

NetCOBOL V10.5



使用手引書

Windows

B1WD-1251-05Z0(00)
2012年12月

まえがき

このたびは、NetCOBOLをお買い上げいただき、誠にありがとうございます。NetCOBOLは、Windowsシステムの32ビットモードで動作し、32ビットアプリケーションの開発環境を提供するCOBOL開発システムです。

NetCOBOLシリーズについて

NetCOBOLシリーズの最新情報については、富士通のサイトをご覧ください。

<http://software.fujitsu.com/jp/cobol/>

登録商標について

- Microsoft、Windows、Windows Server、Windows Vista、Visual C++、Visual Basic、ActiveXは、米国Microsoft Corporationの米国およびその他の国における登録商標または商標です。
- 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。
- OracleとJavaは、Oracle Corporationおよびその子会社、関連会社の米国およびその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。
- HP、HP-UXは、米国Hewlett-Packard Companyの商標です。
- Pervasive、Pervasive Software、Pervasive SQL、Btrieve、Pervasive Softwareロゴは、米国Pervasive Software, Inc.の米国および他の国における登録商標または商標です。
- Micro Focus、Micro Focus COBOLおよびMicro Focus COBOL/2は、Micro Focus International Limitedの登録商標または商標です。
- Linuxは、Linus Torvalds氏の米国およびその他の国における登録商標または商標です。
- Microsoft Corporationのガイドラインに従って画面写真を使用しています。
- その他の会社名または製品名は、それぞれ各社の登録商標または商標です。

製品の呼び名について

本書では、各製品を以下のように略記しています。あらかじめご了承ください。

正式名称	略称
Microsoft(R) Windows(R) XP Home Edition Operating System Microsoft(R) Windows(R) XP Professional Operating System	Windows XP
Windows Vista(R) Home Basic Windows Vista(R) Home Premium Windows Vista(R) Business Windows Vista(R) Enterprise Windows Vista(R) Ultimate	Windows Vista
Microsoft(R) Windows Server(R) 2003, Standard Edition Microsoft(R) Windows Server(R) 2003, Enterprise Edition Microsoft(R) Windows Server(R) 2003 R2, Standard Edition Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition	Windows Server 2003
Microsoft(R) Windows Server(R) 2003, Standard x64 Edition Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition	Windows Server 2003 または Windows Server 2003(x64)

正式名称	略称
Microsoft(R) Windows Server(R) 2003, Enterprise Edition for Itanium-based Systems Microsoft(R) Windows Server(R) 2003, Datacenter Edition for Itanium-based Systems	Windows Server 2003(Itanium)
Microsoft(R) Windows Server(R) 2008 Foundation Microsoft(R) Windows Server(R) 2008 Standard Microsoft(R) Windows Server(R) 2008 Standard without Hyper-V(TM) Microsoft(R) Windows Server(R) 2008 Enterprise Microsoft(R) Windows Server(R) 2008 Enterprise without Hyper-V(TM) Microsoft(R) Windows Server(R) 2008 Datacenter Microsoft(R) Windows Server(R) 2008 Datacenter without Hyper-V(TM) Microsoft(R) Windows Server(R) 2008 R2 Foundation Microsoft(R) Windows Server(R) 2008 R2 Standard Microsoft(R) Windows Server(R) 2008 R2 Enterprise Microsoft(R) Windows Server(R) 2008 R2 Datacenter	Windows Server 2008 または Windows Server 2008(x64)
Microsoft(R) Windows Server(R) 2008 for Itanium-Based Systems	Windows Server 2008(Itanium)
Windows(R) 7 Home Premium Windows(R) 7 Professional Windows(R) 7 Enterprise Windows(R) 7 Ultimate	Windows 7 または Windows 7(x64)
Windows(R) 8 Windows(R) 8 Pro Windows(R) 8 Enterprise	Windows 8 または Windows 8(x64)
Microsoft(R) Windows Server(R) 2012 Datacenter Microsoft(R) Windows Server(R) 2012 Standard Microsoft(R) Windows Server(R) 2012 Essentials Microsoft(R) Windows Server(R) 2012 Foundation	Windows Server 2012 または Windows Server 2012 (WOW64)
Microsoft(R) Windows(R) operating system Version 3.1	Windows 3.1
Microsoft(R) Visual C++(R) development system	Visual C++
Microsoft(R) Visual Basic(R) programming system	Visual Basic
Oracle Solaris	Solaris

・ 以下をすべて指す場合は、「32ビットWindows」または「Windows」と表記します。

- Windows XP
- Windows Vista
- Windows 7
- Windows 8
- Windows Server 2003
- Windows Server 2008
- Windows Server 2012 (WOW64)

- ・ 以下をすべて指す場合は、「Windows(Itanium)」と表記します。
 - － Windows Server 2003(Itanium)
 - － Windows Server 2008(Itanium)
- ・ 以下をすべて指す場合は、「Windows(x64)」と表記します。
 - － Windows 7(x64)
 - － Windows 8(x64)
 - － Windows Server 2003(x64)
 - － Windows Server 2008(x64)
 - － Windows Server 2012

本書の目的

本書は、NetCOBOLを利用したCOBOLプログラムの作成、そのプログラムの実行およびデバッグの方法について説明しています。
また、NetCOBOLを使用したオブジェクト指向プログラミング機能についても説明しています。
COBOLの文法規則については、“COBOL文法書”をお読みください。

本書の対象読者

本書は、NetCOBOLを利用してCOBOLプログラムを開発される方を対象としています。

前提知識

本書を読むにあたって、以下の知識が必要です。

- ・ COBOLの文法に関する基本的な知識
- ・ Windows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 NetCOBOLの概要

NetCOBOLの機能および動作環境について説明しています。

第2章 プログラムを作成・編集する

COBOLプログラムの書き方について説明しています。

第3章 プログラムを翻訳する

COBOLプログラムを翻訳する方法について説明しています。

第4章 プログラムをリンクする

COBOLプログラムをリンクする方法について説明しています。

第5章 プログラムを実行する

翻訳・リンクしたCOBOLプログラムを実行する方法について説明しています。

第6章 プロジェクト管理機能

COBOLアプリケーション開発および保守に有効なプロジェクト管理機能について説明しています。

第7章 ファイルの処理

ファイルを使用する方法について説明しています。

第8章 印刷処理

データや帳票を印刷する方法について説明しています。

第9章 画面を使った入出力

画面を使ってデータのやりとりを行う方法について説明しています。

第10章 サブプログラムを呼び出す～プログラム間連絡機能～

プログラムからプログラムを呼び出す方法について説明しています。

第11章 ACCEPT文およびDISPLAY文の使い方

ACCEPT文およびDISPLAY文を使った機能について説明しています。

第12章 SORT文およびMERGE文の使い方～整列併合機能～

SORT文およびMERGE文を使って整列併合(ソート・マージ)を行う方法について説明しています。

第13章 CSV形式データの操作

CSV形式データを操作する方法について説明しています。

第14章 システムプログラムを記述するための機能～SD機能～

システムプログラムを記述する場合に有効となる機能について説明しています。

第15章 オブジェクト指向プログラミングとは

オブジェクト指向プログラミングの概念について説明しています。

第16章 オブジェクト指向プログラミング機能～基本的な使い方～

オブジェクト指向プログラミング機能の基本的な使い方について説明しています。

第17章 オブジェクト指向プログラミングの開発

オブジェクト指向プログラミングの開発方法について説明しています。

第18章 オブジェクト指向プログラミング機能～さらに進んだ使い方～

オブジェクト指向プログラミング機能のさらに進んだ使い方について説明しています。

第19章 プログラムのデバッグ

NetCOBOLのデバッグ機能について説明しています。

第20章 対話型デバッガの使い方

NetCOBOLの対話型デバッガの使い方について説明しています。

第21章 通信機能

表示ファイルを使用した通信機能および簡易アプリ間通信機能の使い方について説明しています。

第22章 リモートデータベースアクセス

リモートデータベースアクセスについて説明しています。

第23章 サーバ・タイプのアプリケーション

Web環境で動作可能なCOBOLアプリケーションの作成方法とバックグラウンドで動作させる際のCOBOLのGUIの抑止方法について説明しています。

なお、Web機能は、サーバ向け運用環境製品固有の機能です。

第24章 マルチスレッド

マルチスレッドプログラムについて説明しています。

なお、マルチスレッド機能は、サーバ向け運用環境製品固有の機能です。

第25章 Unicode

Unicodeで動作するCOBOLアプリケーションの作成方法について説明しています。

第26章 COM機能

COBOLによるCOM連携機能の実装方法について説明しています。

なお、DCOM機能を利用したCOMサーバ機能は、サーバ向け運用環境製品固有の機能です。

第27章 CORBAアプリケーション

プロジェクトマネージャを使用して、CORBAのサーバクライアントアプリケーションを作成する方法について説明しています。

付録A 翻訳オプション

COBOLコンパイラに与える翻訳オプションについて説明しています。

付録B 入出力状態一覧

入出力文を実行したときに返却される入出力状態を示す値およびその意味について説明しています。

付録C 広域最適化

COBOLコンパイラが翻訳時に行う最適化の内容について説明しています。

付録D 組込み関数の使用

NetCOBOLで利用できる組込み関数について説明しています。

付録E 特殊な定数の書き方

システム固有の各種定数の書き方について説明しています。

付録F OSIV系システムとの機能比較

OSIV系システムのCOBOL85と本システムのNetCOBOLの機能比較について説明しています。

付録G COBOLが提供するサブルーチン

COBOLが提供するサブルーチンについて説明しています。

付録H オブジェクト指向と従来機能の組合せ

オブジェクト指向と従来機能の組合せについて説明しています。

付録I その他のコマンドの形式

COBOLで提供している翻訳コマンド、リンクコマンド以外のその他のコマンドの形式について説明しています。

付録J 移行上の注意事項

Windows 3.1の既存COBOL85アプリケーションを32ビットWindowsのNetCOBOLに移植するための作業について説明しています。

付録K COBOL G移行支援オプション

SX/GシリーズおよびKシリーズのCOBOL G資産を活用するための移行支援オプションについて説明しています。

付録L XMLの利用

XMLの概要とCOBOLアプリケーションからのXMLデータの利用方法および特徴について説明しています。

付録M 文字コード系

文字コード系について概要を説明しています。

付録N セキュリティ

セキュリティについて概要を説明しています。

付録O COBOLプログラムの作成技法

効率よいCOBOLプログラムの作成技法について説明しています。

本書の位置付け

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

マニュアル名称 (注1)	製品名	使用目的
PowerCOBOL ユーザーズガイド PowerCOBOL リファレンス	PowerCOBOL (コンポーネント) (注3)	GUI画面を持つCOBOLアプリケーションの設計および実行
FORM 説明書 FORMヘルプ (注2) PowerFORMヘルプ (注2)	FORM FORMオーバーレイオプション(オプション製品)	画面帳票定義体およびフォームオーバーレイパターンの作成

マニュアル名称 (注1)	製品名	使用目的
MeFt リリース情報 MeFt ユーザーズガイド MeFt メッセージ集	MeFt	画面帳票定義体を使用したプログラムの実行
MeFt/NET-SV ヘルプ(注2)	MeFt/NET-SV	画面帳票定義体を使用したプログラムのリモート環境での実行
MeFt/Web 説明書	MeFt/Web (コンポーネント) (注3)	画面帳票定義体を使用したプログラムのWeb環境でのリモート実行
Jアダプタクラスジェネレータ 使用手引書	Jアダプタクラスジェネレータ(コンポーネント) (注3)	COBOLプログラムからJavaのクラスを利用するためのアダプタクラスの生成
NetCOBOL Studio使用手引書	32ビットWindowsで動作するNetCOBOLシリーズ製品	NetCOBOL Studioを使用したCOBOLアプリケーションの開発
PowerSORT リリース情報 PowerSORT ユーザーズガイド PowerSORT Server リリース情報 PowerSORT Server ユーザーズガイド	PowerSORT PowerSORT Server	PowerSORTを使用した整列併合を行うプログラムの実行

注1： マニュアル名称は、製品の適応機種およびバージョンレベルによって異なります。なお、本文中では、バージョンレベルは記載されていません。

注2： ヘルプまたはオンラインマニュアルは、各製品の中にあります。

注3： NetCOBOLシリーズに同梱されています。



注意

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

本書で使用する書体と記号

書体および記号	意味
[参照]	参照先を示します。
→	操作結果を示します。
<u>あいいうえお</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. プログラム名. → PROGRAM-ID. SAMPLE1.
<u>{ あい うえお }</u> 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、使用されているシステムにより、実際の画面と異なる場合がありますので注意してください。

- 本書では、“COBOL文法書”で“原始プログラム”と記述されている用語を“ソースプログラム”と記述しています。
- OSIV/MSP、OSIV/XSPなどのOSIV系システムを総称して、“OSIV系システム”と記述しています。
- “Interstage Application Server”を総称して、“Interstage”と記述しています。なお、本書で記述している“Interstage”は、V4以前では“INTERSTAGE”に置き換えてお読みください。

輸出管理規制について

本ドキュメントを輸出または提供する場合は、外国為替および外国貿易法および米国輸出管理関連法規等の規制をご確認の上、必要な手続きをおとり下さい。

2012年12月

Copyright 1992-2012 FUJITSU LIMITED

All Rights Reserved, Copyright(C) Microsoft Corporation. 1992-2012

謝辞

COBOLの言語仕様は、データシステムズ言語協議会(COnference on DAta SYstems Languages)の作業によって開発された原仕様に基づくものであり、本書で記述される仕様もまたそれに由来する。データシステムズ言語協議会の要求によって、以下の文章を掲げる。

COBOLは産業界の言語であって、いかなる会社、組織、団体等の占有物でもない。COBOLの委員会は、このプログラミング方式及び言語の正確さと機能について、いかなる保証を与えるものでもなく、またそれに関連して、いかなる責任を負うものでもない。

次に示す著作権者は、原仕様書の作成に当たって、それぞれの著作物の一部分を利用することを承認した。この承認は、原仕様書をほかのCOBOLの仕様書で利用する場合にまで拡張されるものである。

- FLOW-MATIC(スペリランド社の商標), Programming for the Univac(R) I and II, Data Automation Systems, スペリランド社 1958年, 1959年, 著作権。
- IBM Commercial Translator, 図書番号 F28-8013, IBM社 1959年, 著作権。
- FACT, 図書番号 27A5260-2760, ミネアポリスハニウェル社 1960年, 著作権。

目 次

第1章 NetCOBOLの概要	1
1.1 NetCOBOLの機能	1
1.1.1 COBOLの機能	1
1.1.2 NetCOBOLが提供するプログラムおよびユーティリティ	1
1.2 開発環境	4
1.2.1 環境変数の設定	4
1.2.2 関連製品	7
1.3 プログラム開発の手順	7
1.4 オブジェクト指向プログラムの開発環境	9
1.5 オブジェクト指向プログラム開発の手順	9
第2章 プログラムを作成・編集する	12
2.1 プログラムの作成	12
2.1.1 COBOLソースプログラムの作成/編集	12
2.1.2 登録集原文の作成/編集	13
2.1.3 プログラムの形式	13
2.1.4 翻訳指示文	14
第3章 プログラムを翻訳する	15
3.1 サンプルプログラムの翻訳	15
3.2 翻訳に必要な資源	17
3.2.1 COBOLコンパイラが使用するファイル	17
3.2.2 COBOLコンパイラが出力する情報	20
3.2.2.1 診断メッセージ	21
3.2.2.2 オプション情報リスト、翻訳単位統計情報リスト	21
3.2.2.3 相互参照リスト	22
3.2.2.4 ソースプログラムリスト	23
3.2.2.5 目的プログラムリスト	24
3.2.2.6 データエリアに関するリスト	26
3.3 翻訳の手順	33
3.3.1 1つのソースファイルを翻訳するときの手順	33
3.3.2 複数のソースファイルを翻訳するときの手順	33
3.4 WINCOBウィンドウ	34
3.4.1 WINCOBウィンドウの起動	34
3.4.2 複数翻訳の指定	35
3.4.3 ファイル名の指定	36
3.4.4 翻訳オプションの設定	37
3.4.5 主プログラムの指定	37
3.4.6 翻訳の開始と終了	37
3.5 [翻訳オプション]ダイアログ	38
3.5.1 翻訳オプションの設定	38
3.5.1.1 翻訳オプションの追加	39
3.5.1.2 翻訳オプションの変更	40
3.5.1.3 翻訳オプションの取消し	40
3.5.2 登録集名の設定	40
3.5.2.1 登録集名の追加	41
3.5.2.2 登録集名の変更	41
3.5.2.3 登録集名の取消し	42
3.5.2.4 登録集名の設定の終了	42
3.5.3 設定したオプションの格納	42
3.6 翻訳エラーの修正	42
3.6.1 概要	42
3.6.2 翻訳時メッセージ	44
3.6.3 ビルダのエラージャンプ機能の使い方	44
3.7 翻訳コマンド	44

3.7.1 COBOL32コマンド	44
3.8 翻訳コマンドのオプション	46
3.8.1 オプション一覧	46
3.8.1.1 -A (サブスキーマ定義ファイルのフォルダの指定)	47
3.8.1.2 -Dc (COUNT機能を使用する指定)	48
3.8.1.3 -dd (デバッグ情報ファイルのフォルダの指定)	48
3.8.1.4 -Dk (CHECK機能を使用する指定)	48
3.8.1.5 -do (オブジェクトファイルのフォルダの指定)	49
3.8.1.6 -dp (翻訳リストファイルのフォルダの指定)	49
3.8.1.7 -Dr (TRACE機能を使用する指定)	50
3.8.1.8 -dr (リポジットファイルの入出力先フォルダの指定)	50
3.8.1.9 -ds (ソース解析情報ファイルのフォルダの指定)	51
3.8.1.10 -Dt (対話型デバグおよび診断機能を使用する指定)	51
3.8.1.11 -f (ファイル定義体ファイルのフォルダの指定)	51
3.8.1.12 -I (登録集ファイルのフォルダの指定)	51
3.8.1.13 -i (オプションファイルの指定)	52
3.8.1.14 -M (主プログラムを翻訳するときの指定)	52
3.8.1.15 -m (画面帳票定義体ファイルのフォルダの指定)	52
3.8.1.16 -O (広域最適化を適用する指定)	53
3.8.1.17 -P (各種翻訳リストの出力および出力先の指定)	53
3.8.1.18 -R (リポジットファイルの入力先フォルダの指定)	53
3.8.1.19 -WC (翻訳オプションの指定)	54
第4章 プログラムをリンクする	55
4.1 サンプルプログラムのリンク	55
4.2 リンクに必要な資源	56
4.2.1 WINLINKでリンクを行う場合	56
4.2.2 実行可能ファイル	58
4.2.3 DLL (ダイナミックリンクライブラリ)	58
4.2.4 インポートライブラリ	58
4.3 プログラム構造	58
4.3.1 静的構造	59
4.3.2 動的構造	59
4.3.3 プログラム構造とCALL文の書き方および翻訳オプションの関係	60
4.3.4 プログラム構造とCANCEL文の関係	61
4.4 リンクの手順	61
4.4.1 1つの目的プログラムをリンクするときの手順	61
4.4.2 DLLを作成するときの手順	61
4.4.3 単純構造の実行可能プログラムを作成するときの手順	62
4.4.4 動的リンク構造の実行可能プログラムを作成するときの手順	62
4.4.5 動的プログラム構造の実行可能プログラムを作成するときの手順	63
4.4.6 ライブラリを作成するときの手順	63
4.5 WINLINKウィンドウ	64
4.5.1 WINLINKウィンドウの起動	64
4.5.2 WINLINK[リンク]ウィンドウ	64
4.5.2.1 ターゲット名の指定	65
4.5.2.2 ファイルの指定	66
4.5.2.3 インポートライブラリの作成	66
4.5.2.4 リンクオプションの設定	66
4.5.2.5 リンクの開始と終了	67
4.5.3 WINLINK[ライブラリモジュール結合]ウィンドウ	68
4.5.3.1 ライブラリ名の指定	69
4.5.3.2 ファイルの指定	69
4.5.3.3 ライブラリ結合の開始と終了	69
4.6 モジュール定義ファイル	70
4.6.1 モジュール定義ファイルの内容	70
4.6.2 リンクの手順	71

4.6.3 モジュール定義ファイルの自動生成	71
4.7 リンクコマンドを使ったリンク操作	71
4.7.1 LINKコマンド	71
4.7.2 リンクコマンドの使用例	71
4.8 リンクコマンドの形式	73
4.8.1 LINKコマンドの形式	73
4.8.2 インポートライブラリの結合	74
4.9 リンクメッセージ	78
第5章 プログラムを実行する	80
5.1 サンプルプログラムの実行	80
5.2 実行の手順	82
5.2.1 実行環境情報の設定	82
5.2.2 COBOLプログラムの実行	82
5.2.2.1 WINEXECウィンドウを使用する方法	82
5.2.2.2 その他の方法	83
5.2.2.3 サービス配下の注意事項(実行時)	84
5.3 実行環境情報の設定	85
5.3.1 実行環境情報の種類	85
5.3.2 実行環境情報の設定方法	85
5.3.2.1 コントロールパネルのシステムで設定する方法	86
5.3.2.2 SETコマンドで設定する方法	86
5.3.2.3 実行用の初期化ファイルに設定する方法	86
5.3.2.3.1 実行用の初期化ファイルとは	86
5.3.2.3.2 実行用の初期化ファイルの内容	86
5.3.2.3.3 実行用の初期化ファイルの検索順序について	88
5.3.2.3.4 DLL配下の実行用の初期化ファイルの使用について	89
5.3.2.4 実行環境設定ツールで設定する方法	91
5.3.2.5 コマンドラインで設定する方法	91
5.4 実行環境情報の指定形式	92
5.4.1 環境変数情報	92
5.4.1.1 @AllFileExclusive (ファイルの排他処理の指定)	95
5.4.1.2 @CBR_ATTACH_TOOL (プログラムからデバッグまたは診断機能を起動する指定)	96
5.4.1.3 @CBR_CBRFILE (実行用の初期化ファイルの指定)	98
5.4.1.4 @CBR_CBRINFO (簡略化した動作状態を出力する指定)	98
5.4.1.5 @CBR_CIINF (論理宛先定義ファイルの指定)	98
5.4.1.6 @CBR_COMPOSER_CONSOLE (Interstage Business Application Serverのログ定義ファイルで定義されている管理名の指定)	98
5.4.1.7 @CBR_COMPOSER_MESS (実行時メッセージをInterstage Business Application Serverの汎用ログに出力する指定)	99
5.4.1.8 @CBR_COMPOSER_SYSERR (Interstage Business Application Serverのログ定義ファイルで定義されている管理名の指定)	99
5.4.1.9 @CBR_COMPOSER_SYSOUT (Interstage Business Application Serverのログ定義ファイルで定義されている管理名の指定)	99
5.4.1.10 @CBR_CONSOLE (コンソールウィンドウの種別の指定)	100
5.4.1.11 @CBR_CSV_OVERFLOW_MESSAGE (CSV形式データ操作時のメッセージ抑止指定)	100
5.4.1.12 @CBR_CSV_TYPE (生成するCSV形式のバリエーション)	100
5.4.1.13 @CBR_DISPLAY_CONSOLE_EVENTLOG_LEVEL (DISPLAY UPON CONSOLEのイベントログ出力時のイベント種類指定)	101
5.4.1.14 @CBR_DISPLAY_CONSOLE_EVENTLOG_SRCNAME (DISPLAY UPON CONSOLEのイベントログ出力時のイベントソース名指定)	101
5.4.1.15 @CBR_DISPLAY_CONSOLE_OUTPUT (DISPLAY UPON CONSOLEのイベントログ出力指定)	102
5.4.1.16 @CBR_DISPLAY_SYSERR_EVENTLOG_LEVEL (DISPLAY UPON SYSERRのイベントログ出力時のイベント種類指定)	102
5.4.1.17 @CBR_DISPLAY_SYSERR_EVENTLOG_SRCNAME (DISPLAY UPON SYSERRのイベントログ出力時のイベントソース名指定)	103
5.4.1.18 @CBR_DISPLAY_SYSERR_OUTPUT (DISPLAY UPON SYSERRのイベントログ出力指定)	103

5.4.1.19 @CBR_DISPLAY_SYSOUT_EVENTLOG_LEVEL (DISPLAY UPON SYSOUTのイベントログ出力時のイベント種類指定)	104
5.4.1.20 @CBR_DISPLAY_SYSOUT_EVENTLOG_SRCNAME (DISPLAY UPON SYSOUTのイベントログ出力時のイベントソース名指定)	104
5.4.1.21 @CBR_DISPLAY_SYSOUT_OUTPUT (DISPLAY UPON SYSOUTのイベントログ出力指定)	105
5.4.1.22 @CBR_DocumentName_xxxx (I制御レコードによる文書名の指定)	105
5.4.1.23 @CBR_ENTRYFILE (エントリ情報ファイルの指定)	105
5.4.1.24 @CBR_EXFH_API (外部ファイルハンドラで結合するファイルシステムの入り口名の指定)	106
5.4.1.25 @CBR_EXFH_LOAD (外部ファイルハンドラで結合するファイルシステムのDLL名の指定)	106
5.4.1.26 @CBR_FILE_BOM_READ (Unicodeの行順ファイルを参照する時の識別コードの扱いの指定)	106
5.4.1.27 @CBR_FILE_LFS_ACCESS (COBOLファイルのサイズを拡張する指定)	107
5.4.1.28 @CBR_FILE_SEQUENTIAL_ACCESS (ファイルの高速処理を一括して有効にする指定)	107
5.4.1.29 @CBR_FILE_USE_MESSAGE (入出力エラーの実行時メッセージの出力)	108
5.4.1.30 @CBR_FUNCTION_NATIONAL (NATIONAL関数の変換モードの指定)	108
5.4.1.31 @CBR_JOBDATE (任意の日付を取得)	108
5.4.1.32 @CBR_JUSTINTIME_DEBUG (異常終了時にデバッグまたは診断機能を使って調査を行う指定)	109
5.4.1.33 @CBR_MESSAGE (実行時メッセージの出力先の指定)	110
5.4.1.34 @CBR_MESS_LEVEL_CONSOLE (実行時メッセージの重大度指定)	111
5.4.1.35 @CBR_MESS_LEVEL_EVENTLOG (実行時メッセージの重大度指定)	112
5.4.1.36 @CBR_MEMORY_CHECK (メモリチェック機能を使って検査を行う指定)	113
5.4.1.37 @CBR_OverlayPrintOffset (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)	113
5.4.1.38 @CBR_OverlayPrintSPEC (フォームオーバーレイ印刷方法の指定)	115
5.4.1.39 @CBR_PrinterANK_Size (ANK文字サイズの指定)	115
5.4.1.40 @CBR_PrintFontTable (印刷ファイルで使用するフォントテーブルの指定)	116
5.4.1.41 @CBR_PrintInfoFile (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)	116
5.4.1.42 @CBR_PrintTextPosition (文字配置座標の計算方法の指定)	117
5.4.1.43 @CBR_PSFILE_xxx (表示ファイルから使用する接続製品名 (宛先ごとの指定))	117
5.4.1.44 @CBR_SCR_KEYDEFFILE (スクリーン操作のキー定義ファイルの指定)	118
5.4.1.45 @CBR_SCREEN_POSITION (スクリーン画面の表示位置の指定)	118
5.4.1.46 @CBR_SYSERR_EXTEND (SYSERR出力情報の拡張の指定)	119
5.4.1.47 @CBR_TextAlign (文字行内配置時の上端/下端合わせの指定)	119
5.4.1.48 @CBR_TRACE_FILE (トレース情報の出力ファイルの指定)	119
5.4.1.49 @CBR_TRACE_PROCESS_MODE (TRACEファイルのプロセス毎の出力指定)	119
5.4.1.50 @CBR_TRAILING_BLANK_RECORD (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)	120
5.4.1.51 @CnslBufLine (コンソールウィンドウのバッファ数の指定)	120
5.4.1.52 @CnslFont (コンソールウィンドウのフォントの指定)	120
5.4.1.53 @CnslWinSize (コンソールウィンドウの大きさの指定)	121
5.4.1.54 @DefaultFCB_Name (デフォルトFCB名の指定)	121
5.4.1.55 @ExitSessionMSG (COBOLアプリケーション起動中のWindows終了指定)	121
5.4.1.56 @GOPT (実行時オプションの指定)	122
5.4.1.57 @IconDLL (アイコンリソースのDLL名の指定)	122
5.4.1.58 @IconName (アイコンリソースの識別名の指定)	123
5.4.1.59 @MessOutFile (メッセージを出力するファイルの指定)	123
5.4.1.60 @MGPRM (OSIV系形式の実行時パラメタの指定)	123
5.4.1.61 @NoMessage (実行時メッセージおよびSYSERRの出力抑止指定)	124
5.4.1.62 @ODBC_Inf (ODBC情報ファイルの指定)	124
5.4.1.63 @PrinterFontName (印刷ファイルで使用するフォントの指定)	124
5.4.1.64 @PRN_FormName_xxx (用紙名の指定)	125
5.4.1.65 @ScrnFont (スクリーン操作で使用するフォントの指定)	126
5.4.1.66 @ScrnSize (スクリーン操作の論理画面の大きさの指定)	126
5.4.1.67 @ShowIcon (NetCOBOLのアイコン表示の抑止指定)	126
5.4.1.68 @WinCloseMsg (ウィンドウを閉じるときのメッセージ表示の指定)	126
5.4.1.69 ファイル識別名 (プログラムで使用するファイルの指定)	127
5.4.1.70 ファイル識別名 (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)	128
5.4.1.71 ファイル識別名 (プログラムで使用するプリンタ情報ファイルおよび各種パラメタの指定)	128
5.4.1.72 ファイル識別名 (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)	129

5.4.1.73 FCBxxxx (FCB制御文の指定)	132
5.4.1.74 FOVLDIR (フォームオーバーレイパターンフォルダの指定)	132
5.4.1.75 FOVLTYPE (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)	132
5.4.1.76 FOVLNAME (フォームオーバーレイパターンのファイル名の指定)	133
5.4.1.77 OVD_SUFFIX (フォームオーバーレイパターンのファイルの拡張子の指定)	133
5.4.1.78 SYSCOUNT (COUNT情報の出力ファイルの指定)	133
5.4.1.79 SYSINのアクセス名 (小入出力機能の入力ファイルの指定)	134
5.4.1.80 SYSOUTのアクセス名 (小入出力機能の出力ファイルの指定)	134
5.4.2 副プログラムのエントリ情報	135
5.4.2.1 エントリ情報の指定方法	135
5.4.2.2 副プログラム名の指定形式	138
5.4.2.3 二次入口点名の指定形式	141
5.5 WINEXECウィンドウ	141
5.5.1 WINEXECウィンドウの起動	141
5.5.2 ファイル名の指定	142
5.5.3 実行の開始と終了	143
5.6 実行環境設定ツール	143
5.6.1 環境変数情報の設定	144
5.6.1.1 実行用の初期化ファイルのオープン	144
5.6.1.2 環境変数情報の追加	145
5.6.1.3 環境変数情報の変更	145
5.6.1.4 環境変数情報の取消し	145
5.6.1.5 実行用の初期化ファイルのクローズ	145
5.6.2 エントリ情報の設定	146
5.6.3 実行用の初期化ファイルへの保存	146
5.6.4 プリンタの設定	146
5.6.5 実行環境設定ツールの終了	146
5.7 実行用の初期化ファイル名の指定形式	146
5.8 実行時オプションの指定形式	148
5.9 注意事項	149
5.9.1 COBOLプログラムの実行時にスタックオーバーフローが発生する場合	149
5.9.2 COBOLプログラムの実行時に仮想メモリ不足が発生する場合	154
第6章 プロジェクト管理機能	156
6.1 プロジェクト管理機能とは	156
6.2 プロジェクト管理に必要な資源	156
6.3 プロジェクト管理の手順	159
6.4 プロジェクトマネージャウィンドウ	160
6.4.1 プロジェクトファイルの作成	162
6.4.2 翻訳オプションの設定	162
6.4.3 リンクオプションの設定	163
6.4.3.1 [リンクオプション]ダイアログ	163
6.4.4 ファイルの登録	165
6.4.4.1 最終ターゲットファイルの登録	165
6.4.4.2 COBOLソースファイルの登録	167
6.4.4.3 主プログラムの指定	168
6.4.4.4 登録集ファイルの登録	168
6.4.4.5 各種定義体ファイルの登録	169
6.4.4.6 その他必要なファイルの登録	169
6.4.4.7 フォルダの作成	170
6.4.4.8 プロジェクトファイルの保存	172
6.4.5 プロジェクトのビルド・リビルド	172
6.4.5.1 デバッグモジュールの作成方法	173
6.4.6 ファイルの内容の修正	173
6.4.7 アプリケーションプログラムの実行	173
6.4.8 プリコンパイラの登録とプリコンパイルの実行	173
6.4.8.1 資産とツール	173

6.4.8.2 プロジェクト管理	175
6.4.8.3 プロジェクト管理の手順	175
6.4.8.4 [プリコンパイラ設定]ダイアログ	177
6.4.8.4.1 新規登録の場合	178
6.4.8.4.2 更新登録の場合	182
6.4.8.4.3 マクロ	184
6.4.8.5 選択項目と対象プリコンパイラ	185
6.4.8.6 インクルードファイル	185
6.4.8.7 多段プリコンパイル	186
6.4.8.8 注意事項	187
6.4.9 実行可能プログラムのバージョン設定機能	187
6.4.10 リストビュー	189
6.5 モジュール定義ファイル	191
6.5.1 モジュール定義ファイルの生成方法	191
6.6 MAKEファイル	192
6.6.1 MAKEファイルの用途	192
6.6.2 プロジェクトマネージャのMAKEファイル	193
6.6.3 MAKEファイルとプロジェクトファイル	193
6.6.4 コマンドファイル	194
第7章 ファイルの処理	195
7.1 ファイルの種類	195
7.1.1 ファイルの種類と特徴	195
7.1.2 レコードの設計	197
7.1.2.1 レコード形式	198
7.1.2.2 索引ファイルのレコードキー	198
7.1.3 ファイルの処理方法	198
7.2 レコード順ファイルの使い方	199
7.2.1 レコード順ファイルの定義	199
7.2.2 レコード順ファイルのレコードの定義	200
7.2.3 レコード順ファイルの処理	201
7.3 行順ファイルの使い方	203
7.3.1 行順ファイルの定義	203
7.3.2 行順ファイルのレコードの定義	203
7.3.3 行順ファイルの処理	204
7.4 相対ファイルの使い方	208
7.4.1 相対ファイルの定義	208
7.4.2 相対ファイルのレコードの定義	210
7.4.3 相対ファイルの処理	211
7.5 索引ファイルの使い方	213
7.5.1 索引ファイルの定義	214
7.5.2 索引ファイルのレコードの定義	216
7.5.3 索引ファイルの処理	217
7.6 入出力エラー処理	220
7.6.1 AT END指定	221
7.6.2 INVALID KEY指定	221
7.6.3 FILE STATUS句	221
7.6.4 誤り処理手続き	222
7.6.5 入出力エラーが発生したときの実行結果	222
7.7 ファイル処理の実行	223
7.7.1 ファイルの割当て	223
7.7.2 ファイルの排他制御	226
7.7.2.1 ファイルを排他モードにする方法	226
7.7.2.2 レコードを排他状態にする方法	228
7.7.3 ファイル処理の結果	229
7.7.4 ファイルの高速処理	230
7.7.5 ファイル追加書き	234

7.7.6 ファイルの連結	234
7.7.7 ダミーファイル	234
7.7.8 注意事項	235
7.8 COBOLファイルユーティリティ	237
7.8.1 COBOLファイルユーティリティとは	237
7.8.2 COBOLファイルユーティリティの使い方	238
7.8.3 COBOLファイルユーティリティの機能	239
7.8.3.1 ファイルの創成	239
7.8.3.2 ファイルの拡張	240
7.8.3.3 レコードの表示	242
7.8.3.4 レコードの編集	243
7.8.3.5 レコードの整列	245
7.8.3.6 ファイルの操作	245
7.8.3.7 ファイルの印刷	246
7.8.3.8 ファイルの構造の変換	246
7.8.3.9 索引ファイルの操作	247
7.8.4 COBOLファイルユーティリティコマンドモード	248
7.8.4.1 ファイル変換コマンド (cobfconv)	248
7.8.4.2 ファイルロードコマンド (cobfload)	252
7.8.4.3 ファイルアンロードコマンド (cobfulod)	254
7.8.4.4 ファイル表示コマンド (cobfbrws)	255
7.8.4.5 ファイル整列コマンド (cobfsort)	258
7.8.4.6 ファイル属性コマンド (cobfattr)	262
7.8.4.7 ファイル復旧コマンド (cobfrcov)	263
7.8.4.8 ファイル再編成コマンド (cobfreog)	263
7.8.5 COBOLファイルユーティリティ関数	264
7.8.5.1 ファイル変換関数	265
7.8.5.2 ファイルロード関数	268
7.8.5.3 ファイルアンロード関数	271
7.8.5.4 ファイル整列関数	272
7.8.5.5 ファイル再編成関数	274
7.8.5.6 ファイルコピー関数	275
7.8.5.7 ファイル移動関数	276
7.8.5.8 ファイル削除関数	276
7.9 他のファイルシステムの使用法	277
7.9.1 Btrieveファイル	280
7.9.1.1 ファイル環境の指定	280
7.9.1.2 外部10進項目のデータ形式変換	281
7.9.1.3 注意事項	283
7.9.2 RDMファイル	283
7.9.2.1 ファイル環境の指定	284
7.9.2.2 注意事項	285
7.9.3 XLデータパイプ	285
7.9.3.1 使用方法	285
7.9.3.2 機能範囲	285
7.9.4 外部ファイルハンドラ	286
7.10 索引ファイルの復旧	287
7.10.1 索引ファイル復旧関数 (CFURCOV)	288
7.10.2 索引ファイル簡易復旧関数 (CFURCOVS)	288
7.10.3 注意事項	289
7.10.4 COBOLから呼び出す場合の使用例	290
7.10.5 メッセージの内容とコード	292
7.11 COBOLファイルアクセスルーチン	292
第8章 印刷処理	293
8.1 印刷方法の種類	293
8.1.1 各印刷方法の概要	293

8.1.2 印字文字	295
8.1.3 フォームオーバーレイパターン	299
8.1.4 FCB	300
8.1.5 I制御レコード/S制御レコード	301
8.1.6 帳票定義体	309
8.1.7 特殊レジスタ	310
8.1.8 印字文字の配置座標	311
8.1.9 印刷情報ファイル	311
8.1.10 印刷ファイル/表示ファイルの決定方法	316
8.1.11 サービス配下の注意事項(印刷時)	317
8.1.12 帳票設計について	318
8.1.13 印刷不可能な領域について	319
8.1.14 フォントテーブル	320
8.2 行単位のデータを印刷する方法	321
8.2.1 概要	321
8.2.2 プログラムの記述	321
8.2.3 プログラムの翻訳・リンク	324
8.2.4 プログラムの実行	324
8.3 フォームオーバーレイおよびFCBを使う方法	329
8.3.1 概要	329
8.3.2 プログラムの記述	330
8.3.3 プログラムの翻訳・リンク	332
8.3.4 プログラムの実行	332
8.3.4.1 フォームオーバーレイパターンを使うプログラム	332
8.3.4.2 FCBを使うプログラム	333
8.4 帳票定義体を使う印刷ファイルの使い方	334
8.4.1 概要	334
8.4.1.1 帳票のパーティション	335
8.4.1.2 帳票の電子化	337
8.4.2 プログラムの記述	339
8.4.3 プログラムの翻訳・リンク	342
8.4.4 プログラムの実行	342
8.5 表示ファイル(帳票印刷)の使い方	343
8.5.1 概要	343
8.5.2 作業手順	345
8.5.3 帳票定義体の作成	346
8.5.4 プログラムの記述	346
8.5.5 プログラムの翻訳・リンク	349
8.5.6 プリンタ情報ファイルの作成	349
8.5.7 プログラムの実行	349
8.6 電子帳票出力機能を使う方法	350
8.6.1 電子帳票出力機能の概要	350
8.6.2 帳票を電子化する方法	351
8.6.3 電子帳票の出力例	355
8.6.4 プリンタ(紙)出力時と電子帳票出力時の機能差(留意事項/制限事項)	355
8.6.5 実行時エラーについて	358
第9章 画面を使った入出力	359
9.1 画面を使った入出力の種類	359
9.2 表示ファイル(画面入出力)の使い方	359
9.2.1 概要	359
9.2.2 動作環境	360
9.2.3 作業手順	362
9.2.4 画面定義体の作成	362
9.2.5 プログラムの記述	363
9.2.6 プログラムの翻訳・リンク	367
9.2.7 ウィンドウ情報ファイルの作成	367

9.2.8 プログラムの実行.....	367
9.3 スクリーン操作機能の使い方.....	368
9.3.1 概要.....	369
9.3.2 スクリーンウィンドウ.....	369
9.3.3 キー定義ファイルの利用.....	369
9.3.4 プログラムの記述.....	372
9.3.5 プログラムの翻訳・リンク.....	374
9.3.6 プログラムの実行.....	374
第10章 サブプログラムを呼び出す～プログラム間連絡機能～.....	376
10.1 呼出し関係の概要.....	376
10.1.1 呼出し関係の形態.....	376
10.1.2 各種呼出し規約の違い.....	376
10.1.3 呼出し規約と指定形式.....	377
10.1.4 COBOLの言語間の環境.....	379
10.1.5 動的プログラム構造.....	382
10.1.5.1 動的プログラム構造の特徴.....	382
10.1.5.2 副プログラムのエントリ情報.....	383
10.1.5.3 注意事項.....	383
10.2 COBOLプログラムからCOBOLプログラムを呼び出す.....	386
10.2.1 呼出し方法.....	387
10.2.2 二次入口点.....	387
10.2.3 制御の復帰とプログラムの終了.....	387
10.2.4 パラメタの受渡し方法.....	387
10.2.5 データの共用.....	389
10.2.5.1 外部データ使用時の注意事項.....	390
10.2.5.2 外部ファイル使用時の注意事項.....	390
10.2.6 復帰コード.....	391
10.2.7 内部プログラム.....	392
10.2.8 注意事項.....	393
10.3 C言語プログラムとの結合.....	394
10.3.1 COBOLプログラムからCプログラムを呼び出す方法.....	395
10.3.1.1 呼出し方法.....	395
10.3.1.2 パラメタの受渡し方法.....	395
10.3.1.3 復帰コード(関数値).....	397
10.3.2 CプログラムからCOBOLプログラムを呼び出す方法.....	400
10.3.2.1 呼出し方法.....	401
10.3.2.2 パラメタの受渡し方法.....	401
10.3.2.3 復帰コード(関数値).....	401
10.3.3 データ型の対応.....	403
10.3.4 プログラムの翻訳.....	406
10.3.5 プログラムのリンク.....	406
10.3.5.1 COBOLプログラムを呼び出すCプログラムの結合方法.....	407
10.3.5.2 Cプログラムを呼び出すCOBOLプログラムの結合方法.....	410
10.3.6 プログラムの実行.....	413
10.4 Visual BasicからCOBOLプログラムを呼び出す.....	414
10.4.1 呼出し方法.....	414
10.4.2 データ型の対応.....	414
10.4.3 Visual BasicからCOBOLの呼出し例.....	414
第11章 ACCEPT文およびDISPLAY文の使い方.....	417
11.1 小入出力機能.....	417
11.1.1 概要.....	417
11.1.2 入出力先の種類と指定方法.....	417
11.1.3 コンソールウィンドウを使ったデータの入出力.....	419
11.1.3.1 COBOLのコンソールウィンドウ.....	419
11.1.3.2 コマンドプロンプト.....	419
11.1.3.3 システムのコンソールウィンドウ.....	420

11.1.3.4 プログラムの記述	420
11.1.3.5 プログラムの翻訳・リンク	421
11.1.3.6 プログラムの実行	421
11.1.4 メッセージボックスにメッセージを出力する方法	422
11.1.4.1 メッセージボックス	422
11.1.4.2 プログラムの記述	422
11.1.4.3 プログラムの翻訳・リンク	423
11.1.4.4 プログラムの実行	423
11.1.5 ファイルを使うプログラム	423
11.1.5.1 プログラムの記述	423
11.1.5.2 プログラムの翻訳・リンク	425
11.1.5.3 プログラムの実行	425
11.1.5.4 DISPLAY文のファイル出力拡張機能	426
11.1.5.5 ACCEPT文のファイル入力拡張機能	427
11.1.6 現在の日付および時刻の入力	428
11.1.6.1 プログラムの記述	428
11.1.6.2 プログラムの翻訳・リンク	429
11.1.6.3 プログラムの実行	429
11.1.7 任意の日付の入力	429
11.1.7.1 プログラムの記述	429
11.1.7.2 プログラムの翻訳・リンク	430
11.1.7.3 プログラムの実行	430
11.1.8 Interstage Business Application Serverの汎用ログを使うプログラム	431
11.1.8.1 プログラムの記述	431
11.1.8.2 プログラムの翻訳・リンク	431
11.1.8.3 プログラムの実行	431
11.1.9 イベントログを使うプログラム	432
11.1.9.1 プログラムの記述	432
11.1.9.2 プログラムの翻訳・リンク	432
11.1.9.3 プログラムの実行	432
11.2 コマンド行引数の取出し	434
11.2.1 概要	434
11.2.2 プログラムの記述	435
11.2.3 プログラムの翻訳・リンク	436
11.2.4 プログラムの実行	437
11.3 環境変数の操作機能	437
11.3.1 概要	437
11.3.2 プログラムの記述	437
11.3.3 プログラムの翻訳・リンク	438
11.3.4 プログラムの実行	438
第12章 SORT文およびMERGE文の使い方～整列併合機能～	439
12.1 ソート・マージ処理の概要	439
12.2 ソートの使い方	440
12.2.1 ソート処理の種類	440
12.2.2 プログラムの記述	440
12.2.3 プログラムの翻訳・リンク	443
12.2.4 プログラムの実行	443
12.3 マージの使い方	443
12.3.1 マージ処理の種類	443
12.3.2 プログラムの記述	444
12.3.3 プログラムの翻訳・リンク	446
12.3.4 プログラムの実行	446
第13章 CSV形式データの操作	447
13.1 CSV形式データとは	447
13.2 CSV形式データの作成 (STRING文)	448
13.2.1 基本操作	448

13.2.2 処理異常の検出.....	449
13.3 CSV形式データの分解 (UNSTRING文).....	449
13.3.1 基本操作.....	449
13.3.2 処理異常の検出.....	450
13.4 CSV形式のバリエーション.....	451
第14章 システムプログラムを記述するための機能～SD機能～.....	453
14.1 SD機能の種類.....	453
14.2 ポインタ付けの使い方.....	453
14.3 ADDR関数およびLENG関数の使い方.....	454
14.4 終了条件なしのPERFORM文の使い方.....	455
第15章 オブジェクト指向プログラミングとは.....	456
15.1 オブジェクト指向の歴史的背景.....	456
15.1.1 ソフトウェア工学以前.....	456
15.1.2 構造化プログラミングの出現.....	456
15.1.3 モジュール化.....	456
15.1.4 抽象データ型.....	457
15.1.5 オブジェクト指向の誕生.....	458
15.2 オブジェクト指向の基本的な考え方.....	459
15.2.1 オブジェクトとクラス.....	459
15.2.2 オブジェクトの作成・参照.....	460
15.2.3 メソッドの呼出し.....	460
15.2.4 継承.....	462
15.2.5 多態と動的束縛.....	465
15.3 オブジェクト指向のメリット.....	467
第16章 オブジェクト指向プログラミング機能～基本的な使い方～.....	468
16.1 ソース定義.....	468
16.1.1 クラス定義.....	468
16.1.2 ファクトリ定義.....	469
16.1.3 オブジェクト定義.....	471
16.1.4 メソッド定義.....	473
16.2 オブジェクトインスタンスの操作.....	475
16.2.1 メソッドの呼出し.....	475
16.2.1.1 オブジェクト参照項目.....	476
16.2.1.2 INVOKE文.....	477
16.2.1.3 パラメタの指定.....	478
16.2.2 オブジェクトの寿命.....	480
16.3 継承.....	481
16.3.1 継承の概念と実現.....	482
16.3.2 FJBASEクラス.....	484
16.3.3 メソッドの上書き.....	486
16.4 適合.....	487
16.4.1 適合の概念.....	487
16.4.2 オブジェクト参照項目と適合チェック.....	489
16.4.3 翻訳時の適合チェックと実行時の適合チェック.....	490
16.4.3.1 代入時の適合チェック.....	490
16.4.3.2 メソッド呼出し時の適合チェック.....	490
16.5 リポジトリ.....	491
16.5.1 リポジトリファイルの概要.....	491
16.5.1.1 クラスブラウザへの入力.....	491
16.5.1.2 継承の実現.....	491
16.5.1.3 適合チェックの実現.....	492
16.5.2 リポジトリファイル更新の影響.....	493
16.6 メソッドの束縛.....	493
16.6.1 メソッドの静的束縛.....	494
16.6.2 メソッドの動的束縛と多態.....	494

16.6.3 定義済みオブジェクト一意名SUPER	497
16.6.4 定義済みオブジェクト一意名SELF	497
16.7 少し進んだ使い方	499
16.7.1 メソッドのPROTOTYPE宣言	500
16.7.2 多重継承	500
16.7.3 行内呼出し	502
16.7.4 オブジェクト指定子	503
16.7.5 PROPERTY句	504
16.7.6 初期化处理メソッドと終了処理メソッド	507
16.7.7 間接参照クラス	509
16.7.8 相互参照クラス	510
16.7.8.1 相互参照パターン	510
16.7.8.2 相互参照クラスの翻訳	514
16.7.8.3 相互参照クラスのリンク	515
16.7.8.4 相互参照クラスの実行	517
第17章 オブジェクト指向プログラミングの開発	518
17.1 オブジェクト指向プログラミングで使用する資源	518
17.2 開発手順	519
17.3 クラスの検索とクラス情報の取得	519
17.3.1 クラスブラウザ	520
17.3.1.1 概要	520
17.3.1.2 使用する資源	521
17.3.1.3 クラス情報データベースの生成	521
17.3.1.3.1 オプション設定ウィンドウ	522
17.3.1.3.2 データベース生成	523
17.3.1.4 クラスブラウザの表示/操作手順	523
17.3.1.5 クラスブラウザウィンドウ	523
17.3.1.6 クラスの検索条件の入力	526
17.4 プロジェクト管理	527
17.4.1 プロジェクト管理のメリット	527
17.4.2 プログラム構造	527
17.4.2.1 翻訳単位とリンク単位	527
17.4.2.2 プログラム構造の概要	528
17.4.3 翻訳処理	531
17.4.3.1 リポトリファイル	531
17.4.4 リンク処理	538
17.4.4.1 インポートライブラリ	538
17.4.5 相互参照クラス	541
17.4.5.1 相互参照クラスのプロジェクト管理	541
17.4.5.2 相互参照クラスの翻訳処理	542
17.4.5.3 相互参照クラスのリンク処理	544
17.4.6 プロジェクトブラウザ	546
17.4.6.1 概要	546
17.4.6.2 使用する資源	546
17.4.6.3 プロジェクト情報データベース生成	547
17.4.6.4 プロジェクトブラウザの表示/操作手順	548
17.4.6.5 プロジェクトブラウザウィンドウ	548
17.5 クラスの公開	553
第18章 オブジェクト指向プログラミング機能～さらに進んだ使い方～	555
18.1 例外処理	555
18.1.1 概要	555
18.1.2 例外オブジェクト	555
18.1.3 RAISE文の動作	556
18.1.4 RAISING指定のEXIT文の動作	556
18.2 動的プログラム構造	558
18.2.1 動的プログラム構造の特徴	558

18.2.2 クラスの動的プログラム構造	558
18.2.3 メソッドの動的プログラム構造	559
18.2.4 動的プログラム構造の作成から実行	560
18.2.4.1 翻訳の方法	560
18.2.4.2 リンクの方法	561
18.2.4.3 DLLの構成とファイル名	562
18.2.4.4 クラスとメソッドのエントリ情報	562
18.3 メモリに関するチューニング	565
18.3.1 概要	565
18.3.2 使用メモリの節約	565
18.3.3 実行性能の向上	566
18.3.4 メモリのチューニングに関する実行環境情報	567
18.3.4.1 環境変数情報	567
18.3.4.1.1 @CBR_ClassInfFile (クラス情報ファイルの指定)	567
18.3.4.1.2 @CBR_InstanceBlock (オブジェクトインスタンスの獲得方法の指定)	567
18.3.4.2 クラス情報	568
18.3.4.2.1 InstanceBlockセクション (オブジェクトインスタンスの格納数の指定)	568
18.4 Visual C++プログラムとの連携	568
18.4.1 概要	568
18.4.2 Visual C++連携の方法	568
18.4.3 Visual C++連携の概要	569
18.4.3.1 COBOLおよびVisual C++でのクラスの対応	569
18.4.3.2 処理の概要	569
18.4.3.3 インタフェースプログラムの仕組み	570
18.4.4 Visual C++連携のプログラム手順	572
18.4.4.1 Visual C++で定義されているクラスを調べる	573
18.4.4.2 COBOL側での定義	573
18.4.4.3 Visual C++側での定義	574
18.4.5 COBOLからの利用	574
18.4.6 サンプルプログラム	575
18.5 オブジェクトの永続化	577
18.5.1 オブジェクトの永続化とは	577
18.5.2 概要	577
18.5.3 クラス構造の例	578
18.5.4 索引ファイルとオブジェクトの対応	579
18.5.4.1 クラスとファイルの対応	579
18.5.4.2 索引ファイルの定義	581
18.5.5 オブジェクトの保存/復元	581
18.5.5.1 索引ファイル操作クラス	582
18.5.5.2 保存するオブジェクトのメソッドの追加	582
18.5.5.3 処理の流れ	583
18.6 特殊クラス	584
18.6.1 *COB-BINDTABLEクラス	584
18.6.1.1 概要	584
18.6.1.2 *COB-BINDTABLEクラスの説明	586
18.6.1.2.1 *COB-BINDTABLEクラスファクトリメソッド	586
18.6.1.2.2 *COB-BINDTABLEクラスオブジェクトメソッド	587
18.7 ANY LENGTH句を使用したプログラミング	592
18.7.1 文字列を扱うクラス	592
18.7.2 ANY LENGTH句の使用	593
第19章 プログラムのデバッグ	595
19.1 デバッグ機能の概要	595
19.1.1 ステートメント番号	597
19.2 CHECK機能の使い方	597
19.2.1 デバッグ作業の流れ	597
19.2.2 出力メッセージ	598

19.2.2.1 メッセージの内容	598
19.2.2.2 メッセージの出力回数	601
19.2.3 CHECK機能の使用例	601
19.2.4 注意事項	605
19.3 TRACE機能の使い方	606
19.3.1 デバッグ作業の流れ	606
19.3.2 トレース情報	607
19.3.3 注意事項	609
19.4 COUNT機能の使い方	610
19.4.1 デバッグ作業の流れ	610
19.4.2 COUNT情報	611
19.4.3 COUNT機能を使用したプログラムのデバッグ	614
19.4.4 注意事項	614
19.5 メモリチェック機能の使い方	615
19.5.1 デバッグ作業の流れ	615
19.5.2 出力メッセージ	616
19.5.2.1 メッセージの内容	616
19.5.3 診断レポート	617
19.5.4 プログラムの特定	618
19.5.5 破壊箇所の特定制	618
19.5.5.1 指定方法	618
19.5.6 注意事項	619
19.6 診断機能の使い方	619
19.6.1 診断機能の概要	619
19.6.2 診断機能が使用する資源	619
19.6.2.1 対象プログラム	619
19.6.2.2 プログラムの翻訳・リンク方法と出力情報との関係	620
19.6.2.3 プログラムの翻訳・リンク	621
19.6.2.4 デバッグ情報ファイルの格納先	622
19.6.3 診断機能の起動	622
19.6.3.1 アプリケーションエラーやUレベルの実行時メッセージが発生するアプリケーションに対する起動	622
19.6.3.2 応答しなくなったアプリケーションに対する起動	623
19.6.3.3 領域破壊が発生するアプリケーションに対する起動	623
19.6.3.4 起動パラメタ	624
19.6.4 診断レポート	626
19.6.4.1 診断レポートの出力先	626
19.6.4.2 診断レポートの出力情報	627
19.6.4.2.1 アプリケーションエラー、実行時メッセージおよび応答なしの場合	627
19.6.4.2.2 領域破壊監視の場合	633
19.6.4.3 診断レポートの見方	635
19.6.4.3.1 アプリケーションエラー	635
19.6.4.3.2 実行時メッセージ	636
19.6.4.3.3 応答なし	640
19.6.4.3.4 領域破壊監視	640
19.6.5 ダンプ	643
19.6.5.1 ダンプとは	643
19.6.5.2 ダンプの出力先	643
19.6.5.3 ダンプファイル数の管理	644
19.6.6 注意事項	644
19.7 翻訳リストを使ったデバッグ	645
19.7.1 デバッグ作業の流れ	645
19.7.2 障害発生箇所の特定制方法	647
19.7.2.1 障害発生箇所の特定制方法 (EXEファイルの場合)	648
19.7.2.2 障害発生箇所の特定制方法 (DLLファイルの場合)	651
19.7.2.3 データの参照方法	652
第20章 対話型デバッグの使い方	657

20.1 デバッガの概要	657
20.2 デバッグの準備	658
20.2.1 被デバッグプログラム	658
20.2.2 プログラムの翻訳	659
20.2.3 プログラムのリンク	659
20.3 デバッガの環境変数	659
20.3.1 @SVD_PERFORM_CHECK (PERFORMレベルの監視を行う指定)	660
20.3.2 @SVD_ENV (デバッグ環境の指定)	660
20.3.3 @SVD_COMPATIBILITY (互換性の指定)	661
20.4 デバッガの起動	662
20.4.1 起動方法	662
20.4.1.1 一般的なプログラムに対するデバッグ起動方法	662
20.4.1.2 Interstageなどのサーバ環境で動作するプログラムに対するデバッグ起動方法	663
20.4.1.3 ジャストインタイムデバッグ機能を使用したデバッグ起動方法	663
20.4.2 デバッグに必要な資源やパラメタの指定	663
20.4.2.1 [デバッグを開始する]ダイアログを使ったデバッグ指定	663
20.4.2.2 起動コマンドを使ったデバッグ指定	668
20.5 デバッガの操作	669
20.5.1 デバッガの画面	669
20.5.1.1 メインウィンドウ	669
20.5.1.2 ソースファイルウィンドウ	671
20.5.1.3 プログラム一覧ウィンドウ	671
20.5.1.4 監視ウィンドウ	672
20.5.1.5 ラインコマンド入力ウィンドウ	672
20.5.1.6 呼出し経路ウィンドウ	673
20.5.1.7 スレッド一覧ウィンドウ	673
20.5.2 デバッガの機能	674
20.5.2.1 デバッグ機能の使い方	674
20.5.2.1.1 データの内容を確認しながらプログラムの誤りを検出する	674
20.5.2.1.2 プログラムの実行順序を変更する	675
20.5.2.1.3 データの変更箇所を調べる	675
20.5.2.1.4 実行経路をたどる	676
20.5.2.1.5 プログラムの呼出し経路をたどる	677
20.5.2.1.6 デバッガの状態を知る	677
20.5.2.2 デバッグ操作の自動化	678
20.5.2.3 その他の補助的な機能	679
20.6 プログラムの特徴に合わせたデバッグ方法	682
20.6.1 動的構造に対するデバッグ方法	682
20.6.2 例外ハンドラで処理されるシグナルのデバッグ方法	682
20.6.3 オブジェクト指向に対応したデバッグ機能	683
20.6.4 マルチスレッドでのデバッガの対応	683
20.6.4.1 デバッグ機能のマルチスレッドでの基本的な動作	683
20.6.4.2 デバッグ機能のマルチスレッド特有な動作	683
20.6.4.3 マルチスレッドのデバッグを補助する機能	684
20.6.4.4 マルチスレッドでのデバッグ操作の自動化	684
20.6.5 デバッガのUnicode対応	685
20.6.5.1 Unicodeプログラムのデバッグ方法	685
20.6.5.2 Unicodeを扱う上での注意事項	685
20.7 注意事項	685
20.8 デバッガのコマンド一覧	687
20.9 対話型リモートデバッガの使い方	688
20.9.1 リモートデバッガの概要	689
20.9.1.1 リモートデバッガ導入による効果	689
20.9.1.2 リモートデバッグ時の資産の格納場所	690
20.9.1.3 リモートデバッグ時の被デバッグプログラムの作成方法	690
20.9.2 リモートデバッグの開始手順	690
20.9.2.1 一般的なプログラムに対するリモートデバッグ起動方法	691

20.9.2.2 Interstageなどのサーバ環境で動作するプログラムに対するリモートデバッグ起動方法	691
20.9.2.3 ジャストインタイムデバッグ機能を使用したリモートデバッグ起動方法	693
20.9.3 リモートデバッグコネクタの使い方	693
20.9.3.1 サーバ側のリモートデバッグコネクタの使い方	693
20.9.3.2 サーバ側のリモートデバッグコネクタの起動方法	693
20.9.3.3 接続制限ファイルの形式	695
20.9.3.4 クライアント側のリモートデバッグコネクタの使い方	696
20.9.3.4.1 クライアント側のリモートデバッグコネクタの起動方法	696
20.9.3.4.2 [リモートデバッグコネクタ]ダイアログの設定方法	696
20.9.3.5 リモートデバッグを許可するコンピュータを制限する	699
20.9.4 リモートデバッグの環境変数	700
20.9.4.1 @SVD_COMPATIBILITY (互換性の指定)	700
20.9.5 リモートデバッグの起動	701
20.9.5.1 [デバッグを開始する]ダイアログを使ったリモートデバッグ指定	701
20.9.5.2 起動コマンドを使ったリモートデバッグ指定	708
20.9.6 リモートデバッグの操作	710
20.9.7 リモートデバッグの注意事項	710
20.9.7.1 Win32/Win64リモートデバッグの注意事項	710
20.9.7.2 Solaris/Linuxリモートデバッグの注意事項	711
第21章 通信機能	712
21.1 通信の種類	712
21.2 表示ファイル(非同期型メッセージ通信)の使い方	713
21.2.1 概要	713
21.2.2 プログラムの記述	713
21.2.3 プログラムの翻訳・リンク	716
21.2.4 プログラムの実行	716
21.2.5 ACM通信機能の利用	716
21.3 簡易アプリ間通信機能の使い方	716
21.3.1 概要	717
21.3.2 作業手順	717
21.3.3 サーバ操作ウィンドウ	719
21.3.3.1 [通信システム環境設定]ダイアログ	719
21.3.3.2 簡易アプリ間通信サーバウィンドウ	720
21.3.3.3 サーバ定義ファイルを利用した起動	726
21.3.4 クライアント操作ウィンドウ	729
21.3.4.1 論理宛先定義ファイル作成ユーティリティ	729
21.3.5 メモリ資源の見積り	732
21.3.6 注意事項	732
21.4 簡易アプリ間通信のサブルーチン	734
21.4.1 COBCI_OPEN	735
21.4.2 COBCI_CLOSE	736
21.4.3 COBCI_READ	737
21.4.4 COBCI_WRITE	739
21.5 簡易アプリ間通信のサブルーチンのエラーコード	740
第22章 リモートデータベースアクセス	744
22.1 プリコンパイラを使用したアクセス	744
22.2 ODBC経由によるアクセス	744
22.2.1 概要	744
22.2.1.1 COBOLプログラムの構成	745
22.2.1.2 埋込みSQL文による操作	746
22.2.2 コネクション操作	747
22.2.2.1 コネクションを接続する	747
22.2.2.2 コネクションを切断する	748
22.2.2.3 コネクションを変更する	748
22.2.3 データ操作	749
22.2.3.1 サンプルデータベース	749

22.2.3.2 データの参照	751
22.2.3.2.1 表の全行を参照する	751
22.2.3.2.2 条件を指定して参照する	753
22.2.3.2.3 1つの行を参照する	754
22.2.3.2.4 表を関連付けてデータを参照する	754
22.2.3.3 データの更新	756
22.2.3.4 データの削除	756
22.2.3.5 データの挿入	756
22.2.3.6 動的SQL	757
22.2.3.7 可変長文字列の使用	760
22.2.3.8 複数コネクションでのカーソル操作	761
22.2.4 高度なデータ操作	762
22.2.4.1 高度なデータ操作を可能とするホスト変数	762
22.2.4.1.1 複数行指定ホスト変数	763
22.2.4.1.2 表指定ホスト変数	767
22.2.4.2 動的SQL文で使用方法	769
22.2.4.3 SQLERRDによる処理行数の確認方法	769
22.2.4.4 FOR句による処理行数制御	770
22.2.4.5 FETCH PRIOR文によるデータの取得	772
22.2.5 ストアドプロシージャの呼出し	775
22.2.5.1 ストアドプロシージャとは	775
22.2.5.2 ストアドプロシージャの呼出し例	776
22.2.6 オブジェクト指向プログラミング機能を使用したデータベースアクセス	776
22.2.6.1 サンプルデータベース	776
22.2.6.2 クラス定義中で表のデータを取り出しデータを更新する	777
22.2.6.3 コネクションをオブジェクトインスタンス単位で利用する	778
22.2.7 プログラムの翻訳・リンク	780
22.2.8 プログラムの実行	780
22.2.8.1 実行環境の構築	780
22.2.8.1.1 実行環境情報の設定	781
22.2.8.1.2 ODBC情報ファイルの作成	782
22.2.8.2 ODBC情報設定ツールの使い方	786
22.2.8.3 ODBC情報ファイルの設定内容の最大長	789
22.2.8.4 連携ソフトウェアおよびハードウェア環境の整備	789
22.2.9 埋込みSQL文のキーワード一覧	790
22.2.10 SQL文と指定可能なホスト変数	797
22.2.11 ODBCで扱うデータとの対応	798
22.2.12 SQLSTATE/SQLCODE/SQLMSG	800
22.2.13 ODBCドライバ使用時の注意事項	801
22.2.13.1 SQL文の文法上の制限事項	801
22.2.13.2 埋込みSQL文の実行時の注意事項	802
22.2.13.3 各ODBCドライバ固有の留意事項	803
22.2.13.4 埋込みSQL文の実行時の定量制限	805
22.2.14 デッドロック出口	806
22.2.14.1 デッドロック出口スケジュールの概要	806
22.2.14.2 注意事項	806
第23章 サーバ・タイプのアプリケーション	808
23.1 バックグラウンド処理	808
23.1.1 画面による入出力操作の抑止方法	808
23.1.2 コマンドプロンプトウィンドウの使用方法	809
23.1.3 サービス配下で動作するプログラム	809
23.2 Web連携	810
23.3 イベントログ	811
23.3.1 実行時メッセージをイベントログに出力する機能	811
23.3.2 利用者定義の情報をイベントログに出力する機能	811
第24章 マルチスレッド	813

24.1 概要	813
24.1.1 特徴	813
24.1.2 機能範囲	813
24.2 マルチスレッドのメリット	813
24.2.1 スレッドとは	813
24.2.2 マルチスレッドプログラムとは	814
24.2.3 マルチスレッドの効果	815
24.3 マルチスレッドプログラムの基本動作	817
24.3.1 実行環境と実行単位	817
24.3.2 マルチスレッドプログラムのデータの扱い	819
24.3.2.1 プログラム定義に宣言されたデータ	820
24.3.2.2 ファクトリオブジェクトとオブジェクトインスタンス	822
24.3.2.3 メソッド定義に宣言されたデータ	825
24.3.2.4 スレッド間共有外部データと外部ファイル	826
24.3.3 プログラムの実行とスレッドモード	827
24.4 スレッド間の資源の共有	828
24.4.1 競合状態	829
24.4.2 資源の共有	830
24.4.2.1 スレッド間共有外部データと外部ファイル	830
24.4.2.2 ファクトリオブジェクト	832
24.4.2.3 オブジェクトインスタンス	834
24.5 基本的な使い方	836
24.5.1 入出力機能の利用	836
24.5.2 リモートデータベースアクセス(ODBC)の利用	839
24.5.3 プリコンパイラの利用によるSymfoware連携	840
24.5.4 DISPLAY文およびACCEPT文の利用	840
24.5.4.1 小入出力機能について	840
24.5.4.2 コマンド行引数および環境変数の操作機能について	843
24.5.4.2.1 コマンド行引数の操作機能	843
24.5.4.2.2 環境変数の操作機能	844
24.5.5 スクリーン機能の利用	844
24.6 少し進んだ使い方	844
24.6.1 入出力機能の利用	844
24.6.1.1 スレッド間共有外部ファイル	845
24.6.1.2 ファクトリオブジェクト内に定義したファイル	848
24.6.1.3 オブジェクト内に定義したファイル	850
24.6.2 リモートデータベースアクセス(ODBC)の利用	853
24.6.2.1 コネクションをスレッド間で共有する	853
24.6.2.2 注意事項	857
24.6.3 CプログラムからCOBOLプログラムをスレッドとして起動する方法	857
24.6.3.1 概要	857
24.6.3.2 起動方法	858
24.6.3.3 パラメタの受渡し方法	858
24.6.3.4 復帰コード(関数値)	858
24.6.3.5 翻訳とリンク	860
24.6.4 スレッド間で実行単位の資源を引き継ぐ方法	863
24.6.4.1 概要	863
24.6.4.2 利用方法	865
24.6.4.3 サブルーチンの使い方	866
24.6.4.3.1 COBOL実行単位ハンドル取得サブルーチン	866
24.6.4.3.2 COBOL実行単位ハンドル設定サブルーチン	867
24.6.4.4 注意事項	868
24.6.4.5 翻訳とリンク	869
24.7 翻訳から実行までの方法	870
24.7.1 翻訳とリンク	870
24.7.1.1 COBOLプログラムだけでDLLを作成する場合	870
24.7.1.2 COBOLプログラムとCプログラムでDLLを作成する場合	871

24.7.2 実行	872
24.7.2.1 実行環境情報の設定	872
24.7.2.2 実行環境情報の指定形式	875
24.8 マルチスレッドプログラムのデバッグ方法	876
24.8.1 TRACE機能	876
24.8.2 CHECK機能	877
24.8.3 COUNT機能	877
24.8.4 対話型デバッガ	878
24.9 スレッド同期制御サブルーチン	878
24.9.1 データロックサブルーチン	878
24.9.1.1 COB_LOCK_DATA	879
24.9.1.2 COB_UNLOCK_DATA	880
24.9.2 オブジェクトロックサブルーチン	880
24.9.2.1 COB_LOCK_OBJECT	881
24.9.2.2 COB_UNLOCK_OBJECT	881
24.9.3 エラーコード	882
24.10 注意事項	883
24.10.1 COM連携機能	883
24.10.1.1 COMのマルチスレッド環境	883
24.10.1.2 COBOLの初期化方法	883
24.10.1.3 COM連携で使用するオブジェクトの解放について	884
24.10.1.4 COMサーバの寿命について	884
24.10.2 オブジェクト指向プログラミング機能	884
24.10.3 簡易アプリ間通信機能	884
24.10.4 印刷機能	885
24.10.5 動的プログラム構造	885
第25章 Unicode	886
25.1 Unicodeサポート概要	886
25.1.1 資源	886
25.1.2 表現形式	887
25.1.3 言語要素	888
25.1.4 コード系の混在	890
25.1.5 資産移行支援	890
25.2 Unicodeアプリケーションの作成	890
25.3 コーディング上の注意点	891
25.3.1 利用者語	891
25.3.2 文字定数	891
25.3.3 項目の再定義	891
25.3.4 転記	892
25.3.5 比較	893
25.3.6 ACCEPT/DISPLAY文	893
25.3.7 COBOLファイル	894
25.3.8 スクリーン操作機能	896
25.4 実行時の注意点	896
25.4.1 メッセージを出力するファイル	896
25.4.2 フォントについて	897
25.5 関連製品連携	897
25.5.1 FORM/MeFt	897
25.5.2 プリコンパイラ	898
25.5.3 ODBC	898
25.5.4 他言語間結合	898
25.5.5 他のファイルシステム	899
25.5.6 通信機能	899
25.5.7 Web連携	900
25.5.8 分散オブジェクト	900
第26章 COM機能	901

26.1 COM機能を利用したアプリケーション	901
26.1.1 一般的なシステム形態	901
26.1.2 COMを利用したアプリケーション形態	902
26.1.3 COM機能を利用するアプリケーションの作成	903
26.1.4 開発環境と資源	903
26.2 COMクライアント機能	905
26.2.1 概要	905
26.2.2 アーリバインドとレイトバインド	906
26.2.2.1 アーリバインドとレイトバインドの違い	906
26.2.2.2 アーリバインドのメリットとデメリット	908
26.2.2.3 バインド方法選択の基準	908
26.2.2.4 .NET言語で作成されたCOMサーバとの連携	908
26.2.3 *COMクラス	909
26.2.3.1 プログラミング例	909
26.2.4 *COMレイトバインドクラス	913
26.2.4.1 *COMレイトバインドクラスのファクトリメソッド	913
26.2.4.2 *COMレイトバインドクラスのオブジェクトメソッド/プロパティ	914
26.2.5 *COMアーリバインドクラス	914
26.2.5.1 *COMアーリバインドクラスのファクトリメソッド	915
26.2.5.2 *COMアーリバインドクラスのオブジェクトメソッド/プロパティ	915
26.2.6 名前付きパラメタ	915
26.2.7 パラメタの省略	915
26.2.8 COM機能の例外処理	916
26.2.8.1 例外の種類	916
26.2.8.2 例外発生時に通知される情報	917
26.2.8.3 例外処理の流れ	917
26.2.8.4 *COM-EXCEPTIONクラスの使用例	918
26.2.9 オブジェクト指定子	920
26.3 COMサーバ機能	921
26.3.1 COMサーバとは	921
26.3.2 COBOLで作成するCOMサーバの特徴	921
26.3.3 COMサーバの作成	922
26.3.4 COMサーバの登録と削除	924
26.3.5 COMサーバの利用	924
26.3.6 例外の通知	924
26.3.7 既存プログラムのCOMサーバ化	927
26.3.8 注意事項	928
26.4 DCOMとMTS	928
26.4.1 MTSの概要	929
26.4.2 MTS環境への登録方法	929
26.4.3 クライアントマシンへのインストール	934
26.4.4 MTSのトランザクション管理	935
26.4.4.1 COMサーバからのトランザクション制御	935
26.4.4.2 オブジェクトコンテキストオブジェクトのメソッド	937
26.4.4.3 COMクライアントからのトランザクション制御	938
26.4.4.4 トランザクションコンテキストオブジェクトのメソッド	940
26.4.4.5 注意事項	941
26.4.5 MTSのトランザクション処理	942
26.4.5.1 トランザクションの基本的な考え方	942
26.4.5.2 トランザクションを制御するための追加処理	942
26.4.5.3 MTSを使ったトランザクションのプログラム例	943
26.4.5.4 トランザクション範囲	947
26.4.5.5 MTS配下での動作上の注意	951
26.5 COM連携で扱うデータ型の対応	953
26.6 COM連携のためのその他の特殊クラス	955
26.6.1 *COM-ARRAYクラス	955
26.6.1.1 *COM-ARRAYクラスのファクトリメソッド	955

26.6.1.2 *COM-ARRAYクラスのオブジェクトメソッド	956
26.6.1.3 *COM-ARRAYの型指定子	956
26.6.1.4 *COM-ARRAYクラスの使用例	957
26.6.1.5 バリエント型データの扱い	959
26.6.1.6 *COM-ARRAYクラスで扱うデータ型	960
26.6.2 *COM-EXCEPTIONクラス	961
26.6.2.1 *COM-EXCEPTIONクラスのメソッド	961
26.6.3 特殊クラスの注意事項	963
26.7 COMサーバおよびCOMクライアントのデバッグ	964
26.7.1 通常のCOBOLプログラムとしてデバッグする場合の変更点について	964
26.7.2 COMクライアントのデバッグ方法	965
26.7.3 COMサーバのデバッグ方法	966
26.7.3.1 デバッグを起動する	966
26.7.3.2 デバッグを行う	966
26.7.4 COMクライアントとCOMサーバを通じたデバッグ方法	967
26.8 開発環境	967
26.8.1 COMサーバ名の設定	967
26.8.2 COMサーバ作成機能の使い方	969
26.9 COM関連の用語の説明	974
第27章 CORBAアプリケーション	976
27.1 CORBAアプリケーションの開発	976
27.1.1 CORBAアプリケーションとは	976
27.1.2 概要	976
27.1.3 プロジェクト管理	977
27.1.3.1 サーバアプリケーションの作成	977
27.1.3.2 クライアントアプリケーションの作成	979
27.1.4 [Interstage]ダイアログ	979
27.1.5 ターゲットオブジェクトファイル	982
27.1.6 注意事項	982
27.2 AADアプリケーション	983
27.2.1 概要	983
27.2.2 AADアプリケーションの作成手順	984
27.2.3 配付ソース生成	986
27.2.3.1 [AAD配付ソース生成]ページ	989
27.2.4 資産転送	989
27.2.4.1 [送信]ダイアログ	991
27.2.4.2 [参照] (送信元ファイル名指定用)ダイアログ	994
27.2.4.3 受信ダイアログ	995
27.2.5 ビルド用JCL/CLISTの生成	998
27.2.5.1 [ビルド用JCL/CLIST生成]ダイアログ	998
27.2.5.2 JCL/CLISTファイル	999
27.2.6 ターゲットビルド	1004
27.2.7 配付ソース対応表示	1005
27.3 IDL-COBOL変換機能	1006
付録A 翻訳オプション	1008
A.1 翻訳オプション一覧	1008
A.2 翻訳オプションの指定形式	1010
A.2.1 AIMLIB (サブスキーマ定義ファイルのフォルダの指定)	1012
A.2.2 ALPHAL (英小文字の扱い)	1013
A.2.3 ASCOMP5 (2進項目の解釈の指定)	1013
A.2.4 BINARY (2進項目の扱い)	1014
A.2.5 CHECK (CHECK機能の使用の可否)	1015
A.2.6 CONF (規格の違いによるメッセージの出力の可否)	1016
A.2.7 COPY (登録集原文の表示)	1017
A.2.8 COUNT (COUNT機能の使用の可否)	1017
A.2.9 CREATE (創成ファイルの指定)	1018

A.2.10 CURRENCY (通貨編集用文字の扱い)	1018
A.2.11 DLOAD (プログラム構造の指定)	1018
A.2.12 DUPCHAR (重複文字の扱い)	1019
A.2.13 EQUALS (SORT文での同一キーデータの処理方法)	1019
A.2.14 FILEEXT (ファイル定義体ファイルの拡張子の指定)	1019
A.2.15 FILELIB (ファイル定義体ファイルのフォルダの指定)	1020
A.2.16 FLAG (診断メッセージのレベル)	1020
A.2.17 FLAGSW (COBOL文法の言語要素に対しての指摘メッセージ表示の可否)	1021
A.2.18 FORMEXT (画面帳票定義体ファイルの拡張子の指定)	1022
A.2.19 FORMLIB (画面帳票定義体ファイルのフォルダの指定)	1022
A.2.20 GEN (FCOMおよびUWAの表示)	1023
A.2.21 INITVALUE (作業場所節でのVALUE句なし項目の扱い)	1023
A.2.22 LANGLVL (ANSI COBOL規格の指定)	1023
A.2.23 LIB (登録集ファイルのフォルダの指定)	1024
A.2.24 LIBEXT (登録集ファイルの拡張子の指定)	1024
A.2.25 LINECOUNT (翻訳リストの1ページあたりの行数)	1025
A.2.26 LINESIZE (翻訳リストの1行あたりの文字数)	1025
A.2.27 LIST (目的プログラムリストの出力の可否)	1026
A.2.28 MAIN (主プログラム/副プログラムの指定)	1026
A.2.29 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の出力の可否)	1027
A.2.30 MESSAGE (オプション情報リスト、翻訳単位統計情報リストの出力の可否)	1027
A.2.31 MODE (ACCEPT文の動作の指定)	1028
A.2.32 NAME (翻訳単位ごとのオブジェクトファイルの出力の可否)	1028
A.2.33 NCW (日本語利用者語の文字集合の指定)	1028
A.2.34 NSPCOMP (日本語空白の比較方法の指定)	1029
A.2.35 NUMBER (ソースプログラムの一連番号領域の指定)	1030
A.2.36 OBJECT (目的プログラムの出力の可否)	1031
A.2.37 OPTIMIZE (広域最適化の扱い)	1031
A.2.38 PRINT (各種翻訳リストの出力の可否および出力先の指定)	1032
A.2.39 QUOTE/APOST (表意定数QUOTEの扱い)	1032
A.2.40 RCS (実行時コード系の指定)	1032
A.2.41 REP (リポジットファイルの入出力先フォルダの指定)	1033
A.2.42 REPIN (リポジットファイルの入力先フォルダの指定)	1033
A.2.43 RSV (予約語の種類)	1034
A.2.44 SAI (ソース解析情報ファイルの出力の可否)	1035
A.2.45 SCS (ソースファイルのコード系)	1035
A.2.46 SDS (符号付き10進項目の符号の整形の可否)	1036
A.2.47 SHREXT (マルチスレッドプログラムの外部属性に関する扱い)	1036
A.2.48 SMSIZE (PowerSORTが使用するメモリ容量を指定)	1036
A.2.49 SOURCE (ソースプログラムリストの出力の可否)	1037
A.2.50 SQLGRP (SQLのホスト変数定義の拡張)	1037
A.2.51 SRF (正書法の種類)	1038
A.2.52 SSIN (ACCEPT文のデータの入力先)	1038
A.2.53 SSOUT (DISPLAY文のデータの出力先)	1039
A.2.54 STD1 (英数字の文字の大小順序の指定)	1040
A.2.55 TAB (タブの扱い)	1040
A.2.56 TEST (対話型デバグおよび診断機能の使用の可否)	1040
A.2.57 THREAD (マルチスレッドプログラム作成の指定)	1041
A.2.58 TRACE (TRACE機能の使用の可否)	1041
A.2.59 TRUNC (桁落とし処理の可否)	1042
A.2.60 XREF (相互参照リストの出力の可否)	1043
A.2.61 ZWB (符号付き外部10進項目と英数字項目の比較)	1043
A.3 プログラム定義にだけ指定可能な翻訳オプション	1044
A.4 メソッド原型定義と分離されたメソッド定義間での翻訳オプション	1044
付録B 入出力状態一覧	1046
付録C 広域最適化	1050

C.1 最適化の項目.....	1050
C.2 共通式の除去.....	1050
C.3 不変式の移動.....	1051
C.4 誘導変数の最適化.....	1051
C.5 PERFORM文の最適化.....	1051
C.6 隣接転記の統合.....	1052
C.7 無駄な代入の除去.....	1052
C.8 広域最適化での注意事項.....	1052
付録D 組込み関数の使用.....	1054
D.1 関数の型と記述の関係.....	1054
D.2 引数の型によって決定される関数の型.....	1055
D.3 CURRENT-DATE関数を利用した西暦の取得.....	1056
D.4 任意の基準日からの通日計算.....	1057
D.5 RANDOM関数を利用した疑似乱数列の生成.....	1058
D.6 組込み関数一覧.....	1058
付録E 特殊な定数の書き方.....	1061
E.1 プログラム名定数.....	1061
E.2 原文名定数.....	1061
E.3 ファイル識別名定数.....	1061
E.4 外部名を指定するための定数.....	1062
付録F OSIV系システムとの機能比較.....	1063
F.1 機能比較一覧.....	1063
F.2 プログラムの動作確認における注意事項.....	1067
付録G COBOLが提供するサブルーチン.....	1070
G.1 COBOLプログラムから呼び出すサブルーチン.....	1070
G.1.1 ウィンドウハンドル獲得サブルーチン.....	1070
G.1.2 インスタンスハンドル獲得サブルーチン.....	1071
G.1.3 イベントログ出力サブルーチン.....	1072
G.1.4 プロセスID取得サブルーチン.....	1074
G.1.5 スレッドID取得サブルーチン.....	1074
G.1.6 デッドロック出口スケジュールサブルーチン.....	1075
G.1.7 メモリ割当てサブルーチン.....	1075
G.1.8 プロセス終了サブルーチン.....	1082
G.2 他言語連携で使用するサブルーチン.....	1084
G.2.1 実行単位の開始サブルーチン.....	1084
G.2.2 実行単位の終了サブルーチン.....	1085
G.2.3 実行環境の閉鎖サブルーチン.....	1086
付録H オブジェクト指向と従来機能の組合せ.....	1088
H.1 クラス定義で使用できない機能.....	1088
H.2 分離翻訳されるメソッド定義で使用できない機能.....	1089
付録I その他のコマンドの形式.....	1090
I.1 DUMPBINコマンド.....	1090
I.2 INSDBINFコマンド.....	1091
付録J 移行上の注意事項.....	1095
J.1 ソースプログラムの互換性について.....	1095
J.2 プロジェクトファイルの互換性について.....	1096
付録K COBOL G移行支援オプション.....	1098
付録L XMLの利用.....	1100
L.1 XMLの概要.....	1100
L.2 NetCOBOL XMLオプションの利用.....	1102

付録M 文字コード系.....	1103
M.1 文字コードの概説.....	1103
M.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.....	1103
M.1.2 文字種の混在方式による分類.....	1104
M.2 文字コード変換.....	1105
M.3 シフトJISへのコード変換時の問題.....	1107
M.4 他システムからの移行上の注意.....	1110
M.4.1 日本語空白と英数字空白の文字コード.....	1110
M.4.2 JIS非漢字の負号について.....	1112
付録N セキュリティ.....	1114
N.1 資源の保護.....	1114
N.2 アプリケーション作成のための指針.....	1114
N.3 インターネット接続対象外の通信機能.....	1115
N.4 Web連携機能のセキュリティ.....	1115
N.5 リモート開発デバッグ.....	1115
N.6 リモート開発機能.....	1115
付録O COBOLプログラムの作成技法.....	1116
O.1 効率のよいプログラム.....	1116
O.1.1 一般的な注意.....	1116
O.1.2 データ項目の属性の選択.....	1116
O.1.3 数字転記・数字比較・算術演算.....	1117
O.1.4 英数字転記・英数字比較.....	1118
O.1.5 入出力.....	1118
O.1.6 プログラム間連絡.....	1119
O.1.7 デバッグ.....	1119
O.2 数字項目の標準規則.....	1119
O.2.1 10進項目.....	1119
O.2.2 2進項目.....	1120
O.2.3 浮動小数点項目.....	1121
O.2.4 数字項目の注意事項.....	1121
O.3 注意事項.....	1121
索引.....	1122

第1章 NetCOBOLの概要

本章では、本製品の機能および動作環境について説明します。本製品をはじめてお使いになる方は、必ずお読みください。

1.1 NetCOBOLの機能

ここでは、NetCOBOLが持つCOBOLの機能およびNetCOBOLが提供する各種ユーティリティについて説明します。

1.1.1 COBOLの機能

NetCOBOLは、以下に示すCOBOLの機能を持っています。

これらの機能を使用するためのCOBOLの文の書き方は、“COBOL文法書”に規定されています。

- ・ 中核機能
- ・ 順ファイル機能
- ・ 相対ファイル機能
- ・ 索引ファイル機能
- ・ プログラム間連絡機能
- ・ 整列併合機能
- ・ 原始文操作機能
- ・ 表示ファイル機能
- ・ 組込み関数機能
- ・ 浮動小数点数
- ・ スクリーン操作機能
- ・ コマンド行引数の操作機能
- ・ 環境変数の操作機能
- ・ 報告書作成機能
- ・ データベース(SQL)
- ・ システムプログラム記述向け(SD)機能
- ・ オブジェクト指向プログラミング機能
- ・ 分散オブジェクト機能(COM機能/CORBAアプリケーション)

NetCOBOLは、サーバサイドアプリケーション(注)で有効となる以下の機能を提供しています。

- ・ マルチスレッド機能
- ・ DCOM機能を利用したCOMサーバ機能
- ・ COBOL Webサブルーチン
- ・ バッチタイプアプリケーションの開発・運用

注：サーバセントリックな環境で運用するアプリケーションを示します。たとえば、Webサーバアプリケーションがこれに該当します。

1.1.2 NetCOBOLが提供するプログラムおよびユーティリティ

NetCOBOLでは、プログラム開発を行うために、以下に示すプログラムおよびユーティリティを提供しています。

名称	使用目的
COBOLコンパイラ	COBOLを使って記述したプログラムの翻訳
COBOLランタイムシステム	COBOLアプリケーションの実行
COBOL対話型デバッガ	COBOLアプリケーションのデバッグ
プロジェクトマネージャ	COBOLアプリケーションの開発支援
COBOLファイルユーティリティ	COBOLファイルの処理
表示ファイル単体テスト	表示ファイル単体テストダイアログの起動
WINCOB コマンド	COBOLコンパイラの起動
WINLINK コマンド	リンカの起動
WINEXEC コマンド	COBOLアプリケーションの起動
簡易アプリ間通信	アプリケーション間でのデータのやりとり
プロジェクトブラウザ (注)	プログラム構造の解析
クラスブラウザ (注)	クラスライブラリの検索
実行環境設定ツール	実行用の初期化ファイルの編集
COBOL診断機能	COBOLアプリケーションのデバッグ

注：オブジェクト指向プログラムを開発する場合に使用します。

COBOLコンパイラ

COBOLコンパイラは、COBOLソースプログラムを翻訳し、目的プログラムを生成します。COBOLコンパイラは、以下のサービス機能を提供しています。

- ・ 翻訳リストの出力
- ・ 規格仕様のチェック
- ・ 広域最適化
- ・ 翻訳時メッセージとエディタ間のエラージャンプ
- ・ FORM(画面・帳票定義)との連携

これらの機能は、COBOLソースプログラムを翻訳するときに、翻訳オプションによって指示します。

COBOLランタイムシステム

COBOLランタイムシステムは、COBOLを使って作成したアプリケーションプログラム(以降、COBOLアプリケーションと略します)を実行するときに呼び出され、動作します。

COBOL対話型デバッガ

COBOL対話型デバッガは、COBOLアプリケーションをデバッグするときに使用します。対話型デバッガを使用すると、ウィンドウを使った簡単な操作でデバッグを行うことができます。COBOL対話型デバッガには、次の機能があります。

- ・ プログラムの実行の中断・再開
- ・ データ域の値の参照・変更
- ・ データ域の値の変更時中断
- ・ プログラムの実行経路の表示
- ・ デバッグ操作の再現

プロジェクトマネージャ(旧称 PROGRAMMING-STAFF)

プロジェクトマネージャでは、COBOLアプリケーションの開発で各種操作(プログラムの編集・翻訳・リンク・実行・デバッグ・保守)を行うための、簡易操作ビューを提供しています。プロジェクトマネージャを使ってプログラム開発を行うことにより、プログラム開発に必要な各種操作を、ウィンドウの切替えなしにスムーズに行うことができます。

COBOLファイルユーティリティ

COBOLファイルシステムが使用できるファイルの処理を、COBOLのアプリケーションを介することなく、ユーティリティのコマンドによって行うためのものです。

表示ファイル単体テスト

OSIV系システムのアプリケーションの開発を本システム上で行う場合に、表示ファイル機能を使った対話型処理を行うアプリケーションの単体テストを行うための機能です。

WINCOBコマンド/WINLINKコマンド/WINEXECコマンド

WINCOBコマンド、WINLINKコマンドおよびWINEXECコマンドは、COBOLコンパイラ、リンカおよびCOBOLアプリケーションを起動するための簡易操作ビューを提供しています。これらのコマンドを使用することにより、COBOLコンパイラ、リンカおよびCOBOLアプリケーションの起動を、ウィンドウを使った簡単な操作で行うことができます。

簡易アプリ間通信(サーバ操作ウィンドウ/クライアントサーバウィンドウ)

簡易アプリ間通信では、サーバを介して利用者プログラム間でメッセージのやりとりを行う手段を提供しています。

プロジェクトブラウザ

プロジェクトに含まれるプログラム、クラスおよびメソッド間の論理的な関係を表示します。プロジェクトブラウザでは、以下の情報を表示することができます。

- ・ プログラム・メソッドとプログラム間の呼出し関係
- ・ プログラム・クラス・メソッドとクラスの参照関係
- ・ クラス間の継承関係
- ・ クラス・メソッドのインタフェース

クラスブラウザ

COBOL開発環境に登録されているすべてのクラスを管理するユーティリティです。クラス情報の表示および検索を行うことができます。クラスブラウザでは、以下の情報を表示することができます。

- ・ クラス間の継承関係
- ・ クラス・メソッドのインタフェース

実行環境設定ツール

実行環境設定ツールは、COBOLアプリケーションを実行するときに使用する実行用の初期化ファイルを編集するためのものです。

COBOL診断機能

COBOL診断機能は、COBOLアプリケーションでアプリケーションエラーやUレベルの実行時メッセージが発生した場合に、診断レポートおよびダンプを出力します。診断レポートには、以下の情報を出力します。

- ・ エラーの種別
- ・ エラー発生箇所
- ・ エラー発生箇所への呼出経路
- ・ システムの種別とバージョン情報
- ・ コマンドライン

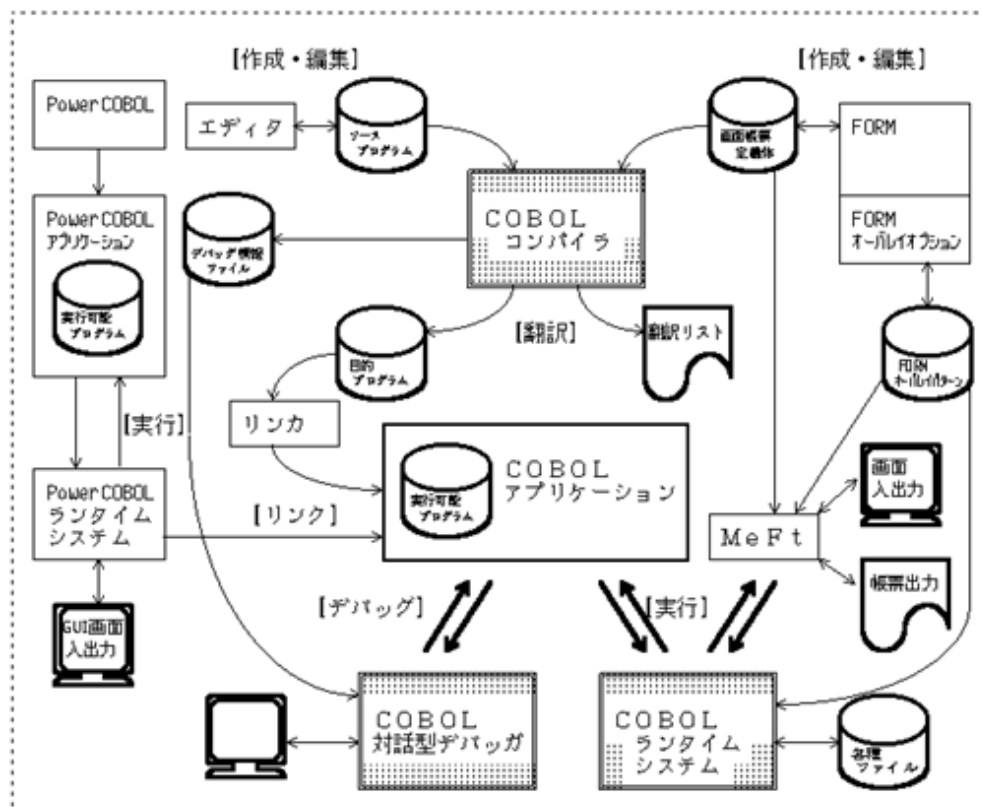
- ・ プロセスの環境変数
- ・ COBOL実行環境情報
- ・ システム上のプロセス一覧
- ・ プロセス内のモジュール一覧
- ・ プロセス内のスレッド情報
- ・ スレッドごとのスタック情報

また、COBOL診断機能は、COBOLアプリケーションでランタイムシステム領域が破壊された場合に、COBOLランタイムシステムと連携してデバッグすることもできます。COBOL診断機能は、以下の情報出力を提供しています。

- ・ 書き込まれた領域の内容
- ・ 書き込みを行ったプログラムおよび命令
- ・ 書き込みを行ったプログラムへの呼出経路

1.2 開発環境

NetCOBOLを利用したプログラムの開発環境の概要を以下に示します。



1.2.1 環境変数の設定

NetCOBOLの機能を使用するために必要な環境変数は、以下の方法で設定します。

- ・ SETコマンドを使って設定する。
- ・ コントロールパネルのシステムで設定する。

SETコマンドによる設定方法については、“[5.3.2.2 SETコマンドで設定する方法](#)”を、コントロールパネルから設定する方法については、Windowsのヘルプを参照してください。

設定時	環境変数名	設定する内容	条件	
共通	PATH	NetCOBOLのインストール先フォルダ	NetCOBOLを使用する場合	必須
		MeFtのインストール先フォルダ	画面帳票定義体を使用する場合	必須
	TMP	作業用フォルダ	プロジェクトマネージャを使用する場合	必須
	TEMP	作業用フォルダ		必須
翻訳	SMED_SUFFIX	画面帳票定義体の拡張子(任意の文字列)	画面帳票定義体の拡張子を変更する場合	任意
	FORMLIB	画面帳票定義体ファイルのフォルダ(注1)	画面帳票定義体を使用する場合	任意
	FFD_SUFFIX	ファイル定義体の拡張子(任意の文字列)	ファイル定義体の拡張子を変更する場合	任意
	FILELIB	ファイル定義体ファイルのフォルダ	ファイル定義体を使用する場合	任意
	COB_COBCOPY	登録集ファイルのフォルダ	登録集ファイルを使用する場合	任意
	COB_登録集名(注2)	IN/OF指定の登録集ファイルのフォルダ	IN/OF指定の登録集ファイルを使用する場合	任意
	COB_LIBSUFFIX	登録集ファイルの拡張子(任意の文字列)	登録集ファイルの拡張子を変更する場合	任意
	COB_REPIN	リポジトリファイルの入力フォルダ	リポジトリファイルを使用する場合	任意
	COB_AIMLIB	サブスキーマ定義ファイルのフォルダ	サブスキーマ定義ファイルを使用する場合	任意
	COB_OPTIONS	翻訳コマンドのオプション	翻訳コマンドのオプションを指定する場合	任意
リンク	LIB	NetCOBOLのインストール先フォルダ	リンクを行う場合	必須
		リンク時に結合するファイルのフォルダ		必須
	TMP	リンク時に使用する作業用フォルダ		必須
実行	BSORT_TMPDIR	作業用フォルダ	整列併合機能を使用する場合	任意
			COBOLファイルユーティリティを使用する場合	任意
	MEFTDIR	プリンタ情報ファイルのフォルダ	画面帳票定義体を使用する場合	任意
	TEMP	作業用フォルダ	整列併合機能を使用する場合	必須
			COBOLファイルユーティリティを使用する場合	必須

- ・ 注1
画面帳票定義体ファイルの絶対パス名は、シフトJISで256バイト以内の長さでなければなりません。
- ・ 注2
 - ー 環境変数名には、COPY文に指定した登録集名を用います。登録集名には“XMDLIB”または“XFDLIB”は使用できません。
 - ー 環境変数名がNetCOBOLでサポートする環境変数名と重なる名前を指定してはいけません。
 - ー 環境変数COB_登録集名には、フォルダは1つしか指定できません。



例

COPY文の記述例: COPY TEST OF TESTLIB1

COB_TESTLIB1=C:¥COPYLIB

フォルダを設定する環境変数

以下の環境変数には、複数のフォルダが指定できます。複数のフォルダを指定した場合、指定した順序でフォルダを検索します。

- 環境変数FORMLIB
- 環境変数FILELIB
- 環境変数COB_COBCOPY
- 環境変数COB_REPIN
- 環境変数COB_AIMLIB



例

```
FORMLIB=C:¥FORM;"C:¥Program Files¥NetCOBOL¥FORM"
```



注意

フォルダ名に空白を含む場合、フォルダ名を二重引用符(")で囲む必要があります。

拡張子を設定する環境変数

以下の環境変数には、複数の拡張子が指定できます。複数の拡張子を指定した場合、指定した順序でファイルを検索します。また、“None”を指定した場合、拡張子なしのファイルを検索します。

- 環境変数SMED_SUFFIX
- 環境変数FFD_SUFFIX
- 環境変数COB_LIBSUFFIX



例

```
SMED_SUFFIX=SMD, None
```

これらの環境変数を指定しない場合の拡張子の省略値は、以下を参照してください。

- “[A.2.18 FORMEXT](#) (画面帳票定義体ファイルの拡張子の指定)”
- “[A.2.14 FILEEXT](#) (ファイル定義体ファイルの拡張子の指定)”
- “[A.2.24 LIBEXT](#) (登録集ファイルの拡張子の指定)”

翻訳オプションを設定する環境変数

環境変数COB_OPTIONSには、翻訳コマンドのオプションを指定します。翻訳コマンドのオプションについては、“[3.8.1 オプション一覧](#)”を参照してください。また、翻訳オプションの指定方法による優先順位については、“[A.2 翻訳オプションの指定形式](#)”の“翻訳オプションの指定形式と優先順位”を参照してください。



例

```
COB_OPTIONS=-WC,"OPTIMIZE,SRF(FREE)" -iC:¥TEST¥DEFAULT.CBI
```

ただし、以下のオプションは環境変数COB_OPTIONSに指定できません。指定した場合、無効とみなして翻訳処理を続行します。

-A、-I、-P、-R、-f、-m、-dd、-do、-dp、-dr、-ds

1.2.2 関連製品

下表に、NetCOBOLを使ってプログラム開発を行うときに使用する関連製品を示します。

製品名	機能
PowerCOBOL	プログラムが使用するGUI画面の定義および実行
FORM	プログラムが出力する画面や帳票の定義および帳票設計ツール (PowerFORM) の提供
FORMオーバレイオプション	プログラムが出力するオーバレイパターンの定義
MeFt	画面や帳票の入出力
MeFt/Web	Webブラウザを使用した画面や帳票の入出力
ファウンデーションクラス	COBOLのオブジェクト指向プログラムから利用するクラスライブラリ

以下に各製品の概要を説明します。

PowerCOBOL

PowerCOBOLは、GUI画面を持つCOBOLアプリケーションを設計、実行するために使用します。画面上にボタンなどのアイテムを配置し、イベント発生時の処理をCOBOLで記述することができます。なお、PowerCOBOLは、NetCOBOLシリーズに同梱されるコンポーネントです。

FORM

FORMは、COBOLプログラムが表示・印刷する画面や帳票を設計するために使用します。利用者は、FORMを利用して、対話的に画面や帳票のレイアウトなどをデザインすることができます。また、帳票設計ツール(PowerFORM)では、Windowsの各OSのスタイルガイドに準拠し、ウィザード機能を装備してユーザへの使いやすさを配慮しています。さらに、フルカラービットマップデータをサポートすることにより、Windows機能を最大限に生かした機能を提供しています。

FORMオーバレイオプション

FORMオーバレイオプションは、COBOLプログラムが印刷するオーバレイパターンを設計するために使用するFORMのオプション製品です。利用者は、対話的にオーバレイパターンのレイアウトなどをデザインすることができます。

MeFt

MeFtは、画面や帳票の入出力を行うプログラムを実行するときに暗黙的に使用されます。MeFtは、プログラムが発行する画面や帳票の入出力要求に対して、フォーマット編集を行います。

MeFt/Web

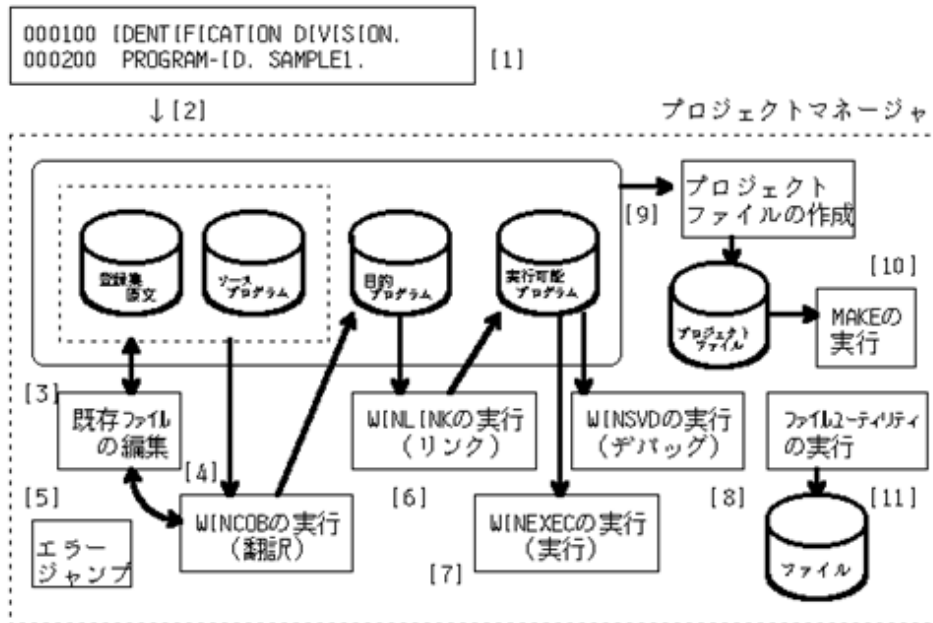
MeFt/Webは、Webサーバ上で動作するMeFt経由の画面や帳票の入出力を行うプログラムのディスプレイ装置やプリンタ装置に対する入出力を、Webブラウザに対して行うことができます。なお、MeFt/Webは、NetCOBOLシリーズに同梱されるコンポーネントです。

ファウンデーションクラス

ファウンデーションクラスは、COBOLのオブジェクト指向プログラムから利用できるクラスライブラリです。帳票印刷クラスライブラリ、コレクションクラスライブラリ、Javaクラスを呼び出すCOBOLクラスを生成するJアダプタクラスジェネレータおよびDBアクセスクラスライブラリを提供しています。なお、ファウンデーションクラスは、NetCOBOLシリーズに同梱されるコンポーネントです。

1.3 プログラム開発の手順

プロジェクトマネージャを使ったプログラム開発の標準的な手順を以下に示します。



- [1]
各種エディタを使って、COBOLソースプログラムを記述します。
- [2]
プロジェクトマネージャを起動します。
- [3]
エディタを使用して、プログラムを編集します。
- [4]
WINCOBを実行し、プログラムの翻訳を行います。[参照]“[第3章 プログラムを翻訳する](#)”
- [5]
翻訳エラーが発生した場合、エラージャンプ機能を使って、エラーの発生したCOBOLの文を修正します。[参照]“[3.6 翻訳エラーの修正](#)”
- [6]
WINLINKを実行し、プログラムのリンクを行います。[参照]“[第4章 プログラムをリンクする](#)”
- [7]
WINEXECを実行し、プログラムを実行します。[参照]“[第5章 プログラムを実行する](#)”
- [8]
プログラムが期待どおり動かない場合やプログラムの単体テストを行う場合、対話型デバッガを使ってデバッグを行います。[参照]“[第20章 対話型デバッガの使い方](#)”
- [9]
1つ以上のファイルからアプリケーションを生成する場合、プロジェクトという形でファイルの依存関係を管理することのできる、プロジェクト管理機能を使ってアプリケーションの開発および保守を行います。[参照]“[第6章 プロジェクト管理機能](#)”
- [10]
プロジェクト管理を行っているアプリケーションを構成するプログラムを修正した場合、アプリケーションを作り直す必要があります。そのような場合、ビルド・リビルドを実行します。[参照]“[6.4.5 プロジェクトのビルド・リビルド](#)”
- [11]
COBOLファイルの処理を行いたい場合、ファイルユーティリティを実行します。[参照]“[7.8 COBOLファイルユーティリティ](#)”



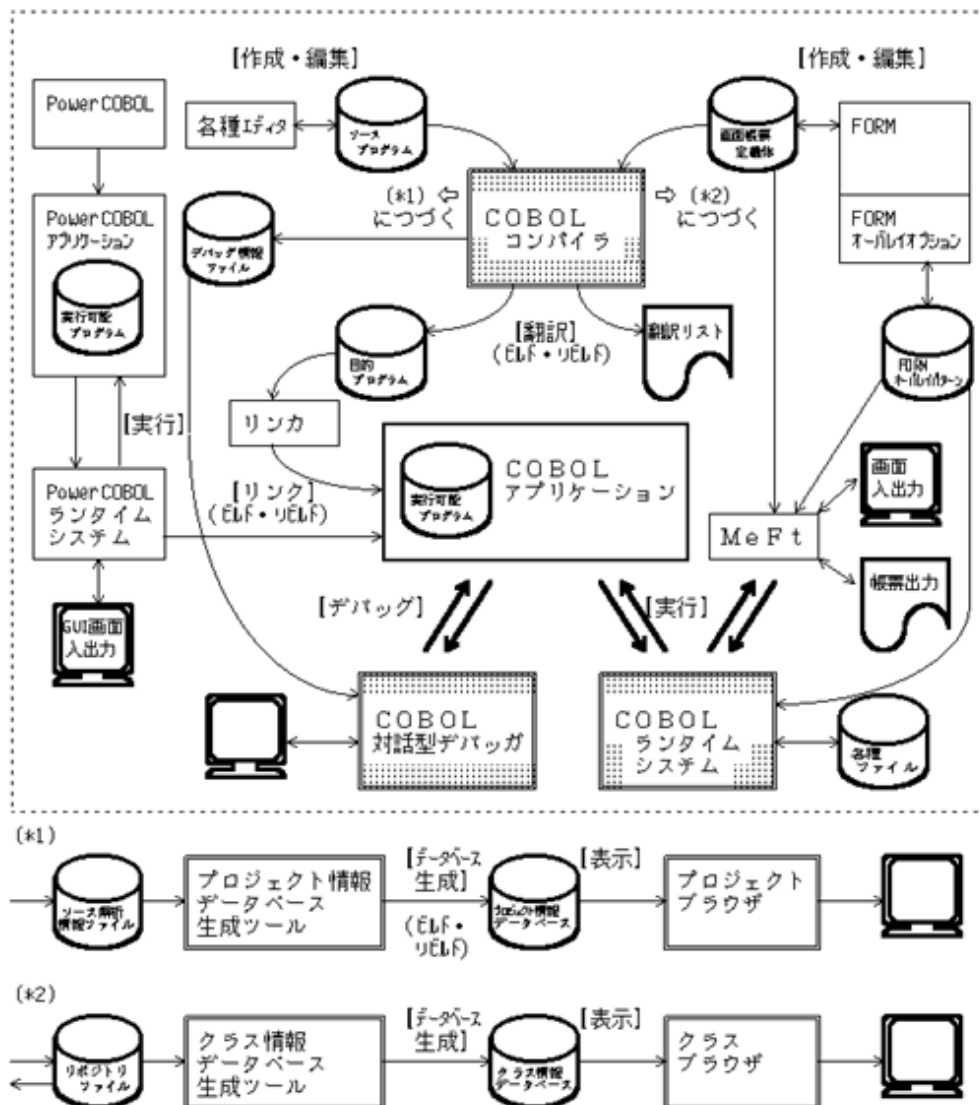
参照

プロジェクトマネージャを使わずに、コマンドプロンプト上から翻訳、リンクを行うこともできます。これらの方法については、以下を参照してください。

- “3.7 翻訳コマンド”
- “3.8 翻訳コマンドのオプション”
- “4.7 リンクコマンドを使ったリンク操作”
- “4.8 リンクコマンドの形式”

1.4 オブジェクト指向プログラムの開発環境

NetCOBOLを利用してオブジェクト指向プログラミングを行う場合の開発環境の概要を以下に示します。



1.5 オブジェクト指向プログラム開発の手順

プロジェクトマネージャを使った、オブジェクト指向プログラム開発の標準的な手順を以下に示します。

- プロジェクトマネージャを使わずに、コマンドプロンプト上から翻訳、リンクを行うこともできます。これらの方法については、以下を参照してください。

- “[3.7 翻訳コマンド](#)”

- “[3.8 翻訳コマンドのオプション](#)”

- “[4.7 リンクコマンドを使ったリンク操作](#)”

- “[4.8 リンクコマンドの形式](#)”

.....

第2章 プログラムを作成・編集する

本章では、プログラムの作成と編集、登録集原文の作成、プログラムの形式および翻訳指示文について説明します。

2.1 プログラムの作成

COBOLソースプログラムおよび登録集原文は、各種エディタを使用して作成します。

ここでは、COBOLソースプログラムおよび登録集原文の作成について説明します。

2.1.1 COBOLソースプログラムの作成/編集

ここでは、COBOLソースプログラムを作成・編集する方法を説明します。

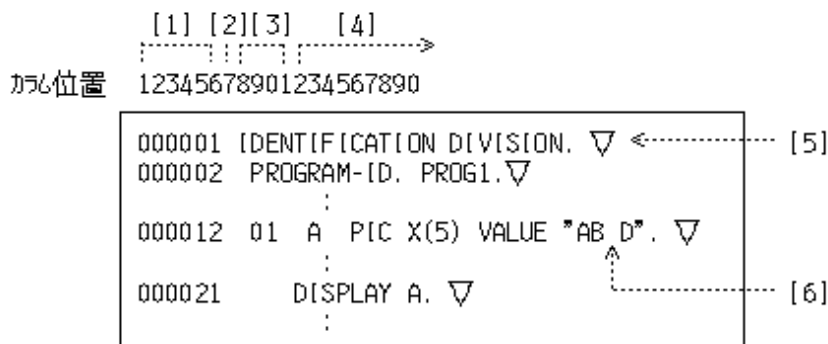
エディタの起動

エディタを起動し、編集画面を表示してください。

作成/編集

COBOLソースプログラムの1行の形式は、正書法としてCOBOLの文法で規定しています。エディタを使ってCOBOLソースプログラムを作成する場合、行番号やCOBOLの文および手続き名など、正書法に従った位置から記述する必要があります。正書法の概要については“[2.1.3 プログラムの形式](#)”を、詳細は“COBOL文法書”を参照してください。

正書法が可変形式または固定形式の場合、行番号やCOBOLの文を、エディタの編集画面に以下の形式で入力してください。



[1] 一連番号領域(カラム1~6)

一連番号領域には、行番号を記述します。行番号は、省略することもできます。

[2] 標識領域(カラム7)

標識領域は、行を継続する場合や行を注記行にするために使用します。行を継続しない場合や注記行にしない場合には、必ず空白を入力してください。

[3] A領域(カラム8～11)

通常、COBOLの部、節、段落およびプログラムの終わり見出しなどは、この領域から記述します。レベル番号が77や01のデータ項目もこの領域から記述します。

[4] B領域(カラム12以降)

通常、COBOLの文およびレベル番号が77や01でないデータ項目などは、この領域から記述します。

[5] 改行文字(行の終わり)

各行の終わりには、改行文字を入力します。

[6] TAB文字

COBOLソースプログラムの文字定数には、TAB文字が使用できます。
TAB文字は、文字定数中で1バイトを占めます。

プログラムの格納とエディタの終了

ソースプログラムの作成・編集が終了したら、作成・編集したプログラムをファイルに格納し、エディタを終了してください。通常、COBOLソースプログラムを格納するファイルの名前には、拡張子COB、CBL、COBOLを付加します。拡張子COB、CBL、COBOLを付加することにより、翻訳を行うときのファイル名の指定が簡単になります。

2.1.2 登録集原文の作成/編集

COPY文によってCOBOLソースプログラムに取り込む登録集原文は、COBOLソースプログラムを作成するときと同様に、各種エディタを利用して作成します。登録集原文の正書法の形式は、その登録集原文を取り込むCOBOLソースプログラムの形式と異なってもかまいません。ただし、1つのプログラムで複数の登録集原文を取り込む場合には、すべての登録集原文の正書法の形式を同一にする必要があります。登録集原文の正書法の形式は、COBOLソースプログラムと同様に翻訳オプションで指定します。

通常、登録集原文を格納するファイルの名前には、拡張子CBL、COB、COBOLを付加します。翻訳オプションLIBEXTまたは環境変数COB_LIBSUFFIXを指定することにより、拡張子を任意の文字列にすることもできます。[参照]“[1.2.1 環境変数の設定](#)”、“[A.2.24 LIBEXT\(登録集ファイルの拡張子の指定\)](#)”



参考

画面帳票定義体およびファイル定義体は、正書法をどの形式にしても利用できます。

2.1.3 プログラムの形式

COBOLソースプログラムの各行は、改行文字で区切ります。COBOLソースプログラムを作成する場合、その1行の形式は、正書法で定める規則に従って記述しなければなりません。正書法には、固定形式、可変形式および自由形式の3つの形式があり、固定形式または自由形式を使用する場合は、翻訳時に翻訳オプションSRFで指定する必要があります。[参照]“[A.2.51 SRF\(正書法の種類\)](#)”

以下にそれぞれの形式について説明します。なお、各行の区切りの改行文字は、行の一部とはみなされません。

固定形式

固定形式では、COBOLソースプログラムの1行の長さを80バイトの固定として記述します。

(カラム位置)

1	6	7	8	12	72	73	80
一連番号領域	*	A 領域	B 領域			**	

* : 標識領域

** : プログラム識別番号領域

可変形式

可変形式では、COBOLソースプログラムの1行の長さを、251バイト以下の任意のバイト数で記述します。

(カラム位置)

1	6	7	8	12	≤ 251
一連番号領域	*	A 領域	B 領域		

* : 標識領域

自由形式

自由形式では、注記、デバッグ行および継続のための特別な規則があることを除いて、COBOLソースプログラムは、行のどの文字位置からでも記述することができます。

1行の文字位置の数は、行ごとに最小0から最大251の範囲で変更することができます。

(カラム位置)

1	≤ 251

自由形式で記述した例題プログラムをサンプルプログラムとして提供していますので、参考になさってください。

2.1.4 翻訳指示文

翻訳指示文は、翻訳オプションをCOBOLコンパイラに指示するために使います。通常、翻訳オプションは[翻訳オプション]ダイアログから指定しますが、ソースプログラム中に記述して指定することもできます。[参照]“3.5 [翻訳オプション]ダイアログ”

翻訳指示文の記述形式

```
@OPTIONS [翻訳オプション[, 翻訳オプション] ... ]
```

- @OPTIONSは、8カラム目から記述します。
- @OPTIONSと翻訳オプションの間には、1つ以上の空白が必要です。
- 各翻訳オプションの間は、1つのコンマ(,)で区切る必要があります。
- 翻訳指示文は、翻訳単位の先頭に記述します。指定する翻訳オプションは、その翻訳指示文の翻訳単位にだけ適用されます。



例

翻訳指示文の記述例

```
000100 @OPTIONS MAIN, APOST  
000200 IDENTIFICATION DIVISION.  
:
```



注意

- @OPTIONS翻訳指示文の分離符としてTAB文字を用いることはできません。
- 翻訳オプションによっては、翻訳指示文に指定できないものもあります。詳細は、“A.2 翻訳オプションの指定形式”を参照してください。

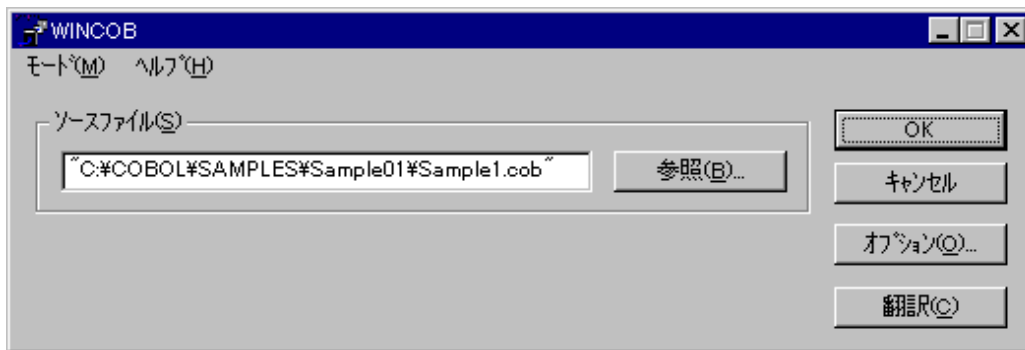
第3章 プログラムを翻訳する

本章では、まず、サンプルプログラムの翻訳手順について説明し、翻訳に必要な資源、手順、使用するウィンドウの使い方など翻訳時に必要な作業について説明します。

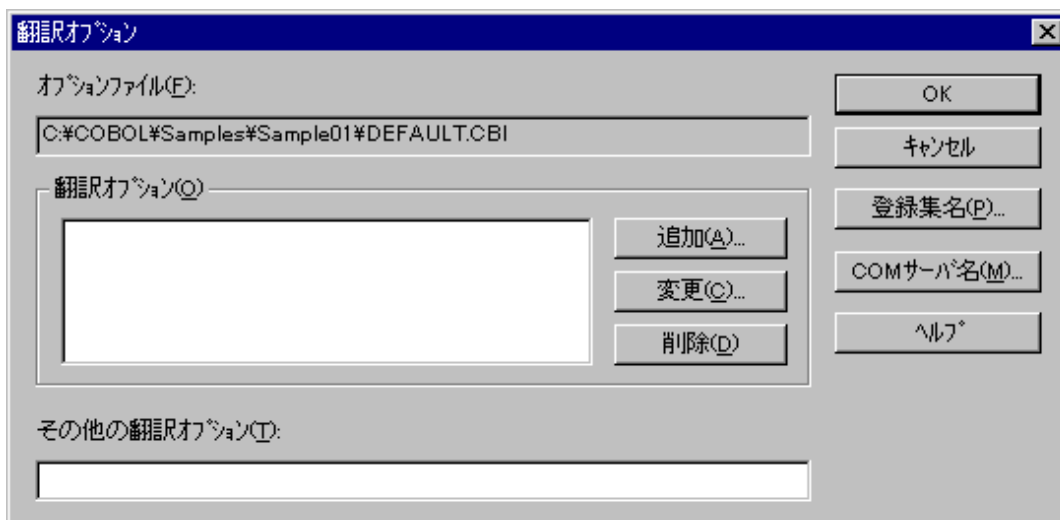
3.1 サンプルプログラムの翻訳

まず、実際にサンプルプログラムの翻訳を行ってみましょう。サンプルプログラムは、NetCOBOLのインストール先フォルダの下のSAMPLESというフォルダの中に、例題ごとに格納されています。ここでは、WINCOBを使って、例題1のプログラムを翻訳します。なお、以下の説明では、サンプルプログラムの格納先ファイルをC:\COBOL\SAMPLES\SAMPLE01\SAMPLE1.COBとします。

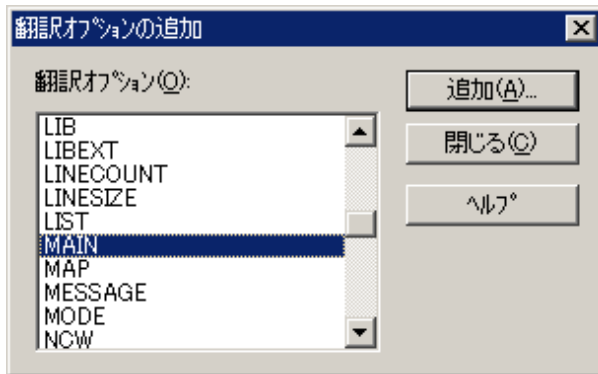
1. プロジェクトマネージャを起動します。
→ プロジェクトマネージャウィンドウが表示されます。
2. プロジェクトマネージャウィンドウの[ツール]メニューから“翻訳”を選択します。
→ WINCOBウィンドウが表示されます。
3. [参照]ボタンを使って、ソースファイルのエディットボックスにサンプルプログラムの格納先ファイル名を指定します。ファイル名は、エディットボックスに直接キー入力することもできます。



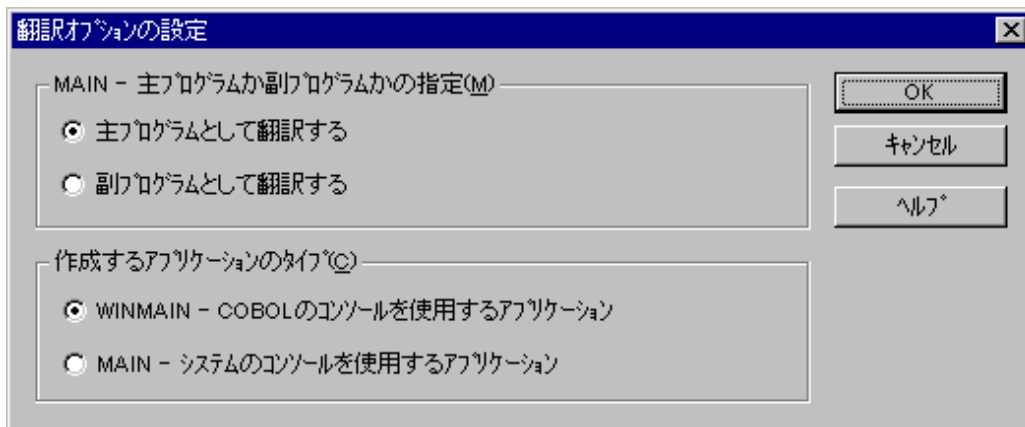
4. WINCOBウィンドウの[オプション]ボタンをクリックします。
→ [翻訳オプション]ダイアログが表示されます。



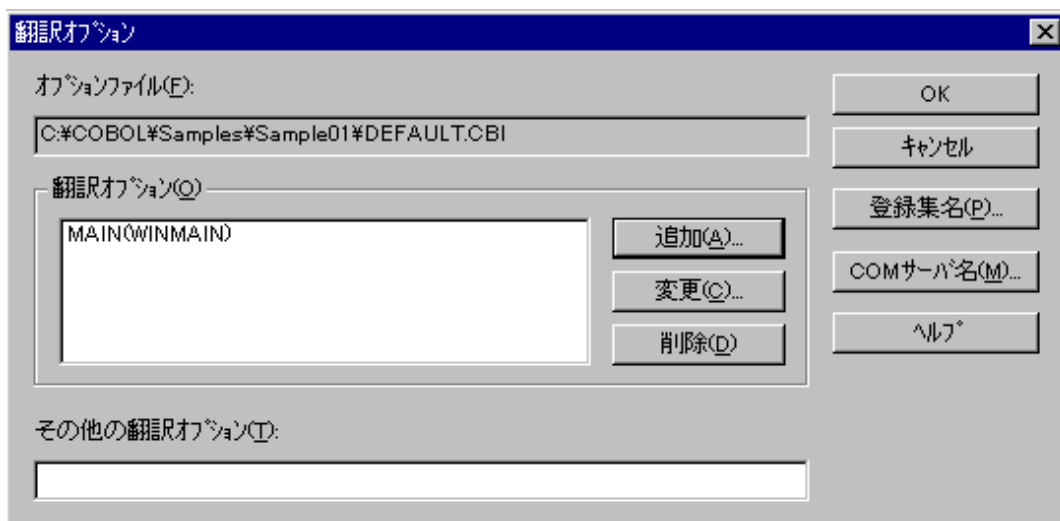
5. [追加]ボタンをクリックします。
→ [翻訳オプションの追加]ダイアログが表示されます。



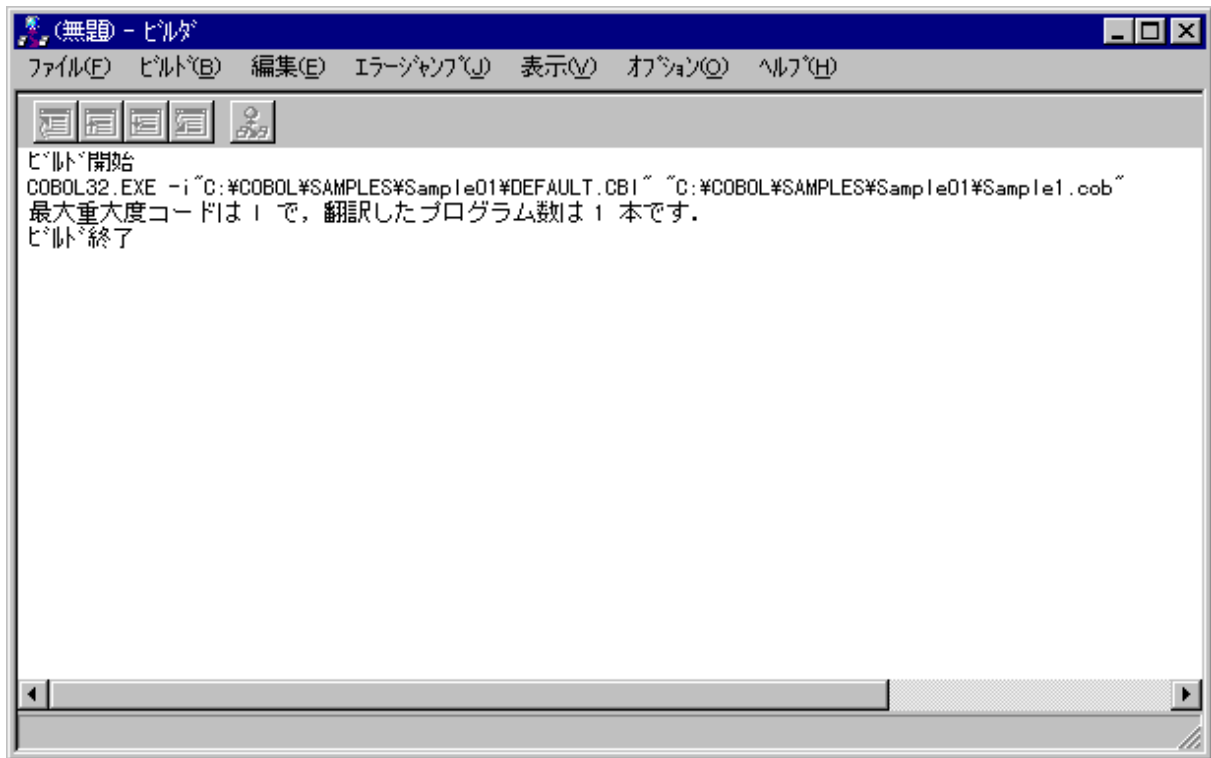
6. 翻訳オプションのリストボックスから“MAIN”を選択し、[追加]ボタンをクリックします。
→ [翻訳オプションの設定]ダイアログが表示されます。



7. “主プログラムとして翻訳する”および“WINMAIN-COBOLのコンソールを使用するアプリケーション”を選択し、[OK]ボタンをクリックします。
→ [翻訳オプションの設定]ダイアログが閉じます。
8. 追加するオプションはないので、[翻訳オプションの追加]ダイアログの[閉じる]ボタンをクリックします。
→ [翻訳オプションの追加]ダイアログが閉じます。



9. [翻訳オプション]ダイアログの翻訳オプションのリストボックスに“MAIN(WINMAIN)”と表示されていることを確認したら、[OK]ボタンをクリックします。
→ [翻訳オプション]ダイアログが閉じます。
10. WINCOBウインドウの[翻訳]ボタンをクリックします。
→ 翻訳が開始されます。
11. 翻訳が終了します。
→ ビルダウインドウにメッセージが表示されます。



12. 翻訳が正常に行われたことを確認したら、WINCOBウインドウの[キャンセル]ボタンをクリックします。
→ WINCOBウインドウが閉じます。

以上でサンプルプログラムの翻訳は終わりです。サンプルプログラムの格納先(ここでは、C:\COBOL\SAMPLES\SAMPLE01)に、目的のプログラム(SAMPLE1.OBJ)が作成されているか確認してください。

なお、翻訳が正常に行われなかった場合には、インストールが正しく行われていない可能性があります。インストールが正しく行われているか確認してください。

3.2 翻訳に必要な資源

COBOLソースプログラムの翻訳は、COBOLコンパイラが行います。ここでは、COBOLコンパイラが使用するファイルおよび出力する情報について説明します。

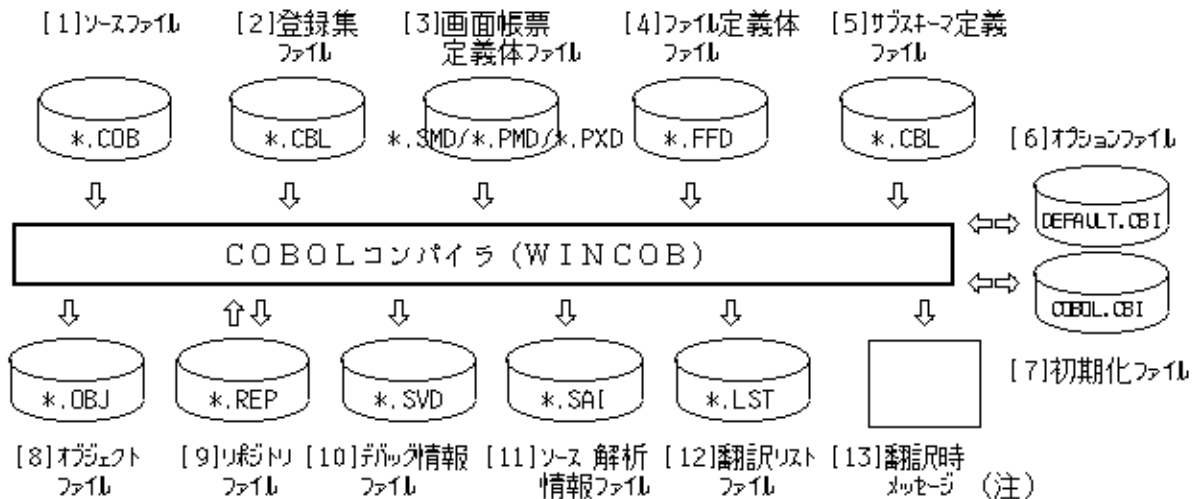
3.2.1 COBOLコンパイラが使用するファイル

COBOLコンパイラは、以下のファイルを使用します。

- ・ ソースファイル(*.COB)
- ・ 登録集ファイル(*.CBL)
- ・ 画面帳票定義体ファイル(*.SMD/*.PMD/*.PXD)
- ・ ファイル定義体ファイル(*.FFD)
- ・ サブスキーマ定義ファイル(*.CBL)

- ・ オプションファイル(DEFAULT.CBI)
- ・ 初期化ファイル(COBOL.CBI)
- ・ オブジェクトファイル(*.OBJ)
- ・ リポジットファイル(*.REP)
- ・ デバッグ情報ファイル(*.SVD)
- ・ ソース解析情報ファイル(*.SAI)
- ・ 翻訳リストファイル(*.LST)

COBOLコンパイラが使用するファイルの関係



注：翻訳終了時に、ビルダウィンドウに翻訳時メッセージが出力されます。

COBOLコンパイラが使用するファイル

	ファイルの内容	ファイル名の形式	入出力	使用する条件または作成する条件	関連する翻訳オプション	関連する環境変数
[1]	COBOLソースプログラム	任意(通常、拡張子COB、CBLまたはCOBOL) (注1)	入力	必須	—	—
[2]	登録集原文	原文名.CBL (注2)	入力	COPY文を使ったソースプログラムを翻訳する場合 (注3)	LIB LIBEXT	COB_COBCOPY COB_LIBSUFFIX COB_登録集名
[3]	画面帳票定義体	画面帳票定義体名 .SMD (注4) .PMD (注5) .PXD	入力	画面帳票定義体を使ったソースプログラムを翻訳する場合 (注6)	FORMLIB FORMEXT	FORMLIBSMED_SUFFIX
[4]	ファイル定義体	ファイル定義体名.FFD (注4)	入力	ファイル定義体を使ったソースプログラムを翻訳する場合 (注6)	FILELIB FILEEXT	FILELIB FFD_SUFFIX
[5]	サブスキーマ定義(注10)	サブスキーマ名.CBL	入力	サブスキーマを使ったソースプログラムを翻訳する場合	AIMLIB	COB_AIMLIB

	ファイルの内容	ファイル名の形式	入出力	使用する条件または作成する条件	関連する翻訳オプション	関連する環境変数
[6]	オプション文字列	DEFAULT.CBI または 任意(注7)	入力	WINCOBウィンドウまたはCOBOL32 コマンドでファイルに格納した翻訳オプションを指定する場合	—	—
			出力	WINCOBウィンドウの〔翻訳オプション〕ダイアログで翻訳オプションの設定を行った場合		
[7]	COBOLコンパイラが使用する情報	COBOL.CBI	入出力	NetCOBOLのインストール時にNetCOBOLをインストールしたフォルダに作成する	—	—
[8]	目的プログラム	ソースファイル名 .OBJ	出力	ソースプログラムを翻訳し、翻訳に成功した場合	OBJECT	—
[9]	クラス情報 (注8)	クラス名.REP	入力	リポジトリ段落を指定したソースプログラムを翻訳する場合	REP REPIN	COB_REPIN
[10]	デバッグ情報	ソースファイル名 .SVD	出力	翻訳オプションTESTを指定した場合	TEST	—
[11]	ソース解析情報 (注9)	ソースファイル名 .SAI	出力	翻訳オプションSAIを指定した場合	SAI	—
[12]	翻訳リスト	ソースファイル名 .LST	出力	翻訳リストを出力する場合	PRINT	—
[13]	翻訳時メッセージ	—		ウィンドウ上に表示する	—	—

注1

COBOLソースファイル名に、以下の拡張子を使用してはなりません。

- 拡張子OBJ
- SVD
- REP
- SAI
- LST

注2

拡張子は、翻訳オプションLIBEXTまたは環境変数COB_LIBSUFFIXを使って任意の文字列に変更できます。翻訳オプションLIBEXTおよび環境変数COB_LIBSUFFIXのどちらも指定しない場合、以下の順で検索します。[参照]“[A.2.24 LIBEXT\(登録集ファイルの拡張子の指定\)](#)”、“[1.2.1 環境変数の設定](#)”

1. 拡張子CBL
2. 拡張子COB
3. 拡張子COBOL

注3

登録集ファイルの検索方法は、COPY文の書き方によって異なります。以下に登録集ファイルの検索順序を示します。

IN/OF 登録集名の指定がないCOPY文を記述した場合

1. 翻訳オプションLIBに指定したフォルダを検索します。複数のフォルダを指定した場合、指定した順序で検索します。[参照]“[A.2.23 LIB\(登録集ファイルのフォルダの指定\)](#)”
2. 環境変数COB_COBCOPYに指定したフォルダを検索します。複数のフォルダを指定した場合、指定した順序で検索します。
3. カレントフォルダを検索します。カレントフォルダについては、“[A.2 翻訳オプションの指定形式](#)”を参照してください。

IN/OF 登録集名の指定があるCOPY文を記述した場合

1. 登録集名に関連付けたフォルダを検索します。登録集名にフォルダを関連付けていない場合、エラーとなります。
2. [登録集名の設定]ダイアログで指定したフォルダを検索します。[参照]“[3.5.2 登録集名の設定](#)”
3. [登録集名の設定]ダイアログの指定がない場合、環境変数COB_登録集名に指定したフォルダを検索します。

注4

それぞれの拡張子は、翻訳オプションFORMEXT/FILEEXTまたは環境変数SMED_SUFFIX/FFD_SUFFIXを使って任意の文字列に変更できます。[参照]“[A.2.18 FORMEXT\(画面帳票定義体ファイルの拡張子の指定\)](#)”、“[A.2.14 FILEEXT\(ファイル定義体ファイルの拡張子の指定\)](#)”、“[1.2.1 環境変数の設定](#)”

注5

拡張子PMD、PXDは、帳票定義体だけです。

注6

画面帳票定義体の検索順序は、“[A.2.19 FORMLIB\(画面帳票定義体ファイルのフォルダの指定\)](#)”を参照してください。
ファイル定義体の検索順序は、“[A.2.15 FILELIB\(ファイル定義体ファイルのフォルダの指定\)](#)”を参照してください。

注7

オプションファイルの作成方法および使用方法は、“[3.8.1.13 -i\(オプションファイルの指定\)](#)”を参照してください。

注8

リポトリファイルの詳細は、“[17.4.3.1 リポトリファイル](#)”を参照してください。

注9

ソース解析情報ファイルの詳細は、“[17.4.6.2 使用する資源](#)”を参照してください。

注10

サブスキーマ定義ファイルの詳細は、“[OSIV分散開発の手引き](#)”を参照してください。

3.2.2 COBOLコンパイラが出力する情報

COBOLコンパイラが出力する情報には、以下があります。

- ・ 診断メッセージ
- ・ オプション情報リスト
- ・ 翻訳単位統計情報リスト
- ・ 相互参照リスト
- ・ ソースプログラムリスト
- ・ 目的プログラムリスト
- ・ データエリアに関するリスト

3.2.2.1 診断メッセージ

COBOLコンパイラは、プログラムの翻訳結果を診断メッセージとして通知します。診断メッセージは、翻訳終了後にウィンドウに表示されます。診断メッセージをファイルに格納するには、以下の方法があります。

- ・ 翻訳オプションPRINTを指定して翻訳を行う。
- ・ ビルドウィンドウの[オプション]メニューから“ビルド結果ファイルの設定”を選択し、表示されたメッセージをファイルに保存する指定を行う。

診断メッセージリスト

NetCOBOL Vxx. x. x	ソース名	2009年10月01日 (木) 13時35分50秒	0002
[3]	[1]		
* * 診断メッセージ * * (ソース名)			
[1]			
JMN2503I-S 4	利用者語'A' が定義されていません。	...	[2]

[1] ソース名(プログラム名、クラス名、メソッド名)を表示します。

[2] COBOLコンパイラが出力する診断メッセージを表示します。診断メッセージの形式については、“メッセージ説明書”の“第2章 翻訳時メッセージ”を参照してください。

[3] 本製品のバージョンを表示します。

3.2.2.2 オプション情報リスト、翻訳単位統計情報リスト

翻訳オプションMESSAGEを指定すると、翻訳時に指定した翻訳オプションや翻訳したプログラムの情報を含むオプション情報リストおよび翻訳単位統計情報リストを出力します。このように翻訳時に出力する各種リストを翻訳リストといいます。

以下に翻訳オプションPRINTおよびMESSAGEを指定した場合に出力する翻訳リストの形式を示します。[参照]“[A.2.38 PRINT \(各種翻訳リストの出力の可否および出力先の指定\)](#)”、“[A.2.30 MESSAGE \(オプション情報リスト、翻訳単位統計情報リストの出力の可否\)](#)”

オプション情報リスト

NetCOBOL Vxx. x. x	2009年10月01日 (木) 13時35分50秒	0001
[10]	[1]	[2]
* * 指定翻訳オプション * *		
MAIN, MESSAGE	...	[3]
* * 確定翻訳オプション * *		
ALPHAL (ALL)	NOEQUALS	NONAME
THREAD (SINGLE)		
BINARY (WORD, MLBON)	FLAG (I)	NCW (STD)
:	:	:
		↑
		↓
		...
		[4]

翻訳単位統計情報リスト

NetCOBOL Vxx. x. x	TEST01	2009年10月01日 (木) 13時35分50秒	0003
* * プログラム特性リスト * *			
ファイル名 = TEST. COB			
ソース名 = TEST01			
翻訳日付 = 2009年10月01日 (木) 13時35分50秒			

原始プログラムのレコード数	=	4 レコード	… [5]
目的プログラムの大きさ (CODEサイズ)	=	0 バイト	… [6]
制御レベル	=	0101 レベル	… [7]
翻訳に要した時間	=	0.28 秒	… [8]
最大重大度コード	=	S	… [9]

- [1] 翻訳日時を表示します。
- [2] ページ番号を表示します。
- [3] 利用者が指定した翻訳オプションを表示します。
- [4] COBOLコンパイラで確立した翻訳オプションの一覧を表示します。
- [5] ソースプログラムのレコード数を表示します。登録集原文が取り込まれている場合、登録集原文のレコードの数も含みます。
- [6] 生成された目的プログラムの大きさを表示します。なお、正常終了してオブジェクトファイルが出力された場合には、DATAサイズも出力します。
- [7] 動作したコンパイラのレベルを表示します。
- [8] 翻訳に要した時間を表示します。
- [9] 出力する診断メッセージの重大度のうち、最大の重大度コードを表示します。
- [10] 本製品のバージョンを表示します。

3.2.2.3 相互参照リスト

翻訳オプションXREFを指定すると、相互参照リストを出力します。[参照]“[A.2.60 XREF\(相互参照リストの出力の可否\)](#)”

相互参照リスト

```

NetCOBOL Vxx.x.x ARITH          2009年10月01日 (木) 12時12分47秒 0001
[4]

定義行 名標      属性と参照行  A:ARGUMENT D:DATA P:PERFORM R:REFER S:SET

[1] [2]          [3]
1-1  A          8S 12R 15R 18R 24R
   2  ARITH
1-2  B          10S 12R 15R 18R 21R 24R
   2# BASE      2D 6D
1-3  C          12S 13R 15S 16R 18S 19R 24S 25R
1-4  D

```

[1] 定義行の行番号

行番号を次の形式で表示します。

[COPY修飾値-]行番号[別翻訳単位定義記号]

- COPY修飾値
ソースプログラムに組み込まれた登録集原文に付加される識別番号です。COPY文に対して、1から1きざみに昇順に割り当てます。
- 行番号
名前を定義した行番号を表示します。暗黙定義されたものは、“*”で表示します。
- 別翻訳単位定義記号
名前を別の翻訳単位中で定義している場合、行番号に“#”を付加して表示します。

[2] 名標

ソースプログラムで定義した名標を表示します。名標を表示する領域は、ANK文字または日本語文字で30文字分です。

[3] 属性と参照行

名前を明示参照している行の行番号および参照形態を表示します。参照形態は、以下の記号で表示します。

- A: CALL文、INVOKE文、メソッドの行内呼出しのパラメタ
- D: 見出し部、環境部、データ部での参照
- P: PERFORM文による参照
- R: 手続き部での参照
- S: 設定

[4] バージョン

本製品のバージョンを表示します。

3.2.2.4 ソースプログラムリスト

翻訳オプションSOURCEを指定すると、翻訳リストファイルにソースプログラムリストが出力されます。また、翻訳オプションCOPYを指定すると、ソースプログラムリスト中に、COPY文によって取り込まれた登録集原文が出力されます。

ソースプログラムリストの出力形式

以下に、ソースプログラムリストの出力形式を示します。

行番号	一連番号	A	B
[1]			[2]
1	000100	IDENTIFICATION	DIVISION.
2	000200	PROGRAM-ID.	A.
3	000300*		
4	000400	DATA	DIVISION.
5	000500	WORKING-STORAGE	SECTION.
6	000600	COPY	A1.
1-1 C	000600 77	答え	PIC 9(2).
1-2 C	000700 77	除数	PIC 9(2).
1-3 C	000800 77	被除数	PIC 9(2).
7	000900*		
8	001000	PROCEDURE	DIVISION.
9	001100*		
10	001200	MOVE 10 TO	被除数.
11	001300	MOVE 0 TO	除数.
12	001400*		
13	001500	COMPUTE	答え = 被除数 / 除数.
14	001600*		
15	001700	EXIT	PROGRAM.
16	001800	END	PROGRAM A.

[1] 行番号

行番号を次の形式で表示します

1. 翻訳オプションNUMBER有効時

[COPY修飾値-]利用者行番号

- COPY修飾値
ソースプログラムに組み込まれた登録集原文の識別番号、または昇順になっていない一連番号を持つ行に付加する識別番号です。COPY命令または昇順になっていない一連番号に対して1から1きざみに割り当てます。
- 利用者行番号
ソースプログラムの一連番号領域の値を使用します。一連番号領域に数字以外の文字が含まれている場合には、その行の一連番号は直前の正しい一連番号に1を加えた値に変更します。また、同一の一連番号が連続していても、誤りとはしないでそのまま使用します。

2. 翻訳オプションNONUMBER有効時

[COPY修飾値-] ソースファイル内相対番号

- **COPY修飾値**
ソースプログラムに組み込まれた登録集原文の識別番号に付加する識別番号です。COPY命令に対して、1から1きざみに割り当てます。
- **ソースファイル内相対番号**
コンパイラは、行番号として1から1きざみに昇順に割り当てます。COPY命令により組み込んだソースに対しても1から1きざみに割り当て、行番号とソースプログラムの間にCOPY命令による組込み表示(“C”で表示)を行います。

[2] ソースプログラム

ソースプログラムそのものを表示します。

3.2.2.5 目的プログラムリスト

翻訳オプションLISTを指定すると、翻訳リストファイルに目的プログラムリストが出力されます。

目的プログラムリストの出力形式

以下に、目的プログラムリストの出力形式を示します。

```

BEGINNING OF NONDECLARATIVE PROCEDURES

[9]
BBK=00004 (01)
..... LX
..... PN. PX

[1]      [2]      [4]
0000017A 8D05A0000000      lea      eax,%GWA+000000A0
00000180 A39C000000      mov      %GWA+0000009C,eax
00000185 C7059800000000000000      mov      %GWA+00000098,00000000 : 0THAR. 1  [6]
0000018F      GLB. 3  [3]
--- 12 ---      MOVE [5]
0000018F 66A114000000      mov      ax,%COA+00000014 : +10  [8]
00000195 66A3DA000000      mov      %GWA+000000DA,ax : 除数  [7]
0000019B A0DB000000      mov      al,%GWA+000000DB : 除数+1
000001A0 240F      and      al,0F
000001A2 0C30      or       al,30
000001A4 A2DB000000      mov      %GWA+000000DB,al : 除数+1
--- 13 ---      MOVE
000001A9 66A112000000      mov      ax,%COA+00000012 : +00
000001AF 66A3DC000000      mov      %GWA+000000DC,ax : 被除数
000001B5 A0DD000000      mov      al,%GWA+000000DD : 被除数+1
000001BA 240F      and      al,0F
000001BC 0C30      or       al,30
000001BE A2DD000000      mov      %GWA+000000DD,al : 被除数+1
--- 15 ---      COMPUTE
<REDUNDANT STORE>
<REDUNDANT STORE>
<REDUNDANT STORE>
<REDUNDANT STORE>
--- 17 ---      EXIT PROGRAM
000001C3 EB8D      jmp      GLB. 2

BBK=00005 (00)
NEVER EXECUTED
..... PX

END OF A

```

[1] オフセット

機械語の文のオブジェクト内相対オフセットを表しています。

[2] 機械語の命令コード

機械語の文(オブジェクトコード)を表しています。

[補足]前方参照している分岐命令のとき、機械語コード部が変更されている場合があります。

[3] 手続き名と手続き番号

コンパイラが生成した手続き名と手続き番号を表しています。

[4] アセンブラ形式の命令

機械語の文をMicrosoft(R)マクロアセンブラ言語に準じた形式で表しています。

[5] 動詞名と行番号

COBOLプログラム中に記述された動詞名と行番号を表しています。

[6] 領域名

COBOLコンパイラが使用している領域名を表しています。

[7] COBOLプログラム中に記述された定義語

目的プログラムリスト中に表示される定義語と表現は、以下のとおりです。

定義語の内容	目的プログラムリスト上の表現
定義語(英数字)	定義語(英数字)  例 mov %GWA+00000108, AX :IDX
定義語(日本語)	定義語(日本語)  例 mov %GWA+00000108, AX :あ

[8] COBOLプログラム中に記述された定数

目的プログラムリスト中に表示される定数と表現は、以下のとおりです。

定数の内容	目的プログラムリスト上の表現
数字	数字  例 sub esp, %COA+00000040 :+8
文字(英数字)	文字(英数字)  例 mov ax, %COA+000000012 : "ABCD"
文字(日本語)	文字(日本語)

定数の内容	目的プログラムリスト上の表現
	 例 mov ax, %COA+000000012 :NC“あい”
数字(内部浮動小数点) 数字(外部浮動小数点) ブール(内部) ブール(外部)	表示されません。

[9] 最適化情報

翻訳オプションOPTIMIZEを指定したCOBOLプログラムの目的プログラムリストにだけ存在します。どんな最適化がどこで実施されたかの概要を表します。[参照]“[A.2.37 OPTIMIZE \(広域最適化の扱い\)](#)”

3.2.2.6 データエリアに関するリスト

翻訳オプションMAPを指定すると、翻訳リストファイルにデータエリアに関するリストが出力されます。

データエリアに関するリストには、以下があります。

- ・ データマップリスト
- ・ プログラム制御情報リスト
- ・ セクションサイズリスト

データマップリストの出力形式

[1] 行番号	[2] 番地	[3] オフセット	[4] 変位	[5] レベル 名標	[6] 長さ(10)	[7] 属性	[8] 基点	[9] 次元数
[10]**MAIN**								
15	GWA+00000124			FD OUTFILE		LSAM	BGW. 000000	
16	[GWA+00000020]+00000000		0 01	印刷レコード	60	ALPHANUM	BVA. 000003	
18	GWA+000001F8		0 77	答え	8	EXT-DEC	BGW. 000000	
19	GWA+00000200		0 77	除数	4	EXT-DEC	BGW. 000000	
21	COA+00000060		0 01	CSTART	8	ALPHANUM	BCO. 000000	
22	COA+00000068		0 01	CEND	8	ALPHANUM	BCO. 000000	
23	COA+00000070		0 01	CRES	8	ALPHANUM	BCO. 000000	
25	[GWA+00000018]+00000000		0 01	被除数	2	EXT-DEC	BVA. 000001	
[11]**SUB1**								
48	[GWA+0000001C]+00000000		0 01	表示	8	EXT-DEC	BVA. 000002	

ソースプログラムのデータ部(作業場所節、ファイル節、定数節、連絡節、報告書節)に記述されたデータについて、目的プログラム内におけるデータ領域の割り付け情報とデータの属性情報を出力します。

[1] 行番号

行番号を次の形式で表示します

COPY修飾値、利用者行番号、ソース内相対番号の詳細は、“[3.2.2.4 ソースプログラムリスト](#)”を参照してください。

翻訳オプションNUMBER有効時

[COPY修飾値-] 利用者行番号

翻訳オプションNONUMBER有効時

[COPY修飾値-] ソースファイル内相対番号

[2] 番地、オフセット

番地は、目的プログラム内に割り付けられたデータ項目の領域を、次の形式で表示します。

セクション名+相対アドレス

ー セクション名

以下のいずれかを表示します。

- COA (.rodata)
- GWA (.data)
- GW2 (.bss)
- EBP (スタック)
- HGW (ヒープ)
- HPA (ヒープ)

それぞれのセクションの基点アドレスは、以下の方法で求めます。

- COA(.rodata)、GWA(.data)およびGW2(.bss)の基点アドレスは、リンクマップリストを参照して求めます。参照方法については、“[19.7 翻訳リストを使ったデバッグ](#)”を参照してください。
- EBPはEBPレジスタです。
- HGWおよびHPAの基点アドレスは、スタック域に格納されています。基点アドレスを求めるには、プログラム制御情報リストを参照する必要があります。



例

プログラム制御情報リスト

** STK **		
:		
* SGM *		
.....	SGM POINTERS AREA	0
:		
EBP+FFFFFFB8	BOD	4
EBP+FFFFFFBC	BHG	4

HGWの基点アドレスは、BHGに格納されます。また、HPAの基点アドレスはBODに格納されます。

上の例の場合、HGWは、EBP+FFFFFFBCの指す位置に基点アドレスが格納され、HPAはEBP+FFFFFFB8の指す位置に基点アドレスが格納されていることを示します。

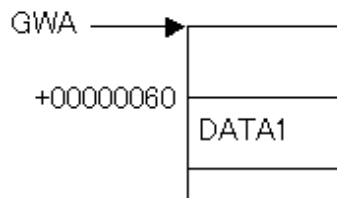
ー 相対アドレス

セクションの基点アドレスからの相対アドレスを表示します。

オフセットは、“[]”付きで表示された番地に対して表示します。

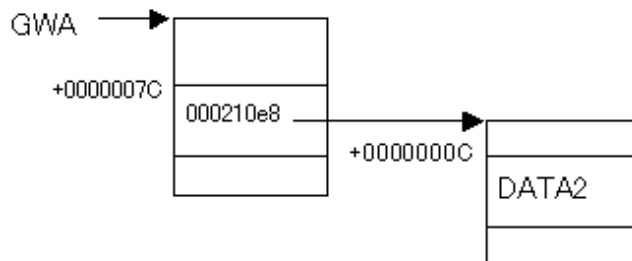
データ領域の参照方法は、オフセットが表示される場合と表示されない場合とで異なります。

ー オフセットが表示されない場合 (DATA1: GWA+00000060)



GWAの基点アドレスに0x60を足した位置に、DATA1のデータ領域があることを示します。

ー オフセットが表示される場合 (DATA2: [GWA+0000007C]+0000000C)



GWAの基点アドレスに0x7Cを足した位置に、アドレスが格納されています(例では000210e8)。このアドレスに0x0Cを足した位置に、DATA2のデータ領域があることを示します(例では0x000210e8+0x0cの位置にDATA2のデータ領域がある)。

[3] 変位

レコード内オフセットを10進数で表示します。

[4] レベル

ソースプログラムに記述されたレベル番号を表示します。

[5] 名標

ソースプログラムに記述されたデータ名を表示します。表示するデータ名の長さが30バイトを超える場合は、30バイト以降は表示しません。

[6] 長さ(10)

データ項目の長さを10進数で表示します。ファイル名の場合は表示しません。

[7] 属性

データの属性を次の記号で表示します。

GROUP-F	固定長集団項目
GROUP-V	可変長集団項目
ALPHA	英字
ALPHANUM	英数字
AN-EDIT	英数字編集
NUM-EDIT	数字編集
INDEX-DATA	指標データ
EXT-DEC	外部10進
INT-DEC	内部10進
FLOAT-L	倍精度内部浮動小数点
FLOAT-S	単精度内部浮動小数点
EXT-FLOAT	外部浮動小数点
BINARY	2進数
COMP-5	2進数
INT-BINARY	Int型2進数データ
INDEX-NAME	指標名
INT-BOOLE	内部ブール
EXT-BOOLE	外部ブール

NATIONAL	日本語
NAT-EDIT	日本語編集
OBJ-REF	オブジェクト参照データ
POINTER	ポインタデータ

FD項目の場合は、ファイル種別と呼出し法を以下の記号で表示します。

SSAM	順ファイル、順呼出し
LSAM	行順ファイル、順呼出し
RSAM	相対ファイル、順呼出し
RRAM	相対ファイル、乱呼出し
RDAM	相対ファイル、動的呼出し
ISAM	索引ファイル、順呼出し
IRAM	索引ファイル、乱呼出し
IDAM	索引ファイル、動的呼出し
PSAM	表示ファイル、順呼出し

[8] 基点

データ項目が割り付けられるベースレジスタとベース位置を表示します。

[9] 次元数

添字または指標付けの次元数を表示します。

[10]、[11] プログラム名

プログラムが入れ子の場合の区切りとしてプログラム名を表示します。
ただし、クラス定義の場合は、各定義の区切りを以下の形式で表示します。

```

**クラス名**
** FACTORY **
** OBJECT **
** MET(メソッド名)**

```

プログラム制御情報リストの出力形式

番地	フィールド名	長さ(10)
[1]		
** GWA **		
* GCB *		
GWA+00000000	GCB FIXED AREA	8
GWA+00000008	TL 1ST AREA	16
* GMB *		
GWA+00000018	GMB POINTERS AREA	12
.....	MUTEX HANDLE AREA	0
.....	FMBE	0
.....	BEA	0
GWA+00000018	BVA	12
.....	VPA	0
.....	PSA	0
GWA+00000024	LIA FIXED AREA	120
GWA+0000009C	IWA1 AREA	4
.....	ALTINX	0
.....	USESARE	0
.....	PCT	0
.....	CONTENT	0
.....	USEOSAVE	0

.....	LINAGE COUNTER	0
GWA+0000009C	OTHER AREA	4
GWA+000000A0	LCB AREA	132
GWA+00000124	GMB CONTROL BLOCKS AREA	208
GWA+00000124	FMB1	208
.....	SMB0	0
.....	SPECIAL REGISTERS	0
.....	S-A LIST	0
.....	DPA	0
.....	SQL CONTROL AREA	0
.....	DMA	0
.....	EXCEPTION PROC LIST	0
.....	SCREEN CONTROL AREA	0
.....	STRONG TYPE AREA	0
.....	FAI AREA	0
.....	ATTR INFO FIXED AREA	0
.....	REUSE INFO TABLE	0
.....	METHOD TABLE	0
.....	INTERFACE TABLE	0
.....	METHOD COPY TABLE	0
.....	IAI AREA	0
.....	ATTR INFO FIXED AREA	0
.....	REUSE INFO TABLE	0
.....	METHOD TABLE	0
.....	INTERFACE TABLE	0
.....	METHOD COPY TABLE	0
.....	CFOR	0
.....	INVOKE PARM INFO	0
.....	AS MODIFY INFO	0
* GWS *		
GWA+000001F8	DATA AREA	12
** COA **		
* CCB *		
COA+00000000	CCB FIXED AREA	4
* CMB *		
COA+00000004	CMB POINTERS AREA	56
.....	VNAS	0
COA+00000004	VNAL	56
.....	IPA	0
.....	BVOF1	0
.....	BVO11	0
.....	FARA	0
.....	VNAM	0
COA+00000040	LITERAL AREA	28
COA+00000060	CONSTANT SECTION DATA	24
COA+00000078	CMB CONTROL BLOCKS AREA	676
COA+00000078	FMB2	148
.....	FMB1	0
.....	SMB1	0
.....	SMB2	0
.....	EDT	0
.....	EFT	0
COA+0000010C	FMB2 ADDR-PRM LIST	8
.....	ERROR PROCEDURE	0
.....	CALL PARM ADD INFO	0
.....	OLE PARM INFO	0
.....	CALL PARM INFO	0
.....	ALPHABETIC NAME	0
.....	CLASS NAME	0
.....	CLASS TEST TABLE	0

.....	TRANS-TABLE PROTO	0
.....	CIPB	0
.....	DOG TABLE	0
.....	EXTERNAL DATA NAME	0
.....	NATIONAL-MSG F-NAM	0
.....	FLOW BLOCK INFO	0
.....	SQL CONTROL AREA	0
.....	SQL INIT INFO	0
.....	NUMEDIT CONV TABLE	0
.....	SPECIAL REGISTERS	0
COA+00000114	EPA CONTROL AREA	504
COA+0000030C	SIGN TABLE	16
.....	SCREEN CONTROL AREA	0
.....	UMA RECORD INFO AREA	0
.....	UMA INFO	0
.....	UMA RECORD LIST	0
.....	FCM FMB1 OFFSET LIST	0
.....	FCM OBJ-REF OFFSET LIST	0
.....	ICM FMB1 OFFSET LIST	0
.....	ICM OBJ-REF OFFSET LIST	0
.....	MCM AREA / OBJ-REF LIST	0
.....	CFOR OFFSET LIST	0
.....	CLASS NAME INFO	0
.....	AS MODIFY / PARAM INFO	0
.....	METHOD NAME INFO	0
.....	CALL PARM ADDR LIST	0
** GW2 **		
* GC2 *		
.....	GCB2 FIXED AREA	0
* GM2 *		
.....	LIA AREA	0
.....	GMB2 POINTERS AREA	0
.....	FMBE	0
.....	BEA / BEAD / BEAR	0
.....	CONTROL AREA ADD TBL	0
.....	BCC	0
.....	MUTEX HANDLE AREA	0
.....	GMB2 CONTROL BLOCKS	0
.....	DSA	0
.....	METHOD INIT INFO	0
.....	CFOR	0
* GS2 *		
.....	DATA AREA	0
** STK **		
* SCB *		
EBP+FFFFFF48	SCB FIXED AREA	92
.....	TL 1ST AREA	0
* SGM *		
.....	SGM POINTERS AREA	0
.....	VPA	0
.....	PSA	0
.....	BVA	0
.....	BOD	0
.....	BHG	0
EBP+FFFFFFA4	IWA3 AREA	4
EBP+FFFFFFA4	PRMSAVE	4
.....	PCT	0
.....	CONTENT	0

.....	RTNAREA	0
.....	USESAVE	0
.....	USEOSAVE	0
.....	LINAGE COUNTER	0
.....	OTHER AREA	0
.....	CALL PARM INFO	0
.....	TRG AREA	0
.....	TSG AREA	0
.....	SOR AREA	0
EBP+FFFFFFA8	LIA VARIABLE	48
.....	LCB AREA	0
.....	SGM CONTROL BLOCKS AREA	0
.....	FMB1	0
.....	SMBO	0
.....	SPECIAL REGISTERS	0
.....	DPA	0
.....	DMA	0
.....	MIA	0
.....	SQL CONTROL AREA	0
* SGW *		
.....	DATA AREA	0
* STL *		
EBP+FFFFFFD8	TL 2ND AREA	8
EBP+FFFFFFEC	LIA ADDRESS	20
* * 定数領域 * *		
[2]		
番地	0 . . . 4 . . . 8 . . . C . . .	0123456789ABCDEF
COA+00000040	10000000 01000000 08000000 04000000	
COA+00000050	40000000 5359534F 55542020	@. . . SYSOUT

[1] 目的プログラム中に存在する各種作業域やデータ領域の割り付け位置を表示します。

[2] 目的プログラム中に存在する定数領域を表示します。

セクションサイズリストの出力形式

* * 目的プログラムの大きさ * *	
[1]	
. textサイズ	= 2409バイト
. dataサイズ	= 516バイト
* * 実行に必要な領域の大きさ * *	
[2]	
. bssサイズ	= 0バイト
ヒープサイズ	= 0バイト
スタックサイズ	= 184バイト

[1] 目的プログラム内の.textセクションと.dataセクションの大きさを表示します。

[2] 実行に必要な領域の大きさを表示します。ただし、クラス定義の場合は以下の形式で表示します。

* * 実行に必要な領域の大きさ * *	
クラス名	
. bssサイズ	= 272バイト
ヒープサイズ	= 0バイト
メソッド名	
スタックサイズ	= 384バイト
:	

[3] スタックサイズは、メソッド定義ごとに表示します。

3.3 翻訳の手順

COBOLソースプログラムの翻訳は、WINCOBウィンドウを使用する方法と、翻訳コマンドを使ってコマンドプロンプト上から行う方法があります。ここでは、WINCOBウィンドウを使って翻訳するときの手順について説明します。翻訳コマンドを使った翻訳操作については、“[3.7 翻訳コマンド](#)”で説明しています。



注意

1つのファイル中に複数の翻訳単位を含むソースプログラムを翻訳する場合、翻訳オプションNAMEを指定してください。[参照]“[A.2.32 NAME \(翻訳単位ごとのオブジェクトファイルの出力の可否\)](#)”

3.3.1 1つのソースファイルを翻訳するときの手順

1つのソースファイルを翻訳するときの手順を以下に示します。

1. WINCOBウィンドウを起動します。[参照]“[3.4.1 WINCOBウィンドウの起動](#)”
2. 翻訳するソースファイル名をソースファイルのエディットボックスに指定します。[参照]“[3.4.3 ファイル名の指定](#)”(単数翻訳の場合)
3. 翻訳オプションを設定します。[参照]“[3.5.1 翻訳オプションの設定](#)”



注意

翻訳オプションの設定は、ソースファイル名を指定してから行ってください。

4. 登録集名を設定します。[参照]“[3.5.2 登録集名の設定](#)”
5. 翻訳を開始します。[参照]“[3.4.6 翻訳の開始と終了](#)”
6. 翻訳が終了すると、ビルダウィンドウに翻訳時メッセージが表示されます。
翻訳エラーが検出された場合には、ビルダのエラージャンプ機能を使ってプログラムの修正を行い、再翻訳してください。[参照]“[3.6 翻訳エラーの修正](#)”
7. 翻訳を終了します。[参照]“[3.4.6 翻訳の開始と終了](#)”
 - 翻訳操作を続ける場合
2.からの操作を繰り返し行ってください。
 - 翻訳操作を終了する場合
WINCOBウィンドウの[キャンセル]ボタンをクリックし、WINCOBウィンドウを閉じてください。

COBOLソースプログラムの記述が正しく、翻訳が正常に終了すると、目的プログラムが作成されます。目的プログラムは、そのままでは実行することはできません。目的プログラムを実行可能プログラムにするには、リンクを行います。リンクについては、“[第4章 プログラムをリンクする](#)”で説明しています。

3.3.2 複数のソースファイルを翻訳するときの手順

WINCOBでは、1回の翻訳操作で、複数のファイルを連続して翻訳することができます。複数のソースファイルを翻訳するときの手順を以下に示します。

1. WINCOBウィンドウを起動します。[参照]“[3.4.1 WINCOBウィンドウの起動](#)”
2. 複数翻訳の指定をします。[参照]“[3.4.2 複数翻訳の指定](#)”
3. 翻訳するソースファイル名を一覧のリストボックスに指定します。[参照]“[3.4.3 ファイル名の指定](#)”(複数翻訳の場合)
4. 翻訳オプションを設定します。[参照]“[3.5.1 翻訳オプションの設定](#)”
5. 登録集名を設定します。[参照]“[3.5.2 登録集名の設定](#)”



注意

複数翻訳では、同一フォルダのソースファイルには、同じ翻訳オプションしか指定できません。異なる翻訳オプションを指定したい場合には、COBOLソースプログラム中に翻訳指示文を記述してください。[参照]“[2.1.4 翻訳指示文](#)”

- 主プログラムを指定します。[参照]“[3.4.5 主プログラムの指定](#)”



注意

[翻訳オプション]ダイアログで翻訳オプションMAINを指定しても、ここで指定した内容の方が優先されます。

- 翻訳を開始します。[参照]“[3.4.6 翻訳の開始と終了](#)”
1つのファイルの翻訳が終了すると、次のファイルの翻訳を行います。
- 翻訳が終了すると、ビルダウィンドウに翻訳時メッセージが表示されます。
翻訳エラーが検出された場合には、ビルダのエラージャンプ機能を使ってプログラムの修正を行い、再翻訳してください。[参照]“[3.6 翻訳エラーの修正](#)”
- 翻訳を終了します。[参照]“[3.4.6 翻訳の開始と終了](#)”
 - 翻訳操作を続ける場合
3.からの操作を繰り返し行ってください。
 - 翻訳操作を終了する場合
WINCOBウィンドウの[キャンセル]ボタンをクリックし、WINCOBウィンドウを閉じてください。

COBOLソースプログラムの記述に間違いがなく、翻訳が正しく終了すると、目的プログラムが作成されます。目的プログラムは、そのままでは実行することはできません。目的プログラムを実行可能プログラムにするには、リンクを行います。リンクについては、“[第4章 プログラムをリンクする](#)”で説明しています。

3.4 WINCOBウィンドウ

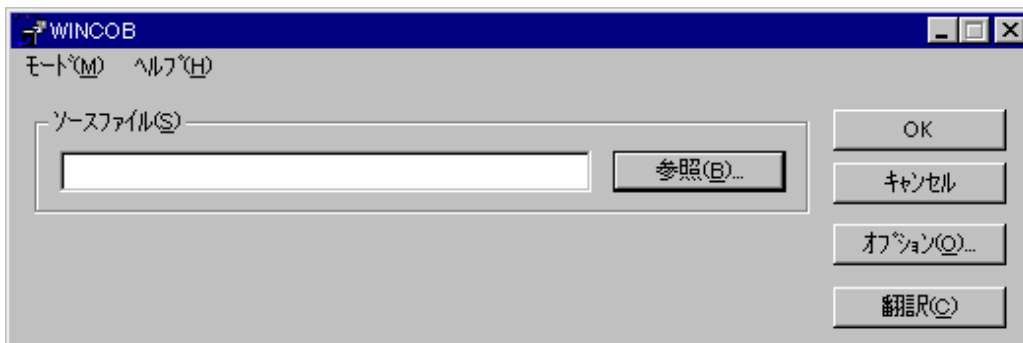
WINCOBウィンドウは、翻訳操作を行うためのウィンドウです。ここでは、WINCOBウィンドウについて説明します。ウィンドウの使い方の詳細は、ヘルプを参照してください。

3.4.1 WINCOBウィンドウの起動

WINCOBウィンドウを起動する方法を以下に示します。

- プロジェクトマネージャの[ツール]メニューから“翻訳”を選択します。

単数翻訳ウィンドウ



メニューバー

モード

翻訳単位の指定(単数/複数)を行います。

ヘルプ

WINCOBウィンドウのヘルプを表示します。

エディットボックス

ソースファイル

翻訳するCOBOLソースファイルを指定します。

ボタン

OK

翻訳を開始するときに選択します。翻訳終了後、WINCOBウィンドウを閉じます。

キャンセル

WINCOBウィンドウを閉じます。

オプション

翻訳オプションを指定するときに選択します。

翻訳

翻訳を開始するときに選択します。

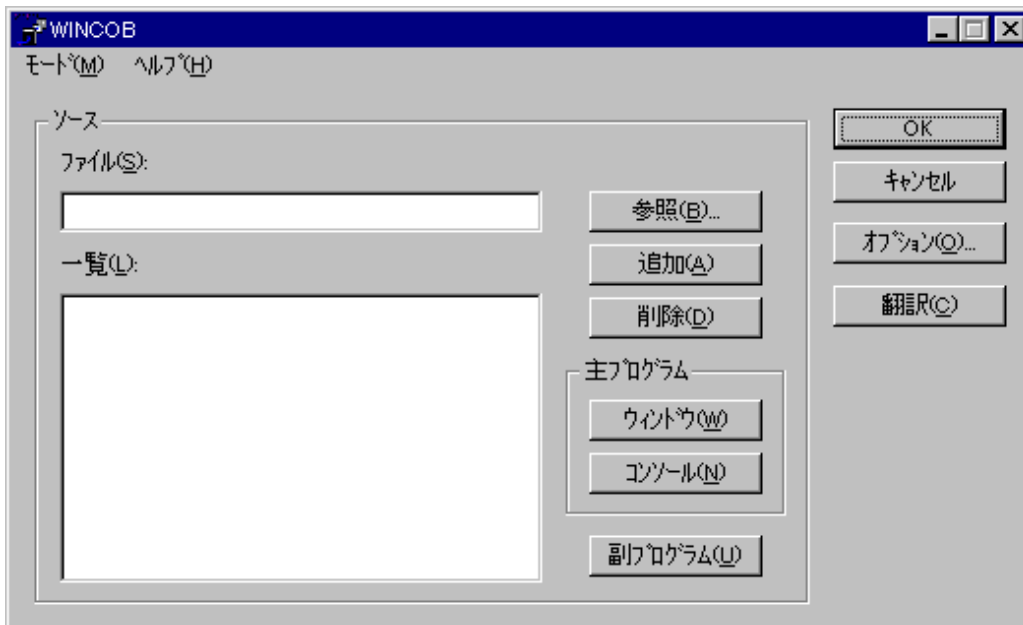
参照

ファイルを参照するときに選択します。

3.4.2 複数翻訳の指定

複数翻訳を行う場合は、WINCOBウィンドウの[モード]メニューから“複数翻訳”を選択し、複数翻訳ウィンドウで指定します。

複数翻訳ウィンドウ



メニューバー

モード

翻訳単位の指定(単数/複数)を行います。

ヘルプ

WINCOBウィンドウのヘルプを表示します。

エディットボックス

ファイル

リストボックスに追加するCOBOLソースファイルを指定します。

リストボックス

一覧

翻訳対象とするファイルを表示します。

ボタン

OK

翻訳を開始するときに選択します。翻訳終了後、WINCOBウィンドウを閉じます。

キャンセル

WINCOBウィンドウを閉じます。

オプション

翻訳オプションを指定するときに選択します。

翻訳

翻訳を開始するときに選択します。

参照

ファイルを参照するときに選択します。

追加

リストボックスにファイルを追加するときに選択します。

削除

リストボックスからファイルを削除するときに選択します。

ウィンドウ(主プログラム)

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、また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場合に指定します。

コンソール(主プログラム)

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)を使用する場合に指定します。

副プログラム

副プログラムを指定します。

3.4.3 ファイル名の指定

ファイル名の指定方法は、単数翻訳と複数翻訳とは異なります。

以下にそれぞれの場合の指定方法を示します。

単数翻訳の場合

ソースファイルのエディットボックスに指定します。

翻訳するファイルの名前は、直接キー入力するか、[参照]ボタンを使って指定します。

複数翻訳の場合

一覧のリストボックスに指定します。

1. 単数翻訳と同様に、翻訳するファイルの名前を、ファイルのエディットボックスに指定します。
2. [追加]ボタンをクリックします。

3.4.4 翻訳オプションの設定

翻訳オプションの設定は、[オプション]ボタンをクリックして表示される[翻訳オプション]ダイアログで行います。[参照]“[3.5 \[翻訳オプション\]ダイアログ](#)”



注意

.....

翻訳オプションの設定は、ソースファイル名を指定してから行ってください。

.....

3.4.5 主プログラムの指定

主プログラムの指定をするためには、翻訳オプションMAINを指定する必要があります。

ただし、翻訳オプションMAINには優先順位があるため、単数翻訳と複数翻訳で指定する場所に注意してください。

単数翻訳の場合

[翻訳オプション]ダイアログで、翻訳オプションMAIN(WINMAIN)またはMAIN(MAIN)を指定します。

複数翻訳の場合

[ウィンドウ]または[コンソール]ボタンを使って指定します。

1. 一覧のリストボックスから主プログラムとして翻訳するファイル名を選択します。複数のファイルを選択できます。
2. [ウィンドウ]または[コンソール]ボタンをクリックします。
→ 選択されたファイルの小アイコンが、ウィンドウメイン型またはコンソールメイン型に変更されます。



注意

-
- ・ 翻訳オプションMAIN(WINMAIN)とMAIN(MAIN)の相違については、“[11.1.3 コンソールウィンドウを使ったデータの入出力](#)”を参照してください。
 - ・ [ウィンドウ]ボタンをクリックすると翻訳オプションMAIN(WINMAIN)を指定した場合と同じ意味になります。また、[コンソール]ボタンをクリックすると、翻訳オプションMAIN(MAIN)を指定した場合と同じ意味になります。
 - ・ [翻訳オプション]ダイアログで翻訳オプションMAINを指定しても、[ウィンドウ]または[コンソール]ボタンで指定した内容の方が優先されます。
 - ・ 複数翻訳の場合、[ウィンドウ]または[コンソール]ボタンをクリックしないと、副プログラムとみなされてしまうため、主プログラムに対しては必ず指定してください。
-



参考

.....

翻訳オプションMAINの優先順位

1. 翻訳指示文による指定
 2. [ウィンドウ]または[コンソール]ボタンによる指定
 3. [翻訳オプション]ダイアログでの指定
-

3.4.6 翻訳の開始と終了

翻訳を開始するには、[翻訳]ボタンをクリックします。

翻訳が終了すると、ビルダウィンドウに翻訳時メッセージが表示されます。

エラーが検出された場合には、ビルダのエラージャンプ機能を使ってプログラムの修正を行ってください。[参照]“[3.6 翻訳エラーの修正](#)”

翻訳を終了する場合には、[キャンセル]ボタンをクリックし、WINCOBウィンドウを閉じてください。

3.5 [翻訳オプション]ダイアログ

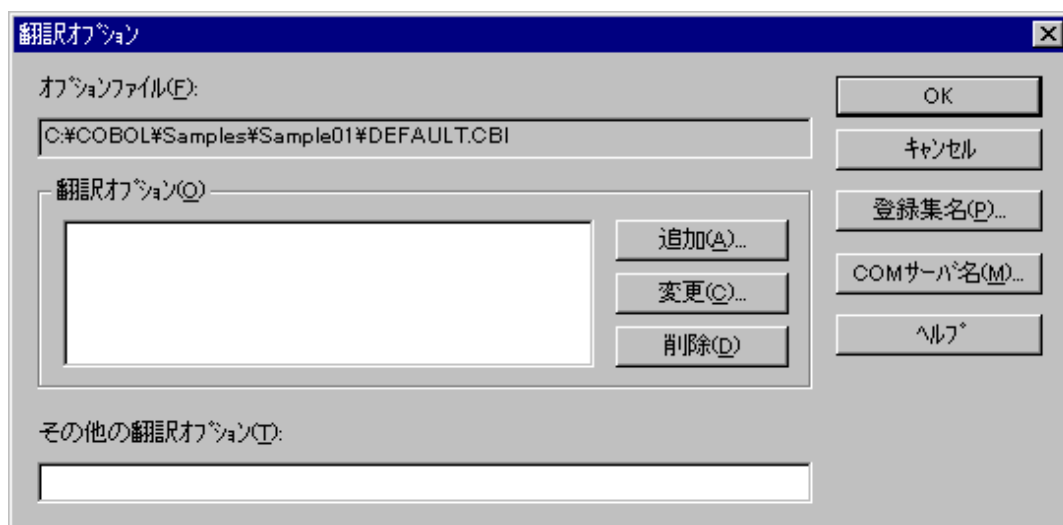
翻訳オプションの設定は、WINCOBウィンドウの[オプション]ボタンをクリックしたときに表示される、[翻訳オプション]ダイアログで行います。翻訳するソースプログラム中に登録集名の指定があるCOPY文が記述されている場合は、[翻訳オプション]ダイアログから[登録集名の設定]ダイアログを表示し、登録集名の設定を行います。また、翻訳するソースプログラム中に*COMアーライバインドクラスが記述されている場合は、[翻訳オプション]ダイアログから[COMサーバの設定]ダイアログを表示し、COMサーバ名の設定を行います。ここでは、翻訳オプションの設定および登録集名の設定について説明します。COMサーバ名の設定については、“[26.8.1 COMサーバ名の設定](#)”を参照してください。

3.5.1 翻訳オプションの設定

翻訳オプションは、翻訳オプションのリストボックスおよびその他の翻訳オプションのエディットボックスを使って設定します。

翻訳オプションのリストボックスを使う場合には、[追加]、[変更]、[削除]ボタンを使って行います。その他の翻訳オプションのエディットボックスを使う場合には、翻訳オプションの文字列を直接キー入力します。

ここでは、リストボックスを使うときの操作方法について説明します。



エディットボックス

オプションファイル

編集集中のオプションファイルが表示されます。

その他の翻訳オプション

翻訳オプションの文字列を直接キー入力することができます。

リストボックス

翻訳オプション

すでに指定されている翻訳オプションおよび指定内容が表示されます。

ボタン

OK

オプションの設定が終了したときに選択します。

キャンセル

[翻訳オプション]ダイアログを開く前の状態に戻し、[翻訳オプション]ダイアログを閉じます。

登録集名

登録集名を指定するときに選択します。

COMサーバ名

COMサーバ名を指定するときに選択します。

ヘルプ

ヘルプウィンドウを開くときに選択します。

追加

翻訳オプションの追加を行うときに選択します。

変更

リストボックスの翻訳オプションを変更するときに選択します。

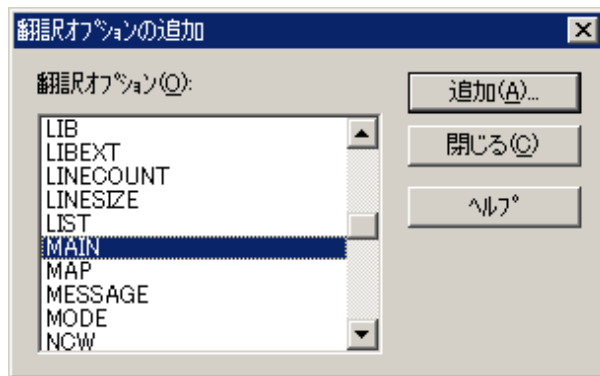
削除

リストボックスから翻訳オプションを削除するときに選択します。

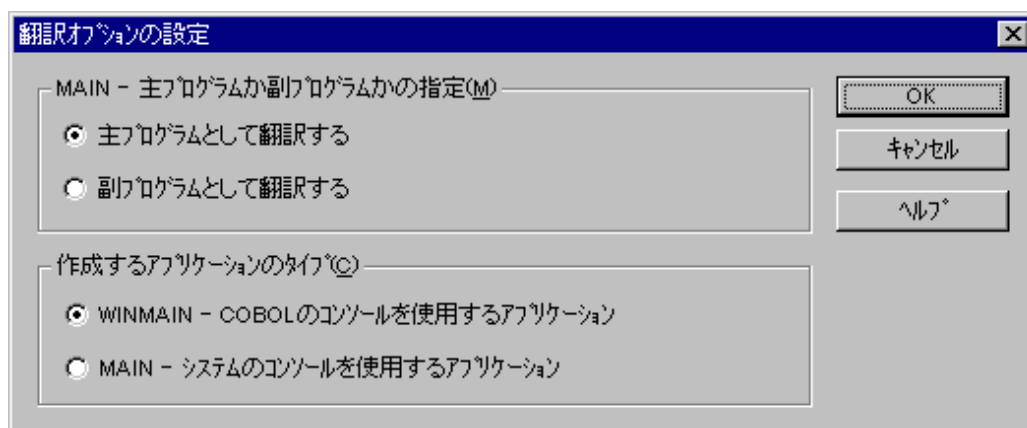
3.5.1.1 翻訳オプションの追加

翻訳オプションのリストボックスに表示されていない翻訳オプションを設定する場合、以下の操作を行います。

1. [翻訳オプション]ダイアログの[追加]ボタンをクリックします。
→ [翻訳オプションの追加]ダイアログが表示されます。



2. 翻訳オプションのリストボックスから、追加する翻訳オプションを選択し、[追加]ボタンをクリックします。
→ [翻訳オプションの設定]ダイアログが表示されます。



3. 翻訳オプションに設定する内容を指定します。
[翻訳オプションの設定]ダイアログの内容は、各翻訳オプションによって異なるため、ヘルプまたは“付録A 翻訳オプション”を参照して指定を行ってください。
4. [翻訳オプションの設定]ダイアログの[OK]ボタンをクリックします。翻訳オプションの設定を取り消したい場合には、[キャンセル]ボタンをクリックしてください。
→ [翻訳オプションの設定]ダイアログが閉じます。

5. 翻訳オプションの追加操作を続ける場合には、2.からの操作を繰り返し行ってください。
6. 追加する翻訳オプションの設定がすべて終わったら、[翻訳オプションの追加]ダイアログの[閉じる]ボタンをクリックします。
→ [翻訳オプションの追加]ダイアログが閉じます。
7. 設定した翻訳オプションは、[翻訳オプション]ダイアログの翻訳オプションのリストボックスに表示されます。

3.5.1.2 翻訳オプションの変更

翻訳オプションのリストボックスに表示されている翻訳オプションの設定内容を変更する場合、以下の操作を行います。

1. 翻訳オプションのリストボックスから、設定内容を変更したい翻訳オプションを選択し、[変更]ボタンをクリックします。
→ [翻訳オプションの設定]ダイアログが表示されます。
2. 変更する内容を設定します。
3. 設定が終わったら、[OK]ボタンをクリックします。翻訳オプションの設定内容を変更しない場合には、[キャンセル]ボタンをクリックします。
→ [翻訳オプションの設定]ダイアログが閉じます。
4. 変更した翻訳オプションの内容は、[翻訳オプション]ダイアログの翻訳オプションのリストボックスに表示されます。

3.5.1.3 翻訳オプションの取消し

翻訳オプションのリストボックスに表示されている翻訳オプションを取り消す場合、以下の操作を行います。

1. 翻訳オプションの変更の場合と同様に、翻訳オプションのリストボックスから取り消したい翻訳オプションを選択し、[削除]ボタンをクリックします。
2. 翻訳オプションのリストボックスから、選択した翻訳オプションが削除されます。

3.5.2 登録集名の設定

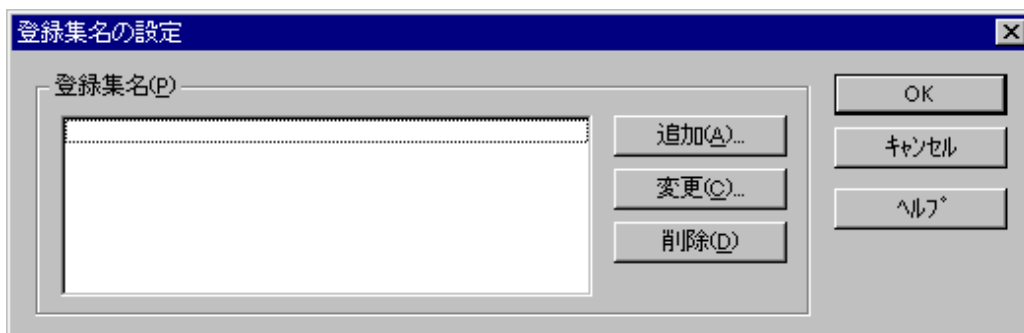
登録集名を指定したCOPY文を使ったCOBOLソースプログラムを翻訳する場合、登録集名と登録集ファイルのフォルダを関係付ける必要があります。登録集名を指定しないCOPY文を使ったCOBOLソースプログラムを翻訳する場合には、翻訳オプションLIBに登録集ファイルのフォルダを指定します。[参照]“[A.2.23 LIB \(登録集ファイルのフォルダの指定\)](#)”

登録集名の設定は、[翻訳オプション]ダイアログの[登録集名]ボタンをクリックしたときに表示される、[登録集名の設定]ダイアログで行います。

以下に、登録集名の追加、すでに指定してある登録集名の設定内容の変更および登録集名の取消しを行う操作について説明します。



ソースプログラム中に登録集名が小文字で記述されているとき、翻訳オプションALPHAL(ALL)またはALPHAL(WORD)を指定して翻訳すると、登録集名は大文字として扱われます。そのため、ここで指定した登録集名が小文字の場合、意図した登録集ファイルが読み込めません。翻訳オプションALPHAL(ALL)またはALPHAL(WORD)を指定して翻訳する場合、ここで指定する登録集名は大文字で指定してください。[参照]“[A.2.2 ALPHAL \(英小文字の扱い\)](#)”



リストボックス

登録集名

すでに指定されている登録集名が表示されます。

ボタン

OK

登録集名の設定が終了したときに選択します。

キャンセル

[登録集名の設定]ダイアログを開く前の状態に戻し、ダイアログを閉じます。

ヘルプ

ヘルプウィンドウを開くときに選択します。

追加

登録集名の追加を行うときに選択します。

変更

リストボックスの登録集名を変更するときに選択します。

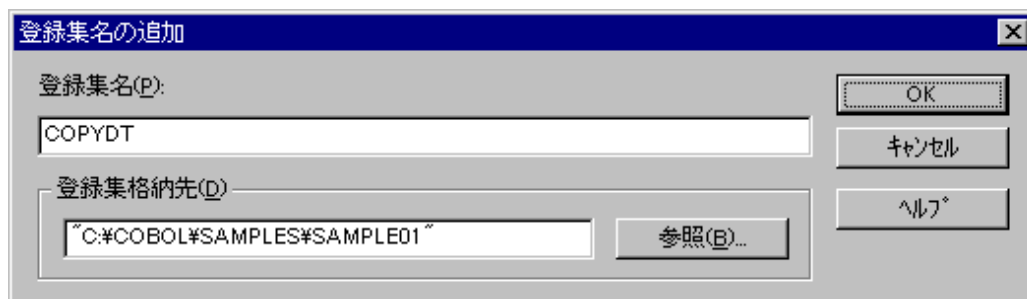
削除

リストボックスから登録集名を削除するときに選択します。

3.5.2.1 登録集名の追加

登録集名のリストボックスに表示されていない登録集名を指定する場合、以下の操作を行います。

1. [登録集名の設定]ダイアログの[追加]ボタンをクリックします。
→ [登録集名の追加]ダイアログが表示されます。
2. COBOLソースプログラムのCOPY文に記述した登録集名と、登録集ファイルのフォルダを指定します。



3. [OK]ボタンをクリックします。指定した内容を取り消したい場合には、[キャンセル]ボタンをクリックします。
→ [登録集名の追加]ダイアログが閉じます。



[OK]ボタンをクリックしないでダイアログを閉じた場合、指定は無効となります。

4. [登録集名の設定]ダイアログの登録集名のリストボックスに、指定した登録集名が表示されます。

3.5.2.2 登録集名の変更

登録集名に対応付けるフォルダを変更する場合、以下の操作を行います。

1. 登録集名のリストボックスから、変更したい登録集名を選択し、[変更]ボタンをクリックします。
→ [登録集格納先の変更]ダイアログが表示されます。
2. 内容を変更します。

3. 変更が終わったら、[OK]ボタンをクリックします。
→ [登録集格納先の変更]ダイアログが閉じます。
4. [登録集名の設定]ダイアログの登録集名のリストボックスに変更した内容が表示されます。

3.5.2.3 登録集名の取消し

登録集名のリストボックスに表示されている登録集名の指定を取り消す場合、以下の操作を行います。

1. 登録集名のリストボックスから、取り消したい登録集名を選択し、[削除]ボタンをクリックします。
2. 登録集名のリストボックスから、選択した登録集名が削除されます。

3.5.2.4 登録集名の設定の終了

登録集名の設定が終了したら、[登録集名の設定]ダイアログの[OK]ボタンをクリックします。

→ [翻訳オプション]ダイアログに戻ります。

3.5.3 設定したオプションの格納

[翻訳オプション]ダイアログの[OK]ボタンをクリックすると、設定した内容がオプションファイルに格納されます。オプションファイルは、WINCOBウィンドウで指定したソースファイルと同じフォルダに“DEFAULT.CBI”というファイル名で作成されます。すでに“DEFAULT.CBI”という名前のファイルが存在する場合、ファイルの内容は更新されます。

それまでに指定した内容を取り消す場合には、[キャンセル]ボタンをクリックします。

[キャンセル]ボタンをクリックした場合、設定内容のオプションファイルへの格納は行われません。また、[OK]ボタンをクリックしないでダイアログを終了した場合にも、それまでの設定はオプションファイルに格納されません。



注意

- 複数翻訳では、同一フォルダのソースファイルには、同じ翻訳オプションしか指定できません。異なる翻訳オプションを指定したい場合には、COBOLソースプログラム中に翻訳指示文を記述してください。[参照]“[2.1.4 翻訳指示文](#)”
- 複数翻訳では、[翻訳オプション]ダイアログで翻訳オプションMAINを指定しても、WINCOBウィンドウの[ウィンドウ]または[コンソール]ボタンで指定した内容の方が優先されます。[参照]“[3.4.5 主プログラムの指定](#)”

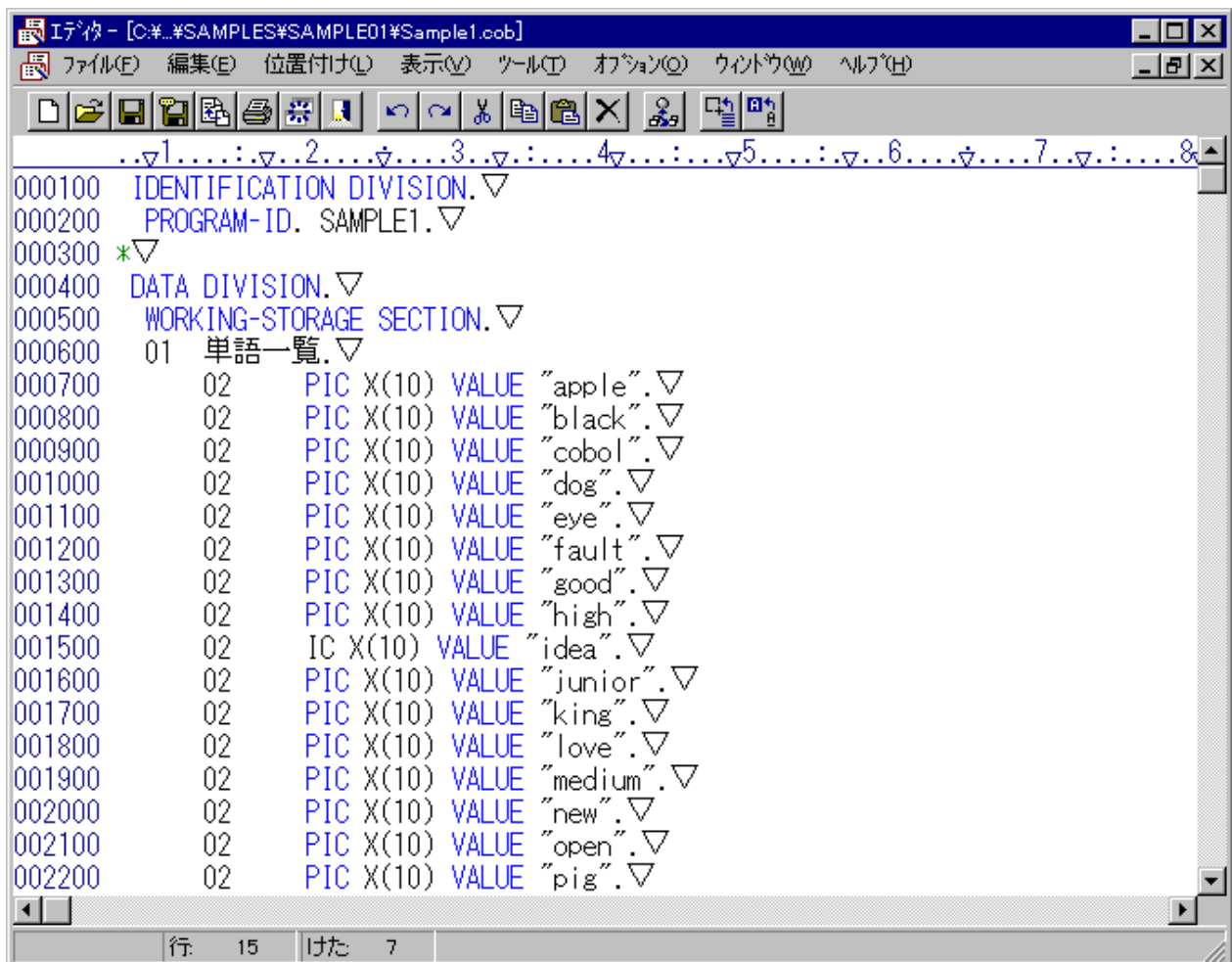
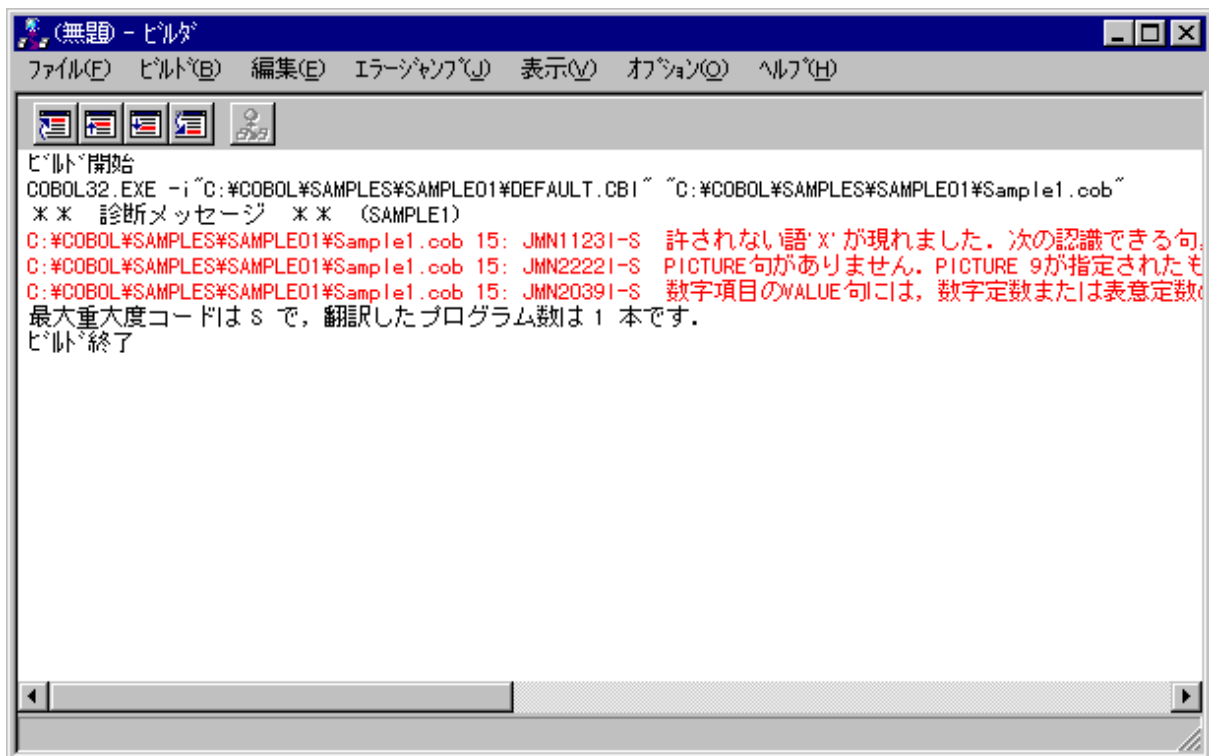
3.6 翻訳エラーの修正

ここでは、COBOLソースプログラムの翻訳エラーの修正に有効な手段であるビルダのエラージャンプ機能について説明します。

3.6.1 概要

WINCOBウィンドウから翻訳を行い、翻訳エラーが検出された場合、翻訳終了時にビルダウィンドウに翻訳時メッセージ(診断メッセージ)が表示されます。

ビルダのエラージャンプ機能では、ビルダウィンドウに表示された診断メッセージから、COBOLソースプログラムの編集画面の翻訳エラーが検出された行にエラージャンプすることができます。COBOLソースプログラムの編集画面の表示は、最初のエラージャンプを行ったときに自動的に行われるので、利用者が事前にエディタを起動しておく必要はありません。このように、ビルダのエラージャンプ機能を使うと、翻訳エラーの発生した文の検出を簡単に行うことができるため、プログラムの修正作業が効率よく行えます。



3.6.2 翻訳時メッセージ

翻訳が終了すると、ビルダウィンドウに翻訳時メッセージが表示されます。

表示されたメッセージは、ビルダウィンドウの[オプション]メニューから“ビルド結果ファイルの設定”を選択し、[ビルド結果ファイル]ダイアログでメッセージをファイルとして保存する指定を行うことによって、ファイルに保存することができます。



- ・ 翻訳オプションMESSAGEが有効な場合、オプション情報リスト、診断メッセージリスト、翻訳単位統計情報リストが翻訳時メッセージの内容となります。[参照]“[A.2.30 MESSAGE \(オプション情報リスト、翻訳単位統計情報リストの出力の可否\)](#)”“[3.2.2 COBOLコンパイラが出力する情報](#)”
- ・ 環境変数TEMPおよび十分なハードディスク上のスペースがない場合、メッセージが作成できないことがあります。

3.6.3 ビルダのエラージャンプ機能の使い方

ビルダのエラージャンプ機能およびエディタの使い方については、ヘルプを参照してください。



- ・ エラージャンプ機能は、翻訳時メッセージとして出力されるメッセージだけを対象とします。翻訳リスト(翻訳オプションPRINTで指定)へ出力したメッセージでは利用できません。
- ・ 翻訳オプションNUMBERが有効な場合は、エラージャンプ機能は利用できません。

3.7 翻訳コマンド

COBOLソースプログラムの翻訳は、翻訳コマンドを使って行うこともできます。

翻訳コマンドは、COBOLソースプログラムを翻訳し、目的プログラムを生成します。

翻訳コマンド実行時に必要となるファイルおよび生成されるファイルの詳細は、“[3.2 翻訳に必要な資源](#)”を参照してください。

翻訳コマンドの指定場所および基本的な使い方は、“[3.7 翻訳コマンド](#)”を参照してください。

3.7.1 COBOL32コマンド

COBOL32コマンドは、翻訳操作をコマンドプロンプト上で行うためのコマンドです。

入力形式

コマンド	オペランド
COBOL32	[オプションの並び] ファイル名 …

オペランドの説明

コマンド名および各オペランドの間には、1つ以上の空白が必要です。[]で囲んである部分は省略できることを示します。なお、以降の説明で、フォルダ名およびファイル名には、絶対パス名または相対パス名を指定することができます。また、以下の文字を含む場合には、フォルダ名またはファイル名を二重引用符(“)で囲む必要があります。

空白、「+」、「,」、「;」、「=」、「[」、「]」、「(」、「)」、「'」



指定例

-I"C:\COMMON\COPY LIBRARIES\TEMPLATE"

オプションの並び

COBOLコンパイラに通知する情報を指定します。

指定する内容は、“[3.8.1 オプション一覧](#)”を参照してください。

オプションの指定方法による優先順位は、“[A.2 翻訳オプションの指定形式](#)”の“翻訳オプションの指定方法と優先順位”を参照してください。

ファイル名

翻訳するソースファイルの名前を指定します。ファイル名は複数個指定することができます。

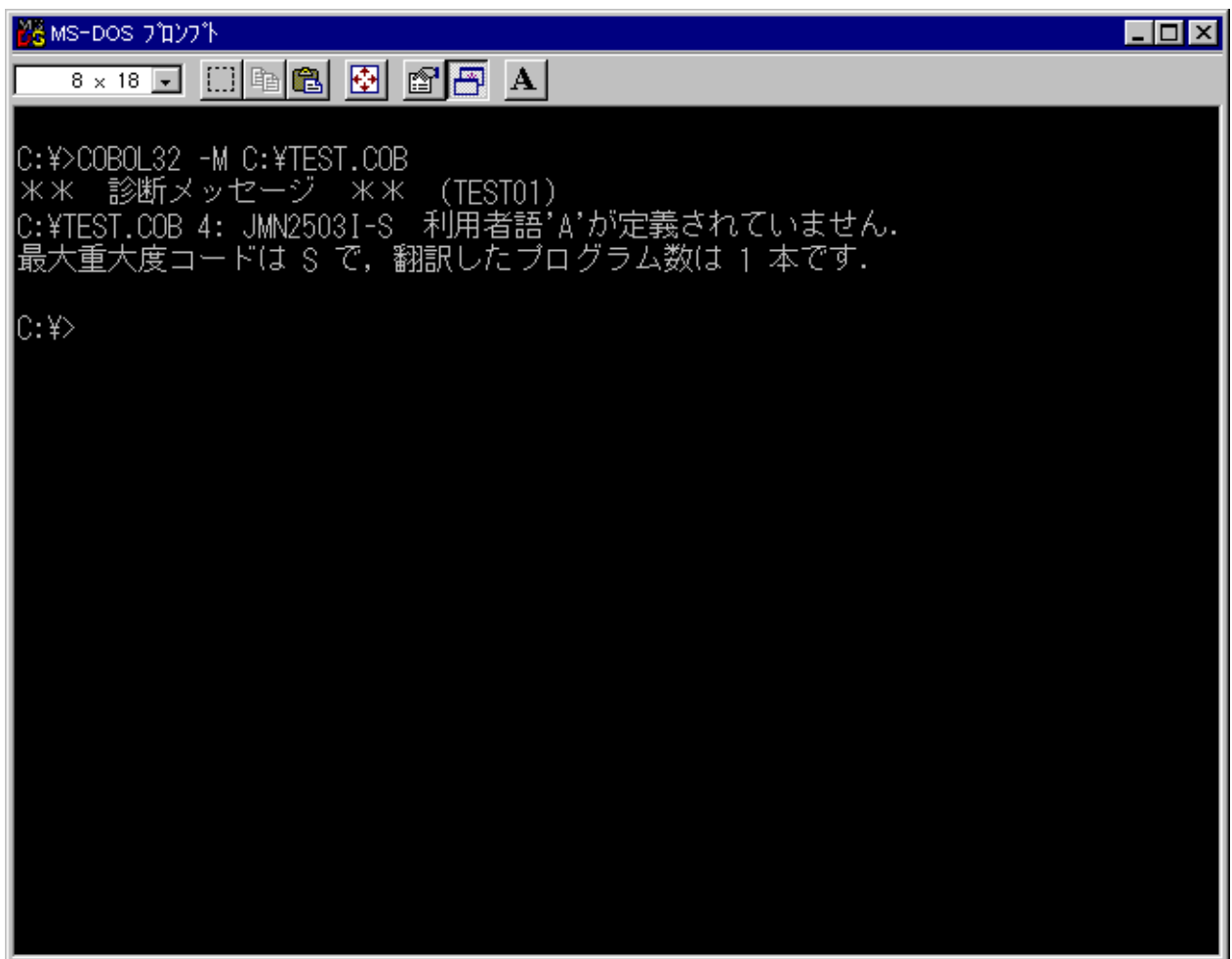
出力形式

COBOL32コマンドによるコマンドプロンプト画面での翻訳では、翻訳結果や診断メッセージなどの翻訳情報は、コマンドプロンプト画面に表示されます。

以下は、COBOL32コマンドによる翻訳操作およびその出力例です。

通常の出力形式

翻訳オプションMESSAGEを指定しない場合、通常の出力形式として翻訳終了メッセージが出力されます。また、翻訳時にエラーが発生した場合は、その診断メッセージも出力されます。



```
MS-DOS プロンプト
8 x 18
C:\>COBOL32 -M C:\¥TEST.COB
*** 診断メッセージ *** (TEST01)
C:\¥TEST.COB 4: JMN2503I-S 利用者語'A'が定義されていません。
最大重大度コードは S で、翻訳したプログラム数は 1 本です。

C:\>
```

各種情報の出力

翻訳オプションMESSAGEを指定した場合、オプション情報リストと翻訳単位統計情報リストが出力されます。また、翻訳時にエラーが発生した場合は、その診断メッセージも出力されます。

```

C:\>COBOL32 -M -WC, "MESSAGE" TEST.COB
** 指定翻訳オプション **
MAIN, MESSAGE
** 確定翻訳オプション **
ALPHAL (ALL)          NOLIST          NOSOURCE
BINARY (WORD, MLBON)  MAIN (WINMAIN)  SQLGRP
NOCHECK              MESSAGE          SRF (VAR, VAR)
NOCONF              NONAME            SSIN (SYSIN)
NOCOPY              NCW (STD)         SSOUT (SYSOUT)
NOCOUNT            NSPCOMP (NSP)      STD1 (JIS2)
CREATE (OBJ)        NONUMBER         TAB (8)
CURRENCY (¥)        OBJECT           NOTEST
NODLOAD            NOOPTIMIZE        THREAD (SINGLE)
NOEQUALS           QUOTE             NOTRACE
FLAG (I)           RCS (SJIS)        NOTRUNC
NOFLAGSW           RSV (ALL)         NOXREF
LANGVL (85)        NOSAI            ZWB
LINECOUNT (60)    SDS
LINESIZE (136)     NOSHREXT
** 診断メッセージ ** (TEST01)
TEST.COB 4: JMN2503I-S 利用者語 'A' が定義されていません.
** プログラム特性リスト **
ファイル名 = TEST.COB
ソース名   = TEST01
翻訳日付   = 2000年10月20日(金) 13時18分20秒

```

WINCOBによる翻訳操作との違い

COBOL32コマンドによる翻訳操作は、WINCOBによる翻訳操作と以下の点が異なります。

- 翻訳の中断は、通常のDOSコマンドと同様の方法で行います。

COBOL32コマンドの復帰コード

COBOL32コマンドの復帰コードは、プログラム翻訳時の最大重大度コードにより設定されます。以下に最大重大度コードおよび復帰コードの関係を示します。

最大重大度コード	復帰コード
I	0
W	
E	1
S	2
U	3

3.8 翻訳コマンドのオプション

3.8.1 オプション一覧

以下に、翻訳コマンドのオプションの一覧を示します。

翻訳の資源に関するもの

- “3.8.1.1 -A(サブスキーマ定義ファイルのフォルダの指定)”
- “3.8.1.8 -dr(リポジトリファイルの入出力先フォルダの指定)”
- “3.8.1.11 -f(ファイル定義体ファイルのフォルダの指定)”
- “3.8.1.12 -I(登録集ファイルのフォルダの指定)”
- “3.8.1.15 -m(画面帳票定義体ファイルのフォルダの指定)”
- “3.8.1.18 -R(リポジトリファイルの入力先フォルダの指定)”

翻訳リストに関するもの

- “3.8.1.6 -dp(翻訳リストファイルのフォルダの指定)”
- “3.8.1.17 -P(各種翻訳リストの出力および出力先の指定)”

ソースプログラムの解釈に関するもの

- “3.8.1.9 -ds(ソース解析情報ファイルのフォルダの指定)”

目的プログラムの作成に関するもの

- “3.8.1.5 -do(オブジェクトファイルのフォルダの指定)”
- “3.8.1.14 -M(主プログラムを翻訳するときの指定)”
- “3.8.1.16 -O(広域最適化を適用する指定)”

実行時のデバッグ機能に関するもの

- “3.8.1.2 -Dc(COUNT機能を使用する指定)”
- “3.8.1.3 -dd(デバッグ情報ファイルのフォルダの指定)”
- “3.8.1.4 -Dk(CHECK機能を使用する指定)”
- “3.8.1.7 -Dr(TRACE機能を使用する指定)”
- “3.8.1.10 -Dt(対話型デバッガおよび診断機能を使用する指定)”

その他

- “3.8.1.13 -i(オプションファイルの指定)”
- “3.8.1.19 -WC(翻訳オプションの指定)”

3.8.1.1 -A(サブスキーマ定義ファイルのフォルダの指定)

-Aフォルダ

サブスキーマ段落にサブスキーマ定義ファイルを記述した場合、-Aオプションにサブスキーマ定義ファイルの存在するフォルダ指定します。使用するサブスキーマ定義ファイルが複数のフォルダに存在する場合、-Aオプションを複数指定します。-Aオプションを複数指定した場合、指定した順序でフォルダを検索します。



参考

- -Aオプションは、翻訳オプションAIMLIBおよび環境変数COB_AIMLIBと同じ意味です。
- オプションの優先順位は、“A.2.1 AIMLIB(サブスキーマ定義ファイルのフォルダの指定)”を参照してください。

3.8.1.2 -Dc(COUNT機能を使用する指定)

-Dc

COUNT機能を使用する場合、-Dcオプションを指定します。

COUNT機能については、“[19.4 COUNT機能の使い方](#)”を参照してください。



注意

-Dcオプションを指定すると、COUNT情報を出力するための処理が目的プログラム中に組み込まれるため、実行性能が低下します。デバッグ終了後は、-Dcオプションを指定しないで再翻訳してください。



参考

-Dcオプションは、翻訳オプションCOUNTと同じ意味です。



参照

“[A.2.8 COUNT \(COUNT機能の使用の可否\)](#)”

3.8.1.3 -dd(デバッグ情報ファイルのフォルダの指定)

-dd フォルダ

-ddオプションは、-Dtオプションまたは翻訳オプションTESTを指定した場合だけ意味を持ちます。

-ddオプションを省略した場合、-Dtオプションまたは翻訳オプションTESTの出力規則に従ってデバッグ情報ファイルが出力されます。



参考

-ddオプションは翻訳オプションTESTの出力先フォルダ指定に相当します。-ddオプションと翻訳オプションTESTの出力先フォルダが同時に指定されている場合、-ddの指定が優先されます。



参照

“[3.8.1.10 -Dt \(対話型デバッガおよび診断機能を使用する指定\)](#)”

“[A.2.56 TEST \(対話型デバッガおよび診断機能の使用の可否\)](#)”

3.8.1.4 -Dk(CHECK機能を使用する指定)

-Dk

CHECK機能を使用する場合、-Dkオプションを指定します。

CHECK機能については、“[19.2 CHECK機能の使い方](#)”を参照してください。



注意

-Dkオプションを指定すると、添字・指標および部分参照に関する検査を行うための処理が目的プログラム中に組み込まれるため、実行性能が低下します。デバッグ終了後は、-Dkオプションを指定しないで再翻訳してください。



参考

-Dkオプションは、翻訳オプションCHECKと同じ意味です。



参照

“A.2.5 CHECK (CHECK機能の使用の可否)”

3.8.1.5 -do (オブジェクトファイルのフォルダの指定)

-do フォルダ

-doオプションは、翻訳オプションOBJECTが有効な場合だけ意味を持ちます。

-doオプションを省略した場合、翻訳オプションOBJECTの出力規則に従ってオブジェクトファイルが出力されます。



参考

-doオプションは翻訳オプションOBJECTの出力先フォルダ指定に相当します。翻訳オプションOBJECTの出力先フォルダが同時に指定されている場合、-doの指定が優先されます。



参照

“A.2.36 OBJECT (目的プログラムの出力の可否)”

3.8.1.6 -dp (翻訳リストファイルのフォルダの指定)

-dp フォルダ

-dpオプションは、-Pオプションまたは翻訳オプションPRINTを指定した場合だけ意味を持ちます。

-dpオプションを指定した場合、翻訳リストファイルは以下のようになります。

-Pオプションにファイル名の指定がある場合

翻訳リストファイル名は、“-dpオプションで指定したフォルダ名”に“-Pオプションで指定したファイル名”を結合した名前になります。



例

```
cobol32 -Pout.LST -dpd:%test cobtest.cob
→ d:%test%out.LST
```

-Pオプションにファイル名の指定がない場合

-dpオプションで指定したフォルダ名に“ソースファイル名.LST”を結合した名前になります。



例

```
cobol32 -P -dpd:%test cobtest.cob
→ d:%test%cobtest.LST
```

-dpオプションを省略した場合、翻訳リストファイルは-Pオプションまたは翻訳オプションPRINTの出力規則に従って出力されます。



参考

-dpオプションは翻訳オプションPRINTの出力先フォルダ指定に相当します。翻訳オプションPRINTに出力先フォルダが同時に指定されている場合、-dpの指定が優先されます。



参照

“3.8.1.17 -P(各種翻訳リストの出力および出力先の指定)”

“A.2.38 PRINT(各種翻訳リストの出力の可否および出力先の指定)”

3.8.1.7 -Dr(TRACE機能を使用する指定)

-Dr

TRACE機能を使用する場合、-Drオプションを指定します。

TRACE機能については、“19.3 TRACE機能の使い方”を参照してください。



注意

-Drオプションを指定すると、トレース情報を出力するための処理が目的プログラム中に組み込まれるため、実行性能が低下します。デバッグ終了後は、-Drオプションを指定しないで再翻訳してください。



参考

-Drオプションは、翻訳オプションTRACEと同じ意味です。



参照

“A.2.58 TRACE(TRACE機能の使用の可否)”

3.8.1.8 -dr(リポジトリファイルの入出力先フォルダの指定)

-dr フォルダ

-drオプションは、クラス定義の翻訳でだけ意味を持ちます。

-drオプションを省略した場合、翻訳オプションREPの出力規則に従ってリポジトリファイルが出力されます。

また、-drオプションで指定されたフォルダは、外部リポジトリを取り込む場合の入力先フォルダとしても使用されます。



参考

-drオプションは翻訳オプションREPに相当します。翻訳オプションREPが同時に指定されている場合、リポジトリファイルの出力フォルダは-drの指定が優先されます。



参照

“A.2.41 REP(リポジトリファイルの入出力先フォルダの指定)”

3.8.1.9 -ds(ソース解析情報ファイルのフォルダの指定)

-ds フォルダ

-ds オプションは、翻訳オプションSAIを指定した場合だけ意味を持ちます。

-ds オプションを省略した場合、翻訳オプションSAIの出力規則に従ってソース解析情報ファイルが出力されます。



参考

-ds オプションは翻訳オプションSAIの出力先フォルダ指定に相当します。翻訳オプションSAIの出力先フォルダが同時に指定されている場合、-dsの指定が優先されます。



参照

“A.2.44 SAI(ソース解析情報ファイルの出力の可否)”

3.8.1.10 -Dt(対話型デバグおよび診断機能を使用する指定)

-Dt

-Dt オプションを指定すると、対話型デバグや診断機能で使用するデバグ情報ファイルが作成され、通常はソースプログラムと同じフォルダに格納されます。変更したい場合は、-dd オプションで格納先フォルダを指定してください。

対話型デバグについては、“第20章 対話型デバグの使い方”を参照してください。

診断機能については、“19.6 診断機能の使い方”を参照してください。



参考

-Dt オプションは、翻訳オプションTESTと同じ意味です。



参照

“3.8.1.3 -dd(デバグ情報ファイルのフォルダの指定)”

“A.2.56 TEST(対話型デバグおよび診断機能の使用の可否)”

3.8.1.11 -f(ファイル定義体ファイルのフォルダの指定)

-f フォルダ

IN/OF XFDLIB指定のCOPY文によりファイル定義体からレコード定義を取り込む場合、-f オプションにファイル定義体ファイルが存在するフォルダを指定します。使用するファイル定義体ファイルが複数のフォルダに存在する場合、-f オプションを複数指定します。-f オプションを複数指定した場合、指定した順序でフォルダを検索します。



参考

- ・ -f オプションは、翻訳オプションFILELIBおよび環境変数FILELIBと同じ意味です。
- ・ オプションの優先順位は、“A.2.15 FILELIB(ファイル定義体ファイルのフォルダの指定)”を参照してください。

3.8.1.12 -l(登録集ファイルのフォルダの指定)

-l フォルダ

ソースプログラムでCOPY文を使用している場合、-IオプションにソースプログラムのCOPY文に記述した登録集ファイルが存在するフォルダを指定します。使用する登録集ファイルが複数のフォルダに存在する場合、-Iオプションを複数指定します。-Iオプションを複数指定した場合、指定した順序でフォルダを検索します。



注意

IN/OF 登録集名指定のCOPY文の場合、-Iオプションの指定は無視されます。

登録集名を設定したオプションファイルまたは環境変数COB_登録集名を使用してください。[参照]“3.5.2 登録集名の設定”



参考

- ・ -Iオプションは、翻訳オプションLIBおよび環境変数COB_COBCOPYと同じ意味です。
- ・ オプションの優先順位については、“A.2.23 LIB(登録集ファイルのフォルダの指定)”を参照してください。

3.8.1.13 -i(オプションファイルの指定)

-iファイル

翻訳オプションをオプションファイル(翻訳オプションの文字列が格納されているファイル)で指定する場合、-iオプションにオプションファイルのファイル名を指定します。オプションファイルは、WINCOBコマンドを使って、リストボックスからの選択操作により簡単に作成することができます。オプションファイルを作成するときの、WINCOBコマンドの指定形式を以下に示します。

WINCOB -iオプションファイルのパス名

3.8.1.14 -M(主プログラムを翻訳するときの指定)

-M

実行時に主プログラムとなるソースプログラムの翻訳を行う場合、-Mオプションを指定します。



参考

-Mオプションは、翻訳オプションMAINと同じ意味です。



参照

“A.2.28 MAIN(主プログラム/副プログラムの指定)”

3.8.1.15 -m(画面帳票定義体ファイルのフォルダの指定)

-mフォルダ

IN/OF XMDLIB指定のCOPY文により画面帳票定義体からレコード定義を取り込む場合、-mオプションに画面帳票定義体ファイルが存在するフォルダを指定します。使用する画面帳票定義体ファイルが複数のフォルダに存在する場合、-mオプションを複数指定します。-mオプションを複数指定した場合、指定した順序でフォルダを検索します。



注意

画面帳票定義体ファイルの絶対パス名は、シフトJISで256バイト以内の長さでなければなりません。



参考

- ・ `-m` オプションは、翻訳オプション `FORMLIB` および環境変数 `FORMLIB` と同じ意味です。
- ・ オプションの優先順位は、“[A.2.19 FORMLIB \(画面帳票定義体ファイルのフォルダの指定\)](#)”を参照してください。

3.8.1.16 -O (広域最適化を適用する指定)

-O

広域最適化された目的プログラムを作成する場合、`-O` オプションを指定します。広域最適化については、“[付録C 広域最適化](#)”を参照してください。



参考

`-O` オプションは、翻訳オプション `OPTIMIZE` と同じ意味です。



参照

“[A.2.37 OPTIMIZE \(広域最適化の扱い\)](#)”

3.8.1.17 -P (各種翻訳リストの出力および出力先の指定)

-Pファイル

翻訳リストファイルを出力する場合、`-P` オプションに出力先のファイル名を指定します。ファイル名には、拡張子 `COB`、`CBL`、`COBOL` を使用してはいけません。ファイル名を省略した場合、翻訳リストファイルは、ソースファイル名 `.LST` として、ソースファイルと同じフォルダに出力されます。



注意

- ・ `-dp` オプションを同時に指定した場合、`-dp` オプションに指定されたフォルダを結合したファイル名が最終的な翻訳リストファイル名となるため、`-P` オプションに指定するファイル名には、絶対パス名を指定できません。
- ・ `-P` オプションのファイル名を省略する場合、`-P` オプションの直後にソースファイル名を指定してはいけません。`-P` オプションとソースファイル名の間には、1つ以上の他のオプションを指定してください。

×: `COBOL32 -P ソースファイル名`

○: `COBOL32 -P -WC, " OBJECT" ソースファイル名`

3.8.1.18 -R (リポジトリファイルの入力先フォルダの指定)

-Rフォルダ

リポジトリ段落の指定により、外部リポジトリを取り込む場合、`-R` オプションにリポジトリファイルが存在するフォルダを指定します。使用するリポジトリファイルが複数のフォルダに存在する場合、`-R` オプションを複数指定します。`-R` オプションを複数指定した場合、指定した順序でフォルダを検索します。



参考

- ・ `-R` オプションは、翻訳オプション `REPIN` および環境変数 `COB_REPIN` と同じ意味です。
- ・ オプションの優先順位は、“[A.2.42 REPIN \(リポジトリファイルの入力先フォルダの指定\)](#)”を参照してください。

3.8.1.19 -WC(翻訳オプションの指定)

-WC, "翻訳オプション"

COBOLコンパイラに指示する翻訳オプションを指定する場合、「-WC,"翻訳オプション"」の形式で指定することができます。
翻訳オプションの内容および指定形式は、「[付録A 翻訳オプション](#)」を参照してください。

翻訳オプションは複数個指定することができ、各翻訳オプションの間は1つのコンマ(,)で区切ります。同一の翻訳オプションが複数個指定された場合には、最後に指定した翻訳オプションが有効となります。

翻訳オプションの指定方法によるオプションの優先順位は、「[A.2 翻訳オプションの指定形式](#)」の「翻訳オプションの指定方法と優先順位」を参照してください。

注意

- -WCでは、以下のオプションを指定することはできません。
 - AIMLIB
 - FILEEXT
 - FORMEXT
 - LIB
 - PRINT
 - REP
 - REPIN
 - LIBEXT
 - FILELIB
 - FORMLIB
- 以下のオプションを-WCに指定する場合、フォルダを指定してはなりません。フォルダの指定は、各コマンドオプションを使用してください。
 - OBJECT
 - SAI
 - TEST

例

- ×: -WC, " OBJECT (フォルダ)"
- : -WC, " OBJECT" -doフォルダ

第4章 プログラムをリンクする

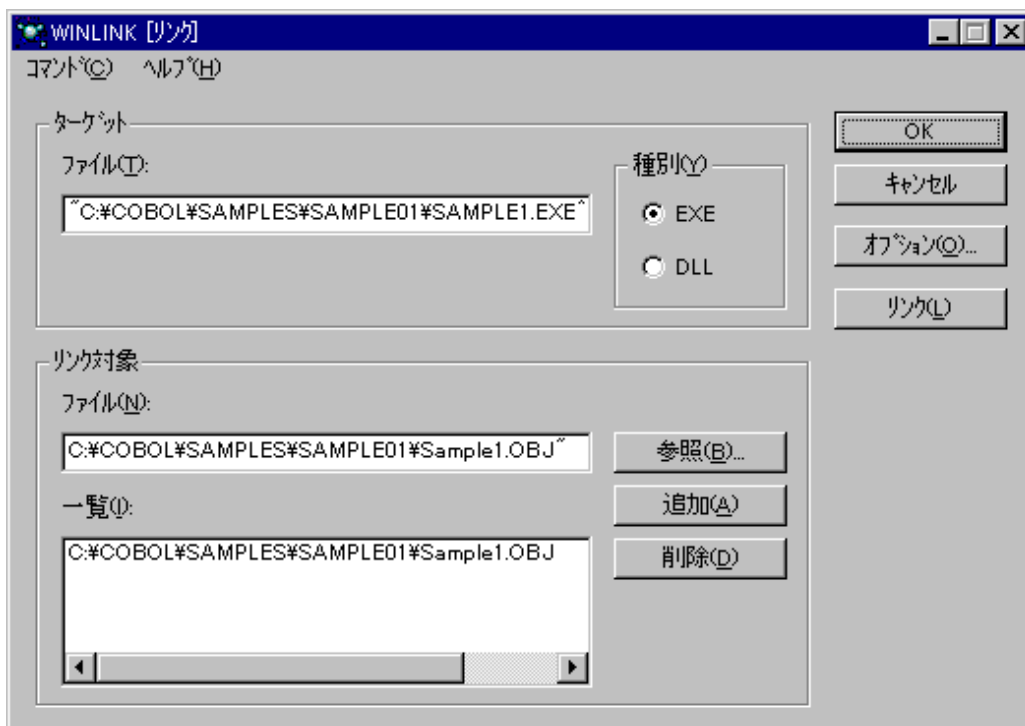
本章では、まず、サンプルプログラムのリンク手順について説明し、以降、必要な資源、プログラム構造、手順、使用するウィンドウの使い方などリンク時に必要な作業について説明します。

4.1 サンプルプログラムのリンク

翻訳が終わったら、作成された目的プログラムをリンクしてみましょう。

ここでは、WINLINKを使って、例題1のプログラムをリンクします。なお、以下の説明では、オブジェクトファイルをC:\¥COBOL¥SAMPLES¥SAMPLE01¥SAMPLE1.OBJとします。

1. プロジェクトマネージャウィンドウの[ツール]メニューから“リンク”を選択します。
→ WINLINKウィンドウが表示されます。
2. 必要な情報を設定します。



- a. 種別で“EXE”を選択します。
 - b. ファイル(N)のエディットボックスにオブジェクトファイルを指定します。
 - c. [追加]ボタンをクリックします。
→ オブジェクトファイル名が一覧のリストボックスに表示されます。
→ 作成される実行可能プログラム名がファイル(T)に表示されます。
3. WINLINKウィンドウの[リンク]ボタンをクリックします。
→ ビルダウィンドウに切り替わり、リンク中のメッセージが表示されます。
 4. リンクが終了します。
→ ビルダウィンドウにビルド終了のメッセージが表示されます。リンクが正常に行われたことを確認したら、ビルダウィンドウを閉じてください。

以上で、サンプルプログラムのリンクは終わりです。サンプルプログラムの格納先と同じフォルダ(本書の説明の場合、C:\¥COBOL¥SAMPLES¥SAMPLE01)に、実行可能プログラム(SAMPLE1.EXE)が作成されているか確認してください。

4.2 リンクに必要な資源

COBOLソースプログラムを翻訳して作成した目的プログラムのリンクは、通常WINLINKを使って行います。WINLINKを使うと、ライブラリ、インポートライブラリ、ダイナミックリンクライブラリ(以降の説明では単にDLLといいます)および実行可能ファイルを、簡単な画面操作によって作成することができます。ここでは、リンクに必要な各ファイルについて簡単に説明します。

リンクを行う場合には、プログラム構造を考慮してリンクする必要があります。[参照]“[4.3 プログラム構造](#)”

注意

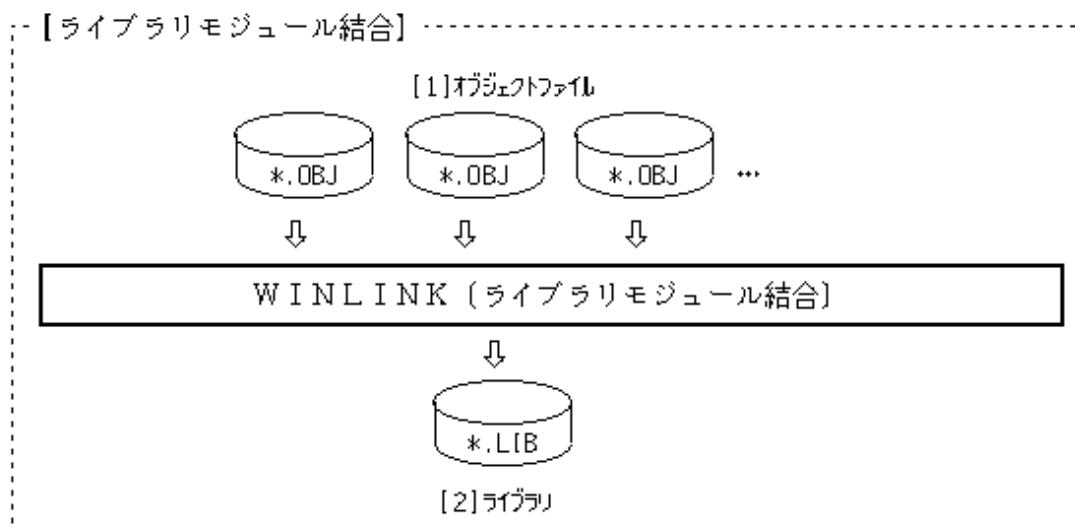
WINLINKを使ってリンクを行う場合、環境変数TMPに作業用のフォルダを設定しておく必要があります。

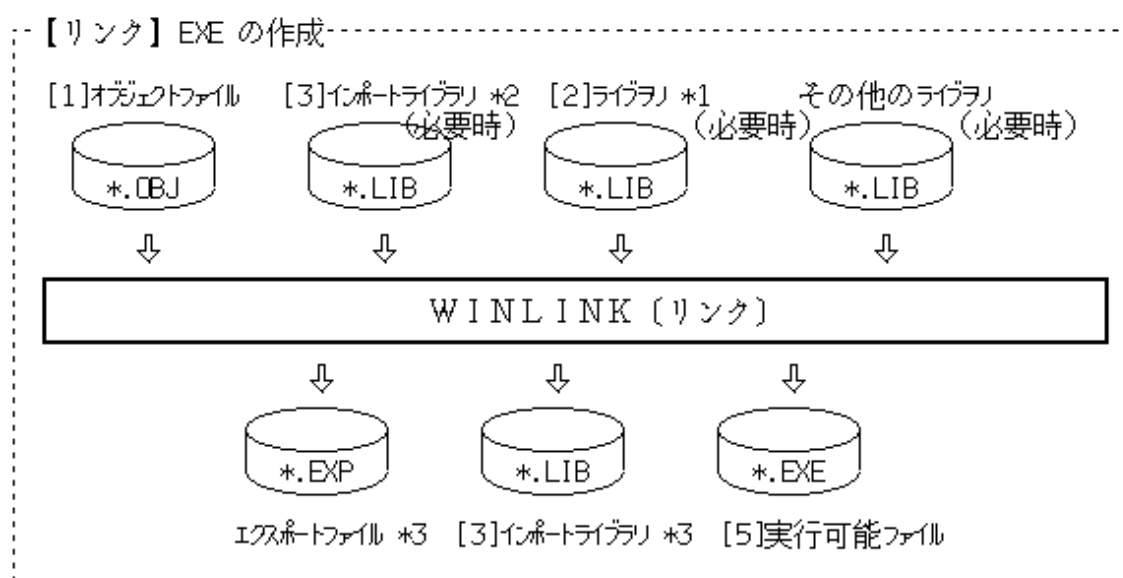
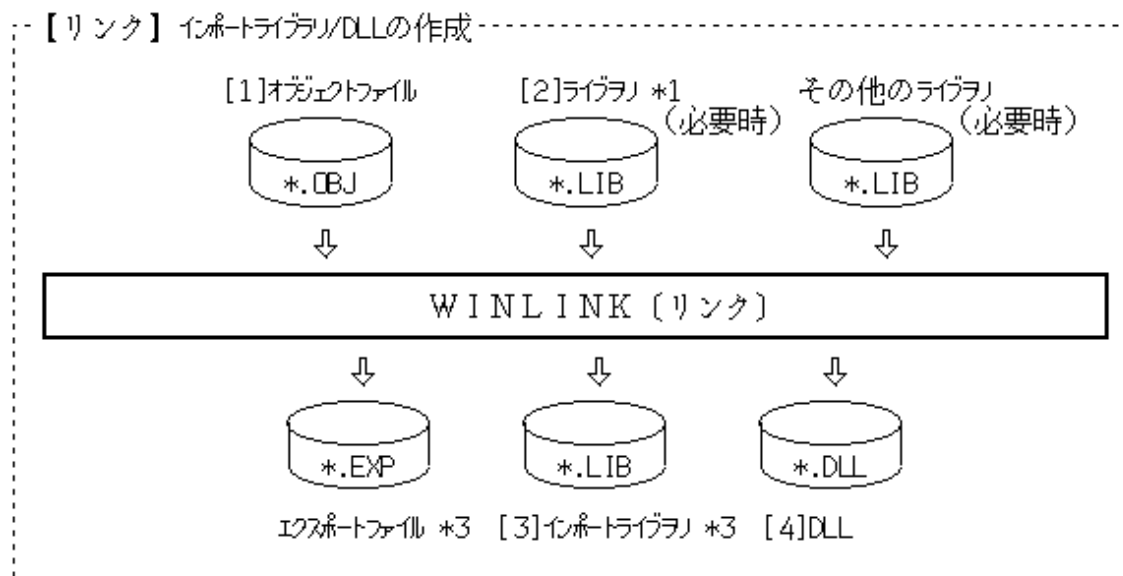
4.2.1 WINLINKでリンクを行う場合

WINLINKは、以下のファイルを使用します。

- ・ オブジェクトファイル(*.OBJ)
- ・ ライブラリ(*.LIB)
- ・ インポートライブラリ(*.LIB)
- ・ エクスポートファイル(*.EXP)
- ・ ダイナミックリンクライブラリ(*.DLL)
- ・ 実行可能ファイル(*.EXE)

以下に、WINLINKが使用するファイルをWINLINKの機能別に示します。





*1: WINLINK[ライブラリモジュール結合] ウィンドウで作成しておきます。

*2: 動的リンク構造の場合に必要となります。インポートライブラリ/DLL の作成を行い、あらかじめ作成しておく必要があります。

*3: EXE の作成時に作成されたインポートライブラリおよびエクスポートファイルは不要となります。

WINLINKが使用するファイルを、“[表4.1 WINLINKが使用するファイル](#)”に示します。

表4.1 WINLINKが使用するファイル

	ファイルの内容	ファイル名の形式	入出力	使用する条件または作成する条件
[1]	目的プログラム	ソースファイル名.OBJ	入力	COBOL ソースプログラムを翻訳して作成した目的プログラムを使用します。
[2]	標準ライブラリ(オブジェクトコードライブラリ)	任意の名前.LIB	入力	DLL または実行可能ファイルを作成するときに必要な場合、指定してください。
		オブジェクトファイル名.LIB または 任意の名前.LIB	出力	WINLINK[ライブラリモジュール結合] を実行すると作成されます。

	ファイルの内容	ファイル名の形式	入出力	使用する条件または作成する条件
[3]	インポートライブラリ	任意の名前.LIB	入力	動的リンク構造の実行可能ファイルを作成する場合に指定します。
		DLL 名.LIB	出力	WINLINK[リンク] でDLL を選択すると、自動的に作成されます。
[4]	ダイナミックリンクライブラリ(DLL)	オブジェクトファイル名.DLL または 任意の名前.DLL	出力	リンクが成功した場合に作成されます。
[5]	実行可能プログラム	オブジェクトファイル名.EXE または 任意の名前.EXE	出力	リンクが成功した場合に作成されます。



注意

リンクオプション"/DEBUG、/DEBUGTYPE:CV"または"/DEBUG、/DEBUGTYPE:BOTH"を指定すると“.pdbファイル”が生成されます。このファイルは、ワトソン博士のログ情報やVisual C++のデバッグ機能などが使用します。

4.2.2 実行可能ファイル

COBOLソースプログラムを翻訳して作成した目的プログラムをリンクして作成した実行可能ファイルが、32ビットWindows上で動作するアプリケーション(以降、アプリケーションと略します)となります。

4.2.3 DLL(ダイナミックリンクライブラリ)

DLLは、アプリケーションが呼び出す何らかの処理を行う関数を含んだ実行可能なモジュールです。DLLがアプリケーションとリンクされるのは、リンク時ではなく実行時です。DLLを利用すれば、複数のアプリケーションが同一のライブラリを使用することができます。

4.2.4 インポートライブラリ

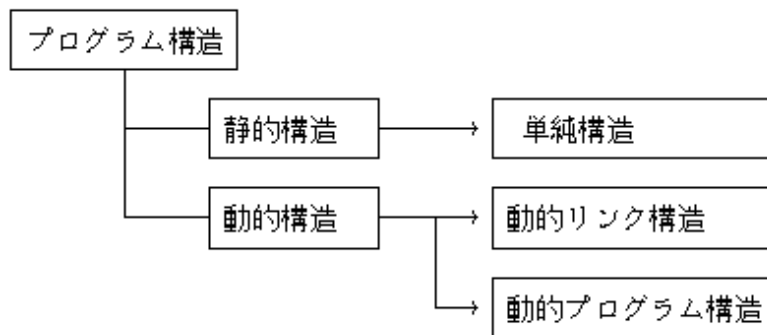
インポートライブラリは、アプリケーション実行時に、実行中のアプリケーションとDLLとの間にダイナミックリンクを確立するための情報を含んでいます。動的リンク構造で実行可能ファイルを作成する場合には、必ずインポートライブラリが必要となります。必要なライブラリが実行時にどこにあるかを探すための情報をインポートライブラリから実行可能プログラムにコピーします。つまり、インポートライブラリはアプリケーションとDLLとの仲立ちの役目をするものです。

4.3 プログラム構造

ここでは、リンクによって作成される実行可能プログラムの構造について説明します。

プログラム構造には、“[図4.1 プログラム構造](#)”に示すような構造があります。以下に、それぞれの構造について説明します。なお、以降の説明で、主プログラムとは、最初に動作するプログラムのことをいい、副プログラムとは、主プログラムから呼ばれるプログラムのことをいいます。また、副プログラムから呼ばれるプログラムも、副プログラムといいます。

図4.1 プログラム構造

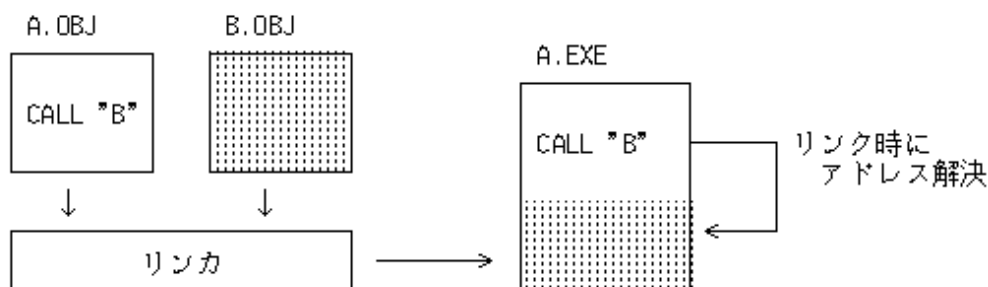


4.3.1 静的構造

静的構造とは、実行時に必要となるモジュールを、リンカによって、単一の実行可能ファイルとし、実行させる構造です。

単純構造

単純構造とは、目的プログラムとその目的プログラムから呼び出されている副プログラムの目的プログラムを、リンクによって1つの実行可能ファイルにしたものです。したがって、実行開始時に、主プログラムおよび副プログラムのすべてが仮想記憶上にロードされ、副プログラムを呼び出すときの効率がよくなります。ただし、単純構造の実行可能ファイルを作成するときには、リンク時にすべての副プログラムを必要とします。



4.3.2 動的構造

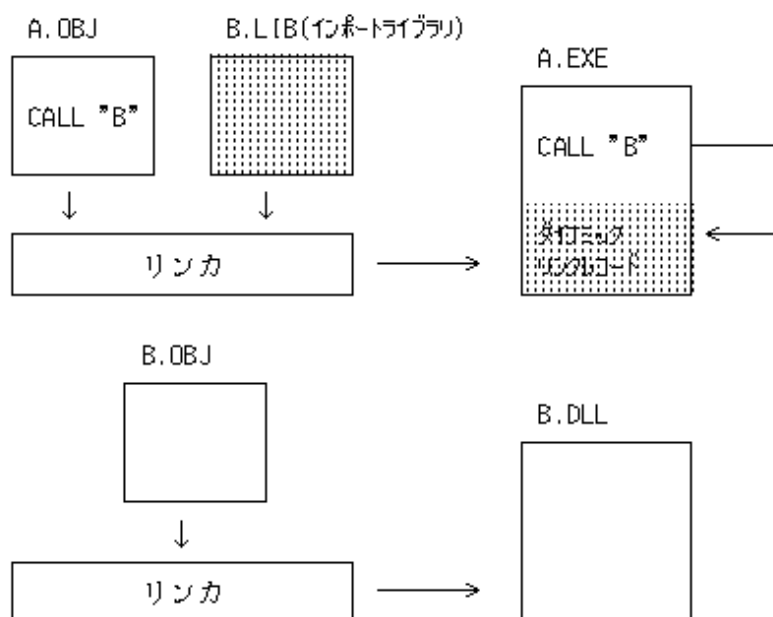
動的構造とは、実行時に必要となるモジュールを、リンカによっていくつかのDLLとし、プログラムの実行時に必要なDLLを仮想記憶上にロードし、実行させる構造です。

DLLに含まれるモジュールは、そのモジュールを利用するすべてのプログラムによって共用されるため、ファイルを格納するディスク空間、および複数のプログラムを実行したときの仮想記憶の節約となります。また、プログラムを変更した場合、変更したプログラムを含むDLLだけを再リンクすればよいので、プログラムの変更が容易に行えます。

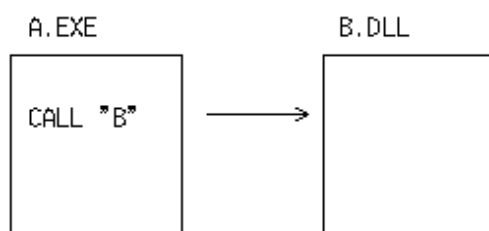
動的リンク構造

動的リンク構造とは、目的プログラムとその目的プログラムから呼び出されている副プログラムのインポートライブラリを、リンクすることによって1つの実行可能ファイルにしたものです。

動的リンク構造では、単純構造と異なり、実行可能ファイル中に副プログラムは含まれません。主プログラムがロードされたときに、副プログラムも仮想記憶上にロードされます。ローディングは、リンク時に実行可能ファイル中に生成される副プログラム情報を使って、システムのダイナミックリンクが行います。動的リンク構造の実行可能ファイルを作成するときには、リンク時にそのプログラムが呼び出すすべての副プログラムのインポートライブラリが必要となります。



実行時にダイナミックリンクがアドレス解決



動的プログラム構造

動的プログラム構造とは、動的リンク構造と異なり、副プログラムのインポートライブラリを使用しないで、目的プログラムだけをリンクすることによって、1つの実行可能ファイルにしたものです。そのため、動的リンク構造のように、実行可能ファイル中に副プログラムの情報を含みません。動的プログラム構造の詳細については、“[10.1.5 動的プログラム構造](#)”を参照してください。

4.3.3 プログラム構造とCALL文の書き方および翻訳オプションの関係

プログラム構造は、CALL文の書き方および翻訳時に指定する翻訳オプションで決定されます。プログラム構造とCALL文および翻訳オプションの関係を“[表4.2 CALL文の書き方/翻訳オプションとプログラム構造の関係](#)”に示します。なお、翻訳オプションDLOADについては、“[A.2.11 DLOAD \(プログラム構造の指定\)](#)”を参照してください。

表4.2 CALL文の書き方/翻訳オプションとプログラム構造の関係

		翻訳オプション	
		DLOAD	NODLOADまたは省略
CALL文の書き方	CALL "プログラム名"	動的プログラム構造	単純構造または動的リンク構造(注1)
	CALL データ名	動的プログラム構造	動的プログラム構造
	CALL "プログラム名"	動的プログラム構造	動的リンク構造 (注2)
	CALL データ名の混在		動的プログラム構造 (注3)

- 注1：動的リンク構造ではリンク時に呼び出し先のDLLが必要です。
- 注2：プログラム名指定の呼び出しは動的リンク構造となります。
- 注3：データ名指定の呼び出しは動的プログラム構造となります。

4.3.4 プログラム構造とCANCEL文の関係

CANCEL文は、二回目以降に呼び出されたプログラムの状態を初期化します。ただし、プログラム構造によっては、CANCEL文が有効にならない場合があるため、CANCEL文を使用する場合には注意してください。

プログラム構造とCANCEL文の関係を以下に示します。

表4.3 プログラム構造とCANCEL文の関係

プログラム構造		CANCEL文
外部プログラム	単純構造	無効
	動的リンク構造	無効
	動的プログラム構造	有効
内部プログラム		有効

内部プログラムの詳細については、“[10.2.7 内部プログラム](#)”を参照してください。

4.4 リンクの手順

COBOLソースプログラムを翻訳して作成した目的プログラム(以降の説明ではCOBOL目的プログラムといいます)のリンクは、WINLINKコマンドを実行してウィンドウ操作により行う方法と、リンクコマンドを使ってコマンドプロンプトから指定する方法があります。

ここでは、WINLINKウィンドウを使ってリンクするときの手順について説明します。リンクコマンドを使ったリンク操作については、“[4.7 リンクコマンドを使ったリンク操作](#)”で説明しています。

4.4.1 1つの目的プログラムをリンクするときの手順

ここでは、副プログラムを呼び出さない実行可能プログラムを作成するときの手順を説明します。

1. WINLINK[リンク]ウィンドウを起動します。[参照]“[4.5.1 WINLINKウィンドウの起動](#)”
2. 種別で“EXE”を選択します。
3. ターゲット名を指定します。[参照]“[4.5.2.1 ターゲット名の指定](#)”
リンクする主プログラムのオブジェクトファイルを一覧のリストボックスに指定します。
→ “オブジェクトファイル名.EXE”がファイル(T)のエディットボックスに表示されます。
4. リンクに必要なその他のファイルを一覧のリストボックスに指定します。[参照]“[4.5.2.2 ファイルの指定](#)”
5. リンクオプションを設定します。[参照]“[4.5.2.4 リンクオプションの設定](#)”
6. リンクを開始します。[参照]“[4.5.2.5 リンクの開始と終了](#)”
7. リンクを終了します。[参照]“[4.5.2.5 リンクの開始と終了](#)”

4.4.2 DLLを作成するときの手順

ここでは、副プログラムのCOBOL目的プログラムをリンクして、DLLを作成するときの手順を説明します。

1. WINLINK[リンク]ウィンドウを起動します。[参照]“[4.5.1 WINLINKウィンドウの起動](#)”
2. 種別で“DLL”を選択します。
3. ターゲット名を指定します。[参照]“[4.5.2.1 ターゲット名の指定](#)”
リンクする副プログラムのオブジェクトファイルの名前を一覧のリストボックスに指定します。
→ “オブジェクトファイル名.DLL”がファイル(T)のエディットボックスに表示されます。



注意

翻訳時に翻訳オプションMAINを指定して作成したオブジェクトファイルを指定してはいけません。

4. リンクに必要なその他のファイルを一覧のリストボックスに指定します。[参照]“4.5.2.2 ファイルの指定”
5. リンクオプションを設定します。[参照]“4.5.2.4 リンクオプションの設定”
6. リンクを開始します。[参照]“4.5.2.5 リンクの開始と終了”
7. リンクを終了します。[参照]“4.5.2.5 リンクの開始と終了”

4.4.3 単純構造の実行可能プログラムを作成するときの手順

ここでは、副プログラムを呼び出すCOBOL目的プログラムをリンクして、単純構造の実行可能プログラムを作成するときの手順を説明します。なお、COBOL目的プログラムは、データ名指定のCALL文を記述していないCOBOLソースプログラムを翻訳して作成したもの、かつ、翻訳時に翻訳オプションNODLOADを指定して作成したものを使用してください。

1. 副プログラムは、目的プログラムまたはライブラリとして作成しておきます。[参照]“4.4.6 ライブラリを作成するときの手順”
2. WINLINK[リンク]ウィンドウを起動します。[参照]“4.5.1 WINLINKウィンドウの起動”
3. 種別で“EXE”を選択します。
4. ターゲット名を指定します。[参照]“4.5.2.1 ターゲット名の指定”
リンクする主プログラムのオブジェクトファイルを一覧のリストボックスに指定します。
→ “オブジェクトファイル名.EXE”がファイル(T)のエディットボックスに表示されます。
5. リンクに必要なその他のファイルを一覧のリストボックスに指定します。[参照]“4.5.2.2 ファイルの指定”
 - 1.で作成した副プログラム
 - オブジェクトファイル(副プログラムが目的プログラムの場合)
 - ライブラリ(副プログラムがライブラリの場合)
6. リンクオプションを設定します。[参照]“4.5.2.4 リンクオプションの設定”
7. リンクを開始します。[参照]“4.5.2.5 リンクの開始と終了”
8. リンクを終了します。[参照]“4.5.2.5 リンクの開始と終了”

4.4.4 動的リンク構造の実行可能プログラムを作成するときの手順

ここでは、副プログラムを呼び出すCOBOL目的プログラムをリンクして、動的リンク構造の実行可能プログラムを作成するときの手順を説明します。なお、COBOL目的プログラムは、データ名指定のCALL文を記述していないCOBOLソースプログラムを翻訳して作成したもの、かつ、翻訳時に翻訳オプションNODLOADを指定して作成したものを使用してください。

1. WINLINK[リンク]ウィンドウを起動します。[参照]“4.5.1 WINLINKウィンドウの起動”
以下の作業を行い、副プログラムをDLLとして作成します。
2. 種別で“DLL”を選択します。
3. ターゲット名を指定します。[参照]“4.5.2.1 ターゲット名の指定”
リンクする副プログラムのオブジェクトファイルの名前を一覧のリストボックスに指定します。
→ “オブジェクトファイル名.DLL”がファイル(T)のエディットボックスに表示されます。
4. 副プログラムのリンクに必要なその他のファイルを一覧のリストボックスに指定します。[参照]“4.5.2.2 ファイルの指定”
5. リンクオプションを設定します。[参照]“4.5.2.4 リンクオプションの設定”
6. 副プログラムのリンクを開始します。[参照]“4.5.2.5 リンクの開始と終了”
→ DLL、副プログラムのインポートライブラリ(DLL名.LIB)およびエクスポートファイルが作成されます。



注意

すでに作成するインポートライブラリ名と同じ名前のライブラリが存在する場合、注意をうながすメッセージが表示されます。

以下の作業を行い、実行可能プログラムを作成します。

7. 種別で“EXE”を選択します。
8. ターゲット名を指定します。[参照]“4.5.2.1 ターゲット名の指定”
リンクする主プログラムのオブジェクトファイルを一覧のリストボックスに指定します。
→ “オブジェクトファイル名.EXE”がファイル(T)のエディットボックスに表示されます。
9. 主プログラムのリンクに必要なその他のファイル(インポートライブラリ: (6)で作成されたもの)を一覧のリストボックスに指定します。[参照]“4.5.2.2 ファイルの指定”
10. リンクオプションを設定します。[参照]“4.5.2.4 リンクオプションの設定”
11. リンクを開始します。[参照]“4.5.2.5 リンクの開始と終了”
12. リンクを終了します。[参照]“4.5.2.5 リンクの開始と終了”

4.4.5 動的プログラム構造の実行可能プログラムを作成するときの手順

ここでは、副プログラムを呼び出すCOBOL目的プログラムをリンクして、動的プログラム構造の実行可能プログラムを作成するときの手順を説明します。なお、COBOL目的プログラムは、データ名指定のCALL文を記述したCOBOLソースプログラムを翻訳して作成したもの、または翻訳時に翻訳オプションDLOADを指定して作成したものを使用してください。

1. 副プログラムは、DLLとして作成しておきます。[参照]“4.4.4 動的リンク構造の実行可能プログラムを作成するときの手順”の2.～6.
2. WINLINK[リンク]ウィンドウを起動します。[参照]“4.5.1 WINLINKウィンドウの起動”
3. 種別で“EXE”を選択します。
4. ターゲット名を指定します。[参照]“4.5.2.1 ターゲット名の指定”
リンクする主プログラムのオブジェクトファイルを一覧のリストボックスに指定します。
→ “オブジェクトファイル名.EXE”がファイル(T)のエディットボックスに表示されます。
5. リンクに必要なその他のファイルを一覧のリストボックスに指定します。[参照]“4.5.2.2 ファイルの指定”
6. リンクオプションを設定します。[参照]“4.5.2.4 リンクオプションの設定”
7. リンクを開始します。[参照]“4.5.2.5 リンクの開始と終了”
8. リンクを終了します。[参照]“4.5.2.5 リンクの開始と終了”

4.4.6 ライブラリを作成するときの手順

ここでは、1つ以上の副プログラムの目的プログラムを結合して、1つのライブラリを作成するときの手順を説明します。

1. WINLINK[リンク]ウィンドウを起動します。[参照]“4.5.1 WINLINKウィンドウの起動”
2. WINLINK[ライブラリモジュール結合]ウィンドウに切り替えます。
[コマンド]メニューから“ライブラリモジュール結合”を選択します。
3. ライブラリ名を指定します。[参照]“4.5.3.1 ライブラリ名の指定”
結合するオブジェクトファイルを一覧のリストボックスに指定します。
→ “オブジェクトファイル名.LIB”がライブラリファイルのエディットボックスに表示されます。



注意

すでに存在するファイル名を指定した場合、そのファイルは削除され、新しいファイルが作成されます。

4. ライブラリ結合に必要なその他のファイル(ライブラリ)を一覧のリストボックスに指定します。[参照]“4.5.3.2 ファイルの指定”
5. ライブラリ結合を開始します。[参照]“4.5.3.3 ライブラリ結合の開始と終了”
6. ライブラリ結合を終了します。[参照]“4.5.3.3 ライブラリ結合の開始と終了”

4.5 WINLINKウィンドウ

WINLINKは、COBOLで翻訳したプログラムをリンクして実行可能プログラムやDLLを作成したり、ライブラリおよびインポートライブラリを作成するためのユーティリティです。WINLINKには、次の2種類のウィンドウがあります。

WINLINK[リンク]ウィンドウ

インポートライブラリ/DLL、実行可能プログラムの作成

WINLINK[ライブラリモジュール結合]ウィンドウ

ライブラリの作成

これらのウィンドウは、それぞれのウィンドウの[コマンド]メニューによって切り替えます。ここでは、これらの2つのWINLINKウィンドウについて説明します。ウィンドウの使い方の詳細については、ヘルプを参照してください。

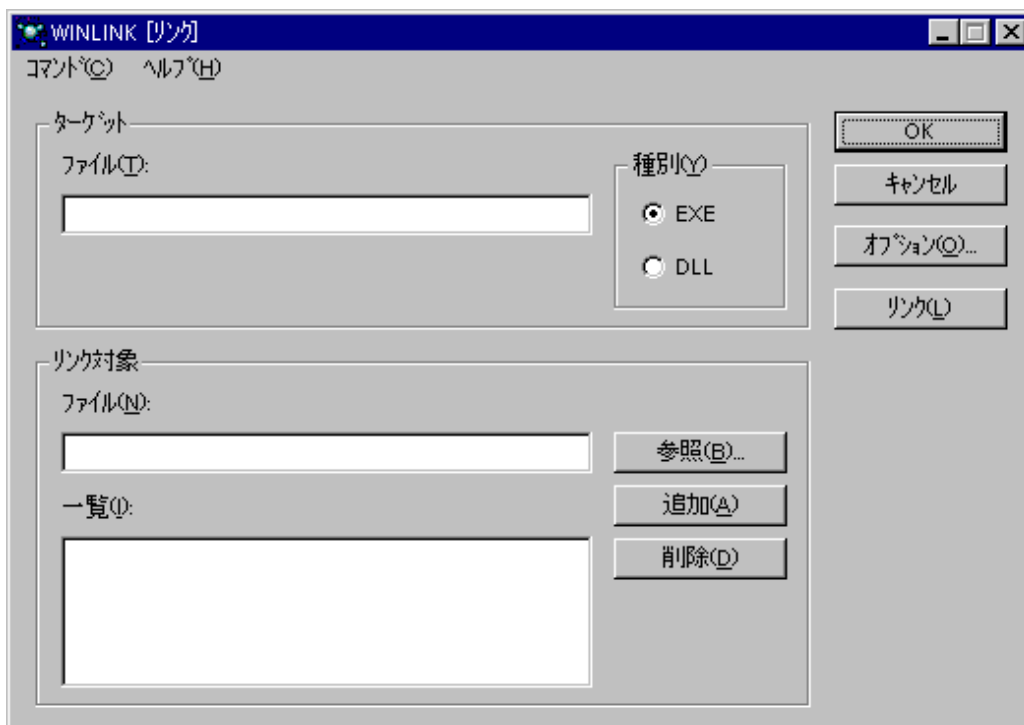
4.5.1 WINLINKウィンドウの起動

WINLINKウィンドウを起動する方法を以下に示します。

- プロジェクトマネージャの[ツール]メニューから“リンク”を選択します。

4.5.2 WINLINK[リンク]ウィンドウ

WINLINK[リンク]ウィンドウは、リンクによってインポートライブラリ/DLLおよび実行可能プログラムを作成するときに使用します。



メニューバー

コマンド

ウィンドウの切替えを行います。

ヘルプ

WINLINK[リンク]ウィンドウのヘルプを表示します。

エディットボックス

ファイル(T)

作成するファイル名を指定します。

ファイル(N)

リストボックスに追加するファイルを指定します。

リストボックス

一覧

リンクの対象となるファイルを表示します。

グループボックス(ラジオボタン)

種別

ターゲットの種類を選択します。

ボタン

OK

リンクを開始するときに選択します。リンクが開始された後、WINLINK[リンク]ウィンドウを閉じます。

キャンセル

WINLINK[リンク]ウィンドウを閉じます。

オプション

リンクオプションを指定するときに選択します。

リンク

リンクを開始するときに選択します。

参照

ファイルを参照するときに選択します。

追加

リストボックスにファイルを追加するときに選択します。

削除

リストボックスからファイルを削除するときに選択します。

4.5.2.1 ターゲット名の指定

以下のどちらかの方法で、リンクによって作成するファイルの名前をファイル(T)のエディットボックスに指定します。

方法1

1. ターゲットの種別を選択します。
2. ファイル(N)のエディットボックスにオブジェクトファイル名を指定します。
3. [追加]ボタンをクリックします。
 - 一覧のリストボックスにオブジェクトファイル名が追加されます。
 - “オブジェクトファイル名.EXE”または“オブジェクトファイル名.DLL”がファイル(T)のエディットボックスに表示されます。

方法2

ファイル(T)のエディットボックスに、作成するファイルの名前を直接キー入力してください。



注意

ファイル(T)のエディットボックスにファイル名を指定した後に種別のラジオボタンを選択すると、指定したファイル名の拡張子は、ラジオボタンで選択したファイルの拡張子に変更されます。

4.5.2.2 ファイルの指定

以下の方法で、リンクに必要なファイルの名前を一覧のリストボックスに指定します。

1. [参照]ボタンをクリックし、[ファイルの参照]ダイアログでリンクに必要なファイルを選択します。リンク時に必要なファイルを以下に示します。
 - ー オブジェクトファイル
 - ー ライブラリ
 - ー インポートライブラリ(動的リンク構造の実行可能ファイルを作成する場合)
2. [ファイルの参照]ダイアログの[開く]ボタンをクリックします。
 - 選択したファイル名が、ファイル(N)のエディットボックスに表示されます。なお、直接キー入力によって、ファイル(N)のエディットボックスにファイル名を指定することもできます。
3. [追加]ボタンをクリックします。
 - 一覧のリストボックスにファイル名が追加されます。
4. リンクに必要なすべてのファイルの指定が終わるまで、1.～3.の操作を繰り返し行ってください。

4.5.2.3 インポートライブラリの作成

DLLを作成すると、自動的にインポートライブラリおよびエクスポートファイルが作成されます。

4.5.2.4 リンクオプションの設定

リンクオプションの設定は、WINLINK[リンク]ウィンドウの[オプション]ボタンをクリックすると表示される[リンクオプション]ダイアログで行います。主なリンクオプション(デバッグは除く)は自動的に設定されます。[参照]“表4.5 LINKコマンドのオプション”

エディットボックス

リンクオプション

リンクオプションを指定します。

Cランタイムライブラリ

Cランタイムライブラリとして、LIBC.LIB以外のライブラリをリンクする場合に指定します。

ラジオボタン

DLLのエントリオブジェクト

COBOLで作成されたオブジェクトだけでDLLを作成するか、他言語で作成したオブジェクトもあわせてDLLを作成するかを指定します。

ボタン

OK

リンクオプションの設定が終了したときに選択します。

キャンセル

[リンクオプション]ダイアログを開く前の状態に戻し、[リンクオプション]ダイアログを閉じます。

ヘルプ

ヘルプウィンドウを開くときに選択します。

デバッグ

デバッグを行うときに選択します。エディットボックスにデバッグを行うためのリンクオプションが表示されます。

マップ

リンクオプション/MAPを指定するときに選択します。エディットボックスにリンクオプション/MAPが表示されます。

参照

リンクするCランタイムライブラリファイル名を選択します。

リセット

Cランタイムライブラリの指定を解除し、デフォルト(LIBC.LIB)に戻すときに選択します。

4.5.2.5 リンクの開始と終了

リンクに必要な情報の設定が終わったらリンクを開始します。

1. リンクを開始するには、[リンク]ボタンをクリックします。
→ ビルダウィンドウに切り替わり、リンク中のメッセージが表示されます。
2. リンクを終了します。
→ ビルダウィンドウにビルド終了のメッセージが表示されます。
3. リンクが正常に行われたことを確認してください。
→ リンクエラーになった場合、“[4.9 リンクメッセージ](#)”を参照し、対処してください。
4. そのままリンク操作を続ける場合には、ターゲット名の指定からの操作を繰り返して行ってください。
リンク操作を終了する場合には、WINLINK[リンク]ウィンドウの[キャンセル]ボタンをクリックし、WINLINK[リンク]ウィンドウを閉じてください。



注意

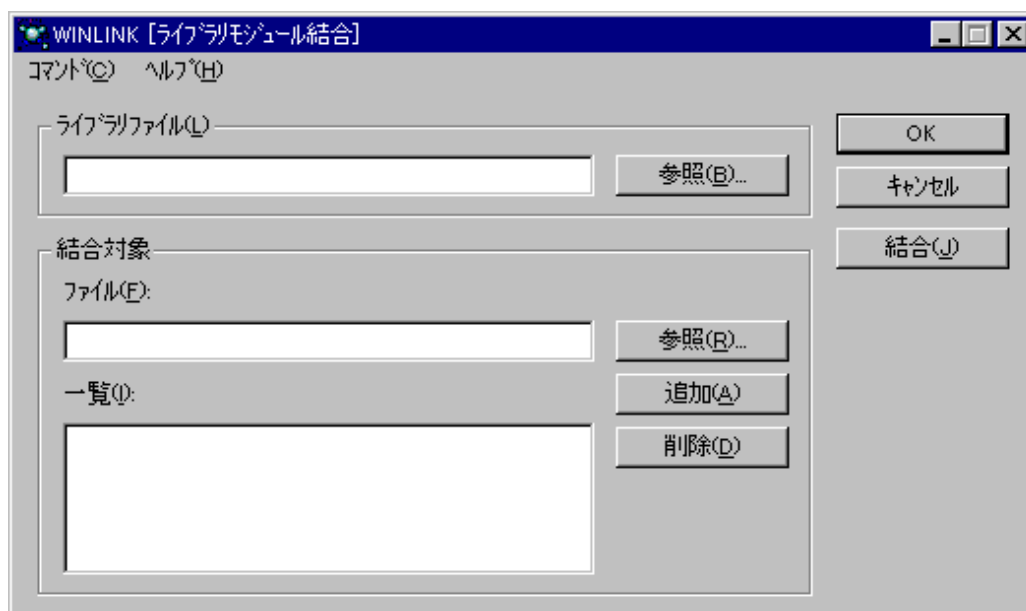
リンク時には、以下の点に注意してください。リンクエラーとなります。[参照]“[4.9 リンクメッセージ](#)”

- ・ 翻訳オプションNOMAINが有効となっている(翻訳時に翻訳オプションMAINを指定しないで翻訳した)目的プログラムだけで実行可能プログラムを作成しないでください。
- ・ 翻訳オプションMAINが有効となっている複数のプログラムから1つの実行可能プログラムを作成しないでください。
- ・ 実行可能ファイルを作成する場合、“実行可能プログラム名.LIB”という名前でインポートライブラリが作成されることがあるため、同一のパス上に同一名のDLLを作成しないでください。

- ・ 前バージョンレベルのコンパイラで作成したオブジェクトファイルは、リンクできない場合があります。必ず、COBOLソースファイルを再翻訳してからリンクしてください。

4.5.3 WINLINK[ライブラリモジュール結合]ウィンドウ

WINLINK[ライブラリモジュール結合]ウィンドウは、1つ以上の目的プログラムを結合して1つのライブラリにするときに使用します。



メニューバー

コマンド

ウィンドウの切替えを行います。

ヘルプ

WINLINK[ライブラリモジュール結合]ウィンドウのヘルプを表示します。

エディットボックス

ライブラリファイル

作成するライブラリ名を指定します。

ファイル

リストボックスに追加するファイルを指定します。

リストボックス

一覧

結合対象となるファイルを表示します。

ボタン

OK

結合を開始するときに選択します。結合が開始された後、WINLINK[ライブラリモジュール結合]ウィンドウを閉じます。

キャンセル

WINLINK[ライブラリモジュール結合]ウィンドウを閉じます。

結合

ライブラリの作成を開始するときに選択します。

参照(B)

ライブラリファイルを参照するときに選択します。

参照(R)

ファイルを参照するときに選択します。

追加

リストボックスにファイルを追加するときに選択します。

削除

リストボックスからファイルを削除するときに選択します。

4.5.3.1 ライブラリ名の指定

以下のどちらかの方法で、作成するライブラリの名前をライブラリファイルのエディットボックスに指定します。

方法1

1. ファイルのエディットボックスにオブジェクトファイル名を指定します。
2. [追加]ボタンをクリックします。
 - 一覧のリストボックスにオブジェクトファイル名が追加されます。
 - “オブジェクトファイル名.LIB”がライブラリファイルのエディットボックスに表示されます。

方法2

ライブラリファイルのエディットボックスに、作成するファイルの名前を直接キー入力してください。



注意

すでに存在するライブラリと同じ名前を指定した場合、そのライブラリは削除され、新しいライブラリが作成されます。

4.5.3.2 ファイルの指定

以下の方法で、結合するオブジェクトファイルの名前を一覧のリストボックスに指定します。

1. [参照(R)]ボタンをクリックし、[ファイルの参照]ダイアログで結合に必要なファイルを選択します。
2. [ファイルの参照]ダイアログの[開く]ボタンをクリックします。
 - 選択したファイル名がファイルのエディットボックスに表示されます。

なお、直接キー入力によって、ファイルのエディットボックスにファイル名を指定することもできます。
3. [追加]ボタンをクリックします。
 - 一覧のリストボックスにファイル名が追加されます。
4. 結合するオブジェクトファイル名の指定が終わるまで、1.～3.の操作を繰り返し行ってください。



注意

翻訳オプションMAINが有効となっているプログラムは指定しないでください。

4.5.3.3 ライブラリ結合の開始と終了

結合するオブジェクトファイル名の指定がすべて終わったら、ライブラリの結合を開始します。

1. ライブラリの結合を開始するには、[結合]ボタンをクリックします。
 - ビルダウインドウに切り替わり、ライブラリ結合中のメッセージが表示されます。
2. ライブラリ結合を終了します。
 - ビルダウインドウにビルド終了のメッセージが表示されます。

3. ライブラリ結合が正常に行われたことを確認してください。
4. そのままライブラリ結合を続ける場合には、ライブラリ名の指定からの操作を繰り返し行ってください。
ライブラリ結合操作を終了する場合には、WINLINK[ライブラリモジュール結合]ウィンドウの[キャンセル]ボタンをクリックし、WINLINK[ライブラリモジュール結合]ウィンドウを閉じてください。

4.6 モジュール定義ファイル

モジュール定義ファイルについて、以下に示します。

4.6.1 モジュール定義ファイルの内容

モジュール定義ファイルは、以下の条件でインポートライブラリを作成するときに必要となります。

- ・ STDCALL呼出し規約を指定した動的プログラム構造のプログラムを作成する場合

モジュール定義ファイルの作成は、通常のテキストエディタを使って行います。なお、WINLINK[リンク]ウィンドウにモジュール定義ファイルを自動的に生成する機能があります。

モジュール定義ファイル内のモジュール定義文では、DLLの内容およびシステムへの要求事項を定義します。

“表4.4 モジュール定義文”に、モジュール定義文および指定する内容を示します。

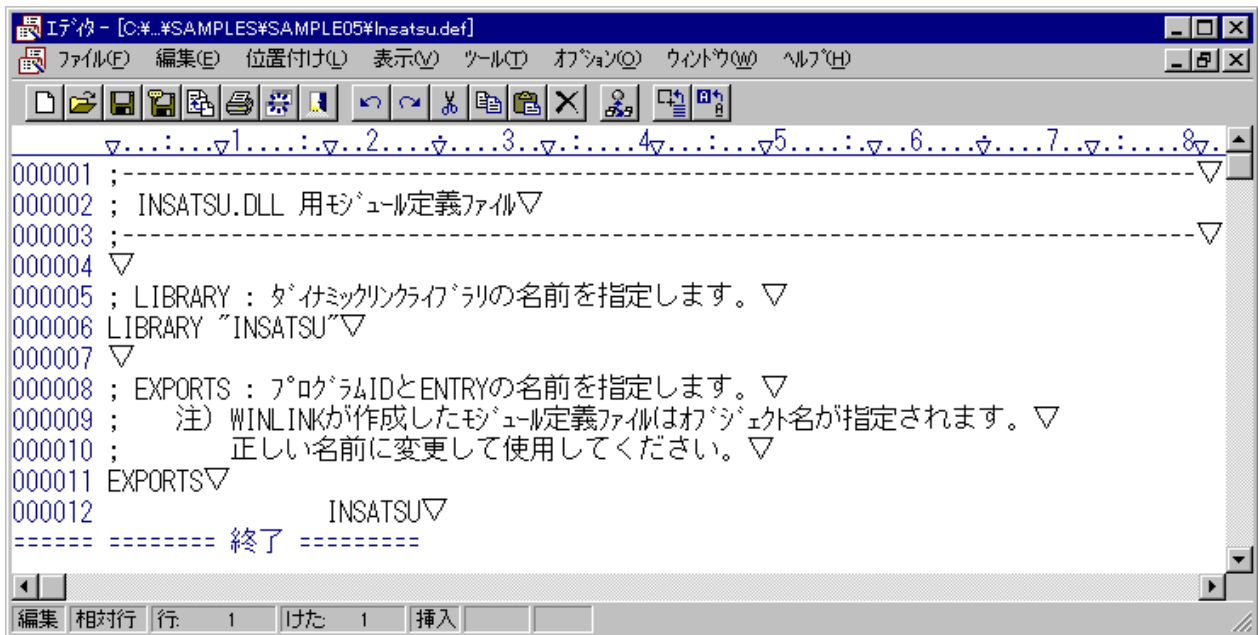
表4.4 モジュール定義文

モジュール定義文	指定する内容
NAME	アプリケーションのモジュール名を定義します。
LIBRARY	ダイナミックリンクライブラリの名前を指定します。
DESCRIPTION	モジュールに対する簡単な説明を記述します。
STACKSIZE	スタックのサイズをバイト単位で指定します。
SECTIONS	特定のセクションの属性を設定します。
EXPORTS	プログラムIDとENTRYの名前を指定します。
VERSION	バージョン番号を埋め込みます。

モジュール定義ファイルには、以下に示す情報を必ず記述してください。

- ・ LIBRARY
- ・ EXPORTS(自動生成した場合、EXPORTSには指定されたオブジェクト名が入ります)

以下にWINLINK[リンク]ウィンドウで作成されたモジュール定義ファイルの内容を示します。



4.6.2 リンクの手順

モジュール定義ファイルを使用してインポートライブラリを作成する場合は、WINLINK[リンク]ウィンドウの一覧のリストボックスにモジュール定義ファイルを指定します。

4.6.3 モジュール定義ファイルの自動生成

モジュール定義ファイルの自動生成を行う場合、[コマンド]メニューから“モジュール定義ファイル生成”を選択します。

4.7 リンクコマンドを使ったリンク操作

リンクコマンドを使って目的プログラムのリンクを行うこともできます。

4.7.1 LINKコマンド

LINKコマンドは、リンク操作をコマンドプロンプト上で行うためのコマンドです。

コマンドの形式の詳細については、“[4.8.1 LINKコマンドの形式](#)”を参照してください。

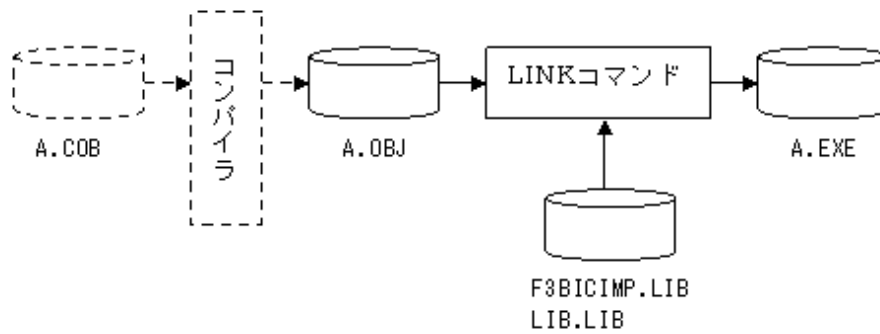
LINKコマンドを使う場合には、以下に示すライブラリを指定する必要があります。

- F3BICIMP.LIB
- LIBC.LIB

4.7.2 リンクコマンドの使用例

ここでは、LINKコマンドを使った例をいくつか示します。

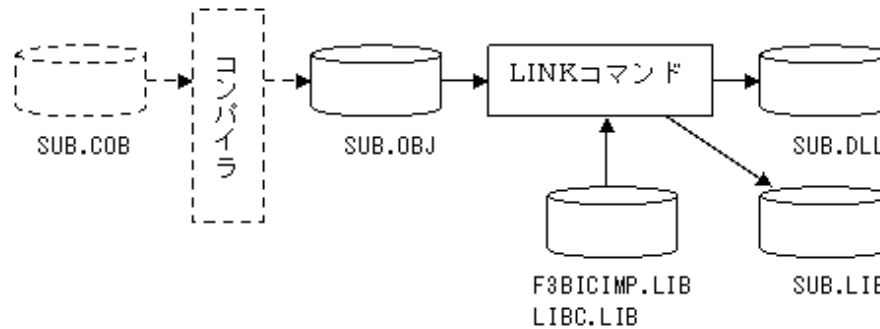
1つの目的プログラムをリンクする場合



```
LINK A.OBJ F3BICIMP.LIB LIBC.LIB /OUT:A.EXE
```

/OUT: メイン出力ファイルの名前を指定

DLLを作成する場合

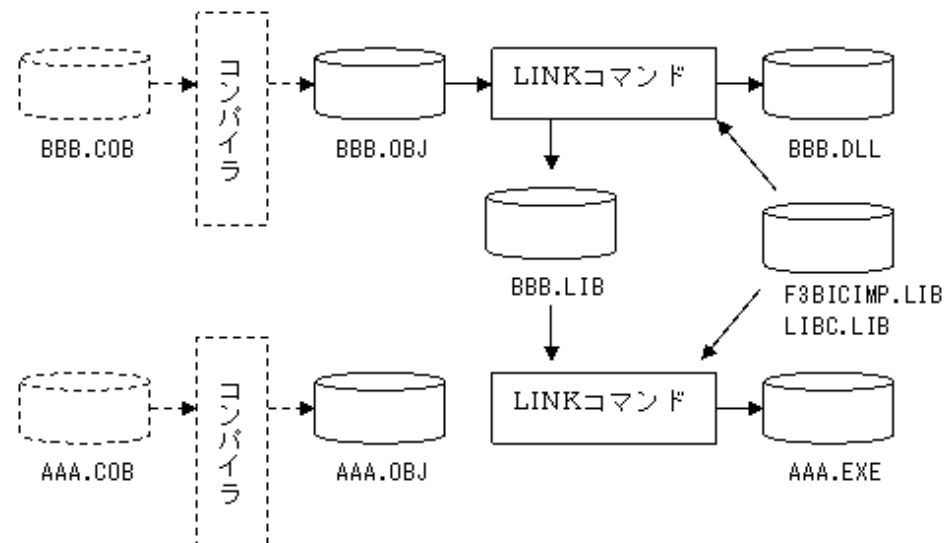


```
LINK SUB.OBJ F3BICIMP.LIB LIBC.LIB /DLL /OUT:SUB.DLL
```

/DLL: ダイナミックリンクライブラリ (DLL) を作成するときに指定

/OUT: メイン出力ファイルの名前を指定

動的リンク構造の実行可能プログラムを作成する場合



- DLL およびインポートライブラリの作成

```
LINK BBB.OBJ F3BICIMP.LIB LIBC.LIB /DLL /OUT:BBB.DLL
```

/DLL : ダイナミックリンクライブラリ (DLL) を作成するときに指定
/OUT : メイン出力ファイルの名前を指定

- 実行可能プログラムの作成

```
LINK AAA.OBJ F3BICIMP.LIB LIBC.LIB BBB.LIB /OUT:AAA.EXE
```

/OUT : メイン出力ファイルの名前を指定

4.8 リンクコマンドの形式

リンクコマンドは、プログラムをリンクし、実行可能プログラムを生成します。

リンクコマンド実行時に必要となるファイルおよび生成されるファイルの詳細については、“[4.2 リンクに必要な資源](#)”を参照してください。リンクコマンドの指定場所および使い方については、“[4.7 リンクコマンドを使ったリンク操作](#)”を参照してください。

4.8.1 LINKコマンドの形式

ここでは、LINKコマンドの形式について説明します。

入力形式

コマンド	オペランド
LINK	ファイル名の並び [オプションの並び]

オペランドの説明

コマンド名および各オペランドの間には、1つ以上の空白が必要です。[]で囲んである部分は、省略できることを示します。なお、以降の説明で、ファイル名には、絶対パス名または相対パス名を指定することができます。また、空白を含むファイル名を指定する場合は、ファイル名を二重引用符(”)で囲む必要があります。

ファイル名の並び

以下に指定形式を示します。

```
オブジェクトファイル名 ライブラリ名
```

オブジェクトファイル名

リンクするオブジェクトファイルの名前を1つ以上指定します。目的プログラム名を複数個指定する場合、空白またはタブで区切ります。

ライブラリ名

標準(オブジェクトコード)ライブラリ、インポートライブラリまたはエクスポートファイルを指定します。標準オブジェクトライブラリを指定した場合、外部参照を解決するために必要なオブジェクトモジュールだけがリンクされます。

以下のファイルは、必ず指定してください。

- COBOLのインポートライブラリ
- LIBC.LIB

オプションの並び

LINKコマンドのオプションを指定します。指定する内容については、下表を参照してください。

表4.5 LINKコマンドのオプション

内容	指定形式
COBOL 対話型デバッガおよび診断機能を使用するための準備をする指定をします。	/DEBUG /DEBUGTYPE:COFF他 (注1)
ダイナミックリンクライブラリ (DLL) を作成するときに指定します。	/DLL
モジュール定義ファイルを指定します。	/DEF:モジュール定義ファイル名
外部参照情報を生成します。	/EXPORT:外部参照名
メイン出力ファイルの名前を指定します。	/OUT:filename
スタックサイズの変更を指定します。	/STACK:スタックサイズ (注2)
マップファイルを生成する場合に指定します。	/MAP:filename

注1 : COBOL対話型デバッガおよび診断機能に関するリンクオプションの詳細は、“[19.6.2.3 プログラムの翻訳・リンク](#)”および“[20.2.3 プログラムのリンク](#)”を参照してください。

注2 : STACKオプションの指定を省略した場合、スタックサイズは4Mバイトになります。スタックサイズは、バイト単位で指定してください。[参照]“[5.9 注意事項](#)”

4.8.2 インポートライブラリの結合

複数のインポートライブラリを、LIBコマンドを使用して1つのインポートライブラリに結合することができます。

以下の場合、複数のインポートライブラリを1つのインポートライブラリに結合すると便利です。

例1 複数のDLLを動的リンク構造で呼び出す場合

```

プログラムA
IDENTIFICATION DIVISION.
PROGRAM-ID. A

:
PROCEDURE DIVISION.
CALL "SUB1".

CALL "SUB2".

CALL "SUB3".

