



B1WW-7071-01Z0(00)

Microsoft® Windows® 98

Microsoft® Windows® Me

Microsoft® Windows NT®

Microsoft® Windows® 2000

Microsoft® Windows® XP

Microsoft® Windows Server™ 2003

NetCOBOL for Windows V8.0

# UNIX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UNIX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) 98 operating system
- Microsoft(R) Windows(R) Millennium Edition
- Microsoft(R) Windows NT(R) Workstation operating system Version 4.0
- Microsoft(R) Windows NT(R) Server Network operating system Version 4.0
- Microsoft(R) Windows NT(R) Server Network operating system Version 4.0, Terminal Server Edition
- Microsoft(R) Windows NT(R) Server Network operating system, Enterprise Edition Version 4.0
- 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(TM) 2003, Standard Edition operating system
- Microsoft(R) Windows Server(TM) 2003, Enterprise Edition operating system

## 本書の目的

本書は、Windows版のNetCOBOLを利用して、UNIX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。

## 本書の対象読者

本書は、Windows版のNetCOBOLを利用してUNIX系システムで動作するCOBOLプログラムを開発する方を対象としています。

本書で説明するUNIX分散開発支援機能は、UNIX系システムの操作知識が十分でなくとも利用可能です。しかし、その機能を使用するための環境設定を行うには、UNIX系システムについてある程度の知識が必要です。



このため、本書では対象とする読者層を大きく2つに分類します。

- 開発管理者  
UNIX系プログラムの分散開発にあたって、その開発計画の立案と開発環境の構築に当たります。開発に使用するUNIX系サーバの環境の設定や一般の開発者が使用する簡単なツール（開発環境設定のためのもの）を用意する必要があり、ある程度のUNIX系システムの知識が必要となります。
- 一般の開発者  
開発管理者の指示に従って、UNIX系プログラムの分散開発を行うために必要な個人の開発環境を設定し、開発作業を実施します。

本書では、この2つの読者層に向けた説明を、できる限り分けて記述しています。

また、2章以降では、各記事がどちらの読者層を主な読者として想定しているかを、見出しに以下のアイコンをおいて示します。

---

見出しのアイコン	対象読者
	開発管理者
	一般の開発者

## 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
- COBOLプログラム開発に関する基本的な知識
- Windows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](#)

Windows版のNetCOBOLを使用したUNIX系プログラムの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ます。

### 第2章 [分散開発環境の構築](#)

Windows版のNetCOBOLを使用したUNIX系プログラムの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ます。

### 第3章 [Windowsクライアントでの開発作業](#)

Windows版のNetCOBOLを使用したUNIX系プログラムの分散開発において、Windowsクライアント上での開発作業の詳細について説明します。

### 第4章 [サーバ環境での開発作業](#)

Windows版のNetCOBOLを使用したUNIX系プログラムの分散開発において、UNIXサーバ上での開発作業の詳細について説明します。

### 第5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ます。

### 付録A [NetCOBOL製品の相違点](#)

オープン系のCOBOL (NetCOBOL) 製品間の仕様上／機能上の相違点について説明します。

### 付録B [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ます。

### 付録C [V7.2との分散開発支援機能の機能差](#)

NetCOBOL for Windows V7.2を使用して、UNIX系プログラムの分散開発を行う場合に注意する必要がある機能差について説明します。

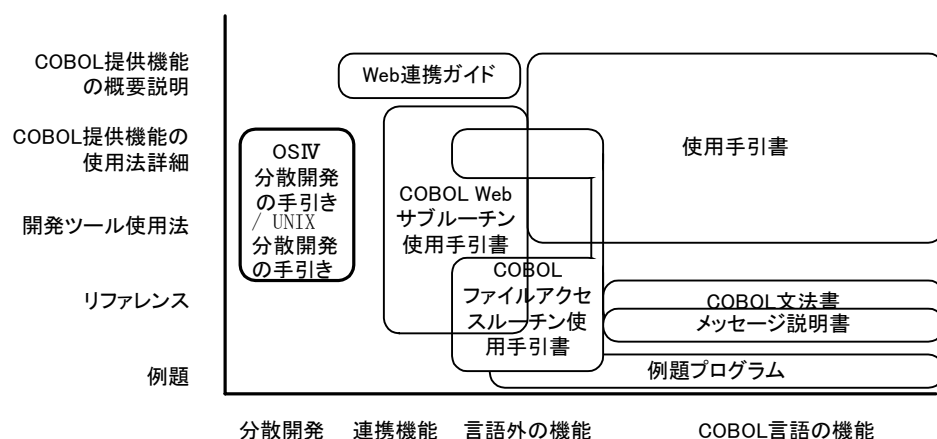
## 本書の位置付け

このマニュアルは、UNIX系プログラムの分散開発を行う際の手順と、そのために提供される開発ツール類の一般的な使用法を説明するものです。

COBOL言語として提供される機能の詳細については、“COBOL文法書” および各システム向けの“NetCOBOL 使用手引書”を参照してください。

NetCOBOL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
OSIV分散開発の手引き	OSIV系システム上で動作するCOBOLプログラムの分散開発の手順の説明
メッセージ説明書	NetCOBOLが出力するメッセージの説明
例題プログラム	サンプルプログラムの解説および実行方法の説明
Web連携ガイド	COBOLによるWebアプリケーションの作成および実行方法の説明
COBOL Webサブルーチン使用手引書	
COBOLファイルアクセスルーチン使用手引書	C言語からCOBOLファイルをアクセスする関数群の説明



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）	製品名	使用目的
FORM V8.0 説明書 FORM V8.0 ヘルプ PowerFORM V8.0 ヘルプ	FORM V8.0 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 V8.0 説明書	MeFt V8.0	画面帳票定義体を使用したプログラムの実行
PowerSORT Workstation V4.0 使用手引書 PowerSORT Server V4.0 使用手引書	PowerSORT Workstation V4.0 PowerSORT Server V4.0	PowerSORTを使用した整列併合を行うプログラムの実行

#### 注

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。



“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。

## 製品の呼び名について

本書では、説明の中で以下に示す名称、略称および記号を使用しています。

マニュアル上の表記	正式名称
Windows	<ul style="list-style-type: none"><li>● Microsoft(R) Windows(R) 98 operating system</li><li>● Microsoft(R) Windows(R) Millennium Edition</li><li>● Microsoft(R) Windows NT(R) Workstation operating system Version 4.0</li><li>● Microsoft(R) Windows NT(R) Server Network operating system Version 4.0</li><li>● Microsoft(R) Windows NT(R) Server Network operating system Version 4.0, Terminal Server Edition</li><li>● Microsoft(R) Windows NT(R) Server Network operating system, Enterprise Edition Version 4.0</li><li>● Microsoft(R) Windows(R) 2000 Professional operating system</li><li>● Microsoft(R) Windows(R) 2000 Server operating system</li><li>● Microsoft(R) Windows(R) 2000 Advanced Server operating system</li><li>● Microsoft(R) Windows(R) XP Professional operating system</li><li>● Microsoft(R) Windows(R) XP Home Edition operating system</li><li>● Microsoft(R) Windows NT(R) Server Network operating system Version 4.0</li><li>● Microsoft(R) Windows NT(R) Server Network operating system Version 4.0, Terminal Server Edition</li><li>● Microsoft(R) Windows NT(R) Server Network operating system, Enterprise Edition Version 4.0</li><li>● Microsoft(R) Windows Server(TM) 2003, Standard Edition operating system</li><li>● Microsoft(R) Windows Server(TM) 2003, Enterprise Edition operating system</li></ul>
Solaris	<ul style="list-style-type: none"><li>● Solaris™ 7 オペレーティングシステム</li><li>● Solaris™ 8 オペレーティングシステム</li><li>● Solaris™ 9 オペレーティングシステム</li></ul>
Linux32	<ul style="list-style-type: none"><li>● Red Hat Enterprise Linux AS (v.3 for x86)</li><li>● Red Hat Enterprise Linux ES (v.3 for x86)</li><li>● Red Hat Enterprise Linux AS (v.2.1 for x86)</li><li>● Red Hat Enterprise Linux ES (v.2.1 for x86)</li></ul>
Linux64	<ul style="list-style-type: none"><li>● Red Hat Enterprise Linux AS (v.4 for Itanium)</li></ul>
Linux	<ul style="list-style-type: none"><li>● Red Hat Enterprise Linux AS (v.3 for x86)</li><li>● Red Hat Enterprise Linux ES (v.3 for x86)</li><li>● Red Hat Enterprise Linux AS (v.2.1 for x86)</li><li>● Red Hat Enterprise Linux ES (v.2.1 for x86)</li><li>● Red Hat Enterprise Linux AS (v.4 for Itanium)</li></ul>
Interstage	Interstage Application Server
CharsetMGR	<ul style="list-style-type: none"><li>● SystemWalker/CharsetMGR</li><li>● Interstage Charset Manager</li></ul>

## 用語の説明

本書では、以下の用語を使用します。

### UNIX系システム:

NetCOBOLが動作可能なUNIX系オペレーティングシステムの総称。

## UNIX系プログラム:

UNIX系システムで動作するプログラム

## Windowsシステム:

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。

## 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。
<u>あいうえお</u>	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. <u>プログラム名</u> . → PROGRAM-ID. SAMPLE1.
$\left\{ \begin{array}{l} \text{あい} \\ \text{うえお} \end{array} \right\}$ 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 (R) 2000で採取したものです。ただし、Windows XPでのみ発生する問題を説明する場合は、その限りではありません。
- 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。

## 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、Windows NT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。

UNIXは、米国およびその他の国におけるオープン・グループの登録商標です。

X Window Systemは、オープン・グループの商標です。

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。

Linuxは、Linus Torvalds氏の米国およびその他の国における登録商標あるいは商標です。

HPおよびHP-UXは、米国Hewlett-Packard Companyの商標です。

C-ISAMは、米国Informix Software, Inc. の米国での登録商標です。

Micro Focusは、Micro Focus International Limited. の商標です。

INTERSTAGE、Interstageは、富士通株式会社の登録商標です。

Red Hat、RPMおよびRed Hatをベースとしたすべての商標とロゴは、Red Hat, Inc. の米国およびその他の国における登録商標あるいは商標です。

Intel、Itaniumは、Intel Corporationの登録商標です。

その他の会社名または製品名は、それぞれ各社の商標または登録商標です。

2005年6月

All Rights Reserved, Copyright (C) 富士通株式会社 1992-2005

---

---



---

# 目次

第1章 分散開発の概要	1
1.1 分散開発とは？	2
1.1.1 NetCOBOLの提供する分散開発の支援機能	2
1.2 UNIX系プログラムの分散開発	4
1.2.1 分散開発の作業の流れ	4
1.2.2 分散開発のメリットとデメリット	5
1.2.3 分散開発の適用範囲	6
1.3 UNIX系プログラムの分散開発環境概要	14
1.3.1 分散開発環境の基本的なシステム構成	14
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	15
第2章 分散開発環境の構築	17
2.1 分散開発環境構築の前提知識	18
2.1.1 COBOLプロジェクトマネージャにおけるプロジェクト	18
2.1.2 分散開発時のUNIXサーバ上の環境	20
2.1.3 リモートデバッグの概要	20
2.1.4 資産管理方法と資産の配置	27
2.2 分散開発計画の立案	32
2.3 分散開発のための環境設定	34
2.3.1 サーバ側の環境設定	34
2.3.2 クライアント側の環境設定	41
第3章 Windowsクライアントでの開発作業	49
3.1 Windowsクライアントでの作業概要	50
3.2 プロジェクトの作成	51
3.2.1 基本的なプロジェクトの作成	51
3.2.2 分散開発固有の設定	61
3.2.3 特殊なプロジェクトの作成	63
3.3 開発資産のWindowsクライアント環境への移行	68
3.3.1 OSIVからのプログラミング資産の移行	68
3.3.2 UNIX系システムからのプログラミング資産の移行	69
3.4 プログラミング	71
3.4.1 ソース・登録集原文の作成	71
3.4.2 各種定義体の作成、修正	81
3.5 翻訳チェックとリンク	83
3.5.1 UNIX系プログラムの翻訳	83
3.5.2 UNIX系プログラムのリンク	83
3.6 Windowsクライアントでの単体テスト	85
3.6.1 Windowsクライアントでの単体テストの概要	85
3.6.2 COBOLのデバッグ機能	86
3.6.3 対話型デバッガによるデバッグ	87
第4章 サーバ環境での開発作業	93
4.1 サーバ環境へのプログラム資産の登録	94
4.1.1 COBOLソース・登録集の送信	94
4.1.2 各種定義体の送信	96
4.2 ビルド制御文生成機能	97
4.2.1 ビルド制御文雛型の生成時の規則	97
4.2.2 ビルド制御文雛型の生成手順	105
4.2.3 生成したビルド制御文雛型とその修正	114
4.3 ターゲットビルド	122
4.3.1 サーバ環境へのビルド制御文の転送	122

---

4.3.2	ターゲットビルドの実行.....	122
4.3.3	ターゲットビルド後のプログラム資産の再修正.....	123
4.4	リモートデバッグ.....	125
4.4.1	リモートデバッグの概要.....	125
4.4.2	リモートデバッグ時の注意点.....	133
第5章	トラブルシューティング.....	135
5.1	サーバ連携のトラブル.....	136
5.1.1	分散開発環境の構築.....	136
5.1.2	サーバへの資産の送信.....	137
5.1.3	ビルド制御文生成.....	139
5.1.4	ターゲットビルド.....	140
5.2	リモートデバッグのトラブル.....	144
5.2.1	リモートデバッグの準備.....	144
5.2.2	リモートデバッグの開始.....	146
5.2.3	リモートデバッグの操作.....	150
付録A	NetCOBOL製品の相違点.....	153
A.1	富士通のCOBOL製品の概要.....	153
A.1.1	富士通のCOBOL製品の変遷.....	153
A.2	言語の機能の違い.....	154
A.2.1	概要.....	154
A.2.2	NetCOBOL製品間の機能差の詳細.....	155
A.3	翻訳・リンク処理に関する互換性.....	163
A.3.1	コマンド名と指定形式.....	163
A.3.2	翻訳オプション.....	164
A.3.3	コマンドラインオプション.....	169
A.3.4	環境変数.....	172
A.3.5	ライブラリ.....	173
A.4	実行環境.....	175
A.4.1	環境変数情報.....	175
付録B	文字コード系.....	181
B.1	文字コードの概要.....	181
B.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.....	181
B.1.2	文字種の混在方式による分類.....	182
B.1.3	Unicode.....	184
B.2	COBOL製品のサポートするコード系.....	187
B.3	文字コードの違いのCOBOLプログラミングへの影響.....	188
B.3.1	コード変換とその影響.....	188
B.3.2	コード値の非互換とその影響.....	192
付録C	V7.2における分散開発時の操作性の違い.....	195
C.1	環境設定.....	195
C.1.1	サーバ連携情報.....	195
C.2	Windowsクライアントでの作業.....	195
C.2.1	プロパティ.....	195
C.2.2	翻訳オプション.....	196
C.3	UNIXサーバ側での作業.....	196
C.3.1	ビルド制御文生成.....	196
索引	.....	199

---

---

# 第1章 分散開発の概要

---

NetCOBOLは、COBOL85規格および国際規格COBOL2002の一部を採用しています。この仕様の範囲でプログラムを記述することにより、各種オペレーティングシステムで動作するプログラムを開発できます。UNIX系システムで動作するプログラムの開発に、NetCOBOLのWindows版製品を使用することができます。

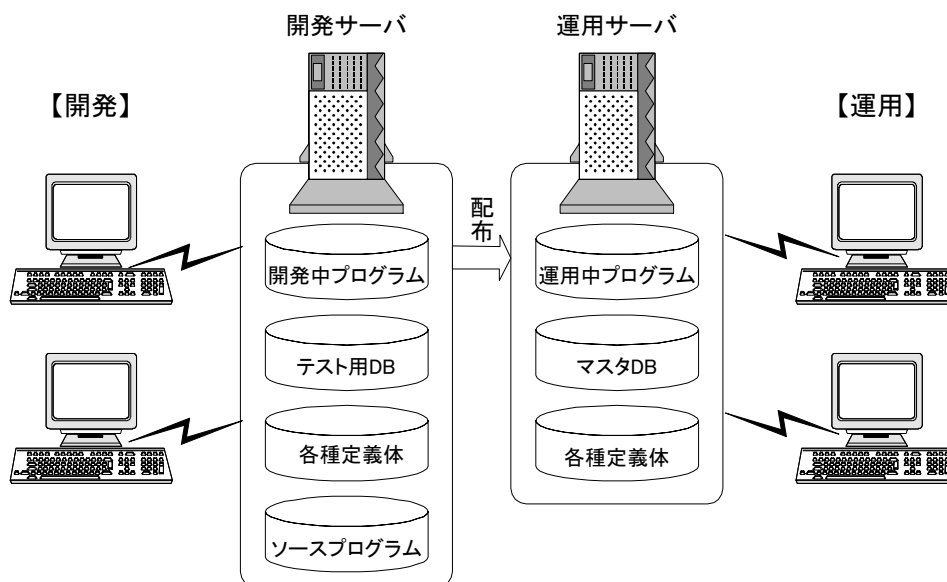
プログラムが実際に使用される環境とは異なる環境で、プログラム開発を行う形態を分散開発と呼びます。NetCOBOLのWindows版製品では、この分散開発の形態でUNIX系システムで動作するプログラムの開発をより効率的に行うための機能を提供しています。

---

## 1.1 分散開発とは？

通常のシステムやアプリケーションの開発は、最終的にそれを運用するときと同一のハードウェア、オペレーティングシステムを用いて行われます。

図1-1 通常の開発環境と運用環境の関係



しかし、システムやアプリケーションを構成するプログラムの一部を、最終的にシステムやアプリケーションを運用するハードウェアやオペレーティングシステムとは異なる環境で開発することがあります。このような開発形態を分散開発と呼びます。分散開発は、その目的や使用する開発ツールによってさまざまな形態を取るもので、広い意味を持つ言葉です。

このマニュアルでは、UNIX系システムで動作するプログラムの開発に、Windows PC上で動作するNetCOBOL製品を使用する場合について扱います(以降、分散開発の意味は、この開発形態に限定します)。

NetCOBOLでは、次のようなUNIX系システムで動作するプログラムの分散開発が可能です。

表1-1 NetCOBOLのサポートするUNIX系システム

No	オペレーティングシステム名	対象バージョン	CPU アーキテクチャ
1	日本語Solaris(TM) オペレーティングシステム	7.0、8.0 or 9.0	SPARC
2	Red Hat Enterprise Linux AS/ES	v. 2.1 or v. 3	x86系
3	Red Hat Enterprise Linux AS	v. 4	Itanium

### 1.1.1 NetCOBOLの提供する分散開発の支援機能

NetCOBOLの各プラットフォーム向けの製品は、基本的に同じCOBOLの言語仕様を提供しています。このため、多くの場合は、Windowsシステム上で開発したプログラム資産を使用して、同じ動作をするUNIX系プログラムを作成することが可能です。

NetCOBOLでは、この言語の基本的な機能だけではなく、分散開発を積極的に支援するために次のような機能を提供しています。

- プログラム資産の移行ための機能

Windows製品のCOBOLプロジェクトマネージャのプロジェクト単位でプログラム資産を

UNIX系システムに転送するために次のような機能を持ちます。

- 転送先のUNIX系システムの設定と選択
- Windowsシステム→UNIX系システム方向のファイル転送(送信)
- UNIX系システム→Windowsシステム方向のファイル転送(受信)

● UNIX系プログラムの翻訳・リンクを支援する機能

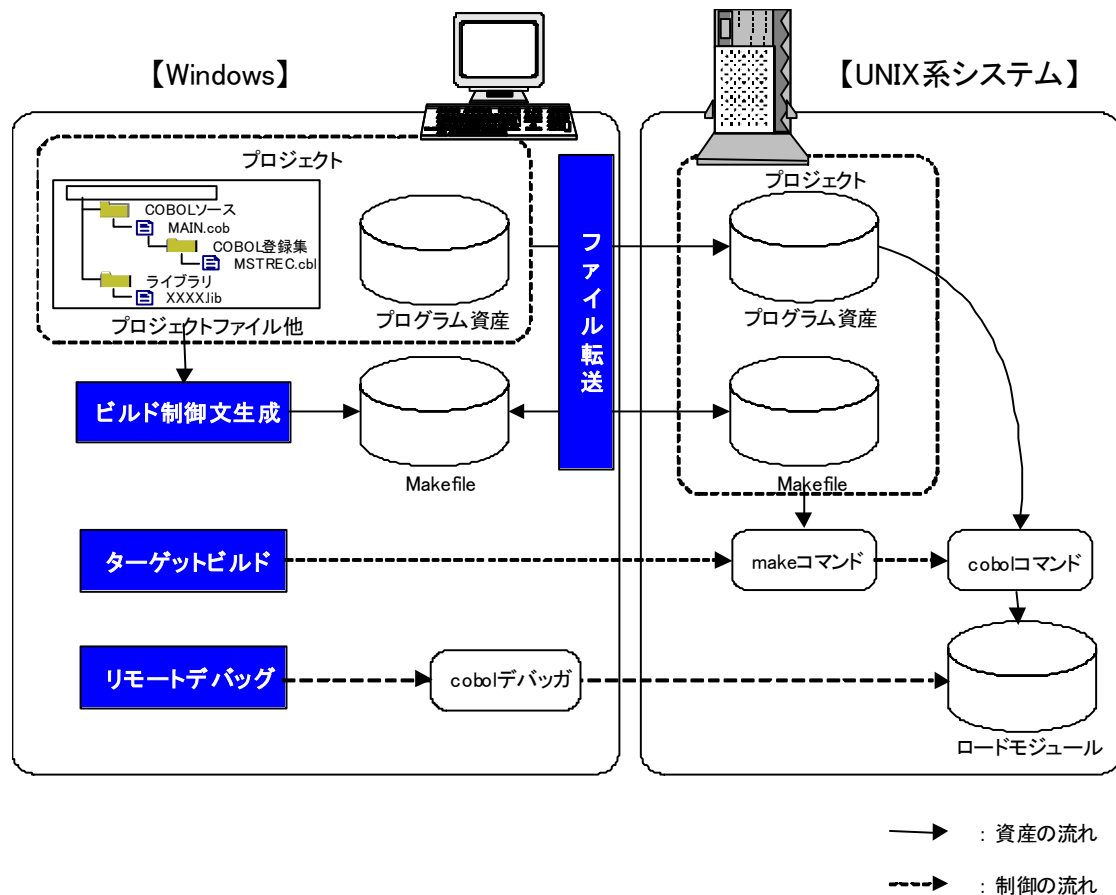
UNIX系システムに転送したプログラム資産を、UNIX系システム上で翻訳・リンクするために次のような機能を持ちます。

- UNIX系システム上での翻訳・リンクのための制御文生成(ビルド制御文生成)
- UNIX系システム上での翻訳・リンク操作の実施(ターゲットビルド)

● リモートデバッグ機能

UNIX系システム上で動作するプログラムをソースレベルでデバッグするために、Windows版の対話型デバッガが使用できます。

図1-2 NetCOBOLの提供する分散開発支援機能の概要



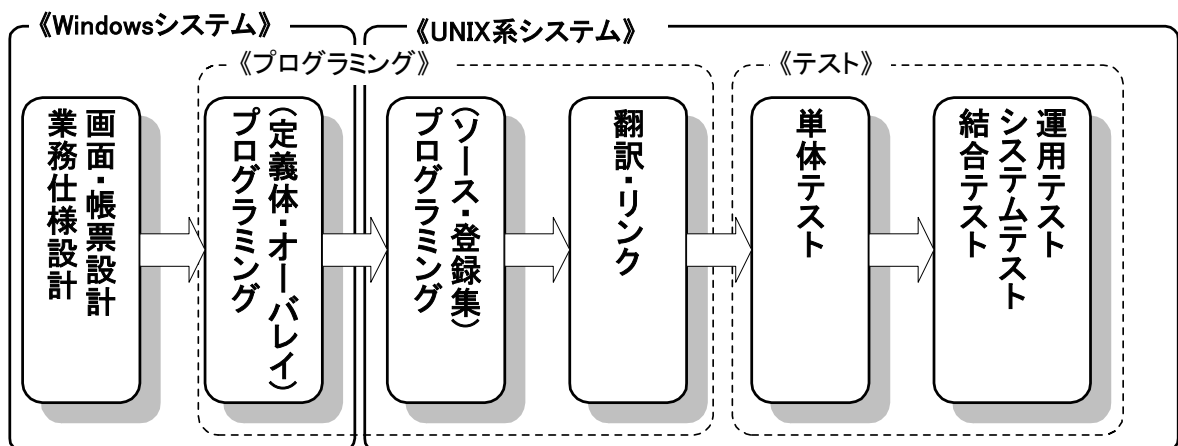
## 1.2 UNIX系プログラムの分散開発

ここでは、NetCOBOLを使用して、UNIX系プログラムの分散開発を行う場合の作業概要とその効果について、説明します。

### 1.2.1 分散開発の作業の流れ

これまでのUNIX系プログラムの開発は、“図1-3 通常のUNIX系プログラム開発の作業の流れ”で示すように、開発からテストに至るすべての作業をUNIX系システム上で行っていました。

図1-3 通常のUNIX系プログラムの開発作業の流れ

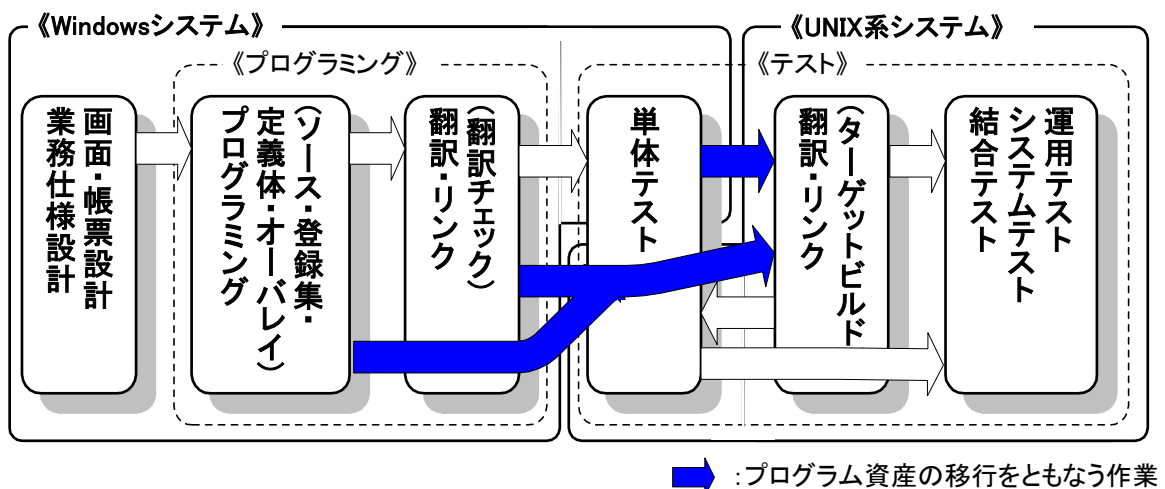


UNIX系システム向けのNetCOBOL製品による開発を行う場合、以下の示すプログラム資産の作成・修正を除き、すべての作業をUNIX系システムで行っていました。

- 画面帳票定義体
- フォームオーバーレイパターン
- ファイル定義体

一方、分散開発を行う場合は、“図1-4 分散開発時のUNIX系プログラム開発の作業の流れ”に示すように、Windowsシステムで実施する作業とUNIX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1-4 分散開発時のUNIX系プログラム開発の作業の流れ



➡ : プログラム資産の移行をともなう作業

“図1-4 分散開発時のUNIX系プログラム開発の作業の流れ”で青い矢印で示した部分では、プログラム資産をWindowsシステムからUNIX系システムに移行させる作業が必要となります。

NetCOBOLのWindows製品を使用して分散開発を行う場合、プログラミングから単体テストまでの作業をWindowsシステム上で実施できます。それ以降のUNIX系システムへのプログラム資産の移行やUNIX系システムで翻訳・リンク(ターゲットビルド)、テスト等の作業はWindows上のGUIツールを使用して実施することができます。

ただし、実際の分散開発でどの作業工程までをWindowsシステムで実施するかは、次の点に依存します。これらの詳細については“1.2.3 [分散開発の適用範囲](#)”で説明します。

- 分散開発を適用するプログラム開発の目的
- 分散開発を適用するプログラムの使用する機能

## 1.2.2 分散開発のメリットとデメリット

NetCOBOLを用いてのUNIX系プログラムの分散開発には、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で、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

- UNIX系システムの知識があまり必要ない  
UNIX系システムは、優れたネットワーク機能と安定性、セキュリティ強度の高さなどの利点から主にサーバ用途で利用される反面、開発作業もtelnet端末などを介してのコマンドベースの作業になります。このためには、UNIX系システムで使用するコマンドや環境設定などの知識が必要になります。  
分散開発を適用する場合、開発管理者が基本的な設定を行った後では、ほとんどの操作がWindowsシステム上からGUIによって可能となるため、一般の開発者はUNIX系システムについて最小限の知識で開発を行うことができます。
- Windowsシステムと同等の操作性  
一般の業務クライアントには、PC/Windowsシステムが採用されているため、多くの作業にとって、Windowsシステムでの操作がより慣れたものとなっています。このような場合、開発作業もUNIX系システム上での作業より、Windows上での作業の方が効率的になります。
- 豊富な開発ツール群の使用が可能  
COBOLプログラムの開発・テスト・保守等に使用される製品の一部は、Windowsシステム上の製品しか存在しない、あるいは大きな機能差が存在する場合があります。  
分散開発を適用した場合、これらの機能差はWindowsシステム上の製品によって補うことができます。

### デメリット

- 製品のライセンスが余分に必要  
分散開発を行うためには、NetCOBOL製品の開発系製品が、UNIX系システム側とWindowsシステム側の両方に必要となるため、それぞれに製品ライセンスを購入する必要があります。また、ミドルウェア製品(例えばデータベース)と連携するプログラムの単体テストまでWindowsシステムで実施するような場合、Windowsシステムにも連携するミドルウェア製品のライセンスが必要になります。
- 開発プラットフォームの違いによる機能差  
UNIX系システムとWindows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、さまざまな箇所で細かな違いがあります。NetCOBOLでは、COBOL言語の機能内では、これらの違いを意識しないための仕組みを用意していますが、完全に同じ動作を期待できない場合もあります。  
また、オペレーティングシステムそのものによって提供される機能を使用しているため、

まったく使用できない機能も存在します。

これらの詳細については“1.2.3 [分散開発の適用範囲](#)”で詳しく説明します。

## 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生かし、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。

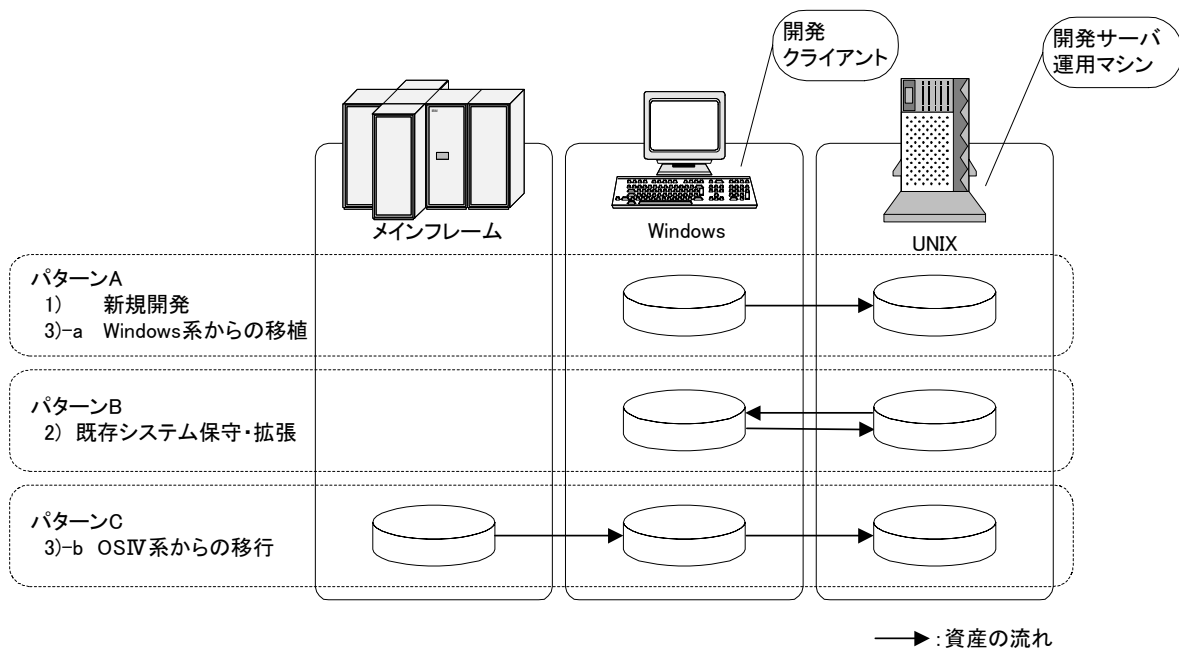
### 1.2.3.1 開発の目的から見た分散開発の適用範囲

分散開発の対象となるプログラムの開発目的として、大きく次の4つが考えられます。

1. UNIX系プログラムの新規開発
2. 既存のUNIX系プログラムの保守・機能拡張
3. 他システムで動作する既存プログラムのUNIX系システムへの移植
  - a) Windowsシステムからの移植
  - b) OSIV系システム(富士通製メインフレーム)からの移植

ただし、実際に分散開発を適用する場合の主なプログラム資産の流れを考えると、これは次の図のような3つのパターンに分類できます。ここでは、この3つのパターンに対して、どのように分散開発を適用すべきかを説明します。

図1-5 開発目的別の主なプログラム資産の流れ



#### パターンA

新規にUNIX系システムのプログラムを開発する、あるいは既にWindowsシステムで稼働しているCOBOLプログラム(NetCOBOLで開発したもの)をUNIX系システムに移植するために、分散開発を適用する場合です。

NetCOBOLの提供する分散開発支援機能は、このパターンでの開発を想定して提供されているため、このマニュアルで説明する手順をそのまま適用できます。以下に、その概要を説明します。

##### ● プログラミング(ソース、登録集、定義体・オーバーレイ)

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



— オーバレイパターン

この際、作成・更新するプログラム資産をNetCOBOLのプロジェクトマネージャで作成するプロジェクトに登録します。この登録情報を元にUNIX系システムでの翻訳・リンク用の制御文(Makefile)を生成することができます。

⇒ 3.2 [プロジェクトの作成](#)

⇒ 3.4 [プログラミング](#)

● 翻訳・リンク(構文チェック)

NetCOBOLを使用して、Windowsシステムで作成・更新したプログラム資産を翻訳・リンクします。この作業は、次のような目的で行います。

- 作成したプログラム資産に誤りや矛盾がないことをまず確認する。
- プロジェクトマネージャに登録したプログラム資産の依存関係をチェックする。
- 単体テスト用の実行形式プログラムを作成する。

⇒ 3.5 [翻訳チェックとリンク](#)

● 単体テスト

Windows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。NetCOBOLの提供するデバッグ機能(CHECK、COUNT、TRACE)と対話型デバッガを使用して、Windowsシステム上でプログラムの誤りを発見することができます。

⇒ 3.6 [Windowsクライアントでの単体テスト](#)

● 翻訳・リンク(ターゲット翻訳)

Windowsシステムで翻訳・リンクしたプログラムは、UNIX系システムでは動作しません。このため、Windowsシステムで作成・更新したプログラム資産をUNIX系システムに転送して、UNIX系システムのNetCOBOLを用いて、改めて翻訳・リンクします。このために必要となる以下の操作は、NetCOBOLのプロジェクトマネージャの操作によって可能です。

- UNIX系システムへのプログラム資産の転送
- UNIX系システム上での翻訳・リンクのための制御文(Makefile)の生成
- UNIX系システム上での翻訳・リンクの実行

⇒ 4.1 [サーバ環境へのプログラム資産の登録](#)

⇒ 4.2 [ビルド制御文生成機能](#)

⇒ 4.3 [ターゲットビルド](#)

● 結合テスト以降

UNIX系システム上で翻訳・リンクし直したプログラムによるテストを実施します。

この際、Windowsシステム上の対話型デバッガを使用して、UNIX系システム上のプログラムをリモートデバッグすることができます。

⇒ 4.4 [リモートデバッグ](#)

対象となるプログラムが分散開発の適用に不向きなCOBOL言語の機能を使用していない限り、全工程に分散開発を適用することが、作業の効率化につながります。

## パターンB

既にUNIX系システムで稼働しているCOBOLプログラム保守および機能拡張のために分散開発を適用する場合は、

この場合でも、次の作業を先立って行うことだけで、パターンAで示した各作業を実施することは可能です。

● プログラム資産のWindowsシステムへの移行

UNIX系システム上に存在するCOBOLソースを始めとする各種プログラム資産を、Windows-PC上に移行します。

- COBOLソースプログラム
- COBOL登録集原文(COPY句)
- 画面帳票定義体
- オーバレイパターン

NetCOBOLのプロジェクトマネージャで空のプロジェクトファイルを作成し、UNIX系の分散開発の設定を行うと、プロジェクトマネージャを使用してUNIX系システムからファイルを受信することができるようになります。その後、受信したプログラム資産をプロジェクトに登録します。

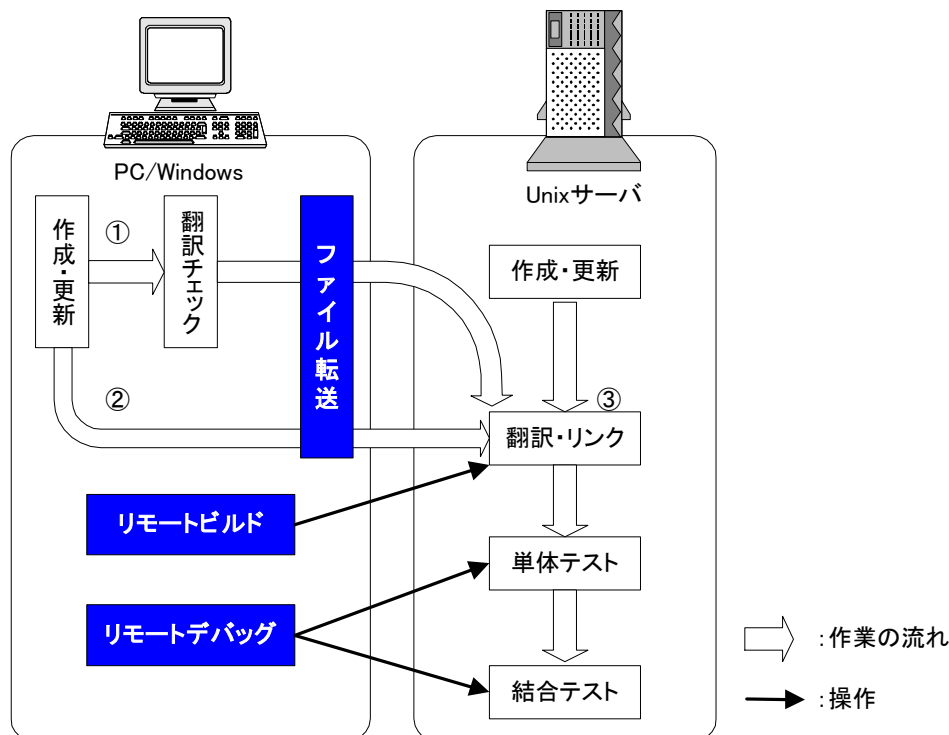
### ⇒ 3.3.2 [UNIX系システムからのプログラミング資産の移行](#)

しかし、このような作業の進め方による開発作業の効率化が期待できるのは、複数のプログラム資産にまたがる規模の大きな修正を実施する場合だけです。

修正の範囲が限定される、あるいは規模が小さい場合は、プログラム資産のWindowsシステムへの移行とプロジェクトファイルを作成する作業は無駄になります。むしろ、開発環境は基本的にUNIX系システム上に既に存在するものを使用し、特定の作業に限定して、分散開発を適用するほうが、効率的です。

以下にその考え方を図で示します。

図1-6 既存システムの保守・拡張時の分散開発の部分的適用



リモートビルドやリモートデバッグなどの機能の有用性は、いずれの場合も変わりません。また、UNIX系システム側で作成したプログラムのデバッグにのみ、分散開発を適用することも可能です。COBOLソース・登録集などのプログラム資産をWindowsシステム側に転送して、適切に配置することでリモートデバッグが可能となります。リモートデバッグ時のプログラム資産の配置については“表4-8 [リモートデバッグ時の資産格納場所](#)”を参照してください。

## パターンC

OS/IV系システムで稼働しているCOBOLプログラムをUNIX系システムに移植する際に分散開発を適用する場合です。

この場合でも、次の作業を先立って行うことだけで、パターンAで示した各作業を実施することは可能です。

### ● プログラム資産のWindowsシステムへの移行

OS/IV系システム上に存在するCOBOLソースを始めとする各種プログラム資産を、

Windows-PC上に移行します。

- COBOLソースプログラム
- COBOL登録集原文(COPY句)
- フォーマット定義体(オープン系システムの画面帳票定義体に相当)
- オーバレイ定義体

NetCOBOLのプロジェクトマネージャで空のプロジェクトファイルを作成し、OSIV系の分散開発の設定を行うと、プロジェクトマネージャを使用してOSIVシステムからファイルを受信することができるようになります(ただし、定義体類は移行に先立ってファイルの形式を変換する必要があります)。その後、受信したプログラム資産をプロジェクトに登録します。

⇒ 3.3.1 [OSIVからのプログラミング資産の移行](#)

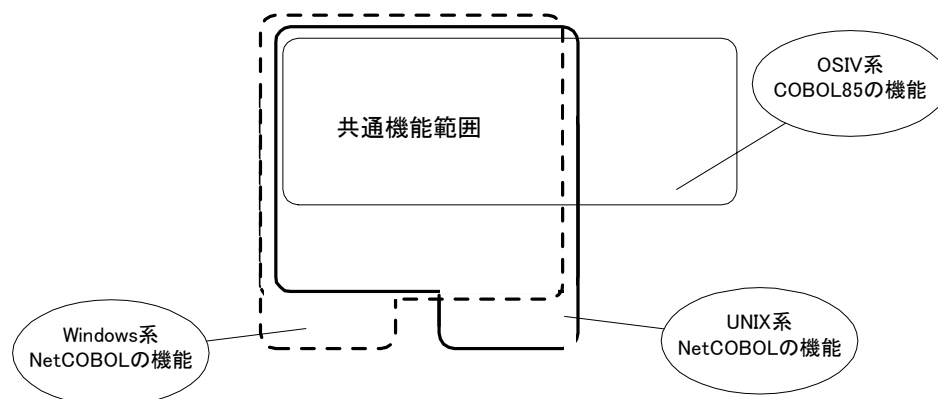
⇒ OSIV分散開発の手引き

このような作業の進め方は、プログラム資産およびその開発環境の移行を次の2段階で行うことに相当します。

4. OSIV系システムからWindowsシステムへの移行
5. WindowsシステムからUNIX系システムへの移行

あえて、このように2つの段階を持って、移行を実施することは無意味にも見えますが、このような手順を取る方が、UNIX系システムに直接移行するよりも効率的な開発が可能になります。これは、OSIV系システムのCOBOL85とNetCOBOLの機能差が、各プラットフォーム向けのNetCOBOL製品間の機能差よりも遥かに大きいからです。

図1-7 各プラットフォーム向けのCOBOLの機能差概観



### 1.2.3.2 COBOLの機能範囲から見た分散開発の適用範囲

UNIX系システムとWindowsシステムのNetCOBOLの機能範囲を“図1-8 [UNIX系システムとWindowsシステムの機能範囲](#)”に示します。

以降では、“図1-8 [UNIX系システムとWindowsシステムの機能範囲](#)”に従って、各機能範囲に対し分散開発が適用可能かを説明します。

図1-8 UNIX系システムとWindowsシステムの機能範囲

【Windows システムの機能範囲】		【UNIX 系システムの機能範囲】	
《国際規格 COBOL85》		《国際規格 COBOL2002》	
— 中核	***	— オブジェクト指向機能	***
— 順ファイル	***	— 利用者定義型	***
— 相対ファイル	***	— Unicodeサポート	*
— 索引ファイル	***	— 2進データ型(BINARY-SHORT 等)	***
— プログラム間連絡	***	《Micro Focus 互換仕様》	
— 整列併合	***	— スクリーン操作	**
— 原始文操作	***	— 名前付き定数(78レベル)	***
— 報告書作成	***	— 16進数字定数	***
— デバッグ	***	《富士通拡張仕様》	
— 区分化	***	— 表示ファイル(画面, 帳票)	*
— データベース操作(SQL)	*	— 表示ファイル(APL, ACM)	**
《XPG仕様》		— 日本語プログラミング	***
— 行順ファイル	***	— 日本語処理	*
— C言語間結合	***	— 拡張日本語印刷	*
— (値渡し、復帰値)		— ビット操作	***
— ファイル共用/レコード排他	***	— 浮動小数点数操作	***
— スクリーン操作	**	— FORMAT 句付き印刷ファイル	***
— コマンド行引数操作	***	— (順ファイル)	
— 環境変数操作	***	— 拡張索引ファイル	***
— 連結式	***	— (多重キー、逆順検索)	
《国際規格 1989 年追補》		— 定数節	***
— 組み込み関数	***	— システムプログラム記述(SD)	***
		— マルチスレッド対応	***
		— Web連携機能	*
		— COM連携機能	

#### 記号の説明

- \*\*\* : 共通機能範囲
- \*\* : Linux 版に機能差あり
- \* : Windows-UNIX 系間で機能差あり

#### 共通仕様範囲の機能

“図1-8 [UNIX系システムとWindowsシステムの機能範囲](#)”で“\*\*\*”を付けて表示した機能は、各オペレーティングシステム用のNetCOBOL製品間で、共通の仕様を持ちます。これらの機能のみを使用しているプログラムには、最大限に分散開発が適用可能です。

通常のWindows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を、Windowsシステム上で可能です。

### Linux版に機能差がある機能

“図1-8 [UNIX系システムとWindowsシステムの機能範囲](#)”で“\*\*”を付けて表示した機能は、Linux版のNetCOBOL製品では提供されていません。これらの機能を使用し、Linux上で動作するプログラムは作成することはできません。

Solaris版のNetCOBOL製品では、共通仕様範囲と同様に、分散開発が適用可能です。

### Windows-UNIX系間で機能差がある機能

“図1-8 [UNIX系システムとWindowsシステムの機能範囲](#)”で“\*”を付けて表示した機能は、Windows-UNIX系間で機能差があります(Solaris版/Linux版の間でも機能差を持つ場合があります)。機能差の原因はさまざまですが、その現れ方は次のいずれかです。

1. ソース記述上の違い
2. 実行結果の違い
3. 組み合わせ可能なミドルウェア製品に依存した機能差

以下、それぞれの場合について分散開発がどこまで適用可能となるかを説明します。

#### ソース記述上の違い

“図1-8 [UNIX系システムとWindowsシステムの機能範囲](#)”における次のような機能が、これに当たります。

- Unicodeサポート

Unicodeサポート機能は、NetCOBOLのWindowsシステム用の製品とUNIX系システム用の製品の間に仕様の違い(詳細については“付録A [NetCOBOL製品の相違点](#)”を参照)が存在します。特にWindowsシステム用のNetCOBOLはソースプログラムやCOBOL登録集を作成する際にシフトJISコードを使用する必要があり、Unicode固有文字をソースプログラムやCOBOL登録集の中で使用できません(定数の値としては、日本語16進文字定数を使用することで使用可能です)。このため、無条件にソースレベルの互換性を保つことができません。

対応方法としては、次のいずれかが考えられます。

- a. ソースレベルでの互換性を保てるように注意してプログラムを作成し、全面的に分散開発を適用する。
- b. プログラム資産の作成はUNIX系システムで実施し、UNIX系システム上でのプログラムの翻訳・リンクおよびテストなどに限定して分散開発を適用する。

WindowsシステムとUNIX系システムの両方で動作するプログラムを作成するのでない限り、bの方法がお勧めです。

#### 実行結果の違い

“図1-8 [UNIX系システムとWindowsシステムの機能範囲](#)”における次のような機能が、これに当たります。

- 表示ファイル機能(画面・帳票)
- 日本語処理
- 拡張日本語印刷

これらの機能については、ソースレベルでの互換性はありますが、実行時の動作については完全に互換性が保証される訳ではありません(詳細は“付録A [NetCOBOL製品の相違点](#)”を参照)。

これらの機能を使用する場合、単体テストをWindowsシステムで実施することはあまり意味を持ちません。

#### 組み合わせ可能なミドルウェア製品に依存した機能差

“図1-8 [UNIX系システムとWindowsシステムの機能範囲](#)”における次のような機能が、これに当たります。

- データベース操作(SQL)
- Web連携機能

これらの機能は、COBOL言語の機能だけでは実現できないため、他のミドルウェア製品が必要となりますが、組み合わせ可能なミドルウェア製品（データベース、Webサーバ等）の有無・連携方法の違いから機能差が生じます。

次のような条件を満たすなら、共通範囲と同じように分散開発をすることができます。

- a. WindowsシステムとUNIX系システムで同じミドルウェア製品が組み合わせ可能である。
- b. データベース製品であれば、接続方法についても同じ方法を使用できる。

ただし、Windowsシステムでこれらのミドルウェア製品と連携するプログラムの単体テストを実施する場合、これらのWindows版製品のライセンスが必要になってきます。

### 1.2.3.3 開発ツール群の整備状況から見た分散開発の適用範囲

NetCOBOLのUNIX系システム用製品とWindowsシステム用の製品の提供するCOBOL言語の機能差は“図1-8 [UNIX系システムとWindowsシステムの機能範囲](#)”で示したとおり、大きなものではありません。

しかし、開発系製品で提供する開発用の各種ツールの機能差は大きなものがあります。“表1-2 [NetCOBOL開発系製品のコンポーネントの機能差](#)”にUNIX系の各開発製品で提供される開発用のコンポーネントの機能差を示します。

なお、表中の①～③はそれぞれ、次の製品を表します。

①：Solaris版製品

②：Linux32版製品

③：Linux64版製品

表1-2 NetCOBOL開発系製品のコンポーネントの機能差

コンポーネント名	機能概要	各製品の状況			備考
		①	②	③	
FORM/PowerFORM	画面・帳票定義体、オーバーレイパターンの作成	×	×	×	
プロジェクトマネージャ	統合開発環境	△	×	×	Solaris版はX-Windows環境があれば使用可能。 その他、コマンド形式の開発支援ツールが存在。
SIMPLIA	保守用ドキュメント作成 テスト支援・データ移行	△	×	×	Solaris版はデータ移行機能のみを提供。
PowerGEM Plus	資産管理・構成管理	△	×	×	Solaris版は資産管理機能のみを提供。
対話型デバッガ	デバッグ	△	△	×	Solaris版はX-Windows環境があれば、GUIを使用した対話型のデバッグが可能、X-Windows環境がない場合はラインモードでのデバッグのみとなる。 Linux32版では、一部の機能が制限されるラインモードでのデバッグが可能。 Linux64版では、COBOLデバッガは提供されないが、gdbによるデバッグが可能

表中の記号の意味：

△：Windows版と機能差はあるが提供

×：未提供

分散開発を適用することで、UNIX系のNetCOBOLの開発製品で提供されるコンポーネントの機能差を補うことができます。

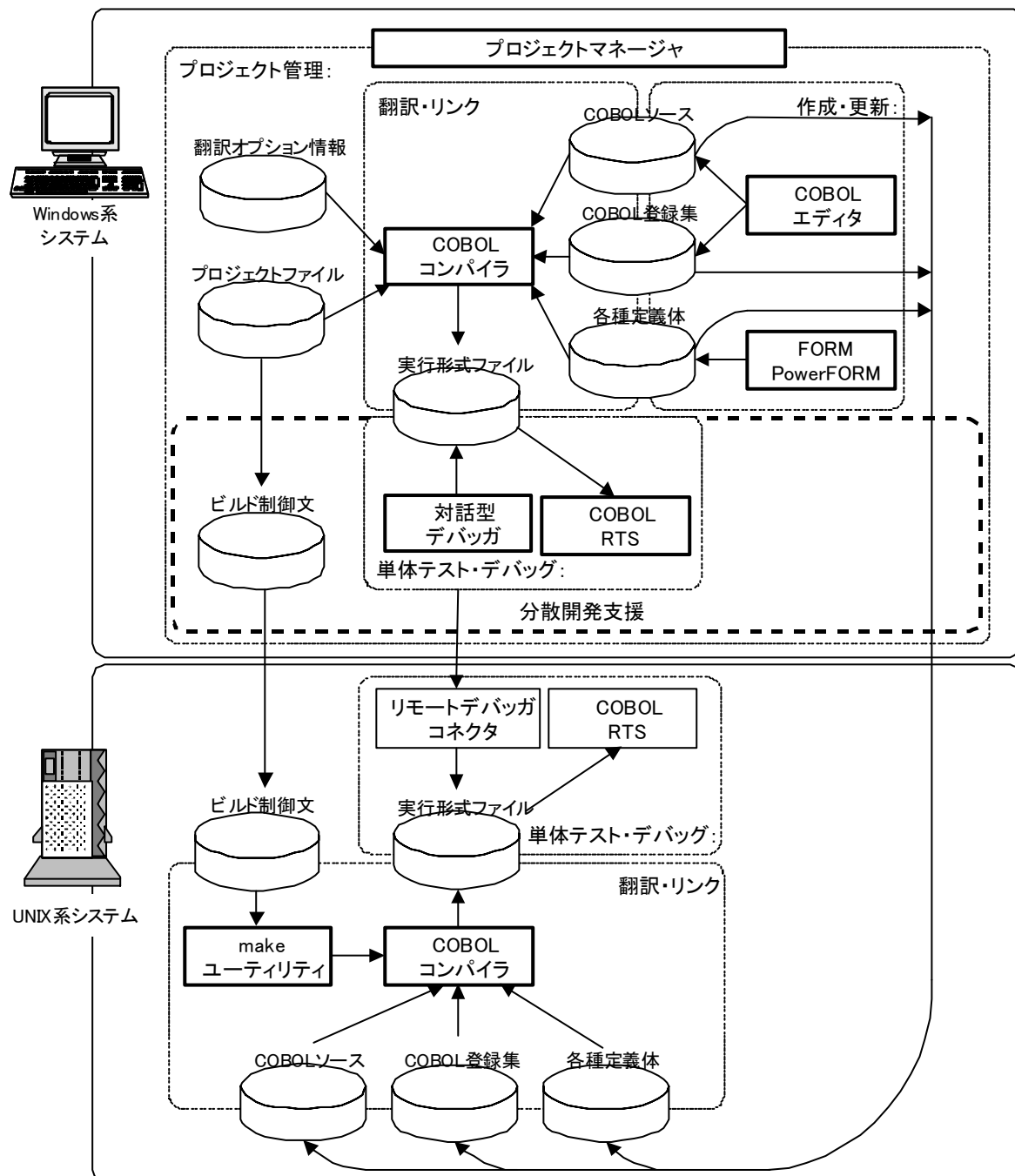
## 1.3 UNIX系プログラムの分散開発環境概要

ここではNetCOBOLを用いて、UNIX系プログラムの分散開発を実施する場合のシステム構成と必要ソフトウェアについて説明します。

### 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

### UNIX分散開発に利用可能なNetCOBOL製品組合せ

以下にUNIX分散開発を行うために必要なNetCOBOL製品の組合せを示します。

表1-3 NetCOBOL製品の組合せと分散開発のサポート

Windows製品		Base Edition	Standard Edition	Professional Edition
UNIX系製品				
Solaris	Base Edition	×	×	×
	Standard Edition	×	×	×
	Professional Edition	○	○	○
Linux32	Base Edition	×	×	×
	Standard Edition	○	○	○
Linux64	Enterprise Edition	○	○	○

表中の記号の意味：

○：分散開発をサポートする

×：分散開発をサポートしない

### UNIX分散開発に使用されるソフトウェア製品・コンポーネントの一覧

UNIX系プログラムをWindowsシステム上で分散開発するために、UNIX系システム上およびWindowsシステム上に必要となるソフトウェア製品・コンポーネントの一覧を“表1-4 [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1-4 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	Windowsシステム	UNIX系システム
分散開発基盤		
文字コード変換	ADJUSTまたはCharset Manager(*1)	Charset Manager(*1)
UNIXサーバ連携	Windows TCP/IPサービス(*2)	telnetd/ftpd/rexec(*3)
	プロジェクトマネージャ(*5)	サーバ側分散開発マネージャ(*4)
開発系		
開発言語(COBOL)	NetCOBOL(*5)	NetCOBOL(*5)
開発環境	プロジェクトマネージャ(*5)	makeユーティリティ
ソース等の編集	COBOLエディタ(*5)	viその他システムに付属のエディタ
プログラムのデバッグ	COBOLデバッガ(*5)	COBOLデバッガ(*5)(*8)
画面、帳票などの設計	FORM/PowerFORM(*6)	なし
オーバレイパターンの作成	FORM/PowerFORM(*6)	なし
画面、帳票表示、出力	MeFt(*6)	MeFt(*6)
資産管理	PowerGEM Plus(*7) (+ PowerGEM Plus Administrator)	PowerGEM(*7)
実行基盤		
データベース	Symfoware Server他	Symfoware Server他
通信制御、 トランザクション管理	Interstageファミリー群	Interstageファミリー群

\*1 外字を使用する場合など、システムの提供するコード変換機能では不十分な場合に必要となります。

\*2 Windows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 UNIX系システムが提供するサービスです。詳細については“第2章 [分散開発環境の構築](#)”で説明します。
- \*4 各UNIX系システム向けのNetCOBOL製品で分散開発をサポートするEditionに含まれます。
- \*5 各プラットフォーム向けのNetCOBOL製品のBase Edition以上に含まれます。
- \*6 各プラットフォーム向けのNetCOBOL製品のStandard Edition以上に含まれます。
- \*7 Windows版およびSolaris版のNetCOBOL製品のProfessional Editionに含まれます。
- \*8 Linux64版のNetCOBOL製品には含まれません。

---

## 第2章 分散開発環境の構築

---

本章では、UNIX系システムのアプリケーションの開発をWindows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 分散開発環境構築の前提知識

UNIX系プログラムの分散開発を効率よく行うには、適切な分散開発環境の構築が必要となります。この分散開発環境の構築にはUNIX系システムの知識がどうしても必要となります。そこでUNIX系プログラムの分散開発を効率よく行うためには、どのように分散開発環境を構築するかは、開発管理者がまとめて検討・決定し、個々の開発者はその指示に従って作業を進める必要があります。ここでは、まず分散開発環境構築のために前提となる次の知識を説明します。

- COBOLプロジェクトマネージャにおけるプロジェクト
- 分散開発時のUNIXサーバ上の環境
- リモートデバッグ機能の概要
- 資産管理方法と資産の配置

### 2.1.1 COBOLプロジェクトマネージャにおけるプロジェクト



COBOLプロジェクトマネージャにおけるプロジェクトとは、1つ以上のCOBOLプログラムを翻訳・リンクして、実行可能ファイルまたは共用ライブラリを作成するために必要となる資産の集合です。

実際にプロジェクトを作成・使用するための詳しい説明は、次章以降で行います。ここでは、分散開発の単位としてのプロジェクトの概念のみを説明します。

プロジェクトに含まれる資産は、大きく2つに分けられます。

- プロジェクト資産  
プログラム資産から、COBOLプログラムを作成するための各種の設定が保存されているものです。
- プログラム資産  
開発対象のCOBOLプログラムそのものを構成する資産です。COBOLソースや登録集のように翻訳・リンク時に必要とされるものから、フォームオーバーレイやhtml文書など実行時に必要な資産も含まれます。

以下にプロジェクトを構成する各プロジェクト資産の一覧を示します。

表2-1 Windowsクライアント側のプロジェクト構成ファイル

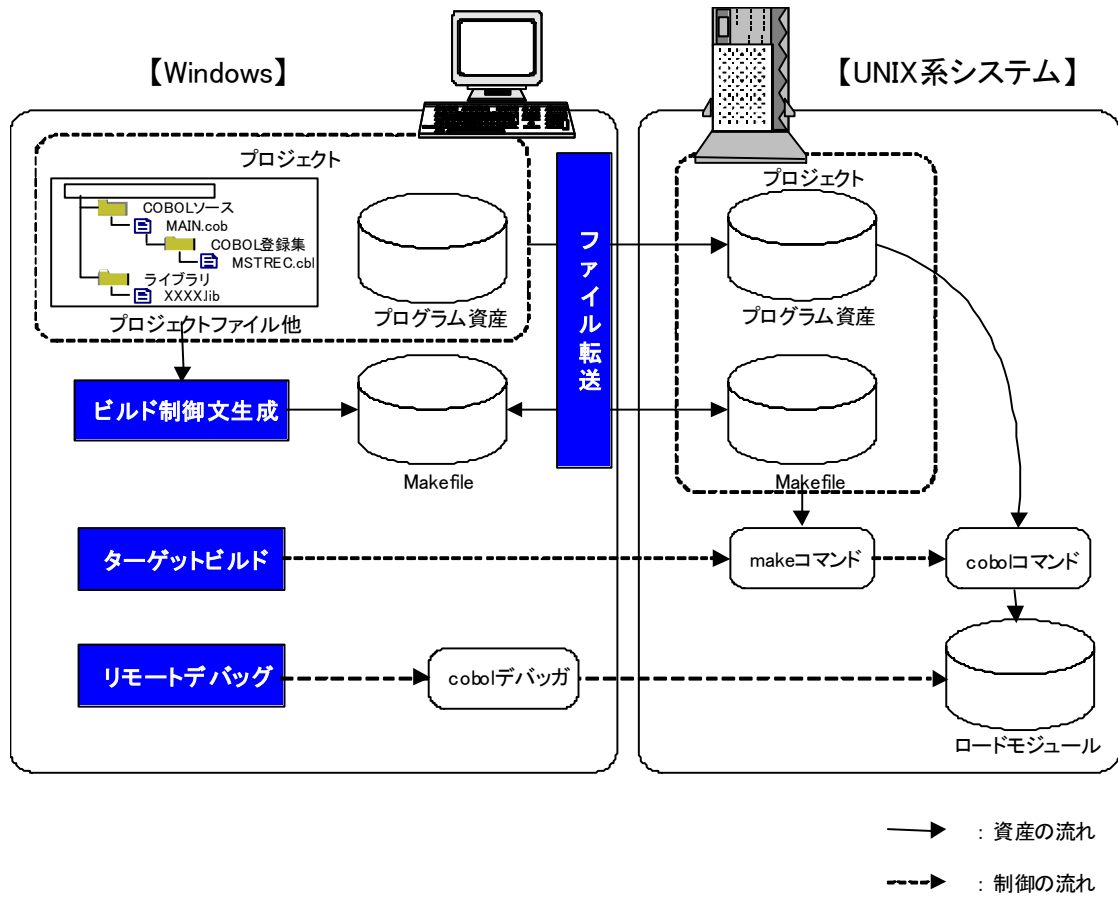
資産の種別	ファイルの種別
プロジェクト資産	プロジェクトファイル
	翻訳オプションファイル
	リンクオプションファイル
プログラム資産	COBOLソースプログラム
	COBOL登録集原文(COPY句)
	画面帳票定義体
	ファイル定義体
	フォームオーバーレイパターン
	プリコンパイラソース
	IDLソース
	html文書

