)),DCB=BLKSIZE=800     …[3]      (a)
//CORBA     DD  DSN=AIM1.AADCPLIB,DISP=SHR                          …[4]
//SYSLIB    DD  DSN=登録集格納ファイル名,UNIT=SYSDA,DISP=SHR,       …[5]
                PASSWORD=パスワード,VOL=SER=VOL通番
//SYSIN     DD  DSN=ソース格納ファイル名(ソースファイル名),UNIT=SYSDA,DISP=SHR,
                PASSWORD=パスワード,VOL=SER=VOL通番                 …[6]
/*
//LKED1   EXEC PGM=JQAL,REGION=2048K,
//           PARM=(リンクエディットオプション…)                      …[7]
//SYSLIB    DD  DSN=SYS1.COBLIB,DISP=SHR
//OBJ       DD  DSN=&&LOADSET,UNIT=SYSDA,DISP=(OLD,DELETE)
//SYSLMOD   DD  DSN=SAMPLE.LOAD,UNIT=SYSDA,DISP=OLD                 …[8]      (b)
//SYSUT1    DD  UNIT=SYSDA,SPACE=(1024,(50,20))
//SYSPRINT DD  SYSOUT=*
//SYSLIN    DD  *                                                   …[9]
              INCLUDE   OBJ(ソースファイル名)
              ENTRY     ENTRY1                                      …[10]
              NAME      ロードモジュール名(R)                       …[11]
/*
//
```

図:OSIV XSP用に生成されるJCLの雛型

```
¥ JOB    JOB1                                                    …[1]
¥COMP1   EX   COBOL,RSIZE=2048                                          ┐
¥ PARA   翻訳オプション…                                          …[2]  │
¥ FD     U01=DA,VOL=WORK,BLK=(460*250,100)                              │
¥ FD     U02=DA,VOL=WORK,BLK=(460*500,100)                              │
¥ FD     U03=DA,VOL=WORK,BLK=(460*500,100)                              │
¥ FD     U04=DA,VOL=WORK,BLK=(460*700,100)                              │
¥ FD     LIST=DA,VOL=WORK,BLK=(255*900,500),SOUT=A                      │ [a]
¥ FD     CORBA=DA,FILE=AIM1.AADCPLIB                             …[4]   │
¥ FD     RLIB=DA,VOL=WORK,FILE=(/,AD),BLK=(80*500,100),          …[3]   │
         MEMBER=ソースファイル名,DRTY=(5,BLK),FCB=BLKSIZE=3200,DISP=CONT │
¥ FD     MLIB=DA,FILE=登録集格納ファイル名,                      …[5]   │
         PSW=パスワード,VOL=VOL通番                                     │
¥ FD     SLIB=DA,FILE=ソース格納ファイル名,MEMBER=ソースファイル名,  …[6]   │
         PSW=パスワード,VOL=VOL通番                                     ┘
¥*                                                                       ┐
¥LKED    EX   LIED,RSIZE=512                                            │
¥ PARA   リンクエディットオプション                                …[7]  │
¥ SW     RLIB=RLIB,DISP=DLT                                      …[12]  │
¥ FD     U01=DA,VOL=WORK,TRK=(10,2)                                     │ [b]
¥ FD     LIST=DA,VOL=WORK,BLK=(255*900,100),SOUT=A                      │
¥ FD     ALIB=DA,FILE=C.ALIB                                            │
¥ FD     ELIB=DA,FILE=SAMPLE.LOAD,DISP=LOCK                      …[8]   │
¥ FD     COIN=*                                                  …[9]   │
         INCLUDE RLIB(ソースファイル名)                                 │
         NAME ロードモジュール名(R),ENT=ENTRY1                …[10] [11] ┘
¥ JEND
```

生成されるJCLの雛型は大きく2つの部分に分かれます。

[a] 翻訳部分です。プロジェクトに登録しているCOBOLソースの数だけ出力します。

[b] リンクエディット部分です。“〔ビルド用JCL/CLIST生成〕ダイアログ”で、“リンクエディット”を指定した場合に出力します。

図中に赤字で示した部分がプロジェクトの情報とファイルの送信情報に依存して生成する部分です。以下、その詳細を説明します。

[1] JOB文です。ジョブ名、課金情報その他を必要に応じて修正します。

[2] 翻訳オプションです。

[3] オブジェクトモジュールの指定です。翻訳オプション“NOOBJECT”が指定されている場合は生成しません。なお、スタブまたは出口プログラムのJCLの場合、次のようになります。

システム	生成する情報
MSP	//SYSLIN DD DSN=&&LOADSET,UNIT=SYSDA,DISP=(NEW,PASS), // SPACE=(80,(500,100)),DCB=BLKSIZE=800
XSP	¥ FD RLIB=DA,VOL=WORK,FILE=(/,AD),BLK=(80*500,100), FCB=BLKSIZE=3200,DISP=CONT

[4] AADアプリケーション用のCOBOL登録集の指定です。“〔ビルド用JCL/CLIST生成〕ダイアログ”の、“Interstage用のロードモジュールを生成する”を指定した場合に生成します。

[5] OSIV系システム上での登録集の指定です。登録集ファイルの送信の有無、翻訳オプション“LIB”の指定により次のように生成します。

プロジェクトの状態	生成	生成する情報
登録集の送信済み	○	“〔送信〕ダイアログ”の〔ファイルの種別〕で“登録集”を選択したときに設定した送信先ファイル名、パスワードおよびVOL通番を使用します。パスワードおよびVOL通番の設定がない場合、パスワードおよびVOL通番の指定は生成しません。
登録集の送信なしで LIBオプション指定	○	登録集格納ファイル名を“SAMPLE.COBOL”として生成するので、適切な名前に変更します。
上記以外	×	なし

[6] OSIV系システム上でのソースファイルの指定です。プロジェクトに登録しているCOBOLソースの、パス名と拡張子を除いた部分を区分編成ファイルのメンバ名として使用します。生成する区分編成ファイル名は、ソースファイルの送信の有無によって、次のように決められます。

プロジェクトの状態	生成	生成する情報
ソースファイル 送信済み	○	“〔送信〕ダイアログ”の〔ファイルの種別〕で“ソース”を選択したときに設定した送信先ファイル名、パスワードおよびVOL通番を使用します。パスワードおよびVOL通番の設定がない場合、パスワードおよびVOL通番の指定は生成しません。
ソースファイル 送信未	○	ソースファイルを格納する区分編成ファイル名を“SAMPLE.COBOL”として生成するので、適切な名前に変更します。

[7] リンクオプションの指定です。 “〔ビルド用JCL/CLIST生成〕ダイアログ”で、“Interstage用のロードモジュールを生成する”を指定した場合、AADアプリケーションを作成するためのリンクオプションの指定を生成します。その他に必要なリンクオプションがある場合は修正してください。

[8] ロードモジュール格納ファイルの指定です。“SAMPLE.LOAD”の名前で生成するので、適切な名前に変更します。

[9] リンクエディット制御文の指定です。スタブまたは出口プログラムのJCLの場合、次のようになります。

システム	生成する情報
MSP	生成しません。
XSP	¥ FD COIN=/,SW=RLIB

[10]入口名の指定です。“ENTRY1”の名前で生成するので、実際の入口名に変更します。なお、“〔ビルド用JCL/CLIST生成〕ダイアログ”で、“Interstage用のロードモジュールを生成する”を指定した場合には、生成しません。

[11]ロードモジュール名の指定です。プロジェクトで指定した最終ターゲットファイル名から拡張子を除いた名前で生成します。スタブまたは出口プログラムの場合、生成しません。

[12]スタブまたは出口プログラムの場合、生成しません。

5.2.3.2 CLISTの雛型とその修正

以下にOSIV MSPおよびOSIV XSPそれぞれのシステムに対して、生成されるCLISTの雛型を示します。

図5-19 OSIV MSP用に生成されるCLISTの雛型

```
PROC 0
  CONTROL  LIST
  CRTFILE SAMPLE.OBJ PARTITIONED(10) BLOCKS SPACE(500 100) -
        FORMAT(FIXED) RLENGTH(80) BLENGTH(3200)
  ALLOC F(CORBA) DA('AIM1.AADCPLIB') SHR            …[3]
  COBOL 'ソース格納ファイル名(ファイル名)' -          …[5]
    OBJECT(SAMPLE.OBJ(ファイル名)) -                 …[2]
    LIB('登録集格納ファイル名')  -                   …[4]
    PRINT(*) -
    翻訳オプション…                                  …[1]
  LINK (SAMPLE.OBJ(ファイル名)) -                    …[6]
    LOAD(SAMPLE.LOAD(ロードモジュール名)) -           …[7] [10]
    PRINT(*) -
    NOLIBDD -
    COBLIB -
    ENT(ENTRY1) -                                    …[9]
    リンクエディットオプション…                      …[8]
  FREE DA(SAMPLE.OBJ)
  DLTFILE SAMPLE.OBJ
END
```

(a)

(b)

生成されるJCLの雛型は大きく2つの部分に分かれます。

[a]翻訳部分です。プロジェクトに登録しているCOBOLソースの数だけ出力します。

[b]リンクエディット部分です。“〔ビルド用JCL/CLIST生成〕ダイアログ”で、“リンクエディット”を指定した場合に出力します。

図中に赤字で示した部分がプロジェクトの情報とファイルの送信情報に依存して生成する部分です。以下、その詳細を説明します。

[1] 翻訳オプションです。

[2] オブジェクトモジュールの指定です。翻訳オプション“NOOBJECT”が指定されている場合は生成しません。

[3] AADアプリケーション用のCOBOL登録集の指定です。“〔ビルド用JCL/CLIST生成〕ダイアログ”の、“Interstage用のロードモジュールを生成する”を指定した場合に生成します。

[4] OSIV系システム上での登録集の指定です。登録集ファイルの送信の有無、翻訳オプション“LIB”の指定により次のように生成します。

プロジェクトの状態	生成	生成する情報
登録集の送信済み	○	“〔送信〕ダイアログ”の〔ファイルの種別〕で“登録集”を選択したときに設定した送信先ファイル名を使用します。
登録集の送信なしでLIBオプション指定	○	登録集格納ファイル名を“SAMPLE.COBOL”として生成するので、適切な名前に変更します。

上記以外	×	なし

[5] OSIV系システム上でのソースファイルの指定です。プロジェクトに登録しているCOBOLソースの、パス名と拡張子を除いた部分を区分編成ファイルのメンバ名として使用します。生成する区分編成ファイル名は、ソースファイルの送信の有無によって、次のように決められます。

プロジェクトの状態	生成	生成する情報
ソースファイル 送信済み	○	“〔送信〕ダイアログ”の〔ファイルの種別〕で“ソース”を選択したときに設定した送信先ファイル名を使用します。
ソースファイル 送信未	○	ソースファイルを格納する区分編成ファイル名を“SAMPLE.COBOL”として生成するので、適切な名前に変更します。

[6] リンクするオブジェクトファイル名の指定です。翻訳したソースファイルの数分生成されます。

[7] ロードモジュール格納ファイルの指定です。“SAMPLE.LOAD”の名前で生成するので、適切な名前に変更します。

[8] リンクオプションの指定です。 “〔ビルド用JCL/CLIST生成〕ダイアログ”で、“Interstage用のロードモジュールを生成する”を指定した場合、AADアプリケーションを作成するためのリンクオプションの指定を生成します。その他に必要なリンクオプションがある場合は修正してください。

[9] 入口名の指定です。“ENTRY1”の名前で生成するので、実際の入口名に変更します。XSPの場合は、この部分が次のように展開されます。

```
    ENTRY(ENTRY1) -
    REP -
```

なお、“〔ビルド用JCL/CLIST生成〕ダイアログ”で、“Interstage用のロードモジュールを生成する”を指定した場合には、生成しません。

[10] ロードモジュール名の指定です。プロジェクトで指定した最終ターゲットファイル名から拡張子を除いた名前で生成します。スタブまたは出口プログラムの場合、生成しません。

5.3 ターゲットビルド

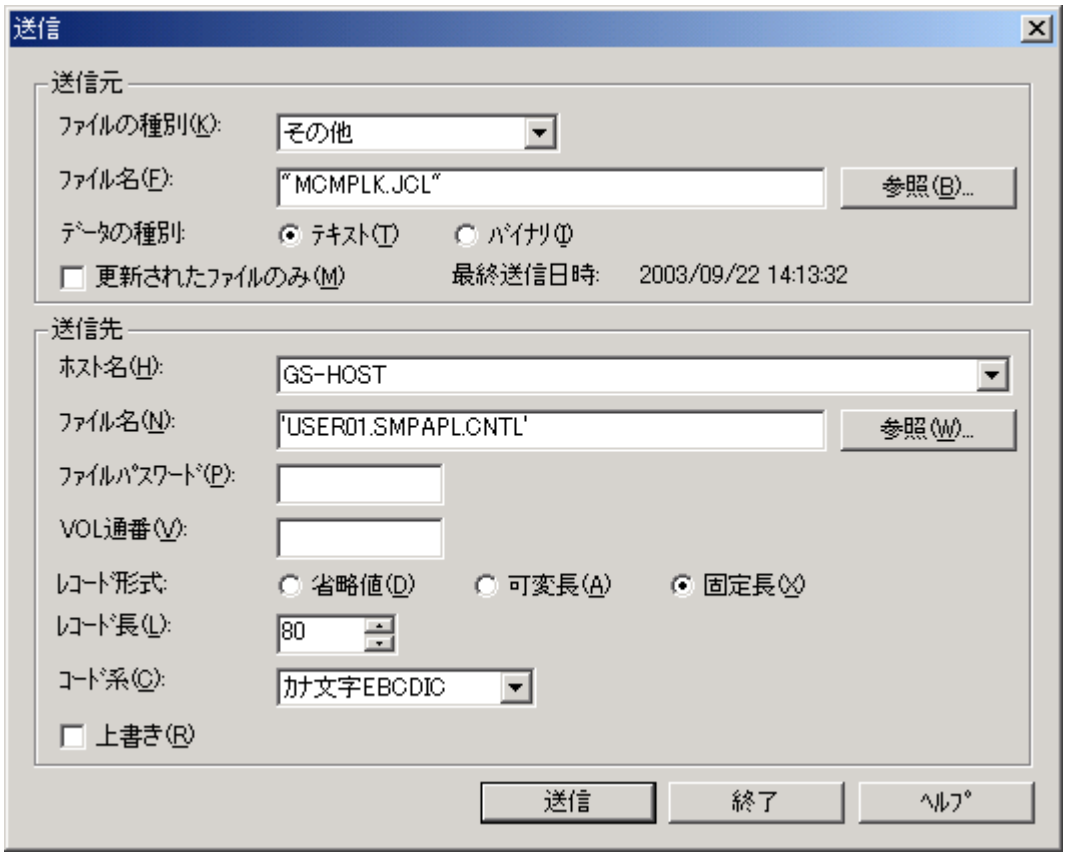
ターゲットビルド機能は、分散開発によって開発し、OSIV系システムに登録したプログラム資産をOSIV系システム上で、翻訳・リンクする機能です。
翻訳・リンクに必要なビルド制御文(JCLまたはCLIST)は、ビルド制御文生成機能によって生成されたものを使用することができます。

5.3.1 OSIV系システムへの送信

ビルド制御文生成機能によって生成したビルド制御文(JCLまたはCLIST)は、NetCOBOLのファイル送信の機能を使用して、OSIV系システムに送信します。
基本的な手順は、COBOLソース・登録集原文を送信する場合と同じです。ここでは、異なる設定が必要となる項目のみ説明します。送信の手順の詳細については“5.1.1 COBOLソース/登録集の登録”を参照してください。

1. 〔送信元〕は次のように指定してください。
   - ファイルの種別:
     必ず“その他”を選択します。
   - データの種別:
     必ず“テキスト”を選択します。
2. 〔送信先〕は次のように指定してください。
   - ファイル名:
     送信先の区分編成ファイルの名前を完全修飾名で1つだけ指定します。メンバ名を指定すること、ファイル名を複数指定することはできません。
   - レコード形式:
     必ず“固定長”を選択します。
   - ―レコード長:
     必ず“80”を指定します。
3. その他の項目については、必要に応じて指定します。
4. 以上の設定が済んだら、〔送信〕ボタンをクリックして、ファイルの送信処理を開始します。

図5-20 JCLをOSIV系システムに送信する際の設定の例



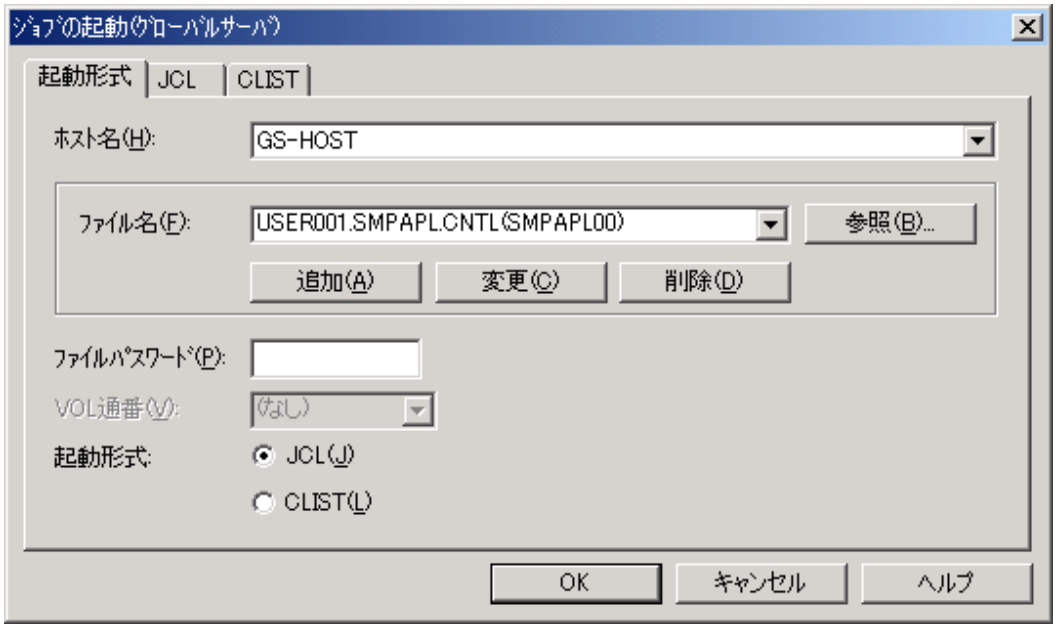
5.3.2 ターゲットビルドの実行

送信したビルド制御文(JCLまたはCLIST)を使用して、COBOLプロジェクトマネージャからOSIV系システムでの翻訳・リンクのジョブを実行します。

以下、ジョブの実行の手順について説明します。

1. 〔プロジェクト〕-〔分散開発〕メニューで“ターゲットビルド”を選択します。〔ジョブの起動(OSIV系システム)〕ダイアログが表示されます。
2. 〔起動形式〕ページの各項目を設定します。
   - ファイル名:送信したJCL/CLISTファイル名を指定します。
   - 起動形式:送信したビルド制御文の種類にあわせて、“JCL”または“CLIST”を選択します。

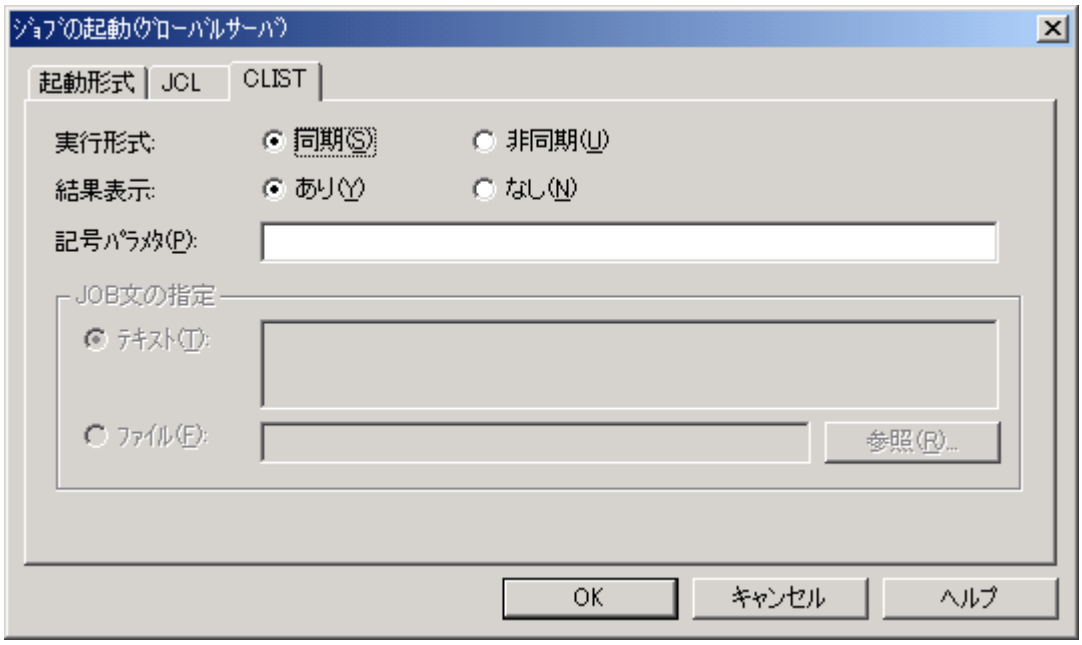
図5-21 〔ジョブの起動(OSIV系システム)〕ダイアログ（〔起動形式〕ページ）の設定例



3. 送信したビルド制御文がJCLの場合、特に〔JCL〕ページの設定は必要ありません。〔JCL〕ページの設定が必要となるのは、次のような場合です。
   — CLISTをバッチTMPを利用して、バッチ起動する場合（非同期型での実行）。
   — CLISTとJOB文を別けて記述し、実行時にJOB文とJCLを結合する場合。

   このような場合の、〔JCL〕ページの設定については“PowerGEM Plus開発マネージャ”のヘルプ、“ジョブ起動の定義”参照してください。
4. 送信したビルド制御文がCLISTの場合、必要に応じて〔CLIST〕ページの各項目を設定します。
   — 実行形式:
     コマンド形式で実行するか(同期)、バッチTMPを利用して、バッチ形式で実行するか(非同期)を選択します。
   — 結果表示:
     “あり”を指定するとビルド結果を確認することができます。
   — 起動パラメタ:
     CLISTに渡すパラメタがあるときに指定します。パラメタは各パラメタを引用符で囲み、１２６文字以内の英数字で記述します。
     例:　’NUMBER MEMLIST’

図5-22 〔ジョブの起動(OSIV系システム)〕ダイアログ（〔CLIST〕ページ）の設定例

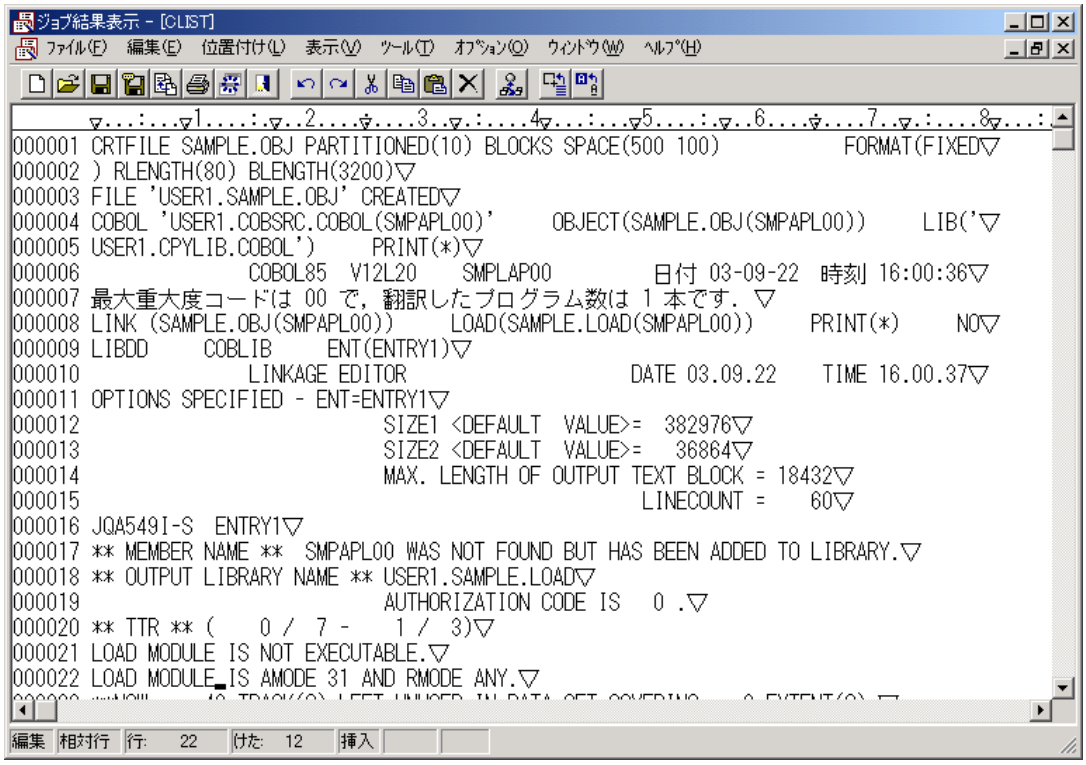


5. 必要な設定が完了したら、〔OK〕ボタンをクリックするとターゲットビルドが開始されます。
6. ジョブの起動に使用したビルド制御文がJCLである場合、次のようなメッセージボックスでジョブの起動が通知されるだけで、ジョブの実行結果は帰ってきません。

図5-23 JCLによるジョブ起動の確認メッセージ

7. ジョブの起動に使用したビルド制御文がCLISTである場合、ジョブの終了後、エディタが起動して、ジョブの実行結果が表示されます。

図5-24 CLISTによるリモートビルドの実行結果の表示



第6章 CORBAアプリケーションの分散開発

ここまで、説明してきたNetCOBOLの分散開発のための機能は、CORBAアプリケーションを開発する場合でも有効です。その上で、NetCOBOLはCORBAアプリケーションの開発に特化した機能をいくつか用意しています。
ここでは、それについて説明します。

6.1 OSIV系のCORBAアプリケーション

CORBA(Common Object Request Broker Architecture)は、OMG(Object Management Group)が提唱しているオブジェクト指向の分散処理環境のアーキテクチャで、OSや開発言語、プラットホームに依存しない相互接続を実現します。このCORBAに準拠したアプリケーションをCORBAアプリケーションと呼びます。
OSIV系システムでも、INTERSTAGE/AIMApplicationDirector配下で動作するアプリケーションとして、CORBAアプリーションを実現可能です。
OSIV系システムで動作するCORBAアプリーションは、機能的な違いからではなく、その開発スタイルから大きく2種類にわけられます。

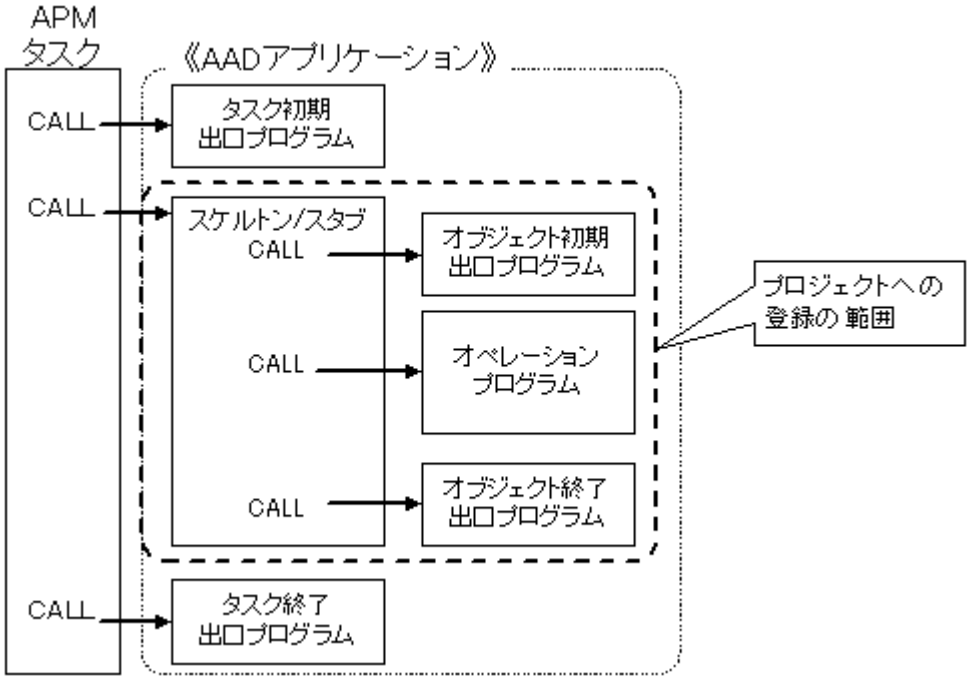
- AIMApplicationDirectorアプリケーション(以降、AADアプリケーション)
  UNIXサーバおよびPCサーバと共通のAPIを利用して、Interstageスタイルで開発したアプリケーションです。
  必要となるツールの一部がOSIV系システムには存在しないため、OSIV系システムで動作するCORBAアプリケーションの開発は常に分散開発の形態で行う必要があります。
- AIMアプリケーション
  従来のAIM表示ファイルインタフェースを利用して、開発したアプリケーションです。
  開発には従来からの手法とツールが使用できますから、分散開発は必須ではありません。

開発スタイルの違いから、それぞれの開発で必要な手順、ツール等も異なります。このそれぞれに対して、NetCOBOLが分散開発の支援のために提供している機能を説明します。

6.2 AADアプリケーションの開発

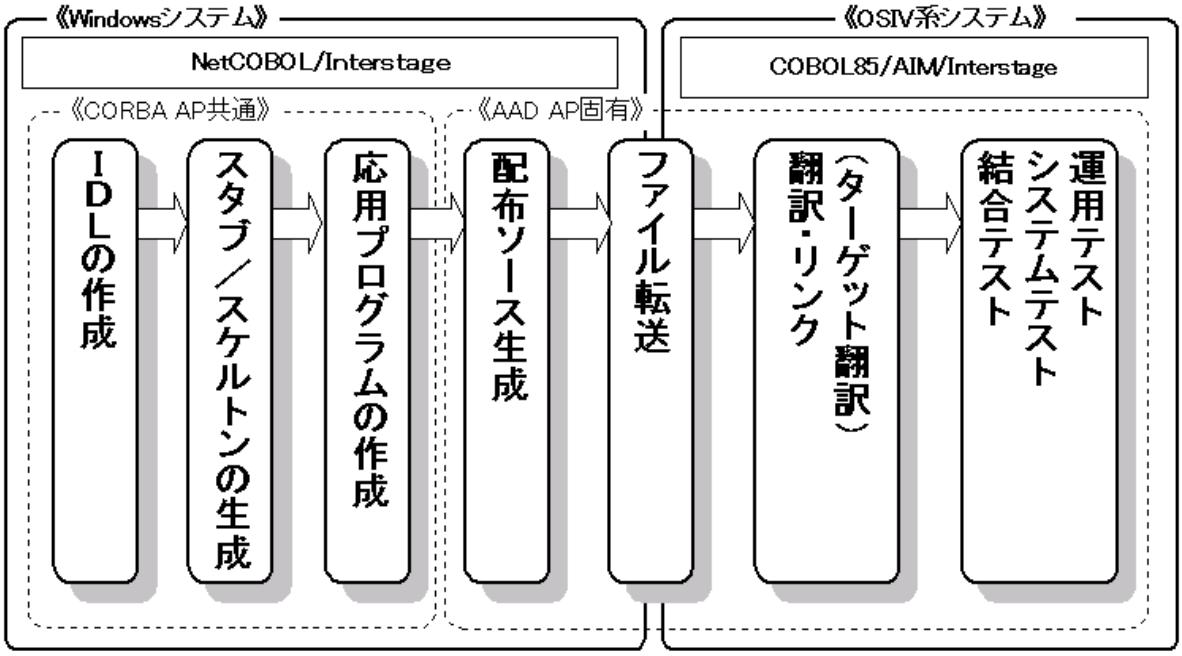
一般的にAADアプリケーションとは次のような構成を持ちます。

図6-1 AADアプリケーションの構成



その開発手順は “図: AADアプリケーション開発の流れ”に示すものとなります。

図6-2 AADアプリケーション開発の流れ

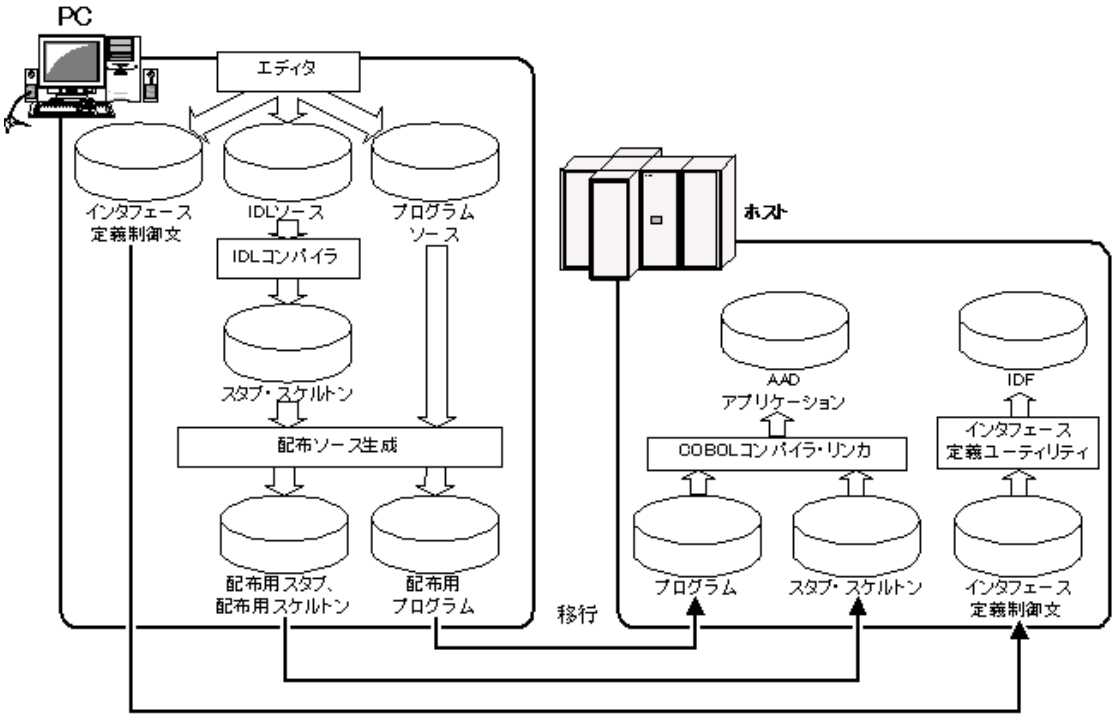


AADアプリケーションの開発は、Interstageスタイルで行うため、PC/UNIXサーバ向けのCORBAアプリケーションとおおまかには同じ手順を踏みます。一方、OSIV系システムで動作するプログラ

ムであるため、プログラム資産の流通などのAADアプリケーションに固有の手順も存在します。

以下にAADアプリケーション開発時の資産の流れを示します。

図6-3 AADアプリケーションの作成



プロジェクトマネージャは以下の処理を行うことにより、AADアプリケーションの作成を支援します。

- 〔Interstage〕ダイアログでプロジェクトにCORBAアプリケーション固有の設定をします。IDLコンパイラを使用することでIDLソースファイルを翻訳し、生成したスケルトンファイルまたはスタブファイルをプロジェクトに自動登録するなどの機能を含みます。
- 配付ツールを使用することで、PC上の資源をグローバルサーバで使用可能な資源(これを配付ソースと呼びます)に変換し、配付ソースをプロジェクトに登録します。
- PC上のファイルをグローバルサーバ上に転送します。
- グローバルサーバ上で翻訳、リンクするためのJCLまたはCLISTの雛型を生成します。
- グローバルサーバ上のジョブを起動し、その結果を表示します。
- プログラムの修正を容易にするため、配付ソース生成前のファイル名およびプログラム名と、配付ソース生成後の名前の対応を表示します。

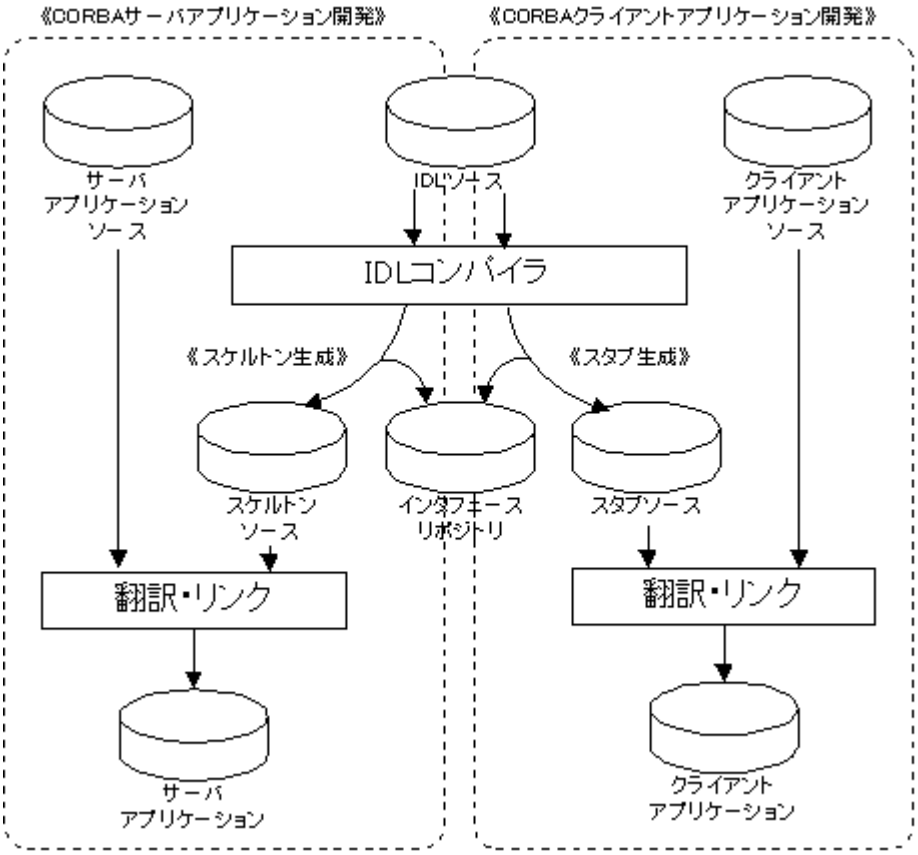
PC上のファイルをグローバルサーバに送信し、翻訳・リンク用のJCLまたはCLISTを生成して、グローバルサーバ上で翻訳・リンクのジョブを実行するまでの部分は“第5章 サーバ連携機能”を参照してください。ここでは、AADアプリケーション開発手順の概観を示し、その後でそれを支援するためにNetCOBOLの提供する機能について説明します。
なお、AADアプリケーション開発の詳細については、“INTERSTAGEシステム開発手引書(AIM連携)”を参照してください。

6.2.1 NetCOBOLでのAADアプリケーションの開発手順

AADアプリケーションではCORBAアプリケーション間のインタフェースを定義したIDLソースを必要とします。このIDLソースはサーバ側のCORBAアプリケーションを作成する場合とクライアント側のCORBAアプリケーションを作成する場合でそれぞれ次のように使われます。

- サーバ側のCORBAアプリケーションの作成
  IDLコンパイラにより、IDLソースを翻訳してスケルトンソースを生成します。
- クライアント側のCORBAアプリケーションの作成
  IDLコンパイラにより、IDLソースを翻訳してスタブソースを生成します。

図6-4 IDLソースのCORBAアプリケーション開発への使われ方



NetCOBOLのCOBOLプロジェクトマネージャは、以下の処理を自動で行うことにより、CORBAアプリケーションの作成を支援します。

1. 利用者がIDLソースファイルを指定することによって、生成されるスケルトンファイルまたはスタブファイルのファイル名を類推し、プロジェクトの依存関係に登録します。
2. IDLコンパイラを使用してIDLソースファイルを翻訳し、COBOLインタフェースによるスケルトンファイルまたはスタブファイルを生成します。
3. スケルトンファイルまたはスタブファイルをCOBOLコンパイラで翻訳し、利用者アプリケーションとリンクします。

6.2.1.1 サーバアプリケーションの開発

サーバアプリケーションを作成する場合、IDLソースファイルの運用方法によってプロジェクトの構成が変わります。

IDLソースファイルを登録する場合

IDLソースファイルをプロジェクトに登録するパターンです。AADアプリケーションの作成は、以下の手順で行います。

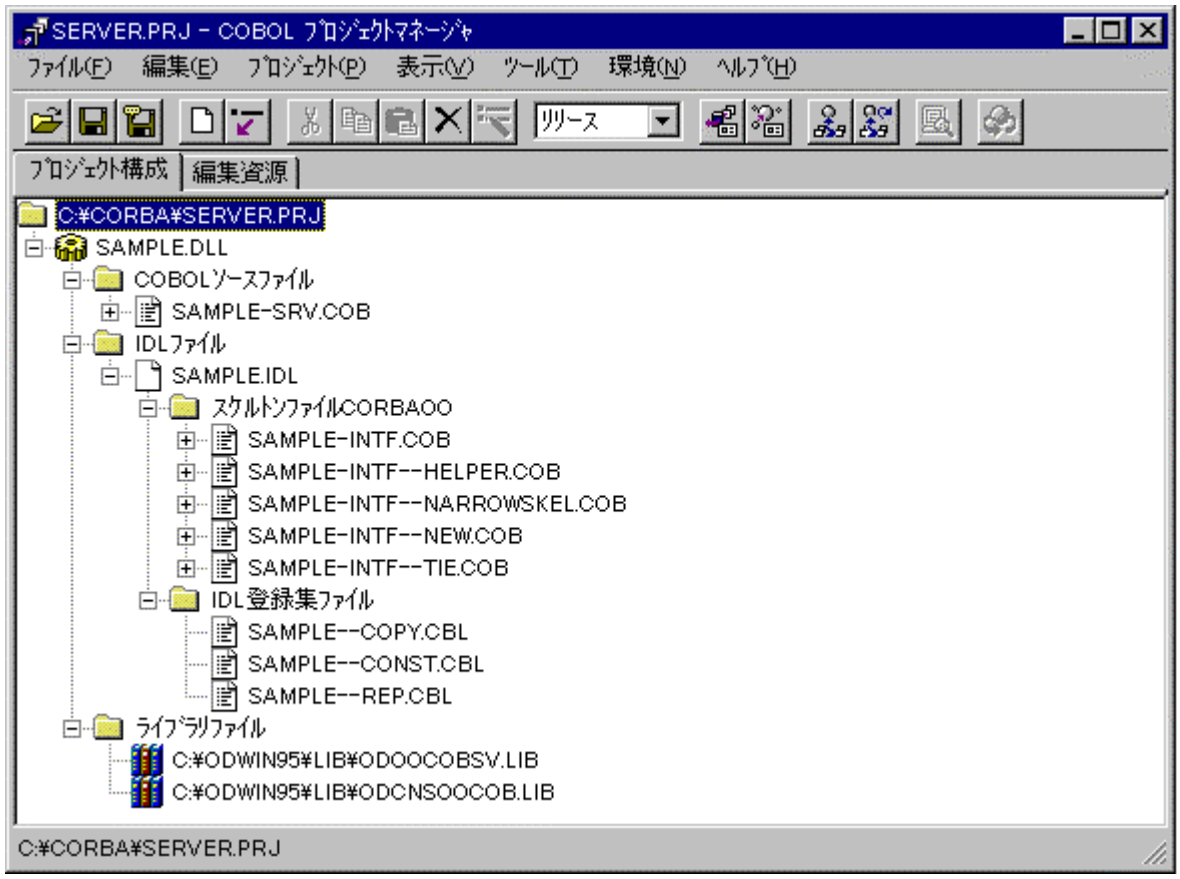
1. 最終ターゲットファイルをプロジェクトに登録します。
   最終ターゲットファイルは以下の要素から作成されます（“図6-1 AADアプリケーションの構成”を参照)。

— スケルトンおよびオペレーション処理プログラム
— 出口プログラム
— スタブ

2. IDLソースファイルを登録します。
   最終ターゲットファイルを選択し、〔編集〕-〔フォルダ作成〕メニューから“IDLファイル”を選択して、IDLファイルのフォルダを登録します。
   IDLファイルのフォルダに対してIDLソースファイルを〔編集〕メニューの“新規作成”または“追加”でプロジェクトに登録します。

   注意
   IDLソースファイルはプロジェクトファイルと同じフォルダに格納してください。
3. 〔Interstage〕ダイアログでプロジェクトにCORBAアプリケーション固有の設定をし、依存関係を作成します。〔Interstage〕ダイアログについては“6.2.2.1 〔Interstage〕ダイアログ”を参照してください。

図6-5 IDLソースファイルを登録したサーバアプリケーション開発プロジェクト例



4. COBOLソースファイルをプロジェクトに登録します。
5. その他の必要なファイルを登録します。Windows系システムで単体テストまで実施するのであれば、Interstage Application Serverの提供するライブラリファイルを登録します。詳細については“Interstage Application Server アプリケーション作成ガイド (CORBAサービス編)”を参照してください。
6. 配付ソースを生成します。生成された配付ソースファイルはプロジェクトマネージャのツリーに登録されます。配付ソースの生成については“6.2.2.2 配付ソース生成”を参照してください。
7. インタフェース定義制御文を登録します。
   最終ターゲットファイルを選択し、〔編集〕-〔フォルダ作成〕メニューから“その他”を

選択して、その他のフォルダを登録します。その他のフォルダに対してインタフェース定義制御文を〔編集〕メニューの“追加”でプロジェクトに登録します。

インタフェース定義制御文の詳細については、“INTERSTAGEシステム開発手引書(AIM連携)”を参照してください。

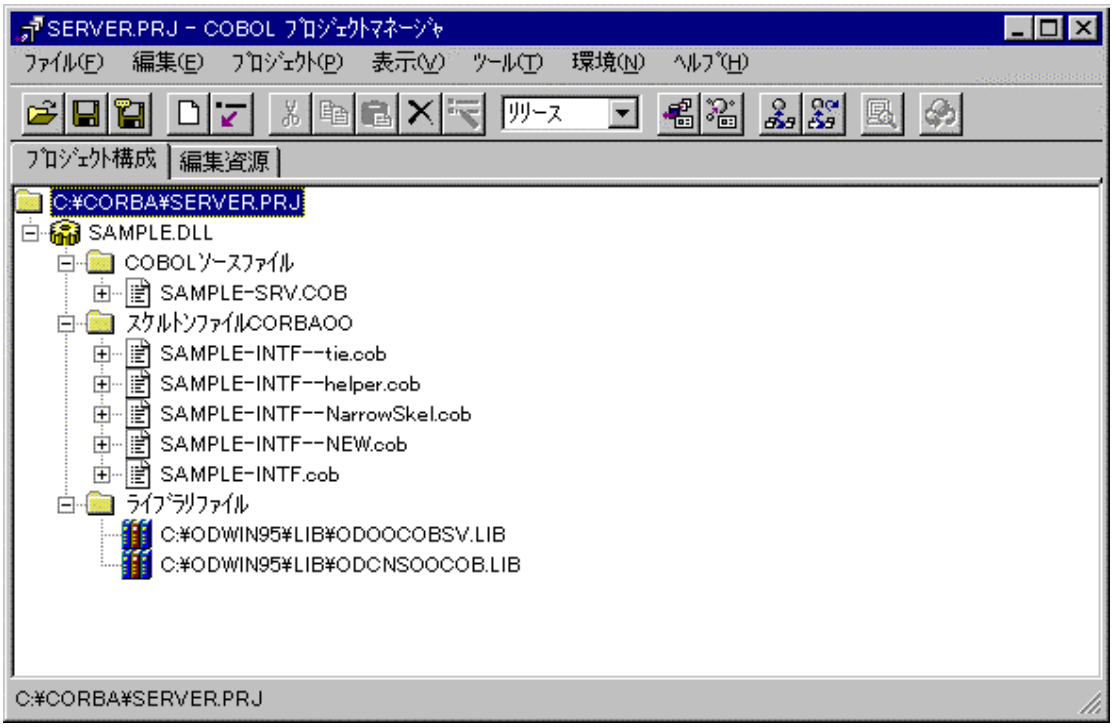
8. 資産をグローバルサーバへ送信します。
9. ロードモジュールを作成するためのJCLまたはCLISTの雛型ファイルを生成します。
   → プロジェクトマネージャのツリーに、JCL/CLISTの雛型ファイルが登録されます。
10. グローバルサーバの環境に合わせて、JCL/CLISTの雛型ファイルを修正します。
11. JCL/CLISTをグローバルサーバ上へ転送します。
12. JCL/CLISTを起動し、ロードモジュールを作成します。
13. 配付ソース生成前のファイル名およびプログラム名と、配付ソース生成後の名前の対応を表示して、プログラムの修正をします。[参照]“6.2.2.3 配付ソース対応表示”

IDLソースファイルを登録しない場合

IDLソースファイルをプロジェクトに登録しないパターンです。この場合、利用者はプロジェクト外で必要に応じてIDLコンパイルを実行しなければなりません。AADアプリケーションの作成は、以下の手順で行います。

1. 最終ターゲットファイルを登録します。
   最終ターゲットファイルは以下の要素から作成されます（“図6-1 AADアプリケーションの構成”を参照)。
   - スケルトンおよびオペレーション処理プログラム
   - 出口プログラム
   - スタブ
2. 〔Interstage〕ダイアログでプロジェクトにCORBAアプリケーション固有の設定をし、依存関係を作成します。〔Interstage〕ダイアログについては“6.2.2.1 〔Interstage〕ダイアログ”を参照してください。
3. COBOLソースファイルをプロジェクトに登録します。以降の作業は、IDLソースファイルを登録する場合の5.以降と同じです。

図6-6 IDLソースファイルを登録しないサーバアプリケーション開発プロジェクト例



6.2.1.2 クライアントアプリケーションの開発

クライアントアプリケーションは、サーバアプリケーションと同じ要領で作成できます。
クライアント側でIDLソースをもとにスタブファイルを生成する場合は、“6.2.1.1 サーバアプリケーションの開発”の“IDLソースファイルを登録する場合”を参照してください。
また、サーバ側からスタブファイルが提供される場合は、“6.2.1.1 サーバアプリケーションの作成”の“IDLソースファイルを登録しない場合”を参照してください。このとき、説明中の“スケルトンファイル”は“スタブファイル”に読み替えてください。

6.2.2 AADアプリケーション開発支援機能

ここでは、NetCOBOLがAADアプリケーション開発を支援するために提供している機能について説明します。

6.2.2.1 〔Interstage〕ダイアログ

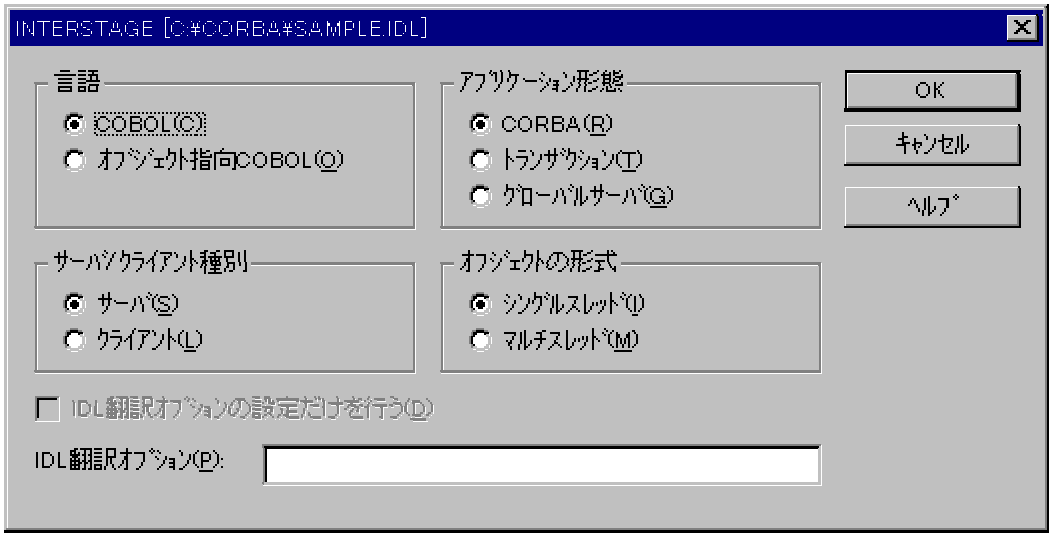
〔Interstage〕ダイアログは、CORBAアプリケーションに固有の情報を設定し、構成ファイル間の依存関係を自動的に作成するものです。CORBAアプリケーションを作成する場合は、必ずこのダイアログを使用する必要があります。
〔Interstage〕ダイアログの機能と構成は、プロジェクト内にIDLソースファイルを登録する場合と登録しない場合で、次のように異なります。

- IDLソースファイルを登録する場合
  登録したIDLソースファイルからスタブファイルまたはスケルトンファイルを生成して使用する場合です。IDLコンパイラの起動から生成したスタブファイルまたはスケルトンファイルのプロジェクトへの登録までが自動で行われます。
- IDLソースファイルを登録しない場合
  プロジェクト外で作成されたスタブファイルまたはスケルトンファイルを使用する場合です。指定したファイルをスタブファイルまたはスケルトンファイルとして、プロジェクトに追加します。

IDLソースファイルを登録する場合

プロジェクトにIDLソースファイルを登録してある状態で、〔編集〕メニューから“Interstage”を選択すると、次の〔Interstage〕ダイアログが表示されます。

図6-7 〔Interstage〕ダイアログ(IDLソースファイル登録時)



以下、各項目の設定方法について説明します。

1. 言語:
   IDLソースファイルをコンパイルする際に使用するインタフェースを指定します。
   AADアプリケーションの開発時には“COBOL”(プログラムインタフェース使用)を選択してください。
2. アプリケーションの形態:
   作成するCORBAアプリケーションの形態を指定します。この指定によりIDLコンパイルに使用されるIDLコンパイラが切り替わります。
   AADアプリケーションの開発時には“グローバルサーバ”(aadidlcコマンド使用)を選択してください。
3. サーバ/クライアント種別:
   作成するAADアプリケーションがサーバ/クライアント種別を指定します。
   - サーバ:
     サーバアプリケーションを作成することを指定します。IDLソースの翻訳の結果、スケルトンファイルが生成されます。
   - クライアント:
     クライアントアプリケーションを作成することを指定します。IDLソースの翻訳の結果、スタブファイルが生成されます。
4. オブジェクトの形式:
   生成するCOBOLのオブジェクトをシングルスレッドでのみ動作可能として生成するか、マルチスレッドでも動作可能として生成するかを指定します。AADアプリケーションの開発時には特に意味を持ちません。“シングルスレッド”を指定しておいてください。
5. IDL翻訳オプション:
   IDLコンパイル時の翻訳オプションを指定します。通常は指定不要です。詳細については“INTERSTAGEリファレンスガイド(AIM連携)”を参照してください。

以上を設定して、〔OK〕ボタンをクリックすると、以下の事が行われます。

- IDLソースの翻訳とスタブまたはスケルトンファイルの生成
- 生成されたスタブまたはスケルトンファイルのプロジェクトへの登録
- 依存関係の更新

IDLソースファイルを登録しない場合

プロジェクトにIDLソースファイルを登録していない状態で、〔編集〕メニューから“Interstage”を選択すると、次の〔Interstage〕ダイアログが表示されます。

図6-8 〔Interstage〕ダイアログ(IDLソースファイル未登録)

各項目の設定方法について、“スケルトン／スタブファイルを取り込む”の部分を除いて同じです。“スケルトン／スタブファイルを取り込む”の部分についてのみ説明します。

1. スケルトン／スタブファイル名:
   プロジェクトに追加するスケルトン／スタブファイル名を指定します。〔参照〕ボタンをクリックすることで〔ファイルの参照〕ダイアログを開いて、ファイル名を選択することもできます。
2. 一覧:
   プロジェクトに追加するスケルトン／スタブファイル名の一覧を表示します。
   〔スケルトン／スタブファイル名〕にファイル名を指定して、〔追加〕ボタンをクリックするとそのファイル名が追加されます。また、一覧中でファイル名を選択して、〔削除〕ボタンをクリックすると選択したファイル名が一覧から削除されます。

以上を設定して、〔OK〕ボタンをクリックすると、以下の事が行われます。

- 指定されたスタブまたはスケルトンファイルのプロジェクトへの登録
- 依存関係の更新

6.2.2.2 配付ソース生成

IDLコンパイラによって生成されたスタブファイルやスケルトンファイルは、ファイル名の命名規則その他がOSIV系システムの規約に反しているため、そのままではOSIV系システムに移行することができません。配付ソース生成とは、このシステム間の違いを認識して、Windows系システム上で作成した資源をOSIV系システムで使用するための資源に変換する操作です。

プロジェクトに登録している、次のファイルから配付ソースを生成することができます。

- COBOLソースファイル
- 登録集ファイル
- スケルトンファイルAADフォルダ配下のスケルトンファイル
- スタブファイルAADフォルダ配下のスタブファイル
- IDL登録集ファイルAADフォルダ配下のインタフェース定義用登録集ファイル

生成した配付ソースファイルは、プロジェクトファイルを格納したフォルダのサブフォルダに出力します。

また、配付ソース生成機能については、“INTERSTAGEシステム開発手引書(AIM連携)”も参照してください。

配付ソース生成機能の説明

配付ソース生成機能では、対象となるファイルに以下の操作を行って配付ソースを生成します。

- ファイル名の変換
- プログラム名の変換
- COPY文で指定する登録集のメンバ名(原文名)の変換
- COBOLソース内におけるCOMP-5(COMPUTATIONAL-5)指定のBINARY指定への変換

なお、変換の前後の名前をこのマニュアルではロングネームおよびショートネームと呼びます。

この名前の変換の規則は、次のようにして決定されます。

表6-1 配付ソース生成時の名前変換規則の決定方法

入力ファイル	規則の適用対象	必要性
IDLソースファイル スケルトンプレフィクス一覧 スタブプレフィクス一覧	システム提供ファイル名 システム提供プログラム名 IDLコンパイラが生成したファイル名 IDLコンパイラが生成したプログラム名	必須
ユーザファイル名対応表 ユーザプログラム名対応表	上記以外でWindows系システム上とOSIV系システム上で名前を変更する必要があるもの	任意

以下、配付ソース生成における名前の変換規則決定のための入力ファイルについて説明します。

スケルトンプレフィクス一覧

この一覧は、IDLソース単位に生成されるロングネームをショートネームに変換するための一覧です。配付ソース生成機能を使用する場合に名称対応表を生成する時に使用します。開発者個々人で作成するのではなく、OSIV系システムにおいて、開発資産を管理する責任者が作成するべきです。

- 形式

  CSV形式のテキストファイル。ファイル名“aad-skpre.txt”で作成します。

- 内容

  AIMディレクトリに対応して1つのスケルトンプレフィクス一覧を作成します。

  IDLのインタフェースごとに、以下の名称を “,”で区切って記述します。以下の名称と“,”の間に空白やタブなどの区切り文字を記述することはできません。

  — オブジェクト名

  IDLのモジュール名とインタフェース名を“-”でつないだ文字列をダブルクォーテーションで囲みます。

  — スケルトンプレフィクス

  6文字以内の英字で始まる英数字の文字列を、大文字で指定します。

  スケルトンプレフィクス一覧にはコメントを記述することができます。1カラム目と2カラム目に“/”を記述した場合、当該行全体がコメントの扱いとなります。ま

た、スケルトンプレフィクスの直後に1文字の空白を記述した場合、当該行においてその空白以降から改行またはファイルの最終位置までの文字列がコメントの扱いとなります。コメントには任意の文字が指定可能です。

- 留意事項
  - スケルトンプレフィクス一覧に複数のスケルトンプレフィクスを指定する場合は、プレフィクスを一意に指定してください。
  - スタブプレフィクスとスケルトンプレフィクスには、異なるプレフィクスを指定してください。
- 例

以下、スケルトンプレフィクス一覧の記述例を示します。