```

SUB1.DLLを作成する

```
LINK SUB1.OBJ F3BICIMP.LIB LIBC.LIB /DLL /OUT:SUB1.DLL
```

SUB1.OBJ : プログラムSUB1のオブジェクトファイル
F3BICIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
LIBC.LIB : C ランタイムライブラリ

LINKすると、インポートライブラリSUB1.LIBが作成されます。

SUB2.DLLを作成する

```
LINK SUB2.OBJ F3BICIMP.LIB LIBC.LIB /DLL /OUT:SUB2.DLL
```

SUB2.OBJ : プログラムSUB2のオブジェクトファイル
F3B1CIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
LIBC.LIB : C ランタイムライブラリ

LINKすると、インポートライブラリSUB2.LIBが作成されます。

SUB3.DLLを作成する

```
LINK SUB3.OBJ F3B1CIMP.LIB LIBC.LIB /DLL /OUT:SUB3.DLL
```

SUB3.OBJ : プログラムSUB3のオブジェクトファイル
F3B1CIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
LIBC.LIB : C ランタイムライブラリ

LINKすると、インポートライブラリSUB3.LIBが作成されます。

SUB1.DLLからSUB3.DLLのインポートライブラリを結合する

```
LIB SUB1.LIB SUB2.LIB SUB3.LIB /OUT:SUB123.LIB
```

SUB1.LIB : SUB1.DLLのインポートライブラリ
SUB2.LIB : SUB2.DLLのインポートライブラリ
SUB3.LIB : SUB3.DLLのインポートライブラリ

コマンドを実行すると、SUB1.DLLからSUB3.DLLのインポートライブラリが結合されたインポートライブラリSUB123.LIBが作成されます。

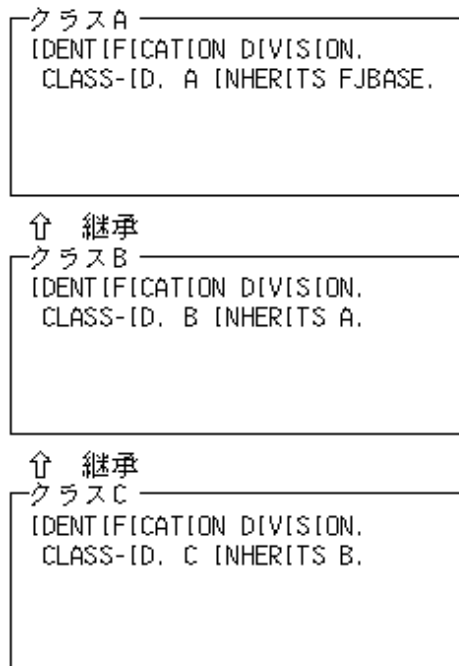
作成されたインポートライブラリを使用して、プログラムAをLINKします。

プログラムAをLINKする

```
LINK A.OBJ F3B1CIMP.LIB LIBC.LIB SUB123.LIB /OUT:A.EXE
```

A.OBJ : プログラムA のオブジェクトファイル
F3B1CIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
LIBC.LIB : C ランタイムライブラリ
SUB123.LIB : SUB1.DLLからSUB3.DLLのインポートライブラリを1つに結合したもの

例2 クラスの継承関係が深い場合



クラスAのDLLを作成する

```
LINK A.OBJ F3BICIMP.LIB LIBC.LIB /DLL /OUT:A.DLL
```

A.OBJ : クラスA のオブジェクトファイル
F3BICIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
LIBC.LIB : C ランタイムライブラリ

LINKすると、インポートライブラリA.LIBが作成されます。

クラスBのDLLを作成する

```
LINK B.OBJ F3BICIMP.LIB LIBC.LIB A.LIB /DLL /OUT:B.DLL
```

B.OBJ : クラスB のオブジェクトファイル
F3BICIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
LIBC.LIB : C ランタイムライブラリ
A.LIB : クラスA のインポートライブラリ

LINKすると、インポートライブラリB.LIBが作成されます

クラスCのDLLを作成する場合、クラスAとクラスBのインポートライブラリが必要となるので、クラスAとクラスBのインポートライブラリを結合します。

```
LIB A.LIB B.LIB /OUT:AB.LIB
```

A.LIB : クラスA のインポートライブラリ
B.LIB : クラスB のインポートライブラリ

コマンドを実行すると、クラスAとクラスBのインポートライブラリが結合されたインポートライブラリAB.LIBが作成されます。

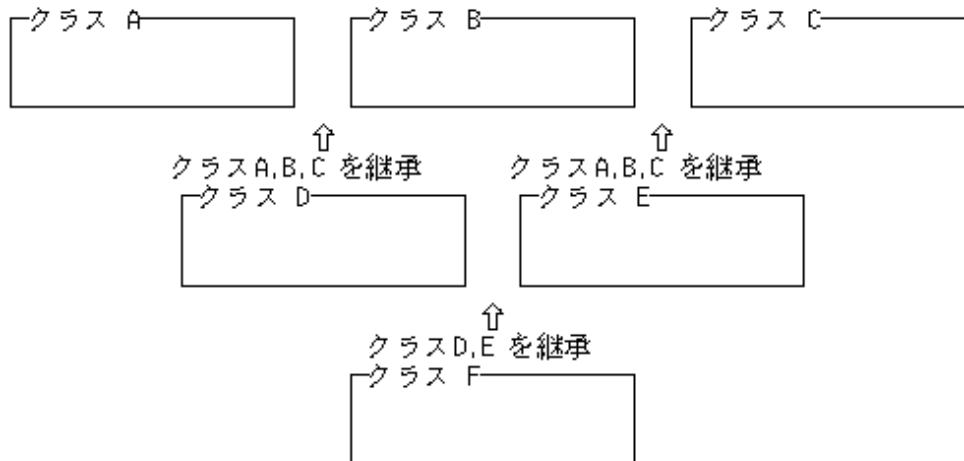
クラスCのDLLを作成する

```
LINK C.OBJ F3BICIMP.LIB LIBC.LIB AB.LIB /DLL /OUT:C.DLL
```


C.OBJ : クラスC のオブジェクトファイル
 F3BICIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
 LIBC.LIB : C ランタイムライブラリ
 AB.LIB : クラスA とクラスB のインポートライブラリを1つに結合したもの

LINKすると、インポートライブラリC.LIBが作成されます

例3 クラスを多重継承している場合



クラスAのDLLを作成する

```
LINK A.OBJ F3BICIMP.LIB LIBC.LIB /DLL /OUT:A.DLL
```

A.OBJ : クラスA のオブジェクトファイル
 F3BICIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
 LIBC.LIB : C ランタイムライブラリ

LINKすると、インポートライブラリA.LIBが作成されます

クラスBのDLLを作成する

```
LINK B.OBJ F3BICIMP.LIB LIBC.LIB /DLL /OUT:B.DLL
```

B.OBJ : クラスB のオブジェクトファイル
 F3BICIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
 LIBC.LIB : C ランタイムライブラリ

LINKすると、インポートライブラリB.LIBが作成されます

クラスCのDLLを作成する

```
LINK C.OBJ F3BICIMP.LIB LIBC.LIB /DLL /OUT:C.DLL
```

C.OBJ : クラスC のオブジェクトファイル
 F3BICIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
 LIBC.LIB : C ランタイムライブラリ

LINKすると、インポートライブラリC.LIBが作成されます

クラスDのDLLを作成する

クラスDのDLLを作成する場合、クラスA、クラスBおよびクラスCのインポートライブラリが必要となるので、クラスA、クラスBおよびクラスCのインポートライブラリを結合します。

```
LIB A.LIB B.LIB C.LIB /OUT:ABC.LIB
```

コマンドを実行すると、クラスA、クラスBおよびクラスCのインポートライブラリが結合されたインポートライブラリABC.LIBが作成されます。

```
LINK D.OBJ F3BICIMP.LIB LIBC.LIB ABC.LIB /DLL /OUT:D.DLL
```

D.OBJ : クラスD のオブジェクトファイル
F3BICIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
LIBC.LIB : C ランタイムライブラリ
ABC.LIB : クラスA、クラスB、クラスC のインポートライブラリを1つに結合したもの

LINKすると、インポートライブラリD.LIBが作成されます

クラスEのDLLを作成する

クラスEのDLLを作成する場合、クラスA、クラスBおよびクラスCのインポートライブラリが必要となるので、クラスD作成時に作成したインポートライブラリを指定します。

```
LINK E.OBJ F3BICIMP.LIB LIBC.LIB ABC.LIB /DLL /OUT:E.DLL
```

E.OBJ : クラスEのオブジェクトファイル
F3BICIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
LIBC.LIB : C ランタイムライブラリ
ABC.LIB : クラスA、クラスB、クラスC のインポートライブラリを1つに結合したもの

LINKすると、インポートライブラリE.LIBが作成されます

クラスFのDLLを作成する

クラスFのDLLを作成する場合、クラスA、クラスB、クラスC、クラスDおよびクラスEのインポートライブラリが必要となるので、クラスA、クラスB、クラスC、クラスDおよびクラスEのインポートライブラリを結合します。

```
LIB A.LIB B.LIB C.LIB D.LIB E.LIB /OUT:ABCDE.LIB
```

コマンドを実行すると、クラスA、クラスB、クラスC、クラスDおよびクラスEのインポートライブラリが結合されたインポートライブラリABCDE.LIBが作成されます。

```
LINK F.OBJ F3BICIMP.LIB LIBC.LIB ABCDE.LIB /DLL /OUT:F.DLL
```

F.OBJ : クラスF のオブジェクトファイル
F3BICIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
LIBC.LIB : C ランタイムライブラリ
ABCDE.LIB : クラスA、クラスB、クラスC、クラスD、クラスE のインポートライブラリを1つに結合したもの



注意

インポートライブラリ結合時、メッセージ番号LNK4006のメッセージが出力される場合がありますが、作成されたインポートライブラリは問題ありません。

4.9 リンクメッセージ

ここでは、NetCOBOL使用時に主に出力されるリンクメッセージについて説明しています。

LNK1104

原因

ディスクまたはルートフォルダに空き領域がありません。

対処

ファイルを消去してスペースを空けてください。

LNK1123

原因

32ビット用のリンカで16ビットのオブジェクトをリンクしようとした。

対処

32ビット用のオブジェクトを指定してください。

LNK1561

原因

主プログラムを翻訳するときに、翻訳オプションMAINを指定しないで翻訳し、実行可能プログラム(EXE)を作成しようとした。

対処

主プログラムを翻訳するときには、翻訳オプションMAINを指定して翻訳してください。

LNK2001

原因

- ・ ライブラリまたはオブジェクトファイルの中に、外部参照名が存在しません。
- ・ モジュール定義ファイルにあるEXPORTSの定義の中に、外部参照名が存在しません。

対処

モジュール定義ファイルの中のEXPORTS文に指定した外部参照名が、ライブラリまたはオブジェクトファイル中に存在しているか確認してください。

プログラムの呼び出す側と呼び出される側で外部参照名が一致しているかどうかを確認するには、DUMPBINコマンドを使用します。DUMPBINコマンドの使用方法については、“[1.1 DUMPBINコマンド](#)”を参照してください。

なお、ライブラリ、オブジェクトファイル中の外部参照名は、プログラム構造および呼出し規約によって異なります。具体的なプログラム構造については、“[10.1 呼出し関係の概要](#)”を参照してください。

LNK2005

原因

翻訳オプションMAINが有効となっている複数のプログラムから1つの実行可能ファイルを作成しようとした。

対処

主プログラム以外に翻訳オプションMAINは指定しないでください。

LNK4006

原因

- ・ ライブラリまたはオブジェクトファイル中にすでに定義されているデータの内部名があります。
- ・ モジュール定義ファイルにあるEXPORTSの定義が重複しています。

対処

モジュール定義ファイルの中のEXPORTS文に指定したexportnameがライブラリまたはオブジェクトファイル中に重複していないか確認してください。

第5章 プログラムを実行する

本章では、プログラムを実行するときの手順、実行環境情報の設定方法および実行操作で使用するウィンドウの使い方について説明します。

5.1 サンプルプログラムの実行

リンクが終わったら、作成した実行可能ファイルを実行してみましょう。

ここでは、WINEXECを使って、例題1のプログラムを実行します。なお、以下の説明では、実行可能ファイルをC:\COBOL\SAMPLES\SAMPLE01\SAMPLE1.EXEとします。

例題1では、特に設定が必要な実行環境情報はありますが、実行環境情報を設定する場合は、プログラムの実行前に以下の操作を行います。

1. プロジェクトマネージャウィンドウの[ツール]メニューから“実行環境設定ツール”を選択します。
→ [実行環境設定ツール]ダイアログが表示されます。

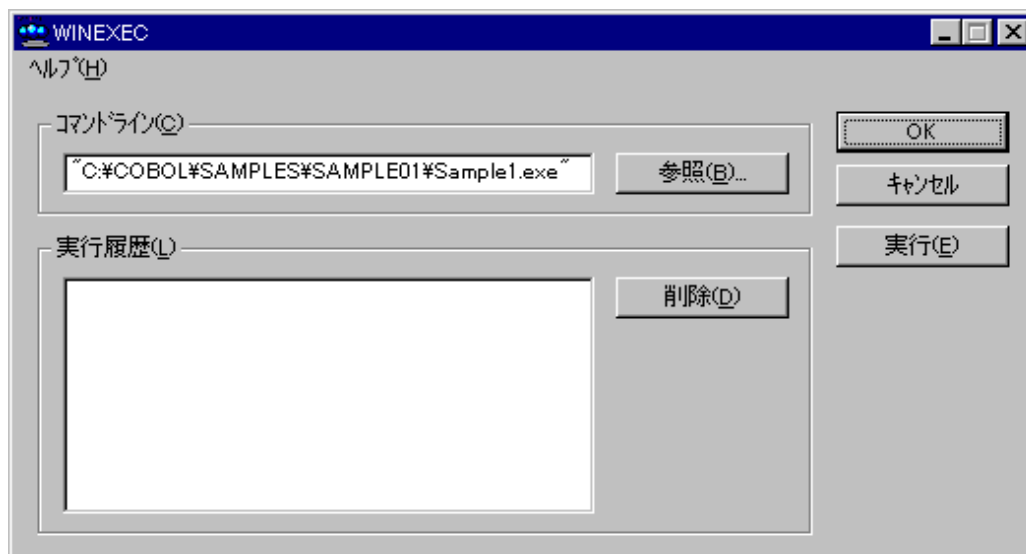


2. [ファイル]メニューから“開く”を選択します。
→ 実行用の初期化ファイルの指定ダイアログが表示されます。
3. EXEファイルのあるフォルダ(C:\COBOL\SAMPLES\SAMPLE01)に実行用の初期化ファイル(COBOL85.CBR)を作成します。
4. 必要な実行環境情報を“設定”し、“適用”します。
→ 実行用の初期化ファイル(C:\COBOL\SAMPLES\SAMPLE01\COBOL85.CBR)の内容が変更されます。
5. [ファイル]メニューから“閉じる”を選択します。
6. [ファイル]メニューから“終了”を選択します。
→ 実行環境設定ツールを終了します。

では、実際に実行可能ファイルを実行してみましょう。

1. プロジェクトマネージャウィンドウの[ツール]メニューから“実行”を選択します。
→ WINEXECウィンドウが表示されます。

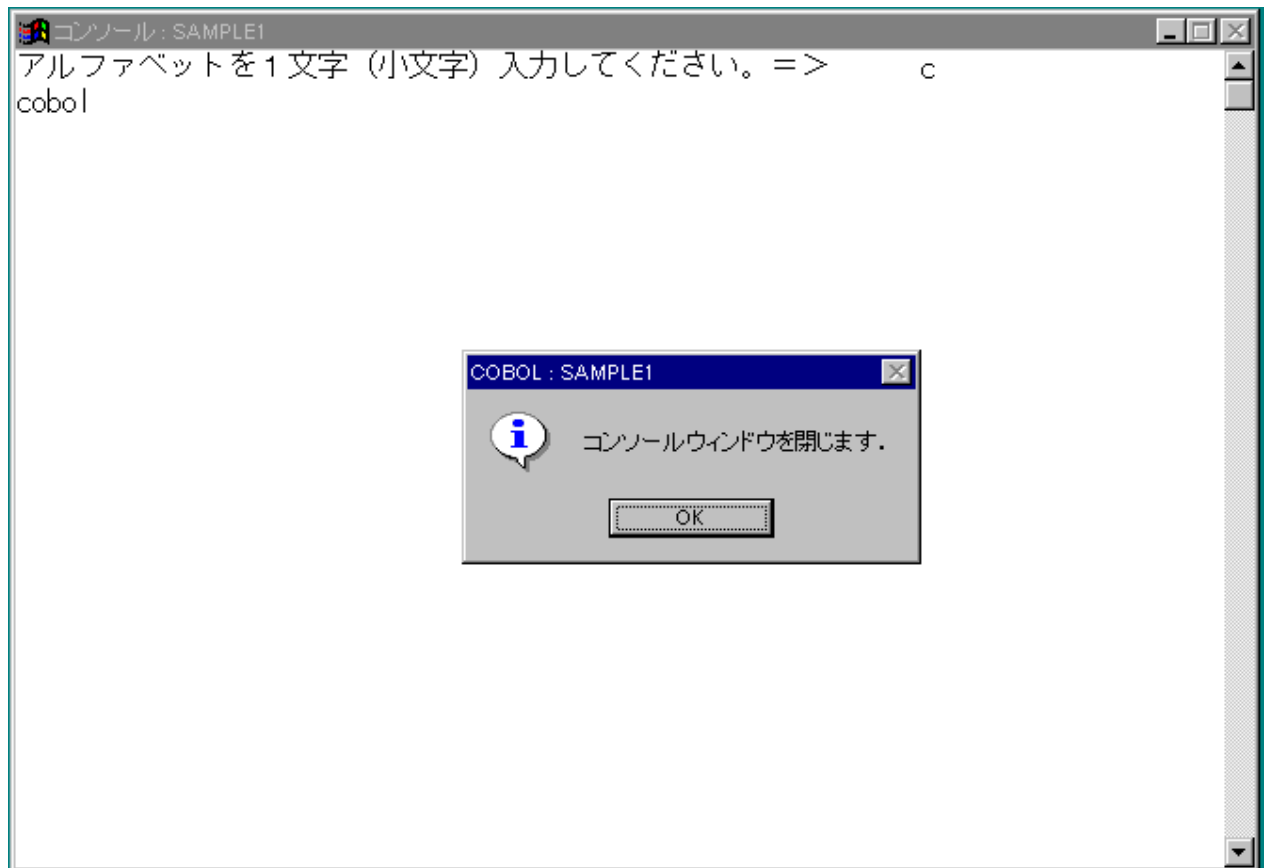
2. 実行するファイルの名前(SAMPLE1.EXE)をコマンドラインのエディットボックスに指定します。



3. [実行]ボタンをクリックします。
4. 例題1では、COBOLの小入出力機能を使ったデータの入出力が行われます。
以下のウィンドウが表示されるので、適当な英小文字を入力し、リターンキーを押してください。



→ 入力した英小文字を頭文字とする英単語が表示されます。



5. 実行結果を確認したら、メッセージボックスの[OK]ボタンをクリックします。
→ メッセージボックスおよびコンソールウィンドウが閉じます。

以上でサンプルプログラムの実行は終わりです。

5.2 実行の手順

COBOLソースプログラムを翻訳・リンクして作成した実行可能プログラム(以降、COBOLプログラムと略します)は、通常の32ビットWindows上で動作するアプリケーション(以降、アプリケーションと略します)と同様の方法で実行することができます。NetCOBOLでは、これらのプログラムを実行するのに便利な、WINEXECコマンドを提供しています。ここでは、COBOLプログラムを実行するときに必要な操作およびプログラムを実行するときの手順について説明します。

5.2.1 実行環境情報の設定

COBOLプログラムを実行するには、実行環境情報の設定を行う必要があります。

NetCOBOLでは、COBOLプログラムを実行するために割り当てる資源や情報のことを実行環境情報といいます。実行環境情報の設定方法については、“[5.3 実行環境情報の設定](#)”を参照してください。

5.2.2 COBOLプログラムの実行

COBOLプログラムを実行するには、WINEXECウィンドウを使用する方法およびその他の方法があります。

5.2.2.1 WINEXECウィンドウを使用する方法

WINEXECウィンドウは、COBOLプログラムの実行を行うためのウィンドウです。ここでは、WINEXECウィンドウを使用してCOBOLプログラムを実行する方法について説明します。ウィンドウの使い方の詳細については、ヘルプを参照してください。

1. WINEXECウィンドウを起動します。



参照

“5.5.1 WINEXECウィンドウの起動”

2. 実行するファイル名をコマンドラインのエディットボックスに指定します。



参照

“5.5.2 ファイル名の指定”

3. 引数がある場合、コマンドラインのエディットボックスに、コマンドラインの形式と同様に、直接キー入力します。
4. 実行を開始します。
WINEXECウィンドウの[実行]ボタンをクリックします。



参照

“5.5.3 実行の開始と終了”

5. すでに実行したプログラムを再度実行する場合には、実行履歴のリストボックスに表示されているファイル名をダブルクリックしてください。
6. 実行操作を終了します。
実行操作を終了する場合は、WINEXECウィンドウの[キャンセル]ボタンをクリックし、WINEXECウィンドウを閉じてください。



参照

“5.5.3 実行の開始と終了”

5.2.2.2 その他の方法

WINEXECウィンドウを使用する方法以外にも、各ウィンドウシステムで使用されているアプリケーションの実行方法を使用して、COBOLプログラムを実行することができます。

例としては、“コマンドプロンプトから実行する”、“エクスプローラでファイルを選択して実行する”、“バッチファイル中でファイルを指定して実行する”などがあります。これらの実行方法は各ウィンドウシステムに依存するため、詳細については各ウィンドウシステムのマニュアルまたはオンラインヘルプを参照してください。

また、COBOLプログラムをバッチファイルなどで実行する場合、以下の環境変数情報を実行用の初期化ファイルまたは環境変数に設定しておくと、オペレータの入力待ちなしにCOBOLプログラムを実行することができます。

環境変数情報	意味
@MessOutFile=ファイル名	COBOL の実行時メッセージを指定されたファイルに出力する。
@WinCloseMsg=OFF	ウィンドウを閉じるときのメッセージを表示しない。

各環境変数情報の詳細は、“5.4.1 環境変数情報”を参照してください。

バッチファイルを使用したCOBOLプログラムの実行方法

バッチファイルからプログラムを起動した場合に、起動したプログラムの終了同期をとるためには、OSのSTARTコマンドに/WAITオプションを指定して、直前のCOBOLプログラムの終了を待つ必要があります。



注意

- ・ CALLコマンドでプログラムを起動した場合は、起動したプログラムの終了同期がとれません。
- ・ ネットワーク接続されている環境のファイル(ローカル環境のファイルをUNC指定で使用する場合を含む)をアクセスしている場合、書込みに遅延が発生する場合があります。そのため、終了同期をとる考慮が必要となることがあります。

また、バッチファイルのERRORLEVELでCOBOLプログラムの終了状態を調べる場合も、STARTコマンドに/WAITオプションを指定します。

ERRORLEVELおよびSTARTコマンドは、OSによって仕様が異なる場合があります。詳細は、各OSのヘルプを参照してください。

以下に、COBOLプログラム“COBPROG1”と“COBPROG2”をバッチファイル“COBGO.BAT”で実行する方法の例を示します。

実行用の初期化ファイルを使用する場合

実行用の初期化ファイルを使用する場合は、“COBOL85.CBR”に設定します。

COBOL85.CBR

```
[COBPROG1]
@MessOutFile=COBPROG1.MES
@WinCloseMsg=OFF
:
[COBPROG2]
@MessOutFile=COBPROG2.MES
@WinCloseMsg=OFF
:
```

COBGO.BAT

```
REM COBOL プログラムの実行 1
start /wait COBPROG1
if errorlevel 12 echo エラーが発生しました ← errorlevel が12以上であれば真
REM COBOL プログラムの実行 2
start /wait COBPROG2
if errorlevel 12 echo エラーが発生しました ← errorlevel が12以上であれば真
```

環境変数に環境変数情報を指定する場合

環境変数に指定する場合には、コントロールパネルのシステムまたはSETコマンドで設定する方法があります。ここでは、バッチファイルでSETコマンドを使用して設定する方法について記述します。

COBGO.BAT

```
REM 環境変数情報の設定 1
SET @MessOutFile=COBPROG1.MES
SET @WinCloseMsg=OFF
SET @ExitSessionMSG=OFF
REM COBOL プログラムの実行 1
start /wait COBPROG1
if errorlevel 12 echo エラーが発生しました ← errorlevel が12以上であれば真
REM 環境変数情報の設定 2
SET @MessOutFile=COBPROG2.MES
REM COBOL プログラムの実行 2
start /wait COBPROG2
if errorlevel 12 echo エラーが発生しました ← errorlevel が12以上であれば真
```

バッチファイルを使用した場合の復帰コードについては、“メッセージ説明書”の“第3章 実行時メッセージ”を参照してください。

5.2.2.3 サービス配下の注意事項(実行時)

COBOLプログラムをサービス配下で実行する場合には、以下の点に注意してください。なお、サービス配下での印刷機能の使用時の注意点は、“[8.1.11 サービス配下の注意事項\(印刷時\)](#)”を参照してください。また、サービスの設定については、使用している各サービスの説明書を参照してください。

COBOLプログラムを呼び出すサービスが「デスクトップとの対話を許可」している場合

COBOLプログラムの実行環境情報には、特別な指定はありません。

COBOLプログラムを呼び出すサービスが「デスクトップとの対話を許可」していない場合

COBOLプログラムからデスクトップ上に各ウィンドウおよびメッセージボックスを出力できないため、COBOLプログラムの実行時にオペレータからの入力待ちを行う可能性がある場合、動作は保証されません。

このような場合は、以下のどちらかの方法で対処してください。

- ー サービスの設定を「デスクトップとの対話を許可」するように変更してください。
- ー ウィンドウの表示を前提としている以下の機能をCOBOLプログラムから使用しないようにして、かつ、オペレータからの入力待ちを行わないように実行環境情報を設定してください。
 - コンソールウィンドウを使用した小入出力機能(ACCEPT/DISPLAY)
 - スクリーン操作機能
 - 表示ファイルの画面機能

オペレータからの入力待ちを回避するための実行環境情報の設定については、“[5.2.2.2 その他の方法](#)”を参照してください。



注意

サービス配下で動作するCOBOLプログラムのデバッグのためにDISPLAY文を使用する場合は、入出力先にファイルを指定してください。なお、小入出力機能での入出力先については、“[11.1.2 入出力先の種類と指定方法](#)”を参照してください。

5.3 実行環境情報の設定

ここでは、実行環境情報の種類と設定方法の関係およびそれぞれの設定方法について説明します。

5.3.1 実行環境情報の種類

COBOLのアプリケーションを実行するために必要となる情報を実行環境情報といいます。

実行環境情報は、環境変数情報とエントリ情報の2種類に分けられます。環境変数情報とは、コンソールウィンドウの大きさ、コンソールフォント、ファイル識別名などを指定するための情報です。エントリ情報とは、動的プログラム構造の情報を示します。

実行環境情報の詳細については、“[5.4 実行環境情報の指定形式](#)”を参照してください。

5.3.2 実行環境情報の設定方法

実行環境情報の設定方法を以下に示します。

- a. コントロールパネルのシステムで設定する
- b. SETコマンドで設定する
- c. 実行用の初期化ファイルに設定する
- d. 実行環境設定ツールで設定する
- e. コマンドラインで設定する(実行時オプション、実行時パラメタ(OSIV系形式だけ))

実行用の初期化ファイルの環境変数情報は、COBOLの実行時にアプリケーションの環境変数に反映されます。

実行環境情報は、a.またはb.で設定することをおすすめします。また、エントリ情報は、エントリ情報ファイルに設定することをおすすめします。[参照]“[5.4.2 副プログラムのエントリ情報](#)”



注意

- ・ 実行用の初期化ファイルおよびエントリ情報ファイルに指定された実行環境情報は、COBOLプログラムの実行環境開設時に取り込まれるため、実行性能に影響します。したがって、以下のように設定することをおすすめします。
 - ー 環境変数情報は、プログラムの起動前にバッチファイルなどでユーザの環境変数に設定してください。
 - ー 実行用の初期化ファイルおよびエントリ情報ファイルには、実行するプログラムで必要な情報だけを設定してください。
- ・ Interstage Application ServerのCORBAワークユニットで動作する場合、ワークユニットの環境設定を行い、環境変数を設定してください。
ワークユニットの環境設定を行う方法については、“Interstage Application Server OLTPサーバ運用ガイド”を参照してください。

5.3.2.1 コントロールパネルのシステムで設定する方法

プログラムの実行前に、コントロールパネルのシステムで環境変数情報を設定します。設定方法については、ヘルプを参照してください。複数のアプリケーションに共通な環境変数情報の場合は、ここであらかじめ設定しておくくと便利です。

5.3.2.2 SETコマンドで設定する方法

この方法は、コマンドプロンプトまたはバッチファイルで、SETコマンドを使って環境変数情報を設定する方法です。

コマンドプロンプトで環境変数情報を設定すると、そのコマンドプロンプトから起動されたプログラムでは、その環境変数情報が有効となります。

また、バッチファイルを使うことによって、環境設定から実行までを1回の作業で行うこともできます。バッチファイルで設定した環境変数情報は、そのバッチファイルで起動したプログラムにだけ有効となります。

5.3.2.3 実行用の初期化ファイルに設定する方法

実行用の初期化ファイルを作成し、実行環境情報を設定する方法です。

5.3.2.3.1 実行用の初期化ファイルとは

実行用の初期化ファイルとは、NetCOBOLで作成したプログラムを実行するための情報を保存するファイルで、プログラムを実行するときに使用されます。

通常は、実行可能プログラム(以降、EXEファイルと略します)が格納されているフォルダの“COBOL85.CBR”を実行用の初期化ファイルとして扱います。ただし、アプリケーションの作成方法によっては、以下のファイルを実行用の初期化ファイルとして扱うこともできます。

- ・ ダイナミックリンクライブラリ(以降、DLLと略します)が格納されているフォルダのCOBOL85.CBR
- ・ COBOL85.CBR以外の名前で作成したファイル

DLLが格納されているフォルダのCOBOL85.CBRを実行用の初期化ファイルとして扱う場合は、“[5.3.2.3.4 DLL配下の実行用の初期化ファイルの使用について](#)”を参照してください。また、COBOL85.CBR以外の名前で作成したファイルを実行用の初期化ファイルとして扱う場合については、以下を参照してください。

- ・ 他言語(CプログラムまたはVisual Basicプログラム)を主プログラムとする場合は、“[5.7 実行用の初期化ファイル名の指定形式](#)”を参照してください。
- ・ 環境変数情報で指定する場合は、“[5.4.1.3 @CBR_CBRFILE\(実行用の初期化ファイルの指定\)](#)”を参照してください。
- ・ コマンドラインのオプションで指定する場合は、“[5.3.2.5 コマンドラインで設定する方法](#)”を参照してください。

なお、実行用の初期化ファイルがなくても、プログラムは実行できます。

5.3.2.3.2 実行用の初期化ファイルの内容

実行用の初期化ファイルは、共通部といくつかのセクションから構成されます。

共通部にはそれぞれのプログラムに共通する環境変数情報を記述し、セクションにはプログラムごとの環境変数情報およびエントリ情報を記述します。セクションは、角括弧(“[]”)で囲んで記述し、次のセクション名が現れるまでを1つのセクションとみなします。

実行用の初期化ファイルの内容を以下に示します。

環境変数情報名= 設定内容	… [1]	共通部
[プログラム名]	… [2]	セクション (環境変数情報)
環境変数情報名= 設定内容	… [3]	
:		
[プログラム名.ENTRY]	… [4]	セクション (エントリ情報)
エントリ名= 設定内容	… [5]	
:		

注意

1つの行に2個以上の環境変数情報およびエントリ情報を記述することはできません。

[1] プログラムに共通する環境変数情報を記述します。環境変数情報の指定形式については、“[5.4.1 環境変数情報](#)”を参照してください。ここに記述した環境変数情報は、アプリケーションが終了するまで有効になります。

共通部に記述した環境変数情報は、マルチスレッドモードおよびシングルスレッドモードの両方のモードで有効になります。シングルスレッドモードの場合、実行するプログラムのプログラム名と同じ名前を持つセクションに、共通部と同じ環境変数情報が記述されていた場合、セクションに記述された内容が有効になります。

[2] プログラムごとの環境変数情報の開始を表します。セクション名には、COBOLの主プログラムのプログラム名を指定します。^(注1)このセクションは、1つのプログラム名に対して1つだけ記述できます。

[3] プログラムごとの環境変数情報を記述します。環境変数情報の指定形式については、“[5.4.1 環境変数情報](#)”を参照してください。ここに記述した環境変数情報は、アプリケーションが終了するまで有効になります。

マルチスレッドモードの場合、ここに記述した環境変数情報は無視されます。

[4] プログラムごとのエントリ情報の開始を表します。エントリ情報のセクション名には、COBOLの主プログラムのプログラム名^(注1)に“.ENTRY”を付加した名前を指定します。このセクションは、1つのプログラム名に対して1つだけ記述できます。

[5] プログラムごとのエントリ情報を記述します。^(注2)エントリ情報の指定形式については、“[5.4.2 副プログラムのエントリ情報](#)”を参照してください。ここに記述したエントリ情報は、COBOLの実行環境が終了するまで有効になります。

マルチスレッドモードの場合、ここに記述したエントリ情報は無視されます。

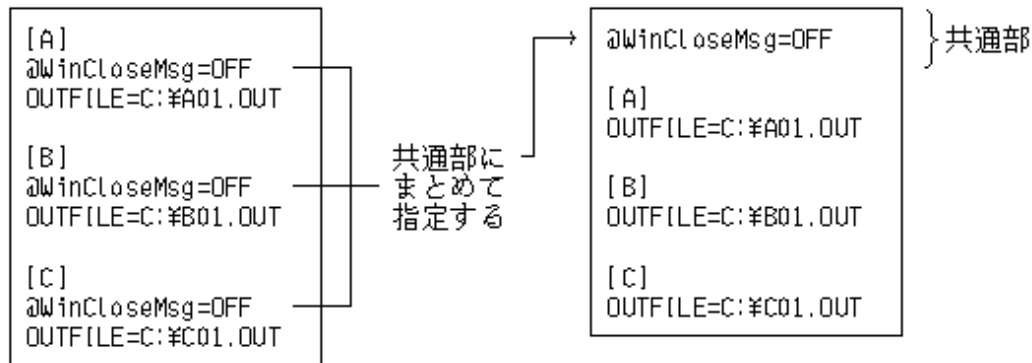
注1：他言語プログラムからCOBOLプログラムをJMPCINT2およびJMPCINT3を使用して呼び出す場合、最初に呼び出すCOBOLプログラムのプログラム名を指定してください。COBOLの主プログラムおよび実行環境については、“[10.1.4 COBOLの言語間の環境](#)”を参照してください。

注2：エントリ情報は、エントリ情報ファイルでも指定できます。実行用の初期化ファイルの場合、エントリ情報はプログラムごとにしか指定できませんが、エントリ情報ファイルではプログラムで共通な情報を指定できます。このため、エントリ情報ファイルで指定する方法をおすすめしています。エントリ情報の詳細については、“[5.4.2 副プログラムのエントリ情報](#)”を参照してください。

注意

- 実行用の初期化ファイルのエントリ情報は、マルチスレッドモードでは無効になります。エントリ情報ファイルに情報を指定してください。

- 複数のセクションで同じ環境変数情報の記述を行う場合、共通部に記述するとファイルサイズを節約できます。



- 同一の環境変数名、または副プログラム名を複数個指定しないでください。同一の環境変数名、または副プログラム名を複数個指定した場合の動作については保証されません。



例

実行用の初期化ファイルの記述例

```

@MessOutFile=MESSAGE. TXT
@Cns1WinSize=(80, 24)
@Cns1BufLine=100
@WinCloseMsg=ON
@IconName=COB85EXE

[PROG1]
INFILE=C:\¥FILE¥INFILE. TXT

[PROG1. ENTRY]
A=X. DLL
B=Y. DLL

```

 ↑
 共通部 (環境変数情報)
 ↓

 ↑
 PROG1の環境変数情報
 ↓

 ↑
 PROG1のエントリ情報
 ↓

実行用の初期化ファイル中のコメントの記述方法

行の先頭に;(セミコロン)がある場合、セミコロンから改行までの間はコメントとして認識されます。

コメント行が多い場合、コメント行の読み飛ばし処理のために、処理速度が低下する可能性があります。

5.3.2.3.3 実行用の初期化ファイルの検索順序について

実行用の初期化ファイルの検索順序を以下に示します。

1. EXEファイルのフォルダ配下のCOBOL85.CBR
2. DLLエントリオブジェクトを結合したDLLのうち、最初に動作するDLLのフォルダ配下のCOBOL85.CBR
3. 環境変数情報@CBR_CBRFILEに指定された実行用の初期化ファイル



例

検索順序の例



上記では、実行用の初期化ファイルの検索順序は次のようになります。

1. C:¥APL01¥COBOL85.CBR
2. 1.がない場合、C:¥DLL01¥COBOL85.CBR
3. 1.および2.がない場合、C:¥CBR¥TEST.CBR

ただし、JMPCINTC/JMPCINTBで実行用の初期化ファイルを指定した場合またはコマンドラインで実行用の初期化ファイルを指定した場合には、上記の検索は行われずに指定したファイルが有効になります。[参照]“[5.7 実行用の初期化ファイル名の指定形式](#)”、“[5.3.2.5 コマンドラインで設定する方法](#)”

環境変数情報@CBR_CBRINFO=YESを指定すると、使用している実行用の初期化ファイルの情報を得ることができます。この情報は、実行環境の開設時に実行時メッセージで通知されます。[参照]“[5.4.1.4 @CBR_CBRINFO \(簡略化した動作状態を出力する指定\)](#)”

実行用の初期化ファイルの読み込み位置についてまとめると、次のようになります。ただし、実行用の初期化ファイル名を環境変数などで直接指定しない場合の動作です。

ここで説明しているEXEファイルは、COBOLプログラムおよび他言語プログラムを指します。また、DLLは、DLLエントリオブジェクトを結合し、COBOLの実行環境開設時にローディングされているDLLを指します。

EXEファイルがあるフォルダ	DLLがあるフォルダ	
	COBOL85.CBRあり	COBOL85.CBRなし
COBOL85.CBRあり	EXEファイルがあるフォルダのCOBOL85.CBRが有効	EXEファイルがあるフォルダのCOBOL85.CBRが有効
COBOL85.CBRなし	DLLがあるフォルダのCOBOL85.CBRが有効	—

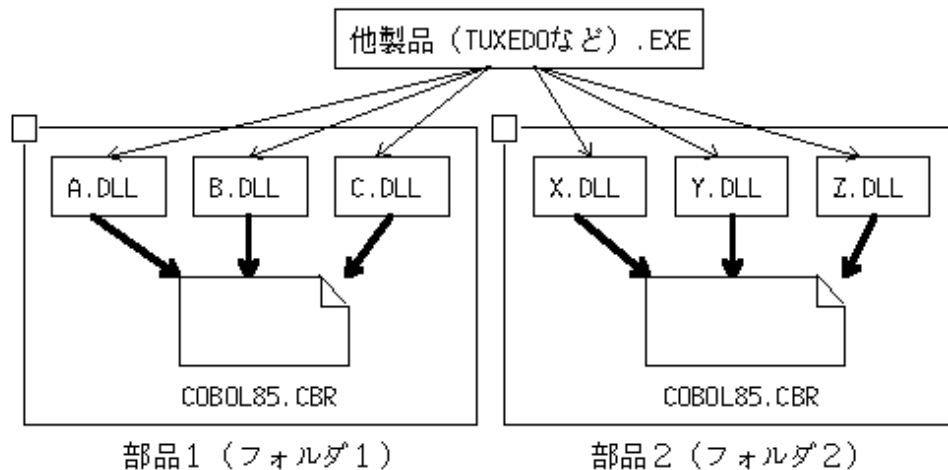
5.3.2.3.4 DLL配下の実行用の初期化ファイルの使用について

通常、実行用の初期化ファイルは、EXEファイルのあるフォルダ配下のCOBOL85.CBRが使用されます。このため、COBOLのアプリケーションを部品化して他のプログラムから呼び出す場合、以下のような問題があります。

1. 部品化したCOBOLアプリケーション(DLL)を起動するEXEファイル(他製品)が格納されているフォルダに、COBOL85.CBRを置く必要がある。
→ 他製品のEXEファイルの格納フォルダを意識しなければならない。
2. EXEファイルのあるフォルダのCOBOL85.CBRに、部品化したCOBOLアプリケーションの情報をすべて登録しなければならない。
→ COBOL85.CBRのファイルサイズが大きくなり、性能が劣化する。

上記の問題を解決する方法として、部品化したCOBOLアプリケーション(DLL)があるフォルダにCOBOL85.CBRを置いて使用する方法があります。この方法を使うことで、上記の問題が解決されます。

- ・ COBOLアプリケーション(DLL)のフォルダ配下にCOBOL85.CBRを置くため、他製品のEXEファイルの格納場所を意識しなくてもよい。
- ・ それぞれのアプリケーション用の情報だけを記述したCOBOL85.CBRを使用することができる。



たとえば上記の場合、同じプロセス(実行環境)単位で動作するアプリケーション(A.DLL、B.DLL、C.DLL)を、このプロセス単位で有効なCOBOL85.CBRとともにフォルダ1に置きます。他製品から部品1を起動すると、フォルダ1のCOBOL85.CBRの情報が有効になり、そのプロセスが終了するまでその実行環境情報が保証されます。同じように、他製品から部品2を起動すると、フォルダ2のCOBOL85.CBRの情報が、部品2のプロセス単位で有効になります。

このように、DLLのあるフォルダ配下のCOBOL85.CBRを使用する場合、DLLの作成時に、DLLエントリオブジェクトを結合する処理が必要になります。

注意

- DLLエントリオブジェクトを結合して作成したDLLは、実行環境の開設時にメモリ上にロードされていなければなりません。たとえば、動的プログラム構造を使用している場合は、実行環境の開設後にメモリ上にDLLがロードされるため、DLLのあるフォルダ配下のCOBOL85.CBRは使用できません。
- 実行用の初期化ファイルは、EXEファイルのあるフォルダから検索されるため、EXEファイルのあるフォルダ配下にCOBOL85.CBRを置かないでください。
- DLLエントリオブジェクトを結合して作成したDLLは、プロセス(実行環境)単位の情報を持つCOBOL85.CBRとともに、プロセス(実行環境)単位で1つのフォルダに格納してください。DLLがプロセス(実行環境)単位で1つのフォルダにない場合、実行時に意図しない動作を取る原因になります。
- COBOL85.CBRはプロセス(実行環境)ごとに有効になります。このため、それぞれのアプリケーションで同一の環境変数情報に別の値を割り当てたい場合は、それぞれのアプリケーションを別プロセスとして起動してください。

DLLエントリオブジェクトの結合

DLLエントリオブジェクトを結合して、DLLを作成する方法について説明します。

DLLを作成する場合、リンク時に以下のDLLエントリオブジェクトを結合します。

- F3BICBDM.OBJ : COBOL のDLL を作成する場合に指定します。
- F3BIMLDM.OBJ : COBOL と他言語を結合してDLL を作成する場合に指定します。

COBOLのDLLを作成する場合

```
LINK COBSUB.OBJ F3BICBDM.OBJ F3BICIMP.LIB KERNEL32.LIB /ENTRY:COBDMAN /DLL /OUT:COBSUB.DLL
```

COBSUB.OBJ : COBOL プログラムのオブジェクトファイル
 F3BICBDM.OBJ : DLL エントリオブジェクトファイル
 F3BICIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
 KERNEL32.LIB : Windows関数のインポートライブラリ



注意

LINKオプション"/ENTRY"に、必ず"COBDMAIN"を指定してください。

COBOLと他言語を結合してDLLを作成する場合

LINK CPROG.OBJ COBSUB.OBJ F3BIMLDM.OBJ F3BICIMP.LIB KERNEL32.LIB C ランタイムライブラリ /DLL /OUT:CPROG.DLL

CPROG.OBJ : C プログラムのオブジェクトファイル
COBSUB.OBJ : COBOL プログラムのオブジェクトファイル
F3BIMLDM.OBJ : DLL エントリオブジェクトファイル
F3BICIMP.LIB : COBOL ランタイムシステムのインポートライブラリ
KERNEL32.LIB : Windows関数のインポートライブラリ
Cランタイムライブラリ :
C プログラムを/ML オプションで翻訳した場合はLIBC.LIBを指定してください。
C プログラムを/MD オプションで翻訳した場合はMSVCRT.LIBを指定してください。



注意

プロジェクトマネージャウィンドウまたはWINLINKウィンドウでDLLのリンクを行う場合、[リンクオプション]ダイアログで以下の設定を行ってください。

- DLLのエントリオブジェクトで“言語間結合用”を選択する。
- Cランタイムライブラリに“MSVCRT.LIB”を指定する。

5.3.2.4 実行環境設定ツールで設定する方法

この方法は、実行環境設定ツールを使って、実行環境情報の設定を行う方法です。[参照]“5.6 実行環境設定ツール”

5.3.2.5 コマンドラインで設定する方法

この方法は、実行するプログラムが格納されたファイルをコマンド名として起動するときに、実行環境情報の内容をコマンドの引数として指定する方法です。この方法では、OSIV系形式の実行時パラメタ(環境変数情報@MGPRM)、実行用の初期化ファイル名および実行時オプション(環境変数情報@GOPT)を指定することができます。[参照]“5.4.1.60 @MGPRM(OSIV系形式の実行時パラメタの指定)”、“5.4.1.56 @GOPT(実行時オプションの指定)”

以下に指定形式を示します。

実行可能ファイル [実行時パラメタ] [-CBR 実行用の初期化ファイル名] [-CBL 実行時オプション]



注意

-CBRおよび-CBLは順不同です。

OSIV系形式の実行時パラメタの指定方法

OSIV系形式で実行時パラメタを使用する場合は、コマンド名の直後に指定した引数が、OSIV系形式の実行時パラメタとみなされます。



例

PROG1.EXE "ABCDE"

OSIV系形式の実行時パラメタとして、ABCDEが指定されました。

実行用の初期化ファイル名の指定方法

実行用の初期化ファイル名は、識別子-CBRまたは/CBRの後ろに指定します。空白を含むファイル名を指定する場合は、ファイル名を二重引用符(“)で囲む必要があります。

実行用の初期化ファイル名の指定形式については、“[5.7 実行用の初期化ファイル名の指定形式](#)”を参照してください。



例

```
PROG1.EXE -CBR ABC.INI
```

実行用の初期化ファイル名として、ABC.INIが指定されました(PROG1.EXEが主プログラムの場合)。

実行時オプションの指定方法

実行時オプションは、識別子-CBLまたは/CBLの後ろに指定します。

実行時オプションの指定形式については、“[5.8 実行時オプションの指定形式](#)”を参照してください。



例

```
PROG1.EXE -CBL r20 c20
```

実行時オプションとして、r20とc20が指定されました。

5.4 実行環境情報の指定形式

ここでは、実行環境情報の指定形式について説明します。

5.4.1 環境変数情報

以下に、環境変数情報の一覧を示します。



参考

コマンドライン(実行時オプション、実行時パラメタ(OSIV系形式だけ))、実行用の初期化ファイルおよび環境変数に設定した実行環境情報が重複した場合の優先順位は以下のとおりです。

1. コマンドライン(実行時オプション、実行時パラメタ(OSIV系形式だけ))の値
2. 実行用の初期化ファイルの値
3. 環境変数の値

実行時オプションに関するもの

- ・ “[5.4.1.56 @GOPT](#)(実行時オプションの指定)”
- ・ “[5.4.1.60 @MGPRM](#)(OSIV系形式の実行時パラメタの指定)”

プログラムの実行に関するもの

- ・ “[5.4.1.3 @CBR_CBRFILE](#)(実行用の初期化ファイルの指定)”
- ・ “[5.4.1.4 @CBR_CBRINFO](#)(簡略化した動作状態を出力する指定)”
- ・ “[5.4.1.23 @CBR_ENTRYFILE](#)(エントリ情報ファイルの指定)”
- ・ “[5.4.1.55 @ExitSessionMSG](#)(COBOLアプリケーション起動中のWindows終了指定)”

- “5.4.1.57 @IconDLL (アイコンリソースのDLL名の指定)”
- “5.4.1.58 @IconName (アイコンリソースの識別名の指定)”
- “5.4.1.67 @ShowIcon (NetCOBOLのアイコン表示の抑止指定)”

スクリーン操作に関するもの

- “5.4.1.45 @CBR_SCREEN_POSITION (スクリーン画面の表示位置の指定)”
- “5.4.1.44 @CBR_SCR_KEYDEFFILE (スクリーン操作のキー定義ファイルの指定)”
- “5.4.1.65 @ScrnFont (スクリーン操作で使用するフォントの指定)”
- “5.4.1.66 @ScrnSize (スクリーン操作の論理画面の大きさの指定)”

小入出力に関するもの

- “5.4.1.10 @CBR_CONSOLE (コンソールウィンドウの種別の指定)”
- “5.4.1.13 @CBR_DISPLAY_CONSOLE_EVENTLOG_LEVEL (DISPLAY UPON CONSOLEのイベントログ出力時のイベント種類指定)”
- “5.4.1.14 @CBR_DISPLAY_CONSOLE_EVENTLOG_SRCNAME (DISPLAY UPON CONSOLEのイベントログ出力時のイベントソース名指定)”
- “5.4.1.15 @CBR_DISPLAY_CONSOLE_OUTPUT (DISPLAY UPON CONSOLEのイベントログ出力指定)”
- “5.4.1.16 @CBR_DISPLAY_SYSERR_EVENTLOG_LEVEL (DISPLAY UPON SYSERRのイベントログ出力時のイベント種類指定)”
- “5.4.1.17 @CBR_DISPLAY_SYSERR_EVENTLOG_SRCNAME (DISPLAY UPON SYSERRのイベントログ出力時のイベントソース名指定)”
- “5.4.1.18 @CBR_DISPLAY_SYSERR_OUTPUT (DISPLAY UPON SYSERRのイベントログ出力指定)”
- “5.4.1.19 @CBR_DISPLAY_SYSOUT_EVENTLOG_LEVEL (DISPLAY UPON SYSOUTのイベントログ出力時のイベント種類指定)”
- “5.4.1.20 @CBR_DISPLAY_SYSOUT_EVENTLOG_SRCNAME (DISPLAY UPON SYSOUTのイベントログ出力時のイベントソース名指定)”
- “5.4.1.21 @CBR_DISPLAY_SYSOUT_OUTPUT (DISPLAY UPON SYSOUTのイベントログ出力指定)”
- “5.4.1.31 @CBR_JOBDATE (任意の日付を取得)”
- “5.4.1.51 @CnslBufLine (コンソールウィンドウのバッファ数の指定)”
- “5.4.1.52 @CnslFont (コンソールウィンドウのフォントの指定)”
- “5.4.1.53 @CnslWinSize (コンソールウィンドウの大きさの指定)”
- “5.4.1.68 @WinCloseMsg (ウィンドウを閉じるときのメッセージ表示の指定)”
- “5.4.1.6 @CBR_COMPOSER_CONSOLE (Interstage Business Application Serverのログ定義ファイルで定義されている管理名の指定)”
- “5.4.1.8 @CBR_COMPOSER_SYSERR (Interstage Business Application Serverのログ定義ファイルで定義されている管理名の指定)”
- “5.4.1.9 @CBR_COMPOSER_SYSOUT (Interstage Business Application Serverのログ定義ファイルで定義されている管理名の指定)”

メッセージに関するもの

- “5.4.1.33 @CBR_MESSAGE (実行時メッセージの出力先の指定)”
- “5.4.1.34 @CBR_MESS_LEVEL_CONSOLE (実行時メッセージの重大度指定)”
- “5.4.1.35 @CBR_MESS_LEVEL_EVENTLOG (実行時メッセージの重大度指定)”

- “5.4.1.46 @CBR_SYSERR_EXTEND(SYSERR出力情報の拡張の指定)”
- “5.4.1.59 @MessOutFile(メッセージを出力するファイルの指定)”
- “5.4.1.61 @NoMessage(実行時メッセージおよびSYSERRの出力抑止指定)”
- “5.4.1.7 @CBR_COMPOSER_MESS(実行時メッセージをInterstage Business Application Serverの汎用ログに出力する指定)”

ファイルに関するもの

- “5.4.1.1 @AllFileExclusive(ファイルの排他処理の指定)”
- “5.4.1.50 @CBR_TRAILING_BLANK_RECORD(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)”
- “5.4.1.69 ファイル識別名(プログラムで使用するファイルの指定)”
- “5.4.1.79 SYSINのアクセス名(小入出力機能の入力ファイルの指定)”
- “5.4.1.80 SYSOUTのアクセス名(小入出力機能の出力ファイルの指定)”
- “5.4.1.29 @CBR_FILE_USE_MESSAGE(入出力エラーの実行時メッセージの出力)”
- “5.4.1.26 @CBR_FILE_BOM_READ(Unicodeの行順ファイルを参照する時の識別コードの扱いの指定)”
- “5.4.1.24 @CBR_EXFH_API(外部ファイルハンドラで結合するファイルシステムの入り口名の指定)”
- “5.4.1.25 @CBR_EXFH_LOAD(外部ファイルハンドラで結合するファイルシステムのDLL名の指定)”
- “5.4.1.28 @CBR_FILE_SEQUENTIAL_ACCESS(ファイルの高速処理を一括して有効にする指定)”
- “5.4.1.27 @CBR_FILE_LFS_ACCESS(COBOLファイルのサイズを拡張する指定)”

表示ファイルに関するもの

- “5.4.1.43 @CBR_PSFILE_xxx(表示ファイルから使用する接続製品名(宛先ごとの指定))”
- “5.4.1.70 ファイル識別名(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)”

プリンタに関するもの

- “5.4.1.22 @CBR_DocumentName_xxxx(I制御レコードによる文書名の指定)”
- “5.4.1.37 @CBR_OverlayPrintOffset(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)”
- “5.4.1.38 @CBR_OverlayPrintSPEC(フォームオーバーレイ印刷方法の指定)”
- “5.4.1.39 @CBR_PrinterANK_Size(ANK文字サイズの指定)”
- “5.4.1.40 @CBR_PrintFontTable(印刷ファイルで使用するフォントテーブルの指定)”
- “5.4.1.41 @CBR_PrintInfoFile(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)”
- “5.4.1.42 @CBR_PrintTextPosition(文字配置座標の計算方法の指定)”
- “5.4.1.47 @CBR_TextAlign(文字行内配置時の上端/下端合わせの指定)”
- “5.4.1.54 @DefaultFCB_Name(デフォルトFCB名の指定)”
- “5.4.1.63 @PrinterFontName(印刷ファイルで使用するフォントの指定)”
- “5.4.1.64 @PRN_FormName_xxx(用紙名の指定)”
- “5.4.1.71 ファイル識別名(プログラムで使用するプリンタ情報ファイルおよび各種パラメタの指定)”
- “5.4.1.72 ファイル識別名(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)”
- “5.4.1.73 FCBxxxx(FCB制御文の指定)”
- “5.4.1.74 FOVLDIR(フォームオーバーレイパターンフォルダの指定)”
- “5.4.1.75 FOVLTYPE(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)”

- ・ “5.4.1.76 FOVLNAME (フォームオーバーレイパターンのファイル名の指定)”
- ・ “5.4.1.77 OVD_SUFFIX (フォームオーバーレイパターンのファイルの拡張子の指定)”

デバッグに関するもの

- ・ “5.4.1.2 @CBR_ATTACH_TOOL (プログラムからデバッグまたは診断機能を起動する指定)”
- ・ “5.4.1.32 @CBR_JUSTINTIME_DEBUG (異常終了時にデバッグまたは診断機能を使って調査を行う指定)”
- ・ “5.4.1.36 @CBR_MEMORY_CHECK (メモリチェック機能を使って検査を行う指定)”
- ・ “5.4.1.48 @CBR_TRACE_FILE (トレース情報の出力ファイルの指定)”
- ・ “5.4.1.49 @CBR_TRACE_PROCESS_MODE (TRACEファイルのプロセス毎の出力指定)”
- ・ “5.4.1.78 SYSCOUNT (COUNT情報の出力ファイルの指定)”

簡易アプリ間通信に関するもの

- ・ “5.4.1.5 @CBR_CIINF (論理宛先定義ファイルの指定)”

リモートデータベースアクセスに関するもの

- ・ “5.4.1.62 @ODBC_Inf (ODBC情報ファイルの指定)”

マルチスレッドに関するもの

- ・ “@CBR_SYMFOWARE_THREAD (Symfoware連携でマルチスレッド動作可能にする指定)”
- ・ “@CBR_THREAD_MODE (スレッドモードの指定)”
- ・ “@CBR_THREAD_TIMEOUT (スレッド同期制御サブルーチンの待ち時間の指定)”
- ・ “@CBR_SSIN_FILE (スレッド単位に入力ファイルをオープンする指定)”

オブジェクト指向に関するもの

- ・ “18.3.4.1.1 @CBR_ClassInfFile (クラス情報ファイルの指定)”
- ・ “18.3.4.2.1 InstanceBlockセクション (オブジェクトインスタンスの格納数の指定)”

組み込み関数に関するもの

- ・ “5.4.1.30 @CBR_FUNCTION_NATIONAL (NATIONAL関数の変換モードの指定)”

CSV形式データの操作に関するもの

- ・ “5.4.1.11 @CBR_CSV_OVERFLOW_MESSAGE (CSV形式データ操作時のメッセージ抑止指定)”
- ・ “5.4.1.12 @CBR_CSV_TYPE (生成するCSV形式のバリエーション)”



注意

フォルダ名およびファイル名の指定方法

フォルダ名またはファイル名にコンマ(,)を含む場合、フォルダ名またはファイル名を二重引用符(”)で囲む必要があります。

5.4.1.1 @AllFileExclusive (ファイルの排他処理の指定)

@AllFileExclusive = $\left\{ \begin{array}{c} \text{YES} \\ \end{array} \right\}$

NO

プログラムを実行するときに、COBOLソースプログラムに記述したファイルの排他に関する処理を無視して、無条件にすべてのファイルを排他処理する(YES)か、COBOLソースプログラムに記述したファイルの排他に関する処理どおりに実行する(NO)か、を指定します。排他制御を指定した場合、プログラムで使用しているファイルは、他のプログラムからはアクセス不可能(オープンエラー)となります。

参考

ファイルを排他処理することにより、ファイルアクセス時間が短縮される場合があります。

5.4.1.2 @CBR_ATTACH_TOOL(プログラムからデバグまたは診断機能を起動する指定)

$$@CBR\_ATTACH\_TOOL = \left\{ \begin{array}{l} \text{[接続先/]TEST [起動パラメタ]} \\ \text{SNAP 起動パラメタ} \end{array} \right\}$$

プログラムからデバグまたは診断機能を起動する場合に指定します。

TEST

デバグを起動します。デバグの起動パラメタについては、“表20.1 起動パラメタ”を参照してください。

接続先を指定した場合は、リモートデバグを使用します。リモートデバグの起動パラメタについては、“表20.7 起動パラメタ”を参照してください。

接続先には、クライアント側のリモートデバグコネクタのポート番号と動作しているコンピュータを以下の形式で指定します。

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

IPアドレスは、IPv4またはIPv6の形式で指定します。

ポート番号は、1024から65535の範囲の数字を指定します。ポート番号を省略した場合は、59999が指定されたと見なされます。

IPv6形式の場合は、スコープ付きアドレスを指定することができます。スコープ付きアドレスを指定する場合、アドレスの後にスコープ識別子を指定します。

なお、IPv6アドレスにポート番号を指定する場合は、アドレス部分を中括弧([])で囲んでください。

例

[IPv4アドレス(192.168.0.1)およびポート番号(2000)での指定]

@CBR_ATTACH_TOOL=192.168.0.1:2000/TEST

[IPv6アドレス(fe80::1:23:456:789a)およびポート番号(2000)での指定]

@CBR_ATTACH_TOOL=[fe80::1:23:456:789a]:2000/TEST

[IPv6アドレス(fe80::1:23:456:789a)およびスコープ識別子(%eth0)での指定]

@CBR_ATTACH_TOOL=fe80::1:23:456:789a%eth0/STUDIO

[IPv6アドレス(fe80::1:23:456:789a)、スコープ識別子(%5)およびポート番号(2000)での指定]

@CBR_ATTACH_TOOL=[fe80::1:23:456:789a%5]:2000/STUDIO

〔ホスト名 (client-1) およびポート番号 (2000) での指定〕
@CBR_ATTACH_TOOL=client-1:2000/TEST

〔ポート番号省略の指定 (IPv4アドレス)〕
@CBR_ATTACH_TOOL=192.168.0.1/TEST

〔ポート番号省略の指定 (IPv6アドレス)〕
@CBR_ATTACH_TOOL=fe80::1:23:456:789a/TEST

.....

SNAP

診断機能を起動します。診断機能の起動パラメタについては、“[表19.4 起動パラメタ](#)”を参照してください。



注意

.....

- ・ デバッガなどでデバッグしている場合は無効です。

.....

本機能において、以下の範囲でIPv6アドレスに対応しています。

表5.1 デバッガのIPv6対応

項目	仕様
対応可能なネットワーク	以下の環境に対応しています。 ・ IPv4/IPv6 デュアルスタック環境 ・ IPv6 のみの環境
使用可能なIPv6アドレス	以下のIPv6 アドレスが使用できます。 ・ グローバルユニキャストアドレス(GUA) ・ ユニークローカルアドレス(ULA) ・ リンクローカルアドレス(LLA) また、IP アドレスのインターフェースID 部は任意をサポートしています。
IPv4/IPv6のアドレス選択	IPv4/IPv6デュアルスタック環境においてはIPv6を優先的に使用します。 以下は、対応していません。 ・ IPv4のみまたはIPv6のみの使用限定 ・ 優先順位の切り替え ・ IPv6による接続に失敗した場合のIPv4による接続(フォールバック)
IP アドレスの表記	以下のIPv6アドレス表記をサポートしています。 1. ポート番号の指定 ([])の有無 2. 通常の16進法による表記 3. 表記の単純化ルール(RFC5952)による表記 4. 英字部の大文字表記 5. 英字部の小文字表記 6. 英字部の大小文字混在表記 IPv4/IPv6 デュアルスタックOS 時やIPv4/IPv6 併用時(デュアルスタック時)にIPv4 アドレスを指定する場合は、従来どおりのIPv4 アドレス表記を使用してください。



注意

一定時間ごとに変わる一時アドレス(プライバシ拡張)は、サポートしていません。IP アドレスが変更された場合は、リモートデバッガコネクタおよびデバッガの再起動が必要です。

5.4.1.3 @CBR_CBRFILE(実行用の初期化ファイルの指定)

@CBR_CBRFILE = 実行用の初期化ファイル名

使用する実行用の初期化ファイル名を指定します。

JMPCINTC/JMPCINTBおよびコマンドラインで実行用の初期化ファイル名が指定されておらず、EXEファイルおよびDLLの存在するフォルダ配下にCOBOL85.CBRファイルがない場合に、有効となります。

ファイル名には、絶対パスと相対パスが指定できます。相対パスが指定された場合は、EXEからの相対パスになります。[参照]“5.3.2.3 実行用の初期化ファイルに設定する方法”

5.4.1.4 @CBR_CBRINFO(簡略化した動作状態を出力する指定)

@CBR_CBRINFO = YES

COBOLプログラムの実行時の情報を実行環境の開設時に実行時メッセージ(JMP0070I-I)として出力します。出力される情報には、ランタイムシステムのバージョンレベル、スレッドモードおよび実行用の初期化ファイル名などがあります。[参照]“メッセージ説明書”の“第3章 実行時メッセージ”

5.4.1.5 @CBR_CIINF(論理宛先定義ファイルの指定)

@CBR_CIINF = 論理宛先定義ファイル名

簡易アプリ間通信機能を使用する場合、論理宛先を定義したファイル名を指定します。

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

5.4.1.6 @CBR_COMPOSER_CONSOLE(Interstage Business Application Serverのログ定義ファイルで定義されている管理名の指定)

@CBR_COMPOSER_CONSOLE = ログ定義ファイルで定義されている管理名

機能名CONSOLEに対応づけた呼び名を指定した小入出力の出力先に、Interstage Business Application Serverの汎用ログを使用する場合に指定します。



注意

- ・ DISPLAY文の汎用ログへの出力レベルは5です。

- Interstage Business Application Serverの汎用ログに出力する場合、標準出力(stdout)には出力されません。

5.4.1.7 @CBR_COMPOSER_MESS(実行時メッセージをInterstage Business Application Serverの汎用ログに出力する指定)

@CBR_COMPOSER_MESS = ログ定義ファイルで定義されている管理名

ランタイムシステムが出力する実行時メッセージを、Interstage Business Application Serverの汎用ログに出力する場合に指定します。



注意

実行時メッセージを汎用ログに出力する場合、以下の環境変数は無効になります。また、すべての重大度のメッセージが汎用ログに出力されます。

- @CBR_MESS_LEVEL_CONSOLE
- @CBR_MESS_LEVEL_EVENTLOG
- @NoMessage
- @CBR_MESSAGE
- @MessOutFile

5.4.1.8 @CBR_COMPOSER_SYSERR(Interstage Business Application Serverのログ定義ファイルで定義されている管理名の指定)

@CBR_COMPOSER_SYSERR = ログ定義ファイルで定義されている管理名

機能名SYSERRに対応づけた呼び名を指定した小入出力の出力先に、Interstage Business Application Serverの汎用ログを使用する場合に指定します。



注意

- DISPLAY文の汎用ログへの出力レベルは5です。
- Interstage Business Application Serverの汎用ログに出力する場合、標準エラー出力(stderr)やファイル出力などには出力されません。

5.4.1.9 @CBR_COMPOSER_SYSOUT(Interstage Business Application Serverのログ定義ファイルで定義されている管理名の指定)

@CBR_COMPOSER_SYSOUT = ログ定義ファイルで定義されている管理名

UPON指定なし、または機能名SYSOUTに対応づけた呼び名を指定した小入出力の出力先に、Interstage Business Application Serverの汎用ログを使用する場合に指定します。



注意

- ・ DISPLAY文の汎用ログへの出力レベルは5です。
- ・ Interstage Business Application Serverの汎用ログに出力する場合、標準出力(stdout)には出力されません。

5.4.1.10 @CBR_CONSOLE(コンソールウィンドウの種別の指定)

```
@CBR_CONSOLE = { SYSTEM
                  COBOL }
```

小入出力機能でデータの入出力に使用するコンソールウィンドウおよび実行時メッセージの出力先として使用するコンソールウィンドウの種別を指定します。

SYSTEM

システムのコンソールを使用します。

COBOL

COBOLのコンソールウィンドウおよびメッセージボックスを使用します。

省略された場合、COBOLが主プログラムであり、かつ、翻訳オプションMAIN(MAIN)を指定して翻訳した場合は、システムのコンソールを使用します。その他の場合(翻訳オプションがMAIN(WINMAIN)または他言語が主プログラムの場合は、COBOLのコンソールウィンドウを使用します。



注意

システムのコンソールを使用する場合、環境変数情報@CnslBufLine、@CnslFontおよび@CnslWinSizeは有効になりません。

5.4.1.11 @CBR_CSV_OVERFLOW_MESSAGE(CSV形式データ操作時のメッセージ抑止指定)

```
@CBR_CSV_OVERFLOW_MESSAGE = NO
```

STRING文(書き方2)およびUNSTRING文(書き方2)の実行時に出力される以下のメッセージを抑止します。

- ・ JMP0262I-W
- ・ JMP0263I-W

5.4.1.12 @CBR_CSV_TYPE(生成するCSV形式のバリエーション)

```
@CBR_CSV_TYPE = { MODE-1
                   MODE-2
                   MODE-3
                   MODE-4 }
```

STRING文(書き方2)で生成するCSV形式のバリエーションを指定します。

この指定は、TYPE指定を省略したSTRING文だけに有効です。

生成されるCSV形式の詳細は、“[13.4 CSV形式のバリエーション](#)”を参照してください。

5.4.1.13 @CBR_DISPLAY_CONSOLE_EVENTLOG_LEVEL (DISPLAY UPON CONSOLEのイベントログ出力時のイベント種類指定)

$$@CBR\_DISPLAY\_CONSOLE\_EVENTLOG\_LEVEL = \left\{ \begin{array}{c} I \\ W \\ E \end{array} \right\}$$

DISPLAY UPON CONSOLEの出力をイベントログに出力するときのイベント種類を指定します。

@CBR_DISPLAY_CONSOLE_OUTPUTが有効でない場合、本環境変数の値は意味を持ちません。

@CBR_DISPLAY_CONSOLE_EVENTLOG_LEVELに指定できるパラメタは以下のとおりです。

I

情報イベント。イベント種類には情報と表示されます。

W

警告イベント。イベント種類には警告と表示されます。

E

エラーイベント。イベント種類にはエラーイベントと表示されます。



例

イベント種類を警告としてイベントログに出力する

`@CBR_DISPLAY_CONSOLE_EVENTLOG_LEVEL=W`



注意

@CBR_DISPLAY_CONSOLE_EVENTLOG_LEVELの指定がない場合、または上記以外のパラメタが右辺に指定された場合、イベント種類はI(情報イベント)となります。

5.4.1.14 @CBR_DISPLAY_CONSOLE_EVENTLOG_SRCNAME (DISPLAY UPON CONSOLEのイベントログ出力時のイベントソース名指定)

@CBR_DISPLAY_CONSOLE_EVENTLOG_SRCNAME = イベントソース名

DISPLAY UPON CONSOLEの出力をイベントログに出力するときのイベントソース名を指定します。

@CBR_DISPLAY_CONSOLE_OUTPUTが有効でない場合、本環境変数の値は意味を持ちません。

イベントソース名

イベントソース名には、出力先コンピュータに設定したレジストリキー名を指定します。



@CBR_DISPLAY_CONSOLE_EVENTLOG_SRCNAMEの指定がない場合、“NetCOBOL Application”となります。

5.4.1.15 @CBR_DISPLAY_CONSOLE_OUTPUT (DISPLAY UPON CONSOLEのイベントログ出力指定)

@CBR_DISPLAY_CONSOLE_OUTPUT = EVENTLOG[(コンピュータ名)]

DISPLAY UPON CONSOLEの出力先をイベントログにします。

EVENTLOG

イベントログへ出力します。

コンピュータ名には、イベントログの出力先コンピュータ名を指定します。ネットワーク上の他のコンピュータ名を指定することができます。コンピュータ名を省略した場合は、プログラムを実行しているコンピュータに出力します。

以下の場合、実行中のコンピュータのイベントログに出力対象メッセージを出力します。

- ー ネットワーク上に存在しないコンピュータ名を指定した場合
- ー 指定したコンピュータで32ビットWindowsが動作していない場合



イベントログを出力するコンピュータには、必ず本製品をインストールしてください。

5.4.1.16 @CBR_DISPLAY_SYSERR_EVENTLOG_LEVEL (DISPLAY UPON SYSERRのイベントログ出力時のイベント種類指定)

@CBR_DISPLAY_SYSERR_EVENTLOG_LEVEL = $\left\{ \begin{array}{c} I \\ W \\ E \end{array} \right\}$

DISPLAY UPON SYSERRの出力をイベントログに出力するときのイベント種類を指定します。

@CBR_DISPLAY_SYSERR_OUTPUTが有効でない場合、本環境変数の値は意味を持ちません。

@CBR_DISPLAY_SYSERR_EVENTLOG_LEVELに指定できるパラメタは以下のとおりです。

I

情報イベント。イベント種類には情報と表示されます。

W

警告イベント。イベント種類には警告と表示されます。

E

エラーイベント。イベント種類にはエラーイベントと表示されます。



例

イベント種類を警告としてイベントログに出力する

```
@CBR_DISPLAY_SYSERR_EVENTLOG_LEVEL=W
```



注意

@CBR_DISPLAY_SYSERR_EVENTLOG_LEVELの指定がない場合、または上記以外のパラメタが右辺に指定された場合、イベント種類はI(情報イベント)となります。

5.4.1.17 @CBR_DISPLAY_SYSERR_EVENTLOG_SRCNAME(DISPLAY UPON SYSERRのイベントログ出力時のイベントソース名指定)

```
@CBR_DISPLAY_SYSERR_EVENTLOG_SRCNAME = イベントソース名
```

DISPLAY UPON SYSERRの出力をイベントログに出力するときのイベントソース名を指定します。

@CBR_DISPLAY_SYSERR_OUTPUTが有効でない場合、本環境変数の値は意味を持ちません。

イベントソース名

イベントソース名には、出力先コンピュータに設定したレジストリキー名を指定します。



注意

@CBR_DISPLAY_SYSERR_EVENTLOG_SRCNAMEの指定がない場合、“NetCOBOL Application”となります。

5.4.1.18 @CBR_DISPLAY_SYSERR_OUTPUT(DISPLAY UPON SYSERRのイベントログ出力指定)

```
@CBR_DISPLAY_SYSERR_OUTPUT = EVENTLOG[(コンピュータ名)]
```

DISPLAY UPON SYSERRの出力先をイベントログにします。

EVENTLOG

イベントログへ出力します。

コンピュータ名には、イベントログの出力先コンピュータ名を指定します。ネットワーク上の他のコンピュータ名を指定することができます。コンピュータ名を省略した場合は、プログラムを実行しているコンピュータに出力します。

以下の場合、実行中のコンピュータのイベントログに出力対象メッセージを出力します。

- ネットワーク上に存在しないコンピュータ名を指定した場合
- 指定したコンピュータで32ビットWindowsが動作していない場合



注意

イベントログを出力するコンピュータには、必ず本製品をインストールしてください。

5.4.1.19 @CBR_DISPLAY_SYSOUT_EVENTLOG_LEVEL(DISPLAY UPON SYSOUT のイベントログ出力時のイベント種類指定)

$$@CBR\_DISPLAY\_SYSOUT\_EVENTLOG\_LEVEL = \left\{ \begin{array}{c} I \\ W \\ E \end{array} \right\}$$

DISPLAY UPON SYSOUTの出力をイベントログに出力するときのイベント種類を指定します。

@CBR_DISPLAY_SYSOUT_OUTPUTが有効でない場合、本環境変数の値は意味を持ちません。

@CBR_DISPLAY_SYSOUT_EVENTLOG_LEVELに指定できるパラメタは以下のとおりです。

I

情報イベント。イベント種類には情報と表示されます。

W

警告イベント。イベント種類には警告と表示されます。

E

エラーイベント。イベント種類にはエラーイベントと表示されます。



例

イベント種類を警告としてイベントログに出力する

`@CBR_DISPLAY_SYSOUT_EVENTLOG_LEVEL=W`



注意

@CBR_DISPLAY_SYSOUT_EVENTLOG_LEVELの指定がない場合、または上記以外のパラメタが右辺に指定された場合、イベント種類はI(情報イベント)となります。

5.4.1.20 @CBR_DISPLAY_SYSOUT_EVENTLOG_SRCNAME(DISPLAY UPON SYSOUTのイベントログ出力時のイベントソース名指定)

@CBR_DISPLAY_SYSOUT_EVENTLOG_SRCNAME = イベントソース名

DISPLAY UPON SYSOUTの出力をイベントログに出力するときのイベントソース名を指定します。

@CBR_DISPLAY_SYSOUT_OUTPUTが有効でない場合、本環境変数の値は意味を持ちません。

イベントソース名

イベントソース名には、出力先コンピュータに設定したレジストリキー名を指定します。



注意

@CBR_DISPLAY_SYSOUT_EVENTLOG_SRCNAMEの指定がない場合、“NetCOBOL Application”となります。

5.4.1.21 @CBR_DISPLAY_SYSOUT_OUTPUT(DISPLAY UPON SYSOUTのイベントログ出力指定)

@CBR_DISPLAY_SYSOUT_OUTPUT = EVENTLOG[(コンピュータ名)]

DISPLAY UPON SYSOUTの出力先をイベントログにします。

EVENTLOG

イベントログへ出力します。

コンピュータ名には、イベントログの出力先コンピュータ名を指定します。ネットワーク上の他のコンピュータ名を指定することができます。コンピュータ名を省略した場合は、プログラムを実行しているコンピュータに出力します。

以下の場合、実行中のコンピュータのイベントログに出力対象メッセージを出力します。

- ー ネットワーク上に存在しないコンピュータ名を指定した場合
- ー 指定したコンピュータで32ビットWindowsが動作していない場合



注意

イベントログを出力するコンピュータには、必ず本製品をインストールしてください。

5.4.1.22 @CBR_DocumentName_xxxx(I制御レコードによる文書名の指定)

@CBR_DocumentName_xxxx = 文書名

FORMAT句なし印刷ファイルで、I制御レコードのDOC-INFO(文書名識別情報)フィールドに指定した任意の4文字以内の文字列(英数字)を本環境変数情報名のxxxx部分に置き換え、本環境変数情報名に対して文書名を対応付けて指定します。このとき、指定可能な文書名は、128バイト以内の英字/数字/カナ/日本語の組合せでなければなりません。[参照]“8.1.5 I制御レコード/S制御レコード”

なお、ここで指定した文書名は、Windowsシステムが提供するプリントマネージャまたはプリンタの画面に表示されます。



例

I制御レコードのDOC-INFOフィールドの指定

“ABCD”

xxxx部分置換後の環境変数情報名

@CBR_DocumentName_ABCD

環境変数情報名と文書名の対応付け

@CBR_DocumentName_ABCD=富士通パソコン売上伝票

5.4.1.23 @CBR_ENTRYFILE(エントリ情報ファイルの指定)

@CBR_ENTRYFILE = エントリ情報ファイル名

実行用の初期化ファイル(COBOL85.CBR)のエントリ記述部以外からエントリ情報を指定したい場合、エントリ情報ファイルを作成し、本環境変数情報に指定します。なお、エントリ情報については、“5.4.2 副プログラムのエントリ情報”または“18.2.4.4 クラスとメソッドのエントリ情報”を参照してください。

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

5.4.1.24 @CBR_EXFH_API(外部ファイルハンドラで結合するファイルシステムの入り口名の指定)

@CBR_EXFH_API = 入り口名

外部ファイルハンドラを使用する時、結合するファイルシステムの入り口名を指定します(必須)。



参照

[参照]“7.9.4 外部ファイルハンドラ”



注意

入り口名を指定する時、大文字と小文字を区別して記述してください。

5.4.1.25 @CBR_EXFH_LOAD(外部ファイルハンドラで結合するファイルシステムのDLL名の指定)

@CBR_EXFH_LOAD = DLL名

外部ファイルハンドラを使用する時、結合するファイルシステムのDLL名を指定します。

ファイルのパスには、絶対パス、相対パスのどちらも指定することができます。相対パスを用いた場合、カレントディレクトリからの相対パスとなります。



参照

“7.9.4 外部ファイルハンドラ”

5.4.1.26 @CBR_FILE_BOM_READ(Unicodeの行順ファイルを参照する時の識別コードの扱いの指定)

@CBR_FILE_BOM_READ =	{	CHECK	}
		DATA	
		AUTO	

Unicodeの行順ファイルを参照する場合、ファイルに付加されている認識コードの扱いを指定します。



参照

“7.3.3 行順ファイルの処理”

CHECK

レコード定義の字類と一致したBOMが付加されている場合は、識別コードと認識し、READ文の実行でBOMを読み飛ばします。
レコード定義の字類と一致していないBOMが付加されている場合、またはBOMが付加されていない場合は、OPEN文の実行が失敗します。

DATA

BOMが付加されている場合は、BOMをレコードデータの一部として読み込みます。
BOMが付加されていない場合は、ファイルの先頭からレコードを読み込みます。

AUTO

レコード定義の字類と一致したBOMが付加されている場合は、識別コードと認識し、READ文の実行でBOMを読み飛ばします。
レコード定義の字類と一致していないBOMが付加されている場合は、OPEN文の実行が失敗します。
BOMが付加されていない場合は、ファイルの先頭からレコードを読み込みます。

5.4.1.27 @CBR_FILE_LFS_ACCESS(COBOLファイルのサイズを拡張する指定)

@CBR_FILE_LFS_ACCESS = YES

COBOLファイルのファイルサイズをシステム制限まで拡張することができます。
ファイルサイズについては、“7.9 他のファイルシステムの使用法”を参照してください。



注意

- 以下の環境で本環境変数を指定する場合、全ての処理で本環境変数を有効にしてください。
 - 他のCOBOLプログラムとファイルを共有している
 - 対象となるCOBOLファイルを、ツールおよび他製品からアクセスしている
- 本環境変数が有効でない場合、COBOLファイルの排他制御の動作は保証されません。
- 相対ファイルに対して、本環境変数は有効になりません。本環境変数を指定しても、ファイルサイズは拡張されません。

5.4.1.28 @CBR_FILE_SEQUENTIAL_ACCESS(ファイルの高速処理を一括して有効にする指定)

@CBR_FILE_SEQUENTIAL_ACCESS = BSAM

ファイルの高速処理を一括して有効にします。



参照

“7.7.4 ファイルの高速処理”

5.4.1.29 @CBR_FILE_USE_MESSAGE(入出力エラーの実行時メッセージの出力)

@CBR_FILE_USE_MESSAGE = YES

有効な誤り処理手続きがある場合に、入出力エラーの実行時メッセージを出力します。

5.4.1.30 @CBR_FUNCTION_NATIONAL(NATIONAL関数の変換モードの指定)

@CBR_FUNCTION_NATIONAL = [{ MODE-1
MODE-2 }], { MODE-3
MODE-4 }]

NATIONAL関数の変換モードを指定します。

NATIONAL関数の変換モードには、第1オペランドに従来(V50L10以前)どおりの変換を行う(MODE1)か、視覚的に近い変換を行う(MODE2)かを指定します。本指定を省略した場合、“MODE1”が指定されたものとみなします。

翻訳オプションRCS(UTF16)が有効な場合、第2オペランドにUNIX系システムに近い変換を行うMODE3か、Windows系システムに近い変換を行うMODE4かを指定します。本指定を省略した場合、“MODE4”が指定されたものとみなします。



本指定に誤りがある場合、省略したものとみなします。翻訳オプションがRCS(UTF16)以外の場合に“MODE3”および“MODE4”を指定した場合も誤りとしてみなします。“MODE3”および“MODE4”を省略した場合の動作はプラットフォームにより異なります。

MODE1とMODE2の違いを以下に示します。

表5.2 翻訳オプションRCS(SJIS)が有効な場合

指定	変換方法
MODE1	「一」(0xB0)を「-」(0x815C)に変換します。「`」(0x60)を「'」(0x8165)に変換します。
MODE2	「一」(0xB0)を「ー」(0x815B)に変換します。「`」(0x60)を「`」(0x814D)に変換します。

表5.3 翻訳オプションRCS(UTF16)が有効な場合

指定	変換方法
MODE1	「一」(0xB0)を「-」(0x2015)に変換します。「`」(0x60)を「'」(0x2018)に変換します。
MODE2	「一」(0xB0)を「ー」(0x30FC)に変換します。「`」(0x60)を「`」(0xFF40)に変換します。

MODE3とMODE4の違いを以下に示します。

表5.4 翻訳オプションRCS(UTF16)が有効な場合

指定	変換方法
MODE3	「-」(0x2D)を「-」(0x2212)に変換します。「~」(0x7E)を「~」(0x301C)に変換します。
MODE4	「-」(0x2D)を「ー」(0xFF0D)に変換します。「~」(0x7E)を「~」(0xFF5E)に変換します。

5.4.1.31 @CBR_JOBDATE(任意の日付を取得)

@CBR_JOBDATE=年.月.日

ACCEPT文 (FROM指定にDATEを指定) または組込み関数CURRENT-DATEで任意の日付を取得する場合に指定します。
年.月.日は以下の形式で指定します。

- 年:(00-99)または(1900-2099)
- 月:(01-12)
- 日:(01-31)

西暦1900年代なら、“年”の値は西暦年の下2けた、または4けたの西暦年です。
西暦2000年代なら、“年”の値は4けたの西暦年です。

5.4.1.32 @CBR_JUSTINTIME_DEBUG (異常終了時にデバグまたは診断機能を使って調査を行う指定)

$$@CBR\_JUSTINTIME\_DEBUG = \left\{ \begin{array}{l} \left[\left\{ \begin{array}{l} APLERR \\ CBLERR \\ \underline{ALLERR} \end{array} \right\} \right], \left\{ \begin{array}{l} \text{[接続先/]TEST [起動パラメタ]} \\ \underline{SNAP} \text{ [起動パラメタ]} \end{array} \right\} \\ NO \end{array} \right\}$$

異常終了時の調査で、調査を行う対象事象と調査手段を指定します。

本環境変数を指定しない場合、“@CBR_JUSTINTIME_DEBUG=ALLERR,SNAP”が指定されたものと見なし、標準で診断機能が動作します。

APLERR

アプリケーションエラー発生時に調査を行う。

CBLERR

Uレベルの実行時メッセージ出力時に調査を行う。

ALLERR

アプリケーションエラー発生時とUレベルの実行時メッセージ出力時に調査を行う。

NO

アプリケーションエラー発生時と実行時メッセージ出力時に調査を行わない。

TEST

調査手段にデバグを使う。デバグの起動パラメタについては、“[表20.1 起動パラメタ](#)”を参照してください。

接続先を指定した場合は、リモートデバグを使用します。リモートデバグの起動パラメタについては、“[表20.7 起動パラメタ](#)”を参照してください。

接続先には、クライアント側のリモートデバグコネクタのポート番号と動作しているコンピュータを以下の形式で指定します。

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

IPアドレスは、IPv4またはIPv6の形式で指定します。

ポート番号は、1024から65535の範囲の数字を指定します。ポート番号を省略した場合は、59999が指定されたと見なされます。

IPv6形式の場合は、スコープ付きアドレスを指定することができます。スコープ付きアドレスを指定する場合、アドレスの後にスコープ識別子を指定します。

なお、IPv6形式のIPアドレスにポート番号を指定する場合は、アドレス部分の中括弧([])で囲んでください。



例

[IPv4アドレス (192.168.0.1) およびポート番号 (2000) での指定]

@CBR_JUSTINTIME_DEBUG=ALLERR, 192.168.0.1:2000/TEST

[IPv6アドレス (fe80::1:23:456:789a) およびポート番号 (2000) での指定]

@CBR_JUSTINTIME_DEBUG=ALLERR, [fe80::1:23:456:789a]:2000/TEST

[IPv6アドレス (fe80::1:23:456:789a) およびスコープ識別子 (%eth0) での指定]

@CBR_JUSTINTIME_DEBUG=ALLERR, fe80::1:23:456:789a%eth0/TEST

[IPv6アドレス (fe80::1:23:456:789a)、スコープ識別子 (%5) およびポート番号 (2000) での指定]

@CBR_JUSTINTIME_DEBUG=ALLERR, [fe80::1:23:456:789a%5]:2000/TEST

[ホスト名 (client-1) およびポート番号 (2000) での指定]

@CBR_JUSTINTIME_DEBUG=ALLERR, client-1:2000/TEST

[ポート番号省略の指定 (IPv4アドレス)]

@CBR_JUSTINTIME_DEBUG=ALLERR, 192.168.0.1/TEST

[ポート番号省略の指定 (IPv6アドレス)]

@CBR_JUSTINTIME_DEBUG=ALLERR, fe80::1:23:456:789a/TEST

SNAP

調査手段に診断機能を使う。診断機能の起動パラメタについては、“[表19.4 起動パラメタ](#)”を参照してください。



注意

- デバッガなどでデバッグしている場合は無効です。

IPv6への対応仕様については、“[表5.1 デバッガのIPv6対応](#)”を参照してください。

5.4.1.33 @CBR_MESSAGE (実行時メッセージの出力先の指定)

$$@CBR\_MESSAGE = \left\{ \begin{array}{l} \text{CBLERROUT} \\ \text{EVENTLOG}[(\text{コンピュータ名})] \\ \text{ALL}[(\text{コンピュータ名})] \end{array} \right\}$$

メッセージの出力先を指定します。

出力先を省略した場合、CBLERROUTの出力先へ出力します。

CBLERROUT

メッセージボックス、システムのコンソール(コマンドプロンプト)またはファイルへ出力します。

- @MessOutFileを指定した場合は、ファイルへ出力します。
- @CBR_CONSOLEを指定した場合は、指定した種別のコンソールへ出力します。
- @MessOutFileと@CBR_CONSOLEを同時に指定した場合、@MessOutFileが有効になり、@CBR_CONSOLEは無効になります。

EVENTLOG

イベントログへ出力します。

コンピュータ名には、イベントログの出力先コンピュータ名を指定します。ネットワーク上の他のコンピュータ名を指定することができます。コンピュータ名を省略した場合は、プログラムを実行しているコンピュータに出力します。

ALL

CBLERROUTの出力先とイベントログの両方へ出力します。

コンピュータ名には、イベントログの出力先コンピュータ名を指定します。ネットワーク上の他のコンピュータ名を指定することができます。コンピュータ名を省略した場合は、プログラムを実行しているコンピュータに出力します。

以下の場合、指定したコンピュータへの出力に失敗した旨のメッセージを、プログラムを実行しているコンピュータのイベントログへ出し、続いて、出力対象のメッセージを出力します。

- ・ ネットワーク上に実在しないコンピュータ名を指定した場合
- ・ 指定したコンピュータで32ビットWindowsが動作していない場合



注意

イベントログを出力するコンピュータには、必ず本製品(COBOLまたはCOBOLランタイムシステム)をインストールしてください。インストールしていない場合、以下のようなメッセージがイベントログに出力されます。

ソース (NetCOBOL) 内のイベントID (nnnn) に関する説明が見つかりません。 次の挿入文字列が含まれています。: \$1, \$2...\$n

nnnn:メッセージ番号

\$1~\$n:挿入文字列

5.4.1.34 @CBR_MESS_LEVEL_CONSOLE(実行時メッセージの重大度指定)

@CBR_MESS_LEVEL_CONSOLE = $\left\{ \begin{array}{c} \text{NO} \\ \text{I} \\ \text{W} \\ \text{E} \\ \text{U} \end{array} \right\}$

ランタイムシステムが出力する実行時メッセージの出力抑止や、実行時メッセージを出力する重大度コードを指定します。
@CBR_MESSAGEの宛先CBLERROUTが有効でない場合、本環境変数の値は意味を持ちません。

@CBR_MESS_LEVEL_CONSOLE に指定できるパラメタは以下のとおりです。

NO

実行時メッセージを出力しません。

I

重大度コードがI以上のメッセージを出力します。

W

重大度コードがW以上のメッセージを出力します。

E

重大度コードがE以上のメッセージを出力します。

U

重大度コードがUのメッセージを出力します。



例

重大度コードがI以上の実行時メッセージを出力する

```
@CBR_MESS_LEVEL_CONSOLE=I
```



注意

- @CBR_MESS_LEVEL_CONSOLE の指定がない場合、または上記以外のパラメタが右辺に指定された場合、重大度コードがI以上のメッセージが出力されます。(I指定と同意)
- @CBR_MESS_LEVEL_CONSOLE の指定は、環境変数情報@MessOutFileで指定したファイルに出力される実行時メッセージにも有効となります。
- @CBR_MESS_LEVEL_CONSOLE にNOを指定した場合、オペレータの応答が不要な実行時メッセージはすべて出力されません。実行時エラーの原因特定が困難になりますので、目的を十分理解した上で指定してください。
- @CBR_MESS_LEVEL_CONSOLEにNOを指定した場合でも、オペレータの応答が必要な実行時メッセージは常に出力されます。
- @NoMessage=YESを同時に指定している場合、@CBR_MESS_LEVEL_CONSOLEに指定した重大度が有効となります。

5.4.1.35 @CBR_MESS_LEVEL_EVENTLOG(実行時メッセージの重大度指定)

@CBR_MESS_LEVEL_EVENTLOG = $\left\{ \begin{array}{c} \text{NO} \\ \text{I} \\ \text{W} \\ \text{E} \\ \text{U} \end{array} \right\}$

ランタイムシステムが出力する実行時メッセージのイベントログへの出力抑止や、実行時メッセージを出力する重大度コードを指定します。

@CBR_MESS_LEVEL_EVENTLOG に指定できるパラメタは以下のとおりです。

NO

実行時メッセージを出力しません。

I

重大度コードがI以上のメッセージを出力します。

W

重大度コードがW以上のメッセージを出力します。

E

重大度コードがE以上のメッセージを出力します。

U

重大度コードがUのメッセージを出力します。



例

重大度コードがI以上の実行時メッセージをイベントログに出力する

```
@CBR_MESS_LEVEL_EVENTLOG=I
```



注意

- @CBR_MESS_LEVEL_EVENTLOG の指定がない場合、または上記以外のパラメタが右辺に指定された場合、重大度コードがI以上のメッセージが出力されます。(I指定と同意)
- @CBR_MESS_LEVEL_EVENTLOG にNOを指定した場合、すべての実行時メッセージが出力されません。実行時エラーの原因特定が困難になりますので、目的を十分理解した上で指定してください。
- @NoMessage=YESを同時に指定している場合、@CBR_MESS_EVENTLOGに指定した重大度が有効となります。

5.4.1.36 @CBR_MEMORY_CHECK(メモリチェック機能を使って検査を行う指定)

```
@CBR_MEMORY_CHECK = MODE1
```

アプリケーション実行時、ランタイムシステム領域を、メモリチェック機能を使って検査します。メモリチェック機能の使い方については、“[19.5 メモリチェック機能の使い方](#)”を参照してください。

5.4.1.37 @CBR_OverlayPrintOffset(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 = { VALID
                             INVALID }
```

FORMAT句なし印刷ファイルで、I制御レコードに指定されたとじしろ方向(BIND)、とじしろ幅(WIDTH)および印刷原点位置(OFFSET)機能をフォームオーバーレイに対しても有効にする(VVALID)か、無効にする(INVALID)かを指定します。本指定の省略時は、“INVALID”が指定されたものとみなされます。なお、本指定の有効範囲は実行単位内です。ファイル単位に有効な指定については、“[8.1.9 印刷情報ファイル](#)”を参照してください。I制御レコードについては、“[8.1.5 I制御レコード/S制御レコード](#)”を参照してください。



参考

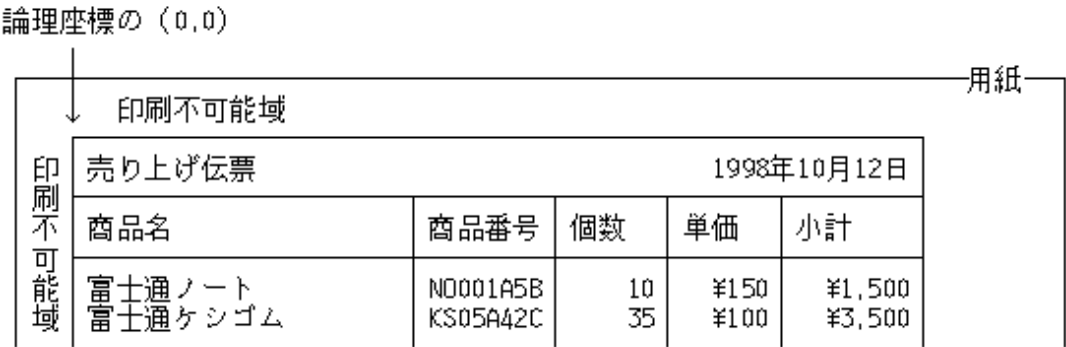
V40L20以前のバージョンでは、I制御レコードに指定されたとじしろ方向(BIND)、とじしろ幅(WIDTH)および印刷原点位置(OFFSET)機能は、行レコード(文字データ)にだけ有効でした(本実行環境情報のINVALID指定時と同じ)。

これに対し、本実行環境情報にVALIDを指定することで、I制御レコードの同機能を行レコードおよびフォームオーバーレイの両方に対して有効にすることができます。

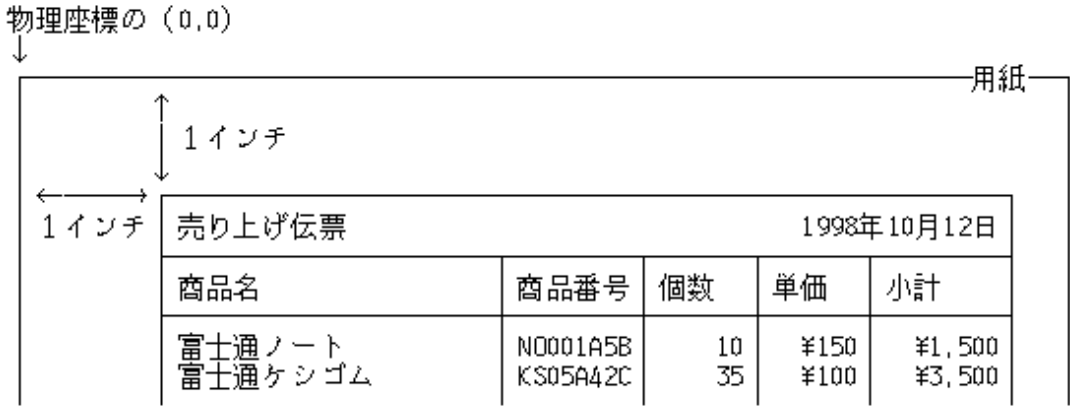
以下に、原点オフセットを設けない通常の印刷結果と、上方向および左方向にそれぞれ1インチの原点オフセットを設けた場合の印刷結果を例にとり、VALID/INVALID指定時(または省略時)の印刷結果の相違について図示します。

以下の図では、罫線=オーバーレイ、文字=行レコードを示しています。

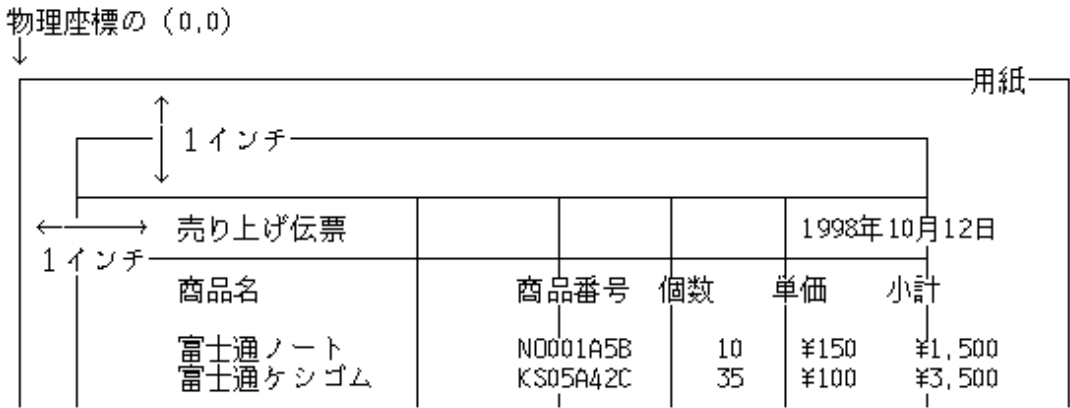
- 印刷原点位置を指定しない場合の印刷結果



- X,Y座標にそれぞれ1インチ原点オフセットを設けた場合の印刷結果
 - VALID指定の場合



- INVALID指定の場合



5.4.1.38 @CBR_OverlayPrintSPEC(フォームオーバーレイ印刷方法の指定)

@CBR_OverlayPrintSPEC = $\left\{ \begin{array}{l} \text{COBOL} \\ \text{PRINTMONITOR} \end{array} \right\}$

FORMAT句なし印刷ファイルで、I/S制御レコードを使用したフォームオーバーレイの印刷を行う場合の実現方法を指定します。COBOLランタイムシステムによるオーバーレイ印刷を行う場合”COBOL”を指定します。PrintWalker/OVLオプションが提供する高速オーバーレイ機能を利用してオーバーレイ印刷を行う場合”PRINTMONITOR”を指定します。本指定が省略された場合、”COBOL”が指定されたものとみなされます。なお、本指定の有効範囲は実行単位内です。ファイル単位に有効な指定については、“[8.1.9 印刷情報ファイル](#)”を参照してください。I/S制御レコードについては、“[8.1.5 I制御レコード/S制御レコード](#)”を参照してください。

なお、FORMAT句付き印刷ファイルおよび表示ファイルを使用した高速オーバーレイ印刷については、“MeFtのオンラインマニュアル”を参照してください。

注意

- PrintWalker/OVLオプションは、富士通のVSPシリーズの特定のプリンタにだけ適用される装置オプションです。したがって、PrintWalker/OVLオプションの高速オーバーレイ機能は、すべてのプリンタ装置で利用できるわけではありません。なお、高速オーバーレイ機能に関する詳細情報については、“PrintWalker/OVLオプションのオンラインマニュアル”を参照してください。
- 利用できるフォームオーバーレイの形式は、FORMで作成したKOL2およびKOL5形式です(KOL6形式は使用できません)。
- オーバーレイ定義体格納フォルダ指定(環境変数情報FOVLDIR)、オーバーレイ定義体形式(環境変数情報FOVLTYPE)およびオーバーレイ定義体拡張子(環境変数情報OVD_SUFFIX)の指定方法は、従来のCOBOLランタイムシステムによるオーバーレイ印刷方式および高速オーバーレイによる印刷方式とも同様です。ただし、高速オーバーレイ指定時にオーバーレイ定義体格納フォルダ指定(環境変数情報FOVLDIR)を省略した場合は、PrintWalker/OVLオプションのデフォルトフォルダが指定されたものとみなされます。

FOVLDIR=¥¥DATASERVER¥FOVL	←C/S 運用時の例
FOVLDIR=C:¥FOVL	←スタンドアローン運用時の例
FOVLDIR=省略	←PrintWalker/OVL オプションが定めるデフォルトフォルダ指定の例
FOVLTYPE=KOL5	
OVD_SUFFIX=None	

- COBOLアプリケーションから縮小印刷指定(I制御レコードのPRT-FORM:印刷形式フィールド)に”LP”(ラインプリンタモード)を指定しても有効になりません。
- COBOLアプリケーションから印刷原点指定(I制御レコードのWIDTH:とじしろ幅フィールドおよびOFFSET:印刷開始原点位置フィールド)を指定することはできません。COBOLアプリケーションから印刷原点指定を行った場合、その印刷結果は保証されません。印刷原点指定を行いたい場合は、COBOLからの印刷原点指定は行わず、プリンタドライバ側で印刷原点を設定してください。印刷原点指定を行わない場合、I制御レコードのWIDTH:とじしろ幅フィールドおよびOFFSET:印刷開始原点位置フィールドに9999または空白を指定してください。
- COBOLアプリケーションから両面印刷時の印刷開始面を裏面に位置付けた場合、本来白紙となるべき表面にオーバーレイパターンが印刷されることがあります。印刷開始面が裏面に位置付く場合とは、I制御レコードのSIDE:印刷面フィールドに”B”(両面印刷)を指定し、かつ、POSIT(印刷開始面)に”B”(裏面位置付け)を指定した場合です。

5.4.1.39 @CBR_PrinterANK_Size(ANK文字サイズの指定)

@CBR_PrinterANK_Size = $\left\{ \begin{array}{l} \text{TYPE-M} \\ \text{TYPE-PC} \\ \text{TYPE-G} \end{array} \right\}$

印刷ファイルで、デフォルトのANK文字サイズ(CHARACTER TYPE句およびPRINTING POSITION句を指定していない英数字項目の文字サイズ)を変更する場合に指定します。なお、本指定の有効範囲は実行単位内です。

TYPE-M

デフォルトANK文字サイズをOSIV系システムでのサイズに近づけ、9.6ポイントで印字します。

TYPE-PC

デフォルトANK文字サイズをPCで標準的なサイズとし、10.5ポイントで印字します。

TYPE-G

デフォルトANK文字サイズをFMGシリーズでのサイズに近づけ、10.8ポイントで印字します。

省略された場合、7.0ポイントで印字します。



注意

固定サイズのフォント(ラスタフォント)が印刷用フォントとして選択されている場合、本指定は有効とならないことがあります。この場合、そのフォントがサポートするサイズのうち、一番近いサイズで印字されます。一般的な例として、シリアルプリンタに搭載されているデバイスフォントは、通常、10.5ポの固定サイズでしか印字できません。このようなフォントが選択された場合、本実行環境情報の指定にかかわらず、つねに10.5ポで印字されます。

5.4.1.40 @CBR_PrintFontTable (印刷ファイルで使用するフォントテーブルの指定)

@CBR_PrintFontTable = フォントテーブル名

印刷ファイルで使用するフォントテーブルのファイル名を絶対パスで指定します。

ここで指定されたフォントテーブルは、同じ実行環境で動作するすべての印刷ファイルに対して有効になります。ファイルごとのフォントテーブルの指定については、“[5.4.1.71 ファイル識別名 \(プログラムで使用するプリンタ情報ファイルおよび各種パラメタの指定\)](#)”、“[5.4.1.72 ファイル識別名 \(プログラムで使用するプリンタおよび各種パラメタの指定\)](#)”を参照してください。

また、フォントテーブルの詳細については、“[8.1.2 印字文字](#)”の“印字文字の書体”および“印字文字のスタイル”、“[8.1.14 フォントテーブル](#)”を参照してください。

5.4.1.41 @CBR_PrintInfoFile (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 = 印刷情報ファイル名

FORMAT句なし印刷情報ファイルで、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名前を絶対パスで指定します。

印刷情報ファイルの詳細については、“[8.1.9 印刷情報ファイル](#)”を参照してください。



例

COBOL ソースプログラムのASSIGN句の記述

ASSIGN TO PRINTER

実際に使用する印刷情報ファイル名とそのパス


```
C:¥PRINT. INF
```

環境変数情報

```
@CBR_PrintInfoFile=C:¥PRINT. INF
```

5.4.1.42 @CBR_PrintTextPosition(文字配置座標の計算方法の指定)

$$@CBR\_PrintTextPosition = \left\{ \begin{array}{c} TYPE1 \\ \underline{TYPE2} \end{array} \right\}$$

FORMAT句なし印刷ファイルで、印字する文字を配置する座標(x,y)の計算方法を指定します。なお、本指定の有効範囲は実行単位内です。

文字配置の座標の補正を行わない(TYPE1)か、行う(TYPE2)かを指定します。

TYPE1

印刷するプリンタ装置の解像度をアプリケーションで指定した行間隔(LPI)や文字間隔(CPI)で除算し、余りは切り捨てた値で文字と文字の間隔を決定し配置します。

TYPE2

TYPE1指定時と同様、印刷するプリンタ装置の解像度をアプリケーションで指定した行間隔(LPI)や文字間隔(CPI)で除算した値で文字と文字の間隔を決定し配置します。しかし、割り切れないような解像度を持つプリンタ装置に出力する場合、1インチ単位内で座標の補正処理を行います。



注意

V40L20以前のバージョンでは、本指定の省略時の扱いはTYPE1が指定されたものとみなしていました。しかし、V50以降のバージョンでは省略時の扱いはTYPE2が指定されたものとみなします。これにより、不都合(フォームオーバーレイと文字の重畳印刷時のズレなど)が出る場合は、TYPE1を明に指定してください。

5.4.1.43 @CBR_PSFILE_xxx(表示ファイルから使用する接続製品名(宛先ごとの指定))

@CBR_PSFILE_xxx = 接続製品名

xxxには、表示ファイルのSYMBOLIC DESTINATION句に記述した、以下の宛先名のどれかを指定してください。

- DSP
- PRT
- ACM
- APL

また、接続製品名には、実際に使用する接続製品を示す文字列を指定します。指定する文字列を以下に示します。

表5.5 接続製品名の指定形式

SYMBOLIC DESTINATION句の指定	使用する製品	接続製品名を示す文字列
DSP または SYMBOLIC DESTINATION句省略	MeFt	MEFT または 省略
	MeFt/NET	MEFTNET
ACM	SymfoWARE7000 for Windows NT または ACM Server for Windows NT	ACM
PRT	MeFt	MEFT または 省略
	MeFt/NET	MEFTNET



- ・ファイル識別名に接続製品名の指定がある表示ファイルでは、ここで指定する接続製品名は無効になります。[参照]“5.4.1.70 ファイル識別名 (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)”
- ・“ACM”を指定し、かつ接続製品名の指定を省略した場合、ファイル識別名に接続製品名の指定がない表示ファイルでは、実行時にエラーメッセージが出力されます。
- ・SYMBOLIC DESTINATION句を指定していない場合には、必ず“DSP”を指定してください。

5.4.1.44 @CBR_SCR_KEYDEFFILE (スクリーン操作のキー定義ファイルの指定)

@CBR_SCR_KEYDEFFILE = キー定義ファイル

スクリーン操作機能で、ファンクションキーの利用者定義を使用する場合、ファンクションキーの利用者定義を定義したファイル名を指定します。

ファイル名には、絶対パスと相対パスが指定できます。相対パスが指定された場合は、カレントフォルダからの相対パスになります。

5.4.1.45 @CBR_SCREEN_POSITION (スクリーン画面の表示位置の指定)

@CBR_SCREEN_POSITION = $\left\{ \begin{array}{l} \text{CENTER} \\ \text{SYSTEM} \\ (x, y) \end{array} \right\}$

スクリーン操作機能で表示されるスクリーン画面の表示位置を指定します。

省略された場合、“CENTER”が指定されたものとみなされます。

CENTER

画面の中央にスクリーン画面を表示します。

SYSTEM

システムのデフォルトの表示位置に表示します。

(x,y)

指定された位置から画面を表示します。xおよびyには、画面を表示する位置(数)を指定してください。

5.4.1.46 @CBR_SYSERR_EXTEND(SYSERR出力情報の拡張の指定)

@CBR_SYSERR_EXTEND = YES

DISPLAY文でSYSERRに出力するとき、プロセスID、スレッドIDの情報を付加します。

5.4.1.47 @CBR_TextAlign(文字行内配置時の上端/下端合わせの指定)

@CBR_TextAlign = $\left\{ \begin{array}{c} \text{TOP} \\ \text{BOTTOM} \end{array} \right\}$

FORMAT句なし印刷ファイルで、行内に配置する印字文字の位置を指定します。なお、本指定の有効範囲は実行単位内です。

印字する文字を文字セルの左上端を基準点として行内に上端合わせで配置する(TOP)か、左下端を基準点として行内に下端合わせで配置する(BOTTOM)かを指定します。

5.4.1.48 @CBR_TRACE_FILE(トレース情報の出力ファイルの指定)

@CBR_TRACE_FILE = ファイル名

TRACE機能を使用する場合、トレース情報の出力先となるファイルの名前を指定します。



参照

.....
“19.3 TRACE機能の使い方”
.....

ファイル名には、絶対パスと相対パスを指定できます。相対パスが指定された場合は、カレントフォルダからの相対パスになります。

ファイル名に拡張子が指定されていた場合、その拡張子は無視され、TRCおよびTROに置き換わります。

なお、この環境変数情報を指定しない場合は、実行可能ファイルの名前に、拡張子TRCおよびTROを付加したファイルに格納されます。

5.4.1.49 @CBR_TRACE_PROCESS_MODE(TRACEファイルのプロセス毎の出力指定)

@CBR_TRACE_PROCESS_MODE = MULTI

同じ名前の実行可能プログラムを複数同時に実行した場合、プロセス毎に固有のファイル名を作成することを指定します。

本環境変数が指定された場合のファイル名は、以下のようになります。

実行形式名-プロセスID_日付(YYYYMMDD)_時間(HHMMSS). TRC or TRO

5.4.1.50 @CBR_TRAILING_BLANK_RECORD (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 = $\left\{ \begin{array}{c} \text{REMOVE} \\ \text{VALID} \end{array} \right\}$

行順ファイルのWRITE文の実行時に、レコード内の後置空白を取り除く(REMOVE)か、有効にする(VALID)かを指定します。
本指定を省略した場合、“VALID”が指定されたものとみなします。

注意

削除される空白はコード系に依存し、以下のようになります。

- ・コード系がUnicodeで字類が英数字の場合は、半角空白が削除されます。
 - ・コード系がUnicodeで字類が日本語の場合は、全角空白が削除されます。
 - ・コード系がUnicode以外の場合は、半角空白および全角空白が削除されます。
-

5.4.1.51 @CnslBufLine (コンソールウィンドウのバッファ数の指定)

@CnslBufLine = $\left\{ \begin{array}{c} \text{バッファ行数} \\ \underline{100} \end{array} \right\}$

小入出力機能で使用するコンソールウィンドウのバッファの数を指定します。バッファ行数は、1～9999の数字で指定します。

ただし、(コンソールウィンドウの桁数+1)×(バッファ行数)が65000を超える場合は、バッファ行数は縮小されます。たとえば、桁数が80の場合は802まで、桁数が91の場合は706までのバッファ行数が指定できます。

注意

システムのコンソールを使用する場合は無効です。[参照]“5.4.1.10 @CBR_CONSOLE (コンソールウィンドウの種別の指定)”

5.4.1.52 @CnslFont (コンソールウィンドウのフォントの指定)

@CnslFont = (フォント名, フォントサイズ)

小入出力機能で使用するコンソールウィンドウのフォントを指定します。

注意

- ・システムのコンソールを使用する場合は無効です。[参照]“5.4.1.10 @CBR_CONSOLE (コンソールウィンドウの種別の指定)”
- ・フォント名には、固定幅フォントを使用してください。“MS P明朝”や“MS Pゴシック”など、プロポーショナルフォントを選択した場合の動作は保証していません。この場合、横幅の狭い文字は、左端に寄せられます。

- ・ サロゲートペア文字を使用する場合は、サポートされているフォント名を指定してください。
- ・ 合成文字の入力および表示はサポートしていません。

5.4.1.53 @CnslWinSize(コンソールウィンドウの大きさの指定)

```
@CnslWinSize = { (桁数,行数)
                  (80,24) }
```

小入出力機能で使用するコンソールウィンドウの大きさを指定します。桁数および行数は、それぞれ1～999の数字で指定します。ただし、ウィンドウの大きさの最小値および最大値は、システムの最小値および最大値となるため、それより小さい値または大きい値が指定された場合には、システムの最小値および最大値の大きさとなります。



注意

システムのコンソールを使用する場合は無効です。[参照]“5.4.1.10 @CBR_CONSOLE(コンソールウィンドウの種別の指定)”

5.4.1.54 @DefaultFCB_Name(デフォルトFCB名の指定)

```
@DefaultFCB_Name = FCBxxxx
```

デフォルトで使用するFCBの名前を指定します。ここで指定するFCB名(FCBxxxx)は、I制御レコードからFCBを動的に変更する場合と同様、FCB制御文をあらかじめ定義しておく必要があります。

以下に、FCB制御文を複数定義しておき、行間隔=6LPI、ページ内行数=66行、ページ内印刷開始行位置=1行目、用紙縦長=11インチと定義されているFCBをデフォルトFCBに設定する例を示します。



例

```
@DefaultFCB_Name=FCB6LPI
FCB6LPI=LPI((6, 66)), CH1(1), SIZE(110)
FCB8LPI=LPI((8, 88)), CH1(4), SIZE(110)
FCBA4LD=LPI((12)), CH1(1), FORM(A4, LAND)
```

5.4.1.55 @ExitSessionMSG(COBOLアプリケーション起動中のWindows終了指定)

```
@ExitSessionMSG = { ON
                     OFF }
```

COBOLアプリケーションの起動中にWindowsシステムを終了しようとした場合、COBOLが動作中である旨の警告メッセージを出力し、Windowsシステムを終了させない(ON)か、警告メッセージを出力しないでシステムと同時にCOBOLプログラムを終了する(OFF)かを指定します。



注意

OFFを指定して、Windowsシステムの終了を許可した場合、COBOLプログラムが、ファイルなどの資源に対して入出力を行っている最中であっても、システムそのものが終了することになります。この結果、入出力中であった資源が破壊されることがありますので、注意してください。

5.4.1.56 @GOPT(実行時オプションの指定)

@GOPT = 実行時オプションの並び

実行時オプションの並びには、実行時オプションを指定します。
実行時オプションの指定形式については、“5.8 実行時オプションの指定形式”を参照してください。



例

@GOPT=r20 c20

5.4.1.57 @IconDLL(アイコンリソースのDLL名の指定)

@IconDLL = アイコンリソースのDLL名

アイコンとして表示したいアイコンリソースを含むDLL名を指定します。

指定したDLL中のアイコンリソースを表示させるためには、さらに環境変数情報@IconNameにアイコンリソース名を指定する必要があります。

アイコンリソースのDLL名には、絶対パスまたは相対パスが使用できます。相対パスで指定した場合、以下の検索順序に従ってDLLが検索されます。

1. 実行可能ファイルの存在するフォルダ
2. カレントフォルダ
3. Windowsのシステムフォルダ
4. Windowsのフォルダ
5. 環境変数PATHに指定されているフォルダ

@IconDLLが指定されていない場合は、主プログラムの実行可能ファイルの中から、@IconNameに指定されたアイコンリソースを検索します。



参照

“5.4.1.58 @IconName(アイコンリソースの識別名の指定)”



注意

マルチスレッドモードでの動作時には、翻訳オプションMAINを指定して翻訳されていない場合、アイコンは表示されません。

5.4.1.58 @IconName(アイコンリソースの識別名の指定)

@IconName = アイコンリソースの識別名

アイコンを変更したい場合に、アイコンリソースの識別名を指定します。

表示するアイコンリソースは、あらかじめアプリケーションにリンクしておくか、アイコンリソースを含むDLLを作成しておく必要があります。

DLL中のアイコンリソースを表示させる場合は、環境変数情報@IconDLLにアイコンリソースを含むDLL名を指定する必要があります。
[参照]“5.4.1.57 @IconDLL(アイコンリソースのDLL名の指定)”

プロジェクト管理機能を使用して作成したアプリケーションには、COBOLのアイコンリソースがリンクされています。このアプリケーションの実行時に、本環境変数情報が指定されていない場合は、COBOLのアイコンが表示されます。プロジェクト管理機能を使用していない場合は、システムのデフォルトのアイコンが表示されます。



マルチスレッドモードでの動作時には、翻訳オプションMAINを指定して翻訳されていない場合、アイコンは表示されません。

5.4.1.59 @MessOutFile(メッセージを出力するファイルの指定)

@MessOutFile = ファイル名

実行時メッセージおよびUPON SYSERR指定のDISPLAY文による出力を、指定されたファイルに出力します。ファイル名を指定した場合、メッセージボックスは画面に表示されません。なお、Unicodeでの扱いについては、“[実行](#)”を参照してください。

各種アプリケーションサーバ配下で動作するプログラムの実行時メッセージは、@MessOutFileを必ず指定して出力させてください。詳細は、“[第23章 サーバ・タイプのアプリケーション](#)”を参照してください。



- ・ ファイル名には、絶対パスと相対パスを指定できます。相対パスが指定された場合は、カレントフォルダからの相対パスになります。
- ・ 同一のファイルが存在する場合は、そのファイルに情報を追加します。
- ・ 入出力機能の出力ファイルとして指定したファイルを@MessOutFileに指定した場合、出力されるファイルの内容は保証されません。
- ・ ファイルに出力できない場合、実行時メッセージはメッセージボックスに出力されます。
- ・ ファイルの最大サイズは、システムの制限までです。

5.4.1.60 @MGPRM(OSIV系形式の実行時パラメタの指定)

@MGPRM = "実行時パラメタの文字列"

プログラムに渡す文字列を二重引用符(")で囲んで指定します。指定した文字列は、OSIV系システムでプログラムを実行させたときと同様の形式で、プログラムに渡されます。

実行時パラメタの文字列は、最大で100バイトまで指定できます。

OSIV系形式の実行時パラメタを受け取る方法については、“[付録F OSIV系システムとの機能比較](#)”を参照してください。



例

@MGPRM="ABCDE"

5.4.1.61 @NoMessage(実行時メッセージおよびSYSERRの出力抑止指定)

@NoMessage = YES

以下の出力を抑止します。

- Uレベル以外の実行時メッセージ
- オペレータの応答を必要としない実行時メッセージ
- DISPLAY文のSYSERR



参考

ウィンドウを閉じるときのメッセージ表示を抑止する場合には、環境変数情報@WinCloseMsgを使用してください。[参照]“[5.4.1.68 @WinCloseMsg\(ウィンドウを閉じるときのメッセージ表示の指定\)](#)”

5.4.1.62 @ODBC_Inf(ODBC情報ファイルの指定)

@ODBC_Inf = ODBC情報ファイル名

ランタイムシステムがODBCを使用するために必要な、ODBC情報ファイル名を指定します。

ODBC情報ファイルに指定された情報は、主にクライアントとサーバ間のコネクション(CONNECT文などで指定する)を確立するために使用されます。

5.4.1.63 @PrinterFontName(印刷ファイルで使用するフォントの指定)

@PrinterFontName = (明朝体フォント名, ゴシック体フォント名)

印刷ファイルで使用するフォントのフォントフェイス名を指定します。



注意

- フォントフェイス名は、32バイト以内の英数字または日本語文字で指定してください。
- フォントフェイス名の前後に、全角または半角の空白が含まれる場合、これらの空白もフォントフェイス名の一部とみなされます。
- フォントフェイス名に半角のコンマ(,)が含まれているフォント名は指定できません。

- ・ フォントフェイス名に指定する書体名は、以下のどちらかの方法で求めてください。

- ー 実行用初期化ファイルに指定する方法

- 実行環境設定ツールの[環境設定]→[フォント指定]→[プリンタフォント]で、プリンタフォントの指定ダイアログを開きます。
- [明朝]、[ゴシック]の各ダイアログから該当フォントを選択します。

- ー 環境変数@PrinterFontNameに直接指定する方法

- コントロールパネルの“フォント”を選択します。
- 表示されるフォントの一覧から該当フォントを選択し、ダブルクリックします。
- 表示ウィンドウの「書体名: ____」のアンダーライン箇所 to 書体名が表示されます。



参考

明朝体フォントの検索順序は、(1)本環境変数情報で指定した明朝体フォント、(2)明朝、(3)MS 明朝となります。

ゴシック体フォントの検索順序は、(1)本環境変数情報で指定したゴシック体フォント、(2)ゴシック、(3)MS ゴシックとなります。

検索の結果、上記のどのフォントもシステムにインストールされていない場合には、アプリケーション実行時にJMP0320I 'FONT'エラーとなります。

5.4.1.64 @PRN_FormName_xxx(用紙名の指定)

@PRN_FormName_xxx = 用紙名

FORMAT句なし印刷ファイルで、I制御レコードのSIZE(用紙サイズ)フィールドに指定した任意の3文字以内の文字列を本環境変数情報名のxxx部分に置き換え、本環境変数情報名に対応して用紙名を対応付けて指定します。

各プリンタの[プロパティ]ダイアログの[デバイスの設定]シートの“給紙方法と用紙の割り当て”で、各給紙方法の右側にあるリストボックスに表示される用紙名のことをいいます。



例

- ・ 用紙名は、プリンタまたはプリンタドライバによりサポート範囲が異なります。プリントマネージャまたは“プリンタ”の“プロパティ”で、目的のプリンタドライバがサポートしている用紙サイズを確認するか、プリンタの取扱い説明書を参照してください。
- ・ 作成した用紙名を指定する場合、用紙定義後、システムを再起動する必要があります。システムを再起動しないと、定義した用紙名が有効とならない場合がありますので注意してください。用紙の作成は、以下の手順で行います。
 - ー [スタート]メニューで“設定”を選択し、[プリンタ]フォルダを表示させます。
 - ー [プリンタ]フォルダの[ファイル]メニューから“サーバのプロパティ”を選択し、[用紙]ページで作成します。



例

I制御レコードのSIZEフィールドの指定

ABC

xxx 部分置換後の環境変数情報名

@PRN_FormName_ABC

@PRN_FormName_ABC=15x11

5.4.1.65 @ScrnFont(スクリーン操作で使用するフォントの指定)

@ScrnFont = (フォント名, フォントサイズ)

スクリーン操作で使用するフォントを指定します。

5.4.1.66 @ScrnSize(スクリーン操作の論理画面の大きさの指定)

@ScrnSize = $\left\{ \begin{array}{l} \text{(桁数, 行数)} \\ \text{(80,25)} \end{array} \right\}$

スクリーン操作機能で使用するウィンドウの論理画面の大きさを指定します。桁数および行数は、それぞれ1～999の数字で指定します。本環境変数情報で指定した大きさが、物理画面のデフォルトの大きさになります。

ただし、(桁数+1)×行数が16250より大きい場合、プログラム実行時にエラーとなります。

5.4.1.67 @ShowIcon(NetCOBOLのアイコン表示の抑止指定)

@ShowIcon = $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$

NetCOBOLのアイコンをタスクバーに表示させる(YES)か、させない(NO)かを指定します。



注意

- アイコン表示を抑止した場合、システムのタスクマネージャ上に該当アプリケーション名が出力されなくなり、該当アプリケーションを強制終了する方法がなくなります。

また、アイコン表示を抑止した場合、環境変数情報@ExitSessionMSGの指定は無効になります。この場合、COBOLプログラムの実行中にWindowsシステムを終了させると、システムの警告メッセージが出力されます。システムの警告メッセージが出力され、プログラムの強制終了を行った場合、強制終了したプログラムで行っていた入出力処理の結果は保証されません。

- マルチスレッドモードでの動作時には、翻訳オプションMAINを指定して翻訳されていない場合、アイコンは表示されません。

5.4.1.68 @WinCloseMsg(ウィンドウを閉じるときのメッセージ表示の指定)

@WinCloseMsg = $\left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \end{array} \right\}$

小入出力機能で使用するコンソールウィンドウや、スクリーン操作機能で使用するウィンドウを閉じるときに、確認のためのメッセージを表示する(ON)か、しない(OFF)かを指定します。

5.4.1.69 ファイル識別名(プログラムで使用するファイルの指定)

ファイル識別名 = ファイル名[,アクセス種別またはファイルシステム種別][,MOD][,CONCAT(ファイル名…)][,DUMMY]

ファイル識別名には、COBOLソースプログラムのASSIGN句に記述したファイル識別名を指定し、ファイル名には実際に処理を行うファイルの名前を指定します。この指定は、プログラム中のファイルと実際に処理するファイルを関連付けるために行います。

ファイル名には、絶対パスと相対パスを指定できます。相対パスが指定された場合は、カレントフォルダからの相対パスになります。

なお、プログラム中でファイル識別名を英小文字で記述した場合でも、環境変数情報は英大文字で指定してください。



例

COBOL ソースプログラムのASSIGN句の記述

ASSIGN TO OUTFILE

実際に処理を行うファイルの名前

F:¥WORK. DAT

環境変数情報

OUTFILE=F:¥WORK. DAT

アクセス種別またはファイルシステム種別には、対応する種別に応じて、以下の文字列を指定することができます。

BSAM

ファイルの高速処理を実現します。[参照]“[7.7.4 ファイルの高速処理](#)”

BTRV

Btrieveファイル进行处理します。[参照]“[7.9.1 Btrieveファイル](#)”

RDM

RDMファイル进行处理します。(SymfoWARE7000 for Windows NT)[参照]“[7.9.2 RDMファイル](#)”

EXFH

外部ファイルハンドラ进行处理します。[参照]“[7.9.4 外部ファイルハンドラ](#)”

MOD

ファイルの追加書き进行处理します。[参照]“[7.7.5 ファイル追加書き](#)”

CONCAT

ファイルの連結进行处理します。[参照]“[7.7.6 ファイルの連結](#)”

DUMMY

ダミーファイル機能を実現します。[参照]“[7.7.7 ダミーファイル](#)”

指定可能な組合せ、および注意事項については、“[7.7.8 注意事項](#)”を参照してください。

5.4.1.70 ファイル識別名(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 = [情報ファイル名][,接続製品名]

表示ファイルを利用する場合のファイル識別名の指定方法について説明します。

ファイル識別名には、COBOLソースプログラムのASSIGN句に記述したファイル識別名を指定します。プログラム中でファイル識別名を英小文字で記述した場合でも、環境変数情報のファイル識別名は英大文字で指定してください。

情報ファイル名の指定は、MeFtを使用する場合は必須となりますが、MeFt以外を使用する場合は省略できます。情報ファイルには、実際に使用するMeFtのウィンドウ情報ファイル名またはプリンタ情報ファイル名を指定します。

接続製品名には、実際に使用する接続製品を示す文字列を指定します。接続製品を示す文字列については“表5.5 接続製品名の指定形式”を参照してください。接続製品名の指定を省略した場合、環境変数情報@CBR_PSFILE_xxxに指定された接続製品名が有効になります。[参照]“5.4.1.43 @CBR_PSFILE_xxx(表示ファイルから使用する接続製品名(宛先ごとの指定))”

接続製品名の指定が省略され、かつ環境変数情報@CBR_PSFILE_xxxの指定もない場合、表示ファイルの宛先指定に応じた省略値が採用されます。省略値については、“表5.5 接続製品名の指定形式”の“省略”と記述された欄を参照してください。



例

COBOLソースプログラムのASSIGN句の記述

```
ASSIGN TO GS-PRTFILE
```

COBOLソースプログラムのSYMBOLIC DESTINATION句の記述

```
SYMBOLIC DESTINATION IS "PRT"
```

実際に使用する情報ファイルの名前

```
C:¥WORK. PRC
```

実際に使用する接続製品

```
MeFt
```

環境変数情報

```
PRTFILE=C:¥WORK. PRC, MEFT
```

5.4.1.71 ファイル識別名(プログラムで使用するプリンタ情報ファイルおよび各種パラメタの指定)

ファイル識別名 = プリンタ情報ファイル名[,FONT(フォントテーブル名)]

FORMAT句付き印刷ファイルを利用する場合のファイル識別名の指定方法について説明します。

ファイル識別名には、COBOLソースプログラムのASSIGN句に記述したファイル識別名を指定します。プログラム中でファイル識別名を英小文字で記述した場合でも、環境変数情報のファイル識別名は英大文字で指定してください。

プリンタ情報ファイルには、実際に使用するMeFtのプリンタ情報ファイル名を指定します。

フォントテーブル名には、フォントテーブルのファイル名を絶対パスで指定します。フォントテーブルには、PRINTING MODE句のFONT-nnn(書体番号)指定に対応するフォントフェース名や印字スタイルの情報を定義します。ここで指定されたフォントテーブルは、そのフ

イルに対してだけ有効なフォントテーブルとなります。すべての印刷ファイルに共通なフォントテーブル(実行環境単位で有効なフォントテーブル)の指定については、“5.4.1.40 @CBR_PrintFontTable(印刷ファイルで使用するフォントテーブルの指定)”を参照してください。

また、フォントテーブルの詳細については、“8.1.2 印字文字”の“印字文字の書体”および“印字文字のスタイル”、“8.1.14 フォントテーブル”を参照してください。

なお、実行環境単位で有効なフォントテーブル名とファイル単位で有効なフォントテーブル名が異なる場合、ファイル単位の指定が優先されます。



例

例1

COBOL ソースプログラムのASSIGN句の記述

```
ASSIGN TO PRTFILE
```

実際に使用するプリンタ情報ファイルの名前

```
C:¥WORK. WRC
```

環境変数情報

```
PRTFILE=C:¥WORK. WRC
```

例2

COBOL ソースプログラムのASSIGN句の記述

```
ASSIGN TO PRTFILE
```

実際に使用するプリンタ情報ファイルの名前

```
C:¥WORK. PRC
```

実際に使用するフォントテーブルの名前

```
C:¥DEFAULT. FTB
```

環境変数情報

```
PRTFILE=C:¥WORK. PRC, , FONT (C:¥DEFAULT. FTB
```

5.4.1.72 ファイル識別名(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)

$$\text{file-Identifier} = \left\{ \begin{array}{l} \text{LPTn:} \\ \text{COMn:} \\ \text{PRTNAME:プリンタ名} \end{array} \right\} \quad [, [\text{INF}(\text{xx})] [, \text{FONT}(\text{yy})]]$$

xx : 印刷情報ファイル名

yy : フォントテーブル名

FORMAT句なし印刷ファイルを利用する場合のファイル識別名の指定方法について説明します。

ファイル識別名には、COBOLソースプログラムのASSIGN句に記述したファイル識別名を指定します。プログラム中でファイル識別名を英小文字で記述した場合でも、環境変数情報のファイル識別名は英大文字で指定してください。

“LPTn:”/“COMn:”/“PRTNAME:プリンタ名”には、実際に帳票出力を行うプリンタが接続されているローカルプリンタポート名(LPTn:)/通信ポート名(COMn:)/プリンタの名前(PRTNAME:プリンタ名)を指定します。ローカルプリンタポート名(LPTn:)/通信ポート名(COMn:)/プリンタの名前(PRTNAME:プリンタ名)の指定は、省略することもできます。省略した場合、印刷情報ファイルの“PRTOUT”キーでこれらを指定する必要があります。ファイル識別名および印刷情報ファイルに、それぞれ異なるローカルプリンタポート名(LPTn:)/通信ポート名(COMn:)/プリンタの名前(PRTNAME:プリンタ名)が指定された場合、印刷情報ファイルの指定が優先されます。どちらも省略された場合または指定に誤りがある場合は、実行時エラーとなります。

Systemwalker/ListWORKS または Interstage List Works (以降ListWORKS)との連携による電子帳票出力機能を使用する場合、“LPTn:”および“COMn:”は指定できません。この場合、“PRTNAME:プリンタ名”を使用し、プリンタ名にはListWORKSで電子化・登録するときに使用する電子保存装置、またはListWORKS仮想プリンタ名を指定する必要があります。詳細については、“[8.6 電子帳票出力機能を使う方法](#)”を参照してください。

印刷情報ファイル名には、印刷情報ファイル名を絶対パスで指定します。印刷情報ファイルには、出力される帳票に関する状態制御を行ういくつかの情報を定義します。印刷情報ファイルの詳細については、“[8.1.9 印刷情報ファイル](#)”を参照してください。

フォントテーブル名には、フォントテーブルのファイル名を絶対パスで指定します。フォントテーブルには、PRINTING MODE句のFONT-nnn(書体番号)指定に対応するフォントフェイス名や印字スタイルの情報を定義します。ここで指定されたフォントテーブルは、そのファイルに対してだけ有効なフォントテーブルとなります。すべての印刷ファイルに共通なフォントテーブル(実行環境単位で有効なフォントテーブル)の指定については、“[5.4.1.40 @CBR_PrintFontTable \(印刷ファイルで使用するフォントテーブルの指定\)](#)”を参照してください。

また、フォントテーブルの詳細については、“[8.1.2 印字文字](#)”の“印字文字の書体”および“印字文字のスタイル”、“[8.1.14 フォントテーブル](#)”を参照してください。

なお、実行環境単位で有効なフォントテーブル名とファイル単位で有効なフォントテーブル名が異なる場合、ファイル単位の指定が優先されます。

印刷情報ファイル名とフォントテーブル名の指定順序は、どちらが先でもかまいません。



例

例1

COBOL ソースプログラムのASSIGN句の記述

```
ASSIGN TO PRTFILE
```

実際に帳票を出力するプリンタの名前

```
FUJITSU VSP4620
```

実際に使用する印刷情報ファイル名とそのパス

```
C:¥PRINT. INF
```

実際に使用するフォントテーブルとそのパス

```
C:¥DEFAULT. FTB
```

環境変数情報

```
PRTFILE=PRTNAME:FUJITSU VSP4620, , INF (C:¥PRINT. INF) , FONT (C:¥DEFAULT. FTB
```

または

```
PRTFILE=PRTNAME:FUJITSU VSP4620, , FONT (C:¥DEFAULT. FTB) , INF (C:¥PRINT. INF)
```

例2

COBOL ソースプログラムのASSIGN句の記述

ASSIGN TO PRINTER-1

実際に帳票を出力するローカルプリンタポート名

LPT1:

実際に使用する印刷情報ファイル名とそのパス

使用しない

実際に使用するフォントテーブルとそのパス

C:¥DEFAULT.FTB

環境変数情報

PRINTER-1=LPT1:, FONT (C:¥DEFAULT.FTB)

例3

COBOL ソースプログラムのASSIGN句の記述

ASSIGN TO PRINTER-2

実際に帳票を出力するローカルプリンタポート名

COM1:

実際に使用する印刷情報ファイル名とそのパス

使用しない

実際に使用するフォントテーブルとそのパス

使用しない

環境変数情報

PRINTER-2=COM1:

例4

COBOL ソースプログラムのASSIGN句の記述

ASSIGN TO PRTFILE

実際に帳票を出力するローカルプリンタポート名

FUJITSU VSP4620

実際に使用する印刷情報ファイル名とそのパス

C:¥PRINT.INF

実際に使用するフォントテーブルとそのパス

使用しない

環境変数情報

PRTFILE=, INF (C:¥PRINT.INF)

印刷情報ファイル(C:¥PRINT.INF)の指定

```
PRTOUT=PRTNAME:FUJITSU VSP4620
```

例5

COBOL ソースプログラムのASSIGN句の記述

```
ASSIGN TO PRTFILE
```

実際に帳票を出力するローカルプリンタポート名

```
FUJITSU VSP4620A
```

実際に使用する印刷情報ファイル名とそのパス

```
C:¥PRINT. INF
```

実際に使用するフォントテーブルとそのパス

使用しない

環境変数情報

```
PRTFILE=PRTNAME:FUJITSU VSP4620A, INF (C:¥PRINT. INF)
```

印刷情報ファイル(C:¥PRINT.INF)の指定

```
PRTOUT=PRTNAME:FUJITSU VSP4620A
```

5.4.1.73 FCBxxxx(FCB制御文の指定)

FCBxxxx = FCB 制御文

xxxxには、I制御レコードに指定したFCB名を指定します。FCB制御文の指定形式については、“[8.1.4 FCB](#)”を参照してください。

5.4.1.74 FOVLDIR(フォームオーバーレイパターンのフォルダの指定)

FOVLDIR = フォルダ

フォームオーバーレイパターンの格納先フォルダを絶対パス名で指定します。省略された場合、フォームオーバーレイパターンの焼付けは行われません。

なお、複数のフォルダを指定することはできません。

5.4.1.75 FOVLTYPE(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 = 形式

フォームオーバーレイパターンのファイル名の先頭4文字が“KOL5”(省略値)以外の場合、形式に、ファイル名の先頭4文字を指定します。ファイル名に形式を持たない場合は、“None”を指定してください。

5.4.1.76 FOVLNAME(フォームオーバーレイパターンのファイル名の指定)

FOVLNAME = ファイル名

フォームオーバーレイパターンのファイル名から先頭4文字の形式部分を除いた、後半のファイル名を4文字以内で指定します。



例

フォームオーバーレイパターンファイルが“C:¥FOVLDATA¥KOL5FOVL.OVD”の場合

```
FOVLDIR=C:¥FOVLDATA
FOVLTYPE=KOL5
FOVLNAME=FOVL
OVD_SUFFIX=OVD
```



注意

本指定は、PrintWalker/OVLオプションによるフォームオーバーレイ印刷およびSystemwalker/ListWORKS連携のフォームオーバーレイ印刷時は有効になりません。

5.4.1.77 OVD_SUFFIX(フォームオーバーレイパターンのファイルの拡張子の指定)

OVD_SUFFIX = 拡張子

フォームオーバーレイパターンのファイルの拡張子が“OVD”(省略値)以外の場合、拡張子に、拡張子の文字列を指定します。ファイル名に拡張子がない場合には、文字列“None”を指定してください。

5.4.1.78 SYSCOUNT(COUNT情報の出力ファイルの指定)

SYSCOUNT = ファイル名[,MOD]

COUNT機能を使用する場合、COUNT情報の出力先となるファイルの名前を指定します。[参照]“[19.4 COUNT機能の使い方](#)”

ファイル名には、絶対パスと相対パスを指定できます。相対パスが指定された場合は、カレントフォルダからの相対パスになります。

フォルダ名またはファイル名にコンマ(,)を含む場合、フォルダ名またはファイル名を二重引用符(")で囲む必要があります。

第2引数にMOD以外の文字列が指定された場合、JMP0726I-Wのメッセージを出力します。この際、COUNT情報は出力されません。

MODが指定されない場合、ファイル名に指定されたファイルが存在すれば上書きします。ファイル名に指定されたファイルが存在しなければ新規にファイルを作成します。

MODが指定された場合、ファイル名に指定されたファイルが存在すれば追加書きします。ファイル名に指定されたファイルが存在しなければ新規にファイルを作成します。

COUNT情報ファイルの上限サイズは1GBになります。ファイルのサイズが1GB以上のCOUNT情報ファイルに対して追加書きを行うと、JMP0727I-Wのメッセージを出力してCOUNT情報を追加書きしません。

5.4.1.79 SYSINのアクセス名（小入出力機能の入力ファイルの指定）

SYSIN のアクセス名 = ファイル名[,DUMMY]

SYSINのアクセス名には、翻訳オプションSSINに指定した環境変数情報名を指定し、ファイル名には、小入出力機能のACCEPT文を実行したときのデータの入力先となるファイルの名前を指定します。[参照]“[A.2.52 SSIN \(ACCEPT文のデータの入力先\)](#)”

ファイル名には、絶対パスと相対パスを指定できます。相対パスが指定された場合は、カレントフォルダからの相対パスになります。



例

COBOL ソースプログラムを翻訳するときの翻訳オプションSSINの指定

SSIN (INFILE)

ACCEPT文の入力先ファイルの名前

A:¥INDATA. TXT

環境変数情報

INFILE=A:¥INDATA. TXT

指定した文字列は以下のような意味を持ちます。

DUMMY

ダミーファイル機能を使用します。

[参照]“[11.1.5.5 ACCEPT文のファイル入力拡張機能](#)”

5.4.1.80 SYSOUTのアクセス名（小入出力機能の出力ファイルの指定）

SYSOUTのアクセス名 = ファイル名[,MOD][,DUMMY]

SYSOUTのアクセス名には、翻訳オプションSSOUTに指定した環境変数情報名を指定し、ファイル名には、小入出力機能のDISPLAY文を実行したときのデータの出力先となるファイルの名前を指定します。[参照]“[A.2.53 SSOUT \(DISPLAY文のデータの出力先\)](#)”

ファイル名には、絶対パスと相対パスを指定できます。相対パスが指定された場合は、カレントフォルダからの相対パスになります。すでに、同一のファイルが存在する場合は、ファイルは上書きされます。



例

COBOL ソースプログラムを翻訳するときの翻訳オプションSSOUT の指定

SSOUT (OUTFILE)

DISPLAY 文の出力先ファイルの名前

A:¥OUTDATA. TXT

環境変数情報

OUTFILE=A:¥OUTDATA. TXT

指定した文字列は以下のような意味を持ちます。

MOD

既に存在するファイルにデータを追加します。

DUMMY

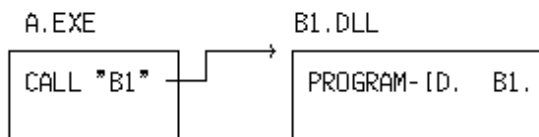
ダミーファイル機能を使用します。

[参照]“[11.1.5.4 DISPLAY文のファイル出力拡張機能](#)”

5.4.2 副プログラムのエントリ情報

実行可能プログラムの構造が動的プログラム構造の場合、呼び出すプログラムが格納されているDLLを特定するために、エントリ情報の指定が必要になります。

ただし、動的プログラム構造の場合でも、呼び出すプログラムのDLL名が“プログラム名.DLL”の場合は、エントリ情報を指定する必要はありません。たとえば、以下のように、呼び出すエントリ名が“B1”で、DLL名が“B1.DLL”の場合、エントリ情報の指定は必要ありません。



動的プログラム構造については、“[4.3.2 動的構造](#)”を参照してください。

5.4.2.1 エントリ情報の指定方法

エントリ情報の指定方法には、以下の2つがあります。

- ・ エントリ情報ファイルに指定する方法
- ・ 実行用の初期化ファイルに指定する方法

NetCOBOLでは、複数のアプリケーションから共通のDLLを呼び出す場合に、共通のDLLのエントリ情報をまとめて指定できる方法として、エントリ情報ファイルに指定する方法をおすすめしています。

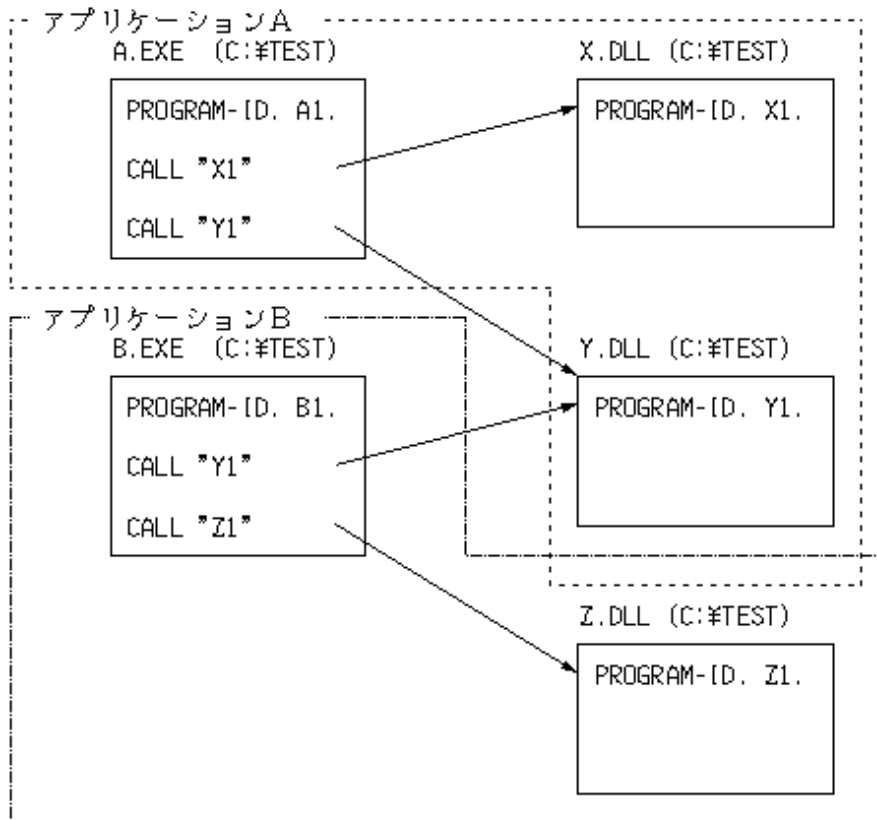
エントリ情報ファイルと実行用の初期化ファイルでエントリ情報の指定が重なる場合は、実行用の初期化ファイルに指定した情報が有効になります。



注意

マルチスレッドモードでは、実行用の初期化ファイルに指定したエントリ情報は無効になります。エントリ情報ファイルに情報を指定してください。

エントリ情報の指定方法について、以下の例を使って説明します。アプリケーションAおよびアプリケーションBはともに動的プログラム構造で、X.DLL、Y.DLLおよびZ.DLLを呼び出します。この中で、Y.DLLはアプリケーションAおよびアプリケーションBから呼び出される共通のDLLです。



エントリ情報ファイルに指定する方法

エントリ情報ファイルを使用する場合、環境変数情報@CBR_ENTRYFILEにエントリ情報ファイル名を指定しなければなりません。[参照]“5.4.1.23 @CBR_ENTRYFILE(エントリ情報ファイルの指定)”



例

エントリ情報ファイル名:C:¥TEST¥FILE.ENT

```
@CBR_ENTRYFILE = C:¥TEST¥FILE.ENT
```

エントリ情報ファイルの内容を以下に示します

エントリ情報ファイル (C:¥TEST¥FILE.ENT)

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

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

[2] エントリ情報を指定します。エントリ情報の指定形式については、“[5.4.2.2 副プログラム名の指定形式](#)”、“[5.4.2.3 二次入口点名の指定形式](#)”および“[18.2.4.4 クラスとメソッドのエントリ情報](#)”を参照してください。

.....

行の先頭に;(セミicolon)がある場合、セミicolonから改行までの間はコメントして認識されます。
コメント行が多い場合、コメント行の読み飛ばし処理のため、処理速度が低下する可能性があります。
上記の例の場合、エントリ情報ファイルの内容は以下のようになります。

エントリ情報ファイル (C:\TEST\FILE.ENT)

```
[ENTRY]
X1=C:\TEST\X.DLL
Y1=C:\TEST\Y.DLL
Z1=C:\TEST\Z.DLL
```

[ENTRY]セクションに指定された内容は、すべてのアプリケーションに有効です。つまり、アプリケーションAおよびアプリケーションBで共通に使用するY.DLLは、ここで1回定義するだけで、両方のアプリケーションで有効になります。

注意

-
- ・ エントリ情報では、英小文字と英大文字が区別されますので、指定時には注意してください。
 - ・ 同一の副プログラム名を複数個指定しないでください。同一の副プログラム名を複数個指定した場合の動作については保証されません。
-

実行用の初期化ファイルに指定する場合

実行用の初期化ファイルの内容および使用方法については、“[5.3.2.3 実行用の初期化ファイルに設定する方法](#)”を参照してください。
ここでは、実行用の初期化ファイルのエントリ情報の指定方法についてだけ説明します。

例

実行用の初期化ファイル (COBOL85.CBR)

```

:
[プログラム名]                ... [1]
:
[プログラム名.ENTRY]          ... [2]
 エントリ情報
:
:
```

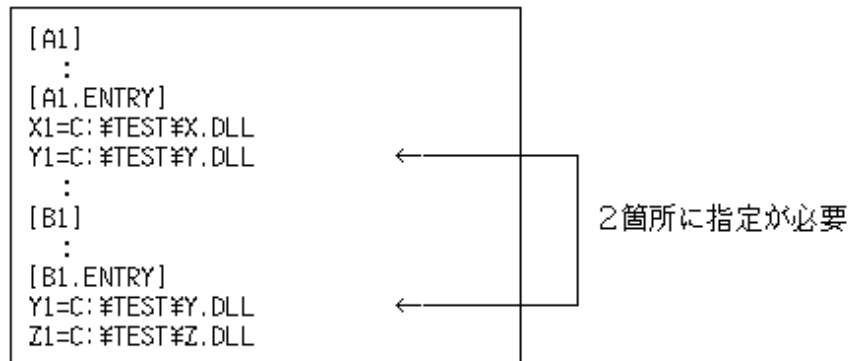
[1] 主プログラムのプログラム名(PROGRAM-ID段落に指定した名前)をセクション名として指定します。

[2] 主プログラムのプログラム名に“.ENTRY”を付加した名前をセクション名として指定します。そのプログラムから直接的または間接的に呼び出されるプログラムのエントリ情報を指定します。

.....

上記の例の場合、実行用の初期化ファイルの内容は以下のようになります。

実行用の初期化ファイル (C:\%TEST%\COBOL85.CBR)



アプリケーション単位でセクションを指定し、セクションの下にそのアプリケーションに対するエントリ情報を指定します。つまり、複数のアプリケーションで共通で使用するDLLは、それぞれのセクションの下で定義しなければなりません。

5.4.2.2 副プログラム名の指定形式

副プログラム名 = DLL名

副プログラム名とその副プログラムを含むDLL名を関連付けます。

副プログラム名には、呼び出されるプログラムのプログラム名 (PROGRAM-ID段落に指定した名前) を指定し、DLL名には、呼び出されるプログラムが格納されているDLLのファイル名を絶対パスまたは相対パスで指定します。相対パスで指定した場合は、以下の検索順序に従ってDLLが検索されます。

1. 実行可能ファイルの存在するフォルダ
2. カレントフォルダ
3. Windowsのシステムフォルダ
4. Windowsのフォルダ
5. 環境変数PATHに指定されているフォルダ

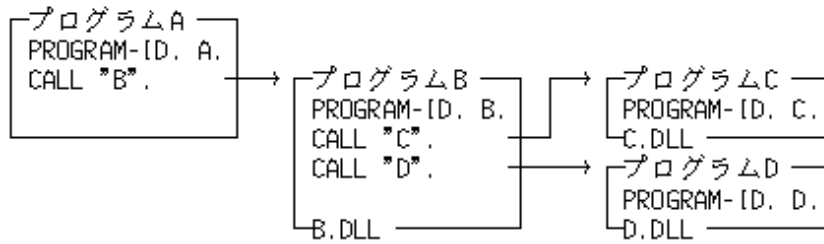
DLL名の拡張子は、“DLL”でなければなりません。



例

DLLが1つの副プログラムで構成されている場合の例

(プログラムの呼出し関係)



(1) エントリ情報ファイルに指定する場合

@CBR_ENTRYFILE=FILE

```
FILE
[ENTRY]
B=B.DLL
C=C.DLL
D=D.DLL
```

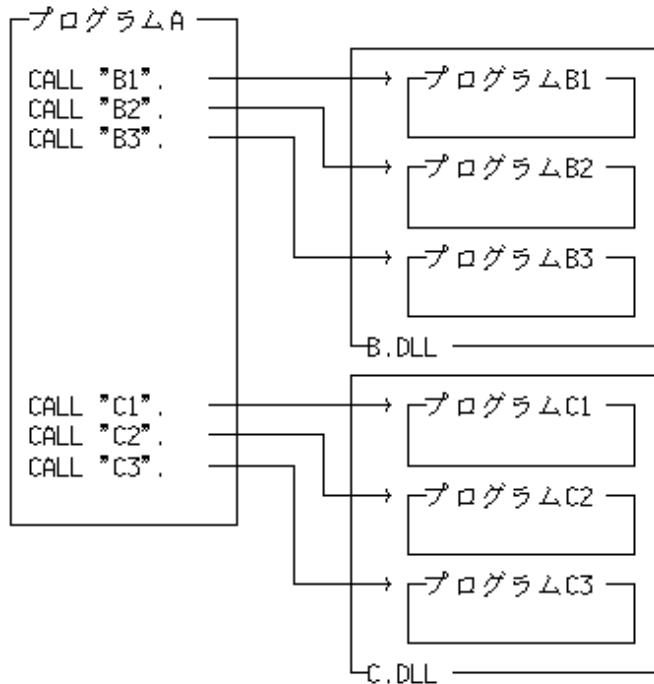
(2) 実行用の初期化ファイルに指定する場合

```
[A.ENTRY]
B=B.DLL
C=C.DLL
D=D.DLL
```

この例では、DLL名が“プログラム名.DLL”となっているため、エントリ情報を省略することができます。

DLLが複数の副プログラムで構成されている場合の例

〔プログラムの呼出し関係〕



(1) エントリ情報ファイルに指定する場合

@CBR_ENTRYFILE=FILE

```
FILE
[ENTRY]
B1=B.DLL
B2=B.DLL
B3=B.DLL
C1=C.DLL
C2=C.DLL
C3=C.DLL
```

(2) 実行用の初期化ファイルに指定する場合

```
[A.ENTRY]
B1=B.DLL
B2=B.DLL
B3=B.DLL
C1=C.DLL
C2=C.DLL
C3=C.DLL
```

DLL内の呼び出されたすべての副プログラムに対してCANCEL文が実行されたときに、DLLは仮想記憶上から削除されます。

上記の例では、B1、B2およびB3のすべての副プログラムに対してCANCEL文が実行されたときに、B.DLLが仮想記憶上から削除されます。また、C1、C2およびC3のすべての副プログラムに対してCANCEL文が実行されたときに、C.DLLが仮想記憶上から削除されます。

マルチスレッドで利用する場合は、“[24.10.5 動的プログラム構造](#)”を参照してください。

二次入口点名を指定する場合は、“[5.4.2.3 二次入口点名の指定形式](#)”の指定を副プログラム名の指定形式に追加します。

5.4.2.3 二次入口点名の指定形式

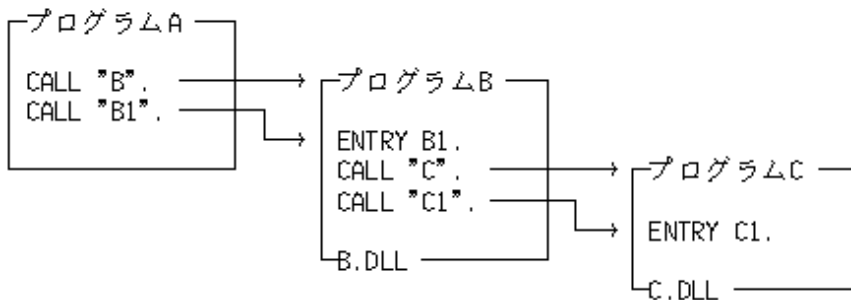
二次入口点名 = 副プログラム名

二次入口点名には、呼び出すプログラムのENTRY文に記述されている名前を指定し、副プログラム名には、そのENTRY文を持つプログラム名を指定します。



例

(プログラムの呼出し関係)



(1) エントリ情報ファイルに指定する場合

@CIBR_ENTRYFILE=FILE

FILE

[ENTRY]

B=B.DLL

C=C.DLL

B1=B

C1=C

(2) 実行用の初期化ファイルに指定する場合

[A.ENTRY]

B=B.DLL

C=C.DLL

B1=B

C1=C

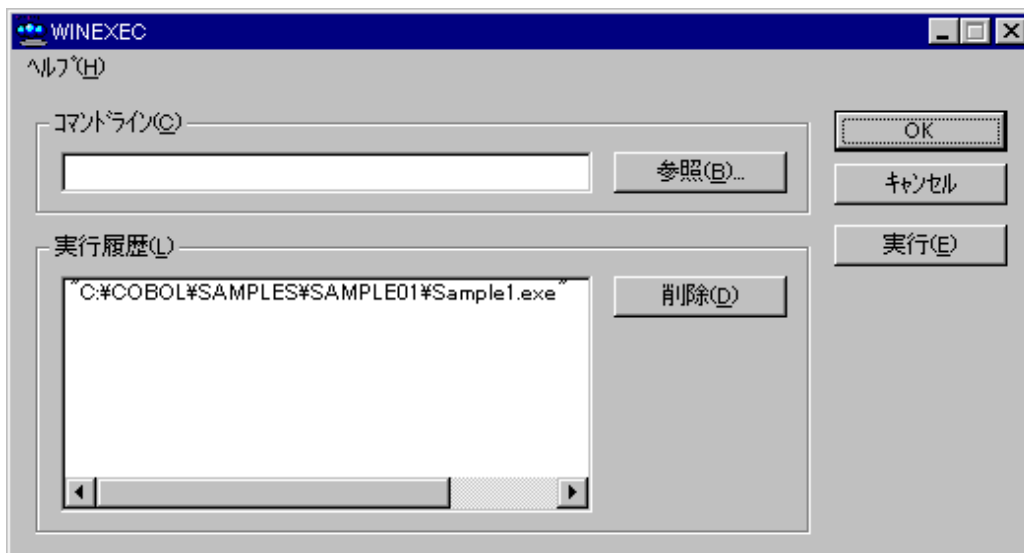
この例では、DLL名が“プログラム名.DLL”となっているため、“B=B.DLL”および“C=C.DLL”のエントリ情報は省略することができます。

5.5 WINEXECウィンドウ

WINEXECウィンドウは、実行操作を行うためのウィンドウです。ここでは、WINEXECウィンドウについて説明します。ウィンドウの使い方の詳細は、ヘルプを参照してください。

5.5.1 WINEXECウィンドウの起動

WINEXECウィンドウを起動には、プロジェクトマネージャの[ツール]メニューから、“実行”を選択してください。



メニューバー

ヘルプ

WINEXECウィンドウのヘルプを表示します。

エディットボックス

コマンドライン

実行するファイルを指定します。

リストボックス

実行履歴

実行したファイルの履歴を表示します。

ボタン

OK

実行を開始するときに選択します。実行後、WINEXECウィンドウを閉じます。

キャンセル

WINEXECウィンドウを閉じます。

実行

実行を開始するときに選択します。

参照

ファイルを参照するときに選択します。

削除

リストボックスからファイルの実行履歴を削除するときに選択します。

5.5.2 ファイル名の指定

以下のどちらかの方法で、実行するファイルの名前をコマンドラインのエディットボックスに指定します。

- 実行するファイルの名前を直接キー入力するか、[参照]ボタンを使って指定します。
- すでに実行したことのあるプログラムを再度実行する場合、実行履歴のリストボックスの中から実行したいファイル名を選択します。



参考

実行履歴のリストボックスには、最大10個のファイルが表示されます。11個以上になると、1個目のファイルから順に置き換えられていきます。

5.5.3 実行の開始と終了

実行を開始するには、[実行]ボタンをクリックします。

実行が終了したら、WINEXECウィンドウの[キャンセル]ボタンをクリックし、WINEXECウィンドウを閉じてください。



注意

COBOLプログラムの場合、実行時に必要な実行環境情報は、あらかじめ実行環境設定ツール([参照]“5.6 実行環境設定ツール”)などで設定しておいてください。

5.6 実行環境設定ツール

実行環境設定ツール(COBENVUT.EXE)は、COBOLプログラムの実行時に使用する実行用の初期化ファイルの内容を編集するためのツールです。COBOLプログラムの実行時に実行用の初期化ファイルを使用する場合は、プログラムの実行前にあらかじめこのツールで実行用の初期化ファイルの内容を編集してください。このとき、ファイルのパス名やプリンタ名の指定は、動作環境によって異なる場合がありますため、注意が必要です。

実行用の初期化ファイルを格納する位置については、“5.3.2.3 実行用の初期化ファイルに設定する方法”を参照してください。

以下に、実行環境設定ツールを示します。

設定する実行環境情報の指定形式の詳細は、“5.4 実行環境情報の指定形式”を参照してください。ツールの使い方の詳細は、ヘルプを参照してください。



メニューバー

ファイル

実行用の初期化ファイルのオープン/クローズとツールの終了を行います。

環境設定

プリンタ名の取込み、文字コードの設定および各種フォントの指定を行います。

ヘルプ

実行環境設定ツールのヘルプを表示します。

エディットボックス

セクション名

編集対象のセクション名(COBOLの主プログラム)を入力します。現在読み込んでいる実行用の初期化ファイル内のセクション名が、リストに表示されます。

変数名

環境変数名を入力します。COBOLランタイムシステムで予約されている環境変数名については、リストに表示されます。

変数値

環境変数の値を入力します。

リストボックス

共通タブ

現在設定されている、実行用の初期化ファイルの共通部の設定内容を表示します。

セクションタブ

“セクション名”に表示されているセクションに対応した実行用の初期化ファイルの設定内容を表示します。

ラジオボタン

スレッドモード

編集を行おうとしている対象のプログラムがシングルスレッドモードで動作するものか、マルチスレッドモードで動作するものかを指定します。マルチスレッドモードで動作するCOBOLプログラムでは、実行用の初期化ファイルの共通部の情報だけが有効となります。

ボタン

設定

選択されているリストボックスに、変数名と変数値に記述された情報を設定するときに選択します。実行用の初期化ファイルに設定内容を反映させるには、さらに“適用”ボタンを押下する必要があります。

削除

リストボックスで選択されている変数名と変数値をリストボックスから削除するときに選択します。

適用

選択されているリストボックスの内容を実行用の初期化ファイルに反映するときに選択します。

スタティックテキスト

ファイル名

実行用の初期化ファイル名が表示されます。

5.6.1 環境変数情報の設定

環境変数情報の設定は、実行環境設定ツールのセクション名、変数名、変数値のエディットボックスを使用して行います。

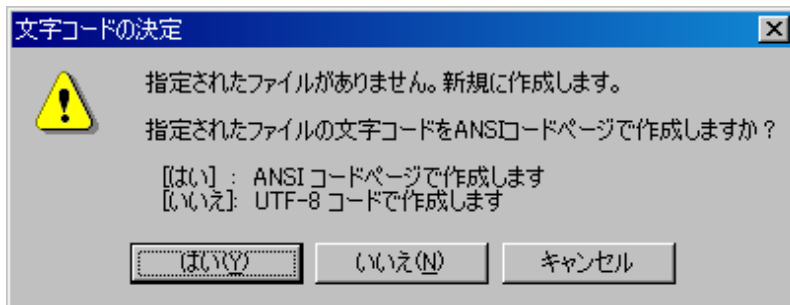
ここでは、環境変数情報を設定するときの操作方法について説明します。

5.6.1.1 実行用の初期化ファイルのオープン

編集を行う実行用の初期化ファイルをオープンする場合、以下の操作を行います。

1. [ファイル]メニューの“開く”を選択します。

2. [実行用の初期化ファイルの指定]ダイアログが表示されますので、既存のファイルを開く場合は、ファイルを選択してください。新規に実行用の初期化ファイルを作成する場合は、そのファイル名をエディットボックスに入力してください。指定できるファイルの文字コードは、ANSIコードページ(シフトJIS)とBOM付きUTF-8です。ファイルを新規に作成する際は、以下のダイアログにより、文字コードを選択してください。「はい」を指定するとシフトJISでファイルを作成します。「いいえ」を指定するとBOM付きUTF-8でファイルを作成します。



既存ファイルのオープンでは、シフトJISまたは、BOM付きUTF-8を自動的に判定してファイルを開きます。
BOM付きUTF-8のファイルを開いた時は、実行環境設定ツールのタイトルに (UTF-8) の文字が表示されます。

3. [実行用の初期化ファイルの指定]ダイアログの[開く]ボタンをクリックします。
→ リストボックスに実行用の初期化ファイルの内容が表示されます。

5.6.1.2 環境変数情報の追加

新しく環境変数情報を追加する場合、以下の操作を行います。

1. [変数名]コンボボックスのリストから追加する環境変数情報を選択します。
2. [変数値]に情報を設定します。
3. [設定]ボタンをクリックします。
→ リストボックスに環境変数情報が追加されます。

5.6.1.3 環境変数情報の変更

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

1. リストボックスから変更する環境変数情報を選択します。
→ 選択した情報は、変数名および変数値のエディットボックスに表示されます。
2. 指定内容を変更します。
3. [設定]ボタンをクリックします。
→ 変更した環境変数情報がリストボックスに表示されます。

5.6.1.4 環境変数情報の取消し

リストボックスに表示されている環境変数情報を取り消す場合、以下の操作を行います。

1. リストボックスから削除する環境変数情報を選択します。
2. [削除]ボタンをクリックします。
→ 選択した環境変数情報がリストボックスから削除されます。

5.6.1.5 実行用の初期化ファイルのクローズ

現在オープンしている実行用の初期化ファイルをクローズする場合、以下の操作を行います。

1. [ファイル]メニューの“閉じる”を選択します。



注意

実行用の初期化ファイルをクローズする場合、その時点で未適用の情報は実行用の初期化ファイルに反映されません。

5.6.2 エントリ情報の設定

エントリ情報については、あらかじめエントリ情報ファイルを作成し、環境変数情報@CBR_ENTRYFILEに、エントリ情報ファイル名を指定してください。

指定形式については、“5.4.1.23 @CBR_ENTRYFILE (エントリ情報ファイルの指定)”を参照してください。

5.6.3 実行用の初期化ファイルへの保存

[適用]ボタンをクリックすると、リストボックス内の情報が実行用の初期化ファイルに保存されます。

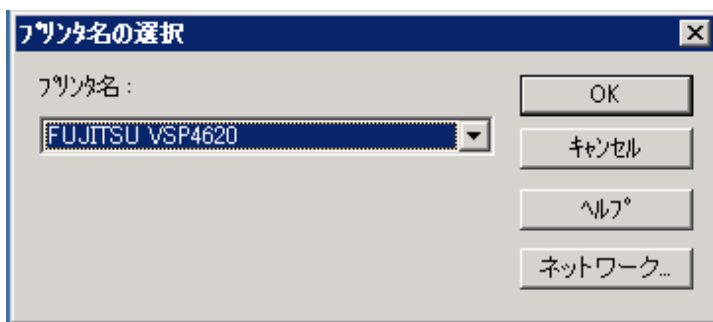


注意

適用を行うと、前回の実行用の初期化ファイルの保存内容は書き換えられます。

5.6.4 プリンタの設定

プリンタを使用する場合、プリンタ情報を設定することができます。実行環境設定ツールの[環境設定]メニューから“プリンタ名”を選択し、[プリンタ名の選択]ダイアログで必要な情報を設定してください。



5.6.5 実行環境設定ツールの終了

実行環境設定ツールの[ファイル]メニューから“終了”を選択すると、実行環境設定ツールが終了します。



注意

実行環境設定ツールを終了する場合、その時点で未適用の情報は実行用の初期化ファイルに反映されません。

5.7 実行用の初期化ファイル名の指定形式

ここでは、プログラムの主プログラムがCOBOLの場合および他言語(Cプログラム/Visual Basicプログラム)の場合の実行用の初期化ファイル名の指定方法について説明します。なお、ここでいう、プログラムの主プログラムとは、システムから最初に制御が渡ったプログラムのことをいいます。

プログラムの主プログラムがCOBOLプログラムの場合

実行用の初期化ファイル名は、識別子-CBRまたは/CBRの後ろに指定します。

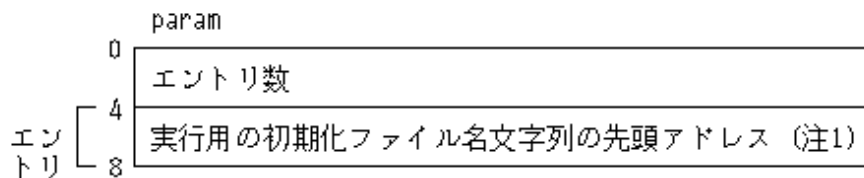
プログラムの主プログラムがCプログラムの場合

JMPCINTCをJMPCINT2の呼出し直後に呼び出してください。

参考

JMPCINTCは、C呼出し規約で呼び出してください。詳細については、“[10.1.2 各種呼出し規約の違い](#)”を参照してください。

```
int JMPCINTC(int 0,void &param)
```



エントリ数：現在は1 となる。

注：パス名(省略可)を含む実行用の初期化ファイル名を表すNULL(終端記号)で終わる文字列の先頭アドレス。なお、指定できる文字列の最大長は、255文字(終端記号を含む)です。

JMPCINTCの返却値	意味
0	正常終了
-1	JMPCINT2の未呼出し／機能コードの指定誤り／作業領域不足
1	多重呼出し (JMPCINTCがすでに呼び出されている場合を意味します。実行用の初期化ファイル名は、最初に実行されたJMPCINTCで指定されたものが有効となります。)

プログラムの主プログラムがVisual Basicプログラムの場合

Declare句に以下の記述を行った上で、JMPCINT2の呼出し直後にJMPCINTBを呼び出してください。

参照

JMPCINTBは、STDCALL呼出し規約で呼び出してください。詳細は、“[10.1.2 各種呼出し規約の違い](#)”を参照してください。

Declare句への記述

```
Private Declare Sub JMPCINT2 LIB "F3BIPRCT.DLL" ()  
Private Declare Sub JMPCINT3 LIB "F3BIPRCT.DLL" ()  
Private Declare Function JMPCINTB LIB "F3BIPRCT.DLL" _ (注1)  
    (ByVal A As Long, ByVal D As String) As long
```

注1：この行は次行とつながっています。実際には、1行で記述してください。

例

JMPCINTBの呼出し例 (注2)

```
Static DATA As String * 12  
Dim ANS As Long  
  
DATA = "C:\¥TEST.CBR" & Chr(0)  
ANS = JMPCINTB(0, DATA)
```

注2：JMPCINTBの各引数/返却値の意味は以下のとおりです。

JMPCINTBの引数	意味
第1引数	機能コード(CBR 名の変更の場合は0を設定)
第2引数	実行用の初期化ファイル名文字列(Chr(0)(終端記号)を付加してください。指定できる文字列の最大長は、127 文字(終端記号を含む)です。)

JMPCINTBの返却値	意味
0	正常終了
-1	JMPCINT2の未呼出し／機能コードの指定誤り／作業領域不足
1	多重呼出し (JMPCINTBがすでに呼び出されている場合を意味します。実行用の初期化ファイル名は、最初に実行されたJMPCINTBで指定されたものが有効となります。)

5.8 実行時オプションの指定形式

実行時オプションは、実行時にCOBOLソースプログラムに対していくつかの情報や動作を指示します。COBOLプログラムで使用している機能や、COBOLソースプログラムを翻訳するときに指定したオプションによっては、実行時オプションを指定する必要があります。実行時オプションは、以下の方法で指定できます。

- ・ コマンドラインで指定する方法
- ・ 環境変数情報@GOPTで指定する方法



参照

“5.3.2.5 コマンドラインで設定する方法”

“5.4.1.56 @GOPT(実行時オプションの指定)”

実行時オプションの種類と指定形式を下表に示します。なお、実行時オプションを複数個指定する場合には、コンマ(,)で区切って指定します。

表5.6 実行時オプション

機能	オプション
トレース情報の個数の指定、およびTRACE機能の抑制指定	[r ^{個数} nor]
エラー検出時の処理実行回数の指定、およびCHECK機能の抑制指定	[c ^{回数} {noc nocb noci nocl nocn nocp}]
外部スイッチの値の指定	[s ^値]
PowerSORTが使用するメモリ容量を指定	[smsize ^{値k}]

[r^{個数} | nor] (トレース情報の個数の指定、およびTRACE機能の抑制指定)

TRACE機能が出力するトレース情報の数を変更したい場合に指定します。

個数には、出力するトレース情報の数を1～999999で指定します。

TRACE機能を抑制する場合は、norを指定します。

これらのオプションは、翻訳時に、-Drオプションまたは翻訳オプションTRACEを指定したプログラムにだけ有効です。



参照

“3.8.1.7 -Dr(TRACE機能を使用する指定)”

“A.2.58 TRACE(TRACE機能の使用の可否)”

[c回数 | {noc | nocb | noci | nocl | nocn | nocp}] (エラー検出時の処理実行回数の指定、およびCHECK機能の抑制指定)

CHECK機能でエラーを検出したときの処理続行回数を変更したい場合に指定します。

回数には、処理続行回数を0～999999で指定します。ただし、0が指定された場合は、上限なしとみなされます。

また、CHECK機能を抑制することもできます。抑制の対象となるCHECK機能は、以下のとおりです。複数指定することができます。

- noc : 全てのCHECK機能
- nocb : CHECK(BOUND)
- noci : CHECK(ICONF)
- nocl : CHECK(LINKAGE)
- nocn : CHECK(NUMERIC)
- nocp : CHECK(PRM)

これらのオプションは、翻訳時に、-Dkオプション、翻訳オプションCHECK(ALL)または対応する翻訳オプションを指定したプログラムにだけ有効です。



参照

“3.8.1.4 -Dk (CHECK機能を使用する指定)”

“A.2.5 CHECK (CHECK機能の使用の可否)”

[s値] (外部スイッチの値の指定)

COBOLプログラムの特殊名段落で指定した外部スイッチSWITCH-0～SWITCH-7に値を設定したい場合に指定します。値には、連続した8個のスイッチ値を、一番左がSWITCH-0に、その隣がSWITCH-1と順にSWITCH-7に対応するように指定します。外部スイッチには、それぞれ0または1が指定できます。省略した場合は、“s00000000”が指定されたとみなされます。なお、SWITCH-8を指定した場合、SWITCH-8はSWITCH-0に等しいため、スイッチ値の一番左がSWITCH-8に、その隣がSWITCH-1と順にSWITCH-7に対応します。

[smsize値k] (PowerSORTが使用するメモリ容量を指定)

SORT文およびMERGE文から呼び出されるPowerSORTが使用するメモリ空間の容量を限定したい場合に指定します。指定する値は、キロバイト単位の数字です。指定された値を、PowerSORTのBSRTPRIM構造体のmemory_sizeに設定します。指定された値が実際に有効になるかについては、PowerSORTの“オンラインマニュアル”をお読みください。

このオプションは、翻訳オプションSMSIZE()および特殊レジスタSORT-CORE-SIZEに指定する値の意味と等価ですが、同時に指定された場合の優先順位は、特殊レジスタSORT-CORE-SIZEが一番強く、以降、実行時オプションsmsize、翻訳オプションSMSIZE()の順で弱くなります。

5.9 注意事項

ここでは、プログラム実行時の注意事項について説明します。

5.9.1 COBOLプログラムの実行時にスタックオーバーフローが発生する場合

COBOLプログラムの実行時にスタックオーバーフローが発生する原因として、リンク時に指定したスタックサイズまたは省略時のスタックサイズ(1Mバイト)より、動作時に実際に必要とするスタックサイズの方が大きい場合が考えられます。このような場合は、まず、プログラム構造に問題がないかを見直し、問題がなければリンクオプションでスタックサイズを大きくして再リンクし、実行してください。



参照

“4.8.1 LINKコマンドの形式”

“4.5.2.4 リンクオプションの設定”

スタックサイズの求め方

$\text{トータルスタックサイズ} \quad \doteq \quad (\alpha_1 + \alpha_2 \cdots + \alpha_n) + (\beta_1 + \beta_2 \cdots + \beta_m)$

α_n : プログラム定義での使用スタックサイズ値

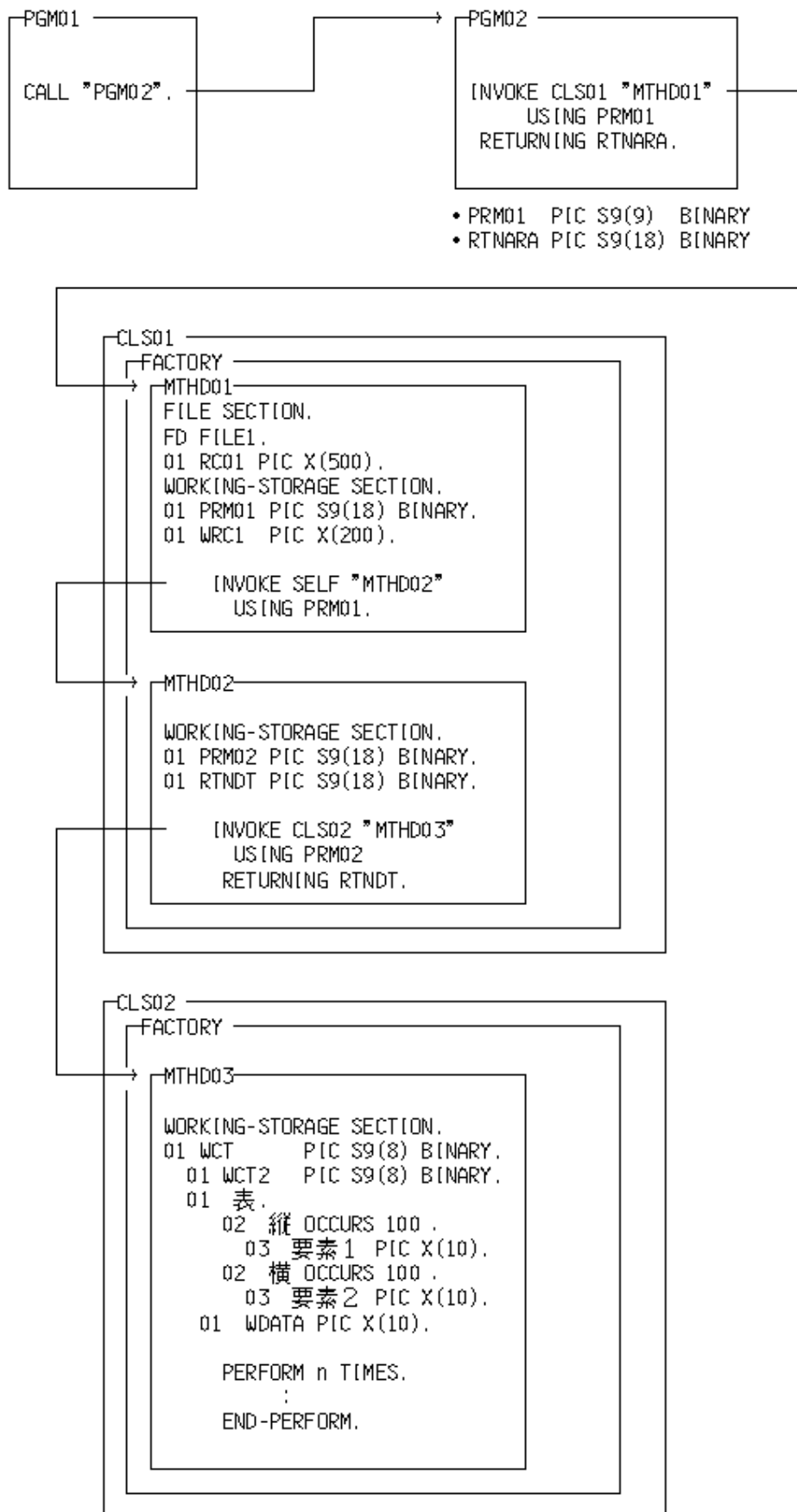
(nは実行されるプログラムの数で、n=1、2、…)

β_m : クラス/プロトタイプメソッドで呼び出されるメソッドの使用スタックサイズ値

(mは実行されるメソッドの数で、m=1、2、…)

個々のスタックサイズはセクションサイズリストから求めることができます。セクションサイズリストについては、“[3.2.2.6 データエリアに関するリスト](#)”を参照してください。

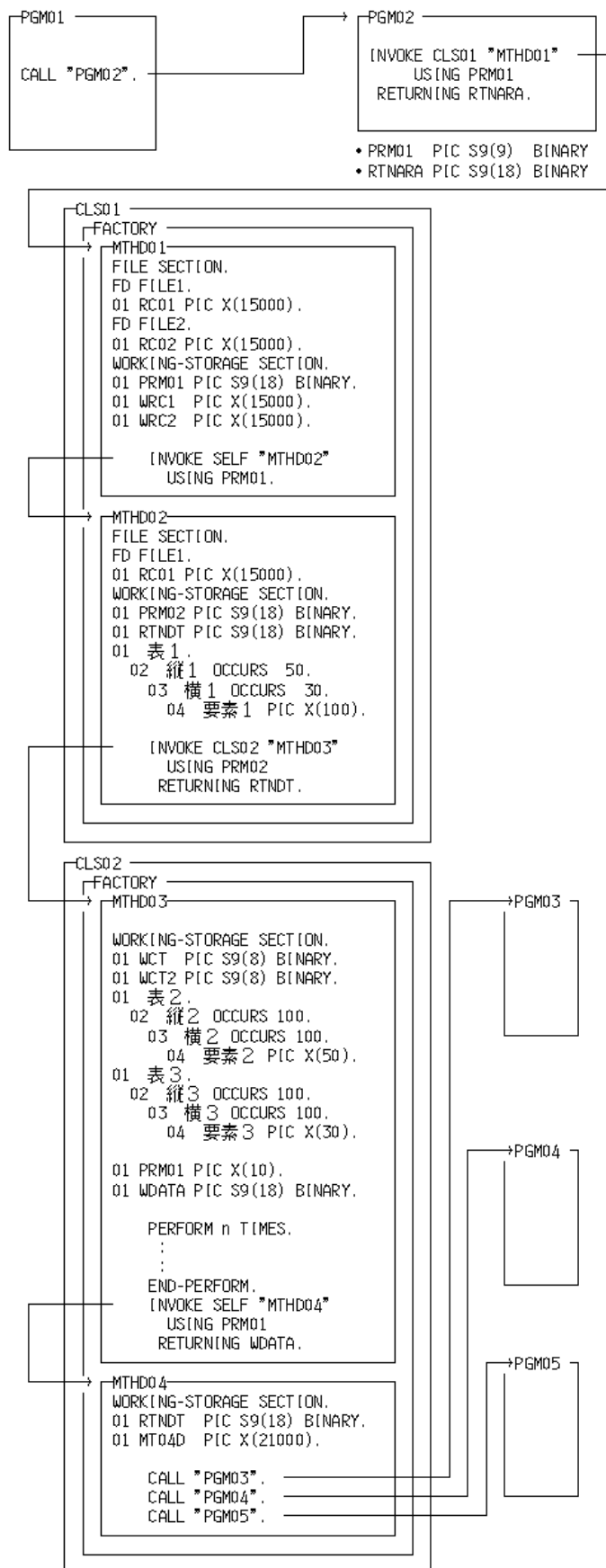
(例題1)オーバーフローしないケース



例題1でのスタック使用状況

P G M (プログラム定義)		
P G M 0 1		1 7 6 バイト
P G M 0 2		2 0 0 バイト
C L S (クラス定義)		
C L S 0 1		
M E T H O D (1)		1 , 0 6 4 バイト
M E T H O D (2)		3 7 6 バイト
C L S 0 2		
M E T H O D (3)		2 , 5 2 0 バイト
計		4 , 3 6 0 バイト

(例題2)オーバーフローするケース



例題2でのスタック使用状況

PGM（プログラム定義）

PGM01	176バイト
PGM02	200バイト
PGM03、PGM04、PGM05の最大スタックサイズ	240バイト

CLS（クラス定義）

CLS01	
METHOD (1)	60, 656バイト
METHOD (2)	165, 520バイト
CLS02	
METHOD (3)	4, 000, 392バイト
METHOD (4)	21, 296バイト

計	4, 248, 960バイト

スタックのオーバフローを回避する方法

オーバフローを回避するためには、リンク時にスタックサイズを拡張する/STACKオプションを指定します。

```
/STACK: reserve [, commit]
```

reserve

仮想メモリ内に確保するスタックの総サイズを指定します。

デフォルト値は1Mバイトです。指定した値は4バイト単位で切り上げられます。

commit

仮想メモリ内に確保したスタックの中の初期値を指定します。



例

```
/STACK: 8192000, 1024000  
          [1]      [2]
```

- [1] 仮想メモリ内に確保するスタックの総サイズ(この場合、8Mバイト)を指定します。
- [2] 確保したうち、初期値として指定するスタックサイズ(この場合、1Mバイト)を指定します。

5.9.2 COBOLプログラムの実行時に仮想メモリ不足が発生する場合

COBOLプログラムの実行時に仮想メモリ不足の発生する原因として、以下のような場合が考えられます。このような場合は、動作環境の見直しおよびプログラム構造の見直しを行ってください。

環境の問題

- ・実装メモリが少ない。
→ 必要であれば増設してください。

- ・ 仮想メモリが少ない。
→ 必要であれば大きくしてください。
- ・ 同時に実行している他のアプリケーションがメモリ領域を使用している。
→ 同時に実行している他のアプリケーションを停止してください。

プログラム構造の問題

- ・ 実行単位で同時にオープンしているファイルの数が多。
- ・ 実行単位でEXTERNAL句を指定したデータおよびファイルの宣言が多い。
- ・ 実行単位で同時に使用しているオブジェクト(インスタンス)の数が多など。

その他

- ・ 実行したアプリケーションがメモリ領域を破壊している。
→ 対話型デバッガ、CHECK機能およびメモリチェック機能などを使用して、領域破壊の原因を調査し、プログラムを修正してください。



参照

“第20章 対話型デバッガの使い方”

“19.2 CHECK機能の使い方”

“19.5 メモリチェック機能の使い方”

第6章 プロジェクト管理機能

本章では、COBOLアプリケーション開発および保守に有効なプロジェクト管理機能について説明します。

6.1 プロジェクト管理機能とは

ここでは、プロジェクト管理機能の概要を説明します。

プロジェクトマネージャでは、COBOLアプリケーション開発および保守に有効な、プロジェクト管理機能を提供しています。

プロジェクト管理機能では、1つ以上のファイル(COBOLソースプログラムおよび登録集)から生成されるアプリケーションを、プロジェクトという形で管理することができます。ファイルの依存関係などの情報は、プロジェクトファイルに格納され、翻訳オプションの情報は翻訳オプションファイルに、リンクオプションの情報はリンクオプションファイルにそれぞれ格納されます。

プロジェクトで管理されているプロジェクトファイル、翻訳オプションファイルおよびリンクオプションファイルの内容が更新されると、その情報を基に、更新されたファイルと依存関係にあるファイルを自動的に再翻訳および再リンクすることができます。これを“ビルド”または“MAKE(メイク)”といいます。また、プロジェクトに登録したすべてのプログラムを再翻訳および再リンクして、アプリケーションを作り直すことを“リビルド”といいます。

本製品では、プロジェクト管理機能を使ったサンプルプログラムを提供しています。

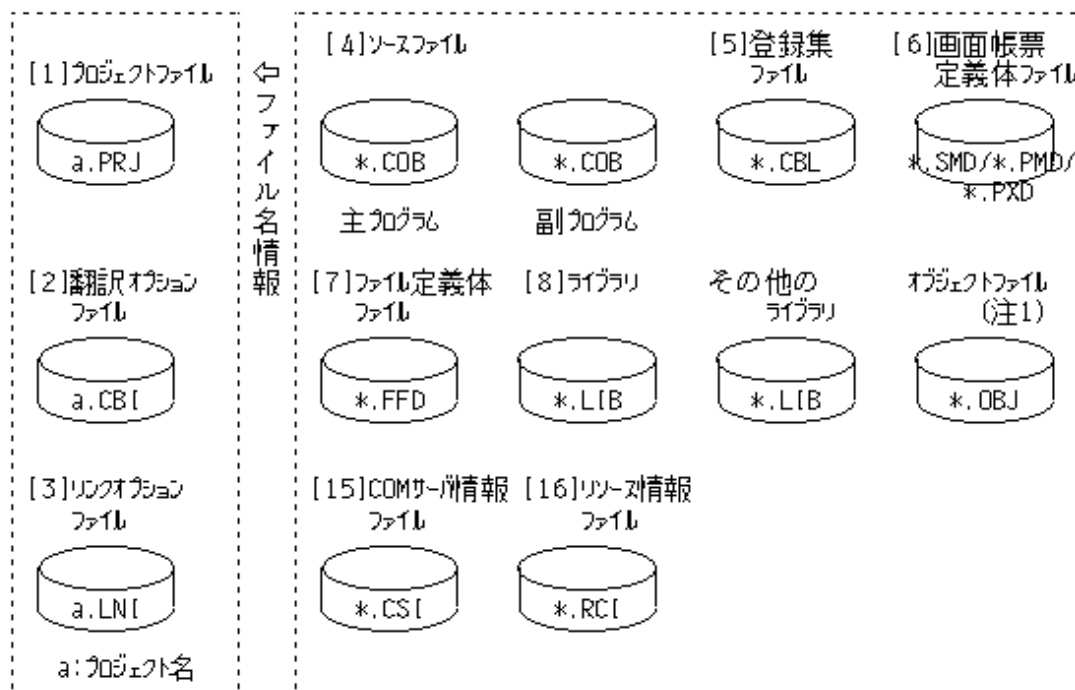
6.2 プロジェクト管理に必要な資源

ここでは、プロジェクト管理に必要なファイルについて説明します。

プロジェクト管理では、以下のファイルを使用します。

- ・ プロジェクトファイル(a.PRJ)
- ・ 翻訳オプションファイル(a.CBI)
- ・ リンクオプションファイル(a.LNI)
- ・ ソースファイル(*.COB)
- ・ 登録集ファイル(*.CBL)
- ・ 画面帳票定義体ファイル(*.SMD/*.*PMD/*.*PXD)
- ・ ファイル定義体ファイル(*.FFD)
- ・ ライブラリ(*.LIB)
- ・ COMサーバ情報ファイル(*.CSI)
- ・ リソース情報ファイル(*.RCI)
- ・ オブジェクトファイル(*.OBJ)
- ・ リポジットファイル(*.REP)
- ・ インポートライブラリ(*.LIB)
- ・ エクスポートファイル(*.EXP)
- ・ ダイナミックリンクライブラリ(*.DLL)
- ・ 実行可能ファイル(*.EXE)
- ・ MAKEファイル(a.MAK)
- ・ コマンドファイル(a.nnn)

プロジェクト管理で使用するファイルの関係を以下に示します。



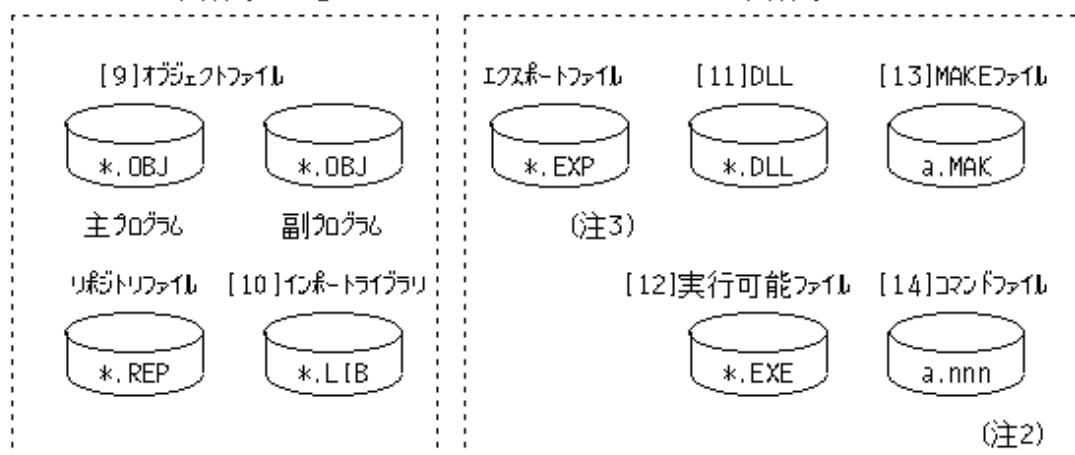
↑作成 ↓ビルド

↓ビルド

プロジェクト管理機能

↓作成 ↑ビルド

↓作成



注1: 翻訳まで行った他言語のオブジェクトファイル

注2: コマンドファイルは複数個作成されます。拡張子のnnnは数字になります。

注3: プロジェクト管理機能配下では、DLLまたは副プログラムを持つ実行可能ファイルを作成するとエクスポートファイルが生成されます。しかし、作成されたエクスポートファイルは不要となります。

表6.1 プロジェクト管理機能で使用するファイル

	ファイルの内容	ファイル名の形式	入出力	使用する条件または作成される条件
[1]	プロジェクト管理のための情報	プロジェクト名.PRJ	入力	“ビルド”または“リビルド”を行うときに、プロジェクトファイルに格納されている情報が参照されます。

	ファイルの内容	ファイル名の形式	入出力	使用する条件または作成される条件
	・ ファイルの依存関係 ・ 最終ターゲット名 ・ COBOL ソースファイル名 ・ 登録集ファイル名 ・ 各種定義体ファイル名 ・ その他必要なファイル名		出力	プロジェクト管理を行う場合、必ず作成します。
[2]	翻訳オプション	プロジェクト名.CBI	入力	“ビルド”または“リビルド”で、ソースプログラムを翻訳するときに使用されます。
			出力	プロジェクト管理で翻訳オプションを設定した場合に作成されます。
[3]	リンクオプション	プロジェクト名.LNI	入力	“ビルド”または“リビルド”で、目的プログラムをリンクするときに使用されます。
			出力	プロジェクト管理でリンクオプションを設定した場合、自動的に作成されます。
[4]	COBOL ソースプログラム	任意の名前.COB	入力	“ビルド”または“リビルド”で、ソースプログラムを翻訳するときに使用されます。
[5]	登録集原文	登録集原文名.CBL	入力	“ビルド”または“リビルド”で、ソースプログラムを翻訳するときに使用されます。(注1)
[6]	画面帳票定義体	画面帳票定義体名 .SMD/ .PMD/ .PXD/	入力	“ビルド”または“リビルド”で、ソースプログラムを翻訳するときに使用されます。(注2)
[7]	ファイル定義体	ファイル定義体名.FFD	入力	“ビルド”または“リビルド”で、ソースプログラムを翻訳するときに使用されます。(注2)
[8]	標準ライブラリ(オブジェクトコードライブラリ)	任意の名前.LIB	入力	“ビルド”または“リビルド”で、目的プログラムをリンクするときに使用されます。
[9]	目的プログラム	ソースファイル名.OBJ	入力	“ビルド”または“リビルド”で、目的プログラムをリンクするときに使用されます。
			出力	ソースプログラムの翻訳に成功したときに作成されます。
[10]	インポートライブラリ	ターゲット名.LIB	入力	動的リンク構造の実行可能ファイルを作成するときに使用されます。
			出力	“ビルド”または“リビルド”を行ってDLLまたは副プログラムを含む実行可能ファイルを作成します。ただし、実行可能ファイルを作成した場合は不要なファイルとなります。
[11]	ダイナミックリンクライブラリ(DLL)	ターゲット名.DLL	入力	動的構造の実行可能ファイルを作成するときに使用されます。
			出力	“ビルド”または“リビルド”を行ったときに作成されます。
[12]	実行可能プログラム	ターゲット名.EXE	出力	“ビルド”または“リビルド”を行ったときに作成されます。
[13]	MAKEファイル	プロジェクト名.MAK	出力	“ビルド”、“リビルド”またはプロジェクトファイルを保存したときに作成されます。

	ファイルの内容	ファイル名の形式	入出力	使用する条件または作成される条件
[14]	コマンドファイル	プロジェクト名.nnn	出力	“ビルド”、“リビルド”またはプロジェクトファイルを保存したときに作成されます。
[15]	COMサーバ情報ファイル	ターゲット名.CSI	入力	“ビルド”または“リビルド”でCOMサーバを作成するときに使用されます。
			出力	COMサーバの情報を設定したときに作成されます。
[16]	リソース情報ファイル	ターゲット名.RCI	入力	“ビルド”または“リビルド”でアプリケーションを作成するときに使用されます。
			出力	バージョン情報を設定したときに作成されます。

注1：COBOLソースプログラムにCOPY文を記述した場合の登録集ファイルの検索フォルダについては、“[3.2.1 COBOLコンパイラが使用するファイル](#)”を参照してください。なお、プロジェクト管理機能を使用した場合のカレントフォルダは、プロジェクトファイルが格納されているフォルダとなります。

注2：画面帳票定義体またはファイル定義体を検索するフォルダについては、“[3.2.1 COBOLコンパイラが使用するファイル](#)”を参照してください。なお、プロジェクト管理機能を使用した場合のカレントフォルダは、プロジェクトファイルが格納されているフォルダになります。

6.3 プロジェクト管理の手順

ここでは、プロジェクト管理を行うときの手順を説明します。

- プロジェクトマネージャウインドウの[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。
→ [ファイルを開く]ダイアログが表示されます。
- プロジェクトファイル名を指定します。[参照]“[6.4.1 プロジェクトファイルの作成](#)”
- 翻訳オプションを指定します。[参照]“[6.4.2 翻訳オプションの設定](#)”
- リンクオプションを指定します。[参照]“[6.4.3 リンクオプションの設定](#)”
- プロジェクトで作成する最終ターゲットファイルをツリーに登録します。[参照]“[6.4.4.1 最終ターゲットファイルの登録](#)”
- 最終ターゲットファイルを選択し、[編集]－[フォルダ作成]メニューから“COBOLソースファイル”を選択して、COBOLソースファイルのフォルダをツリーに登録します。[参照]“[6.4.4.7 フォルダの作成](#)”
- フォルダに対して必要なCOBOLソースファイルを[編集]メニューの“新規作成”または“追加”で登録します。[参照]“[6.4.4.2 COBOLソースファイルの登録](#)”
- 7.で登録したCOBOLソースファイルで、主プログラムの指定が必要な場合は、ファイルを選択し、[プロジェクト]－[オプション]－[主プログラム]メニューから“ウインドウ”または“コンソール”を選択します。[参照]“[6.4.4.3 主プログラムの指定](#)”
- 登録集が必要な場合は、COBOL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“登録集ファイル”を選択して、登録集ファイルのフォルダをツリーに登録します。[参照]“[6.4.4.7 フォルダの作成](#)”
- フォルダに対して必要な登録集ファイルを、[編集]メニューの“新規作成”または“追加”で登録します。[参照]“[6.4.4.4 登録集ファイルの登録](#)”
- 各種定義体ファイルを登録するために、COBOL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“定義体ファイル”を選択して定義体ファイルのフォルダをツリーに登録します。[参照]“[6.4.4.7 フォルダの作成](#)”
- フォルダに対して必要な各種定義体ファイルを、[編集]メニューの“新規作成”または“追加”で登録します。[参照]“[6.4.4.5 各種定義体ファイルの登録](#)”
- その他の必要なファイルを登録するために、最終ターゲットファイルを選択し、[編集]－[フォルダ作成]メニューから必要なフォルダを選択して新規のフォルダをツリーに登録します。[参照]“[6.4.4.7 フォルダの作成](#)”
- フォルダに対して必要なファイルを、[編集]メニューの“新規作成”または“追加”で登録します。[参照]“[6.4.4.6 その他必要なファイルの登録](#)”
- [プロジェクト]メニューから“ビルド”または“リビルド”を選択し、プロジェクトのビルド・リビルドを行います。[参照]“[6.4.5 プロジェクトのビルド・リビルド](#)”

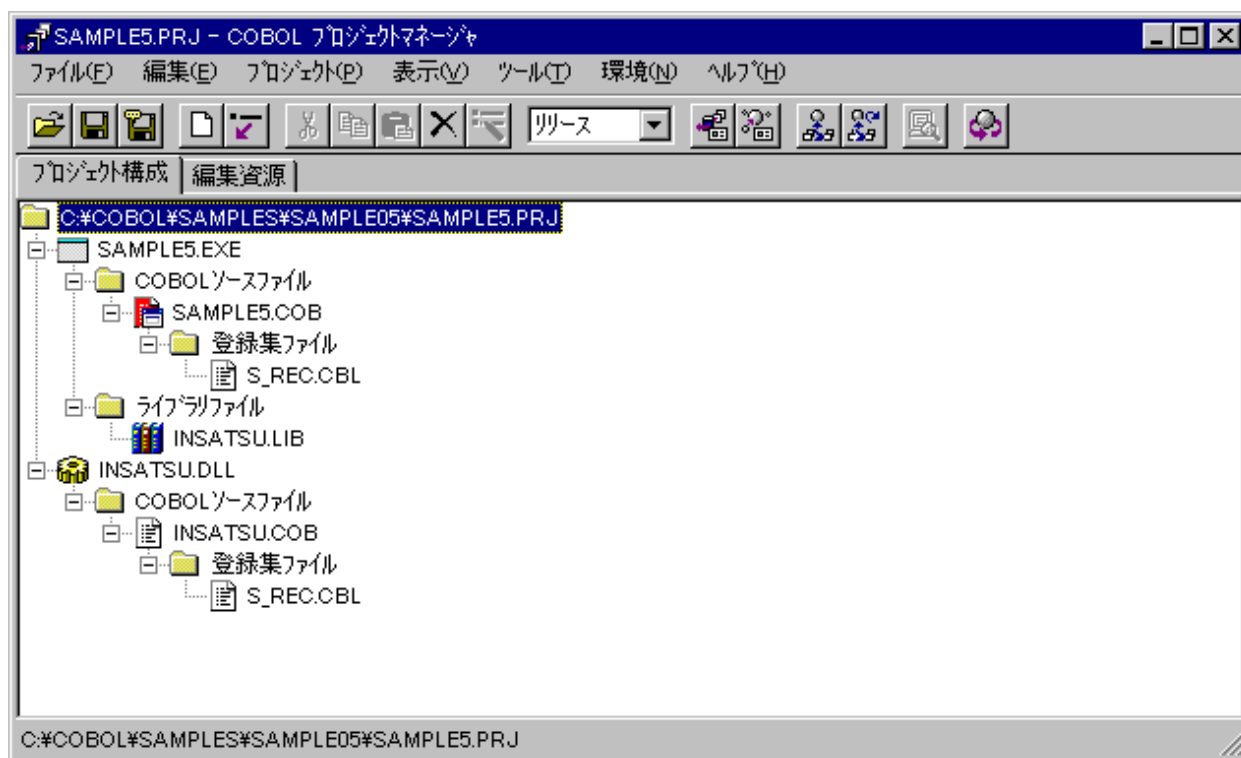
16. ビルド・リビルドの結果は、ビルダウィンドウに表示されます。エラー時は、ビルダのエラージャンプ機能を使用したソースプログラムの修正が行えます。[参照]“[6.4.6 ファイルの内容の修正](#)”
17. [プロジェクト]メニューから“実行”または“パラメタを指定して実行”を選択し、プログラムを実行します。[参照]“[6.4.7 アプリケーションプログラムの実行](#)”
18. [ファイル]メニューから“プロジェクトを閉じる”、“上書き保存”、“名前を付けて保存”または“終了”のどれかを指定することにより、プロジェクトファイルを更新します。[参照]“[6.4.4.8 プロジェクトファイルの保存](#)”

6.4 プロジェクトマネージャウィンドウ

プロジェクトマネージャウィンドウは、プロジェクトマネージャを起動すると表示されるウィンドウです。プロジェクトマネージャウィンドウは、プロジェクトに登録されたファイルの依存関係をツリー構造で表示するとともに、以下の操作を行うことができます。

- ・ プロジェクトファイルの編集(ファイルの登録、翻訳オプションの設定、リンクオプションの設定)
- ・ プロジェクトの“ビルド”および“リビルド”
- ・ 登録したファイルの編集
- ・ 作成したアプリケーションプログラムの実行

ここでは、まずプロジェクトファイルの作成方法を説明し、その後プロジェクトマネージャウィンドウを使った各種操作について説明します。



ボタン

プロジェクトを開く

プロジェクトファイルを開くときに選択します。

上書き保存

ツリービュー上の依存関係をプロジェクトファイルに保存するときに選択します。

名前を付けて保存

ツリービュー上に登録した依存関係を別のプロジェクトファイル名に保存するときに選択します。

新規作成

ツリービュー上に新しいファイル名を登録するときに選択します。

追加

ターゲット名や依存ファイル(COBOLソースファイル、登録集ファイルなど)を登録するときに選択します。

切り取り

あるターゲットに依存するファイルを別のターゲットに移動するときに選択します。

コピー

あるターゲットの依存ファイルを別のターゲットにも依存させたいときに選択します。

貼り付け

“切り取り”、“コピー”で選択されたファイルを目的の場所に貼り付けるときに選択します。

削除

ファイルをツリー表示から削除するときに選択します。

名前の変更

名前を変更するときに選択します。

翻訳オプション

翻訳オプションを設定するときに選択します。

リンクオプション

リンクオプションを設定するときに選択します。

ビルド

ビルドを行うときに選択します。

リビルド

リビルドを行うときに選択します。

実行

アプリケーションを実行するときに選択します。

デバッグ

デバッグを起動するときに選択します。

コンボボックス

ビルドモード

ビルドまたはリビルド時に、リリース(非デバッグ)版のモジュールを作成するか、デバッグ版のモジュールを作成するかを指定します。デバッグ版モジュールの作成指示がある場合は、翻訳オプションTEST、CHECK(BOUND)およびリンクオプションDEBUG、/DEBUGTYPEが強制的に指定されます。新規プロジェクト作成時はリリース版のモジュール作成指示となります。

タブ

プロジェクトマネージャウィンドウにおける資源の表示形態を切り替えます。表示形態には、ツリー形態とリスト形態の二種類があります。

プロジェクト資源

資源の関係をツリー形態で表示します。プロジェクト定義の基本操作で使います。

編集資源、プロジェクト資源、AAD配付ソース生成対象、AAD配付ソース対応

資源の一覧をリスト形態で表示します。複数の資源を同時に操作する場合に使います。

6.4.1 プロジェクトファイルの作成

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

1. プロジェクトマネージャを起動します。
→ プロジェクトマネージャウィンドウが表示されます。
2. [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。
→ [ファイルを開く]ダイアログが表示されます。
3. 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直接キー入力してください。
4. [開く]ボタンをクリックします。
→ プロジェクトファイルが作成されます。



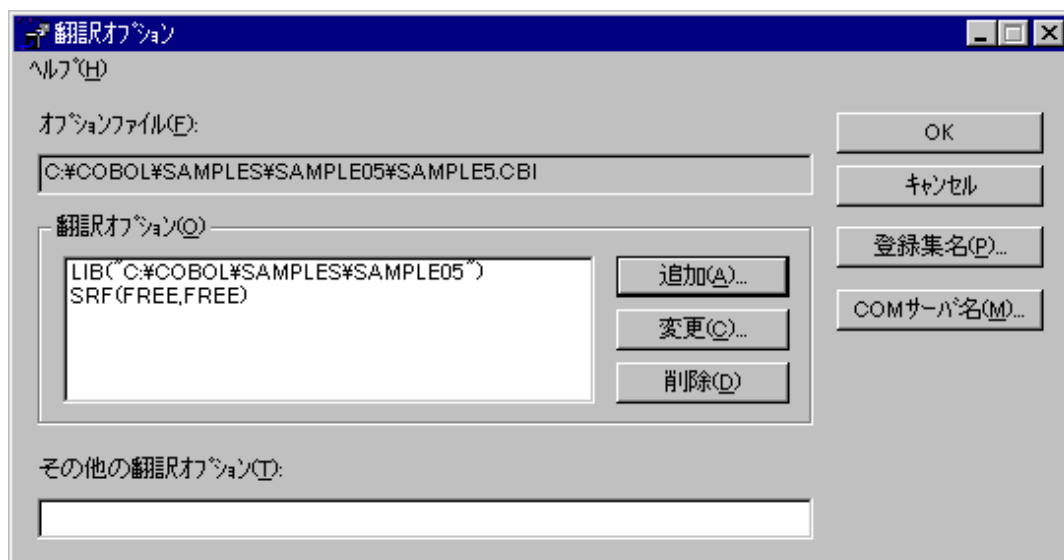
注意

- ・ プロジェクトの名前は、プロジェクトファイル名から決定します。
- ・ COBOLプロジェクトマネージャで登録されたプロジェクトファイル名の表示はすべて大文字となります。このとき大小文字の区別はされません。
- ・ プロジェクトファイルの存在するフォルダに対するアクセスが許可されていなければなりません。

6.4.2 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)に格納されます。以下に、翻訳オプションの指定方法を示します。

1. プロジェクトマネージャウィンドウの[プロジェクト]－[オプション]メニューから“翻訳オプション”を選択します。
→ [翻訳オプション]ダイアログが表示されます。[参照]“[3.5 \[翻訳オプション\]ダイアログ](#)”
2. 翻訳オプション情報を設定します。



3. [OK]ボタンをクリックします。



注意

- ・ 翻訳オプションMAINは、翻訳オプション設定ダイアログで指定しても無効となります。プロジェクトマネージャ上で主プログラムの指定を行ってください。[参照]“[6.4.4.3 主プログラムの指定](#)”

- 登録集を使用する場合、翻訳オプションLIBの設定を必ず行ってください。[参照]“[A.2.23 LIB \(登録集ファイルのフォルダの指定\)](#)”
- 複数のCOBOLソースプログラムを登録した場合、異なる翻訳オプションは設定できないので注意してください。
- ここで設定した翻訳オプションの情報は、プロジェクト内でだけ有効となります。WINCOBウィンドウから翻訳する場合には、有効になりません。
- WINCOBと異なり、プロジェクト管理機能を使用した場合のカレントフォルダは、プロジェクトファイルが存在するフォルダになります。

登録集ファイルおよび各種定義体ファイルは、翻訳オプションで指定されたフォルダ、環境変数で指定されたフォルダとともにプロジェクトフォルダが存在するフォルダが検索されることになります。

6.4.3 リンクオプションの設定

プロジェクトで管理している実行可能ファイルまたはDLLをリンクするときに有効になるリンクオプションを指定します。

設定したリンクオプションの情報は、リンクオプションファイル(プロジェクト名.LNI)に格納されます。以下に、指定方法を示します。

1. プロジェクトマネージャウィンドウの[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”を選択します。
→ [リンクオプション]ダイアログが表示されます。[参照]“[6.4.3.1 \[リンクオプション\]ダイアログ](#)”
2. 必要な情報を設定します。
3. [OK]ボタンをクリックします。



参考

翻訳オプションTESTを指定した場合、ビルド・リビルド時には、デバッグに必要なリンクオプションも自動的に設定されます。[参照]“[A.2.56 TEST \(対話型デバッグおよび診断機能の使用の可否\)](#)”

6.4.3.1 [リンクオプション]ダイアログ

リンクオプションの設定は、[リンクオプション]ダイアログで行います。主なリンクオプション(デバッグは除く)は自動的に設定されます。
[参照]“[表4.5 LINKコマンドのオプション](#)”

エディットボックス

コマンドライン

リンクオプションを指定します。

MAPファイル

MAP情報を出力するMAPファイル名を指定します。

Cランタイムライブラリ

Cランタイムライブラリとして、LIBC.LIB以外のライブラリをリンクする場合に指定します。

ラジオボタン

デバッグオプション

リンクオプションDEBUGTYPEの値をCOFF/BOTHのいずれかで指定します。

DLLのエントリオブジェクト

COBOLで作成されたオブジェクトだけでDLLを作成するか、他言語で作成したオブジェクトもあわせてDLLを作成するかを指定します。

チェックボタン

/MAP

リンクオプション/MAPの指定をします。

ボタン

OK

リンクオプションの設定が終了したときに選択します。

キャンセル

[リンクオプション]ダイアログを開く前の状態に戻し、[リンクオプション]ダイアログを閉じます。

ヘルプ

ヘルプウィンドウを開くときに選択します。

参照(コマンドライン)

コマンドラインに指定するパス名を選択します。

参照(MAPファイル)

MAP情報を出力するMAPファイル名を選択します。

参照(Cランタイムライブラリ)

リンクするCランタイムライブラリのファイル名を選択します。

リセット

Cランタイムライブラリの指定を解除し、デフォルト(LIBC.LIB)に戻すときに選択します。

6.4.4 ファイルの登録

プロジェクト管理を行うには、そのプロジェクトで管理するファイルを登録する必要があります。登録したファイル名は、プロジェクトファイルに格納されます。プロジェクトファイルには、以下のファイルの情報を登録します。

- ・ 最終ターゲットファイル
- ・ COBOLソースファイル
- ・ 登録集ファイル
- ・ 各種定義体ファイル
- ・ その他必要なファイル



注意

ソースファイルの存在するフォルダに対するアクセスが許可されていなければなりません。

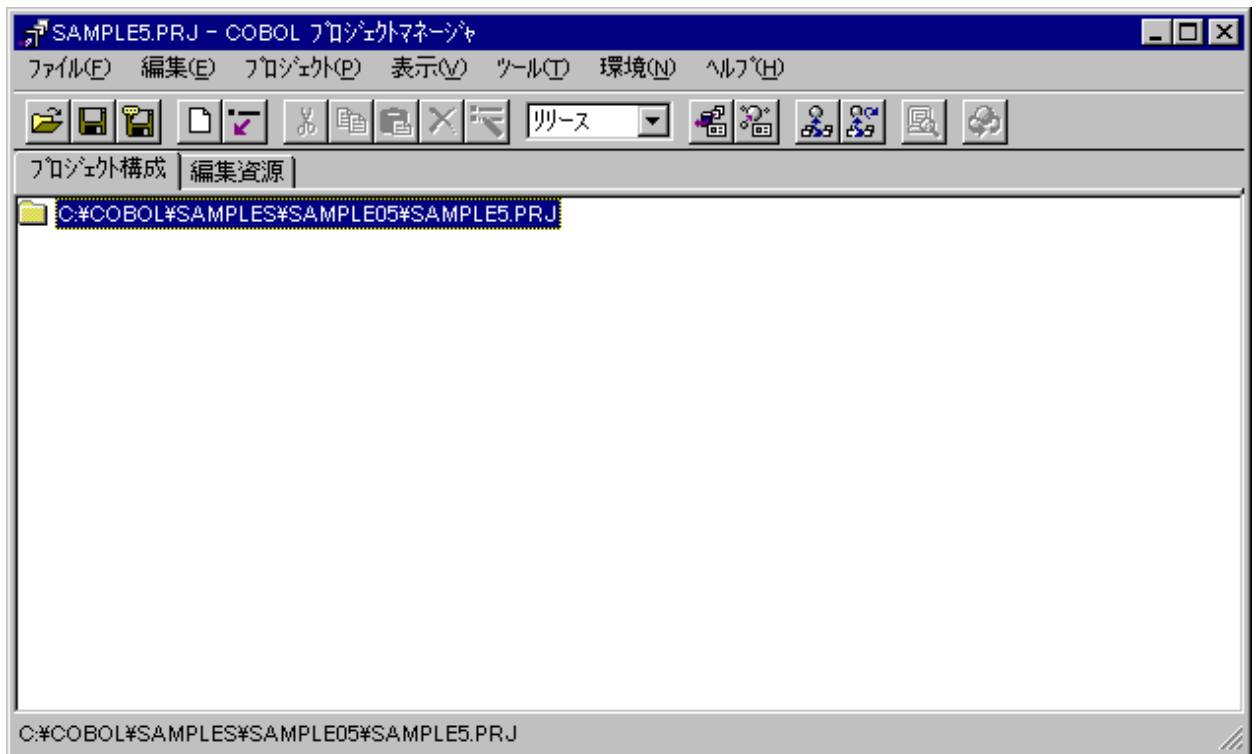
6.4.4.1 最終ターゲットファイルの登録

最終ターゲットファイルとは、実行可能ファイルまたはDLLを指します。

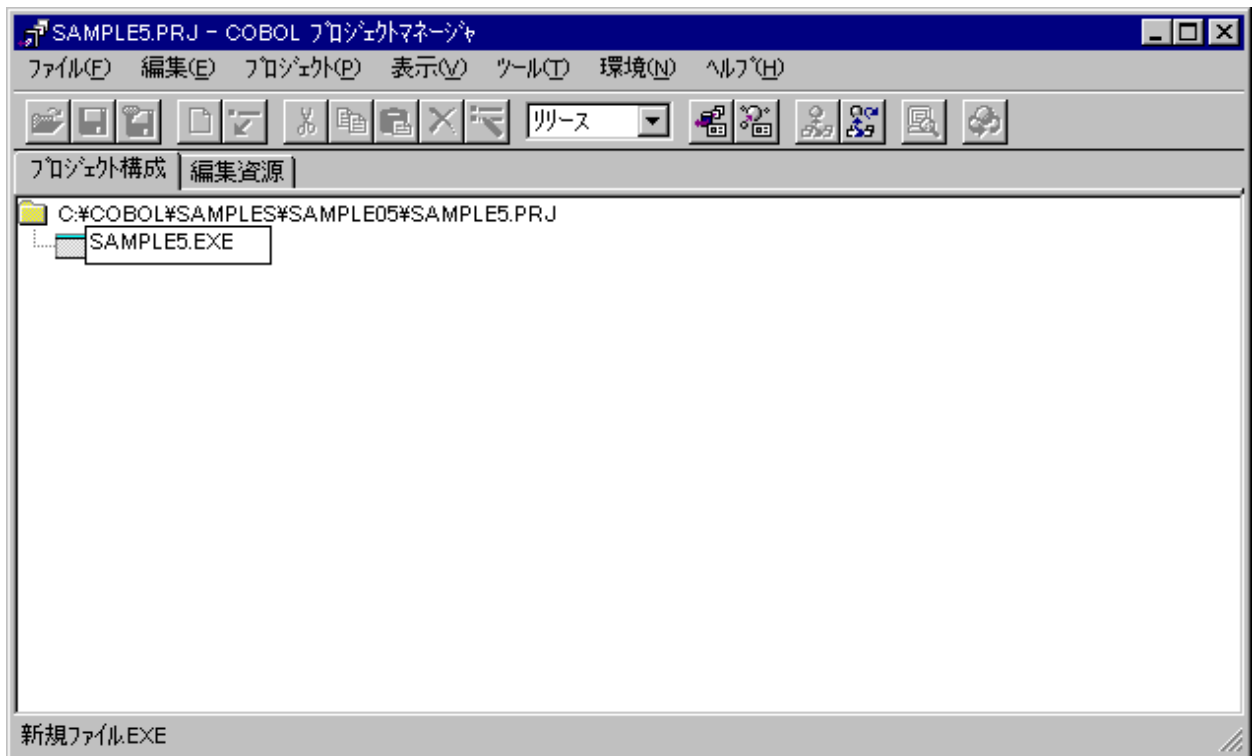
最終ターゲットファイルは、プロジェクトマネージャウィンドウ上のツリーを操作することにより登録します。最終ターゲットファイルは、複数個登録することができます。

以下に、最終ターゲットファイルの登録方法を説明します。

1. ツリー上で、プロジェクトファイルを選択します。



2. 新規に最終ターゲットファイルを作成する場合は、[編集]メニューから“新規作成”を選択します。
→ 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表示されます。
3. 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。



4. 既存の最終ターゲットファイルを登録する場合には、[編集]メニューから“追加”を選択します。
→ [ファイルを開く]ダイアログが表示されます。

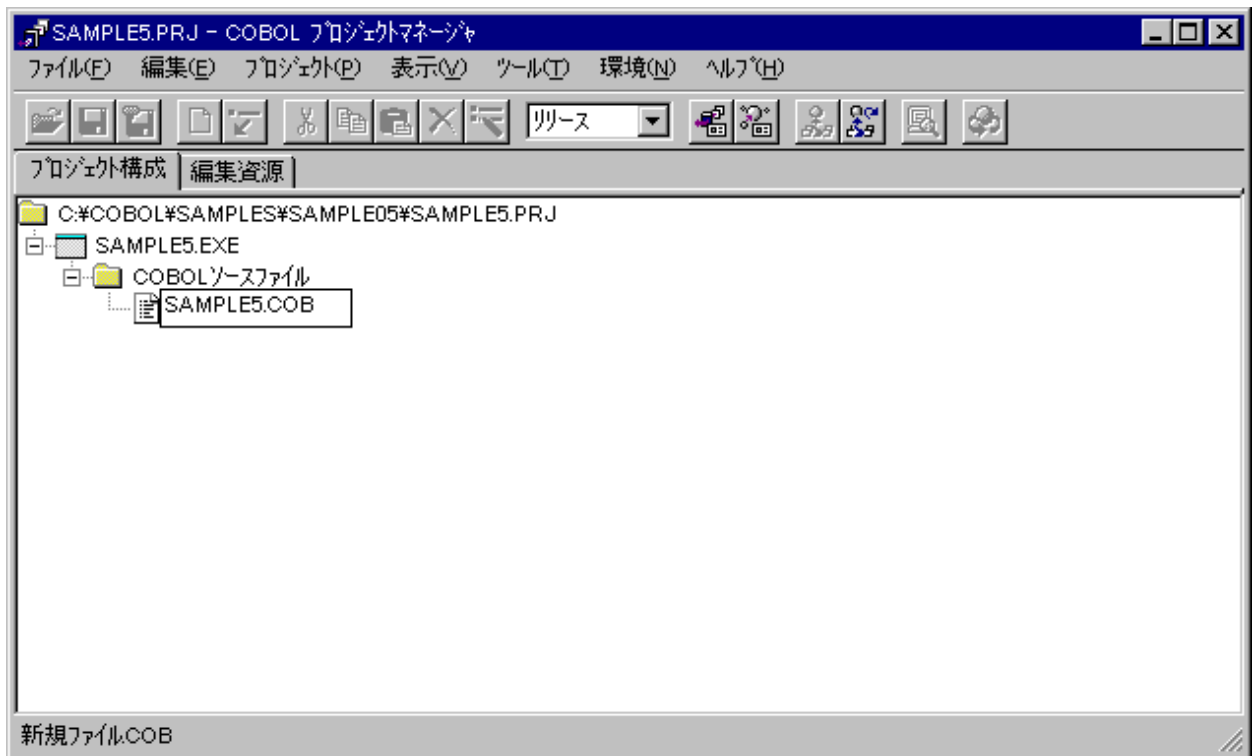
5. 最終ターゲットファイルを選択します。
→ プロジェクトファイルの配下に選択した最終ターゲットファイルが追加されます。
6. 最終ターゲットファイル名を変更する場合には、プロジェクトマネージャウィンドウの中で更新したい最終ターゲットファイルを選択します。
→ 選択した最終ターゲットファイルが反転表示されます。
7. [編集]メニューから“名前の変更”を選択します。
→ エディットコントロールが表示されます。
8. エディットコントロールにファイル名を入力します。
9. 最終ターゲットファイルの登録が終わるまで、2.から8.の作業を繰り返してください。

6.4.4.2 COBOLソースファイルの登録

COBOLソースファイルは、プロジェクトマネージャウィンドウ上のツリーを操作することによって登録します。

以下に、COBOLソースファイルの登録方法を説明します。

1. プロジェクトマネージャウィンドウ中の、COBOLソースファイルを追加する最終ターゲットファイルに“COBOLソースファイル”フォルダが存在しない場合、フォルダを作成します。[参照]“[6.4.4.7 フォルダの作成](#)”
2. “COBOLソースファイル”フォルダを選択します。
3. 新規にCOBOLソースファイルを作成する場合には、[編集]メニューから“新規作成”を選択します。
→ フォルダの配下にエディットコントロールが表示されます。
4. エディットコントロールにCOBOLソースファイル名を入力します。
5. 既存のCOBOLソースファイルを登録する場合には、[編集]メニューから“追加”を選択します。
→ [ファイルを開く]ダイアログが表示されます。
6. COBOLソースファイルを選択します。
→ フォルダの配下に選択したCOBOLソースファイルが追加されます。



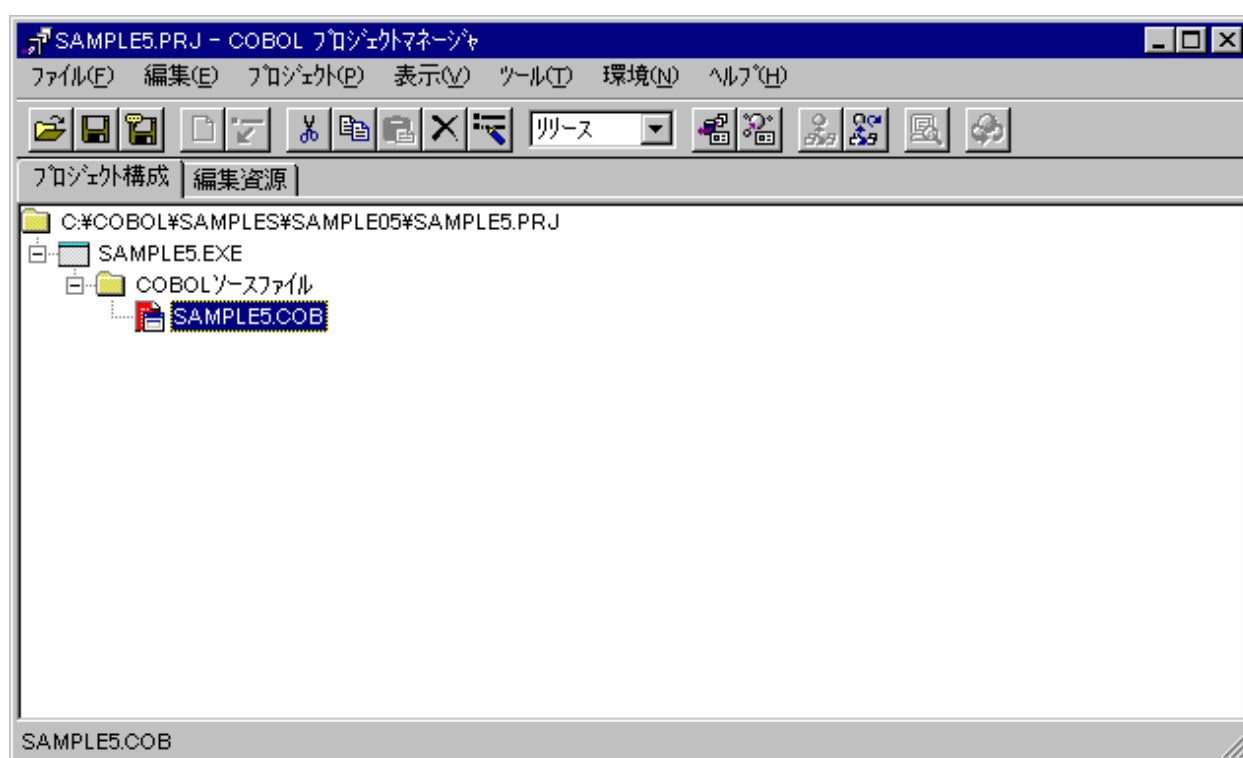
7. 新規作成または既存で登録したCOBOLソースファイルが主プログラムの場合は、主プログラムの指定を行ってください。[参照]“[6.4.4.3 主プログラムの指定](#)”

8. COBOLソースファイル名を更新する場合には、プロジェクトマネージャウィンドウの中で変更したいCOBOLソースファイルを選択します。
→ 選択したCOBOLソースファイルが反転表示されます。
9. [編集]メニューから“名前の変更”を選択します。
→ エディットコントロールが表示されます。
10. エディットコントロールにファイル名を入力します。
11. COBOLソースファイルの登録が終わるまで、3.から10.までの作業を繰り返してください。
12. 最終ターゲットファイルごとに1.から11.までの作業を繰り返してください。

6.4.4.3 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。以下に、主プログラムの指定方法を説明します。

1. プロジェクトマネージャウィンドウに表示されているCOBOLソースファイルの中から、主プログラムとなるファイルを選択します。
2. [プロジェクト]－[オプション]－[主プログラム]メニューから“ウィンドウ”または“コンソール”を選択します。
→ 選択したファイルの小アイコンが、ウィンドウメイン型またはコンソールメイン型に変更されます。



注意

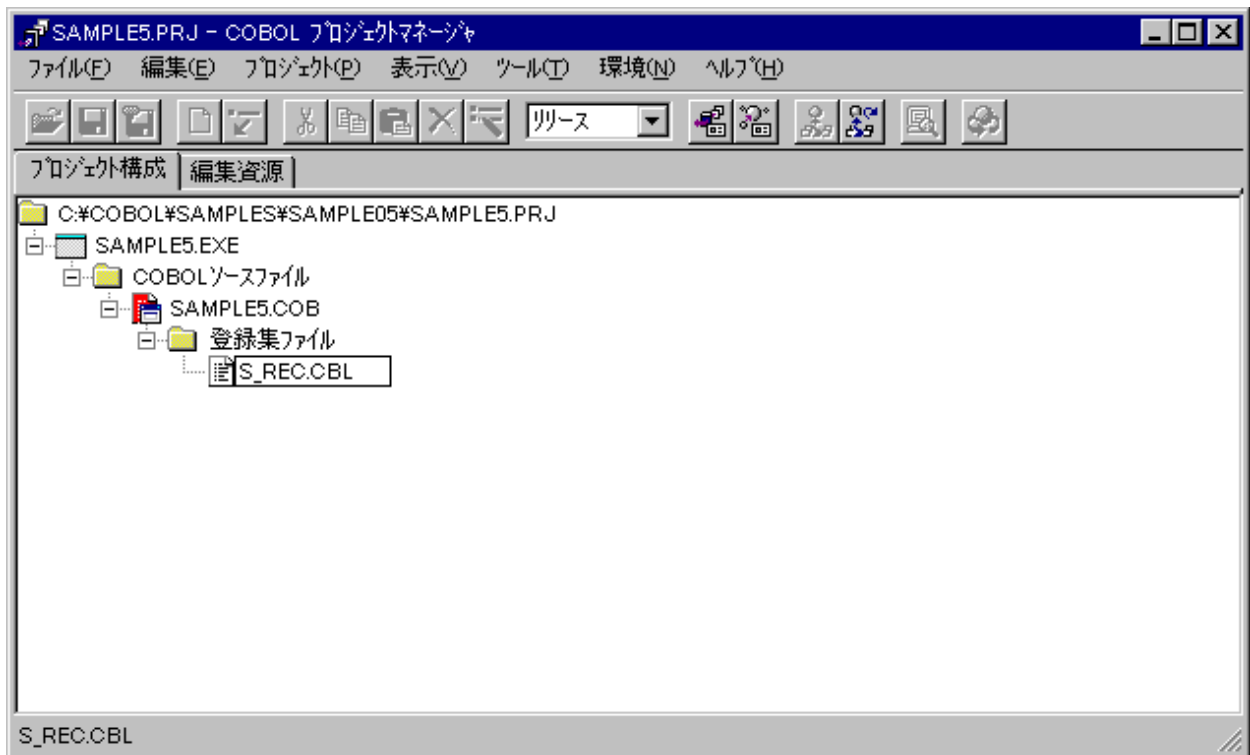
- ・すでに他のCOBOLソースファイルが主プログラムとして指定されている場合、新しく指定したCOBOLソースファイルが主プログラムとみなされ、元のCOBOLソースファイルの主プログラム指定は解除されます。
- ・[主プログラム]メニューから“ウィンドウ”を選択した場合、翻訳オプションMAIN(WINMAIN)を指定したのと同じ意味になります。また、“コンソール”を選択した場合は、MAIN(MAIN)を選択したのと同じ意味になります。翻訳オプションMAIN(WINMAIN)とMAIN(MAIN)の相違については、“[11.1.3 コンソールウィンドウを使ったデータの入出力](#)”を参照してください。

6.4.4.4 登録集ファイルの登録

登録集ファイルは、プロジェクトマネージャウィンドウ上のツールを操作することによって登録します。以下に、登録集ファイルの登録方法を説明します。

1. プロジェクトマネージャウィンドウ中の、登録集ファイルを追加するCOBOLソースファイルに“登録集ファイル”フォルダが存在しない場合、フォルダを作成します。[参照]“[6.4.4.7 フォルダの作成](#)”
2. “登録集ファイル”フォルダを選択します。

登録に関する以降の操作は、COBOLソースファイルの登録と同様の操作になります。[参照]“[6.4.4.2 COBOLソースファイルの登録](#)”



6.4.4.5 各種定義体ファイルの登録

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
- ・ サブスキーマ定義ファイル

定義体ファイルは、プロジェクトマネージャウィンドウ上のツリーを操作することによって登録します。ツリー上の操作では、“定義体ファイル”フォルダを使用すること以外は、登録集ファイルの登録と同様の操作になります。[参照]“[6.4.4.4 登録集ファイルの登録](#)”

6.4.4.6 その他必要なファイルの登録

その他の必要なファイルは、プロジェクトマネージャウィンドウ上のツリーを操作することによって登録します。ツリー上の操作では、ファイルの種類に応じたフォルダを使用すること以外は、COBOLソースファイルの登録と同様の操作になります。[参照]“[6.4.4.2 COBOLソースファイルの登録](#)”

プロジェクトに登録するファイルは、プログラム構造を把握して必要なファイルを指定する必要があります。[参照]“[4.3 プログラム構造](#)”

プロジェクトに登録できるファイルを以下に示します。

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

前バージョンレベルのコンパイラで作成したオブジェクトファイルは、リンクできない場合があります。必ず、COBOLソースファイルを再翻訳して、オブジェクトファイルを作成してください。

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。



注意

- インポートライブラリは、必ず“DLL名.LIB”という名前で指定してください。
- DLLを作成する場合、インポートライブラリ(ターゲット名.LIB)が作成されます。この名前のファイルがすでに存在する場合、ファイルは置き換えられます。
- 実行可能ファイルを作成する場合、“実行可能プログラム名.LIB”という名前でインポートライブラリが作成されることがあるため、同一のパス上に同一名のDLLを作成しないでください。

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。

モジュール定義ファイル

[参照]“[6.5 モジュール定義ファイル](#)”

6.4.4.7 フォルダの作成

フォルダとは、プロジェクト上で同種の資源をまとめて扱うための論理的な入れ物です。

フォルダの種類

フォルダの種類およびプロジェクトのツリー上での指定可能な位置を“[表6.2 フォルダの種類と指定可能位置](#)”に示します。

表6.2 フォルダの種類と指定可能位置

種類	指定可能な位置
COBOL ソースファイル	最終ターゲットファイル配下
オブジェクトファイル	最終ターゲットファイル配下
ライブラリファイル	最終ターゲットファイル配下
IDEファイル	最終ターゲットファイル配下
モジュール定義ファイル	最終ターゲットファイル配下 ライブラリファイル配下
ターゲットリポジトリファイル (注1)	COBOL ソースファイル配下
依存リポジトリファイル (注1)	COBOL ソースファイル配下
相互参照依存リポジトリファイル (注2)	COBOL ソースファイル配下
登録集ファイル	COBOL ソースファイル配下
定義体ファイル	COBOL ソースファイル配下

注1：“ターゲットリポジトリファイル”フォルダおよび“依存リポジトリファイル”フォルダの詳細については、“[17.4.3.1 リポジトリファイル](#)”を参照してください。

注2：“相互参照依存リポジトリファイル”フォルダの詳細については、“[17.4.5.2 相互参照クラスの翻訳処理](#)”を参照してください。

フォルダの作成方法

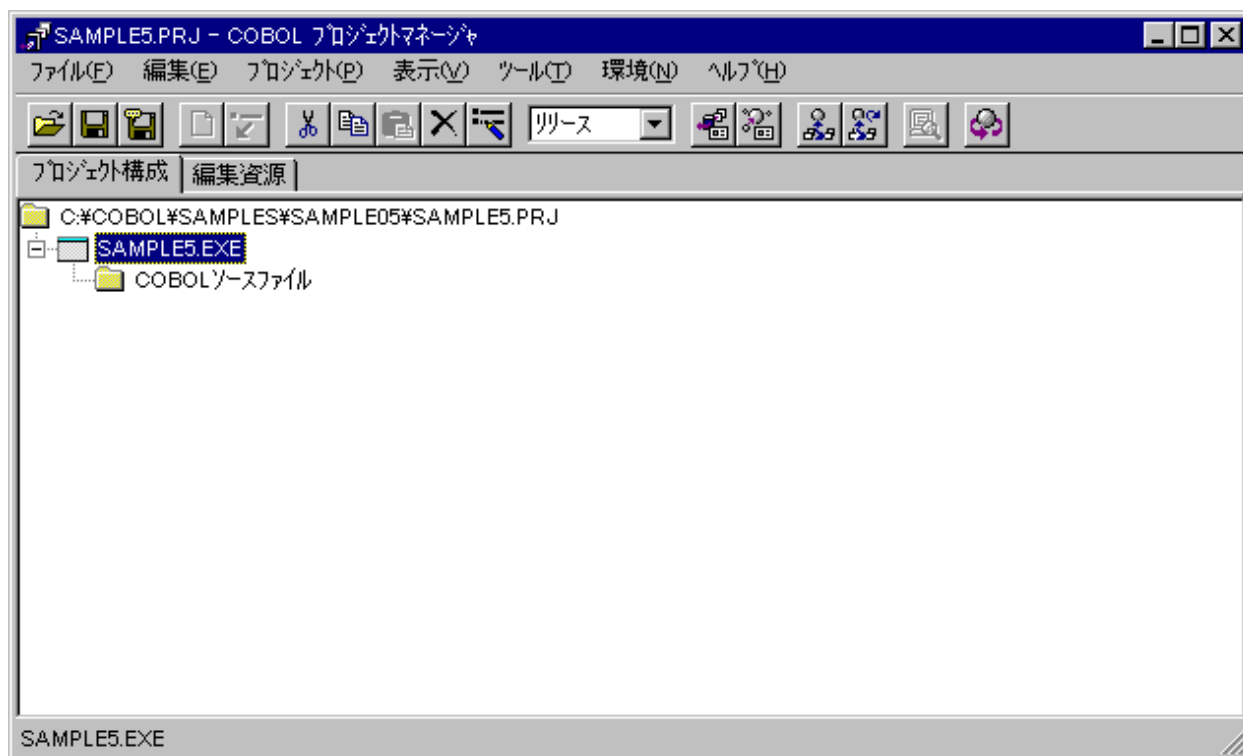
以下に、フォルダの作成方法を説明します。

1. フォルダを作成したい最終ターゲットファイルまたはCOBOLソースファイルを選択します。

2. [編集]メニューから“フォルダ作成”を選択します。
→ フォルダの一覧が表示されます。



3. フォルダ一覧の中から、必要な種類のフォルダを選択します。





注意

対象の最終ターゲットファイルまたはCOBOLソースファイルに、すでにフォルダが登録されている場合、同じ種類のフォルダは作成できません。

6.4.4.8 プロジェクトファイルの保存

プロジェクトマネージャ上に表示されている依存関係は、[ファイル]メニューの以下の項目のどれかを選択することによって、プロジェクトファイルに保存されます。

- ・ プロジェクトを閉じる
- ・ 上書き保存
- ・ 名前を付けて保存
- ・ 終了

以下に、動作の違いについて説明します。

選択項目	動作
プロジェクトを閉じる	プロジェクトファイルに対して変更があった場合、プロジェクトファイルの内容の更新を確認した上で、プロジェクトマネージャウィンドウの表示内容を空白にします。
上書き保存	プロジェクトマネージャウィンドウ上の依存関係を、ウィンドウ上に表示されているプロジェクトファイルに保存します。
名前を付けて保存	プロジェクトマネージャウィンドウ上の依存関係を、利用者が指定したプロジェクトファイルに保存します。プロジェクトマネージャウィンドウ上のプロジェクトファイル名が変更されます。
終了	プロジェクトファイルに対して変更があった場合、プロジェクトファイルの内容の更新を確認した上で、プロジェクトマネージャを終了します。

6.4.5 プロジェクトのビルド・リビルド

プロジェクトに登録したファイルを更新したり、翻訳オプションやリンクオプションを変更した場合、プロジェクトを“ビルド”または“リビルド”します。プロジェクトの“ビルド”および“リビルド”は、プロジェクトマネージャウィンドウの[プロジェクト]メニューから“ビルド”または“リビルド”を選択すると実行されます。翻訳エラーの修正は、ビルダウィンドウからエラージャンプ機能を利用して行います。

以下に、“ビルド”を実施した場合に行われる処理を示します。“リビルド”を実施した場合には、登録されている全プログラムの再翻訳および再リンクが行われます。

更新した内容	ビルド
COBOL ソースプログラム	(1) 更新したプログラムの再翻訳 (2) 更新したプログラムと依存関係にあるプログラムの再リンク
登録集ファイル	(1) 更新した登録集を取り込んでいる全プログラムの再翻訳 (2) (1)で再翻訳したプログラムと依存関係にあるプログラムの再リンク
翻訳オプション	全プログラムの再翻訳および再リンク
リンクオプション	全プログラムの再リンク



注意

前バージョンレベルで作成したプロジェクトを使用する場合、必ず最初にリビルドを行ってください。

6.4.5.1 デバッグモジュールの作成方法

デバッグモジュールとは、対話型デバッガや診断機能を使用したデバッグ作業が可能な実行可能ファイルおよびダイナミックリンクライブラリのことをいいます。

プロジェクトマネージャでは、[ビルドモード]コンボボックスで“デバッグ”を選択してビルドまたはリビルドすることにより、デバッグモジュールを作成することができます。



- ・ 翻訳オプションでOPTIMIZEを指定してデバッグモジュールを作成した場合、対話型デバッガではこのデバッグモジュールのデバッグ作業を行うことはできません。
- ・ [ビルドモード]コンボボックスの設定を変更した場合、ビルドを指定してもプロジェクト内のすべてのCOBOLソースファイルが翻訳されます(リビルド指定時と同じ動作になります)。
- ・ デバッグモジュール作成時は、同一プロジェクトから作成されたリリース指定のモジュールを置き換えます。逆に、リリース指定でモジュールを作成する場合も、デバッグモジュールを置き換えます。

6.4.6 ファイルの内容の修正

プロジェクトに登録したファイルの内容(COBOLソースプログラム、登録集原文およびモジュール定義文)をエディタを使って修正することができます。プロジェクトマネージャウィンドウに表示されているファイル名をダブルクリックすると、エディタの編集画面が表示されます。

6.4.7 アプリケーションプログラムの実行

作成したアプリケーションプログラムを、プロジェクトマネージャウィンドウを使って実行することができます。以下に、実行の手順を説明します。

1. 実行するアプリケーションを選択します。
アプリケーションが複数登録されているときは、プロジェクトマネージャウィンドウ中から実行するアプリケーションを選択します。
2. 実行時パラメタがある場合は、プロジェクトマネージャウィンドウの[プロジェクト]メニューから“パラメタを指定して実行”を選択し、エディットボックスにパラメタを指定します。
3. 実行時パラメタがない場合は、プロジェクトマネージャウィンドウの[プロジェクト]メニューから“実行”を選択します。



COBOLのアプリケーションの場合、必要な実行環境情報は、あらかじめ実行環境設定ツールなどで設定し、実行を開始してください。
[参照]“[5.6 実行環境設定ツール](#)”

6.4.8 プリコンパイラの登録とプリコンパイルの実行

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。ここでは、プリコンパイラのプロジェクトマネージャ上での扱い方について説明します。

なお、プリコンパイラによっては、INSDBINFコマンドとの組み合わせで注意が必要な場合があります。詳細については、“[I.2 INSDBINF コマンド](#)”を参照してください。

6.4.8.1 資産とツール

プリコンパイラを扱う場合の資産とツールの関係は、下図のようになります。

図6.1 INSDBINFを使用しない場合

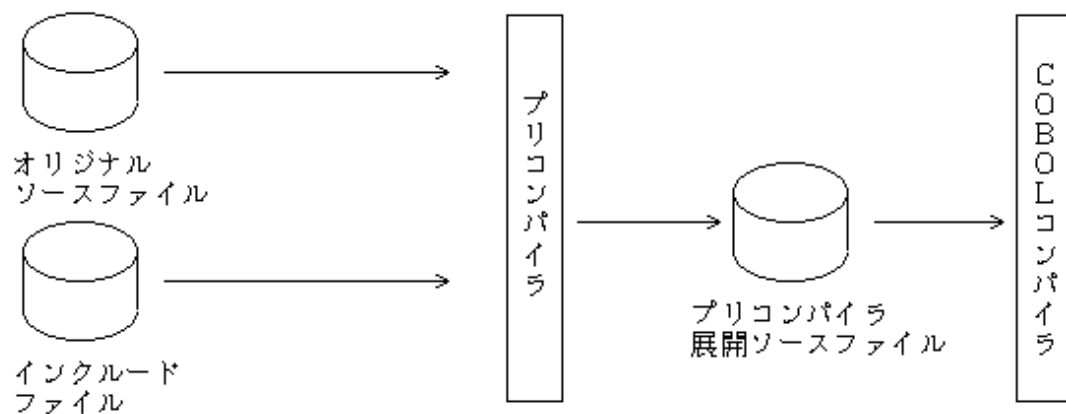
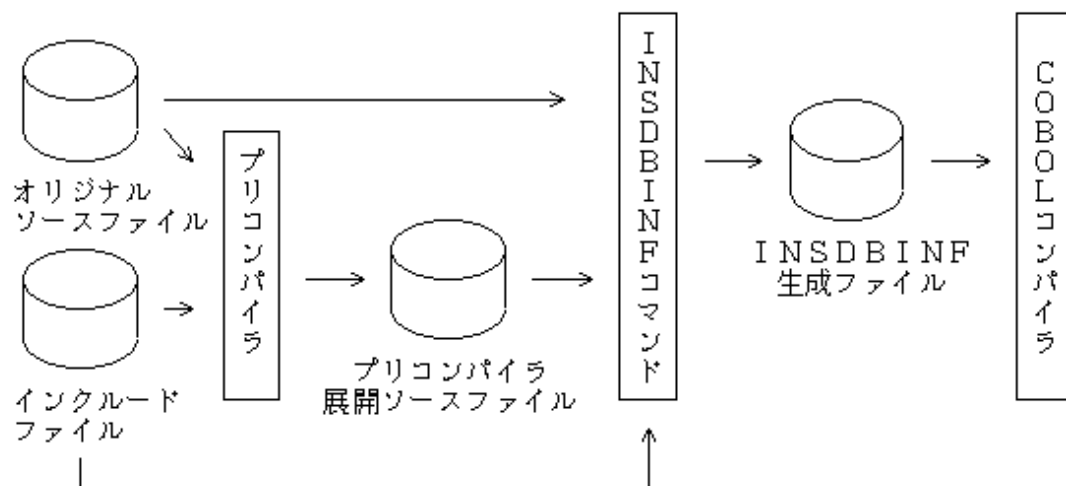


図6.2 INSDBINFを使用する場合



資産

オリジナルソースファイル

利用者が作成したソースファイルです。ただし、多段プリコンパイルを行う場合は、前の段のプリコンパイラまたはINSDBINFコマンドが生成したファイルとなります。

インクルードファイル

オリジナルソースファイルをプリコンパイルするときに必要となるインクルードファイルです。
インクルードファイルについては、“[6.4.8.6 インクルードファイル](#)”を参照してください。

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をプリコンパイルすることにより生成されるソースファイルです。

INSDBINF生成ファイル

INSDBINFコマンドにより生成されるファイルです。

ツール

INSDBINFコマンド

INSDBINFコマンドについては、“[L.2 INSDBINFコマンド](#)”を参照してください。

6.4.8.2 プロジェクト管理

プロジェクト管理では、プリコンパイラを扱うために、以下の4種類のファイルと3種類のフォルダが用意されます。

ファイル

入力ファイル

プリコンパイラに入力するオリジナルソースファイルです。
入力ファイルには、必ず拡張子を指定してください。

出力ファイル

プリコンパイラが生成するソースファイルです。COBOLコンパイラの入力ソースファイルとなるファイルです。
出力ファイルには、必ず拡張子を指定してください。

中間ファイル

INSDBINFコマンドを使用する場合にプリコンパイラが生成するソースファイルです。

インクルードファイル

プリコンパイル時に必要となるインクルードファイルです。

フォルダ

各種プリコンパイラフォルダ

各種プリコンパイラを使用するために、[プリコンパイラ設定]ダイアログを使用して利用者が作成するフォルダです。詳細については、“[6.4.8.4 \[プリコンパイラ設定\]ダイアログ](#)”を参照してください。
このフォルダには、プリコンパイルコマンド、翻訳オプションなどの情報が保持されます。

“INSDBINF”フォルダ

INSDBINFコマンドを使用することを示すフォルダです。“INSDBINF”フォルダも、プリコンパイラ設定ダイアログを使用することにより作成されます。
このフォルダには、INSDBINFコマンドのオプション情報が保持されます。

“インクルードファイル”フォルダ

インクルードファイルを管理するためのフォルダです。

6.4.8.3 プロジェクト管理の手順

プリコンパイラを使用するプロジェクトは、下図のようになります。

図6.3 INSDBINFコマンドを使用しない場合

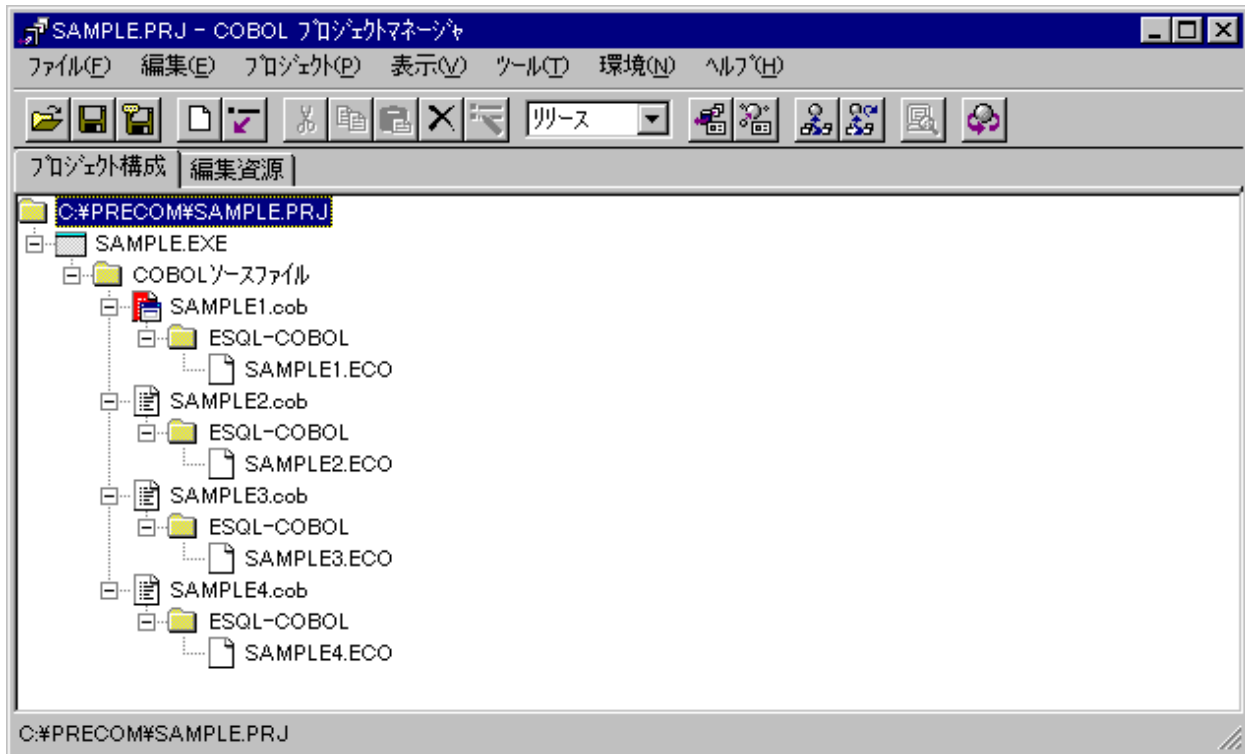
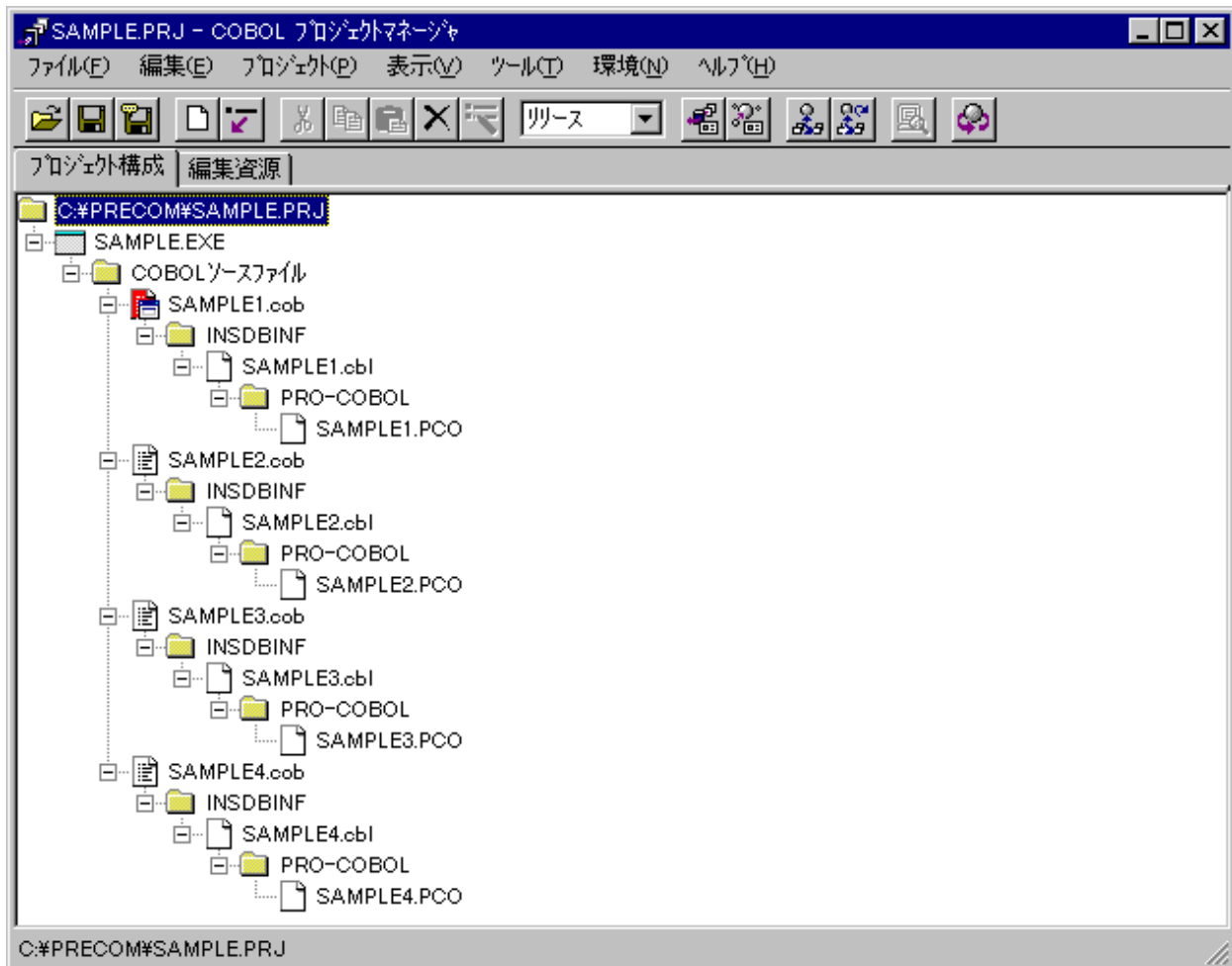


図6.4 INSDBINFコマンドを使用する場合



プロジェクトにプリコンパイラを登録する場合は、以下のような手順になります。

プロジェクトにプリコンパイラを新規に登録する場合

1. プロジェクトに、最終ターゲットファイル、プリコンパイルの必要のないCOBOLソースファイルを登録します。
2. “COBOLソースファイル”フォルダを選択して、[編集]メニューから“プリコンパイラ”を選択します。
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。
詳細については、“[6.4.8.4 \[プリコンパイラ設定\]ダイアログ](#)”を参照してください。
4. 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。
7. “ビルド”または“リビルド”を実行します。

プロジェクトに登録済みのプリコンパイラを更新する場合

1. プロジェクトに登録されているプリコンパイラの入出力ファイル、INSDBINF入力ファイル、プリコンパイラフォルダ、INSDBINFフォルダのいずれかを選択して、[編集]メニューから“プリコンパイラ”を選択します。
2. 表示された[プリコンパイラ設定]ダイアログに対して、必要な情報を変更します。
コマンドページで、プリコンパイラ定義名、コマンド、コマンドパラメタを更新します。
入力ページで入力ファイル名を更新します。入力ファイルは1つ設定できます。
出力ページで出力ファイル名を更新します。
INSDBINFコマンドを使用する場合には、INSDBINFページで中間ファイル名、INSDBINFオプションを更新します。
詳細については、“[6.4.8.4 \[プリコンパイラ設定\]ダイアログ](#)”を参照してください。
3. プロジェクトマネージャの画面上に、更新された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。
4. “ビルド”または“リビルド”を実行します。

6.4.8.4 [プリコンパイラ設定]ダイアログ

プリコンパイラおよびINSDBINFコマンドに関する種々の指定は、[プリコンパイラ設定]ダイアログで行います。

[プリコンパイラ設定]ダイアログは、[編集]メニューから“プリコンパイラ”を選択することにより表示されます。

プリコンパイラ設定

コマンド | 入力 | 出力 | INSDBINF

プリコンパイラ定義名(D): プリコンパイラ

コマンド(C): 参照(B)...

コマンドパラメータ(P):

このページでは、使用するプリコンパイラのコマンドとコマンドパラメータを「プリコンパイラ定義名」と関連付けて定義します。

コマンドとコマンドパラメータの内容が変更されると、定義済みの同名のプリコンパイラ定義名のコマンドとコマンドパラメータの内容も、指定された値で更新されます。

OK キャンセル ヘルプ

ボタン

OK

プリコンパイラの設定が終了したときに選択します。

キャンセル

[プリコンパイラ設定]ダイアログが開く前の状態に戻し、[プリコンパイラ設定]ダイアログを閉じるときに選択します。

ヘルプ

ヘルプウィンドウを開くときに選択します。

6.4.8.4.1 新規登録の場合

[プリコンパイラ設定]ダイアログの各ページについて、以下に説明します。

コマンドページ

プリコンパイラ設定

コマンド | 入力 | 出力 | INSDBINF

プリコンパイラ定義名(D): PRO-COBOL

コマンド(C): PROCOB.EXE 参照(B)...

コマンドパラメータ(P): INAME=%INFILE% ONAME=%OUTFILE%

このページでは、使用するプリコンパイラのコマンドとコマンドパラメータを「プリコンパイラ定義名」と関連付けて定義します。

コマンドとコマンドパラメータの内容が変更されると、定義済みの同名のプリコンパイラ定義名のコマンドとコマンドパラメータの内容も、指定された値で更新されます。

OK キャンセル ヘルプ

エディットボックス

プリコンパイラ定義名

作成する各種プリコンパイラフォルダのフォルダ名を指定します。
ドロップダウンリストには、以前に指定したプリコンパイラ定義名が表示されます。
1つのプロジェクトの中では、プリコンパイラ定義名とコマンド、コマンドパラメタは関連付けて定義されます。このため、指定したプリコンパイラ定義名と同名の定義名がプロジェクトに登録済みであった場合に、コマンド、コマンドパラメタを変更すると、登録済みのプリコンパイラのコマンド、コマンドパラメタも変更されます。

コマンド

プリコンパイラの実行形式ファイル名を指定します。
ドロップダウンリストには、以前に指定したコマンドが表示されます。

コマンドパラメタ

プリコンパイルコマンドのパラメタ文字列を指定します。
ドロップダウンリストには、以前に指定したコマンドパラメタが表示されます。
プリコンパイルコマンドのパラメタ文字列として“入力ファイル名”、“出力ファイル名”、“出力ファイルフォルダ”を指定する場合はマクロを使用してください。マクロの詳細については、“[6.4.8.4.3 マクロ](#)”を参照してください。

ボタン

参照

プリコンパイルコマンドのファイル名を選択します。

入力ページ



リストボックス

入力ファイル名一覧

プリコンパイラに入力するファイル名の一覧を表示します。

ボタン

追加

入力ファイル名一覧にファイル名を追加するためのダイアログボックスを起動します。

変更

入力ファイル名一覧中の選択されているファイル名を変更するためのダイアログボックスを起動します。
入力ファイル名一覧中のファイル名を1つ選択している場合に使用可能です。

削除

入力ファイル名一覧中の選択されているファイル名を削除します。

出力ページ

プリコンパイラ設定

コマンド 入力 出力 INSDBINF

出力ファイル名(O): 参照(B)...

出力ファイル拡張子(E):

出力ファイルフォルダ(F): 参照(R)...

OK キャンセル ヘルプ

エディットボックス

出力ファイル名

プリコンパイラまたはINSDBINFコマンドにより生成されるファイル名を指定します。
入力ページの入力ファイル名一覧にファイル名を1つ指定した場合に指定します。

出力ファイル拡張子

プリコンパイラまたはINSDBINFコマンドにより生成されるファイル名の拡張子を指定します。
入力ページの入力ファイル名一覧にファイル名を複数指定した場合に指定します。

出力ファイルフォルダ

プリコンパイラまたはINSDBINFコマンドにより生成されるファイルを格納するためのフォルダ名を指定します。
指定を省略した場合、プロジェクトファイルのフォルダに格納されます。
入力ページの入力ファイル名一覧にファイル名を複数指定した場合に指定します。

ボタン

参照(出力ファイル名)

プリコンパイラまたはINSDBINFコマンドが出力するファイル名を選択します。

参照(出力ファイルフォルダ名)

プリコンパイラまたはINSDBINFコマンドが出力するファイルを格納するフォルダ名を選択します。

INSDBINFページ

プリコンパイラ設定

コマンド 入力 出力 INSDBINF

☒ INSDBINFコマンドを使用する(N)

中間ファイル名(M): 参照(B)...

中間ファイル拡張子(E):

中間ファイルフォルダ(F): 参照(R)...

INSDBINF オプション(O):

OK キャンセル ヘルプ

チェックボックス

INSDBINFコマンドを使用する

プリコンパイラ後に、INSDBINFコマンドを使用するか否かを指定します。
INSDBINFコマンドを使用しない場合、INSDBINFページ内のエディットボックスは入力不可となります。

エディットボックス

中間ファイル名

プリコンパイラにより作成されるファイル名を指定します。
入力ページの入力ファイル名一覧にファイル名を1つ指定した場合に指定します。

中間ファイル拡張子

プリコンパイラにより作成されるファイル名の拡張子を指定します。
入力ページの入力ファイル名一覧にファイル名を複数指定した場合に指定します。

中間ファイルフォルダ

プリコンパイラにより作成されるファイルを格納するフォルダ名を指定します。
指定を省略した場合、プロジェクトファイルのフォルダに格納されます。
入力ページの入力ファイル名一覧にファイル名を複数指定した場合に指定します。

INSDBINFオプション

INSDBINFコマンドに対するオプションを指定します。
1つのプロジェクトの中では、INSDBINFフォルダとINSDBINFオプションは関連づけて定義されます。このため、INSDBINFフォルダがプロジェクトに登録済みであった場合にINSDBINFオプションを変更すると、登録済みのINSDBINFオプションも変更されます。

ボタン

参照(中間ファイル名)

プリコンパイラが出力するファイル名を選択します。

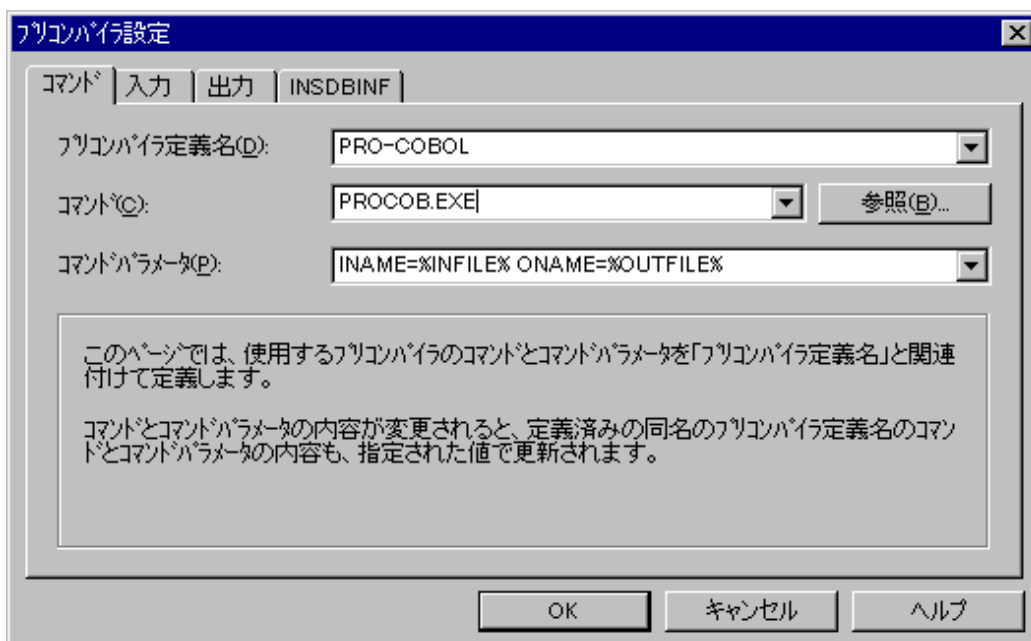
参照(中間ファイルフォルダ名)

プリコンパイラが出力するファイルを格納するフォルダ名を選択します。

6.4.8.4.2 更新登録の場合

〔プリコンパイラ設定〕ダイアログの各ページについて、以下に説明します。

コマンドページ



新規登録の場合と同じ手順で設定を行います。

入力ページ



リストボックス

入力ファイル名一覧

プリコンパイラに入力するファイル名の一覧を表示します。更新登録のとき表示される入力ファイル名は1つとなります。

ボタン

変更

入力ファイル名一覧中の選択されているファイル名を変更するためのダイアログボックスを起動します。

なお、更新登録の場合、[追加]ボタンと[削除]ボタンは使用できません。

出力ページ

The screenshot shows the 'File Conversion Settings' dialog box with the 'Output' tab selected. The dialog has four tabs: 'Command', 'Input', 'Output', and 'INSDBINF'. The 'Output' tab contains the following fields and buttons:

- '出力ファイル名(O):' (Output File Name) with a text box containing 'SAMPLE1.cob' and a '参照(B)...' (Browse...) button.
- '出力ファイル拡張子(E):' (Output File Extension) with an empty text box.
- '出力ファイルフォルダ(F):' (Output File Folder) with an empty text box and a '参照(R)...' (Browse...) button.

At the bottom of the dialog are three buttons: 'OK', 'キャンセル' (Cancel), and 'ヘルプ' (Help).

更新登録の場合、出力ファイル名の変更はできません。

INSDBINFページ

The screenshot shows the 'File Conversion Settings' dialog box with the 'INSDBINF' tab selected. The dialog has four tabs: 'Command', 'Input', 'Output', and 'INSDBINF'. The 'INSDBINF' tab contains the following fields and buttons:

- A checked checkbox labeled 'INSDBINFコマンドを使用する(N)' (Use INSDBINF command).
- '中間ファイル名(M):' (Intermediate File Name) with a text box containing 'SAMPLE1.cbl' and a '参照(B)...' (Browse...) button.
- '中間ファイル拡張子(E):' (Intermediate File Extension) with an empty text box.
- '中間ファイルフォルダ(F):' (Intermediate File Folder) with an empty text box and a '参照(R)...' (Browse...) button.
- 'INSDBINF オプション(O):' (INSDBINF Option) with a dropdown menu.

At the bottom of the dialog are three buttons: 'OK', 'キャンセル' (Cancel), and 'ヘルプ' (Help).

チェックボックス

INSDBINFコマンドを使用する

プリコンパイラ後に、INSDBINFコマンドを使用するか否かを指定します。
INSDBINFコマンドを使用しない場合、INSDBINFページ内のエディットボックスは入力不可となります。

エディットボックス

中間ファイル名

プリコンパイラにより作成されるファイル名を指定します。
入力ページの入力ファイル名一覧にファイル名を1つ指定した場合に指定します。

中間ファイル拡張子

プリコンパイラにより生成されるファイル名の拡張子を指定します。
[中間ファイル名]が指定されている場合は、入力不可となります。

中間ファイルフォルダ

プリコンパイラにより生成されるファイルを格納するフォルダ名を指定します。
[中間ファイル名]が指定されている場合は、入力不可となります。入力不可の状態、中間ファイルの格納先を変更したい場合、[中間ファイル名]に指定されたファイル名を、フォルダ名を付加したファイル名に変更してください。

INSDBINFオプション

INSDBINFコマンドに対するオプションを指定します。
1つのプロジェクトの中では、INSDBINFフォルダとINSDBINFオプションは関連づけて定義されます。このため、INSDBINFフォルダがプロジェクトに登録済みであった場合にINSDBINFオプションを変更すると、登録済みのINSDBINFオプションも変更されます。

ボタン

参照(中間ファイル名)

プリコンパイラが出力するファイル名を選択します。

参照(中間ファイルフォルダ名)

プリコンパイラが出力するファイルを格納するフォルダ名を選択します。
[中間ファイル名]が指定されている場合は、入力不可となります。

6.4.8.4.3 マクロ

プロジェクトマネージャでは、種々のプリコンパイラの利用を可能とするために、プリコンパイラのオプションはすべて記述しなければなりません。

このため、[コマンドパラメタ]エディットボックスに直接“入力ファイル”、“出力ファイル”などを記述すると、同名のプリコンパイラ定義名で異なる“入力ファイル”、“出力ファイル”などを定義しても、[コマンドパラメタ]エディットボックスに記述されたファイルしかプリコンパイルされなくなります。

そこで、プロジェクトマネージャでは、入力ファイル名、出力ファイル名および出力ファイルフォルダ名を代替するためのマクロを用意しています。同名のプリコンパイラ定義名で異なる“入力ファイル”、“出力ファイル”、“出力ファイルフォルダ”を定義する場合は必ずマクロを使用して[コマンドパラメタ]エディットボックスに記述してください。

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。

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%



例

[入力ファイル名]リストボックス : C:¥COBTEST¥A. PCO
 [出力ファイル名]エディットボックス : C:¥COBTEST¥A. COB
 [コマンドパラメタ]エディットボックス : %INFILE%, oname=%OUTFILE%

と指定した場合は、

[コマンドパラメタ]エディットボックス : C:¥COBTEST¥A. PCO, oname=C:¥COBTEST¥A. COB

と指定したのと同じ意味になります。

6.4.8.5 選択項目と対象プリコンパイラ

プロジェクトマネージャでは、プロジェクトのツリー上で直列に並んでいる、出力ファイルー“INSDBINFコマンド”フォルダー中間ファイルー各種プリコンパイラフォルダ、をひとかたまりとみなします。INSDBINFコマンドを使用しない場合は、出力ファイルー各種プリコンパイラフォルダをひとかたまりとみなします。

便宜上、ここでは、このかたまりを「ブロック」と呼びます。

あるブロック中のファイルまたはフォルダが選択された状態で、[プリコンパイラ設定]ダイアログが呼び出された場合は、対象となるプリコンパイラおよびINSDBINFコマンドは、そのブロック中に含まれるものとなります。

選択されているプロジェクトのツリー上のアイテムと、[プリコンパイラ設定]ダイアログにより対象とされているプリコンパイラおよびINSDBINFコマンドの関係は下表のようになります。

選択アイテム	対象
最終ターゲットファイル または “COBOL ソースファイル”フォルダ	指定した最終ターゲットファイルまたはCOBOL ソースファイルフォルダの配下に、新しいブロック(プリコンパイラおよびINSDBINFコマンド)を生成することが要求されたと判断します。
INSDBINFコマンドの指定されているブロック 中に含まれるアイテム	指定されたブロック中に含まれるプリコンパイラおよびINSDBINFコマンドが対象となります。(注1)
INSDBINFコマンドの指定されていないブロック 中に含まれるアイテム	指定されたブロック中に含まれるプリコンパイラが対象となり、INSDBINFコマンドは非使用と判断します。したがって、“INSDBINFコマンドを使用する”チェックボックスを選択した場合は、ブロック内の各種プリコンパイラフォルダの上に“INSDBINFコマンド”フォルダおよび中間ファイルが生成されます。(注1)
どこのブロックにも属さない(配下に、各種プリコンパイラフォルダ および “INSDBINFコマンド”フォルダが存在しない) ソースファイル	指定されたソースファイルを出力ファイルとする、新しいブロックを生成することを要求されたと判断します。(注2)
上記以外のアイテム	“プリコンパイラ”メニューがマスク状態となり、プリコンパイラ設定ダイアログが使用できなくなります。

注1: どこかのブロックに属しているファイルまたはフォルダを選択して[プリコンパイラ設定]ダイアログを開いた場合、出力ファイル名の変更はできません。

注2: どのブロックにも属していないファイルを選択して[プリコンパイラ設定]ダイアログを開いた場合、選択されているファイル自身の名前を変更することはできません。

6.4.8.6 インクルードファイル

プリコンパイラの種類によっては、COBOLの登録集と同様に、プログラム中の一部を外に切り出して、翻訳時に自動的に取り込むことができます。

このように、プログラム中の一部を切り出して別ファイルとしたものを「インクルードファイル」と呼びます。

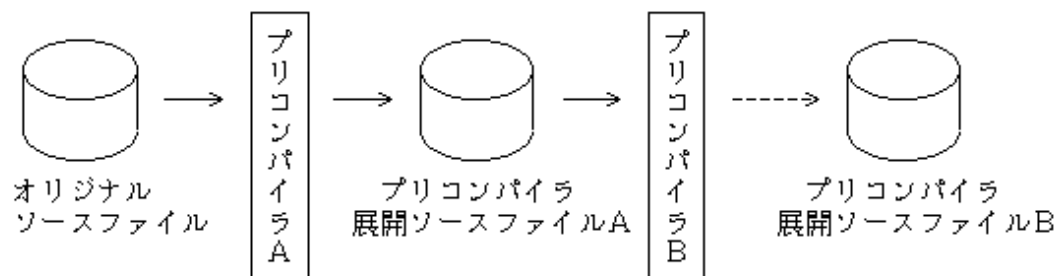
プロジェクトマネージャでは、COBOLの登録集と同様に扱います。

プロジェクトマネージャでインクルードファイルを指定する場合は、以下のような手順となります。

1. 対象となるプリコンパイラの入力ファイルのアイテムを、依存関係のツリー上で選択します。
2. [編集]メニューの“フォルダ作成”から“インクルードファイル”を選択し、“インクルードファイル”フォルダを生成します。
3. 生成したフォルダに、インクルードファイルを“追加”または“新規作成”します。

6.4.8.7 多段プリコンパイル

オリジナルソースファイルに対して、複数のプリコンパイラで、複数回プリコンパイルを行うことを「多段プリコンパイル」と呼びます。

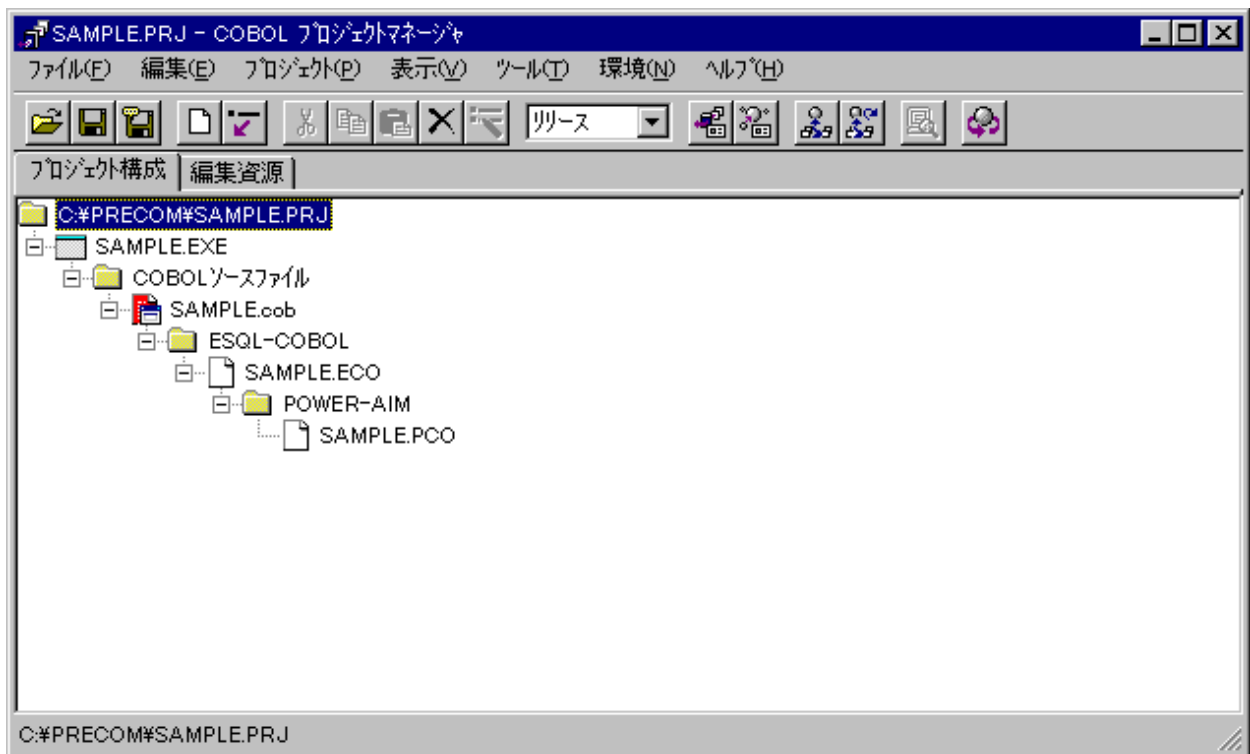


プロジェクトマネージャでは、あるプリコンパイラの入力ファイルを出力ファイルとする、新規のブロックを作成することにより、多段プリコンパイルが実現できます。

多段プリコンパイルを行う場合、プロジェクトに登録されているプリコンパイラの入力ファイルを選択して、[編集]メニューから“プリコンパイラ”を選択します。この場合、複数のプリコンパイラに対する入出力ファイルを一度にプロジェクトに登録することはできません。プリコンパイルが必要なファイルの数だけ“[6.4.8.4 \[プリコンパイラ設定\]ダイアログ](#)”を使用して登録を行ってください。

多段プリコンパイルを行う場合、プリコンパイラが動作する順番は、ツリー上で下層のものからになります。

多段プリコンパイルを行う場合、プロジェクト画面は下図のようになります。



6.4.8.8 注意事項

プロジェクトマネージャで、プリコンパイラが指定されたプロジェクトを扱う場合には、以下の注意事項があります。

- プリコンパイラとしてOracleプリコンパイラを指定したプロジェクトをビルド/リビルドする場合、プリコンパイラのエラーメッセージは赤く色づけされません。また、エラージャンプ機能も無効になります。

6.4.9 実行可能プログラムのバージョン設定機能

[バージョン情報設定]ダイアログを使用して、アプリケーションのバージョン情報を設定することができます。

バージョン情報には、以下の情報を設定できます。

- ファイルバージョン
- 製品バージョン
- 説明
- 会社名
- 製品名
- 著作権
- コメント

[バージョン情報設定]ダイアログは、プロジェクトマネージャに表示されているアプリケーションを選択した後、[プロジェクト]-[オプション]メニューから“バージョン情報設定”を選択することにより表示されます。

バージョン情報設定

ファイルバージョン

メジャー(M): マイナ(M): リリース番号(R):

1 0 0

製品バージョン

メジャー(A): マイナ(N): リリース番号(E):

1 0 0

種類(T): 値(V):

説明
会社名
製品名
著作権
コメント

OK キャンセル ヘルプ

ボタン

OK

バージョン情報の設定が終了したときに選択します。

キャンセル

[バージョン情報設定]ダイアログが表示される前の状態に戻して、ダイアログを終了します。

ヘルプ

ヘルプウィンドウを開くときに選択します。

エディットボックス

ファイルバージョン

メジャー

メジャーバージョンを指定します。スピンボタンで値を変更できます。

マイナ

マイナバージョンを指定します。スピンボタンで値を変更できます。

リリース番号

リリース番号を指定します。スピンボタンで値を変更できます。

製品バージョン

メジャー

メジャーバージョンを指定します。スピンボタンで値を変更できます。

マイナ

マイナバージョンを指定します。スピンボタンで値を変更できます。

リリース番号

リリース番号を指定します。スピンボタンで値を変更できます。

リストボックス

種類

説明、会社名、製品名、著作権およびコメントが表示されます。

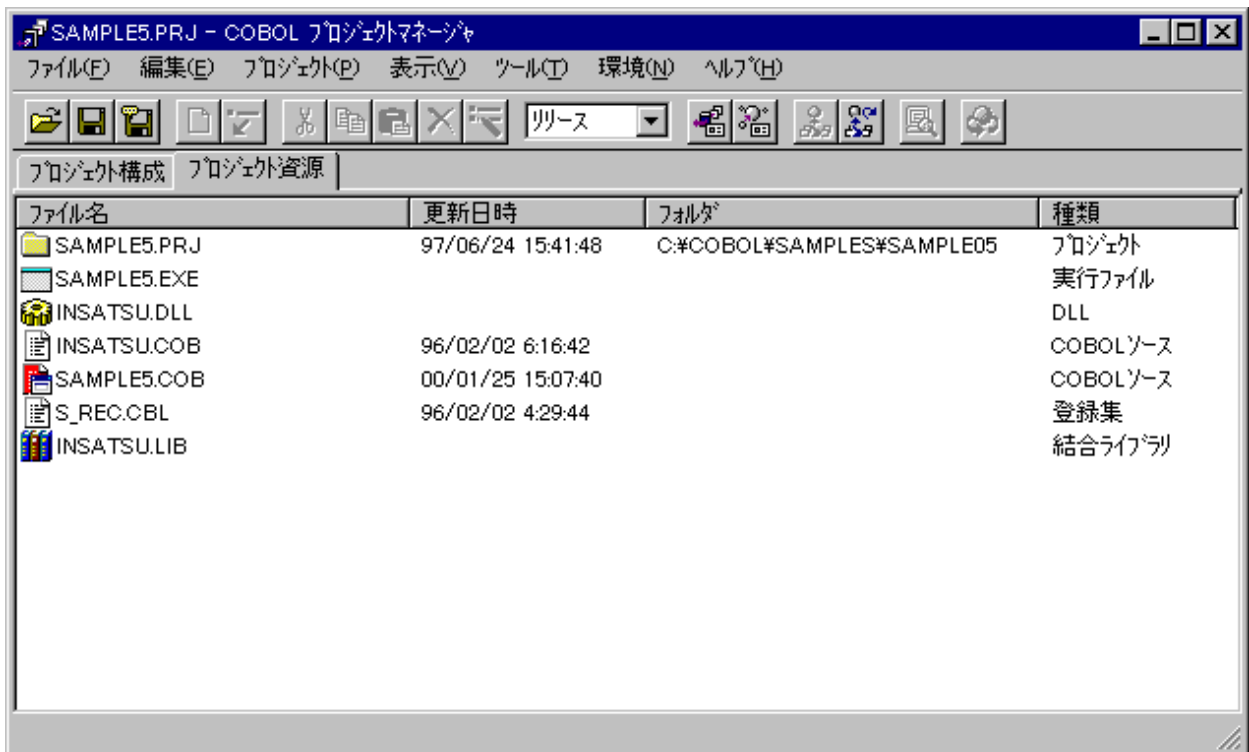
値

種類リストボックスで選択したものに対して情報を入力します。

6.4.10 リストビュー

プロジェクトマネージャでは、プロジェクトの構成資源および資源に関する情報を、リストビューを使用して、一覧形式で表示することができます。また、リストビュー上で複数の資源を選択して、一度に操作することができます。

リストビューは、画面上部にある右側のタブを選択することによって表示します。



プロジェクトの構成資源は、その種類を表すアイコンとファイル名によって表現されます。

情報表示

リストビューの表示内容には、以下の4種類があります。

編集資源

プロジェクトの構成資源のうち、ソース、登録集などの編集対象資源を表示します。プロジェクトマネージャ起動直後は常にこのモードとなります。

ファイル名

表示資源のファイル名です。

更新日時

表示資源の最終更新日時です。

フォルダ

表示資源が存在するフォルダです。ツリーに定義した資源名のパス文字列を表示します。

種類

表示資源の種類です。

プロジェクト資源

プロジェクトツリーに登録されている全資源を表示します。資源に関する情報は、“編集資源”と同様です。

AAD配付ソース生成対象

配布ソース生成時に使用します。詳細は、“[27.2.3 配付ソース生成](#)”を参照してください。

AAD配付ソース対応

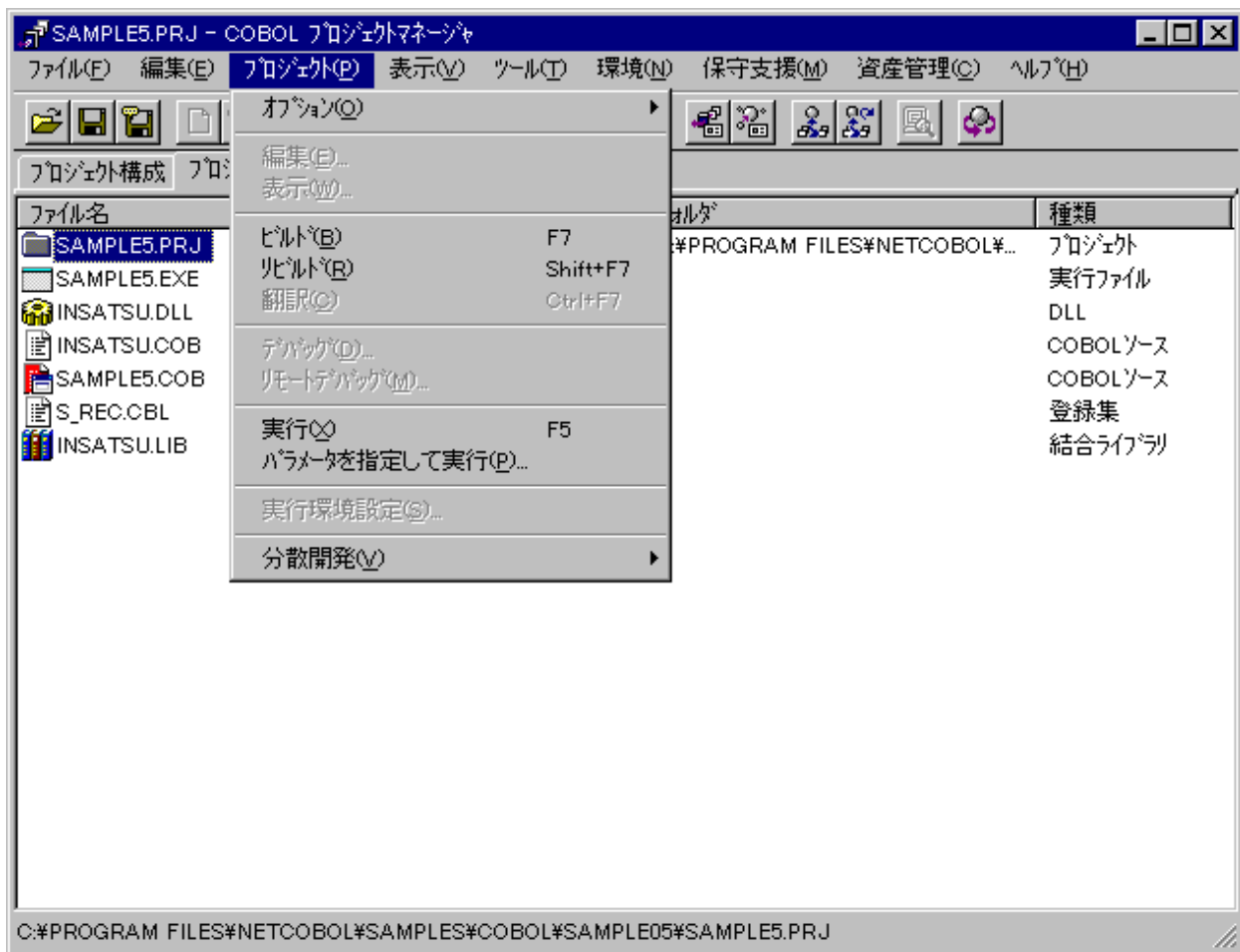
配付ソースの対応関係を表示します。詳細は、“[27.2.7 配付ソース対応表示](#)”を参照してください。

各表示内容は、[表示]メニューの“リスト表示内容”を選択して切り替えます。



資源操作

リストビューでファイル进行操作するには、ファイルを選択してからメニューを選択します。マウスまたはキーボードを用いて複数のファイルを選択し、同時に操作することができます。



注意

異なる種類の資源を同時に選択し、その中の一部の資源だけに有効な操作を行った場合、対象となる資源だけが操作されます。例えば、実行ファイルとCOBOLソースを同時に選択して、“編集”操作を行った場合、COBOLソースファイルだけがエディタで開かれます。ただし、“翻訳”、“ビルド”および“配付ソース生成”は、対象資源だけを選択している場合しか実行できません。

6.5 モジュール定義ファイル

ここでは、モジュール定義ファイルについて説明します。

プロジェクトのビルド・リビルドを行うことにより、モジュール定義ファイルの生成も行うことができます。

6.5.1 モジュール定義ファイルの生成方法

モジュール定義ファイルは、以下の条件でインポートライブラリを作成するときに必要となります。

- ・ STDCALL呼出し規約を指定した動的プログラム構造のプログラムを作成する場合

モジュール定義ファイルのテンプレートを生成する場合には、以下の操作を行ってください。

1. プロジェクトマネージャウィンドウ上で、以下のファイルまたはフォルダを選択します。
 - プロジェクトファイル
 - 最終ターゲットファイル
 - モジュール定義ファイルフォルダ

ー モジュール定義ファイル

2. [編集]メニューから“モジュール定義ファイル生成”を選択します。

注意

- ここでの“モジュール定義ファイル”フォルダとは、最終ターゲットファイル配下に生成するものを指します。ライブラリファイル配下に生成するものについては、“[17.4.5.3 相互参照クラスのリンク処理](#)”を参照してください。
- DLLのモジュール定義文EXPORTSには、最終ターゲットファイルに依存するソースファイルおよびオブジェクトファイルの、拡張子を除いたファイル名が自動的に設定されます。そのため、プログラムIDがファイル名と異なる場合には、設定内容を変更する必要があります。
- モジュール定義ファイル自動生成時に指定したファイルまたはフォルダによって、動作は下表のように異なります。

指定したファイル	動作
プロジェクトファイル	プロジェクト上に登録されているすべてのDLL に対して、モジュール定義ファイルを作成します。 その際、モジュール定義ファイルが依存ファイルとして登録されているDLL は指定されたファイル名、登録されていないDLLは“DLL 名.DEF”で作成されます。
ターゲットファイル	指定したターゲットファイルに対してモジュール定義ファイルを作成します。ただし、指定できるターゲットファイルは、DLLだけです。 その際、モジュール定義ファイルが依存ファイルとして登録されているDLL は指定されたファイル名、登録されていないDLL は“DLL 名.DEF”で作成されます。
モジュール定義ファイルフォルダ	指定されたフォルダの配下にモジュール定義ファイルを作成します。 その際、フォルダ配下にすでにモジュール定義ファイルが指定されている場合は指定されたファイル名、指定されていない場合は“DLL 名.DEF”で作成されます。
モジュール定義ファイル	指定されたファイル名でモジュール定義ファイルが作成されます。

- 前述の用途以外では、モジュール定義ファイルは不要となります。
ビルド・リビルド時には、プロジェクトのツリー上からモジュール定義ファイルを削除してください。

6.6 MAKEファイル

プロジェクトマネージャは、ビルダを使用して“ビルド”または“リビルド”を実現しています。

ビルダは、複数資源の管理およびそれら資源に対するコマンドの順序付けと実行を行います。

MAKEファイルとは、そのルールが記述してあるファイルのことです。

プロジェクトマネージャは、プロジェクトの依存関係を元にMAKEファイルを作成し、ビルダを起動しています。

ここでは、プロジェクトマネージャが作成するMAKEファイルについて説明します。

6.6.1 MAKEファイルの用途

MAKEファイルは、プロジェクトファイルと比較すると以下のメリットがあります。

- MAKEファイルはテキストファイルです。そのため、テキストエディタを使用して内容の変更が行えます。
ただし、ビルダでビルドする場合には、MAKEファイルの記述形式に注意する必要があります。詳細については、ヘルプを参照してください。
また、プロジェクトマネージャからビルドした場合は、MAKEファイルはつねに更新して上書きされるため、MAKEファイルを変更した場合は、名前を付けて保存(別名で保存)してください。
- プロジェクトマネージャで作成されたMAKEファイルはNMAKEコマンドでも動作可能です。
NMAKEコマンドはバッチとして動作可能なため、バッチ環境下での構築処理が可能になります。

上記のメリットから、利用者がMAKEファイルを使用することにより、以下に示すような作業が可能になります。

- MAKEファイルに直接書き込むことにより、プロジェクトマネージャがサポートしていないコマンドの実行が可能になります。

- ・ バッチ環境下で動作させることにより、対話型のオペレーションから開放されます。

6.6.2 プロジェクトマネージャのMAKEファイル

プロジェクトマネージャが作成するMAKEファイルを以下に示します。

```

:
COBOL_PATH=C:¥COBOL ... [1]
PROJECT_PATH=C:¥SAMPLE ... [2]
:
SAMP1.OBJ : SAMP1.COB SAMP2.REP $(COBOL_PATH)¥COBOL85.CBI ... [3]
           $(COBOL_PATH)¥COBOL32 -i $(COBOL_PATH)¥COBOL85.CBI -M SAMP1.COB ... [4]
:
SAMP1.EXE : SAMP1.OBJ ... SAMP2.LIB ... [5]
           $(COBOL_PATH)¥LINK ... [6]
           /OUT:SAMP1.EXE ¥
           @$ (PROJECT_PATH)¥SAMPLE.001
:

```

- [1] コンパイラがインストールされているパス
- [2] プロジェクトファイルが格納されているパス
- [3] 翻訳を実行するための依存関係
- [4] 翻訳コマンド
- [5] リンクを実行するための依存関係
- [6] リンクコマンド

プロジェクトマネージャは、以下のタイミングで“プロジェクトファイル名.MAK”という名前のMAKEファイルを生成します。

- ・ ビルド・リビルド
- ・ プロジェクトファイルの保存時
- ・ プロジェクトマネージャ終了時

6.6.3 MAKEファイルとプロジェクトファイル

MAKEファイルとプロジェクトファイルは、ともにファイルの依存関係を指定したファイルです。

“[表6.3 MAKEファイルとプロジェクトファイルの比較](#)”で、MAKEファイルとプロジェクトファイルの比較を行います。

表6.3 MAKEファイルとプロジェクトファイルの比較

比較項目	MAKEファイル	プロジェクトファイル
ファイル形式	テキストファイル	バイナリファイル
編集用ツール	テキストファイルエディタ (注1)	プロジェクトマネージャ
編集方法	テキストファイルを編集	依存関係のツリー構造を編集
構築(実行)ツール	・ ビルダ ・ NMAKEコマンド	プロジェクトマネージャ
動作可能なコマンド	すべての実行可能ファイル	COBOL コンパイラ リンカ (注2)
登録可能なファイル	すべてのファイル	・ 実行可能ファイル ・ COBOL ソースファイル ・ オブジェクトファイル ・ リポジットファイル

比較項目	MAKEファイル	プロジェクトファイル
		・登録集ファイル ・各種定義体ファイル ・ライブラリファイル ・モジュール定義ファイル
目的	プロジェクトマネージャ非サポートコマンドの実行およびバッチ環境下での運用	プロジェクトマネージャを使用して、COBOL アプリケーションの構築を行う

注1：プロジェクトマネージャも、MAKEファイルを作成します。ただし、これはプロジェクトファイルの内容をもとに新規に作成しているものであり、既存のMAKEファイルを編集しているわけではありません。

注2：プロジェクトマネージャは、ファイルの種類を判断して実行するコマンドを決めています。そのため、利用者がコマンドを意識する必要はありません。

6.6.4 コマンドファイル

“6.6.2 プロジェクトマネージャのMAKEファイル”に示したMAKEファイルの、下線部に記述されているファイルを「コマンドファイル」と呼びます。

コマンドファイルとは、実行可能ファイルまたはDLLを作成するときにリンクするオブジェクトファイル、オブジェクトライブラリ、インポートライブラリおよびリンクオプションを記述しているファイルのことです。

コマンドファイルは、1つのMAKEファイル中で作成するライブラリ(実行可能ファイルまたはDLL)の数だけ作成されます。

コマンドファイル名は、“プロジェクト名.nnn”(nnnは000～999)になります。

MAKEファイルをビルダで動作させる場合は、MAKEファイル中に記述されているコマンドファイルは必須になります。

第7章 ファイルの処理

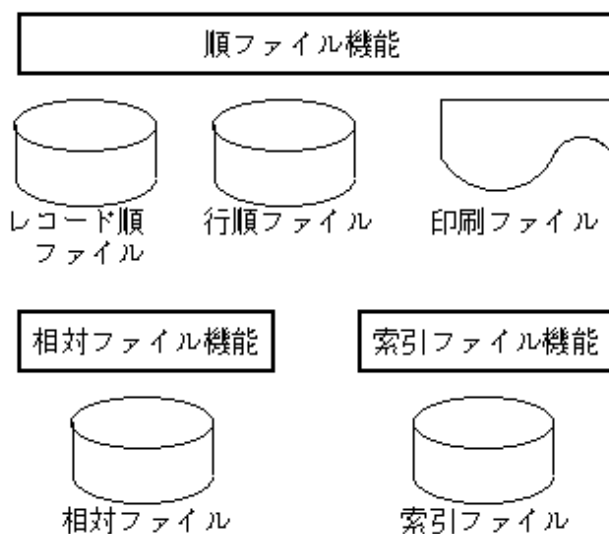
本章では、ファイルからデータを読み込んだり、ファイルにデータを書き出したりする処理、COBOLファイルユーティリティの使い方および他のファイルシステムの使用法について説明します。

7.1 ファイルの種類

ここでは、ファイルの種類と特徴、レコードの設計方法およびファイルの処理方法について説明します。

7.1.1 ファイルの種類と特徴

NetCOBOLでは、順ファイル機能、相対ファイル機能および索引ファイル機能を使って、以下のファイル进行处理することができます。



それぞれのファイルの特徴を“[表7.1 ファイルの種類と特徴](#)”に示します。

表7.1 ファイルの種類と特徴

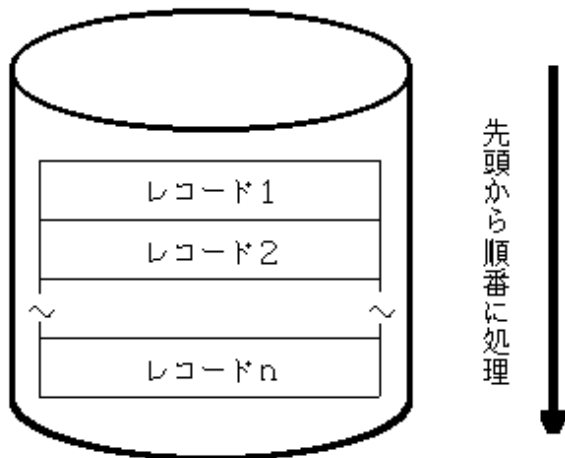
	ファイルの種類				
	レコード順ファイル	行順ファイル	印刷ファイル	相対ファイル	索引ファイル
レコードの処理	レコードが格納されている順番			相対レコード番号	レコードキーの値
使用できる媒体	ハードディスク(*) フロッピーディスク	ハードディスク(*) フロッピーディスク	印刷装置	ハードディスク(*) フロッピーディスク	ハードディスク(*) フロッピーディスク
利用例	データ退避 作業ファイル	テキストファイル	データの印刷	作業ファイル	マスタファイル

* : 仮想デバイス(NULなど)は指定できません。

ファイルの種類は、ファイルを作成するときに決定され、あとで変更することはできません。ファイルを作成するときには、ファイルの特徴を十分に理解し、用途に合ったファイルの種類を選択してください。以下にそれぞれのファイルについて説明します。

レコード順ファイル

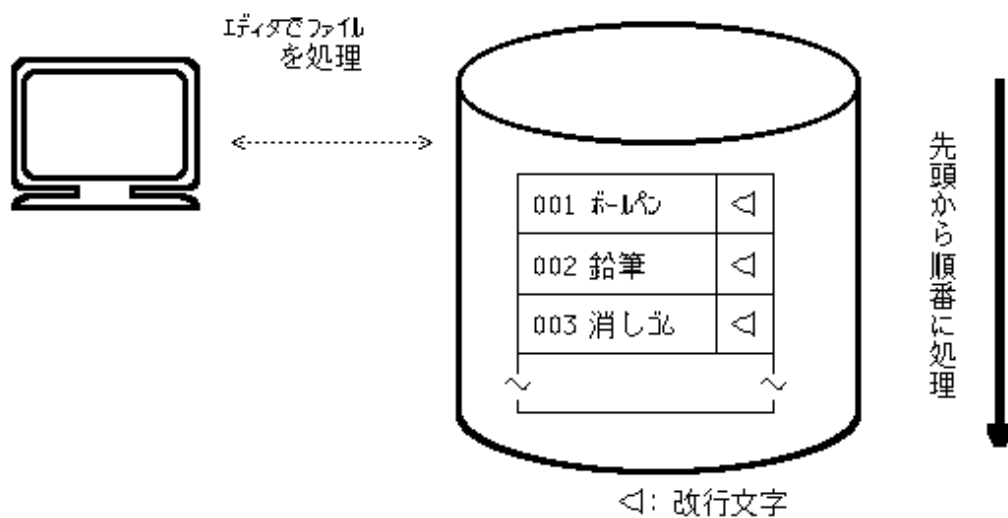
レコード順ファイルでは、ファイルの先頭レコードからファイルに書き出した順番でレコードを読んだり、更新したりできます。レコード順ファイルは、最も簡単に扱うことのできるファイルで、データを順次蓄積する場合や、大量データを保存する場合などに効果的です。



行順ファイル

行順ファイルでは、ファイルの先頭レコードからファイルに書き出した順番でレコードを読むことができます。行順ファイルでは、改行文字をレコードの区切りとします。

行順ファイルは、エディタで作成したテキストファイル进行操作するときなどに利用します。



改行文字は、2バイトの大きさです。改行文字の内容を16進数表記で示します。

0x0D	0x0A

注意

- レコード読み込み時の改行文字の扱いについては、“[レコード内の制御文字の扱い](#)”を参照してください。
- ADVANCING指定付きのWRITE文を実行した場合、改行文字以外の制御文字が出力されます。詳細は、“[ADVANCING指定時の動作](#)”を参照してください。
- CSV形式データの操作については、“[第13章 CSV形式データの操作](#)”を参照してください。

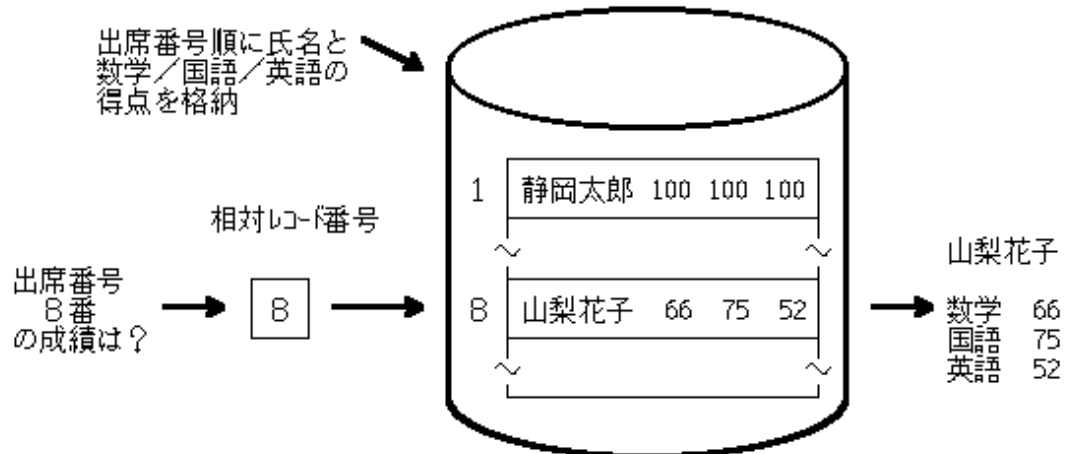
印刷ファイル

印刷ファイルとは、順ファイル機能を使用した印刷するためのファイルのことで、印刷ファイルという特別なファイルがあるわけではありません。

印刷ファイルについては、“[第8章 印刷処理](#)”で説明します。

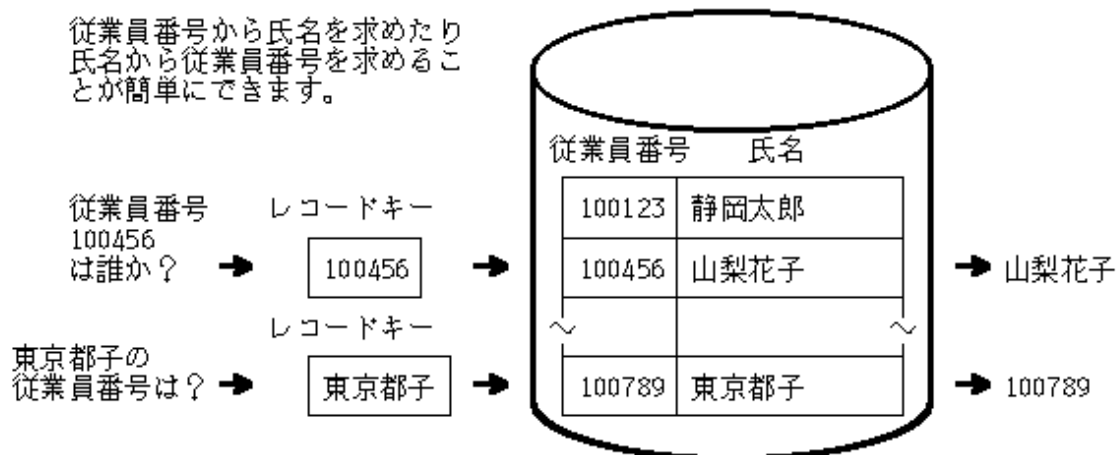
相対ファイル

相対ファイルでは、ファイルの先頭のレコードを1とする相対レコード番号を指定することによって、レコードを読んだり、更新したりできます。相対ファイルは、相対レコード番号をキーとしてアクセスする作業ファイルなどに利用します。



索引ファイル

索引ファイルでは、レコード中のある項目の値(レコードキー)を指定することによって、レコードを読んだり、更新したりできます。索引ファイルは、レコード中のある項目の値から他の情報を引き出すマスタファイルなどに利用します。



7.1.2 レコードの設計

ここでは、レコード形式の種類と特徴および索引ファイルを使用するときのレコードキーについて説明します。

7.1.2.1 レコード形式

レコード形式には、固定長レコード形式と可変長レコード形式があります。それぞれのレコード形式について、以下に説明します。

固定長レコード形式

固定長レコード形式では、1つのレコードは一定のレコード長で区切られ、ファイル中のレコードの大きさはすべて同じになります。

可変長レコード形式

可変長レコード形式では、レコードごとにレコードの大きさが異なります。1つのレコードの大きさは、そのレコードがファイルに書き出されたときの大きさとなります。可変長レコード形式は、必要な大きさにレコードを書き出すことができるため、ファイル容量を小さくしたい場合に有効です。

7.1.2.2 索引ファイルのレコードキー

索引ファイルのレコードの設計では、レコードキーを決定する必要があります。レコードキーは、レコード中の項目で、複数個指定することもできます。レコードキーには、主レコードキー(主キー)と副レコードキー(副キー)があり、ファイル中のレコードは主キーの昇順に格納されます。ファイル中のどのレコードを処理するかは、主キーおよび副キーの両方またはどちらかの値を指定することにより決定します。また、あるレコードから昇順に処理していくこともできます。



注意

レコードキーを決定するときには、以下の注意が必要です。

- ・ 同一ファイルを複数のレコード構成で処理したい場合、主キーは、すべてのレコード構成で同じ位置と大きさである必要があります。
- ・ 可変長レコード形式の場合、レコードキーの位置は固定部になければなりません。

7.1.3 ファイルの処理方法

ファイルに対する処理は、以下の6種類があります。

処理	内容
ファイルの作成	ファイルにレコードを書き出します。
ファイルの拡張	ファイルの最後のレコードの後にレコードを書き出します。
レコードの挿入	ファイルの任意の位置にレコードを書き出します。
レコードの参照	ファイルの中のレコードを読み込みます。
レコードの更新	ファイルの中のレコードを書き換えます。
レコードの削除	ファイルの中のレコードを削除します。

これらの処理が可能かどうかは、ファイルに対するアクセス形態で異なります。アクセス形態には、以下の3種類があります。

アクセス形態	内容
順呼出し	一連のレコードを一定の順序で処理します。
乱呼出し	任意のレコードを単独で処理します。
動的呼出し	順呼出しと乱呼出しの両方の処理ができます。

各ファイル編成で行うことのできる処理を“表7.2 ファイルの種類と処理”に示します。

表7.2 ファイルの種類と処理

ファイルの種類	アクセス形態	処理					
		作成	拡張	挿入	参照	更新	削除
レコード順ファイル	順呼出し	○	○	×	○	○	×

ファイルの種類	アクセス形態	処理					
		創成	拡張	挿入	参照	更新	削除
行順ファイル	順呼出し	○	○	×	○	×	×
印刷ファイル	順呼出し	○	○	×	×	×	×
相対ファイル／索引ファイル	順呼出し	○	○	×	○	○	○
	乱呼出し	○	×	○	○	○	○
	動的呼出し	○	×	○	○	○	○

○:処理可能
×:処理不可能

また、ファイル処理では、ファイル自体を排他モードにしたり、使用中のレコードを排他状態にしたりすることにより、他からのアクセスを不可能にすることができます。これをファイルの排他制御といいます。ファイルの排他制御については、“[7.7.2 ファイルの排他制御](#)”で説明します。

7.2 レコード順ファイルの使い方

ここでは、レコード順ファイルについて以下の項目を説明します。

- ファイルの定義方法
- レコードの定義方法
- ファイルの処理方法

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT ファイル名
ASSIGN TO ファイル参照子
[ORGANIZATION IS SEQUENTIAL].
DATA DIVISION.
FILE SECTION.
FD ファイル名
[RECORD レコードの大きさ].
01 レコード名.
レコード記述項
PROCEDURE DIVISION.
OPEN オープンモード ファイル名.
[READ ファイル名.]
[REWRITE レコード名.]
[WRITE レコード名.]
CLOSE ファイル名.
END PROGRAM プログラム名.
```

7.2.1 レコード順ファイルの定義

ここでは、COBOLプログラムでレコード順ファイルを使うときに必要なファイルの定義について説明します。

ファイル名とファイル参照子

まず、COBOLプログラムで使用するファイル名を決定し、SELECT句に記述します。このファイル名は、COBOLの利用者語の規則に従った名前にします。次に、ファイル参照子を決定し、ASSIGN句に記述します。ファイル参照子には、ファイル識別名、ファイル識別名定数、データ名または文字列DISKのどれかを指定します。ファイル参照子は、SELECT句に指定したCOBOLプログラムのファイル名と実際の入出力媒体のファイルとを関連付けるために使用します。ファイル参照子に何を指定したかによって、COBOLプログラムのファイル名と実際の入出力媒体のファイルを関連付ける方法が異なります。ファイル参照子に何を指定するかは、実際の入出力媒体のファイル名がいつ決まるかにより、以下のように決定することをおすすめします。

- ・ COBOLプログラム作成時に実際の入出力媒体のファイル名が決定し、その後変更されない場合には、ファイル識別名定数または文字列DISKを指定します。
- ・ COBOLプログラム作成時に実際の入出力媒体のファイル名が決定しなかったり、毎回のプログラム実行時にファイル名を決定したい場合には、ファイル識別名を指定します。
- ・ プログラムの中でファイル名を決定したい場合には、データ名を指定します。
- ・ プログラムの終了時には必要のない、一時的なファイルである場合には、文字列DISKを指定します。



注意

文字列DISKを指定した場合、プログラムの終了時に、使用したファイルが削除されるわけではありません。

また、実際の入出力媒体のファイル名には、以下の文字を含むことができます。

空白、「+」、「,」、「:」、「=」、「[」、「]」、「(」、「)」、「'」

なお、コンマ(,)を含む場合には、ファイル名を二重引用符(")で囲む必要があります。



参考

レコード順ファイルおよび行順ファイルでは、ファイルを高速に処理することができます。指定方法については、“[7.7.4 ファイルの高速処理](#)”を参照してください。

SELECT句およびASSIGN句の記述例を“[表7.3 SELECT句およびASSIGN句の記述例](#)”に示します。

表7.3 SELECT句およびASSIGN句の記述例

ファイル参照子の種類	記述例	備考
ファイル識別名	SELECT ファイル 1 ASSIGN TO INFILE	プログラム実行時に実際の入出力媒体と結び付ける必要があります。
データ名	SELECT ファイル 2 ASSIGN TO データ名	データ名はデータ部の作業場所節で定義する必要があります。
ファイル識別名定数	SELECT ファイル 3 ASSIGN TO "C.DAT"	
文字列DISK	SELECT DATA1 ASSIGN TO DISK	ファイル名を絶対パス名で指定することはできません。

ファイル編成

ORGANIZATION句にSEQUENTIALを指定します。なお、ORGANIZATION句を省略した場合には、SEQUENTIALを指定したものとみなされます。

7.2.2 レコード順ファイルのレコードの定義

ここでは、レコード形式とレコード長およびレコードの構成について説明します。

レコード形式とレコード長

レコード順ファイルのレコード形式には、固定長レコード形式と可変長レコード形式があります。固定長レコード形式のレコード長は、RECORD句に指定した値、またはRECORD句が省略された場合はレコード記述項の最大値となります。可変長レコード形式では、レコードを書き出したときのレコードの長さが、そのレコードのレコード長になります。書き出すレコードの長さは、RECORD句の“DEPENDING

ON データ名”に記述したデータ名に設定することができます。また利用者は、このデータ名を使って、レコードの入力時にレコード長を得ることもできます。

レコードの構成

レコード中のデータの属性、位置および大きさは、レコード記述項で定義します。



例

固定長レコード形式のレコードの定義例

データ1 (100バイト)	データ2 (100バイト)

```
FD データ保存用ファイル.
01 データレコード.
   02 データ 1 PIC X(100).
   02 データ 2 PIC X(100).
```

可変長レコード形式のレコードの定義例

データ1 (100バイト)	データ2 (1～100バイト)

```
FD データ保存用ファイル
RECORD IS VARYING IN SIZE FROM 101 TO 200 CHARACTERS
      DEPENDING ON レコード長.
01 データレコード.
   02 データ 1 PIC X(100).
   02 データ 2.
      03 PIC X OCCURS 1 TO 100 TIMES DEPENDING ON 長さ.
      :
WORKING-STORAGE SECTION.
01 レコード長 PIC 9(3) BINARY.
01 長さ PIC 9(3) BINARY.
      :
```



注意

OCCURS句を指定した可変長データ項目を含む複数のデータ項目から構成される可変長レコード形式の場合、以下の注意が必要です。

レコードの書出し時

レコード項目にデータを転記するとき、最初の可変長データ項目から順にデータおよびデータ長を転記しなければなりません。データを転記する順番によっては、意図しない転記結果となる場合があります。

レコードの読み込み時

レコードの読み込みでは、読み込んだレコードの全体の長さは返却されますが、レコード内に含まれる可変長データ項目の長さは返却されません。この場合、全体のレコード長から固定部の長さを引いて可変長データ項目の長さを求めてください。ただし、複数の可変長データ項目が定義されている場合、計算では長さを求めることはできません。この場合、レコード内に可変部分の終わりを示す情報を埋め込むか、または、各可変部分の長さを持つなどして、個々の可変長データ項目の長さを求めてください。

7.2.3 レコード順ファイルの処理

レコード順ファイルの処理では、入出力文を使って、創成、拡張、参照、更新を行うことができます。ここでは、レコード順ファイルの処理で使用する入出力文の種類と使い方およびそれぞれの処理の概要について説明します。

入出力文の種類と使い方

入出力文の種類	使い方
OPEN文	OPEN文はファイル処理の最初に、CLOSE文はファイル処理の最後にそれぞれ1回だけ必ず実行します。OPEN文に指定するオープンモードは、ファイルに対してどのような処理を行うかによって決定されます。たとえば、創成処理を行う場合には、出力モード(OUTPUT指定)のOPEN文を実行します。
CLOSE文	
READ文	ファイル中のレコードを読み込むときに使用します。
REWRITE文	ファイル中のレコードを更新するときに使用します。
WRITE文	ファイルにレコードを書き出すときに使用します。

処理の概要

創成

レコード順ファイルを創成するには、ファイルを出力モードで開いて、WRITE文を使ってファイルにレコードを書き出していきます。

```
OPEN OUTPUT ファイル名.
レコードの編集処理
WRITE レコード名 ~.
CLOSE ファイル名.
```



注意

すでに存在するファイルに対して創成処理を行った場合、そのファイルは新しく作り直され、元の内容は失われます。

拡張

レコード順ファイルの拡張は、ファイルを拡張モードで開いて、WRITE文を使ってファイルにレコードを書き出していきます。このとき、ファイルの最後のレコードの後ろにレコードが追加されます。

```
OPEN EXTEND ファイル名.
レコードの編集処理
WRITE レコード名 ~.
CLOSE ファイル名.
```

参照

レコードの参照では、ファイルを入力モードで開いて、READ文を使ってファイルのレコードを先頭から順番に読み込みます。

```
OPEN INPUT ファイル名.
READ ファイル名 ~.
CLOSE ファイル名.
```



注意

ファイル管理記述項のSELECT句にOPTIONALを指定した場合、OPEN文実行時にファイルが存在しなくてもOPEN文は成功となり、最初のREAD文の実行でファイル終了条件が成立します。ファイル終了条件については、“[7.6 入出力エラー処理](#)”を参照してください。

更新

レコードの更新では、ファイルを入出力モードで開いて、まずREAD文を使ってファイルのレコードを読み込み、次にREWRITE文を使ってレコードを書き換えます。

```
OPEN I-O ファイル名.
READ ファイル名 ~.
レコードの編集処理
```

```
REWRITE レコード名 ~.  
CLOSE ファイル名.
```

注意

- REWRITE文を実行した場合、直前のREAD文により読み込まれたレコードの内容が更新されます。
- レコード形式が可変長の場合、レコードの長さを変更することはできません。

7.3 行順ファイルの使い方

ここでは、行順ファイルについて以下の項目を説明します。

- ファイルの定義方法
- レコードの定義方法
- ファイルの処理方法

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. プログラム名.  
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT ファイル名  
        ASSIGN TO ファイル参照子  
        ORGANIZATION IS LINE SEQUENTIAL.  
DATA DIVISION.  
FILE SECTION.  
FD ファイル名  
    [RECORD レコードの大きさ].  
01 レコード名.  
    レコード記述項  
PROCEDURE DIVISION.  
    OPEN   オープンモード ファイル名.  
    [READ  ファイル名.]  
    [WRITE レコード名.]  
    CLOSE ファイル名.  
END PROGRAM プログラム名.
```

7.3.1 行順ファイルの定義

ここでは、COBOLプログラムで行順ファイルを使うときに必要なファイルの定義について説明します。

ファイル名とファイル参照子

行順ファイルでは、レコード順ファイルと同様に、ファイル名とファイル参照子を指定します。指定方法については、“[7.2.1 レコード順ファイルの定義](#)”を参照してください。

ファイル編成

ORGANIZATION句にLINE SEQUENTIALを指定します。

7.3.2 行順ファイルのレコードの定義

ここでは、レコード形式とレコード長およびレコードの構成について説明します。

レコード形式とレコード長

行順ファイルのレコード形式とレコード長の説明は、レコード順ファイルの説明と同じです。“[7.2.2 レコード順ファイルのレコードの定義](#)”を参照してください。なお、行順ファイルの1つのレコードは改行文字で区切られるので、レコード形式に関係なく1つのレコードの最後は改行文字になります。ただし、レコード長にはこの改行文字の長さは含まれません。

レコードの構成

レコード中のデータの属性、位置および大きさは、レコード記述項で定義します。なお、レコードを区切るための改行文字は、レコードを書き出すときに付加されるため、レコード記述項で定義する必要はありません。



例

可変長レコード形式のレコードの定義例

テキスト文字列 (1～80文字の英数字)	<

< : 改行文字

```
FD テキストファイル
RECORD IS VARYING IN SIZE FROM 1 TO 80 CHARACTERS
    DEPENDING ON レコード長.
01 テキストレコード.
    02 テキスト文字列.
        03 文字 PIC X OCCURS 1 TO 80 TIMES
            DEPENDING ON レコード長.
:
WORKING-STORAGE SECTION.
01 レコード長 PIC 9(3) BINARY.
```

7.3.3 行順ファイルの処理

行順ファイルの処理では、入出力文を使って、創成、拡張、参照を行うことができます。ここでは、行順ファイルの処理で使用する入出力文の種類と使い方およびそれぞれの処理の概要について説明します。

入出力文の種類と使い方

入出力文の種類	使い方
OPEN文	OPEN文はファイル処理の最初に、CLOSE文はファイル処理の最後にそれぞれ1回だけ必ず実行します。OPEN文に指定するオープンモードは、ファイルに対してどのような処理を行うかによって決定します。たとえば、創成処理を行う場合には、出力モード(OUTPUT指定)のOPEN文を実行します。
CLOSE文	
READ文	ファイル中のレコードを読み込むときに使用します。
WRITE文	ファイルにレコードを書き出すときに使用します。

処理の概要

創成

行順ファイルを創成するには、ファイルを出力モードで開いて、WRITE文を使ってファイルにレコードを書き出していきます。

```
OPEN OUTPUT ファイル名.
レコードの編集処理
WRITE レコード名 ~.
CLOSE ファイル名.
```

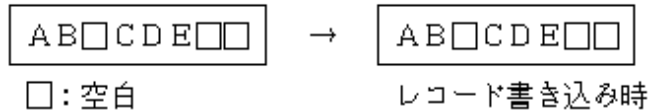


注意

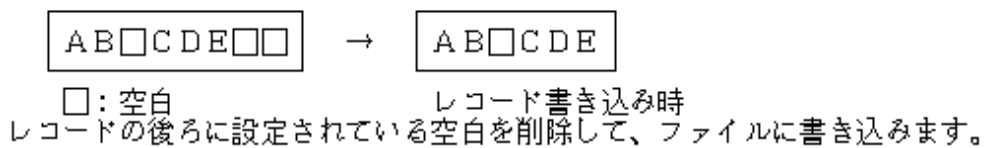
- すでに存在するファイルに対して創成処理を行った場合、そのファイルは新しく作り直され、元の内容は失われます。
- レコードの書出しでは、レコードの領域の内容と改行文字が書き出されます。

- ー レコード内の後置空白を取り除いてレコードを書き出す場合は、環境変数情報@CBR_TRAILING_BLANK_RECORDに文字列“REMOVE”を指定します。[参照]“5.4.1.50 @CBR_TRAILING_BLANK_RECORD(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)”

後置空白を削除する機能を無効にした場合(省略時)



後置空白を削除する機能を有効にした場合



拡張

行順ファイルの拡張は、ファイルを拡張モードで開いて、WRITE文を使って順番にレコードを書き出していきます。このとき、ファイルの最後のレコードの後ろにレコードが追加されます。

```
OPEN EXTEND ファイル名.
  レコードの編集処理
WRITE レコード名 ~.
CLOSE ファイル名.
```

注意

レコード内の後置空白を取り除いてレコードを書き出す場合は、環境変数情報@CBR_TRAILING_BLANK_RECORDに文字列“REMOVE”を指定します。[参照]“5.4.1.50 @CBR_TRAILING_BLANK_RECORD(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)”

参照

レコードの参照では、ファイルを入力モードで開いて、READ文を使ってファイルのレコードを先頭から順番に読み込みます。

```
OPEN INPUT ファイル名.
READ ファイル名 ~.
CLOSE ファイル名.
```

注意

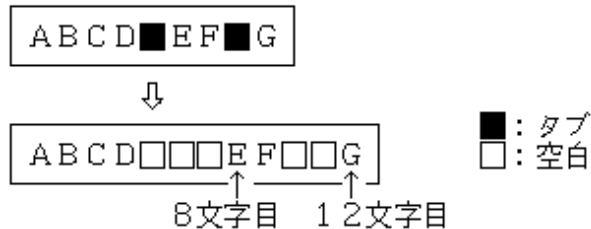
- ー ファイル管理記述項のSELECT句にOPTIONALを指定した場合、OPEN文実行時にファイルが存在していなくても、OPEN文は成功となり、最初のREAD文の実行でファイル終了条件が成立します。ファイル終了条件については、“7.6 入出力エラー処理”を参照してください。
- ー 読み込んだレコードの大きさがレコード長より大きいときには、1回のREAD文の実行でレコード長と同じ長さのデータがレコード領域に設定されます。次のREAD文の実行では、同じレコードのデータの続きから、データがレコード領域に設定されます。設定するデータがレコード長より小さい場合には、レコード領域の残りの領域に、空白が設定されます。

レコード内のタブの扱い

読み込んだレコードにタブが存在した場合、以下のように空白が設定されます。

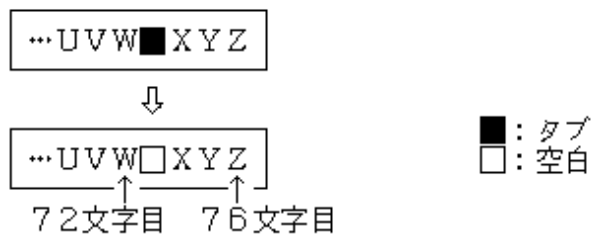
72文字以内にタブが存在する場合

先頭の文字位置を1として、8、12、16、20、24、28、32、36、40、44、48、52、56、60、64、68、72の文字位置にタブの次の文字が配置されるように、空白が設定されます。



72文字を超えた位置にタブが存在する場合

1文字の空白が設定されます。



レコード内の制御文字の扱い

読み込むレコードに制御文字が含まれている場合の動作は、以下のとおりです。

制御文字	動作
0x0C(改行)	レコードの区切り文字として扱います。
0x0D(復帰)	レコードの区切り文字として扱います。
0x1A(データ終了記号)	ファイルの終端として扱います。

注:()内は、制御文字の意味を示します。

ADVANCING指定時の動作

ADVANCING指定付きのWRITE文を実行した場合、改行文字以外の制御文字が出力されます。ADVANCING指定によりファイルに出力されるデータを以下に示します。

ADVANCING指定	出力されるデータ
なし	{レコードデータ}0x0D 0x0A

ADVANCING指定		出力されるデータ
BEFORE	n LINES (*1)	{レコードデータ} 0x0D 0x0A ... 0x0A n回
	PAGE	{レコードデータ} 0x0D 0x0C
AFTER	n LINES (*1)	0x0A ... 0x0A {レコードデータ} 0x0D n回
	PAGE (*2)	0x0C {レコードデータ} 0x0D

*1:行数に0を指定した場合、レコードの後ろに0x0D(復帰)のみを付加したレコードが出力されます。0x0A(改行)は出力されません。

*2:OPEN OUTPUT直後のWRITE文では、0x0C(改頁)は出力されません。



注意

ADVANCING指定のWRITE文で書き出されたレコードを読み込む場合、特に前後のレコードの制御文字が連続している場合は、意図したレコード区切りで読み込まれないことがあります。

- 以下の連続する制御文字は、1つのレコード区切り文字として扱います。
 - 0x0D(復帰)に続き、0x0A(改行)や0x0C(改頁)が存在する
 - 0x0A(改行)や0x0C(改頁)に続き、0x0D(復帰)が存在する
- 0x0A(改行)や0x0C(改頁)が連続する場合、1つの制御文字として扱います。
- レコードの先頭にある0x0A(改行)や0x0C(改頁)は、読み飛ばされます。

制御文字が連続しない場合の動作については、“[レコード内の制御文字の扱い](#)”を参照してください。

Unicodeの行順ファイルを参照する場合

Unicodeアプリケーションにおいて行順ファイルを参照する場合、ファイルの先頭に付加されているBOM(Byte Order Mark)と呼ばれる識別コードの扱いを選択することができます。

使い方

環境変数情報@CBR_FILE_BOM_READに、文字列“CHECK”、“DATA”または“AUTO”を指定します。

```
@CBR_FILE_BOM_READ = { CHECK
                        DATA
                        AUTO }
```

CHECK

レコード定義の字類と一致したBOMが付加されている場合は、識別コードと認識し、READ文の実行でBOMを読み飛ばします。レコード定義の字類と一致していないBOMが付加されている場合、またはBOMが付加されていない場合は、OPEN文の実行が失敗します。

DATA

BOMが付加されている場合は、BOMをレコードデータの一部として読み込みます。
BOMが付加されていない場合は、ファイルの先頭からレコードを読み込みます。

AUTO

レコード定義の字類と一致したBOMが付加されている場合は、識別コードと認識し、READ文の実行でBOMを読み飛ばします。
レコード定義の字類と一致していないBOMが付加されている場合は、OPEN文の実行が失敗します。
BOMが付加されていない場合は、ファイルの先頭からレコードを読み込みます。

7.4 相対ファイルの使い方

ここでは、相対ファイルについて以下の項目を説明します。

- ・ ファイルの定義方法
- ・ レコードの定義方法
- ・ ファイルの処理方法

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. プログラム名.  
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
SELECT ファイル名  
ASSIGN TO ファイル参照子  
ORGANIZATION IS RELATIVE  
[ACCESS MODE IS アクセス形態]  
[RELATIVE KEY IS 相対レコード番号格納域].  
DATA DIVISION.  
FILE SECTION.  
FD ファイル名  
[RECORD レコードの大きさ].  
01 レコード名.  
レコード記述項  
WORKING-STORAGE SECTION.  
[01 相対レコード番号格納域 PIC 9(5) BINARY.]  
PROCEDURE DIVISION.  
OPEN オープンモード ファイル名.  
[MOVE 相対レコード番号 TO 相対レコード番号格納域.]  
[READ ファイル名.]  
[REWRITE レコード名.]  
[DELETE ファイル名.]  
[START ファイル名.]  
[WRITE レコード名.]  
CLOSE ファイル名.  
END PROGRAM プログラム名.
```

7.4.1 相対ファイルの定義

ここでは、COBOLプログラムで相対ファイルを使うときに必要なファイルの定義について説明します。

ファイル名とファイル参照子

相対ファイルでは、レコード順ファイルと同様に、ファイル名とファイル参照子を指定します。指定方法については、“[7.2.1 レコード順ファイルの定義](#)”を参照してください。

ファイル編成

ORGANIZATION句にRELATIVEを指定します。

アクセス形態

ACCESS MODE句に以下のアクセス形態のどれかを指定します。

順呼出し法(SEQUENTIAL)

ファイルの先頭またはある相対レコード番号のレコードから、相対レコード番号の昇順にレコードを処理することができる、順呼出しの処理を行うことができます。

乱呼出し法(RANDOM)

ある相対レコード番号のレコードだけを単独に処理することができる、乱呼出しの処理を行うことができます。

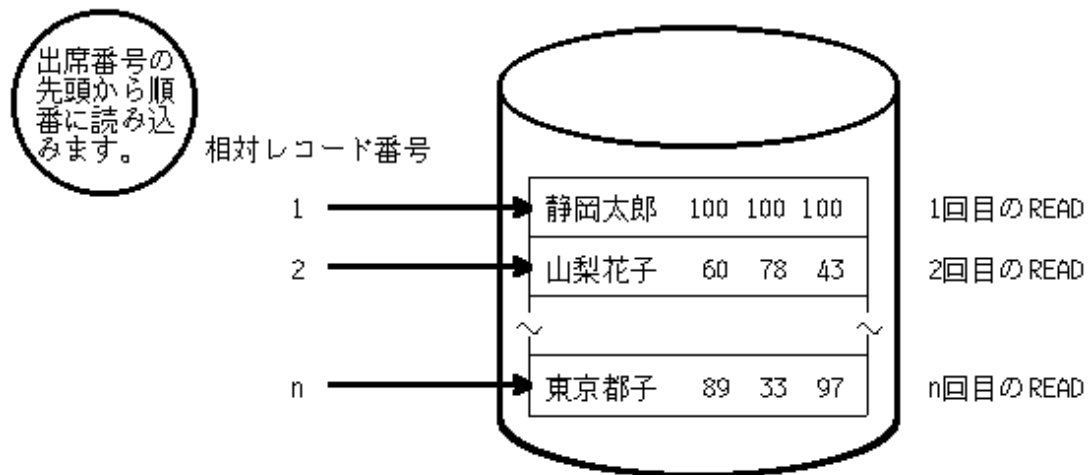
動的呼出し法(DYNAMIC)

順呼出しと乱呼出しの両方の処理を行うことができます。

以下に、レコードの参照処理を例に、順呼出しの場合と乱呼出しの場合での処理の違いを示します。

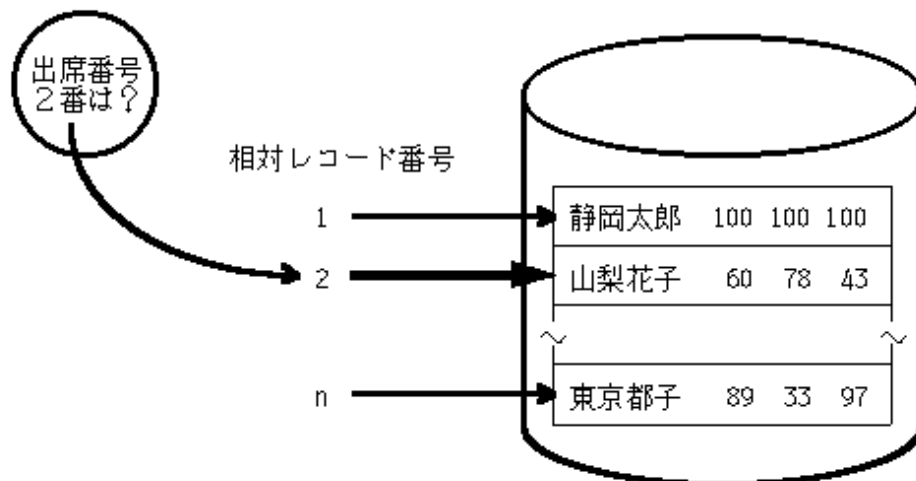
順呼出し

出席番号順に格納されているレコードを先頭から順番に処理する場合



乱呼出し

出席番号順に格納されているレコードの、ある出席番号の者のデータを処理する場合



相対レコード番号

相対レコード番号を設定するデータ名を、RELATIVE KEY句に指定します。ただし、順呼出しの場合は、この句を省略することができます。このデータ名には、レコードの入力時には入力したレコードの相対レコード番号が設定され、出力時には書き出すレコードの相対レコード番号を利用者が設定します。ただし、レコードの出力を順呼出しでアクセスする場合、利用者が設定した相対レコード番号は無視されます。なお、このデータ名は、符号なし整数項目として作業場所節に定義する必要があります。

7.4.2 相対ファイルのレコードの定義

ここでは、レコード形式とレコード長およびレコードの構成について説明します。

レコード形式とレコード長

相対ファイルのレコード形式とレコード長の説明は、レコード順ファイルの説明と同じです。“[7.2.2 レコード順ファイルのレコードの定義](#)”を参照してください。

レコードの構成

レコード中のデータの属性、位置および大きさは、レコード記述項で定義します。なお、相対レコード番号を設定するための領域を定義する必要はありません。



例

固定長レコード形式のレコードの定義例

氏名 (10 文字の日本語文字)	国語 (3桁の数字)	数学 (3桁の数字)	英語 (3桁の数字)

```
FD 学級ファイル.
01 成績レコード.
   02 氏名          PIC N(10).
   02 国語          PIC 9(3).
```

7.4.3 相対ファイルの処理

相対ファイルの処理では、入出力文を使って、創成、拡張、挿入、参照、更新、削除を行うことができます。ここでは、相対ファイルの処理で使用する入出力文の種類と使い方およびそれぞれの処理の概要について説明します。

入出力文の種類と使い方

入出力文の種類	使い方
OPEN文	OPEN文はファイル処理の最初に、CLOSE文はファイル処理の最後にそれぞれ1回だけ必ず実行します。OPEN文に指定するオープンモードは、ファイルに対してどのような処理を行うかによって決定されます。たとえば、創成処理を行う場合には、出力モード(OUTPUT指定)のOPEN文を実行します。
CLOSE文	
DELETE文	ファイル中のレコードを削除するときに使用します。
READ文	ファイル中のレコードを読み込むときに使用します。
START文	処理を開始するレコードを指示するときに指定します。
REWRITE文	ファイル中のレコードを更新するときに使用します。
WRITE文	ファイルにレコードを書き出すときに使用します。

処理の概要

創成〔順/乱/動的〕

相対ファイルを創成するには、ファイルを出力モードで開いて、WRITE文でファイルにレコードを書き出していきます。レコードは、WRITE文に指定したレコードの長さで書き出されます。順呼出し法の場合、書き出したレコードの順番に、相対レコード番号が1、2、3…となります。乱呼出し法または動的呼出し法の場合、相対レコード番号で指定した位置にレコードが書き出されます。

```
OPEN OUTPUT ファイル名.
レコードの編集処理
[相対レコード番号の設定]
WRITE レコード名 ~.
CLOSE ファイル名.
```



すでに存在するファイルに対して創成処理を行った場合、そのファイルは新しく作り直され、元の内容は失われます。

拡張〔順〕

相対ファイルの拡張は、ファイルを拡張モードで開いて、WRITE文で順番にファイルにレコードを書き出します。このとき、ファイルの最後のレコードの後ろにレコードが追加されます。書き出すレコードの相対レコード番号は、ファイルの最大の相対レコード番号から1つ大きい値となります。ファイルの拡張が可能なアクセス形態は、順呼出し法だけです。

```
OPEN EXTEND ファイル名.
レコードの編集処理
WRITE レコード名 ~.
CLOSE ファイル名.
```

参照〔順/乱/動的〕

レコードの参照では、ファイルを入力モードで開いて、READ文でファイルのレコードを読み込みます。順呼出しの場合、START文を使って読み込むレコードの開始位置を指定し、そのレコードから相対レコード番号の順番にレコードを読み込んでいきます。乱呼出しの場合、READ文実行時に設定されている相対レコード番号のレコードが読み込まれます。

順呼出し

```
OPEN INPUT   ファイル名.  
[相対レコード番号の設定  
START   ファイル名  ～.]  
READ     ファイル名 [NEXT]  ～.  
CLOSE    ファイル名.
```

乱呼出し

```
OPEN INPUT   ファイル名.  
相対レコード番号の設定  
READ     ファイル名  ～.  
CLOSE    ファイル名.
```



注意

- ファイル管理記述項のSELECT句にOPTIONALを指定した場合、OPEN文実行時にファイルが存在していなくてもOPEN文は成功となり、最初のREAD文の実行でファイル終了条件が成立します。ファイル終了条件については、“[7.6 入出力エラー処理](#)”を参照してください。
- 乱呼出し法または動的呼出し法で、指定した相対レコード番号のレコードが存在しないとき、無効キー条件が成立します。無効キー条件については、“[7.6 入出力エラー処理](#)”を参照してください。

更新〔順/乱/動的〕

レコードの更新では、ファイルを入出力モードで開きます。順呼出しの場合、まずREAD文でレコードを読み込み、次にREWRITE文でレコードを書き換えます。REWRITE文の実行で、直前のREAD文により読み込まれたレコードの内容が変更されます。乱呼出しの場合、内容を変更したいレコードの相対レコード番号を設定してREWRITE文を実行します。

順呼出し

```
OPEN I-O   ファイル名.  
[相対レコード番号の設定  
START   ファイル名  ～.]  
READ     ファイル名 [NEXT]  ～.  
レコードの編集処理  
REWRITE  レコード名  ～.  
CLOSE    ファイル名.
```

乱呼出し

```
OPEN I-O   ファイル名.  
相対レコード番号の設定  
[READ   ファイル名  ～.]  
レコードの編集処理  
REWRITE  レコード名  ～.  
CLOSE    ファイル名.
```



注意

乱呼出し法または動的呼出し法で、指定した相対レコード番号のレコードが存在しないときに無効キー条件が成立します。無効キー条件については、“[7.6 入出力エラー処理](#)”を参照してください。

削除〔順/乱/動的〕

レコードの削除では、ファイルを入出力モードで開きます。順呼出しの場合、まずREAD文でレコードを読み込み、次にDELETE文でレコードを削除します。DELETE文の実行で、直前のREAD文により読み込まれたレコードが削除されます。乱呼出しの場合、削除したいレコードの相対レコード番号を設定してDELETE文を実行します。

順呼出し

```
OPEN I-O ファイル名.  
[相対レコード番号の設定  
START ファイル名 ～.]  
READ ファイル名 [NEXT] ～.  
DELETE ファイル名 ～.  
CLOSE ファイル名.
```

乱呼出し

```
OPEN I-O ファイル名.  
相対レコード番号の設定  
DELETE ファイル名 ～.  
CLOSE ファイル名.
```



注意

乱呼出し法または動的呼出し法で、指定した相対レコード番号のレコードが存在しないときに無効キー条件が成立します。無効キー条件については、“[7.6 入出力エラー処理](#)”を参照してください。

挿入〔乱/動的〕

レコードの挿入では、ファイルを入出力モードで開いて、挿入したい位置の相対レコード番号を設定してWRITE文を実行します。レコードは、設定した相対レコード番号の位置に挿入されます。

```
OPEN I-O ファイル名.  
レコードの編集処理  
相対レコード番号の設定  
WRITE レコード名 ～.  
CLOSE ファイル名.
```



注意

指定した相対レコード番号のレコードがすでに存在するとき、無効キー条件が成立します。無効キー条件については、“[7.6 入出力エラー処理](#)”を参照してください。



参考

相対レコード番号は、RELATIVE KEY句に指定したデータ名に設定します。

```
例：MOVE 1 TO データ名.
```

7.5 索引ファイルの使い方

ここでは、索引ファイルについて以下の項目を説明します。

- ファイルの定義方法
- レコードの定義方法
- ファイルの処理方法

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. プログラム名.  
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
SELECT ファイル名
```

```

    ASSIGN TO   ファイル参照子
    ORGANIZATION IS INDEXED
    [ACCESS MODE IS   アクセス形態]
    RECORD KEY   IS   主キー名 1 [主キー名 n] ... [WITH DUPLICATES]
    [[ALTERNATE RECORD KEY IS   副キー名 1 [副キー名 n] ...
                                     [WITH DUPLICATES]] ... ].