UNIX系プログラムの分散開発を支援するために提供される次のような機能は、基本的にこのプロジェクトを単位として行われます。

- プログラム資産のUNIXサーバへの送受信
- ビルド制御文(Makefile)生成

- ターゲットビルド
- リモートデバッグ

図2-1 プロジェクトと分散開発機能



プログラム資産はファイル単位でUNIXサーバ側に送受信を行います。プロジェクト資産はビルド制御文(Makefile)に変換して、UNIXサーバ側に送信することになります。したがって、“表2-1 [Windowsクライアント側のプロジェクト構成ファイル](#)” に示した“Windowsクライアント側のプロジェクト構成ファイル” に対応する“UNIXサーバ側のプロジェクト構成ファイル” は次のようになります。

表2-2 UNIXサーバ側のプロジェクト構成ファイル

資産の種別	ファイルの種別
プロジェクト資産	ビルド制御文ファイル
プログラム資産	COBOLソースプログラム
	COBOL登録集原文 (COPY句)
	画面帳票定義体
	ファイル定義体
	フォームオーバーレイパターン
	プリコンパイラソース
	IDLソース
	html文書

NetCOBOLにおけるプロジェクトは、Windowsクライアント側におけるプロジェクトもUNIXサーバ側におけるプロジェクトも基本的に1人の開発者による開発の単位でもあります。したがって、開発対象プログラムが大規模で、かつ、多数のプログラム資産からなるような場合、

それをすべて登録した1つのプロジェクトとするのではなく、何らかの単位で分割すべきです。

## 2.1.2 分散開発時のUNIXサーバ上の環境

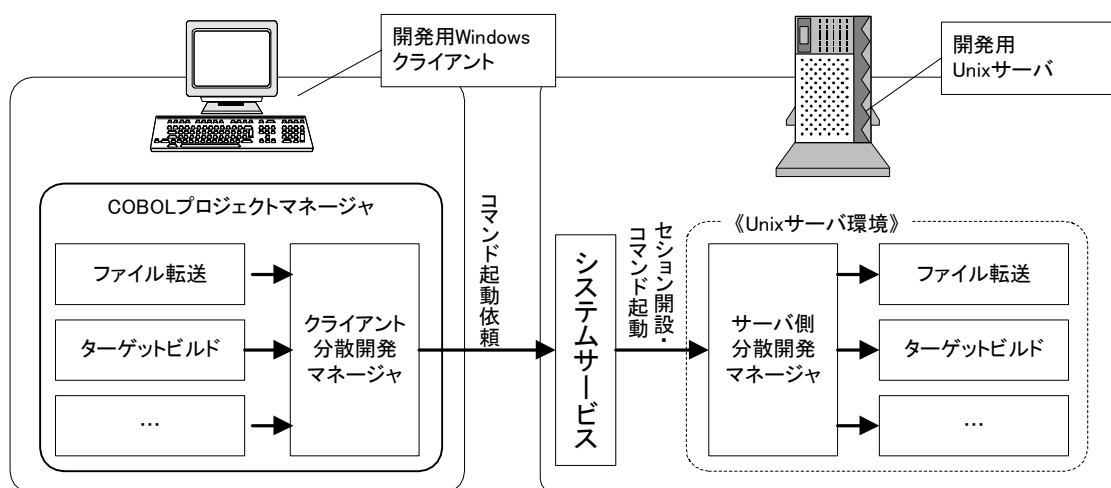


UNIX系サーバ側で実行される操作は、環境変数を始めとするUNIX系サーバ側の環境設定に影響を受けます。このため、各作業に必要な環境変数を適切に設定する必要があります。

リモートデバッガによるCOBOLプログラムのデバッグ時に有効となる環境変数の設定は、リモートデバッグの方法によって異なるので“2.1.3 [リモートデバッグの概要](#)”の中で説明します。ここでは、プログラム資産の送受信からターゲットビルドに至る過程で有効となる環境変数の設定について説明します。

プログラム資産の送受信からターゲットビルドに至る操作は、実際は次の図に示すような流れで行われています。

図2-2 UNIX連携機能の概要



有効となる環境変数は、WindowsクライアントからUNIXサーバに接続する際に用いたユーザのログイン環境で設定されるものです。

表2-3 UNIXサーバ環境の設定ファイル

システム	ログインシェル	設定ファイル
Solaris	csh使用时	.cshrcまたは.cshrcから呼び出すシェルスクリプト
Linux	bash使用时	.bashrcまたは.bashrcから呼び出すシェルスクリプト
	csh(tcsh)使用时	.cshrcまたは.cshrcから呼び出すシェルスクリプト

このため、プログラム資産の送受信からターゲットビルドに至る操作で必要となる次のような環境変数は、予め分散開発に使用するユーザのログイン環境に設定しておく必要があります。

- COBOL資産の送受信時に行うコード変換プログラムのための環境変数
- COBOLソースの翻訳のための環境変数
- COBOLプログラムのリンクのための環境変数

## 2.1.3 リモートデバッグの概要



リモートデバッグは、ネットワーク上の別のコンピュータで動作しているプログラムをデバッグする機能です。

UNIX系プログラムの分散開発では、次の構成でリモートデバッグを行うことになります。

- UNIX系サーバ

COBOLプログラムの動作しているコンピュータ

- Windowsクライアント  
COBOLデバッガのウィンドウが表示されるコンピュータ

ここでは、上記の構成でのリモートデバッグの機能の概要とそのためにUNIXサーバ側に設定が必要となるリモートデバッガコネクタについて説明します。



Linux64版ではリモートデバッガを使用できません。

### 2.1.3.1 リモートデバッグの2つの方式

リモートデバッグには、デバッグ対象のプログラムとデバッガの関係から、2つの方式が存在します。ここでは、それぞれについて次のような点についての概要を説明します。

- リモートデバッガ使用前の準備
- デバッグの開始方法
- デバッグ時に有効となる環境変数の設定

#### 一般形式

COBOLプログラムをリモートデバッグする通常の方法です。

#### リモートデバッガ使用前の準備

デバッグ対象のCOBOLプログラムが動作するUNIXサーバ側で、サーバ側のリモートデバッガコネクタが起動されている必要があります。

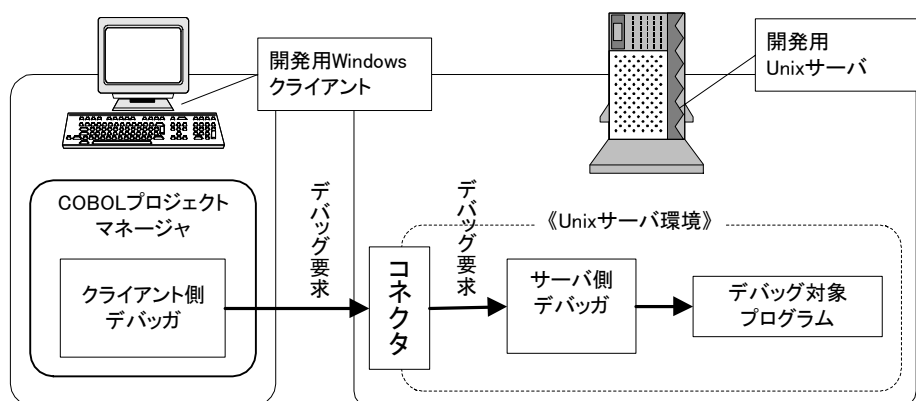
サーバ側のリモートデバッガコネクタの起動については、“2.1.3.2 [リモートデバッグのためのサーバ側の環境設定](#)”で詳細に説明します。

#### デバッグの開始方法

Windowsクライアント側のリモートデバッガからデバッグ対象プログラムを起動します。

実際のUNIXサーバ側でのデバッグは、Windowsクライアント側のデバッグ要求をUNIXサーバ上のデバッガに中継するためのリモートデバッガコネクタというプログラムを介して次のように行われます。

図2-3 一般形式のリモートデバッグ



#### デバッグ時に有効となる環境変数の設定

サーバ側のデバッガとそれに起動されるデバッグ対象プログラムに有効となる環境変数は、リモートデバッガコネクタが起動された際に有効であったものがそのまま引き継がれます。

同じリモートデバッガコネクタを使用してデバッグするプログラムに有効な環境変数は皆同じものになってしまうため、COBOLプログラムの実行時に必要な環境変数をリモートデバッガコネクタ起動時の環境変数として与えることは、次のような不都合を生じます。

- デバッグ対象プログラムを変更するごとにリモートデバッガコネクタの停止・環境変数の再設定・リモートデバッガコネクタの再起動を繰り返すことが必要となる。
- 予めデバッグ対象のプログラムすべてが必要とする環境変数を設定した上で、リモートデバッガコネクタ起動することが必要となる。

したがって、リモートデバッグ時にデバッグ対象のプログラムが参照する環境変数は別の形で与えることが必要です。

これには、次の機能が使用できます。

1. リモートデバッガによって、デバッグ対象のCOBOLプログラムを起動する際、実行時パラメタを与えることができる。
2. COBOLプログラムは、必要な環境変数の設定を記述したファイルの名前(初期化ファイル)を実行時パラメタとして受け取ることができる。

したがって、一般形式でリモートデバッグする際の環境変数の設定は次のように行うべきです。

- a. どのプログラムをデバッグする場合でも、常に必要で同じ値を持つ環境変数をリモートデバッガコネクタの起動時の環境変数として設定する。
- b. プログラムごとに必要となる環境変数は、プログラムごとに用意する初期化ファイルに指定し、リモートデバッグの開始時に与える実行時パラメタでこの初期化ファイル名を指定する。

### アタッチ形式

一般形式でデバッグを行う場合、デバッガの起動画面からデバッグ対象プログラムを含む、または呼び出す実行可能プログラムを指定する必要があります。しかし、システムのサービスとして起動されるようなプログラム(以降の説明ではサーバプログラムと呼びます)から、デバッグ対象となるCOBOLプログラムが呼び出されるような場合、これができません。アタッチ形式のデバッグをそのような場合に使用します。

次のようなCOBOLプログラムが該当します。

- Webサーバ配下のCOBOLプログラム
- Interstage配下のCOBOLプログラム

### リモートデバッガ使用前の準備

リモートデバッグを行うWindowsクライアントで、クライアント側のリモートデバッガコネクタが起動されている必要があります。

クライアント側のリモートデバッガコネクタの起動については、“第4章 [サーバ環境での開発作業](#)”で詳細に説明します。

また、UNIXサーバ側にリモートデバッグを行うWindowsクライアントを指定するための設定が必要です。これについては、“2.1.3.2 [リモートデバッグのためのサーバ側の環境設定](#)”で説明します。

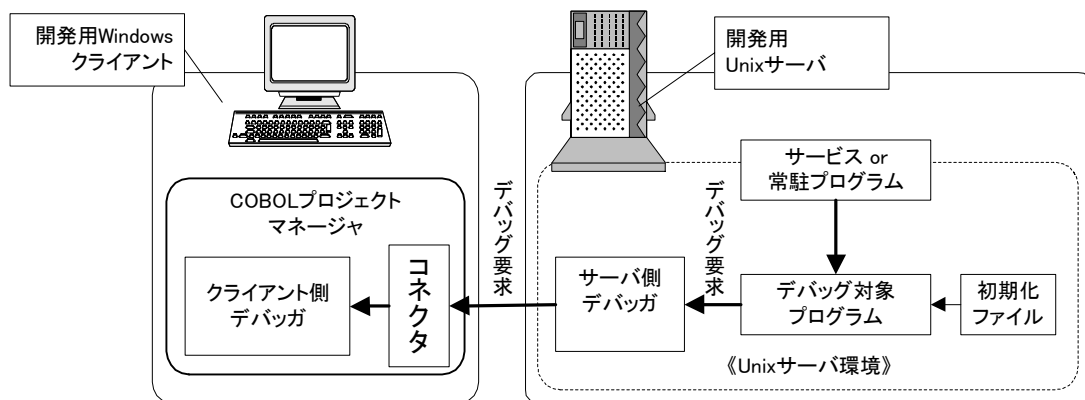
### デバッグの開始方法

デバッグ対象のCOBOLプログラムが呼び出された際に、COBOLプログラムからWindowsクライアント側のリモートデバッガを起動する形でデバッグを行います。

Windowsクライアント側のリモートデバッガの起動は、UNIXサーバ側のデバッガのデバッグ要求を、Windowsクライアント側のリモートデバッガに中継するためのリモートデバッガコネクタというプログラムを介して次のように行われます。



図2-4 アタッチ形式のリモートデバッグ



### デバッグ時に有効となる環境変数の設定

サーバ側のデバッガとそれを起動するデバッグ対象プログラムに有効となる環境変数は、COBOLの実行環境が次のようにして、決定したものです。

1. サーバプログラムを起動した際の環境変数の設定を引き継ぐ。
2. サーバプログラムから最初に呼び出されるCOBOLプログラムに対して、初期化ファイルが用意されていないなら、1で引き継いだものがそのまま有効となる。
3. サーバプログラムから最初に呼び出されるCOBOLプログラムに対して、初期化ファイルが用意されているなら、初期化ファイルの内容でサーバプログラムを起動した際の環境変数の設定を上書きする。同じ環境変数に対して、サーバプログラムの起動時の指定と初期化ファイルの指定で異なる値が指定されているなら、初期化ファイルで指定した環境変数の値が有効となる。

上記のように決定された環境変数の情報は、サーバプログラムが終了するまで有効です。

同じサーバ配下で動作する複数のCOBOLプログラムの中で、それぞれ異なる環境変数の値を指定することはできません。

### 2.1.3.2 リモートデバッグのためのサーバ側の環境設定

ここでは、リモートデバッグを行うために、UNIXサーバ側で実施する必要のある環境設定の詳細について説明します。

#### サーバ側リモートデバッガコネクタ

一般形式(Windowsクライアントのデバッガからプログラムを起動)でリモートデバッグを行う場合に、Windowsクライアントからのリモートデバッグ要求を受け取るために起動します。

#### 起動方法

サーバ側リモートデバッガコネクタは、UNIXサーバ上で起動され、Windowsクライアント側からのリモートデバッグ要求を待ち受けるプログラムです。

もっとも単純には、telnet端末などでUNIXサーバに接続して、次のようにして起動します（以降の例では、UNIXサーバのIPアドレスは“192.168.0.103”であるものとして説明します）。

図2-5 サーバ側リモートデバッガコネクタ起動例(1)

```
# svdrds
192.168.0.103:59998
Press Ctrl+C To End
```

起動時に表示されるメッセージは、Windowsクライアントからのリモートデバッグ要求を持ち受けるIPアドレスとポート番号を示しています(ポート番号59998はデフォルト値です)。

この状態で、このネットワークに接続されているすべてのWindowsクライアントからのリモート

デバッグ要求を受付、UNIXサーバ側の複数の異なるプログラムを同時にリモートデバッグをすることは可能です。しかし、このような使い方をお勧めできるのは、1人の開発者がリモートデバッグを行う場合、あるいは複数の開発者が同時にリモートデバッグを行わない場合だけです。複数プログラムのデバッグ中、次のような出力が、区別されることなくリモートデバッグコネクタを起動したtelnet端末に表示されるためです。

- 標準入力 (SYSIN) を宛先とするACCEPT文の入力
- 標準出力 (SYSOUT)、標準エラー出力 (SYSERR) を宛先とするDISPLAY文の出力
- 実行時のエラーメッセージ

複数の開発者が同時にリモートデバッグ機能を使用する場合、開発者毎に使用するポート番号を予め決めておいて、リモートデバッグ機能を使用する開発者がそれぞれ自分用にサーバ側のリモートデバッグコネクタを起動します。

以下にその例を示します。

図2-6 サーバ側リモートデバッグコネクタ起動例(2)

```
# svdrds -p 10000 -h client-1
192.168.0.3:10000
Press Ctrl+C To End
```

ここでは、サーバ側のリモートデバッグコネクタは次のような意味を持ちます。

- リモートデバッグ要求を受け付けるポート番号を10000番としている。
- リモートデバッグ要求を受け付けるWindowsクライアントをclient-1に制限している。

このように指定してサーバ側リモートデバッグコネクタを起動した場合、誤って他の開発者（例えば、client-2）がこのリモートデバッグコネクタにデバッグ要求を送ったとしても、その要求は拒否されます。

なお、リモートデバッグコネクタを起動する際の指定の詳細は続いて説明します。



#### 注意

次の理由から、UNIXサーバ側のリモートデバッグコネクタをシステムの起動時にシステムのサービスの様な形態で使用するべきではありません。

- 起動時のメッセージ以外にリモートデバッグコネクタの使用しているポート番号を調べることができない。
- リモートデバッグコネクタを終了させるためのコマンドが用意されていない。

### 起動形式の詳細

ここでは、サーバ側のリモートデバッグコネクタの起動形式を説明します。

```
svdrds [ポート指定] [接続制限指定]
```

起動時に次の形式でポート指定と接続制限指定をすることができます。

- ポート指定  
Windowsクライアント側からのリモートデバッグ要求を監視するためのポート番号を指定します。次の形式で指定します。

-p ポート番号

表2-4 ポート指定の内容

指定形式	内容
-p ポート番号	監視するポート番号を1024から65535の範囲の数字で指定します。

※ポート指定を省略した場合は、標準のポート(59998)を監視します。

● 接続制限指定

リモートデバッグを許可するWindowsクライアントを制限するための指定です。接続制限ファイルの内容については、“[接続制限と接続制限ファイル](#)”で説明します。

$$\left\{ \begin{array}{l} -h \text{ ホスト名} \\ -s \text{ 接続制限ファイル名} [-e] \end{array} \right\}$$

表2-5 接続制限指定の内容

指定形式	内容
-h ホスト名	接続を許可するホスト名またはIPアドレスを一つだけ指定します。
-s 接続制限ファイル名	接続制限ファイルを指定します。
-e	接続制限ファイルに指定された内容进行处理した結果、どのような接続制限が行われるようになったかを表示します。

※-s、-h共に省略した場合は、すべてのコンピュータからの接続を許可します。

### 接続制限と接続制限ファイル

接続制限とは、サーバ側のリモートデバッグコネクタがリモートデバッグ要求を受け付けるWindowsクライアントを制限する方法です。

ただ1つのWindowsクライアントにリモートデバッグを許すような場合は、サーバ側リモートデバッグコネクタ起動時に-hオプションで指定します。より複雑な指定が必要な場合は、接続制限ファイルを使用します。

接続制限ファイルは以下で説明する形式の指定を含むテキストファイルです。ファイル名として任意の名前を使用することができます。

### 接続制限ファイルの形式

接続制限を行う接続制限ファイルは、以下の形式で指定します。

```
-----
[ ALLOW={ ALL | 接続対象[接続対象...] } ]
[ DENY={ ALL | 接続対象[接続対象...] } ]
-----
```

- ALLOWに指定された接続対象からの接続を許可します。省略した場合はALLを指定したものとみなされます。
- DENYに指定された接続対象からの接続を拒否します。省略した場合はALLを指定したものとみなされます。
- ALLを指定した場合は、すべてのコンピュータからの接続を許可、または拒否します。
- ALLOWとDENYの指定が重複する場合、ALLOWの指定を記述していると、ALLOWの指定が優先されます。
- 複数行に渡って記述する場合は、末尾に¥を書きます。

例：

```
ALLOW=192.168.0.1 192.168.0.3 ¥
192.168.0.8-192.168.0.10
```

“表2-6 [接続対象の指定形式](#)”に接続対象の指定形式を示します。

表2-6 接続対象の指定形式

指定形式	指定例	IPアドレスの適用範囲例
IPアドレス指定	192.168.0.1	192.168.0.1
範囲指定	192.168.0.1-192.168.0.10	192.168.0.1 ～ 192.168.0.10

ワイルドカード	192.168.0.*	192.168.0.0 ～ 192.168.0.255
ホスト名	hostname	192.168.0.1

- ワイルドカードに指定できる“\*”は、各オクテッドに指定できます。
- オクテッドとは、10進数部分を“.”単位に区切った範囲です。
- IPアドレス192.168.0.1に対して、hostnameという名前のホスト名が割り当てられている場合の例です。

**注意**

なお、接続制限ファイルの内容に誤りがある場合は、接続制限ファイルによる指定が無効になり、すべてのコンピュータからの接続を許可します。

“表2-7 [接続制限の代表例](#)”に一般的な接続制限の方式とその際の接続制限ファイルの内容について示します。

表2-7 接続制限の代表例

接続制限	接続制限ファイルの内容
特定の接続対象のみを許可する。 接続を許可するIP アドレスが、以下の場合 － 192.168.0.1	ALLOW=192.168.0.1 DENY=ALL
複数の接続対象からの接続を許可する。 接続を許可するIP アドレスが以下の場合 － 192.168.0.1 － 192.168.0.2 － 192.168.0.10	ALLOW=192.168.0.1 ¥ 192.168.0.2 192.168.0.10 DENY=ALL
特定範囲の接続を許可する。 接続を許可するIP アドレスの範囲が以下の場合 下限：192.168.0.1 上限：192.168.0.10	ALLOW=192.168.0.1-192.168.0.10 DENY=ALL
オクテッド単位に接続を許可する。 接続を許可するIP アドレスの範囲が以下の場合 下限：192.168.0.0 上限：192.168.0.255	ALLOW=192.168.0.* DENY=ALL
すべてのコンピュータからの接続を許可する。	ALLOW=ALL DENY=ALL

## 環境変数CBR\_ATTACH\_TOOL

アタッチ形式(サービスなどからCOBOLプログラムが呼び出される際にデバグが起動される)でリモートデバグを行う場合、リモートデバグを起動するWindowsクライアントを指定するために設定します。

### 一般的な設定方法

環境変数CBR\_ATTACH\_TOOLは、アタッチ形式でリモートデバグを行う際のWindows側クライアントを指定します。

アタッチ形式のデバグの対象とするサーバプログラムから呼び出されるCOBOLプログラムに有効となる環境変数は、基本的にサーバプログラムの起動時に有効であった環境変数です。

このため、サーバプログラムの起動スクリプトあるいは、起動スクリプトに先立って実行されるシステムのデフォルトの環境変数などに、例えば次のような指定を追加します。

```
CBR_ATTACH_TOOL="client-1/TEST"
```

この指定はclient-1というWindowsクライアントをサーバプログラムから呼び出されるCOBOLプ

プログラムのリモートデバッグに使用することを指定します。  
 サーバプログラムから呼び出されるプログラムは、同じ環境、資源、プロセス等を共有するため、複数のプログラムを同時にデバッグすることはお勧めしません。このため、サーバプログラムから呼び出されるプログラムをリモートデバッグする際には、専用のWindowsクライアントを使用してください。

### 設定の詳細

ここでは、環境変数CBR\_ATTACH\_TOOLの指定形式を説明します。

CBR\_ATTACH\_TOOL=接続先/TEST [起動パラメタ]

接続先には、Windowsクライアント側のリモートデバッグコネクタの位置とポート番号を以下の形式で指定します。

$$\left\{ \begin{array}{l} \text{IPアドレス} \\ \text{ホスト名} \end{array} \right\} \quad [:\text{ポート番号}]$$

ポート番号は、1024から65535の範囲の数字でなければなりません。ポート番号の指定を省略した場合は、59999が指定されたとみなされます。

例

〔IPアドレス (192. 168. 0. 1) での指定〕

CBR\_ATTACH\_TOOL="192. 168. 0. 1:2000/TEST"

〔ホスト名 (client-1) での指定〕

CBR\_ATTACH\_TOOL="client-1:2000/TEST"

〔ポート番号省略の指定〕

CBR\_ATTACH\_TOOL="192. 168. 0. 1/TEST"

起動パラメタには、デバッグに渡す起動パラメタを指定することができます。リモートデバッグを行う場合、次の起動パラメタが有効です。

表2-8 リモートデバッグ時に有効な起動パラメタ

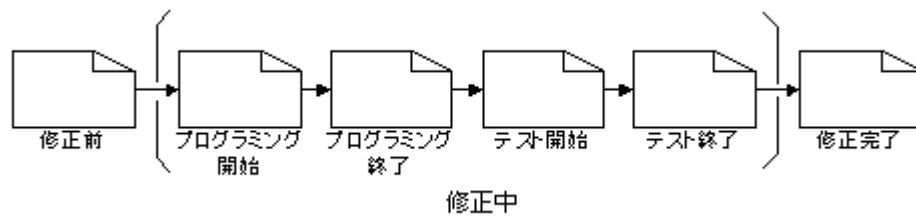
指定形式	内容
-G 開始プログラム名	〔デバッグを開始する〕ダイアログでの開始プログラム名を指定します。
-B コマンドファイル名	〔デバッグを開始する〕ダイアログでのコマンドファイル名を指定します。
-L 操作履歴ファイル名	〔デバッグを開始する〕ダイアログでの操作履歴ファイル名を指定します。
-N	〔デバッグを開始する〕ダイアログを表示しないで、デバッグを開始します。

### 2.1.4 資産管理方法と資産の配置



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、それを管理していくことが要求されます。

図2-7 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いく必要があります。

- 最新の資産はどこにあるか？
- 資産はどこで作成・修正するべきか？
- 資産の転送はいつ行われたか？
- 資産は修正の対象か、参照専用か？
- 資産の文字コード系は？

UNIX系プログラムの分散開発を行う場合、それぞれ対応するプログラム資産がWindowsクライアント側とUNIXサーバ側の両方に存在するため、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。

資産の管理方法の考え方としては、次の2つがあります。

- プログラム資産の作成・修正は主にWindowsクライアント側で行うため、資産管理はWindowsクライアント側で行う。
- 最終的な開発生産物はUNIXサーバ側に作成されるものであるため、資産管理はUNIXサーバ側で行う。

NetCOBOLを使用している分散開発機能は、Windowsクライアント側でプログラム資産の修正を行うことを前提としています。このため、Windowsクライアント側の修正を直ちに、UNIXサーバ側に反映する方法を提供しており、その設定が有効であれば、プログラム資産の同期についてはあまり考慮する必要がありません。次のような事情がない限り、Windowsクライアント側で資産管理を行うことをお勧めします。

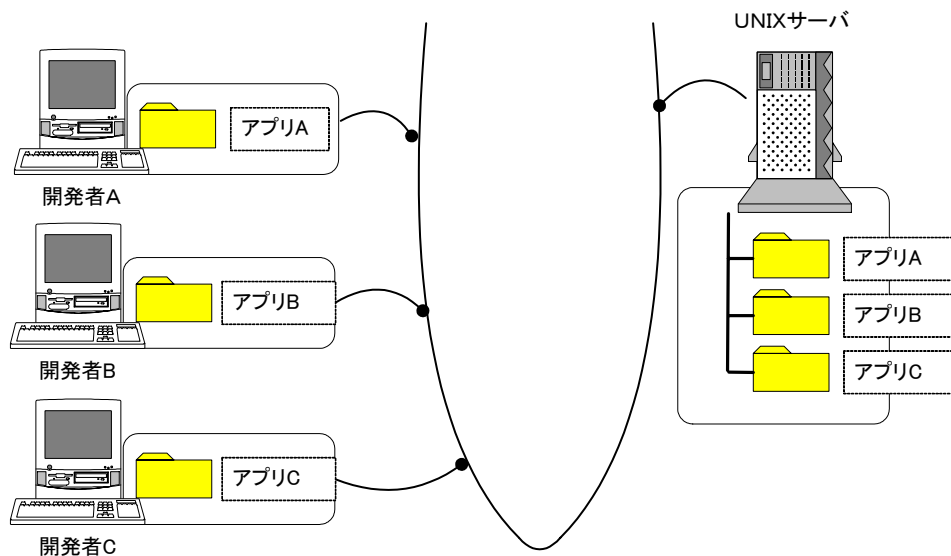
- 従来からUNIXサーバ側で資産を管理している。
- Windowsクライアント側で資産管理を行うための仕組み・ソフトウェアを用意できない。

### Windowsクライアント側とUNIXサーバ側の資産の配置

Windowsクライアント側、UNIXサーバ側、それぞれに資産をどのように配置するかは、開発対象のUNIX系プログラムとそれをどのような単位に分割し、開発を行うかに依存します。

1つの形態として、以下に示す図のように各開発者が自身の作業するPC上に資産をそれぞれ持つことが考えられます。

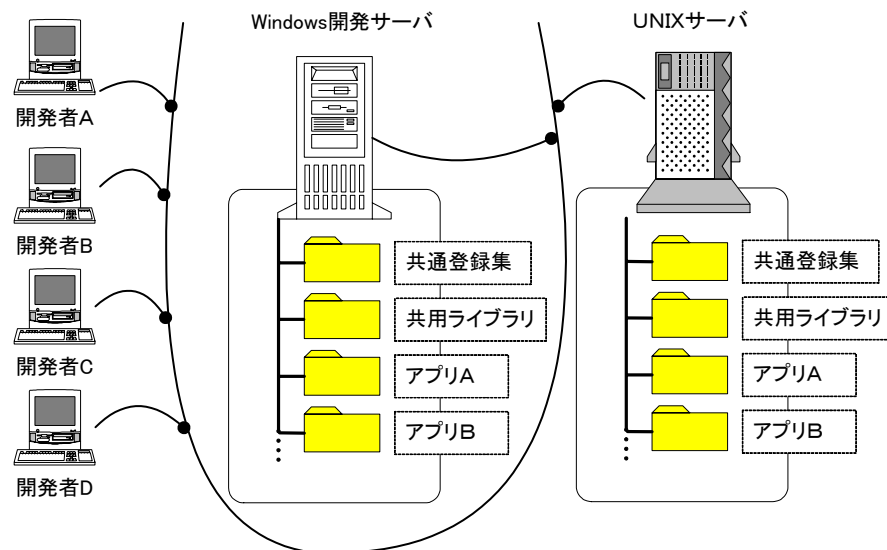
図2-8 Windowsクライアント側にそれぞれ資産を持つ場合



開発規模が小さい場合や、各開発者の担当するプログラム間の独立性が高いような場合（例えばバッチ処理型のプログラム等）は、簡単に作業環境を整えることができるため便利です。

これに対して、開発規模が大きい、あるいは各開発者の担当するプログラム間で共有する登録集や共通プログラムが多数存在するような場合は、開発者が共有する登録集や共通プログラムをそれぞれ自分のPCに所有することは望ましくありません。そのような場合、各開発者のPCに開発資産を置くのではなく、Windows側も資産を管理する専用のサーバを用意して、開発を実施する必要があります。

図2-9 Windows側に開発サーバを置く場合



### PowerGEM Plusによる資産管理

Windows版またはSolaris版のNetCOBOL製品では、資産管理の機能を提供するPowerGEM Plusが含まれます。以下に、分散開発に使用するNetCOBOL製品の組み合わせとPowerGEM Plusを使用している資産管理が可能かどうか、また可能な場合はWindowsクライアント、UNIXサーバのどちらで可能であるのかを対応表として示します。

表2-9 NetCOBOL製品の組合せとPowerGEM Plusによる資産管理

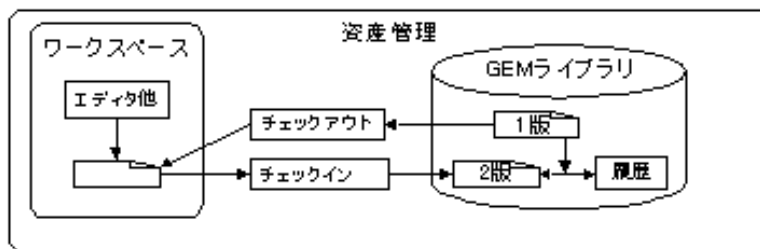
Windows製品		Standard Edition	Professional Edition
UNIX系製品			
Solaris	Professional Edition	⒰	⒰⒱
Linux32	Standard Edition	×	⒱
Linux64	Enterprise Edition	×	⒱

表中の記号の意味:

- ⒱ : Windowsクライアント側で資産管理が可能  
 ⒰ : UNIXサーバ側で資産管理が可能  
 × : 使用できるPowerGEM Plusがない

PowerGEM Plusでは、資産の管理機能として次のような機能を提供します。

図2-10 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。

### GEMライブラリ:

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システムのPowerGEMのGEMライブラリを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、GEMライブラリ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GEMライブラリ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GEMライブラリ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GEMライブラリに戻す操作です。

### 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、…)
- GEMライブラリの格納場所
- GEMライブラリの階層構造
- 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。

## 2.2 分散開発計画の立案



これまで説明してきた前提知識を元に、どのように分散開発環境を構築するか計画を立てます。

### UNIX系サーバ側の環境

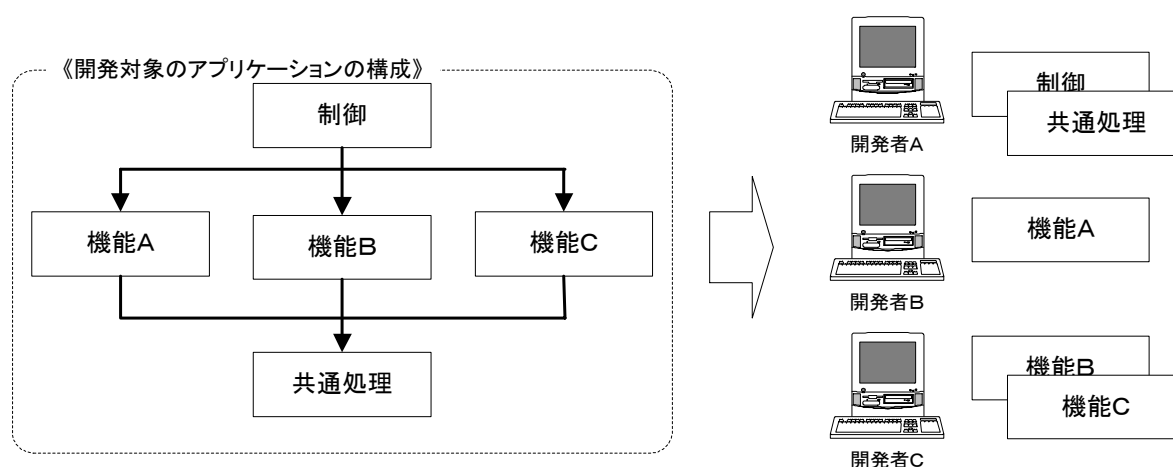
開発管理者は、次のようなことを検討・決定する必要があります。

- 開発対象の分担

“2.1.1 [COBOLプロジェクトマネージャにおけるプロジェクト](#)”で説明したように、NetCOBOLを用いての分散開発の単位は、個人で開発可能な単位に分割されている必要があります。

分散開発の対象となるアプリケーションが規模の大きなものである場合、それを構成するプログラムのサイズや機能の単位を元に適切な開発単位に分割し、それぞれに開発者を割り当てます。

図2-11 開発対象アプリケーションの分担例



- 分散開発時のUNIXサーバ側の環境設定

“2.1.2 [分散開発時のUNIXサーバ上の環境](#)”で説明したように、プログラム資産の送受信からターゲットビルドまでの操作に必要な環境変数は、UNIXサーバに接続する際のログインシェルで指定する必要があります。これは、分散開発時に使用するユーザ識別名に既存のものを使用する場合、そのログイン環境を書き換えることになります。

このため、開発管理者は、各開発者が使用するUNIXサーバ側のユーザ環境の割り当てについて次の事を決定する必要があります。

- 分散開発に既存のユーザ識別名を使用するか、新しくユーザ識別名を割り当てるか
- 新しくユーザ識別名を割り当てる場合、開発者個人単位に割り当てるかどうか

また、設定すべき環境変数を決定します。これについては“2.3 [分散開発のための環境設定](#)”で設定例を交えてより詳細に説明します。

- リモートデバッガの使用方法

“2.1.3 [リモートデバッグの概要](#)”で説明したように、リモートデバッグの方法には2つの方式があります。開発対象のプログラムにより、どちらの方式を使用するかを選択します。

リモートデバッガからデバッグ対象プログラムを起動する方式(一般形式)では、サーバ側リモートデバッガコネクタを使用します。この場合は、各開発者が使用するサーバ側リモートデバッガコネクタのポート番号を割り当てておく必要があります。

- 資産管理をどのように行うか

“2.1.4 [資産管理方法と資産の配置](#)”での説明を参考にプログラム資産管理の方法と、プログラム資産をWindowsクライアント、UNIXサーバにどのように配置するかを決定してください。

### Windowsクライアント側の環境

開発管理者は、個々の開発者が利用するWindowsクライアントで次の項目についてどのように設定すべきかを、開発者に伝えます。

- 開発に使用するUNIXサーバのホスト名またはIPアドレス
- 開発に使用するユーザ識別名およびパスワード
- 開発対象プログラムが日本語文字として使用するコード系
- 文字コード変換をサーバ・クライアントどちら側で行うか
- ファイルの送受信にPASVモードを使用する必要(使用するUNIXサーバがファイアウォール外にある)があるかどうか

一般の開発者は、その決定に従ってWindowsクライアント側の環境設定を行います。設定方法の詳細については“2.3.2 [クライアント側の環境設定](#)”で説明します。

## 2.3 分散開発のための環境設定

ここでは、実際の分散開発環境構築のため、次の設定について詳細を説明します。

- UNIXサーバ側の環境設定
- Windowsクライアント側の環境設定

### 2.3.1 サーバ側の環境設定



ここではUNIXサーバ側の環境設定について説明します。なお、これらの操作にはUNIXサーバの管理者(root)権限が必要です。

#### 2.3.1.1 サーバ側へのプログラムの導入と起動

NetCOBOLのWindows版製品の提供する分散開発支援機能は、UNIX系システム側の次のようなサービスが動作していることが前提となっています。したがって、これらのサービスの導入と起動が必要です。

- ftpd
- rexec

またリモートデバッグ機能を使用する場合は、次のサービスも必要となります。

- telnetd

以下、UNIXシステムが、Solarisサーバである場合およびLinuxサーバである場合それぞれについて説明します。

#### Solarisサーバの場合

Solarisサーバの場合、ftpdおよびrexecサービスは、デフォルトではオペレーティングシステムの導入時にインストールされ、常に起動するようになっています。

Solarisサーバの場合、システム設定の変更は設定ファイルの内容を直接確認し、必要ならそれを修正することが一般的ですので、その方法のみを説明します。

##### 1. サービスの状態の確認

Solarisサーバではftpdおよびrexecは、inetd(インターネットデーモン)から呼び出されるサービスです。したがって、次のファイルの内容を確認します。

- /etc/services
- /etc/inetd.conf

“/etc/services” および “/etc/inetd.conf” 内のftpdおよびrexecに関する記述が存在し、それが有効であれば、以降の作業は必要ありません。次にその例を示します。

図2-12 /etc/servicesの例

```
#
# Network services, Internet style
#
...
ftp          21/tcp
...
## UNIX specific services
## these are NOT officially assigned
#
...
exec         512/tcp
...
```

図2-13 /etc/inet.confの例

```
...
# FTPD - FTP server daemon
ftp      stream  tcp6    nowait  root    /usr/sbin/in.ftpd      in.ftpd -a
...
# REXECD - rexec daemon (BSD protocols)
exec     stream  tcp      nowait  root    /usr/sbin/in.rexecd    in.rexecd
exec     stream  tcp6     nowait  root    /usr/sbin/in.rexecd    in.rexecd
...
```

一方、ftpd、rexecの両方またはどちらかについての設定行がコメント化(行頭に“#”)されている場合は、以降の作業を行ってください。

2. サービスの設定の変更  
“/etc/services” および “/etc/inetd.conf” を修正します。
3. サービスの起動

ftpdおよびrexecは、inetd配下で起動されるサービスであるため、inetdを再起動します。次のコマンドを実行してください。

```
# kill -HUP `cat /var/run/inetd.pid`
```

## Linuxサーバの場合

Linuxサーバの場合、ftpdおよびrexecサービスは、オペレーティングシステムの導入時にインストールされていない場合もあるため、これらのパッケージが導入済みかどうかから説明します。なお、Linuxでは、この種のシステム設定のためにGUIを持つツールが用意されている場合がありますが、GUIツールはバージョンおよび個々のシステムの設定による違いが大きいため、コマンドによる操作方法のみを説明します。

1. パッケージの確認  
パッケージがインストール済みか確認するには次の形式でrpmコマンドを実行します。

```
rpm -query パッケージ名
```

ftpdおよびrexecに必要なパッケージ名は次のとおりです。

- ftpd : vsftpd
- rexec : rsh

次に示すようにインストールされているパッケージの情報が表示されている場合、パッケージはインストール済みです。サービスの状態の確認に進んでください。

```
# rpm -query vsftpd
vsftpd-2.0.1-5
# rpm -query rsh
rsh-0.17-2.5
```

パッケージの情報が表示されない場合、パッケージのインストールを行う必要があります。



**注意**

Linuxシステムで使用するftpdのパッケージには、バージョンやディストリビューションの違いにより幾つか種類があります。次のようなものが使われている場合もあります。

- wu-ftp
- proftpd

2. パッケージの導入  
次のrpmコマンドを使用してパッケージを導入します。

```
rpm -Uvh パッケージ名
```

### 3. サービスの状態の確認

/sbin/chkconfigコマンドを次の形式で使用して、システム起動時のサービス開始の設定を確認します。

```
/sbin/chkconfig --list サービス名
```

例えば、次のような結果が得られる場合、ftpd(vsftpd)とrexecはシステム起動時に開始されない設定になっています。

```
# /sbin/chkconfig --list vsftpd
vsftpd          0:off  1:off  2:off  3:off  4:off  5:off  6:off
# /sbin/chkconfig --list rexec
rexec           off
```

### 4. サービスの設定の変更

/sbin/chkconfigコマンドを次の形式で使用して、システム起動時にサービスが開始されるように設定を確認します。

```
/sbin/chkconfig [--level レベル] サービス名 on
```

以下、システム開始時にサービスが開始されるように設定を変更し、その変更を確認する例について示します。

```
# /sbin/chkconfig --level 5 vsftpd on
# /sbin/chkconfig --list vsftpd
vsftpd          0:off  1:off  2:off  3:off  4:off  5:on   6:off
# /sbin/chkconfig rexec on
# /sbin/chkconfig --list rexec
rexec           on
```

### 5. サービスの起動

/sbin/serviceコマンドを次の形式で使用して、サービスを開始します。

なお、rexecの場合は、xinetd配下で起動するサービスであるため、xinetdを再起動する必要があります。

```
# sbin/service vsftpd start
vsftpd 用の vsftpd を起動中: [ OK ]
# /sbin/service xinetd stop
xinetd を停止中: [ OK ]
# /sbin/service xinetd start
xinetd を起動中: [ OK ]
```

## 2.3.1.2 UNIXサーバ側のユーザ環境の設定

UNIXプログラムの分散開発時に、UNIXサーバ側の設定が必要な環境変数の詳細と、その設定方法について説明します。

### プログラム資産の送受信

プログラム資産の送受信に関係する環境変数は次のものです。

- LD\_LIBRARY\_PATH

プログラム資産の送受信に必要なコード変換をUNIXサーバ側のCharset MGRを使用して行う場合に設定が必要です。

## ターゲットビルド

ターゲットビルドは、UNIXサーバ側のcobol翻訳コマンドを使用して、COBOLプログラムの翻訳・リンクが行われます。このため、次に示す3つの環境変数の指定は必須です。なお、以下の説明では、{COB\_BASED}がNetCOBOLのインストール先を示すものとして説明します。

- PATH

COBOL翻訳コマンドの格納パスを指定するため、以下の指定を環境変数PATHに追加します。

```
{COB_BASED}/bin
```

- LD\_LIBRARY\_PATH

COBOLランタイムの共用ライブラリを格納したパスを指定するために、以下の指定を環境変数LD\_LIBRARY\_PATHに追加します。

```
{COB_BASED}/lib
```

- NLS\_PATH

COBOL翻訳時およびCOBOLプログラム実行時に出力されるメッセージの格納先を指定するため、以下の指定を環境変数NLS\_PATHに追加します。

```
{COB_BASED}/lib/nls/%L/%N.cat: {COB_BASED}/lib/nls/C/%N.cat
```

- LANG

COBOLプログラムで使用する文字コード系を指定します。翻訳時はこの指定がCOBOLソース中の日本語文字の有無とそのコード系の判定に使用されます。

**表2-10 COBOLプログラムで使用する文字コード系環境変数LANGの指定**

	日本語の使用の有無と文字コード			
	無し	有り		
		EUC	シフトJIS	Unicode(UTF8)
Soalris	C	ja	ja_JP. PCK	ja_JP. UTF-8
Linux	C	ja_JP. eucJP	—	ja_JP. UTF-8

LANGを除く環境変数の設定は、そのためのシェルスクリプトが各UNIX系システムのNetCOBOL製品に用意されており、通常はそれを使用します。

以下にその一覧を示します。

**表2-11 翻訳・リンク時に必須の環境変数を設定するためのシェルスクリプト**

システム	格納場所	ファイル名	備考
Solaris	/opt/FJSCbl/config	cobol.csh	csh用
Linux	/opt/FJSCbl/config	cobol.sh	sh/bash用
		cobol.csh	csh/tcsh用

その他、必要に応じて次のような環境変数を指定します。

- COBOLOPTS

開発対象の個々のプログラムに依存せず共通に指定する必要がある翻訳オプションがある場合、この環境変数を使用します。次のようなオプションを指定するのに有効です。

- COBOLのデバッグ機能に関するオプション
- 翻訳リストに関するオプション

- COBCOPY/FORMLIB/FILELIB

複数の開発者が共用する必要があるCOBOL登録集、画面帳票定義体、ファイル定義体等がある場合、その格納ディレクトリを指定します。

### 環境変数設定用のシェルスクリプト例

ここでは、設定する必要がある環境変数が次のようであると仮定して、その環境変数を設定するためのスクリプトの例を示します。

- 資産の送受信時に関係する環境変数  
サーバ側でCharset MGRを使用するための設定を環境変数LD\_LIBRARY\_PATHに追加する。
- ターゲットビルドに関係する環境変数
  - COBOLプログラムの翻訳・リンクに必須の環境変数は、NetCOBOLで提供されているシェルスクリプトで設定する。
  - 開発対象のプログラムが使用する文字コードはEUCとし、それを環境変数LNAGに指定する。
  - COBOLソースの翻訳リストは、共通のフォルダに保存する。
  - 開発者が共通して参照する登録集の格納フォルダを指定する。

### Solarisサーバの場合

Solarisサーバを使用して、分散開発を行う場合、ログインシェルとしてcshを使用する必要があります。各開発者の使用するホームディレクトリにある“.cshrc”に以下のテキストを追加編集してください。

図2-14 Solarisサーバでの.cshrcへの修正例

```
## COBOL環境設定
source /opt/FJSVcbl/config/cobol.csh
## Charset MGRのための環境設定
if(${?LD_LIBRARY_PATH}) then
    setenv LD_LIBRARY_PATH /opt/FSUNiconv/lib:${LD_LIBRARY_PATH}
else
    setenv LD_LIBRARY_PATH /opt/FSUNiconv/lib
endif
## 開発者共通の翻訳・リンク時設定
setenv COBOLOPTS "-dp ../list"
setenv COBCOPY ../COPYLIB:${COBCOPY}
## 開発対象プログラムの使用する文字コード
setenv LANG ja
```

### Linuxサーバの場合

Linuxサーバを使用して、分散開発を行う場合、ログインシェルとしてcshまたはbashを使用することができます。

ログインシェルとしてcshを使用する場合、各開発者の使用するホームディレクトリにある“.cshrc”に以下のテキストを追加編集してください。

図2-15 Linuxサーバでの.cshrcへの修正例

```
## COBOL環境設定
source /opt/FJSVcbl/config/cobol.csh
## Charset MGRのための環境設定
if(${?LD_LIBRARY_PATH}) then
    setenv LD_LIBRARY_PATH /opt/FSUNiconv/lib:${LD_LIBRARY_PATH}
else
    setenv LD_LIBRARY_PATH /opt/FSUNiconv/lib
endif
## 開発者共通の翻訳・リンク時設定
setenv COBOLOPTS "-dp ../list"
setenv COBCOPY ../COPYLIB:${COBCOPY}
```



```
## 開発対象プログラムの使用する文字コード
setenv LANG ja_JP.eucJP
```

また、ログインシェルとしてbashを使用する場合、各開発者の使用するホームディレクトリにある“.bashrc”に以下のテキストを追加編集してください。

図2-16 Linuxサーバでの.bashrcへの修正例

```
## COBOL環境設定
. /opt/FJSVcbl/config/cobol.sh
## Charset MGRのための環境設定
## Charset MGRのための環境設定
if [ ${LD_LIBRARY_PATH:-""} = "" ] ; then
    LD_LIBRARY_PATH=/opt/FSUNiconv/lib; export LD_LIBRARY_PATH
else
    LD_LIBRARY_PATH=/opt/FSUNiconv/lib:${LD_LIBRARY_PATH}; export LD_LIBRARY_PATH
fi
## 開発者共通の翻訳・リンク時設定
COBOLOPTS=" -dp ../list" ; export COBOLOPTS
COBCOPY=../COPYLIB:${COBCOPY}; export COBCOPY
## 開発対象プログラムの使用する文字コード
LANG=ja_JP.eucJP; export LANG
```

### 2.3.1.3 リモートデバッグ時のサーバ側のユーザ環境の設定

ここではWindowsクライアントのリモートデバッガを使用して、UNIXサーバ側のプログラムをリモートデバッグする際に必要な環境変数の設定について説明します。

なお、ここでの説明は、“2.3.1.2 [UNIXサーバ側のユーザ環境の設定](#)”で説明した環境変数が設定済みであることを前提とします。

#### 一般形式でのリモートデバッグ

“2.1.3.1 [リモートデバッグの2つの方式](#)”で説明した通り、デバッグ対象のプログラムに有効となる環境変数は、サーバ側のリモートデバッガコネクタの起動時に有効であったものが引き継がれます。

デバッグ対象のプログラムに対して指定する必要がある環境変数は大きく2つに分かれます。

1. 開発対象のプログラムで共通である環境変数の設定  
ファイル操作時のふるまい、印刷ファイルで使用するフォント等、マルチスレッド時の設定など開発対象プログラムで共通する設定です。
2. 開発対象のプログラムそれぞれに固有である環境変数の設定  
入出力先ファイルを指定する環境変数が代表的なものです。

通常は、1に該当する環境変数はリモートデバッガコネクタの開始前に設定します。一方、2に該当する環境変数はデバッグ対象プログラム毎に実行環境初期化ファイルを用意して、リモートデバッグの開始時にそのファイルを指定することで設定します。

また、“2.1.3.2 [リモートデバッグのためのサーバ側の環境設定](#)”で説明したとおり、リモートデバッグを行うにあたって、サーバ側のリモートデバッガコネクタの起動が必要です。

しかし、個々の開発者がサーバ側のリモートデバッガコネクタを起動するにあたって、次の問題があります。

- Windowsクライアントから直接サーバ側のリモートデバッガコネクタを起動・終了する機能が用意されていない。各開発者はtelnet端末でUNIXサーバに接続し、リモートデバッガコネクタを起動する必要がある。
- 複数の開発者が同時にリモートデバッグを行う場合、それぞれにリモートデバッガコネクタを用意しないと、標準入力・標準出力・標準エラー出力・実行時メッセージなどが混信

してしまう。このため、各開発者が接続制限とポート指定で個々にリモートデバッガコネクタを起動する必要がある。

このため、開発管理者は一般の開発者がリモートデバッガコネクタの起動・終了を簡単、かつ、安全に行うために次の準備をする必要があります。

- a. 各開発者にリモートデバッグ時に使用するポート番号を割り当てる。
- b. リモートデバッガコネクタの起動前に適切な環境変数の設定を行うシェルスクリプトを用意する。
- c. リモートデバッガコネクタの起動・終了などの操作を開発者が誤りなく行えるように手順を用意する。

これらについては、リモートデバッガコネクタ起動・終了用の専用のシェルスクリプトを用意し、一般の開発者にはそれを使用してもらうようにするのが有効です。

以下にその例を示します。

図2-17 リモートデバッガコネクタ起動終了用のシェルスクリプト例

```
#!/bin/sh
PORT=10000
CLIENT=192.168.1.13
if [ "$1" = "start" ] ; then
    if [ -r svdrds.pid ] ; then
        echo "リモートデバッガコネクタは既に起動済みです(ポート番号:${PORT})."
    else
        ## 開発対象プログラム共通の環境変数の設定開始 ##
        # 行順ファイルの扱い
        CBR_TRAILING_BLANK_RECORD=YES; export CBR_TRAILING_BLANK_RECORD
        ...

        ## 開発対象プログラム共通の環境変数の設定終了##
        echo "リモートデバッガコネクタを起動します (ポート番号:${PORT})"
        svdrds -h ${CLIENT} -p ${PORT} > /dev/null&
        ps -u user001 | grep svdrds | sed -e "s/^ */;/s/ .*//" >svdrds.pid
    fi
elif [ "$1" = "end" ] ; then
    if [ -r svdrds.pid ] ; then
        echo "リモートデバッガコネクタを終了します"
        kill -s KILL `cat svdrds.pid`
        rm -f svdrds.pid
    else
        echo "リモートデバッガコネクタは起動されていません。"
    fi
elif [ "$1" = "port" ] ; then
    echo "リモートデバッガコネクタはポート番号:${PORT}を使用します。"
else
    echo "Usage : rdebug [ start | end | port ]"
fi
```

## 2.3.2 クライアント側の環境設定



UNIX系アプリケーションを分散開発する場合、各開発者が使用するWindows版のNetCOBOLにUNIXサーバと連携するための情報を設定する必要があります。ここでは、仮に次の条件で分散開発を行うものとして説明します。

- 開発に使用するUNIXサーバ : host01 (IPアドレス:192.168.10.123)
- 開発に使用するユーザ識別名: devuser01
- 開発対象プログラム : 実行時、日本語文字としてEUCを使用する。

なお、Windows XP SP2以降を適用済みのシステムを使用する場合、次の設定を先に行う必要があります。

- 2.3.2.5 [Windows XP SP2適用時の設定](#)

### 2.3.2.1 サーバ連携情報の設定

サーバ連携情報の設定はCOBOLプロジェクトマネージャ から行います。

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、[サーバ連携情報] ダイアログが表示されます。

図2-18 サーバ連携情報ダイアログの初期状態



3. [追加] ボタンをクリックするとサーバ連携情報の[追加] ダイアログが開きます。このダイアログから以下の情報を設定します。なお、[ホストでコード変換する]以降の設定の詳細については別途説明します(2.3.2.4 [サーバ連携情報設定に関する補足事項](#))。
  - [ホスト名]:  
分散開発に使用するUNIXサーバのホスト名を指定します。ここで指定した名前が[サーバ連携情報] ダイアログボックスのホスト名一覧に表示されます。
  - [ホストのアドレス]:  
ネットワーク上のUNIXサーバを識別するための名前、または、IPアドレスを指定します。
  - [ユーザ名]:  
分散開発時に[ホストのアドレス]で指定したUNIXサーバに接続するためのユーザ識別名を指定します。
  - [パスワード]:  
ユーザ識別名に付与されたパスワードを指定します。
  - [ホストでコード変換する]:  
テキストモードでファイルの送信・受信をする場合に、ホストまたはクライアントのどちらでコード変換するかを指定します。詳細は後述します。
  - [ホストのコード]:

UNIXサーバ側の日本語文字のコード系を指定します。以下の4種類から選択します。

- EUC
- EUC(U90)
- シフトJIS
- UTF-8

— [PASVモードでファイル転送をする]：

PASVモードでファイル転送をするか否かを指定します。チェックボックスをチェックするとPASVモードでファイル転送しますが、通常はその必要はありません。

図2-19 サーバ連携情報の追加ダイアログの初期状態

図2-19は「追加」ダイアログの初期状態を示しています。ダイアログには以下の項目があります：

- ホスト名(H): 空のテキストボックス
- ホストのアドレス(A): 空のテキストボックス
- ユーザ名(U): 空のテキストボックス
- パスワード(P): 空のテキストボックス
- コード変換セクション:
  - ☐ ホストでコード変換する(V)
  - ホストのコード(S): プルダウンメニューで「EUC」が選択されている
- ☐ PASVモードでファイル転送をする(M)
- ボタンの列: OK, キャンセル, 確認(O)..., ヘルプ°

4. 最初に説明した条件で分散開発を行う場合、次のように設定します。

図2-20 サーバ連携情報の追加ダイアログの設定例

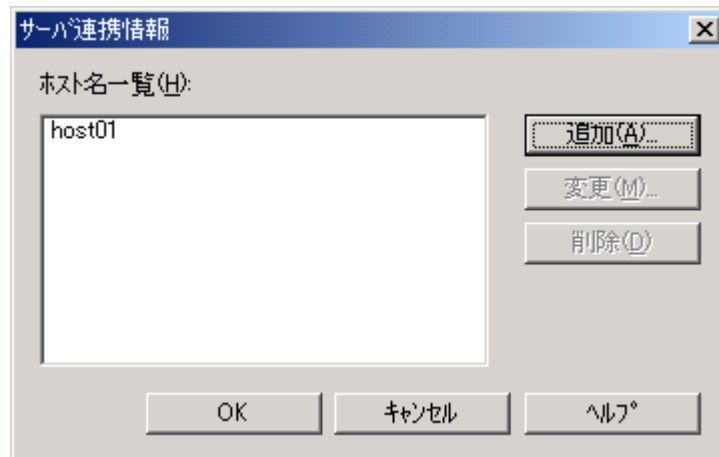
図2-20は「追加」ダイアログの設定例を示しています。ダイアログには以下の項目があります：

- ホスト名(H): host01
- ホストのアドレス(A): 192.168.10.123
- ユーザ名(U): dvuser01
- パスワード(P): \*\*\*\*\*\*
- コード変換セクション:
  - ☐ ホストでコード変換する(V)
  - ホストのコード(S): プルダウンメニューで「EUC」が選択されている
- ☐ PASVモードでファイル転送をする(M)
- ボタンの列: OK, キャンセル, 確認(O)..., ヘルプ°

5. [確認] ボタンをクリックすると、設定したサーバ連携情報を使用してのUNIXサーバとの接続結果を確認することができます。[確認] ボタンによる接続結果の確認については、次で詳細に説明します。
6. [OK] ボタンをクリックすると、サーバ連携情報が追加され、[サーバ連携情報] ダイアログに戻ります。

ここで、[サーバ連携情報] ダイアログの [OK] ボタンをクリックすると、設定が保存されます。[サーバ連携情報] ダイアログの [キャンセル] ボタンをクリックすると、設定は破棄されます。

図2-21 追加されたサーバ連携情報



### 2.3.2.2 サーバ連携情報の変更・削除

サーバ連携情報を変更・削除する場合、[サーバ連携情報] ダイアログの [ホスト名一覧] から対象となるホスト名を選択して、[変更] ボタンまたは [削除] ボタンをクリックします。各ボタンをクリックした場合の動作は次の通りです。

- [変更]:  
選択したホスト名について、サーバ連携情報の [変更] ダイアログが開きます。[変更] ダイアログで可能な設定可能な項目は [追加] ダイアログと同じです。[追加] ダイアログの説明を参照してください。
- [削除]:  
選択したホスト名が [ホスト名一覧] から削除されます。

いずれの場合も、最後に [OK] ボタンをクリックしなければ、設定の変更は保存されません。

### 2.3.2.3 サーバ連携情報による接続確認

サーバ連携情報の [追加] ダイアログおよび [変更] ダイアログにある [確認] ボタンをクリックすると、ダイアログ上に表示されているサーバ連携情報を使用したUNIXサーバへの接続が行われ、その結果が [確認] ダイアログに表示されます。

これによって、以下のことを確認することができます。

- サーバ連携機能が使用可能か
- UNIXサーバ側の環境変数

#### サーバ連携機能が使用可能か

次のいずれかの設定に誤りがある場合、

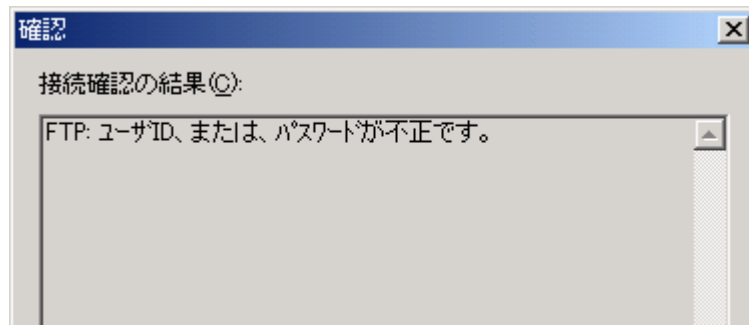
- a. サーバ側のサービス (ftpdまたはrexec) の設定
- b. サーバ連携情報で設定した次のような情報
  - [ホストのアドレス]:
  - [ユーザ名]:
  - [パスワード]:

UNIXサーバとの接続に失敗した場合、次の形式のエラーメッセージが [確認] ダイアログの [接続確認の結果] に表示されます。

サービス名: エラーメッセージ

例えば、ユーザ名/パスワードなどに誤りがあった場合には、次のように表示されます。

図2-22 サーバ連携情報による接続確認による失敗時の例



### UNIXサーバ側の環境

指定のサーバ連携情報で接続に成功した場合は、接続されたUNIXサーバ側の環境変数の設定を表示します。

図2-23 サーバ連携情報による接続確認による成功時の例



### 2.3.2.4 サーバ連携情報設定に関する補足事項

サーバ連携情報設定に関する補足事項として、ここでは次の2点について説明します。

- コード変換の設定
- PASVモードでのファイル転送

#### コード変換の設定

Windowsシステムでは、日本語文字を表現するための文字コード系としてシフトJISが使用されています。開発するUNIXアプリケーションが日本語文字コードとして、EUCやUnicodeを使用する場合は、COBOLソースなどのテキストファイルを送信・受信する際に文字コードの変換が必要となってきます。

このための設定が、サーバ連携情報の〔追加〕ダイアログおよび〔変更〕ダイアログの次の項目です。

- [ホストでコード変換する] :
- [ホストのコード] :

この文字コードの変換をUNIX系サーバ側で行うか、Windowsクライアント側で行うかを指定するのが、[ホストでコード変換する] チェックボックスです。チェックボックスをチェックすると、

UNIX系サーバ側でコード変換処理が実行されます。チェックボックスのチェックを解除すると、Windowsクライアント側でコード変換処理が実行されます。

コード変換処理は、通常はシステムの提供するコード変換の機能を使用して行われます。しかし、ADJUST (Windowsのみ) またはCharsetMGRが導入されているならば、これらの製品を使用してコード変換が行われます。

表2-12 コード変換の設定と変換プログラム

		[ホストでコード変換する] の設定			
		指定しない場合		指定した場合	
		UNIX サーバ側へのCharsetMGRの導入		UNIX サーバ側へのCharsetMGRの導入	
		No	Yes	No	Yes
Windowsクライアント側へのADJUSTまたはCharset MGRの導入	No	①	①	③	④
	Yes	②	②	③	④

表中の記号の説明：

- ①：Windowsシステムの提供するコード変換プログラム
- ②：Windowsシステムに導入したADJUSTまたはCharset MGR
- ③：UNIX系システムの提供するコード変換プログラム
- ④：UNIX系システムに導入したCharset MGR

### PASVモードでのファイル転送

COBOLプロジェクトマネージャが、UNIXアプリケーションの分散開発支援のために提供するサーバ連携には、ftpというファイル転送プロトコルが使用されています。

しかし、分散開発に使用するUNIX系サーバがファイアウォールの外にあるなどの理由から、デフォルトの設定ではftpプロトコルを使用して、UNIX系サーバに接続できない場合があります。このような場合、サーバ連携情報の〔追加〕ダイアログおよび〔変更〕ダイアログの次のチェックボックスをチェックしてください。

— [PASVモードでファイル転送をする]：

### 2.3.2.5 Windows XP SP2適用時の設定

Windows XP SP2でセキュリティ強化のために追加された“Windowsファイアウォール”が有効になっている場合、UNIXプログラムの分散開発に使用する次の機能が使用できなくなります。

- ファイルの送受信
- ターゲットビルド
- リモートデバッグ機能

アタッチ形式でのリモートデバッグが行えません。一般形式でのリモートデバッグは可能です。

この問題を回避するためには、次の表に示すプログラムを“Windowsファイアウォール”によるチェックの対象外とするように設定を変更します。

表2-13 Windowsファイアウォールに例外として登録するプログラム

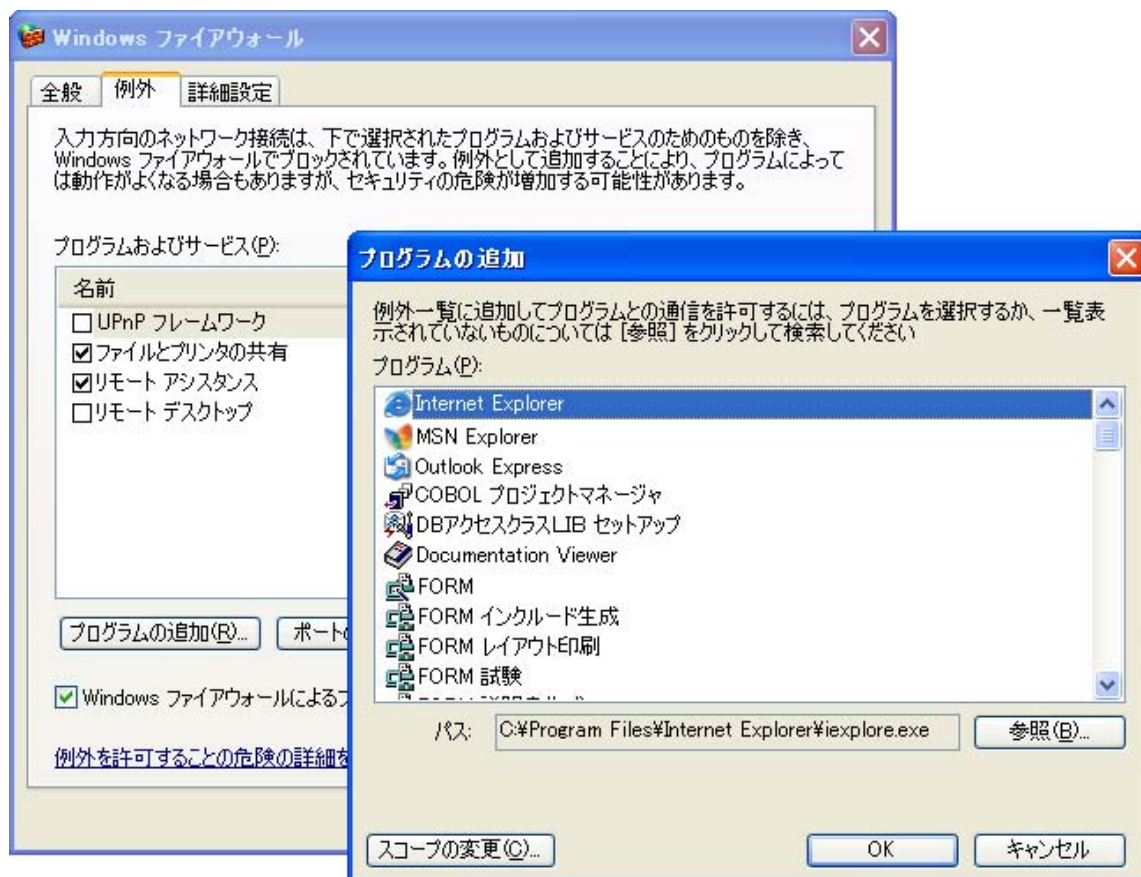
No	プログラム名	格納フォルダ	備考
1	F3BHSTBD. exe	NetCOBOLインストールフォルダ	ファイルの送受信, ターゲットビルド
2	F3BXVFRM. exe	PowerGEM Plusインストールフォルダ (NetCOBOLインストールフォルダ配下のPGEM)	
3	COBRDC32. exe	NetCOBOLインストールフォルダ	リモートデバッグ

### 回避方法

以下の手順で、“F3BHSTBD. exe”、“F3BXVFRM. exe”および“COBRDC32. exe”を例外として登録します。なお、機能を使用しないのであれば、対応するプログラムを登録する必要はありません。

1. [Windowsファイアウォール]設定画面の[例外]タブにおいて、[プログラムの追加]をクリックしてください。
2. [プログラムの追加]ダイアログの[参照]ボタンから、NetCOBOL製品のインストールフォルダに存在する“F3BHSTBD.exe”を選択し、[OK]ボタンをクリックして項目を追加してください。
3. “COBRDC32.exe”を上記と同様に選択し、[OK]ボタンをクリックして項目を追加してください。
4. PowerGEM Plusのインストールフォルダに存在する“F3BXVFRM.exe”を上記と同様に選択し、[OK]ボタンをクリックして項目を追加してください。

図2-24 [Windowsファイアウォール]設定画面と[プログラムの追加]ダイアログ



### スコープの変更について

上記の方法で、必要なプログラムを“Windowsファイアウォール”によるチェックの対象外として登録した場合、そのプログラムに対するスコープを変更することによって、セキュリティを強化することが可能です。スコープの変更は、次の手順で行います。

1. [Windowsファイアウォール]設定画面の[例外]タブに登録されているプログラムから、スコープを変更するプログラムを選択し、[編集]ボタンをクリックします。
2. [プログラムの編集]ダイアログが現れるので、その[スコープの変更]ボタンをクリックしてください。
3. [スコープの変更]ダイアログで、[ユーザのネットワーク (サブネットのみ)]を選択、または、[カスタムの一覧]を選択します。
4. [カスタムの一覧]を選択した場合、対象コンピュータのIPアドレスを設定し、[OK]ボタンをクリックします。



図2-25 [Windowsファイアウォール]設定画面と[プログラムの編集]ダイアログ

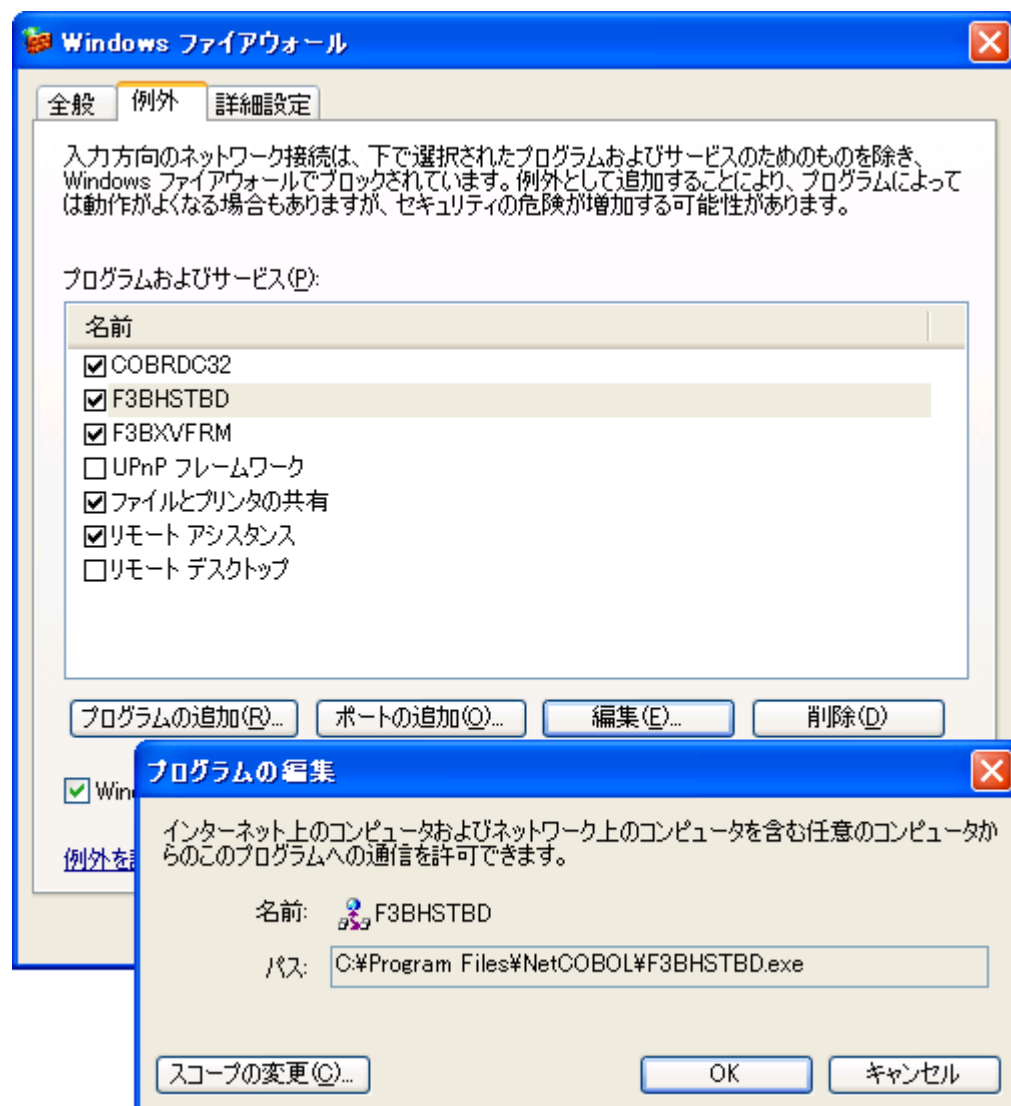
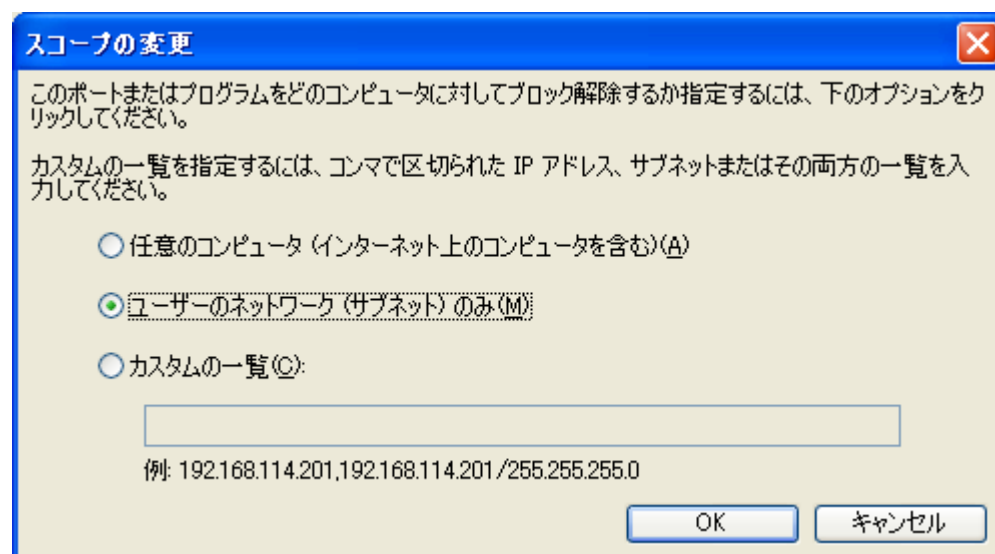


図2-26 [スコープの変更]ダイアログ





---

## 第3章 Windowsクライアントでの開発作業

---

本章では、分散開発のうち、Windowsシステム上で行なう作業について説明します。

## 3.1 Windowsクライアントでの作業概要

Windowsシステム上のNetCOBOLで、UNIX系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。プロジェクト管理機能で必要となる、プロジェクトファイルを作成します。
- 開発資産の移行  
UNIX系システムあるいはOS/IV系システムの既存の資産を利用して分散開発を行なう場合は、これらの資産をWindowsシステムに移行します。
  - 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。
- 翻訳チェックとリンク  
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プログラムを翻訳およびリンクします。翻訳エラーがあった場合、メッセージ連携機能(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。
- 単体テスト  
NetCOBOLの、COBOLデバッガを利用して、可能な範囲での単体テストをWindowsシステム上で行います。

## 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。UNIX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
- UNIX系システム、OS/IV系システムとのプログラム資源の移出入
- UNIX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
- 最終ターゲットファイル名
- ソースファイル、登録集ファイルなどプログラム資産名
- 分散開発時固有の設定

### 3.2.1 基本的なプロジェクトの作成



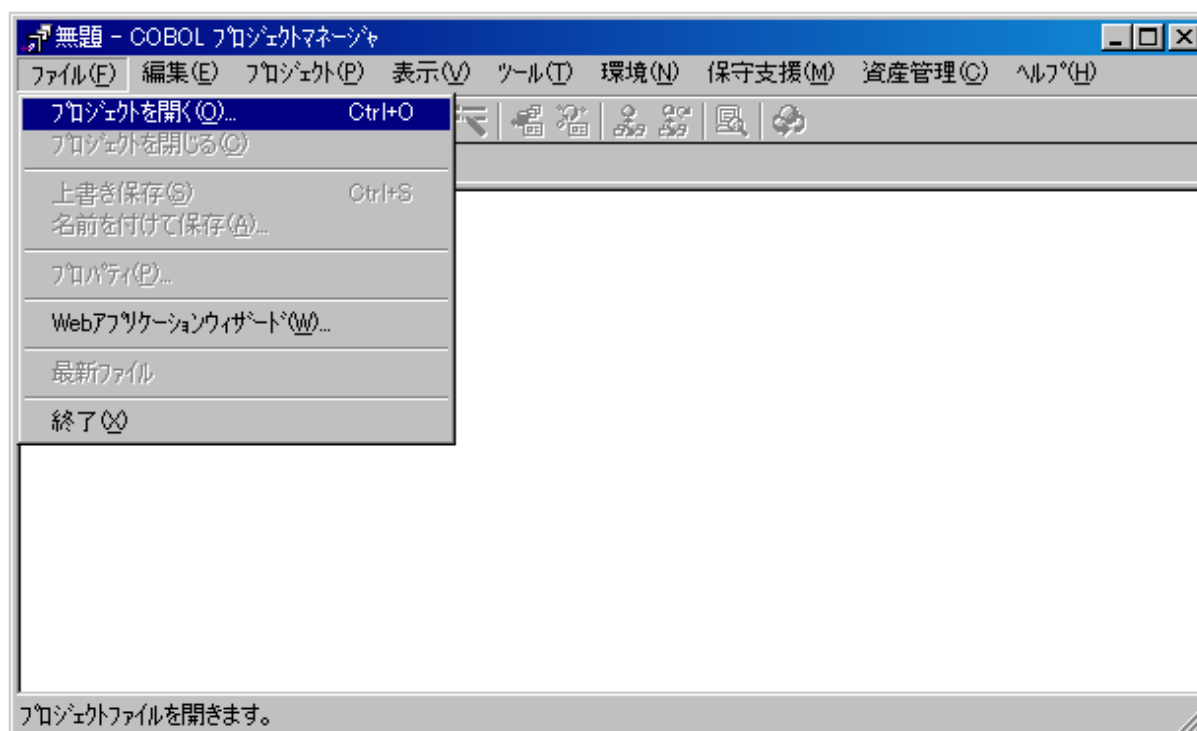
基本的なプロジェクトファイルの作成手順を説明します。

#### 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。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。

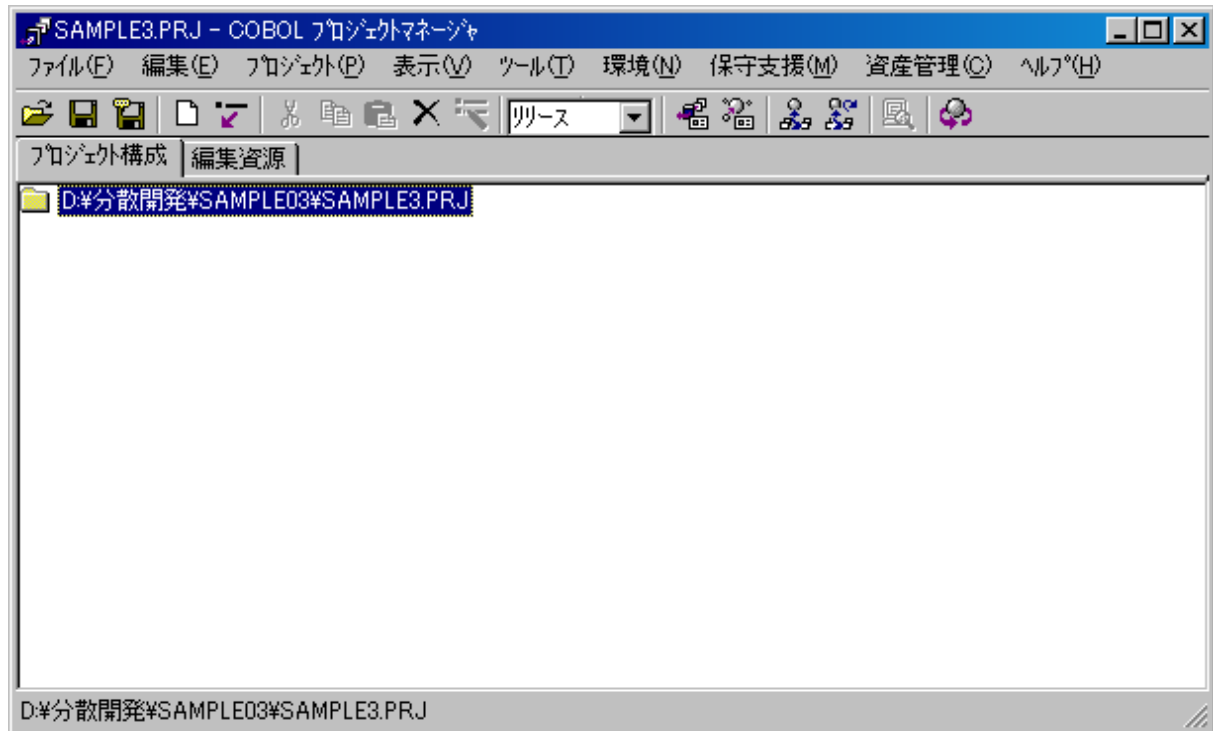
1. COBOLプロジェクトマネージャを起動して、[ファイル]メニューから“プロジェクトを開く”を選択します。

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設定になっていません。「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。

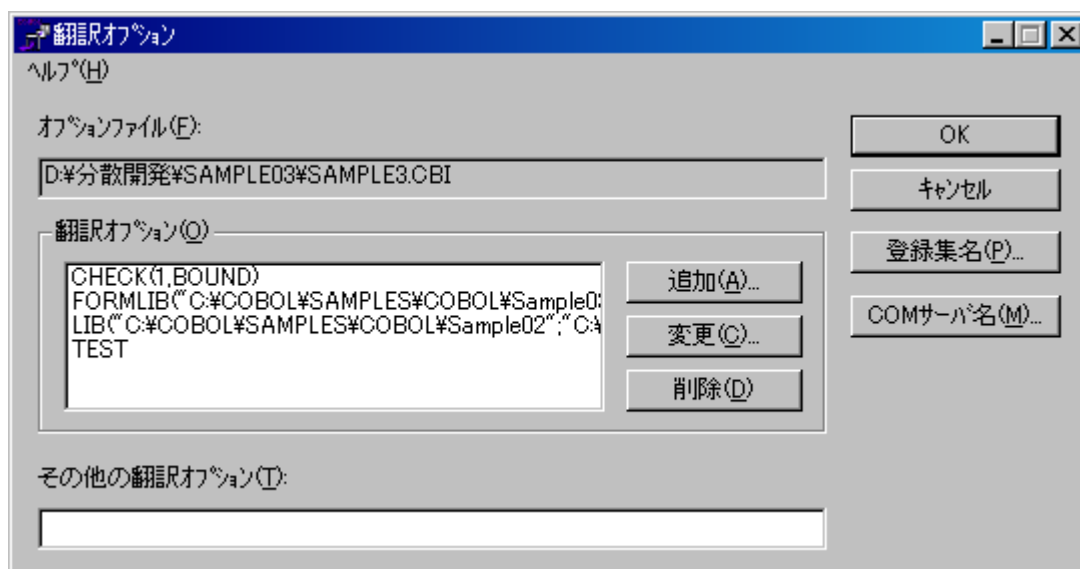
### 翻訳オプションの設定

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メイクファイルに反映されます。

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。
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

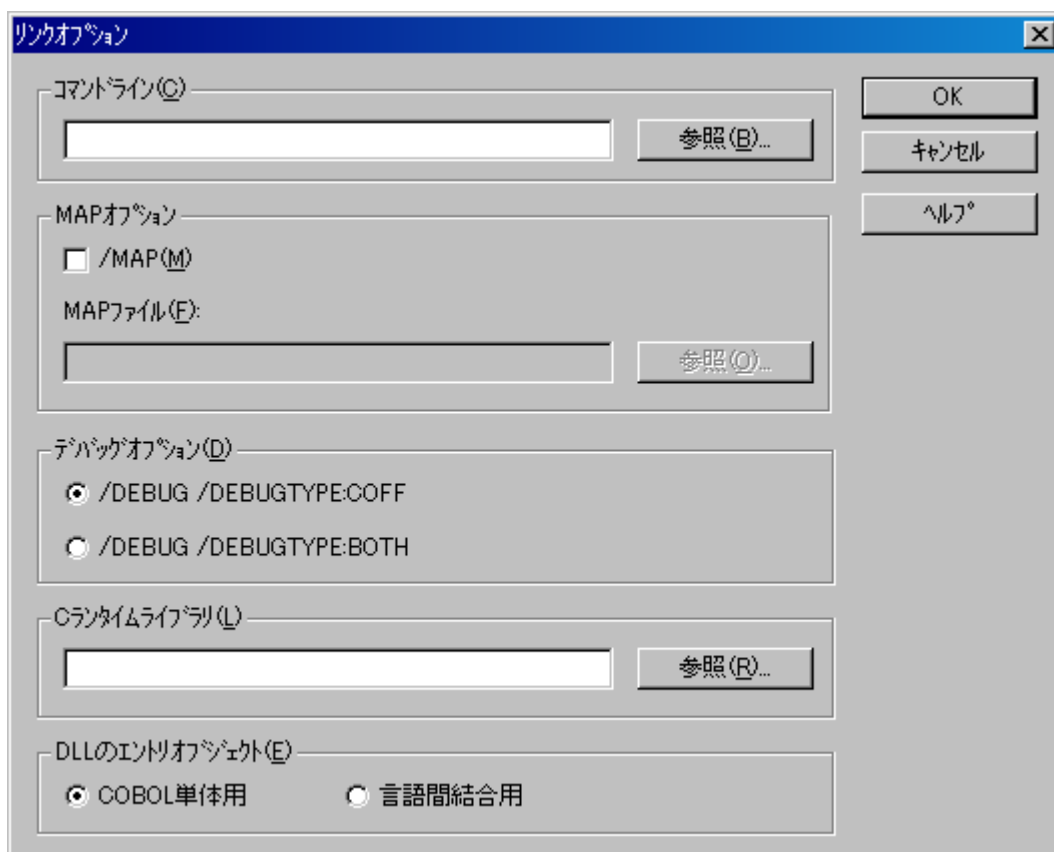
プロジェクトで管理している実行可能ファイルまたはダイナミックリンクライブラリファイルをリンクするときに有効になるリンクオプションを指定します。

ただし、指定したリンクオプションは、ビルド制御文生成機能を使用して生成したメイクファイルに反映されません。

Windowsシステム上で単体テストなどを行なうのに必要な場合のみ設定してください。

1. [プロジェクト] - [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3-4 [「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。

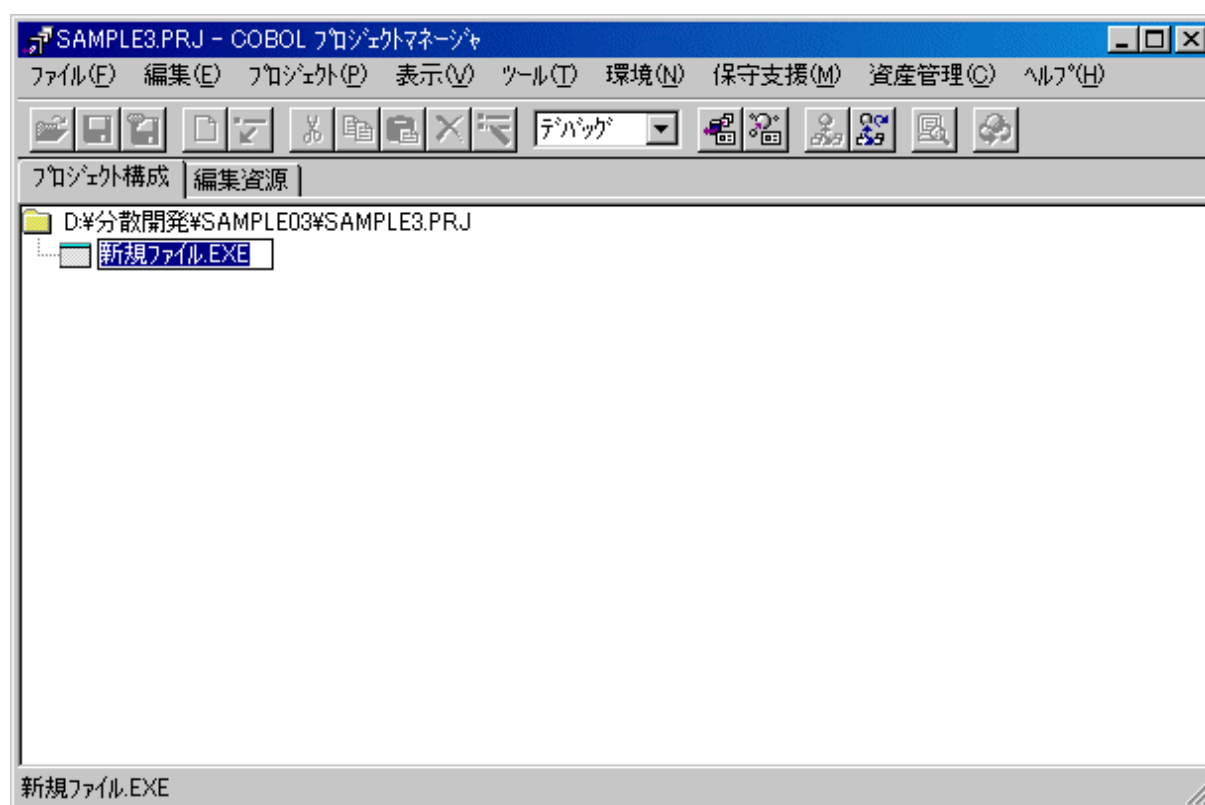
### 最終ターゲットファイルの追加

プロジェクトには最終ターゲットファイルとして、Windowsシステムの実行形式ファイルかダイナミックリンクライブラリファイル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、通常は一つのプロジェクトにつき、実行形式ファイルまたはダイナミックリンクライブラリファイルを一つのみ登録してください。

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”



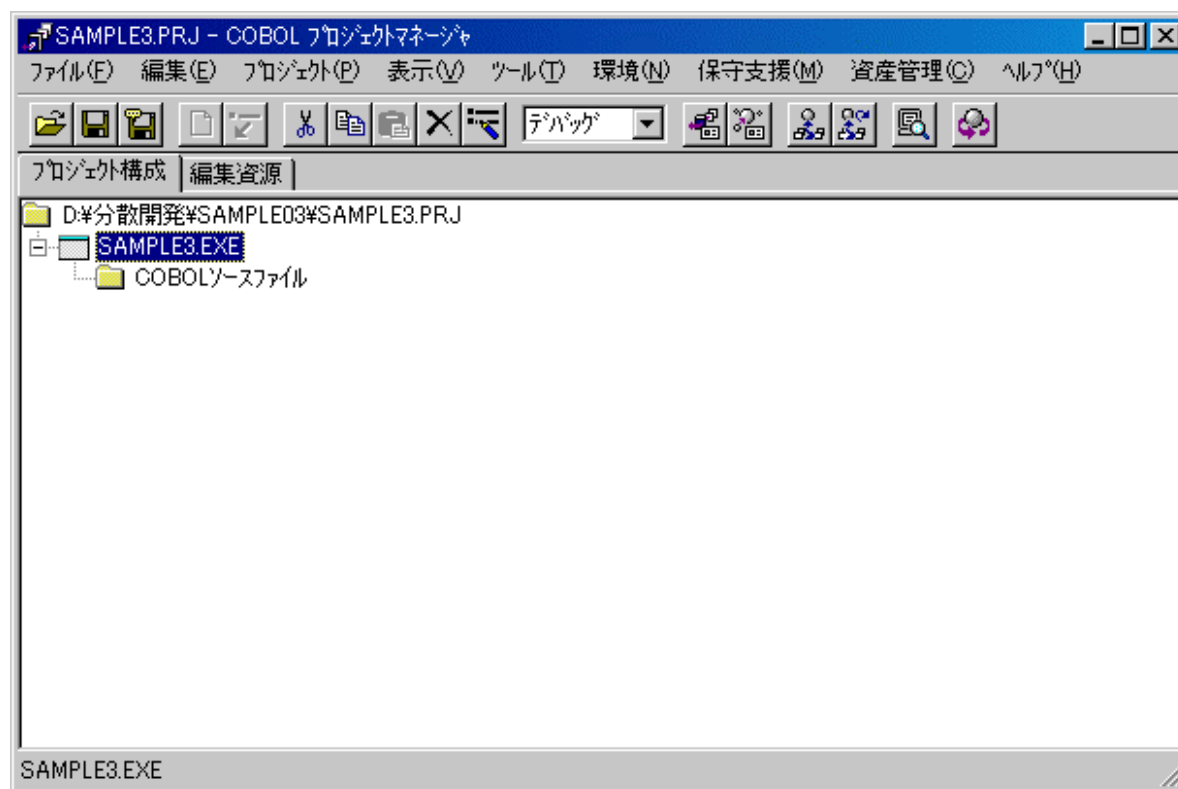
なお、最終ターゲットファイル名は、これをツリービュー上で選択して、〔編集〕メニューから“名前の変更”を選択することで、いつでも変更可能です。

### 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〕－〔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。

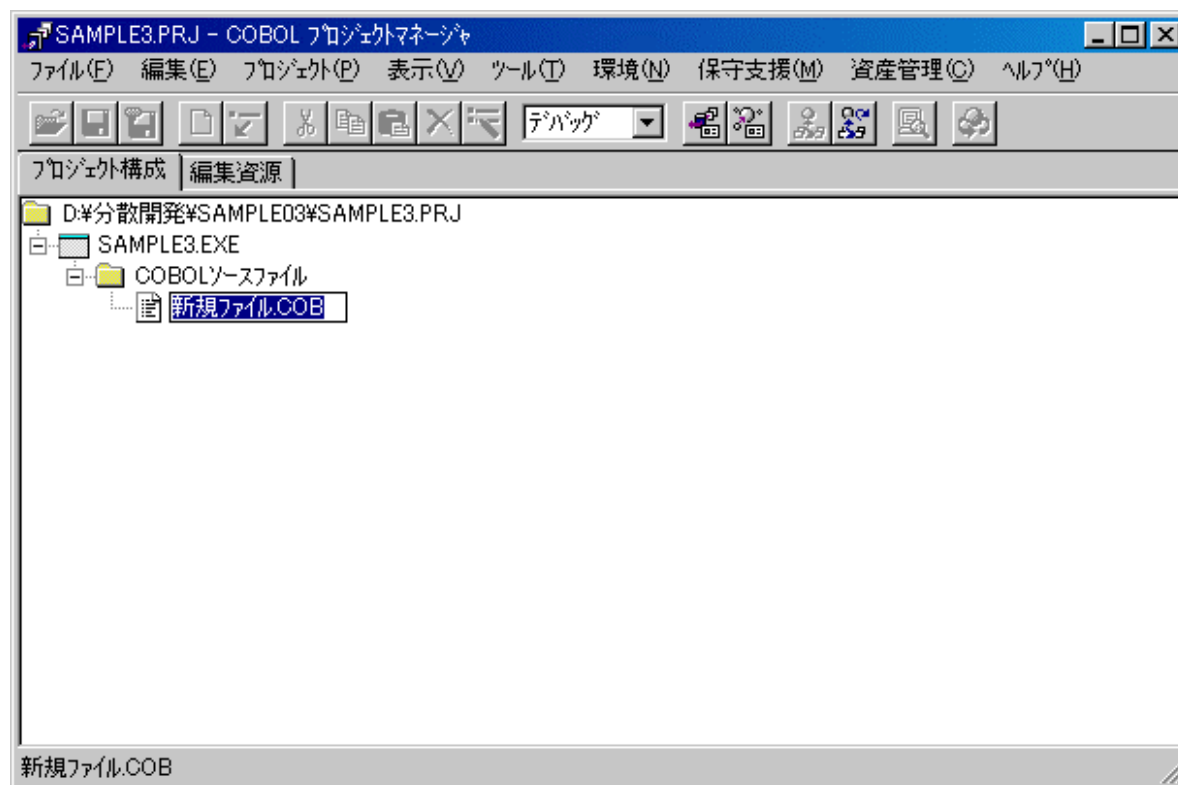
図3-6 追加された“COBOLソースファイル”フォルダ



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。

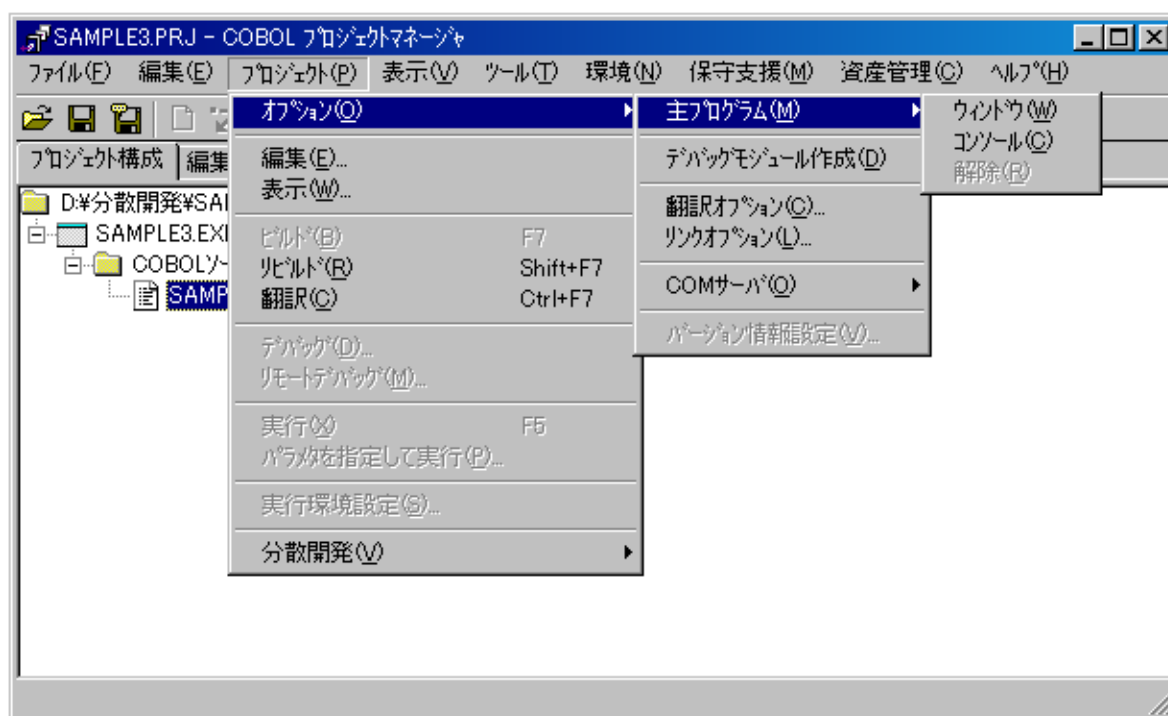
### 主プログラムの指定

登録するCOBOLソースファイルが主プログラムの場合、主プログラムの指定を行う必要があります。

以下の手順で、主プログラムを指定します。

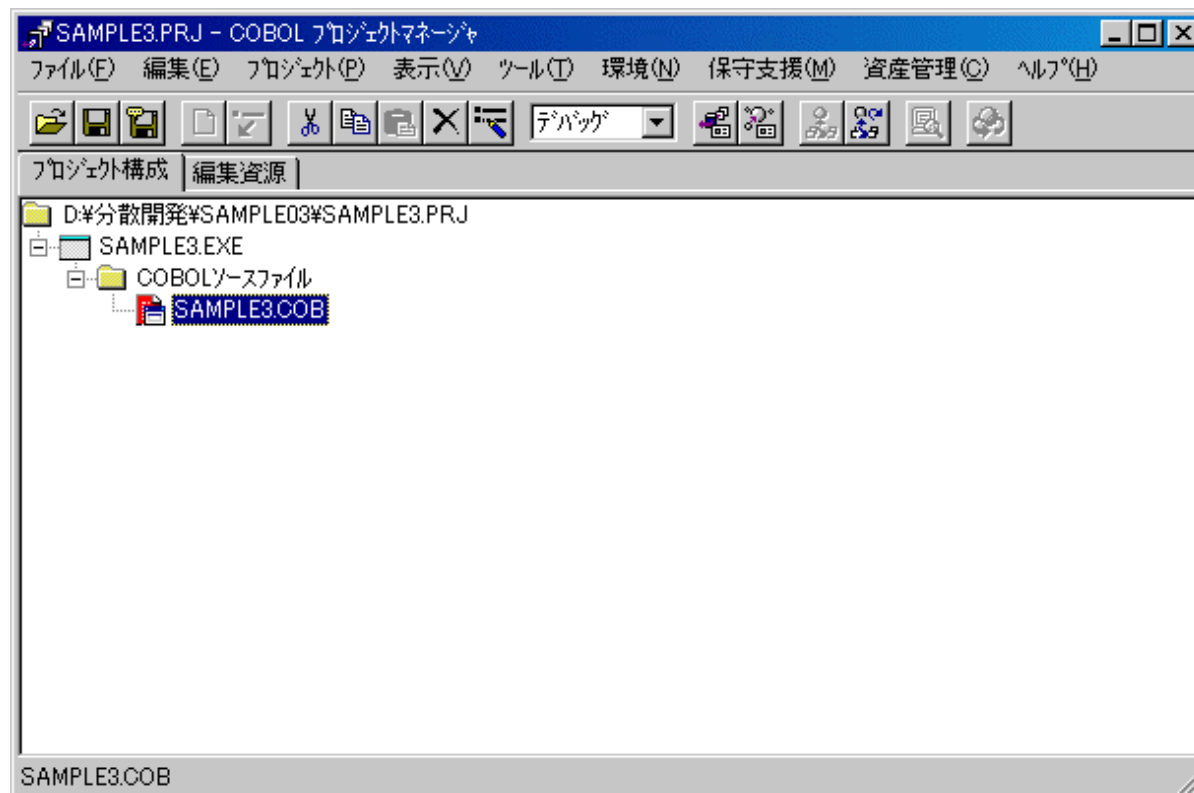
1. 主プログラムに設定するソースプログラムを選択します。
2. [プロジェクト] - [オプション] - [主プログラム] メニューで“ウィンドウ”または“コンソール”を選択します。

図3-8 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3-9 設定された主プログラム

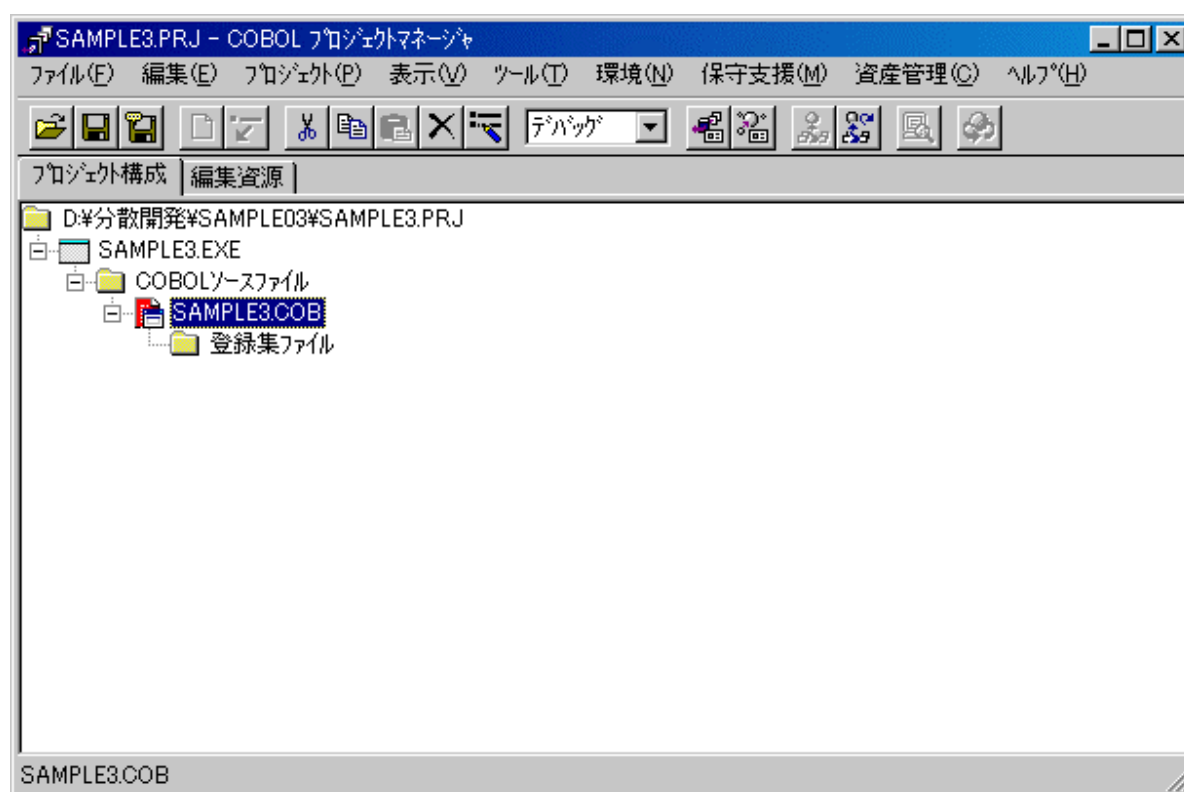


### 登録集ファイルフォルダの作成と登録集(COPY句)ファイルの追加

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。

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。

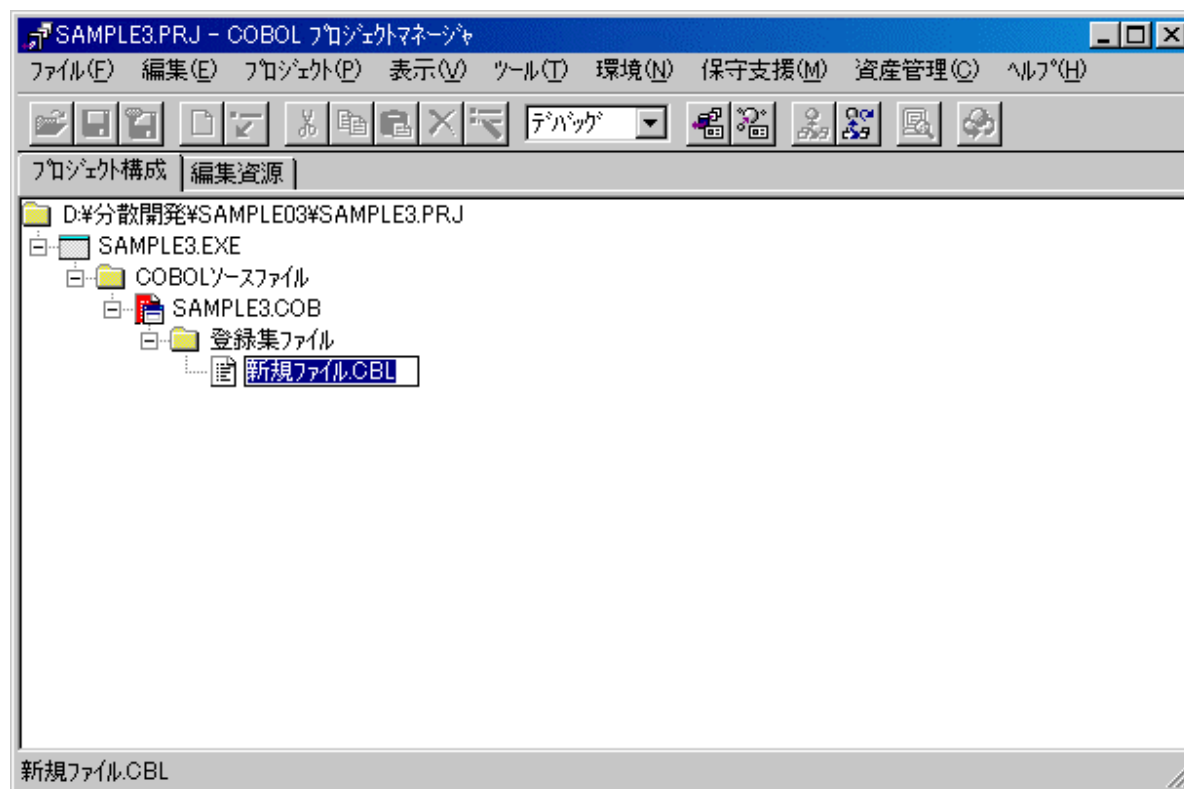
図3-10 追加された“登録集ファイル”フォルダ



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3-11 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。
- 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。

- 各種定義体ファイル  
COBOLで使用する定義体ファイルを以下に示します。
  - 画面帳票定義体ファイル
  - ファイル定義体ファイル
 定義体ファイルは、COBOL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。
- オブジェクトファイル  
COBOLで作成したオブジェクトファイルおよび他言語のオブジェクトファイルが指定でき

ます。



### 注意

Windowsシステムで使用するオブジェクトファイルは、UNIX系システムでは使用できません。あらかじめ、UNIX系システムで使用可能なオブジェクトファイルを用意しておいてください。

- インポートライブラリ  
動的リンク構造の実行可能プログラムを作成する場合、プロジェクトで作成されるインポートライブラリが必要となります。  
インポートライブラリは、最終ターゲットファイル配下に“ライブラリ”フォルダを作成し、“ダイナミックリンクライブラリファイル名.LIB”というファイル名で作成します。
- その他のライブラリ  
NetCOBOLが提供するライブラリ以外のライブラリが指定できます。インポートライブラリと同様に、“ライブラリ”フォルダに登録します。

## 3.2.2 分散開発固有の設定



UNIX系プログラムの分散開発を実施する場合、プロジェクトに、以下に示す設定が必要となります。

- プロパティ

### 3.2.2.1 プロパティ

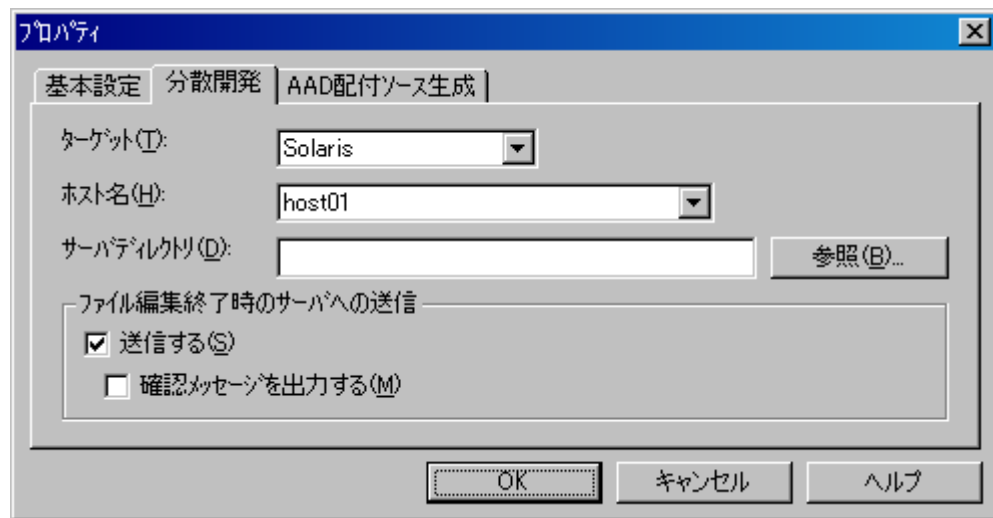
プロジェクトのプロパティに分散開発情報を設定することにより、そのプロジェクトで分散開発を行なうための、以下の機能が使用できるようになります。

- プログラム資源の送受信
- ターゲットビルド
- ビルド制御文生成

分散開発情報を設定するには、あらかじめサーバ連携情報を設定しておく必要があります。[参照] “2.3.2.1 [サーバ連携情報の設定](#)”

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”を選択して、プロジェクトを開きます。
2. [ファイル] メニューから“プロパティ”を選択します。[プロパティ] ダイアログが表示されます。

図3-12 サーバ連携情報ダイアログ



3. 「分散開発」ページで以下の情報を設定します。

#### 【画面の説明】

- ターゲット：
 

分散開発時のターゲットの種別を選択します。

  - なし  
Windowsシステムをターゲットとしたプロジェクトとなります。
  - Solaris  
Solarisをターゲットとしたプロジェクトとなります。
  - Linux(32ビット)  
Linux32をターゲットとしたプロジェクトとなります。
  - Linux(64ビット)  
Linux64をターゲットとしたプロジェクトとなります。
  - グローバルサーバ  
グローバルサーバをターゲットとしたプロジェクトとなります。
- ホスト名：
 

分散開発でターゲットとなるサーバのホスト名を選択します。  
〔サーバ連携情報〕ダイアログで指定したホスト名の一覧が表示されます。
- サーバディレクトリ：
 

サーバ上の起点となるディレクトリをフルパス名で指定します。  
ファイルの送信／受信、ビルド制御文生成、ターゲットビルドはこのディレクトリをサーバのカレントディレクトリとして処理します。
- 参照：
 

サーバの〔ディレクトリの参照〕ダイアログが表示されます。
- ファイル編集終了時のサーバへの送信：
 

ファイルの編集を終了または編集中のファイルを格納したときに、編集したファイルをサーバへ自動的に送信するか否かの情報を設定します。

  - 送信する：
 

チェックボックスをチェックすると、ファイルの編集を終了または編集中のファイルを格納したときに、編集したファイルをサーバへ自動的に送信します。  
ターゲットでSolaris、Linux(32ビット)またはLinux(64ビット)が選択されている場合に有効となります。  
送信の対象となるファイルは、〔送信〕ダイアログで送信対象として指示されているファイルであり、かつ、COBOLソース、登録集、インクルード、メイクファイル、画面帳票定義体ファイルです。



エディタのカスタマイズでPowerGEM Plusエディタ以外のエディタが定義されている場合は、画面帳票定義体ファイル以外のファイルは、この機能が有効となりません。

— 確認メッセージを出力する:

チェックボックスをチェックすると、サーバへファイルを送信する前に、送信するか否かを確認するメッセージが表示されます。

4. [プロパティ] ダイアログの [OK] ボタンをクリックします。

### 3.2.3 特殊なプロジェクトの作成



以下のようなアプリケーションを作成する場合は、通常のプロジェクトの設定のほかに、特別な設定が必要です。

- CORBAアプリケーション
- プリコンパイラを使用するアプリケーション
- Webアプリケーション

#### 3.2.3.1 CORBAアプリケーションの設定

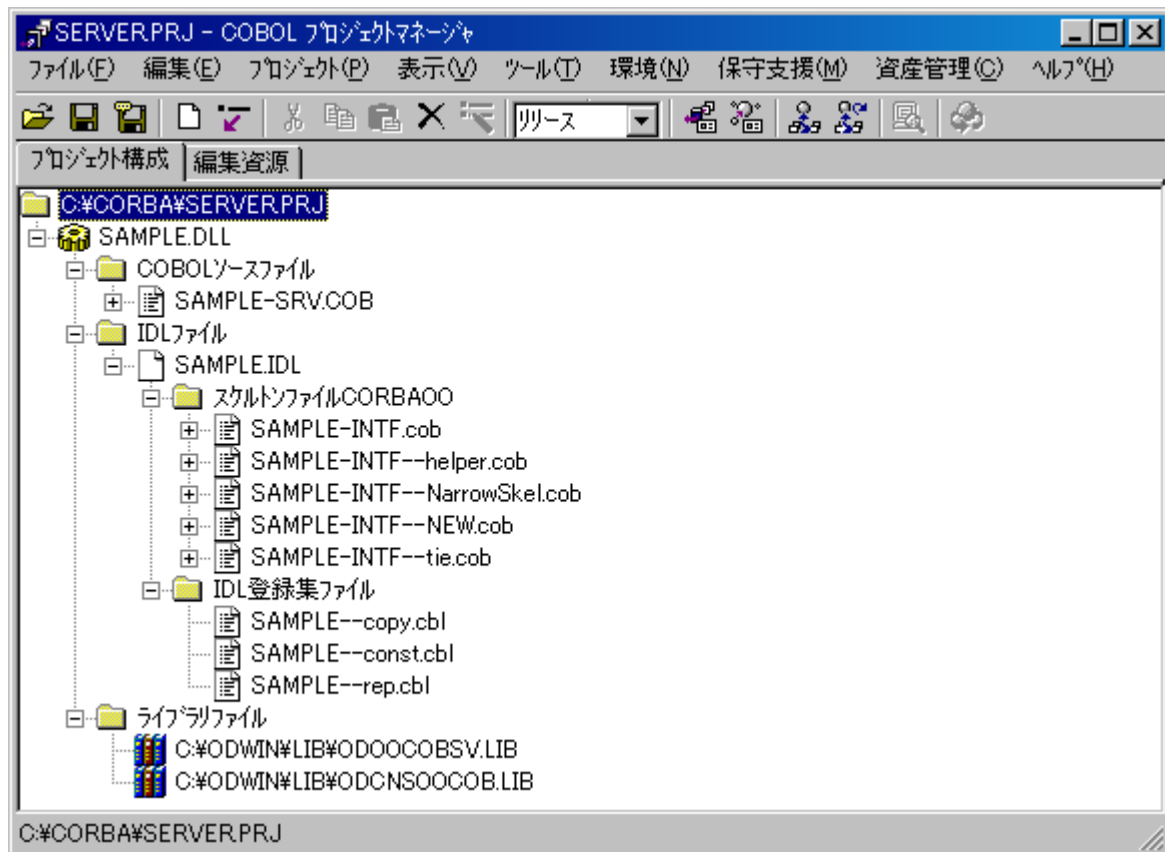
COBOLプロジェクトマネージャを使用して、Interstage配下で動作するCORBAアプリケーションを作成することが可能です。

CORBAアプリケーションを作成するには、IDLソースファイルをプロジェクトに登録して作成する方法と、登録しないで作成する方法があります。

##### IDLソースファイルを登録する場合

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] - [フォルダ作成] メニューから “IDL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 “IDLファイル” という名前のフォルダが追加されます。
3. [編集] メニューの “新規作成” または “追加” で、“IDLファイル” フォルダに、IDLソースファイルを登録します。
4. インクルードファイルがある場合は、IDLソースファイルを選択し、[編集] - [フォルダ作成] メニューから “インクルードファイル” を指定して、インクルードファイルのフォルダを登録します。さらに、インクルードファイルのフォルダに対して必要なインクルードファイルを [編集] メニューの “新規作成” または “追加” でツリーに登録します。
5. 3. で登録したIDLソースファイルを選択して、[編集] メニューから “Interstage” を選択すると、[Interstage] ダイアログが表示されます。
6. 表示された [Interstage] ダイアログに対して、必要な情報を登録します。
7. [Interstage] ダイアログの [OK] ボタンをクリックすると、プロジェクトマネージャの画面上に、IDLソースファイルに対する依存関係が表示されます。

図3-13 CORBAアプリケーション(IDLソースファイルを登録する場合)



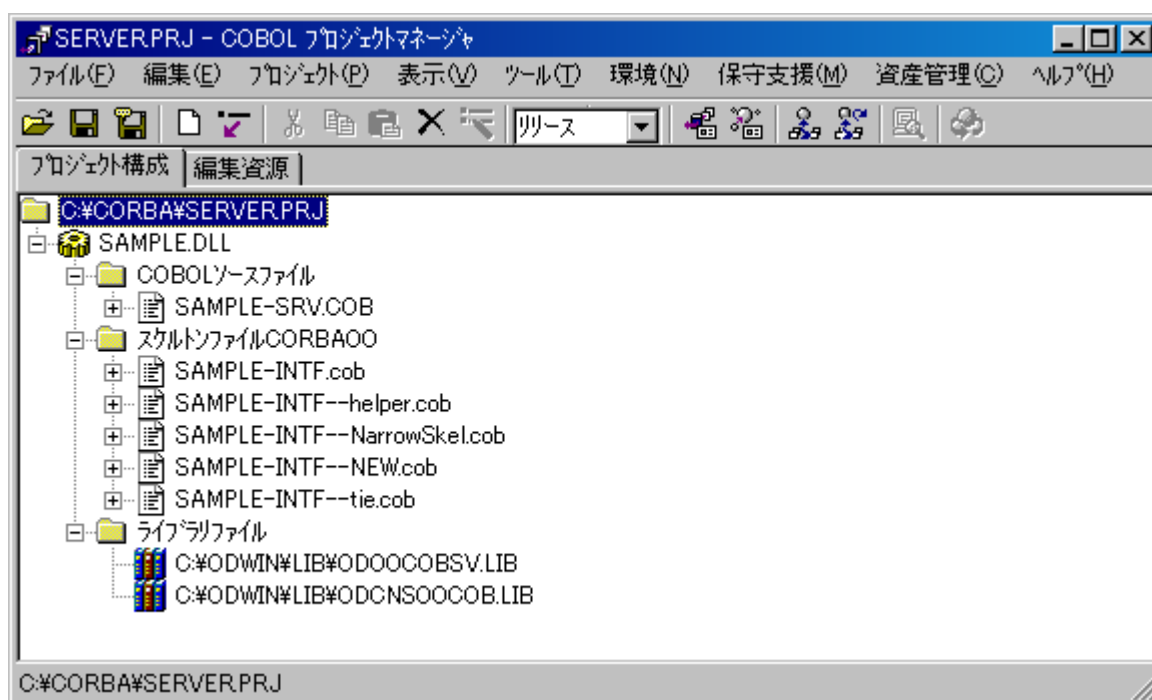
8. その他の必要なファイルを登録します。

### IDLソースファイルを登録しない場合

プロジェクト外でIDLソースファイルをIDLコンパイルし、生成したスタブファイルやスケルトンファイルをプロジェクトに登録してアプリケーションを作成する方法です。

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」メニューから「Interstage」を選択すると、「Interstage」ダイアログが表示されます。
3. 表示された「Interstage」ダイアログに対して、スケルトンファイルまたはスタブファイル、およびその他の必要な情報を登録します。
4. 「Interstage」ダイアログの「OK」ボタンをクリックすると、プロジェクトマネージャの画面上に、スケルトンファイルに対する依存関係が表示されます。

図3-14 CORBAアプリケーション(IDLソースファイルを登録しない場合)