```
//スケルトンプレフィクス一覧
"mod1-int1",KLMN モジュールmod1 インタフェースint1用
"mod2-int2",KLMA モジュールmod2 インタフェースint2用
"mod3-int3",KLMB モジュールmod3 インタフェースint3用
```

スタブプレフィクス一覧

この一覧は、IDLソース単位に生成されるロングネームをショートネームに変換するための一覧です。配付ソース生成機能を使用する場合に名称対応表を生成する時に使用します。開発者個々人で作成するのではなく、OSIV系システムにおいて、開発資産を管理する責任者が作成するべきです。

- 形式

  CSV形式のテキストファイル。ファイル名 “aad-stpre.txt”で作成します。
- 内容

  AIMディレクトリに対応して1つのスタブプレフィクス一覧を作成します。

  スタブプレフィクス一覧にはIDLソースファイルごとに、以下の名称を“,”で区切って記述します。以下の名称と“,”の間に空白やタブなどの区切り文字を記述することはできません。
  - IDLソースファイル名

    IDLソースファイルの名前(拡張子“.idl”を除く)をダブルクォーテーションで囲みます。左端のダブルクォーテーションは行の1カラム目に記述してください。
  - スタブプレフィクス

    6文字以内の英字で始まる英数字の文字列を、大文字で記述します。

    スタブプレフィクス一覧にはコメントを記述することができます。1カラム目と2カラム目に“/”を記述した場合、当該行全体がコメントの扱いとなります。また、スタブプレフィクスの直後に1文字の空白を記述した場合、当該行においてその空白以降から改行またはファイルの最終位置までの文字列がコメントの扱いとなります。コメントには任意の文字が指定可能です。
- 留意事項
  - スタブプレフィクス一覧に複数のスタブプレフィクスを指定する場合は、プレフィクスを一意に指定してください。
  - スタブプレフィクスとスケルトンプレフィクスとは、異なるプレフィクスを指定してください。
- 例

以下、スタブプレフィクス一覧の記述例を示します。

```
//スタブプレフィクス一覧
"sample",ABCD IDLソースファイルsample用
"sampleidl001",AA01 IDLソースファイルsampleidl001用
"sampleidl002",AA02 IDLソースファイルsampleidl002用
```

ユーザファイル名対応表

- 形式
  CSV形式のテキストファイル。ファイル名“aad-usrfnm.txt”で作成します。
- 内容
  ダブルクォーテーションで囲んだInterstage応用プログラムが格納されたファイル名と、当該ファイル名に対応する任意の名称(英字で始まる8文字の英数字)を“,”で区切ったテキストファイルです。
  ユーザファイル名対応表にはコメントを記述することができます。1カラム目と2カラム目に“/”を記述した場合、当該行全体がコメントの扱いとなります。また、ショートネームの直後に1文字の空白を記述した場合、当該行においてその空白以降から改行またはファイルの最終位置までの文字列がコメントの扱いとなります。コメントには任意の文字が指定可能です。
- 留意事項
  - 複数のロングネームに対し、同一のショートネームを指定しないでください。
  - ロングネームは拡張子を含め、ショートネームは拡張子を含めずに指定してください。
- 例
  以下に、ユーザファイル名対応表の記述例を示します。

```
//ユーザファイル名対応表
"interstagefile01.cbl",SUBFILE1 検索処理プログラム用ファイル
"interstagefile02.cbl",SUBFILE2 更新処理プログラム用ファイル
```

ユーザプログラム名対応表

- 形式
  CSV形式のテキストファイル。ファイル名“aad-usrpnm.txt”で作成します。

- 内容
  ダブルクォーテーションで囲んだInterstage応用プログラムのプログラム名(ロングネーム)または登録集のメンバ名(ロングネーム)と、当該プログラム名または登録集のメンバ名に対応する任意の名称(英字で始まる8文字の英数字)を“,”で区切ったテキストファイルです。
  ただし、プログラム名(ロングネーム)の内、“モジュール名-インタフェース名-オペレーション名”については名称対応表生成機能で生成された対応表に格納されています。ユーザプログラム名対応表には、これ以外のInterstage応用プログラムのプログラム名(ロングネーム)と登録集のメンバ名(ロングネーム)を対象に、ロングネームとショートネームの対応表を格納してください。
  ユーザプログラム名対応表にはコメントを記述することができます。1カラム目と2カラム目に“/”を記述した場合、当該行全体がコメントの扱いとなります。また、ショートネームの直後に1文字の空白を記述した場合、当該行においてその空白以降から改行またはファイルの最終位置までの文字列がコメントの扱いとなります。コメントには任意の文字が指定可能です。
- 留意事項
  - 複数のロングネームに対し、同一のショートネームを指定しないでください。
  - 以下の書き出しで始まるロングネームをユーザプログラム名として使用することはできません。
    - “AAD”
    - “CORBA”
    - “COSNAMING”
    - “EX-COSNAMING”
    - “EX-ISAAD”

- “EX-ISTD”
- “EX-ORB”
- “EX-STEXCEP”
- “ISAAD”
- “ISTD”
- “MQD”
- “TD”

● 例

以下に、ユーザファイル名対応表の記述例を示します。