DATA DIVISION.
FILE SECTION.
FD   ファイル名
    [RECORD   レコードの大きさ].
01   レコード名.
    02   主キー名 1   ~.
    [02   主キー名 n   ~.]
    [02   副キー名 1   ~.]
    [02   副キー名 n   ~.]
    [02   キー以外のデータ ~.]
PROCEDURE DIVISION.
    OPEN   オープンモード ファイル名.
    [MOVE   主キーの値 TO 主キー名 n.]
    [MOVE   副キーの値 TO 副キー名 n.]
    [READ   ファイル名.]
    [REWRITE レコード名.]
    [DELETE ファイル名.]
    [START  ファイル名.]
    [WRITE  レコード名.]
    CLOSE  ファイル名.
END PROGRAM プログラム名.

```

7.5.1 索引ファイルの定義

ここでは、COBOLプログラムで索引ファイルを使うときに必要なファイルの定義について説明します。

ファイル名とファイル参照子

索引ファイルでは、レコード順ファイルと同様に、ファイル名とファイル参照子を指定します。指定方法については、“[7.2.1 レコード順ファイルの定義](#)”を参照してください。

ファイル編成

ORGANIZATION句にINDEXEDを指定します。

アクセス形態

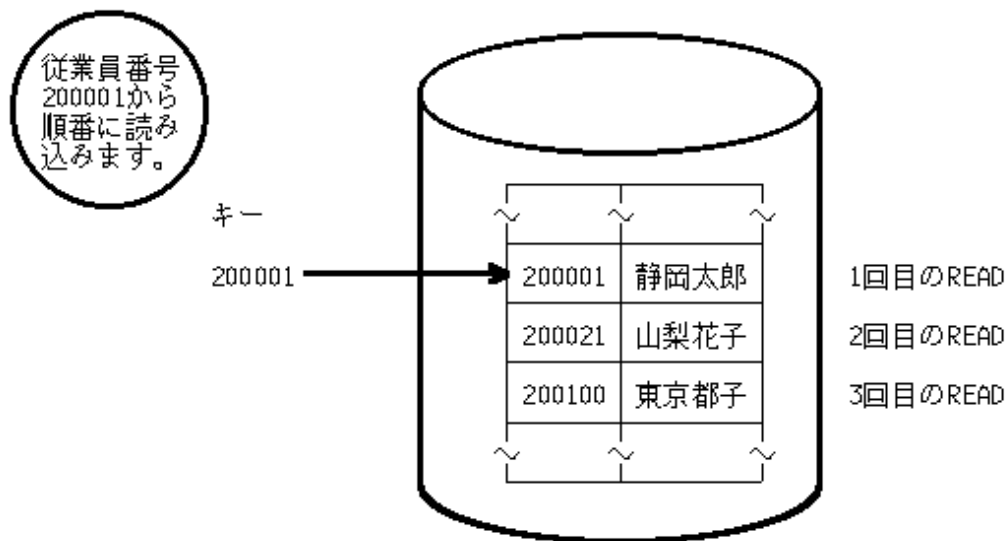
ACCESS MODE句に以下のアクセス形態のどれかを指定します。

- ・ 順呼出し法(SEQUENTIAL): ファイルの先頭またはある特定の値のキーを持つレコードから、キーの昇順にレコードを処理することができます、順呼出しの処理を行うことができます。
- ・ 乱呼出し法(RANDOM): ある特定の値のキーを持つレコードだけを単独に処理することができます、乱呼出しの処理を行うことができます。
- ・ 動的呼出し法(DYNAMIC): 順呼出しと乱呼出しの両方の処理を行うことができます。

以下に、レコードの参照処理を例に、順呼出しの場合と乱呼出しの場合での処理の違いを示します。

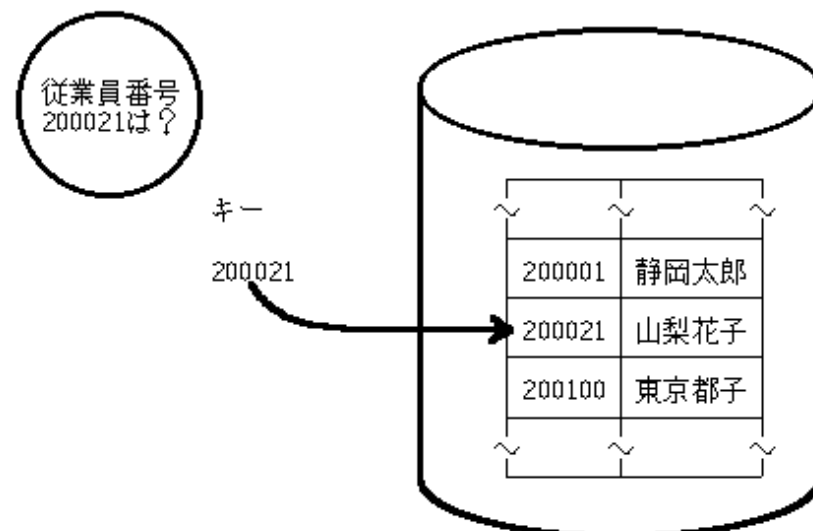
順呼出し

従業員番号の先頭が2に該当する人のデータを順番に取り出す場合



乱呼出し

ある従業員番号に該当する人のデータを取り出す場合



主キーと副キー

キーには主レコードキー(主キー)と副レコードキー(副キー)があり、キーの個数やレコード中での位置および大きさはファイル創成時に決定され、以後変更することはできません。ファイル中のレコードは、論理的に主キーの値の昇順に並んでいて、主キーの値によって特定のレコードを選択することができます。索引ファイルの定義では、必ず主キーとなるデータ項目の名前を**RECORD KEY**句に指定します。副キーは、主キーと同様にファイル中の特定のレコードを選択するための情報となります。副キーとなるデータ項目の名前は、必要に応じて**ALTERNATE RECORD KEY**句に指定します。

RECORD KEY句および**ALTERNATE RECORD KEY**句には、複数のデータ項目を指定することができます。**RECORD KEY**句に複数のデータ項目を指定した場合、それらのデータ項目をつなげたものが主キーとなります。**RECORD KEY**句に指定するデータ項目は、連続している必要はありません。

RECORD KEY句およびALTERNATE RECORD KEY句にDUPLICATESを指定することにより、複数のレコードが同じキーの値を持つこと(キーの値の重複)ができます。DUPLICATESが指定されていない場合、キーの値が重複するとエラーとなります。

7.5.2 索引ファイルのレコードの定義

ここでは、レコード形式とレコード長およびレコードの構成について説明します。

レコード形式とレコード長

索引ファイルのレコード形式とレコード長の説明は、レコード順ファイルの説明と同じです。“[7.2.2 レコード順ファイルのレコードの定義](#)”を参照してください。

レコードの構成

レコード中のキーおよびキー以外のデータの属性、位置および大きさは、レコード記述項で定義します。キーの定義については、以下の点に注意する必要があります。

- 既存のファイル进行处理する場合、主キーまたは副キーに指定する項目の数および位置と大きさは、ファイルが創成されたときとすべて同じでなければなりません。また、主キー、副キーの指定順序および個数が一致していなければなりません。
- 1つのファイルに対してレコード記述項を2つ以上記述する場合、主キーとなるデータ項目は、これらのレコード記述項のうち1つにだけ記述します。そのレコード記述項以外のレコード記述項でも、主キーを定義した文字位置および大きさが主キーとして使用されます。
- 可変長レコード形式の場合、キーの位置は固定部(レコードの先頭からの位置がつねに一定のところ)になければなりません。



例

可変長レコード形式のレコードの定義例

主キー	副キー
従業員番号 (6桁の数字)	氏名 (10文字の日本語)
	所属 (1~16文字の日本語)

```

:
RECORD KEY IS 従業員番号
ALTERNATE RECORD KEY IS 氏名.
:
FD 従業員ファイル
RECORD IS VARYING IN SIZE FROM 28 TO 58 CHARACTERS
      DEPENDING ON レコード長.
:
01 従業員レコード.
02 従業員番号 PIC 9(6).
02 氏名      PIC N(10).
02 所属.
    03 PIC N OCCURS 1 TO 16 TIMES DEPENDING ON 所属の長さ.
:
WORKING-STORAGE SECTION.
01 レコード長 PIC 9(3) BINARY.
01 所属の長さ PIC 9(3) BINARY.
:
```

7.5.3 索引ファイルの処理

索引ファイルの処理では、入出力文を使って、創成、拡張、挿入、参照、更新、削除を行うことができます。ただし、これらの処理はアクセス形態によっては使用不可能な場合があります。ここでは、索引ファイルの処理で使用する入出力文の種類と使い方およびそれぞれの処理の概要について説明します。

入出力文の種類と使い方

入出力文の種類	使い方
OPEN文	OPEN文はファイル処理の最初に、CLOSE文はファイル処理の最後にそれぞれ1回だけ必ず実行します。OPEN文に指定するオープンモードは、ファイルに対してどのような処理を行うかによって決定されます。たとえば、創成処理を行う場合には、出力モード(OUTPUT指定)のOPEN文を実行します。
CLOSE文	
DELETE文	ファイル中のレコードを削除するときに使用します。
READ文	ファイル中のレコードを読み込むときに使用します。
START文	処理を開始するレコードを指示するときに指定します。
REWRITE文	ファイル中のレコードを更新するときに使用します。
WRITE文	ファイルにレコードを書き出すときに使用します。

処理の概要

創成〔順/乱/動的〕

索引ファイルを創成するには、ファイルを出力モードで開いて、WRITE文でファイルにレコードを書き出していきます。

```
OPEN OUTPUT ファイル名.  
レコードの編集処理  
MOVE 主キー値 TO 主キー名.  
WRITE レコード名 ~.  
CLOSE ファイル名.
```



注意

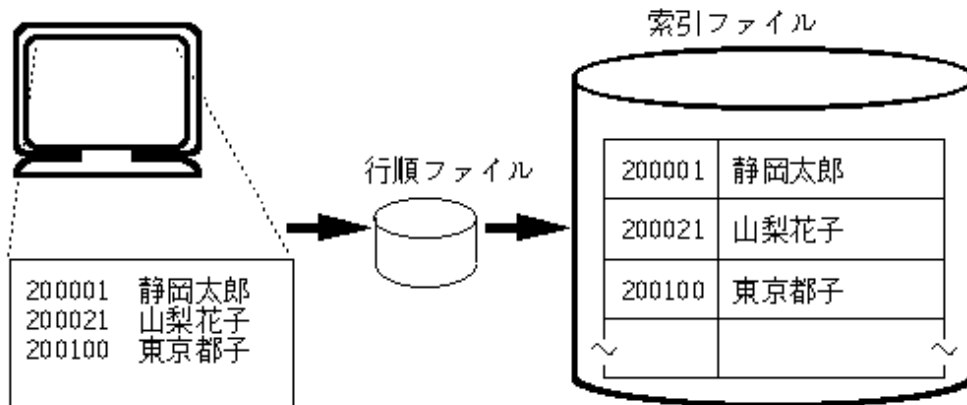
- すでに存在するファイルに対して創成処理を行った場合、そのファイルは新しく作り直され、元の内容は失われます。
- レコードを書き出すとき、主キーの値を設定する必要があります。また、順呼出しの場合、主キーの内容が昇順になるように書き出す必要があります。



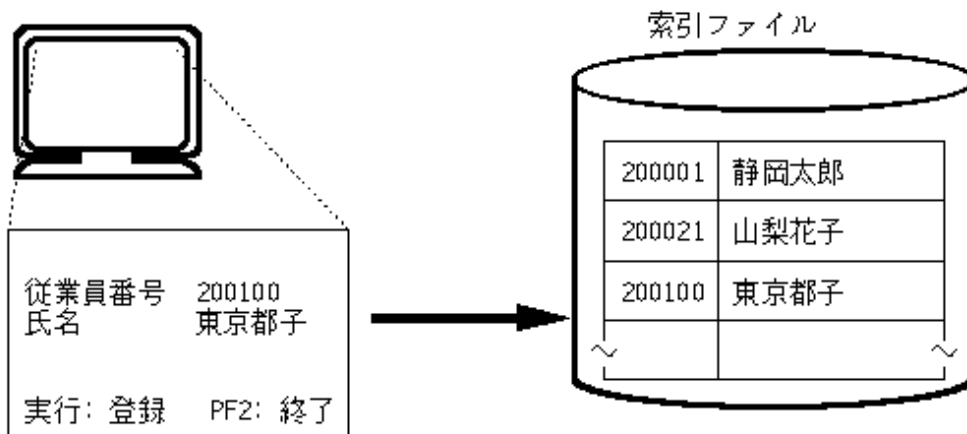
参考

索引ファイルを作成するときのデータの収集方法として次の機能を使うと便利です。

- ー エディタでデータを作成し、行順ファイル機能を使ってそのデータを読み込み、索引ファイルに書き出します。



- ー 画面処理機能(表示ファイル機能・スクリーン操作機能)を使って、画面からデータを入力し、索引ファイルに書き出します。



拡張[順]

索引ファイルの拡張は、ファイルを拡張モードで開いて、順番にレコードを書き出します。このとき、ファイルの最後のレコードの後ろにレコードが追加されます。ファイルの拡張が可能なアクセス形態は、順呼出し法だけです。

```
OPEN EXTEND   ファイル名.
               レコードの編集処理
WRITE   レコード名  ~.
CLOSE   ファイル名.
```

注意

書き出すレコードの内容を編集するときに、主キーの値が昇順となるように設定する必要があります。また、ファイル管理記述項の **RECORD KEY** 句に **DUPLICATES** 指定がない場合、最初に処理する主キーの値は、そのファイルに存在する最大の主キーの値より大きくなければなりません。 **DUPLICATES** 指定がある場合、そのファイルに存在する最大の主キーの値と等しいか、それより大きくなければなりません。

参照〔順/乱/動的〕

レコードの参照では、ファイルを入力モードで開いて、READ文でファイルのレコードを読み込みます。順呼出しの場合、START文を使って読み込むレコードの開始位置を指定し、そのレコードから主キーまたは副キーの値で昇順に読み込んでいきます。乱呼出しの場合、読み込まれるレコードは、主キーまたは副キーの値により決定されます。

順呼出し

```
OPEN INPUT   ファイル名.  
MOVE   キーの値 TO キー名.  
START   ファイル名.  
READ   ファイル名 [NEXT] ~.  
CLOSE   ファイル名.
```

乱呼出し

```
OPEN INPUT   ファイル名.  
MOVE   キーの値 TO キー名.  
READ   ファイル名 ~.  
CLOSE   ファイル名.
```



注意

- ファイル管理記述項のSELECT句にOPTIONALを指定した場合、OPEN文実行時にファイルが存在していなくてもOPEN文は成功となり、最初のREAD文の実行でファイル終了条件が成立します。ファイル終了条件については、“[7.6 入出力エラー処理](#)”を参照してください。
- 乱呼出し法または動的呼出し法で、指定したキーの値を持つレコードが存在しないときに無効キー条件が成立します。無効キー条件については、“[7.6 入出力エラー処理](#)”を参照してください。
- 複数のキーを指定してSTART文またはREAD文を実行することもできます。

更新〔順/乱/動的〕

レコードの更新では、ファイルを入出力モードで開いて、ファイルのレコードを書き換えます。順呼出しの場合、直前のREAD文により読み込まれたレコードの内容が変更されます。乱呼出しの場合、設定したキー値を持つ主キーのレコードの内容が変更されます。

順呼出し

```
OPEN I-O   ファイル名.  
MOVE   キーの値 TO キー名.  
START   ファイル名.  
READ   ファイル名 [NEXT] ~.  
レコードの編集処理  
REWRITE   レコード名 ~.  
CLOSE   ファイル名.
```

乱呼出し

```
OPEN I-O   ファイル名.  
MOVE   キーの値 TO キー名.  
[READ   ファイル名 ~.]  
レコードの編集処理  
REWRITE   レコード名 ~.  
CLOSE   ファイル名.
```



注意

- 乱呼出し法または動的呼出し法で、指定したキーの値を持つレコードが存在しないときに無効キー条件が成立します。無効キー条件については、“[7.6 入出力エラー処理](#)”を参照してください。

ー 副キーの内容を変更することはできますが、主キーの内容を変更することはできません。

削除〔順/乱/動的〕

レコードの削除では、ファイルを入出力モードで開いて、ファイルのレコードを削除します。順呼出しの場合、直前のREAD文により読み込まれたレコードが削除されます。乱呼出しの場合、設定したキー値を持つ主キーのレコードが削除されます。

順呼出し

```
OPEN I-O   ファイル名.  
MOVE      キーの値 TO キー名.  
START     ファイル名.  
READ      ファイル名 [NEXT] ~.  
DELETE    ファイル名 ~.  
CLOSE     ファイル名.
```

乱呼出し

```
OPEN I-O   ファイル名.  
MOVE      キーの値 TO キー名.  
DELETE    ファイル名 ~.  
CLOSE     ファイル名.
```



注意

乱呼出し法または動的呼出し法で、指定したキーの値を持つレコードが存在しないときに無効キー条件が成立します。無効キー条件については、“[7.6 入出力エラー処理](#)”を参照してください。

挿入〔乱/動的〕

レコードの挿入では、ファイルを入出力モードで開いて、ファイルにレコードを挿入します。レコードの挿入される位置は、主キーの値によって決定されます。

```
OPEN I-O   ファイル名.  
レコードの編集処理  
MOVE      キーの値 TO キー名.  
WRITE     レコード名 ~.  
CLOSE     ファイル名.
```



注意

ファイル管理記述項のRECORD KEY句またはALTERNATE RECORD KEY句にDUPLICATES指定がない場合、指定したキーの値を持つレコードがすでに存在するとき、無効キー条件が成立します。無効キー条件については、“[7.6 入出力エラー処理](#)”を参照してください。

7.6 入出力エラー処理

ここでは、入出力エラーの検出方法および入出力エラーが発生したときの実行結果について説明します。入出力エラーの検出方法には、次の4つがあります。

- ・ AT END指定(ファイル終了条件発生の検出)
- ・ INVALID KEY指定(無効キー条件発生の検出)
- ・ FILE STATUS句(入出力状態のチェックによる入出力エラーの検出)
- ・ 誤り処理手続き(入出力エラーの検出)

7.6.1 AT END指定

ファイル中のレコードを順番に読み、すべてのレコードを読み終わったときに、次に読み込むレコードが存在しないとファイル終了状態になります。これを、ファイル終了条件の発生といいます。ファイル終了条件の発生を検出するには、READ文にAT END指定を記述します。AT END指定には、ファイル終了条件が発生したときに行う処理を記述することができます。



例

AT END指定のREAD文の記述例

```
READ  順ファイル AT END
      GO TO  ファイルの終了処理
END-READ.
```

ファイルの最後のレコードを読んだ次のREAD文の実行で、ファイル終了条件が発生し、GO TO文が実行されます。

7.6.2 INVALID KEY指定

索引ファイルまたは相対ファイルの入出力処理で、指定したキーや相対レコード番号を持つレコードが存在しなかった場合、入出力エラーとなります。これを、無効キー条件の発生といいます。無効キー条件の発生を検出するには、READ文、WRITE文、REWRITE文、START文およびDELETE文にINVALID KEY指定を記述します。INVALID KEY指定には、無効キー条件が発生したときに行う処理を記述することができます。



例

INVALID KEY指定のREAD文の記述例

```
MOVE  "Z" TO  主キー.
READ  索引ファイル INVALID KEY
      GO TO  無効キーの処理
END-READ.
```

主キーの値に“Z”を持つレコードが存在しないとき、無効キー条件が発生し、GO TO文が実行されます。

7.6.3 FILE STATUS句

ファイル管理記述項にFILE STATUS句を記述すると、入出力文実行時に、FILE STATUS句に指定したデータ名に入出力状態が通知されます。入出力文の後に、このデータ名の内容(入出力状態値)をチェックする文(IF文またはEVALUATE文)を記述することによって、プログラムで入出力文の結果に応じた処理手続きを実行することができます。ただし、入出力文の後に入出力状態値をチェックしない場合、入出力エラーが発生してもプログラムの後続処理の実行が続けられる場合があるため、その後の動作は保証されません。

通知される入出力状態値については、“付録B 入出力状態一覧”を参照してください。入出力状態の成功の分類には、入出力文の実行は成功していても、その入出力の結果について何らかの情報が通知されるものが含まれます。



例

FILE STATUS句の記述例

```
SELECT  ファイル
      FILE STATUS IS  入出力状態値
      :
WORKING-STORAGE SECTION.
01  入出力状態値  PIC X(2).
      :
OPEN  INPUT  ファイル.
IF  入出力状態値 NOT = "00"
  THEN GO TO  オープン失敗の処理.
```

ファイルのオープンに失敗すると、入出力状態値に“00”以外の値が設定されます。IF文でこの値をチェックしているので、オープンに失敗した場合にはGO TO文が実行されます。

7.6.4 誤り処理手続き

手続き部の宣言節部分で、USE AFTER ERROR/EXCEPTION文を記述することにより、誤り処理手続きを指定することができます。誤り処理手続きを記述すると、入出力エラー発生時に誤り処理手続きに記述した処理が実行されます。誤り処理を実行後、入出力エラーが発生した入出力文の直後の文に制御が渡るので、入出力文の直後には、入出力エラーが発生したファイルの処理の制御を指示する文を記述する必要があります。ここでの入出力エラーとは、入出力状態の成功の分類に含まれるもの以外の条件が発生したことを示します。

以下の場合には、誤り処理手続きに制御は渡りません。

- ・ ファイル終了条件が発生したREAD文にAT END指定がある場合
- ・ 無効キー条件が発生した入出力文にINVALID KEY指定がある場合
- ・ ファイルが開かれる前に入出力文を実行した場合(オープンモードの指定をした場合)

以下の場合、誤り処理手続き中から手続き中へGO TO文を使って分岐してください。

- ・ 誤り処理手続きの中で、入出力エラーが発生したファイルに対して入出力文を実行した場合
- ・ 誤り処理手続きが終了する前に、その誤り処理手続きが再び実行された場合

```
PROCEDURE DIVISION.  
  DECLARATIVES.  
    誤り処理 SECTION.  
      USE AFTER ERROR PROCEDURE ON   ファイル.  
        MOVE   エラー発生 TO   ファイルの状態.           ... [1]  
  END DECLARATIVES.  
  :  
  OPEN  INPUT   ファイル.  
  IF   ファイルの状態 =   エラー発生           ... [2]  
    THEN GO TO   オープン失敗の処理.  
  :
```

ファイルのオープンに失敗すると、誤り処理手続き([1]のMOVE文)が実行され、OPEN文の直後の文([2]のIF文)に制御が渡ります。

7.6.5 入出力エラーが発生したときの実行結果

入出力エラーが発生したときの実行結果は、AT END指定の有無、INVALID KEY指定の有無、FILE STATUS句の有無および誤り処理手続きの有無によって異なります。入出力エラーが発生したときの実行結果を“表7.4 入出力エラーが発生したときの実行結果”に示します。ここでの入出力エラーとは、入出力状態の成功の分類に含まれるもの以外の条件が発生したことを示します。

表7.4 入出力エラーが発生したときの実行結果

誤り処理手続き	有	有	有	有	無	無	無	無
FILE STATUS 句	有	無	有	無	有	無	有	無
AT END指定 または INVALID KEY 指定	有	有	無	無	有	有	無	無
ファイル終了条件発生時 または 無効キー条件発生時の処理	1	1	2	2	1	1	3	5
その他の入出力エラー発生時の処理	2	2	2	2	4	5	4	5

- 1 : AT END指定／INVALID KEY 指定に記述した文が実行されます。
- 2 : 誤り処理手続きが実行された後、エラーが発生した入出力文の直後の文から処理が続行されます。
- 3 : エラーが発生した入出力文の直後の文から処理が続行されます。
- 4 : Iレベルメッセージが出力されて、エラーが発生した入出力文の直後の文から処理が続行されます。
- 5 : Uレベルメッセージが出力されて、プログラムが異常終了します。

参考

- 有効な誤り処理手続きがある場合、環境変数情報@CBR_FILE_USE_MESSAGE=YESを指定すると、Iレベルメッセージが出力されます。

```
@CBR_FILE_USE_MESSAGE=YES
```

- 環境変数情報の指定誤りを示す入出力エラーが発生した場合、Uレベルの実行時メッセージを出力させて、診断機能を利用してアプリケーション実行時に設定されていた環境変数および実行環境情報を確認する方法もあります。

診断機能で診断レポートを出力するには、“表7.4 入出力エラーが発生したときの実行結果”の組合せの通り、誤り処理手続きとFILE STATUS句の指定が“無”でなければなりません。この場合は、プログラムのデバッグを目的として、一時的にプログラムを修正する必要があります。

診断機能を利用したプログラムのデバッグについては、“19.6 診断機能の使い方”を参照してください。

7.7 ファイル処理の実行

ここでは、ファイルの割当て、ファイル処理の結果、ファイルの排他制御およびファイル処理を向上させるための方法について説明します。

7.7.1 ファイルの割当て

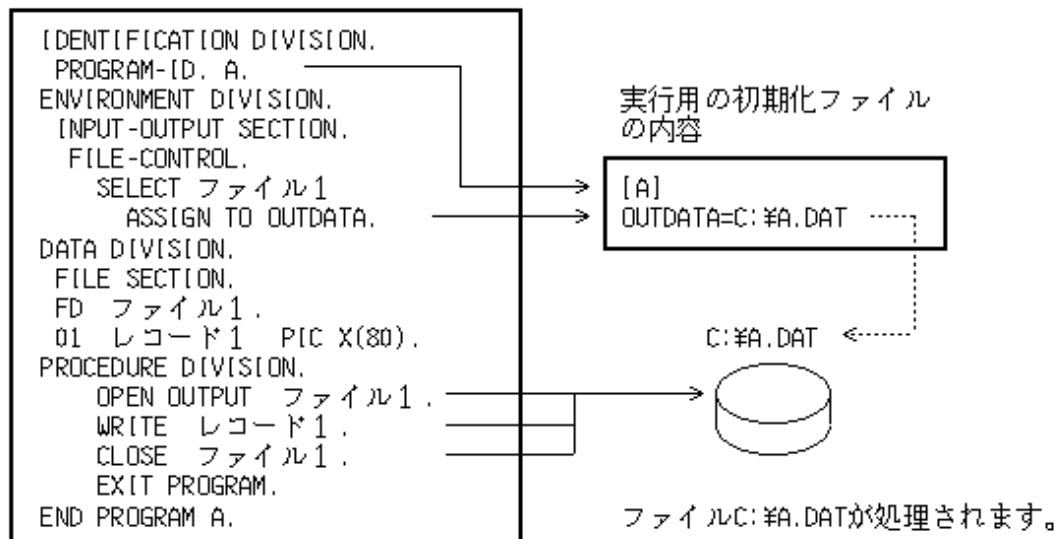
プログラムの実行時に入出力処理の対象となるファイルの決定方法は、ファイル管理記述項のASSIGN句の記述内容によって異なります。ASSIGN句の記述内容とファイルの関係を以下に示します。

ASSIGN句にファイル識別名を記述した場合

ファイル識別名を環境変数情報名として、プログラム実行時に、入出力処理の対象となるファイルの名前を設定します。環境変数情報の設定方法については、“5.3 実行環境情報の設定”を参照してください。

例

実行用の初期化ファイルを使った環境変数情報を設定する場合の例



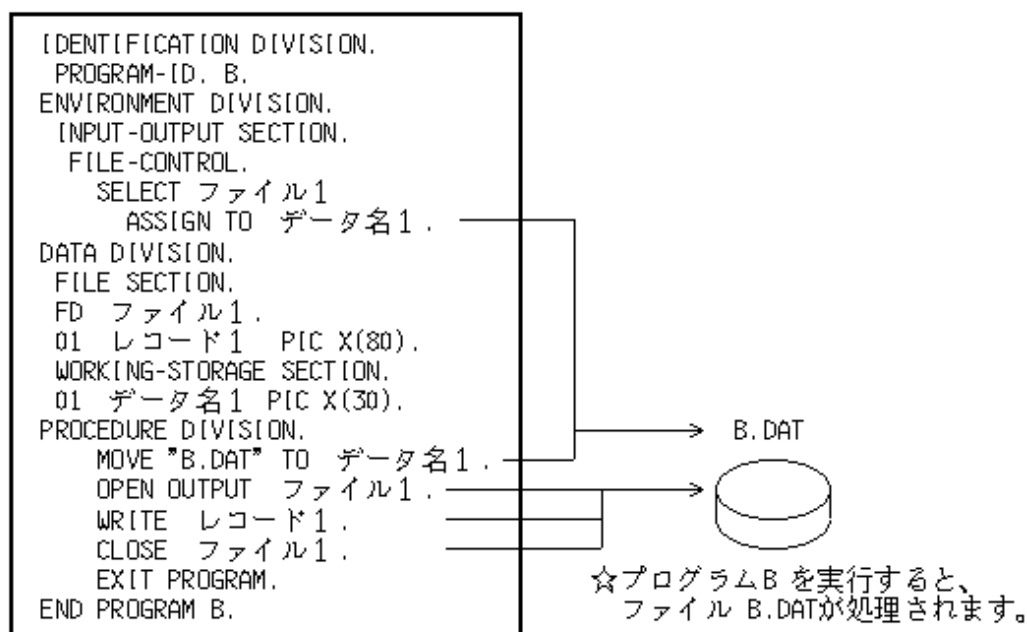
注意

- ASSIGN句に英小文字のファイル識別名を指定した場合、以下の注意が必要です。
 - 翻訳オプションNOALPHALを指定して翻訳すると、翻訳エラーとなります。
 - 環境変数情報を設定するときには、英大文字で指定してください。
- 環境変数情報の内容が空白の場合、ファイルの割当てエラーとなります。

ASSIGN句にデータ名を記述した場合

データ名に設定されたファイル名のファイルに対して入出力処理が行われます。

例



注意

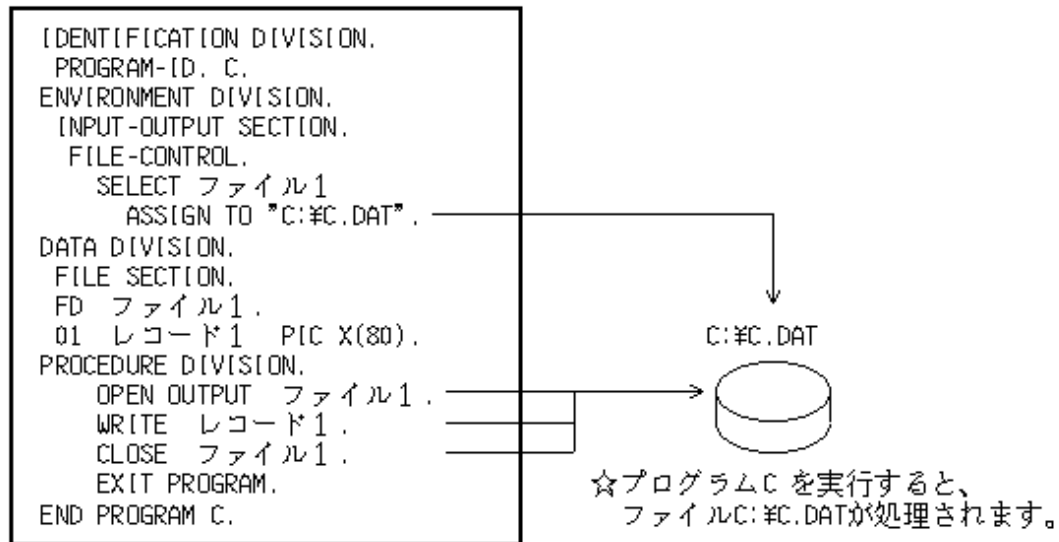
- プログラムに記述されたファイル名が相対パス名の場合、カレントフォルダを先頭に付加したファイルが入出力処理の対象となります。
- データ名の内容が空白の場合、ファイルの割当てエラーとなります。

ASSIGN句にファイル識別名定数を記述した場合

ファイル識別名定数として記述したファイル名のファイルに対して入出力処理が行われます。



例



注意

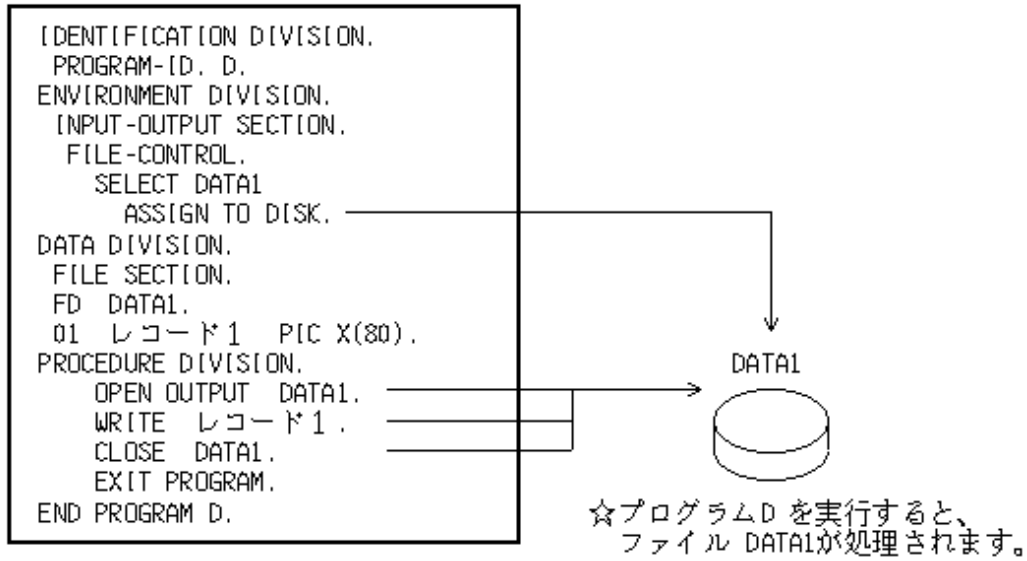
プログラムに記述されたファイル名が相対パス名の場合、カレントフォルダを先頭に付加したファイルが入出力処理の対象となります。

ASSIGN句に文字列DISKを記述した場合

SELECT句に記述したファイル名のファイルに対して入出力処理が行われます。



例



注意

ファイルはカレントフォルダが対象となります。

7.7.2 ファイルの排他制御

ファイル処理では、ファイル自体を排他モードにしたり、使用中のレコードを排他状態にすることにより、他からのアクセスを不可能にすることができます。これをファイルの排他制御といいます。

ファイルの排他制御は、COBOLプログラム、COBOLファイルアクセスルーチンまたはCOBOLファイルユーティリティを使用したアクセスに対して有効です。他言語や各種ツールからのアクセスでは、ファイルの排他制御は無効となり、同時にアクセスした場合の動作は保証されません。

ここでは、ファイル処理とファイルの排他制御の関係について説明します。

7.7.2.1 ファイルを排他モードにする方法

ファイルを排他モードで開くと、他の使用者はそのファイルをアクセスすることができません。

次の場合、ファイルは排他モードで開かれます。

- ・ ファイル管理記述項のLOCK MODE句にEXCLUSIVEを指定したファイルに対してOPEN文を実行します。
- ・ ファイル管理記述項のLOCK MODE句を指定しないファイルに対してINPUTモード以外のOPEN文を実行します。
- ・ WITH LOCK指定のOPEN文を実行します。
- ・ OUTPUTモードのOPEN文を実行します。

上記組合せによる、ファイルのモードの状態を下表に示します。

表7.5 LOCK MODE句が指定されていない場合

OPEN文のモード	入力INPUT		入出力I-O		拡張EXTEND		出力OUTPUT	
OPEN文のWITH LOCK 指定	なし	あり	なし	あり	なし	あり	なし	あり
ファイルのモード	共用	排他	排他	排他	排他	排他	排他	排他

表7.6 LOCK MODE句にEXCLUSIVEが指定されている場合

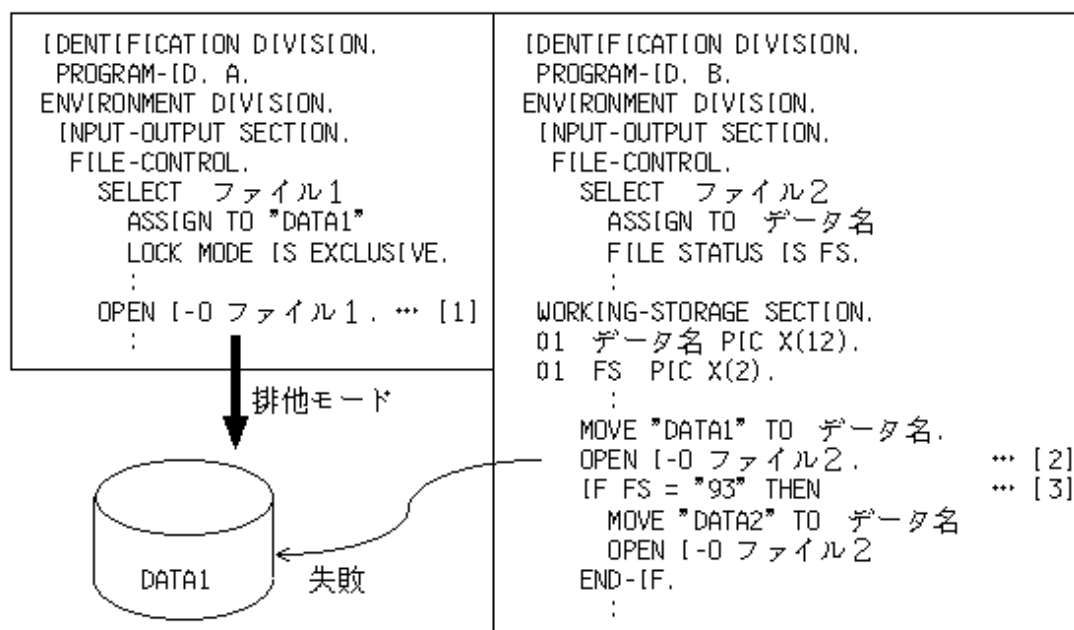
OPEN文のモード	入力INPUT		入出力I-O		拡張EXTEND		出力OUTPUT	
OPEN文のWITH LOCK 指定	なし	あり	なし	あり	なし	あり	なし	あり
ファイルのモード	排他	排他	排他	排他	排他	排他	排他	排他

表7.7 LOCK MODE句にAUTOMATICまたはMANUALが指定されている場合

OPEN文のモード	入力INPUT		入出力I-O		拡張EXTEND		出力OUTPUT	
OPEN文のWITH LOCK 指定	なし	あり	なし	あり	なし	あり	なし	あり
ファイルのモード	共用	排他	共用	排他	共用	排他	排他	排他



例



- [1] 排他モードでファイル(DATA1)を開きます。
- [2] プログラムAですでに排他モードで使用されているファイル(DATA1)に対して、OPEN文を実行しても失敗となります。
- [3] FILE STATUS句に指定したデータ名に入出力状態値“93”(ファイルの排他によるエラー発生)が設定されます。

7.7.2.2 レコードを排他状態にする方法

任意のレコードを排他状態にすると、他の利用者はそのレコードを処理することができません。任意のレコードを排他状態にするためには、まず、ファイルを共用モードで開きます。

次の場合、ファイルは共用モードで開かれます。

- ・ ファイル管理記述項のLOCK MODE句にAUTOMATICまたはMANUALを指定したファイルに対して、OUTPUTモード以外のWITH LOCK指定なしのOPEN文を実行します。
- ・ LOCK MODE句を指定しないファイルに対してINPUTモードのOPEN文を実行します。

共用モードで開かれたファイルは、他の利用者と共用して使用することができます。ただし、すでに他の利用者がそのファイルを排他モードで使用しているとき、OPEN文は失敗となります。共用モードで開かれたファイルのレコードは、排他処理を指定した入出力文の実行により排他状態となります。

次の場合、レコードが排他状態となります。

- ・ ファイル管理記述項のLOCK MODE句にAUTOMATICを指定したファイルを入出力モードで開き、WITH NO LOCK指定なしのREAD文を実行します。
- ・ ファイル管理記述項のLOCK MODE句にMANUALを指定したファイルを入出力モードで開き、WITH LOCK指定ありのREAD文を実行します。

上記組合せによる、レコードの状態を下表に示します。

表7.8 レコードの状態(排他/共用)

LOCK MODE 句の記述	AUTOMATIC			MANUAL		
READ文の記述	記述なし	WITH LOCK	WITH NO LOCK	記述なし	WITH LOCK	WITH NO LOCK
レコードの排他状態	する	する	しない	しない	する	しない

また、次の場合、レコードの排他状態が解除されます。

LOCK MODE句にAUTOMATICを指定したファイルの場合

- READ文/REWRITE文/WRITE文/DELETE文/START文を実行します。
- UNLOCK文を実行します。
- CLOSE文を実行します。

LOCK MODE句にMANUALを指定したファイルの場合

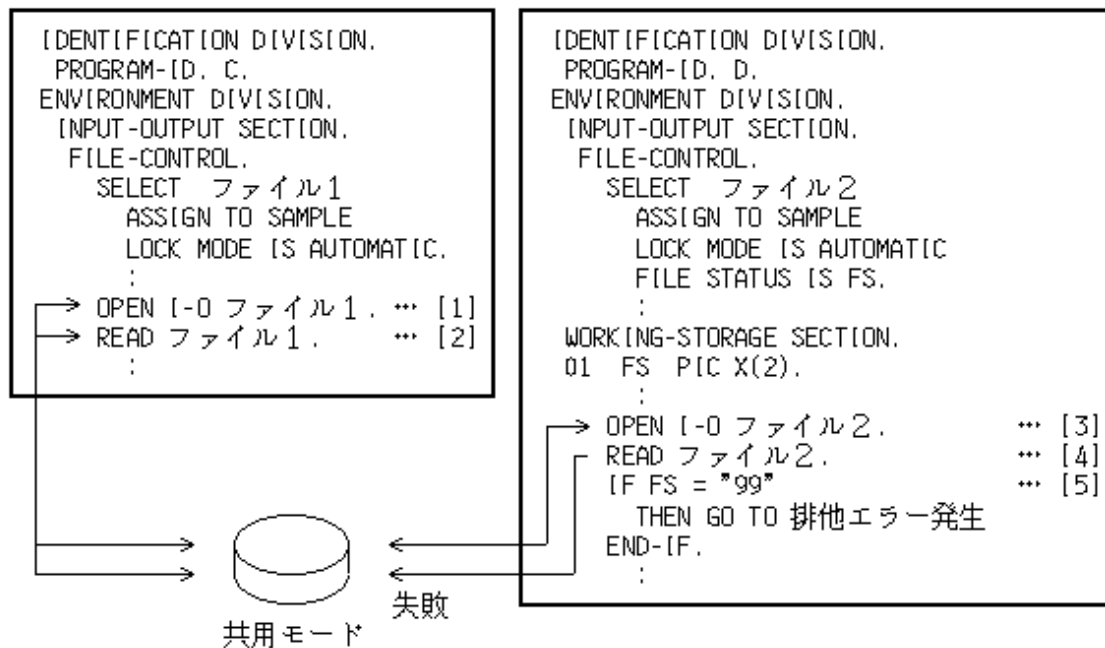
- UNLOCK文を実行します。
- CLOSE文を実行します。



例

.....

レコードの排他処理



- [1] 共用モードでファイルを開きます。
- [2] READ文の実行により、ファイルの先頭レコードは排他状態となります。
- [3] 共用モードでファイルを開きます。
- [4] 排他状態となっているレコードに対するREAD文は失敗となります。
- [5] FILE STATUS句に指定したデータ名に入出力状態値“99”(レコードの排他によるエラー発生)が設定されます。

7.7.3 ファイル処理の結果

ファイル処理を行うことにより、新しいファイルが作られたり、ファイル中の内容が書き換えられたりします。ここでは、ファイル処理を行ったときのファイルの状態について説明します。

ファイルの創成処理を行ったとき

創成処理では、OPEN文の実行により、新しいファイルが作られます。このとき、同じファイル名のファイルがすでに存在している場合、そのファイルは新しく作り直され、元の内容は失われます。

ファイルの拡張処理を行ったとき

既存ファイルに対しての拡張処理では、WRITE文の実行によってファイルが拡張されます。プログラム実行時に存在しないファイル(不定ファイル)に対しての拡張処理では、OPEN文の実行により、新しいファイルが作られます。

レコードの参照処理を行ったとき

参照処理では、参照したファイルの内容は変更されません。不定ファイルに対しての参照処理では、最初のREAD文でファイル終了条件が発生します。

レコードの更新/削除/挿入処理を行ったとき

既存ファイルに対しての更新/削除/挿入処理では、REWRITE文/DELETE文/WRITE文の実行により、ファイルの内容が変更されます。不定ファイルに対しての更新/削除/挿入処理では、OPEN文の実行により、新しいファイルが作られます。ただし、このファイルにはデータが存在しないので、最初のREAD文でファイル終了条件が発生します。



注意

- ・ CLOSE文を実行しないでプログラムを終了すると、そのファイルは強制的に閉じられます。これを強制クローズといいます。強制クローズは、以下の場合に行なわれます。
 - － STOP RUN文の実行
 - － 主プログラムでのEXIT PROGRAM文の実行
 - － 外部プログラムに対するCANCEL文の実行
 - － JIMPCINT3の呼出し
- ・ もし、強制クローズに失敗した場合、メッセージが出力され、そのファイルは開かれた状態のまま使用不可能となります。
- ・ ファイル識別名を環境変数情報名としてファイルを割り当て、ファイルが開かれた状態のまま環境変数操作機能によりファイルの割当て先を変更しても、入出力文はファイル識別名で割り当てたファイルに対して行われます。ファイル識別名で割り当てたファイルをCLOSE文で閉じて、OPEN文を実行すると、その後の入出力文は環境変数操作機能で変更したファイルに対して行われます。ファイルの一連の処理は、OPEN文で開始し、CLOSE文で終了するようにしてください。
- ・ ファイルの創成・拡張処理またはレコードの更新・挿入処理で領域不足が発生した場合、それ以降のそのファイルに対する処理の正常な動作は保証できません。また、領域不足発生時に書き出そうとしたレコードの内容が、ファイル内にどのように格納されるかは規定できません。
- ・ 索引ファイルをOUTPUT、I-OまたはEXTENDモードで開いているとき、ファイルを閉じる前にプログラムが異常終了すると、そのファイルが使用できなくなることがあります。異常終了する可能性のあるプログラムを実行する前には、事前にバックアップすることをおすすめします。なお、使用できなくなったファイルは、COBOLファイルユーティリティの[復旧]コマンドにより、再び使用できる状態に復旧することができます。また、そのコマンドと同じ機能を持つ関数をアプリケーションから呼び出すこともできます。[参照] “7.8 COBOLファイルユーティリティ”、“7.10 索引ファイルの復旧”

7.7.4 ファイルの高速処理

レコード順ファイルおよび行順ファイルについて、使用範囲を限定することでアクセス性能を高速化することができます。

本機能は、以下のような場合に使用すると効果的です。

- ・ ファイルを排他モードでOPENし、出力ファイルとして書き込みのみを行う
- ・ 入力専用ファイルに対し、読み込みを行う

ここでは、ファイルの高速処理を行うために必要な指定方法および注意事項について説明します。

指定方法

ファイルの高速処理を行うために必要な指定方法を、以下に示します。なお、レコード順ファイルおよび行順ファイルの指定方法は同じです。

プログラム中のファイル参照子にファイル識別名を指定する場合

環境変数情報の設定時に、割り当てるファイルのファイル名に続き“BSAM”を指定します。環境変数情報の設定方法については、“5.4.1.69 ファイル識別名 (プログラムで使用するファイルの指定)”を参照してください。

```
ファイル識別名= [パス名] ファイル名, BSAM
```

プログラム中のファイル参照子にデータ名を指定する場合

プログラム中のデータ名の指定時に、割り当てるファイルのファイル名に続き“BSAM”を指定します。ドライブ名を省略した場合、カレントドライブとみなされます。

```
MOVE “[パス名] ファイル名, BSAM” TO データ名.
```

プログラム中のファイル参照子にファイル識別名定数を指定する場合

プログラム中のファイル識別名定数の指定時に、割り当てるファイルのファイル名に続き“BSAM”を指定します。

```
ASSIGN TO “[パス名] ファイル名, BSAM”
```

注意

- レコードの更新(REWRITE文)は、できません。レコードの更新を行った場合は、実行時にエラーとなります(レコード順ファイルだけ)。
- ファイル共用する場合には、以下の注意が必要です。

他プロセス間でのファイル共用は、すべてのファイルが共用モードで、かつINPUT指定で開かれている必要があります。INPUT指定以外で開いたファイルがある場合、動作は保証されません。また、同一プロセス内はファイル共用できません。同一プロセス内でファイル共用した場合、動作は保証されません。

なお、OPENモードがINPUT指定以外の場合は、常に排他モードでOPENします。このため、他のプログラムからアクセスした際、OPENエラーになる場合があります。[参照]“[ファイル共用時の注意事項](#)”
- ファイル参照子にDISKを指定した場合、この機能は使用できません。
- 行順ファイルの場合、読み込んだレコードにタブが存在していても、そのタブコードを空白に置き換えません。また、制御文字(0x0C(改頁)、0x0D(復帰)、0x1A(データ終了記号))が含まれていても、レコードの区切り文字やファイルの終端として扱いません。ファイルの高速処理を指定しない場合のタブや制御文字の扱いについては、“[7.3.3 行順ファイルの処理](#)”の“レコード内のタブの扱い”および“レコード内の制御文字の扱い”を参照してください。
- レコードの書込み(WRITE文)におけるADVANCING指定は有効となりません。指定した場合は、ADVANCING指定のないWRITE文と同じ結果となります。

一括指定

ファイルの高速処理を一括して有効とする指定方法について説明します。

使い方

ファイルの高速処理を有効とする場合、環境変数情報@CBR_FILE_SEQUENTIAL_ACCESSに“BSAM”を指定します。

```
@CBR_FILE_SEQUENTIAL_ACCESS = BSAM
```

本環境変数は、以下のファイルに対して有効になります。

ファイル編成	- レコード順ファイル - 行順ファイル
ASSIGN句の指定	- ファイル識別名 - ファイル識別名定数 - データ名 - DISK

ただし、以下の機能を指定したファイルに対しては、有効になりません。

機能	ファイル機能名
ファイルの追加書き(注)	MOD
ファイルの連結(注)	CONCAT
ダミーファイル	DUMMY
他のファイルシステム	BTRV
	RDM
	EXFH
	(XLデータパイプ)

注:ファイルの高速処理を同時に有効にしたい場合は、ファイル単位に指定してください。ファイル単位に指定する方法は、“7.7.8 注意事項”を参照してください。

注意

本環境変数の指定により、ファイルの高速処理の制限事項が適用されます。

特に、ファイルを共用モードでOPENしている場合、アプリケーションの動作が変わる場合があります。制限事項に該当するファイルがある場合、本環境変数を指定しないでください。

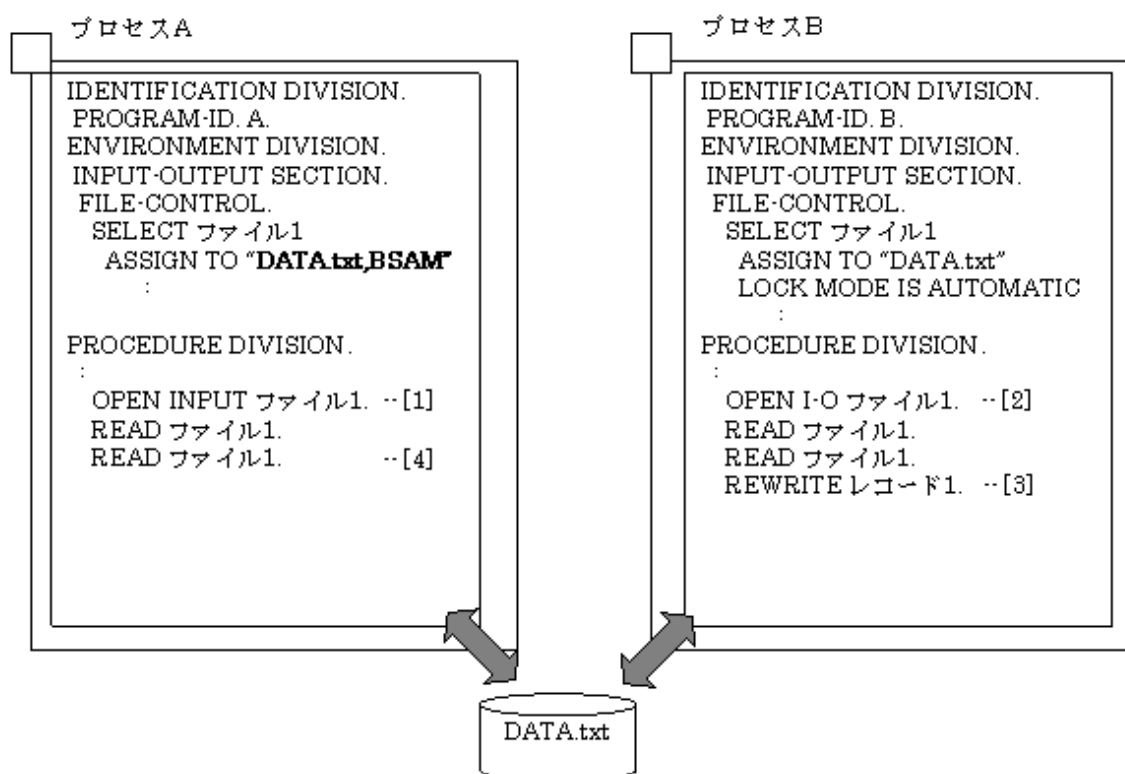
[参照]“ファイル共用時の注意事項”

ファイル共用時の注意事項

ファイルを共用する際に問題が発生する例を、以下に示します。

例

例1) 他プロセスからレコードが更新される運用環境の場合、本機能は使用できません。



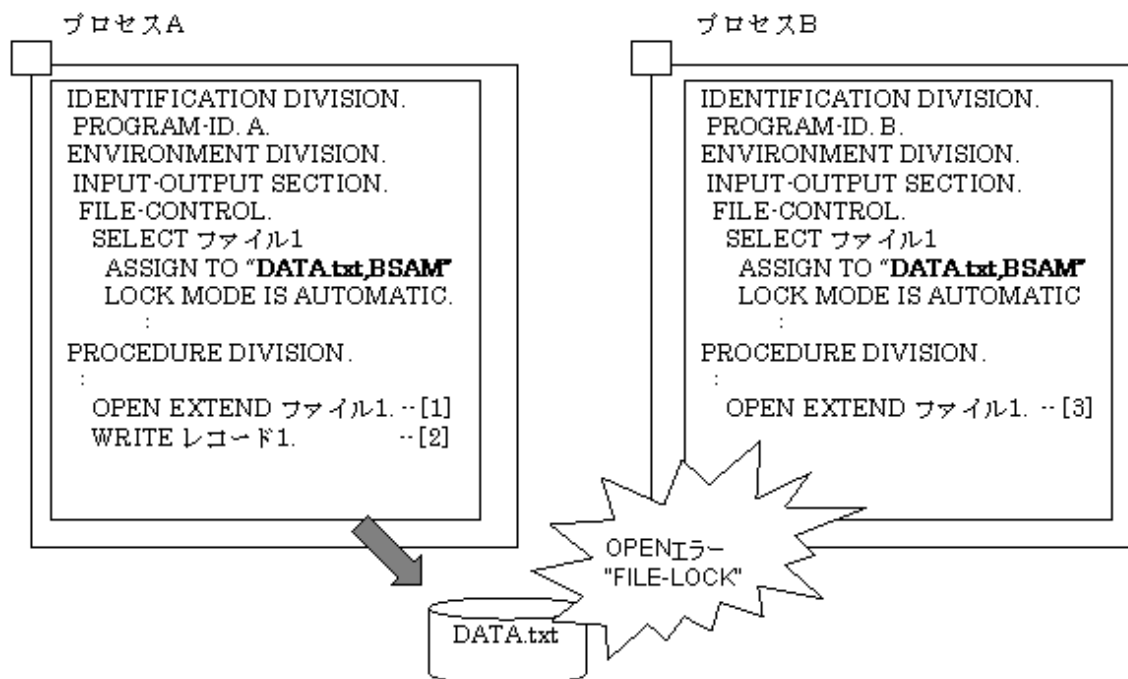
[1] プロセスA：共用モード、INPUT指定でファイルを開きます。(ファイルの高速処理)

[2] プロセスB：共用モード、I-O指定でファイルを開きます。

[3] プロセスB：2件目のレコードを更新します。

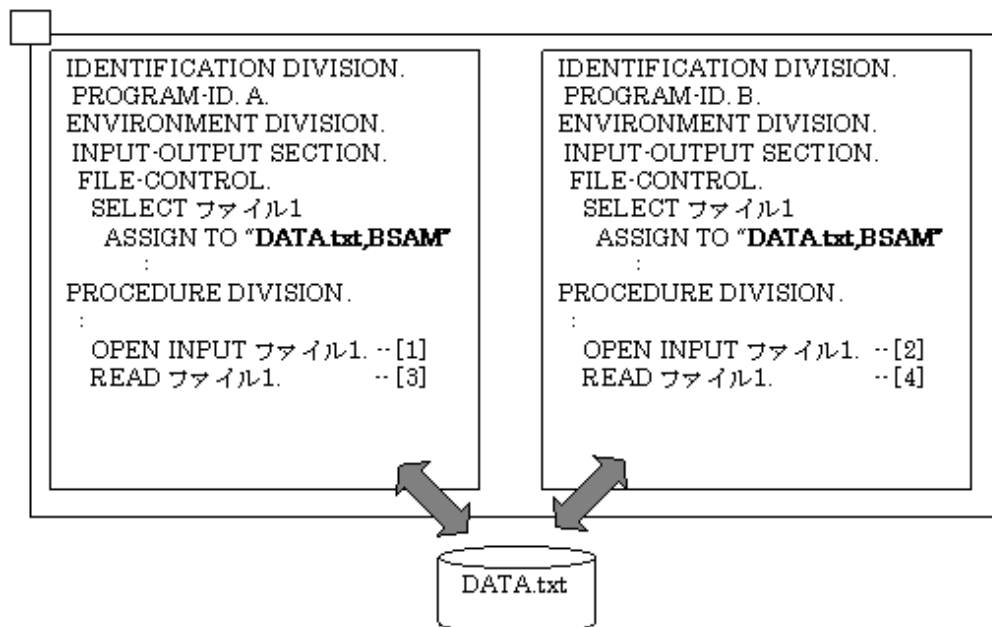
[4] プロセスA：2件目のレコードを読み込みます。ここで、更新前のデータが読み込まれる可能性があります。

例2) ファイルを共用して書き込みする場合、後続のOPENがエラーになります。



- [1] プロセスA：共用モード、EXTEND指定でファイルを開きます。
- [2] プロセスA：レコードを書き出します。
- [3] プロセスB：共用モード、EXTEND指定でファイルを開きます。ここで、OPEN文がエラーになります。

例3) 同一プロセス内でファイルを共用する場合、本機能は使用できません。



- [1] プログラムA：共用モード、INPUT指定でファイルを開きます。
- [2] プログラムB：共用モード、INPUT指定でファイルを開きます。
- [3] プログラムA：1件目のレコードを読み込みます。

.....

使い方

ファイル識別名=ファイル名.,MOD

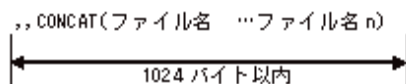


使い方

ファイル識別名=, , CONCAT (ファイル名1 ファイル名2··)



- ファイル識別子にファイル連結機能だけを指定する場合は、以下のように指定してください。



ダミーファイルは以下のような場合に使用すると便利です。

- ・ 出力ファイルが不要な場合
- ・ プログラムの開発途中で入力ファイルがない場合

出力ファイルが不要な場合の例として、トラブル発生時のログファイルを出力する場合があります。通常の運用時はダミーファイルとしてファイルの生成を抑止し、トラブル発生時にダミーファイルとしての扱いを外して、ログファイルを出力するという使い方があります。

また、プログラム開発途中では入力ファイルがない場合があります。ダミーファイルとすることで、空のファイルなどの不要なファイルを用意する手間が省け、作業の効率を向上させることができます。

ここでは、ダミーファイルの使い方と機能範囲について説明します。

使い方

ファイル識別名に、割り当てるファイルのファイル名に続き、“,DUMMY”を指定します。

ファイル名の指定は任意です。省略してもかまいません。

ファイル識別名=[ファイル名], DUMMY



注意

- ・ 文字列“DUMMY”の前にコンマ(,)が必要です。コンマ(,)がない場合、文字列“DUMMY”をファイル名とみなして、通常のファイルと同じ動作をします。エラーにはならないことに注意してください。
- ・ ファイル名の指定の有無による動作の違いはありません。指定したファイル名は意味を持ちません。指定したファイルが存在した場合も、既存ファイルに対する操作は行われません。

機能範囲

ダミーファイルを使用する場合、有効な機能範囲を以下に示します。

表7.9 ダミーファイルの機能範囲

ファイル編成	レコード順ファイル 行順ファイル 相対ファイル 索引ファイル		
オープンモード	OUTPUT EXTEND I-O INPUT		
入出力文	OPEN文	入出力文の実行が成功します。	
	CLOSE文		
	WRITE文		
	START文		
	UNLOCK文		
	READ文	順呼出し	ファイル終了条件が発生します。
		乱呼出し	無効キー条件が発生します。
	REWRITE文 DELETE文	順呼出し	先行するREAD文が不成功になるため、実行順序誤りになります。
		乱呼出し	無効キー条件が発生します。

7.7.8 注意事項

ファイル識別名に指定できる文字列のバイト数

ファイル識別名には、1024バイト以内の文字列を指定してください。

文字列が1024バイトを超えた場合は、1024バイトまでの文字列を有効とみなして処理します。

ただし、ファイル連結機能の指定の途中で文字列が1024バイトを超えた場合は、OPEN文実行時にエラーになります。

同時に指定可能なファイル機能の組合せ

ファイル機能は、以下の3つの種別に分類することができます。

種別	ファイル機能名	機能
(1) アクセス種別	BSAM	ファイルの高速処理
	DUMMY	ダミーファイル
(2) ファイルシステム種別	BTRV	Btrieveファイル
	RDM	RDMファイル
	EXFH	外部ファイルハンドラ
(3) その他	MOD	ファイルの追加書き
	CONCAT	ファイルの連結

ファイル機能は、次の形式で指定してください。指定順序が異なる場合、ファイル機能は有効になりません。

ファイル名, { (1)アクセス種別
(2)ファイルシステム種別 } ,(3)その他

同時に指定可能な組合せとその動作は、以下の通りです。

- ・ (1)アクセス種別のファイルル高速処理(BSAM)と(3)その他は、同時に指定することができます。また、いずれも有効になります。



例

.....

例1 ファイル高速処理(BSAM)とその他を同時に指定

ファイル名,BSAM,MOD

,BSAM,CONCAT(ファイル名1 ファイル名2 ...)

- ・ (1)アクセス種別のダミーファイル(DUMMY)は、すべての機能と同時に指定することができます。この場合、ダミーファイル(DUMMY)だけが有効になり、他の機能は無効になります。



例

.....

例2 ファイル高速処理(BSAM)とダミーファイル(DUMMY)を同時に指定

ファイル名,BSAM,DUMMY

例3 ファイルシステム種別とダミーファイル(DUMMY)を同時に指定

ファイル名, { BTRV
RDM
EXFH [,INF(情報ファイル名)] } ,DUMMY

例4 その他とダミーファイル(DUMMY)を同時に指定

ファイル名,,MOD,DUMMY

.,CONCAT(ファイル名1 ファイル名2 ...),DUMMY

1つ目のカンマの直後にDUMMYを指定することも可能です。

.,DUMMY,CONCAT(ファイル名1 ファイル名2 ...)

例5 ファイル高速処理(BSAM)およびその他と、ダミーファイル(DUMMY)を同時に指定

ファイル名,BSAM,MOD,DUMMY

.,BSAM,CONCAT(ファイル名1 ファイル名2 ...),DUMMY

-
- (1)アクセス種別のダミーファイル(DUMMY)および(3)その他の機能は、ASSIGN句の指定がファイル識別名以外の場合、OPEN文実行時にエラーになります。
 - 指定が有効にならない組合せを指定した場合の動作は、以下の通りです。
 - ー (2)ファイルシステム種別を先に指定した場合、後に指定した機能は無効にし、処理を続行
 - ー 上記以外の場合、OPEN文実行時にエラー

7.8 COBOLファイルユーティリティ

ここでは、COBOLファイルユーティリティについて説明します。

COBOLファイルユーティリティは、COBOLファイルシステムが持つファイル操作機能を、COBOLのアプリケーションを介することなくユーティリティのコマンドによって行うためのものです。なお、ここでは、COBOLファイルシステムが扱うファイル(レコード順/行順/相対/索引ファイル)のことを単にCOBOLファイルといいます。

7.8.1 COBOLファイルユーティリティとは

COBOLファイルユーティリティは、COBOLファイルを処理するためのユーティリティです。具体的には以下の機能を持ちます。

- 各種テキストエディタを使って作成したデータからCOBOLファイルを作成する。
- COBOLファイルに対する以下の操作を行う。
 - ー ファイルの複写/移動/削除
 - ー ファイル構造の変換
 - ー 索引ファイルの再編成/復旧/属性の表示
- COBOLファイルのレコードの操作(表示/編集/整列など)を行う。

本ユーティリティでは、これらの操作をウィンドウを使ったメニュー選択で簡単に行うことができます。また、ウィンドウを使わずに、コマンドや関数を使用してCOBOLファイルを操作することもできます。[参照]“[7.8.4 COBOLファイルユーティリティコマンドモード](#)”、“[7.8.5 COBOLファイルユーティリティ関数](#)”



-
- 本ユーティリティで各COBOLファイルに対して行うことのできる処理は、COBOLプログラムでのファイル処理と同様に、ファイル編成ごとに異なります。たとえば、レコード順ファイルおよび行順ファイルは順呼出しで処理されるため、一定の順序による処理しか行うことができなかつたり、レコードの挿入や削除を行うことができなかつたりします。[参照]“[表7.2 ファイルの種類と処理](#)”

- ・本ユーティリティのウィンドウを使い、出力ファイルに指定したファイルが既に存在していた場合、そのファイルを上書きするかどうか確認を求めます。しかし、コマンドや関数を使用した場合、上書き確認をせず、エラーとします。
- ・本ユーティリティは、ファイルの高速処理に対応していません。ファイル名に続き「,BSAM」を指定した場合、エラーとします。
- ・本ユーティリティは、操作対象となるファイルがCOBOLプログラムや他のユーティリティからアクセスされている場合、エラーとします。このため、同じファイルに対するユーティリティの同時実行はできません。なお、索引ファイルの復旧処理では、同時実行した場合の動作は保証されません。直前の操作が完了したことを確認してから再度実行してください。

7.8.2 COBOLファイルユーティリティの使い方

ここでは、COBOLファイルユーティリティの使い方を、操作手順に従って説明していきます。

1. 環境設定

- ー COBOLランタイムシステムが格納されているパス名を環境変数PATHに設定してください。
- ー 環境変数TMPまたはTEMPに、作業用一時ファイルを作成するフォルダを、ドライブ名から始まるパス名で設定してください。本ユーティリティで、[復旧]コマンドを実行した場合、指定されたフォルダに処理するファイルと同じ大きさの一時的な作業用のファイル(UTYnnnn.TMP(nnnnは英数字を示す))を作成します。
- ー 環境変数BSORT_TMPDIRに整列併合用ファイルを作成するフォルダを、ドライブ名から始まるパス名で設定してください。本ユーティリティで、[整列]コマンドを実行した場合、指定されたフォルダに一時的な作業用のファイル(~SRTnnnn.TMP(nnnnは英数字を示す))を作成します。環境変数BSORT_TMPDIRを設定していない場合、環境変数TEMPに指定されたフォルダに整列併合用ファイルを作成します。PowerSORTがインストールされている場合の作業用ファイルについては、“[12.2.4 プログラムの実行](#)”を参照してください。

2. COBOLファイルユーティリティの起動

プロジェクトマネージャの[ツール]メニューから“ファイルユーティリティ”を選択します。

3. 処理の選択

[コマンド]メニューから、実行するコマンドを選択します。選択を行うと、各コマンドを実行するためのダイアログボックスが表示されます。

各機能については、“[7.8.3 COBOLファイルユーティリティの機能](#)”を参照してください。



コマンド名	機能
変換	テキストファイルから可変長レコード形式のレコード順ファイルを作成します。または、可変長レコード形式のレコード順ファイルからテキストファイルを作成します。

コマンド名	機能
ロード	可変長レコード形式のレコード順ファイルから、任意のCOBOLファイルを作成します。また、可変長レコード形式のレコード順ファイルの全レコードを、任意の属性のファイルに拡張(レコードの追加)することができます。
アンロード	任意のCOBOLファイルから、可変長レコード形式のレコード順ファイルを作成します。
表示	任意のCOBOLファイルのレコードを1件ずつ、画面に表示します。
印刷	任意のCOBOLファイルのレコードを印刷します。
編集	任意のCOBOLファイルのレコードを1件ずつ、画面に表示して、レコード内容を編集することができます。
拡張	任意のCOBOLファイルにレコードを1件ずつ、追加することができます。
整列	任意のCOBOLファイルのレコードを整列し、その整列結果を可変長レコード形式のレコード順ファイルに格納します。
属性	索引ファイルの属性情報(レコード長など)を画面に表示します。
復旧	アクセスできなくなった索引ファイルを復旧します。
再編成	索引ファイルの未使用空間を削除し、ファイルサイズを小さくします。
複写	ファイルを複写します。
削除	ファイルを削除します。
移動	ファイルを別の場所に移動します。

4. COBOLファイルユーティリティの終了

[コマンド]メニューから“終了”を選択します。

7.8.3 COBOLファイルユーティリティの機能

COBOLファイルユーティリティは、以下に示す機能を備えています。

- ・ ファイルの創成
- ・ ファイルの拡張
- ・ レコードの表示
- ・ レコードの編集
- ・ レコードの整列
- ・ ファイルの操作(複写、削除、移動)
- ・ ファイルの印刷
- ・ ファイル構造の変換
- ・ 索引ファイルの操作(属性情報の表示、復旧、再編成)

これらの機能については、以下で説明します。なお、『』内は機能を使用するときのコマンド([]:コマンド名、[:省略可能])を示します。各コマンドのダイアログの操作方法については、ヘルプを参照してください。

7.8.3.1 ファイルの創成

各種テキストエディタを使って作成したデータを入力として、COBOLファイルを新規に作成することができます。『[変換][+[ロード]]』

図7.1 操作手順



注意

拡張指定の[ロード]コマンドでエラーとなった場合、バックアップのチェックボックスを選択していないと、既存ファイルの内容は失われます。

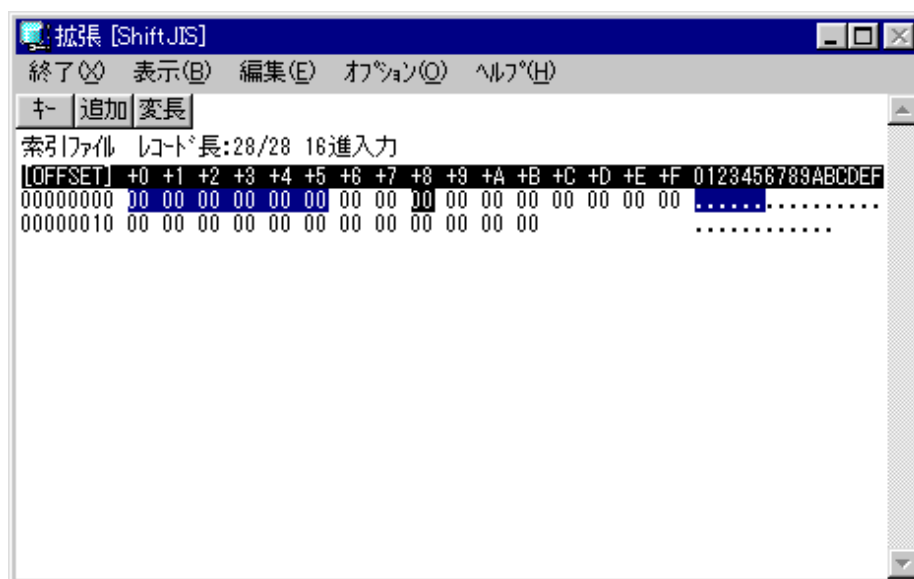
7.8.3.2 ファイルの拡張

レコードの追加および他ファイルからのレコードの追加を行うことができます。

レコードの追加

既存のCOBOLファイルにレコードを1件ずつ追加することができます。『[拡張]』

図7.2 操作手順



→ レコードを1件ずつ追加します。

メニューバー	終了	拡張ウィンドウを終了します。
	表示	次に表示するレコードをどのように指示するかを選択します。
	編集	表示中のレコードを操作します。
	オプション	レコード操作画面の入力方法(16進入力/文字入力)の指定と、フォントの指定を行います。
	ヘルプ	拡張ウィンドウのヘルプを表示します。
ボタン	キー	読み込み順序を変更する場合に選択します。読み込み順序を変更すると、画面に表示されるレコードの順番は、選択したレコードキーの順番に依存するようになります。
	追加	画面に表示しているレコードをファイルに追加する場合に選択します。
	変長	画面に表示しているレコードの長さを変更する場合に選択します。
レコード操作画面(上記画面の場合)	索引ファイル	表示中のファイルの属性を示しています。
	レコード長:28/28	表示中のレコードのレコード長/最大レコード長を示しています。
	16進入力	レコード操作画面へのデータ入力方法を示しています。
	[OFFSET]の下の文字列および“+0 +1 … +E +F”	画面に表示されているレコードデータ内の位置のオフセットを16進数で示しています。上記画面でカーソルの位置(レコードデータ内のオフセット)は、カーソル行の“+8”とカーソルの左の“00000000”を加えた0x00000008となります。
	“+0 +1 … +E +F”の下 の2個の文字列	レコードデータを1バイトごとに16進数で表示しています。
	“0123456789ABCDEF”の 下の文字列	レコードデータを1バイト表記の文字で表示しています。



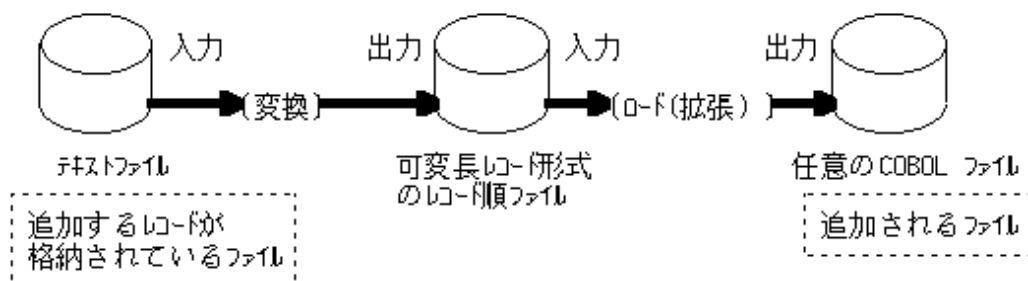
注意

日本語文字などの表示できない文字は、ピリオドに置き換えられて表示されます。
詳細は、ヘルプを参照してください。

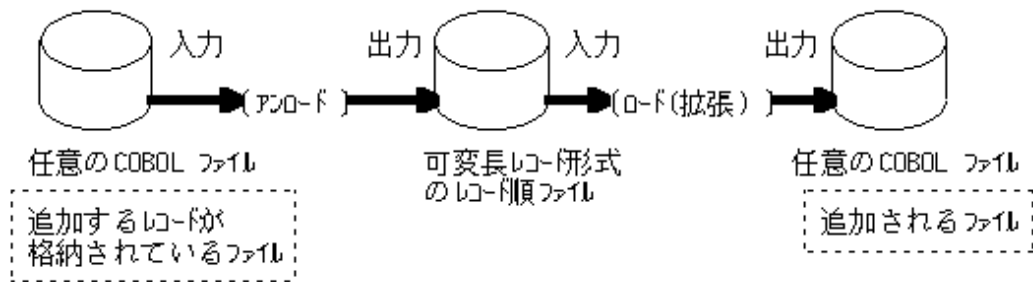
他ファイルからのレコードの追加

既存のCOBOLファイルに他のファイルの内容(レコード)を追加することができます。『[[変換]+[ロード(拡張)]または[[アンロード]+[ロード(拡張)]]』

図7.3 操作手順



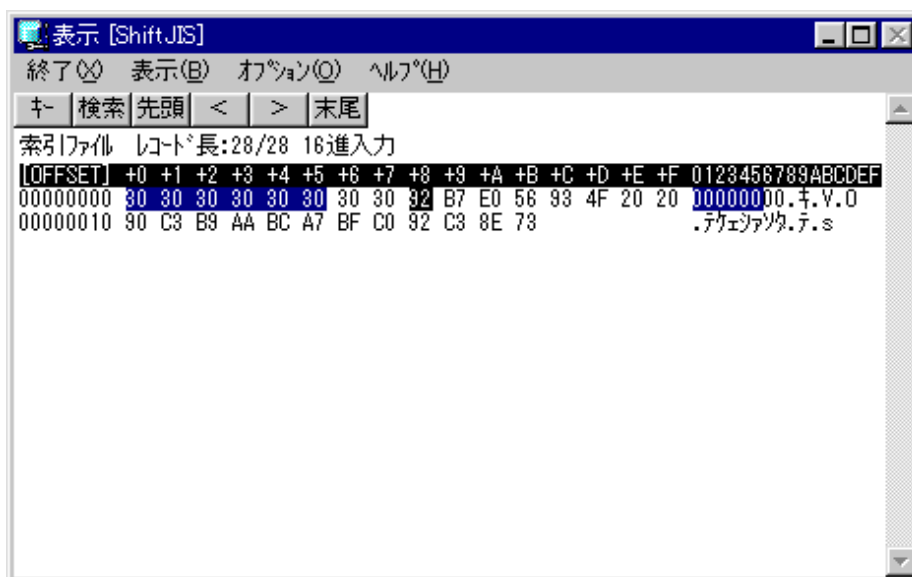
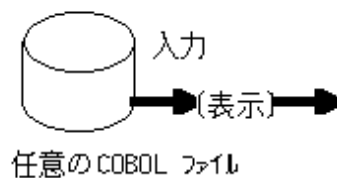
または、



7.8.3.3 レコードの表示

COBOLファイルのレコードの内容を表示することができます。『[表示]』

図7.4 操作手順



→ ファイルの内容をレコード単位に文字と16進数で表示します。

メニューバー	終了	表示ウィンドウを終了します。
	表示	次に表示するレコードをどのように指示するかを選択します。
	オプション	レコード操作画面の入力方法(16進入力/文字入力)の指定と、フォントの指定を行います。
	ヘルプ	表示ウィンドウのヘルプを表示します。

ボタン	キー	読み込み順序を変更する場合に選択します。読み込み順序を変更すると、画面に表示されるレコードの順番は、選択したレコードキーの順番に依存するようになります。
	検索	画面に表示するレコードを直接指示する場合に選択します。
	先頭	ファイル内で論理的に先頭に格納されているレコードを表示する場合に選択します。
	<	現在表示しているレコードより論理的に前に格納されているレコードを表示する場合に選択します。
	>	現在表示しているレコードより論理的に後ろに格納されているレコードを表示する場合に選択します。
	末尾	ファイル内で論理的に末尾に格納されているレコードを表示する場合に選択します。
レコード操作画面(上記画面の場合)	索引ファイル	表示中のファイルの属性を示しています。
	レコード長:28/28	表示中のレコードのレコード長/最大レコード長を示しています。
	16進入力	レコード操作画面へのデータ入力方法を示しています。
	[OFFSET]の下文字列および“+0 +1 … +E +F”	画面に表示されているレコードデータ内の位置のオフセットを16進数で示しています。上記画面でカーソルの位置(レコードデータ内のオフセット)は、カーソル行の“+8”とカーソルの左の“00000000”を加えた0x00000008となります。
	“+0 +1 … +E +F”の下2個の文字列	レコードデータを1バイトごとに16進数で表示しています。
	“0123456789ABCDEF”の下文字列	レコードデータを1バイト表記の文字で表示しています。



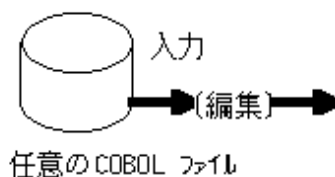
注意

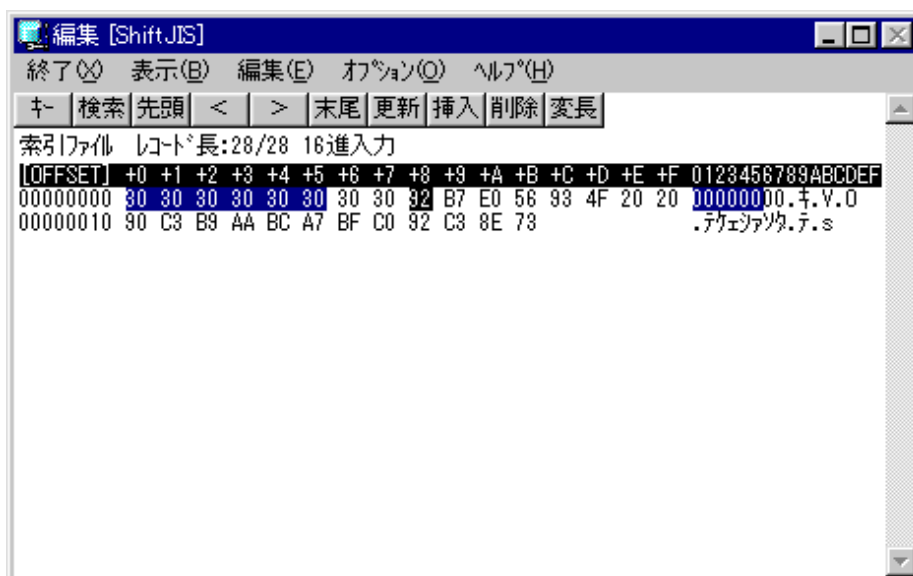
日本語文字などの表示できない文字は、ピリオドに置き換えられて表示されます。
詳細は、ヘルプを参照してください。

7.8.3.4 レコードの編集

COBOLファイルのレコードを更新、挿入および削除することができます。『[編集]』

図7.5 操作手順





→ ファイルの内容をレコード単位に編集します。

文字と16進数で表示されるレコードを編集し、挿入/削除/更新を行います。

メニューバー	終了	編集ウィンドウを終了します。
	表示	次に表示するレコードをどのように指示するかを選択します。
	編集	表示中のレコードを操作します。
	オプション	レコード操作画面の入力方法(16進入力/文字入力)の指定と、フォントの指定を行います。
	ヘルプ	編集ウィンドウのヘルプを表示します。
ボタン	キー	読込み順序を変更する場合に選択します。読込み順序を変更すると、画面に表示されるレコードの順番は、選択したレコードキーの順番に依存するようになります。
	検索	画面に表示するレコードを直接指示する場合に選択します。
	先頭	ファイル内で論理的に先頭に格納されているレコードを表示する場合に選択します。
	<	現在表示しているレコードより論理的に前に格納されているレコードを表示する場合に選択します。
	>	現在表示しているレコードより論理的に後ろに格納されているレコードを表示する場合に選択します。
	末尾	ファイル内で論理的に末尾に格納されているレコードを表示する場合に選択します。
	更新	画面に表示しているレコードを更新する場合に選択します。
	挿入	画面に表示しているレコードをファイルに追加する場合に選択します。
	削除	画面に表示しているレコードを削除する場合に選択します。
	変長	画面に表示しているレコードの長さを変更する場合に選択します。
レコード操作画面(上記画面の場合)	索引ファイル	表示中のファイルの属性を示しています。
	レコード長:28/28	表示中のレコードのレコード長/最大レコード長を示しています。
	16進入力	レコード操作画面へのデータ入力方法を示しています。
	[OFFSET]の下の文字列および“+0 +1 … +E +F”	画面に表示されているレコードデータ内の位置のオフセットを16進数で示しています。上記画面でカーソルの位置(レコードデータ内のオフセット)は、カーソル行の“+8”とカーソルの左の“00000000”を加えた0x00000008となります。

	“+0 +1 … +E +F”の下 の2文字列	レコードデータを1バイトごとに16進数で表示しています。
	“0123456789ABCDEF”の 下の文字列	レコードデータを1バイト表記の文字で表示しています。

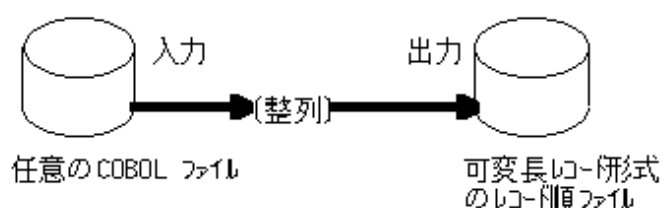
注意

索引ファイルの最大レコード長が16Kバイト以上の場合、[編集]コマンドでファイルを開けないことがあります。

7.8.3.5 レコードの整列

COBOLファイル中のレコードを整列することができます。『[整列]』

図7.6 操作手順



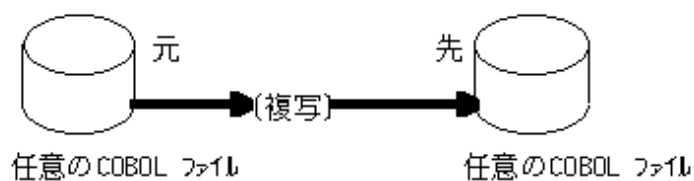
7.8.3.6 ファイルの操作

COBOLファイルを複写、削除および移動することができます。

ファイルの複写

COBOLファイルを複写します。『[複写]』

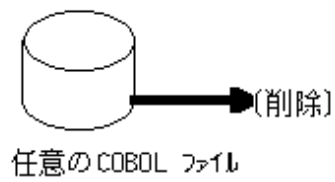
図7.7 操作手順



ファイルの削除

COBOLファイルを削除します。『[削除]』

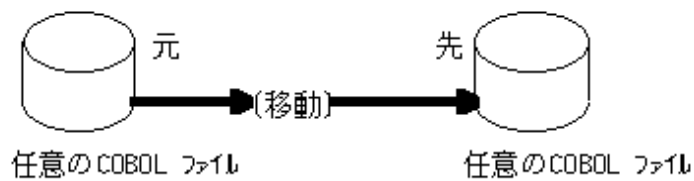
図7.8 操作手順



ファイルの移動

COBOLファイルを移動します。『移動』

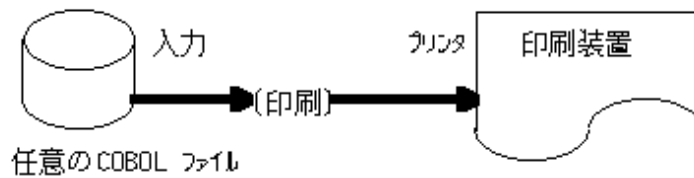
図7.9 操作手順



7.8.3.7 ファイルの印刷

COBOLファイルの内容を印刷することができます。『印刷』

図7.10 操作手順



7.8.3.8 ファイルの構造の変換

COBOLファイルの構造を、別の種類のファイル構造に変換します。『ロード』、『アンロード』

図7.11 操作手順



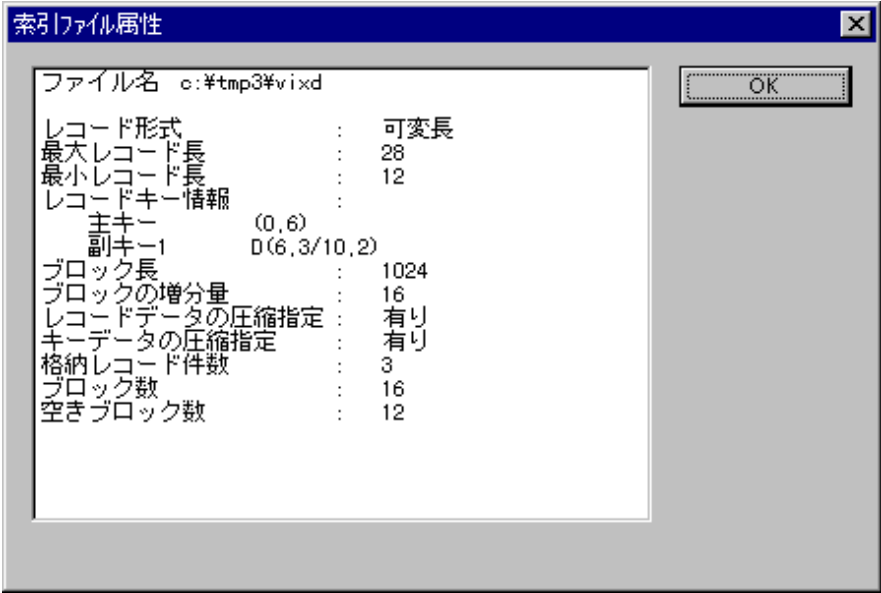
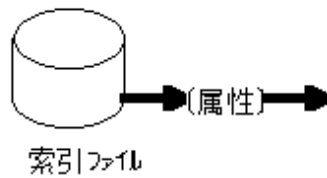
7.8.3.9 索引ファイルの操作

索引ファイルに対して、以下の操作を行うことができます。

属性情報の表示

索引ファイルの属性情報を表示します。『〔属性〕』

図7.12 操作手順



ファイル名	属性を表示している索引ファイル名を示しています。
レコード形式	レコード形式を示しています。(固定長/可変長)
最大レコード長	固定長の場合、レコード長を示しています。可変長の場合、最大レコード長を示しています。
最小レコード長	固定長の場合、表示されません。可変長の場合、最小レコード長を示しています。
レコードキー情報	索引ファイルのキー情報を示しています。(注)
ブロック長	索引ファイルの1ブロックの大きさを示しています。
ブロックの増分量	索引ファイルの増分単位をブロックの数で示しています。
レコードデータの圧縮指定	格納されているレコードデータが圧縮されているかどうかを示しています。
キーデータの圧縮指定	格納されているキーデータが圧縮されているかどうかを示しています。
格納レコード件数	索引ファイル中に格納されているレコードの件数を示しています。
ブロック数	索引ファイルを構成するブロックの数を示しています。
空きブロック数	索引ファイルを構成するブロックのうち、未使用なブロックの数を示しています。

注：キー情報の形式を以下に示します。

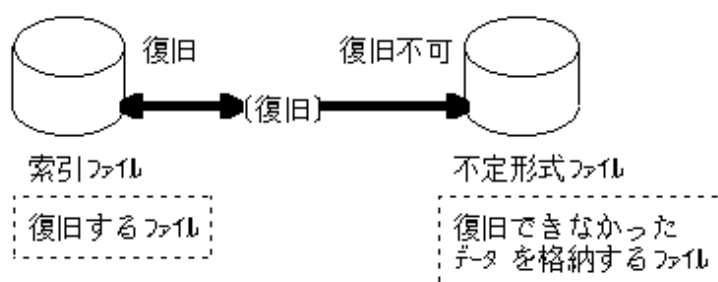
[D] (オフセット, 長さ[, N] [/オフセット, 長さ[, N]]...)

D : キーとするデータ項目が重複を許す指定の場合に表示します。WITH DUPLICATES指定に相当します。
オフセット : キーとするデータ項目のレコードの先頭からの相対位置 (0バイトから始まるバイト数) を表示します。
長さ : キーとするデータ項目の長さをバイト数で表示します。
N : UCS-2リトルエンディアン形式のデータがキーである場合に表示します。
/ : 非連続な複数のデータ項目により1つのキーが構成されている場合「/」で区切って表示します。

復旧

アクセスできなくなった索引ファイルを復旧することができます。『[復旧]』

図7.13 操作手順



再編成

索引ファイルの未使用空間を削除し、ファイルサイズを小さくします。『[再編成]』

図7.14 操作手順



7.8.4 COBOLファイルユーティリティコマンドモード

ここでは、コマンドモードでCOBOLファイルユーティリティを使用する方法について説明します。各コマンドは、MS-DOSコマンド(DOSコマンド)の画面上で直接使用します。



注意

各コマンドの入力・出力ファイル名に、Unicode固有の文字を指定することができます。ただし、各コマンドのメッセージは、シフトJIS範囲内の文字しか出力されません。Unicode固有の文字は、“?”に置き換えられて出力されます。

7.8.4.1 ファイル変換コマンド(cobfconv)

テキストファイルから可変長形式のレコード順ファイルの作成、または、可変長形式のレコード順ファイルからテキストファイルの作成を行います。

テキストファイルでは、バイナリデータは16進表記の文字列として表現されます。

指定方法

cobfconv [-k文字コード] -o出力ファイル名 -c変換方法, データ形式 入力ファイル名

インタフェース

文字コード

ファイルのコード体系を以下の形式で指定します。

$$-k \left\{ \begin{array}{l} \text{sjis} \\ \text{ebcdica} \\ \text{ebcdick} \\ \text{unicode[, } \left\{ \begin{array}{l} \text{LE} \\ \text{BE} \end{array} \right\}] \end{array} \right\}$$

sjis : ASCII/シフトJIS

ebcdica : JEF(EBCDIC/ASCII)

ebcdick : JEF(EBCDIC/KANA)

unicode : Unicode

LE : Unicode リトルエンディアン形式 [省略値]

BE : Unicode ビッグエンディアン形式

- -kオプションを省略した場合または-ksjisを指定した場合、テキストファイルおよびレコード順ファイルの文字コードはASCII/シフトJISコード体系として扱います。
- -kebcdicaまたは-kebcdickを指定した場合、テキストファイルの文字コードはASCII/シフトJISコード体系として扱います。レコード順ファイルの文字コードはJEF(EBCDIC/KANA)コード体系またはJEF(EBCDIC/ASCII)コード体系として扱います。
- -kunicodeを指定した場合、テキストファイルの文字コードはUCS-2リトルエンディアンコード体系として扱います。レコード順ファイルの内容はレコードデータ項目の定義に従って、UCS-2コード体系、UTF-8コード体系または16進表記のバイナリ値として扱います。
カンマ(,)に続き、レコード順ファイルのUCS-2コード体系のエンディアンを指定します。
省略した場合には、LE(リトルエンディアン形式)が指定されたものとして扱います。

出力ファイル名

作成するテキストファイルまたはレコード順ファイルのパス名を指定します。既に存在しているファイルを指定した場合、エラーとなります。

変換方法、データ形式

変換方法およびレコード順ファイルのレコード構成(データ形式)を以下の形式で指定します。

-kオプションを省略した場合または-ksjis/-kebcdica/-kebcdickを指定した場合

$$-c \left\{ \begin{array}{l} \text{b} \\ \text{t} \end{array} \right\} , \left\{ \begin{array}{l} \text{allchar} \\ \text{alltxtbin} \\ " \left\{ \begin{array}{l} \text{c長さ} \\ \text{t長さ} \\ \text{k長さ} \end{array} \right\} [;...] " \end{array} \right\}$$

-kunicodeを指定した場合

$$-c \left\{ \begin{array}{c} b \\ t \end{array} \right\}, \left\{ \begin{array}{c} allucs2 \\ allutf8 \\ " \left\{ \begin{array}{c} u長さ \\ f長さ \\ k長さ \end{array} \right\} [...] \end{array} \right\}$$

変換方法

変換方法を以下の文字で指定します。

- b : テキストファイルからレコード順ファイルに変換します。
- t : レコード順ファイルからテキストファイルに変換します。

データ形式

レコード順ファイルの1レコードを構成するデータ項目を、データ形式で指定します。データ形式の定義を以下に示します。

- ASCII/シフトJISコード体系の場合、「用途が表示用のデータ項目」(注)を文字形式、そのほかのデータ項目を16進形式とします。
- JEFコード体系の場合、日本語項目および日本語編集項目を日本語形式、日本語項目および日本語編集項目を除いた「用途が表示用のデータ項目」を文字形式、そのほかのデータ項目を16進形式とします。
- Unicodeコード体系の場合、日本語項目および日本語編集項目をUCS-2形式、日本語項目および日本語編集項目を除いた「用途が表示用のデータ項目」をUTF-8形式、そのほかのデータ項目を16進形式とします。

注意

「用途が表示用のデータ項目」については“COBOL文法書”を参照してください。

データ形式を以下の文字で指定します。

allchar

レコード中の全データを文字形式とみなします。

alltxtbin

レコード中の全データを16進形式とみなします。

"{c長さ|t長さ|k長さ};[...]"

レコード中にデータ形式が混在する場合、キーワード文字に続けて、そのデータ形式の長さ(日本語形式の場合は文字数)を指定します。それぞれのキーワード文字の意味を以下に示します。

- c : 文字形式
- t : 16進形式
- k : 日本語形式

ただし、日本語形式は、JEFコード体系の場合しか指定できません。

allucs2

レコード中の全データをUCS-2形式とみなします。

allutf8

レコード中の全データをUTF-8形式とみなします。

"{u長さ|f長さ|t長さ};[...]"

レコード中にデータ形式が混在する場合、キーワード文字に続けて、そのデータ形式の長さ(UCS-2形式の場合は文字数)を指定します。それぞれのキーワード文字の意味を以下に示します。

u : UCS-2形式
f : UTF-8形式
t : 16進形式



例

[データ形式が混在する場合の指定例]

レコード順ファイルのレコード構成

```
FD ファイル
01 データレコード.
02 データ1 PIC X(8).
02 データ2 PIC N(4).
02 データ3 PIC S9(8) BINARY.
```

コード系の違いによるデータ形式の表現は以下のようになります。

ASCII/シフトJISの場合

文字形式	8バイト
文字形式	8バイト
16進形式	4バイト

指定形式: "c16;t4"

JEFの場合

文字形式	8バイト
日本語形式	4文字
16進形式	4バイト

指定形式: "c8;k4;t4"

Unicodeの場合

UTF-8	8バイト
UCS-2	4文字
16進形式	4バイト

指定形式: "f8;u4;t4"



注意

データ形式が混在する場合、指定できるデータ形式の最大個数は256です。

入力ファイル名

変換元のテキストファイルまたはレコード順ファイルのパス名を指定します。



例

指定例

- レコード順ファイルからテキストファイルを作成する

文字コード : ASCII/シフトJIS
入力ファイル名 : infile
出力ファイル名 : outfile.txt
変換方法 : レコード順ファイル→テキストファイル
データ形式 : すべて文字形式

```
cobfconv -outfile.txt -ct,allchar infile
```

- テキストファイルからレコード順ファイルを作成する(JEFコード体系)

文字コード : JEF(EBCDIC/ASCII)
入力ファイル名 : infile.txt

出力ファイル名 : outfile
変換方法 : テキストファイル→レコード順ファイル
データ形式 : 混在(文字形式3バイト、16進形式2バイト、文字形式3バイト)

```
cobfconv -kebedica -outfile -cb,"c3;t2;c3" infile.txt
```

- ・テキストファイルからレコード順ファイルを作成する(Unicodeコード体系)

文字コード : Unicode
入力ファイル名 : infile.txt
出力ファイル名 : outfile
変換方法 : テキストファイル→レコード順ファイル
データ形式 : すべてUCS-2形式

```
cobfconv -kunicode -outfile -cb,"allucs2" infile.txt
```

7.8.4.2 ファイルロードコマンド(cobfload)

可変長形式のレコード順ファイルからレコード順ファイル/相対ファイル/索引ファイルを作成します。また、可変長形式のレコード順ファイルのレコードデータを、すでに存在するレコード順ファイル/相対ファイル/索引ファイルに追加することもできます。これをファイルの拡張といいます。ファイルの拡張でエラーが発生した場合、出力ファイルはコマンドの実行前の状態に戻されます。

指定方法

```
cobfload [-k文字コード] -o出力ファイル名 [-e] -dファイル属性 入力ファイル名
```

インタフェース

文字コード

Unicode形式の索引ファイルを作成する場合、以下の形式で指定します。

```
-kunicode[, {LE | BE}]
```

LE : Unicode リトルエンディアン形式 [省略値]

BE : Unicode ビッグエンディアン形式

出力ファイル名

作成または拡張するファイルのパス名を指定します。作成時に、既に存在しているファイルを指定した場合、エラーとなります。また、拡張時に、指定したファイルが存在していない場合、エラーとなります。

拡張指定

ファイルの拡張を行う場合、-eオプションを指定します。

ファイル属性

作成または拡張するファイルの属性を以下の形式で指定します。

$$-d \left\{ \begin{array}{c} S \\ R \\ I \end{array} \right\}, \left\{ \begin{array}{c} f \\ v \end{array} \right\}, \text{レコード長, " (オフセット, 長さ[N] / オフセット, 長さ[N] \cdots [D] ; \cdots)"$$

ファイル編成

ファイル編成を以下の文字で指定します。

S : レコード順ファイル
R : 相対ファイル
I : 索引ファイル

索引ファイルの拡張を行う場合、レコード形式、レコード長およびレコードキー情報を指定することはできません。

レコード形式

レコード形式を以下の文字で指定します。

f : 固定長
v : 可変長

レコード長

レコード長を指定します。可変長の場合は、最大レコード長で指定します。

レコードキー情報

索引ファイルを作成する場合、以下のレコードキー情報を指定します。

オフセット

レコードキーとするデータ項目のレコード内位置を、レコードの先頭を0とした相対バイト数で指定します。

長さ

レコードキーとするデータ項目の長さをバイト数で指定します。

N

レコードキーとするデータ項目がUCS-2(リトルエンディアン)形式である場合に指定します。
文字コードとして-kunicode, BEを指定した場合は意味を持ちません。

/

1つのレコードキーとして、連続しない複数のデータ項目を指定する場合、それぞれのデータ項目のオフセットと長さを“/”で区切って指定します。

D

レコードキーの重複を許す場合に指定します。

;

副レコードキーを定義する場合、それぞれの副レコードキー情報を“;”で区切って指定します。

入力ファイル名

変換元のレコード順ファイルのパス名を指定します。



例

指定例

- レコード順ファイルから索引ファイルを作成する

入力ファイル名	:	infile
出力ファイル名	:	ixdfile
ファイル属性	:	ファイル編成 : 索引ファイル
	:	レコード形式 : 可変長
	:	レコード長 : 80
	:	レコードキー情報 : データ項目 1 (レコード内位置 : 0 / 長さ : 5)
	:	データ項目 2 (レコード内位置 : 10 / 長さ : 5)
	:	重複指定 : あり
	:	副レコードキー (レコード内位置 : 5 / 長さ : 5)

```
cobfload -oixdfile -dI, v, 80, "(0, 5/10, 5), D: (5, 5)" infile
```

- ・レコード順ファイルの内容を相対ファイルに追加する(拡張)

```

入力ファイル名  : infile
出力ファイル名  : relfile
ファイル属性    : ファイル編成 : 相対ファイル
                  レコード形式 : 固定長
                  レコード長   : 80

```

```
cobfload -orelfile -e -dR,f,80 infile
```



注意

入力ファイルに出力ファイルの最大レコード長より大きいレコードが存在した場合、コマンド実行時にエラーとなります。

7.8.4.3 ファイルアンロードコマンド(cobfulod)

レコード順ファイル/相対ファイル/索引ファイルから可変長形式のレコード順ファイルを作成します。

指定方法

```
cobfulod -o出力ファイル名 -iファイル属性 入力ファイル名
```

インタフェース

出力ファイル名

作成するレコード順ファイルのパス名を指定します。既に存在しているファイルを指定した場合、エラーとなります。

ファイル属性

変換元のファイルの属性を以下の形式で指定します。

$$-i \quad \left\{ \begin{array}{c} S \\ R \\ I \end{array} \right\} , \quad \left\{ \begin{array}{c} f \\ v \end{array} \right\} , \text{レコード長}$$

ファイル編成

ファイル編成を以下の文字で指定します。

S : レコード順ファイル
R : 相対ファイル
I : 索引ファイル

レコード形式

レコード形式を以下の文字で指定します。ただし、ファイル編成が索引ファイルの場合は指定できません。

f : 固定長
v : 可変長

レコード長

レコード長を指定します。可変長の場合は、最大レコード長で指定します。ただし、ファイル編成が索引ファイルの場合は指定できません。

入力ファイル名

変換元のレコード順ファイル/相対ファイル/索引ファイルのパス名を指定します。



例

指定例

- ・ 相対ファイルからレコード順ファイルを作成する

入力ファイル名 : relfile
出力ファイル名 : outfile
ファイル属性 : ファイル編成 : 相対ファイル
レコード形式 : 固定長
レコード長 : 80

```
cobfulod -ooutfile -iR,f,80 relfile
```

7.8.4.4 ファイル表示コマンド(cobfbrws)

ファイルの内容をレコード単位に文字形式と16進形式で表示します。なお、英数字(0x20～0x7e)以外のデータは、ピリオドに置き換えて表示します。また、表示範囲をレコード単位で指定することができます。表示範囲を指定しなかった場合は、ファイル中の全レコードを表示します。

指定方法

```
cobfbrws [-k文字コード] -iファイル属性 [-ps開始位置] [-pe終了位置] [-po表示順序] [-pk検索キー番号] 入力ファイル名
```

インタフェース

文字コード

表示するファイルのコード体系を以下の形式で指定します。

-k { sjis
ebcdica
ebcdick
unicode[, { LE
BE }] }

sjis : ASCII/シフトJIS

ebcdica : JEF(EBCDIC/ASCII)

ebcdick : JEF(EBCDIC/KANA)

unicode : Unicode

LE : Unicode リトルエンディアン形式 [省略値]

BE : Unicode ビッグエンディアン形式

- -kオプションを省略した場合または-ksjisを指定した場合、ファイルの文字コードをASCII/シフトJISコード体系として扱います。
- -kebcdicaまたは-kebcdickを指定した場合、ファイルの文字コードをJEF(EBCDIC/KANA)コード体系またはJEF(EBCDIC/ASCII)コード体系として扱います。
- -kunicodeを指定した場合、ファイルの文字コードをUnicodeのコード体系として扱います。
カンマ(,)に続き、UCS-2コード体系のエンディアンを指定します。省略した場合には、LE(リトルエンディアン形式)が指定されたものとして扱います。

ファイル属性

表示するファイルの属性を以下の形式で指定します。

$$-i \left\{ \begin{array}{c} S \\ L \\ R \\ I \end{array} \right\}, \left\{ \begin{array}{c} f \\ v \end{array} \right\}, \text{レコード長}$$

ファイル編成

ファイル編成を以下の文字で指定します。

S : レコード順ファイル
L : 行順ファイル
R : 相対ファイル
I : 索引ファイル

レコード形式

レコード形式を以下の文字で指定します。ただし、ファイル編成が索引ファイルの場合は指定できません。

f : 固定長
v : 可変長

レコード長

レコード長を指定します。可変長の場合は、最大レコード長で指定します。ただし、ファイル編成が索引ファイルの場合は指定できません。

開始位置

表示を開始するレコード位置を以下の形式で指定します。

$$-ps \left\{ \begin{array}{l} \text{格納順番} \\ r \text{ 相対レコード番号} \\ ic \text{ レコードキー値} \\ it \text{ レコードキー値} \end{array} \right\}$$

ファイル編成によって指定する内容が異なります。

- レコード順ファイルおよび行順ファイルでは、レコードの格納順番を指定します。
- 相対ファイルでは、キーワード文字“r”に続けて相対レコード番号を指定します。
- 索引ファイルでは、レコードキー値を文字形式で指定するか16進形式で指定するかによって指定方法が異なります。文字形式の場合は、キーワード文字列“ic”に続けてレコードキー値を指定します。16進形式の場合は、キーワード文字列“it”に続けてレコードキー値を指定します。

開始位置を省略した場合、ファイルの先頭レコードから表示します。

終了位置

表示を終了するレコード位置を以下の形式で指定します。

-pe { 格納順番
r 相対レコード番号
ic レコードキー値
it レコードキー値
in レコードキー値
t 出力件数 }

出力件数の指定以外は、開始位置と同じです。終了位置を出力件数で指定する場合は、キーワード文字“t”に続けて出力レコード件数を指定します。

終了位置を省略した場合、ファイルの最後のレコードまで表示します。

表示順序

開始位置と終了位置の範囲に複数のレコードが存在する場合、それらのレコードの表示順序を以下の形式で指定します。

-po { A
D }

A

レコードを昇順に表示します。レコード順ファイルおよび行順ファイルの場合は、先に格納されたレコードから表示します。相対ファイルの場合は、相対レコード番号の小さいレコードから表示します。索引ファイルの場合は、レコードキー値が小さいレコードから表示します。

D

レコードを降順に表示します。相対ファイルの場合は、相対レコード番号の大きいレコードから表示します。索引ファイルの場合は、レコードキー値が大きいレコードから表示します。なお、レコード順ファイルおよび行順ファイルでは降順は指定できません。

表示順序を省略した場合、昇順を指定したとみなします。

検索キー番号

索引ファイルを表示する場合、検索キーになるレコードキーの番号を指定します。レコードキーの番号は、主レコードキーを0、最初の副レコードキーを1として、それ以降の副レコードキーを2以上の追番で表現した数字です。

検索キー番号を省略した場合、主レコードキーが指定されたとみなします。

入力ファイル名

表示するファイルのパス名を指定します。



例

指定例

- レコード順ファイルの内容を表示する

入力ファイル名 : seqfile
ファイル属性 : ファイル編成 : レコード順ファイル
レコード形式 : 可変長
レコード長 : 80
開始位置（格納順番） : 5
終了位置（出力件数） : 10

```
cobfbrws -iS,v,80 -ps5 -pet10 seqfile
```

- 相対ファイルの内容を表示する

```
入力ファイル名      : relfile
文字コード          : JEF (EBCDIC/ASCII)
ファイル属性        : ファイル編成  : 相対ファイル
                      レコード形式  : 固定長
                      レコード長   : 50
開始位置 (相対レコード番号) : 20
終了位置 (相対レコード番号) : 10
表示順序            : 降順
```

```
cobfbrws -kebcdica -iR,f,50 -psr20 -per10 -poD relfile
```

- 索引ファイルの内容を表示する

```
入力ファイル名      : idxfile
文字コード          : JEF (EBCDIC/KANA)
ファイル属性        : ファイル編成  : 索引ファイル
開始位置 (レコードキー値) : 0001 (16進)
終了位置 (レコードキー値) : 0010 (16進)
表示順序            : 昇順
検索キー番号        : 副レコードキー
```

```
cobfbrws -kebcdick -iI -psit0001 -peit0010 -poA -pkI idxfile
```



注意

- 文字コードがJEF(EBCDIC/KANA)の場合、英数字および1バイトカナ文字以外のデータは、ピリオドに置き換えて表示します。
- 文字コードがJEF(EBCDIC/ASCII)の場合、英数字および1バイト英小文字以外のデータは、ピリオドに置き換えて表示します。
- 文字コードがUnicodeの場合、UTF-8コード体系の1バイト表記可能な英数字以外のデータは、ピリオドに置き換えて表示します。
- -psicおよび-peicに指定するレコードキー値には、ASCII/シフトJISコード体系で表記可能な値だけを設定できます。

7.8.4.5 ファイル整列コマンド(cobfsort)

レコード中の任意のデータ項目をキーとして、ファイル中のレコードを昇順または降順に整列し、整列された結果を可変長形式のレコード順ファイルに出力します。

指定方法

```
cobfsort [-k文字コード] -o出力ファイル名 -s整列条件 -iファイル属性 入力ファイル名
```

インタフェース

文字コード

文字コードのコード体系を以下の形式で指定します。

$$-k \left\{ \begin{array}{l} \text{sjis} \\ \text{ebcdica} \\ \text{ebcdick} \\ \text{unicode}[\left\{ \begin{array}{l} \text{LE} \\ \text{BE} \end{array} \right\}] \end{array} \right\}$$

sjis : ASCII/シフトJIS
 ebcdica : JEF(EBCDIC/ASCII)
 ebcdick : JEF(EBCDIC/KANA)
 unicode : Unicode
 LE : Unicode リトルエンディアン形式 [省略値]
 BE : Unicode ビッグエンディアン形式

- -kオプションを省略した場合または-ksjisを指定した場合、整列するファイルの文字コードをASCII/シフトJISコード体系として扱います。
- -kebcdicaまたは-kebcdickを指定した場合、整列するファイルの文字コードをJEF(EBCDIC/KANA)コード体系またはJEF(EBCDIC/ASCII)コード体系として扱います。
- -kunicodeを指定した場合、整列するファイルの文字コードをUnicodeコード体系として扱います。
カンマ(,)に続き、UCS-2コード体系のエンディアンを指定します。省略した場合には、LE(リトルエンディアン形式)が指定されたものとして扱います。

出力ファイル名

整列結果を出力するレコード順ファイルのパス名を指定します。既に存在しているファイルを指定した場合、エラーとなります。

整列条件

キーとするデータ項目の属性と整列順序を以下の形式で指定します。

$$-S"(\left\{ \begin{array}{l} 1[B] \\ X \\ N \\ [S]9 \\ S9L \\ S9T \\ S9LS \\ S9TS \\ [S]9PD \\ [S]9B \\ [S]9C5 \\ C1 \\ C2 \end{array} \right\} , \text{オフセット, 大きさ}) [\left\{ \begin{array}{l} A \\ D \end{array} \right\}][; \cdots]"$$

注意

指定できる整列条件の最大個数は64です。

キーの項目属性

キーの項目属性を以下の値で指定します。

1[B]	PIC 1() [BIT]
X	PIC X()
N	PIC N()
[S]9	PIC [S]9()
S9L	PIC S9() LEADING
S9T	PIC S9() TRAILING
S9LS	PIC S9() LEADING SEPARATE
S9TS	PIC S9() TRAILING SEPARATE
[S]9PD	PIC [S]9() PACKED-DECIMAL
[S]9B	PIC [S]9() BINARY
[S]9C5	PIC [S]9() COMP-5
C1	COMP-1
C2	COMP-2

オフセット

キー項目のレコード内オフセットを、レコードの先頭を0とした相対バイト位置で指定します。

大きさ

キー項目の大きさを、PICTURE句での数字項目の桁数または英数字項目/日本語項目の文字数で指定します。

注意

- キーの項目属性に“1B”を指定した場合、ここには、整列の対象になるビットのマスク値を10進数で指定します。キーの大きさは無条件に1バイトになります。
- キーの項目属性に“C1”または“C2”を指定した場合、大きさを省略することができます。ただし、大きさを指定する場合は、項目属性“C1”に対しては4、項目属性“C2”に対しては8を指定します。

整列順序

整列順序には、レコードをキーの項目属性に従って昇順に整列するか、降順に整列するかを指定します。

A : 昇順

D : 降順

整列順序を省略した場合は昇順となります。

ファイル属性

整列するファイルの属性を以下の形式で指定します。

$$-i \left\{ \begin{array}{c} S \\ L \\ R \\ I \end{array} \right\}, \left\{ \begin{array}{c} f \\ v \end{array} \right\}, \text{レコード長}$$

ファイル編成

ファイル編成を以下の文字で指定します。

S : レコード順ファイル
L : 行順ファイル
R : 相対ファイル
I : 索引ファイル

レコード形式

レコード形式を以下の文字で指定します。ただし、ファイル編成が索引ファイルの場合は指定できません。

f : 固定長
v : 可変長

レコード長

レコード長を指定します。可変長の場合は、最大レコード長で指定します。ただし、ファイル編成が索引ファイルの場合は指定できません。

入力ファイル名

整列するファイルのパス名を指定します。



例

指定例

- 相対ファイルを整列してレコード順ファイルに出力する

入力ファイル名 : relfile
出力ファイル名 : outfile
整列条件 : [第1キー] 項目属性 : 日本語(N)
レコード内オフセット : 0
大きさ : 2
整列順序 : 昇順
[第2キー] 項目属性 : 英数字
レコード内オフセット : 10
大きさ : 5
整列順序 : 降順
ファイル属性 : ファイル編成 : 相対ファイル
レコード形式 : 可変長
レコード長 : 80

```
cobfsort -outfile -s"(N,0,2),A:(X,10,5),D"-iR,v,80 relfile
```

- レコード順ファイルのレコード構成

```
入力ファイル名 : infile
出力ファイル名 : outfile
整列条件       :   [第 1 キー] 項目属性                   : ブール(1B)
                                     レコード内オフセット : 4
                                     マスク値(00001111)    : 15
                                     整列順序              : 昇順
ファイル属性   : ファイル編成 : レコード順ファイル
                  レコード形式 : 固定長
                  レコード長   : 5
```

- 262 -

```
cobfattr ixdfile
```

7.8.4.7 ファイル復旧コマンド(cobfrcov)

プロセスの異常終了などで、クローズ処理が正常に行われなかった索引ファイルを、再度正常にアクセスできるように復旧します。ただし、データに異常があるために復旧できないレコードについては、それらのデータを未復旧データファイルとして可変長形式のレコード順ファイルに出力します。

指定方法

```
cobfrcov 復旧ファイル名 未復旧データファイル名
```

インタフェース

復旧ファイル名

復旧処理を行う索引ファイルのパス名を指定します。

未復旧ファイル名

復旧できないレコードのデータを出力するファイルのパス名を指定します。復旧できないデータがなければ、未復旧データファイルは作成されません。



例

指定例

- 索引ファイルを復旧する

復旧ファイル名 : ixdfile
未復旧ファイル名 : seqfile

```
cobfrcov ixdfile seqfile
```



注意

同じ索引ファイルに対し、同時にファイル復旧コマンドを実行した場合の動作は保証されません。直前のコマンド実行の完了を確認後、実行してください。

7.8.4.8 ファイル再編成コマンド(cobfreog)

索引ファイルの空きブロックを削除し、再編成した内容を別の索引ファイルに出力します。再編成した索引ファイルのファイルサイズは、再編成前のファイルサイズよりも小さくなります。

指定方法

```
cobfreog -o出力ファイル名 入力ファイル名
```

インタフェース

出力ファイル名

再編成後の索引ファイルのパス名を指定します。既に存在しているファイルを指定した場合、エラーとなります。

入力ファイル名

再編成を行う索引ファイルのパス名を指定します。



例

指定例

- ・ 索引ファイルを再編成してoutfileに出力する

入力ファイル名 : ixdfile
出力ファイル名 : outfile

```
cobfreog -outfile ixdfile
```



注意

空きブロックの削除により、ファイルアクセス性能が低下する場合があります。

7.8.5 COBOLファイルユーティリティ関数

COBOLファイルユーティリティ関数には次の機能があります。

機能	関数名
変換	7.8.5.1 ファイル変換関数
ロード	7.8.5.2 ファイルロード関数
アンロード	7.8.5.3 ファイルアンロード関数
整列	7.8.5.4 ファイル整列関数
再編成	7.8.5.5 ファイル再編成関数
コピー	7.8.5.6 ファイルコピー関数
移動	7.8.5.7 ファイル移動関数
削除	7.8.5.8 ファイル削除関数

登録集

COBOLファイルユーティリティ関数を呼び出す場合には、作業場所節で以下の登録集をCOPY文で取り込んでください。
なお、この登録集は、NetCOBOLをインストールしたフォルダに格納されています。

登録集名

COBF-INF.CBL

```
WORKING-STORAGE SECTION.  
COPY COBF-INF.
```



注意

- COBOLファイルユーティリティ関数の登録集を取り込むCOPY文のREPLACING指定、またはこのCOPY文が置き換えの対象となるようなREPLACE文の記述はしないでください。
- 登録集を変更した場合、動作は保証されません。
- 各COBOLファイルユーティリティ関数を呼び出す前に、必ずインタフェース領域のCOBF-INFをLOW-VALUEで初期化してください。

リンク時

翻訳オプションDLOADを指定していないプログラムから利用する場合、F3BIFUTC.LIBを結合してください。

なお、このライブラリは、NetCOBOLをインストールしたフォルダに格納されています。

実行時

翻訳オプションDLOADを指定したプログラムから利用する場合、以下のエントリ情報が必要になります。エントリ情報の指定方法については、“[5.4.2 副プログラムのエントリ情報](#)”を参照してください。

```
[ENTRY]
COB_FILE_CONVERT=F3BIFUTC.DLL
COB_FILE_LOAD=F3BIFUTC.DLL
COB_FILE_UNLOAD=F3BIFUTC.DLL
COB_FILE_SORT=F3BIFUTC.DLL
COB_FILE_REORGANIZE=F3BIFUTC.DLL
COB_FILE_COPY=F3BIFUTC.DLL
COB_FILE_MOVE=F3BIFUTC.DLL
COB_FILE_DELETE=F3BIFUTC.DLL
```

7.8.5.1 ファイル変換関数

機能

テキストファイルから可変長形式のレコード順ファイルの作成、または、可変長形式のレコード順ファイルからテキストファイルの作成を行います。

テキストファイルでは、バイナリデータは16進表記の文字列として表現されます。

指定方法

呼出しの記述

```
CALL "COB_FILE_CONVERT" USING BY REFERENCE COBF-INF
```

呼出し時のデータ設定

本関数では、レコード順ファイルの1レコードを構成するデータ項目を、データ形式で指定します。データ形式の定義を以下に示します。

- ASCII/シフトJISコード系で動作する場合、「用途が表示用のデータ項目」(注)を文字形式、その他のデータ項目を16進形式といいます。
- Unicodeコード系で動作する場合、日本語項目および日本語編集項目をUCS-2形式、日本語項目および日本語編集項目を除いた「用途が表示用のデータ項目」をUTF-8形式、その他のデータ項目を16進形式といいます。



参照

「用途が表示用のデータ項目」については“COBOL文法書”を参照してください。

以下に、関数呼出し時のデータ設定について説明します。

COBF-INPUT-FILENAME

変換元のテキストファイルまたはレコード順ファイルのパス名を指定します。

COBF-OUTPUT-FILENAME

作成するテキストファイルまたはレコード順ファイルのパス名を指定します。既に存在しているファイルを指定した場合、エラーとなります。

COBF-CONVERT-COND

変換方法を、以下の値で指定します。

"B": テキストファイルからレコード順ファイルに変換する
"T": レコード順ファイルからテキストファイルに変換する

COBF-CONVERT-NUM

レコード順ファイルの1レコードを構成するデータ形式の個数を指定します。



指定できる値の最大は256です。

COBF-CONVERT-TYPE

レコード順ファイルの1レコードを構成するデータ項目を、データ形式で指定します。データ形式の指定には、以下の条件名を使用します。

条件名	値	意味	コード体系		
			ASCII/シフトJIS	JEF	Unicode
COBF-CONVERT-TYPE-ALLCHAR	"1"	レコード中の全データを文字形式とみなします。	○	○	—
COBF-CONVERT-TYPE-ALLTXTBIN	"2"	レコード中の全データを16進形式とみなします。	○	○	—
COBF-CONVERT-TYPE-CHAR	"3"	指定長だけ文字形式とみなします。	○	○	—
COBF-CONVERT-TYPE-TXTBIN	"4"	指定長だけ16進形式とみなします。	○	○	○
COBF-CONVERT-TYPE-KANJI	"5"	指定長だけ2バイト表記の漢字形式とみなします。	—	○	—
COBF-CONVERT-TYPE-ALLUCS2	"6"	レコード中の全データをUCS-2形式とみなします。	—	—	○
COBF-CONVERT-TYPE-ALLUTF8	"7"	レコード中の全データをUTF-8形式とみなします。	—	—	○
COBF-CONVERT-TYPE-UCS2	"8"	指定長だけUCS-2形式とみなします。	—	—	○
COBF-CONVERT-TYPE-UTF8	"9"	指定長だけUTF-8形式とみなします。	—	—	○

COBF-CONVERT-LEN

データ形式の長さ(UCS-2形式の場合は文字数)を指定します。COBF-CONVERT-TYPEが以下の場合、必ず指定してください。

- COBF-CONVERT-TYPE-CHAR
- COBF-CONVERT-TYPE-TXTBIN
- COBF-CONVERT-TYPE-UCS2
- COBF-CONVERT-TYPE-UTF8
- COBF-CONVERT-TYPE-KANJI

なお、COBF-CONVERT-TYPEが以下の場合、この指定は無視されます。

- COBF-CONVERT-TYPE-ALLCHAR
- COBF-CONVERT-TYPE-ALLTXTBIN
- COBF-CONVERT-TYPE-ALLUCS2
- COBF-CONVERT-TYPE-ALLUTF8



例

[データ形式が混在する場合の指定例]

レコード順ファイルのレコード構成

```

FD ファイル
01 データレコード.
  02 データ1 PIC X(8).
  02 データ2 PIC N(4).
  02 データ3 PIC S9(8) BINARY.

```

コード系の違いによるデータ形式の表現は以下のようになります。

ASCII/シフトJISの場合

```

文字形式  8バイト
文字形式  8バイト
16進形式  4バイト

```

データ形式の個数:2

⇒

呼出し時のデータ設定

```

MOVE 2 TO COBF-CONVERT-NUM
SET COBF-CONVERT-TYPE-CHAR(1) TO TRUE
MOVE 16 TO COBF-CONVERT-LEN(1)
SET COBF-CONVERT-TYPE-TXTBIN(2) TO TRUE
MOVE 4 TO COBF-CONVERT-LEN(2)

```

Unicodeの場合

```

UTF-8      8バイト
UCS-2      4文字
16進形式   4バイト

```

データ形式の個数:3

⇒

呼出し時のデータ設定

```

MOVE 3 TO COBF-CONVERT-NUM
SET COBF-CONVERT-TYPE-UTF8(1) TO TRUE
MOVE 8 TO COBF-CONVERT-LEN(1)
SET COBF-CONVERT-TYPE-UCS2(2) TO TRUE
MOVE 4 TO COBF-CONVERT-LEN(2)
SET COBF-CONVERT-TYPE-TXTBIN(3) TO TRUE
MOVE 4 TO COBF-CONVERT-LEN(3)

```

復帰コード

本関数からの復帰コードは、特殊レジスタPROGRAM-STATUSを使用して受け取ります。

復帰値	意味
0	ファイルの変換に成功しました。
-1	ファイルの変換に失敗しました。

復帰値-1が返却された場合、エラーの詳細は、COBF-MESSAGEに文字列で設定されます。



例

指定例

- レコード順ファイルからテキストファイルを作成する

```

入力ファイル名 : C:¥INFILE
出力ファイル名 : C:¥OUTFILE.TXT
変換方法       : レコード順ファイル→テキストファイル
データ形式     : すべて文字形式

```

```

MOVE LOW-VALUE TO COBF-INF.
MOVE "C:¥INFILE" TO COBF-INPUT-FILENAME.

```

```

MOVE "C:¥OUTFILE.TXT" TO COBF-OUTPUT-FILENAME.
MOVE "T" TO COBF-CONVERT-COND.
MOVE 1 TO COBF-CONVERT-NUM.
SET COBF-CONVERT-TYPE-ALLCHAR(1) TO TRUE.
CALL "COB_FILE_CONVERT" USING BY REFERENCE COBF-INF.

```

- ・ テキストファイルからレコード順ファイルを作成する

入力ファイル名 : C:¥INFILE.TXT
 出力ファイル名 : C:¥OUTFILE
 変換方法 : テキストファイル→レコード順ファイル
 データ形式 : 混在(文字形式3バイト、16進形式2バイト、文字形式3バイト)

```

MOVE LOW-VALUE TO COBF-INF.
MOVE "C:¥INFILE.TXT" TO COBF-INPUT-FILENAME.
MOVE "C:¥OUTFILE" TO COBF-OUTPUT-FILENAME.
MOVE "B" TO COBF-CONVERT-COND.
MOVE 3 TO COBF-CONVERT-NUM.
SET COBF-CONVERT-TYPE-CHAR(1) TO TRUE.
MOVE 3 TO COBF-CONVERT-LEN(1).
SET COBF-CONVERT-TYPE-TXTBIN(2) TO TRUE.
MOVE 2 TO COBF-CONVERT-LEN(2).
SET COBF-CONVERT-TYPE-CHAR(3) TO TRUE.
MOVE 3 TO COBF-CONVERT-LEN(3).
CALL "COB_FILE_CONVERT" USING BY REFERENCE COBF-INF.

```

7.8.5.2 ファイルロード関数

機能

可変長形式のレコード順ファイルからレコード順ファイル/相対ファイル/索引ファイルを作成します。また、可変長形式のレコード順ファイルのレコードデータを、すでに存在するレコード順ファイル/相対ファイル/索引ファイルに追加することもできます。これをファイルの拡張といいます。

ファイルの拡張でエラーが発生した場合、出力ファイルはファイルの拡張を行う前の状態に戻されます。

指定方法

呼出しの記述

```
CALL "COB_FILE_LOAD" USING BY REFERENCE COBF-INF.
```

呼出し時のデータ設定

COBF-INPUT-FILENAME

変換元のレコード順ファイルのパス名を指定します。

COBF-OUTPUT-FILENAME

作成または拡張するファイルのパス名を指定します。作成時に、既に存在しているファイルを指定した場合、エラーとなります。また、拡張時に、指定したファイルが存在していない場合、エラーとなります。

COBF-OUTPUT-ATTR

作成または拡張するファイルのファイル編成を、以下の文字で指定します。

"S" : レコード順ファイル

"R" : 相対ファイル

"I" : 索引ファイル

COBF-OUTPUT-RECFM

作成または拡張するファイルのレコード形式を、以下の文字で指定します。

"F" : 固定長

"V": 可変長

COBF-OUTPUT-RECLLEN

作成または拡張するファイルのレコード長を指定します。可変長の場合は、最大レコード長を指定します。

COBF-OUTPUT-KEYNUM

索引ファイルを作成または拡張する場合に、レコードキーとして扱うデータ項目の個数を指定します。

COBF-OUTPUT-USE-EXTKEY

索引ファイルを作成するまたは拡張する場合、レコードキーとして扱うデータ項目に属性を指定するかしないかを以下の文字で指定します。

属性の指定は、COBF-OUTPUT-EXT-KEY-TYPEで行います。

"Y": データ項目の属性を使用する

" ": データ項目の属性を使用しない

UCS-2形式のデータ項目をキーにする場合、“Y”を指定します。

COBF-LOAD-COND

ファイルを作成するか、ファイルを拡張するかを、以下の文字で指定します。

"C": ファイルを作成する

"E": ファイルを拡張する

以下の情報は、レコードキーの個数分、指定します。

COBF-OUTPUT-OFFSET

レコードキーとするデータ項目のレコード内位置を、レコードの先頭を0とした相対位置で指定します。

COBF-OUTPUT-KEYLEN

レコードキーとするデータ項目の長さをバイト数で指定します。

COBF-OUTPUT-KEYCONT

1つのレコードキーの構成を、以下の文字で指定します。

"C": キーの連続性を示す

"E": 1つのキー定義の終了を示す

COBF-OUTPUT-KEYDUP

レコードキーの重複を許す場合に“D”を指定します。なお、この項目はCOBF-OUTPUT-KEYCONTに“E”が指定された場合だけ有効です。

COBF-OUTPUT-EXT-KEY-TYPE

COBF-OUTPUT-USE-EXTKEYに“Y”を指定した場合、レコードキーとするデータ項目の属性を、以下の条件名を使用して指定します。

条件名	値	意味
COBF-EXT-KEY-TYPE-PICX	1	PIC N以外の項目
COBF-EXT-KEY-TYPE-PICN	2	PIC Nの項目

復帰コード

本関数からの復帰コードは、特殊レジスタPROGRAM-STATUSを使用して受け取ります。

復帰値	意味
0	ファイルのロードに成功しました。
-1	ファイルのロードに失敗しました。

復帰値-1が返却された場合、エラーの詳細は、COBF-MESSAGEに文字列で設定されます。

注意

- 出力ファイルの最大レコード長より大きいレコードが入力ファイルに存在した場合、実行時にエラーとなります。
- 索引ファイルを拡張する場合、最大キー値より小さいキー値のレコードを書き出すことができます。

例

指定例

- レコード順ファイルから索引ファイルを作成する

入力ファイル名 : C:¥INFILE
出力ファイル名 : C:¥IXDFILE
ファイル編成 : 索引ファイル
レコード形式 : 可変長
レコード長 : 80バイト
主レコードキー : 2つのデータ項目で構成する。また、レコードキーの重複を許す。
 [キー1] 0バイトオフセットから始まる、5バイトの項目
 [キー2] 10バイトオフセットから始まる、5バイトの項目
副レコードキー : 5バイトオフセットから始まる、5バイトの項目

```
MOVE LOW-VALUE TO COBF-INF.  
MOVE "C:¥INFILE" TO COBF-INPUT-FILENAME.  
MOVE "C:¥IXDFILE" TO COBF-OUTPUT-FILENAME.  
MOVE "I" TO COBF-OUTPUT-ATTR.  
MOVE "V" TO COBF-OUTPUT-RECFM.  
MOVE 80 TO COBF-OUTPUT-RECLEN.  
MOVE 3 TO COBF-OUTPUT-KEYNUM.  
MOVE 0 TO COBF-OUTPUT-OFFSET(1).  
MOVE 5 TO COBF-OUTPUT-KEYLEN(1).  
MOVE "C" TO COBF-OUTPUT-KEYCONT(1).  
MOVE 5 TO COBF-OUTPUT-OFFSET(2).  
MOVE 10 TO COBF-OUTPUT-KEYLEN(2).  
MOVE "E" TO COBF-OUTPUT-KEYCONT(2).  
MOVE "D" TO COBF-OUTPUT-KEYDUP(2).  
MOVE 5 TO COBF-OUTPUT-OFFSET(3).  
MOVE 5 TO COBF-OUTPUT-KEYLEN(3).  
MOVE "E" TO COBF-OUTPUT-KEYCONT(3).  
MOVE "C" TO COBF-LOAD-COND.  
CALL "COB_FILE_LOAD" USING BY REFERENCE COBF-INF.
```

- レコード順ファイルの内容を相対ファイルに追加する(拡張)

入力ファイル名 : C:¥INFILE
出力ファイル名 : C:¥RELFILE
ファイル編成 : 相対ファイル
レコード形式 : 固定長
レコード長 : 80バイト

```
MOVE LOW-VALUE TO COBF-INF.  
MOVE "C:¥INFILE" TO COBF-INPUT-FILENAME.  
MOVE "C:¥RELFILE" TO COBF-OUTPUT-FILENAME.  
MOVE "R" TO COBF-OUTPUT-ATTR.  
MOVE "F" TO COBF-OUTPUT-RECFM.  
MOVE 80 TO COBF-OUTPUT-RECLEN.  
MOVE "E" TO COBF-LOAD-COND  
CALL "COB_FILE_LOAD" USING BY REFERENCE COBF-INF.
```

7.8.5.3 ファイルアンロード関数

機能

レコード順ファイル/相対ファイル/索引ファイルから可変長形式のレコード順ファイルを作成します。

指定方法

呼出しの記述

```
CALL "COB_FILE_UNLOAD" USING BY REFERENCE COBF-INF.
```

呼出し時のデータ設定

COBF-INPUT-FILENAME

変換元のレコード順ファイル/相対ファイル/索引ファイルのパス名を指定します。

COBF-INPUT-ATTR

変換元のファイルのファイル編成を以下の文字で指定します。

"S" : レコード順ファイル

"R" : 相対ファイル

"I" : 索引ファイル

COBF-INPUT-RECFM

変換元のファイルのレコード形式を以下の文字で指定します。

"F" : 固定長

"V" : 可変長

COBF-INPUT-RECLLEN

変換元のファイルのレコード長を指定します。可変長の場合は、最大レコード長を指定します。ただし、入力ファイルが索引ファイルの場合は指定できません。

COBF-OUTPUT-FILENAME

作成するレコード順ファイルのパス名を指定します。既に存在しているファイルを指定した場合、エラーとなります。

復帰コード

本関数からの復帰コードは、特殊レジスタPROGRAM-STATUSを使用して受け取ります。

復帰値	意味
0	ファイルのアンロードに成功しました。
-1	ファイルのアンロードに失敗しました。

復帰値-1が返却された場合、エラーの詳細は、COBF-MESSAGEに文字列で設定されます。



注意

指定例

- 相対ファイルからレコード順ファイルを作成する

入力ファイル名 : C:¥RELFILE
出力ファイル名 : C:¥OUTFILE
ファイル編成 : 相対ファイル
レコード形式 : 固定長
レコード長 : 80バイト

```
MOVE LOW-VALUE TO COBF-INF.  
MOVE "C:¥RELFILE" TO COBF-INPUT-FILENAME.
```

```
MOVE "R" TO COBF-INPUT-ATTR.  
MOVE "F" TO COBF-INPUT-RECFM.  
MOVE 80 TO COBF-INPUT-RECLEN.  
MOVE "C:¥OUTFILE" TO COBF-OUTPUT-FILENAME.  
CALL "COB_FILE_UNLOAD" USING BY REFERENCE COBF-INF.
```

7.8.5.4 ファイル整列関数

機能

レコード中の任意のデータ項目をキーとして、ファイル内のレコードを昇順または降順に整列し、整列された結果を可変長形式のレコード順ファイルに出力します。

指定方法

呼出しの記述

```
CALL "COB_FILE_SORT" USING BY REFERENCE COBF-INF.
```

呼出し時のデータ設定

COBF-INPUT-FILENAME

整列するファイルのパス名を指定します。指定するファイルは、レコード順ファイル、行順ファイル、相対ファイルまたは索引ファイルのどれかでなければなりません。

COBF-INPUT-ATTR

整列するファイルのファイル編成を、以下の文字で指定します。

"S" : レコード順ファイル

"L" : 行順ファイル

"R" : 相対ファイル

"I" : 索引ファイル

COBF-INPUT-RECFM

整列するファイルのレコード形式を、以下の文字で指定します。

"F" : 固定長

"V" : 可変長

COBF-INPUT-RECLEN

整列するファイルのレコード長を指定します。可変長の場合は、最大レコード長を指定します。整列するファイルが索引ファイルの場合は指定できません。

COBF-OUTPUT-FILENAME

整列結果を出力するレコード順ファイルのパス名を指定します。既に存在しているファイルを指定した場合、エラーとなります。

COBF-SORT-KEYNUM

整列キーの個数を指定します。



注意

指定できる値の最大は64です。

COBF-SORT-OFFSET

キーとする任意のデータ項目のレコードの先頭からの相対位置を指定します。

COBF-SORT-KEYLEN

キーとする任意のデータ項目の大きさを指定します。なお、キー項目にPIC 1()BITを指定した場合、1バイトの大きさに対するマスク値を指定します。

COBF-SORT-KEYATTR

任意のキーの項目属性を指定します。項目属性の指定には以下の条件名を使用します。

条件名	値	意味
COBF-SORT-KEYATTR-PIC1	1	PIC 1()
COBF-SORT-KEYATTR-PIC1B	2	PIC 1() BIT
COBF-SORT-KEYATTR-PICX	3	PIC X()
COBF-SORT-KEYATTR-PICN	4	PIC N()
COBF-SORT-KEYATTR-PIC9	5	PIC 9()
COBF-SORT-KEYATTR-PICS9	6	PIC S9()
COBF-SORT-KEYATTR-PICS9L	8	PIC S9() LEADING
COBF-SORT-KEYATTR-PICS9T	10	PIC S9() TRAILING
COBF-SORT-KEYATTR-PICS9LS	12	PIC S9() LEADING SEPARATE
COBF-SORT-KEYATTR-PICS9TS	14	PIC S9() TRAILING SEPARATE
COBF-SORT-KEYATTR-PIC9PD	15	PIC 9() PACKED-DECIMAL
COBF-SORT-KEYATTR-PICS9PD	16	PIC S9() PACKED-DECIMAL
COBF-SORT-KEYATTR-PIC9B	17	PIC 9() BINARY
COBF-SORT-KEYATTR-PICS9B	18	PIC S9() BINARY
COBF-SORT-KEYATTR-PIC9C5	19	PIC 9() COMP-5
COBF-SORT-KEYATTR-PICS9C5	20	PIC S9() COMP-5
COBF-SORT-KEYATTR-PICC1	21	COMP-1
COBF-SORT-KEYATTR-PICC2	22	COMP-2

COBF-SORT-KEYSEQ

整列順序を、以下の文字で指定します。

"A": 昇順にレコードを整列する

"D": 降順にレコードを整列する

復帰コード

本関数からの復帰コードは、特殊レジスタPROGRAM-STATUSを使用して受け取ります。

復帰値	意味
0	ファイルの整列に成功しました。
-1	ファイルの整列に失敗しました。

復帰値-1が返却された場合、エラーの詳細は、COBF-MESSAGEに文字列で設定されます。



注意

指定例

- ```
入力ファイル名 : RELFILE
出力ファイル名 : OUTFILE
整列条件 : [第１キー] 項目属性 : 日本語
 レコード内オフセット : 0
 サイズ : 2
 整列順序 : 昇順
 [第２キー] 項目属性 : 英数字
 レコード内オフセット : 10
 サイズ : 5
 整列順序 : 降順

ファイル属性 : ファイル編成 : 相対ファイル
 レコード形式 : 可変長
 レコード長 : 80
```

```

MOVE LOW-VALUE TO COBF-INF.
MOVE "RELFILE" TO COBF-INPUT-FILENAME.
MOVE "R" TO COBF-INPUT-ATTR.
MOVE "V" TO COBF-INPUT-RECFM.
MOVE 80 TO COBF-INPUT-RECLEN.
MOVE "OUTFILE" TO COBF-OUTPUT-FILENAME.
MOVE 2 TO COBF-SORT-KEYNUM.
SET COBF-SORT-KEYATTR-PICN(1) TO TRUE.
MOVE 0 TO COBF-SORT-OFFSET(1).
MOVE 2 TO COBF-SORT-KEYLEN(1).
MOVE "A" TO COBF-SORT-KEYSEQ(1).
SET COBF-SORT-KEYATTR-PICX(2) TO TRUE.
MOVE 10 TO COBF-SORT-OFFSET(2).
MOVE 5 TO COBF-SORT-KEYLEN(2).
MOVE "D" TO COBF-SORT-KEYSEQ(2).
CALL "COB FILE SORT" USING BY REFERENCE COBF-INF.

```

## 機能

### 指定方法

CALL "COB FILE REORGANIZE" USING BY REFERENCE COBF-INF.

COBF-INPUT-FILENAME

COBF-OUTPUT-FILENAME

復帰コード”

| 復帰値 | 意味               |
|-----|------------------|
| 0   | ファイルの再編成に成功しました。 |
| -1  | ファイルの再編成に失敗しました。 |

復帰値-1が返却された場合、エラーの詳細は、COBF-MESSAGEに文字列で設定されます。



## 例

### 指定例

- 索引ファイルを再編成し、OUTFILEに出力する

入力ファイル名 : C:¥IXDFILE  
出力ファイル名 : C:¥OUTFILE

```
MOVE LOW-VALUE TO COBF-INF.
MOVE "C:¥IXDFILE" TO COBF-INPUT-FILENAME.
MOVE "C:¥OUTFILE" TO COBF-OUTPUT-FILENAME.
CALL "COB_FILE_REORGANIZE" USING BY REFERENCE COBF-INF.
```

## 7.8.5.6 ファイルコピー関数

### 機能

ファイルを複写します。

### 指定方法

#### 呼出しの記述

```
CALL "COB_FILE_COPY" USING BY REFERENCE COBF-INF.
```

#### 呼出し時のデータ設定

##### COBF-INPUT-FILENAME

複写元ファイルのパス名を指定します。

##### COBF-OUTPUT-FILENAME

複写先ファイルのパス名を指定します。既に存在しているファイルを指定した場合、エラーとなります。

#### 復帰コード

本関数からの復帰コードは、特殊レジスタPROGRAM-STATUSを使用して受け取ります。

| 復帰値 | 意味              |
|-----|-----------------|
| 0   | ファイルの複写に成功しました。 |
| -1  | ファイルの複写に失敗しました。 |

復帰値-1が返却された場合、エラーの詳細は、COBF-MESSAGEに文字列で設定されます。



## 注意

ファイル名にワイルドカード(?,\*)を指定することはできません。



## 例

### 指定例

- INFILEをOUTFILEにコピーする

複写元ファイル名 : C:¥INFILE  
複写先ファイル名 : C:¥OUTFILE

```
MOVE LOW-VALUE TO COBF-INF.
MOVE "C:¥INFILE" TO COBF-INPUT-FILENAME.
MOVE "C:¥OUTFILE" TO COBF-OUTPUT-FILENAME.
CALL "COB_FILE_COPY" USING BY REFERENCE COBF-INF.
```

## 7.8.5.7 ファイル移動関数

### 機能

ファイルを移動します。

### 指定方法

呼出しの記述

```
CALL "COB_FILE_MOVE" USING BY REFERENCE COBF-INF.
```

呼出し時のデータ設定

COBF-INPUT-FILENAME

移動元ファイルのパス名を指定します。

COBF-OUTPUT-FILENAME

移動先ファイルのパス名を指定します。既に存在しているファイルを指定した場合、エラーとなります。

復帰コード

本関数からの復帰コードは、特殊レジスタPROGRAM-STATUSを使用して受け取ります。

| 復帰値 | 意味              |
|-----|-----------------|
| 0   | ファイルの移動に成功しました。 |
| -1  | ファイルの移動に失敗しました。 |

復帰値-1が返却された場合、エラーの詳細は、COBF-MESSAGEに文字列で設定されます。



### 注意

ファイル名にワイルドカード(?,\*)を指定することはできません。



### 例

指定例

- ・ C:¥INFILEをF:¥OUTFILEに移動する

移動元ファイル名 : C:¥IXDFILE  
移動先ファイル名 : F:¥OUTFILE

```
MOVE LOW-VALUE TO COBF-INF.
MOVE "C:¥INFILE" TO COBF-INPUT-FILENAME.
MOVE "F:¥OUTFILE" TO COBF-OUTPUT-FILENAME.
CALL "COB_FILE_MOVE" USING BY REFERENCE COBF-INF.
```

## 7.8.5.8 ファイル削除関数



## 機能

ファイルを削除します。

## 指定方法

呼出しの記述

```
CALL "COB_FILE_DELETE" USING BY REFERENCE COBF-INF.
```

呼出し時のデータ設定

COBF-INPUT-FILENAME

削除するファイルのパス名を指定します。

復帰コード

本関数からの復帰コードは、特殊レジスタPROGRAM-STATUSを使用して受け取ります。

| 復帰値 | 意味              |
|-----|-----------------|
| 0   | ファイルの削除に成功しました。 |
| -1  | ファイルの削除に失敗しました。 |

復帰値-1が返却された場合、エラーの詳細は、COBF-MESSAGEに文字列で設定されます。



### 注意

ファイル名にワイルドカード(?,\*)を指定することはできません。



### 例

指定例

C:¥INFILEを削除する

```
MOVE LOW-VALUE TO COBF-INF.
MOVE "C:¥INFILE" TO COBF-INPUT-FILENAME.
CALL "COB_FILE_DELETE" USING BY REFERENCE COBF-INF.
```

## 7.9 他のファイルシステムの使用法

NetCOBOLでは、製品に同梱されているCOBOLファイルシステム以外にも、ファイル名に特定の文字列を指定することで、他のファイルシステムを使用することができます。

利用できるファイルシステムを以下に示します。

Btrieve

Btrieveは、Pervasive Software社のレコードマネジメントシステムです。

RDM

RDMは、SymfoWARE7000を使用したデータベースシステム、またはPowerRW+で扱うことができるファイルです。

PowerRDBconnector

PowerRDBconnectorと連携することで、入出力機能を使用してデータベースをアクセスすることが可能となります。PowerRDBconnectorの使用法、機能範囲については、“PowerRDBconnector説明書”を参照してください。

PowerRDBconnector連携では、デッドロック出口に対応しています。PowerRDBconnector連携でデッドロック状態が発生すると、デッドロック出口が登録されている場合はデッドロック出口のスケジュールを行います。デッドロック出口については、“[22.2.14 デッドロック出口](#)”を参照してください。



## 注意

PowerRDBconnector連携では、デッドロック出口スケジュールを行うためのデッドロック出口スケジュールサブルーチンの呼出しは不要です。入出力文の実行でデッドロック状態が発生した場合、COBOLランタイムシステムが自動的にデッドロック出口スケジュールを行います。

### 外部ファイルハンドラ

外部ファイルハンドラを利用して、Micro Focus COBOLが公開しているFCD構造を持ったファイルシステムを呼び出すことができます。外部ファイルハンドラで利用できる入出力機能の範囲は、結合するファイルシステムの仕様に依存します。

また、サーバ間パイプ機能であるXLデータパイプと連携してデータの受け渡しを行うことができます。

COBOLファイルシステム、BtrieveおよびRDMのそれぞれのファイルシステムで利用できる入出力機能の範囲を“表7.10 各ファイルシステムの機能差”に示します。

XLデータパイプについては、“7.9.3 XLデータパイプ”を参照してください。

表7.10 各ファイルシステムの機能差

| 分類        | 項目               |                      | COBOLファイルシステム   | Btrieve(注1)     | RDM(注1)         |
|-----------|------------------|----------------------|-----------------|-----------------|-----------------|
| ファイル      | ファイルの最大サイズ(バイト数) | レコード順ファイル            | 1G(注11)         | 128G            | 2G (注4)         |
|           |                  | レコード順ファイル(ファイルの高速処理) | システム制限まで        | —               | —               |
|           |                  | 行順ファイル               | 1G(注11)         | —               | —               |
|           |                  | 行順ファイル(ファイルの高速処理)    | システム制限まで        | —               | —               |
|           |                  | 相対ファイル               | 1G              | —               | 2G (注4)         |
|           |                  | 索引ファイル               | 約1.7G(注11)      | 128G            | 2G (注4)         |
| レコード      | レコード形式           | 固定長レコード形式            | ○               | ○               | ○               |
|           |                  | 可変長レコード形式            | ○               | ○               | —               |
|           | レコードの最大長(バイト数)   | 固定長レコード形式            | 32,760          | 32,760          | 32,760          |
|           |                  | 可変長レコード形式            | 32,760          | 32,760          | —               |
|           | レコードの最小長(バイト数)   | レコード順                | 1               | 4               | 1               |
|           |                  | 行順                   | 0               | —               | —               |
|           |                  | 相対                   | 1               | —               | 1               |
|           |                  | 索引                   | キーを構成する項目までの大きさ | キーを構成する項目までの大きさ | キーを構成する項目までの大きさ |
| ファイル管理記述項 | SELECT句          | OPTIONAL指定           | ○               | ○               | ○ (注2)          |
|           | ASSIGN句          | ファイル識別名指定            | ○               | ○               | ○               |
|           |                  | ファイル識別名定数指定          | ○               | ○               | ○               |
|           |                  | データ名指定               | ○               | ○               | ○               |
|           |                  | DISK指定               | ○               | —               | —               |
|           | FILE STATUS句     | ファイル状態               | ○               | ○ (注3)          | ○ (注3)          |
|           | LOCK MODE句       | AUTOMATIC            | ○               | ○               | ○               |
|           |                  | EXCLUSIVE            | ○               | ○               | ○               |

| 分類 | 項目                    |                     | COBOLファイルシステム | Btrieve(注1) | RDM(注1) |
|----|-----------------------|---------------------|---------------|-------------|---------|
|    |                       | MANUAL              | ○             | ○           | —       |
|    | RECORD KEY句           | 指定できるデータの最大個数       | 254 (注5)      | 119         | 128     |
|    |                       | 指定できるデータ総長の最大(バイト数) | 254           | 255         | 255     |
|    | ALTERNATE RECORD KEY句 | 指定できるデータの最大個数       | 254 (注5)      | 119         | 128     |
|    |                       | 指定できるデータ総長の最大(バイト数) | 254           | 255         | 255     |
|    | レコードキーの項目             | 英数字                 | ○             | ○           | ○       |
|    |                       | 日本語                 | ○             | ○           | ○       |
|    |                       | 符号なし外部10進数          | ○             | ○           | ○       |
|    |                       | 符号付き外部10進数          | —             | ○ (注6)      | ○       |
|    |                       | 符号なし内部10進数          | ○             | ○ (注6)      | ○       |
|    |                       | 符号付き内部10進数          | —             | ○           | ○       |
|    |                       | 符号なし2進数(正順格納形式)     | ○             | ○           | ○       |
|    |                       | 符号付き2進数(正順格納形式)     | —             | ○           | ○       |
|    |                       | 符号なし2進数(逆順格納形式)     | —             | ○           | ○       |
|    |                       | 符号付き2進数(逆順格納形式)     | —             | ○           | ○       |
| 文  | READ文                 | WITH LOCK指定         | ○             | ○           | —       |
|    | START文                | キー全体を指定             | ○             | ○           | ○       |
|    |                       | キーの一部を指定            | ○             | ○           | —       |
|    | DELETE文               | キー値が重複しているレコードを削除   | ○             | ○ (注7)      | ○       |
|    | UNLOCK文               |                     | ○             | ○           | —       |
| 機能 | スレッド                  | シングルスレッド            | ○             | ○           | ○       |
|    |                       | マルチスレッド             | ○             | ○           | ○ (注8)  |
|    | トランザクション管理            | トランザクション開始指示        | —             | ○ (注9)      | ○ (注9)  |
|    |                       | COMMIT指示            | —             | ○ (注9)      | ○ (注9)  |
|    |                       | ROLLBACK指示          | —             | ○ (注9)      | ○ (注9)  |

○ : サポート  
 — : 非サポート

- ・ 注1  
Btrieveについては、“Pervasive PSQL V9”の情報をもとにしています。  
RDMについては、“RDB/7000”、“SymfoWARE7000”および“PowerRW+”の情報をもとにしています。
- ・ 注2  
RDMでは、OPEN INPUTで不定ファイルをオープンできます。ただし、OPEN I-OまたはOPEN EXTENDでは不定ファイルをオープンできません。
- ・ 注3  
BtrieveおよびRDMは、FILE STATUS=02を返却しません。また、RDMの場合、排他エラー(ファイル排他/レコード排他とも)を示すFILE STATUS値は"92"です。
- ・ 注4  
ファイル管理情報で使用するため、実際の制限値は多少小さくなります。
- ・ 注5  
COBOLファイルシステムでは、RECORD KEY句とALTERNATE RECORD KEY句に指定できるデータ個数の総和が最大255です。
- ・ 注6  
BtrieveファイルのNUMERICタイプは、COBOLではSEPARATE指定のない符号付き外部10進項目のデータ型に相当しますが、NUMERICの内部形式は88コンソーシアム形式と呼ばれる内部形式となっており、NetCOBOLの内部形式とは異なります。そのため、Btrieveファイルを使用して、SEPARATE指定のない符号付き外部10進項目を入力または出力項目として扱う場合、書込み前および読込み後にデータを変換する必要があります。データ変換の詳細については、“[7.9.1.2 外部10進項目のデータ形式変換](#)”を参照してください。
- ・ 注7  
Btrieveでは、ファイル位置指示子が確定しているレコードを乱呼出し法のDELETE文で削除した場合、直前までのファイル位置指示子は不定となります。また、副キーで順呼出し法のREAD文を実行し、重複している副キーの先頭から2番目以降のレコードを読み込んだ後、以下のどれかを実行した場合も、ファイル位置指示子は不定となります。
  - 乱呼出し法のREWRITE文
  - 乱呼出し法のDELETE文
  - 乱呼出し法のWRITE文
- ・ 注8  
RDMのマルチスレッドモードでは、スレッド間で同じファイル結合子を操作してファイルを共用することはできません。
- ・ 注9  
BtrieveまたはRDMに用意されている機能(関数)を、CALL文で呼び出す必要があります。  
Btrieveでトランザクション操作を行うためには、別途AG-TECH社から発売されているプログラムサポートを購入する必要があります。RDMでトランザクション操作を行う方法については、“RDB/7000説明書”、“SymfoWARE7000のRDB-7000連携ヘルプ”または“PowerRW+デベロッパーズマニュアル”を参照してください。
- ・ 注10  
COBOLファイルシステムでは、アクセスできなくなった索引ファイルを復旧する機能を提供しています。BtrieveおよびRDMでは、トランザクション管理の機能と合わせたリカバリ機構や各種ツールが提供されています。
- ・ 注11  
実行環境変数 @CBR\_FILE\_LFS\_ACCESS を指定した場合、システム制限まで拡張することができます。

## 7.9.1 Btrieveファイル

Btrieveファイルは、順ファイルおよび索引ファイルとして利用できます。Btrieveで利用できる機能については、“[表7.10 各ファイルシステムの機能差](#)”を参照してください。

ここでは、Btrieveファイルの使用方法について説明します。

### 7.9.1.1 ファイル環境の指定

COBOLファイルシステムを使用するか、Btrieveファイルシステムを使用するかは、ファイル管理記述項のASSIGN句の記述によって決まります。

#### ASSIGN句にファイル識別名定数を記述した場合

ファイル識別名定数は、以下に示す形式で指定します。

---

$$\text{ASSIGN TO "[パス名] ファイル名} \left\{ \begin{array}{l} \text{指定なし} \\ \text{,BTRV} \end{array} \right\} \text{"}$$

---

##### ファイル名

入出力対象となるファイルの実際のファイル名を指定します。

##### 指定なし

COBOLファイルシステムを使用する場合には何も指定しません。

##### BTRV

Btrieveレコードマネージャを使用します。

#### ASSIGN句にデータ名を記述した場合

データ名に設定するファイル名は、ファイル識別名定数の場合と同じ形式で指定します。

---

$$\text{MOVE "[パス名] ファイル名} \left\{ \begin{array}{l} \text{指定なし} \\ \text{,BTRV} \end{array} \right\} \text{" TO データ名}$$

---

#### ASSIGN句にDISKを記述した場合

Btrieveレコードマネージャを使用することはできません。COBOLファイルシステムが使用されます。

#### ASSIGN句にファイル識別名を記述した場合

ファイル識別名を環境変数情報名とした環境変数に、ファイル識別名定数の場合と同様の形式でファイル名を指定します。

##### 環境変数の指定形式

---

$$\text{ファイル識別名} = \text{[パス名] ファイル名} \left\{ \begin{array}{l} \text{指定なし} \\ \text{,BTRV} \end{array} \right\}$$

---

### 7.9.1.2 外部10進項目のデータ形式変換

SEPARATE指定のない符号付き外部10進項目を扱うために、内部形式を変換する2種類の方法を用意しています。

- NetCOBOLで扱う形式からBtrieveファイルで扱う形式(88コンソーシアム形式)に変換する方法
- 88コンソーシアム形式からNetCOBOLで扱う形式に変換する方法

このデータ変換は、Btrieveファイルを使用してSEPARATE指定のない符号付き外部10進項目を入力または出力項目として扱う場合、書込み前および読込み後に行う必要があります。

BtrieveのNUMERICタイプのデータの内部形式については、"Btrieve"または"Pervasive.SQL"のマニュアルをお持ちの方は、こちらを参照してください。お持ちでない方は、技術員(SE)に連絡してください。

## 記述形式

- 88コンソーシアム形式からNetCOBOLの内部形式への変換

|                                               |
|-----------------------------------------------|
| CALL "#DEC88TOFJ" USING [BY REFERENCE]    一意名 |
|-----------------------------------------------|

- NetCOBOLの内部形式から88コンソーシアム形式への変換

|                                              |
|----------------------------------------------|
| CALL "#DECJTO88" USING [BY REFERENCE]    一意名 |
|----------------------------------------------|

## 機能概要

- CALL文で呼ぶプログラム名が"#DEC88TOFJ"の場合には、一意名または一意名中に含まれるすべてのSEPARATE指定のない符号付き外部10進項目の内部形式を、88コンソーシアム形式からNetCOBOLの内部形式に変換します。
- CALL文で呼ぶプログラム名が"#DECJTO88"の場合には、一意名または一意名中に含まれるすべてのSEPARATE指定のない符号付き外部10進項目の内部形式を、NetCOBOLの内部形式から88コンソーシアム形式に変換します。
- 一意名が基本項目の場合、その項目がSEPARATE指定のない符号付き外部10進項目でなければ、何も行われません。
- 一意名が集団項目で、かつ従属する項目に以下の項目が含まれている場合、その項目は変換の対象とはなりません。
  - SEPARATE指定のない符号付き外部10進項目ではない項目
  - REDEFINES句が指定された項目、およびその項目に従属する項目
  - RENAMES句が指定された項目
- 以下の場合、実行結果は保証されません。
  - CALL文で呼ぶプログラム名が一意名で指定されており、その一意名により"#DEC88TOFJ"または"#DECJTO88"が指定されている。
  - 上記記述形式とは異なる形式で、"#DEC88TOFJ"または"#DECJTO88"を呼び出している(USING句に複数の一意名が指定されている、BY CONTENT句が指定されているなど)。
  - 一意名が集団項目であり、従属する項目にDEPENDING指定付きのOCCURS句を持つ項目が存在する。
  - 一意名が定数節に定義されたデータ項目である。
  - 一意名が部分参照付けされたデータ項目である。

## 使用上の注意

- 本関数の呼出しは、レコード(キーデータ項目)をBtrieveファイルに渡す直前または受け取った直後に行ってください。Btrieveファイルから入力した直後や本関数で変換済みの88コンソーシアム形式の外部10進項目をCOBOLで扱うことはできません。  
なお、「COBOLで扱うことができない」とは、演算したり比較したり、また、外部10進項目として転記した場合の結果が保証されないことを意味します。88コンソーシアム形式の外部10進項目を含むレコードを他のレコードへ集団項目転記した場合は、転記されたレコード内で88コンソーシアム形式の外部10進項目としての値が保証されます。
- Btrieveファイルに対しては、整列併合機能は利用できません。
- 本関数呼出しを使用して、リンク時に"#DECJTO88"または"#DEC88TOFJ"の外部参照エラーとなった場合は、呼び出す関数名が正しくないか、本関数の使い方を機能概要で示した規則に違反して使用している場合です。



### 例

使用例

|                                                                 |
|-----------------------------------------------------------------|
| :<br>FILE-CONTROL.<br>SELECT ファイル    ASSIGN TO    "ファイル名, BTRV" |
|-----------------------------------------------------------------|

```

 ORGANIZATION IS INDEXED
 ACCESS MODE SEQUENTIAL
 RECORD KEY IS 主キー OF レコード.

DATA DIVISION.
FILE SECTION.
FD ファイル.
01 レコード.
 02 主キー PIC 9(4).
 02 データ 1 PIC S9(8).
 02 データ 2 PIC X(50).
WORKING-STORAGE SECTION.
01 作業域.
 02 主キー PIC 9(4).
 02 データ 1 PIC S9(8).
 02 データ 2 PIC X(50).
PROCEDURE DIVISION.
 :
*Btrieveファイルへの書き込み
 OPEN OUTPUT ファイル.
 CALL "#DECFT088" USING 作業域. ... [1]
 WRITE レコード FROM 作業域.
 CLOSE ファイル.
 :
*Btrieveファイルからの読み込み
 OPEN INPUT ファイル.
 MOVE 6 TO 主キー OF レコード.
 START ファイル.
 READ ファイル INTO 作業域.
 CALL "#DEC88T0FJ" USING 作業域. ... [2]
 DISPLAY "データ 1 : " データ 1 OF 作業域.
 CLOSE ファイル.
 :

```

[1] Btrieveファイルへの書き込みの前に、書き込まれるデータとなる“作業域”内のすべてのSEPARATE指定のない符号付き外部10進項目を、NetCOBOLの内部形式から88コンソーシアム形式に変換します。

[2] Btrieveファイルからのレコード読み込み後、読み込み先である“作業域”内のすべてのSEPARATE指定のない符号付き外部10進項目を、88コンソーシアム形式からNetCOBOLの内部形式に変換します。

### 7.9.1.3 注意事項

LOCK MODE句にAUTOMATICを指定した場合、ロックされているレコード以外のレコードに対して、WRITE文/REWRITE文/DELETE文/START文を実行しても、ロックは解除されません。ただし、ロックされているレコードに対して、REWRITE文/DELETE文を実行した場合、ロックは解除されます。

## 7.9.2 RDMファイル

RDMファイルは、順ファイル、相対ファイルおよび索引ファイルとして利用できます。RDMで利用できる機能については、“[表7.10 各ファイルシステムの機能差](#)”を参照してください。

ここでは、RDMファイルの使用方法について説明します。



### 参照

RDMファイルの文字コードの取り扱いや各種固有機能の詳細な使用方法については、“RDB/7000”、“SymfoWARE7000”または“PowerRW+”のマニュアルを参照してください。

### 7.9.2.1 ファイル環境の指定

COBOLファイルシステムを使用するか、RDMファイルシステムを使用するかは、ファイル管理記述項のASSIGN句の記述によって決まります。

#### ASSIGN句にファイル識別名定数を記述した場合

ファイル識別名定数は、以下に示す形式で指定します。

---

|                        |                                                                            |    |
|------------------------|----------------------------------------------------------------------------|----|
| ASSIGN TO "[パス名] ファイル名 | $\left\{ \begin{array}{l} \text{指定なし} \\ \text{,RDM} \end{array} \right\}$ | ". |
|------------------------|----------------------------------------------------------------------------|----|

---

#### ファイル名

入出力対象となるファイルのファイル名を指定します。

#### 指定なし

COBOLファイルシステムを使用する場合には何も指定しません。

#### RDM

RDMファイルシステムを使用します。

#### ASSIGN句にデータ名を記述した場合

データ名の設定するファイル名は、ファイル識別名定数の場合と同じ形式で指定します。

---

|                   |                                                                            |            |
|-------------------|----------------------------------------------------------------------------|------------|
| MOVE "[パス名] ファイル名 | $\left\{ \begin{array}{l} \text{指定なし} \\ \text{,RDM} \end{array} \right\}$ | " TO データ名. |
|-------------------|----------------------------------------------------------------------------|------------|

---

#### ASSIGN句にDISKを記述した場合

RDMファイルシステムを使用することはできません。COBOLファイルシステムが使用されます。

#### ASSIGN句にファイル識別名を記述した場合

ファイル識別名を環境変数情報名とした環境変数には、ファイル識別名定数の場合と同じ形式でファイル名を指定します。

#### 環境変数の指定形式

---

|                       |                                                                            |
|-----------------------|----------------------------------------------------------------------------|
| ファイル識別名 = [パス名] ファイル名 | $\left\{ \begin{array}{l} \text{指定なし} \\ \text{,RDM} \end{array} \right\}$ |
|-----------------------|----------------------------------------------------------------------------|

---

#### RDMファイルの拡張機能を使用する場合

上記の指定の他に以下の環境変数(CRDB\_XLIB\_OVR)を指定します。

|                                                   |
|---------------------------------------------------|
| CRDB_XLIB_OVR= [パス名] ファイル名, RDM, 拡張機能を表す文字列 [ ; ] |
|---------------------------------------------------|



## ファイル名

拡張機能を利用するファイルを指定します。

## 拡張機能を表す文字列

“RDB/7000”、“SymfoWARE7000”または“PowerRW+”のマニュアルを参照してください。

## 7.9.2.2 注意事項

- ・ COBOLプログラムで初めて使用する相対ファイルは、まず“相対ファイルの創成”を行ってください。相対ファイルの創成は、OUTPUT指定のOPEN文を実行すると行われます。創成していないファイルをI-OまたはEXTEND指定のOPEN文で開き、WRITE文を実行しても、“レコードが存在しない”(入出力状態FS=23または、実行時エラーJMP0324I-I/U)というエラーになります。

- ・ ディスク容量不足が発生した場合、CLOSE文の実行時に以下のエラーメッセージが出力されます。

JMP0310I-I/U 'アクセス名またはファイル名'ファイルで'CLOSE'エラーが発生しました.'ERFLD=1C'

- ・ RDMファイルの排他制御は、COBOLファイルシステムと比較して動作が異なる場合があります。詳細については、“RDB/7000説明書”を参照してください。

## 7.9.3 XLデータパイプ

XLデータパイプとは、マルチサーバ形態の業務システムで、OSIV系システム(グローバルサーバまたはPRIMEFORCEシリーズで動作するOS)とSolaris/Windowsサーバで実行するアプリケーションの間で、中間ファイルを必要としない動的なデータの受け渡しを実現することができる製品です。XLデータパイプと連携することで、異種サーバ間のCOBOLアプリケーションでデータを直接流通することが可能となり、従来、FTP等を利用して別ステップで行っていたデータ転送処理が不要となります。

XLデータパイプについては“XLデータパイプ説明書”を参照してください。

ここでは、XLデータパイプの使用方法、ならびに機能範囲について説明します。

### 7.9.3.1 使用方法

XLデータパイプでは、データの受け渡しを行うために、サーバ間パイプデータセットを提供しています。COBOLでは、このサーバ間パイプデータセットを定義したパイプIDをファイル識別名としてASSIGN句に定義するだけで、XLデータパイプを使用することができます。



サーバ間パイプデータセットを定義したパイプIDを環境変数としてファイルを割り当てないでください。割り当てた環境変数をファイル識別名としてASSIGN句に定義した場合、COBOLファイルとして処理されます。

### 7.9.3.2 機能範囲

XLデータパイプの機能範囲を以下に示します。

|           |                     |                                                    |
|-----------|---------------------|----------------------------------------------------|
| ファイル編成    | レコード順ファイル<br>行順ファイル |                                                    |
| レコード形式    | 固定長形式<br>可変長形式      |                                                    |
| ファイル管理記述項 | SELECT句             | ファイル識別名(注1)                                        |
|           | LOCK MODE句          | 指定しても意味を持ちません                                      |
| 入出力文      | OPEN文               | ・OUTPUT/INPUTモード(注2)<br>・WITH LOCK指定は指定しても意味を持ちません |
|           | READ文               | WITH LOCK/WITH NO LOCK指定は意味を持ちません                  |
|           | WRITE文              |                                                    |
|           | REWRITE文            | 指定不可                                               |

|  |         |               |
|--|---------|---------------|
|  | UNLOCK文 | 指定しても意味を持ちません |
|--|---------|---------------|

注1：ファイル識別名定数、データ名、DISK指定は指定できません。

注2：オープンモードI-Oは指定できません。

## 7.9.4 外部ファイルハンドラ

外部ファイルハンドラを使用して、Micro Focus COBOLが公開しているFCD構造を持ったファイルシステムを呼び出すことができます。外部ファイルハンドラは、レコード順ファイル、行順ファイル、相対ファイル、索引ファイルに対応しています。

ここでは、外部ファイルハンドラの使い方と指定方法について説明します。

### 使い方

#### プログラム中のファイル参照子にファイル識別名を指定する場合

環境変数の設定時に、割り当てるファイルのファイル名に続き、“,EXFH”を指定します。

```
ファイル識別名=ファイル名, EXFH
```

#### プログラム中のファイル参照子にファイル識別名定数を指定する場合

プログラム中のファイル識別名定数の指定時に、割り当てるファイルのファイル名に続き、“,EXFH”を指定します。

```
ASSIGN TO “ファイル名, EXFH”.
```

#### プログラム中のファイル参照子にデータ名を指定する場合

プログラム中のデータ名への値の設定時に、割り当てるファイルのファイル名に続き、“,EXFH”を指定します。

```
MOVE “ファイル名, EXFH” TO データ名.
```

### 指定方法

外部ファイルハンドラの指定方法には、実行環境に有効とする指定方法とファイル単位に有効とする指定方法があります。どちらも、DLL名と入口名を指定する必要があります。

2つとも同時に指定されている場合は、ファイル単位の指定が優先されます。

#### 実行環境に有効とする方法

実行時に以下の環境変数を設定します。

```
@CBR_EXFH_API=入口名
```

入口名：結合するファイルシステムの入口名を指定します。(必須)

```
@CBR_EXFH_LOAD=DLL名
```

DLL名：結合するファイルシステムのDLL名を指定します。

DLLのパスには、絶対パス、相対パスのどちらも指定することができます。相対パスを用いた場合、カレントフォルダからの相対パスとなります。



環境変数@CBR\_EXFH\_LOADが指定されていない場合、DLL名には、環境変数@CBR\_EXFH\_APIに指定された入口名を“入口名.DLL”とみなして処理します。



例1) 結合するファイルシステムの入口名を“flsys”、DLL名を“filesys.dll”とする場合

```
@CBR_EXFH_API=flsys
@CBR_EXFH_LOAD=filesys.dll
```

例2) 結合するファイルシステムの入口名を“file”、DLL名を“file.dll”とする場合

```
@CBR_EXFH_API=file
```

#### ファイル単位に有効とする方法

外部ファイルハンドラ情報ファイルを作成し、以下のように割り当てます。

```
ファイル識別名=ファイル名, EXFH, INF (外部ファイルハンドラ情報ファイル名)
```

外部ファイルハンドラ情報ファイルは、以下の内容のテキストファイルです。

```
[EXFH]
@CBR_EXFH_API=入口名
@CBR_EXFH_LOAD=DLL名
```

入口名 : 結合するファイルシステムの入口名を指定します。(必須)

DLL名 : 結合するファイルシステムのDLL名を指定します。



#### 注意

環境変数@CBR\_EXFH\_LOADが指定されていない場合、DLL名には、環境変数@CBR\_EXFH\_APIに指定された入口名を“入口名.DLL”とみなして処理します。



#### 例

外部ファイルハンドラ情報ファイル名を“aflsys.inf”、ファイル“Afile”に対する結合するファイルシステムの入口名を“aflsys”、DLL名を“afilesys.dll”とする場合

```
ファイル識別名=Afile, EXFH, INF (aflsys. inf)
```

ファイル“aflsys.inf”の内容

```
[EXFH]
@CBR_EXFH_API=aflsys
@CBR_EXFH_LOAD=afilesys.dll
```

#### 注意事項

- 使用できる外部ファイルハンドラはDLLファイルのみです。Micro Focus COBOLと異なり、オブジェクトファイルを使用することはできません。
- 外部ファイルハンドラは、Unicode環境でコンパイルされたCOBOLアプリケーションでは使用できません。
- マルチスレッド用オプションを指定してコンパイルされたCOBOLアプリケーションから外部ファイルハンドラを使う場合、結合するファイルシステムもマルチスレッドに対応していなければなりません。
- FILE STATUS句を使う場合、外部ファイルハンドラから返される入出力状態値が返却されます。このため、“付録B 入出力状態一覧”の値とは異なる場合があります。
- 索引ファイルではFIRSTの指定されたSTART文はサポートされていません。

## 7.10 索引ファイルの復旧

ここでは、索引ファイル復旧関数および索引ファイル簡易復旧関数について説明します。

## 7.10.1 索引ファイル復旧関数(CFURCOV)

ここでは、索引ファイル復旧関数について説明します。

### 機能

索引ファイル復旧関数は、正常に閉じることができなかったために使用できない状態にある索引ファイルを使用可能にするため、ファイルの先頭から正常な部分を再生し、異常部分を別ファイルに出力します。つまり、COBOLファイルユーティリティの[復旧]コマンドと同じ機能を持っています。

再度使用できない状態にある索引ファイルを開いた場合は、入出力状態に“39”または“90”が返却されます。

### 記述形式

```
#include "f3bifutc.h" /* 関数宣言 */
signed long int CFURCOV(char *ixdfilename, char *blkdatname, char *message);
```

ixdfilename : 復旧するファイルの名前  
blkdatname : 復旧できなかったデータを格納するファイルの名前  
message : 索引ファイル復旧関数の結果（メッセージ）を格納する領域

### 関数名

CFURCOV

#### 第一パラメタ(ixdfilename)

復旧する索引ファイルの名前(文字列)が格納されている領域のアドレスを指定します。文字列の終端位置には、NULL文字(0x00)または空白文字(0x20)を設定する必要があります。また、ファイル名に空白またはコンマ(,)を含む場合は、ファイル名を二重引用符で囲む必要があります。(例:"¥"C:¥¥ixd file¥"¥0")

#### 第二パラメタ(blkdatname)

復旧不可能であったレコードを書き込むファイルの名前(文字列)が格納されている領域のアドレスを指定します。文字列の終端位置には、NULL文字(0x00)または空白文字(0x20)を設定する必要があります。また、ファイル名に空白またはコンマ(,)を含む場合は、ファイル名を二重引用符で囲む必要があります。なお、復旧不可能であったレコードを書き込むファイルが必要なければ、0を指定します。

既に存在するファイルの名前を指定した場合、エラーとなります。

#### 第三パラメタ(message)

索引ファイル復旧関数の実行結果を示すメッセージを格納する領域のアドレスを指定します。格納する領域は、呼出し元で確保(512バイト必要)してください。

なお、メッセージが必要なければ、0を指定します。

### 復帰値

索引ファイル復旧関数の結果として、メッセージに対応するコードを返却します。

コードの値とメッセージの内容については、“[表7.11 索引ファイル復旧関数\(簡易復旧関数\)](#)が返却するメッセージの内容とコード”を参照してください。



### 例

```
#include "f3bifutc.h"
void callcobfrcov(void)
{
 char ixdfilename[512] = "c:¥¥ixdfile¥0";
 char blkdatname[512] = "c:¥¥blkdat¥0";
 char message[512];
 CFURCOV(ixdfilename, blkdatname, message);
 return;
}
```

## 7.10.2 索引ファイル簡易復旧関数(CFURCOVS)

ここでは、索引ファイル簡易復旧関数について説明します。

## 機能

索引ファイル簡易復旧関数は、再度使用できない状態にある索引ファイルを使用可能にするため、索引ファイル中のフラグをリセットします。索引ファイル復旧関数とは異なり、索引ファイル中の異常な箇所について修復しないため、復旧後の索引ファイルに対するアクセスで、データの矛盾でエラーになる場合があります。

再度使用できない状態にある索引ファイルを開いた場合は、入出力状態に“39”または“90”が返却されます。

## 記述形式

```
#include "f3bifutc.h" /* 関数宣言 */
signed long int CFURCOVS(char *ixdfilename, char *message);
```

ixdfilename : 復旧するファイルの名前

message : 索引ファイル復旧関数の結果（メッセージ）を格納する領域

## 関数名

CFURCOVS

## 第一パラメタ(ixdfilename)

復旧する索引ファイルの名前(文字列)が格納されている領域のアドレスを指定します。文字列の終端位置には、NULL文字(0x00)または空白文字(0x20)を設定する必要があります。また、ファイル名に空白またはコンマ(,)を含む場合は、ファイル名を二重引用符で囲む必要があります。

## 第二パラメタ(message)

索引ファイル簡易復旧関数の実行結果を示すメッセージを格納する領域のアドレスを指定します。格納する領域は、呼出し元で確保(512バイト必要)してください。

なお、メッセージが必要なければ、0を指定します。

## 復帰値

索引ファイル簡易復旧関数の結果として、メッセージに対応するコードを返却します。コードの値とメッセージの内容については、“[表7.11 索引ファイル復旧関数\(簡易復旧関数\)が返却するメッセージの内容とコード](#)”を参照してください。



## 例

```
#include "f3bifutc.h"
void callcobfrcovs(void)
{
 char ixdfilename[512] = "c:¥¥ixdfile¥0";
 char message[512];
 CFURCOVS(ixdfilename, message);
 return;
}
```

## 7.10.3 注意事項

ここでは、索引ファイルの復旧を行うために必要な作業について説明します。

### 作成時

Cプログラムから呼び出す場合には、以下のファイルをインクルードしてください。

- f3bifutc.h

### 翻訳時

Cプログラムから呼び出す場合には、“[10.3.4 プログラムの翻訳](#)”を参照してください。

### リンク時

- 翻訳オプションにDLOADを指定していないプログラムから利用する場合、F3BIFUTC.LIBを結合してください。

- ・ Cプログラムから呼び出す場合には、“[10.3.5 プログラムのリンク](#)”を参照してください。

## 実行時

以下の環境設定を行ってください。

- ・ 環境変数PATHに、以下のファイルが格納されているフォルダ名を設定してください。

f3bifutc.dll, f3bifuty.dll, f3bifrm.dll

- ・ 翻訳オプションにDLOADを指定したプログラムから利用する場合、以下のようなエントリ情報が必要になります。エントリ情報の指定方法については、“[5.4.2 副プログラムのエントリ情報](#)”を参照してください。

```
[ENTRY]
CFURCOV=F3BIFUTC.DLL
CFURCOVS=F3BIFUTC.DLL
```

## 7.10.4 COBOLから呼び出す場合の使用例

ここで示す例は、オープンの状態により復旧を行うプログラムの例です。



### 例

例1) 索引ファイル復旧関数に対して、復旧できなかったデータを格納するファイルおよびメッセージについて必要でない場合の呼出しを行っています。

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. SAGYOU1.
000030 ENVIRONMENT DIVISION.
000040 CONFIGURATION SECTION.
000050 SPECIAL-NAMES.
000060 ENVIRONMENT-NAME ENV-NAME
000070 ENVIRONMENT-VALUE ENV-VALUE.
000080 INPUT-OUTPUT SECTION.
000090 FILE-CONTROL.
000100 SELECT IXDFILE ASSIGN TO FILE1
000110 ORGANIZATION IS INDEXED
000120 RECORD KEY IS IXDFILE-RECKEY
000130 FILE STATUS IS FSDATA.
000140 DATA DIVISION.
000150 FILE SECTION.
000160 FD IXDFILE.
000170 01 IXDFILE-REC.
000180 03 IXDFILE-RECKEY PIC X(8).
000190 03 IXDFILE-DATA PIC X(20).
000200 WORKING-STORAGE SECTION.
000210 77 IXDFILE-NAME PIC X(512).
000220 77 BLKFILE-NON PIC S9(9) COMP-5 VALUE 0.
000230 77 MESSAGE-NON PIC S9(9) COMP-5 VALUE 0.
000240 77 FSDATA PIC XX.
000250 77 OPEN-FSDATA PIC XX.
000260 PROCEDURE DIVISION.
000270 OPEN I-O IXDFILE.
000280 MOVE FSDATA TO OPEN-FSDATA.
000290 CLOSE IXDFILE.
000300 EVALUATE OPEN-FSDATA
000310 WHEN "00"
000320 GO TO GYOUUMU-KAISHI
000330 WHEN "39"
000335 WHEN "90"
000340* ファイル名を取り出す
000350 MOVE ALL SPACE TO IXDFILE-NAME
000360 DISPLAY "FILE1" UPON ENV-NAME
```

```

000370 ACCEPT IXDFILE-NAME FROM ENV-VALUE
000380* 索引ファイル復旧関数を呼び出す
000390 CALL "CFURCOV" USING BY REFERENCE IXDFILE-NAME
000400 BY VALUE BLKFILE-NON
000410 BY VALUE MESSAGE-NON
000420 EVALUATE TRUE
000430 WHEN PROGRAM-STATUS <= 1
000440 GO TO GYUMU-KAISHI
000450 WHEN OTHER
000460 GO TO GYUMU-TEISHI
000470 END-EVALUATE
000490 WHEN OTHER
000500 GO TO GYUMU-TEISHI
000510 END-EVALUATE.
000520 GYUMU-KAISHI.
000530 OPEN I-O IXDFILE.
000540 :
000580 CLOSE IXDFILE.
000590 GYUMU-TEISHI.
000600 EXIT PROGRAM.

```



## 例

例2) 索引ファイル復旧関数に対して、復旧できなかったデータを格納するファイルおよびメッセージについて必要である場合の呼出しを行っています。

```

000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. SAGYOU2.
000030 ENVIRONMENT DIVISION.
000040 CONFIGURATION SECTION.
000050 SPECIAL-NAMES.
000060 ENVIRONMENT-NAME ENV-NAME
000070 ENVIRONMENT-VALUE ENV-VALUE.
000080 INPUT-OUTPUT SECTION.
000090 FILE-CONTROL.
000100 SELECT IXDFILE ASSIGN TO FILE1 ORGANIZATION IS INDEXED
000110 RECORD KEY IS IXDFILE-RECKEY FILE STATUS IS FSDATA.
000120 DATA DIVISION.
000130 FILE SECTION.
000140 FD IXDFILE.
000150 01 IXDFILE-REC.
000160 03 IXDFILE-RECKEY PIC X(8).
000170 03 IXDFILE-DATA PIC X(20).
000180 WORKING-STORAGE SECTION.
000190 77 IXDFILE-NAME PIC X(512).
000200 77 BLKFILE-NAME PIC X(11) VALUE "C:¥BLKNDAT ".
000210 77 MESSAGE-AREA PIC X(512).
000220 77 FSDATA PIC XX.
000230 77 OPEN-FSDATA PIC XX.
000240 PROCEDURE DIVISION.
000250 OPEN I-O IXDFILE.
000260 MOVE FSDATA TO OPEN-FSDATA.
000270 CLOSE IXDFILE.
000280 EVALUATE OPEN-FSDATA
000290 WHEN "00"
000300 GO TO GYUMU-KAISHI
000310 WHEN "39"
000315 WHEN "90"
000320* ファイル名を取り出す
000330 MOVE ALL SPACE TO IXDFILE-NAME
000340 DISPLAY "FILE1" UPON ENV-NAME

```

```

000350 ACCEPT IXDFILE-NAME FROM ENV-VALUE
000360* 索引ファイル復旧関数を呼び出す
000370 CALL "CFURCOV" USING BY REFERENCE IXDFILE-NAME
000380 BY REFERENCE BLKFILE-NAME
000390 BY REFERENCE MESSAGE-AREA
000400 EVALUATE TRUE
000410 WHEN PROGRAM-STATUS <= 1
000420 GO TO GYUUMU-KAISHI
000430 WHEN OTHER
000440 GO TO GYUUMU-TEISHI
000450 END-EVALUATE
000470 WHEN OTHER
000480 GO TO GYUUMU-TEISHI
000490 END-EVALUATE.
000500 GYUUMU-KAISHI.
000510 OPEN I-O IXDFILE.
000520 :
000560 CLOSE IXDFILE.
000570 GYUUMU-TEISHI.
000580 EXIT PROGRAM.

```

## 7.10.5 メッセージの内容とコード

“表7.11 索引ファイル復旧関数(簡易復旧関数)が返却するメッセージの内容とコード”に、索引ファイルの復旧で出力されるメッセージの内容とコードについて示します。

表7.11 索引ファイル復旧関数(簡易復旧関数)が返却するメッセージの内容とコード

| コード10進 | メッセージの内容                               |
|--------|----------------------------------------|
| 0      | レコードをn件復旧しました。(注1)                     |
| 1      | レコードをn件復旧しました。復旧できないレコードがn件存在しました。(注2) |
| 10     | 復旧すべきファイルが存在しません。                      |
| 11     | 復旧すべきファイルが索引ファイルではありません。               |
| 12     | 復旧すべきファイルへのアクセス権がありません。                |
| 13     | 復旧不可データ出力ファイルが既に存在します。(注2)             |
| 14     | 復旧すべきファイルを他プロセスがアクセスしています。             |
| 15     | 復旧すべきファイルを他プロセスがリカバリしています。             |
| 16     | ファイルの書き込み領域が不足しました。(注2)                |
| 128    | メモリ領域が不足しました。                          |
| 131    | 指定された索引ファイル中にレコードが存在しませんでした。(注2)       |
| 132    | ファイル情報に矛盾がありました。                       |
| 136    | 復旧すべきファイル名に誤りがあります。                    |
| 137    | 復旧不可データ出力ファイル名に誤りがあります。(注2)            |

注1：簡易復旧関数ではメッセージだけが返却されません。

注2：簡易復旧関数では返却されません。

## 7.11 COBOLファイルアクセスルーチン

COBOLファイルアクセスルーチンは、C言語からCOBOLの各編成のファイルをアクセスするためのAPI(Application Program Interface)関数群です。COBOLファイルをアクセスする32ビットアプリケーションソフトの開発/運用を支援します。

COBOLファイルアクセスルーチンの詳細については、“COBOL ファイルアクセスルーチン使用手引書”を参照してください。



## 第8章 印刷処理

本章では、1行単位のデータや帳票形式のデータを印刷装置に出力する方法について説明します。

### 8.1 印刷方法の種類

COBOLプログラムでデータを印刷するには、印刷ファイルまたは表示ファイルを使用します。ここでは、これらのファイルを使った印刷方法の概要、印字文字の種類、フォームオーバーレイパターン、FCBおよび帳票定義体について説明します。なお、印刷機能は、使用する印刷装置により異なる場合がありますので、各印刷装置の説明書を参照してください。



印刷ファイルで出力するデータは、表示用(USAGE IS DISPLAY)のデータ項目で定義されていなければなりません。データ中にバイナリの不正なコードが含まれる場合、正しく印字されないことがあります。

#### 8.1.1 各印刷方法の概要

印刷ファイルには、FORMAT句なし印刷ファイルとFORMAT句付き印刷ファイルがあります。

FORMAT句なし印刷ファイルは、行単位のデータを印刷したり、行単位のデータをフォームオーバーレイパターンと合成したり、FCBを使って印刷情報を設定してデータを印刷するときに使用します。

FORMAT句付き印刷ファイルは、FORMAT句なし印刷ファイルの機能に加えて、帳票定義体を使った帳票形式のデータの印刷を行うことができます。

本章では、印刷ファイルおよび表示ファイルを以下のように分類して説明します。

- [1] FORMAT句なし印刷ファイル(行単位のデータを印刷する)
- [2] FORMAT句なし印刷ファイル(フォームオーバーレイおよびFCBを使用して印刷する)
- [3] FORMAT句付き印刷ファイル
- [4] 表示ファイル

[1]～[4]の印刷方法の特徴、利点および用途を“表8.1 印刷方法の特徴・利点・用途”に、必要な関連製品を“表8.2 関連製品”に示します。

表8.1 印刷方法の特徴・利点・用途

| 使用するファイルの種類 |                           | [1] | [2] | [3] | [4]   |
|-------------|---------------------------|-----|-----|-----|-------|
| 特徴          | 行単位のデータの印刷ができる            | ○   | ○   | ○   | ×     |
|             | フォームオーバーレイパターンと合成した印刷ができる | ×   | ○   | ○   | ○ (注) |
|             | 帳票定義体を使った帳票印刷ができる         | ×   | ×   | ○   | ○     |
| 利点          | プログラムの記述が簡単               | ◎   | ○   | ○   | ◎     |
|             | 帳票形式の印刷が簡単                | ○   | ◎   | ◎   | ◎     |
|             | 他システムで作成した既存の帳票定義体を使用できる  | ×   | ×   | ◎   | ◎     |
|             | プログラム中で各種印刷情報を指示することができる  | ×   | ◎   | ◎   | ○     |
| 用途          | 帳票を印刷する                   | ○   | ◎   | ◎   | ◎     |

◎ : 使用可能であり適している

○ : 使用可能である

× : 使用不可能

注 : 帳票定義体にオーバーレイパターン名が指定されている場合またはプリンタ情報ファイルにオーバーレイパターン名を指定した場合だけ印刷可能です。詳細は、“MeFtのオンラインマニュアル”を参照してください。

表8.2 関連製品

| 使用するファイルの種類                                                                                           | [1] | [2]   | [3] | [4] |
|-------------------------------------------------------------------------------------------------------|-----|-------|-----|-----|
| FORM                                                                                                  | —   | ○(*1) | ○   | ○   |
| FORM オーバレイオプション                                                                                       | —   | ○     | ○   | ○   |
| PowerFORM (*2)                                                                                        | —   | ○     | ○   | ○   |
| MeFt                                                                                                  | —   | —     | ○   | ○   |
| MeFt/NET-SV (MeFt/NET連携機能)                                                                            | —   | —     | —   | ○   |
| MeFt/NET-SV (Web 連携機能)                                                                                | —   | —     | ○   | ○   |
| MeFt/Web (*3)                                                                                         | —   | —     | ○   | ○   |
| PrintWalker/OVLオプション (*4)                                                                             | —   | ○     | ○   | ○   |
| Systemwalker/ListWORKS または<br>Interstage List Works (*5)                                              | ○   | ○     | ○   | ○   |
| Systemwalker/e-DocGenerator または<br>Systemwalker/ListCREATOR EE または<br>Interstage List Creator EE (*6) | —   | —     | ○   | ○   |

\*1 : オーバレイを作成するために使用します。

\*2 : FORMに同梱されています。

\*3 : NetCOBOL Enterprise Edition サーバ運用パッケージ、NetCOBOL Professional Edition開発パッケージ、NetCOBOL Standard Edition開発パッケージまたはNetCOBOL Standard Editionサーバ運用パッケージに同梱されています。

\*4 : VSPプリンタのオプション製品です。PrintWalker/OVLオプションを使用することによりフォームオーバレイの高速印刷が可能となります。

\*5 : 帳票を電子化したり、電子化された帳票に対するさまざまな操作・管理を行ったりする場合に必要となります。この章では、特に断りがない限り、Systemwalker/ListWORKSおよび Interstage List WorksをListWORKSと記述します。

\*6 : 帳票をPDFファイルとして出力する場合に必要となります。この章では、特に断りがない限り、Systemwalker/e-DocGenerator、Systemwalker/ListCREATOR EEおよびInterstage List Creator EEをe-DocGeneratorと記述します。



## 参照

フォームオーバレイの高速印刷の詳細は、以下を参照してください。

- “5.4.1.38 @CBR\_OverlayPrintSPEC (フォームオーバレイ印刷方法の指定)”
- “8.1.9 印刷情報ファイル”
- “PrintWalker/OVLオプションのオンラインマニュアル”

また、電子化された帳票の詳細は、以下を参照してください。

- “8.6 電子帳票出力機能を使う方法”
- “ListWORKSのオンラインマニュアル”

以下に、各印刷方法の概要を説明します。

### [1] FORMAT句なし印刷ファイル(行単位のデータを印刷する)

FORMAT句なし印刷ファイルでは、行単位のデータを印刷装置に出力することができます。このとき、論理ページの大きさを指定したり、行送りやページ替えを指定したりすることもできます。

行単位のデータを印刷するときの印刷ファイルの使い方については、“8.2 行単位のデータを印刷する方法”を参照してください。

## [2] FORMAT句なし印刷ファイル(フォームオーバーレイおよびFCBを使用して印刷する)

FORMAT句なし印刷ファイルでは、1ページ分の出力データをフォームオーバーレイパターンと合成したり、FCBを使って印刷情報を指示したりすることができます。1ページ分の出力データをフォームオーバーレイパターンと合成することを指示するには、制御レコードを使います。FCBを使って印刷情報を指示するには、FCB制御文を実行用の初期化ファイルに記述します。

フォームオーバーレイパターンについては“[8.1.3 フォームオーバーレイパターン](#)”を、FCBについては“[8.1.4 FCB](#)”を、使い方については“[8.3 フォームオーバーレイおよびFCBを使う方法](#)”を参照してください。

## [3] FORMAT句付き印刷ファイル

FORMAT句付き印刷ファイルとは、プログラムのファイル定義でFORMAT句を指定した印刷ファイルのことです。FORMAT句付き印刷ファイルでは、パーティション形式の帳票定義体を使って、帳票形式のデータを印刷することができます。また、前述したフォームオーバーレイパターンおよびFCBを使った帳票印刷を行うこともできます。ただし、FORMAT句付き印刷ファイルによる帳票印刷では、MeFtが必要となります。

帳票定義体については“[8.1.6 帳票定義体](#)”を、帳票定義体を使った印刷ファイルについては“[8.4 帳票定義体を使う印刷ファイルの使い方](#)”を参照してください。

## [4] 表示ファイル

表示ファイルでは、帳票定義体に定義した帳票形式のデータを印刷することができます。前述したFORMAT句付き印刷ファイルとの相違点は、パーティション形式以外の帳票定義体も使用できることです。ただし、行レコードの印刷やフォームオーバーレイパターンおよびFCBをプログラムから変更することはできません。

帳票の出力先には、印刷装置やディスプレイ装置を指定することができます。本章では、印刷装置に出力する方法について説明します。ディスプレイ装置への出力方法については、“[9.2 表示ファイル\(画面入出力\)の使い方](#)”を参照してください。帳票印刷を行う表示ファイルの使い方については、“[8.5 表示ファイル\(帳票印刷\)の使い方](#)”を参照してください。

表示ファイルによる帳票印刷では、MeFtが必要になります。

# 8.1.2 印字文字

印字文字の印字属性(大きさ、書体、スタイル、形態、方向および間隔)を、データ記述項のCHARACTER TYPE句で指定します。CHARACTER TYPE句には、MODE-n、呼び名および印字モード名が指定できます。それぞれの書き方から指定可能な印字属性を以下に示します。

なお、日本語項目を印字する場合には、CHARACTER TYPE句の記述が必須です。

| 指定方法                  | 印字文字の属性 |    |        |        |    |        |
|-----------------------|---------|----|--------|--------|----|--------|
|                       | 大きさ     | 書体 | スタイル   | 形態     | 方向 | 間隔     |
| CHARACTER TYPE MODE-n | ○       | —  | —      | ○ (*2) | —  | ○ (*3) |
| CHARACTER TYPE 呼び名    | ○       | ○  | —      | ○      | ○  | ○ (*3) |
| CHARACTER TYPE 印字モード名 | ○       | ○  | ○ (*1) | ○      | ○  | ○      |

\*1：PRINTING MODE句のFONT指定にFONT-nnnを指定しなければなりません。

\*2：MODE-nの後にBY 呼び名 を指定しなければなりません。

\*3：印字文字の大きさと形態から決定されます。

- CHARACTER TYPE句にMODE-1、MODE-2、MODE-3を指定した場合、それぞれ印字文字の大きさを12ポ、9ポ、7ポとすることができます。
- CHARACTER TYPE句に呼び名を指定した場合、特殊名段落の機能名句でその呼び名と関連付けられた機能名の示す印字属性で印字することができます。機能名については、“COBOL文法書”の“CHARACTER TYPE句”を参照してください。
- CHARACTER TYPE句に印字モード名を指定した場合、特殊名段落のPRINTING MODE句で印字モード名に関連付けて印字属性を定義します。定義された印字属性で印字することができます。PRINTING MODE句の書き方については、“COBOL文法書”の“PRINTING MODE句”を参照してください。

指定できる印字属性について、以下に説明します。

## 印字文字の大きさ

3.0～300.0ポイントの文字サイズを指定できます。

## 指定方法

文字サイズの指定方法を以下に示します。ただし、ラスタフォントタイプ(固定サイズ)のデバイスフォントを選択した場合、各プリンタ装置に搭載されているフォントのポイント数(文字サイズ)で印字されます。  
指定方法の詳細は、“COBOL文法書”を参照してください。

| 指定方法                          | 文字サイズ                                                                                                                                                                                                                                                                                                                 |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MODE-1/ MODE-2/ MODE-3        | 12ポ/ 9ポ/ 7ポ                                                                                                                                                                                                                                                                                                           |
| 呼び名と関連付ける機能名で指定               | 12ポ/ 9ポ/ 7ポ                                                                                                                                                                                                                                                                                                           |
| 印字モード名(PRINTING MODE句のSIZE指定) | <p>3.0～300.0ポを0.1ポイント単位で指定できます。</p> <p>文字サイズの指定を省略した場合、文字間隔の指定に合わせた文字サイズで印字されます。文字サイズと文字間隔の両方を省略した場合、以下の文字サイズで印字します。</p> <ul style="list-style-type: none"> <li>日本語項目：12ポイントで印字します。</li> <li>英数字項目：環境変数情報5.4.1.39 @CBR_PrinterANK_Size (ANK文字サイズの指定) 指定が有効な場合(注)、指定された文字サイズとなります。指定されていない場合、7.0ポイントで印字します。</li> </ul> |

注：CHARACTER TYPE句およびPRINTING POSITION句を使用していない項目だけに有効です。

## ラスタフォントタイプのデバイスフォントを選択した場合

以下に、富士通製のプリンタ装置にラスタフォントタイプのデバイスフォントを選択して印字した場合の、プログラムで指定した文字サイズと、実際に印字される文字サイズの対応を示します。

| プログラムに指定した文字サイズ | プリンタ種別        |               |       |
|-----------------|---------------|---------------|-------|
|                 | FMLBP         |               | FMMPR |
|                 | 漢字ROMカートリッジあり | 漢字ROMカートリッジなし |       |
| 3.0 ～ 8.9       | 7.0           | 10.5          | 10.5  |
| 9.0 ～ 10.4      | 9.0           |               |       |
| 10.5 ～ 11.9     | 10.5          |               |       |
| 12.0 ～ 300.0    | 12.0          |               |       |

単位：ポイント



## 注意

- FORMAT句なし印刷ファイルのフォントは、環境変数情報@PrinterFontNameで指定します。フォントの指定を省略した場合、通常“明朝”や“ゴシック”などのデバイスフォントが選択されます。[参照]“5.4.1.63 @PrinterFontName (印刷ファイルで使用するフォントの指定)”
- 以下の場合、スケーラブルフォントとなり、0.1ポイントきざみで3.0～300.0ポイントの範囲で印字することができます。
  - ー デバイスフォントがアウトラインフォントである場合
  - ー 環境変数情報に“MS 明朝”や“MS ゴシック”などのTrueTypeフォントを指定した場合
- 上記表の漢字ROMカートリッジは、7ポイント/9ポイント/12ポイントすべてのカートリッジを装着した場合を想定しています。
- 他社のプリンタ装置を使用する場合、デバイスフォントの扱いは各社プリンタ装置および各社プリンタドライバに依存します。プリンタ装置の取扱い説明書などを参照してください。

## 印字文字の書体

明朝体/明朝体半角/ゴシック体/ゴシック体半角/ゴシックDP/書体番号を指定できます。

### 指定方法

指定方法の詳細は、“COBOL文法書”を参照してください。

| 指定方法                          | 書体                                                                                                                                             |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| MODE-1/ MODE-2/ MODE-3        | 明朝体                                                                                                                                            |
| 呼び名と関連付ける機能名で指定               | 明朝体／ゴシック体<br>書体の指定を省略した場合、“明朝体”で印字します。                                                                                                         |
| 印字モード名(PRINTING MODE句のFONT指定) | 明朝体／明朝体半角／ゴシック体／ゴシック体半角／ゴシックDP／書体番号<br>書体の指定を省略した場合、以下の書体で印字します。 <ul style="list-style-type: none"><li>日本語項目:明朝体</li><li>英数字項目:ゴシック体</li></ul> |

### 印字の規則

- 書体番号の指定がある場合、環境変数情報@CBR\_PrintFontTableまたはファイル識別名に指定されたフォントテーブル内のそれぞれの書体番号に対応付けたフォントフェイス名の文字書体で印字されます。書体番号とは、PRINTING MODE句に指定された“FONT-nnn”のことを指します。  
フォントテーブル名を指定しない場合またはフォントテーブル内に書体番号に対応付けたフォントフェイス名の指定がない場合は、書体番号の値にかかわらず、すべて明朝体(通常の書体)で印字されます。  
フォントテーブルの詳細および指定方法については以下を参照してください。

- “5.4.1.40 @CBR\_PrintFontTable (印刷ファイルで使用するフォントテーブルの指定)”
- “5.4.1.71 ファイル識別名 (プログラムで使用するプリンタ情報ファイルおよび各種パラメタの指定)”
- “5.4.1.72 ファイル識別名 (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)”
- “8.1.14 フォントテーブル”

- FORMAT句なし印刷ファイルでは、環境変数情報@PrinterFontNameにフォントフェイス名が指定されている場合、指定されているフォントフェイス名の文字書体で印字されます。フォントフェイス名の指定がない場合、以下に示す順番で検索します。検索の結果、どのフォントもシステムにインストールされていない場合には実行時エラーとなります。[参照]“5.4.1.63 @PrinterFontName (印刷ファイルで使用するフォントの指定)”

#### <明朝体/明朝体半角の場合>

1. 明朝
2. MS 明朝

#### <ゴシック体/ゴシック体半角/ゴシックDPの場合>

1. ゴシック(ゴシック体をサポートするプリンタにだけ有効)
2. MS ゴシック

- FORMAT句付き印刷ファイルおよび表示ファイルの場合に選択されるフォントについては、“MeFtのオンラインマニュアル”を参照してください。



### 注意

“MS P明朝”や“MS Pゴシック”など、プロポーションアルフォントを選択した場合でも、文字間隔はプログラムで指定した固定ピッチとなります。この場合、横幅の狭い文字は、左端に寄せられます。

## 印字文字のスタイル

標準/太字/斜体/太字・斜体を指定できます。文字スタイルは、書体番号の指定がある文字書体に対して指定できます。

### 指定方法

文字スタイルの指定方法については、“[8.1.14 フォントテーブル](#)”を参照してください。

文字スタイルの指定を省略した場合、“標準”が指定されたものと解釈します。

### 注意

- ・ 印字文字の書体にアウトラインフォント以外の文字書体が指定または選択された場合、スタイルの指定にかかわらず、文字書体の持つスタイルで印字されます。
- ・ 文字スタイルはフォントテーブル内に指定するため、印字文字の書体には書体番号を指定しなければなりません。印字文字の書体に書体名を指定した場合、印字スタイルはすべて標準になります。

## 印字文字の形態

全角/全角長体/全角平体/全角倍角/半角/半角長体/半角平体/半角倍角を指定できます。

### 指定方法

指定方法の詳細は、“[COBOL文法書](#)”を参照してください。

| 指定方法                          | 形態                                                                |
|-------------------------------|-------------------------------------------------------------------|
| MODE-nの呼び名に関連付ける機能名で指定        | 全角長体／全角平体／全角倍角／半角／半角倍角<br>文字形態の指定を省略した場合、“全角”で印字します。              |
| 呼び名と関連付ける機能名で指定               | 全角長体／全角平体／全角倍角／半角<br>文字形態の指定を省略した場合、“全角”で印字します。                   |
| 印字モード名(PRINTING MODE句のFORM指定) | 全角長体／全角平体／全角倍角／全角／半角長体／半角平体／半角倍角／半角<br>文字形態の指定を省略した場合、“全角”で印字します。 |

### 注意

ラスタフォントタイプのデバイスフォントを選択した場合、指定にかかわらず、すべて全角で印字されます。  
また、他社のプリンタ装置を使用した場合、そのプリンタ装置やプリンタドライバの持つ機能に依存します。

## 印字文字の方向

縦書き/横書きを指定できます。

### 指定方法

指定方法の詳細は、“[COBOL文法書](#)”を参照してください。

| 指定方法                           | 方向                                     |
|--------------------------------|----------------------------------------|
| MODE-1/ MODE-2/ MODE-3         | 横書き                                    |
| 呼び名と関連付ける機能名で指定                | 縦書き／横書き<br>文字方向の指定を省略した場合、“横書き”で印字します。 |
| 印字モード名(PRINTING MODE句のANGLE指定) | 縦書き／横書き<br>文字方向の指定を省略した場合、“横書き”で印字します。 |



## 注意

縦書きは、日本語項目に対してだけ有効となります。

## 印字文字の間隔

0.01～24.00cpiの文字間隔を指定できます。

### 指定方法

指定方法の詳細は、“COBOL文法書”を参照してください。

| 指定方法                            | 文字間隔                                                                                                                                                                                                           |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MODE-1/ MODE-2/ MODE-3          | 印字文字の大きさや形態によって決定されます。(注)                                                                                                                                                                                      |
| 呼び名と関連付ける機能名で指定                 |                                                                                                                                                                                                                |
| 印字モード名 (PRINTING MODE句のPITCH指定) | 0.01～24.00cpiの文字間隔を0.01cpi単位で指定します。<br>文字間隔の指定を省略した場合、文字サイズの指定に合わせた文字間隔で印字されます。文字間隔と文字サイズの両方を省略した場合、以下の文字間隔で印字します。 <ul style="list-style-type: none"> <li>日本語項目 : 6.00cpi</li> <li>英数字項目 : 10.00cpi</li> </ul> |

注 : 印字文字の大きさや形態によって決定される間隔を“表8.3 印字文字の大きさ/形態と文字間隔の関係”に示します。

表8.3 印字文字の大きさ/形態と文字間隔の関係

| 文字の大きさ       | 文字の形態 (単位 : cpi) |      |      |      |      |      |      |      |
|--------------|------------------|------|------|------|------|------|------|------|
|              | 全角               | 半角   | 長体   | 半長体  | 平体   | 半平体  | 倍角   | 半倍角  |
| MODE-1 (12ボ) | 5                | 10   | 5    | —    | 2.5  | —    | 2.5  | 5    |
| MODE-2 ( 9ボ) | 8                | 16   | 8    | —    | 4    | —    | 4    | 8    |
| MODE-3 ( 7ボ) | 10               | —    | 10   | —    | 5    | —    | 5    | —    |
| A ( 9ボ)      | 5                | 10   | 5    | 10   | 2.5  | 5    | 2.5  | 5    |
| B ( 9ボ)      | 20/3             | 40/3 | 20/3 | 40/3 | 10/3 | 20/3 | 10/3 | 20/3 |
| X-12P (12ボ)  | 5                | 10   | 5    | 10   | 2.5  | 5    | 2.5  | 5    |
| X-9P ( 9ボ)   | 8                | 16   | 8    | 16   | 4    | 8    | 4    | 8    |
| X-7P ( 7ボ)   | 10               | 20   | 10   | 20   | 5    | 10   | 5    | 10   |
| C ( 9ボ)      | 7.5              | 15   | 7.5  | 15   | 3.75 | 7.5  | 3.75 | 7.5  |
| D-12P (12ボ)  | 6                | 12   | 6    | 12   | 3    | 6    | 3    | 6    |
| D-9P ( 9ボ)   | 6                | 12   | 6    | 12   | 3    | 6    | 3    | 6    |

## 8.1.3 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、あらかじめ罫線や見出し文字など帳票の固定部分を設定するときに使用します。1ページ分の出力データとフォームオーバーレイパターンを合成して印字することにより、帳票印刷を簡単に行うことができます。

フォームオーバーレイパターンは、FORMオーバーレイオプションまたはPowerFORMを使って画面イメージで簡単に作成することができます。また、1つのフォームオーバーレイパターンを複数のプログラムで使用したり、他システムで作成したものを使用したりすることもできます。

フォームオーバーレイパターンの作成方法については、“FORM説明書”、“FORMヘルプ”および“PowerFORMヘルプ”(注)を参照してください。また、フォームオーバーレイパターンを使った帳票印刷については、“8.3 フォームオーバーレイおよびFCBを使う方法”を参照してください。



注：PowerFORMでオーバーレイを作成時に参照してください。

## 8.1.4 FCB

FCBは、1ページ分の行数、行間隔および印字開始位置を定義します。FCBを使用することにより、1ページの行数、行間隔および印字開始位置を変更することができます。

実行用の初期化ファイルでデフォルトFCB名が指定された場合(@DefaultFCB\_Name=FCBxxxx)、またはI制御レコードでFCB名が指定された場合、FCB情報を外部から指定できます。

FCB情報は、実行用の初期化ファイルに以下の形式で指定します。

FCBxxxx=FCB制御文

XXXX

環境変数情報@DefaultFCB\_Name=FCBxxxxに指定したFCB名(xxxxの部分)またはI制御レコードに指定したFCB名FCB制御文の形式を以下に示します。

[ LPI ( ( 行スペース [ , 行数 ] ) [ , ( 行スペース [ , 行数 ] ) ... ]  
[ , CHm ( 行番号 [ , 行番号 ] ... ) [ , CHm ( 行番号 [ , 行番号 ] ... ) ... ]  
[ , { SIZE ( { 用紙の長さ  
110 } )  
FORM ( { A3  
A4  
B4  
A5  
B5  
LT } [ , { PORT  
LAND } ] ) } ]

### LPI情報

用紙の先頭から順に、(行スペース,行数)の形式で指定できます。行スペースには、LPIの単位で、6、8、12のどれかを指定します。

### CH情報

CHANNEL-01～12までに対して、行番号を指定できます。CHANNEL-01～12は、それぞれチャネル1からチャネル12に割り当てた行番号にスキップすることを意味します。通常、チャネル1(CHANNEL-01)は、印字可能な最初の行を識別するために使用されます。

### SIZE情報

SIZEオペランドでは、帳票の縦方向のサイズを1/10インチ単位で指定します。指定可能な値は、35～159(3.5インチ～15.9インチ)です。省略した場合、110(11インチ)が有効になります。

### FORM情報

FORMオペランドでは、帳票を定型サイズで指定します。定型サイズでは、PORT(ポートレート:向きが縦)とLAND(ランドスケープ:向きが横)の用紙の向きごとに、縦方向のサイズが一意に決定します。

デフォルトFCB名の指定方法については、“5.4.1.54 @DefaultFCB\_Name(デフォルトFCB名の指定)”を参照してください。

I制御レコードでのFCB名の指定方法については“8.3 フォームオーバーレイおよびFCBを使う方法”を参照してください。

デフォルトFCBの指定およびI制御レコードのFCBの指定が省略された場合、COBOLランタイムシステムは以下のデフォルト情報をもとに動作します。

LPI ((6, 66)), CH1 (4), SIZE (110)



ー 6LPI×11インチ(66行)

| チャンネル番号 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|
| 行位置     | 4  | 10 | 16 | 22 | 28 | 34 | 40 | 46 | 66 | 52 | 58 | 64 |

このデフォルト情報と利用者が使用される物理用紙が合致していない場合、デフォルトFCBまたはI制御レコードを使用して物理用紙に合致したFCBを指定しなければなりません。

## 注意

FCBの指定は、プリンタから供給される用紙のサイズや向きを決定するものではありません。用紙サイズおよび印刷形式の指定は、I制御レコードを使用して決定します。詳細は、“[8.1.5 I制御レコード/S制御レコード](#)”を参照してください。

## 参考

### [重要] FCBの必要性

FCBは、物理ページの用紙サイズに合わせてCOBOLランタイムシステムがページ制御および行制御を行うために必ず必要な情報です。

ページ制御および行制御とは、COBOLプログラムの印刷要求(WRITE文のADVANCING句の指定)に応じて出力する印刷データをページ内のどの位置に配置させるのか、改ページが必要かどうかをCOBOLランタイムシステムが自動的に制御する処理を意味しています。

例えば、利用者が実際に次のような帳票印刷を行うものと仮定した場合、

- ・ 物理用紙サイズ:A4
- ・ 印刷形式:縦向き(ポートレートモード)
- ・ ページ内の先頭行位置:1行目
- ・ 行間隔:6LPI(1インチに6行印刷できる間隔)

FCB制御文の指定は以下に行います。

```
LPI ((6)), CH1 (1), FORM (A4, PORT)
```

この場合、COBOLランタイムシステムは、このFCB制御文を解析しCOBOLのWRITE～ADVANCING指定の要求にしたがって次のような制御を行います。

- ・ WRITE文でADVANCING PAGE(改ページ)が要求された場合、1枚ページを送ってCH1オペランドで指定されたページの先頭行である1行目に印刷行を位置付けます。
- ・ WRITE文でADVANCING n LINESが要求された場合、LPIオペランドで指定された行間隔である6LPI単位で行間を計算し指定された行数分だけ行送りを行います。
- ・ また、COBOLランタイムシステムは、FORMオペランドの情報からA4用紙を縦向き(ポートレートモード)に使用した場合のページ内に印刷可能な行数を計算します。このため、COBOLのWRITE文で明にADVANCING PAGE(改ページ要求)が行われない場合でも、ページ内の印刷可能な行数を超える印刷要求に対して自動的に改ページする制御を行います。

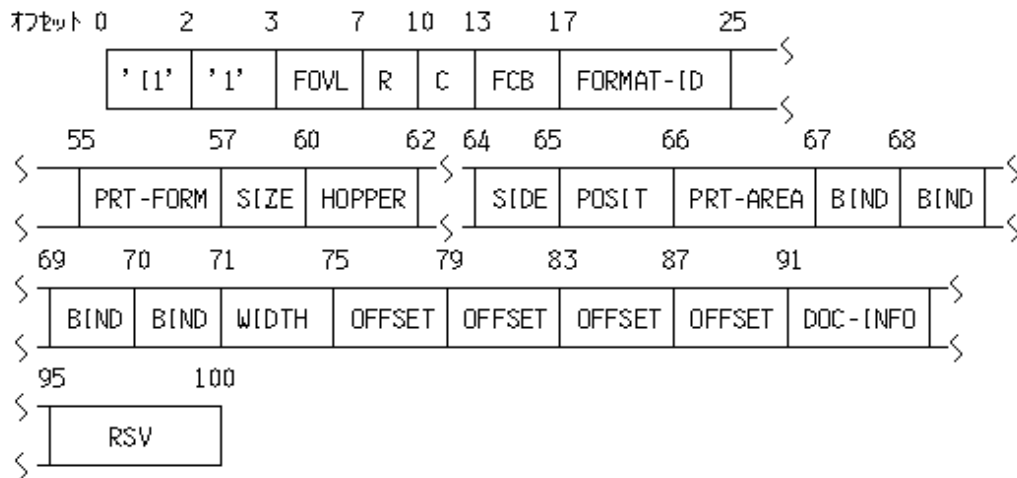
このように、COBOLプログラムで帳票印刷を行う場合、実際に使用する物理用紙サイズに対応したFCBの指定が必ず必要です。物理用紙サイズとFCBの指定が合致していない場合、意図した設計どおりの印刷結果が得られないことがありますので十分注意してください。

## 8.1.5 I制御レコード/S制御レコード

レコード内の各領域への設定値の組合せ条件や各フィールドの詳細な説明および本書で説明していない領域については、“COBOL文法書”を参照してください。なお、有効となる機能が使用する印刷装置により異なる場合がありますので、各印刷装置の説明書を参照してください。

## I制御レコード

I制御レコードの形式を以下に示します。



|    |             |                          |             |
|----|-------------|--------------------------|-------------|
| 01 | I 制御レコード.   |                          |             |
| 03 | 識別子         | PIC X(2) VALUE "11".     |             |
| 03 | 形式          | PIC X(1) VALUE "1".      |             |
| 03 | オーバーレイ名     | PIC X(4).                | → FOVL      |
| 03 | オーバーレイ焼付け回数 | PIC 9(3).                | → R         |
| 03 | 複写数         | PIC 9(3).                | → C         |
| 03 | FCB名        | PIC X(4).                | → FCB       |
| 03 | 画面帳票定義体名    | PIC X(8).                | → FORMAT-ID |
| 03 |             | PIC X(30).               |             |
| 03 | 印刷形式        | PIC X(2).                | → PRT-FORM  |
| 03 | 用紙サイズ       | PIC X(3).                | → SIZE      |
| 03 | 用紙供給口       | PIC X(2).                | → HOPPER    |
| 03 |             | PIC X(2).                |             |
| 03 | 印刷面         | PIC X.                   | → SIDE      |
| 03 | 印刷面位置付け     | PIC X.                   | → POSIT     |
| 03 | 印字禁止域       | PIC X.                   | → PRT-AREA  |
| 03 | とじしろ方向.     |                          | → BIND      |
| 04 |             | PIC X OCCURS 4 TIMES.    |             |
| 03 | 印刷位置情報.     |                          |             |
| 04 | とじしろ幅       | PIC 9(4).                | → WIDTH     |
| 04 | 印刷原点位置.     |                          |             |
| 05 |             | PIC 9(4) OCCURS 4 TIMES. | → OFFSET    |
| 03 | 文書名識別情報     | PIC X(4).                | → DOC-INFO  |
| 03 |             | PIC X(5) VALUE SPACE.    | → RSV       |

以降の説明中で“ファイルオープン時の指定”とある場合、そのファイルがオープンされたときのシステムが持つ印刷環境のことを指します。

FORMAT句付き印刷ファイルの場合、帳票定義体の指定値が有効になります。帳票定義体の指定がない場合は、プリンタ情報ファイルに指定された情報が有効になります。プリンタ情報ファイルの詳細は、“MeFtのオンラインマニュアル”を参照してください。

### FOVL (フォームオーバーレイパターン名)

使用するフォームオーバーレイパターン名を指定します。オーバーレイグループが指定された場合、先頭のオーバーレイパターンに対して単一オーバーレイの処理を行います。このフィールドが空白の場合、ファイルオープン時の指定が有効となります。

### R (フォームオーバーレイパターン焼付け回数)

フォームオーバーレイパターンの焼付け回数(0~255)を指定します。  
ただし、本システムでは複写数と同じ値が指定されたものと扱われます。

## C (複写数)

ページ単位の複写数(0～255)を指定します。0を指定するとファイルオープン時の指定が有効となります。



### 注意

両面印刷指定時は、複写数の指定は有効となりません。1が指定されたものとみなします。また、複写数の指定はプリンタドライバが複写機能を持っている場合だけ有効となります。

## FCB (FCB名)

FCB名を指定します。このフィールドが空白の場合、環境変数情報@DefaultFCB\_Nameに指定されたFCB名が有効となります。なお、環境変数情報の指定については、“[5.4.1.54 @DefaultFCB\\_Name \(デフォルトFCB名の指定\)](#)”を参照してください。デフォルトFCB名が指定されていない場合、COBOLランタイムシステムのデフォルト情報が有効となります。デフォルト情報の詳細は、“[8.1.4 FCB](#)”を参照してください。FCB名は帳票定義体名と同時に指定することはできません。

## FORMAT-ID (帳票定義体名)

I制御レコードで指定する情報を適用する帳票定義体名を指定します。この指定により、固定形式ページとなります。このフィールドが空白の場合、不定形式ページとなります。FORMAT-IDはFCB名と同時に指定することはできません。



### 注意

帳票定義体名の指定は、FORMAT句付き印刷ファイルの場合だけ有効となります。使用方法については、“[8.4.2 プログラムの記述](#)”を参照してください。

## PRT-FORM (印刷形式)

印刷形式を指定します。設定可能な値を以下に示します。

- "P" (ポートレートモード)
- "L" (ランドスケープモード)
- "LP" (ラインプリンタモード)
- "PZ" (縮小印刷のポートレートモード)
- "LZ" (縮小印刷のランドスケープモード)

このフィールドが空白の場合、用紙サイズの指定(SIZE)にも空白が指定されていなければなりません。この場合、ファイルオープン時の指定が有効となります。



### 注意

- FORMAT句なし印刷ファイルの場合、“PZ”、“LZ”指定は、80%縮刷をサポートしているプリンタおよびプリンタドライバに対してだけ有効となります。80%縮刷をサポートしていないプリンタおよびプリンタドライバでは、“PZ”、“LZ”の縮小指定は無視され、“P”、“L”が指定されたものとみなします。また、“LP”指定は、連続用紙(ストックフォーム)サイズで設計された帳票を、A4用紙を横向きに使用したときのサイズに文字間および行間を縮小して印刷します。この場合、文字サイズ自身は縮小されませんので、文字の大きさや形態(平体や倍角など)によっては、隣接する文字同士が重なって印刷されることがあります。

なお、“PZ”、“LZ”、“LP”指定で、フォームオーバーレイは縮小されません。

- プリンタから供給される用紙の印刷形式(用紙の方向)は、本フィールドの指定により決定します。本フィールドの指定を省略した場合、以下の解釈になります。
  - FORMAT句なし印刷ファイル  
プリンタドライバでの設定値
  - FORMAT句付き印刷ファイル  
帳票定義体での設定値→プリンタ情報ファイルでの設定値

なお、実際に使用される印刷形式に合わせて、FCB制御文を定義・指定する必要があります。詳細は、“[8.1.4 FCB](#)”を参照してください。

## SIZE (用紙サイズ)

用紙サイズを指定します。指定可能なサイズを以下に示します。

- "A3"
- "A4"
- "A5"
- "B4"
- "B5"
- "LTR"(レター)

このフィールドが空白の場合、印刷形式(PRT-FORM)にも空白が指定されていなければなりません。この場合、ファイルオープン時の指定が有効となります。

また、FORMAT句なし印刷ファイルでは、上記用紙サイズに加えて任意の3文字以内の文字列を指定することができます。任意の3文字以内の文字列は、連帳の用紙サイズなど上記以外の用紙サイズを動的に変更する場合に、環境変数情報@PRN\_FormName\_xxxに用紙名を対応付けて指定します。なお、環境変数情報の指定方法については、“[5.4.1.64 @PRN\\_FormName\\_xxx\(用紙名の指定\)](#)”を参照してください。



### 注意

- 任意の3文字以内の文字列は、FORMAT句付き印刷ファイルには指定できません。FORMAT句付き印刷ファイルを利用して、連帳の用紙サイズなどあらゆる用紙サイズを指定する場合、本フィールドに空白を指定し、プリンタ情報ファイルで実際の用紙サイズを指定します。詳細は、“[MeFtのオンラインマニュアル](#)”を参照してください。
- プリンタから供給される用紙サイズ(用紙の種類)は、本フィールドの指定により決定します。本フィールドの指定を省略した場合、以下の解釈になります。
  - FORMAT句なし印刷ファイル  
プリンタドライバでの設定値
  - FORMAT句付き印刷ファイル  
帳票定義体での設定値→プリンタ情報ファイルでの設定値→プリンタドライバでの設定値

なお、実際に使用される用紙サイズに合わせて、FCB制御文を定義・指定する必要があります。詳細は、“[8.1.4 FCB](#)”を参照してください。

## HOPPER (用紙供給口)

用紙供給時の用紙供給口を指定します。設定可能な値を以下に示します。

- "P1"(主供給口1)
- "P2"(主供給口2)
- "S"(副供給口)
- "P"(任意の供給口)

このフィールドが空白の場合、ファイルオープン時の指定が有効となります。



### 注意

用紙供給口は用紙サイズ指定に従ってシステムが自動的に選択するため、アプリケーションからI制御レコードを利用して、用紙供給口を変更することはできません。

## SIDE (印刷面)

印刷面を指定します。設定可能な値を以下に示します。

- "F"(片面印刷)
- "B"(両面印刷)

このフィールドが空白の場合、ファイルオープン時の指定が有効となります。



### 注意

両面印刷指定時は、複写数の指定は有効となりません。1が指定されたものとみなします。

## POSIT (印刷面位置付け)

両面印刷時の印刷開始面を指定します。指定可能な値を以下に示します。

- "F"(表面位置付け)
- "B"(裏面位置付け)

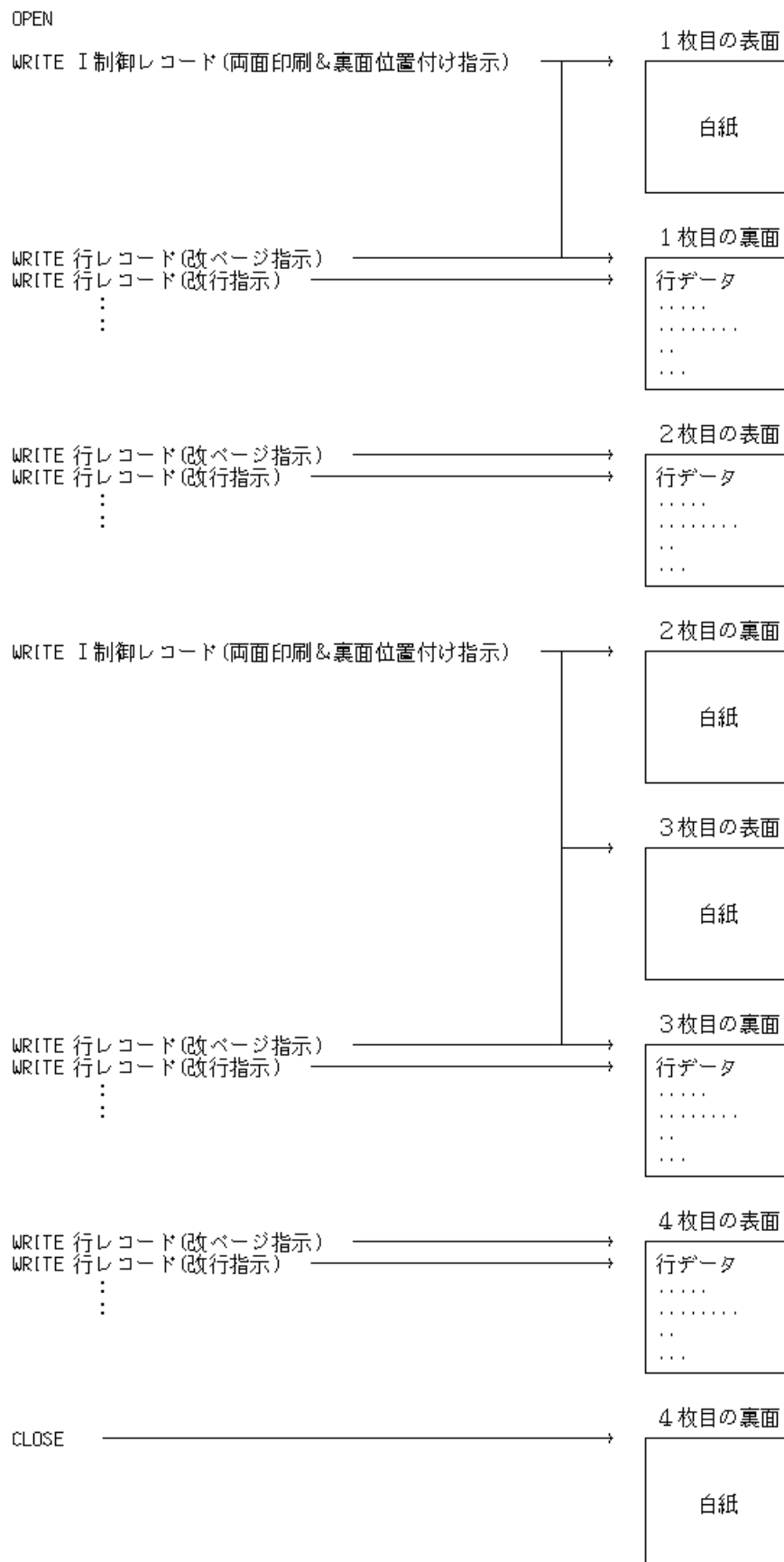
空白が指定された場合は、表面に位置付けられます。



### 例

- プリンタまたはプリンタドライバで白紙ページの扱いが指定できる場合、この指定に“白紙を印刷しない”を設定すると、印刷面位置付け(POSIT)機能は、正しく制御されないことがあります。印刷面位置付け(POSIT)機能を利用する場合は、必ず“白紙を印刷する”を設定してください。

- I制御レコードを出力すると、それまでの印刷ジョブが終了し、新しい印刷ジョブが開始されます。したがって、I制御レコード出力直前の印刷面が表面だったのか裏面だったのかに関係なく、それまでの用紙が出力され、新しい用紙への印刷が開始されます。



## PRT-AREA (印字禁止域)

印字禁止領域の設定を行う("L")か、行わない("N")かを指定します。このフィールドが空白の場合、ファイルオープン時の指定が有効になります。

ただし、帳票定義体を使用する場合は無効となります。

## BIND (とじしろ方向)

連続して出力される複数ページを製本するときのとじしろ方向を指定します。とじしろ方向は複数ページに対して意味を持つ指定であるため、このフィールドが空白の場合、直前のページでのとじしろ方向がそのまま引き継がれます。



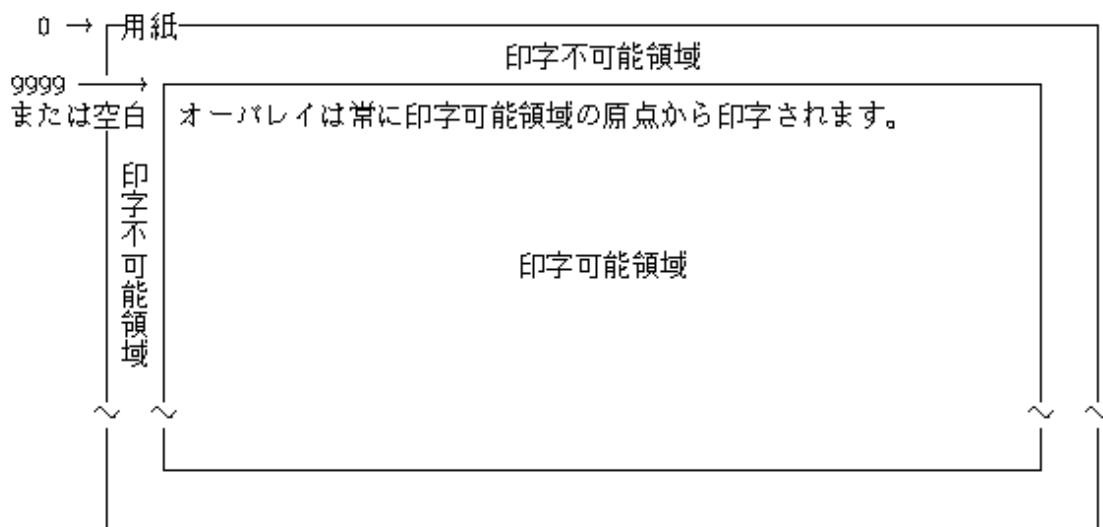
とじしろ方向の指定は、左とじ指定("L")および上とじ指定("U")だけ有効となります。したがって、右とじ指定("R")および下とじ指定("D")は無視されます。右方向および下方向にとじしろ幅を設ける場合、アプリケーションで意識する必要があります。

## WIDTH (とじしろ幅)

とじしろ幅を0～9999(単位:1/1440インチ)で指定します。

FORMAT句なし印刷ファイルの場合、本指定は、環境変数情報@CBR\_OverlayPrintOffsetの指定値により有効範囲が異なります。環境変数情報@CBR\_OverlayPrintOffsetに“VALID”が指定されている場合は、フォームオーバーレイおよび行レコードに対して有効となります。“INVALID”が指定された場合および指定が省略された場合は、行レコードに対してだけ有効となります。この場合、フォームオーバーレイパターンには有効となりませんので注意してください(行レコードとオーバーレイが正しく合成されず、ずれて印刷されることがあります)。

また、0が指定された場合の印刷開始位置は、用紙の左端角となります。この場合、プリンタの印刷不可能な領域に該当する可能性があり、印刷データが欠落するおそれがありますので注意してください。印字可能領域の左端角を原点として印字する場合は、本フィールドに9999または空白を指定してください。詳細は、“[8.1.13 印刷不可能な領域について](#)”を参照してください。



FORMAT句付き印刷ファイルの場合、このフィールドが9999または空白のとき、ファイルオープン時の指定が有効となります。

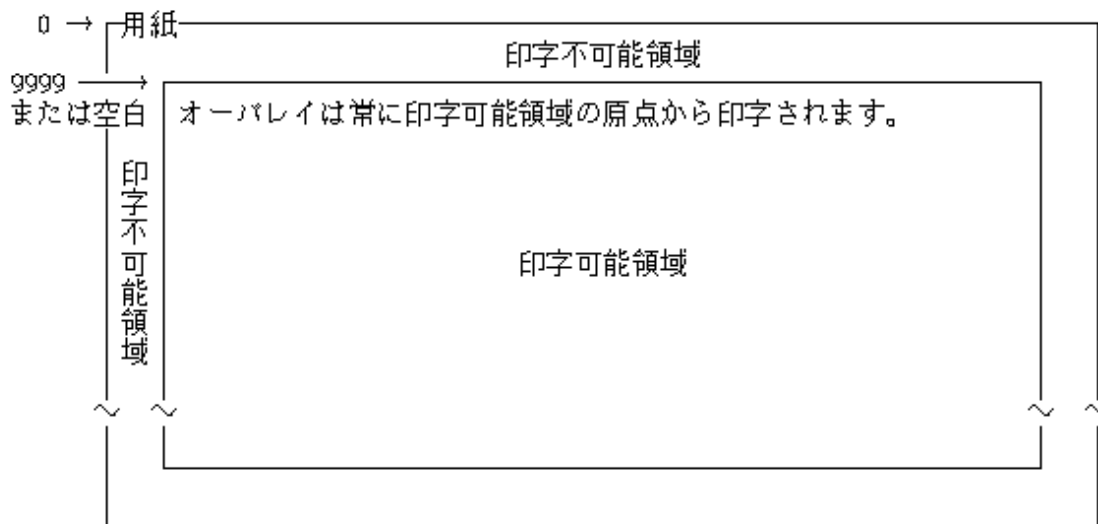
## OFFSET (印刷原点位置)

印刷原点位置を0～9999(単位:1/1440インチ)で指定します。

FORMAT句なし印刷ファイルの場合、本指定は、環境変数情報@CBR\_OverlayPrintOffsetの指定値により有効範囲が異なります。環境変数情報@CBR\_OverlayPrintOffsetに“VALID”が指定されている場合は、フォームオーバーレイおよび行レコードに対して有効となります。“INVALID”が指定された場合および指定が省略された場合は、行レコードに対してだけ有効となります。この

場合、フォームオーバーレイパターンには有効となりませんので注意してください(行レコードとオーバーレイが正しく合成されず、ずれて印刷されることがあります)。

また、0が指定された場合の印刷開始位置は、用紙の左端角となります。この場合、プリンタの印刷不可能な領域に該当する可能性があり、印刷データが欠落するおそれがありますので注意してください。印字可能領域の左端角を原点として印字する場合は、本フィールドに9999または空白を指定してください。詳細は、“[8.1.13 印刷不可能な領域について](#)”を参照してください。



FORMAT句付き印刷ファイルの場合、このフィールドが9999または空白のとき、ファイルオープン時の指定が有効となります。なお、PRT-FORM(印刷形式)と同時に指定できますが、LP(ラインプリンタモード)を指定した場合、OFFSETの指定は無効となります。また、縮小印刷形式を指定した場合のOFFSETは縮小前の長さで指定します。

#### DOC-INFO (文書名識別情報)

文書名を識別するための情報を指定します。指定可能な値は、4文字以内の任意の英数字です。ここで指定した値は、実行時に環境変数情報@CBR\_DocumentName\_xxxxのxxxx部分に置き換えて設定しておく必要があります。また、本環境変数情報名には、128バイト以内の英字/数字/カナ/日本語の組合せで構成される文書名を対応付けておかなければなりません。なお、環境変数情報の指定形式については、“[5.4.1.22 @CBR\\_DocumentName\\_xxxx \(I制御レコードによる文書名の指定\)](#)”を参照してください。



#### 注意

DOC-INFOフィールドの指定は、FORMAT句付き印刷ファイルには指定できません。FORMAT句付き印刷ファイルを利用して、プリントマネージャに任意の文書名を表示させる場合、本フィールドに空白を指定し、プリンタ情報ファイルで文書名を指定します。詳細は、“MeFtのオンラインマニュアル”を参照してください。

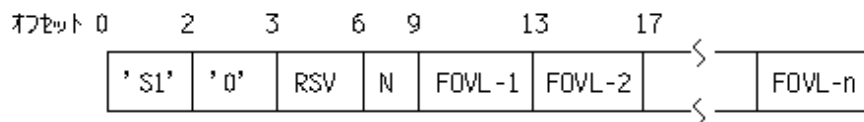
#### RSV (システム使用領域)

システムの使用する領域です。空白を設定しておきます。

#### S制御レコード

S制御レコードの形式を以下に示します。





|    |           |                       |          |
|----|-----------|-----------------------|----------|
| 01 | S 制御レコード. |                       |          |
| 03 | 識別子       | PIC X(2) VALUE "S1".  |          |
| 03 | 形式        | PIC X(1) VALUE "0".   |          |
| 03 |           | PIC X(3) VALUE SPACE. | → RSV    |
| 03 | 個数        | PIC 9(3).             | → N      |
| 03 | オーバーレイ名 1 | PIC X(4).             | → FOVL-1 |
| :  |           |                       |          |
| 03 | オーバーレイ名 n | PIC X(4).             | → FOVL-n |

#### RSV (システム使用領域)

システムの使用する領域です。空白を設定しておきます。

#### N (フォームオーバーレイパターン名の個数)

指定するフォームオーバーレイパターン名を指定します。

#### FOVL-n (フォームオーバーレイパターン名)

フォームオーバーレイパターン名を設定します。ただし、本システムでは、先頭に指定されたフォームオーバーレイパターン名だけが有効になります。

### 制御レコードの有効範囲

I制御レコードが有効となる範囲は、そのレコードが出力されてから、次のI制御レコードが出力されるまでです。ただし、I制御レコードで指定するフォームオーバーレイパターン名は、次のI制御レコードまたはS制御レコードが出力されるまで有効です。

S制御レコードが有効となる範囲は、そのレコードが出力されてから、次のS制御レコードまたはI制御レコードが出力されるまでです。



#### 注意

制御レコードの各フィールドに指定された値に誤りがあると、FILE STATUS句または誤り手続きの指定に関係なく、プログラムの実行を中断し、終了処理を行います。

## 8.1.6 帳票定義体

FORMを使って帳票を設計すると、帳票定義体を作成されます。NetCOBOLでは、帳票定義体に定義したデータ項目をプログラムに取り込み、そのデータ項目に値を設定して出力することにより、帳票を印刷することができます。また、帳票定義体には、フォームオーバーレイパターンを取り込むこともできます。

帳票定義体を使って帳票の印刷を行う場合は、MeFtが必要です。MeFtを使用する場合、MeFtが使用するプリンタ情報ファイルが必要となります。プリンタ情報ファイルの詳細は、“MeFtのオンラインマニュアル”を参照してください。

帳票定義体の作成方法については“FORM説明書”、“FORMヘルプ”または“PowerFORMヘルプ”を参照してください。

帳票定義体を使った帳票印刷については“8.4 帳票定義体を使う印刷ファイルの使い方”または“8.5 表示ファイル(帳票印刷)の使い方”を参照してください。



#### 注意

NetCOBOLで使う帳票定義体の拡張子を除いたファイル名は、英字で始まる8文字以内の英数字となるようにしてください。

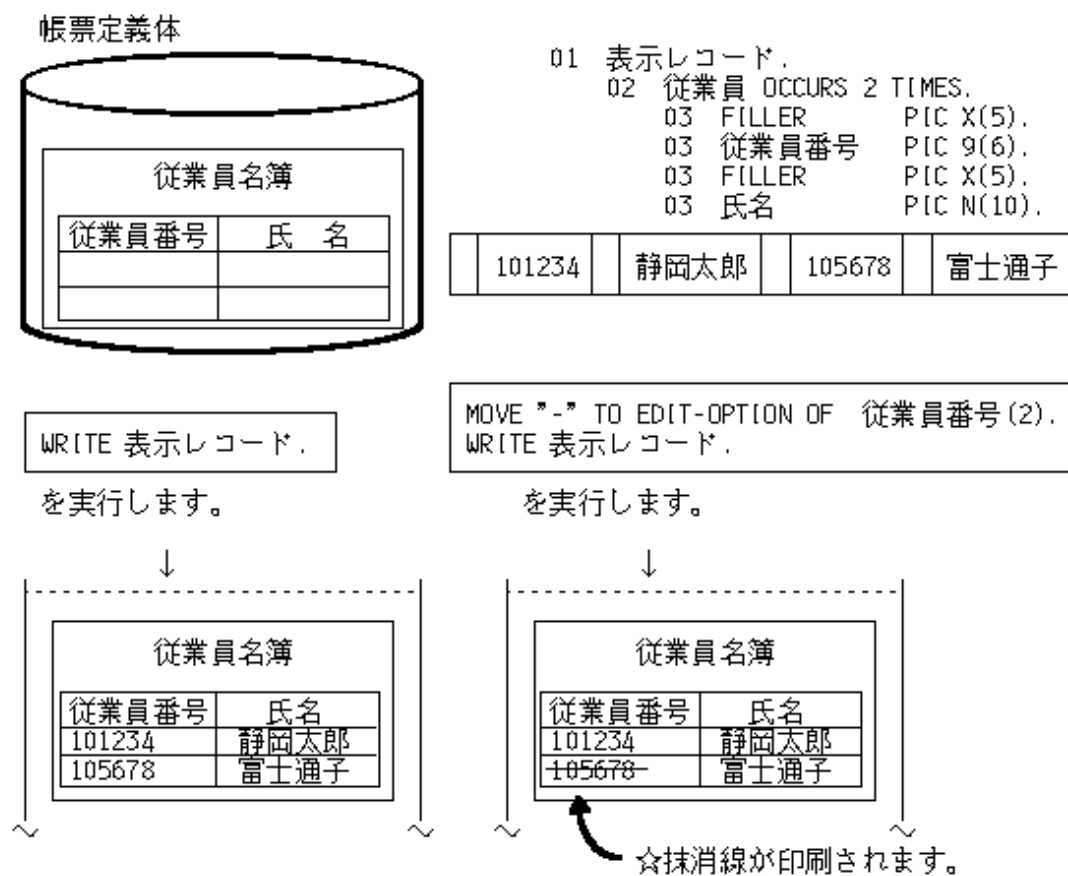
## 8.1.7 特殊レジスタ

FORMAT句付き印刷ファイルおよび表示ファイル(帳票印刷)では、帳票定義体で定義されている出力データの属性を、COBOLの特殊レジスタを使って、プログラムの実行中に変更することができます。

帳票機能の特殊レジスタには、次の種類があります。

- EDIT-MODE : 出力処理の対象にする/しないなどを指定します。
- EDIT-OPTION : 下線付き、抹消線付きなどを指定します。
- EDIT-COLOR : 色を指定します。
- EDIT-OPTION2 : 背景色を指定します。
- EDIT-OPTION3 : 網がけを指定します。

これらの特殊レジスタは、帳票定義体で定義したデータ名で修飾して使います。たとえば、データ名Aの色属性の設定は、“EDIT-COLOR OF A”のように記述します。各特殊レジスタに設定する値については、“MeFtのオンラインマニュアル”を参照してください。



### 注意

項目制御部なしを指定した帳票定義体では、特殊レジスタを使用することはできません。

また、1つのプログラム中で項目制御部なしを指定した帳票定義体と項目制御部を指定した帳票定義体を混在して使うことはできません。

EDIT-OPTION2およびEDIT-OPTION3は5バイトの項目制御部を指定した帳票定義体のみで使用することができます。ただし、EDIT-OPTION2およびEDIT-OPTION3の指示を有効にするためには、MeFt V7.2以降が必要です。このとき、プリンタ情報ファイルに

PRTITEMCTL(項目制御部拡張指定)で拡張の指定が必要です。プリンタ情報ファイルの指定の詳細は“MeFtのオンラインマニュアル”を参照してください。

---

## 8.1.8 印字文字の配置座標

配置座標は、プリンタ解像度(DPI)をアプリケーションで指定された文字間隔(cpi)または行間隔(LPI)で除算し、除算した値をもとに求めます。

FORMAT句なし印刷ファイルを使用して帳票印刷を行う場合、除算した余りを切り捨てた座標に配置する印字方法と、除算して割り切れないときに1インチ単位内で補正した座標に配置する印字方法があります。

除算して余りを切り捨てる方法では、切り捨てたドット数の分だけ文字間隔(cpi)や行間隔(LPI)がそれぞれ左方向、上方向に詰めて印字されます。出力するプリンタの解像度に依存しない印刷結果を得るために、1インチ単位内で補正した座標に配置する方法を選択することをおすすめします。

印字文字の配置座標の指定方法については、“[5.4.1.42 @CBR\\_PrintTextPosition \(文字配置座標の計算方法の指定\)](#)”を参照してください。

---

## 8.1.9 印刷情報ファイル

印刷情報ファイルとは、FORMAT句なし印刷ファイルを利用して帳票出力を行う場合に、出力する帳票の状態制御情報を設定するテキスト形式のファイルです。

印刷情報ファイルの書式および出力される帳票の状態を制御する情報は、次のとおりです。



### 注意

- ・ 印刷情報ファイルに指定する絶対パスのファイル名および絶対パス名は、二重引用符("")で囲まないでください。
- ・ 行の先頭に;(セミコロン)がある場合、セミコロンから改行までの間は、コメントとして認識されます。

## 書式および設定情報

|                                             |                                       |
|---------------------------------------------|---------------------------------------|
| [PrintInformation]                          | ⇒ セクション名（固定、かつ必須）                     |
| TextAlign= { TOP<br>BOTTOM }                | ⇒ 上端／下端合わせ指定（省略可能）                    |
| DocumentName= 文書名                           | ⇒ 文書名指定（省略可能）                         |
| OverlayPrintSPEC= { PRINTMONITOR<br>COBOL } | ⇒ オーバレイ印刷方式（省略可能）                     |
| OverlayPrintOffset= { VALID<br>INVALID }    | ⇒ オーバレイ原点移動機能を有効または無効にする指定（省略可能）      |
| STREAM= { LW<br>PR }                        | ⇒ 帳票の出力方式 [電子帳票出力／プリンタ出力]（省略可能）       |
| STREAMENV=電子帳票情報ファイル名                       | ⇒ 電子帳票情報ファイルへの絶対パス名（省略可能）             |
| PRTOUT= { LPTn:<br>COMn:<br>PRTNAME:プリンタ名 } | ⇒ 出力プリンタ名を指定（省略可能）                    |
| FOVLDIR=オーバレイファイルのフォルダ名                     | ⇒ フォームオーバレイパターンファイルを格納したフォルダ名（省略可能）   |
| FOVLTYPE=オーバレイファイルの形式                       | ⇒ フォームオーバレイパターンファイルのプレフィクス（形式）名（省略可能） |
| FOVLNAME=オーバレイファイル名                         | ⇒ フォームオーバレイパターンファイルのファイル名（省略可能）       |
| OVD_SUFFIX=オーバレイファイル(拡張子)名                  | ⇒ フォームオーバレイパターンファイルのサフィクス（拡張子）名（省略可能） |

### セクション名(必須)

セクション名 “[PrintInformation]” は、印刷情報ファイルを識別するための名前で、常に固定です。

### TextAlign

行内に配置する印字文字の位置を指定します。印字する文字を文字セルの左上端を基準点として行内に上端合わせで配置する (TOP) か、左下端を基準点として行内に下端合わせで配置する (BOTTOM) かを指定します。なお、本指定は、環境変数情報 @CBR\_TextAlign の指定よりも優先されます。[参照] “[5.4.1.47 @CBR\\_TextAlign \(文字行内配置時の上端/下端合わせの指定\)](#)”

### DocumentName

Windows システムが提供するプリントマネージャに文書名を表示したい場合に指定します。指定可能な文書名は、128 バイト以内の英字/数字/カナ/日本語の組合せでなければなりません。なお、環境変数情報 @CBR\_DocumentName\_xxxx の指定が有効な場合、本指定は無視されます。[参照] “[5.4.1.22 @CBR\\_DocumentName\\_xxxx \(I 制御レコードによる文書名の指定\)](#)”

### OverlayPrintSPEC

I/S 制御レコードを使用したフォームオーバレイの印刷を行う場合の実現方法を指定します。COBOL ランタイムシステムによるオーバレイ印刷を行う場合 “COBOL” を指定します。PrintWalker/OVL オプションが提供する高速オーバレイ機能を利用してオーバレイ印刷を行う場合 “PRINTMONITOR” を指定します。本指定が省略された場合、“COBOL” が指定されたものとみなされます。なお、

本指定は、環境変数情報@CBR\_OverlayPrintSPECの指定よりも優先されます。I/S制御レコードについては、“8.3.2 プログラムの記述”を参照してください。[参照]“5.4.1.38 @CBR\_OverlayPrintSPEC (フォームオーバーレイ印刷方法の指定)”

## 注意

- PrintWalker/OVLオプションは、富士通のVSPシリーズの特定のプリンタにだけ適用される装置オプションです。したがって、PrintWalker/OVLオプションの高速オーバーレイ機能は、すべてのプリンタ装置で利用できるわけではありません。なお、高速オーバーレイ機能に関する詳細情報については、“PrintWalker/OVLオプションのオンラインマニュアル”を参照してください。
- 利用できるフォームオーバーレイの形式は、FORMで作成したKOL2およびKOL5形式です(KOL6形式は使用できません)。
- オーバーレイ定義体格納フォルダ指定(環境変数情報FOVLDIR)、オーバーレイ定義体形式(環境変数情報FOVLTYPE)およびオーバーレイ定義体拡張子(環境変数情報OVD\_SUFFIX)の指定方法は、従来のCOBOLランタイムシステムによるオーバーレイ印刷方式および高速オーバーレイによる印刷方式と同じです。ただし、高速オーバーレイ指定時にオーバーレイ定義体格納フォルダ指定(環境変数情報FOVLDIR)を省略した場合は、PrintWalker/OVLオプションのデフォルトフォルダが指定されたものとみなします。

## 例

```
FOVLDIR=¥¥DATASERVER¥FOVL ← C/S 運用時の例
FOVLDIR=C:¥FOVL ← スタンドアローン運用時の例
FOVLDIR= ← PrintWalker/OVLオプションが定める
 デフォルトフォルダ指定の例

FOVLTYPE=KOL5
OVD_SUFFIX=None
```

- COBOLアプリケーションから縮小印刷指定(I制御レコードのPRT-FORM:印刷形式フィールド)に“LP”(ラインプリンタモード)を指定しても有効になりません。
- COBOLアプリケーションから印刷原点指定(I制御レコードのWIDTH:とじしろ幅フィールドおよびOFFSET:印刷開始原点位置フィールド)を指定した場合、その印刷結果は保証されません。印刷原点指定を行いたい場合は、COBOLからの印刷原点指定は行わず、プリンタドライバ側で印刷原点を設定してください。この場合、I制御レコードのWIDTH:とじしろ幅フィールドおよびOFFSET:印刷開始原点位置フィールドに9999または空白を指定してください。
- COBOLアプリケーションから両面印刷時の印刷開始面を裏面に位置付けた場合、本来白紙となるべき表面にオーバーレイパターンが印刷されることがあります。両面印刷時の印刷開始面を裏面に位置付けている場合とは、I制御レコードのSIDE:印刷面フィールドに“B”(両面印刷)、および、POSIT(印刷開始面)に“B”(裏面位置付け)を指定している場合です。

## OverlayPrintOffset

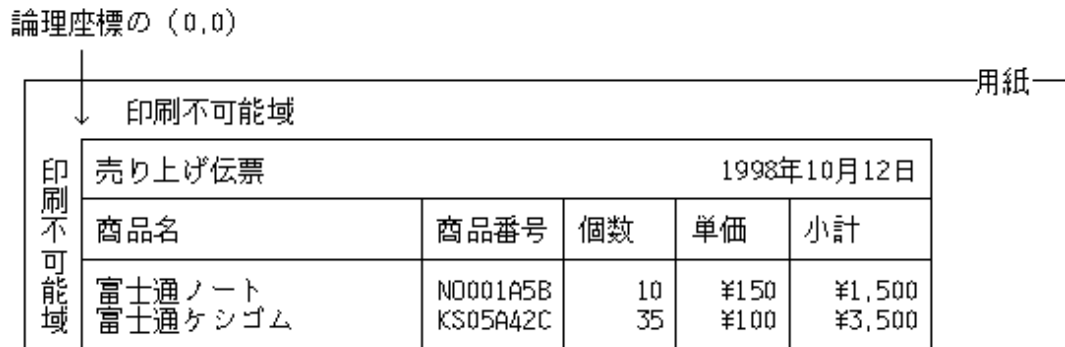
FORMAT句なし印刷ファイルで、I制御レコードに指定されたとじしろ方向(BIND)、とじしろ幅(WIDTH)および印刷原点位置(OFFSET)機能をフォームオーバーレイに対しても有効にする(VALID)か、無効にする(INVALID)かを指定します。本指定が省略された場合、“INVALID”が指定されたものとみなされます。なお、本指定は、環境変数情報@CBR\_OverlayPrintOffsetの指定よりも優先されます。[参照]“5.4.1.37 @CBR\_OverlayPrintOffset (I制御レコードのとじしろ方向、とじしろ幅および印刷原点位置指定をフォームオーバーレイに対して有効または無効にする指定)”

## 参考

V40L20以前のバージョンでは、I制御レコードに指定されたとじしろ方向(BIND)、とじしろ幅(WIDTH)および印刷原点位置(OFFSET)機能は、行レコード(文字データ)にだけ有効でした(本環境変数情報のINVALID指定時と同じ)。これに対し、本環境変数情報にVALIDを指定することで、I制御レコードの同機能を行レコードおよびフォームオーバーレイの両方に対して有効にすることができます。以下に、原点オフセットを設けない通常の印刷結果と、上方向および左方向にそれぞれ1インチの原点オフセットを設けた場合の印刷結果を例にとり、VALID/INVALID指定時(または省略時)の印刷結果の相違について図示します。

なお、以下の図では、罫線=オーバーレイ、文字=行レコードを示しています。

- ー 印刷原点位置を指定しない場合の印刷結果



- ー X,Y座標にそれぞれ1インチの原点オフセットを設けた場合の印刷結果

- VALID指定の場合



- INVALID指定の場合



.....