5. その他の必要なファイルを登録します。

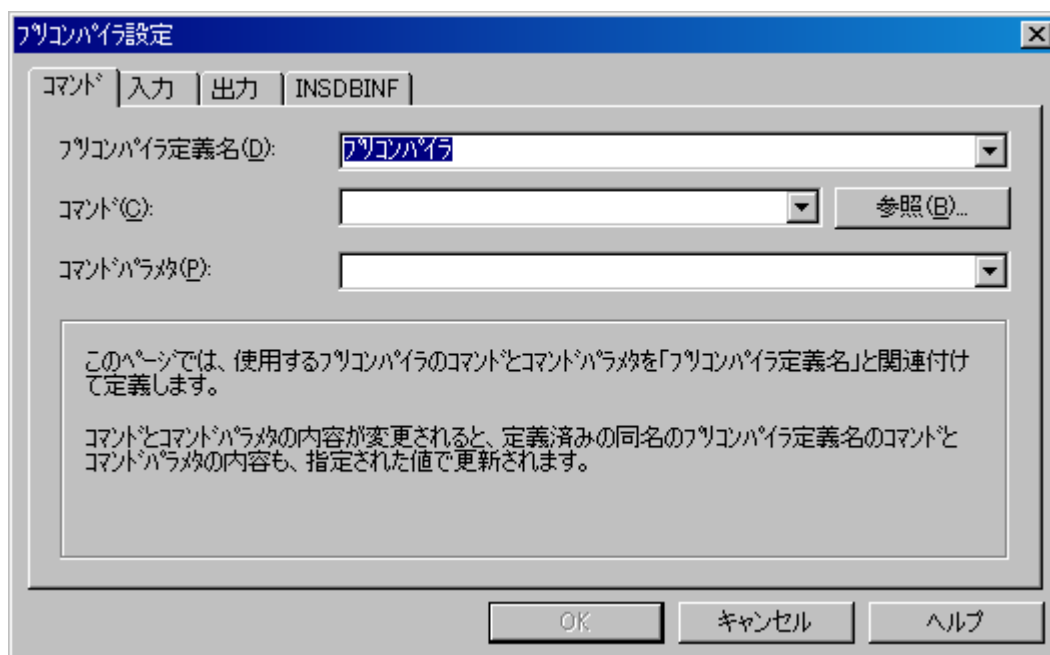
なお、プロジェクトマネージャでのCORBAアプリケーションの作成についての詳細は、“NetCOBOL 使用手引書”を参照してください。

### 3.2.3.2 プリコンパイラを使用するアプリケーションの設定

プロジェクトマネージャでは、各種のプリコンパイラを登録し実行することができます。プロジェクトにプリコンパイラを登録する手順を説明します。

1. COBOLプロジェクトマネージャの「プロジェクト構成」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」メニューから“プリコンパイラ”を選択すると、「プリコンパイラ設定」ダイアログが表示されます。

図3-15 プリコンパイラ設定ダイアログ



3. 表示された「プリコンパイラ設定」ダイアログに対して、必要な情報を登録します。
  - 「コマンド」ページで、プリコンパイラ定義名、コマンド、コマンドパラメータを設定します。
  - 「入力」ページで入力ファイル名を設定します。入力ファイルは複数設定できます。
  - 「出力」ページで出力ファイル名の拡張子、必要であれば出力ファイルの格納先フォルダ名を設定します。出力ファイル名は、入力ファイル名の拡張子とフォルダ名を、ここで設定したものと置き換えたものが登録されます。
  - INSDBINFコマンドを使用する場合には、「INSDBINF」ページで「INSDBINFコマンドを使用する」チェックボックスをチェックし、中間ファイル名の拡張子、INSDBINFオプション、必要であれば中間ファイルの格納先フォルダ名を設定します。中間ファイル名は、入力ファイル名の拡張子とフォルダ名を、ここで設定したものと置き換えたものが登録されます。
4. 「プリコンパイラ設定」ダイアログの「OK」ボタンをクリックすると、プロジェクトマネージャの画面上に、新規のフォルダおよびプリコンパイラに対する入出力ファイルが表示されます。
5. “インクルードファイル”フォルダをプリコンパイラの入力ファイルに対して作成し、インクルードファイルを設定します。
6. その他の必要なファイルを登録します。

なお、プロジェクトマネージャでのプリコンパイラを使用したアプリケーションの作成についての詳細は、“NetCOBOL 使用手引書”を参照してください。

### 3.2.3.3 Webアプリケーションの設定

プロジェクトマネージャを使用して、Webアプリケーションを作成することができます。Webアプリケーションを作成するには、Webアプリケーションウィザードを利用して、Webアプリケーションプログラムの雛型およびプロジェクトの情報設定を行います。

Webアプリケーションウィザードの使い方には、次の3つがあります。

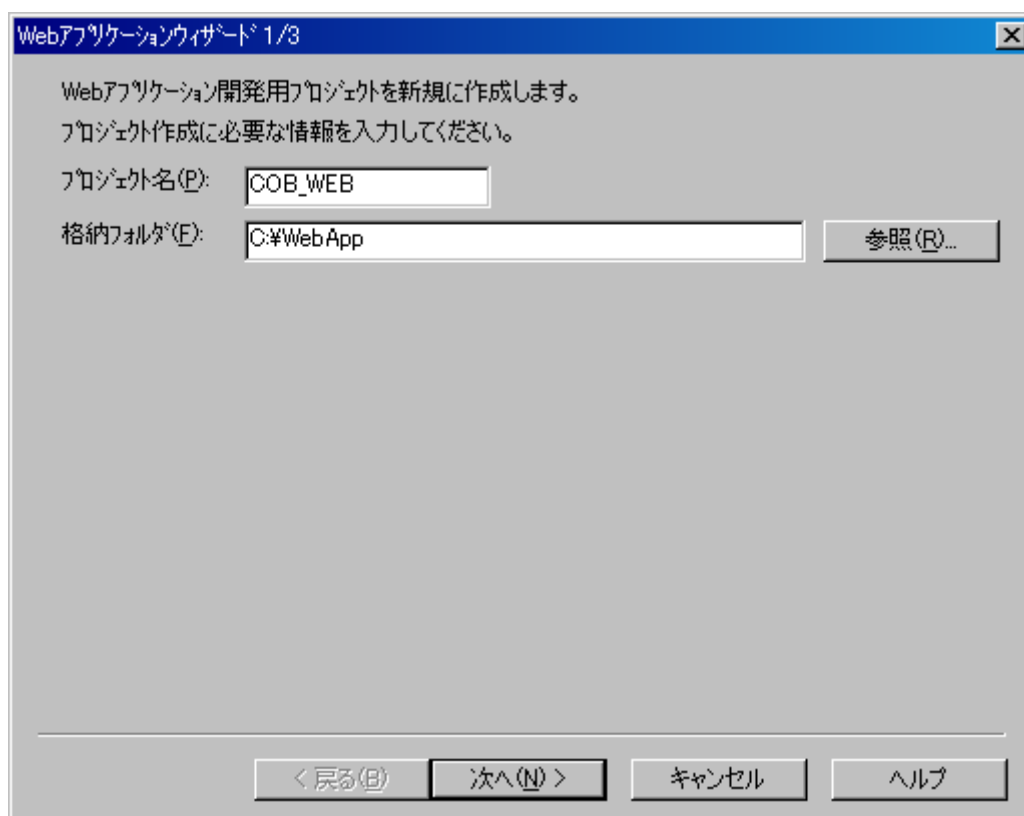
- a. Webアプリケーション開発用のプロジェクトとWebアプリケーションの雛形となるプログ

- ラムを生成する。
- b. 既存のプロジェクトに新しいターゲットファイルとして、Webアプリケーションを追加する。
  - c. 既存のプロジェクトにすでに存在するターゲットファイルの構成要素にWebアプリケーションの雛形などを追加する。

ここでは、a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“Webアプリケーションウィザード”を選択します。[Webアプリケーションウィザード]が表示されます。

図3-16 Webアプリケーションウィザード



2. Webアプリケーションウィザードは3つの画面と1つのダイアログボックスを持ちます。
  - 画面1ではプロジェクト関連の情報を設定します。必要な設定をしたら、[次へ]ボタンをクリックします。
  - 画面2では、実行ファイル名、使用するWebサブルーチンの種類など、生成するアプリケーションの雛形についての基本情報を設定します。必要な情報を設定したら、[次へ]ボタンをクリックします。
  - 画面3では、ウィザードが生成するWebアプリケーションの雛形の生成方法を指定します。
3. 各画面で必要な設定を行なった後、[完了]ボタンをクリックすると、Webアプリケーションの雛形を構成する各種のプログラム資産が生成され、プロジェクトに追加されます。
4. 生成された雛型プログラムに、必要なビジネスロジックを実装します。

WebアプリケーションおよびWebアプリケーションウィザードの詳細については、“COBOL Webサブルーチン使用手引書”を参照してください。

## 3.3 開発資産のWindowsクライアント環境への移行

OSIV系システムまたはUNIX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系システムまたはUNIX系システム上からWindows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”機能を使用する方法を説明します。

### 3.3.1 OSIVからのプログラミング資産の移行



OSIV系システムからWindowsシステムへ移行する資産として、次のものがあります。

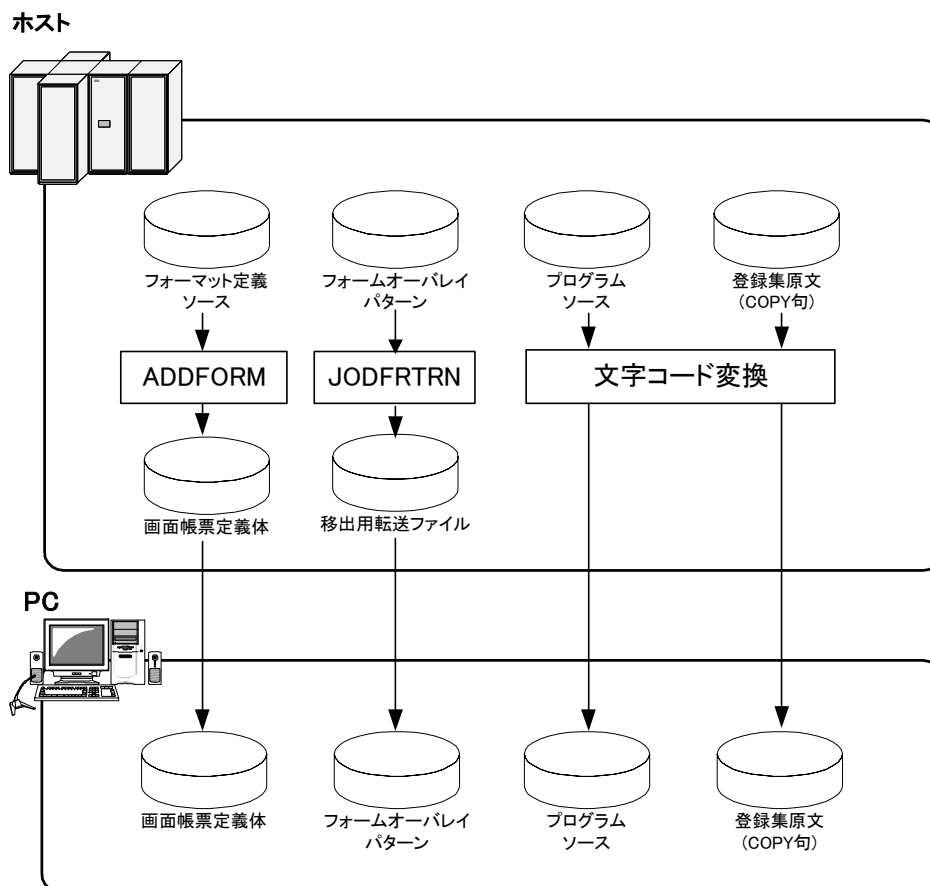
- COBOLソース
- 登録集原文 (COPY句)
- フォーマット定義体
- オーバーレイ定義体

COBOLソースや登録集原文 (COPY句)の一部を除き、OSIV系システムで使用していた資産は、Windowsシステムのものとは形式や使用法が異なるため、移行の前に、ツールを使用して形式を変換する必要があります。

OSIV系システムのデータセットから、Windowsシステムのファイルへ資産を受信するには、COBOLプロジェクトマネージャの受信機能を使用します。

資産の移行の概要を図に示します。

図3-17 OSIV系システムからWindowsシステムへの資産の移行



OSIV系システムからWindowsシステムへのプログラミング資産の移行についての詳細は、“NetCOBOL OSIV分散開発の手引き”を参照してください。

### 3.3.2 UNIX系システムからのプログラミング資産の移行



プロジェクトマネージャの受信機能を使用して、UNIX系システム上の資産を受信することができます。資産は、〔プロパティ〕ダイアログの〔分散開発〕ページで指定したホストから受信します。そのため、受信を行なう前に、プロジェクトのプロパティに分散開発情報を設定しておきます。

1. 〔プロジェクト〕－〔分散開発〕メニューから“受信”を選択します。〔受信〕ダイアログが開きます。

図3-18 受信ダイアログ

2. 〔受信〕ダイアログで以下の情報を設定します。

#### 〔画面の説明〕

- ホスト名：  
〔プロパティ〕ダイアログで選択したホスト名が表示されます。
- 受信元ディレクトリ名：  
受信元のディレクトリを指定します。初期値は〔プロパティ〕ダイアログの〔サーバディレクトリ〕で指定した値が表示されます。
- 受信元ファイル名：  
受信元のファイル名をフルパス名または〔受信元ディレクトリ名〕の相対パス名で指定します。  
フルパス名が指定された場合、〔受信元ディレクトリ名〕で指定されたディレクトリ名は無視されます。  
複数ファイル名を指定することができます。複数ファイル名を指定する場合は、ファイル名を“”で囲んで指定します。  
ファイル名にはワイルドカードを指定することができます。ワイルドカード文字には、\*と?を指定することができます。  
ファイル名の指定例を以下に示します。  
“/home/user1/src/test1.cob” “copy/\*.cbl” “readme.txt”
- 受信元ファイル名の〔参照〕：  
受信元となるサーバの〔ファイルの参照〕ダイアログが表示されます。
- 受信先フォルダ名：  
受信先のフォルダ名を絶対パス名またはプロジェクトのカレントフォルダの相対パス名で指定します。  
受信先フォルダ名が指定されていない場合は、プロジェクトのカレントフォルダが受信先

となります。

- 受信先フォルダ名の〔参照〕：  
受信先となる〔フォルダの参照〕ダイアログが表示されます。
  - データの種別：  
テキストまたはバイナリを選択します。テキストを選択した場合は、〔サーバ連携情報〕ダイアログで指定されたコード変換の手順でコード変換されます。  
COBOLソースや登録集を受信する場合は、〔テキスト〕を選択します。定義体を受信する場合は、〔バイナリ〕を選択します。その他のファイルについては、受信するファイルにあわせて選択します。
  - 上書き：  
受信先に既存のファイルが存在する場合、上書きするか否かを指定します。チェックボックスをチェックすると、受信先のファイルを上書きします。
3. 〔OK〕ボタンをクリックします。ホストから、資産の受信を開始します。



## 3.4 プログラミング

### 3.4.1 ソース・登録集原文の作成



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します。

- 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。
- 既存資産の保守および拡張の場合は、OSIV系システムからソース・登録集原文をWindowsシステムに転送し、プロジェクトに登録したパスに格納している。

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可能です。

図3-19 プロジェクトマネージャの〔プロジェクト構成〕ページ

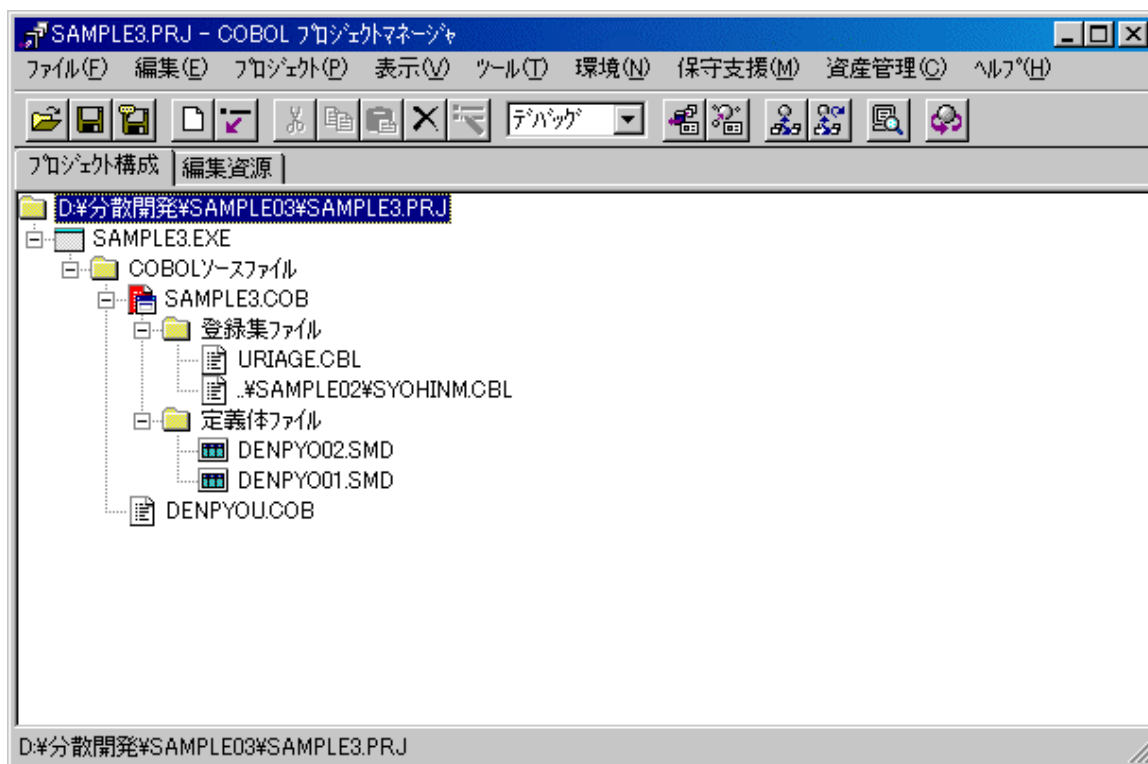
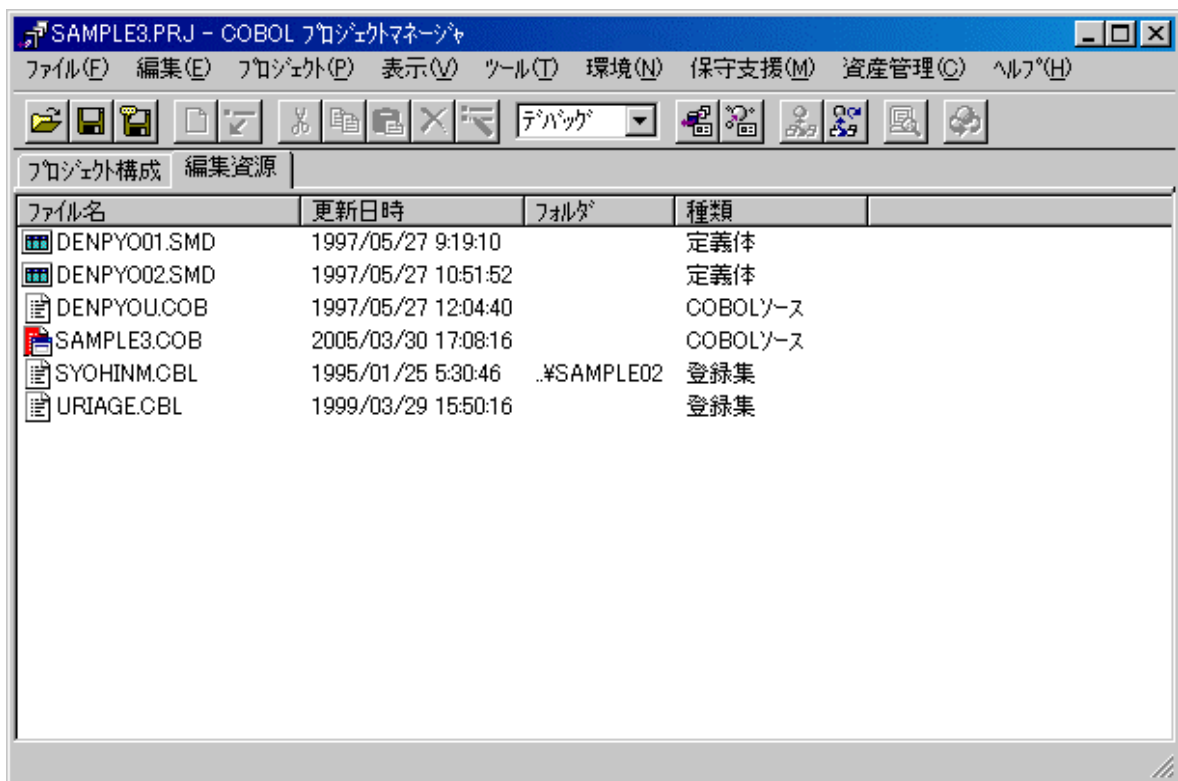


図3-20 プロジェクトマネージャの〔編集資源〕ページ



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。

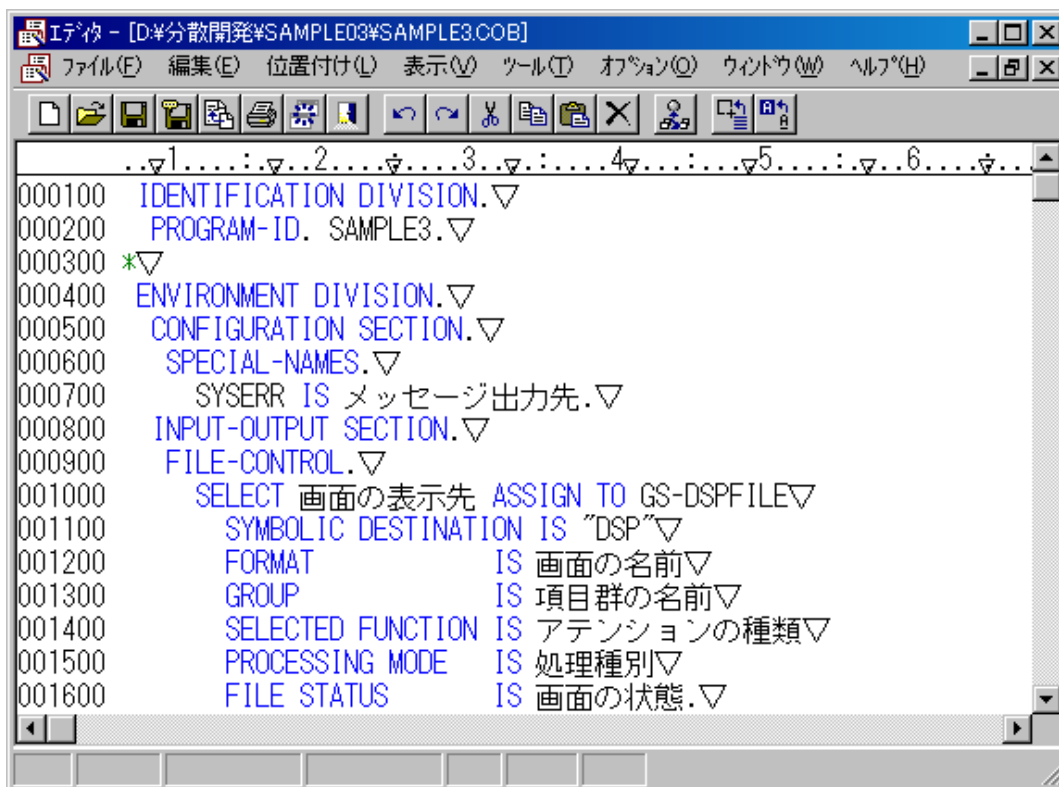
### 3.4.1.1 COBOLエディタ

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(エラージャンプ機能)

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3-21 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)

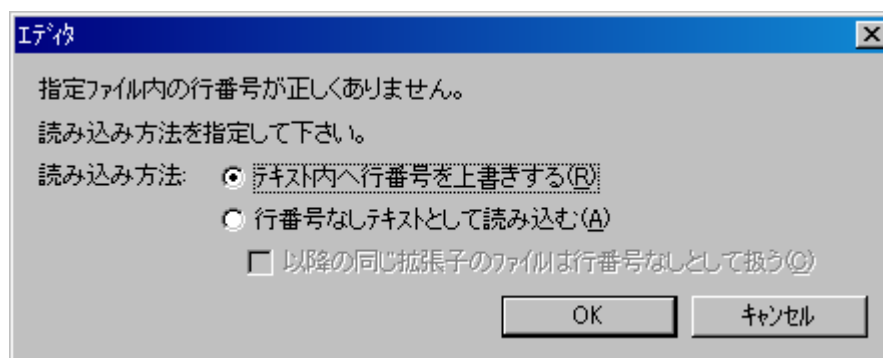


なお、このエディタの編集対象ファイルの管理方式は次の2つの形式があります。

- 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3-22 エディタの入力ファイル確認のダイアログ

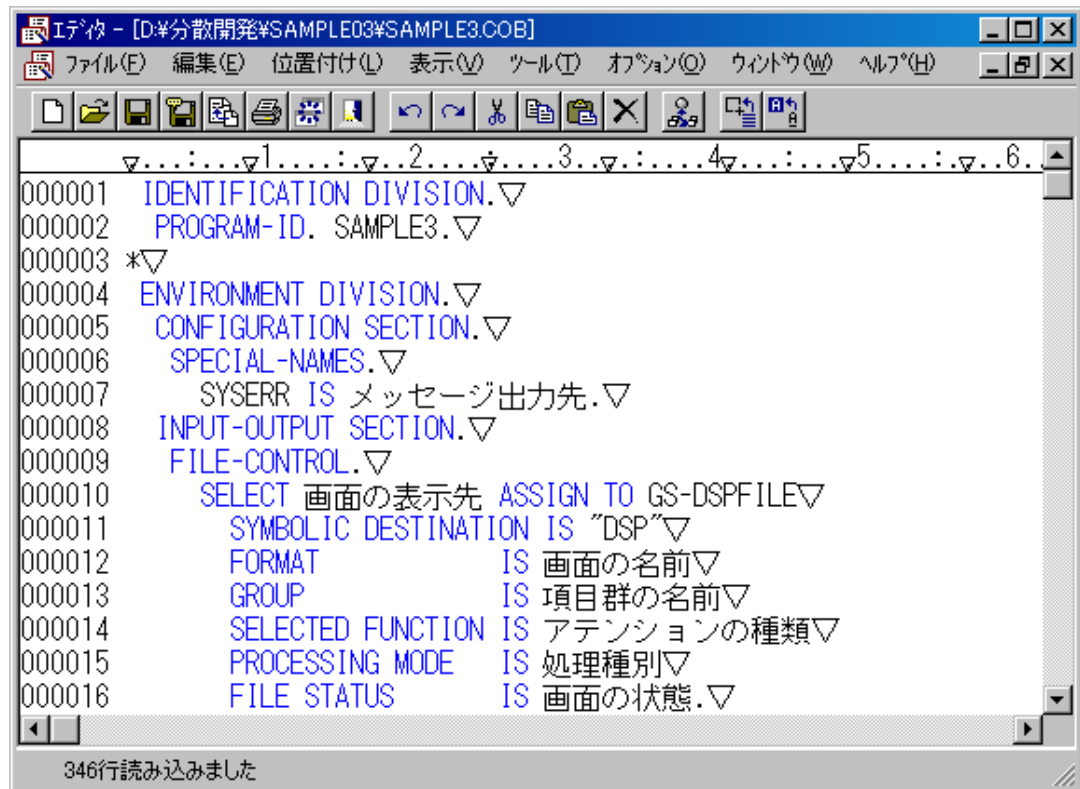


そのまま [OK] ボタンをクリックした場合、一連番号領域に新しい昇順に生成した行番号を上書きした上で、行番号付きテキストとしてファイルを開きます。

“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がふるまいが変わってくるので注意してください。

図3-23 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

ファイルを行番号付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)

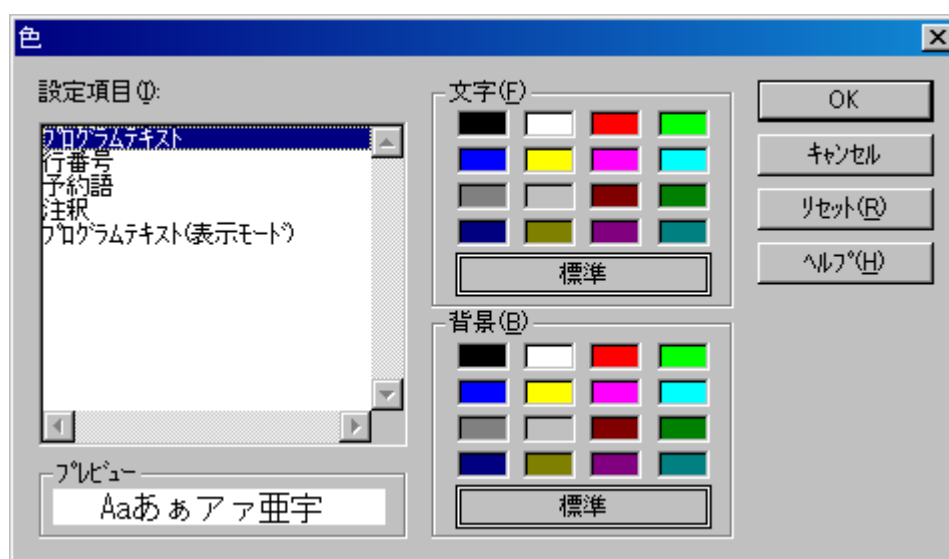


#### 注意

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す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、[色]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3-24 カラー構文表示のカスタマイズ用のダイアログ



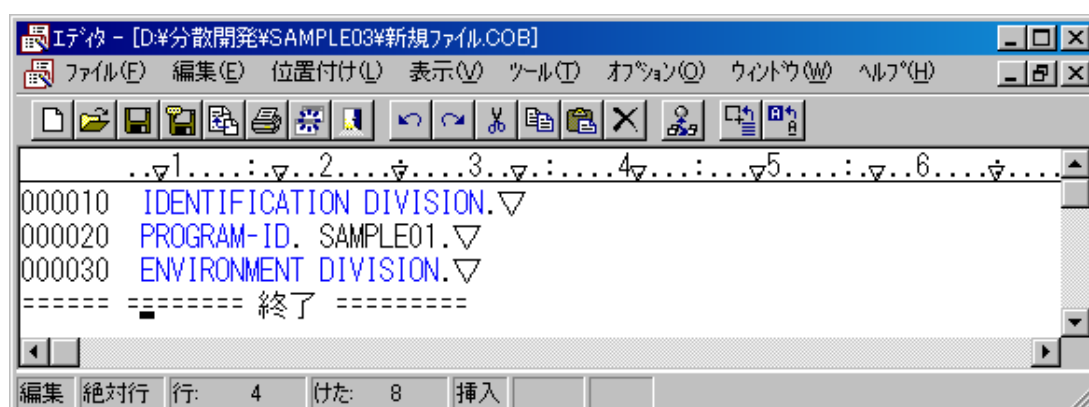
### テンプレート展開機能

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。

テンプレート展開機能の使用法を説明します。

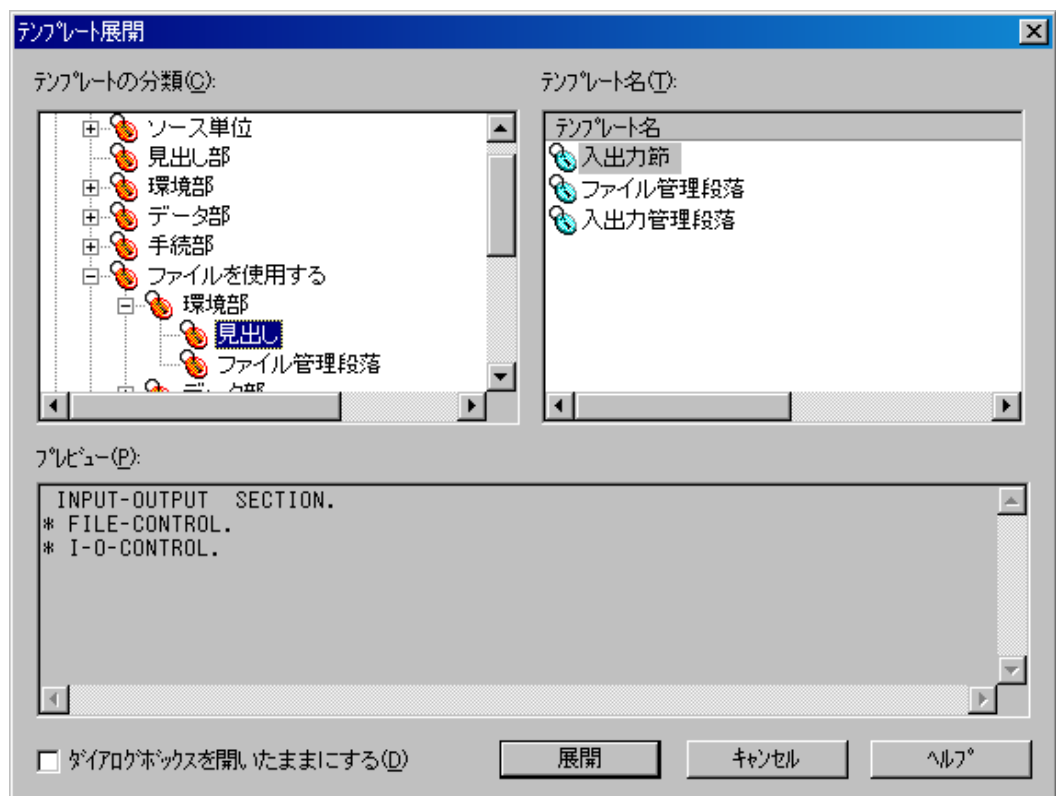
1. テンプレートを展開しようとする行にカーソルを合わせます。

図3-25 テンプレート展開前の状態



2. エディタの「編集」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3-26 「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、「展開」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3-27 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
- データ部
- 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3-28 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。別途設定が必要です。

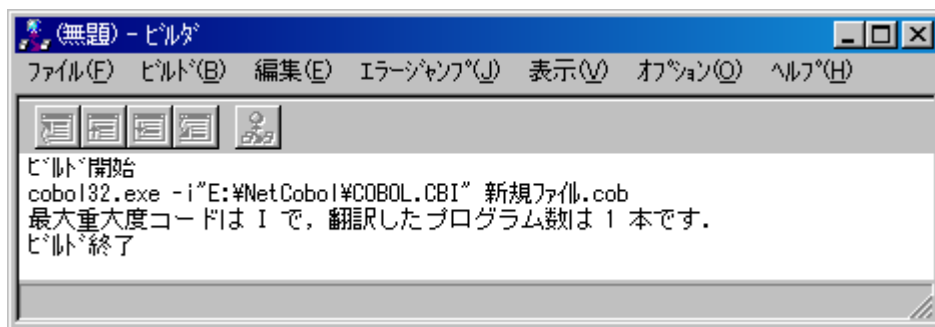
1. エディタの「ツール」メニューから“翻訳”を選択します。

図3-29 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」ウィンドウに表示します。

図3-30 簡易翻訳処理の実行結果



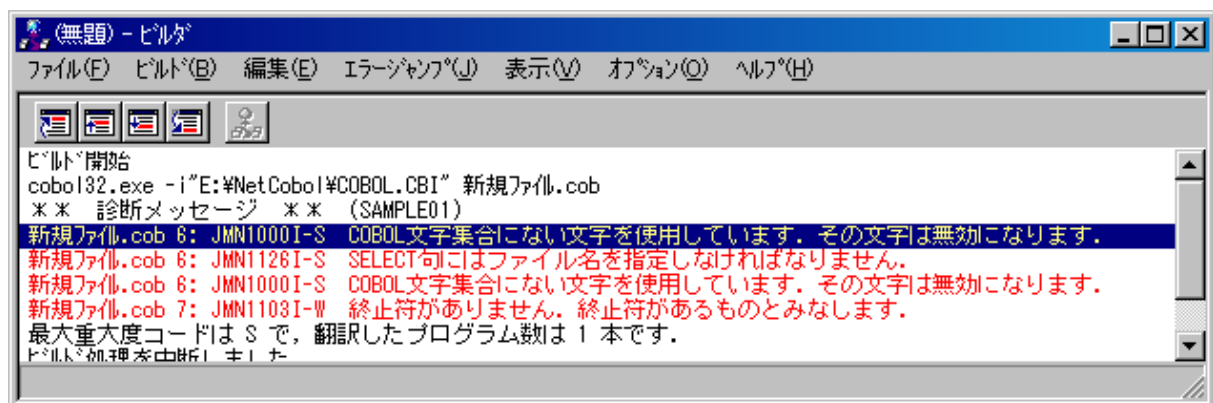
### メッセージ連携機能(エラージャンプ機能)

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。

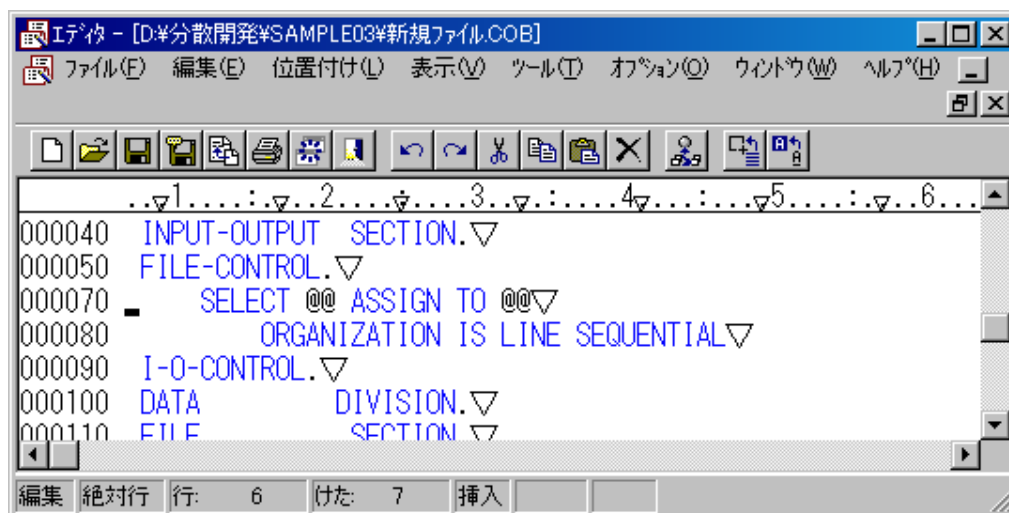
図3-31 修正対象の診断メッセージの選択





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3-32 メッセージ連携機能(エラージャンプ機能)の実行結果



### 3.4.1.2 COBOLエディタのカスタマイズ

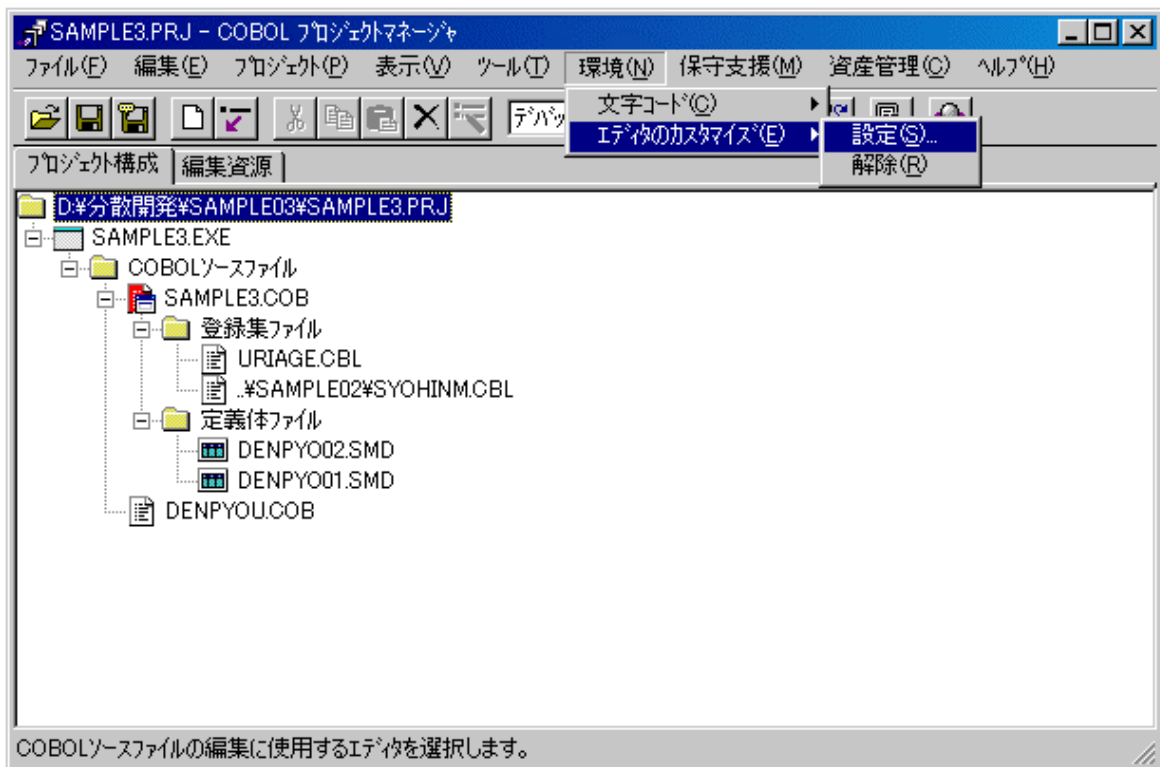
COBOLプロジェクトマネージャから起動するエディタは、初期状態ではNetCOBOLに組み込まれているエディタです。普段使い慣れているエディタが他に存在する場合、プロジェクトマネージャから起動するエディタをそのエディタで置き換えることもできます。これを“エディタのカスタマイズ”と呼びます。

“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

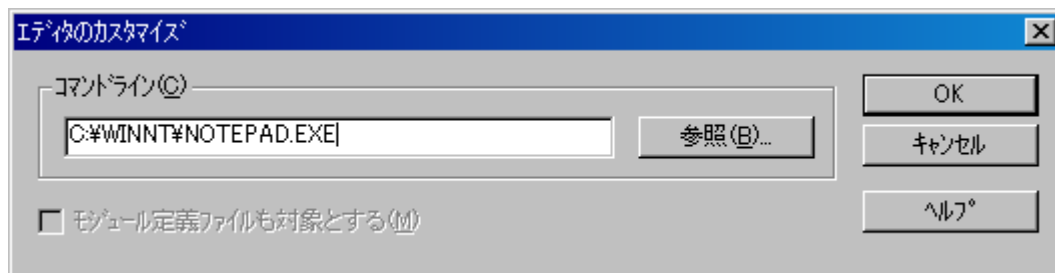
1. プロジェクトマネージャの〔環境〕－〔エディタのカスタマイズ〕メニューから、“設定”を選択します。

図3-33 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、置き換える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3-34 エディタのカスタマイズ例 (NOTEPADを選択)



3. [OK] ボタンをクリックします。

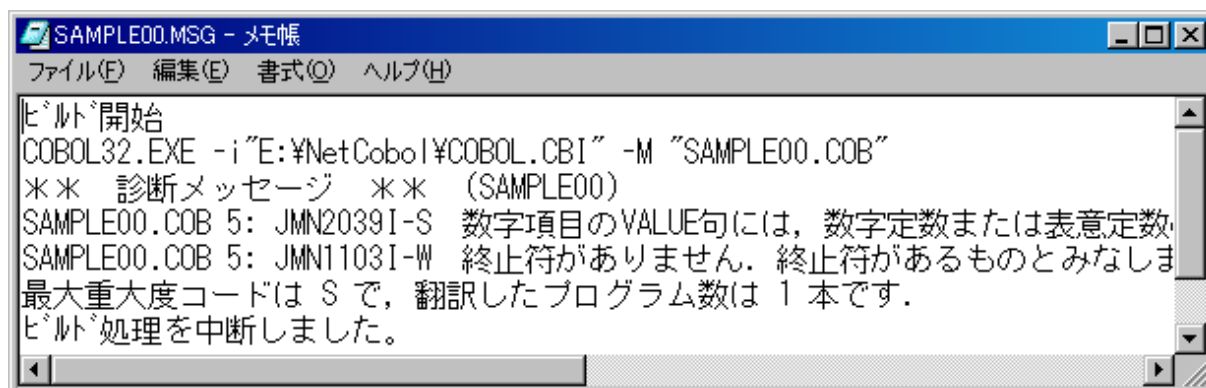
### エディタのカスタマイズ後の動作

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は次のように変わります。

- 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「エディタのカスタマイズ」ダイアログ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(エラージャンプ機能)は使用できなくなります。

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。

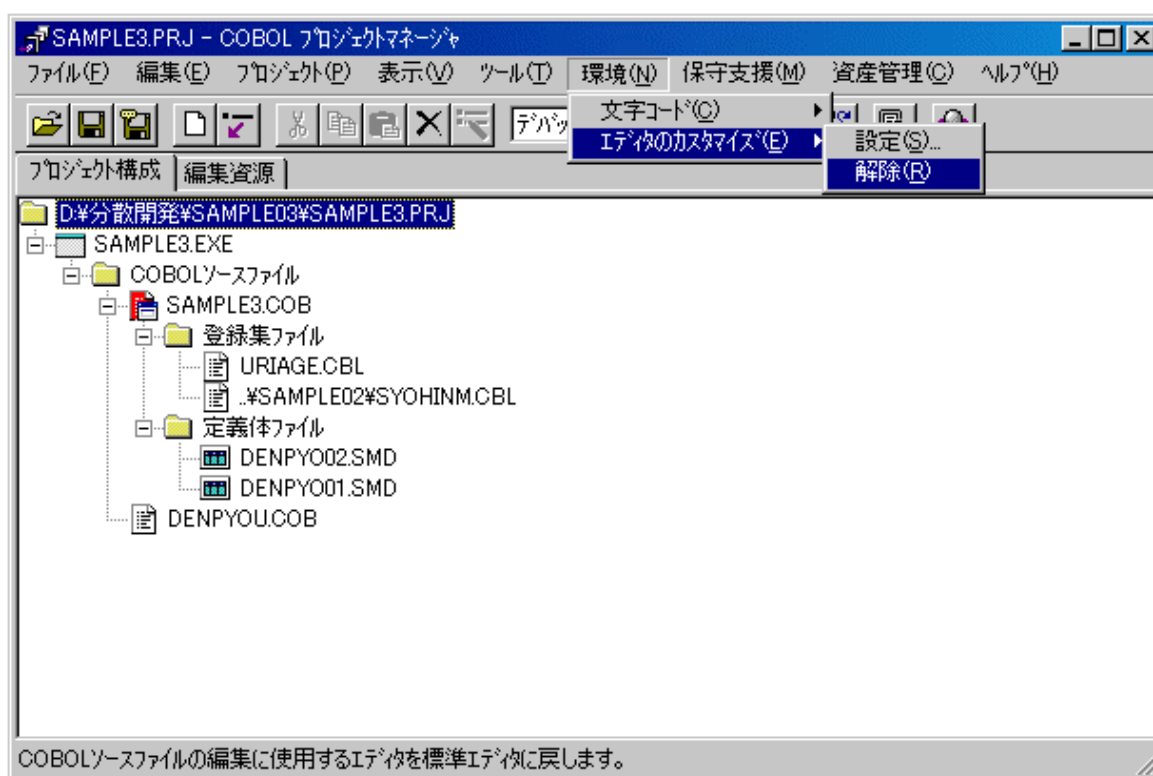
図3-35 カスタマイズしたエディタで翻訳結果を開いた場合の例



### 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、COBOL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3-36 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



COBOLプロジェクトマネージャに登録する以下の定義体は、Windows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。

- 画面帳票定義体

- フォームオーバーレイパターン
- ファイル定義体

COBOLプロジェクトマネージャに登録している、各種定義体を編集するには、[プロジェクト構成]ビューまたは[編集資源]ビューで、編集するアイテムをダブルクリックします。または、Windowsの[スタート]メニューより、各開発ツールを起動します。

各開発ツールの使用法についての詳細は、それぞれのマニュアルを参照してください。

定義体ごとの、開発ツールを以下に示します。

#### 3.4.2.1 画面帳票定義体の作成・編集

---

画面帳票定義体は、NetCOBOLに含まれる開発ツール「FORM」または「PowerFORM」を使用して作成・編集を行ないます。

画面帳票定義体を画面用として使用する場合は「FORM」、帳票用に使用する場合は「FORM」または「PowerFORM」を使用します。

#### 3.4.2.2 フォームオーバーレイパターンの作成・編集

---

フォームオーバーレイパターンを作成・編集するには、NetCOBOLに含まれる開発ツール「FORM」を使用します。

#### 3.4.2.3 ファイル定義体の作成・編集

---

ファイル定義体を作成・編集するには、PowerRW+に含まれる開発ツール「FILE」を使用します。

## 3.5 翻訳チェックとリンク

COBOLプロジェクトマネージャの機能を使用して、作成／修正したUNIX系プログラムの翻訳およびリンクを行います。

Windows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、可能な限り、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 UNIX系プログラムの翻訳



NetCOBOLにおいて、Windowsのプログラムの開発時もUNIX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。いくつか方法がありますが、ここではCOBOLプロジェクトマネージャからの操作を説明します。

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3-37 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。
4. 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を選択して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を移動することができます。

### 3.5.2 UNIX系プログラムのリンク



COBOLプロジェクトマネージャでのプログラムのリンクは、“ビルド”機能を使用して行います。

1. 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。

2. [プロジェクト] メニューから“ビルド”を選択します。
3. [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。

図3-38 プログラムのリンク処理(“ビルド”機能を使用)



なお、[プロジェクト] メニューから“ビルド”を選択して、翻訳からリンクまで一気に行なうこともできます。

## 3.6 Windowsクライアントでの単体テスト

### 3.6.1 Windowsクライアントでの単体テストの概要



分散開発では、単体テストまでをWindows上で行ないます。

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

ここでは、次のことを説明します。

- Windowsクライアントのデバッグ機能
- Windowsクライアントでの単体テスト実施の判断

#### 3.6.1.1 Windowsクライアントのデバッグ機能

NetCOBOLでは、Windowsシステム上でのデバッグの手段として、次の機能を提供しています。

- NetCOBOLのデバッグ機能
- COBOL対話型デバッガ機能

NetCOBOLのデバッグ機能は、NetCOBOL自身の持つデバッグ機能を使用するものです。具体的には、翻訳オプション等で指定するデバッグ機能です。

COBOL対話型デバッガ機能は、NetCOBOLに含まれる対話型デバッガを使用してデバッグを行なうものです。対話型デバッガは、GUIを使用した、使いやすいツールのため、効率的にデバッグ作業を行なうことができます。

#### 3.6.1.2 Windowsクライアントでの単体テスト実施の判断

Windows上で実行可能プログラムをビルドし、デバッグを行なうメリットは次の通りです。

- 単体テストでは、各作業者が開発するPCごとに独立したデバッグ環境を構築するため、各作業者が他の作業者に影響を与えることなく、並行してテストを行なうことができます。

反面、次のような場合には、Windows上での単体テストは困難です。

- 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、オペレーティングシステムの機能差の影響を受けるプログラムは、UNIX系システムとは実行結果が異なる。
- NetCOBOL以外の製品を、テストのためだけにWindows上に用意しては、コストと設定の手間がかかる。

単体テストは、先に述べたメリットから、Windows上で実行可能プログラムをビルドし、Windowsになるべく閉じた環境で単体テストを行なうことをお勧めします。

しかし、場合によっては、UNIX上へ資産を送信してターゲットビルドを行い、リモートデバッガを使用して単体テストを実施した方がよい場合もあります。

どちらの形態でテストを行なうかの判断基準の一つとして、以下のことが言えます。

- オペレーティングシステム間の機能差が処理に影響するのか
- UNIX上の製品を使用するなど、サーバ上の環境に依存した処理があるか

オペレーティング間の機能差については、“付録A [NetCOBOL製品の相違点](#)”を参照してください。

### 3.6.2 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、NetCOBOL自身の持つ次のデバッグ機能を使用することができます。

- 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、結果はファイルに出力されます。このため、対話型デバッガによるデバッグでは見つけにく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行後に発生する問題
- 他のプログラムから多数回呼び出され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

ここでは、これらの機能について概要を述べます。詳細については、“NetCOBOL 使用手引書”を参照してください。

#### 3.6.2.1 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)。

#### 3.6.2.2 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。

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

TRACE機能では、プログラムの異常終了時に、それまでに実行したCOBOLの文のトレース情報をファイルに出力します。出力されたトレース情報により、異常終了した文やそこまでの経緯を知ることができます。

#### 3.6.2.3 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
- プログラムの効率化を図りたい

翻訳オプションCOUNTが有効な場合、環境変数情報SYSCOUNTに指定されたファイルに、COUNT情報が出力されます。

利用者は、COUNT機能により、各文の実行頻度を的確に把握し、プログラムの最適化に役立てることができます。



### 3.6.3 対話型デバッガによるデバッグ

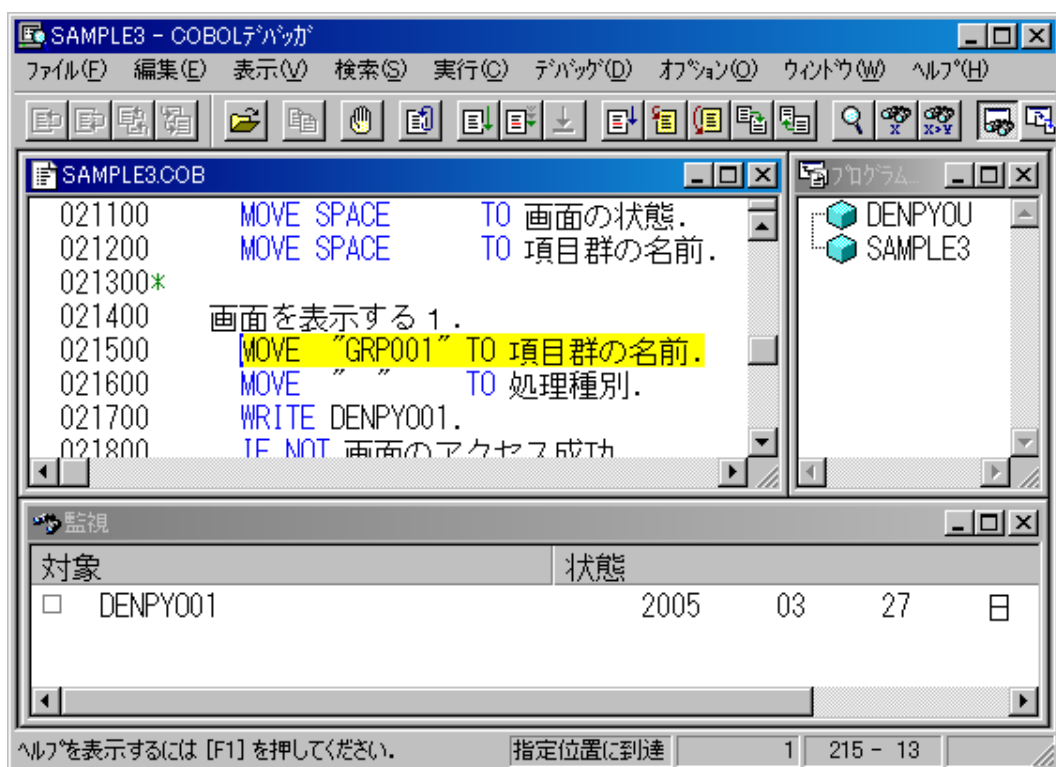


NetCOBOLに含まれる対話型デバッガを使用すると、実行可能プログラムをそのままデバッグの対象とし、プログラムの論理的な誤りを、プログラムを動作させながら検出することができます。ここでは、対話型デバッガについて、以下の説明をします。

- 対話型デバッガの特徴
- 対話型デバッグのための準備
- 対話型デバッグの開始

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3-39 対話型デバッガによるデバッグ例



#### 3.6.3.1 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ことで、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/ステップアウトなど）
- 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し、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。

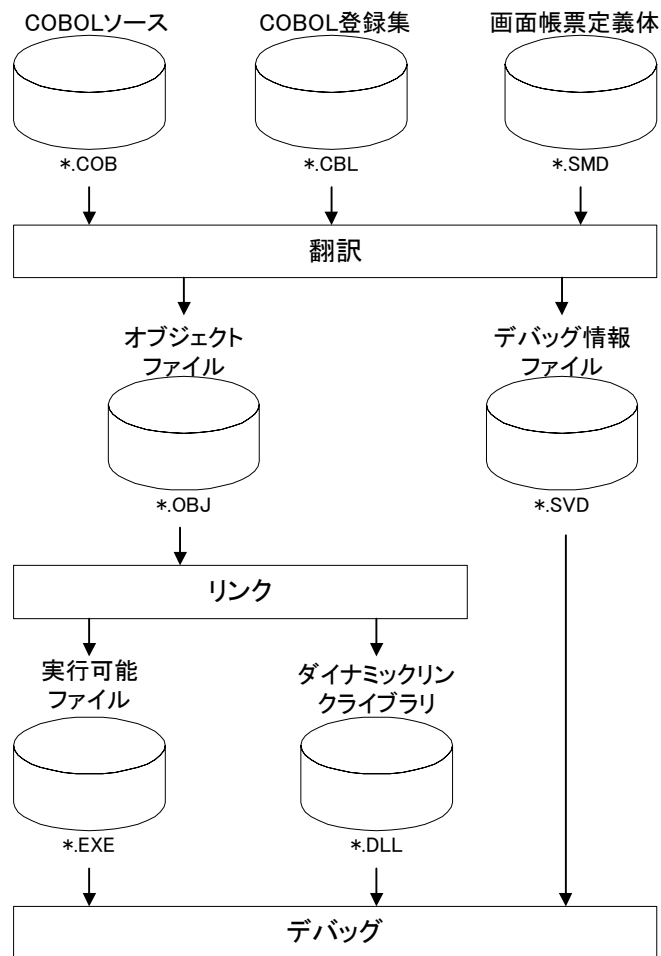
---

## 3.6.3.2 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。

図3-40 対話型デバッグに必要な資源



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。

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。

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

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場合は、この条件は自然に満たされます。

図3-41 “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

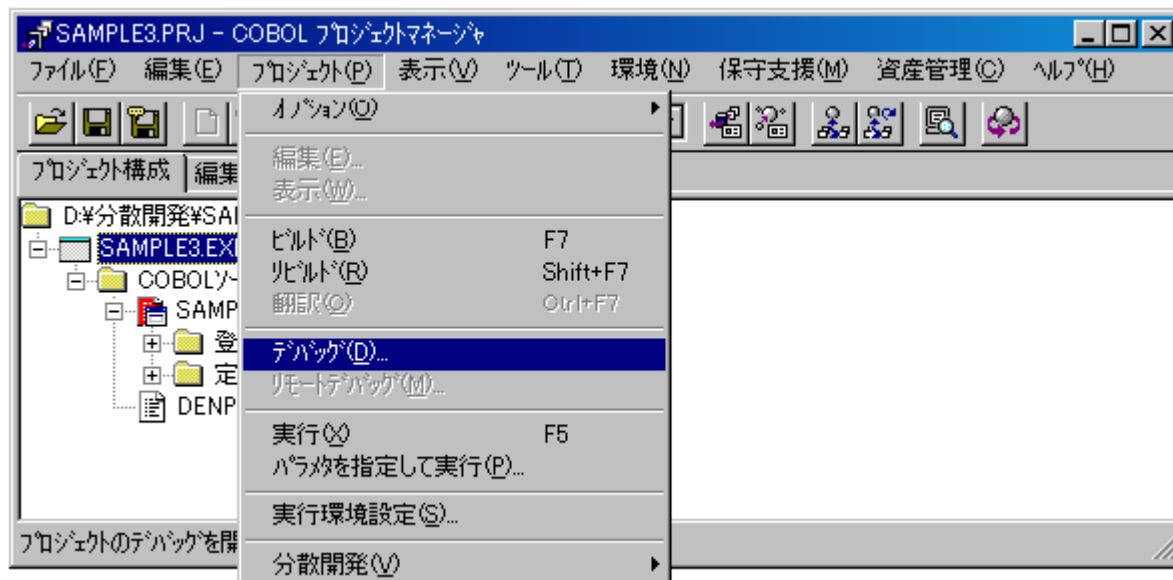


### 3.6.3.3 対話型デバッグの開始

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。

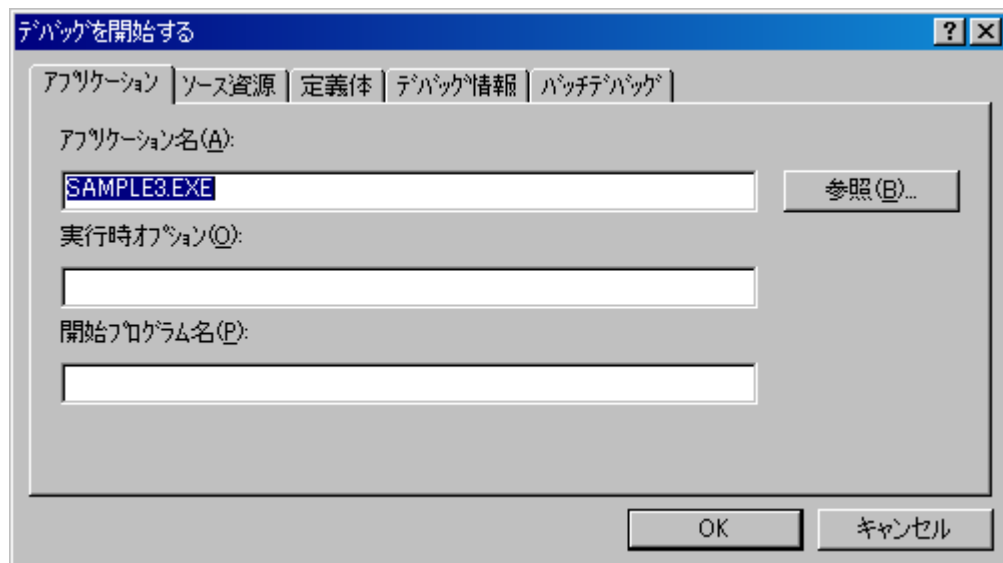
1. COBOLプロジェクトマネージャを起動します。プロジェクトの最終ターゲットファイルを選択します。
2. [プロジェクト] メニューの“デバッグ”を選択します。

図3-42 COBOLプロジェクトマネージャからデバッグを開始する



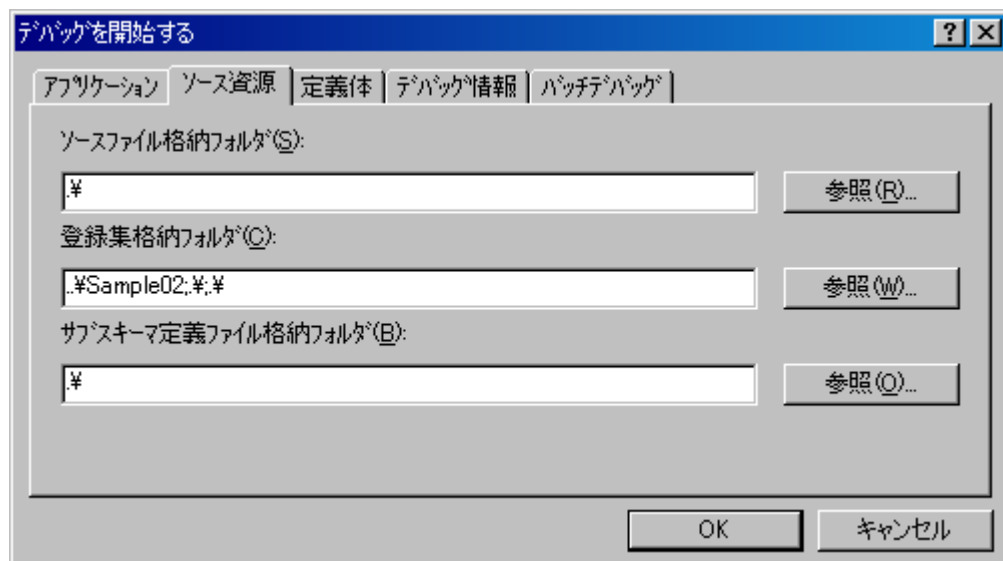
3. 対話型デバッガが起動して、“デバッグを開始する”ダイアログが表示されます。
4. プロジェクトの最終ターゲットが実行可能ファイルである場合、[デバッグを開始する]ダイアログの[アプリケーション名]に実行可能ファイル名が指定されています。プロジェクトの最終ターゲットがダイナミックリンクライブラリである場合、[アプリケーション名]は空白になっているため、テスト対象プログラム(ダイナミックリンクライブラリ)を呼び出す実行可能ファイル(テストドライバなど)を指定します。

図3-43 「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た資源(例えば、COBOL登録集)がある場合は、正しい格納フォルダを指定してください。

図3-44 必要な資源の指定



7. 「OK」ボタンをクリックするとデバッグが開始されます。



---

## 第4章 サーバ環境での開発作業

---

Windowsシステムでプログラミングが完了した資産を転送して、UNIX系システム上で翻訳・リンク・テストを行なう手順を説明します。

開発の目的により、必要となる作業は異なります。  
分散開発の適用範囲に合せ、必要な作業を行なってください。

⇒1. 2. 3. 1 [開発の目的から見た分散開発の適用範囲](#)



### 注意

本章の作業において、UNIX系システム上の指定で、ルートディレクトリおよびルートディレクトリ直下のファイルを指定することはできません。

---

## 4.1 サーバ環境へのプログラム資産の登録

UNIX系システムの分散開発では、Windowsシステムで開発したプログラム資産をUNIX系システムへ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。

- UNIX系システムのファイル送信

### 4.1.1 COBOLソース・登録集の送信



COBOLソース・登録集原文(COPY句)は、UNIX系システムでもWindows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、UNIX系システムではEUC/シフトJIS/Unicode、WindowsシステムではシフトJIS/Unicode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なわれます。

コード系についての詳細は“B.2 [COBOL製品のサポートするコード系](#)”を参照してください。



#### 注意

UNIX系システム上のファイル名はWindowsシステムと同一名となるように送信されますが、拡張子が“COB”，“COBOL”，“CBL”，“IDL”のファイルは、UNIX系システム上では拡張子部分が英小文字に変換されます。

#### 4.1.1.1 Windows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システムのファイルから、UNIX系システムのファイルとして送信します。

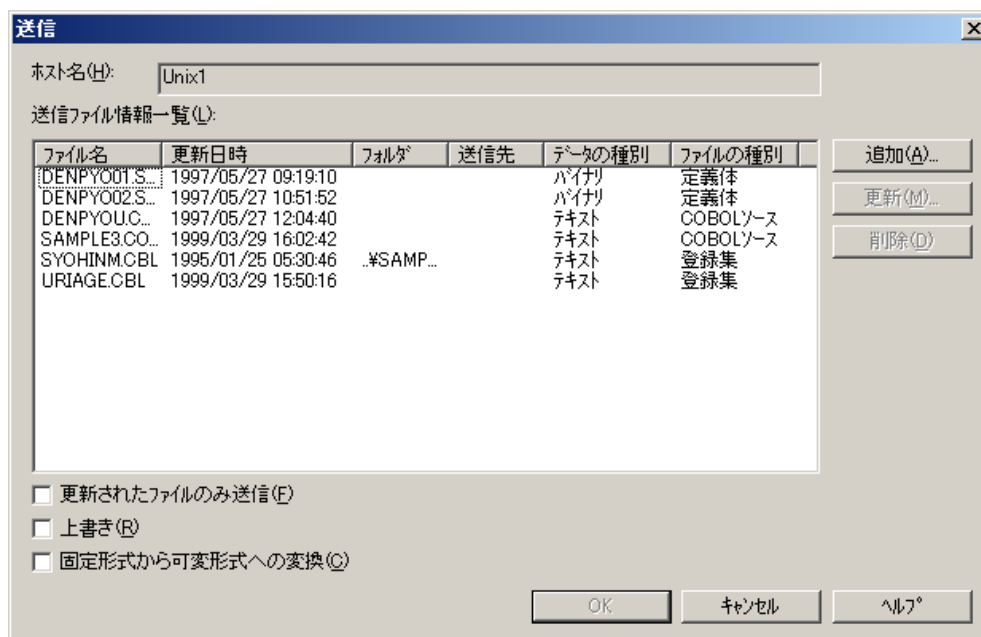
以下、その手順を説明します。

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、送信時の情報設定を行ないます。  
送信先の“ホスト名”は、〔プロパティ〕ダイアログの〔分散開発〕タブの〔ホスト名〕コンボボックスで指定したホスト名となります。

〔追加〕ボタンをクリックして〔送信ファイル情報一覧〕に送信するファイルの追加を行います。



図4-1 送信ダイアログ



3. 「送信ファイル情報の追加」ダイアログが表示されるので、送信するファイル名を「送信元ファイル」から選択します。

また、以下の設定も行ないます。

- 送信先ディレクトリ

送信先となるUNIX上のディレクトリをフルパス形式で指定します。

指定しない場合には、「プロパティ」ダイアログの「分散開発」タブの「サーバディレクトリ」エディットボックスに指定した、サーバディレクトリに送信されます。

- データの種類

ファイルの種別に合せて、テキストファイルとして送信を行なうか、バイナリファイルとして送信するかを選択します。“テキスト”を指定したファイルは、ファイルを転送する過程で文字コードの変換が行なわれます。

COBOLソース・登録集原文(COPY句)は文字コードの変換が必要となるため、“テキスト”を選択します。

図4-2 送信ファイル情報の追加ダイアログ



4. 「OK」ボタンをクリックすると、「送信ファイル情報の追加」ダイアログで選択したファイルと設定情報が「送信」ダイアログの「送信ファイル情報一覧」に追加されます。

5. 資産の内容や開発形態に合わせて、以下の指定を行ってください。

— 「上書き」 チェックボックス

送信先に既存のファイルが存在する場合、上書きするか否かを指定します。

チェックボックスをチェックすると、送信先のファイルを上書きします。

— 「更新されたファイルのみ送信」 チェックボックス

チェックボックスがチェックされている場合は、「送信ファイル情報一覧」の選択に関係なく、サーバの最終更新日時より後に更新されたファイルだけが送信されます。チェックされていない場合は、「送信ファイル情報一覧」で選択されているファイルだけが送信されます。

— 「固定形式から可変形式への変換」 チェックボックス

固定形式のCOBOLソースファイル、登録集ファイル中にマルチバイト文字がある場合、コード変換後に行の長さが80バイトを超えてしまうことがあります。この問題を解決するために、固定形式のファイルを可変形式に変換してファイルを送信します。可変形式に変換すると、プログラム識別番号領域は行内注記となります。

チェックボックスをチェックすると、COBOLソースファイル、登録集ファイルを固定形式とみなして可変形式へ変換して送信します。ただし、「サーバ連携情報」ダイアログの「ホストのコード」でシフトJISが選択されている場合は、チェックボックスをチェックしても固定形式から可変形式への変換はされません。

## 4.1.2 各種定義体の送信



バイナリファイルとして送信します。

「送信ファイル情報の追加」ダイアログの「データの種別」で“バイナリ”を選択してください。

それ以外の基本的な手順はCOBOLソース・登録集の送信と同様です。

図4-3 送信ファイル情報の追加ダイアログ

ファイル名	フォルダ	ファイルの種別
DENPY002.SMD		定義体

送信先ディレクトリ(D):  参照(B)...