```
//ユーザプログラム名対応表
"interstagesubroutine01",SUBAPL01 検索処理プログラム
"interstagesubroutine02",SUBAPL02 更新処理プログラム
```

配付ソース生成機能の使用にあたっての準備

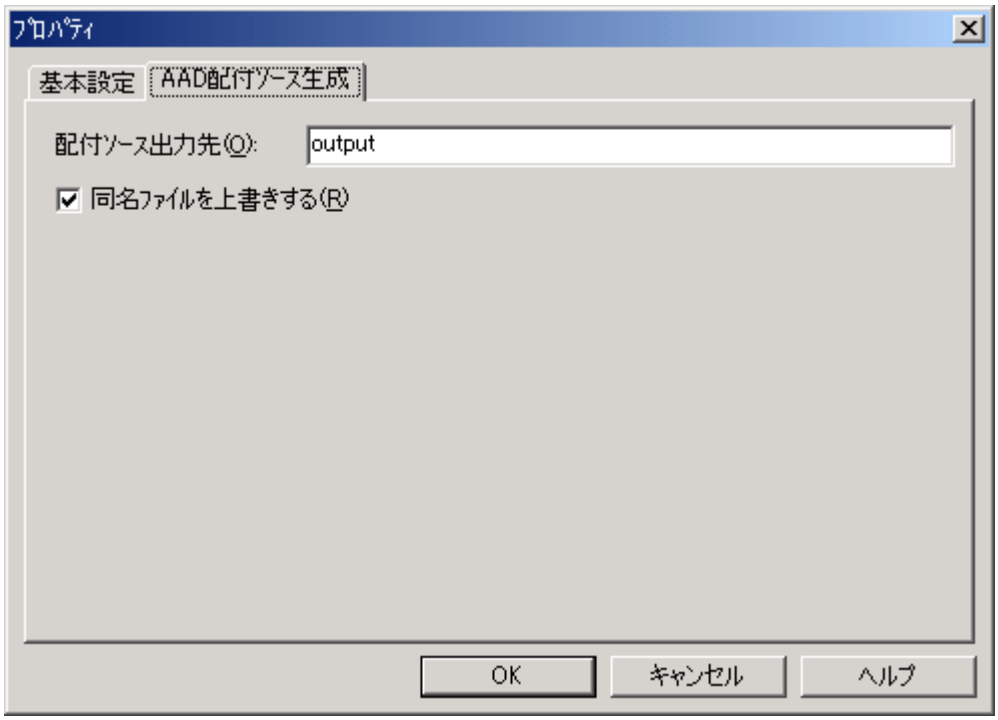
プロジェクトのプロパティ

配付ソース生成機能を使用する場合、プロジェクトのプロパティで配付ソース生成のしかたについての情報を設定しておく必要があります。

以下、その手順を説明します。

1. COBOLプロジェクトマネージャの〔プロジェクト構成〕ページで、プロジェクトファイルを選択して、〔ファイル〕メニューから“プロパティ”を選択するとプロジェクトの“プロパティ”ダイアログボックスが表示されます。
2. 〔AAD配付ソース生成〕のページの設定選択します。デフォルトでは、“図:〔プロパティ〕ダイアログの〔AAD配付ソース生成〕ページの初期状態”のようになっています。必要に応じて、次の設定を変更してください。
   - ― 配付ソース出力先:
     グローバルサーバに配付するためのソースファイルの出力先フォルダ名を指定します。フォルダはプロジェクトのサブフォルダ名を指定します。
     省略することはできません。また、指定した名前のフォルダが存在しないなら、新しいフォルダが作成されます。
   - ― 同名ファイルを上書きする:
     配付ソースファイル生成時に同名のファイルを上書きするかどうか指定します。

図6-9 〔プロパティ〕ダイアログの〔AAD配付ソース生成〕ページの初期状態



配付ソース生成機能で必要とするファイル

配付ソース生成機能で使用なる名前の変換規則を決定するための以下のファイルをプロジェクトファイルと同じフォルダに格納します。

- IDLソースファイル
- スケルトンプレフィクス一覧
- スタブプレフィクス一覧
- ユーザファイル名対応表
- ユーザプログラム名対応表

配付ソース生成機能の使用法

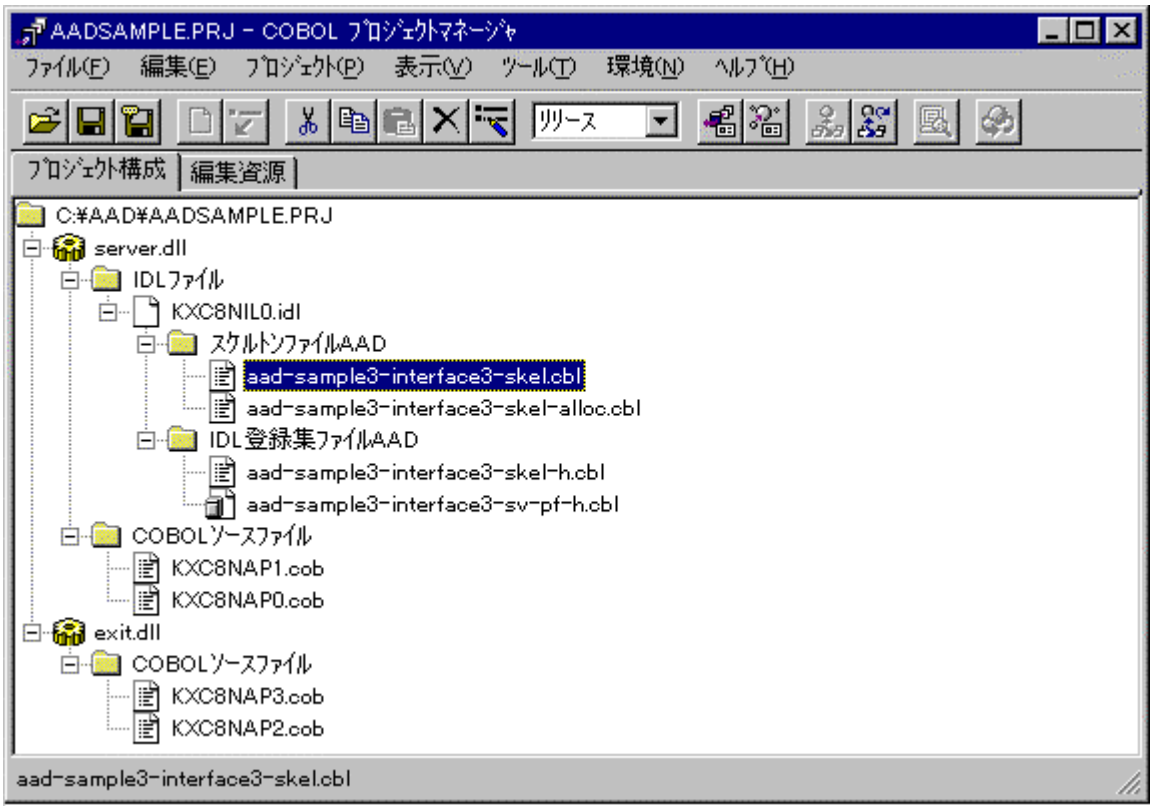
配付ソース生成を使用する場合、対象ファイルを1つずつ選択して行う方法と複数ファイルに対して、一括して処理する方法の2通りのやり方があります。

対象ファイルを1つずつ選択して行う方法

COBOLプロジェクトマネージャのツリービューから操作する場合は対象ファイル1つずつの操作となります。

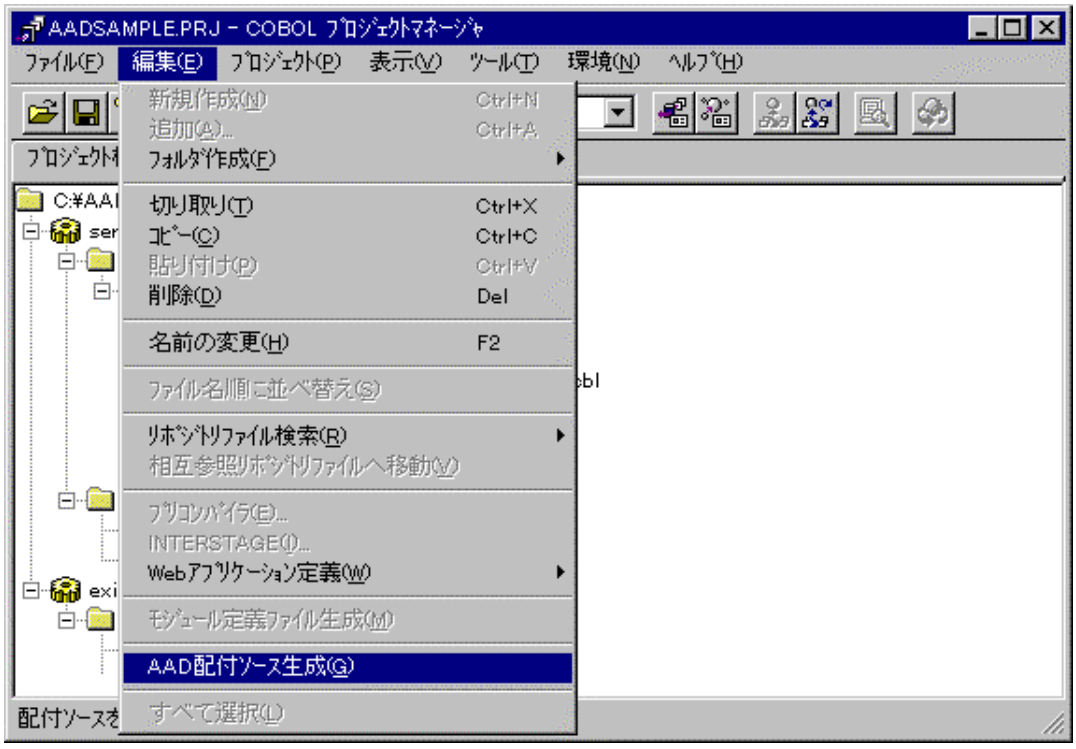
1. COBOLプロジェクトマネージャの〔プロジェクト構成〕ページで、対象となるファイルを選択します。

図6-11 図6-10 配付ソース生成対象の選択



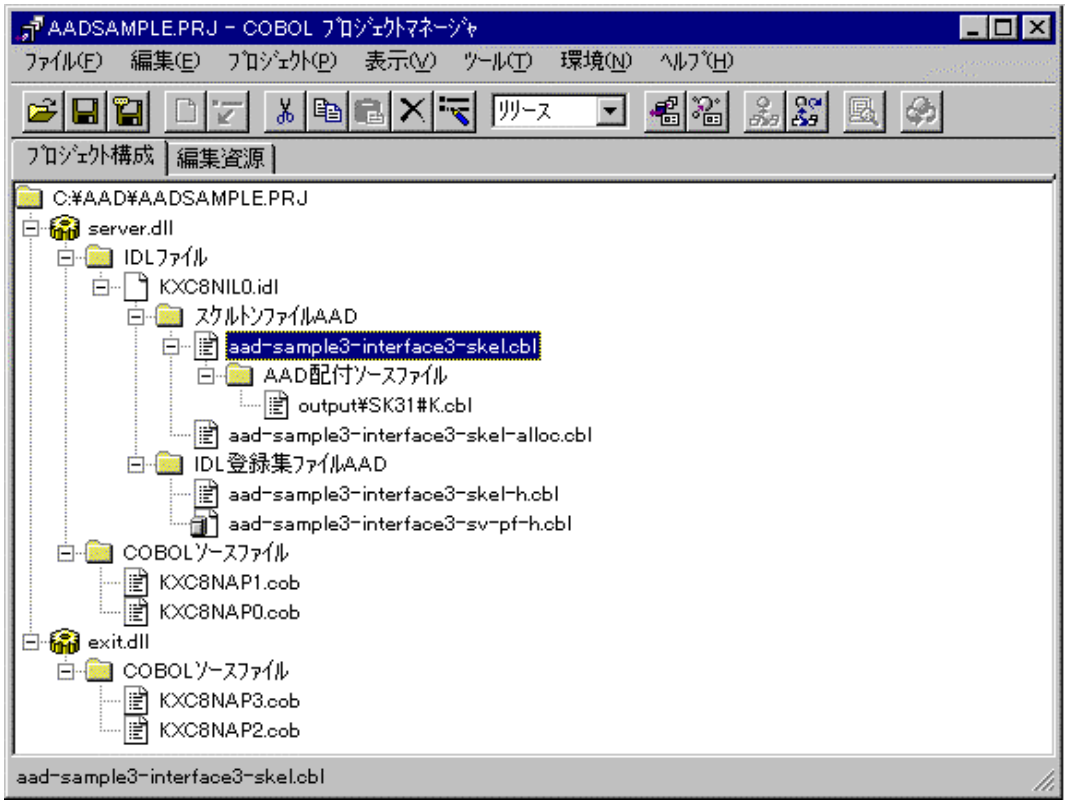
2. 〔編集〕メニューの“AAD配付ソース生成”を選択します。

図6-12 〔編集〕メニューからの“AAD配付ソース生成”の選択



3. 生成に成功すると、生成元のファイルの下にAAD配付ソースファイルフォルダと生成したAAD配付ソースファイルが登録されます。

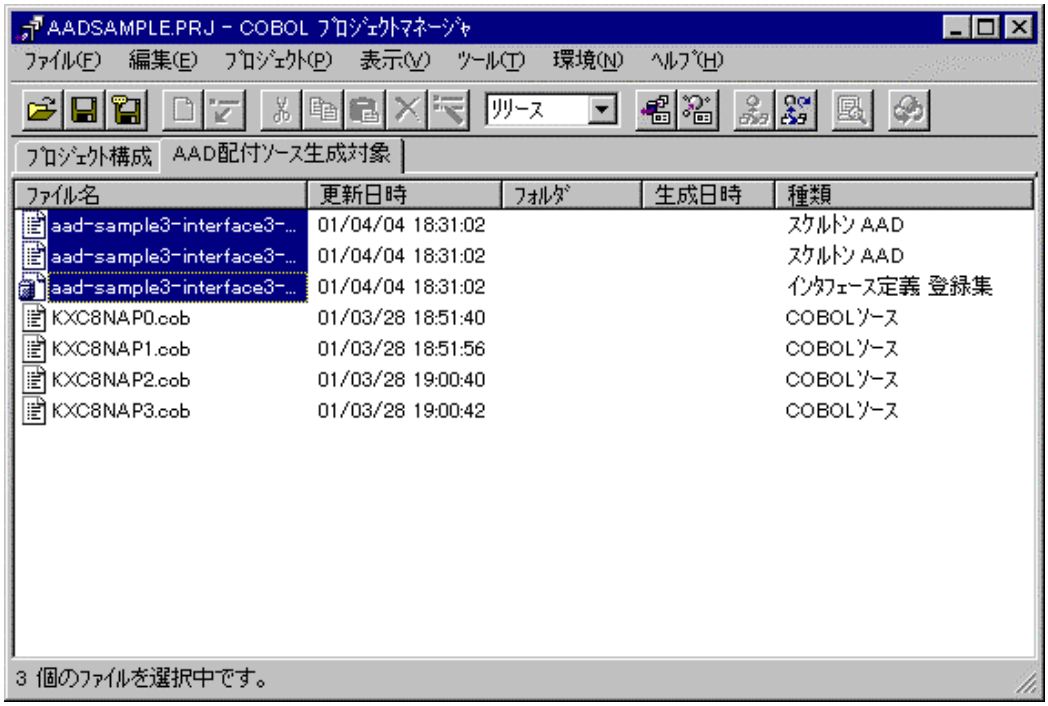
図6-13 生成された配付ソース



複数ファイルを一括して配付ソース生成する方法

“リストビュー”を使用することによって、複数のファイルから一括して配付ソースを生成することもできます。

図6-14 “リストビュー”のAAD配付ソース生成対象表示



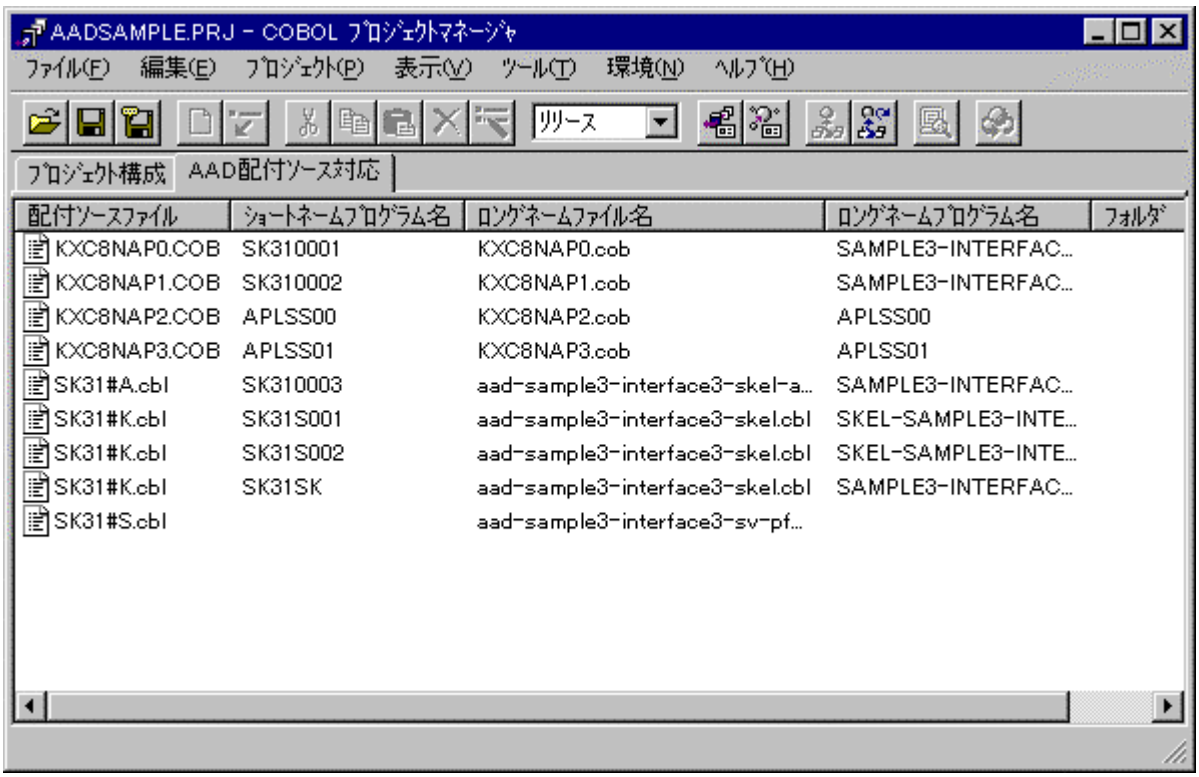
以下に手順を示します。

1. 〔表示〕-〔リスト表示内容〕メニューから“AAD配付ソース生成対象”を選択します。これによって、リストビューに、生成対象ファイルだけが一覧表示されます。また、同時に、以下の情報も画面上に表示されます。
   - ファイル名:
     生成対象のファイル名です。
   - 更新日時:
     生成対象ファイルの最終更新日時です。
   - フォルダ:
     生成対象ファイルが存在するフォルダです。
   - 生成日時:
     最後に配付ソース生成を実行した日時です。
   - 種類:
     生成対象ファイルの種類です。
2. リストビューに表示されたファイルのうち、配付ソース生成が必要なものを選択します。
3. 〔編集〕メニューの“AAD配付ソース生成”を選択します。

6.2.2.3 配付ソース対応表示

配付ソース生成前のロングネームと、生成後のショートネームの対応は、リストビューで表示することができます。
対応を表示するには、〔表示〕-〔リスト表示内容〕メニューの“AAD配付ソース対応”を選択します。

図6-15 配付ソース対応表示



〔画面の説明〕

- 配付ソースファイル:
  配付ソースのファイル名です。
- ショートネームプログラム名:

配付ソース生成で生成したプログラム名です。

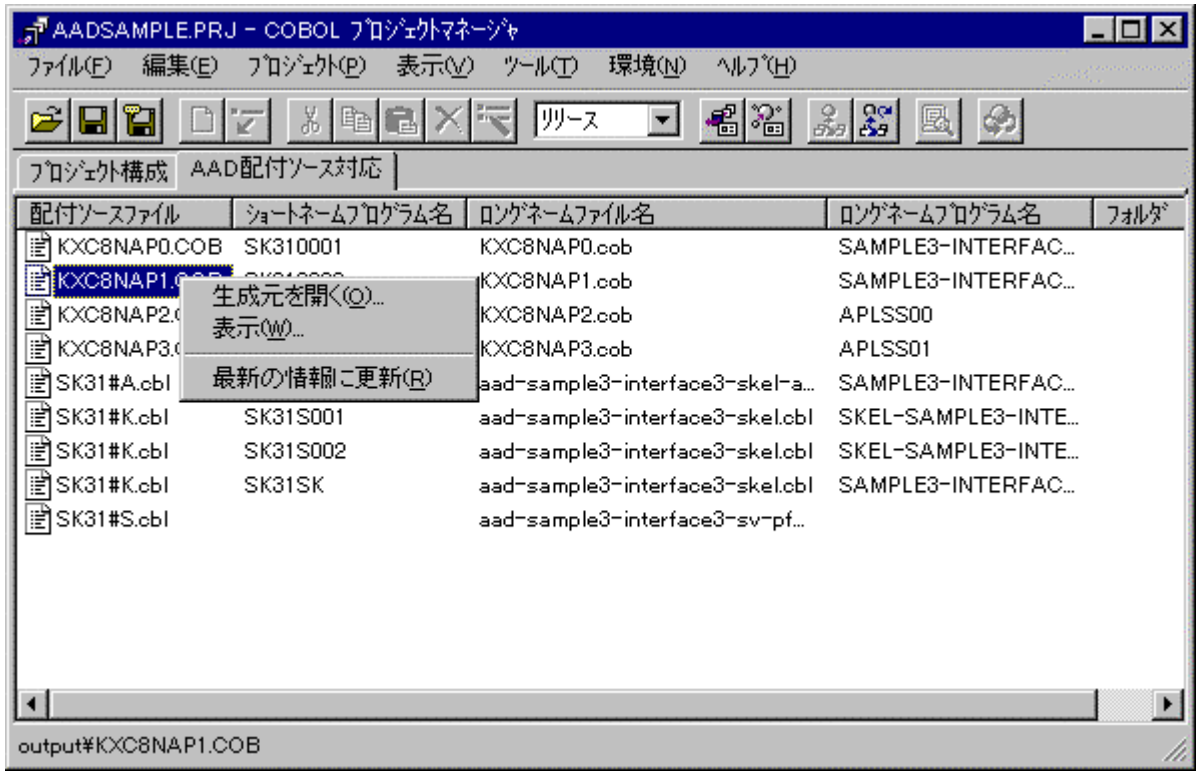
- ロングネームファイル名:
  配付ソースの生成元のファイル名です。
- ロングネームプログラム名:
  配付ソースの生成元のプログラム名です。
- フォルダ:
  生成元のファイルを格納しているフォルダです。ツリーに登録しているファイルにフォルダ名がある場合に表示します。

配付ソース対応表示画面でのファイルの操作

このソースファイル対応表示の画面で、ファイルを選択してマウスの右ボタンをクリックすると、メニューが表示されます。表示されたメニューを使用して、次のような操作が可能です。

- 生成元を開く:
  配付ソースの生成元のファイルを開きます。
  生成元のファイルがCOBOLソースファイルまたは登録集の場合、編集画面が開きます。それ以外のファイルの場合、表示状態で開きます。
- 表示:
  選択したファイルを表示状態で開きます。

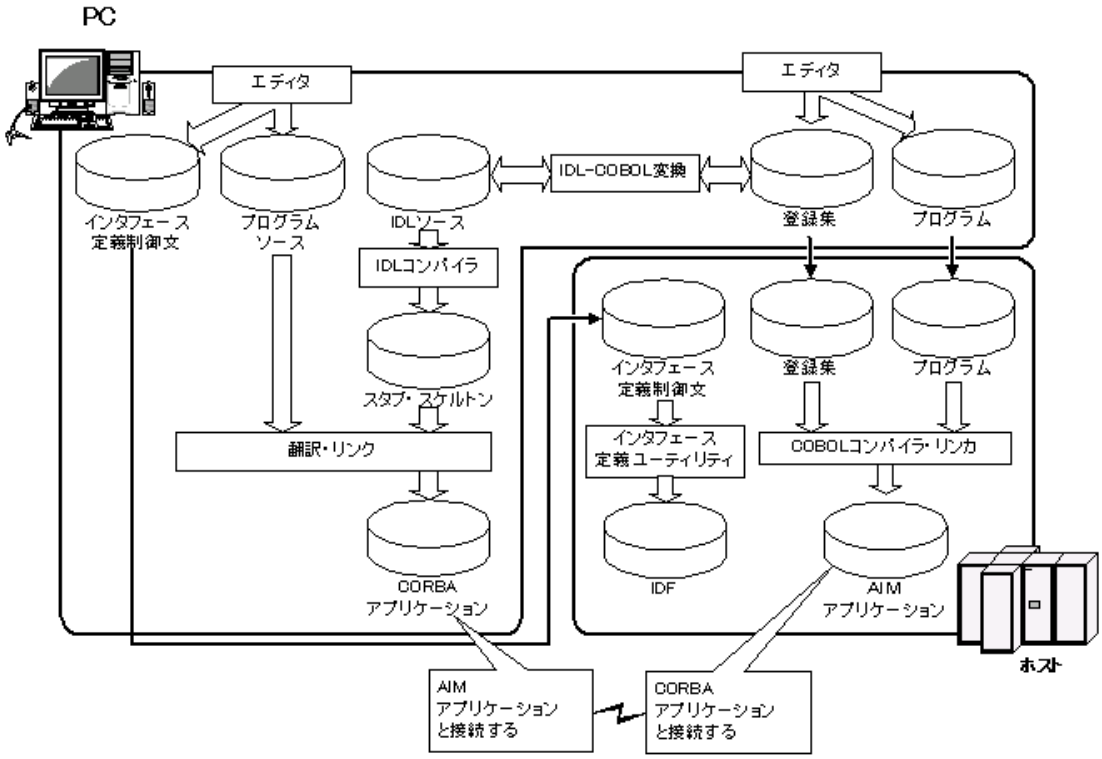
図6-16 配付ソース対応表示画面でのファイル操作メニュー



6.3 AIMアプリケーションの開発

AIMアプリケーションをCORBAアプリケーションの機能を持つものとして開発する場合、従来型の表示ファイルインタフェースを使用するため、分散開発は必須であるとは言えません。
しかし、AIMアプリケーションがCORBAのサーバあるいはクライアントアプリケーションと接続するためには、AIMアプリケーションの使用する表示ファイルインタフェースに適合したインタフェース定義(IDL)が必要であり、Windows系システムで開発作業は必要です。
以下にCORBAのサーバあるいはクライアントと接続するAIMアプリケーションを分散開発する場合の資産の流れを示します。

図6-17 CORBAアプリケーションと接続するAIMアプリケーション開発時の資産の流れ



NetCOBOLでは、このような開発手順を支援するために次の機能を提供しています。

- COBOL→IDL変換機能
  COBOL登録集から、IDLソースファイルを生成する機能
- IDL→COBOL変換機能
  IDLソースファイルから、AIM応用プログラムが使用するCOBOL登録集を生成する機能

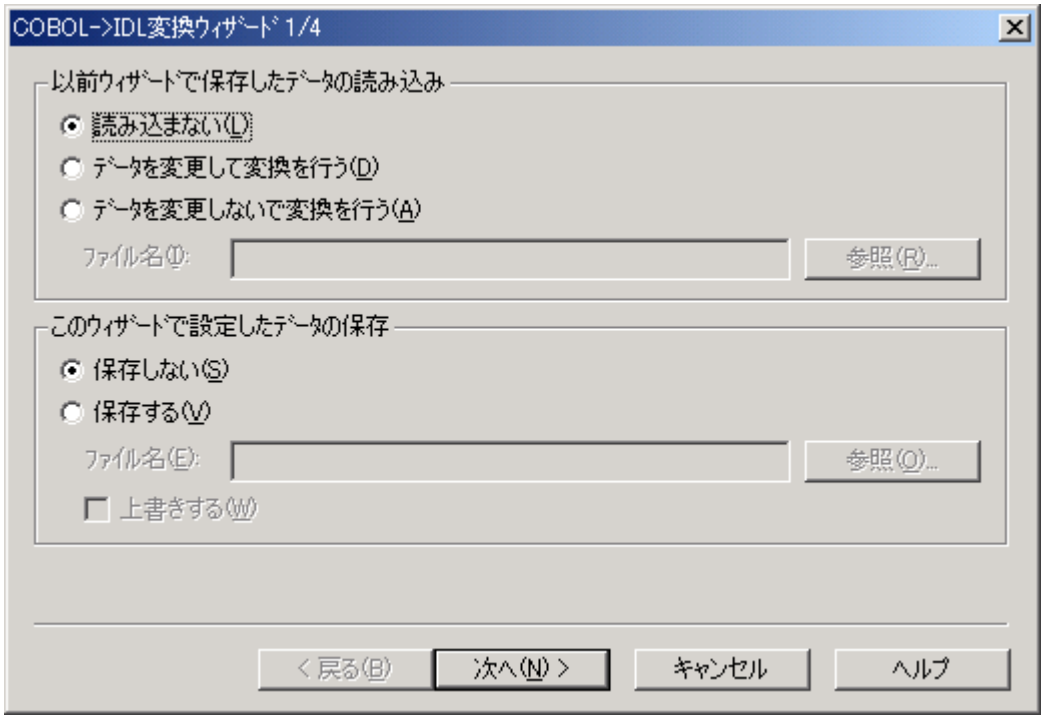
6.3.1 COBOL-IDL変換機能

AIMアプリケーションは、表示ファイルインタフェース(READ/WRITE)でCORBAアプリケーションと連携します。CORBAサーバとなるAIMアプリケーションの開発時、インタフェースはREAD/WRITEで使用するレコードを宣言した登録集で定義されています。これからクライアントへ配付するIDLを作成する必要があります。このための機能として、NetCOBOLでは、COBOL-IDL変換機能を提供しています。
IDLの生成は通信の形態が同期通信か、非同期通信かによって、一部の操作が異なります。
COBOL-IDL変換機能はウィザード形式で共通の設定の上で、それぞれの通信形態ごとの情報を設定するようになっています。
以下、共通の設定を説明した上で、通信形態ごとの操作の手順を示します。

共通の設定

1. COBOLプロジェクトマネージャの〔ツール〕－〔IDL-COBOL変換〕メニューから“COBOL-IDL変換”を選択すると、COBOL->IDL変換ウィザードが起動します。
2. COBOL->IDL変換ウィザードの最初の画面では、COBOL->IDL変換の設定の読み込みと保存について指定します。

図6-18 COBOL->IDL変換ウィザード1/4



各部分についての機能は次のとおりです。

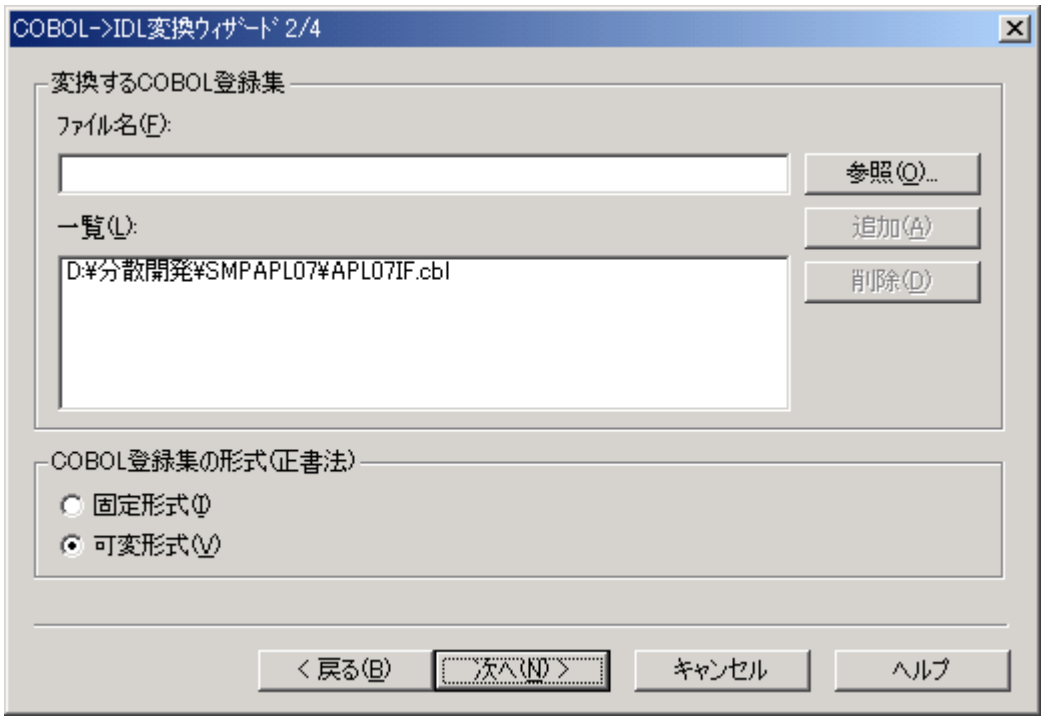
— 以前ウィザードで保存したデータの読み込み:
以前ウィザードで設定したデータを保存したファイル(拡張子itc)を参照するかどうか指定します。それぞれ、次のような場合に選択します。

- 読み込まない
  新規にCOBOL->IDL変換を行う場合に指定します。
- データを変更して変換を行う
  保存してあるデータをファイルから読み込み、データを更新または確認して変換を行う場合に指定します。
- データを変更しないで変換を行う
  保存してあるデータに従って、COBOL登録集からIDLファイルへの再変換を行う際に指定します。このチェックボックスをチェックすると、ウィザードの残りの画面を飛ばして、ただちに変換操作を行えます。

— このウィザードで設定したデータの保存:
設定したデータをファイルに保存するかどうかを指定します。
“保存する”を選択した場合、ウィザードで設定したデータは“ファイル名”に指定したファイル(拡張子itc)に保存されます。既存のファイルに上書きする場合は、“上書きする”のチェックボックスをチェックします。

3. ウィザードの最初の画面で必要な情報を設定し、〔次へ〕のボタンをクリックするとウィザードの2つ目の画面が現れます。
この画面では、変換するCOBOL登録集ファイルの情報を指定します。

図6-19 COBOL->IDL変換ウィザード2/4

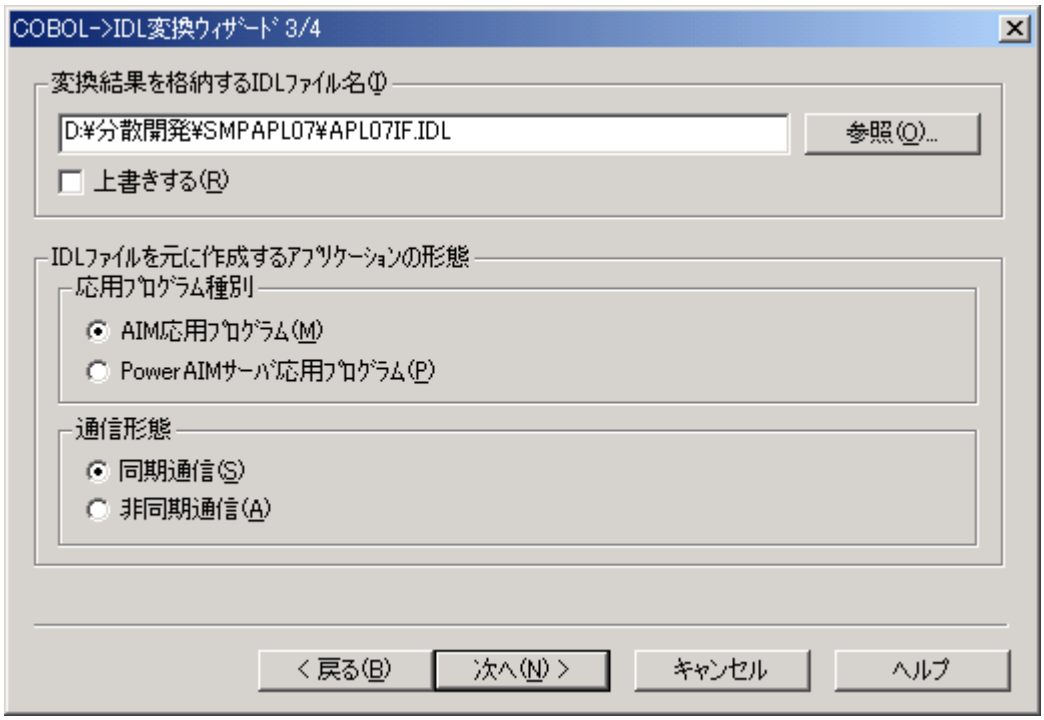


各部分についての機能は次のとおりです。

- 変換するCOBOL登録集:
  変換対象となるCOBOL登録集を指定します。変換対象となるCOBOL登録集が〔一覧〕に表示されます。
  対象ファイル名を絶対パスで指定するか、あるいは〔参照〕ボタンで〔ファイルの参照〕ダイアログを開いてファイルを選択して、“ファイル名”を指定して、〔追加〕ボタンをクリックすると、〔一覧〕に追加されます。
  〔一覧〕から削除する場合は、〔一覧〕内でファイル名を選択し、〔削除〕ボタンをクリックします。
- COBOL登録集の形式(正書法):
  変換対象となるCOBOL登録集のソースの正書法を指定します。COBOLソースの正書法についての詳細は“COBOL文法書”を参照してください。

4. ウィザードの2つ目の画面で必要な情報を設定し、〔次へ〕のボタンをクリックするとウィザードの3つ目の画面が現れます。
   この画面では、変換結果を格納するIDLファイルの情報を指定します。

図6-20 COBOL->IDL変換ウィザード3/4



各部分についての機能は次のとおりです。

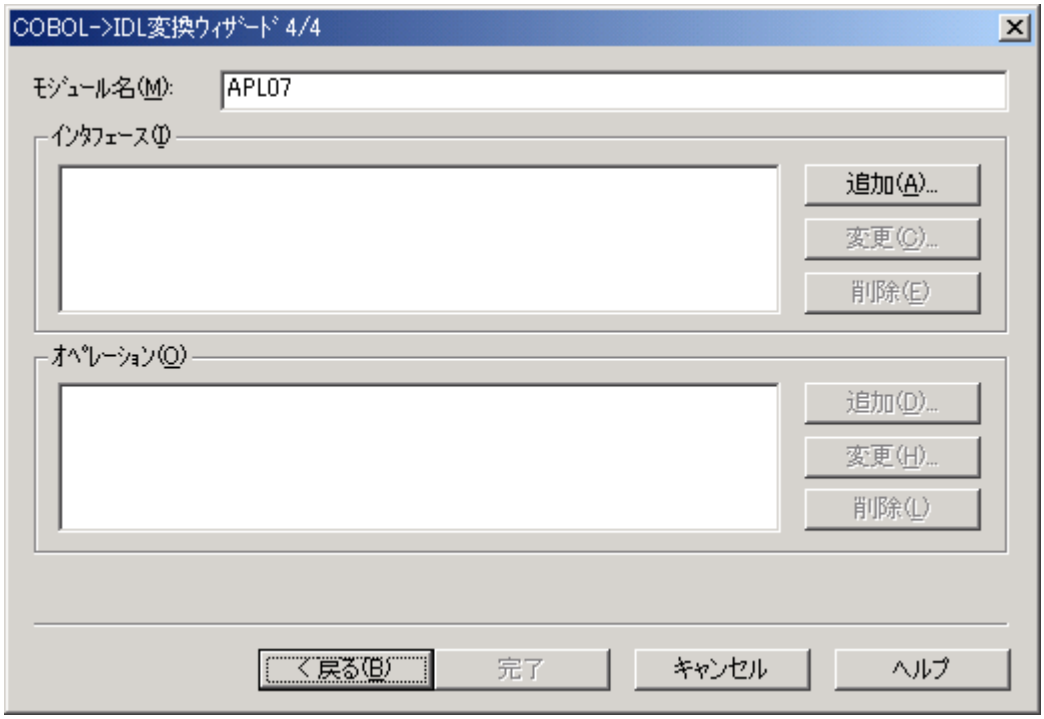
- 変換結果を格納するIDLファイル名:
  変換結果を格納するIDLファイル名を絶対パスで指定します。〔参照〕ボタンで〔ファイルの参照〕ダイアログを開いてファイルを選択することもできます。既存のファイルを上書きする場合には〔上書き〕のチェックボックスをチェックします。
- 応用プログラム種別:
  変換対象となるCOBOL登録集を使用するアプリケーションの種別を指定します。
  - AIM応用プログラム
    アプリケーションがAIMの表示ファイルインタフェースを使用するものであることを指定します。
  - PowerAIM応用プログラム
    アプリケーションがPowerAIMのDCSQLインタフェースを使用するものであることを指定します。
- 通信形態:
  変換対象となるCOBOL登録集を使用するアプリケーションの通信形態を指定します。
  - 同期通信
    IDLファイルには、[COBOL->IDL変換オペレーション設定]ダイアログで指定する情報に対応したオペレーション宣言が出力されます。
  - 非同期通信
    IDLには、オペレーション宣言が出力されず、[COBOL->IDL変換インタフェース設定]ダイアログの[インタフェースを定義しているデータ項目]エディットボックスで指定するデータ項目に対応した型宣言だけが出力されます。

5. ウィザードの3つ目の画面で必要な情報を設定し、〔次へ〕のボタンをクリックするとウィザードの最後の画面が現れます。
   〔通信形態〕の選択により、最後の画面では、設定する項目と操作が異なります。

同期通信使用時の設定

1. 同期通信使用時、ウィザードの最後の画面ではIDLに出力するモジュール名およびインタフェース／オペレーションの情報を指定します。

図6-21 COBOL->IDL変換ウィザード4/4(同期通信時)



各部分についての機能は次のとおりです。

- モジュール名:
  IDLに出力するモジュール名を指定します。
- インタフェース名:
  IDLに出力するインタフェース名を指定します。〔追加〕ボタンをクリックすると、〔COBOL->IDL変換インタフェース設定〕ダイアログが現れるので、インタフェース名を指定します。追加されたインタフェース名はリストボックスに表示されます。
  既に追加済みのインタフェース名を変更あるいは削除する場合には、リストボックスでインタフェース名を選択して、〔変更〕あるいは〔削除〕ボタンをクリックします。
- オペレーション名:
  IDLに出力するオペレーション名を指定します。
  〔インタフェース〕のリストボックスでオペレーションを追加するインタフェース名を選択して、〔追加〕ボタンをクリックすると、〔COBOL->IDL変換オペレーション設定〕ダイアログが現れるので、オペレーション名とそれに付随する情報を指定します。追加されたオペレーション名はリストボックスに表示されます。
  既に追加済みのオペレーション名を変更あるいは削除する場合には、リストボックスでオペレーション名を選択して、〔変更〕あるいは〔削除〕ボタンをクリックします。

2. 〔COBOL->IDL変換インタフェース設定〕ダイアログは次のようなもので、出力するモジュールに含まれるインタフェース名を指定します。

図6-22 〔COBOL->IDL変換インタフェース設定〕ダイアログ(同期通信時)

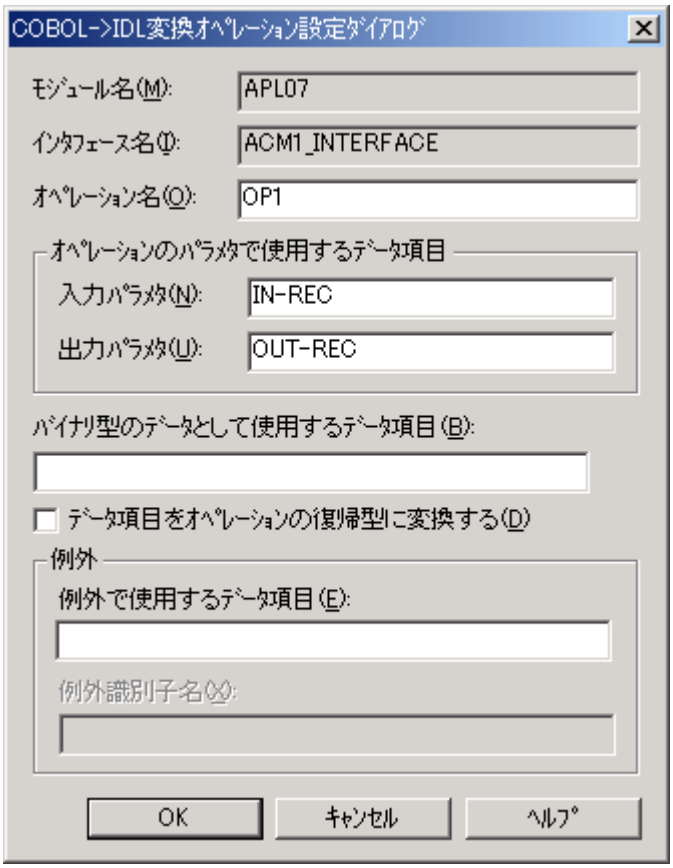
各部分についての機能は次のとおりです。

— モジュール名:
前の画面で設定したモジュール名を表示します。

— インタフェース名:
“モジュール名”に含まれるインタフェース名を指定します。〔OK〕ボタンをクリックすると、設定した情報を有効にして前のウィザード画面に戻ります。前のウィザード画面の[インタフェース]リストボックスに、設定したインタフェース名が表示されます。

3. アプリケーションの形態として“AIM応用プログラム”を選択した場合、以下のような〔COBOL->IDL変換オペレーション設定〕ダイアログが表示されます。

図6-23 〔COBOL->IDL変換オペレーション設定〕ダイアログ(AIM応用プログラム時)



各部分についての機能は次のとおりです。

— モジュール名:
前の画面で設定したモジュール名を表示します。

— インタフェース名:
前の画面で選択したインタフェース名を表示します

— オペレーション名:

IDLファイルに出力するオペレーション名を指定します。

— 入力パラメタ:
IDLファイルに出力するオペレーションの入力パラメタをCOBOLのデータ項目名で指定します。指定できるデータ項目は、COBOL登録集ファイルに記述されたレベル番号01のデータ項目です。

— 出力パラメタ:
IDLファイルに出力するオペレーションの出力パラメタをCOBOLのデータ項目名で指定します。指定できるデータ項目は、COBOL登録集ファイルに記述されたレベル番号01のデータ項目です。

— バイナリ型のデータとして使用するデータ項目:
バイナリ型のデータとして使用するデータ項目を指定します。空白で区切ることで、複数のデータ項目を指定できます。バイナリ型のデータはIDLファイルではoctet型になります。ここで指定できるデータ項目は、以下のとおりです。
  - レベル番号49のデータ項目LEN、及びDATが従属するデータ項目(集団項目)
  - PIC X(n)のデータ項目 (ただし、(a)のデータ項目DATを除く)

— データ項目をオペレーションの復帰型に変換する:
IDLファイルに出力するオペレーションの復帰型を出力パラメタで指定したデータ項目から変換することを指定します。“出力パラメタ”にデータ項目を指定した場合に指定することができます。復帰型に変換するデータ項目は、出力パラメタで指定したデータ項目に従属する先頭のレベル番号02のデータ項目です。

— 例外で使用するデータ項目:
IDLファイルに出力する例外宣言をCOBOLデータ項目名で指定します。指定できるデータ項目は、COBOL登録集ファイルに記述されたレベル番号01のデータ項目です。

— 例外識別子名:
IDLファイルに出力する例外宣言の例外識別子名を指定します。“例外で使用するデータ項目”にデータ項目を指定した場合に指定することができます。

4. アプリケーションの形態として“PowerAIM応用プログラム”を選択した場合、以下のような〔COBOL->IDL変換オペレーション設定〕ダイアログが表示されます。
“AIM応用プログラム”と異なる部分についての機能は次のとおりです。

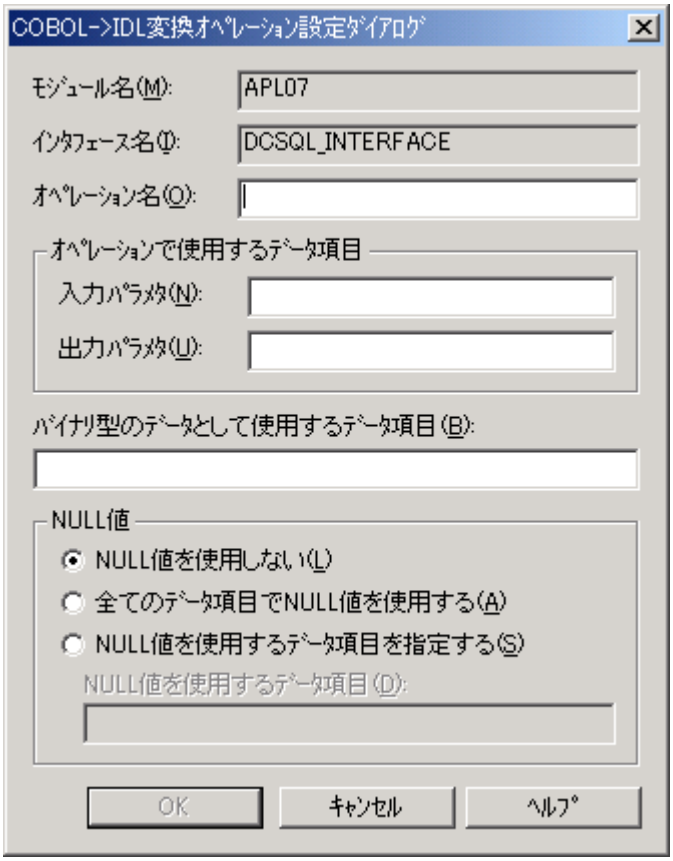
— NULL値を使用しない:
NULL値を使用するデータ項目がない場合に指定します。

— 全てのデータ項目でNULL値を使用する:
入力パラメタまたは出力パラメタで指定したデータ項目に従属する、すべてのレベル番号02のデータ項目で、NULL値を使用する場合に指定します。

— NULL値を使用するデータ項目を指定する:
NULL値を使用するデータ項目を指定する場合に指定します。

— NULL値を使用するデータ項目:
NULL値を使用するデータ項目を指定します。[NULL値を使用するデータ項目を指定する]ラジオボタンを指定した場合に指定することができます。指定できるデータ項目は、入力パラメタまたは出力パラメタで指定したデータ項目に従属するレベル番号02のデータ項目です。

図6-24 COBOL->IDL変換オペレーション設定ダイアログ(PowerAIM応用プログラム時)



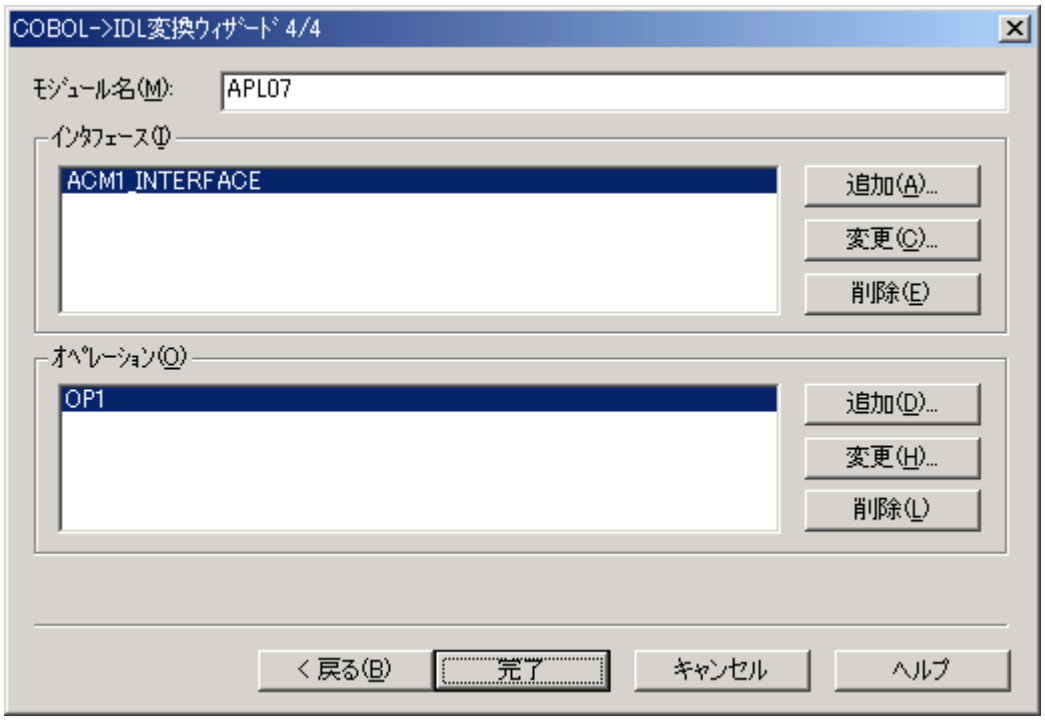
各部分についての機能は次のとおりです。

- モジュール名:
  IDLに出力するモジュール名を指定します。
- インタフェース名:
  IDLに出力するモジュールに含まれるインタフェース名を指定します。〔追加〕ボタンをクリックすると、〔COBOL->IDL変換インタフェース設定〕ダイアログが現れるので、インタフェース名を指定します。追加されたインタフェース名はリストボックスに表示されます。

既に追加済みのインタフェース名を変更あるいは削除する場合には、リストボックスでインタフェース名を選択して、〔変更〕あるいは〔削除〕ボタンをクリックします。

5. 以上で、COBOL->IDL変換に必要な情報の設定は終了です。〔完了〕ボタンをクリックするとCOBOL->IDL変換が実施されます。

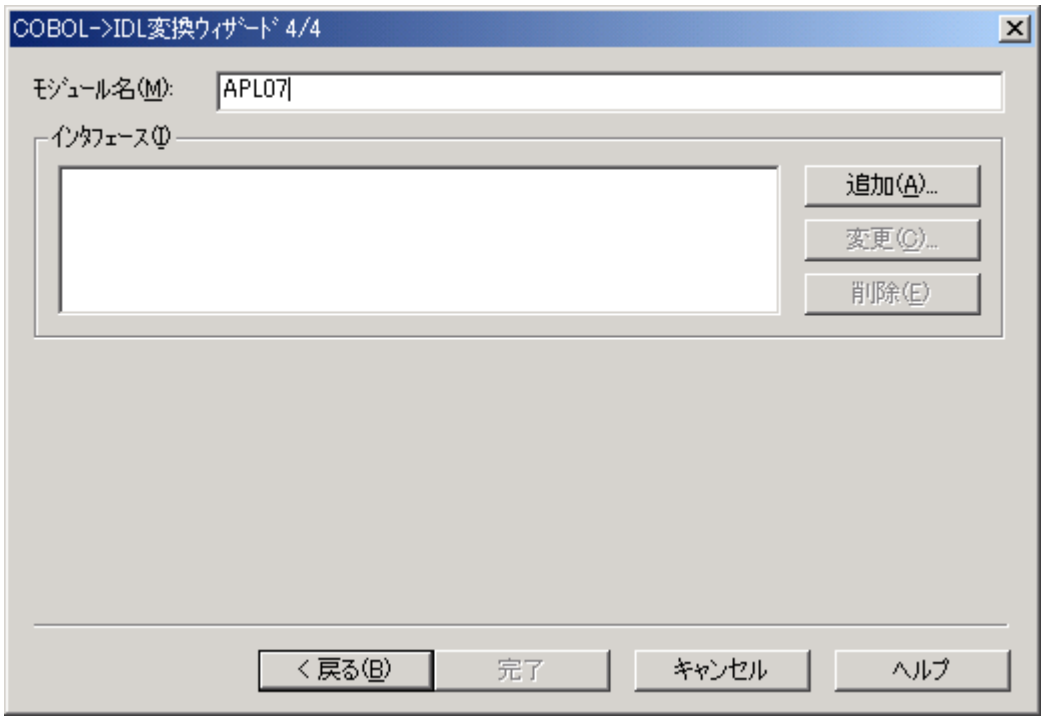
図6-25 COBOL->IDL変換ウィザード4/4(同期通信時)



非同期通信使用時の設定

1. 非同期通信使用時、ウィザードの最後の画面ではIDLに出力するモジュール名およびインタフェースの情報を指定します。

図6-26 COBOL->IDL変換ウィザード4/4(非同期通信時)



各部分についての機能は次のとおりです。

— モジュール名:

IDLに出力するモジュール名を指定します。

— インタフェース名:
IDLに出力するモジュールに含まれるインタフェース名を指定します。〔追加〕ボタンをクリックすると、〔COBOL->IDL変換インタフェース設定〕ダイアログが現れるので、インタフェース名を指定します。追加されたインタフェース名はリストボックスに表示されます。
既に追加済みのインタフェース名を変更あるいは削除する場合には、リストボックスでインタフェース名を選択して、〔変更〕あるいは〔削除〕ボタンをクリックします。

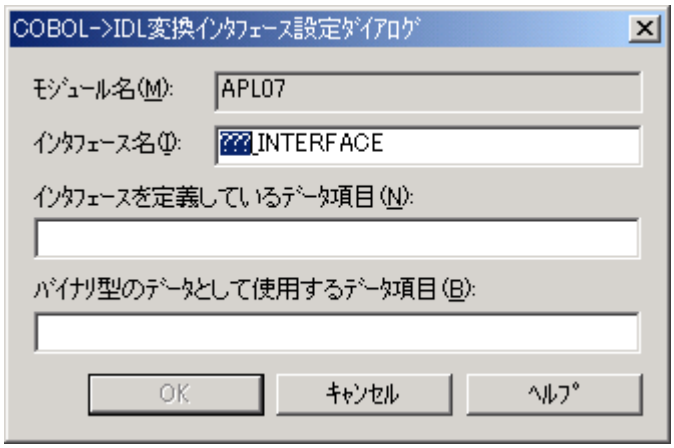
2. 〔COBOL->IDL変換インタフェース設定〕ダイアログでは、インタフェース名とCOBOL登録集内に含まれるそれを定義するデータ項目名を指定します。
各部分についての機能は次のとおりです。

— インタフェース名:
IDLに出力するモジュールに含まれるインタフェース名を指定します。

— インタフェースを定義しているデータ項目名:
IDLファイルに出力する型宣言をCOBOLのデータ項目名で指定します。指定できるデータ項目は、COBOL登録集ファイルに記述されたレベル番号01のデータ項目です。

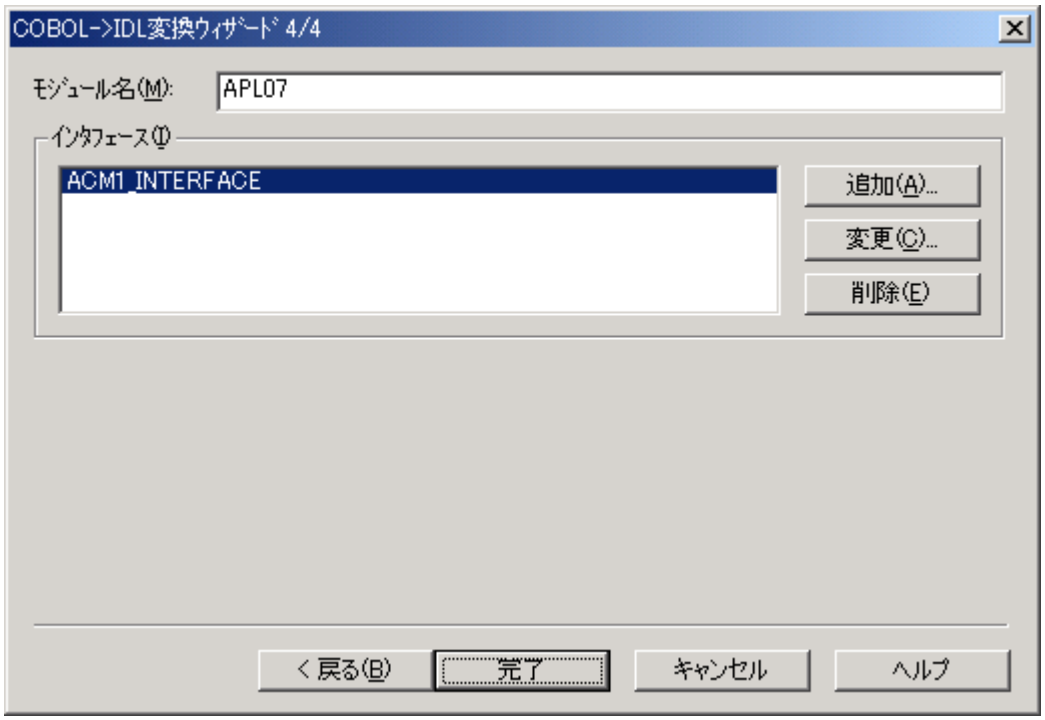
— バイナリ型のデータとして使用するデータ項目:
バイナリ型のデータとして使用するデータ項目を指定します。バイナリ型のデータはIDLファイルではoctet型になります。空白で区切ることで、複数のデータ項目を指定できます。なお、ここで指定できるデータ項目は、以下のとおりです。
- レベル番号49のデータ項目LEN、及びDATが従属するデータ項目(集団項目)
- PIC X(n)のデータ項目 (ただし、(a)のデータ項目DATを除く)

図6-27 COBOL->IDL変換インタフェース設定ダイアログ (非同期通信時)



3. 以上で、COBOL->IDL変換に必要な情報の設定は終了です。〔完了〕ボタンをクリックするとCOBOL->IDL変換が実施されます。

図6-28 COBOL->IDL変換ウィザード4/4(非同期通信時)



6.3.2 IDL-COBOL変換機能

AIMアプリケーションは、表示ファイルインタフェース(READ/WRITE)でCORBAアプリケーションと連携します。CORBAクライアントとなるAIMアプリケーションの開発時、サーバ側のインタフェースはIDLで定義され、配付されるので、これをCOBOLのレコード定義に変換する必要があります。
また、CORBAクライアントとなるAIMアプリケーションの開発時でも、インタフェースを最初はIDLで定義する場合もあります。そのような場合、このIDLからCOBOLのレコード定義を生成する必要があります。
このための機能として、NetCOBOLでは、COBOL-IDL変換機能を提供しています。

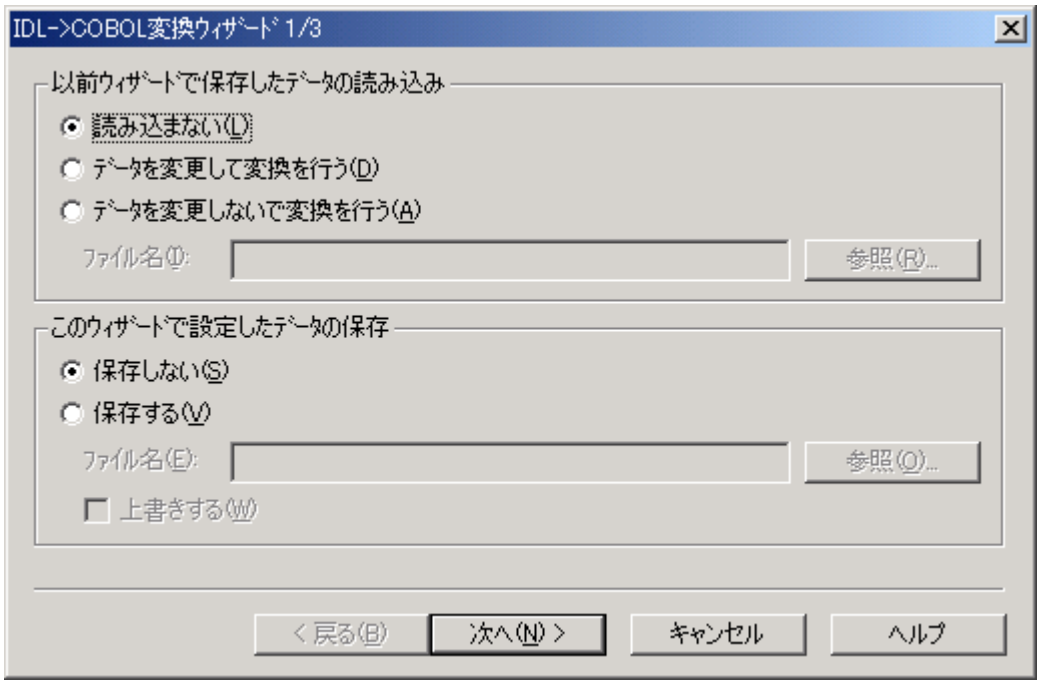
注意

この機能の使用には、Interstageが必須です。Interstageのインストールと設定が完了していない場合、〔ツール〕－〔IDL-COBOL変換〕メニューの“IDL-COBOL変換”は無効化されています。

以下、その操作手順を説明します。

1. COBOLプロジェクトマネージャの〔ツール〕－〔IDL-COBOL変換〕メニューから“IDL-COBOL変換”を選択すると、IDL->COBOL変換ウィザードが起動します。
2. IDL->COBOL変換ウィザードの最初の画面では、IDL-COBOL変換の設定の読み込みと保存について指定します。

図6-29 IDL-COBOL変換ウィザード1/3



各部分についての機能は次のとおりです。

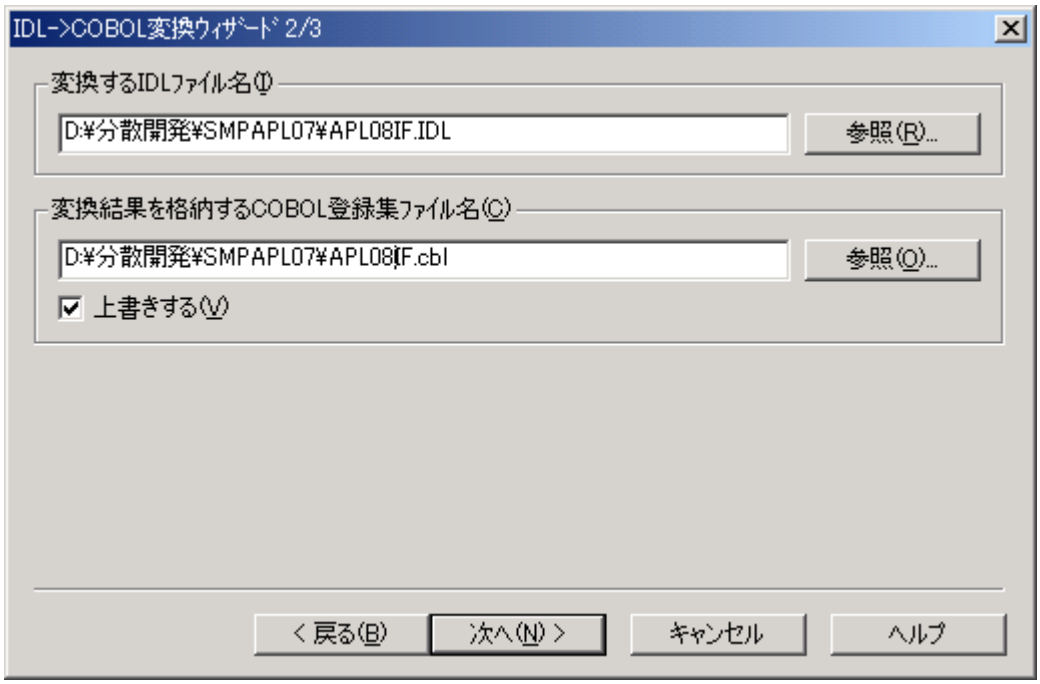
— 以前ウィザードで保存したデータの読み込み:
以前ウィザードで設定したデータを保存したファイル(拡張子itc)を参照するかどうか指定します。それぞれ、次のような場合に選択します。

- 読み込まない
新規にIDL->COBOL変換を行う場合に指定します。
- データを変更して変換を行う
保存してあるデータをファイルから読み込み、データを更新または確認して変換を行う場合に指定します。同じIDLからサーバ用とクライアント用の両方の登録集を生成するような場合に使用します。
- データを変換しないで変換を行う
保存してあるデータに従って、IDLからCOBOL登録集の再変換を行う際に指定します。このチェックボックスをチェックすると、ウィザードの残りの画面を飛ばして、ただちに変換操作を行えます。

— このウィザードで設定したデータの保存:
設定したデータをファイルに保存するかどうかを指定します。
“保存する”を選択した場合、ウィザードで設定したデータは“ファイル名”に指定したファイル(拡張子itc)に保存されます。既存のファイルに上書きする場合は、“上書きする”のチェックボックスをチェックします。

3. ウィザードの最初の画面で必要な情報を設定し、〔次へ〕のボタンをクリックするとウィザードの2つ目の画面が現れます。
この画面では、変換するIDLファイルおよび変換結果を格納するCOBOL登録集ファイルの情報を指定します。

図6-30 IDL-COBOL変換ウィザード2/3

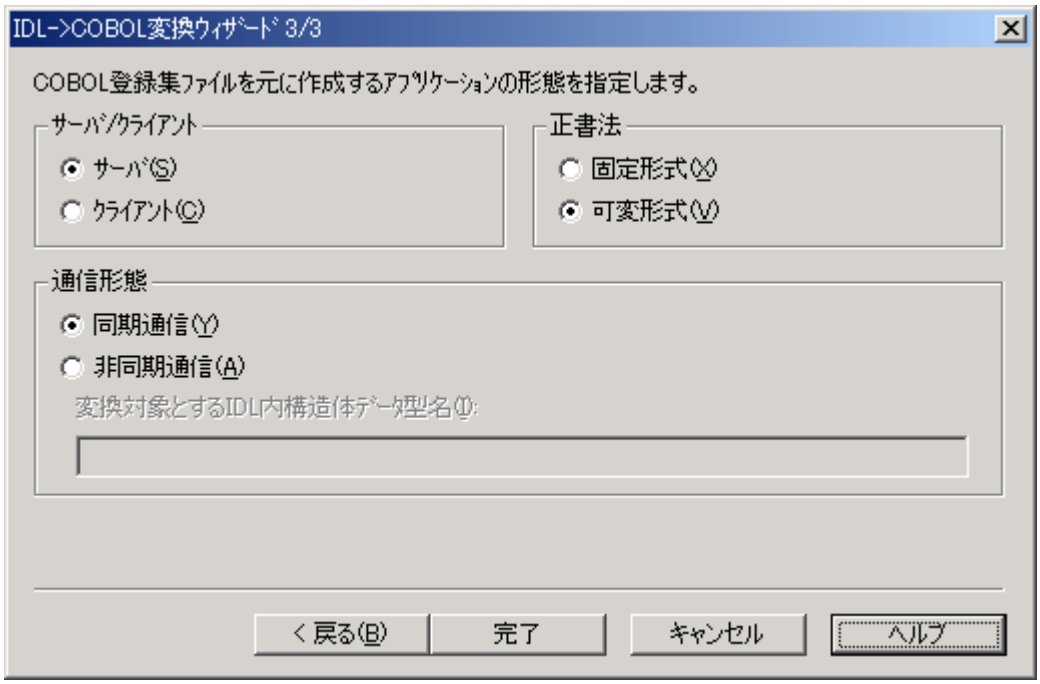


各部分についての機能は次のとおりです。

- 変換するIDLファイル名:
  入力となるIDLのファイル名を指定します。〔参照〕ボタンをクリックして、〔ファイル名の参照〕ダイアログを開いてファイルを選択することもできます。
- 変換結果を格納するCOBOL登録集ファイル:
  出力先となるCOBOL登録集の格納ファイル名を指定します。〔参照〕ボタンをクリックして、〔ファイル名の参照〕ダイアログを開いてファイルを選択することもできます。また、既存のファイルを上書きするときは、〔上書き〕のチェックボックスをチェックします。

4. ウィザードの2つ目の画面で必要な情報を設定し、〔次へ〕のボタンをクリックするとウィザードの最後の画面が現れます。
   この画面では、COBOL登録集ファイルへの出力形式を指定します。

図6-31 IDL-COBOL変換ウィザード3/3



各部分についての機能は次のとおりです。

- サーバ／クライアント変換するIDLファイル名:
  出力するCOBOL登録集がサーバアプリケーションで使用されるか、クライアントアプリケーションで使用されるかを指定します。

  クライアント用COBOL登録集には、AAD連携情報域が出力されません。使用者がAAD連携情報域を追加する必要があります。AAD連携情報域の詳細については「OS IV INTERSTAGE システム開発手引書(AIM連携)」を参照してください。
- 正書法:
  出力先となるCOBOLソースの正書法を指定します。COBOLソースの正書法の詳細については“COBOL文法書”を参照してください。
- 通信形態:
  出力するCOBOL登録集を使用するアプリケーションの通信形態を指定します。
  - 同期通信
    COBOL登録集には、IDLファイル内のオペレーション宣言で記述されたパラメタ、復帰値および例外に対応するデータ項目が出力されます。
  - 非同期通信
    COBOL登録集には、“変換対象とするIDL内構造体データ型名”で指定したデータ型名に対応するデータ項目が出力されます。

5. ウィザードの最後の画面で必要な情報を設定し、〔完了〕のボタンをクリックするとIDL-COBOL変換が行われます。

第7章 トラブルシューティング

分散開発は、OSIV系システムとWindows系システムという異なる2つのシステムをまたがって開発を行うため、通常のアプリケーション開発では起こり得ないトラブルに遭遇する場合があります。また、その種のトラブルは原因や解決方法を調査するのが困難な傾向があります。
そこで本章では、OSIV系システムのアプリケーションの分散開発を実施する際に起こりやすいトラブルについて、その原因と対応方法を説明します。

7.1 資産移行上のトラブル

分散開発の対象がまったくの新規のプログラムではなく、既にOSIV系システムで動作しているプログラムである場合、分散開発に先立って、必要な資産をWindows系システムに移行する必要があります。
以下、そのような場合に問題となりやすい点について解説し、適切な回避方法がある場合はそれを示します。

7.1.1 COBOLソース・登録集原文の移行

COBOLソース・登録集原文(COPY句)は、OSIV系システムでもWindows系システムでもテキスト形式のファイルですが、システムで採用する文字コード系(OSIV系システムではEBCDIC/JEF、Windows系ではASCII/SJIS)の違いのため、ファイルを転送する過程で文字コードを変換しています。
COBOLソース・登録集原文の移行では、この文字コード変換に関してのトラブルが多く発生します。

記号、半角カナ、英小文字などの文字化け

現象

次のような記述を含むCOBOLソース・登録集原文をOSIV系システムからWindows系システムに移行したが、移行の前後でソースを見比べると一部の文字が異なっている。

図7-1 文字化けの発生する移行前のソースの一部