## STREAM

FORMAT句なし印刷ファイルを使用して出力する帳票の出力方式を指定します。通常のプリンタ出力(紙印刷)を行う場合“PR”を指定します。ListWORKSと連携して電子帳票出力を行う場合“LW”を指定します。本指定を省略した場合、“PR”が指定されたものとみなされます。

なお、電子帳票出力を行う場合、ListWORKSが必要になります。電子帳票出力の詳細は、“[8.6 電子帳票出力機能を使う方法](#)”、“[ListWORKSのオンラインマニュアル](#)”および“[ListWORKSのヘルプ](#)”を参照してください。

## STREAMENV

ListWORKSと連携して電子帳票出力を行う場合(STREAM=LWを指定した場合)、電子帳票出力に関する静的な情報を定義した電子帳票情報ファイルのファイル名を絶対パスで指定します。電子帳票情報ファイルの指定は、省略することも可能です。省略した場合、ListWORKSが保持する省略時の解釈に従って動作します。

なお、電子帳票情報ファイルの定義は、ListWORKSによって規定されています。詳しくは、“[ListWORKSのオンラインマニュアル](#)”および“[ListWORKSのヘルプ](#)”を参照してください。

## PRTOUIT

プログラムで使用するプリンタを指定します。以下のような場合に指定します。

- ファイル識別名に対して割り当てた出力先(“LPTn:”/“COMn:”/“PRTNAME:プリンタ名”)を変更したい場合
- ファイル識別名に対する出力先の割り当てを省略した場合に有効となる出力先を指定したい場合

本指定は、プログラムを変更することなく出力先を変更することが可能になるため、以下の場合には特に便利な指定となります。

- ASSIGN句にデータ名指定または定数指定を使用して、プログラム中に直接出力先を指定している場合

指定可能な出力先は、ファイル識別名の割り当て方法の場合と同じです。実際に帳票出力を行うプリンタが接続されているローカルプリンタポート名(LPTn:)/通信ポート名(COMn:)/プリンタの名前(PRTNAME:プリンタ名)を指定します。ただし、ここでは印刷情報ファイル名(INF(~))およびフォントテーブル名(FONT(~))は指定できません。

PRTOUIT指定は、省略することも可能です。PRTOUIT指定を省略した場合、ファイル識別名に対する指定が有効となります。ファイル識別名および印刷情報ファイルのPRTOUIT指定の両方にそれぞれ異なるローカルプリンタポート名(LPTn:)/通信ポート名(COMn:)/プリンタの名前(PRTNAME:プリンタ名)を指定した場合、印刷情報ファイルのPRTOUIT指定が優先されます。両方とも省略された場合または指定に誤りがある場合は、実行時エラーとなります。ファイル識別名の詳細は、“[5.4.1.72 ファイル識別名 \(プログラムで使用するプリンタおよび各種パラメタの指定\)](#)”を参照してください。

なお、ListWORKS連携により電子帳票出力機能を使用する場合、“LPTn:”および“COMn:”は指定できません。この場合、“PRTNAME:プリンタ名”を使用し、プリンタ名には、ListWORKSで電子化・登録するときに使用するListWORKS仮想プリンタ名またはデータ転送コネクタ名(以降、総称して電子保存装置と呼びます)を指定する必要があります。詳細は、“[8.6 電子帳票出力機能を使う方法](#)”を参照してください。

## FOVLDIR

フォームオーバーレイパターンファイルの格納先フォルダを絶対パス名で指定します。省略された場合、フォームオーバーレイパターンの焼付けは行われません。また、複数のフォルダを指定することはできません。

なお、本指定は、実行環境変数“[5.4.1.74 FOVLDIR \(フォームオーバーレイパターンのフォルダの指定\)](#)”の指定よりも優先されます。

## FOVLTYPE

フォームオーバーレイパターンのファイル名の先頭4文字が“KOL5”(省略値)以外の場合、形式に、ファイル名の先頭4文字を指定します。ファイル名に形式を持たない場合は、文字列“None”を指定してください。



### 例

.....

フォームオーバーレイパターンファイルが“C:\FOVLDATA\FKOL2OVD1.OVD”の場合

```
FOVLDIR=C:\FOVLDATA
FOVLTYPE=KOL2
FOVLNAME=OVD1 または I/S制御のFOVLフィールドで “OVD1” を指定してWRITE
```

.....

なお、本指定は、実行環境変数“[5.4.1.75 FOVLTYPE \(フォームオーバーレイパターンのファイル名の形式の指定\)](#)”の指定よりも優先されます。



## FOVLNAME

フォームオーバーレイパターンファイル名から先頭4文字の形式部分を除いた、後半のファイル名を4文字以内で指定します。本指定は、I/S制御レコードが出力されていない場合、およびI/S制御レコードでオーバーレイ出力が指示されていない(FOVLフィールドが空白指定)の場合に意味を持ちます。



### 例

フォームオーバーレイパターンファイルが“C:¥FOVLDATA¥KOL2TEST.OVL”の場合

```
FOVLDIR=C:¥FOVLDATA
FOVLTYPE=KOL2
FOVLNAME=TEST かつ (I/S制御レコードのWRITEなし
 または I/S制御のFOVLフィールドに空白を指定してWRITE)
OVD_SUFFIX=OVL
```

なお、本指定は、実行環境変数“5.4.1.76 FOVLNAME (フォームオーバーレイパターンファイル名の指定)”の指定よりも優先されます。

## OVD\_SUFFIX

フォームオーバーレイパターンファイルの拡張子が“OVD”(省略値)以外の場合、拡張子として使用する文字列を指定します。ファイル名に拡張子を持たない場合は、文字列“None”を指定してください。



### 例

フォームオーバーレイパターンファイルが“C:¥FOVLDATA¥KOL5ABCD”の場合

```
FOVLDIR=C:¥FOVLDATA
FOVLTYPE=KOL5
FOVLNAME=ABCD または I/S制御のFOVLフィールドで “ABCD” を指定してWRITE
OVD_SUFFIX=None
```

なお、本指定は、実行環境変数“5.4.1.77 OVD\_SUFFIX (フォームオーバーレイパターンファイルの拡張子の指定)”の指定よりも優先されます。



### 注意

FOVLDIR、FOVLTYPE、FOVLNAMEおよびOVD\_SUFFIXの指定は、PrintWalker/OVLオプションによるフォームオーバーレイ印刷およびListWORKS連携によるフォームオーバーレイ印刷時は有効になりません。

## 8.1.10 印刷ファイル/表示ファイルの決定方法

COBOLプログラムで帳票印刷を行う場合、FORMAT句なし印刷ファイル、FORMAT句付き印刷ファイルまたは表示ファイルを使用します。

これらのファイル種別は、翻訳時にCOBOLコンパイラがソースプログラム中の特定の記述の有無を解析することにより決定付けられます。

以下に、特定の記述によって決定付けられるファイル種別の組合せを示します。

### 特定の記述

- [1] FORMAT句
- [2] PRINTER指定のASSIGN句
- [3] PRINTER-n指定のASSIGN句
- [4] LINAGE句



- [5] ADVANCING指定のWRITE文  
 [6] ファイル参照子"GS-ファイル識別名"

### 特定の記述によって決定付けられるファイル種別の組合せ

上記[1]～[6]の記述によって決定付けられるファイル種別の組合せを下表に示します。

| ファイル種別              | [1] | [2] | [3] | [4] | [5] | [6] |
|---------------------|-----|-----|-----|-----|-----|-----|
| FORMAT句なし印刷ファイル (注) | ×   | ○   | ○   | ○   | ○   | ×   |
| FORMAT句付き印刷ファイル     | ○   | ×   | ×   | ×   | △   | ×   |
| 表示ファイル              | △   | ×   | ×   | ×   | ×   | ○   |

- ：ファイル種別を決定付ける記述。  
 △：記述可能。ただし、ファイル種別を決定付ける条件にはならない。  
 ×：記述不可能。  
 注：[2]、[3]、[4]、[5]のどれか1つでも記述されていれば、FORMAT句なし印刷ファイルであると決定付けられます。

## 8.1.11 サービス配下の注意事項(印刷時)

Windowsシステムのサービス配下でCOBOLプログラムを動作させる場合、通常サービスはシステムアカウントでプロセスを立ち上げ、COBOLプログラムをバックグラウンドジョブとして動作させます。

この場合、システムアカウントに対するプリンタ環境が存在しないため、COBOLランタイムシステムはプリンタ名などの情報を取得することができず、印刷できないことがあります。

- ・FORMAT句付き/なし印刷ファイルおよび表示ファイル(宛て先PRT)を使用したCOBOLプログラムをサービス配下で実行する場合、そのファイルの出力先への割当ては明にプリンタ名を指定してください。出力先の指定にデフォルトのプリンタ、ローカルプリンタポート名(LPTn:)および通信ポート名(COMn:)を指定した場合、印刷されないことがあります(ファイルの割当てエラーが発生します)。



例

#### サービス配下で印刷可能なファイルの割当て例

- ー FORMAT句なし印刷ファイルの場合
  - ASSIGN句の指定  
SELECT 印刷ファイル ASSIGN TO S-PRTF.
  - 環境変数情報の指定  
PRTF=PRTNAME:FUJITSU VSP4620A
- ー FORMAT句付き印刷ファイルの場合
  - プリンタ情報ファイルの指定  
PRTDRV FUJITSU VSP4620A

サービスがシステムアカウントまたは特定のユーザアカウントでシステムに乗り込むかは、各サービスの仕様を確認してください。

なお、サービスによっては、コントロールパネルのサービスのスタートアップの設定により、特定のユーザアカウントでログインするように設定できる場合があります。この場合、使用するプリンタドライバをインストールしたときのユーザアカウントを設定することで、印刷できることがあります。

- ・サービス配下で印刷ファイルを使用する場合、プリンタ用紙サイズは、以下に設定した情報が有効になります。
  - ー サービス配下の場合は、プリンタのプロパティに設定された情報
  - ー 通常のアプリ動作の場合は、プリンタの印刷設定に設定された情報

## 8.1.12 帳票設計について

COBOLの帳票印刷には、行と桁の概念が不可欠です。特にFORMAT句なし印刷ファイルのように行レコード主体で帳票印刷を行う場合、実際にプログラムを作成する前にきめ細かい帳票設計が必要です。

このため、実際にプログラミングに入る前に、スペーシングチャートのような設計用紙(または、FORMのようなグリッド表示可能なツール)を用いて、実際の印刷イメージで設計した後、この帳票設計をもとにプログラミングを行ってください。

スペーシングチャートの縦方向のマスは、COBOLのWRITE～ADVANCINGの行制御およびFCB制御文のLPIオペランドやCHオペランドに対応(反映)させ、横方向のマスはPICTURE句の桁数およびCHARACTER TYPE句の文字間隔(CPI)PRINTING POSITION句の水平スキップの情報に対応(反映)させてください。また、帳票設計時は、縦方向は1インチ内に何行、横方向は1インチ内に何文字配置するかを把握してください。



例

[スペーシングチャートの例]

|                            |                  |       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|----------------------------|------------------|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|                            |                  | 1 インチ |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|                            |                  | 10CPI |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 印刷<br>先頭行<br>↓<br>1<br>インチ | B<br>L<br>P<br>I |       | * | * |   | T | E | S | T |   | L | I | S | T |   | * | * |
|                            |                  |       |   |   | S | U | B |   | T | I | T | T | L | E |   | X | X |
|                            |                  |       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|                            |                  | N     | N |   | X | X | X | X | X |   | Z | Z | Z | , | Z | Z | Z |
|                            |                  | N     | N |   | X | X | X | X | X |   | Z | Z | Z | , | Z | Z | Z |
|                            |                  | N     | N |   | X | X | X | X | X |   | Z | Z | Z | , | Z | Z | Z |
|                            |                  | N     | N |   | X | X | X | X | X |   | Z | Z | Z | , | Z | Z | Z |

[FCB制御文]

```
LPI ((6)) , CH1 (1) , FORM (...
```

[COBOLプログラム]

```
SPECIAL-NAMES.
 PRINTING MODE PM1 IS FOR ALL
 AT PITCH 10.00 CPI.
:
DATA DIVISION.
:
01 印刷レコード PIC X(80)
 CHARACTER TYPE IS PM1.
:
PROCEDURE DIVISION.
:
 OPEN OUTPUT 印刷ファイル.
 MOVE " ** TEST LIST **" TO 印刷データ.
 WRITE 印刷レコード FROM 印刷データ
 AFTER ADVANCING PAGE.
:
```

## 8.1.13 印刷不可能な領域について

### 用紙内で印刷可能な文字数

用紙内に印字可能な最大文字数は、用紙サイズ(横方向)および文字ピッチ(CPI)によって決まります。たとえば、使用する用紙サイズが15×11インチの連続用紙で文字ピッチが10CPIであると仮定した場合、15インチ(用紙サイズ横方向)×10CPIで単純計算により150文字印刷可能であることがわかります。

しかし、後述の注意事項でも述べているように、連続用紙の場合は左右にトラクタに掛けるための穴が空いており、一般的には左右合わせて約1.4インチ(プリンタ機種により異なります)は物理的に印字が不可能な領域があります。したがって、実際には横15インチすべてが印字可能な領域ではなく、印字不可能な領域を間引きした約13.6インチが印字可能な領域であり、この場合の印字可能文字数は136文字ということになります。

### 用紙内で印字可能な行数

用紙内に印字可能な行数は、FCB制御文の定義により決定します。

FCB制御文は、プリンタ装置にセットされている物理的な用紙のサイズ(縦方向)、行間隔(LPI)およびチャネル位置を指定します。用紙内の印字可能行数は、FCB制御文で定義された用紙サイズと行間隔から決まります。

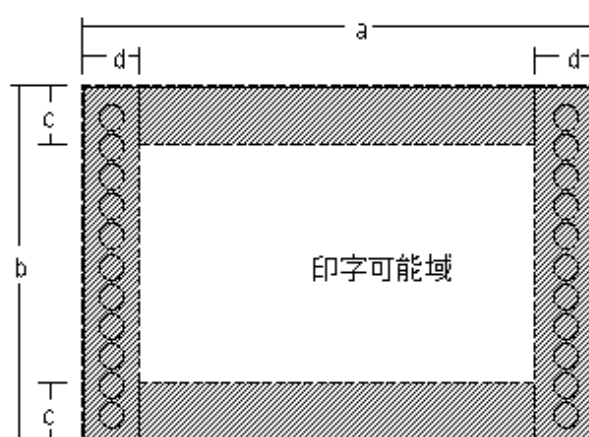
以下に、15×11インチの連続用紙を使用した場合のFCB制御文の定義例を示します。

- 行間隔:6LPI
- 用紙内最大印字可能行数:66行
- 用紙サイズ(縦方向):11インチ(10倍にした値を指定)

#### FCB制御文の定義例

FCBXXXX=LP((6, 66)), CH1(1), SIZE(110)

↑ ↑ ↑  
(1) 6LPI (2) 66行 (3) 11インチ (10倍にした値を指定)



調整可能だが、出荷時の状態は以下の設定

- a: 用紙サイズ(横方向)=15インチ
- b: 用紙サイズ(縦方向)=11インチ
- c: 印字不可能域=22.0mm
- d: 印字不可能域=

| 用紙幅(インチ)  | 打ち出し位置(mm) |
|-----------|------------|
| 4.0～ 4.7  | 5.08 ～ 9.0 |
| 15.3～16.0 | 15.0 ～30.0 |



## 注意

プリンタのハード仕様に依存する部分で、用紙の上下左右に物理的な印字不可能域が設けられているプリンタがあります。これらのプリンタに対応したプリンタドライバはこの部分への印字を抑止しています(データが印字可能域までシフトされたり捨てられます)。物理的な印字不可能域の大きさはプリンタによりさまざまであり、利用者はプリンタの取扱い説明書などを参照し、印字不可能域を考慮した帳票設計を行う必要があります。このため、上記の印字可能文字数および行数はあくまでも目安であり、すべてがこの限りではありませんので注意してください。

## 8.1.14 フォントテーブル

フォントテーブルとは、印刷ファイルを利用して帳票出力を行うときの書体情報を定義するテキスト形式のファイルのことです。

書体情報には、印字する文字のフォントフェイス名および印字スタイルを書体番号と対応付けて指定します。

CHARACTER TYPE句に印字モード名を指定して、その印字モード名を定義したPRINTING MODE句に書体番号を指定した場合、書体番号の情報を含むフォントテーブルが必要になります。

フォントテーブルの書式を、以下に示します。

### 書式および設定情報

|                              |                      |
|------------------------------|----------------------|
| [書体番号]                       | ↔ セクション名 (書体番号ごとに指定) |
| FontName=フォントフェイス名           | ↔ 文字書体指定 (省略可)       |
| Style={<br>R<br>B<br>I<br>BI | ↔ 印字スタイル指定 (省略可)     |

### セクション名

セクション名[書体番号]は、書体情報がどの書体番号に対応付けられた情報かを識別するための情報です。書体番号ごとに書体情報の設定が必要です。

書体番号には、COBOLソースプログラムに記述した書体番号("FONT-nnn")を記述します。文字列は、英数字の半角文字で記述してください。



## 例

[FONT-001]

### FontName

フォントフェイス名には、印字に使用する文字書体を指定します。省略した場合、明朝(通常の手体)で印字されます。



## 注意

- ・ 印字に使用するフォントフェイス名は、32バイト以内の英数字または日本語文字で指定してください。
- ・ フォントフェイス名は、以下の操作で表示されるウィンドウ内の書体名を指定してください。
  - ー コントロールパネルのフォントを選択します。
  - ー 表示されるフォントの一覧から該当フォントを選択し、ダブルクリックします。

## Style

印字に使用する文字書体のスタイルを指定します。省略した場合、標準(R)で印字されます。

- R : 標準
- B : 太字
- I : 斜体
- BI : 太字・斜体



例

```
[FONT-001] ←MS 明朝 を太字で印字指定
FontName=MS 明朝
Style=B
```

```
[FONT-002] ←MS 明朝 を斜体で印字指定
FontName=MS 明朝
Style=I
```

:

印刷ファイルでフォントテーブルを使用する場合は、環境変数情報@CBR\_PrintFontTableまたはファイル識別名にフォントテーブル名を指定します。



参照

“5.4.1.40 @CBR\_PrintFontTable (印刷ファイルで使用するフォントテーブルの指定)”

“5.4.1.71 ファイル識別名 (プログラムで使用するプリンタ情報ファイルおよび各種パラメタの指定)”

“5.4.1.72 ファイル識別名 (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)”

## 8.2 行単位のデータを印刷する方法

ここでは、FORMAT句なし印刷ファイルを使って、行単位のデータを印刷する方法について説明します。なお、FORMAT句なし印刷ファイルを使った例題プログラムがサンプルとして提供されていますので、参考になしてください。

### 8.2.1 概要

印刷ファイルは、レコード順ファイルと同様に定義し、レコード順ファイルの創成処理と同様の処理を行います。FORMAT句なし印刷ファイルでは、以下を指示することができます。

- 論理的な1ページの大きさ(ファイル記述項のLINAGE句)
- 文字の大きさ、書体、形態、方向および間隔(データ記述項のCHARACTER TYPE句)
- 行送りやページ替え(WRITE文のADVANCING指定)

### 8.2.2 プログラムの記述

ここでは、行単位のデータを使った印刷ファイルのプログラムの記述内容について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
```

```

[機能名 IS 呼び名].
INPUT-OUTPUT SECTION.
FILE-CONTROL.
 SELECT ファイル名
 ASSIGN TO PRINTER
 [ORGANIZATION IS SEQUENTIAL]
 [FILE STATUS IS 入出力状態].
DATA DIVISION.
FILE SECTION.
FD ファイル名
 [RECORD レコードの大きさ]
 [LINAGE IS 論理ページ構成の指定].
01 レコード名 [CHARACTER TYPE IS {MODE-1 | MODE-2 | MODE-3 | 呼び名}].
 レコード記述項
WORKING-STORAGE SECTION.
[01 入出力状態 PIC X(2) .]
[01 データ名 [CHARACTER TYPE IS {MODE-1 | MODE-2 | MODE-3 | 呼び名}].]
PROCEDURE DIVISION.
 OPEN OUTPUT ファイル名.
 WRITE レコード名 [FROM データ名] [AFTER ADVANCING ~].
 CLOSE ファイル名.
END PROGRAM プログラム名.

```

## 環境部(ENVIRONMENT DIVISION)

環境部には、機能名と呼び名の対応付け(プログラム中で印字文字を指示する場合)および印刷ファイルの定義を記述します。

### 機能名と呼び名の対応付け

印字文字の大きさ、書体、形態、方向および間隔の値を示す機能名を呼び名に対応付けます。この呼び名は、レコード中のデータ項目および作業用のデータ項目を定義するときに、CHARACTER TYPE句に指定します。機能名の種類については、“COBOL文法書”を参照してください。

### 印刷ファイルの定義

ファイル管理記述項を記述するために必要な情報を“表8.4 ファイル管理記述項に指定する情報”に示します。

表8.4 ファイル管理記述項に指定する情報

|    | 指定する場所        | 情報の種類        | 指定する内容および用途                                                                                                            |
|----|---------------|--------------|------------------------------------------------------------------------------------------------------------------------|
| 必須 | SELECT句       | ファイル名        | COBOL プログラム中で使用するファイル名を指定します。このファイル名は、COBOL の利用者語の規則に従った名前にします。                                                        |
|    | ASSIGN句       | ファイル参照子      | PRINTER、PRINTER-n、アクセス名、ローカルプリンタポート名(LPTn:)、シリアルポート名(COMn:) またはプリンタ名を指定します。PRINTER を指定すると、通常使うプリンタに設定されているプリンタに出力されます。 |
| 任意 | ORGANIZATION句 | ファイル編成を示す文字列 | SEQUENTIALを指定します。                                                                                                      |
|    | FILE STATUS句  | データ名         | 作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、入出力処理の実行結果が設定されません。(注)                                                 |

注：設定される値については、“付録B 入出力状態一覧”を参照してください。

## データ部(DATA DIVISION)

データ部には、レコードの定義およびファイル管理記述項に指定したデータ名の定義を記述します。レコードの定義は、ファイル記述項とレコード記述項で記述します。ファイル記述項を記述するために必要な情報を“表8.5 ファイル記述項に指定する情報”に示します。

表8.5 ファイル記述項に指定する情報

|    | 指定する場所  | 情報の種類    | 指定する内容および用途                                                                                |
|----|---------|----------|--------------------------------------------------------------------------------------------|
| 任意 | RECORD句 | レコードの大きさ | 印字可能領域の大きさを指定します。                                                                          |
|    | LINAGE句 | 論理ページの構成 | 論理的な1ページを構成する行数、上端と下端の余白の大きさおよび脚書き領域が始まる位置を指定します。この句にデータ名を指定すると、これらの情報をプログラム中で変更することができます。 |

## 手続き部(PROCEDURE DIVISION)

以下の順序で入出力文を実行します。

1. OUTPUT指定のOPEN文：印刷処理の開始
2. WRITE文：データの出力
3. CLOSE文：印刷処理の終了

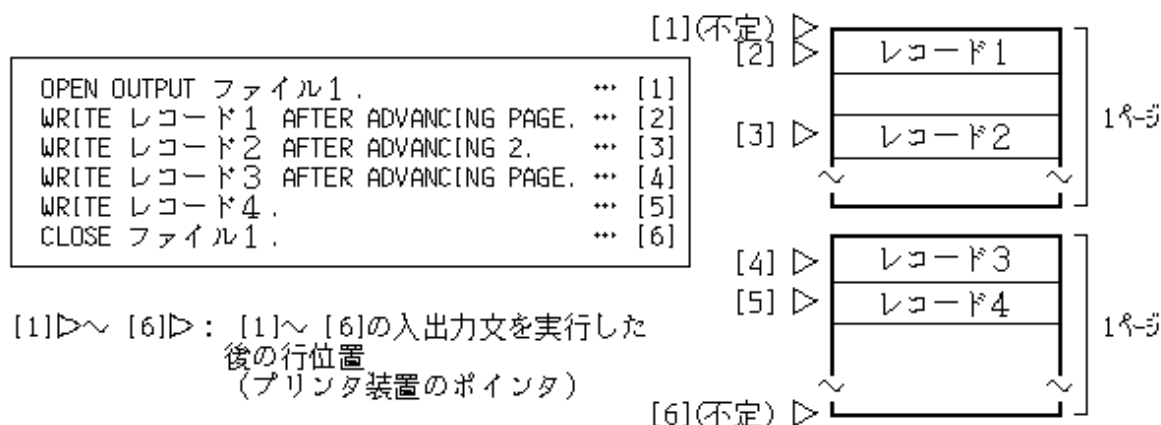
### OPEN文およびCLOSE文

OPEN文は印刷処理の開始時に、CLOSE文は印刷処理の終了時に、それぞれ1回だけ実行します。

### WRITE文

1回のWRITE文は、1行のデータを出力します。WRITE文のADVANCING指定にPAGEを記述すると、ページ替えが行われます。また、行数を記述すると、指定した行数が行送りされます。ADVANCING指定には、AFTER指定とBEFORE指定があり、データの出力をページ替えまたは行送りの後に行うか、前に行うかを指定します。ADVANCING指定を省略した場合、“AFTER ADVANCING 1”を指定したものとみなされます。

AFTER ADVANCING指定による印刷行の制御を以下に示します。



## 注意

- FROM指定を記述したWRITE文で、“WRITE A FROM B.”と記述した場合、CHARACTER TYPE句は、AではなくBに定義します。CHARACTER TYPE句が両方に定義された場合には、Bの指定が有効となります。
- OPEN文の実行直後のAFTER ADVANCING PAGE指定のWRITE文の実行では、ページ替えは行われません。

## 入出力エラー処理

入出力エラーの検出方法および入出力エラーが発生したときの実行結果については、“7.6 入出力エラー処理”を参照してください。

## 8.2.3 プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

## 8.2.4 プログラムの実行

はじめに、印刷装置の割当てを行います。印刷装置の割当ては、ファイル管理記述項のASSIGN句の記述内容によって異なります。ここでは、まず、プログラムまたは実行用の初期化ファイルでの印刷装置の指定方法について説明し、次にASSIGN句の記述内容と出力先の関係を例を用いて説明します。なお、印刷ファイルの出力先を印刷装置以外にした場合、出力内容は保証されません。

### 印刷装置の指定方法

印刷装置の指定は、以下の2種類の方法で設定することができます。

- ・ プリンタ名
- ・ ローカルプリンタポート名("LPTn:")またはシリアルポート名("COMn:")  
nは、1～9の範囲です。

プリンタ名の情報には、各プリンタの[プロパティ]ダイアログの情報を 사용합니다。

[プロパティ]ダイアログは、スタートボタンからプリンタを起動し、メニューから“プロパティ”を選択すると表示されます。以下に[プロパティ]ダイアログの[全般]シートを示します。



### プリンタ名

プリンタ名とは、[プロパティ]ダイアログの[全般]シートのプリンタマークの右横のエディットボックスに表示されているプリンタ名の先頭に、“PRTNAME:”を付けた名前のことです。



例

PRTNAME:FUJITSU VSP4620A



ネットワーク上のプリンタに接続している場合は、プリンタの名前に¥¥サーバ名¥の指定が必要です。



## 例

PRTNAME:¥¥PRTSVR¥FUJITSU VSP4620A



## 参考

プリンタ名の取得は、実行環境設定ツールの[環境設定]メニューから“プリンタ名”を選択することによって表示される[プリンタ名の選択]ダイアログを利用すると便利です。



## 注意

プリンタのプロパティにネットワークプリンタの名前は表示されませんので、必ず[プリンタ名の選択]ダイアログで指定するようにしてください。

### ローカルプリンタポート名/シリアルポート名

[プロパティ]ダイアログの[ポート]シートの印刷先のポートのリストボックスに表示されている名前を指定します。



## 例

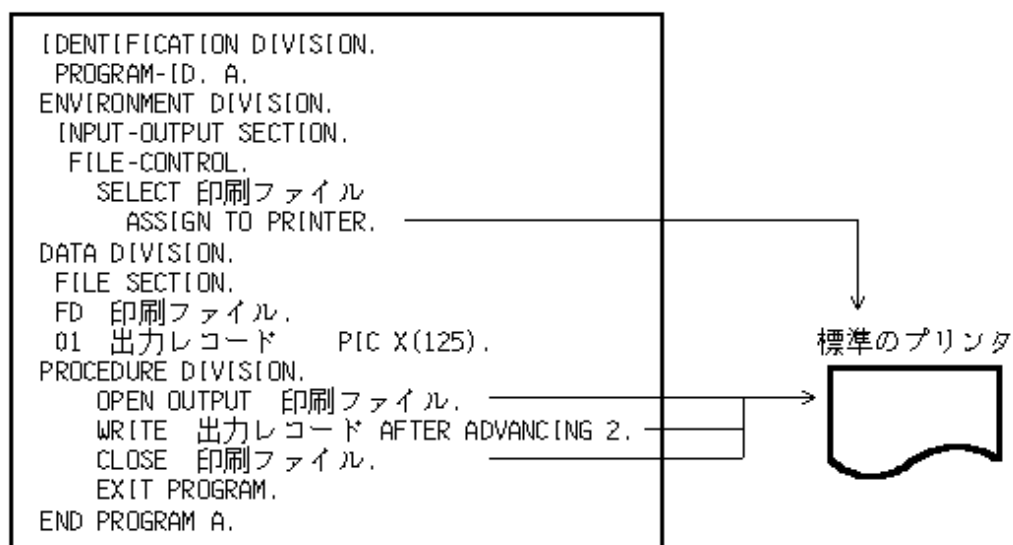
“LPT1:”

### プログラム例

ASSIGN句に文字列PRINTERを記述した場合

出力先は、標準のプリンタになります。

標準のプリンタとは、プリンタの設定で“通常使うプリンタに設定”がチェックされているプリンタのことです。



なお、特定のポートに接続されているプリンタ装置に出力する場合は、以降で説明する方法で行ってください。

#### ASSIGN句にファイル識別名を記述した場合

ファイル識別名を環境変数情報名として、出力先のプリンタ名またはローカルプリンタポート名/シリアルポート名を設定します。環境変数情報の設定方法は、“[5.3 実行環境情報の設定](#)”を参照してください。

なお、ファイル識別名にプリンタが割り当てられていない場合、ファイルの割当てエラーとなります。

図8.1 プリンタ名を指定した場合の例

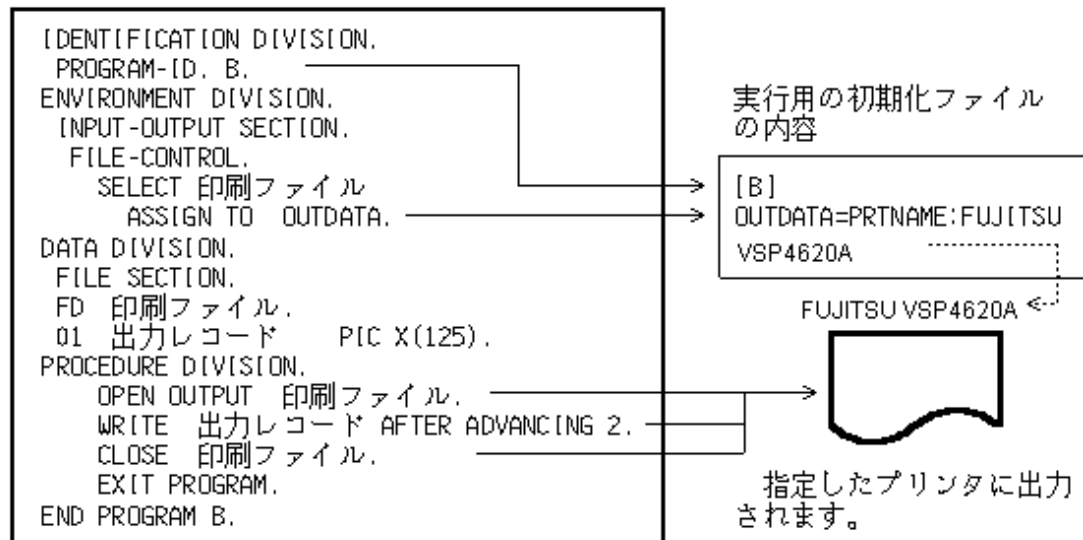
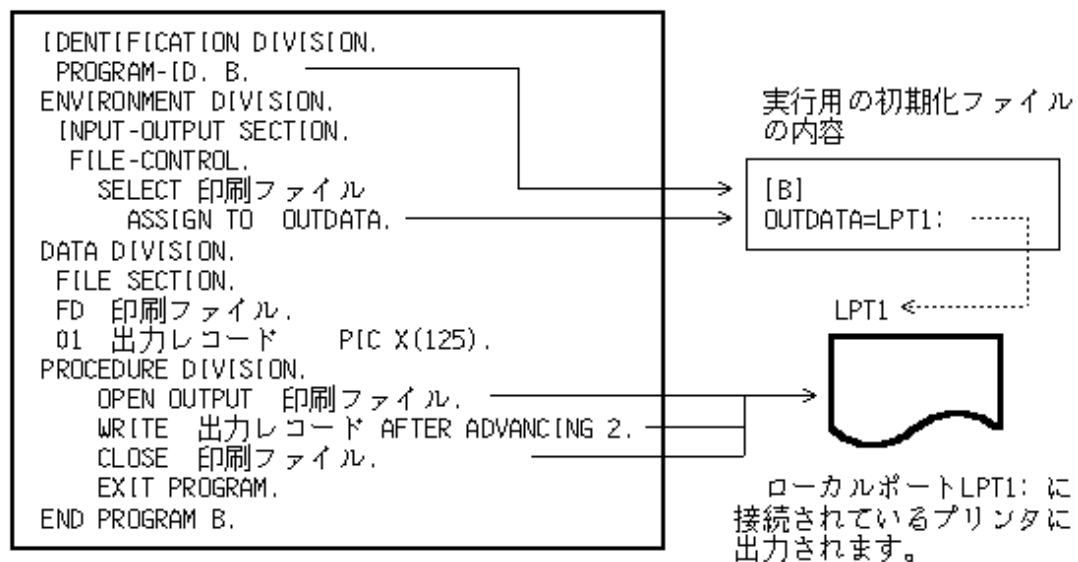


図8.2 ローカルポート名を指定した場合の例



#### ASSIGN句にファイル識別名定数を記述した場合

ファイル識別名定数に直接記述されたプリンタまたはファイル識別名定数に記述したローカルポート/シリアルポートに接続されているプリンタが出力先になります。

図8.3 プリンタ名を指定した場合の例

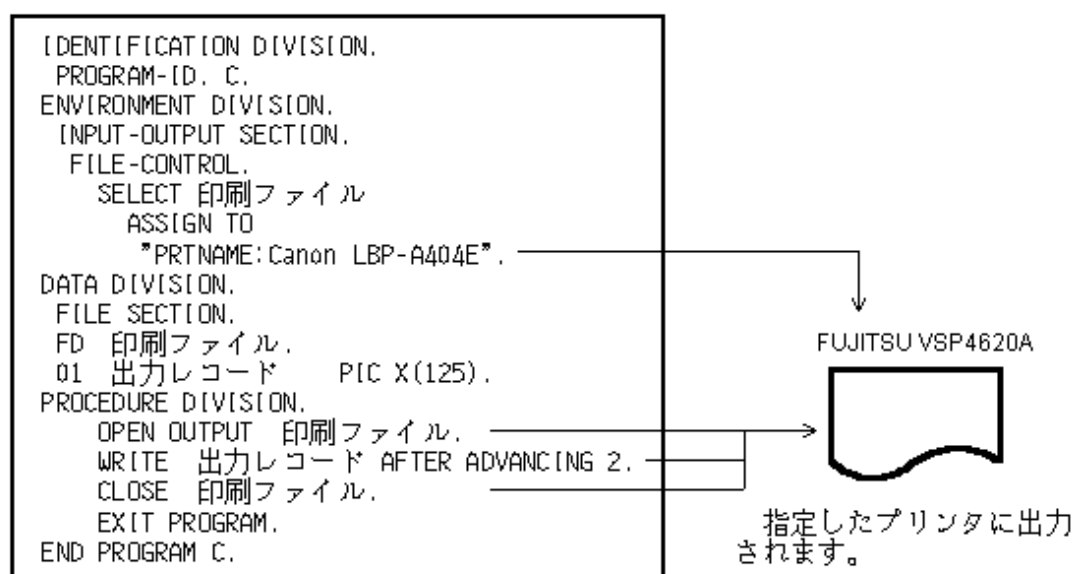
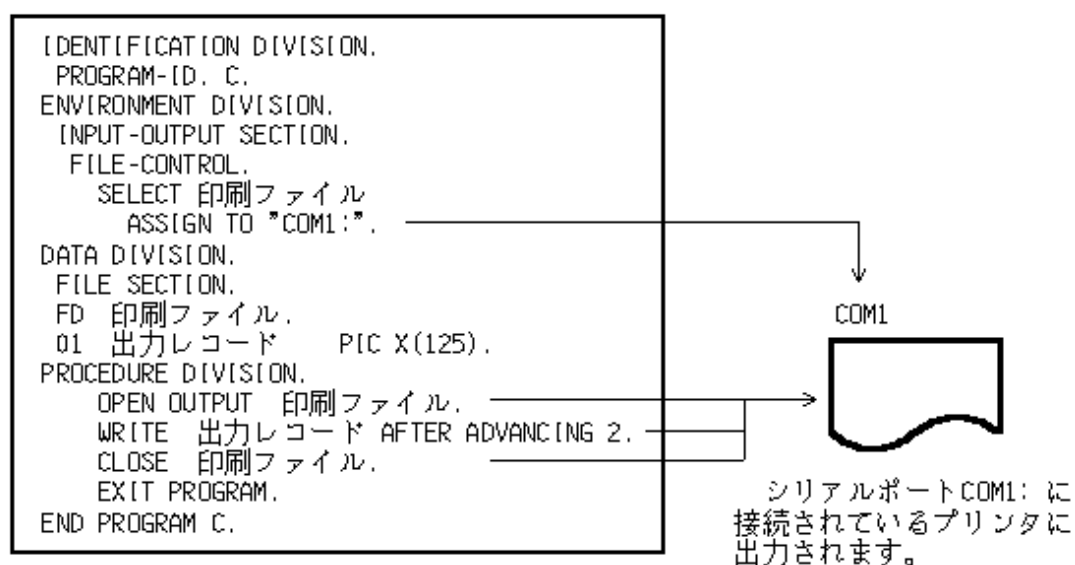


図8.4 シリアルポート名を指定した場合の例



#### ASSIGN句にデータ名を記述した場合

データ名に直接記述されたプリンタまたはデータ名に設定されたローカルポート/シリアルポートに接続されているプリンタが出力先になります。データ名の内容が空白の場合、ファイルの割当てエラーとなります。

図8.5 プリンタ名を指定した場合の例

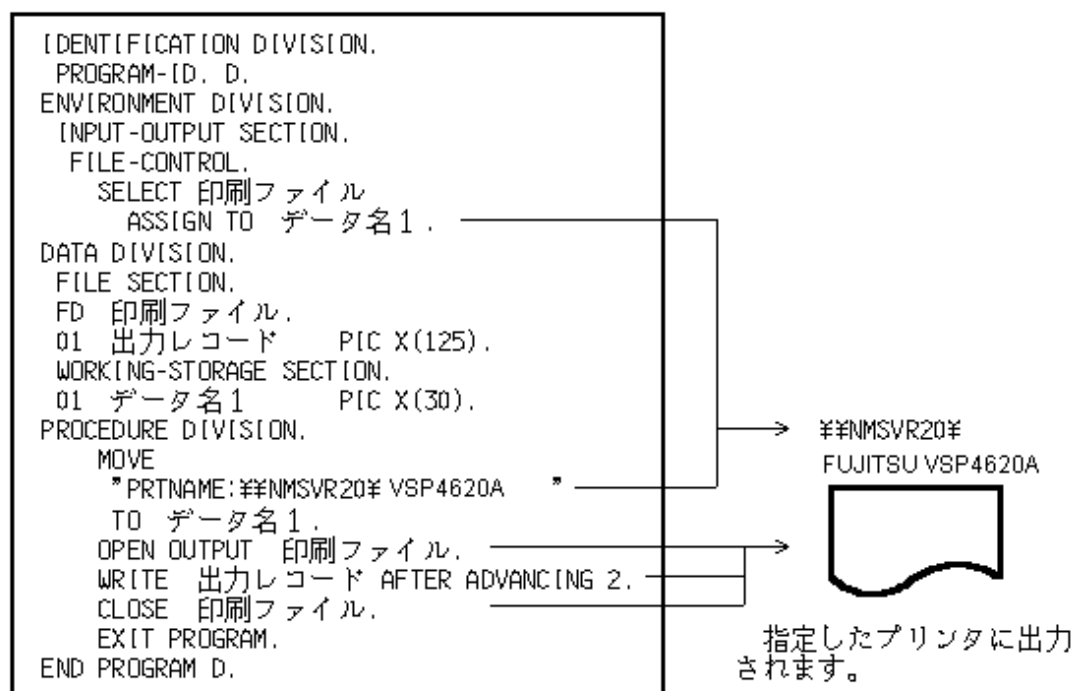
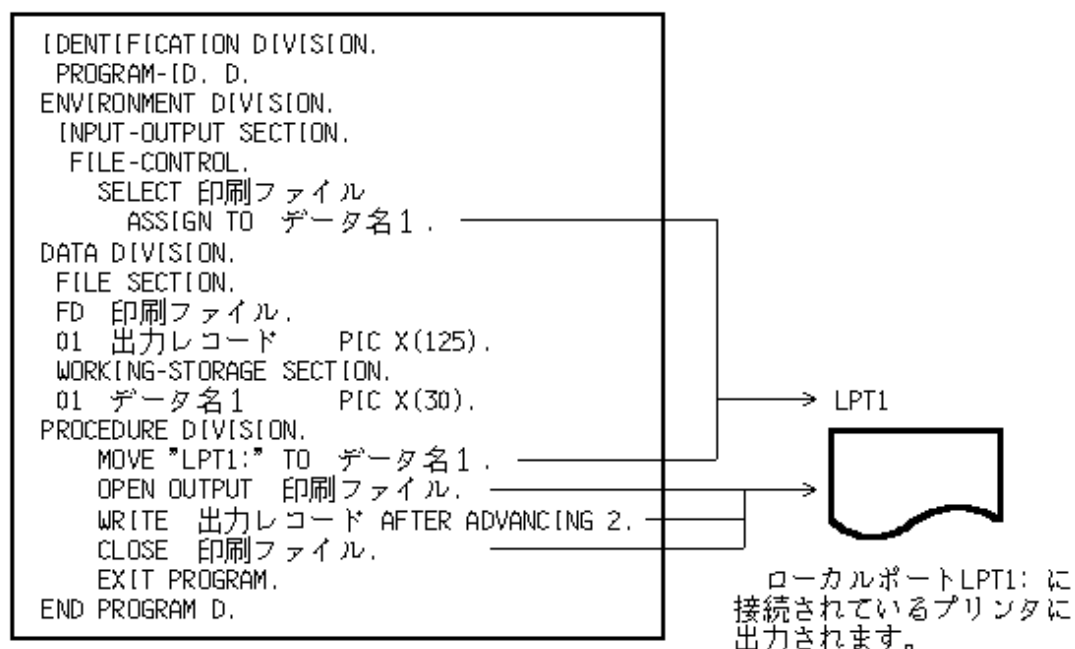


図8.6 ローカルポート名を指定した場合の例



#### ASSIGN句に文字列PRINTER-nを記述した場合

ファイル識別名を記述した場合と同様に、文字列PRINTER-n(n=1~9)を環境変数情報名として、出力先のプリンタ名またはローカルプリンタポート名/シリアルポート名を設定します。環境変数情報の設定方法については、“[5.3 実行環境情報の設定](#)”を参照してください。

なお、PRINTER-nにプリンタが割り当てられていない場合、ファイルの割当てエラーとなります。以下に、実行用の初期化ファイルを使った場合の例を示します。

図8.7 プリンタ名を指定した場合の例

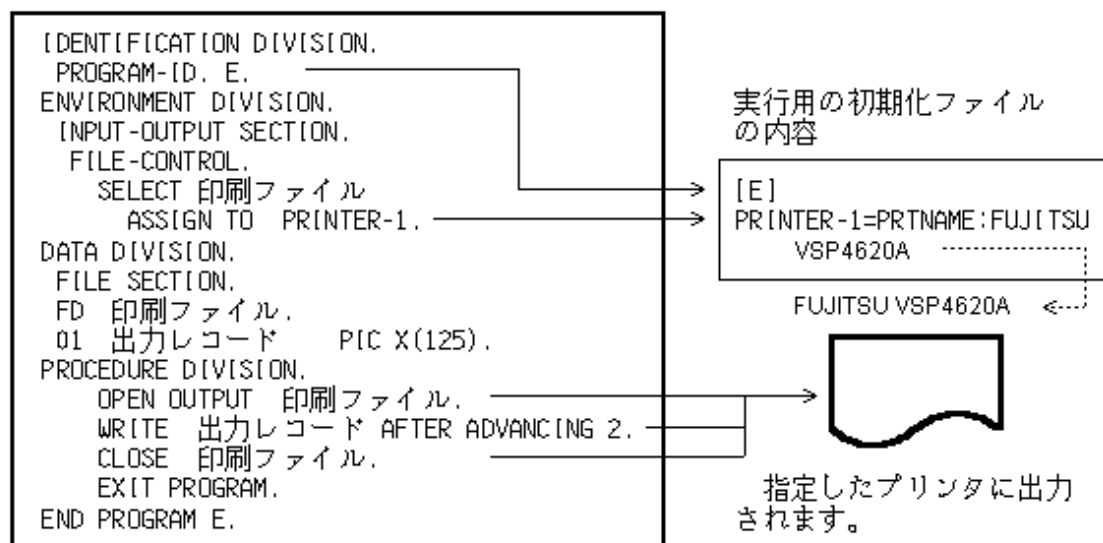
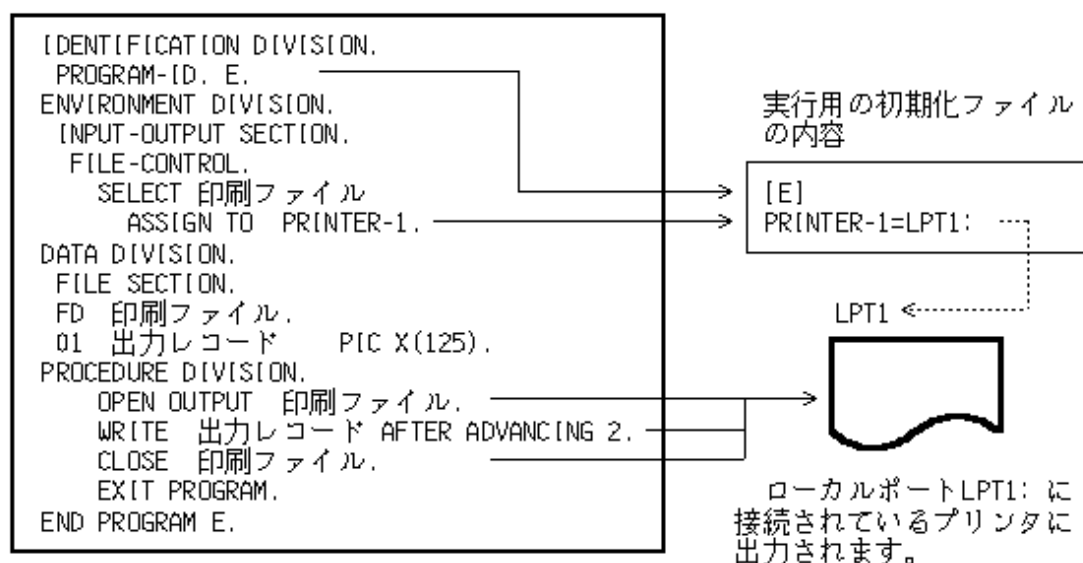


図8.8 ローカルポート名を指定した場合の例



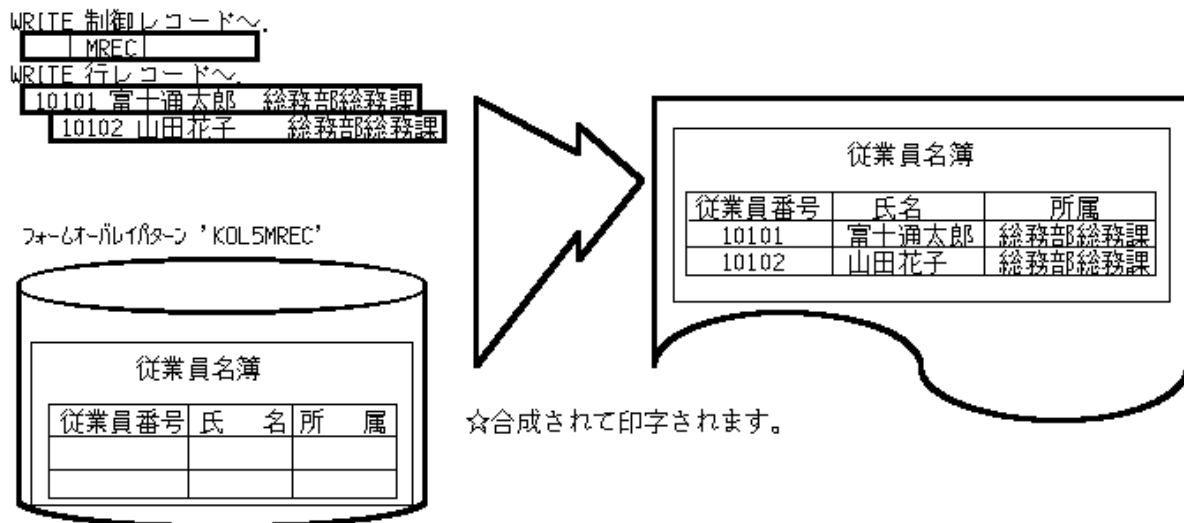
## 8.3 フォームオーバーレイおよびFCBを使う方法

ここでは、FORMAT句なし印刷ファイルでフォームオーバーレイパターンやFCBを使う方法について説明します。

### 8.3.1 概要

印刷ファイルでフォームオーバーレイパターンを使う場合には、制御レコードを使います。制御レコードは、通常のデータと同様に、WRITE文で出力します。フォームオーバーレイパターン名を設定した制御レコードを出力すると、その次のページに書き出したデータと、フォー

ムオーバーレイパターンが合成されて印字されます。データを印刷するときの印字する文字の大きさ、書体、形態、方向および間隔は、COBOLプログラムおよびフォームオーバーレイパターンで定義することができます。指定できる内容については、“8.1.2 印字文字”および“FORMヘルプ”を参照してください。



### 8.3.2 プログラムの記述

ここでは、フォームオーバーレイパターンを使って帳票を印刷するときのプログラムの記述内容について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
[機能名 IS 呼び名 1]
CTL IS 呼び名 2.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT ファイル名
ASSIGN TO PRINTER
[ORGANIZATION IS SEQUENTIAL]
[FILE STATUS IS 入出力状態].
DATA DIVISION.
FILE SECTION.
FD ファイル名
[RECORD レコードの大きさ]
[LINAGE IS 論理ページ構成の指定].
01 行レコード名 [CHARACTER TYPE IS {MODE-1 | MODE-2 | MODE-3 | 呼び名}].
レコード記述項
01 制御レコード名.
レコード記述項
WORKING-STORAGE SECTION.
[01 入出力状態 PIC X(2).]
[01 データ名 [CHARACTER TYPE IS {MODE-1 | MODE-2 | MODE-3 | 呼び名}].]
PROCEDURE DIVISION.
OPEN OUTPUT ファイル名.
WRITE 制御レコード名 AFTER ADVANCING 呼び名 2.
WRITE 行レコード名 AFTER ADVANCING PAGE.
[WRITE 行レコード名 [FROM データ名] [AFTER ADVANCING ~].]
```

```
CLOSE ファイル名.
END PROGRAM プログラム名.
```

## 環境部(ENVIRONMENT DIVISION)

環境部には、機能名と呼び名の対応付けおよび印刷ファイルの定義を記述します。

### 機能名と呼び名の対応付け

制御レコードを指定するための呼び名を、機能名CTLに対応付けます。この呼び名は、制御レコードを出力するときにWRITE文に指定します。

CHARACTER TYPE句を使って印字文字を指示する場合、印字文字の大きさ、形態、方向、書体および間隔の値を示す機能名を呼び名に対応付けます。機能名の種類については、“COBOL文法書”を参照してください。

### 印刷ファイルの定義

印刷ファイルは、ファイル管理記述項で定義します。ファイル管理記述項に記述する内容については、“[表8.4 ファイル管理記述項に指定する情報](#)”を参照してください。

## データ部(DATA DIVISION)

データ部には、レコードの定義および環境部で使ったデータ名の定義を記述します。

### レコードの定義

レコードは、ファイル記述項とレコード記述項で定義します。ファイル記述項に記述する内容については、“[表8.5 ファイル記述項に指定する情報](#)”を参照してください。レコード記述項には、以下のレコードを定義します。

#### 行レコード

プログラム中で編集したデータを印刷するためのレコードを定義します。行レコードは複数個記述することができます。行レコードの1つのレコードの内容は、印字可能領域の左端から順に印字されます。行レコードの大きさは、印字可能領域の1行の大きさを超えないように指定します。また、印字する文字の大きさを、データ記述項のCHARACTER TYPE句に指定することができます。CHARACTER TYPE句に指定できる内容については、“[8.1.2 印字文字](#)”を参照してください。

#### 制御レコード

制御レコードには、I制御レコードとS制御レコードがあります。I制御レコードおよびS制御レコードについては、“[8.1.5 I制御レコード/S制御レコード](#)”を参照してください。

## 手続き部(PROCEDURE DIVISION)

以下の順序で入出力文を実行します。

1. OUTPUT指定のOPEN文：印刷処理の開始
2. WRITE文：データの出力
3. CLOSE文：印刷処理の終了

### OPEN文およびCLOSE文

OPEN文は印刷処理の開始時に、CLOSE文は印刷処理の終了時に、それぞれ1回だけ実行します。

### WRITE文

WRITE文は、制御レコードおよび行レコードを出力するときに実行します。行レコードの出力は、行単位のデータを出力するときのWRITE文の使い方と同じです。

詳細は、“[8.2.2 プログラムの記述](#)”を参照してください。

制御レコードを出力するには、ADVANCING指定に、機能名CTLに対応付けた呼び名を記述します。

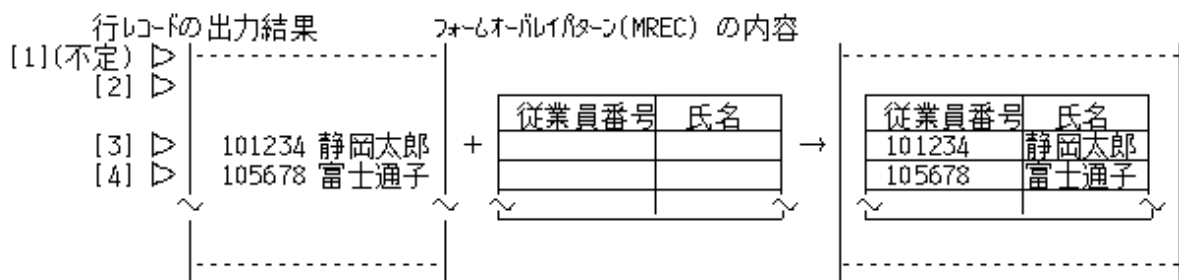
行レコードを使って出力したデータをフォームオーバーレイパターンと合成して印字するには、フォームオーバーレイパターン名を設定した制御レコードを出力します。また、フォームオーバーレイパターンと合成しないで印字するには、フォームオーバーレイパターン名に空白を設定した制御レコードを出力します。制御レコードを出力すると、その後に出力するページの形式が制御レコードの内容のとおり設定されます。ただし、制御レコードを出力すると、現在のページには行レコードを出力できなくなるので、制御レコード出力直後の行レコードの出力には、AFTER ADVANCING PAGEを指定する必要があります。

```
:
FILE SECTION.
```

```

FD ファイル 1.
:
01 制御レコード.
:
 02 FOVL PIC X(4).
:
MOVE "MREC" TO FOVL.
WRITE 制御レコード AFTER ADVANCING 呼び名. ... [1]
MOVE SPACE TO 行レコード.
WRITE 行レコード AFTER ADVANCING PAGE. ... [2]
MOVE 101234 TO 従業員番号.
MOVE NC"静岡太郎" TO 氏名.
WRITE 行レコード AFTER ADVANCING 2. ... [3]
MOVE 105678 TO 従業員番号.
MOVE NC"富士通子" TO 氏名.
WRITE 行レコード AFTER ADVANCING 1. ... [4]
:

```



▷ : [2] ~ [4] の入出力文を実行した後の行位置

[1] で出力した制御レコードの指示により、フォームオーバーレイパターン(MREC)と合成して印字されます。

## 入出力エラー処理

入出力エラーの検出方法および入出力エラーが発生したときの実行結果については、“[7.6 入出力エラー処理](#)”を参照してください。

## 8.3.3 プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

## 8.3.4 プログラムの実行

以下に、フォームオーバーレイパターンおよびFCBを使うプログラムの実行方法を示します。

### 8.3.4.1 フォームオーバーレイパターンを使うプログラム

印刷ファイルでフォームオーバーレイパターンを使用する場合、以下の設定を行っておく必要があります。

- 環境変数情報FOVLDIRまたは印刷情報ファイルのFOVLDIRキーにフォームオーバーレイパターンの格納先フォルダを指定します。環境変数情報FOVLDIRの設定が省略された場合、フォームオーバーレイパターンの焼付けは行われません。[参照]“[5.4.1.74 FOVLDIR \(フォームオーバーレイパターンのフォルダの指定\)](#)”、“[8.1.9 印刷情報ファイル](#)”

なお、FORMAT句付き印刷ファイルの場合、プリンタ情報ファイルに指定します。



- ・ フォームオーバーレイパターン格納ファイルの拡張子が“OVD”以外の場合、環境変数情報OVD\_SUFFIXまたは印刷情報ファイルのOVD\_SUFFIXキーに拡張子の文字列を指定します。なお、拡張子がない場合、文字列“None”を指定します。[参照]“[5.4.1.77 OVD\\_SUFFIX \(フォームオーバーレイパターンのファイルの拡張子の指定\)](#)”、“[8.1.9 印刷情報ファイル](#)”

なお、FORMAT句付き印刷ファイルの場合、プリンタ情報ファイルに指定します。

- ・ フォームオーバーレイパターン格納ファイル名の先頭4文字が“KOL5”以外の場合、環境変数情報FOVLTYPEまたは印刷情報ファイルのFOVLTYPEキーにファイル名の先頭4文字を指定します。先頭4文字を省略する場合、文字列“None”を指定します。[参照]“[5.4.1.75 FOVLTYPE \(フォームオーバーレイパターンのファイル名の形式の指定\)](#)”、“[8.1.9 印刷情報ファイル](#)”



## 例

### 実行用の初期化ファイルの記述例

フォームオーバーレイパターンを使ったプログラムAを実行するときの実行用の初期化ファイルの内容

|                     |       |
|---------------------|-------|
| [A]                 |       |
| FOVLDIR=C:\¥FOVLDIR | … [1] |
| OVD_SUFFIX=         | … [2] |
| FOVLTYPE=FOVL       | … [3] |

[1] フォームオーバーレイパターンが格納されているフォルダ(C:\¥FOVLDIR)を設定します。

[2] フォームオーバーレイパターン格納ファイルの拡張子は、省略時の“OVD”となります。

[3] フォームオーバーレイパターン格納ファイル名の先頭4文字は、“FOVL”となります。

## フォームオーバーレイパターンの出力

### フォームオーバーレイパターンの出力が可能なプリンタ

Windowsシステムのグラフィックデバイスインタフェース(GDI)を利用したソフトオーバーレイ機能に対応しています。したがって、Windowsシステムにプリンタドライバが用意されているプリンタの場合、フォームオーバーレイパターンの出力が可能です。

## 8.3.4.2 FCBを使うプログラム

印刷ファイルでFCBを使用する場合、実行用の初期化ファイルにFCB制御文を記述します。

FCB制御文の指定形式および内容については、“[8.1.4 FCB](#)”を参照してください。なお、I制御レコードにFCB名を指定し、実行用の初期化ファイルに該当するFCB制御文がない場合、エラーとなります。



## 例

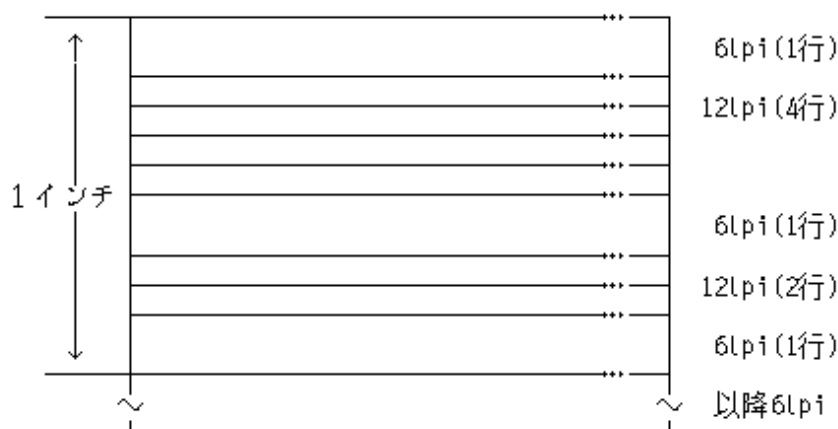
### FCB制御文の例

- ・ FCB名FCB1を使ったプログラムAを実行するときの実行用の初期化ファイルの内容

|                                                        |
|--------------------------------------------------------|
| [A]                                                    |
| FCBFCB1=LPI ((6, 1), (12, 4), (6, 1), (12, 2), (6, 1)) |

- ・ 出力形式

FCB制御文で設定される1ページの形式は、以下のとおりです。



## 8.4 帳票定義体を使う印刷ファイルの使い方

ここでは、FORMAT句付き印刷ファイルでパーティション形式の帳票定義体を使う方法について説明します。

### 8.4.1 概要

パーティション形式の帳票定義体を使った帳票の印刷には、図表レコードを使います。図表レコードには、帳票定義体に定義したパーティション(または項目群)を指定します。図表レコードは、通常データを出力するときと同様に、WRITE文を使って出力します。このとき、1回のWRITE文の実行でパーティションに含まれる複数行を印刷することができます。帳票の1ページは1つ以上のパーティションから構成されます。そのページがどのような構成で印刷されるのかは、図表レコードと行レコードの出力順序から決定されます。

図表レコードの定義項目は、COBOLのCOPY文を使って、帳票定義体から取り込むことができるため、利用者自身が記述する必要はありません。

印字する文字の大きさ、書体、形態、方向および間隔は、行レコードはCOBOLプログラムで、図表レコードは帳票定義体の作成時に指示することができます。行レコードのデータに対して指定できる内容については“[8.1.2 印字文字](#)”、図表レコードのデータに対して指定できる内容については“FORMヘルプ”を参照してください。

WRITE 図表レコード ～.

10101 富士通太郎 総務部総務課  
10102 山田花子 総務部総務課

帳票定義体

| 従業員名簿 |    |    |  |
|-------|----|----|--|
| 従業員番号 | 氏名 | 所属 |  |
|       |    |    |  |
|       |    |    |  |

| 従業員名簿 |       |        |
|-------|-------|--------|
| 従業員番号 | 氏名    | 所属     |
| 10101 | 富士通太郎 | 総務部総務課 |
| 10102 | 山田花子  | 総務部総務課 |

☆固定パーティションの場合、図表レコード中のデータは、帳票定義体で定義した位置に印字されます。  
☆浮動パーティションの場合、COBOL プログラムから印刷位置を指定できます。

## 注意

- 自由形式の帳票定義体は、FORMAT句付き印刷ファイルでは使用できません。表示ファイルの帳票印刷機能を使用してください。
- 帳票定義体を作成する場合、COBOLプログラムで設定/参照される名前は、以下の注意が必要です。
  - 帳票定義体名は8文字以内の半角英数字で指定します。
  - 項目群名およびパーティション名は6文字以内の半角英数字で指定します。
  - 項目名はCOBOLの利用者語の記述規則に従って指定します。

### 8.4.1.1 帳票のパーティション

パーティションには、固定パーティション、浮動パーティションと呼ばれる、2種の属性があります。

#### 固定パーティション

固定パーティションとはページ内での印刷位置が固定のパーティションで、帳票定義体で指定された位置に印刷されます。

#### 浮動パーティション

浮動パーティションとはページ内での印刷位置が出力順序により決められるパーティションで、WRITE文を実行したときの印刷位置にしがって印刷されます。

## 例

図表レコードに対するWRITE文と、それに対して出力される固定パーティションおよび浮動パーティションの例を以下に示します。

# 帳票定義体：見積書 (ESTIMATE.PMD)

|    |    |                         |
|----|----|-------------------------|
| 氏名 |    | 固定パーティション<br>(HEAD)     |
| 項目 | 金額 | 浮動パーティション<br>(ITEM)     |
| 小計 |    | 浮動パーティション<br>(SUBTOTAL) |
| 合計 |    | 固定パーティション<br>(TOTAL)    |

```

MOVE "ESTIMATE" TO 帳票定義体名通知域
MOVE "HEAD" TO 項目群名通知域
WRITE ESTIMATE ⇒ ⇒ ⇒ ⇒

MOVE "ITEM" TO 項目群名通知域
WRITE ESTIMATE ⇒ ⇒ ⇒ ⇒
WRITE ESTIMATE ⇒ ⇒ ⇒ ⇒

MOVE "SUBTOTAL" TO 項目群名通知域
WRITE ESTIMATE ⇒ ⇒ ⇒ ⇒

MOVE "TOTAL" TO 項目群名通知域
WRITE ESTIMATE ⇒ ⇒ ⇒ ⇒

```

1 ページ

|        |    |
|--------|----|
| 氏名     |    |
| 項目     | 金額 |
| 小計     |    |
| 印字されない |    |
| 合計     |    |

浮動パーティションは、WRITE文の実行順にしたがって(ADVANCING指定が記述されていればその行数分だけ行送りされた後に)印刷されます。点線内は、パーティションTOTALが帳票定義体で定義された固定位置に印刷されるため、印字されません。

## パーティションと行レコードの組合せ

図表レコードを使うことで、帳票定義体に定義したパーティションを印刷することができます。また、そのページの任意の位置に、通常の行レコードを出力することもできます。

パーティションと行レコードはWRITE文の実行順にしたがって(ADVANCING指定が記述されていればその行数分だけ行送りされた後に)印刷されます。



## 例

図表レコードと行レコードを混在した場合のWRITE文と、それに対して出力される固定パーティション、浮動パーティションおよび行データの例を以下に示します。

# 帳票定義体：見積書 (ESTIMATE.PMD)

|                                                 |                         |
|-------------------------------------------------|-------------------------|
| 氏名 <input type="text"/>                         | 固定パーティション<br>(HEAD)     |
| 項目 <input type="text"/> 金額 <input type="text"/> | 浮動パーティション<br>(ITEM)     |
| 小計 <input type="text"/>                         | 浮動パーティション<br>(SUBTOTAL) |
| 合計 <input type="text"/>                         | 固定パーティション<br>(TOTAL)    |

01 行レコード.  
02 値引き金額 PIC 9(9).

```

MOVE "ESTIMATE" TO 帳票定義体名通知域
MOVE "HEAD" TO 項目群名通知域
WRITE ESTIMATE

MOVE "ITEM" TO 項目群名通知域
WRITE ESTIMATE
WRITE ESTIMATE

MOVE "SUBTOTAL" TO 項目群名通知域
WRITE ESTIMATE
MOVE SPACE TO 帳票定義体名通知域
MOVE SPACE TO 項目群名通知域
WRITE 行レコード

MOVE "ESTIMATE" TO 帳票定義体名通知域
MOVE "TOTAL" TO 項目群名通知域
WRITE ESTIMATE

```

1 ページ

|                                                 |
|-------------------------------------------------|
| 氏名 <input type="text"/>                         |
| 項目 <input type="text"/> 金額 <input type="text"/> |
| 小計 <input type="text"/>                         |
| 値引き金額 999999999                                 |
| 合計 <input type="text"/>                         |



## 注意

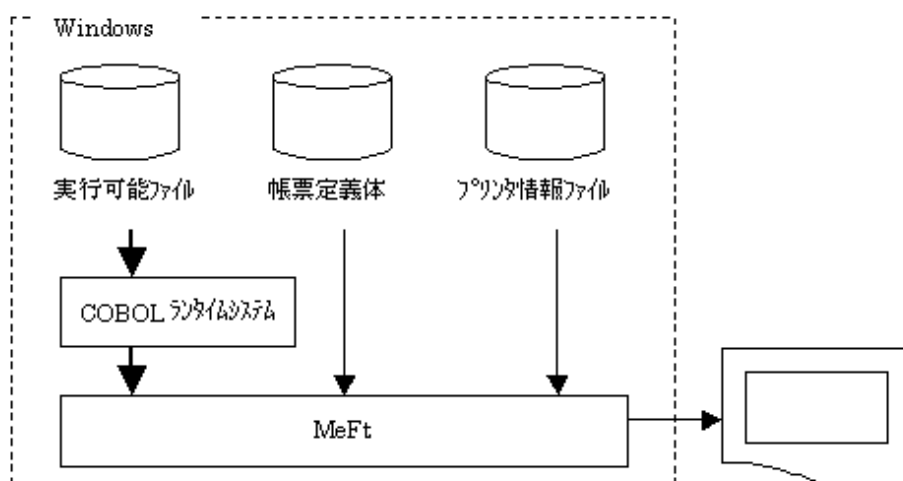
行レコードまたは浮動パーティションを固定パーティションの印刷位置に出力した場合には、その位置に印刷する固定パーティションは、そのページ中では印刷できません。そのページは改ページされて、固定パーティションは次のページの印刷位置に印刷されます。

## 8.4.1.2 帳票の電子化

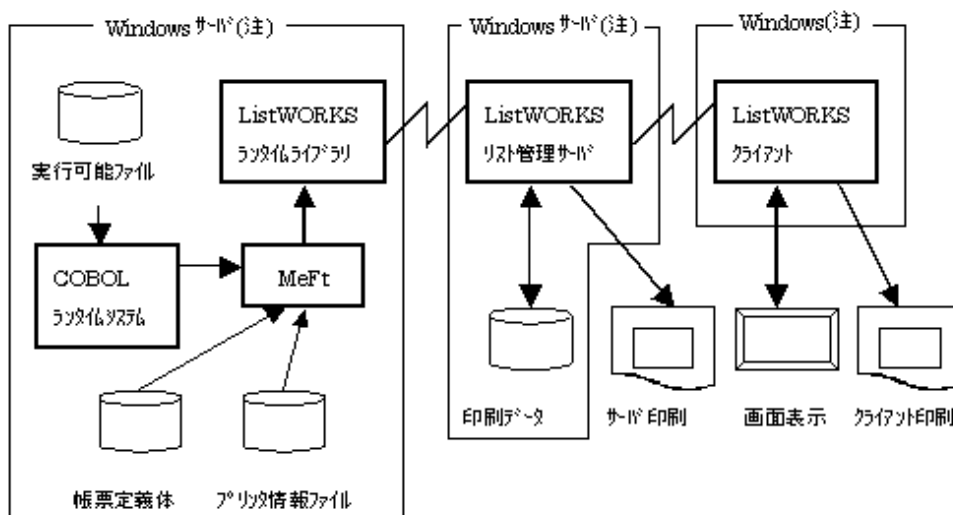
プリンタ情報ファイルに指定を追加することにより、MeFt経由で出力する帳票を電子化することができます。帳票を電子化する方法には、ListWORKS連携とe-DocGenerator連携があります。それぞれの詳細は、“MeFtのオンラインマニュアル”および各連携製品のマニュアルを参照してください。

以下にそれぞれの関連図を示します。

- ・ プリント出力の場合

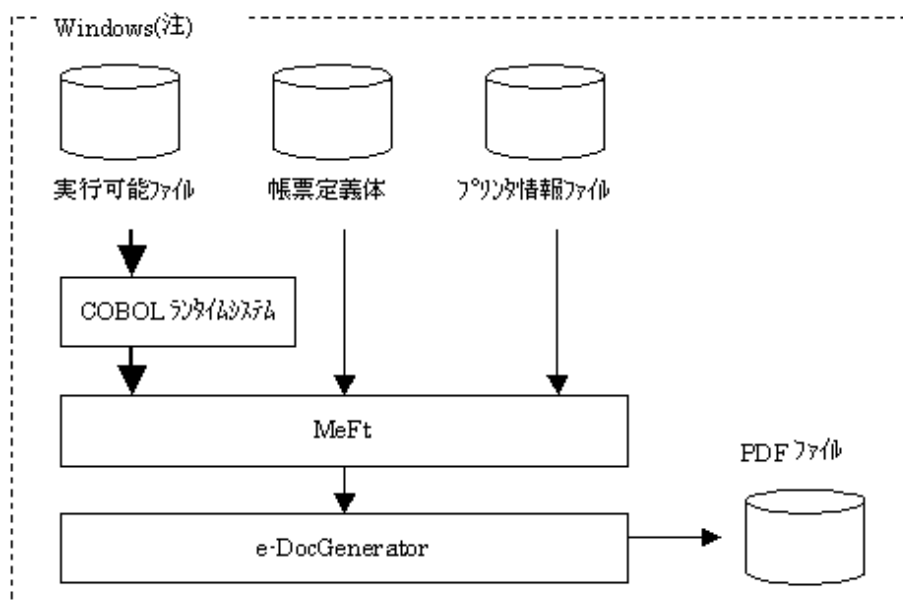


- ・ ListWORKS連携の場合



ListWORKS連携では、MeFtの出力をListWORKSで扱うことのできる電子帳票の形式にします。

- ・ e-DocGenerator連携の場合



e-DocGenerator連携では、MeFtの出力をPDF(Portable Document Format)ファイルにします。

注：ListWORKSおよびe-DocGeneratorが動作可能なオペレーティングシステムについては、各製品のマニュアルおよびソフトウェア説明書を確認してください。

## 8.4.2 プログラムの記述

ここでは、FORMAT句付き印刷ファイルで帳票定義体を使うプログラムの記述方法について説明します。

```

IDENTIFICATION DIVISION.
 PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
 SPECIAL-NAMES.
 [機能名 IS 呼び名.]
INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 SELECT ファイル名
 ASSIGN TO ファイル参照子
 [ORGANIZATION IS SEQUENTIAL]
 FORMAT IS 帳票定義体名通知域
 GROUP IS 項目群名通知域
 [FILE STATUS IS 入出力状態 1 入出力状態 2].
DATA DIVISION.
 FILE SECTION.
 FD ファイル名
 [RECORD レコードの大きさ]
 [CONTROL RECORD IS 制御レコード名].
 01 行レコード名 [CHARACTER TYPE IS {MODE-1 | MODE-2 | MODE-3 | 呼び名}].
 レコード記述項
 01 制御レコード名.
 レコード記述項
 COPY 帳票定義体名 OF XMDLIB.
 (01 図表レコード名.) (注)
 (レコード記述項)
WORKING-STORAGE SECTION.
 01 帳票定義体名通知域 PIC X(8).
 01 項目群名通知域 PIC X(8).
 [01 入出力状態 1 PIC X(2).]

```

```

[01 入出力状態 2 PIC X(4).]
[01 データ名 [CHARACTER TYPE IS {MODE-1 | MODE-2 | MODE-3 | 呼び名}].]
PROCEDURE DIVISION.
 OPEN OUTPUT ファイル名.
 MOVE 帳票定義体名 TO 帳票定義体名通知域.
 MOVE 項目群名 TO 項目群名通知域.
 WRITE 図表レコード名 [AFTER ADVANCING ~].
 WRITE 制御レコード名.
 MOVE SPACE TO 帳票定義体名通知域.
 MOVE SPACE TO 項目群名通知域.
 WRITE 行レコード名 [FROM データ名] AFTER ADVANCING PAGE.
 CLOSE ファイル名.
END PROGRAM プログラム名.

```

注：()内は、COPY文の展開を表します。

## 環境部(ENVIRONMENT DIVISION)

環境部には、機能名と呼び名の対応付け(プログラム中で印字文字を指示する場合)および印刷ファイルの定義を記述します。

### 機能名と呼び名の対応付け

CHARACTER TYPE句を使って印字文字を指示する場合、印字文字の大きさ、形態、書体、方向および間隔の値を示す機能名を呼び名に対応付けます。機能名の種類については、“COBOL文法書”を参照してください。

### FORMAT句付き印刷ファイルの定義

印刷ファイルは、ファイル管理記述項で定義します。ファイル管理記述項を記述するために必要な情報を“[表8.6 ファイル管理記述項に指定する情報](#)”に示します。

表8.6 ファイル管理記述項に指定する情報

|    | 指定する場所       | 情報の種類   | 指定する内容および用途                                                                                    |
|----|--------------|---------|------------------------------------------------------------------------------------------------|
| 必須 | SELECT句      | ファイル名   | COBOL プログラム中で使用するファイル名を指定します。このファイル名は、COBOL の利用者語の規則に従った名前にします。                                |
|    | ASSIGN句      | ファイル参照子 | ファイル識別名、ファイル識別名定数またはデータ名のどれかを記述します。ファイル参照子は、実行時にMeFtの使用するプリンタ情報ファイルを割り当てるために使用します。             |
|    | FORMAT句      | データ名    | 作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名は、帳票定義体名を設定するために使用します。                            |
|    | GROUP句       | データ名    | 作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名は、帳票定義体で定義した項目群名を設定するために使用します。                    |
| 任意 | FILE STATUS句 | データ名    | 作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、入出力処理の実行結果が設定されます。(注)<br>詳細情報は、4桁の英数字項目を指定します。 |

注：設定される値については、“[付録B 入出力状態一覧](#)”を参照してください。

ファイル参照子に、ファイル識別名、ファイル識別名定数またはデータ名のどれを指定したかによって、実行時にプリンタ情報ファイルを割り当てる方法が異なります。ファイル参照子に何を指定するかは、プリンタ情報ファイルの名前がいつ決まるかで決定されます。COBOLプログラム作成時にプリンタ情報ファイルの名前が決定し、その後変更されない場合には、ファイル識別名定数を指定します。COBOLプログラム作成時に名前が決定しなかった場合や、毎回のプログラム実行時に名前を決定したい場合には、ファイル識別名を指定します。また、プログラムの中で名前を決定したい場合には、データ名を指定します。

## データ部(DATA DIVISION)

データ部には、レコードの定義および環境部で指定したデータ名の定義を記述します。



## レコードの定義

レコードは、ファイル記述項とレコード記述項で定義します。ファイル記述項を記述するために必要な情報を“[表8.7 ファイル記述項に指定する情報](#)”に示します。

表8.7 ファイル記述項に指定する情報

|    | 指定する場所          | 情報の種類    | 指定する内容および用途       |
|----|-----------------|----------|-------------------|
| 任意 | RECORD句         | レコードの大きさ | 印字可能領域の大きさを指定します。 |
|    | CONTROL RECORD句 | 制御レコード名  | 制御レコード名を指定します。    |

レコード記述項には、行レコード、制御レコードおよび図表レコードを定義することができます。行レコードと制御レコードの定義方法および使い方については、フォームオーバーレイパターンを使うときと同様のため、“[8.3.2 プログラムの記述](#)”を参照してください。

図表レコードに定義するレコード記述文は、翻訳時にIN/OF XMDLIBを指定したCOPY文によって、帳票定義体から取り込むことができます。COPY文で展開される内容については、“[8.5.4 プログラムの記述](#)”を参照してください。

## 手続き部(PROCEDURE DIVISION)

以下の順序で入出力文を実行します。

1. OUTPUT指定のOPEN文：印刷処理の開始
2. WRITE文：データの出力
3. CLOSE文：印刷処理の終了

### OPEN文およびCLOSE文

OPEN文は印刷処理の開始時に、CLOSE文は印刷処理の終了時に、それぞれ1回だけ実行します。

### WRITE文

WRITE文では、行レコード、制御レコードおよび図表レコードを出力することができます。これらのレコードの出力内容により、1ページは固定形式ページまたは不定形式ページのどちらかとなります。固定形式ページとは、帳票定義体によってその構成が定義されているページであり、帳票定義体に定義された図表レコードまたは行レコードを印字できます。不定形式ページとは、帳票定義体によって定義されないページ、すなわち、FORMAT句なし印刷ファイルの印字ページと同じ意味のページであり、行レコードだけを印字できます。

固定形式ページに図表レコードと行レコードを混在させる場合は、ファイル記述項のCONTROL RECORD句に指定した制御レコードに、その図表レコードを定義した帳票定義体名を設定して出力しておかなければなりません。

OPEN文の実行の直後および帳票定義体名として空白を設定した制御レコードを出力した場合、そのページは不定形式ページとなります。ファイル管理記述項のFORMAT句に指定したデータ名に帳票定義体名を設定した図表レコード、または帳票定義体名を設定した制御レコードを出力すると、固定形式ページとなります。

これらのページの形式は、ページの形式を変更する上記のWRITE文を実行しないかぎり、次のページへも引き継がれます。

図表レコードの出力時に、そのページの形式を決定した帳票定義体名から別の帳票定義体名に変更して出力する場合は、改ページされます。

なお、制御レコードの出力の直後のWRITE文には、AFTER ADVANCING PAGEを指定します。

WRITE文のADVANCING指定については、“[8.2.2 プログラムの記述](#)”を参照してください。

```

MOVE 101234 TO 従業員番号 OF 図表レコード(1).
MOVE 105678 TO 従業員番号 OF 図表レコード(2).
MOVE NC"静岡太郎" TO 氏名 OF 図表レコード(1).
MOVE NC"富士通子" TO 氏名 OF 図表レコード(2).
MOVE "MEIBO" TO 帳票定義体名通知域.
MOVE "A" TO 項目群名通知域.
WRITE 図表レコード AFTER ADVANCING PAGE. ... [1]
MOVE SPACE TO 帳票定義体名通知域.
MOVE "1997.07.07" TO 印刷日付 OF 行レコード.
WRITE 行レコード AFTER ADVANCING 3. ... [2]

```

帳票定義体 (MEIBO)

| 従業員番号   | 氏名 |
|---------|----|
|         |    |
|         |    |
| 項目群名: A |    |

#### 【出力結果】

|     |            |      |
|-----|------------|------|
| [1] | 従業員名簿      |      |
|     | 従業員番号      | 氏名   |
|     | 101234     | 静岡太郎 |
|     | 105678     | 富士通子 |
| [2] | 1997.07.07 |      |



#### 注意

改ページ指定(AFTER ADVANCING PAGE指定)がないWRITE文の実行では、印字開始位置は有効になりません。改ページ指定のないWRITE文では、印刷装置の印字可能な先頭行から印字されます。

### 入出力エラー処理

入出力エラーの検出方法および入出力エラーが発生したときの実行結果については、“[7.6 入出力エラー処理](#)”を参照してください。

## 8.4.3 プログラムの翻訳・リンク

### 翻訳

翻訳オプションFORMLIBで、帳票定義体を格納したファイルの格納先を指定します。[参照]“[A.2.19 FORMLIB\(画面帳票定義体ファイルのフォルダの指定\)](#)”

複数の帳票定義体を使用し、その拡張子がそれぞれ異なっている場合、翻訳オプションFORMEXTで、拡張子を指定します。[参照]“[A.2.18 FORMEXT\(画面帳票定義体ファイルの拡張子の指定\)](#)”

### リンク

とくに結合の必要なライブラリはありません。

## 8.4.4 プログラムの実行

印刷ファイルで帳票定義体を使うプログラムを実行するときには、MeFtの使用するプリンタ情報ファイルが必要です。プリンタ情報ファイルの作成については、“[8.5.6 プリンタ情報ファイルの作成](#)”を参照してください。

印刷ファイルで帳票定義体を使うプログラムを実行するときには、以下の環境設定が必要です。

- 環境変数PATHにMeFtの格納されているフォルダを追加しておく必要があります。

- MeFtの使用するプリンタ情報ファイルを作成し、ASSIGN句の記述内容に従い、ファイルの割り当てを行います。ファイルの割り当て方法は、通常のファイルを割り当てるときと同様のため、“7.7.1 ファイルの割り当て”を参照してください。なお、プリンタ情報ファイルの内容および作成方法については、“MeFtのオンラインマニュアル”を参照してください。

## 参考

プリンタ情報ファイルを相対パス名で指定した場合、次の順序で検索されます。

1. 環境変数MEFTDIRに設定したフォルダ
2. カレントフォルダ

印刷ファイルで帳票定義体を使う場合の実行例を以下に示します。

## 例

〔COBOL プログラムのASSIGN句の記述内容〕

```
ASSIGN TO PRTFILE
```

〔実行用の初期化ファイルの内容〕

```
:
PRTFILE=C:¥DIR1¥MEFPRC
:
```

COBOL プログラムのASSIGN句に指定したファイル識別名に、プリンタ情報ファイルを割り当てます。

## 注意

FORMAT句付き印刷ファイルでフォームオーバーレイパターンを使用する場合、フォームオーバーレイパターンが格納されているフォルダのパス名およびファイル名の拡張子は、プリンタ情報ファイルに設定します。環境変数情報FOVLDIRおよびOVD\_SUFFIXに設定しても有効となりません。

## 8.5 表示ファイル(帳票印刷)の使い方

ここでは、表示ファイルを使って、帳票を印刷する方法について説明します。

表示ファイル機能を使って帳票を出力する例題プログラムがサンプルプログラムとして提供されていますので、参考にしてください。

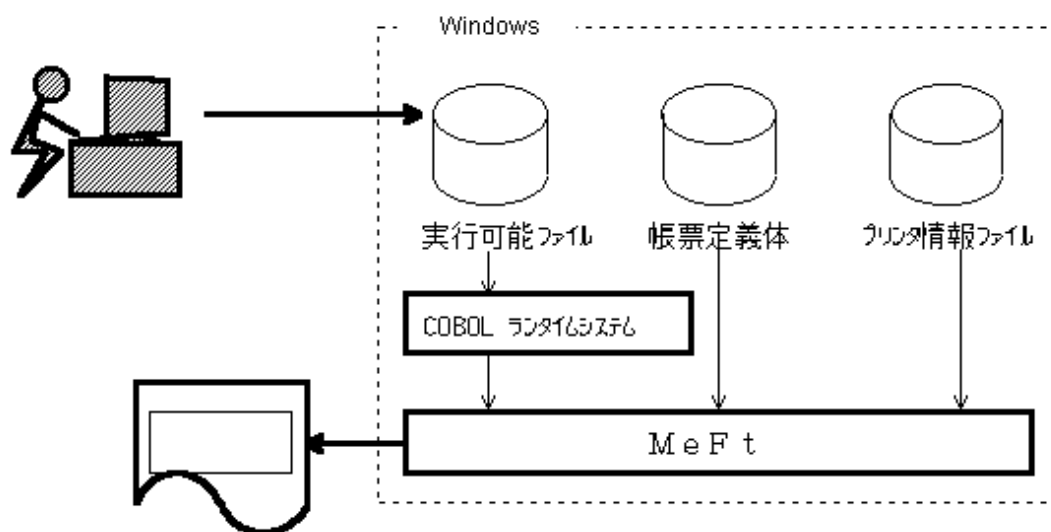
### 8.5.1 概要

表示ファイル機能では、FORMで定義した帳票形式(帳票定義体)を使って帳票印刷を行います。帳票定義体は、FORMを使って画面イメージで簡単に作成することができます。帳票定義体に定義したデータ項目は、COBOLのCOPY文を使って、翻訳時にCOBOLプログラムに取り込むことができます。そのため、帳票印刷のためのデータ項目の定義を、利用者自身がCOBOLプログラムに記述する必要はありません。

表示ファイル機能を実際に使用するときには、帳票定義体およびMeFtが必要です。

これらの関連図を以下に示します。

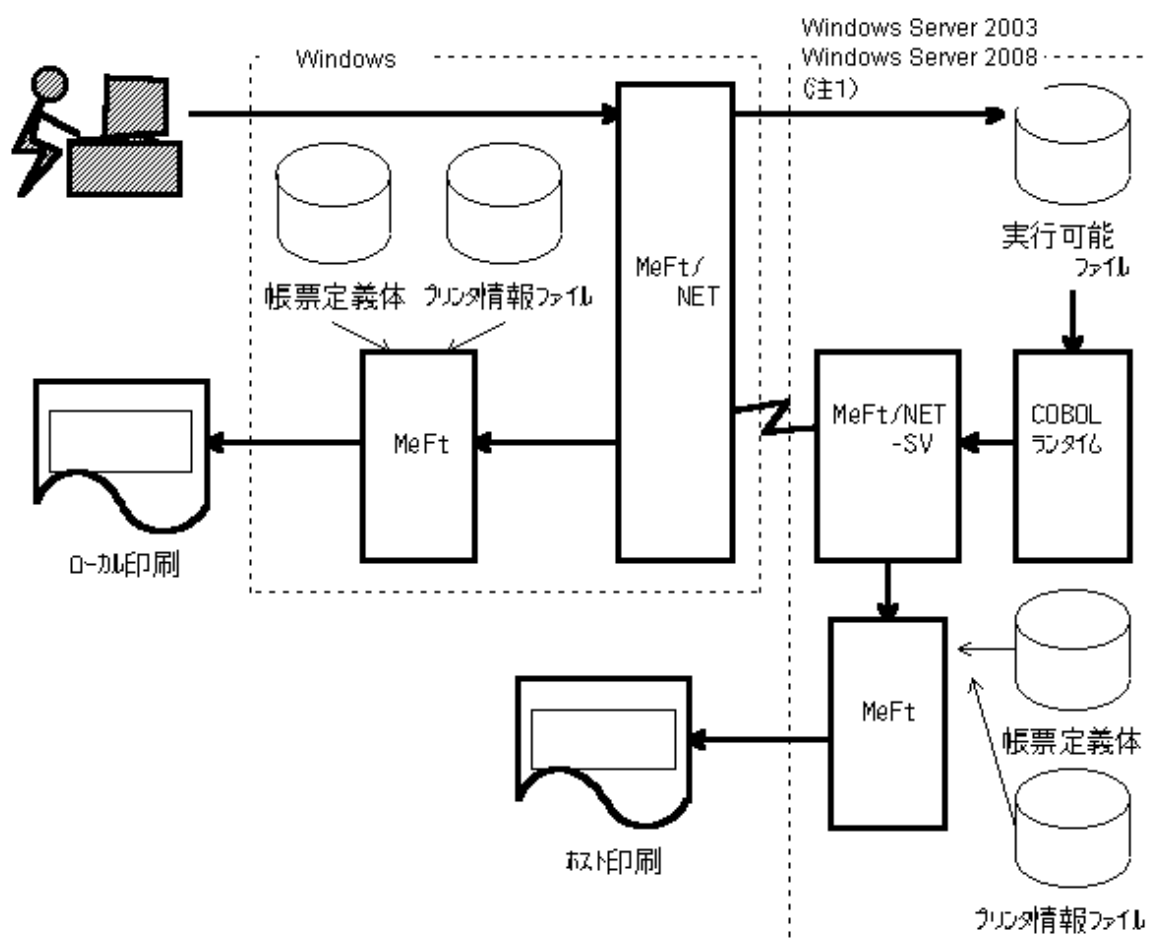
- ・ ローカル環境で使用する場合



- ・ リモート環境で使用する場合

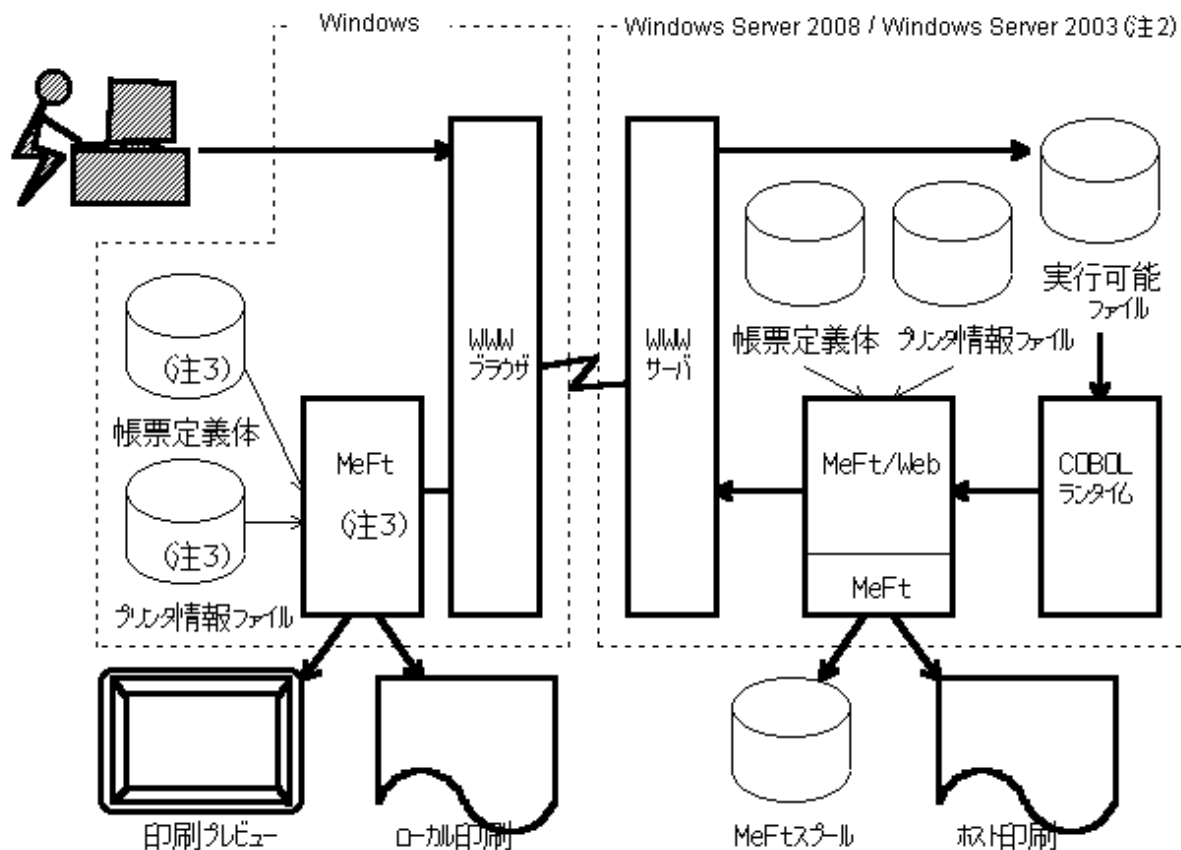
ー MeFt/NET-SV (MeFt/NET連携機能)

アプリケーション起動時に、ローカル印刷とホスト印刷を選択することができます。



## ー MeFt/Web

アプリケーション起動時に、ローカル印刷、ホスト印刷、印刷プレビューまたはスプールのいずれかを選択することができます。



注1 : SolarisまたはHP-UXでも使用できます。ただし、それぞれのシステムに対応したMeFtが必要になります。

注2 : Solaris、Linuxでも使用できます。ただし、そのシステムに対応したMeFtおよびMeFt/Webが必要になります。

注3 : サーバから自動的にダウンロードされます。

MeFt、MeFt/NET、MeFt/NET-SVおよびMeFt/Webが動作可能なオペレーティングシステムについては、各製品のマニュアルおよびソフトウェア説明書を確認してください。

なお、表示ファイル(帳票印刷)では、FORMAT句付き印刷ファイルと同様に帳票の電子化機能を使用することができます。詳細は、“[8.4.1.2 帳票の電子化](#)”を参照してください。

## 8.5.2 作業手順

表示ファイル機能を使って帳票印刷を行うには、帳票定義体、COBOLソースプログラムおよびプリンタ情報ファイルが必要です。帳票定義体およびCOBOLソースプログラムは翻訳時までに、プリンタ情報ファイルは実行時までに作成します。以下に表示ファイル機能を使って帳票印刷を行うときの標準的な作業手順を示します。

1. FORMを使って帳票定義体を作成します。
2. テキストエディタを使ってCOBOLソースプログラムを作成します。
3. COBOLソースプログラムを翻訳・リンクし、実行可能プログラムを作成します。
4. テキストエディタを使ってプリンタ情報ファイルを作成します。
5. 実行可能プログラムを実行します。

## 8.5.3 帳票定義体の作成

ここでは、表示ファイル機能で使用するための帳票定義体を作成するときに設定する情報および注意事項について記述します。FORMの詳しい機能や使用方法については、“FORMヘルプ”を参照してください。

帳票定義体を作成するときに設定する情報を“表8.8 帳票定義体に設定する情報”に示します。

表8.8 帳票定義体に設定する情報

| 情報の種類 |           | 指定する内容および用途                                                              |
|-------|-----------|--------------------------------------------------------------------------|
| 必須    | ファイル名     | 帳票定義体を格納するファイルの名前を指定します。                                                 |
|       | 定義サイズ     | 帳票の大きさを行数と桁数で指定します。                                                      |
|       | 定義体形式     | 形式を指定します。                                                                |
|       | データ項目     | 印刷するデータを設定するためのデータ項目を指定します。ここで指定した項目名は、COBOLプログラムを記述するときにデータ名として使用されます。  |
|       | 項目群       | 1回の印刷処理で印字する1つ以上の項目を1つの項目群としてまとめます。ここで指定した項目群名は、COBOLプログラムを記述するときに使用します。 |
| 任意    | 項目制御部 (注) | COBOL プログラム中で帳票定義体の定義内容を、特殊レジスタを使って変更したい場合、5バイトの項目制御部を指定します。             |

注：項目制御部は、帳票定義体に定義したデータ項目に付加される情報で、入力処理と出力処理で“共用する(3バイト)”と“共用しない(5バイト)”または“なし”の3種類があります。COBOLプログラムで特殊レジスタを使用する場合、“共用しない(5バイト)”を指定する必要があります。



帳票定義体を作成する場合、COBOLプログラムで設定/参照される名前は、以下の注意が必要です。

- ・ 帳票定義体名は8文字以内の半角英数字で指定します。
- ・ 項目群名およびパーティション名は6文字以内の半角英数字で指定します。
- ・ 項目名はCOBOLの利用者語の記述規則に従って指定します。

## 8.5.4 プログラムの記述

ここでは、表示ファイル機能を使って帳票を印刷するときのプログラムの記述内容について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
 SELECT ファイル名
 ASSIGN TO GS-ファイル識別名
 SYMBOLIC DESTINATION IS "PRT"
 FORMAT IS 帳票定義体名通知域
 GROUP IS 項目群名通知域
 [PROCESSING MODE IS 処理種別通知域]
 [UNIT CONTROL IS 特殊制御情報通知域]
 [FILE STATUS IS 入出力状態 1 入出力状態 2].
DATA DIVISION.
FILE SECTION.
 FD ファイル名.
 COPY 帳票定義体名 OF XMDLIB.
(01 表示レコード名.) (注)
(レコード記述項)
WORKING-STORAGE SECTION.
 01 帳票定義体名通知域 PIC X(8).
```

```

01 項目群名通知域 PIC X(8).
[01 処理種別通知域 PIC X(2).]
[01 特殊制御情報通知域 PIC X(6).]
[01 入出力状態 1 PIC X(2).]
[01 入出力状態 2 PIC X(4).]
PROCEDURE DIVISION.
 OPEN OUTPUT ファイル名.
 [MOVE 出力の指定 TO EDIT-MODE OF データ名.]
 [MOVE 強調の指定 TO EDIT-OPTION OF データ名.]
 [MOVE 色 TO EDIT-COLOR OF データ名.]
 [MOVE 背景色の指定 TO EDIT-OPTION2 OF データ名.]
 [MOVE 網がけの指定 TO EDIT-OPTION3 OF データ名.]
 MOVE 帳票定義体名 TO 帳票定義体名通知域.
 MOVE 項目群名 TO 項目群名通知域.
 [MOVE 処理種別 TO 処理種別通知域.]
 [MOVE 制御情報 TO 特殊制御情報通知域.]
 WRITE 表示レコード名.
 CLOSE ファイル名.
END PROGRAM プログラム名.

```

注：()内は、COPY文の展開を表します。

## 環境部(ENVIRONMENT DIVISION)

表示ファイルを定義します。表示ファイルは、通常のファイルを定義するときと同様に、入出力節のファイル管理段落にファイル管理記述項を記述します。ファイル管理記述項に記述する内容を“表8.9 ファイル管理記述項に指定する情報”に示します。なお、これらの情報は、FORMで作成した帳票定義体の定義内容とは関係なく値を決めることができます。

表8.9 ファイル管理記述項に指定する情報

|    | 指定する場所                   | 情報の種類   | 指定する内容および用途                                                                                         |
|----|--------------------------|---------|-----------------------------------------------------------------------------------------------------|
| 必須 | SELECT句                  | ファイル名   | COBOL プログラム中で使用するファイル名を指定します。このファイル名は、COBOL の利用者語の規則に従った名前にします。                                     |
|    | ASSIGN句                  | ファイル参照子 | “GS-ファイル識別名”の形式で指定します。このファイル識別名は、実行時に接続製品が使用するプリンタ情報ファイルのパス名を設定する環境変数情報となります。                       |
|    | FORMAT句                  | データ名    | 作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名には、帳票印刷を行うとき、帳票定義体名を設定します。                             |
|    | GROUP句                   | データ名    | 作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名には、帳票印刷を行うとき、出力の対象となる項目群名を設定します。                       |
|    | SYMBOLIC<br>DESTINATION句 | 出力先の指定  | "PRT" を指定します。                                                                                       |
| 任意 | FILE STATUS句             | データ名    | 作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、入出力処理の実行結果が設定されます。<br>(注1)<br>詳細情報は、4桁の英数字項目を指定します。 |
|    | PROCESSING<br>MODE句      | データ名    | 作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、帳票入出力を行うとき、入出力処理の処理種別を設定します。(注2)                    |
|    | UNIT CONTROL<br>句        | データ名    | 作業場所節または連絡節で、6桁の英数字項目として定義したデータ名を指定します。このデータ名には、印刷処理を行うとき、入出力処理の制御情報を設定します。(注2)                     |

注1：設定される値については、“付録B 入出力状態一覧”を参照してください。

注2：詳細は、“表8.10 入出力処理の種類と指定する値”を参照してください。

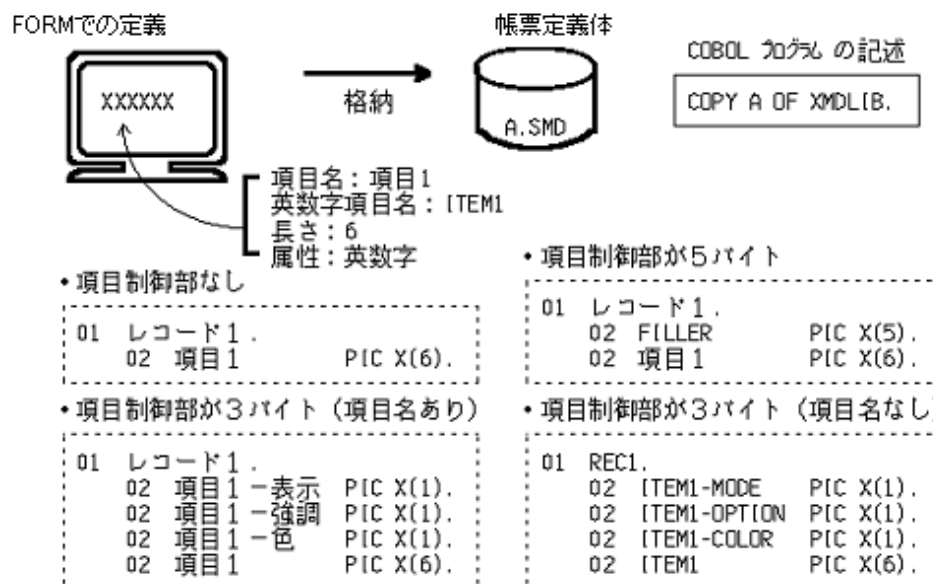
表8.10 入出力処理の種類と指定する値

| 処理種別      | 値    | 制御情報       |        |
|-----------|------|------------|--------|
| プリンタ装置の制御 | "CT" | 改ページ       | "PAGE" |
| パーティション出力 | "PW" | nnn 行改行後出力 | "Annn" |
|           |      | 出力後nnn 行改行 | "Bnnn" |
|           |      | nnn 行位置に出力 | "Pnnn" |
| 行移動出力     | "FW" | 後ろへnnn 行移動 | "Annn" |
|           |      | 前へnnn 行移動  | "Snnn" |

## データ部(DATA DIVISION)

データ部には、表示レコードの定義およびファイル管理記述項に指定したデータ名の定義を記述します。

表示レコードに定義するレコード記述文は、IN/OF XMDLIBを指定したCOPY文を使って帳票定義体から取り込むことができます。展開されるレコードの内容を以下に示します。



注意

表示ファイルに対してEXTERNAL句を指定する場合には、“[10.2.5.2 外部ファイル使用時の注意事項](#)”を必ずお読みください。

## 手続き部(PROCEDURE DIVISION)

帳票の印刷処理には、通常のファイル処理を行うときと同様に、入出力文を使います。

入出力は、以下に示す順序で実行します。

1. OUTPUTまたはI-O指定のOPEN文 : 印刷処理の開始
2. WRITE文 : 帳票の出力
3. CLOSE文 : 印刷処理の終了

### OPEN文およびCLOSE文

OPEN文は印刷処理の開始時に、CLOSE文は印刷処理の終了時に、それぞれ1回だけ実行します。



## WRITE文

1回のWRITE文では、1つの帳票が印刷されます。印刷に使用する帳票定義体の名前は、WRITE文を実行する前に、FORMAT句に指定したデータ名に設定しておく必要があります。

WRITE文では、GROUP句に指定したデータ名に設定されている項目群に属するデータ項目が印刷の対象となります。また、WRITE文の実行前に、特殊レジスタに値を設定することにより、データ項目の属性を変更することもできます。特殊レジスタの使い方については、“[8.1.7 特殊レジスタ](#)”を参照してください。

## 入出力エラー処理

入出力エラーの検出方法および入出力エラーが発生したときの実行結果については、“[7.6 入出力エラー処理](#)”を参照してください。

## 8.5.5 プログラムの翻訳・リンク

### 翻訳

翻訳オプションFORMLIBで、帳票定義体を格納したファイルの格納先を指定します。[参照]“[A.2.19 FORMLIB\(画面帳票定義体ファイルのフォルダの指定\)](#)”

複数の帳票定義体を使用し、その拡張子がそれぞれ異なっている場合、翻訳オプションFORMEXTで、拡張子を指定します。[参照]“[A.2.18 FORMEXT\(画面帳票定義体ファイルの拡張子の指定\)](#)”

### リンク

とくに結合の必要なライブラリはありません。

## 8.5.6 プリンタ情報ファイルの作成

ここでは、表示ファイル機能を使って帳票印刷を行うときのプリンタ情報ファイルに設定する情報および注意事項について記述します。プリンタ情報ファイルの詳しい内容や作成方法については、“[MeFtのオンラインマニュアル](#)”を参照してください。

プリンタ情報ファイルに設定する代表的な情報を“[表8.11 プリンタ情報ファイルの設定情報](#)”に示します。

表8.11 プリンタ情報ファイルの設定情報

| 情報の種類  | 指定する内容および用途                                               |
|--------|-----------------------------------------------------------|
| PRTDRV | 出力するプリンタ装置のデバイス名を指定します。                                   |
| MEDDIR | 帳票定義体を格納したフォルダのパス名を設定します。                                 |
| MEDSUF | 帳票定義体を格納したファイルの拡張子を指定します。<br>省略した場合の拡張子はMeFtの定めた省略値となります。 |

## 8.5.7 プログラムの実行

表示ファイル機能を使った帳票印刷を行うプログラムを実行するときには、以下の環境設定が必要です。

### MeFtを使用する場合

- 環境変数PATHにMeFtの格納されているフォルダを追加しておく必要があります。
- ファイル識別名を環境変数情報名として、プリンタ情報ファイル名を設定します。



### 参考

プリンタ情報ファイルを相対パス名で指定した場合、次の順序で検索されます。

1. 環境変数MEFTDIRに設定したフォルダ
2. カレントフォルダ

#### MeFt/NET-SV(MeFt/NET連携機能)を使用する場合

- 環境変数PATHにMeFt/NET-SVの格納されているフォルダを追加しておく必要があります。
- ファイル識別名を環境変数情報名として、プリンタ情報ファイル名を設定します。実際のプリンタ情報ファイルは、クライアント側に用意しておく必要があります。
- 接続製品としてMeFt/NETを使用することを示す環境変数情報を指定する必要があります。指定方法には、以下の2つの方法があります。
  1. ファイル識別名を環境変数情報名として、接続製品名にMEFTNETを設定する。[参照]“[5.4.1.70 ファイル識別名 \(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\)](#)”
  2. 環境変数情報@CBR\_PSFILE\_PRTにMEFTNETを設定する。[参照]“[5.4.1.43 @CBR\\_PSFILE\\_xxx \(表示ファイルから使用する接続製品名 \(宛先ごとの指定\)\)](#)”

MeFt/NET-SVの詳細は、“MeFt/NET-SVのヘルプ”を参照してください。

表示ファイル機能を使って、帳票印刷を行う場合の実行例を以下に示します。



#### 例

- COBOL プログラムのASSIGN句の記述内容

```
ASSIGN TO GS-PRTFILE
```

- 実行用の初期化ファイルの内容

- MeFtを使用する場合

```
:
PRTFILE=MEFPRC
:
```

- MeFt/NET-SV (MeFt/NET連携機能)を使用する場合

```
:
PRTFILE=MEFPRC, MEFTNET
:
```

COBOL プログラムのASSIGN句に指定したファイル識別名に、プリンタ情報ファイルを割り当てます。MeFtを使用する場合の例では、プリンタ情報ファイル名を相対指定で割り当てたので、プリンタ情報ファイルは環境変数MEFTDIR に設定したフォルダから検索されます。

#### MeFt/Webを使用する場合

MeFt/Webを使用する場合には、“MeFt/Web 説明書”を参照してください。

## 8.6 電子帳票出力機能を使う方法

ここでは、電子帳票出力機能の概要とFORMAT句なし印刷ファイルを使用して帳票を電子化する方法およびその注意事項について説明します。FORMAT句付き印刷ファイルおよび表示ファイル(帳票印刷)を使用した電子帳票出力機能については、“[8.4.1.2 帳票の電子化](#)”および“MeFtのオンラインマニュアル”を参照してください。

### 8.6.1 電子帳票出力機能の概要

COBOLでは、従来よりFORMAT句なし印刷ファイル、FORMAT句付き印刷ファイルおよび表示ファイル(宛て先PRT)機能を使用して帳票をプリンタ(紙)に印刷する機能を提供していました。これに加え、ListWORKSと連携することにより、従来のアプリケーションを変更することなく簡単な環境定義(印刷情報ファイルまたはプリンタ情報ファイル)だけで、帳票を電子化することが可能となります。

## ListWORKSとは

ListWORKSとは、紙に出力して利用している帳票を電子化し、コンピュータの画面で参照することにより、帳票のデータを活用するシステムです。これにより、経費削減、情報提供のスピード化、信頼性の向上およびデータの有効利用を図ることが可能となります。

## 電子帳票ファイルの有効活用

ListWORKS連携により電子化した帳票を、紙に出力したイメージでコンピュータの画面にそのまま表示できます。電子化された帳票に対して、ListWORKSが提供する機能を利用することで以下の効果があります。

- ・ 帳票を画面に表示して、付せん、メモ、チェックマークなどを記入できます。なお、電子化された帳票は、必要に応じていつでもプリンタ(紙)に印刷することもできます。
- ・ 帳票は電子データであるため、紙より、迅速で正確にデータを検索できます。
- ・ 表計算ソフト、ワープロソフトなど、ほかのアプリケーションとデータ連携できます。データの再入力をなくし、帳票業務の効率化を図ることができます。
- ・ 電子化された帳票を、FAXや電子メールに添付して配信することができます。
- ・ 紙での運用と比べて、仕分けや配送にかかっていた人件費や用紙コスト、保管費用などのコストを削減することができます。

なお、電子帳票およびListWORKSに関する詳細は、“ListWORKSのオンラインマニュアル”および“ListWORKSのヘルプ”を参照してください。

## 8.6.2 帳票を電子化する方法

ここでは、FORMAT句なし印刷ファイルを使用して帳票を電子化する方法について説明します。電子帳票出力機能を利用する場合、ListWORKSが必要となります。

### プログラムの記述

#### 既存アプリケーションを利用する場合

プログラム修正および再翻訳は一切必要ありません。既存アプリケーションに対して後述の環境定義(印刷情報ファイル)の指定を追加するだけで帳票を電子化することができます。

#### 新規アプリケーションを作成する場合

帳票を電子化するための特別な記述は一切必要ありません。通常のプリンタ出力(紙)の場合と同様に、FORMAT句なし印刷ファイルの言語仕様に従ったプログラムを記述し、このアプリケーションに対する環境定義(印刷情報ファイル)の指定を追加するだけで帳票を電子化することができます。

### 印刷情報ファイル名の指定

印刷情報ファイル名の指定方法は、ASSIGN句の指定によって異なります。

#### ASSIGN句にファイル識別名を指定した場合

“PRTNAME:プリンタ名”の形式でListWORKSの電子保存装置名を指定した後に、„INF(印刷情報ファイルの絶対パス名)を指定します。指定方法の詳細は、“[5.4.1.72 ファイル識別名 \(プログラムで使用するプリンタおよび各種パラメタの指定\)](#)”を参照してください。



例

ファイル識別名“PRTFILE”に仮想プリンタ名“FUJITSU ListWORKS Storage”を割り当てる場合

```
[COBOL ソースプログラムのASSIGN句の記述]
SELECT 印刷ファイル ASSIGN TO PRTFILE
[環境変数情報]
PRTFILE=PRTNAME:FUJITSU ListWORKS Storage, INF (C:¥PRTAPL¥PRINTINF.TXT)
```

ファイル識別名“PRTFILE”に対するプリンタの割当てを省略した場合

```
[COBOL ソースプログラムのASSIGN句の記述]
SELECT 印刷ファイル ASSIGN TO PRTFILE
```

[環境変数情報]  
PRTFILE=, , INF (C:¥PRTAPL¥PRINTINF. TXT)



## 注意

プリンタの割当てを省略した場合、印刷情報ファイルのPRTOUT指定でプリンタの割当てを行う必要があります。

### ASSIGN句にPRINTER-nを記述した場合

印刷情報ファイル名の指定方法は、ファイル識別名の場合と同じです。



## 例

#### “PRINTER-1”に仮想プリンタ名“FUJITSU ListWORKS Storage”を割り当てる場合

[COBOL ソースプログラムのASSIGN句の記述]  
SELECT 印刷ファイル ASSIGN TO PRINTER-1  
[環境変数情報]  
PRINTER-1=PRTNAME:FUJITSU ListWORKS Storage, , INF (C:¥PRTAPL¥PRINTINF. TXT)

#### “PRINTER-2”に対するプリンタの割当てを省略した場合

[COBOL ソースプログラムのASSIGN句の記述]  
SELECT 印刷ファイル ASSIGN TO PRINTER-2  
[環境変数情報]  
PRINTER-2=, , INF (C:¥PRTAPL¥PRINTINF. TXT)



## 注意

プリンタの割当てを省略した場合、印刷情報ファイルのPRTOUT指定でプリンタの割当てを行う必要があります。

### ASSIGN句にデータ名を記述した場合

データ名領域に対して、“PRTNAME:プリンタ名”の形式でListWORKSの電子保存装置名を指定した後に、, , INF(印刷情報ファイルの絶対パス名)を指定したデータを設定します



## 例

#### データ名“PRTDATA”に仮想プリンタ名“FUJITSU ListWORKS Storage”を割り当てる場合

[COBOL ソースプログラムのASSIGN句の記述]  
SELECT 印刷ファイル ASSIGN TO PRTDATA  
[COBOL ソースプログラムのデータ記述項]  
01 PRTDATA PIC X(128).  
[COBOL ソースプログラムの手続き部]  
MOVE “PRTNAME:FUJITSU ListWORKS Storage, , INF (C:¥PRTAPL¥PRINTINF. TXT)”  
TO PRTDATA

#### データ名“PRTDATA”に対するプリンタの割当てを省略した場合

[COBOL ソースプログラムのASSIGN句の記述]  
SELECT 印刷ファイル ASSIGN TO PRTDATA  
[COBOL ソースプログラムのデータ記述項]  
01 PRTDATA PIC X(128).  
[COBOL ソースプログラムの手続き部]  
MOVE “, , INF (C:¥PRTAPL¥PRINTINF. TXT)” TO PRTDATA



## 注意

プリンタの割当てを省略した場合、印刷情報ファイルのPRTOUT指定でプリンタの割当てを行う必要があります。

### ASSIGN句にファイル識別名定数を記述した場合

プログラム中に定数で、“PRTNAME:プリンタ名”の形式でListWORKSの電子保存装置名を指定した後に、“INF(印刷情報ファイルの絶対パス名)”を記述します。



## 例

### 仮想プリンタ名“FUJITSU ListWORKS Storage”を割り当てる場合

```
[COBOL ソースプログラムのASSIGN句の記述]
SELECT 印刷ファイル ASSIGN
 TO "PRTNAME:FUJITSU ListWORKS Storage, , INF(C:¥PRTAPL¥PRINTINF. TXT)"
```

### プリンタの割当てを省略した場合

```
[COBOL ソースプログラムのASSIGN句の記述]
SELECT 印刷ファイル ASSIGN TO ", , INF(C:¥PRTAPL¥PRINTINF. TXT)"
```



## 注意

プリンタの割当てを省略した場合、印刷情報ファイルのPRTOUT指定でプリンタの割当てを行う必要があります。

### ASSIGN句にPRINTERを記述した場合

環境変数情報@CBR\_PrintInfoFileに印刷情報ファイルの絶対パス名を指定します。[参照]“5.4.1.41 @CBR\_PrintInfoFile (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)”



## 例

```
[COBOL ソースプログラムのASSIGN句の記述]
SELECT 印刷ファイル ASSIGN TO PRINTER
[環境変数情報]
@CBR_PrintInfoFile=C:¥PRTAPL¥PRINTINF. TXT
```



## 注意

電子帳票出力機能を利用する場合、ASSIGN TO PRINTER(通常使用するプリンタの設定)は意味を持ちません。印刷情報ファイルのPRTOUT指定に指定されたプリンタ名が有効となります。

### 印刷情報ファイルの定義

FORMAT句なし印刷ファイルの出力先をListWORKSに向ける(帳票を電子化する)ために、印刷情報ファイルに以下のパラメタを設定します。

STREAM=LW

帳票を電子化することを意味します。STREAM指定の詳細は、“8.1.9 印刷情報ファイル”を参照してください。

## STREAMENV=電子帳票情報ファイル名

電子帳票情報ファイルは、帳票を電子化する際に、ListWORKSが使用する電子帳票に関するさまざまな定義情報を含むファイルです。STREAMENV指定の詳細は、“[8.1.9 印刷情報ファイル](#)”を、電子帳票情報ファイルの詳細は、“ListWORKSのオンラインマニュアル”および“ListWORKSのヘルプ”を参照してください。

## PRTOUT=PRTNAME:プリンタ名

プリンタ名にはListWORKSの電子保存装置名を指定します。ListWORKSの仮想プリンタを使用する場合、ListWORKSのインストール時に特に変更しない時の仮想プリンタ名は、“FUJITSU ListWORKS Storage”となります。

PRTOUT指定は、省略することもできます。PRTOUT指定を省略した場合、ASSIGN句の記述形式に従って、ListWORKSの電子保存装置名の指定が必要になります。両方指定された場合は、PRTOUTの指定が優先されます。PRTOUT指定の詳細は、“[8.1.9 印刷情報ファイル](#)”を、環境変数情報の指定の詳細は“[5.4.1.72 ファイル識別名 \(プログラムで使用するプリンタおよび各種パラメタの指定\)](#)”を参照してください。

## ListWORKSの準備・設定

ListWORKSのインストールおよび環境設定を行います。これらの詳細は、“ListWORKSのオンラインマニュアル”および“ListWORKSのヘルプ”を参照してください。

## 既存の帳票アプリケーションを電子化する場合の指定例



例

### 既存のプリンタ(紙)出力アプリケーションの指定例

```
[COBOL ソースプログラムのASSIGN句の記述]
SELECT 印刷ファイル ASSIGN TO S-PRTFILE
[環境変数情報]
PRTFILE=PRTNAME:FUJITSU FMLBP
[印刷情報ファイルの記述]
なし
```

### 上記既存アプリケーションを電子化する指定例

```
[COBOL ソースプログラムのASSIGN句の記述]
SELECT 印刷ファイル ASSIGN TO S-PRTFILE
[環境変数情報]
PRTFILE=PRTNAME:FUJITSU FMLBP, INF(C:¥PRTAPL¥PRINTINF.TXT)
[印刷情報ファイル(C:¥PRTAPL¥PRINTINF.TXT)の内容]
STREAM=LW
STREAMENV=C:¥LW.TXT
PRTOUT=PRTNAME:FUJITSU ListWORKS Storage
```

### [解説]

印刷情報ファイルに、STREAM=LWを指定することで、電子帳票出力を行うことが決定付けられます。STREAMENVに指定された電子帳票情報ファイルLW.TXTは、ListWORKSに渡され帳票を電子化する際のさまざまな定義情報として利用されます。PRTOUTにListWORKSの仮想プリンタ名を指定することで、ファイル識別名に指定されたFUJITSU FMLBPが無視され、帳票の出力先がFUJITSU ListWORKS Storageに向けられます。帳票の電子化に必要な定義はこれだけです。





|  | 指定内容              | プリンタ出力時の指定 | 電子帳票出力時の指定 | 電子帳票出力時のみなし処理または代替処理            |
|--|-------------------|------------|------------|---------------------------------|
|  | @CBR_TextAlign    | ○          | ×          | 常に“BOTTOM”が指定された場合と同じ状態になります。   |
|  | @DefaultFCB_Name  | ○          | ○          |                                 |
|  | @PrinterFontName  | ○          | ○          |                                 |
|  | @PRN_FormName_xxx | ○          | ○          |                                 |
|  | ファイル識別名 (注)       | ○          | ○          |                                 |
|  | FCBxxxx           | ○          | ○          |                                 |
|  | FOVLDIR           | ○          | ○          |                                 |
|  | FOVLTYPE          | ○          | ○          |                                 |
|  | FOVLNAME          | ○          | ×          | 電子帳票情報ファイルのOVL_NAMEキーに指定してください。 |
|  | OVD_SUFFIX        | ○          | ○          |                                 |

注：ファイル識別名 (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)

## 印刷情報ファイルを利用する機能

印刷情報ファイルの詳細は、“[8.1.9 印刷情報ファイル](#)”を参照してください。

|          | 指定内容               | プリンタ出力時の指定 | 電子帳票出力時の指定 | 電子帳票出力時のみなし処理または代替処理            |
|----------|--------------------|------------|------------|---------------------------------|
| 印刷情報ファイル | TextAlign          | ○          | ×          | 常に“BOTTOM”が指定された場合と同じ状態になります。   |
|          | DocumentName       | ○          | ○          |                                 |
|          | OverlayPrintSPEC   | ○          | ×          | 常に“COBOL”が指定された場合と同じ状態になります。    |
|          | OverlayPrintOffset | ○          | ×          | 常に“VALID”が指定された場合と同じ状態になります。    |
|          | STREAM (*1)        | ○          | ○ 必須       |                                 |
|          | STREAMENV          | ×          | ○          |                                 |
|          | PRTOUT (*3)        | ○          | ○          |                                 |
|          | FOVLDIR            | ○          | ×          | 電子帳票情報ファイルのOVL_NAMEキーに指定してください。 |
|          | FOVLTYPE           | ○          | ×          |                                 |
|          | FOVLNAME           | ○          | ×          |                                 |
|          | OVD_SUFFIX         | ○          | ×          |                                 |

\*1：電子帳票出力時は、常に“LW”を指定しなければなりません。プリンタ出力時は“PR”を指定するか、指定を省略します。

\*2：プリンタ出力時は、指定しても無視されます。

\*3：ファイル識別名のプリンタの割当てを変更する場合またはファイル識別名のプリンタの割当てを省略する場合に指定します。プリンタ出力時は、通常のプリンタに割り当てられたローカルプリンタポート名(LPTn:)/通信ポート名(COMn:)/プリンタ名(PRTNAME:)を指定し、電子帳票出力時は、ListWORKSの電子保存装置名を“PRTNAME:プリンタ名”の形式で指定します。

## プログラムの指定関連

ページ属性に関する機能



| COBOLの機能                        | プリンタ  | ListWORKS |
|---------------------------------|-------|-----------|
| フォームオーバーレイ                      | ○ [a] | ○ [a]     |
| フォームオーバーレイ焼き付け回数 1～255          | ○ [b] | ○ [b]     |
| 複写数 1～255                       | ○     | ○         |
| FCB 6/8/12LPI                   | ○     | ○         |
| フォーマット定義体                       | —     | —         |
| 複写修正モジュール名                      | □ [c] | □ [c]     |
| 複写修正開始番号 1～255                  | □ [c] | □ [c]     |
| 複写修正文字配列テーブル番号 0～3              | □ [c] | □ [c]     |
| 用紙識別名                           | □ [c] | □ [c]     |
| 文字配列テーブル／追加文字セット                | □ [c] | □ [c]     |
| ダイナミックロード                       | □ [c] | □ [c]     |
| オフセットスタック                       | □ [c] | □ [c]     |
| 印刷形式 P/PZOOM/L/LZOOM/LP         | ○     | ○ [d]     |
| 用紙サイズ A3/A4/A5/B4/B5/LETTER/ 任意 | ○     | ○ [e]     |
| 用紙供給口                           | ○     | — [f]     |
| 用紙排出口                           | —     | — [f]     |
| 印刷面片面/ 両面                       | ○     | — [f]     |
| 印刷面位置付け表面/ 裏面                   | ○     | — [f]     |
| 印字禁止領域                          | ○     | ○         |
| とじしろ方向                          | ○     | ○ [g]     |
| とじしろ幅 0～9999:1/1440 インチ単位       | ○     | ○ [g]     |
| 印刷原点位置                          | ○     | ○ [g]     |
| 文書名                             | ○     | ○ [h]     |

□：指定しても意味を持たない機能

#### [a] フォームオーバーレイ指定

プリンタ出力および電子帳票出力のどちらの場合でも、単一オーバーレイだけサポートしています。

#### [b] フォームオーバーレイ焼き付け回数

プリンタ出力および電子帳票出力のどちらの場合でも、焼き付け回数は複写数と同じ値が採用されます。

#### [c] OSIV系システム固有またはプリンタ固有情報

OSIV系システム固有機能またはプリンタ固有情報であるため、プリンタ出力および電子帳票出力のどちらの場合も、意味を持たない指定です。

#### [d] 印刷形式

電子帳票出力時は、PZOOM/LZOOM/LP縮刷は有効になりません。PZOOMはポートレートモード、LZOOMおよびLP縮刷はランドスケープモードとみなされます。また、電子帳票出力時は1つのファイル内で複数の印刷形式をページ単位で混在させることができません。この場合、最初に指定された印刷形式が後続のページに対しても引き継がれます。

#### [e] 用紙サイズ指定

電子帳票出力時は1つのファイル内で複数の用紙サイズをページ単位で混在させることができません。この場合、最初に指定された用紙サイズが後続のページに対しても引き継がれます。

ListWORKSを利用してFAX送付を行う場合、Systemwalker/PrintMGRまたはInterstage Print Manager(以降PrintMGR)のFAX連携機能と連携する必要があります。PrintMGRのFAX連携機能で利用できる用紙サイズについては、“PrintMGRのオンラインマニュアル”を参照してください。

#### [f] 電子帳票では意味のない機能

これらの機能は、プリンタ(紙)に出力する場合に意味を持つ機能です。電子帳票出力時は、これらの指定は無視されます。

#### [g] とじしろ方向・とじしろ幅・印刷原点指定

電子帳票出力時の画面表示では、これらの指定は無視されます。本機能は、紙に印刷される時の概念であるため電子帳票を画面に表示する際は意味がありません。ただし、一度電子化された帳票を後で紙に印刷する場合には有効となります。

#### [h] 文書名指定

電子帳票出力時は、I制御レコードの“DOC-INFO”および環境変数情報@CBR\_DocumentName\_xxxで指定した文書名は無視されます。印刷情報ファイルのDocumentName指定を使用してください。

#### 文字属性

| COBOLの機能                            |                         | プリンタ | ListWORKS |
|-------------------------------------|-------------------------|------|-----------|
| 文字サイズ 3.0 ～ 300.0 ポ : 0.1ポ単位        |                         | ○    | ○         |
| 文字ピッチ 0.01 ～ 24.00 CPI : 0.01CPI単位  |                         | ○    | ○         |
| 書体                                  | 書体名                     | ○    | ○         |
|                                     | 書体番号 FONT-001 ～ FONT999 | ○    | ○         |
| 文字回転                                | 横書き                     | ○    | ○         |
|                                     | 縦書き (反時計回り90度)          | ○    | ○         |
| 文字形態                                | 全角／半角／平体／長体／倍角          | ○    | ○         |
|                                     | ボールド・イタリック              | ○    | ○         |
| 水平スキップ 0.01 ～ 24.00 CPI : 0.01CPI単位 |                         | ○    | ○         |

文字属性に関する機能差は、特にありません

#### その他(ListWORKSの制限事項)

ListWORKSの制限によりCOBOLの一部の機能が利用できないことがあります。ListWORKSの制限事項およびその解除に関する詳細は、“ListWORKSのオンラインマニュアル”および“ListWORKSのヘルプ”を参照してください。

### 8.6.5 実行時エラーについて

ListWORKSランタイムライブラリでエラーが発生すると、以下の実行時メッセージが出力されます。

JMP0362I-U ' \$2' ファイルに対する' \$1' 文の実行で、レコード生成処理のエラーが発生しました。 CODE=\$3. \$4

ListWORKSランタイムライブラリから通知されるエラー状態は、上記メッセージの\$3に示されます。“ListWORKSのヘルプ”を参照し、“詳細コード”の“ListWORKSランタイムライブラリの内部コード”から\$3に示されるエラーコードの意味を調べ、エラーの原因を取り除いた後、再度実行してください。

#### 参考

以下に、“ListWORKSのヘルプ”に記載されていないエラーコードについて示します。

| CODE | 意味                                                                | 対処                                                                |
|------|-------------------------------------------------------------------|-------------------------------------------------------------------|
| 1052 | COBOLプログラムの翻訳時に指定されたコード系(翻訳オプションRCS)が、現在使用しているListWORKSでは未サポートです。 | 現在使用しているListWORKSがサポートするコード系を調べ、COBOLプログラムの翻訳時に指定するコード系を変更してください。 |
| 2000 | WRITE文に記述したレコードにおいて、同一基本項目内に1バイトコード系と2バイトコード系を混在してはなりません。         | エラーが発生したWRITE文を調べ同一基本項目内に1バイトコード系と2バイトコード系が混在しないように修正してください。      |

## 第9章 画面を使った入出力

本章では、画面を表示したり、表示した画面からデータを入力したりする方法について説明します。

### 9.1 画面を使った入出力の種類

COBOLプログラムでは、ディスプレイ装置に画面を表示し、表示した画面からデータを入力する(以降の説明では、単に画面入出力といえます)ことができます。画面入出力を行う機能には、次の2種類があります。

- ・ 表示ファイル機能
- ・ スクリーン操作機能

それぞれの機能の特徴および用途を下表に示します。

表9.1 表示ファイル機能とスクリーン操作機能の特徴・用途

| 特徴・用途 |                   | 表示ファイル機能                                                                                                                                                          | スクリーン操作機能                                                                                    |
|-------|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| 特徴    | 画面の設計             | 画面イメージで設計 (FORMを使用)                                                                                                                                               | 画面を行の集まりとして設計                                                                                |
|       | 画面の数              | プログラムに定義した表示ファイルの数                                                                                                                                                | 1つ                                                                                           |
|       | プログラム実行中の画面の属性の変更 | 可能                                                                                                                                                                | 可能                                                                                           |
|       | プログラムの記述内容        | <ul style="list-style-type: none"><li>・ 表示ファイルの定義</li><li>・ 表示レコードの定義</li><li>・ OPEN文</li><li>・ READ文</li><li>・ WRITE文</li><li>・ CLOSE文</li></ul>                 | <ul style="list-style-type: none"><li>・ 画面の定義</li><li>・ ACCEPT文</li><li>・ DISPLAY文</li></ul> |
|       | 関連製品              | <ul style="list-style-type: none"><li>・ FORM (画面の定義)</li><li>・ MeFt (ローカル入出力処理)</li><li>・ MeFt/NET-SV (リモート入出力処理)</li><li>・ MeFt/Web (Web環境でのリモート入出力処理)</li></ul> | なし                                                                                           |
| 用途    |                   | 複雑な画面を使って画面入出力を行いたい場合 (帳票や伝票の形式など)                                                                                                                                | 簡単な画面を使って画面入出力を行いたい場合                                                                        |

### 9.2 表示ファイル(画面入出力)の使い方

ここでは、画面入出力を行う表示ファイル機能の概要、動作環境、画面定義体の作成、COBOLソースプログラムの作成および注意事項について説明します。

なお、表示ファイル機能を使って画面入出力を行う例題プログラムがサンプルプログラムとして提供されていますので、参考にしてください。

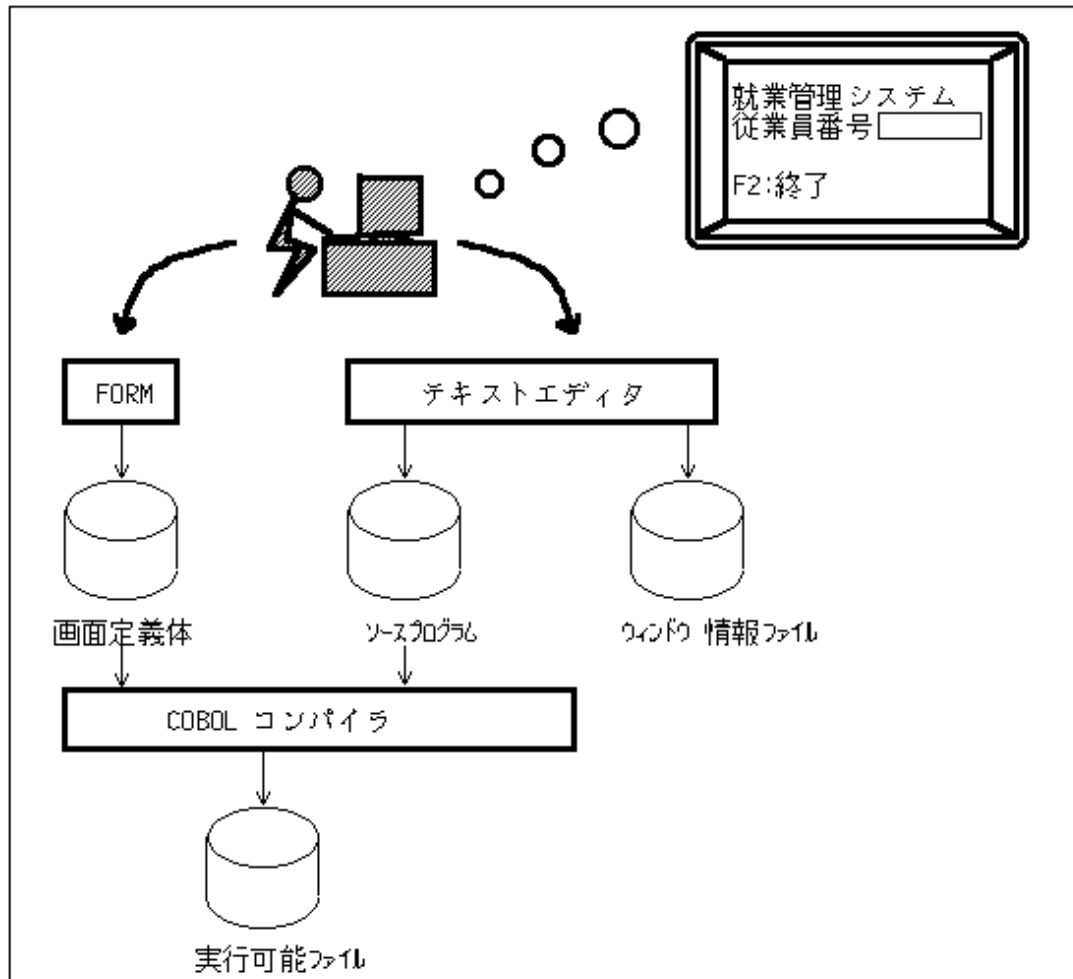
#### 9.2.1 概要

表示ファイル機能では、FORMで定義した画面(画面定義体)を使って画面入出力を行います。画面定義体は、FORMを使って画面イメージで簡単に作成することができます。画面定義体に定義した入出力処理のためのデータ項目は、COBOLのCOPY文を使って、翻訳時にCOBOLプログラムに取り込むことができます。そのため、入出力処理のためのデータ項目の定義を、利用者自身がCOBOLプログラムに記述する必要はありません。また、画面定義体で定義されている出力データの属性を、COBOLの特殊レジスタを使って、プログラムの実行中に変更することができます。

## 9.2.2 動作環境

表示ファイル(画面入出力)機能を使用するときには、FORMで定義した画面定義体が必要です。COBOLプログラムの実行可能ファイルを作成するまでの関連図を以下に示します。

表示ファイル機能を使用するプログラム作成時の関連図

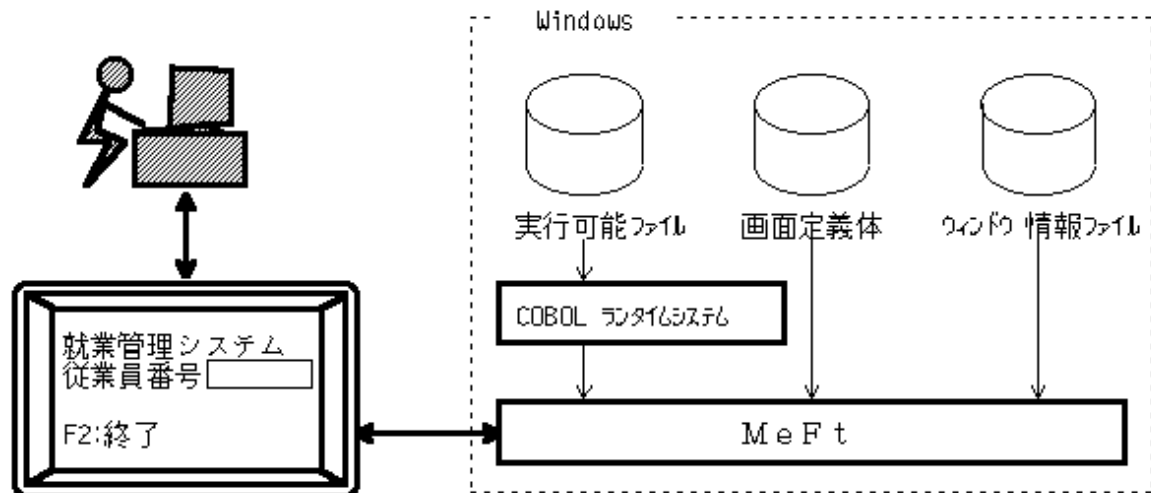


表示ファイル機能を実際に使用するときには、画面定義体およびMeFtが必要です。

これらの関連図を以下に示します。

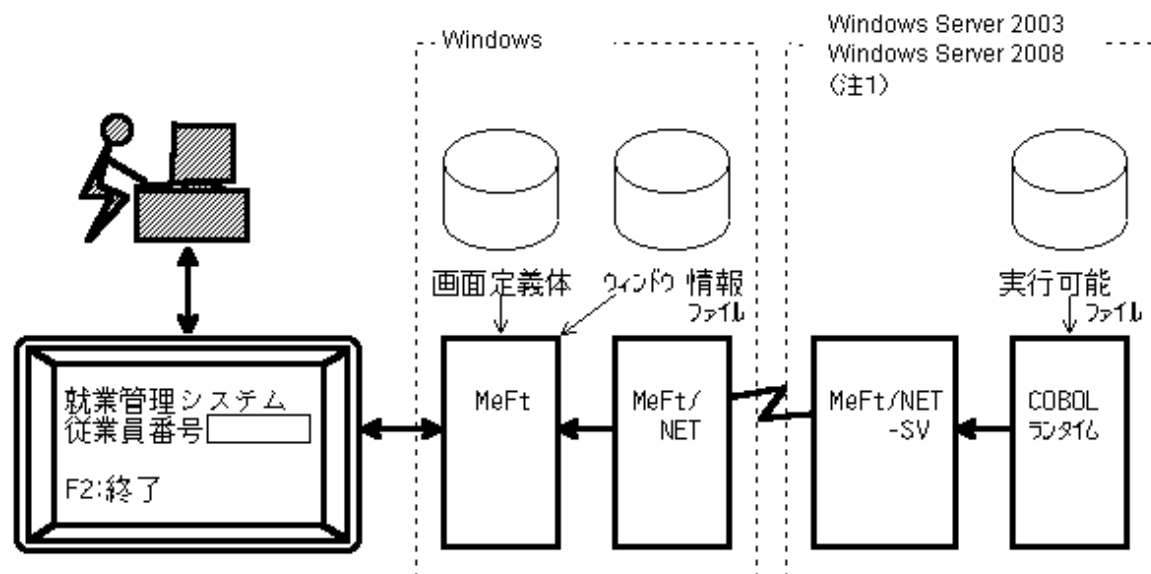
表示ファイル機能を使用するプログラム運用時の関連図

ローカル環境で使用する場合

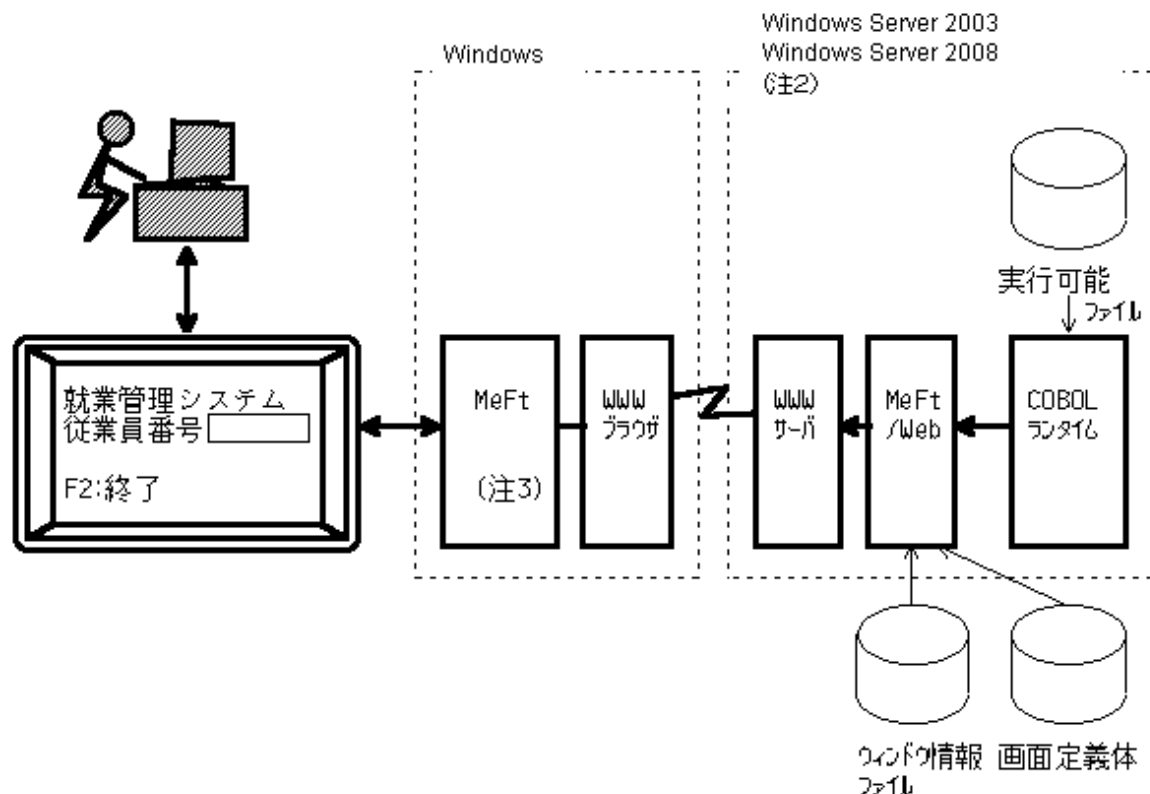


リモート環境で使用する場合

MeFt/NET-SV (MeFt/NET連携機能)



MeFt/Web



注1：UXP/DS、SolarisまたはHP-UXでも使用できます。ただし、MeFtが必要になります。

注2：UXP/DSまたはSolaris、Linuxでも使用できます。ただし、MeFtおよびMeFt/Webが必要になります。

注3：サーバから自動ダウンロードされます。

MeFt、MeFt/NET、MeFt/NET-SVおよびMeFt/Webが動作可能なオペレーティングシステムについては、各製品のマニュアルおよびソフトウェア説明書を確認してください。

### 9.2.3 作業手順

表示ファイル機能を使って画面入出力を行うには、画面定義体、COBOLソースプログラムおよびウィンドウ情報ファイルが必要です。画面定義体およびCOBOLソースプログラムは翻訳時まで、ウィンドウ情報ファイルは実行時まで作成します。以下に表示ファイル機能を使って画面入出力を行うときの標準的な作業手順を示します。

1. FORMを使って画面定義体を作成します。
2. テキストエディタを使ってCOBOLソースプログラムを作成します。
3. COBOLソースプログラムを翻訳・リンクし、実行可能プログラムを作成します。
4. テキストエディタを使ってウィンドウ情報ファイルを作成します。
5. 実行可能プログラムを実行します。

### 9.2.4 画面定義体の作成

ここでは、表示ファイル機能で使用するための画面定義体を作成するときに設定する情報および注意事項について記述します。FORMの詳しい機能や使用方法については、“FORMのヘルプ”を参照してください。

画面定義体を作成するときに設定する情報を“表9.2 画面定義体に設定する情報”に示します。

表9.2 画面定義体に設定する情報

| 情報の種類 |           | 指定する内容および用途                                                                        |
|-------|-----------|------------------------------------------------------------------------------------|
| 必須    | ファイル名     | 画面定義体を格納するファイルの名前を指定します。                                                           |
|       | 定義サイズ     | 表示する画面の大きさを行数と桁数で指定します。                                                            |
|       | 定義体形式     | 自由形式を指定します。                                                                        |
|       | データ項目     | 画面入出力を行うための領域を指定します。ここで指定した項目名は、COBOL プログラムを記述するときにデータ名として使用されます。                  |
|       | 項目群       | 1回の入出力処理で表示またはデータ入力する1つ以上の項目を1つの項目群としてまとめます。ここで指定した項目群名は、COBOL プログラムを記述するときに使用します。 |
| 任意    | 項目制御部 (注) | COBOL プログラム中で画面定義体の定義内容を変更したい場合、5バイトの項目制御部を指定します。                                  |
|       | アテンション情報  | COBOL プログラム中で入力キーを判定する場合、指定します。                                                    |

注：項目制御部は、画面定義体に定義したデータ項目に付加される情報で、入力処理と出力処理で“共用する(3バイト)”と“共用しない(5バイト)”または“なし”の3種類があります。COBOLプログラムで特殊レジスタを使用する場合、“共用しない(5バイト)”を指定することをおすすめします。

## 9.2.5 プログラムの記述

ここでは、表示ファイル機能を使うプログラムの記述内容について、COBOLの各部ごとに説明します。

```

IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT ファイル名
 ASSIGN TO GS-ファイル識別名
 SYMBOLIC DESTINATION IS "DSP"
 FORMAT IS 画面定義体名通知域
 GROUP IS 項目群名通知域
 [SELECTED FUNCTION IS アテンション種別通知域]
 [PROCESSING MODE IS 処理種別通知域]
 [UNIT CONTROL IS 特殊制御情報通知域]
 [FILE STATUS IS 入出力状態 1 入出力状態 2].
DATA DIVISION.
FILE SECTION.
FD ファイル名.
COPY 画面定義体名 OF XMDLIB.
(01 表示レコード名.) (注)
(02 データ名 ～.)
WORKING-STORAGE SECTION.
01 画面定義体名通知域 PIC X(8).
01 項目群名通知域 PIC X(8).
[01 アテンション種別通知域 PIC X(4).]
[01 処理種別通知域 PIC X(2).]
[01 特殊制御情報通知域 PIC X(6).]
[01 入出力状態 1 PIC X(2).]
[01 入出力状態 2 PIC X(4).]
PROCEDURE DIVISION.
OPEN I-O ファイル名.
[MOVE 出力の指定 TO EDIT-MODE OF データ名.]
[MOVE 強調の指定 TO EDIT-OPTION OF データ名.]
[MOVE 色 TO EDIT-COLOR OF データ名.]
[MOVE 入力の指定 TO EDIT-STATUS OF データ名.]
[MOVE カーソルの位置 TO EDIT-CURSOR OF データ名.]
MOVE 画面定義体名 TO 画面定義体名通知域.
MOVE 項目群名 TO 項目群名通知域.

```

|             |          |                |
|-------------|----------|----------------|
| [MOVE       | 処理種別     | TO 処理種別通知域.]   |
| [MOVE       | 制御情報     | TO 特殊制御情報通知域.] |
| WRITE       | 表示レコード名. |                |
| READ        | ファイル名.   |                |
| CLOSE       | ファイル名.   |                |
| END PROGRAM | プログラム名.  |                |

注：()内はCOPY文の展開を表します。

## 環境部(ENVIRONMENT DIVISION)

表示ファイルを定義します。表示ファイルは、通常のファイルを定義するときと同様に、入出力節のファイル管理段落にファイル管理記述項を記述します。ファイル管理記述項に記述する内容を“表9.3 ファイル管理記述項に指定する情報”に示します。なお、これらの情報は、FORMで作成した画面定義体の定義内容とは関係なく値を決めることができます。

表9.3 ファイル管理記述項に指定する情報

|    | 指定する場所                | 情報の種類   | 指定する内容および用途                                                                                     |
|----|-----------------------|---------|-------------------------------------------------------------------------------------------------|
| 必須 | SELECT句               | ファイル名   | COBOL プログラム中で使用するファイル名を指定します。このファイル名は、COBOLの利用者語の規則に従った名前にします。                                  |
|    | ASSIGN句               | ファイル参照子 | “GS-ファイル識別名”の形式で指定します。このファイル識別名は、実行時に接続製品が使用するウィンドウ情報ファイルのパス名を設定する環境変数情報名となります。                 |
|    | FORMAT句               | データ名    | 作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名には、画面入出力を行うとき、画面定義体名を設定します。                        |
|    | GROUP句                | データ名    | 作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名には、画面入出力を行うとき、入出力の対象となる項目群名を設定します。                 |
| 任意 | SYMBOLIC DESTINATION句 | 画面の表示先  | "DSP" を指定します。(この句の省略値は"DSP"のため省略することができます。)                                                     |
|    | FILE STATUS句          | データ名    | 作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、入出力処理の実行結果が設定されます。(注1)<br>詳細情報は、4桁の英数字項目を指定します。 |
|    | PROCESSING MODE 句     | データ名    | 作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、画面入出力を行うとき、入出力処理の処理種別を設定します。(注2)                |
|    | SELECTED FUNCTION句    | データ名    | 作業場所節または連絡節で、4桁の英数字項目として定義したデータ名を指定します。このデータ名には、READ文完了時にアテンション種別が設定されます。(注3)                   |
|    | UNIT CONTROL句         | データ名    | 作業場所節または連絡節で、6桁の英数字項目として定義したデータ名を指定します。このデータ名には、画面入出力を行うとき、入出力処理の制御情報を設定します。(注2)                |

注1：設定される値については、“付録B 入出力状態一覧”を参照してください。

注2：詳細は“表9.4 入出力処理の種類と指定する値”を参照してください。

注3：詳細は“表9.5 アテンション情報の値とアテンション種別”を参照してください。

表9.4 入出力処理の種類と指定する値

|    | 処理種別     |      | 制御情報 (注) |
|----|----------|------|----------|
| 入力 | 通常入力     | 空白   | なし       |
|    | 非消去入力    | "NE" |          |
|    | アラーム鳴動入力 | "AL" |          |
|    | 全画面消去入力  | "CL" |          |



|    | 処理種別           |      | 制御情報 (注)   |          |
|----|----------------|------|------------|----------|
|    | 変更通知入力         | "NI" |            |          |
|    | アラーム鳴動変更通知入力   | "AI" |            |          |
| 出力 | 通常出力           | 空白   | なし         |          |
|    | 全画面消去出力        | "CL" |            |          |
|    | メニュー項目の選択禁止    | "PF" | メニュー項目     | アテンション情報 |
|    | メニュー項目の選択禁止の解除 | "PN" |            |          |
|    | 端末制御出力         | "CT" | ウィンドウ操作の種類 |          |

注：制御情報の値と詳細については、“MeFtのオンラインマニュアル”を参照してください。

表9.5 アテンション情報の値とアテンション種別

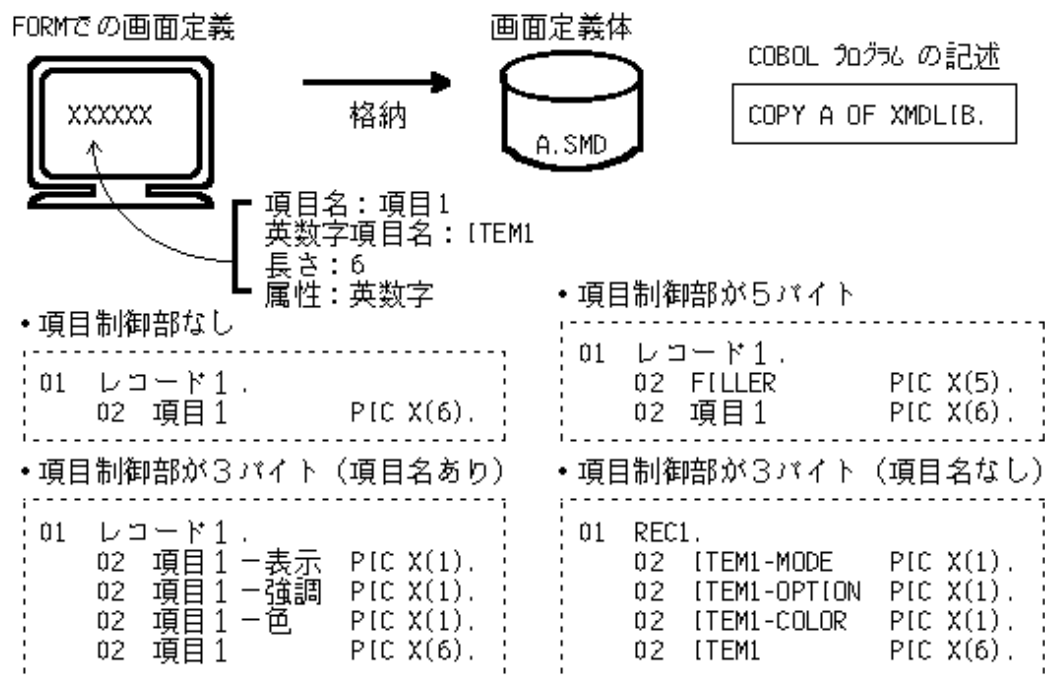
| アテンション情報の値     | アテンション種別        |
|----------------|-----------------|
| カスタマイズした値 (注1) | 画面定義時に利用者が指定した値 |
| "C000"         | クリアキー           |
| "E000"         | 実行キー (注2)       |
| "E000"         | データフルキー         |
| "E000"         | 項目脱出キー          |

注1：ファンクションキー、メニュー項目など。

注2：実行キーの存在しないシステムでは、その代替キーを設定する必要があります。この情報は、ウィンドウ情報ファイルに設定します。指定方法および内容については、“MeFtのオンラインマニュアル”を参照してください。

## データ部(DATA DIVISION)

データ部には、表示レコードの定義およびファイル管理記述項に指定したデータの定義を記述します。表示レコードに定義するレコード記述文は、IN/OF XMDLIBを指定したCOPY文を使って画面定義体から取り込むことができます。展開されるレコードの内容を以下に示します。





## 注意

表示ファイルに対してEXTERNAL句を指定する場合には、“[10.2.5.2 外部ファイル使用時の注意事項](#)”を必ずお読みください。

## 手続き部(PROCEDURE DIVISION)

画面入出力処理には、通常のファイル処理を行うときと同様に、入出力文を使います。

入出力は、次に示す順序で実行します。

1. I-O指定のOPEN文: 画面入出力処理の開始
2. READ文およびWRITE文: 画面入出力処理
3. CLOSE文: 画面入出力処理の終了

### OPEN文およびCLOSE文

OPEN文は画面入出力処理の開始時に、CLOSE文は画面入出力処理の終了時に、それぞれ1回だけ実行します。

### READ文またはWRITE文

画面を表示するときには表示レコードを指定したWRITE文を、画面からデータを読み込むときには表示ファイルを指定したREAD文を使います。WRITE文を実行する前には、FORMAT句に指定したデータ名に画面定義体名、GROUP句に指定したデータ名に項目群名を設定しておく必要があります。GROUP句に指定したデータ名に設定されている項目群に属するデータ項目が入出力の対象となります。

WRITE文またはREAD文の実行前に、特殊レジスタに値を設定することにより、表示レコード中のデータ項目の属性を変更することができます。各特殊レジスタに設定する値については、“MeFtのオンラインマニュアル”を参照してください。READ文の実行時に入力データに誤りがあり、画面定義体で再入力要求指示を定義している場合には、誤りがなくなるまで再入力のための画面表示と入力編集が繰り返し行われます。

誤りがない場合または再入力要求指示を定義していない場合は、入力編集結果の情報がCOBOLプログラムに通知されます。また、SELECTED FUNCTION句で指定したデータ名にアテンション情報が通知されます。

READ文の実行後、入力結果が特殊レジスタEDIT-STATUSに返却されます。設定される値については、“MeFtのオンラインマニュアル”を参照してください。



## 注意

画面定義体で定義された項目属性のうち、英数字日本語混在項目は、COBOLプログラム上では英数字項目として扱われます。また、この項目の内容をプログラム上で明に操作することはできません。すなわち、入力した英数字日本語混在項目は、そのまま英数字日本語混在項目として出力することしかできません。

## 特殊レジスタの使い方

特殊レジスタを使って、表示レコード中の項目制御部を持つデータ項目(画面定義体で定義したデータ項目)の属性を変更することができます。

表示ファイルの特殊レジスタには、次の5種類があります。

| 特殊レジスタ      | 用途                                     |
|-------------|----------------------------------------|
| EDIT-MODE   | 出力処理の対象とする/しないなどを指定します。                |
| EDIT-OPTION | 強調、下線付き、反転表示などを指定します。                  |
| EDIT-COLOR  | 色を指定します。                               |
| EDIT-STATUS | 入力処理の対象とする/しないなどを指定します。また、入力結果が通知されます。 |
| EDIT-CURSOR | カーソル位置を指定します。                          |

これらの特殊レジスタは、画面定義体で定義したデータ名で修飾して使います。たとえば、データ名Aの色属性の設定は、“EDIT-COLOR OF A”のように記述します。各特殊レジスタに設定する値については、“MeFtのオンラインマニュアル”を参照してください。



## 注意

- 1つのプログラム中で、項目制御部の長さ(“共用する(3バイト)”/“共用しない(5バイト)”/“なし”)が異なる複数の画面定義体を混在して使うことはできません。
- “共用する(3バイト)”の項目制御部を使用する場合、以下のような注意が必要です。  
項目制御部を入力処理と出力処理で共用する3バイト系の画面帳票定義体では、画面帳票定義体に定義された各項目について、EDIT-MODEとEDIT-STATUS、EDIT-CURSORとEDIT-OPTIONは同じ記憶領域が使用されます。たとえば、データ名Aについて、“EDIT-MODE OF A”と“EDIT-STATUS OF A”、“EDIT-CURSOR OF A”と“EDIT-OPTION OF A”は、同じ記憶領域を使用します。

## 入出力エラー処理

入出力エラーの検出方法および入出力エラーが発生したときの実行結果については、“[7.6 入出力エラー処理](#)”を参照してください。

## 9.2.6 プログラムの翻訳・リンク

### 翻訳

翻訳オプションFORMLIBで、画面定義体を格納したファイルの格納先を指定します。[参照]“[A.2.19 FORMLIB \(画面帳票定義体ファイルのフォルダの指定\)](#)”

複数の画面定義体を使用し、その拡張子がそれぞれ異なっている場合、翻訳オプションFORMEXTで、拡張子を指定します。[参照]“[A.2.18 FORMEXT \(画面帳票定義体ファイルの拡張子の指定\)](#)”

### リンク

結合が必要なライブラリは特にありません。

## 9.2.7 ウィンドウ情報ファイルの作成

ここでは、表示ファイル機能を使って画面入出力処理を行うときのウィンドウ情報ファイルに設定する情報および注意事項について記述します。ウィンドウ情報ファイルの詳しい内容や作成方法については、“MeFtのオンラインマニュアル”を参照してください。

ウィンドウ情報ファイルに設定する情報を“[表9.6 ウィンドウ情報ファイルの設定情報](#)”に示します。

表9.6 ウィンドウ情報ファイルの設定情報

| 情報の種類  | 設定する内容および用途                                         |
|--------|-----------------------------------------------------|
| MEDDIR | 画面定義体を格納したフォルダのパス名を設定します。                           |
| MEDSUF | 画面定義体を格納したファイルの拡張子を指定します。省略した場合、拡張子は“SMD”となります。     |
| KEYDEF | アテンション情報で使用するキーが実際のキーボードに存在しない場合は、キーの割付けを行う必要があります。 |

## 9.2.8 プログラムの実行

表示ファイル機能を使った画面入出力を行うプログラムを実行するときには、以下の環境設定が必要です。

### MeFtを使用する場合

- 環境変数PATHにMeFtの格納されているフォルダを追加しておく必要があります。
- ファイル識別名を環境変数情報名として、ウィンドウ情報ファイル名を設定します。



## 参考

ウィンドウ情報ファイルを相対パス名で指定した場合、次の順序で検索されます。

1. 環境変数MEFTDIRに設定したフォルダ

## 2. カレントフォルダ

### MeFt/NET-SV(MeFt/NET連携機能)を使用する場合

- 環境変数PATHにMeFt/NET-SVの格納されているフォルダを追加しておく必要があります。
- ファイル識別名を環境変数情報名として、ウィンドウ情報ファイル名を設定します。実際のウィンドウ情報ファイルは、クライアント側に用意しておく必要があります。
- 接続製品としてMeFt/NETを使用することを示す環境変数情報を指定する必要があります。指定方法には、以下の2つの方法があります。
  - ファイル識別名を環境変数情報名として、接続製品名にMEFTNETを設定する。[参照]“5.4.1.70 ファイル識別名(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)”
  - 環境変数情報@CBR\_PSFIL\_DSPにMEFTNETを設定する。[参照]“5.4.1.43 @CBR\_PSFIL\_XXX(表示ファイルから使用する接続製品名(宛先ごとの指定))”

MeFt/NET-SVの詳細については、“MeFt/NET-SVのヘルプ”を参照してください。



### 例

#### 表示ファイル機能を使って画面入出力を行う場合の実行例

- COBOL プログラムのASSIGN句の記述内容

```
ASSIGN TO GS-DSPFILE
```

- 実行用の初期化ファイルの内容

#### ー MeFtを使用する場合

```
:
DSPFILE=C:¥DIR1¥MEFWRC
:
```

#### ー MeFt/NET-SV (MeFt/NET連携機能)を使用する場合

```
:
DSPFILE=DSPFILE, MEFTNET
:
```

COBOL プログラムのASSIGN句に指定したファイル識別名に、ウィンドウ情報ファイルを割り当てます。

### MeFt/Webを使用する場合

MeFt/Webを使用する場合には、“MeFt/Web 説明書”を参照してください。

### 表示ファイルの入力中断

入力待ち状態の画面を、他のアプリケーションからMeFtの関数を直接呼び出して、入力中断することができます(MeFtを使用する場合)。

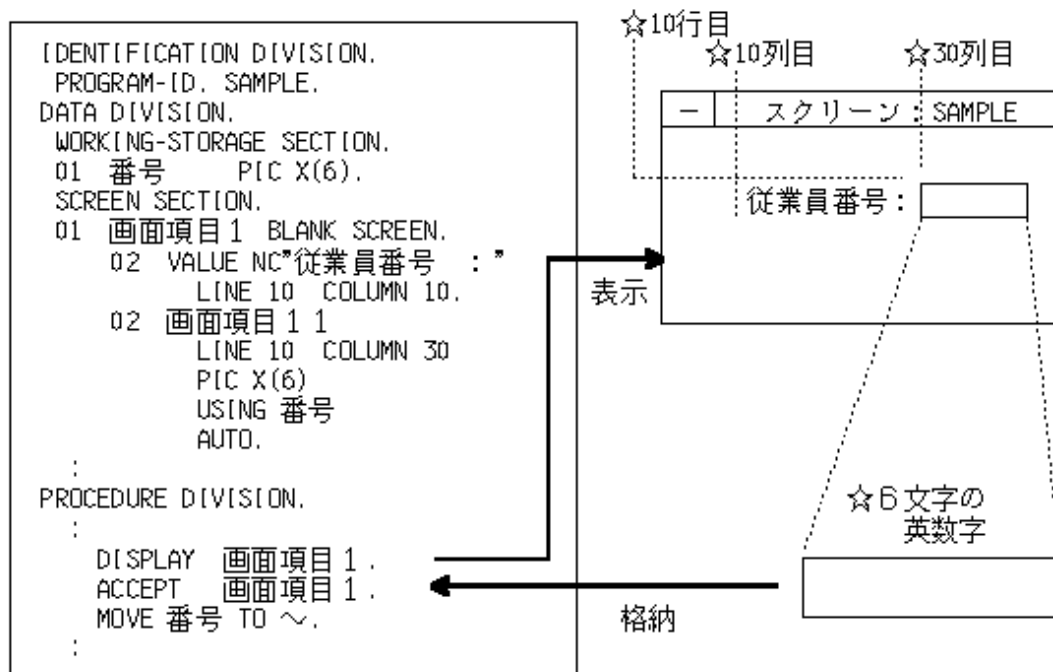
詳細については、“G.1.1 ウィンドウハンドル獲得サブルーチン”を参照してください。

## 9.3 スクリーン操作機能の使い方

ここでは、スクリーン操作機能を使って、画面入出力を行う方法について説明します。なお、スクリーン操作機能を使った例題プログラムがサンプルプログラムとして提供されていますので、参考にしてください。

### 9.3.1 概要

スクリーン操作機能では、DISPLAY文を使って画面を表示し、ACCEPT文を使って画面からデータを入力します。画面のレイアウトは、データ部の画面節に画面データ記述項を使って定義します。画面節に定義された画面項目は、行番号と列番号によって、画面に配置されます。



### 9.3.2 スクリーンウィンドウ

スクリーン操作機能によって画面入出力を行うためのウィンドウは、1つの実行単位につき1つで、最初に実行されるACCEPT文またはDISPLAY文によって創成され、実行単位の終了時に消去されます。

通常このウィンドウの論理画面は、縦25行、横80列の大きさで創成されますが、環境変数情報@ScrnSizeによって変更することもできます。また、このウィンドウを自動的に消去するか、利用者がメッセージを確認してから消去するかは、環境変数情報@WinCloseMsgによって指定することができます。実行時に変更することができるウィンドウの属性を“表9.7 スクリーン操作作用のウィンドウの属性の変更”に示します。

環境変数情報の指定方法については、以下を参照してください。

- “5.4.1.66 @ScrnSize(スクリーン操作の論理画面の大きさの指定)”
- “5.4.1.68 @WinCloseMsg(ウィンドウを閉じるときのメッセージ表示の指定)”

表9.7 スクリーン操作作用のウィンドウの属性の変更

| 属性         | 環境変数情報              | 設定値   | 意味                          |
|------------|---------------------|-------|-----------------------------|
| ウィンドウの自動消去 | @WinCloseMsg<br>(注) | ON    | メッセージ表示後、消去します。             |
|            |                     | OFF   | メッセージを表示しないで消去します。          |
| ウィンドウの大きさ  | @ScrnSize           | (m,n) | 論理画面の大きさを桁数(m)と行数(n)で指定します。 |

注：この環境変数情報は、小入出力機能で使用するコンソールウィンドウに対しても有効になります。

### 9.3.3 キー定義ファイルの利用

スクリーン操作機能では、プログラム実行時にキー定義ファイルを使用して、ファンクションキーの利用者定義および、小数点位置固定入力の選択を行うことができます。

ファンクションキーの利用者定義は、キー定義ファイルにキー入力の定義を行うことで、特定のファンクションキーの入力を無効にしたり、入力されたファンクションキーでプログラム中の処理を切り分けることができます。また、画面からのデータ入力の終了を、定義したファンクションキーの入力によって指示することができます。これにより、スクリーン画面上でのキー押下の受取り方をユーザが任意にカスタマイズすることができます。

小数点位置固定入力の選択は、キー定義ファイルにデータ入力方法の定義を行うことで、数字項目、数字編集項目の入力操作を変更することができます。これにより、小数点位置でのカーソルの動作を選択することができます。

キー定義ファイルの指定方法については、“[5.4.1.44 @CBR\\_SCR\\_KEYDEFFILE \(スクリーン操作のキー定義ファイルの指定\)](#)”を参照してください。

## キー定義ファイルの記述形式

キー定義ファイルの記述内容を以下に示します。

```
[COBOL. KBD]
ESC=11000
F01=11001
SHIFT+F01=11020
ENTER=2

[COBOL. ACCPMODE]
numeric_input=MODE2
```

## キー入力の定義(セクション名:[COBOL.KBD])

キー入力の定義は、定義したキーが画面からのデータ入力を終了するキー(以降では終了キーといいます)として有効か無効か、状態キー1に返却される値、状態キー2に返却される値を指定します。終了キーとして有効にしたキーを入力した場合、画面からのデータ入力が終了し、特殊名段落のCRT STATUS句に指定したデータ項目にキー定義ファイルで定義した状態キー1および状態キー2の情報が返却されます。終了キーとして無効にしたキーを入力した場合、データ入力の終了とはみなされません。

キー定義ファイルの指定を省略した場合、ENTERキーの入力によって、画面からのデータ入力を終了することができます。

キー名=XYZZZ

X : 終了キーとして有効か無効かを指定するフラグ ('1' 有効、'0' 無効)  
Y : 状態キー1の値 ('1' または '2' )  
ZZZ : 状態キー2の値 ('000' - '999')



### 例

ENTER=2

ENTERキーの入力を、Tabキーの入力として扱います。

## データ入力方法の定義(セクション名:[COBOL.ACCPMODE])

数字項目および数字編集項目のデータ入力時の入力方法を指定します。

numeric\_input = { MODE1 | MODE2 }

### MODE1

通常入力。

通常入力と同様に、小数点位置での自動桁合わせは行いません。

### MODE2

小数点位置固定入力。

小数点の入力により、小数点位置で自動桁合わせが行われます。

入力方法によるデータの入力結果の違いを以下の手順にしたがって入力したときの例で示します。



## 例

PIC 9999 の場合(△はカーソル位置を指します)

| 入力手順 |        | MODE1        | MODE2        |
|------|--------|--------------|--------------|
| ↓    | 初期表示   | 0 0 0 0<br>△ | 0 0 0 0<br>△ |
|      | ”1”を入力 | 1 0 0 0<br>△ | 1 0 0 0<br>△ |
|      | ”2”を入力 | 1 2 0 0<br>△ | 1 2 0 0<br>△ |
|      | ”.”を入力 | 1 2 0 0<br>△ | 0 0 1 2      |

PIC ZZ99.99 の場合(□は空白、△はカーソル位置を指します)

| 入力手順 |              | MODE1         | MODE2         |
|------|--------------|---------------|---------------|
| ↓    | 初期表示         | □□00. 00<br>△ | □□00. 00<br>△ |
|      | ”1”を入力       | 1□00. 00<br>△ | □□10. 00<br>△ |
|      | ”2”を入力       | 1200. 00<br>△ | □□12. 00<br>△ |
|      | ”3”を入力       | 1230. 00<br>△ | □123. 00<br>△ |
|      | ”4”を入力       | 1234. 00<br>△ | 1234. 00<br>△ |
|      | ”5”を入力       | 1234500<br>△  | 1234. 50<br>△ |
|      | ”6”を入力       | 1234560<br>△  | 1234. 56<br>△ |
|      | ”ENTERキー”を入力 | 4560. 00      | 1234. 56      |

上例は、TO指定で使用した例です。USING指定の場合またはFROM指定とTO指定が同時に指定されている場合、あらかじめ0で初期化されていないデータ項目では、0表示されている箇所が空白表示になることがあります。

キー定義のセクション名は固定です。

キー名の記述方法および指定可能なキーについては、製品に含まれるキー定義ファイルのサンプルを参照してください。



## 注意

- 状態キー2の値については、現在は'000'-'255'まで使用可能です。これ以上の値を有効なキーとして利用した場合は、CRT STATUS句に返却できる範囲を超えることになります。この場合の動作は、保証しません。
- “ENTER=2”を指定した場合、ENTERキーで画面からのデータ入力を終了させることはできません。この場合、別のキーを終了キーとして設定しておく必要があります。“ENTER=2”を指定しないかぎり、ENTERキーは終了キーとして使用できます。ENTERキーには、2(Tabキー)以外の値は割当てられません。

- ・キー定義ファイル内に[COBOL.KBD]セクションおよび[COBOL.ACCPMODE]セクションが複数ある場合、最初に現れたセクションが有効になります。セクション内の各キーの定義についても、同じキー名の複数の情報がある場合は、最初に現れたキー名の情報が有効になります。
- ・キー定義ファイル内の記述には、空白を含めてはなりません。
- ・キー定義ファイル内の記述は、大文字、小文字を区別して記述してください。
- ・キー定義ファイル内の記述で、行の先頭にセミicolon(;)のある行はコメント行とみなします。
- ・キー定義ファイルに、ファンクションキー(Fxxキー)でないキーが無効として設定されている場合、そのキーが押下された場合には、それぞれのキーの機能を実行します。
- ・キー定義ファイルに記述可能なキーのうち、キー定義ファイル中に記述されていないキーについては、省略値として、値'01999'が使用されます。

### 9.3.4 プログラムの記述

ここでは、スクリーン操作機能を使ったプログラムの記述内容について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
[CURSORS IS データ名 1.]
[CRT STATUS IS データ名 2.]
DATA DIVISION.
WORKING-STORAGE SECTION.
[01 データ名 1.]
[02 行番号 PIC 9(3).]
[02 列番号 PIC 9(3).]
[01 データ名 2.]
[02 状態キー 1 PIC 9.]
[02 状態キー 2 PIC 9.]
[02 PIC X.]
SCREEN SECTION.
01 画面項目 1 ~.
PROCEDURE DIVISION.
DISPLAY 画面項目 1 ~.
ACCEPT 画面項目 1 ~ [ON EXCEPTION ~].
END PROGRAM プログラム名.
```

#### 環境部(ENVIRONMENT DIVISION)

特殊名段落に以下の情報を定義することができます。

- ・カーソル位置の設定または受取りを行うデータ名をCURSOR句に指定します。
- ・画面入出力状態を受け取るためのデータ名をCRT STATUS句に指定します。このデータ名に返却される値を“表9.8 画面入出力状態の設定値”に示します。

表9.8 画面入出力状態の設定値

| 状態キー1(1文字目) | 状態キー2(2文字目) | 意味                                                        |
|-------------|-------------|-----------------------------------------------------------|
| "0"         | "0"         | オペレータにより終了キーが入力されました。                                     |
|             | "1"         | 最終項目が入力されました。                                             |
| "1"         | X"00"ーX"FF" | 利用者定義のファンクションキーが入力されました(状態キー2には、ファンクションキー番号が設定されます)。(注1)  |
| "2"         | X"00"ーX"FF" | システム定義のファンクションキーが入力されました(状態キー2には、ファンクションキー番号が設定されます)。(注2) |



| 状態キー1(1文字目) | 状態キー2(2文字目) | 意味               |
|-------------|-------------|------------------|
| "9"         | X"00"       | 入力項目がありません。(エラー) |

注1：利用者定義のファンクションキーとは、キー定義ファイルで定義されたファンクションキーのことです。

注2：キー定義ファイルを使用する場合、状態キー2の内容はキー定義ファイルで定義された内容となります。キー定義ファイルを使用しない場合またはキー定義ファイルに入力されたファンクションキーの定義がない場合、状態キー2にはファンクションキー番号が返却されます。

## データ部(DATA DIVISION)

データ部の最後に画面節を記述します。画面節には、画面データ記述項を用いて、画面項目を定義します。画面項目には、定数項目、入力項目、出力項目および更新項目があります。これらの項目は、画面データ記述項の書き方によって区別されます。画面項目の属性と画面データ記述項に指定できる句の関係は“表9.9 画面項目に指定するCOBOLの句”を、画面データ記述項の書き方は“COBOL文法書”を参照してください。

なお、画面節に基底場所節で定義したデータ項目を指定することはできません。

FROM句とTO句の両方に同じデータ項目を指定した場合は、更新項目と同じ扱いになります。

表9.9 画面項目に指定するCOBOLの句

| 目的   |              | COBOLの句           | 画面項目の属性 |    |    |    | (注) |    |
|------|--------------|-------------------|---------|----|----|----|-----|----|
|      |              |                   | 定数      | 入力 | 出力 | 更新 | 集団  | 基本 |
| 強調   | 高輝度表示        | HIGHLIGHT句        | ○       | ○  | ○  | ○  | ×   | ○  |
|      | 低輝度表示        | LOWLIGH句          | △       | △  | △  | △  | ×   | △  |
|      | 点滅表示         | BLINK句            | △       | △  | △  | △  | ×   | △  |
|      | 下線付き表示       | UNDERLINE句        | ○       | △  | ○  | ○  | ×   | ○  |
| 色    | 背景色の指定       | BACKGROUND-COLOR句 | ○       | ○  | ○  | ○  | ○   | ○  |
|      | 前景色の指定       | FOREGROUND-COLOR句 | ○       | ○  | ○  | ○  | ○   | ○  |
|      | 背景色と前景色の反転   | REVERSE-VIDEO句    | ○       | ○  | ○  | ○  | ×   | ○  |
| 音    | オーディオトーンを鳴らす | BELL句             | ○       | △  | ○  | ○  | ×   | ○  |
| 表現形式 | ゼロの時の空白表示    | BLANK WHEN ZERO句  | ×       | △  | ○  | ○  | ×   | ○  |
|      | 桁よせの指定       | JUSTIFIED句        | ×       | ○  | ○  | ○  | ×   | ○  |
|      | 演算符号の位置指定    | SIGN句             | ×       | ○  | ○  | ○  | ○   | ○  |
| 表示方法 | 全画面消去の指定     | BLANK SCREEN句     | ○       | △  | ○  | ○  | ○   | ○  |
|      | 画面の部分消去の指定   | ERASE EOS句        | ○       | △  | ○  | ○  | ×   | ○  |
|      | 全行消去の指定      | BLANK LINE句       | ○       | △  | ○  | ○  | ×   | ○  |
|      | 行の部分消去の指定    | ERASE EOL句        | ○       | △  | ○  | ○  | ×   | ○  |
|      | 非表示状態の指定     | SECURE句           | ×       | ○  | ×  | ×  | ○   | ○  |
| 位置   | 表示位置の列の指定    | COLUMN NUMBER句    | ○       | ○  | ○  | ○  | ×   | ○  |
|      | 表示位置の行の指定    | LINE NUMBER句      | ○       | ○  | ○  | ○  | ×   | ○  |
| 入力   | 入力形態の指定      | FULL句             | ×       | ○  | △  | ○  | ○   | ○  |
|      | 入力形態の指定      | REQUIRED句         | ×       | ○  | △  | ○  | ○   | ○  |
| その他  | カーソルの自動スキップ  | AUTO句             | ×       | ○  | △  | ○  | ○   | ○  |
|      | 一般的性質の指定     | PICTURE句          | ×       | ○  | ○  | ○  | ×   | ○  |
|      | 表現形式の指定      | USAGE句            | ×       | ○  | ○  | ○  | ○   | ○  |
|      | 定数項目の指定      | VALUE句            | ○       | ×  | ×  | ×  | ×   | ○  |

- : 指定可能
- △ : 指定できるが有効にならない
- × : 指定できない
- 注 : 集団は集団項目として定義し、基本は基本項目として定義してください。

## 手続き部(PROCEDURE DIVISION)

画面を表示するには、画面項目を指定したDISPLAY文を使います。DISPLAY文を実行すると、ディスプレイ装置に、指定した画面項目で定義された画面が表示されます。

画面からデータを入力するには、画面項目を指定したACCEPT文を使います。ACCEPT文を実行すると、ディスプレイ装置に表示された画面からデータを入力することができます。入力操作が終了すると、特殊名段落のCRT STATUS句に指定したデータ名に、画面入力状態が設定されるので、その値によって処理を切り分けることができます。

## 9.3.5 プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

## 9.3.6 プログラムの実行

1. ファンクションキーの利用者定義を行う場合、環境変数情報@CBR\_SCR\_KEYDEFFILEにキー定義ファイルを指定します。[参照]“5.4.1.44 @CBR\_SCR\_KEYDEFFILE (スクリーン操作のキー定義ファイルの指定)”



例

キー定義ファイルを使用する場合

```
@CBR_SCR_KEYDEFFILE=SAMPLE. KBD
```

SAMPLE.KBDファイルの内容

```
[COBOL. KBD]
F01=11001
F02=11002
F03=11003
F04=11004
F05=01005
F06=01006
F07=01007
F08=01008
F09=01009
F10=01010
F11=01011
F12=01012
F13=11013
F14=11014
F15=11015
F16=11016
```

F1～F4キーおよびF13～F16キーの入力を有効とし、F5～F12キーの入力を無効とします。

2. ウィンドウの属性を変更する場合、“表9.7 スクリーン操作作用のウィンドウの属性の変更”の環境変数情報の設定を行います。
3. プログラムを実行します。



注意

- ・ 小入出力機能とスクリーン操作機能は、ACCEPT文またはDISPLAY文でウィンドウを操作しますが、それぞれ別のウィンドウを使います。
- ・ スクリーン操作機能で使用するウィンドウは、アイコン化できません。

- 論理画面の大きさは、画面項目で定義したデータ項目の大きさより小さくならないようにしてください。
  - 入力項目は、つねに下線付きで表示されます。
  - 環境変数情報@ScrnSizeに設定された論理画面の大きさ((桁数+1)×行数)が16250を超えた場合、プログラム実行時にエラーとなります。
  - スクリーンウィンドウの[閉じる]ボタンをクリックした場合、およびスクリーンウィンドウのポップアップメニューから「閉じる」コマンドを選択した場合は、プログラムを強制終了するかどうかを確認するダイアログが表示されます。このダイアログで強制終了を選択した場合は、プログラムを終了します。
  - 本システムでは、以下のキーが無効とされている場合、システムにそのキーの制御を任せます。
    - ー F10キー(システムメニューの表示)
    - ー ALT+F4キー(アプリケーションの終了)
    - ー ALT+F6キー(アクティブなウィンドウを変更する)
  - 本システムでは、以下のキーの押下はキー定義ファイルによる有効無効の指定に関係なく、システムにその制御を任せます。
    - ー CTRL+ESC
    - ー ALT+ESC
    - ー CTRL+TAB
    - ー ALT+TAB
- .....

## 第10章 サブプログラムを呼び出す～プログラム間連絡機能～

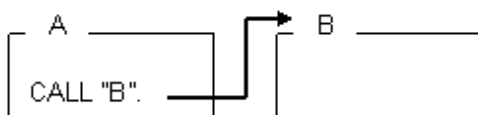
本章では、プログラムからプログラムを呼び出す機能について説明します。この機能をプログラム間連絡機能といいます。

### 10.1 呼出し関係の概要

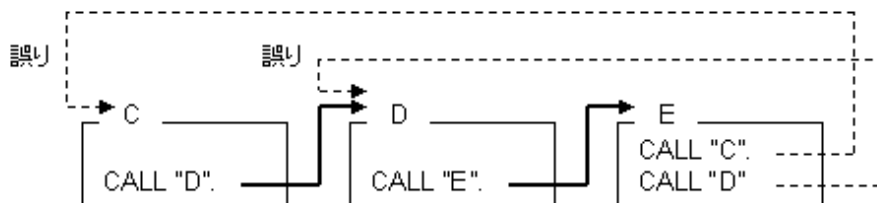
ここでは、プログラムの呼出し関係の概要および動的プログラム構造について説明します。

#### 10.1.1 呼出し関係の形態

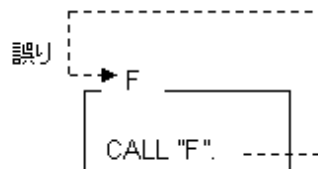
COBOLプログラムは、他のプログラムを呼び出したり、他のプログラムから呼び出されたりすることができます。



他のプログラムは、他言語で記述されたプログラムでもかまいません。ただし、COBOLプログラムを再帰的に呼び出すことはできません。



また、COBOLプログラム自身を呼び出すこともできません。



#### 10.1.2 各種呼出し規約の違い

NetCOBOLは、3つの呼出し規約をサポートしています。

1. COBOL呼出し規約
2. C呼出し規約
3. STDCALL呼出し規約



#### 参考

2.と3.は、Microsoft社のCコンパイラVisual C++がサポートする\_cdecl、\_stdcallにそれぞれ対応しています。  
本書では、使用するCコンパイラをMicrosoft社のVisual C++として記載しています。

表10.1 呼出し規約の違い

| 各呼出し規約          | パラメタのスタックへの積み方と取出し方                | スタックポインタ(SP)の復元  | 呼出し名(注2)と外部参照名の規則                                 |
|-----------------|------------------------------------|------------------|---------------------------------------------------|
| COBOL呼出し規約 (注1) | パラメタは右から左へとスタックに積まれ、左から右へと取り出されます。 | 呼び出した側がSPを戻します。  | 呼出し名の先頭に“_”を付加して外部参照名とします。                        |
| C呼出し規約          |                                    |                  |                                                   |
| STDCALL呼出し規約    |                                    | 呼び出された側がSPを戻します。 | 呼出し名の先頭に“_”を付加し、呼出し名の末尾に“@###”(注3)を付加して外部参照名とします。 |

注1：本システムでは、COBOL呼出し規約とC呼出し規約は同じです。

注2：COBOLで指定する呼出し名は、英大文字と英小文字を区別して記述し、翻訳オプションNOALPHALを指定します。

注3：###は、パラメタのバイト数を表す10進数です。



## 参考

アンダースコアは、COBOLの利用者語として使用できます。

## 10.1.3 呼出し規約と指定形式

NetCOBOLがサポートする呼出し規約と指定形式の対応を“表10.2 呼出し規約と指定形式”に示します。

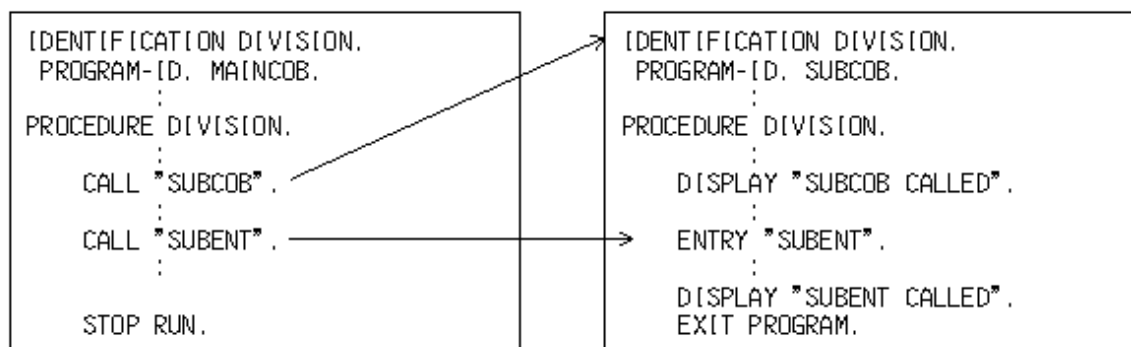
表10.2 呼出し規約と指定形式

| 各呼出し規約        | CALL文、ENTRY文、PROCEDURE DIVISIONでの指定 |
|---------------|-------------------------------------|
| COBOL 呼出し規約   | なし                                  |
| C呼出し規約        | WITH C LINKAGE                      |
| STDCALL 呼出し規約 | WITH STDCALL LINKAGE                |

以下に、指定形式の例を示します。

## COBOL呼出し規約

図10.1 COBOLプログラムからCOBOLプログラムを呼び出す場合



## C呼出し規約

図10.2 COBOLプログラムからCプログラムを呼び出す場合

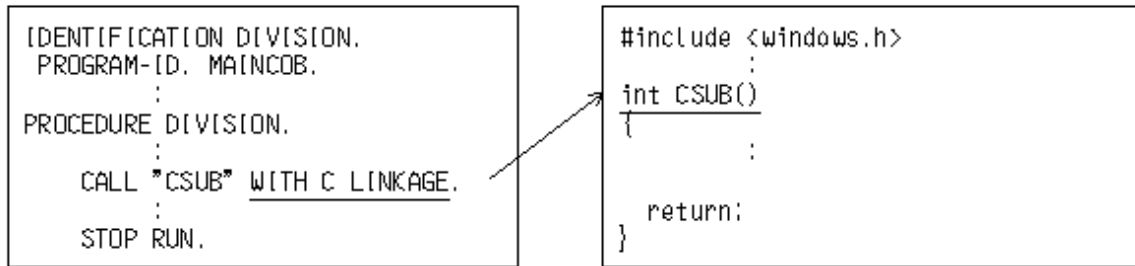
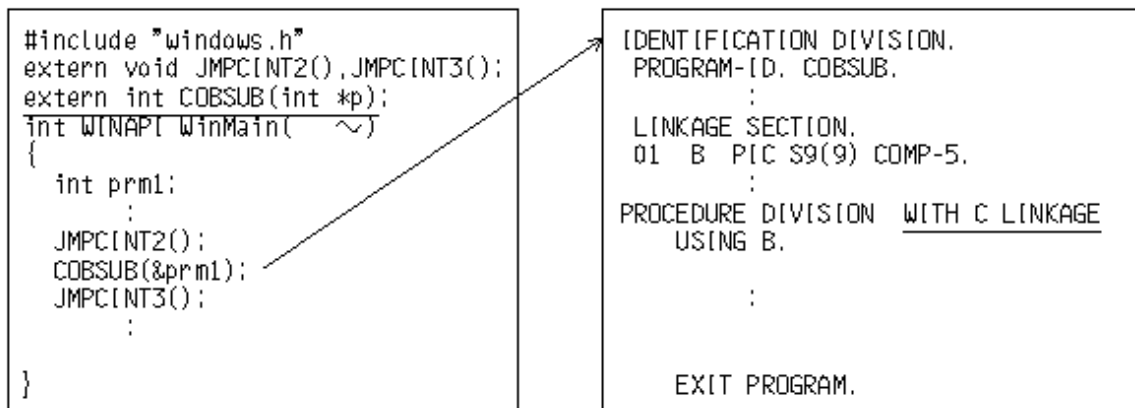
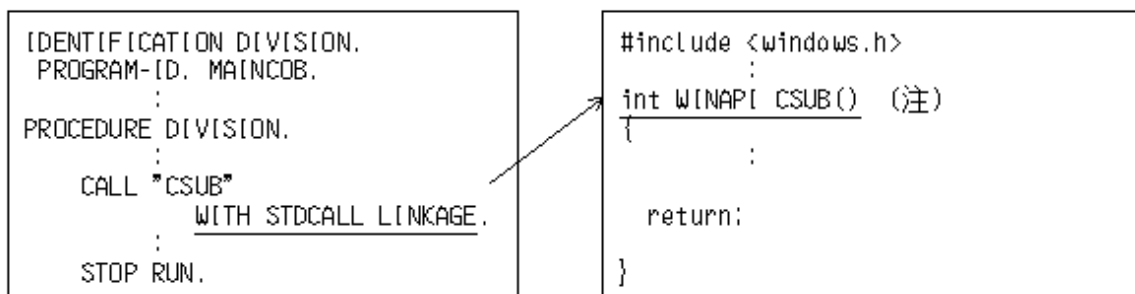


図10.3 CプログラムからCOBOLプログラムを呼び出す場合



## STDCALL呼出し規約

図10.4 COBOLプログラムからCプログラムを呼び出す場合

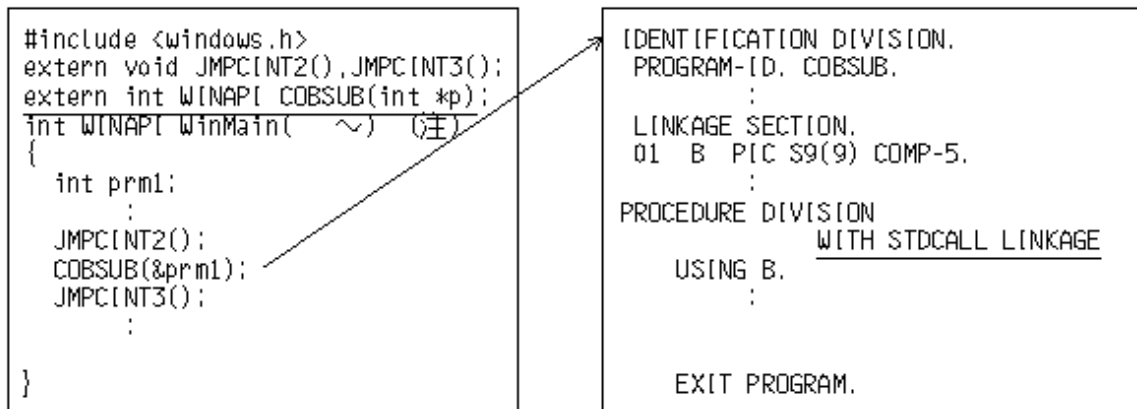


注：関数名の宣言が以下の場合、STDCALL呼出し規約となります。

- int WINAPI CSUB()
- int CALLBACK CSUB()
- int PASCAL CSUB()

- int far pascal CSUB()
- int \_stdcall CSUB()

図10.5 CプログラムからCOBOLプログラムを呼び出す場合



注：呼び出す関数名の外部宣言が以下の場合、STDCALL呼出し規約となります。

- extern int WINAPI COBSUB(int \*p)
- extern int CALLBACK COBSUB(int \*p)
- extern int PASCAL COBSUB(int \*p)
- extern int far pascal COBSUB(int \*p)
- extern int \_stdcall COBSUB(int \*p)



#### 注意

翻訳する前に、呼び出すプログラムと呼び出されるプログラムの呼出し規約を確認してください。呼出し規約が一致していない場合、リンク時または実行時にエラーとなります。

## 10.1.4 COBOLの言語間の環境

COBOLには、実行環境と実行単位という概念があります。ここでは、実行環境と実行単位について説明します。

### 実行環境と実行単位

COBOLプログラムの実行環境は、COBOLプログラムがはじめて呼び出されたときに開設され、COBOLの実行単位の終了時に閉鎖されます。ただし、マルチスレッドプログラムでは、実行環境の閉鎖のタイミングは異なりますので、“[24.3.1 実行環境と実行単位](#)”を参照してください。

COBOLの実行単位とは、COBOLの主プログラムに制御が渡ってからCOBOLの主プログラムが呼出し元に復帰する(下図(a))か、STOP RUN文が実行される(下図(b))までをいいます。ただし、他言語で記述されたプログラム(以降、他言語プログラムといいます)からCOBOLプログラムを呼び出す場合は例外があります。

ここでいう、COBOLの主プログラムとは、COBOLの実行単位中で最初に制御が渡ったCOBOLプログラムをいいます。

他言語プログラムからCOBOLプログラムを呼び出す場合は、最初のCOBOLプログラムの呼出しの前にJMPCINT2を呼び出し、最後のCOBOLプログラムの呼出しの後でJMPCINT3を呼び出すようにしてください。JMPCINT2の呼び出しで、COBOLプログラムの実行単位が開始されます。JMPCINT2を呼び出さずに、他言語プログラムからCOBOLプログラムを呼び出した場合、他言語プログラムから呼び出されたCOBOLプログラムがCOBOLの主プログラムとなります。このため、呼び出された回数だけCOBOLプログラムの実行環境の開設と閉鎖が行われ、実行性能が低下します(下図(c))。しかし、JMPCINT2を呼び出すことにより、JMPCINT3の呼び出しまでをCOBOLの実行単位とすることができ、JMPCINT3が呼び出されるまで実行環境の閉鎖は行われなくなります。(下図(d))。

JMPCINT2とJMPCINT3の呼び出し方については、“[G.2 他言語連携で使用するサブルーチン](#)”を参照してください。

## 実行環境の開設と閉鎖時の処理

実行環境の開設時には、COBOLプログラムが実行するために必要となる実行用の初期化ファイルの情報などが取り込まれます。実行環境閉鎖時には、COBOLプログラムで使用された資源を解放します。このときに行われる処理としては、小入出力機能で使用されたファイルのクローズ、オープンされたままのファイルのクローズ、外部ファイルのクローズ、外部データの解放、ファクトリオブジェクトの解放、未解放のオブジェクトインスタンスの解放などがあります。たとえば、下図(c)のような使い方をした場合、プログラムAとプログラムBの実行環境と、プログラムCの実行環境は別になるので、プログラムAとプログラムBの外部データと、プログラムCの外部データはそれぞれ別の領域となります。また、このような使い方をした場合、以下の注意が必要となります。このため、下図(d)のような使い方をするようにしてください。

図10.6 COBOLプログラムだけ

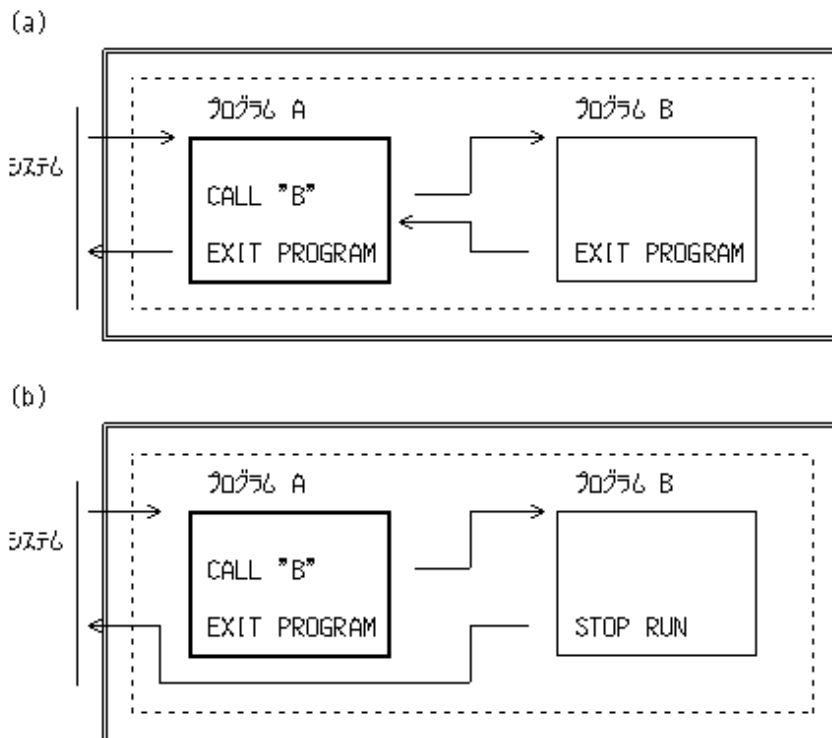
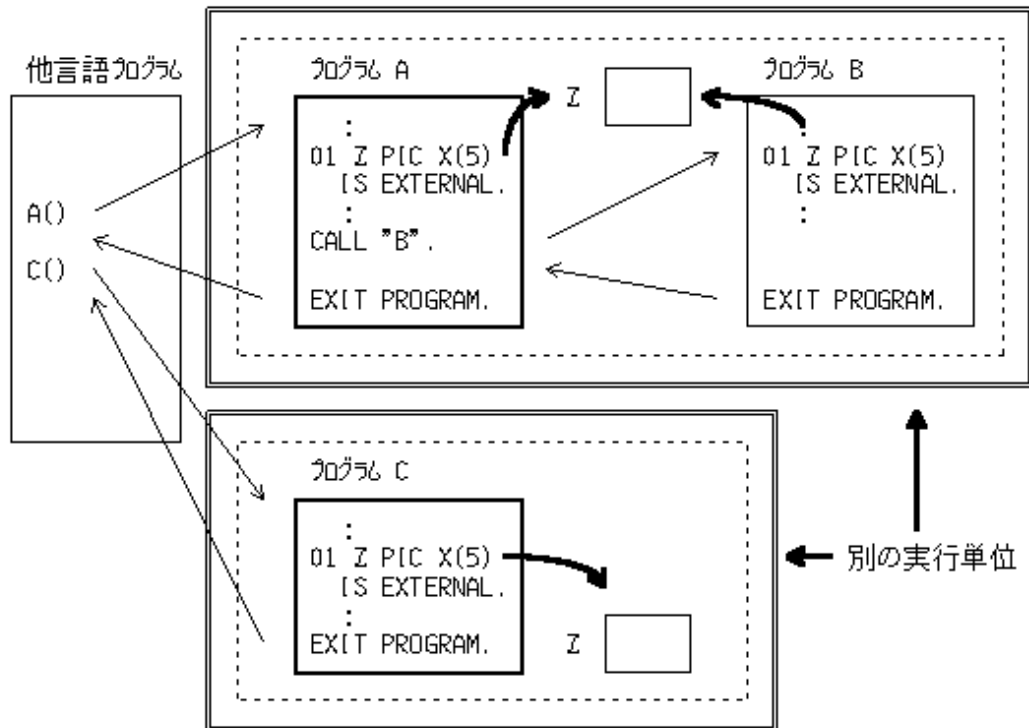


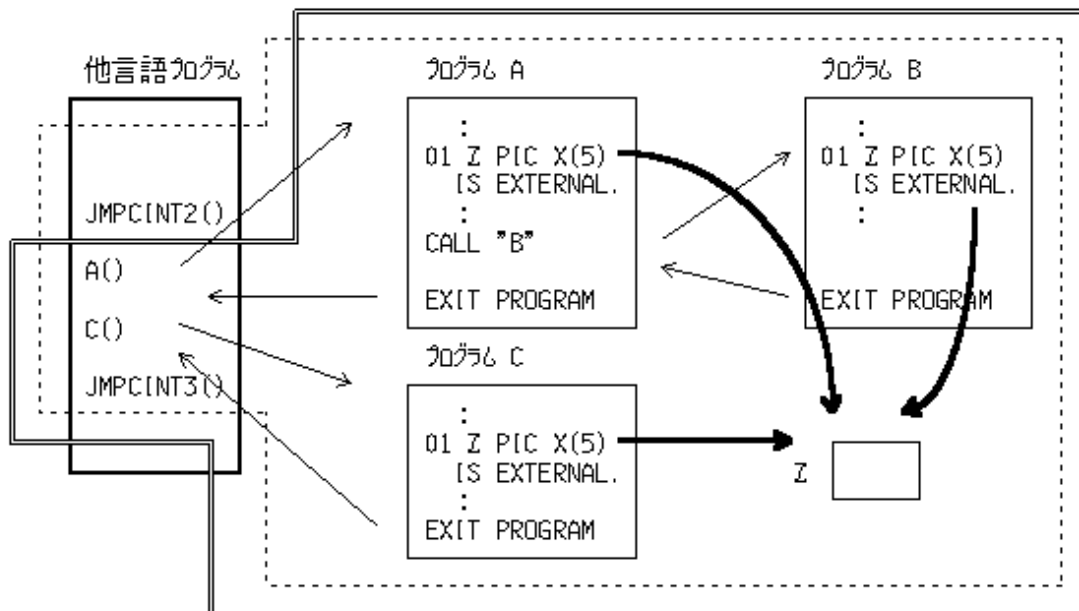


図10.7 他言語プログラムからCOBOLプログラムの呼出し

(c) JMPCINT2とJMPCINT3を未使用



(d) JMPCINT2とJMPCINT3を使用



□ : COBOL の主プログラムを示す。 □ : COBOL の実行単位を示す。

□ : COBOL の実行環境を示す。



## 注意

次の場合、他言語プログラムからCOBOLの実行単位を複数回呼び出してはいけません。

- ・ 外部データまたは外部ファイルを使用している場合
- ・ オープンされたままのファイルに対して、強制クローズが行われた場合
- ・ COBOLの主プログラム以外でSTOP RUN文を実行した場合
- ・ オブジェクト指向プログラミング機能を使用している場合



## ポイント

JMPCINT2およびJMPCINT3は、C呼出し規約で呼び出してください。

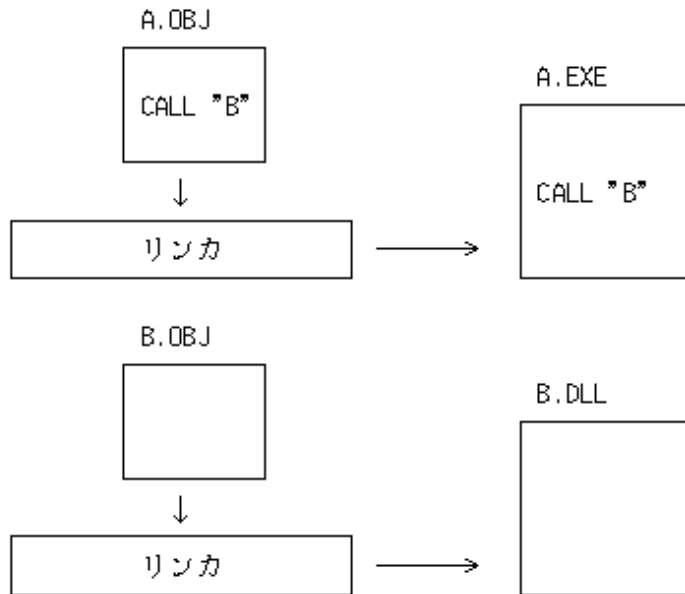
## 10.1.5 動的プログラム構造

ここでは、動的プログラム構造の特徴、副プログラムのエントリ情報および注意事項について説明します。

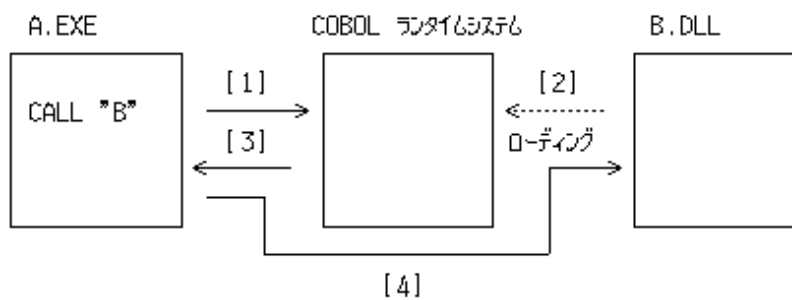
### 10.1.5.1 動的プログラム構造の特徴

動的プログラム構造を利用すると、副プログラムの仮想メモリ上へのローディングは、実際に副プログラムがCALL文によって呼び出されたときに、COBOLのランタイムシステムによって行われます。また、一度ローディングされた副プログラムは、CANCEL文により、仮想メモリ上から削除することができます。このため、実行可能ファイルの起動時に、副プログラムがすべてロードされる単純構造や動的リンク構造よりも実行可能ファイルの起動は速く、仮想メモリおよび実メモリの節約も期待できます。ただし、副プログラムの呼出しは、COBOLのランタイムシステムを介するために遅くなります。

マルチスレッドで動的プログラム構造を利用する場合は、“[24.10.5 動的プログラム構造](#)”を参照してください。



実行時に COBOL ランタイムシステムがアドレス解決



- [1] COBOLランタイムシステムが呼び出されます。
- [2] COBOLランタイムシステムは、プログラムBをローディングします。
- [3] プログラムAに復帰します。
- [4] プログラムBに分岐します。

[1]～[4]は、処理する順番を示します。

動的プログラム構造のプログラムを実行する場合は、副プログラムのエントリ情報が必要です。ただし、副プログラムのDLLファイル名を“プログラム名.DLL”とした場合、エントリ情報は必要ありません。このため、動的プログラム構造で呼び出す副プログラムのDLLは、1つの副プログラムを1つのDLLとし、ファイル名は“プログラム名.DLL”にすることをおすすめします。

このプログラム構造を使用する利用者は全体の構造をよく理解した上で使用しなければなりません(“[10.1.5.3 注意事項](#)”は必ずお読みください)。

### 10.1.5.2 副プログラムのエントリ情報

エントリ情報は、実行するプログラムの構造が動的プログラム構造の場合に必要となります。エントリ情報の指定形式については、“[5.4.2 副プログラムのエントリ情報](#)”を参照してください。

### 10.1.5.3 注意事項

ここでは、動的プログラム構造を使用する場合の注意事項について説明します。

- CALL文、CANCEL文に指定されるプログラム名として日本語を使用する場合、プログラム名がリンカの規則に従っているかどうかは利用者が判断します。なお、ツールによっては日本語をサポートしていないものがあるため、英数字の使用を推奨します。
- CALL文によって呼ばれるプログラムが再び呼び出されたときの状態は、実行用の初期化プログラムを除き、最後に制御を戻したときの状態ですが、CANCEL文の実行後、CALL文により呼び出される場合は初期状態に戻されます。
- 1つのプログラムを同時に動的プログラム構造と動的リンク構造で呼び出すと、そのプログラムはCANCEL文により、仮想メモリ上から削除することができなくなります。
- CALL文に一意名を指定した場合、プログラム名として有効となる文字列の最大は、指定された領域の先頭から255バイトまでです。256バイト以降の文字列は無視されます。また、このとき、文字列の後ろに埋められた空白も無視されます。
- データベースを使用しているプログラムに対してCANCEL文を実行する場合は、次の点に注意してください。  
リモートデータベースアクセス機能(ODBC)を利用している場合、CANCEL文に指定された副プログラムでオープンされたままのカーソルはクローズされ、PREPARE文により準備されたSQL文は解放されます。しかし、プリコンパイラを利用している場合、CANCEL文の実行後でも、カーソルはオープンされたままの状態であり、SQL文はPREPARE文により準備されたままの状態です。このため、CANCEL文の実行前にカーソルをクローズし、PREPARE文により準備されているSQL文を解放するようにしてください。
- 動的プログラム構造と単純構造または動的リンク構造が混在している環境でCANCEL文を利用する場合は、次の点に注意してください。

CANCEL文を実行した場合、DLLが仮想メモリから削除されることにより、CANCEL文に指定された副プログラムが呼び出している副プログラムの延長上で、オープンされたファイルおよびカーソルがオープンされたままになることがあります。この場合の動作については保証されませんので、副プログラムでオープンしたファイルおよびカーソルは、CANCEL文の実行前に必ずクローズしてください。

図10.8 キャンセルされる副プログラムが単純構造の副プログラムを呼び出している場合

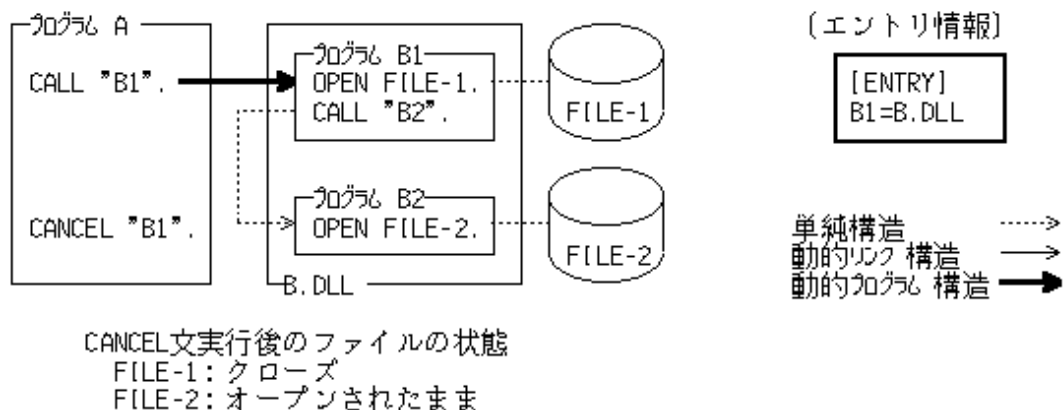
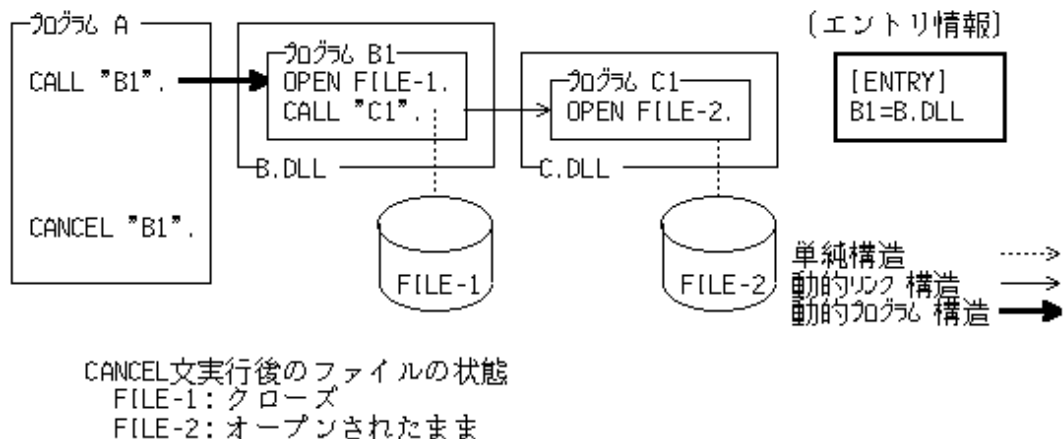


図10.9 キャンセルされる副プログラムが動的リンク構造の副プログラムを呼び出している場合



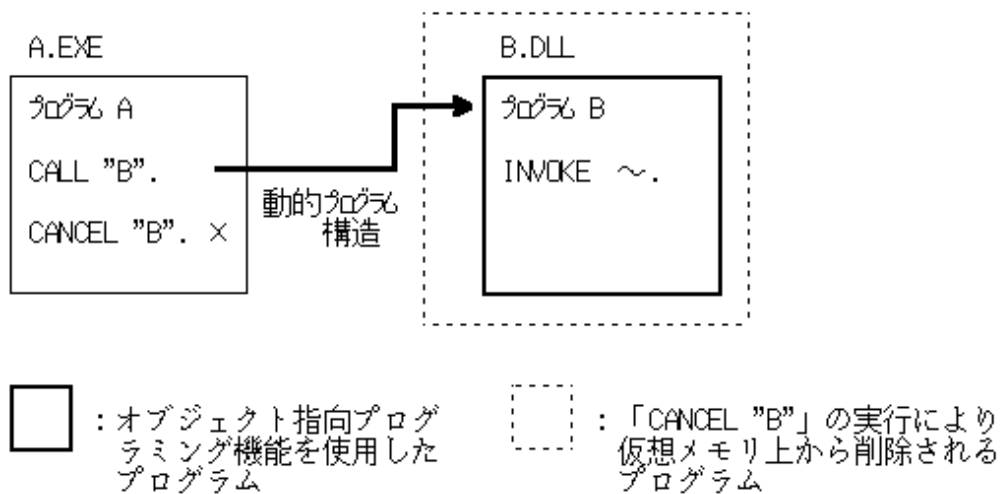
## 参考

この例では「CALL "B1"」の際にB.DLL、C.DLLがロードされ、「CANCEL "B"」の際にB.DLL、C.DLLが仮想メモリ上から削除されます。

- オブジェクト指向プログラミング機能を使用したプログラムをCANCEL文により削除してはいけません。

## 例

[例1] 「CANCEL "B"」の実行により、オブジェクト指向プログラミング機能を使用したプログラムBが仮想メモリ上から削除されるため、このCANCEL文は使用できません。

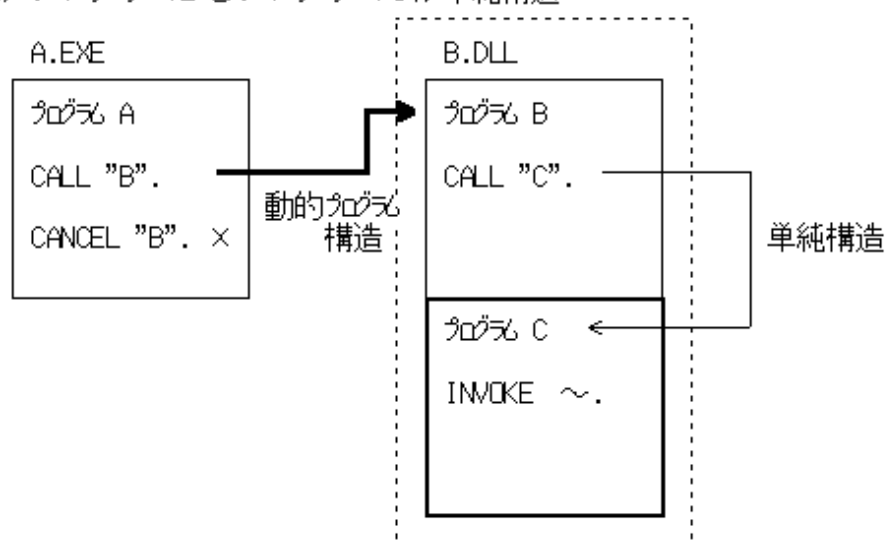


[例2] 以下の場合、CANCEL文の実行により、プログラムBが仮想メモリ上から削除されると、プログラムCも削除されるため、このCANCEL文は使用できません。

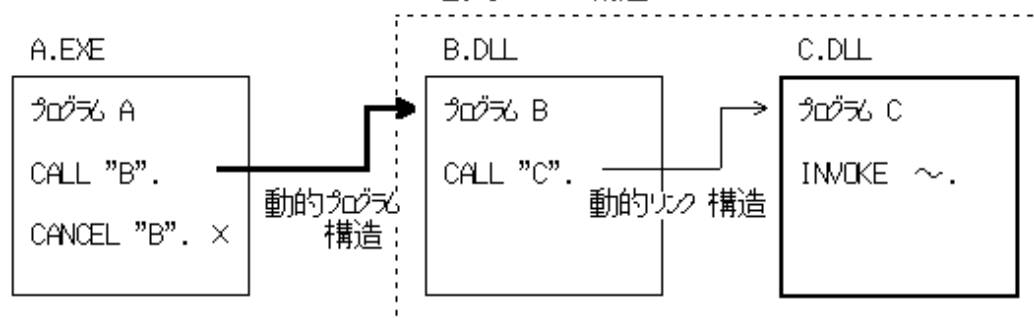
- プログラムBとオブジェクト指向プログラミング機能を使用したプログラムCが単純構造(下図(a))
- 動的リンク構造(下図(b))

この場合、プログラムBとプログラムCの結合を動的プログラム構造(下図(c))に変更することにより、CANCEL文は使用できるようになります。

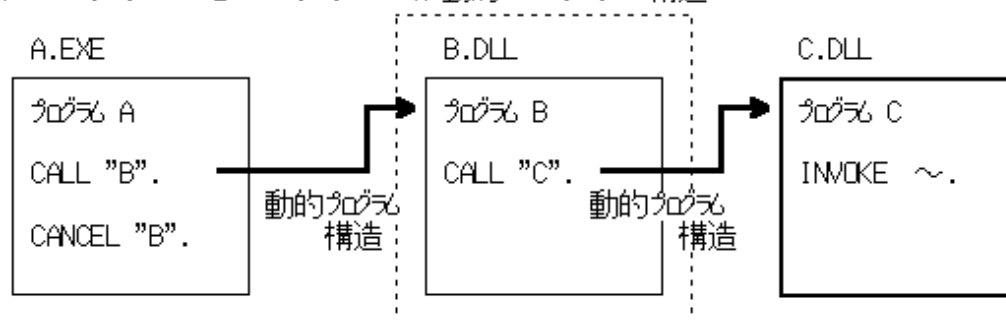
(a) プログラムBとプログラムCが単純構造



(b) プログラムBとプログラムCが動的リンク構造



(c) プログラムBとプログラムCが動的プログラム構造



□ : オブジェクト指向プログラミング機能を使用したプログラム

□ : 「CANCEL "B"」の実行により仮想メモリ上から削除されるプログラム

## 10.2 COBOLプログラムからCOBOLプログラムを呼び出す

ここでは、COBOLプログラム(呼ぶプログラム)から他のCOBOLプログラム(副プログラム)を呼び出す方法について説明します。

## 10.2.1 呼出し方法

---

COBOLプログラムから副プログラムを呼び出すには、副プログラムのプログラム名を指定したCALL文を使います。プログラム名の指定方法は、呼び出す副プログラムの名前がプログラムの作成時に決定するか、プログラムの実行時に決定するかによって、次の2種類があります。

### プログラム作成時に副プログラムの名前が決定している場合

プログラム名定数を使って、CALL文に直接プログラム名を指定します。

### プログラム実行時に副プログラムの名前が決定する場合

CALL文にデータ名を指定し、CALL文を実行する直前にデータ名にプログラム名を設定します。ただし、データ名を指定したCALL文を使用すると、呼ぶプログラムと副プログラムの間のプログラム構造は、動的プログラム構造となります。プログラム構造については、“[4.3 プログラム構造](#)”を参照してください。

## 10.2.2 二次入口点

---

COBOLプログラムでは、手続きの途中で、プログラム呼出しのための入口点を設定することができます。プログラムの手続きの開始点を一次入口点といい、手続きの途中で設定した入口点を二次入口点といいます。プログラム名を指定したCALL文を実行すると、副プログラムは一次入口から実行されます。副プログラムを二次入口から実行するには、CALL文に二次入口点名を、プログラム名を指定するときに同様に指定します。

COBOLプログラムに二次入口点を設定するためには、ENTRY文を記述します。ENTRY文は、プログラムが順次実行されてくる場合には迂回されます。なお、ENTRY文は、内部プログラムに記述することはできません。

## 10.2.3 制御の復帰とプログラムの終了

---

副プログラムから呼ぶプログラムに復帰するには、EXIT PROGRAM文を実行します。EXIT PROGRAM文を実行すると、呼ぶプログラムの実行したCALL文の直後に制御が戻ります。また、すべてのCOBOLプログラムの実行を終了させるには、STOP RUN文を実行します。STOP RUN文を実行すると、COBOLの主プログラムの呼出し元に制御が戻ります。

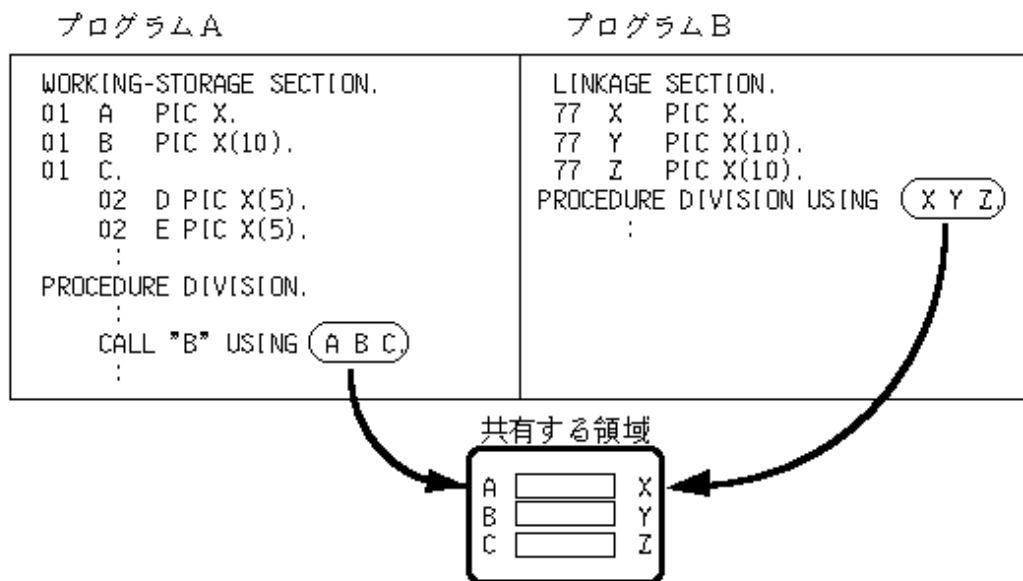
## 10.2.4 パラメタの受渡し方法

---

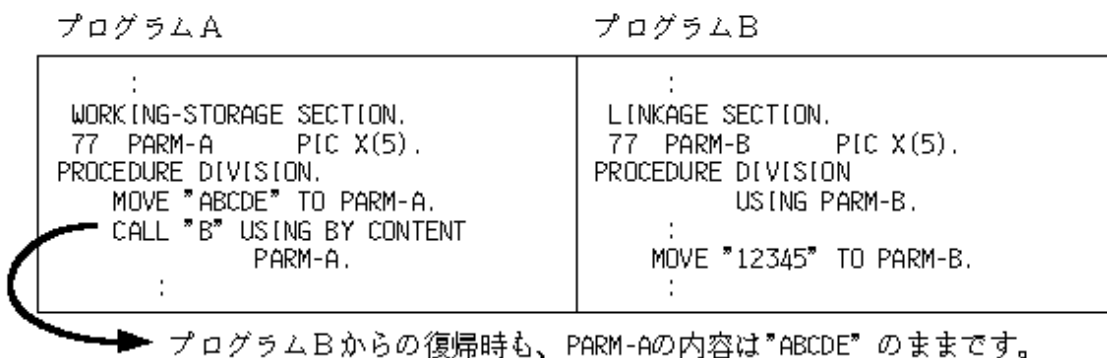
呼ぶプログラムと副プログラムの間で、パラメタを受け渡すことができます。

呼ぶプログラムでは、ファイル節、作業場所節、または連絡節で定義したデータ項目をCALL文のUSING指定に記述します。副プログラムでは、パラメタを受け取るデータ項目を連絡節に定義して、手続き部の見出しまたはENTRY文のUSING指定に記述します。

呼ぶプログラムのCALL文のUSING指定に記述したデータ名の順序が、副プログラムの手続き部の見出しまたはENTRY文のUSING指定で記述したデータ名の順序に対応します。各データ名は、呼ぶプログラムと副プログラムで同じ名前である必要はありません。ただし、対応するデータの属性、長さおよびデータ項目数は同じにします。



なお、呼ぶプログラムにおいて、副プログラムの実行によってパラメタの内容を変更されたくないときには、CALL文のUSING指定に“BY CONTENT データ名”を記述します。



副プログラムで正しくパラメタを受け取るためには、以下が重要です。

- パラメタを受け取るデータ項目を副プログラムの連絡節に定義する。
- 副プログラム側の手続き部の見出しまたはENTRY文のUSING指定にパラメタを受け取るデータ項目を記述する。
- 呼ぶプログラムのCALL文に指定したパラメタの個数と副プログラム側の手続き部の見出しまたはENTRY文のUSING指定に記述したパラメタの個数が一致している、かつ、対応するパラメタの長さが一致している。
- 呼ぶプログラムのCALL文に指定した呼出し規約と副プログラム側の手続き部の見出しまたはENTRY文に指定した呼出し規約が一致する。

これらの記述に誤りがある場合、プログラムを正しく動作させることはできません。プログラムの翻訳時や実行時には、次の範囲でこれらの誤りのチェックを行うことができます。

| チェック項目                       | 翻訳時    | 実行時    |
|------------------------------|--------|--------|
| パラメタ受取り用のデータ項目が連絡節に定義されていない  | ○      | —      |
| パラメタ受取り用のデータ項目のUSING指定への記述漏れ | ○ (*1) | —      |
| パラメタ個数の不一致およびパラメタの長さの不一致     | ○ (*2) | ○ (*2) |



| チェック項目    | 翻訳時 | 実行時    |
|-----------|-----|--------|
| 呼出し規約の不一致 | ×   | ○ (*3) |

\*1：プログラムの手続き部の見出しのUSING指定とENTRY文のUSING指定に記述したパラメタが異なる場合、誤った記述がチェックされない場合があります。

\*2：プログラムの翻訳時に翻訳オプションCHECKの指定が必要です。内部プログラムを呼ぶCALL文は翻訳時にチェックされ、外部プログラムを呼ぶCALL文は実行時にチェックされます。詳しくは、“[19.2 CHECK機能の使い方](#)”を参照してください。

\*3：プログラムの翻訳時に翻訳オプションCHECKの指定が必要です。詳しくは、“[19.2 CHECK機能の使い方](#)”を参照してください。



## 注意

- ・ COBOLプログラムからCOBOLプログラムを呼び出す場合、呼出し側では、“USING BY VALUE”は使用できません。
- ・ パラメタとしてオブジェクト参照項目を受け渡す場合、オブジェクト参照項目のUSAGE句は一致していなければなりません。

## 10.2.5 データの共用

データ記述項またはファイル記述項にEXTERNAL句を指定することで、名前(データ項目名またはファイル名)によって、複数の外部プログラム間でデータ領域を共用することができます。EXTERNAL句が指定されたデータ項目またはファイルは、外部属性を持ちます。外部属性を持つデータ項目を外部データといい、外部属性を持つファイルを外部ファイルといいます。

この機能を用いることにより、プログラム間で引数(パラメタの受渡しの機能)を使用しなくてもデータの受渡しが可能になります。ただし、領域の参照/設定の関係が分かりにくくなるため、利用には注意が必要です。

ファイルのレコード記述項にEXTERNAL句を指定することはできませんが、外部ファイルのレコード記述項に定義されたデータ項目については、外部データになります。

外部データまたは外部ファイルの定義をCOBOL登録集として作成しておき、その登録集をCOPY文でそれぞれのプログラムに取り込むと、保守性のよいプログラムを作成することができます。

## プログラムA

```

SELECT EXFILE
 ASSIGN TO "C:\¥A.DAT"
 FILE STATUS IS EXFS.
.
FILE SECTION.
 COPY EXFILE.
.
WORKING-STORAGE SECTION.
01 EXDATA01 IS EXTERNAL.
 02 A PIC X(5).
01 EXFS IS EXTERNAL.
 02 FS PIC X(2).
.
PROCEDURE DIVISION.
 MOVE SPACE TO A.
 OPEN INPUT EXFILE.
 CALL "B".
.

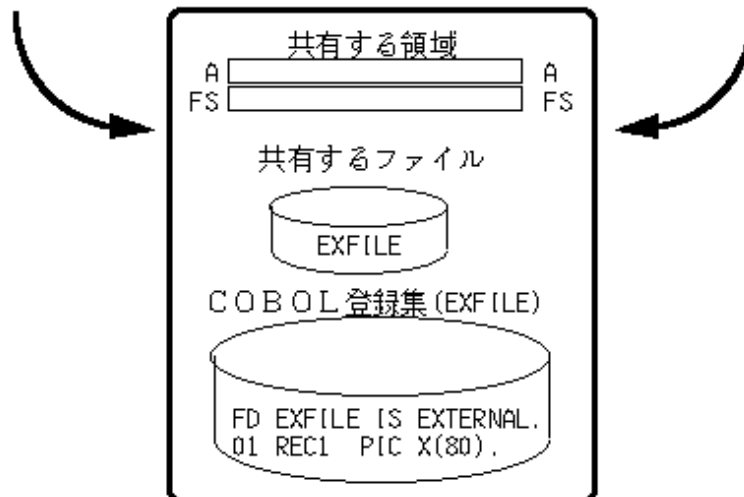
```

## プログラムB

```

SELECT EXFILE
 ASSIGN TO "C:\¥A.DAT"
 FILE STATUS IS EXFS.
.
FILE SECTION.
 COPY EXFILE.
.
WORKING-STORAGE SECTION.
01 EXDATA01 IS EXTERNAL.
 02 A PIC X(5).
01 EXFS IS EXTERNAL.
 02 FS PIC X(2).
.
PROCEDURE DIVISION.
 MOVE "ABCDE" TO A.
 .
 READ EXFILE.
 .

```



### 10.2.5.1 外部データ使用時の注意事項

外部データの同一性のチェックは、最大領域長および最小領域長(可変長データ項目)が対象になります。したがって、外部データが集団項目の場合などは、外部データを構成しているそれぞれのデータ項目の属性はチェックの対象ではないため、不用意に使用すると、データ例外や実行結果の異常などの原因になります。これを避けるためには、COBOL登録集を使用するなどして、データを共用するプログラムの間で同一のレコード構造になるように注意しなければなりません。

また、外部データのデータ領域は、その外部データが記述されたプログラムに一度でも制御が渡った時点で確保され、ランタイムシステムの実行単位の終了時に解放されます。すなわち、外部データのデータ領域はCANCEL文などによりプログラムを消去しても解放されません。このため、繰り返して呼び出されるプログラム内で外部データを使用する場合には、注意が必要です。

### 10.2.5.2 外部ファイル使用時の注意事項

- 外部ファイルは、複数のプログラムで共用できるファイルであり、OPEN文を実行したプログラムとは別のプログラムで入出力処理を行うことができます。
- 外部ファイルは、通常の外部属性を持たないファイルと同様にプログラムで扱うことができますが、外部ファイル特有の注意事項として、「共用するプログラム間でそれぞれ同一の属性で定義されていなければならない」ということがあります。複数のプログラムから1つのファイルを共用するので、必然的に属性の定義を一致させる必要があります。

- 外部ファイルの同一の属性とは、“COBOL文法書”の“4.3.1 ファイル管理段落(FILE-CONTROL)”の一般規則に示される項目の定義を一致させることをいいます。
- 外部ファイルの属性は、同一であるべき項目が多いため、できるだけCOBOL登録集の使用をおすすめします。なぜなら、これらの項目のチェックは実行時に行われるため、最終結合段階にエラーとなり、開発作業の手戻りが発生する可能性があるからです。
- 外部ファイルは、その外部ファイルの記述があるプログラムが一度でも実行されると、外部ファイルのレコード領域および制御用の領域はCANCEL文などによりプログラムを消去しても解放されません。これらの領域が解放されるのは、ランタイムシステムの実行単位の終了時です(通常の外部属性を持たないファイルの場合には、ファイルを記述したプログラムを消去した時点で解放されます)。このため、繰り返して呼び出されるプログラム内で外部ファイルを使用する場合には、注意が必要です。

## 10.2.6 復帰コード

副プログラムから呼ぶプログラムへ制御が戻るときに、RETURNING指定または特殊レジスタPROGRAM-STATUS(またはRETURN-CODE)を使用して、復帰コードを受け渡すことができます。

RETURNING指定は、利用者が定義した項目を使用して復帰コードを受け渡します。呼び出すプログラムのCALL文にRETURNING指定を記述し、副プログラムの手続き部の見出し(PROCEDURE DIVISION)にもRETURNING指定を記述します。RETURNING指定の有無、データの型および長さは、一致していなければなりません。

- プログラムA

```
*> :
 WORKING-STORAGE SECTION.
 01 RTN-ITM PIC S9(2) DISPLAY.
 PROCEDURE DIVISION.
 *> :
 CALL "B" RETURNING RTN-ITM.
 IF RTN-ITM NOT = 0 THEN
 *> :
```

- プログラムB

```
*> :
 LINKAGE SECTION.
 01 RTN-CD PIC S9(2) DISPLAY.
 PROCEDURE DIVISION RETURNING RTN-CD.
 IF エラー発生
 THEN
 MOVE 99 TO RTN-CD
 ELSE
 MOVE 0 TO RTN-CD
 END-IF.
```



### 注意

RETURNING指定が記述されたCALL文では、呼ぶプログラムの特殊レジスタPROGRAM-STATUSの値は変更されないことに注意してください。また、副プログラム側では、RETURNING指定に記述された項目には、値を設定しなければなりません。値が設定されていない場合、呼び出したプログラムのCALL文のRETURNING指定に記述された項目の値は、不定となります。

特殊レジスタPROGRAM-STATUSは、暗に“PIC S9(9) COMP-5”として宣言され、利用者自身がプログラム中で定義する必要はありません。副プログラムが特殊レジスタPROGRAM-STATUSに値を設定すると、その値は呼ぶプログラムの特殊レジスタPROGRAM-STATUSに設定されます。

- プログラムA

```
*> :
 MOVE 0 TO PROGRAM-STATUS.
 CALL "B".
 IF PROGRAM-STATUS NOT = 0 THEN
 *> :
```

- ・ プログラムB

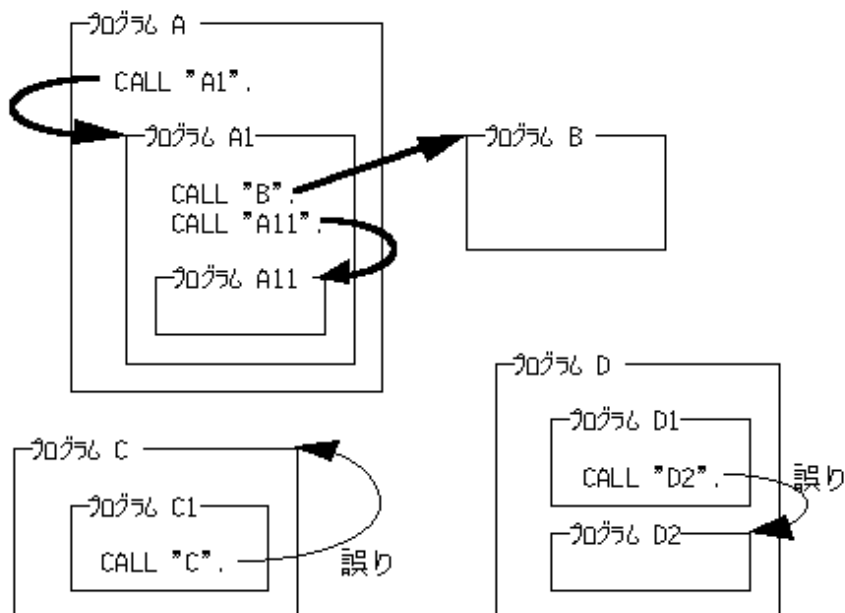
```
*> :
 IF エラー発生
 MOVE 99 TO PROGRAM-STATUS
 EXIT PROGRAM
 END-IF.
```

特殊レジスタPROGRAM-STATUSを下層のプログラムから暗黙に上層のプログラムへ引き継ぐようなプログラム構造をとっていた場合、その中間層のプログラムに手続き部の見出しのRETURNING指定で復帰コードを渡すようにすると、PROGRAM-STATUSの引継ぎが行われなくなります。

## 10.2.7 内部プログラム

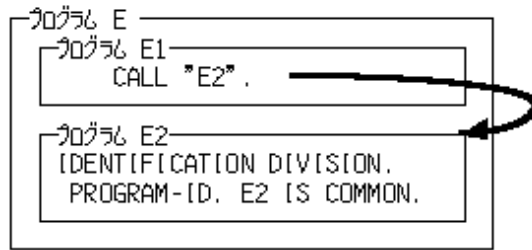
COBOLプログラムは、プログラムの構造の観点から、外部プログラムと内部プログラムに分類されます。他のプログラムに含まれない一番外側のプログラムを外部プログラムといいます。外部プログラムに直接的または間接的に含まれているプログラムを内部プログラムといいます。

外部プログラムからそのプログラムに含まれる内部プログラム(AからA1)を呼び出すことができます。また、内部プログラムから他の外部プログラムやその内部プログラムに含まれる内部プログラム(A1からBやA11)を呼び出すことができます。ただし、内部プログラムから、その内部プログラムの外側にある、共通プログラム以外のプログラム(C1からC、D1からD2)を呼び出すことはできません。



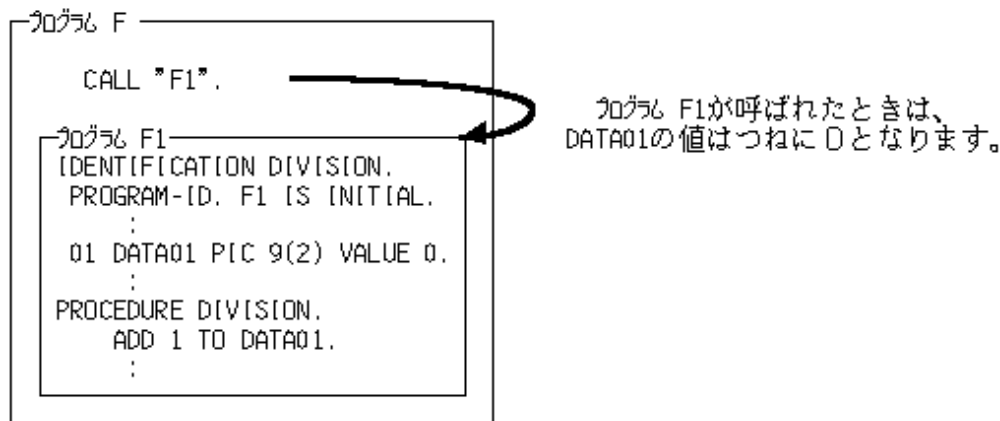
### 共通プログラム

内部プログラムから、その内部プログラムの外側にある内部プログラムを呼び出したい場合、呼ばれる内部プログラムのプログラム名段落にCOMMONを指定します。COMMONを指定したプログラムを共通プログラムといい、COMMONを指定したプログラムを含まない内部プログラムからも呼び出すことができます。



## 初期化プログラム

呼び出されたときに、つねにプログラムを初期状態としたい場合、プログラム名段落にINITIALを指定します。このプログラムを初期化プログラムといいます。初期化プログラムが呼ばれるとき、プログラムの状態はつねに初期状態となります。



## 名前の有効範囲

外側のプログラムで定義したデータ項目をその内部プログラムで使いたい場合には、データ記述項にGLOBAL句を指定します。通常、名前はそのプログラム内でだけ有効となりますが、GLOBAL句を指定したデータ項目は、内部プログラムからも使用することができます。ただし、外側のプログラムは、その内部プログラム中で定義されたGLOBAL指定のデータ項目を使用することはできません。

## 10.2.8 注意事項

- 呼ぶプログラムと副プログラムの両方で翻訳オプションALPHALが有効な場合、プログラム名の文字列は以下のように扱われます。[参照]“[A.2.2 ALPHAL \(英小文字の扱い\)](#)”
  - ー 呼ぶプログラム  
CALL文に定数で指定されたプログラム名は、つねに英大文字のプログラム名として扱われます。
  - ー 副プログラム  
プログラム名段落に記述されたプログラム名は、つねに英大文字として扱われます。
- 呼ぶプログラムおよび副プログラムを翻訳するときには、翻訳オプションALPHALまたはNOALPHALの指定を同じにしてください。とくに、プログラム名に英小文字を使用するときには、翻訳オプションNOALPHALを指定することをおすすめします。
- 主プログラムを翻訳するときには、翻訳オプションMAINを指定する必要があります。また、副プログラムを翻訳するときには、翻訳オプションNOMAINを指定する必要があります。[参照]“[A.2.28 MAIN \(主プログラム/副プログラムの指定\)](#)”

- STDCALL呼出し規約で動的プログラム構造を使用する場合は、以下のようにリンクを行ってください。

ー プログラム MAINPRO

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINPRO.
*> :
DATA DIVISION.
WORKING-STORAGE SECTION.
01 A PIC X.
01 B PIC X(10).
*> :
PROCEDURE DIVISION.
MOVE "A" TO A.
MOVE "parameterB" TO B.
CALL "SUBPRO" WITH STDCALL LINKAGE
USING A B.
STOP RUN.
```

ー プログラム SUBPRO

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUBPRO.
*> :
DATA DIVISION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
01 X PIC X.
01 Y PIC X(10).
PROCEDURE DIVISION WITH STDCALL LINKAGE
USING X Y.
DISPLAY X.
DISPLAY Y.
*> :
EXIT PROGRAM.
```

1. それぞれのプログラムを翻訳します。

プログラムMAINPROを翻訳します。

```
COBOL32 -M -WC, "DLOAD" MAINPRO. COB
```

プログラムSUBPROを翻訳します。

```
COBOL32 SUBPRO. COB
```

2. それぞれのプログラムをリンクします。

プログラムMAINPROをリンクして、実行可能ファイル(EXE)を作成します。

```
LINK MAINPRO. OBJ F3BICIMP. LIB LIBC. LIB /OUT:MAINPRO. EXE
```

プログラムSUBPROをリンクしてDLLを作成します。

```
LINK SUBPRO. OBJ F3BICIMP. LIB LIBC. LIB /EXPORT:SUBPRO /DLL /OUT:SUBPRO. DLL
```

## 10.3 C言語プログラムとの結合

ここでは、COBOLプログラムからC言語で記述されたプログラム(関数)を呼び出す方法、およびC言語で記述されたプログラム(関数)からCOBOLプログラムを呼び出す方法について説明します。なお、ここでは、C言語で記述されたプログラム(関数)を単にCプログラムといいます。

## 10.3.1 COBOLプログラムからCプログラムを呼び出す方法

ここでは、COBOLプログラムからCプログラムを呼び出す方法について説明します。

呼出し規約を省略するとCOBOL呼出し規約となります。

- COBOLプログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 データ名1.
02 要素1 PIC 9(4).
02 要素2 PIC X(10).
01 データ名2 PIC S9(4) COMP-5.
01 データ名3 PIC X(1).
PROCEDURE DIVISION.
CALL 関数名 WITH C LINKAGE
 USING データ名1 データ名2
 BY VALUE データ名3.
IF PROGRAM-STATUS = .
END PROGRAM プログラム名.
```

- Cプログラム

```
typedef struct
{
 char 要素1[4];
 char 要素2[10];
} 構造体名;

long int 関数名(
 構造体名 *仮引数1,
 short int *仮引数2,
 char *仮引数3)
{
 long int 関数値 = 0;
 // :
 return(関数値);
}
```

### 10.3.1.1 呼出し方法

COBOLプログラムからCプログラムを呼び出す場合、COBOLのCALL文に関数名を指定します。呼ばれたCプログラムでreturn文を実行すると、COBOLのCALL文の直後に復帰します。

### 10.3.1.2 パラメタの受渡し方法

COBOLプログラムからCプログラムへパラメタを渡す場合には、CALL文のUSING指定にデータ名を記述します。Cプログラムに渡すパラメタの内容は、領域のアドレスまたはデータ名の内容となります。パラメタは、CALL文のUSING指定に記述します。以下にUSING指定の記述とパラメタの内容の関係を説明します。

#### BY REFERENCE データ名を指定した場合 … 領域のアドレス

COBOLプログラムがCプログラムに渡す実引数の値は、指定したデータ名の領域のアドレスとなります。Cプログラムでは、渡されるパラメタの属性に対応するデータ型を持つポインタを仮引数として宣言します。COBOLとCのデータ型の対応については、“[表10.3 COBOLのデータ項目とCのデータ型との対応例](#)”を参照してください。

#### BY CONTENT データ名(または定数)を指定した場合 … 領域のアドレス

COBOLのプログラムがCプログラムに渡す実引数の値は、指定したデータ名の値が設定された領域のアドレスとなります。Cプログラムでは、渡されるパラメタの属性に対応するデータ型を持つポインタを仮引数として宣言します。Cプログラムで、実引数の指す領域の内容を変更しても、その結果は、COBOLプログラムのデータ名の内容を変更することにはなりません。

## BY VALUE データ名を指定した場合 … 領域の内容

COBOLプログラムがCプログラムに渡す実引数の値は、指定したデータ名の内容となります。Cプログラムで実引数の内容を変更しても、その結果は、COBOLプログラムのデータ名の内容を変更することにはなりません。

## USING指定の記述の違い

ここでは、BY REFERENCE指定とBY CONTENT指定の違いおよびBY REFERENCE指定とBY VALUE指定の違いについてプログラム例を用いて説明します。

### BY REFERENCE指定とBY CONTENT指定の違い

#### ー COBOLプログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINCOB.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 PRM1 PIC S9(9) COMP-5.
01 PRM2 PIC S9(9) COMP-5.
PROCEDURE DIVISION.
 MOVE 10 TO PRM1.
 MOVE 10 TO PRM2.
 CALL "SUBC" WITH C LINKAGE
 USING BY REFERENCE PRM1
 BY CONTENT PRM2.
 DISPLAY "PRM1=" PRM1.
 DISPLAY "PRM2=" PRM2.
```

#### ー Cプログラム

```
#include <windows.h>

long int SUBC(
 long int *p1,
 long int *p2)
{
 *p1 = *p1 +10;
 *p2 = *p2 +10;
 return(0);
}
```

#### ー 実行結果

```
PRM1=+000000020
PRM2=+000000010
```

上記のようなCOBOLプログラムからCプログラムを呼んだ結果は、BY REFERENCE指定のPRM1の内容が20になり、BY CONTENT指定のPRM2の内容は10のままとなります。これは、BY REFERENCE指定で受け渡すパラメタは、呼ばれたプログラムで値を変更した場合、呼ぶプログラムのデータを更新することを意味します。これに対して、BY CONTENT指定で受け渡すパラメタは、呼ばれたプログラムで値を変更しても呼ぶプログラムのデータの内容に影響を与えないことを意味します。上記の説明は、呼ばれるプログラムがCOBOLである場合も同じになります。

### BY REFERENCE指定とBY VALUE指定の違い

#### ー COBOLプログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINCOB.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 PRM1 PIC S9(9) COMP-5.
01 PRM2 PIC S9(9) COMP-5.
PROCEDURE DIVISION.
 MOVE 10 TO PRM1.
```



```

MOVE 10 TO PRM2.
CALL "SUBC" WITH C LINKAGE
 USING BY REFERENCE PRM1
 BY VALUE PRM2.
DISPLAY "PRM1=" PRM1.
DISPLAY "PRM2=" PRM2.

```

#### ー Cプログラム

```

#include <windows.h>

long int SUBC(
 long int *p1,
 long int p2)
{
 *p1 = *p1 +10;
 p2 = p2 +10;
 return(0);
}

```

#### ー 実行結果

```

PRM1=+000000020
PRM2=+000000010

```

BY REFERENCE指定がその項目のアドレスを渡す方法であるのに対して、BY VALUE指定は項目の値そのものを渡すことになります。したがって、BY VALUE指定もBY CONTENT指定と同じように、呼ばれるプログラムでその内容を変更しても呼ぶプログラムのデータの内容は変更することにはなりません。また、呼び出されるCプログラムでは、アドレスで受け取る場合と値で受け取る場合では、その記述に差異があるため、注意が必要です。



#### 注意

BY指定を省略した場合の扱いは、BY REFERENCE指定となります。

### 10.3.1.3 復帰コード(関数値)

Cプログラムから復帰コード(関数値)を受け取るには、CALL文のRETURNING指定または特殊レジスタPROGRAM-STATUSを使います。

#### RETURNING指定

RETURNING指定に記述する項目の属性は、USING指定に記述する項目と同様にCのデータ型と対応がとれている必要があります。データ型の対応については、“[10.3.3 データ型の対応](#)”を参照してください。

- COBOLプログラム

```

WORKING-STORAGE SECTION.
01 AGRP.
 02 AITEM1 PIC X(10).
 02 AITEM2 PIC X(20).
77 B PIC S9(4) COMP-5.
01 RTN-ITM PIC S9(4) COMP-5.
PROCEDURE DIVISION.
 CALL "C" WITH C LINKAGE
 USING AGRP B
 RETURNING RTN-ITM.
 IF RTN-ITM NOT = 0 THEN
*> :

```

- Cプログラム

```
#include <windows.h>

typedef struct
{
 char aitem1[10];
 char aitem2[20];
} agrp;

short int C(agrp *agrpp, short int *b)
{
 return(0);
}
```



## 注意

RETURNING指定に記述する項目の値の受渡しは、データ名の内容渡しとなります。したがって、Cソース側でも集団項目の内容を返却するように記述することで、COBOLプログラムとの整合がとれます。

- COBOLプログラム

```
WORKING-STORAGE SECTION.
01 AGRP.
 02 AITEM1 PIC X(10).
 02 AITEM2 PIC X(20).
77 B PIC S9(4) COMP-5.
01 RTN-ITM.
 02 RITEM1 PIC X(10).
 02 RITEM2 PIC S9(4) COMP-5.
PROCEDURE DIVISION.
 CALL "C" WITH C LINKAGE
 USING AGRP B
 RETURNING RTN-ITM.
 IF RITEM1 = SPACE AND
 RITEM2 = 0 THEN
*> :
```

- Cプログラム

```
#include <windows.h>
#include <string.h>

typedef struct
{
 char aitem1[10];
 char aitem2[20];
} agrp;

typedef struct rtnitm
{
 char ritem1[10];
 short int ritem2;
};

struct rtnitm C (agrp *agrpp, short int *b)
{
 struct rtnitm rttbl;
 strcpy(&(rttbl.ritem1[0]), " ");
 rttbl.ritem2=0;
```

```
 return(rttbl);
}
```

## PROGRAM-STATUS

特殊レジスタPROGRAM-STATUSで受け取る場合、Cの関数型は、long int型の関数として記述する必要があります。

### C呼出し規約の場合

#### ー COBOLプログラム

```
WORKING-STORAGE SECTION.
01 AGRP.
 02 AITEM1 PIC X(10).
 02 AITEM2 PIC X(20).
77 B PIC S9(4) COMP-5.
PROCEDURE DIVISION.
 CALL "C" WITH C LINKAGE
 USING AGRP B
 IF PROGRAM-STATUS = 0 THEN
*> :
```

#### ー Cプログラム

```
#include <windows.h>

typedef struct
{
 char aitem1[10];
 char aitem2[20];
} agrp;

long int C(agrpp *agrpp, short int *b)
{
 return(0);
}
```

### STDCALL呼出し規約の場合

#### ー COBOLプログラム

```
WORKING-STORAGE SECTION.
01 AGRP.
 02 AITEM1 PIC X(10).
 02 AITEM2 PIC X(20).
77 B PIC S9(4) COMP-5.
PROCEDURE DIVISION.
 CALL "C" WITH STDCALL LINKAGE
 USING AGRP B
 IF PROGRAM-STATUS = 0 THEN
*> :
```

#### ー Cプログラム

```
#include <windows.h>

typedef struct
{
 char aitem1[10];
 char aitem2[20];
} agrp;

long int WINAPI C(agrpp *agrpp, short int *b) // (注)
```

```
{
 return(0);
}
```

注：関数名の宣言が以下の場合、STDCALL呼出し規約となります。

- int WINAPI C()
- int CALLBACK C()
- int PASCAL C()
- int far pascal C()
- int \_stdcall C()



## 注意

### long int型以外のCプログラムの関数値を受け取る場合

long int型以外の関数値は、CALL文のRETURNING指定で受け取ります。short int型のCプログラムを呼出す場合の例を以下に示します。

- short int型のCプログラム呼出し

```
 :
01 関数値 PIC S9(4) COMP-5.
 :
 CALL "Cprog" RETURNING 関数値.
 IF 関数値 = 0 THEN ~
 :
```

備考:特殊レジスタPROGRAM-STATUSの属性は、Cのlong int型に対応します。したがって、short int型のCプログラムを呼び出した場合、特殊レジスタPROGRAM-STATUSでは、正しい関数値を受け取ることができません。

### void型のCプログラム呼出しの場合

void型のCプログラムを呼出した場合、特殊レジスタPROGRAM-STATUSが不定な値で更新されてしまいます。特殊レジスタPROGRAM-STATUSが更新されないようにするためには、以下の例に示すようにダミーのデータ項目(PIC S9(9) COMP-5)をRETURNING指定に記述してください。

- void型のCプログラム呼出し

```
 :
01 DUMMY-RET PIC S9(9) COMP-5.
 :
 CALL "Cprog" RETURNING DUMMY-RET.
 :
```

## 10.3.2 CプログラムからCOBOLプログラムを呼び出す方法

ここでは、CプログラムからCOBOLプログラムを呼び出す方法について説明します。

- Cプログラム

```
関数名()
{
 typedef struct
 { char 要素1[4];
 char 要素2[10];
 }構造体名;
 extern void JMPGINT2(), JMPGINT3();
 extern long int プログラム名(構造体名 *, short int *);
```

```

構造体名 実引数1:
short int 実引数2:
:
JMPCINT2();
if (プログラム名(&実引数1, &実引数2))
:
JMPCINT3();
return(0);
}

```

- COBOLプログラム

```

IDENTIFICATION DIVISION.
 PROGRAM-ID. プログラム名.
DATA DIVISION.
LINKAGE SECTION.
01 データ名1.
 02 要素1 PIC 9(4).
 02 要素2 PIC X(10).
77 データ名2 PIC S9(4) COMP-5.
PROCEDURE DIVISION WITH C LINKAGE
 USING データ名1 データ名2.
:
 MOVE 0 TO PROGRAM-STATUS.
 EXIT PROGRAM.
END PROGRAM プログラム名.

```

### 10.3.2.1 呼出し方法

CプログラムからCOBOLプログラムを呼び出すには、Cの関数呼出しの形式でCOBOLのプログラム名を指定します。呼ばれたCOBOLプログラムでEXIT PROGRAM文を実行すると、Cプログラムの関数呼出しの直後に復帰します。

### 10.3.2.2 パラメタの受渡し方法

CプログラムからCOBOLプログラムへ引数を渡す場合には、Cの関数呼出しで実引数を指定します。CプログラムからCOBOLプログラムへ渡すことのできる実引数の値は、記憶領域のアドレスでなければなりません。COBOLプログラムでは、手続き部の見出しまたはENTRY文のUSING指定にデータ名を記述することにより、実引数に指定したアドレスにある領域の内容を受け取ります。実引数で指定したアドレスにある変数の宣言または定義でCONST型指定子を指定した場合、実引数に指定したアドレスにある領域の内容を変更してはいけません。

### 10.3.2.3 復帰コード(関数値)

手続き部の見出し(PROCEDURE DIVISION)のRETURNING指定の項目や特殊レジスタPROGRAM-STATUSに設定した値は、Cプログラムに関数値として渡ります。

### RETURNING指定

手続き部の見出しのRETURNING指定に記述する項目は、Cプログラムのデータ型(下図の[1]および[2])と対応していなければなりません。データ型の対応については、“[10.3.3 データ型の対応](#)”を参照してください。

- Cプログラム

```

C()
{
 typedef struct
 { char aitem1[10];
 char aitem2[20];
 } agrp;
 extern void JMPCINT2(), JMPCINT3();
 extern short int COB(agrp *agrpp, short int *b); // [1]
 agrp prm1;
 short int prm2;
}

```

```

:
JMPINT2();
if (COB(&prm1, &prm2)==0)
:
JMPINT3();
}

```

- COBOLプログラム

```

PROGRAM-ID. COB.
DATA DIVISION.
LINKAGE SECTION.
01 AGRP.
 03 AITEM1 PIC X(10).
 03 AITEM2 PIC X(20).
77 B PIC S9(4) COMP-5.
01 RTN-ITM PIC S9(4) COMP-5. *>[2]
PROCEDURE DIVISION WITH C LINKAGE
 USING AGRP B
 RETURNING RTN-ITM.

:
IF エラー発生
THEN
MOVE 99 TO RTN-ITM.

```



## 注意

- 手続き部の見出しにRETURNING指定を記述すると、特殊レジスタPROGRAM-STATUSに設定した値は、呼び出したCのプログラムには渡りません。
- RETURNING指定に記述する項目の値の受渡しはデータ名の内容渡しとなります。記述する項目が集団項目の場合は、“[10.3.1 COBOLプログラムからCプログラムを呼び出す方法](#)”を参照してください。

## PROGRAM-STATUS

特殊レジスタPROGRAM-STATUSで関数値を渡す場合、Cプログラムでは関数値をlong int型として受け取る必要があります。

- Cプログラム

```

C()
{
typedef struct
{ char aitem1[10];
 char aitem2[20];
} agrp;
extern void JMPINT2(), JMPINT3();
extern long int COB(agrp *agrpp, short int *b);
agrp prm1;
short int prm2;
:
JMPINT2();
if (COB(&prm1, &prm2)==0)
:
JMPINT3();
}

```

- COBOLプログラム

```

PROGRAM-ID. COB.
DATA DIVISION.
LINKAGE SECTION.
01 AGRP.

```

```

03 AITEM1 PIC X(10).
03 AITEM2 PIC X(20).
77 B PIC S9(4) COMP-5.
PROCEDURE DIVISION WITH C LINKAGE
 USING AGRP B.
MOVE 0 TO PROGRAM-STATUS.
EXIT PROGRAM.

```

### 10.3.3 データ型の対応

COBOLプログラムとCプログラム間で受け渡されるデータの属性の組合せは任意です。参考までに、COBOLプログラムとCプログラム間のデータ項目の基本的な対応付けの一例を“表10.3 COBOLのデータ項目とCのデータ型との対応例”に示します。COBOLのデータの内部表現形式については“COBOL文法書”を、Cのデータの内部表現については、C言語のマニュアルを参照してください。



- COBOLの内部ブール項目とCのビットフィールドを使用する場合は、記憶領域の配置に注意する必要があります。COBOLの内部ブール項目の記憶領域の配置については“COBOL文法書”を、Cのビットフィールドの記憶領域の配置については、C言語のマニュアルを参照してください。
- 文字列の受け渡しには、以下の注意が必要です。
  - COBOLプログラムから文字列を受け取る場合は終端文字(¥0)が設定されないため長さを意識して処理する必要があります。
  - COBOLプログラムに文字列を渡す場合は、対応するCOBOLプログラムの変数の領域長に合わせて、空白を詰める必要があります。この時、終端文字(¥0)を設定しないようにしてください。

表10.3 COBOLのデータ項目とCのデータ型との対応例

| COBOL のデータ項目 | Cのデータ型                        | COBOL での記述例                                                              | Cでの宣言例                                             | 大きさ   |
|--------------|-------------------------------|--------------------------------------------------------------------------|----------------------------------------------------|-------|
| 英字／英数字       | char、charの配列型またはstruct (構造体型) | 77 A PIC X.                                                              | char A;                                            | 1バイト  |
|              |                               | 01 B PIC X(20).                                                          | char B[20];                                        | 20バイト |
| 外部10進 (注1)   | charの配列型またはstruct (構造体型)      | 77 C PIC S9(5)<br>SIGN IS LEADING SEPARATE.                              | char C[6];                                         | 6バイト  |
|              |                               | 01 D PIC S9(9)<br>SIGN IS TRAILING SEPARATE.                             | char D[10];                                        | 10バイト |
| 2進 (注2)(注3)  | unsigned char                 | 01 E USAGE IS BINARY-CHAR<br>UNSIGNED.                                   | unsigned char E;                                   | 1バイト  |
|              | short int                     | 01 F PIC S9(4) COMP-5.<br><br>あるいは<br>01 F USAGE IS BINARY-SHORT SIGNED. | short int F;                                       | 2バイト  |
|              | long int                      | 77 G PIC S9(9) COMP-5.<br><br>あるいは<br>77 G USAGE IS BINARY-LONG SIGNED.  | long int G;                                        | 4バイト  |
| 集団項目 (注4)    | char、charの配列型またはstruct (構造体型) | 01 HGRP.<br>02 H1 PIC S9(4) COMP-5.<br>02 H2 PIC X(4).                   | struct{<br>short int H1;<br>char H2[4];<br>} HGRP; | 6バイト  |
| 内部浮動小数点(単精度) | float                         | 01 I COMP-1.                                                             | float I;                                           | 4バイト  |

| COBOL のデータ項目 | C のデータ型 | COBOL での記述例  | C での宣言例   | 大きさ  |
|--------------|---------|--------------|-----------|------|
| 内部浮動小数点(倍精度) | double  | 01 J COMP-2. | double J; | 8バイト |

・ 注1

COBOLでの外部10進項目の内部表現は、符号を表す文字と数字からなる文字列です。したがって、Cプログラムではこれを、数値データとしてではなく文字データとして取り扱います。Cプログラムで、外部10進項目を数値データとして扱いたい場合には、Cプログラムで型変換する必要があります。

・ 注2

USAGE IS COMP-5の2進項目は、その桁数によってCプログラムのshort intまたはlong intに以下のように対応します。

- ー 1～4桁(BINARY(BYTE)オプション指定時は3～4桁) : short int
- ー 5～9桁(BINARY(BYTE)オプション指定時は7～9桁) : long int

ただし、2進項目に小数部がある場合は、次のように浮動小数点を介して受渡しを行ってください。

ー COBOLプログラム

```
WORKING-STORAGE SECTION.
01 H PIC 9V9 COMP-5.
01 FLOAT COMP-1.
PROCEDURE DIVISION.
 MOVE H TO FLOAT.
 CALL "C" WITH C LINKAGE
 USING FLOAT.
```

ー Cプログラム

```
long int C(float *h)
{
 :
 return(0);
}
```

・ 注3

Cプログラムのshort intまたはlong intの値をUSAGE IS COMP-5の項目に受け取る場合、受け取った値がPICTURE句の桁を超えていると、その後の処理において意図した結果が得られない場合があります。そのような場合は、USAGE IS BINARY-SHORT SIGNEDまたはUSAGE IS BINARY-LONG SIGNEDの項目を使用してください。

・ 注4

集団項目を構造体として宣言するときには、その構造体に含まれる変数の記憶領域の境界に注意する必要があります。COBOLのデータ項目の記憶領域の境界調整については、“COBOL文法書”を参照してください。また、Cの変数の記憶領域の境界調整については、C言語のマニュアルを参照してください。

## 参考

- ・ COBOLプログラムから動的プログラム構造で、関数名が以下のように宣言されたCプログラムを呼び出す場合、以下のようにリンクしてください。

```
#include <windows.h>
:
_declspec(dllexport) void WINAPI CSUB() (注)
{
 :
 MessageBox(~);
 :
 return;
}
```



注：関数名の宣言が以下の場合でも同様です。

- `_declspec(dllexport) void PASCAL CSUB()`
- `_declspec(dllexport) void far pascal CSUB()`
- `_declspec(dllexport) void _stdcall CSUB()`

`LINK CSUB.OBJ LIBC.LIB USER32.LIB /DLL /EXPORT:CSUB /OUT:CSUB.DLL`

リンクオプション“/EXPORT”にCプログラムの関数名(Cプログラムの入口名)を指定してリンクしてください。ここでは、/EXPORT オプションに“CSUB”を指定します。

- 以下の場合、「extern "C"」を指定してください。

- Visual C++で作成したC++プログラム(拡張子がCPP,CXX)から、COBOLプログラムを呼び出す場合
  - Cプログラム

```
#include <windows.h>
extern "C" void JMPCINT2();
extern "C" void JMPCINT3();
extern "C" void COBSUB(int *P);
int WINAPI WinMain(~)
{
 int prm1=0;
 :
 JMPCINT2();
 COBSUB(&prm1);
 JMPCINT3();
 :
 return(0);
}
```

- COBOLプログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBSUB.
DATA DIVISION.
LINKAGE SECTION.
01 C-PRM PIC S9(9) COMP-5.
:
PROCEDURE DIVISION USING C-PRM.
:
EXIT PROGRAM.
```

- COBOLプログラムからVisual C++で作成したC++プログラムを呼び出す場合

- COBOLプログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBMAIN.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 COBPRM1.
02 COBPRM1-1 PIC X(10).
02 COBPRM1-2 PIC X(20).
77 COBPRM2 PIC S9(9) COMP-5.
PROCEDURE DIVISION.
:
CALL "CSUB" USING COBPRM1 COBPRM2.
:
```

- Cプログラム

```
#include <windows.h>
#include <stdio.h>
typedef struct
{ char prm1[10];
 char prm2[20];
} argp;

extern "C" void CSUB(argp *argpp, int *b)
{
 :
 return;
}
```

### 10.3.4 プログラムの翻訳

ここでは、プログラムの翻訳時の注意事項について説明します。

- 定数指定のCALL文によってCプログラムを呼び出すとき、そのプログラムを翻訳オプションALPHALを指定して翻訳すると、プログラム名がつねに大文字として扱われ、意図したプログラムが呼び出せない場合があります。したがって、翻訳オプションNOALPHALを指定して翻訳することをおすすめします。[参照]“[A.2.2 ALPHAL \(英小文字の扱い\)](#)”



例

```
CALL "abc".
```

- 翻訳オプションNOALPHALを指定した場合、CALL文を実行するとプログラム"abc"が呼び出されます。
- 翻訳オプションALPHALが有効な場合、CALL文を実行するとプログラム"ABC" が呼び出されます。

```
MOVE "abc" TO A.
CALL A.
```

- 翻訳オプションALPHAL、NOALPHALにかかわらず、プログラム"abc" が呼び出されます。

- CプログラムからCOBOLプログラムを呼び出すとき、呼び出されるCOBOLプログラムを翻訳オプションALPHALを指定して翻訳すると、プログラム名段落に記述したプログラム名がつねに大文字として扱われ、意図したプログラムが呼び出せない場合があります。したがって、翻訳オプションNOALPHALを指定して翻訳することをおすすめします。



例

```
PROGRAM-ID. abc.
```

- 翻訳オプションNOALPHALを指定した場合、プログラム名はabc となります。
- 翻訳オプションALPHALが有効な場合、プログラム名はABC となります。

### 10.3.5 プログラムのリンク

ここでは、呼出し規約およびプログラム構造の違いによるプログラムのリンク方法について説明します。



注意

JMPCINT2またはJMPCINT3を呼び出す場合には、F3BICIMP.LIBが必要になります。

### 10.3.5.1 COBOLプログラムを呼び出すCプログラムの結合方法

以下に、COBOLプログラムを呼び出すCプログラムを、呼出し規約ごとに、単純構造、動的リンク構造および動的プログラム構造で結合する場合の指定例を示します。

プログラム構造については“[4.3 プログラム構造](#)”を参照してください。呼出し規約については“[10.1.2 各種呼出し規約の違い](#)”を参照してください。

#### C呼出し規約で結合する場合

- Cプログラム

```
#include <windows.h>

extern void JMPINT2(), JMPINT3();
extern int COBSUB(int *p);
int WINAPI WinMain(~)
{
 int prm1;
 :
 JMPINT2();
 COBSUB(&prm1);
 JMPINT3();
 :
}
```

- COBOLプログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBSUB.
:
DATA DIVISION.
LINKAGE SECTION.
01 B PIC S9(9) COMP-5.
PROCEDURE DIVISION USING B.
:
EXIT PROGRAM.
```

#### 単純構造で結合する場合

- COBOLプログラムを呼び出すCプログラムと呼び出されるCOBOLプログラムを単純構造で結合する場合

```
LINK Cprog.obj COBSUB.OBJ F3BICIMP.LIB LIBC.LIB Windows関数のインポートライブラリ /OUT:Cprog.EXE
```

Cprog.obj : Cプログラムのオブジェクトファイル  
COBSUB.OBJ : COBOLプログラムCOBSUBのオブジェクトファイル  
F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
LIBC.LIB : Cランタイムライブラリ  
Windows関数のインポートライブラリ : Cプログラムで使用しているWindows関数のインポートライブラリ（例：USER32.LIBなど）



プロジェクトマネージャウィンドウまたはWINLINKウィンドウでDLLのリンクを行う場合、[リンクオプション]ダイアログのDLLのエントリオブジェクトで“言語間結合用”を選択し、Cランタイムライブラリとして“MSVCRT.LIB”を指定します。

#### 動的リンク構造で結合する場合

- Cプログラムから呼び出されるCOBOLプログラムのDLLを作成

```
LINK COBSUB.OBJ F3BICIMP.LIB LIBC.LIB /DLL /OUT:COBSUB.DLL
```

COBSUB.OBJ : COBOL プログラムCOBSUBのオブジェクトファイル  
 F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
 LIBC.LIB : Cランタイムライブラリ

- COBOLプログラムを呼び出すCプログラムの実行可能ファイルを作成

```
LINK Cprog.obj F3BICIMP.LIB LIBC.LIB COBSUB.LIB /OUT:Cprog.EXE
```

Cprog.obj : Cプログラムのオブジェクトファイル  
 F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
 LIBC.LIB : Cランタイムライブラリ  
 COBSUB.LIB : COBOL プログラムCOBSUBのインポートライブラリ

## 動的プログラム構造で結合する場合

CプログラムからCOBOLプログラムを動的プログラム構造で呼び出す場合のCプログラムの例を以下に示します。

```
#include <windows.h>
extern void JMPINT2(), JMPINT3();
int (far *COBSUB)(int *p);
int WINAPI WinMain(~)
{
 int prm1=0;
 HANDLE COBHND;
 :
 COBHND = LoadLibrary("COBSUB.DLL");
 (FARPROC) COBSUB = GetProcAddress(COBHND, "COBSUB");
 JMPINT2();
 COBSUB(&prm1);
 JMPINT3();
 FreeLibrary(COBHND);
 :
}
```

- Cプログラムから呼び出されるCOBOLプログラムのDLLを作成

```
LINK COBSUB.OBJ F3BICIMP.LIB LIBC.LIB /DLL /OUT:COBSUB.DLL
```

COBSUB.OBJ : COBOL プログラムCOBSUBのオブジェクトファイル  
 F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
 LIBC.LIB : Cランタイムライブラリ

- COBOLプログラムを呼び出すCプログラムの実行可能ファイルを作成

```
LINK Cprog.obj F3BICIMP.LIB LIBC.LIB /OUT:Cprog.EXE
```

Cprog.obj : Cプログラムのオブジェクトファイル  
 F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
 LIBC.LIB : Cランタイムライブラリ

## STDCALL呼出し規約で結合する場合

- Cプログラム

```
#include <windows.h>
extern void JMPINT2(), JMPINT3();
extern int WINAPI COBSUB(int *p);
int WINAPI WinMain(~)
{
 int prm1=0;
 :
 JMPINT2();
 COBSUB(&prm1);
 JMPINT3();
}
```

```
}
:
```

- COBOLプログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBSUB.
:
LINKAGE SECTION.
01 B PIC S9(9) COMP-5.
:
PROCEDURE DIVISION WITH STDALL LINKAGE USING B.
:
EXIT PROGRAM.
```

#### 単純構造で結合する場合

- COBOLプログラムを呼び出すCプログラムと呼び出されるCOBOLプログラムを単純構造で結合する場合

```
LINK Cprog.obj COBSUB.OBJ F3BICIMP.LIB LIBC.LIB Windows関数のインポートライブラリ /OUT:Cprog.EXE
```

Cprog.obj : Cプログラムのオブジェクトファイル  
COBSUB.OBJ : COBOLプログラムCOBSUBのオブジェクトファイル  
F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
LIBC.LIB : Cランタイムライブラリ  
Windows関数のインポートライブラリ : Cプログラムで使用しているWindows 関数のインポートライブラリ (例: USER32.LIBなど)



#### 注意

プロジェクトマネージャウィンドウまたはWINLINKウィンドウでDLLのリンクを行う場合、[リンクオプション]ダイアログのDLLのエントリーオブジェクトで“言語間結合用”を選択し、Cランタイムライブラリとして“MSVCRT.LIB”を指定します。

#### 動的リンク構造で結合する場合

- Cプログラムから呼び出されるCOBOLプログラムのDLLを作成

```
LINK COBSUB.OBJ F3BICIMP.LIB LIBC.LIB /DLL /OUT:COBSUB.DLL
```

COBSUB.OBJ : COBOLプログラムCOBSUBのオブジェクトファイル  
F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
LIBC.LIB : Cランタイムライブラリ

- COBOLプログラムを呼び出すCプログラムの実行可能ファイルを作成

```
LINK Cprog.obj F3BICIMP.LIB LIBC.LIB COBSUB.LIB /OUT:Cprog.EXE
```

Cprog.obj : Cプログラムのオブジェクトファイル  
F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
LIBC.LIB : Cランタイムライブラリ  
COBSUB.LIB : COBOLプログラムCOBSUBのインポートライブラリ

#### 動的プログラム構造で結合する場合

CプログラムからCOBOLプログラムを動的プログラム構造で呼び出す場合のCプログラムの例を以下に示します。

```
#include <windows.h>
extern void JMWPCINT2(), JMWPCINT3();
int (far WINAPI *COBSUB)(int *p);
int WINAPI WinMain(~)
{
 int prm1=0;
 HANDLE COBHND;
 :
```

```

COBHND = LoadLibrary("COBSUB.DLL");
(FARPROC) COBSUB = GetProcAddress(COBHND, "COBSUB");
JMPCINT2();
COBSUB(&prm1);
JMPCINT3();
FreeLibrary(COBHND);
:
}

```

- Cプログラムから呼び出されるCOBOLプログラムのDLLを作成

```
LINK COBSUB.OBJ F3BICIMP.LIB LIBC.LIB /DLL /EXPORT:COBSUB /OUT:COBSUB.DLL
```

```

COBSUB.OBJ : COBOLプログラムCOBSUBのオブジェクトファイル
F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ
LIBC.LIB : Cランタイムライブラリ

```

- COBOLプログラムを呼び出すCプログラムの実行可能ファイルを作成

```
LINK Cprog.obj F3BICIMP.LIB LIBC.LIB /OUT:Cprog.EXE
```

```

Cprog.obj : Cプログラムのオブジェクトファイル
F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ
LIBC.LIB : Cランタイムライブラリ

```

### 10.3.5.2 Cプログラムを呼び出すCOBOLプログラムの結合方法

以下に、Cプログラムを呼び出すCOBOLプログラムを、呼出し規約ごとに、単純構造、動的リンク構造、動的プログラム構造で結合する場合の指定例を示します。

プログラム構造については、“[4.3 プログラム構造](#)”を参照してください。呼出し規約については、“[10.1.2 各種呼出し規約の違い](#)”を参照してください。



#### 参考

Cプログラムの外部参照情報の生成は、モジュール定義ファイルのEXPORT文に指定する方法、リンクオプションの/EXPORTオプションに指定する方法、Cプログラムの関数宣言に\_declspec(dllexport)を指定する方法があります。ここではCプログラムの関数宣言に\_declspec(dllexport)を指定して外部参照情報を生成します。

#### C呼出し規約で結合する場合

- COBOLプログラム

```

IDENTIFICATION DIVISION.
PROGRAM-ID. MAINCOB.
:
DATA DIVISION.
WORKING-STORAGE SECTION.
01 B PIC S9(9) COMP-5.
:
PROCEDURE DIVISION.
:
CALL "CSUB" USING B.
:
STOP RUN.

```

- Cプログラム

```

#include <windows.h>
:
_declspec(dllexport)
void CSUB(int *b)

```

```
{
 :
 return;
}
```

#### 単純構造で結合する場合

- Cプログラムを呼び出すCOBOLプログラムと呼び出されるCプログラムを単純構造で結合する場合

```
LINK MAINCOB.OBJ CSUB.OBJ F3BICIMP.LIB LIBC.LIB Windows関数のインポートライブラリ /OUT:MAINCOB.EXE
```

MAINCOB.OBJ : COBOLプログラムMAINCOB のオブジェクトファイル  
 CSUB.OBJ : CプログラムCSUBのオブジェクトファイル  
 F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
 LIBC.LIB : Cランタイムライブラリ  
 Windows関数のインポートライブラリ : Cプログラムで使用しているWindows 関数のインポートライブラリ (例 : USER32.LIBなど)



プロジェクトマネージャウィンドウまたはWINLINKウィンドウでDLLのリンクを行う場合、[リンクオプション]ダイアログのDLLのエントリオブジェクトで“言語間結合用”を選択し、Cランタイムライブラリとして“MSVCRT.LIB”を指定します。

#### 動的リンク構造で結合する場合

- COBOLプログラムから呼び出されるCプログラムのDLLを作成

```
LINK CSUB.OBJ LIBC.LIB Windows関数のインポートライブラリ /DLL /OUT:CSUB.DLL
```

CSUB.OBJ : CプログラムCSUBのオブジェクトファイル  
 LIBC.LIB : Cランタイムライブラリ  
 Windows関数のインポートライブラリ : Cプログラムで使用しているWindows関数のインポートライブラリ (例 : USER32.LIBなど)

- Cプログラムを呼び出すCOBOLプログラムの実行可能ファイルを作成

```
LINK MAINCOB.OBJ F3BICIMP.LIB LIBC.LIB CSUB.LIB /OUT:MAINCOB.EXE
```

MAINCOB.OBJ : COBOLプログラムMAINCOB のオブジェクトファイル  
 F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
 LIBC.LIB : Cランタイムライブラリ  
 CSUB.LIB : CプログラムCSUBのインポートライブラリ

#### 動的プログラム構造で結合する場合

- COBOLプログラムから呼び出されるCプログラムのDLLを作成

```
LINK CSUB.OBJ LIBC.LIB Windows関数のインポートライブラリ /DLL /OUT:CSUB.DLL
```

CSUB.OBJ : CプログラムCSUBのオブジェクトファイル  
 LIBC.LIB : Cランタイムライブラリ  
 Windows関数のインポートライブラリ : Cプログラムで使用しているWindows関数のインポートライブラリ (例 : USER32.LIBなど)

- Cプログラムを呼び出すCOBOLプログラムの実行可能ファイルを作成

```
LINK MAINCOB.OBJ F3BICIMP.LIB LIBC.LIB /OUT:MAINCOB.EXE
```

MAINCOB.OBJ : COBOLプログラムMAINCOB のオブジェクトファイル  
 F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
 LIBC.LIB : Cランタイムライブラリ

## STDCALL呼出し規約で結合する場合

- COBOLプログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINCOB.
:
DATA DIVISION.
WORKING-STORAGE SECTION.
01 B PIC S9(9) COMP-5.
PROCEDURE DIVISION.
:
CALL "CSUB" WITH STDCALL LINKAGE USING B.
:
STOP RUN.
```

- Cプログラム

```
#include <windows.h>

_declspec(dllexport)
void WINAPI CSUB(int *b)
{
:
return;
}
```

### 単純構造で結合する場合

- Cプログラムを呼び出すCOBOLプログラムと呼び出されるCプログラムを単純構造で結合する場合

```
LINK MAINCOB.OBJ CSUB.OBJ F3BICIMP.LIB LIBC.LIB Windows 関数のインポートライブラリ /OUT:MAINCOB.EXE
```

MAINCOB.OBJ : COBOL プログラムMAINCOB のオブジェクトファイル  
CSUB.OBJ : CプログラムCSUBのオブジェクトファイル  
F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
LIBC.LIB : Cランタイムライブラリ  
Windows関数のインポートライブラリ : Cプログラムで使用しているWindows関数のインポートライブラリ (例: USER32.LIBなど)



### 注意

プロジェクトマネージャウィンドウまたはWINLINKウィンドウでDLLのリンクを行う場合、[リンクオプション]ダイアログのDLLのエントリオブジェクトで“言語間結合用”を選択し、Cランタイムライブラリとして“MSVCRT.LIB”を指定します。

### 動的リンク構造で結合する場合

- COBOLから呼び出されるCプログラムのDLLを作成

```
LINK CSUB.OBJ LIBC.LIB Windows関数のインポートライブラリ /DLL /OUT:CSUB.DLL
```

CSUB.OBJ : CプログラムCSUBのオブジェクトファイル  
LIBC.LIB : Cランタイムライブラリ  
Windows関数のインポートライブラリ : Cプログラムで使用しているWindows関数のインポートライブラリ (例: USER32.LIBなど)

- Cプログラムを呼び出すCOBOLプログラムの実行可能ファイルを作成

```
LINK MAINCOB.OBJ F3BICIMP.LIB LIBC.LIB CSUB.LIB /OUT:MAINCOB.EXE
```

MAINCOB.OBJ : COBOL プログラムMAINCOB のオブジェクトファイル  
F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ



LIBC.LIB : Cランタイムライブラリ  
CSUB.LIB : CプログラムCSUBのインポートライブラリ

#### 動的プログラム構造で結合する場合

- COBOLプログラムから呼び出されるCプログラムのDLLを作成

```
LINK CSUB.OBJ LIBC.LIB Windows関数のインポートライブラリ /DLL /EXPORT:CSUB /OUT:CSUB.DLL
```

CSUB.OBJ : CプログラムCSUBのオブジェクトファイル  
LIBC.LIB : Cランタイムライブラリ  
Windows関数のインポートライブラリ  
: Cプログラムで使用しているWindows関数のインポートライブラリ (例: USER32.LIBなど)



“STDCALL呼出し規約”の“動的プログラム構造”で結合する場合、CプログラムのDLLを作成する際、必ずリンクオプション“/EXPORT”に、Cプログラムの関数名(Cプログラムの入口名)を指定してリンクしてください。ここでは、/EXPORTオプションにCSUBを指定します。

- Cプログラムを呼び出すCOBOLプログラムの実行可能ファイルを作成

```
LINK MAINCOB.OBJ F3BICIMP.LIB LIBC.LIB /OUT:MAINCOB.EXE
```

MAINCOB.OBJ : COBOLプログラムMAINCOBのオブジェクトファイル  
F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
LIBC.LIB : Cランタイムライブラリ

### リンク時の注意事項

COBOL97 V50以降の製品で提供されるリンカで生成したインポートライブラリ(以降では新形式のインポートライブラリといいます)は、従来のインポートライブラリ(以降では旧形式のインポートライブラリといいます)よりサイズが小さくなります。

Visual C++ 6.0より前の製品で提供されるリンカでは、新形式のインポートライブラリを利用できません。このため、Visual C++ 6.0より前の製品で提供されるリンカを使用する場合は、旧形式のインポートライブラリを生成する必要があります。旧形式のインポートライブラリの生成方法について、以下に説明します。

#### LINKコマンドでインポートライブラリを作成する場合

LINKコマンドに、/LINK50COMPATオプションを指定してリンクする。

#### LIBコマンドでモジュール定義ファイルからインポートライブラリを作成する場合

LIBコマンドに、/LINK50COMPATオプションを指定してリンクする。

また、以下のように、LIBコマンドに/CONVERTオプションを指定することによって、新しい形式のインポートライブラリを旧形式に変換することができます。

```
LIB /CONVERT 新しい形式のインポートライブラリ名 /OUT:旧形式のインポートライブラリ名
```

## 10.3.6 プログラムの実行

ここでは、プログラムの実行時の注意事項について説明します。

- CプログラムからCOBOLプログラムを呼び出す場合、COBOLプログラムを呼び出す関数の引数にCOBOLの実行時オプションは指定できません(指定しても他の引数と同様に扱われ、COBOLの実行時オプションとして有効にはなりません)。
- COBOLプログラムから呼び出されたCプログラムで、exit関数などによりCプログラムを強制終了してはいけません。
- CプログラムからJMPCINT2を使用して呼び出されたCOBOLプログラムで、STOP RUN文を実行してプログラムを終了してはいけません。
- COBOLの文字列の終わりには、C言語の文字列のように自動的にナール文字が挿入されることはありません。

## 10.4 Visual BasicからCOBOLプログラムを呼び出す

ここでは、Visual BasicからCOBOLプログラム(副プログラム)を呼び出す方法について説明します。なお、Visual BasicからCOBOLプログラムを呼び出す例題プログラムがサンプルとして提供されていますので、参考にしてください。

### 10.4.1 呼出し方法

Visual Basicは、他のプログラムをSTDCALL呼出し規約で呼び出します。したがって、Visual BasicからCOBOLを呼び出す場合、呼び出されるCOBOLは、STDCALL呼出し規約で作成しなければなりません。COBOLをSTDCALL呼出し規約で作成するには、手続き部の見出し(PROCEDURE DIVISION)やENTRY文のUSING指定の前に“WITH STDCALL LINKAGE”を指定します。

```
PROCEDURE DIVISION WITH STDCALL LINKAGE
 USING ...

ENTRY 定数－1 WITH STDCALL LINKAGE
 USING ...
```

Visual BasicからCOBOLプログラムを呼び出すときにパラメタを値渡し(ByVal指定)にしないでください。

ただし、Visual Basicから文字列を渡す場合、ByVal指定が必要となります。このときのByVal指定は、“値による引渡し”という意味ではなく、ASCII文字列に変換するという意味となりますので注意してください。

### 10.4.2 データ型の対応

パラメタの受渡しにおいて、データ型を以下のように対応させてください。

| Visual Basic |         |       | COBOL        |
|--------------|---------|-------|--------------|
| データ型         | 型宣言文字列  | 型宣言文字 | PICTURE 句の記述 |
| 整数型          | Integer | %     | S9(4) COMP-5 |
| 長整数型         | Long    | &     | S9(9) COMP-5 |
| 短精度浮動小数点型    | Single  | !     | COMP-1       |
| 倍精度浮動小数点型    | Double  | #     | COMP-2       |
| 文字型          | String  | \$    | X            |

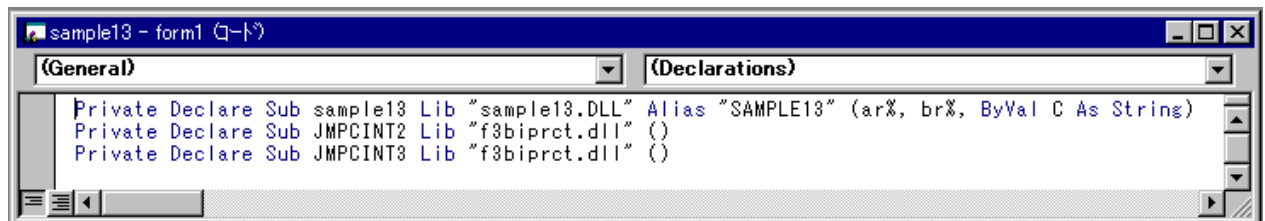
### 10.4.3 Visual BasicからCOBOLの呼出し例

以下に、例題プログラムのサンプルの例を示します。

このプログラムでは、Visual Basicウィンドウを作成し、入力した値をCOBOLアプリケーションに渡し、計算、編集した結果をVisual Basicに返します。

Visual Basicのコード

図10.10 外部プロシージャへの参照の宣言



JMPCINT2およびJMPCINT3については、“[10.1.4 COBOLの言語間の環境](#)”を参照してください。

図10.11 JMPCINT2の呼出し

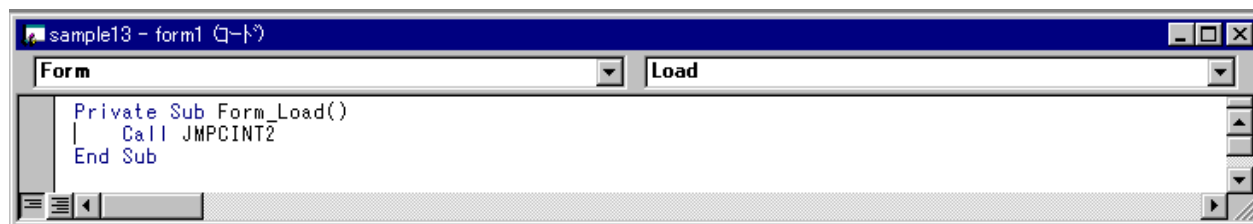


図10.12 JMPCINT3の呼出し

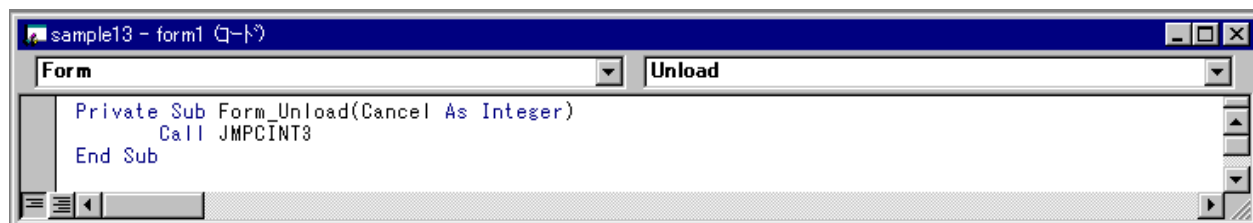
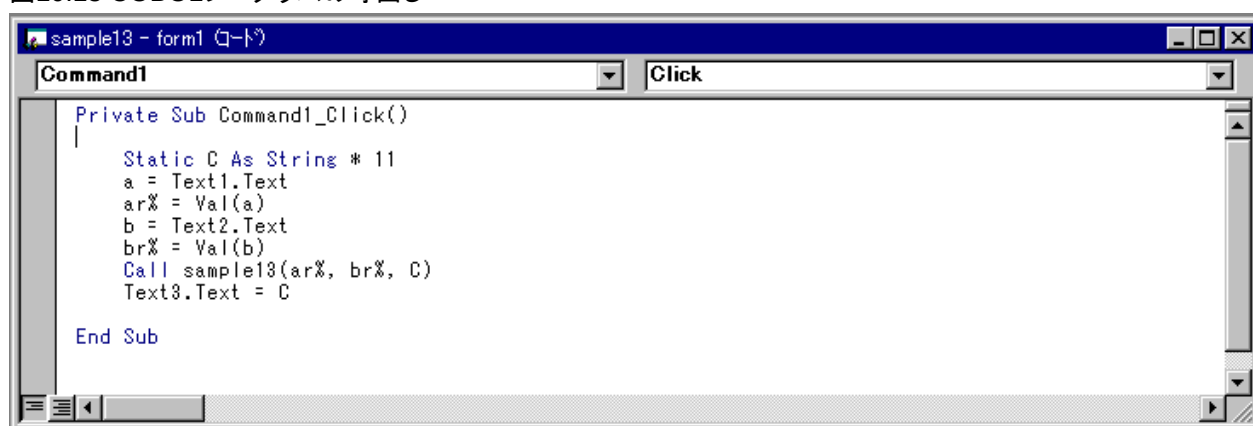


図10.13 COBOLプログラムの呼出し

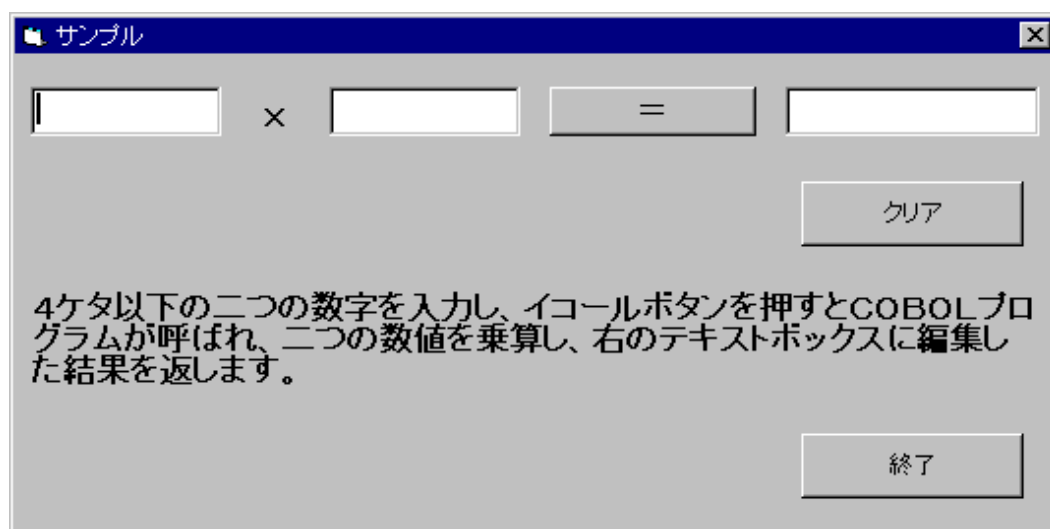


#### COBOLソースプログラム

```

000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. SAMPLE13.
000030 ENVIRONMENT DIVISION.
000040 DATA DIVISION.
000050 WORKING-STORAGE SECTION.
000060 01 WKDATA PIC S9(9).
000070 LINKAGE SECTION.
000080 01 DATA1 PIC S9(4) COMP-5.
000090 01 DATA2 PIC S9(4) COMP-5.
000100 01 DATA3 PIC ZZZ,ZZZ,ZZ9.
000110 PROCEDURE DIVISION WITH STDALL LINKAGE USING DATA1 DATA2 DATA3.
000120 COMPUTE WKDATA = DATA1 * DATA2
000130 MOVE WKDATA TO DATA3
000140 EXIT PROGRAM.