データの種別: ☐ テキスト(T) ☒ バイナリ(B)

OK キャンセル ヘルプ

## 4.2 ビルド制御文生成機能

ビルド制御文生成機能は、UNIX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(Makefile)の雛型を生成し、指定したファイルに出力する機能です。

生成したビルド制御文をUNIX系システムに転送し、makeコマンドへの入力メイクファイルとすることで、ロードモジュールを作成することができます。

ビルド制御文の中には、Windowsシステムで開発したプロジェクトのプロジェクトファイルを元に、開発資産の依存関係や設定されているオプション情報が出力されます。

ただし、WindowsシステムとUNIX系システムの機能差から、UNIX系システムの情報として別途、設定・更新が必要な情報が存在します。これらの情報は、ビルド制御文生成ダイアログで設定・更新を行います。また、資産の構成によっては生成後のビルド制御文を直接編集して正しい内容に更新する必要もあります。

ビルド制御文は以下の操作を行った場合は、更新する必要があります。

- 1) プロジェクトマネージャのWindowsシステムの翻訳オプションを変更した。
- 2) 新たな資産を追加した。
- 3) UNIX系システムの翻訳オプション・リンクオプションを変更する。
- 4) プロジェクトファイルを現在のフォルダから他のフォルダに移動した。



**注意**

以下のプロジェクトに対しては、ビルド制御文の生成は行えません。

- 1) 以下のアイテムが最終ターゲットとしてプロジェクトツリーに登録されている
  - Webアプリケーション-ISAPI、WRB
  - AADアプリケーション
  - COMサーバアプリケーション
- 2) Windowsシステムの翻訳オプションに以下が指定されている
  - CREATE (REP)

### 4.2.1 ビルド制御文雛型の生成時の規則



以下では、プロジェクトの情報とビルド制御文生成ダイアログの情報から、どのような規則でビルド制御文の雛型を生成するかを説明します。

#### ● 翻訳対象ファイル

プロジェクトマネージャの以下のフォルダに登録されているファイルのうち、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。

- [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。
- [登録集ファイル] フォルダに登録されている登録集ファイル。
- [定義体ファイル] フォルダに登録されている画面帳票定義体ファイルおよびファイル定義体ファイル。
- [プリコンパイラ] フォルダの最下位階層の[プリコンパイラ] フォルダに登録されたプリコンパイラソースファイルと、プリコンパイラソースファイルの[インクルードファイル] フォルダに登録されている登録集ファイル。
- [IDLソース] フォルダに登録されているIDLソースファイルと、IDLソースファイルの[インクルードファイル] フォルダに登録されているインクルードファイル。
- IDLコンパイラにより翻訳を行わない場合の、設定されたスタブ/スケルトンファイル。

ただし、“ビルド制御文生成”ダイアログで〔送信済のファイルだけをビルドの対象とする〕を指定した場合は、“送信”機能で送信した際の送信情報を元に翻訳対象ファイルの絞込みが行われます。〔送信〕ダイアログの〔送信ファイル情報一覧〕に設定されていないファイルは翻訳対象ファイルとなりません。

ファイルの存在位置は、“送信”時に指定した“送信先”のディレクトリとなります。

〔送信済のファイルだけをビルドの対象とする〕を指定しない場合は、プロジェクトマネージャに登録されている上記のファイルすべてが翻訳対象ファイルとなります。

この場合、未送信のファイルの存在位置は“サーバディレクトリ”となります。

V7.2では未送信のファイルを翻訳対象ファイルとすることはできません。

詳細については、“付録C [V7.2との分散開発支援機能の機能差](#)”を参照してください。

#### ● 翻訳オプション

〔プロジェクト〕－〔オプション〕－〔翻訳オプション〕メニューから設定したWindowsの翻訳オプションを、UNIXの翻訳オプションの形式に変換したものをビルド制御文に展開します。

ただし、ディレクトリ名を指定するオプションについては、WindowsシステムとUNIX系システムの機能差から、〔ビルド制御文生成〕の〔オプション設定〕で更新が必要となります。

UNIX固有のオプションについては、同様に〔ビルド制御文生成〕の〔オプション設定〕で設定します。

また、登録集名の設定については、生成されたビルド制御文を編集する必要があります。

また、Windowsシステム固有の翻訳オプションについては展開されません。

図4-4 翻訳オプションの情報展開の流れ

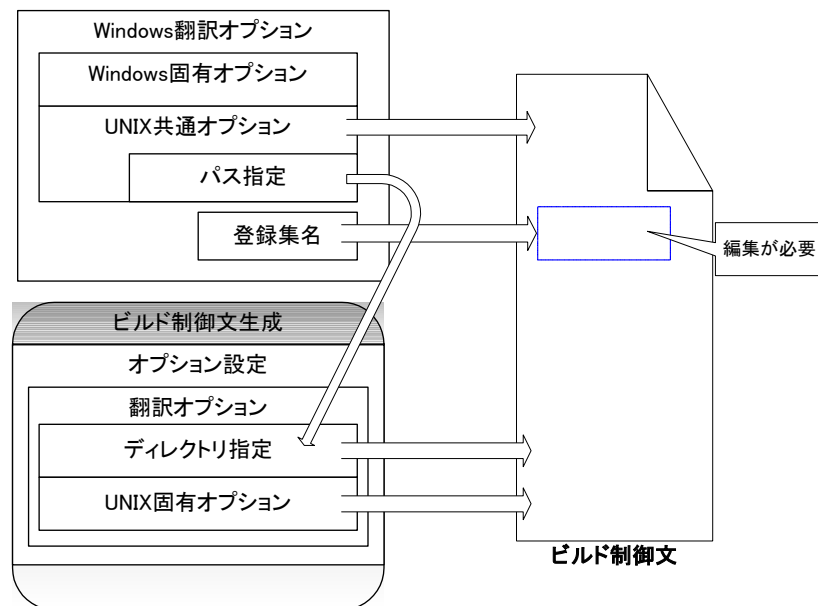


表4-1 ビルド制御文に展開する翻訳オプションの一覧

翻訳オプション名	説明	ビルド制御文生成で必要な操作
ALPHAL	英小文字を英大文字と同一視するかどうかを指定する。	－
ASCOMP5	2進項目の解釈方法を指定する。	－
BINARY	2進項目の割り付け領域長をワード単位、バイト単位にするかを指定する。	－

CHECK	CHECK機能を使用するかどうかを指定する。	-
CODECHK	実行時のコード系チェックの指定。	UNIX固有として、[オプション設定]で追加
CONF	新／旧規格の非互換を指摘する診断メッセージを出力するかどうかを指定する。	-
COPY	ソースプログラムリストへの登録集原文を表示するかどうかを指定する。	-
COUNT	COUNT機能を使用するかどうかを指定する。	-
CREATE	オブジェクト生成のために翻訳するか、リポジトリ生成のために翻訳するかを指定する。	-
CURRENCY	通貨編集用文字を指定する。	-
DLOAD	プログラム構造を指定する。	-
DUPCHAR	Unicode環境でソースを翻訳する際の全角ハイフン文字の扱いを指定する。	UNIX固有として、[オプション設定]で追加
EQUALS	SORT文での同一キーデータの処理方法を指定する。	-
FILEEXT	ファイル定義体ファイルの拡張子を指定する。	環境変数FFD_SUFFIXとして設定される
FILELIB	ファイル定義体ファイルの格納フォルダを指定する。	“ファイル定義体ファイルの入力先ディレクトリ” (-f)を更新する
FLAG	表示する診断メッセージのレベルを指定する。	-
FLAGSW	COBOL文法の言語要素に対する指摘メッセージを出力するかどうかを指定する。	-
FORMEXT	画面帳票定義体ファイルの拡張子を指定する。	環境変数SMED_SUFFIXとして設定される
FORMLIB	画面帳票定義体ファイルの格納フォルダを指定する。	“画面帳票定義体ファイルの入力先ディレクトリ (-m)”を更新する
INITVALUE	作業場所節でのVALUE句なし項目の扱い。	-
KANA	カナ文字の文字コードの扱いを指定する。	UNIX固有として、[オプション設定]で追加
LALIGN	連絡節データ宣言の扱いを指定する。	UNIX固有として、[オプション設定]で追加
LANGVL	ANSI COBOL規格の非互換項目をどの規格に従って解釈するかを指定する。	-
LIB	登録集ファイルの格納フォルダを指定する。	“登録集ファイルの入力先ディレクトリ (-I)”を更新する
LIBEXT	登録集ファイルの拡張子を指定する。	環境変数COB_LIB_SUFFIXとして設定される
LINECOUNT	翻訳リストの1ページあたりの行数を指定する。	-

LINESIZE	翻訳リストの1行あたりの文字数を指定する。	-
LIST	目的プログラムリストの出力を出力するかどうかを指定する。	-
MAIN	主プログラム/副プログラムの違いを指定する。	-
MAP	データマップリスト、プログラム制御情報リストおよびセクションサイズリストを出力するかどうかを指定する。	-
MESSAGE	オプション情報リスト、翻訳単位統計情報リストを出力するかどうかを指定する。	-
MODE	ACCEPT文の動作の指定を指定する。	-
NCW	日本語利用者語の文字集合を指定する。	-
NSPCOMP	日本語空白の比較方法を指定する。	-
NUMBER	行番号としてソースプログラムの一連番号領域の値を使用するかどうかを指定する。	-
OBJECT	目的プログラムを出力するかどうかを指定する。	“オブジェクトファイルの出力先ディレクトリ(-do)”を更新する NOOBJECTが指定されていた場合、“-do”は無効
OPTIMIZE	広域最適化を行うかどうかを指定する。	-
PRINT	各種翻訳リストの出力を出力するかどうかとその出力先を指定する。	“翻訳リストファイルの出力先ディレクトリ(-dp)”を更新する PRINTが指定されていない場合、“-dp”は無効
QUOTE/APOST	表意定数QUOTEの扱いを指定する。	-
RCS	実行時コード系を指定する。	-
REP	リポジトリファイルの入出力先フォルダを指定する。	“リポジトリファイルの入出力先ディレクトリの指定(-dr)”を更新する
REPIN	リポジトリファイルの入力元フォルダを指定する。	“リポジトリファイルの入力先ディレクトリ(-R)”を更新する
RSV	使用する予約語セットを指定する。	-
SAI	ソース解析情報ファイルを出力するかどうかを指定する。	“ソース解析情報ファイルの出力先ディレクトリ(-ds)”を更新する SAIが指定されていない場合、“-ds”は無効
SDS	符号付き10進項目の符号整形を実施するかどうかを指定する。	-
SHREXT	マルチスレッドプログラムにおける外部属性に関する扱いを指定する。	-
SMSIZE	PowerSORTが使用するメモリ容量を指定する。	-

SOURCE	ソースプログラムリストを出力するかどうかを指定する。	-
SQLGRP	SQLのホスト変数定義の拡張仕様を使用するかどうかを指定する。	-
SRF	正書法の種類を指定する。	-
SSIN	ACCEPT文のデータの出力先を指定する。	-
SSOUT	DISPLAY文のデータの出力先を指定する。	-
STD1	英数字の文字の大小順序の比較方法を指定する。	-
TAB	ソース中のタブ文字の扱いを指定する。	-
TEST	対話型デバッガおよび診断機能を使用するかどうかを指定する。	“デバッグ情報ファイルの出力先ディレクトリ(-dd)”を更新する TESTが指定されていない場合、“-dd”は無効
THREAD	マルチスレッドプログラム作成の指定する。	-
TRACE	TRACE機能を使用するかどうかを指定する。	-
TRUNC	桁落とし処理を実施するかどうかを指定する。	-
XREF	相互参照リストの出力するかどうかを指定する。	-
ZWB	符号付き外部10進項目と英数字項目の比較方法を指定する。	-
〔登録集名〕	登録集名	*1 ビルド制御文生成後に編集

\*1 出力されるパス名は“サーバディレクトリ”となります。他のパスに変更する場合は、ビルド制御文の生成後、ビルド制御文をエディタ等で直接編集する必要があります。

また、V7.2では、ビルド制御文への登録集名の出力は行われません。

詳細については、“付録C [V7.2との分散開発支援機能の機能差](#)”を参照してください。

#### ● リンクオプション

リンクオプションについては、WindowsシステムとUNIX系システムで機能が異なるため、Windowsのリンクオプションはビルド制御文に展開されません。

ただし、マルチスレッドモデルのプログラムをリンクするオプション(“-Tm”)は、Windowsの翻訳オプションに“THREAD (MULTI)”を指定した場合、自動的に出力されます。

〔ビルド制御文生成〕ダイアログの〔オプション設定〕で設定した情報および、プロジェクトツリーに登録されたNetCOBOL製品の提供するライブラリファイルを、cobolコマンドラインオプションとしてビルド制御文に展開します。

これらの指定は、プロジェクトに複数のターゲットファイルが含まれる場合、すべてのターゲットファイルに対して指定されます。特定のターゲットファイルにのみ指定を行わなければならない場合には、生成されたビルド制御文を直接編集して追加してください。

##### — NetCOBOL製品の提供するライブラリファイル

NetCOBOL製品の提供するライブラリファイルがプロジェクトに登録してある場合、リンクオプションに出力します。ただし、WindowsシステムとUNIX系システムではファイル名が異なるため、リンクオプションへは以下の下線部分のような形式で出力します。

例) COBLDFLAGS = \$(F3BICFPR\_LIB)

下線部分は、ビルド実行時に、UNIX系システムにインストールしている変換用ファ

イルを使用し、リンクオプションへ指定する形式へと変換されます。  
例に示したファイルの場合、次のように変換しています。

プロジェクトへの登録ファイル名	F3BICFPR.lib
ビルド制御文での表記	\$(F3BICFPR_LIB)
ビルド実行時の変換	-lFJPRT

変換用ファイルはインストールディレクトリ配下の“config/mkinc/LIBLIST”に提供されており、ファイルの内容は以下の通りです。

イコール(=)の左側の文字列の、“\_”を“.”に変換したものが、WindowsシステムにNetCOBOLが提供しているライブラリファイル名となります。

```
F1FVDENT_LIB = -ldbalcob
F3BICCOL_LIB = -lFJCOL-LINKED-LIST -lFJCOL-ALPHANUMERIC-MAP -lFJCOL-IDENTITY-SET
F3BICFPR_LIB = -lFJPRT
F3BIJART_LIB = -ljjart
F3BICWSR_LIB = -lcobw3cgi
F3BIFCFA_LIB = -lcobfa
F3BIFUTC_LIB =
F3BINSRT_LIB = -lcobw3saf
F3BISAPI_LIB =
```



#### 注意

V7.2のNetCOBOLでは、NetCOBOL製品の提供するライブラリファイルの出力は行われません。  
V7.2での指定方法については、“付録C [V7.2との分散開発支援機能の機能差](#)”を参照してください。

#### ● オブジェクトファイル名

翻訳対象ファイルのCOBOLソース、スタブ/スケルトン(IDLあり/なし)、プリコンパイラ展開ソースから生成されます。

COBOLソースファイルの拡張子が.cob, .cbl, .coblの場合、COBOLソースの拡張子を.oに置き換えたものとなり、COBOLソースファイルの拡張子が.cob, .cbl, .coblの以外場合、COBOLソースファイル名に拡張子.oを付加したものとなります。

(例: foo.cob → foo.o, foo.abc → foo.abc.o)

また、オブジェクトファイルの格納先は翻訳オプション“オブジェクトファイルの出力先ディレクトリ(-do)”を指定した場合は指定したディレクトリとなります。

指定していない場合には、ソースファイルと同じディレクトリとなります。

#### ● リポジトリファイル名

プロジェクトマネージャに登録されている、COBOLソースファイル、スタブ/スケルトンファイル(IDLソースあり/なし)、プリコンパイラ展開ソースのターゲットリポジトリファイル名より生成します。

生成するソースファイルが翻訳対象ファイルでない場合には出力されません。

ビルド時に生成されるリポジトリファイル名は、翻訳オプションALPHALの指定に従って大文字/小文字が決定されますが、ビルド制御文に出力されるリポジトリファイル名はプロジェクトマネージャに登録されているファイル名がそのまま出力されます。プロジェクトマネージャに登録されているリポジトリファイル名とビルド時に生成されるリポジトリファイル名が異なる場合は、生成されたビルド制御文を修正してください。

⇒ 4.2.3 [生成したビルド制御文雛型とその修正](#)

また、リポジトリファイルの格納先は翻訳オプション“リポジトリファイルの入出力先ディレクトリの指定(-dr)”を指定した場合は指定したディレクトリとなります。

指定していない場合には、ソースファイルと同じディレクトリとなります。



● Interstageのスタブ/スケルトンファイル

IDLコンパイラによる翻訳を行う場合は、プロジェクトマネージャに登録されているIDLコンパイラ(IDLcおよびtdc)が作成するスタブ/スケルトンファイル名が出力されます。ファイルの格納先はIDLソースと同じディレクトリとなります。

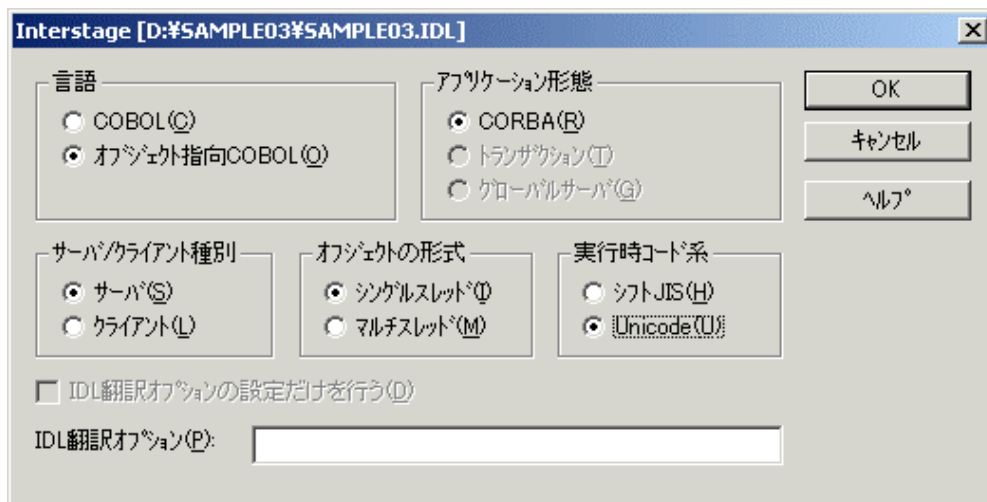
IDLコンパイラにより翻訳を行わない場合は、設定されたスタブ/スケルトンファイル名が“サーバディレクトリ”に出力されます。

IDLコンパイラのコマンドオプションについては、プロジェクトマネージャに登録されているWindowsシステムの情報がそのまま出力されます。

ただし、WindowsシステムとUNIX系システムのIDLコンパイラの差異から、[Interstage]の指定が以下の条件に当てはまる場合には“-unic”オプションが追加されます。

- 1) [言語] が [オブジェクト指向COBOL] かつ
- 2) [アプリケーション形態] が [CORBA] かつ
- 3) [実行時コード系] が [Unicode]

図4-5 Interstageダイアログ



また、フォルダを指定するオプションについては、以下の規則で変換されます。

表4-2 IDLコンパイラにおけるフォルダ指定オプションの変換

オプション	意味	変換規則
-I	インクルードファイル検索ディレクトリ	a), c)
-T	一時ファイル作成ディレクトリ	b), c)

- a) インクルードファイルを転送した場合は、転送先ディレクトリとします。インクルードファイルを転送しなかった場合は、相対パス指定のときはディレクトリ構成をそのままにUNIX形式に変換し、相対パス以外の場合は削除します。
- b) 相対パスならディレクトリ構成をそのままにUNIX形式に変換し、相対パス以外なら“サーバディレクトリ”になります。
- c) 指定されているパス名が“”で囲まれている場合、UNIX形式のパス名も“”で囲まれます。

● プリコンパイラ展開ソースファイル

プロジェクトマネージャに登録されているプリコンパイラ展開ソースファイル名が出力されます。ファイルの格納先ディレクトリは、プロジェクトマネージャ上の登録形式が相対パス指定の場合は、パス構成をそのままUNIX形式に変換して出力します。相対パス以外の場合は、“サーバディレクトリ”を出力します。ファイル名はプロジェクトマネージャに登録されているファイル名のまま出力します。

オリジナルソース(プリコンパイラフォルダで最下位階層に登録されているソース)が翻訳対象ファイルであるときに出力します。

ビルド制御文には、プリコンパイラのコマンドの情報も出力されます。

コマンドの設定はプリコンパイラ展開ソースファイル単位に、マクロ定義としてビルド制御文に出力されます。定義されるマクロ名は、“PRECOMFLAGSn”(nは1からの通し番号)となります。

プロジェクトマネージャで行った〔プリコンパイラ設定〕の設定情報を元に以下の規則で変換されます。

— プリコンパイラのコマンド名：

〔コマンド〕エディットボックスに指定した、プリコンパイラの実行形式ファイル名から、パスと拡張子を削除し、ファイル名を英小文字に変換したものを使用します。

— プリコンパイラコマンドのパラメタ

Windowsシステムで設定した内容をそのまま使用します。

ただし、以下のマクロ指定部分については、以下の様に展開されます。

- %INFILE%：  
オリジナルソース名
- %OUTFOLDER%：  
プリコンパイラ展開ソースファイルの格納先ディレクトリ
- %OUTFILE%：  
プリコンパイラ展開ソースファイル

● insdbinf生成ファイル

プロジェクトマネージャに登録されているinsdbinf生成ファイル名が出力されます。

オリジナルソース(プリコンパイラフォルダで最下位の階層に登録されているソース)が翻訳対象ファイルであるときに出力します。

Windowsのフォルダ指定オプションの内容は、以下の規則で変換されます。

**表4-3 insdbinfにおけるフォルダ指定オプションの変換**

オプション	意味
-I	インクルードファイル検索ディレクトリ

インクルードファイルを転送した場合は転送先ディレクトリとなります。転送しなかった場合は、相対パス指定のときはディレクトリ構成をそのままにUNIX形式に変換し、相対パス以外のときは削除します。

● 実行形式プログラム

プロジェクトマネージャに登録されている、EXEの最終ターゲット名より生成します。

出力されるファイル名は、最終ターゲット名から拡張子(.EXE)を取り除いたものとなります。(例: Foo.EXE → Foo)

実行形式プログラムを構成するオブジェクトファイルをターゲットとする依存関係がすべて出力された場合のみ出力されます。

● 共用オブジェクトプログラム

プロジェクトマネージャに登録されている、DLLの最終ターゲット名より生成します。

出力されるファイル名は、最終ターゲット名の先頭にlibを付加し、拡張子(.DLL)を.soに変換したものとなります。(例: Foo.DLL → libFoo.so)

生成するソースファイルがビルド制御文に出力される翻訳対象ファイルでない場合には出力されません。



## 1) 特殊文字を含む項目の変換

Makefileの文法上特別な意味を持つ文字が存在します。これらの文字はビルド制御文にそのまま出力すると、正しく動作しない可能性があります。

そのため、以下の文字については該当の文字の前に'¥'を追加(クォーティング)して出力を行っていますが、やむを得ない場合を除き使用しないことをお勧めします。

表4-4 クォーティングする特殊文字の一覧

TAB	空白	!	"	&		^	`	<	>
-----	----	---	---	---	--	---	---	---	---

また、以下の文字については特殊文字として判定されますが、ビルド制御文の出現個所の構成により対処方法が異なるため(対処不要も含む)、変換処理は行われずにそのまま出力されます。

表4-5 変換を行わない特殊文字の一覧

#	\$	%	'	(	)	=	~	-	{	[
+	*	}	;	:	]	?	.	/	¥	

※-(ハイフン)および+(プラス)は、ファイル名先頭に指定してある場合のみ特殊文字とする。

これらの特殊文字がビルド制御文に出力される場合、ビルド制御文生成実行時に確認を促すメッセージが出力されます。

生成されたビルド制御文の内容を確認し、必要がある場合には修正を行ってください。

修正の方法については、“4.2.3 [生成したビルド制御文雛型とその修正](#)”を参照してください。

## 2) ファイル名の大文字・小文字変換

ビルド制御文に出力するファイル名の太文字・小文字は、プロジェクトマネージャに登録されているWindowsシステムの資産のファイル名をそのまま使用します。しかし、以下のファイルについては、拡張子の小文字化が行われます。

- 拡張子が“.COB”，“.CBL”，“.COBOL”のファイル
- 拡張子が“.IDL”のファイル

## 3) ビルド制御文生成時に変換されるファイル名がソースファイル中に記述されている場合、UNIX系システム上のファイル名とソースファイル中のファイル名が一致しない場合があります。その場合は、ソースファイル中のファイル名の記述をUNIX系システム上の実際のファイル名に修正してください。

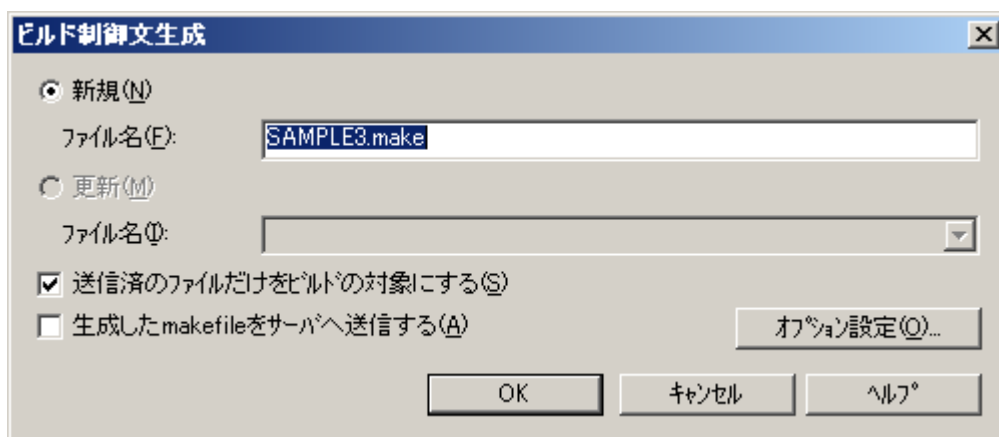
## 4.2.2 ビルド制御文雛型の生成手順



ビルド制御文の雛型の生成は次の手順で行います。

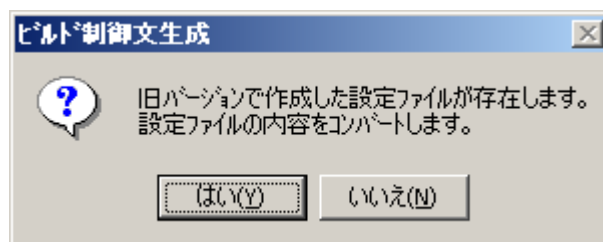
1. [プロジェクト] - [分散開発] メニューの[ビルド制御文生成]を選択します。[ビルド制御文生成]ダイアログが表示されます。

図4-6 ビルド制御文生成ダイアログ

**注意**

ビルド制御文生成時のオプション情報は、プロジェクト資産であるオプションファイルに保存されています。V7.2で作成したプロジェクトに対してビルド制御文生成を実行する場合には、オプションファイルのコンバートを行う必要があります。コンバートが行われていないオプションファイルが存在する場合、〔ビルド制御文生成〕の選択後、〔ビルド制御文生成〕ダイアログが表示される前に以下の確認メッセージが表示されます。

図4-7 コンバート確認メッセージ



〔はい〕をクリックすると、コンバートが行われていないすべてのオプションファイルに対してコンバートが実行され、〔ビルド制御文生成〕ダイアログが表示されます。

〔いいえ〕をクリックすると、オプションファイルのコンバートは行われません。〔ビルド制御文生成〕ダイアログが表示され、前回のビルド制御文生成時のオプション情報が参照できますが、ビルド制御文の生成は行うことはできません。（〔OK〕ボタンをクリックすることはできません）

2. 〔ビルド制御文生成〕ダイアログに情報を設定します。

— 新規にビルド制御文を生成する場合

- 1) 〔新規〕ボタンをクリックします。（初期値）
- 2) 〔ファイル名〕に新規に生成するビルド制御文のファイル名を指定します。はじめてビルド制御文を生成するときは、“プロジェクト名.make”が初期値になっています。ファイル名にはパスを指定することはできません。ビルド制御文はプロジェクトファイルと同じ場所に生成されます。また、ビルド制御文には以下の拡張子を指定することはできません。

.PRJ	.CBI	.LNI	.UCBI	.ULNI
------	------	------	-------	-------

〔OK〕ボタンをクリックして、ビルド制御文を作成すると、〔その他〕フォルダにビルド制御文が追加されます。

— 生成済のビルド制御文を更新する場合

- プロジェクトツリーの〔その他〕フォルダに登録されている既存のビルド制御文を更新します。プロジェクトツリーの〔その他〕フォルダにビルド制御文が登録されていない場合は指定できません。
- 3) 〔更新〕ボタンをクリックします。
  - 4) プロジェクトツリーの〔その他〕フォルダに登録されているビルド制御文が表示されます。コンボボックスから更新を行うビルド制御文を選択します。
3. 〔送信済のファイルだけをビルドの対象にする〕チェックボックスの指定を行います。指定した場合は、“送信”機能で送信した際の送信情報を元に翻訳対象ファイルの絞込みが行われます。〔送信〕ダイアログの〔送信ファイル情報一覧〕に設定されていないファイルは翻訳対象ファイルとなりません。ファイルの存在位置は、“送信”時に指定した“送信先”のディレクトリとなります。指定しない場合は、プロジェクトマネージャに登録されているオブジェクトファイル名を生成するすべてのファイルが翻訳対象ファイルとなります。この場合、未送信のファイルの存在位置は“サーバディレクトリ”として出力されます。
4. 〔生成したmakefileをサーバへ送信する〕チェックボックスの指定を行います。指定した場合は、〔ビルド制御文生成〕ダイアログの〔OK〕ボタンをクリックすると、生成完了後にビルド制御文がサーバディレクトリへ転送されます。指定せずにビルド制御文を生成した場合は、ビルド実行前にUNIX系システムのファイル送信機能を使用してビルド制御文の送信を行ってください。
- ⇒ 4.1.1 [COBOLソース・登録集の送信](#)
5. 〔OK〕ボタンをクリックすると、ビルド制御文が生成されます。
  6. ビルド制御文が生成されると、生成結果がダイアログ表示されます。〔メッセージ〕内には、ビルド制御文の内容確認が必要な項目も出力されます。メッセージの内容を必ず確認し、対処が必要な項目については修正を行ってください。



#### 注意

プロジェクトに、そのプロジェクトでは作成しないリポジトリが、依存リポジトリとして登録してある場合、生成結果ダイアログに以下の項目を出力します。

「次のリポジトリファイルはこのプロジェクトで生成されていないため、必要に応じてメイクファイルに追加してください。」

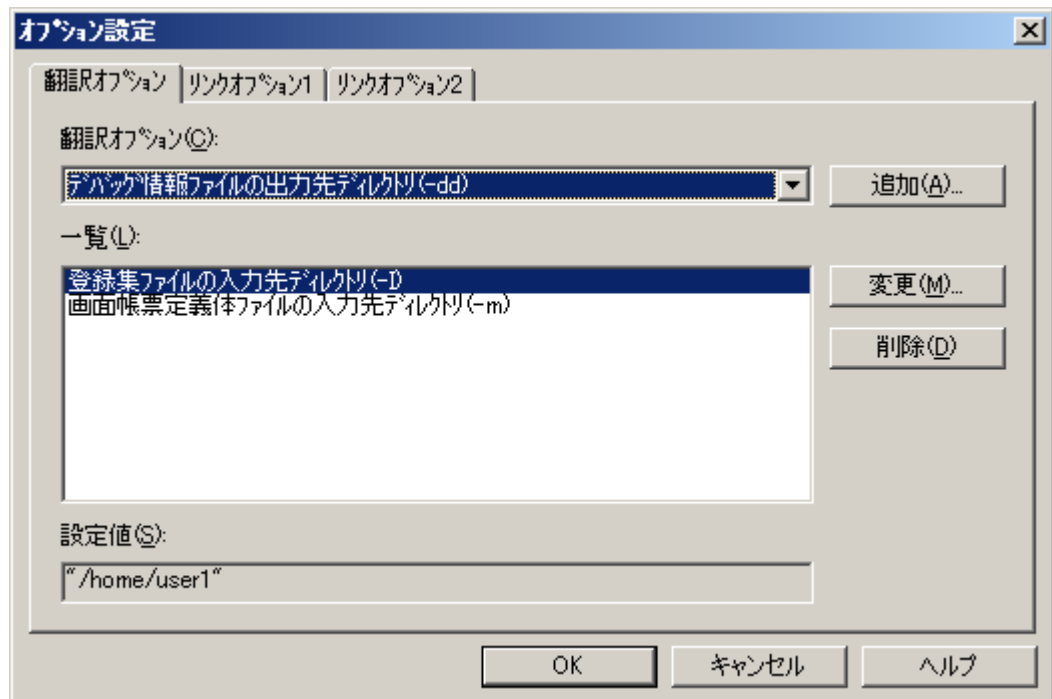
依存リポジトリがNetCOBOLの提供するリポジトリである場合は、この項目は無視してかまいません。

- ビルド制御文へ出力されるオプションの情報のうち、基本的な設定はプロジェクトマネージャに設定された情報から取得されます。

WindowsシステムとUNIX系システムの差異により設定の追加・変更が必要な場合は、〔OK〕ボタンをクリックしてビルド制御文を生成する前に、〔オプション設定〕ボタンをクリックし、〔オプション設定〕プロパティシートを表示させて必要な設定を行ってください。以下に、〔オプション設定〕の手順を示します。

1. 〔オプション設定〕ボタンをクリックし、〔オプション設定〕プロパティシートを表示します。  
〔オプション設定〕プロパティシートには、〔翻訳オプション〕ページ、〔リンクオプション1〕ページ、〔リンクオプション2〕ページがあります。
2. 〔翻訳オプション〕ページの設定を行います。

図4-8 翻訳オプションページ



〔翻訳オプション〕ページでは、ディレクトリの指定を行う翻訳オプションおよび、UNIX系システム固有の翻訳オプションの設定を行います。

設定できる翻訳オプションは以下のオプションとなります。

- デバッグ情報ファイルの出力先ディレクトリ (-dd)
- オブジェクトファイルの出力先ディレクトリ (-do)
- 翻訳リストファイルの出力先ディレクトリ (-dp)
- リポジトリファイルの入出力先ディレクトリの指定 (-dr)
- ソース解析情報ファイルの出力先ディレクトリ (-ds)
- ファイル定義体ファイルの入力先ディレクトリ (-f)
- 登録集ファイルの入力先ディレクトリ (-I)
- 画面帳票定義体ファイルの入力先ディレクトリ (-m)
- リポジトリファイルの入力先ディレクトリ (-R)
- 実行時コード系チェック (CODECHK)
- 文字コードの扱い (KANA)
- 連絡節のデータ宣言の扱い (LALIGN)
- 重複文字の扱い (DUPCHAR)

〔一覧〕には指定済の翻訳オプションが表示されます。

ディレクトリの指定を行う翻訳オプションは、Windowsの翻訳オプションの指定有無により初期表示内容が変更されます。

- 新規にビルド制御文を生成する場合  
Windowsの翻訳オプションの指定がある翻訳オプションは〔一覧〕リストに追加されて表示されます。  
設定値（ディレクトリ）には“サーバディレクトリ”が初期値として設定されています。他のディレクトリを指定する場合には、〔変更〕を行ってください。
- 生成済のビルド制御文を更新する場合  
前回指定した翻訳オプションが表示されます。  
しかし、以下の指定については、Windowsの翻訳オプションと対になっており、Windowsの翻訳オプションが指定されていない場合は有効な指定となりません。  
そのため、前回生成後にWindowsの翻訳オプションの指定に対して削除や無効とな

る設定値に変更した場合は、〔一覧〕から削除されます。

前回の状態に戻す場合には、再度〔翻訳オプション〕コンボボックスから選択し、〔追加〕を行ってください。

**表4-6 Windowsの翻訳オプションと対となる翻訳オプション**

翻訳オプション	Windowsの翻訳オプション
デバッグ情報ファイルの出力先ディレクトリ (-dd)	TEST
オブジェクトファイルの出力先ディレクトリ (-do)	OBJECT * NOOBJECTの場合のみ無効
翻訳リストファイルの出力先ディレクトリ (-dp)	PRINT
ソース解析情報ファイルの出力先ディレクトリ (-ds)	SAI

〔一覧〕に設定されたオプションの内容がビルド制御文に出力されます。

必要なオプションが出力される様に、以下の操作で〔一覧〕リストを編集してください。

— 追加：

〔翻訳オプション〕コンボボックスで選択したオプションに対して情報を入力するための追加画面を表示します。表示するダイアログは、オプションによって異なります。追加したオプションは、〔一覧〕リストビューに登録されます。

— 変更：

〔一覧〕リストボックスで選択したオプションの設定値を変更する場合に指定します。指定したオプションに対応するダイアログが表示されます。

— 削除：

〔一覧〕リストビューで選択した翻訳オプションを削除するときに指定します。

〔一覧〕リストのレコードを選択すると、〔設定値〕（読み取り専用）に選択した翻訳オプションの設定値が表示されます

### 3. 〔リンクオプション1〕ページの設定を行います

ここでは、プロジェクトマネージャに登録しているインポートライブラリファイルに対応する、UNIX系システム上のライブラリファイル名を指定します。

ただし、次に示すインポートライブラリについては、ビルド制御文に、自動的に出力されるため、指定は不要です。

— プロジェクトツリーに登録されたアプリケーションの間で動的リンク構造が成立している場合（アプリケーションが一方のアプリケーションのインポートライブラリをリンクしている場合）。

— WindowsシステムとUNIX系システムで共にNetCOBOL製品が提供しているライブラリファイル。NetCOBOL製品が提供しているライブラリファイルについては、“4.2.1 [ビルド制御文雛型の生成時の規則](#)”の“リンクオプション”を参照してください。

例えば、以下のようなプロジェクトツリーの場合、SAMPLE.LIBを除くインポートライブラリファイルは、前述した条件に当てはまりますので、〔リンクオプション1〕ページで指定する必要はありません。SAMPLE.LIBのみ、対応するUNIX系システム上のライブラリファイル名を指定する必要があるため、〔リンクオプション1〕ページの〔一覧〕リストにファイル名が表示されます（前回ビルド制御文を生成した時に削除した場合は除きます）。

図4-9 インポートライブラリを指定したプロジェクト

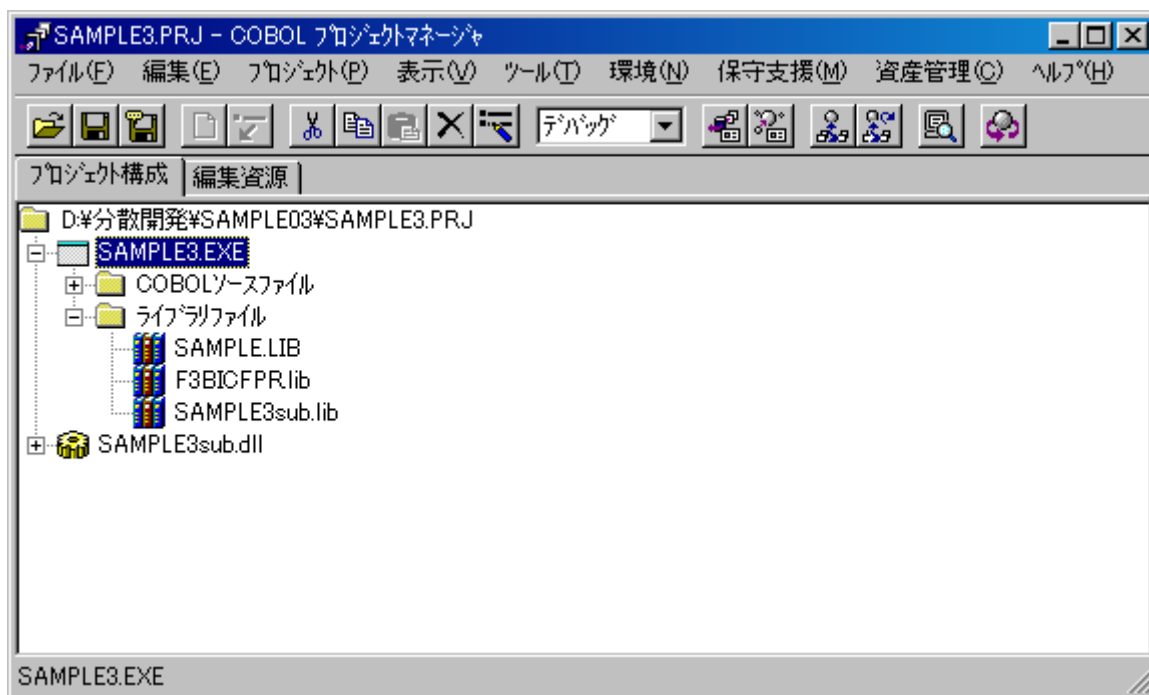
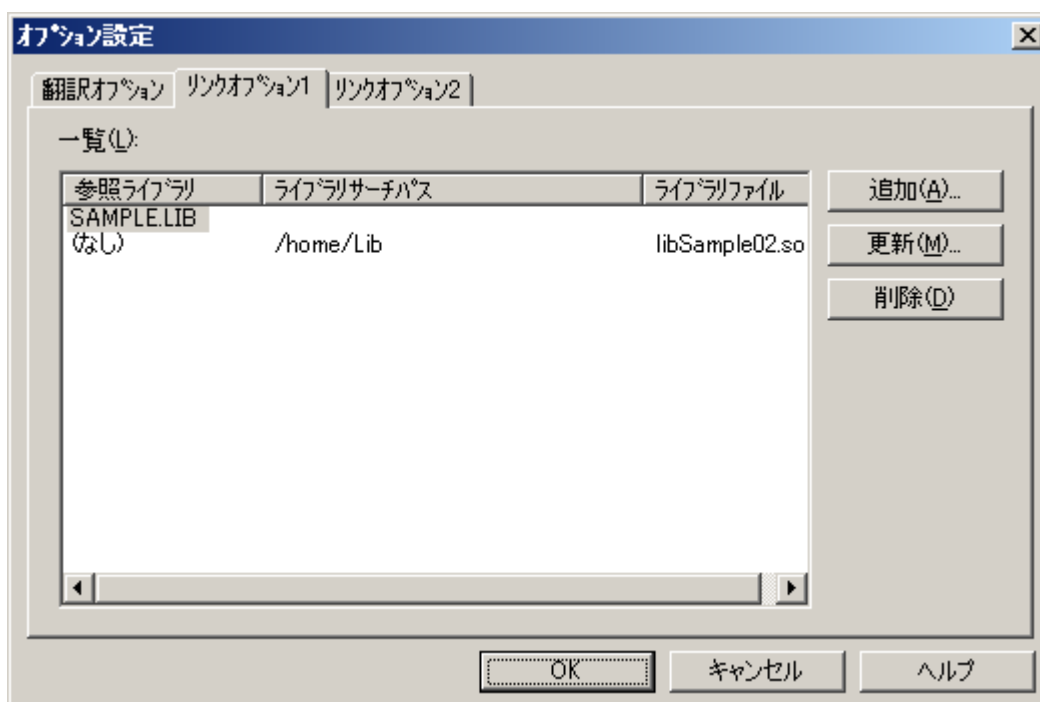


図4-10 リンクオプション1 ページ



〔一覧〕に設定されているライブラリの情報が表示されます。

— 参照ライブラリ

プロジェクトマネージャに登録されている、Windowsシステムのインポートライブラリ名を表示します。

UNIX系システム上でのみ必要なライブラリファイルが存在する場合には、“参照ライブラリ”を“(なし)”で指定してください。

— ライブラリサーチパス



指定したUNIX系システム上のライブラリファイルの存在するパスが表示されます。プロジェクトマネージャに登録されているライブラリファイルに対して指定が行われていない場合は、何も表示されません。

— ライブラリファイル

指定したUNIX系システム上のライブラリファイル名が表示されます。

プロジェクトマネージャに登録されているライブラリファイルに対して指定が行われていない場合は、何も表示されません。

ライブラリファイルが指定されていないレコードの情報は、リンクオプションへの出力は行われません。

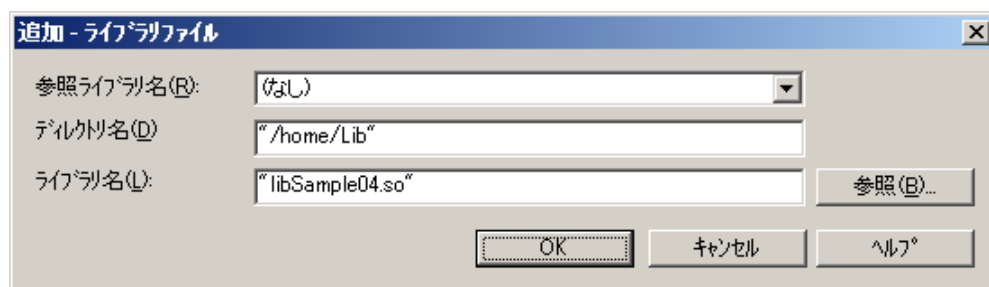
必要なライブラリの指定が出力される様に、以下の操作で「一覧」リストを編集してください。

— 追加：

リンクオプションへ出力するライブラリファイルの追加画面を表示します。

追加したライブラリファイル名は「一覧」リストへ表示されます。

図4-11 追加 - ライブラリファイルダイアログ



— 参照ライブラリ名：

参照ライブラリのファイル名を選択します。

通常は“(なし)”のみが表示されます。

プロジェクトマネージャに登録されているライブラリファイルのレコードを削除した場合のみ、削除したライブラリファイル名がプルダウンに追加されます。

— ディレクトリ名：

リンクオプションに指定するライブラリファイルの存在するディレクトリを指定します。

ディレクトリの指定は絶対パスでも相対パスでも構いません。記述した形式でリンクオプションに指定されます。

「参照」ボタンでライブラリファイルを選択した場合には、選択したファイルのパス名が自動的に設定されます。

キーボードから指定する場合は、ダブルコーテーションで囲んでください。

— ライブラリ名：

リンクオプションに指定するライブラリファイルのファイル名を指定します。

ライブラリ名に指定可能なファイル名は、拡張子に“.so”か“.a”をもつファイル名に限られます。また、ファイル名の先頭は“lib”でなければなりません。

「参照」ボタンでライブラリファイルを選択した場合には、選択したファイルのファイル名が自動的に設定されます。

キーボードから指定する場合は、ダブルコーテーションで囲んでください。

「参照ライブラリ」に“(なし)”を指定した場合は、複数のファイル名を指定することができます。その場合は、ファイル名を空白で区切ります。

ライブラリ名は以下の形式で指定可能です。

a- ファイル名のみ：

記述したファイル名がライブラリファイルとして設定されます。

## b- 相対パス指定：

パス部分がディレクトリ名の後ろ（下位階層）に付加され、ライブラリサーチパスに設定されます。

ライブラリファイルにはファイル名のみが設定されます。

## c- 絶対パス指定：

パス部分がライブラリサーチパスに設定されます。この場合、ディレクトリ名に記述した内容は無効となり反映されません。

ライブラリファイルにはファイル名のみが設定されます。

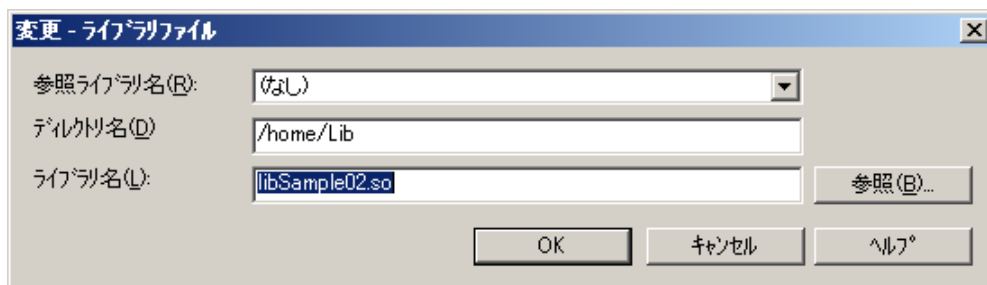
## — 更新：

〔一覧〕リストで選択したライブラリファイルの更新画面を表示します。

（複数のレコードを選択して〔更新〕することはできません）

更新したライブラリファイル名は〔一覧〕リストで選択したレコードへ反映されます。

図4-12 更新 - ライブラリファイルダイアログ



## — 参照ライブラリ名：

〔一覧〕リストで選択した参照ライブラリ名が表示されます。

プロジェクトマネージャに登録されているファイルを更新する場合は、変更はできません。

## — ディレクトリ名：

〔追加 - ライブラリファイル〕ダイアログと同様です。

## — ライブラリ名：

〔追加 - ライブラリファイル〕ダイアログと同様です。

ただし、“更新”の場合に複数ファイルを指定することはできません。

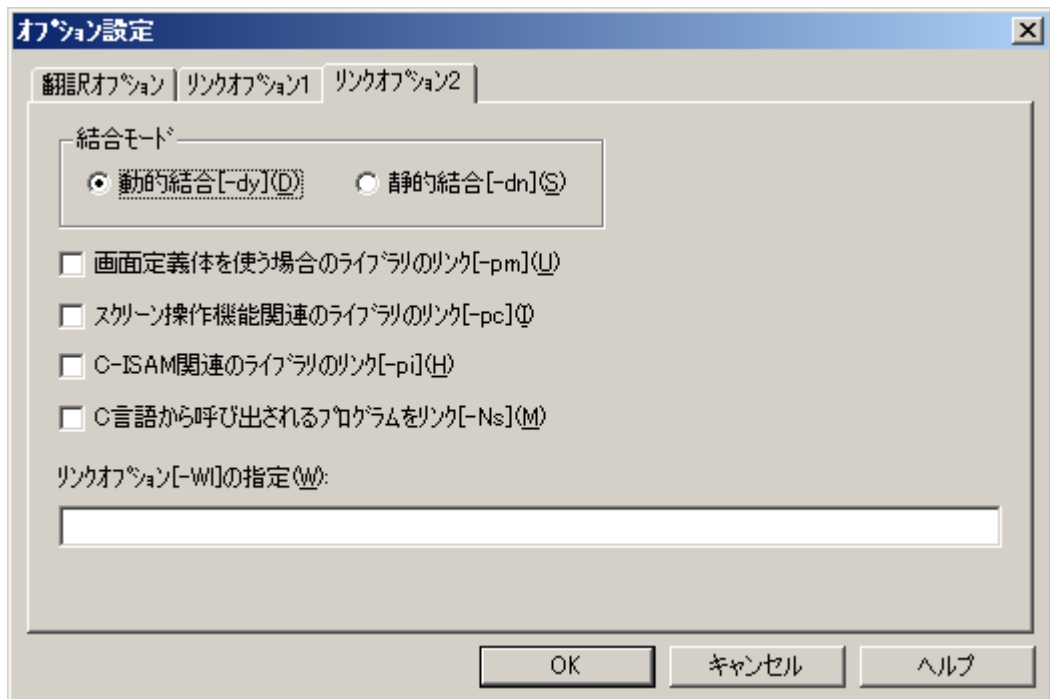
## — 削除：

〔一覧〕リストで選択したライブラリファイルの情報を削除します。

## 4. 〔リンクオプション2〕ページの設定を行います。

ここで選択したオプションが、リンクオプションとして出力されます。

図4-13 リンクオプション2 ページ



- 動的結合 [-dy] :  
アプリケーションをUNIX系システムで動的結合により作成する場合に指定します。  
初期状態は、動的結合となります。
- 静的結合 [-dn] :  
アプリケーションをUNIX系システムで静的結合により作成する場合に指定します。
- 画面定義体を使う場合のライブラリのリンク [-pm] :  
画面定義体を使用しているプログラムをリンクする場合に指定します。
- スクリーン操作機能関連のライブラリのリンク [-pc] :  
スクリーン操作機能を使用しているプログラムをリンクする場合に指定します。
- C-ISAM関連のライブラリのリンク [-pi] :  
C-ISAMを使用しているプログラムをリンクする場合に指定します。
- C言語から呼び出されるプログラムをリンク [-Ns] :  
C言語から呼び出されるプログラムをリンクする場合に指定します。
- リンクオプション [-Wl] の指定 :  
リンクコマンドが使用するリンクオプションを指定します。