```
…
251000    01 社名                VALUE "Fujitsu".
…
```

図7-2 文字化けの発生した移行後のソースの同じ行

```
…
251000    01 社名                VALUE " ﾌｼﾞﾂﾎﾍﾏ ".
…
```

解説

OSIV系のシステムでは、文字コード系としてEBCDIC/JEFを採用しています。
このうち、EBCDICコード系はIBM社が考案した文字コードで、次のような文字の割り当てを持つものです。

表7-1　EBCDICコード系に共通の文字の割り当て

コードの範囲	割り当て	備考
0x00～0x3F	制御文字	未定義部分あり
0x40	空白	－
0x41～0Xef	数字以外の図形文字	未定義部分あり
0xF0～0xF9	数字	－
0xFA～0xFE	未定義	－
0xFF	制御文字	－

ただし、EBCDICコード系には多くの変種があり、OSIV系システムに使用可能なものでも、次の3

種類があります(端末の入出モードの切り換えで変更可能です)。

表7-2 OSIV系システムで使用可能なEBCDICコード系

名称	英小文字	半角カナ文字	日本語特有の記号
EBCDIC(カナ)	含まない	含む	含む
EBCDIC(ASCII)	含む	含まない	含まない
EBCDIC(英小文字)	含む	含まない	含む

同じEBCDICコード系でも、半角カナ文字を含むEBCDICと半角カナ文字を含まないEBCDICでは同じコードに対して、異なる文字を割り当てています。
例えば、X”81”はEBCDIC(カナ)では”ｱ”に、EBCDIC(ASCII)とEBCDIC(英小文字)では”a”に割り当てています。
また、記号の一部でも、対応関係の不一致から、同じコードにEBCDICの変種により違う文字が割り当ててある場合もあります。
このため、COBOLソース・登録集原文をOSIV系システムからWindows系システムに移行する際に指定する変換元の文字コード系の指定を誤ると、このような文字化けと呼ばれる現象が発生します。

回避方法

ソース移行時の文字コード変換の指定を見直してください。

日本語文字の文字化け

現象

次のような記述を含むCOBOLソース・登録集原文をOSIV系システムからWindows系システムに移行したが、移行の前後でソースを見比べると一部の文字が異なっている。

図7-3 日本語文字の文字化けの発生する移行前のソースの一部

```
…
251000        MOVE  NC”森  鷗外″  TO 表記 OF 名前.
…
```

図7-4　日本語文字の文字化けの発生した移行後のソースの同じ行

```
…
251000        MOVE  NC”森  鴎外″  TO 表記 OF 名前.
…
```

解説

OSIV系のシステムで日本語文字を表現するに用いるJEFコード系とWindows系システムで日本語を表現するのに用いるSJISコード系の違いによるものです。
大きく分けて、次の3つの問題があります。

1. 文字そのものが存在しない。
   SJISとJEFでは使用可能な文字数がまったく異なります(“付録F 文字コード系”参照)。
   使用可能な文字はJEFのほうが圧倒的に多いため、SJISに該当する文字がない場合、文字化けが発生します(“■”で表示される場合が多い)。
2. 同じ自体の文字は存在しないが、異字体が存在する。
   文字コードに関するJIS規格の1983年の改定(83JISと呼ぶ)は、それ以前のもの(78JISと呼ぶ)から大きな変更が行われました。なかでも文字の字体のみを変更したものが多く存在します。JEFは78JISを元に作成したもので、かつ、互換性を重視したため、83JISで変更した文字の字体は別のコードに割り当てることになりました(JEFは78JIS/83JIS両方の字

体を含む)。
このような経緯から、78JISに従った旧字体の文字を使用していた場合、Windows系システム用に文字コード変換をする際に、文字が異字体に変換されてしまいます。

3. 新旧2種類の字体が存在するが、その対応関係が適切でない。
JIS規格で規定する文字コードに、1つの文字に対する2つの字体が両方とも含まれている場合が少数存在します。1983年の改定で、そのような文字の2つの字体とコードの割り当てが交換したものがあります。JEFでは、そのような入れ換えは行われなかったため、コード変換により、文字が異字体に変換されてしまいます。

対処方法

問題の原因が2)または3)の場合、Windows系システムで開発したソースをOSIV系システムに登録する際に、同等の文字コード変換が行われ、移行前に使用されていた文字が復元されるため、最終的には問題となりません。ただし、単体テストまでWindows系システム上で行う場合、テストデータの作成やテスト結果の確認などで、どの文字がどの文字に置き換わったか意識する必要があります。
一方、1)の問題はOSIV系システム/Windows系システムのサポートする文字コード系の根本的な性質の違いからに関する問題ですが、以下の方法で回避可能である場合があります。

- Windows系システムの外字として、問題の発生する文字を登録する。
- ADJUSTやCharsetMGRなどの文字コード変換の設定で、登録した外字と問題の発生する文字の変換を定義する。

固定形式ソースのプログラム識別番号領域の問題

現象

次のような固定形式で、プログラム識別番号領域(73～80カラム目)に文字が含まれるCOBOLソース・登録集をOSIV系システムからWindows系システムに移行すると、次のような見た目そのままのソースが移行されてしまう場合があります。

図7-5 日本語を含む固定形式ソースの移行前後でのソースの見え方

```
----+----1----+----2----+----3----+----4 …+----7----+-----
000100 IDENTIFICATION DIVISION.                      20030710
000200 PROGRAM-ID.    TEST1.                         20030710
000300 DATA           DIVISION.                      20030710
000400 WORKING-STORAGE SECTION.                       20030710
000500 PROCEDURE      DIVISION.                      20030710
000600      DISPLAY NC"日本語"                       20030710
```

プログラム識別番号領域は正しく73カラム目から始まる必要があるので、Windows系システムへ移行後のソースを正しく翻訳できません。

解説

OSIV系のシステムの日本語エディタで日本語文字を入力した場合、日本語文字の前後にA/Kシフトコードと呼ぶ不可視のコードが自動的に挿入されています。このため、上の例の600行目のプログラム識別番号領域は見た目はずれているに関わらず73カラム目から始まります。しかし、このソースをWindows系システムに移行する際、通常のコード変換では単にA/Kシフトコードが削除され、600行目の文字列”20030710”は71カラム目から始まるものになってしまいます。

対処方法

NetCOBOLのプロジェクトマネージャが分散開発を支援するために提供するファイルの受信機能を使用することで、この問題は基本的に回避可能です。

［受信］ダイアログで“固定長のCOBOLソースまたは登録集の受信”を必ずチェックしてください。プログラム識別番号領域に含まれていた文字列が73カラム目から配置されるように適切な数の空白を72カラム目より前に補います。

図7-6　プログラム識別番号領域の補正

```
----+----1----+----2----+----3----+----4 …+----7----+-----
 000100 IDENTIFICATION DIVISION.                    20030710
 000200 PROGRAM-ID.     TEST1.                      20030710
 000300 DATA            DIVISION.                   20030710
 000400 WORKING-STORAGE SECTION.                    20030710
 000500 PROCEDURE       DIVISION.                   20030710
 000600       DISPLAY NC"日本語"                    20030710
```

“　“部分が補った空白

複数行に継続する文字定数に日本語文字が含まれている場合、この方法では問題を回避することはできません。

7.1.2 フォーマット定義体の移行

フォーマット定義体は、PSAMで処理するOSIV系システム固有の形式で、Windows系システム上で使用される画面帳票定義体(MeFtで処理する)と異なる形式をもつものです。PSAMとMeFtの間には大小さまざまな非互換が存在することから、フォーマット定義体から画面帳票定義体への変換と移行に関してはさまざまなトラブルが発生します。

項目制御部のサイズの変更

現象

フォーマット定義体ソースで項目制御部を5バイト以外となるように定義してあっても移出機能(ADDFORM)を使用して、画面帳票定義体に変換すると無条件に項目制御部が5バイトとなってしまう。

図7-7 項目制御部を1バイトとするフォーマット定義体ソース例

```
 …
   HOKKAIIN RECORD    TYPE=INOUT DUSAGE=CTRL1
     INO      DATA      USAGE=GRP NAME=INPUT-DATA OCCURS=3
 …
```

解説

画面帳票定義体は、項目制御部が5バイトであるもののみがシステム間の流通形式として認められます。このため、移出機能(ADDFORM)によって画面帳票定義体に変換する場合も項目制御部は5バイトとなってしまいます。

対処方法

一般的な対処方法はありません。

PSAM/MeFtの機能差からくる移行時の変換エラー

現象

フォーマット定義体ソースから移出機能(ADDFORM)を使用して、画面帳票定義体に変換するジョ

ブでエラーや警告が出力される。

図7-8 ADDFORMのジョブのエラー出力例

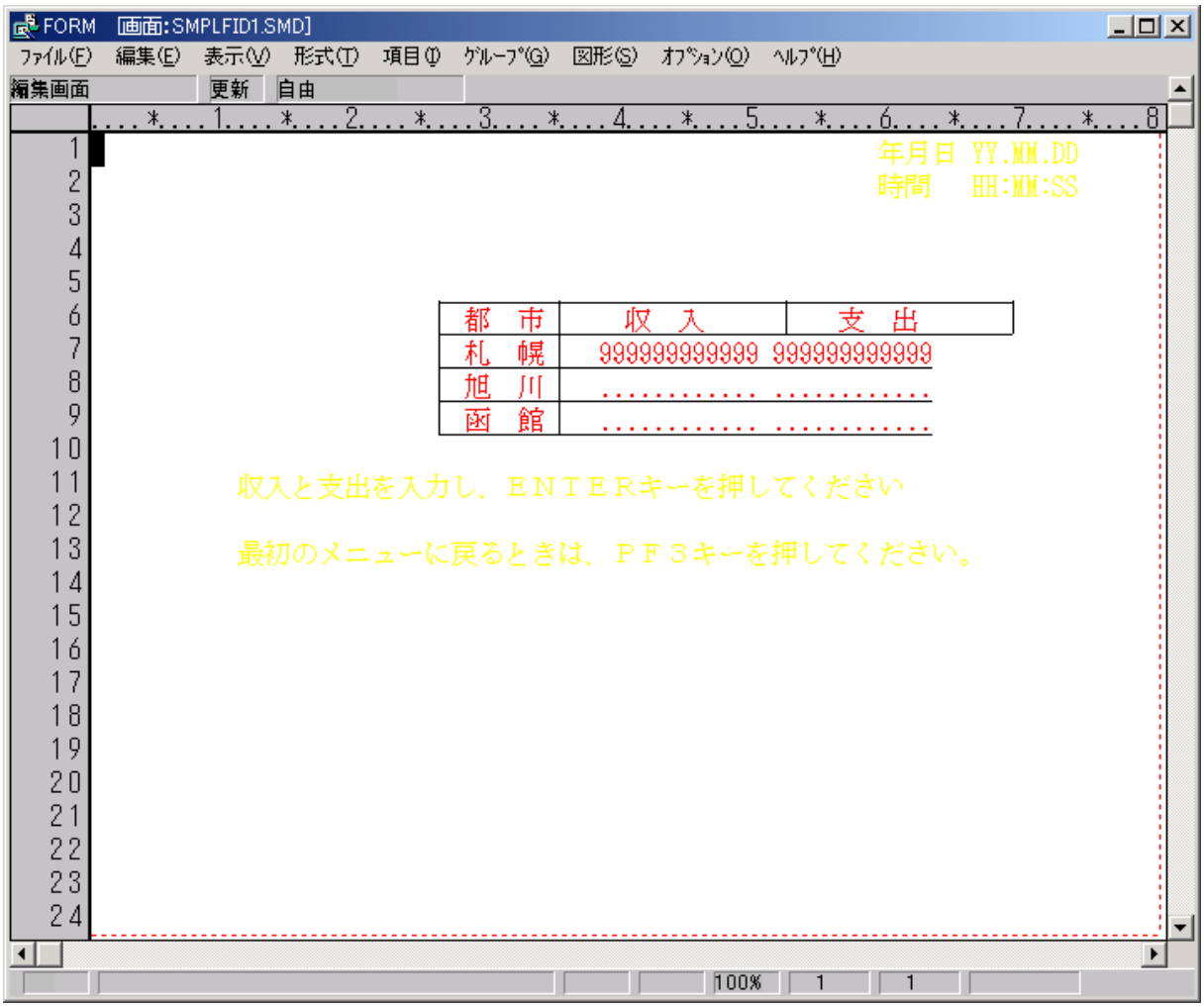
```
JYBR192I-I CONVERSION OF THE FORMAT(SMPLFID1) HAS STARTED.
JYBR106I-W AN INVALID OPERAND(AID) WAS SPECIFIED FOR THE USAGE OPERAND DATA STATEMENT.
JYBR106I-W AN INVALID OPERAND(MSG1) WAS SPECIFIED FOR THE USAGE OPERAND DATA STATEMENT.
JYBR147I-W THE POSITION(7,48) IS NOT ALLOWABLE OF THE FIELD OR THE WINDOW.
  …
```

解説

PSAMとMeFt、フォーマット定義体と画面帳票定義体の機能差のため、ADDFORMでの変換処理が失敗しています。
重度の変換エラーが発生した場合、画面帳票定義体は生成されません。発生したエラーが警告レベルの場合、画面帳票定義体は生成されますが、見た目などが異なるものとなります。

図7-9 IFDによるフォーマット定義体の画面表示の確認結果

図7-10 一部の変換エラー後に生成された画面帳票定義体のFORMによる表示例



対処方法

一般的な対処方法はありません。

変換エラーが発生するのが、画面レイアウトに関する部分のみなら見た目の問題はありますが、分散開発の対象とすることが可能です。しかし、PSAMのレコード定義に関する部分にエラーが発生している場合、変換によって得られた画面帳票定義体にエラーの発生したデータ項目の情報が含まれていないため、分散開発には使用できません。

7.2 プログラミング時のトラブル

7.2.1 翻訳チェック

Windows系システムでCOBOLソースおよび登録集原文(COPY句)を作成し、翻訳チェックを行うまでの作業で陥りやすいトラブルについて説明します。

COBOLソースの正書法の指定誤り

現象

プログラム識別番号領域域(73～80カラム)に空白文字以外を含む場合、翻訳エラーとなる。

図7-11 固定形式の正書法で記述されたCOBOLソースの例と翻訳結果

```
----+----1----+----2----+----3----+----4----+----5 …     7----+----8
000100 IDENTIFICATION DIVISION.                              00000000
000200 PROGRAM-ID.      PG01.                                00000000
000300 ENVIRONMENT      DIVISION.                            00000000
…
```

```
JMN1356I-W  見出し部中に誤った語'00000000'が指定されています.
JMN1123I-S  許されない語'00000000'が現れました.次の認識できる
            句, 段落, 節または部まで無効になります.
```

解説

富士通のCOBOL製品は、次の2種類のソース記述形式(正書法)をともにサポートしています。

- 固定形式(上記の例の形式)
- 可変形式

ホストではソースを格納したデータセットの形式から、どの記述形式が用いられているかをCOBOLコンパイラが自動的に判定します。しかし、Windows系のシステムではテキスト形式のファイルは改行に区切られた可変形式しか存在しません。このため、NetCOBOLはデフォルトではソースをすべて可変形式であるものとして扱います。固定形式のソース・登録集原文を入力とする場合、翻訳オプションSRF(Source Reference Format)で指定する必要があります。

対応方法

翻訳オプションにSRF(FIX,FIX)を指定してください。

日本語定数中の英数字空白文字の問題

現象

OSIV系のシステムのCOBOL85では正常に翻訳ができていたCOBOLソースの日本語定数が翻訳エラーとなる。

図7-12 日本語定数中に英数字空白文字を含むソースの例と翻訳結果

```
…
 DATA            DIVISION.
 WORKING-STORAGE SECTION.
```

```
 01 日本語項目   PIC N(10) VALUE NC"あ　い".
                        …
```

JMN1672I-S　日本語定数の中に日本語の文字に変換できない文字があります. 日本語定数を1字の空白とみなします.

解説

OSIV系システムのEBCDIC/JEFコード系では、日本語空白の内部表現(X”4040”)は、2個分の英数字空白(X”40”)と等価です。このため、COBOLコンパイラは日本語定数中の英数字空白について、あまり厳密なチェックを行っていません。
しかし、Windows系システムで使用されるSJISコード系では、英数字空白の内部表現(X”20”)と日本語定数の内部表現(X”8140”)はまったく異なっているため、単純に英数字空白2個分を日本語空白１個分と扱うことができないため、厳密なチェックを行います。

対応方法

NetCOBOL JEFオプションを使用することで、この問題は回避可能です。JEFオプションは日本語定数中の英数字空白をOSIV系のCOBOL85と同等に処理します。
NetCOBOL JEFオプションを使用しない場合は、翻訳エラーが指摘する行の日本語定数中の英数字空白2文字を日本語空白1文字となるように修正する必要があります。NetCOBOL では、この問題をチェックして修正したソースを出力するためのツール(CNVNSP)が製品媒体中に含まれています。より詳細な情報については、NetCOBOL の製品媒体に含まれる説明書(CNVNSP.TXT)を参照してください。

日本語16進文字定数

現象

OSIV系のシステムのCOBOL85では正常に翻訳ができていたCOBOLソースの日本語16進定数が翻訳エラーとなる。

図7-13 不適切な値を持つ日本語16進文字定数を含むソースの例と翻訳結果

```
…
 DATA           DIVISION.
 WORKING-STORAGE SECTION.
 01 日本語項目   PIC N(10) VALUE NX"F120".
…
```

JMN1672I-S　日本語定数の中に日本語の文字に変換できない文字があります. 日本語定数を1字の空白とみなします.

解説

OSIV系のシステムで日本語文字を表現するに用いるJEFコード系とWindows系システムで日本語を表現するのに用いるSJISコード系の違いによるものです。
JEFコード系のほうが使用できる文字の種類が多く、かつ、より多くの外字の定義も可能となるため、SJISコード系ではサポートしていないコード値となっています。

対応方法

翻訳オプションFLAGSW(GSS)またはFLAGSW(GSW)を指定してください。
FLAGSW(GSS)、FLAGSW(GSW)の指定時、NetCOBOLはこの記述に関するエラーチェックを抑止します。
NetCOBOL JEFオプションを使用することで、この問題は回避可能です。JEFオプションでは、この日本語16進文字定数の値をそのまま受け入れます。

予約語の非互換

現象

利用者語(データ名、ファイル名、ラベル、呼び名等)の定義、参照が翻訳エラーとなる。

図7-14 NetCOBOLの予約語をデータ名に使用するソースの例と翻訳結果

```
…
000500 WORKING-STORAGE SECTION.
000000 01 CRT          PIC X.
000600 PROCEDURE       DIVISION.
000000 INTERFACE.
…
```

JMN1376I-S　レベル番号'01'の直後に許されない語'CRT'が指定されています．このデータ記述項を無名項目とみなします．

JMN1004I-W　予約語'INTERFACE'は，B領域から書き始めなければなりません．B領域から書き始められているものとみなします．

解説

エラーの発生する語は、NetCOBOLの予約語となっています。NetCOBOLは、何度かの機能エンハンスにともない予約語が追加されました。このため、デフォルトではOSIV系のCOBOL85では予約語でなかった語のいくつかを予約語として解釈します。

なお、追加された予約語と予約語セットについては“A.4 予約語”を参照してください。

対応方法

COBOLでは、互換性を考慮して予約語は特定のバージョンの予約語の集合を予約語セットとして定義し、翻訳オプションRSVで指定して、翻訳時に使用する予約語セットを選択することができます。ホスト系のCOBOL85と同じ予約語セットを使用する場合、翻訳オプションの指定にRSV(V122)を追加してください。

ASSIGN句の指定の非互換

現象

ASSIGN句に指定したファイル識別名が翻訳エラーとなる。

図7-15 ファイル識別名と同名のデータ項目を含むソースの例と翻訳結果

```
…
000300 ENVIRONMENT    DIVISION.
000400 INPUT-OUTPUT    SECTION.
000500 FILE-CONTROL.
000600     SELECT SQFILE1 ASSIGN TO SQF1
000700       ORGANIZATION   IS SEQUENTIAL
000800       FILE STATUS    IS WFS1.
….
001000 DATA           DIVISION.
001100 FILE            SECTION.
001200 FD SQFILE1.
001300 01 SQF1.
001400   02                  PIC X(80).
…
```

JMN2954I-S　ファイル名'SQFILE1'のASSIGN句のデータ名は，項類が英数字の256文字までのデータ項目でなければなりません．かつ，作業場所節または連絡節に定義されていなければなりません．

解説

OSIV系のCOBOL85では、ASSIGN句に指定する名前はDD名(MSP)/アクセス名(XSP)と対応づけるファイル識別名だけを指定するもので、これと同じ名前をデータ名等に使用することができました。一方、NetCOBOLでは、”ASSIGN TO データ名”の書き方(使用するファイルの名前をプログラム中で動的に変更する機能)を新たにサポートしたため、ASSIGN句に指定した名前と同じ名前のデータ名を不用意に使用すると翻訳エラーや実行時エラーが発生します。

対応方法

翻訳オプションFLAGSW(GSS)またはFLAGSW(GSW)を指定してください。
FLAGSW(GSS)、FLAGSW(GSW)の指定時、同じ記述がOSIV系のCOBOL85とNetCOBOLで異なる意味に解釈可能な場合は、OSIV系のCOBOL85と同じ意味で解釈します。

ファイル編成の認識

現象

ASSIGN句に指定したファイル識別名による編成の指定が有効とならず、思わぬ翻訳エラーとなる。

図7-16 ファイル識別名による編成指定を含むソースの例と翻訳結果

```
…
000600     SELECT IXFILE1 ASSIGN TO I-SYS001
000700       RECORD KEY     IS WRKY1.
…
```

JMN1322I-S　索引ファイル以外に，RECORD KEY句は指定できません．

解説

OSIV系のCOBOL85では、ASSIGN句に指定するファイル識別名の一部は次の形式を持ちます。
編成－名前

この場合、編成を表す文字によって、実際のファイル編成が決定されます。以下に編成を指定する文字とファイル編成の関係を示します。

表7-3 ファイル編成文字の一覧

編成文字	実際のファイル編成
S	順編成ファイル
R	相対編成ファイル
D	直接編成ファイル
W	
I	インデックス付き編成ファイル

NetCOBOLでは、この形式のファイル識別名はサポートしていません。編成を表す文字は無視されORGANIZATION句の指定がなければ、順編成のファイルとして扱います(名前の指定の部分は有効となる)。このため、特定の編成のファイルにしか指定できない句の記述が翻訳エラーとなります。

対応方法

翻訳オプションFLAGSW(GSS)またはFLAGSW(GSW)を指定してください。
FLAGSW(GSS)、FLAGSW(GSW)の指定時、同じ記述がOSIV系のCOBOL85とNetCOBOLで異なる意味に解釈可能な場合は、OSIV系のCOBOL85と同じ意味で解釈します。

連絡節データ

現象

連絡節データ項目を定義、参照するプログラムの翻訳でこれまでと違ったエラーが出力されるようになる。
対象となるプログラムを主プログラムとして翻訳した場合、以下のような結果となる。

図7-17 連絡節データの使用法が不適切なプログラム例と翻訳結果(主プログラム時)

```
…
000500 LINKAGE          SECTION.
000600 01 LK1          PIC X(10).
000700 01 LK2          PIC X(10).
000800 01 LK3          PIC X(10).
000900 PROCEDURE       DIVISION USING LK1 LK2.
…
001500        DISPLAY LK3
…
```

JMN5595I-S　主プログラムの手続き部見出しのUSING指定に記述したパラメタは､大きさが102バイトを越えないただ1つの集団項目で､従属する最初の基本項目が2バイトの2進数項目でなければなりません.指定されたパラメタは無効になります.

JMN3483I-S　主プログラムの手続き部見出しのUSING指定に記述したパラメタは､大きさが102バイトを越えないただ1つの集団項目で､従属する最初の基本項目が2バイトの2進数項目でなければなりません.指定されたパラメタは無効になります.

同じプログラムを主プログラムとせずに翻訳した場合、以下のような結果となる。

図7-18　連絡節データの使用法が不適切なプログラム例と翻訳結果(主プログラム時以外)

```
…
000500 LINKAGE          SECTION.
000600 01 LK1          PIC X(10).
000700 01 LK2          PIC X(10).
000800 01 LK3          PIC X(10).
000900 PROCEDURE       DIVISION USING LK1 LK2.
…
001500        DISPLAY LK3
…
```

JMN3482I-S　連絡節に定義された'LK3'はPROCEDURE DIVISIONのUSING指定またはRETURNING指定,またはENTRY文のUSING指定に記述しなければなりません.

解説

NetCOBOLでは連絡節で定義したデータ項目の参照についてのチェックを強化しました。連絡節で定義したデータ項目の誤った使い方が原因となるプログラムの障害は、しばしば異常の現れ方が多様で、かつ、現象の再現性が乏しいため、原因の調査がひどく困難となる傾向が強いためです。

表7-4 連絡節データの使用法と誤り時の診断メッセージ

プログラムの種別	正しい使用法	誤った使用に対する診断メッセージ
主プログラム	起動パラメタ(GS形式)を受け取る	JMN3483I-S or JMN5595I-S
	FCOM/UWAを受け取る	
	ADDRESS OF特殊レジスタの作用対象として使用	
主プログラム以外	CALL文から渡されるパラメタを受け取る	JMN3482I-S
	FCOM/UWAを受け取る	
	ADDRESS OF特殊レジスタの作用対象として使用	

GS形式の起動パラメタは、次の構造を持ちます(“パラメタ文字列”は100バイト以下なら任意の定義が可能)。

```
01 パラメタ
  02 パラメタ文字列長       PIC S9(4) BINARY.
  02 パラメタ文字列         PIC X(1)  OCCURS 1 TO 100
                                       DEPENDING ON パラメタ文字列長.
```

OSIV系のCOBOL85では、主プログラムと他のプログラムから呼び出されるプログラムの区別がありません。このため、このようなチェックをすることができませんでした。

対応方法

JMN3482I-Sが出力される場合、プログラムの処理に誤りがあります。プログラムの処理を見直してください。

主プログラムに対して、JMN3483I-SまたはJMN5595I-Sが出力される場合については、翻訳オプションFLAGSW(GSS)またはFLAGSW(GSW)を指定することで、この診断メッセージの出力を抑止することができます(ただし、JEFオプションでは抑止できません)。

埋め込みSQL文の解析

現象

埋め込みSQL文が正しく翻訳されず、思わぬ翻訳エラーとなる。

図7-19 SymfoWARE/RDBII固有のSQL文を含むソースの例と翻訳結果

```
…
010600      EXEC SQL
010700        START SQL
010800      END-EXEC.
…
```

```
JMN2633I-S  SQL文に誤りがあります. ODBC-7600E 実行不可能なSQL
            文です.
```

解説

OSIV系のCOBOL85では、埋め込みSQL文の解析にはSymfoWARE/RDBのSQLパーザを使用します。一方、

NetCOBOLでは、製品に組み込まれているSQLパーザを使用しますが、このSQLパーザは次の点でSymfoWARE/RDBのSQLパーザと違いがあります。

- 使用可能な埋込みSQL文の種類
  SymfoWARE/RDB固有の埋込みSQL文は、翻訳エラーになります。
  例:START SQL文、END SQL文
- 使用可能な埋込みSQL文の長さの制限
  1つの埋め込みSQL文として使用可能な長さに制限があります。長さが16386バイトを超える埋込みSQL文を使用すると実行時エラーになります。

NetCOBOL では、製品単体でリレーショナルデータベース機能をサポートしますが、これはMicrosoft社の提唱によるODBC (Open Database Connectivity)インタフェースを使用するものです。このため、ODBCインタフェースをサポートする多くのデータベース製品に接続可能ですが、OSIV系システムでの動作と細かな違いがあります。

対応方法

データベース製品として、富士通のSymfoWAREを使用します。埋め込みSQL文の翻訳の際もSymfoware Serverに含まれるプリコンパイラを使用します。

7.2.2 リンク

Windows系システムで単体テストを実施するために、翻訳したCOBOLプログラムをリンクする作業で陥りやすいトラブルについて説明します。

主プログラムの指定漏れ

現象

次のメッセージボックスが表示されて、COBOLプロジェクトマネージャで“ビルド”操作が実行できない。

図7-20 主プログラム指定漏れ時のエラーメッセージ

解説

OSIV系システムでは、オペレーティングシステムから直接呼び出されるプログラムと他のプログラムから呼び出されるプログラムに、オブジェクトファイルの構造上の違いはありません。
しかし、Windows系システムでは、オペレーティングシステムから直接呼び出されるプログラムと他のプログラムから呼び出されるプログラムは異なる特別なコードが含まれている必要があります。
プロジェクトの最終ターゲットの種類により、主プログラムの指定の必要性は異なります。

- 実行可能ファイル(EXEファイル)
  プロジェクトに含まれる1つのCOBOLソースファイルが主プログラムとして指定されている必要があります。
- ダイナミックリンクライブラリファイル(DLLファイル)
  主プログラムの指定は必要ありません。

なお、COBOLプロジェクトマネージャで“ビルド”操作以外(WINLINKあるいはLINK.EXEなどを使用

して)で直接オブジェクトをリンクしようとした場合は次のようなメッセージが出力されるのも同じ現象です。

図7-21 主プログラムの指定なしで発生するリンク時のエラー例

```
LINK : fatal error LNK1561: ｴﾝﾄﾘｰ ﾎﾟｲﾝﾄを定義しなければなりません
```

対応方法

主プログラムの設定をしてください。COBOLプロジェクトマネージャでの操作については、“基本的なプロジェクトファイルの作成”の“主プログラムの設定”を参照してください。

主プログラムの指定誤り

現象

プログラムのリンクの結果、次のいずれかのエラーが発生する。

図7-22 主プログラムの指定誤りで発生するリンク時のエラー例1

```
EXPGM2.OBJ : error LNK2005: _WinMain@16 はすでに EXPGM1.OBJ で定義されています
EXPGM1.EXE : fatal error LNK1169: 1 つ以上の複数回定義されているｼﾝﾎﾞﾙが見つかりました
```

図7-23 主プログラムの指定誤りで発生するリンク時のエラー例2

```
EXPGM3.OBJ : error LNK2001: 外部ｼﾝﾎﾞﾙ "_WinMainCRTStartup" は未解決です
EXPGM3.DLL : fatal error LNK1120: 外部参照 1 が未解決です。
```

解説

主プログラムの指定に誤りがあります。上にしめしたエラーが発生するのはそれぞれ次の誤りによるものです。

- エラー例1
  実行可能ファイル(EXEファイル)を構成するCOBOLプログラムに主プログラムの指定をもつものが複数あります。
- エラー例2
  ダイナミックリンクライブラリファイル(DLLファイル)を構成する1つ以上のCOBOLプログラムに主プログラムの指定があります。

COBOLプロジェクトマネージャで主プログラム設定の操作をする以上、このような問題が発生することはありません。主プログラムの指定は翻訳オプションMAINによりますが、これが@OPTIONS翻訳指示文などで直接ソースファイル中に記述されている可能性があります。

対応方法

不要な主プログラムの設定を削除します。基本的に@OPTIONS翻訳指示文に翻訳オプションMAINを指定しないでください。

呼び出すプログラムが見つからない

現象

プログラムのリンクの結果、次のエラーが発生する。

図7-24 呼び出すプログラムが見つからない場合、発生するリンク時のエラー例

```
EXPGM1.OBJ : error LNK2001: 外部ｼﾝﾎﾞﾙ "_SUBPGM1" は未解決です
EXPGM1.EXE : fatal error LNK1120: 外部参照 1 が未解決です。
```

解説

CALL文で呼び出すプログラム(上記の例では“SUBPGM1”。メッセージに現れる名前の先頭のアンダースコアは名前に含まれないので注意すること)が見つからないためです。原因として次のことが考えられます。

- 該当するプログラムがプロジェクト内に含まれない
- プロジェクトに含まれるCOBOLソース中に複数の外部プログラム(他のプログラムに含まれないプログラム)が含まれている場合

対応方法

原因および作成しようとするプログラムのプログラム構造により対処方法が異なります。

- 静的構造の場合

  必要とするすべてのプログラムのオブジェクトを結合して、単一の実行可能ファイルを作成した場合のプログラム構造です。それぞれ、以下の方法で対応します。

  — 該当するプログラムがプロジェクト内に含まれない場合:
  必要なプログラムのCOBOLソースをプロジェクトに追加します。

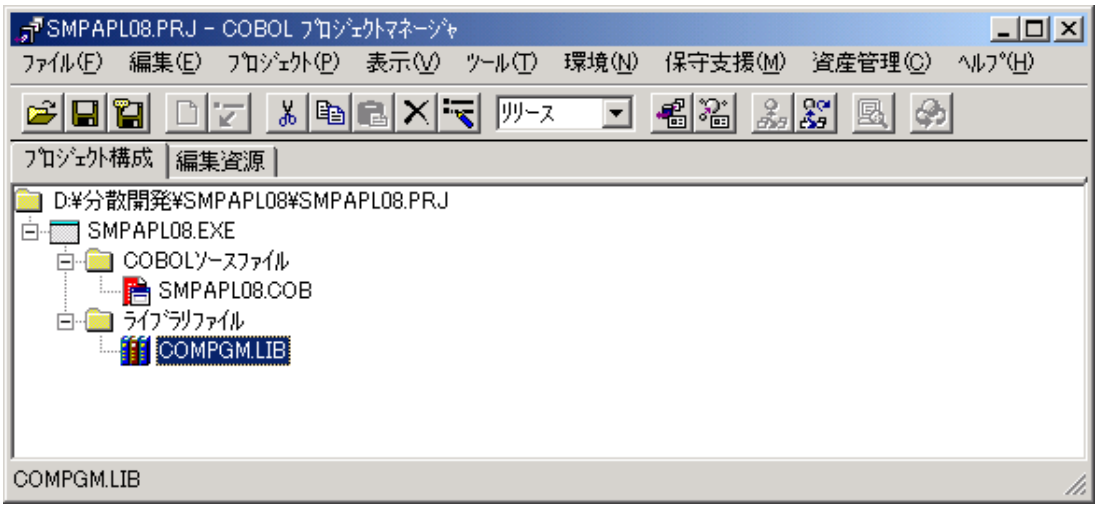
  — プロジェクトに含まれるCOBOLソース中に複数の外部プログラムがある場合:
  基本的にはソースファイルをCOBOLプログラム毎に分割して、それぞれをプロジェクトに登録しなおすべきです。

- 動的リンク構造

  呼び出されるプログラムをダイナミックリンクライブラリ(DLLファイル)とし、呼び出すプログラムにはインポートライブラリ(LIBファイル)を結合して実行可能ファイルを作成した場合のプログラム構造です。

  この場合、以下の図のように“ライブラリファイル”フォルダをプロジェクトに追加し、インポートライブラリ (LIBファイル)を追加します。インポートライブラリは、ダイナミックリンクライブラリ(DLLファイル)を作成した際に、同じフォルダに作成されています。

図7-25 ライブラリファイルを追加したプロジェクト



7.3 単体テスト時のトラブル

7.3.1 COBOLプログラムのふるまい

Windows系システムで単体テストを実施する場合、COBOLプログラムの動作がOSIV系システムで期待される動作と異なる場合があります。このようなトラブルについて説明します。

文字の大小順序の逆転

現象

文字の大小順序の比較で、OSIV系システムで期待する通りの結果が得られない。
例えば、次の例でOSIV系システムでは“EBCDIC/JEF”と表示されることを期待できるが、Windows系システムでは“ASCII/SJIS”と表示される。

図7-26 文字の大小比較結果が逆転するソース記述例

```
01  英数字     PIC  X  VALUE ″A″.
01  数字       PIC  9  VALUE 1.
          :
     IF     英数字 < 数字  THEN
            DISPLAY  ″EBCDIC/JEF ″
     ELSE
            DISPLAY  ″ASCII/SJIS″
     END-IF.
```

解説

システムの標準の文字コード系の違いから発生する問題です。OSIV系システムにおけるEBCDIC/JEFコード系と、Windows系システムにおけるASCII/SJISコード系では、次の点で文字の大小順序が異なります。

- 英字、数字、カナの大小順序が逆転します。
- 外字と外字以外の日本語の大小順序が逆転します。

この文字の大小順序の逆転は次のような操作の結果に影響を与えます。

- IF文、EVALUATE文、PERFORM文などの比較条件の結果
- 索引ファイルのSTART文でのキー比較の結果
- SORT文またはMERGE文による整列併合の結果

対応方法

比較対象の文字が1バイト文字に限定されるなら、次のようにすることで索引ファイルのキー順以外の問題は回避することが可能です。

1. 特殊名段落のALPHABEL句でEBCDICを指定して符号系名を定義する。
2. 上記の符号系名を実行計算機段落のPROGRAM COLLATING SEQUENCE句に指定する。

図7-27　PROGRAM COLLATING SEQUENCE句による文字の大小比較結果が逆転の回避例

```
        …
CONFIGURATION   SECTION.
OBJECT-COMPUTER. GS8000
```

```
        PROGRAM COLLATING SEQUENCE GS-PCS.
SPECIAL-NAMES.
      ALPHABET
      GS-PCS IS EBCDIC.
…
```

あるいは、NetCOBOL JEFオプションを使用することで、この問題は回避可能です。JEFオプションを使用する場合、索引ファイルのキー順や日本語文字の大小順序も含めて、OSIV系システムと完全に同じ順序で文字の大小順序が評価されます。

日本語を含む集団項目に関する比較結果

現象

日本語項目を含む集団項目と他のデータ項目、あるいは定数との比較結果がOSIV系システムで期待される結果と異なる。
以下に示すのは、その最も典型的な例である。

図7-28 日本語を含む集団項目に関する比較結果が異なるソース記述例1

```
        …
01 GRP.
  02 N-DATA1 PIC N(3).
  02 N-DATA2 PIC N(3).
    …
    MOVE SPACE TO GRP.
    IF N-DATA1 = SPACE THEN
      DISPLAY "OK"
    ELSE
      DISPLAY "NG"
    END-IF.
      …
```

解説

OSIV系システムのEBCDIC/JEFコード系では、日本語空白文字の内部表現(X”4040”)は、2個分の英数字空白文字(X”40”)と等価です。このため、上記の例の結果は“OK”と表示されることになります。しかし、Windows系システムでは日本語空白文字は、英数字空白文字の間にそのような関係はありません(X”8140”とX”20”)。上記の例では、N-DATA1が日本語項目であるため、比較条件は日本語比較となり、SPACEは日本語空白文字とみなされます。実際には英数字空白文字がN-DATA1に格納されているため、Windows系システムでの実行結果は“NG”と表示されることになります。
また、次のような例でも結果が異なります。

図7-29 日本語を含む集団項目に関する比較結果が異なるソース記述例2

```
        …
01 GRP1.
  02 N-DATA1 PIC N(5) VALUE NC”日本語”.
01 GRP2.
  02 N-DATA1 PIC N(3) VALUE NC”日本語”.
      …
      IF GRP1 = GRP2 THEN
        DISPLAY "OK"
```

```
        ELSE
          DISPLAY "NG"
        END-IF.
        …
```

対応方法

問題となる比較が次のような条件に該当しないなら、翻訳オプションとしてNSPCOMP(ASP)を指定することによって、この問題を回避することが可能です。

- 日本語項目を含まない集団項目どうしの比較
- 明または暗に属性が表示用でない項目を含む集団項目の比較
- INSPECT文、STRING文、UNSTRING文および索引ファイルのキー操作で行われる比較

あるいは、NetCOBOL JEFオプションを使用することで、この問題は回避可能です。

外部10進項目の再定義

現象

次のように外部10進項目を再定義している場合、どちらの項目に対して操作をする場合でもOSIV系システムで期待される動作と異なる。

図7-30 外部10進項目の再定義例を含むソースの例

```
        …
01 GRP1.
  02 NUM-AREA  PIC S9(9) DISPLAY.
  02 R-NUMN-AREA REDEFINES NUM-AREA
        …
```

解説

OSIV系システムのCOBOL85と、Windows系システムのNetCOBOLでは外部10進項目の符号位置の内部表現に違いがあります。
以下に、外部10進項目の定義と内部表現について、OSIV系システムのCOBOL85とNetCOBOLの違いを示します。

表7-5 外部10進項目の内部表現の比較一覧

PICTURE句	SIGN句	値	内部表現		備考
			COBOL85	NetCOBOL	
9(4)	なし	1234	F1F2F3F4	31323334	
S9(4)		+1234	F1F2F3C4	31323344	
		-1234	F1F2F3D4	31323354	
	LEADING	+1234	C1F2F3F4	41323334	
		-1234	D1F2F3F4	51323334	
	TRAILING	+1234	F1F2F3C4	31323344	
		-1234	F1F2F3D4	31323354	
	LEADING SEPARATE	+1234	4EF1F2F3F4	2B31323334	
		-1234	60F1F2F3F4	2D31323334	
	TRAILING SEPARATE	+1234	F1F2F3F44E	313233342B	
		-1234	F1F2F3F460	313233342D	

符号	COBOL85	NetCOBOL
ゾーン	F	3
正符号	C	4
負符号	D	5

符号	COBOL85	NetCOBOL
正符号	4E	2B
負符号	60	2D

対応方法

ふるまいの違いが問題となる箇所が明らかである場合、デバッガ上で動作させ、一次的にデータ項目の内容を書き換えて動作させることは可能です。
しかし、NetCOBOL JEFオプションを使用する以外に一般的な回避方法はありません。JEFオプションを使用する場合、外部10進項目の符号位置の内部表現はOSIV系システムのCOBOL85とまったく同じになります。

浮動小数点項目の演算結果

現象

浮動小数点項目の演算結果がOSIV系システムのCOBOL85と一致しない。

解説

OSIV系システムのCOBOL85とWindows系システムのNetCOBOLでは内部浮動小数点データ項目の内部表現に違いがあるため、仮数部の演算精度に違いがあります。
また、符号位置の内部表現も違いがあるため、外部10進データ項目を再定義しているような場合、動作が異なる場合があります。

対応方法

一般的な対処方法はありません。

FILE STATUS句の詳細情報

現象

FILE STATUS句の詳細情報に詳細情報を指定したが、値が返らない、あるいは期待した値と異なっている。

解説

FILE STATUS句の詳細情報は、表示ファイル(編成GS)、FORMAT句付き印刷ファイル(FORMAT句付き順ファイル)およびVSAMファイル(編成VS)にのみ指定することができますが、NetCOBOLでの動作はそれぞれ次のようになります。

- 表示ファイル、FORMAT句付き順ファイル
  詳細情報の値は、連携する製品あるいはシステムによって固有なコードが設定されます。
  動作するシステムの違いや連携する製品の違いにより、値は異なります。
- VSAMファイル
  NetCOBOLではVSAMファイルの実行時には、詳細情報に値が設定されません。

対応方法

一般的な対処方法はありません。
デバッガ上で、詳細情報の値を変更して動作させることはできます。

出力ファイルの内容の確認

現象

COBOLプログラムで生成したファイルをテキストエディタで開いた場合、結果が正しくないように見える。

- 1行が定義したファイルのレコード長とあっていない。
- 一部が文字化けして見える。

解説

OSIV系システムでは、COBOLで生成したファイルの各種の情報(レコード形式/レコード長等)は、システムによって管理されています。しかし、Windows系システムではそうではありません。
Windows系システムではファイルの種類は基本的に２種類しかなく、単なるテキストファイルに

あたる行順ファイル(OSIV系システムのCOBOL85はサポートしない)以外は、COBOLのランタイムを介さずにレコード形式/レコード長等を認識することはできません。

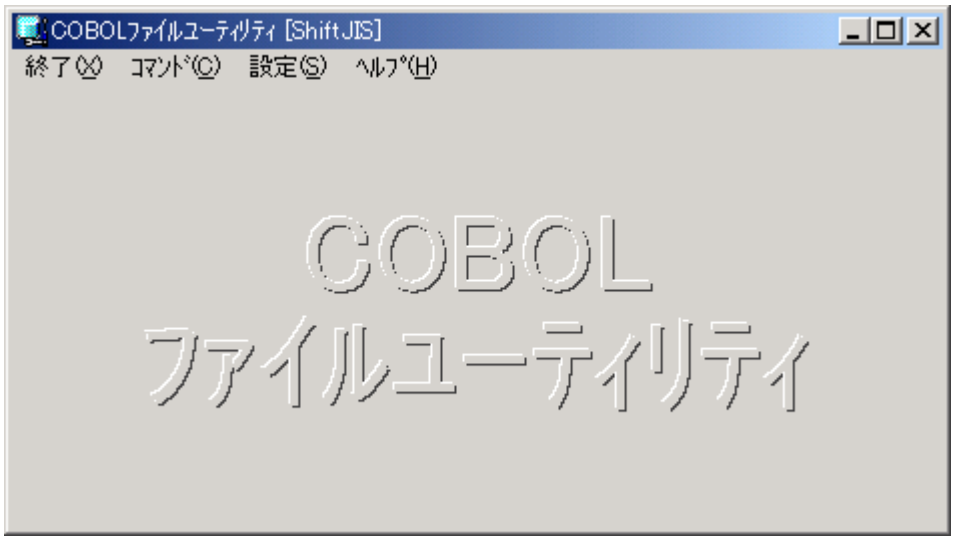
対応方法

出力されたファイルのレコードを確認するためには、NetCOBOLのファイルユーティリティを使用する必要があります。

以下、ファイルのレコード内容を確認する場合について、簡単なファイルティリティの使用法を説明します。

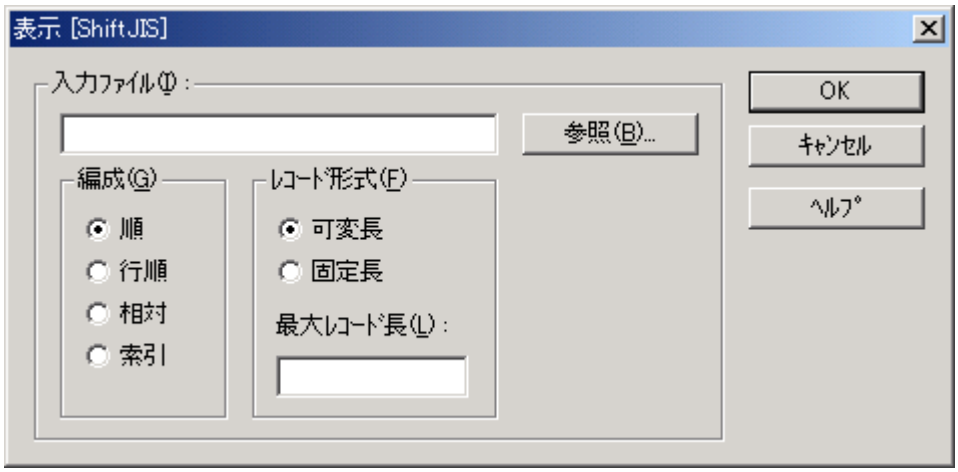
1. NetCOBOLのプロジェクトマネージャの〔ツール〕メニューから“ファイルユーティリティ”を選択すると、ファイルユーティリティが起動し、以下の画面が現れます。

図7-31 ファイルユーティリティの起動画面



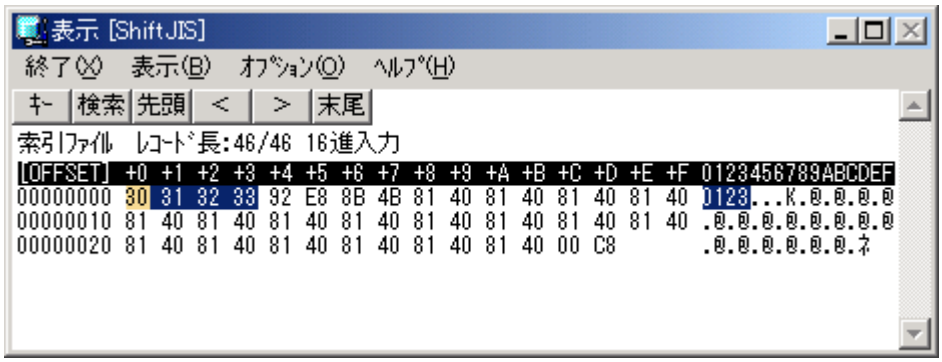
2. ファイルユーティリティの〔コマンド〕メニューから“表示”を選択するとレコードを表示するファイルの情報を設定するためのダイアログボックスが現れます。

図7-32　レコードを表示するファイルの情報を設定するためのダイアログ



3. “ファイル名”、“ファイル編成”、“レコード形式”を指定し、〔OK〕ボタンをクリックするとレコードの内容を表示するウィンドウが開きます。

図7-33　ファイルユーティリティによるレコードの表示例



印刷ファイルの再入力

現象

COBOLプログラムで生成した印刷ファイルを入力モードでオープン使用とすると実行時エラーとなる。