```



サンプル

×  =

クリア

4ケタ以下の二つの数字を入力し、イコールボタンを押すとCOBOLプログラムが呼ばれ、二つの数値を乗算し、右のテキストボックスに編集した結果を返します。

終了

## 第11章 ACCEPT文およびDISPLAY文の使い方

本章では、ACCEPT文およびDISPLAY文を使った機能について説明します。

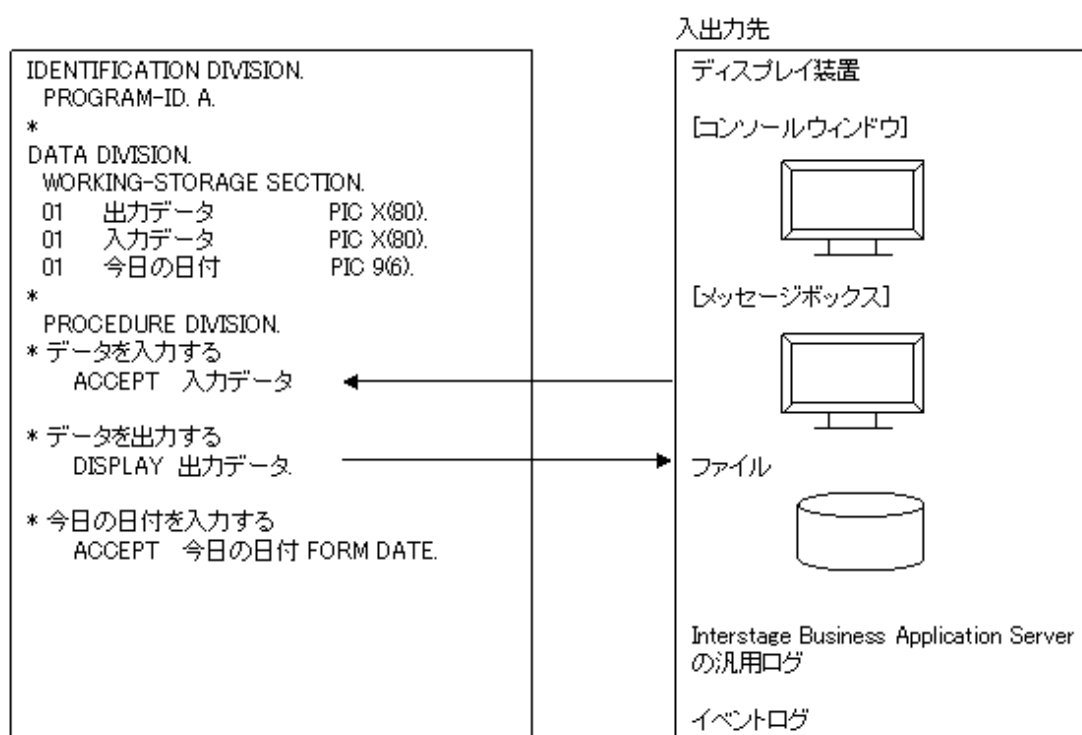
### 11.1 小入出力機能

ここでは、ACCEPT文およびDISPLAY文を使ってデータの入出力を行う、小入出力機能について説明します。なお、小入出力機能を使用した例題プログラムがサンプルプログラムとして提供されていますので、参考にしてください。

#### 11.1.1 概要

小入出力機能では、コンソールウィンドウ、メッセージボックス、Interstage Business Application Serverの汎用ログ、イベントログおよびファイルを使って、データの入出力を簡単に行うことができます。また、システムから現在の日付や時刻を読み込むこともできます。

小入出力機能を使ったデータの入力にはACCEPT文を、データの出力にはDISPLAY文を使います。



#### 11.1.2 入出力先の種類と指定方法

小入出力機能を使ってデータの入出力を行うときの入出力先は、ACCEPT文のFROM指定およびDISPLAY文のUPON指定の記述や、翻訳オプションの指定、環境変数情報の設定内容によって異なります。これらの指定と入出力先の関係を下表に示します。

表11.1 小入出力機能の入出力先

|     | FROM指定またはUPON指定の記述   | 指定する翻訳オプション                     | 環境変数情報の設定内容                         | 入出力先                           |
|-----|----------------------|---------------------------------|-------------------------------------|--------------------------------|
| [1] | ・ なし                 | なし                              | —                                   | ディスプレイ装置(コンソールウィンドウ)(注2)       |
|     | ・ 機能名SYSINに対応付けた呼び名  | SSIN(環境変数情報名)<br>SSOUT(環境変数情報名) | 環境変数情報名にファイル名を設定                    | ファイル(注1)                       |
|     | ・ 機能名SYSOUTに対応付けた呼び名 | なし                              | @CBR_COMPOSER_SY<br>SOUTにログ定義参照名を設定 | 右辺に指定したログ定義参照名の設定に従います(注4)(注5) |

|     | FROM指定またはUPON指定の記述      | 指定する翻訳オプション | 環境変数情報の設定内容                                     | 入出力先                                                   |
|-----|-------------------------|-------------|-------------------------------------------------|--------------------------------------------------------|
|     |                         | なし          | @CBR_DISPLAY_SYSO<br>UT_OUTPUTに<br>EVENTLOGを設定  | イベントログ(注6)(注7)                                         |
| [2] | 機能名SYSERRに対<br>応付けた呼び名  | —           | @MessOutFileに空白を設<br>定                          | ディスプレイ装置(メッセージボックス、コマ<br>ンドプロンプトまたはシステムのコンソール)<br>(注3) |
|     |                         |             | @MessOutFileにファイル名<br>を設定                       | ファイル                                                   |
|     |                         |             | @CBR_COMPOSER_SY<br>SERRにログ定義参照名を<br>設定         | 右辺に指定したログ定義参照名の設定に<br>従います(注5)                         |
|     |                         |             | @CBR_DISPLAY_SYSE<br>RR_OUTPUTに<br>EVENTLOGを設定  | イベントログ(注7)                                             |
| [3] | 機能名CONSOLEに対<br>応付けた呼び名 | —           | —                                               | ディスプレイ装置(コンソールウィンドウ)(注<br>2)                           |
|     |                         |             | @CBR_COMPOSER_CO<br>NSOLEにログ定義参照名<br>を設定        | 右辺に指定したログ定義参照名の設定に<br>従います(注4)(注5)                     |
|     |                         |             | @CBR_DISPLAY_CONS<br>OLE_OUTPUTに<br>EVENTLOGを設定 | イベントログ(注6)(注7)                                         |

- ・ 注1  
翻訳オプションSSIN/SSOUTに、環境変数情報名SYSIN/SYSOUTを指定した場合、ACCEPT文/DISPLAY文の入出力先はコン  
ソールウィンドウとなります。
- ・ 注2  
コンソールウィンドウには、以下があります。
  - COBOLのコンソールウィンドウ(COBOLが独自に生成するウィンドウ)
  - コマンドプロンプト
  - システムのコンソールウィンドウ
 これらのウィンドウは、プログラムの実行開始から終了までを通して、複数同時に使うことはできません。主プログラムの翻訳オプショ  
ンおよび環境変数情報の設定で、使用するウィンドウを指定します。
- ・ 注3  
コンソールウィンドウとして、COBOLのコンソールウィンドウを使用する場合にメッセージボックスへの出力を行います。コマンドプロ  
ンプトまたはシステムのコンソールウィンドウを使用する場合、コマンドプロンプトまたはシステムのコンソールウィンドウの標準エラー  
に出力します。
- ・ 注4  
Interstage Business Application Serverの汎用ログにデータの入力を行うことはできません。
- ・ 注5  
Interstage Business Application Serverの汎用ログに出力する場合、他の出力先の指定は無効になり出力されません。
- ・ 注6  
イベントログにデータの入力を行うことはできません。
- ・ 注7  
イベントログに出力する場合、他の出力先の指定は無効になり出力されません。Interstage Business Application Serverの汎用ログ  
への出力を同時に指定している場合は、汎用ログに出力され、イベントログには出力されません。

### 11.1.3 コンソールウィンドウを使ったデータの入出力

コンソールウィンドウとは、小入出力機能によってディスプレイ装置からデータを読み込んだり、データを表示したりするときに使われるウィンドウです。コンソールウィンドウは、COBOLプログラムの1つの実行単位について1つです。

ここでは、コンソールウィンドウを使うプログラムの記述方法のうち、最も簡単な記述方法について、プログラムの書き方、翻訳・リンクおよび実行方法について説明します。

#### 11.1.3.1 COBOLのコンソールウィンドウ

COBOLのコンソールウィンドウとは、COBOLがGUIベースで独自に生成するウィンドウです。ウィンドウは、実行単位の最初のACCEPT文またはDISPLAY文の実行により創成され、通常実行単位の終了時に消去されます。このウィンドウを使用するためには、主プログラムがCOBOLアプリケーションならば主プログラムの翻訳オプションにMAIN(WINMAIN)を指定します。また、主プログラムがCなどの他言語ならば、環境変数情報@CBR\_CONSOLE=COBOL(省略値)を指定します。



- ・ 翻訳オプションMAINのサブオプションWINMAINは省略値です。また、環境変数情報@CBR\_CONSOLEの設定値SYSTEMは、翻訳オプションMAIN(WINMAIN)を指定した場合および主プログラムがCなど他言語の場合の省略値です。
- ・ 翻訳オプションにMAIN(MAIN)を指定して、環境変数情報@CBR\_CONSOLE=COBOLを指定した場合、アイコン起動などコマンドプロンプト以外から起動すると、起動と同時にシステムのコンソールが生成されます。また、これとは別に、入出力に使用するためのCOBOLのコンソールが生成されます。

COBOLのコンソールウィンドウの属性は、COBOLの実行用の初期化ファイルで変更することができます。変更できる属性および指定方法を下表に示します。

表11.2 コンソールウィンドウの属性の変更

| 属性              | 環境変数情報              | 設定値             | 意味                                          |
|-----------------|---------------------|-----------------|---------------------------------------------|
| ウィンドウの自動消去      | @WinCloseMsg<br>(注) | ON              | メッセージ表示後消去します。                              |
|                 |                     | OFF             | メッセージを表示しないで消去します。                          |
| ウィンドウの大きさ       | @CnslWinSize        | (m,n)           | ウィンドウの大きさを、桁数(m)と行数(n)で指定します。               |
| 保持するデータの行数      | @CnslBufLine        | 桁数              | 指定した行数のデータが保持され、ウィンドウの縦スクロールにより参照することができます。 |
| コンソールウィンドウのフォント | @CnslFont           | (フォント名,フォントサイズ) | コンソールウィンドウのフォントを指定できます。                     |

注：この環境変数情報は、スクリーン操作機能で使用するウィンドウに対しても有効になります。

#### 11.1.3.2 コマンドプロンプト

コマンドプロンプトからアプリケーションを起動した場合、起動したウィンドウをコンソールウィンドウとして使用することができます。また、この場合、リダイレクションを使用することができます。ACCEPT文は標準入力、機能名SYSOUTおよびCONSOLEに対応付けた呼び名を指定したDISPLAY文は標準出力、機能名SYSERRに対応付けた呼び名を指定したDISPLAY文は標準エラーにそれぞれの入出力先が対応付けられます。

コマンドプロンプトを使用する場合は、翻訳オプションまたは環境変数情報の指定が必要です。主プログラムがCOBOLアプリケーションならば主プログラムの翻訳オプションにMAIN(MAIN)を指定します。また、主プログラムがCなどの他言語ならば他言語の入り口関数をmain型(main関数)で作成して、環境変数情報@CBR\_CONSOLE=SYSTEMを指定します。



- ・ 以下の場合、環境変数情報@CBR\_CONSOLE=SYSTEMを指定しても、アプリケーションを起動したプロンプトウィンドウをコンソールウィンドウとして使用することはできません。起動したウィンドウとは別にシステムのコンソールウィンドウが生成されます。
  - ー 主プログラムがCOBOLで翻訳オプションMAIN(WINMAIN)を指定した場合

ー 主プログラムがCなどの他言語でWinMain型(WinMain関数)の場合

- ・ 環境変数情報@CBR\_CONSOLEの設定値COBOLは、翻訳オプションMAIN(MAIN)を指定した場合の省略値です。
- ・ コマンドプロンプトの終了ボタンをクリックすると、ウィンドウが閉じるのと同時に、OPEN中ファイルのCLOSE処理などの資産の回収処理が行われない状態でCOBOLアプリケーションが強制的に終了します。

### 11.1.3.3 システムのコンソールウィンドウ

システムのコンソールウィンドウは、アプリケーションの起動と同時または実行中に生成される、コマンドプロンプトと同様なデザインのウィンドウのことです。ウィンドウの属性について、[コントロールパネル]－“コンソール”により、フォント・レイアウト・画面の色などを指定することができます。この設定はログインユーザ単位の設定ですが、COBOLのコンソールウィンドウと同等の変更ができます。

システムのコンソールウィンドウを使用するためには、主プログラムに翻訳オプションMAIN(WINMAIN)を指定し、環境変数情報@CBR\_CONSOLE=SYSTEMを指定します。また、コマンドプロンプトを使用するための指定と同じ指定を行った実行ファイルを、アイコン起動などコマンドプロンプト以外で起動した場合、コンソールウィンドウとしてシステムのコンソールウィンドウを使用します。



- ・ システムのコンソールウィンドウの終了ボタンをクリックすると、ウィンドウが閉じるのと同時に、OPEN中ファイルのCLOSE処理などの資産の回収処理が行われない状態でCOBOLアプリケーションが強制的に終了します。
- ・ MAIN(WINMAIN)またはMAINオプションで翻訳したCOBOLアプリケーションでシステムのコンソール(コマンドプロンプトを除く)を使用する場合、およびWinMain型の他の言語からCOBOLを呼び出してシステムのコンソール(コマンドプロンプトを除く)を使用する場合、アプリケーションをコマンドプロンプトから起動すると正常に動作しないことがあります。このような場合は、STARTコマンドを使用してください。



START COBOL-APL.EXE

### 11.1.3.4 プログラムの記述

ここでは、プログラムの記述内容について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 データ名 ～.
PROCEDURE DIVISION.
ACCEPT データ名.
DISPLAY データ名.
DISPLAY “文字定数”.
EXIT PROGRAM.
END PROGRAM プログラム名.
```

#### 環境部(ENVIRONMENT DIVISION)

とくに必要な記述はありません。

#### データ部(DATA DIVISION)

入力したデータを格納するためのデータ項目および出力するデータを設定するためのデータ項目を定義します。これらのデータ項目は、次の属性のどれかで定義します。

- ・ 集団項目
- ・ 英字項目



- ・ 英数字項目
- ・ 2進項目
- ・ 内部10進項目
- ・ 外部10進項目
- ・ 英数字編集項目
- ・ 数字編集項目
- ・ 日本語項目(注)
- ・ 日本語編集項目(注)

注 : ACCEPT文では使用できません

## 手続き部(PROCEDURE DIVISION)

コンソールウィンドウからデータを入力するには、ACCEPT文を使います。入力データは、ACCEPT文に指定したデータ名に、そのデータ名に定義した長さ(たとえば、01 入力データ PIC X(80).と定義した場合、英数字が80文字)の分だけ格納されます。なお、入力データの長さが格納するデータの長さより短い場合、文字のときには長さ分のデータが入力されるまで、数字のときには実行キーまたはリターンキーが入力されるまで入力要求が行われます。

コンソールウィンドウにデータを出力するには、DISPLAY文を使います。DISPLAY文にデータ名を指定した場合には、データ名に格納されているデータが出力されます。また、DISPLAY文に文字定数を指定した場合には、指定した文字列が出力されます。

### 11.1.3.5 プログラムの翻訳・リンク

翻訳オプションSSINおよびSSOUTを指定してはいけません。

COBOLのコンソールウィンドウを使用する場合は、主プログラムに翻訳オプションMAIN(WINMAIN)を指定します。

コマンドプロンプトまたはシステムのコンソールウィンドウを使用する場合は、翻訳オプションMAIN(MAIN)を指定します。

### 11.1.3.6 プログラムの実行

通常のプログラムを実行するときと同様に実行してください。ただし、コマンドプロンプトを使用する場合は、コマンドプロンプトで実行してください。

プログラム中のACCEPT文が実行されると、コンソールウィンドウにデータの入力が必要になるので、必要なデータを入力してください。入力データはACCEPT文に指定した長さ分(80バイト)だけそのデータに設定されます。入力したデータの長さがデータ項目の長さより短い場合、文字のときには長さ分のデータが入力されるまで、数字のときには実行キーまたはリターンキーが入力されるまで入力要求が行われます。プログラム中のDISPLAY文が実行されると、コンソールウィンドウにデータが出力されます。



#### 参考

1回の入力データを1回のACCEPT文に対応付ける場合は、機能名CONSOLEを使用します。機能名CONSOLEを使用した場合、入力したデータの長さがデータ項目の長さより短いときには空白詰めが行われます。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. A.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
CONSOLE IS CONS.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 データ1 PIC X(80).
PROCEDURE DIVISION.
ACCEPT データ1 FROM CONS.
DISPLAY データ1 UPON CONS.
```

```
EXIT PROGRAM.
END PROGRAM A.
```



#### 注意

コンソールウィンドウの「閉じる」ボタンをクリックした場合、およびコンソールウィンドウのポップアップメニューから「閉じる」コマンドを選択した場合は、プログラムを強制終了するかどうかを確認するダイアログが表示されます。このダイアログで強制終了を選択した場合は、プログラムを終了します。

## 11.1.4 メッセージボックスにメッセージを出力する方法

ここでは、メッセージボックスにメッセージを出力する方法について説明します。

### 11.1.4.1 メッセージボックス

通常、COBOLプログラムの実行時メッセージは、メッセージボックスを使って表示されます。COBOLプログラムでは、小入出力機能を使って、実行時メッセージと同様にメッセージボックスにメッセージを表示することができます。メッセージボックスは、DISPLAY文の実行により表示され、メッセージボックスの[OK]ボタンをクリックすると消去されます。



#### 注意

コマンドプロンプトまたはシステムのコンソールウィンドウを使用する場合は、使用できません。COBOLのコンソールウィンドウを使用する指定を行ってください。

### 11.1.4.2 プログラムの記述

ここでは、メッセージボックスを使うプログラムの記述内容について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
 SYSERR IS 呼び名.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 データ名 ～.
PROCEDURE DIVISION.
 DISPLAY データ名 UPON 呼び名.
 EXIT PROGRAM.
END PROGRAM プログラム名.
```

#### 環境部(ENVIRONMENT DIVISION)

機能名SYSERRに呼び名を対応付けます。

#### データ部(DATA DIVISION)

出力するデータを設定するためのデータ項目を定義します。これらのデータ項目は、次の属性のどれかで定義します。

- ・ 集団項目
- ・ 英字項目
- ・ 英数字項目
- ・ 外部10進項目
- ・ 英数字編集項目
- ・ 数字編集項目

- ・ 日本語項目
- ・ 日本語編集項目

## 手続き部(PROCEDURE DIVISION)

メッセージボックスにメッセージを出力するには、UPON指定に機能名SYSERRに対応付けた呼び名を記述したDISPLAY文を使います。DISPLAY文にデータ名を指定した場合には、指定したデータ名に格納されているデータが出力され、文字定数を指定した場合には、指定した文字列が出力されます。1回に出力できるメッセージの長さの最大は、メッセージボックスに入るデータの長さとなります。

### 11.1.4.3 プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

### 11.1.4.4 プログラムの実行

通常のプログラムを実行するとと同様に実行します。

データをファイルに出力する場合、環境変数情報@MessOutFileに出力先を指定します。[参照]“[5.4.1.59 @MessOutFile \(メッセージを出力するファイルの指定\)](#)”



#### 注意

- ・ 翻訳オプションMAIN(MAIN)を指定した場合は、実行環境情報@CBR\_CONSOLE=COBOLを指定してください。
- ・ 入出力先に仮想デバイス(NULなど)を使用することはできません。

## 11.1.5 ファイルを使うプログラム

ここでは、小入出力機能を使ってファイル処理を行うプログラムの記述方法のうち、最も簡単な記述方法について、プログラムの書き方、翻訳・リンクおよび実行方法について説明します。

### 11.1.5.1 プログラムの記述

ここでは、小入出力機能を使ってファイル処理を行うプログラムの記述内容について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 データ名 ～.
PROCEDURE DIVISION.
ACCEPT データ名.
DISPLAY データ名.
EXIT PROGRAM.
END PROGRAM プログラム名.
```

## 環境部(ENVIRONMENT DIVISION)

とくに必要な記述はありません。

## データ部(DATA DIVISION)

入力したデータを格納するためのデータ項目および出力するデータを設定するためのデータ項目を定義します。これらのデータ項目は、次の属性のどれかで定義します。

- ・ 集団項目
- ・ 英字項目
- ・ 英数字項目

- ・ 外部10進項目
- ・ 英数字編集項目
- ・ 数字編集項目
- ・ 日本語項目(注)
- ・ 日本語編集項目(注)

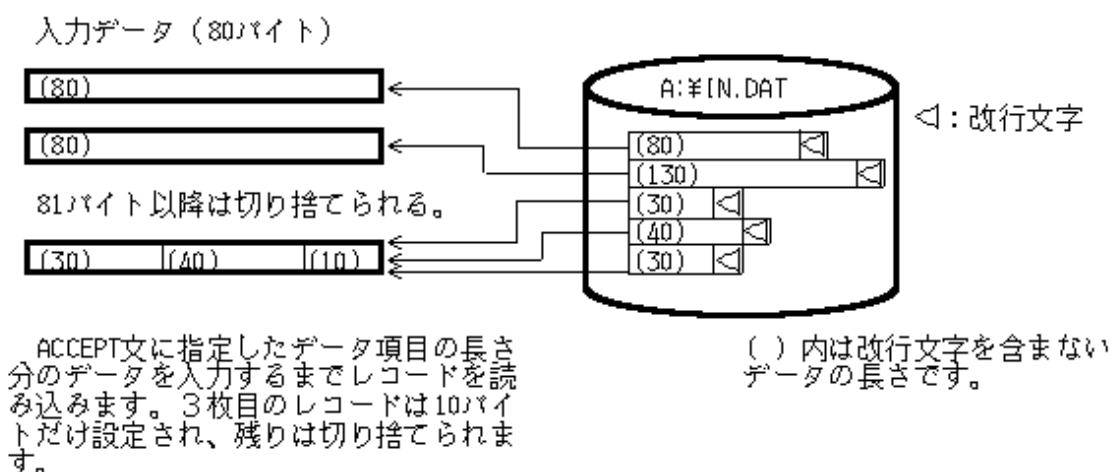
注：ACCEPT文では使用不可

## 手続き部(PROCEDURE DIVISION)

プログラム実行時には、プログラムの実行開始から終了までに動作するプログラム全体の、最初のACCEPT文で入力ファイルが開き、最初のDISPLAY文で出力ファイルが開かれます。2回目以降のACCEPT文およびDISPLAY文では、データの読み込みおよびデータの出力だけが行われます。入力ファイルおよび出力ファイルは、プログラムの実行が終了するときに閉じられます。

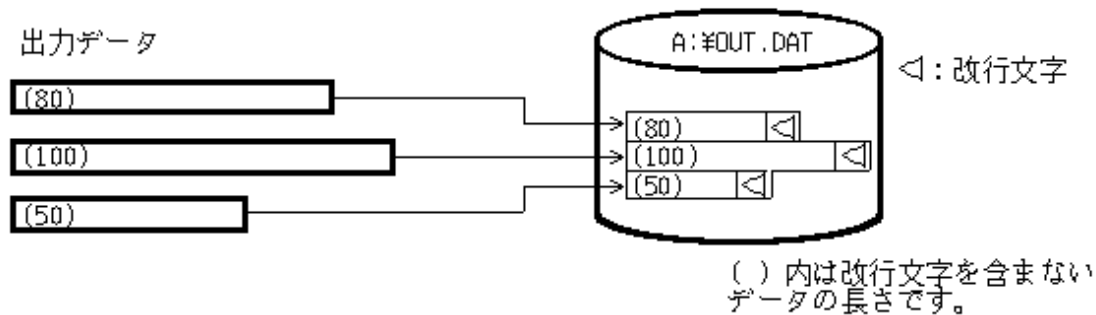
### データの入力

ファイルからデータを入力するには、ACCEPT文を使います。ファイル中のデータは改行文字までを1レコードとし、入力データは1レコードずつ読み込まれます。入力データは、ACCEPT文に指定したデータ名に、そのデータ名に定義した長さ(たとえば、01 入力データ PIC X(80).と定義した場合、英数字が80文字)の分だけ格納されます。なお、入力データの長さが格納するデータの長さより短い場合、次のレコードが読み込まれ、前に読み込まれたデータの後に連結されます。このとき、改行文字はデータとして扱われません。長さ分のデータが入力されるまでレコードの読み込みが行われます。



### データの出力

ファイルにデータを出力するには、DISPLAY文を使います。DISPLAY文にデータ名を指定した場合にはデータ名に格納されている内容が、DISPLAY文に文字定数を指定した場合には指定した文字列が、出力データとなります。1回のDISPLAY文では、出力データの長さに改行文字の長さを加えた長さのデータが出力されます。



### 11.1.5.2 プログラムの翻訳・リンク

小入出力機能のデータの入力先をファイルにする場合には、翻訳オプションSSINを指定します。[参照]“[A.2.52 SSIN \(ACCEPT文のデータの入力先\)](#)”

また、データの出力先をファイルにする場合には、翻訳オプションSSOUTを指定します。[参照]“[A.2.53 SSOUT \(DISPLAY文のデータの出力先\)](#)”



例

SSIN (INDATA) , SSOUT (OUTDATA)



注意

- 翻訳オプションSSINに環境変数情報名としてSYSINを指定した場合、ACCEPT文のデータ入力先は、環境変数情報SYSINの設定にかかわらず、コンソールウィンドウとなります。
- 翻訳オプションSSOUTに環境変数情報名としてSYSOUTを指定した場合、DISPLAY文のデータ出力先は、環境変数情報SYSOUTの指定にかかわらず、コンソールウィンドウとなります。
- FROM指定またはUPON指定の記述に機能名CONSOLEに対応付けた呼び名を指定した場合、データの入出力先は、翻訳オプション(SSIN/SSOUT)の指定にかかわらず、コンソールウィンドウになります。

### 11.1.5.3 プログラムの実行

翻訳オプションSSINおよびSSOUTに指定した環境変数情報名に、入出力処理を行うファイルの名前を設定します。



例

INDATA=A:¥IN.DAT  
OUTDATA=A:¥OUT.DAT



注意

- 入力ファイルは入力モードでオープンされ、共用モードで使用します。また、レコード読み込み時にレコードがロックされることはありません。
- 出力ファイルは出力モードでオープンされ、排他モードで使用します。
- 出力先にすでに存在するファイルを指定した場合、そのファイルは作り直されます。(以前の情報は消去されます。)
- 改行文字はデータとして扱われません。

- ・ ファイルをオープンした後(最初のACCEPT文およびDISPLAY文を実行した後)に、環境変数操作機能を使って入出力先を変更することはできません。
- ・ ファイルの最大サイズおよびレコードの最大長については、“[表7.10 各ファイルシステムの機能差](#)”を参照してください。
- ・ 入出力先には仮想デバイス(NULなど)を使用することはできません。

#### 11.1.5.4 DISPLAY文のファイル出力拡張機能

DISPLAY文のファイル出力では、以下の拡張機能を使用することができます。

- ・ ファイルの追加書き
- ・ ダミーファイル

以下に拡張機能とその使い方について説明します。

##### ファイルの追加書き

DISPLAY文のファイル出力では、既に存在するファイルにデータを追加することができます。

##### 使い方

翻訳オプションSSOUTに指定した環境変数名に、出力処理を行うファイル名に続き、“MOD”を指定します。



例

```
OUTDATA=A:¥OUT.DAT, MOD
```



注意

“MOD”を指定した場合、ファイルが存在すれば、既存ファイルにデータを追加します。ファイルが存在しなければ、新規にファイルを作成します。

##### ダミーファイル

DISPLAY文のファイル出力で、ダミーファイル機能を使用することができます。ダミーファイルの詳細については、“[7.7.7 ダミーファイル](#)”を参照してください。

出力ファイルをダミーファイルにした場合、DISPLAY文の実行は成功しますが、出力ファイルは生成されません。

##### 使い方

翻訳オプションSSOUTに指定した環境変数名に“DUMMY”を指定します。  
ファイル名の指定は任意です。省略してもかまいません。



例

```
OUTDATA=A:¥OUT.DAT, DUMMY
または
OUTDATA=, DUMMY
```



注意

- ・ “DUMMY”の前にコンマ(,)が必要です。コンマ(,)がない場合、文字列“DUMMY”をファイル名とみなして、通常のファイルと同じ動作をします。エラーにならないことに注意してください。

- ・ “,DUMMY”を指定した場合、ファイル名の指定の有無による動作の違いはありません。指定したファイル名は意味を持ちません。指定したファイルが存在した場合も、既存ファイルに対する操作は行われません。

## 共通の注意事項

- ・ ファイル名にコンマ(,)を含む場合、ファイル名を二重引用符(")で囲む必要があります。
- ・ “,MOD”と“,DUMMY”は、同時に指定することができます。指定順序は問いません。ただし、“,DUMMY”と同時に“,MOD”を指定しても、“,MOD”は有効になりません。ダミーファイル機能が指定されたとみなされます。
- ・ コンマ(,)に続き“,MOD”または“,DUMMY”以外の文字列を指定した場合、最初のDISPLAY文実行時でエラーになります。

## 11.1.5.5 ACCEPT文のファイル入力拡張機能

ACCEPT文のファイル入力では、以下の拡張機能を使用することができます。

- ・ ダミーファイル
- ・ スレッド単位でのファイルオープン

以下に拡張機能とその使い方について説明します。

### ダミーファイル

ACCEPT文のファイル入力で、ダミーファイル機能を使用することができます。ダミーファイルの詳細については、“[7.7.7 ダミーファイル](#)”を参照してください。

入力ファイルをダミーファイルにした場合、入力ファイルが存在しなくてもACCEPT文の実行は成功します。

なお、ACCEPT文に指定したデータ項目の値は更新されません。

#### 使い方

翻訳オプションSSINに指定した環境変数名に、“,DUMMY”を指定します。  
ファイル名の指定は任意です。省略してもかまいません。



#### 例

```
INDATA=A:¥IN. DAT, DUMMY
```

または

```
INDATA=, DUMMY
```



#### 注意

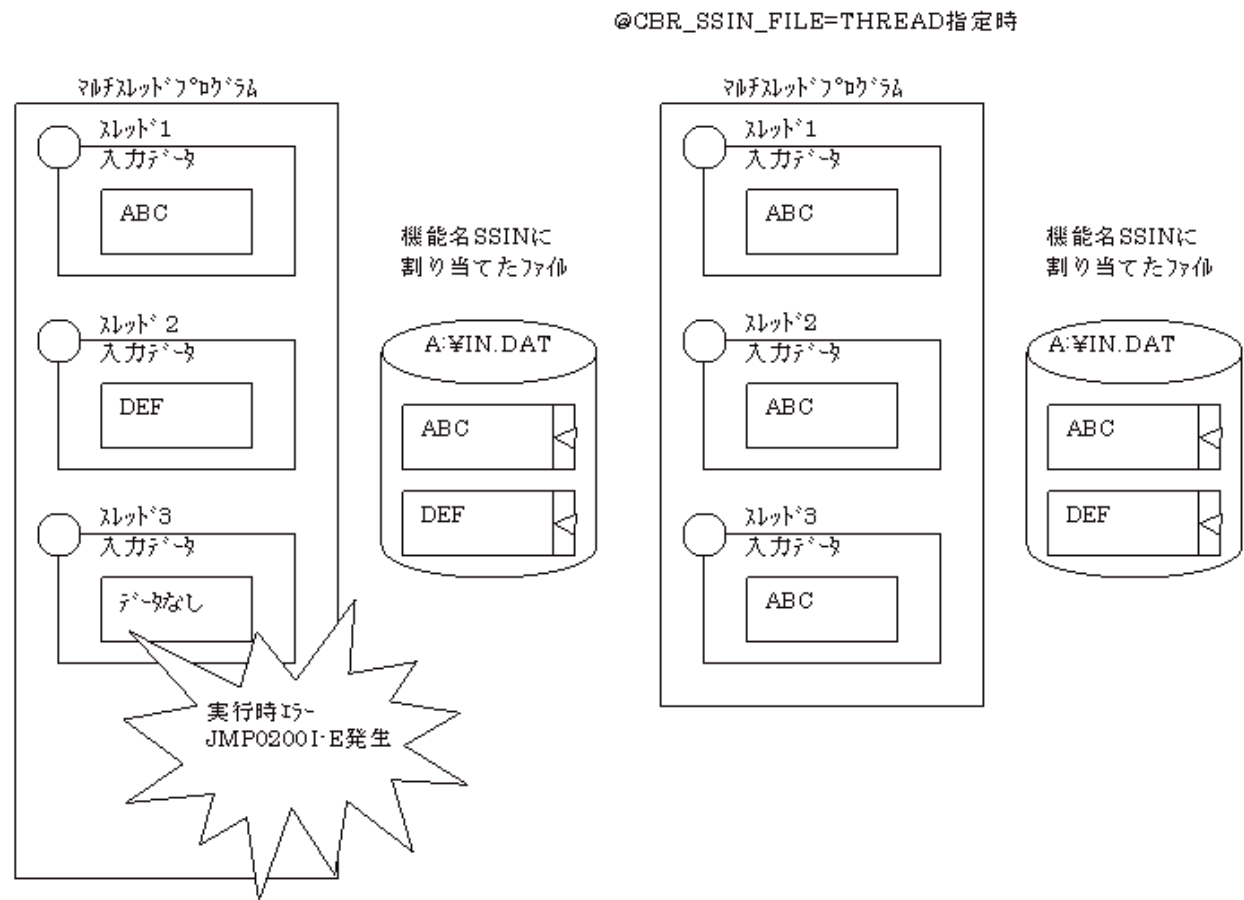
- ・ ファイル名にコンマ(,)を含む場合、ファイル名を二重引用符(")で囲む必要があります。
- ・ 文字列“,DUMMY”の前にコンマ(,)が必要です。コンマ(,)がない場合、文字列“,DUMMY”をファイル名とみなして、通常のファイルと同じ動作をします。
- ・ ファイル名の指定の有無による動作の違いはありません。指定したファイル名は意味を持ちません。
- ・ コンマ(,)に続き“,DUMMY”以外の文字列を指定した場合、最初のACCEPT文の実行でエラーになります。

### スレッド単位でのファイルオープン

ACCEPT文のファイル入力で、スレッド単位に入力ファイルをオープンすることができます。

通常マルチスレッドプログラムでは、プロセスでひとつの入力ファイルを共有します。複数のスレッドからACCEPT文が実行される場合、それらの実行順序は、システムのスレッド制御の順序に依存するため、あるスレッドでは入力データがない状態になることがあります。  
(実行時メッセージJMP0200I-Eを出力)

スレッド単位に入力ファイルをオープンする機能を使うことで、それぞれのスレッドで入力ファイルをオープンし、レコードを入力することができます。



使い方

実行環境変数@CBR\_SSIN\_FILEに、“THREAD”を指定します。[参照]“[24.7.2.2 実行環境情報の指定形式](#)”



例

.....  
@CBR\_SSIN\_FILE=THREAD  
.....

11.1.6 現在の日付および時刻の入力

ここでは、小入出力機能を使ってシステム時計による現在の日付や時刻を入力するプログラムの書き方、翻訳・リンクおよび実行方法について説明します。

11.1.6.1 プログラムの記述

ここでは、小入出力機能を使ってシステム時計による現在の日付や時刻を入力するプログラムの記述内容について、COBOLの各部分ごとに説明します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 年月日 PIC 9(6).
01 年日 PIC 9(5).
01 曜日 PIC 9(1).
```



```

01 時刻 PIC 9(8).
PROCEDURE DIVISION.
ACCEPT 年月日 FROM DATE.
ACCEPT 年日 FROM DAY.
ACCEPT 曜日 FROM DAY-OF-WEEK.
ACCEPT 時刻 FROM TIME.
EXIT PROGRAM.
END PROGRAM プログラム名.

```

## 環境部(ENVIRONMENT DIVISION)

とくに必要な記述はありません。

## データ部(DATA DIVISION)

入力したデータを格納するためのデータ項目を定義します。

## 手続き部(PROCEDURE DIVISION)

現在の日付および時刻を入力するには、FROM指定にDATE(年月日)、DAY(年日)、DAY-OF-WEEK(曜日)またはTIME(時刻)を指定したACCEPT文を使います。

### 11.1.6.2 プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

### 11.1.6.3 プログラムの実行

通常のプログラムを実行するときと同様に実行します。

プログラム中のACCEPT文が実行されると、ACCEPT文に指定したデータ名にそのときの日付および時刻が設定されます。



#### 例

1994年12月23日(金) 14時15分45秒

| FROM句の書き方        | データ名に設定される内容 |
|------------------|--------------|
| FROM DATE        | 941223       |
| FROM DAY         | 94357        |
| FROM DAY-OF-WEEK | 5            |
| FROM TIME        | 14154500     |



#### 参考

CURRENT-DATE関数を利用すれば、西暦を4桁で取得することもできます。詳細については、“[D.3 CURRENT-DATE関数を利用した西暦の取得](#)”を参照してください。

## 11.1.7 任意の日付の入力

ここでは、小入出力を使って任意の日付を入力するプログラムの書き方、翻訳・リンク時の注意事項およびプログラムの実行方法について説明します。

### 11.1.7.1 プログラムの記述

ここでは、小入出力を使って任意の日付を入力するプログラムの記述内容について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 年月日 PIC 9(6).
PROCEDURE DIVISION.
ACCEPT 年月日 FROM DATE.
EXIT PROGRAM.
END PROGRAM プログラム名.
```

## 環境部(ENVIRONMENT DIVISION)

特に必要な記述はありません。

## データ部(DATA DIVISION)

入力したデータを格納するためのデータ項目を定義します。

## 手続き部(PROCEDURE DIVISION)

任意日付を入力するには、FROM指定にDATE(年月日)を指定したACCEPT文を使います。

### 11.1.7.2 プログラムの翻訳・リンク

特に必要な翻訳・リンクオプションはありません。

### 11.1.7.3 プログラムの実行

任意日付を入力するプログラムを実行するときには、以下の環境変数情報の設定が必要です。

```
@CBR_JOBDATE=年.月.日
```

年.月.日は以下のように指定します。

- ・ 年:(00-99)または(1900-2099)
- ・ 月:(01-12)
- ・ 日:(01-31)

“年”の値は、西暦1900年代ならば、西暦年の下2けた又は4けたの西暦年です。西暦2000年代ならば、4けたの西暦年です。



#### 例

任意日付として、1990年10月1日を指定します。

```
@CBR_JOBDATE=90.10.01
```

| FROM句の書き方 | データ名に設定される内容 |
|-----------|--------------|
| FROM DATE | 901001       |

任意日付として、2004年10月1日を指定します。

```
@CBR_JOBDATE=2004.10.01
```

| FROM句の書き方 | データ名に設定される内容 |
|-----------|--------------|
| FROM DATE | 041001       |

## 11.1.8 Interstage Business Application Serverの汎用ログを使うプログラム

ここでは、Interstage Business Application Serverの汎用ログを使うプログラムの書き方、翻訳・リンク時の注意事項およびプログラムの実行方法について説明します。

Interstage Business Application Serverの概要、導入、使用方法については、Interstage Business Application Serverのマニュアルを参照してください。

### 11.1.8.1 プログラムの記述

ここでは、Interstage Business Application Serverの汎用ログを使うプログラムの記述内容について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
 CONSOLE IS 呼び名.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 データ名 ~.
PROCEDURE DIVISION.
 DISPLAY データ名 UPON 呼び名.
 DISPLAY "文字定数" UPON 呼び名.
 EXIT PROGRAM.
END PROGRAM プログラム名.
```

#### 環境部(ENVIRONMENT DIVISION)

機能名SYSOUT、SYSERRまたはCONSOLEに呼び名を対応付けます。

#### データ部(DATA DIVISION)

特に必要な記述はありません。

#### 手続き部(PROCEDURE DIVISION)

Interstage Business Application Serverの汎用ログにデータを出力するには、UPON指定に機能名SYSOUT、SYSERRまたはCONSOLEに対応付けた呼び名を指定したDISPLAY文を使います。DISPLAY文にデータ名を指定した場合には、データ名に格納されているデータが出力されます。また、DISPLAY文に文字定数を指定した場合には、指定した文字列が出力されます。汎用ログはプログラムの実行開始時に開かれ、手続き中のDISPLAY文ではデータの出力だけが行われます。そしてプログラムの実行が終了するときに閉じられます。

### 11.1.8.2 プログラムの翻訳・リンク

特に必要な翻訳・リンクオプションはありません。

### 11.1.8.3 プログラムの実行

出力先にInterstage Business Application Serverの汎用ログを使用するプログラムを実行するときには、DISPLAY文のUPON指定ごとに以下の環境変数の設定が必要です。

UPON指定なしまたは機能名SYSOUTの出力先を汎用ログにする場合

```
@CBR_COMPOSER_SYSOUT = ログ定義ファイルで定義されている管理名
```

機能名SYSERRの出力先を汎用ログにする場合

```
@CBR_COMPOSER_SYSERR = ログ定義ファイルで定義されている管理名
```

機能名CONSOLEの出力先を汎用ログにする場合

```
@CBR_COMPOSER_CONSOLE = ログ定義ファイルで定義されている管理名
```



注意

DISPLAY文の汎用ログへの出力レベルは5です。

## 11.1.9 イベントログを使うプログラム

ここではイベントログを使うプログラムの書き方、翻訳・リンク時の注意事項およびプログラムの実行方法について説明します。

### 11.1.9.1 プログラムの記述

ここでは、イベントログを使うプログラムの記述内容について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
 CONSOLE IS 呼び名.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 データ名 ~.
PROCEDURE DIVISION.
 DISPLAY データ名 UPON 呼び名.
 DISPLAY "文字定数" UPON 呼び名.
 EXIT PROGRAM.
END PROGRAM プログラム名.
```

#### 環境部(ENVIRONMENT DIVISION)

機能名SYSOUT、SYSERRまたはCONSOLEに呼び名を対応付けます。

#### データ部(DATA DIVISION)

出力するデータを設定するためのデータ項目を定義します。

#### 手続き部(PROCEDURE DIVISION)

イベントログにデータを出力するには、UPON指定に機能名SYSOUT、SYSERRまたはCONSOLEに対応付けた呼び名を指定したDISPLAY文を使います。

### 11.1.9.2 プログラムの翻訳・リンク

特に必要な翻訳・リンクオプションはありません。

### 11.1.9.3 プログラムの実行

出力先にイベントログを使用するプログラムを実行するときには、DISPLAY文のUPON指定ごとに以下の環境変数の設定が必要です。

UPON指定なしまたは機能名SYSOUTの出力先をイベントログにする場合

```
@CBR_DISPLAY_SYSOUT_OUTPUT = EVENTLOG(コンピュータ名)
```

機能名SYSERRの出力先をイベントログにする場合

```
@CBR_DISPLAY_SYSERR_OUTPUT = EVENTLOG(コンピュータ名)
```

機能名CONSOLEの出力先をイベントログにする場合

```
@CBR_DISPLAY_CONSOLE_OUTPUT = EVENTLOG(コンピュータ名)
```



## 注意

- 一度に出力できるデータは1024文字です。それ以降は出力されません。
- Interstage Business Application Serverの汎用ログと同時に使用することはできません。
- イベントログに出力する場合、他の出力先(コンソール、ファイル等)の指定は無効となり出力されません。
- イベントログ出力時のイベントソース名は“NetCOBOL Application”となります。イベントソース名を変更する場合は、“イベントログ出力サブルーチン用のレジストリキーの追加・削除”ツール(COBSETER.exe)を使用します。ツールを使用する場合、Administratorsグループのユーザなどレジストリキーのアクセス権(値の照会・値の設定・サブキーの作成・削除)のあるユーザで使用してください。変更後、DISPLAY文のUPON指定ごとに以下の環境変数にイベントソース名を設定してください。

UPON指定なしまたは機能名SYSOUTのイベントソース名を変更する場合

```
@CBR_DISPLAY_SYSOUT_EVENTLOG_SRCNAME = イベントソース名
```

機能名SYSERRのイベントソース名を変更する場合

```
@CBR_DISPLAY_SYSERR_EVENTLOG_SRCNAME = イベントソース名
```

機能名CONSOLEのイベントソース名を変更する場合

```
@CBR_DISPLAY_CONSOLE_EVENTLOG_SRCNAME = イベントソース名
```

- イベントログ出力時のイベント種類は“情報イベント”となります。イベント種類を変更したい場合は、DISPLAY文のUPON指定ごとに以下の環境変数の設定が必要です。

UPON指定なしまたは機能名SYSOUTのイベント種類を変更する場合

```
@CBR_DISPLAY_SYSOUT_EVENTLOG_LEVEL = { I | W | E }
```

機能名SYSERRのイベント種類を変更する場合

```
@CBR_DISPLAY_SYSERR_EVENTLOG_LEVEL = { I | W | E }
```

機能名CONSOLEのイベント種類を変更する場合

```
@CBR_DISPLAY_CONSOLE_EVENTLOG_LEVEL = { I | W | E }
```

レベルは以下を意味します。

I : 情報イベント  
W : 警告イベント  
E : エラーイベント

- ネットワーク上の他のコンピュータに出力することができます。他のコンピュータに出力する場合、出力先のコンピュータに、本製品(COBOLまたはCOBOLランタイムシステム)をインストールする必要があります。
- 他のコンピュータへの出力が失敗した場合、実行中のコンピュータのイベントログに出力対象メッセージを出力します。
- より詳細な情報をイベントログに出力したい場合は、イベントログ出力サブルーチン(COB\_REPORT\_EVENT)を使用することをお勧めします。詳細は“[23.3.2 利用者定義の情報をイベントログに出力する機能](#)”を参照してください。
- 出力先のWindowsシステム上でイベントログへの書き込み権限の設定(ユーザアカウント、ファイアウォール等)が必要な場合があります。設定方法の詳細については、Microsoftのドキュメントを参照してください。



## 例

イベントソース名に"APPLTEST"、イベント種類に"W"、DISPLAY文のデータに"APPL01:START"を指定した場合の出力例



ここでは、プログラムを呼び出したコマンドに指定した引数の数および値を参照する方法について説明します。

プログラム実行中に、プログラムを呼び出したコマンドに指定された引数の数を求めたり、引数の値を参照することができます。引数の数を求めるには、機能名ARGUMENT-NUMBERに対応付けた呼び名を指定したACCEPT文を使います。引数の値を参照するには、機能名ARGUMENT-NUMBERに対応付けた呼び名を指定したDISPLAY文と機能名ARGUMENT-VALUEに対応付けた呼び名を指定したACCEPT文を使います。なお、空白または二重引用符(”)で囲まれた文字列が1つの引数として数えられます。



例

SAMPLE M. HORIUCHI 901234 SHIZUOKA

## 11.2.2 プログラムの記述

ここでは、コマンド行引数の操作機能を使うときのプログラムの記述について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
 ARGUMENT-NUMBER IS 呼び名 1
 ARGUMENT-VALUE IS 呼び名 2.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 データ名 1 ～.
01 データ名 2 ～.
01 データ名 3 ～.
PROCEDURE DIVISION.
 ACCEPT データ名 1 FROM 呼び名 1.
 [DISPLAY 数字定数 UPON 呼び名 1.]
 [DISPLAY データ名 2 UPON 呼び名 1.]
 ACCEPT データ名 3 FROM 呼び名 2
 [ON EXCEPTION ～].
END PROGRAM プログラム名.
```

### 環境部(ENVIRONMENT DIVISION)

次の機能名を呼び名に対応付けます。

- ARGUMENT-NUMBER
- ARGUMENT-VALUE

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
 ARGUMENT-NUMBER IS 呼び名 1
 ARGUMENT-VALUE IS 呼び名 2.
```

### データ部(DATA DIVISION)

値の受渡しを行うためのデータ項目を定義します。

| 内容                  | 属性              |
|---------------------|-----------------|
| 引数の数                | 符号なし整数項目        |
| 引数の位置(定数で指定する場合は不要) | 符号なし整数項目        |
| 引数の値                | 固定長集団項目または英数字項目 |

以下にデータ項目の定義例を示します。

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 引数の数 PIC 9(2) BINARY.
01 引数の位置 PIC 9(2) BINARY.
01 引数.
```



02 引数の値 PIC X(10) OCCURS 1 TO 10 TIMES  
DEPENDING ON 引数の数.

## 手続き部(PROCEDURE DIVISION)

引数の数を求めるには、機能名 ARGUMENT-NUMBER に対応付けた呼び名を指定した ACCEPT 文を使います。引数の数は、ACCEPT 文に指定したデータ名に設定されます。

引数の値を参照するには、まず、機能名 ARGUMENT-NUMBER に対応付けた DISPLAY 文([1])を使って、引数の位置を指定します。次に、機能名 ARGUMENT-VALUE に対応付けた ACCEPT 文([2])を使って、値を取り出します。引数の値は、ACCEPT 文に指定したデータ名に設定されます。このとき、存在しない引数の位置が指定されていた(引数の数が3つなのに引数の位置に4が指定されるなど)場合、例外条件が発生します。例外条件が発生すると、ACCEPT 文に ON EXCEPTION が指定された場合にはそこに記述された文([3])が実行されます。なお、DISPLAY 文を実行しない場合のプログラムの実行開始時には引数の位置は1に位置付けられ、その後、ACCEPT 文を実行するごとに、次の引数に位置付けられます。

引数の位置付けには、0～99を指定することができ、0はコマンド名に位置付けられます。

```
DISPLAY 5 UPON 呼び名 1. ... [1]
ACCEPT 引数の値(5) FROM 呼び名 2 ... [2]
 ON EXCEPTION MOVE 5 TO 誤り番号 ... [3]
 GO TO 誤り処理
END-ACCEPT.
```



- 位置付けのための DISPLAY 文を実行しないで引数の値を参照した場合、プログラム実行開始時に引数の位置は1に位置付けられ、その後 ACCEPT 文を実行するごとに、次の引数に位置付けられます。
- 引数の値の長さを得ることはできません。
- 引数の数および値のデータ項目への設定は、COBOL の MOVE 文の規則が適用されます。

```
 :
01 個数 PIC 9.
01 引数 PIC X(10).
 :
ACCEPT 個数 FROM 引数の番号. ... [1]
ACCEPT 引数 FROM 引数の値. ... [2]
 :
```

コマンドに指定された引数の数が10の場合、[1]を実行すると“個数”の内容は0となります。

取り出す引数の値が“ABCDE”の場合、[2]を実行すると“引数”の内容は以下のようになります。

|   |   |   |   |   |  |  |  |  |  |
|---|---|---|---|---|--|--|--|--|--|
| A | B | C | D | E |  |  |  |  |  |
|---|---|---|---|---|--|--|--|--|--|

空白

取り出す引数の値が“ABCDE12345FGHIJ”の場合、[2]を実行すると“引数”の内容は以下のようになります。

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|

## 11.2.3 プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。



## 11.2.4 プログラムの実行

通常のプログラムを実行するときと同様に実行してください。



- ・ システムから起動したプログラムがCOBOLプログラムの場合だけ使用することができます。
- ・ COBOLプログラムから呼び出されたCOBOLプログラムの場合、参照する引数の値は、システムから起動されたコマンド行に指定した引数の値となります。

## 11.3 環境変数の操作機能

ここでは、環境変数の値を参照・更新する方法について説明します。なお、本節で説明する環境変数とは、プログラム実行時に設定する実行環境情報のことをいいます。

### 11.3.1 概要

プログラムの実行中に、環境変数の値を参照したり、更新したりすることができます。

環境変数の値を参照するには、機能名 ENVIRONMENT-NAME に対応付けた呼び名を指定した DISPLAY 文と、機能名 ENVIRONMENT-VALUE に対応付けた呼び名を指定した ACCEPT 文を使います。環境変数の値を更新するには、機能名 ENVIRONMENT-NAME に対応付けた呼び名を指定した DISPLAY 文と、機能名 ENVIRONMENT-VALUE に対応付けた呼び名を指定した DISPLAY 文を使います。

### 11.3.2 プログラムの記述

ここでは、環境変数の操作機能を使うときのプログラムの記述について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
 ENVIRONMENT-NAME IS 呼び名 1.
 ENVIRONMENT-VALUE IS 呼び名 2.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 データ名 1 ~.
 01 データ名 2 ~.
PROCEDURE DIVISION.
 DISPLAY { "文字定数" | データ名 1 } UPON 呼び名 1.
 ACCEPT データ名 2 FROM 呼び名 2 [ON EXCEPTION ~].
 DISPLAY { "文字定数" | データ名 1 } UPON 呼び名 1.
 DISPLAY データ名 2 UPON 呼び名 2 [ON EXCEPTION ~].
END PROGRAM プログラム名.
```

#### 環境部(ENVIRONMENT DIVISION)

次の機能名を呼び名に対応付けます。

- ・ ENVIRONMENT-NAME
- ・ ENVIRONMENT-VALUE

#### データ部(DATA DIVISION)

値の受渡しを行うためのデータ項目を定義します。

| 内容                    | 属性              |
|-----------------------|-----------------|
| 環境変数名 (定数で指定する場合は不要)  | 固定長集団項目または英数字項目 |
| 環境変数の値 (定数で指定する場合は不要) | 固定長集団項目または英数字項目 |

## 手続き部(PROCEDURE DIVISION)

環境変数の値を参照するには、まず、機能名ENVIRONMENT-NAMEに対応付けた呼び名を指定したDISPLAY文([1])を使って、参照する環境変数名を指定します。次に、機能名ENVIRONMENT-VALUEに対応付けた呼び名を指定したACCEPT文([2])で環境変数の値を参照します。環境変数の値は、ACCEPT文に指定したデータ名に設定されます。ただし、参照する環境変数名が指定されていない場合や、存在しない環境変数名が指定されていた場合、例外条件が発生します。例外条件が発生すると、ACCEPT文にON EXCEPTIONが指定されている場合には、そこに指定された文([3])が実行されます。

環境変数の値を更新するには、まず、機能名ENVIRONMENT-NAMEに対応付けた呼び名を指定したDISPLAY文([4])を使って、更新する環境変数名を指定します。次に、機能名ENVIRONMENT-VALUEに対応付けた呼び名を指定したDISPLAY文で環境変数の値を更新します。更新する環境変数の値は、DISPLAY文([5])に指定したデータ名に設定しておきます。ただし、更新する環境変数名が指定されていない場合や、環境変数の値を設定する領域を割り付けることができなかった場合、例外条件が発生します。例外条件が発生すると、DISPLAY文にON EXCEPTIONが指定されている場合にはそこに指定された文([6])が実行されます。

```

DISPLAY "TMP1" UPON 呼名－環境変数名. ... [1]
ACCEPT TMP 1 の値 FROM 呼名－環境変数の値 ... [2]
ON EXCEPTION ... [3]
 MOVE エラー発生 TO ~
END-ACCEPT.
:
DISPLAY "TMP2" UPON 呼名－環境変数名. ... [4]
DISPLAY TMP 2 の値 UPON 呼名－環境変数の値 ... [5]
ON EXCEPTION ... [6]
 MOVE エラー発生 TO ~
END-DISPLAY.

```



### 注意

環境変数の値の長さを得ることはできません。

## 11.3.3 プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

## 11.3.4 プログラムの実行

通常のプログラムを実行するときと同様に実行してください。



### 注意

プログラムの実行中に変更した環境変数の値は、そのプログラムの実行しているプロセス内でだけ有効となります。

## 第12章 SORT文およびMERGE文の使い方～整列併合機能～

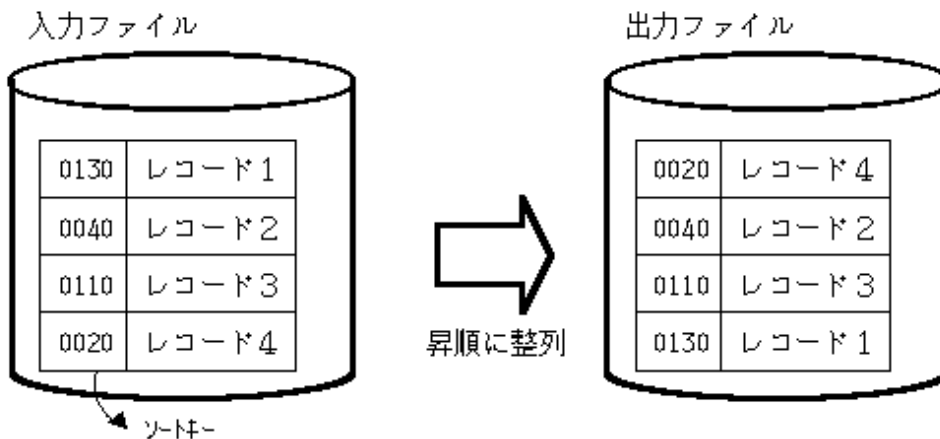
ファイルのレコードを一定の順序に並べ替えることをソート(整列)といい、複数のファイルを1つのファイルに並べ替えることをマージ(併合)といいます。本章では、ソート・マージ(整列・併合)機能について説明します。

### 12.1 ソート・マージ処理の概要

ここでは、ソート(整列)処理と、マージ(併合)処理の概要を説明します。

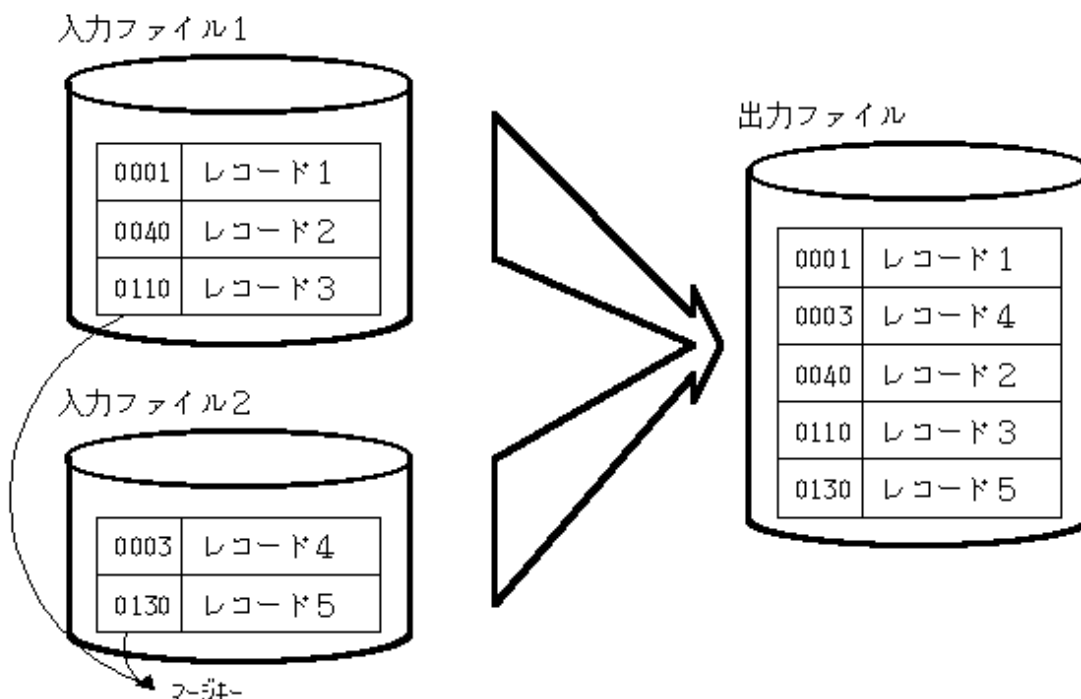
#### ソート

ソートとは、ファイル中のレコードの情報をキーとして、レコードを昇順または降順に並べ替える処理です。レコードの並べ替えは、プログラムのキー項目の属性に従って行われます。



#### マージ

マージとは、昇順または降順に整列(ソート)された複数のファイルを1つのファイルにまとめることです。



## 12.2 ソートの使い方

ここでは、ソート処理の種類、プログラムの書き方、翻訳・リンクおよび実行方法について説明します。

### 12.2.1 ソート処理の種類

ソート処理には、次の4種類の方法があります。

|     | 方法                                   | 入力   | 出力   |
|-----|--------------------------------------|------|------|
| [1] | 入力ファイルのレコードすべてを昇順または降順に出力ファイルに出力する方法 | ファイル | ファイル |
| [2] | 入力ファイルのレコードすべてを昇順または降順に出力データとして扱う方法  | ファイル | レコード |
| [3] | 特定のレコードまたはデータを昇順または降順に出力ファイルに出力する方法  | レコード | ファイル |
| [4] | 特定のレコードまたはデータを昇順または降順に出力データとして扱う方法   | レコード | レコード |

通常、入力ファイル(ソートするファイル)のレコードの内容を変更しないで、そのままソートする場合は、[1]または[2]を使います。入力ファイルを使用しない場合やレコードの内容の変更を行う場合は、[3]または[4]を使います。

また、ソートしたレコードの内容を変更しないで、そのまま出力ファイルに書き出す場合は、[1]または[3]を使います。出力ファイルを使用しない場合やレコードの内容を変更する場合は、[2]または[4]を使います。

### 12.2.2 プログラムの記述

ここでは、ソートを使うプログラムの記述内容について、COBOLの各部ごとに説明します。

```

IDENTIFICATION DIVISION.
 PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 SELECT ソートファイル1 ASSIGN TO SORTWK01.

```

```

[SELECT 入力ファイル 1 ASSIGN TO ファイル参照子 1 ～.]
[SELECT 出力ファイル 1 ASSIGN TO ファイル参照子 2 ～.]
DATA DIVISION.
FILE SECTION.
SD ソートファイル 1
[RECORD レコードの大きさ].
01 ソートレコード 1.
 02 ソートキー 1 ～.
 02 データ 1 ～.
FD 入力ファイル 1 ～.
01 入力レコード 1.
 レコード記述項
FD 出力ファイル 1 ～.
01 出力レコード 1.
 レコード記述項
PROCEDURE DIVISION.
SORT ソートファイル 1 ON
 { ASCENDING | DESCENDING } KEY ソートキー 1
 { USING 入力ファイル 1 | INPUT PROCEDURE IS 入力手続き 1 }
 { GIVING 出力ファイル 1 | OUTPUT PROCEDURE IS 出力手続き 1 } .
入力手続き 1 SECTION.
OPEN INPUT 入力ファイル 1.
入力開始.
READ 入力ファイル 1 AT END GO TO 入力終了.
MOVE 入力レコード 1 TO ソートレコード 1.
RELEASE ソートレコード 1.
GO TO 入力開始.
入力終了.
CLOSE 入力ファイル 1.
入力開始終了.
EXIT.
出力手続き 1 SECTION.
OPEN OUTPUT 出力ファイル 1.
出力開始.
RETURN ソートファイル 1 AT END GO TO 出力終了 END-RETURN.
MOVE ソートレコード 1 TO 出力レコード 1.
WRITE 出力レコード 1.
GO TO 出力開始.
出力終了.
CLOSE 出力ファイル 1.
出力開始終了.
EXIT.
END PROGRAM プログラム名.

```

## 環境部(ENVIRONMENT DIVISION)

以下のファイルを定義します。

### 整列併合用ファイル

ソート処理を行うための作業ファイルを定義します。英字で始まる8文字以内の英数字を指定する必要があります。なお、ASSIGN句は注釈とみなされます。



同じプログラムでマージ処理を行う場合、整列併合用ファイルの定義は1つだけ行います。

### 入力ファイル

ソート処理で入力ファイルを使用するときには、通常のファイル処理を行うときと同様にファイルを定義します。

## 出力ファイル

ソート処理で出力ファイルを使用するときには、通常のファイル処理を行うときと同様にファイルを定義します。

## データ部(DATA DIVISION)

環境部に定義したファイルのレコードを定義します。

## 手続き部(PROCEDURE DIVISION)

ソート処理には、SORT文を使います。ソート処理の入力と出力がファイルかレコードかによって、SORT文の記述内容が異なります。

### 入力がファイルの場合

“USING 入力ファイル名”を記述します。

### 入力がレコードの場合

“INPUT PROCEDURE 入力手続き名”を記述します。

### 出力がファイルの場合

“GIVING 出力ファイル名”を記述します。

### 出力がレコードの場合

“OUTPUT PROCEDURE 出力手続き名”を記述します。

INPUT PROCEDUREで指定した入力手続きでは、RELEASE文を使って、ソートするレコードを1件ずつ受け渡します。

OUTPUT PROCEDUREで指定した出力手続きでは、RETURN文を使って、ソート済みのレコードを1件ずつ受け取ります。

ソートキーは複数指定することができます。

ソート処理が終了すると、ソート処理の結果が特殊レジスタSORT-STATUSに設定されます。特殊レジスタSORT-STATUSは、COBOLコンパイラによって自動的に生成されるので、COBOLプログラムの中で定義する必要はありません。SORT文の実行後に特殊レジスタSORT-STATUSの値を検査することにより、ソート処理が異常終了しても、COBOLプログラムの実行を続行させることができます。また、SORT文で指定した入力手続きまたは出力手続きの中で、SORT-STATUSに16を設定することにより、ソート処理を終了させることもできます。

下表に、特殊レジスタSORT-STATUSに設定される値と意味を示します。

表12.1 SORT-STATUSに設定される値と意味

| 値  | 意味   |
|----|------|
| 0  | 正常終了 |
| 16 | 異常終了 |



## 注意

USING指定の入力ファイルおよびGIVING指定の出力ファイルを使用する場合、SORT文実行時にそれらのファイルが開かれた状態であってはいけません。

特殊レジスタSORT-CORE-SIZEを用いると、PowerSORTが使用するメモリ空間の容量を限定することができます。この特殊レジスタは、暗にPIC S9(8) COMP-5で定義された数字項目です。設定する値は、バイト単位の数値です。

この特殊レジスタは、翻訳オプションSMSIZE()および実行時オプションsmsizeに指定する値の意味と等価ですが、同時に指定された場合の優先順位は、特殊レジスタSORT-CORE-SIZEが一番強く、以降、実行時オプションsmsize、翻訳オプションSMSIZE()の順で弱くなります。



## 例

|          |                               |                  |
|----------|-------------------------------|------------------|
| 特殊レジスタ   | MOVE 102400 TO SORT-CORE-SIZE | (102400=100キロです) |
| 翻訳オプション  | SMSIZE(500K)                  |                  |
| 実行時オプション | smsize300k                    |                  |

この場合、一番強い特殊レジスタSORT-CORE-SIZEの値 100キロバイトを優先します。



この特殊レジスタは、別製品PowerSORTをインストールしている場合に有効であり、インストールしていない場合には無効です。

### 12.2.3 プログラムの翻訳・リンク

必要に応じて翻訳オプションEQUALSを指定します。[参照]“A.2.13 EQUALS (SORT文での同一キーデータの処理方法)”

EQUALSは、ソート処理でソートキーの値が同じレコードが複数存在する場合、出力するレコードの順序をレコードを入力した順序と同じにすることを保証することを指定します。ただし、この翻訳オプションを指定すると実行性能が低下します。

### 12.2.4 プログラムの実行

ソートを使ったプログラムは、以下の手順で実行します。

1. 環境変数BSORT\_TMPDIRの設定

ソート処理では、整列併合用ファイルという作業用ファイルが必要です。整列併合用ファイルは、環境変数BSORT\_TMPDIRに指定したフォルダに一時的に作成されます。環境変数の指定がない場合には、環境変数TEMPに指定したフォルダに一時的に作成されます。なお、これらの環境変数は、あらかじめ設定しておく必要があります。

2. 入力ファイルおよび出力ファイルの割当て

入力ファイルおよび出力ファイルの定義にファイル識別名を使用した場合、ファイル識別名を環境変数情報名として、入力ファイルおよび出力ファイルの名前を設定します。

3. プログラムの実行

プログラムを実行します。



ソートマージ処理は、通常、COBOLランタイムシステムを使用して行いますが、PowerSORTをインストールした場合、PowerSORTを使用します。

PowerSORTを使用する場合の作業用ファイルは、以下の優先順位に従います。

1. 環境変数BSORT\_TMPDIRで指定されたフォルダ
2. 環境変数TEMPで指定されたフォルダ
3. 環境変数TMPで指定されたフォルダ
4. Windowsシステムのフォルダ

## 12.3 マージの使い方

ここでは、マージ処理の種類、プログラムの書き方、翻訳・リンクおよび実行方法について説明します。

### 12.3.1 マージ処理の種類

マージ処理には、次の2種類の方法があります。

|     | 方法                                                      | 入力   | 出力   |
|-----|---------------------------------------------------------|------|------|
| [1] | すでに昇順または降順にソートされた複数のファイルのレコードすべてを、昇順または降順に出力ファイルに出力する方法 | ファイル | ファイル |

|     | 方法                                                     | 入力   | 出力   |
|-----|--------------------------------------------------------|------|------|
| [2] | すでに昇順または降順にソートされた複数のファイルのレコードすべてを、昇順または降順に出力データとして扱う方法 | ファイル | レコード |

通常、マージしたレコードの内容を変更しないで、そのまま出力ファイルに書き出す場合は、[1]を使います。出力ファイルを使用しない場合やレコードの内容を変更する場合は、[2]を使います。

## 12.3.2 プログラムの記述

ここでは、マージを使うプログラムの記述内容について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
 SELECT マージファイル 1 ASSIGN TO SORTWK01.
 SELECT 入力ファイル 1 ASSIGN TO ファイル参照子 1 ～.
 SELECT 入力ファイル 2 ASSIGN TO ファイル参照子 2 ～.
 [SELECT 出力ファイル 1 ASSIGN TO ファイル参照子 3 ～.]
DATA DIVISION.
FILE SECTION.
SD マージファイル 1
[RECORD レコードの大きさ].
01 マージレコード 1.
02 マージキー 1 ～.
02 データ 1 ～.
FD 入力ファイル 1 ～.
01 入力レコード 1.
レコード記述項
FD 入力ファイル 2 ～.
01 入力レコード 2.
レコード記述項
FD 出力ファイル 1 ～.
01 出力レコード 1.
レコード記述項
PROCEDURE DIVISION.
MERGE マージファイル 1 ON
{ ASCENDING | DESCENDING } KEY マージキー 1
USING 入力ファイル 1 入力ファイル 2 ～
{ GIVING 出力ファイル 1 | OUTPUT PROCEDURE IS 出力手続き 1 } .
出力手続き 1 SECTION.
OPEN OUTPUT 出力ファイル 1.
出力開始.
RETURN マージファイル 1 AT END GO TO 出力終了 END-RETURN.
MOVE マージレコード 1 TO 出力レコード 1.
WRITE 出力レコード 1.
GO TO 出力開始.
出力終了.
CLOSE 出力ファイル 1.
出力開始終了.
EXIT.
END PROGRAM プログラム名.
```

### 環境部(ENVIRONMENT DIVISION)

以下のファイルを定義します。



## 整列併合用ファイル

マージ処理を行うための作業ファイルを定義します。英字で始まる8文字以内の英数字を指定する必要があります。なお、ASSIGN句は注釈とみなされます。



同じプログラムでソート処理を行う場合、整列併合用ファイルの定義は1つだけ行います。

## 入力ファイル

マージ処理の入力ファイルは、通常のファイル処理を行うときと同様にファイルを定義します。

## 出力ファイル

マージ処理で出力ファイルを使用するときには、通常のファイル処理を行うときと同様にファイルを定義します。

## データ部(DATA DIVISION)

環境部に定義したファイルのレコードを定義します。

## 手続き部(PROCEDURE DIVISION)

マージ処理には、MERGE文を使います。マージ処理の出力がファイルかレコードかによって、MERGE文の記述内容が異なります。

### 出力がファイルの場合

“GIVING 出力ファイル名”を記述します。

### 出力がレコードの場合

“OUTPUT PROCEDURE 出力手続き名”を記述します。

OUTPUT PROCEDUREで指定した出力手続きでは、RETURN文を使って、マージ済みのレコードを1件ずつ受け取ります。

マージキーは複数指定することができます。

マージ処理が終了すると、マージ処理の結果が特殊レジスタSORT-STATUSに設定されます。特殊レジスタSORT-STATUSは、COBOLコンパイラによって自動的に生成されるので、一般のデータとは異なり、COBOLプログラムの中で定義する必要はありません。MERGE文の実行後に特殊レジスタSORT-STATUSの値を検査することにより、マージ処理が異常終了しても、COBOLプログラムの実行を続行させることができます。また、MERGE文で指定した出力手続きの中で、特殊レジスタSORT-STATUSに16を設定することにより、マージ処理を終了させることができます。

下表に、特殊レジスタSORT-STATUSに設定される値と意味を示します。

表12.2 SORT-STATUSに設定される値と意味

| 値  | 意味   |
|----|------|
| 0  | 正常終了 |
| 16 | 異常終了 |



入力ファイルおよびGIVING指定の出力ファイルは、MERGE文実行時にそれらのファイルが開かれた状態であってはいけません。

特殊レジスタSORT-CORE-SIZEを用いると、PowerSORTが使用するメモリ空間の容量を限定することができます。この特殊レジスタは、暗にPIC S9(8) COMP-5で定義された数字項目です。設定する値は、バイト単位の数値です。

この特殊レジスタは、翻訳オプションSMSIZE()および実行時オプションsmsizeに指定する値の意味と等価ですが、同時に指定された場合の優先順位は、特殊レジスタSORT-CORE-SIZEが一番強く、以降、実行時オプションsmsize、翻訳オプションSMSIZE()の順で弱くなります。



## 例

|          |                               |                  |
|----------|-------------------------------|------------------|
| 特殊レジスタ   | MOVE 102400 TO SORT-CORE-SIZE | (102400=100キロです) |
| 翻訳オプション  | SMSIZE (500K)                 |                  |
| 実行時オプション | smsize300k                    |                  |

この場合、一番強い特殊レジスタSORT-CORE-SIZEの値 100キロバイトを優先します。



## 注意

この特殊レジスタは、別製品PowerSORTをインストールしている場合に有効であり、インストールしていない場合には無効です。

## 12.3.3 プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

## 12.3.4 プログラムの実行

マージを使ったプログラムは、以下の手順で実行します。

### 1. 環境変数BSORT\_TMPDIRの設定

マージ処理では、整列併合用ファイルという作業用ファイルが必要です。整列併合用ファイルは、環境変数BSORT\_TMPDIRに指定したフォルダに一時的に作成されます。環境変数の指定がない場合には、環境変数TEMPに指定したフォルダに一時的に作成されます。なお、これらの環境変数は、あらかじめ設定しておく必要があります。

### 2. 入力ファイルおよび出力ファイルの割当て

入力ファイルおよび出力ファイルの定義にファイル識別名を使用した場合、ファイル識別名を環境変数情報名として、入力ファイルおよび出力ファイルの名前を設定します。

### 3. プログラムの実行

プログラムを実行します。



## 参考

ソートマージ処理は、通常、COBOLランタイムシステムを使用して行いますが、PowerSORTをインストールした場合、PowerSORTを使用します。

PowerSORTを使用する場合の作業用ファイルは、以下の優先順位に従います。

### 1. 環境変数BSORT\_TMPDIRで指定されたフォルダ

### 2. 環境変数TEMPで指定されたフォルダ

### 3. 環境変数TMPで指定されたフォルダ

### 4. Windowsシステムのフォルダ

## 第13章 CSV形式データの操作

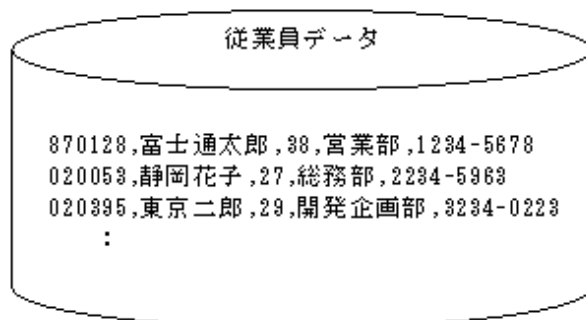
本章では、STRING文およびUNSTRING文を使用したCSV形式データの操作について説明します。

NetCOBOLでは、容易にCSV形式データを操作できるよう、STRING文およびUNSTRING文に拡張機能を用意しました。

### 13.1 CSV形式データとは

CSV(Comma Separated Values)形式データは、カンマで区切られた複数の文字列データの並びで、表計算ソフトやデータベースソフトで多く用いられてきました。最近では、これらソフトウェアに限らず、各種ツール類やミドルウェアとのデータ連携でも用いられるようになっていきます。

例えば、以下のようにテキストファイルで流通します。



これは、従業員番号、氏名、年齢、所属、内線をカンマで区切ったCSVデータです。

このデータをCOBOLプログラムで入力および編集する場合、行順ファイルとしてレコード単位に読み込み、カンマで区切られた文字列データを集団項目に従属する基本項目に分解して操作するのが一般的です。しかし、従来の言語仕様で実現するのは容易ではありませんでした。

上記の従業員データを以下の集団項目へ転記する場合を考えます。

```
01 従業員.
 02 従業員番号 PIC 9(6).
 02 氏名 PIC N(10).
 02 年齢 PIC 9(2).
 02 所属 PIC N(10).
 02 内線 PIC X(10).
```

うちPERFORM文を用いてCSVデータを先頭から1文字ずつ検査しながら転記することもできますが、ここでは、UNSTRING文(書き方1)を使用した場合を考えます。

```
FILE-SECTION.
FD CSV-FILE
01 CSV-REC PIC X(80).
:
WORKING-STORAGE SECTION.
01 従業員.
 02 従業員番号 PIC 9(6).
 02 氏名 PIC N(10).
 02 氏名ーR REDEFINES 氏名 PIC X(20). *> 字類を合わせるために追加
 02 年齢 PIC 9(2).
 02 所属 PIC N(10).
 02 所属ーR REDEFINES 所属 PIC X(20). *> 字類を合わせるために追加
 02 内線 PIC X(10).
:
READ CSV-FILE.
UNSTRING CSV-REC
 DELIMITED BY "," *> カンマで分解
 INTO 従業員番号 氏名ーR 年齢 所属ーR 内線
```

```
END UNSTRING.
:
```

簡単に記述できるように見えますが、上の例には以下の問題があります。

- UNSTRING文は、転記の規則に従って、受取り側の字類を合わせなければなりません。上の例では、REDEFINE句を使って解決していますが、実行時コード系がUnicodeの場合は、エンコードが異なるため、解決できません。
- CSV形式データでは、データ全体を引用符で囲めば、カンマをデータとして使用することができます。しかし、上の例では、そのようなデータを処理することはできません。
- 引用符をデータとして使用する場合、連続する2つの引用符で1つの引用符を表現します。しかし、上の例では、そのようなデータを処理することはできません。

上記の通り、UNSTRING文(書き方1)を使ってCSV形式データを分解することは、困難でした。

また、逆に、STRING文(書き方1)を使ってCSV形式データを生成する場合も、同様の問題(後置空白の処理などを考慮すると、更に大きな問題)を抱えていました。

NetCOBOLでは、STRING文およびUNSTRING文に、CSV形式のデータ操作に特化した新たな構文を追加して、容易に操作できるようにしました。

## 13.2 CSV形式データの作成 (STRING文)

ここでは、CSV形式データを作成する方法を説明します。

### 13.2.1 基本操作

集団項目に格納されている文字列データを、従属する項目単位でCSV形式へ編集する場合、STRING文(書き方2)を使用します。

```
WORKING-STORAGE SECTION.
77 従業員編集 PIC X(80).
01 従業員
 02 従業員番号 PIC 9(6). *> 870128 が格納されている状態
 02 氏名 PIC N(10). *> 富士通太郎 (以下同)
 02 年齢 PIC 9(2). *> 38
 02 所属 PIC N(10). *> 営業部
 02 内線 PIC X(10). *> 1234-5678
 :
MOVE SPACE TO 従業員編集.
STRING 従業員 INTO 従業員編集 BY CSV-FORMAT.
```

上の例のSTRING文を実行すると、データ項目“従業員編集”には、以下のCSV形式データが格納されます。

```
870128, 富士通太郎, 38, 営業部, 1234-5678
```

なお、CSV形式を生成する際、格納されているデータを以下のとおり編集します。

- 送出し側項目と受取り側項目の字類が異なる場合、受取り側項目の字類に合わせ、以下のように変換します。
  - 送出し側項目が日本語の場合、Unicode動作時にはエンコードの変換を行います。
  - 送出し側項目が符号つき数字項目の場合、SIGN句の指定に関わらず、符号を左端に付加します。また、小数部を含む場合、小数点文字を付加します。
- 送出し側データ中に区切り文字が含まれていた場合、データ全体を二重引用符で囲みます。
- 送出し側データ中に二重引用符が含まれていた場合、連続する2つの二重引用符に置き換え、データ全体を二重引用符で囲みます。
- 送出し側項目の字類が英数字または日本語の場合、後置空白は削除します。
- 送出し側項目が数字項目の場合、先行するゼロ列は削除します。ただし、値がゼロだった場合は、1けたのゼロを転記します。また、小数部を含む場合、後置ゼロは削除します。

- ・ TYPE指定に従い、データ全体を二重引用符で囲みます。  
詳細は、“[13.4 CSV形式のバリエーション](#)”を参照してください。

STRING文の文法や仕様の詳細は、“COBOL文法書”を参照してください。

## 注意

- ・ 受取り側への転記処理は、STRING文が作用した部分しか実行されません。したがって、文字転記のような後続領域への空白づめは期待できません。  
受取り側項目は、STRING文を実行する前に必ず初期化してください。
- ・ List Creatorでは、CSV形式データ中に2つの二重引用符が含まれていた場合でも、1つの二重引用符に置き換えません。

## 13.2.2 処理異常の検出

CSV形式データ作成時における異常とは、以下の状態を指します。

表13.1 CSV形式データ作成時における異常

|     |                          |
|-----|--------------------------|
| (1) | 送出し側項目に不正なデータが格納されている    |
| (2) | 受取り側項目が小さく、全てのデータが入り切らない |
| (3) | POINTER指定のデータ項目の値が1より小さい |

STRING文では、ON OVERFLOW指定を記述することによって異常発生時の動作を記述することができます。このとき、正常に処理できたところまで、受取り側に格納されます。

例えば、前述の例で受取り側領域の大きさが20けたしか用意されてなかった場合、以下のようにON OVERFLOW指定を記述すると、異常を検知することができます。

```

WORKING-STORAGE SECTION.
77 従業員編集 PIC X(20).
01 従業員.
 02 従業員番号 PIC 9(6). *> 870128 が格納されている状態
 02 氏名 PIC N(10). *> 富士通太郎 (以下同)
 02 年齢 PIC 9(2). *> 38
 02 所属 PIC N(10). *> 営業部
 02 内線 PIC X(10). *> 1234-5678
 :
MOVE SPACE TO 従業員編集.
STRING 従業員 INTO 従業員編集 BY CSV-FORMAT
 ON OVERFLOW DISPLAY "編集に失敗しました。データ=" 従業員編集
END-STRING.
```

なお、ON OVERFLOW指定が記述されてない場合は、“[表13.1 CSV形式データ作成時における異常](#)”が発生した時、以下のように動作します。

### 異常(1),(3)の場合

実行時エラーを出力後、異常終了します。

### 異常(2)の場合

実行時エラーを出力後、STRING文の次の文へ制御を移します。

## 13.3 CSV形式データの分解 (UNSTRING文)

ここでは、CSV形式データを分解する方法を説明します。

### 13.3.1 基本操作

CSV形式データを分解して、集団項目に従属する項目へ転記する場合、UNSTRING文(書き方2)を使用します。

```

WORKING-STORAGE SECTION.
77 従業員データ PIC X(80)
 VALUE "870128, 富士通太郎, 38, 営業部, 1234-5678".

01 従業員.
 02 従業員番号 PIC 9(6).
 02 氏名 PIC N(10).
 02 年齢 PIC 9(2).
 02 所属 PIC N(10).
 02 内線 PIC X(10).
 :
UNSTRING 従業員データ(1:FUNCTION STORED-CHAR-LENGTH(従業員データ))
 INTO 従業員 BY CSV-FORMAT.

```

上の例のUNSTRING文を実行すると、集団項目“従業員”に従属する各基本項目には、先頭から順番にカンマで分割されたデータが格納されます。

なお、CSV形式データの分解では、データを以下のとおり編集します。

- ・ 送出し側項目と受取り側項目の字類が異なる場合、受取り側項目の字類に合わせ、次のような変換を行います。
  - ー 受取り側項目が日本語の場合、Unicode動作時には文字コードの変換を行います。
  - ー 受取り側項目が数字の場合、受取り側のSIGN句の指定に応じ、符号処理を行います。また、小数部を含む数値の場合、桁合わせを行います。
- ・ 分割したデータが二重引用符で囲まれていた場合、二重引用符を除いてから転記します。
- ・ 二重引用符で囲まれた分割データ中に、連続する二重引用符が含まれていた場合、1つの二重引用符に置き換えます。
- ・ 分解したデータを受取り側項目へ転記する際は、転記の規則に従います。

UNSTRING文の文法や仕様の詳細は、“COBOL文法書”を参照してください。



#### 注意

TSV形式データが格納されているテキストファイルを、行順ファイルを用いて読み込み、UNSTRING文を使用して分割すると、意図したとおりに動作しません。これは、READ文が実行されるタイミングで、タブが空白に置き換えられるためです。行順ファイルに高速処理(“BSAM”)を指定すれば、タブが空白に置き換えられず、正しく処理することができます。

[参照]“7.3.3 行順ファイルの処理”、“7.7.4 ファイルの高速処理”

## 13.3.2 処理異常の検出

CSV形式のデータ分解時における異常とは、以下の状態を指します。

表13.2 CSV形式データ分解時における異常

|     |                                              |
|-----|----------------------------------------------|
| (1) | 送出し側項目に不正なデータが格納されている。                       |
| (2) | 受取り側項目への転記の際、けたあふれが発生する。                     |
| (3) | 分解した文字列データの数が、受取り側項目の数より多い。                  |
| (4) | POINTER指定のデータ項目の値が1より小さい、もしくは受取り側項目の桁数より大きい。 |

UNSTRING文では、ON OVERFLOW指定を記述することで、異常発生時の動作を記述することができます。このとき、正常に処理できたところまで、受取り側に格納されます。

例えば、前述の例で、受取り側項目に“内線”の定義を忘れていた場合、以下のようにON OVERFLOW指定を記述すると、異常を検知することができます。

```

WORKING-STORAGE SECTION.
77 従業員データ PIC X(80)
 VALUE "870128, 富士通太郎, 38, 営業部, 1234-5678".

01 従業員.
 02 従業員番号 PIC 9(6).

```

```

02 氏名 PIC N(10).
02 年齢 PIC 9(2).
02 所属 PIC N(10).
 :
UNSTRING 従業員データ (1:FUNCTION STORED-CHAR-LENGTH(従業員データ))
 INTO 従業員 BY CSV-FORMAT
 ON OVERFLOW DISPLAY "データの分解に失敗しました。"
END-UNSTRING.

```

このとき、POINTER指定を記述しておく、異常発生の原因となった送し側データの文字位置を取得できるので、より詳細な情報を得ることができます。

また、TALLYING指定を記述した場合には、転記に成功した項目数を得ることもできます。

```

WORKING-STORAGE SECTION.
77 CNT PIC 9(2).
 :
MOVE 1 TO CNT.
UNSTRING 従業員データ (1:FUNCTION STORED-CHAR-LENGTH(従業員データ))
 INTO 従業員 BY CSV-FORMAT POINTER CNT
 ON OVERFLOW DISPLAY "データの分解に失敗しました。"
 DISPLAY " 失敗データ= " 従業員データ (CNT:)
END-UNSTRING.

```

なお、ON OVERFLOW指定が記述されていない場合は、“表13.2 CSV形式データ分解時における異常”が発生した時、以下のように動作します。

#### 異常(1),(4)の場合

実行時エラーを出力後、異常終了します。

#### 異常(2)の場合

実行時エラーを出力後、UNSTRING文の処理を継続します。

#### 異常(3)の場合

実行時エラーを出力後、UNSTRING文の次の文へ制御を移します。

## 13.4 CSV形式のバリエーション

ここでは、CSV形式のバリエーションについて説明します。

CSV形式には、ISOが制定した国際規格のように正式に成立した仕様はありません。そのため、Microsoft社の表計算ソフトであるExcelの仕様がデファクトスタンダードとして、いくつかの派生形式が流通している状況にあります。

NetCOBOLでは、STRING文(書き方2)によって生成するCSV形式について、以下の4つのバリエーションを選択することができます。

データ連携する相手に合わせて指定してください。

| バリエーション | 内容                                                                                                                                                                |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MODE-1  | <ul style="list-style-type: none"> <li>送し側データ中に区切り文字または二重引用符が存在する場合、データ全体を二重引用符で囲みます。</li> <li>送し側データ項目の字類が英数字または日本語の場合で、全て空白が格納されていた場合、区切り文字のみを転記します。</li> </ul> |
| MODE-2  | <ul style="list-style-type: none"> <li>送し側データを二重引用符で囲みます。</li> <li>送し側データ項目の字類が英数字または日本語の場合で、全て空白が格納されていた場合、区切り文字のみを転記します。</li> </ul>                             |
| MODE-3  | <ul style="list-style-type: none"> <li>送し側データ項目の字類が数字以外の場合、データ全体を二重引用符で囲みます。</li> <li>送し側データの字類が英数字または日本語の場合で、全て空白が格納されていた場合、区切り文字のみを転記します。</li> </ul>            |

| バリエーション | 内容                                                                                                                                                                |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MODE-4  | <ul style="list-style-type: none"> <li>・ 送出し側データ項目の字類が数字以外の場合、データ全体を二重引用符で囲みます。</li> <li>・ 送出し側データの字類が英数字または日本語の場合で、全て空白が格納されていた場合、連続する2つの二重引用符を転記します。</li> </ul> |

これらバリエーションは、STRING文(書き方2)のTYPE指定、または実行環境変数“5.4.1.12 @CBR\_CSV\_TYPE (生成するCSV形式のバリエーション)”で指定します。なお、省略時は、MODE-1が選択されたものとみなします。

以下に例を示します。

|          |                                    |
|----------|------------------------------------|
| 01 従業員.  |                                    |
| 02 従業員番号 | PIC 9(6) VALUE 870128.             |
| 02 氏名    | PIC N(10) VALUE NG"富士通太郎".         |
| 02 所属    | PIC N(10) VALUE NG"営業部".           |
| 02 役職    | PIC X(10) VALUE SPACE.             |
| 02 内線    | PIC X(20) VALUE "1234-5678, 4536". |

上の例のデータ項目“従業員”を送出し側項目に指定した場合、バリエーションの指定によって、それぞれ以下の結果となります。

| バリエーション | 結果                                       |
|---------|------------------------------------------|
| MODE-1  | 870128,富士通太郎,営業部,, "1234-5678,4536"      |
| MODE-2  | "870128","富士通太郎","営業部",,"1234-5678,4536" |
| MODE-3  | 870128,"富士通太郎","営業部",,"1234-5678,4536"   |
| MODE-4  | 870128,"富士通太郎","営業部","","1234-5678,4536" |

なお、いずれのCSV形式データも、UNSTRING文(書き方2)を用いて集団項目に従属する項目へ分解および転記することができます。



## 第14章 システムプログラムを記述するための機能～SD機能～

本章では、システムプログラムを記述する場合に有効となる機能(SD機能)について説明します。

### 14.1 SD機能の種類

NetCOBOLには、システムプログラムを記述する場合に有効な以下の機能があります。これらの機能をNetCOBOLでは、システムプログラム記述向け(SD)機能といいます。

- ・ ポインタ付け
- ・ ADDR関数およびLENG関数
- ・ 終了条件なしのPERFORM文

以下に、各機能の概要および特徴について説明します。

#### ポインタ付け

ポインタ付けでは、ある特定のアドレスを持つ領域を参照または更新することができます。たとえば、COBOLプログラムが他言語で記述されたプログラムから呼び出されるときのパラメタが領域アドレスの場合、COBOLプログラム中では、ポインタ付けを使って、そのアドレスを持つ領域の内容を参照または更新することができます。

#### ADDR関数およびLENG関数

ADDR関数では、COBOLで定義したデータ項目のアドレスを得ることができます。また、LENG関数では、COBOLで定義したデータ項目および定数の長さをバイト数として得ることができます。たとえば、COBOLプログラムから他言語で記述されたプログラムを呼び出すときのパラメタとして、領域のアドレスや領域の長さを渡すことができます。

#### 終了条件なしのPERFORM文

NetCOBOLでは、PERFORM文に終了条件を設定しない書き方ができます。たとえば、繰返し処理の中で行われる処理の結果を判定し、繰返し処理を終了することができます。

### 14.2 ポインタ付けの使い方

ここでは、ポインタ付けの使い方について説明します。

#### 概要

ポインタ付けは、ある特定のアドレスを持つ領域を参照する場合に使います。ポインタ付けを行うためには、次のデータ項目が必要です。

- ・ 基底場所節で定義したデータ項目(a)
- ・ 属性がポインタデータ項目のデータ項目(b)

ポインタ付けは、通常、ポインタ修飾子(->)によって行われます。(a)は、(b)によって次のようにポインタ付けされます。これをポインタ修飾といいます。

(b)->(a)

この場合、(a)の内容は、(b)に設定されたアドレスを持つ領域の内容となります。

#### プログラムの記述

ここでは、ポインタ付けを使うプログラムの記述について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
DATA DIVISION.
BASED-STORAGE SECTION.
01 データ名1 [BASED ON ポインタ1].
77 データ名2 ~ [BASED ON ポインタ2].
WORKING-STORAGE SECTION.
```

```
01 ポインタ 1 POINTER.
LINKAGE SECTION.
01 ポインタ 2 POINTER.
PROCEDURE DIVISION USING ポインタ 2.
 MOVE [ポインタ 1->] データ名 1 ~.
 IF [ポインタ 2->] データ名 2 ~.
END PROGRAM プログラム名.
```

#### 環境部(ENVIRONMENT DIVISION)

とくに必要な記述はありません。

#### データ部(DATA DIVISION)

基底場所節で、アドレスを指定して参照または更新を行うためのデータ名を定義します。また、ファイル節、作業場所節、基底場所節および連絡節でアドレスを格納するためのデータ名(属性をポインタデータ項目(PONTER)とする)を定義します。

#### 基底場所節でのデータ名の定義

基底場所節でのデータ名の定義は、通常のデータを定義するときと同様にデータ記述項を使います。基底場所節に定義したデータ名に対しては、プログラム実行時に実際の領域は確保されません。したがって、基底場所節に定義したデータ名を参照するときには、参照する領域のアドレスを指定する必要があります。データ記述項にBASED ON句を記述すると、そのデータ名は、BASED ON句に指定したデータ名により暗にポインタ付けされ、ポインタ修飾を行わないで使用することができます。BASED ON句を記述していないデータ名を使用するときには、ポインタ修飾を行う必要があります。

#### 手続き部(PROCEDURE DIVISION)

ポインタ付けされたデータ名は、MOVE文やIF文などに、通常のデータ名と同様に記述することができます。

### プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

### プログラムの実行

とくに必要な環境設定はありません。

## 14.3 ADDR関数およびLENG関数の使い方

ここでは、ADDR関数およびLENG関数の使い方について説明します。

#### 概要

ADDR関数は、関数値としてデータ項目のアドレスを返却します。また、LENG関数は、データ項目または定数の大きさをバイト数で返却します。

#### プログラムの記述

ここでは、ADDR関数およびLENG関数を使うプログラムの記述について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 データ名 1 ~.
01 ポインタ 1 POINTER.
01 データ名 2.
 02 ~ [OCCURS ~ DEPENDING ON ~].
01 データ名 3 PIC 9(4) BINARY.
PROCEDURE DIVISION.
 MOVE FUNCTION ADDR(データ名 1) TO ポインタ 1.
 MOVE FUNCTION LENG(データ名 2) TO データ名 3.
END PROGRAM プログラム名.
```

#### 環境部(ENVIRONMENT DIVISION)

とくに必要な記述はありません。

#### データ部(DATA DIVISION)

ADDR関数およびLENG関数の関数値を格納するデータ名を定義します。ADDR関数の関数値の属性はポインタデータ項目、LENG関数の関数値の属性は数字項目です。

#### 手続き部(PROCEDURE DIVISION)

ADDR関数およびLENG関数は、MOVE文やIF文などに、通常のデータ名と同様に記述することができます。

### プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

### プログラムの実行

とくに必要な環境設定はありません。

## 14.4 終了条件なしのPERFORM文の使い方

ここでは、終了条件なしのPERFORM文の使い方について説明します。

### 概要

繰返し処理の中で終了条件を判定したい場合、通常のPERFORM文ではプログラムの記述が複雑になります。このような場合、終了条件を設定しないPERFORM文を使用すると、プログラムの記述が簡単になります。

### プログラムの記述

ここでは、終了条件なしのPERFORM文を使うプログラムの記述について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. プログラム名.
PROCEDURE DIVISION.
PERFORM WITH NO LIMIT
 ~
 IF ~
 EXIT PERFORM
 END-IF
 ~
END-PERFORM.
END PROGRAM プログラム名.
```

#### 環境部(ENVIRONMENT DIVISION)

とくに必要な記述はありません。

#### データ部(DATA DIVISION)

とくに必要な記述はありません。

#### 手続き部(PROCEDURE DIVISION)

終了条件なしのPERFORM文には、WITH NO LIMITを指定します。PERFORM文による繰返し処理の中では、終了条件を判定し、繰返し処理を脱出する文を記述します。繰返し処理の中に繰返し処理を脱出する文がない場合、この繰返し処理は無限に繰り返されます(無限ループ)。

### プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

### プログラムの実行

とくに必要な環境設定はありません。

## 第15章 オブジェクト指向プログラミングとは

本章では、オブジェクト指向プログラミングについて、簡単に紹介します。

はじめに、オブジェクト指向プログラミングが誕生した背景について、従来のプログラミング技法の問題点を挙げながら説明します。次に、オブジェクト指向の基本的な概念について、従来のプログラミング技法と比較しながら説明します。

### 15.1 オブジェクト指向の歴史的背景

1950年代の終わりに最初のプログラミング言語FORTRANが誕生してから、すでに30年以上経っています。その間、多くのプログラミング言語が誕生し、消えていきました。そして、よりよいプログラムを書くために、さまざまなプログラミングスタイルが考案されてきました。

“プログラミングスタイル”とは、よりよいプログラムを書くための約束ごとです。構造化プログラミングをはじめ、これまで多くのプログラミングスタイルが提唱されてきました。オブジェクト指向も、こうしたプログラミングスタイルの1つです。

ここでは、オブジェクト指向が生まれた背景として、プログラミングスタイルの変遷について説明します。

#### 15.1.1 ソフトウェア工学以前

1950年ごろ作られた、世界最初のプログラム内蔵式コンピュータEDSACでは、機械語でプログラムが書かれていました。初期のプログラマたちは、限られたメモリと少ない命令で、芸術的ともいえるコードを書いていた。

しかし、技術の発展とコンピュータの普及にともない、より多くの、より大規模のソフトウェアが要求されるようになりました。1950年代の終わりに、最初の高級言語であるFORTRANが登場しました。これにより、プログラムの生産性は飛躍的に向上しました。しかし、それ以上に、ソフトウェアに対する要求は複雑になり、また、コンピュータによるシステム化の要求も増大しました。これらの要求に、高級言語の出現だけでは対応しきれませんでした。そして、その対策として“ソフトウェア工学”が誕生しました。

#### 15.1.2 構造化プログラミングの出現

ソフトウェア工学の最初の成果が、1970年代始めに現れた構造化プログラミングです。これは、プログラムの制御の流れとデータ構造を、わかりやすく記述するための手法です。

プログラム制御の構造化は、プログラムは以下の3つの構造を使って表現できるという“構造化定理”に基づいています。

- ・ 接続処理(文の並び)
- ・ 選択処理(IF文、EVALUATE文に相当)
- ・ 繰り返し処理(うちPERFORM文に相当)

GO TO文を多用すると、全体の流れがわかりにくいプログラムになってしまいます。しかし、これら3つの構造の組合せで記述すれば、大変わかりやすいプログラムを書くことができます。

また、データ構造についても同じです。従来は、個々のデータを別々に扱うしかありませんでした。しかし、データに構造を持たせることにより、複数のデータをまとめて1つのデータとして扱えるようになりました。データ構造も、基本的には以下の3つの構造の組合せです。

- ・ 接続(集団項目)
- ・ 選択(REDEFINES句)
- ・ 繰り返し(OCCURS句)

これらの考えは、COBOLにも取り入れられています。

#### 15.1.3 モジュール化

1970年代半ばには、プログラム開発の規模はますます大きくなりました。そのため、モジュール化という方法が考え出されました。

モジュール化とは、プログラムを独立性の高いモジュールに分割し、さらにそのモジュールを、より小さなモジュールに分割する手法です。この手法を使うと、プログラムの規模が大きくても、全体の見通しは非常によくなります。これにより、大規模プログラムの共同開発が可能になりました。

モジュール化は、見方を変えれば、プログラムの持つ機能を抽象化することです。プログラミングする対象を分析し、それを抽象的な機能単位に分けます。もちろん、内部のコードがどうなっているのか考える必要はありません。これを段階的に繰り返すと、全体的に見通しのよいプログラムを作ることができます。

### 15.1.4 抽象データ型

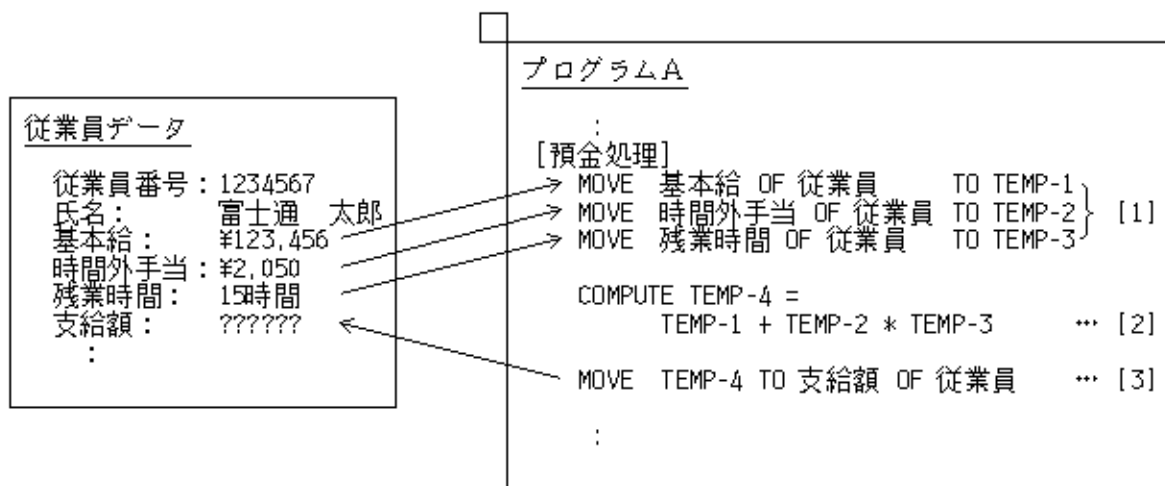
1970年代後半になると、開発済みのプログラムの保守作業の負荷が増大しました。それを軽減するために、抽象データ型という考え方が現れました。

モジュール化は手続きの抽象化でしたが、抽象データ型はデータの抽象化にあたります。

従来、プログラムで扱うデータは、数値、文字列などのコンピュータで表現しやすいものを中心でした。しかし、プログラムの対象となる現実の世界は、そんなに単純ではありません。たとえば、従業員管理システムで従業員データをモデル化しようすると、従業員番号、氏名、住所、基本給など、さまざまな情報が集まった複合的なデータになります。データの構造化により、そのようなデータを定義することができるようになりましたが、データ操作は、あいかわらず個々のデータごとに行っていました。たとえば、ある従業員の給与を計算するコードを書く場合、以下ようになります。

1. 従業員データから基本給、時間外手当および残業時間を取り出します。
2. 取り出したデータを基に、総支給額を計算します。
3. 計算した金額を、従業員データの支給額フィールドに書き込みます。

以下の図に示すように、プログラムAから従業員データの中の“基本給”、“時間外手当”、“残業時間”および“支給額”に直接アクセスします。



しかし、この方法には以下の問題があります。

- そのデータを扱うすべてのプログラムは、データの構造を知っていなければならない。
- データ内の個々の情報へのアクセスは制限されていないので、誤った操作により破壊される危険がある。
- データの構造を変更した場合、そのデータを利用していたモジュールをすべて変更しなければならない。

そこで登場したのが、抽象データ型という考えです。

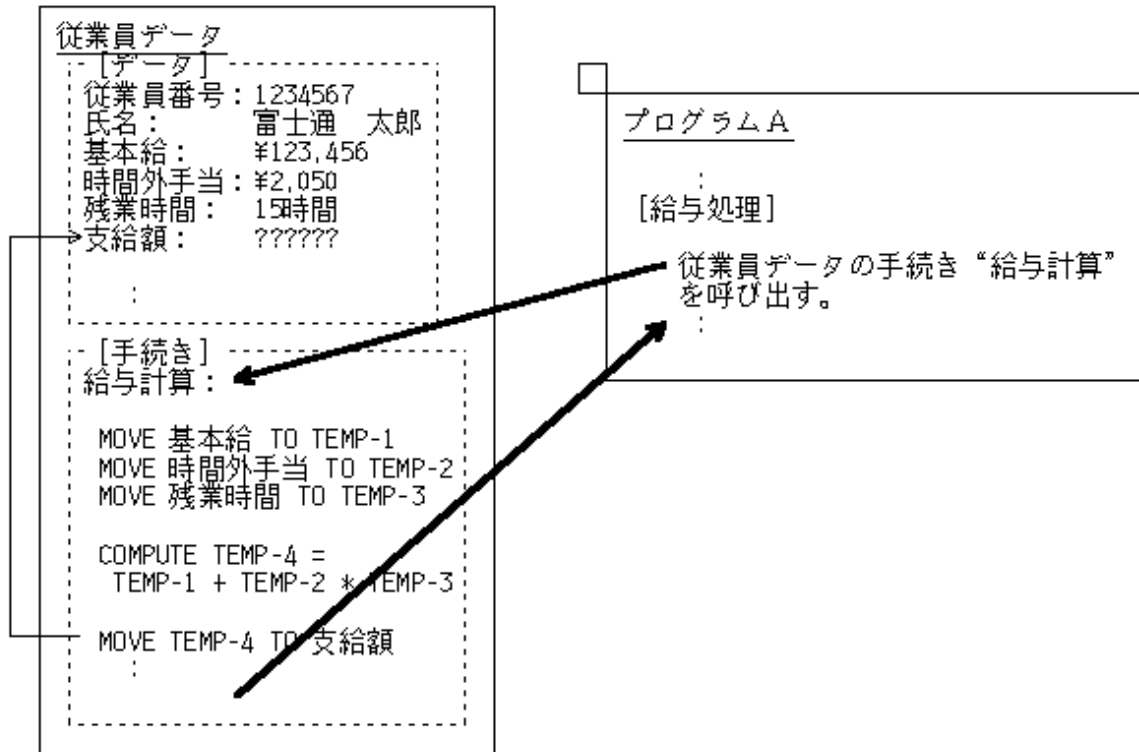
抽象データ型では、現実世界の“物”の持つ性質を、抽象的なデータ型としてモデル化します。抽象データ型には、以下の特徴があります。

- 内部にデータと、それにアクセスする手続きのコードを持ちます。
- 内部のデータの手続きコードは、外部からはまったく見えません。
- データをアクセスするためには、あらかじめ用意された手続きを呼び出さなければなりません。
- 外部(そのデータを使用するプログラム)に公開されているのは、手続きのインタフェースだけです。

つまり、内部のデータおよびプログラム構造は外部から完全に隠されており、外部仕様である手続きのインタフェースを通じてだけアクセスできます。このような考え方を「情報隠蔽」といいます。また、データと手続きを一体化することを「カプセル化」といいます。

以下の図は、従業員データの中に“給与計算”という手続きを持たせています。プログラムから見えるのは“給与計算”手続きだけで、これと呼ばないかぎり、給与関連の情報にはアクセスできません。また、“給与計算”という手続きは従業員データの一部として管理されます。

カプセル化されたデータ



このような構成にしておくと、それぞれのデータ型との接点は公開されたインタフェースだけになります。そのため、内部の実現方法(たとえば、内部のデータ構造や手続きのコード)を変更しても、インタフェースさえ変わらなければ、それを利用するプログラムには影響ありません。つまり、変更に対する影響を局所化でき、保守作業の負荷を大幅に減らせます。また、公開されているのはインタフェースだけなので、あらかじめ決められた正しい方法以外ではアクセスできません。そのため、自然にプログラムの信頼性が向上します。

### 15.1.5 オブジェクト指向の誕生

1980年ごろに、抽象データ型の延長として現れたのがオブジェクト指向です。

このころになると、ソフトウェア危機がより深刻になり、プログラムの飛躍的生産性向上が重要な課題になっていました。それに対する最も効果的な方法は、“プログラムを作らない”ことです。つまり、あらかじめプログラムを部品として用意しておき、それらを組み合わせるだけで新しいプログラムを作るということです。ICやLSIの出現で、家電製品の製造コストが大幅に下がったのと同じ論理です。

これまで、モジュール化、抽象データ型により部品化が試みられてきました。しかし、これらの手法による部品化には、以下の欠点がありました。

- ・ 再利用しにくい
- ・ 柔軟性に欠ける

たとえば、既存の従業員管理システムに新しい勤務形態(たとえば年俸制)を加えるとします。しかし、年俸制の従業員であっても、ほとんどの部分は既存の一般従業員と同じ処理が使えます。それにもかかわらず、年俸制従業員に対しても、一般従業員と同じ処理を定義する必要があります。たとえば、両者の住所変更処理がまったく同じであっても、それぞれの手続きを書く必要がありました。

また、一般従業員と管理職という2つの従業員データがあったとします。従来の方法では、これらを同時に扱う処理を作るのは困難でした。これらのデータは似ていますが、一般従業員用の処理と管理職用の処理は別々に作らなければなりません。さらに、これに年俸制従業員が加わると、もう1つ処理を作る必要があります。

オブジェクト指向では、前者は継承により、後者は多態と動的束縛により解決しました。これらの概念については、次節以降で詳しく説明します。

代表的なオブジェクト指向言語には、Smalltalk、C++などがあります。そして、最近では、COBOLでも取り入れられています。

## 15.2 オブジェクト指向の基本的な考え方

ここでは、オブジェクト指向の基本的な概念である、オブジェクト、クラス、メソッド、継承、多態および動的束縛について説明します。プログラムの具体的な書き方については、“[第16章 オブジェクト指向プログラミング機能～基本的な使い方～](#)”を参照してください。

### 15.2.1 オブジェクトとクラス

以下に、オブジェクトおよびクラスについて説明します。

#### オブジェクト

オブジェクト指向では、プログラムで扱う“もの”はすべてオブジェクトと呼びます。それは、実在する“もの”であったり、概念上の“もの”であったりします。オブジェクト指向プログラミングとは、すべてオブジェクトを中心にしてプログラムを作ろうという試みです。

プログラム上では、オブジェクトは“データ”と“手続き”をカプセル化したものとして表現されます。データは、オブジェクトの中に隠蔽されており、公開されたインタフェースを通してしかアクセスできません。これは、抽象データ型の考え方そのものであり、その長所(保守の容易さ、高い信頼性)をすべて引き継いでいます。

また、手続きは“メソッド”と呼ばれます。本章でも、これ以降の説明では“メソッド”という用語を使います。

#### クラス

一般に、同じ形をしたオブジェクトは複数存在します。たとえば、従業員データの場合、富士通太郎さんの従業員データもあれば、他の人の従業員データもあります。これらはそれぞれ別々の値を持っていますが、すべて同じ形であり、かつ、同じメソッドを持っています。つまり、従業員データの定義情報があって、そこから作られた従業員オブジェクトが複数存在するわけです。オブジェクト指向では、この定義情報のことをクラスと呼びます。

クラスは、共通の属性を持ったオブジェクトのグループであると考えられることもできます。あるクラスに属するオブジェクトは、すべて同じ構造のデータを持ち、同じメソッドを持っています。つまり、クラスは、そのクラスに属するオブジェクトの振舞いを定義しているわけです。また、すべてのオブジェクトは必ずいずれかのクラスに属しています。



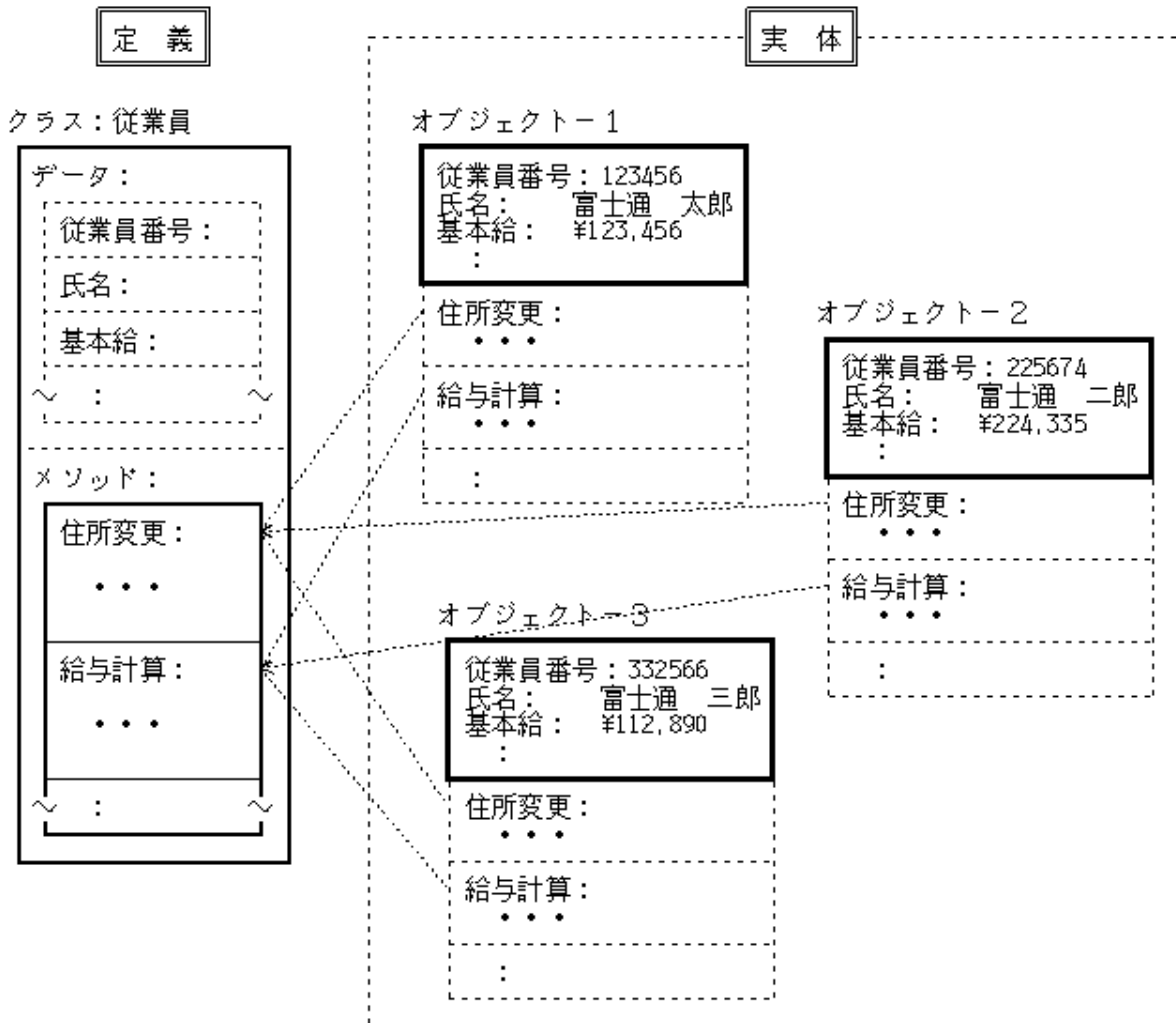
#### 例

##### 従業員クラスとオブジェクト

従業員管理プログラムでは、たくさんの従業員データを扱います(ここでは、単純化するために、従業員の種類は1種類だけと仮定します)。それぞれの従業員データは、いろいろな情報(たとえば、従業員番号、氏名や住所)を持っています。そして、それぞれの従業員データは、外部からアクセスするために、いくつかの手続き(たとえば、住所変更や給与計算)を持っています。そして、すべての従業員データは、同じデータ構造と同じメソッドを持っています。

これら個々の従業員データがオブジェクトで、それらの性質(データ構造と手続き)を定義しているのが、従業員クラスです。

すべての従業員オブジェクトは同じように振る舞います。従業員データのクラスとオブジェクトの関係は、以下の図のようになります。



この例では、従業員クラスのオブジェクトが3つあります。データ構造はクラスで定義されていますが、実際に値を持っているのは、3つのオブジェクトです。これらは、同じ構造ですが、持っている値は異なります。

また、メソッドもクラスで定義されています。メソッドは、一見それぞれのオブジェクトに含まれているように振る舞いますが、実は1つしか存在しません。メソッドの手続きは、データと違って、異なる値を持つ必要がないからです。それぞれのオブジェクト上でメソッドを実行すると、実際には、クラスで定義されたメソッドが実行されます。

## 15.2.2 オブジェクトの作成・参照

個々のオブジェクトは、実行時に動的に作成されます。作成方法はプログラミング言語により異なりますが、COBOLの場合はファクトリオブジェクトという特殊なオブジェクトで作成します。詳細については、“[第16章 オブジェクト指向プログラミング機能～基本的な使い方～](#)”を参照してください。

オブジェクトは、それぞれオブジェクト参照という値を持っています。これは、オブジェクトを一意に指すためのポインタです。クラスが同じであっても、異なるオブジェクトであれば、オブジェクト参照の値は異なります。

オブジェクトを生成すると、必ずオブジェクト参照が割り当てられます。この値は、オブジェクト参照データ項目に格納します。そして、それ以降は、そのデータ項目を使うことにより、オブジェクトを一意に識別できます。

## 15.2.3 メソッドの呼出し

オブジェクト指向のプログラムは、オブジェクトにメッセージを送ることから始まります。オブジェクトは、メッセージを受け取ると、そのメッセージに対応したメソッドを実行します。メッセージの送信は、プログラム上ではオブジェクトに対するメソッド呼出しとして表します。



オブジェクトに対してメソッドを呼び出すためには、以下の情報が必要です。

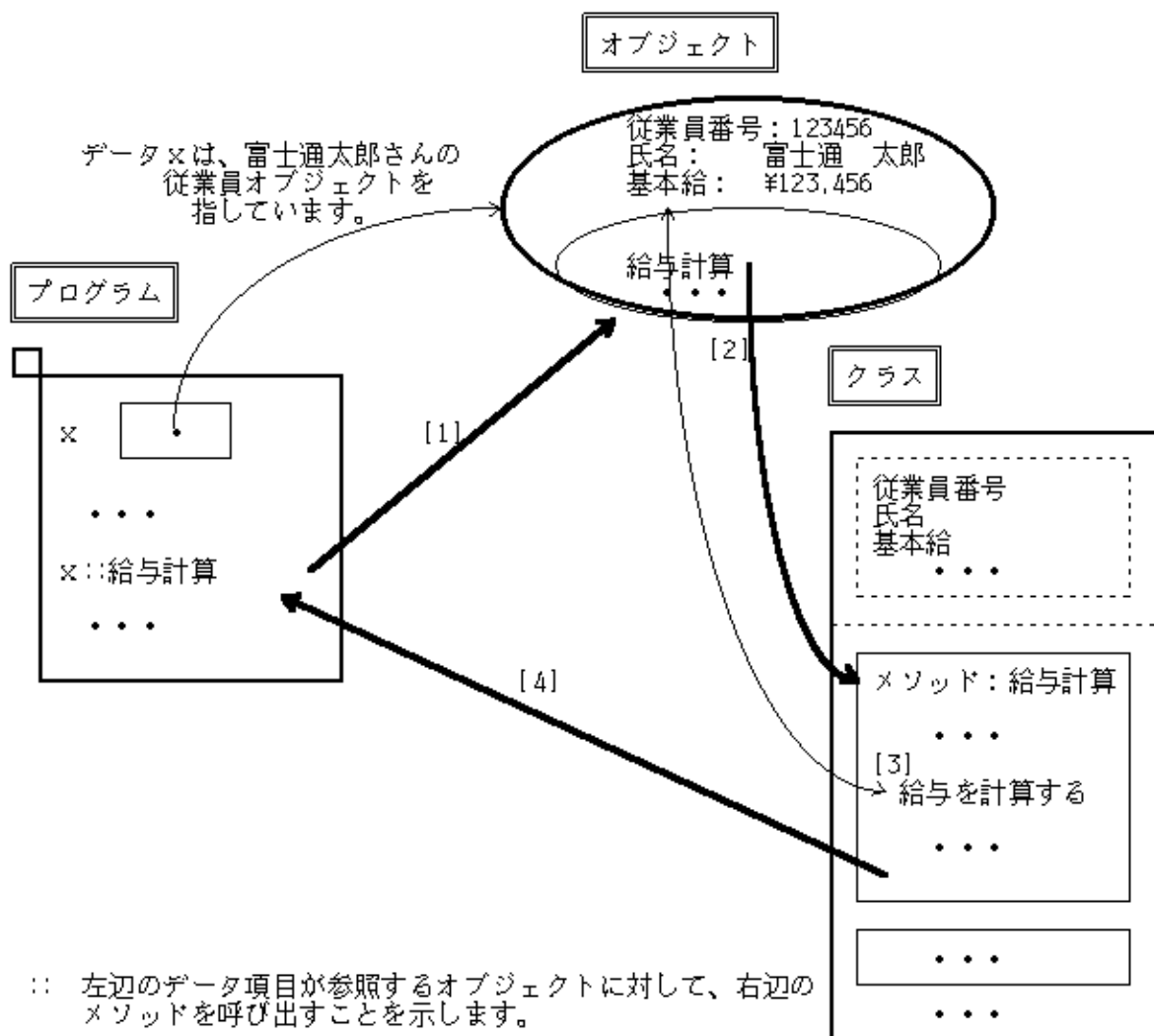
- ・ メソッドを実行するオブジェクトはどれか?
- ・ メソッド名
- ・ パラメタ(必要な場合だけ)

メソッドを実行するオブジェクトは、オブジェクト参照データ項目で指定します。



## 例

### 従業員クラスの給与計算処理



[1] 富士通太郎さんの従業員オブジェクトに対して、“給与計算”メソッドを呼び出します。

[2] 富士通太郎さんの従業員オブジェクト上で、“給与計算”メソッドを実行します。

[3] “給与計算”メソッドでは、富士通太郎さんの従業員オブジェクト内のデータを参照・設定しながら、給与計算処理を行います。

[4] 給与計算処理が終わったら、呼出し元に戻ります。

## 15.2.4 継承

オブジェクト指向の長所の1つは、既存のプログラムを簡単に再利用できることです。これは、継承と呼ばれる仕組みで実現されています。

継承とは、文字どおり、他のクラスの性質をそのまま受け継ぐことです。新しいクラスを作る場合に継承を使用すると、既存のクラスの性質をそのまま受け継ぐことができます。さらに、新しいデータを追加したり、新しいメソッドを追加したり、メソッドを置き換えたり、さまざまな改造ができます。つまり、既存のクラスを継承すれば、そこからの差分をコーディングするだけで、新しいクラスを作ることができます。もちろん、既存のクラスで定義されていたデータやメソッドは、新しいクラスでもそのまま使えます。

継承される(基となる)クラスを親クラス、継承する(性質を受け継ぐ)クラスを子クラスと呼びます。親クラスで定義されたデータは、子クラスでも暗黙に定義されます。また、親クラスで定義されたメソッドは、子クラスでも定義されているかのように使用できます。



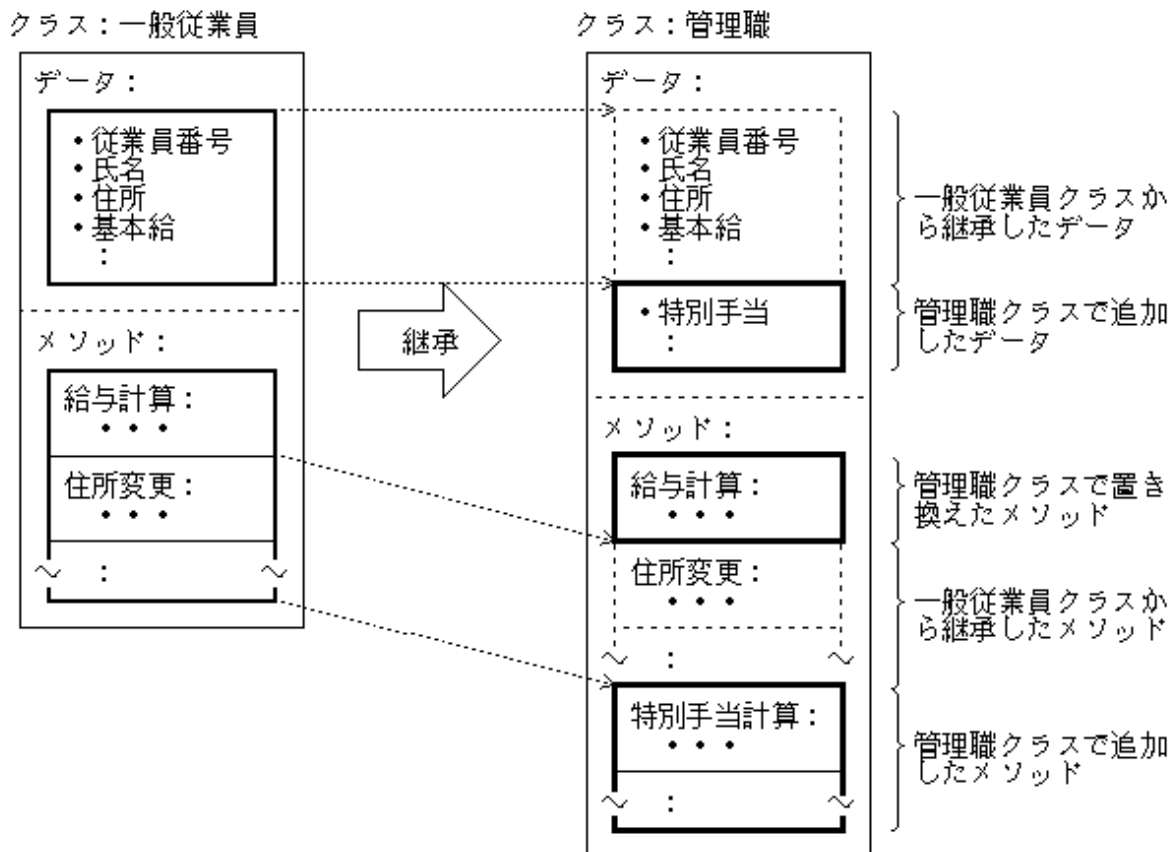
### 例

#### 新しい従業員クラス作成

継承による再利用の機構は、既存のシステムに新しい機能を追加するときに有効です。

たとえば、一般従業員クラスに対して、特別手当の支給だけが異なる管理職クラスを作る場合、一般従業員クラスを継承して、新たに“管理職”クラスを作ります。管理職クラスでは、一般従業員クラスに対して、特別手当の処理を追加します。そのためには、それに関するデータ(特別手当)とメソッド(特別手当の計算)の追加が必要です。また、特別手当を加算するために給与計算処理も変更する必要があります。

一般従業員クラスと管理職クラスの関係は、以下の図のようになります。





## 例

### クラスの汎化

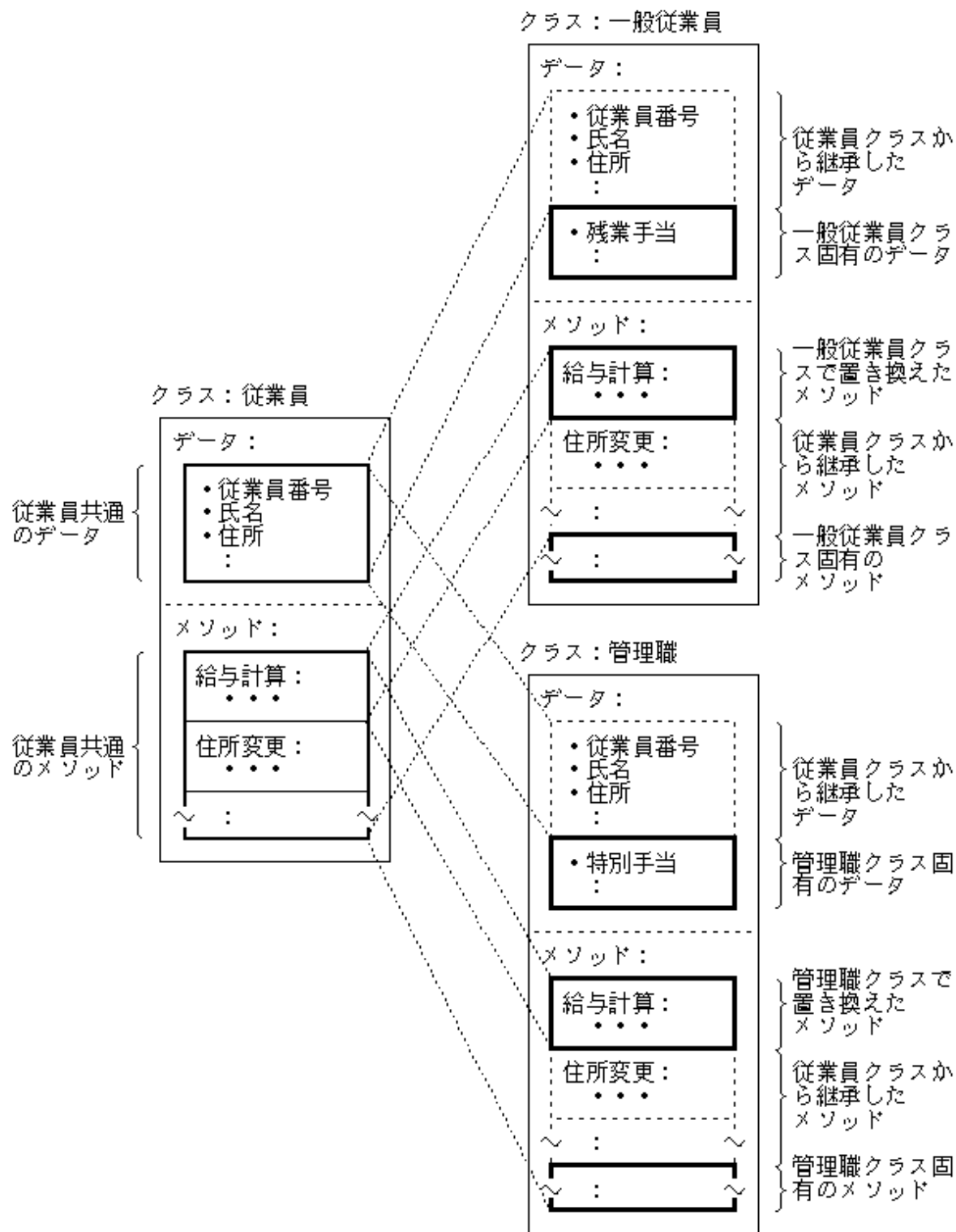
継承には、もう1つの使い方があります。

前の例では、一般従業員クラスに特別な処理を追加し、管理職クラスを作成しました。これは、一般従業員の特殊化されたものが管理職だと仮定したからです。しかし、実際には、一般従業員と管理職は、多くの共通点もあれば、多くの相違点もあります。見方を変えれば、一般従業員と管理職は、従業員という共通の性質の上に、一般従業員・管理職それぞれ固有の性質を追加したものです。

たとえば、一般従業員と管理職は、給与の計算方法だけが異なっていると仮定します。一般従業員は残業時間に比例する残業手当が、管理職には一定額の特別手当が支給されると仮定します。それ以外の性質については、まったく同じとします。このような場合、まず、従業員共通の性質を持った従業員クラスを作ります。そして、そのクラスを継承して一般従業員クラスと管理職クラスを作ります。つまり、従業員クラスには、すべての従業員に共通なデータとメソッドを定義しておき、一般従業員クラスには一般従業員固有の性質を、管理職クラスには管理職固有の性質を追加します。

このような構成にしておくと、新たに勤務形態を追加しても、既存のコードの修正はほとんど不要です。たとえば、パートタイム制を追加する場合でも、従業員クラスを継承したパートタイム従業員クラスを追加するだけです。

従業員クラス、一般従業員クラスおよび管理職クラスの関係は、以下の図のようになります。



## 参考

- 一般従業員クラス、管理職クラスのオブジェクトは存在しますが、従業員クラスのオブジェクトは存在しません。従業員クラスは、一般従業員クラス、管理職クラスに共通の性質を定義するために作成されたクラスです。このようなクラスを抽象クラスと呼びます。

- ・ 従業員クラスの給与計算メソッドは、インタフェースを定義しているだけです。実際の処理は、子クラスである一般従業員クラスおよび管理職クラスで定義されています。このように、抽象クラスは、子クラスで定義しなければならないメソッドの宣言のためによく使われます。

## 15.2.5 多態と動的束縛

オブジェクト指向のもう1つの特徴は、柔軟性です。そのための仕組みが、多態および動的束縛です。

### 多態

多態とは、“1つのものが複数の形態をとることができる”ことです。

あるクラスAと、それを継承するクラスB、Cがある場合、BもCもAの性質を受け継いでいます。見方を変えれば、B、CはAの一種です(B is a A, C is a A)。そのため、継承関係はis\_a関係とも呼びます。

前節で説明した一般従業員クラスと管理職クラスは、従業員クラスの子クラスです。つまり、一般従業員オブジェクトも管理職オブジェクトも、従業員オブジェクトの一種です。そして、従業員オブジェクトの性質をすべて引き継いでおり、従業員オブジェクトとして振る舞うことができます。たとえば、プログラム上で“従業員クラスのオブジェクトに対し、給与計算を行う”と書かれていても、一般従業員オブジェクトや管理職オブジェクトが従業員オブジェクトの代わりになることができます。それは、どちらのオブジェクトも“給与計算”メソッドを持つという性質を従業員クラスから引き継いでいるからです。

1つのもの(従業員オブジェクト)が、複数の形態(一般従業員オブジェクトと管理職オブジェクト)をとれたわけです。

### 動的束縛

一般従業員クラスと管理職クラスは、同じインタフェースの給与計算処理を持つと仮定しました。しかし、その中の処理はまったく異なります。一般従業員の場合は一般従業員用の、管理職の場合は管理職用の給与計算を実行しなければなりません。従業員オブジェクト上で給与計算メソッドを呼び出しますが、オブジェクトが一般従業員クラスのものであれば一般従業員用の給与計算メソッドが、管理職クラスのオブジェクトであれば管理職用の給与計算メソッドが呼び出されなければなりません。

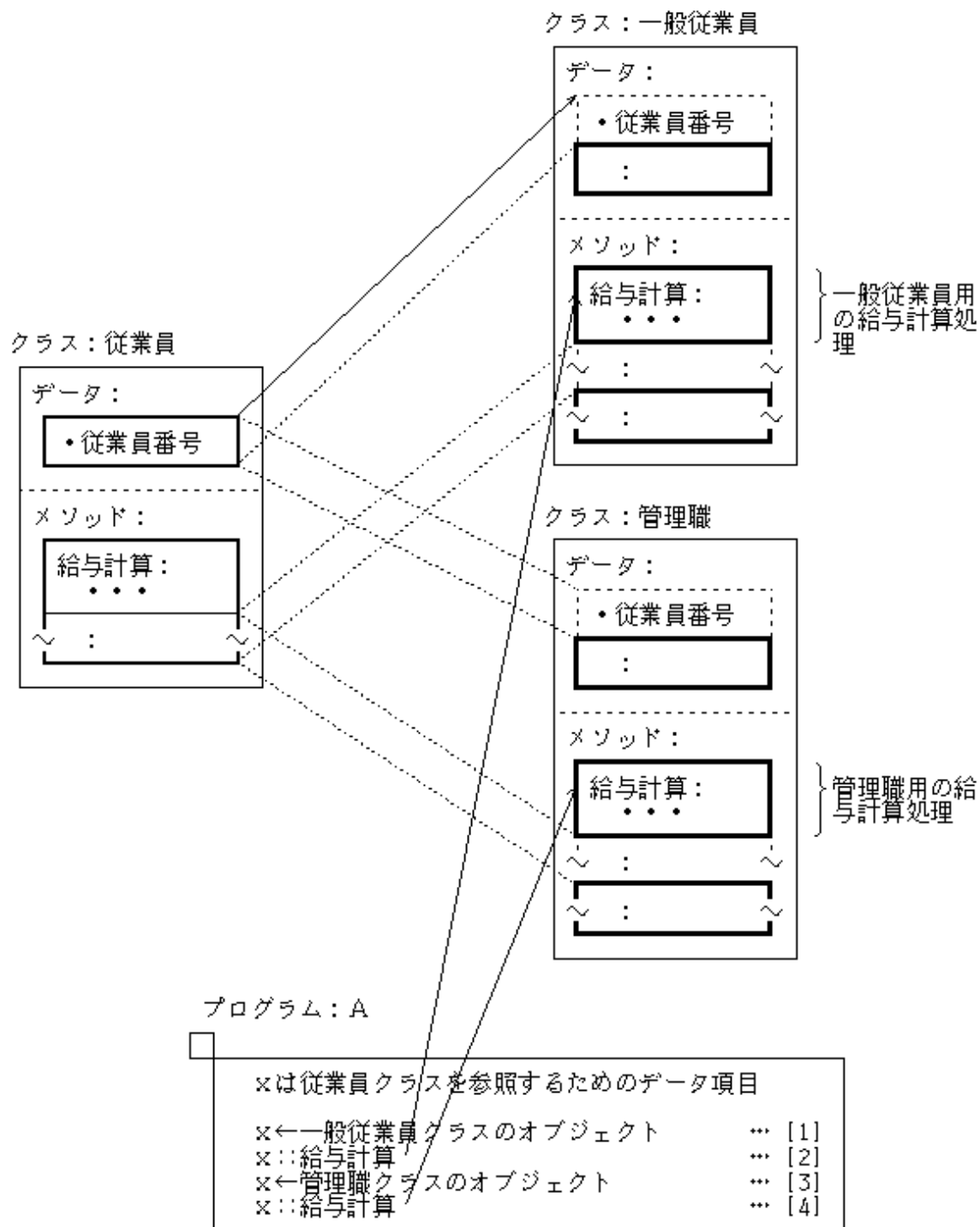
このように、オブジェクト指向では、同じメソッド呼出しであっても、そのメソッドを実行するオブジェクトによって呼び出されるメソッドが変わります。つまり、呼び出されるメソッドは実行時に動的に決まります。このような機構を、呼び出すメソッドが実行時に動的に決まるということで、動的束縛と呼びます。



### 例

#### 一般従業員と管理職

一般従業員クラスも管理職クラスも、従業員クラスを継承しています。そして、それぞれのクラスは、固有の“給与計算”というメソッドを持っています。従業員クラスのオブジェクトを参照するデータ項目を用意すると、それは一般従業員オブジェクトも管理職オブジェクトも参照できます。一般従業員オブジェクトと管理職オブジェクトを使った多態と動的束縛の例を、以下の図に示します。



← 右辺のオブジェクトを左辺のデータ項目に代入することを示します。  
 :: 左辺のデータ項目が参照するオブジェクトに対して、右辺のメソッドを呼び出すことを示します。

xは、従業員クラスのオブジェクトを参照するデータ項目です。そのため、xには一般従業員クラスのオブジェクト([1])も管理職クラスのオブジェクト([3])も代入できます。また、[2]と[4]でメソッドを呼び出しています。これらはまったく同じ構文ですが、実際に呼び出されるメソッドは異なります。[2]では一般従業員クラスの給与計算メソッドが、[4]では管理職クラスの給与計算メソッドが呼び出されます。

このような作りにおくと、すべての従業員の処理を一箇所でできるだけでなく、新たな勤務形態(たとえばパートタイム)を追加しても、コードを修正する必要がありません。

## 15.3 オブジェクト指向のメリット

---

オブジェクト指向プログラミングには、以下の特徴があります。

- データと手続きをカプセル化し、外部から隠蔽することにより、オブジェクトとの接点はインタフェースだけになります。
  - ー 内部の実現方法(データ構造や手続きのコード)を変更しても、インタフェースさえ変更しなければ、それを利用するプログラムには影響がありません。つまり、変更に対する影響を局所化できます。
  - ー 公開されているのはインタフェースだけなので、あらかじめ決められた正しい方法以外では使用できません。その結果、必然的にプログラムの信頼性が向上します。
- 継承により、親クラスの性質を subclasses に引き継ぐことができます。
  - ー 既存のクラスをもとに新しいクラスを作る場合、差分だけのコーディングですみます。親クラスの変更は subclasses にも反映されるので、変更時の影響を局所化できます。
  - ー クラス間の共通の性質を抜き出して、1つのクラスにまとめることができます。共通処理が一箇所にまとめられているので、変更時の影響を局所化できます。
- 多態により、柔軟なプログラミングができます。
  - ー 1つの手続きで複数のクラスを扱うことができます。そのため、扱うクラス(従来ならデータ)の違いにより処理を分ける必要はありません。
  - ー 新しいクラスを追加しても、処理を変更する必要はありません。

これらの特徴から、オブジェクト指向を導入することにより、以下の効果が期待できます。

### 作成時

- ー 部品化が容易にできます。
- ー 既存の部品の流用が容易にできます。 → 開発作業の効率向上

### 保守時

- ー システム変更時の影響を局所化できます。
- ー システムの変更に対し、柔軟に対応できます。 → 保守作業の効率向上

このように、オブジェクト指向言語を導入することにより、開発効率および保守効率を大幅に向上させることができます。

## 第16章 オブジェクト指向プログラミング機能～基本的な使い方～

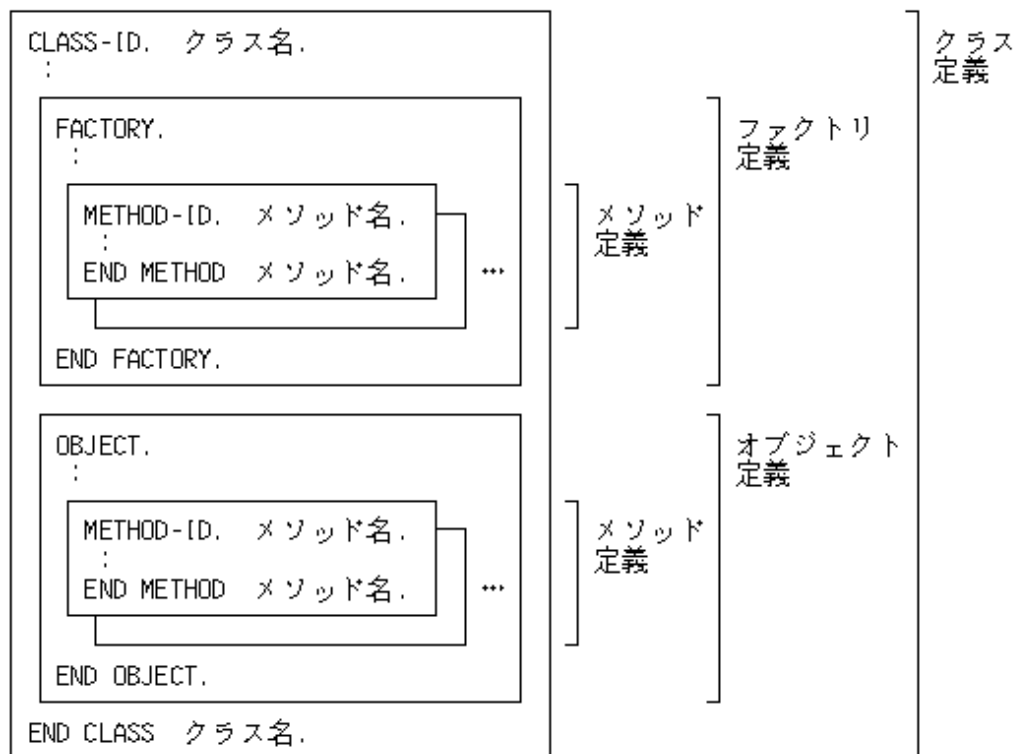
本章では、“[第15章 オブジェクト指向プログラミングとは](#)”で紹介したオブジェクト指向プログラミングについて、より具体的にコーディングレベルで説明します。

### 16.1 ソース定義

オブジェクト指向プログラミングでは、オブジェクトおよびオブジェクトを操作するためのメソッドを定義するために、従来のプログラム定義(プログラム始め見出し～プログラム終わり見出しで構成)に加えて以下の定義が追加されます。

- ・ クラス定義
- ・ ファクトリ定義
- ・ オブジェクト定義
- ・ メソッド定義

これら定義の関係は、下図のとおりです。



それぞれの定義について、具体的な定義方法や役割について説明します。

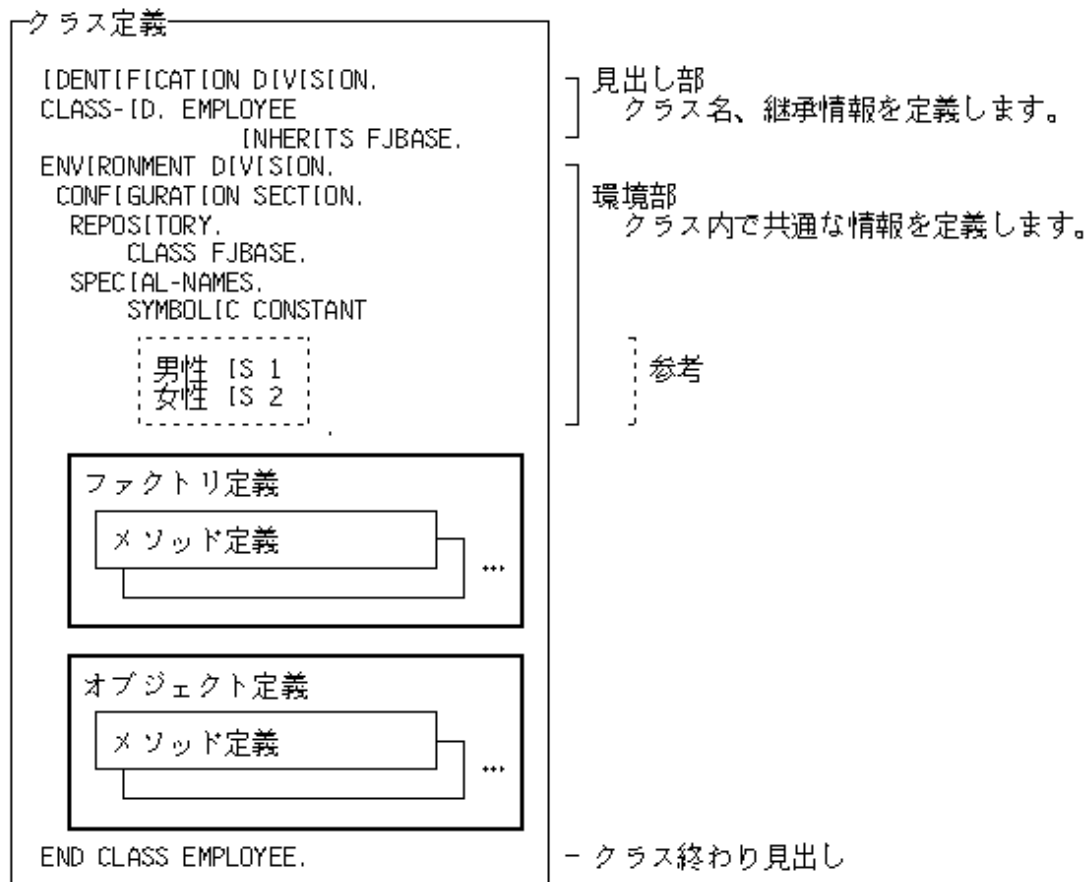
なお、以降の説明は、当製品に添付してあるサンプル(従業員管理プログラム)を基に行っています。ただし、より理解しやすくするため、データや処理を簡素化したり、クラス構成やメソッド、データ名、利用している機能などを変えていますので、あらかじめご了承ください。

#### 16.1.1 クラス定義

クラス定義は、オブジェクトを定義するときに基本となる定義で、データおよびそのデータを操作するための手続き(メソッド)を1つにカプセル化したものです。



クラス定義は、オブジェクトの管理(生成や共通情報の定義など)を行うファクトリ定義とオブジェクトの属性や形式の定義、データの操作を行うオブジェクト定義から構成されます。つまり、クラス定義は、ファクトリ定義とオブジェクト定義を入れておく入れ物(枠)のようなものです。そのため、クラスの継承情報を定義するために見出し部を、クラス定義内で共通の情報を定義するために環境部を定義することはできますが、データや手続きを記述することはできません。



クラス定義の環境部で宣言されたデータ、つまり、リポジトリ段落で宣言されたクラス名、特殊名段落で宣言された機能名や呼び名、記号定数などの有効範囲はクラス定義内のすべてのソース定義です(上図の太線で囲まれている部分)。

## 16.1.2 ファクトリ定義

ファクトリ定義は、オブジェクトに共通なデータ(ファクトリデータと呼びます)を定義したり、オブジェクトの管理(生成など)を行うメソッド(ファクトリメソッドと呼びます)を定義します。

ファクトリ定義は、ファクトリ定義を表すための見出し部、ファクトリデータを定義するための環境部およびデータ部、ファクトリメソッドを定義するための手続き部から構成されます。



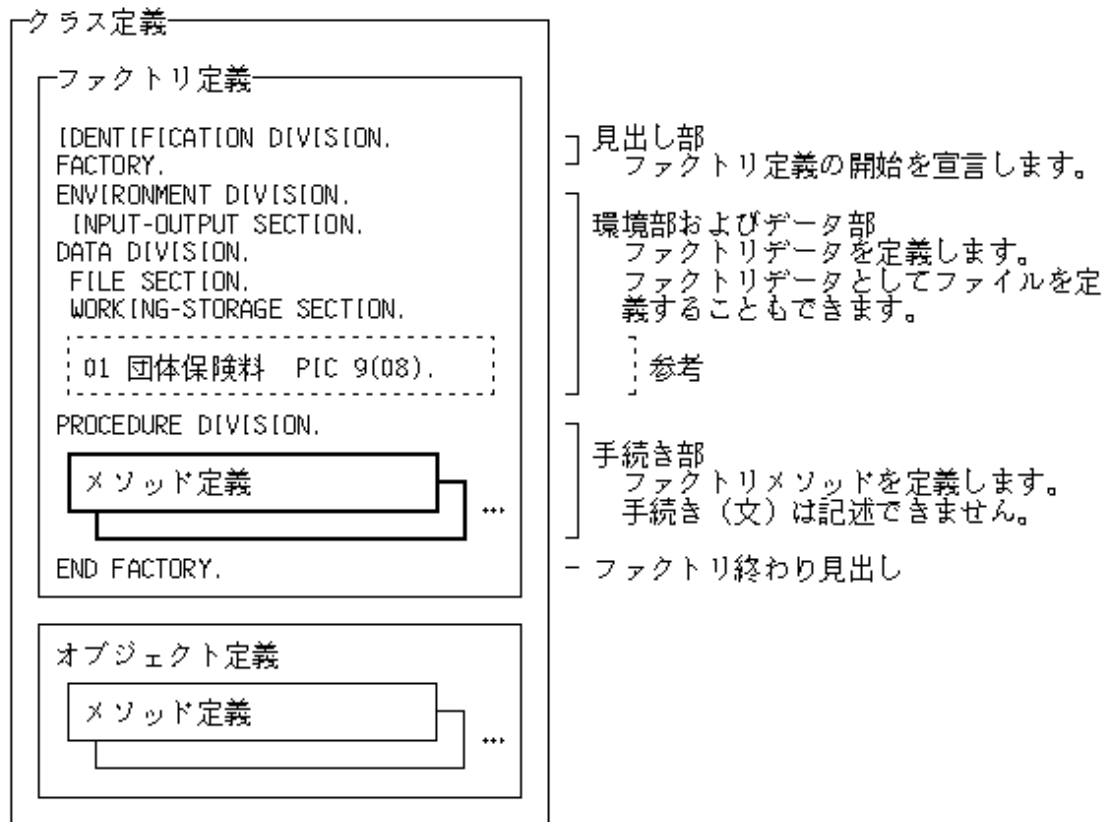
注意

手続き部は、ファクトリメソッドを定義するだけであり、手続き(COBOLの文)を記述することはできません。



参考

これらの定義を必要としないクラスの場合、ファクトリ定義(ファクトリ始め見出し～ファクトリ終わり見出しまで)を省略することもできます。



ファクトリ定義の環境部およびデータ部で定義されたデータは、ファクトリメソッドでだけアクセスすることができます(上図の太線で囲まれている部分)。

では、具体的にファクトリ定義の役割について説明します。

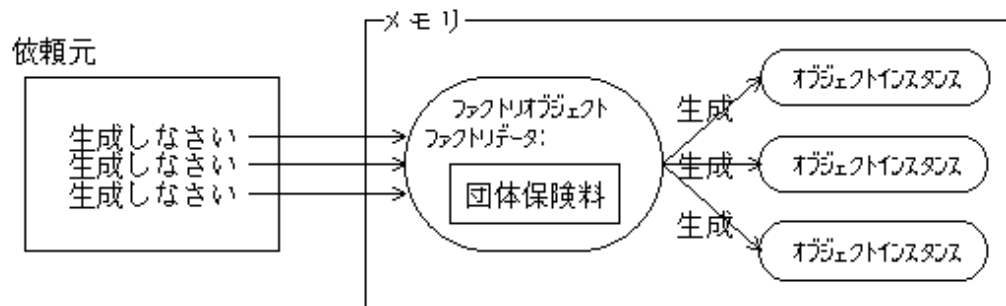
オブジェクトインスタンスの概念については“[第15章 オブジェクト指向プログラミングとは](#)”で説明しました。これとは別に、COBOLにはファクトリオブジェクトという概念があります。

## ファクトリオブジェクト

オブジェクトインスタンスが複数個生成されるのに対して、このファクトリオブジェクトは、1クラスにつき、1個だけしか存在しません。このファクトリオブジェクトを定義するのがファクトリ定義です。

ファクトリオブジェクトは、アプリケーションが起動されたタイミングでメモリ上に生成され、最後までメモリ上に存在し続けます。

たとえば、あるオブジェクトインスタンスを生成する場合、そのオブジェクトが定義されているクラスに対して「生成しなさい」という指示を出します。その指示を受け取るのがファクトリオブジェクトです。ファクトリオブジェクトは、その指示を受け取ると、新しくオブジェクトインスタンスを生成します。



上図では、ファクトリ定義が持つ代表的な機能である「生成」の動作を説明しています。この図からわかるように、ファクトリオブジェクトとは、その名のとおりオブジェクトインスタンスを生成する「工場」なのです。

通常、この「生成処理」は、FJBASEクラス(注)を継承することによってクラスに組み込まれるため、「生成処理」を利用者がコーディングする必要はありません。では、他に何を定義するかというと、生成したオブジェクトインスタンスを初期化するメソッドや、生成された複数のオブジェクトインスタンスで共通に利用されるデータなどを定義しておきます。たとえば、上の例のように、ファクトリデータとして「団体保険料」を定義しておきます。これは、給与計算時に全従業員を対象に給与天引きする額であり、ファクトリメソッドを呼び出すことにより設定、参照することができます。クラス共通情報をファクトリデータとして保持することによって、額の増減などに容易に対応できるようになります。

注：標準で提供。詳細は“[16.3.2 FJBASEクラス](#)”を参照してください。

### 16.1.3 オブジェクト定義

オブジェクト定義には、オブジェクトデータの定義およびオブジェクトを操作するためのメソッド(オブジェクトメソッドといいます)を定義します。

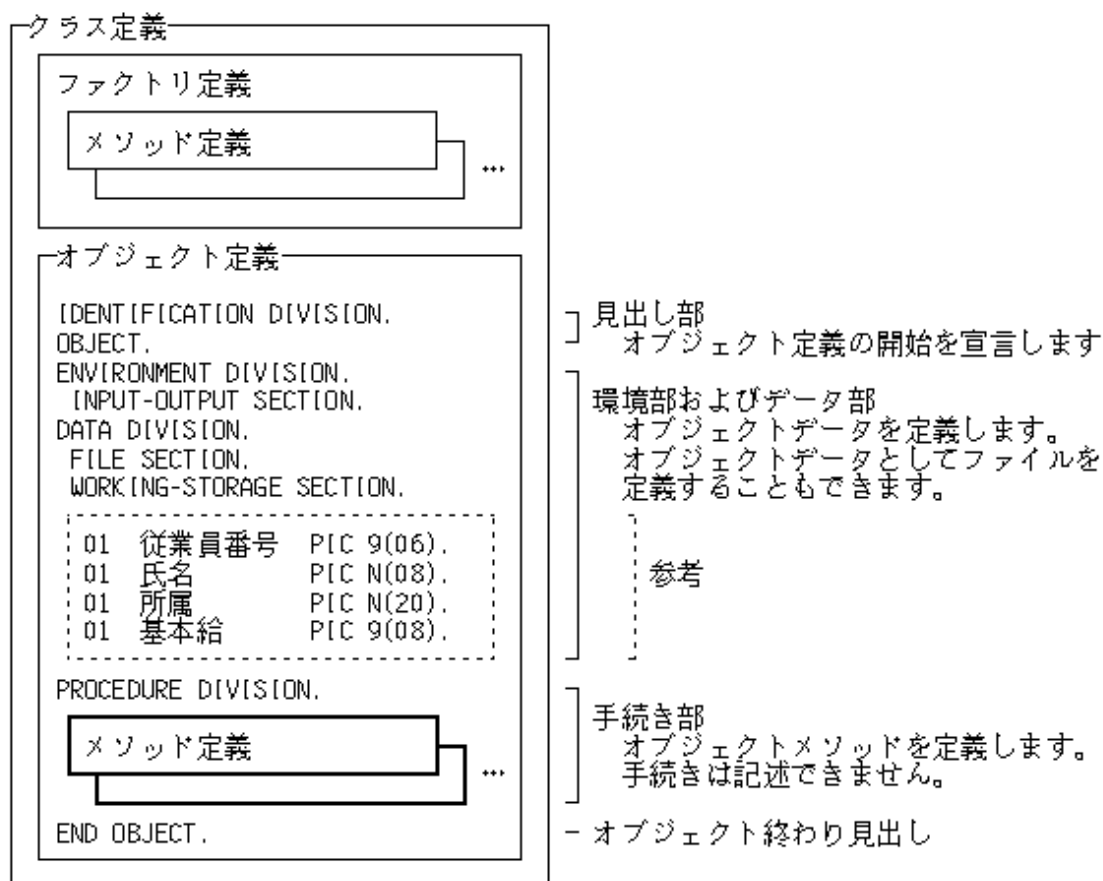
オブジェクト定義の構成はファクトリ定義と同じで、オブジェクト定義を表すための見出し部、オブジェクトデータを定義するための環境部およびデータ部、オブジェクトを定義するための手続き部からなります。

#### 注意

手続き部は、オブジェクトメソッドを定義するだけであり、手続きを記述することはできません。

#### 参考

オブジェクト定義(オブジェクト始め見出し～オブジェクト終わり見出しまで)を省略することもできます。

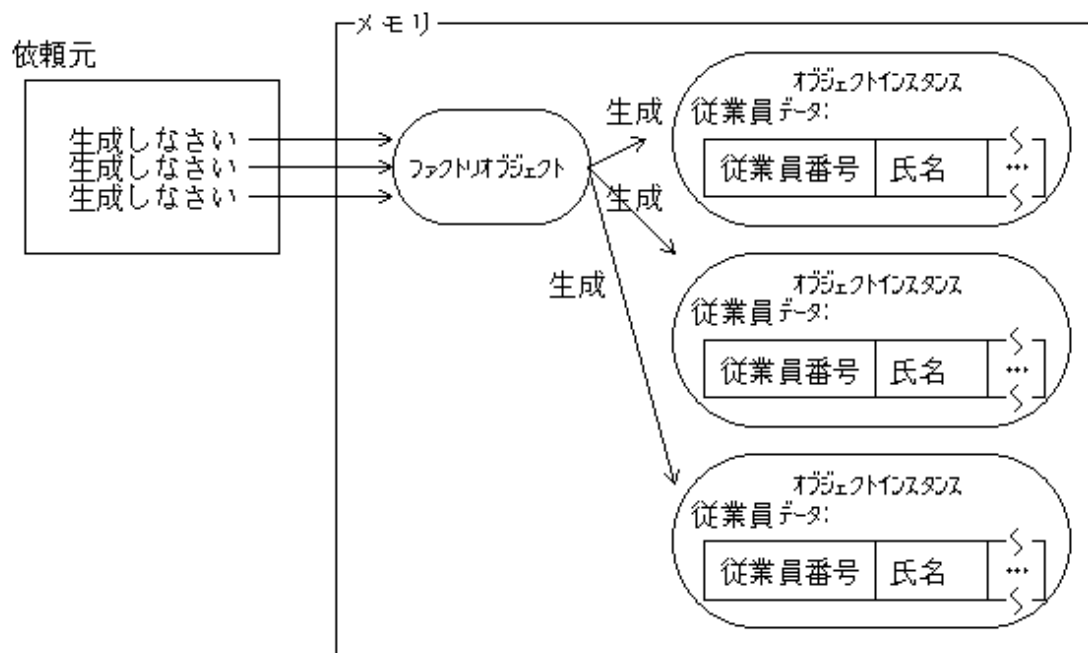


オブジェクト定義の環境部およびデータ部で定義されたデータは、オブジェクトメソッドでだけアクセスすることができます(上図の太線で囲まれている部分)。

では、具体的にオブジェクトデータには何を、オブジェクトメソッドとしてどのようなメソッドを定義すればよいかについて説明します。

オブジェクトデータの定義をデータ部(および環境部)に、そのオブジェクトデータを操作するための手続きをオブジェクトメソッドとして定義します。

たとえば、上図のように、従業員に関するデータを記述した場合、その従業員データがオブジェクトデータになります。この場合、実行中のオブジェクトインスタンスを表すと、下図のとおりになります。

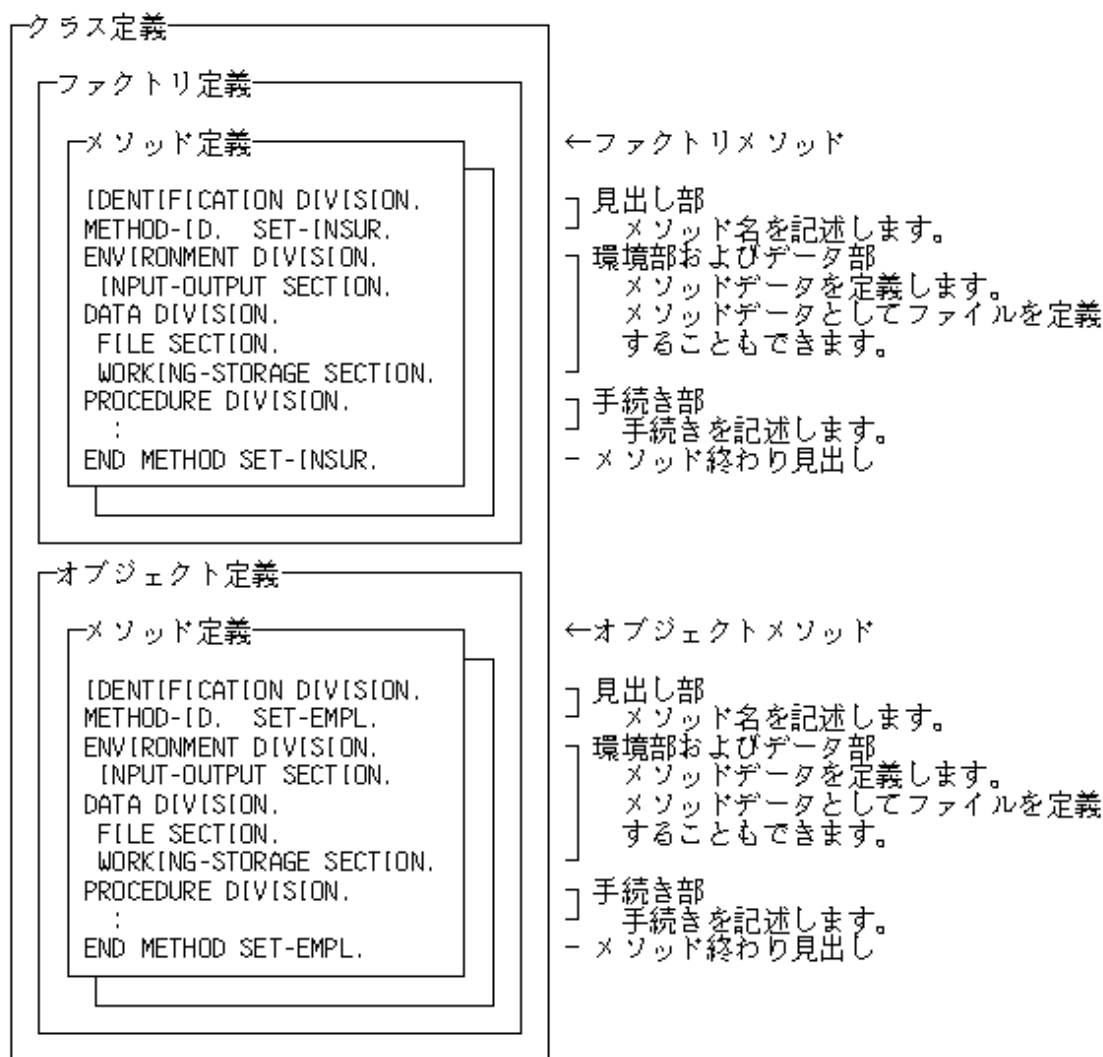


### 16.1.4 メソッド定義

メソッド定義には、オブジェクトインスタンスを管理するファクトリメソッドおよびオブジェクトデータを操作するオブジェクトメソッドがあります。

メソッド定義の構成は、メソッド名を定義するための見出し部、メソッドデータを定義するための環境部およびデータ部、手続きを記述するための手続き部からなります。つまり、従来の内部プログラムと同じ構成を持つことができます。また、ファクトリメソッドおよびオブジェクトメソッドの数に制限はないため、それぞれ必要なだけ定義することができます。

なお、ファクトリメソッドとオブジェクトメソッドは同じ構成です。ファクトリ定義内に定義されたメソッドをファクトリメソッド、オブジェクト定義内に定義されたメソッドをオブジェクトメソッドと呼び分けます。

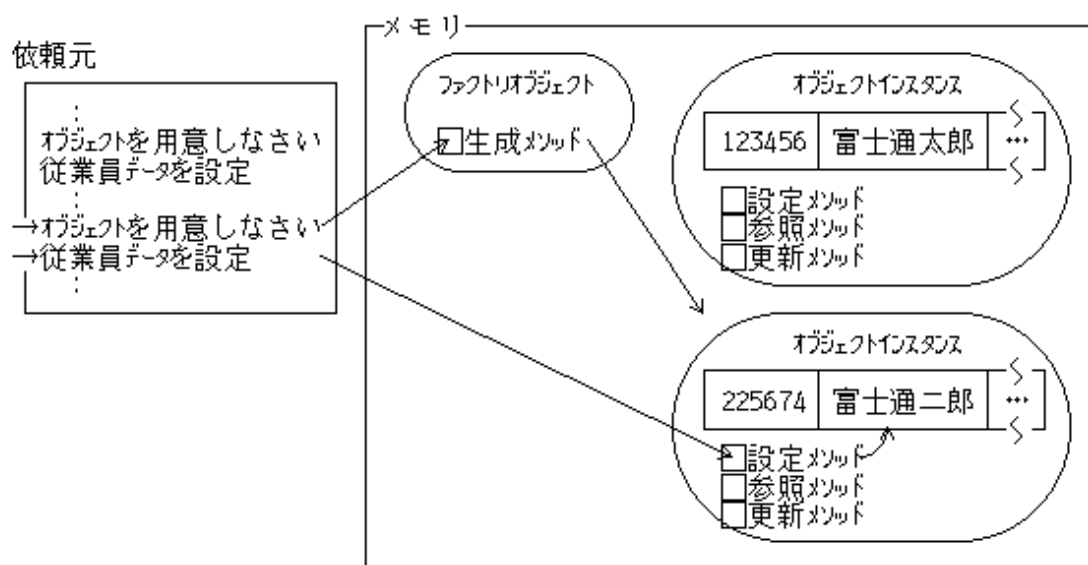
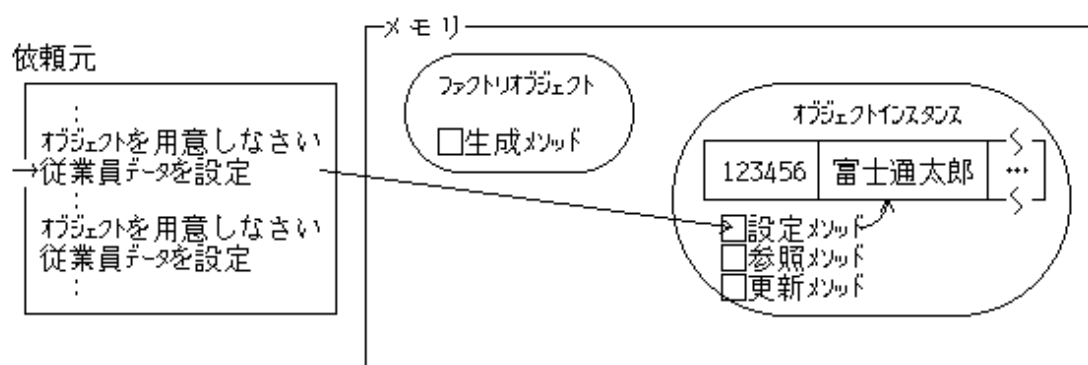
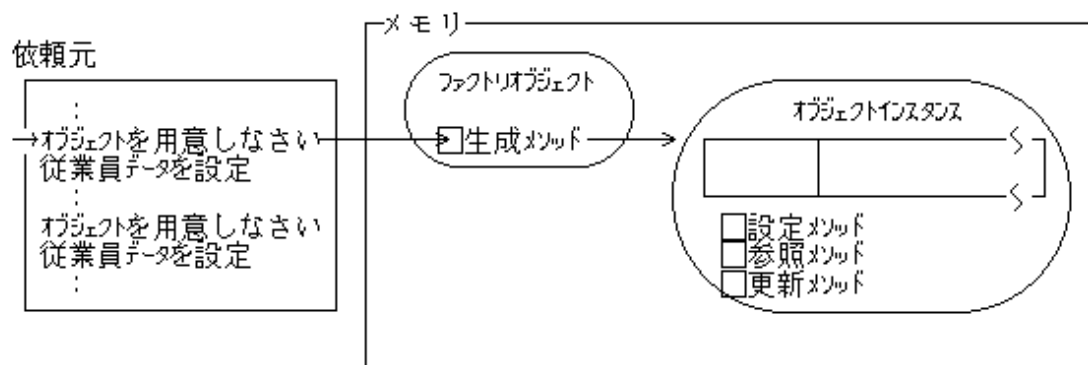


メソッド定義には、ファクトリメソッドとオブジェクトメソッドがあります。どちらも、メソッド内で定義されたデータは、そのメソッド内でだけアクセスすることができます。

では、メソッドの役割について説明します。

ファクトリデータおよびオブジェクトデータは、外部から隠蔽されています。これらのデータを操作(取出しや更新など)するためには、それぞれにメソッドを用意するしかありません。つまり、ファクトリデータは、ファクトリメソッドを介してしかアクセスできません。また、オブジェクトデータは、オブジェクトメソッドを介してしかアクセスできません。

メソッドの実行時イメージは、それぞれのオブジェクトインスタンス中にメソッド(手続き)が存在すると考えた方が分かりやすいでしょう。つまり、オブジェクトインスタンスの操作は以下のイメージとなります。



## 16.2 オブジェクトインスタンスの操作

ここでは、オブジェクトインスタンスの操作方法について説明します。

### 16.2.1 メソッドの呼出し

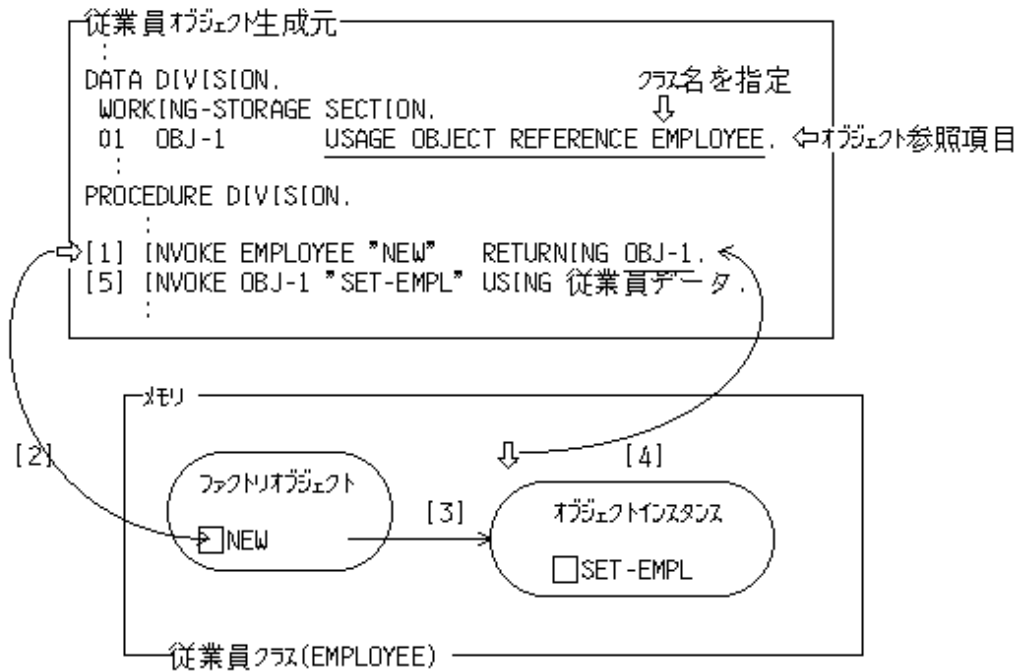
オブジェクトインスタンスを操作するためには、オブジェクトメソッドを呼び出す必要があります。このとき、「どのオブジェクトインスタンス」の「どのメソッド」を呼び出すかを指定します。「どのオブジェクトインスタンス」を表現するのに、オブジェクト参照項目と呼ばれるデータ項目を利用します。

### 16.2.1.1 オブジェクト参照項目

オブジェクト参照項目は、USAGE OBJECT REFERENCE句を指定することにより定義できます。用途は、オブジェクト参照の格納用です。そのため、主にメソッドの呼出し(INVOKE文)で利用されます。

オブジェクトインスタンスを生成するメソッドを呼び出すと、生成したオブジェクトのオブジェクト参照(アドレスに相当)が返却されます。以降、このオブジェクトインスタンスを操作する場合には、このオブジェクト参照項目を使用します。

図16.1 従業員オブジェクトインスタンスの生成



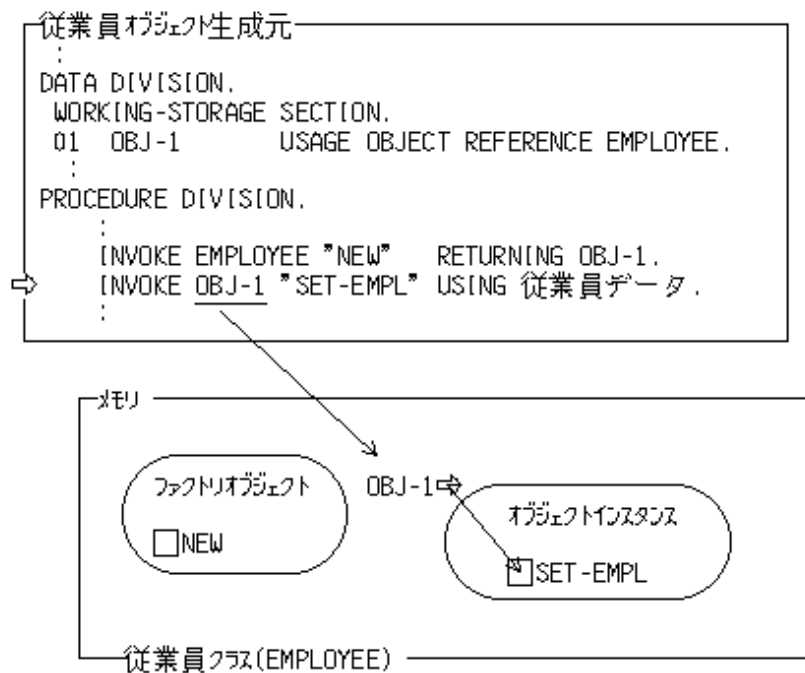
[1][2] : NEW メソッドを呼び出す。

[3] : NEW メソッドはオブジェクトインスタンスを生成する。

[4] : オブジェクト参照を呼出し元へ返却する。



図16.2 従業員オブジェクトインスタンスへのデータ登録



上図のとおり、生成したオブジェクトインスタンスを操作する場合(オブジェクトメソッドを呼び出す場合は、処理対象となるオブジェクトインスタンスを指しているオブジェクト参照項目を指定しなければなりません。

オブジェクト参照項目は、SET文を用いて他のオブジェクト参照項目に値を代入することができます(MOVE文による転記はできません)。また、IF文などにより、内容を比較することもできます。ただし、この場合、代入または比較されるのはオブジェクト参照データであり、オブジェクトインスタンスが代入または比較されるわけではないので、注意してください。

```

:
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ-1 USAGE OBJECT REFERENCE EMPLOYEE.
01 OBJ-X USAGE OBJECT REFERENCE EMPLOYEE.
:
PROCEDURE DIVISION.
:
 INVOKE EMPLOYEE "NEW" RETURNING OBJ-1.
 SET OBJ-X TO OBJ-1.
 INVOKE OBJ-X "SET-EMPL" USING 従業員データ.
:

```

## 注意

- 初期値はNULLで、VALUE句により初期値を与えることはできません。
- オブジェクト参照項目の内部形式については、意識しないようにコーディングしてください。COBOLシステムが管理するデータのため、無理に内容を変更した場合、正常に動作できなくなります。
- CALL文でオブジェクト参照項目を受け渡す場合、USAGE句が一致していないと正しく受け渡すことができません。

### 16.2.1.2 INVOKE文

従来のプログラム定義の場合、別プログラムの呼出しにはCALL文を利用していましたが、メソッドを呼び出す場合には、INVOKE文を利用しなければなりません。INVOKE文は、「どのオブジェクト」の「どのメソッド」を「どのようなパラメタ」で呼び出すかを指定できるようになっています。

以下に“[図16.1 従業員オブジェクトインスタンスの生成](#)”のINVOKE文について説明します。

[1]は、オブジェクトインスタンスを生成するためにEMPLOYEEクラスのNEWメソッドを呼び出しています。ここでは、

- ・「どのオブジェクト」 → ファクトリオブジェクトの、(注)
- ・「どのメソッド」 → NEWメソッドを、
- ・「どのようなパラメタ」 → OBJ-1(オブジェクト参照)を復帰値として

呼び出す。という意味になります。

注：通常、ファクトリオブジェクトはクラス名で表現されます。

[5]は、[1]で生成したオブジェクトインスタンスに初期データとして従業員の情報を設定するためにSET-EMPLメソッドを呼び出しています。

このときのINVOKE文は、

- ・「どのオブジェクト」 → OBJ-1で表されるオブジェクトインスタンスの、
- ・「どのメソッド」 → SET-EMPLメソッドを、
- ・「どのようなパラメタ」 → 従業員情報を入力として

呼び出す。という意味になります。



#### 注意

INVOKE文にメソッド名を識別する一意名を指定した場合、メソッド名として有効となる文字列の最大は、指定された領域の先頭から255バイトまでです。256バイト以降の文字列は無視されます。また、このとき、文字列の後ろに埋められた空白も無視されます。

### 16.2.1.3 パラメタの指定

CALL文で、呼出し先モジュールに対してパラメタが指定できたのと同様に、INVOKE文についても、呼出し先メソッドに対してパラメタを指定することができます。

パラメタの指定は、USING指定およびRETURNING指定により行います。

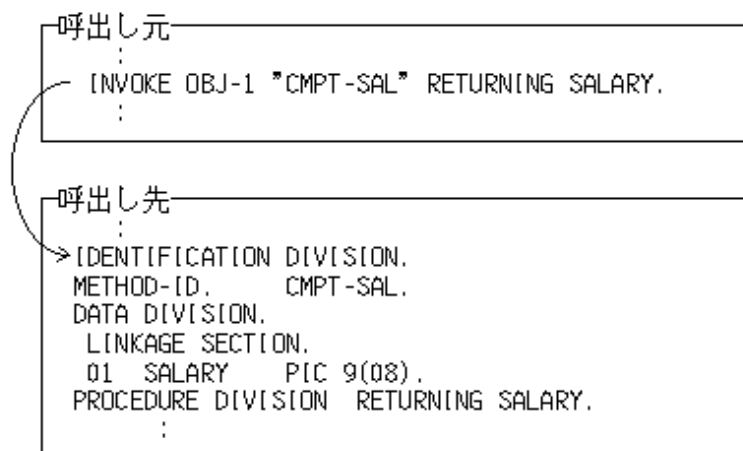
#### USING指定

これは従来のCALL文と同じ使い方で、呼び出されるメソッドの連絡節および手続き部見出しでのパラメタ定義によって、データの受渡しが可能になります。



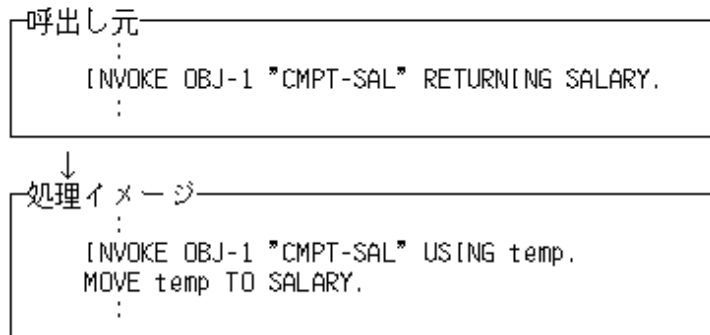
## RETURNING指定

RETURNING指定は、呼出し先からの復帰値を受け取るために指定するもので、USING指定とは少し意味合いが異なります。また、USING指定が複数個のパラメタを指定できるのに対し、RETURNING指定は、1つだけ指定可能です。



RETURNING指定は、復帰値であるため、呼出し元で設定された値を呼出し先で参照することはできません。つまり、一方通行の関係となります。

呼出し元の処理イメージは、下図のとおりです。



USING指定とRETURNING指定を同時に指定することもできます。それぞれの用途に合わせて利用してください。

## 16.2.2 オブジェクトの寿命

---

ここでは、オブジェクトの削除について説明します。

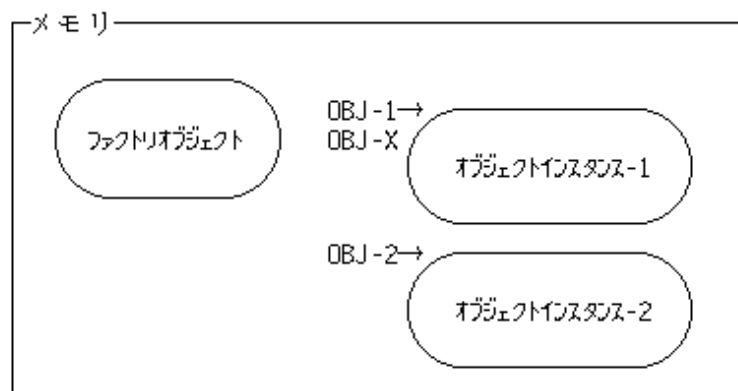
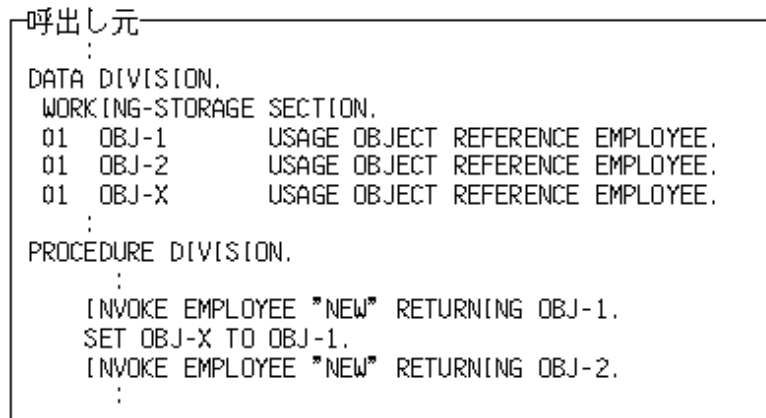
### ファクトリオブジェクトの寿命

ファクトリオブジェクトは、クラスがローディングされてからCOBOLの実行環境が閉鎖されるまでメモリ上に存在し続けます。なお、アプリケーションの動作中に削除する手段はありません。

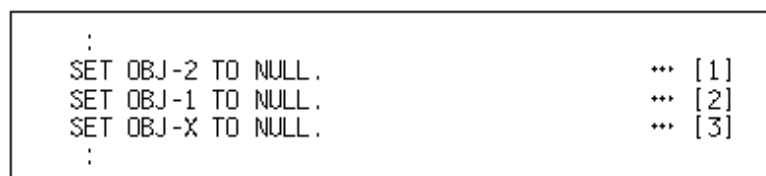
### オブジェクトインスタンスの寿命

オブジェクトインスタンスは、どこからも参照されなくなったとき、つまり、そのオブジェクトインスタンスのオブジェクト参照を持つオブジェクト参照項目がなくなったときに削除されます。このため、オブジェクトインスタンスが不要になったときは、そのオブジェクトインスタンスのオブジェクト参照を持つオブジェクト参照項目にNULLオブジェクトを代入し、初期化してください。そして、必ず、オブジェクト参照項目にNULLオブジェクトを代入してから実行単位を終了してください。このような終り方をしない場合、COBOLランタイムシステムは、COBOLの実行環境閉鎖時に、残っているオブジェクトインスタンスを強制的にメモリ上から解放します。このとき、終了処理メソッド `_FINALIZE` は呼び出されません。また、マルチスレッドプログラムでは、メモリリークが発生するため注意が必要です。[参照]“[24.10.2 オブジェクト指向プログラミング機能](#)”

以下に、オブジェクトインスタンスの削除を具体的に説明します。



上図の状態で、以下の手続きが実行された場合



[1]の実行により、OBJ-2だけで管理されていたオブジェクトインスタンス-2は、削除されます。

[2]の実行により、OBJ-1にNULLオブジェクトを代入しても、オブジェクトインスタンス-1はOBJ-Xによって管理されているため、削除されません。

[3]の実行により、さらにOBJ-XにNULLオブジェクトを代入すると、オブジェクトインスタンス-1を管理しているオブジェクト参照項目はなくなるため、削除されます。

ただし、ここでいう「削除」とは、アプリケーションから論理的に見えなくなるだけであり、メモリ上から解放されるわけではありません。メモリ上からの解放は、COBOLランタイムシステムが最適なタイミングで自動的に行います。これをガーベージコレクションと呼びます。

## 16.3 継承

オブジェクト指向プログラミングには、継承と呼ばれる概念があります。この継承を利用することにより、下記のメリットを得ることができます。

- ・ 既存の部品の流用が容易にできる。
- ・ システムの変更に對し、柔軟に對応できる。

ここでは、継承の概念や利用方法について、具体例を用いて説明します。

## 16.3.1 継承の概念と実現

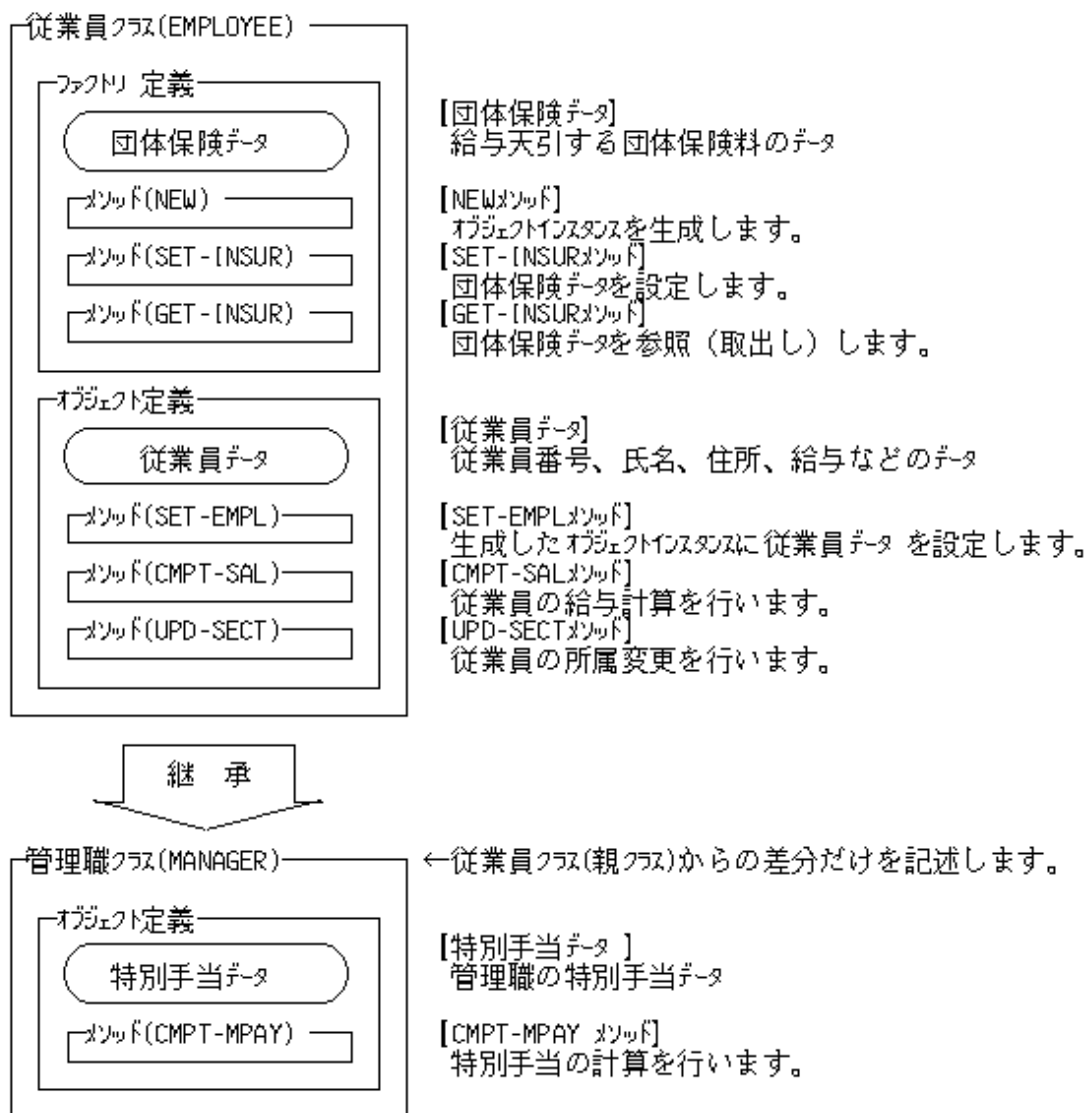
継承の概念については、すでに“[第15章 オブジェクト指向プログラミングとは](#)”で説明しているので、詳しい説明は省略します。簡単に復習してみましょう。

### 継承の概念

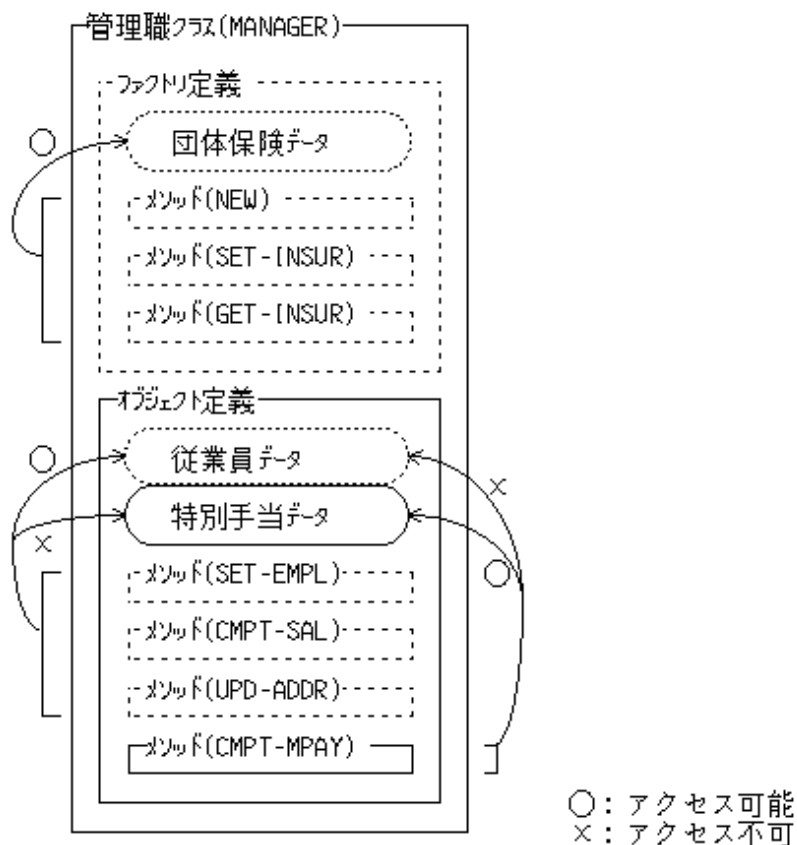
従来のプログラミング手法では、既存の部品(ルーチン)とよく似た機能を持った部品を作成する場合、既存のソースプログラムを複写し、複写したソースプログラムをベースにしてプログラミングする方法をとっていました。

ところが、オブジェクト指向の場合、既存の部品(クラス)との差分をコーディングするだけで同様のことが実現できます。つまり、「あるクラスが持つ機能をすべて引き継ぐ」ことが簡単にできるのです。これを継承と呼びます。

ここでは、具体例として、従業員クラスを継承した管理職クラスを作成します。



この場合、管理職クラスの論理的な構成は、以下のとおりになります。



上図の実線は明示定義されたデータおよびメソッドを、点線は継承によって暗黙定義されたデータおよびメソッドを表しています。

メソッド呼出しの場合には、明示定義または暗黙定義を区別する必要はありません。暗黙定義されたメソッドも明示定義されたメソッドと同じように呼び出すことができます。

また、継承の階層(深さ)に制限はないので、必要に応じて継承を利用してください。ただし、あまり階層が深すぎると資源の管理が大変になるので、極端に深くならないように設計することをおすすめします。

なお、継承関係にあるクラスを表現する場合、あるクラスから派生したクラス(継承したクラス)を子クラス、継承されたクラスを親クラスと呼びます。上図では、従業員クラス(EMPLOYEE)が親クラスで、管理職クラス(MANAGER)が子クラスになります。

## データのアクセス

暗黙定義されたデータ(団体保険データ、従業員データ)および明示定義されたデータ(特別手当データ)のアクセスについて説明します。

ファクトリデータおよびオブジェクトデータは、そのクラス定義で明示定義されたメソッドでだけアクセスすることができます。つまり、上図の場合、オブジェクトデータ(オブジェクトインスタンス)としては、従業員データと特別手当データの両方を持ちますが、明示定義された特別手当データは、明示定義されたオブジェクトメソッド(CMPT-MPAY)でだけアクセス可能です。逆に、暗黙定義された従業員データは、暗黙定義されたオブジェクトメソッド(SET-EMPL、CMPT-SALおよびUPD-SECT)でだけアクセス可能になります。

## 継承の定義方法

では、実際に継承を定義してみましょう。

継承は、クラス名段落(CLASS-ID)のINHERITS句に親クラス名を指定することで実現できます。このとき、環境部のリポジトリ段落に必ず親クラスを宣言してください。

### 管理職クラス

```

** クラス定義 **

```

|                             |                   |
|-----------------------------|-------------------|
| IDENTIFICATION DIVISION.    |                   |
| CLASS-ID. MANAGER           |                   |
| <u>INHERITS EMPLOYEE.</u>   | ←親クラスを指定します。      |
| ENVIRONMENT DIVISION.       |                   |
| CONFIGURATION SECTION.      |                   |
| <u>REPOSITORY.</u>          |                   |
| <u>CLASS EMPLOYEE.</u>      | ←親クラスを宣言します。      |
| **-----**                   |                   |
| ** オブジェクト定義 **              |                   |
| **-----**                   |                   |
| IDENTIFICATION DIVISION.    |                   |
| OBJECT.                     |                   |
| DATA DIVISION.              | ←オブジェクトデータの定義     |
| WORKING-STORAGE SECTION.    | 追加するデータだけを指定します。  |
| 01 MPAY          PIC 9(08). | (差分だけをコーディング)     |
| PROCEDURE DIVISION.         |                   |
| **-----**                   |                   |
| ** メソッド定義 **                |                   |
| **-----**                   |                   |
| IDENTIFICATION DIVISION.    | ←オブジェクトメソッドの定義    |
| METHOD-ID. CMPT-MPAY.       | 追加するメソッドだけを指定します。 |
| :                           | (差分だけをコーディング)     |
| END METHOD CMPT-MPAY.       |                   |
| END OBJECT.                 |                   |
| END CLASS MANAGER.          |                   |

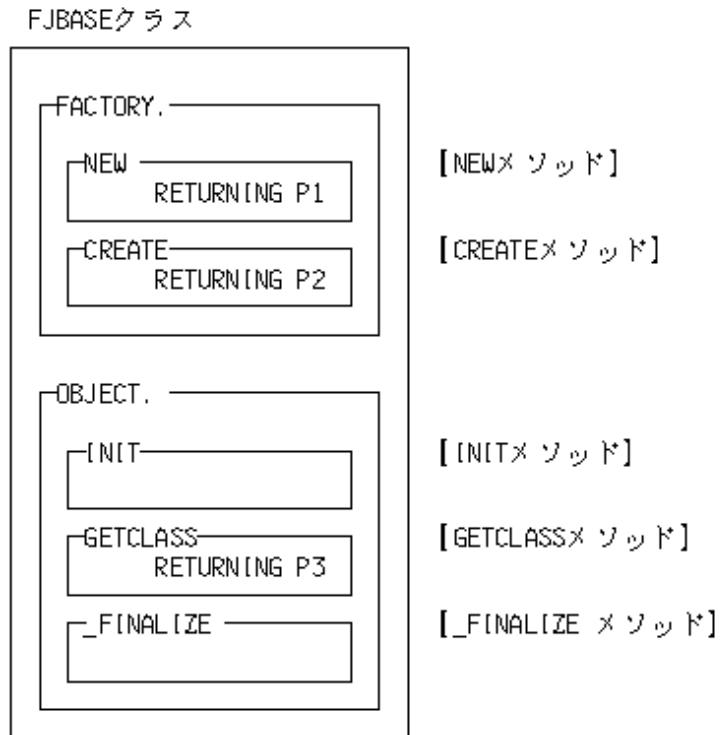
このとおり、簡単に定義することができます。つまり、現存するアプリケーションの機能追加に容易に対応できることになります。

## 16.3.2 FJBASEクラス

COBOLシステムでは、汎用的に利用するクラスを標準で提供しています。その1つにFJBASEクラスと呼ばれる、すべてのクラスで必要と思われるメソッド(たとえば、オブジェクトインスタンスの生成など)を定義したクラスがあります。新規にクラスを作成する場合は、このFJBASEクラスを継承することによって、これらの機能を簡単に組み込むことができます。

以下に、FJBASEクラスについて説明します。



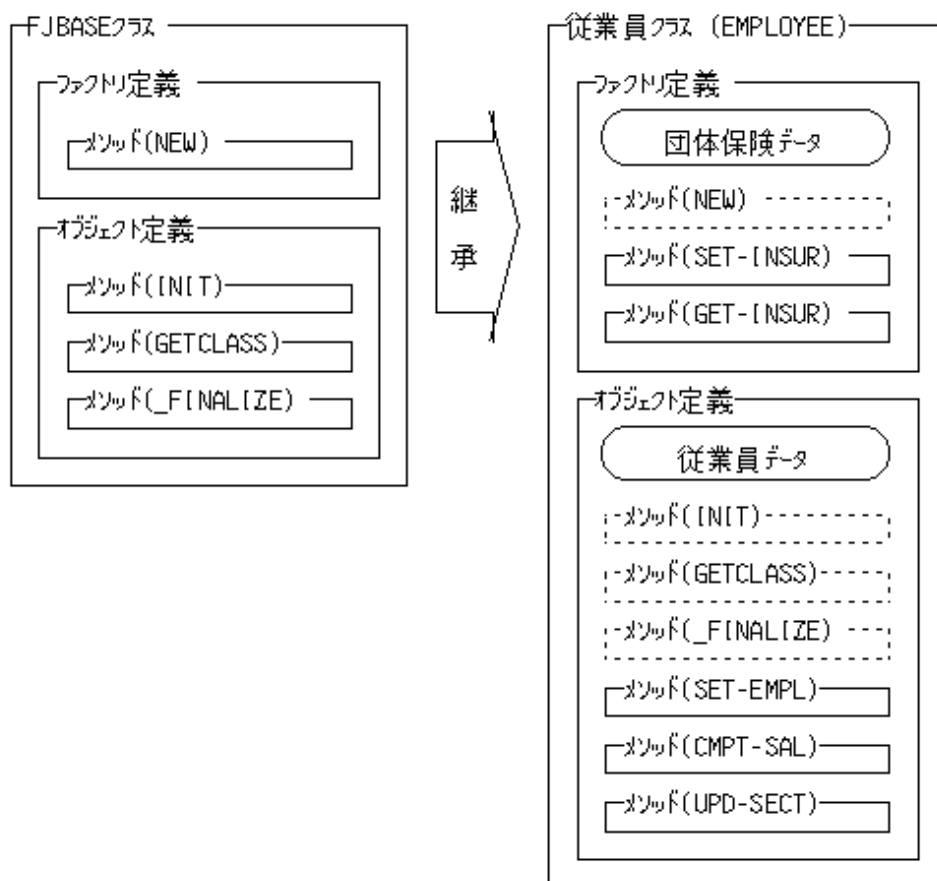


各メソッドの詳細については、“COBOL文法書”を参照してください。

INITメソッドおよび\_FINALIZEメソッドについては、“[16.7.6 初期化処理メソッドと終了処理メソッド](#)”を参照してください。

通常、利用されるのはNEWメソッドとGETCLASSメソッドです。その他のメソッドは、より高度なプログラミングを行う場合にだけ利用されます。しかし、継承はクラス定義に対して行われるため、NEWメソッドだけ必要であっても、他のメソッド(CREATEメソッド、INITメソッド、GETCLASSメソッドおよび\_FINALIZEメソッド)も組み込まれることになります。ただし、メソッドを呼び出さないかぎりは何も影響を与えないため、あまり意識する必要はありません。また、FJBASEクラスはオブジェクトデータを持っていません。したがって、FJBASEクラスを継承することによって、オブジェクトデータが大きくなるという心配もありません。新しくクラスを定義する場合は、必ずFJBASEクラスを継承すると考えてください。

なお、これまでの説明では、NEWメソッドは従業員クラス(EMPLOYEE)で定義されているように表現してきましたが、実際には、FJBASEクラスを継承することによって暗黙定義されたメソッドだったのです。つまり、以下の継承関係があったことを付け加えておきます。



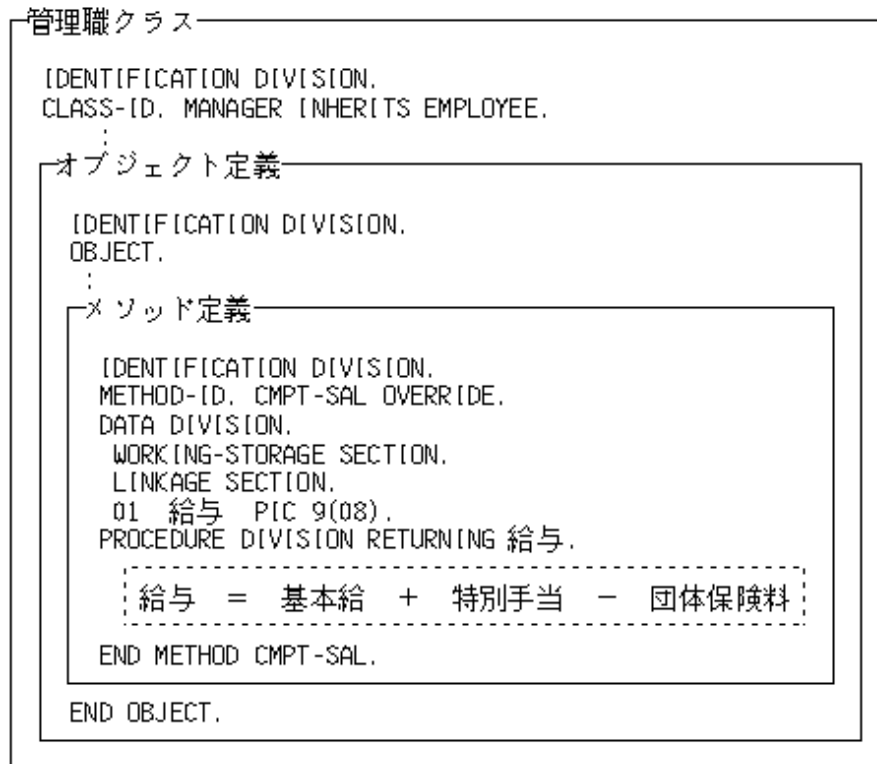
### 16.3.3 メソッドの上書き

クラスを継承する場合、「メソッド名やインタフェースは同じでよいが、処理を少し変更(追加)したい」ということが多くあります。このようなとき、継承をあきらめて新規に似たようなクラスを作成する必要はありません。OVERRIDE句を利用することによって、メソッドを上書きすることができます。

従業員クラスを継承した管理職クラスの場合を考えてみましょう。

管理職の場合、給与計算メソッド(CMPT-SAL)で、「特別手当を加算する」必要があります。つまり、従業員クラスから継承したCMPT-SALメソッドに処理を加える必要があります。

このような場合、OVERRIDE句を利用してメソッドを上書きすることができます。



メソッドの上書きは、直接の親クラスで明示または暗黙に定義されたメソッドに対して行うことができます。また、親クラスで上書きされているメソッドをさらに上書きすることも可能です。

ただし、インタフェース(パラメタ)は上書きされるメソッドと同じでなければなりません。

## 16.4 適合

オブジェクト指向プログラミングには、適合と呼ばれる概念があります。適合とは、クラス間の関係を表現するもので、オブジェクト参照項目を利用(操作)する場合に意識する必要があります。

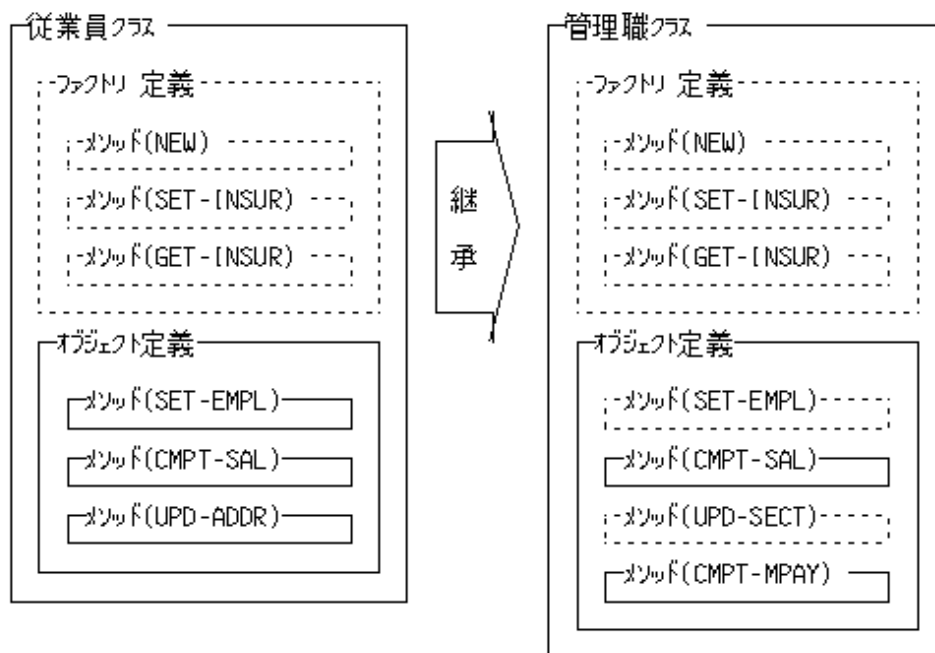
ここでは、適合の概念や規則について、具体例を用いて説明します。

### 16.4.1 適合の概念

オブジェクト指向では、メソッドを呼び出す場合、呼出し時のインタフェースが正しいかどうかをチェックします。これは、より早い段階での障害検出を実現したもので、翻訳時にチェックできるものは翻訳時に、実行時でしかチェックできないものは実行時に行います。このチェックの際に適合と呼ばれる概念が適用されます。

適合とはクラス間の関係であり、あるクラス(A)のインタフェースを完全に含むクラス(B)があった場合、「BはAに適合する」と表現します。このときのインタフェースとは、クラスで定義されたメソッド(暗黙定義も含む)とそのメソッドのパラメタを指します。つまり、継承により親子関係にあるクラスでは、適合関係は成立(子は親に適合)します。

図解すると以下のとおりです。



上図の場合、管理職クラスは従業員クラスの持つ全機能を包含しています。このとき、「管理職クラスは従業員クラスに適合している」と表現します。

逆に、従業員クラスは管理職クラスの持つ全機能を包含していません。したがって、「従業員クラスは管理職クラスに適合していない」となります。つまり、適合の関係は相互に成立するものではなく、単一方向に成立するものといえます。

この適合関係は、オブジェクト参照の代入時や、オブジェクト参照項目を使用したメソッド呼出し時などに意味を持ってきます。たとえば、代入(SET文)の場合、適合関係が成立するクラスのオブジェクト参照項目間の代入(管理職クラスのオブジェクト参照項目を従業員クラスのオブジェクト参照項目へ)はできますが、適合関係が成立しない場合、転記はできません(翻訳エラーとなります)。このように、適合関係をチェックすることを適合チェックと呼んでいます。

```

:
WORKING-STORAGE SECTION.
01 OBJ-1 USAGE OBJECT REFERENCE EMPLOYEE.
01 OBJ-2 USAGE OBJECT REFERENCE MANAGER.
:
PROCEDURE DIVISION.
:
 INVOKE EMPLOYEE "NEW" RETURNING OBJ-1.
 SET OBJ-2 TO OBJ-1. ... [1]
:
 INVOKE MANAGER "NEW" RETURNING OBJ-2.
 SET OBJ-1 TO OBJ-2. ... [2]
:

```

[1] 従業員クラスから管理職クラスへの適合関係は成立しないため、翻訳エラーになります。

[2] 管理職クラスから従業員クラスへの適合関係が成立するため、問題なくオブジェクト参照が転記されます。

メソッド呼出しの適合チェックの場合、メソッドのインタフェースに対してチェックが行われます。たとえば、INVOKE文に指定されたメソッドがクラス中に存在しない場合や、メソッドに渡すパラメタが異なる場合などにチェックされます。

```

:
WORKING-STORAGE SECTION.
01 OBJ-1 USAGE OBJECT REFERENCE EMPLOYEE.
01 OBJ-2 USAGE OBJECT REFERENCE MANAGER.

```

```

:
PROCEDURE DIVISION.
:
 INVOKE EMPLOYEE "NEW" RETURNING OBJ-1.
 INVOKE OBJ-1 "CMPT-MPAY" RETURNING 特別手当. ... [1]
:
 INVOKE MANAGER "NEW" RETURNING OBJ-2.
 INVOKE OBJ-2 "CMPT-MPAY" USING 特別手当. ... [2]
:

```

- [1] 存在しないメソッドが指定されたためにエラーとなります。
- [2] CMPT-MPAYメソッドへのパラメタが異なるためにエラーとなります。

では、なぜこのような適合という概念があるのか、従業員クラスと管理職クラスの関係为例にして説明しておきます。

管理職クラスは、従業員クラスの持つ全機能を包含しているので、従業員クラスと同じように動作することができます。つまり、従業員クラスのインタフェースを用いて管理職クラスのオブジェクトを操作することが可能です。これに対して、従業員クラスは管理職クラスの全機能を包含していないため、管理職クラスのインタフェースを用いて従業員クラスのオブジェクトを操作することはできません。このような関係を表現するために、オブジェクト指向では適合という言葉が定義されたのです。

## 16.4.2 オブジェクト参照項目と適合チェック

オブジェクト参照項目の定義には、いくつかの種類があり、それぞれ格納されるオブジェクト参照データや、適合チェックのされ方が異なります。

オブジェクト参照項目は、大きく分けると、以下の3種類があり、多態や動的束縛(詳細は、“[16.6.2 メソッドの動的束縛と多態](#)”を参照してください)を実現する際に使い分けます。

```

:
01 OBJ-1 USAGE OBJECT REFERENCE. ... [1]
01 OBJ-2 USAGE OBJECT REFERENCE EMPLOYEE. ... [2]
01 OBJ-3 USAGE OBJECT REFERENCE EMPLOYEE ONLY. ... [3]
:

```

[1] どのクラスのオブジェクト参照も格納することができる定義です。この場合、翻訳時の適合チェックは行われないため、コーディング(目的プログラムができるまで)は容易ですが、実行時の適合チェックによって、手戻りの発生する可能性が大きくなります。

[2] これまでの例でも用いられた定義で、指定されたクラス(例では従業員クラス)のオブジェクト参照が格納されることを明示指定する定義です。この場合、従業員クラスまたは従業員クラスの子クラス(管理職クラス)のオブジェクト参照を格納することができます。

[3] 指定されたクラスのオブジェクト参照だけを格納する定義です。この場合、指定されたクラスに適合するクラスのオブジェクト参照を格納することはできなくなります。

また、[2]および[3]については、ファクトリオブジェクトまたはオブジェクトインスタンスによって指定が異なります。例では、オブジェクトインスタンスのオブジェクト参照を格納する指定で、ファクトリオブジェクトの場合は、以下のとおり定義します。

```

:
01 OBJ-2 USAGE OBJECT REFERENCE FACTORY OF EMPLOYEE.
01 OBJ-3 USAGE OBJECT REFERENCE FACTORY OF EMPLOYEE ONLY.
:

```

なお、ファクトリオブジェクトのオブジェクト参照項目については、これまで説明していませんでしたが、使い方はオブジェクトインスタンスの場合と同じです(下図を参照してください)。

```

:
WORKING-STORAGE SECTION.
01 OBJ-F USAGE OBJECT REFERENCE FACTORY OF EMPLOYEE.
01 OBJ-1 USAGE OBJECT REFERENCE EMPLOYEE.
:
PROCEDURE DIVISION.
:
 SET OBJ-F TO EMPLOYEE.

```

```
INVOKE OBJ-F "NEW" RETURNING OBJ-1.
:
```

### 16.4.3 翻訳時の適合チェックと実行時の適合チェック

適合チェックには、翻訳時に行われるものと、実行時に行われるものがあります。

ここでは、適合チェックのタイミングについて説明します。

#### 16.4.3.1 代入時の適合チェック

ここでは、代入時の適合チェックについて説明します。

```
:
WORKING-STORAGE SECTION.
01 OBJ-1 USAGE OBJECT REFERENCE.
01 OBJ-2 USAGE OBJECT REFERENCE EMPLOYEE.
01 OBJ-3 USAGE OBJECT REFERENCE EMPLOYEE ONLY.
:
01 OBJ-X USAGE OBJECT REFERENCE.
01 OBJ-Y USAGE OBJECT REFERENCE MANAGER.
01 OBJ-Z USAGE OBJECT REFERENCE MANAGER ONLY.
:
PROCEDURE DIVISION.
:
 SET OBJ-1 TO OBJ-X. ... [1]
 SET OBJ-1 TO OBJ-Y. ... [2]
 SET OBJ-1 TO OBJ-Z. ... [3]
 :
 SET OBJ-2 TO OBJ-X. ... [4]
 SET OBJ-2 TO OBJ-Y. ... [5]
 SET OBJ-2 TO OBJ-Z. ... [6]
 :
 SET OBJ-3 TO OBJ-X. ... [7]
 SET OBJ-3 TO OBJ-Y. ... [8]
 SET OBJ-3 TO OBJ-Z. ... [9]
 :
```

OBJ-1は、どのようなクラスのオブジェクト参照も格納可能のため、代入(SET文)時に適合チェックはされません。したがって、[1]、[2]、[3]は適合エラーにはなりません。

OBJ-2は、従業員クラスおよび従業員クラスの子クラスのオブジェクト参照が格納可能のため、[4]は適合エラー(翻訳時)となりますが、[5]、[6]は適合エラーにはなりません。

OBJ-3は、従業員クラスのオブジェクト参照だけ格納可能のため、[7]、[8]、[9]はどれも適合エラーとなります。適合チェックは翻訳時に行われます。

#### 16.4.3.2 メソッド呼出し時の適合チェック

ここでは、メソッド呼出し時の適合チェックについて説明します。

```
:
WORKING-STORAGE SECTION.
01 OBJ-1 USAGE OBJECT REFERENCE.
01 OBJ-2 USAGE OBJECT REFERENCE EMPLOYEE.
01 OBJ-3 USAGE OBJECT REFERENCE EMPLOYEE ONLY.
:
PROCEDURE DIVISION.
:
 INVOKE OBJ-1 "CMPT-SAL" RETURNING SALARY.
 INVOKE OBJ-2 "CMPT-SAL" RETURNING SALARY.
 INVOKE OBJ-3 "CMPT-SAL" RETURNING SALARY.
 :
```

OBJ-1には、どのようなクラスのオブジェクト参照も格納可能なため、呼び出すメソッドの特定は実行時までできません。したがって、誤ったパラメータやメソッド名が指定されたかどうかの適合チェックは実行時に行われます。上図の場合、実行時、OBJ-1に従業員クラスまたは従業員クラスの子クラスのオブジェクト参照以外が設定されていた場合、メソッドが見つからない旨のエラーが出力されます。

OBJ-2には、従業員クラスと従業員クラスの子クラスのオブジェクト参照だけ格納可能です。メソッド名やパラメータの情報は翻訳時にわかるため(後述のリポジトリ情報を利用)、適合チェックは翻訳時に行われます。

OBJ-3には、従業員クラスのオブジェクト参照だけ格納可能です。したがって、翻訳時に適合チェックが行われます。

このように、オブジェクト参照項目にクラス名を指定することによって、翻訳時の適合チェックが可能になります。これにより、メソッド呼出し時のパラメータ不整合による障害が翻訳時に取り除けるというメリットを得ることができます。

## 16.5 リポジトリ

クラス定義を翻訳すると、目的プログラムと同時にリポジトリファイル(クラス名.REP)と呼ばれる資源が生成されます。

リポジトリファイルは、そのクラスを利用するプログラムまたはクラスの翻訳時にコンパイラへの入力となるファイルで、適合チェックなどに利用されます。

ここでは、リポジトリファイルの概要について説明します。

なお、詳細については、“[17.4.3.1 リポジトリファイル](#)”を参照してください。

### 16.5.1 リポジトリファイルの概要

リポジトリファイルは、クラス定義を翻訳することによって生成される、クラス情報を格納したファイルです。翻訳が正常に終了した場合は、必ず出力されます。

リポジトリファイル中には、そのクラスに関する情報が格納されていますが、テキスト形式ではないため、利用者が直接ファイルを参照することはできません。リポジトリ内の情報を参照する場合には、クラスブラウザを利用してください。

以下に、リポジトリファイルの利用方法を示します。

- ・ クラス情報を表示するためにクラスブラウザへ入力する。
- ・ 継承を実現するため、コンパイラへ入力する。
- ・ 適合チェックを行うため、コンパイラへ入力する。

以下にそれぞれについて説明します。

#### 16.5.1.1 クラスブラウザへの入力

オブジェクト指向では、クラスに関する情報を表示する手段として、クラスブラウザを提供しています。このクラスブラウザへリポジトリファイルを入力することにより、ファイルに格納されているクラス情報を表示することができます。クラスブラウザの詳細については、“[17.3.1 クラスブラウザ](#)”を参照してください。

#### 16.5.1.2 継承の実現

継承は、親クラスのリポジトリファイルを入力することによって実現されます。

たとえば、従業員クラスを継承して管理職クラスを作成する場合、管理職クラスの翻訳時に従業員クラスのリポジトリファイルを入力しなければなりません。

管理職クラス

```
IDENTIFICATION DIVISION.
CLASS-ID. MANAGER INHERITS EMPLOYEE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS EMPLOYEE.
:
```



## 注意

親クラス(INHERITS句に指定したクラス)は、リポジトリ段落に指定しなければなりません。



## 参考

2階層以上の継承の場合、直接の親クラスのリポジトリファイルを入力するだけで翻訳できます。つまり、管理職クラスを翻訳する場合、FJBASEクラスも間接的に継承していることとなりますが、翻訳時に、FJBASEクラスのリポジトリファイルを入力する必要はありません。

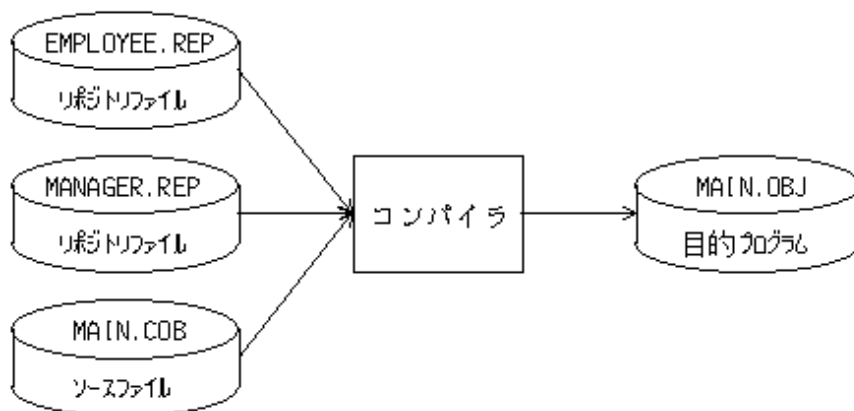
### 16.5.1.3 適合チェックの実現

適合チェックは呼び出すクラスのリポジトリファイルを入力することによって実現されます。

たとえば、従業員管理プログラムで従業員クラスと管理職クラスを利用する場合、従業員管理プログラムの翻訳時にこれらのクラスのリポジトリファイルを入力する必要があります。

#### 従業員管理プログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS EMPLOYEE
 CLASS MANAGER.
:
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ-1 USAGE OBJECT REFERENCE EMPLOYEE.
01 OBJ-2 USAGE OBJECT REFERENCE MANAGER.
:
PROCEDURE DIVISION.
:
 INVOKE EMPLOYEE "NEW" RETURNING OBJ-1.
 INVOKE OBJ-1 "SET-EMPL" USING 従業員データ.
:
 INVOKE MANAGER "NEW" RETURNING OBJ-2.
 INVOKE OBJ-2 "SET-EMPL" USING 従業員データ.
:
```

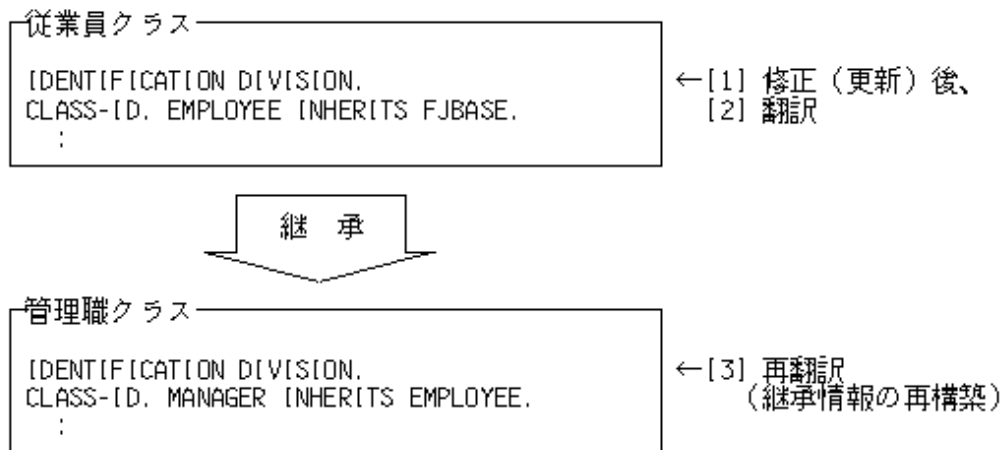




## 16.5.2 リポジトリファイル更新の影響

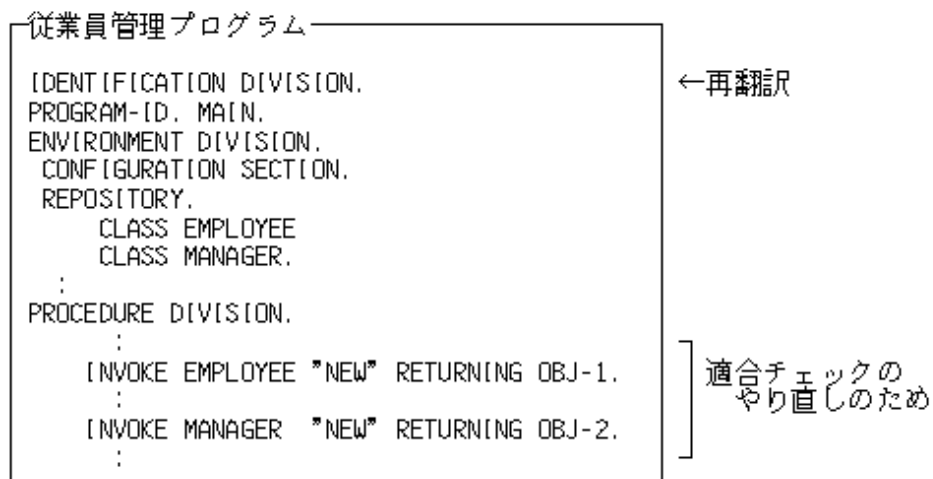
目的プログラムの生成後、親クラスや呼び出すクラスに修正が入った場合、どのように対応すればよいのでしょうか。

コンパイラは、リポジトリファイルに格納されている情報だけから、継承や適合チェックを実現しています。そのため、リポジトリファイルが更新された場合、継承情報の再構築や適合チェックのやり直しを行う必要があります。つまり、親クラスのインタフェースに修正が入れた場合、子クラスは、何も修正がなくても再翻訳を行わなければなりません。ただし、インタフェースに何も変更が生じない修正の場合は再翻訳する必要はありません。



上図のとおり、修正したクラス(従業員クラス)の子クラス(管理職クラス)は、何も修正していなくても再翻訳が必要になります(継承情報の再構築のため)。

また、修正したクラスを呼び出しているプログラムやクラスについても再翻訳が必要です(適合チェックのやり直しのため)。



これらの再翻訳は、利用者が行わなければなりません。そのため、一度構築したクラス定義を修正する場合は、十分注意してください。なお、プロジェクト管理機能を利用すれば、依存関係を最初に登録しておくだけで必要な再翻訳を実行するので修正が容易に行えます。プロジェクト管理機能については、“[第17章 オブジェクト指向プログラミングの開発](#)”を参照してください。

## 16.6 メソッドの束縛

メソッドを呼び出す場合、呼び出すメソッドは、下記の2つの情報によって決定されます。

- どのオブジェクト上のメソッドなのか?
- 何という名前のメソッドなのか?

この「呼び出すメソッドを決定する」ことを、オブジェクト指向では「メソッドの束縛」と呼びます。

ここでは、メソッドの束縛について説明します。

## 16.6.1 メソッドの静的束縛

静的束縛とは、呼び出すメソッドが翻訳時に決定できることを意味します。これは、呼出し方法(INVOKE文の記述形式)によって自動的に決定されるため、利用者が明示する必要はありません。

以下の場合、静的束縛になります。

```

:
WORKING-STORAGE SECTION.
01 OBJ-1 USAGE OBJECT REFERENCE EMPLOYEE ONLY.
:
PROCEDURE DIVISION.
:
 INVOKE EMPLOYEE "NEW" RETURNING OBJ-1. ... [1]
:
 INVOKE OBJ-1 "CMPT-SAL" RETURNING SALARY. ... [2]
:
 INVOKE SUPER "CMPT-SAL" RETURNING SALARY. ... [3]
:

```

[1] クラス名を指定してファクトリメソッドを呼び出す場合です。

[2] ONLY指定が記述されたオブジェクト参照項目を指定してメソッドを呼び出す場合です。

[3] 定義済みオブジェクト参照一意名SUPERを指定してメソッドを呼び出す場合です。なお、定義済みオブジェクト参照一意名SUPERは親クラスを表します。詳細については、“[16.6.3 定義済みオブジェクト一意名SUPER](#)”を参照してください。

## 16.6.2 メソッドの動的束縛と多態

呼び出すメソッドが翻訳時に決定できない場合、つまり、呼び出すメソッドを実行時に決定することを動的束縛と呼びます。

これは、以下のとおり、実行時にオブジェクト参照項目に格納されたオブジェクト参照の値によりメソッドを特定しなければならない場合にとられます。

```

:
WORKING-STORAGE SECTION.
01 OBJ-1 USAGE OBJECT REFERENCE.
01 OBJ-2 USAGE OBJECT REFERENCE EMPLOYEE.
:
PROCEDURE DIVISION.
:
 INVOKE OBJ-1 "CMPT-SAL" RETURNING SALARY. ... [1]
:
 INVOKE OBJ-2 "CMPT-SAL" RETURNING SALARY. ... [2]
:
 INVOKE SELF "CMPT-SAL" RETURNING SALARY. ... [3]
:

```

[1] どのクラスのメソッドを呼び出せばよいのか実行時までわからないため、動的束縛になります。

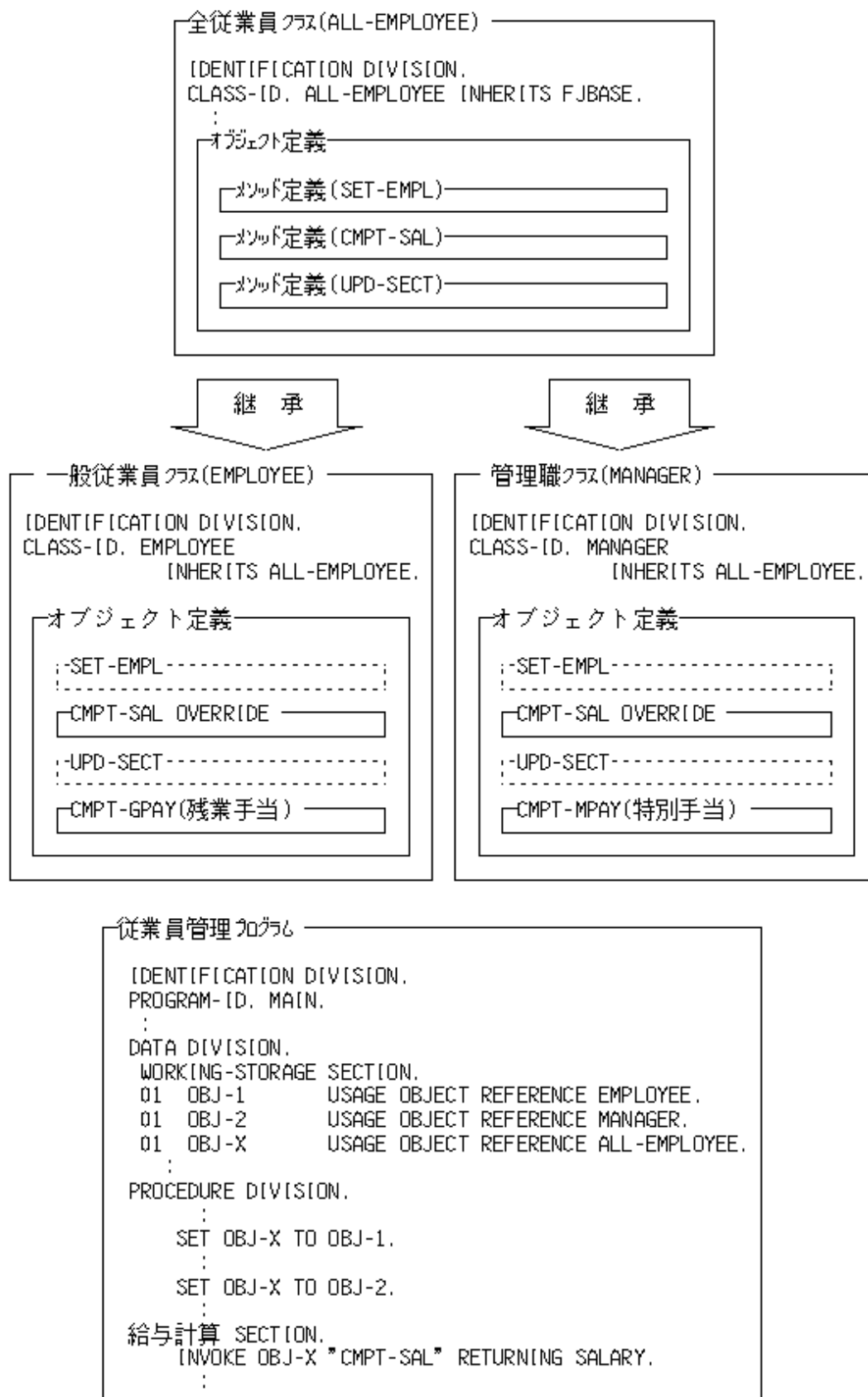
[2] OBJ-2には、従業員クラスに適合するクラスのオブジェクト参照が格納できるため、[1]の場合と同様に動的束縛になります。

[3] SELFは、実行中のオブジェクトを表現します。つまり、これも動的束縛になります。詳細については、“[16.6.4 定義済みオブジェクト一意名SELF](#)”を参照してください。

この動的束縛を利用して「多態」と呼ばれる機能を実現することができます。

多態とは、適合関係を利用して、実際のオブジェクトを意識しないで多種のオブジェクトを処理する方法で、共通のインターフェースを持つオブジェクトの処理時に利用することができます。

これまで、管理職クラスは従業員クラスの子クラスに位置付けられていましたが、実際には、一般従業員に固有な処理も必要になります(たとえば、給与計算処理での残業手当の加算など)。そのため、抽象化クラスとして、管理職を含めた全従業員対象のクラス(**ALL-EMPLOYEE**クラス)を作成します。この抽象化クラスの定義によって、多態を利用した共通処理が実現できます。



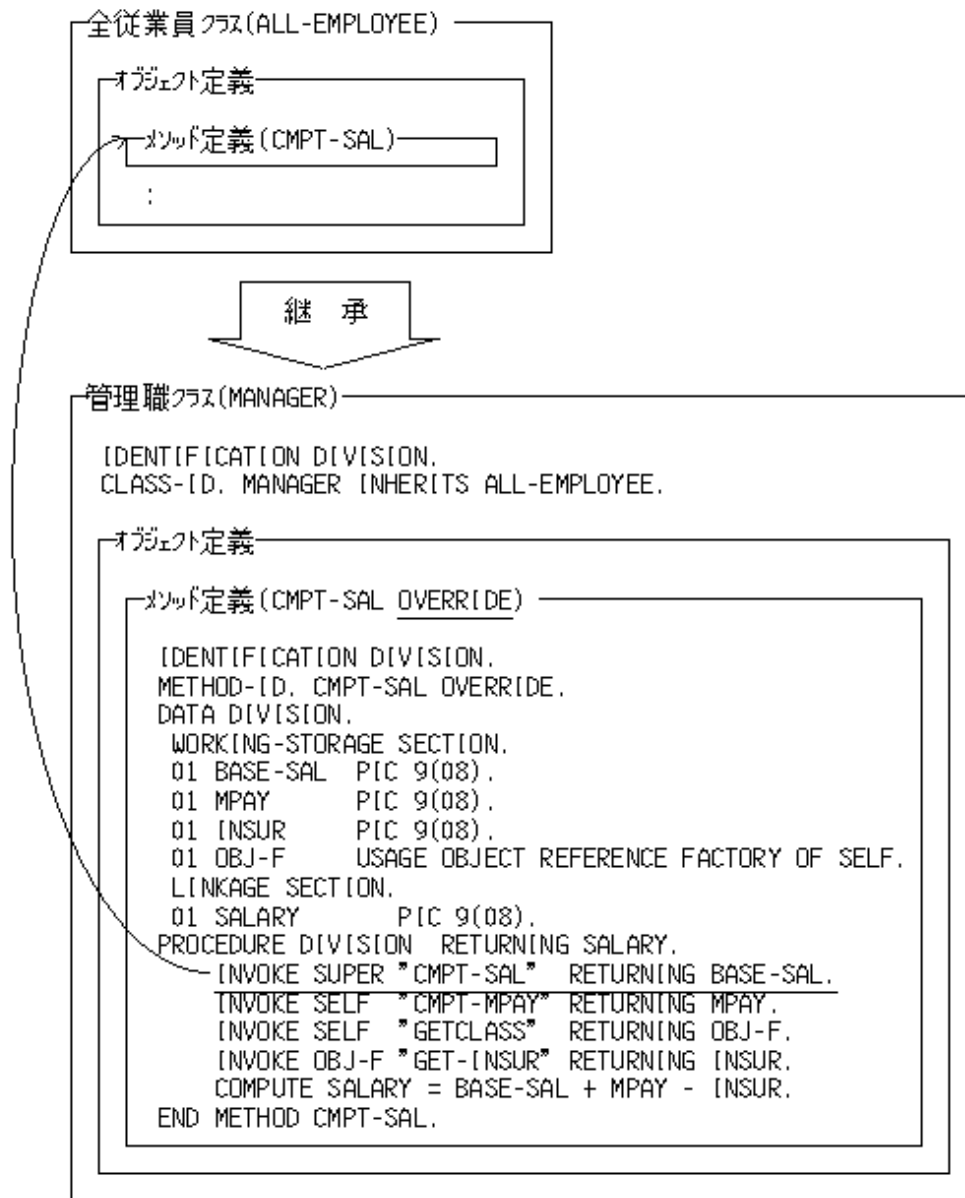
上図のようにオブジェクトを意識せず(一般従業員なのか、管理職なのか)に給与計算処理を行うことができます。つまり、1つのオブジェクト参照項目によって、複数の別定義オブジェクトを操作することができたことになります。これを多態と呼びます。

### 16.6.3 定義済みオブジェクト一意名SUPER

オブジェクト指向では、親クラスを表現するために、あらかじめ定義済みオブジェクト一意名SUPERが用意されています。

この定義済みオブジェクト一意名SUPERの使用方法について、全従業員クラスと管理職クラスの関係を利用して説明します。

管理職クラスは、特別手当の加算があるために給料計算メソッド(CMPT-SAL)を上書きしています。しかし、特別手当以外の処理は、継承元(全従業員クラス)の処理と同じだったとします。このような場合、上書きしたメソッドから親クラスで定義しているメソッドを呼び出すことができます。



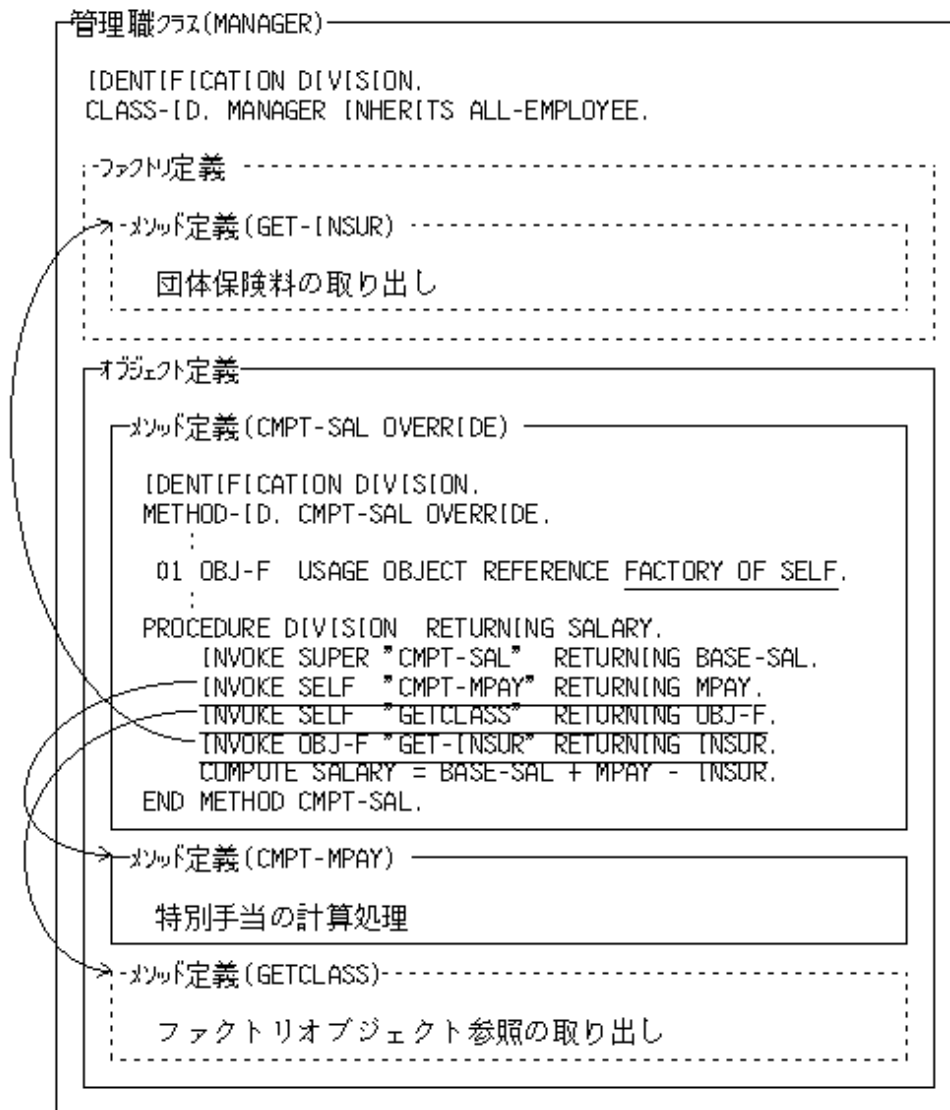
このように、親クラスを表現する場合に、定義済みオブジェクト一意名SUPERが利用されます。

### 16.6.4 定義済みオブジェクト一意名SELF

オブジェクト指向では、自オブジェクト参照(現在実行中のオブジェクト参照)を表すために、定義済みオブジェクト一意名SELFが用意されています。

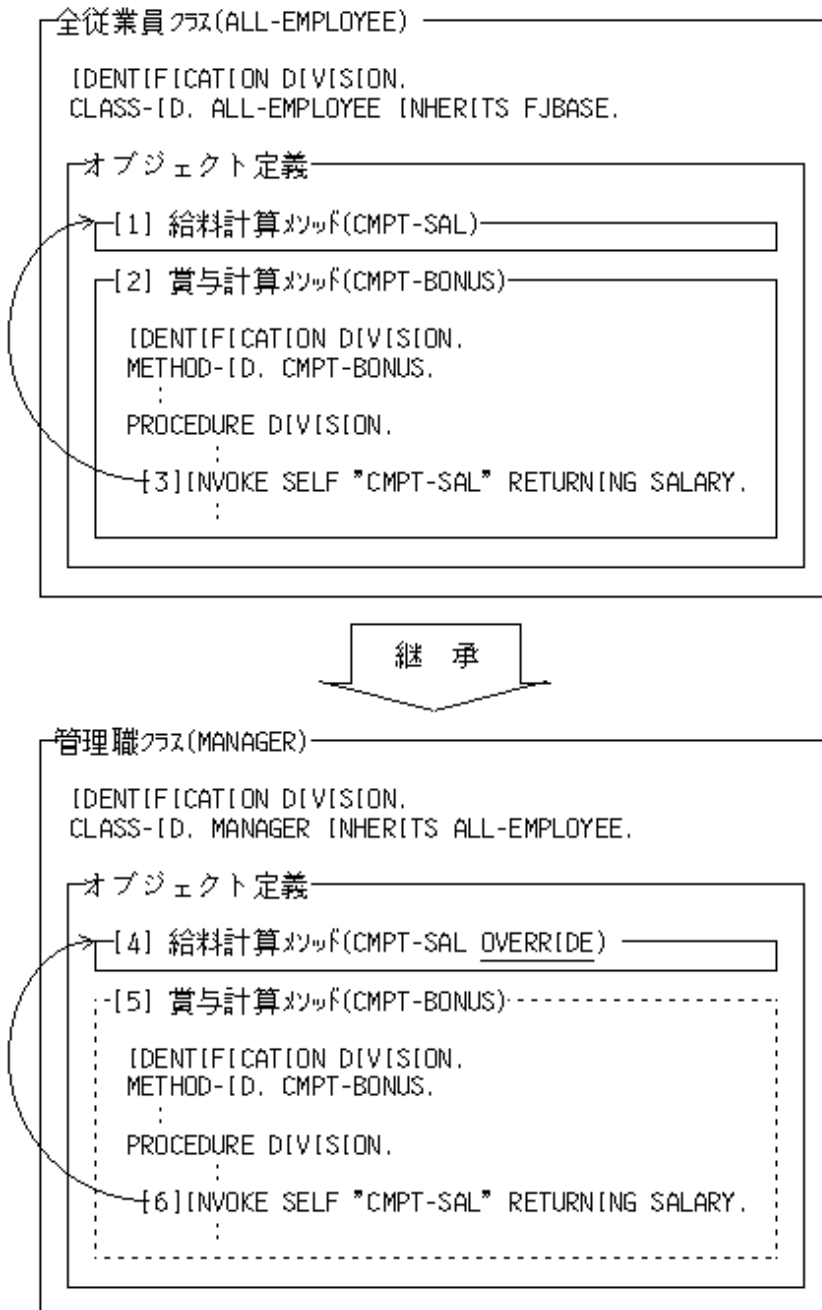
この定義済みオブジェクト一意名SELFの使用方法について説明します。

管理職クラスで給料計算(CMPT-SAL)を行なう場合、特別手当を求めるために特別手当の計算メソッド(CMPT-MPAY)を呼び出す、というときに定義済オブジェクト一意名SELFを利用します。



定義済みオブジェクト一意名SELFの使用時に注意しなければならないのは、呼び出すメソッドは、つねに実行時に決定する(動的束縛)ということです。つまり、上書きされたメソッドが存在する場合、実行中のオブジェクトによって呼び出されるメソッドが変わります。

たとえば、前述の例で全従業員クラス中に賞与(ボーナス)を計算するメソッドがあったとして、そのメソッド中で給料を求める処理が必要だった場合、定義済みオブジェクト一意名SELFを使用して以下のとおり記述することができます。



従業員クラスのオブジェクト参照によって賞与計算メソッド([2])が呼び出された場合、[3]のINVOKE文によって[1]の給料計算メソッドが呼び出されます。管理職クラスのオブジェクト参照によって賞与計算メソッド([5]暗黙定義メソッド)が呼び出された場合、[6]のINVOKE文によって[4]の給料計算メソッド(上書きメソッド)が呼び出されます。つまり、それぞれのクラスに適した給料計算ができることになります。

これも多態の1つの形態です。

## 16.7 少し進んだ使い方

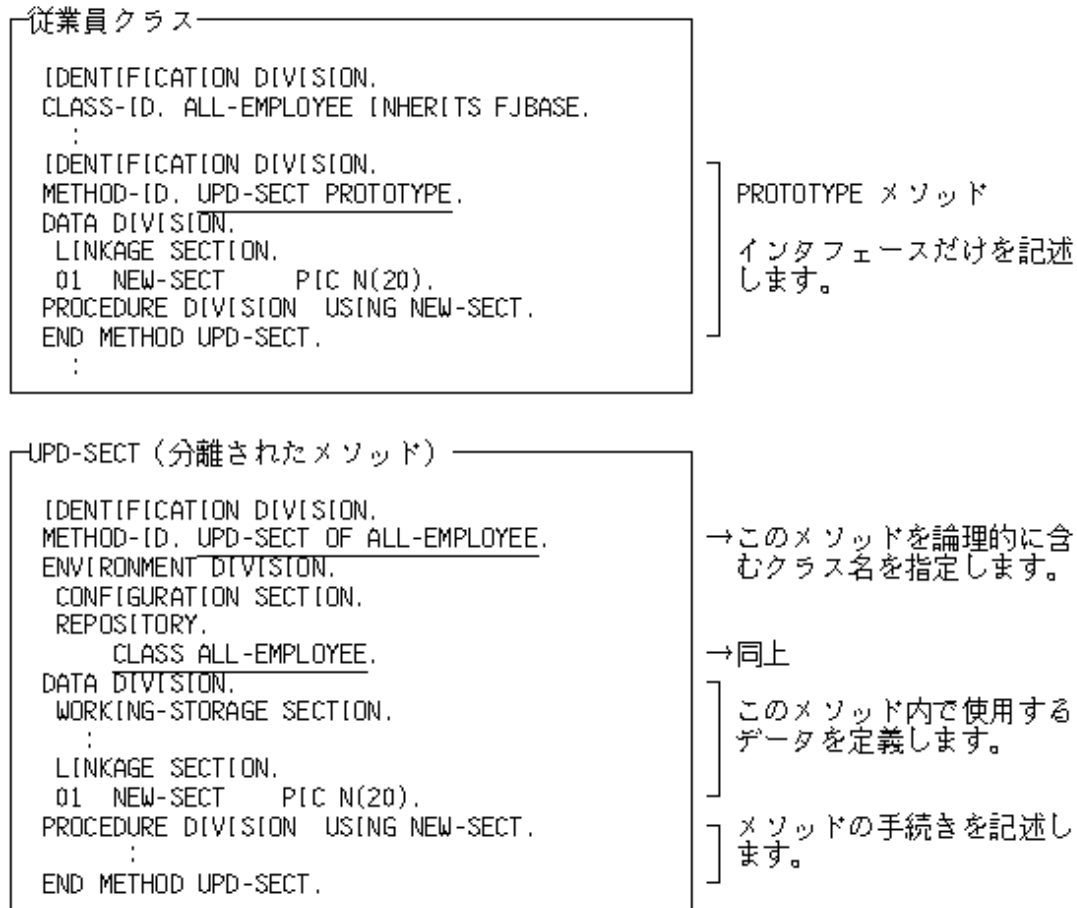
これまでの説明で、オブジェクト指向プログラミングの基本的な概念や機能の説明は、すべて完了しました。つまり、ここまでに出てきた機能だけを利用してオブジェクト指向を実現することが可能です。しかし、これら基本機能をより利用(記述)しやすくする機能や、一歩進めた機能などを他にも多く提供しています。

ここでは、それらの少し進んだ使い方について説明します。

## 16.7.1 メソッドのPROTOTYPE宣言

通常、クラス定義内に記述するメソッド定義を、物理的に別ファイルに定義することができます。

このとき、クラス定義内には、メソッド名とそのインタフェースだけを定義し、メソッドデータや手続きは別翻訳単位内で定義します。クラス定義内に記述されたメソッドを「PROTOTYPEメソッド」と呼び、別翻訳単位で定義したメソッドを「分離されたメソッド」と呼びます。



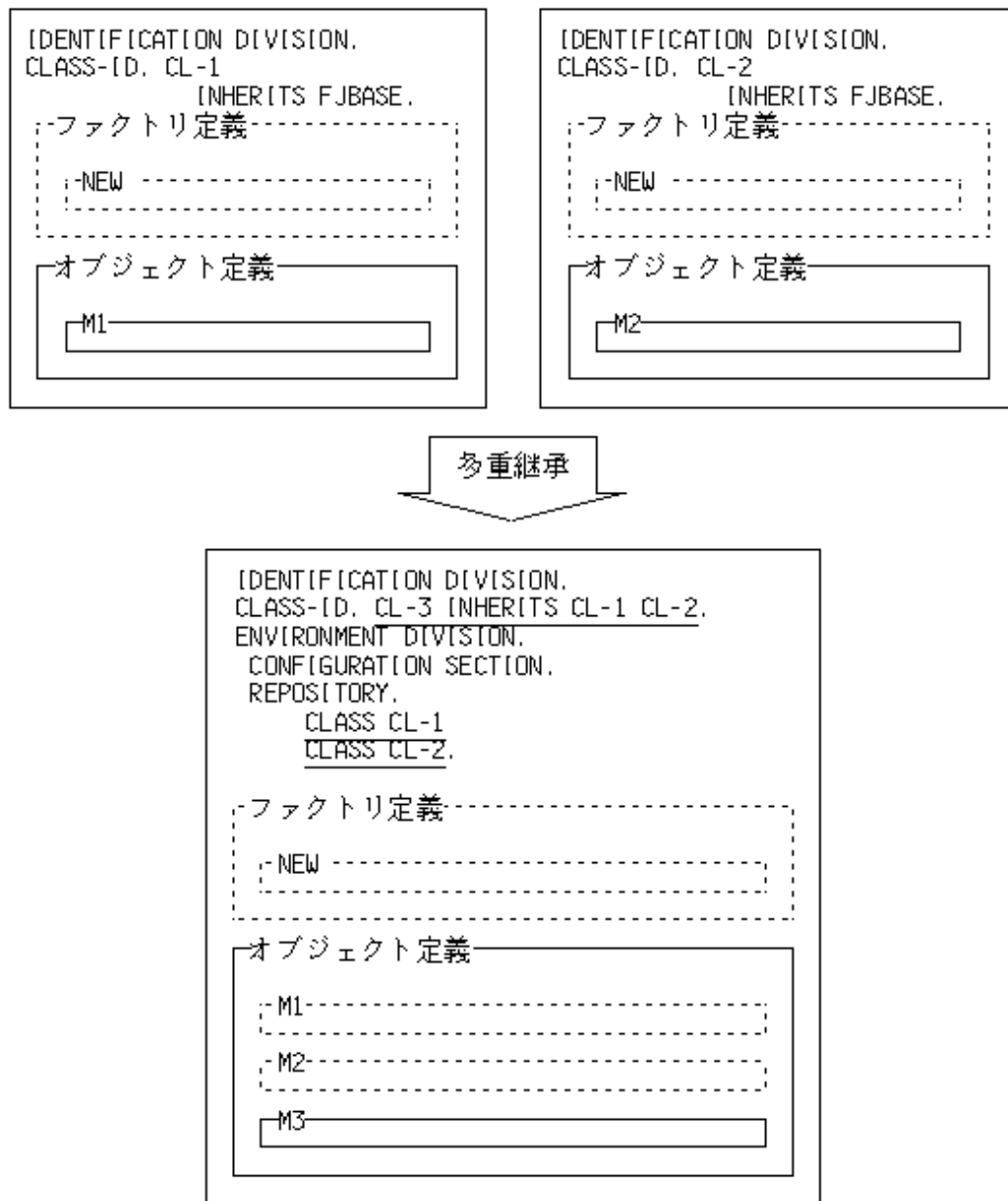
このようにメソッド定義を別翻訳単位にすることによって、1つのクラス定義を複数人で開発できるなどのメリットがあります。

なお、分離されたメソッドの翻訳時に、そのメソッドを論理的に含むクラスのリポジトリファイルを入力する必要があるため、メソッドを翻訳するためには、先にクラス定義を翻訳しておく必要があります。詳細については、“[第17章 オブジェクト指向プログラミングの開発](#)”を参照してください。

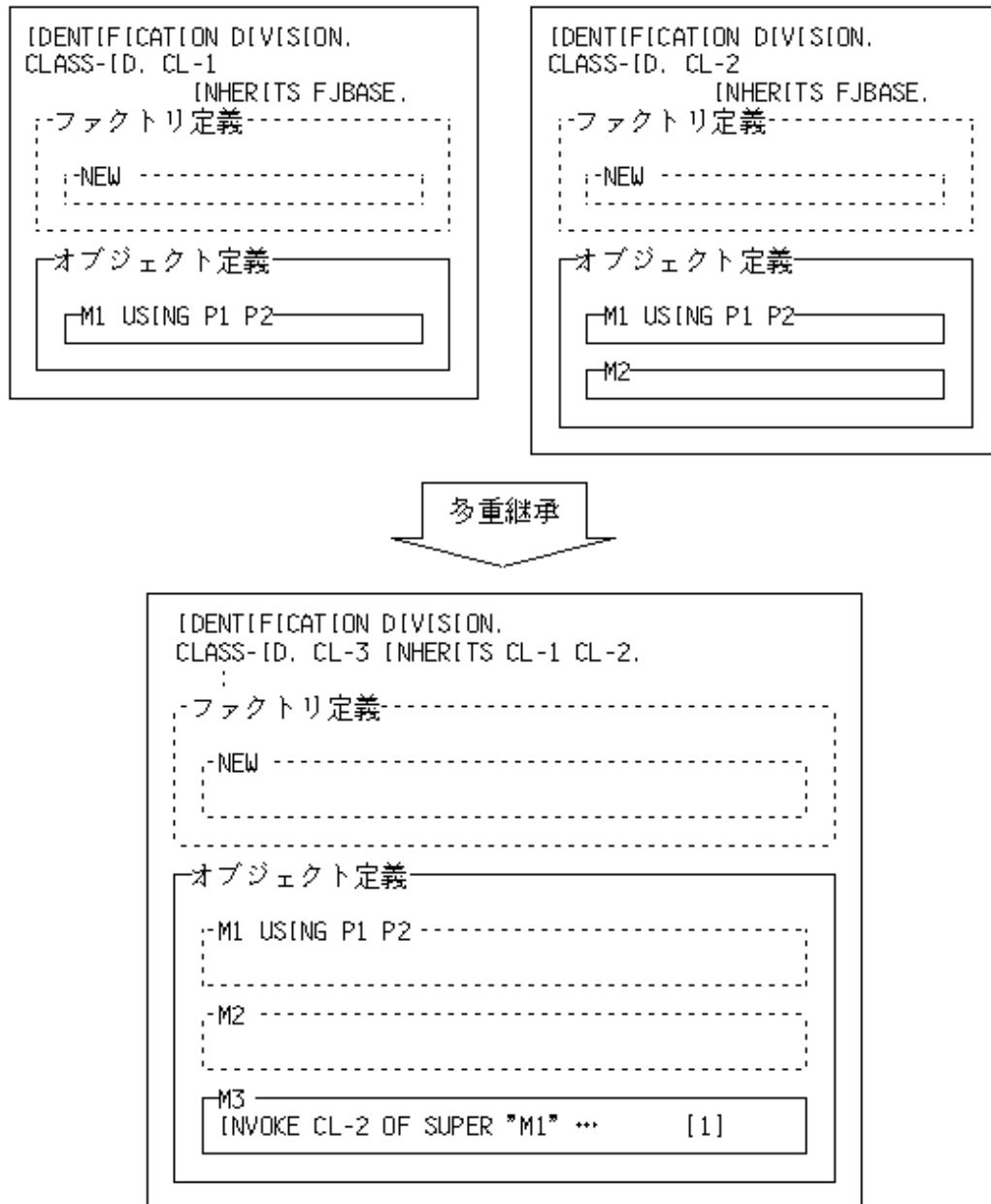
## 16.7.2 多重継承

複数のクラスを同時に継承することも可能です。これを多重継承と呼びます。





継承の論理については、1つのクラスを継承する場合と同じです。ただし、「同名のメソッドが複数の親クラスで定義されていた場合、それらのメソッドのインタフェースは同じでなければならない」という規則があります(下図を参照してください)。



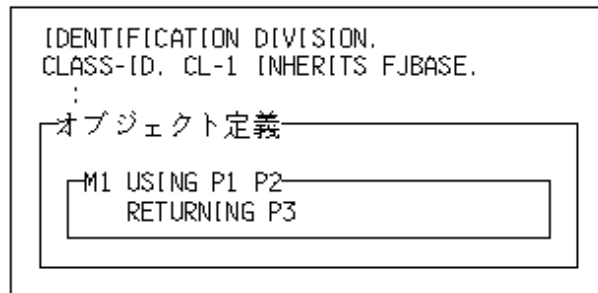
このように複数の親クラスで同名のメソッドが定義されていた場合、利用者が明にこのメソッドを上書きしない限り、引き継がれます。引き継がれるメソッドは、INHERITS句に指定されたクラス名の並びを左から検索して、最初にそのメソッドが見つかるクラスのメソッドになります。

上図の例では、複数のクラスCL-1、CL-2からメソッドM1がクラスCL-3に引き継がれています。このM1はCL-3のINHERITS句に指定されたクラス名を左から順に探した結果、CL-1のM1であると判断されます。このため、クラスCL-3から、クラスCL-2のM1を呼び出したい場合は、上図[1]のように定義済みオブジェクト一意名SUPERに明示的にクラス名を指定して呼び出す必要があります。

### 16.7.3 行内呼出し

通常、メソッドの呼出しにはINVOKE文を使用しますが、INVOKE文を使用しない方法(書き方)があります。これを「メソッドの行内呼出し」と呼びます。

ただし、この行内呼出しは、メソッドからの復帰値(RETURNINGに指定された項目の値)を参照する場合にだけ利用できるため、復帰項目を持たないメソッドに対しては利用することができません。



メソッドM1を呼び出した後、復帰値P3を参照する場合、INVOKE文を利用すると以下の書き方になります。

```

:
WORKING-STORAGE SECTION.
01 OBJ-1 USAGE OBJECT REFERENCE CL-1.
PROCEDURE DIVISION.
:
 INVOKE OBJ-1 "M1" USING P1 P2
 RETURNING P3.
 IF P3 = 0 THEN ...
:

```

行内呼出しを利用すると、以下のように記述できます。

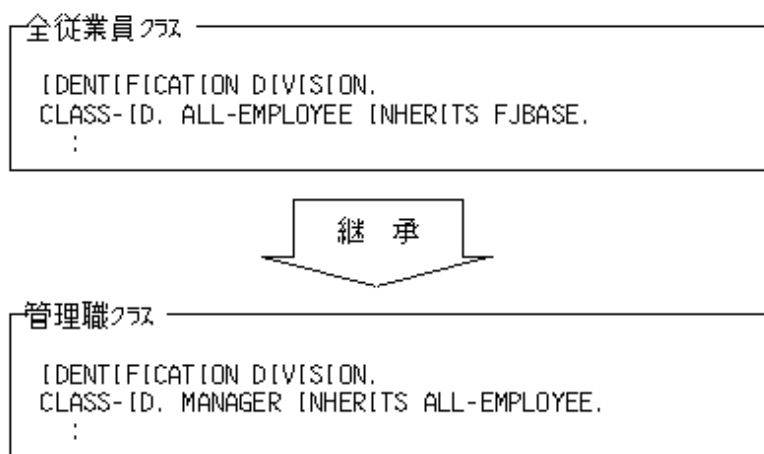
```

:
WORKING-STORAGE SECTION.
01 OBJ-1 USAGE OBJECT REFERENCE CL-1.
PROCEDURE DIVISION.
:
 IF OBJ-1 :: "M1"(P1 P2) = 0 THEN ...
:

```

## 16.7.4 オブジェクト指定子

適合の規則に違反している場合、翻訳時に実施する適合チェックでエラーとなることがあります。しかし、オブジェクト指定子を利用することで、翻訳時の適合チェックをゆるめ、適合の規則に違反している場合でも問題なく翻訳できるようになります。



たとえば、上のような継承関係があった場合

#### 従業員管理プログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ-1 USAGE OBJECT REFERENCE ALL-EMPLOYEE.
01 OBJ-2 USAGE OBJECT REFERENCE MANAGER.
PROCEDURE DIVISION.
:
 INVOKE MANAGER "NEW" RETURNING OBJ-2.
:
 SET OBJ-1 TO OBJ-2. ... [1]
 SET OBJ-2 TO OBJ-1. ... [2]
 ↵エラー
:
```

[1]では、管理職クラスは全従業員クラスの子クラスであるため、問題なく代入することができます。しかし、その後、[2]で元の項目に代入しよう(戻そう)とした場合、全従業員クラスは管理職クラスの子クラスではないことからエラーになってしまいます。つまり、格納されているデータを考えると何も問題ない代入だったとしても、クラス間の継承関係によって代入不可となってしまうのです。

このような場合、オブジェクト指定子を利用することによって代入可能になります。

#### 従業員管理プログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ-1 USAGE OBJECT REFERENCE ALL-EMPLOYEE.
01 OBJ-2 USAGE OBJECT REFERENCE MANAGER.
PROCEDURE DIVISION.
:
 INVOKE MANAGER "NEW" RETURNING OBJ-2.
:
 SET OBJ-1 TO OBJ-2.
 SET OBJ-2 TO OBJ-1 AS MANAGER.
:
```

上図のように記述すると、OBJ-1は、USAGE OBJECT REFERENCE句にMANAGERが指定されたとみなして適合チェックが行われるため、代入可能となります。

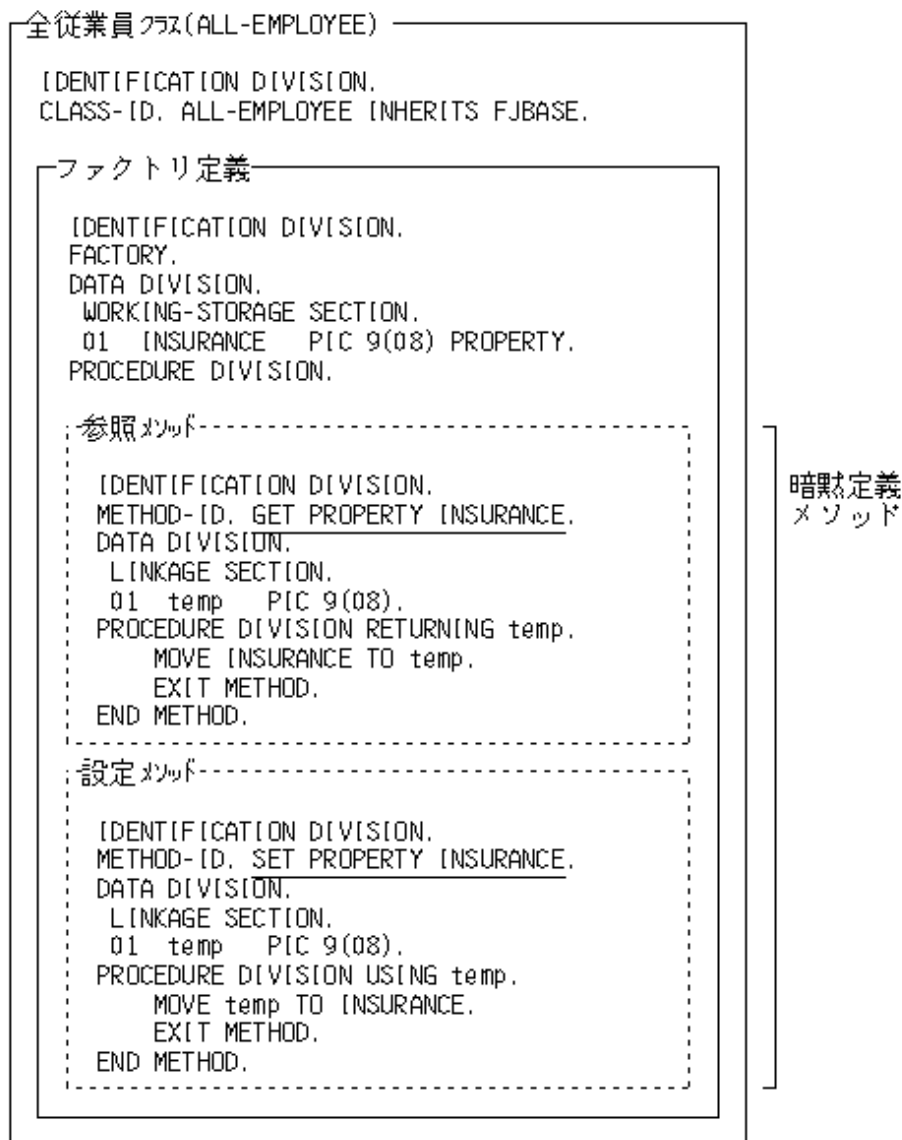
このように、オブジェクト指定子を用いた場合、翻訳時は指定されたクラス名で適合チェックが行われますが、実行時は実際に格納されているオブジェクト参照によって適合チェックが行われます。つまり、適合に違反している場合は、実行時にチェックアウトされます。

## 16.7.5 PROPERTY句

PROPERTY句を使用することによって、ファクトリデータおよびオブジェクトデータの参照および設定が容易に実現できます。

これは、PROPERTY句の指定によってデータを設定、参照するメソッドを自動生成することにより実現しています。

たとえば、全従業員クラスの例では、団体保険料(ファクトリデータ)の設定/参照メソッドを定義していました。この設定/参照メソッドと同じ役割を持つメソッドが、データ宣言にPROPERTY句を指定することで、以下のように暗黙メソッド(ソース記述はないが、論理的に存在するメソッド)として自動生成されます。



上図のようにPROPERTY句によって暗黙定義されるメソッドをプロパティメソッドと呼びます。

ただし、プロパティメソッドはINVOKE文を利用して呼び出すことはできません。以下のようにオブジェクトプロパティと呼ばれる一意参照を利用します。

# 従業員管理プログラム

```

IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
.
PROCEDURE DIVISION.
.

* 団体保険料の設定処理 *

 MOVE 保険額 TO INSURANCE OF EMPLOYEE. ←設定メソッドの呼出し
 MOVE 保険額 TO INSURANCE OF MANAGER. ←設定メソッドの呼出し
 .