5. [OK] ボタンをクリックして [オプション設定] プロパティシートを終了します。

### 4.2.3 生成したビルド制御文雛型とその修正



生成したビルド制御文は、必要に応じて編集します。

編集するには、プロジェクトツリーの“その他”フォルダに登録されているビルド制御文を選択して、エディタを起動します。

生成したビルド制御文雛型と、各部分の説明、および編集方法を示します。

#### ビルド制御文雛型

図4-14 ビルド制御文雛型

```
#####
#                               Makefile NetCOBOL バージョンレベル
#####

#=====
# Macro
#=====

SHELL = /bin/sh                               ...[1]
# NetCOBOL install directory                 ...[2]
UCOBDIR = /opt/FJSVcbl/
include $(UCOBDIR)COBOL/config/mkinc/LIBLIST
# Link option                               ...[3]
COBLDFLAGS = ライブラリ名, リンクオプション
# IDL compile option                         ...[4]
IDLFLAGS1 = IDLc/tdcオプション
# Precompiler and insdbinf command           ...[5]
PRECOMFLAGS1 = プリコンパイラコマンド コマンドオプション
INSDBFLAGS1 = insdbinf insdbinfオプション
# Server Directory                         ...[6]
SERVERDIR = サーバディレクトリ
# Copy library name                         ...[7]
登録集名 = $(SERVERDIR)

# Do not modify Start                       ...[8]

COBFLAGS = -WC, "翻訳オプション", cobolコマンドラインオプション
COBOL = cobol
COMPILE.cob = $(COBOL) -c $(COBFLAGS)
LINKELF.cob = $(COBOL) -o $$ $(COBLDFLAGS)
LINKLIB.cob = $(COBOL) -G -o $$ $(COBLDFLAGS)
RM = rm -f
OBJDIR = オブジェクトファイル格納先
REPDIR = リポジトリファイル入出力先
SVDDIR = デバッグ情報ファイル格納先
WIN_COB_LIBSUFFIX = LIBEXT設定値
WIN_SMED_SUFFIX = FORMEXT設定値
WIN_FFD_SUFFIX = FILEEXT設定値
ENVSETFILE = 環境変数設定用スクリプトファイル
```

```

# Do not modify End

#=====
# Suffix rule
#=====

# Do not modify Start

.SUFFIXES :                                     ...[9]
.SUFFIXES : .cobol .cob .cbl $(SUFFIXES)

.cobol.o:
    . $(ENVSETFILE) ; ¥
    $(COMPILE.cob) $<

.cob.o:
    . $(ENVSETFILE) ; ¥
    $(COMPILE.cob) $<

.cbl.o:
    . $(ENVSETFILE) ; ¥
    $(COMPILE.cob) $<

# Do not modify End

#=====
# Default targets
#=====
all: 相互参照リポジトリ 共用オブジェクトプログラム 実行形式プログラム ...[10]

clean:                                         ...[11]
    $(RM) 共用オブジェクトプログラム
    $(RM) 実行形式プログラム
    $(RM) オブジェクトファイル...
    $(RM) リポジトリファイル...
    $(RM) デバッグ情報ファイル ...
    $(RM) Interstage関連ファイル ...
    $(RM) プリコンパイラ展開ソースファイル ...
    $(RM) insdbinf生成ファイル ...
    $(RM) $(SERVERDIR)core
    $(RM) $(ENVSETFILE)

rebuild: clean all                           ...[12]

$(ENVSETFILE) : Makefile                     ...[13]
    @$(RM) $(ENVSETFILE)
    @echo '#!/bin/sh' >>$(ENVSETFILE)
    @echo 'if [ "$$COB_LIBSUFFIX" = "" ]; then' >>$(ENVSETFILE)
    @echo '    COB_LIBSUFFIX=${WIN_COB_LIBSUFFIX}' >>$(ENVSETFILE)
    @echo 'else' >>$(ENVSETFILE)
    @echo '    COB_LIBSUFFIX=${WIN_COB_LIBSUFFIX}, $$ {COB_LIBSUFFIX}' >>$(ENVSETFILE)

```

```

@echo 'fi' >>$(ENVSETFILE)
@echo 'export COB_LIBSUFFIX' >>$(ENVSETFILE)
@echo 'if [ "$$SMED_SUFFIX" = "" ]; then' >>$(ENVSETFILE)
@echo '    SMED_SUFFIX=${WIN_SMED_SUFFIX}' >>$(ENVSETFILE)
@echo 'else' >>$(ENVSETFILE)
@echo '    SMED_SUFFIX=${WIN_SMED_SUFFIX}, $$ {SMED_SUFFIX}' >>$(ENVSETFILE)
@echo 'fi' >>$(ENVSETFILE)
@echo 'export SMED_SUFFIX' >>$(ENVSETFILE)
@echo 'if [ "$$FFD_SUFFIX" = "" ]; then' >>$(ENVSETFILE)
@echo '    FFD_SUFFIX=${WIN_FFD_SUFFIX}' >>$(ENVSETFILE)
@echo 'else' >>$(ENVSETFILE)
@echo '    FFD_SUFFIX=${WIN_FFD_SUFFIX}, $$ {FFD_SUFFIX}' >>$(ENVSETFILE)
@echo 'fi' >>$(ENVSETFILE)
@echo 'export FFD_SUFFIX' >>$(ENVSETFILE)
@echo '登録集名=${登録集名}' >>$(ENVSETFILE)
@echo 'export 登録集名' >>$(ENVSETFILE)
@chmod +x $(ENVSETFILE)

#=====
# Build program
#=====
オブジェクトファイル: 依存リポジトリ 登録集 定義体 相互参照リポジトリ Makefile ¥
$(ENVSETFILE)                                     ...[14]

オブジェクトファイル: COBOLソースファイル 依存リポジトリ 登録集 定義体 ¥
相互参照リポジトリ Makefile $(ENVSETFILE)         ...[15]
. $(ENVSETFILE) ; ¥
$(COMPILE.cob) COBOLソースファイル

オブジェクトファイル: COBOLソースファイル 依存リポジトリ 登録集 定義体 ¥
相互参照リポジトリ Makefile $(ENVSETFILE)         ...[16]
. $(ENVSETFILE) ; ¥
$(COMPILE.cob) -M COBOLソースファイル

リポジトリファイル: COBOLソースファイル 依存リポジトリ 登録集 定義体 $(ENVSETFILE)
...[17]
. $(ENVSETFILE) ; ¥
$(COMPILE.cob) -WC, "CREATE(REP)" COBOLソースファイル

スタブ/スケルトンファイル: IDLソースファイル インクルードファイル ...[18]
IDLコンパイラ 言語オプション $(IDLFLAGSn) IDLソースファイル

実行形式プログラム: オブジェクトファイル ライブラリ Makefile ...[19]
$(LINKELF.cob) ライブラリ オブジェクトファイル

共用オブジェクトプログラム : オブジェクトファイル ライブラリ Makefile ...[20]
$(LINKLIB.cob) ライブラリ オブジェクトファイル

プリコンパイラ展開ソースファイル : オリジナルソースファイル インクルードファイル
...[21]
$(PRECOMFLAGSn)

```

```
insdbinf生成ファイル：プリコンパイラ展開ソースファイル          ...[22]
$(INSDBFLAGSn) -f オリジナルソースファイル < プリコンパイラ展開ソースファイル > $@
```

### ビルド制御文難型の説明と編集方法

- [1] メイクファイルの動作するシェルの指定です。
- [2] UNIX 上の NetCOBOL インストール先です。
- [3] cobol コマンドのコマンドラインに指定する、リンクに関するオプションです。  
 [制御文生成] ダイアログの [リンクオプション1] タブ、[リンクオプション2] タブ  
 に入力した内容を出力します。
- [4] IDL コンパイラである IDLc コマンド、tdc コマンドに指定するオプションです。  
 [Interstarge] ダイアログを使用してCORBAアプリケーションを生成するための依存関係  
 を設定した場合に出力します。  
 オプションは、[Interstarge] ダイアログの [IDL翻訳オプション] エディットボックス  
 に設定したオプションを元に生成します。  
 プロジェクトで [Interstarge] ダイアログを複数回使用している場合、ツリーの上から  
 順に、IDLFLAGS1、IDLFLAGS2、…に対応して設定します。

オプションを変更する場合は、次に示す下線部分を書き換えてください。

```
IDLFLAGS1 = IDLc/tdcオプション
```

- [5] プリコンパイラコマンドと、オプションの指定です。insdbinf コマンドを使用する場合、  
 insdbinf コマンドと、オプションもここへ出力します。  
 [プリコンパイラ設定] ダイアログで、プリコンパイラの設定を行なった場合、出力しま  
 す。  
 プロジェクトで複数のプリコンパイラやinsdbinfコマンドを使用している場合、ツリーの  
 上の方から順に、プリコンパイラの場合はPRECOMFLAGS1、PRECOMFLAGS2、…が対応します。  
 同様に、insdbinfコマンドの場合、INSDBFLAGS1、INSDBFLAGS2、…が対応します。  
 [プリコンパイラ設定] ダイアログの、[コマンドパラメタ] ドロップダウンリストに指  
 定したパラメタ文字列にマクロを使用している場合、対応するファイル名に変換して出力  
 します。  
 ファイルの種類と、マクロ名の対応関係は、下表のようになります。

表4-7 ファイルの種類とマクロ名の対応

ファイルの種類	マクロ名
入力ファイル	%INFILE%
出力ファイル	%OUTFILE%
出力ファイルフォルダ	%OUTFOLDER%

プリコンパイラのコマンドやオプションは、UNIX上のプリコンパイラの仕様や、環境に合  
 わせて書き換える必要があります。プリコンパイラのコマンドやオプション、insdbinf  
 コマンドのオプションを変更する場合は、次に示す下線部分を書き換えてください。

```
PRECOMFLAGS1 = プリコンパイラコマンド コマンドオプション
INSDBFLAGS1 = insdbinf insdbinfオプション
:
```

- [6] サーバディレクトリです。  
 [プロパティ] ダイアログの [分散開発] タブの [サーバディレクトリ] エディットボッ

クスに指定したディレクトリ名を出力します。

ファイル名の前に\$(SERVERDIR)を付加している場合、サーバディレクトリに格納するファイルを示します。

[7] 登録集名の指定です。

〔登録集名〕ダイアログで、登録集名を設定した場合に出力します。

登録集名に定義する、UNIX上での登録集ファイルを格納するディレクトリ名を変更する場合は、次に示す下線部分を書き換えてください。

```
登録集名 = $(SERVERDIR)
:
```

なお、〔Interstage〕ダイアログを使用してCORBAアプリケーションを生成する場合、指定の内容によっては、登録集名“CORBA”が設定される場合があります。この場合、ビルド制御文にも登録集名“CORBA”を出力します。次に示す下線部分を、IDLコンパイルを行なう環境に合わせて書き換えてください。設定する値については、Interstageのマニュアルを参照してください。

```
CORBA = /opt/FSUNod/include/COBOL
```

[8] アプリケーションのビルドに必要な、その他の設定です。

〔Do not modify Start〕から、〔Do not modify End〕で囲まれる部分は、ほとんどの場合、編集する必要はありません。

この部分の内容は次の通りです。

— COBFLAGS :

cobolコマンドの翻訳に関するオプションです。〔翻訳オプション〕ダイアログに指定した翻訳オプションを、UNIX上で使用できる形式に変換して出力します。また、〔制御文生成〕ダイアログの〔翻訳オプション〕タブで指定したオプションも、COBFLAGSに出力します。

ほとんどの場合、修正する必要はありませんが、次に示す場合は、修正が必要になります。

- 〔その他の翻訳オプション〕がWindowsとUNIXで異なる
- CORBAアプリケーションを作成している
  
- 〔その他の翻訳オプション〕がWindowsとUNIXで異なる  
 〔翻訳オプション〕ダイアログの、〔その他の翻訳オプション〕エディットボックスに指定したオプションがWindowsとUNIXで異なる場合は、次に示す下線部分のうち、指定したオプションのみを書き換えてください。なお、他の翻訳オプションの設定状態によっては、位置がずれる場合があります。

```
COBFLAGS = -WC,"翻訳オプション その他の翻訳オプション",cobolコマンド  
ラインオプション
```

- CORBAアプリケーションを作成している  
 〔Interstage〕ダイアログを使用してCORBAアプリケーションを生成する場合、指定の内容によっては、COBFLAGSに次のようなオプションが追加されます。

```
-I"/opt/FSUNod/include/oocob"  
-R"/opt/FSUNod/rep"  
-R"/opt/FSUNod/rep/unicode"
```



IDLコンパイルを行なう環境に合わせて、下線部分を書き換えてください。  
設定する値については、Interstageのマニュアルを参照してください。

- COBOL :  
cobolコマンドを表わします。
  - COMPILE.cob :  
COBOLソースを翻訳するコマンドの指定です。
  - LINKELF.cob :  
実行形式プログラムを作成するためのコマンドの指定です。
  - LINKLIB.cob :  
共用オブジェクトプログラムを作成するためのコマンドの指定です。
  - RM :  
以前のビルドで生成したファイルを削除するためのコマンドの指定です。
  - OBJDIR :  
オブジェクトファイル格納先ディレクトリです。  
翻訳オプション-doを指定した場合に出力します。
  - REPDIR :  
リポジトリファイル格納先ディレクトリです。  
翻訳オプション-drを指定した場合に出力します。
  - SVDDIR :  
デバッグ情報ファイル格納先ディレクトリです。  
翻訳オプション-ddを指定した場合に出力します。
  - WIN\_COB\_LIBSUFFIX :  
環境変数COB\_LIBSUFFIXに設定する値です。  
〔翻訳オプション〕ダイアログで、翻訳オプションLIBEXTを指定した場合に出力します。
  - WIN\_SMED\_SUFFIX :  
環境変数SMED\_SUFFIXに設定する値です。  
〔翻訳オプション〕ダイアログで、翻訳オプションFORMEXTを指定した場合に出力します。
  - WIN\_FFD\_SUFFIX :  
環境変数FFD\_SUFFIXに設定する値です。  
〔翻訳オプション〕ダイアログで、翻訳オプションFILEEXTを指定した場合に出力します。
  - ENVSETFILE :  
環境変数を設定するスクリプトファイル名です。
- [9] COBOL ソースの翻訳に関するサフィックスルールです。
- [10] 最終的に生成するファイルです。  
ただし、最終ターゲットを生成するのに必要なCOBOLソースファイルなどを送信していない場合は、最終ターゲットファイルは生成されません。
- [11] 以前のビルドで生成したファイルを削除するときに、対象となるファイルです。
- [12] リビルドを行なうための指定です。
- [13] 環境変数を設定するスクリプトファイルを生成するための指定です。  
〔翻訳オプション〕ダイアログで、翻訳オプションLIBEXT、FORMEXT、FILEEXTを指定した場合、〔登録集名〕ダイアログで、登録集名を設定した場合に出力します。
- [14] 副プログラムをコンパイルし、オブジェクトファイルを生成する指定です。  
[9]のサフィックスルールを適用できる場合は、この形式で出力します。
- [15] [14]と同じく、副プログラムをコンパイルし、オブジェクトファイルを生成する指定です。  
サフィックスルールを適用しない場合は、この形式で出力します。
- [16] 主プログラムをコンパイルし、オブジェクトファイルを生成する指定です。
- [17] ターゲットリポジトリファイルを生成する指定です。

プロジェクトのツリーに、ターゲットリポジトリファイルを登録している場合に出力します。

- [18] IDL ソースファイルから、スタブ/スケルトンファイルを生成する指定です。  
 [Interstage] ダイアログを使用してCORBAアプリケーションを生成するための依存関係を設定した場合に出力します。
- [19] 実行形式プログラムを生成する指定です。  
 実行形式プログラムは、サーバディレクトリに生成されます。
- [20] 共用オブジェクトプログラムを生成する指定です。  
 共用オブジェクトプログラムは、サーバディレクトリに生成されます。
- [21] プリコンパイラを使用して、プリコンパイラ展開ソースファイルを生成する指定です。  
 [プリコンパイラ設定] ダイアログで、プリコンパイラを登録した場合に出力します。
- [22] insdbinf 生成ファイルを生成する指定です。  
 [プリコンパイラ設定] ダイアログの、[INSDBINF] タブで、[INSDBINFコマンドを使用する] チェックボックスにチェックをした場合に出力します。

## 各アプリケーションでの注意点

以下のアプリケーションでの注意事項について説明します。

- プリコンパイラを使用するプロジェクト
- CORBAアプリケーション
- Webアプリケーションウィザードで作成したアプリケーション
- プロジェクト外で提供するリポジトリファイル
  
- プリコンパイラを使用するプロジェクト  
 [プリコンパイラ設定] ダイアログを使用して、プロジェクトにプリコンパイラを登録している場合、ビルド制御文に次のように出力します([ ]の数字は「ビルド制御文雛型」の番号に対応します)。
  - プリコンパイラコマンドとオプション、insdbinfコマンドのオプション[5]
  - プリコンパイラ展開ソースファイル、insdbinf生成ファイルを削除するコマンド[11]
  - プリコンパイラ展開ソースファイルを生成する指定[21]
  - insdbinf生成ファイルを生成する指定[22]

プリコンパイラ展開ソースファイルから、オブジェクトファイルやリポジトリファイルを生成する場合、“プリコンパイラ展開ソースファイル”は、「ビルド制御文雛型の説明と編集方法」の“COBOLソースファイル”に相当します。

プリコンパイラのコマンドや、オプションがWindowsと異なる場合、これらの設定を修正する必要があります。修正の方法は、「ビルド制御文雛型の説明」の該当する部分を参照してください。

- CORBAアプリケーション  
 [Interstage] ダイアログを使用して、CORBAアプリケーションを生成するための情報を登録しているプロジェクトの、以下について説明します。
  - IDLソースファイルを登録しているCORBAアプリケーション
  - IDLソースファイルを登録していないCORBAアプリケーション
  - Interstageが提供するライブラリファイルの指定

### IDLソースファイルを登録しているCORBAアプリケーション

IDLソースファイルを使用してCORBAアプリケーションを作成する場合は、ビルド制御文に次のように出力します。

- IDLcコマンド、tdcコマンドに指定するオプション[4]
- 登録集名CORBA[7]
- スタブ/スケルトンファイル、IDL登録集ファイルを削除するコマンド[11]
- スタブ/スケルトンファイルを生成する指定[18]

スタブ/スケルトンファイルから、オブジェクトファイルやリポジトリファイルを生成する場合、“スタブ/スケルトンファイル”は、「ビルド制御文雛型の説明と編集方法」の“COBOLソースファイル”に相当します。

IDLcコマンド、tdcコマンドに指定するオプション[4]は、[Interstage] ダイアログの[IDL翻訳オプション] エディットボックスに設定したオプションを元に生成しています。そのため、お使いの環境に合わせて修正が必要です。特にオプション-Iや-Tが指定してある場合は確認が必要です。

修正の方法は、「ビルド制御文雛型の説明」の該当する部分を参照してください。

#### IDLソースファイルを登録していないCORBAアプリケーション

プロジェクト外から提供されたスタブ/スケルトンファイルを使用する場合、これらのスタブ/スケルトンファイルは、UNIX上のサーバディレクトリにあるものと仮定してビルド制御文に出力します。

#### Interstageが提供するライブラリファイルの指定

CORBAアプリケーションをビルドするときに必要になるライブラリファイルは、[制御文生成] ダイアログの[リンクオプション1] タブで指定します。

指定が必要なライブラリファイルについては、Interstageのマニュアルを参照してください。

#### ● Webアプリケーションウィザードで作成したアプリケーション

プロジェクトマネージャのWebアプリケーションウィザードで生成したHTMLファイルには、実行可能ファイル名や、ダイナミックリンクライブラリ名が直接記述してある場合があります。

これらのHTMLファイルをUNIX上に送信して利用する場合は、ファイル名を、UNIX上の実行形式プログラム名や共用オブジェクトプログラム名に変更する必要があります。

Webアプリケーションウィザードで生成したファイルについては、“COBOL Webサブルーチン使用手引書”を参照してください。

#### ● プロジェクト外で提供するリポジトリファイル

プロジェクト外で作成したリポジトリファイルは、ビルド制御文には出力しません。[制御文生成] ダイアログの[翻訳オプション] タブで、入力となるリポジトリファイルの格納ディレクトリを指定する必要があります。

## 4.3 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、UNIX系システムに登録したプログラム資産をUNIX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。

### 4.3.1 サーバ環境へのビルド制御文の転送



ビルド制御文生成機能によって生成したビルド制御文は、[ビルド制御文生成] ダイアログで[生成したmakefileをサーバへ送信する] チェックボックスを指定した場合にはすでにUNIX系システムに送信されているため、再度送信する必要はありません。

[生成したmakefileをサーバへ送信する] チェックボックスを指定しなかった場合や、生成後にプロジェクトマネージャの[その他] フォルダに登録されたビルド制御文を編集した場合には、NetCOBOLのファイル送信の機能を使用して、UNIX系システムに送信します。

手順はCOBOLソース・登録集原文を送信する場合と基本的に同じです。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“4.1.1 [COBOLソース・登録集の送信](#)”を参照してください。

- ターゲットビルドに使用できるビルド制御文は、“サーバディレクトリ”にあるファイルのみとなります。そのため、[送信先] には以下のどちらかを指定してください。
  - － 指定なし (“サーバディレクトリ” に送信されます)
  - － “サーバディレクトリ” のパスを指定

### 4.3.2 ターゲットビルドの実行



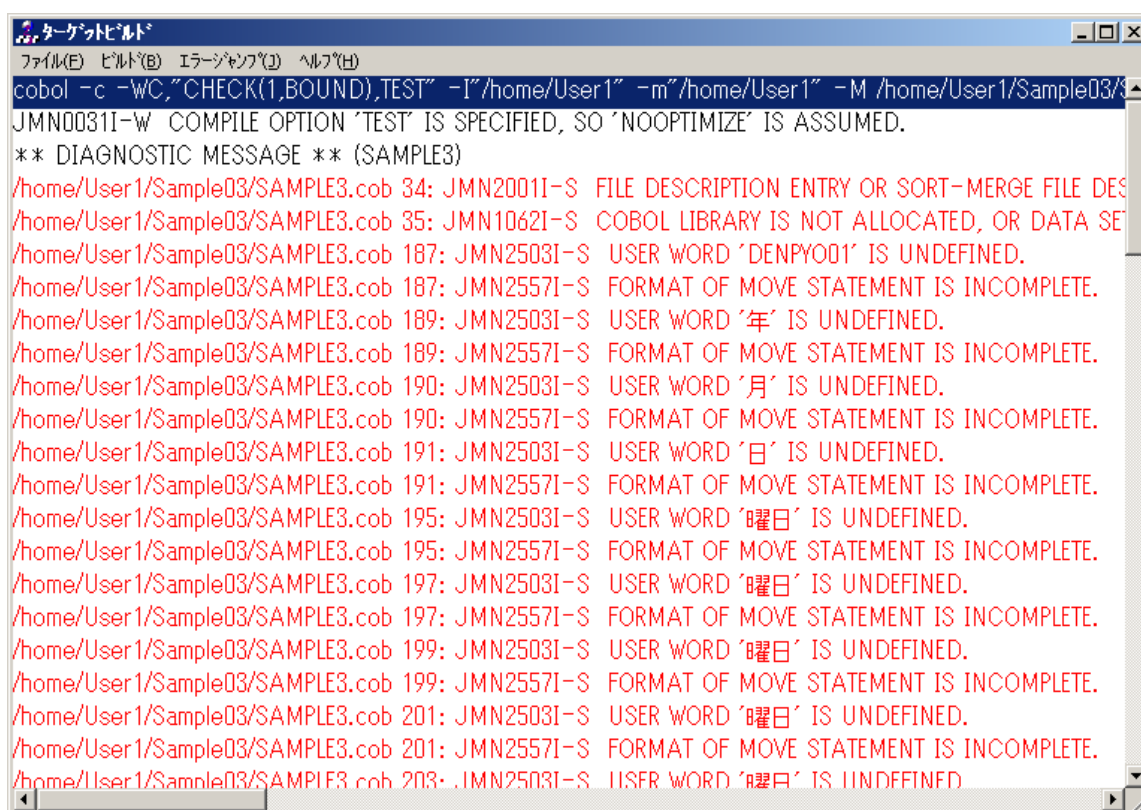
送信したビルド制御文を使用して、プロジェクトマネージャからUNIX系システムでの翻訳・リンクを実行します。

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。[ターゲットビルド] ダイアログが表示されます。

図4-15 ターゲットビルドダイアログ

2. [メイクファイル名] に送信したビルド制御文を指定します。  
プロジェクトマネージャの[その他] フォルダに登録されているビルド制御文は、プルダウンから選択することができます。
3. [ビルド] ボタンまたは[リビルド] ボタンをクリックすると、UNIX系システム上でビルドが実行されます。
4. ビルドが完了すると、[ターゲットビルド] ウィンドウにビルド結果が表示されます。コンパイルエラーは赤字で表示されます。

図4-16 ターゲットビルドウィンドウ



### 4.3.3 ターゲットビルド後のプログラム資産の再修正



#### 4.3.3.1 翻訳エラー時の再修正

1. コンパイルエラーになったCOBOLソースをWindowsシステム上で修正します。  
 [ターゲットビルド] ウィンドウに表示された情報からエラー行を選択することにより、自動的にWindowsシステム上のCOBOLソースを編集するためのエディタが起動され、エラー行にカーソルが位置付けられます。  
 エラージャンプにはメニューによる操作と、キーボードまたはマウスによる操作があります。
  - 1) メニューによる操作  
 [エラージャンプ] メニューのメニュー項目を選択することにより、エラージャンプすることができます。各メニュー項目の動作について説明します。
    - [先頭へ]:  
ビルド結果の最初のコンパイルエラー行へエラージャンプします。
    - [前へ]:  
現在選択されている行の1つ上側のコンパイルエラー行へエラージャンプします。
    - [次へ]:  
現在選択されている行の1つ下側のコンパイルエラー行へエラージャンプします。
    - [末尾へ]:  
ビルド結果の最後のコンパイルエラー行へエラージャンプします。
  - 2) キーボードまたはマウスによる操作  
 コンパイルエラー行を選択し、マウスの左ボタンをダブルクリックするか、リター

- ンキーを押すことによりエラージャンプします。
2. 修正したCOBOLソースをUNIX系システムに送信します。  
送信の手順は“4. 1. 1 [COBOLソース・登録集の送信](#)”を参照してください。
  3. ビルド・リビルドを実行します。  
〔ターゲットビルド〕ウィンドウの〔ビルド〕メニューから、〔ビルド〕または〔リビルド〕を選択してください。  
ターゲットビルド実行時のビルド制御文に対して再度ビルドが実行されます。

#### 4. 3. 3. 2 ビルド制御文の再修正

---

コンパイルエラーの修正のため、ビルド制御文を更新する必要がある場合は、“ビルド制御文生成”を再度実行し、ビルド制御文を再作成してください。

1. ビルド制御文の“更新”を行います。  
ターゲットビルド実行時に指定したビルド制御文を選択し、更新を実行してください。  
手順については“4. 2. 2 [ビルド制御文雛型の生成手順](#)”を参照してください。
2. 再作成したビルド制御文をUNIX系システムに送信します。  
送信の手順はCOBOLソースと同様です。
3. ビルド・リビルドを実行します。  
〔ターゲットビルド〕ウィンドウの〔ビルド〕メニューから、〔ビルド〕または〔リビルド〕を選択してください。  
ビルド制御文に対して再度ビルドが実行されます。

## 4.4 リモートデバッグ

リモートデバッグを使って、作成したプログラムをデバッグする方法について説明します。

### 4.4.1 リモートデバッグの概要



リモートデバッグは、ネットワーク上の別のUNIX系システムで動作しているプログラムをデバッグすることができます。

リモートデバッグでデバッグが行えるプログラムは、通常のデバッグと同様に被デバッグプログラムです。

リモートデバッグ機能を使用するためには、サーバ、クライアントの両方でTCP/IPプロトコルがサポートされている必要があります。

#### 4.4.1.1 リモートデバッグによるデバッグ機能

UNIX系システムで動作しているプログラムに対してリモートデバッグを開始するには、以下の2つの方法があります。

- 一般形式のリモートデバッグ起動方法  
デバッグの起動画面からデバッグ対象プログラムを含む、または呼び出す実行可能プログラムを指定し実行します。
- アタッチ形式のリモートデバッグ起動方法  
システムのサービスとして起動されるようなプログラムから、デバッグ対象となるCOBOLプログラムが呼び出されるような場合に使用します。  
次のようなCOBOLプログラムが該当します。
  - Webサーバ配下のCOBOLプログラム
  - Interstage配下のCOBOLプログラム

それぞれの方法の概要については、“2.1.3.1 [リモートデバッグの2つの方式](#)”を参照してください。

#### 4.4.1.2 リモートデバッグの手順

1. デバッグに必要な資産が以下の格納場所にあることを確認します。  
デバッグに必要な資産がサーバ側とクライアント側のどちらか一方または、両方に適切に格納されている必要があります。  
デバッグに必要な資産を以下に示します。

表4-8 リモートデバッグ時の資産格納場所

デバッグ資産	Windowsクライアント側	UNIXサーバ側
実行可能プログラム	-	○
共用オブジェクトファイル	-	○
デバッグ情報ファイル	-	○
ソースファイル	○ *1	-
登録集原文	○ *1	-
画面帳票定義体	○ *1	○
ファイル定義体	○ *1	○
コマンドファイル	○ *1	-
操作履歴ファイル	○ *1	-

\*1 Windowsクライアント側の資産のコード系は、UNIXサーバ側のロケールに関係なく、常

- に「シフトJIS」です。
- 以下の設定を行います。  
一般形式とアタッチ形式では環境設定および手順が異なりますので、個別に手順を示します。
  - その他のリモートデバッガの使用方法是通常のデバッガと同じです。使用方法については、“3. 6. 3 [対話型デバッガによるデバッグ](#)”を参照してください。

### 一般形式のリモートデバッガ起動方法

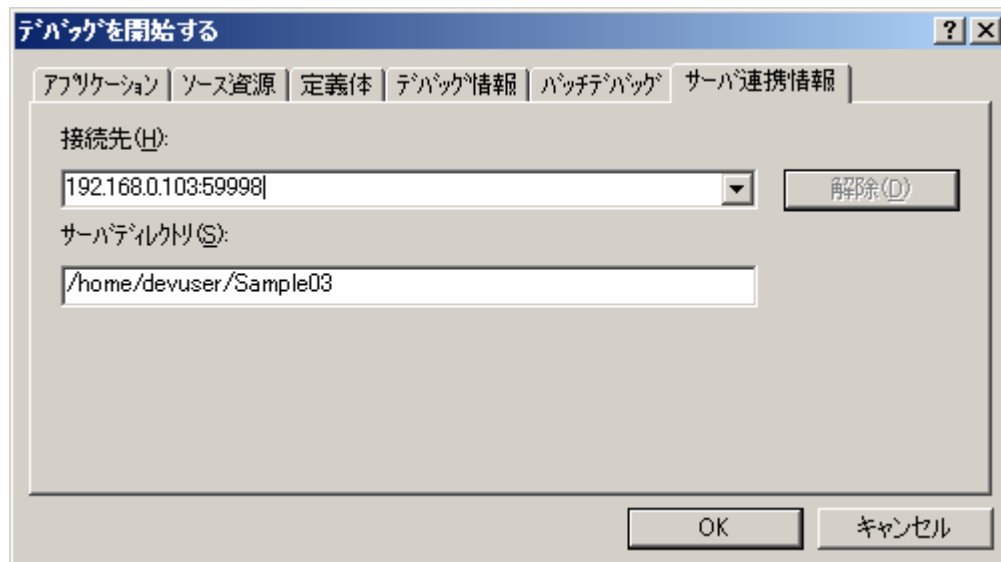
- UNIXサーバ側でリモートデバッガコネクタを起動します。  
UNIXサーバのリモートデバッガコネクタの起動および終了方法については、“2. 1. 3. 2 [リモートデバッグのためのサーバ側の環境設定](#)”および“2. 3. 1. 3 [リモートデバッグ時のサーバ側のユーザ環境の設定](#)”を参照してください。
- リモートデバッガを起動します。  
プロジェクトマネージャの〔プロジェクト〕メニューから〔リモートデバッグ〕を選択します。COBOLデバッガが起動し、〔デバッグを開始する〕ダイアログが表示されます。
- 〔デバッグを開始する〕ダイアログで通常のデバッガと異なる設定は以下の通りです。  
〔サーバ連携情報〕ページが追加されています。〔接続先〕に、リモートデバッガコネクタの起動時に指定したポート番号を追加します。  
指定形式は以下の通りです。

{IPアドレス|ホスト名} [:ポート番号]

指定例

IPアドレスによる指定	192.168.0.103:59998
ホスト名による指定	host1:59998

図4-17 〔デバッグを開始する〕ダイアログ



- 〔サーバ連携情報〕ページ以外のページについては、プロジェクトマネージャの設定よりデバッグに必要な基本的な設定が読み込まれ、設定されて表示されます。  
追加、更新が必要な項目がある場合は、“3. 6. 3 [対話型デバッガによるデバッグ](#)”を参照し設定を行ってください。

### 参考

リモートデバッガは以下の方法で起動することもできます。

- プロジェクトマネージャの〔ツール〕 - 〔デバッガ種別〕で、起動するデバッガ



を選択します。

- 2) プロジェクトマネージャの「ツール」 - 「デバッガ」を選択します。

この場合、プロジェクトマネージャの設定は読み込まれず、初期情報が表示されません。必要な情報をすべて記述する必要があります。

手順については、“NetCOBOL 使用手引書”を参照してください。

### アタッチ形式のリモートデバッグ起動方法

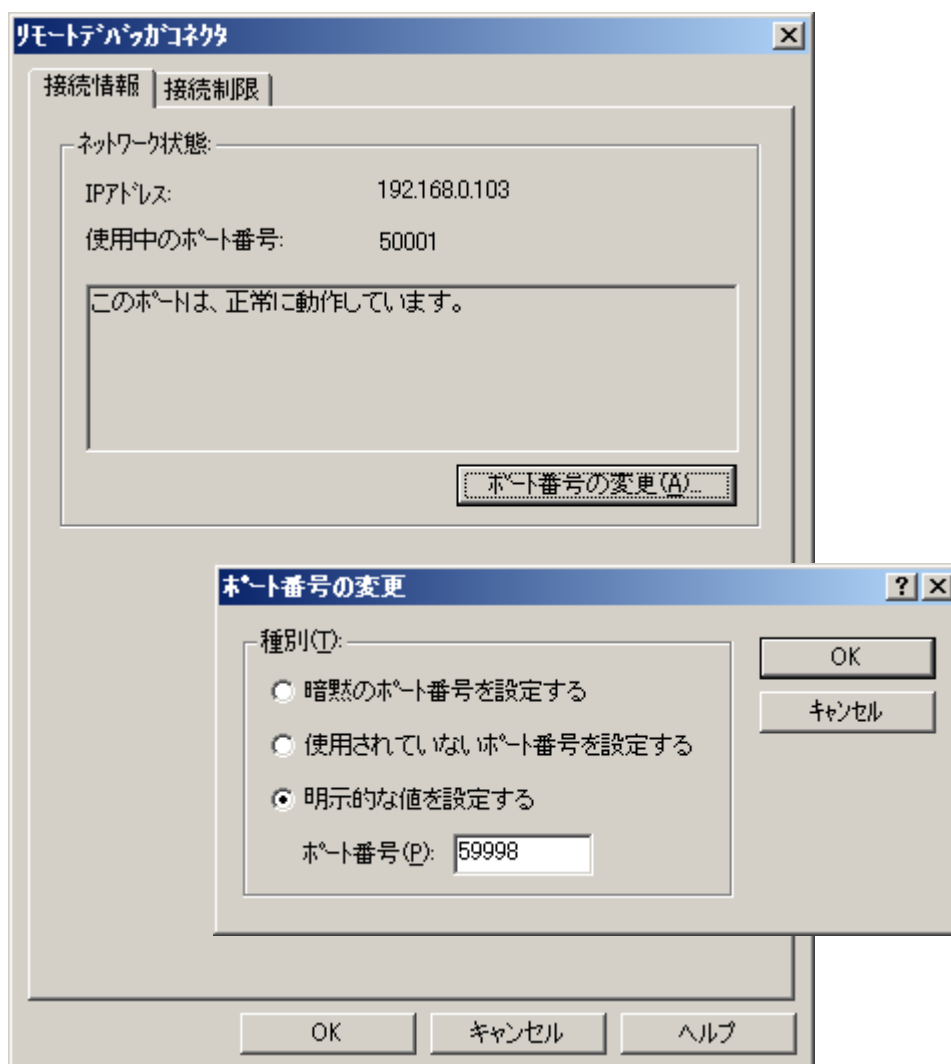
1. Windowsクライアント側でリモートデバッガコネクタを起動します。  
プロジェクトマネージャでデバッグを行うプロジェクトを開き、「ツール」 - 「リモートデバッガコネクタ」を選択します。  
リモートデバッガコネクタが起動されると、タスクトレイに以下のアイコンが表示されて、前回起動したときの設定でUNIX系システム側からのデバッグ開始の指示を監視します。

図4-18 リモートデバッガコネクタアイコン



監視するポート番号を変更する場合は、タスクトレイに常駐しているリモートデバッガコネクタのアイコンを右クリックして、表示されるメニューから“環境設定”を選択してください。“リモートデバッガコネクタ”ダイアログが表示されます。[ポート番号の変更]ボタンをクリックし、[ポート番号の変更]ダイアログを表示して指定してください。

図4-19 リモートデバッガコネクタダイアログ



その他の“リモートデバッガコネクタ”ダイアログの操作方法については、ヘルプや“使用手引書”を参照してください。

デバッグ完了後には、リモートデバッガコネクタを終了する必要があります。

タスクトレイに常駐しているリモートデバッガコネクタのアイコンを右クリックして、表示されるメニューから“リモートデバッガコネクタの終了”を選択します。

2. 環境変数CBR\_ATTACH\_TOOLを設定します。

設定方法については、“2.1.3.2 [リモートデバッグのためのサーバ側の環境設定](#)”を参照してください。

環境変数CBR\_ATTACH\_TOOLに指定するポート番号は、リモートデバッガコネクタで指定したポート番号を使用してください。

環境変数CBR\_ATTACH\_TOOLは初期化ファイルに指定することもできます。

3. デバッグしたいプログラムを実行します。

以下にWebブラウザから実行を行う例を示します。

以下の様なhtmlファイルから、“cgismp01”を呼び出すアプリケーションをデバッグします。

被デバッグプログラムを“cgismp01.cob”とします。

```
<HTML>

  <HEAD>
  <TITLE>ホームページ</TITLE>
</HEAD>
<BODY>

  <FORM METHOD="POST" ACTION="cgismp01">

  <BR>
  名前 : <INPUT SIZE=20 MAXLENGTH=20 TYPE="TEXT" NAME="NAME"><BR>
  <BR>

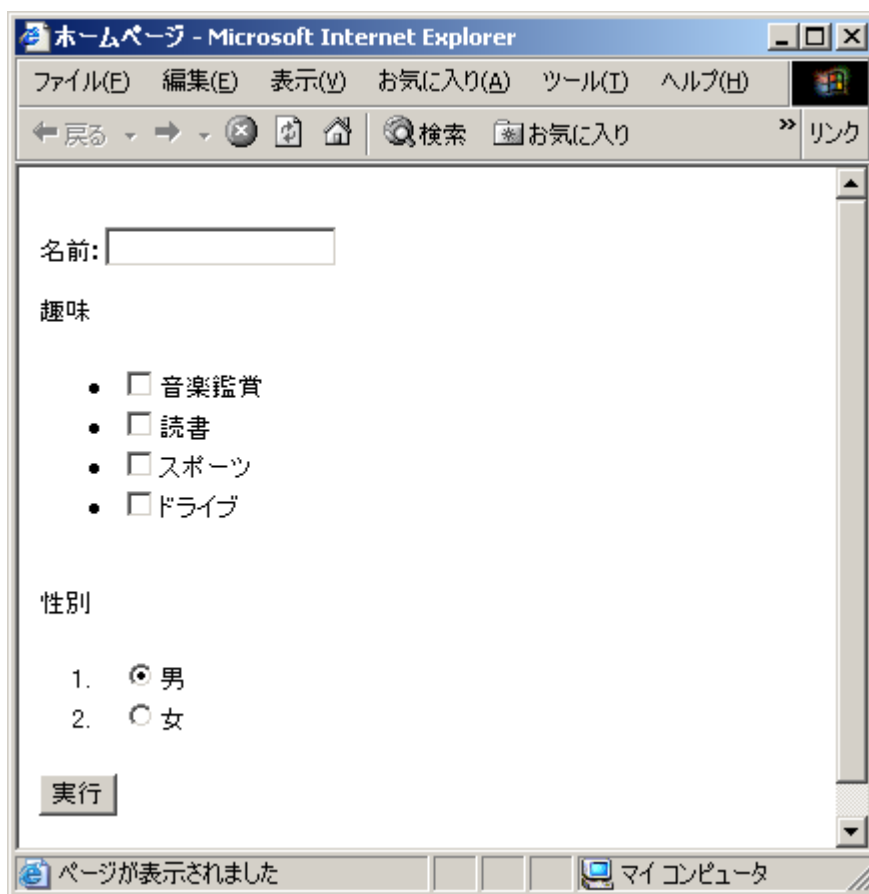
  趣味
  <UL>
  <LI><INPUT TYPE="CHECKBOX" NAME="CHECK1" VALUE="音楽鑑賞">音楽鑑賞<BR>
  <LI><INPUT TYPE="CHECKBOX" NAME="CHECK1" VALUE="読書">読書<BR>
  <LI><INPUT TYPE="CHECKBOX" NAME="CHECK1" VALUE="スポーツ">スポーツ<BR>
  <LI><INPUT TYPE="CHECKBOX" NAME="CHECK1" VALUE="ドライブ">ドライブ<BR>
  </UL>
  <BR>

  性別<BR>
  <OL>
  <LI><INPUT TYPE="RADIO" NAME="RADIO1" VALUE="男" CHECKED>男<BR>
  <LI><INPUT TYPE="RADIO" NAME="RADIO1" VALUE="女">女<BR>
  </OL>

  <INPUT TYPE="SUBMIT" NAME="OK" VALUE="実行"><P>
  </FORM>
  </BODY>
</HTML>
```

- 1) htmlを表示します。

図4-20 html表示例



- 2) “実行” ボタンをクリックします。  
 3) “cgismp01” が呼び出され、Windowsクライアントでリモートデバッグが自動的に起動します。

〔デバッグを開始する〕ダイアログが表示されるので、必要な情報を入力して〔OK〕ボタンをクリックします。

なお、環境変数CBR\_ATTACH\_TOOLに起動パラメタを指定しておく、指定した内容が〔デバッグを開始する〕ダイアログに表示された状態で起動するため、毎回指定する必要がなくなります。

#### 4.4.1.3 デバッグ操作の自動化

デバッガでは、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ための、デバッグ作業を自動化することができます。

コマンドを格納したコマンドファイルを指定することにより、デバッガはコマンドファイル内に記述されたデバッグ操作を、順次自動的に実行します。

デバッグの形態によりバッチデバッグおよび自動デバッグに分類されます。

##### ● バッチデバッグ

すべてのデバッグ操作を自動で行う場合に使用します。

実行するデバッグ操作をすべてコマンドファイルに格納し、デバッグの開始時にコマンドファイルを指定することで、同一のデバッグ操作を再現することができます。

利用者がデバッグ中に操作を指示する必要はありません。

コマンドファイル内のデバッグ操作が完了するとデバッガは終了します。

##### ● 自動デバッグ

頻繁に使用するコマンド列を実行したり、デバッグ手順を途中まで再現するために以前に

採取しておいた操作履歴ファイルのコマンド列を実行する場合に使用します。  
 デバッグ中にコマンドファイルを読み込み、任意の実行個所からコマンドファイル内のデバッグ操作を自動で順次実行することができます。  
 コマンドファイル内のデバッグ操作が完了すると、完了した時点の実行個所に位置付きま  
 すので、決まった手順の実行完了後からデバッグを再開することができます。

以下に、コマンドファイルの作成方法と、バッチデバッグおよび自動デバッグの開始手順を示します。

### コマンドファイルの作成方法

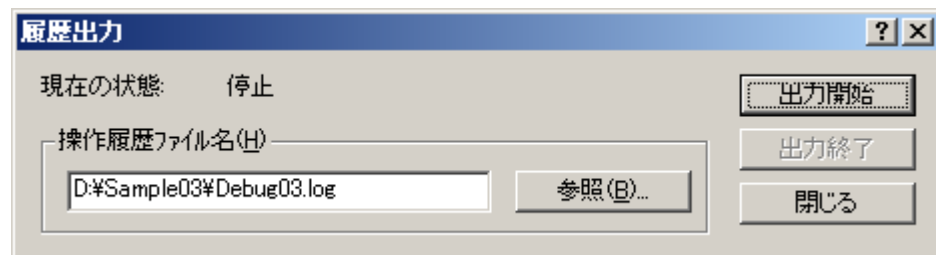
- 操作履歴ファイルを利用する方法

デバッガが出力した操作履歴ファイルをコマンドファイルとして利用することができます。内容を変更しないのでそのまま使用すれば、デバッグ操作を再現できます。また、テキストエディタにより必要箇所を修正することもできます。

操作履歴ファイルは以下の手順で出力します。

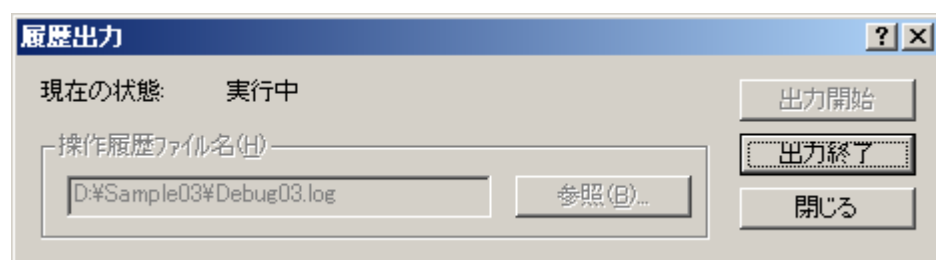
- 1) デバッガの〔オプション〕 - 〔履歴出力〕を選択します。  
 〔履歴出力〕ダイアログが表示されます。

図4-21 履歴出力ダイアログ



- 2) 〔操作履歴ファイル名〕にファイル名を設定します。  
 ファイルはWindowsクライアントのファイル名を指定します。
- 3) 〔出力開始〕ボタンをクリックすると、操作履歴ファイルが設定されます。  
 〔現在の状態〕に“実行中”と表示されることを確認してください。

図4-22 履歴出力ダイアログ（実行中）



- 4) 〔閉じる〕ボタンをクリックし〔履歴出力〕ダイアログを終了します。
- 5) 履歴を取得するデバッグ動作を実行してください。
- 6) 履歴を取得するデバッグ動作が完了した場合は、再度〔履歴出力〕ダイアログを表示し、〔出力終了〕ボタンをクリックしてください。  
 〔現在の状態〕に“停止”が表示され、操作履歴ファイルへの出力が解除されます。

### 参考

操作履歴ファイルは、デバッガの起動パラメタまたはENVコマンドで指定することもできます。手順については、“使用手引書”を参照してください。

- テキストエディタで作成する方法

利用者がテキストエディタを使って作成します。  
 コマンドファイルへ格納できるコマンドは以下の通りです。

表4-9 デバッガのコマンド一覧

分類	コマンド	機能
実行	CONTINUE	実行の再開
	RUNTO	中断点指定実行
	SKIP	実行開始位置の変更
	RERUN	再デバッグ
中断点	BREAK	中断点の設定
	DELETE	中断点の解除
通過カウント点	COUNT	通過カウント点の設定
	DELCOUNT	通過カウント点の解除
データ	LIST	データ域の内容参照
	SET	データ域の内容変更
	LINKAGE	連絡節のデータ域の獲得
	ASSIGNDATA	データファイルの割当て
	OPENDATA	データファイルのオープン
	READDATA	データファイルからの読み込み
	WRITEDATA	データファイルへの書き込み
	CLOSEDATA	データファイルのクローズ
データ域の監視	MONITOR	データ域の変更時中断
	DELMON	データ域変更時中断の解除
	DTRACE	データ域の変更の監視
	DELDTR	データ域変更監視の解除
条件式の監視	DATACHK	条件式の成立の監視
	DELDCHK	条件式監視の解除
状態	ENV	デバッグ環境の設定／変更
	STATUS	デバッグ状態の表示
	SCOPE	プログラム修飾の変更
	WHERE	現在位置の表示
	CALLS	呼出し経路の表示
	THREADLIST	スレッドの状態の表示
スレッド操作	CURRENTTHREAD	暗黙スレッドの切り替え
	ALTERTHREAD	スレッドの状態の変更
操作の再現	AUTORUN	自動デバッグ
終了	QUIT	デバッグの終了

#### 備考

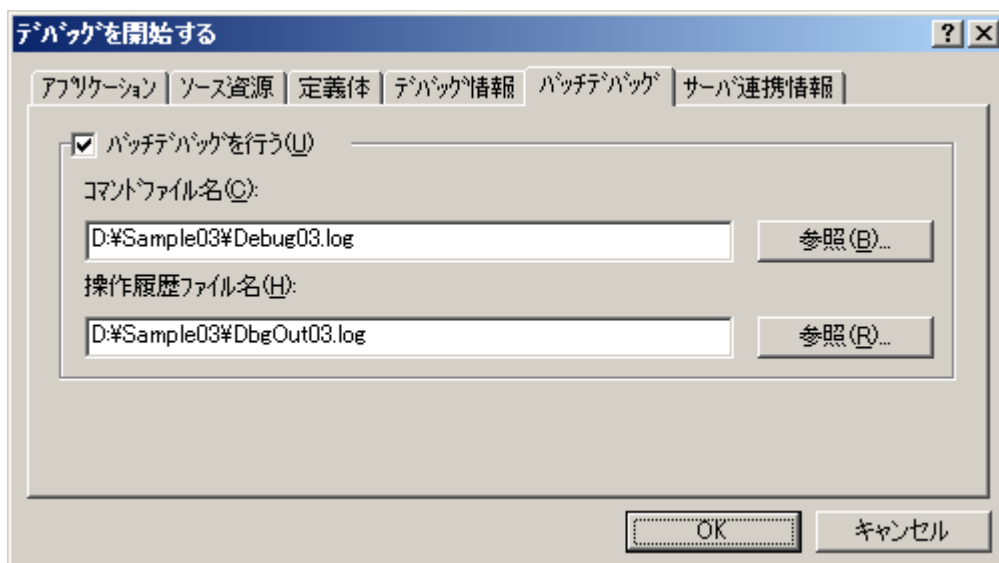
セミコロン（;）から行末までの記述はコメントとして読み飛ばされます。

コマンドの指定形式の詳細については、デバッガのヘルプの“リファレンス”を参照してください。

#### バッチデバッグの手順

1. デバッガを起動し、[デバッグを開始する] ダイアログを表示します。
2. [バッチデバッグ] ページの [バッチデバッグを行う] を指定し、ファイル名の設定を行います。

図4-23 バッチデバッグページ



— コマンドファイル名：

作成したWindowsクライアントのコマンドファイル名を指定します。

— 操作履歴ファイル名

デバッグ結果を出力するWindowsクライアントのファイル名を指定します。

バッチデバッグの実行結果は、ここで指定した操作履歴ファイルに出力されます。

コマンドファイル名と同一のファイル名を指定することはできません。

3. 他のページについては、通常のリモートデバッグ時と同様です。必要な設定を行い [OK] ボタンをクリックしてください。

コマンドファイル内に格納されたデバッグ操作が順次実行されます。

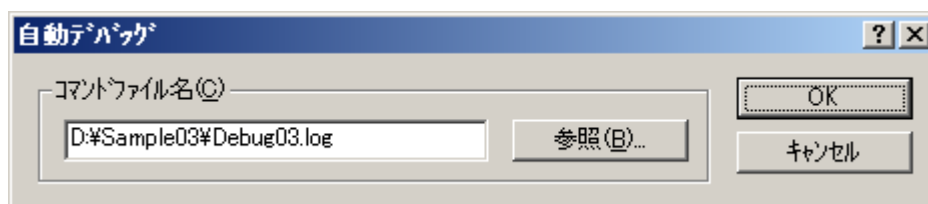
4. 全てのデバッグ操作完了後、デバッグが終了します。

指定した操作履歴ファイルに出力された内容を確認してください。

### 自動デバッグの手順

1. デバッグを起動し、自動デバッグを開始したい位置までデバッグを実行します。
2. [オプション] メニュー - [自動デバッグ] を選択し、[自動デバッグ] ダイアログを表示します。

図4-24 自動デバッグダイアログ



3. [コマンドファイル名] に、作成したWindowsクライアントのコマンドファイル名を指定します。
  4. [OK] ボタンをクリックすると、コマンドファイル内に格納されたデバッグ操作が順次実行されます。
  5. 全てのデバッグ操作が完了すると、完了した時点の実行個所に位置付きます。デバッグを再開してください。
- ただし、コマンドファイルにQUIT コマンドが現れた場合は、その時点でデバッグを終了し、デバッグも終了します。

## 4.4.2 リモートデバッグ時の注意点



### 4.4.2.1 Solaris/Linuxリモートデバッグの注意事項

#### 実行時のコード系の決定方法

リモートデバッグは、以下のいずれかの方法で実行時のコード系を決定します。

- 一般形式のリモートデバッグ起動方法の場合  
UNIXサーバ側のリモートデバッグコネクタを起動した時のコード系に従って動作します。
- アタッチ形式のリモートデバッグ起動方法の場合  
InterstageやWebサーバなどのサーバ配下で動作しているプログラムのコード系に従って動作します。

#### 日本語文字コードに関する注意事項

日本語文字コードの扱いに関して、以下の注意事項があります。

- Unicode固有の文字やEUC固有の文字の扱い  
[データの表示/変更]ダイアログおよび監視ウィンドウでは、Unicode固有の文字は表示されます。ただし、EUC固有の文字は表示されません。  
また、Unicode固有の文字やEUC固有の文字を入力することはできません。Unicode固有の文字やEUC固有の文字を入力したい場合は、16進形式で指定してください。
- バッチデバッグ/自動デバッグ/履歴出力  
この機能で扱うコマンドファイル/操作履歴ファイルのコード系は、UNIXサーバ側のロケールに関係なく、常にシフトJISです。  
そのため、Unicode固有の文字やEUC固有の文字は、“\_”に置き換えて出力されます。

#### その他の注意事項

- アプリケーション名とデバッグ情報ファイル格納フォルダはUNIXサーバ側のフォルダ構成に従って指定します。
- 実行時オプションにフォルダ名やファイル名を指定する場合は、UNIXサーバ側のフォルダ構成に従って指定します。
- UNIXサーバ側のファイル名とフォルダ名は大文字と小文字を区別して指定します。
- アプリケーション名の拡張子を省略しても、拡張子EXEを補いません。





---

## 第5章 トラブルシューティング

---

分散開発は、UNIX系システムとWindows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、UNIX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。なお、OSIVから移行する資産のトラブルについては、“OSIV分散開発の手引き”もあわせて参照してください。

---

## 5.1 サーバ連携のトラブル

以下の作業で起きるトラブルについて説明します。

- 分散開発環境の構築
- サーバへの資産の送信
- ビルド制御文生成
- ターゲットビルド

### 5.1.1 分散開発環境の構築

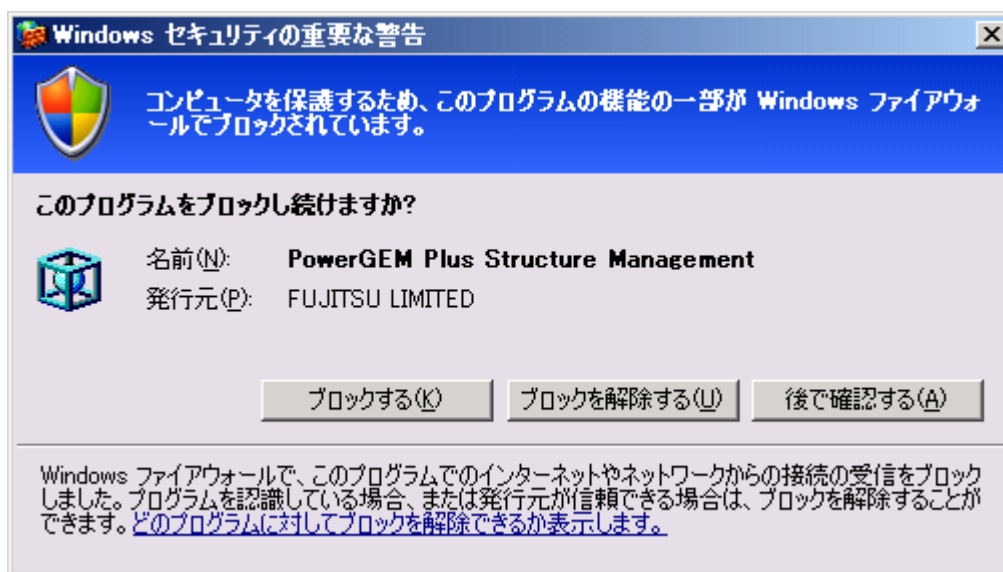
#### Windows XPで、サーバとの連携操作に失敗する

##### 現象

以下に示す操作を行なったとき、エラーメッセージが表示されて失敗する。

- ファイルの送受信
- ターゲットビルド
- リモートデバッグ機能

図5-1 エラーメッセージ



##### 解説

Windows XP SP2でセキュリティ強化のために追加された“Windowsファイアウォール”が有効となっています。

##### 対処

“2.3.2.5 [Windows XP SP2適用時の設定](#)”を参照してください。

#### サーバとの連携操作に失敗する

##### 現象

以下に示す操作を行なったとき、エラーメッセージが表示されて失敗する。

- プロパティの設定や送信/受信でサーバのディレクトリを参照するために、各ダイアログ

で〔参照〕ボタンをクリックした。

- ターゲットビルドをした。

エラーメッセージ

(Linuxの場合)  
ホストでのコマンド処理中にエラーが発生しました。  
stty:標準入力:無効な引数です  
(Solarisの場合)  
ホストでのコマンド処理中にエラーが発生しました。  
stty::引数が正しくありません。

### 対処

.bashrc(bash使用時)、または.cshrc(csh使用時)に記述している、sttyコマンドをコメントにしてください。

例えば、以下のように記述している場合、

```
stty erase ^H
```

次のように変更します。

```
#stty erase ^H
```

## COBOLプロジェクトマネージャがフリーズする

### 現象

以下に示すような、サーバと連携する操作を行なったとき、プロジェクトマネージャや、プロジェクトマネージャの一部のウィンドウがフリーズする。

- サーバディレクトリの参照
- サーバのコマンド実行のための環境変数の情報を表示
- ターゲットビルド  
など。

### 解説

ユーザがサーバへログインする場合に使用する設定ファイルに、ユーザの入力を要求して応答待ちを引き起こすスクリプトなどが記述されている場合に発生します。

### 対処

ユーザの入力を要求するなど、応答待ちになるスクリプトを、サーバへ接続する場合に使用される設定ファイルから削除します。

設定ファイルについては、“2.1.2 [分散開発時のUNIXサーバ上の環境](#)”を参照してください。

## 5.1.2 サーバへの資産の送信

### 送信情報が保存されない

#### 現象

サーバへ資産を送信した時に指定した送信先などの情報が保存されず、次に〔送信〕ダイアログを開いても表示されない。

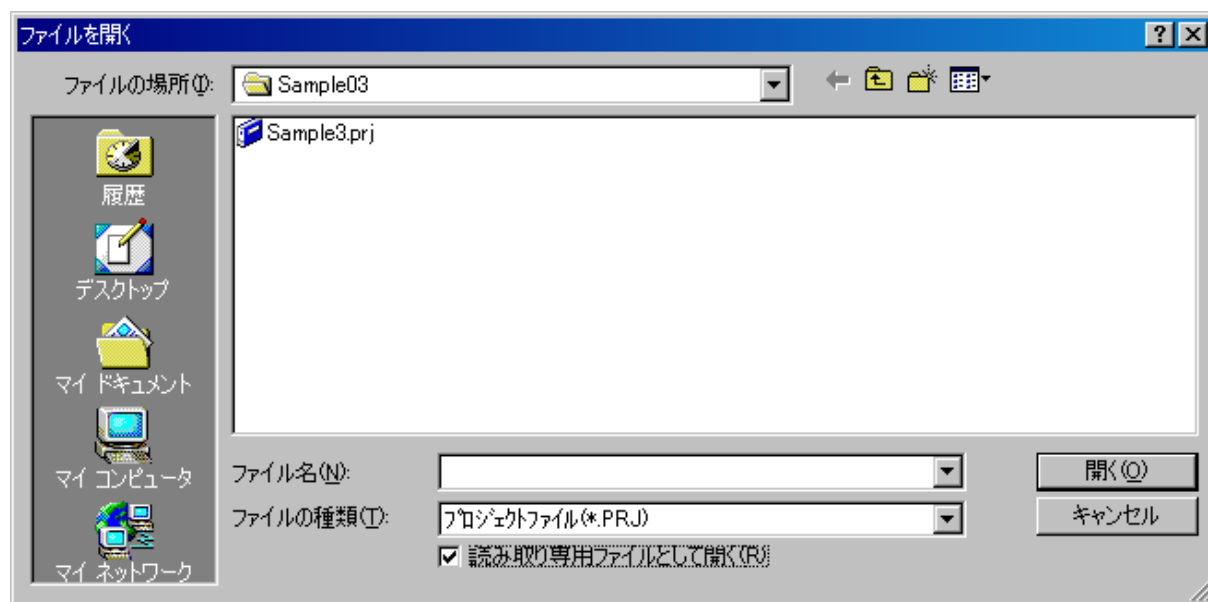
#### 解説

〔送信〕ダイアログへ指定した内容は、プロジェクトファイルに保存されます。そのため、プロジェクトファイルを読み取り専用で開いている場合は情報が保存されません。

プロジェクトファイルが読み取り専用で開かれる原因として、以下の場合が考えられます。

- プロジェクトを開くとき、〔ファイルを開く〕ダイアログの〔読み取り専用ファイルとして開く〕にチェックがついている。

図5-2 〔読み取り専用ファイルとして開く〕チェックボックス



- ファイルの属性が〔読み取り専用〕になっている。

#### 対処

プロジェクトファイルを開いている場合は、一度閉じて、以下のいずれかを行ないます。

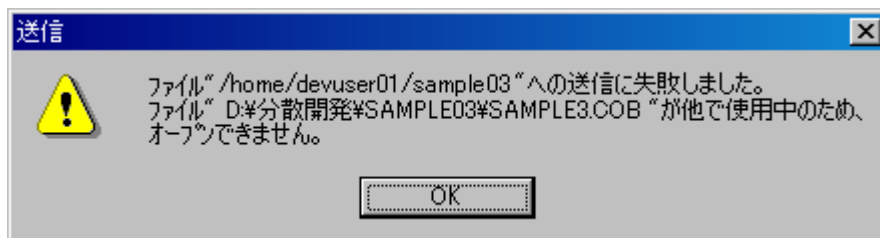
- 〔ファイルを開く〕ダイアログの〔読み取り専用ファイルとして開く〕のチェックを外してプロジェクトを開きなおしてください。
- エクスプローラなどでプロジェクトファイル(拡張子PRJ)のプロパティを開き、ファイルの属性から〔読み取り専用〕を削除してください。