```
JMP0310I-U [PID:xxxxxxxx TID:yyyyyyyyy] '@1@'ファイルで'OPEN'エラーが発生しました.
                                        'OPEN-MODE '…
```

解説

以下のような定義あるいは操作によって、生成されたファイルをNetCOBOLでは、印刷ファイルと呼びます。

- ファイル管理記述項のASSIGN句にPRINTERまたはPRINTER-n(n=1～9)を指定したファイル。
- ファイル記述項にLINKAGE句が記述されているファイル。
- ADVANCING指定付きのWRITE文によって生成したファイル。

これらの印刷ファイルは、OSIV系システムのCOBOL85における物理順ファイルに相当するものですが、これを入力モードでオープンすることはできません。

対応方法

ありません。

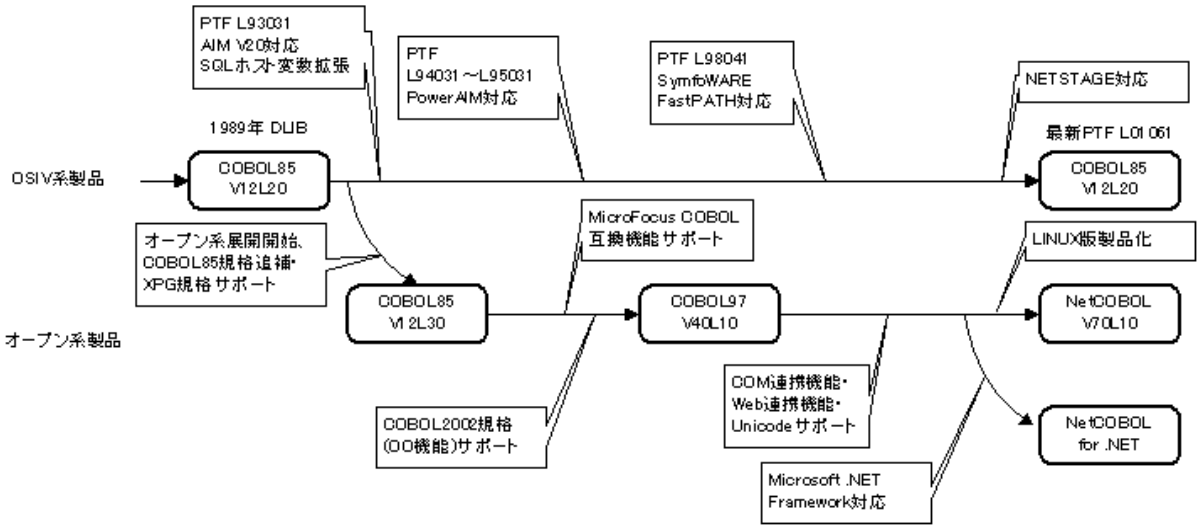
付録A OSIV系COBOLとオープン系COBOLの相違点

COBOLは互換性がきわめて重要視される言語ですが、OSVI系のシステムとWindows/UNIXなどのオープン系のシステムというまったく異なるオペレーティングシステム上では、完全な互換性を保証することはできません。このため、OSIV系のCOBOL85とオープン系のCOBOL97/NetCOBOLでは大小さまざまな仕様上、機能上の違いが存在します。
ここではCOBOL85とCOBOL97/NetCOBOLの仕様差／機能差について、いくつかのカテゴリに分けて説明します。

A.1 富士通のCOBOL製品系列

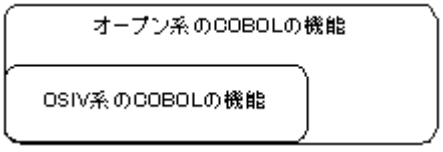
富士通のCOBOL製品は、その製品体系がOSIV系とオープン系の２系統に分かれています。
OSIV系はANSI/ISO COBOL85規格を実装したCOBOL85という製品です。この製品は、他のミドルウェアとの連携機能の強化がいくつかの点で行われていることを除けば、言語としての機能追加はまったく行われていない安定した製品です(2004年3月現在で最新はV12L20 PTF L01061)。
これに対して、オープン系製品はCOBOL85規格の1989年追補機能やXPG仕様をサポートしたCOBOL85 V12L30を皮切りに、数多くの機能エンハンスにともなってバージョンアップを行い、現在では製品名もNetCOBOLに変わっています。
以下に、富士通のCOBOL製品の製品系列を示します。

図A-1 富士通のCOBOL製品の系列



この“図：富士通のCOBOL製品の系列”で示すようにオープン系のCOBOL製品は、OSIV系のCOBOL85を元に作成されたものです。しかし、その機能差は次のような単純な包含関係にはありません。

図A-2 理想的なOSIV系COBOLとオープン系COBOLの機能差



これは、次のような理由からオープン系のCOBOLでは、OSIV系のCOBOLの機能を再現できない場合

があるからです。

- システム(オペレーティングシステム)の提供する機能に依存している。
- システムの文字コード系の違い
- 他のミドルウェアと連携して機能するが、オープン系ではそれに相当するミドルウェアが存在しない、あるいはそのミドルウェアに機能差がある。
- オープン系で機能的に意味がない。
- オープン系COBOLで追加された機能と矛盾する。

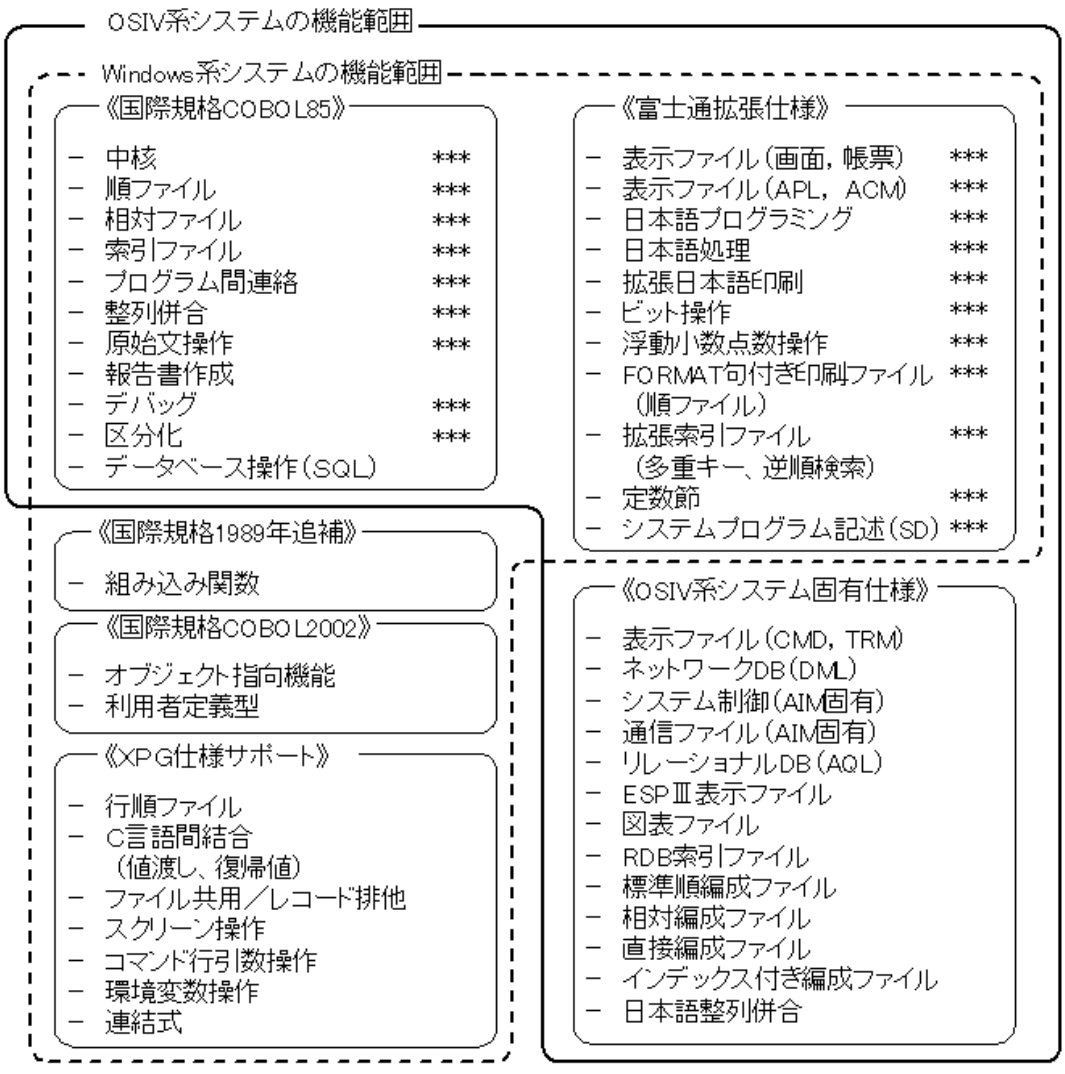
A.2 言語の機能の違い

ここでは、OSIV系のCOBOL製品とオープン系の製品の違いを言語の機能に絞って説明します。

A.2.1 概要

以下に、OSIV系のCOBOL製品とオープン系の製品がそれぞれサポートする言語の機能とその重なりを示します。

図A-3 OSIV系COBOLとオープン系COBOLの機能範囲



***: 共通仕様範囲

A.2.2 オープン系のCOBOLでは使用できない機能

以下に示す機能は、OSⅣ系のCOBOL製品に固有の機能であり、オープン系のCOBOL製品では使用できません。代替機能がある場合はその旨を示します。

表A-1 オープン系のCOBOLでは使用できない機能

No	機能	該当機能の使用の有無を判定するために有効なキーワード	代替機能	
			有無	説明
1	表示ファイル (宛先CMD、TRM、WST)	SYMBOLIC DESTINATION句に指定した定数の値が“CMD”、“TRM”または“WST” SYMBOLIC DESTINATION句に指定したデータ項目に設定する値が“CMD”、“TRM”または“WST”	×	AIMと連携する表示ファイルのため、代替機能はない。 オープン系COBOLでは使用した場合、翻訳エラー(JMN2774I-S)または実行時エラーとなる。
2	ネットワーク データベース機能	SUBSCHEMA-NAME段落 以下の文(DML)の使用 + CONNECT文 + DISCONNECT文 + ERASE文 + FIND文 + FINISH文 + GET文 + IF文 + MODIFY文 + READY文 + STORE文	×	Windows版では、ホストアプリの分散開発のため、デバッガ上でデバッグすることができるが、実際にPC/UNIXで動作するアプリケーションの開発に使用することはできない。 UNIX版では、翻訳エラーとなる。
3	システム制御機能 (AIM固有)	TRANSACTION文 USE FOR DEAD-LOCK文 USE IN TRANSACTION文	×	AIMの機能に強く依存するため、代替機能はない。 使用した場合、USE FOR DEAD-LOCK文以外は翻訳エラーとなる。
4	通信ファイル (AIM固有)	固有キーワードなし。 通常の順ファイルの操作とソース記述も違わない。	○	宛先ACMの表示ファイル(PowerRW+が必要)またはNetCOBOLの提供するプロセス間通信機能で代替は可能だが、プログラム修正が必要である。 なお、ソース記述上は順ファイル機能と区別がつかないため、特に翻訳エラーにはならない。
5	リレーショナル データベース(AQL)	EXEC AQL文 INCLUDE AQLCA	×	AIM/RDBを操作する機能であるため、代替機能はない。 使用した場合、翻訳エラーとなる。
6	図表ファイル	ASSIGN句のファイル識別名“PS-名前”形式 FORMATTED RECORD句	×	オープン系のCOBOLでは図表ファイルを認識せず、順ファイルと見なすため、順ファイルに指定できない記述は翻訳エラーとなる(例：JMN2838I-S)。
7	RDB索引ファイル	ASSIGN句のファイル識別名“DB名-VI-テーブル名”形式	×	AIM/RDBをファイルとして操作する機能であるため、代替機能はない。
8	標準順編成ファイル	ASSIGN句のファイル識別名“S-名前”形式	○	オープン系COBOLでは次の2つの固有機能はサポートしていない。

				- AFTER POSITIONING指定のWRITE文 →翻訳エラー(JMN2764I-S) - USE LABEL文 →注釈扱い(JMN2766I-W) これらを除けば、COBOLの順ファイル機能で基本的に代替可能である。 また、コンパイラが左の形式のファイル識別名指定を無視、通常の順ファイルと見なして翻訳処理を続行するため、固有機能を使用しない場合はソース修正も不要である。
9	相対編成ファイル	ASSIGN句のファイル識別名 “R-名前”形式	○	オープン系COBOLの相対ファイル機能で、基本的に代替可能である。 しかし、コンパイラが左の形式のファイル識別名の指定を無視、順ファイルと見なして翻訳処理を続行するため、不適切な翻訳エラー(JMN1309I-S)が出力される場合がある。 次のように修正することで、相対ファイルとなる。 - ファイル識別名から”R-”を除く。 - ファイル管理段落にORGANIZATION IS RELATIVE句を追加する
10	直接編成ファイル	ASSIGN句のファイル識別名 “D-名前”形式 ACTUAL KEY句	×	代替方法はない。
11	インデックス付き編成ファイル	ASSIGN句のファイル識別名 “I-名前”形式 NOMINAL KEY句	△	オープン系COBOLの索引ファイル機能で、部分的に代替可能である。 しかし、コンパイラが左の形式のファイル識別名指定を無視、索引ファイルと見なして翻訳処理を続行されるため、不適切な翻訳エラーが出力される場合がある。 次のように修正することで、索引ファイルとなる。 - ファイル識別名から”I-”を除く。 - NOMINAL KEY句を削除。 - ファイル管理段落にORGANIZATION IS INDEXED句を追加する。
12	日本語整列併合機能	以下の機能名と対応づけた呼び名を使用 + BUSHU + SOKAKU + ON-YOMI + KUN-YOMI	×	代替方法はない。 使用した場合、翻訳エラー(JMN2773I-E)となる。
13	拡張日本語定数	以下の文字に引用符文字がついた分離符で区切られる定数 + NN + NA + NK + NH	×	代替方法はない。 使用した場合、翻訳エラー(JMN1576I-S)となる。

14	索引ファイルのPOSITIONING POINTER句	POSITIONING POINTER句	×	ESPⅢ固有機能のため、代替方法はない。指定した場合は、翻訳エラー(JMN1576I-S)となる。
15	登録集名SYSDBDCT	SYSDBDCT	×	ESPⅢ固有機能のため、代替方法はない。指定しても、通常の登録集として扱われる。

A.2.3 オープン系のCOBOLでは動作の異なる機能

以下に示す機能は、ホストのCOBOL85でも、NetCOBOLでも使用可能ですが、動作が異なる場合があります。

表A-2 オープン系のCOBOLでは動作の異なる機能

No	機能	動作の異なる点	対処方法と解説
1	16進文字定数 日本語16進定数	表示／出力の結果が異なる。	【対処方法】 使用するシステムのコード系にあった値に変更する。 【解説】 MSP/XSP、Windows、Solarisそれぞれのシステムで使用する文字コード系が異なる。 実行するシステムで、特定の文字を表現するためのコードを直接記述する必要のある16進文字定数/日本語16進定数は分散開発など複数システムをまたがる開発では使用すべきではない。
2	日本語定数	大小比較の結果が異なる。	【対処方法】 なし。 【解説】 MSP/XSP、Windows、Solarisそれぞれのシステムで使用する文字コード系が異なるため。
3	文字の大小比較 - 条件式 - レコードキー(索引ファイル) - ソートキー - マージキー	文字の大小比較の結果が異なる。	【対処方法】 EBCDIC指定(以下参照)で定義した符合系名をPROGRAM COLLATING SEQUENCE句に指定する。 … CONFIGURATION SECTION. OBJECT COMPUTER. FMV6000. PROGRAM COLLATING SEQUENCE IS PCS1. SPECIAL-NAMES. ALPHABET PCS1 IS EBCDIC. … ただし、次のことに注意する必要がある。 ― 日本語の操作をするプログラムで使用すべきでない ― 索引ファイルのキー比較には有効にならない。 ― ソートキー／マージキーで使用する場合、

<table>
<tr><td></td><td></td><td></td><td>SORT/MERGE文のCOLLATING SEQUENCE指定に直接指定してもよい。
【解説】
MSP/XSP、Windows、Solarisそれぞれでシステム使用する文字コード系が異なるため。
― MSP/XSP：a～z／ｱ～ﾟ＜ A～Z ＜ 0～9
― Windows、Solaris：0～9＜ A～Z ＜ a～z ＜ｱ～ﾟ</td></tr>
<tr><td>4</td><td>外部10進項目のゾーン部の内部表現</td><td>例： PIC S9(4) VALUE +1234.
- MSP/XSP
→ X”F1F2F3C4”
- Windows/Solaris
→ X”31323344”

例： PIC S9(4) VALUE +1234 SIGN LEADING SEPARATE.
- MSP/XSP
→ X”4EF1F2F3F4”
- Windows/Solaris
→ X”2B31323334”</td><td>【対処方法】
なし。
外部10進項目を再定義している場合は要注意。
【解説】
外部10進項目の内部表現は、(そのまま文字として表示可能なように)システムの文字コード系に依存して決められている。
<table><tr><td></td><td colspan="2">SIGN句の指定</td></tr><tr><td></td><td>なし or SEPARATEなし</td><td>あり and SEPARATEあり</td></tr><tr><td>MSP
XSP</td><td>F：ゾーンビット
C：正符合
D：負符合</td><td>4E：正符合
60：負符合</td></tr><tr><td>Windows
Solaris</td><td>3：ゾーンビット
4：正符合
5：負符合</td><td>2B：正符合
2D：負符合</td></tr></table>
EBCDICでX”4E”は”+”、X”60”は”-”、
ASCIIでX”2B”は”+”、X”2D”は”-”を表現する。</td></tr>
<tr><td>5</td><td>浮動小数点項目
浮動小数点数の操作</td><td>演算結果が異なる場合がある。
また、浮動小数点項目を再定義している場合、再定義項目を使用しての操作結果が異なる。</td><td>【対処方法】
なし。
【解説】
浮動小数点数の内部表現形式がMSP/XSP(Fujitsu固有)とWindows/Solaris(IEEE)で異なる。このため、符合部の位置や仮数部の演算精度が異なってしまう。</td></tr>
<tr><td>6</td><td>日本語を含む集団項目の比較</td><td>比較結果がMSP/XSPと異なる場合がある。
例： 以下の例でMSP/XSPなら”OK”、Windows/Solarisなら”NG”

…
01 GRP.</td><td>【対処方法】
集団項目が2進項目を含まない場合なら、翻訳オプションNSPCOMP(ASP)を指定することで回避できる。
ただし、実行時コード系としてUnicodeを採用する場合、NSPCOMP(ASP)は指定しても有効にならない。
【解説】
MSP/XSPで使用する文字コード系EBCDIC/JEFの環境では英数字空白(X”40”)2文字を日本語空白(X”4040”) 1文字と等価に扱うことが可能である。
Windows、Solarisで使用される文字コード系では英数字空白と日本語空白は全く関連のない値が割り当てられているため、単純な比較では一致しない。
<table><tr><td>英数字空白</td><td colspan="2">日本語空白</td></tr><tr><td rowspan="2">X”20”</td><td>SJIS</td><td>X”8140”</td></tr><tr><td>EUC</td><td>X”A1A1”</td></tr></table></td></tr>
</table>

		END-IF …	| | EUC | X”A1A1” | 左の例の場合、表意定数SPACEは、集団項目GRPへの転記時は英数字空白として評価されるが、NCH1との比較時は日本語空白と評価される。 上記の理由で、英数字空白２文字を日本語空白１文字と等価と扱うことができないなら、比較結果は偽となる。
7	表示ファイル(宛先DSP、PRT)	実行時エラーとなるか、実行結果が異なる場合がある。	【対処方法】 なし。 プログラムおよび定義体の修正が必要になる。 【解説】 宛先をDSP、PRTとする表示ファイルは、画面表示・印刷用のサブシステムを必要とするが、それがMSP/XSPとWindows/Solarisで異なる。 - MSP/XSP：PSAM - Windows/Solaris：MeFt このPSAMとMeFtのサポートする機能範囲が異なるため、非互換が発生する。
8	FORMAT句付き印刷ファイル	図表作成機能が使用できない。	【対処方法】 なし。 【解説】 図表作成機能はPSAMとその定義体が提供する機能であるため、MeFtを使用するNetCOBOLでは使用できない。
9	特殊レジスタ SHIFT-IN SHIFT-OUT	動作が異なる。	【対処方法】 使用する必要がない。 【解説】 特殊レジスタSHIFT-IN、SHIFT-OUTはEBCDICコードとJEFコードが混在する場合にその切り替え位置を示すA/N制御コードを意味する。Windows/Solarisで使用されるSJISまたはEUCは、はじめから１バイト文字と２バイト文字の混合を意識して作られたコード系であるため、このような制御コードは必要ない。 Windows/Solarisでは特殊レジスタSHIFT-IN、SHIFT-OUTは１文字の半角空白として扱われる。
10	FUNCTION LENG	SET文の送り出し側に指定できない。	【対処方法】 なし。 【解説】 翻訳エラーになる。
11	報告書作成機能	日本語項目を含む報告書は正しく動作しない。	【対処方法】 なし。

A.2.4 オープン系のCOBOLでは意味を持たない機能

以下に示す機能は、ホストのCOBOL85のみ有効であって、COBOL97/NetCOBOLでは記述しても意味を持ちません。ただし、NetCOBOLでは特に意味を持たずともホストのCOBOL85と同様に動作可能なもの(例えばBLOCK CONTAINS句)については特に述べません。
句あるいは指定レベルの記述であるため、どのような場所で使用されるかも含めて示します。

表A-3 オープン系のCOBOLでは意味を持たない機能

No	NetCOBOLでは意味を持たない記述			MSP/XSPのCOBOL85での機能
	機能	記述場所	句・指定	
1	特殊レジスタ	手続き部	SHIFT-IN/SHIFT-OUT	A/N制御コードを表す。SHIFT-INはX”28”。 SHIFT-OUTはX”29”。
2			SORT-CORE-SIZE	ソート・マージプログラムの使用する記憶域の大きさを指定する。
3			SORT-FILE-SIZE	ソート・マージの対象となるファイルのレコード数の推定値を指定する。
4			SORT-MESSAGE	ソート・マージプログラムのメッセージの出力先を変更したい場合に指定する。
5			SORT-MODE-SIZE	ソート・マージの対象となるファイルのレコードが可変長の場合、もっとも頻度の高いレコード長を指定する。
6	機能名	SPECIAL-NAMES段落	SYSPUNCH, SYSPCH	カード読み取り装置
7			CSP	機能名SLCと同義
8			S01, S02	それぞれSTACKER-01、STACKER-02と同義
9			BUSHU、SOKAKU、ON-YOMI、KUN-YOMI	日本語データで整列合を行う際の比較方法を指定する。 (対応づけた呼び名をSORT/MERGE文に指定)
10	順ファイル	FILE-CONTROL段落	PASSWORD句	ファイルに対するパスワードを指定する。
11		I-O-CONTROL段落	APPLY WRITE-ONLY句	媒体上のスペース効率を向上させる。
12			MULTIPLE FILE TAPE 句	複数ファイルリール上の位置を指定する(ANSI ‘85廃要素)。
13			RERUN句	再開を行う時点と媒体を記録する(ANSI ‘85廃要素)。
14			RESERVE AREA句	コンパイラの割り当てる入出力領域の個数を指定する
15		ファイル(FD)記述項	CODE-SET句	外部記憶媒体上のデータ表現に使用する符合系(文字コード)を指定する。
16	相対ファイル	FILE-CONTROL段落	PASSWORD句	ファイルに対するパスワードを指定する
17		I-O-CONTROL段落	RERUN句	再開を行う時点と媒体を記録する(ANSI ‘85廃要素)。
18	相対ファイル	FILE-CONTROL段落	PASSWORD句	ファイルに対するパスワードを指定する
19		I-O-CONTROL 段落	RERUN句	再開を行う時点と媒体を記録する(ANSI ‘85廃要素)。
20	表示ファイル	FILE-CONTROL段落	PROCESSING TIME句	非同期メッセージ通信、プログラム間通信時の待ち時間を指定する。

A.3 翻訳オプション

この項では、OSⅣ系のCOBOL85とCOBOL97/NetCOBOLの翻訳オプションの違いについて説明します。
なお、翻訳オプションはその機能からいくつかのカテゴリに分類できるため、ここではそのカテゴリを以下のように分け、また、どのカテゴリに含まれるかをアルファベット１文字の記号で示します。

- L：翻訳時のリスト出力に関するオプション。
- M：ソースプログラムの解釈に関するオプション。
- P：ソースプログラムの扱いに関するオプション。
- O：生成するオブジェクトに関するオプション。
- C：翻訳の手続きに関するオプション。
- X：実行時の扱いに関するオプション。
- D：実行時のデバッグ機能に関するオプション。
- S：翻訳用の資源の指定に関するオプション。
- T：その他のオプション。

A.3.1 COBOL97/NetCOBOLでは使用できない翻訳オプション

以下の一覧で示す翻訳オプションは、その機能がNetCOBOLには存在しないか、意味を持たないため使用できません。
なお、表中の指定結果は指定して翻訳した結果出力される診断メッセージのエラーレベルです。

表A-4 COBOL97/NetCOBOLでは使用できない翻訳オプション

No	オプション名	機能概要		指定結果	解説
		カテゴリ	説明		
1	ACS	O	ACS配下のファイルの処理を行うかどうかを指定する。	W	COBOL97/NetCOBOLでは意味を持たない。
2	AIMLIBDD	S	AIMディレクトリのデータセットをDD名(MSP)/アクセス名(XSP)で指定する。	E	本来的な意味では使用できない。 分散開発のため、AIMディレクトリから生成したサブスキーマ定義体の格納パスを指定するためのオプションとして、AIMLIBがある。
3	CMODE1/CMODE2	L	翻訳リスト等での日本語文字の表示サイズを指定する。	W	COBOL97/NetCOBOLでは意味を持たない。
4	CTLCHR	M	ADVANCING付きのWRITE文で使用するレコードに制御文字の領域が用意されているかどうか指定する。	W	ホストのCOBOL85と同等の処理が行われるが、制御文字はCOBOL97/NetCOBOLの動作環境では意味を持たないため指定してはならない。
5	DCT	C	ADAMSディクショナリ作成用の情報を出力するか同化を指定する。	W	COBOL97/NetCOBOLでは意味を持たない。
6	ELM	C	区分データセットまたはGEMライブラリ中から翻訳するメンバ	W	COBOL97/NetCOBOLでは意味を持たない。 機能的には開発環境が代行する。

			を指定する。		
7	FLOW	D	プログラムの異常終了時に逆トレース情報を表示するかどうかを指定する。	W	COBOL97/NetCOBOLではこの機能は存在しない。 機能的にはTRACEで代用可能である。
8	FSORT	O	ソート・マージプログラムにシステムのものを使用するか外部のプログラムを使用するかを指定する。	W	COBOL97/NetCOBOLでは意味を持たない。 PowerSORTがインストール済なら、COBOL97/NetCOBOLはソート・マージプログラムとして、無条件にPowerSORTを使用する。
9	JIMLIB	S	日本語拡張辞書のデータセットを指定する。	S	COBOL97/NetCOBOLではこの機能は存在しない。
10	JIMLIBDD	S	日本語拡張辞書のデータセットをDD名(MSP)/アクセス名(XSP)で指定する。	S	COBOL97/NetCOBOLではこの機能は存在しない。
11	LEAVE	O	実行単位の終了時にロードされたモジュールを削除するか、しないかを指定する。	W	COBOL97/NetCOBOLでは意味を持たない。
12	LIBDD	S	登録集のデータセットをDD名(MSP)/アクセス名(XSP)で指定する。	E	COBOL97/NetCOBOLではこの機能は存在しない。OF/IN指定を持つCOPY文で参照する登録集名とフォルダを対応づける場合は環境変数で指定する。
12	LIL	M	FORTRANプログラムとの言語間結合時のプログラム環境の切り換えを行うかどうかを指定する。	E	COBOL97/NetCOBOLではこの機能は存在しない。
13	PINT	O	実行時にプログラム割り込みが発生した場合の動作を指定する。	W	COBOL97/NetCOBOLではこの機能は存在しない。
14	PSF	O	表示サービス機能(Presentation Service Function)を利用して,AIM配下またはEXCEFW配下でプログラムが同環境下のVSAMファイル等にアクセスするかどうかを指定する。	W	COBOL97/NetCOBOLでは意味を持たない。
15	RENT	O	リエントラント(再入可能)属性を持つプログラムを作成するかどうか指定する。	W	COBOL97/NetCOBOLではこの機能は存在しない。
16	SIZE	C	コンパイラに与える仮想記憶領域の大きさを指定する。	I	COBOL97/NetCOBOLでは意味を持たない。
17	SMOUT	X	ソート・マージプログ	W	COBOL97/NetCOBOLではこの機能は存在し

			ラムの出力するメッセージの出力先を指定する。		ない。 ソート・マージプログラムの出力するメッセージはCOBOLの実行時メッセージに埋め込まれて出力される。
18	SQLLVL	T	埋め込みSQL文で使用可能とする機能を指定する。	W	Solaris版はCOBOL97/NetCOBOLが埋め込みSQL文を直接処理できないため、意味を持たない。 Windows版では、埋め込みSQL文を直接処理する機能を持つが、ホストのCOBOL85と機能範囲が一致しないため、翻訳オプションも異なる。
19	STATEMENT	D	実行時に異常終了した場合、異常終了の原因となった文に関する情報を表示するかどうか指定する。	W	COBOL97/NetCOBOLではこの機能は存在しない。 機能的にはTRACEまたはTESTで代用可能である。
20	TRAP	X	実行時メッセージの出力先を指定する。	W	COBOL97/NetCOBOLではこの機能は翻訳オプションではなく、実行時の環境変数として指定する。 - Solaris版： CBL_MESSAGEFILE - Windows版： @CBL_MESSAGEまたは @MessOutFile
21	UWA	M	AIMデータベース機能を使用するとき,英数字のUWAを使用するか,日本語のUWAを使用するかを指定する。	W	COBOL97/NetCOBOLでは意味を持たない。
22	VRFILE	X	VSAM相対ファイルの実現方法を指定する。	W	COBOL97/NetCOBOLでは意味を持たない。

A.3.2 COBOL97/NetCOBOLでは未サポートの翻訳オプション

以下の一覧で示す翻訳オプションは指定しても翻訳エラーになりませんが、COBOL97/NetCOBOLでは動作が保証されていません。このため、COBOL97/NetCOBOLの使用手引書のオプションの説明に記載されていません。MAPを除き基本的にここで示すオプションは指定しないでください。

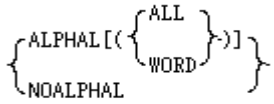
表A-5 COBOL97/NetCOBOLでは未サポートの翻訳オプション

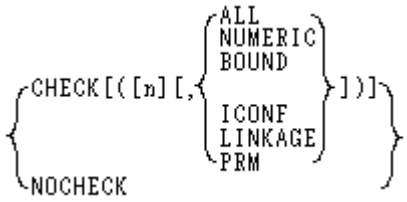
No	オプション名	機能概要	
		カテゴリ	説明
1	ATTRIBUTE	L	ソース中に含まれる利用者語の属性情報を表示するかどうかを指定する。
2	JCONST	L	拡張リスト中で日本語定数を16進表示するかどうかを指定する。
3	MAP	L	データマップリスト等を表示するかどうかを指定する。
4	PC01	O	ファイルのページ換えにチャネル１を使用するかどうかを指定する。
5	SPACE	L	翻訳リスト中の行間隔を指定する。

A.3.3 COBOL97/NetCOBOLと機能差のあるオプション

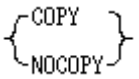
以下の一覧で示す翻訳オプションは、OSVI系のCOBOL85とCOBOL97/NetCOBOLの間で指定の方法が異なったり、機能差があったりする。なお、指定方法の違いや機能差はあってもホストのCOBOL85における指定がそのまま受け入れられる場合は、互換性の欄に○をつけて、それを示します。

表A-6 COBOL97/NetCOBOLと機能差のあるオプション

No	オプション名	機能概要(ホストのCOBOL85)		互換性	機能差の説明
		カテゴリ	説明		
1	AIMLIB	S	AIMディレクトリのデータセットを指定する。	×	COBOL97/NetCOBOLでは、サブスキーマ取り出しツール(GETSSCH)によってAIMディレクトリから取り出したサブスキーマ登録集を格納したフォルダの指定に用いる。
2	ALPHAL	M	ソース中の英小文字を英大文字と等価に扱うことを指定する。 ただし、定数中の英小文字は以下の場所に指定されたもの以外は、英小文字と英大文字が区別される。 - CALL文の直後 - CANCEL文の直後 - ENTRY文の直後	○	定数中の英小文字の扱いに関してのサブオペランドが追加された。形式は以下のとおり。  ALPHAL(ALL)：ホストのCOBOL85のALPHAL相当であり、定数中の英小文字は以下の場所に指定されたもの以外は、英小文字と英大文字を区別する。 - CALL文の直後 - CANCEL文の直後 - ENTRY文の直後 - その他、プログラム名、メソッドの外部名を指定する場所 ALPHAL(WORD)：定数中の英小文字／英大文字は区別する。

<table>
<tr><td>3</td><td>CHECK</td><td>D</td><td>CHECK機能を使用するかどうか指定する。</td><td>×</td><td>サブオペランドの指定方法が異なる。新しい指定形式は次の通り。
 

CHECK(ALL) : NUMERIC, BOUND, ICONF LINKAGE,PRMのチェックをする。
CHECK(NUMERIC)：数字項目のデータ例外と0による除算のチェックをする。
CHECK(BOUND)：添字、指標および部分参照の範囲が適切かのチェックをする。
CHECK(ICONF)：INVOKE文のパラメタの適合をチェックする。
CHECK(LINKAGE)：CALL文のLINKAGE規約が不一致でないかのチェックをする。
CHECK(PRM)：プログラムを呼び出すCALL文のパラメタのチェックをする。

完全に機能が一致するわけではないが、次のように見なすことができる。
<table><tr><td>COBOL85 の指定</td><td>COBOL97/ NetCOBOLの指定</td></tr><tr><td>CHECK</td><td>CHECK(BOUND)</td></tr><tr><td>CHECK(EXTEND)</td><td>CHECK(ALL)</td></tr></table></td></tr>
<tr><td>4</td><td>CONF</td><td>M</td><td>COBOLの旧規格との非互換を指摘する。</td><td>○</td><td>JIS COBOLとの比較を行う機能はサポートされていない。以下の指摘のみ行う。
CONF(68)： ANS’68 COBOLとANS‘85 COBOLの非互換を指摘する。
CONF(74))：ANS‘74 COBOLとANS‘85 COBOLの非互換を指摘する。
CONF(OBS)：COBOL85の廃要素を指摘する。</td></tr>
<tr><td>5</td><td>COPY</td><td>L</td><td>ソースプログラムリストにCOPY文で組み込まれる登録集原文を表示するかどうか指定する。</td><td>○</td><td>COBOL97/NetCOBOLでは次の指定形式を持つ。
 

サブオペランドは存在せず、COPYの指定はホストのCOBOL85におけるCOPY(FULL)の指定に相当する動作をする。</td></tr>
<tr><td>6</td><td>DLOAD</td><td>X</td><td>動的プログラム構造を使用するかどうか指定する。</td><td>×</td><td>COBOL97/NetCOBOLでは次の指定形式を持つ。
 

サブオペランドは存在せず、DLOADの指定はホストのCOBOL85におけるDLOAD(SUB)の指定に相当する動作をする。</td></tr>
</table>

7	LIST	L	オブジェクトリストを表示するかどうか指定する。	×	COBOL97/NetCOBOLでは次の指定形式を持つ。 {LIST / NOLIST} サブオペランドは指定できない。
8	QUOTE	M	ソースプログラム中で使用する引用符文字および表意定数QUOTE/QUOTESの値を指定する。	○	ホストのCOBOL85では、このオプションで引用符文字にクォーテーションマーク(“)を使用するか、アポストロフィー(‘)を使用するか指定する必要があったが、NetCOBOLではソースプログラム中でどちらの引用符文字も使用可能である。 このオプションは表意定数QUOTE/QUOTESの値に対してのみ有効となる。
9	SOURCE	L	ソースプログラムリストを表示するかどうかを指定する。	×	COBOL97/NetCOBOLでは次の指定形式を持つ。 {SOURCE / NOSOURCE} サブオペランドは指定すれば受け入れるが、SOURCE(2)は正しく動作しない。
10	TEST	D	対話型デバッガを使用するかどうか指定する。	○	COBOL97/NetCOBOLでは次の指定形式を持つ。 {TEST / NOTEST} サブオペランドは指定すれば受け入れるが、意味を持たない。

A.4 予約語

オープン系のCOBOL製品はバージョンアップを重ねるにつれて、より多くの機能が追加されてきましたが、同時に多くの語を予約語として追加する必要が生じました。このため、オープン系のCOBOL製品の最新版では、OSIV系のCOBOL85より多くの語を予約語と見なします。

予約語と重なる語は、COBOLソースおよび登録集原文(COPY句)中で利用者語（データ名、ファイル名、ラベル、呼び名等）として使用することができない。このため、新たな予約語の追加により、これまで利用者語として使用可能であった語があるバージョンから使えなくなるという現象が発生します。

COBOLはプログラムの互換性を重視するプログラム言語であるため、最新の予約語セットのサブセットにあたる過去のバージョンで使用していた予約語セットを翻訳オプションRSVによって選択する機能を提供しています。

しかし、追加された予約語は新しい機能をサポートするために追加されたものですから、過去のバージョンの予約語セットを選択した場合、NetCOBOLの一部の機能が使用できなくなります。そこで予約語セットとそれによってサポートされる機能の関係を知ることが必要になります。

ここではホストのCOBOL85を使用していた方の観点にたって、ホストのCOBOL85の予約語セットを基準に、それにどんな語が追加され、それによってどんな機能がサポートされたかを示します。

表A-7 オープン系のCOBOL製品で追加された予約語

予約語セット名	対応製品およびバージョン	予約語セット内容	関連する追加機能
V122	COBOL85 V12L20	V122(ホストのCOBOL85の予約語セット)	

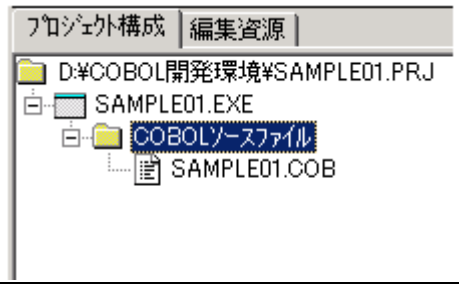
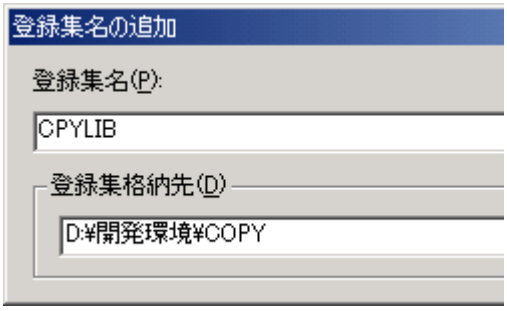
V125	COBOL85 V12L30～ V12L50	V122の予約語セット + MANUAL, AUTOMATIC, COMP-5, COMPUTATIONAL-5 AUTO, BACKGROUND-COLOR, BELL, BLINK, CRT, CURSOR, END-ACCEPT, END-DISPLAY, EOL, EOS, FOREGROUND-COLOR, FULL, HIGHLIGHT, LOWLIGHT, REQUIRED, REVERSE-VIDEO, SCREEN, SECURE, UNDERLINE,	XPG仕様サポート (システム２進、 ファイル排他、 スクリーン機能)
V30	COBOL85 V20L10～ V30L10	V125の予約語セット + COMP-X, COMPUTATIONAL-X, CRT-UNDER, GRID, JAPANESE, LEFT-JUSTIFY, LEFTLINE, OVERLINE, PREVIOUS, PROMPT, RIGHT-JUSTIFY, SPACE-FILL, TRAILING-SIGN, ZERO-FILL,	MicroFocus 社COBOL/2 互換機能のサポート (主にスクリーン機能)
V40	COBOL97 V40L10～ V60L10	V30の予約語セット + CLASS-ID, END-INVOKE, EXCEPTION-OBJECT, FACTORY, INHERITS, INVARIANT, INVOKE, METHOD, METHOD-ID, OBJECT, OVERRIDE, PROPERTY, PROTOTYPE, RAISING, RAISE, REPOSITORY, RETURNING, SELF, SUPER, SYSTEM-OBJECT, UNIVERSAL,	オブジェクト指向機能サポート (2002年規格の先取り)
V61	COBOL97 V61L10	V40の予約語セット + TYPEDEF	型(ユーザ定義型)の定義／参照機能のサポート
V70	NetCOBOL V70L10	V61の予約語セット + BINARY-CHAR, BINARY-DOUBLE, BINARY-LONG, BINARY-SHORT, CUSTOM-ATTRIBUTE, DELEGATE-ID, DELEGATE, ENUM-ID, ENUM, FLOAT-EXTENDED, FLOAT-LONG, FLOAT-SHORT, INTERFACE-ID, INTERFACE, INTERNAL, PRIVATE, PUBLIC, STATIC,	Microsoft .NET 連携機能サポート
ALL	NetCOBOL V71L10	V70の予約語セット + EDIT-OPTION2, EDIT-OPTION3	PMD12版対応

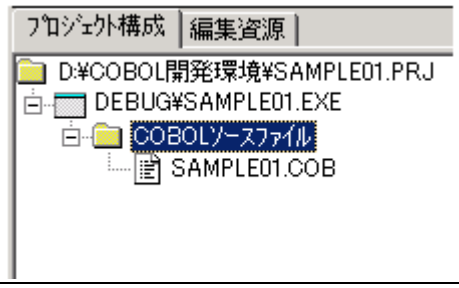
A.5 特定のDD名／アクセス名に相当するファイルの指定

ホストのCOBOL85を使用する場合、特定の資源をDD名(MSP)あるいはアクセス名(XSP)で指定する場合があります。この項では、NetCOBOLで同等のことを行うためにどのように指定する必要があるか、それを示します。

表A-8 特定のDD名／アクセス名に相当するファイルの指定

NO	DD名 (MSP)	アクセス名(XSP)	使用目的	NetCOBOLでの指定
翻訳時に使用されるDD名/アクセス名				
1	SYSIN	SLIB	ソースプログラムを指定する。	プロジェクトマネージャの<<プロジェクト構成>>タブでCOBOLソースファイルフォルダを作成し、これに追加する。

				
2	SYSLIB	MLIB	登録集のデータセットを指定する。	翻訳オプションLIBで格納フォルダ名を指定する。
3	任意(登録集名)		OF/IN指定で登録集名を指定したCOPY文を使用している場合、登録集のデータセットを指定する。	翻訳オプションダイアログの<登録集名(P)>ボタンで押すと登録集の設定ダイアログが開くので、ここで登録集名とフォルダ名の対応づけを追加する。 例： “COPY XXXX OF CPYLIB”に対する設定 
4	SYSUT1	U01	COBOLコンパイラの作業用データセットを指定する。	設定の必要はない。
5	SYSPRINT	LIST	翻訳リストの出力先データセットを指定する。	翻訳オプションPRINTの設定で、<<翻訳リスト出力先(D)>>テキストボックスに出力先フォルダを指定する。ファイル名はソースファイルの拡張子を”.lst”に置き換えたものとなる。
6	SYSLIN	RLIB	オブジェクトファイルの出力先を指定する。	翻訳オプションOBJECTの設定で、〔目的プログラム出力先〕に出力先フォルダを指定する。ファイル名はソースファイルの拡張を”.o”(Solaris版)、または”.obj”(Windows版)に置き換えたものとなる。
7	SYSDCT		ADAMSディクショナリ作成用の情報の出力先を指定する。翻訳オプションDCT指定時に必要となる。	NetCOBOLでは翻訳オプションDCTは意味を持たないので指定する必要はない。
8	AIMLIB		AIMディレクトリのデータセットを指定する。	翻訳オプションAIMLIBで格納フォルダ名を指定する。

9	JIMLIB JIMLIB1 JIMLIB2 JIMLIB4		日本語項目変換辞書のデータセットを指定する。次のような拡張日本語機能を使用する場合に必要となる。 - 日本語項目定数 （例： NN”ｶﾝｼﾞ”） - 日本語英数字定数 （例: NA”ABCD”） - 日本語連想定数 （例： NK”ﾔﾏｶﾜ”） - 日本語ひらがな定数 （例: NH”ﾋﾗｶﾞﾅ”）	NetCOBOLは左の形式の日本語定数をサポートしていないため、この設定を必要とすることはない。
10	STEPLIB	PRGLIB	翻訳処理でCOBOL以外のプログラムが必要な場合、これにそのデータセットを追加する。	環境変数PATHに必要なプログラムのパスを追加する。
11	STEPCAT		日本語メッセージのユーザカタログを指定する。	COBOLの開発環境の環境設定で行う。 - Windows版 ： インストール時に自動設定 - Solaris版 ： インストールの環境設定で
リンク時に使用されるDD名/アクセス名				
12	SYSLIN	RLIB	オブジェクトの格納データセットを指定する。	NetCOBOLでは特に指定しない。
13	SYSPRINT	LIST	リンク時のメッセージ等の出力先を指定する。	NetCOBOLではデフォルトではビルダウィンドウに表示される。
14	SYSLMOD	ELIB	ロードモジュール(実行可能ファイル)の出力先を指定する。	プロジェクトマネージャの<<プロジェクト構成>>タブで多ターゲットファイルのパスを変更する。 
15	SYSUT1	U01	リンク時の作業用のデータセットを指定する。	NetCOBOLでは特に指定する必要はない。
プログラムの実行時に使用するDD名/アクセス名				
18	任意(ファイルの宛先)		プログラム中でファイルを使用する場合、そのファイル識別名に対してデータセットを割り当てる。	環境変数で指定する。環境変数名はDD名またはアクセス名と同じ名前とする。 例： ファイル識別名がSYS001の場合 - Windows版 ： SET SYS001=.¥MASTER.DAT - Solaris版 ： setenv SYS001 ./MASTER.DAT
19	SYSIN	UIN	機能名SYSINに対応づけた呼び名を宛先とするACCEPT文(FROM指定を省略時も)の入力元を割り当てる。	通常は必要ない。 翻訳オプションSSINで宛先を変更している場合のみ、環境変数で入力元ファイルと対応づける必要がある。 例： SSIN(A)と指定して翻訳 - Windows版 ： SET A=.¥INPUT.DAT - Solaris版 ： setenv A ./INPUT.DAT

20	SYSOUT	LIST	機能名SYSOUTに対応づけた呼び名を宛先とするDISPLAY文(UPON指定を省略時も)の出力先を割り当てる。	通常は必要ない。 翻訳オプションSSOUTで宛先を変更している場合のみ、環境変数で入力元ファイルと対応づける必要がある。 例： SSOUT(B)と指定して翻訳 - Windows版 ： SET B=.¥INPUT.DAT - Solaris版 ： setenv B ./INPUT.DAT
21	SYSDBPRT		TESTCOBを使用する（翻訳オプションTESTを指定した）場合に使用する。	NetCOBOLの対話型デバッガでは必要ない。
22	SYSDBIN			
23	SYSDBOUT		翻訳オプションCOUNT、FLOW、STATEMENT等の指定時、実行時デバッグ情報の出力先データセットを指定する。	NetCOBOLはFLOW、STATEMENT機能をサポートしていない。 また、COUNT機能ではSYSDBOUTを使用しない。
24	SYSCOUNT		COUNT機能を使用する際、その実行時文統計情報の出力先データセットを指定する。	環境変数SYSCOUNTに出力先のファイル名を指定する。
25	STEPLIB	PRGLIB	COBOLライブラリや私的ライブラリ(副プログラム等)のデータセットを指定する。	環境変数で指定する。COBOLライブラリについては、次のタイミングで設定さる。 - Windows版 ： インストール時に自動設定 - Solaris版 ： インストールの環境設定時 その他(ダイナミックリンクライブラリ/共用ライブラリ)のためには、それぞれ次の環境変数にそのパスを追加する。 - Windows版 ： 環境変数PATHに追加 - Solaris版 ： 環境変数LD_LIBRARY_PATHに追加
26	STEPCAT or JOBCAT		使用するファイルがVSAMファイルでカタログされている場合、そのカタログを指定する。	NetCOBOLでは、VSAM相当の機能をライブラリで実現しているため、必要ない。
27	SORTWRK01、SORTWRK02 …		整列併合機能で使用する作業域のデータセットを指定する。	NetCOBOLでは特に指定する必要はない。

付録B 定義体移行時の留意点

COBOLソース、COBOLソース以外のOSIV系システムでの以下のプログラム資産をオープン系システムに移行する際の留意点について説明します。

- メッセージ定義体
- フォームオーバレイパターン

B.1 フォーマット定義体移行時の留意事項

OSIV系のPSAMで使用するフォーマット定義体とオープン系のMeFtで使用される画面帳票定義体には様々な機能差があります。
このため、フォーマット定義ソースからフォーマット定義体移出機能(ADDFORM)によって、画面帳票定義体への変換を行う際に、次の留意点があります。

なお、表中の記号は次の意味があります。

ホストでの変換の可否

○：変換可能
△：変換可能（留意事項あり）
－：意味がありません。
×：変換不可（画面帳票定義体に変換できない）

警告の有無

M ：フォーマット定義ソースを画面帳票定義体に変換する際に検出され、変換可否が“－”または“△”の場合は警告メッセージが出力され、変換可否が“×”の場合は処理中断メッセージが出力されます。
－：検出されません（該当機能は無視される）

表B-1 フォーマット定義体ソース変換時の留意事項１

名称	命令	オペランド	変換可否	警告の有無	留意事項
文字列	EQU	マクロ命令	○		
		オペランド	○		
		キーワード	○		
		ブオペランド値	○		
		サブオペランド	○		
フォーマットID(続く)	FID（続く）	TYPE=OUT	○		
		DSP	○		
		CMD	×	M	該当ハードウェアなし
		PGM=STD	○		
		AIM	－	M	STDとして処理する
		ACS	－	M	STDとして処理する
		BATCH	－	M	STDとして処理する
		CMN	－	M	STDとして処理する
		LANG=COBOL	○		
		PL1	－	M	COBOLとして処理する
		FORTRAN	－	M	COBOLとして処理する
		ANY	－	M	COBOLとして処理する
		COPYLIB=FID	○		
		YES	－	－	FIDとして処理する
		NO	－	－	FIDとして処理する

名称	命令	オペランド	変換可否	警告の有無	留意事項
フォーマットID(続き)	FID (続き)	FALT=X'nnnn'	－	－	OSIV系固有オプション
		PARM=ALL	－	－	
		TRANS	－	－	
		HRER=S	－	－	
		F	－	－	
		HREFCONV=YES	－	－	
		NO	－	－	
		SV2000=S	－	－	
		F	－	－	
		CNVSOC=MED	－	－	
		MFG	－	－	
		CRIGHT=nnnn	－	－	
		GID=帳票ID	－	－	VIR情報
		DECIMAL=COMMA	○		
		FERIOD	○		
レコードID	RECORD	TYPE=INOUT	○		
		OUT	○		出力レコードまたは入出力レコードに変換します。
		IN	△	M	入出力レコードに変換します。
		IMG	×	M	イメージ機能はありません。
		NAME=レコード名	○		プログラミング定義体名となります。省略時、入出力レコード文の名札がプログラミング定義体名となります。
		DUSAGE=DATA	－	－	項目制御部付きとなります。
		CTRLn	－	－	項目制御部は5バイト固定です。
変数名(続く)	DATA (画面) (続く)	NAME=データ名	○		プログラミング項目名となります。省略時、変数名がプログラミング項目名となります。
		USAGE=DATA	－	－	項目制御部付きとなります。
		CTRLn	－	－	項目制御部は5バイト固定です。
		GRP	○		集団項目に変換します。
		CUR	－	M	機能なし(空項目にならず)
		MSGn	－	M	機能なし(空項目にならず)
		DCAn	－	M	機能なし(空項目にならず)
		MODE	－	M	機能なし(空項目にならず)
		IMGID	△	－	項目長は8となります。
		AID	－	M	機能なし(空項目にならず)
		AIDP	－	M	機能なし(空項目にならず)
		FNAME	－	M	機能なし(空項目にならず)
		IDCR	－	M	機能なし(空項目にならず)
		FIGID	△	－	項目長は8となります。
		FIFIDM	△	M	FIGIDと見なします。
		VSCROLL	×	M	論理画面はありません。

名称	命令	オペランド	変換可否	警告の有無	留意事項
変数名(続く)	DATA (画面) (続き)	HSCROLL	×	M	論理画面はありません。
		FIX	×	M	論理画面はありません。
		DRCTFIX	×	M	論理画面はありません。
		PHRASE	×	M	論理画面はありません。
		VER=NONB	−	−	機能がありません。
		(RANGE, 1, h)	−	−	機能がありません。
		(LIST, a, b…)	−	−	機能がありません。
		VALID	−	−	機能がありません。
		RC1	○		
		RC2	○		
		VEMSG=(変数名, メッセージID)	−	−	メッセージ機能がありません。
		INIT=’初期値’	△	−	振り分けキー項目だけ有効。
		PIC=(X, w)	○		
		(N, w)	○		
		(M, w)	○		
		(9, n, d)	○		
		(S, n, d)	○		
		(C, n, d)	△	M	外部10進数に変換します。
		(D, n, d)	△	M	外部10進数に変換します。
		(B, n)	△	M	外部10進数に変換します。
		(R, 1)	−	M	実数がありません。
		(E, 編集文字列)	−	M	
		EDIT=’外部精度’	△	M	パターンNoに変換します。
		Pnn	○		
		OCCURS=n	○		
		(n, m)	○		
		RNAME=データ項目名	○		ボディ件数は、OCCURSオペランドの値。当オペランドの存在時のみ伝票形式となります。データ項目名は無視します。
		PADDING=BLANK	−	−	埋め込み機能はありません。
		NULL	−	−	埋め込み機能はありません。
		X’16進数’	−	−	埋め込み機能はありません。
	SDATA (画面) (続く)	NAME=データ名	○		プログラミング項目名となります。省略時、変数名がプログラミング項目名となります。
		USAGE=DATA	−	−	項目制御部付きとなります。
		CTRLn	−	−	項目制御部は5バイト固定です。
		VER=NONB	−	−	機能がありません。
		(RANGE, 1, h)	−	−	機能がありません。
		(LIST, a, b…)	−	−	機能がありません。
		VALID	−	−	機能がありません。
		RC1	○		
		RC2	○		
		VEMSG=(変数名, メッセージID)	−	−	メッセージ機能がありません。

名称	命令	オペランド	変換可否	警告の有無	留意事項
変数名(続き)	SDATA (画面) (続き)	PIC=(X, w)	○		
		(N, w)	○		
		(M, w)	○		
		(9, n, d)	○		
		(S, n, d)	○		
		(C, n, d)	△	M	外部10進数に変換します。
		(D, n, d)	△	M	外部10進数に変換します。
		(B, n)	△	M	外部10進数に変換します。
		(R, 1)	－	M	実数がありません。
		(E, 編集文字列)	－	M	
		EDIT=’外部精度’	△	M	パターンNoに変換します。
		Pnn	○		
		INIT=’初期値’	△	－	振り分けキー項目だけ有効。
		PADDING=BLANK	－	－	埋め込み機能はありません。
		NULL	－	－	埋め込み機能はありません。
		X’16進数’	－	－	埋め込み機能はありません。
		OCCURS=m	○		
項目群名	GROUP (画面)	DATA=(可変データ, …)	○		
変数名	DIST (画面)	DATA=(文字列, …, ELSE)	○		
		PARM=(処理, 処理, …)	○		
		KEY=(変数名,位置,長さ)	－	M	
		CASE=(文字列, …, ELSE)	－	M	
		CALL=(‘処理’,…)	－	M	
	KEYS (画面)	DATA=(文字列, …, ELSE)	－	－	記号変数インタフェースはありません。
		PARM=(処理, 処理, …)			
項目群名	GROUP (画面)	DATA=(可変データ, …)	○		
変数名	DIST (画面)	DATA=(文字列, …, ELSE)	○		
		PARM=(処理, 処理, …)	○		
		KEY=(変数名,位置,長さ)	－	M	
		CASE=(文字列, …, ELSE)	－	M	
		CALL=(‘処理’,…)	－	M	
	KEYS (画面)	DATA=(文字列, …, ELSE)	－	－	記号変数インタフェースはありません。
		PARM=(処理, 処理, …)			
	TRANS (画面)	DATA=(文字列, …, ELSE)	－	－	記号変数インタフェースはありません。
		PARM=(処理, 処理, …)			
メッセージID	MSG (画面)	VALUE=固定データ	－	M	メッセージ機能はありません。
変数名 (続く)	DATA (帳票) (続く)	NAME=データ名	○		プログラミング項目名となります。省略時、変数名がプログラミング項目名となります。
		USAGE=DATA	－	－	項目制御部付きとなります。
		CTRLn	－	－	項目制御部は5バイト固定です。
		GRP	○		集団項目に変換します。
		DCAn	－	M	機能なし(空項目にならず)
		IMGID	△	－	項目長は8となります。

名称	命令	オペランド	変換可否	警告の有無	留意事項
変数名 (続き)	DATA (帳票) (続き)	FIGID	△	－	項目長は8となります。
		FIFIDM	△	M	FIGIDと見なします。
		PIC=(X, w)	○		
		(N, w)	○		
		(M, w)	○		
		(9, n, d)	○		
		(S, n, d)	○		
		(C, n, d)	△	M	外部10進数に変換します。
		(D, n, d)	△	M	外部10進数に変換します。
		(B, n)	△	M	外部10進数に変換します。
		(R, 1)	－	M	実数がありません。
		(E, 編集文字列)	－	M	
		EDIT=’外部精度’	△	M	パターンNoに変換します。
		Pnn	○		
		OCCURS=n	○		
		(n, m)	○		
	SDATA (帳票)	NAME=データ名	○		プログラミング項目名となります。省略時、変数名がプログラミング項目名となります。
		USAGE=DATA	－	－	項目制御部付きとなります。
		CTRLn	－	－	項目制御部は5バイト固定です。
		PIC=(X, w)	○		
		(N, w)	○		
		(M, w)	○		
		(9, n, d)	○		
		(S, n, d)	○		
		(C, n, d)	△	M	外部10進数に変換します。
		(D, n, d)	△	M	外部10進数に変換します。
		(B, n)	△	M	外部10進数に変換します。
		(R, 1)	－	M	実数がありません。
		(E, 編集文字列)	－	M	
		EDIT=’外部精度’	△	M	パターンNoに変換します。
		Pnn	○		
		OCCURS=m	○		
グラフ レコード名 (続く)	GRECORD	GRAPH=(グラフ種, グラフパターン名)	×	M	グラフ機能はありません。
	PDATA	UNIT=グラフユニット数			
		ELEMENT=グラフ要素数			
		MASK=省略			
		PIC=省略			
		CLASS=省略			
	UNIT	NAME=グラフユニット名			
		PIC=省略			
	GDATA	TEXT=省略			
		REFINE=省略			
		PIC=省略			

名称	命令	オペランド	変換可否	警告の有無	留意事項
グラフレコード名(続く)	DEVICE(画面)	DVK=DP10	○		
		SIZE=(行数,列数)	△	M	行数:1～100 列数:1～240 行数×列数<=10000
		DEKETE=(C1,C2,C3,C4)	－	－	機能がありません。
		GPLANE=(縦方向ドット数, 横方向ドット数)	×	M	NDG専用情報です。
		ATTKEY= (ATTNコード:文字種)	△	M	
		REDUC=省略	－	－	
	FRAME(画面)	FLENGTH=FIX	○		
		VAR	－	M	すべて固定とします。
		RETAIN=YES	－	－	
		NO	○		
		VSIZE=省略	×	M	論理画面はありません。
		VSCROLL=省略			
		HSCROLL=省略			
		FIX=省略			
		DRCTFIX=省略			
		PHASE=省略			
		SYSRVS=NO	○		
		BOTTOM	－	M	ACS専用です。
	PART(画面)(続く)	WRITE=ERASE	○		
		PARTIAL	－	M	上書き表示はありません。
		CONTROL=(|ALARM|)	－		
		|LOCK|	－		
		|RESETMDT|	－		
		|BOTRESET|	－		
		|PRINT|	－		
		|RETRY|	○		
		AKCHECK=YES	○		
		NO	－	M	すべて表示します。
		ATT= (アテンションコード,…)	△	M	
		AEMSG=(変数名, メッセージID)	－	－	メッセージ機能はありません。
		VEMSG=(変数名, メッセージID)	－	－	メッセージ機能はありません。
		TYPE=GRAPHIC	×	M	NGD専用情報です。
		IMG	×	M	メッセージ出力はありません。
		ORG=(縦ドット数, 横ドット数)	×	M	NGD専用情報です。
		URLBASE=(データ)	－	－	
		URLBCONV=YES	－	－	
		NO	－	－	
		IDCR=データ項目名	－	－	IDカード機能はありません。
		SIZE=(行数,列数)	－	－	パーティションサイズはDEVICE文によります。

名称	命令	オペランド	変換可否	警告の有無	留意事項
グラフ レコード名 (続き)	PART (画面) (続き)	REC=レコードID	－	－	
		DUMMY	－	－	
‘x’ (続く)	SCHAR (画面) (続く)	TYPE=ATTR	○		
		SEP	○		
		CHAR	○		
		FILED=TEXT	○		
		=IN	○		
		=OUT	○		
		=END	○		
		ATTR=(|Ixx|)	○		
		|Oxx|	○		
		|xDx|	－	－	ライトペン機能はありません。
		|xUx|	○		
		|xxH|	○		
		|xxL|	○		
		|xxS|	○		
		|NUM|	△	－	数字項目以外では無視します。
		|ALP|	○		
		|CUR|	－	－	機能がありません。
		|MDT|	－	－	機能がありません。
		|FIL|	○		
		|END|	○		
		|TRG|	－	－	機能がありません。
		|NOC|	－	－	機能がありません。
		|AEN|	○		
		|FXR|	○		
		|UDL|	○		
		|OVL|	○		
		|VTL|	○		
		|VTR|	○		
		|OTL|	○		
		|SPF|	○		
		|NSH|	○		
		|PTN|	－	－	パターン表示機能はありません
		|BLK|	○		
		|RVS|	○		
		|UDS|	－	－	機能がありません。
		|BLU|	○		
		|RED|	○		
		|PNK|	○		
		|GRN|	○		
		|TUR|	○		
		|YLW|	○		
		|WHT|	○		
		ATTR2=(|省略|)	△	－	先頭入出力フィールドのATTR2属性をエラー強調属性に変換します。

名称	命令	オペランド	変換可否	警告の有無	留意事項
‘x’ (続き)	SCHAR (画面) (続き)	DISPLAY=ABLE	－	－	すべて項目制御部に従います。
		RONLY	－	－	
		LPEN=フィールド名	－	－	ライトペン機能はありません。
		SEP=PARA	○		
		SYSFIG	○		
		END	○		
		TYPE=GRAPHIC	×	M	グラフ機能表示はありません。
ウィンドウ名	SWINDOW (画面)	IMG	○		
		FIG	○		
		GRAPH=(グラフ種, グラフパターン名	×	－	グラフ機能表示はありません。
		PIC=省略	×		グラフ機能表示はありません。
		LENGCHK=YES	－	－	
	LAYOUT (画面)	NO	－	－	
		CRTPIC=ALP	－	－	
		ATTR	－	－	
		RECTYPE=INOUT	○		
		IN/OUT	－	－	INOUTとみなします。
	LAYEND	POS=(行位置,列位置)	○		
フィールド名 (続く)	FIELD (画面)	ATTR=(属性表示文字,…)			
		ATTR2=(属性表示文字,…)			
		VALUE=(データ,…) データ：	△	M	サブフィールド形式で指定した場合、変換できないことがあります。複数行にまたがるフィールドは指定できません。
		変数名	○		
		(変数名,n)	○		
		(変数名,n,m)	○		
		文字列リテラル	○		
		X’16進数’	－	M	
		(ALL,繰り返し文字,n)	○		
		システム表意定数	△	M	ページ表意定数は指定できません
		日本語定数	○		
		スペース文字	○		
		MSG=メッセージデータの名札	－	－	メッセージ定義はありません。
		LENGTH=フィールド長	○		項目領域長とします。
		CATTR=(属性表意定数,…)	－	－	対応するサブフィールドが変換可能な場合だけです。
		DISPLAY=ABLE	－	－	項目制御部に従います。
		RONLY	－	－	
		DATAIN=変数名	△	M	入出力が列レコードの場合、入出力共用レコードに合成します。入力レコードのみの場合は変換しません。
		(変数名,n)			
		(変数名,n,m)			
		DUMMY			

名称	命令	オペランド	変換可否	警告の有無	留意事項
フィールド名(続く)	WINDOW(画面)	GREC=グラフレコード名	×	M	グラフ表示機能はありません。
		IMG=イメージデータ名	○		
		FIG=図形データ名	○		
		ORG=(行数,列数)	△	M	ドット数による指定はできません。
		(1,1)	○		
		SIZE=(行数,列数)	△	M	ドット数による指定はできません。
		FULL	○		
		PRESS=NO	－	－	
		NH	－	－	
		MR2	－	－	
		MR4	－	－	
		DENSITY=1	－	－	
		2	－	－	
		3	－	－	
		4	－	－	
	DEVICE(帳票)(続く)	DVK=SP10	○		区別はありません。
		PR10	○		
		PR20	○		
		OPTION=(|VECTOR|)	－	－	
		|HSKIP|	－	－	
		|IMAGE|	－	－	
		|TAB |	－	－	
		COPY=n	－	－	
		OVLY=(オーバレイID, …)	○		
		SUPPLY=ANY	○		
		CASET1	○		
		CASET2	○		
		CASET3	○		
		MANUAL	○		
		STACKER=ANY	－	－	
		STACK1	－	－	
		STACK2	－	－	
		STACK3	－	－	
		PSIDE=FRONT	－	－	
		(BOTH,FACE)	－	－	
		BACK	－	－	
		AUTONL=YES	○		
		NO	－	－	YESとみなす
		FROM=(JISA3,V)	○		
		(JISA3,H)	○		
		(JISA3,L)	×	M	
		(JISA4,V)	○		
		(JISA4,H)	○		
		(JISA4,L)	○		

名称	命令	オペランド	変換可否	警告の有無	留意事項
フィールド名(続く)	DEVICE(帳票)(続き)	(JISA5,V)	○		
		(JISA5,H)	○		
		(JISA5,L)	×	M	
		(JISA6,V)	○		
		(JISA6,H)	○		
		(JISA6,L)	×	M	
		(JISB4,V)	○		
		(JISB4,H)	○		
		(JISB4,L)	○		
		(JISB5,V)	○		
		(JISB5,H)	○		
		(JISB5,L)	×	M	
		(LETTER,V)	○		
		(LETTER,H)	○		
		(LETTER,L)	○		
		(A4/B4,V)	○		
		(A4/B4,H)	○		
		(A4/B4,L)	○		
		(A5/B5,V)	○		
		(A5/B5,H)	○		
		(A5/B5,L)	×	M	
		(B4/A3,V)	○		
		(B4/A3,H)	○		
		(B4/A3,L)	×	M	
		(B5/A4,V)	○		
		(B5/A4,H)	○		
		(B5/A4,L)	×	M	
		(LETTER/B4,V)	△		B4とみなします。
		(LETTER/B4,H)	△		B4とみなします。
		(LETTER/B4,L)	△		B4とみなします。
		DOT=TYPE1	－	－	
		TYPE2	○		
		SIZE=省略	－	－	
		REDUC=省略	－	－	
	FRAME(帳票)(続く)	HALF=ALL	－	－	
		(|BLANK|)	－	－	
		|NUM |	－	－	
		|ALP |	－	－	
		|SALP |	－	－	
		|KANA |	－	－	
		USER	－	－	
		TOPM=(n,m)	－	－	
		RETAIN=YES	－	－	
		NO	○		
		DIRECT=H	○		印刷方向は項目毎に設定可能（FIELD文参照）です。
		V	○		

名称	命令	オペランド	変換可否	警告の有無	留意事項
フィールド名(続く)	FRAME(帳票)(続き)	VPOS=FIXED	－	－	
		FLOAT	－	－	
		EJECT=BGNP	－	－	
		ENDP	－	－	
		SPACE	－	－	
		NO	－	－	
	PART(帳票)	REC=レコードID	△	M	複数パーティションの場合、先頭のみ。
		DUMMY			
		TYPE=FIXED	○		
		FLOATA	－	－	
		FLOAT	△	M	先頭パーティションのみ処理。
		IMG	×	M	イメージ出力ありません。
		GRAPHIC	×	M	グラフ機能ありません。
		ANCPI=10	○		
		12	×	M	12CPIありません。
		15	×	M	15CPIありません。
		NPO=12	△	－	２ビットに変換します。
		9	△	－	BCPI=10Aなら、1.5ピッチに変換します。
		7	○		
		BCPI=15	×	M	
		12	×	M	
		10	○		
		10A	○		
		8	×	M	
		5	×	M	
		SIZE=(行数,列数)	△		1<=行数<=255, 1<=列数<=255
		CTYPE=M	○		
		G	○		
		CATTR=BLU	○		色は項目毎に設定可能（FIELD文参照）です。
		RED	○		
		PNK	○		
		GRN	○		
		TUR	○		
		YLW	○		
		BLK	○		
		ORG=(縦ドット数, 横ドット数)	×		イメージパーティション情報
	LINE(帳票)	POS=行数	○		
		LPI=12	○		
		8	○		
		6	○		
	WINDOW(帳票)(続く)	GREC=グラフレコード名	×	M	グラフ表示機能はありません。
		IMG=イメージデータ名	○		
		FIG=図形データ名	○		
		ORG=(行数,列数)	○		

名称	命令	オペランド	変換可否	警告の有無	留意事項
フィールド名(続く)	WINDOW(帳票)(続き)	(1,1)	○		
		SIZE=(行数,列数)	△	M	ドット数による指定はできません。
		FULL	○		
		PRESS=NO	－	－	
		NH	－	－	
		MR2	－	－	
		MR4	－	－	
		DENSITY=1	－	－	
		2	－	－	
		3	－	－	
		4	－	－	
	FIELD(帳票)(続く)	POS=行数	○		
		(行数,列数)	○		
		VALUE=(データ,…) データ：			
		変数名	○		
		(変数名,n)	○		
		(変数名,n,m)	○		
		文字列リテラル	○		
		X’16進数’	－	M	
		(ALL,繰り返し文字,n)	○		
		システム表意定数	○		
		日本語定数	○		
		スペース文字	○		
		書式文字	○		
		POINT=(文字サイズ, …)			省略時はPART文のPOINTオペランドに従います。
		12	△		２ピッチに変換します。
		9	△		PART文でBCPI=10Aなら、1.5ピッチに変換します。
		7	○		
		H	○		
		D	○		
		DH	○		
		W	○		
		WH	○		
		L	○		
		LH	○		
		IX	○		
		SX	○		
		CATTR=(文字色, …)	○		省略時はPART文のCATTRオペランドに従います。
		CTYPE=(M, …)	○		省略時はPART文のCTYPEオペランドに従います。
		G	○		
		SVALUE=(データ, …)	－	M	通常創出機能はありません。
		SPOINT=(文字サイズ, …)			

名称	命令	オペランド	変換可否	警告の有無	留意事項
フィールド名(続く)	FIELD(帳票)(続き)	SCATTR=(文字色, …)			
		SCTYPE=(文字形態, …)			
		RVALUE=(データ, …)	－	M	通常創出機能はありません
		RPOINT=(文字サイズ, …)			
		RCATTR=(文字色, …)			
		RCTYPE=(文字形態, …)			
		KLINE=(|OVL|)	－		
		|UDL|	－		
		|VTL|	－		
		|OTL|	－		
		|MSL|	－		
		PREFIX=X’16進数’	－	M	
		SUFFIX=X’16進数’	－	M	
		LENGTH=フィールド長	○		項目領域長とします、
		DIRECT=H	○		省略時はFRAME文のDIRECTオペランドに従います。
		V	○		
	BARCODE(帳票)	POS=(行数,列数)	○		
		TYPE=NW7	○		
		JAN	○		
		JANS	○		
		C39	○		
		D25	○		
		L25	○		
		VALUE=変数名	○		
		(変数名,n)	○		
		BWIDTH=2/160	○		
		=3/160	○		
		=4/160	○		
		=2/240	○		
		=3/240	○		
		=4/240	○		
		=5/240	○		
		=6/240	○		
		HIGHT=n	○		
		CHKCHAR=YES	○		
		NO	○		
		UDRCHAR=YES	○		
		NO	○		
		FLAGCHAR=STD	○		
		LOW	○		
	XBARCODE(帳票)	省略	－	M	範囲指定バーコードは変換できません。
	SHADOW(帳票)	AREA=(行,列,行,列)	－	－	
		PAINT=パターン	－	－	
		COLOR=色	－	－	

名称	命令	オペランド	変換可否	警告の有無	留意事項
フィールド名(続き)	PLINE(帳票)	UDL=(行位置, 列位置,長さ)	－	－	
		OVL=(行位置, 列位置,長さ)	－	－	
		VTL=(行位置, 列位置,長さ)	－	－	
		VTR=(行位置, 列位置,長さ)	－	－	
	REPEAT(タイプ1)	COLOR=色	－	－	
		LKIND=(線種,太さ)	－	－	
		OCCURS=n1	○		
		LINE=1	○		
		BY=1	○		
		M1			
	REPEAT(タイプ2)	LINE=(n1,1)	○		
		ROW=(n2,r)	○		
		BY1=1	○		
		M1			
		BY2=1	○		
		M2			
	REPEND	－	－	－	
	HREF	省略	－	－	
	END	－	－	－	
	BACK	省略	－	M	バックグラウンドありません。
	SET	省略			
	DRAW	省略			
	CHAR	省略			
	PREPROC(画面)	0	－	－	組み込みプロシジャ機能ありません。コマンドプロシジャは無視します(注)。
	EPIPROC(画面)	0	－	－	組み込みプロシジャ機能ありません。コマンドプロシジャは無視します(注)。
	PROCEND(画面)	－	－	－	組み込みプロシジャ機能ありません。
コントロールエントリテーブルID	MEDCTRL	TYPE=省略	×	M	MEDGEN制御文
		VALUE=省略			
	CONVRT	省略	×	M	
	SOCLIB	省略	×	M	OSIV/F2制御文
	INCLUDE	省略			
	FMTLIB	省略			

(注)移出機能をバッチ起動した場合、組み込みプロシジャ機能を記述したフォーマット定義ソースはFMTGENのエラーとなり移出できません。組み込みプロシジャを削除するか、または移出機能をバッチTMP配下で起動してください。

表B-2 フォーマット定義体ソース変換時の留意事項２

アテンションキー種別	アテンションコードの扱い		移出時の処理
	ホスト	PC	
CLEARキー	C000	C000	なし
PAキー	A00n(1<=n<=3)	A00n(1<=n<=3)	なし
PFキー	F0nn(1<=nn<=99)	F0nn(1<=nn<=99)	メッセージ出力
ライトペン	L000	S000	メッセージ出力
IDカード	I000	I000	なし
トリガフィールドからのカーソル抜け	T000	－	メッセージ出力
実行キー	E000	E000	なし

B.2 オーバレイ定義体移行時の留意事項

ADJUSTのJODFRTRNを使用して、イメージライブラリに格納されているホスト形式のフォームオーバレイを流通形式のオーバレイ定義体に変換する際に、次の留意点があります。

なお、表中の記号は次の意味があります。

ホストでの変換の可否

○：変換可能
△：制限あり
－：機能なし。

表B-3　オーバレイ定義体移行時の留意事項

項目			サポート状況		備考
			MSP/XSP	PC/UNIX	
基本座標系の設定			○	×	
用紙サイズ	B4 B5 A3 A4 A5		○	○	(注1)任意指定可能
	A6		○	○	
	レター		○	○	
	リーガル		×(注1)	×(注1)	
	自由		○	○	
用紙方向	縦		○	○	
	横		○	○	
	LP		○	○	
ドット密度	160dpi		×	○	(注1)高速LBPは解像度無依存で指定可能
	180dpi		×	×	
	240dpi		○(注1)	○	
	400dpi		×(注1)	○	
イメージデータの組み込み			○(注1)	○(注2)	(注1)VCTは不可 (注2)V1.2L21以降
全体色	７色		×(注1)	○(注2)	(注1)DOTは黒/赤/緑/青で可能 (注2)個々に指定可能
文字 (続く)	回転	文字回転:４方	○	○	
		文字列回転: ４方縦文字変換	○	○	
			○	○	
	書体	明朝体	○		(注1)VCT印刷時、ゴシック体CGが必要 (注2)高速LBPは可能
		ゴシック体	○(注1)		
		欧文書体等	×(注2)		
	配置	一次元	○	○	
		二次元	○	×	
		行中央	○	○	
		桁中央	×	○	
		複数行	○	△(注1)	(注1)繰り返しで可能
	種類	ストローク文字	×	×	
		斜体 0～90度	×	○(注1)	(注1)90,75,60,45度だけ
		利用者定義文字	○	○	
	揃え	中央揃え	×	○	
		左揃え	○(注1)	○	(注1)無条件で左揃え
	色	８色	○	○(注1)	(注1)UNIX系は白を除く７色

項目				サポート状況		備考
				MSP/XSP	PC/UNIX	
文字 (続く)	間隔	文字サイズ間隔		○	○(注1)	(注1)文字サイズを１として、 -0.9～99.9
		メモリ間隔		○	－	
	サイズ (続く)	4.5ポ	全角	○	○	(注1) (注2)自由サイズ文字 (999×999ドットで対応可能) (注3)12ポは未サポート (自由サイズで近似可能) (注4)高速LBPは可能 (注5)10.5ポは160dpiローカルプリンタ用だけ設定可能 (注6)9ポ倍角 (注7)9ポ (注8)VCTは設定不可能 (自由サイズで対応可能) (注9)18ポは読み込み時に次のように変換される。 標準→9ポ倍角 半角→9ポ長体 倍角→36ポ標準 長体→36ポ半角 平体→18ポ平体 上添→9ポ標準 下添→9ポ標準 (注10)近いCGフォントで近似 (注11) 上添は下添になる (注12)1～180ポまで
			半角	○	○	
			倍長平	×(注2)	○	
			半＋ 倍長平	×(注2)	○	
			上添 下添	○	○ (注11)	
		7ポ 9ポ 12ポ	全角	○	○	
			半角	○	○	
			倍長平	×(注2)	○	
			半＋ 倍長平	×(注2)	○	
			上添 下添	△(注3)	○ (注11)	
		10.5 ポ	全角	×(注2) (注4)	○	
			半角	×(注2)	○	
			倍長平	×(注2)	○	
			半＋ 倍長平	×(注2)	○	
			上添 下添	×(注2)	○(注11)	
		18ポ	全角	○	○(注9)	
			半角	○(注8)	○(注9)	
			倍長平	×(注2)	○(注9)	
			半＋ 倍長平	×(注2)	○(注9)	
			上添 下添	×(注2)	○(注9)	
		36ポ	全角	○(注8)	△(注10)	
			半角	×(注2)	○	
			倍長平	×(注2)	○	
			半＋ 倍長平	×(注2)	○	
			上添 下添	×(注2)	○ (注11)	
		1 ～ 300ポ 任意	全角	×(注2) (注4)	○ (注12)	
			半角	×(注2) (注4)	○ (注12)	
			倍長平	×(注2)	○	
			半＋ 倍長平	×(注2)	○	
			上添 下添	×(注2)	○ (注11)	

項目			サポート状況		備考
			MSP/XSP	PC/UNIX	
文字(続き)	サイズ(続き)	任意文字幅	○	○ (注13)	(注12)PowerFORMで定義可能
線分 - 水平 - 垂直 - 斜線		実線	○	○	(注1)破線で代用 (注2)点線で代用 (注3)線幅は1～64ドット
		点線	○(注1)	○	
		一点鎖線	○	○	
		破線	○	○	
		任意ピッチ	○	○	
		線幅	○	○(注3)	
		線色(７色)	×	○	
円		実線	○	○	(注1)破線で代用 (注2)VCTは固定ピッチ・太さ制限 (注3)固定ピッチ・太さ制限 (注4)固定ピッチ・固定太さ (注5)VCTは不可 (注6)線幅は1～64ドット (注7)半径未満 (注8)1/240 3/240 5/240 7/240 9/240
		点線	○(注1) (注2)	○	
		一点鎖線	×	○	
		破線	○(注2)	○	
		任意ピッチ	○(注5)	○	
		線幅	○	○(注6)	
		線色(７色)	×	○	
網がけ矩形		濃さ　10段階	○(注1)	○(注2)	(注1)高速LBP、VCTでは濃さは近似 (注2)濃さは近似
		角丸め	○	○	
		色(７色)	×	○	
枠		実線	○	○	(注1)矩形だけ (注2)DOT、高速LBPは破線で代用 (注3)VCTは不可 (注4)破線で代用 (注5)破線で代用 (注6)高速LBPでは円弧と線分がきれいに交わらない (注7) 固定ピッチ・固定太さ (注8)円弧と線分がきれいに交わらない
		点線	×(注2)	○	
		一点鎖線	○(注3)	○	
		破線	○(注3)	○	
		任意ピッチ	○	○	
		線幅	○	○	
		角丸め　実線	○	○	
		角丸め　点線、一点鎖線	○(注3) (注6)	○(注8)	
		色(７色)	○	○	
定型パターン９種類		実線	○	○	(注1)８種類 (注2)枠で代行 (注3)線幅は1～64ドット
		点線	×(注2)	○	
		一点鎖線	×(注2)	○	
		破線	○	○	
		任意ピッチ	○	○(注3)	
		線幅	○	○	
		色(７色)	×	○	

付録C COBOL85非互換指摘機能

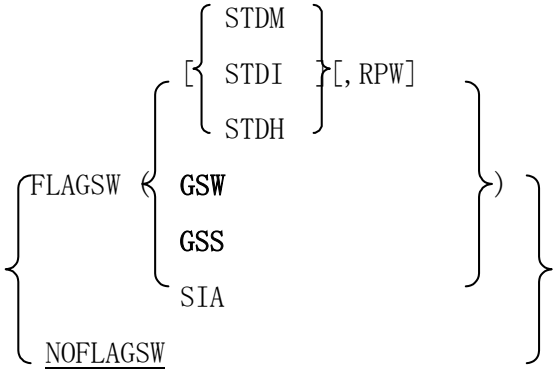
富士通のオープン系COBOL製品であるNetCOBOLには、OSIV系のCOBOL85との仕様差を指摘するための機能があります。ここでは、その機能について、以下のことを説明します。

- 使用法
- 指摘項目

C.1 使用法

NetCOBOLにおいて、COBOL85非互換項目の指摘機能を有効にするには、翻訳オプションFLAGSWを指定します。

翻訳オプションFLAGSWは、翻訳したCOBOLソース中に記述されているCOBOLの言語要素を指摘する機能を有効にするもので、次の指定形式を持ちます。



FLAGSWオプションのサブオペランドとして、GSWまたはGSSを指定した場合、COBOL85非互換項目に対する指摘メッセージが出力されるようになります。どちらのサブオペランドを指定するかで、指摘される項目は一部異なります。

- FLAGSW(GSW):
  COBOL85の非互換項目の一部だけを指摘します。
  AADアプリケーション要の分散開発支援機能(配付ソース生成)と組み合わせて使用し、この機能で処理できない項目のみを指摘します。
- FLAGSW(GSS):
  COBOL85の非互換項目のすべてを指摘します。通常のOSIVプログラムの分散開発時に指定します。

C.2 指摘対象項目一覧

COBOL85非互換項目の指摘機能の対象となる言語の要素についての一覧表を以下に示します。

表C-1 COBOL85非互換項目の指摘機能の対象となる言語の要素（続く）

機能分類			機能差		FLAGSWオプションの指定と指摘の有無	
大分類	中分類	詳細	種別	COBOL85での翻訳結果	GSW	GSS
COBOLの語	利用者語	文字“_”	×	JMN1252I-S or JMN1000I-S	JMN9011I-W	JMN9011I-W
		英小文字	△		－	－
		半角かな	△		－	－
		英小文字、半角かな混在	×	JMN2504I-S他	－	－
		8文字を越える入口名	×	JMN1108I-S or JMN1140I-S	×	JMN9902I-E
		ハイフンを含む入口名	△	JMN1548I-I	×	JMN9904I-W
		英数字以外の文字を含む入口名	×	JMN1107I-S or JMN1252I-S	×	JMN9903I-E
	特殊レジスタ	PROGRAM-STATUS	△	ノーエラー	－	－
		RETURN-CODE	△	ノーエラー	－	－
	機能名	SYSERR	×	JMN1216I-S	JMN9909I-E	JMN9909I-E
		SWITCH-8	×	JMN1216I-S	JMN9909I-E	JMN9909I-E
		ENVIRONMENT-NAME	×	JMN1216I-S	JMN9909I-E	JMN9909I-E
		ENVIRONMENT-VALUE	×	JMN1216I-S	JMN9909I-E	JMN9909I-E
		ARGUMENT-NUMBER	×	JMN1216I-S	JMN9909I-E	JMN9909I-E
		ARGUMENT-VALUE	×	JMN1216I-S	JMN9909I-E	JMN9909I-E
		その他の機能名(SORT/印刷)	○		×	×
	定数	16進文字定数	○	実行結果に差異	×	×
		日本語16進文字定数	×	JMN1038I-S or 実行結果に差異		JMN9908I-W
		浮動小数点定数	△	演算結果に差異	×	×
	引用符文字	自動判定	×	JMN1000I-S他	JMN9907I-E	JMN9907I-E
式	連結式	連結式の使用	×	JMN1000I-S他	JMN9911I-E	JMN9911I-E
	字類条件	JAPANESE	×	JMN2503I-S他	JMN9929I-E	JMN9929I-E
一意名（続く）	組み込み関数（続く）	ACOS	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		ANNUITY	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		ASIN	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		ATAN	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		CAST-ALPHANUMERIC	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		CHAR	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		COS	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		CURRENT-DATE	○	ノーエラー	×	×
		DATE-OF-INTEGER	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		DAY-OF-INTEGER	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		FACTORIAL	×	JMN1608I-S	JMN9910I-E	JMN9910I-E

機能分類			機能差		FLAGSWオプションの指定と指摘の有無	
大分類	中分類	詳細	種別	COBOL85での翻訳結果	GSW	GSS
一意名（続き）	組み込み関数（続き）	INTEGER	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		INTEGER-OF-DATE	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		INTEGER-OF-DAY	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		INTEGER-PART	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		LENGTH	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		LOG	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		LOG10	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		LOWER-CASE	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		MAX	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		MEAN	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		MEDIAN	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		MIDRANGE	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		MIN	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		MOD	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		NATIONAL	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		NUMVAL	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		NUMVAL-C	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		ORD	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		ORD-MAX	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		ORD-MIN	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		PRESENT-VALUE	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		RANDOM	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		RANGE	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		REM	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		REVERSE	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		SIN	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		SQRT	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		STANDARD-DEVIATION	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		STORED-CHAR-LENGTH	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		SUM	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		TAN	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		UCS2-OF	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		UPPER-CASE	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		UTF8-OF	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		VARIANCE	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
		WHEN-COMPILED	×	JMN1608I-S	JMN9910I-E	JMN9910I-E
	オブジェクト一意名	メソッドの行内呼出し	×	JMN2503I-S	JMN9910I-E	JMN9910I-E
		オブジェクト指定子	×	JMN2503I-S	クラスID段落、メソッドID段落、リポジトリ段落がチェックで代用	
		EXCEPTION-OBJECT	×	JMN2503I-S		
		NULL	×			
		SELFとSUPER	×	JMN2503I-S		
		オブジェクトプロパティ	×	JMN2503I-S		
		クラス名	×	JMN2503I-S		
正書法		自由形式	×	JMN1102I-S	JMN9900I-E	JMN9900I-E

機能分類			機能差		FLAGSWオプションの指定と指摘の有無	
大分類	中分類	詳細	種別	COBOL85での翻訳結果	GSW	GSS
見出し部	部の見出し	省略	×	JMN1102I-S	JMN9913I-E	JMN9913I-E
	段落	クラスID段落	×	JMN1104I-S	JMN9912I-E	JMN9912I-E
		FACTROY段落	×	JMN1291I-S	クラスID段落のチェックで代用	
		OBJECT段落	×	JMN1291I-S		
		メソッドID段落	×	JMN1104I-S or JMN1129I-S	JMN9912I-E	JMN9912I-E
	AS“定数”	外部名指定	×	JMN1356I-W	JMN9906I-E	JMN9906I-E
	終り見出し	END CLASS	×	JMN1004I-W	クラスID段落のチッェクで代用	
		END FACTORY	×	JMN1004I-W		
		END OBJECT	×	JMN1004I-W		
		END METHOD	×	JMN1004I-W	クラスID段落/メソッドID段落のチッェクで代用	
環境部	構成節 特殊名段落	ALPHABET句のEBCDIC指定	○	ノーエラー	×	×
		CURSOR句	×	JMN1216I-S	JMN9914I-S	JMN9914I-S
		CRT STATUS句	×	JMN1216I-S	JMN9914I-S	JMN9914I-S
		CONSOLE IS CRT句	△	ノーエラー。通常の機能名句として解釈される。	×	×
	構成節 リポジトリ段落	段落見出し	×	JMN1123I-S and JMN1004I-W	JMN9912I-E	JMN9912I-E
		クラス指定子	×	段落見出しエラーのため解析されない	リポジトリ段落見出しのチェックで代用	
		プロパティ指定子	×			
	入出力節 ファイル管理段落	ASSIGN PRINTER	△	ノーエラー。異なる意味に解釈。	JMN2763I-W	JMN2763I-W
		ASSIGN PRINTER-n	×	JMN1330I-S	JMN2763I-W	JMN2763I-W
		ASSIGN DISK	×	ノーエラー。異なる意味に解釈。	JMN2763I-W	JMN2763I-W
		ASSIGN ″定数″ (整列併合機能)	×	JMN2896I-S	JMN9915I-E	JMN9915I-E
		ASSIGN データ名	△	異なる意味に解釈。		
		ORGANIZATION LINE SEQUENTAIL	×	JMN1127I-S	JMN9916I-E	JMN9916I-E
		LOCK MODE句	×	JMN1123I-S	JMN9914I-E	JMN9914-E
データ部（続く）	画面節	見出し	×	JMN1123I-S	JMN9917I-E	JMN9917I-E
		画面節データ	×	見出しがエラーとなるため、作業場所データ項目としてチェックされる	JMN9917I-E	JMN9917I-E
	名前付き定数(続く)	定義	×	JMN1123I-S 、JMN1123I-S	JMN9918I-E	JMN9918I-E

機能分類			機能差		FLAGSWオプションの指定と指摘の有無	
大分類	中分類	詳細	種別	COBOL85での翻訳結果	GSW	GSS
データ部（続き）	名前付き定数(続き)	参照	×	JMN1143I-S(PICTURE句) JMN2039I-S(VALUE句) JMN2503I-S(手続き部)	定義側のチェックで代用	
	型	型定義(TYPEDEF句)	×	JMN1123I-S	JMN9919I-E	JMN9919I-E
		型を参照してのデータ記述(TYPE句)	×	JMN1123I-S、 JMN2222I-S	定義側のチェックで代用	
	句	ANY LENGTH句	×	JMN1123I-S	クラス定義･メソッド定義のチェックで代用	
		PROPERTY句	×	JMN1123I-S	クラス定義のチェックで代用	
	データ型	COMP-1	△	内部表現が異なる	チェックせず	
		COMP-2	△	内部表現が異なる	チェックせず	
		COMP-5	×	JMN9920I-E	JMN9920I-E	JMN9920I-E
		外部10進	△	内部表現が異なる	チェックせず	
		内部10進	△	内部表現が異なる	チェックせず	
		OBJECT REFERENCE	×	JMN1123I-S、 JMN2222I-S	JMN9920I-E	JMN9920I-E
		PROCEDURE-POINTER	×	JMN1123I-S、 JMN2222I-S	JMN9920I-E	JMN9920I-E
手続き部（続く）	手続き部見出し	WITH指定	×	JMN2500I-S	JMN9921I-E	JMN9921I-E
		RETURNING指定	×	JMN2503I-S	JMN9921I-E	JMN9921I-E
		RAISING指定	×	JMN2503I-S	JMN9921I-E	JMN9921I-E
	ACCEPT文	2進数/内部10進数を受け取り項目に(FROM SYSIN/CONSOLE時)	×	JMN3053I-S	JMN9924I-E	JMN9924I-E
		スクリーン機能(XPG互換)	×	異なる解釈がされている	JMN9925I-E	JMN9925I-E
		スクリーン機能(MF互換)	×	JMN2503I-S(宛先CRT)、 JMN2549I-S(WITH指定)、 JMN2500I-S(AT指定)	JMN9925I-E	JMN9925I-E
		環境変数操作	×	JMN2503I-S(宛先未定義)	機能名のチェックで代用	
		コマンド行操作	×	JMN2503I-S(宛先未定義)	機能名のチェックで代用	
	ADD文	浮動小数点数演算	△	実行結果が異なる場合がある	チェックせず	
	CALL文（続く）	プログラム名の長さの制限(8バイト)	×	JMN2531I-S	JMN9926I-E	JMN9926I-E
		英小文字を含むプログラム名	△	NOALPAHL指定時はJMN2532I-S	JMN9927I-E	JMN9927I-E

機能分類			機能差		FLAGSWオプションの指定と指摘の有無	
大分類	中分類	詳細	種別	COBOL85での翻訳結果	GSW	GSS
手続き部(続く)	CALL文(続き)	日本語プログラム名(文字定数)	×	JMN2532I-S	JMN9927I-E	JMN9927I-E
		日本語プログラム名(日本語定数)	×	JMN2532I-S	JMN9927I-E	JMN9927I-E
		一意名指定(日本語項目)	×	JMN3028I-S	JMN9929I-E	JMN9929I-E
		WITH指定	×	JMN2500I-S	JMN9923I-E	JMN9923I-E
		USING BY VALUE指定	×	JMN2518I-S	JMN9923I-E	JMN9923I-E
		RETURNING指定	×	JMN2503I-S	JMN9923I-E	JMN9923I-E
	CANCEL文	プログラム名の長さの制限(8バイト)	×	JMN2531I-S	JMN9926I-E	JMN9926I-E
		英小文字を含むプログラム名	△	NOALPAHL指定時はJMN2532I-S	JMN9927I-E	JMN9927I-E
		日本語プログラム名(文字定数)	×	JMN2532I-S	JMN9927I-E	JMN9927I-E
		日本語プログラム名(日本語定数)	×	JMN2532I-S	JMN9927I-E	JMN9927I-E
		一意名指定(日本語項目)	×	JMN3030I-S	JMN9929I-E	JMN9929I-E
	COMPUTE文	浮動小数点数演算	△	実行結果が異なる場合がある	チェックせず	
	DISPLAY文	スクリーン機能(XPG互換)	×	異なる解釈がされている	JMN9925I-E	JMN9925I-E
		スクリーン機能(MF互換)	×	JMN2503I-S(宛先CRT)、JMN2549I-S(WITH指定)、JMN2500I-S(AT指定)	JMN9925I-E	JMN9925I-E
		環境変数操作	×	JMN2503I-S(宛先未定義)	機能名のチェックで代用	
		コマンド行操作	×	JMN2503I-S(宛先未定義)	機能名のチェックで代用	
	DIVIDE文	浮動小数点数演算	△	実行結果が異なる場合がある	チェックせず	
	ENTRY文	プログラム名の長さの制限(8バイト)	×	JMN2531I-S	JMN9902I-E	JMN9902I-E
		英小文字を含むプログラム名	△	NOALPAHL指定時はJMN2532I-S	JMN9903I-E	JMN9903I-E
		日本語プログラム名(文字定数)	×	JMN2532I-S	JMN9903I-E	JMN9903I-E
		WITH指定	×	JMN2500I-S	JMN9923I-E	JMN9923I-E
	END PROGRAM文	RAISING指定	×	JMM2503I-S	リポジトリ段落のチェックで代用	
	END METHOD文		×	JMN2540I-W	クラスID段落/メソッドID段落のチッェクで代用	
		RAISING指定	×	上記エラーにともない読み飛ばし		

機能分類			機能差		FLAGSWオプションの指定と指摘の有無	
大分類	中分類	詳細	種別	COBOL85での翻訳結果	GSW	GSS
手続き部(続き)	EXIT PARAGRAPH文		×	JMN2503I-S	JMN9922I-E	JMN9922I-E
	EXIT SECTION文		×	JMN2500I-S	JMN9922I-E	JMN9922I-E
	INVOKE文		×	JMN2503I-S		
	MULTIPLY文	浮動小数点数演算	△	実行結果が異なる場合がある	チェックせず	
	OPEN文	WITH LOCK指定	×	JMN2513I-S	JMN9923I-E	JMN9923I-E
	RAISE文		×	JMN2503I-S		
	READ文	WITH LOCK指定	×	JMN2500I-S	JMN9923I-E	JMN9923I-E
		PREVIOUS指定(MF互換)	×	JMN2503I-S	JMN9923I-E	JMN9923I-E
	SET文	オブジェクト参照の転記	×	JMN3034I-S	オブジェクト参照データのチェックで代用	
		TO ENTRY指定	×	JMN2552I-S	手続きポインタデータのチェックで代用	
	START文	正順のKEY IS [< | NOT > | <=]	×	JMN2709I-S	JMN9931I-E	JMN9931I-E
	SUBTRACT文	浮動小数点数演算	△	実行結果が異なる場合がある	チェックせず	
	UNLOCK文		×	JMN2680I-W	JMN9922I-E	JMN9922I-E
	USE EXCEPTION文		×	JMN2505I-S	リポジトリ段落のチェックで代用	
原始文操作	COPY文	原文名定数	×		JMN9905I-E	JMN9905I-E

付録D NetCOBOL JEFオプション

NetCOBOL JEF オプション(以下、JEF オプションと略します)は、NetCOBOL に対して、Windows 2000またはWindows XP環境でEBCDIC/JEFコード系を使用する機能を提供するオプション製品です。ここでは、JEFオプションの製品概要とJEFオプション使用時の機能上の特徴と制約事項について説明します。

D.1 JEFオプションの概要

ここでは、NetCOBOL JEFオプションと言う製品について、以下のことを説明します。

- 適用条件
- 開発環境概要
- 運用環境概要
- 機能差概要

D.1.1 JEFオプションの適用条件

JEFオプションの適用には、次のハードウェア条件およびソフトウェア条件を満たすことが必要です。

表D-1　ハードウェア条件

カテゴリ	要件
マシンおよびメモリ	Windows 2000 または Windows XPが動作可能なマシンおよびメモリ
ハードディスク	9.0 MByte以上

表D-2　ソフトウェア条件

ソフトウェア	要件	備考
OS	Windows 2000　Windows XP	
NetCOBOL	V70L10以降	NetCOBOL JEFオプションの本体製品です。
SystemWalker/CharsetMGR または Interstage Charset Manager	V10.0以降	COBOLプログラム(定義体を含む)の記述にシフトJISの外字領域に割り当てられた文字を使用する場合に必要となります。
JEF拡張漢字サポート	V4.1 L10以降	COBOLアプリケーション 実行時に必要です。
MeFt	V7.0 L10以降	表示ファイルで帳票印刷や画面入出力を行う場合および印刷ファイルで帳票印刷を行う場合に使用します。
FORM	V7.0 L10以降	帳票・伝票イメージで、画面アクセスを行うプログラムを作成する場合に、画面や帳票を設計するために使用します。
FORMオーバレイ オプション	V7.0 L10以降	帳票作成およびオーバレイの作成作業を画面と対話しながら設計するために使用します。
PowerSORT	V1.0 L30　以降	NetCOBOL JEFオプションにおいて、ソート(整列)処理と、マージ(併合)処理およびファイルユーティリティ［整列］コマンドを使用する場合に必要です。

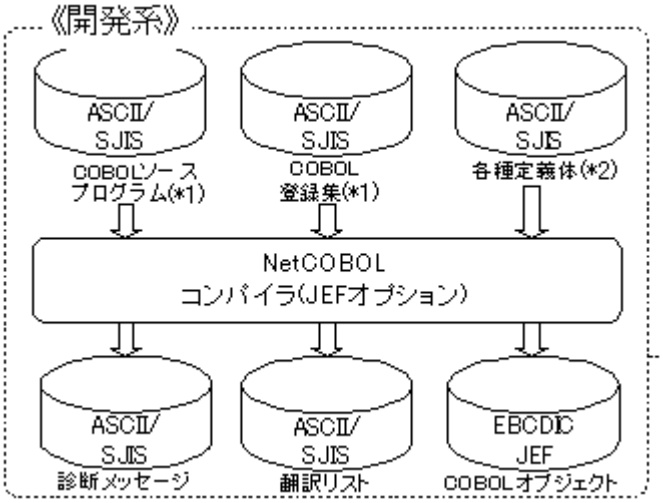
DPCLIB	V4.1 L60以降	データベース(SQL)機能のリモートデータベースアクセスを行う場合に必要となります。

D.1.2 JEF オプションの開発環境

JEFオプションを使用して開発するプログラムは、EBCDIC/JEFコード系を使用して動作しますが、開発作業そのものは、Windows系システムにおける通常のコード系であるASCII/シフトJIS コード系です。
JEF オプションを使用しての開発作業をASCII/シフトJIS コード系で行うのは、プログラム資産に外字が含まれる可能性は小さく、使い慣れたエディタを利用する利点の方が大きいからです。開発環境がASCII/シフトJIS コード系であるため、外字はプログラム中には16進定数で記述する必要がありますが、CharsetMGR を使用すれば、1880文字の範囲でプログラムが外字を利用することが可能になります。
なお、EBCDIC/JEFコード系の開発を行う場合であっても、コンパイラが生成する翻訳リスト、翻訳時メッセージなどはASCII/シフトJISコード系で出力します。

以下に、JEF オプションの開発環境の概要図を示します。

図D-1 JEF オプションの開発環境の概要



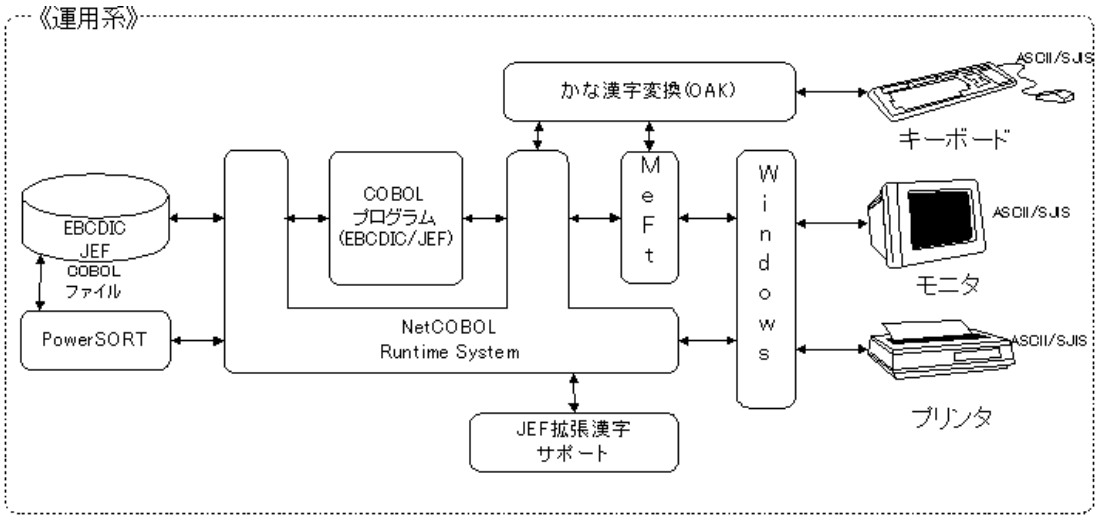
*1 外字に割り当てられた文字を使用する場合、CharsetMGR が必要となります。
*2 EBCDIC/JEFも可能(第１水準漢字、第２水準漢字、基本非漢字のみ使用可能)

D.1.3 JEF オプションの運用環境

JEF オプションの運用環境では文字コードとしてはEBCDIC/JEFコード系が用いられます。画面、帳票機能などでは、EBCDIC/JEF文字をすべてWindowsシステムの画面に表示でき、またプリンタに印刷できますが、そのためにはJEF 拡張漢字サポートがインストールされている必要があります。

以下に、JEF オプションの運用環境の概要図を示します。

図D-2　JEF オプションの開発環境の概要



なお、データベース機能を使用する場合の運用環境については、別途説明します。

D.1.4 JEF オプションの利用のメリットとデメリット

JEF オプションの導入によるメリット、デメリットを以下に示します。

メリット

- ASCII/シフトJIS コード系では、1880文字の外字(第１水準、第２水準、JIS 非漢字を除く文字)しか使用できませんが、EBCDIC/JEFコード系では、シフトJIS コード系の外字に相当する文字として、利用者定義文字3102文字、JEF 拡張漢字4039文字、JEF 拡張非漢字1010文字が使用できます。
- コード系がOSIV系システムと同じEBCDIC/JEFコード系であるため、プログラムのコード系に依存する処理に関してOSIV系システムと同じ動作が期待できます。このため、OSIV系システムで動作していたプログラムを変更することなく、プログラムの移行や分散開発が可能となります。
- データが外字を含む場合、その資産を移行は困難であったり不可能であったりしました。しかし、コード系がOSIV系システムと同じEBCDIC/JEFコード系であるために移行が容易になります。

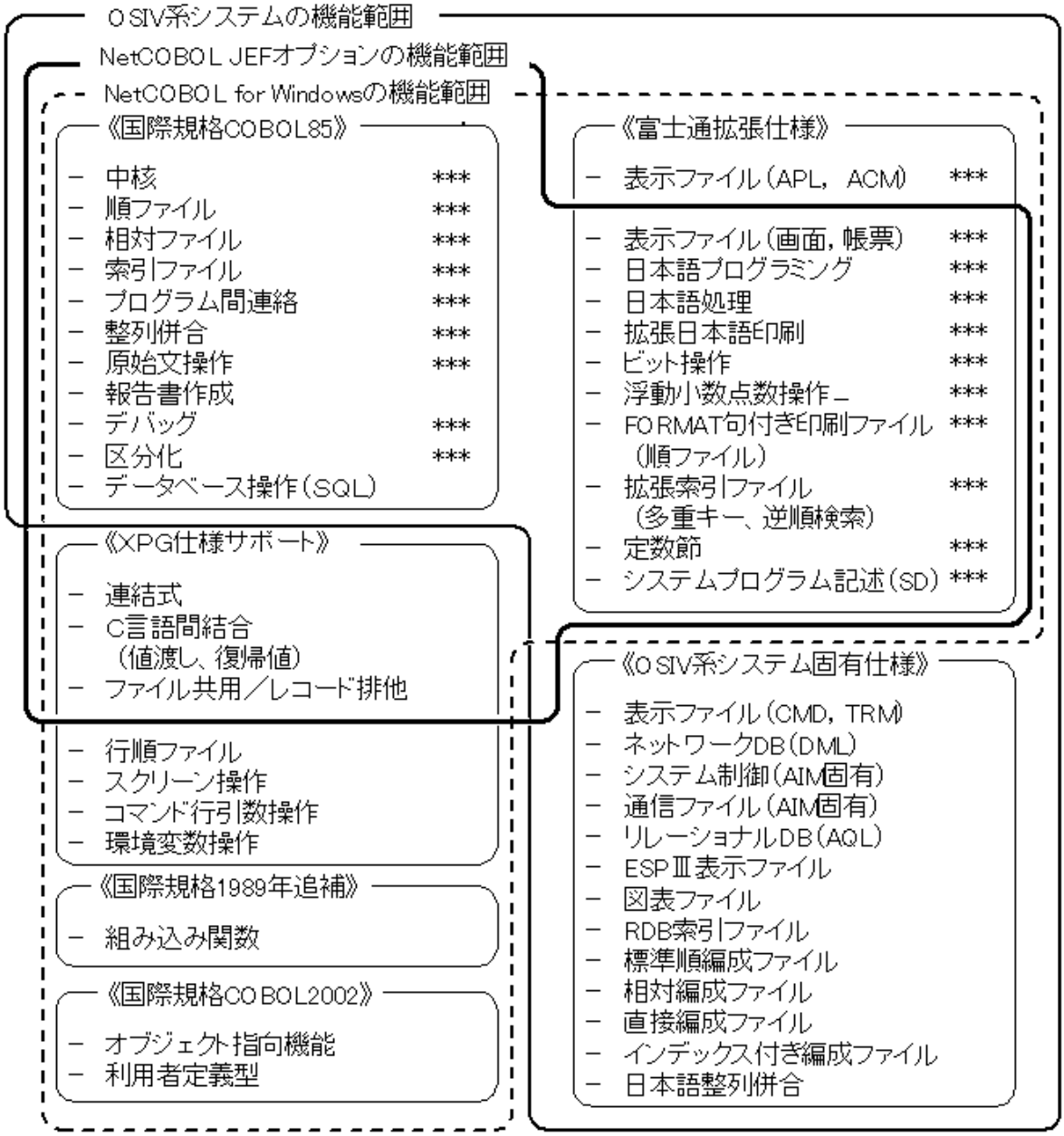
デメリット

- コード系がWindows系システムのコード系と異なるため、資産をNetCOBOL と他のアプリケーションの間で共有できなくなります。
- オブジェクト指向機能等のCOBOLの先進的な機能が使用できなくなります。
- Btrieve などのファイルシステムが利用できなくなります。

D.1.5 JEFオプションの機能概要

以下にJEFオプションとNetCOBOL およびOSIV COBOL85のサポート機能の機能範囲を示します。

図D-3 COBOL85およびNetCOBOL とJEFオプションの機能範囲



以下にJEFオプションがサポートしない機能について、“COBOL文法書”および“NetCOBOL 使用手引書”における記述箇所を示します。

表D-3 JEFオプション非サポート機能のマニュアル記述箇所一覧

制限となる機能	文法書(REF)/使用手引書(UG)の対象箇所
行順編成のファイル	REF 2.2.1 ファイルの編成 REF 4.3.1.17 ORGANIZATION句(順ファイル) UG 7.3 行順ファイルの使い方
組込み関数機能(注)	REF 2.7 組込み関数機能
スクリーン操作機能	REF 2.8 スクリーン操作機能
コマンド行引数の操作	REF 2.9 コマンド行引数と環境変数の操作機能 UG 11.2 コマンド行引数の取出し
環境変数の操作	REF 2.9 コマンド行引数と環境変数の操作機能

	UG　11.3 環境変数の操作機能
機能名SYSERR	REF 6.4.12　DISPLAY 文(中核)
Micro Focus 固有機能	REF 第10章　Micro Focus 固有機能
浮動小数点	REF 2.1.11　浮動小数点数の操作
オブジェクト指向機能	REF 第11 章オブジェクト指向プログラミング機能 UG　第14～17章 オブジェクト指向プログラミング
利用者定義型	REF 5.4.13　TYPE句 REF 5.4.14　TYPEDEF句 REF 付録G　型を使用したデータ項目の定義
マルチスレッド対応	UG 第23章 マルチスレッド
Unicodeサポート	UG 第24章 Unicode
COM連携機能	UG 第25章 COM機能

注)ADDR関数、CURRENT-DATE関数およびLENG関数はサポート

D.2 JEFオプションの機能上の特徴と制約

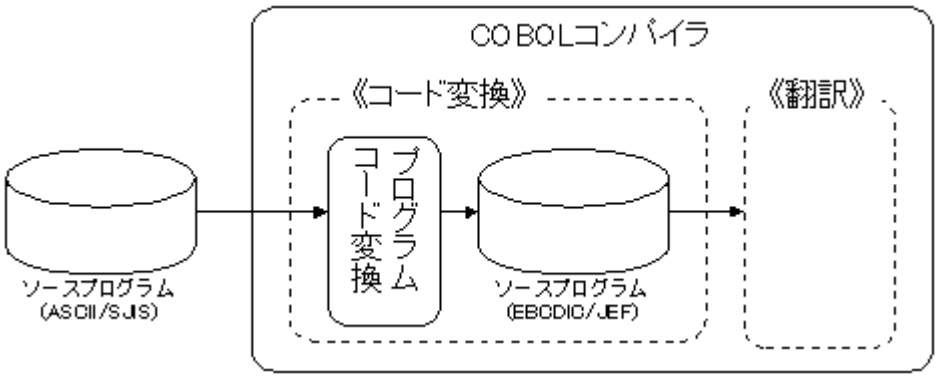
ここでは、JEFオプションを使用してプログラムを開発する場合に利用できるJEFオプションに特徴的な機能と、JEFオプションを使用する場合の制約を説明します。

D.2.1 プログラミング全般

プログラムの記述に使用できる文字

JEFオプションでは次のように一度ソースをEBCDIC/JEFに変換してから、翻訳処理を行います。

図D-4 JEFオプションでのCOBOLソース翻訳の概要



この変換処理に関する問題から、JEFオプションを使用しての開発時にプログラムの記述に使用できない文字があります。

英小文字と記号の一部

JEFオプションを使用して、プログラムを開発する場合、英小文字は使用できません。
ひとくちにEBCDICコード系と言いますが、EBCDICコード系には多くの変種があり、OSIV系システムに使用可能なものでも、次の3種類があります。

表D-4 OSIV系システムで使用可能なEBCDICコード系

名称	英小文字	半角カナ文字	日本語特有の記号
EBCDIC(カナ)	含まない	含む	含む
EBCDIC(ASCII)	含む	含まない	含まない
EBCDIC(英小文字)	含む	含まない	含む

JEFオプションを使用して開発したプログラムで使用できるのは、上記のうちEBCDIC(カナ)のみです。このため、EBCDIC(カナ)に対応するコード値を持たない文字をソースプログラムまたは登録集原文に使用することはできません。このため、英小文字は使用できません。
また、一部の記号を表現する文字も、別の文字に変換されてしまうものがあります。以下にその一覧を示します。

表D-5 ASCII→EBCDICコード系変換で異なる文字に変換される文字

ASCII		EBCDIC	
表示	コード値16進表現	表示	コード値16進表現
!	0x21	|	0x4F
[	0x5B	£	0x4A
]	0x5D	!	0x5A
^	0x5E	^	0x5F
|	0x7C		0x6A

一部の日本語文字

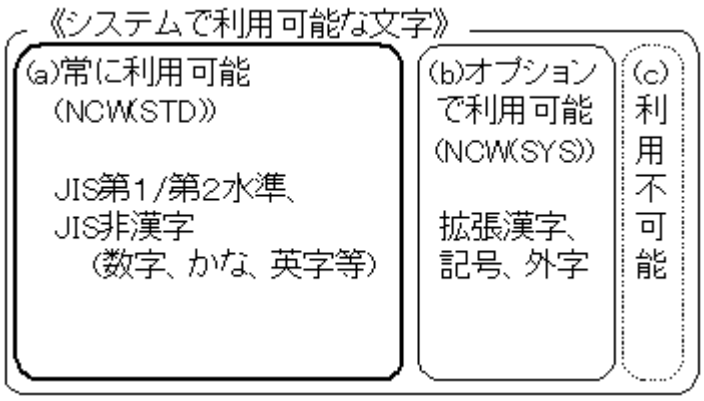
CharsetMGR、ADJUST などの文字コード管理用製品がインストールされている場合、一部の日本語文字に診断メッセージJMN1008I-Sが出力される場合があります。