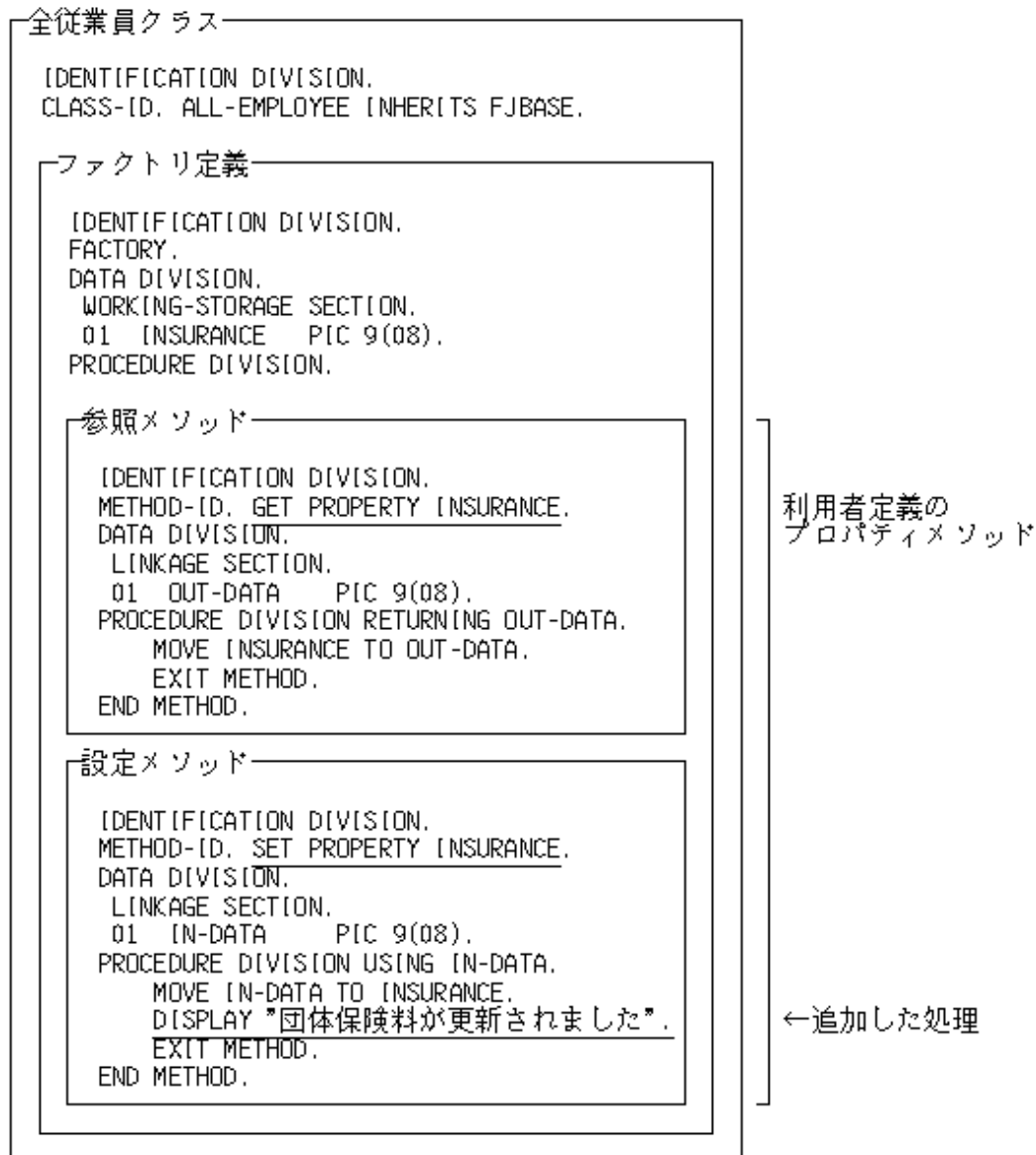
* 団体保険料の取り出し処理 *

 EVALUATE EMP-ID
 WHEN ID-EMPL
 MOVE INSURANCE OF EMPLOYEE TO 保険額 ←参照メソッドの呼出し
 WHEN ID-MAN
 MOVE INSURANCE OF MANAGER TO 保険額 ←参照メソッドの呼出し
 END-EVALUATE
 .

```

参照メソッドを呼び出すか、設定メソッドを呼び出すかは、オブジェクトプロパティが送出し側に指定されたか、受取り側に指定されたかによって決定されます。

また、プロパティメソッドにプラスアルファの処理を持たせたい場合には、利用者がプロパティメソッドを明示定義することもできます。この場合、データにPROPERTY句を指定する必要はありません。



このとき、プロパティ名と同名のデータ名(INSURANCE)にPROPERTY句は指定できないため、設定および参照の両メソッドが必要な場合、両方を明示定義しなければなりません。

## 16.7.6 初期化処理メソッドと終了処理メソッド

オブジェクトインスタンスの生成は、“NEW”メソッドを呼ぶことで行われますが、オブジェクトインスタンスの削除は、COBOLシステムがオブジェクトインスタンスの寿命が過ぎたことを自動的に判断して行います。[参照]“[16.2.2 オブジェクトの寿命](#)”

FJBASEクラスでは、オブジェクトインスタンスを生成した直後に呼び出すメソッドとオブジェクトインスタンスが削除される直前に呼び出すメソッドをオブジェクトメソッドとして用意しています。前者をINITメソッドといい、VALUE句ではできないような初期化処理が必要な場合に使用します。また、後者をFINALIZEメソッドといい、オブジェクトインスタンスが削除される時にやりたい終了処理があるときに使用します。これらのメソッドは、利用者が直接INVOKE文で呼び出す必要はなく、当該メソッドを上書き(上書きについては、“[16.3.3 メソッドの上書き](#)”を参照)して処理を書いておくことによって、呼ばれるようになります。上書きをしていない場合でも、FJBASEクラスのメソッドが呼び出されますが、実際の処理は何もしません。

#### プログラム定義

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
CLASS [-F-SAMPLE.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJREF USAGE OBJECT REFERENCE [-F-SAMPLE.
PROCEDURE DIVISION.
 INVOKE [-F-SAMPLE "NEW" RETURNING OBJREF.
 INVOKE OBJREF "XXX".
 SET OBJREF TO NULL.
END PROGRAM SAMPLE.
```

→ OBJECT INSTANCE IS GENERATED を表示  
→ XXX IS INVOKED を表示  
→ OBJECT INSTANCE IS TERMINATED を表示

#### クラス定義

```
IDENTIFICATION DIVISION.
CLASS-ID. [-F-SAMPLE INHERITS FJBASE.
```

##### オブジェクト定義

```
IDENTIFICATION DIVISION.
OBJECT.
```

##### メソッド定義

```
IDENTIFICATION DIVISION.
METHOD-ID. INIT OVERRIDE.
DATA DIVISION.
PROCEDURE DIVISION.
 DISPLAY "OBJECT INSTANCE IS GENERATED".
END METHOD INIT.
```

##### メソッド定義

```
IDENTIFICATION DIVISION.
METHOD-ID. _FINALIZE OVERRIDE.
DATA DIVISION.
PROCEDURE DIVISION.
 DISPLAY "OBJECT INSTANCE IS TERMINATED".
END METHOD _FINALIZE.
```

##### メソッド定義

```
IDENTIFICATION DIVISION.
METHOD-ID. XXX.
DATA DIVISION.
PROCEDURE DIVISION.
 DISPLAY "XXX IS INVOKED".
END METHOD XXX.
```

```
END OBJECT.
```





## 注意

終了処理メソッド `_FINALIZE` は、COBOL97 V40L20よりFJBASEに新しく追加された機能です。このため、以前のバージョンで作成したプログラムからこの機能を使用する場合、以下の処理が必要です。

- ・ `_FINALIZE` を使用するクラスおよびその継承元となるすべてのクラスを最新版のコンパイラで再翻訳する。
- ・ 再翻訳したクラスを参照するプログラム/クラスを最新版のコンパイラで再翻訳する。

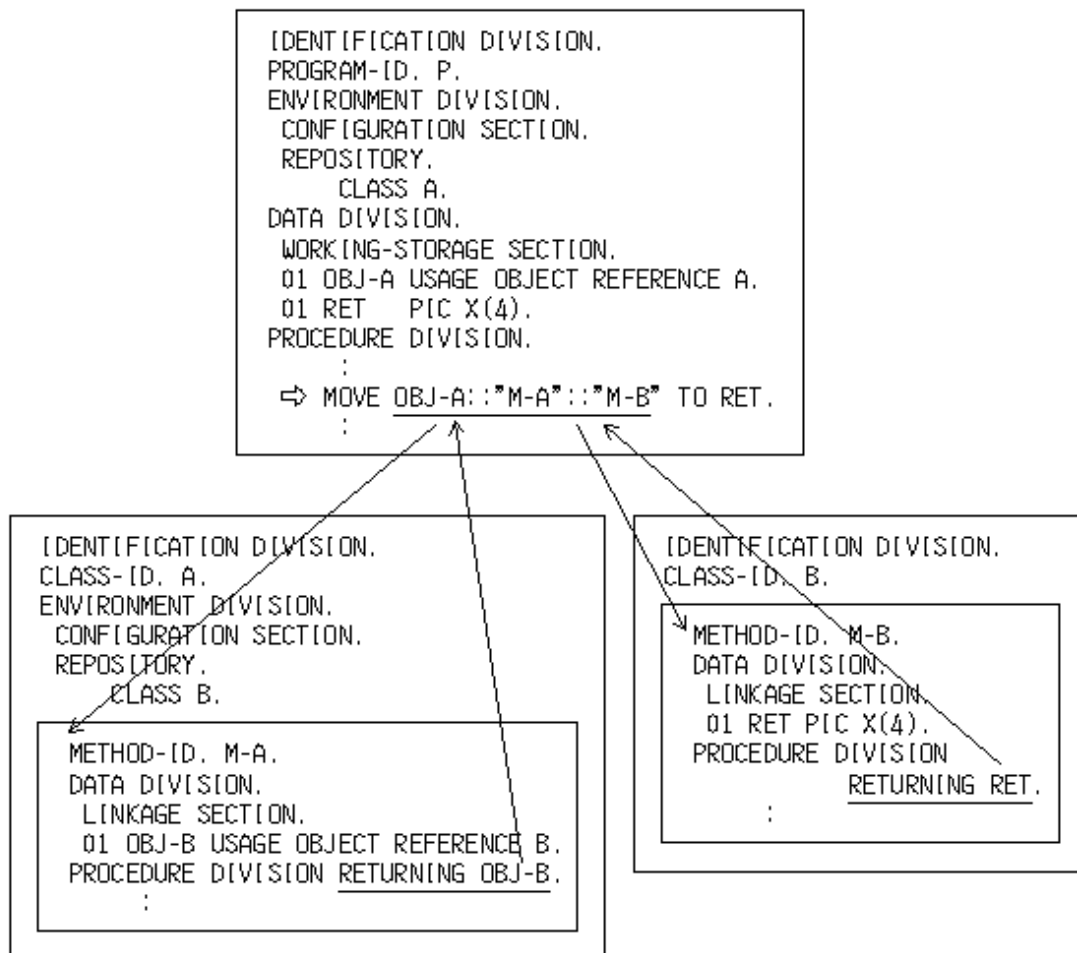
なお、`_FINALIZE` を使用しないのであれば、従来どおり、ソースの一部を再翻訳してアプリケーションを構成することが可能です。

`_FINALIZE` を使用する場合、`_FINALIZE` メソッド中に `STOP RUN` 文を記述してはいけません。また、`_FINALIZE` から呼び出されるプログラムまたはメソッド中にも `STOP RUN` 文を記述してはいけません。

## 16.7.7 間接参照クラス

呼び出したメソッドの復帰値がオブジェクト参照項目だった場合、ソース上には現れないクラス、つまり暗黙的な参照クラスが必要となる場合があります。この暗黙的に参照するクラスのことを間接参照クラスと呼びます。

ここでは、間接参照クラスが顕著に現れるパターンとして、行内呼出しの入れ子を例に使用方法を説明します。



上図で、プログラムPに記述された行内呼出しの入れ子は、内部的には以下のように分解されます。

```

:
WORKING-STORAGE SECTION.
01 OBJ-A USAGE OBJECT REFERENCE A.
01 RET PIC X(4).
01 temp USAGE OBJECT REFERENCE B. ←内部的に一時域を生成
PROCEDURE DIVISION.
:
** MOVE OBJ-A::"M-A"::"M-B" TO RET.
**
 SET temp TO OBJ-A::"M-A".
 MOVE temp ::"M-B" TO RET.] 生成した一時域を利用して
: 入れ子を展開
:

```

このとき、内部的に生成される一時域(上図temp)は、メソッドM-Aの復帰値と同じ属性がとられるため、クラスBのオブジェクト参照項目として定義されます。つまり、内部的に生成される一時域(暗黙に定義されたデータ項目)によってクラスBが参照されることになります。このようなクラスを間接参照クラスと呼び、明示的に参照されるクラスと同様、リポジトリ段落で宣言しなければなりません。つまり、プログラムPのリポジトリ段落にはクラスBの宣言が必要になります。

以上、行内呼出しの入れ子の場合を例に説明しましたが、このほかにも、

- ・ オブジェクトプロパティの入れ子
- ・ 復帰値に別のクラス(適合関係が成立するクラス)のオブジェクト参照項目が指定されたメソッドを呼び出す場合

などに間接参照クラスの宣言が必要となることがあります。行内呼出し、オブジェクトプロパティを含めて、復帰値がオブジェクト参照項目のメソッドを呼び出す場合には意識してコーディングしてください。

なお、間接参照クラスをリポジトリ段落で宣言しないで翻訳した場合、翻訳時にエラーメッセージが出力されるので、メッセージに従ってソースを修正してください。

## 16.7.8 相互参照クラス

実行時、複数のオブジェクトインスタンスを結びつけたい場合、つまり、オブジェクトデータ中にオブジェクト参照項目を定義したい場合があります。このような場合、直接的または間接的に相互に参照関係が成立することがあります。この相互に参照関係が成立するクラスのことを相互参照クラスと呼び、実行形式の作成にはテクニックが必要となります。

ここでは、相互参照クラスが成立するいくつかのパターンと、それらの実行形式を作成するために必要な作業について説明します。

なお、プロジェクト配下での実現方法については、“[17.4.5 相互参照クラス](#)”で詳しく述べていますが、本説明を一読した上で参照すると、より理解しやすくなります。

### 16.7.8.1 相互参照パターン

相互参照のパターンには、以下の3つがあります。

- ・ 自クラスの相互参照
- ・ 他クラスとの直接相互参照
- ・ 他クラスとの間接相互参照

それぞれのパターンについて、以下に具体的に説明します。

#### 自クラスの相互参照

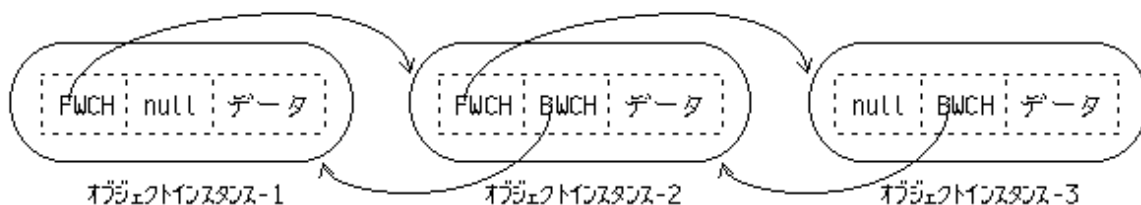
自クラスのオブジェクトインスタンスをリスト構造で管理するような場合、オブジェクトデータ中に自クラスを保持するオブジェクト参照項目を宣言します。

```

IDENTIFICATION DIVISION.
CLASS-ID. A INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS FJBASE.
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 FWCH OBJECT REFERENCE A.
01 BWCH OBJECT REFERENCE A.
01 OBJ-DATA PIC X(20).
PROCEDURE DIVISION.
 ;

```

実行時のオブジェクトインスタンスイメージ



このようにオブジェクトインスタンスを結合しておくことによって、生成したオブジェクトインスタンスを順/逆順に処理したり、全オブジェクトインスタンスを走査したりする処理が容易に実現できます。



## 注意

通常、クラス定義内で参照するクラスはリポジトリ段落で宣言しなければなりません、自クラスについては不要です。宣言した場合、翻訳時エラーとなりますので、注意してください。

## 他クラスとの直接相互参照

他クラスのオブジェクトインスタンスと密接に関係するような構成、つまり、片方のオブジェクトインスタンスから、もう片方のオブジェクトインスタンスをたどれるような構成を構築する場合、直接的な相互参照関係が成立します。以下、名前クラス(NAME)と住所クラス(ADDR)の場合を例に説明します。

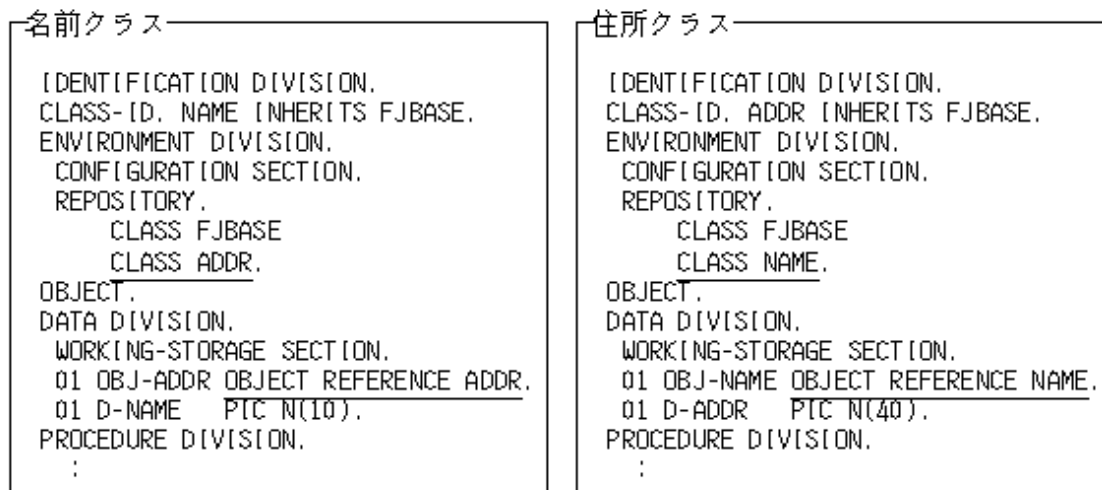
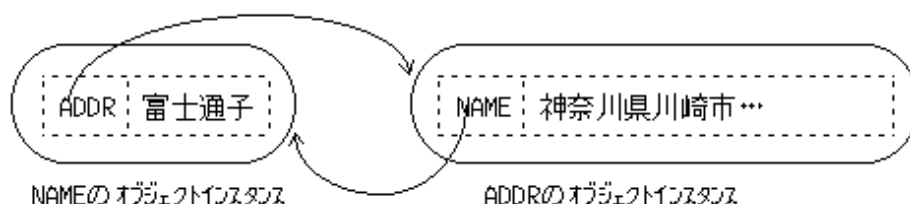


図16.3 実行時のオブジェクトインスタンスイメージ



このようにオブジェクトインスタンスを相互結合しておくことによって、名前から住所を求めることも、逆に住所から名前を求めることも可能になります。

### 他クラスとの間接相互参照

他クラスのオブジェクトインスタンスと密接に関係するような構成で、間接的に相互参照関係が成立する場合があります。以下、

- ・ 名前クラス(NAME)が住所クラスのオブジェクトインスタンスを、
- ・ 住所クラス(ADDR)が所属クラスのオブジェクトインスタンスを、
- ・ 所属クラス(SECT)が所属長の名前クラスの名前クラスのオブジェクトインスタンスを

保持する場合を例に説明します。

名前クラス

```

IDENTIFICATION DIVISION.
CLASS-ID. NAME INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS FJBASE
 CLASS ADDR.
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 OBJ-ADDR OBJECT REFERENCE ADDR.
 01 D-NAME PIC N(10).
PROCEDURE DIVISION.
 ;

```

住所クラス

```

IDENTIFICATION DIVISION.
CLASS-ID. ADDR INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS FJBASE
 CLASS SECT.
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 OBJ-SECT OBJECT REFERENCE SECT.
 01 D-ADDR PIC N(40).
PROCEDURE DIVISION.
 ;

```

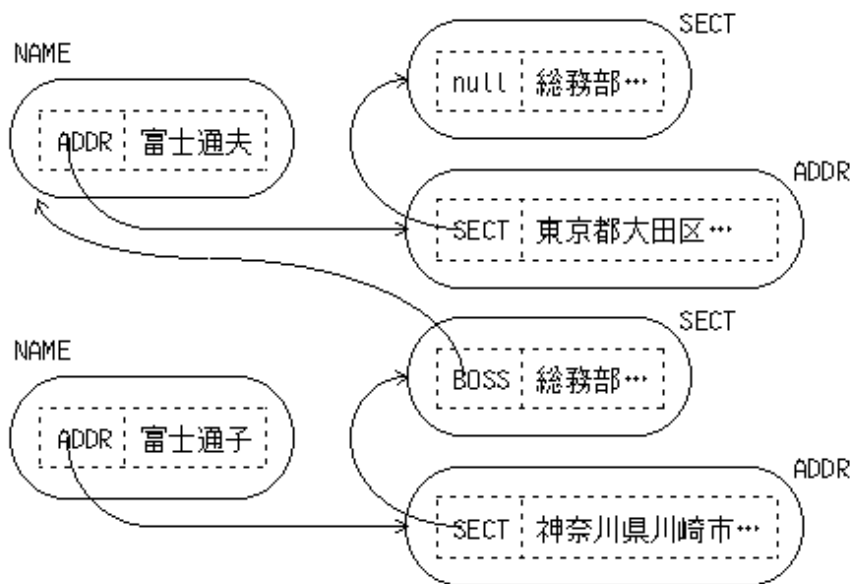
所属クラス

```

IDENTIFICATION DIVISION.
CLASS-ID. SECT INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS FJBASE
 CLASS NAME.
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 OBJ-BOSS OBJECT REFERENCE NAME.
 01 D-SECT PIC N(20).
PROCEDURE DIVISION.
 ;

```

図16.4 実行時のオブジェクトインスタンスイメージ



このように、オブジェクトインスタンスが複雑に結合し合うような場合、容易に間接相互参照関係が成立します。

### 16.7.8.2 相互参照クラスの翻訳

前述のとおり、相互参照クラスには大きく3つのパターンがありますが、自クラスの相互参照の場合は、翻訳およびリンク時に特別な考慮をする必要はありません。通常のクラス定義と同様に翻訳、リンクすれば実行形式が作成できます。これに対して、他クラスとの相互参照の場合、翻訳時に必要なリポジトリファイルがそろわないため、翻訳前にリポジトリファイルの準備をしなければなりません。

以下、直接相互参照(名前クラスと住所クラス)の場合を例に説明します。

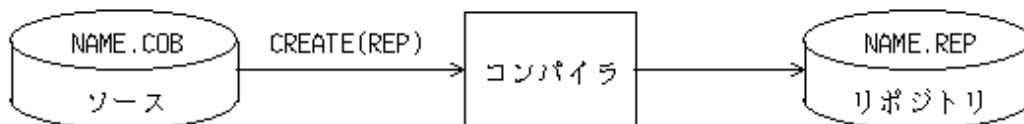
名前クラスを翻訳するためには、住所クラスのリポジトリファイルが必要です。住所クラスのリポジトリファイルを生成するには住所クラスを翻訳しなければなりません、その際、名前クラスのリポジトリファイルが必要になります。いわゆる、「タマゴが先か、ニワトリが先か」の状態に陥ってしまうわけです。

このような状態を回避するために、翻訳オプションCREATE(REP)を用意しました。翻訳時にCREATE(REP)を指定した場合、コンパイラはリポジトリファイルだけを生成します。このオプションを利用して、名前クラスと住所クラスを翻訳してみます。

#### ステップ1：名前クラスのリポジトリを生成

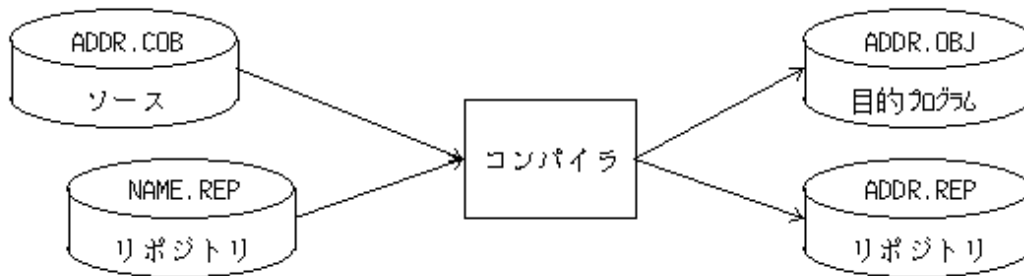
翻訳オプションCREATE(REP)を指定して名前クラスを翻訳します。

この場合、あくまでリポジトリの生成が目的のため、参照するクラス(ADDR)のリポジトリファイルを入力する必要はありません。ただし、親クラスのリポジトリは必要です(下図ではFJBASEの入力は省略しています)。また、登録集が存在する場合は、登録集も入力してください。



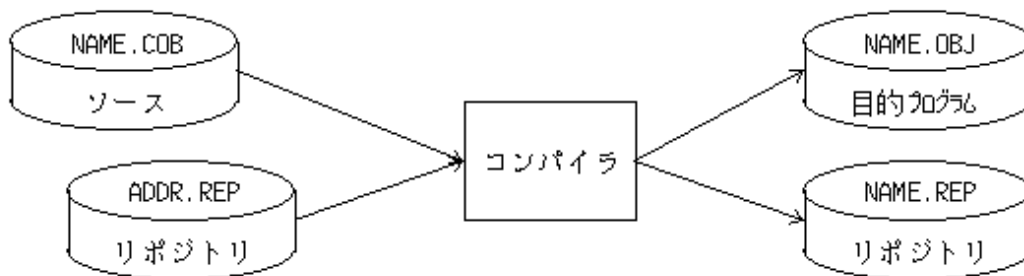
## ステップ2：生成したリポジトリを利用して、住所クラスを翻訳

ステップ1で生成した名前クラスのリポジトリ(NAME.REP)を利用して、住所クラスの目的プログラムを生成します。このとき、翻訳オプションCREATE(OBJ)を有効(省略値のため、指定なしでよい)にしてください。



## ステップ3：住所クラスのリポジトリを利用して、名前クラスを翻訳

ステップ2で生成した住所クラスのリポジトリ(ADDR.REP)を利用して、名前クラスの目的プログラムを生成します。ステップ2と同様、翻訳オプションCREATE(OBJ)を有効にしてください。



このように、まず、いずれかのクラスのリポジトリだけを生成し、その後、順番に通常翻訳することで目的プログラムが作成できます。間接相互参照の場合も同じで、いずれかのクラスのリポジトリだけを生成し、後は芋づる式に目的プログラムを作成します。

翻訳オプションCREATE(REP)を指定して作成されたリポジトリファイルは仮リポジトリと呼びます。仮リポジトリは正式なリポジトリを生成するまでの一時的なものであり、相互参照クラスを実現する場合にだけ利用できます。仮リポジトリには以下の制限がありますので注意してください。

- ・ 分離されたメソッドでは、PROTOTYPE宣言されたクラスのリポジトリファイルとして使用できません。
- ・ クラスブラウザへの入力として使用できません。



### 注意

CREATE(REP)オプションが指定された場合、コンパイラは手続き部の解析を行いません。このため、手続き部にエラーが存在してもメッセージは出力されません。後の目的プログラム生成時にエラーチェックされるため、その際に必要に応じて修正してください。

## 16.7.8.3 相互参照クラスのリンク

実行形式を静的リンク構造や動的プログラム構造で構築する場合、通常、クラス定義と同じようにリンクします。しかし、動的リンク構造で、かつ、他クラスとの相互参照を構築する場合には、特別な考慮が必要となります(相互参照関係にあるクラスをひとつのDLLにするのであれば特別な考慮は不要です。それぞれを独立したDLLにする場合に考慮が必要となります)。

直接相互参照(名前クラスと住所クラス)の場合の例を、以下に説明します。

名前クラスをリンクするためには、住所クラスのインポートライブラリが必要です。住所クラスのインポートライブラリを生成するには住所クラスをリンクしなければなりません、その際、名前クラスのインポートライブラリが必要です。翻訳時と同様に、いわゆる、「タマゴが先か、ニワトリが先か」の状態に陥ってしまうわけです。

このような状態を回避するために、まず、いずれかのクラスのインポートライブラリを作成します。インポートライブラリは、リンク時に作成する方法とモジュール定義ファイルより作成する方法があるため、後者を利用します。

### ステップ1：名前クラスのモジュール定義ファイルを作成

エディタを利用して、名前クラスのモジュール定義ファイル(NAME.DEF)を作成します。モジュール定義ファイルは以下の形式です。

|                       |            |
|-----------------------|------------|
| NAME.DEF              |            |
| LIBRARY "NAME"        | ←ライブラリ名を指定 |
| EXPORTS _NAME_FACTORY | ←エントリ名を指定  |

LIBRARYにはライブラリ名を、EXPORTSにはエントリ名を指定します。

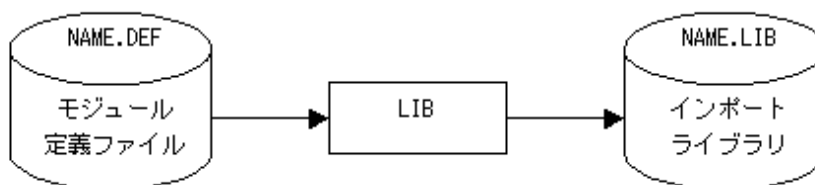
エントリ名の規約は、以下のとおりです。

- ・ 先頭の1バイトは半角のアンダースコア：固定文字
- ・ 続けてクラス名：可変文字
- ・ 続けて半角のアンダースコア：固定文字
- ・ 最後にFACTORY：固定文字

### ステップ2：名前クラスのインポートライブラリを作成

ステップ1で作成したモジュール定義ファイルを利用して、名前クラスのインポートライブラリを作成します。

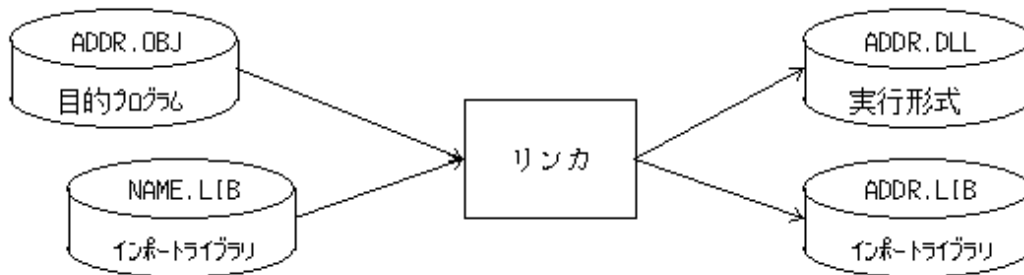
```
LIB /OUT:NAME.LIB /DEF:NAME.DEF /MACHINE:IX86
```



### ステップ3：住所クラスの実行形式を作成

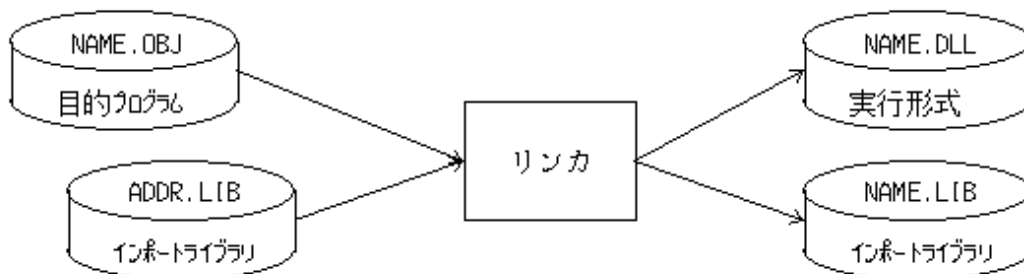
ステップ2で作成した名前クラスのインポートライブラリを利用して住所クラスの実行形式(.DLL)を作成します。このとき、住所クラスのインポートライブラリが作成されます。





#### ステップ4：名前クラスの実行形式を作成

ステップ3で作成した住所クラスのインポートライブラリを利用して名前クラスの実行形式を作成します。このとき、名前クラスのインポートライブラリは上書きします。



考え方は翻訳時と同じです。まず、いずれかのクラスのインポートライブラリを作成し、その後、順番にリンクすることで実行形式が作成できます。間接相互参照の場合も同様に、いずれかのクラスのインポートライブラリを作成し、後は芋づる式に実行形式を作成します。

#### 参考

以下の場合にも、リンク時に注意が必要です。

- ・ 分離されたメソッドで、自クラスファクトリメソッドを呼び出す(INVOKE 自クラス名“ファクトリメソッド名”)。かつ、
- ・ クラス定義と分離されたメソッドを別DLLで構築する。かつ、
- ・ 動的リンク構造とする。

このような場合、事前にインポートライブラリを準備することによって実現可能ですが、基本的には、クラス定義と分離されたメソッドはひとつのDLLで構築することをおすすめします。

#### 16.7.8.4 相互参照クラスの実行

実行の際には、特に考慮すべきことはありません。

通常のクラス定義と同じように実行することができます。

## 第17章 オブジェクト指向プログラミングの開発

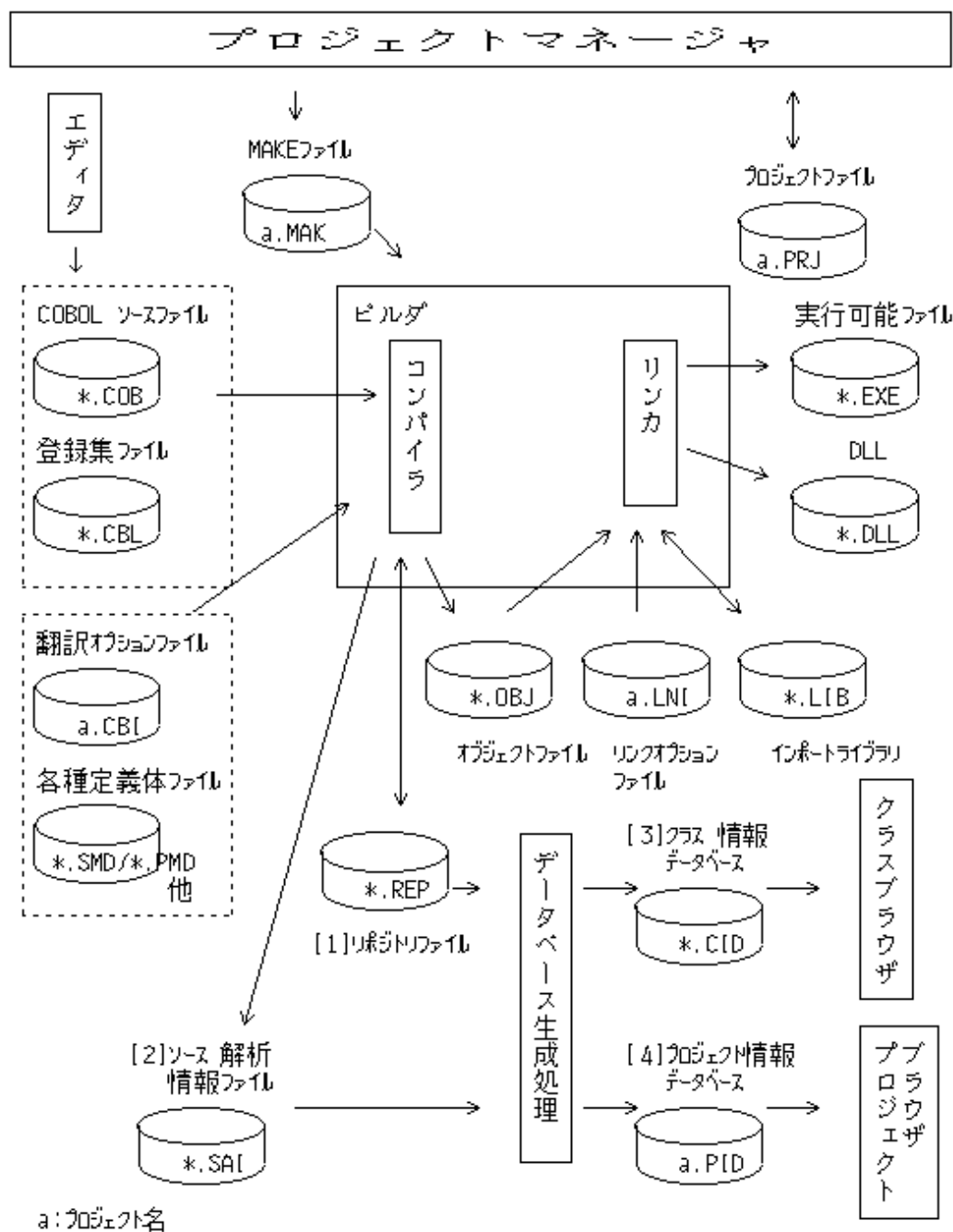
本章では、オブジェクト指向プログラミングの開発方法について説明します。

### 17.1 オブジェクト指向プログラミングで使用する資源

ここでは、オブジェクト指向プログラミングを実現するために必要な資源およびツールと資源の関係について説明します。

オブジェクト指向プログラミングを行う場合に使用するファイルの関係を以下に示します。

なお、本節の説明は、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することを前提としています。



## オブジェクト指向プログラミングで使用するファイル

|     | ファイルの内容           | ファイル名の形式     | 入出力 | 使用する条件または作成される条件                                 |
|-----|-------------------|--------------|-----|--------------------------------------------------|
| [1] | クラス情報             | クラス外部名.REP   | 入力  | COBOL ソースファイルを翻訳するときおよびクラス情報データベースを生成するときに使用します。 |
|     |                   |              | 出力  | クラスが定義されているCOBOL ソースファイルを翻訳すると作成されます。            |
| [2] | ソースファイル定義情報       | ソースファイル名.SAI | 入力  | プロジェクト情報データベースを生成するときに使用します。                     |
|     |                   |              | 出力  | COBOL ソースファイルを翻訳すると作成されます。                       |
| [3] | クラスブラウザで表示する情報    | 任意の名前.CID    | 入力  | “クラスブラウザ”メニューが選択された場合に使用します。                     |
|     |                   |              | 出力  | “クラス情報データベース”メニューの“生成”が選択されたときに作成されます。           |
| [4] | プロジェクトブラウザで表示する情報 | プロジェクト名.PID  | 入力  | “プロジェクトブラウザ”メニューが選択された場合に使用します。                  |
|     |                   |              | 出力  | “ビルド”または“リビルド”を行ったときに作成されます。                     |

## 17.2 開発手順

ここでは、オブジェクト指向プログラミングを行う手順を説明します。

1. オブジェクトの設計を行います。[参照]“[第16章 オブジェクト指向プログラミング機能～基本的な使い方～](#)”  
その際、クラスブラウザを使用して、既存資産の再利用を図ります。[参照]“[17.3 クラスの検索とクラス情報の取得](#)”
2. プロジェクトを作成し、COBOLソースファイルを作成/編集します。[参照]“[第6章 プロジェクト管理機能](#)”
3. プロジェクトに登録されているCOBOLソースファイルに対して、ターゲットリポジトリファイル、依存リポジトリファイルを登録します。  
[参照]“[17.4.3.1 リポジトリファイル](#)”
4. プロジェクトに登録されているCOBOLソースファイルに対して、依存関係を持つファイル(登録集ファイル、各種定義体ファイル)を登録します。
5. プロジェクトに登録されている実行可能ファイルおよびDLLに対して、インポートライブラリを登録します。[参照]“[17.4.4.1 インポートライブラリ](#)”
6. プロジェクトのビルド・リビルドを行います。[参照]“[6.4.5 プロジェクトのビルド・リビルド](#)”
7. プログラムを実行します。[参照]“[6.4.7 アプリケーションプログラムの実行](#)”
8. 開発作業が終了したら、クラスを公開します。[参照]“[17.5 クラスの公開](#)”

## 17.3 クラスの検索とクラス情報の取得

“[15.3 オブジェクト指向のメリット](#)”で説明したように、オブジェクト指向プログラミングには、以下のメリットがあります。

- ・ 部品化が容易にできます。
- ・ 既存の部品の流用が容易にできます。

しかし、どんなに部品化が容易であっても、その部品を使用するための情報が伝わらなければ、他の人が使用することはできません。また、既存の部品の流用が容易であっても、その部品を使用するための情報を入手しなければ、使用することはできません。

オブジェクト指向プログラミングで、既存の部品を流用する場合に入手しておかなければならない情報として、以下のようなものがあります。

- ・ クラス名とその機能
- ・ クラスの持っているメソッドの名前およびその機能
- ・ 各メソッドのインタフェース

## クラス名とその機能

プログラミング作業は、何らかの目的を実現するために行います。

そのため、再利用するクラスもその目的のために使用できるものでなければなりません。



### 例

ある会社の従業員管理プログラムを作成するために従業員オブジェクトを作成するような場合、目的に対して無関係なクラス(たとえば、農場で牛の健康管理を行っているようなクラス)の流用は意味がありません。

このように、利用者は目的のプログラムを作成するために再利用するクラスとして、どのような機能を持った、どのような名前のクラスがあるかという情報を事前に入手していなければなりません。

## クラスの持っているメソッド名およびその機能

オブジェクト指向プログラミングでは、クラスからオブジェクトを生成することも、そのオブジェクトを動作させることにも、メソッドの呼出しを行う必要があります。

既存のクラスを再利用したい場合、目的に合った処理を行うメソッドがない、またはあっても名前がわからなければ、利用者にとってはクラスが利用できないのと同じことになります。



### 例

従業員オブジェクトを使用して従業員の平均の月給を算出するようなプログラムを作成する場合には、従業員オブジェクトの中に従業員の月給を知るメソッドがあり、かつ、利用者はそのメソッドの名前がわからなければ、使用することはできません。

このように、利用者は、自分が使用するクラスに定義されているメソッドの名前および機能という情報も入手しておく必要があります。

## 各メソッドのインタフェース

自分の作成したいプログラムの目的に合ったクラスおよびメソッドが見つかって、利用者はそのメソッドに対して、どのような値をどのような形式で渡すと、どのような値がどのような形式で返ってくるのかがわからないと、思った通りの結果を得ることはできません。



### 例

ある職場の従業員オブジェクトを集めた職場オブジェクトがあり、その中の個々の従業員オブジェクトを検索するための検索オブジェクトが用意されていたとします。

この職場オブジェクトから特定の人のオブジェクトを検索する場合、検索メソッドに対して何の情報を渡せばよいのか(たとえば、名前や従業員番号など)、また、どのような情報が返ってくるのかといったインタフェース情報がわからなければ、そのメソッドを使用することはできません。

このように、利用者は、自分が使用するメソッドのインタフェースも理解している必要があります。

クラスの検索およびクラス情報の取得を行うために、プロジェクトマネージャでは「クラスブラウザ」を用意しています。

### 17.3.1 クラスブラウザ

クラスブラウザは、既存クラスの検索を補助して再利用するための情報を提示する、クラスの再利用のための補助ツールです。

ここでは、クラスブラウザの概要および操作方法について説明します。

#### 17.3.1.1 概要

前述したように、名前、インタフェースといった情報は、クラスを再利用するための必須情報です。

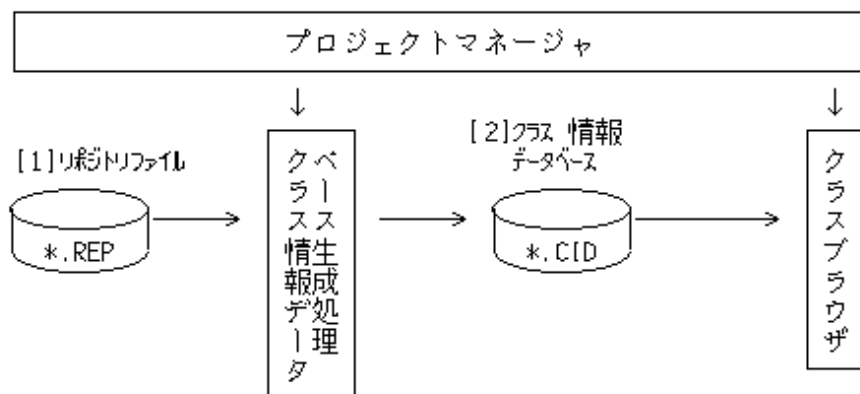
クラスブラウザは、これらの情報を利用者に提供するツールです。

クラスブラウザは、以下の機能を持っています。

- ・ クラス外部名および継承関係の表示
- ・ クラス単位のメソッド外部名一覧の表示
- ・ メソッドの入出力パラメタ情報の表示
- ・ 各種テンプレート機能

### 17.3.1.2 使用する資源

クラスブラウザを使用する場合の資源の流れは、下図のようになります。



#### [1] リポトリファイル

リポトリファイルは、翻訳時に再利用するクラスの情報コンパイラに通知するために使用する資源ですが、それ以外にクラスの情報クラスブラウザに通知するためにも使用します。

クラスブラウザで表示する情報は、まず、翻訳処理時にリポトリファイルに取り出します。

なお、翻訳時の資源としてのリポトリの説明については、“[17.4.3 翻訳処理](#)”を参照してください。

#### [2] クラス情報データベース

クラス情報データベースは、クラスブラウザが表示するクラス情報をまとめて、管理するための資源です。

「クラス情報データベース生成処理」では、リポトリファイルに格納されているクラス情報を集めて、クラス情報データベースを生成します。

### 17.3.1.3 クラス情報データベースの生成

ここでは、クラス情報データベースの生成方法について説明します。

#### 生成手順

クラス情報データベースを生成するための手順を以下に説明します。

1. リポトリファイル公開パス<sup>(注)</sup>に、リポトリファイルが複写されていることを確認します。
2. リポトリファイルが格納されているパス名およびクラス情報データベースファイル名を指定します。[参照]“[17.3.1.3.1 オプション設定ウィンドウ](#)”
3. クラス情報データベースを生成します。[参照]“[17.3.1.3.2 データベース生成](#)”



#### 注意

リポトリファイル公開パスについては、“[17.5 クラスの公開](#)”を参照してください。

### 17.3.1.3.1 オプション設定ウィンドウ

クラス情報データベースを生成するためには、以下の情報が必要になります。

- ・ リポジットファイル収集パス
- ・ クラス情報データベースファイル名

これらの情報は、オプション設定ウィンドウを使用して設定します。



オプション設定ウィンドウで設定した情報は、レジストリに格納されます。  
この情報は、利用者が変更するまで有効になります。

### リポジットファイル収集パスの設定

リポジットファイル収集パスとは、クラス情報データベースを生成するときに、クラス情報を収集するためのリポジットファイルが格納されているパスを指します。

クラス情報データベースを生成するときに、ここで指定したパス上のリポジットファイルが解析されます。

なお、リポジットファイル収集パスは、複数個登録することができます。

以下に、リポジットファイル収集パスの設定方法について説明します。

1. リポジット格納フォルダの[追加]ボタンをクリックします。
2. コモンダイアログが表示されたら、設定するパス名を選択します。
3. パス情報の設定が終了するまで、1.から2.の操作を繰り返して行ってください。



パス指定はコモンダイアログを使用しているため、生成済みのパスだけが指定できます。

### クラス情報データベースファイル名の設定

クラス情報データベースファイル名は、オプション設定ウィンドウで設定します。

以下に、クラス情報データベースファイル名の設定方法について説明します。

1. クラスデータベースファイルのエディットボックスにファイル名を入力します。または、[参照]ボタンをクリックします。
2. コモンダイアログが表示されたら、ファイル名を選択します。

### 17.3.1.3.2 データベース生成

クラス情報データベースを生成するためには、プロジェクトマネージャウィンドウの[ツール]-[クラスデータベース]メニューから、“生成”を選択します。

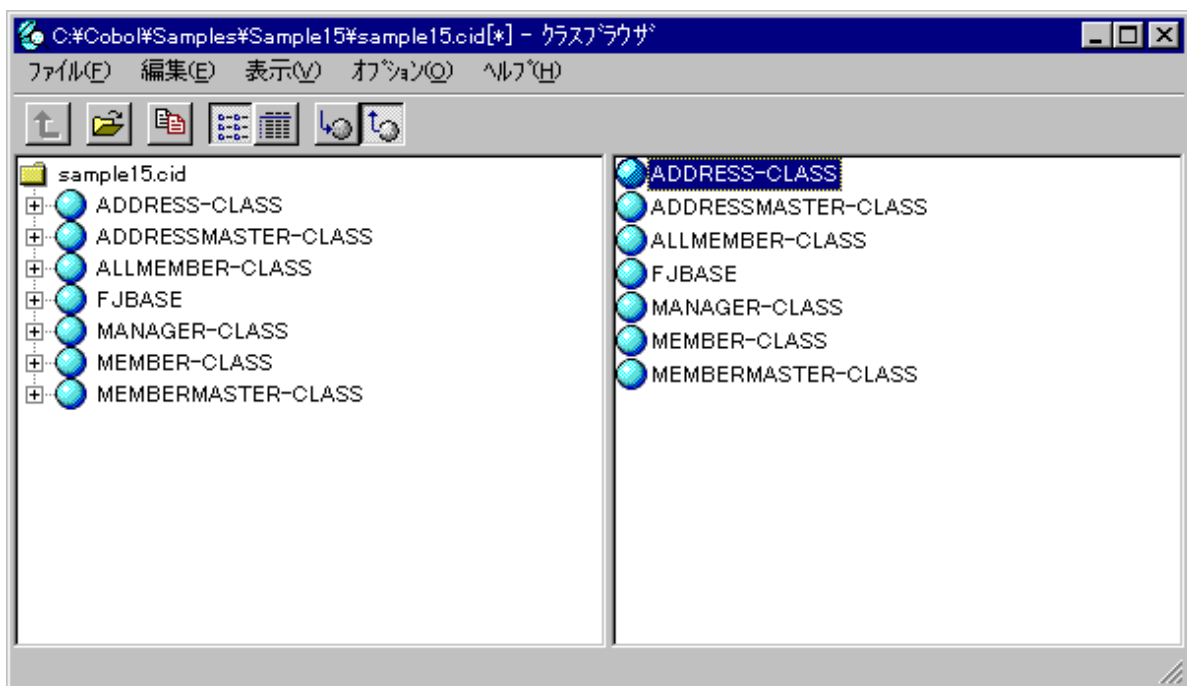
### 17.3.1.4 クラスブラウザの表示/操作手順

クラスブラウザを表示して各種機能を使用する手順の概要を説明します。

1. プロジェクトマネージャの[ツール]メニューから“クラスブラウザ”を選択します。  
→ クラスブラウザウィンドウが表示されます。
2. 継承関係をたどることにより、目的のクラスを検索します。[参照]“[17.3.1.5 クラスブラウザウィンドウ](#)”の“表示可能情報および表示方法”
3. 目的のクラスのメソッド名を確認します。[参照]“[17.3.1.5 クラスブラウザウィンドウ](#)”の“表示可能情報および表示方法”
4. 各メソッドのインタフェースを確認します。[参照]“[17.3.1.5 クラスブラウザウィンドウ](#)”の“表示可能情報および表示方法”
5. [ファイル]メニューから“閉じる”を選択して終了します。

### 17.3.1.5 クラスブラウザウィンドウ

ここでは、クラスブラウザを使った各操作について説明します。



#### 表示可能情報および表示方法

クラスブラウザが表示するウィンドウ上では、各種の情報を確認することができます。表示される情報および表示方法を“[表17.1 クラスブラウザの情報表示](#)”に示します。

表17.1 クラスブラウザの情報表示

| 情報      | 定義   | 表示方法          |
|---------|------|---------------|
| 継承関係    | クラス  | ツリーの1つ下の階層に表示 |
| メソッド一覧  | クラス  | ツリーの1つ下の階層に表示 |
| インタフェース | メソッド | リストビューで表示     |

## 継承関係の表示

クラスブラウザウィンドウで、クラスの継承関係を表示する手順について説明します。

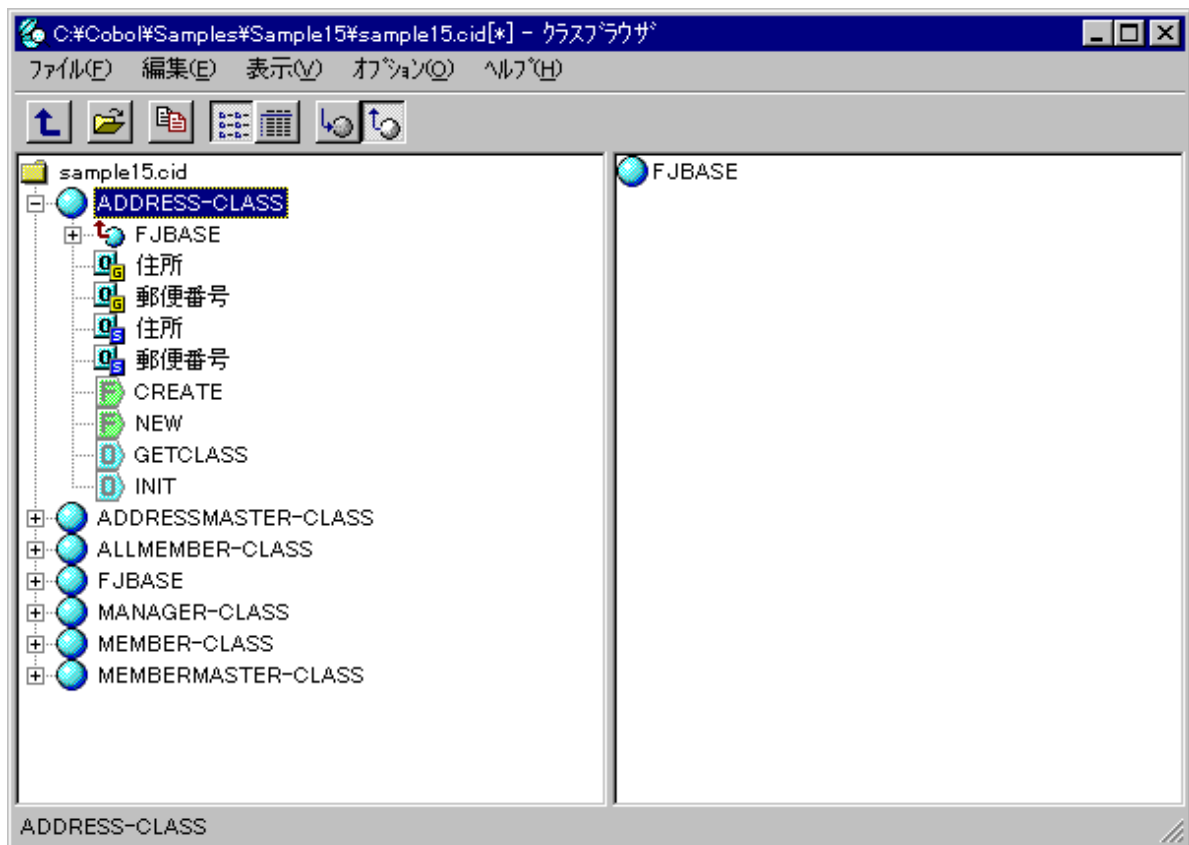
1. [表示]メニューから、“継承”または“継承元”を選択します。
  - ー 継承:ツリーは子クラス方向に拡張していきます。
  - ー 継承元:ツリーは親クラス方向に拡張していきます。
2. クラスのアイコンをダブルクリックする、またはアイコンの前に表示されている+記号をクリックすることにより、ツリーは拡張していきます。

## メソッド一覧の表示

メソッド表示機能は、クラス定義中に定義されているファクトリメソッド/オブジェクトメソッド(プロパティメソッド/プロパティデータも含む)および継承関係により自クラスのメソッドとして使用できる親クラスのファクトリメソッド/オブジェクトメソッドを表示します。

クラスブラウザウィンドウを使用して、そのクラスで使用できるメソッド一覧を表示するためには、以下の操作を行ってください。

- ・ クラスのアイコンをダブルクリックします。または、アイコンの前に表示されている+記号をクリックします。

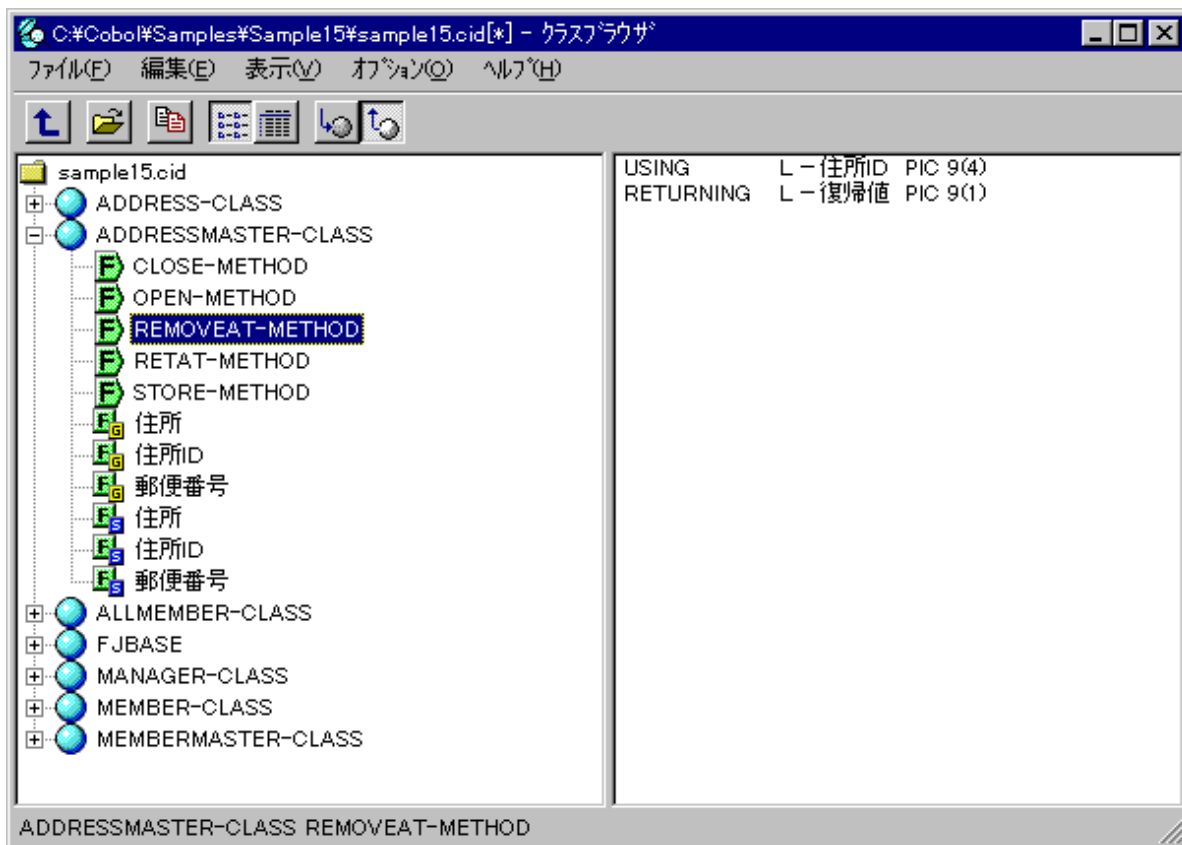


## インタフェースの表示

クラスブラウザでは、各メソッドのPROCEDURE DIVISIONのUSING指定またはRETURNING指定に記述されているデータ項目の、データ名および属性を表示することができます。

メソッド名をクリックすることにより、情報が表示されます。





## テンプレート展開機能

クラスブラウザは、他のCOBOLソースファイルに対して、特定の構文のテンプレートを展開する機能を持っています。この機能をテンプレート展開機能と呼びます。

クラスブラウザで以下の構文のテンプレートを展開することができます。

- ・ リポジトリ段落のクラス指定子
- ・ オブジェクトプロパティ
- ・ INVOKE文

これらの構文のテンプレートの展開形式と修正方法について説明します。



### 注意

ブラウザのテンプレート展開機能は、システムのクリップボードを使用しています。そのため、クリップボードからの貼り付け機能を持っているエディタであれば、プロジェクトマネージャに標準で組み込まれているエディタでなくても、この機能を使用することができます。

## リポジトリ段落のクラス指定子

リポジトリ段落に記述するクラス指定子のテンプレート展開を用いる手順を以下に説明します。

1. クラスブラウザ上でクラス名を選択します。
2. [編集]メニューから“変換コピー”を選択します。
3. エディタ上で、リポジトリ段落の後へカーソルを位置付け、“貼り付け”を行ってください。

クラス指定子のテンプレート展開を行うと、選択したクラスを参照するための次のようなテンプレートが展開されます。

|                |
|----------------|
| CLASS    クラス名. |
|----------------|

なお、最後のピリオドは最後のクラス指定子の場合だけ必要です。それ以外の場合は削除してください。

### オブジェクトプロパティのテンプレート展開

プロパティのテンプレート展開を用いる手順を以下に説明します。

1. クラスブラウザ上でプロパティメソッド名を選択します。
2. [編集]メニューから“変換コピー”を選択します。
3. エディタ上で、文を挿入したい場所にカーソルを位置付け、“貼り付け”を行ってください。

これにより、選択したプロパティメソッドを呼び出す次のようなオブジェクトプロパティのテンプレートが展開されます。

|                |
|----------------|
| プロパティ名 OF クラス名 |
|----------------|

なお、テンプレートではプロパティメソッドを呼び出すオブジェクト参照はクラス名として展開されます。呼び出すプロパティメソッドがオブジェクトメソッドである場合、このクラスを参照する適切なオブジェクト参照データに置き換えてください。

### INVOKE文のテンプレート展開

INVOKE文のテンプレート展開を用いる手順を以下に説明します。

1. クラスブラウザ上でプロパティメソッドを除くメソッド名を選択します。
2. [編集]メニューから“変換コピー”を選択します。
3. エディタ上の文で、展開する場所にカーソルを位置付け、“貼り付け”を行ってください。

これにより、選択したメソッドを呼び出すINVOKE文の次のようなテンプレートが展開されます。

|                                                 |
|-------------------------------------------------|
| INVOKE クラス名 “メソッド名” USING データ名… RETURNING データ名. |
|-------------------------------------------------|

展開されたテンプレートは、そのままでは正しく翻訳できない場合がありますので、以下の点を確認して必要ならソースを修正してください。

1. テンプレートではメソッドを呼び出すオブジェクト参照はクラス名として展開されます。呼び出すメソッドがオブジェクトメソッドである場合、このクラスを参照する適切なオブジェクト参照データに置き換えてください。
2. USING指定およびRETURNING指定のデータ名には、テンプレート展開時に選択したメソッドの手続き部見出しのUSING指定またはRETURNING指定のデータ名の指定がそのまま複製されています。この際、対応するデータ名の定義は展開されていないため、ブラウザに表示されるメソッドのインタフェースに従ってデータ名の定義を追加してください。
3. 最後のピリオドは、IF文などの条件文やPERFORM文による繰り返し構造の内部に指定できません。テンプレートを展開した場所がそのような場所ならピリオドを削除してください。

### 17.3.1.6 クラスの検索条件の入力

クラスブラウザは、検索条件を入力することにより、表示するクラスを限定することができます。

[ファイル]メニューから“開く”を指定すると、検索条件を入力するためのダイアログが表示されます。



## エディットボックス

### ファイル名

クラス情報データベースファイル名を指定します。

### 名前

クラスブラウザのクラス情報データベース名の下に、表示されるクラス名を指定します。指定可能な文字は、クラス名として使用可能な文字および“?”、“\*”です。

“?”は任意の1文字、“\*”は長さ0を含む任意の文字列を示します。

## チェックボックス

### 大文字/小文字を区別する

大文字/小文字を区別して検索するか、区別しないで検索するかを指定します。

## ラジオボタン

### クラス

以下のどちらかを選択します。

### すべて

名前の入力に従って表示されます。

### FJBASE

基底クラスが表示されます。

## 17.4 プロジェクト管理

---

再利用するクラス(利用できる部品)が決定すると、次はプログラムの実装段階になります。

プロジェクト管理機能を使用すると、とくにオブジェクト指向プログラム開発の場合は、開発作業を容易に行うことができます。

ここでは、オブジェクト指向プログラム開発にプロジェクト管理を使用することによるメリット、翻訳の手順およびリンクの手順を中心に、開発手順の説明を行います。

### 17.4.1 プロジェクト管理のメリット

---

オブジェクト指向の概念を使用しないプログラム開発作業では、ライブラリ(実行可能ファイルまたはDLL)間のリンク関係によってリンク処理の順序に規定が発生し、誤った順序でリンク処理を行うと、リンクエラーまたは実行異常が発生しました。

オブジェクト指向プログラムの開発作業では、リンク処理の順序を決定する要因が従来の開発作業よりもさらに多様化し、翻訳処理の順序付けも発生します。

プロジェクト管理機能は、ファイル間の関係を定義することにより、自動的に翻訳/リンク処理が必要なファイルを判断し、適切な順序で実行します。そのため、とくにオブジェクト指向プログラミングを行う場合は、プロジェクト管理機能を使用することをおすすめします。

翻訳の手順については“[17.4.3.1 リポジトリファイル](#)”の“翻訳の手順”、リンクの手順については“[17.4.4.1 インポートライブラリ](#)”の“リンクの手順”を参照してください。

### 17.4.2 プログラム構造

---

プロジェクトを作成するためには、まずプロジェクト内のプログラム構造を設計しなければなりません。

ここでは、プログラム構造について簡単に説明します。

#### 17.4.2.1 翻訳単位とリンク単位

翻訳単位とは、翻訳処理によって1つのオブジェクトファイルを生成する単位を指します。

オブジェクト指向プログラミングでは、以下の定義を1翻訳単位とみなします。

- ・ クラス定義

- ・ プログラム定義
- ・ PROTOTYPE宣言により分離されたメソッド定義

また、リンク単位とは、リンク処理によって1つの実行可能ファイルまたはDLLを作成する単位を指します。

1つの実行可能ファイルまたはDLLを構成する要素は、複数のオブジェクトファイルであってもかまわないため、翻訳単位とリンク単位は一致しないこともあります。

翻訳単位とリンク単位の下図に示します。

図17.1 翻訳単位とリンク単位が一致する場合

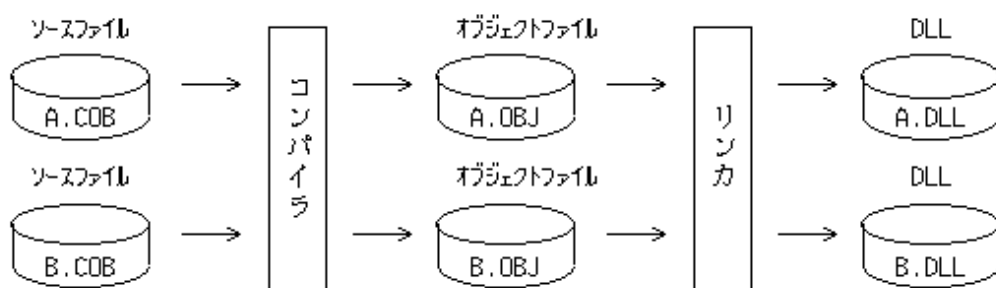
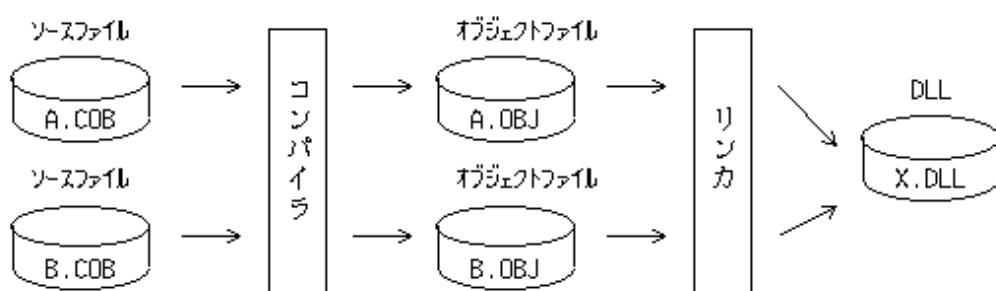


図17.2 翻訳単位とリンク単位が一致しない場合



## 17.4.2.2 プログラム構造の概要

“4.3 プログラム構造”で説明したように、リンクのプログラム構造には以下の2種類があります。

- ・ 静的構造
- ・ 動的構造

オブジェクト指向プログラムを静的構造または動的構造でリンクする場合について、以下に説明します。

### 静的構造

静的構造には、単純構造があります。

静的構造は、同一のライブラリ中でリンク関係を解決するために、リンク時にリンク先のインポートライブラリは必要ありませんが、オブジェクトファイルは必要になります。

静的構造は、複数のオブジェクトファイルで構成されるため、実行可能ファイルまたはDLLのサイズが大きくなります。

また、汎用性のあるプログラムまたはクラスのオブジェクトファイルを作成した場合に、それを使用するすべての実行可能ファイルまたはDLLに組み込む必要があります。

そのため、リンク関係が少数のパターンに特定されるような、以下の場合に適しています。

- ・ PROTOTYPE宣言したクラス定義と、それによって分離されたメソッド定義
- ・ 特定のプログラムまたはクラスからしか利用されないようなプログラムとその呼出し元

## 動的構造

動的構造には、動的リンク構造と動的プログラム構造があります。

ここでは、動的リンク構造について説明します。

動的プログラム構造については、“[18.2 動的プログラム構造](#)”を参照してください。

動的リンク構造は、リンク関係が複数のライブラリ(実行可能ファイルまたはDLL)にまたがった形で解決されるので、リンク時にオブジェクトファイルは必要ありませんが、インポートライブラリは必要になります。

動的構造は、単一または少数のオブジェクトファイルで構成されるため、実行可能ファイルまたはDLLの大きさは小さくて済みます。

さらに、静的構造と異なり、ソースファイルを修正して再翻訳を行った場合も、対象のDLLをリンクするだけです(他のライブラリに影響を与えません)。

そのため、汎用的でリンク関係が多彩な、以下の場合に適しています。

- ・ クラス定義とそのクラスを使用するクラス/プログラム/メソッド定義
- ・ 汎用性の高いプログラムとその呼出し元

プロジェクト中のオブジェクトファイルを、どのようなプログラム構造でリンクするかは、利用者の自由です。

ここでの説明を参考にして、プログラム構造を設計してください。

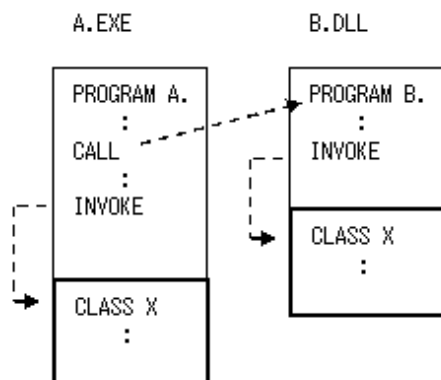
なお、オブジェクト指向プログラム開発での、プログラム/クラス/メソッドの各定義間で発生するリンク関係については、“[17.4.4.1 インポートライブラリ](#)”を参照してください。



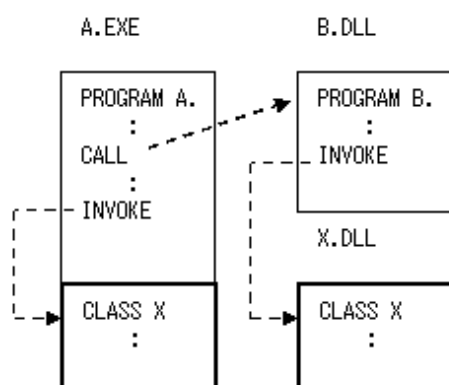
## 注意

下図の1.または2.のように、1実行単位に同一クラスのオブジェクトファイルが複数存在してはいけません。3.のような結合形態で使用してください。(図中の点線は、呼出し/参照関係を表します)

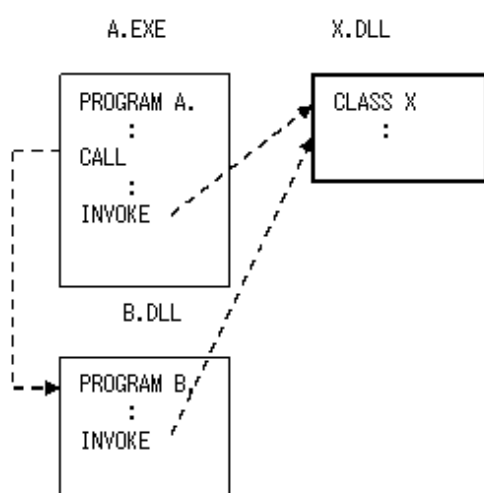
1. 静的構造により、同一クラスのオブジェクトが2つ存在する場合



2. 静的構造と動的構造により、同一クラスのオブジェクトが2つ存在する場合



3. クラスのオブジェクトファイルを動的構造に修正

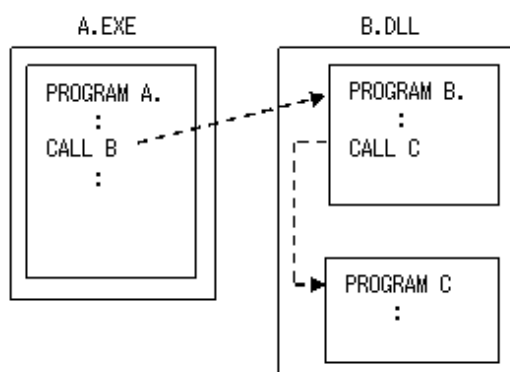


## 参考

ここで説明している静的構造とは、呼び出す定義のオブジェクトファイルと呼び出される定義のオブジェクトファイルの間のリンク関係が静的に決まることを指します。

そのため、同一のDLLに含まれていれば、この両者の間の関係は静的構造になります。

たとえば、下図のような場合、プログラムAとプログラムBの間は動的構造になりますが、プログラムBとプログラムCの間は静的構造になります。



## 17.4.3 翻訳処理

ここでは、プロジェクト管理機能を使用してオブジェクト指向プログラム開発を行う場合の、翻訳処理について説明します。

### 17.4.3.1 リポジトリファイル

ここでは、リポジトリファイルについて説明します。

#### 概要

リポジトリファイルとは、クラス定義を翻訳したときに生成される、クラスの情報を格納したファイルです。リポジトリファイルは、翻訳時に再利用するクラスの情報をコンパイラに通知するために使用します。

リポジトリファイルのファイル名は、“クラス外部名.REP”になります。

図17.3 リポジトリファイルの出力

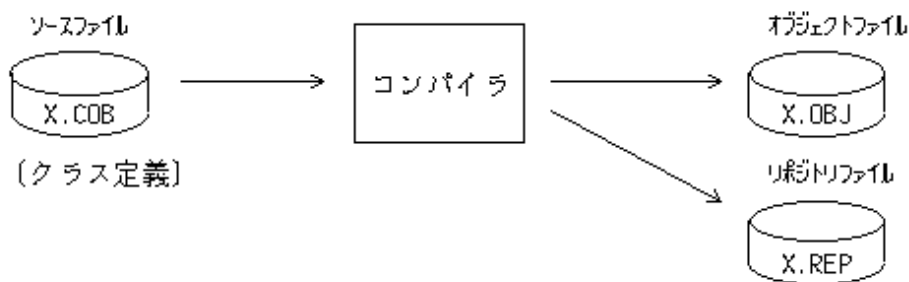
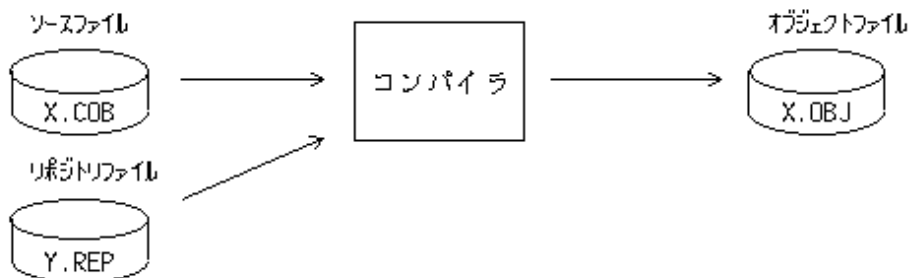


図17.4 リポジトリファイルの入力



翻訳時にコンパイラが入力するリポジトリファイルは、リポジトリ段落にクラス名が記述されたものだけです。

リポジトリ段落に記述されていなければならないリポジトリファイル(クラス名)は以下のとおりです。

- ・ 継承を利用する場合の直接の親クラス
- ・ オブジェクト参照データ項目で指定したクラス
- ・ 分離されたメソッドで、自メソッドのPROTOTYPE宣言が含まれるクラス



#### 参考

リポジトリファイルは、クラス情報データベースにクラス情報を通知する場合にも使用されます。

詳細については、“[17.3.1 クラスブラウザ](#)”を参照してください。

## 継承を利用する場合の直接の親クラス

例題15の“MEMBER.COB”を例にとってみます。

```
 :
CLASS-ID. Member-class INHERITS AllMember-class.
ENVIRONMENT DIVISION. [a]
CONFIGURATION SECTION.
REPOSITORY.
 CLASS AllMember-class.
 : [b]
```

このプログラムでは、[a]が継承する直接の親クラス名になります。

このプログラムは、AllMember-classクラスを継承しているので、翻訳時にリポジトリファイルAllMember-class.REPが必要になります。

そのため、そのクラス名はリポジトリ段落に記述されていなければなりません(プログラム中の[b])。

このクラス名を元に、コンパイラは対応するリポジトリファイルを検索します。

## オブジェクト参照変数で指定したクラス

“ALLMEM.COB”を例にとってみます。

```
 :
CLASS-ID. AllMember-class INHERITS FJBASE.
 :
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 :
 CLASS Address-class.
 : [b]
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
 :
01 住所参照 OBJECT REFERENCE Address-class ...
 : [a]
```

このプログラムでは、[a]がオブジェクト参照変数で指定されているクラス名になります。

このプログラムは、Address-classクラスを参照しているので、翻訳時にリポジトリファイルAddress-class.REPが必要になります。

この場合にも、コンパイラがリポジトリファイルを検索するために、クラス名をリポジトリ段落に記述しなければなりません(プログラム中の[b])。

## 分離されたメソッドを作成する場合のメソッド定義されているクラス

“SALA\_MEM.COB”を例にとってみます。

```
METHOD-ID. Salary-method OF Member-class.
ENVIRONMENT DIVISION. [a]
CONFIGURATION SECTION.
REPOSITORY.
 CLASS Member-class.
 : [b]
```

このプログラムでは、[a]がメソッドの所属しているクラス名になります。

このプログラムは、Member-classクラスの情報を参照するので、翻訳時にMember-class.REPというリポジトリファイルが必要になります。

この場合も、コンパイラが必要なリポジトリファイルを検索するために、クラス名をリポジトリ段落に記述する必要があります(プログラム中の[b])。



## 翻訳の手順

COBOLソースファイルを翻訳するときに、コンパイラは参照する必要があるリポジトリファイル(リポジトリ段落に記述されたクラスのリポジトリファイル)を検索します。このとき、必要なリポジトリファイルが存在しないと翻訳エラーとなります。

また、クラス定義を再翻訳すると、リポジトリファイルも更新されます。その場合、更新されたリポジトリファイルを翻訳時に入力しているソースファイルも、再度翻訳しなければなりません。

このように、リポジトリファイルの入力の関係により、翻訳順序に制約が生じます。

例題15の“MEMBER.COB”は、以下のように記述されています。

```
 :
CLASS-ID. Member-class INHERITS AllMember-class.
 :
REPOSITORY.
 CLASS AllMember-class.
 :
```

このプログラムの内容から、翻訳時にはAllMember-classクラスのリポジトリファイルが必要ことがわかります。

“ALLMEM.COB”のソースプログラムは、以下のように記述されています。

```
 :
CLASS-ID. AllMember-class INHERITS FJBASE.
 :
REPOSITORY.
 CLASS FJBASE
 CLASS Address-class.
 :
```

このプログラムの内容から、翻訳時にはFJBASEクラスおよびAddress-classクラスのリポジトリファイルが必要ことがわかります。

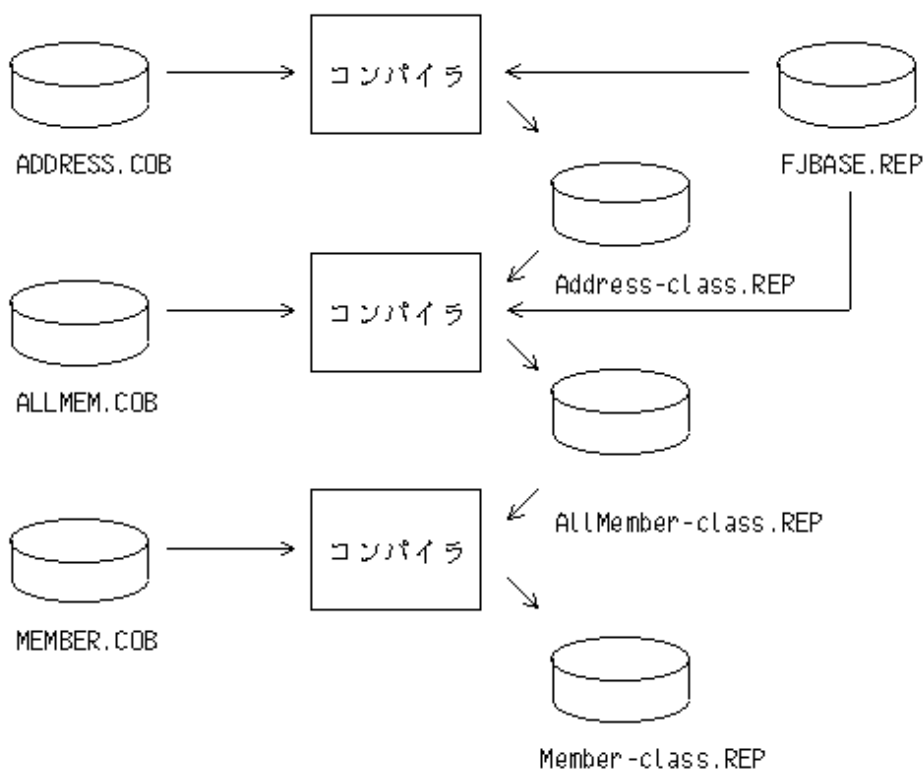
“ADDRESS.COB”のソースプログラムは、以下のように記述されています。

```
 :
CLASS-ID. Address-class INHERITS FJBASE.
 :
REPOSITORY.
 CLASS FJBASE.
 :
```

このプログラムの内容から、翻訳時にはFJBASEクラスのリポジトリファイルが必要ことがわかります。

以上を整理すると、資源の関係は以下ようになります。

図17.5 翻訳順序



“[図17.5 翻訳順序](#)”からもわかるように、以下の順序で翻訳処理が行われる必要があります。

1. ADDRESS.COB
2. ALLMEM.COB
3. MEMBER.COB

プロジェクトマネージャは、ターゲットリポジトリファイルおよび依存リポジトリファイルを指定することによって、自動的に正しい順序で翻訳処理を行います。

## ターゲットリポジトリファイル

ターゲットリポジトリファイルとは、クラス定義を翻訳した結果生成されるリポジトリファイルを指します。

“[図17.5 翻訳順序](#)”の“MEMBER.COB”のターゲットリポジトリファイルは、“Member-class.REP”になります。

## ターゲットリポジトリファイル生成パス

クラス定義を翻訳したときにターゲットリポジトリファイルが生成されるパスは、翻訳オプションREPの指定によって、下表のように決まります。[参照]“[A.2.41 REP \(リポジトリファイルの入出力先フォルダの指定\)](#)”

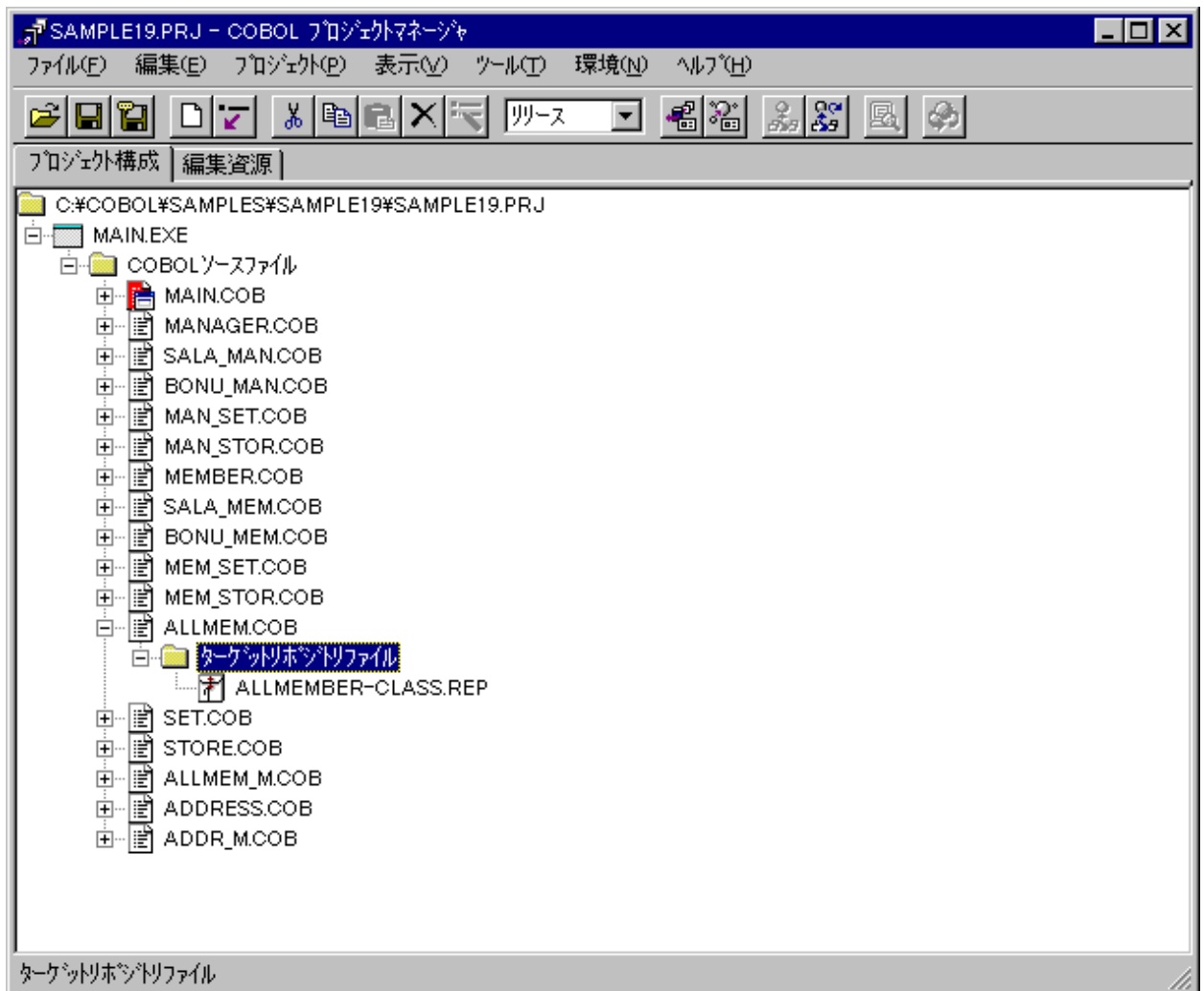
| REP オペランド | パス名                  |
|-----------|----------------------|
| あり        | REP オペランドで指定したパス     |
| なし        | COBOL ソースファイルが存在するパス |

## ターゲットリポジトリファイルの登録手順

プロジェクトマネージャは、ターゲットリポジトリファイルおよび依存リポジトリファイルの関係により、翻訳順序を決定します。

ターゲットリポジトリファイルをプロジェクトに登録するための手順を、以下に説明します。

1. ターゲットリポジトリファイルを登録したいソースファイルを、プロジェクトのツリー上で選択し、“ターゲットリポジトリファイル”フォルダを作成します。
2. 1.で作成したフォルダを選択し、[編集]メニューの“新規作成”を指定してターゲットリポジトリファイル名を入力します。



## 依存リポジトリファイル

依存リポジトリファイルとは、ソースファイルを翻訳するときに必要になるリポジトリファイルを指します。

“[図17.5 翻訳順序](#)”の“MEMBER.COB”の依存リポジトリファイルは、“AllMember-class.REP”になります。

## 依存リポジトリファイル検索パス

クラス/メソッド/プログラム定義のソースファイルを翻訳する場合、依存リポジトリファイルが検索されるパスは、翻訳オプションREPINおよびREPの指定によって、下表のように順序付けられます。[参照]“[A.2.42 REPIN \(リポジトリファイルの入力先フォルダの指定\)](#)”、“[A.2.41 REP \(リポジトリファイルの入出力先フォルダの指定\)](#)”

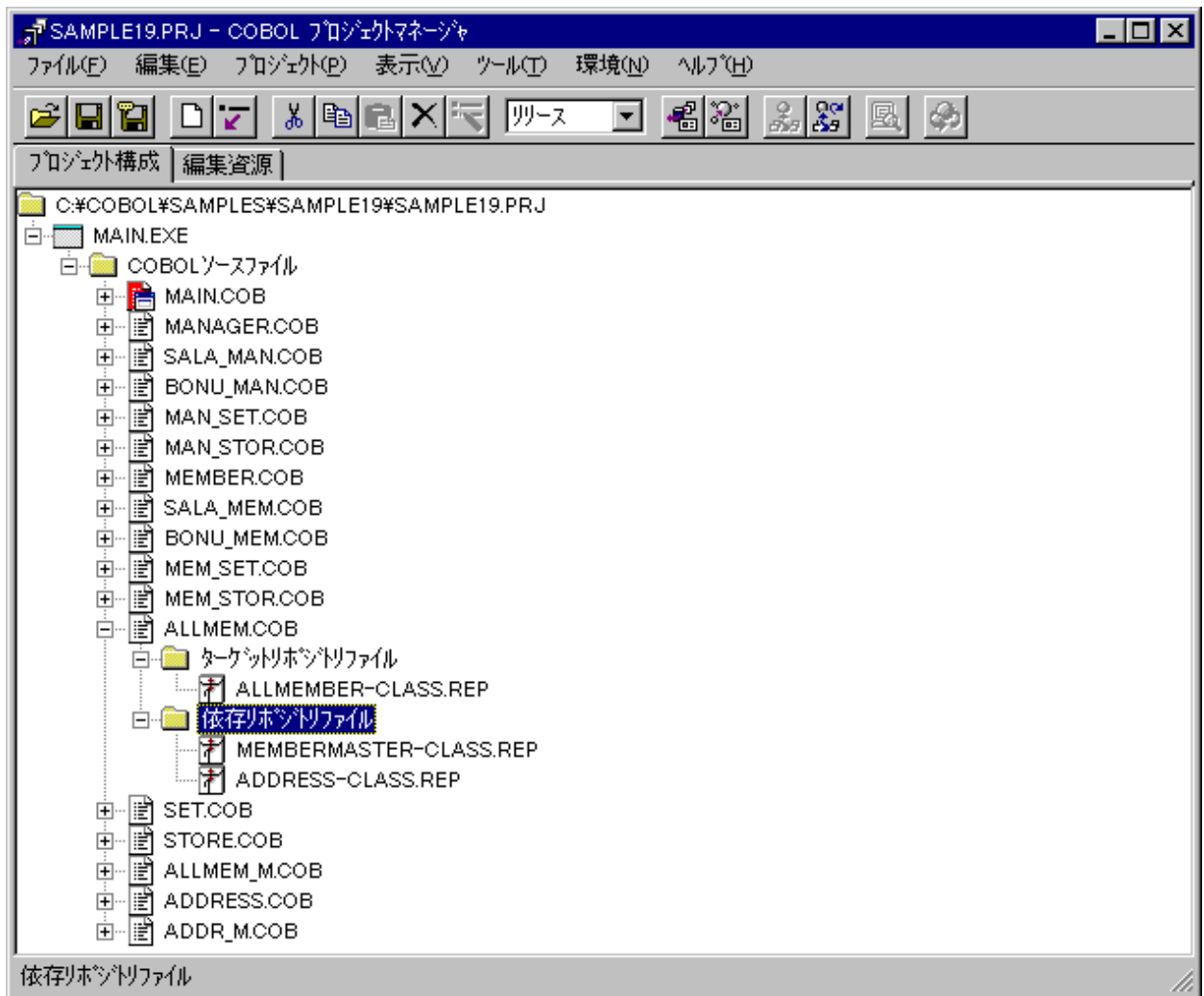
| REPIN オペランド | REP オペランド | 検索パスの順序                                        |
|-------------|-----------|------------------------------------------------|
| あり          | あり        | (1) REPIN オペランドで指定したパス<br>(2) REP オペランドで指定したパス |
|             | なし        | (1) REPIN オペランドで指定したパス<br>(2) プロジェクトファイルのあるパス  |
| なし          | あり        | (1) REP オペランドで指定したパス<br>(2) プロジェクトファイルのあるパス    |

| REPIN オペランド | REP オペランド | 検索パスの順序             |
|-------------|-----------|---------------------|
|             | なし        | (1) プロジェクトファイルのあるパス |

## 依存リポジトリファイルの登録手順

依存リポジトリファイルをプロジェクトに登録するための手順を、以下に説明します。

1. 依存リポジトリファイルを登録したいソースファイルを、プロジェクトのツリー上で選択し、“依存リポジトリファイル”フォルダを作成します。
2. 1.で作成したフォルダを選択し、[編集]メニューの“新規作成”を指定して依存リポジトリファイル名を入力します。または、[編集]メニューの“追加”を指定して、コモンダイアログから依存リポジトリファイルを選択します。



## リポジトリ検索

プロジェクトマネージャには、プロジェクトの依存関係上に自動的にターゲットリポジトリファイルと依存リポジトリファイルを登録する機能があります。この機能をリポジトリ検索と呼びます。

リポジトリ検索機能は、ソースファイル内のクラス名段落およびリポジトリ段落を解析することにより実現されます。

リポジトリ検索機能を使用する場合は、[編集]-[リポジトリファイル検索]メニューから項目を選択します。

選択した項目と、選択時に実現される機能の関係は下表のようになります。

| 選択した項目 | ターゲットリポジトリ検索          | 依存リポジトリ検索             |
|--------|-----------------------|-----------------------|
| すべて    | <input type="radio"/> | <input type="radio"/> |

| 選択した項目  | ターゲットリポジトリ検索 | 依存リポジトリ検索 |
|---------|--------------|-----------|
| ターゲットだけ | ○            | ×         |
| 依存だけ    | ×            | ○         |

## ターゲットリポジトリ検索

ターゲットリポジトリ検索は、ターゲットリポジトリファイルを対象としたリポジトリ検索機能です。

### ターゲットリポジトリ検索の手順

ターゲットリポジトリファイルを検索する手順を以下に説明します。

1. ターゲットリポジトリファイルを検索するファイルまたはフォルダを選択します。
2. [編集]-[リポジトリファイル検索]メニューから、“すべて”または“ターゲットだけ”を選択します。  
→ 検索対象ファイル配下のターゲットリポジトリフォルダの下に、ターゲットリポジトリファイル名が登録されます。
3. ターゲットリポジトリフォルダが存在しない場合は、フォルダを自動的に作成します。



### 注意

- ・ 選択可能なファイル/フォルダ、および選択した場合の検索対象COBOLソースファイルは、下表のようになります。

| 選択アイテム種別          | 検出の対象となるCOBOL ソースファイル          |
|-------------------|--------------------------------|
| プロジェクトファイル        | プロジェクトに登録しているすべてのCOBOL ソースファイル |
| 最終ターゲットファイル       | 指定したファイル配下のすべてのCOBOL ソースファイル   |
| COBOL ソースファイルフォルダ | 指定したフォルダ配下のすべてのCOBOL ソースファイル   |
| COBOL ソースファイル     | 指定したCOBOL ソースファイル              |

- ・ 登録されるターゲットリポジトリファイルのファイル名およびパス名は、以下のように決まります。
  - ー ファイル名  
COBOLソースプログラムのクラス名段落に記述されているクラス外部名.REPになります。
  - ー パス名  
翻訳オプションREP指定の有無によって下表のようになります。[参照]“[A.2.41 REP \(リポジトリファイルの入出力先フォルダの指定\)](#)”

| REP オペランド | パス名                  |
|-----------|----------------------|
| あり        | REP オペランドで指定したパス     |
| なし        | COBOL ソースファイルが存在するパス |

## 依存リポジトリ検索

依存リポジトリ検索とは、依存リポジトリファイルを対象にしたリポジトリ検索機能です。

### 依存リポジトリ検索の手順

依存リポジトリファイルを検索する手順を以下に説明します。

1. 依存リポジトリファイルを検出するファイルまたはフォルダを選択します。
2. [編集]-[リポジトリファイル検索]メニューから、“すべて”または“依存だけ”を選択します。  
→ 検索対象ファイル配下の依存リポジトリフォルダの下に、依存リポジトリファイル名が登録されます。
3. 依存リポジトリフォルダが存在しない場合は、フォルダを自動的に作成します。



## 注意

- 選択可能なファイル/フォルダおよび、選択した場合の検索対象COBOLソースファイルは下表のようになります。

| 選択アイテム種別          | 処理の対象となるCOBOL ソースファイル          |
|-------------------|--------------------------------|
| プロジェクトファイル        | プロジェクトに登録しているすべてのCOBOL ソースファイル |
| 最終ターゲットファイル       | 指定したファイル配下のすべてのCOBOL ソースファイル   |
| COBOL ソースファイルフォルダ | 指定したフォルダ配下のすべてのCOBOL ソースファイル   |
| COBOL ソースファイル     | 指定したCOBOL ソースファイル              |

- 登録されるリポジトリファイルのファイル名およびパス名は、以下のようになります。

- ファイル名

COBOLソースプログラムのリポジトリ段落に記述されているクラス外部名.REPになります。

- パス名

依存リポジトリファイルがあらかじめ存在している場合は、以下の順序で検索されます。

1. 翻訳オプションREPINで指定したパス
2. 翻訳オプションREPで指定したパス
3. プロジェクトファイルが存在するパス

依存リポジトリファイルが存在していない場合は、翻訳オプションREPINおよびREPの指定により下表のようになります。[参照]  
[“A.2.42 REPIN \(リポジトリファイルの入力先フォルダの指定\)”](#)、[“A.2.41 REP \(リポジトリファイルの入出力先フォルダの指定\)”](#)

| REPIN オペランド | REP オペランド | 検索パス               |
|-------------|-----------|--------------------|
| あり          | —         | REPIN オペランドで指定したパス |
| なし          | あり        | REP オペランドで指定したパス   |
|             | なし        | プロジェクトファイルが存在するパス  |

## 17.4.4 リンク処理

ここでは、プロジェクト管理機能を使用してオブジェクト指向プログラミングを行う場合の、リンク処理について説明します。

### 17.4.4.1 インポートライブラリ

ここでは、オブジェクト指向プログラミングを行う場合の、インポートライブラリの扱いおよびリンク処理への影響について説明します。

#### リンク時に必要なインポートライブラリ

COBOL85の言語仕様の範囲では、静的または動的にリンク(結び付け)を行う必要がある関係は、呼出し(CALL文)関係を持ったプログラムだけでした。しかし、オブジェクト指向プログラミングでは、クラス/プログラム/メソッド各定義間でリンク関係が発生するパターンが増えました。

オブジェクト指向プログラミングでリンクが必要となる場合を下表に示します。

表17.2 オブジェクト指向プログラミングでのリンク関係

| 定義名  |         | リンク先                                 |                     |                            |
|------|---------|--------------------------------------|---------------------|----------------------------|
|      |         | クラス定義                                | プログラム定義             | メソッド定義                     |
| リンク元 | クラス定義   | [1] リポジトリ段落に記述したクラス<br>[2] 親クラス (注1) | [3] CALL文で呼び出すプログラム | [4] PROTOTYPE 宣言で分離されたメソッド |
|      | プログラム定義 | [5] リポジトリ段落に記述したクラス                  | [6] CALL文で呼び出すプログラム | —                          |

| 定義名 |        | リンク先                                                                                            |                     |        |
|-----|--------|-------------------------------------------------------------------------------------------------|---------------------|--------|
|     |        | クラス定義                                                                                           | プログラム定義             | メソッド定義 |
|     | メソッド定義 | [7] リポジトリ段落に記述したクラス(ただし、自メソッドのPROTOTYPE宣言を含むクラスは除く)<br>[8] 自メソッドのPROTOTYPE宣言を含むクラスの親クラス(注1)(注2) | [9] CALL文で呼び出すプログラム | —      |

注1:直接の親クラスを含んだ、継承関係での上位のクラスを指します。

注2:手続き部の中にSUPERが指定されている場合だけです。

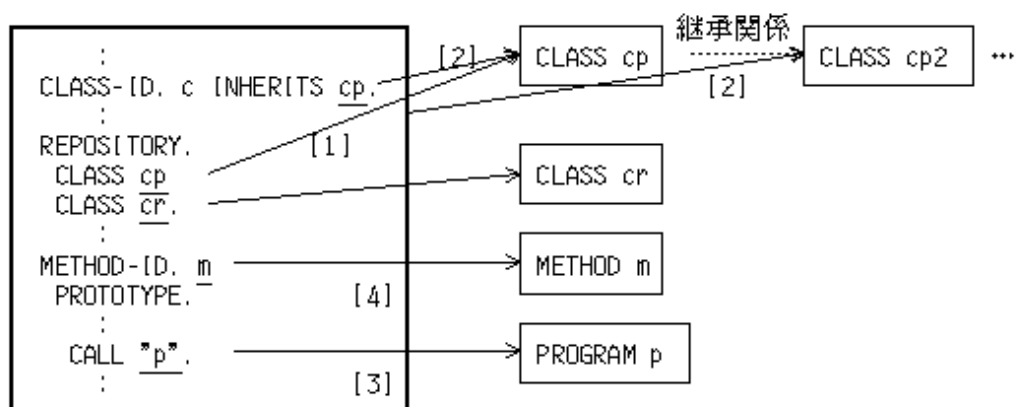
各定義間のリンク関係を下図に示します。

なお、図中の実線矢印がリンクを表します。

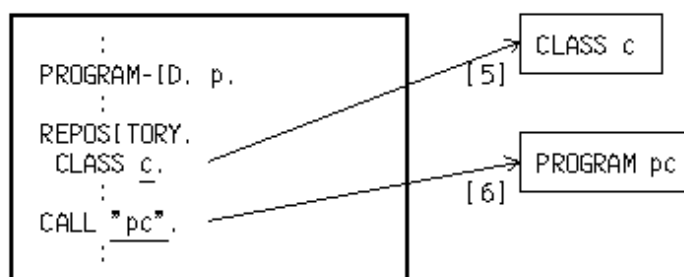
実線矢印で示されたリンク関係を、動的リンク構造で解決する場合にはインポートライブラリが必要になります。

#### 各定義間のリンク

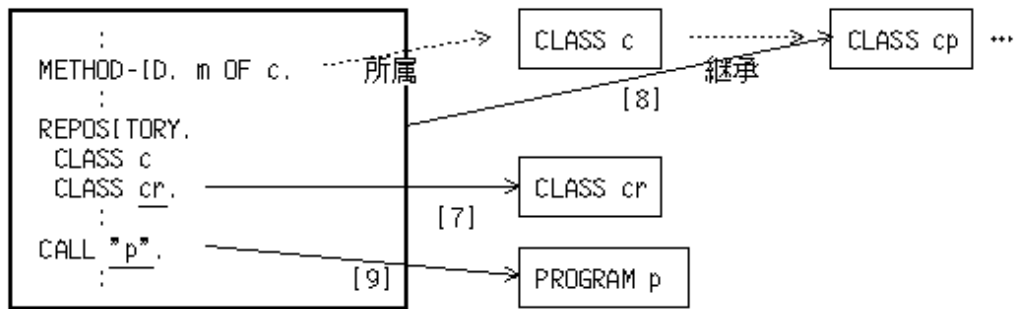
##### クラス定義のリンク



##### プログラム定義のリンク



## メソッド定義のリンク



## リンクの手順

インポートライブラリは、リンカを使用してDLLを作成するときに同時に作成されます。また、リンカを使用して実行可能ファイルまたはDLLを作成するときに、リンクするDLLのインポートライブラリが必要になります。そのため、複数の実行可能ファイルまたはDLLを作成する場合、リンク処理の順序に制約が発生します。

例題プログラムの“MEMBER.COB”、“ALLMEM.COB”および“ADDRESS.COB”を例にとります。

### • MEMBER.COB

```

:
CLASS-ID. Member-class INHERITS AllMember-class.
:
REPOSITORY.
 CLASS AllMember-class.
:

```

### • ALLMEM.COB

```

:
CLASS-ID. AllMember-class ...
:
REPOSITORY.
:
 CLASS Address-class.
:

```

### • ADDRESS.COB

```

:
CLASS-ID. Address-class ...
:

```

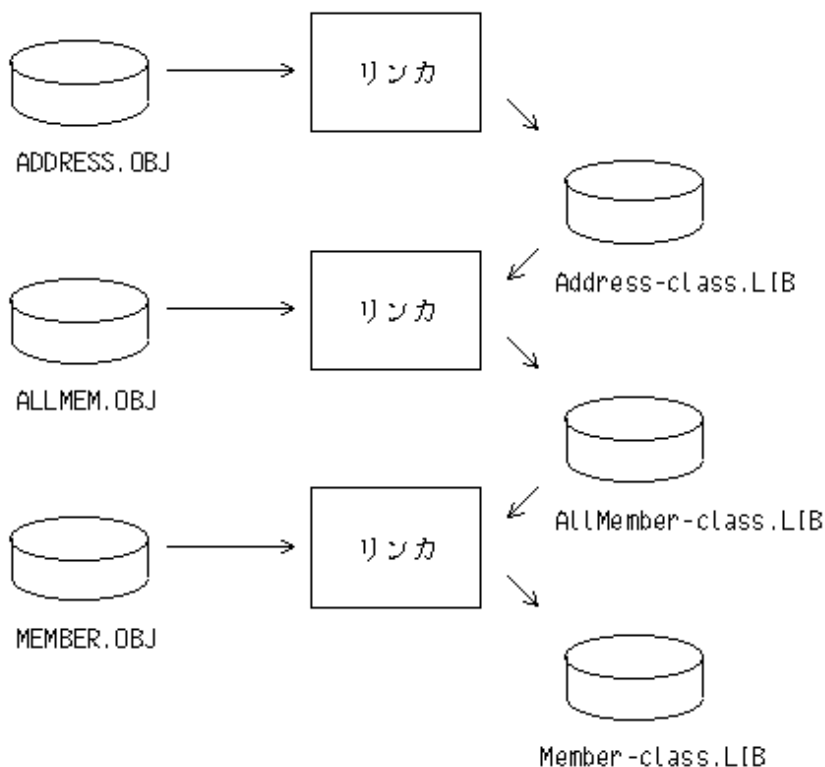
翻訳単位とリンク単位を同じにした場合、それぞれのリンク処理で必要なインポートライブラリおよび生成されるインポートライブラリは下表のようになります。

| ソースファイル／<br>オブジェクトファイル名   | 生成するインポートライブラリ      | 使用するインポートライブラリ      |
|---------------------------|---------------------|---------------------|
| MEMBER.COB / MEMBER.OBJ   | Member-class.LIB    | AllMember-class.LIB |
| ALLMEM.COB / ALLMEM.OBJ   | AllMember-class.LIB | Address-class.LIB   |
| ADDRESS.COB / ADDRESS.OBJ | Address-class.LIB   | —                   |

この関係を“[図17.6 インポートライブラリとリンク処理の順序](#)”に示します。



図17.6 インポートライブラリとリンク処理の順序



## 参考

FJBASEクラスのインポートライブラリは、利用者が指定しなくても、自動的にリンクされます。

“[図17.6 インポートライブラリとリンク処理の順序](#)”からわかるように、以下の順序でリンク処理が行われる必要があります。

1. ADDRESS.OBJ
2. ALLMEM.OBJ
3. MEMBER.OBJ

プロジェクトにインポートライブラリを指定することによって、プロジェクトマネージャは、自動的に正しい順序でリンク処理を実行します。

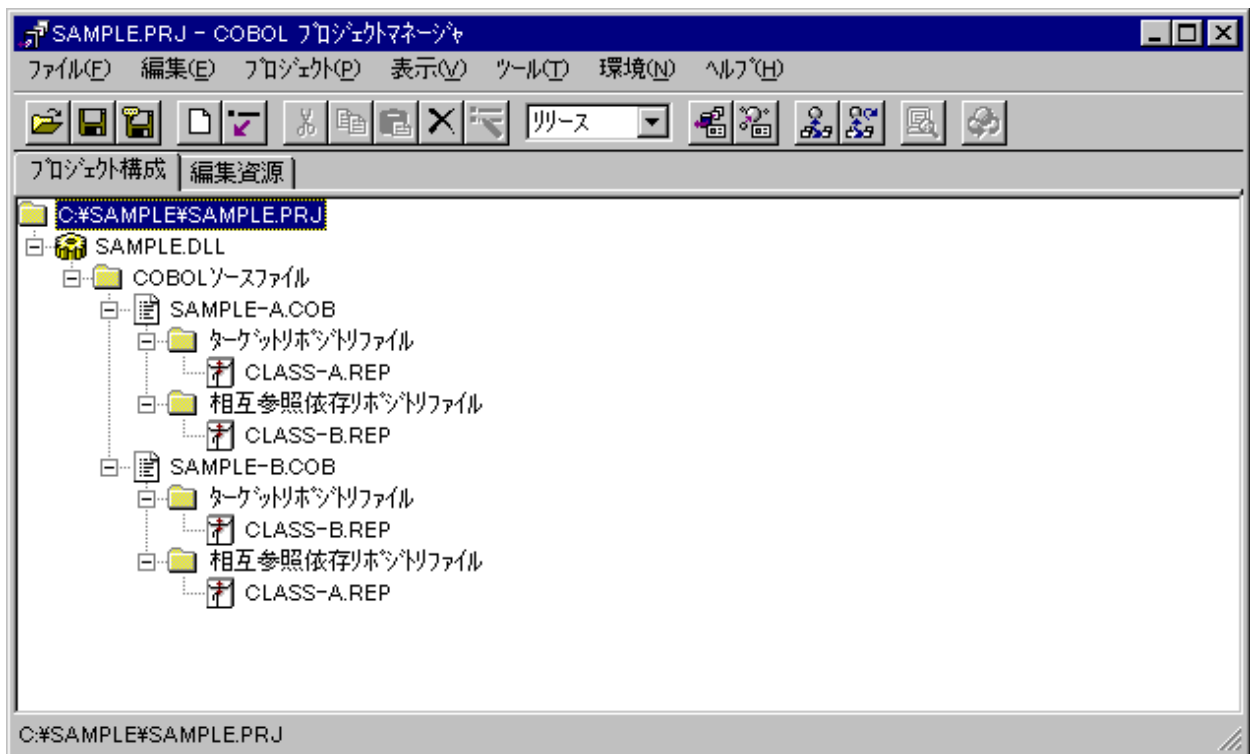
## 17.4.5 相互参照クラス

ここでは、プロジェクト管理を使用して相互参照クラスをビルドする方法について説明します。

なお、相互参照クラスの詳細については、“[16.7.8 相互参照クラス](#)”を参照してください。

### 17.4.5.1 相互参照クラスのプロジェクト管理

相互参照クラスを含むプロジェクトは、下図のようになります。



相互参照クラスを構築するには、以下の手順でプロジェクトを作成します。

1. 最終ターゲットファイル、COBOLソースファイル、登録集ファイルおよびその他の各種ファイルをプロジェクトへ登録します。  
詳細については、“[第6章 プロジェクト管理機能](#)”を参照してください。
2. プロジェクトに登録した各COBOLソースファイルに、“ターゲットリポジトリファイル”、“依存リポジトリファイル”、“相互参照依存リポジトリファイル”の各フォルダを追加し、それぞれにリポジトリファイルを登録します。  
“ターゲットリポジトリファイル”フォルダおよび“依存リポジトリファイル”フォルダに登録するリポジトリファイルについては、“[17.4.3 翻訳処理](#)”を参照してください。  
“相互参照依存リポジトリファイル”フォルダに登録するリポジトリファイルについては、“[17.4.5.2 相互参照クラスの翻訳処理](#)”を参照してください。
3. ビルドまたはリビルドを実施します。

さらに、動的リンク構造を用いて相互参照しているクラスをそれぞれ別のDLLとして作成する場合は、以下の作業が必要になります。

1. プロジェクトツリー上のプロジェクトファイル配下に、最終ターゲットファイルとして、それぞれのDLLに指定したインポートライブラリファイル名を登録します。
2. 1で指定したインポートライブラリを選択して、“モジュール定義ファイル”フォルダを追加します。
3. 2で作成したフォルダにモジュール定義ファイルを登録します。

詳細は、“[17.4.5.3 相互参照クラスのリンク処理](#)”を参照してください。

## 17.4.5.2 相互参照クラスの翻訳処理

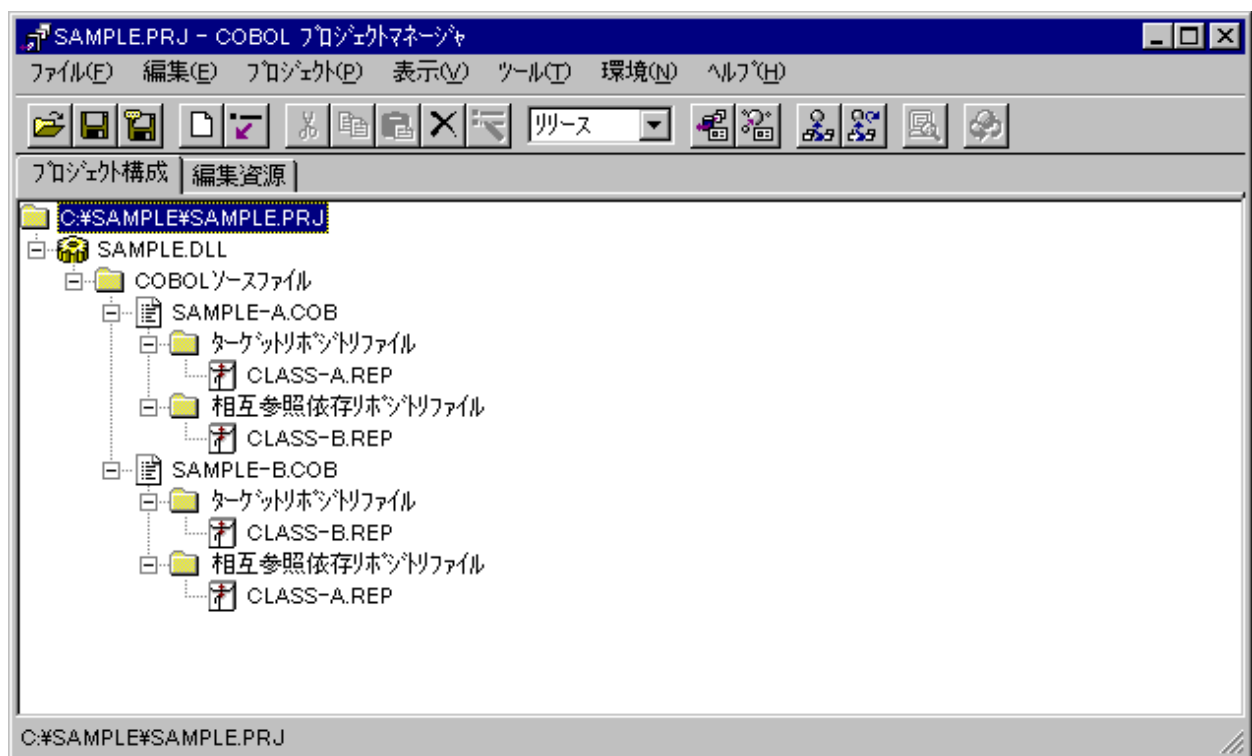
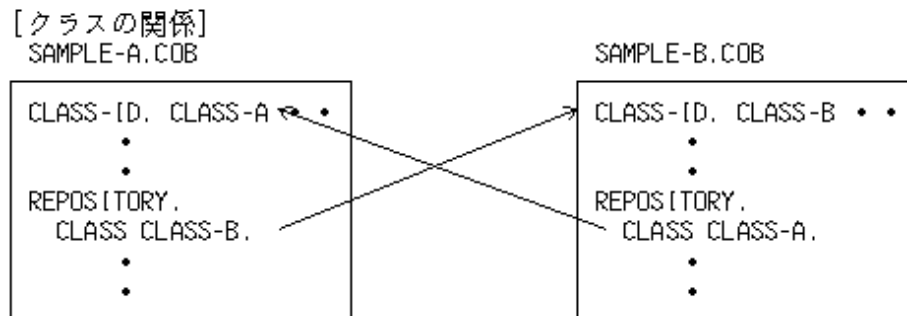
自クラス参照を除く相互参照クラスの翻訳処理では、“[16.7.8 相互参照クラス](#)”で説明したように、リポジトリファイルの依存関係に起因したデッドロック状態が発生します。

それを回避するためにプロジェクトマネージャでは、“相互参照依存リポジトリファイル”フォルダを用意しました。

### “相互参照依存リポジトリファイル”フォルダ

“相互参照依存リポジトリファイル”フォルダとは、相互参照関係にあるクラスのリポジトリファイルを登録するためのフォルダです。

“相互参照依存リポジトリファイル”フォルダと“依存リポジトリファイル”フォルダの違いは、事前に翻訳オプションCREATE(REP)を指定した翻訳処理を行う(“相互参照依存リポジトリファイル”フォルダ)か、行わない(“依存リポジトリファイル”フォルダ)かです。



1. 翻訳オプションCREATE(REP)を指定してSAMPLE-A.COBを翻訳
2. 翻訳オプションCREATE(REP)を指定してSAMPLE-B.COBを翻訳
3. SAMPLE-A.COBを翻訳
4. SAMPLE-B.COBを翻訳

プロジェクトマネージャでは、相互参照関係にあるどのクラスが修正されても、的確な翻訳順序が決定できなければなりません。そのためには、相互参照関係にあるクラス定義と相互参照関係にあるクラスのリポジトリファイルは、すべてクラス定義が指定されているソースファイル配下の“相互参照依存リポジトリファイル”フォルダに登録しなければなりません。

このようなクラスの関係の場合は、まず、SAMPLE-A.COB配下に“相互参照依存リポジトリファイル”フォルダを生成してCLASS-B.REPを登録します。次に、SAMPLE-B.COBの配下にも“相互参照依存リポジトリファイル”フォルダを生成して、CLASS-A.REPを登録します。



## 注意

リポジトリ検索機能で検出された依存リポジトリファイルは、すべて“依存リポジトリファイル”フォルダ配下に登録されます。

相互参照クラスの構築を行う場合は、相互参照クラスに該当するクラスのリポジトリファイルは、すべて“相互参照依存リポジトリファイル”フォルダに登録し直してください。

“相互参照依存リポジトリファイル”フォルダに登録したリポジトリファイルは、“依存リポジトリファイル”フォルダには登録しないでください。

### 17.4.5.3 相互参照クラスのリンク処理

相互に参照しているクラスを単純構造でリンクする場合、および動的プログラム構造でリンクする場合は、リンク処理に関して特別な考慮は必要ありません。

しかし、動的リンク構造でリンクする場合には、翻訳処理時のリポジトリファイルと同様なデッドロック現象が、インポートライブラリファイルに関して発生します。

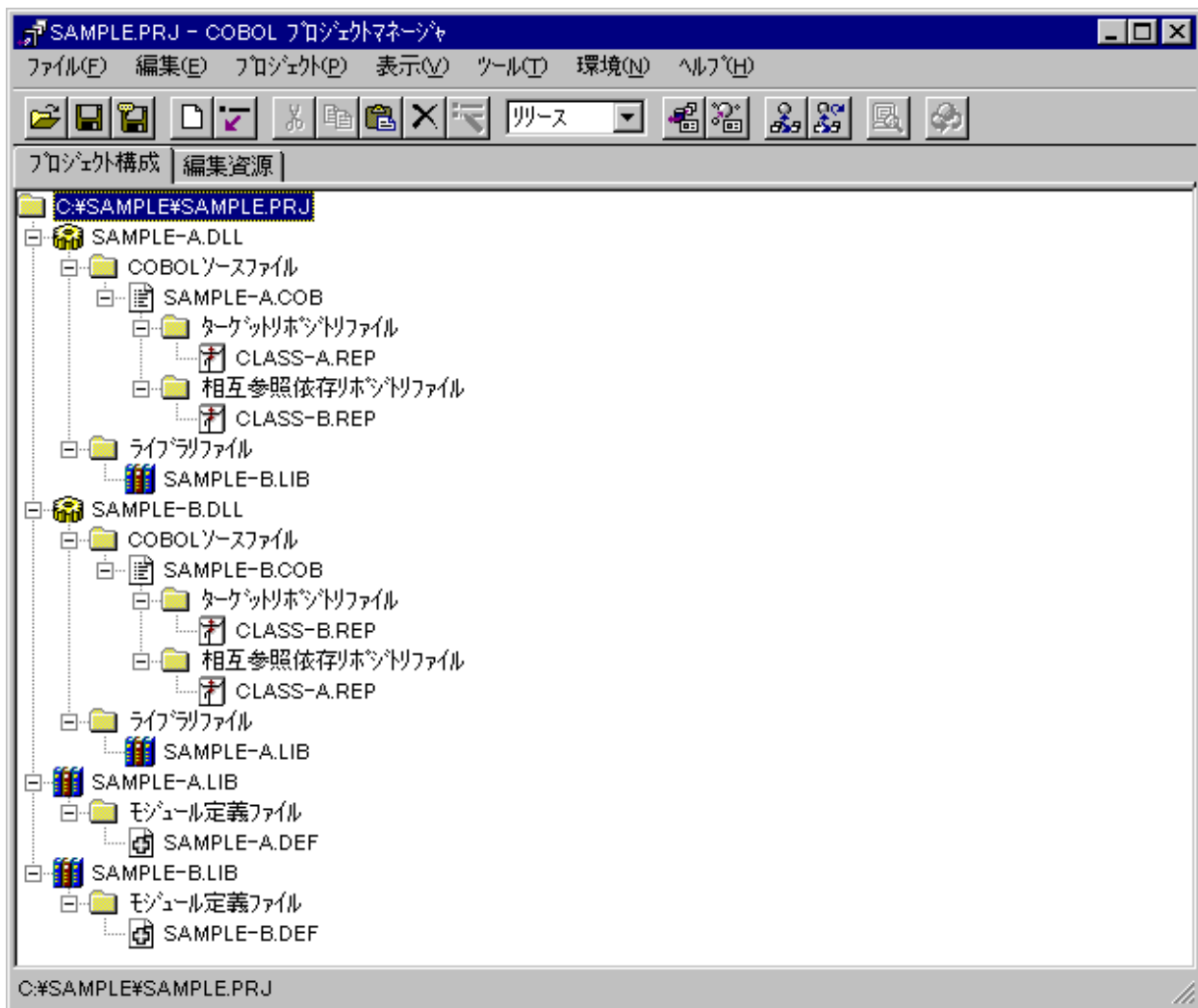
そこで、プロジェクトマネージャでは、インポートライブラリファイル生成機能を用意しました。

#### インポートライブラリファイル生成機能

インポートライブラリファイル生成機能とは、モジュール定義ファイルを指定することにより、LIBコマンドを使用して、自動的にインポートライブラリファイルだけを生成する機能です。

これにより、相互参照クラスの構築時に、リンク以外の処理でインポートライブラリファイルを生成するため、デッドロック状態を回避することができます。

“モジュール定義ファイル”フォルダを使用した場合に、プロジェクトマネージャの内部では以下のように動作します。



1. SAMPLE-A.DEFからLIBコマンドを使用して、SAMPLE-A.LIBを作成します。
2. SAMPLE-B.DEFからLIBコマンドを使用して、SAMPLE-B.LIBを作成します。
3. SAMPLE-A.LIBを指定してリンクを使用することにより、SAMPLE-B.DLLを作成します。
4. SAMPLE-B.LIBを指定してリンクを使用することにより、SAMPLE-A.DLLを作成します。

インポートライブラリファイル生成機能は、以下の手順により指定することができます。

1. プロジェクトマネージャ上のプロジェクトファイルを選択し、クラス定義を含むDLLに指定されている、相互参照するクラスのインポートライブラリを、[編集]メニューの“新規作成”または“追加”を使用して指定します。
2. 1.で作成したインポートライブラリを選択して、“モジュール定義ファイル”フォルダを作成します。
3. 2.で作成した“モジュール定義ファイル”フォルダに、モジュール定義ファイルを登録します。

モジュール定義ファイルの作成方法は、“[16.7.8.3 相互参照クラスのリンク](#)”を参照してください。



## 注意

“モジュール定義ファイル”フォルダには、各種ライブラリファイル配下に指定できるものと、最終ターゲットファイル(DLL)配下に登録できるものの2種類があります。

ここで説明しているのは前者です。

後者については、“[6.4.4.6 その他必要なファイルの登録](#)”を参照してください。

## 17.4.6 プロジェクトブラウザ

プロジェクトブラウザは、プロジェクト内に登録されている資源の論理情報を管理/表示するツールです。

ここでは、プロジェクトブラウザで使用する資源およびプロジェクトブラウザの使用方法について説明します。

### 17.4.6.1 概要

プロジェクトマネージャで管理している情報は、ファイル名またはCOBOLソースファイルとライブラリ(実行可能ファイルまたはDLL)といった物理的な情報です。しかし、物理的な情報よりも、クラス名やプログラム名といった論理的な情報の方が利用者にとっては理解しやすく、扱いやすくなります。

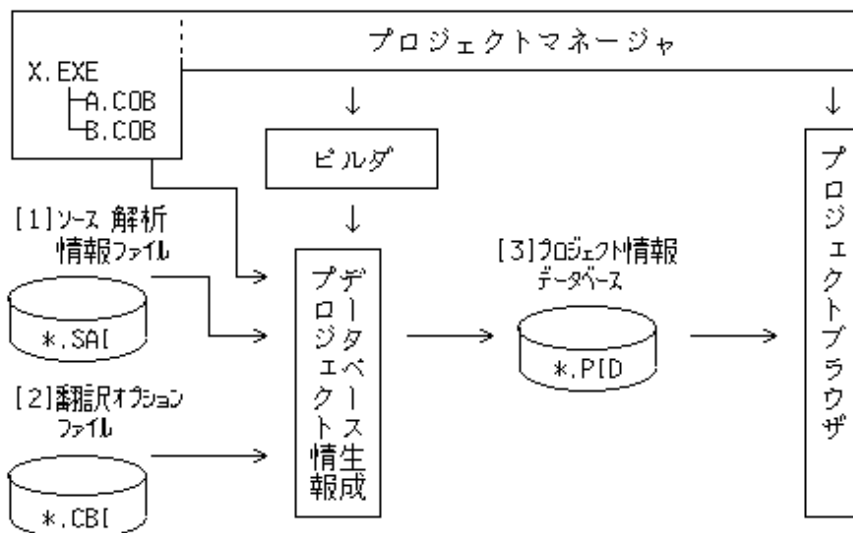
プロジェクトブラウザは、プロジェクト内の資源およびそれらの関係を論理的な情報として表示するとともに、物理的な情報へのマッピングを行います。

プロジェクトブラウザは、以下の機能を持っています。

- ・ クラス外部名および継承関係の表示
- ・ プログラム外部名の表示
- ・ クラス単位のメソッド外部名一覧の表示
- ・ クラス/プログラムのオブジェクト参照の表示
- ・ クラス/プログラムからのプログラム呼出しの表示
- ・ クラス/プログラム/メソッドのファイル名の表示およびエディタとの連携
- ・ プログラム/メソッドの入出力パラメタ情報の表示
- ・ 各種テンプレート機能

### 17.4.6.2 使用する資源

プロジェクトブラウザで使用する資源およびその流れは下図のようになります。



#### [1] ソース解析情報ファイル

ソース解析情報ファイルは、COBOLソースファイルに定義されている各種の情報が格納されているファイルです。

ソース解析情報ファイルは、翻訳オプションSAIが指定された場合に、コンパイラがソースファイル単位で作成します。その場合に、ファイル名は“ソースファイル名.SAI”になります。[参照]“[A.2.44 SAI\(ソース解析情報ファイルの出力の可否\)](#)”

プロジェクトブラウザは、ソース解析情報ファイルからクラス/メソッド/プログラムの各定義情報を取り出して、プロジェクト情報データベースを作成します。

## 注意

クラス定義の情報をプロジェクトブラウザに取り込む場合、ソース解析情報ファイルが存在しないとリポジトリファイルを使用します。ただし、リポジトリファイルを使用した場合は、以下の情報だけが表示されます。

- 継承関係
- メソッド名一覧
- メソッドのインタフェース情報

リポジトリファイルを使用する場合、プロジェクトマネージャに登録されているターゲットリポジトリファイルを使用します。

プロジェクト情報データベースを作成するときは、ターゲットリポジトリファイルを必ず登録してください。

プロジェクトブラウザでは、参照するクラスのメソッドを表示する場合、親クラスで定義されているメソッドも含めて表示することができます。ただし、親クラスのリポジトリが、以下のフォルダのどこにもない場合は表示されません。

- 翻訳オプションREPINに指定したフォルダ
- 翻訳オプションREPに指定したフォルダ
- プロジェクトファイルが存在するフォルダ
- NetCOBOLインストール先フォルダ

## [2] 翻訳オプションファイル

翻訳オプションファイルは、プロジェクト上に登録されているCOBOLソースファイルに対応したソース解析情報ファイルの格納されているパス(翻訳オプションSAIに指定したパス)を知るために、プロジェクト情報データベース生成時に参照します。

## [3] プロジェクト情報データベース

プロジェクト情報データベースは、ソース解析情報ファイルで取り出されたクラス/メソッド/プログラムの情報をまとめて管理するファイルです。

プロジェクト情報データベースは、ビルド・リビルド時に“プロジェクト名.PID”という名前で生成されます。

プロジェクトブラウザは、プロジェクト情報データベースの内容を元に表示します。

## 参考

プロジェクト情報データベースを生成するとき、プロジェクト依存情報も参照されます。

プロジェクト依存情報とは、プロジェクトマネージャに表示されているファイル依存関係のツリー情報です。

プロジェクト情報データベースを生成するときに、プロジェクト依存情報から各定義に対応したソースファイルのパス名を取得しています。

## 17.4.6.3 プロジェクト情報データベース生成

プロジェクトブラウザは、プロジェクト情報データベースを使用して情報を表示します。

プロジェクト情報データベースを作成する手順を以下に示します。

1. プロジェクトマネージャウィンドウの[ファイル]メニューから“プロパティ”を選択します。
2. [プロパティ]ダイアログの[基本設定]ページで、“プロジェクトデータベース生成”チェックボタンがチェックされていることを確認します。
3. 2.でチェックされていない場合、“プロジェクトデータベース生成”チェックボタンをクリックしてチェック状態にします。
4. プロジェクトマネージャから、“ビルド”または“リビルド”を行います。



## 注意

プロジェクトブラウザは、ソースファイルの記述内容に基づいて、参照関係、インタフェースなどの情報を表示するツールです。そのため、プロジェクトにソースファイルが登録されているものが対象となります。プロジェクトにオブジェクトファイルを登録しても、そのオブジェクトに関する情報はプロジェクトブラウザには表示されません。

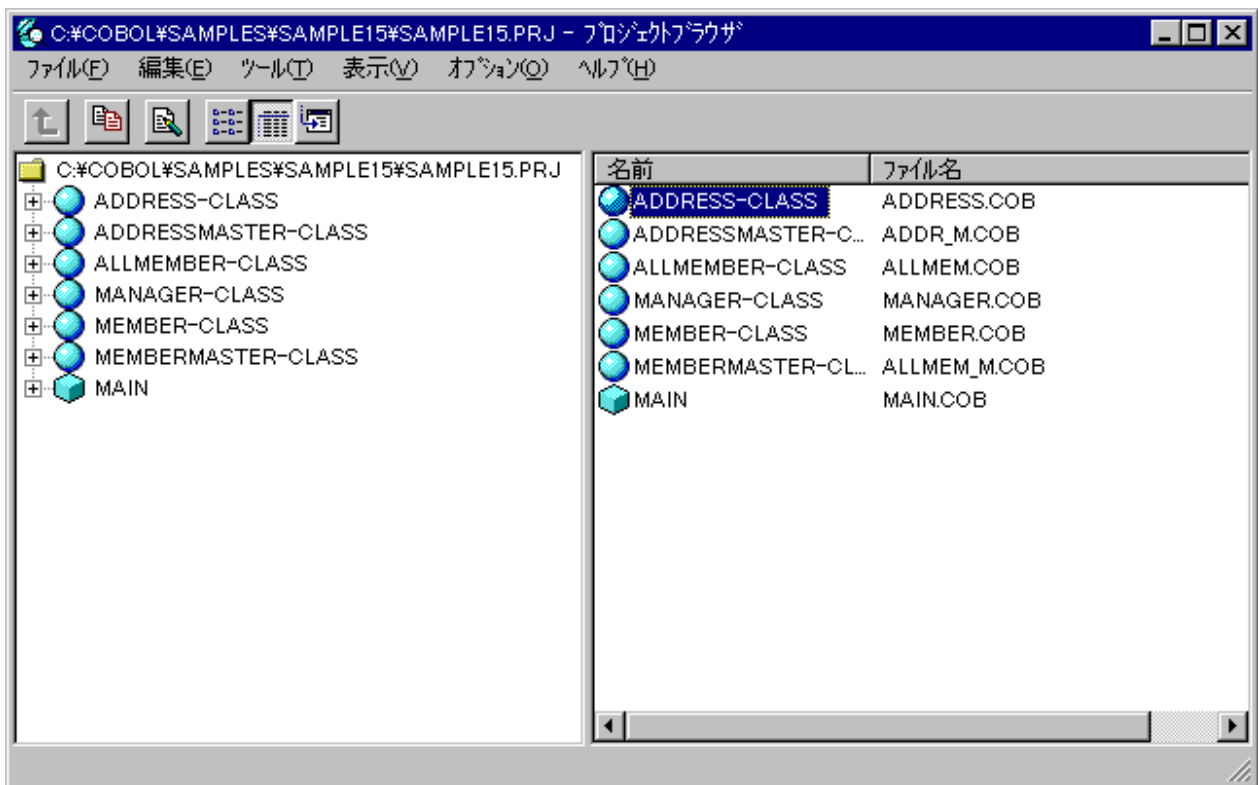
### 17.4.6.4 プロジェクトブラウザの表示/操作手順

プロジェクトブラウザの操作手順の概要を以下に示します。

1. プロジェクトマネージャの[ツール]メニューから“プロジェクトブラウザ”を選択すると、プロジェクトブラウザ画面が表示されます。
2. 各種関係をたどります。[参照]“[17.4.6.5 プロジェクトブラウザウィンドウ](#)”
3. 各メソッドおよびプログラムのインタフェースを確認します。[参照]“[17.4.6.5 プロジェクトブラウザウィンドウ](#)”
4. 各定義が記述されているCOBOLソースファイル名を確認します。[参照]“[17.4.6.5 プロジェクトブラウザウィンドウ](#)”
5. [ファイル]メニューから“閉じる”を選択して終了します。

### 17.4.6.5 プロジェクトブラウザウィンドウ

ここでは、プロジェクトブラウザを使った各操作について説明します。



#### 表示可能情報および表示方法

プロジェクトブラウザウィンドウ上で、各種の情報を確認することができます。

表示可能な情報および表示方法を“[表17.3 プロジェクトブラウザの情報表示](#)”に説明します。

表17.3 プロジェクトブラウザの情報表示

| 情報   | 定義  | 表示方法          |
|------|-----|---------------|
| 継承関係 | クラス | ツリーの1つ下の階層に表示 |

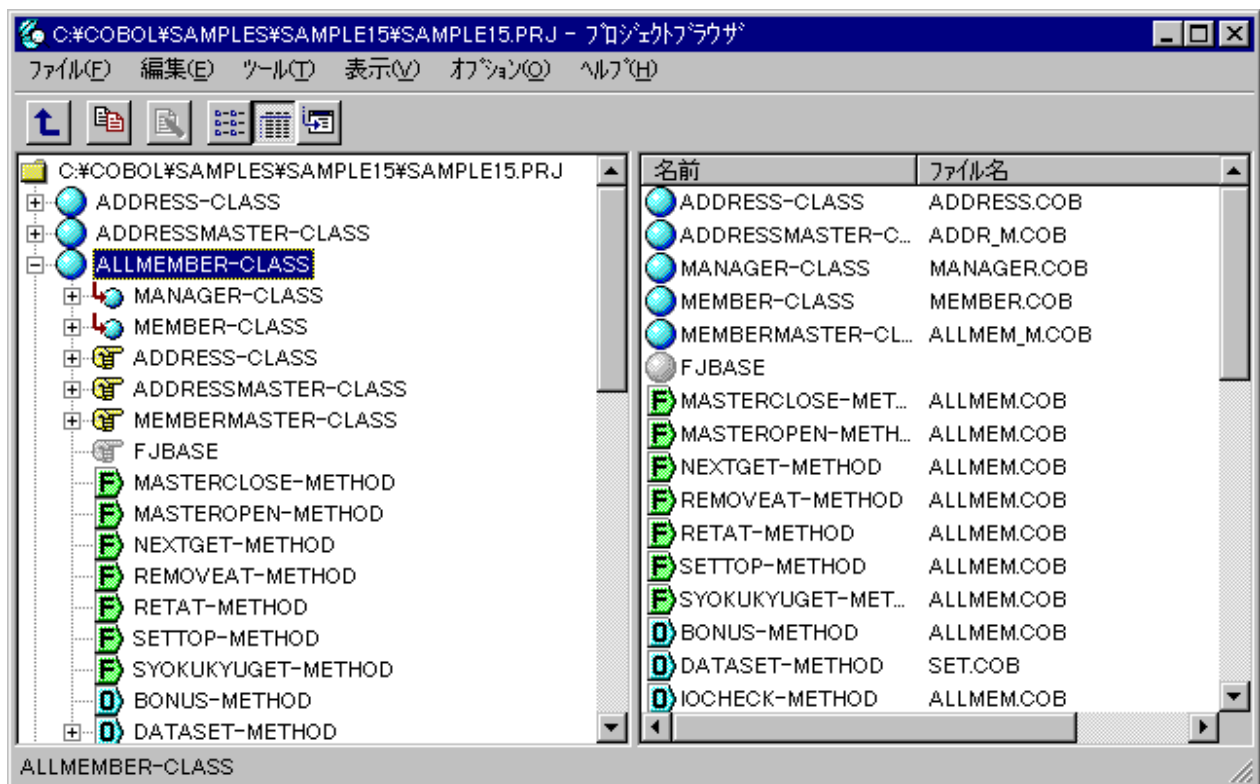


| 情報      | 定義                   | 表示方法          |
|---------|----------------------|---------------|
| 参照関係    | クラス<br>プログラム<br>メソッド | ツリーの1つ下の階層に表示 |
| 呼出し関係   | プログラム<br>メソッド        | ツリーの1つ下の階層に表示 |
| メソッド一覧  | クラス                  | ツリーの1つ下の階層に表示 |
| 二次入口点一覧 | プログラム                | ツリーの1つ下の階層に表示 |
| インタフェース | プログラム<br>メソッド        | リストビューで表示     |
| ファイル名   | クラス<br>プログラム<br>メソッド | リストビューで表示     |

## 継承の表示

プロジェクトブラウザウィンドウで、クラスの継承関係を表示する場合、以下の操作を行ってください。

- ・ クラス名のアイコンをダブルクリックします。または、アイコンの前に表示されている+記号をクリックします。



## クラス参照の表示

プロジェクトブラウザウィンドウで、クラスの参照関係を表示する場合、以下の操作を行ってください。

- ・ クラス名のアイコンをダブルクリックします。または、アイコンの前に表示されている+記号をクリックします。



プロジェクトブラウザに表示される参照関係は、クラス/メソッド/プログラム各定義のリポジトリ段落に指定されたクラスです。そのため、実際にプログラム中で参照していないクラスが表示されることもあります。

## 呼出しプログラムの表示

プロジェクトブラウザウィンドウで、プログラムの呼出し関係を表示する場合、以下の操作を行ってください。

- ・ クラス名/プログラム名のアイコンをダブルクリックします。または、アイコンの前に表示されている+記号をクリックします。

## メソッド一覧の表示

メソッド表示機能は、以下のメソッドを表示します。

- ・ クラス定義中に定義されているファクトリメソッド/オブジェクトメソッド(プロパティメソッド/プロパティデータも含む)
- ・ 継承関係により、自クラスのメソッドとして使用できる親クラスのファクトリメソッド/オブジェクトメソッド

プロジェクトブラウザウィンドウで、メソッド一覧を表示する場合、以下の操作を行ってください。

- ・ クラス名のアイコンをダブルクリックします。または、アイコンの前に表示されている+記号をクリックします。

## 二次入口点一覧の表示

プロジェクトブラウザウィンドウで、プログラム定義中に記述されている二次入口点一覧を表示する場合、以下の操作を行ってください。

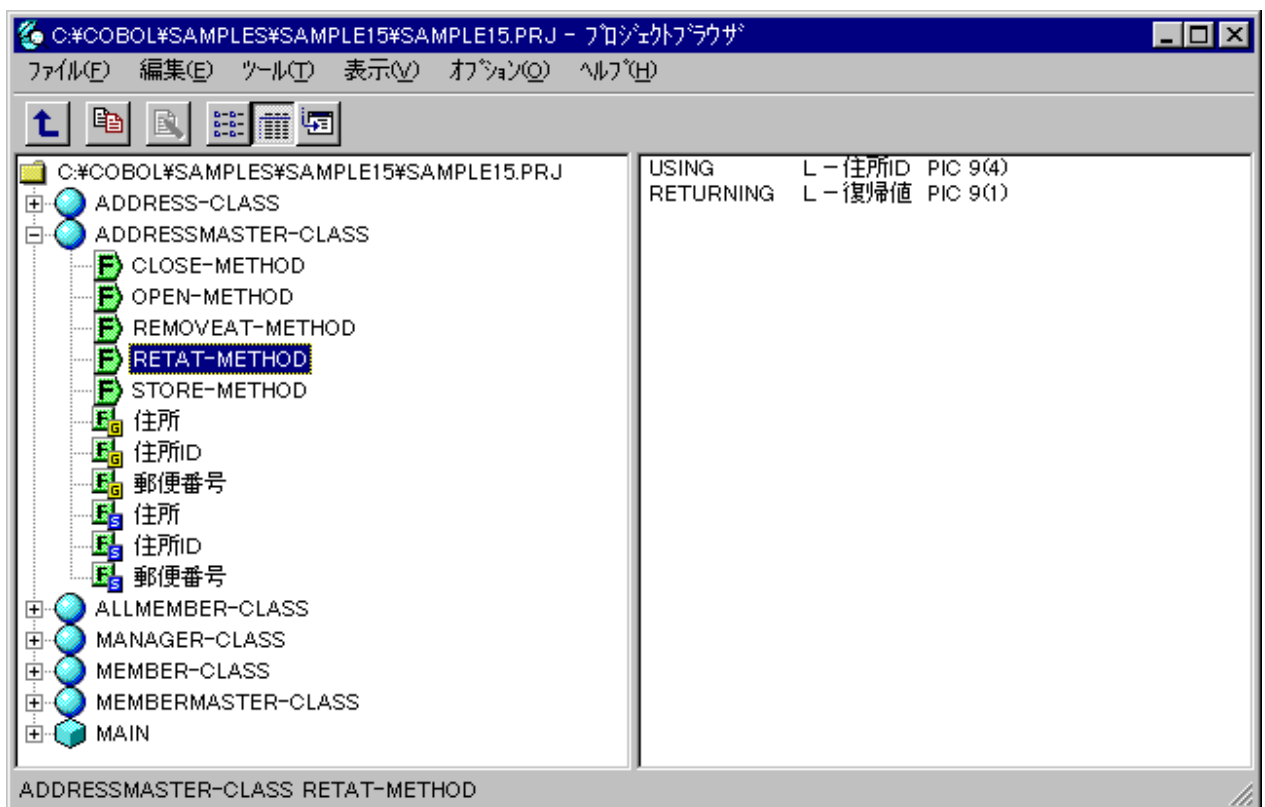
- ・ プログラム名のアイコンをダブルクリックします。または、アイコンの前に表示されている+記号をクリックします。

## インタフェースの表示

プロジェクトブラウザは、各プログラムおよびメソッド定義のPROCEDURE DIVISIONのUSING指定またはRETURNING指定に記述されたデータ項目の、データ名および属性を表示します。

インタフェースの表示を行う場合、以下の操作を行ってください。

1. [表示]メニューの“インタフェース”がチェックされていることを確認します。
2. 1.でチェックされていない場合、“インタフェース”を選択してチェック状態にします。
3. プロジェクトブラウザのウィンドウ上に表示されているプログラム名またはメソッド名をクリックします。



## COBOLソースファイル名表示

プロジェクトブラウザは、各クラス/メソッド/プログラム定義が記述されているCOBOLソースファイル名を表示することができます。

COBOLソースファイル名の表示を行う場合、以下の操作を行ってください。

1. [表示]メニューの“インタフェース”がチェックされていないことを確認します。
2. 1.でチェックされていた場合、“インタフェース”を選択してチェックを解除状態にします。
3. プロジェクトブラウザのウィンドウ上に表示されているクラス名またはプログラム名をクリックします。  
→ 選択したクラスまたはプログラムのツリー上で1つ下層の定義のCOBOLソースファイル名が表示されます。



- “インタフェース”がチェックされた状態でも、クラス定義を選択した場合はソースファイル名が表示されます。
- SymfowareおよびPowerAIMのプリコンパイラを使用してコンパイルした定義をプロジェクトブラウザで表示する場合、ブラウザ画面  
上に表示されるソースファイル名および「編集」機能によりエディタ表示されるソースファイル名は、プリコンパイル前のものになります。

## エディタ連携機能

プロジェクトブラウザは、表示されているクラス/メソッド/プログラム各定義のCOBOLソースファイルをエディタ表示して、定義されている  
行に位置付ける機能があります。

エディタ連携の手順を以下に説明します。

1. プロジェクトブラウザ上で、クラス名、メソッド名またはプログラム名を選択します。
2. プロジェクトブラウザウィンドウの[ツール]メニューから“編集”を選択します。

## テンプレート展開機能

プロジェクトブラウザは、他のCOBOLソースファイルに対して、特定の構文のテンプレートを展開する機能を持っています。この機能を  
テンプレート展開機能と呼びます。

プロジェクトブラウザで以下の構文のテンプレートを展開することができます。

- リポジトリ段落のクラス指定子
- オブジェクトプロパティ
- CALL文
- INVOKE文

これらの構文のテンプレートの展開形式と修正方法について説明します。



ブラウザのテンプレート展開機能は、システムのクリップボードを使用しています。そのため、クリップボードからの貼り付け機能を持っ  
ているエディタであれば、プロジェクトマネージャに標準に組み込まれているエディタでなくても、この機能は使用できます。

## リポジトリ段落のクラス指定子

リポジトリ段落に記述するクラス指定子のテンプレート展開を用いる手順を以下に説明します。

1. プロジェクトブラウザ上でクラス名を選択します。
2. [編集]メニューから“変換コピー”を選択します。
3. エディタ上で、リポジトリ段落の後へカーソルを位置付け、“貼り付け”を行ってください。

これにより、選択したクラスを参照するための次のようなテンプレートが展開されます。

CLASS クラス名.

なお、最後のピリオドは最後のクラス指定子の場合だけ必要です。それ以外の場合は削除してください。

## オブジェクトプロパティのテンプレート展開

プロパティのテンプレート展開を用いる手順を以下に説明します。

1. プロジェクトブラウザ上でプロパティメソッド名を選択します。
2. [編集]メニューから“変換コピー”を選択します。
3. エディタ上の文を挿入したい場所にカーソルを位置付け、“貼り付け”を行ってください。

これにより、選択したプロパティメソッドを呼び出す次のようなオブジェクトプロパティのテンプレートが展開されます。

プロパティ名 OF クラス名

なお、テンプレートではプロパティメソッドを呼び出すオブジェクト参照はクラス名として展開されます。呼び出すプロパティメソッドがオブジェクトメソッドである場合、このクラスを参照する適切なオブジェクト参照データに置き換えてください。

## CALL文の自動生成

CALL文の自動生成を行う手順を以下に説明します。

1. プログラム名を選択します。
2. [編集]メニューから“変換コピー”を選択します。
3. エディタ上の、文を挿入したい場所にカーソルを位置付け、“貼り付け”を行ってください。

これにより、選択したプログラムを呼び出すCALL文の次のようなテンプレートが展開されます。

CALL プログラム名 USING データ名… RETURNING データ名.

なお、USING指定およびRETURNING指定のデータ名には、選択したプログラムの手続き部見出しのUSING指定またはRETURNING指定のデータ名の指定がそのまま複製されています。この際、対応するデータ名の定義は展開されていないため、ブラウザに表示されるプログラムのインタフェースに従ってデータ名の定義を追加してください。

## INVOKE文のテンプレート展開

INVOKE文のテンプレート展開を用いる手順を以下に説明します。

1. プロジェクトブラウザ上でプロパティメソッドを除くメソッド名を選択します。
2. [編集]メニューから“変換コピー”を選択します。
3. エディタ上で、文を挿入したい場所にカーソルを位置付け、“貼り付け”を行ってください。

これにより、選択したメソッドを呼び出すINVOKE文の次のようなテンプレートが展開されます。

INVOKE クラス名 “メソッド名” USING データ名… RETURNING データ名.

展開されたテンプレートはそのままでは正しく翻訳できない場合もありますので、以下の点を確認して必要ならソースを修正してください。

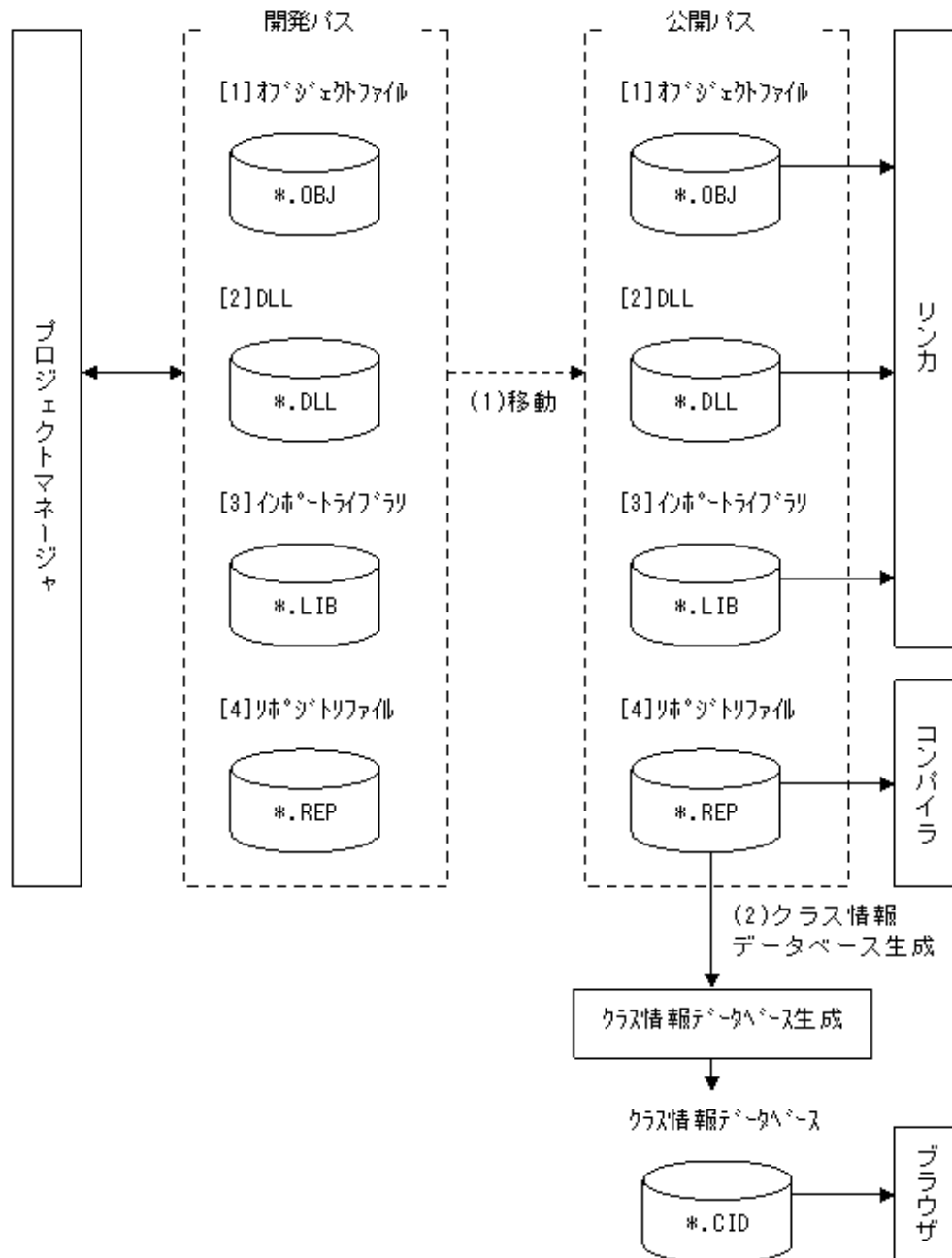
1. テンプレートではメソッドを呼び出すオブジェクト参照はクラス名として展開されます。呼び出すメソッドがオブジェクトメソッドである場合、このクラスを参照する適切なオブジェクト参照データに置き換えてください。
2. USING指定およびRETURNING指定のデータ名には、選択したメソッドの手続き部見出しのUSING指定またはRETURNING指定のデータ名の指定がそのまま複製されています。この際、対応するデータ名の定義は展開されていないため、ブラウザに表示されるメソッドのインタフェースに従ってデータ名の定義を追加してください。
3. 最後のピリオドは、IF文などの条件文やPERFORM文による繰り返し構造の内部に指定できません。テンプレートを展開した場所がそのような場所ならピリオドを削除してください。

## 17.5 クラスの公開

作成およびテストが完了したクラスは、新たな部品として、他プログラム開発の再利用の対象となります。その場合に、そのクラスは開発者以外からも利用可能でなければなりません。

作成したクラスのアクセス(再利用)を、開発者以外からも可能とすることを、「クラスの公開」と呼びます。

クラスを公開するための作業を、以下に説明します。



### (1) 移動

クラスを再利用するために必要な資源を、開発パスから公開パスへ移動します。

開発パスとは、プロジェクトマネージャを使用して開発作業を行っているときに、開発中の各資源が格納されていたパスを指します。公開パスとは、開発したクラスを他プログラムの再利用の対象とするために、開発に必要な資源を格納する、他開発者と共通で参照できるパスを指します。

クラスを再利用するために必要な資源は、このパス上に格納することにより他者のアクセスを可能にします。

クラス公開で公開対象となる資源およびその用途は、“表17.4 公開資源”のようになります。

表17.4 公開資源

|     | 資源名        | 用途                                                            |
|-----|------------|---------------------------------------------------------------|
| [1] | オブジェクトファイル | 再利用するクラス／プログラムと静的構造でリンクする場合に必要です。                             |
| [2] | DLL        | 再利用するクラス／プログラムと動的リンク構造または動的プログラム構造でリンクする場合に必要です。              |
| [3] | インポートライブラリ | 再利用するクラス／プログラムと動的リンク構造または動的プログラム構造でリンクする場合に必要です。              |
| [4] | リポジトリ      | 再利用するクラス／プログラムを翻訳する場合に必要です。<br>クラス情報データベースにクラス情報を反映する場合に必要です。 |

## (2) クラス情報データベース作成

作成したクラスの情報をクラスブラウザで表示可能とするために、クラス情報データベースの再作成を行います。  
クラス情報データベースの作成方法については、“[17.3.1 クラスブラウザ](#)”を参照してください。

## 第18章 オブジェクト指向プログラミング機能～さらに進んだ使い方～

本章では、オブジェクト指向プログラミング機能のさらに進んだ使い方について説明します。

### 18.1 例外処理

ここでは、例外処理の概要および書き方について説明します。

#### 18.1.1 概要

手続き部の宣言節部分にUSE文を記述することにより、例外手続きを指定することができます。例外手続きを記述すると、例外条件が発生したときに例外手続きに記述した処理が実行されます。発生する例外条件には、例外オブジェクトがあります。

#### 18.1.2 例外オブジェクト

例外オブジェクトは、エラー処理を1箇所ですべて行いたいような場合に使用します。たとえば、誤ったデータが入力されたらメッセージを表示する処理を行うとします。

データエラーメッセージを表示する手続きを持つクラス(DATA-ERROR)を定義し、データに誤りがあった場合、これをオブジェクト化し、RAISE文またはRAISING指定のEXIT文にそのオブジェクトを指定します。

このとき指定したオブジェクトが例外オブジェクトとなり、「データに誤りが発生した」という例外条件になります。

例外オブジェクトが発生すると、例外オブジェクトのクラス名と継承関係を持つクラス名が宣言節部分のUSE文に記述された場合、その例外手続きが実行されます。上記の例でいうと、USE文にDATA-ERRORと記述するとその手続きが実行されます。例外手続きで例外オブジェクトを使用したい場合、EXCEPTION-OBJECTと記述することにより使用することができます。例外手続きが正常に終了した場合、例外条件が発生した直後の文に制御が移ります。

プログラムのどこで例外が発生しても、必ずこの例外手続きが実行されます。したがって、データ入力エラーに対する処理を1箇所ですべて記述することができます。



#### 注意

- 例外オブジェクトには、オブジェクトインスタンスを指定してください。
- USE文に記述するクラス名は、リボジトリ段落に宣言してください。
- 宣言節に例外オブジェクトに対するUSE文が複数記述されている場合、先頭の例外手続きが実行されます。以下のような例外条件が発生した場合、[1]の例外手続きが実行されます。
  - 例外オブジェクトとしてCLASS-Aのオブジェクトが指定されている。
  - CLASS-AがCLASS-BおよびCLASS-Cを継承している。

```
DECLARATIVES.
 ERR-1 SECTION.
 USE AFTER EXCEPTION CLASS-C.
 DISPLAY "ERR CLASS-C". ... [1]
 ERR-2 SECTION.
 USE AFTER EXCEPTION CLASS-B.
 DISPLAY "ERR CLASS-B".
 ERR-3 SECTION.
 USE AFTER EXCEPTION CLASS-A.
 DISPLAY "ERR CLASS-A".
END DECLARATIVES.
```

- 発生した例外オブジェクトに対する例外手続きが存在しない、または例外オブジェクトがNULLオブジェクトなど例外手続きが実行できない場合、例外を発生させた文によって以下のように動作します。

ー RAISE文

例外条件を発生させた文の直後の文に制御が移ります。

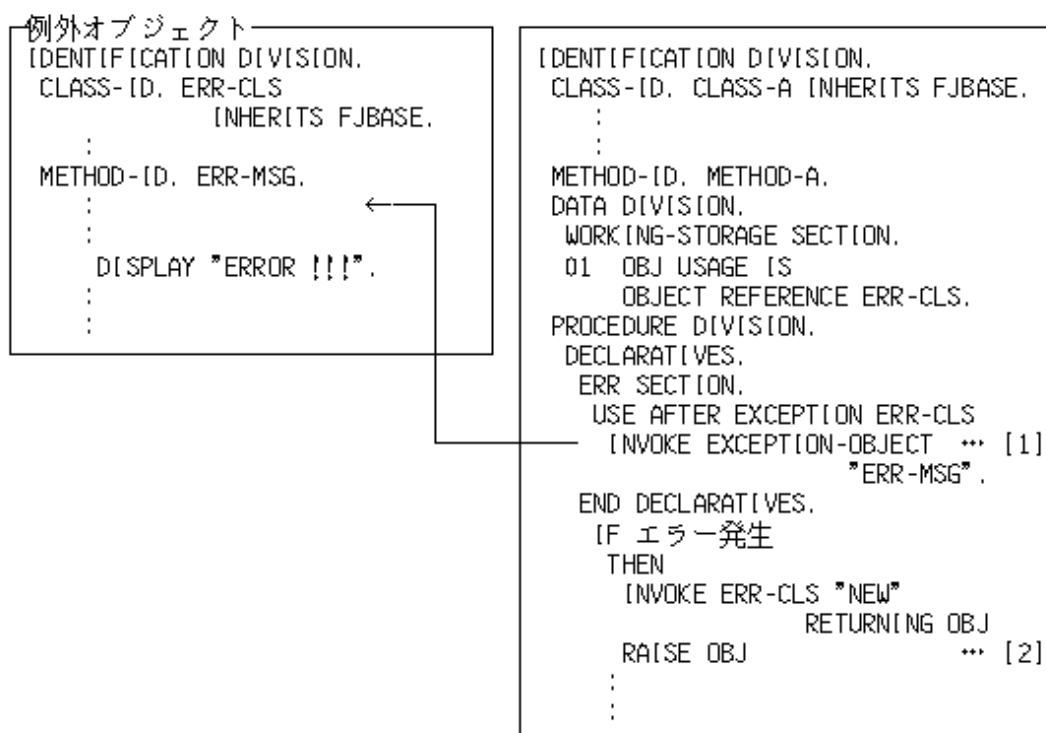
ー RAISING指定のEXIT文

プログラムまたはメソッドは異常終了します。

- 例外の発生により特定のUSE手続きを実行している途中で、再び同じUSE手続きに移行する例外が発生し、USE手続きが再帰的に呼び出されたとき、実行の制御がそのUSE手続きの最後まで到達したならば、プログラムまたはメソッドは異常終了します。

### 18.1.3 RAISE文の動作

RAISE文は、手続きの途中で例外条件を発生させ、その手続きが属するプログラムまたはメソッドに記述された例外手続きを実行します。

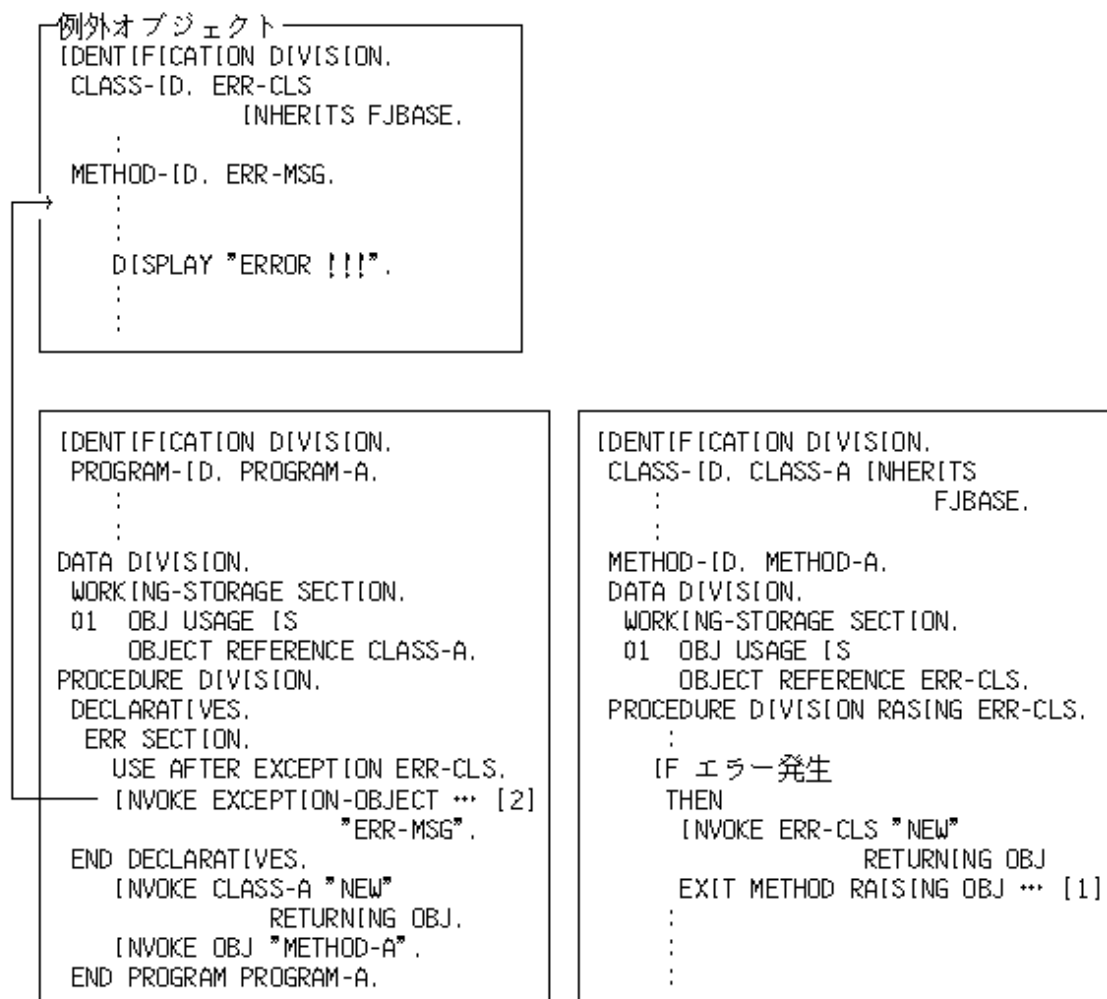


[2]のRAISE文を実行すると、発生した例外条件に対する手続き([1]のINVOKE文)が実行されます。

### 18.1.4 RAISING指定のEXIT文の動作

RAISING指定のEXIT文は、プログラムまたはメソッドの手続きを終了させ、呼出し元に復帰した後、例外条件を発生させます。したがって、呼出し元プログラムまたはメソッドに記述された例外手続きを実行します。





呼び出したプログラムまたはメソッドでRAISING指定のEXIT文を実行する([1])と、呼出し元に戻った後、例外手続き([2])のINVOKE文が実行されます。

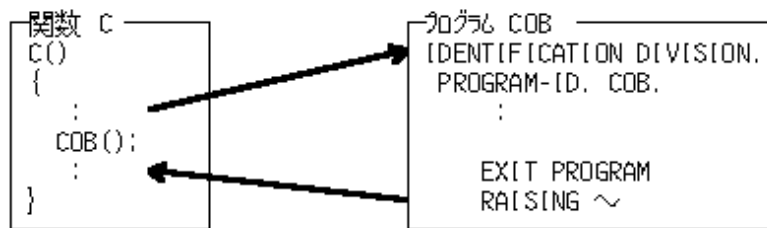
## 注意

以下のCOBOLプログラムでは、EXIT文で例外条件を発生させることができません。

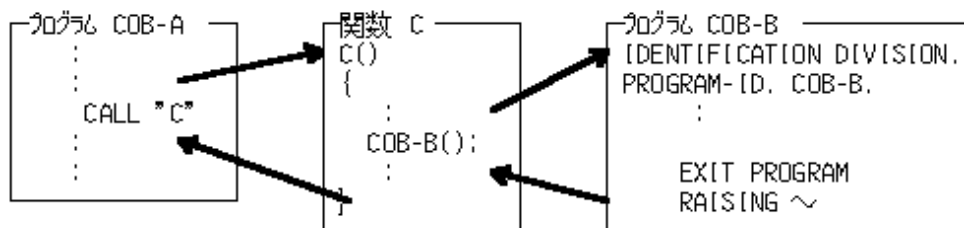
- ・ 主プログラム
- ・ 他言語プログラムから呼び出されたCOBOLプログラム(下図[1])

ただし、COBOLプログラムから他言語プログラムを介して呼び出されたCOBOLプログラム(下図[2])の場合、EXIT文で例外条件を発生させることができます。

[1] 例外条件が発生しません。



[2] 例外条件が発生し、呼出し元のCOBOLプログラム(COB-A)に記述された例外手続きを実行します。



## 18.2 動的プログラム構造

ここでは、クラスとメソッドの動的プログラム構造について説明します。なお、プログラムの動的プログラム構造については、“[10.1.5 動的プログラム構造](#)”を参照してください。

また、プログラムの動的プログラム構造と併用する場合は、“[10.1.5.3 注意事項](#)”を必ずお読みください。

### 18.2.1 動的プログラム構造の特徴

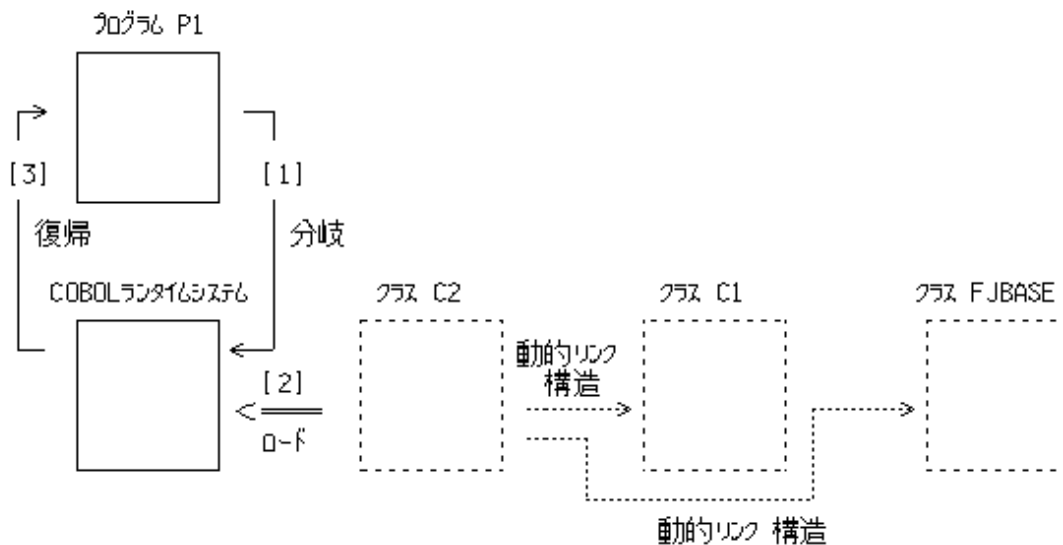
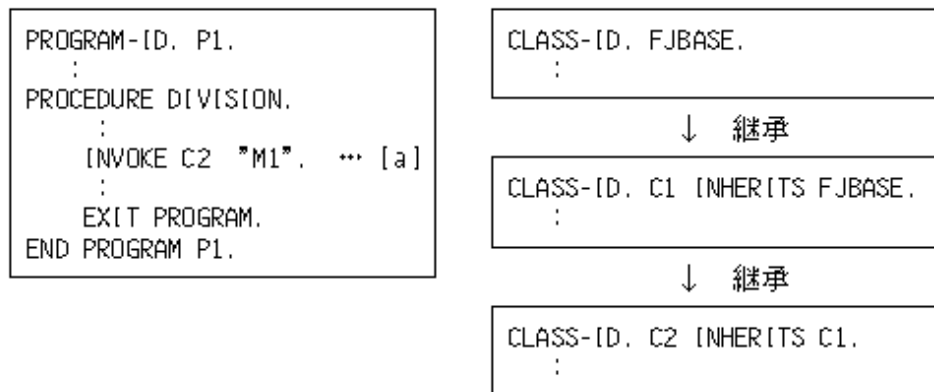
動的プログラム構造では、クラスおよびメソッドの仮想記憶上へのローディングは、実際に参照または呼び出されたときに、COBOLランタイムシステムによって行われます。

このため、実行可能ファイルの起動時にすべてのファイルがロードされる単純構造および動的リンク構造と比べると、実行可能ファイルの起動が速くなります。また、実行に必要なクラスおよびメソッドだけがロードされるため、仮想記憶および実記憶の節約も期待できます。ただし、クラスを参照している文の実行およびメソッドの呼出しは、COBOLランタイムシステムを介して行われるため、遅くなります。

### 18.2.2 クラスの動的プログラム構造

クラスを動的プログラム構造にすると、アプリケーションで使用しているクラスの仮想記憶上へのローディングは、クラスが参照されたときにCOBOLランタイムシステムによって行われます。このとき、ロードされるクラスが直接および間接的に継承しているクラスも同時にロードされます。これは、ロードされるクラスによって直接および間接的に継承しているクラスは、動的リンク構造となるためです。

以下のようなプログラムでC2クラスが動的プログラム構造の場合、C2クラスは、[a]のINVOKE文が実行されたときに仮想記憶上にロードされます。C2クラスが直接継承しているC1クラスおよび間接的に継承しているFJBASEクラスも、このときロードされます。



[1]～[3]は処理する順序を示します。

[1] INVOKE文の実行により、C2クラスが参照されるため、COBOLランタイムシステムが呼び出されます。

[2] COBOLランタイムシステムは、C2クラスをロードします。このとき、C2クラスが継承しているC1クラスとFJBASEクラスは、システムによってロードされます。

[3] プログラムP1に復帰します。



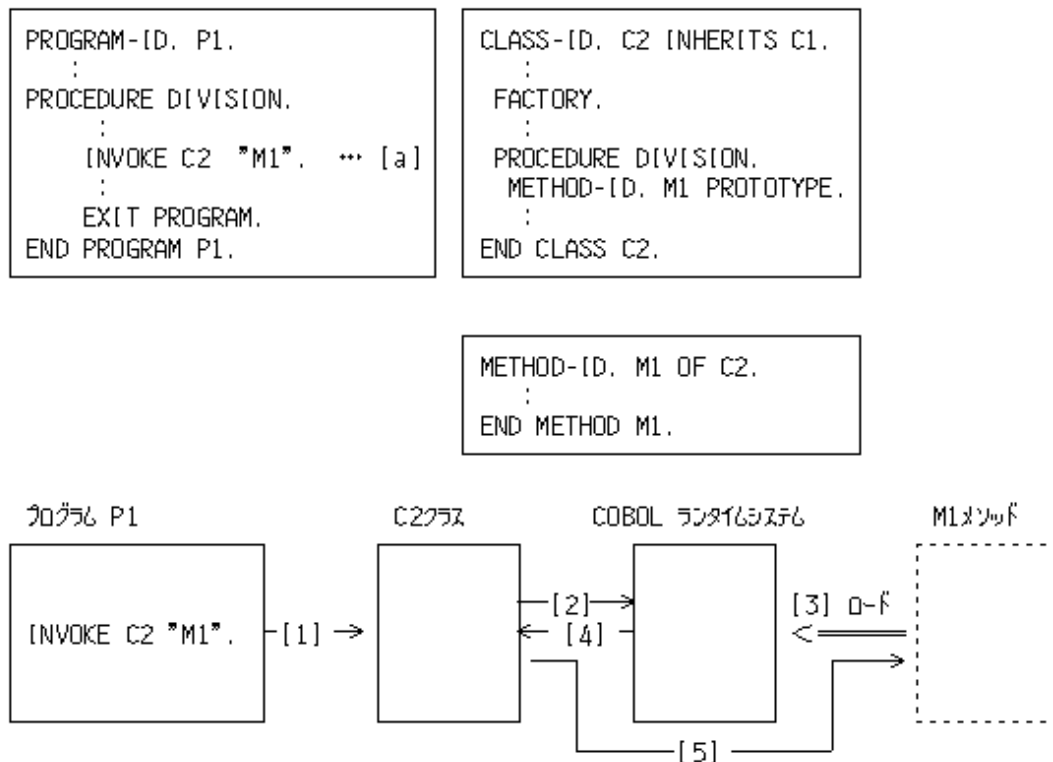
#### 注意

クラス名として日本語を使用する場合、クラス名がリンカの規則に従っているかどうかは利用者が判断します。なお、ツールによっては日本語をサポートしていないものがあるため、英数字の使用を推奨します。

### 18.2.3 メソッドの動的プログラム構造

PROTOTYPE宣言により分離されたメソッドを動的プログラム構造にすると、アプリケーションで使用しているメソッドの仮想記憶上へのローディングは、メソッドが呼び出されたときに、COBOLランタイムシステムによって行われます。

以下のようなプログラムでM1メソッドが動的プログラム構造の場合、M1メソッドは、[a]のINVOKE文が実行されたときに仮想記憶上にロードされます。



[1]～[5]は処理する順序を示します。

- [1] M1メソッドの呼出しにより、C2クラスが呼び出されます。
- [2] COBOLランタイムシステムが呼び出されます。
- [3] COBOLランタイムシステムはM1メソッドをロードします。
- [4] C2クラスに復帰します。
- [5] M1メソッドが呼び出されます。

## 18.2.4 動的プログラム構造の作成から実行

ここでは、動的プログラム構造の作成から実行までの作業について説明します。

### 18.2.4.1 翻訳の方法

ここでは、翻訳時に必要となる翻訳オプションおよび注意事項について説明します。

翻訳の手順については、“[6.4.5 プロジェクトのビルド・リビルド](#)”を参照してください。

#### クラスを動的プログラム構造にする場合

クラスを動的プログラム構造にするには、クラスを参照しているソースプログラムの翻訳時に、翻訳オプションDLOADを指定します。この指定により、このソースプログラムから参照されているクラスは、動的プログラム構造になります。[参照]“[A.2.11 DLOAD\(プログラム構造の指定\)](#)”

ただし、他のソースプログラムでそのクラスが動的リンク構造になっていると、実行可能ファイルの起動時にそのクラスはシステムによってロードされてしまいます。このため、COBOLの実行単位でプログラム構造を統一してください。

## メソッドを動的プログラム構造にする場合

メソッドを動的プログラム構造にするには、動的プログラム構造にしたいメソッドを原型定義し、そのメソッドの原型定義を含むクラスの翻訳時に、翻訳オプションDLOADを指定します。このように、クラスの翻訳の方法によって、分離されたメソッドのプログラム構造が決まります。

プロパティメソッドを動的プログラム構造にするには、利用者がプロパティメソッドを原型定義し、そのメソッドの原型定義を含むクラスの翻訳時に、翻訳オプションDLOADを指定します。[参照]“[A.2.11 DLOAD\(プログラム構造の指定\)](#)”



### 例

以下のような継承関係のあるC1クラスとC2クラスがあり、C1クラスが翻訳オプションNODLOADで翻訳され、C2クラスが翻訳オプションDLOADで翻訳されているとします。

この場合、C2クラスで定義されているメソッドのうち、原型定義されているM5メソッドとM7メソッドが動的プログラム構造となります。M6メソッドとM8メソッドは原型定義されていないため、C2クラスが翻訳オプションDLOADで翻訳されても動的プログラム構造になりません。

C1クラスで定義されているメソッドは、C1クラスが翻訳オプションNODLOADで翻訳されているため動的プログラム構造にはなりません。

#### 翻訳オプションNODLOAD 指定

```
CLASS-[D, C1 INHERITS FJBASE.
:
FACTORY.
:
PROCEDURE DIVISION.
 METHOD-[D, M1 PROTOTYPE.
 :
 METHOD-[D, M2, _____
 :
END FACTORY.
OBJECT.
:
PROCEDURE DIVISION.
 METHOD-[D, M3 PROTOTYPE.
 :
 METHOD-[D, M4, _____
 :
END OBJECT.
END CLASS C1.
```

#### 翻訳オプションDLOAD 指定

```
CLASS-[D, C2 INHERITS C1.
:
FACTORY.
:
PROCEDURE DIVISION.
 METHOD-[D, M5 PROTOTYPE.
 :
 METHOD-[D, M6, _____
 :
END FACTORY.
OBJECT.
:
PROCEDURE DIVISION.
 METHOD-[D, M7 PROTOTYPE.
 :
 METHOD-[D, M8, _____
 :
END OBJECT.
END CLASS C2.
```

## 18.2.4.2 リnkの方法

ここでは、リンク時に必要となるインポートライブラリについて説明します。

リンクの手順については、“[6.4.5 プロジェクトのビルド・リビルド](#)”を参照してください。

### プログラムの実行可能ファイルまたはDLLを作成する場合

インポートライブラリは不要です。

### クラスのDLLを作成する場合

DLLを作成するクラスが直接および間接的に継承しているすべてのクラスのインポートライブラリが必要となります。

### 分離されたメソッドのDLLを作成する場合

DLLを作成する分離されたメソッドの手続き部で、定義済みオブジェクト一意名SUPERが利用されている場合だけ、そのメソッドが定義されているクラスが直接および間接的に継承しているすべてのクラスのインポートライブラリが必要となります。

### 18.2.4.3 DLLの構成とファイル名

ここでは、クラスおよびメソッドのDLLについて、標準的な構成とファイル名について説明します。以下のように指定することにより、実行時に必要となるエントリ情報を省略できます。

#### クラスのDLLの構成とファイル名

クラス単位でDLLを作成し、ファイル名を“クラス名.DLL”とします。

#### メソッドのDLLの構成とファイル名

メソッド単位でDLLを作成し、ファイル名を“クラス名\_メソッド名.DLL”とします。ここでいう“クラス名”とは、メソッド原型が定義されているクラスのクラス名を示します。

### 18.2.4.4 クラスとメソッドのエントリ情報

ここでは、動的プログラム構造のアプリケーションを実行するときに必要なエントリ情報について説明します。

副プログラムが動的プログラム構造の場合は副プログラムのエントリ情報が、クラスが動的プログラム構造の場合はクラスのエントリ情報が、メソッドが動的プログラム構造の場合はメソッドのエントリ情報がそれぞれ必要となります。

クラスおよびメソッドのエントリ情報は、副プログラムのエントリ情報と同様に環境変数情報@CBR\_ENTRYFILEにエントリ情報ファイル名を指定します。詳細については、“[5.4.1.23 @CBR\\_ENTRYFILE \(エントリ情報ファイルの指定\)](#)”を参照してください。

なお、副プログラムのエントリ情報については、“[5.4.2 副プログラムのエントリ情報](#)”を参照してください。



副プログラムのエントリ情報は実行用の初期化ファイルに指定できますが、クラスおよびメソッドのエントリ情報は、実行用の初期化ファイルに指定できません。

### エントリ情報の記述形式

ここでは、クラスおよびメソッドのエントリ情報の記述形式について説明します。

#### クラスのエントリ情報

クラスのエントリ情報とは、クラスとそのクラスが格納されているDLLを関連付けるための情報です。

以下にクラスのエントリ情報の記述形式について説明します。

|                 |     |     |
|-----------------|-----|-----|
| [CLASS]         | ... | [1] |
| クラス名= DLL ファイル名 | ... | [2] |
| :               |     |     |

##### [1] クラスのエントリ情報の定義の開始を示すセクション名

セクション名は、“CLASS”の固定文字列です。このセクションは、1つのエントリ情報ファイルに1つしか記述できません。

##### [2] クラスのエントリ情報

クラス名には利用するクラスのクラス名を指定し、DLLファイル名には利用するクラスが格納されているDLLのファイル名を絶対パス名または相対パス名で指定します。DLLファイル名の拡張子は“DLL”でなければなりません。相対パス名で指定した場合のDLLの検索順序については、“[5.4.2 副プログラムのエントリ情報](#)”を参照してください。

DLLファイル名が“クラス名.DLL”の場合は、クラスのエントリ情報は省略できます。

#### メソッドのエントリ情報

メソッドのエントリ情報とは、メソッドとそのメソッドが格納されているDLLを関連付けるための情報です。

以下にメソッドのエントリ情報の記述形式について説明します。

[クラス名.METHOD] ... [1]  
メソッド名= DLL ファイル名 ... [2]  
:

## [1] メソッドのエントリ情報の定義の開始を示すセクション名

セクション名は、呼び出すメソッドが定義されているクラス名に、".METHOD"の固定文字列を付加した文字列です。このセクションはエントリ情報ファイルの1つのクラスに対して1つしか記述できません。

## [2] メソッドのエントリ情報

メソッド名には呼び出すメソッドのメソッド名を指定し、DLLファイル名には呼び出すメソッドが格納されているDLLのファイル名を絶対パス名または相対パス名で指定します。DLLファイル名の拡張子は"DLL"でなければなりません。相対パス名で指定した場合のDLLの検索順序については、"[5.4.2 副プログラムのエントリ情報](#)"を参照してください。

DLLファイル名が"クラス名\_メソッド名.DLL"の場合は、メソッドのエントリ情報は省略できます。



## 参考

プロパティメソッドが動的プログラム構造の場合、プロパティメソッドが原型定義されているクラスのメソッドのエントリ情報のメソッド名は次のように指定する必要があります。

### GETメソッドの場合

"\_GET\_プロパティ名"を指定します。ただし、GETメソッドのDLLファイル名が"クラス名\_\_GET\_プロパティ名"(注)の場合は、メソッドのエントリ情報は省略できます。

### SETメソッドの場合

"\_SET\_プロパティ名"を指定します。ただし、SETメソッドのDLLファイル名が"クラス名\_\_SET\_プロパティ名"(注)の場合は、メソッドのエントリ情報は省略できます。

注：クラス名とGETまたはSETの間には、アンダースコアが2つ入っています。



## 注意

クラスおよびメソッドのソースプログラムが翻訳オプションALPHALで翻訳された場合、クラス名、メソッド名は大文字と小文字は等価に扱われます。[参照]"[A.2.2 ALPHAL \(英小文字の扱い\)](#)"

このとき、エントリ情報のクラス名とメソッド名は、大文字で指定してください。

## 実行に必要なエントリ情報

アプリケーションを実行するときに必要なエントリ情報は、そのアプリケーション中の翻訳オプションDLOADで翻訳されたプログラム、クラスおよびメソッドに対して、"[表18.1 実行に必要なエントリ情報](#)"に示すエントリ情報が必要となります。

表18.1 実行に必要なエントリ情報

| 種別    | クラスのエントリ情報                                                                  | メソッドのエントリ情報                  |
|-------|-----------------------------------------------------------------------------|------------------------------|
| プログラム | ・ 手続き部に記述されたクラス                                                             | 呼び出しているメソッドの中で動的プログラム構造のメソッド |
| メソッド  |                                                                             |                              |
| クラス   | ・ 手続き部に記述されたクラス<br>・ メソッドの連絡節での復帰項目の定義でオブジェクト参照データ項目のUSAGE 句に指定されているクラス (注) |                              |

注：このクラスのエントリ情報は、メソッドの呼出し元のソースプログラムに実行時の適合チェックが有効な場合に必要となります。実行時の適合チェックについては、"[16.4 適合](#)"、"[A.2.5 CHECK \(CHECK機能の使用の可否\)](#)"のCHECK(ICONF)の説明を参照してください。



## 例

プログラムP1を実行するために必要なエントリ情報を説明します。以下のすべてのソースプログラムに翻訳オプションDLOADが指定されているものとします。

### P1.EXE

```
PROGRAM-ID, P1.
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ1 USAGE OBJECT REFERENCE
 FACTORY C1.
01 OBJ2 USAGE OBJECT REFERENCE C2.
01 OBJ3 USAGE OBJECT REFERENCE C3.
PROCEDURE DIVISION.
...
SET OBJ1 TO C1.
...
[INVOKE C2 "NEW" RETURNING OBJ2.
[INVOKE OBJ2 "M1" RETURNING OBJ3.
...
[INVOKE OBJ2 "M2" USING OBJ3.
...
EXIT PROGRAM.
END PROGRAM P1.
```

### エントリ情報ファイル

```
[CLASS]
C1=C1.DLL
C2=C2.DLL
C3=C3.DLL

[C1.METHOD]
M1=C1_M1.DLL

[C2.METHOD]
M2=C2_M2.DLL
```

### C1.DLL

```
CLASS-ID, C1 INHERITS FJBASE.
:
OBJECT.
PROCEDURE DIVISION.
METHOD-ID, M1 PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
01 OBJ3 USAGE OBJECT REFERENCE C3.
PROCEDURE DIVISION RETURNING OBJ3.
END METHOD M1.
:
END CLASS C1.
```

### C2.DLL

```
CLASS-ID, C2 INHERITS C1.
:
OBJECT.
PROCEDURE DIVISION.
METHOD-ID, M2 PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
01 OBJ3 USAGE OBJECT REFERENCE C3.
PROCEDURE DIVISION USING OBJ3.
END METHOD M2.
:
END CLASS C2.
```

### C1\_M1.DLL

```
METHOD-ID, M1 OF C1.
:
END METHOD M1.
```

### C2\_M2.DLL

```
METHOD-ID, M2 OF C2.
:
END METHOD M2.
```

### C3.DLL

```
CLASS-ID, C3 INHERITS FJBASE.
:
END CLASS C3.
```

\_\_ : エントリ情報の必要なクラスおよびメソッド





## 参考

上記のエントリ情報では、クラスのDLLファイル名は“クラス名.DLL”、メソッドのDLLファイル名は“クラス名\_メソッド名.DLL”となっているため、省略することができます。

## 18.3 メモリに関するチューニング

ここでは、COBOLのオブジェクト指向プログラムでのメモリに関するチューニングについて説明します。

### 18.3.1 概要

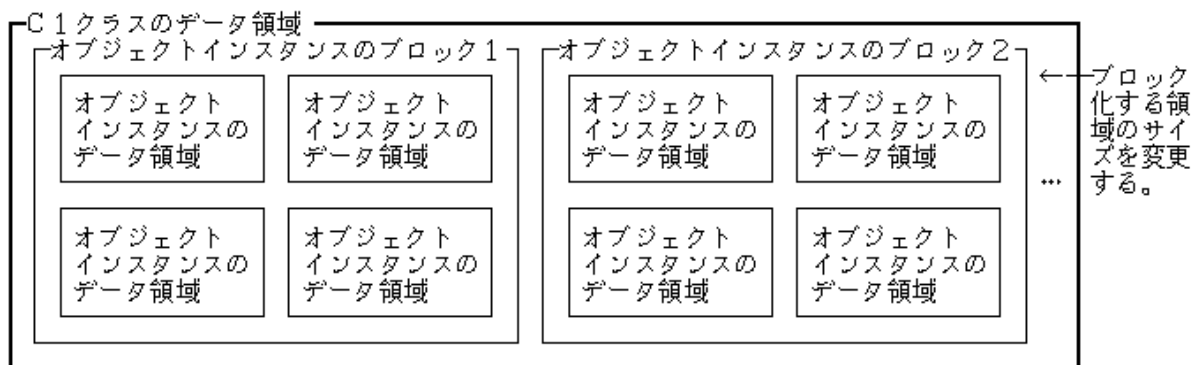
オブジェクト指向プログラムでは、NEWメソッドを実行することにより、オブジェクトインスタンスが生成されます。このとき、オブジェクトインスタンスのデータ部に記述したデータ項目およびオブジェクトインスタンスを動作させるために必要な作業域などが、オブジェクトインスタンスのデータ領域として獲得されます。

オブジェクトインスタンスのデータ領域は、COBOLソースプログラムに記述したオブジェクト定義の内容や、生成するオブジェクトインスタンスが含まれるクラスの継承関係によって、大きさが異なります。

オブジェクトインスタンスのデータ領域は、通常、NEWメソッドの実行のタイミングでブロック化して獲得されます。ブロック化とは、複数個分のオブジェクトインスタンスのデータ領域をまとめて獲得することを指します。

クラスが必要とするオブジェクトインスタンスのデータ領域は、クラスごとに固定となる領域であるため、実行時に領域長が変更されることはありません。しかし、オブジェクトインスタンスのデータ領域をブロック化したデータ領域のサイズは、実行時に決定することが可能です。

オブジェクト指向プログラムでのメモリに関するチューニングとは、ブロック化を含むオブジェクトインスタンスのデータ領域の獲得方法をコントロールすることにより、最適なメモリ環境を構築することをいいます。具体的には、アプリケーションでのクラスの使用方法に適したオブジェクトインスタンスの獲得方法の設定を行うことで、使用メモリの節約または実行性能の向上を図ることができます。



### 18.3.2 使用メモリの節約

使用メモリを抑えるためには、アプリケーションの実行中に獲得するオブジェクトインスタンスのデータ領域を最小限にする必要があります。

オブジェクトインスタンスのデータ領域は、とくに指定がなければ、COBOLランタイムシステムが実行性能も考慮し、最適な値でブロック化処理を行います。ただし、ブロック化することで、アプリケーションの動作によっては、使用しない可能性のある領域を獲得していることになります。したがって、オブジェクトインスタンスのデータ領域をブロック化しなければ、必要以上の領域を獲得することがなく、使用メモリを削減することができます。

オブジェクトインスタンスのデータ領域をブロック化しないで、メモリ優先で領域の獲得を行うためには、以下の指定をします。クラス情報および環境変数情報の指定の詳細については、“[18.3.4 メモリのチューニングに関する実行環境情報](#)”を参照してください。

## クラスに対する指定(クラス情報)

使用メモリの節約を行いたいクラスのオブジェクトインスタンスの格納数に1を指定します。

クラス情報ファイル

```
[InstanceBlock]
C1=1
```

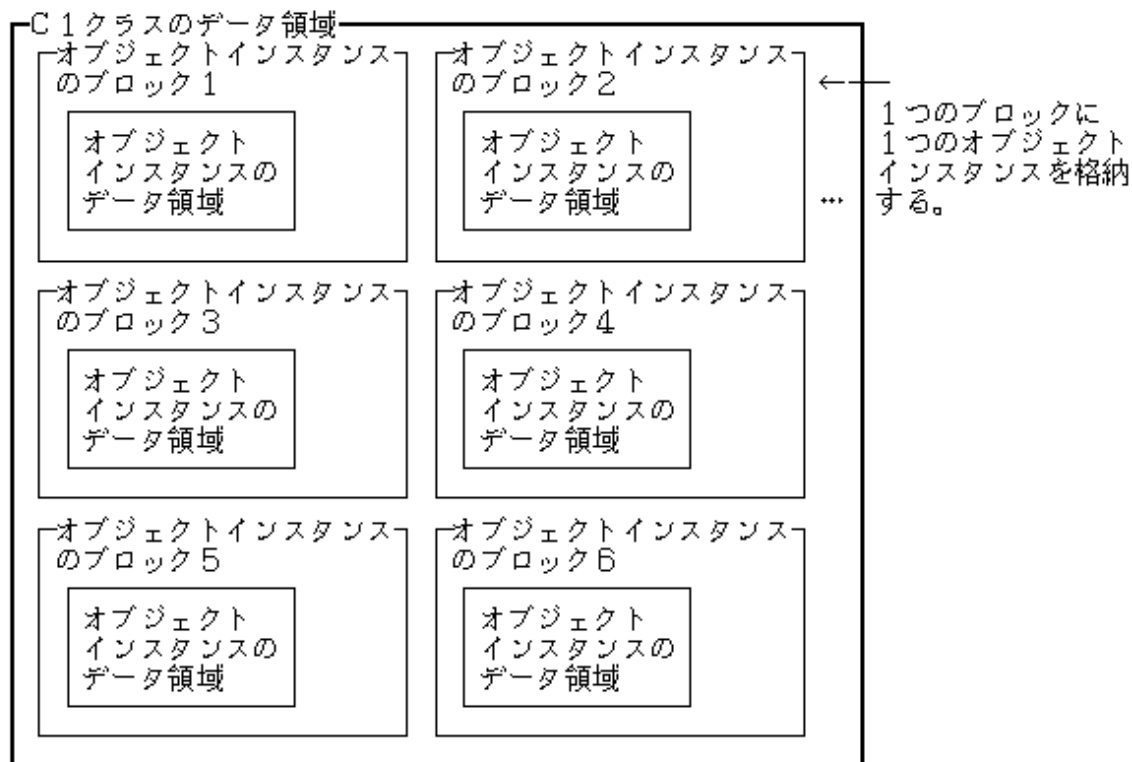
## アプリケーションの実行単位に対する指定(環境変数情報)

オブジェクトインスタンスをブロック化しないで獲得するように指定します。

```
@CBR_InstanceBlock=UNUSE
```

上記のどちらかの指定を行うことにより、オブジェクトインスタンスで必要となる最小の領域の獲得を行います。主に、オブジェクト定義のデータ部に記述したデータ領域が大きい場合、クラスが継承する階層が深い場合などに効果があります。

ただし、指定によりオブジェクトインスタンスの生成ごとに領域の獲得を行うことになるため、アプリケーションによっては、実行性能が低下します。



## 18.3.3 実行性能の向上

実行性能を向上させるためには、オブジェクトインスタンスのデータ領域を、アプリケーションの目的に応じて効率的にブロック化する必要があります。ブロック化すると、将来使用する可能性のあるオブジェクトインスタンスのデータ領域をまとめて獲得するため、オブジェクトの生成のタイミングでは、獲得済みの領域から必要となる領域を割り当てて使用します。この結果、領域獲得処理のオーバーヘッドが削減され、指定値によっては実行性能が向上します。

アプリケーションの実行時にオブジェクトインスタンスの領域をどの単位でまとめて獲得するのかは、以下の指定に従います。クラス情報の指定の詳細については、“[18.3.4 メモリのチューニングに関する実行環境情報](#)”を参照してください。

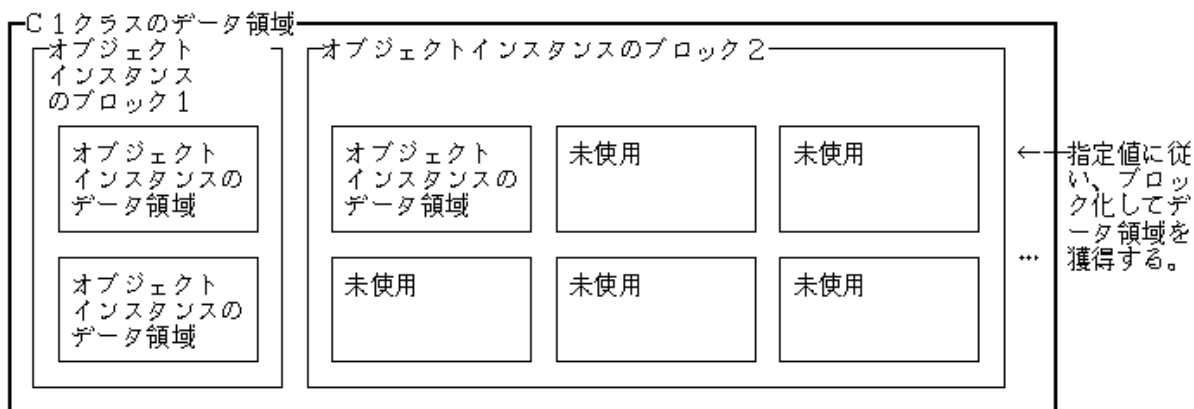
## クラスごとの指定(クラス情報)

ブロック化するクラスのオブジェクトインスタンスの格納数(初期数,増分数)に任意の数を指定します。

### クラス情報ファイル

```
[InstanceBlock]
C1=2, 6
```

上記の場合、アプリケーションの実行中にC1クラスに対するオブジェクトインスタンスの初回生成時にオブジェクトインスタンスのデータ領域が2個格納可能な領域を獲得します。その後、3個目のオブジェクトインスタンスの生成が行われたとき、初回に獲得した領域内に割り当てたデータ領域が存在しなければ、さらにオブジェクトインスタンスのデータ領域が6個格納可能な領域を獲得します。



同一クラスのオブジェクトインスタンスの生成、削除が繰り返された場合、ブロック化したオブジェクトインスタンスの領域内に再利用可能な領域があれば、その領域をあらたなオブジェクトインスタンスに割り当てます。

## 18.3.4 メモリのチューニングに関する実行環境情報

メモリのチューニングに関する実行環境情報について説明します。なお、実行環境情報の設定方法については、“[5.3.2 実行環境情報の設定方法](#)”を参照してください。

### 18.3.4.1 環境変数情報

以下に、環境変数情報の一覧を示します。

#### ファイルに関するもの

“[18.3.4.1.1 @CBR\\_ClassInfFile](#)(クラス情報ファイルの指定)”

#### オブジェクトインスタンスに関するもの

“[18.3.4.1.2 @CBR\\_InstanceBlock](#)(オブジェクトインスタンスの獲得方法の指定)”

#### 18.3.4.1.1 @CBR\_ClassInfFile(クラス情報ファイルの指定)

```
@CBR_ClassInfFile = クラス情報ファイル名
```

オブジェクト指向プログラムで使用するクラス情報を定義したクラス情報ファイル名を指定します。なお、クラス情報の詳細については、“[18.3.4.2 クラス情報](#)”を参照してください。

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

#### 18.3.4.1.2 @CBR\_InstanceBlock(オブジェクトインスタンスの獲得方法の指定)

```
@CBR_InstanceBlock = [{ USE }]
 UNUSE
```

クラスごとのオブジェクトインスタンスの領域を、ブロック化して獲得する(USE)か、しない(UNUSE)かを指定します。ブロック化しない(UNUSE)を指定した場合、オブジェクトの生成のタイミングごとに、オブジェクトインスタンスで必要となる最小の領域を獲得します。

当指定は、オブジェクト指向プログラムで使用するすべてのCOBOLのクラスに対して有効になります。ただし、クラス情報ファイルにオブジェクトインスタンスの格納数の指定がされているクラスは、クラス情報に指定された値に従ってオブジェクトインスタンスの領域をブロック化して獲得します。

### 18.3.4.2 クラス情報

クラス情報は、オブジェクト指向プログラムで使用するクラスに対する情報をセクションごとに指定します。指定する情報を以下に示します。

#### クラス情報

“18.3.4.2.1 InstanceBlockセクション(オブジェクトインスタンスの格納数の指定)”

これらのクラス情報を格納したテキストファイルをクラス情報ファイルといいます。プログラムを実行するときのクラス情報ファイルの指定方法については、“18.3.4.1 環境変数情報”を参照してください。

#### 18.3.4.2.1 InstanceBlockセクション(オブジェクトインスタンスの格納数の指定)

```
[InstanceBlock]
クラス名= 初期数[, 増分数]
```

プログラムの実行時に獲得するオブジェクトインスタンスの領域に格納するオブジェクトインスタンスの数を初期数および増分数で指定します。初期数、増分数ともに指定できる値は、1以上の整数です。増分数が省略された場合は、1が指定されたものとみなします。

初回のオブジェクトインスタンスの領域の獲得時には、初期数で指定した数のオブジェクトインスタンスのデータ領域が格納可能な領域を獲得します。プログラムの実行中に、初期数で指定した数を超えるオブジェクトインスタンスを生成する場合は、増分数で指定した数のオブジェクトインスタンスのデータ領域が格納可能な領域を獲得します。

クラスに対する指定がない場合、環境変数情報@CBR\_InstanceBlockの指定に従います。

[参照]“18.3.4.1.2 @CBR\_InstanceBlock(オブジェクトインスタンスの獲得方法の指定)”

## 18.4 Visual C++プログラムとの連携

ここでは、COBOLからVisual C++のオブジェクトを操作する方法について説明します。

### 18.4.1 概要

ここでは、COBOLと他言語との連携のうち、Visual C++のオブジェクトを操作する方法について説明します。Visual C++プログラムとの連携では、内部的にCまたはVisual C++の関数呼出しを利用するので、“10.3 C言語プログラムとの結合”の知識を前提とします。

### 18.4.2 Visual C++連携の方法

Visual C++は、オブジェクト指向プログラミング言語として広く使用されており、多くの有用なクラスが定義されています。オブジェクトコードの構造の違いから、COBOLからはVisual C++で定義されたクラスを直接利用することはできませんが、次の2つの方法を用いて、Visual C++のオブジェクトを操作することができます。

1. Visual C++側でオブジェクトを操作する関数を定義し、COBOLからCまたはVisual C++をプログラム連携することで、オブジェクトを操作した結果だけを利用する。
2. Visual C++側、COBOL側にインタフェースプログラムを定義し、COBOLからオブジェクトとして操作できるようにする。

1.の方法は、単純な外部プログラム呼出しで実現できます。オブジェクトの操作の結果だけを利用する場合には、この方法で十分です。2.の方法では、Visual C++のクラスをCOBOLのクラスと同じように操作することができるようになります。つまり、COBOLのINVOKE文によりVisual C++のメンバ関数を呼び出し、プロパティの参照/設定の構文でメンバ変数の参照/設定ができるようになります。

ここでは2.の連携方法について説明します。

### 18.4.3 Visual C++連携の概要

ここでは、Visual C++連携の概要について説明します。

#### 18.4.3.1 COBOLおよびVisual C++でのクラスの対応

Visual C++のオブジェクト操作では、そのクラスのpublicでないメンバ関数/メンバ変数にどのようなものがあるかは意識する必要はありません。また、そのクラスがどのようなクラスを継承して定義されたかも意識する必要はありません。そのクラスが外部に見せているインタフェースだけが問題になります。この観点から、COBOLとVisual C++のオブジェクトは“表18.2 COBOLおよびVisual C++のクラスの対応”のように対応付けられます。

表18.2 COBOLおよびVisual C++のクラスの対応

| 概念        | Visual C++       | COBOL          |
|-----------|------------------|----------------|
| クラス       | クラス              | クラス            |
| オブジェクト    | オブジェクト           | オブジェクト(インスタンス) |
| オブジェクトデータ | public宣言されたメンバ変数 | プロパティ          |
| メソッド      | public宣言されたメンバ関数 | メソッド           |

#### 18.4.3.2 処理の概要

Visual C++のオブジェクトを操作するために、COBOLおよびVisual C++でインタフェースプログラムを作成します。

##### COBOL側

Visual C++のクラスをCOBOLのクラスとして見せるためのインタフェースクラスを定義します。

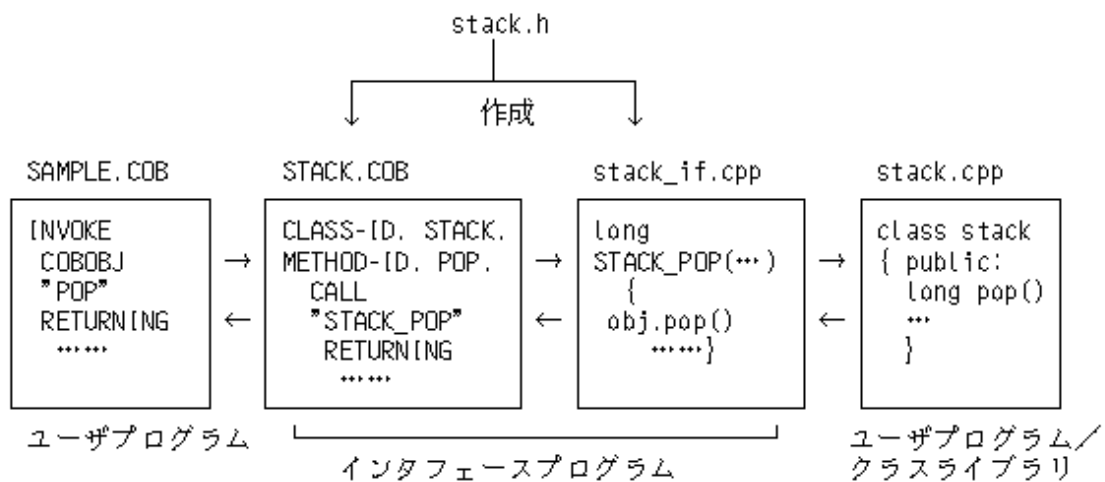
##### Visual C++側

COBOLのインタフェースクラスから呼び出されるクラスおよびオブジェクトの操作を行う関数を定義します。

それぞれのインタフェースプログラムは、Visual C++のクラス定義に依存するので、クラス定義ごとに必要になります。通常、Visual C++のクラスはヘッダファイルに定義されています。このヘッダファイルを参考にしてインタフェースプログラムを作成します。

“図18.1 インタフェースプログラムのイメージ”にインタフェースプログラムのイメージを示します。“STACK.COB”と“stack\_if.cpp”はクラス定義(stack.h)から作成します。COBOLプログラムのINVOKE文は、COBOL、Visual C++のインタフェースプログラムを中継して、最終的にVisual C++で定義したクラスのメンバ関数の呼出しになります。

図18.1 インタフェースプログラムのイメージ



### 18.4.3.3 インタフェースプログラムの仕組み

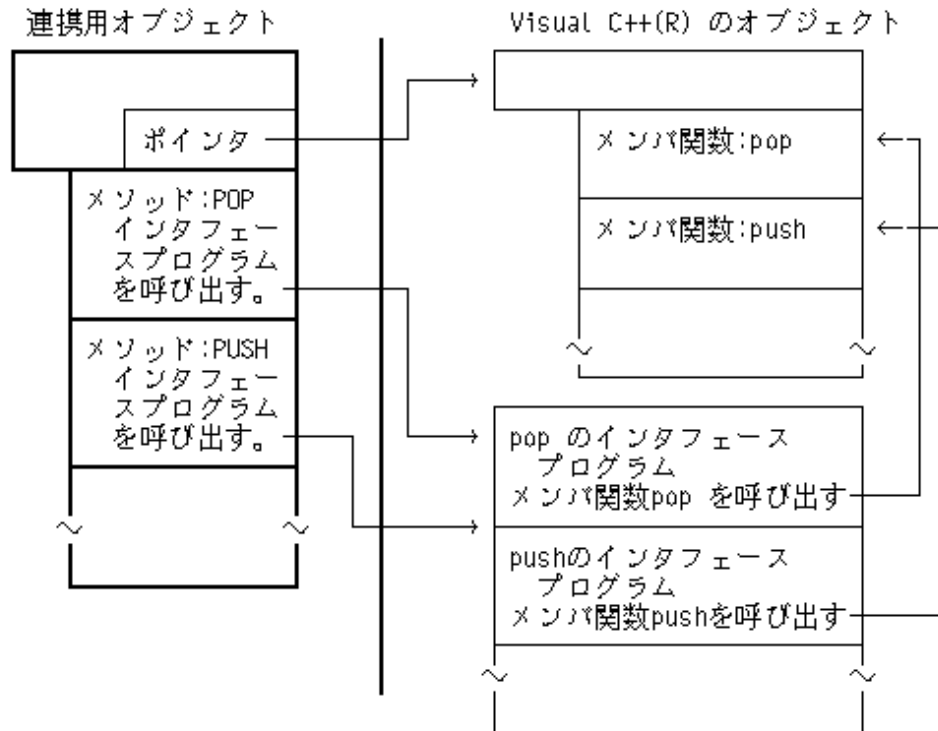
Visual C++のオブジェクトを操作する仕組みは以下になります。

#### 連携プログラムの構造

- COBOL側でVisual C++のクラス定義と同じ構造のクラスを定義します。
- COBOL側のオブジェクトは、Visual C++側のオブジェクトへのポインタを保持しておく領域を持ちます。インタフェースプログラムの呼出し時にこのポインタを引数とともに渡します。

- Visual C++側のインタフェースプログラムは、オブジェクトの対応するメンバ関数を呼び出します。

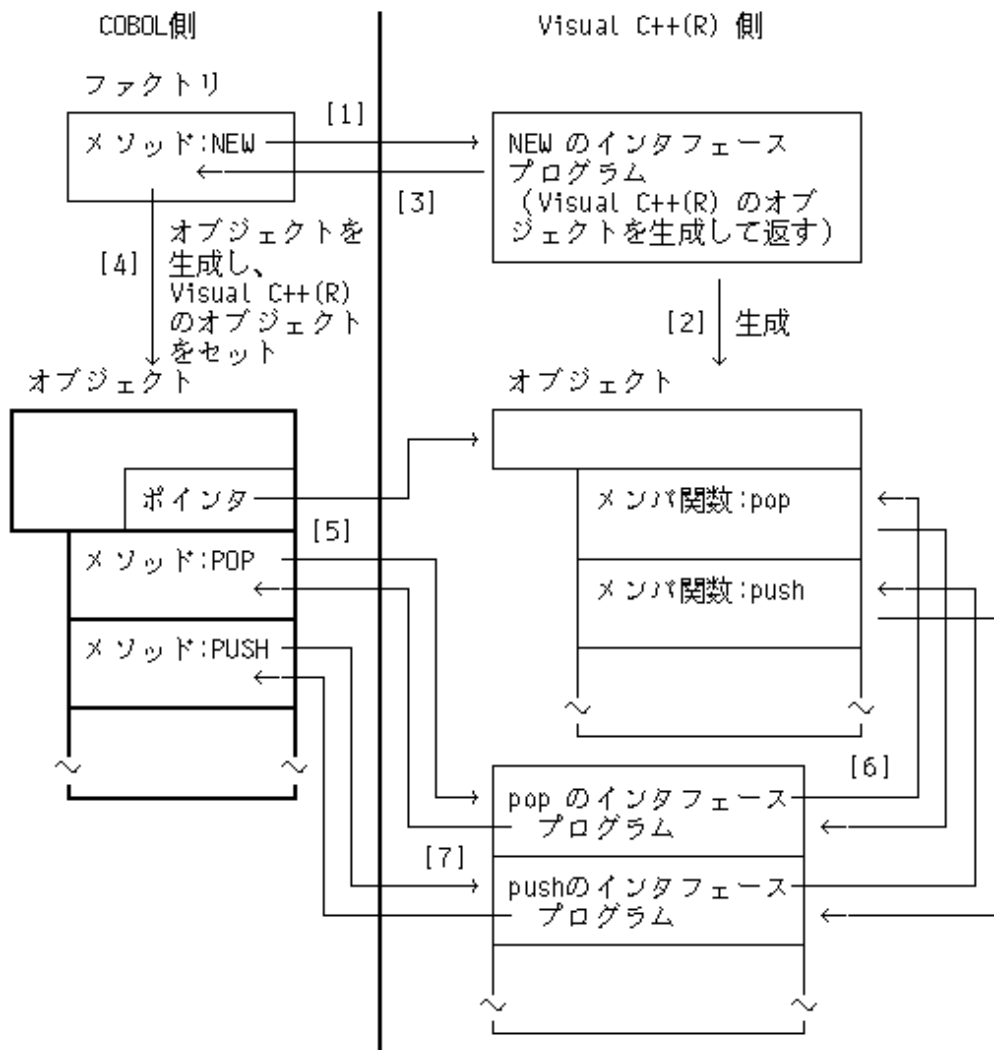
図18.2 インタフェースプログラムの構造



### 実行時の動き

実行時の制御の流れを“[図18.3 実行時の制御の流れ](#)”に示します。

図18.3 実行時の制御の流れ



COBOL側で連携用に定義したクラスのNEWメソッドが呼び出されると、NEWメソッドはVisual C++のインタフェースプログラムを呼び出します([1])。Visual C++のインタフェースプログラムは、Visual C++のオブジェクトを生成し([2])、それをCOBOLのNEWメソッドに返します([3])。NEWメソッドはさらに、COBOLの連携用のオブジェクトを生成し、オブジェクトデータとしてVisual C++のオブジェクトへのポインタを設定します([4])。

COBOLの連携用オブジェクトのメソッド呼出しは、対応するVisual C++のインタフェースプログラムを呼び出します([5])。Visual C++のインタフェースプログラムは、Visual C++で定義されたメンバ関数を呼び出し([6])、結果があればそれを返します([7])。

#### 18.4.4 Visual C++連携のプログラム手順

Visual C++連携のプログラム手順を以下に示します。

1. Visual C++で定義されているクラスを調べます。
2. COBOL側のクラス定義をします。
3. Visual C++側のインタフェースプログラムを定義します。

以下、“クラス定義の例”をもとに説明します。



## クラス定義の例

```
class stack {
public:
 unsigned long pntr;
 long pop();
 void push(long val);
private:
 long data[100];
};
```

### 18.4.4.1 Visual C++で定義されているクラスを調べる

まず、COBOLで操作するVisual C++のクラスを調べます。Visual C++で定義されているクラスのメンバ関数、メンバ変数のうち、COBOLから操作する必要のあるものだけを抜き出します。

この例では、すべてのメンバ関数、メンバ変数を操作できるようにします。すなわち、次のものが対象となります。

#### メンバ関数

- pop
- push

#### メンバ変数

- pntr

### 18.4.4.2 COBOL側での定義

COBOL側では、以下のようなクラスを定義します。



#### 参考

クラス名、メソッド名、プロパティ名などは予約語との重複、COBOLで利用できない文字が使われているなどの制約がないかぎり、Visual C++での定義と同じ名前にした方がわかりやすくなります。

#### クラス

COBOLで定義するクラス名を決めます。Visual C++側のクラスはこの名前で参照します。

この例では、STACKとします。

#### プロパティ

プロパティとして、Visual C++側のメンバ変数に対応するものを定義します。

プロパティの参照および設定では、Visual C++側のメンバ変数参照/設定インタフェースプログラムを呼び出すメソッドを自分で定義します。

この例では、プロパティとしてPNTRを定義します。

また、Visual C++側のオブジェクトを保持するためのポインタ領域を用意します。

この例では、CPP-OBJ-POINTERという名前でポインタデータを定義します。

#### ファクトリメソッド

ファクトリメソッドとしてNEWメソッドを再定義します。NEWメソッドは以下の処理を行います。

- COBOL側で自クラスのオブジェクトを生成します。
- Visual C++のオブジェクト生成インタフェースプログラムを呼び出し、Visual C++側で生成するオブジェクトを取得します。このオブジェクトへのポインタを、Visual C++側のオブジェクトを保持するための領域CPP-OBJ-POINTERに保存します。

#### オブジェクトメソッド

オブジェクトメソッドとしてVisual C++側の定義と同じ名前のメソッドを定義します。

個々のメソッドは、対応するVisual C++側のメンバ関数呼出しインタフェースプログラムを呼び出します。

この例では、オブジェクトメソッドとしてPOPメソッド、PUSHメソッドを定義します。また、Visual C++側で生成したオブジェクトを削除するDELETE-OBJメソッドを定義します。DELETE-OBJメソッドは、オブジェクト削除インタフェースプログラムを呼び出します。

### 18.4.4.3 Visual C++側での定義

Visual C++側で以下のインタフェースプログラムを定義します。

#### オブジェクト生成インタフェースプログラム

オブジェクト生成インタフェースプログラムは、COBOL側のNEWメソッドから呼び出され、Visual C++のオブジェクトを生成して返します。

#### メンバ関数呼出しインタフェースプログラム

メンバ関数呼出しインタフェースプログラムは、COBOL側の対応するメソッドからオブジェクト、メンバ関数への引数で呼び出され、オブジェクトのメンバ関数を呼び出し、その値を返します。メンバ関数呼出しプログラムは、個々のメンバ関数ごとに定義します。

#### メンバ変数参照/設定インタフェースプログラム

メンバ変数参照/設定インタフェースプログラムは、COBOL側のプロパティ参照/設定メソッドから呼び出されます。引数は、参照の場合はオブジェクト、設定の場合はオブジェクトと設定する値です。インスタンスデータ参照/設定プログラムは、メンバ変数ごとに定義します。

#### オブジェクト削除インタフェースプログラム

オブジェクト削除インタフェースプログラムは、COBOLのDELETE-OBJメソッドから呼び出され、Visual C++側で生成したオブジェクトを削除します。

表18.3 例題で定義されるインタフェースプログラム

|       | COBOL のクラス定義 | Visual C++の<br>インタフェースプログラム | Visual C++のクラス定義 |
|-------|--------------|-----------------------------|------------------|
| クラス   | STACK        | —                           | stack            |
| メソッド  | NEW          | CPP_STACK_NEW               | new              |
|       | POP          | CPP_STACK_POP               | pop              |
|       | PUSH         | CPP_STACK_PUSH              | push             |
|       | DELETE-OBJ   | CPP_STACK_DELETE_OBJ        | delete           |
| プロパティ | PNTRのGET     | CPP_STACK_GET_PNTR          | pntr(メンバ変数)      |
|       | PNTRのSET     | CPP_STACK_SET_PNTR          |                  |

### 18.4.5 COBOLからの利用

以上の定義により、“COBOLからの使用例”のようにCOBOLからVisual C++のオブジェクトを操作することができるようになります。

#### COBOLからの使用例

```
WORKING-STORAGE SECTION.
01 STACKOBJ USAGE OBJECT REFERENCE STACK.
01 POP-VALUE PIC S9(9) COMP-5.
PROCEDURE DIVISION.
 INVOKE STACK "NEW" RETURNING STACKOBJ.
 INVOKE STACKOBJ "PUSH" USING 10.
 INVOKE STACKOBJ "PUSH" USING 20.
 INVOKE STACKOBJ "PUSH" USING 30.
 INVOKE STACKOBJ "POP" RETURNING POP-VALUE.
 INVOKE STACKOBJ "DELETE-OBJ".
 SET STACKOBJ TO NULL.
 :
```

### 18.4.6 サンプルプログラム

ここで説明したインタフェースプログラムの定義の例を“Visual C++連携のCOBOLのクラス定義”および“Visual C++連携のVisual C++側のインタフェースプログラム”に示します。

## Visual C++連携のCOBOLのクラス定義

```

*>>
*> CLASS STACKを定義する
*>>
IDENTIFICATION DIVISION.
CLASS-ID. STACK INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS FJBASE.
*>
*> FACTORYの定義
*> NEWメソッドを再定義する
IDENTIFICATION DIVISION.
FACTORY.
PROCEDURE DIVISION.
IDENTIFICATION DIVISION.
METHOD-ID. NEW OVERRIDE.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 CPP-STACK USAGE IS POINTER.
LINKAGE SECTION.
 01 STACKOBJ USAGE OBJECT REFERENCE SELF.
PROCEDURE DIVISION RETURNING STACKOBJ.
 INVOKE SUPER "NEW" RETURNING STACKOBJ.
 CALL "CPP_STACK_NEW" RETURNING CPP-STACK.
 INVOKE STACKOBJ "SET-CPP-OBJ-POINTER" USING CPP-STACK.
 EXIT METHOD.
END METHOD NEW.
END FACTORY.
*>
*> OBJECTの定義
*> オブジェクトデータとしC++側のオブジェクトを保持するデータ
*> (CPP-OBJ-POINTER)を定義する
IDENTIFICATION DIVISION.
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 CPP-OBJ-POINTER USAGE IS POINTER.
PROCEDURE DIVISION.
*>
*> SET-CPP-OBJ-POINTERメソッドの定義
*>
IDENTIFICATION DIVISION.
METHOD-ID. SET-CPP-OBJ-POINTER.
DATA DIVISION.
LINKAGE SECTION.
 01 USE-VALUE USAGE IS POINTER.
PROCEDURE DIVISION USING USE-VALUE.
 MOVE CPP-OBJ-POINTER TO USE-VALUE.
 EXIT METHOD.
END METHOD SET-CPP-OBJ-POINTER.
*>
*> POPメソッドの定義
*>
IDENTIFICATION DIVISION.
METHOD-ID. POP.

```

```

DATA DIVISION.
LINKAGE SECTION.
 01 RET-VALUE PIC S9(9) COMP-5.
PROCEDURE DIVISION RETURNING RET-VALUE.
 CALL "CPP_STACK_POP" USING CPP-OBJ-POINTER RETURNING RET-VALUE.
 EXIT METHOD.
END METHOD POP.
*>
*> PUSHメソッドの定義
*>
IDENTIFICATION DIVISION.
METHOD-ID. PUSH.
DATA DIVISION.
LINKAGE SECTION.
 01 SET-VALUE PIC S9(9) COMP-5.
PROCEDURE DIVISION USING SET-VALUE.
 CALL "CPP_STACK_PUSH" USING CPP-OBJ-POINTER SET-VALUE.
 EXIT METHOD.
END METHOD PUSH.
*>
*> PNTRを参照するメソッドの定義
*>
IDENTIFICATION DIVISION.
METHOD-ID. GET PROPERTY PNTR.
DATA DIVISION.
LINKAGE SECTION.
 01 RET-VALUE PIC S9(9) COMP-5.
PROCEDURE DIVISION RETURNING RET-VALUE.
 CALL "CPP_STACK_GET_PNTR" USING CPP-OBJ-POINTER RETURNING RET-VALUE.
 EXIT METHOD.
END METHOD.
*>
*> PNTRを設定メソッドの定義
*>
IDENTIFICATION DIVISION.
METHOD-ID. SET PROPERTY PNTR.
DATA DIVISION.
LINKAGE SECTION.
 01 SET-VALUE PIC S9(9) COMP-5.
PROCEDURE DIVISION USING SET-VALUE.
 CALL "CPP_STACK_SET_PNTR" USING CPP-OBJ-POINTER SET-VALUE.
 EXIT METHOD.
END METHOD.
*>
*> DELETE-OBJメソッドの定義
*>
IDENTIFICATION DIVISION.
METHOD-ID. DELETE-OBJ.
DATA DIVISION.
LINKAGE SECTION.
PROCEDURE DIVISION.
 CALL "CPP_STACK_DELETE_OBJ" USING CPP-OBJ-POINTER.
 EXIT METHOD.
END METHOD DELETE-OBJ.
END OBJECT.
END CLASS STACK.

```

#### Visual C++連携のVisual C++側のインタフェースプログラム

```

#include "stack.h"

extern "C" stack* CPP_STACK_NEW() {
 return new stack;
}

```

```

}

extern "C" long int CPP_STACK_POP(stack** stk) {
 return (*stk)->pop();
}

extern "C" void CPP_STACK_PUSH(stack** stk, long int* value) {
 (*stk)->push(*value);
}

extern "C" long int CPP_STACK_GET_PNTR(stack** stk) {
 return (*stk)->pointer;
}

extern "C" void CPP_STACK_SET_PNTR(stack** stk, long int* value) {
 (*stk)->pointer = *value;
}

extern "C" void CPP_STACK_DELETE_OBJ(stack** stk) {
 delete *stk;
}

```

## 18.5 オブジェクトの永続化

ここでは、オブジェクトの永続化について説明します。

### 18.5.1 オブジェクトの永続化とは

COBOLで生成したオブジェクトは、プログラムの終了とともに消滅します。しかし、実用的なアプリケーションでは、プログラム中で生成したオブジェクトを後で参照したり、他のプログラムで参照したりする必要があります。この実行単位を超えて存在するオブジェクトを「永続オブジェクト」と呼びます。

### 18.5.2 概要

オブジェクトの永続化は、COBOLの実行中に生成したオブジェクトを外部記憶装置に保存し、別の実行単位で読み戻すことで実現します。保存のための記憶装置には、データベース、ファイルなどが用いられます([参照]“[図18.4 永続オブジェクトの流れ](#)”)。

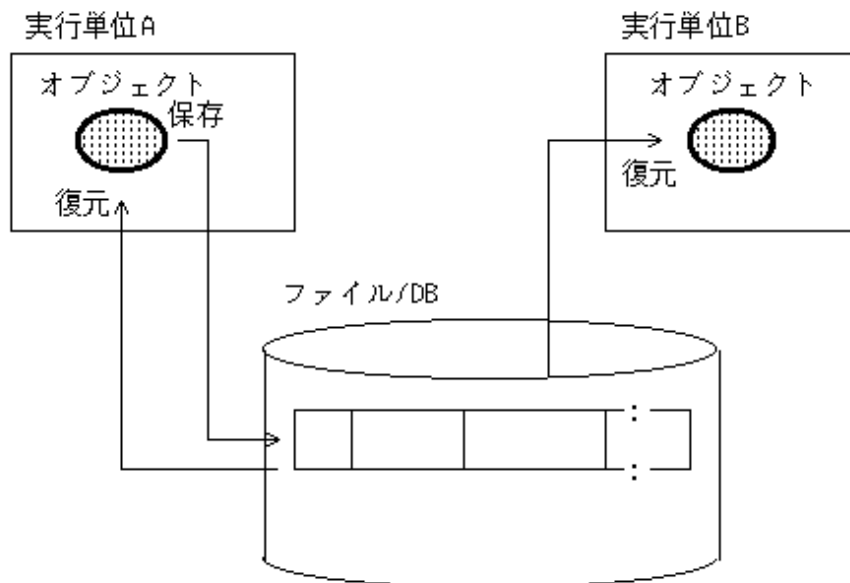
オブジェクトには適当な識別子を付け、その識別子を元に保存/復元を行います。



#### 参考

オブジェクトの保存/復元は、実際にはオブジェクトデータの保存/復元によって実現します。しかし、オブジェクトに対して保存メソッドを呼び出したり、読み戻したデータを元にオブジェクトを生成したりするので、ここでの説明では、オブジェクトデータの保存/復元ではなく、オブジェクトの保存/復元ということで説明します。

図18.4 永続オブジェクトの流れ



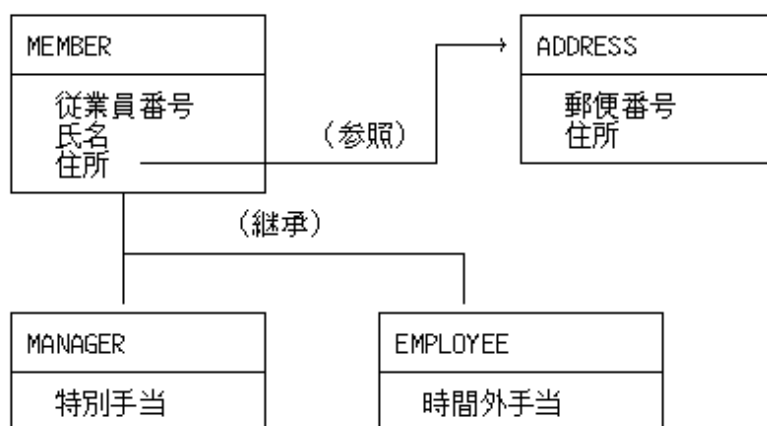
ここでは、索引ファイルを利用してオブジェクトの永続化を行う方法について説明します。

データベースを利用してオブジェクトの永続化を行う場合は、“ファイル”を“データベースの表”に置き換えてお読みください。[参照]  
[“22.2.6 オブジェクト指向プログラミング機能を使用したデータベースアクセス”](#)

### 18.5.3 クラス構造の例

本節の説明で使用するクラス構造は、“[図18.5 クラス構造の例](#)”のようになっています。以下の説明では、この図を基に説明しますので、必要に応じて参照してください。ここで使うクラスは、説明に関係がないので、メソッドについては省略してあります。また、オブジェクトデータについても、説明に必要な最小限のものにしてあります。

図18.5 クラス構造の例



クラスには、従業員全体を表すMEMBER、管理職を表すMANAGER、一般従業員を表すEMPLOYEE、住所を表すADDRESSがあります。

- MEMBERは、オブジェクトデータとして、従業員番号、氏名、住所を持ちます。住所はADDRESSオブジェクトに関連付けられます。

- ・ MANAGERは、MEMBERを継承し、オブジェクトデータとして特別手当を持ちます。
- ・ EMPLOYEEは、MEMBERを継承し、オブジェクトデータとして時間外手当を持ちます。
- ・ ADDRESSは、オブジェクトデータとして、郵便番号、住所を持ちます。ADDRESSクラスは、他のクラスとの継承関係はありません。

## 18.5.4 索引ファイルとオブジェクトの対応

ここでは、索引ファイルとオブジェクトの対応について説明します。

### 18.5.4.1 クラスとファイルの対応

以下の説明では、関連性のあるオブジェクトを保存するファイルをどのように分割するかを説明します。

保存するオブジェクトとファイルの対応には、いくつかのモデルがあります。

#### クラスごとに保存ファイルを分ける方法

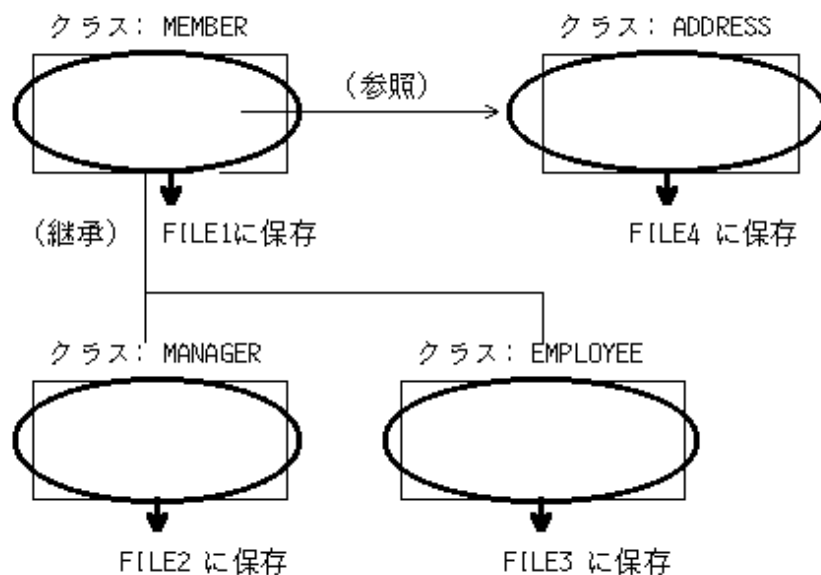
オブジェクトと保存ファイルの対応の基本は、クラスとファイルを一対一に対応付けることです。この例では、MEMBERをFILE1に、MANAGERをFILE2に、EMPLOYEEをFILE3に、ADDRESSをFILE4に保存することになります([参照]“[図18.6 クラスごとに異なるファイルに保存する場合](#)”。

MANAGERクラスのオブジェクトを保存する場合、MANAGER固有のオブジェクトデータをFILE2に、MEMBER固有のオブジェクトデータはFILE1に保存します。

MEMBERオブジェクトを復元する場合、最初にFILE1からMEMBER固有のオブジェクトデータを読み込み、それがMANAGERであればFILE2からMANAGER固有のデータ読み込んで、最終的にMANAGERオブジェクトを生成して返します。

この方法では、クラスごとにファイル定義が独立しているため、クラス定義の変更、新しいクラスの追加があった場合、そのクラスに対応したファイルだけを修正すればよく、保守性が向上します。ただし、クラスの数だけファイルが必要になるため管理が難しくなるという特徴があります。

図18.6 クラスごとに異なるファイルに保存する場合



#### 親クラスのオブジェクトデータも含めて1つのファイルに保存する方法

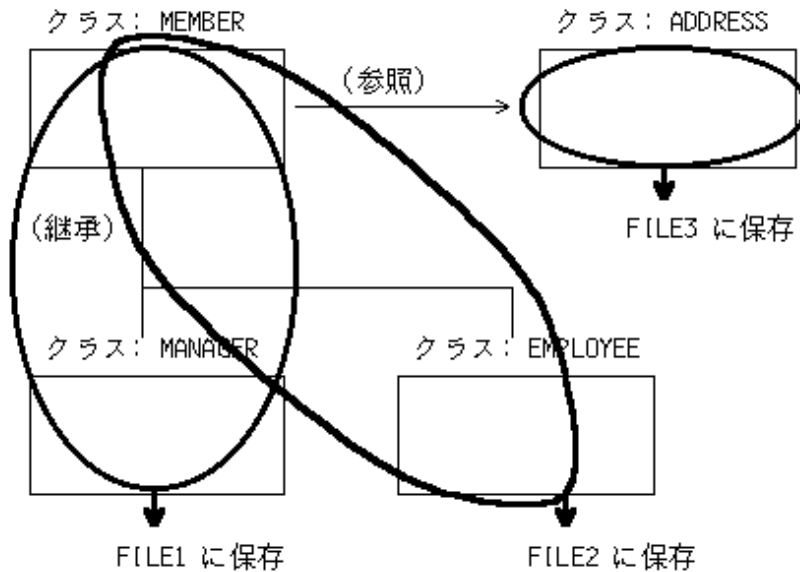
MEMBERクラスが純粋に抽象クラスとして定義されている場合、MEMBERクラスとして独立したオブジェクトは存在しません。この場合、MANAGERクラスの保存では、MEMBERクラスのオブジェクトデータとMANAGERクラスのオブジェクトデータとを同じファイルに保存した方が、クラスごとに保存ファイルを分ける方法に比べて処理が簡単になります。EMPLOYEEクラスについても同じように、

MEMBERクラスのオブジェクトデータとEMPLOYEEクラスのオブジェクトデータを同じファイルに保存することができます。[参照]“[図18.7 親クラスのオブジェクトデータも含めて1つのファイルに保存する方法](#)”

MANAGERクラスのオブジェクトを復元する場合、FILE1からオブジェクトデータを読み込み、MANAGERオブジェクトを生成して返します。しかし、MANAGERであるか、EMPLOYEEであるかは、クラスが持っている情報であり、オブジェクトを復元してはじめてわかる情報です。たとえば、従業員番号の情報だけでは、FILE1とFILE2のどちらのファイルからオブジェクトデータを読み戻したらよいのか判断できません。

この方法は、保存するクラスが1つしかない場合には有効な方法ですが、“[図18.7 親クラスのオブジェクトデータも含めて1つのファイルに保存する方法](#)”の説明のように、親クラスを複数のクラスが継承している場合、うまく処理できない場合があります。また、クラス定義の変更が継承関係にあるクラスのファイルも変更しなければならなくなります。

図18.7 親クラスのオブジェクトデータも含めて1つのファイルに保存する方法

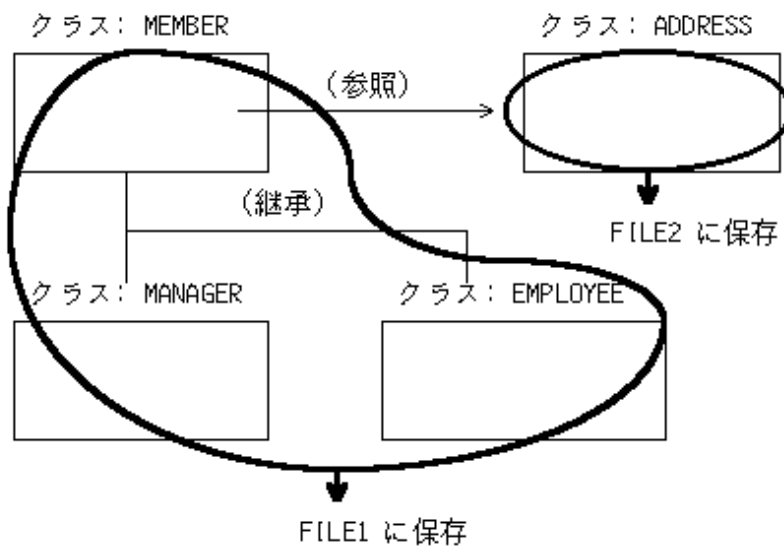


#### 同じ親クラスを持つクラスを同じファイルに保存する方法

“[図18.7 親クラスのオブジェクトデータも含めて1つのファイルに保存する方法](#)”のように同じ親クラスを継承しているクラスを別のファイルにするのではなく、同じファイルに保存することで、欠点を補うことができます。この方法では、従業員番号から、MANAGER、EMPLOYEEの適切なオブジェクトを復元することができるようになります。



図18.8 同じ親クラスを持つクラスを同じファイルに保存する方法



以上の3つの方法のどれが適しているかは、アプリケーションの性質に依存します。

ここでは、“[図18.8 同じ親クラスを持つクラスを同じファイルに保存する方法](#)”に従って説明します。

#### 18.5.4.2 索引ファイルの定義

オブジェクトの識別子は、オブジェクトに対して一意に決まるものでなければなりません。この例ではMEMBERおよびそのクラスを継承しているMANAGER、EMPLOYEEは、従業員番号をオブジェクトの識別子とし、索引ファイルの主キーとして利用します。また、索引ファイル中のレコードにMANAGERであるか、EMPLOYEEであるかを区別するための領域を設けます。この例では、MANAGERを1、EMPLOYEEを2とします。

ADDRESSクラスのオブジェクトの識別子として、オブジェクトを区別するオブジェクトIDを付ける必要があります。この例では、ADDRESSクラスのオブジェクトはMEMBERクラスのオブジェクトのオブジェクトデータとしてしか参照されることはないため、そのアドレスの従業員の従業員番号を識別子として利用します。

図18.9 索引ファイル中のレコード

|          | 主キー    | クラス識別領域 |    |        |       |
|----------|--------|---------|----|--------|-------|
|          |        | ↓       |    |        |       |
| MANAGER  | 従業員番号  | 1       | 氏名 | アドレスID | 特別手当  |
| EMPLOYEE | 従業員番号  | 2       | 氏名 | アドレスID | 時間外手当 |
| ADDRESS  | アドレスID | 郵便番号    | 住所 |        |       |

アドレスIDは、アドレスオブジェクトを一意に識別するための識別子です。

#### 18.5.5 オブジェクトの保存/復元

ここでは、オブジェクトの保存および復元について説明します。

### 18.5.5.1 索引ファイル操作クラス

オブジェクトの保存/復元のためのメソッドをそれぞれSAVE、RETRIEVEとします。

SAVE/RETRIEVEメソッドの中で索引ファイルをオープン/クローズしてレコードの書込み/読出しを行う方法もあります。しかし、SAVE/RETRIEVEのたびにファイルのオープン/クローズが行われるため、効率のよい方法ではありません。この例では、保存/復元のためのファイルを扱うクラスを用意し、プログラムの開始時にオープンし、終了時にクローズするようにします。

索引ファイル操作クラスのオブジェクトメソッドには次のメソッドを定義します。

#### RETRIEVE

引数に識別子を受け取り、その識別子のオブジェクトを復元して返します。

#### OPEN-DATA-FILE

保存するファイルをオープンします。

#### CLOSE-DATA-FILE

保存するファイルをクローズします。

#### SAVE

引数に保存するオブジェクトを受け取り、そのオブジェクトをファイルに保存します。

索引ファイル操作クラスは、保存に利用するファイルの数だけ定義します。

### 18.5.5.2 保存するオブジェクトのメソッドの追加

保存するクラスには、以下のメソッドを追加します。

#### ファクトリメソッド

##### RETRIEVE

RETRIEVEメソッドは、オブジェクトの識別子を引数で受け取り、該当するオブジェクトを復元して返します。実際には、対応する索引ファイル操作クラスのRETRIEVEメソッドを呼び出します。RETRIEVEメソッドは、親クラスで定義すれば十分で、個々のクラスに定義する必要はありません。

#### オブジェクトメソッド

##### SAVE

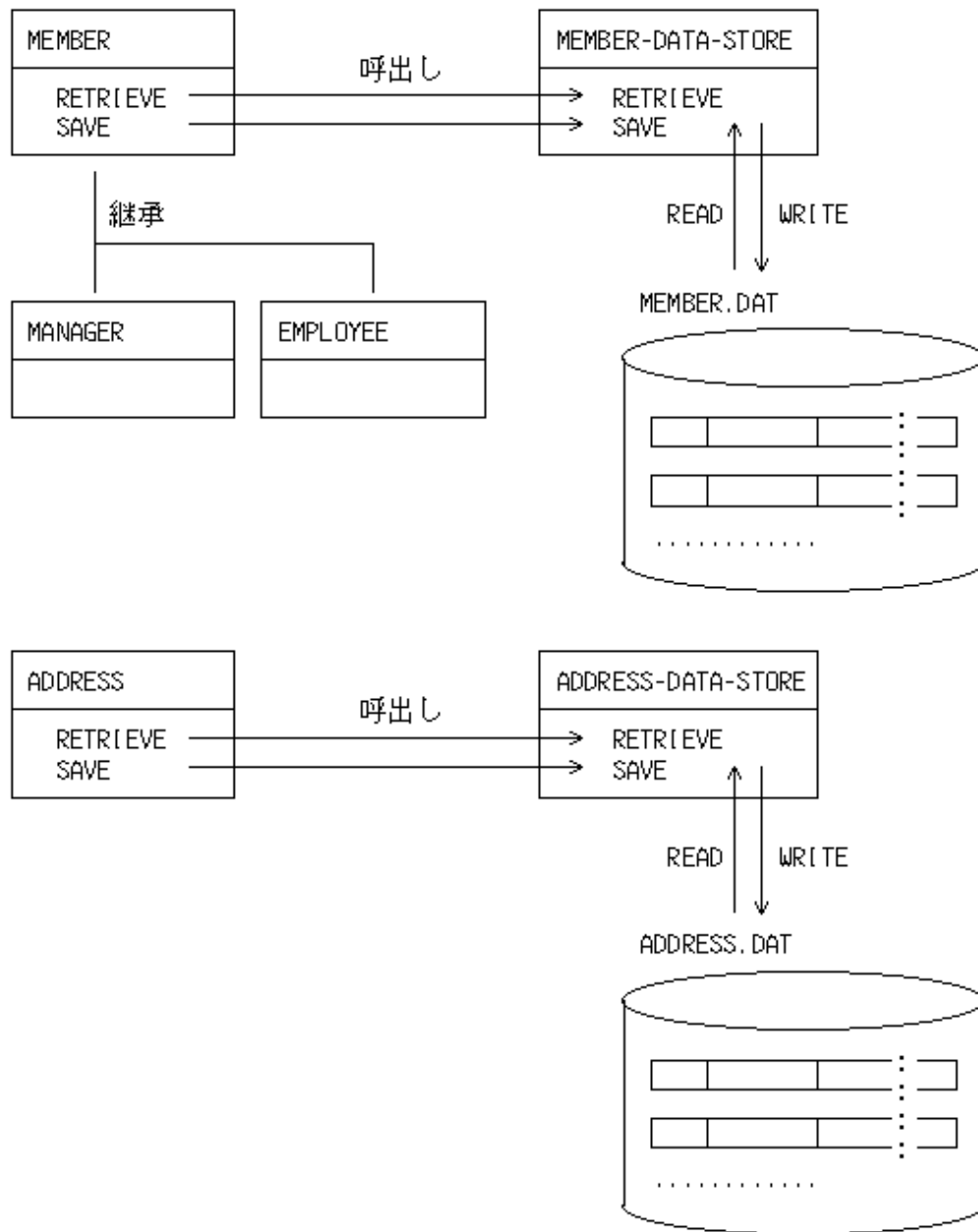
SAVEメソッドはオブジェクトを保存するメソッドです。実際には、自分自身を引数として、対応する索引ファイル操作クラスのSAVEメソッドを呼び出します。SAVEメソッドは、親クラスで定義すれば十分で、個々のクラスに定義する必要はありません。

本節で利用している例を基に、保存するクラス、索引ファイル名、索引ファイル操作クラスについて、“[表18.4 保存クラス/索引ファイル/索引ファイル操作クラス](#)”に示します。また、メソッドの呼出し関係、索引ファイルを“[図18.9 索引ファイル中のレコード](#)”に示します。

表18.4 保存クラス/索引ファイル/索引ファイル操作クラス

| 保存するクラス  | 親クラス   | 索引ファイル名     | 索引ファイル操作クラス        |
|----------|--------|-------------|--------------------|
| MANAGER  | MEMBER | MEMBER.DAT  | MEMBER-DATA-STORE  |
| EMPLOYEE |        |             |                    |
| ADDRESS  | —      | ADDRESS.DAT | ADDRESS-DATA-STORE |

図18.10 索引ファイル操作クラスとの関係



### 18.5.5.3 処理の流れ

保存/復元の処理の流れを“[図18.10 索引ファイル操作クラスとの関係](#)”に従って説明します。

#### 保存

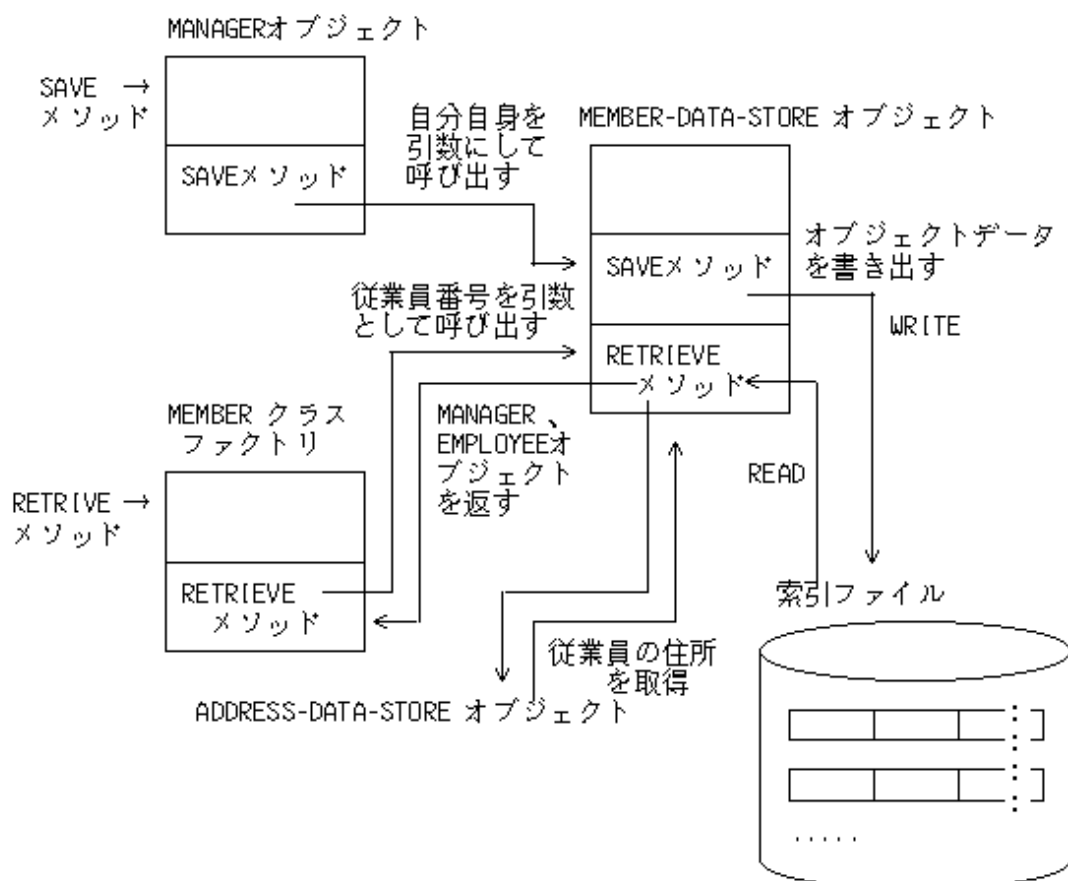
MANAGERオブジェクトに対してSAVEメソッドが呼ばれると、SAVEメソッドは自分自身を引数にして、MEMBER用の索引ファイル操作オブジェクト(MEMBER-DATA-STOREオブジェクト)のSAVEメソッドを呼び出します。MEMBER-DATA-STOREオブジェクトのSAVEメソッドは、引数のオブジェクトのクラスを調べ(この場合MANAGERクラスになる)、MANAGERクラス用のレコードにデータを転記し、索引ファイルに書き込みます。

#### 復元

RETRIEVEメソッドは、対応する索引ファイル操作オブジェクト(MEMBER-DATA-STOREオブジェクト)に対して、従業員番号を引数として、RETRIEVEメソッドを呼び出します。MEMBER-DATA-STOREオブジェクトのRETRIEVEメソッドは、索引ファイルから、

従業員番号を主キーとしてレコードを読み込み、副キーを調べて、MANAGERクラスまたはEMPLOYEEクラスのオブジェクトを返します。また、RETRIEVEメソッドは、従業員番号を引数として、ADDRESS-DATA-STOREオブジェクトのRETRIEVEメソッドを呼び出し、その従業員の住所(ADDRESS)オブジェクトを取得します。取得したADDRESSオブジェクトをオブジェクトデータの住所に転記します(ADDRESSオブジェクトの復元については“[図18.11 保存/復元の処理の流れ](#)”では省略)。

図18.11 保存/復元の処理の流れ



## 18.6 特殊クラス

ここでは、特殊クラスを使用したアプリケーションの概要および使用方法について説明します。

なお、\*COM特殊クラスについては、“[第26章 COM機能](#)”を参照してください。

### 18.6.1 \*COB-BINDTABLEクラス

ここでは、業務用の各種クラスライブラリとの連携で、特殊クラス「\*COB-BINDTABLE」を使用して、COBOLの変数をクラスライブラリに登録および通知する方法について説明します。

#### 18.6.1.1 概要

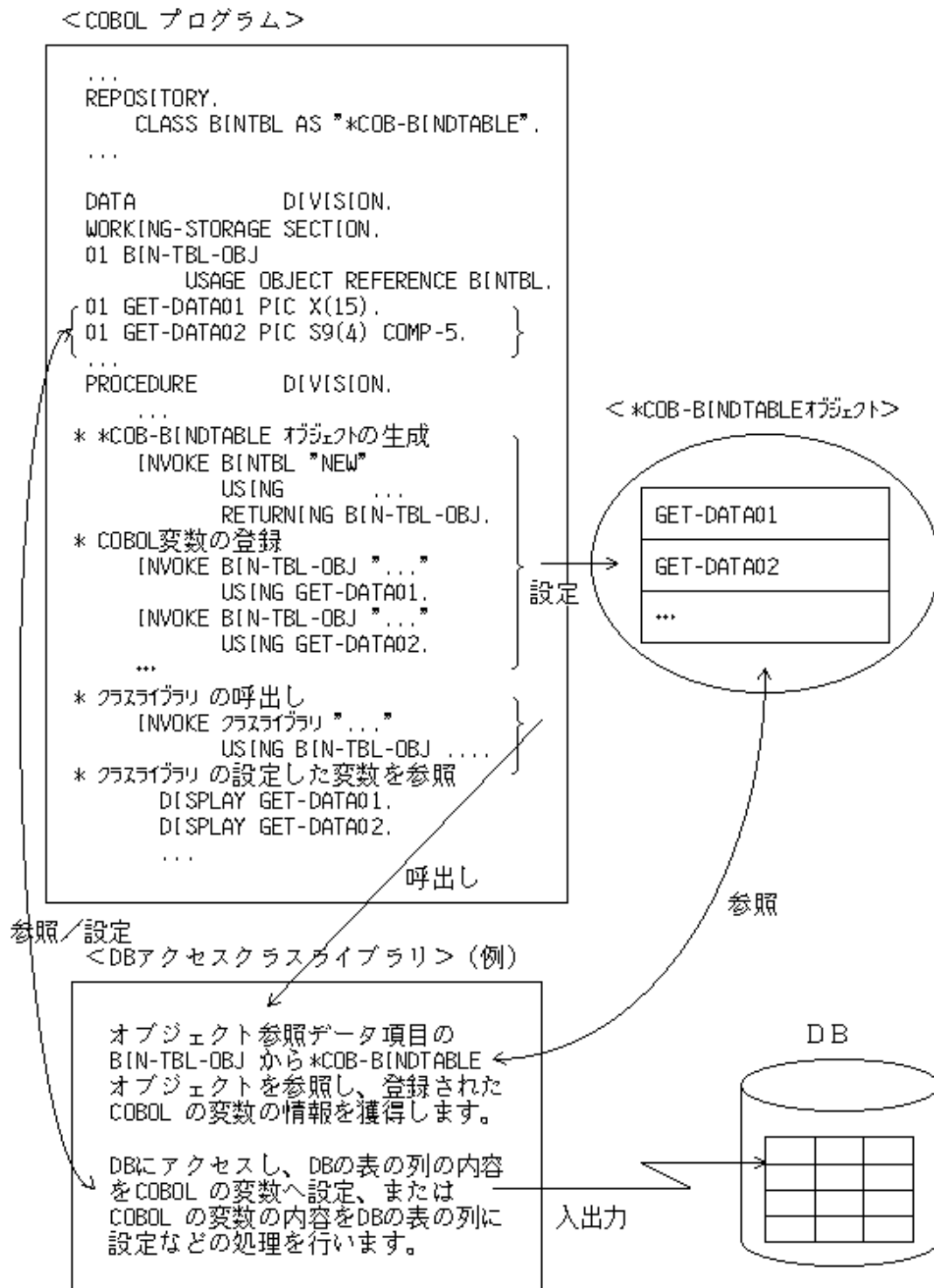
NetCOBOLのアプリケーションの作成のために、業務用の各種クラスライブラリが提供されます。各種クラスライブラリを利用することで、データベースのアクセスなどの業務用アプリケーションを簡単に作成することができます。

クラスライブラリのメソッドの中には、使用するCOBOLの変数の情報を、メソッド呼出し時にアプリケーションから通知しなくてはならないものがあります。

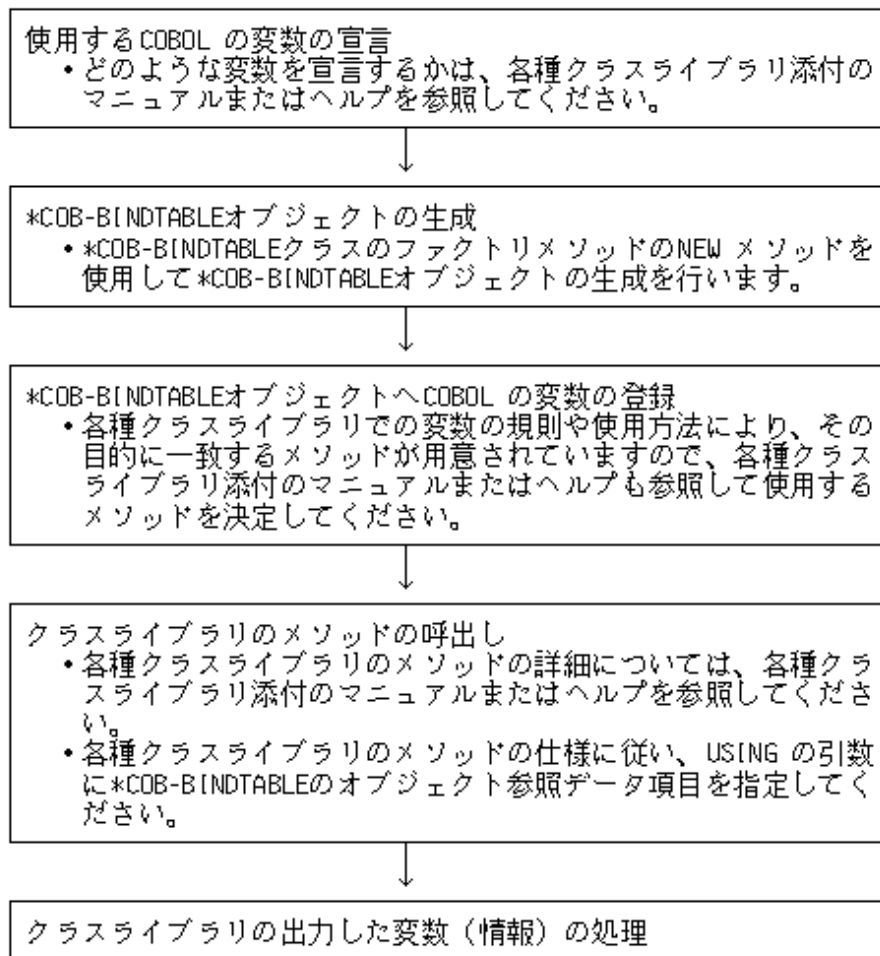
たとえば、データベースアクセスのためのクラスライブラリのデータ参照のメソッドを呼び出す場合、データベースから取り出したデータを受け取るためのCOBOL変数をアプリケーションから通知する必要があります。

COBOLの変数をクラスライブラリに通知するには、特殊クラス\*COB-BINDTABLEを使用します。

以下に、データベースアクセス用クラスライブラリを例とした概要図を示します。



特殊クラス\*COB-BINDTABLEを使用するアプリケーションの処理の流れおよび補足事項を以下に示します。



### 18.6.1.2 \*COB-BINDTABLE クラスの説明

ここでは、\*COB-BINDTABLE クラスのメソッドについて説明します。

#### 18.6.1.2.1 \*COB-BINDTABLE クラスのファクトリメソッド

ファクトリメソッドには、以下のメソッドがあります。

- NEW メソッド

#### NEW メソッド

NEW メソッドは、\*COB-BINDTABLE のオブジェクトの生成を行います。

NEW メソッドの機能および引数を以下に示します。

| メソッド名 | 引数                     | 引数の型                             | 意味                                                                                       |
|-------|------------------------|----------------------------------|------------------------------------------------------------------------------------------|
| NEW   | USING 第1<br>最大登録データ項目数 | S9(9) BINARY   COMP-5            | USING の第1引数で指定されたデータ項目の値を登録データ項目数の最大個数としてオブジェクトを作成します。<br>また、そのオブジェクトのオブジェクト参照データを返却します。 |
|       | RETURNING<br>作成オブジェクト  | USAGE OBJECT<br>REFERENCE 特殊クラス名 |                                                                                          |

## 使用規則

登録するデータ項目の個数の最大値を第1引数で指定しなければなりません。

## 使用例

```
....
REPOSITORY.
 CLASS BINTBL AS "*COB-BINDTABLE".
....
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ-SPCL USAGE OBJECT REFERENCE BINTBL.
01 ELM-NO PIC S9(9) COMP-5.
....
として
 MOVE 20 TO ELM-NO.
 INVOKE BINTBL "NEW" USING ELM-NO
 RETURNING OBJ-SPCL.
```

### 18.6.1.2.2 \*COB-BINDTABLEクラスのオブジェクトメソッド

オブジェクトメソッドには、以下のメソッドがあります。

- REGISTERメソッド
- REGISTER-INDICATORメソッド
- REGISTER-VARYINGメソッド
- REGISTER-VARYING-INDICATORメソッド
- REGISTER-LARGE-DATAメソッド
- REGISTER-LARGE-DATA-INDICATORメソッド

それぞれのメソッドについて、以下に説明します。

## REGISTERメソッド

REGISTERメソッドは、データ項目の登録を行います。

REGISTERメソッドの機能および引数を以下に示します。

| メソッド名    | 引数        | 引数の型     | 意味                            |
|----------|-----------|----------|-------------------------------|
| REGISTER | USING 第1  | 任意(基本項目) | USING の第1引数で指定されたデータ項目を登録します。 |
|          | 登録データ項目   |          |                               |
|          | RETURNING | —        |                               |
|          | なし        |          |                               |

## 使用規則

1. このメソッドを呼び出す順序で第1引数に指定されたデータ項目が登録されるため、呼び出す順序を考慮する必要があります。
2. 第1引数は基本項目でなければなりません。
3. 第1引数に指定するデータ項目の型については、各種クラスライブラリの取り扱い可能なデータ型でなければなりません。

## 使用例

```
01 OBJ-SPCL USAGE OBJECT REFERENCE BINTBL.
01 GET-DATA1 PIC X(15).
01 GET-DATA2 PIC X(20).
```

```
01 ELM-NO PIC S9(4) COMP-5 VALUE 2.
```

として

```
INVOKE BINTBL "NEW" USING ELM-NO
 RETURNING OBJ-SPCL.
INVOKE OBJ-SPCL "REGISTER" USING GET-DATA1.
INVOKE OBJ-SPCL "REGISTER" USING GET-DATA2.
```

## REGISTER-INDICATORメソッド

REGISTER-INDICATORメソッドは、データ項目の登録を行います。また、登録するデータ項目に対応する標識変数の指定を行います。

REGISTER-INDICATORメソッドの機能および引数を以下に示します。

| メソッド名              | 引数                  | 引数の型                      | 意味                                                           |
|--------------------|---------------------|---------------------------|--------------------------------------------------------------|
| REGISTER-INDICATOR | USING 第1<br>登録データ項目 | 任意(基本項目)                  | USING の第1引数で指定されたデータ項目を登録します。<br>登録するデータ項目に対応する標識変数の指定を行います。 |
|                    | USING 第2<br>標識変数    | PIC S9(4) BINARY   COMP-5 |                                                              |
|                    | RETURNING<br>なし     | —                         |                                                              |
|                    |                     |                           |                                                              |

### 使用規則

- 第1引数は、REGISTERメソッドの使用規則に従わなければなりません。
- 第2引数は、符号付き2進2バイトのデータ項目でなければなりません。
- 標識変数の使用方法および設定する値、設定される値については、各クラスライブラリの使用規則に従います。

### 使用例

```
01 OBJ-SPCL USAGE OBJECT REFERENCE BINTBL.
01 GET-DATA1 PIC X(15).
01 GET-DATA2 PIC X(20).
01 IND2 PIC S9(4) COMP-5.
01 ELM-NO PIC S9(4) COMP-5 VALUE 2.
```

として

```
INVOKE BINTBL "NEW" USING ELM-NO
 RETURNING OBJ-SPCL.
INVOKE OBJ-SPCL "REGISTER" USING GET-DATA1.
INVOKE OBJ-SPCL "REGISTER-INDICATOR"
 USING GET-DATA2
 IND2.
```

## REGISTER-VARYINGメソッド

REGISTER-VARYINGメソッドは、可変長データに対応するCOBOLのデータ項目の登録を行います。

REGISTER-VARYINGメソッドの機能および引数を以下に示します。



| メソッド名            | 引数                     | 引数の型                             | 意味                                                             |
|------------------|------------------------|----------------------------------|----------------------------------------------------------------|
| REGISTER-VARYING | USING 第1<br>可変長データ長さ部  | PIC S9(4   9) BINARY  <br>COMP-5 | USING の第1引数で指定されたデータ項目を長さ部、第2引数に指定されたデータ項目をデータ部とするデータ項目を登録します。 |
|                  | USING 第2<br>可変長データデータ部 | PIC X(p)   PIC N(q)              |                                                                |
|                  | RETURNING<br>なし        | —                                |                                                                |

#### 使用規則

1. 基本的な規則は、REGISTERメソッドの規則に従わなければなりません。
2. 第1引数と第2引数で指定するデータ項目は、以下の形式の同じ集団項目に従属する基本項目1、基本項目2でなければなりません。

|                                          |
|------------------------------------------|
| nn 集団項目 1.                               |
| mm 基本項目 1 PIC S9(4   9) BINARY   COMP-5. |
| mm 基本項目 2 PIC X(p)   N (q).              |

- 基本項目1は、符号付き2進でなければなりません。
- 基本項目1および基本項目2には、上記に記述されていない句を指定してはなりません。

3. 登録するデータ項目は1つであるとみなします。

#### 使用例

|                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 01 OBJ-SPCL USAGE OBJECT REFERENCE BINTBL.<br>01 GET-DATA1.<br>02 GET-DATA1-LNG PIC S9(4) BINARY.<br>02 GET-DATA1-DAT PIC X(255).<br>01 GET-DATA2 PIC X(20).<br>01 ELM-NO PIC S9(4) COMP-5 VALUE 2. |
| として                                                                                                                                                                                                 |
| INVOKE BINTBL "NEW" USING ELM-NO<br>RETURNING OBJ-SPCL.                                                                                                                                             |
| INVOKE OBJ-SPCL "REGISTER-VARYING" USING GET-DATA1-LNG<br>GET-DATA1-DAT.                                                                                                                            |
| INVOKE OBJ-SPCL "REGISTER" USING GET-DATA2.                                                                                                                                                         |

### REGISTER-VARYING-INDICATORメソッド

REGISTER-VARYING-INDICATORメソッドは、可変長データに対応するCOBOLのデータ項目の登録を行います。また、登録するデータ項目に対応する標識変数の指定を行います。

REGISTER-VARYING-INDICATORメソッドの機能および引数を以下に示します。

| メソッド名                      | 引数                     | 引数の型                             | 意味                                                                                            |
|----------------------------|------------------------|----------------------------------|-----------------------------------------------------------------------------------------------|
| REGISTER-VARYING-INDICATOR | USING 第1<br>可変長データ長さ部  | PIC S9(4   9) BINARY  <br>COMP-5 | USING の第1引数で指定されたデータ項目を長さ部、第2引数に指定されたデータ項目をデータ部とするデータ項目を登録します。<br>登録するデータ項目に対応する標識変数の指定を行います。 |
|                            | USING 第2<br>可変長データデータ部 | PIC X(p)   PIC N(q)              |                                                                                               |

| メソッド名 | 引数               | 引数の型                      | 意味 |
|-------|------------------|---------------------------|----|
|       | USING 第3<br>標識変数 | PIC S9(4) BINARY   COMP-5 |    |
|       | RETURNING<br>なし  | —                         |    |

#### 使用規則

1. 第1引数および第2引数は、REGISTER-VARYINGメソッドの使用規則に従わなければなりません。
2. 第3引数は、符号付き2進2バイトのデータ項目でなければなりません。
3. 標識変数の使用方法および設定する値、設定される値については、各クラスライブラリの使用規則に従います。

#### 使用例

```

01 OBJ-SPCL USAGE OBJECT REFERENCE BINTBL.
01 GET-DATA1.
 02 GET-DATA1-LNG PIC S9(4) BINARY.
 02 GET-DATA1-DAT PIC X(255).
01 GET-DATA2 PIC X(20).
01 IND1 PIC S9(4) COMP-5.
01 ELM-NO PIC S9(4) COMP-5 VALUE 2.

```

として

```

 INVOKE BINTBL "NEW" USING ELM-NO
 RETURNING OBJ-SPCL.
 INVOKE OBJ-SPCL "REGISTER-VARYING-INDICATOR"
 USING GET-DATA1-LNG
 GET-DATA1-DAT
 IND1.
 INVOKE OBJ-SPCL "REGISTER" USING GET-DATA2.

```

### REGISTER-LARGE-DATAメソッド

REGISTER-LARGE-DATAメソッドは、バイナリデータに対応するCOBOLのデータ項目の登録を行います。

REGISTER-LARGE-DATAメソッドの機能および引数を以下に示します。

| メソッド名               | 引数                       | 引数の型                      | 意味                                                                                   |
|---------------------|--------------------------|---------------------------|--------------------------------------------------------------------------------------|
| REGISTER-LARGE-DATA | USING 第1<br>バイナリデータリザーブ部 | PIC S9(9) BINARY   COMP-5 | USING の第1引数で指定されたデータ項目をリザーブ部、第2引数で指定されたデータ項目を長さ部、第3引数で指定されたデータ項目をデータ部とするデータ項目を登録します。 |
|                     | USING 第2<br>バイナリデータ長さ部   | PIC 9(9) BINARY   COMP-5  |                                                                                      |
|                     | USING 第3<br>バイナリデータデータ部  | PIC X(p)   PIC N(q)       |                                                                                      |
|                     | RETURNING<br>なし          | —                         |                                                                                      |
|                     |                          |                           |                                                                                      |

#### 使用規則

1. 基本的な規則は、REGISTERメソッドの規則に従わなければなりません。

2. 第1引数、第2引数および第3引数で指定するデータ項目は、以下の形式の同じ集団項目に従属する基本項目1、基本項目2および基本項目3でなければなりません。

```
nn 集団項目 1.
mm 基本項目 1 PIC S9(9) BINARY | COMP-5.
mm 基本項目 2 PIC 9(9) BINARY | COMP-5.
mm 基本項目 3 PIC X(p) | N(q).
```

- 基本項目1は、符号付き2進4バイトのデータ項目でなければなりません。
- 基本項目2は、符号なし2進4バイトのデータ項目でなければなりません。
- 基本項目3は、英数字項目または日本語項目でなければなりません。
- 基本項目1、基本項目2および基本項目3には、上記に記述されていない句を指定してはなりません。

3. 登録するデータ項目は1つであるとみなします。

#### 使用例

```
01 OBJ-SPCL USAGE OBJECT REFERENCE BINTBL.
01 GET-DATA1.
02 GET-DATA1-RSV PIC S9(9) BINARY.
02 GET-DATA1-LNG PIC 9(9) BINARY.
02 GET-DATA1-DAT PIC X(32767).
01 ELM-NO PIC S9(4) COMP-5 VALUE 1.
```

として

```
INVOKE BINTBL "NEW" USING ELM-NO
 RETURNING OBJ-SPCL.
INVOKE OBJ-SPCL "REGISTER-LARGE-DATA" USING GET-DATA1-RSV
 GET-DATA1-LNG
 GET-DATA1-DAT.
```

### REGISTER-LARGE-DATA-INDICATORメソッド

REGISTER-LARGE-DATA-INDICATORメソッドは、バイナリデータに対応するCOBOLのデータ項目の登録を行います。また、登録するデータ項目に対応する標識変数の指定を行います。

REGISTER-LARGE-DATA-INDICATORメソッドの機能および引数を以下に示します。

| メソッド名                         | 引数                       | 引数の型                      | 意味                                                                                                              |
|-------------------------------|--------------------------|---------------------------|-----------------------------------------------------------------------------------------------------------------|
| REGISTER-LARGE-DATA-INDICATOR | USING 第1<br>バイナリデータリザーブ部 | PIC S9(9) BINARY   COMP-5 | USING の第1引数で指定されたデータ項目をリザーブ部、第2引数で指定されたデータ項目を長さ部、第3引数で指定されたデータ項目をデータ部とするデータ項目を登録します。登録するデータ項目に対応する標識変数の指定を行います。 |
|                               | USING 第2<br>バイナリデータ長さ部   | PIC 9(9) BINARY   COMP-5  |                                                                                                                 |
|                               | USING 第3<br>バイナリデータデータ部  | PIC X(p)   PIC N(q)       |                                                                                                                 |
|                               | USING 第4<br>標識変数         | PIC S9(4) BINARY   COMP-5 |                                                                                                                 |
|                               | RETURNING<br>なし          | —                         |                                                                                                                 |

## 使用規則

1. 第1引数、第2引数および第3引数はREGISTER-LARGE-DATAメソッドの使用規則に従わなければなりません。
2. 第4引数は、符号付き2進2バイトのデータ項目でなければなりません。
3. 標識変数の使用方法および設定する値、設定される値については、各クラスライブラリの使用規則に従います。

## 使用例

```
01 OBJ-SPCL USAGE OBJECT REFERENCE BINTBL.
01 GET-DATA1.
 02 GET-DATA1-RSV PIC S9(9) BINARY.
 02 GET-DATA1-LNG PIC 9(9) BINARY.
 02 GET-DATA1-DAT PIC X(32767).
01 IND1 PIC S9(4) COMP-5.
01 ELM-NO PIC S9(4) COMP-5 VALUE 1.
```

として

```
INVOKE BINTBL "NEW" USING ELM-NO
RETURNING OBJ-SPCL.
INVOKE OBJ-SPCL "REGISTER-LARGE-DATA" USING GET-DATA1-RSV
GET-DATA1-LNG
GET-DATA1-DAT
IND1.
```

## 18.7 ANY LENGTH句を使用したプログラミング

ここでは、データ項目にANY LENGTH句を用いたアプリケーションの作成について説明します。

### 18.7.1 文字列を扱うクラス

オブジェクト指向機能を用いて、文字列を扱うクラスを作成する場合、いろいろな長さの文字列を扱いたいことがあります。そのような場合、扱う文字列の最大長を決定することが必要になります。それはCOBOLの文字列として、最大長を決めずに宣言する方法がないことによります。また、オブジェクト指向機能のインタフェース誤りを防ぐための適合規則により、呼出し側は実際は違う長さの文字列を渡したくても、呼び出すメソッドに合わせた最大長で宣言した変数に格納して、呼び出さなければなりません。

そうすると、もし文字列の長さを最初に決めた最大長から変更したくなった場合、呼び出すメソッドが定義されているクラスだけでなく、そのクラスを参照しているプログラムやクラスの最大長をすべて同じに修正し、再翻訳しなければなりません。変更能耐えられるよう十分余裕をみて最大長を決めたとなると、通常に使用する長さが短いときの性能が悪くなります。これを解決しようとすれば、呼び出すメソッドに文字列の実際の長さを渡し、その長さで部分参照付けするような複雑な処理が必要になってしまいます。

たとえば、下の例のように名前とパスワードを認証するクラス(メソッド)を作成したい場合、文字列の最大長を決めておく必要があります。以下の例は、名前を日本語で10文字、パスワードを英数字で8文字として作成しています。

#### 文字列を渡す側

```
PROGRAM-ID. 情報変更.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
 REPOSITORY.
 CLASS 認証クラス.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 名前文字列.
 02 名前 PIC N(10).
01 パスワード PIC X(8).
01 認証結果 PIC X(2).
01 認証オブジェクト USAGE OBJECT REFERENCE 認証クラス.
PROCEDURE DIVISION.
 INVOKE 認証クラス "NEW" RETURNING 認証オブジェクト.
 DISPLAY NC"名前とパスワードを入力してください".
 ACCEPT 名前文字列.
```

```

ACCEPT パスワード.
INVOKE 認証オブジェクト “認証メソッド” USING 名前 パスワード
RETURNING 認証結果.

IF 認証結果 = “OK” THEN
CALL “情報変更処理”
ELSE
DISPLAY NC“変更する資格がありません”
END-IF.
END PROGRAM 情報変更.

```

渡された文字列を扱う汎用クラス

```

CLASS-ID. 認証クラス INHERITS FJBASE.
:
OBJECT.
PROCEDURE DIVISION.
METHOD-ID. 認証メソッド.
DATA DIVISION.
LINKAGE SECTION.
01 名前 PIC N(10).
01 パスワード PIC X(8).
01 認証結果 PIC X(2).
PROCEDURE DIVISION USING 名前 パスワード
RETURNING 認証結果.
EVALUATE 名前 ALSO パスワード
WHEN NC“富士一夫” ALSO “A1”
WHEN NC“富士二郎” ALSO “XXXXYYY2”
MOVE “OK” TO 認証結果
WHEN OTHER
MOVE “NG” TO 認証結果
END-EVALUATE.
END METHOD 認証メソッド.
END OBJECT.
END CLASS 認証クラス.

```

この状態で、認証クラスを利用する処理が他にも発生し、パスワードの最大長を変更したくなった場合、まず、認証クラスを修正します。すると、それに引きずられる形で情報変更のプログラムの修正が必要になります。また、インタフェースが変更されるため、認証クラスを継承するクラスの再翻訳が必要となります。

## 18.7.2 ANY LENGTH句の使用

COBOLでは前述の問題を解決するために、ANY LENGTH句をサポートしています。ANY LENGTH句は、メソッドの連絡節の英数字項目または日本語項目に指定することができ、その長さは呼び出されたときに自動的に呼出し側で指定された項目の長さとして評価されます。したがって下記のように記述しておけば、どんな長さの項目が指定されても扱えるようなクラスの作成が可能になります。ANY LENGTH句が指定された項目の文字数を求めるときはLENGTH関数、長さ(バイト数)を求めるときはLENG関数が使用できます。また、復帰項目にANY LENGTH句を指定することにより、呼出し側の復帰項目の長さで文字列を返却することも可能です。

ANY LENGTH句を用いた例

```

CLASS-ID. 認証クラス INHERITS FJBASE.
:
OBJECT.
PROCEDURE DIVISION.
METHOD-ID. 認証メソッド.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 名前長 PIC 9(4) COMP-5.
01 パスワード長 PIC 9(4) COMP-5.
LINKAGE SECTION.
01 名前 PIC N ANY LENGTH. ← … 認証メソッドに制御が渡されたとき、呼出し側の対応する項目の
01 パスワード PIC X ANY LENGTH. | 長さに決まる。
01 認証結果 PIC X ANY LENGTH. ←
PROCEDURE DIVISION USING 名前 パスワード

```

```
 RETURNING 認証結果.
 COMPUTE 名前長 = FUNCTION LENGTH(名前).
 COMPUTE パスワード長 = FUNCTION LENG(パスワード).
 :
 END METHOD 認証メソッド.
 END OBJECT.
 END CLASS 認証クラス.
```

汎用的なクラスを作成する場合、そのクラスが抽象的であればあるほどインタフェースの変更の影響は大きくなります。そのため、最大長を意識しなくても汎用クラスの作成ができることは重要なテクニックといえます。

## 第19章 プログラムのデバッグ

本製品で作成したプログラムをデバッグする手段として、NetCOBOLのデバッグ機能とCOBOL対話型デバッグ機能が用意されています。

本章では、NetCOBOLのデバッグ機能の使い方について説明します。

COBOL対話型デバッグ機能については、“[第20章 対話型デバッグの使い方](#)”を参照してください。

### 19.1 デバッグ機能の概要

NetCOBOLには、次のデバッグ機能があります。

- ・ 誤った領域の参照、データ例外、パラメタのチェック(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)
- ・ ランタイムシステム領域のチェック(メモリチェック機能)
- ・ アプリケーションエラーおよびUレベルの実行時メッセージ発生時の診断レポートおよびダンプの出力(診断機能)
- ・ 異常終了した文を特定する目的プログラムリストの出力(目的プログラムリストを使ったデバッグ)

デバッグ機能を使うには、COBOLプログラムを翻訳するときに各デバッグ機能の翻訳オプションを指定し、そのプログラムを実行するときにデバッグ機能を動作させるための環境を指定します。

各デバッグ機能の概要と指定する翻訳オプションを、下表に示します。

表19.1 デバッグ機能の使い方と指定する翻訳オプション

| 機能名     | 使い方                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 翻訳オプション |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| CHECK機能 | 次の検査を行います。 <ul style="list-style-type: none"><li>・ 表を参照時、添字・指標がその表の範囲外を指していないか</li><li>・ 部分参照時、その参照位置がデータの長さを超えていないか</li><li>・ OCCURS DEPENDING ON 句を含むデータを参照するとき、その目的語の内容に誤りがな</li><li>いか</li><li>・ 数字項目参照時、属性形式と異なる値が入っていないか</li><li>・ 除算のとき、除数がゼロでないか</li><li>・ メソッド呼出し時、呼出し側と呼び出されるメソッドのパラメタの数と属性が一致しているか</li><li>・ プログラムの呼出し時、呼出し側と呼び出されるプログラムの呼出し規約が一致しているか</li><li>・ プログラムの呼出し時、呼出し側と呼び出されるプログラムのパラメタの数と長さが一致してい</li><li>るか</li></ul> 【用途】 <ul style="list-style-type: none"><li>・ メモリの参照誤りによるプログラムの誤動作を防ぎたい場合</li><li>・ 数値異常によるプログラムの誤動作を防ぎたい場合</li><li>・ パラメタの誤りによるプログラムの誤動作を防ぎたい場合</li><li>・ 異なる呼び出し規約によるプログラムの誤動作を防ぎたい場合</li></ul> | CHECK   |
| TRACE機能 | 次の情報を出力します。 <ul style="list-style-type: none"><li>・ 実行した文のトレース結果</li><li>・ 異常終了したときに実行した文の行番号および動詞番号</li><li>・ 実行した文を含むプログラム名とプログラム属性情報</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | TRACE   |

| 機能名                | 使い方                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 翻訳オプション                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
|                    | <ul style="list-style-type: none"> <li>・ 実行中に出力されたメッセージ</li> </ul> <p>【用途】</p> <ul style="list-style-type: none"> <li>・ どの文で異常終了したのかを知りたい場合</li> <li>・ 異常終了までに実行した文の経路を知りたい場合</li> <li>・ 実行の途中で出力されたメッセージを確認したい場合</li> </ul>                                                                                                                                                                                                                                                          |                         |
| COUNT機能            | <p>次の情報を出力します。</p> <ul style="list-style-type: none"> <li>・ プログラム上の各文の実行回数および全文の全実行回数に対する各文の実行比率</li> <li>・ プログラム上の文種別ごとの実行回数および全文の全実行回数に対する文種別ごとの実行比率</li> </ul> <p>【用途】</p> <ul style="list-style-type: none"> <li>・ プログラムの実行した全ルートの走行を確認したい場合</li> <li>・ プログラムの効率化を図りたい場合</li> </ul>                                                                                                                                                                                               | COUNT                   |
| メモリチェック機能          | <p>次の検査を行います。</p> <ul style="list-style-type: none"> <li>・ プログラムおよびメソッドの手続き部の開始、終了でランタイムシステム領域をチェックします。領域が破壊されていれば、以下の情報を出力します。 <ul style="list-style-type: none"> <li>ー 破壊を検出したプログラム名またはメソッド名</li> <li>ー 破壊を検出した場所(手続き部の開始／終了)</li> <li>ー 破壊された領域のアドレス</li> </ul> </li> </ul> <p>【用途】</p> <ul style="list-style-type: none"> <li>・ ランタイムシステム領域を破壊したプログラムを特定したい場合。診断機能と併用することによって、破壊箇所を特定することもできます。</li> </ul>                                                                        | —                       |
| 診断機能               | <p>次の情報を診断レポートに出力します。</p> <ul style="list-style-type: none"> <li>・ エラーの種別(例外コードまたは実行時メッセージ)</li> <li>・ エラー箇所(モジュール名、プログラム名、ソースファイル名、行番号)</li> <li>・ 呼出経路</li> <li>・ システム情報</li> <li>・ 環境変数</li> <li>・ 実行環境情報</li> <li>・ プロセス一覧</li> <li>・ モジュール一覧</li> <li>・ スレッド情報</li> </ul> <p>また、ダンプを出力します。</p> <p>【用途】</p> <ul style="list-style-type: none"> <li>・ どの文でどのようなエラーが発生したのかを知りたい場合</li> <li>・ エラー発生までに実行したプログラムの呼出経路を知りたい場合</li> <li>・ エラー発生時点でのアプリケーションやコンピュータの状態を知りたい場合</li> </ul> | TEST                    |
| 目的プログラムリストを使ったデバッグ | <p>目的プログラムごとに次の情報を出力します。</p> <ul style="list-style-type: none"> <li>・ オブジェクト内相対オフセット</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                       | LIST<br>PRINT<br>SOURCE |



| 機能名 | 使い方                                                                                                                                                                                                                                                     | 翻訳オプション     |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
|     | <ul style="list-style-type: none"> <li>機械語の命令コード</li> <li>手続き名と手続き番号</li> <li>アセンブラ形式の命令</li> <li>動詞名と行番号</li> <li>領域名</li> <li>COBOLプログラム中に記述された定義語</li> </ul> <p>【用途】</p> <ul style="list-style-type: none"> <li>アプリケーションを異常終了させた文を特定したい場合</li> </ul> | COPY<br>MAP |



## 注意

TRACE機能とCOUNT機能は、同時に使用できません。

### 19.1.1 ステートメント番号

以降の説明で“ステートメント番号”と記述した場合には、次の表現を意味します。

行番号[. 行番号内動詞追番]

[参照][3.2.2.4 ソースプログラムリスト](#)

#### 行番号

翻訳オプションNUMBER有効時は、“[COPY修飾値-]利用者行番号”の形式となり、NONUMBER有効時は、コンパイラがファイル内の先頭行を“1”とし、1きざみに昇順に与えた値になります。

#### 行番号内動詞追番

同じ行番号内に複数の文が書かれた場合を考慮し、各文に一意性を持たせるための追番です。最初の文に対して1、その次に2、3…というように昇順に割り当てられます。

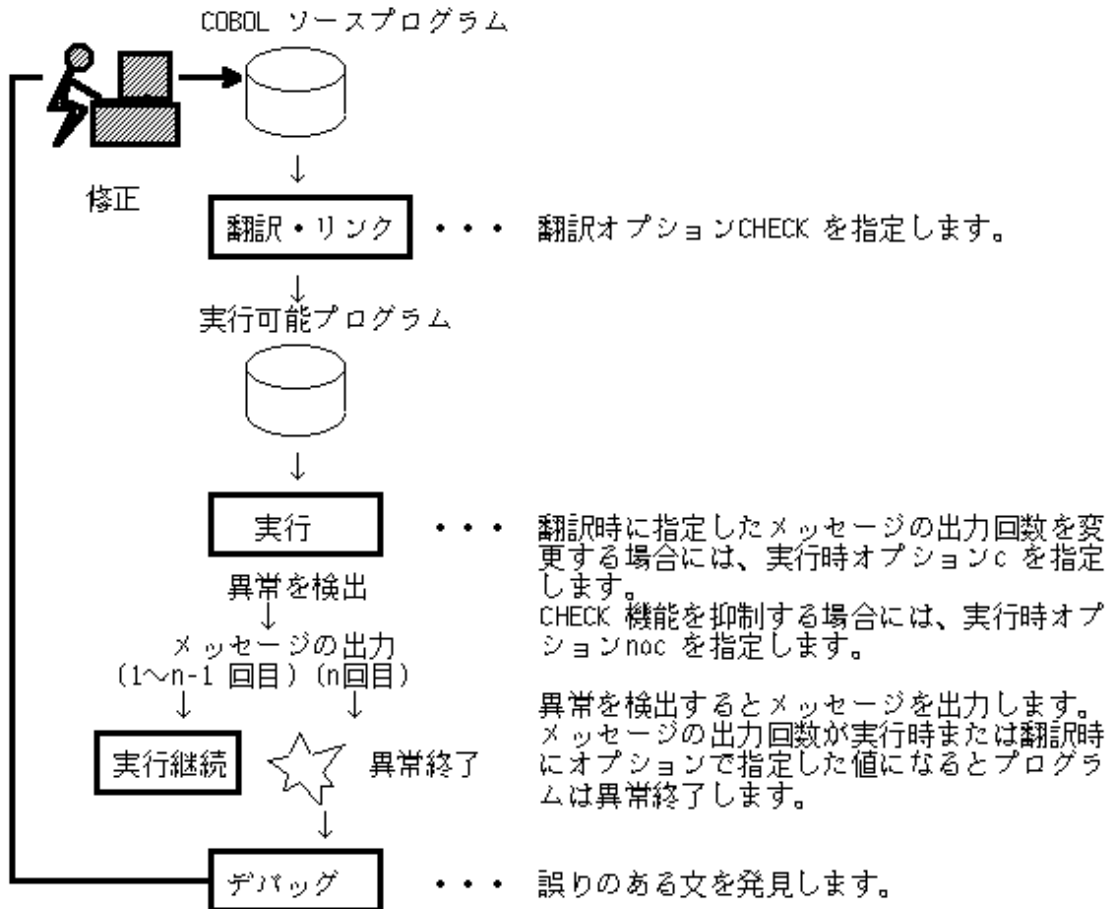
## 19.2 CHECK機能の使い方

CHECK機能では、以下の検査を行い、異常を検出するとメッセージを出力し、異常終了します。そのため、プログラムの誤動作を防ぐことができます。

- 範囲外の添字・指標、部分参照
- 数字のデータ例外および除数のゼロ検査
- メソッド呼出しのパラメタ
- プログラム呼出しの規約
- 内部プログラム呼出しのパラメタ
- 外部プログラム呼出しのパラメタ

### 19.2.1 デバッグ作業の流れ

以下にCHECK機能を使ったデバッグ作業の流れを示します。



## 19.2.2 出力メッセージ

CHECK機能では、検査によって異常を検出すると、メッセージを出力します。このメッセージは、通常メッセージボックスに出力されます。CHECK機能で出力されるメッセージの重大度コードは通常Eレベルですが、メッセージの出力回数が指定された回数になるとUレベルとなります。重大度コードについては、“メッセージ説明書”の“表3-1 実行時メッセージの重大度コードの意味と復帰コードへの影響”を参照してください。

CHECK(PRM)の内部プログラム呼出しパラメタは、翻訳時の検査で、翻訳時診断メッセージとして出力され、出力回数の指定は意味をもちません。

以下にCHECK機能のメッセージについて説明します。

### 19.2.2.1 メッセージの内容

#### メソッド呼出しのパラメタ検査

##### JMP0810I-E/U

[PID:xxxxxxx TID:xxxxxxx] 'メソッド名' メソッドのUSING(またはRETURNING)指定のパラメタに誤りがあります.[ 'NUMBER' (またはPARAMETER=パラメタの順序番号)] PGM=プログラム名 LINE=ステートメント番号

〔対処〕

メッセージで指摘されたメソッドを正しいパラメタで呼び出すように、プログラムを修正し、再度実行してください。

#### 添字および指標検査

---

#### JMP0820I-E/U

[PID:xxxxxxxx TID:xxxxxxxx] 添字または指標の値が範囲外を指しています.PGM=プログラム名.LINE=ステートメント番号.OPD=データ名(データ名の次元数)

〔対処〕

メッセージで指摘された添字または指標に正しい値が設定されるようにプログラムを修正してください。

#### 部分参照検査

---

#### JMP0821I-E/U

[PID:xxxxxxxx TID:xxxxxxxx] 参照可能範囲外の部分参照を行っています.PGM=プログラム名.LINE=ステートメント番号.OPD=データ名.

〔対処〕

メッセージで指摘されたデータ名の部分参照を、その範囲を超えないようにプログラムを修正してください。

#### OCCURS DEPENDING ON句の目的語検査

---

#### JMP0822I-E/U

[PID:xxxxxxxx TID:xxxxxxxx] ODO句の目的語の値が許容範囲を超えています.PGM=プログラム名.LINE=ステートメント番号.OPD=データ名.ODO=ODO句の目的語の名前.

〔対処〕

メッセージで指摘されたOCCURS DEPENDING ON句の目的語に正しい値が設定されるようにプログラムを修正してください。

#### 数字のデータ例外検査

---

#### JMP0828I-E/U

[PID:xxxxxxxx TID:xxxxxxxx] 属性と異なる形式のデータが格納されています.PGM=プログラム名.LINE=ステートメント番号.OPD=データ名.

〔対処〕

メッセージで指摘されたオペランドに正しい値を設定し、再度実行してください。

#### 除数のゼロ検査

---

#### JMP0829I-E/U

[PID:xxxxxxxx TID:xxxxxxxx] 除数にゼロが指定されています.PGM=プログラム名.LINE=ステートメント番号.OPD=データ名

〔対処〕

メッセージで指摘されたオペランドに除数がゼロにならないように値を設定し、再度実行してください。

#### プログラム呼出し規約検査

---

#### JMP0811I-E/U

[PID:xxxxxxxx TID:xxxxxxxx] '呼び出されるプログラム名'プログラムの呼出し規約またはパラメタが一致していません.PGM=プログラム名.LINE=ステートメント番号.

〔対処〕

呼出し規約およびパラメタが同じになるようにプログラムを修正し、再度実行してください。

#### 内部プログラム呼出しのパラメタの検査

---

#### JMN3333I-S

CALL文のUSING指定に記述したパラメタの個数は、PROCEDURE DIVISIONのUSING指定に記述したパラメタの個数と一致していなければなりません。

〔対処〕

パラメタの個数が同じになるようにプログラムを修正してください。

---

**JMN3334I-S**

**CALL文のUSING指定またはRETURNING指定に記述したパラメタ@2@の型は、プログラム@1@のPROCEDURE DIVISIONのUSING指定またはRETURNING指定に記述したパラメタ@3@の型と一致していなければなりません。**

- @1@:プログラム名
- @2@:一意名
- @3@:一意名

〔対処〕

パラメタに指定したオブジェクト参照のUSAGE OBJECT REFERENCE句に指定されたクラス名、FACTORY指定およびONLY指定が同じになるようにプログラムを修正してください。

---

**JMN3335I-S**

**CALL文のUSING指定またはRETURNING指定に記述したパラメタ@2@の長さは、プログラム@1@のPROCEDURE DIVISIONのUSING指定またはRETURNING指定に記述したパラメタ@3@の長さとは一致していなければなりません。**

- @1@:プログラム名
- @2@:一意名
- @3@:一意名

〔対処〕

パラメタの長さが同じになるようにプログラムを修正してください。

---

**JMN3414I-S**

**@1@を呼ぶCALL文にはRETURNING指定を記述しなければなりません。プログラム@1@のPROCEDURE DIVISIONにRETURNING指定があります。**

- @1@:プログラム名

〔対処〕

RETURNING指定の有無を一致するようにプログラムを修正してください。

---

**JMN3508I-S**

**@1@を呼ぶCALL文にはRETURNING指定を記述することはできません。プログラム@1@のPROCEDURE DIVISIONにRETURNING指定がありません。**

- @1@:プログラム名

〔対処〕

RETURNING指定の有無を一致するようにプログラムを修正してください。

## 外部プログラム呼出しのパラメタの検査

---

**JMP0812I-E/U**

**[PID:xxxxxxx TID:xxxxxxx] CALL文のパラメタが一致していません。' \$1 ' PGM=プログラム名. LINE=ステートメント番号.**

- \$1:検出した誤りを示す文字列

〔対処〕

\$1で指摘された内容をもとに、下表に示す処置を施してください。

表19.2 JMP0812I-E/Uの\$1の内容

| \$1                                                  | 処置                                 |
|------------------------------------------------------|------------------------------------|
| USING PARAMETER NUMBER                               | USING に指定したパラメタの個数を一致させてください。      |
| USING nTH PARAMETER<br>(nTH = 1ST, 2ND, 3RD, 4TH...) | USING に指定したn番目のパラメタの大きさを一致させてください。 |
| RETURNING PARAMETER                                  | RETURNING に指定したパラメタの大きさを一致させてください。 |

### 19.2.2.2 メッセージの出力回数

メッセージの出力回数は、翻訳時に翻訳オプションCHECKに指定します。実行単位中にCHECKオプションを指定したCOBOLプログラムが複数存在する場合、最初に動作したプログラムの翻訳オプションに指定した出力回数が有効になります。[参照]“[A.2.5 CHECK \(CHECK機能の使用の可否\)](#)”

メッセージの出力回数は、実行時オプションcを指定して変更することができます。また、実行時オプションを指定して、CHECK機能を抑制することもできます。実行時オプションおよび抑制対象となるCHECK機能は、以下のとおりです。

- noc:全てのCHECK機能
- nocb:CHECK(BOUND)
- noci:CHECK(ICONF)
- nocl:CHECK(LINKAGE)
- nocn:CHECK(NUMERIC)
- nosp:CHECK(PRM)

プログラムの実行は、異常検出後も、メッセージの出力回数が指定した回数になるまで継続されます。

[参照]“[5.8 実行時オプションの指定形式](#)”

## 19.2.3 CHECK機能の使用例

ここでは、CHECK機能の使用例を示します。

### 添字および指標検査

プログラムA

|        |          |              |                                  |
|--------|----------|--------------|----------------------------------|
| 000000 | @OPTIONS | CHECK(BOUND) |                                  |
|        | :        |              |                                  |
| 000500 | 77       | 添字           | PIC S9(4).                       |
| 000600 | 01       | 表            |                                  |
| 000700 | 02       | 表 1          | OCCURS 10 TIMES INDEXED BY 指標 1. |
| 000800 | 03       | 要素 1         | PIC 9(5).                        |
|        | :        |              |                                  |
| 001100 | MOVE     | 15           | TO 添字.                           |
| 001200 | ADD      | 1            | TO 要素 1 (添字).                    |
| 001300 | SET      | 指標 1         | TO 0.                            |
| 001400 | SUBTRACT | 1            | FROM 要素 1 (指標 1).                |
|        | :        |              |                                  |

ADD/SUBTRACT文を実行するときに以下のメッセージが出力されます。

JMP0820I-E/U [PID:xxxxxxx TID:xxxxxxx] 添字または指標の値が範囲外を指しています。PGM=A. LINE=1200. 1. OPD=要素 1  
JMP0820I-E/U [PID:xxxxxxx TID:xxxxxxx] 添字または指標の値が範囲外を指しています。PGM=A. LINE=1400. 1. OPD=要素 1

### 部分参照検査

プログラムA

```

000000 @OPTIONS CHECK (BOUND)
 :
000500 77 データ 1 PIC X(12).
000600 77 データ 2 PIC X(12).
000700 77 参照する長さ PIC 9(4) BINARY.
 :
001100 MOVE 10 TO 参照する長さ.
001200 MOVE データ 1 (1:参照する長さ) TO データ 2 (4:参照する長さ).
 :

```

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

JMP0821I-E/U [PID:xxxxxxx TID:xxxxxxx] 参照可能範囲外の部分参照を行っています。PGM=A。LINE=1200.1。OPD= データ 2。

## OCCURS DEPENDING ON句の目的語検査

プログラムA

```

000000 @OPTIONS CHECK (BOUND)
 :
000050 77 添字 PIC S9(4).
000060 77 個数 PIC S9(4).
000070 01 表.
000080 02 表 1 OCCURS 1 TO 10 TIMES DEPENDING ON 個数.
000090 03 要素 PIC X(5).
 :
000110 MOVE 5 TO 添字.
000120 MOVE 25 TO 個数.
000130 MOVE "ABCDE" TO 要素 (添字).
 :

```

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

JMP0822I-E/U [PID:xxxxxxx TID:xxxxxxx] ODO句の目的語の値が許容範囲を超えています。PGM=A。LINE=120.1。OPD=要素。ODO=個数。

## 数字のデータ例外検査

プログラムA

```

000000 @OPTIONS CHECK (NUMERIC)
 :
000050 01 文字 PIC X(4) VALUE "ABCD".
000060 01 外部10進 REDEFINES 文字 PIC S9(4).
000070 01 数字 PIC S9(4).
 :
000150 MOVE 外部10進 TO 数字.
 :

```

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

JMP0828I-E/U [PID:xxxxxxx TID:xxxxxxx] 属性と異なる形式のデータが格納されています。PGM=A。LINE=150。OPD=外部10進。

## 除数のゼロ検査

プログラムA

```

000000 @OPTIONS CHECK (NUMERIC)
 :
000060 01 被除数 PIC S9(8) BINARY VALUE 1234.
000070 01 除数 PIC S9(4) BINARY VALUE 0.
000080 01 結果 PIC S9(4) BINARY VALUE 0.
 :

```

```
000150 COMPUTE 結果 = 被除数 / 除数.
 :
```

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

JMP0829I-E/U [PID:xxxxxxx TID:xxxxxxx] 除数にゼロが指定されています。 PGM=A. LINE=150. OPD=除数

## メソッド呼出しのパラメタの検査

プログラムA

```
000000 @OPTIONS CHECK(ICONF)
000010 PROGRAM-ID. A.
 :
000030 01 PRM-01 PIC X(9).
000040 01 OBJ-U USAGE IS OBJECT REFERENCE.
 :
000060 SET OBJ-U TO B.
000070 INVOKE OBJ-U "C" USING BY REFERENCE PRM-01.
 :
```

クラスB／メソッドC

```
000010 CLASS-ID. B.
 :
000030 FACTORY.
000040 PROCEDURE DIVISION.
 :
000060 METHOD-ID. C.
 :
000080 LINKAGE SECTION.
000090 01 PRM-01 PIC 9(9) PACKED-DECIMAL.
000100 PROCEDURE DIVISION USING PRM-01.
 :
```

プログラムAのINVOKE文を実行するときに以下のメッセージが出力されます。

JMP0810I-E/U [PID:xxxxxxx TID:xxxxxxx] 'C' メソッドのUSING 指定のパラメタに誤りがあります。 PARAMETER=1. PGM=A LINE=70.1

## プログラム呼出し規約検査

Windows上で動作するアプリケーションの呼出し規約は複数あり、標準の呼出し規約、呼出し規約の指定方法は開発言語によって異なります。

NetCOBOLでは、CALL文や手続き部の始め見出しのWITH指定によって、パラメタの受渡しにどの呼出し規約を使用するかが決まります。[参照][“10.1.2 各種呼出し規約の違い”](#)

その呼出し規約が呼び出す側と呼び出される側で異なると、静的リンク構造や動的リンク構造であればリンク時にエラーとなります。しかし、動的プログラム構造の場合、リンク時にエラーとならないため、実行時スタックが破壊されプログラムが異常終了してしまう可能性があります。また、呼び出す側の呼出し規約がSTDCALLの場合は、たとえ呼出し規約が同じであってもパラメタの個数が異なると呼出し規約が異なる場合と同じ理由で異常終了することがあるため、当チェックの対象となります。

プログラムA

```
000000 @OPTIONS CHECK(LINKAGE)
000010 PROGRAM-ID. A.
 :
000030 01 PRM-01 PIC S9(9) COMP-5.
 :
000070 CALL "B" WITH C LINKAGE USING PRM-01.
 :
```

プログラムB（Cの関数）

```
#include <windows.h>
:
int WINAPI B(long FAR* data1). ... (注)
{
:
}
```

(注) 関数の宣言が以下の場合、STDCALL呼出し規約となります。

|                |        |
|----------------|--------|
| int WINAPI     | 関数名 () |
| int CALLBACK   | 関数名 () |
| int PASCAL     | 関数名 () |
| int far pascal | 関数名 () |
| int _stdcall   | 関数名 () |

プログラムAのCALL文を実行するときに以下のメッセージが出力されます。

JMP0811I-E/U [PID:xxxxxxx TID:xxxxxxx] 'B' プログラムの呼出し規約またはパラメタが一致していません。 PGM=A LINE=7.1

## 内部プログラム呼出しのパラメタの検査

プログラムA

```
000001 @OPTIONS CHECK (PRM)
000002 PROGRAM-ID. A.
000003 ENVIRONMENT DIVISION.
000004 CONFIGURATION SECTION.
000005 REPOSITORY.
000006 CLASS CLASS1.
000007 DATA DIVISION.
000008 WORKING-STORAGE SECTION.
000009 01 P1 PIC X(20).
000010 01 P2 PIC X(10).
000011 01 P3 USAGE OBJECT REFERENCE CLASS1.
000012 PROCEDURE DIVISION.
000013 CALL "SUB1" USING P1 P2 *> JMN3333I-S
000014 CALL "SUB2" *> JMN3414I-S
000015 CALL "SUB1" USING P1 RETURNING P2 *> JMN3508I-S
000016 CALL "SUB1" USING P2 *> JMN3335I-S
000017 CALL "SUB3" USING P3 *> JMN3334I-S
000018 EXIT PROGRAM.
000019*
000020 PROGRAM-ID. SUB1.
000021 DATA DIVISION.
000022 LINKAGE SECTION.
000023 01 L1 PIC X(20).
000024 PROCEDURE DIVISION USING L1.
000025 END PROGRAM SUB1.
000026*
000027 PROGRAM-ID. SUB2.
000028 DATA DIVISION.
000029 LINKAGE SECTION.
000030 01 RET PIC X(10).
000031 PROCEDURE DIVISION RETURNING RET.
000032 END PROGRAM SUB2.
000033*
000034 PROGRAM-ID. SUB3.
000035 DATA DIVISION.
000036 LINKAGE SECTION.
000037 01 L-OR1 USAGE OBJECT REFERENCE.
000038 PROCEDURE DIVISION USING L-OR1.
000039 END PROGRAM SUB3.
000040 END PROGRAM A.
```



プログラムAを翻訳すると、以下の翻訳時診断メッセージが出力されます。

\*\*\* 診断メッセージ \*\*\* (A)

13: JMN3333I-S CALL文のUSING指定に記述したパラメタの個数は、PROCEDURE DIVISIONのUSING指定に記述したパラメタの個数と一致していません。

14: JMN3414I-S 'SUB2'を呼ぶCALL文にはRETURNING指定を記述しなければなりません。プログラム'SUB2'のPROCEDURE DIVISIONにRETURNING指定があります。

15: JMN3508I-S 'SUB1'を呼ぶCALL文にはRETURNING指定を記述することはできません。プログラム'SUB1'のPROCEDURE DIVISIONにRETURNING指定がありません。

16: JMN3335I-S CALL文のUSING指定またはRETURNING指定に記述したパラメタ'P2'の長さは、プログラム'SUB1'のPROCEDURE DIVISIONのUSING指定またはRETURNING指定に記述したパラメタ'L1'の長さと一致していません。

17: JMN3334I-S CALL文のUSING指定またはRETURNING指定に記述したパラメタ'P3'の型は、プログラム'SUB3'のPROCEDURE DIVISIONのUSING指定またはRETURNING指定に記述したパラメタ'L-OR1'の型と一致していません。

最大重大度コードは S で、翻訳したプログラム数は 1 本です。

## 外部プログラム呼出しのパラメタの検査

プログラム呼出しにおいてパラメタ受渡しに誤りがあると、思わぬ所を参照したり、更新したりするため、プログラムを誤動作させてしまいます。

翻訳オプションCHECK(PRM)を指定して翻訳したCOBOLプログラムから、翻訳オプションCHECK(PRM)を指定して翻訳したCOBOLプログラムを呼び出した際、パラメタ個数および、それぞれのパラメタの長さが一致していない場合には、メッセージが出力されます。

```
000010 @OPTIONS CHECK (PRM)
000020 IDENTIFICATION DIVISION.
000030 PROGRAM-ID. A.
000040 DATA DIVISION.
000050 WORKING-STORAGE SECTION.
000060 01 USE-PRM01 PIC 9(04).
000070 01 USE-PRM02 PIC 9(04).
000080 01 RET-PRM01 PIC 9(04).
000090 PROCEDURE DIVISION.
000100 CALL 'B' USING USE-PRM01 USE-PRM02
000110 RETURNING RET-PRM01.
000120 END PROGRAM A.
```

```
000000 @OPTIONS CHECK (PRM)
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. B.
000030 DATA DIVISION.
000070 LINKAGE SECTION.
000080 01 USE-PRM01 PIC 9(08).
000090 01 USE-PRM02 PIC 9(04).
000100 01 RET-PRM01 PIC 9(04).
000120 PROCEDURE DIVISION USING USE-PRM01 USE-PRM02
000130 RETURNING RET-PRM01.
000140 END PROGRAM B.
```

プログラムAのCALL文を実行するときに以下のメッセージが出力されます。

JMP0812I-E/U [PID:xxxxxxx TID:xxxxxxx] CALL文のパラメタが一致していません。'USING 1ST PARAMETER' PGM=A. LINE=10.1.