### ファイル編集終了時のサーバへの送信に失敗する

#### 現象

〔プロパティ〕ダイアログで〔ファイル編集終了時のサーバへの送信〕の〔送信する〕にチェックしている場合、エディタでCOBOLソースや登録集を編集し、保存すると、送信がエラーになる。

図5-3 送信エラー



#### 対処

編集後に送信する場合は、エディタの〔ファイル〕メニューの“終了”を選択して、エディタを終了してください。

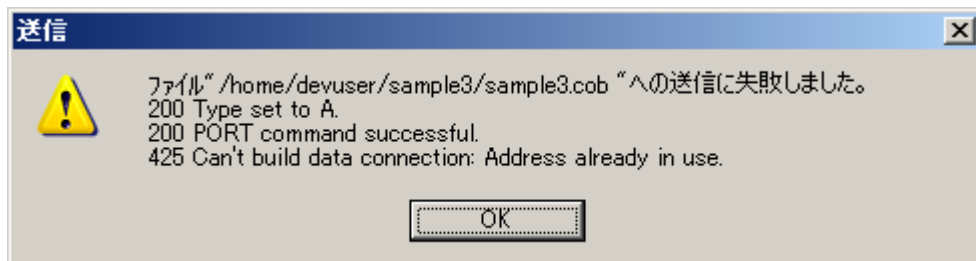
## ファイルの送信に失敗する

### 現象

サーバがSolarisのとき、資産の転送を実行すると、以下の現象が発生する場合がある。

- ファイルの転送状況を示すダイアログが、無反応になる。
- 以下のエラーメッセージが表示される。

図5-4 エラーメッセージ



### 解説

資産の転送に失敗しています。

### 対処

〔サーバ連携情報〕ダイアログの〔ホスト名一覧〕で、資産の転送に失敗したサーバを選択し、〔変更〕ボタンをクリックします。表示した〔変更〕ダイアログの〔PASVモードでファイルを転送する〕チェックボックスをチェックしてください。

〔参照〕“2.3.2.1 [サーバ連携情報の設定](#)”

## 5.1.3 ビルド制御文生成

### ビルド制御文に依存関係が出力されない

#### 現象

ツリーに登録しているファイルの依存関係が、ビルド制御文に出力されない。

#### 解説

〔ビルド制御文生成〕ダイアログの、〔送信済みのファイルだけをビルドの対象にする〕にチェックが付いている場合、サーバへ送信していないファイルや、それらのファイルを元に生成するファイルの依存関係は、ビルド制御文に出力しません。

#### 対処

依存関係を出力するファイルを、サーバへ送信してください。

ファイルを送信せずに、すべてのファイルの依存関係を出力したビルド制御文を生成したい場合は、〔ビルド制御文生成〕ダイアログの、〔送信済みのファイルだけをビルドの対象にする〕のチェックを外してください。〔参照〕“4.2.2 [ビルド制御文雛型の生成手順](#)”

### プリコンパイラを使用しているプロジェクトのビルド制御文生成に失敗する

#### 現象

ツリーに登録しているプリコンパイラの出力ソースまたは、INSDBINFの入力ファイルのパス名が

“¥” 一文字だけの場合、エラーメッセージが表示され、ビルド制御文の生成に失敗する。

エラーメッセージ

以下のファイル名またはディレクトリ名から、UNIXのファイル名またはディレクトリ名への変換に失敗しました。  
UNIXで使⽤できない文字を含んでいないか確認してください。  
/

#### 解説

プリコンパイラの出力ソースまたは、INSDBINFの入力ファイルのパス名が“¥”の場合、ビルド制御文は生成できません。

#### 対処

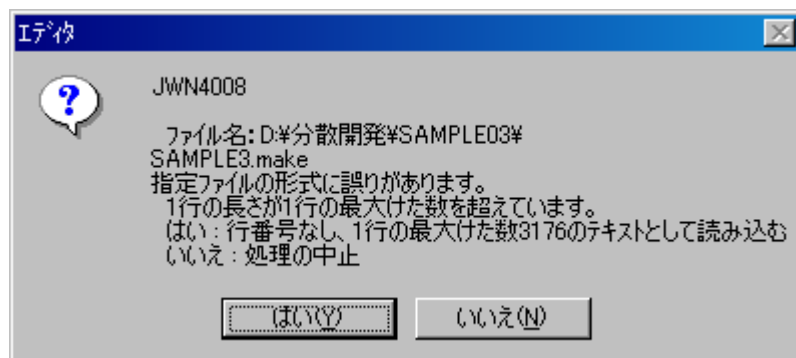
ツリーに登録しているプリコンパイラの出力ソースまたは、INSDBINFの入力ファイルのパス名を、“¥”以外のものに変更してください。

### ビルド制御文をエディタで開けない

#### 現象

生成したビルド制御文を、NetCOBOL組み込みのエディタで開こうとすると、エラーメッセージが表示され、開くことができない。

図5-5 エラーメッセージ



#### 解説

生成したビルド制御文に、一行の長さが3176桁を超える行が含まれています。  
例えば、プロジェクトに、COBOLソースや登録集を、大量に登録している場合などに、ビルド制御文の中に長い行が生成されることがあります。

#### 対処

エディタのカスタマイズで、長い行を表示することができるエディタを指定してください。  
または、そのようなエディタをNetCOBOLの外で起動し、メイクファイルを開いてください。

## 5.1.4 ターゲットビルド

### エラーメッセージ「依存関係が破棄されました」が表示される

#### 現象

ターゲットビルドを行なった場合、[ターゲットビルド] ウィンドウに、次のようなメッセージ

が表示される。

エラーメッセージ

```
make : 循環 COBOLソースファイル名 <- オブジェクトファイル名 依存関係が破棄されました。
```

### 解説

COBOLソースファイルの拡張子がcob、cobl、cbl以外の場合に表示されることがあります。

### 対処

COBOLソースファイルの拡張子はcob、cobl、cblのいずれかを使用してください。

## コンパイルエラー以外のエラーで失敗する

### 現象

ターゲットビルドを行なった場合、コンパイルエラー以外のエラーが発生する。

### 解説

様々な要因が考えられますが、おもに以下の原因が考えられます。

- プロジェクトマネージャの「サーバ連携情報」ダイアログから表示した、「追加」または「変更」ダイアログで設定したホストのコードが、連携するサーバの環境変数LANGと一致していない
- ビルドを行なうサーバの環境構築に失敗している。
- ビルド制御文の編集に失敗している。

### 対処

- プロジェクトマネージャの「サーバ連携情報」ダイアログから表示する、「追加」または「変更」ダイアログで指定するホストのコードと、サーバの環境変数LANGを一致させてください。[参照] “2.3.1.2 [UNIXサーバ側のユーザ環境の設定](#)”
- 環境の設定については、“2.3 [分散開発のための環境設定](#)”を参照してください。
- ビルド制御文の編集については、“4.2 [ビルド制御文生成機能](#)”を参照してください。

## 各種定義体を使用したアプリケーションのビルドに失敗する

### 現象

各種定義体を使用したアプリケーションをビルドした場合、次のようなエラーが発生する。

```
JMN1671I-S 登録集原文の組込み中に入出力エラーが発生しました。登録集原文の組込みを中止します。
```

### 解説

分散開発の場合、原因として、以下が考えられます。

- 各種定義体ファイルが、何らかの原因により壊れた。
- 各種定義体ファイルを、データの種別にテキストを指定してサーバへ送信した。

### 対処

- Windows上の開発ツールで、各種定義体ファイルを開き、ファイルが正しいことを確認してください。
- 各種定義体を送信する場合のデータの種別が、バイナリになっていることを確認して、サーバへ送信しなおしてください。  
[参照] “4.1.2 [各種定義体の送信](#)”

## 変更のないファイルのビルドが実行される

### 現象

〔制御文生成〕ダイアログの、〔翻訳オプション〕ページで、翻訳オプション-dr(リポジトリファイルの入出力先ディレクトリの指定)を指定している場合、依存するファイルの変更がないにもかかわらず、リポジトリを登録しているCOBOLソースの翻訳が行なわれる。

### 解説

以下の原因が考えられます。

- -drオプションに、次に示すようなディレクトリを指定した。
  - ― ディレクトリ名に空白を含む かつ
  - ― ディレクトリ名を"\"で囲んでいない

このような指定をした場合、ビルド制御文に、-drが複数出力されます。そのため、依存関係を正しく調査することができず、常にビルドを行なうことになります。

### 対処

〔翻訳オプション〕ページで、空白を含むディレクトリ名を指定する場合は、"\"で囲んでください。

## プリコンパイルインクルードファイルがないためビルドに失敗する

### 現象

多段プリコンパイルを行なうプロジェクトで、ビルド制御文を生成してターゲットビルドを行なった場合、ビルドが失敗して、ターゲットビルドウィンドウに、以下のメッセージが表示される場合がある。

メッセージ

make:\*\*\*' COBOLソースファイル名'に必要なターゲット'インクルードファイル名'をmakeするルールがありません. 中止。

### 解説

以下の原因が考えられます。

- ビルド制御文の依存関係に、送信対象ではないインクルードファイルが出力されている。

### 対処

ビルド制御文をエディタで開き、インクルードファイル名を削除してください。

## エラージャンプでカーソルが正しい行に位置付かない

### 現象

ターゲットビルドで〔ターゲットビルド〕ウィンドウに表示している、診断メッセージを選択してダブルクリックしても、エラーの発生した行に位置付かない。

### 解説

- PowerGEM Plusエディタの〔ツール連携の動作環境〕ダイアログで、〔ツールからの位置付け要求〕が〔相対行〕になっていないと、ビルダからのエラージャンプ機能で正しい行にカーソルが位置付きません。
- エディタのカスタマイズをしている場合、タグジャンプをサポートしているエディタではない場合は位置付きません。

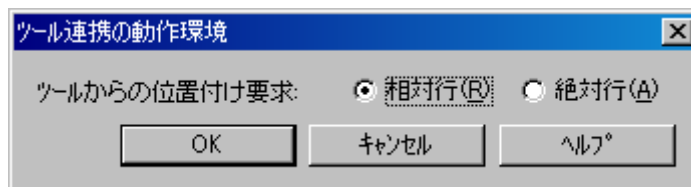


## 対策

以下のいずれかの処置を行ないます。

- PowerGEM Plusエディタの「ツール連携の動作環境」ダイアログで、「ツールからの位置付け要求」に「相対行」を指定します。
- 1. エディタの「ツール」メニューから、「ツール連携の動作環境」を選択します。
- 2. 表示された「ツール連携の動作環境」ダイアログで、「相対行」を選択します。

図5-6 「ツール連携の動作環境」ダイアログ



- 3. 「OK」ボタンをクリックします。

- エディタのカスタマイズをする場合は、タグジャンプをサポートしているエディタを指定します。

## 意図しないプログラムが実行される

### 現象

共用オブジェクトプログラムをリンクしているアプリケーションを実行した場合、意図しないプログラムが実行される。

### 解説

以下の原因が考えられます。

- 同じ名前の共用オブジェクトプログラムファイルを、複数リンクしている。
- 実行形式ファイル、リンクしている共用オブジェクトプログラムファイルに、同じ名前の呼ばれるプログラムが複数存在する。

上記の原因により、意図しないプログラムが呼ばれている可能性があります。

### 対処

アプリケーション内で使用する共用オブジェクトプログラム名、プログラム名は、全体を通して一意になるように変更してください。

## 5.2 リモートデバッグのトラブル

以下の作業で起きるトラブルについて説明します。

- リモートデバッグの準備
- リモートデバッグの開始
- リモートデバッグの操作

### 5.2.1 リモートデバッグの準備

#### サーバ側のリモートデバッガコネクタの起動に失敗する

##### 現象

一般形式のリモートデバッグにおいて、サーバ側のリモートデバッガコネクタの起動を行なったとき、エラーメッセージが表示されて失敗する。

エラーメッセージ

Address already in use

##### 解説

サーバ側のリモートデバッガコネクタを起動するとき、クライアント側との通信に必要なポートを開こうとしますが、指定されたポート番号(指定されない場合には標準のポート(59998))は既に使用されているため、ポートが使用できずに起動に失敗します。

ポート番号が既に使用されている原因として、以下の場合が考えられます。

- 別のリモートデバッガコネクタで既に使用している。
- 以前リモートデバッグを行ったプログラムがゾンビプロセスとして残っている。

##### 対処

以下のいずれかを行ないます。

- 別のポート番号を指定して、サーバ側のリモートデバッガコネクタを起動し直してください。
- ゾンビプロセスを終了させてから、リモートデバッガコネクタを起動し直してください。
  - 1) psコマンドを実行して、実行されているプロセス一覧を表示します。(※)
  - 2) 表示されたプロセスのうち、以下がゾンビプロセスの場合には、killコマンドで終了させます。(※)
    - (ア) サーバ側のリモートデバッガコネクタ(svdrrds)
    - (イ) サーバ側のデバッガ(svdrrdm)
    - (ウ) デバッグ対象プログラム
  - 3) サーバ側のリモートデバッガコネクタを起動し直します。

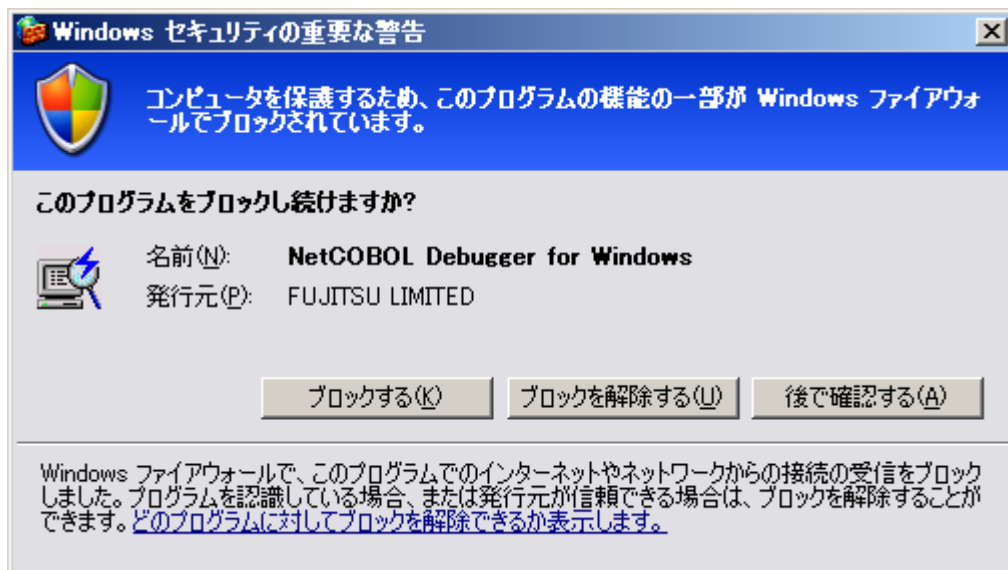
※ 開発管理者のみ実施してください。一般の開発者は開発管理者に相談してください。

#### Windows XPで、クライアント側のリモートデバッガコネクタの起動に失敗する

##### 現象

アタッチ形式のリモートデバッグにおいて、クライアント側のリモートデバッガコネクタの起動を行なったとき、エラーメッセージが表示されて失敗する。

図5-7 エラーメッセージ

**解説**

Windows XP SP2でセキュリティ強化のために追加された“Windowsファイアウォール”が有効となっています。

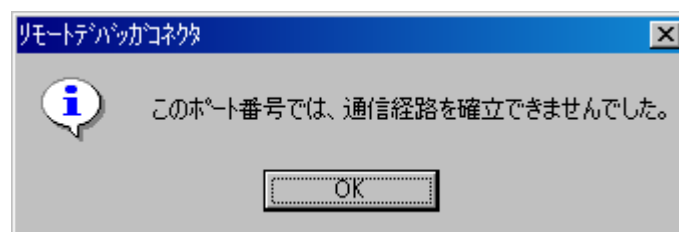
**対処**

“2.3.2.5 [Windows XP SP2適用時の設定](#)”を参照してください。

**クライアント側のリモートデバッグコネクタの起動に失敗する****現象**

アタッチ形式のリモートデバッグにおいて、クライアント側のリモートデバッグコネクタの起動を行なったとき、エラーメッセージが表示されて失敗する。

図5-8 エラーメッセージ

**解説**

クライアント側のリモートデバッグコネクタを起動するとき、クライアント側との通信に必要なポートを開こうとしますが、設定されているポート番号は既に使用されているため、ポートが使用できずに起動に失敗します。

ポート番号が既に使用されている原因として、以下の場合が考えられます。

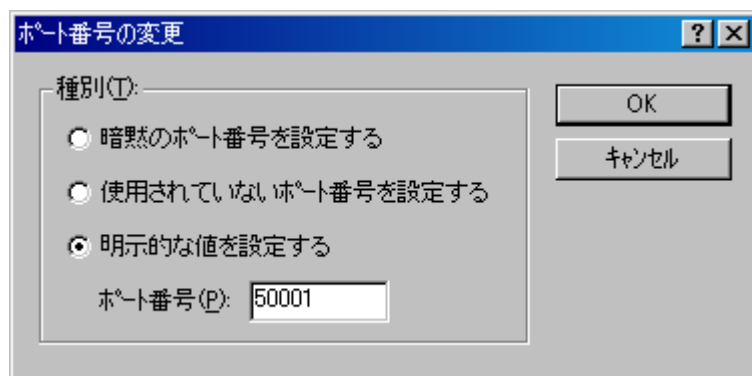
- 別のアプリケーションで既に使用している。
- 以前リモートデバッグを行ったプログラムがゾンビプロセスとして残っている。

**対処**

以下のいずれかを行ないます。

- 「ポート番号の変更」ダイアログで、別のポート番号を指定します。
  - 1) タスクトレイに常駐しているリモートデバッガコネクタのアイコンを右クリックして、表示されるメニューから“環境設定”を選択します。
  - 2) 「リモートデバッガコネクタ」ダイアログの「ポート番号の変更」ボタンをクリックします。
  - 3) 「ポート番号の変更」ダイアログで、別のポート番号を指定します。

図5-9 「ポート番号の変更」ダイアログ



- 4) 「ポート番号の変更」ダイアログの「OK」ボタンをクリックします。
  - 5) 「リモートデバッガコネクタ」ダイアログの「OK」ボタンをクリックします。
- 以前リモートデバッグを行ったプログラムがゾンビプロセスとして残っている。
    - 1) psコマンドを実行して、実行されているプロセス一覧を表示します。(※)
    - 2) 表示されたプロセスのうち、以下がゾンビプロセスの場合には、killコマンドで終了させます。(※)
      - (ア) サーバ側のリモートデバッガコネクタ (svdrds)
      - (イ) サーバ側のデバッガ (svdrdm)
      - (ウ) デバッグ対象プログラム
    - 3) クライアント側のリモートデバッガコネクタを起動し直します。
- ※ 開発管理者のみ実施してください。一般の開発者は開発管理者に相談してください。

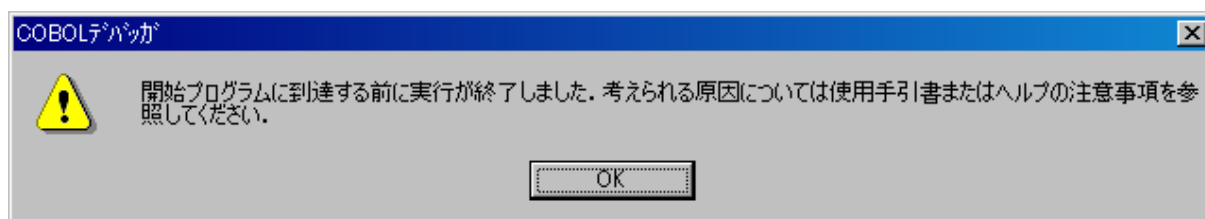
## 5.2.2 リモートデバッグの開始

### リモートデバッグが開始されずにプログラムが終了する

**現象**

リモートデバッグ時、「デバッグを開始する」ダイアログの「OK」ボタンをクリックして、デバッグを開始するが、次のメッセージボックスが出力されて、デバッグができない。

図5-10 デバッグが開始されずにプログラムが終了した場合のメッセージ



**解説**

分散開発支援機能を使用して、リモートデバッグを行う場合、COBOLプロジェクトマネージャで〔デバッグモジュールの作成〕が無効になっている状態で生成したビルド制御文を使用してビルドしたプログラムをデバッグしようとしたことが原因です。〔デバッグモジュールの作成〕が無効になっていることは、次のいずれかの方法で確認できます。

1. ツールバー上でビルドモードがリリースと表示されている。

図5-11 ツールバー



2. 〔プロジェクト〕メニューから、〔プロジェクト〕－〔オプション〕－〔デバッグモジュールの作成〕がチェックされていない。

**対処**

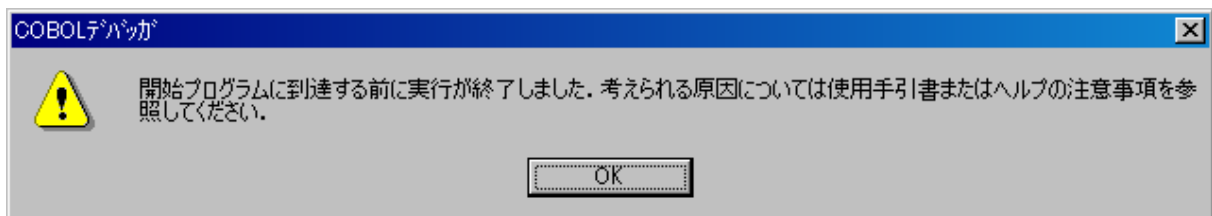
COBOLプロジェクトマネージャの設定で〔デバッグモジュールの作成〕を有効にして、再度以下の操作を行ってデバッグ対象プログラムを作り直してください。

1. ビルド制御文生成
2. 生成したビルト制御文の送信
3. ターゲットビルド

**デバッグが開始されずにプログラムが終了する****現象**

〔デバッグを開始する〕ダイアログの〔OK〕ボタンをクリックしてデバッグを開始するが、次のメッセージボックスが出力されて、デバッグができない。

図5-12 デバッグが開始されずにプログラムが終了した場合のメッセージ

**解説**

原因として、以下が考えられます。

- 〔デバッグを開始する〕ダイアログの〔開始プログラム名〕に、制御が渡らないプログラム名を指定した。
- 〔デバッグを開始する〕ダイアログの〔開始プログラム名〕に、誤ったプログラム名（存在しないプログラム名）を指定した。
- 〔デバッグを開始する〕ダイアログの〔デバッグ情報ファイル格納フォルダ〕に、デバッグ情報ファイルが存在しないフォルダを指定した。

**対処**

- 〔デバッグを開始する〕ダイアログの〔開始プログラム名〕に、制御が渡るプログラム名を指定してください。
- 〔デバッグを開始する〕ダイアログの〔開始プログラム名〕に、正しいプログラム名（存在するプログラム名）を指定してください。
- 〔デバッグを開始する〕ダイアログの〔デバッグ情報ファイル格納フォルダ〕に、デバッグ情報ファイルが存在するフォルダを指定してください。なお、複数のフォルダを指定し

た場合、途中に存在しないフォルダを指定すると、それ以降のフォルダにデバッグ情報ファイルが存在していても検索されませんので注意してください。

## デバッグが開始されず、プログラムも終了しない

### 現象

〔デバッグを開始する〕ダイアログの〔OK〕ボタンをクリックしてデバッグを開始するが、デバッグの応答がなくなり、デバッグができない。

### 解説

原因として、以下が考えられます。

- デバッグ対象でないプログラム（C言語プログラム等）が無限ループして、デバッグ対象プログラム（COBOLプログラム）まで制御が渡ってこない。

### 対処

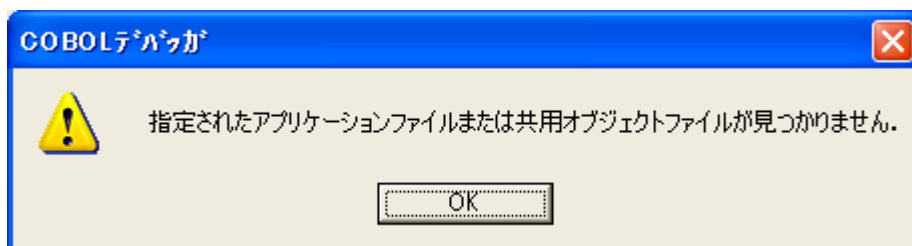
- アテンションキーを押して、デバッグを終了してください。

## アプリケーションが見つからないエラーが出る

### 現象

〔デバッグを開始する〕ダイアログの〔OK〕ボタンをクリックしてデバッグを開始するが、次のメッセージボックスが出力されて、デバッグができない。

図5-13 アプリケーションが見つからないエラーメッセージ



### 解説

原因として、以下が考えられます。

- 〔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、誤った相対パスを指定した。
- 環境変数LD\_LIBRARY\_PATHに誤ったパスを指定した。

### 対処

- 〔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、パスなしでアプリケーション名だけを指定するか、または絶対パスを指定してください。
- 環境変数LD\_LIBRARY\_PATHに、正しいパス（共用オブジェクトライブラリが存在するパス）を指定してください。

## SolarisリモートまたはLinuxリモートが起動しない

### 現象

SolarisリモートデバッグまたはLinuxリモートデバッグが起動しない。

**解説**

原因として、以下が考えられます。

- COBOLプロジェクトマネージャの〔プロジェクト〕メニューの〔デバッグ〕を選択して起動した。
- COBOLプロジェクトマネージャの〔ツール〕メニューの〔デバッグ〕を選択して起動した。

**対処**

- COBOLプロジェクトマネージャの〔プロジェクト〕メニューの〔リモートデバッグ〕を選択して起動してください。

**アタッチ形式リモートデバッグでクライアントとの接続に失敗する****現象**

デバッグしたいプログラムを実行すると、次のエラーメッセージが出力されて、アタッチ形式のリモートデバッグでクライアントとの接続に失敗する。

COBOL:rts: WARNING: JMP0774I-W [PID:000031E7 TID:00000001] リモートデバッグにおいて、クライアントとの接続に失敗しました。 20.

**解説**

原因として、以下が考えられます。

- 環境変数CBR\_ATTACH\_TOOLに、誤った接続先を指定した。
- 〔リモートデバッグコネクタ〕ダイアログの〔接続制限〕ページで、サーバ側コンピュータからの接続を拒否する指定を行った。

**対処**

以下のいずれかを行ないます。

- 環境変数CBR\_ATTACH\_TOOLに、正しい接続先（IPアドレス、ホスト名、ポート番号）を正しい書き方で指定します。接続先には、Windowsクライアント側のリモートデバッグコネクタの位置とポート番号を指定します。
- 〔リモートデバッグコネクタ〕ダイアログの〔接続制限〕ページで、サーバ側コンピュータからの接続を許可する指定を行います。
  - 1) タスクトレイに常駐しているリモートデバッグコネクタのアイコンを右クリックして、表示されるメニューから〔環境設定〕を選択します。
  - 2) 〔リモートデバッグコネクタ〕ダイアログの〔接続制限〕ページで、〔許可する対象〕ラジオボタンをクリックし、〔追加〕ボタンをクリックします。
  - 3) 〔制限対象〕ダイアログで、サーバ側コンピュータのIPアドレスまたはホスト名を指定し、〔OK〕ボタンをクリックします。
  - 4) 再表示された〔リモートデバッグコネクタ〕ダイアログで、〔OK〕ボタンをクリックします。

図5-14 「リモートデバッガコネクタ」ダイアログ

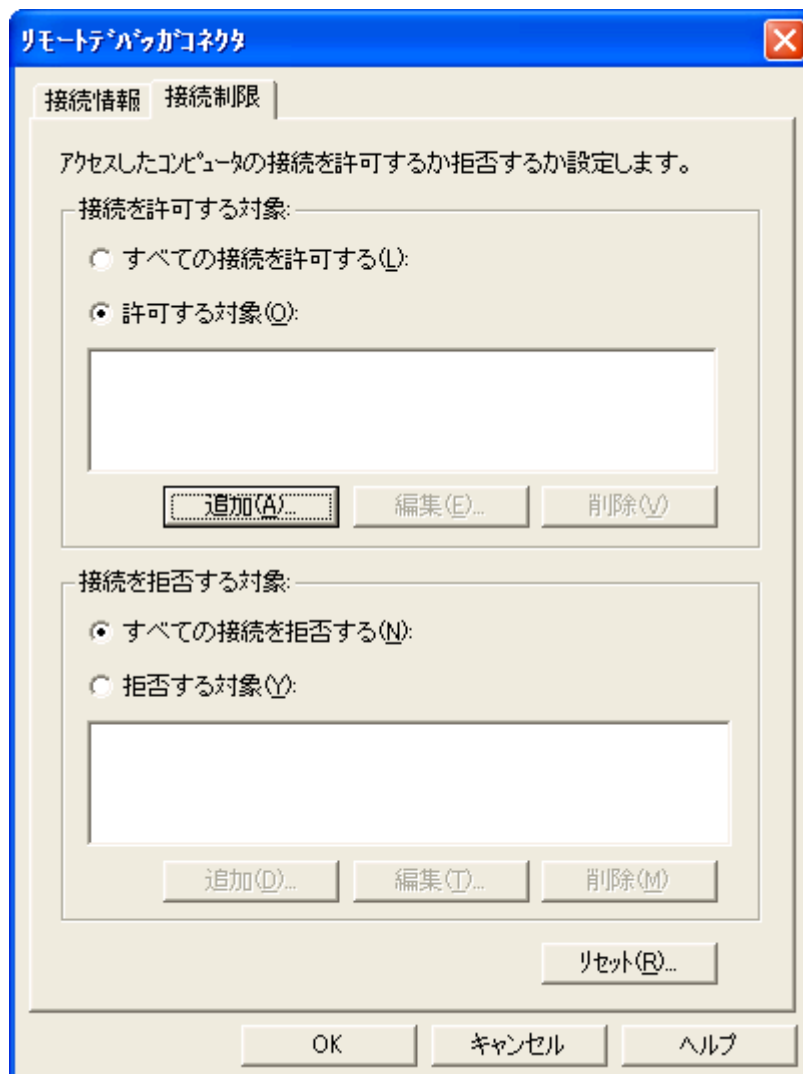
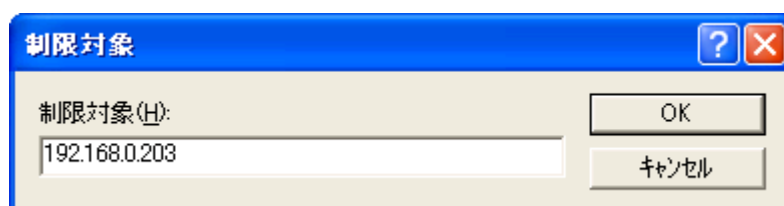


図5-15 「ポート番号の変更」ダイアログ



### 5.2.3 リモートデバッグの操作

#### クライアント側デバッガの応答がなくなる

##### 現象

デバッグ操作しているときに、クライアント側デバッガの応答がなくなってしまう。

##### 解説

様々な要因が考えられますが、主に以下の原因が考えられます。



- a. 被デバッグプログラムがキー入力 (ACCEPT文, STOP文など) を要求している。
- b. 被デバッグプログラムが無限ループをしている。
- c. マルチスレッド環境において、他言語プログラムとの間でスレッドデッドロックが発生している。

#### 対処

- a. 端末からキー入力ができる場合は、キー入力を行って下さい。端末からキー入力できない場合は、アテンションキーを押してデバッグを終了して下さい。
- b. アテンションキーを押して無限ループを一時停止し、ループ処理を迂回するなどの操作をしてください。
- c. アテンションキーを押してデバッグを終了してください。デバッグはマルチスレッドのプログラムをデバッグできません。

### 被デバッグプログラムの応答がなくなる

#### 現象

タッチ形式でリモートデバッグをしているときに、クライアント側デバッグを強制終了しても、サーバ側の被デバッグプログラムの実行継続がされない。

#### 解説

様々な要因が考えられますが、主に以下の原因が考えられます。

- a. ネットワーク環境など、何らかの理由により、被デバッグプログラムをデバッグ配下から解放できなかった。

#### 対処

- a. 以下の方法で、被デバッグプログラムのプロセスを強制中断させてください。
    - コンソール画面からCtrl+Cを入力する。
    - コンソール画面を別途立ち上げ、被デバッグプログラムのプロセスを強制終了させます。
      - 1- psコマンドを実行して、実行されているプロセス一覧を表示します。(※)
      - 2- 表示されたプロセスのうち、被デバッグプログラムのプロセスIDに対してkillコマンドを実行し、プロセスを終了させます。(※)
- ※ 開発管理者のみ実施してください。一般の開発者は開発管理者に相談してください。



## 付録A NetCOBOL製品の相違点

COBOLは互換性がきわめて重要視される言語です。しかし、プラットフォームの機能差や連携するミドルウェアとの関係から、完全な互換性を保証することはできません。このため、同じNetCOBOL製品であっても、各プラットフォーム向けの製品間に細かな仕様上、機能上の違いが存在します。

ここでは各プラットフォーム向けのNetCOBOL製品間の仕様上／機能上の相違点について、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製品の概要

ここでは富士通のCOBOL製品の概要について説明します。

#### A.1.1 富士通のCOBOL製品の変遷

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(最新はV12L20 PTF L04061)。

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製品の製品系列とその変遷を示します。

表A-1 富士通のCOBOL製品の変遷

時期 (西暦)	汎用機 (OSIVシステム)		オープン系プラットフォーム	
	製品名	主な機能追加等	製品名	主な機能追加等
1985 : 1989 1990 1991 1992 1993 1995 1997 1998 1999 2000 2001 2002 2003 2004 2005	COBOL85	初版提供開始 : V12L20提供開始 SQL機能対応 RDB II R/W対応 : トランザクション出口対応 : 通信データベース機能対応 AIMエンハンス対応 : USAGE句に新機能追加	          PowerCOBOL85          PowerCOBOL97       NetCOBOL	          Solaris、Windows対応 XPG仕様、組込み関数          オブジェクト指向機能       .NET Framework対応 Linux32対応   Linux64対応

## A.2 言語の機能の違い

各プラットフォーム向けのNetCOBOL製品間の機能差を言語の機能に絞って説明します。

### A.2.1 概要

以下に、各プラットフォーム向けのNetCOBOL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-1 NetCOBOL製品間の機能差

【Windows システムの機能範囲】		【UNIX 系システムの機能範囲】	
《国際規格 COBOL85》		《国際規格 COBOL2002》	
－ 中核	***	－ オブジェクト指向機能	***
－ 順ファイル	***	－ 利用者定義型	***
－ 相対ファイル	***	－ Unicodeサポート	*
－ 索引ファイル	***	－ 2進データ型(BINARY-SHORT 等)	***
－ プログラム間連絡	***		
－ 整列併合	***	《Micro Focus 互換仕様》	
－ 原始文操作	***	－ スクリーン操作	**
－ 報告書作成	***	－ 名前付き定数(78レベル)	***
－ デバッグ	***	－ 16進数字定数	***
－ 区分化	***		
－ データベース操作(SQL)	*	《富士通拡張仕様》	
		－ 表示ファイル(画面, 帳票)	*
《XPG仕様》		－ 表示ファイル(APL, ACM)	**
－ 行順ファイル	***	－ 日本語プログラミング	***
－ C言語間結合	***	－ 日本語処理	*
(値渡し、復帰値)		－ 拡張日本語印刷	*
－ ファイル共用/レコード排他	***	－ ビット操作	***
－ スクリーン操作	**	－ 浮動小数点数操作	***
－ コマンド行引数操作	***	－ FORMAT 句付き印刷ファイル	***
－ 環境変数操作	***	(順ファイル)	
－ 連結式	***	－ 拡張索引ファイル	***
		(多重キー、逆順検索)	
《国際規格 1989 年追補》		－ 定数節	***
－ 組み込み関数	***	－ システムプログラム記述(SD)	***
		－ マルチスレッド対応	***
		－ Web連携機能	*
		－ COM連携機能	

#### 記号の説明

- \*\*\* : 共通機能範囲
- \*\* : Linux 版に機能差あり
- \* : Windows-UNIX 系間で機能差あり

“図A-1 [NetCOBOL製品間の機能差](#)”で示した機能差の理由の1つは、各プラットフォーム向けの製品での機能のサポート状況の違いです。“図A-1 [NetCOBOL製品間の機能差](#)”で示した機能差のある機能のサポート状況を“表A-2 [各プラットフォーム向けのNetCOBOL製品の機能サポート状況](#)”に示します。

表A-2 各プラットフォーム向けのNetCOBOL製品の機能サポート状況

No	機能名	概要	OS毎のサポート状況		
			Solaris	Linux32	Linux64
1	日本語処理	COBOLのデータ項目で、日本語データを使用する機能。	●	●	●
2	Unicodeサポート	COBOLプログラムの使用する文字コード系としてUnicodeを使用する機能。	●	●	●
3	JEF/EBCDICサポート	COBOLプログラムの使用する文字コード系としてJEF/EBCDICを使用する機能。	×	×	×
4	拡張日本語印刷	フォームオーバーレイパターン、帳票定義体、FCBなどを使用した高度な印刷機能	●	●	●
5	表示ファイル(画面)	表示ファイルを用いて、画面入出力を行う機能。	△	×	×
6	表示ファイル(印刷)	表示ファイルを用いて、帳票出力を行う機能。	●	●	●
7	表示ファイル(宛先APL/ACM)	表示ファイルを用いて、同期(APL)/非同期(ACM)のプロセス間通信を行う機能。	△	×	×
8	表示ファイル(多画面)	多画面定義体を使用することで、画面設計に依存しないプログラミングを可能とする機能。	×	×	×
9	スクリーン操作	ACCEPT/ DISPLAY文を用いて、レイアウト付きの画面からの入出力をする機能。	△	×	×
10	データベース操作	埋め込みSQL文を使用して、COBOLプログラムからデータベースを操作する機能。	●	●	●
11	Web連携機能	COBOLでWebアプリケーションを開発する機能。	△	△	×
12	COM連携機能	COBOLでCOMサーバ/COMクライアントを作成する機能	×	×	×

表中の記号の意味:

Windows版の製品の機能を基準に

- : 同じ動作を期待できない。
- △ : 機能に制限がある。
- × : 機能が提供されていない。

## A.2.2 NetCOBOL製品間の機能差の詳細

“図A-1 [NetCOBOL製品間の機能差](#)” / “表A-2 [各プラットフォーム向けのNetCOBOL製品の機能サポート状況](#)” で示した各プラットフォーム向け製品間での機能差について、次のようなものについて、ここでは機能ごとに詳しく説明します。

- 機能は提供されているが、同じ動作が期待できない。
- 機能は提供されているが、制限がある。

### 日本語処理

NetCOBOLでは、COBOLのデータ項目で日本語データ(半角カナ文字を含む)を使用することができます。しかし、各プラットフォームでの文字コードの違いから、各プラットフォーム向けのNetCOBOL製品での日本語データの扱いに違いが生じています。したがって、日本語処理の結果の

違いについて詳細を理解するためには、文字コード系について理解することが必要となります。  
ここでは概要レベルの説明に止め、詳細については“付録B [文字コード系](#)”で説明します。

## 日本語処理に使用可能な文字コード系

以下に、各NetCOBOL製品で日本語処理に使用可能な文字コード系を示します。

**表A-3 各NetCOBOL製品で日本語処理に利用可能な文字コード系の違い**

文字コード系	Windows	Solaris	Linux32	Linux64
シフトJIS (SJIS)	○	○	×	×
日本語EUC (EUC)	×	○	○	○
Unicode	○	○	○	○

なお、Unicodeサポートについては、実際は更に詳細な違いがありますが、それについては後述の“Unicodeサポート”を参照してください。

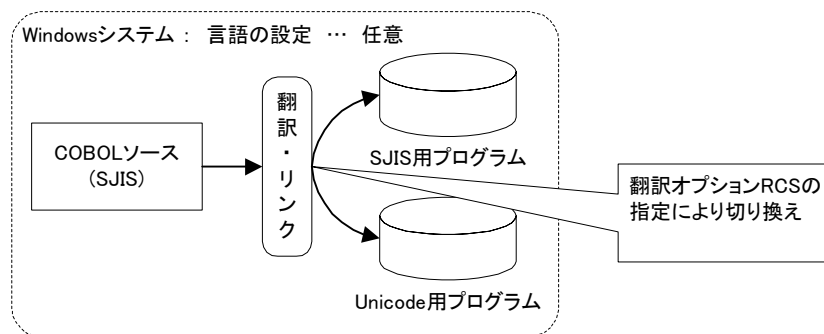
## プログラム動作時の文字コード系の選択

COBOLプログラムが動作する場合の文字コードの選択と指定の方法が、NetCOBOLのWindowsシステム向け製品とUNIX系システム向け製品で根本的に異なります。

### ● Windowsシステム

翻訳オプションRCSによって指定します(デフォルトはシフトJIS)。このため、同じCOBOLソースから異なるコード系を使用するプログラムを作成することができます。

**図A-2 Windowsシステムでの文字コード系の選択**

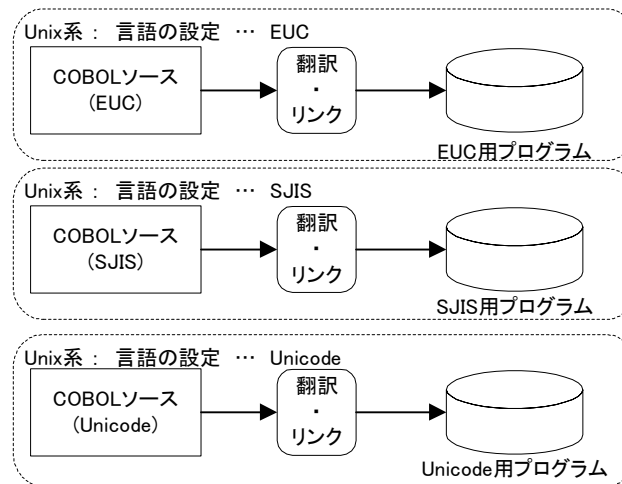


作成した実行可能プログラムは、Windowsシステムの言語の設定に依存することなく、翻訳時に指定した文字コード系を使用して動作します。

### ● UNIX系システム

UNIX系システムの環境変数LANGの設定によって、翻訳時および実行時に使用される文字コード系が自動的に選択されます。このため、実行時に使用する文字コード系を使用して記述されたCOBOLソースが必要となります。

図A-3 UNIX系システムでの文字コード系の選択



作成した実行可能プログラムは、UNIXシステム的环境変数LANGの設定に従って動作します。このため、COBOLソースで使用している文字コード系と実行時の環境変数LANGの設定が食い違った場合、実行時エラーが発生します。

### 日本語処理結果の違いの概要

詳細は“付録B [文字コード系](#)”で説明しますが、次のような操作で結果が異なる場合があります。

- 日本語定数/日本語16進定数  
使用を選択した文字コード系に含まれない値を指定した場合、翻訳エラーになります。
- 文字比較  
半角カナ文字、日本語文字の大小順序の結果が異なる場合があります。
- 索引ファイルのキー順序  
半角カナ文字、日本語文字の大小順序の結果が異なる場合があります。
- ソートキーの順序  
半角カナ文字、日本語文字の大小順序の結果が異なる場合があります。
- 日本語字類条件  
コード系の違いにより、結果が異なる場合があります。

### 拡張日本語印刷

NetCOBOLでは、フォームオーバーレイパターン、FCB、帳票定義体などの帳票資源を使用してきめ細かい帳票印刷を行うことができます。しかし、プラットフォームごとに提供される印刷環境の違いやプリンタに出力する印刷オーダーなどの違いから、プラットフォーム間での動作に違いが生じています。更に各プラットフォームで利用可能なフォント、プリンタおよび接続製品提供の違いがありますので、各プラットフォームで全く同じ印刷結果を得ることは困難です。このため、印刷機能を使用した帳票出力結果の確認は、運用するプラットフォーム上で実際の印刷環境を使用して確認することが必要となります。

### 印刷オーダー

各プラットフォームで出力できる印刷オーダーは以下のとおりです。

	Windows	Solaris	Linux32	Linux64
GDI	○	×	×	×
FM(UVPI)	×	○	×	×
FP(UVPI)	×	○	○	○
PostScript	×	○	○	○
ESC/P	×	○	×	×
ESC/Page	×	○	×	×
LIPS	×	○	×	×

なお、MeFt経由で印刷する場合には、MeFtのマニュアルを参照してください。

## フォームオーバーレイパターン

フォームオーバーレイパターンはWindows版のFORM/PowerFORMで作成します。

FORM/PowerFORMで作成したフォームオーバーレイパターンは他のプラットフォームでも使用することができます。ただし、フォームオーバーレイパターンの形式によっては、プラットフォームごとに利用可能な機能範囲が異なります。プラットフォーム間で共用する目的でフォームオーバーレイパターンを作成する場合には、KOL5形式(システム間流通可能な形式)で作成することをお勧めします。

各プラットフォームで使用できる機能については、FORMのインストールフォルダに格納される「オーバーレイ定義機能(OVDDEF.TXT)」を参照してください。

## FCB

FCBは、印刷ファイルを使用して行レコードで構成された帳票を出力する場合に、1ページ分の論理的な構成(行数、行間隔、印字開始位置)を定義します。各プラットフォームで定義できる情報は以下のとおりです。

	Windows	Solaris	Linux32	Linux64
定義方法	FCB制御文	FCBファイル	FCBファイル	FCBファイル
LPI情報(行間隔)	○	○	○	○
LPI情報(行数)	○	○	○	○
CH情報(チャンネル)	○	×	×	×
SIZE情報(用紙サイズ)	○	○	○	○
FORM情報(定型サイズ)	○	×	×	×
印字開始位置	△ 注	○	○	○

注：CH情報で定義したチャンネル1が印字開始位置を識別するために使用されます。

## 帳票定義体

各プラットフォームのMeFtと連携することで、COBOLプログラムから帳票定義体に定義した帳票フォーマットにしたがって印刷することができます。帳票定義体はWindows版のFORM/PowerFORMで作成します。

FORM/PowerFORMで作成した帳票定義体は他のプラットフォームでも使用することができます。ただし、プラットフォームごとに利用可能な機能範囲が異なります。プラットフォーム間で共用する目的で帳票定義体を作成する場合には、FORMを使用してSMD形式(システム間流通可能な形式)で作成することをお勧めします。

各プラットフォームで使用する場合のサポート状況や留意事項については、FORMのインストールフォルダに格納される「OS別留意事項(SYSTEM.TXT)」を参照してください。

また、出力するプリンタや印刷オーダーの違いによっても利用可能な機能範囲が異なります。詳細は、各プラットフォームの「MeFt説明書」を参照してください。

## 外字

外字を使用する場合は、各プラットフォームで文字コードの扱いが異なりますので、外字はプラットフォームごとに使用する文字コード系にあわせて登録する必要があります。

## Unicodeサポート

以下に、各プラットフォームのNetCOBOL製品でのUnicodeサポートの状況を示します。

なお、NetCOBOLがサポートするUnicodeの表現形式のうち、UCS-2にはビッグエンディアンとリトルエンディアンの形式があります。以降では、UCS-2のビッグエンディアンをUCS-2(BE)、UCS-2のリトルエンディアンをUCS-2(LE)と表記します。



表A-4 各OS上のNetCOBOL製品でのUnicodeサポートの状況

		Windows	Solaris	Linux32	Linux64
COBOLソースのコード系		シフトJIS	UTF-8	UTF-8	UTF-8
Unicode使用の選択(翻訳時)		オプション	ロケール	ロケール	ロケール
実行時コード系の内部表現	英数字項目	UTF-8	UTF-8	UTF-8	UTF-8
	日本語項目	UCS-2 (LE)	UCS-2 (BE)	UCS-2 (LE)	UCS-2 (LE) または UCS-2 (BE) *1

\*1：翻訳オプションRCSによって指定します。

このサポート状況の違いから次の問題が生じます。

#### Unicode固有文字が使用できない

Windows版のNetCOBOLに固有の問題です。

#### 日本語項目を含む集団項目を大小比較した場合、正しく判定できない

次のプログラム例を元に説明します。

```
WORKING-STORAGE SECTION.
01 GRP.
  02 SMALL PIC N(1) VALUE NC" あ" . *> X" 3042"
01 LARGE PIC N(1) VALUE NC" A" . *> X" FF21"
  :
  IF SMALL < LARGE THEN *> (1)
    DISPLAY "OK!!" .
  ELSE
    DISPLAY "NG!!"
  END-IF.
  IF GRP < LARGE THEN *> (2)
    DISPLAY "OK!!" .
  ELSE
    DISPLAY "NG!!"
  END-IF.
```

このプログラムの実行結果は、各プラットフォーム向けのNetCOBOL製品で次のように異なります。

表A-5 各OS上のNetCOBOL製品の上記プログラムの実行結果

	Windows	Solaris	Linux32	Linux64	
日本語項目の内部表現	UCS-2 (LE)	UCS-2 (BE)	UCS-2 (LE)	UCS-2 (LE)	UCS-2 (BE)
(1)の結果	○	○	○	○	○
(2)の結果	×	○	×	×	○

(1)の結果はすべての製品で一致しますが、日本語項目の内部表現にUCS-2 (BE)を採用しているSolaris版以外の製品では、日本語を含む集団項目と日本語項目の比較結果は期待したとおりに動作しません。

Linux64版のNetCOBOL製品では、この問題に対応するための翻訳オプションが用意されています。

#### 表示ファイル(画面)

宛先DSPの表示ファイル機能を使用して画面入出力を行うためには、連係する製品が必要となります。この連係製品によって接続する画面の種類や機能のサポート状況が異なります。

	Windows	Solaris	Linux32	Linux64
ローカル画面 (MeFt)	○	○	×	×
リモート画面 (MeFt/NET)	○	○	×	×
Web画面 (MeFt/Web)	○	○	○	×

画面入出力ではWindows版のFORMで作成した画面定義体を使用します。FORMで作成した画面定義体は他のプラットフォームでも使用することができます。ただし、プラットフォームごとに利用可能な機能範囲が異なります。各プラットフォームで使用する場合のサポート状況や留意事項については、FORMのインストールフォルダに格納される「OS別留意事項(SYSTEM.TXT)」を参照してください。

また、連係製品によっても利用可能な機能範囲が異なります。詳細は、各製品のマニュアルを参照してください。

### 表示ファイル(印刷)

宛先PRTの表示ファイル機能を使用して帳票出力を行うためには、連係する製品が必要となります。この連係製品によって機能のサポート状況が異なります。

	Windows	Solaris	Linux32	Linux64
サーバ印刷 (MeFt)	○	○	○	○
Web印刷 (MeFt/Web)	○	○	○	×
電子帳票 (MeFt + List WORKS)	○	○	×	×
PDF (MeFt + List Creator EE)	○	○	○	○
帳票管理サーバ (MeFt + List Manager)	○	○	×	×

帳票出力ではWindows版のFORM/PowerFORMで作成した帳票定義体を使用します。FORM/PowerFORMで作成した画面定義体は他のプラットフォームでも使用することができます。ただし、プラットフォームごとに利用可能な機能範囲が異なります。各プラットフォームで使用する場合のサポート状況や留意事項については、FORMのインストールフォルダに格納される「OS別留意事項(SYSTEM.TXT)」を参照してください。

また、連係製品によっても利用可能な機能範囲が異なります。詳細は、各製品のマニュアルを参照してください。

### 表示ファイル(宛先APL/ACM)

宛先APLまたはACMの表示ファイル機能を使用して、同期型(APL)/非同期型(ACM)のプログラム間通信を行うためには、NetCOBOLだけではなく連携するミドルウェア製品が必要となります。このミドルウェア製品の有無により、NetCOBOLの各プラットフォーム向け製品での宛先APLまたはACMの表示ファイル機能のサポート状況が異なります。以下に、そのサポート状況を示します。

**表A-6 各OS上のNetCOBOL製品の宛先APL/ACMの表示ファイルサポートの状況**

	Windows	Solaris	Linux32	Linux64
同期型プログラム間通信 (APL)	○	○	×	×
非同期型プログラム間通信 (ACM)	○	○	×	×

連携するミドルウェア製品の組合せ可能なバージョンおよびサポート状況については、NetCOBOLのバージョンに対応して変わる可能性があります。詳細は、各製品の“ソフトウェア説明書”または“インストールガイド”等を参照してください。

### スクリーン操作

レイアウト指定を含むデータ項目(スクリーン項目)やレイアウト指定付きのACCEPT/DISPLAY文

を使用して、COBOLで画面入出力を行うプログラムを簡単に記述する機能です。Linux版の製品ではサポートされていません。

一方、Windows版とSolaris版では、次のように異なります。

表A-7 スクリーン操作機能のNetCOBOL製品間での相違点

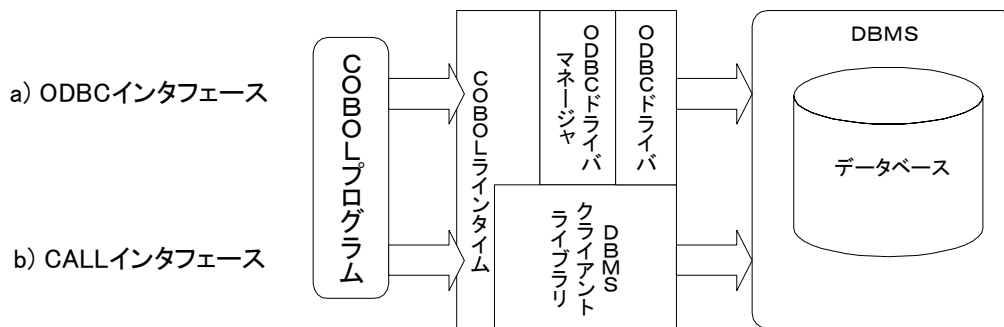
比較ポイント	Windows	Solaris
XPG仕様	○	○
Microfocus互換仕様	○(MF COBOL/2相当 + $\alpha$ )	△(MF COBOL/2相当)
動作コード	SJIS、Unicode(UTF8/UCS2)	EUC
GUI設計ツール	×	○(X Windowシステム必須)

## データベース操作

COBOLプログラムからデータベース操作を行うためには、埋め込みSQL文を使用する必要がありますが、使用可能な埋め込みSQL文の種類や埋め込みSQL文の詳細なふるまいは、次のような条件によって異なります。

1. 接続するデータベース製品
2. 接続方法(ODBCインタフェース/CALLインタフェース)

図A-4 COBOLプログラムからのデータベースへの接続方式の違い



COBOLプログラム～データベースの接続インタフェースの違いは、開発時に埋め込みSQL文を含むCOBOLプログラムを翻訳する際の処理方式の違いとしても現れてきます。

- a. パーザ方式  
NetCOBOL製品組み込みのSQLパーサを使用して埋め込みSQL文を含むプログラムを直接翻訳します。ODBCインタフェースを使用する場合、この方式で処理する必要があります。
- b. プリプロセッサ方式  
接続するデータベース製品によって提供される埋め込みSQLプリプロセッサで埋め込みSQL文を含むプログラムを処理し、プリプロセッサの出力したCOBOLプログラムを翻訳します。CALLインタフェースを使用する場合、この方式で処理する必要があります。

各プラットフォーム用のNetCOBOL製品がサポートするSQL文の処理方式は次のとおりです。

表A-8 各NetCOBOL製品における埋め込みSQL文の処理方式のサポート状況

埋め込みSQL文の処理方式	Windows	Solaris	Linux32	Linux64
パーザ方式	○	×	○	×
プリプロセッサ方式	○	○	○	○

なお、各プラットフォームのNetCOBOL製品が接続をサポートするデータベース製品については、製品のバージョンにより違いがあることが多いため、ここでは述べません。各製品のインストールガイド、ソフトウェア説明書等を参照してください。

## Web連携機能

Web連携機能を使用して、COBOLでインターネット/イントラネット環境の業務システムを構築す

るには、以下の2つの方法があります。

- MeFt/Webの利用  
表示ファイル(画面／帳票)機能を使用するプログラムを再翻訳することなく、Web連携のシステムを構築することができます。
- COBOL Webサブルーチンを利用したアプリケーションの開発  
Webブラウザとのデータ入出力のためのサブルーチンを使用することで、COBOLの知識だけでWebアプリケーションを開発することができます。  
WebサブルーチンにはWebブラウザの標準的なインタフェースを使用するものと、特定のWebサーバのAPIに依存したものが存在します。

**表A-9 各NetCOBOL製品におけるWeb連携機能のサポート状況**

Web連携の機能の実現方法		Windows	Solaris	Linux32	Linux64
MeFt/Web		○	○	○	×
Web サブルーチン	CGI	○	○	○	×
	ISAPI(*1)	○	×	×	×
	SRF(*2)	○	○	×	×
	拡張CGI(*3)	○	×	×	×

\*1： ISAPI(Internet Server Application Programming Interface)。

Windows標準のWebサーバであるIISの拡張インタフェースを使用します。

\*2： SAF(Server Application Functions)。

NSAPI(Netscape Server Application Programming Interface)を使用します。

\*3： Interstageの提供するインタフェースを使用します。

なお、COBOL Webサブルーチンを利用したアプリケーションを開発する場合、サポート状況の違いだけでなく、各システムにおけるファイルやパス名の規則の違いについても注意する必要があります。

## A.3 翻訳・リンク処理に関する互換性

各プラットフォーム向けNetCOBOL製品に含まれるCOBOLコマンドで、ソースプログラムを翻訳・リンクする際、その動作と出力は次のような指定の影響を受けます。

- コマンド名と指定形式
- 翻訳オプション
- COBOL翻訳コマンドのコマンドラインオプション
- 環境変数
- ライブラリ名

ここでは、これらの指定方法および効果が、各プラットフォーム向けNetCOBOL製品に含まれるCOBOLコマンドでどのように違っているかを説明します。

### A.3.1 コマンド名と指定形式

Windows版のNetCOBOL製品とUNIX系システム向けのNetCOBOL製品では、提供されているCOBOLコマンドのコマンド名・機能・指定形式が異なります。

#### Windows版のNetCOBOL製品

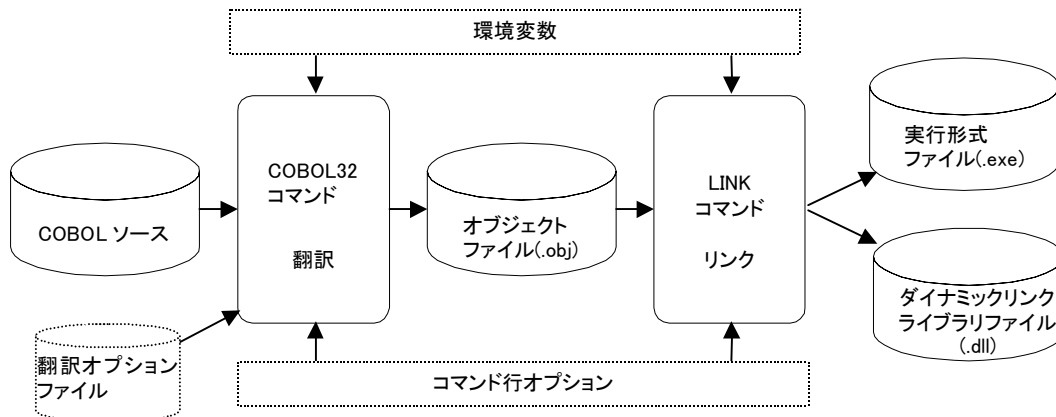
Windows版のNetCOBOL製品に含まれるCOBOLコマンドのコマンドの指定形式は次の通りです。

コマンド名	オペランド
cobol32	[翻訳に関するオプションの並び] ファイル名 …

Windows版のNetCOBOL製品に含まれるCOBOL32コマンドは、COBOLソースプログラムを翻訳し、オブジェクトファイルを作成することのみを行います。

オブジェクトファイルをリンクして、実行形式ファイル(.exe)やダイナミックリンクライブラリ(.dll)を作成するためには、別途linkコマンドを用いる必要があります。

図A-5 cobol32コマンドによるプログラムの構築



#### UNIX系システム向けのNetCOBOL製品

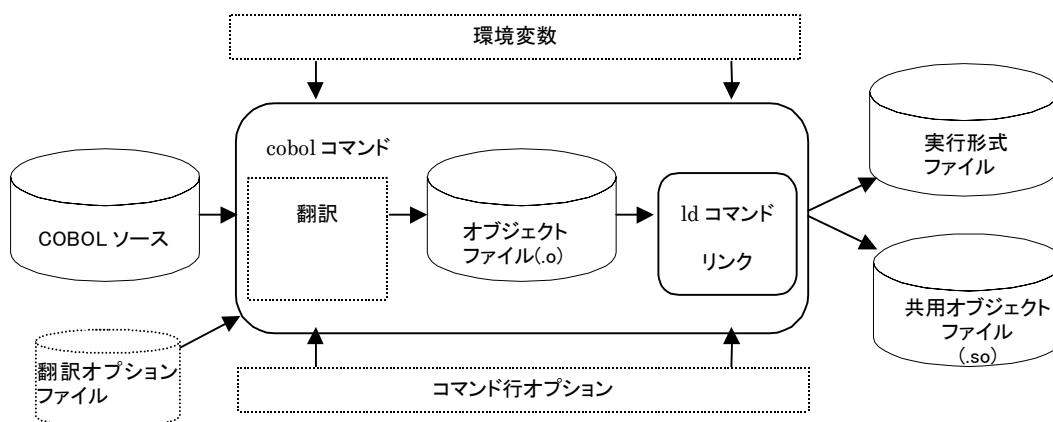
UNIX系システム向けのNetCOBOL製品に含まれるCOBOLコマンドのコマンドの指定形式は次の通りです。

コマンド名	オペランド
cobol	[翻訳に関するオプション、リンクに関するオプションの並び] ファイル名 …

UNIX系システム向けのNetCOBOL製品に含まれるCOBOLコマンドは、COBOLソースプログラムを翻訳し、オブジェクトファイルを作成するだけでなく、オブジェクトファイルをリンクして、実行形

式ファイル(拡張子任意)や共用オブジェクトファイル(.so)を作成することができます(cobolコマンドは、リンクのためにldコマンドを呼び出しています)。

図A-6 cobolコマンドによるプログラムの構築



### A.3.2 翻訳オプション

各プラットフォーム向けNetCOBOL製品が提供するCOBOLコマンド間で仕様が異なる翻訳オプションについて、そのサポート状況を一覧で示した上で、その違いについて説明します。