```
JMN1008I-S  日本語利用者語の中に日本語文字として使用できない文字があります.
            '@1@'を日本語利用者語とみなします.
```

これは次に示す２つの点についての設定がかみ合っていないため発生します。

- COBOLが日本語利用者語として使用可能な文字の範囲
  利用可能な日本語文字のすべてがCOBOLの日本語利用者語に使用できるわけではなく、次のような３つの集合に分けられます。

図D-5 COBOLで利用可能な文字の範囲



NetCOBOLは翻訳時に明に指定しないかぎり、NCW(STD)が指定されているものとして動作するため、上の図の(b)と(c)に含まれる文字はCOBOLソースおよび登録集ファイル内で使用

できません。もし、使用した場合、その文字に対して診断メッセージJMN1008I-Sが出力されることになります。

- コード変換プログラムの設定
  JEFオプションでは、“図:JEFオプションでのCOBOLソース翻訳の概要”で示すとおり、ソース全体の文字コード変換を行います。この際に行われる、SJIS->JEFの変換方法は、使用可能文字数が「SJIS < JEF」(JEFの方でより多くの文字が使用できる)であることから１通りではありません。
  NetCOBOL JEFオプションは、のみでは、その変換方法を変更することができませんが、CharsetMGR、ADJUSTなどの文字コード管理用製品がインストールされている場合、変換方法を変更することが可能です(SJIS代表コード系の設定)。変換方法の選択によっては、JIS第１/第２水準に含まれる文字が(b)の領域に含まれる文字に変換されてしまいます(CharsetMGRではデフォルト)。

したがって、上記のどちらかの設定を変更することによって問題を解決することが可能です。どちらの変更を設定することを選択するかは文字データをどう扱いたいかによって異なります。
次のような要件がある場合は、CharsetMGRやADJUSTの変換方法を変更することで対応してください。

- 帳票印刷等のため文字の字形を重視している。
- OSIV系システムとの煩雑なデータ交換が必要である。
- 複数のシステム(OS)で運用するプログラムの開発をしている。

上記の要件がない場合、ソースプログラムを翻訳する際に翻訳オプションにNCW(SYS)を追加してください。

文字の大小順序

ASCII/シフトJIS とEBCDIC/JEFでは、コード系の違いにより、文字の大小順序が異なります。

- 英字、数字、カナの大小順序が逆転します。
- 外字と外字以外の日本語の大小順序が逆転します。

このため、文字の大小比較はJEFオプションを使用する場合、NetCOBOL ではなく、むしろOSIV系システムのCOBOL85と同じにふるまいます。