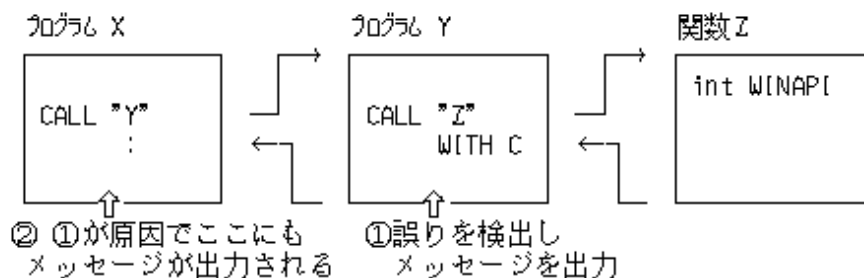
## 19.2.4 注意事項

ここでは、CHECK機能使用時の注意事項について説明します。

- CHECK機能による検査は必ず実施し、検出された情報を元に誤りを修正してください。検出された誤りが修正されない場合、メモリ破壊などの表面化しにくい重大なトラブルが発生することにつながります。検出された誤りが修正されていないアプリケーションの実行結果は保証できません。
- メッセージ出力回数を指定することにより、異常検出後も実行を継続することができますが、異常検出後の動作は保証されません。

- CHECK機能では、データ内容の検査など、COBOLプログラムで記述した以外の処理を行います。そのため、CHECK機能使用時には、プログラムのサイズが大きくなり、実行速度も遅くなります。CHECK機能は、デバッグ時にだけ使用し、デバッグ終了後は、翻訳オプションCHECKを指定しないで再翻訳してください。
- ON SIZE ERROR指定またはNOT ON SIZE ERROR指定の算術文では、ON SIZE ERRORの除数のゼロ検査が行われ、CHECK(NUMERIC)の除数のゼロ検査は行われません。
- 除数のゼロ検査を行った場合、メッセージの出力回数に関係なく、プログラムは異常終了します。
- CHECK(LINKAGE)の検査では、出力回数を指定し、異常検出後も実行が継続された場合、以下の現象が起こります。

呼出しの階層が深く、その下位の階層の呼出しで異常を発見した場合、異常を発見した呼出しだけでなく、上位の呼出しに対してもメッセージが出力されます。上位の呼出しが正しければ、最下層の呼出しを修正するだけでメッセージは出力されなくなります。



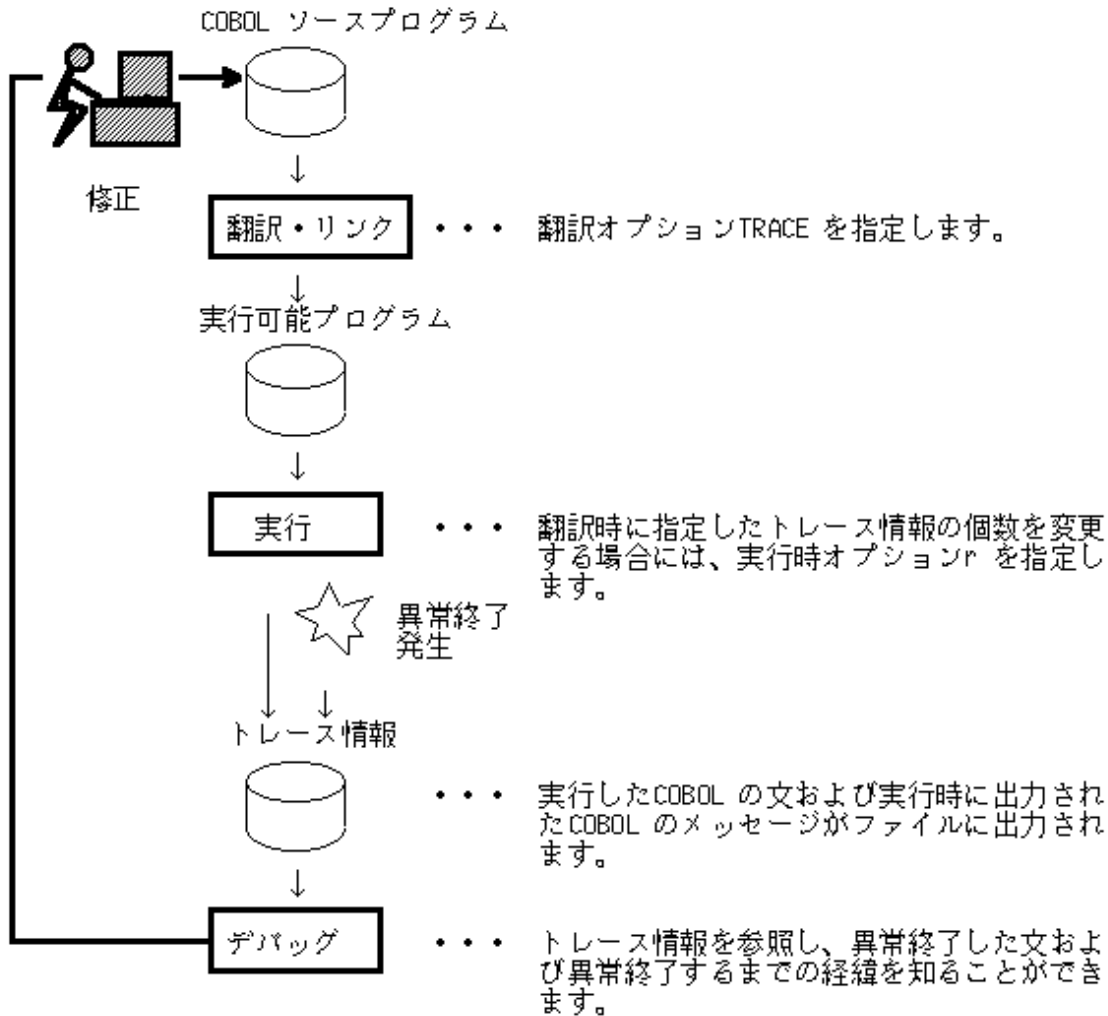
- CHECK(PRM)は、プログラム名として一意名を指定したCALL文によって内部プログラムを呼出す場合は検査しません。
- CHECK(PRM)で外部プログラム呼出しのパラメタを検査する場合、呼出し元および呼出し先のプログラムの両方をNetCOBOL V7.2L10以降のコンパイラで翻訳オプションCHECK(PRM)を指定し、翻訳する必要があります。他言語で作成されたプログラムを呼び出すCALL文、または、他言語で作成されたプログラムから呼び出された場合、パラメタの検査は行われません。
- CHECK(PRM)の外部プログラム呼出しの検査で、呼出し元のパラメタ個数と呼出し先のパラメタ個数の差が4個以上の場合、誤りが検出されないことがあります。
- CHECK(PRM)のパラメタの検査では、可変長項目のパラメタの長さは実行時の長さではなく最大長が使われます。そのため、可変長項目の場合は実際にはパラメタの長さが一致していても、メッセージが出力されることがあります。
- CHECK(PRM)で検査する外部プログラム呼出しの呼出し規約は、COBOL呼出し規約およびC呼出し規約です。STDCALL呼出し規約は、パラメタの検査は行われません。  
[参照]"10.1.2 各種呼出し規約の違い"
- CHECK(PRM)の外部プログラム呼出しの検査で、呼出し元または呼出し先のプログラムにRETURNING指定の記述がない場合、暗黙にPROGRAM-STATUSが受け渡されるため、長さ4バイトのRETURNINGパラメタが指定されたものとして検査します。
- CHECK機能は、CHECKオプションを指定して翻訳したプログラムにだけ有効になります。  
複数のプログラムをリンクしている場合に特定のプログラムだけをCHECK機能の対象にするには、対象にするプログラムはCHECKオプションを指定して翻訳し、対象にしないプログラムはCHECKオプションを指定しないで翻訳してください。

## 19.3 TRACE機能の使い方

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るので、デバッグ作業に役立ちます。ここでは、TRACE機能の使い方について説明します。

### 19.3.1 デバッグ作業の流れ

以下にTRACE機能を使ったデバッグ作業の流れを示します。



### 19.3.2 トレース情報

TRACE機能では、トレース情報として、異常終了するまでに実行したCOBOLの文をステートメント番号で出力します。

#### トレース情報の個数

翻訳時に個数を指定しない翻訳オプションTRACEを指定した場合、トレース情報は200個出力されます。個数を指定した翻訳オプションTRACEを指定した場合、指定した個数のトレース情報が出力されます。実行単位中にTRACEオプションを指定したCOBOLプログラムが複数存在する場合、最初に動作したプログラムの翻訳オプションに指定した個数が有効になります。[参照]“[A.2.58 TRACE \(TRACE機能の使用の可否\)](#)”

なお、トレース情報の個数は、実行時に実行時オプションrを使って変更することができます。また、実行時オプションnorを指定して、TRACE機能を抑制することもできます。[参照]“[5.8 実行時オプションの指定形式](#)”



#### 注意

トレース情報の個数に0は指定できません。

#### トレース情報の格納先

トレース情報は、環境変数情報@CBR\_TRACE\_FILEに指定したファイルに格納されます。[参照]“[5.4.1.48 @CBR\\_TRACE\\_FILE \(トレース情報の出力ファイルの指定\)](#)”

環境変数情報@CBR\_TRACE\_FILEが指定されていない場合は、実行可能ファイルの名前に、拡張子TRCおよびTROを付加したファイルに格納されます。

トレース情報は、常に拡張子TRCのファイルに格納され、格納した情報の数が翻訳時または実行時に指定した個数になると、拡張子TRCのファイルの内容は、拡張子TROのファイルに移されます。

以下に、環境変数情報@CBR\_TRACE\_FILEを指定した場合と指定しない場合との例を示します。



## 例

### 環境変数情報@CBR\_TRACE\_FILEにC:\PROG1.TRCを指定した場合

- ・トレース情報の格納先(最新情報)  
C:\PROG1.TRC
- ・トレース情報の格納先(一世代前の情報)  
C:\PROG1.TRO

### 環境変数情報@CBR\_TRACE\_FILEを指定しない場合

- ・実行可能プログラムの格納先  
D:\PROG2.EXE
- ・トレース情報の格納先(最新情報)  
D:\PROG2.TRC
- ・トレース情報の格納先(一世代前の情報)  
D:\PROG2.TRO

## トレース情報の出力形式

|                            |                                                      |                               |                   |
|----------------------------|------------------------------------------------------|-------------------------------|-------------------|
| NetCOBOL DEBUG INFORMATION |                                                      | DATE 2007-05-01 TIME 11:39:22 |                   |
|                            |                                                      | PID=00000123 [1]              |                   |
| TRACE INFORMATION          |                                                      |                               |                   |
| [2]                        | [3]                                                  | [4]                           | [5] [6]           |
| 1                          | 外部プログラム名 (内部プログラム名)                                  |                               | 翻訳日付 TID=00000099 |
| 2                          | [7]1100.1 TID=00000099                               |                               |                   |
| 3                          | 1200.1 TID=00000099                                  |                               |                   |
| 4                          | 1300.1 TID=00000099                                  |                               |                   |
| 5                          | [8] 1300.2                                           | [9]                           | [5]               |
| 6                          | クラス名 [メソッド名] 翻訳日付 TID=00000099                       |                               |                   |
| 7                          | 2100.1 TID=00000099                                  |                               |                   |
| 8                          | 2200.1 TID=00000099                                  |                               |                   |
| 9                          | JMPnnnnI-x xxxxxxxx xx xxxxxxxx. [10]                |                               |                   |
| 10                         | THE INTERRUPTION WAS OCCURRED. PID=00000123,... [11] |                               |                   |
| 11                         | EXIT-THREAD TID=00000099 [12]                        |                               |                   |
| :                          |                                                      |                               |                   |

#### [1] プロセスID(16進数表記 8桁)

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

#### [2] トレース情報の通番(10進数表記 10桁)

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

#### [3] 外部プログラム名

外部プログラム名が出力されます。

#### [4] 内部プログラム名

内部プログラムが動作したときに出力されます。外部プログラムの場合には表示されません。

#### [5] 翻訳日付

外部プログラムが動作する場合、動作するプログラムの翻訳日時を出力します。

#### [6] スレッドID(16進数表記 8桁)

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

#### [7] 実行した文、手続き名/段落名

実行した文、手続き名または段落名のステートメント番号を出力します。

#### [8] クラス名

クラス名が出力されます。継承したメソッドを実行した場合、メソッドの手続きを定義した継承元のクラス名が出力されます。

#### [9] メソッド名

メソッド名が出力されます。

#### [10] 実行時メッセージ

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

#### [11] 例外通知メッセージ

オペレーティングシステムから例外(不当なアドレスを参照した場合など)が通知された場合、このメッセージを出力します。  
なお、プログラムが正常に終了した場合またはUIレベルエラーが発生した場合は、このメッセージは出力されません。

#### [12] スレッド終了通知メッセージ

プログラムが正常に終了し、スレッドが終了した場合、このメッセージが出力されます。

### トレース情報ファイル

トレース情報ファイルは、実行可能ファイルのプロセス毎に出力されます。同じ名前の実行可能プログラムを複数同時に実行する場合は、プロセス毎にトレース情報の出力ファイル名を変える必要があります。

プロセス毎にトレース情報のファイル名を変える場合は、環境変数情報@CBR\_TRACE\_PROCESS\_MODEを指定します。[参照]  
“5.4.1.49 @CBR\_TRACE\_PROCESS\_MODE (TRACEファイルのプロセス毎の出力指定)”

環境変数@CBR\_TRACE\_PROCESS\_MODEが指定された場合、実行可能ファイル名、プロセスID、実行日付、実行時間に、拡張子TRCおよびTRO付加したファイルが作成されます。

以下に、環境変数情報@CBR\_TRACE\_PROCESS\_MODEを指定した場合の例を示します。



#### 例

環境変数情報@CBR\_TRACE\_PROCESS\_MODEを指定した場合

実行可能ファイル名: SAMPLE. EXE  
プロセスID: 00000EC4  
実行日付: 2010年1月12日  
実行時間: 10時48分50秒

トレース情報ファイル名(最新情報)

SAMPLE-00000EC4\_20100112\_104850. TRC

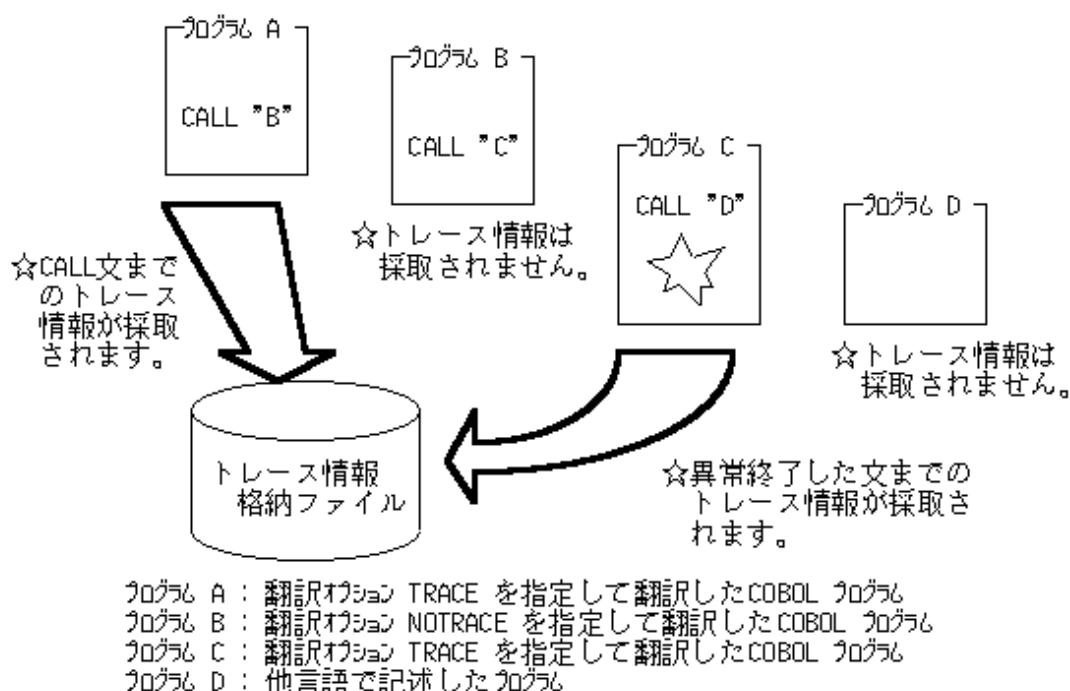
トレース情報ファイル名(一世代前の情報)

SAMPLE-00000EC4\_20100112\_104850. TRO

### 19.3.3 注意事項

ここでは、TRACE機能使用時の注意事項について説明します。

- ・ TRACE機能でトレース情報を採取できるのは、翻訳オプションTRACEを指定して翻訳したCOBOLプログラムだけです。



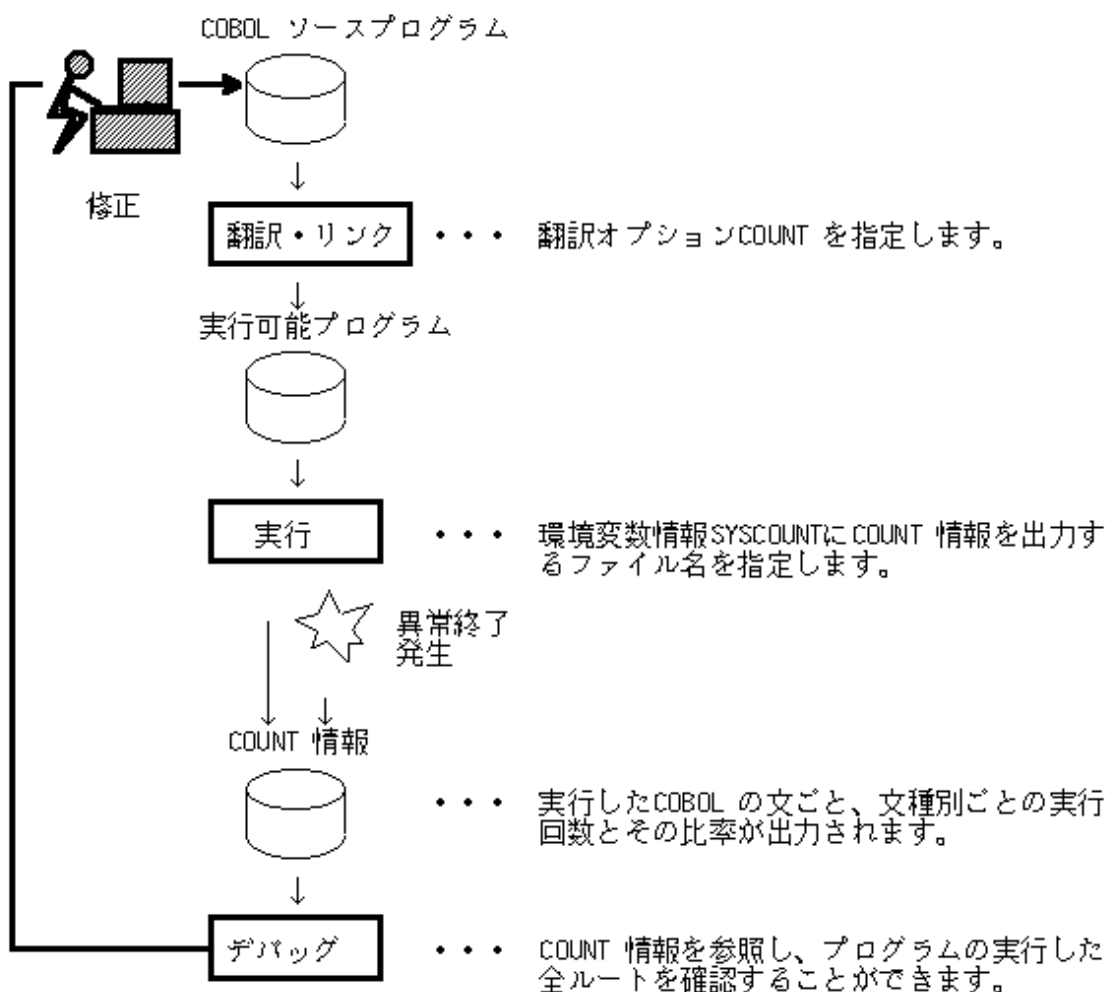
- ・ TRACE機能では、トレース情報の採取など、COBOLプログラムで記述した以外の処理を行います。そのため、TRACE機能使用時には、プログラムのサイズが大きくなり、実行速度も遅くなります。TRACE機能は、デバッグ時にだけ使用し、デバッグ終了後は、翻訳オプションTRACEを指定しないで再翻訳してください。
- ・ TRACE機能が有効な場合、プログラムの正常終了または異常終了にかかわらず、トレース情報は生成されます。
- ・ トレース情報ファイルが存在している状態で、再度プログラムを実行した場合、元のトレース情報ファイルの内容は失われます。
- ・ トレース情報ファイルが不要になった場合には、削除してください。
- ・ 対話型デバッグと併用する場合、例外通知メッセージが出力されないことがあります。
- ・ トレース情報ファイルには、プロトタイプ宣言されたメソッドであることが識別できる情報を出力しません。  
 メソッドのステートメント番号を参照する場合、クラス名とメソッド名を参照して、プロトタイプ宣言により「分離されたメソッド」であるか、そうでないかを確認してください。「分離されたメソッド」の場合、ステートメント番号は、「分離されたメソッド」のソースファイルの行番号で表現します。クラス定義のソースファイルの行番号ではありません。
- ・ トレース情報ファイルは、実行可能ファイルのプロセス毎に出力されます。  
 複数のプロセスから、同時に同じファイルへ出力することはできません。出力した場合は、実行時にエラーになります。  
 同じ名前の実行可能プログラムを複数同時に実行する場合は、プロセス毎にトレース情報の出力ファイル名を変える必要があります。

## 19.4 COUNT機能の使い方

COUNT機能は、ソースプログラム上に書かれた各文の実行回数と全文の実行回数に対する各文の実行回数比率を表示する機能です。また、ソースプログラム中に書かれた文種別ごとの実行回数とその比率などを表示します。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。

### 19.4.1 デバッグ作業の流れ

以下に、COUNT機能を使ったデバッグ作業の流れを示します。



## 19.4.2 COUNT情報

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情報が出力されます。[参照]“[A.2.8 COUNT \(COUNT機能の使用の可否\)](#)”、“[5.4.1.78 SYSCOUNT \(COUNT情報の出力ファイルの指定\)](#)”

### COUNT情報の出力形式

```

[1]
NetCOBOL COUNT INFORMATION(END OF RUN UNIT) DATE 2008-05-01 TIME 20:45:21
 PID=00000123 TID=00000099

[2]
STATEMENT EXECUTION COUNT PROGRAM-NAME : COUNT-PROGRAM

[3] [4] [5] [6]
STATEMENT PROCEDURE-NAME/VERB-ID EXECUTION PERCENTAGE
NUMBER COUNT (%)

 15 PROCEDURE DIVISION COUNT-PROGRAM
 17 DISPLAY 1 14.2857
 19 CALL 1 14.2857
 21 DISPLAY 1 14.2857
 23 STOP RUN 1 14.2857
 31 PROCEDURE DIVISION INTERNAL-PROGRAM
 33 DISPLAY 1 14.2857
 35 INVOKE 1 14.2857

```

37 EXIT PROGRAM 1 14.2857

7

[7]

| VERB EXECUTION COUNT PROGRAM-NAME : COUNT-PROGRAM |             |            |                |                 |                |
|---------------------------------------------------|-------------|------------|----------------|-----------------|----------------|
| [8]                                               | [9]         | [10]       | [11]           | [12]            | [13]           |
| VERB-ID                                           | ACTIVE VERB | TOTAL VERB | PERCENTAGE (%) | EXECUTION COUNT | PERCENTAGE (%) |
| CALL                                              | 1           | 1          | 100.0000       | 1               | 25.0000        |
| DISPLAY                                           | 2           | 2          | 100.0000       | 2               | 50.0000        |
| STOP RUN                                          | 1           | 1          | 100.0000       | 1               | 25.0000        |
|                                                   | 4           | 4          | 100.0000       | 4               |                |

[7]

| VERB EXECUTION COUNT PROGRAM-NAME : COUNT-PROGRAM<br>(INTERNAL-PROGRAM) |             |            |                |                 |                |
|-------------------------------------------------------------------------|-------------|------------|----------------|-----------------|----------------|
| [8]                                                                     | [9]         | [10]       | [11]           | [12]            | [13]           |
| VERB-ID                                                                 | ACTIVE VERB | TOTAL VERB | PERCENTAGE (%) | EXECUTION COUNT | PERCENTAGE (%) |
| DISPLAY                                                                 | 1           | 1          | 100.0000       | 1               | 33.3333        |
| EXIT PROGRAM                                                            | 1           | 1          | 100.0000       | 1               | 33.3333        |
| INVOKE                                                                  | 1           | 1          | 100.0000       | 1               | 33.3333        |
|                                                                         | 3           | 3          | 100.0000       | 3               |                |

[14]

| PROGRAM EXECUTION COUNT PROGRAM-NAME : COUNT-PROGRAM |             |            |                |                 |                |
|------------------------------------------------------|-------------|------------|----------------|-----------------|----------------|
| [15]                                                 | [16]        | [17]       | [18]           | [19]            | [20]           |
| PROGRAM-NAME                                         | ACTIVE VERB | TOTAL VERB | PERCENTAGE (%) | EXECUTION COUNT | PERCENTAGE (%) |
| COUNT-PROGRAM                                        | 4           | 4          | 100.0000       | 4               | 57.1429        |
| INTERNAL-PROGRAM                                     | 3           | 3          | 100.0000       | 3               | 42.8571        |
|                                                      | 7           | 7          | 100.0000       | 7               |                |

[1]

NetCOBOL COUNT INFORMATION(END OF RUN UNIT) DATE 2008-05-01 TIME 20:45:21  
PID=00000123 TID=00000099

[2]

| STATEMENT EXECUTION COUNT CLASS-NAME : COUNT-CLASS |                        |                 |                |  |  |
|----------------------------------------------------|------------------------|-----------------|----------------|--|--|
| [3]                                                | [4]                    | [5]             | [6]            |  |  |
| STATEMENT NUMBER                                   | PROCEDURE-NAME/VERB-ID | EXECUTION COUNT | PERCENTAGE (%) |  |  |
| 15                                                 | PROCEDURE DIVISION     | COUNT-METHOD    |                |  |  |
| 16                                                 | DISPLAY                | 1               | 50.0000        |  |  |
| 37                                                 | EXIT METHOD            | 1               | 50.0000        |  |  |
|                                                    |                        | 2               |                |  |  |

[7]

| VERB EXECUTION COUNT CLASS-NAME : COUNT-CLASS<br>METHOD-NAME : COUNT-METHOD |             |            |                |                 |                |
|-----------------------------------------------------------------------------|-------------|------------|----------------|-----------------|----------------|
| [8]                                                                         | [9]         | [10]       | [11]           | [12]            | [13]           |
| VERB-ID                                                                     | ACTIVE VERB | TOTAL VERB | PERCENTAGE (%) | EXECUTION COUNT | PERCENTAGE (%) |
| DISPLAY                                                                     | 1           | 1          | 100.0000       | 1               | 50.0000        |
| END METHOD                                                                  | 1           | 1          | 100.0000       | 1               | 50.0000        |
|                                                                             | 2           | 2          | 100.0000       | 2               |                |

[14]

METHOD EXECUTION COUNT CLASS-NAME : COUNT-CLASS



| [15]<br>METHOD-NAME                                    | [16]<br>ACTIVE VERB | [17]<br>TOTAL VERB | [18]<br>PERCENTAGE<br>(%) | [19]<br>EXECUTION COUNT | [20]<br>PERCENTAGE<br>(%) |
|--------------------------------------------------------|---------------------|--------------------|---------------------------|-------------------------|---------------------------|
| COUNT-METHOD                                           | 2                   | 2                  | 100.0000                  | 2                       | 100.0000                  |
|                                                        | 2                   | 2                  | 100.0000                  | 2                       |                           |
| [21]<br>PROGRAM/CLASS/PROTOTYPE METHOD EXECUTION COUNT |                     |                    |                           |                         |                           |
| [22]<br>PROGRAM/CLASS<br>/METHOD-NAME                  | [23]<br>ACTIVE VERB | [24]<br>TOTAL VERB | [25]<br>PERCENTAGE<br>(%) | [26]<br>EXECUTION COUNT | [27]<br>PERCENTAGE<br>(%) |
| COUNT-PROGRAM                                          | 7                   | 7                  | 100.0000                  | 7                       | 100.0000                  |
| COUNT-CLASS                                            | 2                   | 2                  | 100.0000                  | 2                       | 100.0000                  |
|                                                        | 9                   | 9                  | 100.0000                  | 9                       |                           |

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

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

[2] 以降の出力がソースプログラムイメージ実行回数リストであることを示します。この情報は、ソースプログラムの翻訳単位で出力します。翻訳単位がプログラムの場合、PROGRAM-NAMEには外部プログラム名を表示します。翻訳単位がクラスの場合、CLASS-NAMEにはクラス名を表示します。翻訳単位がメソッドの場合、CLASS-NAMEにはクラス名を表示し、METHOD-NAMEにはメソッド名を表示します。

[3] ステートメント番号を次の形式で表示します。

[COPY修飾値-]行番号

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

[4] 手続き名および文を表示します。手続き部の始まりには、“PROCEDURE DIVISION”の文字列の後にプログラム名またはメソッド名を表示します。

[5] 実行回数を表示します。最後に実行回数の総数を表示します。

[6] その文の総実行回数に対する比率を表示します。

[7] 以降の出力が文別の実行回数リストであることを示します。この情報は、プログラム単位またはメソッド単位に出力します。したがって、内部プログラムを持つプログラムおよび複数のメソッドを持つクラスでは、複数の文別の実行回数リストを出力します。PROGRAM-NAMEには、プログラム名を次の形式で表示します。

PROGRAM-NAME: プログラム名  
[(呼ばれる内部プログラム名)]

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

[9] ソースプログラム上に書かれている各文のうち、実際に実行した命令数を表示します。

[10] ソースプログラム上に書かれている各文の数を表示します。

[11] ソースプログラム上に書かれている各文の実行比率を表示します。計算式は、[9]÷[10]×100です。

[12] 各文の実行回数を表示します。最後に実行回数の総数を表示します。

[13] 各文の全体に対する実行回数の比率を表示します。各文の実行回数÷実行回数×100で求めます。

- [14] 以降の出力がプログラム別またはメソッド別の実行回数リストであることを示します。このリストは、内部プログラムを持つプログラムの場合およびクラスの場合に出力します。
- [15] プログラム名またはメソッド名をソースプログラム上の出現順に出力します。
- [16] ソースプログラム上に書かれている文のうち、実際に実行した文の数を表示します。
- [17] ソースプログラム上に書かれている文の数を表示します。
- [18] ソースプログラム上に書かれた文の全体に対する比率を表示します。計算式は、[16]÷[17]×100です。
- [19] 各プログラムまたは各メソッドの文実行回数を表示します。最後に合計を表示します。
- [20] 各プログラムまたは各メソッドの全体に対する実行文数の比率を、以下のどちらかで求め、表示します。

各プログラムの文実行回数÷全プログラムの文実行回数合計×100  
 各メソッドの文実行回数÷全メソッドの文実行回数合計×100

- [21] 以降の出力がソースプログラム別(翻訳単位別)の文実行回数リストであることを示します。実行単位中でソースプログラム(翻訳単位)が複数存在する場合、上記情報をプログラムの数だけ繰り返し出力した後、最後に表示します。
- [22] 外部プログラム名、クラス名およびプロトタイプの名を表示します。
- [23] [16]を参照してください。
- [24] [17]を参照してください。
- [25] [18]を参照してください。
- [26] 各翻訳単位の文実行回数を表示します。最後に合計を表示します。
- [27] 各翻訳単位の全体に対する実行文数の比率を、以下で求め、表示します。

各翻訳単位の文実行回数÷全翻訳単位の文実行回数合計×100

### 19.4.3 COUNT機能を使用したプログラムのデバッグ

COUNT機能を利用して行うことのできるプログラムのデバッグ例を以下に示します。

#### プログラムの全ルート走行の確認

COUNT機能の出力リストには実行された文の実行回数が表示されるので、これを調べることで全ルート走行を確認することができます。

#### プログラムの効率化

COUNT機能の出力リストの各文の実行回数の比率およびプログラム単位の文実行回数の比率を調べることで、プログラム中で頻繁に使用される部分を探することができます。こうした部分の文を適正化することにより、プログラム全体の効率化を図ることができます。

### 19.4.4 注意事項

ここでは、COUNT機能使用時の注意事項について説明します。

- COUNT機能では、COUNT情報の採取など、COBOLプログラムで記述した以外の処理を行います。そのため、COUNT機能使用時には、プログラムのサイズが大きくなり、実行速度も遅くなります。COUNT機能は、デバッグ時にだけ使用し、デバッグ終了後は、翻訳オプションCOUNTを指定しないで再翻訳してください。
- 異常終了時の出力ファイルには、異常終了の原因となった文も含まれて出力されています。
- CANCEL文実行時には、取り消されるプログラムのCOUNT情報を出力します。取り消されるプログラムからさらに呼ばれるプログラムがあるとき、そのプログラムのCOUNT情報は、呼ぶプログラムで出力されます。
- 出力ファイル名を定義するために、環境変数情報SYSCOUNTを指定しなければなりません。
- COUNT情報の出力ファイルを印刷する場合、各ページの先頭に見出しを出力するためには、以下の設定を行ってください。

|          |          |     |
|----------|----------|-----|
| 使用するエディタ | : ワードパット |     |
| ページ設定    | : 用紙サイズ  | A 4 |
|          | 印刷の向き    | 横   |

|         |         |                                     |
|---------|---------|-------------------------------------|
|         | 余白      | 左25.4mm 右25.4mm<br>上19.0mm 下31.75mm |
| フォントの指定 | : フォント名 | MSゴシック                              |
|         | フォントサイズ | 9                                   |

- ・他言語プログラムから呼び出されている場合にアプリケーションが異常終了すると、COUNT情報が出力されないことがあります。
- ・COUNT情報は、環境変数SYSCOUNTに指定した出力ファイルに出力されます。複数のプロセスから、同時に同じファイルへ出力することはできません。出力した場合は、実行時にエラーになります。プロセスを複数同時に実行する場合は、プロセスごとに出力ファイル名を変える必要があります。

## 19.5 メモリチェック機能の使い方

メモリチェック機能は、COBOLアプリケーションの実行時、領域破壊が発生した場合に使用します。メモリチェック機能は、COBOLアプリケーションの手続き部の開始、終了でランタイムシステム領域をチェックします。以下の実行時メッセージが出力された場合または事象が発生した場合、領域破壊の可能性があるため、メモリチェック機能を使用して領域破壊の原因を調査してください。

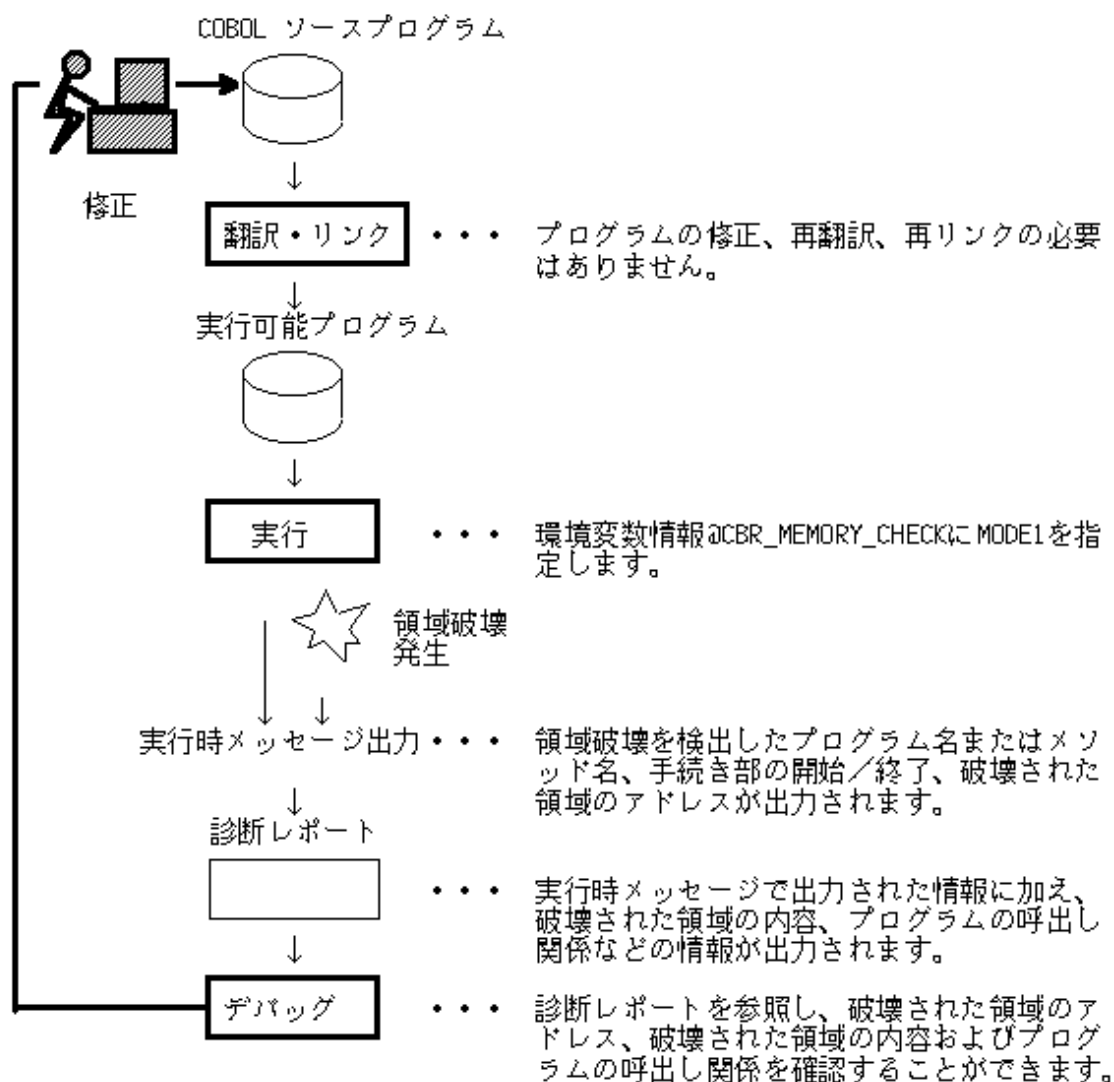
- ・JMP0009I-U ライブラリ作業域が確保できません。(注)
- ・JMP0010I-U ライブラリ作業域が破壊されています。
- ・アプリケーションエラー(アクセス違反)が発生した場合

注:このメッセージは、仮想メモリが不足していない場合にも出力される場合があります。“5.9 注意事項”の“COBOLプログラムの実行時に仮想メモリ不足が発生する場合”を参照してください。

メモリチェック機能を使用する場合、環境変数情報@CBR\_MEMORY\_CHECK=MODE1を指定してください。[参照]“5.4.1.36 @CBR\_MEMORY\_CHECK(メモリチェック機能を使って検査を行う指定)”

### 19.5.1 デバッグ作業の流れ

以下に、メモリチェック機能を使ったデバッグ作業の流れを示します。



## 19.5.2 出力メッセージ

メモリチェック機能では、領域破壊を検出すると以下のメッセージを出力します。メッセージは、通常メッセージボックスに出力されます。

メモリチェック機能のメッセージについて、以下に説明します。

### 19.5.2.1 メッセージの内容

プログラムまたはメソッドの手続き部の開始で領域破壊を検出した場合

**JMP0071I-U**

**[PID:xxxxxxx TID:xxxxxxx] 領域破壊を検出しました. START PGM=プログラム名 BRKADR=破壊された領域の先頭アドレス**

メソッドで領域破壊を検出した場合、“PGM=プログラム名”は“CLASS=クラス名 METHOD=メソッド名”となります。

プログラムまたはメソッドの手続き部の終了で領域破壊を検出した場合

## JMP0071I-U

[PID:xxxxxxx TID:xxxxxxx] 領域破壊を検出しました. END PGM=プログラム名 BRKADR=破壊された領域の先頭アドレス

メソッドで領域破壊を検出した場合、“PGM=プログラム名”は“CLASS=クラス名 METHOD=メソッド名”となります。

## 19.5.3 診断レポート

メモリチェック機能が領域破壊を検出した場合に出力される診断レポートの例を以下に示します。診断レポートとは、NetCOBOL デバッグ機能の診断機能が出力するレポートです。診断機能および診断レポートの出力情報の詳細、見方については、“[19.6 診断機能の使い方](#)”を参照してください。

### NetCOBOL 診断機能

#### <<要約>>

実行時メッセージが発生しました：

アプリケーション：D:\APL\EXDTM.EXE (PID=000000B2)

例外種別：JMP0071I-U [PID:000000B2 TID:00000085] 領域破壊を検出しました。

START PGM=EXDTS BRKADR=0x00DC1198.

発生時間：YYYY/MM/DD (HH:MM:SS)

発生モジュール：D:\APL\EXDTS.DLL

作成時間：YYYY/MM/DD (HH:MM:SS)

ファイルサイズ：51204bytes

#### <<破壊領域>>

破壊位置：00DC1198

内容：アドレス +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +a +b +c +d +e +f

|          |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |                  |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------|
| 00DC1198 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAAAAAAAAAA |
| 00DC11A8 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAAAAAAAAAA |
| 00DC11B8 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAAAAAAAAAA |
| 00DC11C8 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAAAAAAAAAA |
| 00DC11D8 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | AAAAAAAAAAAAAAAA |
| 00DC11E8 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | AAAAAAA. ....    |
| 00DC11F8 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00DC1208 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00DC1218 | 11 | 00 | 11 | 00 | 00 | 01 | 08 | 00 | 54 | 4D | 45 | 4D | 00 | 00 | 00 | ..... TMEM. ...  |
| 00DC1228 | 80 | 00 | 00 | 00 | 98 | 11 | DC | 00 | A8 | 12 | DC | 00 | 46 | 00 | 00 | ..... F. ...     |
| 00DC1238 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00DC1248 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00DC1258 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00DC1268 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00DC1278 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00DC1288 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |

#### <<問題箇所>>

スレッドID：00000085

レジスタ：EAX=0012E27C EBX=1C03F3C8 ECX=00000000 EDX=00000000 ESI=0000005F

EDI=0013AAF0 EIP=77EED7A2 ESP=0012E278 EBP=0012E2C0 EFL=00000246

CS=001B SS=0023 DS=0023 ES=0023 FS=0038 GS=0000

コミット：00004000 (トップ:00130000, ベース:0012C000)

命令：アドレス +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +a +b +c +d +e +f

77EED792 45 C0 00 00 00 00 8D 45 B0 50 FF 15 C0 D3 F0 77

フォールト->77EED7A2 5E 8B E5 5D C2 10 00 A1 18 64 F1 77 8B 4C 24 04

モジュールファイル：D:\APL\EXDTS.DLL

セクション相対位置：.text+000000FF

EXPORT相対位置：EXDTS+000000CB

シンボル相対位置：EXDTS+000000FF

翻訳情報：ShiftJIS, シングルスレッド, NOOPTIMIZE

外部プログラム/クラス：EXDTS

ソースファイル：Exdts.cob

文位置：IN ENTRY-CODE

＜呼出経路＞

[ 1]-----  
以降、省略

## 19.5.4 プログラムの特定

領域破壊を起こしたプログラムの特定方法を以下に示します。

次のような呼出し関係で、領域破壊のメッセージが出力されたとします。



```
JMP0071I-U [PID:00000010 TID:0000000E] 領域破壊を検出しました。
START PGM=C BRKADR=0x00202000
```

メモリチェック機能は、プログラムの手続き部の開始/終了でランタイムシステム領域の検査を行います。この場合、COBOLプログラムAの手続き部の開始検査、COBOLプログラムBの手続き部の開始検査では領域破壊は検出されず、COBOLプログラムCの手続き部の開始検査で領域破壊が検出されたことになります。よって、COBOLプログラムBの手続き部の開始検査以降からCOBOLプログラムCを呼び出すまでに領域破壊が発生したことになります。

## 19.5.5 破壊箇所の特定

破壊箇所を特定するには、診断機能の領域破壊監視機能を使用します。領域破壊を検出した時に出力される実行時メッセージの“破壊された領域の先頭アドレス”を診断機能の起動時に設定することで、破壊箇所を特定するための診断レポートを出力することができます。

### 19.5.5.1 指定方法

診断機能に破壊アドレスを指定する方法を以下に示します。

メモリチェック機能を有効にして、以下のメッセージが出力された場合を例にします。

```
JMP0071I-U [PID:00000010 TID:0000000E] 領域破壊を検出しました。 START PGM=A BRKADR=0x00202020
```

1. COBOLアプリケーションを起動する前に、環境変数情報@CBR\_ATTACH\_TOOLを以下のように指定します。[参照]“[19.6.3.3 領域破壊が発生するアプリケーションに対する起動](#)”

このとき、環境変数情報@CBR\_MEMORY\_CHECKを無効にする必要はありません。

```
@CBR_ATTACH_TOOL=SNAP -w 0x00202020
```

-wの後に指定する破壊領域のアドレスには、メモリチェック機能によって通知されたメッセージの“BRKADR=破壊アドレス”のアドレスを指定します。

2. COBOLアプリケーションを起動します。アプリケーション終了後に、領域破壊を監視した診断レポートが出力されます。出力される診断レポートの形式および見方については、“[19.6.4.3.4 領域破壊監視](#)”を参照してください。

## 19.5.6 注意事項

---

メモリチェック機能では、プログラムの手続き部の開始/終了でランタイムシステム領域の検査を行うため、実行速度が遅くなります。デバッグ終了後には、メモリチェック機能の指定(環境変数情報@CBR\_MEMORY\_CHECK)を無効にしてください。また、診断機能の領域破壊を監視するための指定(環境変数情報@CBR\_ATTACH\_TOOL)も無効にしてください。

## 19.6 診断機能の使い方

---

ここでは、診断機能の概要や使い方および診断機能が出力した診断レポートを使って調査する方法について説明します。

### 19.6.1 診断機能の概要

---

診断機能は、以下の問題に対して診断レポートを出力します。

#### アプリケーションエラーとUレベルの実行時メッセージ

診断機能は、エラーの発生箇所やプログラムの呼出し関係およびアプリケーションの状態など、エラーに関する情報をプログラム名や行番号などのCOBOL言語レベルの情報で出力します。

#### 応答なし

診断機能は、実行中のプログラムやプログラムの呼出し関係およびスレッドの状態など、応答しなくなった箇所を絞り込むための情報をCOBOL言語レベルで出力します。

#### 領域破壊

診断機能は、特定領域の書き込みを監視して、領域の書き込みが行われるたびに書き込まれた内容と書き込みを行った位置をレポートに出力します。この診断レポートから領域を破壊したプログラムを簡単に特定することができます。

また、以下の問題に対しては、診断レポートと合わせてダンプを出力します。

- ・ スタックオーバーフロー例外(0xC00000FD)を除くアプリケーションエラー
- ・ 以下のUレベル実行時メッセージ
  - JMP0009I-U
  - JMP0010I-U
  - JMP0370I-U

診断機能は、アプリケーションのデバッグに使用できるだけでなく、運用時にアプリケーションエラーや実行時メッセージが発生したときのトラブルシューティングにも使用することができます。

診断機能には、以下の特徴があります。

- ・ 運用モジュールをそのまま対象としますので、本機能を使用するためにアプリケーションを作成し直す必要がありません。
- ・ 翻訳時に作成したデバッグ情報ファイルがあるとCOBOL言語レベルの情報を出力することができます。
- ・ 上位ツールであるPowerCOBOLと連携します。システムにPowerCOBOLがインストールされている場合には、PowerCOBOLプログラムに関する情報をPowerCOBOLの言語イメージで出力することができます。PowerCOBOL連携に関する詳細については、“PowerCOBOLユーザズガイド”を参照してください。

### 19.6.2 診断機能が使用する資源

---

ここでは、診断機能を使用して診断レポートを出力する場合に、診断機能が扱うプログラムおよび資源について説明します。

#### 19.6.2.1 対象プログラム

アプリケーションエラーやUレベルの実行時メッセージに対しては、診断機能はCOBOLの実行環境が開設している間だけ有効となります。したがって、以下のプログラムが対象となります。

- ・ COBOLプログラム
- ・ COBOLプログラムと言語間結合している他言語プログラム(ただし、COBOLの実行環境が開設してから終了するまでに動作する他言語プログラムだけが対象)

診断機能は、ユーザ資産の互換性を保っているため、旧資産を再翻訳や再リンクすることなく使用できます。

## 19.6.2.2 プログラムの翻訳・リンク方法と出力情報との関係

診断機能が対象とするプログラムの作成にあたっては、特別な制約はありません。診断機能は、COBOLプログラムであればどのようなプログラムでも診断レポートを出力します。ただし、翻訳オプションおよびリンクオプションの指定の組み合わせによって、診断レポートに出力される情報が異なります。翻訳オプションとリンクオプションの組み合わせ([1]～[6])と診断レポートから得られる情報の関係についてまとめたものを“表19.3 COBOLプログラムと出力情報の関係”に示します。

表19.3 COBOLプログラムと出力情報の関係

| 手順  | 詳細                          | [1]           | [2] | [3] | [4] | [5] | [6] |
|-----|-----------------------------|---------------|-----|-----|-----|-----|-----|
| 翻訳  | TESTオプションの指定                | なし            | なし  | あり  | あり  | あり  | あり  |
| リンク | /DEBUGおよび/DEBUGTYPEオプションの指定 | なし            | あり  | なし  | なし  | あり  | あり  |
| 実行  | デバッグ情報ファイルの有無               | なし            | なし  | なし  | あり  | なし  | あり  |
| 診断  | 情報レベル                       | EXPORT名+オフセット | あり  | あり  | あり  | あり  | あり  |
|     |                             | シンボル名+オフセット   | なし  | あり  | なし  | あり  | あり  |
|     |                             | プログラム名+行番号    | なし  | なし  | なし  | なし  | あり  |



### 参照

EXPORT名やシンボル名の詳細については、“19.6.4.2.1 アプリケーションエラー、実行時メッセージおよび応答なしの場合”の“EXPORT 相対位置”および“シンボル相対位置”を参照してください。

翻訳リンク/実行の条件で、診断レポートに出力される情報レベルが決まります。

たとえば、翻訳リンク時にオプションの指定がない([1]の条件の場合)、診断レポートには“EXPORT名+オフセット”レベルの情報が出力されます。

診断レポートにプログラム名や行番号などのCOBOL言語レベルの情報が出力されていれば簡単に問題が発生した文を特定できるため、翻訳リンク時にはオプションを指定し、かつ、診断機能がデバッグ情報ファイルを参照できる状態([6]の条件)でアプリケーションを実行することをおすすめします。

診断機能が言語レベルまでの情報を出力できる場合は、以下の例の形式で出力します。

- 出力例：“表19.3 COBOLプログラムと出力情報の関係”の条件[6]の場合

```
モジュールファイル: D:\APL\SAMPDLL2.dll
セクション相対位置: .text+0000025D
EXPORT相対位置: SAMPDLL2+00000229
シンボル相対位置: SAMPDLL2+0000025D
翻訳情報: ShiftJIS, シングルスレッド, NOOPTIMIZE
外部プログラム/クラス: SAMPDLL2
ソースファイル: SAMPDLL2.cob
文位置: 35
```

診断機能が以下に示すプログラムを診断した場合には、言語レベルの情報は出力できませんが、EXPORT名やシンボル名とそれぞれの先頭からのオフセットを出力します。

- 翻訳オプションおよびリンクオプションの両方、または一方を指定していない場合
- 翻訳オプションおよびリンクオプションの両方を指定していても、実行時に診断機能がデバッグ情報ファイルを読み込めなかった場合

EXPORT名やシンボル名とそれぞれの先頭からのオフセットは、以下の例の形式で出力します。



- EXPORT名とオフセットだけの出力例(“表19.3 COBOLプログラムと出力情報の関係”の条件[1][3][4]の場合)

```
モジュールファイル: D:\$APL\SAMPDLL2.dll
セクション相対位置: .text+0000025D
EXPORT相対位置: SAMPDLL2+00000229
```

- シンボル名とオフセットまでの出力例(“表19.3 COBOLプログラムと出力情報の関係”の条件[2][5]の場合)

```
モジュールファイル: D:\$APL\SAMPDLL2.dll
セクション相対位置: .text+0000025D
EXPORT相対位置: SAMPDLL2+00000229
シンボル相対位置: SAMPDLL2+0000025D
```

このEXPORT名やシンボル名は翻訳単位を表していますので、言語レベルの情報が出力されていなくても簡単に翻訳単位を特定することができます。

さらに、問題発生箇所を特定するには、目的プログラムリストが必要になります。目的プログラムリストは翻訳オプションLISTおよびPRINTを指定して翻訳すると作成されます。[参照]“A.2.27 LIST(目的プログラムリストの出力の可否)”、“A.2.38 PRINT(各種翻訳リストの出力の可否および出力先の指定)”

目的プログラムリストの例を以下に示します。EXPORT名相対オフセットおよびシンボル名相対オフセットから問題発生箇所を特定する方法を説明します。

| 番地                        | 機械語         | 手続き名          | アセンブラ形式命令         |
|---------------------------|-------------|---------------|-------------------|
| BEGINNING OF SAMPDLL2     |             |               |                   |
| [1] → 00000000 31         | db          |               | 0x31              |
| 00000001 08               | db          |               | 0x08              |
| 00000002 53414D50444C4C32 | dc          |               | "SAMPDLL2"        |
| 0000000A 08               | db          |               | 0x08              |
| 0000000B 4E6574434F424F4C | dc          |               | "NetCOBOL"        |
| 00000013 07               | db          |               | 0x07              |
| 00000014 XXXXXXXXXXXXXXXX | dc          |               | "XXXXXXXX"        |
| 0000001B XXXXXXXXXXXXXXXX | dc          |               | "XXXXXXXX"        |
| 00000023 XXXXXXXXXXXXXXXX | dc          |               | "XXXXXXXX"        |
| 0000002B 2B30303030       | dc          |               | " +0000"          |
| 00000030 2031             | dc          |               | " 1"              |
| 00000032 0000             | db          |               | 0x0000            |
| 00000034                  | PROLOGUEAD: |               |                   |
| 00000034 XXXXXXXX         | db          |               | 0XXXXXXXX : PLB.1 |
| [2] → 00000038            | _SAMPDLL2:  |               |                   |
| 00000038 55               | push        | ebp           |                   |
| 00000039 8BEC             | mov         | ebp, esp      |                   |
| 0000003B 81ECB0000000     | sub         | esp, 000000B0 |                   |
| 00000041 8BD7             | mov         | edx, edi      |                   |
| 00000043 8BFD             | mov         | edi, ebp      |                   |
| 00000045 81EF54000000     | sub         | edi, 00000054 |                   |
| 省略                        |             |               |                   |

シンボル名+オフセット0は図中[1]の位置になります。したがって、シンボル名相対オフセットは目的プログラムリストに出力されるオフセットと一致します。シンボル名相対オフセットと目的プログラムリストがあれば、簡単に問題発生箇所を特定することができます。

EXPORT名+オフセット0は図中[2]の位置になります。EXPORT名相対オフセットをもとに目的プログラムリストから問題発生箇所を特定するには、[1]と[2]の差分を意識しなければなりません。この例の場合は、EXPORT名相対オフセットに38(16進数)を加えると、目的プログラムリストのオフセットと一致します。

### 19.6.2.3 プログラムの翻訳・リンク

診断機能に言語レベルの情報を出力させるには、プログラムの翻訳およびリンク時にオプションを指定する必要があります。

プログラムの翻訳時に指定する翻訳オプションを以下に示します。

なお、ここでは、他言語プログラムの翻訳に使用するコンパイラをVisual C++として記載しています。

|                |                          |
|----------------|--------------------------|
| COBOL プログラムの場合 | : TEST                   |
| C/C++ プログラムの場合 | : 診断機能のために指定する翻訳オプションはない |

COBOLソースプログラムを翻訳するときは、翻訳オプションTESTを指定します。翻訳オプションTESTを指定すると、コンパイラは、オブジェクトファイル内に診断機能を使用する情報を出力するとともにデバッグ情報ファイルを作成します。デバッグ情報ファイルは、COBOLソースファイル名の拡張子をSVDに置き換えたファイル名になります。

翻訳オプションTESTと同時に、最適化を指示する翻訳オプションOPTIMIZEを指定することもできます。翻訳オプションTESTとOPTIMIZEを指定した場合に作成されるデバッグ情報ファイルは、翻訳オプションTESTだけを指定した場合に作成されるファイルよりもファイルサイズが小さくなります。それは、デバッグ情報ファイルに出力される情報量が限定されるためです。

## 注意

翻訳オプションOPTIMIZEを指定したデバッグ情報ファイルは、対話型デバッガでは使用できません。

C/C++ソースプログラムを翻訳するときは、最適化を指示する翻訳オプションのひとつである翻訳オプション/Oy(フレームポインタを作成しない)を指定しないでください。翻訳オプション/Oyを指定すると、診断レポートの呼出経路やスタックサマリから調査に必要な情報が得られなくなります。なお、最適化を指示する翻訳オプション/Ox(最大限の最適化)、/O1(プログラムサイズの最適化)、/O2(実行速度の最適化)の指定によっても翻訳オプション/Oy(フレームポインタを作成しない)が有効になる場合があるので、注意してください。

プログラムのリンク時に指定するリンクオプションを以下に示します。

|                                     |
|-------------------------------------|
| /DEBUG<br>/DEBUGTYPE: {COFF   BOTH} |
|-------------------------------------|

COBOLプログラムおよび他言語プログラムのオブジェクトファイルをリンクするときには、リンクオプション“/DEBUG”および“/DEBUGTYPE”を指定します。これらのオプションを指定すると、リンクはオブジェクトファイル内に出力された診断機能に必要な情報を、実行可能ファイルやダイナミックリンクライブラリに出力します。

診断機能の対象プログラムを作成するには、リンクオプションに“/DEBUG /DEBUGTYPE:COFF”を指定します。

なお、プロジェクト管理機能を利用すると簡単に診断機能の対象プログラムを作成することができます。詳細については、“[6.4.5.1 デバッグモジュールの作成方法](#)”を参照してください。

Visual C++と結合したプログラムのデバッグを行う必要がある場合には、リンクオプションに“/DEBUG /DEBUGTYPE:BOTH”を指定します。他の言語製品については、各製品のマニュアルに従ってください。

## 参考

目的プログラムリストとシンボル名相対オフセットがあれば、簡単に問題発生箇所を特定することができます。したがって、リンクオプション“/DEBUG”および“/DEBUGTYPE”を指定しておくことをおすすめします。

### 19.6.2.4 デバッグ情報ファイルの格納先

診断機能にCOBOL言語レベルの情報を出力させる場合は、デバッグ情報ファイルを実行可能ファイルやダイナミックリンクライブラリと同じフォルダに格納します。

診断機能を運用環境で使用する場合には、運用マシンに実行可能ファイルやダイナミックリンクライブラリとともにデバッグ情報ファイルも一緒に格納します。デバッグ情報ファイルが存在しない場合、診断機能はEXPORT名やシンボル名、相対オフセットなどのアセンブラレベルの情報を出力します。診断機能の出力情報については、“[19.6.2.2 プログラムの翻訳・リンク方法と出力情報との関係](#)”を参照してください。

## 19.6.3 診断機能の起動

### 19.6.3.1 アプリケーションエラーやUIレベルの実行時メッセージが発生するアプリケーションに対する起動

アプリケーションエラーやUIレベルの実行時メッセージが発生すると、診断機能はエラーを検出して自動的に起動します。

診断機能の起動は、環境変数情報@CBR\_JUSTINTIME\_DEBUGの指示によって制御することができます。環境変数情報@CBR\_JUSTINTIME\_DEBUGが設定されていない場合には、標準で診断機能が起動するようになっています。[参照]“5.4.1.32 @CBR\_JUSTINTIME\_DEBUG (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)”

また、診断機能の動作は、環境変数情報@CBR\_JUSTINTIME\_DEBUGに起動パラメタを指定することによって制御できます。[参照]“表19.4 起動パラメタ”

起動パラメタの指定形式の例を以下に示します。以下の例では、診断レポートファイルの出力先フォルダの変更とイベントログへの事象発生通知を指示しています。



例

```
@CBR_JUSTINTIME_DEBUG=ALLERR, SNAP -r c:\¥log -l
```

### 19.6.3.2 応答しなくなったアプリケーションに対する起動

応答しなくなったアプリケーションに対して診断機能を起動するには、手動で直接診断機能を起動します。起動時には、起動パラメタ“-p”で診断するアプリケーションのプロセスIDを指定する必要があります。[参照]“表19.4 起動パラメタ”

アプリケーションのプロセスIDを取得するには、コマンドプロンプトで以下の形式で入力します。

COBSNAP

コマンドが実行されると、以下で示すように、プロセスID(16進数)とアプリケーション名の一覧が表示されます。

| PID | Application      |
|-----|------------------|
| 0   | [System Process] |
| 4   | System           |
| 3F0 | smss.exe         |
| 430 | csrss.exe        |
| 448 | winlogon.exe     |
| 474 | services.exe     |
| 480 | lsass.exe        |
| 52C | testpgm.exe      |

省略

一覧の中から応答しなくなったアプリケーションの名前を検索し、プロセスIDを取得します。

応答しなくなったアプリケーションに対して診断機能を起動する場合の指定形式を、以下に示します。



例

```
COBSNAP -p 0x52C
```

### 19.6.3.3 領域破壊が発生するアプリケーションに対する起動

領域破壊が発生するアプリケーションに対して診断機能を起動するには、環境変数情報@CBR\_ATTACH\_TOOLで診断機能の起動を指示します。環境変数情報@CBR\_ATTACH\_TOOLには起動パラメタ“-w”で監視する領域のアドレスを指定する必要があります。監視領域アドレスには、ランタイムシステム領域のメモリチェック機能によって通知された破壊領域のアドレスを指定します。

領域破壊が発生するアプリケーションに対して診断機能を起動する場合の指定形式を、以下に示します。



例

```
@CBR_ATTACH_TOOL=SNAP -w 0xDC1198
```

環境変数情報@CBR\_ATTACH\_TOOLの指定方法については、“[5.4.1.2 @CBR\\_ATTACH\\_TOOL \(プログラムからデバッグまたは診断機能を起動する指定\)](#)”を参照してください。また、ランタイムシステム領域のメモリチェック機能については、“[19.5 メモリチェック機能の使い方](#)”を参照してください。



環境変数情報@CBR\_JUSTINTIME\_DEBUGを設定している場合には、診断機能を起動する前に設定を削除してください。

### 19.6.3.4 起動パラメタ

診断機能の起動パラメタを“[表19.4 起動パラメタ](#)”に示します。起動パラメタは、診断機能の起動方法により指定する箇所が異なります。

- アプリケーションエラーやUレベルの実行時メッセージが発生するアプリケーションに対する起動の場合は、環境変数情報@CBR\_JUSTINTIME\_DEBUGに指定します。
- 応答しなくなったアプリケーションに対する起動の場合は、COBSNAPコマンドに指定します。
- 領域破壊が発生するアプリケーションに対する起動の場合は、環境変数情報@CBR\_ATTACH\_TOOLに指定します。

診断機能の起動の目的によって各起動パラメタが有効になる場合を、以下の表で○で示します。

表19.4 起動パラメタ

| 指定形式          | a | b | c | 内容                                                                                                                                                                                                                                                                                            |
|---------------|---|---|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -d{YES   NO}  | ○ | — | — | <p>ダンプを出力するかどうかを指定します。</p> <ul style="list-style-type: none"> <li>• YES:ダンプを出力する</li> <li>• NO :ダンプを出力しない</li> </ul> <p>この起動パラメタが省略された場合は、YESが指定されたとみなします。</p>                                                                                                                                |
| -i{0   1   2} | ○ | ○ | — | <p>出力内容を指定します。</p> <ul style="list-style-type: none"> <li>• 2 :環境変数の情報、初期化ファイル名および各種情報ファイルの内容を出力する</li> <li>• 1 :環境変数の情報および初期化ファイル名を出力する</li> <li>• 0 :環境変数および初期化ファイル名を出力しない</li> </ul> <p>この起動パラメタが省略された場合は、2が指定されたとみなします。</p>                                                               |
| -o フォルダ名      | ○ | — | — | <p>ダンプファイルの出力先フォルダを絶対パスまたは相対パスで指定します。フォルダには、実在するフォルダを指定してください。(注)<br/>相対パスで指定されたときは、実行可能ファイルが存在するフォルダからの相対パスになります。<br/>指定された出力先フォルダが実在しない場合および出力ができない場合には、標準の出力先フォルダに出力します。<br/>この起動パラメタが省略された場合、標準の出力先フォルダに出力します。<br/>標準の出力先フォルダについては、“<a href="#">19.6.5.2 ダンプの出力先</a>”を参照してください。</p>       |
| -r フォルダ名      | ○ | ○ | ○ | <p>診断レポートファイルの出力先フォルダを絶対パスまたは相対パスで指定します。フォルダには、実在するフォルダを指定してください。(注)<br/>相対パスで指定されたときは、実行可能ファイルが存在するフォルダからの相対パスになります。<br/>指定された出力先フォルダが実在しない場合および出力ができない場合には、標準の出力先フォルダに出力します。<br/>この起動パラメタが省略された場合、標準の出力先フォルダに出力します。<br/>標準の出力先フォルダについては、“<a href="#">19.6.4.1 診断レポートの出力先</a>”を参照してください。</p> |
| -m [コンピュータ名]  | ○ | — | — | <p>アプリケーションエラーまたは実行時メッセージが発生したことを、メッセージボックスで通知する場合に指定します。<br/>コンピュータ名を指定したときは、指定したコンピュータにメッセージボックスで通知します。</p>                                                                                                                                                                                 |

| 指定形式         | a | b | c | 内容                                                                                                                                                                                                                        |
|--------------|---|---|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              |   |   |   | コンピュータ名が指定されない場合には、問題が発生したコンピュータにメッセージボックスで通知します。<br>この起動パラメタは、Windows Vista、Windows 7、Windows 8、Windows Server 2008、およびWindows Server 2012では無効です。                                                                        |
| -l [コンピュータ名] | ○ | — | — | アプリケーションエラーまたは実行時メッセージが発生したことを、イベントログへ出力する場合に指定します。コンピュータ名を指定したときは、指定したコンピュータのイベントログへ出力します。<br>コンピュータ名が指定されない場合には、問題が発生したコンピュータのイベントログへ出力します。                                                                             |
| -t パラメタ数     | ○ | ○ | — | スタックサマリに出力するパラメタの数を指定します。パラメタ数には1～16の数を指定します。この起動パラメタが省略された場合、8が指定されたとみなします。                                                                                                                                              |
| -s 出力単位数     | ○ | ○ | — | スタックダンプに出力するサイズを出力単位数(1単位=1024バイト)で指定します。出力単位数には0以上の数を指定します。<br><ul style="list-style-type: none"> <li>・ 0 : スタックのすべての内容を出力する</li> <li>・ 1 以上: 指定された値×1024バイトの大きさのスタックの内容を出力する</li> </ul> この起動パラメタが省略された場合、2が指定されたとみなします。 |
| -p プロセスID    | — | ○ | — | 応答しなくなったアプリケーションのプロセスIDを指定します。<br>プロセスIDは、先頭に'0x'を付けた16進数で指定します。                                                                                                                                                          |
| -w アドレス[長さ]  | — | — | ○ | 監視する領域のアドレスと長さを指定します。<br>アドレスは、先頭に'0x'を付けた16進数で指定します。<br>長さは、4、8、12、16のうちのどれかを指定します。長さが省略された場合、4が指定されたとみなします。                                                                                                             |

注: 指定するフォルダには、Everyoneグループに対する以下のアクセス権が必要です。

- ・ 変更
- ・ 読み取りと実行
- ・ フォルダの内容の一覧表示
- ・ 読み取り
- ・ 書き込み

## 注意

- ・ 起動パラメタの先頭文字は、ハイフン(-)でもスラッシュ(/)でも同様に扱います。
- ・ 起動パラメタの先頭文字に続くパラメタ名は、大文字と小文字を同様に扱います。
- ・ パラメタ名とパラメタ名に続く文字列との間は、空白を置いても置かなくてもどちらでも指定できます。
- ・ 複数の起動パラメタを指定する場合には、起動パラメタと次の起動パラメタとの間に1つ以上の空白を置いて指定します。
- ・ パラメタのフォルダ名に空白を含む場合には、フォルダ名を二重引用符(")で囲んで指定します。また、フォルダ名の先頭がハイフン(-)の場合にも、フォルダ名を二重引用符で囲んで指定します。
- ・ 起動パラメタ“-w”に長さを指定する場合には、コンマ(,)の前後に空白を置かずにアドレスと長さを指定します。

## 参考

アプリケーションエラーまたは実行時メッセージの場合に、起動パラメタ“-m”および“-l”の両方が省略されている場合には、問題が発生したコンピュータにメッセージボックスで通知します。

## 19.6.4 診断レポート

### 19.6.4.1 診断レポートの出力先

診断機能は、診断情報を診断レポートファイルに出力します。

ファイル名は、アプリケーションの名前を「アプリケーション名\_プロセスID\_エラー発生時間」に変更し、拡張子を「LOG」に置き換えた名前になります。



#### 例

アプリケーション名 : TESTPGM. EXE  
プロセスID : 2D8  
エラー発生時間 : 2009年07月10日15時04分26秒  
診断レポートファイル名 : TESTPGM\_2D8\_20090710-150426. LOG

診断レポートの標準の出力先は、ユーザ共通のアプリケーションデータフォルダ(コンピュータごとにアプリケーション固有のデータを格納するフォルダ)配下の¥Fujitsu¥NetCOBOL¥COBSNAPです。

通常、診断レポートの標準の出力先は、以下の名前で作成されます。

- Windows XPおよびWindows Server 2003の場合

C:¥Documents and Settings¥All Users¥Application Data¥Fujitsu¥NetCOBOL¥COBSNAP

- Windows Vista、Windows 7、Windows 8、Windows Server 2008、およびWindows Server 2012の場合

C:¥ProgramData¥Fujitsu¥NetCOBOL¥COBSNAP

出力先フォルダを変更する場合は、起動パラメタ“-r”で指定します。起動パラメタの詳細については、“表19.4 起動パラメタ”を参照してください。



#### 注意

ユーザ共通のアプリケーションデータフォルダは、通常、読み取り専用のフォルダです。しかし、診断レポートの標準出力先フォルダは、Everyoneグループに対して以下のアクセス権を持つフォルダとして生成されます。

- 変更
- 読み取りと実行
- フォルダの内容の一覧表示
- 読み取り
- 書き込み

診断機能は、事象発生の通知をメッセージボックスおよびアプリケーションのイベントログに出力します。メッセージボックスには、事象の発生を示すメッセージと診断レポートを開くかどうかを問い合わせるメッセージが出力されます。メッセージボックスの[はい]ボタンをクリックすれば、その場で簡単に診断レポートを開くことができます。



#### 注意

アプリケーションがサービス配下で動作している場合などは、メッセージボックスから診断レポートを開くことはできません。

イベントログの各項目に対しては、ソース名はNetCOBOL SNAP、イベントIDはメッセージ番号、種類は重大度コード、説明にはメッセージ本文を出力します。ただし、種類についてはレベルの区分がシステムと診断機能のメッセージとで一致しないため、次のように対応付けています。



表19.5 診断機能メッセージの重大度コードとイベントログの種類の対応

| 重大度コード            | イベントログの種類        |
|-------------------|------------------|
| I (INFORMATION)   | 情報 (INFORMATION) |
| W (WARNING)       | 警告 (WARNING)     |
| E (ERROR)         |                  |
| U (UNRECOVERABLE) | エラー (ERROR)      |

事象発生通知の標準の出力先は、メッセージボックスです。出力先を変更する場合は、起動パラメタ“-m”および“-l”を指定します。起動パラメタの詳細については、“[表19.4 起動パラメタ](#)”を参照してください。

なお、ディスク容量不足などのエラーにより診断レポートが出力できなかった場合には、事象発生の通知先にエラーメッセージを出力します。



サービスの多くは、画面による入出力を行いません。画面による入出力を行わないサービス配下で利用する場合には、起動パラメタ“-l”を指定して、事象発生の通知先にイベントログを指定してください。

## 19.6.4.2 診断レポートの出力情報

診断機能は、診断レポートに以下の情報を出力します。

### 19.6.4.2.1 アプリケーションエラー、実行時メッセージおよび応答なしの場合

診断レポートには、以下の情報が出力されます。

#### 要約

診断情報の要約を出力します。

#### 検出事象

診断レポートは常に以下の行から始まります。

アプリケーションエラーが発生しました：

または

実行時メッセージが発生しました：

または

応答のないアプリケーションを診断しました：

#### アプリケーション

アプリケーション名(絶対パス形式)とプロセスID(16進数)を出力します。

#### 例外種別

アプリケーションエラーの場合は、例外コード(16進数)と例外名を出力します。例外コードと例外名については、“[表19.6 例外コードと例外名](#)”を参照してください。

実行時メッセージの場合は、実行時メッセージ本文を出力します。応答なしの場合は出力しません。

#### 発生時間

アプリケーションエラーおよび実行時メッセージの場合は、エラーが発生した時間を出力します。応答なしの場合は、診断機能が起動された時間を出力します。

#### 発生モジュール

アプリケーションエラーおよび実行時メッセージの場合に、エラーが発生したモジュールの名前(絶対パス形式)、作成時間およびファイルサイズを出力します。応答なしの場合は出力しません。

## 問題箇所

アプリケーションエラーおよび実行時メッセージの場合に、エラーが発生した位置を出力します。応答なしの場合は出力しません。

### スレッドID

エラーが発生したスレッドのスレッドID(16進数)を出力します。

### レジスタ

エラーが発生した時点でのレジスタの値(16進数)を一覧で出力します。

### スタックコミット

エラーが発生した時点でのスタックの状態(すべて16進数)を出力します。

#### ー コミットサイズ

割り当てられているスタックの大きさ(コミットサイズ=トップアドレス-ベースアドレス)

#### ー トップアドレス

スタックの最上位アドレス

#### ー ベースアドレス

割り当てられているスタックの最下位アドレス

### 命令

エラーが発生した位置の前後16バイトの機械語コード(16進数)を出力します。

### モジュールファイル名

エラーが発生したモジュールの名前を絶対パス形式で出力します。

### セクション相対位置

モジュールの.textセクション先頭からの相対値(16進数)を出力します。

### EXPORT相対位置

モジュールにEXPORT情報が存在する場合に、エラー発生位置のアドレスに最も近いアドレスを持つEXPORT名と、EXPORT名先頭からの相対値(16進数)を出力します。

COBOLプログラムのEXPORT名を以下に示します。

|                 |                                                                                                                                     |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------|
| 外部プログラム         | : 外部プログラム名                                                                                                                          |
| ENTRY           | : エントリ名                                                                                                                             |
| クラス             | : _クラス名_FACTORY                                                                                                                     |
| メソッド            | : __クラス名_メソッド名####<br>または<br>__クラス名__ENTRY_メソッド名####                                                                                |
| メソッド (PROPERTY) | : __クラス名__GET_メソッド名####<br>または<br>__クラス名__SET_メソッド名####<br>または<br>__クラス名__ENTRY_GET_メソッド名####<br>または<br>__クラス名__ENTRY_SET_メソッド名#### |

他言語プログラムのEXPORT名を以下に示します。

関数 : 関数名または\_関数名####



## 参考

メソッド名や関数名に続く“@####”の####は、パラメタのバイト数を表す10進数です。



シンボル名やプログラム名が出力されない場合、翻訳単位ごとにEXPORT名が定義してあれば、EXPORT名から翻訳単位を特定することができます。

## シンボル相対位置

モジュールにCOFFシンボル情報が存在する場合に、シンボル名とシンボル名先頭からの相対値(16進数)を出力します。COBOLプログラムのシンボル名とは、翻訳単位名を指します。COBOLプログラムの翻訳単位名を以下に示します。

|                 |                                           |
|-----------------|-------------------------------------------|
| 外部プログラム         | : 外部プログラム名                                |
| クラス             | : クラス名                                    |
| メソッド            | : クラス名_メソッド名                              |
| メソッド (PROPERTY) | : クラス名_GET_メソッド名<br>または<br>クラス名_SET_メソッド名 |

他言語プログラムのシンボル名は、関数名の先頭に“\_”が付いた名前になります。

## 翻訳情報

COBOLプログラムの場合に、プログラム翻訳時の情報を出力します。他言語プログラムの場合は、出力しません。

- コード系(ShiftJIS、UnicodeまたはJEF)
- オブジェクト形式(シングルスレッドまたはマルチスレッド)
- 最適化(OPTIMIZEまたはNOOPTIMIZE)

## プログラム名

COBOLプログラムの場合は、外部プログラム名またはクラス名、および内部プログラム名またはメソッド名を出力します。他言語プログラムの場合は、関数名を出力します。

## ソースファイル名

エラーが発生したプログラムのソースファイル名を出力します。

## 行番号

エラーが発生した文の行番号を出力します。

## 補足情報

COBOLプログラムの場合で、かつエラー発生位置が宣言節手続き内の文であった場合に、宣言節への分岐元の文を出力します。エラー発生位置が宣言節手続き内の文でない場合には出力しません。

## 呼出経路

エラーが発生したプログラムまでの呼出経路を出力します。なお、デバッグ情報ファイルが存在しない場合は、内部プログラムの呼出し元は出力しません。

## システム情報

コンピュータに関する一般的なシステム情報を出力します。

- コンピュータ名
- ユーザ名
- Windowsバージョン
- バージョンナンバー
- サービスパック

## コマンドライン

アプリケーションの実行時に指定されたコマンド文字列を出力します。

## 環境変数

アプリケーションの実行時に設定されていた環境変数を出力します。

## 実行環境情報

COBOLの環境情報として以下の情報を出力します。なお、実行用の初期化ファイルが存在しない場合は、初期化ファイル名とプログラム名に“なし”と出力します。また、実行用の初期化ファイルが存在している場合でかつプログラムがマルチスレッドで動作している場合には、プログラム名に“なし”と出力します。

### ランタイムシステム

COBOLランタイムシステムのバージョンとモジュールタイプを出力します。

### 実行モード

ランタイムシステムの実行時の情報を出力します。

- コード系(ShiftJIS、UnicodeまたはJEF)
- 動作モード(シングルスレッドまたはマルチスレッド)

### プログラム名

ランタイムシステムが認識しているCOBOLの主プログラム(初期化ファイルのセクション名に対応する)

### 初期化ファイル

COBOLプログラムの実行時にランタイムシステムが参照する実行用の初期化ファイルの名前(絶対パス形式)とファイルの内容

### 各種情報ファイル

以下の情報ファイルの名前(絶対パス形式)とファイルの内容

- エントリ情報ファイル
- 論理宛先定義ファイル
- ODBC情報ファイル
- 印刷情報ファイル

## スタックヒープ情報

実行可能ファイルのリンク時に指定されたスタックとヒープのサイズ(16進数)を出力します。

## タスクリスト

システムで実行していたタスクのプロセスID(16進数)とアプリケーション名の一覧を出力します。

## モジュールリスト

アプリケーションがローディングしたモジュールに関して、ファイル名、バージョン情報、作成時間を出力します。一覧はローディングした順番で出力します。

### スタックサマリ

スタックの概略として以下の情報を出力します。なお、内部プログラムの情報は出力しません。

- フレームポインタ  
呼び出されたプログラムのスタックフレームの先頭アドレス(16進数)
- リターンアドレス  
呼び出されたプログラムの復帰先アドレス(16進数)
- パラメタ  
呼び出されたプログラムに渡されたパラメタ(16進数)
- モジュールファイル名とプログラム名  
呼び出されたプログラムのモジュールファイル名とプログラム名

パラメタには、スタック内のパラメタが格納される位置(フレームポインタ+8バイト)のメモリ内容を出力します。

### スタックダンプ

エラーが発生した時点でのスタックの内容を16進表示とASCII表示で出力します。

## スレッド情報

アプリケーションエラーおよび実行時メッセージの場合は、エラーが発生したスレッド以外のスレッドについて以下の情報を出力します。応答なしの場合は、アプリケーションで生成されているスレッドの数だけ以下の情報を出力します。

- スレッドID
- レジスタ
- スタックコミット
- モジュールファイル名
- セクション相対位置
- EXPORT相対位置
- シンボル相対位置
- 翻訳情報
- プログラム名
- ソースファイル名
- 行番号
- 呼出経路
- スタックサマリ
- スタックダンプ

システムで定義されている例外コードと診断レポートに出力する例外コードに対する例外名を“表19.6 例外コードと例外名”に示します。

表19.6 例外コードと例外名

| コード      | 例外名                                                                |
|----------|--------------------------------------------------------------------|
|          | 説明                                                                 |
| C0000005 | EXCEPTION_ACCESS_VIOLATION                                         |
|          | プログラムが仮想アドレスに対して読取りまたは書込みをしようとしたが、適切なアクセス権を持っていませんでした。             |
| C0000006 | EXCEPTION_IN_PAGE_ERROR                                            |
|          | プログラムが存在しなかったページをアクセスしようとした結果、システムがページのロードに失敗しました。                 |
| C0000017 | EXCEPTION_NO_MEMORY                                                |
|          | メモリ不足、またはヒープが壊れていることが原因で、メモリの割り当てに失敗しました。                          |
| C000001D | EXCEPTION_ILLEGAL_INSTRUCTION                                      |
|          | プログラムがプロセッサに定義されていない命令を実行しようとした。                                   |
| C0000025 | EXCEPTION_NONCONTINUABLE_EXCEPTION                                 |
|          | 継続不可能な例外が発生した後に、プログラムが再実行しようとした。                                   |
| C0000026 | EXCEPTION_INVALID_DISPOSITION                                      |
|          | 例外ハンドラは例外ディスパッチャに不当な配列を返しました。                                      |
| C000008C | EXCEPTION_ARRAY_BOUNDS_EXCEEDED                                    |
|          | プログラムが配列の範囲外をアクセスしようとして、それがハードウェアによって検出されました。                      |
| C000008D | EXCEPTION_FLT_DENORMAL_OPERAND                                     |
|          | 浮動小数点演算内のオペランドのひとつが異常です。この異常値は、値として小さすぎるために標準の浮動小数点値として表すことができません。 |
| C000008E | EXCEPTION_FLT_DIVIDE_BY_ZERO                                       |

| コード      | 例外名                                           |
|----------|-----------------------------------------------|
|          | 説明                                            |
|          | プログラムが、ある浮動小数点値を浮動小数点値0で除算しようとした。             |
| C000008F | EXCEPTION_FLT_INEXACT_RESULT                  |
|          | 浮動小数点演算の結果を小数として正確に表すことができません。                |
| C0000090 | EXCEPTION_FLT_INVALID_OPERATION               |
|          | この例外は、この表に記載されているもの以外の浮動小数点例外を表します。           |
| C0000091 | EXCEPTION_FLT_OVERFLOW                        |
|          | 浮動小数点演算の指数部の値が、対応する型の上限を超えています。               |
| C0000092 | EXCEPTION_FLT_STACK_CHECK                     |
|          | 浮動小数点演算の結果として、スタックのオーバーフローまたはアンダフローが発生しました。   |
| C0000093 | EXCEPTION_FLT_UNDERFLOW                       |
|          | 浮動小数点演算の指数部の値が、対応する型の下限を超えています。               |
| C0000094 | EXCEPTION_INT_DIVIDE_BY_ZERO                  |
|          | プログラムが、ある整数値を0で除算しようとした。                      |
| C0000095 | EXCEPTION_INT_OVERFLOW                        |
|          | 整数演算の結果、その最上位ビットが桁あふれを起こしました。                 |
| C0000096 | EXCEPTION_PRIV_INSTRUCTION                    |
|          | プログラムが命令を実行しようとしたが、その演算は現在のマシンモードでは許可されていません。 |
| C00000FD | EXCEPTION_STACK_OVERFLOW                      |
|          | スタックオーバーフローが発生しました。                           |

COBOLアプリケーションで発生しやすいアプリケーションエラーは、アクセス違反(EXCEPTION\_ACCESS\_VIOLATION)やゼロ除算(EXCEPTION\_INT\_DIVIDE\_BY\_ZERO)です。アクセス違反には、以下の原因が考えられます。

- 添字や指標の値が範囲外である。
- 部分参照で参照可能範囲外を参照する。
- プログラム呼出し時の呼出し規約やパラメタの個数が合っていない。

これらはCHECK機能で簡単に検査することができるため、エラーが発生した場合はCHECK機能で確認してください。[参照]“[19.2 CHECK機能の使い方](#)”

## 参考

COBOLプログラムのソースファイル名と行番号の出力形式は、翻訳オプションNUMBERおよび翻訳オプションOPTIMIZEの指定によって異なります。翻訳オプションと出力形式の関係を“[表19.7 翻訳オプションと出力形式](#)”に示します。

表19.7 翻訳オプションと出力形式

| 出力対象             |           | NOOPTIMIZE指定                         | OPTIMIZE指定                 |                              |
|------------------|-----------|--------------------------------------|----------------------------|------------------------------|
|                  |           | NUMBER/NONNUMBER<br>指定共通             | NUMBER指定                   | NONNUMBER指定                  |
| ソース<br>ファイル<br>名 | ソースファイル   | ファイル名だけ                              |                            |                              |
|                  | 登録集ファイル   | ファイル名だけ                              | 出力しない                      | ファイル名だけ                      |
|                  | 登録集取り込み経路 | 取り込まれた登録集ファイル名とその取り込み元ファイル名、および取り込み元 | 出力しない (ソースファイル名を出力する) (*4) | 取り込まれた登録集ファイル名とソースファイル名を出力し、 |

| 出力対象 |            | NOOPTIMIZE指定                  | OPTIMIZE指定                    |                     |
|------|------------|-------------------------------|-------------------------------|---------------------|
|      |            | NUMBER/NONNUMBER<br>指定共通      | NUMBER指定                      | NONNUMBER指定         |
|      |            | の行番号について、取り込み経路の最後までを出力する(*3) |                               | 取り込み元の行番号は出力しない(*5) |
| 行番号  | 入口コード内(*1) | IN ENTRY-CODE                 |                               |                     |
|      | 出口コード内(*2) | IN EXIT-CODE                  |                               |                     |
|      | 文          | ファイル相対行番号[. 行内追番]             | [COPY修飾値-]エディタ行番号(NUMBER)(*6) | ファイル相対行番号           |

#### [用語の補足]

\*1: 入口コード内とは、プログラムの入口コードの範囲を表しています。具体的には、目的プログラムリストのPROLOGUEAD:+4(16進数)の位置から END OF PROGRAM INITIALIZE ROUTINEで示される範囲を指します。

\*2: 出口コード内とは、プログラムの出口コードの範囲を表しています。具体的には、目的プログラムリストのBEGINNING OF GOBACK COMMON ROUTINEから END OF GOBACK COMMON ROUTINEで示される範囲を指します。

#### [出力例]

\*3: ソースファイル: CPY2.cbl <- CPY1.cbl <- SRC.cob 文位置: 86 <- 55 <- 127

\*4: ソースファイル: SRC.cob 文位置: 2-8600(NUMBER)

\*5: ソースファイル: CPY2.cbl <- SRC.cob 文位置: 86

\*6: 行番号の直後に識別子“(NUMBER)”を付加することによって、行番号がエディタ行番号なのかファイル相対行番号なのかを区別します。文位置: 95100(NUMBER)

## 19.6.4.2.2 領域破壊監視の場合

### 要約

診断情報の要約を出力します。

#### 検出事象

診断レポートは常に以下の行から始まります。

領域の書き込みを監視しました：

#### アプリケーション

アプリケーション名(絶対パス形式)とプロセスID(16進数)を出力します。

#### 起動時間

診断機能が起動された時間を出力します。

#### 監視アドレス

監視領域のアドレス(16進数)を出力します。

#### 監視サイズ

監視領域の大きさ(10進数)を出力します。

### 書き込み情報

監視領域へ書き込みを行った位置と書き込み後の領域の内容を書き込み順に出力します。なお、監視領域への書き込みが一度もなかった場合には、その旨を示す情報を診断レポートに出力します。

## カウンタ

情報を出力するたびにカウントアップした値を出力します。この値により、何番目の書き込みであるかがわかります。

## 監視領域

監視領域の内容を16進形式で出力します。

## 書き込み位置

書き込みを行った位置を出力します。

- スレッドID
- モジュールファイル名
- セクション相対位置
- EXPORT相対位置
- シンボル相対位置
- 翻訳情報
- プログラム名
- ソースファイル名
- 行番号
- 呼出経路

なお、領域破壊の監視中にアプリケーションエラーや実行時メッセージが発生した場合には、以下の情報を診断レポートに追加書きます。

## エラー情報

アプリケーションエラーや実行時メッセージに関する情報を出力します。

## カウンタ

情報を出力するたびにカウントアップした値を出力します。

## エラー要約

検出事象とエラーコードを出力します。

## 問題箇所

エラーが発生した位置を出力します。

- スレッドID
- レジスタ
- スタックコミット
- モジュールファイル名
- セクション相対位置
- EXPORT相対位置
- シンボル相対位置
- 翻訳情報
- プログラム名
- ソースファイル名
- 行番号
- 呼出経路

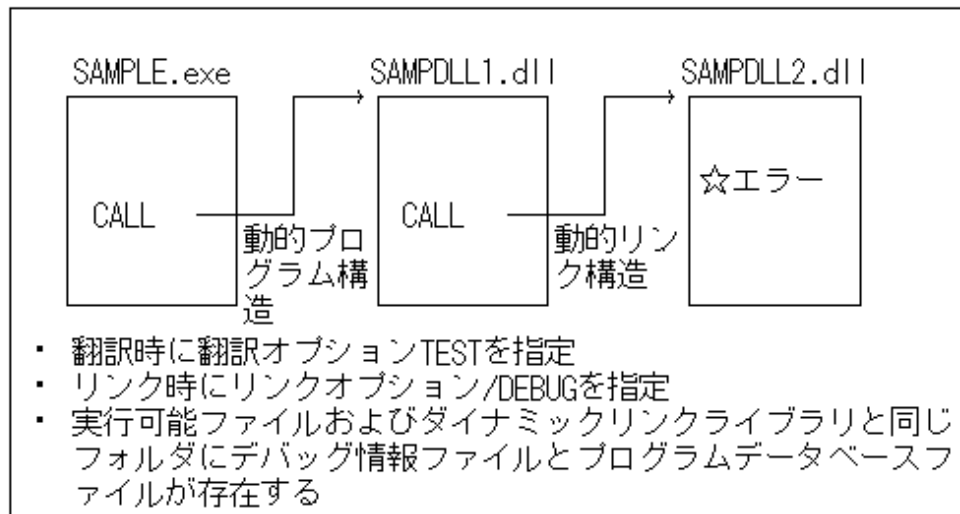
### 19.6.4.3 診断レポートの見方

ここでは、診断機能が出力した診断レポートをもとにトラブル調査する方法について、例を使って説明します。

#### 19.6.4.3.1 アプリケーションエラー

アプリケーションエラーの診断レポート例を以下に示します。

例で使用するプログラムは次のプログラムから構成されています。



#### NetCOBOL 診断機能

##### <<要約>>

アプリケーションエラーが発生しました:

アプリケーション : D:\APL\%SAMPLE.exe (PID=000000A7)  
例外種別 : EXCEPTION\_INT\_DIVIDE\_BY\_ZERO (C0000094)  
発生時間 : YYYY/MM/DD (HH:MM:SS)  
発生モジュール : D:\APL\%SAMPDLL2.dll  
作成時間 : YYYY/MM/DD (HH:MM:SS)  
ファイルサイズ : 30720bytes

##### <<問題箇所>>

スレッドID : 0000005E  
レジスタ : EAX=00000000 EBX=7FFDF065 ECX=00000000 EDX=00000000 ESI=001421A6  
ED1=00405108 EIP=03BB125D ESP=0012FCD8 EBP=0012FD98 EFL=00010256  
CS=001B SS=0023 DS=0023 ES=0023 FS=0038 GS=0000  
コミット : 00004000 (トップ:00130000, ベース:0012C000)  
命令 : アドレス +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +a +b +c +d +e +f  
03BB124D 03 0F BF 05 32 50 BB 03 99 0F BF 0D 2A 50 BB 03  
フォールト->03BB125D F7 F9 0F BF C8 83 F9 00 7D 02 F7 D9 0F C9 C1 E9

モジュールファイル : D:\APL\%SAMPDLL2.dll  
セクション相対位置 : .text+0000025D  
EXPORT相対位置 : SAMPDLL2+00000229  
シンボル相対位置 : SAMPDLL2+0000025D  
翻訳情報 : ShiftJIS, シングルスレッド, NOOPTIMIZE  
外部プログラム/クラス : SAMPDLL2  
ソースファイル : SAMPDLL2.cob  
文位置 : 35

##### <呼出経路>

[ 1]-----

```
モジュールファイル : D:\APL\SAMPDLL1.dll
セクション相対位置 : .text+0000026B
EXPORT相対位置 : SAMPDLL1+00000237
シンボル相対位置 : SAMPDLL1+0000026B
翻訳情報 : ShiftJIS, シングルスレッド, NOOPTIMIZE
外部プログラム/クラス : SAMPDLL1
ソースファイル : SAMPDLL1.cob
文位置 : 14
```

[ 2]-----

```
モジュールファイル : D:\APL\SAMPLE.exe
セクション相対位置 : .text+0000047A
シンボル相対位置 : SAMPLE+0000047A
翻訳情報 : ShiftJIS, シングルスレッド, NOOPTIMIZE
外部プログラム/クラス : SAMPLE
ソースファイル : SAMPLE.cob
文位置 : 24
```

[ 3]-----

```
モジュールファイル : D:\APL\SAMPLE.exe
セクション相対位置 : .text+0000000A
シンボル相対位置 : SAMPLE+0000000A
```

[ 4]-----

```
モジュールファイル : D:\APL\SAMPLE.exe
セクション相対位置 : .text+00000BEB
シンボル相対位置 : _WinMainCRTStartup+0000014B
```

[ 5]-----

```
モジュールファイル : C:\WINDOWS\System32\KERNEL32.dll
セクション相対位置 : .text+0001A623
EXPORT相対位置 : RegisterWaitForInputIdle+00000117
```

以降、省略

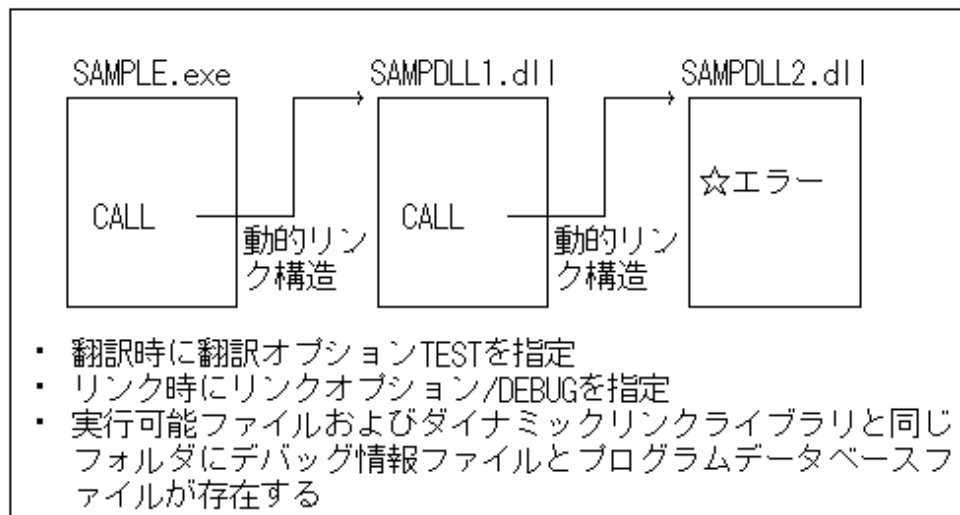
1. <<要約>>に着目します。ここでは、どのモジュールで、どんなアプリケーションエラーが発生したのか知ることができます。  
この例では、D:\APL\SAMPDLL2.DLLで、ゼロ除算例外が発生したことがわかります。
2. 次に<<問題箇所>>に着目します。ここでは、どの翻訳単位でエラーが発生したのか知ることができます。  
この例では、プログラムSAMPDLL2の35行目でエラーが発生したことがわかります。
3. エラーが発生したプログラムのソースファイルまたは目的プログラムリストから、どの文(命令)でエラーが発生したのかを特定します。  
この例では、35行目のCOMPUTE文でエラーが発生したと仮定します。
4. さらに、エラーが発生したプログラムまでの呼出経路から、プログラムの呼出し関係を知ることができます。  
この例では、プログラムSAMPDLL2は、モジュールSAMPDLL1.DLLのプログラムSAMPDLL1から呼び出され、SAMPDLL1はモジュールSAMPLE.EXEのプログラムSAMPLEから呼び出されていることがわかります。
5. エラーの詳細については、対話型デバッガを使用して調査します。エラーが発生したプログラムおよび必要ならば呼出経路上に出現するプログラムを別途作成し直して、デバッグします。

#### 19.6.4.3.2 実行時メッセージ

実行時メッセージの診断レポート例を以下に示します。

例で使用するプログラムは次のプログラムから構成されています。





#### <<要約>>

実行時メッセージが発生しました:

アプリケーション : D:\APL\SAMPLE.exe (PID=000000BB)  
 例外種別 : JMP03111-U [PID:00000CB4 TID:0000034C] 'SYS001' ファイルのファイル参照子の指定に誤りがあります。 'NOT ASSIGNED' PGM=SAMPDLL2. LINE=31

発生時間 : YYYY/MM/DD (HH:SS:MM)  
 発生モジュール : D:\APL\SAMPDLL2.dll  
 作成時間 : YYYY/MM/DD (HH:SS:MM)  
 ファイルサイズ : 66336bytes

#### <<問題箇所>>

スレッドID : 0000034C  
 レジスタ : EAX=0012E238 EBX=1C044B18 ECX=E9999999 EDX=00050001 ESI=00000000  
 EDI=00161EEC EIP=7C812A5B ESP=0012E234 EBP=0012E288 EFL=00000246  
 CS=001B SS=0023 DS=0023 ES=0023 FS=003B GS=0000  
 スタックコミット: 00005000 (トップ:00130000, ベース:0012B000)  
 命令 : アドレス +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +a +b +c +d +e +f  
 7C812A4B 8D 7D C4 F3 A5 5F 8D 45 B0 50 FF 15 08 15 80 7C  
 フォールト -> 7C812A5B 5E C9 C2 10 00 85 FF 0F 8E 36 93 FF FF 8B 55 FC

モジュールファイル : D:\APL\SAMPDLL2.dll  
 セクション相対位置 : .text+0000029A  
 EXPORT相対位置 : SAMPDLL2+00000262  
 シンボル相対位置 : SAMPDLL2+0000029A  
 翻訳情報 : ShiftJIS, シングルスレッド, NOOPTIMIZE  
 外部プログラム/クラス : SAMPDLL2  
 ソースファイル : SAMPDLL2.cob  
 文位置 : 31

#### <呼出経路>

[ 1]-----  
 モジュールファイル : C:\Program Files\NetCOBOL\F3BKATCH.dll  
 セクション相対位置 : .text+00002A1A  
 EXPORT相対位置 : NotifyExecErrorInfo+0000015E  
 [ 2]-----  
 モジュールファイル : C:\Program Files\NetCOBOL\F3B1PRCT.dll  
 セクション相対位置 : .text+0000BEF8  
 EXPORT相対位置 : JMP1MESS+00000B58  
 [ 3]-----

モジュールファイル : C:\Program Files\NetCOBOL\F3B1PRCT.dll  
セクション相対位置 : .text+000226C7  
EXPORT相対位置 : JMP510ER+00000E97

[ 4]-----  
モジュールファイル : C:\Program Files\NetCOBOL\F3B1IO.dll  
セクション相対位置 : .text+00002C71  
EXPORT相対位置 : JMP5VSEQ+00002C71

[ 5]-----  
モジュールファイル : C:\Program Files\NetCOBOL\F3B1IO.dll  
セクション相対位置 : .text+00000622  
EXPORT相対位置 : JMP5VSEQ+00000622

[ 6]-----  
モジュールファイル : C:\Program Files\NetCOBOL\F3B1IO.dll  
セクション相対位置 : .text+00000298  
EXPORT相対位置 : JMP5VSEQ+00000298

[ 7]-----  
モジュールファイル : D:\APL\SAMPDLL2.dll  
セクション相対位置 : .text+0000029E  
EXPORT相対位置 : SAMPDLL2+00000266  
シンボル相対位置 : SAMPDLL2+0000029E  
翻訳情報 : ShiftJIS, シングルスレッド, NOOPTIMIZE  
外部プログラム/クラス : SAMPDLL2  
ソースファイル : SAMPDLL2.cob  
文位置 : 31

[ 8]-----  
モジュールファイル : D:\APL\SAMPLE.exe  
セクション相対位置 : .text+00000823  
シンボル相対位置 : SAMPLE+00000823  
翻訳情報 : ShiftJIS, シングルスレッド, NOOPTIMIZE  
外部プログラム/クラス : SAMPLE  
ソースファイル : SAMPLE.cob  
文位置 : 33

[ 9]-----  
モジュールファイル : D:\APL\SAMPLE.exe  
セクション相対位置 : .text+0000000A  
シンボル相対位置 : SAMPLE+0000000A

[ 10]-----  
モジュールファイル : D:\APL\SAMPLE.exe  
セクション相対位置 : .text+00000B6A  
シンボル相対位置 : \_WinMainCRTStartup+000000CE

[ 11]-----  
モジュールファイル : C:\WINDOWS\system32\KERNEL32.dll  
セクション相対位置 : .text+00015FD7  
EXPORT相対位置 : RegisterWaitForInputIdle+00000049

#### <<システム情報>>

コンピュータ名 : MACHINE01  
ユーザー名 : user01  
Windowsバージョン : Microsoft Windows XP  
バージョンナンバー : 5.1.2600  
サービスパック : Service Pack 3

#### <<コマンドライン>>

"D:\APL\SAMPLE.exe"

#### <<環境変数>>

@WinCloseMsg=OFF  
@CnsIWinSize=(80, 24)  
@CBR\_ENTRYFILE=ENTRY.DAT  
COMPUTERNAME=MACHINE01  
ComSpec=C:\WINDOWS\system32\cmd.exe  
HOMEDRIVE=C:  
HOMEPATH=\users\default

```

LOGONSERVER=¥¥WORK01
LIB=C:¥Program Files¥NetCOBOL
NUMBER_OF_PROCESSORS=1
OS=Windows_NT
Path=C:¥Program Files¥NetCOBOL;C:¥WINDOWS¥system32;C:¥WINDOWS;
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 15 Model 2 Stepping 9, GenuineIntel
PROCESSOR_LEVEL=15
PROCESSOR_REVISION=0209
SystemDrive=C:
SystemRoot=C:¥WINDOWS
TEMP=C:¥TEMP
TMP=C:¥TEMP
USERDOMAIN=WIN-DOMAIN
USERNAME=user01
USERPROFILE=C:¥Documents and Settings¥user01
windir=C:¥WINDOWS

```

#### <<実行環境情報>>

```

ランタイムシステム : V10.2.0 日本語製品版
実行モード : ShiftJIS, シングルスレッド
プログラム名 : SAMPLE
初期化ファイル : D:¥APL¥COBOL85.CBR
 @WinCloseMsg=OFF
 @CnsWinSize=(80, 24)
 [EOF]

```

```

@CBR_ENTRYFILE : D:¥APL¥ENTRY.DAT
 [ENTRY]
 SAMPDLL1=SAMPDLL1.DLL
 SAMPDLL2=SAMPDLL2.DLL
 SAMPDLL3=SAMPDLL3.DLL
 [EOF]

```

以降、省略

1. <<要約>>に着目します。ここでは、どのモジュールで、どんな実行時メッセージが出力されたのか知ることができます。  
この例では、D:¥APL¥SAMPDLL2.DLLで、JMP0311I-Uの実行時メッセージが出力されたことがわかります。
2. <<問題箇所>>に着目します。ここでは、どの翻訳単位でエラーが発生したのか知ることができます。  
この例では、プログラムSAMPDLL2の31行目でエラーが発生したことがわかります。
3. エラーが発生したプログラムのソースファイル、または目的プログラムリストから、どの文(命令)でエラーが発生したのかを特定します。  
この例では、31行目のOPEN文でエラーが発生したと仮定します。  
実行用の初期化ファイルの環境変数情報は、環境変数に反映されます。したがって、ファイル識別名、論理宛先定義ファイルおよび印刷情報ファイルなどは<<環境変数>>で確認できます。エントリ情報は環境変数には反映されませんので、<<実行環境情報>>に出力されている実行用の初期化ファイルやエントリ情報ファイルの内容を確認する必要があります。  
この例の場合、SYSINに対してファイルが割り当てられていないというエラーなので、環境変数の設定状況を確認します。
4. 必要であれば、対話型デバッガを使用してエラーの詳細を調査します。



#### 注意

実行時メッセージの<<問題箇所>>で出力される<呼出経路>は、アプリケーションエラーの場合と異なります。アプリケーションエラーの場合は、呼出経路はエラーが発生したプログラムを直接呼び出しているプログラムから始まります。一方、実行時メッセージの場合には、ランタイムシステムのプログラムから始まります。呼出経路上にエラーが発生した位置と同じ文位置を持つプログラム(エラー発生位置の文位置が出力されていない場合にはセクション相対オフセット+4の値を持つプログラム)が出現します。それ以降に出力される情報が、エラーが発生したプログラムまでの呼出し関係を表していることになります。

例では、エラーが発生したプログラムがSAMPDLL2、そのプログラムまでの呼出し関係は<呼出経路>の7番目以降の情報が表現しています。

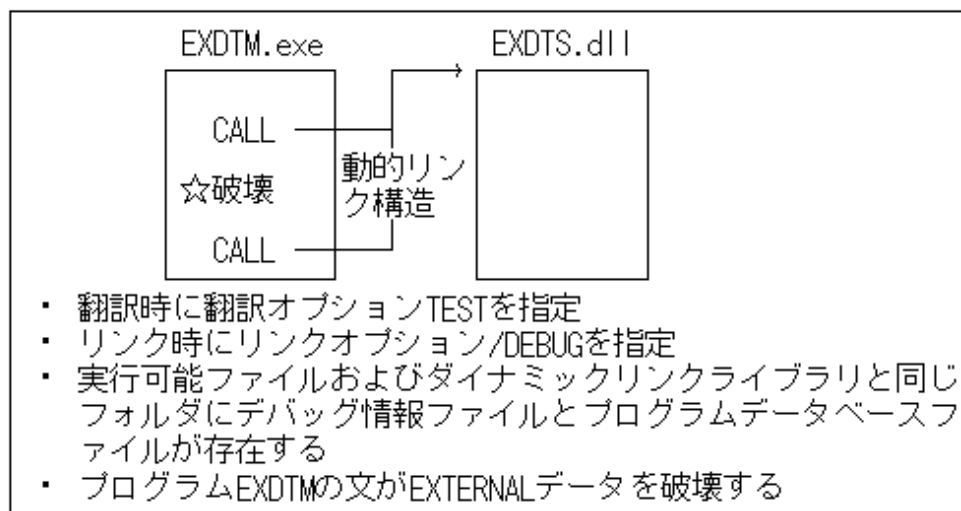
#### 19.6.4.3.3 応答なし

1. <<スレッド情報>>に着目します。ここには、アプリケーションのすべてのスレッドに関する情報が出力されています。
2. スレッドごとに、どの翻訳単位で中断している状態なのかわかります。
3. 次に、プログラムのソースファイルまたは目的プログラムリストから、どの文(命令)で中断しているのかを特定します。
4. ソースファイルまたは目的プログラムリストから、どこで応答しなくなったのか見当をつけます。
5. 詳細については、対話型デバッガを使用して調査します。

#### 19.6.4.3.4 領域破壊監視

領域破壊を監視した場合の診断レポート例を以下に出力します。

例で使用するプログラムは次のプログラムから構成されています。



#### NetCOBOL 診断機能

##### <<要約>>

領域の書き込みを監視しました：

アプリケーション：D:\APL\EXDTM.EXE (PID=000000BE)  
起動時間：YYYY/MM/DD (HH:MM:SS)  
監視アドレス：00DC1198  
監視サイズ：4bytes

##### <<1>>=====

監視領域：+0 +1 +2 +3  
B8 00 DC 00  
スレッドID：00000070  
モジュールファイル：C:\WINDOWS\System32\ntdll.dll  
セクション相対位置：.text+00003A44  
EXPORT相対位置：RtlAllocateHeap+000003C2

##### <呼出経路>

##### [ 1]-----

モジュールファイル：C:\Program Files\NetCOBOL\F3BIPRCT.dll  
セクション相対位置：.text+0002CB05  
EXPORT相対位置：JMP1TMEM+00000295

##### [ 2]-----

モジュールファイル：C:\Program Files\NetCOBOL\F3BIPRCT.dll  
セクション相対位置：.text+0002298D

```

EXPORT相対位置 : JMP1EXDT+000005AD
[3]-----
モジュールファイル : C:\Program Files\NetCOBOL\F3BIPRCT.dll
セクション相対位置 : .text+0002265A
EXPORT相対位置 : JMP1EXDT+0000027A
[4]-----
モジュールファイル : D:\APL\EXDTM.EXE
セクション相対位置 : .text+00000162
シンボル相対位置 : EXDTM+00000162
翻訳情報 : ShiftJIS, シングルスレッド, NOOPTIMIZE
外部プログラム/クラス : EXDTM
ソースファイル : Exdtm.cob
文位置 : IN ENTRY-CODE
[5]-----
モジュールファイル : D:\APL\EXDTM.EXE
セクション相対位置 : .text+0000000A
シンボル相対位置 : EXDTM+0000000A
[6]-----
モジュールファイル : D:\APL\EXDTM.EXE
セクション相対位置 : .text+0000055C
シンボル相対位置 : _WinMainCRTStartup+000000CE
[7]-----
モジュールファイル : C:\WINDOWS\System32\KERNEL32.dll
セクション相対位置 : .text+0001AD15
EXPORT相対位置 : GetProcessPriorityBoost+00000117

<<2>>=====
監視領域 : +0 +1 +2 +3
 00 00 00 00
スレッドID : 00000070
モジュールファイル : C:\WINDOWS\System32\ntdll.dll
セクション相対位置 : .text+00003C85
EXPORT相対位置 : RtlAllocateHeap+00000603

<呼出経路>
[1]-----
モジュールファイル : C:\Program Files\NetCOBOL\F3BIPRCT.dll
セクション相対位置 : .text+0002CB05
EXPORT相対位置 : JMP1TMEM+00000295
[2]-----
モジュールファイル : C:\Program Files\NetCOBOL\F3BIPRCT.dll
セクション相対位置 : .text+0002298D
EXPORT相対位置 : JMP1EXDT+000005AD
[3]-----
モジュールファイル : C:\Program Files\NetCOBOL\F3BIPRCT.dll
セクション相対位置 : .text+0002265A
EXPORT相対位置 : JMP1EXDT+0000027A
[4]-----
モジュールファイル : D:\APL\EXDTS.DLL
セクション相対位置 : .text+00000152
EXPORT相対位置 : EXDTS+0000011E
シンボル相対位置 : EXDTS+00000152
翻訳情報 : ShiftJIS, シングルスレッド, NOOPTIMIZE
外部プログラム/クラス : EXDTS
ソースファイル : Exdts.cob
文位置 : IN ENTRY-CODE
[5]-----
モジュールファイル : D:\APL\EXDTM.EXE
セクション相対位置 : .text+00000238
シンボル相対位置 : EXDTM+00000238
翻訳情報 : ShiftJIS, シングルスレッド, NOOPTIMIZE
外部プログラム/クラス : EXDTM
ソースファイル : Exdtm.cob

```

文位置 : 16

```
[6]-----
モジュールファイル : D:\APL\EXDTM.EXE
セクション相対位置 : .text+0000000A
シンボル相対位置 : EXDTM+0000000A
[7]-----
モジュールファイル : D:\APL\EXDTM.EXE
セクション相対位置 : .text+0000055C
シンボル相対位置 : _WinMainCRTStartup+000000CE
[8]-----
モジュールファイル : C:\WINDOWS\System32\KERNEL32.dll
セクション相対位置 : .text+0001AD15
EXPORT相対位置 : IsProcessorFeaturePresent+00000117
```

```
<<3>>=====
監視領域 : +0 +1 +2 +3
 41 41 41 41
スレッドID : 00000070
モジュールファイル : D:\APL\EXDTM.EXE
セクション相対位置 : .text+0000034E
シンボル相対位置 : EXDTM+0000034E
翻訳情報 : ShiftJIS, シングルスレッド, NOOPTIMIZE
外部プログラム/クラス : EXDTM
ソースファイル : Exdtm.cob
文位置 : 19
```

<呼出経路>

```
[1]-----
モジュールファイル : D:\APL\EXDTM.EXE
セクション相対位置 : .text+0000000A
シンボル相対位置 : EXDTM+0000000A
[2]-----
モジュールファイル : D:\APL\EXDTM.EXE
セクション相対位置 : .text+0000055C
シンボル相対位置 : _WinMainCRTStartup+000000CE
[3]-----
モジュールファイル : C:\WINDOWS\System32\KERNEL32.dll
セクション相対位置 : .text+0001AD15
EXPORT相対位置 : IsProcessorFeaturePresent+00000117
```

```
<<4>>=====
実行時メッセージが発生しました :
 例外種別 : JMP0071I-U [PID:000000B2 TID:00000085] 領域破壊を検出しました。
START PGM=EXDTS BRKADR=0x00DC1198.
スレッドID : 00000070
```

以降、省略

- 書き込み情報(<<カウンタ>>)を先頭から順に調べていきます。  
書き込み情報から、監視領域に書き込まれた内容と書き込みを行った位置を知ることができます。  
診断機能は、ランタイムシステムが直接書き込んだ場合を除くすべての書き込みに対して書き込み情報を出力します。そのため、ランタイムシステムから呼び出されたシステム関数による書き込みに対しても書き込み情報が出力されます。このような場合は、<呼出経路>からシステム関数の呼出し元を調べ、ランタイムシステムからの呼出しであるのか、利用者プログラムからの呼出しであるのかを判断する必要があります。  
この例では、書き込み情報<<1>>および<<2>>は、ランタイムシステムから呼び出されたシステム関数による書き込みであることがわかります。
- 不当な書き込みかどうかは、実行時メッセージJMP0071I-Uが発生した場合に出力された診断レポートの<<破壊領域>>の内容と同じ内容の書き込みを行ったかどうかで判断します。  
この例では、実行時メッセージJMP0071I-Uが発生した場合の診断レポート(“19.5.3 診断レポート”)を調べていくと、3番目の書き込み情報が示す文が領域を破壊していることがわかります。

通常、実行時メッセージJMP0071I-Uのエラー情報の直前に出力されている書き込み情報が領域を破壊したプログラム、命令を示しています。

## 19.6.5 ダンプ

### 19.6.5.1 ダンプとは

ダンプとは、エラー発生時のメモリの内容が記録されたもので、エラーの原因調査を行う際の有効な資料となる情報です。

診断機能は、特定のアプリケーションエラーまたはUレベルの実行時メッセージが発生したときのメモリの内容をダンプファイルに出力します。

ダンプの出力を抑止する場合には、起動パラメタ“-d NO”を指定します。

#### ポイント

ダンプがないと、エラーの原因を特定するまでに時間を要し、問題が長期化する場合があります。ダンプの出力を有効にしておくことをおすすめします。

ダンプは、WinDbg(Microsoft社が提供しているデバッグツール)などのデバッグに読み込ませることにより、デバッグに使用することができます。WinDbgの使用方法については、WinDbgのヘルプを参照してください。

### 19.6.5.2 ダンプの出力先

ダンプファイル名は、アプリケーションの名前を「アプリケーション名\_プロセスID\_エラー発生時間\_COBSNAP」に変更し、拡張子を「DMP」に置き換えた名前になります。

|           |                                            |
|-----------|--------------------------------------------|
| アプリケーション名 | : TESTPGM. EXE                             |
| プロセスID    | : 2D8                                      |
| エラー発生時間   | : 2009年07月10日15時04分26秒                     |
| ダンプファイル名  | : TESTPGM_2D8_20090710-150426_COBSNAP. DMP |

ダンプの標準の出力先は、ユーザ共通のアプリケーションデータフォルダ(コンピュータごとにアプリケーション固有のデータを格納するフォルダ)配下の¥Fujitsu¥NetCOBOL¥COBSNAPです。

通常、ダンプの出力先は、以下の名前で作成されます。

- Windows XPおよびWindows Server 2003の場合

|                                                                               |
|-------------------------------------------------------------------------------|
| C:¥Documents and Settings¥All Users¥Application Data¥Fujitsu¥NetCOBOL¥COBSNAP |
|-------------------------------------------------------------------------------|

- Windows Vista、Windows 7、Windows 8、Windows Server 2008、およびWindows Server 2012の場合

|                                         |
|-----------------------------------------|
| C:¥ProgramData¥Fujitsu¥NetCOBOL¥COBSNAP |
|-----------------------------------------|

出力先フォルダを変更する場合は、起動パラメタ“-o”で指定します。起動パラメタの詳細については、“[表19.4 起動パラメタ](#)”を参照してください。

#### 注意

- ユーザ共通のアプリケーションデータフォルダは、通常、読み取り専用のフォルダです。しかし、ダンプの標準出力先フォルダは、Everyoneグループに対して以下のアクセス権を持つフォルダとして生成されます。
  - 変更
  - 読み取りと実行
  - フォルダの内容の一覧表示
  - 読み取り
  - 書き込み

- ・ ダンプの標準の出力先フォルダは、コンピュータにアクセスできるユーザならば誰でも参照できます。アプリケーションの仕様やその運用条件にも依存しますが、ダンプには、個人情報や秘密データが含まれる可能性があります。運用にあたっては、セキュリティやデータ保護の観点からダンプの出力先フォルダの妥当性を十分検討してください。必要であれば起動パラメタ“-o”を指定し、ダンプの出力先フォルダを変更してください。また、ダンプ出力が不要な場合には、起動パラメタ“-d NO”を指定し、ダンプ出力を抑制してください。

起動パラメタの詳細については“表19.4 起動パラメタ”を参照してください。

- ・ ダンプファイルは、ひとつひとつのファイルサイズが非常に大きくなります。そのため、ダンプの出力先には十分な容量を確保してあるドライブのフォルダを指定してください。

### 19.6.5.3 ダンプファイル数の管理

診断機能が作成するダンプファイル数の上限は、フォルダごとに10個です。

ダンプファイルを作成するときにすでにフォルダ内に10個のファイルが存在する場合、診断機能は日付の最も古いファイルを削除します。

### 19.6.6 注意事項

ここでは、診断機能使用時の注意事項について説明します。

- ・ アプリケーションエラーが発生した場合に自動的に起動する機能には、診断機能のほかに以下のものがあります。
  - Visual C++のジャストインタイムデバッガ
  - ワトソン博士(Windows XPおよびWindows Server 2003の場合)
  - Windows エラー報告(Windows Vista、Windows 7、Windows 8、Windows Server 2008、およびWindows Server 2012の場合)アプリケーションエラーの発生時にこれらの機能が起動するようシステムに登録されていても、診断機能を起動するように指定された場合には、これらの機能に優先して起動します。
- ・ 診断機能は、アプリケーションを対話型デバッガ、PowerCOBOLデバッガまたはVisual C++デバッガなどでデバッグしている場合やSystemwalkerの下で実行している場合には、使用できません。
- ・ 診断機能は、アプリケーション自身がアプリケーションエラーを構造化例外処理の例外ハンドラによって処理している場合には、起動されません。たとえば、例外ハンドラを持つCOBOL以外の言語のプログラムと言語間結合している場合や例外ハンドラを持つWebサーバなどのサーバプログラムから呼び出された場合が該当します。これらのアプリケーションで発生するアプリケーションエラーの調査には、対話型デバッガを使用してください。
- ・ 診断機能は、スタックオーバーフロー例外(0xC00000FD)が発生した場合にはシステムから正常に起動されません。エラーの診断情報を取得したい場合には、以下の機能を利用してください。
  - ワトソン博士(Windows XPおよびWindows Server 2003の場合)
  - Windows エラー報告(Windows Vista、Windows 7、Windows 8、Windows Server 2008、およびWindows Server 2012の場合)
- ・ 翻訳オプションOPTIMIZEを指定して翻訳したプログラムは、コンパイラによって文が移動されたり、削除されたりする場合があります。
- ・ 診断レポートに以下のエラーメッセージが出力された場合は、ソースファイルの記述方法に問題があります。

デバッグ情報ファイル '\$1' 内のプログラム '\$2' の情報に誤りがあるため、このプログラムに対する言語イメージの情報を出力できませんでした。

\$1: デバッグ情報ファイル名

\$2: プログラム名

問題を解決するには以下を確認して、ソースファイルを変更してください。問題が解決しない場合は技術員(SE)に連絡してください。

- COBOLプログラムの場合

“20.7 注意事項”のソースファイルの記述方法に関する項目



#### ー PowerCOBOLプログラムの場合

“PowerCOBOLユーザーズガイド”の登録集ファイルの利用方法についての注意事項

- スタックフレームを作成しない他言語プログラムやシステム関数がプログラムの呼出経路上にいる場合には、それら呼び出しているプログラムの情報が診断レポートの“呼出経路”および“スタックサマリ”に出力されません。たとえば、Visual C++の場合には、翻訳オプション/Ox、/O1、/O2または/Oyを指定して翻訳したプログラムが該当します。
- スタックが破壊されている場合には、診断レポートに破壊される直前までの呼出経路情報を出力するとともに、スタックが破壊されていることを示す情報も出力します。
- 診断機能は、応答しなくなったアプリケーションの診断レポートを出力後、プロセスを強制終了させます。強制終了後にシステムの動作が不安定になる可能性がありますので、正常に動作しているプロセスやWindowsシステムに影響を与えるプロセスに対しては、診断機能を起動しないでください。
- アプリケーションエラーや実行時メッセージに対する診断で、診断機能は問題が発生するまで動作しないため、問題が発生しなければアプリケーションの実行速度は変わりません。ただし、領域破壊の監視を行う場合には、診断機能配下でアプリケーションが実行されるため、アプリケーションの実行速度は2～10倍程度遅くなります。

## 19.7 翻訳リストを使ったデバッグ

---

ここでは、以下の翻訳リストを使用して、異常終了時のデバッグを行う方法について説明します。

### 目的プログラムリスト

障害発生箇所の特定に使用します。

### データマップリスト、プログラム制御情報リスト

異常終了時にデータの内容を参照するために使用します。

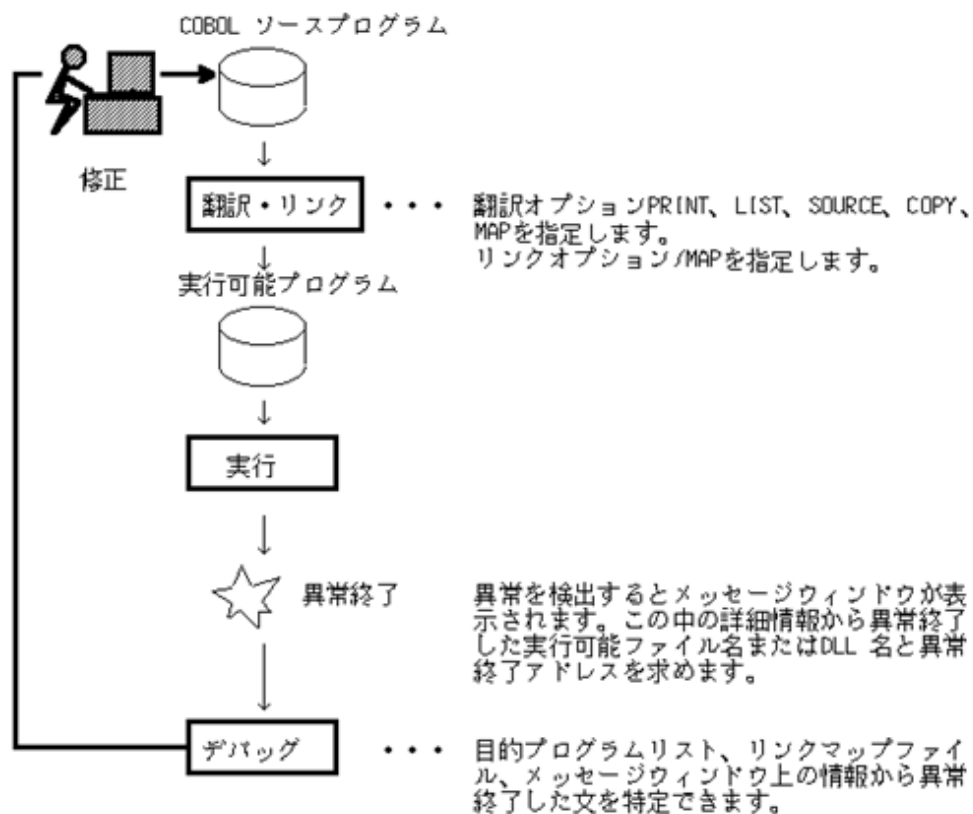
デバッグを行うには、これらの翻訳リストと合わせて以下のファイルが必要です。

- リンクマップファイル
- ワトソン博士で採取したログ情報およびクラッシュダンプ
- Windowsエラー報告で採取したクラッシュダンプ

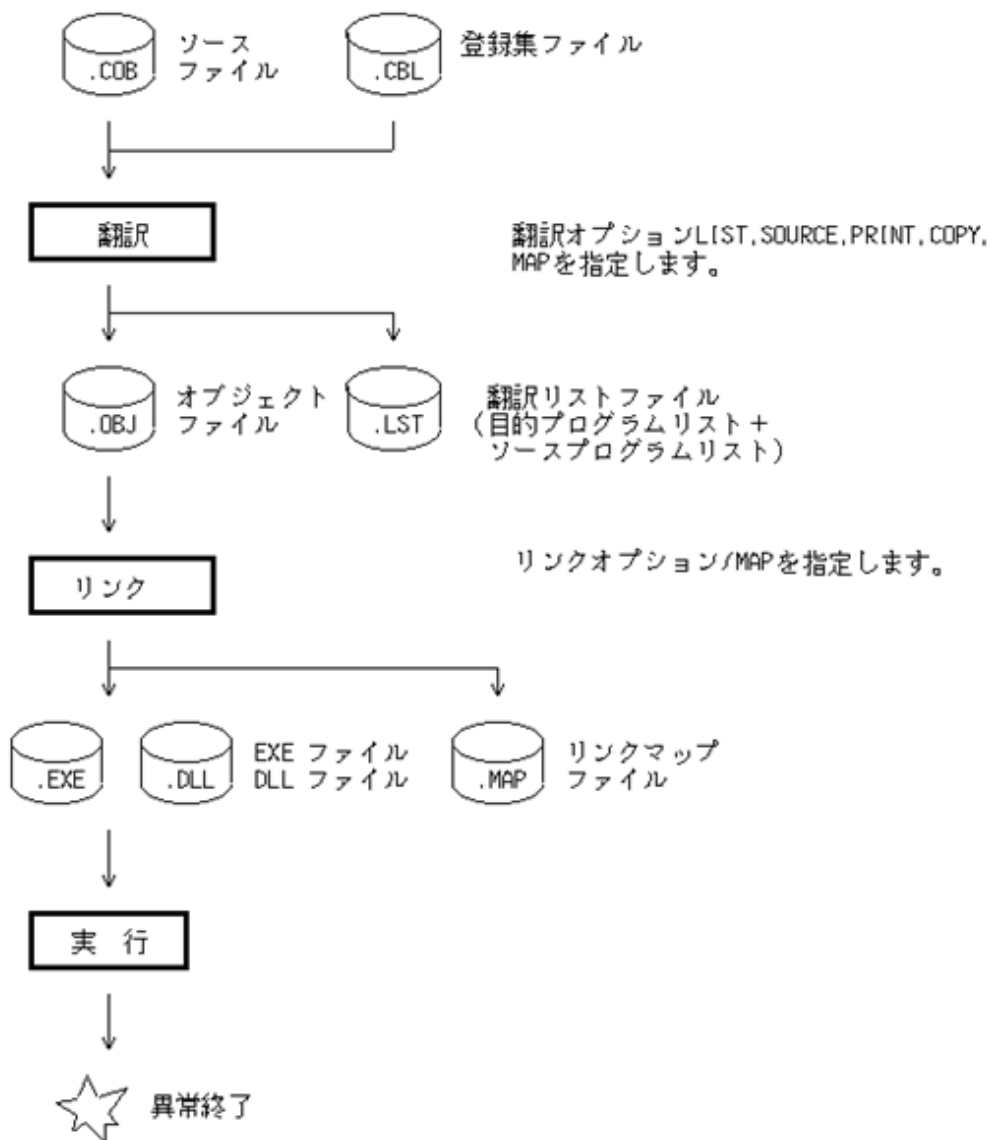
### 19.7.1 デバッグ作業の流れ

---

以下に目的プログラムリストを使った障害発生箇所の特定方法の流れを示します。



以下にファイル関連図を示します。



## 注意

- この機能は、診断機能が有効になっている場合は使用できません。診断機能を無効にするには、プログラムを実行する前に、環境変数情報 @CBR\_JUSTINTIME\_DEBUG に NO を指定してください。また、ワトソンログを採取するには、ワトソン博士 (Windows に添付されている DRWTSN32.EXE または DRWATSON.EXE) を起動してください。ワトソン博士の使用方法については、ワトソン博士のヘルプを参照してください。
- 異常終了時に参照することができるデータは、異常終了したプログラムまたはメソッドで使用するデータのみです。他のプログラムおよびメソッドのデータは参照することができません。また、翻訳オプション OPTIMIZE を指定している場合は、最適化処理により、実行結果が領域に格納されない場合や文が移動・削除される場合があるため、領域を正しく参照することができないことがあります。

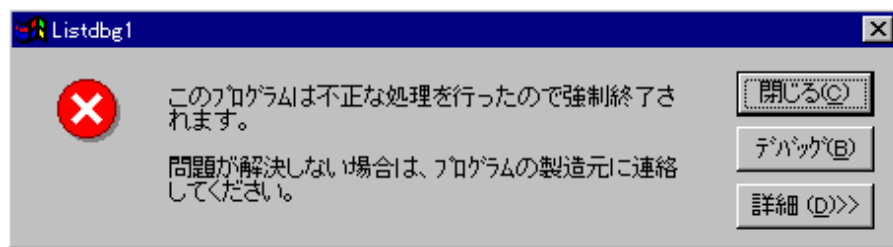
## 19.7.2 障害発生箇所の特定方法

実行時に異常終了したときの障害発生箇所の特定方法について、簡単な例を用いて説明します。

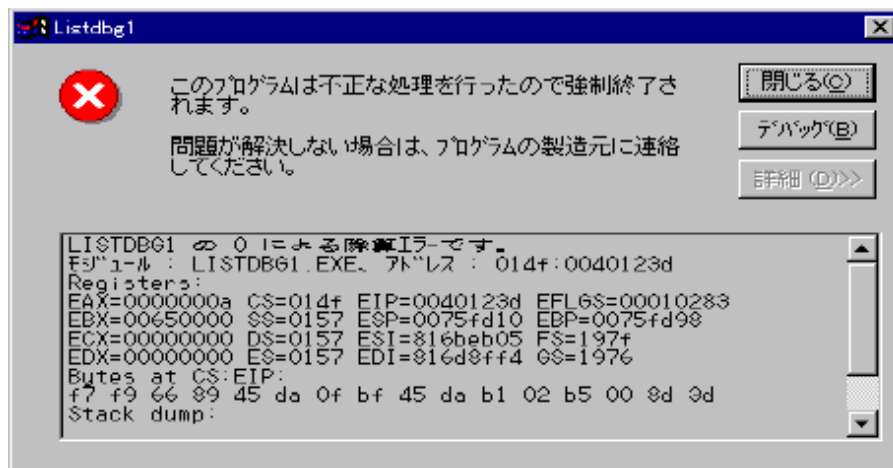
## 19.7.2.1 障害発生箇所の特定方法(EXEファイルの場合)

### 1. 異常終了情報の取得

実行時に異常終了した場合、以下のウィンドウが表示されます。



ウィンドウの詳細ボタンをクリックすると以下の情報が表示されます。



ウィンドウ内の内容から以下の情報を調べます。

- 例外発生理由(上図では、“LISTDBG1の0による除算エラーです。”) (Exception reason (in the figure above, “Division error by 0 in LISTDBG1.”))
- 異常終了したモジュール(上図では、“モジュール:LISTDBG1.EXE”) (Module that terminated abnormally (in the figure above, “Module: LISTDBG1.EXE.”))
- 異常終了したアドレス(上図では、“アドレス:014f:0040123d”) (Address that terminated abnormally (in the figure above, “Address: 014f:0040123d.”))

### 参照

これらの情報はワトソン博士(Windowsに添付されているDRWTSN32.EXEまたはDRWATSON.EXE)を起動することにより採取することができます。ワトソン博士の使用方法については、ワトソン博士のヘルプを参照してください。

### 2. モジュール開始アドレスの取得

ここでは、1.で調べた異常終了アドレスとリンクマップファイルから、どのモジュール内で異常終了したかを調べる方法について記述します。

| LISTDBG1                                         |           |           |       |  |
|--------------------------------------------------|-----------|-----------|-------|--|
| Timestamp is 35170898 (Tue Mar 24 10:12:56 1998) |           |           |       |  |
| Preferred load address is 00400000               |           |           |       |  |
| Start                                            | Length    | Name      | Class |  |
| 0001:00000000                                    | 000023b8H | .text     | CODE  |  |
| 0002:00000000                                    | 00000337H | .rdata    | DATA  |  |
| 0002:00000337                                    | 00000000H | .edata    | DATA  |  |
| 0003:00000000                                    | 00000004H | .CRT\$XCA | DATA  |  |
| 0003:00000004                                    | 00000004H | .CRT\$XCZ | DATA  |  |

```

0003:00000008 00000004H .CRT$XIA DATA
0003:0000000c 00000004H .CRT$XIZ DATA
0003:00000010 00000004H .CRT$XPA DATA
0003:00000014 00000004H .CRT$XPZ DATA
0003:00000018 00000004H .CRT$XTA DATA
0003:0000001c 00000004H .CRT$XTZ DATA
0003:00000020 00000f44H .data DATA
0003:00000f68 0000022cH .bss DATA
0004:00000000 00000028H .idata$2 DATA
0004:00000028 00000014H .idata$3 DATA
0004:0000003c 00000008cH .idata$4 DATA
0004:000000c8 00000008cH .idata$5 DATA
0004:00000154 00000224H .idata$6 DATA
0005:00000000 00000220H .rodata DATA
0006:00000000 000000e4H .rsrc$01 DATA
0006:000000f0 00000d74H .rsrc$02 DATA

```

| Address       | Publics by Value<br>[1] | Rva+Base<br>[2] | Lib:Object              |
|---------------|-------------------------|-----------------|-------------------------|
| 0001:00000000 | _WinMain@16             | 00401000        | f LISTDBG1.OBJ          |
| 0001:00000260 | _JMP1PLAN               | 00401260        | f F3BICIMP:F3BIPRCT.dll |
| 0001:00000266 | _JMP1TJOB               | 00401266        | f F3BICIMP:F3BIPRCT.dll |
| 0001:0000026c | _JMP1IORF               | 0040126c        | f F3BICIMP:F3BIPRCT.dll |
| 0001:0000046e | JMPZNCVZB _MEM          | 0040146e        | F3BICIMP:ncvzb.obj      |
| 0001:00000542 | JMPZNCVZB _REG1         | 00401542        | F3BICIMP:ncvzb.obj      |
| 0001:00000612 | JMPZNCVZB _REG2         | 00401612        | F3BICIMP:ncvzb.obj      |
| 0001:00000690 | WinMainCRTStartup       | 00401690        | f LIBC:wincrt0.obj      |

リンクマップファイル内に表示されているシンボル名(上図の[1])の絶対アドレス(上図の[2])と異常終了したときに表示されるウィンドウから調べた異常終了アドレスの比較を行い、異常終了したCOBOLソースのモジュール開始アドレスを調べます。この例の場合、上図のシンボル名“\_WinMain@16”が指すプログラム“LISTDBG1.OBJ”で異常終了していることがわかります。

## 参考

リンクマップファイル内には、以下のようなシンボル名が出力されます。

|                                  |                                                     |
|----------------------------------|-----------------------------------------------------|
| 翻訳オプションMAIN(WINMAIN)指定時のプログラム入口  | WinMain                                             |
| 翻訳オプションMAIN(MAIN)指定時のプログラム入口     | main                                                |
| 翻訳オプションNOMAIN指定時のプログラム入口         | プログラム名                                              |
| 翻訳オプションNODLOAD指定時のメソッド入口         | クラス名_ENTRY_メソッド名                                    |
| 翻訳オプションDLOAD指定時のメソッド入口           | クラス名_メソッド名                                          |
| 翻訳オプションNODLOAD指定時のPROPERTYメソッド入口 | クラス名_ENTRY_GET_メソッド名<br>または<br>クラス名_ENTRY_SET_メソッド名 |
| 翻訳オプションDLOAD指定時のPROPERTYメソッド入口   | クラス名_GET_メソッド名<br>または<br>クラス名_SET_メソッド名             |

### 3. 異常終了アドレスのオブジェクト内相対オフセットの求め方

異常終了アドレスのオブジェクト内相対オフセットは、以下の式で求めることができます。

ー 翻訳オプションMAIN指定のCOBOLソースの場合

|                                                  |
|--------------------------------------------------|
| 異常終了アドレスのオブジェクト内相対オフセット = 異常終了アドレス - モジュール開始アドレス |
|--------------------------------------------------|

ー 翻訳オプションNOMAIN指定のCOBOLソースの場合、またはオブジェクト指向プログラムのCOBOLソースの場合

$$\begin{aligned} \text{異常終了アドレスのオブジェクト内相対オフセット} &= \\ &(\text{異常終了アドレス} - \text{モジュール開始アドレス}) + \\ &\text{目的プログラムリスト内に出力されているシンボル名が示すオブジェクト内相対オフセット} \end{aligned}$$

この例の場合、異常終了アドレスは、1.で異常終了した場合に表示されるウィンドウ内に表示される異常終了アドレス“0040123d”です。

モジュール開始アドレスは、2.で調査した異常終了しているモジュール“LISTDBG1.OBJ”の開始アドレスである“00401000”です。異常終了しているモジュール“LISTDBG1.OBJ”のシンボル名には、“\_WinMain”が表示されていることから、このモジュールには、翻訳オプションMAINが指定されていたことがわかります。したがって、オブジェクト内相対オフセットを求める式は、翻訳オプションMAIN指定のCOBOLソースの場合の式を使って求めます。計算式は、以下のとおりです。

|                |          |
|----------------|----------|
| 異常終了アドレス       | 0040123d |
| ー モジュール開始アドレス  | 00401000 |
| -----          |          |
| オブジェクト内相対オフセット | 0000023d |

#### 4. 異常終了したCOBOLの文の特定

異常終了アドレスのオブジェクト内相対オフセットを目的プログラムリスト内のオブジェクト内相対オフセットから探し、異常終了したアセンブラ命令を探します。

異常終了したアセンブラ命令の位置から目的プログラムリスト内に出力されているCOBOLソースの記述行とCOBOLの文を探します。

|                         |         |                             |
|-------------------------|---------|-----------------------------|
| --- 13 ---              | MOVE    |                             |
| 000001B8 66A112000000   | mov     | ax, %COA+00000012 : +00     |
| 000001BE 66A3D4000000   | mov     | %GWA+000000D4, ax : 除数      |
| 000001C4 A0D5000000     | mov     | al, %GWA+000000D5 : 除数+1    |
| 000001C9 240F           | and     | al, 0F                      |
| 000001CB 0C30           | or      | al, 30                      |
| 000001CD A2D5000000     | mov     | %GWA+000000D5, al : 除数+1    |
| --- 15 ---              | COMPUTE |                             |
| 000001D2 8A15D2000000   | mov     | dl, %GWA+000000D2 : 被除数     |
| 000001D8 83E20F         | and     | edx, 0F                     |
| 000001DB 8A0DD3000000   | mov     | cl, %GWA+000000D3 : 被除数+1   |
| 000001E1 83E10F         | and     | ecx, 0F                     |
| 000001E4 8D1492         | lea     | edx, ds:edx+edx*4           |
| 000001E7 8D1451         | lea     | edx, ds:ecx+edx*2           |
| 000001EA 8A0DD3000000   | mov     | cl, %GWA+000000D3 : 被除数+1   |
| 000001F0 80E1F0         | and     | cl, F0                      |
| 000001F3 80F950         | cmp     | cl, 50                      |
| 000001F6 75XX           | jne     | GLB. 4                      |
| 000001F8 F7DA           | neg     | edx                         |
| 000001FA                | GLB. 4  |                             |
| 000001F7 02             |         |                             |
| 000001FA A0D4000000     | mov     | al, %GWA+000000D4 : 除数      |
| 000001FF 83E00F         | and     | eax, 0F                     |
| 00000202 8A0DD5000000   | mov     | cl, %GWA+000000D5 : 除数+1    |
| 00000208 83E10F         | and     | ecx, 0F                     |
| 0000020B 8D0480         | lea     | eax, ds:eax+eax*4           |
| 0000020E 8D0441         | lea     | eax, ds:ecx+eax*2           |
| 00000211 8A0DD5000000   | mov     | cl, %GWA+000000D5 : 除数+1    |
| 00000217 80E1F0         | and     | cl, F0                      |
| 0000021A 80F950         | cmp     | cl, 50                      |
| 0000021D 75XX           | jne     | GLB. 5                      |
| 0000021F F7D8           | neg     | eax                         |
| 00000221                | GLB. 5  |                             |
| 0000021E 02             |         |                             |
| 00000221 66A30A000000   | mov     | %GWA+0000000A, ax : TRLP+0  |
| 00000227 66891512000000 | mov     | %GWA+00000012, dx : TRLP+0  |
| 0000022E 0FBF0512000000 | movsx   | eax, %GWA+00000012 : TRLP+0 |

|                         |                                  |
|-------------------------|----------------------------------|
| 00000235 99             | cdq                              |
| 00000236 0FBF0D0A000000 | movsx ecx,%GWA+0000000A : TRLP+0 |
| 0000023D F7F9           | idiv ecx                         |
| 0000023F 668945DA       | mov ss:ebp-26,ax : TRLS+0        |
| 00000243 0FBF45DA       | movsx eax,ss:ebp-26 : TRLS+0     |
| 00000247 B102           | mov cl,02                        |
| 00000249 B500           | mov ch,00                        |
| 0000024B 8D3DD0000000   | lea edi,%GWA+000000D0 : 答え       |
| 00000251 E8XXXXXXX      | call JMPZNCVZB _REG1             |

目的プログラム内に記述されている記述行とCOBOLの文をソースプログラムリスト上から探します。このCOBOLの文が異常終了していることがわかります。

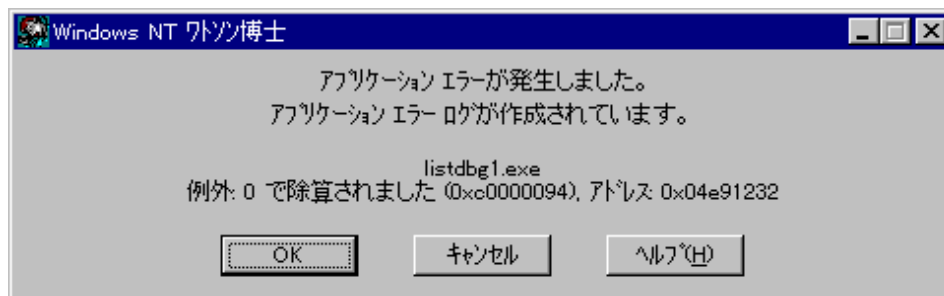
| 行番号 | 一連番号    | A                        | B         |
|-----|---------|--------------------------|-----------|
| 1   | 000100  | IDENTIFICATION DIVISION. |           |
| 2   | 000200  | PROGRAM-ID. LISTDBG1.    |           |
| 3   | 000300* |                          |           |
| 4   | 000400  | DATA DIVISION.           |           |
| 5   | 000500  | WORKING-STORAGE SECTION. |           |
| 6   | 000600  | 77 答え                    | PIC 9(2). |
| 7   | 000700  | 77 除数                    | PIC 9(2). |
| 8   | 000800  | 77 被除数                   | PIC 9(2). |
| 9   | 000900* |                          |           |
| 10  | 001000  | PROCEDURE DIVISION.      |           |
| 11  | 001100* |                          |           |
| 12  | 001200  | MOVE 10 TO 被除数.          |           |
| 13  | 001300  | MOVE 0 TO 除数.            |           |
| 14  | 001400* |                          |           |
| 15  | 001500  | COMPUTE 答え = 被除数 / 除数.   |           |
| 16  | 001600* |                          |           |
| 17  | 001700  | EXIT PROGRAM.            |           |
| 18  | 001800  | END PROGRAM LISTDBG1.    |           |

例の場合、3.で求めたモジュール内相対オフセット“0000023d”を目的プログラムリストから参照すると、異常終了したアセンブラ命令が“idiv ecx”であることがわかります。その異常終了したアセンブラ命令の位置から、COBOLソースの記述行が“---15---”行目で、COBOLの文が“COMPUTE”であることがわかります。

## 19.7.2.2 障害発生箇所の特定制法(DLLファイルの場合)

### 1. 異常終了情報の取得

実行時に異常終了した場合、ワトソン博士が起動されていれば、以下のウィンドウが表示されます。



ワトソンログ(DRWTSN32.LOG)から、以下の情報を調べます。

- 例外発生理由(例外番号の後に出力される()内の内容)

例：例外番号：C0000094 (0 で除算されました)

- 異常終了したアプリケーション(“アプリケーション:”以降の内容)

例：アプリケーション：D:\LISTDBG1.EXE

- ー 異常終了したアドレス(“フォールト>”が指すアドレス)

例：フォールト>04e91232 f7f9          idiv      ecx

- ー 異常終了したDLLファイルの実行時開始アドレス(異常終了したアドレスとワトソンログ内に表示されている実行時の開始/終了アドレスを比較して調べます。)

例：(04e90000 - 04e99000) E:\LISTDBGS.DLL

## 2. 異常終了アドレスの変換

DLLファイルの場合、実行時とリンク時の実行時開始アドレスが異なる場合があります。そのため、1.で調べた異常終了アドレスと異常終了したDLLファイルの実行時開始アドレス、リンクマップファイル内に出力されているリンク時ロードアドレス(Preferred load address)から、異常終了アドレスの変換を以下の式によって行います。

異常終了アドレス =  
(実行時異常終了アドレス - 異常終了したDLLファイルの実行時開始アドレス)  
+ リンク時ロードアドレス (Preferred load address)

この例の場合、

|                                        |              |
|----------------------------------------|--------------|
| 実行時異常終了アドレス                            | : 0x04e91232 |
| ー 異常終了したDLLファイルの実行時開始アドレス              | : 0x04e90000 |
| + リンク時ロードアドレス (Preferred load address) | : 0x10000000 |
| -----                                  |              |
| 異常終了アドレス                               | : 0x10001232 |

## 3. モジュール開始アドレスの取得

モジュール開始アドレスの取得方法については、“[19.7.2.1 障害発生箇所の特定方法\(EXEファイルの場合\)](#)”の“モジュール開始アドレスの取得”を参照してください。

## 4. 異常終了アドレスのオブジェクト内相対オフセットの求め方

異常終了アドレスのオブジェクト内相対オフセットの求め方については、“[19.7.2.1 障害発生箇所の特定方法\(EXEファイルの場合\)](#)”の“異常終了アドレスのオブジェクト内相対オフセットの求め方”を参照してください。

## 5. 異常終了したCOBOLの文の特定

異常終了したCOBOLの文の特定については、“[19.7.2.1 障害発生箇所の特定方法\(EXEファイルの場合\)](#)”の“異常終了したCOBOLの文の特定”を参照してください。

## 19.7.2.3 データの参照方法

異常終了した場所がプログラムまたはメソッドの場合、Microsoft社が提供しているデバッガを使用して、そのプログラムまたはメソッドのデータを参照することができます。以降では、データの参照方法について説明します。



- 参照することができるデータは、異常終了したプログラムまたはメソッドで使用するデータのみです。他のプログラムおよびメソッドのデータは参照することができません。
- 翻訳オプションOPTIMIZEを指定している場合は、最適化処理により、実行結果が領域に格納されない場合や文が移動・削除される場合があるため、データを正しく参照することができないことがあります。

### 1. デバッガの入手

ワトソンログを調査するために、Microsoft社が提供しているデバッガ (Microsoft Visual Studio for WindowsまたはDebugging Tools for Windows) を入手してください。「Debugging Tools for Windows」はMicrosoftのホームページで公開されています。

### 2. セクションの基点アドレスの取得

リンクマップリストとデバッガを使用して、各セクションの基点アドレスを求めます。



- COA (.rodata)  
リンクマップリストの Publibs by Value が '\$プログラム名.LIT' となっている行の Rva+Base にアドレスがあります。[1]
- GWA (.data)  
リンクマップリストの Publibs by Value が '\$プログラム名.DATA' となっている行の Rva+Base にアドレスがあります。[2]
- GW2 (.bss)  
リンクマップリストの Publibs by Value が '\$プログラム名.BSS' となっている行の Rva+Base にアドレスがあります。[3]
- EBP (スタック)  
EBPレジスタを表します。デバッガを使用して、アドレスを求めます。
- HGW (ヒープ)  
HGWの基点アドレスは、BHGに格納されます。プログラム制御情報リストを参照し、デバッガを使用してBHGに格納されているアドレスを求めます。[4]
- HPA (ヒープ)  
HPAの基点アドレスは、BODに格納されます。プログラム制御情報リストを参照し、デバッガを用いてBODに格納されているアドレスを求めます。[5]

リンクマップリストは以下のようになっています。

| Start         | Length           | Name         | Class      |
|---------------|------------------|--------------|------------|
| 0001:00000000 | 000043a2H        | .text        | CODE       |
| :             |                  |              |            |
| 0003:00000030 | 00002580H        | .data        | DATA       |
| 0003:000025b0 | 0000052cH        | .bss         | DATA       |
| 0004:00000000 | 0000025cH        | .rodata      | DATA       |
| :             |                  |              |            |
| Address       | Publibs by Value | Rva+Base     | Lib:Object |
| 0001:00000000 | \$A. CODE        | 00401000     | A. OBJ     |
| 0001:00000000 | _main            | 00401000 f   | A. OBJ     |
| :             |                  |              |            |
| 0003:00000030 | \$A. DATA        | [2] 00406030 | A. OBJ     |
| :             |                  |              |            |
| 0004:00000000 | \$A. BSS         | [3] 00409f00 | A. OBJ     |
| :             |                  |              |            |
| 0005:00000000 | \$A. LIT         | [1] 0040a000 | A. OBJ     |
| :             |                  |              |            |

プログラム制御情報リストのBHG,BODは以下のようになっています。

|              |                   |  |    |
|--------------|-------------------|--|----|
| ** STK **    |                   |  |    |
| * SCB *      |                   |  |    |
| EBP+FFFFFF40 | SCB FIXED AREA    |  | 92 |
| .....        | TL 1ST AREA       |  | 0  |
| * SGM *      |                   |  |    |
| EBP+FFFFFF98 | SGM POINTERS AREA |  | 8  |
| .....        | VPA               |  | 0  |
| .....        | PSA               |  | 0  |
| .....        | BVA               |  | 0  |
| EBP+FFFFFF98 | [5] BOD           |  | 4  |
| EBP+FFFFFF9C | [4] BHG           |  | 4  |
| :            |                   |  |    |

### 3. データのアドレスの取得

データマップリストから、データが存在するアドレスの情報を求めます。データマップリストの形式は“[3.2.2.6 データエリアに関するリスト](#)”を参照してください。

| 行番号          | 番地           | オフセット | 変位 | レベル | 名標 | 長さ(10) | 属性      | 基点          | 次元数 |
|--------------|--------------|-------|----|-----|----|--------|---------|-------------|-----|
| **LISTDBG1** |              |       |    |     |    |        |         |             |     |
| 6            | GWA+000000E8 |       | 0  | 01  | 答え | 2      | EXT-DEC | BGW. 000000 |     |
| 7            | GWA+000000EA |       | 0  | 01  | 除数 | 2      | EXT-DEC | BGW. 000000 |     |

|    |                         |           |           |             |
|----|-------------------------|-----------|-----------|-------------|
| 9  | [GWA+00000020]+00000000 | 0 01 被除数  | 2 EXT-DEC | BVA. 000001 |
| 10 | [HGW+00000024]+00000000 | 0 01 動的領域 | 2 EXT-DEC | BVA. 000002 |

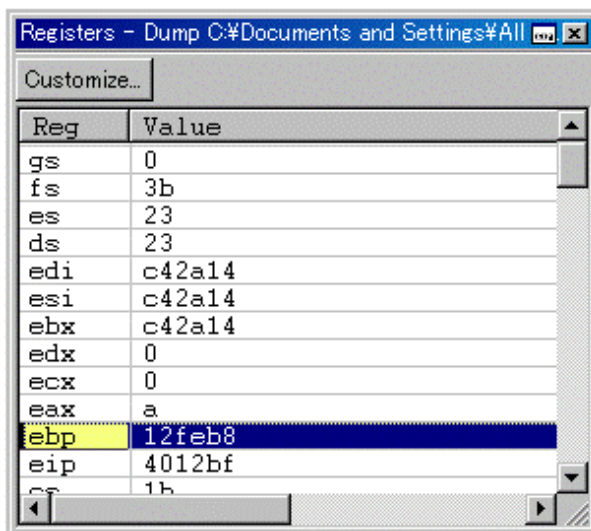
それぞれのデータは「番地」、「オフセット」が示すアドレスに格納されます。格納されるデータ領域長は「長さ」から判断します。上の例の場合、

- ー 答え  
GWA+000000E8が示すアドレスに格納され、データ領域長は2バイトです。
- ー 除数  
GWA+000000EAが示すアドレスに格納され、データ領域長は2バイトです。
- ー 被除数  
GWA+00000020が示すアドレスには、実アドレス(4バイト)が格納されます。そのアドレスにオフセット(この場合は00000000)を加えた位置にデータが格納されます。データ領域長は2バイトです。  
動的領域:HGW+00000024が示すアドレスには、実アドレス(4バイト)が格納されます。そのアドレスにオフセット(この場合は00000000)を加えた位置にデータが格納されます。データ領域長は2バイトです。

#### 4. データの参照

デバッガを使用して、3.で求めたアドレスからデータの内容を参照します。データマップリストは16進表記で記述されています。また、メモリにアドレスが格納されている場合は、表現形式はリトルエンディアン形式であるため、4バイトを逆順で参照する必要があります。

1. 「Debugging Tools for Windows」を立ち上げます。
2. 「File」メニューから「Open Crash Dump」を選択し、ワトソログを開きます。
3. レジスタを参照する場合、「View」メニューから「Registers」を選択します。



4. メモリを参照する場合、「View」メニューから「Memory」を選択し、参照したいアドレスを入力します。



例

#### 例1: “除数”のデータ参照

“除数”のデータマップリストの形式は以下です。

|   |              |         |           |             |
|---|--------------|---------|-----------|-------------|
| 7 | GWA+000000EA | 0 01 除数 | 2 EXT-DEC | BGW. 000000 |
|---|--------------|---------|-----------|-------------|

GWAの基点アドレスは、リンクマップファイルから「00406030」であることがわかります。

アドレスを入力する場合は、「00406030+EA」と入力します。

| Memory - Dump C:\Documents and Settings\All Users\Application Data\Microsoft\ |    |    |    |    |    |    |    |                      |    |    |    |    |    |    |    |
|-------------------------------------------------------------------------------|----|----|----|----|----|----|----|----------------------|----|----|----|----|----|----|----|
| Virtual: 00406030+EA                                                          |    |    |    |    |    |    |    | Display format: Byte |    |    |    |    |    |    |    |
| 0040611a                                                                      | 30 | 30 | 00 | 00 | 00 | 00 | 00 | 47                   | 43 | 42 | 04 | 80 | 00 | 00 | 00 |
| 0040612a                                                                      | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00                   | 00 | 00 | 00 | 00 | 00 | 00 | 78 |
| 0040613a                                                                      | 49 | 41 | 01 | 00 | 00 | 00 | 24 | ff                   | 12 | 00 | 78 | 38 | 04 | 1c | 94 |
| 0040614a                                                                      | a7 | 00 | 80 | 12 | 40 | 00 | 00 | 00                   | 00 | 00 | 58 | 13 | 40 | 00 | 00 |
| 0040615a                                                                      | 00 | 00 | b8 | 10 | 54 | 0c | 00 | 03                   | 44 | 00 | b0 | 00 | 00 | 00 | 20 |
| 0040616a                                                                      | 40 | 00 | 20 | 92 | 40 | 00 | 00 | 00                   | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0040617a                                                                      | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00                   | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0040618a                                                                      | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00                   | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

“除数”のデータの内容は「0x3030」であることがわかります。

## 例2: “被除数”のデータ参照

“被除数”のデータマップリストの形式は以下です。

|   |                         |          |           |             |
|---|-------------------------|----------|-----------|-------------|
| 9 | [GWA+00000020]+00000000 | 0 01 被除数 | 2 EXT-DEC | BVA. 000001 |
|---|-------------------------|----------|-----------|-------------|

このように間接参照になっている場合、[GWA+00000020]の指す位置には実アドレスが格納されています。

GWAの基点アドレスは、上記と同様に「00406030」なので、「00406030+20」と入力します。

| Memory - Dump C:\Documents and Settings\All Users\Application Data\Microsoft\ |    |    |    |    |    |    |    |                      |    |    |    |    |    |    |    |
|-------------------------------------------------------------------------------|----|----|----|----|----|----|----|----------------------|----|----|----|----|----|----|----|
| Virtual: 00406030+20                                                          |    |    |    |    |    |    |    | Display format: Byte |    |    |    |    |    |    |    |
| 00406050                                                                      | f8 | 61 | 40 | 00 | 78 | 4c | 49 | 41                   | 01 | 00 | 00 | 00 | 60 | fe | 12 |
| 00406060                                                                      | 78 | 38 | 04 | 1c | 94 | 06 | a7 | 00                   | 00 | 10 | 40 | 00 | 00 | 00 | 00 |
| 00406070                                                                      | c8 | 10 | 40 | 00 | 00 | 00 | 00 | 00                   | 88 | 10 | 54 | 0c | 00 | 03 | 44 |
| 00406080                                                                      | b8 | 00 | 00 | 00 | 30 | 60 | 40 | 00                   | 00 | 90 | 40 | 00 | 00 | 00 | 00 |
| 00406090                                                                      | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00                   | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 004060a0                                                                      | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00                   | 00 | 00 | 00 | 00 | 00 | 00 | 01 |
| 004060b0                                                                      | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00                   | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 004060c0                                                                      | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00                   | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

メモリにアドレスが格納されている場合、4バイトを逆順で参照する必要があります。これは、アドレスの入力表現形式のリトルエンディアンに合わせるためです。

上記の場合、格納されている実アドレスは「004061f8」です。

“被除数”のデータ内容は「004061f8+00000000」に格納されています。アドレスを入力する場合は、「004061f8」と入力します。(+00000000は省略しています。)

| Memory - Dump C:\Documents and Settings\All Users\Application Data\Microsoft\ |    |    |    |    |    |    |    |                      |    |    |    |    |    |    |    |
|-------------------------------------------------------------------------------|----|----|----|----|----|----|----|----------------------|----|----|----|----|----|----|----|
| Virtual: 004061f8                                                             |    |    |    |    |    |    |    | Display format: Byte |    |    |    |    |    |    |    |
| 004061f8                                                                      | 31 | 30 | 00 | 00 | 30 | 1a | 40 | 00                   | 01 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00406208                                                                      | 05 | 00 | 00 | c0 | 0b | 00 | 00 | 00                   | 00 | 00 | 00 | 00 | 1d | 00 | 00 |
| 00406218                                                                      | 04 | 00 | 00 | 00 | 00 | 00 | 00 | 00                   | 96 | 00 | 00 | c0 | 04 | 00 | 00 |
| 00406228                                                                      | 00 | 00 | 00 | 00 | 8d | 00 | 00 | c0                   | 08 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00406238                                                                      | 8e | 00 | 00 | c0 | 08 | 00 | 00 | 00                   | 00 | 00 | 00 | 00 | 8f | 00 | 00 |
| 00406248                                                                      | 08 | 00 | 00 | 00 | 00 | 00 | 00 | 00                   | 90 | 00 | 00 | c0 | 08 | 00 | 00 |
| 00406258                                                                      | 00 | 00 | 00 | 00 | 91 | 00 | 00 | c0                   | 08 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00406268                                                                      | 92 | 00 | 00 | c0 | 08 | 00 | 00 | 00                   | 00 | 00 | 00 | 00 | 93 | 00 | 00 |

“被除数”のデータの内容は「0x3130」であることがわかります。

## 例3: “動的領域”のデータ参照

“動的領域”のデータマップリストの形式は以下です。

|    |                         |           |           |             |
|----|-------------------------|-----------|-----------|-------------|
| 10 | [HGW+00000024]+00000000 | 0 01 動的領域 | 2 EXT-DEC | BVA. 000002 |
|----|-------------------------|-----------|-----------|-------------|

HGWAの基点アドレスはBHGに格納されているため、プログラム制御情報リストから「EBP+FFFFFF9C」の位置に格納されていることがわかります。そのため、まず「EBP+FFFFFF9C」と入力します。

| Memory - Dump C:\Documents and Settings\All Users\Application Data\Microsoft% |                                              |
|-------------------------------------------------------------------------------|----------------------------------------------|
| Virtual: EBP+FFFFFF9C                                                         | Display format: Byte                         |
| 0012fe54                                                                      | 14 2a c4 00 00 00 00 00 30 4c 49 56 3c 2a c4 |
| 0012fe64                                                                      | b8 2a c4 00 74 ff 12 00 58 60 40 00 24 ff 12 |
| 0012fe74                                                                      | 00 00 00 00 00 00 00 00 00 00 00 00 b8 fe 12 |
| 0012fe84                                                                      | 00 00 00 00 00 00 00 00 78 01 a6 00 5c ff 12 |
| 0012fe94                                                                      | 12 0e 95 7c 08 00 00 00 74 ff 12 00 23 14 40 |
| 0012fea4                                                                      | 18 61 40 00 00 00 00 00 f0 fd 7f 3c 2a c4    |
| 0012feb4                                                                      | 43 42 4c 20 74 ff 12 00 c6 14 40 00 18 61 40 |
| 0012fec4                                                                      | 08 0b a6 00 b8 fe 12 00 03 00 00 00 78 01 a6 |
| 0012fed4                                                                      | 00 00 00 00 a8 11 a6 00 ff ff ff ff 88 11 a6 |

HGWの基点アドレスは、「004c2a14」であることがわかります。

次に[HGW+00000024]に格納されている実アドレスを参照します。この場合、「004c2a14+24」と入力します。

| Memory - Dump C:\Documents and Settings\All Users\Application Data\Microsoft% |                                              |
|-------------------------------------------------------------------------------|----------------------------------------------|
| Virtual: 00C42A14+24                                                          | Display format: Byte                         |
| 00c42a38                                                                      | 18 61 40 00 78 4c 49 41 01 00 00 00 5c fe 12 |
| 00c42a48                                                                      | 78 38 04 1c 94 06 a7 00 00 10 40 00 00 00 00 |
| 00c42a58                                                                      | c8 10 40 00 00 00 00 00 c8 10 54 0c 00 03 44 |
| 00c42a68                                                                      | c0 00 00 00 14 2a c4 00 00 90 40 00 00 00 00 |
| 00c42a78                                                                      | 30 60 40 00 00 00 00 00 00 00 00 00 41 00 00 |
| 00c42a88                                                                      | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 |

上記の場合、格納されている実アドレスは「00406118」です。

“動的領域”のデータ内容は「00406118+00000000」に格納されています。アドレスを入力する場合は、「00406118」と入力します。(+00000000は省略しています。)

| Memory - Dump C:\Documents and Settings\All Users\Application Data\Microsoft% |                                              |
|-------------------------------------------------------------------------------|----------------------------------------------|
| Virtual: 00406118                                                             | Display format: Byte                         |
| 00406118                                                                      | 31 30 00 00 bc 1a 40 00 01 00 00 00 00 00 00 |
| 00406128                                                                      | 05 00 00 c0 0b 00 00 00 00 00 00 00 1d 00 00 |
| 00406138                                                                      | 04 00 00 00 00 00 00 00 96 00 00 c0 04 00 00 |
| 00406148                                                                      | 00 00 00 00 8d 00 00 c0 08 00 00 00 00 00 00 |
| 00406158                                                                      | 8e 00 00 c0 08 00 00 00 00 00 00 00 8f 00 00 |
| 00406168                                                                      | 08 00 00 00 00 00 00 00 90 00 00 c0 08 00 00 |

“動的領域”のデータの内容は「0x3130」であることがわかります。

同様に、[HPA+xxxxxxxx]+yyyyyyyyとなっている場合は、BHGを参照してHPAアドレスを求め、変数の領域を参照します。

.....

## 第20章 対話型デバッガの使い方

本章では、対話型デバッガ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ています。

### 20.1 デバッガの概要

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

デバッグ作業は、画面に表示したCOBOLソースプログラムに対する直接的で簡単な操作で行うことができます。キーボードやマウスを使い、メニュー内のコマンドやツールバーに表示されたボタンを操作することによって行います。また、ラインコマンド入力ウィンドウから、ラインコマンドと呼ばれる一定のルールに従った文字列を入力することによって、デバッグ作業を行うことができます。

デバッガは、デバッガの基本的な機能(中断点、データの表示/変更など)に加えて、以下の機能も持っています。

#### 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。

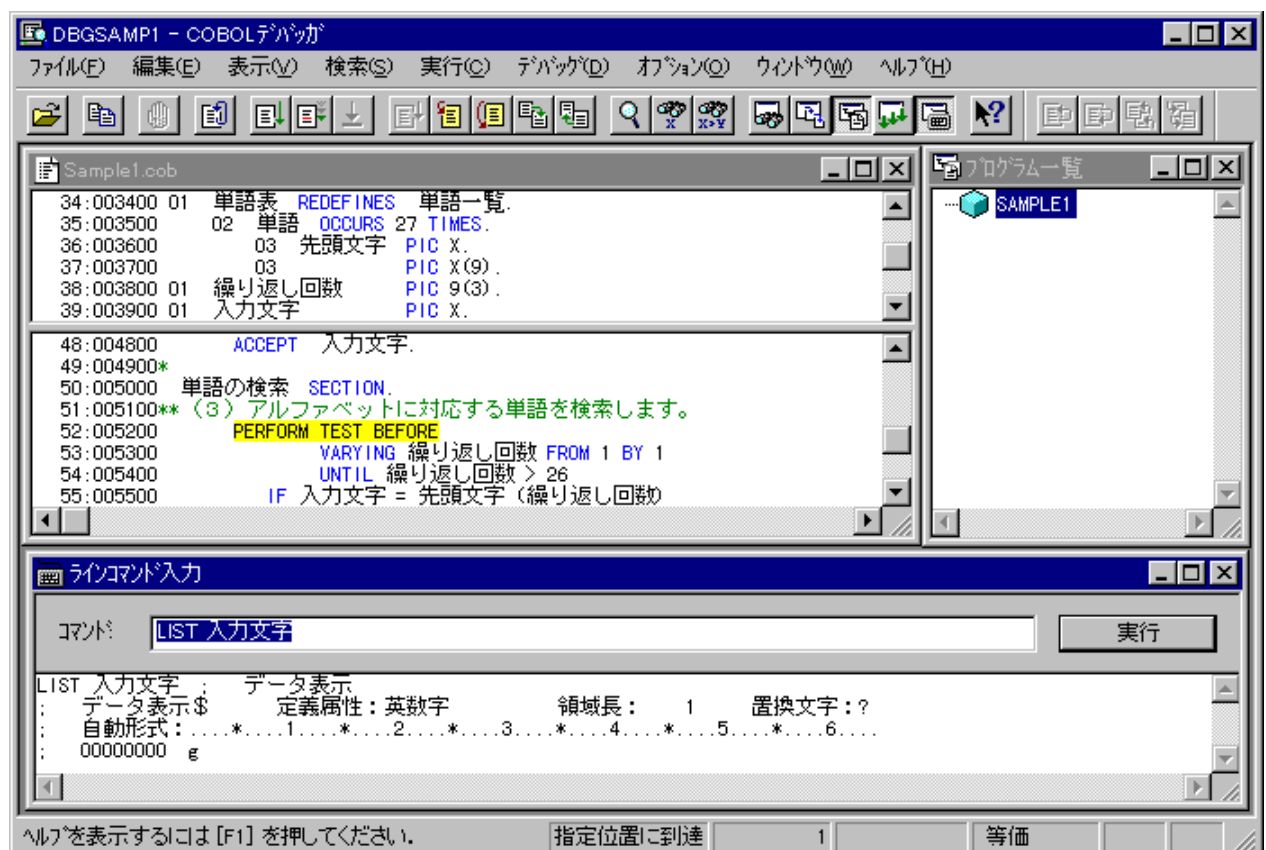
#### 特定実行条件での中断

動詞の種類やファイルを指定し、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

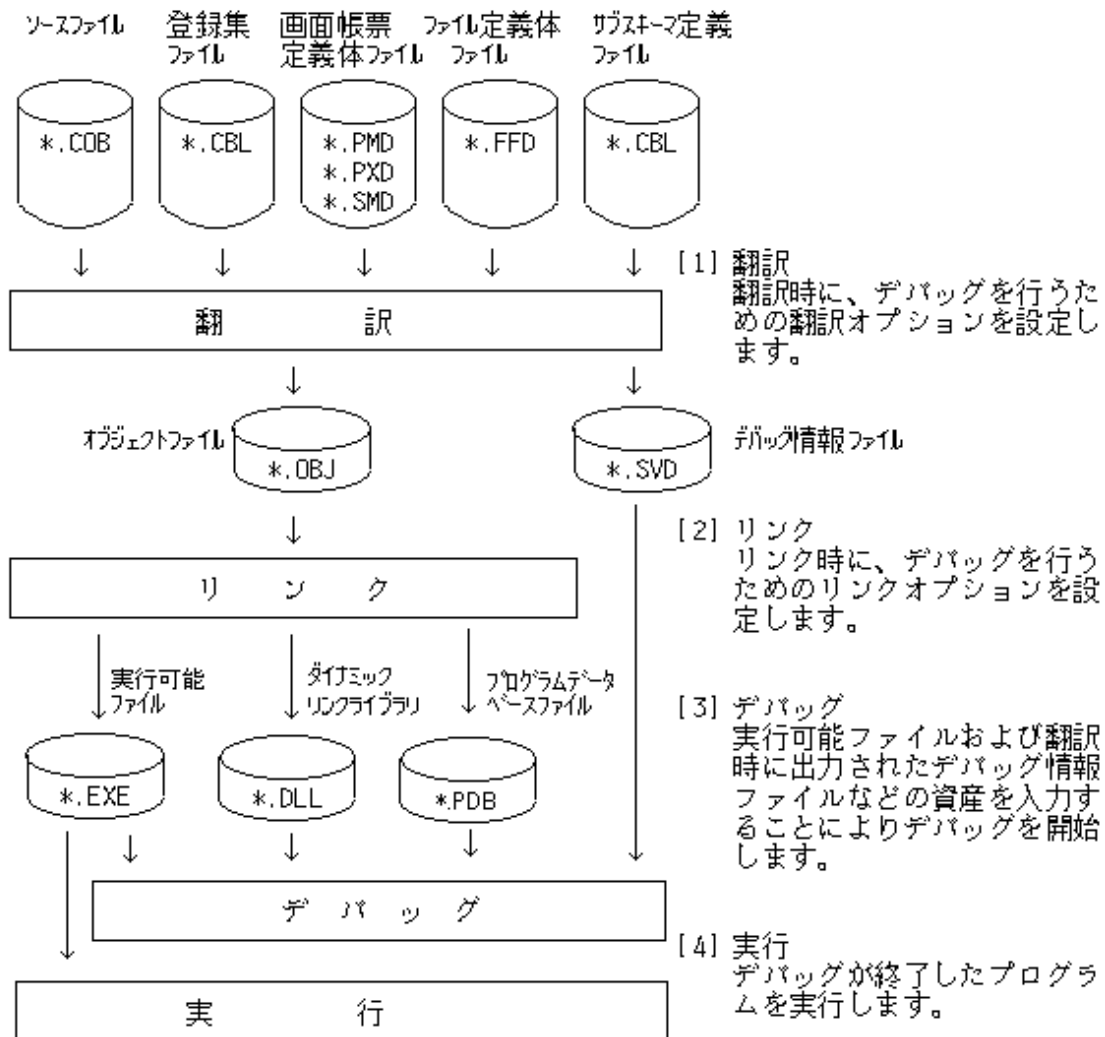
ファイルからコマンドを入力することにより、バッチデバッグを行うことができます。また、操作の履歴をファイルに保存し、これを入力として使用することにより、デバッグ操作を再現させることができます。

また、デバッガと同じコンピュータで動作するプログラムだけでなく、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リモートデバッグすることもできます。本章では、まず通常の対話型デバッガについて説明します。対話型リモートデバッガについては、["20.9 対話型リモートデバッガの使い方"](#)を参照してください。



## 20.2 デバッグの準備

ここでは、デバッガを使用して、COBOLプログラムをデバッグする場合の準備について説明します。



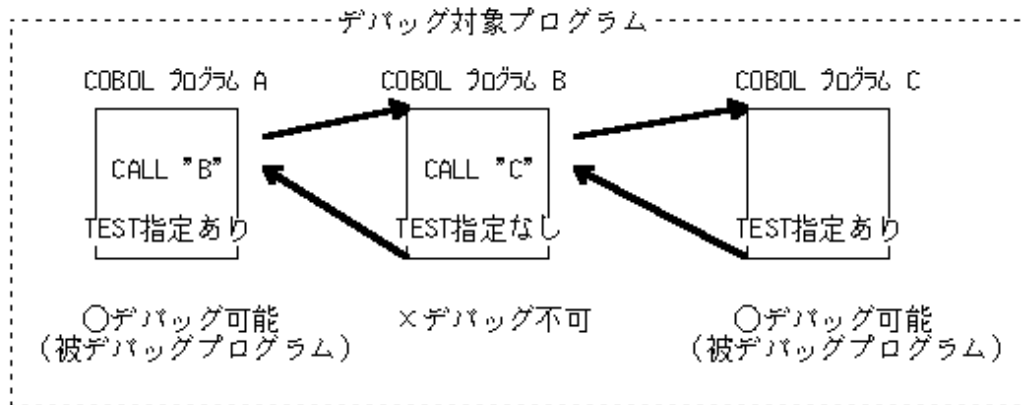
### 20.2.1 被デバッグプログラム

デバッガでは、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。

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。デバッガは、ユーザ資産の互換性を保っているため、旧資産を再翻訳や再リンクすることなく使用できます。





## 20.2.2 プログラムの翻訳

被デバッグプログラムを作成するためには、COBOLソースプログラムを翻訳するときに、翻訳オプションTESTを指定します。翻訳オプションTESTが指定されると、コンパイラは、オブジェクトファイル内にデバッグに必要な情報を出力するとともにデバッグ情報ファイルを作成します。デバッグ情報ファイルは、COBOLソースファイル名の拡張子をSVDに置き換えたファイル名になります。

なお、プロジェクト管理機能を利用すると簡単に被デバッグプログラムを作成することができます。詳細については、“[6.4.5.1 デバッグモジュールの作成方法](#)”を参照してください。

TESTオプションを指定して翻訳したCOBOLソースプログラムはデバッガの機能を利用したデバッグの操作ができます。TESTオプションが指定されていないCOBOLソースプログラムや他の言語で記述されたプログラムは、デバッガ配下で実行することはできませんが、デバッグの操作はできません。また、TESTオプションと同時にOPTIMIZEオプションを指定して翻訳したCOBOLソースプログラムの場合には、デバッグ情報ファイルを使用できないため、デバッグの操作を行うことはできません。[参照]“[A.2.56 TEST \(対話型デバッガおよび診断機能の使用の可否\)](#)”

## 20.2.3 プログラムのリンク

オブジェクトファイルをリンクするときには、リンクオプション“/DEBUG”および“/DEBUGTYPE”を指定します。これらのオプションを指定すると、リンカはオブジェクトファイル内に出力されたデバッグに必要な情報を、実行可能ファイルやダイナミックリンクライブラリに出力します。

以下に、リンクオプション“/DEBUG”および“/DEBUGTYPE”の指定形式を示します。

```
/DEBUG
/DEBUGTYPE: {COFF | BOTH}
```

デバッグ対象プログラムを作成するには、リンクオプションに“/DEBUG /DEBUGTYPE:COFF”を指定します。

なお、プロジェクト管理機能を利用すると簡単にデバッグ対象プログラムを作成することができます。詳細については、“[6.4.5.1 デバッグモジュールの作成方法](#)”を参照してください。

Visual C++を使用してC言語との結合を行うときには、リンクオプションに“/DEBUG /DEBUGTYPE:BOTH”を指定します。他の言語製品については、各製品のマニュアルに従ってください。



**注意**

デバッグ対象プログラムにしないプログラムを作成する場合には、リンクオプション“/DEBUG”および“/DEBUGTYPE”は指定しないでください。

## 20.3 デバッガの環境変数

ここでは、デバッガが使用する環境変数の設定について説明します。

デバッガが使用する環境変数は、バッチデバッグに関係する環境変数であるため、バッチデバッグを行わない場合には、特に指定する必要はありません。

バッチデバッグについては、“[20.5.2.2 デバッグ操作の自動化](#)”を参照してください。

環境変数は、以下のいずれかの方法で設定します。

- ・ コントロールパネルのシステムで設定する。
- ・ SETコマンドを使って設定する。

SETコマンドによる設定方法については、“[5.3.2.2 SETコマンドで設定する方法](#)”を参照してください。

AUTOEXEC.BATの設定方法やコントロールパネルから設定する方法については、各システムのヘルプを参照してください。

### 20.3.1 @SVD\_PERFORM\_CHECK (PERFORMレベルの監視を行う指定)

---

$$@SVD\_PERFORM\_CHECK = \left\{ \begin{array}{c} YES \\ NO \end{array} \right\}$$

デバッガでPERFORMレベルの監視を行う(YES)か、行わない(NO)かを指定します。

この環境変数を設定すると、PERFORMレベルの監視を行うかどうかの初期値を環境変数で指定した状態にできます。

環境変数を設定しない場合は、以下のようになります。

- ・ バッチデバッグの場合  
PERFORMレベルの監視を行いません。
- ・ バッチデバッグ以外の場合  
前回のデバッグ終了時の指定内容に従います。

### 20.3.2 @SVD\_ENV (デバッグ環境の指定)

---

$$\begin{aligned} @SVD\_ENV = & \left[ \left\{ \begin{array}{c} DIFF \\ SAME[ \left( \left\{ \begin{array}{c} ALL \\ WORD \end{array} \right\} ) ] \right\} \right] \text{ II } \left\{ \begin{array}{c} BREAK \\ CONTINUE \end{array} \right\} \text{ ][ SIGNAL( } \left\{ \begin{array}{c} BREAK \\ CONTINUE \end{array} \right\} \text{ )] } \\ & \left[ \text{THREADEND( } \left\{ \begin{array}{c} BREAK \\ CONTINUE \end{array} \right\} \text{ )] } \end{aligned}$$

デバッグ環境に関する指定をします。

この環境変数は、COBOL97 V60L10以前のデバッガで使用されていたコマンドファイルを使ってバッチデバッグを行う場合のために用意されています。

この環境変数を設定するとデバッグ環境に関する初期値を環境変数で指定した状態にできます。



## DIFF

ラインコマンドで英文字を含むデータ名、またはプログラム名を扱う場合に、小文字と大文字を区別して扱います。

## SAME(ALL)

ラインコマンドで英文字を含むデータ名、またはプログラム名を扱う場合に、小文字と大文字を区別せずに扱います。

## SAME(WORD)

ラインコマンドで英文字を含むデータ名、またはプログラム名を扱う場合に、プログラム名のみ小文字を大文字として扱いません。

## BREAK

例外条件が発生した場合に、例外手続きが処理される前でプログラムの実行を中断します。BREAKの省略形としてはBが使えます。

## CONTINUE

例外条件が発生した場合に、例外手続きが処理される前でプログラムの実行を中断しません。CONTINUEの省略形としてはCONTとCが使えます。

## SIGNAL(BREAK)

初回のシグナルを受信した場合にプログラムの実行を中断します。SIGNALの省略形としてはSIGが使えます。BREAKの省略形としてはBが使えます。初回のシグナル受信については、“[20.6.2 例外ハンドラで処理されるシグナルのデバッグ方法](#)”を参照してください。

## SIGNAL(CONTINUE)

初回のシグナルを受信した場合にプログラムの実行を中断しません。SIGNALの省略形としてはSIGが使えます。CONTINUEの省略形としてはCONTとCが使えます。初回のシグナル受信については、“[20.6.2 例外ハンドラで処理されるシグナルのデバッグ方法](#)”を参照してください。

## THREADEND(BREAK)

すべてのスレッド終了時にプログラムの実行を中断します。THREADENDの省略形としてはTHEが使えます。BREAKの省略形としてはBが使えます。

## THREADEND(CONTINUE)

暗黙スレッドが終了したときのみプログラムの実行を中断します。THREADENDの省略形としてはTHEが使えます。CONTINUEの省略形としてはCONTとCが使えます。

環境変数を設定しない場合またはそれぞれのパラメタを省略した場合は、以下のようになります。

- バッチデバッグの場合
  - ラインコマンドで英文字を含むデータ名またはプログラム名を扱う場合には、小文字と大文字を区別せずに扱います。
  - 例外条件が発生した場合には、例外手続きが処理される前でプログラムの実行を中断します。
  - 初回のシグナルを受信した場合には、プログラムの実行を中断しません。
  - 暗黙スレッドが終了したときのみプログラムの実行を中断します。
- バッチデバッグ以外の場合

前回のデバッグ終了時の指定内容に従います。

## 20.3.3 @SVD\_COMPATIBILITY(互換性の指定)

---

---

$$@SVD\_COMPATIBILITY = \left\{ \begin{array}{l} \text{[COBOL97] バージョンレベル} \\ \text{NetCOBOL バージョンレベル} \end{array} \right\}$$

---

デバッグの機能範囲を限定するための指定をします。

この環境変数を指定すると、デバッグの互換性に関する初期値を環境変数で指定した状態にできます。

環境変数を設定しない場合は、前回のデバッグ終了時の指定内容に従います。

環境変数には、コマンドファイルを作成したときのデバッグの製品の種別とバージョンレベルを指定します。

デバッグの製品種別とバージョンレベルは、デバッグの[ヘルプ]メニューの“バージョン情報”を選択することにより開かれる[COBOL97のバージョン情報]ダイアログボックスまたは[NetCOBOLのバージョン情報]ダイアログボックスで確認できます。

バージョンレベルは、[COBOL97のバージョン情報]ダイアログボックスまたは[NetCOBOLのバージョン情報]ダイアログボックスに表示された記述形式で指定してください。

#### [COBOL97] バージョンレベル

バージョンレベルの指定により、以下の動作をします。

- V50L10より前のバージョンレベルを指定した場合、V40系との互換動作を維持します。
- V50L10を指定した場合、V50L10との互換動作を維持します。
- V60L10を指定した場合、V60L10との互換動作を維持します。
- V61L10以降のバージョンレベルを指定した場合、V61L10との互換動作を維持します(最新の動作と同じです)。

#### NetCOBOL バージョンレベル

常に最新の動作をします。

バージョンレベルには、V7.0L10以降のバージョンレベルを指定します。

## 20.4 デバッグの起動

---

ここでは、デバッグの起動方法およびデバッグに必要な情報の設定について説明します。

### 20.4.1 起動方法

---

デバッグを開始する方法には、3つの方法があります。

#### 一般的なプログラムに対するデバッグ起動方法

デバッグを起動し、デバッグしたいプログラムを指定してデバッグを開始する起動方法です。

#### Interstageなどのサーバ環境で動作するプログラムに対するデバッグ起動方法

デバッグしたいプログラムからデバッグを起動し、デバッグを開始する起動方法です。COBOLプログラムがサーバ製品から呼び出されるために、一般的なプログラムに対するデバッグ方法では、デバッグするプログラムを指定できない場合に、この起動方法を使用します。

#### ジャストインタイムデバッグ機能を使用したデバッグ起動方法

プログラムでアプリケーションエラーや実行時メッセージが発生した場合に、自動的にデバッグが起動してプログラムをデバッグできる状態になる起動方法です。

#### 20.4.1.1 一般的なプログラムに対するデバッグ起動方法

デバッグを起動し、デバッグしたいプログラムを指定してデバッグを開始する起動方法です。一般的によく使われる方法を以下に示します。

- プロジェクトマネージャウィンドウの[プロジェクト]メニューから“デバッグ”を選択します。
- プロジェクトマネージャウィンドウの[ツール]メニューの[デバッグ種別]で[Win32]を選択してから、以下のいずれかの方法で起動します。
  - プロジェクトマネージャウィンドウの[ツール]メニューから“デバッグ”を選択します。
  - コマンドプロンプトからコマンド形式を入力して起動します。[参照]“[20.4.2.2 起動コマンドを使ったデバッグ指定](#)”

デバッガが起動されると、デバッガのメインウィンドウが表示されます。デバッグを開始するには、[デバッグを開始する]ダイアログで、必要な情報を入力して[OK]ボタンをクリックします。デバッグが開始されると、ダイアログが閉じてデバッグ操作ができるようになります。

### 20.4.1.2 Interstageなどのサーバ環境で動作するプログラムに対するデバッグ起動方法

デバッグしたいプログラムからデバッガを起動する方法です。

環境変数情報@CBR\_ATTACH\_TOOLで指示し、対話型リモートデバッガを使用することによって、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をデバッグすることができます。

詳細は、“[20.9.2.2 Interstageなどのサーバ環境で動作するプログラムに対するリモートデバッグ起動方法](#)”を参照してください。

### 20.4.1.3 ジャストインタイムデバッグ機能を使用したデバッグ起動方法

プログラムでアプリケーションエラーや実行時メッセージが発生した場合に、自動的にデバッガが起動してプログラムをデバッグできる状態になる起動方法です。

環境変数情報@CBR\_JUSTINTIME\_DEBUGで指示することによって、アプリケーションエラーや実行時メッセージが発生したときにデバッガを起動し、デバッグを開始することができます。[参照]“[5.4.1.32 @CBR\\_JUSTINTIME\\_DEBUG \(異常終了時にデバッグまたは診断機能を使って調査を行う指定\)](#)”

プログラムを実行した時にアプリケーションエラーや実行時メッセージが発生すると、デバッガが起動され、デバッガのメインウィンドウが表示されます。デバッグを開始するには、[デバッグを開始する]ダイアログで[OK]ボタンをクリックします。デバッグが開始されると、アプリケーションエラーまたは実行時メッセージが発生した状態で中断します。

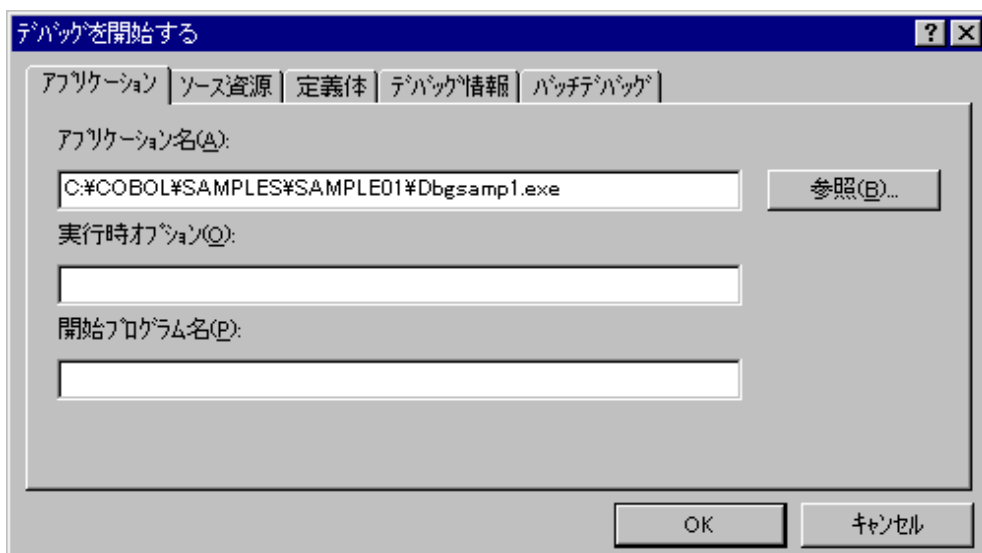
## 20.4.2 デバッグに必要な資源やパラメタの指定

プログラムをデバッグするには、デバッガが必要とする資源(ソースファイルやデバッグ情報ファイルなど)やパラメタを指定しなければなりません。ここでは、[デバッグを開始する]ダイアログおよびコマンド形式での起動パラメタの指定方法について説明します。

### 20.4.2.1 [デバッグを開始する]ダイアログを使ったデバッグ指定

ここでは、[デバッグを開始する]ダイアログを使ってデバッガの起動パラメタを指定する方法について説明します。以下に、[デバッグを開始する]ダイアログを各ページごとに示します。

#### アプリケーションページ



## エディットボックス

### アプリケーション名

デバッグを行う実行可能プログラムを指定します。相対パス名で指定した場合は、デバッガ起動時のカレントフォルダ名が付加されます。拡張子を省略した場合は、拡張子EXEのファイルを指定したものとみなされます。

このパラメタは、省略できません。

### 実行時オプション

実行可能プログラムに渡すオプションを指定します。

このパラメタは、省略可能です。

### 開始プログラム名

デバッグを開始して最初にコマンドの入力が可能になるプログラム名を指定します。指定するプログラムは、被デバッグプログラムでなければなりません。

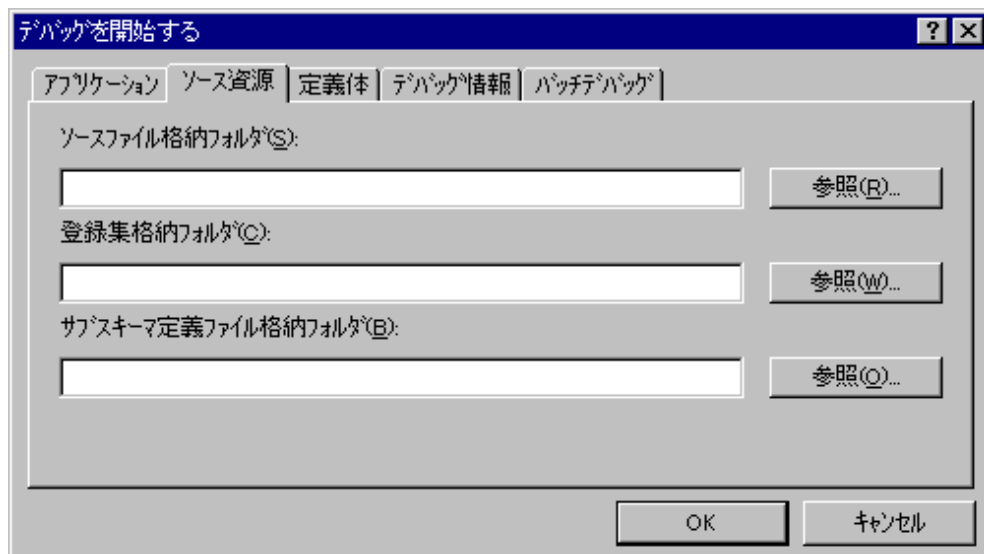
このパラメタは、省略可能です。省略した場合は、プログラムの実行で最初に見つかった被デバッグプログラムでのデバッグ操作が可能になります。

## ボタン

### 参照

ファイルを参照するときに選択します。

## ソース資源ページ



## エディットボックス

### ソースファイル格納フォルダ

ソースファイルを格納しているフォルダを指定します。複数個のフォルダを指定したい場合は、セミコロン(;)で区切って指定します。

このパラメタは、デバッグ中のCOBOLソースプログラムをメインウィンドウに表示するために必要となります。デバッガは、指定されたフォルダ順にソースファイルを検索します。

このパラメタは、省略可能です。省略した場合および指定されたフォルダから検索できなかった場合は、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が格納されているフォルダを検索します。

### 登録集格納フォルダ

登録集ファイルを格納しているフォルダを指定します。複数個のフォルダを指定したい場合は、セミコロン(;)で区切って指定します。

このパラメタは、デバッグ中のCOBOLソースプログラムから取り込まれる登録集ファイルをメインウィンドウに表示するために必要となります。デバッガは、指定されたフォルダ順に登録集ファイルを検索します。

このパラメタは、省略可能です。省略した場合および指定されたフォルダから検索できなかった場合は、以下の順で検索します。

1. 環境変数COB\_COBCOPYで指定されたフォルダ
2. 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が格納されているフォルダ

IN/OF指定の登録集ファイルは、以下の順で検索します。

1. 環境変数COB\_登録集名で指定されたフォルダ
2. 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が格納されているフォルダ

[参照]“[1.2.1 環境変数の設定](#)”

#### サブスキーマ定義ファイル格納フォルダ

サブスキーマ定義ファイルを格納しているフォルダを指定します。複数のフォルダを指定したい場合は、セミicolon(;)で区切って指定します。

このパラメタは、デバッグ中のCOBOLソースプログラムから取り込まれるサブスキーマ定義ファイルをメインウィンドウに表示するために必要となります。デバッガは、指定されたフォルダ順にサブスキーマ定義ファイルを検索します。

このパラメタは、省略可能です。省略した場合および指定されたフォルダから検索できなかった場合は、以下の順で検索します。

1. 環境変数COB\_AIMLIBで指定されたフォルダ
2. 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が格納されているフォルダ

[参照]“[1.2.1 環境変数の設定](#)”

#### ボタン

##### 参照

フォルダを参照するときに選択します。

#### 定義体ページ

The screenshot shows a Windows-style dialog box titled "デバッグを開始する" (Start Debugging). It has five tabs: "アプリケーション" (Application), "ソース資源" (Source Resources), "定義体" (Definition), "デバッグ情報" (Debug Information), and "バッチデバッグ" (Batch Debugging). The "定義体" (Definition) tab is selected. Inside, there are two sections: "画面帳票定義体" (Screen Form Definition) and "ファイル定義体" (File Definition). Each section has a "フォルダ(F):" (Folder) text box, an "拡張子(E):" (Extension) dropdown menu, and a "参照(R)..." (Reference...) button. In the "画面帳票定義体" section, the folder is empty and the extension is "pmd;pxd;smd". In the "ファイル定義体" section, the folder is empty and the extension is "ffd". At the bottom right are "OK" and "キャンセル" (Cancel) buttons.

#### エディットボックス

##### 画面帳票定義体のフォルダ

画面帳票定義体ファイルを格納しているフォルダを指定します。複数のフォルダを指定したい場合は、セミicolon(;)で区切って指定します。

このパラメータは、デバッグ中のCOBOLソースプログラムから取り込まれる画面帳票定義体ファイルをメインウィンドウに表示するために必要となります。デバッガは、指定されたフォルダ順に画面帳票定義体ファイルを検索します。

このパラメータは、省略可能です。省略した場合および指定されたフォルダから検索できなかった場合は、以下の順で検索します。

1. 環境変数FORMLIBで指定されたフォルダ
2. 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が格納されているフォルダ

[参照]“[1.2.1 環境変数の設定](#)”

#### ファイル定義体のフォルダ

ファイル定義体ファイルを格納しているフォルダを指定します。複数のフォルダを指定したい場合は、セミicolon(;)で区切って指定します。

このパラメータは、デバッグ中のCOBOLソースプログラムから取り込まれるファイル定義体ファイルをメインウィンドウに表示するために必要となります。デバッガは、指定されたフォルダ順にファイル定義体ファイルを検索します。

このパラメータは、省略可能です。省略した場合または指定されたフォルダから検索できなかった場合は、以下の順で検索します。

1. 環境変数FILELIBで指定されたフォルダ
2. 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が格納されているフォルダ

[参照]“[1.2.1 環境変数の設定](#)”

#### ドロップダウンコンボボックス

##### 画面帳票定義体の拡張子

画面帳票定義体ファイルの拡張子を指定します。

このパラメータは、初期値としてpmd;pxd;smdが設定されています。

##### ファイル定義体の拡張子

ファイル定義体ファイルの拡張子を指定します。

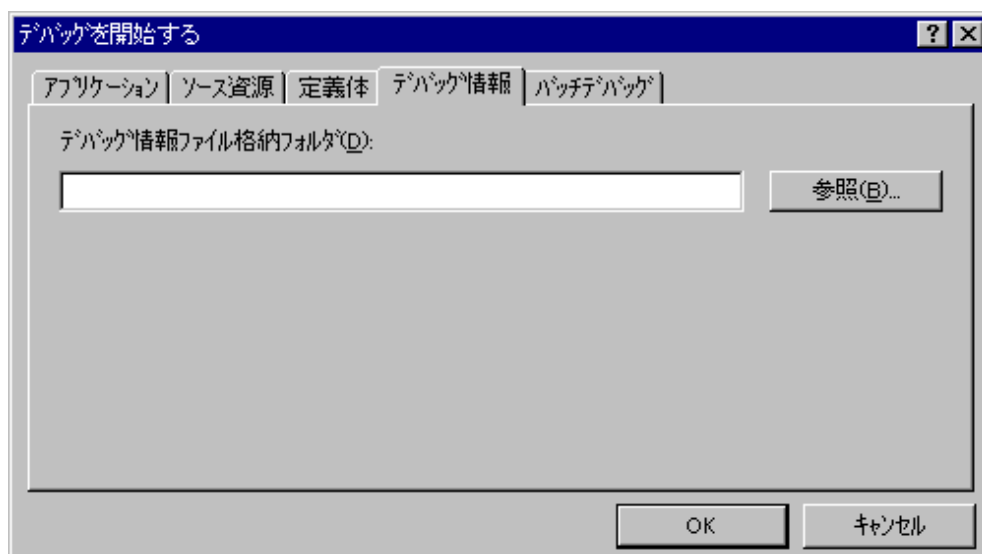
このパラメータは、初期値としてffdが設定されています。

#### ボタン

##### 参照

フォルダを参照するときに選択します。

### デバッグ情報ページ



## エディットボックス

### デバッグ情報ファイル格納フォルダ

デバッグ情報ファイルを格納しているフォルダを指定します。複数のフォルダを指定したい場合は、セミicolon(;)で区切って指定します。

デバッグは、指定されたフォルダ順にデバッグ情報ファイルを検索します。

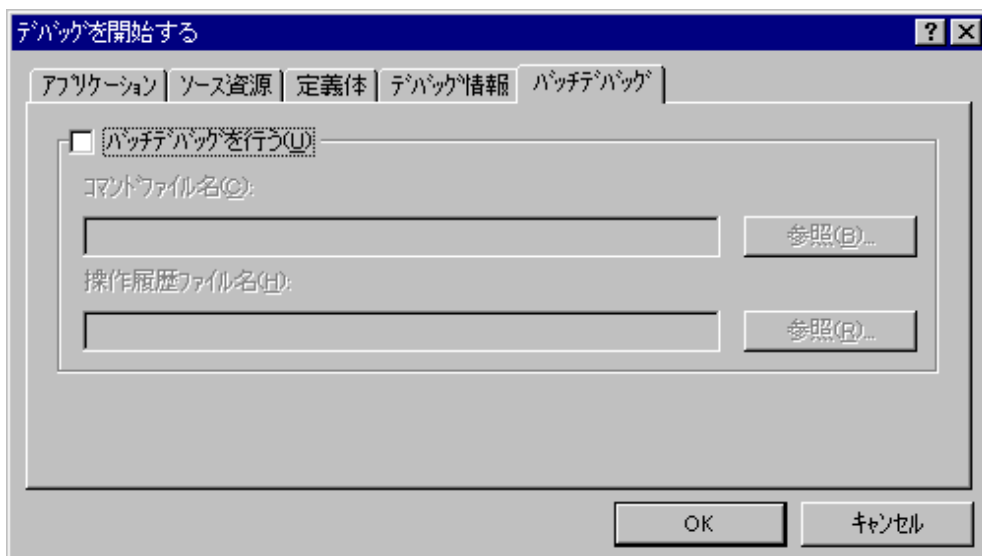
このパラメタは、省略可能です。省略した場合および指定されたフォルダから検索できなかった場合は、実行可能ファイルまたはダイナミックリンクライブラリが格納されているフォルダを検索します。

## ボタン

### 参照

フォルダを参照するときに選択します。

## バッチデバッグページ



## チェックボックス

### バッチデバッグを行う

画面からの対話処理を行わずに、一連のコマンドを格納したコマンドファイルから一括してデバッグ操作を行う場合に、このチェックボックスを選択します。

## エディットボックス

### コマンドファイル名

バッチデバッグを行う場合に、コマンドファイル名を指定します。相対パス名で指定した場合は、デバッグ起動時のカレントフォルダ名が付加されます。拡張子を省略した場合は、拡張子LOGのファイルを指定したものとみなされます。

このパラメタは、バッチデバッグを行うチェックボックスが指定されている場合は省略できません。

バッチデバッグおよびコマンドファイルについては、“[20.5.2.2 デバッグ操作の自動化](#)”を参照してください。

### 操作履歴ファイル名

デバッグ操作のコマンドとコマンドの実行結果を保存する場合に、保存先の操作履歴ファイル名を指定します。相対パス名で指定した場合は、デバッグ起動時のカレントフォルダ名が付加されます。拡張子を省略した場合には、拡張子LOGのファイルを指定したものとみなされます。

このパラメタは、バッチデバッグを行うチェックボックスが指定されている場合は省略できません。

ここに指定されたファイルがすでに存在している場合は、ファイルが上書きされます。

なお、コマンドファイル名のエディットボックスとこの操作履歴ファイル名のエディットボックスに同じファイルは指定できません。



## ボタン

### 参照

ファイルを参照するときに選択します。

## 20.4.2.2 起動コマンドを使ったデバッグ指定

ここでは、デバッグをコマンド形式で起動する場合の指定方法を説明します。

winsvdコマンドで起動するデバッグは、プロジェクトマネージャウィンドウの[ツール]メニューの[デバッグ種別]で選択します。起動コマンドを実行する前に、適切なデバッグ種別を選択しておく必要があります。

以下に、デバッグを起動するコマンドの形式を示します。なお、形式2では、winsvd、起動パラメタ、アプリケーション名、実行時オプションの指定は、以下に示す順序で指定しなければなりません。アプリケーション名は省略できません。起動パラメタと実行時オプションは省略することができます。

### 形式1

```
winsvd
```

形式1で起動すると、メインウィンドウが表示されます。デバッグを開始するには、[ファイル]メニューの“デバッグを開始する”を選択して、[デバッグを開始する]ダイアログを表示して、必要な情報を入力して、[OK]ボタンをクリックします。

### 形式2

```
winsvd [起動パラメタ] アプリケーション名 [実行時オプション]
```

形式2で起動すると、メインウィンドウが表示され、かつ、[デバッグを開始する]ダイアログが表示されます。この[デバッグを開始する]ダイアログの入力フィールドには、指定した起動パラメタ、アプリケーション名および実行時オプションが表示されます。デバッグを開始するには、[デバッグを開始する]ダイアログで必要な情報を入力して、[OK]ボタンをクリックします。

起動パラメタを“表20.1 起動パラメタ”に示します。

表20.1 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。            |
| /S ソースファイル格納フォルダ名      | [デバッグを開始する]ダイアログでのソースファイル格納フォルダ名を指定します。      |
| /C 登録集格納フォルダ名          | [デバッグを開始する]ダイアログでの登録集格納フォルダ名を指定します。          |
| /A サブスキーマ定義ファイル格納フォルダ名 | [デバッグを開始する]ダイアログでのサブスキーマ定義ファイル格納フォルダ名を指定します。 |
| /M 画面帳票定義体格納フォルダ名      | [デバッグを開始する]ダイアログでの画面帳票定義体格納フォルダ名を指定します。      |
| /E 画面帳票定義体拡張子          | [デバッグを開始する]ダイアログでの画面帳票定義体拡張子を指定します。          |
| /H ファイル定義体格納フォルダ名      | [デバッグを開始する]ダイアログでのファイル定義体格納フォルダ名を指定します。      |
| /O ファイル定義体拡張子          | [デバッグを開始する]ダイアログでのファイル定義体拡張子を指定します。          |
| /D デバッグ情報ファイル格納フォルダ名   | [デバッグを開始する]ダイアログでのデバッグ情報ファイル格納フォルダ名を指定します。   |
| /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。          |

起動パラメタの先頭文字は、スラッシュ(/)でもハイフン(-)でも同様に扱われます。また、先頭文字に続くパラメタ名は、大文字と小文字が同様に扱われます。複数の起動パラメタを指定する場合には、起動パラメタと次の起動パラメタとの間に1つ以上の空白を置いて指定します。





## 例

次の起動パラメタを指定する場合の入力コマンド

|                   |                      |
|-------------------|----------------------|
| アプリケーション名         | : e:¥exe¥sample1.exe |
| 実行時オプション          | : param1             |
| ソースファイル格納フォルダ名    | : d:¥source          |
| デバッグ情報ファイル格納フォルダ名 | : e:¥svd             |

```
winsvd /S d:¥source /D e:¥svd e:¥exe¥sample1.exe param1
```

## 20.5 デバッガの操作

ここでは、デバッガの画面および操作方法について説明します。

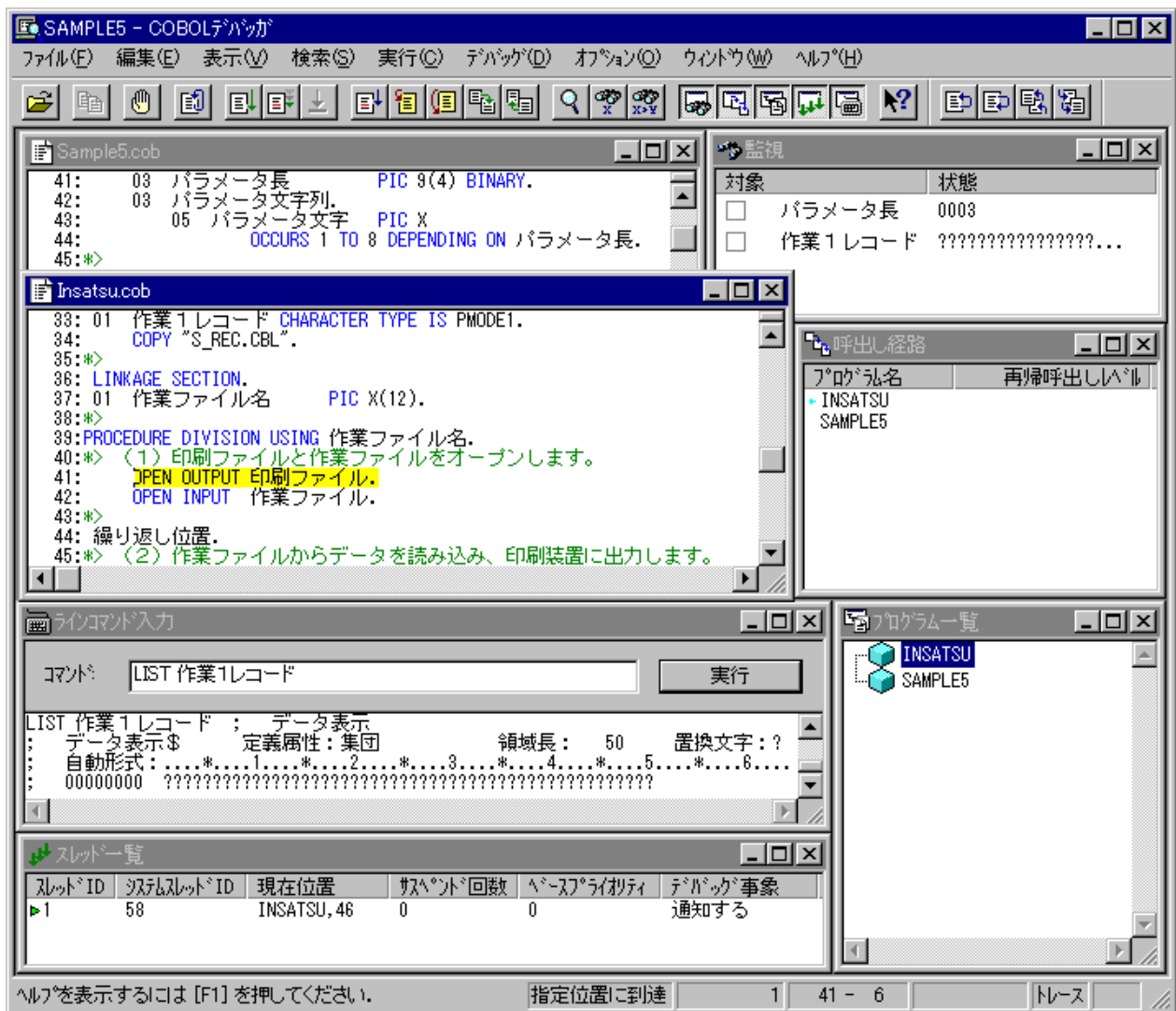
### 20.5.1 デバッガの画面

デバッガの画面構成を以下に示します。

#### 20.5.1.1 メインウィンドウ

デバッガのメインとなるウィンドウであり、このウィンドウ内にソースファイルウィンドウ、プログラム一覧ウィンドウ、監視ウィンドウ、ラインコマンド入力ウィンドウ、呼出し経路ウィンドウおよびスレッド一覧ウィンドウが表示されます。

以下に、メインウィンドウの表示例を示します。



デバッグ操作には、次の4種類があります。

#### メニューコマンドの選択

各コマンドの機能については、ヘルプを参照してください。

#### ツールバーのボタンの選択

ツールバーのボタンを選択することにより、デバッガの機能を実行できます。メニューコマンドでも同等の機能がありますが、ツールバーのボタン選択ですばやく実行できます。

ツールバーは、ドラッグして取り外したり移動したりできます。メインウィンドウ内のどこか一辺に取り付けた状態または取り外したままの状態で使用できます。

各ボタンの詳細については、ヘルプを参照してください。

#### ショートカットメニューのコマンドの選択

各ウィンドウ内をマウスの右ボタンでクリックすることにより、クリックした位置に表示されるメニューです。このメニューからコマンドを選択することにより、デバッグ操作を実行することができます。

ショートカットメニューのコマンドの詳細については、ヘルプを参照してください。

#### ラインコマンドの入力

ラインコマンド入力ウィンドウからラインコマンドを入力することにより、デバッガの機能を実行できます。

各ラインコマンドの使い方については、ヘルプを参照してください。

また、ステータスバーには、以下の情報が表示されます。

- 選択中のメニューコマンドに関する説明

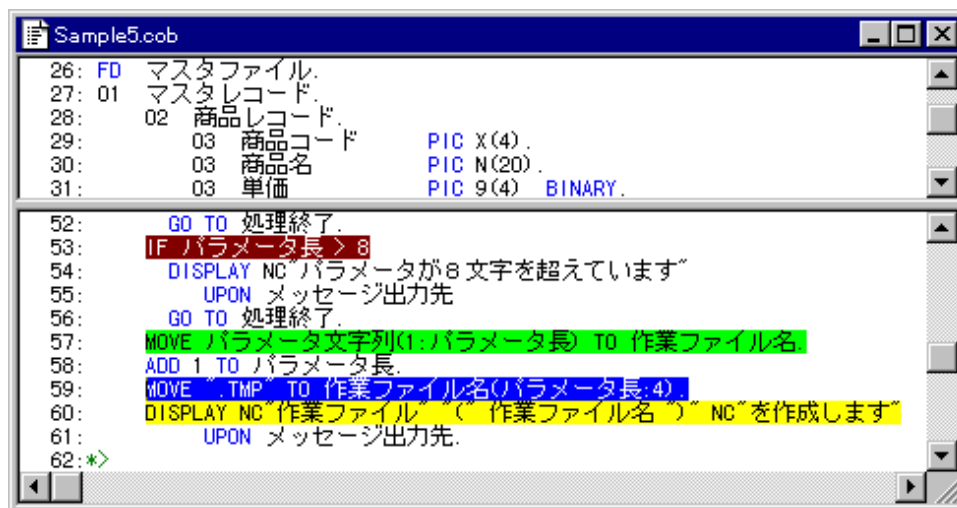
- ・ 実行が中断したときの中断事象
- ・ 現在の暗黙スレッドのスレッドID
- ・ ソースファイルウィンドウ内のカーソル位置
- ・ ラインコマンドにおける大文字/小文字の比較基準
- ・ トレース採取の状態
- ・ 履歴出力の状態

### 20.5.1.2 ソースファイルウィンドウ

ソースファイルウィンドウには、被デバッグプログラムのソースファイルや登録集ファイルが表示されます。これらのファイルを複数表示するときは、表示するファイルの数だけソースファイルウィンドウが開かれます。

ソースファイルウィンドウは、プログラムの実行が中断されたときやプログラムを追尾実行しているときなどに、必要に応じて自動的に開かれます。すでにメモリ上にロードされていて画面に表示されていないソースファイルは、プログラマー一覧ウィンドウを使って表示させることができます。登録集ファイルは、ショートカットメニューのコマンドを使って表示させることができます。

以下に、ソースファイルウィンドウの表示例を示します。



ソースファイルウィンドウは、分割バーによる二分割表示を行うことができます。

以下の文については、文に色を付けて他の文と区別して表示されます。なお、この色は変更することができます。

#### 現在中断している文

現在中断している文に色を付けます。設定を変更しなければ、この文の色は黄です。

#### 現在逆トレースされている文

現在逆トレースされている文に色を付けます。設定を変更しなければ、この文の色は青です。

#### 中断点が設定されている文

中断点が設定されている文に色を付けます。設定を変更しなければ、この文の色は赤です。

#### 通過カウント点が設定されている文

通過カウント点が設定されている文に色を付けます。通過カウント点が設定された後にプログラムを実行した場合、まだ実行されていない文と実行された文を別の色で区別することができます。設定を変更しなければ、未通過が緑、通過済が濃い緑です。

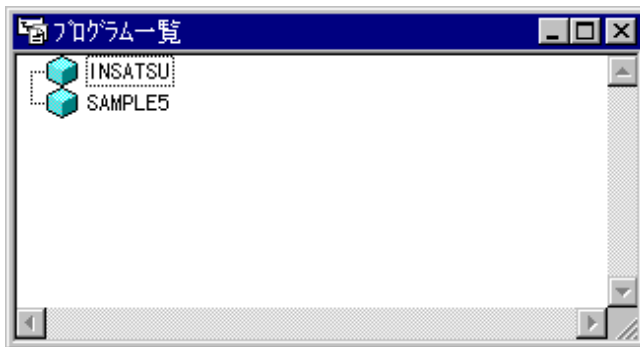
### 20.5.1.3 プログラマー一覧ウィンドウ

プログラマー一覧ウィンドウには、プログラムの一覧が表示されます。メモリ上にロードされているデバッグ対象プログラムに含まれる被デバッグプログラムがすべて表示されます。

プログラマー一覧ウィンドウは、[ウィンドウ]メニューの“プログラマー一覧”を選択することにより表示されます。

プログラム一覧ウィンドウは、デバッグ操作を行うことができる外部プログラムやクラスを確認したり、ソースファイルを開いたりする場合に使用します。

以下に、プログラム一覧ウィンドウの表示例を示します。



#### 20.5.1.4 監視ウィンドウ

監視ウィンドウは、以下の情報を表示します。

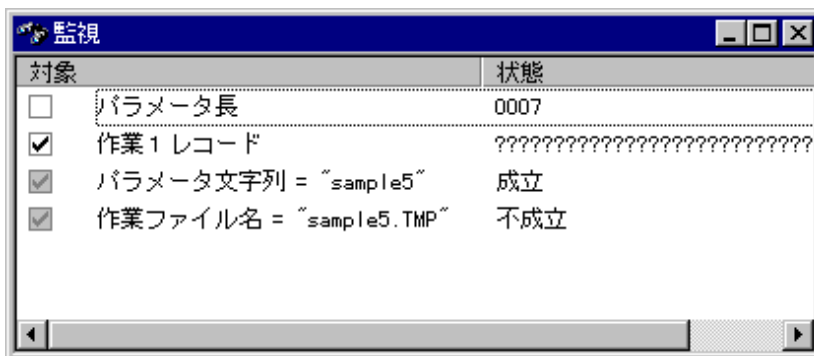
- ・ 値の変更を監視しているデータ名およびその内容
- ・ 条件の成立を監視している条件式およびその状態

監視ウィンドウは、データの監視または条件の監視を開始すると自動的に開かれます。また、[ウィンドウ]メニューの“監視”を選択することにより表示されます。

監視ウィンドウは、データの内容の変化や監視している条件を確認したり、監視しているデータの内容が変更されたときに実行を中断するように設定する場合に使用します。

データ監視および条件監視は、複数設定できます。

以下に、監視ウィンドウの表示例を示します。

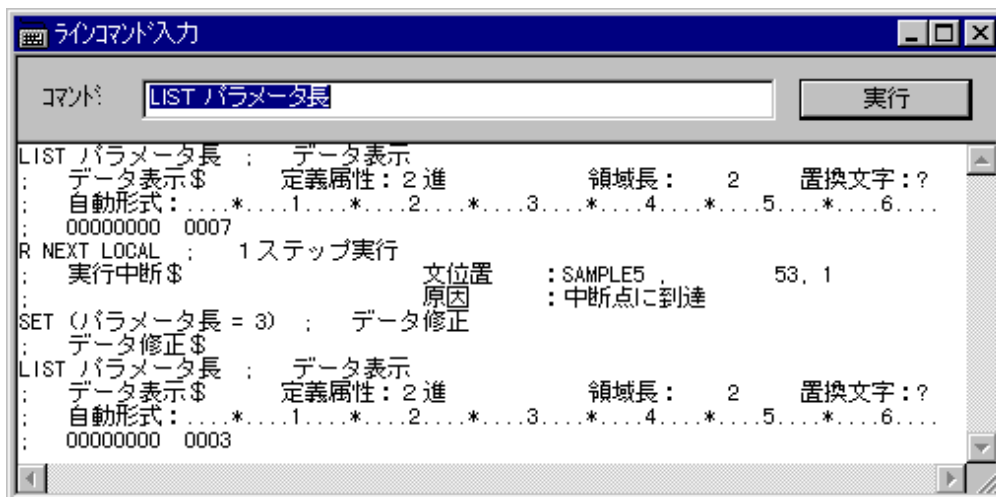


#### 20.5.1.5 ラインコマンド入力ウィンドウ

ラインコマンド入力ウィンドウは、ラインコマンドの入力およびその実行結果を表示します。ラインコマンド入力ウィンドウは、[ウィンドウ]メニューの“ラインコマンド入力”を選択することにより表示されます。

ラインコマンド入力ウィンドウは、ラインコマンドを使ってデバッグ操作を行うときに使用します。

以下に、ラインコマンド入力ウィンドウの表示例を示します。



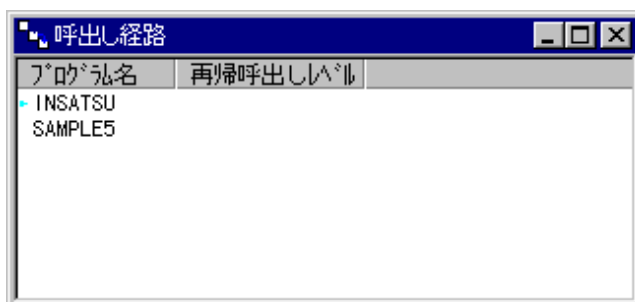
ラインコマンドの機能については、“[20.8 デバッガのコマンド一覧](#)”を参照してください。ラインコマンドの指定形式については、ヘルプを参照してください。

### 20.5.1.6 呼出し経路ウィンドウ

呼出し経路ウィンドウは、プログラムの呼出し経路を表示します。実行されたプログラムが呼び出された順に下から上へと表示されます。

呼出し経路ウィンドウは、[ウィンドウ]メニューの“呼出し経路”を選択することにより表示されます。デバッグ環境の設定により、追尾実行中には呼出し経路ウィンドウの表示を更新しないようにすることも可能です。呼出し経路ウィンドウは、プログラムの呼出し元を確認したり、ソースファイルを開いたりする場合に使用します。

以下に、呼出し経路ウィンドウの表示例を示します。



### 20.5.1.7 スレッド一覧ウィンドウ

スレッド一覧ウィンドウは、現在動作しているスレッドおよびスレッド終了事象が通知されたスレッドの一覧を表示します。

スレッド一覧は、[ウィンドウ]メニューの“スレッド一覧”を選択することにより表示されます。デバッグ環境の設定で、追尾実行中にはスレッド一覧ウィンドウの表示を更新しないようにすることもできます。

スレッド一覧ウィンドウは、現在動作しているスレッドの状態を確認したり、暗黙スレッドを切り替えたりする場合に使用します。

以下に、スレッド一覧ウィンドウの表示例を示します。

| スレッド ID | システムスレッド ID | 現在位置                 | サスペンド回数 | ベース優先度 | デバッグ事象 |
|---------|-------------|----------------------|---------|--------|--------|
| 7       | 0x000000b8  | 0x77f57e27           | 0       | 0      | 通知しない  |
| 8       | 0x000000dd  | HttpExtensionProc,36 | 1       | 0      | 通知する   |
| 9       | 0x000000bc  | 0x77f5817b           | 0       | 0      | 通知しない  |
| 10      | 0x0000005e  | 0x77f57e57           | 0       | 0      | 通知しない  |
| 11      | 0x000000d3  | 0x77f5818b           | 0       | 0      | 通知しない  |
| 12      | 0x000000b7  | 0x77f5817b           | 0       | 0      | 通知しない  |
| 13      | 0x000000d7  | 0x77f5817b           | 0       | 0      | 通知しない  |
| 14      | 0x00000117  | 0x77f5818b           | 0       | 1      | 通知しない  |
| 15      | 0x000000c0  | 0x00000000           | ---     | -----  | -----  |
| 16      | 0x000000c5  | 0x77f577cf           | 0       | 0      | 通知しない  |



## 注意

システムやサーバアプリケーションなどによっては、COBOLプログラムを実行するスレッド以外のスレッドが存在する場合があります、スレッド一覧ウィンドウにはそれらのスレッドを含めプロセス内のすべてのスレッドが表示されます。

## 20.5.2 デバッガの機能

ここでは、デバッグ機能の使い方について説明します。

### 20.5.2.1 デバッグ機能の使い方

ここでは、以下の状況でのデバッガの各コマンドの使い方について説明します。

- ・ データ内容を確認しながらプログラムの誤りを検出する
- ・ プログラムの実行順序を変更する
- ・ データの変更箇所を調べる
- ・ 実行経路をたどる
- ・ プログラムの呼出し経路をたどる
- ・ デバッガの状態を知る

#### 20.5.2.1.1 データの内容を確認しながらプログラムの誤りを検出する

プログラムの論理の正当性を確認するために、プログラムの実行を中断し、プログラムの制御の流れやデータの内容を調べることができます。



## 例

250行でプログラムを止めて、データAが2であることを確認します。

## COBOL ソースプログラム

```

100 01 A PIC 9 VALUE 0.
 :
200 PROCEDURE DIVISION.

210 MOVE 2 TO A.
 :
250 ADD 1 TO B.

```

## デバッガの操作手順

| メニューコマンド                    | ラインコマンド   |
|-----------------------------|-----------|
| 中断点                         | BREAK 250 |
| 実行                          | CONTINUE  |
| [プログラムが実行され<br>250行目で中断します] |           |
| データの表示/変更 LIST A            |           |
| [データAの内容を確認します]             |           |
| [次のデバッグ操作へ]                 |           |

### 20.5.2.1.2 プログラムの実行順序を変更する

デバッガでは、誤りのある文を迂回したり、本来の実行順序ではない文へプログラムの制御が移ってしまったものを本来の実行順序の文へ制御を戻したりすることができます。このようにプログラムの実行順序を変えるためには、実行開始位置変更機能を使います。



例

220行を迂回し、230行からプログラムを実行させます。

## COBOL ソースプログラム

```

100 01 A PIC 9 VALUE 0.
 :
150 PROCEDURE DIVISION.
 :
210 MOVE 1 TO A.
220 CALL "SUB".
230 MOVE A TO B.
 :
500 IDENTIFICATION DIVISION.
510 PROGRAM-ID. SUB.
 :

```

## デバッガの操作手順

| メニューコマンド                     | ラインコマンド   |
|------------------------------|-----------|
| 中断点                          | BREAK 220 |
| 実行                           | CONTINUE  |
| [プログラムが実行され<br>220 行目で中断します] |           |
| 実行開始位置を<br>カーソル位置へ変更         |           |
| [220 行のCALLが迂回されます]          |           |
| 実行                           | CONTINUE  |
| [次のデバッグ操作へ]                  |           |

### 20.5.2.1.3 データの変更箇所を調べる

ある時点でデータの値が正しくないことがわかって、そのデータの内容がプログラムのどこで変更されたかわからない場合があります。そのような場合、データの内容をつねに監視し、値が変更されたときにプログラムを中断させ、原因を調査することができます。

データに着目してプログラムをデバッグするときには、データ監視機能を使用します。データ監視機能では、指定したデータの値が変更されたときに、プログラムの実行を中断します。



例

データ監視機能を使い、データAの変更を監視します。

## COBOL ソースプログラム

```

100 01 A PIC 9 VALUE 0.
110 01 B PIC 9.
 :
200 PROCEDURE DIVISION.
 :
270 KEISAN.
280 CALL "SUB".
290 COMPUTE A = B + 1.
300 COMPUTE B = B + 1.
 :

```

## デバッガの操作手順

| メニューコマンド                               | ラインコマンド   |
|----------------------------------------|-----------|
| データ監視                                  | MONITOR A |
| 実行                                     | CONTINUE  |
| [プログラムが実行されデータAに値が設定された直後の300行目で中断します] |           |
| [データAの内容を確認し、必要に応じて修正します]              |           |
| [次のデバッグ操作へ]                            |           |

### 20.5.2.1.4 実行経路をたどる

現在中断している文までのプログラムの実行経路を知りたいときには、逆トレース機能を使用します。逆トレース機能では、逆トレース位置の文に色が付くので、この文の色が移動することにより、プログラムの制御の流れを視覚的に確認することができます。

逆トレース機能を使用するためには、対象となる一連の処理を実行する際にトレース情報を採取しておく必要があります。トレース情報を採取するには、トレース採取機能を使用します。

逆トレース機能には、以下の種類があります。

- ・ 文単位で実行順序と逆方向にたどる
- ・ 文単位で実行順序と同方向にたどる
- ・ 手続き単位で実行順序と逆方向にたどる
- ・ 手続き単位で実行順序と同方向にたどる



#### 例

手続きPRMがどこから呼び出されたか、呼出し元を確認します。

## COBOL ソースプログラム

```

100 01 A PIC 9 VALUE 0.
 :
200 PROCEDURE DIVISION.
210 MOVE 1 TO A.
220 PERFORM PRM.
 :
250 PRM.
260 ADD 1 TO A.
270 PRM-EXIT.
 :

```

## デバッガの操作手順

| メニューコマンド                   | ラインコマンド   |
|----------------------------|-----------|
| トレース採取                     | 対応コマンドなし  |
| 中断点                        | BREAK 260 |
| 実行                         | CONTINUE  |
| [プログラムが実行され260行目で中断します]    |           |
| 逆トレース-前の文                  | 対応コマンドなし  |
| [220 行目に逆トレース位置を示す色が付きます]  |           |
| [この時、A の値は+1されたままです]       |           |
| 逆トレース-前の文                  | 対応コマンドなし  |
| [210 行目に逆トレース位置を示す色が移動します] |           |
| [次のデバッグ操作へ]                |           |



### 20.5.2.1.5 プログラムの呼出し経路をたどる

連絡節のデータが正しくない場合や、プログラムの呼出しのタイミングがおかしい場合に、呼出し元のプログラムを調べることができます。プログラムの呼出し経路を知るには、呼出し経路表示を使用します。



例

プログラムSUBがどこから呼び出されたか、呼出し元を確認します。

#### COBOL ソースプログラム

```
100 01 A P/C 9 VALUE 0.
 :
200 PROCEDURE DIVISION.
 :
250 CALL "SUB" USING A.
 :
360 IDENTIFICATION DIVISION.
350 PROGRAM-ID. SUB.
 :
410 LINKAGE SECTION.
420 01 X P/C 9.
 :
480 PROCEDURE DIVISION USING X.
490 ADD X TO Y.
 :
```

#### デバッガの操作手順

| メニューコマンド                | ラインコマンド   |
|-------------------------|-----------|
| 中断点                     | BREAK 490 |
| 実行                      | CONTINUE  |
| [プログラムが実行され 490行で中断します] |           |
| データの表示/変更               | LIST X    |
| [データXの内容を確認します]         |           |
| 呼出し経路                   | CALLS     |
| [呼出し元のプログラムを調べます]       |           |
| [次のデバッグ操作へ]             |           |

### 20.5.2.1.6 デバッガの状態を知る

デバッグ操作で指定した各種状態(中断点の設定状況、通過カウント点の設定状況、データ監視の状況、条件監視の状況)を表示して確認することができます。これらの状態を表示するには、各機能の設定や解除を行うダイアログやウィンドウを使用します。

中断点の設定状況は、[中断点]ダイアログに、設定されている中断点の一覧および中断点の状態(設定条件および保留されているか、有効か)を表示します。

通過カウント点の設定状況では、[通過カウント点]ダイアログに、設定されている通過カウント点の一覧およびそれぞれの通過回数を表示します。

データ監視の状況では、監視ウィンドウに、つねに監視中のデータ名およびそのデータ内容を表示します。

条件監視の状況では、監視ウィンドウに、つねに監視中の条件式およびその状態を表示します。

また、現在の中断位置を確認したいときには、現在位置表示機能を使用します。ソースファイルウィンドウ内の現在の中断位置に、カーソルが位置付けられます。



例

プログラムの中断点を調べます。

## COBOL ソースプログラム

```
100 01 A PIC 9 VALUE 0.
 :
200 PROCEDURE DIVISION.
210 MOVE 1 TO A.
220 PERFORM PRM.
230 STOP RUN.
 :
270 PRM.
280 ADD 1 TO A.
290 SUB-EXIT.
```

## デバッガの操作手順

```
メニューコマンド ラインコマンド
中断点 (条件指定) BREAK 280 COND(A>0)
中断点 BREAK 230
実行 CONTINUE
[プログラムが実行され
 280行目で中断します]
中断点 STATUS BREAK
[次の2個の中断点の一覧が表示され、
230 行は条件なし、
280 行は条件ありとわかります]
[次のデバッグ操作へ]
```

## 20.5.2.2 デバッグ操作の自動化

デバッグ操作を再現するときや、同一のデバッグ処理手順を頻繁に行うときには、コマンドを格納したコマンドファイルを使ってデバッグ作業を自動化することができます。デバッグ操作の自動化は、デバッグ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。以下にバッチデバッグおよび自動デバッグについて説明します。

### バッチデバッグ

デバッグの開始時に、コマンドファイルを指定する形態です。デバッグ開始後は、利用者はとくに指示する必要はありません。

### 自動デバッグ

自動デバッグでは、デバッグ中にコマンドファイルを読み込み、コマンドファイル内のサブコマンドを実行することができます。自動デバッグは、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したりするために、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。

#### 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、デバッガの起動パラメタ、[履歴出力]ダイアログまたはENVコマンドで指定することにより作成されます。

##### 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。以下に例を示します。



### 例

プログラムを中断し、データの値を表示する場合のコマンドファイルの内容

```
break 47 ; 中断点設定
list data-out ; データ表示
cont ; 無条件実行
list data-out ; データ表示
cont ; 無条件実行
quit ; 終了
```

セミコロン(;)から行末の記述はコメントとして読み飛ばされます。

### 20.5.2.3 その他の補助的な機能

デバグには補助的な機能として、以下の機能が含まれています。

#### 実行中断

プログラムがループしている場合などで、プログラムの実行を強制的に中断することができます。

#### 再デバグ

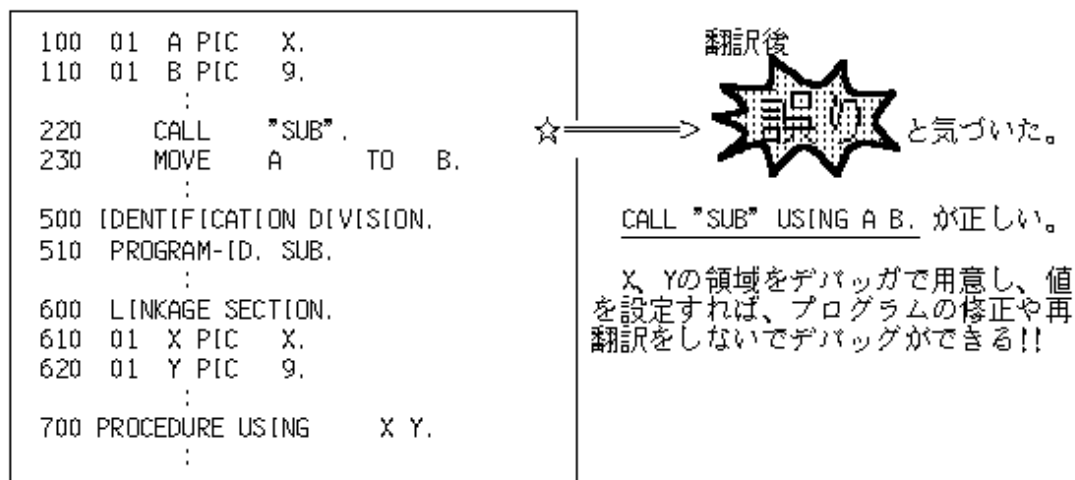
現在デバグ中の利用者プログラムを強制的に終了させ、デバグを終了しないで、同じプログラムのデバグを再びスタートすることができます。

再デバグさせる状態として、以下の指定を行うことができます。

- ・ 中断点を保持したままとするか、解除するか。
- ・ 通過カウント点を保持したままとするか、解除するか。
- ・ データ監視を保持したままとするか、解除するか。
- ・ 条件監視を保持したままとするか、解除するか。

#### 連絡節獲得

副プログラムのデバグ時に、呼出し元のプログラムから渡される連絡節のデータ領域を獲得できます。たとえば、呼出し元のプログラムで誤ったパラメタの個数を記述してしまったときなどに、呼出し元のプログラムの修正や再翻訳なしに副プログラムのデバグが可能です。獲得した領域は、副プログラムから呼出し元のプログラムに制御が戻るときにデバグによって解放されます。



#### デバグ環境の変更

デバグでは、以下の環境を変更することができます。

なお、文識別番号モードと置換文字以外の環境は、次回デバグを起動した時にも有効です。

##### 追尾速度

追尾実行する場合の実行速度。

##### 呼出し先プログラムを表示

逆トレースおよび追尾実行を現在実行中のプログラムだけで行うか、呼出し先プログラムも含めて行うか。

##### すべてのスレッドを表示

逆トレースおよび追尾実行を暗黙スレッドだけで行うか、すべてのスレッドを含めて行うか。

## PERFORMレベルの監視を行う

PERFORMレベルの監視を行うか、行わないか。

監視を行う場合、そとPERFORMエラーを検出し、逆トレースおよび追尾実行で表示しないPERFORM文のネストレベルの指定を有効にする。

## 例外事象が発生したら中断する

例外条件が発生した場合に、例外手続きが処理される前で中断するか、中断しないか。[参照]“[18.1 例外処理](#)”

## 初回のシグナル受信時に中断する

初回のシグナルを受信した場合に実行を中断するか、中断しないか。[参照]“[20.6.2 例外ハンドラで処理されるシグナルのデバッグ方法](#)”

## スレッド終了時に中断する

スレッドが終了したときに実行を中断するか、中断しないか。

## ラインコマンドにおける大文字/小文字の比較基準

ラインコマンドにおいて、大文字/小文字を区別するか、区別しないか。

## 文識別番号モード

文識別番号をエディタ行番号として指定するか、ファイル内相対行番号として指定するか。

## 行番号を表示する

ソースファイルウィンドウに、行番号を表示するか、表示しないか。

## 追尾実行中のウィンドウ更新

追尾実行中に、呼出し経路ウィンドウおよびスレッド一覧ウィンドウの表示をそれぞれ更新するか、更新しないか。

## 置換文字

データの表示で表示できない文字を置き換える文字。

## 中断時のデバッグ状態

実行が中断したときに、デバッグをアクティブにするかしないか。

## バッチデバッグ時に開くウィンドウ指定

バッチデバッグ時に、メインウィンドウ内の各ウィンドウを開くか開かないか。

## 中断時の通知

実行が中断したときに、音を鳴らすか鳴らさないか。

## バッチデバッグでの中断時の通知

バッチデバッグの場合に、実行が中断したときに音を鳴らすか鳴らさないか。

## 互換性

デバッグの機能範囲を特定のバージョンの範囲に限定するか、限定しないか。

## データのファイル入出力

データのファイル入出力機能を使用して、データ値をファイルから読み込ませたり、ファイルに書き込ませたりすることができます。

以下に、各機能の概要について説明します。

### ファイルからの読み込み

この機能は、ファイルから読み込んだ値をデータ項目に設定するものです。事前に設定する値をファイルに格納しておけるので、データの変更(SETコマンド)に比べ、多くの値を設定する場合に適しています。

具体的な用途には、処理ルートの確認を行う場合があります。処理ルートの確認には、通常各ルートを通すための値を設定する必要があります。この値の設定にファイルからの読み込みを使用すると、テスト中に値を入力する作業がないので、処理ルートの確認が容易になります。

## ファイルへの書込み

この機能は、データ項目の値をファイルに書き込むものです。ファイルから値をまとめて参照できるので、データの表示(LISTコマンド)に比べ、多くの値を確認する場合に適しています。

具体的な用途には、処理ルートや入力データによって異なる処理結果を確認する場合があります。ファイルへの書込みを使用すると、テスト中は処理ルートの確認を中心に作業し、テスト終了後は処理結果の確認を中心に作業できます。

なお、データのファイル入出力機能で使用するデータファイルの作成、確認にはSIMPLIA/TF-LINDAを利用すると便利です。



### 例

副プログラムの処理ルート確認を行い、パラメタの返却値をデバッグ終了後にまとめて確認します。

#### 1. 読み込み用ファイルの作成

連絡節のデータ領域(LINK)を1レコードとして、各ルートを通るための値をデータ項目(A)に設定したINFLを作成します。

#### 2. デバッグ

副プログラムの入口で、INFLから連絡節のデータ領域(LINK)に値を読み込んでルートを確認します。

副プログラムの出口で、連絡節のデータ領域(LINK)の値をOUTFLに書き込みます。

#### 3. 書き込んだファイルの確認

副プログラムで変更した連絡節のデータ領域(LINK)の値をOUTFLからまとめて確認します。

#### [副プログラム]

```

 :
100 LINKAGE SECTION.
110 COPY LINKCPY.
200 PROCEDURE DIVISION
210 USING LINK.
220 DISPLAY "START".
230 EVALUATE A
240 WHEN 1 MOVE 11 TO B
250 WHEN 2 MOVE 12 TO B
 :
300 END-EVALUATE.
310 EXIT PROGRAM.

```

#### [COPY句 LINKCPY]

```

100 01 LINK.
110 02 A PIC 9.
120 02 B PIC 99.

```

#### [ファイルの内容](COPY句 LINKCPY)

|     | INFL |   | OUTFL |    |
|-----|------|---|-------|----|
|     | A    | B | A     | B  |
| (1) | 1    |   | 1     | 11 |
| (2) | 2    |   | 2     | 12 |
| ⋮   | ⋮    | ⋮ | ⋮     | ⋮  |
| (n) |      |   |       |    |

#### [デバッグ]

| メニューコマンド               | ラインコマンド                  |
|------------------------|--------------------------|
| [1] 中断点                | BREAK 220                |
| [2] 実行                 | CONTINUE                 |
| [3] (追加)               | ASSIGNDATA LA INFL LINK  |
| [4] (追加)               | ASSIGNDATA LB OUTFL LINK |
| [5] (オフ)               | OPENDATA INPUT LA        |
| [6] (オフ)               | OPENDATA OUTPUT LB       |
| [7] (読み込み)             | READDATA LA              |
| [8] [ ルートを確認するデバッグ操作 ] |                          |
| [9] (書き込み)             | WRITEDATA LB             |
| [10] 実行                | CONTINUE                 |
| [11] (オフ)              | CLOSEDATA LA             |
| [12] (オフ)              | CLOSEDATA LB             |

([3]-[7]、[9]、[11]、[12]のメニューコマンドは、データのファイル入出力)

[1]-[2] 220 行目で中断する  
 [3]-[6] ファイルの入出力のための準備を行う  
 [7]-[10] INFLのn件数分繰り返し行う  
 [7] LINKの値をINFLから読み込む  
 [8] 処理ルートの確認を行って、310 行目で中断させる  
 [9] LINKの値をOUTFLに書き込む  
 [10] 220 行目で中断する  
 [11]-[12] INFL、OUTFLを閉じる

## 20.6 プログラムの特徴に合わせたデバッグ方法

ここでは、プログラムの特徴に合わせたデバッグ方法を説明します。

### 20.6.1 動的構造に対するデバッグ方法

動的構造のプログラムをデバッグする場合、静的構造のプログラムと同じ操作方法でデバッグできます。ただし、以下の注意が必要です。[参照]“[4.3.2 動的構造](#)”

#### 動的リンク構造

動的リンク構造の場合、呼出し元のプログラムがロードされたときに、DLLもロードされます。このため、DLLもデバッグの対象となり、静的構造のプログラムと同じ操作方法でデバッグできます。

#### 動的プログラム構造

動的プログラム構造の場合、DLLは呼出し元から呼び出されたときにロードされます。DLLはロードされた時点から、デバッグの対象になります。

DLLに含まれるプログラムに中断点を設定したい場合、DLLが呼び出される前に中断点の設定を指示しておく必要があります。中断点を設定する場合、デバッガではロードされていないプログラムのソースファイルを表示できないため、コマンドやダイアログボックスで、ロードされるプログラムの名前を指定して指示します。なお、設定した中断点は解除しないかぎり、プログラムがアンロードされた後に再度ロードされたときも有効になります。

たとえば、PROG05という名前のプログラムの行番号5000に中断点を設定する場合、コマンドによる指示は以下のようになります。

```
BREAK 5000 IN(PROG05)
```

### 20.6.2 例外ハンドラで処理されるシグナルのデバッグ方法

アプリケーションの実行によって発生する不正なメモリアクセスやゼロ除算などのアプリケーションエラー(例外)を、デバッガではシグナルと呼びます。

シグナルは、不正な処理を行うことによって発生しますが、シグナルを発生させるシステムAPIを呼び出すことでも発生させることができます。発生したシグナルに対する処理は、例外ハンドラとして記述することができます。これらのシステムの機構は、構造化例外処理と呼ばれています。

COBOLプログラムの呼出し元が、例外ハンドラを用意している場合、COBOLプログラムで発生するアプリケーションエラーもシグナルのひとつとして処理されるため、不正な処理を行ったことがわからない場合があります。たとえば、Webサーバなどのサーバプログラムでは、アプリケーションが不正な処理を行った場合に、サーバプログラム自身に悪影響が及ばないようにするため、例外ハンドラでシグナルを処理していることがあります。

デバッガでは、このような例外ハンドラで処理されるシグナルも通知できるように対応しています。シグナルの通知は以下の2つのタイミングで行います。

#### 初回のシグナル受信

例外ハンドラの有無にかかわらず、システムが例外ハンドラを検索する前のタイミングで通知します。このタイミングでの通知は、例外ハンドラで処理されるシグナルを知るために用意されています。

#### シグナル受信

例外ハンドラが用意されていない場合または例外ハンドラでシグナルが処理されなかった場合に、シグナルを受信し、通知します。例外ハンドラでシグナルが処理された場合は、通知されません。

初回のシグナル受信時に、アプリケーションの実行を中断するかどうかは、[環境設定]ダイアログの[動作]ページで設定します。シグナル受信時は、必ずアプリケーションの実行を中断します。

シグナルは、問題となるシグナル(例外で発生したシグナル)ばかりではありません。シグナルを発生させるシステムAPIの呼び出しで発生したシグナルや、例外ハンドラで処理されることで続行できる場合もあるため、初回のシグナル受信で通知されるシグナルが問題となるシグナルでないこともあります。問題となるシグナルかどうかは、シグナルのコードや発生した位置などから判断する必要があります。したがって、通常は、初回のシグナル受信では中断しないように設定しておき、問題となるシグナルが例外ハンドラで処理されていると思われる場合に、初回のシグナル受信で中断するように設定することをおすすめします。

なお、シグナル受信による中断が発生した後に、アプリケーションの実行を行うと、シグナルを発生させたマシン命令から再度実行するため、再度シグナルが発生します。

## 20.6.3 オブジェクト指向に対応したデバッグ機能

デバッガでは、ファクトリデータおよびオブジェクトデータを柔軟に参照および変更できるように、一意名にデバッガ独自に拡張した指定形式を用意しています。

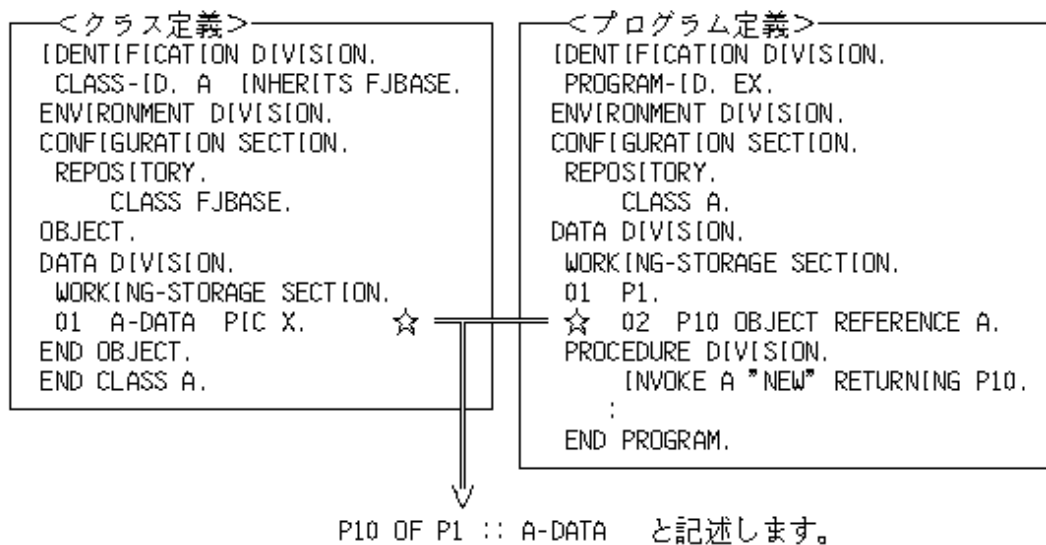
拡張した指定形式であるオブジェクト参照修飾子を用いた一意名を使用することにより、通常はプログラムから参照できないファクトリデータやオブジェクトデータを参照することができます。

オブジェクト参照修飾子を用いた一意名は、以下のように記述します。



例

クラスAのオブジェクトデータを参照する場合



また、この一意名は[従属項目の参照]ダイアログを使って対話的に指定することもできます。

一意名の詳細および[従属項目の参照]ダイアログの使い方については、ヘルプを参照してください。

## 20.6.4 マルチスレッドでのデバッガの対応

ここでは、マルチスレッドのプログラムをデバッグする場合のデバッガの対応について説明します。

### 20.6.4.1 デバッグ機能のマルチスレッドでの基本的な動作

被デバッグプログラムのプロセス内に複数のスレッドが動作している場合は、基本的に、複数スレッドの中の、1つのスレッドに対してだけデバッグ操作を行うことができます。このデバッグ操作が可能なスレッドを暗黙スレッドと呼び、暗黙スレッドを切り替えることで任意のスレッドに対するデバッグ操作が可能になります。暗黙スレッドの切り替えは、スレッド一覧ウィンドウから行うことができます。

デバッガのウィンドウやダイアログには、暗黙スレッドを対象とした内容が表示されるため、マルチスレッドのプログラムでもシングルスレッドの場合と同様の操作でデバッグできます。

システムやサーバアプリケーションなどによっては、COBOLプログラムを実行するスレッド以外のスレッドが存在する場合があります。このようなアプリケーションをデバッグする場合は、マルチスレッドアプリケーションと同様の注意が必要です。

### 20.6.4.2 デバッグ機能のマルチスレッド特有な動作

各デバッグ機能は、基本的に暗黙スレッドを対象にしてシングルスレッドと同様に動作しますが、以下の機能についてはマルチスレッド特有の動作になります。



## 中断点

中断点は、すべてのスレッドに対し有効になります。したがって、すべてのスレッドが中断点を指定した位置でそれぞれ中断します。

なお、通過回数のカウントはスレッドごとに行われます。また、通過回数のカウントは、スレッドが終了した後の実行によってクリアされます。

## 通過カウント点

通過カウント点は、すべてのスレッドに対し有効になります。

通過回数のカウントはスレッドごとに行われます。なお、通過回数のカウントは、スレッドが終了した時まで保持されますが、スレッドが終了した後の実行によってクリアされ0になります。そのため、暗黙スレッド以外のスレッドが終了した時点の通過回数を確認したい場合は、[環境設定]ダイアログの[動作]ページでスレッド終了時に実行を中断するように指定してください。

## 実行

無条件実行した場合は、すべてのスレッドがそれぞれ同時に実行されます。

中断点指定実行した場合は、暗黙スレッドが指定された条件に従って実行されます。このとき、暗黙スレッド以外のスレッドは無条件実行されます。

## データ監視/条件監視

データ監視機能や条件監視機能は、スレッド間共有データを監視している場合には、監視しているデータがどのスレッドから変更された場合でも、監視が成立したことを検出します。

また、スレッド間共有でないデータを監視している場合に、そのデータが別のスレッドから変更されたような場合にも監視が成立したことを検出します。

## 逆トレース

逆トレース機能は、暗黙スレッドだけで逆トレースするか、すべてのスレッドを対象に逆トレースするかを選択できます。

複数のスレッドを対象に逆トレースした場合は、スレッド間で実行された順序に従ってたどられ、ソースファイルウィンドウには現在たどっているところのスレッドが自動的に表示されます。これによって、スレッド間の制御の流れを視覚的に確認することができます。

## 20.6.4.3 マルチスレッドのデバッグを補助する機能

マルチスレッドのプログラムをデバッグする場合、以下のデバッグ機能を利用すると効率的なデバッグを行うことができます。

### デバッグ事象を通知する/通知しないの切り替え

デバッグ事象を通知する/通知しないの切り替え機能は、監視や中断点などによるプログラムの中断などのデバッグ事象を通知するかどうかを切り替えることができます。デバッグ事象を通知するかどうかは、スレッドごとに指定することができます。

プログラムを実行した時に、デバッグ事象を通知しない状態のスレッドは、実行はされますが、監視の成立や中断点への到達などのデバッグのための中断事象は通知しくなります。

この機能は、スレッド一覧ウィンドウから指定できます。

### サスペンド/レジューム

サスペンド/レジューム機能は、プログラムを実行した際のスレッドの動作を制限することができます。

サスペンドされたスレッドは、プログラムが実行されてもサスペンドされた時の位置で止まったままになります。サスペンドされたスレッドをレジュームすると、次にプログラムが実行された時にはスレッドが動作するようになります。

この機能は、スレッド一覧ウィンドウから指定できます。

## 20.6.4.4 マルチスレッドでのデバッグ操作の自動化

デバッガでは、バッチデバッグおよび自動デバッグを使用することでデバッグ操作の再現を行うことができます。

マルチスレッドで作成されたプログラムのデバッグでは、これらの機能を使用することでデバッグの再現を行うことができますが、マルチスレッドのプログラムに対するデバッグ操作を再現させるためには以下の点に注意する必要があります。



スレッドが生成される順番を、デバッグ操作を記録した時と同じにする

デバッガでは、スレッドを一意に識別するために、プロセス内でスレッドが生成された順番をもとにした値をスレッドIDとして使用しています。デバッグ操作がどのスレッドに対する操作なのかは、このスレッドIDを基準にして動作するため、デバッグ操作を再現する際には、スレッドが生成される順番を、デバッグ操作を記録した時と同じにする必要があります。スレッドがシステムやサーバアプリケーションなどによって生成される場合には、スレッドが順番どおりに生成されないことがあるため、自動化を行うことができません。

プログラムの実行時にはデバッグ事象を通知するスレッドはひとつだけにする

複数のスレッドを同時に実行させた場合、OSが個々のスレッドに対してどのようにCPUを割り当てるかによって、デバッグ事象を通知するスレッドが操作を記録した時と操作を再現した時で変わってくることがあります。無条件実行や中断点指定実行でプログラムを実行させる際は、暗黙スレッド以外でデバッグ事象を通知する可能性があるスレッドはサスペンドしておくことで、この問題は回避できます。

## 20.6.5 デバッガのUnicode対応

---

ここでは、Unicodeで翻訳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。

### 20.6.5.1 Unicodeプログラムのデバッグ方法

デバッガでは、デバッグ対象プログラムがUnicodeである場合には、自動的にUnicodeを扱うようになります。

デバッガは、Unicodeプログラムについても、基本的にはUnicodeでないプログラムの場合と同様の操作でデバッグできます。

### 20.6.5.2 Unicodeを扱う上での注意事項

Unicodeプログラムをデバッグする際には、以下の点に注意してください。

Unicode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固有の文字も表示されます。ただし、Unicode固有の文字を入力することはできません。Unicode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。

データ値の表示/変更

日本語項目、日本語編集項目を16進形式で操作する場合のデータ値の表現形式は、ビッグエンディアンです。

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、Unicodeの場合もシフトJISです。そのため、Unicode固有の文字は、“\_”に置き換えて出力されます。

## 20.7 注意事項

---

ここでは、対話型デバッガ使用時の注意事項について説明します。

- デバッグするアプリケーション

デバッグするアプリケーションは、32ビット用アプリケーションでなければなりません。

- アプリケーションのウィンドウ

デバッガは、アプリケーションの実行を中断させるときに、アプリケーションのプロセスを停止させます。したがって、この状態では、アプリケーションのウィンドウやアイコンの上に重なったウィンドウを移動してもウィンドウやアイコンは再描画されません。そのため、アプリケーションのウィンドウへの出力内容を確認するには、デバッガや他のプログラムのウィンドウがアプリケーションのウィンドウの上に重ならないようにウィンドウを配置してください。

- 日本語空白

デバッガは日本語空白をつねに翻訳オプションNSPCOMP(NSP)として比較します。翻訳オプションNSPCOMP(NSP)については、“[A.2.34 NSPCOMP\(日本語空白の比較方法の指定\)](#)”を参照してください。

- ラインコマンド入力ウィンドウの出力

ラインコマンド入力ウィンドウ内のラインコマンド出力領域には、約5,000行までの実行結果が保持されます。

- 文識別番号の指定

翻訳オプションNUMBERを指定したプログラムに対する文識別番号の指定は、エディタ行番号とファイル内相対行番号のどちらでもできます。翻訳オプションNONNUMBERを指定したプログラムに対しては、ファイル内相対行番号でだけ指定できます。[参照] “A.2.35 NUMBER (ソースプログラムの一連番号領域の指定)”

文識別番号をエディタ行番号とファイル内相対行番号のどちらで解釈するかは、環境設定機能で指定します。環境設定機能については、ヘルプを参照してください。

なお、翻訳オプションNUMBERを指定して翻訳した一連番号領域に同じ番号が連続して書かれているプログラムに対して、文識別番号の指定がエディタ行番号のとき、意図した位置に中断点が設定できないなどの問題が発生する場合があります。この問題を回避するには、文識別番号の指定をファイル内相対行番号に変更するか、翻訳オプションNONNUMBERを指定して再度翻訳してから、デバッグを行うようにしてください。



#### 例

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. MAIN.
 :
001000 PROCEDURE DIVISION.
001100 DISPLAY "*** INITIALIZE ***". ... [1]
001100 MOVE ZERO TO DATA-1. ... [2]
001100 MOVE ZERO TO DATA-2. ... [3]
 :
003600 END PROGRAM MAIN.
```

この場合、MOVE文([2]および[3])に中断点を設定しようとしても、DISPLAY文([1])に中断点が設定されてしまいます。

- 埋込みSQL文COBOLプログラム

PROCEDURE DIVISIONに記述された“EXEC SQL”に中断点や通過カウント点を設定できます。ただし、非実行文(WHENEVER文など)を含む“EXEC SQL”には中断点や通過カウント点を設定できません。

また、SQL文に中断点や通過カウント点を直接設定することはできません。

- ソースプログラムの表示

ソースプログラム中にREPLACE文、REPLACING指定、DISJOINING指定またはJOINING指定のCOPY文が記述されている場合、画面には文字列が置き換えられる前のソースプログラムを表示します。置き換える前と後で文や手続き名の位置が変わる場合には、正しくデバッグできません。

- COPY文プログラムの制限

以下の場合、プログラムを正しくデバッグできません。それぞれについて、回避方法を示します。

- 以下を含む登録集原文をCOPY文で取り込んでいる場合

- プログラム定義全体(IDENTIFICATION DIVISIONからEND PROGRAMまで)
- ファクトリ定義全体(IDENTIFICATION DIVISIONからEND FACTORYまで)
- オブジェクト定義全体(IDENTIFICATION DIVISIONからEND OBJECTまで)
- メソッド定義全体(IDENTIFICATION DIVISIONからEND METHODまで)

回避方法として、上記の登録集原文に記述された文を、登録集原文を取り込んでいるプログラムに直接記述します。

- 手続き部にCOPY文を記述し、かつ、取り込んでいる登録集原文中の最後の行がそとPERFORM文である場合

回避方法として、登録集原文の最後にCONTINUE文などを記述します。

- 手続き部にCOPY文を記述し、かつ、手続き部の最後の行がCOPY文であり、かつ、END PROGRAM文の記述がない場合

回避方法として、COPY文を記述しているプログラムにEND PROGRAM文を記述します。

- ー 手続き部にCOPY文を記述し、かつ、取り込んでいる登録集原文中の最後の行がそとPERFORM文であり、かつ、COPY文の直後がそとPERFORM 文である場合

回避方法として、登録集原文の最後とCOPY文の直後にCONTINUE文などを記述します。

- ー 手続き部にCOPY文を記述し、かつ、EXIT文、または、EXIT PROGRAM文、EXIT METHOD文の直後がCOPY文であり、かつ、取り込んでいる登録集原文の最初の行が節名または段落名である場合

回避方法として、節名または段落名を登録集原文の最初の行ではなく、登録集原文を取り込んでいるプログラムに記述します。

- ・ ストアドプロシージャはデバッグできません。
- ・ デバッグを使用しているときに、以下の操作を行わないでください。この操作を行うと、システムが応答なくなります。操作を行う前にデバッグを終了させてください。

- ー [画面のプロパティ]で画面の設定を変更する。

- ー コントロールパネルから設定を変更する。

- ・ アプリケーションを実行しているときに、アプリケーションのウィンドウでF12キーを押さないでください。この操作を行うと、シグナルが発生して実行が中断し、実行を再開させるとアプリケーションが終了します。
- ・ デバッグ中に、アプリケーションの実行がCOBOLランタイムシステム内部やシステム内部で中断している状態でファイルやデータベースなどの資源にアクセスしている最中であったとき、デバッグを終了させたり、再デバッグを行ったりすると、資源が破壊される可能性があります。また、COBOLランタイムシステム内部で同期処理をしている最中であったとき、デバッグを終了させたり、再デバッグを行ったりすると、JMP0034I-Uなどの実行時メッセージが出力される可能性があります。マルチスレッドで動作するアプリケーションをデバッグ中の場合は、スレッドの数が増加するにしたがってその可能性が増加します。

アプリケーションの実行を正常に終了させるか、万一の場合に備え、デバッグで使用する資源は、事前にバックアップを用意しておいてください。

- ・ システムやサーバアプリケーションなどによっては、COBOLプログラムを実行するスレッド以外のスレッドが存在する場合があります。このようなアプリケーションをデバッグする場合は、マルチスレッドアプリケーションと同様の注意が必要です。
- ・ 対話型デバッグは、アプリケーションをPowerCOBOL デバッグまたはVisual C++デバッグなどでデバッグしている場合や、Systemwalkerの下で実行している場合には、使用できません。
- ・ ジャストインタイムデバッグ機能は、アプリケーション自身がアプリケーションエラーを構造化例外処理の例外ハンドラによって処理している場合には、機能しません。たとえば、例外ハンドラを持つCOBOL以外の言語のプログラムと言語間結合している場合や例外ハンドラを持つWebサーバなどのサーバプログラムから呼び出された場合が該当します。これらのアプリケーションで発生するアプリケーションエラーの調査には、一般的なプログラムに対するデバッグ起動方法かデバッグしたいプログラムからのデバッグ起動方法を使用してください。

- ・ 「開始プログラムに到達する前に実行が終了しました。」が表示された場合の対処方法

「開始プログラムに到達する前に実行が終了しました。」というメッセージが表示された場合の対処方法を以下に示します。

- ー 開始プログラムに指定したプログラムが実行されずにプログラムが終了した。または、指定したプログラム名が間違っている。

開始プログラム名に正しいプログラム名を指定してください。

- ー プログラムがデバッグできるプログラムになっていない。

“[20.2 デバッグの準備](#)”を参照してプログラムを作成しなおしてください。

- ー デバッグ情報ファイル格納フォルダで指定したフォルダにデバッグ情報ファイルが存在しない。

デバッグ情報ファイルは、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が格納されているフォルダを検索しますが、それ以外の場所にデバッグ情報ファイルが存在する場合は、デバッグ情報ファイル格納フォルダを指定してください。

- ー デバッグがJEFオプション製品を使用して作成したプログラムをデバッグするモードになっていない。

JEFで作成されたプログラムをデバッグする場合には、プロジェクトマネージャウィンドウの[環境]メニューの[文字コード]で“JEF”を選択してください。

## 20.8 デバッグのコマンド一覧

デバッグのコマンド一覧を“[表20.2 デバッグのコマンド一覧](#)”に示します。

コマンドの指定形式の詳細については、ヘルプを参照してください。

表20.2 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        |

## 20.9 対話型リモートデバッガの使い方

対話型リモートデバッガ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。

ここでは、以下の用語を使用します。

## IPアドレス

特に断りがない限り、「IPアドレスまたはホスト名」を表します。

## Windows

特に断りがない限り、「32ビットWindows」をあらわします。

### 20.9.1 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ことができます。

なお、ここでは、ネットワーク上の2台のコンピュータを以下のように表現します。

#### COBOLプログラムが動作しているコンピュータ

サーバ

#### COBOLデバッグのウィンドウが表示されるコンピュータ

クライアント

リモートデバッグでは、サーバ側のOSとしてWindows、Windows(Itanium)、SolarisおよびLinuxをサポートしています。

また、リモートデバッグの機能は、対応するサーバ側OSによって一部異なるため、各リモートデバッグを区別する場合は、それぞれ以下のように呼びます。

- ・サーバがWindowsの場合、Win32リモートデバッグを使用
- ・サーバがWindows(Itanium)の場合、Win64リモートデバッグを使用
- ・サーバがSolarisの場合、Solarisリモートデバッグを使用
- ・サーバがLinuxの場合、Linuxリモートデバッグを使用

これ以降の説明では、サーバがWindowsの場合を中心に説明します。

サーバがWindows(Itanium)、SolarisまたはLinuxの場合には、必要に応じて、各OS版の“NetCOBOL 使用手引書”の関連する項目を参照してください。



#### 注意

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。

#### 20.9.1.1 リモートデバッグ導入による効果

リモートデバッグを導入することにより、以下のようなコスト削減効果および開発効率向上効果があります。

- ・InterstageやWebサーバなどのサーバ環境で動作するCOBOLプログラムのデバッグを、普段使用しているクライアント環境からデバッグできるようになります。

これにより、高価なサーバ環境を開発者ごとに用意しなくても、複数の開発者が同時にデバッグを行えるようになるため、コスト削減効果および開発効率向上効果があります。

- ・Windows(Itanium)製品のNetCOBOLにはWindows(Itanium)上で動作する対話型デバッグがありません。Win64リモートデバッグを使用することで、対話型リモートデバッグが可能になります。
- ・サーバがSolarisまたはLinuxの場合には、Windows版COBOLデバッグと同じ操作性でデバッグできるようになるという効果があります。

SolarisまたはLinux上で動作するプログラムを、Windowsクライアントを使ってデバッグするには、以下の2つの方法がありました。

- ー 高価なWindows用X ウィンドウサーバを導入して、svdコマンドのスクリーンモードを使用する。(Solarisの場合)
- ー TTY端末を使ってsvdコマンドのラインモードを使用する。

リモートデバッグは、高価なX ウィンドウサーバは不要ですし、Windows版 COBOLデバッグと同じ操作性で使用できるため、コスト削減効果および開発効率向上効果があります。

### 20.9.1.2 リモートデバッグ時の資産の格納場所

リモートデバッグ時には、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“表20.3 リモートデバッグ時の資産格納場所”に示します。

表20.3 リモートデバッグ時の資産格納場所

| デバッグ資産               | クライアント側 | サーバ側 |
|----------------------|---------|------|
| 実行可能プログラム            | —       | ○    |
| ダイナミックリンクライブラリ (*1)  | —       | ○    |
| 共用オブジェクトファイル (*2)    | —       | ○    |
| デバッグ情報ファイル           | —       | ○    |
| プログラムデータベースファイル (*3) | —       | ○    |
| ソースファイル              | ○ (*6)  | —    |
| 登録集原文                | ○ (*6)  | —    |
| 画面帳票定義体              | ○ (*6)  | ○    |
| ファイル定義体 (*4)         | ○ (*6)  | ○    |
| サブスキーマ定義 (*5)        | ○ (*6)  | —    |
| コマンドファイル             | ○ (*6)  | —    |
| 操作履歴ファイル             | ○ (*6)  | —    |

\*1：サーバ側がWindowsまたはWindows(Itanium)の場合

\*2：サーバ側がSolarisまたはLinuxの場合

\*3：サーバ側がWindows(Itanium)の場合

\*4：サーバ側がWindows、SolarisまたはLinuxの場合

\*5：サーバ側がWindowsの場合

\*6：クライアント側の資産のコード系は、サーバ側のロケールに関係なく、常に「シフトJIS」です。

### 20.9.1.3 リモートデバッグ時の被デバッグプログラムの作成方法

リモートデバッグでデバッグするためのプログラム作成方法については、サーバ側となる各システムの使用手引書を参照してください。  
Win64リモートデバッグの場合は、“NetCOBOL Studio 使用手引書”も合わせて参照してください。

## 20.9.2 リモートデバッグの開始手順

リモートデバッグを開始する方法には、通常のデバッグと同様に3つの方法があります。

#### 一般的なプログラムに対するリモートデバッグ起動方法

リモートデバッグを起動し、デバッグしたいプログラムを指定してデバッグを開始する起動方法です。

#### Interstageなどのサーバ環境で動作するプログラムに対するリモートデバッグ起動方法

デバッグしたいプログラムからリモートデバッグを起動し、デバッグを開始する起動方法です。

COBOLプログラムがサーバ製品から呼び出されるために、一般的なプログラムに対するリモートデバッグ方法は使えないため、この起動方法を使用します。

#### ジャストインタイムデバッグ機能を使用したリモートデバッグ起動方法

プログラムでアプリケーションエラーや実行時メッセージが発生した場合に、自動的にリモートデバッグが起動されてプログラムをデバッグできる起動方法です。

この起動方法はサーバがWindowsの場合にのみ使用できます。



## 20.9.2.1 一般的なプログラムに対するリモートデバッグ起動方法

リモートデバッグを起動し、デバッグしたいプログラムを指定してデバッグを開始する起動方法です。

一般的なプログラムをデバッグする場合には、あらかじめサーバ側でリモートデバッグコネクタを起動しておく必要があります。

サーバ側のコンピュータを直接操作するか、サーバ側のコンピュータにログオンして操作してください。サーバ側のコンピュータにログオンする場合は、TELNET、または、サーバがWindowsやWindows(Itanium)の場合にはターミナルサービスなどを使用します。

一般的なプログラムをデバッグする場合のリモートデバッグの起動手順を以下に示します。

### 1. サーバ側でリモートデバッグコネクタを起動します。

サーバ側でリモートデバッグコネクタを起動します。

この際、必要に応じて接続を許可するコンピュータを限定するための指定を行います。

詳しくは、“[20.9.3.1 サーバ側のリモートデバッグコネクタの使い方](#)”および、“[20.9.3.5 リモートデバッグを許可するコンピュータを制限する](#)”を参照してください。

リモートデバッグコネクタが起動されると、IPアドレスやポート番号などの情報が表示されて、クライアント側からのデバッグ開始の指示を監視します。

### 2. クライアント側でリモートデバッグを起動します。

一般的によく使われる方法を以下に示します。

#### ー SolarisリモートデバッグおよびLinuxリモートデバッグの場合

プロジェクトマネージャウィンドウの[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。

#### ー Win32リモートデバッグ、SolarisリモートデバッグおよびLinuxリモートデバッグの場合

プロジェクトマネージャウィンドウの[ツール]メニューの[デバッグ種別]で使用するデバッグを選択してから、以下のいずれかの方法で起動します。

- プロジェクトマネージャウィンドウの[ツール]メニューから“デバッグ”を選択します。
- コマンドプロンプトからコマンド形式を入力して起動します。[参照]“[20.9.5.2 起動コマンドを使ったリモートデバッグ指定](#)”

#### ー Win64リモートデバッグ、SolarisリモートデバッグおよびLinuxリモートデバッグの場合

NetCOBOL Studioの[構造]ビューまたは[依存]ビューでCOBOLプロジェクトを選択し、[プロジェクト]メニューから[リモート開発]、“デバッグ”の順に選択します。

### 3. [デバッグを開始する]ダイアログで必要な情報を指定します。

リモートデバッグが起動されると、デバッグのメインウィンドウが表示されます。起動と同時に[デバッグを開始する]ダイアログが出力されない場合は、リモートデバッグの[ファイル]メニューから“デバッグを開始する”を選択します。

[デバッグを開始する]ダイアログで、アプリケーション名、ソースファイル格納フォルダ、サーバ側のIPアドレスなどデバッグを開始するために必要な情報を入力して[OK]ボタンをクリックします。[参照]“[20.9.5 リモートデバッグの起動](#)”

### 4. リモートデバッグを使ってデバッグを行います。

[デバッグを開始する]ダイアログが閉じてデバッグが開始されます。

リモートデバッグの操作方法は通常のデバッグと同じです。使用方法については、“[20.5 デバッグの操作](#)”を参照してください。

なお、サーバ側のリモートデバッグコネクタが起動している間は、リモートデバッグでデバッグするプログラムを指定することで、何度でもデバッグを行うことができます。

### 5. リモートデバッグを終了させます。

[ファイル]メニューの“デバッグの終了”を選択することで、リモートデバッグを終了します。

### 6. サーバ側のリモートデバッグコネクタを終了させます。

CtrlキーとCキーを同時に押すことで、サーバ側のリモートデバッグコネクタが終了します。

## 20.9.2.2 Interstageなどのサーバ環境で動作するプログラムに対するリモートデバッグ起動方法

デバッグしたいプログラムからリモートデバッグを起動する方法です。

この起動方法は、主にInterstageやWebサーバなどのサーバ環境で動作するCOBOLプログラムをリモートデバッグする場合に使用します。

起動手順を以下に示します。

1. リモートデバッグを行うための設定を行います。

#### クライアント側の設定

クライアント側でリモートデバッグコネクタを起動します。

リモートデバッグコネクタは、前回起動したときの設定で起動され、タスクトレイに常駐します。

必要に応じて、ポート番号の変更や、接続を許可するコンピュータを限定するための指定を行います。

詳しくは、“[20.9.3.4 クライアント側のリモートデバッグコネクタの使い方](#)”および、“[20.9.3.5 リモートデバッグを許可するコンピュータを制限する](#)”を参照してください。

#### サーバ側の設定

クライアント側のリモートデバッグコネクタの[接続情報]ページで表示されるクライアント側のコンピュータのIPアドレスとポート番号を環境変数@CBR\_ATTACH\_TOOLに指定します。[参照]“[5.4.1.2 @CBR\\_ATTACH\\_TOOL \(プログラムからデバッグまたは診断機能を起動する指定\)](#)”

なお、複数の開発者が同時にデバッグを行えるようにするためには、@CBR\_ATTACH\_TOOLに開発者ごとに異なるIPアドレスとポート番号を指定する必要があります。

例えば、以下のようにすることで@CBR\_ATTACH\_TOOLに開発者ごとに異なるIPアドレスとポート番号を指定できるようになります。

1. 開発者ごとに専用のフォルダを用意して、デバッグしたい実行可能ファイルやダイナミックリンクライブラリと実行用の初期化ファイル(COBOL85.CBR)を格納します。
2. 実行用の初期化ファイル内で@CBR\_ATTACH\_TOOLを指定します。

なお、InterstageやWebサーバなどのサーバ製品から呼び出されるダイナミックリンクライブラリのデバッグを行う場合は、ダイナミックリンクライブラリと同じフォルダに格納された実行用の初期化ファイルを参照するように指定する必要があります。[参照]“[5.3.2.3.4 DLL配下の実行用の初期化ファイルの使用について](#)”

2. デバッグしたいプログラムを実行します。

Webブラウザから入力を行うなど、デバッグしたいCOBOLプログラムが動作する操作を行います。

3. クライアント側でリモートデバッグが起動します。

リモートデバッグが起動されると、クライアント側でリモートデバッグのメインウィンドウが表示されます。

デバッグを開始するには、[デバッグを開始する]ダイアログで、必要な情報を入力して[OK]ボタンをクリックします。[参照]“[20.9.5 リモートデバッグの起動](#)”

なお、サーバ側で環境変数@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. リモートデバッグを使ってデバッグを行います。

リモートデバッグの操作方法是通常のデバッグと同じです。

操作方法については、“[20.5 デバッグの操作](#)”を参照してください。

なお、クライアント側のリモートデバッグコネクタが起動している間は、デバッグしたいプログラムを実行することで繰り返しデバッグを行うことができます。

5. リモートデバッグを終了させます。

[ファイル]メニューの“デバッグの終了”を選択することで、リモートデバッグを終了します。

6. クライアント側のリモートデバッグコネクタを終了させます。

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“リモートデバッグコネクタの終了”を選択します。

サーバがSolarisまたはLinuxの場合には、以下の点が異なります。

#### 環境変数@CBR\_ATTACH\_TOOL

環境変数名は、CBR\_ATTACH\_TOOLです。先頭に“@”は付きません。

また、指定形式の“接続先/TEST”の部分で、“接続先”を指定しない場合、“TEST”も指定してはなりません。



実行用の初期化ファイル(COBOL85.CBR)

COBOL.CBRです。

共用オブジェクトファイルと同じフォルダに格納された実行用の初期化ファイルを参照するように指定する方法

各OS版の“NetCOBOL 使用手引書”の“ライブラリ格納ディレクトリの実行用の初期化ファイルの使用について”を参照してください。

### 20.9.2.3 ジャストインタイムデバッグ機能を使用したリモートデバッグ起動方法

プログラムでアプリケーションエラーや実行時メッセージが発生した場合に、自動的にリモートデバッグが起動されてプログラムをデバッグできる起動方法です。

この起動方法は、サーバがWindowsの場合にのみ使用できます。

起動方法は、環境変数名@CBR\_JUSTINTIME\_DEBUGを使用すること以外は、“[20.9.2.2 Interstageなどのサーバ環境で動作するプログラムに対するリモートデバッグ起動方法](#)”と同じです。[参照] “[5.4.1.32 @CBR\\_JUSTINTIME\\_DEBUG \(異常終了時にデバッグまたは診断機能を使って調査を行う指定\)](#)”

プログラムを実行した時にアプリケーションエラーや実行時メッセージが発生すると、リモートデバッグが起動され、リモートデバッグのメインウィンドウが表示されます。デバッグを開始するには、[デバッグを開始する]ダイアログで[OK]ボタンをクリックします。デバッグが開始されると、アプリケーションエラーまたは実行時メッセージが発生した状態で中断します。

## 20.9.3 リモートデバッグコネクタの使い方

リモートデバッグでは、ネットワーク上の異なるコンピュータからのデバッグ開始の指示を監視するプログラムが必要になります。

このプログラムには、以下の2種類があります。

#### サーバ側のリモートデバッグコネクタ

サーバ側で動作するプログラムで、クライアント側からのデバッグ開始の指示を監視します。

“[20.9.2.1 一般的なプログラムに対するリモートデバッグ起動方法](#)”で使用します。

#### クライアント側のリモートデバッグコネクタ

クライアント側で動作するプログラムで、サーバ側からのデバッグ開始の指示を監視します。

“[20.9.2.2 Interstageなどのサーバ環境で動作するプログラムに対するリモートデバッグ起動方法](#)”および、“[20.9.2.3 ジャストインタイムデバッグ機能を使用したリモートデバッグ起動方法](#)”で使用します。

これらのプログラムでは、必要に応じて、接続を許可するコンピュータを制限するための設定を行えます。詳しくは、“[20.9.3.5 リモートデバッグを許可するコンピュータを制限する](#)”を参照してください。

### 20.9.3.1 サーバ側のリモートデバッグコネクタの使い方

サーバ側のリモートデバッグコネクタは、“[20.9.2.1 一般的なプログラムに対するリモートデバッグ起動方法](#)”でリモートデバッグを行う場合に使用します。

このプログラムは、サーバ側で動作させます。

### 20.9.3.2 サーバ側のリモートデバッグコネクタの起動方法

サーバ側のリモートデバッグコネクタの起動形式を以下に示します。

#### サーバ側のリモートデバッグコネクタの起動形式

サーバがWindowsの場合

|                            |
|----------------------------|
| cobrrds32 [ポート指定] [接続制限指定] |
|----------------------------|

サーバがWindows(Itanium)の場合

|                            |
|----------------------------|
| cobrrds64 [ポート指定] [接続制限指定] |
|----------------------------|

サーバがSolarisまたはLinuxの場合

```
svdrds [ポート指定] [接続制限指定]
```

## ポート指定の形式

```
-p ポート番号
```

-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。

## 接続制限指定の形式

```
-h ホスト名 | -s 接続制限ファイル名 [-e]
```

-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。
- リモートデバッガコネクタが起動されると、IPアドレスとポート番号を表示して、クライアント側からのデバッグ開始の指示を監視します。
- リモートデバッガコネクタを終了させる場合は、CtrlキーとCキーを同時に押します。



## 参考

- デバッグしたいプログラムに対して環境変数を指定する必要がある場合は、リモートデバッガコネクタを起動する前に設定しておきます。
- リモートデバッガコネクタは、デバッグしたいプログラムを異なる環境で動作させる必要がある場合には、その環境ごとに起動する必要があります。  
なお、ここでの環境とは、環境変数のようにリモートデバッガコネクタを起動した時点で動作が確定し、それ以降は変更できない設定のことです。
- プログラムを異なる環境で動作させる必要がある場合や、複数の開発者が同時にデバッグを行う場合には、リモートデバッガコネクタの起動時に“-p ポート番号”を指定して異なるポートを使用するようにします。



## 例

### サーバ側のリモートデバッガコネクタの起動例

- 複数の開発者が同時にデバッグを行う場合

```
cobdrds32 -p 10001 -h client-1 ← 開発者-1用
cobdrds32 -p 10002 -h client-2 ← 開発者-2用
cobdrds32 -p 10003 -h client-3 ← 開発者-3用
```

- 開発者ごとに使用するポート番号をあらかじめ決めておき、-pで指定します。
- -hを指定して、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し、他の開発者が誤って接続しても拒否されるようにします。

### 20.9.3.3 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。

#### 接続制限ファイルの形式

```
[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は許可します。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は拒否します。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には、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20.4 接続対象の指定形式(IPv4アドレスの場合)”および“表20.5 接続対象の指定形式(IPv6アドレスの場合)”に接続対象の指定形式を示します。

表20.4 接続対象の指定形式(IPv4アドレスの場合)

| 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(注1) | 192.168.0.*              | 192.168.0.0 ~ 192.168.0.255 |
| ホスト名(注2)    | Hostname                 | 192.168.0.1                 |

注1:ワイルドカード“\*”は、“.”で区切られた各10進数(8ビット)の代わりに指定できます。

注2: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。

表20.5 接続対象の指定形式(IPv6アドレスの場合)

| 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    | fe80::1:23:456:789a                  | fe80::1:23:456:789a                      |
| 範囲指定        | fe80::1:23:456:1-fe80::1:23:456:789a | fe80::1:23:456:1 ~ fe80::1:23:456:789a   |
| ワイルドカード(注3) | fe80::1:23:*:789a                    | fe80::1:23:0:789a ~ fe80::1:23:ffff:789a |
| ホスト名(注4)    | Hostname                             | fe80::1:23:456:789a                      |

注3:ワイルドカード“\*”は、“.”で区切られた各16進数(16ビット)の代わりに指定できます。

注4:IPアドレス fe80::1:23:456:789a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。

なお、代表的な指定方法については、“[20.9.3.5 リモートデバッグを許可するコンピュータを制限する](#)”を参照してください。

また、IPv6への対応仕様については、“[表5.1 デバッガのIPv6対応](#)”を参照してください。



## 注意

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。

### 20.9.3.4 クライアント側のリモートデバッグコネクタの使い方

クライアント側のリモートデバッグコネクタは、“[20.9.2.2 Interstageなどのサーバ環境で動作するプログラムに対するリモートデバッグ起動方法](#)”および、“[20.9.2.3 ジャストインタイムデバッグ機能を使用したリモートデバッグ起動方法](#)”でリモートデバッグを行う場合に使用します。

このプログラムは、クライアント側で動作させます。

#### 20.9.3.4.1 クライアント側のリモートデバッグコネクタの起動方法

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方法を以下に示します。

- プロジェクトマネージャウィンドウの[ツール]メニューから“リモートデバッグコネクタ”を選択します。

リモートデバッグ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サーバ側からのデバッグ開始の指示を監視します。



設定を変更する場合や、設定内容を確認したい場合は、タスクトレイ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。また、リモートデバッグコネクタを終了する場合は、タスクトレイのアイコンを右クリックして、表示されるメニューから“リモートデバッグコネクタの終了”を選択します。



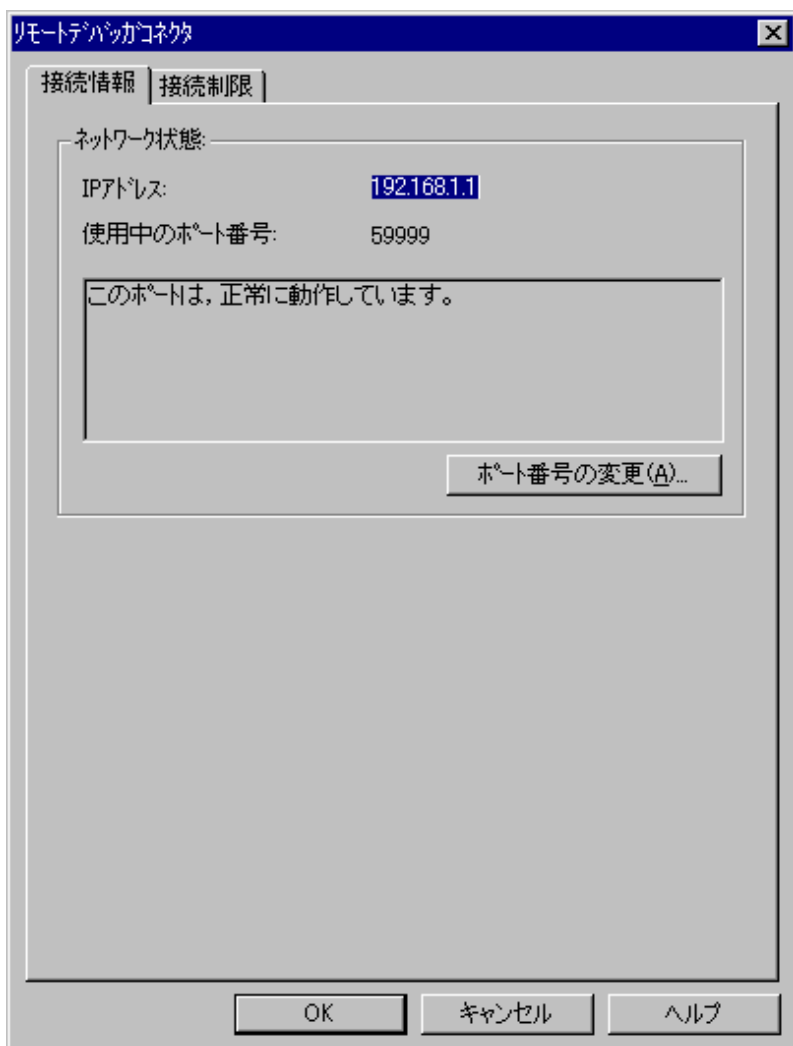
## 参考

リモートデバッグコネクタは、NetCOBOL Studioから起動することもできます。詳細については“NetCOBOL Studio 使用手引書”を参照してください。

#### 20.9.3.4.2 [リモートデバッグコネクタ]ダイアログの設定方法

[リモートデバッグコネクタ]ダイアログの設定方法について説明します。以下に、[リモートデバッグコネクタ]ダイアログをページごとに示します。

## 接続情報ページ



### スタティックテキスト

#### IPアドレス

リモートデバッグコネクタが動作しているコンピュータのIPアドレスまたはホスト名が表示されます。

#### 使用中のポート番号

リモートデバッグコネクタが監視しているポート番号が表示されます。

### スタティックテキストボックス

クライアント側のリモートデバッグコネクタの動作状態を表示します。

### ボタン

#### ポート番号の変更

[ポート番号の変更]ダイアログを表示します。

## 接続制限ページ

リモートデスクトップ接続

接続情報 接続制限

アクセスしたコンピュータの接続を許可するか拒否するか設定します。

接続を許可する対象:

☒ すべての接続を許可する(L):

☐ 許可する対象(Q):

追加(A)... 編集(E)... 削除(R)

接続を拒否する対象:

☒ すべての接続を拒否する(N):

☐ 拒否する対象(Y):

追加(D)... 編集(T)... 削除(M)

リセット(R)...