#### サポート状況

表A-10 各OS上のNetCOBOL製品のサポートする翻訳オプション

名前	機能概要	OS毎のサポート状況			
		Windows	Solaris	Linux 32	Linux 64
AIMLIB	サブスキーマ定義ファイルの格納フォルダを指定する。	○	×	×	×
ALPHAL	英小文字を英大文字と同一視するかどうかを指定する。	○	○	○	○
ASCOMP5	2進項目の解釈方法を指定する。	○	×	○	○
BINARY	2進項目の割り付け領域長をワード単位、バイト単位にするかを指定する。	○	○	○	○
CHECK	CHECK機能を使用するかどうかを指定する。	○	○	○	○
CODECHK	実行時のコード系チェックの指定。	×	○	○	○
CONF	新/旧規格の非互換を指摘する診断メッセージを出力するかどうかを指定する。	○	○	○	○
COPY	ソースプログラムリストへの登録集原文を表示するかどうかを指定する。	○	○	○	○
COUNT	COUNT機能を使用するかどうかを指定する。	○	○	○	○
CREATE	オブジェクト生成のために翻訳するか、リポジトリ生成のために翻訳するかを指定する。	○	○	○	○
CURRENCY	通貨編集用文字を指定する。	○	○	○	○
DLOAD	プログラム構造を指定する。	○	○	○	○

DUPCHAR	Unicode環境でソースを翻訳する際の全角ハイフン文字の扱いを指定する。	×	○	○	○
EQUALS	SORT文での同一キーデータの処理方法を指定する。	○	○	○	○
FILEEXT	ファイル定義体ファイルの拡張子を指定する。	○	×	×	×
FILELIB	ファイル定義体ファイルの格納フォルダを指定する。	○	×	×	×
FLAG	表示する診断メッセージのレベルを指定する。	○	○	○	○
FLAGSW	COBOL文法の言語要素に対する指摘メッセージを出力するかどうかを指定する。	○	○	○	○
FORMEXT	画面帳票定義体ファイルの拡張子を指定する。	○	×	×	×
FORMLIB	画面帳票定義体ファイルの格納フォルダを指定する。	○	×	×	×
GEN	ソースプログラムリストにFCOM及びUWAを表示するかどうかを指定する。	○	×	×	×
INITVALUE	作業場所節でのVALUE句なし項目の扱い。	○	○	○	○
KANA	カナ文字の文字コードの扱いを指定する。	×	○	○	○
LALIGN	連絡節データ宣言の扱いを指定する。	×	○	×	○
LANGVL	ANSI COBOL規格の非互換項目をどの規格に従って解釈するかを指定する。	○	○	○	○
LIB	登録集ファイルの格納フォルダを指定する。	○	×	×	×
LIBEXT	登録集ファイルの拡張子を指定する。	○	×	×	×
LINECOUNT	翻訳リストの1ページあたりの行数を指定する。	○	○	○	○
LINESIZE	翻訳リストの1行あたりの文字数を指定する。	○	○	○	○
LIST	目的プログラムリストの出力を出力するかどうかを指定する。	○	○	○	○
MAIN	主プログラム/副プログラムの違いを指定する。	○	△	△	△
MAP	データマップリスト、プログラム制御情報リストおよびセクションサイズリストを出力するかどうかを指定する。	○	○	×	○
MESSAGE	オプション情報リスト、翻訳単位統計情報リストを出力するかどうかを指定する。	○	○	○	○
MODE	ACCEPT文の動作の指定を指定する。	○	○	○	○
NAME	翻訳単位ごとにオブジェクトファイルを生成するかを指定する。	○	○	○	○
NCW	日本語利用者語の文字集合を指定する。	○	○	○	○
NSPCOMP	日本語空白の比較方法を指定する。	○	○	○	○
NUMBER	行番号としてソースプログラムの一連番号領域の値を使用するかどうかを指定する。	○	○	○	○
OBJECT	目的プログラムを出力するかどうかを指定する。	○	△	△	△
OPTIMIZE	広域最適化を行うかどうかを指定する。	○	○	○	○

PRINT	各種翻訳リストの出力を出力するかどうかとその出力先を指定する。	○	×	×	×
QUOTE/APOST	表意定数QUOTEの扱いを指定する。	○	○	○	○
RCS	実行時コード系を指定する。	○	×	×	△
REP	リポジトリファイルの入出力先フォルダを指定する。	○	×	×	×
REPIN	リポジトリファイルの入力元フォルダを指定する。	○	×	×	×
RSV	使用する予約語セットを指定する。	○	○	○	○
SAI	ソース解析情報ファイルを出力するかどうかを指定する。	○	△	△	△
SDS	符号付き10進項目の符号整形を実施するかどうかを指定する。	○	○	○	○
SHREXT	マルチスレッドプログラムにおける外部属性に関する扱いを指定する。	○	○	○	○
SMSIZE	PowerSORTが使用するメモリ容量を指定する。	○	○	○	○
SOURCE	ソースプログラムリストを出力するかどうかを指定する。	○	○	○	○
SQLGRP	SQLのホスト変数定義の拡張仕様を使用するかどうかを指定する。	○	×	○	×
SRF	正書法の種類を指定する。	○	○	○	○
SSIN	ACCEPT文のデータの入力先を指定する。	○	○	○	○
SSOUT	DISPLAY文のデータの出力先を指定する。	○	○	○	○
STD1	英数字の文字の大小順序の比較方法を指定する。	○	○	○	○
TAB	ソース中のタブ文字の扱いを指定する。	○	○	○	○
TEST	対話型デバグおよび診断機能を使用するかどうかを指定する。	○	○	○	×
THREAD	マルチスレッドプログラムの作成を指定する。	○	○	○	○
TRACE	TRACE機能を使用するかどうかを指定する。	○	○	○	○
TRUNC	桁落とし処理を実施するかどうかを指定する。	○	○	○	○
XREF	相互参照リストを出力するかどうかを指定する。	○	○	○	○
ZWB	符号付き外部10進項目と英数字項目の比較方法を指定する。	○	○	○	○

各プラットフォーム向けNetCOBOL製品に含まれるCOBOLコマンド間で仕様が異なる翻訳オプション（“表A-10 [各OS上のNetCOBOL製品のサポートする翻訳オプション](#)” 中では網かけで表示）は主に次の3つの種類に分類できます。

- Windows版製品に固有の翻訳オプション。
- UNIX系システム向け製品に固有の翻訳オプション。
- 特定の機能に依存する翻訳オプション。

それぞれについて、以降で詳細を説明します。

## Windows版製品に固有の翻訳オプション

Windows版のNetCOBOL製品に固有の翻訳オプションは、更に次の3つに分類されます。



## Windows版固有の機能に関するオプション

### ● MAIN

Windows版のNetCOBOL製品では、作成するプログラムの形態を次のどちらかから選択可能です。

#### — Windowsプログラム：MAIN(WINMAIN)

ACCEPT文、DISPLAY文の入出力先にCOBOLが作成したコンソールウィンドウを、実行時エラーメッセージの出力先にメッセージボックスを使用するプログラム。

#### — コンソールプログラム：MAIN(MAIN)

ACCEPT文、DISPLAY文および実行時エラーメッセージの入出力先としてシステムのコンソール(コマンドプロンプトウィンドウ)を使用するプログラム。

UNIX系OSのNetCOBOL製品によって作成できるプログラムは、常にコンソールプログラムになります。

## OSIV分散開発向けのオプション

Windows版のNetCOBOL製品は、OSIV系システムのCOBOL85向けのプログラムの分散開発をサポートしています。この際に使用するオプションとして、次のものがあります。

### ● AIMLIB

### ● GEN

### ● FLAGSW (GSW/GSS)

## オプションファイル中で指定のみ有効となる翻訳オプション

翻訳オプションの指定を格納したファイルをオプションファイルと呼びます。NetCOBOLのコンパイラは翻訳時にこのオプションファイルを受け取って(-iオプションに続けて指定する)、オプションファイルの指定に従って、翻訳処理を行うことができます。

Windows版のNetCOBOL製品のコンパイラは、翻訳時の資源や出力ファイルに関するオプションを、オプションファイルから受け取ることができますが、UNIX系システム向けのNetCOBOL製品のコンパイラは、環境変数や翻訳コマンドのコマンドラインオプションとして受け取る必要があります。以下に、その対応関係を示します。

表A-11 Windows版のオプションファイル中でのみ有効な翻訳オプションと代替方法

名前	代替方法		備考
	コマンド行オプション	環境変数	
FILEEXT	—	FFD_SUFFIX	
FILELIB	-f フォルダ名	FILELIB	
FORMEXT	—	SMD_SUFFIX	
FORMLIB	-m フォルダ名	FORMLIB	
LIB	-I フォルダ名	COB_COBCOPY	
LIBEXT	—	COBLIB_SUFFIX	
OBJECT	-do フォルダ名	—	オブジェクトファイルの出力の有無の指定はUNIX系システム向け製品でも有効
PRINT	-dp フォルダ名	—	
REP	-dr フォルダ名	—	
REPIN	-R フォルダ名	COB_REPIN	
SAI	-ds フォルダ名	—	ソース解析情報ファイルの出力の有無の指定はUNIX系システム向け製品でも有効

## UNIX系システム向け製品に固有の翻訳オプション

### ● CODECHK

UNIX系システム向けのNetCOBOL製品では、環境変数LANGの設定により以下のことを自動的

に決定します。

- 翻訳時：COBOLソース中で使用されている文字のコード系
- 実行時：COBOLプログラムの使用する文字のコード系

通常のCOBOLプログラムの実行時には、上記の2つの文字コード系が等しいかどうかのチェックが行われ、異なる場合には次のメッセージが出力されて、プログラムを正常に実行することができません。

JMP0029I-U [PID: xxxxxxxx TID:yyyyyyyy] 環境変数(LANG)とプログラムのコード系が一致していません。PGM=プログラム名

翻訳オプションNOCODECHKは、このチェックの実施を制御します。通常の設定ではチェックを行う(CODECHK)です。

チェックを行わない(NOCODECHK)を選択することで、日本語文字のコード系に依存しないプログラムを作成することができるようになります。

#### ● DUPCHAR

Unicodeでは、全角ハイフンと見なすことが可能な文字が2種類あります。COBOLソースの翻訳時、どちらの文字を全角ハイフンと見なして処理をするかを指定します。

次のいずれかの条件に当てはまるプログラムの翻訳時に必要となる場合があります。

- 3バイト項目制御部を指定した画面帳票定義体を取り込んでいる
- COPY文の書き方2および書き方3で日本語利用者語を使用している

EUCまたはシフトJISの全角ハイフンをUnicodeにコード変換した時、システム標準のiconvを使用して変換した場合とInterstage Charset Managerの標準コード変換を使用して変換した場合とで結果が異なります。

システム標準のiconvを使用して変換した場合にはDUPCHAR(STD)を、Interstage Charset Managerの標準コード変換を使用して変換した場合にはDUPCHAR(EXT)を指定してください。

#### ● KANA

実行時の文字コード系としてEUCを使用する場合の文字定数および英字・英数字項目内のカナ文字のコード系を指定します。JISカナ文字(半角カナ文字)は、文字コード系がEUCである場合、2バイトの領域を占める文字として扱われます。このため、文字コード系がEUCである場合、次の記述は正常に翻訳することができません。

01 KANA-DATA PIC X(5) VALUE “アイウエオ”.

このオプションでKANA(JIS8)を指定することにより、COBOLプログラム内でJISカナ文字を文字コード系がEUCである場合でも、1バイトの領域を占める文字として扱わせることができます。

#### ● LALIGN

連絡節に定義されたデータ項目が8バイトの整列境界にあっていることを前提としたオブジェクトの生成を指定します。

SolarisおよびLinux64では、データ項目が整列境界にあっているか/いないかで、データ項目を参照する際の性能が異なるため、このオプションの指定によりプログラムの性能の向上が期待できます。

### 特定の機能に依存する翻訳オプション

#### ● ASCOMP5

NetCOBOLは、USAGE句の指定により2進項目の内部表現形式を変更することができます。

表A-12 USAGE句の指定と2進の内部表現

USAGE句の指定	2進項目の内部表現形式
COMPUTATIONAL (COMP)	規格 2 進
COMPUTATIONAL-4 (COMP-4)	
BINARY	
COMPUTATIONAL-5 (COMP-5)	システム2進

規格2進は、システムのアーキテクチャに依存しないように決められた2進項目の内部表現形式です。

この翻訳オプションは、その指定に従って規格2進として定義された2進項目をシステム2進として扱うためのオプションです。規格2進とシステム2進の形式が異なるWindowsおよびLinux向けのNetCOBOL製品では、このオプションを指定することによってプログラムの性能の向上が期待できます。Solarisでは、規格2進とシステム2進の形式が一致しているため、このオプションは効果を持ちません。

● SQLGRP

COBOLソースプログラム中に埋め込みSQL文を記述する際、ホスト変数の定義方法を拡張するかどうかを指定するものです。COBOLソースプログラム中の埋め込みSQL文をパーザ方式で処理する場合のみ有効になります(“表A-8 [各NetCOBOL製品における埋め込みSQL文の処理方式のサポート状況](#)” 参照)。

### 翻訳オプションのデフォルト値

稀に、すべてのNetCOBOL製品でサポートされているが、そのデフォルト値が異なるという翻訳オプションがあります。以下にその一覧を示します。

表A-13 NetCOBOL製品間でデフォルト値の異なる翻訳オプション

No	オプションの選択形式	製品毎のデフォルト値			
		Windows	Solaris	Linux32	Linux64
1	$\left\{ \begin{array}{l} \text{OPTIMIZE} \\ \text{NOOPTIMIZE} \end{array} \right\}$	NOOPTIMIZE	OPTIMIZE	OPTIMIZE	OPTIMIZE

### A.3.3 コマンドラインオプション

各NetCOBOL製品に含まれるCOBOLコマンドをコマンド形式で実行する場合、そのコマンドラインに指定可能なオプションとオペランドの指定形式について、まずそのサポート状況を一覧で示した上で、その違いについて説明します。

表A-14 各OS上のNetCOBOL製品の翻訳コマンドのコマンドラインオプション

No	名前	機能概要	OS毎のサポート状況			
			Windows	Solaris	Linux32	Linux64
翻訳時のふるまいに関するオプション						
1	-A	サブスキーマ定義ファイルのフォルダの指定	○	×	×	×
2	-c	翻訳だけを行う指定	×	○	○	○
3	-Dc	COUNT機能を使用する指定	○	○	○	○
4	-Dk	CHECK機能を使用する指定	○	○	○	○
5	-Dr	TRACE機能を使用する指定	○	○	○	○
6	-dr	リポジトリファイルの入出力先ディレクトリの指定	○	○	○	○

7	-ds	ソース解析情報ファイルの出力ディレクトリの指定	○	○	○	○
8	-Dt	対話型デバッガを使用する指定	○	○	○	×
9	-dd	デバッグ情報ファイルのディレクトリの指定	○	○	○	×
10	-do	オブジェクトファイルのディレクトリの指定	○	○	○	○
11	-dp	翻訳リストファイルのディレクトリの指定	○	○	○	○
12	-f	ファイル定義体ファイルのディレクトリの指定	○	○	×	×
13	-I	登録集ファイルのディレクトリの指定	○	○	○	○
14	-i	オプションファイルの指定	○	○	○	○
15	-M	主プログラムを翻訳するときの指定	○	○	○	○
16	-m	画面帳票定義体ファイルのディレクトリの指定	○	○	○	○
17	-O	広域最適化の指定	○	△	△	△
18	-P	翻訳リストのファイル名の指定	○	△	△	△
19	-R	リポジトリファイルの入力先ディレクトリの指定	○	○	○	○
20	-WC	翻訳オプションの指定	○	○	○	○
リンク時のふるまいに関するオプション						
21	-dy	結合モードの指定	×	○	○	○
22	-G -share	共用オブジェクトプログラムを生成する指定	×	△	△	△
23	-L	ライブラリサーチパス名を追加する指定	×	○	○	○
24	-l	リンクする副プログラムまたはライブラリの指定	×	○	○	○
25	-Ns	C言語で記述したプログラムから呼び出されるプログラムの指定	×	○	×	×
26	-o	実行可能ファイル/共用ライブラリの出力ファイル名	×	○	○	○
27	-pc	スクリーン操作機能を使うプログラムをリンクする指定	×	○	×	×
28	-pi	C-ISAMを使うプログラムをリンクする指定	×	○	×	×
29	-pm	画面定義体を使うプログラムをリンクする指定	×	○	×	×
30	-Wl	リンクオプションの指定	×	○	○	○
翻訳とリンクの両方に関係するオプション						
31	-Tm	マルチスレッドモデルのプログラムをリンクする指定	×	○	○	○

各プラットフォーム向けNetCOBOL製品に含まれるCOBOLコマンド間で仕様が異なるコマンドラインオプション(“表A-14 [各OS上のNetCOBOL製品の翻訳コマンドのコマンドラインオプション](#)” 中では網かけで表示)は主に次の3つの種類に分類できます。

- 翻訳に関するオプション
- リンクに関するオプション

- 翻訳とリンクの両方に関係するオプション  
それぞれについて、以降で詳細を説明します。

### 翻訳に関するオプションの違い

- -A (サブスキーマ定義ファイルのフォルダの指定)  
Windows版のNetCOBOL製品は、OS/IV系システムのCOBOL85向けのプログラムの分散開発をサポートしています。この際に使用するオプション(AIMLIBと等価)です。
- -f (ファイル定義体のフォルダの指定)  
Linux向けのNetCOBOL製品では、ファイル定義体をサポートしていないため、このコマンドラインオプションは用意されていません。
- -O (広域最適化の指定)  
翻訳オプションOPTIMIZEの指定に相当します。Windows版のNetCOBOLのみ、NOOPTIMIZEがデフォルトであるため、このコマンドラインオプションが用意されています。
- -P (翻訳リストファイル名の指定)  
基本的な使い方 (“-P ファイル名”) は同じですが、翻訳リストファイルを “ソースファイル名.lst” の形式で出力したい場合の指定形式が異なります。

表A-15 -Pオプションの指定形式の違い

製品	指定形式	補足
Windows版	-P	ファイル名省略。ただし、ソースファイル名の直前に指定できない。
Solaris版/Linux32版/Linux64版	-P-	ファイル名の代わりにハイフンを指定。

### リンクに関するオプション

UNIX系プラットフォーム向けのNetCOBOL製品の翻訳コマンドは、内部的にldコマンドを呼び出すことが可能なため、リンクに関する設定を指定することができます。

- -pc (スクリーン操作機能を使うプログラムをリンクする指定)
- -pi (C-ISAMを使うプログラムをリンクする指定)
- -pm (画面定義体を使うプログラムをリンクする指定)
- -Ns (C言語で記述したプログラムから呼び出されるプログラムの指定)  
それぞれの機能を使用する際に必要となる共用ライブラリをリンクするために必要となります。Windows版のNetCOBOLではライブラリの構成の違いから、これらの機能の使用時に必要となるライブラリは常にリンクされるようになっています。
- -dy (結合モードの指定)
- -G/-share (共用オブジェクトプログラムを生成する指定)
- -L (ライブラリサーチパス名を追加する指定)
- -l (リンクする副プログラムまたはライブラリの指定)
- -o (実行可能ファイル/共用ライブラリの出力ファイル名)
- -Wl (リンクオプションの指定)  
これらは基本的にldコマンドに直接渡されます。WindowsシステムのNetCOBOL製品ではCOBOLプロジェクトマネージャから設定します。

### 翻訳とリンクの両方に関係するオプション

- -Tm (マルチスレッドモードの翻訳・リンクの指定)  
COBOLソースの翻訳およびリンク時の次の効果をもたらします。
  - 翻訳時: THREAD (MULTI) オプションの指定に相当します。
  - リンク時: マルチスレッド動作に必要なライブラリをリンクします。
 Windows版のNetCOBOL製品では、翻訳オプションTHREAD (MULTI) 指定時、COBOLプロジェクトマネージャが同等の操作を自動で行います。

## A.3.4 環境変数

各NetCOBOL製品に含まれるCOBOLコマンドは以下の環境変数が設定されている場合、これを参照します。

表A-16 各OS上のNetCOBOL製品の翻訳コマンドの参照する環境変数

No	名前	機能概要	OS毎のサポート状況			
			Windows	Solaris	Linux32	Linux64
翻訳処理に影響を与える環境変数						
1	COB_OPTIONS	翻訳オプションの指定	○	×	×	×
2	COBOLOPTS		×	○	○	○
3	COB_COPYCOPY	登録集ファイルの格納フォルダ/ディレクトリの指定	○	×	×	×
4	COBCOPY		×	○	○	○
5	COB_登録集名	IN/OF 指定の登録集ファイルのフォルダ/ディレクトリ	○	×	×	×
6	登録集名		×	○	○	○
7	COB_COPYNAME	登録集原文の検索条件の指定	×	○	○	○
8	COB_LIBSUFFIX	登録集ファイル名の拡張子の指定	○	○	○	○
9	SMED_SUFFIX	画面帳票定義体ファイル名の拡張子の指定	○	○	○	○
10	FORMLIB	画面帳票定義体ファイルの格納フォルダ/ディレクトリの指定	○	○	○	○
11	FFD_SUFFIX	ファイル定義体ファイル名の拡張子の指定	○	○	×	×
12	FILELIB	ファイル定義体ファイルの格納フォルダ/ディレクトリの指定	○	○	×	×
13	COB_REPIN	リポジトリファイルの入力先フォルダ/ディレクトリの指定	○	○	○	○
14	COB_AIMLIB	サブスキーマ定義ファイルのフォルダ/ディレクトリの指定	○	×	×	×
リンク処理に影響を与える環境変数						
15	LIB	リンク時に必要となるライブラリの検索先/順序を指定する。	○	×	×	×
16	LD_LIBRARY_PATH		×	○	△	△

各プラットフォーム向けNetCOBOL製品に含まれるCOBOLコマンド間で仕様が異なる環境変数(“表A-16 [各OS上のNetCOBOL製品の翻訳コマンドの参照する環境変数](#)” 中では網かけで表示)は主に次の4つの種類に分類できます。

- Windows版とUNIX系システム向け製品で名前が異なる環境変数
- Windows版に固有の環境変数
- UNIX系システム向け製品に固有の環境変数
- リンク処理に影響を与える環境変数

それぞれについて、以降で詳細を説明します。

## Windows版とUNIX系システム向け製品で名前が異なる環境変数

以下の環境変数は、Windows版のNetCOBOL製品とUNIX系システム向けのNetCOBOL製品で同じ効果を持つものですが、名前が異なります。

- COB\_OPTIONS/COBOLOPTS
- COB\_COPYCOPY/COBCOPY
- COB\_登録集名/登録集名

## Windows版に固有の環境変数

- COB\_AIMLIB

Windowsシステム向け製品に固有のOSIVプログラムの分散開発支援機能を使用する際に、サブスキーマ定義ファイルの格納フォルダを指定するために使用します(翻訳オプションAIMLIB/コマンドラインオプション-Aと等価)。OSIVプログラムの分散開発支援機能をサポートしないUNIX系システム向けのNetCOBOL製品では意味を持ちません。

## UNIX系システム向け製品に固有の環境変数

- COB\_COPYNAME

登録集原文、ファイル定義体および画面帳票定義体を格納したファイルのファイル名を検索する方法を指定します。

Windowsシステムと異なり、UNIX系システムではファイル名を構成する文字の英大文字/英小文字は区別されます。ソースプログラム中のCOPY文に記述されたファイル名を、拡張子を含めてすべて英大文字または英小文字として検索を行いたい場合に、この環境変数を使用します。

## リンク処理に影響を与える環境変数

リンク処理に使用されるコマンド(UNIX系ではcobolコマンドから呼び出される)の仕様の違いから、リンク時に必要となるライブラリの検索パスを得るために用いる環境変数名と効果が次のように異なります。

表A-17 リンク処理を行うコマンドと環境変数

製品	コマンド	環境変数	備考
Windows	link.exe	LIB	
Solaris	ld	LD_LIBRARY_PATH	
Linux32/Linux64	ld	なし	LD_LIBRARY_PATHは指定しても無視される

## A.3.5 ライブラリ

NetCOBOL製品を使用して、COBOLプログラムを作成する場合、そのCOBOLプログラムの使用する機能に依存するライブラリがリンク処理時に必要となります。

このライブラリ名およびその構成は、NetCOBOL製品によって異なります。以下にその対応表を示します。なお、UNIX系では一般にシングルスレッドモード時とマルチスレッドモード時で必要となるライブラリが異なります。その場合、シングルスレッドモード時に必要なライブラリを上段にマルチスレッドモード時に必要なライブラリを下段に示します。

表A-18 リンク時に必要なライブラリの製品間の相違

コンポーネント	機能	Windows版	Solaris版	Linux版
C言語 ライブラリ	システム基盤	libc.lib	libc.so	libc.so
		KERNEL32.lib	libdl.so	libdl.so
		USER32.lib	libdl.so	libdl.so
			libthread.so	libpthread.so
NetCOBOL	基本機能	F3BICIMP.lib	libcobol.so, libFJBASE.so	libcobol.so, libFJBASE.so

			librcobol. so, libFJBASE. so	librcobol. so, libFJBASE. so
	簡易アプリ間通信		libisam. a, libcobcicl. so	libisam. a, libcobcicl. so (*1)
			libisam. a, librcobcicl. so	libisam. a, librcobcicl. so (*1)
	C-ISAM		libcobcim. so, libisam. a	—
			libcobcim. so, libisam. a	
	スクリーン機能		libcurses. a, libcobscr. so	—
			libcurses. a, librcobscr. so	
	表示ファイル(画面)機能 (MeFt使用)		libXm. so, libXll. so, libXt. so	—
	Web連携機能(CGI)	F3BICWSR. lib	libcobw3cgi. so	libcobw3cgi. so (*1)
	Web連携機能(SAF)	F3BINSRT. lib	libcobfa. so librcobfa. so	—
	Web連携機能(ISAPI)	F3BISAPI. lib	—	
	ファイルアクセス ルーチン	F3BIFCFA. lib	libcobfa. so librcobfa. so	libcobfa. so librcobfa. so
ファウンデーション クラス	COBOLファイル ユーティリティ関数	F3BIFUTC. lib	—	—
	DBアクセス クラスライブラリ	F1FVDENT. lib	libdbalcob. so	—
	コレクション クラスライブラリ	F3BICCOL. lib	libFJCOL-LINKED -LIST. so, libFJCOL-ALPHAN UMERIC-MAP. so, libFJCOL-IDENTI TY-SET. so	—
	帳票 クラスライブラリ	F3BICFPR. lib	libFJPRT. so, librFJPRT. so	—
	Jアダプタクラス ジェネレータ	F3BIJART. lib	libjart. so	libjart. so

\*1 : Linux64版では提供されていません。



## A.4 実行環境

### A.4.1 環境変数情報

COBOLプログラムは、その実行時に環境変数あるいは初期化ファイルから環境変数情報を取得することで、そのふるまいが変わります。

Windows版のNetCOBOLとUNIX系のプラットフォームのNetCOBOL製品では、実行時に参照する環境変数に次のような違いがあり、完全に互換性を持つものはほとんどありません。

- 環境変数名の形式  
環境変数名は基本となる部分はほとんど共通ですが、Windows版では、しばしば環境変数名は@で始まります。
- 環境変数名の認識  
Windows版では環境変数名に英小文字が含まれている場合があります。これはマニュアル表記上の問題で、実際は英大文字／小文字は区別されていません。これに対して、UNIX系プラットフォームのNetCOBOL製品では、環境変数は必ず大文字で指定しなければなりません。
- 指定可能な値  
各プラットフォーム向けのNetCOBOL製品の機能差から、環境変数に指定可能な値が異なります。特にファイルのパスを指定する場合、パスを構成する文字が異なります。

これらの違いを踏まえた上で、以下に各プラットフォーム向けのNetCOBOL製品で実行時に有効となる環境変数の対応表を示します。

なお、表中の互換性状況の欄の記号は次の意味を持ちます。

- ：同じ指定が有効なもの
- △：名前の形式の違いがあるもの
- ▲：指定可能な値が異なるもの
- ：そのプラットフォームのみで意味を持つ環境変数
- －：そのプラットフォームで対応する環境変数がない

表A-19 実行時に有効となる環境変数情報の製品間の相違

機能概要		環境変数名 (名前が異なる場合、上段はWindows、 下段はUNIX系)	互換性			
			Windows	Solaris	Linux32	Linux64
実行環境に関するもの						
	実行時オプションの指定	@GOPT	△	△	△	▲
		GOPT				
	OSIV系形式の実行時パラ メタの指定	@MGPRM	□	－	－	－
		－				
	実行用の初期化ファイル の指定	@CBR_CBRFILE	▲	○	○	○
		CBR_CBRFILE				
	簡略化した動作状態の出 力	@CBR_CBRINFO	△	○	○	○
		CBR_CBRINFO				
	COBOLアプリケーション起 動中のWindows終了指定	@ExitSessionMSG	□	－	－	－
		－				
副プログラム呼び出しに関するもの						
	エントリ情報ファイルの指 定	@CBR_ENTRYFILE	▲	○	○	○
		CBR_ENTRYFILE				

	共用ライブラリの格納先の検索パス指定		PATH	▲	○	○	○
			LD_LIBRARY_PATH				
ファイル処理に関するもの							
	プログラムで使用するファイルの指定		ファイル識別名	▲	○	○	○
	ファイル排他処理の指定		@AllFileExclusive	□	—	—	—
			—				
	入出力エラーの実行時メッセージの出力		@CBR_FILE_USE_MESSAGE	△	○	○	○
			CBR_FILE_USE_MESSAGE				
	行順ファイルのレコード内後置空白を取り除くまたは有効にする指定		@CBR_TRAILING_BLANK_RECORD	△	○	○	○
			CBR_TRAILING_BLANK_RECORD				
	ファイル入力の先読み処理の指定		—	—	○	○	○
			CBR_INPUT_BUFFERING				
	ファイルクローズ時の即時書き出しの指定		—	—	○	○	○
			CBR_CLOSE_SYNC				
表示ファイルに関するもの							
	表示ファイルで使用する接続製品名	宛先DSP	@CBR_PSFIL_DSP	△	△	—	—
			CBR_PSFIL_DSP				
		宛先PRT	@CBR_PSFIL_PRT	△	△	—	—
			CBR_PSFIL_PRT				
		宛先ACM	@CBR_PSFIL_ACM	▲	▲	—	—
			CBR_PSFIL_ACM				
		宛先APL	@CBR_PSFIL_APL	△	△	—	—
			CBR_PSFIL_APL				
	表示ファイルから使用する情報ファイルおよび接続製品名(ファイルごと)の指定		ファイル識別名	▲	○	○	○
	接続製品が使用する情報ファイルが格納されているパス		MEFTDIR	▲	○	○	○
	クライアントのMeFt/Webコントロールが使用するウィンドウ情報		MEFTWEBDIR	▲	○	○	○
印刷ファイルに関するもの							
	プログラムで使用するプリンタおよび各種パラメタの指定		ファイル識別名	▲	○	○	○
	FCB制御文の指定		FCBxxxx	□	—	—	—
			—				
	FCBモジュールが格納されているフォルダの指定		—	—	○	○	○
			FCBDIR				
	フォームオーバーレイパターンのフォルダの指定		FOVLDIR	▲	○	○	○
	フォームオーバーレイパターンのファイル名の形式の指定		FOVLTYPE	□	—	—	—
			—				
	フォームオーバーレイパターンのファイル名の指定		FOVLNAME	□	—	—	—
			—				
	フォームオーバーレイパターン		OVD_SUFFIX	□	—	—	—

	ンのファイルの拡張子の指定	—				
	I制御レコードによる文書名の指定	@CBR_DocumentName_xxxx —	□	—	—	—
	I制御レコードのとじしろ方向、とじしろ幅および印刷原点位置指定をフォームオーバーレイに対して有効または無効にする指定	@CBR_OverlayPrintOffset —	□	—	—	—
	フォームオーバーレイ印刷方法の指定	@CBR_OverlayPrintSPEC —	□	—	—	—
	ANK文字サイズの指定	@CBR_PrinterANK_Size —	□	—	—	—
	印刷ファイルで使用するフォントテーブルの指定	@CBR_PrintFontTable CBR_PRINTFONTTABLE	▲	○	○	○
	ASSIGN句にPRINTERを指定したファイルに対して有効な印刷情報ファイルの指定	@CBR_PrintInfoFile CBR_PRT_INF	▲	○	○	○
	文字配置座標の計算方法の指定	@CBR_PrintTextPosition —	□	—	—	—
	文字行内配置時の上端/下端合わせの指定	@CBR_TextAlign —	□	—	—	—
	デフォルトFCB名の指定	@DefaultFCB_Name —	□	—	—	—
	印刷ファイルで使用するフォントの指定	@PrinterFontName —	□	—	—	—
	用紙名の指定	@PRN_FormName_xxx —	□	—	—	—
	FORMAT句付き印刷ファイルでFCBの省略値の変更の指定	— CBR_FCB_NAME	—	○	○	○
	FORMAT句なし印刷ファイルでファイル管理記述項のASSIGN句の指定がPRINTERの場合に、lpコマンドに渡すオプションを指定	— CBR_LP_OPTION	—	○	○	○
	Unicode(UTF-8)非サポート機能またはプリンタ使用時のUnicode(UTF-8)データの扱い方を指定	— CBR_PRT_UTF8_CONVERT	—	○	○	○
整列併合機能に関するもの						
	整列併合ファイルの作成場所の指定	BSORT_TMPDIR	▲	○	○	○
小入出力に関するもの						
	入力元に使用するファイルの指定	翻訳オプションSSINに指定した名前	▲	○	○	○
	出力先に使用するファイルの指定	翻訳オプションSSOUTに指定した名前	▲	○	○	○

	コンソールウィンドウの種別の指定	@CBR_CONSOLE	△	○	○	○
		CBR_CONSOLE				
	任意日付の指定	@CBR_JOBDATE	△	○	○	○
		CBR_JOBDATE				
	コンソールウィンドウのバッファ数の指定	@CnslBufLine	□	—	—	—
		—				
	コンソールウィンドウの大きさの指定	@CnslWinSize	□	—	—	—
		—				
	コンソールウィンドウで使用するフォントの指定	@CnslFont	□	—	—	—
		—				
簡易アプリ通信機能に関するもの						
	論理宛先定義ファイルのパス名の指定	@CBR_CIINF	▲	○	○	—
		CBR_CI_INF				
	サーバ定義ファイルの指定	@CBR_CI_SRVINF	□	—	—	—
		—				
	クライアント側のログ採取の指定	—	—	○	○	—
		CBR_CI_CLG				
SQL機能に関するもの						
	ODBC情報ファイルの指定	@ODBC_Inf	▲	—	▲	—
		ODBC_INF				
スクリーン操作機能に関するもの						
	スクリーン画面の表示位置の指定	@CBR_SCREEN_POSITION	□	—	—	—
		—				
	スクリーン操作のキー定義ファイルの指定	@CBR_SCR_KEYDEFFILE	▲	▲	—	—
		CBR_SCR_KEYDEFFILE				
	スクリーン操作の論理画面の大きさの指定	@ScrnSize	△	△	—	—
		LINES、COLUMNS				
	スクリーン操作で使用するフォントの指定	@ScrnFont	□	—	—	—
		—				
ウィンドウ表示に関するもの						
	アイコンリソースのDLL名の指定	@IconDLL	□	—	—	—
		—				
	アイコンリソースの識別名の指定	@IconName	□	—	—	—
		—				
	NetCOBOLのアイコン表示の抑止指定	@ShowIcon	□	—	—	—
		—				
	ウィンドウを閉じるときのメッセージ表示の指定	@WinCloseMsg	□	—	—	—
		—				
オブジェクト指向機能に関するもの						
	クラス情報ファイルの指定	@CBR_ClassInfFile	▲	○	○	○
		CBR_CLASSINFFILE				
	オブジェクトインスタンスの獲得方法の指定	@CBR_InstanceBlock	△	○	○	○
		CBR_INSTANCEBLOCK				
マルチスレッドに関するもの						
	スレッドモードの指定	@CBR_THREAD_MODE	□	—	—	—
		—				
	スレッド同期制御サブルーチンの待ち時間の指定	@CBR_THREAD_TIMEOUT	△	○	○	○
		CBR_THREAD_TIMEOUT				

	Symfaware連携のマルチスレッドプログラムを動作可能にする指定	@CBR_SYMFWARE_THREAD	△	○	○	○
		CBR_SYMFWARE_THREAD				
COBOLのデバッグ機能に関するもの						
	COUNT情報ファイルのパス名の指定	SYSCOUNT	▲	○	○	○
	TRACE情報ファイルのパス名の指定	@CBR_TRACE_FILE	▲	○	○	○
		CBR_TRACE_FILE				
デバッグプログラムからのデバッガ起動に関するもの						
	プログラムからデバッガまたは診断機能を起動する指定	@CBR_ATTACH_TOOL	▲	○	○	○
		CBR_ATTACH_TOOL				
	異常終了時にデバッガまたは診断機能を使って調査を行う指定	@CBR_JUSTINTIME_DEBUG	□	－	－	－
		－				
	メモリチェック機能を使って検査を行う指定	@CBR_MEMORY_CHECK	□	－	－	－
		－				
コード系に関するもの						
	実行時コードチェックの有無を指定	－	－	○	○	○
		CBR_CODE_CHECK				
	COBOLプログラムの使用する実行時コード系を指定する	－	－	○	○	○
		LANG				
	COBOLプログラムの使用する実行時コード系を指定する	－	－	○	○	○
		LC_ALL				
組込み関数に関するもの						
	NATIONAL関数の変換モードの指定	@CBR_FUNCTION_NATIONAL	△	○	○	○
		CBR_FUNCTION_NATIONAL				
実行時メッセージ出力に関するもの						
	実行時メッセージの出力先の指定	@CBR_MESSAGE	□	－	－	－
		－				
	実行時メッセージの重大度指定	@CBR_MESS_LEVEL_CONSOLE	△	○	－	○
		CBR_MESS_LEVEL_CONSOLE				
	実行時メッセージの重大度指定	@CBR_MESS_LEVEL_EVENTLOG	□	－	－	－
	実行時メッセージの重大度指定	CBR_MESS_LEVEL_SYSLOG	－	□	－	□
	SYSERR出力へのプロセスID、スレッドIDの出力の有無を指定する	@CBR_SYSERR_EXTEND	△	○	○	○
		CBR_SYSERR_EXTEND				
	メッセージを出力するファイルの指定	@MessOutFile	▲	○	○	○
		CBR_MESSOUTFILE				
	実行時メッセージの抑止指定	@NoMessage	□	－	－	－
		－				



## 付録B 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。

### B.1 文字コードの概要

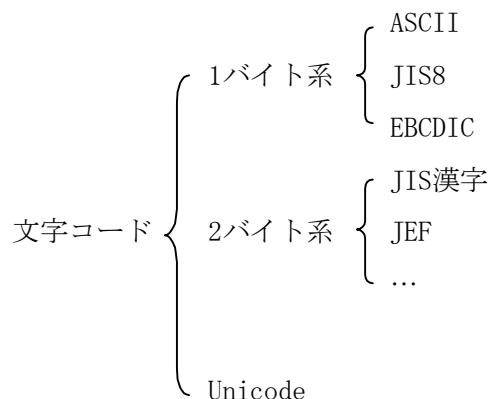
文字コードにはいくつかの方式があり、かつ、分類の方法にも幾つかの種類があります。そのため、実際には単純にあるコード系と別のコード系を比較することはできません。しかし、ここでは単純化のために次の分類に分けてコード系を説明します。

- 文字を表現するバイト数の違い
- 文字種の混在方式

#### B.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B-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(“B. 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“B. 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。

## B. 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。

- シフトJIS(SJIS)  
PCで広く利用されているコード系です。英数字・カナ文字(JIS8)、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。

● EBCDIC-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B-2 各コード系における使用可能な文字の比較

<シフトJIS>		<JEF>	<EUC>	<Unicode>
計	約 7880文字	約 14102文字	約 15393文字	約 18201文字
JIS 第1水準 第2水準 (約6000字)		JIS 第1水準 第2水準 (約6000字)	JIS 第1水準 第2水準 (約6000字)	JIS 第1水準 第2水準 (約6000字)
外字 ( 1880字)		JEF 拡張文字 (約5000字)	拡張文字 (約6231字)	JIS 補助漢字 ( 5801字)
		外字 ( 3102字)	外字 ( 3162字)	外字 ( 6400字)

### B.1.3 Unicode

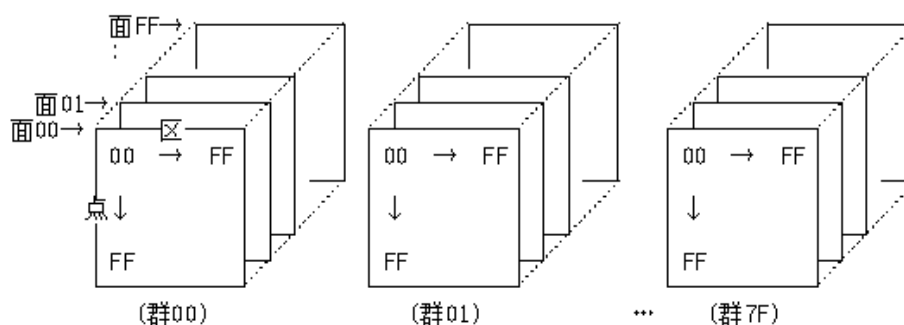
ここまで、説明してきた文字コード系は特定の国や地域での使用を前提として設計されたものです。これに対して、Unicodeは地域や国を限定せず、はじめから世界中の文字を1つの体系で表現できるように設計されました。ただ、世界中で共通に使用可能な文字コード系(ユニバーサル・コードセット)を作成しようとする試みには歴史的な紆余曲折や技術的な困難があり、これがUnicodeを分かりづらいものとしています。

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。

最上位オクテット		最下位オクテット	
群オクテット	面オクテット	区オクテット	点オクテット
X <sup>n</sup> 00 <sup>n</sup> ~X <sup>n</sup> 7F <sup>n</sup>	X <sup>n</sup> 00 <sup>n</sup> ~X <sup>n</sup> FF <sup>n</sup>	X <sup>n</sup> 00 <sup>n</sup> ~X <sup>n</sup> FF <sup>n</sup>	X <sup>n</sup> 00 <sup>n</sup> ~X <sup>n</sup> FF <sup>n</sup>

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ー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)。

00	FF
00	A(Alphabetic) Zone アルファベット や仮名、記号など
4D	
4E	I(Ideographic) Zone CJK 統合漢字 (中国語、日本語、韓国語、台湾語)
9F	
A0	O(Other) Zone 将来のための予約域
DF	
E0	R(Restricted Use) Zone 私用、合成、代替、互換用文字
FF	

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。Windows NT(R) システムの内部コードには、UCS-2のリトルエンディアンが採用されています。

### 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B-2 [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B-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

あ	あ	い	い	…	か	が	き	ぎ	…	は	ば	ぱ	ひ	び	び	…
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

## B.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“表B-2 [富士通のCOBOL製品のサポートするコード系](#)”に示します。

表B-2 富士通の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	OS/IV/MSP	EBCDIC/JEF	COBOL85
	OS/IV/XSP		
UNIX系	Solaris	EUC	COBOL97
		SJIS	NetCOBOL
		Unicode	
	Linux	EUC	NetCOBOL
Windows	Windows(R) 98	SJIS	COBOL97
	Windows(R) Me		NetCOBOL
	WindowsNT (R)	SJIS	
	Windows(R) 2000	Unicode	
	Windows(R) XP	EBCDIC/JEF	JEFオプション
	Windows Server(TM) 2003		



### 注意

COBOLプログラムで使用するコード系を選択する方法は、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	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。

## B.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つに大きく分かります。

- 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。

### B.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。

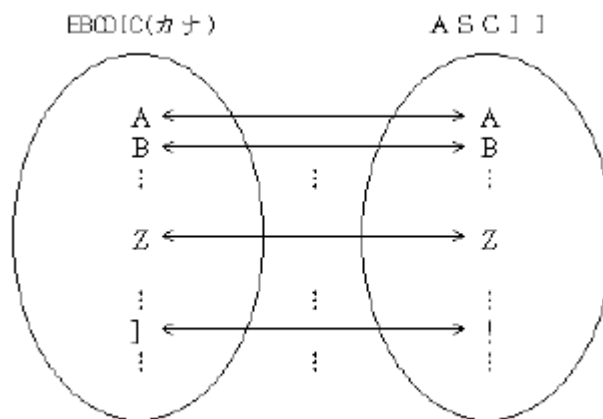
このコード交換の方法は大きく次の2つに分けられます。

- 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。

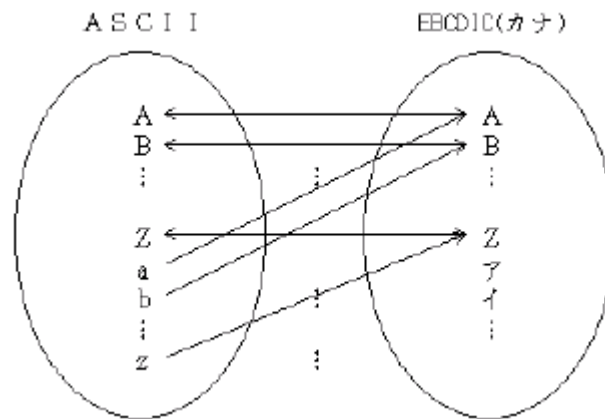
- 代替文字による変換  
変換元と異なる文字を使用せざるを得ないが、変換の前後で1対1の対応を維持できる場合です。  
例えばEBCDICからASCIIへの変換をした場合、“!”が“]”に変換されますが、逆方向の変換をすることによって、元に戻すことができます。

図B-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B-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。

### EBCDIC→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

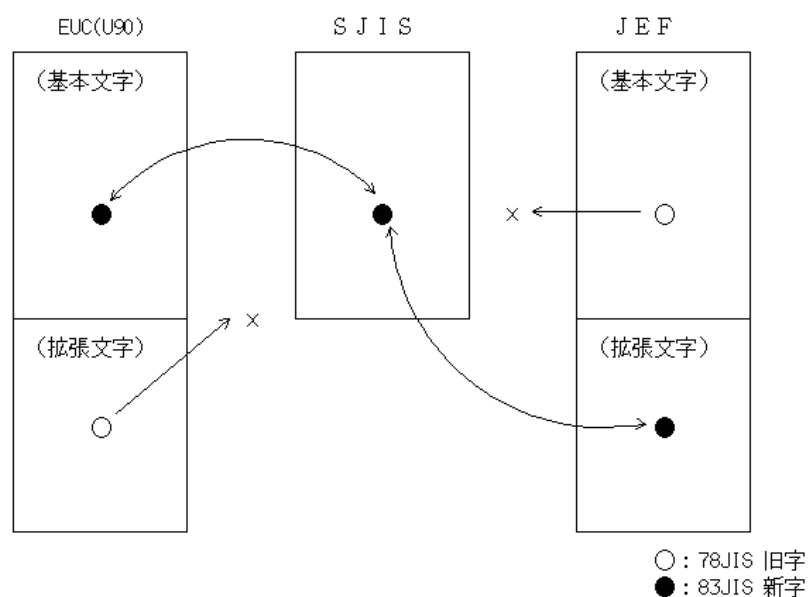
プログラムの記述やデータに上記の表の網かけ部分の文字が含まれていた場合、表示・印刷の結果が以前と異なります。

### 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“鷗”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、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。

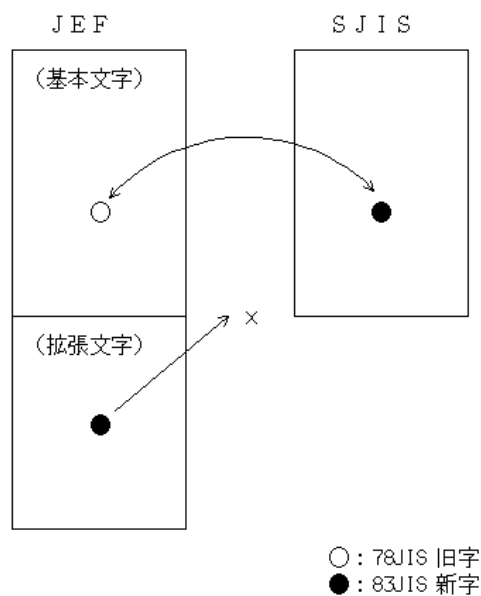


変換前のソースプログラム、データファイル等がEUC(U90)の場合、78JISの旧字体の文字は拡張文字セットに含まれるものであるため、このような変化が行われても問題となることはありません。

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。

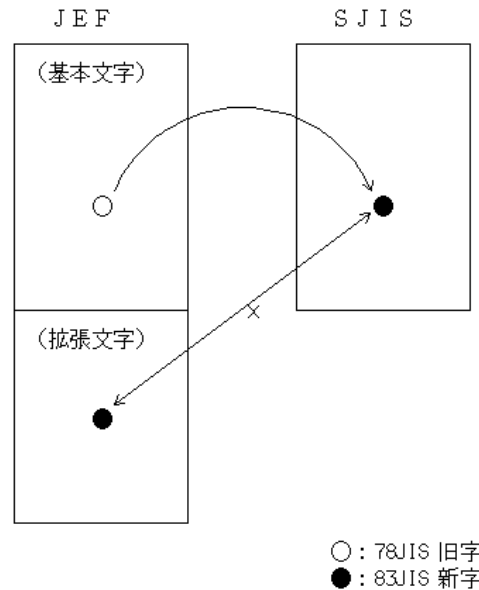


この方法をとる場合、PC上で表示される文字の字形は変換前と異なるものとなってしまいますが、OSIVアプリケーションの分散開発等の場合は、この方法がおすすめです。



● 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う。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み込みの原因となります。

### B.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、意識する必要はありません。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B-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 for WindowsではELSE側のDISPLAY 文が動作します。



## 付録C V7.2における分散開発時の操作性の違い

本章では、NetCOBOL for Windows V7.2を使用して分散開発を行う場合の、NetCOBOL for Windows V8.0との操作性の違いについて説明します。

### C.1 環境設定

#### C.1.1 サーバ連携情報

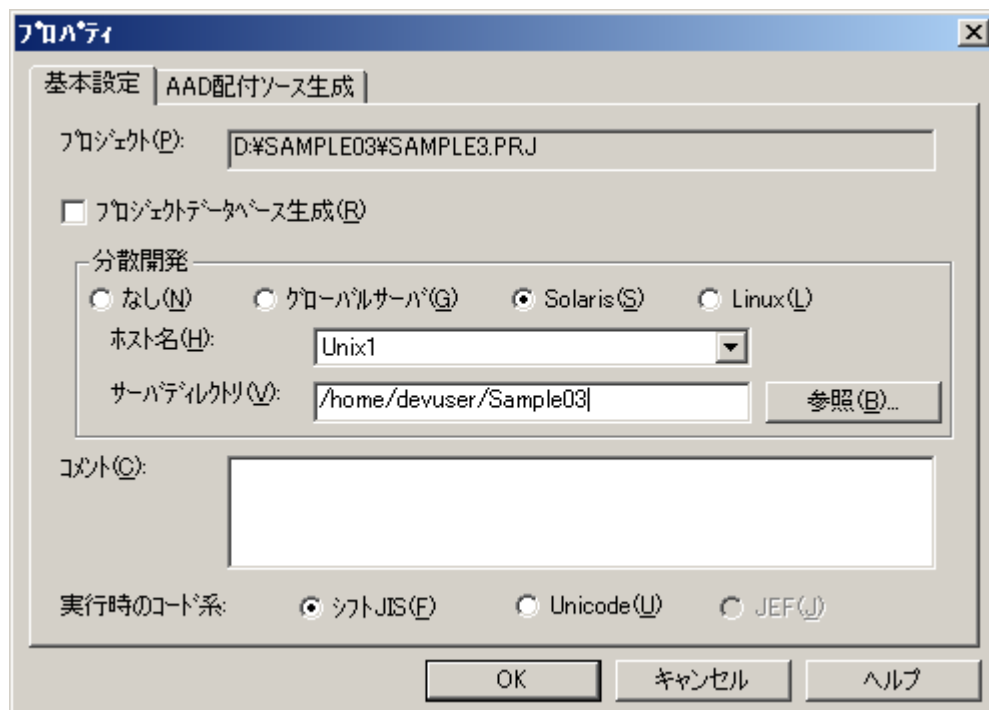
- UNIXサーバとの接続結果の確認機能がサポートされていません。  
サーバ連携情報の設定時に接続確認を行うことはできません。  
正しい値を指定してください。指定値が誤っていた場合、サーバ連携を行う機能の実行時にエラーが発生します。

### C.2 Windowsクライアントでの作業

#### C.2.1 プロパティ

- 「分散開発」ページはありません。  
分散開発の設定は、「基本設定」ページの「分散開発」グループボックスで行います。

図C-1 基本設定ページ



- 分散開発のターゲットとして、Linux(64ビット)を指定することはできません。  
Linux(64ビット)で動作するアプリケーションの開発には、V8.0を使用してください。
- 「分散開発」ページの「ファイル編集終了時のサーバへの送信」機能はサポートされていません。  
プロジェクトのファイルを編集した場合には、「送信」機能を使用して、UNIX系システム

へ転送を行う必要があります。

## C.2.2 翻訳オプション

- [プロジェクト] - [オプション] - [翻訳オプション] メニューで以下の翻訳オプションが設定できません。

— MAP

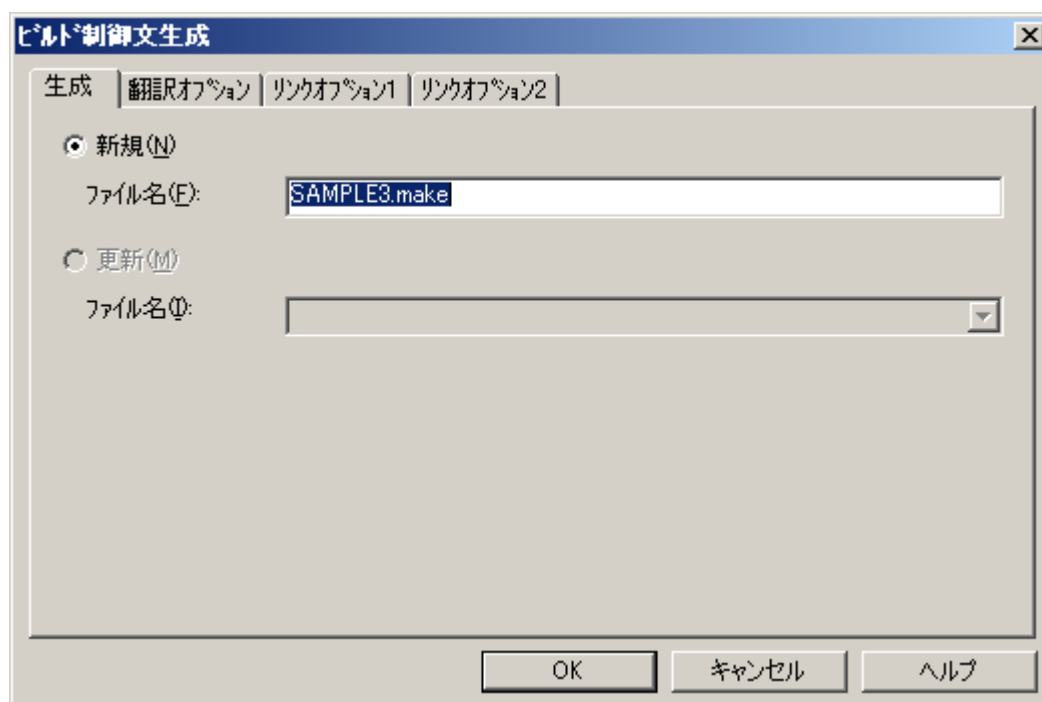
その為、ビルド制御文への出力も行われないので、必要な場合には生成されたビルド制御文を直接編集し、追加してください。

## C.3 UNIXサーバ側での作業

### C.3.1 ビルド制御文生成

- Windows翻訳オプションに“NOOBJECT”が指定されている場合、ビルド制御文の生成は行えません。
- ビルド制御文生成のメイクファイル名指定とオプション設定は同一画面となります。メイクファイル名の指定は[生成]ページで行ってください。必須入力項目はV8.0と同様にメイクファイル名の指定のみとなります。  
翻訳オプションおよびリンクオプションの追加・更新を行う場合は、それぞれのページで指定後、[OK] ボタンをクリックして生成を実行してください。

図C-2 生成ページ



- 未送信のファイルを翻訳対象ファイルとする機能はサポートされていません。  
ビルド制御文生成画面に[送信済のファイルだけをビルドの対象にする]チェックボックスは存在しません。  
送信済のファイルのみが翻訳対象ファイルとなるため、ビルド制御文の生成前に翻訳対象のファイルを“送信”機能を使用して、UNIX系システム上に転送する必要があります。
- ビルド制御文生成後に、UNIXサーバへ自動転送する機能はサポートされていません。  
ビルド制御文生成画面に[生成したmakefileをサーバへ送信する]チェックボックスは存

在しません。

生成したビルド制御文は、“送信”機能を使用してUNIXサーバへ転送してください。

- 翻訳オプション

- DUPCHAR

UNIX系システム固有の翻訳オプション“DUPCHAR”を設定することはできません。

“DUPCHAR”を指定する場合は、生成されたビルド制御文を直接編集し、追加してください。

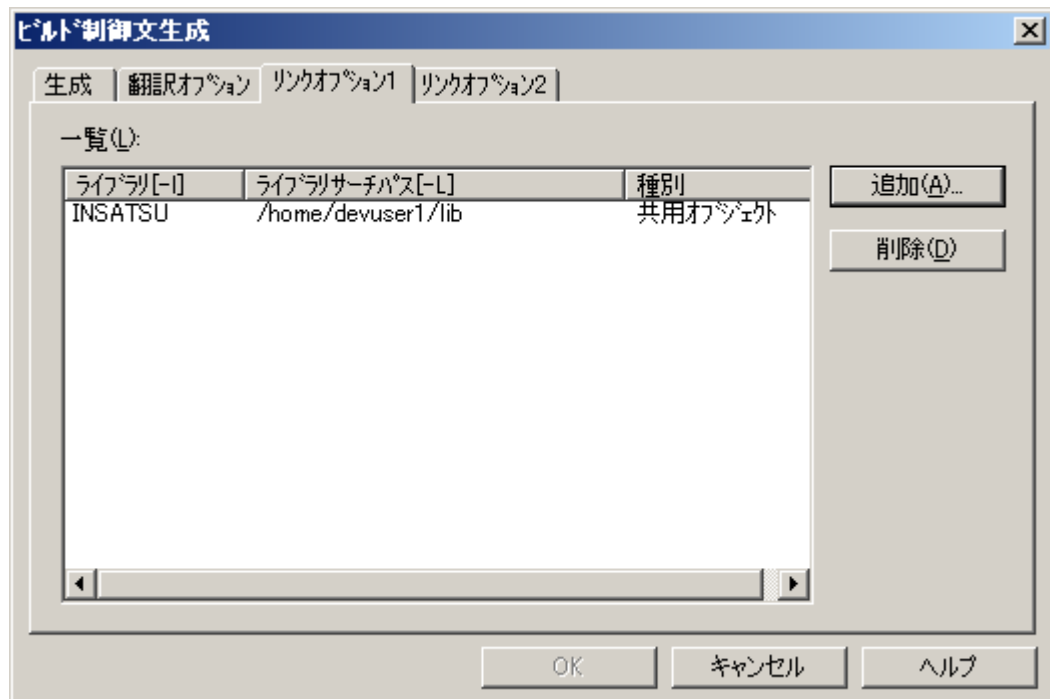
- リンクオプション

- インポートライブラリ

ビルド制御文生成の〔リンクオプション1〕ページで、インポートライブラリ名の初期表示は行われません。必要なインポートライブラリについては作成者が意識して追加する必要があります。

V7.2の〔リンクオプション1〕ページでは、プロジェクトマネージャに設定されたインポートライブラリ名を指定する参照ライブラリが無い為、以下の画面となります。指定内容は、V8.0と同様です。

図C-3 リンクオプション1ページ



- NetCOBOL製品が提供しているライブラリ

“config/mkinc/LIBLIST”を使用した、ビルド制御文への自動展開は行われません。

NetCOBOL製品が提供しているライブラリを使用する場合は、ビルド制御文生成の〔リンクオプション1〕ページで明示的にUNIX系システム上のライブラリファイル名を追加する必要があります。

- 生成されるビルド制御文の内容

- Windowsの〔翻訳オプション〕ダイアログで設定した〔その他の翻訳オプション〕はビルド制御文への出力が行われません。

生成されたビルド制御文を直接編集し、必要なオプションを追加してください。

- Windowsの〔翻訳オプション〕ダイアログで設定した以下のオプションはビルド制御文への出力が行われません。

必要に応じて、ターゲットとなるUNIXサーバの環境変数として設定してください。

- 登録集名

- LIBEXT

環境変数COB\_LIBSUFFIXとして設定する。

- FORMEXT  
環境変数SMED\_SUFFIXとして設定する。
- FILEEXT  
環境変数FFD\_SUFFIXとして設定する。
- マクロ定義  
V8.0では、プロジェクトマネージャに設定されている情報に従うファイル名のパス部分については、マクロ名で記述することにより、ビルド環境を他のディレクトリへ移動した場合にマクロの定義部分を変更するのみで使用できるようになっています。  
V7.0では、本対応は行われていないため、ファイル名はフルパス形式で記述されています。そのため、ビルド環境を他のディレクトリへ移動した場合は、ビルド制御文に記述されているパス名部分をすべて編集する必要があります。
- V8.0でマクロ名出力されている情報は以下のものです。
  - SERVERDIR  
サーバディレクトリを示す。
  - OBJDIR  
オブジェクトファイル格納先を指定するコマンドオプション-doの値
  - SVDDIR  
デバッグ情報ファイル格納先を指定するコマンドオプション-ddの値
  - REPDIR  
リポジトリファイル入出力先を指定するコマンドオプション-drの値
- オブジェクトファイル  
V8.0では、プロジェクト外で生成したオブジェクトファイルもリンク対象のオブジェクトファイルとして出力されます。(オブジェクトフォルダに登録しているオブジェクトファイル)  
しかし、V7.2ではプロジェクト外で生成したオブジェクトファイルはリンク対象のオブジェクトファイルにはなりません。必要な場合は、生成されたビルド制御文を直接編集して追加してください。