```
01  英数字     PIC  X  VALUE "A".
01  数字       PIC  9  VALUE 1.
          :
     IF     英数字 < 数字  THEN
            DISPLAY  "EBCDIC/JEF"
     ELSE
            DISPLAY  "ASCII/SJIS"
     END-IF.
```

OSVI系システムのCOBOL85やJEFオプションではTHEN側のDISPLAY 文が動作し、NetCOBOL ではELSE側のDISPLAY 文が動作します。

空白コードの違い

日本語文字の空白コード(２バイト空白)と、英数字の空白コード(１バイト空白)がEBCDIC/JEFのときは同じ値(X'4040'とX'40')ですが、ASCII/シフトJIS のときは、異なる値(X'8140'とX'20')になります。

```
01  集団.
    02  FILLER     PIC N(2)  VALUE  NC"日本".
    02  FILLER     PIC X(2)  VALUE  SPACE.
```

```
01  日本語        PIC N(3)  VALUE  NC"日本".

    IF  集団  =   日本語  THEN
        DISPLAY   "EBCDIC/JEF"
    ELSE
        DISPLAY   "ASCII/SJIS"
    END-IF.
```

OSVI系システムのCOBOL85やJEFオプションではTHEN側のDISPLAY 文が動作し、NetCOBOL ではELSE側のDISPLAY 文が動作します。

プログラム中の16進定数

プログラム中の16進定数はEBCDIC/JEFの値(コード値)を指定します。

```
01  英数字    PIC  X  VALUE X"C1".

    IF  英数字  =  "A"  THEN
        DISPLAY    "EBCDIC/JEF"
    ELSE
        DISPLAY    "ASCII/SJIS"
    END-IF.
```

OSVI系システムのCOBOL85やJEFオプションではTHEN側のDISPLAY 文が動作し、NetCOBOL ではELSE側のDISPLAY 文が動作します。

型の異なる項目の再定義

外部10進項目や日本語項目を異なる型のデータで再定義(REDEFINES) しているプログラムは、コード系の違いに注意する必要があります。

```
01  英数字    PIC  X(3)  VALUE  "12L".
01  数字      REDEFINES  英数字  PIC  S9(3).

    IF  数字  =  -123  THEN
        DISPLAY  "EBCDIC/JEF"
    ELSE
        DISPLAY  "ASCII/SJIS"
    END-IF.
```

OSVI系システムのCOBOL85やJEFオプションではTHEN側のDISPLAY 文が動作し、NetCOBOL ではELSE側のDISPLAY 文が動作します。

D.2.2 入出力

ファイル識別名定数。

外字は指定できません。

ファイル内のデータ

ファイル内のデータはEBCDIC/JEFコード系で格納されます。
なお、ファイルユーティリティを使用することで、ファイルの内容を参照することができます。
詳細については、ファイルユーティリティのヘルプを参照してください。

行順ファイル

使用できません。

ASCII/シフトJIS コード系で作成したファイル

ASCII/シフトJIS コード系で作成したファイルをEBCDIC/JEFコード系のモードで読むこと、およびEBCDIC/JEFコード系で作成したファイルをASCII/シフトJIS コード系のモードで読むことはできません。

CODE-SET句を記述したプログラムについて

NetCOBOLでは、CODE-SET句をサポートしていません。このため、CODE-SET句の動作を前提としたプログラムについての動作を保証していません。CODE-SET句を使用していた場合、CODE-SET句を注釈として、プログラムを再度翻訳し、動作を確認してください。

BtrieveファイルおよびXLデータパイプ

使用できません。

D.2.3 印刷ファイル

印字可能文字

英小文字を除くすべてのEBCDIC/JEFコード系の文字を印刷することができます。

フォント

表示/印刷のフォントは、JEF フォントが使用されるため、フォントの選択はできません。

その他

- 印刷データ中にバイナリコードが含まれないように注意してください。
- 印刷ファイルのレコード中に直接、デバイス制御コードを含んでいるプログラムの流通は保証されません。

D.2.4 小入出力

使用可能文字

DISPLAY/ACCEPT文では、EBCDIC/JEFコード系の文字を入力したり、表示したりすることができます。ただし、JEF 拡張文字は入力できません。

ACCEPT文に英小文字が入力された場合、対応する英大文字に変換されます。

フォントサイズ

フォントサイズを指定しない場合、画面上へのDISPLAY/ACCEPTは16ポイント文字となります。

UPON CONSOLE指定

DISPLAY文による出力対象が日本語項目、日本語編集項目または日本語定数である場合、正しい文字が表示されない場合があります。

なお、COBOL G のDISPLAY 文におけるUPON CONSOLE指定の機能は、NetCOBOL ではUPON SYSOUT に相当します。したがって、ソースを移行する際には、CONSOLEをSYSOUTに書き換える必要があります。

入出力先変更

翻訳オプションSSIN/SSOUTを指定し、入出力先をファイルとして、入出力を行うことはできません。

D.2.5 ソート・マージ

利用条件

ソート･マージおよびファイルユーティリティで〔整列〕コマンドを使用する場合､PowerSORT V1.0 L30以降が必要となります。

D.2.6 表示ファイル

表示ファイル単体テスト機能

使用できません。

D.2.7 言語間結合

Ｃ言語と言語間結合する上での注意

外部名は、ASCII/シフトJIS コード系のため使用できます。ただし、パラメタとして引き渡されるデータの内容はEBCDIC/JEFコード系なので、必要ならば、Ｃ言語側でコード系を変換してください。

D.2.8 通信機能

簡易アプリ間通信機能

実行環境情報の指定

JEF オプションのCOBOL コンパイラで生成されたオブジェクトで､簡易アプリ間通信機能を使用する場合は、実行環境情報に以下の情報を必ず指定してください。

```
@CBR_CI_CODETYPE=EBCDIC-KANA
```

実行環境情報に"@CBR_CI_CODETYPE"が指定されていないか、"EBCDIC-KANA"が省略されている場合、オブジェクトのコード系はASCII コードとみなされ、実行時エラーが発生します。

エラーコード

JEF オプションでは、以下のエラーコードが追加されます。

表D-6 JEFオプション使用時に追加される簡易アプリ間通信機能のエラーコード

エラーコード	詳細コード	意味	処置	対象関数			
				O	C	R	W
9	XXXX	EBCDIC->ASCII変換エラー詳細コードXXXXは、JEF拡張漢字サポートのコード変換関数のエラーコードを示しています。	コードの詳細については、JEF拡張漢字サポートのエラーコードを参照してください。	○	○	○	－

13	XXXX	システム間通信異常が発生しました。詳細コードXXXXは、Windows Sockets関数のエラーコードを示しています。(注)	コンパイラの生成したオブジェクトのコード系と"@CBR_CI_CODETYPE" の指定が異なっていないか確認してください。コードの詳細については、Sockete関数 のエラーコードを参照して下さい。	○	－	－	－
15	3845	JEF 変換モジュールのロードに失敗しました。	JEF 拡張漢字サポートがインストールされているか確認してください。	○	○	○	－
	3846	@CBR_CI_CODETYPEの指定に誤りがあります。	"EBCDIC-KANA" が指定されているか確認してください。	○	－	－	－

対象関数の記号の意味

O:COBCI_OPEN関数　C:COBCI_CLOSE関数

R:COBCI_READ関数　W:COBCI_WRITE関数

○:通知される

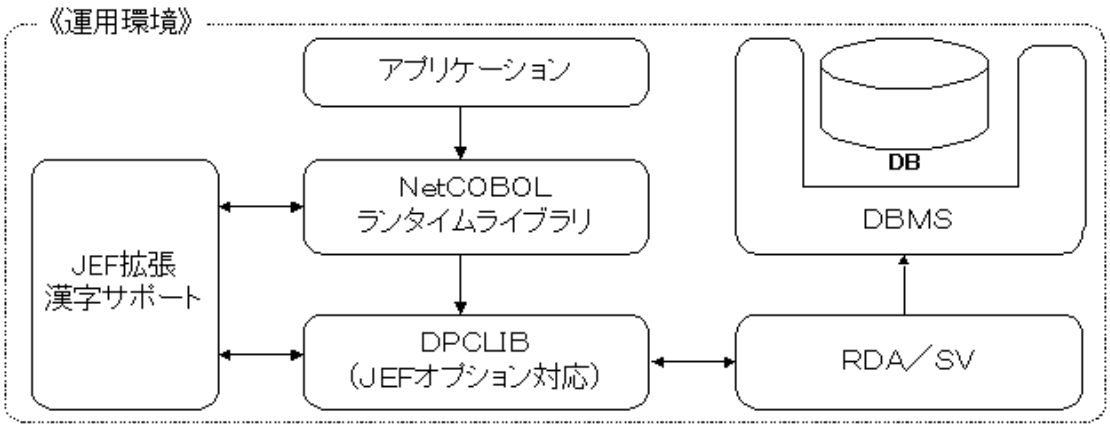
注)詳細コードについては、"NetCOBOL 使用手引書" の "20.6 簡易アプリ間通信のサブルーチンのエラーコード" を参照して下さい。

D.2.9 データベースアクセス機能(SQL)

運用環境の概要

データベースに関する運用環境の概要について、以下に示します。

図D-6 JEFオプション使用時のデータベース機能の運用環境概要



実行時の変更点および注意点

- コネクション情報に指定するユーザID/パスワードは、英数字項目です。この項目に対して、英小文字の情報を設定することはできません。
- ODBC経由のリモートデータベースアクセス(SQL) 機能の場合には、データベースアクセスのためにODBC情報ファイルを使用していました。
  JEF オプションでは、DPCLIBのRDA 機能を経由してアクセスします。また、データベースアクセスのためにDPCLIB情報ファイルを使用します。

DPCLIB情報ファイルは、ODBC情報ファイルと同じ形式です。ただし、次の情報のみが、DPCLIB情報ファイルに指定できます。

- データソース名
- ユーザID
- パスワード
- サーバ名

また、次の情報は、ODBC環境の場合のデータソースが、DPCLIBのRDA 環境のサーバ名となるため、意味が変更となります。

- データソース名 => サーバ名

- 更新系カーソルをOPEN文によりオープンした後、FETCH文によるデータ処理を行わないままUPDATE文(位置づけ)を使用した場合、サーバ側でカーソルを閉じる処理が行われます。このため、このような場合に、CLOSE文によりカーソルをクローズしようとするとエラーが発生します。

実行環境情報の指定方法

ODBC経由のリモートデータベースアクセス(SQL) 機能の場合には、次の実行環境情報にODBC情報ファイルの名前を指定していました。

```
@ODBC_Inf=ODBC情報ファイル名
```

JEF オプションのDPCLIBのRDA 機能経由では、次の実行環境情報にDPCLIB情報ファイルの名前を指定します。

```
@CBR_DPCLIB_Inf=DPCLIB情報ファイル名
```

制限事項

以下は使用できません。

- ストアドプロシージャの呼び出し(CALL文)
- 複数列指定ホスト変数、複数行指定ホスト変数および表指定ホスト変数
- SQLERRD
- FOR句

D.2.10 プログラムの翻訳

文字コードの切換え方法

文字コードの切換えは、プロジェクトマネージャのメニューあるいはプロジェクトのプロパティによって行います。この設定は以下の操作を行う場合に意味を持ちます。

- プロジェクト管理機能を使用した翻訳
- WINCOBウィンドウを使用した翻訳
- 対話型デバッガによるデバッグ

選択した文字コードの情報は、プロジェクトマネージャおよびプロジェクトの情報に残されます。これは文字コード切り替えを再度行うか、異なる文字コードの設定を持つプロジェクトファイルを開くまで有効になります。

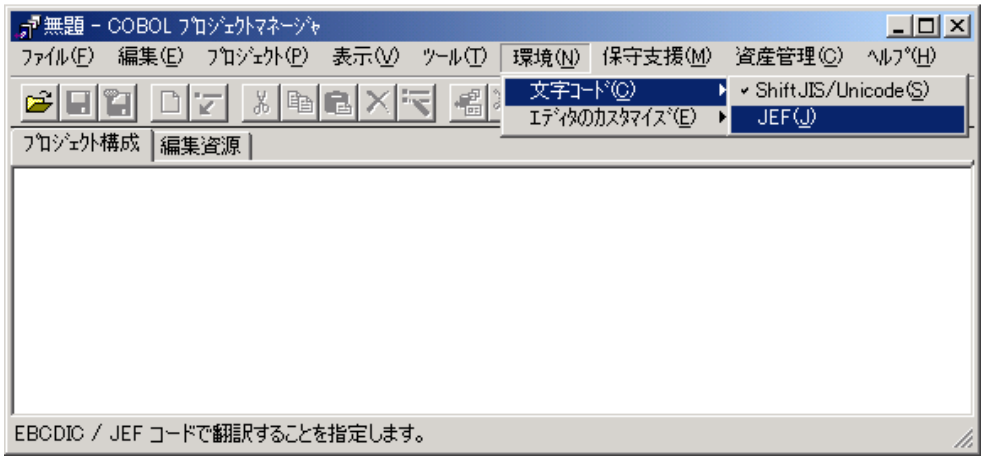

注意

異なる文字コードの設定を持つプロジェクトを同時に複数開くことはできません。最後に開いたプロジェクトの文字コードの設定が有効になります。複数のプロジェクトを開いた状態で、文字コードの設定を変更しないでください。
誤って、上記のような操作を行った場合、プロジェクトマネージャと各プロジェクトの文字コードの設定に矛盾が生じ、そのプロジェクトの管理するプログラムの翻訳・デバッグが正しく行えなくなります。そのような事態が生じた場合、一度すべてのプロジェクトを閉じた上で、各プロジェクトを１つずつ開くことで元に戻すことができます。

プロジェクトマネージャの［環境］メニューによる設定

1. プロジェクトマネージャの［環境］－［文字コード］メニューを選択します。
2. サブメニューが表示されるので、文字コードを選択します。

図D-7　プロジェクトマネージャの［環境］メニューによるコード系の設定

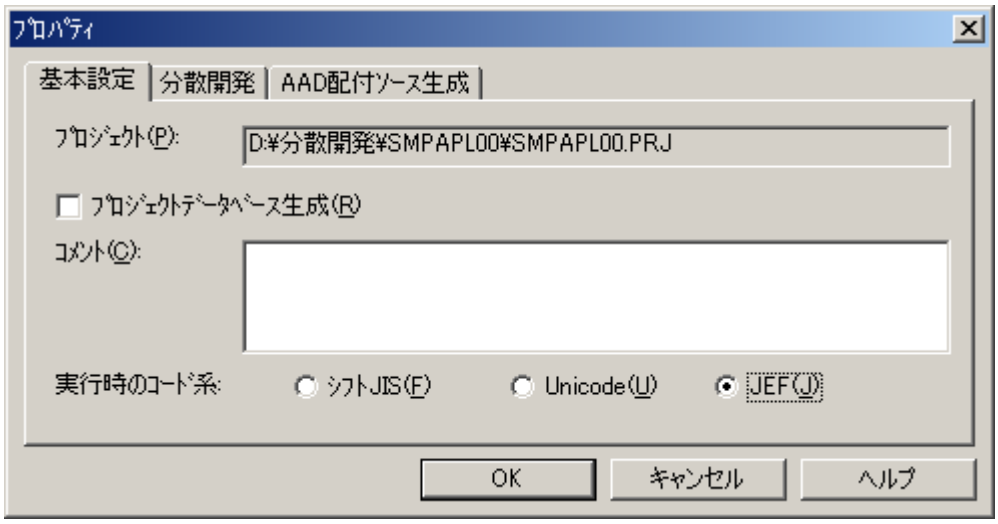


3. 文字コードの変更を確認するメッセージボックスが出力されます。“Yes”ボタンをクリックすると文字コードが変更されます。

プロジェクトのプロパティによる設定

1. COBOLプロジェクトマネージャの〔プロジェクト構成〕ページのツリービューで、プロジェクトファイルを選択して、〔ファイル〕メニューから“プロパティ”を選択するとプロジェクトの“プロパティ”ダイアログボックスが表示されます。
2. 〔基本設定〕のページで実行時コード系を“JEF”を選択します。

図D-8　プロジェクトのプロパティによるコード系の設定



3. 文字コードの変更を確認するメッセージボックスが出力されます。“Yes”ボタンをクリックすると文字コードが変更されます。

翻訳時に使用するファイルのコード系

NetCOBOL コンパイラが使用するファイルのコード系を下表に示します。

表D-7 翻訳時に使用する資源の使用可能なコード系

NetCOBOL コンパイラが使用するファイル	コード系	
	ASCII/SJIS	EBCDIC/JEF
ソースファイル	○	－
登録集ファイル	○	－
画面帳票定義体ファイル	○	○
ファイル定義体ファイル	○	○
オプションファイル(DEFAULT.CBI)	○	－
翻訳リストファイル	○	－
翻訳時メッセージ	○	－

翻訳オプション

以下に示す翻訳オプションを指定すると、WまたはE レベルのメッセージが出力され、指定が無効となります。

表D-8 JEFオプションでは無効となる翻訳オプションの一覧

翻訳オプション名	処置
ALPHAL	NOALPHALが指定されたものとみなされます。
BINARY	BINARY(WORD,MLBON)が指定されたものとみなされます。
NSPCOMP	NSPCOMP(NSP)が指定されたものとみなされます。
RCS	JEFオプションでは意味を持ちません。
REP/REPIN	JEFオプションでは意味を持ちません。
SHREXT	NOSHREXTが指定されたものとみなされます。
SQLGRP	NOSQLGRPが指定されたものとみなされます。
THREAD	THREAD(SINGLE)が指定されたものとみなされます。

以下に示す翻訳オプションはデフォルト値がNetCOBOLと異なります。

表D-9 JEFオプションではデフォルト値が異なる翻訳オプションの一覧

翻訳オプション名	処置
RSV	RSV(V125)が指定されたものと見なされます。 ALL/V30/V40/V61は指定できません。

D.2.11 プログラムのリンク

リンクでは、特にJEF オプションを意識する必要はありません。
外から指定するファイルのファイル名は、すべてASCII/シフトJIS コード系となります。
また、プログラム名などの外部名はASCII/シフトJIS コード系になります。

制限事項

ASCII/シフトJISで作成したNetCOBOLオブジェクトと結合することはできません。

D.2.12 プログラムの実行

実行時の環境設定について

JEF 拡張漢字サポート V4.1 L10 以降が、インストールされている必要があります。

GS形式の実行時パラメタについて

GS形式の実行時パラメタはコード系変換されます。パラメタの値として英小文字が指定された場合、英大文字に自動的に変換されます。
アプリケーションに外から(例えばACCEPT文などで)データを渡す場合の注意事項については、以降の節で説明します。

D.2.13 デバッグ機能(TRACE、CHECK、COUNT)

(参照)→“NetCOBOL 使用手引書”の“第18章 プログラムのデバッグ”

出力ファイルのコード系

TRACE 情報ファイルおよびCOUNT 情報ファイルは、ASCII/シフトJIS コード系で格納されます。
ファイルの内容は、エディタを使って参照できます。

D.2.14 対話型デバッガ

動作コード系

対話型デバッガを使用する前に、プロジェクトマネージャの［環境］メニューの［文字コード］が、デバッグ対象であるプログラムを翻訳した時の文字コードと同じであることを確認してください(“D.2.10 プログラムの翻訳”を参照)。
「開始プログラムに到達する前に実行が終了しました。」というメッセージが表示された場合、デバッガ使用時の文字コードが対象プログラムの翻訳時の文字コードと異なっている可能性があります。

ソースファイルウィンドウ内の表示

デバッグ中のCOBOLソースの記述に日本語が含まれていると、日本語を含む言語要素より後ろで、中断位置や中断点を示すマークがソース表示とずれて見えます。

外字の扱いについて

操作履歴ファイルはASCII/シフトJIS コード系で格納されます。結果をエディタで参照できますが、外字は“□”となります。

デバッグ操作時の文字の入力

対話型デバッガの［データの表示/変更］ダイアログまたは［監視］ウィンドウでデータの値を変更する場合に、JEFシフトコードは自動的に挿入されません。必要な箇所にJEFシフトコードを挿入するようにしてください。

また、デバッグ時の入力において、JEF拡張文字を指定することはできません。JEF拡張文字を入力したい場合には、対応するJEFコードを16進文字定数または日本語16進文字定数で指定してください。

リモートデバック

JEFオプション製品を使用して作成したCOBOLプログラムをリモートデバックすることはできません。

D.2.15 実行時メッセージ

追加メッセージ

JEFオプションでは、以下の実行時メッセージが追加になります。

JMP00801-U

JEF拡張漢字サポートが使用できません. '$1'. CODE=$2.

【システムの処置】

プログラムを異常終了させます。

【プログラマの処置】

拡張漢字サポートのDLL名($1) を参考に環境を確認してください。または、$2に設定されたエラーコードを参考にエラーの原因を取り除き、再度実行してください。

(参照:Visual C++のオンラインヘルプ)

JMP00811-U

実行環境内に異なるコード系のプログラムが存在します. '$1'.

【システムの処置】

プログラムを異常終了させます。

【プログラマの処置】

プログラム($1)の動作コード系を確認し、実行環境内で動作するプログラムのコード系を統一してください。

JMP00821-U

文字コードの変換に失敗しました. STM='$1'. FUNC='$2'. CODE='$3'.

【システムの処置】

プログラムを異常終了させます。

【プログラマの処置】

表示された文字列を参考にエラーの原因を取り除き、再度実行してください。

なお、$1～$3には以下の情報が設定されます。

$1:　エラーが発生したCOBOLの文を示す文字列

$2: JEF 拡張漢字サポートのコード変換関数の種別
$3: JEF 拡張漢字サポートからのエラーコード

表D-10 JMP0082I-Uの$1の内容

$1	エラーの内容	プログラマの処置
CALL	動的プログラム構造でのCALL文またはCANCEL文に指定されたプログラム名のコード変換でエラーが発生しました。	JEF 拡張漢字サポートのユーザーズマニュアルの説明から原因を調査し対処してください。
EXPRM	GS形式の実行時パラメタのコード変換でエラーが発生しました。	
STOP	STOP文に指定された定数のコード変換でエラーが発生しました。	

JMP0083I-U

NetCOBOL JEF オプションがインストールされていません.

【システムの処置】

プログラムを異常終了させます。

【プログラマの処置】

NetCOBOL JEF オプションまたは NetCOBOL JEF オプション 運用パッケージを
インストールしてください。

既存メッセージの変更

JMP0310I-I/U

JMP0320I-I/U

“NetCOBOL 使用手引書”の“表F.6 JMP0310I-I/Uの$3の内容”および“表F.8 JMP0320I-I/Uの$3の内容”に以下の項目が追加されます。

表D-11 追加される$3の内容

$3(文字列)	エラーの内容	プログラマの処置
CNVER=xxxx	JEF 拡張漢字サポートからのエラーコードを示します。	JEF 拡張漢字サポートのユーザーズマニュアルの説明から原因を調査し対処してください。

JMP0613I-I/U

【プログラマの処置】に以下の項目が追加されます。

$2にエラーコードが出力されていない場合、PowerSORT のバージョンレベルを確認してください。
(PowerSORT V1.0L30 以降)

D.2.16 サンプルプログラム

(参照)→ “NetCOBOL 例題プログラム”

NetCOBOL に添付されているサンプルの多くは、JEF オプションでは使用できない機能を使用しています。
使用可能なサンプルを示します。

- そのまま、使用可能
  - 例題8
  - 例題9
- 修正によって、使用可能
  - 例題11
  - 例題29

例題11の修正方法

ソースプログラムの修正

以下のようにホスト変数の定義およびFETCH文に対して、プログラムの変更が必要となります。

図D-9 例題11修正前

```
01 在庫表.
  02 製品番号    PIC S9(4) COMP-5.
  02 製品名      PIC X(20).
  02 在庫数量    PIC S9(9) COMP-5.
  02 倉庫番号    PIC S9(4) COMP-5.
      :
    EXEC SQL FETCH CUR1 INTO :在庫表  END-EXEC
      :
    EXEC SQL FETCH CUR1 INTO :在庫表  END-EXEC
```

図D-10 例題11修正後

```
01 製品番号      PIC S9(4) COMP-5.
01 製品名        PIC X(20).
01 在庫数量      PIC S9(9) COMP-5.
01 倉庫番号      PIC S9(4) COMP-5.
      :
    EXEC SQL
      FETCH CUR1
      INTO :製品番号,:製品名,:在庫数量,:倉庫番号
    END-EXEC.
      :
    EXEC SQL
      FETCH CUR1
      INTO :製品番号,:製品名,:在庫数量,:倉庫番号
    END-EXEC.
```

実行時の環境設定

実行時、以下の設定が必要となります。

```
@CBR_DPCLIB_Inf=DPCLIB情報ファイル名
```

詳細については、本書の“D.2.9 データベースアクセス機能(SQL)”を参照してください。

例題29の修正方法

実行時の環境設定

実行時、以下の設定が必要となります。

```
@CBR_CI_CODETYPE=EBCDIC-KANA
```

詳細については、本書の“D.2.9 データベースアクセス機能(SQL)”を参照してください。

D.2.17 イベントログ出力サブルーチン

制限事項

RETURNING句を記述することはできません。サブルーチンからの復帰コードは特殊レジスタPROGRAM-STATUSで参照できます。

付録E GETSSCH診断メッセージ一覧

サブスキーマ取り出しツールGETSSCHが出力する診断メッセージについて説明します。

E.1 診断メッセージの形式

診断メッセージは次に示す形式で表示されます。

```
ＪＭＮＳｎｎｎＩ－ｓ　メッセージ本文
```

- ｎｎｎはメッセージ番号を示します。
- ｓはエラーの重大度を示す一文字の英字です。エラーの重大度の意味を示します。

重大度コード	意味
I	情報
W	軽度のエラー
E	中程度のエラー
S	重度のエラー
U	致命的なエラー

- “メッセージ本文”は、診断メッセージそのもので、全て英語で出力されます。

E.2 診断メッセージの一覧

JMNS001I-U

I/O ERROR @1@

〔意味〕

＠１＠のファイルに対するアクセス中に入出力エラーが発生しました。

〔システムの処理〕

処理を中止します。

JMNS002I-U

SYSIN FILE MUST BE FIXED LENGTH FORMAT.

〔意味〕

ＳＹＳＩＮ(サブスキーマ名ファイル)のレコード形式は、固定長形式でなければなりません。

〔システムの処理〕

処理を中止します。

JMNS004I-U

@1@ FILE CANNOT BE OPENED.

〔意味〕

＠１＠ファイルがオープンできません。

〔システムの処理〕
処理を中止します。

JMNS0051-U

MAIN STORAGE IS INSUFFICIENT.
〔意味〕
実行に必要な主記憶領域が不足しています。
〔システムの処理〕
処理を中止します。

JMNS0061-U

SSCHLIB FILE MUST BE FIXED LENGTH FORMAT.
〔意味〕
ＳＳＣＨＬＩＢ（サブスキーマ登録集ファイル）のレコード形式は固定長でなければなりません。
〔システムの処理〕
処理を中止します。

JMNS0071-U

SYSIN FILE IS EMPTY.
〔意味〕
ＳＹＳＩＮ（サブスキーマ名ファイル）が空です。
〔システムの処理〕
処理を中止します。

JMNS0081-E

OPTION '@1@' IS INVALID.
〔意味〕
＠１＠は、オプション名として誤りです。
〔システムの処理〕
処理を中止します。

JMNS0091-U

RECORD LENGTH OF @1@ FILE MUST BE 80 BYTES.
〔意味〕
＠１＠ファイルのレコード長は８０バイトでなければなりません。
〔システムの処理〕
処理を中止します。

JMNS0111-U

PROCESS CAN NOT BE CONTINUED.DETAIL-CODE=@1@.
〔意味〕
処理が続行不可能となりました。詳細コードは＠１＠です。
〔システムの処理〕

処理を中止します。

JMNS0121-U

OPEN OR CLOSE ERROR OCCURRED IN AIMLIB.

〔意味〕

ＡＩＭＬＩＢ（ＡＩＭディレクトリデータセット）でオープン／クローズエラーが発生しました。

〔システムの処理〕

処理を中止します。

JMNS0131-S

SUBSCHEMA SPECIFIED IS MISSING IN AIMLIB.

〔意味〕

ＡＩＭＬＩＢ（ＡＩＭディレクトリデータセット）に指定されたサブスキーマがありません。

〔システムの処理〕

処理を中止します。

JMNS0151-S

'@1@'.

〔意味〕

’＠１＠’。ＡＩＭＬＩＢ（ＡＩＭディレクトリデータセット）アクセス中にエラーが発生しました。

〔システムの処理〕

処理を中止します。

〔パラメタの意味〕

＠１＠：ＡＩＭから返却されるメッセージ。

詳細については”ＡＩＭシステムメッセージとシステムコード”を参照してください。

JMNS0161-S

AIMLIB IS NOT ALLOCATED.

〔意味〕

ＡＩＭＬＩＢ（ＡＩＭディレクトリデータセット）が割り当てられていません。

〔システムの処理〕

処理を中止します。

JMNS0171-S

LANGUAGE TYPE OF SUBSCHEMA IS NOT COBOL.

〔意味〕

ＳＵＢＳＣＨＥＭＡの言語形式がＣＯＢＯＬではありません。

〔システムの処理〕

処理を中止します。

JMNS0181-S

FOR UWA(N) OPTION,NATIONAL UWA MUST BE DEFINED.

〔意味〕
翻訳オプションＵＷＡ（Ｎ）の場合、日本語のＵＷＡが定義されていなければなりません。
〔システムの処理〕
処理を中止します。

JMNS0191-S

SUBSCHEMA SPECIFIED IS NOT SUBSCHEMA NAME.
〔意味〕
指定されたサブスキーマ名がサブスキーマ名ではありません。
〔システムの処理〕
処理を中止します。

JMNS0201-S

ERROR OCCURRED DURING SUBSCHEMA READING. '@1@'.
〔意味〕
サブスキーマの読み込み中にエラーが発生しました。
〔システムの処理〕
処理を中止します。

JMNS0221-S

@1@ FILE HAS INVALID ORGANIZATION.
〔意味〕
＠１＠ファイルの編成は不適当です。
〔システムの処理〕
処理を中止します。

付録F 文字コード系

文字コードは、計算機で文字を表現する仕組みです。COBOLでアプリケーションを開発する場合、特別の場合を除いてはこれを意識する必要はありません。ソースを記述するための文字の記述から、データとしての文字の代入や比較までをCOBOLが一貫した規則で扱うためです。
しかし、次のようなアプリケーション開発を行う場合は例外です。

- 他のシステムで動作するアプリケーションの開発(分散開発)
- 他のシステムで動作していたアプリケーションの別システムへの移植
- 複数のシステムで動作可能なアプリケーションの開発

このような場合、システム間の文字コードの違いから問題が生じる可能性があります。そのような問題の理解と解決には文字コードについての理解が必要となります。そのためここでは、次のような内容について説明します。

- 文字コードの概説
- 各オペレーティングシステムのCOBOL製品のサポートするコード系
- 文字コードの違いのCOBOLプログラミングへの影響

文字コードという言葉は、個々の文字に割り当てられた特定の値を意味することもありますし、その割り当て規則の体系を指すこともあります。ここでは主に後者の意味で“文字コード”という言葉を使用し、前者の意味では“コード”という言葉を使用します。

F.1 文字コードの概要

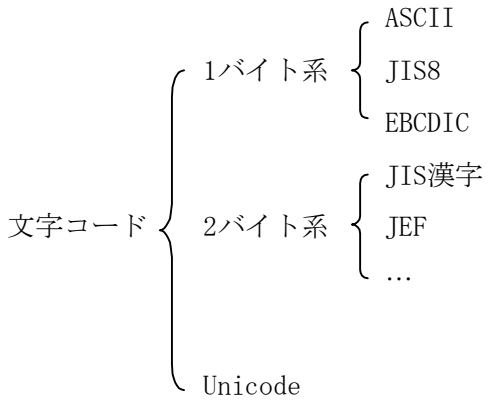
文字コードにはいくつもの方式があり、かつ、分類の方法にも幾つかの種類があります。そのため、実際には単純にあるコード系と別のコード系を比較することはできません。しかし、ここでは単純化のために次の分類に分けてコード系を説明します。

- 文字を表現するバイト数の違い
- 文字種の混在方式

F.1.1 文字を表現するバイト数の違いによるコード系の分類

文字を表現するバイト数の観点から文字コード系は次のように分類されます。

図F-1 内部表現のバイト数による文字コード系の分類



1バイト系は英数字記号等を表現するために用意され、その後の拡張にそれ以外の文字も含むようになったものです。

- ASCII
  アメリカの国家標準化組織であるANSIによって制定されたコード系で、大小のアルファベット文字、数字、制御文字および少数の記号を含みます。
- JIS8
  日本の標準化組織であるJISによって制定されたコードで、ASCIIコード系のほとんどを受け継ぎつつ、次の点で変更を加えたものです。
  - バックスラッシュの代わりに“¥”を含む
  - カナ文字を追加している
- EBCDIC
  元々はIBM社の考案したコード系で、英大文字、数字、制御文字および少数の記号を含みます。文字の割り当ての一部が任意の割り当てを許すようになっており、使用する国、地域、用途などからさまざまな変種が存在します。日本国内では主に次のようなものが使われます。
  - EBCDIC(カナ)…カナ文字と日本固有の記号を追加したもの
  - EBCDIC(ASCII)…英小文字追加したもの
  - EBCDIC(英小文字)…英小文字と日本固有の記号を追加したもの

2バイト系は日本語を表現するために十分な文字を表現するために用意されたものです。漢字以外にカタカナ、ひらがな、英字、数字、その他の記号を多数含みますが、総称して漢字コードと呼ばれます。

- JIS漢字
  JIS X 0208で規定される漢字コードです。1978年に制定され、その後、2度の改定が行われていますが、1983年の改定は400近い文字についての変更が行われたため、この改定より前を78JIS、改定以降を83JISと言って区別する場合があります。
- JEF
  78JISを元にEBCDICとの混在使用に適するように作られた富士通固有の漢字コードで、次の特徴を持ちます。
  - JIS漢字に含まれない多くの漢字を含む
  - 78JISに含まれる漢字についてはすべての文字が同じ順序で含まれる
  - 83JISで追加された文字も含まれる
- その他
  富士通のJEFに相当するものとして、IBMをはじめとするベンダ固有の漢字コードが存在します。ここでは、その存在を示すのみで詳しく述べません。

日本では、これらの文字コード系が単独で用いられる場合は少なく、一般には1バイト系のコードと2バイト系のコードを混在して使用するための混在コード系(“F.1.2 文字種の混在方式による分類”で説明)が用いられます。
それに対して、Unicodeははじめから世界中の文字を1つのコード系で網羅することを目指して設計されたもので、ここまで説明してきた各コード系とまったく性質の異なるものです。Unicodeについては“F.1.3 Unicode”で別途説明しますので、そちらを参照してください。

F.1.2 文字種の混在方式による分類

1バイトで表現される英数字等の文字と2 バイトで表現される文字を混在して使用する方法は文字種の判定の方法と使用可能な文字種によってさまざまな変種が存在します。
ここでは、それを大きく3つに分けて説明します。

- SJIS(シフトJIS)
  PCで広く利用されているコード系です。英数字・カナ文字(SJIS8)、日本語文字(JIS漢字)を混在させる方法で、次のような特徴を持ちます。
  - 文字種の切り換えにシフトコードを使用しない
  - 1バイトで表現される英数字・カナについてはJIS8コード系と同じ値を持つ
  - JIS漢字は4つの領域に分散するが、演算により規則的に対応し、文字のコード値の

大小関係もJIS漢字と一致する

なお、シフトJISにはJIS漢字に含まれない文字を追加するための領域があり、その領域に追加した文字の違いにより、いくつかの変種があります。例えば富士通により78JIS固有の文字やOASYS記号等を追加されたもの(R90)や、またマイクロソフト社により別の文字が追加されているもの(MS-SJIS)があります。Windowsシステムでは通常はMS-SJISが標準となっています。

- EUC-JIS

  UNIX系で広く利用されているコード系です。英数字(ASCII)に、ISO 2022の拡張方法に従って、カナ(JISカナ)、日本語文字(JIS漢字)を混在させる方法で、次のような特徴を持ちます。

  — 文字は1～3バイトで表現されるが、1バイト目で文字種の判定が可能
  — 1バイトで表現される英数字についてはASCIIコード系と同じ値を持つ
  — JIS漢字のすべてが1つの領域として含まれる
  — JISカナは1バイトのシフトコードを付けて表す

  なお、ISO 2022の拡張方式に従って、文字の追加が可能な領域(G3)があります。この領域にSJIS(R90)、JEFとの互換性を考慮して拡張漢字の追加を行ったものにEUC(U90)があります。

- EBCIDIC-JEF

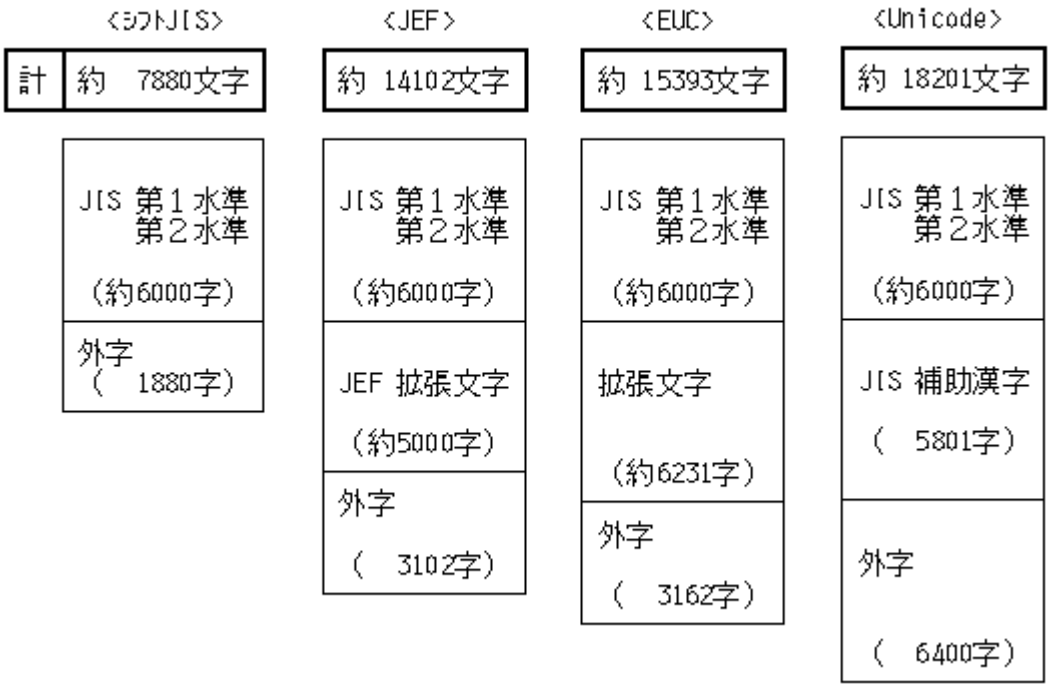
  富士通のOSIVシリーズで使用されるコード系です。英数字・カナ文字(EBCDIC)、日本語文字(JEF)を混在させる方法で、次のような特徴を持ちます。

  — 文字種の切り換えにシフトコードを使用する
  — 各文字のコードはすべてEBCDICおよびJEFに完全に一致する

なお、各文字コード系はコードと対応する文字が実際は規定されていない領域を含みます。これらの領域にはユーザが任意の文字を割り当てられます。これを外字または利用者定義文字と呼びます。

以下に各コード系で使用可能な文字の範囲について、概要を図で示します。

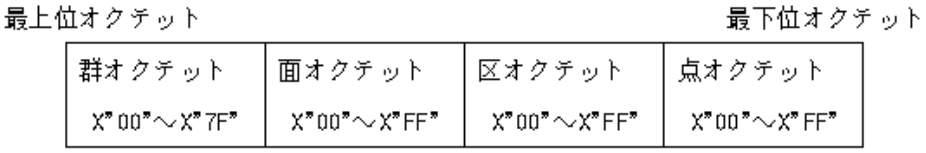
図F-2 各コード系における使用可能な文字の比較



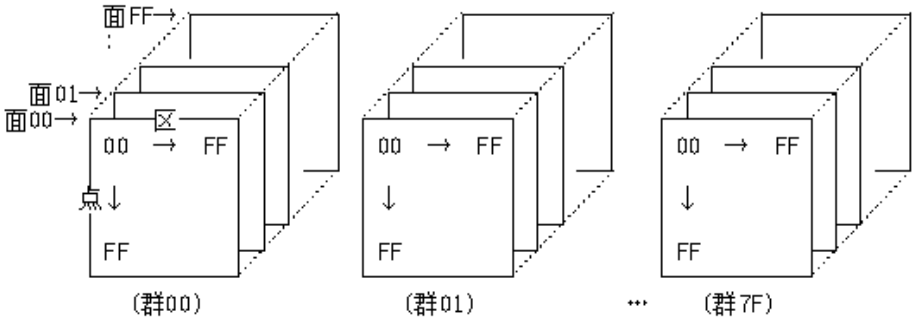
F.1.3 Unicode

ここまで、説明してきた文字コード系は特定の国や地域での使用を前提として設計されたものです。これに対して、Unicodeは地域や国を限定せず、はじめから世界中の文字を１つの体系で表現できるように設計されました。ただ、世界中で共通に使用可能な文字コード系(ユニバーサル・コードセット)を作成しようとする試みには歴史的な紆余曲折や技術的な困難があり、これがUnicodeを分かりづらいものとしています。

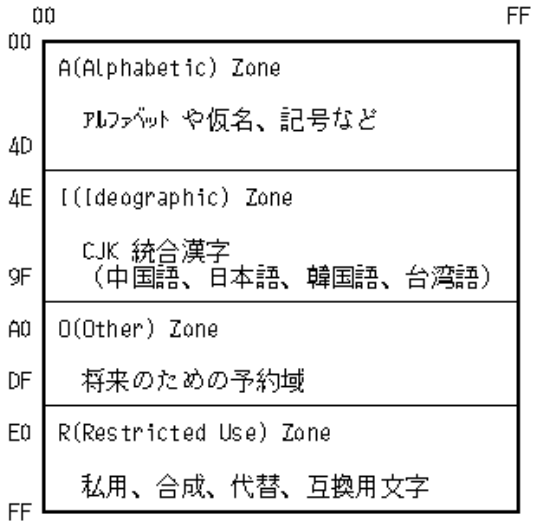
当初、ユニバーサル・コードセットの試みは、国際規格であるISO/IEC 10646として始められました。ISO/IEC 10646は、全世界の文字を1つのコード体系で表現することを目的に制定された国際規格で、マルチオクテット化(1文字を32ビットで表現)により、単純計算で21億を超える文字を収納できるキャパシティを持っています。
1文字は下図のとおり4つのオクテットから構成されます。



このマルチオクテット構造をコード表イメージで図解したものが下図です。



これに対して、米国のコンピュータメーカが中心となって設立されたUnicode consortiumが制定したコード体系がUnicodeです。ISO/IEC 10646の考え方と最も異なる点は、16ビットの枠内に実用範囲の文字を詰め込んだことにあります(下図)。



結局、2つの規格が互いに歩み寄り、ISO/IEC 10646の最初の1面(群00面00、BMPと呼ばれている面)には、Unicodeがそのまま採用されています。また、Unicodeもバージョン2.0以降では対応する文字種の拡張を決定しています。

表現形式

Unicode(ISO/IEC 10646)は、その複雑な背景から複数の表現形式を持っており、これがUnicodeを理解しづらい要因でもあります。ここでは、それら表現形式について説明します。

UCS-4

1文字は4バイト固定で表現されます。

ISO/IEC 10646に収録されるすべての文字を表現することが可能ですが、現在のところ文字の配置が決まっているのはBMP(=Unicode)だけです。このため、上位2バイトには常にX”00”が詰められます。

例：　”富士通”　→　X"00005BCC 000058EB 0000901A"
**　　”AB12”　→　X"00000041 00000042 00000031 00000032"**

UCS-2

1文字は2バイト固定で表現されます。

BMPの範囲しか表現することができませんが、現在のところ最も一般的に使用されている表現形式です。多くの場合、Unicodeと言えばこのUCS-2を意味します。

例：　”富士通”　→　X"5BCC 58EB 901A"
**　　”AB12”　→　X"0041 0042 0031 0032"**

なお、UCS-2(UCS-4も同様)はビッグエンディアンとリトルエンディアンに細分化されます。上の例がビッグエンディアンで、下の例はリトルエンディアンです。

例：　”富士通”　→　X"CC5B EB58 1A90"
**　　”AB12”　→　X"4100 4200 3100 3200"**

リトルエンディアンは、Intelアーキテクチャの CPUを搭載するコンピュータで一般に使用される、バイトスワップ(上位バイトと下位バイトが逆転)された表現形式のことです。

UTF-8

1文字は1～6バイトの可変長で表現されます。

BMPの範囲であれば最大3バイト/1文字で表現できます。半角の英数字(ASCII文字)は1バイト/1文

字、一部記号類は2バイト/1文字、漢字やかなどの日本語は3バイト/1文字になります。

例： ”富士通” → X"E5AF8C E5A3AB E9809A"
”AB12” → X"41 42 31 32"

ASCIIと互換性があることから、欧米でよく使用されているようです。

UTF-16

1文字は2バイトまたは4バイトの可変長で表現されます。
BMPに加えて、BMPの「使用禁止コード」を利用して表現できる文字数を拡大した形式で、UCS-2の拡張形式とも言えます。サロゲート方式とも呼ばれる次世代の表現形式ですが、現在のところ実装しているOSはありません。

Unicodeのメリット

使用可能な文字種

氏名や地名にはJIS第1水準/第2水準外の文字がよく使われます。シフトJISでは、これらの文字をデータとして扱いたい場合、外字として登録するか、常用漢字で代用するしかありませんでしたが、Unicodeでは、多くの場合、問題なく利用できます。“図:各コード系における使用可能な文字の比較”は、日本語を例に収納文字数を比較したものです。
このように、表現できる文字数の差は歴然としており、表現できる文字数に不満を持っている方には、Unicode化が有効な解決策になります。

国際化

Unicodeには、欧州、中近東、インド、東南アジアなどの文字や記号類に加え、中国、ハングル、台湾、もちろん日本語もすべて収納されています。これは、アプリケーションの多国語化が可能になったことを意味します。もちろん、多国語化はコード系だけの問題ではないため、Unicode化しただけでローカライゼーションが不要になるわけではありませんが、多国語化の基盤技術として重要な意味を持ちます。

マルチベンダ対応

マルチベンダシステムを構築する場合、データの流通性が壁となるケースが多くありました。このような場合、世界共通語でもあるUnicodeでデータを統一することによって、国境のない、真のオープン環境を構築することが可能になります。

Unicodeのデメリット

OS/製品の対応状況

サポートの方式や状況がオペーレーティングシステム、製品毎にまちまちです。このため、Unicode対応を謳っている製品であっても、うまく日本語が扱えないような状況もあります。
また、どの表現形式を採用しているかも重要な問題となっています。

文字データの格納サイズ

UTF-8のように可変長の内部表現方式を採用する場合、文字数から単純に格納領域のサイズを求めることが困難になります。

文字の大小順序

Unicodeでの文字の配置(順序)に他のコード゛系との互換性はありません。もちろんASCII文字(半角英数字)の範囲では同じですが、漢字の並びは全く異なっており、かなや英数字の配置も異なります。このため、ソートや文字の大小比較を行う場合、シフトJISと結果が異なることがあります。

表F-1 各コード系における日本語文字の大小順序

コード系	文字の大小順序
Unicode	かな＜ カナ＜ 数字＜ 英字
SJIS/EUC	数字＜ 英字＜ かな＜ カナ

JEF	数字< 英字< かな< カナ

なお、各カテゴリ内での文字の順序にはUnicodeとSJIS/EUCの間に互換性があります。

小文字 < 大文字 < 濁音 < 半濁音

ぁ	あ	ぃ	い	…	か	が	き	ぎ	…	は	ば	ぱ	ひ	び	ぴ	…

F.2 COBOL製品のサポートするコード系

オペレーティングシステム毎に富士通のCOBOL製品がサポートする文字コード系を“F.2 表:COBOL 製品のサポートするコード系”に示します。

表F-2 COBOL 製品のサポートするコード系

オペレーティングシステム		文字コード系	製品名
ホスト系	OSIV/MSP OSIV/XSP	EBCDIC/JEF	COBOL85
UNIX系	Solaris	EUC	COBOL97 NetCOBOL
		SJIS	
		Unicode	
	Linux	EUC	NetCOBOL
		Unicode	
	HP-UX	EUC	COBOL85
		SJIS	
Windows系	Windows 2000 Windows XP Windows Server 2003	SJIS	COBOL97 NetCOBOL
		Unicode	
		EBCDIC/JEF	JEFオプション

コード系のサポートは、オペレーティングシステムのコード系の扱いに強く依存します。
このため、NetCOBOLのUnicodeサポートは次のように異なります。

オペレーティングシステム	Unicode使用の指定方法	プログラム資産のコード系
Windows系	翻訳オプション:RCS(UCS2)	SJIS
UNIX系	環境変数:LANG=ja_JP.UTF-8	UTF-8

より詳しくは、各システムのNetCOBOL製品の使用手引書を参照してください。

F.3 文字コードの違いのCOBOLプログラミングへの影響

文字コードの違いがCOBOLプログラミングに与える影響は次の2つに大きく分かれます。

- コード変換
- コード値の非互換

以下、それぞれについて説明します。

F.3.1 コード変換とその影響

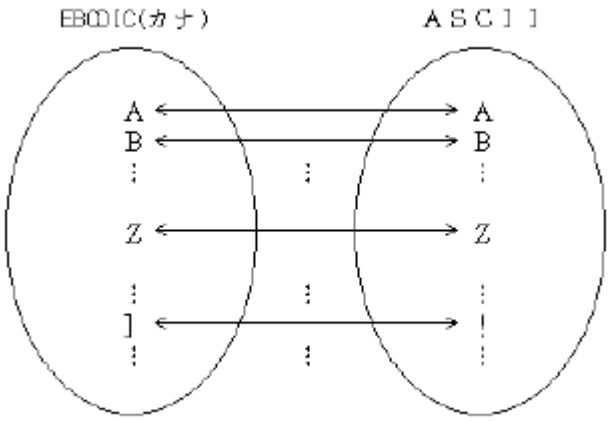
コード変換とは、ある文字コード系で表現されている文字情報を、異なる文字コードを用いた表現に変換することです。コード変換は、異なる文字コード系のシステムの間でデータ交換を行う場合に必須となります。
このコード交換の方法は大きく次の2つに分けられます。

- 変換の前後で文字が一致する変換(対称的な変換)
  変換先のコード系に含まれる文字の種類が、変換元のコード系に含まれる文字の種類と等しいか、より大きい場合です。逆方向の変換により、元の文字を復元できます。
- 変換の前後で文字が一致しない変換(非対称な変換)
  変換先のコード系に含まれる文字の種類が、変換元のコード系に含まれる文字の種類より小さいか、単純な包含関係が成り立たない場合です。

後者の場合、変換を続けるなら、変換元と異なる文字への変換をせざるを得ませんが、その場合についても、さらに2つの方法に分けられます。

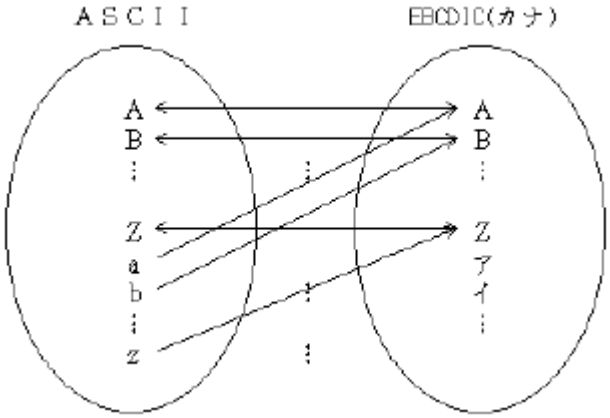
- 代替文字による変換
  変換元と異なる文字を使用せざるを得ないが、変換の前後で1対1の対応を維持できる場合です。
  例えばEBCDICからASCIIへの変換をした場合、“!”が“]”に変換されますが、逆方向の変換をすることによって、元に戻すことができます。

図F-3 代替文字による変換の概要



- 縮退による変換
  変換元と異なる文字を使用するだけでなく、変換先の同じ変換文字に対して、変換元の複数の字が対応する場合です。例えばEBCDIC(カナ)からASCIIへの変換をした場合、“a”と“A”が“A”に変換されます。“a”から“A”への変換が行われた場合、逆変換を行っても“a”に戻すことはできません。

図F-4　縮退による変換の概要



他のシステムで動作するアプリケーションを開発する場合や他のシステムで動作していたアプリケーションを移植する場合、ソースプログラムやデータファイルの一部にコード変換を行う必要があります。この際のコード変換が対称的な変換なら特に問題が発生することはありませんが、非対称なコード変換である場合には問題が生じます。
以下、それについて説明します。

EBCIDC→JIS8変換時の代替文字による変換

説明

以下の一覧に示す文字の変換に関して、網かけ部分の文字は代替文字による変換が行われます。

変換元:EBCDIC				変換先:JIS8	
コード値	ASCII	英小	カナ	コード値	字形
0x4F	“！”	“｜”	“｜”	0x21	“！”
0x4A	“[”	“£”	“£”	0x5B	“[”
0x5A	“]”	“！”	“！”	0x5D	“]”
0x5B	“＄”	“＄”	“￥”	0x23	“＄”
0x5F	“＾”	“¬”	“¬”	0x5E	“＾”
0xA1	“~ ”	“￣”	“￣”	0x7E	“￣”
0xE0	“＼”	“＄”		0x5C	“￥”

注意

上の表中では字形を明らかにするため、あえて日本語文字で表示しています。

COBOLプログラミングへの影響

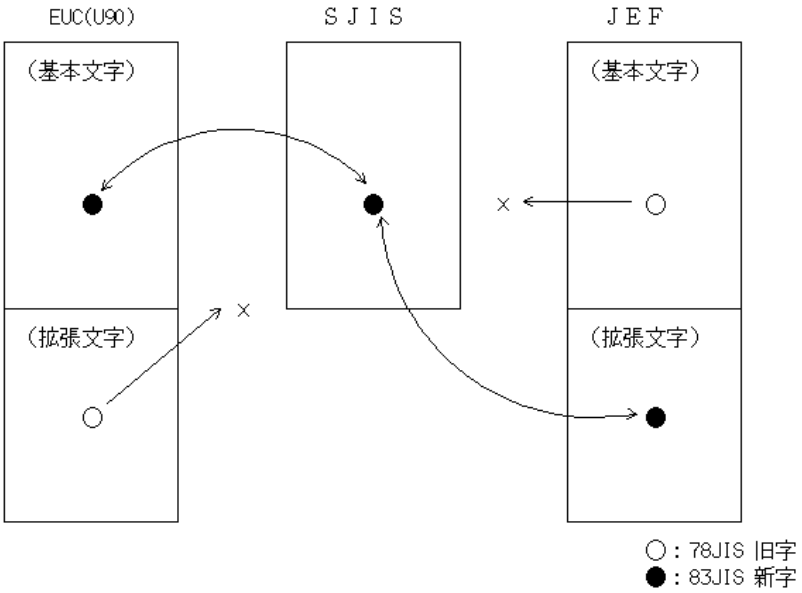
プログラムの記述やデータに上記の表の網かけ部分の文字が含まれていた場合、表示・印刷の結果が以前と異なります。

SJIS へのコード変換時の縮退による変換

説明

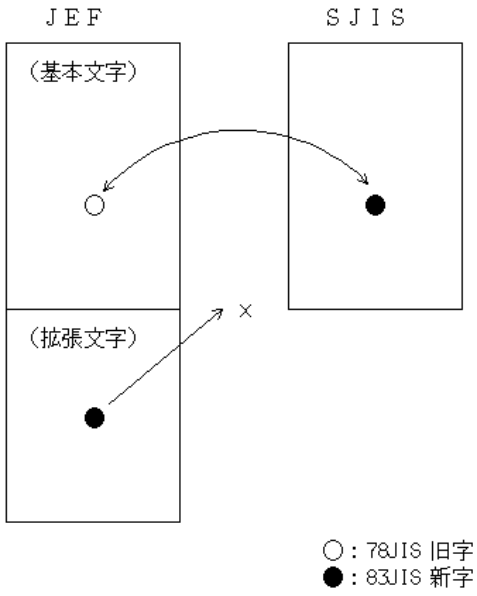
JIS漢字コード系は何度か改定が行われていますが、1983年に行われた改定がもっとも大きなものでその前後で一部の非互換を持ちます(以後、改定前を78SJISと改定後を83JISと呼びます)。
改定による変更のもっとも大きなものは文字の字体の変更です(371字の変更中、248字)。例えば、“鷗”という字体から“鴎”という字体に変更が行われました。

JEFおよびEUC(U90)は78SJISと83JISで非互換のある文字についてそれぞれの字形毎に別のコードが割り当てられていますが、SJISでは1つのコードしか割り当てられていません。
このため、一般的にはJEFおよびEUC(U90)に含まれていた78JISの文字の情報は失われてしまいます。



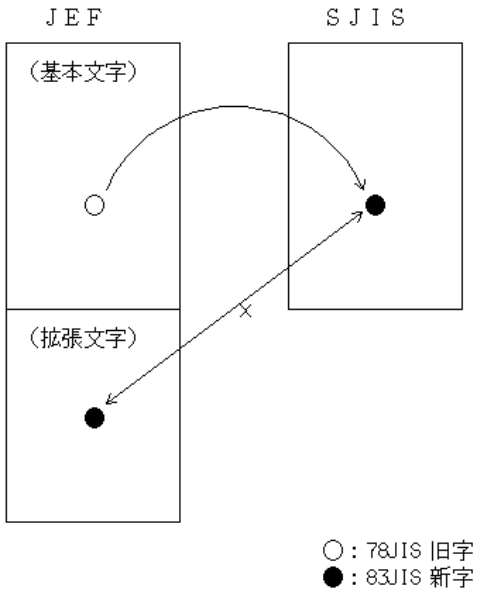
変換前のソースプログラム、データファイル等がEUC(U90)の場合、78JIS の旧字体の文字は拡張文字セットに含まれるものであるため、このような変化が行われても問題となることはあまりありません。
しかし、変換前のソースプログラム、データファイル等がJEFの場合、78JIS の旧字体の文字は基本文字セットに含まれるものであるため、より深刻な影響を被る可能性があります。そのような場合、コード変換の方法を変更し、次のいずれかの変換を行う必要があります。

- 字形を無視した変換
  78JISによる旧字と83JISによる新字の違いを無視して、次のように変換を行います。



この方法をとる場合、PC上で表示される文字の字形は変換前と異なるものとなってしまいますが、OSIVアプリケーションの分散開発等の場合は、この方法がお勧めです。

- 縮退を利用した変換
  78JISによる旧字と83JISによる新字を縮退による変換を用いて変換します。



この方法をとる場合、78JISによる旧字と83JISによる新字の区別がつかなくなってしまいます。このため、あまりお勧めできる方法ではありません。しかし、OSIVからのアプリケーションの移植などで字体の違いを強く意識しないような場合は、この方法での変換が効率的な解決方法になる場合もあります。

COBOLプログラミングへの影響

データとして使用可能な文字（字体）が減少します。
外字などの登録により、一部対応が可能ですが、外字として登録可能な文字数も他のコード系に比べ少なく、根本的な解決にはなりません。
縮退による変換や変換先の文字が存在しないため変換エラーが発生している場合、次のような現象が起こります。

- 日本語利用者語などに使用されていた場合
  翻訳エラー(JMN1008I-S)となる場合があります。
- データとしてい使用されていた場合
  表示・印刷の結果が以前と異なります。

半角カタカナ文字の変換

説明

半角カタカナと呼ばれる1バイトの英数字文字と同じ表示幅を持つ文字はコード系によって、コード値の格納に必要な領域長が次のように異なります。

コード系		格納に必要なサイズ
SJIS		1バイト
EBCDIC(カナ)		1バイト
EUC		2バイト
Unicode	UCS-2	2バイト
	UTF-8	3バイト

このため、変換の前後で半角カタカナ文字を含むデータのサイズが変わってしまいます。

COBOLプログラミングへの影響

しばしば、プログラムのロジックの見直しが必要なる重大な問題となります。主な現象として次のようなものが考えられます。

- VALUE句に指定の文字定数に含まれる場合
  翻訳エラー(JMN2038I-S)となる場合があります。
- 転記などのデータ操作の場合
  データの後ろの部分が欠けてしまうことがあります。
- データファイル内に含まれる場合
  実行時エラーや不正なデータ読込みの原因となります。

F.3.2 コード値の非互換とその影響

文字コード値の非互換は、多くの場合はCOBOLの言語の機能により隠蔽されるため、意識する必要ありません。しかし、以下のような場合については、その限りではありません。

- 16進文字定数、日本語16進文字定数などを使用している場合
- EBCDIC/JEFコード系とその他のコード系との間の非互換

前者については、特に説明の必要はないと思われますので、ここではEBCDIC/JEFコード系とその他のコード系との間の非互換についてのみ説明します。

文字の大小順序

ASCII/シフトJIS とEBCDIC/JEFでは、コード系の違いにより、文字の大小順序が異なります。

- 英字、数字、カナの大小順序が逆転します。
- 外字と外字以外の日本語の大小順序が逆転します。

```
01  英数字     PIC  X  VALUE "A".
01  数字       PIC  9  VALUE 1.
          :
     IF     英数字 < 数字  THEN
            DISPLAY  "EBCDIC/JEF"
     ELSE
            DISPLAY  "ASCII/SJIS"
     END-IF.
```

OSVI系システムのCOBOL85やJEFオプションではTHEN側のDISPLAY 文が動作し、COBOL97/NetCOBOLではELSE側のDISPLAY 文が動作します。

空白コードの違い

日本語文字の空白コード(２バイト空白)と、英数字の空白コード(１バイト空白)がEBCDIC/JEFのときは同じ値(X'4040'とX'40')ですが、その他の文字コード系にはこの種の対応関係はありません。

表F-3 各コード系の英数字文字の空白コードと日本語文字の空白コード

英数字空白文字		日本語空白文字	
ASCII	X”20”	SJIS	X”8140”
		EUC	X”A1A1”
UTF-8	X”20”	“E38080”	
UTF-2	X”0020”	“3000”	

この結果、次のような操作については注意が必要です。

- 表意定数SPACEを使った転記、比較等
- 異なる長さのデータの転記などにより生じる文字列のパディング

● 日本語項目と集団項目の比較

例えば、次のような手続きがあった場合、

```
01  集団.
    02  FILLER    PIC N(2)  VALUE  NC"日本".
    02  FILLER    PIC X(2)  VALUE  SPACE.

01  日本語        PIC N(3)  VALUE  NC"日本".

    IF  集団  =   日本語  THEN
        DISPLAY   "EBCDIC/JEF"
    ELSE
        DISPLAY   "ASCII/SJIS"
    END-IF.
```

OSVI系システムのCOBOL85やJEFオプションではTHEN側のDISPLAY 文が動作し、COBOL97/NetCOBOLではELSE側のDISPLAY 文が動作します。

型の異なる項目の再定義

外部10進項目や日本語項目を異なる型のデータで再定義(REDEFINES) しているプログラムは、コード系の違いに注意する必要があります。
外部10進項目の内部表現は、(そのまま文字として表示可能なように)システムの文字コード系に依存して決められているため、このような影響を避ける事ができません。

```
01  英数字    PIC  X(3)  VALUE  "12L".
01  数字      REDEFINES  英数字  PIC  S9(3).

    IF  数字  =  -123  THEN
        DISPLAY  "EBCDIC/JEF"
    ELSE
        DISPLAY  "ASCII/SJIS"
    END-IF.
```

OSVI系システムのCOBOL85やJEFオプションではTHEN側のDISPLAY 文が動作し、NetCOBOL ではELSE側のDISPLAY 文が動作します