OK キャンセル ヘルプ

### 接続を許可する対象

#### ラジオボタン

##### すべての接続を許可する

全てのコンピュータからの接続を許可します。

##### 許可する対象

接続を許可する接続対象を表示します。

追加、編集、削除の各ボタンを使って、接続を許可する対象を設定できます。

#### ボタン

##### 追加、編集、削除

接続を許可する対象を設定するために使用します。

### 接続を拒否する対象

#### ラジオボタン

##### すべての接続を拒否する

全てのコンピュータからの接続を拒否します。

##### 拒否する対象

接続を拒否する接続対象を表示します。

追加、編集、削除の各ボタンを使って、接続を拒否する対象を設定できます。

#### ボタン

##### 追加、編集、削除

接続を拒否する対象を設定するために使用します。

#### リセット

接続を許可する対象と接続を拒否する対象に設定された接続対象をすべて削除して、ダイアログの内容を初期状態に戻します。

なお、代表的な指定方法については、“[20.9.3.5 リモートデバッグを許可するコンピュータを制限する](#)”を参照してください。

## 20.9.3.5 リモートデバッグを許可するコンピュータを制限する

リモートデバッグを許可するコンピュータを制限する方法を“[表20.6 接続制限の代表例](#)”に示します。表ではIPv4アドレスを使用していますが、同様にIPv6アドレスを指定することもできます。[参照]“[表5.1 デバッグのIPv6対応](#)”

なお、表では、サーバ側のリモートデバッグコネクタをサーバ側、クライアント側のリモートデバッグコネクタをクライアント側と表現しています。

また、サーバ側では、接続制限ファイルを使用します。[参照]“[20.9.3.3 接続制限ファイルの形式](#)”

クライアント側では、リモートデバッグコネクタの[接続制限]ページに設定を行います。[参照]“[20.9.3.4.2 \[リモートデバッグコネクタ\]ダイアログの設定方法](#)”

表20.6 接続制限の代表例

| 指定形式                                                                                        | サーバ側                                                            | クライアント側                                                                                                                                                                       |
|---------------------------------------------------------------------------------------------|-----------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 特定の接続対象のみを許可する。<br>接続を許可するIPアドレスが、以下の場合<br><br>192.168.0.1                                  | ALLOW=192.168.0.1<br><br>DENY=ALL                               | [接続を許可する対象]の設定<br><br>・ [許可する対象]を選択します。<br><br>・ エディットボックスに、以下の設定を行います。<br><br>192.168.0.1<br><br>[接続を拒否する対象]の設定<br><br>・ [すべての接続を拒否する]を選択します。                                |
| 複数の接続対象からの接続を許可する。<br>接続を許可するIPアドレスが以下の場合<br><br>192.168.0.1<br>192.168.0.2<br>192.168.0.10 | ALLOW=192.168.0.1 ¥<br>192.168.0.2 192.168.0.10<br><br>DENY=ALL | [接続を許可する対象]の設定<br><br>・ [許可する対象]を選択します。<br><br>・ エディットボックスに、以下の設定を行います。<br><br>192.168.0.1<br>192.168.0.2<br>192.168.0.10<br><br>[接続を拒否する対象]の設定<br><br>・ [すべての接続を拒否する]を選択します。 |
| 特定範囲の接続を許可する。<br>接続を許可するIPアドレスの範囲が以下の場合<br><br>下限：192.168.0.1<br>上限：192.168.0.10            | ALLOW=192.168.0.1-192.168.0.10<br><br>DENY=ALL                  | [接続を許可する対象]の設定<br><br>・ [許可する対象]を選択します。<br><br>・ エディットボックスに、以下の設定を行います。<br><br>192.168.0.1-192.168.0.10<br><br>[接続を拒否する対象]の設定                                                 |

| 指定形式                                                                                 | サーバ側                          | クライアント側                                                                                                                                                                                                                                 |
|--------------------------------------------------------------------------------------|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                      |                               | <ul style="list-style-type: none"> <li>・ [すべての接続を拒否する]を選択します。</li> </ul>                                                                                                                                                                |
| オクテッド単位に接続を許可する。<br>接続を許可するIPアドレスの範囲が以下の場合<br>下限 : 192.168.0.0<br>上限 : 192.168.0.255 | ALLOW=192.168.0.*<br>DENY=ALL | [接続を許可する対象]の設定<br><ul style="list-style-type: none"> <li>・ [許可する対象]を選択します。</li> <li>・ エディットボックスに、以下の設定を行います。<br/><br/>192.168.0.*</li> </ul> [接続を拒否する対象]の設定<br><ul style="list-style-type: none"> <li>・ [すべての接続を拒否する]を選択します。</li> </ul> |
| すべてのコンピュータからの接続を許可する。                                                                | ALLOW=ALL<br>DENY=ALL         | [接続を許可する対象]の設定<br><ul style="list-style-type: none"> <li>・ [すべての接続を許可する]を選択します。</li> </ul> [接続を拒否する対象]の設定<br><ul style="list-style-type: none"> <li>・ [すべての接続を拒否する]を選択します。</li> </ul>                                                   |

## 20.9.4 リモートデバッガの環境変数

リモートデバッガが使用する環境変数は、Win32リモートデバッガおよびWin64リモートデバッガの場合は通常のデバッガが使用する環境変数と同じです。

ここでは、SolarisリモートデバッガまたはLinuxリモートデバッガが使用する環境変数について説明します。

なお、通常のデバッガと同様に、SolarisリモートデバッガまたはLinuxリモートデバッガが使用する環境変数は、バッチデバッグに関する環境変数であるため、バッチデバッグを行わない場合には、指定する必要はありません。

バッチデバッグについては、“[20.5.2.2 デバッグ操作の自動化](#)”を参照してください。

環境変数の設定方法については、“[20.3 デバッガの環境変数](#)”を参照してください。

### 20.9.4.1 @SVD\_COMPATIBILITY(互換性の指定)

SolarisリモートデバッガまたはLinuxリモートデバッガの機能範囲を限定するための指定をします。

この環境変数を指定すると、SolarisリモートデバッガまたはLinuxリモートデバッガの互換性に関する初期値を環境変数で指定した状態にできます。

環境変数を設定しない場合は、前回のデバッグ終了時の指定内容に従います。

環境変数には、コマンドファイルを作成したときのSolarisデバッガまたはLinuxデバッガの製品の種別とバージョンレベルを指定します。

#### Solarisリモートデバッガの場合

---

```
@SVD_COMPATIBILITY = { COBOL97 V61L10
 NetCOBOL バージョンレベル
 }
```

---



## COBOL97 V61L10

COBOL97 V61L10 svdコマンドに似た動作をします。

この動作モードで作成した操作履歴ファイルに若干の修正を加えるとSolaris上で動作するsvdコマンドのコマンドファイルとして使用できるようになります。

## NetCOBOLバージョンレベル

常に最新の動作をします。

バージョンレベルには、V7.0L10以降のバージョンレベルを指定します。

## Linuxリモートデバッグの場合

---

@SVD\_COMPATIBILITY = NetCOBOL バージョンレベル

---

バージョンレベルの指定により、以下の動作をします。

- V7.0L10を指定した場合

NetCOBOL V7.0L10 svdコマンドに似た動作をします。

この動作モードで作成した操作履歴ファイルに若干の修正を加えると、Linux上で動作するsvdコマンドのコマンドファイルとして使用できるようになります。

- V7.2L10以降のバージョンレベルを指定した場合

常に最新の動作をします。

詳細は、SolarisリモートデバッグまたはLinuxリモートデバッグの[環境設定]ダイアログボックスの[互換性]ページのヘルプを参照してください。



### 注意

- 環境変数@SVD\_COMPATIBILITYは、svdコマンド用のコマンドファイルをリモートデバッグで使用できるようにするための指定ではありません。svdコマンド用のコマンドファイルは、SolarisまたはLinux上のsvdコマンドを使用してバッチデバッグを行ってください。
- リモートデバッグで大量のバッチデバッグを行うとネットワークに負荷がかかることが考えられます。大量のバッチデバッグを行う場合には、SolarisまたはLinux上のsvdコマンドを使用してバッチデバッグを行うことをお勧めします。

## 20.9.5 リモートデバッグの起動

ここでは、リモートデバッグの起動方法およびデバッグに必要な情報の設定について説明します。

### 20.9.5.1 [デバッグを開始する]ダイアログを使ったリモートデバッグ指定

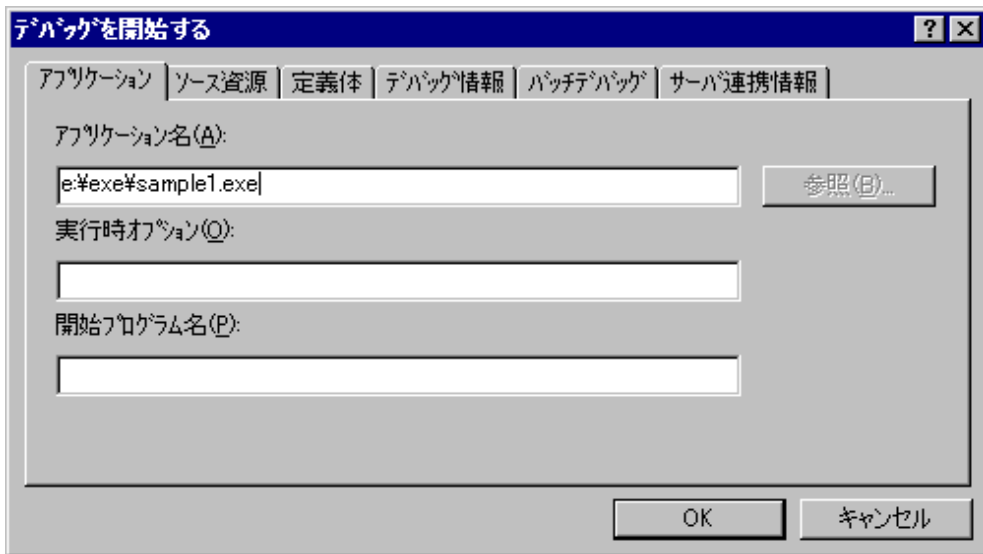
ここでは、[デバッグを開始する]ダイアログの指定方法について説明します。

殆どの項目については、通常のデバッグと同じですが、デバッグに必要な資産がサーバ側または、クライアント側に適切に格納されている必要があります。

リモートデバッグに必要な資産の格納場所については、“[20.9.1.2 リモートデバッグ時の資産の格納場所](#)”を参照してください。

以下に、[デバッグを開始する]ダイアログをページごとに示します。

## アプリケーションページ



デバッグを開始する

アプリケーション | ソース資源 | 定義体 | デバッグ情報 | バッチデバッグ | サーバ連携情報

アプリケーション名(A):  
e:\exe\sample1.exe 参照(R)...

実行時オプション(O):

開始プログラム名(P):

OK キャンセル

### エディットボックス

#### アプリケーション名

デバッグを行う実行可能プログラムを指定します。サーバ側のフォルダ構成に従って指定してください。

Win32リモートデバッガまたはWin64リモートデバッガで拡張子を省略した場合は、拡張子EXEのファイルを指定したものとみなされます。SolarisリモートデバッガまたはLinuxリモートデバッガの場合には、拡張子EXEのファイルを指定したとみなす動作は行われません。

このパラメタは、省略できません。

#### 実行時オプション

実行可能プログラムに渡すオプションを指定します。オプションとしてフォルダ名やファイル名を指定する場合は、サーバ側のフォルダ構成で指定してください。

このパラメタは、省略可能です。

#### 開始プログラム名

デバッグを開始して最初にコマンドの入力が可能になるプログラム名を指定します。指定するプログラムは、被デバッグプログラムでなければなりません。

このパラメタは、省略可能です。省略した場合は、プログラムの実行で最初に見つかった被デバッグプログラムでのデバッグ操作が可能になります。

### ボタン

#### 参照

使用することはできません。アプリケーション名に直接指定してください。

## ソース資源ページ

デバッグを開始する

アプリケーション | **ソース資源** | 定義体 | デバッグ情報 | バッチデバッグ | サーバ連携情報

ソースファイル格納フォルダ(S):

登録集格納フォルダ(O):

サブスキーマ定義ファイル格納フォルダ(B):

参照(R)...

参照(O)...

参照(Q)...

OK キャンセル

### エディットボックス

#### ソースファイル格納フォルダ

ソースファイルを格納しているフォルダを指定します。クライアント側のフォルダ構成に従って指定してください。複数のフォルダを指定したい場合は、セミコロン(;)で区切って指定します。

このパラメータは、デバッグ中のCOBOLソースプログラムをメインウィンドウに表示するために必要となります。デバッガは、指定されたフォルダ順にソースファイルを検索します。

このパラメータは、省略可能です。

#### 登録集格納フォルダ

登録集ファイルを格納しているフォルダを指定します。クライアント側のフォルダ構成に従って指定してください。複数のフォルダを指定したい場合は、セミコロン(;)で区切って指定します。

このパラメータは、デバッグ中のCOBOLソースプログラムから取り込まれる登録集ファイルをメインウィンドウに表示するために必要となります。デバッガは、指定されたフォルダ順に登録集ファイルを検索します。

このパラメータは、省略可能です。省略した場合および指定されたフォルダから検索できなかった場合は、環境変数COB\_COBCOPYで指定されたフォルダを検索します。IN/OF指定の登録集ファイルについては、環境変数COB\_登録集名で指定されたフォルダを検索します。[参照]“[1.2.1 環境変数の設定](#)”

#### サブスキーマ定義ファイル格納フォルダ

この指定は、Win32リモートデバッガの場合のみ有効です。

サブスキーマ定義ファイルを格納しているフォルダを指定します。クライアント側のフォルダ構成に従って指定してください。複数のフォルダを指定したい場合は、セミコロン(;)で区切って指定します。

このパラメータは、デバッグ中のCOBOLソースプログラムから取り込まれるサブスキーマ定義ファイルをメインウィンドウに表示するために必要となります。デバッガは、指定されたフォルダ順にサブスキーマ定義ファイルを検索します。

このパラメータは、省略可能です。省略した場合および指定されたフォルダから検索できなかった場合は、環境変数COB\_AIMLIBで指定されたフォルダを検索します。[参照]“[1.2.1 環境変数の設定](#)”

### ボタン

#### 参照

フォルダを参照するときに選択します。

## 定義体ページ

デバッグを開始する

アプリケーション | ソース資源 | 定義体 | デバッグ情報 | バッチデバッグ | サーバ連携情報

画面帳票定義体

フォルダ(F):  参照(R)...

拡張子(E): pmd;pxd;smd

ファイル定義体

フォルダ(L):  参照(R)...

拡張子(O): ffd

OK キャンセル

### エディットボックス

#### 画面帳票定義体のフォルダ

画面帳票定義体ファイルを格納しているフォルダを指定します。クライアント側のフォルダ構成に従って指定してください。複数のフォルダを指定したい場合は、セミコロン(;)で区切って指定します。

このパラメータは、デバッグ中のCOBOLソースプログラムから取り込まれる画面帳票定義体ファイルをメインウィンドウに表示するために必要となります。デバッグは、指定されたフォルダ順に画面帳票定義体ファイルを検索します。

このパラメータは、省略可能です。省略した場合および指定されたフォルダから検索できなかった場合は、環境変数FORMLIBで指定されたフォルダを検索します。[参照]“[1.2.1 環境変数の設定](#)”

#### ファイル定義体のフォルダ

この指定は、Win32リモートデバッグ、Solarisリモートデバッグ、またはLinuxリモートデバッグの場合のみ有効です。

ファイル定義体ファイルを格納しているフォルダを指定します。クライアント側のフォルダ構成に従って指定してください。複数のフォルダを指定したい場合は、セミコロン(;)で区切って指定します。

このパラメータは、デバッグ中のCOBOLソースプログラムから取り込まれるファイル定義体ファイルをメインウィンドウに表示するために必要となります。デバッグは、指定されたフォルダ順にファイル定義体ファイルを検索します。

このパラメータは、省略可能です。省略した場合または指定されたフォルダから検索できなかった場合は、環境変数FILELIBで指定されたフォルダを検索します。[参照]“[1.2.1 環境変数の設定](#)”

### ドロップダウンコンボボックス

#### 画面帳票定義体の拡張子

画面帳票定義体ファイルの拡張子を指定します。

このパラメータは、初期値としてpmd;pxd;smdが設定されています。

#### ファイル定義体の拡張子

この指定は、Win32リモートデバッグ、Solarisリモートデバッグ、またはLinuxリモートデバッグの場合のみ有効です。

ファイル定義体ファイルの拡張子を指定します。

このパラメータは、初期値としてffdが設定されています。

### ボタン

#### 参照

フォルダを参照するときに選択します。

## デバッグ情報ページ

The screenshot shows a Windows-style dialog box titled "デバッグを開始する" (Start Debugging). It has a tabbed interface with the following tabs: "アプリケーション" (Application), "ソース資源" (Source Resources), "定義体" (Definitions), "デバッグ情報" (Debugging Information), "バッチデバッグ" (Batch Debugging), and "サーバ連携情報" (Server Collaboration Information). The "デバッグ情報" tab is selected. Inside the dialog, there is a label "デバッグ情報ファイル格納フォルダ(D):" (Debugging information file storage folder (D):) followed by a text input field and a "参照(B)..." (Browse...) button. At the bottom of the dialog are "OK" and "キャンセル" (Cancel) buttons.

### エディットボックス

#### デバッグ情報ファイル格納フォルダ

デバッグ情報ファイルを格納しているフォルダを指定します。サーバ側のフォルダ構成に従って指定してください。複数のフォルダを指定したい場合は、セミコロン(;)で区切って指定します。

デバッガは、指定されたフォルダ順にデバッグ情報ファイルを検索します。

このパラメタは、省略可能です。省略した場合および指定されたフォルダから検索できなかった場合は、Win32リモートデバッガまたはWin64リモートデバッガの場合は、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が格納されているフォルダを検索します。SolarisリモートデバッガまたはLinuxリモートデバッガの場合は、[サーバ連携情報]ページの[サーバディレクトリ]で指定したディレクトリを検索します。

### ボタン

#### 参照

使用することはできません。デバッグ情報ファイル格納フォルダに直接指定してください。

## バッチデバッグページ

The screenshot shows the same "デバッグを開始する" (Start Debugging) dialog box, but with the "バッチデバッグ" (Batch Debugging) tab selected. The "デバッグ情報" tab is now disabled. In the "バッチデバッグ" tab, there is a checkbox labeled "バッチデバッグを行う(B)" (Perform batch debugging (B)). Below the checkbox are two groups of controls. The first group has a label "コマンドファイル名(C):" (Command file name (C):) followed by a text input field and a "参照(B)..." (Browse...) button. The second group has a label "操作履歴ファイル名(H):" (Operation history file name (H):) followed by a text input field and a "参照(B)..." (Browse...) button. At the bottom of the dialog are "OK" and "キャンセル" (Cancel) buttons.

## チェックボックス

### バッチデバッグを行う

画面からの対話処理を行わずに、一連のコマンドを格納したコマンドファイルから一括してデバッグ操作を行う場合に、このチェックボックスを選択します。

## エディットボックス

### コマンドファイル名

バッチデバッグを行う場合に、コマンドファイル名を指定します。クライアント側のフォルダ構成に従って指定してください。相対パス名で指定した場合は、デバッグ起動時のカレントフォルダ名が付加されます。拡張子を省略した場合は、拡張子LOGのファイルを指定したものとみなされます。

このパラメタは、バッチデバッグを行うチェックボックスが指定されている場合は省略できません。

バッチデバッグおよびコマンドファイルについては、“[20.5.2.2 デバッグ操作の自動化](#)”を参照してください。

### 操作履歴ファイル名

デバッグ操作のコマンドとコマンドの実行結果を保存する場合に、保存先の操作履歴ファイル名を指定します。クライアント側のフォルダ構成に従って指定してください。相対パス名で指定した場合は、デバッグ起動時のカレントフォルダ名が付加されます。拡張子を省略した場合には、拡張子LOGのファイルを指定したものとみなされます。

このパラメタは、バッチデバッグを行うチェックボックスが指定されている場合は省略できません。

ここに指定されたファイルがすでに存在している場合は、ファイルが上書きされます。

なお、コマンドファイル名のエディットボックスとこの操作履歴ファイル名のエディットボックスに同じファイルは指定できません。

## ボタン

### 参照

ファイルを参照するときに選択します。

## サーバ連携情報ページ

デバッグを開始する

アプリケーション | ソース資源 | 定義体 | デバッグ情報 | バッチデバッグ | **サーバ連携情報**

接続先(H):

192.168.0.1

解除(D)

サーバディレクトリ(S):

OK キャンセル

## ドロップダウンコンボボックス

### 接続先

接続先には、サーバ側のリモートデバッグコネクタのポート番号と動作しているホストを以下の形式で指定します。

-----

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

IPアドレスは、IPv4またはIPv6の形式で指定します。

ポート番号は、1024から65535の範囲の数字を指定します。ポート番号を省略した場合は、59998が指定されたと見なされます。

IPv6形式の場合は、スコープ付きアドレスを指定することができます。スコープ付きアドレスを指定する場合、アドレスの後にスコープ識別子を指定します。

なお、IPv6アドレスにポート番号を指定する場合は、アドレス部分を中括弧([ ])で囲んでください。

IPv6への対応仕様については、“[表5.1 デバッガのIPv6対応](#)”を参照してください。



## 例

〔IPv4アドレス (192. 168. 0. 1) およびポート番号 (2000) での指定〕

192. 168. 0. 1:2000

〔IPv6アドレス (fe80::1:23:456:789a) およびポート番号 (2000) での指定〕

[fe80::1:23:456:789a]:2000

〔IPv6アドレス (fe80::1:23:456:789a) およびスコープ識別子 (%eth0) での指定〕

fe80::1:23:456:789a%eth0

〔IPv6アドレス (fe80::1:23:456:789a)、スコープ識別子 (%5) およびポート番号 (2000) での指定〕

[fe80::1:23:456:789a%5]:2000

〔ホスト名 (server-1) およびポート番号 (2000) での指定〕

server-1:2000

〔ポート番号省略の指定 (IPv4) 〕

192. 168. 0. 1

〔ポート番号省略の指定 (IPv6) 〕

fe80::1:23:456:789a

また、“[20.9.2.2 Interstageなどのサーバ環境で動作するプログラムに対するリモートデバッグ起動方法](#)”や“[20.9.2.3 ジャストインタイムデバッグ機能を使用したリモートデバッグ起動方法](#)”で起動された場合には、サーバ側のIPアドレスを表示します(変更はできません)。

## エディットボックス

### サーバディレクトリ

アプリケーションが動作する時のカレントフォルダを指定します。

なお、[アプリケーション] ページの[アプリケーション名]や[実行時オプション]や[デバッグ情報] ページの[デバッグ情報ファイル格納フォルダ]で相対パス指定を行った場合には、ここで指定したフォルダからの相対パスとして扱われます。

また、“[20.9.2.2 Interstageなどのサーバ環境で動作するプログラムに対するリモートデバッグ起動方法](#)”や“[20.9.2.3 ジャストインタイムデバッグ機能を使用したリモートデバッグ起動方法](#)”で起動された場合には、アプリケーションが動作した時のカレントフォルダを表示します(変更はできません)。

## ボタン

### 解除

現在表示されている接続先が、接続先ドロップダウンコンボボックスから削除されます。

## 20.9.5.2 起動コマンドを使ったリモートデバッグ指定

ここでは、リモートデバッグをコマンド形式で起動する場合の指定方法を説明します。

コマンド形式での起動方法は、Win32リモートデバッグ、SolarisリモートデバッグおよびLinuxリモートデバッグの場合のみ使用できます。ただし、Win64リモートデバッグは、「Interstageなどのサーバ環境で動作するプログラムに対するリモートデバッグ起動方法」において、環境変数@CBR\_ATTACH\_TOOLに設定する値としてwinsvdコマンドの起動パラメタを使用します。なお、winsvdコマンドでリモートデバッグを起動させるためには、プロジェクトマネージャウィンドウの[ツール]メニューの[デバッグ種別]で“Win32リモート”、“Solarisリモート”または“Linuxリモート”を選択しておく必要があります。

殆どの項目については、通常のデバッグと同じですが、デバッグに必要な資産がサーバ側または、クライアント側に適切に格納されている必要があります。

リモートデバッグに必要な資産の格納場所については、“[20.9.1.2 リモートデバッグ時の資産の格納場所](#)”を参照してください。

また、接続先を指定する起動パラメタが指定できます。

以下に、デバッグを起動するコマンドの形式を示します。なお、形式2では、winsvd、起動パラメタ、アプリケーション名、実行時オプションの指定は、以下に示す順序で指定しなければなりません。アプリケーション名は省略できません。起動パラメタと実行時オプションは省略することができます。

### 形式1

winsvd

形式1で起動すると、メインウィンドウが表示されます。デバッグを開始するには、[ファイル]メニューの“デバッグを開始する”を選択して、[デバッグを開始する]ダイアログを表示して、必要な情報を入力して、[OK]ボタンをクリックします。

### 形式2

winsvd [起動パラメタ] アプリケーション名 [実行時オプション]

形式2で起動すると、メインウィンドウが表示され、かつ、[デバッグを開始する]ダイアログが表示されます。この[デバッグを開始する]ダイアログの入力フィールドには、指定した起動パラメタ、アプリケーション名および実行時オプションが表示されます。デバッグを開始するには、[デバッグを開始する]ダイアログで必要な情報を入力して、[OK]ボタンをクリックします。

起動パラメタを“[表20.7 起動パラメタ](#)”に示します。

表20.7 起動パラメタ

| 指定形式   | 内容                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -U 接続先 | <div>リモートデバッグを行う接続先のサーバを以下の形式で指定します。</div> <div style="border: 1px solid black; padding: 2px; margin: 5px 0;">[IPアドレス ホスト名][:ポート番号]</div> <div>IPアドレスは、IPv4またはIPv6の形式で指定します。</div> <div>ポート番号は、1024から65535の範囲の数字を指定します。ポート番号を省略した場合は、前回接続した時のポート番号が指定されたと見なされます(初めての 경우에는、59998が指定されたと見なされます)。</div> <div>IPv6形式の場合は、スコープ付きアドレスを指定することができます。スコープ付きアドレスを指定する場合、アドレスの後にスコープ識別子を指定します。</div> <div>なお、IPv6アドレスにポート番号を指定する場合は、アドレス部分を中括弧([ ])で囲んでください。</div> <div>IPv6への対応仕様については、“<a href="#">表5.1 デバッグのIPv6対応</a>”を参照してください。</div> <div style="margin-top: 10px;"> <b>例</b></div> <div>.....</div> <div>[IPv4アドレス(192.168.0.1)およびポート番号(2000)での指定]<br/>192.168.0.1:2000</div> <div>[IPv6アドレス(fe80::1:23:456:789a)およびポート番号(2000)での指定]<br/>[fe80::1:23:456:789a]:2000</div> |



| 指定形式                   | 内容                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                        | <p>[IPv6アドレス (fe80::1:23:456:789a) およびスコープ識別子 (%eth0) での指定]<br/>fe80::1:23:456:789a%eth0</p> <p>[IPv6アドレス (fe80::1:23:456:789a)、スコープ識別子 (%5) およびポート番号 (2000) での指定]<br/>[fe80::1:23:456:789a%5]:2000</p> <p>[ホスト名 (server-1) およびポート番号 (2000) での指定]<br/>server-1:2000</p> <p>[ポート番号省略の指定 (IPv4)]<br/>192.168.0.1</p> <p>[ポート番号省略の指定 (IPv6)]<br/>fe80::1:23:456:789a<br/>.....</p> <p>本パラメタは、環境変数@CBR_ATTACH_TOOLおよび@CBR_JUSTINTIME_DEBUGに指定することはできません。</p> |
| -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。                                                                                                                                                                                                                                                                                                                                                                                                                        |
| -S ソースファイル格納フォルダ名      | <p>[デバッグを開始する]ダイアログでのソースファイル格納フォルダ名を指定します。</p> <p>クライアント側のフォルダ構成に従って指定します。</p>                                                                                                                                                                                                                                                                                                                                                                           |
| -C 登録集格納フォルダ名          | <p>[デバッグを開始する]ダイアログでの登録集格納フォルダ名を指定します。</p> <p>クライアント側のフォルダ構成に従って指定します。</p>                                                                                                                                                                                                                                                                                                                                                                               |
| -A サブスキーマ定義ファイル格納フォルダ名 | <p>[デバッグを開始する]ダイアログでのサブスキーマ定義ファイル格納フォルダ名を指定します。</p> <p>クライアント側のフォルダ構成に従って指定します。</p> <p>本パラメタは、Win32リモートデバッグの場合のみ有効です。</p>                                                                                                                                                                                                                                                                                                                                |
| -M 画面帳票定義体格納フォルダ名      | <p>[デバッグを開始する]ダイアログでの画面帳票定義体格納フォルダ名を指定します。</p> <p>クライアント側のフォルダ構成に従って指定します。</p>                                                                                                                                                                                                                                                                                                                                                                           |
| -E 画面帳票定義体拡張子          | [デバッグを開始する]ダイアログでの画面帳票定義体拡張子を指定します。                                                                                                                                                                                                                                                                                                                                                                                                                      |
| -H ファイル定義体格納フォルダ名      | <p>[デバッグを開始する]ダイアログでのファイル定義体格納フォルダ名を指定します。</p> <p>クライアント側のフォルダ構成に従って指定します。</p> <p>本パラメタは、Win32リモートデバッグ、SolarisリモートデバッグまたはLinuxリモートデバッグの場合のみ有効です。</p>                                                                                                                                                                                                                                                                                                     |
| -O ファイル定義体拡張子          | <p>[デバッグを開始する]ダイアログでのファイル定義体拡張子を指定します。</p> <p>本パラメタは、Win32リモートデバッグ、SolarisリモートデバッグまたはLinuxリモートデバッグの場合のみ有効です。</p>                                                                                                                                                                                                                                                                                                                                         |
| -D デバッグ情報ファイル格納フォルダ名   | <p>[デバッグを開始する]ダイアログでのデバッグ情報ファイル格納フォルダ名を指定します。</p> <p>サーバ側のフォルダ構成に従って指定します。</p>                                                                                                                                                                                                                                                                                                                                                                           |
| -B コマンドファイル名           | <p>[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。</p> <p>クライアント側のフォルダ構成に従って指定します。</p>                                                                                                                                                                                                                                                                                                                                                                                |
| -L 操作履歴ファイル名           | <p>[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。</p> <p>クライアント側のフォルダ構成に従って指定します。</p>                                                                                                                                                                                                                                                                                                                                                                                |
| -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。                                                                                                                                                                                                                                                                                                                                                                                                                      |

Win32リモートデバッグおよびWin64リモートデバッグの場合は、起動パラメタの先頭文字は、スラッシュ(/)でもハイフン(-)でも同様に扱われます。SolarisリモートデバッグまたはLinuxリモートデバッグの場合は、起動パラメタの先頭文字は、ハイフン(-)のみが使用できます。また、先頭文字に続くパラメタ名は、大文字と小文字が同様に扱われます。複数個の起動パラメタを指定する場合には、起動パラメタと次の起動パラメタとの間に1つ以上の空白を置いて指定します。



## 例

Win32リモートデバッグの場合

— 指定する起動パラメタ

|                   |                      |
|-------------------|----------------------|
| サーバ               | : server-1           |
| アプリケーション名         | : e:\exe\sample1.exe |
| 実行時オプション          | : param1             |
| ソースファイル格納フォルダ名    | : d:\source          |
| デバッグ情報ファイル格納フォルダ名 | : e:\svd             |

— 入力コマンド

```
winsvd /U server-1 /S d:\source /D e:\svd e:\exe\sample1.exe param1
```

SolarisリモートデバッグまたはLinuxリモートデバッグの場合

— 指定する起動パラメタ

|                   |                      |
|-------------------|----------------------|
| サーバ               | : server-1           |
| アプリケーション名         | : /home/usr1/sample1 |
| 実行時オプション          | : param1             |
| ソースファイル格納フォルダ名    | : d:\source          |
| デバッグ情報ファイル格納フォルダ名 | : /home/usr1/svd     |

— 入力コマンド

```
winsvd -U server-1 -S d:\source -D /home/usr1/svd /home/usr1/sample1 param1
```



## 注意

- 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、サーバ側のフォルダ構成に従って指定します。
- SolarisリモートデバッグまたはLinuxリモートデバッグの場合には、更に以下の注意事項があります。
  - 起動パラメタの先頭文字にスラッシュ(/)は使用できません。
  - 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する必要があります。
  - 指定可能な起動パラメタは、svdコマンドとは異なります。

## 20.9.6 リモートデバッグの操作

デバッグを開始した後の操作性は、通常のデバッグと同じです。[参照]“[20.5 デバッグの操作](#)の操作”

ただし、SolarisリモートデバッグまたはLinuxリモートデバッグの場合は、若干の相違点があります。詳細は、ヘルプを参照してください。

## 20.9.7 リモートデバッグの注意事項

ここでは、リモートデバッグを使用する際の注意事項について説明します。

なお、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。

### 20.9.7.1 Win32/Win64リモートデバッグの注意事項

ここでは、Win32リモートデバッグまたはWin64リモートデバッグ使用時の注意事項について説明します。

通常のデバッグと共通の注意事項については、“[20.6.5.2 Unicodeを扱う上での注意事項](#)”および、“[20.7 注意事項](#)”を参照してください。

- デバッグに必要な資産の格納場所に注意する必要があります。[参照] “[20.9.1.2 リモートデバッグ時の資産の格納場所](#)”
- 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、サーバ側のフォルダ構成に従って指定します。
- JEFオプション製品を使用して作成したCOBOLプログラムのデバッグはできません。

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

ここでは、SolarisリモートデバッグまたはLinuxリモートデバッグ使用時の注意事項について説明します。

### 実行時のコード系の決定方法

SolarisリモートデバッグまたはLinux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。

#### 一般的なプログラムに対するリモートデバッグ起動方法の場合

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。

#### Interstageなどのサーバ環境で動作するプログラムに対する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

SolarisリモートデバッグまたはLinuxリモートデバッグでは、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、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

通常のデバッグと共通の注意事項については、各OS版の“NetCOBOL 使用手引書”を参照してください。

- デバッグに必要な資産の格納場所に注意する必要があります。[参照] “[20.9.1.2 リモートデバッグ時の資産の格納場所](#)”
- 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、サーバ側のフォルダ構成に従って指定します。
- 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21章 通信機能

本章では、COBOLプログラムからメッセージ通信を行うための表示ファイル機能(非同期型メッセージ通信)および簡易アプリ間通信機能について説明します。

### 21.1 通信の種類

ここでは、COBOLプログラムから通信を行う場合の特徴と用途について、説明します。

次の方法で通信を行います。

- ・ 表示ファイル機能(非同期型メッセージ通信)
- ・ 簡易アプリ間通信機能

それぞれの機能の特徴および用途を“表21.1 表示ファイル機能と簡易アプリ間通信機能の特徴・用途”に示します。

表中の[1]～[4]は、通信形態および必要な関連製品が対応付いていることを表しています。

表21.1 表示ファイル機能と簡易アプリ間通信機能の特徴・用途

| 特徴・用途 |            | 表示ファイル機能                                                                                                                                                                                                   | 簡易アプリ間通信機能                                                                                                                                                                                                                                                                                |
|-------|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 特徴    | 通信機能       | 非同期型メッセージ通信                                                                                                                                                                                                | 非同期型メッセージ通信                                                                                                                                                                                                                                                                               |
|       | 通信形態       | [クライアント] <ul style="list-style-type: none"> <li>・ Windows XP</li> </ul> [サーバ] (注1) <ul style="list-style-type: none"> <li>・ Windows Server 2003 [2]</li> <li>・ UXP/DS [3]</li> <li>・ K シリーズ [4]</li> </ul> | [クライアント] <ul style="list-style-type: none"> <li>・ Windows</li> <li>・ UXP/DS</li> <li>・ HP-UX</li> <li>・ Solaris</li> <li>・ Linux</li> </ul> [サーバ] (注1) <ul style="list-style-type: none"> <li>・ Windows</li> <li>・ UXP/DS</li> <li>・ HP-UX</li> <li>・ Solaris</li> <li>・ Linux</li> </ul> |
|       | プログラムの記述内容 | <ul style="list-style-type: none"> <li>・ 表示ファイルの定義</li> <li>・ 通信レコードの定義</li> <li>・ OPEN文</li> <li>・ READ文</li> <li>・ WRITE文</li> <li>・ CLOSE文</li> </ul>                                                   | <ul style="list-style-type: none"> <li>・ 通信レコードの定義</li> <li>・ CALL文による簡易アプリ間通信機能のサブルーチンの呼出し               <ul style="list-style-type: none"> <li>— COBCI_OPEN</li> <li>— COBCI_READ</li> <li>— COBCI_WRITE</li> <li>— COBCI_CLOSE</li> </ul> </li> </ul>                                  |
|       | 必要な関連製品    | <ul style="list-style-type: none"> <li>・ SymfoWARE7000 for Windows NT [1]</li> <li>・ PowerRW+ [1][2] (注2)</li> <li>・ ACM Server for Windows NT [1][2]</li> <li>・ SymfoNET [3][4]</li> </ul>                | なし                                                                                                                                                                                                                                                                                        |
| 用途    |            | メッセージ通信を行う場合                                                                                                                                                                                               | COBOL 単体でメッセージ通信を行う場合                                                                                                                                                                                                                                                                     |

注1: サーバシステム内でのアプリケーションの起動もできます。  
注2: サーバ内ローカル運用のみ使用できます。

## 21.2 表示ファイル(非同期型メッセージ通信)の使い方

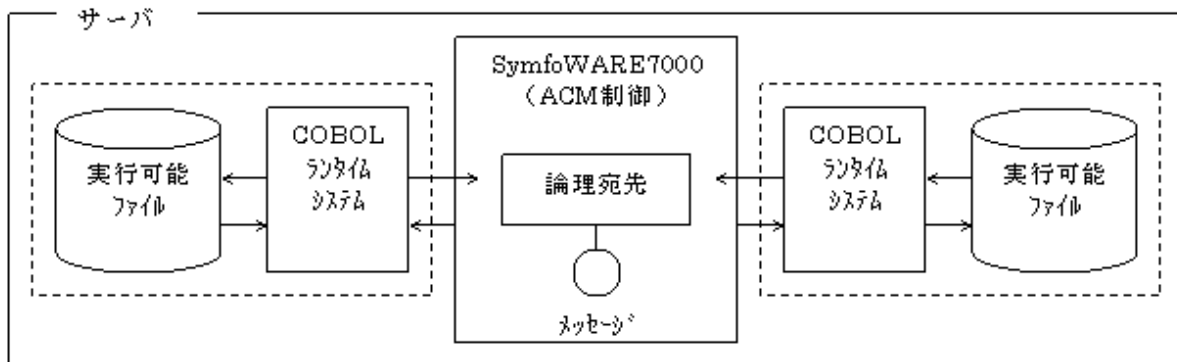
ここでは、メッセージ通信を行う表示ファイル機能の概要、動作環境、COBOLソースプログラムの作成および注意事項について説明します。

### 21.2.1 概要

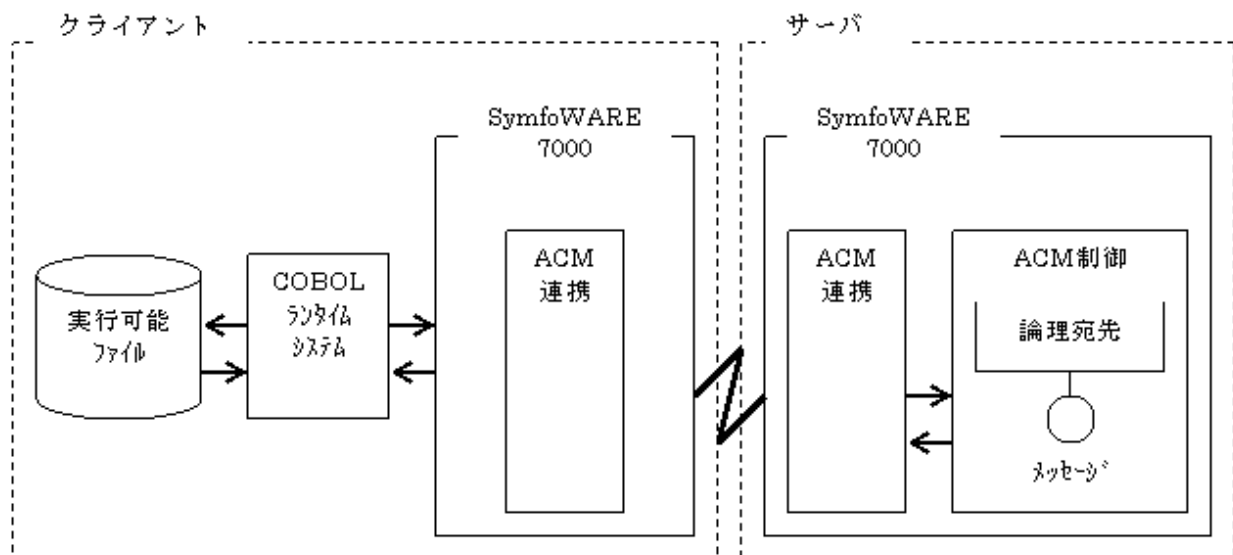
表示ファイル機能を使用するときには、通信機能を実現する接続製品が必要です。

ここでは、例として、SymfoWARE7000 for Windows NTを使用した場合の関連図を“[図21.1 表示ファイル機能\(非同期型メッセージ通信\)を使用するときの動作環境](#)”に示します。

図21.1 表示ファイル機能(非同期型メッセージ通信)を使用するときの動作環境  
【サーバ内ローカルの場合】



【クライアント-サーバの場合】



SymfoWARE7000の動作可能なオペレーティングシステムについては、SymfoWARE7000のマニュアルおよびソフトウェア説明書を確認してください。

### 21.2.2 プログラムの記述

ここでは、表示ファイル機能(非同期型メッセージ通信)を使うプログラムの記述内容について、COBOLの各部ごとに説明します。

```

IDENTIFICATION DIVISION.
 PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 SELECT ファイル名
 ASSIGN TO GS-ファイル識別名
 SYMBOLIC DESTINATION IS "ACM"
 DESTINATION-1 IS 論理宛先名通知域
 [PROCESSING MODE IS 処理種別通知域]
 [PROCESSING TIME IS 監視時間通知域]
 [MESSAGE CLASS IS 優先順位通知域]
 [FILE STATUS IS 入出力状態1 入出力状態2].
DATA DIVISION.
 FILE SECTION.
 FD ファイル名
 [RECORD IS VARYING IN SIZE DEPENDING ON レコードの大きさ].
 01 通信レコード名 ～.
WORKING-STORAGE SECTION.
 01 論理宛先名通知域 PIC X(8).
 [01 処理種別通知域 PIC X(2).]
 [01 監視時間通知域 PIC 9(4).]
 [01 優先順位通知域 PIC 9.]
 [01 入出力状態1 PIC X(2).]
 [01 入出力状態2 PIC X(4).]
 [01 レコードの大きさ PIC 9(5).]
PROCEDURE DIVISION.
 OPEN I-O ファイル名.
 MOVE 論理宛先名 TO 論理宛先名通知域.
 [MOVE 処理種別 TO 処理種別通知域.]
 [MOVE 監視時間 TO 監視時間通知域.]
 [MOVE 優先順位 TO 優先順位通知域.]
 WRITE 通信レコード名.
 READ ファイル名.
 CLOSE ファイル名.
END PROGRAM プログラム名.

```

## 環境部(ENVIRONMENT DIVISION)

表示ファイルを定義します。表示ファイルは、通常のファイルを定義するときと同様に、入出力節のファイル管理段落にファイル管理記述項を記述します。ファイル管理記述項に記述する内容を“表21.2 ファイル管理記述項に指定する情報”に示します。

表21.2 ファイル管理記述項に指定する情報

|    | 指定する場所                | 情報の種類   | 指定する内容および用途                                                                                     |
|----|-----------------------|---------|-------------------------------------------------------------------------------------------------|
| 必須 | SELECT句               | ファイル名   | COBOL プログラム中で使用するファイル名を指定します。このファイル名は、COBOL の利用者語の規則に従った名前にします。                                 |
|    | ASSIGN句               | ファイル参照子 | “GS-ファイル識別名”の形式で指定します。このファイル識別名は、実行時に接続製品名を設定する環境変数情報名となります。                                    |
|    | SYMBOLIC DESTINATION句 | 宛先種別の指定 | "ACM" を指定します。                                                                                   |
|    | DESTINATION-1句        |         | 作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名には、通信を行うとき、入出力処理の論理宛先名を設定します。                      |
| 任意 | FILE STATUS句          | データ名    | 作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、入出力処理の実行結果が設定されます。(注1)<br>詳細情報は、4桁の英数字項目を指定します。 |

|  | 指定する場所           | 情報の種類 | 指定する内容および用途                                                                                                                    |
|--|------------------|-------|--------------------------------------------------------------------------------------------------------------------------------|
|  | MESSAGE CLASS句   | データ名  | 作業場所節または連絡節で、1桁の数字項目として定義したデータ名を指定します。このデータ名には、通信を行うとき、入出力処理の優先順位を1～9の範囲で設定します。1が最も高い優先順位です。省略または0を指定すると、指定した論理宛先の最低レベルとみなします。 |
|  | PROCESSING MODE句 | データ名  | 作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、通信を行うとき、入出力処理の処理種別を設定します。(注2)                                                  |
|  | PROCESSING TIME句 | データ名  | 作業場所節または連絡節で、4桁の数字項目として定義したデータ名を指定します。このデータ名には、通信を行うとき、入出力処理の監視時間を秒単位で設定します。省略または0を指定すると、無限待ち時間となります。                          |

注1:設定される値については、“[付録B 入出力状態一覧](#)”を参照してください。

注2:詳細については、“[表21.3 処理種別の種類と指定する値](#)”を参照してください。

**表21.3 処理種別の種類と指定する値**

|    | 処理種別      |            | 処理内容                                                      |
|----|-----------|------------|-----------------------------------------------------------|
| 入力 | 待ち合わせなし指示 | "NW"       | 論理宛先にデータがない場合、入出力状態に"0A"を通知します。                           |
|    | 待ち合わせ指示   | "WT"<br>空白 | 論理宛先にデータがない場合、監視時間の指定に従い、データが書き込まれるのを待ち合わせます。             |
| 出力 | 待ち合わせなし指示 | "NW"       | 論理宛先にデータが最大数まで書き込まれている場合、書きみを中止し、入出力状態に"9G"を通知します。        |
|    | 待ち合わせ指示   | "WT"<br>空白 | 論理宛先にデータが最大数まで書き込まれている場合、監視時間の指定に従い、データが読み込まれるのを待ち合わせます。  |
|    | 強制実行指示    | "NE"       | 論理宛先にデータが最大数まで書き込まれている場合、最大数を超えて書き込みを行い、入出力状態に"0B"を通知します。 |

## データ部(DATA DIVISION)

データ部には、通信に使用するレコードの定義およびファイル管理記述項に指定したデータの定義を記述します。

## 手続き部(PROCEDURE DIVISION)

通信処理には、通常のファイル処理を行うときと同様に、入出力文を使います。入出力文は、次に示す順序で実行します。

1. OPEN文: メッセージの送受信の開始
2. READ文およびWRITE文: メッセージの送受信
3. CLOSE文: メッセージの送受信の終了

### 〔OPEN文およびCLOSE文〕

OPEN文はメッセージの送受信処理の開始時に、CLOSE文はメッセージの送受信処理の終了時に、それぞれ1回だけ実行します。

### 〔READ文またはWRITE文〕

メッセージを送信するときには通信レコードを指定したWRITE文を、メッセージを受信するときには表示ファイルを指定したREAD文を使います。READ文およびWRITE文を実行する前には、DESTINATION-1句に指定したデータ名に論理宛先名を設定しておく必要があります。設定された論理宛先名に対応する宛先がメッセージの送受信の対象となります。

## 入出力エラー処理

入出力エラーの検出方法および入出力エラーが発生したときの実行結果については、“[7.6 入出力エラー処理](#)”を参照してください。



## 21.2.3 プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

## 21.2.4 プログラムの実行

表示ファイル機能を使ったメッセージの送受信を行うプログラムを実行するときには、以下の環境設定が必要です。

- ・ファイル識別名を環境変数情報名として、接続製品名を設定します。[参照]“[5.4.1.70 ファイル識別名 \(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\)](#)”



例

COBOL プログラムのASSIGN句の記述内容

```
ASSIGN TO GS-ACMFILE
```

実行用の初期化ファイルの内容

```
[プログラム名]
:
ACMFILE=, ACM
:
```

COBOL プログラムのASSIGN句に指定したファイル識別名に、接続製品名である"ACM" を設定します。



注意

“ACM”の前に、コンマ(,)を必ず指定してください。

なお、ファイル識別名に接続製品名の指定が省略された場合、環境変数情報@CBR\_PSFILE\_ACMに指定された接続製品が有効となります。

```
@CBR_PSFILE_ACM = ACM
```

設定方法の詳細については、“[5.4.1.43 @CBR\\_PSFILE\\_xxx \(表示ファイルから使用する接続製品名 \(宛先ごとの指定\)\)](#)”を参照してください。



注意

表示ファイル機能を使ったCOBOLプログラムからACM通信機能を使用する場合、COBOLプログラムの実行より前に、ACMの環境を整えておく必要があります。

詳細については、動作環境に応じて、“ACM制御 説明書”、“SymfoWARE7000のRDB\_7000連携ヘルプ”または“SymfoNET説明書 DB連携編”を参照してください。

## 21.2.5 ACM通信機能の利用

ACM通信機能は、表示ファイル機能からの利用以外に、COBOLプログラムからACMが提供するCOBOLサブルーチンを呼び出すCALL文により利用することができます。

CALL文によるインタフェースの詳細については、動作環境に応じて、“ACM制御 説明書”、“SymfoWARE7000のRDB\_7000連携ヘルプ”または“SymfoNET説明書 DB連携編”を参照してください。

## 21.3 簡易アプリ間通信機能の使い方

ここでは、簡易アプリ間通信機能の使い方について説明します。なお、簡易アプリ間通信機能を使ってメッセージ通信を行う例題プログラムがサンプルプログラムとして提供されていますので、参考にしてください。



### 21.3.1 概要

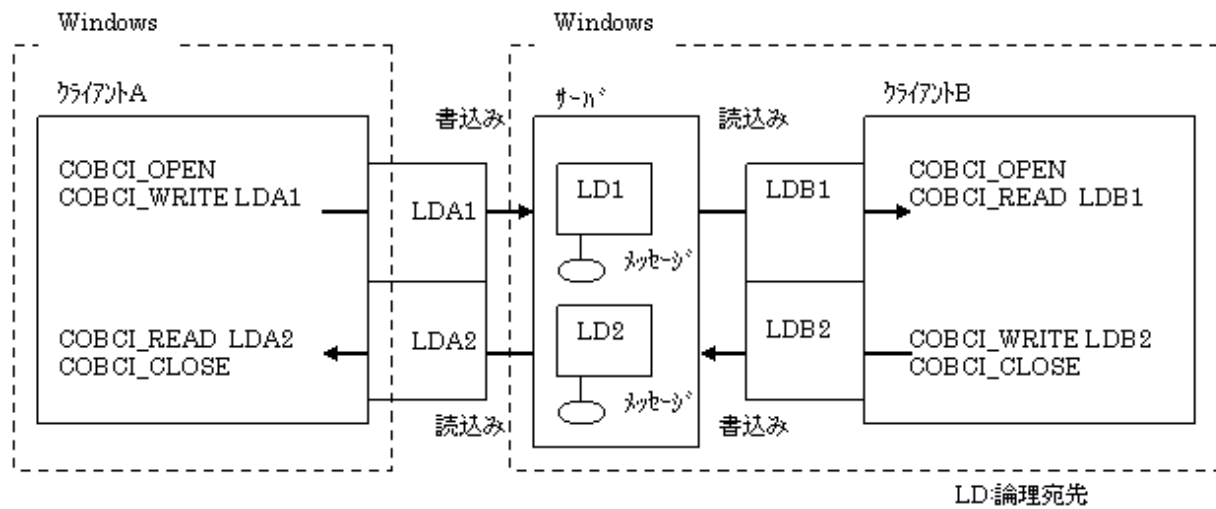
簡易アプリ間通信機能は、プログラム間でメッセージの受渡しを行うための機能です。

プログラム間のメッセージの受渡しは、メッセージを制御する通信システムプロセス(以降サーバと呼びます)の論理宛先を介して行われます。

このため、利用者はあらかじめサーバにメッセージを格納する論理宛先を創成しておく必要があります。また、利用者の送受信プログラム(以降クライアントと呼びます)側では、クライアントを起動する前に、クライアントの論理宛先(自論理宛先)とサーバの論理宛先(相手論理宛先)の関連付けが必要となります。これにより、クライアントの論理宛先に対する読み込み/書き込みは、関連付けられたサーバの論理宛先への読み込み/書き込みとなります。

簡易アプリ間通信の概要を“[図21.2 簡易アプリ間通信の概要図](#)”に示します。

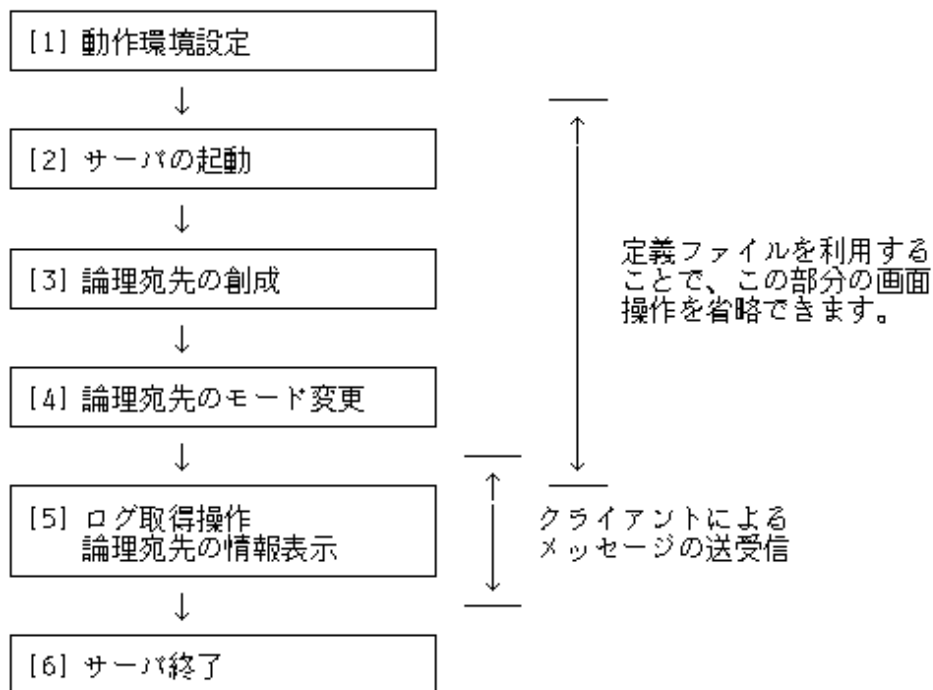
図21.2 簡易アプリ間通信の概要図



### 21.3.2 作業手順

以下に、簡易アプリ間通信を行うときのサーバ側およびクライアント側の作業手順を示します。

## サーバ側の作業手順



### [1] 動作環境設定

簡易アプリ間通信機能は、WindowsソケットAPIにより、TCP/IPプロトコルを使用して通信しています。このため、簡易アプリ間通信機能を使用するためには、WindowsソケットAPIおよびTCP/IPプロトコルが利用できる環境が必要となります。

#### hostsファイルの設定

クライアントが動作するマシンのhostsファイルには、サーバが動作するマシンのホスト名およびIPアドレスが登録されている必要があります。

#### servicesファイルの設定

サーバおよびクライアントが動作する各マシンのservicesファイルには、以下のようなサービス名およびポート番号が登録されている必要があります。

ポート番号は、他のエントリの定義と重ならないように自由を決めることができますが、簡易アプリ間通信機能を使用するLAN間で一貫していることが必要です。

| サービス名 | ポート番号／プロトコル |
|-------|-------------|
| cobci | 20000/tcp   |



### 参考

- TCP/IPプロトコルをインストールすると、hostsファイルおよびservicesファイルが、Windowsをインストールしたフォルダ配下のsystem32¥drivers¥etcフォルダに作成されます。
- 上記ファイルを編集する場合、管理者として起動したエディタ(notepad等)で編集してください。

### [2] サーバの起動 / [3] 論理宛先の創成 / [4] 論理宛先のモード変更

サーバは、“COBCISRV.EXE”を実行することにより起動されます。

画面の操作方法および設定内容については、“[21.3.3.1 \[通信システム環境設定\]ダイアログ](#)”および“[21.3.3.2 簡易アプリ間通信サーバウインドウ](#)”を参照してください。

創成直後の論理宛先には、メッセージの読み込みおよび書き込み許可がないため、必ずモード変更により、読み込みおよび書き込み許可を与える必要があります。この処理完了後、クライアントを起動することができます。

ただし、サーバは、1つのマシン上で複数起動することはできません。

また、[2]～[4]およびログ取得開始操作については、定義ファイルを使用することで、画面上での操作を省略できます。

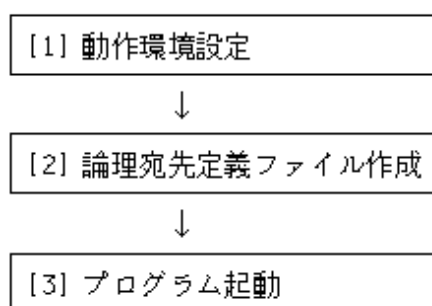
#### [5] ログ取得操作/論理宛先の情報表示

ログ取得操作や論理宛先の情報表示をクライアントの実行時に行うことができます。画面の操作方法および設定内容については、“[21.3.3.2 簡易アプリ間通信サーバウインドウ](#)”を参照してください。

#### [6] サーバ終了

サーバを終了します。このとき、論理宛先の読み込みおよび書き込み許可の取り消し、メッセージの削除、論理宛先の削除は自動で行われます。

### クライアント側の作業手順



#### [1] 動作環境設定

hostsファイルとservicesファイルの設定が必要になります。[参照]サーバ側の作業手順[1] 動作環境設定

#### [2] 論理宛先定義ファイル作成

論理宛先定義ファイル作成ユーティリティ(COBCIU32.EXE)を起動し、論理宛先定義ファイルを作成します。画面の操作方法および設定内容については、“[21.3.4.1 論理宛先定義ファイル作成ユーティリティ](#)”を参照してください。

#### [3] プログラム起動

クライアントは、実行時に論理宛先定義ファイルの情報が必要となります。このため、COBOLの実行用の初期化ファイルに“@CBR\_CIINF=定義ファイル名”を指定する必要があります。また、環境変数に直接指定することもできます。相対パスで指定された場合には、クライアントの実行可能ファイルがあるフォルダからの相対パスとなります。

## 21.3.3 サーバ操作ウインドウ

---

ここでは、サーバで操作するウインドウについて、操作手順に従って説明します。

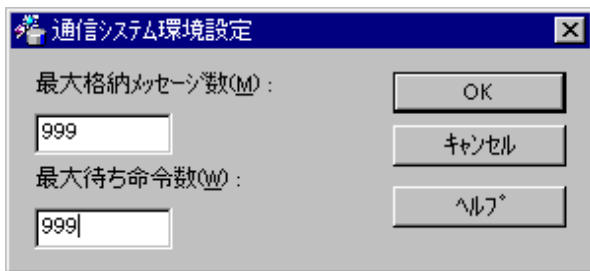
- ・ [通信システム環境設定]ダイアログ
- ・ 簡易アプリ間通信サーバウインドウ

なお、サーバで発生したエラーの表現形式および対処法については、“[21.3.3.2 簡易アプリ間通信サーバウインドウ](#)”の“ログ取得”を参照してください。

### 21.3.3.1 [通信システム環境設定]ダイアログ

サーバを起動し、[通信システム環境設定]ダイアログで以下の情報を設定してください。

設定が完了したら、[OK]ボタンをクリックしてください。



→ サーバが利用できるようになり、簡易アプリ間通信サーバウィンドウが表示されます。

#### [1] (システムの)最大格納メッセージ数[0～999999999]

サーバに格納できるメッセージの最大個数を指定します。  
0が指定された場合、または指定が省略された場合は、最大値が指定されたとみなします。  
指定する値については、“[21.3.5 メモリ資源の見積り](#)”を参考にしてください。

#### [2] (システムの)最大待ち命令数[0～999999999]

サーバで待ち合わせできるクライアントからの読み込み命令および書き込み命令の最大個数を指定します。  
0が指定された場合、または指定が省略された場合は、最大値が指定されたとみなします。  
指定する値については、“[21.3.5 メモリ資源の見積り](#)”を参考にしてください。

### 21.3.3.2 簡易アプリ間通信サーバウィンドウ

簡易アプリ間通信サーバウィンドウでは、論理宛先の操作を行うことができます。

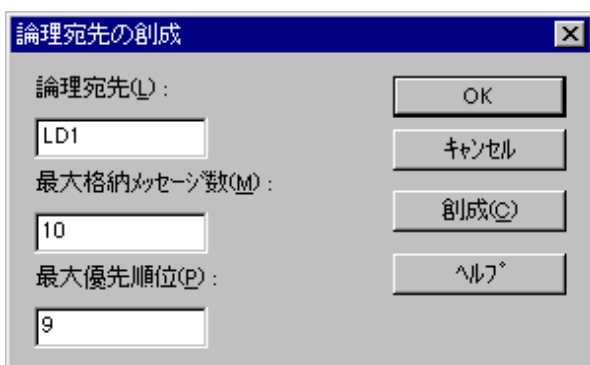


#### 論理宛先作成

[論理宛先]メニューから“作成”を選択します。[論理宛先の作成]ダイアログが表示されるので、以下の情報を設定してください。

設定が完了したら、[OK]ボタンをクリックしてください。

[作成]ボタンをクリックすると、続けて別の論理宛先を作成できます。



→ 論理宛先が作成されます。

#### [1] 論理宛先

作成するサーバ上の論理宛先名を指定します。  
論理宛先名には、8文字以内の英数字を使用できます。  
この論理宛先名は、[クライアント定義]ダイアログで相手論理宛先として指定します。

## [2] 最大格納メッセージ数[0～999999999]

創成する論理宛先に格納できるメッセージの最大個数を指定します。

0が指定された場合、または指定が省略された場合は、[通信システム環境設定]ダイアログで定義した“最大格納メッセージ数”の値が指定されたとみなします。

## [3] 最大優先順位[3～9]

創成する論理宛先に格納できるメッセージに割り当てる優先順位の最大値を指定します。

0～2が指定された場合、または指定が省略された場合は、9が指定されたとみなします。

## 論理宛先削除

[論理宛先]メニューから“削除”を選択します。[論理宛先の削除]ダイアログが表示されるので、削除したい論理宛先を選択し、[OK]ボタンをクリックしてください。

なお、削除する論理宛先は、格納されているメッセージ数が0であり、読み込みおよび書き込みの属性が“不可”である必要があります。

[メッセージ削除]ボタンをクリックすれば、選択された論理宛先に格納されているメッセージを削除できます。

[削除]ボタンをクリックすると、続けて別の論理宛先を削除できます。



→ 論理宛先または格納されているメッセージが削除されます。

## [1] 論理宛先

創成されている論理宛先名を表示します。

## [2] 現メッセージ数

論理宛先に格納されているメッセージの個数を表示します。

## [3] 待ち命令数

論理宛先に待ち合わせしている読み込みおよび書き込み命令の個数を表示します。

## 論理宛先のモード変更

[論理宛先]メニューから“モード変更”を選択します。[論理宛先のモード変更]ダイアログが表示されるので、論理宛先の読み込み/書き込み属性を変更します。

論理宛先名をリストボックスから選択し、読み込みおよび書き込みの“可”または“不可”を選択し、[OK]ボタンをクリックします。

[変更]ボタンをクリックすると、続けて別の論理宛先のモードを変更できます。

論理宛先のモード変更

| 論理宛先(L) | 読込                               | 書込                               |
|---------|----------------------------------|----------------------------------|
| LD1     | <input type="radio"/>            | <input type="radio"/>            |
| LD2     | <input checked="" type="radio"/> | <input checked="" type="radio"/> |
| LD3     | <input checked="" type="radio"/> | <input checked="" type="radio"/> |

OK  
キャンセル  
変更(C)  
ヘルプ

読込(R):  
☒ 可  
☐ 不可

書込(W):  
☒ 可  
☐ 不可

→ 論理宛先の属性が変更されます。

#### [1] 論理宛先

創成されている論理宛先名を表示します。

#### [2] 読込

論理宛先の読込み属性を表示します。○は読込み可、×は読込み不可を示します。

#### [3] 書込

論理宛先の書込み属性を表示します。○は書込み可、×は書込み不可を示します。

#### [4] 読込(R)

論理宛先の読込み属性を“可”にすると、論理宛先からメッセージを読み込むことができます。  
 論理宛先の読込み属性を“不可”にすると、論理宛先からメッセージを読み込むことはできません。

#### [5] 書込(W)

論理宛先の書込み属性を“可”にすると、論理宛先にメッセージを書き込むことができます。  
 論理宛先の書込み属性を“不可”にすると、論理宛先にメッセージを書き込むことはできません。

### 情報表示

〔論理宛先〕メニューから“情報”を選択します。〔論理宛先情報〕ダイアログが表示されます。

〔更新〕ボタンをクリックすると、最新の情報に更新されます。

| 論理宛先情報      |    |     |        |           |        |      |
|-------------|----|-----|--------|-----------|--------|------|
| 通信システム設定情報  |    |     |        |           |        |      |
| 最大格納メッセージ数: |    | 999 |        | 最大待ち命令数:  |        | 999  |
| 論理宛先(L)     | 読込 | 書込  | 最大優先順位 | 最大格納メッセージ | 現メッセージ | 待ち命令 |
| LD1         | ○  | ○   | 9      | 10        | 10     | 0    |
| LD2         | ○  | ×   | 5      | 999       | 0      | 1    |
| LD3         | ×  | ×   | 3      | 100       | 0      | 0    |

更新(R)

キャンセル

ヘルプ

#### [1] 通信システム設定情報

##### 最大格納メッセージ数

サーバに格納できるメッセージの最大個数を表示します。

##### 最大待ち命令数

サーバで待ち合わせできる読込みおよび書込み命令の最大個数を表示します。

#### [2] 論理宛先

創成されている論理宛先名を表示します。

#### [3] 読込

論理宛先の読込み属性を表示します。○は読込み可、×は読込み不可を示します。

#### [4] 書込

論理宛先の書込み属性を表示します。○は書込み可、×は書込み不可を示します。

#### [5] 最大優先順位

論理宛先に格納できるメッセージに割り当てる優先順位の最大値を表示します。

#### [6] 最大格納メッセージ

論理宛先に格納できるメッセージの最大個数を表示します。

#### [7] 現メッセージ

論理宛先に格納されているメッセージの個数を表示します。

#### [8] 待ち命令

論理宛先に待ち合わせしている読込みおよび書込み命令の個数を表示します。

### ログ取得

[論理宛先]メニューから“ログ”を選択します。[ログ取得操作]ダイアログが表示されます。

ログの取得は次の手順で行います。

1. ログを取得するファイル名を指定します。
2. ログ種別の“エラー情報”または“トレース情報”を選択します。

3. [開始]ボタンをクリックします。

→ ログの取得が開始されます。

[終了]ボタンを選択すると、ログの取得を終了します。

## 参考

- ・ ログを取得するファイル名は、絶対パスと相対パスのどちらでも指定できます。相対パスの場合は、サーバのカレントフォルダからのパスとなります。
- ・ 指定されたファイルが存在しない場合は、新規に作成されます。指定されたファイルがすでに存在する場合は、既存のファイルにログ情報が追加されます。

ログの種類には、エラー情報とトレース情報があります。エラー情報では、サーバで発生したエラーがファイルに取得されます。トレース情報では、サーバで発生したエラーおよびクライアントからの処理要求の内容とその処理結果がファイルに取得されます。

サーバで発生したエラーがわかるように、サーバの動作中はエラー情報を取得しておくことをおすすめします。

トレース情報は、出力情報が多いため、クライアントのデバッグ時に使用することをおすすめします。

以下に、ログの出力形式を示します。

< [1] >    [ [2] ]    ID: [3]    Client: [4]    EXE: [5]    [6]

[1] 採取年月日時刻(年/月/日 時:分:秒)

[2] 事象

| 事象       | 意味                     |
|----------|------------------------|
| OPEN 要求  | クライアントから接続要求がありました。    |
| CLOSE 要求 | クライアントから切断要求がありました。    |
| READ 要求  | クライアントから読み込み要求がありました。  |
| WRITE 要求 | クライアントから書き込み要求がありました。  |
| OPEN 終了  | クライアントとの接続処理が終了しました。   |
| CLOSE 終了 | クライアントとの切断処理が終了しました。   |
| READ 終了  | 読み込み処理が終了しました。         |
| WRITE 終了 | 書き込み処理が終了しました。         |
| 通信エラー    | クライアントとの通信でエラーが発生しました。 |
| API エラー  | Windows関数でエラーが発生しました。  |
| 内部矛盾     | 内部矛盾が発生しました。           |

[3] クライアント識別番号



[4] クライアントのIPアドレス

[5] クライアントの実行モジュール名(12バイト)

13バイト目以降は切り捨てられます。

[6] 付加情報

| 事象       | 付加情報                                                                                                                                              |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| OPEN 要求  | —                                                                                                                                                 |
| CLOSE 要求 | —                                                                                                                                                 |
| READ 要求  | COBCI_READの以下のパラメタ<br><ul style="list-style-type: none"><li>・ サーバの論理宛先名</li><li>・ バッファ長</li><li>・ 処理種別</li><li>・ 監視時間</li></ul>                   |
| WRITE 要求 | COBCI_WRITE の以下のパラメタ<br><ul style="list-style-type: none"><li>・ サーバの論理宛先名</li><li>・ メッセージ長</li><li>・ 優先順位</li><li>・ 処理種別</li><li>・ 監視時間</li></ul> |
| OPEN 終了  | エラーコード (rtn:10進表示) (注1)<br>詳細コード (detail: 10進表示) (注1)                                                                                             |
| CLOSE 終了 | エラーコード (rtn:10進表示) (注1)<br>詳細コード (detail: 10進表示) (注1)                                                                                             |
| READ 終了  | エラーコード (rtn:10進表示) (注1)<br>詳細コード (detail: 10進表示) (注1)                                                                                             |
| WRITE 終了 | エラーコード (rtn:10進表示) (注1)<br>詳細コード (detail: 10進表示) (注1)                                                                                             |
| 通信エラー    | Windows Sockets関数名<br>エラーコード (err:10進表示) (注1)<br>エラー内容                                                                                            |
| API エラー  | Windows関数名<br>エラーコード (err:10進表示) (注2)<br>エラー内容                                                                                                    |
| 内部矛盾     | エラー内容                                                                                                                                             |

注1:詳細は、“[21.5 簡易アプリ間通信のサブルーチンのエラーコード](#)”を参照してください。

注2:詳細は、Windows関数のエラーコードまたは“[Visual C++のオンラインヘルプ](#)”を参照してください。



## 参考

— エラー情報には、通信エラー、APIエラーおよび内部矛盾の事象だけが出力されます。

— [3]以降の情報は出力される場合とされない場合があります。

以下にエラー発生時の処置を示します。

| 事象      | 処置                                                                                                                                             |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------|
| 通信エラー   | Windows Sockets関数のエラーコードを参照して対処してください。err の値が0の場合は、クライアント側で通信エラーが発生したと考えられます。<br>詳細は、“ <a href="#">21.5 簡易アプリ間通信のサブルーチンのエラーコード</a> ”を参照してください。 |
| API エラー | メモリ不足 (エラーコード:8) またはディスク資源の不足が原因と考えられますので、動作環境を見直してください。                                                                                       |
| 内部矛盾    | SEに連絡してください。                                                                                                                                   |

## サーバ終了

[論理宛先]メニューから“サーバ終了”を選択します。

“サーバ終了”を選択すると、すべての論理宛先とそこに格納されているメッセージが削除され、ログの取得が停止されます。

→ サーバが終了します。

## 21.3.3.3 サーバ定義ファイルを利用した起動

簡易アプリ間通信サーバは、起動時にファイルから各論理宛先の情報を読み込むことができます。

### ファイルの作成方法

ファイルは、エディタなどを使用して作成してください。

ファイルを使用して起動した後で、画面から論理宛先の情報を変更(新規論理宛先の作成、ログファイル名の変更など)しても、ファイルには反映されませんので、注意してください。

### ファイルの指定方法

COBCISRV.EXEを起動したとき、環境変数情報@CBR\_CI\_SRVINFが指定されていない場合は、COBCISRV.EXEの存在するフォルダにある“COBSVINI.INI”ファイルを使用します。

環境変数情報@CBR\_CI\_SRVINFが指定されていれば、そこに指定されたファイルを使用します。

環境変数情報@CBR\_CI\_SRVINFは、“コントロールパネル”の“システム”のプロパティにある環境変数に設定してください。

環境変数情報@CBR\_CI\_SRVINFに指定されるファイル名には、絶対パス名と相対パス名を使用できます。相対パス名を使用する場合は、簡易アプリ間通信サーバの起動時のカレントフォルダからの相対パスとなります。

指定されたファイルが存在しない場合は、[通信システム環境設定]ダイアログが表示されます。

## サーバ定義ファイルの記述内容

ここでは、サーバ定義ファイルに記述する内容について説明します。

### [SRVINI]セクション

[通信システム環境設定]ダイアログおよび[ログ取得操作]ダイアログで必要な情報を指定します。また、簡易アプリ間通信サーバウィンドウの起動時の状態や終了方法についても指定できます。

#### SysMaxMsg (システム最大格納メッセージ数の指定)

```

SysMaxMsg = n

```

[通信システム環境設定]ダイアログで指定する“最大格納メッセージ数”を指定します。デフォルトは“99999999”です。

#### SysMaxWait (システム待ち命令数の指定)

SysMaxWait = n

---

[通信システム環境設定]ダイアログで指定する“最大待ち命令数”を指定します。デフォルトは“999999999”です。

#### StartMode (サーバ起動時の画面表示の指定)

---

StartMode =  $\left\{ \begin{array}{c} \text{WINDOW} \\ \text{ICON} \end{array} \right\}$

---

サーバ起動時の画面上での表示状態を指定します。

- WINDOW:ウィンドウ画面を表示して起動します。
- ICON:アイコン状態で起動します。

#### IconMenu (アイコンで起動時のシステムメニューの指定)

---

IconMenu =  $\left\{ \begin{array}{c} \text{SYSDEF} \\ \text{CLOSEONLY} \end{array} \right\}$

---

アイコン状態で起動時の、システムメニューの内容を指定します。

- SYSDEF:デフォルトのシステムメニューを表示します。
- CLOSEONLY:システムメニューに“閉じる”だけを表示します。この場合、アイコン状態から通常のウィンドウ状態に戻ること  
はできません。

この項目は、“StartMode”に“ICON”が指定されている場合にかぎり有効になります。

#### EndMode (Windowsの終了時の動作の指定)

---

EndMode =  $\left\{ \begin{array}{c} \text{MANUAL} \\ \text{AUTO} \end{array} \right\}$

---

Windowsの終了(ログオフおよびシャットダウン)時の簡易アプリ間通信サーバの動作を指定します。

- MANUAL:Windowsの終了時、簡易アプリ間通信サーバが起動中であることを示すメッセージを表示します。簡易アプリ間  
通信サーバを画面上で終了させてから、再びWindowsを終了する必要があります。
- AUTO:Windowsの終了とともに、簡易アプリ間通信サーバも終了します。このとき、メッセージは表示されません。

#### Log (ログ取得の有無の指定)

---

Log =  $\left\{ \begin{array}{c} \text{YES} \\ \text{NO} \end{array} \right\}$

---

起動時にログ取得を行うかどうかを指定します。

- YES:ログの取得を行います。
- NO:ログの取得を行いません。

#### LogType (ログ種別の指定)

---

LogType =  $\left\{ \begin{array}{c} \text{ERROR} \\ \text{TRACE} \end{array} \right\}$

---

[ログ取得操作]ダイアログで指定する“ログ種別”を指定します。

- ERROR:エラーログを取得する場合に指定します。
- TRACE:トレースログを取得する場合に指定します。

この項目は、“Log”に“YES”が指定されている場合にかぎり有効になります。

#### LogFileName (ログファイル名の指定)

---

LogFileName = XXXXXXXX.XXX

---

[ログ取得操作]ダイアログで指定する“ログファイル”名を指定します。

ログファイル名には、絶対パス名と相対パス名を使用できます。相対パス名を使用する場合は、カレントフォルダからの相対パスになります。ファイル名に空白を含む場合、ファイル名を二重引用符(”)で囲む必要はありません。

デフォルトは“COBCISRV.LOG”です。

この項目は、“Log”に“YES”が指定されている場合にかぎり有効になります。

#### [LDINF]セクション

[論理宛先の創成]ダイアログおよび[論理宛先のモード変更]ダイアログで必要な情報を指定します。

このセクションの指定方法は、次のとおりです。

---

論理宛先名 = 最大格納メッセージ数, 最大優先順位数, 論理宛先のモード

---

最大格納メッセージ数以降の情報を省略する場合でも、“,”は必ず指定してください。

#### 論理宛先名

[論理宛先の創成]ダイアログで指定する“論理宛先”名を指定します。

この項目には、8文字以内の英数字を使用します。

デフォルトはありません。

### 最大格納メッセージ数

〔論理宛先の創成〕ダイアログで指定する“最大格納メッセージ数”を指定します。  
デフォルトは“システム最大格納メッセージ数(SysMaxMsg)”の値となります。

### 最大優先順位数

〔論理宛先の創成〕ダイアログで指定する“最大優先順位”を指定します。  
デフォルトは“9”です。

### 論理宛先のモード

〔論理宛先のモード変更〕ダイアログで設定する各論理宛先のモードを指定します。サーバの起動時、各論理宛先はそれぞれの指定に従ったモードに設定されます。

- RDWR: 読み込み/書き込み属性がともに“可”になります。
- READ: 読み込み属性が“可”になり、書き込み属性が“不可”になります。
- WRITE: 読み込み属性が“不可”になり、書き込み属性が“可”になります。

デフォルトは“RDWR”です。

定義ファイル内の;(セミコロン)で始まる文は、注釈文として解釈されます。



### 例

サーバ定義ファイルの例

```
[SRVIN]
; システム最大格納メッセージ数
SysMaxMsg=100
; システム待ち命令数
SysMaxWait=50
; サーバの起動時の状態
StartMode=ICON
; アイコン起動時のシステムメニュー
IconMenu=
; 終了方法
EndMode=AUTO
; ログ取得の有無
Log=YES
; ログ種別
LogType=TRACE
; ログファイル名
LogFile=LOGDATA.TXT

[LDINF]
; 論理宛先名= 最大格納メッセージ数, 最大優先順位数, 論理宛先のモード
LD1=100, 9, READ
LD2=, , WRITE
LD3=500, ,
LD4=, ,
```

## 21.3.4 クライアント操作ウィンドウ

ここでは、クライアントで操作する論理宛先定義ファイル作成ユーティリティについて、操作手順に従って説明します。

### 21.3.4.1 論理宛先定義ファイル作成ユーティリティ

論理宛先定義ファイル作成ユーティリティ(COBCIU32.EXE)を起動すると、〔定義ファイルの選択〕ダイアログが表示されるので、作成する定義ファイルの名前を指定し、〔OK〕ボタンをクリックしてください。

なお、指定されたファイルが存在しない場合は新規にファイルが作成され、既存のファイルが指定された場合は既存のファイルに情報

が追加されます。  
→ [クライアント定義]ダイアログが表示されます。

#### [1] サーバ名

クライアントとメッセージのやりとりを行うサーバに対する論理的な名前を指定します。この名前は、利用者が自由に決めることができます。サーバ名には、15文字以内の英数字を使用できます。

#### [2] 相手マシン名

サーバが動作しているマシンのホスト名を指定します。相手マシン名には、15文字以内の英数字を使用できます。

#### [3] 自論理宛先

クライアント側の論理宛先名を指定します。論理宛先名には、8文字以内の英数字を使用できます。

#### [4] 相手論理宛先

自論理宛先と関連付けるサーバの論理宛先名を指定します。論理宛先名には、8文字以内の英数字を使用できます。

- サーバ名を指定してから、相手マシン名、自論理宛先名および相手論理宛先名の設定を行うことができます。選択されたサーバ名に対して、すでに定義されている情報(相手マシン名、自論理宛先名または相手論理宛先名)がある場合は、その情報が表示されます。
- 1つのサーバ名に対して1つの相手マシン名を指定できます。1つのサーバ名に対して、自論理宛先名および相手論理宛先名は複数指定できます。
- 新しいサーバ名を登録する場合、または選択されたサーバ名に対するマシン名を修正する場合は、[登録]ボタンをクリックします。
- すでに登録されているサーバ名に新しく自論理宛先と相手論理宛先を関連付ける場合、または既存の論理宛先名を修正する場合は、[追加]ボタンをクリックします。
- 論理宛先の関連付けを解除する場合は、[論理宛先情報]グループボックス内の[削除]ボタンをクリックします。
- [設定内容]グループボックスの[削除]ボタンをクリックすると、選択されているサーバ名に対応するすべての情報(相手マシン名、自論理宛先および相手論理宛先)を削除できます。
- 指定されたファイルが存在しない場合は、新規に作成します。既存のファイルが指定された場合は、既存のファイルに情報を追加します。



#### 例

このユーティリティで作成される論理宛先定義ファイルの例

```

[SERVERNAME_DEF_LIST]
SERVER1=SERVERNAME -- |
SERVER2=SERVERNAME | [1] サーバ名= SERVERNAME
SERVER3=SERVERNAME -- |

[SERVER1] -----[2] [サーバ名]
@HOSTNM=SRV1 -----[3] @HSTNM= 相手マシン名

[SERVER1.LD] -----[4] [サーバ名.LD]
MYLD1=LD1 -- |
MYLD2=LD2 | [5] 自論理宛先名= 相手論理宛先名
MYLD3=LD3 -- |

[SERVER2]
:
[SERVER2.LD]
:
[SERVER3]
:
[SERVER3.LD]
:

```



## 注意

相手マシン名にIPアドレスを指定することはできません。IPアドレスへの名前解決可能なホスト名を指定してください。

本機能において、以下の範囲でIPv6アドレスに対応しています。

表21.4 簡易アプリ間通信機能のIPv6対応

| 項目               | 仕様                                                                                                                                                                                                         |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 対応可能なネットワーク      | <p>以下の環境に対応しています。</p> <ul style="list-style-type: none"> <li>IPv4/IPv6 デュアルスタック環境</li> <li>IPv6 のみの環境</li> </ul>                                                                                           |
| 使用可能なIPv6アドレス    | <p>以下のIPv6 アドレスが使用できます。</p> <ul style="list-style-type: none"> <li>グローバルユニキャストアドレス(GUA)</li> <li>ユニークローカルアドレス(ULA)</li> <li>リンクローカルアドレス(LLA)</li> </ul> <p>また、IP アドレスのインターフェースID 部は任意をサポートしています。</p>       |
| IPv4/IPv6のアドレス選択 | <p>IPv4/IPv6デュアルスタック環境においてはIPv6を優先的に使用します。</p> <p>以下は、対応していません。</p> <ul style="list-style-type: none"> <li>IPv4のみまたはIPv6のみの使用限定</li> <li>優先順位の切り替え</li> <li>IPv6による接続に失敗した場合のIPv4による接続(フォールバック)</li> </ul> |
| IP アドレスの表記       | <p>以下のIPv6アドレス表記をサポートしています。</p> <ol style="list-style-type: none"> <li>ポート番号の指定([ ]の有無)</li> <li>通常の16進法による表記</li> <li>表記の単純化ルール(RFC5952)による表記</li> <li>英字部の大文字表記</li> </ol>                               |

| 項目 | 仕様                                                                                                                               |
|----|----------------------------------------------------------------------------------------------------------------------------------|
|    | 5. 英字部の小文字表記<br>6. 英字部の大小文字混在表記<br>IPv4/IPv6 デュアルスタックOS 時やIPv4/IPv6 併用時(デュアルスタック時)にIPv4 アドレスを指定する場合は、従来どおりのIPv4 アドレス表記を使用してください。 |



一定時間ごとに変わる一時アドレス(プライバシ拡張)は、サポートしていません。IP アドレスが変更された場合は、簡易アプリ間通信機能のサーバとの接続を切断したあと、サーバとの接続を再度確立する必要があります。

## 21.3.5 メモリ資源の見積り

ここでは、簡易アプリ間通信機能のサーバで使用するメモリの見積り方について説明します。

### 論理宛先に格納されているメッセージの使用メモリ

論理宛先に書き込まれたメッセージは、1つのメッセージに160バイトの制御域が付加されて管理されます。このため、1つのメッセージの格納に必要なメモリは、メッセージ長に160バイトを加えた大きさとなります。

### 待ち合わせしている命令の使用メモリ

待ち合わせしている読み込み命令は、160バイトの制御域で管理されます。このため、1つの待ち合わせしている読み込み命令について、160バイトのメモリが必要となります。

待ち合わせしている書き込み命令は、論理宛先に格納されているメッセージと同じく、メッセージに160バイトの制御域が付加されて管理されます。このため、1つの待ち合わせしている書き込み命令について、メッセージ長に160バイトを加えた大きさのメモリが必要となります。

したがって、サーバが使用するメモリの算出式は以下のようになります。

サーバの最大使用メモリは、システムの最大格納メッセージ数およびシステムの最大待ち命令数に左右されるため、サーバが使用するメモリの算出式を参考に決定してください。

$$\text{サーバ使用メモリ} = M \times (160 + ML) + WR \times 160 + WW \times (160 + ML)$$

ただし、以下の条件が成り立ちます。

|    |   |                        |
|----|---|------------------------|
| MM | ≤ | M                      |
| MW | ≤ | WR + WW                |
| M  | : | 論理宛先に格納されているメッセージ数 (個) |
| ML | : | メッセージ長 (バイト)           |
| WR | : | 待ち合わせしている読み込み命令数 (個)   |
| WW | : | 待ち合わせしている書き込み命令数 (個)   |
| MM | : | システムの最大格納メッセージ数 (個)    |
| MW | : | システムの最大待ち命令数 (個)       |

システムの最大格納メッセージ数およびシステムの最大待ち命令数については、“[21.3.3.1 \[通信システム環境設定\]ダイアログ](#)”および“[21.3.3.3 サーバ定義ファイルを利用した起動](#)”を参照してください。

## 21.3.6 注意事項

ここでは、簡易アプリ間通信機能使用時の注意事項について説明します。

### 簡易アプリ間通信のバージョンについて

簡易アプリ間通信のサーバモジュール(COBCISRV.EXE)とクライアントモジュール(F3BICICL.DLL)は、必ず同一バージョンレベルを使用してください。



## クライアントアプリの強制終了時の動作について

クライアントアプリが読み待ちまたは書き込み待ちの状態になっている場合に、そのクライアントアプリの動作している端末の電源を切断されると、サーバ内の待ち命令が残ったままになります。この状態でのサーバの動作は保証されません。

クライアントアプリのアイコンを“閉じる”ことで強制終了した場合は、そのクライアントアプリに対応した待ち命令は自動的にサーバ内から削除されます。

## 簡易アプリ間通信サーバウィンドウ終了時の残存メッセージについて

簡易アプリ間通信サーバウィンドウを終了したときに、サーバ上に存在するメッセージはすべて削除されます。

## セキュリティについて

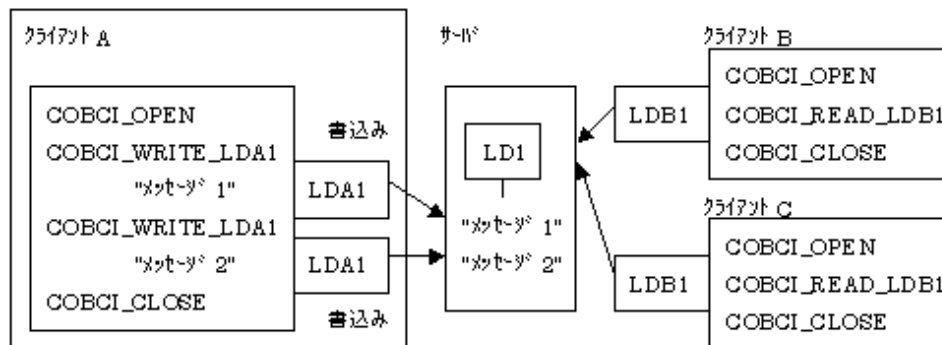
簡易アプリ間通信機能は、イントラネット環境下で使用してください。

## 複数のクライアントからひとつの論理宛先にメッセージを送受信する場合について

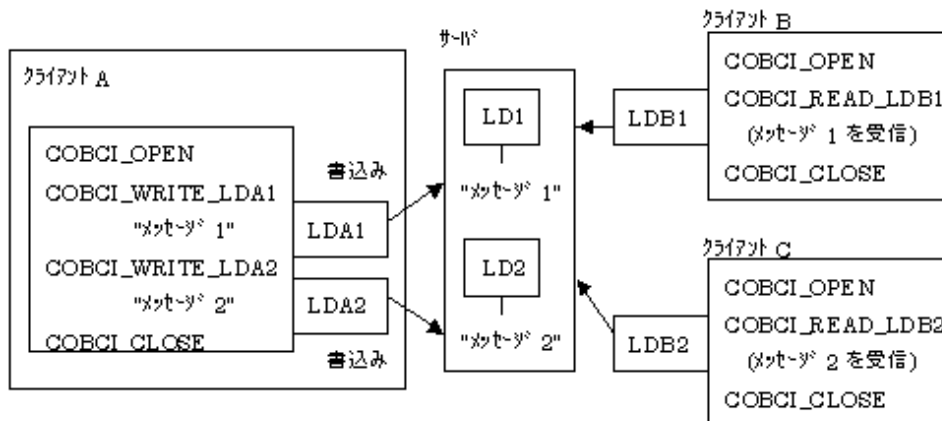
サーバに作成したひとつの論理宛先を複数のクライアントで共有し、複数のクライアントからメッセージの書き込み・読み込みを行った場合、以下の理由から、意図したクライアントにメッセージが伝わらない場合があります。

- 論理宛先に対するメッセージの書き込み・読み込みが行われるタイミングと、各クライアントで実行したCOBCI\_READ、COBCI\_WRITEサブルーチンのタイミングとの同期は、OS仕様やネットワークなどの動作環境に依存するため、保証されません。
- 論理宛先に対するメッセージの読み込みが行われると、メッセージはサーバから削除されます。そのため、複数のクライアントが同じメッセージを受信することはできません。  
また、読み込むクライアントを特定することはできません。

下図では、ひとつの論理宛先を複数のクライアントで共有しています。このような場合、クライアントBとクライアントCがメッセージを読み込む順番は、保証されません。読み込む順番によって、クライアントBが「メッセージ1」を受信する時と「メッセージ2」を受信する場合があります。



クライアントAから特定のクライアントへメッセージを送信したい場合は、下図のようにクライアントごとに論理宛先を作成します。そして、それぞれの宛先に対してメッセージを送受信します。



# 21.4 簡易アプリ間通信のサブルーチン

ここでは、簡易アプリ間通信で使用するサブルーチンについて説明します。

簡易アプリ間通信機能のクライアントは、以下のサブルーチンをCALL文で呼び出すことにより、サーバを介してクライアント間でメッセージの受渡しを行うことができます。

| サブルーチン名     | 機能               |
|-------------|------------------|
| COBCI_OPEN  | サーバとの接続の確立       |
| COBCI_CLOSE | サーバとの接続の切断       |
| COBCI_READ  | サーバからのメッセージの読み込み |
| COBCI_WRITE | サーバへのメッセージの書き込み  |

## 【呼出し順序】



## 例

### アプリケーション内での使用方法の例

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. プログラム名.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 STATUS-REC.
 05 STATUS-ERROR PIC S9(9) COMP-5.
 05 STATUS-DETAIL PIC S9(9) COMP-5.
01 SERVERNAME-REC PIC X(15).
01 SERVERHD-REC PIC S9(9) COMP-5.
01 UNUSED-REC PIC S9(9) COMP-5 VALUE 0.
01 LDNAME-REC PIC X(8).
01 BUFFER-REC PIC X(n).
01 RECEIVETYPE-REC.
 05 BUFFERLEN PIC S9(9) COMP-5.
 05 MSGLEN PIC S9(9) COMP-5.
 05 MSGTYPE PIC S9(9) COMP-5.
 05 WAITTIME PIC S9(9) COMP-5.
 05 UNUSED PIC X(16) VALUE LOW-VALUE.
01 SENDTYPE-REC.
 05 MSGLEN PIC S9(9) COMP-5.
 05 PRIORITY PIC S9(9) COMP-5.
 05 MSGTYPE PIC S9(9) COMP-5.
 05 WAITTIME PIC S9(9) COMP-5.
 05 UNUSED PIC X(16) VALUE LOW-VALUE.
PROCEDURE DIVISION.
```

```

 :
* [サーバとの接続]
 CALL "COBCI_OPEN" WITH C LINKAGE
 USING BY REFERENCE STATUS-REC
 BY REFERENCE SERVERNAME-REC
 BY REFERENCE SERVERHD-REC
 BY VALUE UNUSED-REC.
 :
* [メッセージの読み込み]
 CALL "COBCI_READ" WITH C LINKAGE
 USING BY REFERENCE STATUS-REC
 BY VALUE SERVERHD-REC
 BY REFERENCE LDNAME-REC
 BY REFERENCE RECEIVETYPE-REC
 BY REFERENCE BUFFER-REC
 BY VALUE UNUSED-REC.
 :
* [メッセージの書き込み]
 CALL "COBCI_WRITE" WITH C LINKAGE
 USING BY REFERENCE STATUS-REC
 BY VALUE SERVERHD-REC
 BY REFERENCE LDNAME-REC
 BY REFERENCE SENDTYPE-REC
 BY REFERENCE BUFFER-REC
 BY VALUE UNUSED-REC.
 :
* [サーバとの切断]
 CALL "COBCI_CLOSE" WITH C LINKAGE
 USING BY REFERENCE STATUS-REC
 BY VALUE SERVERHD-REC
 BY VALUE UNUSED-REC.
 :
END PROGRAM プログラム名.

```

## 注意事項

- 動的プログラム構造で当サブルーチンを呼び出す場合、以下のようなエントリ情報が必要になります。エントリ情報の指定方法については、“[5.4.2 副プログラムのエントリ情報](#)”を参照してください。

```

[ENTRY]
COBCI_OPEN=F3BICICL.DLL
COBCI_CLOSE=F3BICICL.DLL
COBCI_READ=F3BICICL.DLL
COBCI_WRITE=F3BICICL.DLL

```

- 動的リンク構造で当サブルーチンを呼び出す場合、呼出し元プログラムのリンク時に、F3BICIMP.LIBを結合してください。

## 21.4.1 COBCI\_OPEN

ここでは、COBCI\_OPENサブルーチンについて説明します。

### 機能

クライアントとサーバの接続を確立します。サーバは、サーバ名によって識別されます。クライアントとサーバの接続が成功すると、サーバ識別子が返されます。

### 呼出し形式

```

 :
DATA DIVISION.
 WORKING-STORAGE SECTION.

```

```

01 STATUS-REC.
 05 STATUS-ERROR PIC S9(9) COMP-5.
 05 STATUS-DETAIL PIC S9(9) COMP-5.
01 SERVERNAME-REC PIC X(15).
01 SERVERHD-REC PIC S9(9) COMP-5.
01 UNUSED-REC PIC S9(9) COMP-5 VALUE 0.

PROCEDURE DIVISION.

 CALL "COBCI_OPEN" WITH C LINKAGE
 USING BY REFERENCE STATUS-REC
 BY REFERENCE SERVERNAME-REC
 BY REFERENCE SERVERHD-REC
 BY VALUE UNUSED-REC.

 :

```

## パラメタ

### STATUS-REC

STATUS-ERRORにはエラーコード、STATUS-DETAILには詳細コードをそれぞれ返します。

### SERVERNAME-REC

クライアントが接続するサーバのサーバ名を指定します。サーバ名は、論理宛先定義ファイルに定義されている必要があります。サーバ名が15バイトに満たない場合、末尾には空白が必要となります。

### SERVERHD-REC

サーバ識別子を返します。

### UNUSED-REC

将来拡張のための予約域であり、0が設定されている必要があります。

## 戻り値

サーバとの接続に成功した場合は、特殊レジスタPROGRAM-STATUSに0を返します。  
サーバとの接続に失敗した場合は、特殊レジスタPROGRAM-STATUSに-1を返します。

## 21.4.2 COBCI\_CLOSE

ここでは、COBCI\_CLOSEサブルーチンについて説明します。

## 機能

クライアントとサーバを切断します。サーバはCOBCI\_OPENで返されたサーバ識別子によって識別されます。

## 呼出し形式

```

:
DATA DIVISION.
WORKING-STORAGE SECTION.
01 STATUS-REC.
 05 STATUS-ERROR PIC S9(9) COMP-5.
 05 STATUS-DETAIL PIC S9(9) COMP-5.
01 SERVERHD-REC PIC S9(9) COMP-5.
01 UNUSED-REC PIC S9(9) COMP-5 VALUE 0.

PROCEDURE DIVISION.

 CALL "COBCI_CLOSE" WITH C LINKAGE
 USING BY REFERENCE STATUS-REC
 BY VALUE SERVERHD-REC

```

⋮

## STATUS-REC

SERVERHD-REC

UNUSED-REC

•

•

$$\vdots$$

## パラメタ

### STATUS-REC

STATUS-ERRORにはエラーコード、STATUS-DETAILには詳細コードをそれぞれ返します。

### SERVERHD-REC

COBCI\_OPENの呼出しで返されたサーバ識別子を指定します。

### LDNAME-REC

メッセージを読み込むクライアント側の論理宛先名を指定します。論理宛先名が8バイトに満たない場合は、末尾に空白が必要となります。

この論理宛先名は、[クライアント定義]ダイアログで自論理宛先として指定します。

### RECEIVETYPE-REC

BUFFERLENには、メッセージを受け取るレコード領域の長さをバイト数で指定します。BUFFERLENの最大長は、32000バイトです。

MSGLENには、論理宛先から読み込んだメッセージのデータ長を返します。

MSGTYPEには、処理種別を指定します。処理種別には、以下のものがあります。

#### 待ち合わせなし指示

MSGTYPEに0を指定することにより、待ち合わせなし指示となります。この場合、論理宛先にメッセージが存在しないためにメッセージが読み込めなければ、COBCI\_READは即時にエラー復帰します。このとき、その旨のエラーコードを返します。

#### 待ち合わせ指示

MSGTYPEに1を指定することにより、待ち合わせ指示となります。この場合、サーバの最大待ち命令数以内で、論理宛先にメッセージが書き込まれるまで待ち合わせることができます。このとき、同時にWAITTIMEに監視時間を指定できます。監視時間は0～999999999(秒)の範囲で指定することができ、0を指定した場合は、無限待ち合わせとなります。

指定した時間内に論理宛先にメッセージが書き込まれなければ、COBCI\_READはエラー復帰します。このとき、その旨のエラーコードを返します。

WAITTIMEには、MSGTYPEに待ち合わせ指示を指定した場合の監視時間を指定します。

UNUSEDは、将来拡張のための予約域であり、0が設定されている必要があります。

### BUFFER-REC

論理宛先のメッセージを受け取るレコード領域を指定します。ここに、メッセージ本文を返します。MSGLENに返された長さ以降の領域の内容は保証されません。

読み込んだメッセージがBUFFERLENに指定された長さより大きい場合は、BUFFERLENを超えた後ろのデータは切り捨てられます。

### UNUSED-REC

将来拡張のための予約域であり、0が設定されている必要があります。

## 戻り値

メッセージの読み込みに成功した場合は、特殊レジスタPROGRAM-STATUSに0を返します。

メッセージの読み込みに失敗した場合は、特殊レジスタPROGRAM-STATUSに-1を返します。



## 参考

監視時間のタイムアウトのチェックには、WM\_TIMERメッセージを使用しているため、実際に経過した時間と指定された監視時間に差がある場合があります。



## 注意

WAITTIMEに0を指定し、無限待ち合わせを行う場合は以下の対処が必要です。

- ・ 次を示すウィンドウを利用する機能は使用することができません。
  - － DISPLAY文/ACCEPT文の実行によるCOBOLコンソールへのデータ小入出力
  - － スクリーン操作機能
  - － 表示ファイル(画面)
- ・ 環境変数情報@ShowIcon=NOを指定し、NetCOBOLのアイコンの表示を抑止してください。

## 21.4.4 COBCI\_WRITE

ここでは、COBCI\_WRITEサブルーチンについて説明します。

### 機能

サーバの論理宛先にメッセージを書き込みます。書き込むメッセージの優先順位を指定できます。論理宛先に空きがない場合、処理種別の指定により処理の待ち合わせ方法を指定できます。

### 呼出し形式

```

 :
DATA DIVISION.
WORKING-STORAGE SECTION.
01 STATUS-REC.
 05 STATUS-ERROR PIC S9(9) COMP-5.
 05 STATUS-DETAIL PIC S9(9) COMP-5.
01 SERVERHD-REC PIC S9(9) COMP-5.
01 LDNAME-REC PIC X(8) .
01 SENDTYPE-REC.
 05 MSGLEN PIC S9(9) COMP-5.
 05 PRIORITY PIC S9(9) COMP-5.
 05 MSGTYPE PIC S9(9) COMP-5.
 05 WAITTIME PIC S9(9) COMP-5.
 05 UNUSED PIC X(16) VALUE LOW-VALUE.
01 BUFFER-REC PIC X(n) .
01 UNUSED-REC PIC S9(9) COMP-5 VALUE 0.

PROCEDURE DIVISION.

 CALL "COBCI_WRITE" WITH C LINKAGE
 USING BY REFERENCE STATUS-REC
 BY VALUE SERVERHD-REC
 BY REFERENCE LDNAME-REC
 BY REFERENCE SENDTYPE-REC
 BY REFERENCE BUFFER-REC
 BY VALUE UNUSED-REC.
 :

```

### パラメタ

#### STATUS-REC

STATUS-ERRORにはエラーコード、STATUS-DETAILには詳細コードをそれぞれ返します。

#### SEVERHD-REC

COBCI\_OPENの呼出しで返されたサーバ識別子を指定します。

#### LDAME-REC

メッセージを書き込むクライアント側の論理宛先名を指定します。論理宛先名が8バイトに満たない場合は、末尾に空白が必要となります。

この論理宛先名は、[クライアント定義]ダイアログで自論理宛先として指定します。

## SENDTYPE-REC

MSGLENには、論理宛先に書き込むメッセージのデータ長を指定します。書き込むことができる最大データ長は、32000バイトです。

PRIORITYには、論理宛先に書き込むメッセージの優先順位を指定します。メッセージは、優先順位の高いものから読み込まれます。優先順位が同一のメッセージは、書き込まれた順に読み込まれます。

優先順位は1～9の範囲であり、1が最高の優先順位です。指定した優先順位がサーバの論理宛先にはない場合は、その値に最も近い優先順位に格納されます。ただし、負の値は指定できません。

MSGTYPEには、処理種別を指定します。処理種別には、以下のものがあります。

### 待ち合わせなし指示

MSGTYPEに0を指定することにより、待ち合わせなし指示となります。この場合、論理宛先に空きがなく、メッセージを書き込めなければ、COBCL\_WRITEは即時にエラー復帰します。このとき、その旨のエラーコードを返します。

### 待ち合わせ指示

MSGTYPEに1を指定することにより、待ち合わせ指示となります。この場合、サーバの最大待ち命令数以内で、論理宛先に空きができるまで待ち合わせることができます(待ち合わせることができるのは、システムの最大待ち命令数以内)。このとき、同時にWAITTIMEに監視時間を指定できます。監視時間は0～999999999(秒)の範囲で指定することができ、0を指定した場合は、無限待ち合わせとなります。

指定した時間内に論理宛先に空きができなければ、COBCL\_WRITEはエラー復帰します。このとき、その旨のエラーコードを返します。

### 強制実行指示

MSGTYPEに2を指定することにより、強制実行指示となります。この場合、論理宛先の最大格納メッセージ数を超えてメッセージの書き込みを行うことができます。論理宛先の最大格納メッセージ数を超えてメッセージの書き込みを行った場合、COBCL\_WRITEの復帰時にその旨のエラーコードを返します。ただし、サーバの最大格納メッセージ数を超えてメッセージの書き込みを行うことはできません。

WAITTIMEには、MSGTYPEに待ち合わせ指示を指定した場合の監視時間を指定します。

UNUSEDは、将来拡張のための予約域であり、0が設定されている必要があります。

## BUFFER-REC

論理宛先に書き込むメッセージを格納したレコード領域を指定します。

## UNUSED-REC

将来拡張のための予約域であり、0が設定されている必要があります。

## 戻り値

メッセージの書き込みに成功した場合は、特殊レジスタPROGRAM-STATUSに0を返します。

メッセージの書き込みに失敗した場合は、特殊レジスタPROGRAM-STATUSに-1を返します。



### 参考

監視時間のタイムアウトのチェックには、WM\_TIMERメッセージを使用しているため、実際に経過した時間と指定された監視時間に差がある場合があります。

## 21.5 簡易アプリ間通信のサブルーチンのエラーコード

ここでは、簡易アプリ間通信機能で使用するサブルーチンのエラーコードについて説明します。

サブルーチンが正常に終了した場合、特殊レジスタPROGRAM-STATUSに0を返し、エラーコード(STATUS-RECのSTATUS-ERROR)および詳細コード(STATUS-RECのSTATUS-DETAIL)にも0を返します。エラーが発生した場合は、特殊レジスタPROGRAM-STATUSに-1を返し、以下に示すエラーコードおよび詳細コードを返します。なお、表中の“対象サブルーチン”の記号の意味は以下のとおりです。

- O: COBCL\_OPEN



- ・ C: COBCI\_CLOSE
- ・ R: COBCI\_READ
- ・ W: COBCI\_WRITE

表21.5 簡易アプリ間通信のサブルーチンのエラーコード一覧

| エラーコード | 詳細コード | 意味                                                                                  | 処置                             | 対象サブルーチン |   |   |   |
|--------|-------|-------------------------------------------------------------------------------------|--------------------------------|----------|---|---|---|
|        |       |                                                                                     |                                | O        | C | R | W |
| 0      | 0     | 正常終了しました。                                                                           | —                              | ○        | ○ | ○ | ○ |
| 1      | 257   | メッセージの読込みに成功しましたがメッセージのデータ長よりバッファ長が小さいため、バッファ長以降のデータは切り捨てられました。                     | —                              |          |   | ○ |   |
| 2      | 513   | 読み込むメッセージがありません。(待ち合わせなし)                                                           | —                              |          |   | ○ |   |
| 3      | 769   | 論理宛先内最大格納メッセージ数を超えてメッセージを格納しました。(強制実行)                                              | —                              |          |   |   | ○ |
| 4      | 1025  | 監視時間が経過しましたが、論理宛先にメッセージがなかったため、メッセージの読込みに失敗しました。(待ち合わせあり)                           | —                              |          |   | ○ |   |
|        | 1026  | 監視時間が経過しましたが、論理宛先内最大格納メッセージ数、またはシステム内最大格納メッセージ数を越えていたため、メッセージの書込みに失敗しました。(待ち合わせあり)  | 不用な未処理メッセージを削除してから実行してください。    |          |   |   | ○ |
| 6      | 1537  | 論理宛先が読込み可能状態ではないため、メッセージの読込みに失敗しました。                                                | サーバの論理宛先を読込み可能状態に変更してください。     |          |   | ○ |   |
|        | 1538  | 論理宛先が書込み可能状態ではないため、メッセージの書込みに失敗しました。                                                | サーバの論理宛先を書込み可能状態に変更してください。     |          |   |   | ○ |
|        | 1539  | 創成されていない論理宛先に対して、メッセージの読込み／書込みを行いました。                                               | サーバに論理宛先を創成してください。             |          |   | ○ | ○ |
| 7      | 1793  | 論理宛先内最大格納メッセージ数を越えたため、メッセージの書込みに失敗しました。(待ち合わせなし)                                    | 不用な未処理メッセージを削除してから実行してください。    |          |   |   | ○ |
| 8      | 2049  | システム内最大格納メッセージ数を越えたため、メッセージの書込みに失敗しました。または、システム内最大待ち命令数を越えたため、メッセージの読込み／書込みに失敗しました。 | 不用な未処理メッセージを削除してから実行してください。    |          |   | ○ | ○ |
| 10     | 2561  | すでに接続されているサーバに対して再度接続要求されました。                                                       | 同一サーバに対して再度接続要求していないか確認してください。 | ○        |   |   |   |
| 11     | 2833  | 論理宛先名に誤りがあります。                                                                      | 論理宛先名指定に誤りがないか確認してください。        |          |   | ○ | ○ |
|        | 2834  | 処理種別に誤りがあります。                                                                       | 処理種別の指定に誤りがないか確認してください。        |          |   | ○ | ○ |

| エラーコード | 詳細コード  | 意味                                                               | 処置                                                                                              | 対象サブルーチン |   |   |   |
|--------|--------|------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|----------|---|---|---|
|        |        |                                                                  |                                                                                                 | O        | C | R | W |
|        | 2835   | メッセージ長／バッファ長に誤りがあります。                                            | メッセージ長／バッファ長の指定に誤りがないか確認してください。                                                                 |          |   | ○ | ○ |
|        | 2836   | 優先順位に誤りがあります。                                                    | 優先順位の指定に誤りがないか確認してください。                                                                         |          |   |   | ○ |
|        | 2837   | 監視時間に誤りがあります。                                                    | 監視時間の指定に誤りがないか確認してください。                                                                         |          |   | ○ | ○ |
|        | 2838   | サーバ名に誤りがあります。                                                    | サーバ名指定に誤りがないか確認してください。                                                                          | ○        |   |   |   |
|        | 2839   | サーバ識別子に誤りがあります。                                                  | サーバに接続されているか、サーバ識別子の指定に誤りがないか確認してください。                                                          |          | ○ | ○ | ○ |
| 12     | 3105   | 論理宛先定義ファイルが、正しく指定されていません。                                        | @CBR_CIINFに論理宛先定義ファイルが正しく設定されているか確認してください。                                                      | ○        |   |   |   |
|        | 3106   | 論理宛先定義ファイル内の指定に誤りがあります。                                          | 論理宛先定義ファイルの記述が正しいか、または定義ファイルに指定されている相手マシン名(@HOSTNMに設定されているホスト名)がhostsファイルに正しく定義されているか、確認してください。 | ○        |   | ○ | ○ |
| 13     | XXXX   | システム間通信異常が発生しました。(詳細コードXXXXはWindows Sockets関数のエラーコードを示しています。(注)) | このエラーが発生した場合、サーバとの接続が切断されます。通信を続けるにはサーバとの接続(COBCI_OPEN)から再度行ってください。                             | ○        | ○ | ○ | ○ |
| 14     | 3633   | メモリ資源不足が発生しました。                                                  | 動作環境の見直しを行ってください。                                                                               | ○        | ○ | ○ | ○ |
| 15     | 3842   | サーバとの接続が切断されました。                                                 | 通信を続ける場合には、サーバとの接続(COBCI_OPEN)から再度行ってください。                                                      | ○        | ○ | ○ | ○ |
|        | 3843   | 使用中のソケットがWindows Sockets V1.1に対応していません。                          | Windows Sockets V1.1に対応したものを使用してください。                                                           | ○        |   |   |   |
|        | 3844   | クライアントが強制終了されました。                                                | —                                                                                               | ○        | ○ | ○ | ○ |
|        | 4080以上 | 内部矛盾を検出しました。                                                     | SEに連絡してください。                                                                                    | ○        | ○ | ○ | ○ |

○:通知される

注:詳細は、“表21.6 WindowsSockets関数のエラーコード一覧”を参照してください。

表21.6 WindowsSockets関数のエラーコード一覧

| WINSOCK.H内の表記      | エラーコード | 意味                                                                        |
|--------------------|--------|---------------------------------------------------------------------------|
| WSAEMFILE          | 10024  | 使用可能なソケット識別子をすべて使い果たしたため、新しいソケット識別子を創成できません。不要な通信接続を終了させてから、再度処理を行ってください。 |
| WSAENOTSOCK        | 10038  | ソケット識別子でないものがソケット関数に指定されています。                                             |
| WSAEPROTONOSUPPORT | 10043  | 指定されたプロトコルはサポートされていません。TCP/IP動作環境が正しく構築されているか確認してください。                    |
| WSAESOCKTNOSUPPORT | 10044  | 指定されたソケットの型は、このアドレスファミリでサポートされていません。TCP/IP動作環境が正しく構築されているか確認してください。       |
| WSAEAFNOSUPPORT    | 10047  | 指定されたアドレスファミリがサポートされていません。TCP/IP動作環境が正しく構築されているか確認してください。                 |

| WINSOCK.H内の表記    | エラーコード | 意味                                                                                                               |
|------------------|--------|------------------------------------------------------------------------------------------------------------------|
| WSAEADDRINUSE    | 10048  | サーバへの接続アドレスが使用中です。                                                                                               |
| WSAEADDRNOTAVAIL | 10049  | サーバに接続できませんでした。クライアントの定義ファイルに指定されている相手マシン名がhosts ファイルに正しく定義されているか確認してください。                                       |
| WSAENETDOWN      | 10050  | ネットワークサブシステムがダウンしています。ネットワークを起動してください。                                                                           |
| WSAENETUNREACH   | 10051  | クライアントが動作しているホストからサーバが動作指定しているホストに接続できませんでした。ネットワークの定義が正しくされているか確認してください。                                        |
| WSAENETRESET     | 10052  | ネットワークがダウンしています。ネットワークを再起動してください。                                                                                |
| WSAECONNABORTED  | 10053  | タイムアウトまたは他のエラーによって接続が強制的に拒絶、または異常終了しました。クライアントまたはサーバが正常に動作しているか確認してください。このエラーは、通信時のノイズまたはLAN カードの異常などの原因が考えられます。 |
| WSAECONNRESET    | 10054  | クライアントまたはサーバがリセットされました。                                                                                          |
| WSAENOBUFS       | 10055  | 通信バッファスペースが不足しています。通信バッファスペースを増やすか、通信量を減らすなどして対処してください。                                                          |
| WSAEISCONN       | 10056  | クライアントとサーバがすでに接続されています。                                                                                          |
| WSAENOTCONN      | 10057  | クライアントとサーバが接続されていません。                                                                                            |
| WSAESHUTDOWN     | 10058  | クライアントとサーバが切断されました。                                                                                              |
| WSAETIMEDOUT     | 10060  | サーバに接続されることなくタイムアウトとなりました。サーバが起動しているか、またはサーバの動作しているマシンとLAN 接続されているか確認してください。                                     |
| WSAECONNREFUSED  | 10061  | サーバへの接続が拒絶されました。サーバが起動されているか確認してください。                                                                            |
| WSAEDISCON       | 10101  | クライアントまたはサーバが切断されました。                                                                                            |
| WSANO_DATA       | 11004  | servicesファイルにサービス名'cobci'が定義されていません。servicesファイルにサービス名'cobci'を定義してください。                                          |

上記以外のエラーについては、“Visual C++のオンラインヘルプ”を参照してください。

## 第22章 リモートデータベースアクセス

データベース(SQL)機能は、埋込みSQLを使用してPCクライアントからサーバ上のデータベースをアクセスするための機能です。埋込みSQLとは、COBOLソースプログラム中に記述されたデータベース操作言語SQLです。

本機能を利用することにより、分散開発や幅広いアプリケーション開発が可能となります。

COBOLプログラムからSQL文を使ってデータベースをアクセスには、以下の2つの方法があります。

- ・ データベース製品の提供元が用意しているプリコンパイラを利用する。
- ・ ODBCインタフェースを利用する。

プリコンパイラが提供されているデータベースについては、各データベースの機能(SQL文の機能、処理性能)を活用することができるプリコンパイラの利用を推奨しています。

プリコンパイラが提供されていないデータベースについては、ODBCインタフェースを使用します。

### 22.1 プリコンパイラを使用したアクセス

プリコンパイラが使用できるデータベースについては、ソフトウェア説明書を参照してください。プリコンパイラの使用方法は、各データベースによって異なります。各データベースのプリコンパイラの使用書を確認してください。

### 22.2 ODBC経由によるアクセス

COBOLプログラムに埋込みSQLを記述し、ODBCドライバを経由してデータベースをアクセスする方法について説明します。

ODBC(Open DataBase Connectivity)は、Microsoft社が提唱している、データベースにアクセスするためのアプリケーションプログラムインタフェースです。

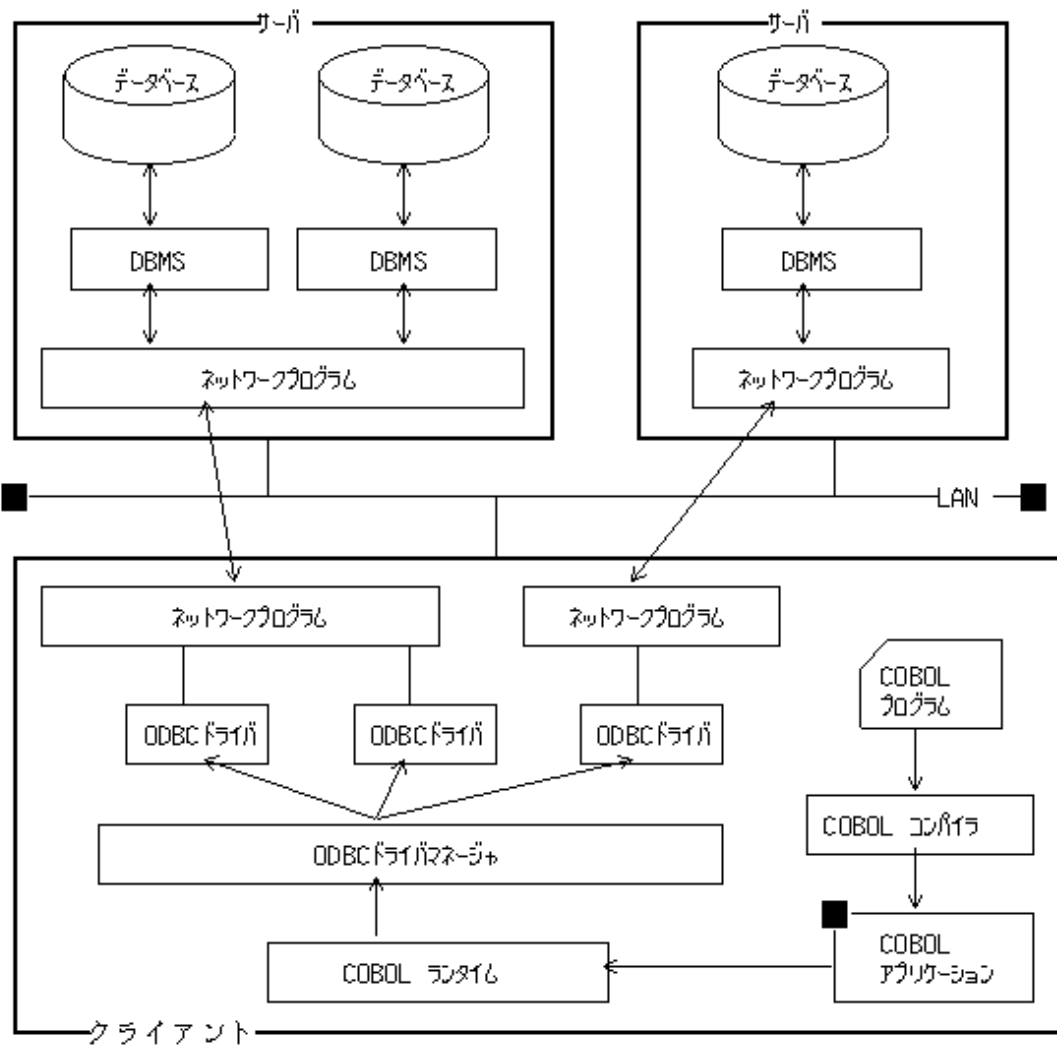
#### 22.2.1 概要

ここでは、COBOLアプリケーションからODBCドライバを経由してデータベースをアクセスする場合の概要について説明します。なお、ODBCドライバを使用するデータベース(SQL)機能を使ったデータの取出しを行う例題プログラムがサンプルプログラムとして提供されていますので、参考にしてください。

ODBCドライバを使用することにより、単一のデータベースだけでなく、複数のデータベースへのアクセスが可能になります。

ODBCドライバを使用したデータベースアクセスの概要図を、“[図22.1 ODBCドライバを使用したデータベースアクセスの概要図](#)”に示します。

図22.1 ODBCドライバを使用したデータベースアクセスの概要図



### 22.2.1.1 COBOLプログラムの構成

COBOLプログラムの全体構成を以下に示します。

COBOLプログラムの全体構成

```

IDENTIFICATION DIVISION.
:
ENVIRONMENT DIVISION.
:
DATA DIVISION.
:
WORKING-STORAGE SECTION.
:
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 SQLSTATE PIC X(5).
:
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.
:
EXEC SQL CONNECT ... END-EXEC.

```

←特にSQL 固有の記述はありません。

←宣言節内でSQLSTATEを宣言します。  
また、必要なら、ホスト変数を定義します。

----- サーバに接続します。

```

 :
EXEC SQL DECLARE CUR1 ... END-EXEC. ----- カーソルを宣言します。 ←カーソルを使用する場合
 :
EXEC SQL OPEN CUR1 END-EXEC. ----- カーソルを開きます。
 :
EXEC SQL FETCH CUR1 ... END-EXEC.
 :
EXEC SQL CLOSE CUR1 END-EXEC. ----- カーソルを閉じます。
 :
EXEC SQL ROLLBACK WORK END-EXEC.
 :
EXEC SQL DISCONNECT ... END-EXEC. ----- サーバを切断します。
 :
STOP RUN.

```

COBOLプログラム中にSQLを記述する場合は、“EXEC SQL”(SQL先頭子)と“END-EXEC”(SQL終了子)で囲んで記述します。手続き部に記述されたSQL文によって、実際のデータベースの処理が行われます。

### 22.2.1.2 埋込みSQL文による操作

埋込みSQL文を記述することにより、以下の操作ができます。

#### コネクションの接続

コネクションとは、データベースをアクセスするためにクライアントとサーバの間を結んだ接続関係のことです。この接続関係を結ぶことにより、クライアントからサーバのデータベースをアクセスするSQL文の実行が可能になります。

コネクションを接続するためには、CONNECT文を使います。[参照]“[22.2.2.1 コネクションを接続する](#)”

#### コネクションの変更

コネクションを変更するためには、SET CONNECTION文を使います。[参照]“[22.2.2.3 コネクションを変更する](#)”

#### データ操作

データ操作とは、応用プログラムからデータベースにデータを設定したり、データベースに格納されているデータを参照したりすることです。データ操作は、データベースの1つの行だけを対象に行うことも、複数の行を対象に行うこともできます。

データを操作するためには、SELECT文、INSERT文、UPDATE文およびDELETE文を使います。カーソルを定義し、FETCH文によってデータを取り出すこともできます。また、これらの文を動的に実行することも可能です。[参照]“[22.2.3 データ操作](#)”、“[22.2.4 高度なデータ操作](#)”



#### 注意

COBOLで扱うデータとODBCで扱うデータの対応は“[22.2.11 ODBCで扱うデータとの対応](#)”を参照してください。

#### ストアドプロシージャの呼出し

ストアドプロシージャを呼び出すためには、CALL文を使います。

呼び出すストアドプロシージャは、データベース上に登録されていなければなりません。[参照]“[22.2.5 ストアドプロシージャの呼出し](#)”



#### 注意

COBOLで扱うデータとODBCで扱うデータの対応は“[22.2.11 ODBCで扱うデータとの対応](#)”を参照してください。

#### トランザクション処理

トランザクションは、データベースに対するデータ操作の一貫性を保証する単位です。トランザクションは、最初のSQL文を実行したときに開始され、COMMIT文またはROLLBACK文を実行したときに終了します。

## コネクションの切断

プログラムとサーバとのコネクションの切断は、DISCONNECT文で行います。

コネクションを切断する前には、トランザクションを終了させておかなくてはなりません。[参照]“[22.2.2.2 コネクションを切断する](#)”以降の節では、これらについて例題をあげて説明します。

## 22.2.2 コネクション操作

ここでは、コネクションの接続方法、切断方法および変更方法について説明します。

### 22.2.2.1 コネクションを接続する

ここでは、CONNECT文でサーバとコネクションを接続する方法について説明します。

以下の手順で、サーバとコネクションを接続します。

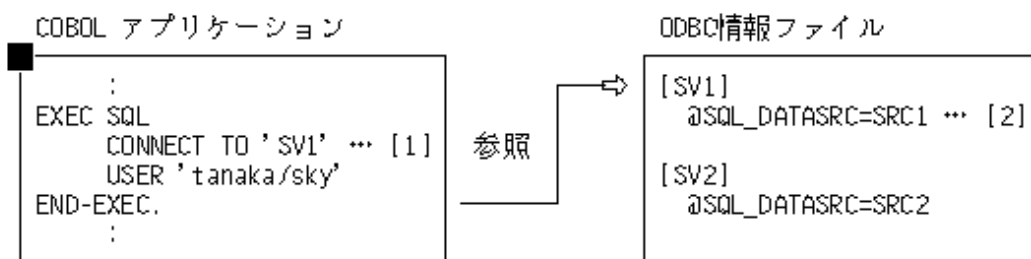
1. サーバの情報を登録します。
2. 以下のどちらかの方法で、サーバとコネクションを接続します。
  - サーバ名を指定して接続する
  - DEFAULTを指定して接続する

#### サーバ名を指定して接続する

プログラムを実行する前に、サーバの情報をODBC情報ファイルに設定しておきます。定義するサーバ情報の詳細および定義方法については、“[22.2.8 プログラムの実行](#)”を参照してください。

サーバ名を指定してCONNECT文を実行すると、指定したサーバ名の情報をODBC情報ファイル内で検索します。同名のセクションが見つかったら、そこからサーバの情報を参照してサーバとコネクションを接続します。

図22.2 サーバ名を指定する接続方法



[1] CONNECT文にサーバ名を指定して実行します。

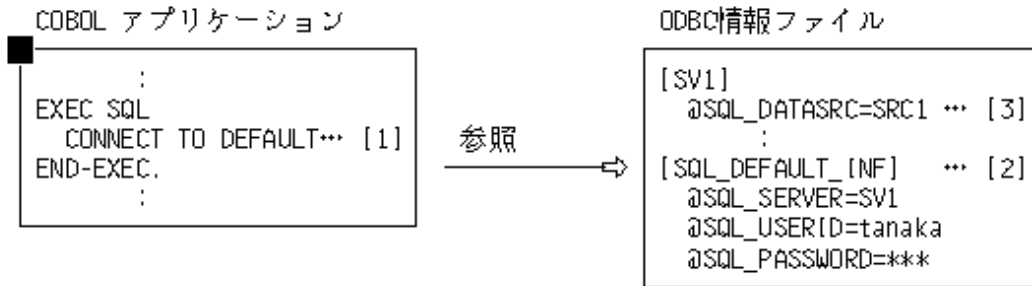
[2] CONNECT文の実行時に、ODBC情報ファイルの情報を参照してサーバとコネクションを接続します。

#### DEFAULTを指定して接続する

プログラムを実行する前に、デフォルトコネクションの情報をODBC情報ファイルに設定しておきます。定義する情報の詳細および定義方法については、“[22.2.8 プログラムの実行](#)”を参照してください。

DEFAULTを指定してCONNECT文を実行すると、ODBC情報ファイルのデフォルトコネクション情報を参照します。デフォルトコネクション情報にはサーバ名の情報が設定されているので、さらにそのサーバ名でODBC情報ファイル内を検索します。サーバ名の情報が見つかったら、その情報を参照してサーバとコネクションを接続します。

### 図22.3 DEFAULTを指定する接続方法



- [1] CONNECT文にDEFAULTを指定して実行します。
- [2] CONNECT文の実行時に、デフォルトコネクション情報を参照します。
- [3] さらに、デフォルトコネクション情報のサーバ名の情報から、SV1の情報を参照してサーバとコネクションを接続します。

#### 22.2.2.2 コネクションを切断する

コネクションを切断するためには、DISCONNECT文を使用します。

DISCONNECT文には、切断の対象となるコネクションを指定します。切断対象となるコネクションとして指定できるのは、コネクション名、DEFAULT、CURRENT、ALLです。それぞれの使い方は、「[複数コネクションを接続/変更/切断する例](#)」を参照してください。

### 22.2.2.3 コネクションを変更する

コネクションを変更するためには、SET CONNECTION文を使用します。

複数のCONNECT文を実行することにより、複数のコネクションが接続されます。アプリケーションが複数コネクションを持つ場合、SQL文の実行対象となるコネクションを決めておく必要があります。複数のCONNECT文を実行した場合、最後のCONNECT文で接続したコネクションが現在のコネクションとなり、SQL文の実行対象となります。現在のコネクションを変更するためには、SET CONNECTION文を実行します。SET CONNECTION文を実行することにより、SQL文の実行対象となるコネクションを変更することができます。

“複数コネクションを接続/変更/切断する例”は、複数コネクションの接続、変更、切断を行うCOBOLプログラム例です。

この例では、デフォルトコネクションの他に、“SV1”、“SV2”、“SV3”および“SV4”という名前のSQLサーバがODBC情報ファイルに登録されています。

## 複数コネクションを接続/変更/切断する例

```
EXEC SQL CONNECT TO DEFAULT END-EXEC. ... [1]
EXEC SQL
CONNECT TO 'SV1' AS 'CNN1' USER 'tanaka/sky' ... [2]
END-EXEC.
EXEC SQL
CONNECT TO 'SV2' AS 'CNN2' USER 'tanaka/sky' ... [3]
END-EXEC.
EXEC SQL
CONNECT TO 'SV3' AS 'CNN3' USER 'tanaka/sky' ... [4]
END-EXEC.
EXEC SQL
CONNECT TO 'SV4' AS 'CNN4' USER 'tanaka/sky' ... [5]
END-EXEC.
EXEC SQL DISCONNECT 'CNN4' END-EXEC. ... [6]
EXEC SQL SET CONNECTION 'CNN1' END-EXEC. ... [7]
EXEC SQL ROLLBACK WORK END-EXEC. ... [8]
EXEC SQL DISCONNECT CURRENT END-EXEC. ... [9]
EXEC SQL SET CONNECTION DEFAULT END-EXEC. ... [10]
```



```

:
EXEC SQL COMMIT WORK END-EXEC. ... [11]
EXEC SQL DISCONNECT DEFAULT END-EXEC. ... [12]
EXEC SQL SET CONNECTION 'CNN2' END-EXEC.
:
EXEC SQL ROLLBACK WORK END-EXEC.
EXEC SQL DISCONNECT ALL END-EXEC. ... [13]

```

- [1] デフォルトコネクションに接続します。
- [2] SQLサーバ“SV1”に接続します。この接続の名前を“CNN1”とします。
- [3] SQLサーバ“SV2”に接続します。この接続の名前を“CNN2”とします。
- [4] SQLサーバ“SV3”に接続します。この接続の名前を“CNN3”とします。
- [5] SQLサーバ“SV4”に接続します。この接続の名前を“CNN4”とします。最後に接続した“CNN4”が現在のコネクションになります。
- [6] コネクション名“CNN4”を指定して、コネクションを切断します。
- [7] コネクション名“CNN1”を指定し、コネクションを変更します。“CNN1”が現在のコネクションになります。
- [8] コネクション“CNN1”中のデータの変更をすべて取り消します。
- [9] 現在のコネクションを切断します。本例では現在のコネクションは“CNN1”です。そのため、“CNN1”が切断されます。
- [10] DEFAULTを指定してコネクションを変更します。デフォルトコネクションが現在のコネクションになります。
- [11] デフォルトコネクション中のデータの変更をすべて保存します。
- [12] デフォルトコネクションを切断します。
- [13] ALLを指定してDISCONNECT文を実行することにより、接続されているすべてのコネクションを切断します。



## 注意

CONNECT文により同時に接続できるコネクション数は、最大128 個です。  
ただし、これは、ODBCドライバおよび関係する環境により、制限を受けることがあります。

## 22.2.3 データ操作

ここでは、以下のデータ操作について説明します。

- データを参照する
- データを更新する
- データを削除する
- データを挿入する
- 動的SQLを使用する
- 可変長文字列を使用する
- 複数コネクションでカーソルを操作する

[参照]“[22.2.11 ODBCで扱うデータとの対応](#)”

### 22.2.3.1 サンプルデータベース

本節のプログラム例で使用するサンプルデータベースの3つの表について説明します。

- STOCK表(在庫表)  
製品番号(GNO)、製品名(GOODS)、在庫数量(QOH)、倉庫番号(WHNO)を格納しています。

- ORDERS表(注文表)

取引番号(ORDERID)、取引先番号(COMPANYNO)、取引製品番号(GOODSNO)、仕入れ価格(PRICE)、発注数量(OOH)を格納しています。

- COMPANY表(会社表)

会社番号(CNO)、会社名(NAME)、電話番号(PHONE)、住所(ADDRESS)を格納しています。

#### STOCK表(在庫表)

| GNO | GOODS           | QOH  | WHNO |
|-----|-----------------|------|------|
| 110 | TELEVISION      | 85   | 2    |
| 111 | TELEVISION      | 90   | 2    |
| 123 | REFRIGERATOR    | 60   | 1    |
| 124 | REFRIGERATOR    | 75   | 1    |
| 137 | RADIO           | 150  | 2    |
| 138 | RADIO           | 200  | 2    |
| 140 | CASSETTE DECK   | 120  | 2    |
| 141 | CASSETTE DECK   | 80   | 2    |
| 200 | AIR CONDITIONER | 4    | 1    |
| 201 | AIR CONDITIONER | 15   | 1    |
| 212 | TELEVISION      | 0    | 2    |
| 215 | VIDEO           | 5    | 2    |
| 226 | REFRIGERATOR    | 8    | 1    |
| 227 | REFRIGERATOR    | 15   | 1    |
| 240 | CASSETTE DECK   | 25   | 2    |
| 243 | CASSETTE DECK   | 14   | 2    |
| 351 | CASSETTE TAPE   | 2500 | 2    |
| 380 | SHAVER          | 870  | 3    |
| 390 | DRIER           | 540  | 3    |

#### ORDERS表(注文表)

| ORDERID | COMPANYNO | GOODSNO | PRICE  | OOH |
|---------|-----------|---------|--------|-----|
| 1       | 61        | 123     | 48000  | 60  |
| 2       | 61        | 124     | 64000  | 40  |
| 3       | 61        | 138     | 6400   | 180 |
| 4       | 61        | 140     | 8000   | 80  |
| 5       | 61        | 215     | 240000 | 10  |
| 6       | 61        | 240     | 80000  | 20  |
| 7       | 62        | 110     | 37500  | 120 |
| 8       | 62        | 226     | 112500 | 20  |
| 9       | 62        | 351     | 375    | 800 |
| 10      | 63        | 111     | 57400  | 80  |
| 11      | 63        | 200     | 123000 | 60  |
| 12      | 63        | 201     | 164000 | 50  |

| ORDERID | COMPANYNO | GOODSNO | PRICE  | OOH  |
|---------|-----------|---------|--------|------|
| 13      | 63        | 212     | 205000 | 30   |
| 14      | 63        | 215     | 246000 | 10   |
| 15      | 71        | 140     | 7800   | 50   |
| 16      | 71        | 351     | 390    | 600  |
| 17      | 72        | 137     | 3500   | 120  |
| 18      | 72        | 140     | 7000   | 70   |
| 19      | 72        | 215     | 210000 | 10   |
| 20      | 72        | 226     | 105000 | 20   |
| 21      | 72        | 243     | 84000  | 10   |
| 22      | 72        | 351     | 350    | 1000 |
| 23      | 73        | 141     | 16000  | 60   |
| 24      | 73        | 380     | 2400   | 250  |
| 25      | 73        | 390     | 2400   | 150  |
| 26      | 74        | 110     | 39000  | 120  |
| 27      | 74        | 111     | 54000  | 120  |
| 28      | 74        | 226     | 117000 | 20   |
| 29      | 74        | 227     | 140400 | 10   |
| 30      | 74        | 351     | 390    | 700  |

COMPANY表(会社表)

| CNO | NAME       | PHONE    | ADDRESS              |
|-----|------------|----------|----------------------|
| 61  | ADAM LTD.  | 731-1111 | SANTA CLARA C.A USA  |
| 62  | IDEA INC.  | 433-2222 | LONDON W.C.2 ENGLAND |
| 63  | MOON CO.   | 143-3333 | FIFTH AVENUE N.Y USA |
| 71  | RIVER CO.  | 344-1212 | SAKAI OOSAKA JAPAN   |
| 72  | DRAGON CO. | 373-7777 | YAO OOSAKA JAPAN     |
| 73  | BIG INC.   | 391-0808 | HARAJUKU TOKYO JAPAN |
| 74  | FIRST CO.  | 255-9955 | SYDNEY AUSTRALIA     |

## 22.2.3.2 データの参照

データベースのデータを参照する方法について説明します。

### 22.2.3.2.1 表の全行を参照する

データベースのすべての行を参照する方法について説明します。

“すべての行を参照する例”は、サンプルデータベースのSTOCK表のすべての行を参照するCOBOLプログラム例です。

すべての行を参照する例

```

 :
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
 02 製品番号 PIC S9(4) COMP-5.
 02 製品名 PIC X(20).
 02 在庫数量 PIC S9(9) COMP-5.
 02 倉庫番号 PIC S9(4) COMP-5.

```

```

01 SQLSTATE PIC X(5). ↓
EXEC SQL END DECLARE SECTION END-EXEC. --
PROCEDURE DIVISION.
EXEC SQL WHENEVER NOT FOUND GO TO :P-END END-EXEC. ... [2]
EXEC SQL
 DECLARE CUR1 CURSOR FOR SELECT * FROM STOCK ... [3]
END-EXEC.
P-START.
EXEC SQL CONNECT TO DEFAULT END-EXEC. ... [4]
EXEC SQL OPEN CUR1 END-EXEC. ... [5]
P-LOOP.
EXEC SQL
 FETCH CUR1 INTO :在庫表 ... [6]
END-EXEC.
 :
GO TO P-LOOP.
P-END.
EXEC SQL CLOSE CUR1 END-EXEC. ... [7]
EXEC SQL ROLLBACK WORK END-EXEC. ... [8]
EXEC SQL DISCONNECT DEFAULT END-EXEC. ... [9]
STOP RUN.

```

[1] 作業場所節に埋込みSQL宣言節を記述し、STOCK表の全列に対応するデータの入力用領域をホスト変数として定義します。ホスト変数宣言の規則については、“COBOL文法書”を参照してください。

[2] 埋込み例外宣言を記述すると、SQL文が例外事象を生じた場合にとる動作を指定することができます。ここでは、条件として“NOT FOUND”を指定しているため、SQLSTATEの値が“データなし”を示すときに有効になります。すなわち、[6]のFETCH文によって次々と行を取り出し、それ以上取り出す行がなくなったときに、手続き名“P-END”の位置に分岐することを指定しています。

[補足] COBOLのIF文でSQLSTATEを判定し、例外事象を生じた場合の動作を指定することもできます。

[3] カーソル宣言によって、STOCK表を参照するカーソル名を定義します。この例では、STOCK表に対してとくに列を選択したり、探索条件を指定したりしていないので、問合せ式から導出される表は、元のSTOCK表と同じものになります。

[4] CONNECT文を実行し、サーバとのコネクションを接続します。

[5] OPEN文を実行し、指定したカーソルを使用可能な状態にします。

[6] FETCH文によって表からデータを1行ずつ取り出し、各列の値を対応するホスト変数の領域に設定します。

[7] CLOSE文を実行し、指定したカーソルを無効な状態にします。

[8] ROLLBACK文を実行し、トランザクションを終了します。

[9] DISCONNECT文を実行し、サーバとのコネクションを切断します。

問合せ式の選択リストに列名の並びを書かずにアスタリスク(\*)を指定した場合、選択される列の順序は、表を定義したときに指定した順序と同じになります。

“すべての行を参照する例”では、複数列指定ホスト変数を使用しています。複数列指定ホスト変数は、データベースの各列に対応するホスト変数を1つの集団項目の従属項目として定義します。これを埋込みSQL文で参照する場合には、集団項目の名前で参照します。これは一種の省略記法であり、次のようにホスト変数を定義または参照する場合と同じです。

```

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
:
01 製品番号 PIC S9(4) COMP-5.
01 製品名 PIC X(20).
01 在庫数量 PIC S9(9) COMP-5.
01 倉庫番号 PIC S9(4) COMP-5.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
:
EXEC SQL
 FETCH CUR1
 INTO :製品番号, :製品名, :在庫数量, :倉庫番号

```

```
END-EXEC.
:
```

複数列指定ホスト変数のように、ホスト変数を集団項目として定義する書き方はV60からサポートされた新しい書き方です。[参照]“[A.2.50 SQLGRP \(SQLのホスト変数定義の拡張\)](#)”

### 22.2.3.2.2 条件を指定して参照する

表のすべての行を参照するのではなく、条件を指定してその条件を満たす行だけを参照する方法について説明します。

“条件を指定して参照する例”は、サンプルデータベースのSTOCK表から、在庫数量が50以下の製品の製品番号、製品名および在庫数量を参照するCOBOLプログラム例です。

#### 条件を指定して参照する例

```
 :
 EXEC SQL BEGIN DECLARE SECTION END-EXEC. --
01 在庫表. ↑
 02 製品番号 PIC S9(4) COMP-5.
 02 製品名 PIC X(20). [1]
 02 在庫数量 PIC S9(9) COMP-5.
01 SQLSTATE PIC X(5). ↓
 EXEC SQL END DECLARE SECTION END-EXEC. --
PROCEDURE DIVISION.
 EXEC SQL WHENEVER NOT FOUND GO TO :P-END END-EXEC.
 EXEC SQL
 DECLARE CUR2 CURSOR FOR ... [2]
 SELECT GNO, GOODS, QOH FROM STOCK
 WHERE QOH <= 50
 END-EXEC.
P-START.
 EXEC SQL CONNECT TO DEFAULT END-EXEC. ... [3]
 EXEC SQL OPEN CUR2 END-EXEC. ... [4]
P-LOOP.
 EXEC SQL
 FETCH CUR2 INTO :在庫表 ... [5]
 END-EXEC.
 :
 GO TO P-LOOP.
P-END.
 EXEC SQL CLOSE CUR2 END-EXEC. ... [6]
 EXEC SQL ROLLBACK WORK END-EXEC. ... [7]
 EXEC SQL DISCONNECT DEFAULT END-EXEC. ... [8]
 STOP RUN.
```

[1] 埋込みSQL文中に指定するホスト変数は、すべて埋込みSQL宣言節で定義します。

[2] カーソル宣言によって実際に表が探索されることはありません。表の探索は、[4]のOPEN文の実行時に行われます。

[3] CONNECT文を実行し、サーバとのコネクションを接続します。

[4] OPEN文の実行により、[2]のカーソル宣言で指定した条件を満たす行からなる仮想的な表が作られます。ここで作られる仮想表は、STOCK表からQOH列の値が50以下である行を抽出し、さらにそこからGNO列、GOODS列およびQOH列を抜き出した形態をしています。一般に、このような仮想表の行の順番は不定です。

[5] FETCH文は、[4]で導出された仮想表の先頭から、データを1行ずつ取り出し、各列の値を対応するホスト変数の領域に設定します。特定の列について、値の昇順または降順で取り出したい場合は、[2]の問合せ式の後にORDER BY句を指定します。

[6] CLOSE文は、[4]で導出された仮想表を無効にします。再びOPEN文を実行しないかぎり、以降のSQL文でこの仮想表を参照することはできません。

[7] ROLLBACK文を実行し、トランザクションを終了します。

[8] DISCONNECT文を実行し、サーバとのコネクションを切断します。

### 22.2.3.2.3 1つの行を参照する

“すべての行を参照する例”および“条件を指定して参照する例”では、複数の行を参照する場合について説明しました。

これに対し、探索結果が1行しかないことがわかっている場合は、SELECT文でデータを参照することができます。カーソルを使用しないため、カーソル宣言、OPEN文、FETCH文およびCLOSE文による一連の処理は不要です。たとえば、STOCK表から製品番号200の製品名と在庫数量を参照する場合は、以下のSELECT文を実行します。

```
EXEC SQL
 SELECT GOODS, QOH INTO :製品名, :在庫数量 FROM STOCK
 WHERE GNO = 200
END-EXEC.
```

SELECT文では、選択リストの値式に集合関数指定を記述することにより、指定された値式の最大値、最小値、平均値、総和(合計)、および表の基数(行数)を求めることができます。たとえば、ORDERS表について仕入れ価格の最大値、最小値、および平均値を求める例では、次のSQL文を実行します。

```
EXEC SQL
 SELECT MAX(PRICE), MIN(PRICE), AVG(PRICE)
 INTO :最大値, :最小値, :平均値 FROM ORDERS
END-EXEC.
```

結果は、ホスト変数“最大値”、“最小値”および“平均値”に設定されます。

### 22.2.3.2.4 表を関連付けてデータを参照する

ここでは、表を関連付けてデータを参照する方法について説明します。

#### 複数表を関連付けて参照する

複数の表を、それぞれの表に含まれる列の値によって関連付けて探索し、データを参照することができます。たとえば、サンプルデータベースを使用して、特定の製品名(TELEVISION)を取り扱っている会社名と発注数量を参照するとします。この場合、サンプルデータベースの3つの表を以下のように関連付け、求めるデータを参照します。

- STOCK表とORDERS表を、製品番号(GNO列)と取引製品番号(GOODSNO列)の値に基づいて関連付ける。
- ORDERS表とCOMPANY表を、取引番号(COMPANYNO列)と会社番号(CNO列)の値に基づいて関連付ける。

問合せ式は、以下のようになります。

```
SELECT NAME, OOH
 FROM STOCK, ORDERS, COMPANY
 WHERE GOODS = 'TELEVISION' AND
 GNO = GOODSNO AND
 COMPANYNO = CNO
```

3つの表を関連付けた表から、探索条件を満たす行が導出されます。

| GNO | ... | WHNO | COMPANYNO | ... | OOH | CNO | ... | ADDRESS              |
|-----|-----|------|-----------|-----|-----|-----|-----|----------------------|
| 110 | ... | 2    | 61        | ... | 60  | 61  | ... | SANTA CLARA C.A USA  |
| 110 | ... | 2    | 61        | ... | 60  | 62  | ... | LONDON W.C.2 ENGLAND |
| 110 | ... | 2    | 61        | ... | 60  | 63  | ... | FIFTH AVENUE N.Y USA |
| ~   | ~   | ~    | ~         | ~   | ~   | ~   | ~   | ~                    |
| 110 | ... | 2    | 61        | ... | 40  | 61  | ... | SANTA CLARA C.A USA  |
| 110 | ... | 2    | 61        | ... | 40  | 62  | ... | LONDON W.C.2 ENGLAND |
| 110 | ... | 2    | 61        | ... | 40  | 63  | ... | FIFTH AVENUE N.Y USA |
| ~   | ~   | ~    | ~         | ~   | ~   | ~   | ~   | ~                    |
| 111 | ... | 2    | 61        | ... | 60  | 61  | ... | SANTA CLARA C.A USA  |
| 111 | ... | 2    | 61        | ... | 60  | 62  | ... | LONDON W.C.2 ENGLAND |

| GNO | ... | WHNO | COMPANYNO | ... | OOH | CNO | ... | ADDRESS              |
|-----|-----|------|-----------|-----|-----|-----|-----|----------------------|
| 111 | ... | 2    | 61        | ... | 60  | 63  | ... | FIFTH AVENUE N.Y USA |
| ～   | ～   | ～    | ～         | ～   | ～   | ～   | ～   | ～                    |
| ～   | ～   | ～    | ～         | ～   | ～   | ～   | ～   | ～                    |
| 390 | ... | 3    | 74        | ... | 700 | 72  | ... | YAO OOSAKA JAPAN     |
| 390 | ... | 3    | 74        | ... | 700 | 73  | ... | HARAJUKU TOKYO JAPAN |
| 390 | ... | 3    | 74        | ... | 700 | 74  | ... | SYDNEY AUSTRALIA     |

↓ 問合せ式の結果

| NAME      | OOH |
|-----------|-----|
| IDEA INC. | 120 |
| MOON CO.  | 80  |
| FIRST CO. | 120 |
| FIRST CO. | 120 |

### 同一表を関連付けて参照する

複数の表の関連付けと同様の考え方で、同一の表に対し、行と行で関連付けて探索することもできます。たとえばSTOCK表で、製品名TELEVISIONと同じ倉庫に在庫のある製品名を参照するとします。このような場合、STOCK表に対して2つの異なる別名(相関名)を与え、それらをあたかも別の表であるかのように扱います。

“同一表を関連付けて参照する例”は、同一の表を関連付け、製品名TELEVISIONと同じ倉庫に在庫のある製品名を参照するCOBOLプログラム例です。

#### 同一表を関連付けて参照する例

```

 :
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 製品名 PIC X(20).
01 SQLSTATE PIC X(5).
 EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
 EXEC SQL WHENEVER NOT FOUND GO TO :P-END END-EXEC.
 EXEC SQL
 DECLARE CUR4 CURSOR FOR
 SELECT DISTINCT X2.GOODS
 FROM STOCK X1,STOCK X2
 WHERE X1.GOODS = 'TELEVISION' AND
 X1.WHNO = X2.WHNO
 END-EXEC.
 P-START.
 EXEC SQL CONNECT TO DEFAULT END-EXEC.
 EXEC SQL OPEN CUR4 END-EXEC.
 P-LOOP.
 EXEC SQL
 FETCH CUR4 INTO :製品名
 END-EXEC.
 :
 GO TO P-LOOP.
 P-END.
 EXEC SQL CLOSE CUR4 END-EXEC.
 EXEC SQL ROLLBACK WORK END-EXEC.
 EXEC SQL DISCONNECT DEFAULT END-EXEC.
 STOP RUN.

```

[1] カーソル宣言文では、FROM句でSTOCK表に対して相関名(例では“X1”および“X2”)を指定し、あたかもそれが別々の表であるかのように扱います。

WHERE句の探索条件では、列名を相関名で修飾し、製品名TELEVISIONと同じ倉庫番号の倉庫に存在する製品に関する行を探索するように指定します。

### 22.2.3.3 データの更新

ここでは、表のデータを更新する方法について説明します。

データを更新するには、UPDATE文を使用します。たとえば、STOCK表の在庫数量(QOH列)の値を一律に10%減らす場合は、CONNECT文によってサーバと接続した後、次の文を実行します。

```
EXEC SQL
 UPDATE STOCK SET QOH = QOH * 0.9
END-EXEC.
```

このUPDATE文では、STOCK表のすべての行のQOH列の値を、それに0.9を乗じた値で置き換えます。特定の条件を満たす行に対してだけ更新したいときには、その条件をUPDATE文のWHERE句の探索条件に指定します。たとえば、製品名TELEVISIONの在庫数量だけを10%減らすのであれば、前述のUPDATE文を次のように変更します。

```
EXEC SQL
 UPDATE STOCK SET QOH = QOH * 0.9
 WHERE GOODS = 'TELEVISION'
END-EXEC.
```



#### 注意

複数の表を関連付けて得られる表に対して、データの更新を行うことはできません。

### 22.2.3.4 データの削除

ここでは、表のデータを削除する方法について説明します。

データを削除するには、DELETE文を使用します。たとえば、STOCK表から製品名が“CASSETTE DECK”である行をすべて削除する場合は、CONNECT文によってサーバと接続した後、次の文を実行します。

```
EXEC SQL
 DELETE FROM STOCK WHERE GOODS = 'CASSETTE DECK'
END-EXEC.
```



#### 注意

複数の表を関連付けて得られる表に対して、データの削除を行うことはできません。

### 22.2.3.5 データの挿入

ここでは、表にデータを挿入する方法について説明します。

データを挿入するには、INSERT文を使用します。INSERT文により、以下のどちらかの方法でデータを挿入できます。

- ・ 1行だけのデータを挿入する
- ・ ある探索条件に従って別の表から抽出した行の集合を挿入する

#### 単一行の挿入

STOCK表に、製品番号301の行を追加する場合について説明します。この行の製品名は“WASHER”、在庫数量は50、倉庫番号は1とします。このデータを挿入するには、CONNECT文によってサーバと接続した後、次の文を実行します。



```
EXEC SQL
 INSERT INTO STOCK (GNO, GOODS, QOH, WHNO)
 VALUES (301, 'WASHER', 50, 1)
END-EXEC.
```

別表からの複数行の挿入

STOCK表と同じデータベース上に別の在庫表(表名は“SUBSTOCK”で列名およびその属性はSTOCK表と同じ)があり、その表から製品名が“MICROWAVE OVEN”であるデータをSTOCK表に挿入する場合について説明します。挿入するすべての行の倉庫番号を2にします。この処理を行うには、CONNECT文によりサーバに接続した後、次の文を実行します。

```
EXEC SQL
 INSERT INTO STOCK (GNO, GOODS, QOH, WHNO)
 SELECT GNO, GOODS, QOH, 2 FROM SUBSTOCK
 WHERE GOODS = 'MICROWAVE OVEN'
END-EXEC.
```



## 注意

カーソルで開かれた状態の表に対して、非カーソルのデータ操作を実行すると、エラーが発生することがあります。表示されるエラーメッセージは使用するデータベースや動作設定に依存します。エラーメッセージが表示された場合は使用するデータベースの仕様を確認してください。

## 22.2.3.6 動的SQL

動的SQLを使用することにより、プログラムの実行時にSQL文を生成し、実行することができます。

### 探索条件を動的に決定する

探索条件で用いる値は、ホスト変数を使用することで実行時に設定することができました。ここでは、探索条件そのものを実行時に決定する方法について説明します。

“探索の条件を実行時に指定するプログラム例”は、動的カーソル宣言を用いて実行時に探索条件を決定するCOBOLプログラム例です。本例でのカーソル定義のための問合せ式は以下のとおりです。

```
SELECT GNO, GOODS, QOH FROM STOCK
 WHERE GOODS = 'REFRIGERATOR' AND QOH < 10
```

問合せ式は、実行時にACCEPT文により読み込まれます。

### 探索の条件を実行時に指定するプログラム例

```

 :
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品番号 PIC S9(4) COMP-5.
02 製品名 PIC X(20).
02 在庫数量 PIC S9(9) COMP-5.
01 STMVAR PIC X(254). ... [1]
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
EXEC SQL WHENEVER NOT FOUND GO TO :P-END END-EXEC.
EXEC SQL
 DECLARE CUR8 CURSOR FOR STMIDT ... [2]
END-EXEC.
ACCEPT STMVAR FROM CONSOLE. ... [3]
P-START.
EXEC SQL CONNECT TO DEFAULT END-EXEC.
EXEC SQL PREPARE STMIDT FROM :STMVAR END-EXEC. ... [4]
EXEC SQL OPEN CUR8 END-EXEC. ... [5]
P-LOOP.
```

```

EXEC SQL
 FETCH CUR8
 INTO :在庫表
 END-EXEC.
 :
GO TO P-LOOP.
P-END.
EXEC SQL CLOSE CUR8 END-EXEC.
EXEC SQL ROLLBACK WORK END-EXEC.
EXEC SQL DISCONNECT DEFAULT END-EXEC.
STOP RUN.

```

- [1] このSQL文変数STMVARは、[4]のPREPARE文で参照されます。
- [2] SQL文識別子STMIDTは、[4]のPREPARE文でSQL文変数STMVARと対応付けられます。
- [3] ACCEPT文により、問合せ式を読み込み、SQL文変数STMVARに設定します。
- [4] PREPARE文により、この時点でSQL文変数に設定されていた文(この例では動的SELECT文)とSQL文識別子STMIDTが対応付けられます。
- [5] 動的OPEN文により、本例ではSTOCK表から在庫が10未満の冷蔵庫に関する行が抽出され、それらの中から、“GNO(製品番号)”、“GOODS(製品名)”および“QOH(在庫数量)”の3つの列からなる表が導出されます。
- [6] 動的FETCH文により、表からデータを1行ずつ取り出し、各列の値を対応するホスト変数に設定します。
- [7] 動的CLOSE文により、指定したカーソルおよびそのカーソルに対応する表を無効状態にします。

## SQL文を動的に決定する

動的SQLでは、カーソル宣言での問合せ式以降の探索条件だけでなく、そこで実行するSQL文を動的に変更することができます。ここでは、EXECUTE文を用いた方法について説明します。

“SQL文を動的に決定するプログラム例(その1)”は、ACCEPT文によって入力されたSQL文を動的に実行するCOBOLプログラム例です。実行時には、以下のUPDATE文を入力します。

```
UPDATE STOCK SET QOH = 0 WHERE GOODS = 'TELEVISION'
```

### SQL文を動的に決定するプログラム例(その1)

```

:
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 STMVAR PIC X(254).
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
 ACCEPT STMVAR FROM CONSOLE.
 EXEC SQL CONNECT TO DEFAULT END-EXEC.
 EXEC SQL PREPARE STMIDT FROM :STMVAR END-EXEC.
 EXEC SQL EXECUTE STMIDT END-EXEC.
 :
 EXEC SQL ROLLBACK WORK END-EXEC.
 EXEC SQL DISCONNECT DEFAULT END-EXEC.
 STOP RUN.

```

- [1] ACCEPT文によってUPDATE文を読み込み、SQL文変数STMVARに設定します。
- [2] PREPARE文によって、この時点でSQL文変数に設定されていた文(この例ではUPDATE文)とSQL文識別子STMIDTが対応付けられます。
- [3] EXECUTE文は、指定されたSQL文識別子に対応付けられた被準備文を実行します。

SQL文を動的に決定する場合、特にパラメタを指定する必要がないときには、EXECUTE IMMEDIATE文を用いて同様の処理を行うことができます。

“SQL文を動的に決定するプログラム例(その2)”は、“SQL文を動的に決定するプログラム例(その1)”と同様の処理をEXECUTE IMMEDIATE文によって行うCOBOLプログラム例です。実行時に入力されるのは、“SQL文を動的に決定するプログラム例(その1)”と同じUPDATE文です。

#### SQL文を動的に決定するプログラム例(その2)

```
 :
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 STMVAR PIC X(254).
01 SQLSTATE PIC X(5).
 EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
 ACCEPT STMVAR FROM CONSOLE. ... [1]
 EXEC SQL CONNECT TO DEFAULT END-EXEC.
 EXEC SQL EXECUTE IMMEDIATE :STMVAR END-EXEC. ... [2]
 :
 EXEC SQL ROLLBACK WORK END-EXEC.
 EXEC SQL DISCONNECT DEFAULT END-EXEC.
 STOP RUN.
```

[1] ACCEPT文によってUPDATE文を読み込み、SQL文変数STMVARに設定します。

[2] EXECUTE IMMEDIATE文は、SQL文変数に設定されているSQL文を直接実行します。

#### 動的パラメタ指定を記述する

“動的パラメタを使用するプログラム例”は、動的パラメタ指定を記述し、STOCK表のデータを参照するCOBOLプログラム例です。実行時には、以下のSELECT文を入力します。

```
SELECT GNO,GOODS FROM STOCK WHERE WHNO = ?
```

#### 動的パラメタを使用するプログラム例

```
 :
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 製品番号 PIC S9(4) COMP-5.
01 製品名 PIC X(20).
01 在庫数量 PIC S9(9) COMP-5.
01 倉庫番号 PIC S9(4) COMP-5.
01 STMVAR PIC X(254).
01 SQLSTATE PIC X(5).
01 SQLMSG PIC X(254).
 EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
 EXEC SQL WHENEVER NOT FOUND GO TO :P-END END-EXEC.
 EXEC SQL
 DECLARE CUR11 CURSOR FOR STMIDT ... [1]
 END-EXEC.
 ACCEPT STMVAR FROM CONSOLE. ... [2]
P-START.
 EXEC SQL CONNECT TO DEFAULT END-EXEC.
 EXEC SQL PREPARE STMIDT FROM :STMVAR END-EXEC. ... [3]
 ACCEPT 倉庫番号 FROM CONSOLE. ... [4]
 EXEC SQL OPEN CUR11 USING :倉庫番号 END-EXEC. ... [5]
P-LOOP.
 EXEC SQL
 FETCH CUR11 INTO :製品番号, :製品名 ... [6]
 END-EXEC.
 :
 GO TO P-LOOP.
P-END.
 EXEC SQL CLOSE CUR11 END-EXEC. ... [7]
 EXEC SQL ROLLBACK WORK END-EXEC.
```

```
EXEC SQL DISCONNECT DEFAULT END-EXEC.
STOP RUN.
```

- [1] 動的カーソル宣言により、CUR11を定義します。
- [2] ACCEPT文により、動的SELECT文を読み込みます。
- [3] PREPARE文によって、この時点でSQL文変数STMVARに設定されていた文と、SQL文識別子STMIDTが対応付けられます。
- [4] 動的パラメタに対応付ける探索条件の値を読み込みます。
- [5] 動的OPEN文により、探索条件を満たす行が表から導出されます。このとき、USING句で指定した値が、被準備文中の動的パラメタの値として参照されます。USING句で指定した値と、被準備文中の動的パラメタは、その出現順序によって対応付けられます。
- [6] 動的FETCH文により、表からデータを1行ずつ取り出し、各列の値をINTO句で指定されたホスト変数に設定します。
- [7] 動的CLOSE文によって、指定されたカーソル名に対応する表を、カーソルとともに無効状態にします。



#### 注意

動的パラメタに複数列指定ホスト変数を使用するときは、この変数が使用できるSQL文を被準備文として用意してください。使用できないSQL文に指定した場合、動作は保証されません。

### 22.2.3.7 可変長文字列の使用

ここでは、可変長文字列データをCOBOLプログラムのホスト変数で使用方法について説明します。

可変長文字列データを操作する場合には、文字列の長さの情報がが必要です。このため、可変長文字列型のホスト変数は、文字列の長さの情報を格納する符号付き2進項目と、文字列そのものを格納する英数字または日本語項目からなる集団項目として定義されています。

“可変長文字列データを操作するプログラム例”は、COMPANY表から住所を参照するCOBOLプログラム例です。探索条件に用いるホスト変数および取り出したデータを格納するホスト変数は、どちらも可変長文字列データです。

#### 可変長文字列データを操作するプログラム例

```
 :
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 会社名 PIC X(20).
01 電話番号. --
 49 電話番号の長さ PIC S9(4) COMP-5. ↑
 49 電話番号列 PIC X(20). [1]
01 住所.
 49 住所の長さ PIC S9(9) COMP-5. ↓
 49 住所の文字列 PIC X(30). --
01 SQLSTATE PIC X(5).
 EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
 DISPLAY "電話番号から会社を検索します."
 DISPLAY "電話番号を入力してください→"
 WITH NO ADVANCING.
 ACCEPT 電話番号列 FROM CONSOLE. ... [2]
 INSPECT 電話番号列 TALLYING 電話番号の長さ ... [3]
 FOR CHARACTERS BEFORE SPACE.
 EXEC SQL CONNECT TO DEFAULT END-EXEC.
 EXEC SQL
 SELECT NAME, ADDRESS INTO :会社名, :住所 FROM COMPANY ... [4]
 WHERE PHONE = :電話番号
 END-EXEC.
 :
 EXEC SQL ROLLBACK WORK END-EXEC.
 EXEC SQL DISCONNECT DEFAULT END-EXEC.
 STOP RUN.
```

- [1] 可変長文字列のホスト変数を宣言します。
- [2] ACCEPT文により、探索条件に用いるホスト変数の値を“電話番号列”に読み込みます。
- [3] 読み込んだ値の長さを“電話番号の長さ”に設定します。
- [4] 単一行SELECT文により、探索条件を満たす行のADDRESS列の値を取り出します。ホスト変数“住所”には、取り出した列の長さ値が設定されます。



探索条件に用いるホスト変数のデータ型が文字列型または各国語文字列型の場合は、そのホスト変数の長さを列の長さ以下に定義してください。

### 22.2.3.8 複数コネクションでのカーソル操作

ここでは、複数のコネクションを接続してカーソルを操作する方法について説明します。

“複数コネクションでカーソルを操作するプログラム例”の例では、SQLサーバとしてSV1およびSV2があり、さらにそれぞれのサーバ上に、同じ表名と形式を持つ表が存在しています。

#### 複数コネクションでカーソルを操作するプログラム例

```

:
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 製品番号 PIC S9(4) COMP-5.
01 製品名 PIC X(20).
01 在庫数量 PIC S9(9) COMP-5.
01 倉庫番号 PIC S9(4) COMP-5.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
01 NEXTFLAG PIC X(4) VALUE SPACE.
PROCEDURE DIVISION.
EXEC SQL DECLARE CUR9 CURSOR FOR ... [1]
SELECT * FROM STOCK
END-EXEC.
EXEC SQL WHENEVER NOT FOUND GO TO :P-NEXT END-EXEC. ... [2]
P-START.
EXEC SQL
CONNECT TO 'SV1' AS 'CNN1' USER 'summer/w43' ... [3]
END-EXEC.
EXEC SQL
CONNECT TO 'SV2' AS 'CNN2' USER 'tanaka/sky' ... [4]
END-EXEC.
P-CNN2-1.
EXEC SQL OPEN CUR9 END-EXEC. ... [5]
GO TO P-LOOP.
P-NEXT.
IF NEXTFLAG = "NEXT" THEN
GO TO P-CNN1-2
END-IF.
P-CNN1-1.
EXEC SQL SET CONNECTION 'CNN1' END-EXEC. ... [7]
EXEC SQL
INSERT INTO STOCK ... [8]
VALUES(:製品番号, :製品名,
:在庫数量, :倉庫番号)
END-EXEC.
EXEC SQL OPEN CUR9 END-EXEC. ... [9]
MOVE "NEXT" TO NEXTFLAG.
GO TO P-LOOP.
P-CNN1-2.
EXEC SQL CLOSE CUR9 END-EXEC. ... [10]

```

```

EXEC SQL ROLLBACK WORK END-EXEC.
EXEC SQL DISCONNECT 'CNN1' END-EXEC.
P-CNN2-2.
EXEC SQL SET CONNECTION 'CNN2' END-EXEC. ... [11]
EXEC SQL CLOSE CUR9 END-EXEC. ... [12]
EXEC SQL ROLLBACK WORK END-EXEC.
EXEC SQL DISCONNECT CURRENT END-EXEC.
P-END.
STOP RUN.
P-LOOP.
EXEC SQL
 FETCH CUR9 ... [6]
 INTO :製品番号, :製品名, :在庫数量, :倉庫番号
END-EXEC.
:
GO TO P-LOOP.

```

- [1] STOCK表のデータを参照するカーソルを定義します。
- [2] 埋込み例外宣言により、取り出す行がない場合は、手続き名“P-NEXT”の位置へ分岐することを指定します。
- [3] サーバ“SV1”とコネクションを接続し、この接続を“CNN1”とします。
- [4] サーバ“SV2”とコネクションを接続し、この接続を“CNN2”とします。以降、コネクションが変更されるまで、“CNN2”が現在のコネクションになります。
- [5] コネクション“CNN2”上でOPEN文を実行し、カーソル“CUR9”を使用可能な状態にします。
- [6] FETCH文によって表からデータを1行ずつ取り出し、各列の値を対応するホスト変数の領域に設定します。
- [7] 現在のコネクションを“CNN1”に変更します。
- [8] コネクション“CNN1”上のSTOCK表に、INSERT文によってデータを1行挿入します。
- [9] コネクション“CNN1”上でOPEN文を実行し、カーソル“CUR9”を使用可能な状態にします。このときのカーソル“CUR9”は、コネクション“CNN2”上でOPENしたカーソルとは別のカーソルとして扱われます。
- [10] コネクション“CNN1”上でCLOSE文を実行し、カーソル“CUR9”を無効な状態にします。
- [11] 現在のコネクションを“CNN2”に変更します。コネクション“CNN2”上では、カーソル“CUR9”はまだ有効な状態であることに注意してください。
- [12] コネクション“CNN2”上でCLOSE文を実行し、カーソル“CUR9”を無効な状態にします。

## 22.2.4 高度なデータ操作

ここでは、高度なデータ操作を可能にするホスト変数について説明します。

説明中のCOBOLプログラムの例は、“[22.2.3.1 サンプルデータベース](#)”のSTOCK表(在庫表)を使用しています。

### 22.2.4.1 高度なデータ操作を可能とするホスト変数

ここまで説明してきた埋込みSQL文によるデータ操作では、基本的に、埋込みSQL文の1回の実行でデータベース表の1行を操作しました。しかし、以下のホスト変数を指定した埋込みSQL文を使用すると、データベース表の複数の行を一度に操作できます。

- ・ 複数行指定ホスト変数
- ・ 表指定ホスト変数

これらを使用することにより、応用プログラムの性能を向上させることが可能です。



**注意**

この機能はODBCを利用したリモートデータベースアクセス固有の機能です。分散開発など、Windowsプラットフォーム以外で動作するアプリケーションの開発時には、この機能を無効化する必要があります。翻訳オプションNOSQLGRPを指定して翻訳してください。  
[参照]“[A.2.50 SQLGRP \(SQLのホスト変数定義の拡張\)](#)”

### 22.2.4.1.1 複数行指定ホスト変数

#### 機能

表の1つの列に対し、複数の行データを操作することができます。

#### 定義

OCCURS句を持つ繰り返し項目として定義した基本項目です。詳細は、“COBOL文法書”を参照してください。

#### データの参照

複数行指定ホスト変数を相手指定に使用すると、複数行指定ホスト変数の繰り返し回数と同じ件数のデータを参照することができます。

STOCK表から、製品名が“TELEVISION”である行の製品番号(GNO列)を3件取り出すCOBOLプログラムの例を以下に示します。

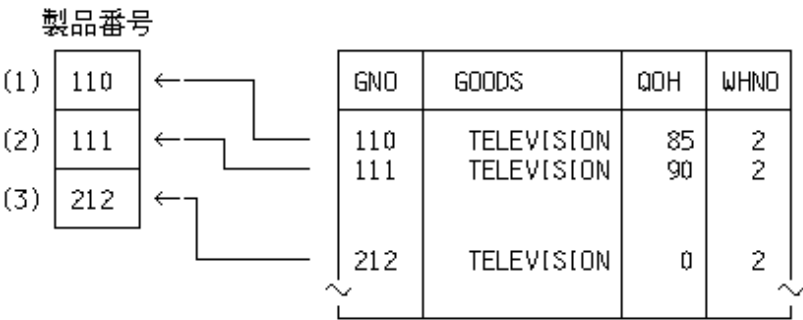
```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品番号 PIC S9(4) COMP-5 OCCURS 3 TIMES. ... [1]
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.
:
EXEC SQL
SELECT GNO FROM STOCK
INTO :製品番号
WHERE GOODS = 'TELEVISION'
END-EXEC.
:

↑
[2]
↓

```

[1] OCCURS句の繰り返し回数を宣言することにより、複数行指定ホスト変数“製品番号”を定義します。

[2] SELECT文で“製品番号”にデータを取り出すと、製品番号(1)には110、製品番号(2)には111、製品番号(3)には212が格納されます。



#### データの挿入

複数行指定ホスト変数を値指定に使用すると、複数行指定ホスト変数の繰り返し回数と同じ件数のデータを挿入することができます。

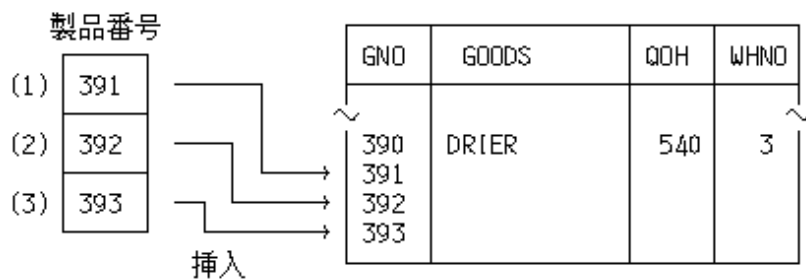
STOCK表の製品番号(GNO列)に3件のデータを挿入するCOBOLプログラムの例を以下に示します。

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品番号 PIC S9(4) COMP-5 OCCURS 3 TIMES.
01 SQLSTATE PIC X(5).
```

```

EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.
:
MOVE 391 TO 製品番号(1).
MOVE 392 TO 製品番号(2).
MOVE 393 TO 製品番号(3).
EXEC SQL
 INSERT INTO STOCK (GNO)
 VALUES (:製品番号)
END-EXEC.
:

```



複数行指定ホスト変数の繰り返し回数と同じ数だけINSERT文が実行され、複数行指定ホスト変数に格納されている先頭の値から順番に、表に挿入されます。つまり、以下のように記述した場合と同じ結果になります。

```

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 製品番号 PIC S9(4) COMP-5.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.
:
MOVE 391 TO 製品番号.
EXEC SQL
 INSERT INTO STOCK (GNO) VALUES (:製品番号)
END-EXEC.
MOVE 392 TO 製品番号.
EXEC SQL
 INSERT INTO STOCK (GNO) VALUES (:製品番号)
END-EXEC.
MOVE 393 TO 製品番号.
EXEC SQL
 INSERT INTO STOCK (GNO) VALUES (:製品番号)
END-EXEC.
:

```

## データの削除

複数行指定ホスト変数を問い合わせ指定に使用すると、複数の条件に合致するデータを一度に削除できます。

STOCK表から、3つの条件に従ってデータの削除を行うCOBOLプログラムの例を以下に示します。

```

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品名 PIC X(20) OCCURS 3 TIMES.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.

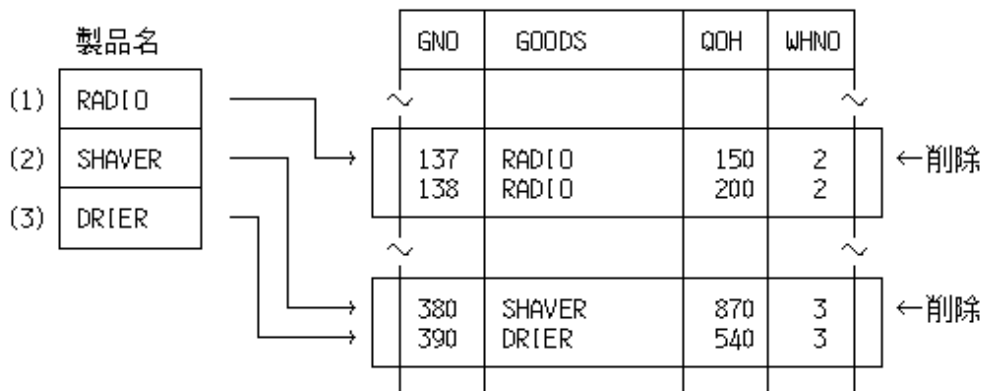
```



```

:
MOVE "RADIO" TO 製品名(1).
MOVE "SHAVER" TO 製品名(2).
MOVE "DRIER" TO 製品名(3).
EXEC SQL
 DELETE FROM STOCK WHERE GOODS = :製品名
END-EXEC.
:

```



これは次のように記述した場合と同じ結果になります。

```

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 製品名 PIC X(20).
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.
:
MOVE "RADIO" TO 製品名.
EXEC SQL
 DELETE FROM STOCK WHERE GOODS = :製品名
END-EXEC.
MOVE "SHAVER" TO 製品名.
EXEC SQL
 DELETE FROM STOCK WHERE GOODS = :製品名
END-EXEC.
MOVE "DRIER" TO 製品名.
EXEC SQL
 DELETE FROM STOCK WHERE GOODS = :製品名
END-EXEC.
:

```

## データの更新

複数行指定ホスト変数を問い合わせ指定に使用すると、複数の条件に合致するデータを一度に更新することができます。

条件に合致する列のデータに、一定の処理を加えながら更新するCOBOLプログラムの例を以下に示します。

```

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品番号 PIC S9(4) COMP-5 OCCURS 3 TIMES.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.

```

```

:
MOVE 212 TO 製品番号(1).
MOVE 215 TO 製品番号(2).
MOVE 226 TO 製品番号(3).
EXEC SQL
 UPDATE STOCK SET QOH = QOH + 100
 WHERE GNO = :製品番号
END-EXEC.
:

```

| 製品番号 |     | GNO | GOODS        | QOH  | WHNO         |
|------|-----|-----|--------------|------|--------------|
| (1)  | 212 | ~   |              |      | ~            |
| (2)  | 215 | 212 | TELEVISION   | 200← | 100+100 (更新) |
|      |     | 215 | VIDEO        | 105← | 5+100 (更新)   |
| (3)  | 226 | 226 | REFRIGERATOR | 108← | 8+100 (更新)   |

また、複数行指定ホスト変数を問合わせ指定と設定句(SET)の値指定に同時に使用して、複数の条件に合致するデータを順番に更新することができます。

以下の例では、設定句および問合わせ指定に複数行指定ホスト変数を使用しています。これらの変数に格納された値は、それぞれの対応する列(GNO列およびQOH列)に対して、配列の先頭から順に使用されます。

```

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品番号 PIC S9(4) COMP-5 OCCURS 3 TIMES.
02 在庫数量 PIC S9(9) COMP-5 OCCURS 3 TIMES.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.
:
MOVE 212 TO 製品番号(1).
MOVE 215 TO 製品番号(2).
MOVE 226 TO 製品番号(3).
MOVE 100 TO 在庫数量(1).
MOVE 200 TO 在庫数量(2).
MOVE 300 TO 在庫数量(3).
EXEC SQL
 UPDATE STOCK SET QOH = QOH + :在庫数量
 WHERE GNO = :製品番号
END-EXEC.
:

```

| 製品番号 在庫数量 |         | GNO | GOODS        | QOH  | WHNO        |
|-----------|---------|-----|--------------|------|-------------|
| (1)       | 212 100 | ~   |              |      | ~           |
| (2)       | 215 200 | 212 | TELEVISION   | 200← | 100+100(更新) |
|           |         | 215 | VIDEO        | 305← | 105+200(更新) |
| (3)       | 226 300 | 226 | REFRIGERATOR | 408← | 108+300(更新) |



## 注意

複数行指定できないドライバもあります。各ドライバの仕様を確認してください。

1つのSQL文に対して複数行指定ホスト変数を複数使用する場合、複数行指定ホスト変数の繰り返し回数は同じ数にしてください。同じ数でない場合、FOR句を使用するようにしてください。[参照]“[22.2.4.4 FOR句による処理行数制御](#)”

複数行指定ホスト変数の繰り返し回数が異なり、かつFOR句の指定もない場合は、複数行指定ホスト変数に指定された最小の繰り返し回数が有効になります。

1つのSQL文で使用される複数行指定ホスト変数の繰り返し回数が、それぞれ異なる場合のCOBOLプログラムの例を以下に示します。以下の例では、繰り返し回数8と繰り返し回数5を持つ複数行指定ホスト変数が定義されていますが、この場合、最小値である5がUPDATE文に対して有効になります。

```

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品番号 PIC S9(4) COMP-5 OCCURS 8 TIMES.
02 在庫数量 PIC S9(9) COMP-5 OCCURS 5 TIMES.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.
:
EXEC SQL
 UPDATE STOCK SET QOH = QOH + :在庫数量
 WHERE GNO = :製品番号
END-EXEC.
:

```

### 22.2.4.1.2 表指定ホスト変数

#### 機能

複数行および複数列にまたがるデータを一度に操作できます。

#### 定義

データベース表の各列に対応する複数行ホスト変数を従属する集団項目として定義します。詳細は、“COBOL文法書”を参照してください。

#### データの参照

表指定ホスト変数を使用すると、表指定ホスト変数の従属項目である複数行指定ホスト変数の繰り返し回数と同じ件数のデータを参照することができます。

STOCK表から、製品名が“TELEVISION”である行のデータ(製品番号、製品名、在庫数量、倉庫番号)を3件取り出すCOBOLプログラムの例を以下に示します。

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品番号 PIC S9(4) COMP-5 OCCURS 3 TIMES.
02 製品名 PIC X(20) OCCURS 3 TIMES.
02 在庫数量 PIC S9(9) COMP-5 OCCURS 3 TIMES.
02 倉庫番号 PIC S9(4) COMP-5 OCCURS 3 TIMES.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.
:
EXEC SQL
SELECT *
INTO :在庫表 FROM STOCK
WHERE GOODS = 'TELEVISION'
END-EXEC.
:
```

[1] 表指定ホスト変数“在庫表”を定義します。

[2] SELECT文でデータを取り出すと、ホスト変数“在庫表”に以下の値が格納されます。

在庫表

| 製品番号 | 製品名        | 在庫数量 | 倉庫番号 |
|------|------------|------|------|
| 110  | TELEVISION | 85   | 2    |
| 111  | TELEVISION | 90   | 2    |
| 212  | TELEVISION | 0    | 2    |

## データの挿入

表指定ホスト変数を使用すると、表指定ホスト変数の従属項目である複数行指定ホスト変数の繰り返し回数と同じ件数のデータを挿入することができます。

STOCK表に、3件のデータを挿入するCOBOLのプログラムの例を以下に示します。

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品番号 PIC S9(4) COMP-5 OCCURS 3 TIMES.
02 製品名 PIC X(20) OCCURS 3 TIMES.
02 在庫数量 PIC S9(9) COMP-5 OCCURS 3 TIMES.
02 倉庫番号 PIC S9(4) COMP-5 OCCURS 3 TIMES.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.
:
MOVE 391 TO 製品番号(1).
MOVE 392 TO 製品番号(2).
MOVE 393 TO 製品番号(3).
MOVE "CASSETTE TAPE" TO 製品名(1).
MOVE "SHAVER" TO 製品名(2).
```

```

MOVE "DRIER" TO 製品名(3).
MOVE 100 TO 在庫数量(1).
MOVE 200 TO 在庫数量(2).
MOVE 300 TO 在庫数量(3).
MOVE 1 TO 倉庫番号(1).
MOVE 2 TO 倉庫番号(2).
MOVE 3 TO 倉庫番号(3).
EXEC SQL
 INSERT INTO STOCK (GNO, GOODS, QOH, WHNO)
 VALUES (:在庫表)
END-EXEC.
:
```

在庫表

| 製品<br>番号 | 製品名           | 在庫<br>数量 | 倉庫<br>番号 |                   | GNO | GOODS         | QOH | WHNO |
|----------|---------------|----------|----------|-------------------|-----|---------------|-----|------|
| 391      | CASSETTE TAPE | 100      | 1        | 挿入<br>→<br>→<br>→ | 390 | DRIER         | 540 | 3    |
| 392      | SHAVER        | 200      | 2        |                   | 391 | CASSETTE TAPE | 100 | 1    |
| 393      | DRIER         | 300      | 3        |                   | 392 | SHAVER        | 200 | 2    |
|          |               |          |          |                   | 393 | DRIER         | 300 | 3    |



## 注意

複数行指定できないドライバもあります。各ドライバの仕様を確認してください。

## 22.2.4.2 動的SQL文で使用方法

複数行指定ホスト変数および表指定ホスト変数は、従来のホスト変数と同じように動的SQL文で使用できます。ただし、これらのホスト変数ができるSQL文は限られています。使用できないSQL文に指定した場合、動作は保証されません。詳細は、“[22.2.10 SQL文と指定可能なホスト変数](#)”を参照してください。

## 22.2.4.3 SQLERRDによる処理行数の確認方法

複数行指定ホスト変数または表指定ホスト変数を使用したデータ操作では、SQLERRD(3)を参照することにより、取り出された行の件数や処理された行の件数を知ることができます。

## 使用例

- SELECT文またはFETCH文で複数行データを取り出す場合、SQLERRD(3)には取り出したデータの件数が格納されます。例えば、繰り返し回数100を持つ複数行指定ホスト変数を使用し、取り出す対象のデータが50件である場合、SQLERRD(3)には50が格納されます。
- INSERT文、UPDATE文(検索)またはDELETE文(検索)で複数行データを操作する場合、SQL文のデータ操作によって処理された行の件数がSQLERRD(3)に格納されます。

STOCK表のWHNOの条件に従ってQOHを更新する動作を2回行うCOBOLプログラムの例を以下に示します。WHNO=1に対するQOHの更新が6件、WHNO=2に対するQOHの更新が11件、合わせて17件であるという値がSQLERRD(3)に格納されます。

```

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 在庫数量 PIC S9(9) COMP-5 OCCURS 2 TIMES.
02 倉庫番号 PIC S9(4) COMP-5 OCCURS 2 TIMES.
01 SQLINFOA.
```

```

02 SQLERRD PIC S9(9) COMP-5 OCCURS 6 TIMES.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.
:
MOVE 100 TO 在庫数量(1).
MOVE 200 TO 在庫数量(2).
MOVE 1 TO 倉庫番号(1).
MOVE 2 TO 倉庫番号(2).
EXEC SQL
UPDATE STOCK SET QOH = QOH + :在庫数量
WHERE WHNO = :倉庫番号
END-EXEC.
DISPLAY SQLERRD(3).
:

```



## 注意

取り出し対象のデータ件数が繰り返し回数より多くても少なくても、SQLSTATEの値は“データなし”にはなりません。SQLSTATEの値が“データなし”になるのは、1つ以上の行が取り出されなかった場合です。

CALL文で呼び出されたストアドプロシージャでの変更行は、SQLERRD(3)では保証されません。

### 22.2.4.4 FOR句による処理行数制御

複数行指定ホスト変数または表指定ホスト変数を使用したデータ操作では、FOR句を指定することで、処理行数または処理回数を制御することができます。

#### データ参照

STOCK表の先頭から5件分のデータを参照するCOBOLプログラムの例を以下に示します。

繰り返し回数が10である複数行指定ホスト変数を従属する表指定ホスト変数“在庫表”を使用し、FETCH文でデータを取り出します。この場合、複数行指定ホスト変数の繰り返し回数は10であっても、FETCH文のFOR句に5が指定されているため、5件分のデータが表指定ホスト変数に格納されます。

```

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品番号 PIC S9(4) COMP-5 OCCURS 10 TIMES.
02 製品名 PIC X(20) OCCURS 10 TIMES.
02 在庫数量 PIC S9(9) COMP-5 OCCURS 10 TIMES.
02 倉庫番号 PIC S9(4) COMP-5 OCCURS 10 TIMES.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.
:
EXEC SQL
DECLARE CUR1 CURSOR FOR SELECT * FROM STOCK
END-EXEC.
:
EXEC SQL
OPEN CUR1
END-EXEC.
:
EXEC SQL
FOR 5
FETCH CUR1 INTO :在庫表
END-EXEC.
:

```

また、“22.2.3.2.1 表の全行を参照する”で示した表の全体を参照するプログラムの例は、次のように書くことができます。

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品番号 PIC S9(4) COMP-5 OCCURS 10 TIMES.
02 製品名 PIC X(20) OCCURS 10 TIMES.
02 在庫数量 PIC S9(9) COMP-5 OCCURS 10 TIMES.
02 倉庫番号 PIC S9(4) COMP-5 OCCURS 10 TIMES.
01 行数 PIC S9(9) COMP-5.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.
:
EXEC SQL
 DECLARE CUR1 CURSOR FOR SELECT * FROM STOCK
END-EXEC.
:
EXEC SQL
 SELECT COUNT(*) INTO :行数 FROM STOCK
END-EXEC.
:
EXEC SQL OPEN CUR1 END-EXEC.
:
EXEC SQL
 FOR :行数
 FETCH CUR1 INTO :在庫表
END-EXEC.
:
```

[1] 表指定ホスト変数“在庫表”を定義します。

[2] 集合関数COUNT(\*)で表の行数を求めます。

[3] 表のすべての行のデータを“在庫表”に取り出します。

## データ挿入

STOCK表に3件の行データを挿入するCOBOLプログラムの例を以下に示します。

繰り返し回数が10の複数行指定ホスト変数を従属する表指定ホスト変数“在庫表”を使用し、INSERT文でデータを挿入します。この場合、複数行指定ホスト変数の繰り返し回数が10であっても、INSERT文のFOR句に3が指定されているため、STOCK表には3件のデータが挿入されます。

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品番号 PIC S9(4) COMP-5 OCCURS 10 TIMES.
02 製品名 PIC X(20) OCCURS 10 TIMES.
02 在庫数量 PIC S9(9) COMP-5 OCCURS 10 TIMES.
02 倉庫番号 PIC S9(4) COMP-5 OCCURS 10 TIMES.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
:
PROCEDURE DIVISION.
:
MOVE 391 TO 製品番号(1).
MOVE 392 TO 製品番号(2).
MOVE 393 TO 製品番号(3).
MOVE "CASSETTE TAPE" TO 製品名(1).
MOVE "SHAVER" TO 製品名(2).
MOVE "DRIER" TO 製品名(3).
MOVE 100 TO 在庫数量(1).
MOVE 200 TO 在庫数量(2).
MOVE 300 TO 在庫数量(3).
MOVE 1 TO 倉庫番号(1).
```

```

MOVE 2 TO 倉庫番号 (2).
MOVE 3 TO 倉庫番号 (3).
EXEC SQL
 FOR 3
 INSERT INTO STOCK (GNO, GOODS, QOH, WHNO)
 VALUES (:在庫表)
END-EXEC.
 :
```



## 注意

FOR句に指定する値は、複数行指定ホスト変数の繰り返し回数と同じか、それより小さい値でなければなりません。繰り返し回数より大きい値を指定した場合、エラーが出力されます。

## 22.2.4.5 FETCH PRIOR文によるデータの取得

FETCH PRIOR文は、最後にFETCHされた行の前のデータを取り出します。

ここでは、FETCH PRIOR文を使用してデータを取得する方法について説明します。

### FETCH PRIOR文の実行に必要なオプションの指定

デフォルトのカーソルの設定ではカーソルは順方向専用で作成されるため、FETCH PRIOR文を実行することができません。FETCH PRIOR文を実行するためには以下のオプションを設定してください。オプションの詳細は、“[22.2.8.1.2 ODBC情報ファイルの作成](#)”を参照してください。

FETCH PRIOR文の実行に必要なオプション指定値

| オプション            | 設定値                                              |
|------------------|--------------------------------------------------|
| @SQL_CURSOR_TYPE | KEYSET DRIVEN<br>または<br>STATIC<br>または<br>DYNAMIC |

@SQL\_CURSOR\_TYPEオプションにはFORWARD\_ONLY以外を指定してください。FORWARD\_ONLYを指定、または@SQL\_CURSOR\_TYPEオプションを指定しない場合、順方向専用カーソルとなりFETCH PRIOR文を実行することができません。

### FETCH PRIOR文によるデータ取得

“STOCK表”の先頭から3件のデータをそれぞれFETCH NEXT文で取り出し、FETCH PRIOR文で“STOCK表”の先頭まで戻るCOBOLプログラムの例を以下に示します。

単数行指定ホスト変数を従属する表指定ホスト変数“在庫表”を使用し、FETCH NEXT文で3件データを取り出し、取り出したらFETCH PRIOR文で2件のデータを取り出します。

FETCH PRIORプログラム例

```

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品番号 PIC S9(4) COMP-5.
02 製品名 PIC X(20).
02 在庫数量 PIC S9(9).
02 倉庫番号 PIC S9(4).
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
EXEC SQL
 DECLARE CUR1 CURSOR FOR SELECT * FROM STOCK
END-EXEC.
EXEC SQL OPEN CUR1 END-EXEC.
EXEC SQL FETCH NEXT CUR1 INTO :在庫表 END-EXEC.
```



```

EXEC SQL FETCH NEXT CUR1 INTO :在庫表 END-EXEC. ... [4]
EXEC SQL FETCH NEXT CUR1 INTO :在庫表 END-EXEC. ... [5]
EXEC SQL FETCH PRIOR CUR1 INTO :在庫表 END-EXEC. ... [6]
EXEC SQL FETCH PRIOR CUR1 INTO :在庫表 END-EXEC. ... [7]
EXEC SQL CLOSE CUR1 END-EXEC. ... [8]
EXEC SQL ROLLBACK WORK END-EXEC. ... [9]
EXEC SQL DISCONNECT DEFAULT END-EXEC. ... [10]
STOP RUN.

```

- [1] 表指定ホスト変数"在庫表"を定義します。
- [2] カーソル"CUR1"をオープンします。
- [3] 在庫表の1件目のデータを"在庫表"に取り出します。
- [4] 在庫表の2件目のデータを"在庫表"に取り出します。
- [5] 在庫表の3件目のデータを"在庫表"に取り出します。
- [6] 在庫表の2件目のデータを"在庫表"に取り出します。
- [7] 在庫表の1件目のデータを"在庫表"に取り出します。
- [8] カーソル"CUR1"をクローズします。
- [9] ROLLBACK文を実行し、トランザクションを終了します。
- [10] DISCONNECT文を実行し、サーバとの接続を切断します。

### 複数行指定ホスト変数を使用したFETCH PRIOR文によるデータ取得

"STOCK表"の先頭から9件のデータを3件ずつ3回のFETCH NEXT文で取り出し、"STOCK表"の先頭まで3件ずつ2回のFETCH PRIOR文で戻るCOBOLプログラムの例を以下に示します。

複数行指定ホスト変数を従属する表指定ホスト変数"在庫表"を使用し、1回のFETCH NEXT文で3件データ取り出しを3回行い、取り出したら1回のFETCH PRIOR文で3件のデータ取り出しを2回行います。

#### 複数行指定ホスト変数を使用したFETCH PRIORプログラム例

```

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 在庫表.
02 製品番号 PIC S9(4) COMP-5 OCCURS 3.
02 製品名 PIC X(20) OCCURS 3.
02 在庫数量 PIC S9(9) OCCURS 3.
02 倉庫番号 PIC S9(4) OCCURS 3.
01 SQLSTATE PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
EXEC SQL
 DECLARE CUR1 CURSOR FOR SELECT * FROM STOCK
END-EXEC.
EXEC SQL OPEN CUR1 END-EXEC.
EXEC SQL FOR 3 FETCH NEXT CUR1 INTO :在庫表 END-EXEC.
EXEC SQL FOR 3 FETCH NEXT CUR1 INTO :在庫表 END-EXEC.
EXEC SQL FOR 3 FETCH NEXT CUR1 INTO :在庫表 END-EXEC.
EXEC SQL FOR 3 FETCH PRIOR CUR1 INTO :在庫表 END-EXEC.
EXEC SQL FOR 3 FETCH PRIOR CUR1 INTO :在庫表 END-EXEC.
EXEC SQL CLOSE CUR1 END-EXEC.
EXEC SQL ROLLBACK WORK END-EXEC.
EXEC SQL DISCONNECT DEFAULT END-EXEC.
STOP RUN.

```

- [1] 表指定ホスト変数"在庫表"を定義します。
- [2] カーソル"CUR1"をオープンします。
- [3] 在庫表の1～3件目のデータを"在庫表"に取り出します。

STOCK-LIST

|     | PRODUCT-<br>-NUMBER | PRODUCT-<br>NAME | QUANTITY-<br>IN-STOCK | WAREHOUSE<br>-NUMBER |   | GNO | GOODS           | QOH | WHNO |
|-----|---------------------|------------------|-----------------------|----------------------|---|-----|-----------------|-----|------|
| (1) | 110                 | TELEVISION       | 85                    | 2                    | ← | 110 | TELEVISION      | 85  | 2    |
| (2) | 111                 | TELEVISION       | 90                    | 2                    | ← | 111 | TELEVISION      | 90  | 2    |
| (3) | 123                 | REFRIGERATOR     | 60                    | 1                    | ← | 123 | REFRIGERATOR    | 60  | 1    |
|     |                     |                  |                       |                      |   | 124 | REFRIGERATOR    | 75  | 1    |
|     |                     |                  |                       |                      |   | 137 | RADIO           | 150 | 2    |
|     |                     |                  |                       |                      |   | 138 | RADIO           | 200 | 2    |
|     |                     |                  |                       |                      |   | 140 | CASSET DECK     | 120 | 2    |
|     |                     |                  |                       |                      |   | 141 | CASSET DECK     | 80  | 2    |
|     |                     |                  |                       |                      |   | 200 | AIR CONDITIONER | 4   | 1    |
|     |                     |                  |                       |                      |   | 201 | AIR CONDITIONER | 15  | 1    |
|     |                     |                  |                       |                      |   | 212 | TELEVISION      | 0   | 2    |
|     |                     |                  |                       |                      |   |     | :               |     |      |

[4] 在庫表の4～6件目のデータを"在庫表"に取り出します。

|     | PRODUCT-<br>-NUMBER | PRODUCT-<br>NAME | QUANTITY-<br>IN-STOCK | WAREHOUSE<br>-NUMBER |   | GNO | GOODS           | QOH | WHNO |
|-----|---------------------|------------------|-----------------------|----------------------|---|-----|-----------------|-----|------|
| (1) | 124                 | REFRIGERATOR     | 75                    | 1                    | ← | 110 | TELEVISION      | 85  | 2    |
| (2) | 137                 | RADIO            | 150                   | 2                    | ← | 111 | TELEVISION      | 90  | 2    |
| (3) | 138                 | RADIO            | 200                   | 2                    | ← | 123 | REFRIGERATOR    | 60  | 1    |
|     |                     |                  |                       |                      |   | 124 | REFRIGERATOR    | 75  | 1    |
|     |                     |                  |                       |                      |   | 137 | RADIO           | 150 | 2    |
|     |                     |                  |                       |                      |   | 138 | RADIO           | 200 | 2    |
|     |                     |                  |                       |                      |   | 140 | CASSET DECK     | 120 | 2    |
|     |                     |                  |                       |                      |   | 141 | CASSET DECK     | 80  | 2    |
|     |                     |                  |                       |                      |   | 200 | AIR CONDITIONER | 4   | 1    |
|     |                     |                  |                       |                      |   | 201 | AIR CONDITIONER | 15  | 1    |
|     |                     |                  |                       |                      |   | 212 | TELEVISION      | 0   | 2    |
|     |                     |                  |                       |                      |   |     | :               |     |      |

[5] 在庫表の7～9件目のデータを"在庫表"に取り出します。

|     | PRODUCT-<br>-NUMBER | PRODUCT-NAME    | QUANTITY-<br>IN-STOCK | WAREHOUSE<br>-NUMBER |   | GNO | GOODS           | QOH | WHNO |
|-----|---------------------|-----------------|-----------------------|----------------------|---|-----|-----------------|-----|------|
| (1) | 140                 | CASSET DECK     | 120                   | 2                    | ← | 110 | TELEVISION      | 85  | 2    |
| (2) | 141                 | CASSET DECK     | 80                    | 2                    | ← | 111 | TELEVISION      | 90  | 2    |
| (3) | 200                 | AIR CONDITIONER | 4                     | 1                    | ← | 123 | REFRIGERATOR    | 60  | 1    |
|     |                     |                 |                       |                      |   | 124 | REFRIGERATOR    | 75  | 1    |
|     |                     |                 |                       |                      |   | 137 | RADIO           | 150 | 2    |
|     |                     |                 |                       |                      |   | 138 | RADIO           | 200 | 2    |
|     |                     |                 |                       |                      |   | 140 | CASSET DECK     | 120 | 2    |
|     |                     |                 |                       |                      |   | 141 | CASSET DECK     | 80  | 2    |
|     |                     |                 |                       |                      |   | 200 | AIR CONDITIONER | 4   | 1    |
|     |                     |                 |                       |                      |   | 201 | AIR CONDITIONER | 15  | 1    |
|     |                     |                 |                       |                      |   | 212 | TELEVISION      | 0   | 2    |
|     |                     |                 |                       |                      |   |     | :               |     |      |

[6] 在庫表の4～6件目のデータを"在庫表"に取り出します。添え字(1)のデータ項目から順にデータが設定されます

| STOCK-LIST         |                  |                       |                      |
|--------------------|------------------|-----------------------|----------------------|
| PRODUCT<br>-NUMBER | PRODUCT-<br>NAME | QUANTITY-<br>IN-STOCK | WAREHOUSE<br>-NUMBER |
| (1) 124            | REFRIGERATOR     | 75                    | 1                    |
| (2) 137            | RADIO            | 150                   | 2                    |
| (3) 138            | RADIO            | 200                   | 2                    |

| GNO | GOODS           | QOH | WHNO |
|-----|-----------------|-----|------|
| 110 | TELEVISION      | 85  | 2    |
| 111 | TELEVISION      | 90  | 2    |
| 123 | REFRIGERATOR    | 60  | 1    |
| 124 | REFRIGERATOR    | 75  | 1    |
| 137 | RADIO           | 150 | 2    |
| 138 | RADIO           | 200 | 2    |
| 140 | CASSET DECK     | 120 | 2    |
| 141 | CASSET DECK     | 80  | 2    |
| 200 | AIR CONDITIONER | 4   | 1    |
| 201 | AIR CONDITIONER | 15  | 1    |
| 212 | TELEVISION      | 0   | 2    |
|     | :               |     |      |

[7] 在庫表の1～3件目のデータを"在庫表"に取り出します。添え字(1)のデータ項目から順にデータが設定されます。

STOCK-LIST

| PRODUCT<br>-NUMBER | PRODUCT-<br>NAME | QUANTITY-<br>IN-STOCK | WAREHOUSE<br>-NUMBER |   | GNO | GOODS           | QOH | WHNO |
|--------------------|------------------|-----------------------|----------------------|---|-----|-----------------|-----|------|
| (1) 110            | TELEVISION       | 85                    | 2                    | ← | 110 | TELEVISION      | 85  | 2    |
| (2) 111            | TELEVISION       | 90                    | 2                    | ← | 111 | TELEVISION      | 90  | 2    |
| (3) 123            | REFRIGERATOR     | 60                    | 1                    | ← | 123 | REFRIGERATOR    | 60  | 1    |
|                    |                  |                       |                      |   | 124 | REFRIGERATOR    | 75  | 1    |
|                    |                  |                       |                      |   | 137 | RADIO           | 150 | 2    |
|                    |                  |                       |                      |   | 138 | RADIO           | 200 | 2    |
|                    |                  |                       |                      |   | 140 | CASSET DECK     | 120 | 2    |
|                    |                  |                       |                      |   | 141 | CASSET DECK     | 80  | 2    |
|                    |                  |                       |                      |   | 200 | AIR CONDITIONER | 4   | 1    |
|                    |                  |                       |                      |   | 201 | AIR CONDITIONER | 15  | 1    |
|                    |                  |                       |                      |   | 212 | TELEVISION      | 0   | 2    |
|                    |                  |                       |                      |   |     | :               |     |      |

[8] カーソル"CUR1"をクローズします。

[9] ROLLBACK文を実行し、トランザクションを終了します。

[10] DISCONNECT文を実行し、サーバとの接続を切断します。

## 22.2.5 ストアドプロシージャの呼出し

ここでは、ストアドプロシージャの呼出しについて説明します。

### 22.2.5.1 ストアドプロシージャとは

ストアドプロシージャとは、サーバに登録された処理手続きのことです。このストアドプロシージャを、クライアント側から呼び出し、サーバ側で処理手続きを実行します。ストアドプロシージャを使用することによる利点は、各データベースにより異なりますが、共通して以下のような利点があります。

- ・ 処理手続きの実行速度の向上
- ・ クライアント、サーバ間の通信負荷の減少
- ・ 開発/保守の生産性の向上
- ・ セキュリティの向上

ストアドプロシージャの詳細、作成方法については、各データベース管理システムのマニュアルを参照してください。



## 注意

ストアードプロシージャの呼び出しには時間がかかる場合があります。そのため、ストアードプロシージャ内の処理が軽い場合、パフォーマンスを発揮できない場合があります。その場合はストアードプロシージャを使用せずにプログラミングしてください。

### 22.2.5.2 ストアドプロシージャの呼出し例

ストアードプロシージャを呼び出すCOBOLプログラムの例を以下に示します。ここでは、ストアードプロシージャPROCを呼び出しています。SQLERRD(1)により、ストアードプロシージャの戻り値を取得することができます。

ストアードプロシージャを呼び出すプログラム例

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 入力変数 PIC S9(4) COMP-5. ... [1]
01 出力変数 PIC S9(4) COMP-5. ... [1]
SQLINFOA.
02 SQLERRD PIC S9(9) COMP-5 OCCURS 6 TIMES.
EXEC SQL END DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
EXEC SQL CONNECT TO DEFAULT END-EXEC. ... [2]
MOVE 100 TO 入力変数. ... [3]
EXEC SQL
CALL PROC (:入力変数, :出力変数) ... [4]
END-EXEC.
DISPLAY SQLERRD(1). ... [5]
EXEC SQL ROLLBACK WORK END-EXEC. ... [6]
EXEC SQL DISCONNECT DEFAULT END-EXEC. ... [7]
STOP RUN.
```

[1] 作業場所節に埋込みSQL宣言節を記述し、呼び出すストアードプロシージャの引数をすべてホスト変数として定義します。ホスト変数宣言の規則については、“COBOL文法書”を参照してください。

[2] CONNECT文を実行し、サーバとのコネクションを接続します。

[3] ストアドプロシージャの入力パラメータをホスト変数に設定します。

[4] CALL文を実行し、サーバに登録されたPROCストアードプロシージャを呼び出します。呼び出し後、ストアードプロシージャからの出力パラメータがホスト変数に設定されます。

[5] ストアドプロシージャの戻り値を返します。

[6] ROLLBACK文を実行し、トランザクションを終了します。

[7] DISCONNECT文を実行し、サーバとのコネクションを切断します。



## 注意

ストアードプロシージャ呼出しのCALL文に指定可能な引数の数は700個です。

ストアードプロシージャが戻り値を返すことが可能かどうかは、データベースによって異なります。使用するデータベースのマニュアルを確認してください。

### 22.2.6 オブジェクト指向プログラミング機能を使用したデータベースアクセス

ここでは、オブジェクト指向プログラミング機能を使用したデータベースアクセス方法について説明します。

[参照]“[第16章 オブジェクト指向プログラミング機能～基本的な使い方～](#)”、“[第17章 オブジェクト指向プログラミングの開発](#)”、“[第18章 オブジェクト指向プログラミング機能～さらに進んだ使い方～](#)”

#### 22.2.6.1 サンプルデータベース

説明中のクラス定義およびプログラム定義の例では、“[22.2.3.1 サンプルデータベース](#)”のSTOCK表(在庫表)を使用しています。

## 22.2.6.2 クラス定義中で表のデータを取り出しデータを更新する

クラス定義中で表のデータを取り出し、データを更新する方法について説明します。

“データベースアクセスのクラス定義例”は、在庫管理メソッドの定義を含むCOBOLのクラス定義です。在庫管理メソッドは、在庫管理対象のデータを取り出し、入庫または出庫を切り分けて在庫数量を計算し、再び表に格納するメソッドです。

また、“データベースアクセスクラスのメソッドを呼び出すプログラムの例”は、在庫管理メソッドを呼び出すCOBOLプログラムの例です。

### データベースアクセスのクラス定義例

```
CLASS-ID. DBACCESS-CLASS INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS FJBASE.
 OBJECT.
 DATA DIVISION.
 PROCEDURE DIVISION.
 METHOD-ID. ZAIKO-KANRI.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 SQLSTATE PIC X(5).
01 在庫数量 PIC S9(9) COMP-5.
 EXEC SQL END DECLARE SECTION END-EXEC.
 LINKAGE SECTION.
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 製品番号 PIC S9(4) COMP-5.
 EXEC SQL END DECLARE SECTION END-EXEC.
01 入出区別 PIC S9(4) COMP-5.
01 入出数量 PIC S9(9) COMP-5.
 PROCEDURE DIVISION USING 製品番号 入出区別 入出数量.
 EXEC SQL
 SELECT QOH INTO :在庫数量 FROM STOCK WHERE GNO = :製品番号
 END-EXEC.
 IF 入出区別 = 1 THEN
 COMPUTE 在庫数量 = 在庫数量 + 入出数量
 ELSE
 COMPUTE 在庫数量 = 在庫数量 - 入出数量.
 EXEC SQL
 UPDATE STOCK SET QOH = :在庫数量 WHERE GNO = :製品番号
 END-EXEC.
 END METHOD ZAIKO-KANRI.
 END OBJECT.
END CLASS DBACCESS-CLASS.
```

[1] メソッドデータを定義します。

[2] 在庫管理メソッドのインタフェースを定義します。

[3] 在庫管理メソッドを定義します。このメソッドでは、該当する製品番号の在庫数量をSTOCK表(在庫表)より取り出し、入庫または出庫を切り分けて在庫数量を計算し、再び表に格納します。

### データベースアクセスクラスのメソッドを呼び出すプログラムの例

```

CONFIGURATION SECTION.
REPOSITORY.
 CLASS DBACCESS-CLASS.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 SQLSTATE PIC X(5).
 EXEC SQL END DECLARE SECTION END-EXEC.
01 オブジェクト.
```

```

02 DBACCESS-OBJECT OBJECT REFERENCE DBACCESS-CLASS.
01 製品番号 PIC S9(4) COMP-5.
01 入出区別 PIC S9(4) COMP-5.
01 入出数量 PIC S9(9) COMP-5.
01 終了要求 PIC X(1).
PROCEDURE DIVISION.
EXEC SQL CONNECT TO DEFAULT END-EXEC.
INVOKE DBACCESS-CLASS "NEW" RETURNING DBACCESS-OBJECT. ... [1]
PERFORM TEST AFTER UNTIL 終了要求 = "Y" -----
 DISPLAY "製品番号、入出区別(1or2)、入出数量を入力してください" ↑
 ACCEPT 製品番号
 ACCEPT 入出区別
 ACCEPT 入出数量 [2]
 INVOKE DBACCESS-OBJECT "ZAIKO-KANRI"
 USING 製品番号 入出区別 入出数量
 DISPLAY "在庫管理を終了しますか？(Y/N)"
 ACCEPT 終了要求 ↓
END-PERFORM. -----
EXEC SQL COMMIT WORK END-EXEC.
EXEC SQL DISCONNECT DEFAULT END-EXEC.
STOP RUN.

```

[1] “NEW”メソッドによりオブジェクトインスタンスを生成します。

[2] 製品番号、入出区別、入出数量の入力を要求し、在庫管理メソッドを呼び出します。終了要求に“Y”が入力されるまで処理を繰り返し行います。

### 22.2.6.3 コネクションをオブジェクトインスタンス単位で利用する

ここでは、コネクションをオブジェクトインスタンス単位で利用する方法について説明します。これにより、分散オブジェクト環境下で、オブジェクトインスタンス単位でトランザクション管理を行うことができます。

#### コネクションをオブジェクトインスタンス単位で利用するプログラムを作成する

コネクションをオブジェクトインスタンス単位で利用するクラス定義例

```

CLASS-ID. DBACCESS-CLASS INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS FJBASE.
OBJECT.
DATA DIVISION.
PROCEDURE DIVISION.
METHOD-ID. ZAIKO-KANRI.
DATA DIVISION.
WORKING-STORAGE SECTION.
 EXEC SQL BEGIN DECLARE SECTION END-EXEC. -----
01 SQLSTATE PIC X(5). ↑
01 在庫数量 PIC S9(9) COMP-5.
01 製品番号 PIC S9(4) COMP-5.
01 入出区別 PIC S9(4) COMP-5. [1]
01 入出数量 PIC S9(9) COMP-5.
 EXEC SQL END DECLARE SECTION END-EXEC.
01 終了要求 PIC X(1). ↓
01 反映要求 PIC X(1). -----
PROCEDURE DIVISION. -----
EXEC SQL CONNECT TO DEFAULT END-EXEC. ↑
PERFORM TEST AFTER UNTIL 終了要求 = "Y"
 DISPLAY "製品番号、入出区別(1or2)、入出数量を入力してください"
 ACCEPT 製品番号
 ACCEPT 入出区別
 ACCEPT 入出数量
 IF 入出区別 = 1 THEN

```

```

EXEC SQL
 UPDATE STOCK SET QOH = QOH + :入出数量
 WHERE GNO = :製品番号
END-EXEC
ELSE
 EXEC SQL
 UPDATE STOCK SET QOH = QOH - :入出数量
 WHERE GNO = :製品番号
 END-EXEC
END-IF
EXEC SQL
 SELECT QOH INTO :在庫数量 FROM STOCK
 WHERE GNO = :製品番号
END-EXEC
DISPLAY "現在の在庫数量=" 在庫数量
DISPLAY "変更結果を反映してもいいですか？(Y/N)"
ACCEPT 反映要求
IF 反映要求 = "Y" THEN
 EXEC SQL COMMIT WORK END-EXEC
ELSE
 EXEC SQL ROLLBACK WORK END-EXEC
END-IF
DISPLAY "在庫管理を終了しますか？(Y/N)"
ACCEPT 終了要求
END-PERFORM.
EXEC SQL DISCONNECT DEFAULT END-EXEC.
EXIT METHOD.
END METHOD ZAIKO-KANRI.
END OBJECT.
END CLASS DBACCESS-CLASS.

```

[2]

↓

-----

[1] メソッドデータを定義します。

[2] 在庫管理メソッドを定義します。このメソッドでは、コネクションの接続からデータ操作、コネクションの切断まで一連のデータベースアクセスを行います。データ操作では、製品番号、入出区別、入出数量の入力を要求し、入庫または出庫を切り分けて在庫数量を再計算します。次に再計算後の在庫数量を表示し、変更結果の反映有無を入力させます。データ操作は、終了要求に“Y”が入力されるまで繰り返し行います。

データベースアクセスクラスのメソッド(在庫管理メソッド)を呼び出すプログラムの例

```

:
CONFIGURATION SECTION.
REPOSITORY.
 CLASS DBACCESS-CLASS.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJECT-COUNT PIC S9(1).
01 オブジェクト.
02 DBACCESS-OBJECT OCCURS 3 OBJECT REFERENCE DBACCESS-CLASS.
PROCEDURE DIVISION.
 PERFORM WITH TEST AFTER VARYING OBJECT-COUNT FROM 1 BY 1
 UNTIL OBJECT-COUNT = 3
 INVOKE DBACCESS-CLASS "NEW"
 RETURNING DBACCESS-OBJECT(OBJECT-COUNT)
 INVOKE DBACCESS-OBJECT(OBJECT-COUNT) "ZAIKO-KANRI"
END-PERFORM.
STOP RUN.

```

-----

↑

[1]

↓

-----

[1] "NEW"メソッドにより3つのオブジェクトインスタンスを生成し、それぞれの在庫管理メソッドを呼び出します。複数クライアントからの要求に対して、それぞれの在庫管理メソッドが実行されます。

## コネクションをオブジェクトインスタンス単位で利用するプログラムを実行する

コネクションをオブジェクトインスタンス単位で利用するには、ODBC情報ファイルのコネクション有効範囲にオブジェクトインスタンスを指定して実行します。

[参照]“[22.2.8.1.2 ODBC情報ファイルの作成](#)”

ODBC情報ファイルは、ODBC情報設定ツールを使用して作成してください。

[参照]“[22.2.8.2 ODBC情報設定ツールの使い方](#)”

コネクション有効範囲にオブジェクトインスタンスを指定することにより、オブジェクトインスタンス単位でコネクションを利用して動作します。

## 22.2.7 プログラムの翻訳・リンク

---

ODBCドライバ経由で、埋込みSQLを使用してデータベースにアクセスするCOBOLプログラムの翻訳は、特定の翻訳オプションを指定することなく、COBOLコンパイラで翻訳することができます。



### 注意

- ・ ホスト変数名に使用している英字の大きさを区別する場合は、翻訳オプションNOALPHALを指定して翻訳します。[参照]“[A.2.2 ALPHAL \(英小文字の扱い\)](#)”
- ・ 利用者が定義する名前に埋込みSQL文のキーワードは使用できません。[参照]“[22.2.9 埋込みSQL文のキーワード一覧](#)”
- ・ マルチスレッドプログラムの翻訳は、翻訳オプションTHREAD(MULTI)を指定する必要があります。

## 22.2.8 プログラムの実行

---

ここでは、プログラムを実行するために必要な実行環境の構築およびODBC情報設定ツールの使い方について説明します。

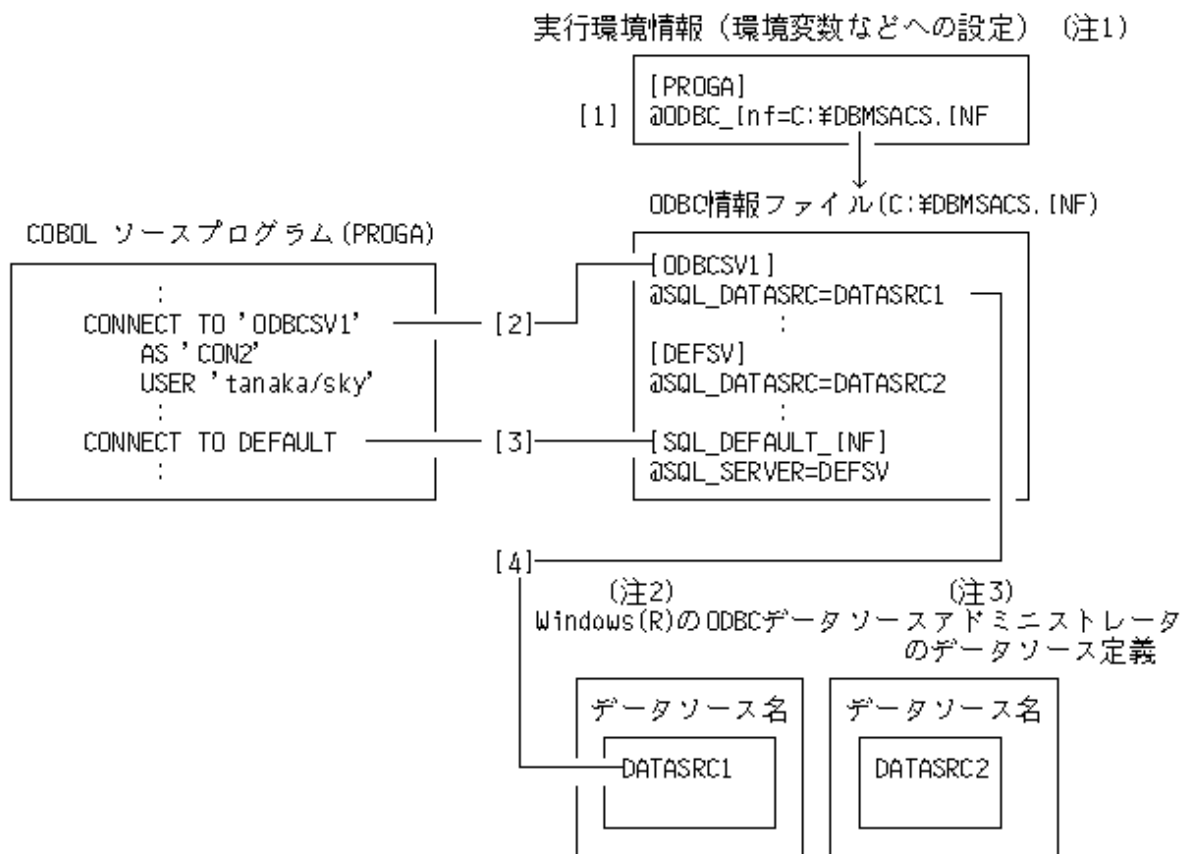
### 22.2.8.1 実行環境の構築

プログラムを実行するためには、実行環境情報を設定した実行用の初期化ファイルおよびODBC情報ファイルが必要になります。各ファイルに設定する情報は、以下に示すように対応付けて指定します。

“[図22.4 各情報ファイルの関連](#)”に、それぞれのファイルの設定情報がどのように関連付けられるかを示します。



図22.4 各情報ファイルの関連



注1: 詳細は、“5.3.2 実行環境情報の設定方法”を参照してください。

注2: データソースとは、ODBCドライバ、ネットワークシステムおよびデータベースなどの環境全般を総称したものです。

[1] ODBC情報ファイル名(C:\¥DBMSACS.INF)を指定します。

[2] CONNECT文にサーバ名を記述した場合、ODBC情報ファイルにサーバ名を指定します。

[3] CONNECT文にDEFAULTを記述した場合、ODBC情報ファイルにデフォルトコネクション情報の定義を示す固定文字列を指定します。

[4] ODBC情報ファイルの各サーバのデータソース名の定義には、WindowsシステムのODBCデータソースアドミニストレータで定義したデータソース名を指定します。

### 22.2.8.1.1 実行環境情報の設定

クライアントとサーバ間の連携ソフトウェアとしてODBC環境を選択する場合には、以下の情報を環境変数情報に設定してください。[参照]“5.3.2 実行環境情報の設定方法”

#### @ODBC\_Inf (ODBC情報ファイルの指定)

@ODBC\_Inf=C:\¥DBMSACS.INF

COBOLランタイムシステムがODBCを使用するために参照するファイル名を指定します。[参照]“22.2.8.1.2 ODBC情報ファイルの作成”

## 22.2.8.1.2 ODBC情報ファイルの作成

ODBC情報ファイルには、主に、クライアントとサーバ間のコネクション(CONNECT文などで指定する)を確立するために必要な情報が設定されています。

ODBC情報ファイルは、ODBC情報設定ツールを使用して作成してください。[参照]“22.2.8.2 ODBC情報設定ツールの使い方”

ODBC情報ファイルの内容は、サーバ情報、デフォルトコネクション情報およびコネクション有効範囲に分類されます。



パスワードをODBC情報ファイルに設定する場合、ODBC情報ファイルのセキュリティには十分注意してください。  
パスワードをODBC情報ファイルに設定しないで、アプリケーションの実行時にパスワードを入力するなど、アプリケーションにて解決する方法もあります。

### サーバ情報に定義される内容

サーバに関する情報の定義内容を下表に示します。

表22.1 サーバ情報の定義内容

| 情報名               | 設定内容      |            | 備考                                                                                                                                                                                                                                                                                                                                                |
|-------------------|-----------|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [サーバ名]            | サーバ名      |            | CONNECT 文またはデフォルトコネクション情報に指定したサーバ名と対応付けます。                                                                                                                                                                                                                                                                                                        |
| @SQL_DATASRC_KIND | データソース    | MACHINE_DS | データソースの種別を指定します。デフォルトはMACHINE_DSです。<br>文字列MACHINE_DSを指定した場合はマシンデータソースを使用します。マシンデータソースはユーザデータソースとシステムデータソースの総称です。<br>文字列FILE_DSを指定した場合はファイルデータソースを使用します。<br>(注1)                                                                                                                                                                                   |
|                   |           | FILE_DS    |                                                                                                                                                                                                                                                                                                                                                   |
| @SQL_DATASRC      | データソース名   |            | WindowsシステムのODBCデータソースアドミニストレータで指定(追加)したデータソース名を指定します。                                                                                                                                                                                                                                                                                            |
| @SQL_USERID       | ユーザID     |            | データソースを操作するためのユーザIDを指定します。                                                                                                                                                                                                                                                                                                                        |
| @SQL_PASSWORD     | パスワード     |            | データソースを操作するためのパスワードを指定します。<br>このパスワードは、ODBC情報設定ツールを使用して、暗号化して設定してください。                                                                                                                                                                                                                                                                            |
| @SQL_ACCESS_MODE  | アクセスモード   | READ_ONLY  | データソースに対するアクセスモードを指定します。デフォルトはREAD_WRITE です。<br>文字列READ_ONLYを指定した場合は読み専用となり、文字列READ_WRITE を指定した場合は読みおよび書き込みが可能となります。<br>ただし、ODBCドライバによって、指定に対する動作が異なる場合があります。                                                                                                                                                                                     |
|                   |           | READ_WRITE |                                                                                                                                                                                                                                                                                                                                                   |
| @SQL_COMMIT_MODE  | COMMITモード | MANUAL     | データソースに対するCOMMITモード(トランザクションの動作)を指定します。デフォルトはMANUALです。<br>文字列MANUALを指定した場合は、COBOL ソースプログラムにCOMMIT文またはROLLBACK文を記述することで、SQL の操作を確定します。<br>文字列AUTOを指定した場合は、COBOL ソースプログラムの記述に関係なく、SQL 文ごとにその操作が確定します。<br>注意) AUTOを指定した場合は、SQL 文を実行した時点でデータベースへその処理が反映されるため、ROLLBACK文でデータベースを元の状態に戻すことができません。MANUALの指定をおすすめします。ただし、ODBCドライバによって、指定に対する動作が異なる場合があります。 |
|                   |           | AUTO       |                                                                                                                                                                                                                                                                                                                                                   |
| @SQL_CONCURRENCY  | カーソルの同時実行 | READ_ONLY  | カーソルの同時実行を指定します。同時実行とは、複数のクライアント(複数のコネクションでも同様)が、同一データを同時に使用する機能を指します。                                                                                                                                                                                                                                                                            |

| 情報名                     | 設定内容          |               | 備考                                                                                                                                                                                                                                                          |
|-------------------------|---------------|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                         |               | LOCK          | @SQL_CONCURRENCY が指定されない場合、デフォルトは READ_ONLY です。(注2)                                                                                                                                                                                                         |
|                         |               | ROWVER        |                                                                                                                                                                                                                                                             |
|                         |               | VALUES        |                                                                                                                                                                                                                                                             |
| @SQL_CURSOR_TYPE        | カーソルの種類       | FORWARD_ONLY  | カーソルの種類を指定します。<br>@SQL_CURSOR_TYPE が指定されない場合、デフォルトは FORWARD_ONLY です。(注3)<br>カーソルの種類は、性能に大きく影響します。読み取り専用のカーソルの場合は、FORWARD_ONLY を指定すると性能が向上します。<br>FORWARD_ONLY 指定では FETCH PRIOR 文を実行することはできません。                                                              |
|                         |               | STATIC        |                                                                                                                                                                                                                                                             |
|                         |               | KEYSET_DRIVEN |                                                                                                                                                                                                                                                             |
|                         |               | DYNAMIC       |                                                                                                                                                                                                                                                             |
| @SQL_ODBC_CURSORS       | ODBCカーソルライブラリ | USE_DRIVER    | ODBCカーソルライブラリは、通常データソースが行うカーソル処理を代替する機能を有します。ODBCカーソルライブラリを使用することにより、データソースがUPDATE文(位置付け)、DELETE文(位置付け)をサポートしない場合でも、これらを実現することが可能になります。デフォルトは USE_DRIVER です。<br>文字列USE_DRIVERを指定した場合は、ODBCカーソルライブラリを使用しません。<br>文字列USE_ODBCを指定した場合は、ODBCカーソルライブラリを使用します。(注4) |
|                         |               | USE_ODBC      |                                                                                                                                                                                                                                                             |
| @SQL_QUERY_TIMEOUT (注1) | タイムアウト時間 (秒)  |               | データソースに対するクエリーのタイムアウト時間を指定します。指定範囲は、0 ～4294967285 です。<br>@SQL_QUERY_TIMEOUT が指定されない場合、デフォルトは0 で、タイムアウトはしません。ただし、ODBCドライバによって、指定に対する動作が異なる場合があります。<br>注意) ODBCドライバによって、指定に対する動作がエラーとなる場合があります。このような場合、0 を指定してください。                                           |

・注1

データソース種別にFILE\_DSを指定したとき、サーバ情報とファイルデータソースの両方にユーザID、パスワードが指定されている場合は、サーバ情報のユーザID、パスワードが有効になります。

・注2

指定値に対する動作は、“表22.2 @SQL\_CONCURRENCYの指定値に対する動作説明”を参照してください。ただし、ODBCドライバによって指定に対する動作が異なる場合があります。ご使用になる前には、“22.2.13.3 各ODBCドライバ固有の留意事項”を参照してください。

・注3

指定値に対する動作は、“表22.3 @SQL\_CURSOR\_TYPEの指定値に対する動作説明”を参照してください。ただし、ODBCドライバによって指定に対する動作が異なる場合があります。ご使用になる前には、“22.2.13.3 各ODBCドライバ固有の留意事項”を参照してください。

・注4

ODBCカーソルライブラリを使用する場合は、以下の注意事項があります。

- UPDATE文(位置付け)またはDELETE文(位置付け)を使用する場合は、カーソル宣言で値が一意な列を必ず1つ以上選択してください。

ODBCカーソルライブラリは、UPDATE文(位置付け)およびDELETE文(位置付け)をそれぞれUPDATE文(探索)およびDELETE文(探索)にシミュレートして実行します。そのため、一意な値を持つ列が選択されていない場合、複数行に処理が影響する場合があります。

- ODBCカーソルライブラリを使用しない場合と比較して、実行性能が劣化する場合があります。
- カーソルの同時実行(@SQL\_CONCURRENCY)に“VALUES”を、カーソルの種類(@SQL\_CURSOR\_TYPE)に“STATIC”を指定しなければなりません。

表22.2 @SQL\_CONCURRENCYの指定値に対する動作説明

| 指定値       | 動作説明                                                                            |                                                                                                                                                                       |
|-----------|---------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| READ_ONLY | カーソル系データ操作文の位置付け・取出し(FETCH 文)が可能です。更新(位置付けUPDATE文)、削除(位置付けDELETE文)は実行できません。(注1) |                                                                                                                                                                       |
| LOCK      | カーソル系データ操作文の位置付け・取出し(FETCH 文)、更新(位置付けUPDATE文)、削除(位置付けDELETE文)が可能です。(注2)         | カーソルは最下位レベルのロックを使用して、更新(位置付けUPDATE文)、削除(位置付けDELETE文)を行います。同一リソースを更新または削除することを防ぎます。                                                                                    |
| ROWVER    |                                                                                 | カーソルは行の変更履歴を使用したオプティミスティック同時実行制御を使用して、更新(位置付けUPDATE文)、削除(位置付けDELETE文)を行います。オプティミスティック同時実行では、行が更新または削除されるまでロックされません。したがって、他のユーザが同時に同一リソースを更新または削除した場合、処理が失敗する可能性があります。 |
| VALUES    |                                                                                 | カーソルは行の値を使用したオプティミスティック同時実行制御を使用して、更新(位置付けUPDATE文)、削除(位置付けDELETE文)を行います。                                                                                              |

・ 注1

一部のデータソースでは、カーソル系データ操作文の更新(位置付けUPDATE文)、削除(位置付けDELETE文)が実行可能です。

・ 注2

カーソルのロックレベルはデータソースに依存します。



注意

ODBCドライバによって、指定に対する動作がエラーとなる場合があります。このような場合、READ\_ONLYを指定してください。

表22.3 @SQL\_CURSOR\_TYPEの指定値に対する動作説明

| 指定値           | 動作説明                        |
|---------------|-----------------------------|
| FORWARD_ONLY  | カーソルを順方向専用カーソルでオープンします。     |
| STATIC        | カーソルを静的カーソルでオープンします。        |
| KEYSET_DRIVEN | カーソルをキーセットドリブンカーソルでオープンします。 |
| DYNAMIC       | カーソルを動的カーソルでオープンします。        |



注意

カーソルの種類は、データソース(ODBCドライバ、データベース、データベース関連製品)に依存します。使用するデータベースに該当するカーソルタイプがあるか確認してください。カーソルの種類は、@SQL\_CONCURRENCYの値に影響します。関連するドライバを参照してください。

## デフォルトコネクション情報に定義される内容

CONNECT文にDEFAULT指定を記述した場合には、ここで説明する情報を基にコネクションが確立されます。

デフォルトコネクション情報の定義内容を“表22.4 デフォルトコネクション情報の定義内容”に示します。

表22.4 デフォルトコネクション情報の定義内容

| 情報名               | 設定内容  | 備考                                        |
|-------------------|-------|-------------------------------------------|
| [SQL_DEFAULT_INF] | 固定文字列 | デフォルトコネクション情報の定義開始を示す固定文字列(セクション名)を指定します。 |

| 情報名           | 設定内容  | 備考                                                                                                                     |
|---------------|-------|------------------------------------------------------------------------------------------------------------------------|
| @SQL_SERVER   | サーバ名  | デフォルトコネクションを確立する対象のサーバ名を指定します。このサーバ名をもとにしてサーバごとの定義情報を検索し、サーバごとのデータソース名に対してコネクションを確立します。したがって、サーバごとの定義情報を記述しておく必要があります。 |
| @SQL_USERID   | ユーザID | デフォルトコネクションサーバに対応するデータソースを操作するためのユーザIDを指定します。                                                                          |
| @SQL_PASSWORD | パスワード | デフォルトコネクションサーバに対応するデータソースを操作するためのパスワードを指定します。このパスワードは、ODBC情報設定ツールを使用して、暗号化して設定してください。                                  |

## コネクション有効範囲に定義される内容

コネクションの有効範囲に関する情報の定義内容を“[表22.5 コネクション有効範囲の定義内容](#)”に示します。

表22.5 コネクション有効範囲の定義内容

表22.3 コネクション有効範囲の定義内容

| 情報名                   | 設定内容       |                 | 備考                                                                          |
|-----------------------|------------|-----------------|-----------------------------------------------------------------------------|
| [CONNECTION_SCOPE]    | 固定文字列      |                 | コネクション有効範囲の定義開始を示す固定文字列(セクション名)を指定します。                                      |
| @SQL_CONNECTION_SCOPE | コネクション有効範囲 | PROCESS         | コネクションの有効範囲を指定します。デフォルトはPROCESSです。<br>コネクションの有効範囲とは、接続したコネクションの利用可能範囲です。(注) |
|                       |            | THREAD          |                                                                             |
|                       |            | OBJECT_INSTANCE |                                                                             |

注: 指定値に対する動作は、“[表22.6 コネクション有効範囲の指定値に対する動作説明](#)”を参照してください。

また、有効な利用方法は、“[24.5.2 リモートデータベースアクセス\(ODBC\)の利用](#)”および“[22.2.6.3 コネクションをオブジェクトインスタンス単位で利用する](#)”を参照してください。

表22.6 コネクション有効範囲の指定値に対する動作説明

| 指定値             | 動作説明                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PROCESS         | <p>接続したコネクションは、実行環境内(プロセス内)で利用可能です。</p> <p>通常、シングルスレッドプログラムを動作させる場合に指定します。</p> <p>マルチスレッドプログラムを動作させる場合の注意事項</p> <ul style="list-style-type: none"> <li>複数のマルチスレッドプログラムが1つのコネクションを共有して利用する場合、いずれかのマルチスレッドプログラムでトランザクション処理を行ったとき、コネクションを共有するすべてのマルチスレッドプログラムのデータ操作に影響します。</li> <li>複数のマルチスレッドプログラムが複数のコネクションを利用する場合、各マルチスレッドプログラムで最初に実行される埋込みSQL文はCONNECT文またはSET CONNECTION文でなければなりません。CONNECT文またはSET CONNECTION文以外の埋込みSQL文を実行した場合は、どのコネクションに対して操作をするか保証できません。これは、各マルチスレッドプログラムがそれぞれの現コネクションを利用するためです。なお、複数のマルチスレッドプログラムが1つのコネクションを共有して利用する場合は必要ありません。</li> <li>カーソルを使用したマルチスレッドプログラムが複数のスレッドで実行された場合、それぞれのスレッドが自分自身のカーソルを持ちます。したがって、カーソルをスレッド間で共有することはできません。</li> </ul> |
| THREAD          | <p>接続したコネクションは、実行単位内(スレッド内)で利用可能です。</p> <p>通常、シングルスレッドプログラムをマルチスレッドプログラムに移行する場合に指定します。</p> <p>実行単位内で複数のコネクションを接続した場合、最後に実行されたCONNECT文またはSET CONNECTION文で指定されたコネクションが、実行単位の現コネクションになります。(注1)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| OBJECT_INSTANCE | 接続したコネクションは、オブジェクトインスタンス内で利用可能です。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

| 指定値 | 動作説明                                                                                                                                                                     |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | 通常、オブジェクト指向機能を利用したシングルスレッドプログラムをマルチスレッドプログラムに移行する場合に指定します。オブジェクトインスタンス内で複数のコネクションを接続した場合、最後に実行されたCONNECT 文またはSET CONNECTION文で指定されたコネクションが、オブジェクトインスタンスの現コネクションになります。(注2) |

注1:クラス定義(オブジェクト指向プログラミング機能)に記述した埋込みSQL文は動作しません。

注2:プログラム定義に記述した埋込みSQL文は動作しません。また、V40以前のコンパイラで翻訳されたプログラムは動作しません。

## 22.2.8.2 ODBC情報設定ツールの使い方

ODBC情報設定ツールは、NetCOBOLのODBC経由のリモートデータベースアクセスに必要な情報を指定したファイル(このファイルをODBC情報ファイルと呼んでいます)に設定するためのツールです。

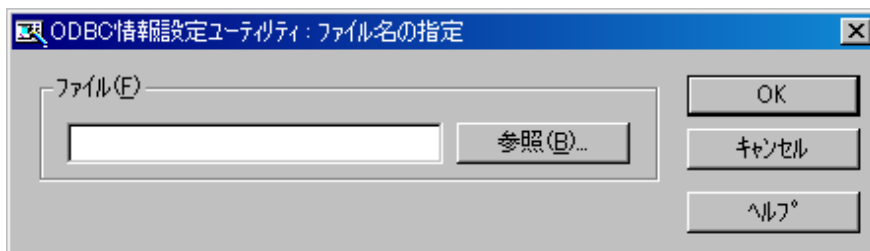
ODBC情報設定ツールには、次のような機能があります。

- ODBC情報ファイルの選択
- サーバ情報の設定
- デフォルトコネクション情報の設定
- コネクション有効範囲の設定

以下にODBC情報設定ツールの使い方を示します。詳細は、ヘルプを参照してください。

### 1. ODBC情報設定ツールの起動

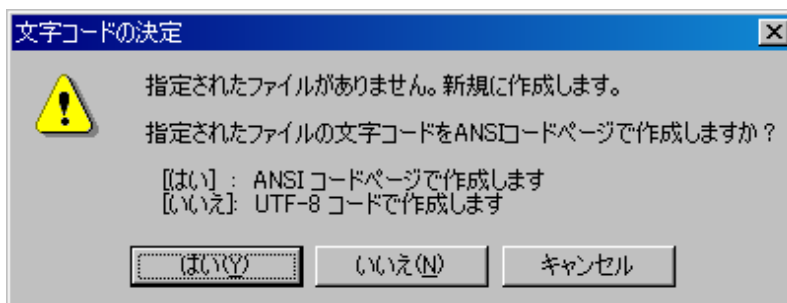
ODBC情報設定ツール(SQLODBCS.EXE)を起動します。



### 2. ODBC情報ファイルの選択

情報を設定するODBC情報ファイルを指定します。

指定できるファイルの文字コードは、ANSIコードページ(シフトJIS)とBOM付きUTF-8です。ファイルを新規に作成する際は、以下のダイアログにより、文字コードを選択してください。「はい」を指定するとシフトJISでファイルを作成します。「いいえ」を指定するとBOM付きUTF-8でファイルを作成します。



既存ファイルのオープンでは、シフトJISまたは、BOM付きUTF-8を自動的に判定してファイルをオープンします。BOM付きUTF-8のファイルをオープンした時は、ODBC情報設定ツールのタイトルに(UTF-8)の文字が表示されます。

### 3. サーバ情報の設定

CONNECT文またはデフォルトコネクション情報で、指定されたサーバ名に対する情報を設定します。

サーバ名を指定して、サーバ名毎の情報を設定します。サーバ名に指定した値は、[サーバ名]のセクションとして、ODBC情報ファイルに追加されます。

サーバ情報タグでは、サーバ名に指定した[サーバ名]セクション毎に以下の情報を設定します。

| ODBC情報設定ユーティリティのサーバ情報 |                              | 情報名               |
|-----------------------|------------------------------|-------------------|
| データソース                | マシンデータソース/ファイルデータソースオプションボタン | @SQL_DATASRC_KIND |
|                       | データソース名                      | @SQL_DATASRC      |
| ユーザID                 |                              | @SQL_USERID       |
| パスワード                 |                              | @SQL_PASSWORD     |
| アクセスモード               |                              | @SQL_ACCESS_MODE  |
| COMMITモード             |                              | @SQL_COMMIT_MODE  |

[拡張オプション]ボタンで拡張オプションのダイアログが表示されます。

拡張オプションダイアログでは、サーバ名に指定した[サーバ名]セクション毎に以下の情報を設定します。

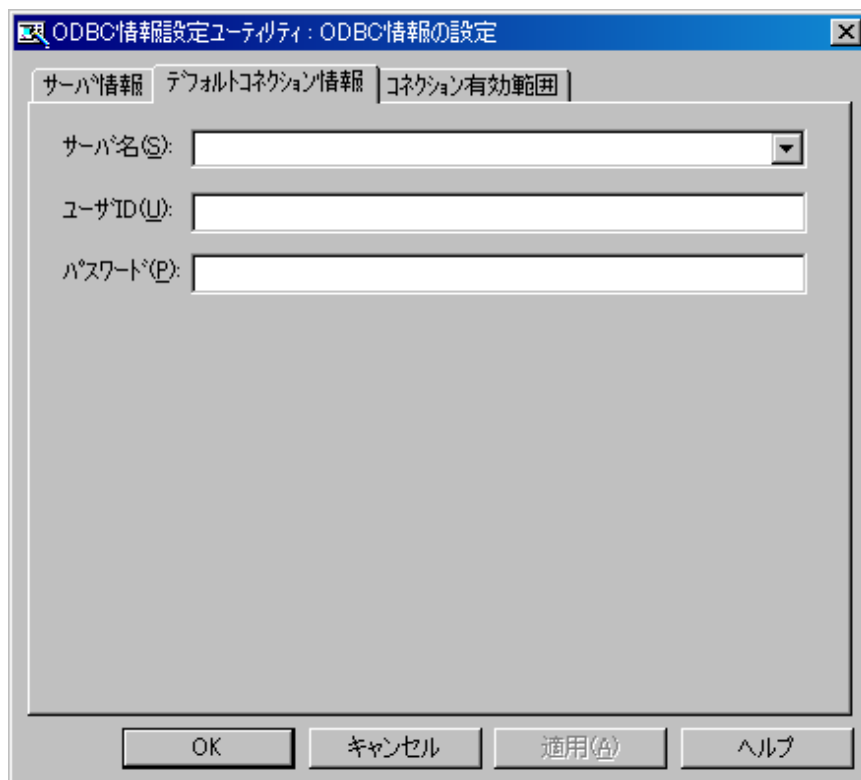


| ODBC情報設定ユーティリティの拡張オプションダイアログ | 情報名                |
|------------------------------|--------------------|
| カーソル同時実行                     | @SQL_CONCURRENCY   |
| カーソルの種類                      | @SQL_CURSOR_TYPE   |
| ODBCカーソルライブラリ                | @SQL_ODBC_CURSORS  |
| クエリータイムアウト時間                 | @SQL_QUERY_TIMEOUT |

詳細は、“[22.2.8.1.2 ODBC情報ファイルの作成](#)”の“サーバ情報に定義される内容”を参照してください。

#### 4. デフォルトコネクション情報の設定

DEFAULT指定のCONNECT文が記述された場合に、使用するデフォルトコネクション情報(サーバ名、ユーザIDおよびパスワード)を設定します。



デフォルトコネクション情報タグでは、[SQL\_DEFAULT\_INFO]セクションに以下の情報を設定します。

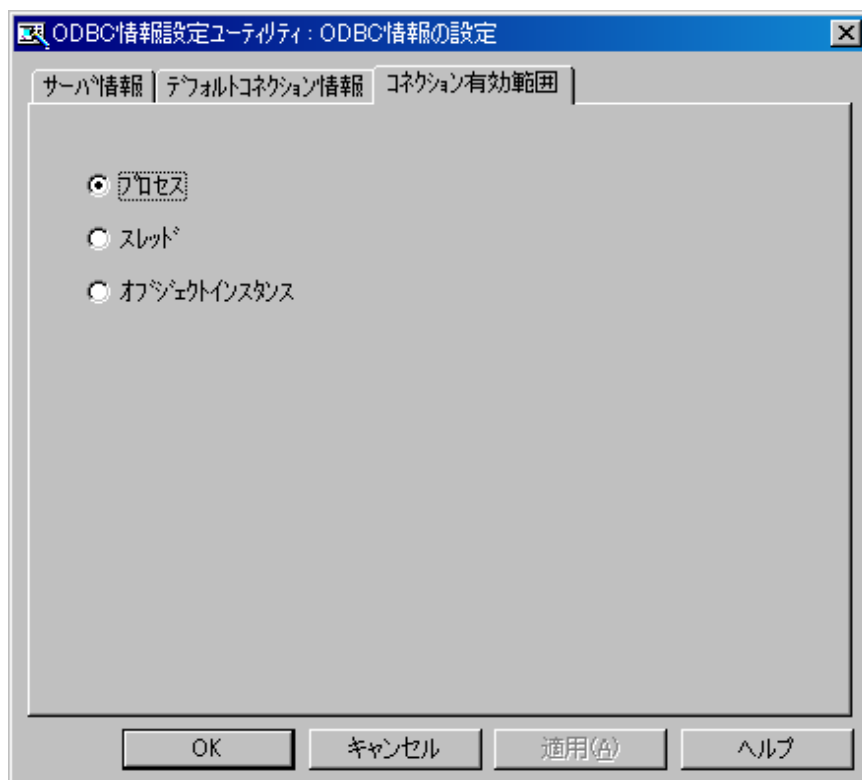
| ODBC情報設定ユーティリティのデフォルトコネクション情報 | 情報名           |
|-------------------------------|---------------|
| サーバ名                          | @SQL_SERVER   |
| ユーザID                         | @SQL_USERID   |
| パスワード                         | @SQL_PASSWORD |

詳細は、“[22.2.8.1.2 ODBC情報ファイルの作成](#)”の“デフォルトコネクション情報に定義される内容”を参照してください。



## 5. コネクション有効範囲の設定

コネクションの有効範囲を設定します。



コネクション有効範囲のタグでは、[CONNECTION\_SCOPE]セクションに以下の情報を設定します。

| ODBC情報設定ユーティリティのコネクション有効範囲      | 情報名                   |
|---------------------------------|-----------------------|
| プロセス/スレッド/オブジェクトインスタンスのオプションボタン | @SQL_CONNECTION_SCOPE |

詳細は、“[22.2.8.1.2 ODBC情報ファイルの作成](#)”の“コネクション有効範囲に定義される内容”を参照してください。

## 22.2.8.3 ODBC情報ファイルの設定内容の最大長

ODBC情報ファイルに設定する内容の最大長を以下に示します。

| 情報の種別   | 最大長   | 備考                                                                 |
|---------|-------|--------------------------------------------------------------------|
| ユーザID   | 32バイト | コネクションを確立するデータソースの仕様により異なります。したがって、ODBCドライバと関係する環境のマニュアルを参照してください。 |
| パスワード   | 32バイト |                                                                    |
| サーバ名    | 32バイト | -                                                                  |
| データソース名 | 32バイト | -                                                                  |



パスワードは、ODBC情報設定ツールを使用して、暗号化して設定してください。エディタでは、直接編集しないでください。

## 22.2.8.4 連携ソフトウェアおよびハードウェア環境の整備

COBOLアプリケーションから、ODBCを使用してサーバのデータベースへアクセスするためには、連携するソフトウェアおよびハードウェアの環境を整備する必要があります。

以下にその手順について簡単に説明します。

## ODBC環境のセットアップ

### WindowsシステムへのODBCの初期導入

WindowsシステムへODBCの環境をはじめて導入する場合は、ODBCドライバと同時に提供されるODBCセットアップを使用してください。

この導入作業により、WindowsのコントロールパネルにODBCが追加されます。

### ODBCドライバの導入とデータソースの作成

ODBCデータソースアドミニストレータ(通常、Windowsのコントロールパネルのグループに存在しています)を起動して、セットアップを行ってください。このセットアップでは、ODBCドライバの導入操作およびデータソースの定義を行います。



リモートデータベースアクセス(ODBC)機能を利用したCOBOLプログラムが、サービスから呼び出される場合は、データソースをシステムデータソースとして定義する必要があります。システムデータソースは、ユーザではなくコンピュータに対して定義するデータソースです。システムデータソースは、サービスを含むコンピュータ上のすべてのユーザが認識することができます。システムデータソースについては、ODBCデータソースアドミニストレータのオンラインヘルプを参照してください。

[参照]“[23.1.3 サービス配下で動作するプログラム](#)”

### ODBCドライバに関する環境の整備

ODBCドライバは、そのドライバが動作するために必要な環境をあらかじめ想定しています。ODBCドライバを動作させるために必要な環境については、各ODBCドライバのヘルプおよびマニュアルを参照してください。ODBCドライバのヘルプは、ODBCデータソースアドミニストレータを起動することにより参照できます。

### データソースとの接続の確認

通常、データベースは、クライアントからサーバのデータベースを操作するためのプログラムを提供しています。ODBCを使用したCOBOLアプリケーションを実行する前に、これらのプログラムを利用して、クライアントとサーバの間で接続が成功しているかどうかを確認しておくことが安全です。

ODBCデータソースアドミニストレータ、実行環境情報、ODBC情報ファイルなどの設定が完了した後にCOBOLアプリケーションを実行してください。

## 22.2.9 埋込みSQL文のキーワード一覧

埋込みSQL文のキーワード一覧を以下に示します。

|     |               |
|-----|---------------|
| 【A】 | ABSOLUTE      |
|     | ADA           |
|     | ADD           |
|     | ALL           |
|     | ALLOCATE      |
|     | ALTER         |
|     | AND           |
|     | ANY           |
|     | ARE           |
|     | AS            |
|     | ASC           |
|     | ASSERTION     |
|     | AT            |
|     | AUTHORIZATION |
|     |               |

|  |     |
|--|-----|
|  | AVG |
|--|-----|

|     |            |
|-----|------------|
| 【B】 | BEGIN      |
|     | BETWEEN    |
|     | BIND       |
|     | BIT        |
|     | BIT_LENGTH |
|     | BY         |

|     |                       |
|-----|-----------------------|
| 【C】 | CALL                  |
|     | CASCADE               |
|     | CASCADEED             |
|     | CASE                  |
|     | CAST                  |
|     | CATALOG               |
|     | CHAR                  |
|     | CHAR_LENGTH           |
|     | CHARACTER             |
|     | CHARACTER_LENGTH      |
|     | CHARACTER_SET_CATALOG |
|     | CHARACTER_SET_NAME    |
|     | CHARACTER_SET_SCHEMA  |
|     | CHECK                 |
|     | CLOSE                 |
|     | COALESCE              |
|     | COBOL                 |
|     | COLLATE               |
|     | COLLATION             |
|     | COLLATION_CATALOG     |
|     | COLLATION_NAME        |
|     | COLLATION_SCHEMA      |
|     | COLUMN                |
|     | COMMIT                |
|     | CONNECT               |
|     | CONNECTION            |
|     | CONSTRAINT            |
|     | CONSTRAINTS           |
|     | CONTINUE              |
|     | CONVERT               |
|     | CORRESPONDING         |
|     | COUNT                 |

|  |                   |
|--|-------------------|
|  | CREATE            |
|  | CURRENT           |
|  | CURRENT_DATE      |
|  | CURRENT_TIME      |
|  | CURRENT_TIMESTAMP |
|  | CURSOR            |

|     |                             |
|-----|-----------------------------|
| 【D】 | DATA                        |
|     | DATE                        |
|     | DATETIME_INTERVAL_CODE      |
|     | DATETIME_INTERVAL_PRECISION |
|     | DAY                         |
|     | DEALLOCATE                  |
|     | DEC                         |
|     | DECIMAL                     |
|     | DECLARE                     |
|     | DEFAULT                     |
|     | DEFERRABLE                  |
|     | DEFERRED                    |
|     | DELETE                      |
|     | DESC                        |
|     | DESCRIBE                    |
|     | DESCRIPTOR                  |
|     | DIAGNOSTICS                 |
|     | DICTIONARY                  |
|     | DISCONNECT                  |
|     | DISPLACEMENT                |
|     | DISTINCT                    |
|     | DOMAIN                      |
|     | DOUBLE                      |
|     | DROP                        |

|     |           |
|-----|-----------|
| 【E】 | ELSE      |
|     | END       |
|     | END-EXEC  |
|     | ESCAPE    |
|     | EXCEPTION |
|     | EXEC      |
|     | EXECUTE   |
|     | EXISTS    |
|     | EXTERNAL  |

|  |         |
|--|---------|
|  | EXTRACT |
|--|---------|

|     |         |
|-----|---------|
| 【F】 | FALSE   |
|     | FETCH   |
|     | FIRST   |
|     | FLOAT   |
|     | FOR     |
|     | FOREIGN |
|     | FORTRAN |
|     | FOUND   |
|     | FROM    |
|     | FULL    |

|     |        |
|-----|--------|
| 【G】 | GET    |
|     | GLOBAL |
|     | GO     |
|     | GOTO   |
|     | GRANT  |
|     | GROUP  |

|     |        |
|-----|--------|
| 【H】 | HAVING |
|     | HOUR   |

|     |             |
|-----|-------------|
| 【I】 | IDENTITY    |
|     | IGNORE      |
|     | IMMEDIATE   |
|     | IN          |
|     | INCLUDE     |
|     | INDEX       |
|     | INDICATOR   |
|     | INITIALLY   |
|     | INNER       |
|     | INPUT       |
|     | INSENSITIVE |
|     | INSERT      |
|     | INTEGER     |
|     | INTERSECT   |
|     | INTERVAL    |
|     | INTO        |
|     | IS          |
|     | ISOLATION   |

|     |      |
|-----|------|
| 【J】 | JOIN |
|-----|------|

|     |     |
|-----|-----|
| 【K】 | KEY |
|-----|-----|

|     |          |
|-----|----------|
| 【L】 | LANGUAGE |
|     | LAST     |
|     | LEFT     |
|     | LENGTH   |
|     | LEVEL    |
|     | LIKE     |
|     | LIST     |
|     | LOCAL    |
|     | LOWER    |

|     |        |
|-----|--------|
| 【M】 | MATCH  |
|     | MAX    |
|     | MIN    |
|     | MINUTE |
|     | MODULE |
|     | MONTH  |
|     | MUMPS  |

|     |          |
|-----|----------|
| 【N】 | NAME     |
|     | NAMES    |
|     | NATIONAL |
|     | NCHAR    |
|     | NEXT     |
|     | NONE     |
|     | NOT      |
|     | NULL     |
|     | NULLABLE |
|     | NULLIF   |
|     | NUMERIC  |

|     |              |
|-----|--------------|
| 【O】 | OCTET_LENGTH |
|     | OF           |
|     | OFF          |
|     | ON           |
|     | ONLY         |
|     | OPEN         |
|     | OPTION       |

|  |          |
|--|----------|
|  | OR       |
|  | ORDER    |
|  | OUTER    |
|  | OUTPUT   |
|  | OVERLAPS |

|     |            |
|-----|------------|
| 【P】 | PARTIAL    |
|     | PASCAL     |
|     | PLI        |
|     | POSITION   |
|     | PRECISION  |
|     | PREPARE    |
|     | PRESERVE   |
|     | PREVIOUS   |
|     | PRIMARY    |
|     | PRIOR      |
|     | PRIVILEGES |
|     | PROCEDURE  |
|     | PUBLIC     |

|     |          |
|-----|----------|
| 【R】 | RELATIVE |
|     | RESTRICT |
|     | REVOKE   |
|     | RIGHT    |
|     | ROLLBACK |
|     | ROWS     |

|     |          |
|-----|----------|
| 【S】 | SCALE    |
|     | SCHEMA   |
|     | SCROLL   |
|     | SECOND   |
|     | SECTION  |
|     | SELECT   |
|     | SEQUENCE |
|     | SET      |
|     | SIZE     |
|     | SMALLINT |
|     | SOME     |
|     | SQL      |
|     | SQLCA    |
|     | SQLCODE  |

|  |            |
|--|------------|
|  | SQLERARY   |
|  | SQLERRD    |
|  | SQLERROR   |
|  | SQLSTATE   |
|  | SQLWARNING |
|  | START      |
|  | SUBSTRING  |
|  | SUM        |
|  | SYSTEM     |

|     |                 |
|-----|-----------------|
| 【T】 | TABLE           |
|     | TEMPORARY       |
|     | THEN            |
|     | TIME            |
|     | TIMESTAMP       |
|     | TIMEZONE_HOUR   |
|     | TIMEZONE_MINUTE |
|     | TO              |
|     | TRANSACTION     |
|     | TRANSLATE       |
|     | TRANSLATION     |
|     | TRUE            |
|     | TYPE            |

|     |         |
|-----|---------|
| 【U】 | UNION   |
|     | UNIQUE  |
|     | UNKNOWN |
|     | UPDATE  |
|     | UPPER   |
|     | USAGE   |
|     | USER    |
|     | USING   |

|     |           |
|-----|-----------|
| 【V】 | VALUE     |
|     | VALUES    |
|     | VARCHAR   |
|     | VARIABLES |
|     | VARYING   |
|     | VIEW      |

|     |      |
|-----|------|
| 【W】 | WHEN |
|-----|------|



|  |          |
|--|----------|
|  | WHENEVER |
|  | WHERE    |
|  | WITH     |
|  | WORK     |

|     |      |
|-----|------|
| 【Y】 | YEAR |
|-----|------|

## 22.2.10 SQL文と指定可能なホスト変数

それぞれの埋込みSQL文に指定できるホスト変数の形式を、以下に示します。

表22.7 埋込みSQL文に対して指定できるホスト変数の形式

| 埋込みSQL文      |                    | 1行ずつ操作     |            | 複数行を同時に操作  |          |
|--------------|--------------------|------------|------------|------------|----------|
|              |                    | 単一列指定ホスト変数 | 複数列指定ホスト変数 | 複数行指定ホスト変数 | 表指定ホスト変数 |
| 非カーソル系データ操作文 | SELECT文            | ○          | ○          | ○          | ○        |
|              | DELETE文(探索)        | ○          | ×          | ○          | ×        |
|              | INSERT文            | ○          | ○          | ○          | ○        |
|              | UPDATE文(探索)        | ○          | ×          | ○          | ×        |
| カーソル系データ操作文  | OPEN文              | ○          | ×          | ×          | ×        |
|              | CLOSE文             | ×          | ×          | ×          | ×        |
|              | FETCH文             | ○          | ○          | ○          | ○        |
|              | DELETE文(位置付け)      | ×          | ×          | ×          | ×        |
|              | UPDATE文(位置付け)      | ×          | ×          | ×          | ×        |
| 動的SQL文       | PREPARE文           | ○          | ×          | ×          | ×        |
|              | EXECUTE文           | ○          | ○          | ○          | ○        |
|              | EXECUTE IMMEDIATE文 | ○          | ×          | ×          | ×        |
|              | 動的SELECT文          | ×          | ×          | ×          | ×        |
|              | 動的カーソル宣言           | ×          | ×          | ×          | ×        |
|              | 動的OPEN文            | ○          | ×          | ×          | ×        |
|              | 動的CLOSE文           | ×          | ×          | ×          | ×        |
|              | 動的FETCH文           | ○          | ○          | ○          | ○        |
|              | 動的DELETE文(位置付け)    | ×          | ×          | ×          | ×        |
|              | 動的UPDATE文(位置付け)    | ×          | ×          | ×          | ×        |
|              |                    |            |            |            |          |
| セッション制御文     | COMMIT文            | ×          | ×          | ×          | ×        |
|              | ROLLBACK文          | ×          | ×          | ×          | ×        |
| コネクション制御文    | CONNECT文           | ○          | ×          | ×          | ×        |
|              | SET CONNECTION文    | ○          | ×          | ×          | ×        |
|              | DISCONNECT文        | ○          | ×          | ×          | ×        |
| その他          | CALL文              | ○          | ×          | ×          | ×        |
|              | FOR句               | ○          | ×          | ×          | ×        |

○:指定可  
×:指定不可

## 22.2.11 ODBCで扱うデータとの対応

COBOLでは、ODBCで扱うデータを対応付けて扱います。

ODBCドライバがODBCのSQLデータ型をどのように扱うかについては、使用するODBCドライバのヘルプおよびマニュアルを参照してください。



### 注意

対応表に記述されていない対応付けを行った場合、データの内容は保証されません。

表22.8 算術データの対応表

| ODBC SQLデータ型 |                         | COBOL での表現                                                                                                                                                                                                                                                                   |
|--------------|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2進           | SQL_SMALLINT (SMALLINT) | PIC S9(4) BINARY<br><br>または<br><br>PIC S9(4) COMP-5                                                                                                                                                                                                                          |
|              | SQL_INTEGER (INTEGER)   | PIC S9(9) BINARY<br><br>または<br><br>PIC S9(9) COMP-5                                                                                                                                                                                                                          |
| 10進          | SQL_DECIMAL (DECIMAL)   | PIC S9(p) PACKED-DECIMAL<br><br>または<br><br>PIC S9(p)V9(q) PACKED-DECIMAL<br><br>$1 \leq p \leq 15$<br>$1 \leq q$<br>$p+q \leq 18$                                                                                                                                            |
|              | SQL_NUMERIC (NUMERIC)   | $\left\{ \begin{array}{l} \text{PIC S9(p)} \\ \text{PIC S9(p)V9(q)} \end{array} \right\}$<br>$[\text{SIGN IS } \left\{ \begin{array}{l} \text{LEADING SEPARATE CHARACTER} \\ \text{TRAILING} \end{array} \right\}]$<br><br>$1 \leq p \leq 15$<br>$1 \leq q$<br>$p+q \leq 18$ |
| 内部浮動小数点      | SQL_REAL (REAL)         | COMP-1                                                                                                                                                                                                                                                                       |
|              | SQL_DOUBLE (FLOAT)      | COMP-2                                                                                                                                                                                                                                                                       |

翻訳オプションRCS(SJIS)を指定した場合、または、翻訳オプションを指定しなかった場合の文字データの対応表は以下のようになります。

表22.9 文字データとの対応表(RCS(SJIS)翻訳オプションを指定した場合)

| ODBC SQLデータ型 |                       | COBOL での表現                                                                                                                                                                                         |
|--------------|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 固定長          | SQL_CHAR (CHAR)       | PIC X(n) (注1)                                                                                                                                                                                      |
|              |                       | PIC N(n) (注1)                                                                                                                                                                                      |
| 可変長          | SQL_VARCHAR (VARCHAR) | 01 データ名－1.<br>49 データ名－2 PIC S9(m) BINARY. (注2)<br>49 データ名－3 PIC X(n). (注1)<br><br>または<br><br>01 データ名－1.<br>49 データ名－2 PIC S9(m) COMP-5. (注2)<br>49 データ名－3 PIC X(n). (注1)<br><br>m=4 または 9           |
|              |                       | 01 データ名－1. (注1)<br>49 データ名－2 PIC S9(m) BINARY. (注2)<br>49 データ名－3 PIC N(n). (注1)<br><br>または<br><br>01 データ名－1. (注1)<br>49 データ名－2 PIC S9(m) COMP-5. (注2)<br>49 データ名－3 PIC N(n). (注1)<br><br>m=4 または 9 |
|              |                       |                                                                                                                                                                                                    |
|              |                       |                                                                                                                                                                                                    |

注1: 文字数nは、データベースドライバがサポートしているODBCドライバマネージャのバージョン、データベースドライバの仕様によって制限されます。例えば、ODBC2.0を使用している場合は、以下のような制限があります。

X(n) 1≦ n ≦ 254  
N(n) 1≦ n ≦ 127

注2: 文字データ可変長の長さ部は文字数を指定します。

翻訳オプションRCS(UTF16)を指定した場合、文字データの対応表は以下のようになります。

表22.10 文字データとの対応表(RCS(UTF16)翻訳オプションを指定した場合)

| ODBC SQLデータ型 |                         | COBOL での表現                                                                                                                                                                               |
|--------------|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 固定長          | SQL_CHAR (CHAR)         | PIC X(n) (注1)                                                                                                                                                                            |
|              | SQL_WCHAR (WCHAR)       | PIC N(n) (注1) (注3)                                                                                                                                                                       |
| 可変長          | SQL_VARCHAR (VARCHAR)   | 01 データ名－1.<br>49 データ名－2 PIC S9(m) BINARY. (注2)<br>49 データ名－3 PIC X(n). (注1)<br><br>または<br><br>01 データ名－1.<br>49 データ名－2 PIC S9(m) COMP-5. (注2)<br>49 データ名－3 PIC X(n). (注1)<br><br>m=4 または 9 |
|              | SQL_WVARCHAR (WVARCHAR) | 01 データ名－1. (注3)<br>49 データ名－2 PIC S9(m) BINARY. (注2)<br>49 データ名－3 PIC N(n). (注1)                                                                                                          |

| ODBC SQLデータ型 |  | COBOL での表現                                                                                                                 |
|--------------|--|----------------------------------------------------------------------------------------------------------------------------|
|              |  | <p>または</p> <p>01 データ名-1. (注3)</p> <p>49 データ名-2 PIC S9(m) COMP-5. (注2)</p> <p>49 データ名-3 PIC N(n). (注1)</p> <p>m=4 または 9</p> |

注1: 文字数nは、データベースドライバがサポートしているODBCドライバマネージャのバージョン、データベースドライバの仕様によって制限されます。例えば、ODBC2.0を使用している場合は、以下のような制限があります。

X(n) 1≦ n ≦ 254  
N(n) 1≦ n ≦ 127

注2: 文字データ可変長の長さ部は文字数を指定します。

注3: Unicodeをサポートしているデータベースおよびその関連製品に依存します。

表22.11 日付データの対応表

| ODBC SQLデータ型 |               | COBOL での表現   |
|--------------|---------------|--------------|
| 日付データ型       | SQL_DATE      | PIC X(n) (注) |
|              | SQL_TIMESTAMP |              |

注: 文字数nは、データベースドライバがサポートしているODBCドライバマネージャのバージョン、データベースドライバの使用により制限されます。

## 22.2.12 SQLSTATE/SQLCODE/SQLMSG

ここでは、ODBCを使用してSQL文を実行した場合に、通知領域へ通知されるCOBOL、ODBCドライバマネージャ、ODBCドライバおよびDBMSから通知される情報について説明します。

以下に、SQLSTATE、SQLCODEおよびSQLMSGに通知される情報を示します。

| 通知情報     | 通知される値         | 通知の意味                                 | プログラムの処理                                                                                                                               |
|----------|----------------|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| SQLSTATE | 00000          | 正常終了                                  | SQLCODEに付加情報が通知されることがあります。<br>(注)                                                                                                      |
|          | 表示可能な英数字5桁の組合せ | SQL文の実行時に、何らかのエラーまたは警告などの付加情報が発生しました。 | SQLCODE、SQLMSGの内容を参照し、エラーの場合は処置を行ってください。<br>SQLSTATEの詳細情報については、Microsoft社のODBC SDKの資料または使用しているODBCの環境(ODBCドライバ、DBMSなど)のマニュアルを参照してください。 |
| SQLCODE  | 0              | 正常終了                                  | —                                                                                                                                      |
|          | 0以外の正または負の整数値  | SQL文の実行時に、何らかのエラーまたは警告などの付加情報が発生しました。 | 使用しているODBCの環境のマニュアルから、原因を調査して対処してください。                                                                                                 |
| SQLMSG   | 空白(出力なし)       | 正常終了                                  | —                                                                                                                                      |
|          | メッセージ文字列       | SQL文実行時エラー、または警告など付加情報の内容が通知されます。     | 設定されたメッセージ文字列から原因を調査し、対処してください。                                                                                                        |

注: SQLSTATEの値が00000、かつ、SQLCODEの値が0の場合が正常終了です。

SQLSTATEの値が00000でも、SQLCODEに付加情報が通知されている場合は、データの内容は保障されません。

次に、埋込みSQL文の実行時にCOBOLが検出するエラーについて説明します。

以下は、SQLSTATE、SQLCODE、SQLMSGに通知される情報です。

| SQLSTATE | SQLCODE    | SQLMSG                  | プログラムの処理                                                                                                                    |
|----------|------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| 99999    | -999999999 | 同じコネクション名で接続しました。       | 同じコネクション名を持つ複数のコネクションの指定は許されません。それぞれコネクション名が一意になるように修正してください。                                                               |
| 9999A    | -999999990 | コネクションの最大設定数を超えました。     | COBOL システムで許されるコネクションの最大数を超えています。コネクション数を減らす対処を行ってください。<br>ただし、コネクションの最大数は、ODBCの環境により異なることがあります。ODBCの環境のマニュアルを参照して対処してください。 |
| 9999B    | -999999800 | 指定したコネクションは存在しません。      | コネクションがアクティブになっていないため、SQL 文が実行できない状態にあります。<br>プログラムのSQL 文の順序を調査し対処してください。                                                   |
| 9999D    | -999999200 | FOR句に指定した繰り返し回数が大きすぎます。 | FOR句に指定する繰り返し回数をOCCURS句の繰り返し以下に修正してください。                                                                                    |
| 9999E    | -999999100 | FOR句に指定した値に誤りがあります。     | FOR句に指定する繰り返し回数を1以上の値に修正してください。                                                                                             |
| 999SA    | -999999700 | カーソルがオープンされていません。       | カーソルが使用可能状態ではありません。カーソルを使用しているSQL 文の順序を調査し対処してください。                                                                         |
| 999SB    | -999999600 | 被準備文が準備されていません。         | 動的SQL 文のシーケンスを調査し対処してください。                                                                                                  |
| 999SC    | -999999500 | カーソルはすでにオープンされています。     | カーソルはすでに使用可能状態になっています。カーソルを使用しているSQL 文の順序を調査し対処してください。                                                                      |
| ?????    | -999999992 | 不正な処理が発生しました。           | システムエラーです。<br>提供元担当者に連絡してください。                                                                                              |

## 22.2.13 ODBCドライバ使用時の注意事項

ここでは、ODBCドライバ使用時の注意事項について記述しています。前述したことも含まれていますが、重要な情報ですので必ずお読みください。

### 22.2.13.1 SQL文の文法上の制限事項

ここでは、SQL文の文法上の制限事項について説明します。

#### データ部

- COBOLでは、ODBCのデータ型とホスト変数のデータ型の対応付けを規定しています。[参照]“[22.2.11 ODBCで扱うデータとの対応](#)”  
ホスト変数を使用する場合は、ODBCドライバの規定するデータ型に対応するホスト変数を使用してください。  
規定外の対応付けを行った場合は、データの内容は保証されません。
- ODBCのドライバの仕様によっては、あるデータ型が、COBOLのホスト変数で扱えない場合もあります。
- 日本語データは、特定のデータベースとODBCドライバの組み合わせでだけ使用できます。

#### 手続き部

- ODBCドライバにより、使用できる埋込みSQL文やその指定方法が異なります。  
COBOLソースプログラム中に埋込みSQL文を記述する場合は、COBOLの仕様だけでなく、ODBCドライバの規定するSQL文を確認し、ODBCドライバに関係するソフトウェア/ハードウェア、およびデータベース管理システムの関連マニュアルも参照してください。
- COMMIT文またはROLLBACK文を使用してトランザクションを終了しなければ、動作を保証できない場合があります。また、DISCONNECT文でコネクションを切断する前には、必ずCOMMIT文またはROLLBACK文を記述してトランザクションを終了してください。

- SQL記述子域は使用できません。したがって、以下の埋込みSQL文または指定は記述できません。
  - ー 記述子名を指定したINTO句およびUSING句
  - ー ALLOCATE DESCRIPTOR 文、DEALLOCATE DESCRIPTOR 文、GET DESCRIPTOR 文、SET DESCRIPTOR 文、DESCRIBE 文などの記述子域に関するSQL文
- DEALLOCATE PREPARE文は記述できません。
- 定義系(DDL系)SQL文は記述できません。
- 動的SQL文であるPREPARE/EXECUTE文、EXECUTE IMMEDIATE文を使用する場合、埋込み例外宣言のNOT FOUND指定は以下のSQL文に対してだけ有効です。
  - ー SELECT文(PREPARE/EXECUTE文の場合だけ)
  - ー UPDATE文(探索)
  - ー UPDATE文(位置付け)
  - ー DELETE文(探索)
  - ー DELETE文(位置付け)
  - ー INSERT文
  - ー 動的UPDATE文(探索)
  - ー 動的UPDATE文(位置付け)
  - ー 動的DELETE文(探索)
  - ー 動的DELETE文(位置付け)
- 動的カーソル宣言で使用する文識別子と同名の文識別子をEXECUTE文で指定することはできません。または、EXECUTE文に指定する文識別子と同名の文識別子を動的カーソル宣言で使用することはできません。
- 埋込みSQL文の文法の詳細は、各データベースおよびデータベース関連製品の仕様に従います。

## 22.2.13.2 埋込みSQL文の実行時の注意事項

ここでは、埋込みSQL文の実行時の注意事項について説明します。

- NetCOBOLよりODBCの環境を使用してデータベースにアクセスする場合は、アクセス対象のデータベースのためのODBCドライバおよびODBCドライバが必要とする環境を構築する必要があります。
- ODBCドライバにより、使用できる埋込みSQL文やその指定方法が異なります。  
COBOLソースプログラム中に埋込みSQL文を記述する場合は、COBOLの仕様だけではなく、ODBCドライバの規定するSQL文を確認し、ODBCドライバに関するソフトウェア/ハードウェア、およびデータベース管理システムのマニュアルも参照してください。
- 埋込みSQL文の記述、ODBC情報ファイルの指定、データソースの設定により、ODBC環境のオプション値が変更されることがあります。この場合、埋込みSQL文を実行する際に各ODBCドライバにより通知メッセージが出力されます。埋込み例外宣言を使用する場合は、エラーが発生したものとして処理されますので、必要に応じてプログラムを変更してください。
- データ操作文以外の埋込みSQL文を動的に実行する場合、各ODBCドライバにより以下の現象が発生することがあります。データ操作文以外の埋込みSQL文を動的に実行しないようにしてください。

|                                                                                     |
|-------------------------------------------------------------------------------------|
| <p>現象：</p> <p>SQLSTATE : 02000</p> <p>SQLCODE : +100</p> <p>SQLMSG : データがありません。</p> |
|-------------------------------------------------------------------------------------|

- COMMIT文またはROLLBACK文を使用してトランザクションを終了しなければ、動作を保証できない場合があります。また、DISCONNECT文でコネクションを切断する前には、必ずCOMMIT文またはROLLBACK文を記述してトランザクションを終了してください。
- 文字型のデータ中に値としてX'00'が格納されている場合、その格納結果および取出し結果に対する保証はできません。

- ・浮動小数型のデータを使用したデータ操作を行う場合、サーバ側とクライアント側でデータの表現方法が異なるため、変換誤差が発生し、期待した結果が得られない場合があります。
- ・埋込みSQL文の動作の詳細は、各データベースおよびデータベース関連製品の仕様に従います。
- ・動的SQL文によるUPDATE文(位置付け)、DELETE文(位置付け)は、@SQL\_CONNECTION\_SCOPE(コネクションの有効範囲の指定)にOBJECT\_INSTANCEまたはTHREADが指定された場合に使用することができます。
- ・UPDATE文(位置付け)または、DELETE文(位置付け)を実行できるかどうかは、@SQL\_CONCURRENCYと@SQL\_CURSOR\_TYPEの組み合わせにより決まります。
- ・@SQL\_CURSOR\_TYPE(カーソルタイプ種別の指定)にFORWARD\_ONLYを指定している、または省略している場合、FETCH PRIOR文を実行すると以下のようなエラーが発生しますが、エラーの内容はデータベースにより異なります。@SQL\_CURSOR\_TYPEにFORWARD\_ONLY以外を指定してください。カーソル種別は、データソース(ODBCドライバ、データベース、データベース関連製品)に依存します。使用するデータベースに該当するカーソルタイプがあるか確認してください。

現象：  
 SQLSTATE : S1106  
 SQLCODE : +0  
 SQLMSG : フェッチの型が範囲を超えています。

### 22.2.13.3 各ODBCドライバ固有の留意事項

ここでは、各ODBCドライバ固有の留意事項について説明します。

#### Oracle(R) ODBCドライバ使用時の留意事項

CALL文で、以下を呼び出すことはできません。

- ・ファンクション
- ・パッケージ化されたストアードプロシージャ
- ・パッケージ化されたファンクション

#### Microsoft(R) SQL Server Native Client 10.0(TM) / Microsoft(R) SQL Native Client(TM) ODBCドライバ使用時の留意事項

- ・CONNECT文の実行時に、以下の現象(通知メッセージの出力)が発生しますが、正常に接続されています。  
埋込み例外宣言がCONNECT文より前に記述されている場合、エラーが発生したものととして処理されますので、埋込み例外宣言はCONNECT文より後ろに記述してください。

現象  
 SQLSTATE : 01000  
 SQLCODE : +5701  
 SQLMSG : データベースのコンテキストを 'データベース名' に変更しました。

- ・カーソル系データ操作文のDELETE文(位置付け)、UPDATE文(位置付け)は特定の条件下でだけ使用可能です。[参照][“22.2.13.3 各ODBCドライバ固有の留意事項”](#)の“特定の条件”  
 なお、@SQL\_CONCURRENCYを指定する場合の留意事項は、以下を参照してください。また、ODBC情報ファイルの作成、データソース(ODBCドライバ、データベース、データベース関連製品)のマニュアルを参照してください。[参照][“22.2.8.1.2 ODBC情報ファイルの作成”](#)
- ・@SQL\_CONCURRENCYを指定した場合、カーソル系データ操作文のDELETE文(位置付け)、UPDATE文(位置付け)が使用できます。ただし、カーソル定義の対象となる表では、1つ以上の一意なインデックスが存在しなければならない場合があります。一意なインデックスを作成する例を以下に記述します。
  - ー 表の作成時に任意の列に対して、PRIMARY KEY制約またはUNIQUE制約を付加します。
  - ー 既存の表の任意の列に対して、一意なインデックスを作成します。(例:CREATE UNIQUE INDEX...)



## 注意

一意なインデックスの作成時に指定した任意の列には、重複値は許されません。つまり、任意の列の値は、それぞれ一意でなければなりません。

既存の表の任意の列に対して、一意なインデックスを作成する場合、任意の列に重複値があるとエラーメッセージが返され、処理が失敗します。

- 一意なインデックスを作成しないで、@SQL\_CONCURRENCYを指定した場合、カーソル系データ操作文のDELETE文(位置付け)、UPDATE文(位置付け)の実行時に以下の現象が発生し、処理が失敗する場合があります。

### 現象：

```
SQLSTATE : S1009
SQLCODE : +16929
SQLMSG : Cursor is read only.
```

- @SQL\_CONCURRENCYを指定した場合の動作例を、以下に説明します。  
2つのクライアントA、Bで、最初にクライアントA、次にクライアントBの順番で同一サーバの同一表、同一行に対して、カーソル系データ操作文のUPDATE文(位置付け)を行います。  
このときにおけるロック、ロック解除のタイミングおよびクライアントの実行状態について説明します。

| クライアントA                        | クライアントB                        |
|--------------------------------|--------------------------------|
| (1) DECLARE カーソル名 CURSOR FOR.. | (2) DECLARE カーソル名 CURSOR FOR.. |
| (3) OPEN カーソル名                 | (4) OPEN カーソル名                 |
| (5) FETCH カーソル名 INTO ..        | (6) FETCH カーソル名 INTO ..        |
| (7) UPDATE 表名 ..               | (8) UPDATE 表名 ..               |
| WHERE CURRENT OF カーソル名         | WHERE CURRENT OF カーソル名         |
| (9) CLOSE カーソル名                | (10) CLOSE カーソル名               |
| (11) COMMIT                    | (12) COMMIT                    |

【注意】( )内の数字は実行順序です。

### 例1

```
クライアントA @SQL_CONCURRENCY = LOCK
クライアントB @SQL_CONCURRENCY = LOCK / ROWVER / VALUES
```

### 結果

- クライアントAが、(5)のFETCH文を実行した時点で(1)のDECLARE文で記述された表全体がロックされます。  
この状態はクライアントAが(11)のCOMMIT文でトランザクションが確定されるまで続きます。
- クライアントBでは(6)のFETCH文が実行待ち状態になります。

### 例2

```
クライアントA @SQL_CONCURRENCY = ROWVER / VALUES
クライアントB @SQL_CONCURRENCY = LOCK
```

### 結果

- クライアントBが、(6)のFETCH文を実行した時点で(2)のDECLARE文で記述された表全体がロックされます。  
この状態はクライアントBが(12)のCOMMIT文でトランザクションが確定されるまで続きます。
- クライアントAでは(7)のUPDATE文が実行待ち状態になります。



### 例3

```
クライアントA @SQL_CONCURRENCY = ROWVER
クライアントB @SQL_CONCURRENCY = ROWVER
```

#### 結果

- クライアントAが、(7) のUPDATE文を実行した時点で(1) のDECLARE文で記述された表全体がロックされます。この状態はクライアントBが(11)のCOMMIT文でトランザクションが確定されるまで続きます。
  - クライアントBでは(8)のUPDATE文がエラーになります。
  - また、(6)と(7)の順序が逆の場合、(6)のFETCH文が実行待ち状態になります。
- ・ データを更新しないカーソルで以下のオプションを指定すると、性能の向上が期待できます。
    - ー @SQL\_CONCURRENCY = READ\_ONLY
    - ー @SQL\_CURSOR\_TYPE = FORWARD\_ONLY
  - ・ 1つのコネクション接続で、カーソルの既定の結果セットを利用中にSQL文を実行した場合、接続エラーが発生します。このエラーは、カーソルタイプをサーバカーソルに変更することで回避できます。サーバカーソルへの変更は、@SQL\_CONCURRENCYとSQL\_CURSOR\_TYPEによって変更できます。デフォルトでは、この値は、@SQL\_CONCURRENCYがREAD\_ONLYで@SQL\_CURSOR\_TYPEがFORWARD\_ONLYになっているため、既定の結果セットになっています。
  - ・ オープン状態のカーソルが存在する状態で、別のカーソルをオープンするか、または、オープン状態のカーソルに関連のない埋込みSQL文を実行した場合、以下の現象が発生します。

#### 現象

```
SQLSTATE : S1000
SQLCODE : 0
SQLMSG : 他のコマンドの結果のために、接続がビジー状態になっています。
```

このような場合、オープン状態のカーソルをクローズしてから、他の埋込みSQL文を実行するようにしてください。

また、これらのSQL文を実行可能にするためには、NetCOBOLおよびデータソース(ODBCドライバ、データベース、データベース関連製品)の環境が、特定の条件に合致している必要があります。[参照]“[22.2.13.3 各ODBCドライバ固有の留意事項](#)”の“特定の条件”

- ・ 次に示すカーソル宣言で定義されたカーソルをオープンしようとすると、以下の現象が発生し、カーソルオープンが失敗します。

#### カーソル宣言

```
DECLARE カーソル名 CURSOR FOR SELECT ... FOR UPDATE
```

#### 現象

```
SQLSTATE : 37000
SQLCODE : +1003
SQLMSG : FOR UPDATE句はDECLARE CURSORでだけ許されます。
```

したがって、上記のような条件下ではカーソル操作を実行することはできません。

このカーソルオープンを成功させ、カーソル操作を実行可能にするためには、NetCOBOLおよびデータソース(ODBCドライバ、データベース、データベース関連製品)の環境が、特定の条件に合致している必要があります。[参照]“[22.2.13.3 各ODBCドライバ固有の留意事項](#)”の“特定の条件”

### 特定の条件

ODBC情報ファイルのサーバ情報に@SQL\_CONCURRENCY(カーソルの同時実行)の値として、LOCK、ROWVERまたはVALUESを指定した場合です。

### 22.2.13.4 埋込みSQL文の実行時の定量制限

ここでは、埋込みSQL文の実行時の定量制限について説明します。

- 埋込みSQL文の最大長は、16384バイトです。ただし、これは、ODBCドライバと関係する環境により制限を受けることがあります。たとえば、ネットワーク部分を受け持つソフトウェアのデータ転送の最大長によって制限を受ける場合があります。

また、ODBCドライバがCOBOLから受け渡された埋込みSQL文を加工している場合は、COBOLソースプログラムの記述より実際の埋込みSQL文が長くなる場合もあります。

- クライアントとサーバ間で入出力できる領域長に制限はありませんが、データソース(ODBCドライバ、データベース、データベース関連製品)の環境により制限を受ける場合があります。
- SQLMSGに設定されるデータソースのメッセージ文字列の最大長は1024バイトです。最大長を超えたメッセージ文字列は切り捨てられます。

## 22.2.14 デッドロック出口

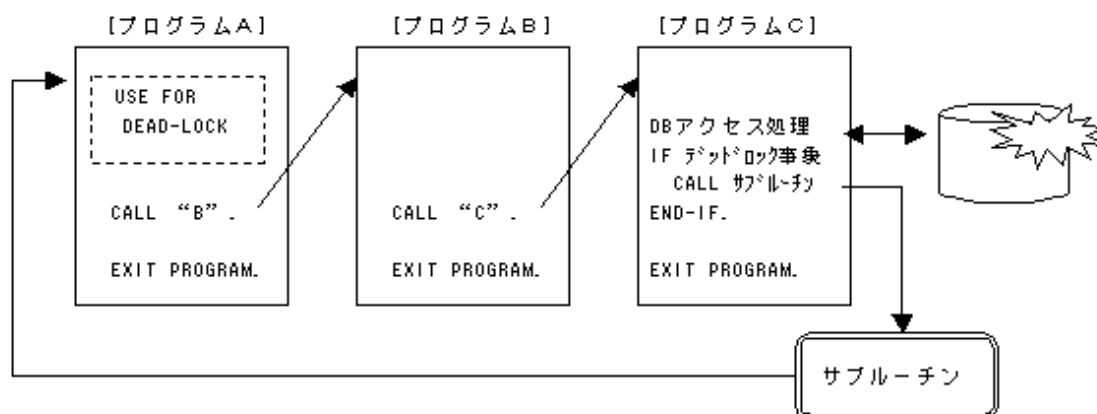
プログラムがデータベースをアクセスする時にアクセス競合が起こり、プログラム同士で占有解除を待つような状態が発生することをデッドロック状態といいます。デッドロック状態が発生すると、デッドロック事象がデータベースからプログラムに通知されます。

デッドロック状態が発生したとき、プログラムにデッドロック出口の記述がある場合には、デッドロック出口のスケジュールを行うことができます。デッドロック出口には、デッドロック発生後の処理手続きを記述できます。

デッドロック出口は、USE FOR DEAD-LOCK文で記述します。USE FOR DEAD-LOCK文については、“COBOL文法書”を参照してください。

### 22.2.14.1 デッドロック出口スケジュールの概要

デッドロック出口スケジュールとは、デッドロック事象が発生したプログラムからデッドロック出口を記述したプログラムへ制御を戻すことを言います。



デッドロック出口スケジュールを行うためには、デッドロック事象を通知されたCOBOLプログラムからNetCOBOLが提供するサブルーチン呼び出します。

デッドロック出口は、デッドロック出口を定義したプログラムを実行した時点でNetCOBOLランタイムシステムに登録され、デッドロック出口を定義したプログラムのEXIT PROGRAM文を実行した時点で登録が解除されます。

サブルーチン呼び出してデッドロック出口スケジュールを実行すると、サブルーチン呼び出したプログラムまたはその上位のプログラムで、サブルーチン呼び出したプログラムに最も近いプログラムに記述されたデッドロック出口に制御が戻ります。このとき、サブルーチン呼び出したプログラムと制御が戻されるデッドロック出口を記述したプログラムの間のプログラムについては、各プログラムのEXIT PROGRAM文相当の終了処理が行われます。

デッドロック出口スケジュールサブルーチンの呼出し方については、“[G.1.6 デッドロック出口スケジュールサブルーチン](#)”を参照してください。

### 22.2.14.2 注意事項

- サブルーチン呼び出したプログラムからデッドロック出口を記述したプログラムの呼出しの間に他言語プログラムがある場合、他言語プログラムの回収処理は行われません。また、デッドロック出口で処理の再開を行う場合には、他言語プログラムに再入することになります。このため、他言語プログラムは再入可能かつ資源回収不要な構造である必要があります。

- デッドロック事象の発生は、利用者の責任で判定処理を行う必要があります。
- デッドロック処理手続き実行後は、GO TO文により宣言節以外の手続きを実行しなければなりません。デッドロック処理手続きの最後に制御が渡ると、プログラムはJMP0004I-Uのメッセージを出力して異常終了します。
- デッドロック事象が通知されるとデータベースのトランザクションはキャンセルされます。データベースによるキャンセル対象となるものの以外で更新しているファイル等についてのリカバリは利用者の責任で行わなければなりません。
- デッドロック出口では、デッドロック処理手続き中からGO TO文等で宣言部分以外の手続きに制御を移すことができますが、プログラムの状態および環境はデッドロック処理手続きに制御が渡る前の状態のままになります。これを適当な状態に戻すのは利用者の責任です。例えば、データ項目の内容やALTER文でGO TO文の飛び先を変更している場合は、デッドロック処理手続き中で適当な値に戻さなければなりません。
- USE 節からデッドロック出口スケジュールサブルーチンを呼び出してはなりません。
- 翻訳オプションLANGLVL(68/74)が指定されている場合、PERFORM文の戻り点については利用者が自由に変更できないので、PERFORM文で参照される節または段落が他の手段(例えばGO TO文など)で実行される可能性があるとき、その節または段落の中にデータベースを操作する文を書いてはなりません。

## 第23章 サーバ・タイプのアプリケーション

本章では、サーバ・タイプのアプリケーションについて説明します。

なお、Web連携機能および利用者定義の情報をイベントログに出力する機能は、サーバ向け運用環境製品固有の機能です。

### 23.1 バックグラウンド処理

#### 23.1.1 画面による入出力操作の抑止方法

一般的にバックグラウンド処理では、画面の操作は必要ありません。ここでは、画面を使用する機能を抑止する方法について説明します。

画面の表示は、利用者がソースプログラム中で画面を使用する機能を使用した場合と、実行時メッセージなど、利用者の使用する機能に関係なくCOBOLランタイムシステムがアプリケーションの状態などを通知するために使用する場合があります。

具体的には以下のとおりです。

- ・ 利用者がソースプログラム中で使用できる画面入出力機能
  - ー 表示ファイル機能(画面入出力機能)
  - ー スクリーン操作機能
  - ー 小入出力機能
- ・ COBOLランタイムがユーザインタフェースとして画面を使用する機能
  - ー 実行時メッセージ
  - ー NetCOBOLのアイコン
  - ー コンソール画面クローズ時の確認メッセージ

画面による入出力を行わないアプリケーションを作成する目的の場合、利用者がソースプログラム中に記述できる画面入出力機能は使用しないでください。ただし、小入出力機能は、画面による入出力を行わずに、ファイルにより入出力を行うことが可能です。詳細は、“[11.1 小入出力機能](#)”を参照してください。

ここでは、COBOLランタイムがユーザインタフェースとして画面を使用する機能の抑止方法について説明します。

##### 1. 実行時メッセージの出力先を変更する方法

環境変数情報にファイル名を指定することでファイルに出力する方法、およびイベントログに出力する方法があります。

- ー 環境変数情報にファイル名を指定することでファイルに出力する方法

環境変数情報@MessOutFile=ファイル名を指定します。詳細は、“[5.4.1.59 @MessOutFile\(メッセージを出力するファイルの指定\)](#)”を参照してください。

- ー イベントログに出力する方法

環境変数情報@CBR\_MESSAGE=EVENTLOGを指定します。詳細は、“[5.4.1.33 @CBR\\_MESSAGE\(実行時メッセージの出力先の指定\)](#)”を参照してください。

上記の指定を同時に行った場合の優先順位は以下のとおりです。

|      |                             |      |
|------|-----------------------------|------|
| (高い) | @CBR_MESSAGE > @MessOutFile | (低い) |
|------|-----------------------------|------|

Uレベル以外の実行時メッセージは抑止したいが、Uレベルのメッセージは出力したい場合は、環境変数情報@NoMessage=YESを指定してください。詳細は、“[5.4.1.61 @NoMessage\(実行時メッセージおよびSYSERRの出力抑止指定\)](#)”を参照してください。

##### 2. NetCOBOLのアイコンの表示を抑止する方法

環境変数情報@ShowIcon=NOを指定します。詳細は、“[5.4.1.67 @ShowIcon\(NetCOBOLのアイコン表示の抑止指定\)](#)”を参照してください。

### 3. コンソール画面クローズ時の確認メッセージの出力を抑止する方法

環境変数情報@WinCloseMsg=OFFを指定します。詳細は、“[5.4.1.68 @WinCloseMsg \(ウィンドウを閉じるときのメッセージ表示の指定\)](#)”を参照してください。

DISPLAY文でログ情報をコンソール画面に出力するような場合、COBOLアプリケーションの終了時に、COBOLがコンソール画面を閉じる旨を通知するメッセージを抑止したい場合に指定します。

## 23.1.2 コマンドプロンプトウィンドウの使用法

コマンドプロンプトウィンドウを使用して、ACCEPT文の入力、DISPLAY文の出力および実行時メッセージの表示を行うことができます。

コマンドプロンプトウィンドウやバッチファイルから起動した場合は、起動したコマンドプロンプトウィンドウまたはバッチファイルが生成するウィンドウをコンソール画面として使用します。また、アイコンの状態で起動した場合は、起動と同時にコマンドプロンプトウィンドウが生成され、これをコンソール画面として使用します。

主プログラムがCOBOLの場合、翻訳オプションMAIN(MAIN)を指定します。主プログラムが他言語の場合、環境変数情報@CBR\_CONSOLE=SYSTEMを指定します。

環境変数情報@CBR\_CONSOLE=SYSTEMの指定は、主プログラムがCOBOLの場合で翻訳オプションMAIN(WINMAIN)を指定した場合、または、主プログラムが他言語でWinMain関数の場合、期待したとおりに動作しない場合があります。この場合、@CBR\_CONSOLE=COBOL(省略値)の指定をおすすめします。また、環境変数情報@CBR\_CONSOLE=SYSTEMを指定した場合の動作は、以下のとおりです。

1. 新しいコマンドプロンプトウィンドウが生成されます。  
起動したコマンドプロンプトウィンドウやバッチファイルが生成するウィンドウはコンソール画面として使用できません。コンソール機能の使用時に新しくコマンドプロンプトウィンドウを生成して、コンソール画面として使用します。
2. 入出力に失敗する場合、STARTコマンドを使用して回避してください。なお、コマンドプロンプトウィンドウへの入出力に失敗する場合、コマンドプロンプトウィンドウへの実行時メッセージの出力にも失敗するため、エラーの事象を確認できないことがあります。実行時メッセージをイベントログやファイルに出力することをおすすめします。

## 23.1.3 サービス配下で動作するプログラム

### サービス配下のアプリケーション環境

サービスからCOBOLプログラムが呼び出される場合、プリンタ情報やネットワークドライブ情報などの取得に失敗して、正常に動作しない場合があります。

原因は、サービスから呼び出されたCOBOLプログラムがプリンタ情報などをシステムから取得する際、システムに設定されているユーザごとの設定情報を参照できないためと考えられます。

通常、サービスから呼び出されるCOBOLプログラムはシステムアカウントで動作します。このとき、システム上にはユーザごとの設定(たとえば、レジストリのHKEY\_CURRENT\_USERなどの情報)は構築されません。このため、COBOLプログラムはデフォルトプリンタなどのユーザごとの情報を取得することができません。ユーザごとの設定は、そのユーザがシステムにログオンしたときに構築されます。

サービスによっては、特定のユーザアカウントでCOBOLプログラムが動作するように設定することが可能な場合もあります。このときは、COBOLプログラムが動作するユーザアカウントと同じユーザであらかじめシステムにログオンしておくと、ユーザの設定が参照できます。また、COBOLプログラムがシステムアカウントで動作しても、ユーザごとの情報をシステムの情報として参照できるように作成されたサービスもあります。しかし、前述したようにサービスから呼び出されたCOBOLプログラムでは、ユーザごとの情報を参照できないのが一般的です。

### プログラム作成時の注意事項

サービスから呼び出されるCOBOLプログラムを作成するときには、以下の注意が必要です。

#### 環境変数

ユーザ環境変数に設定した環境変数情報は参照できません。COBOLプログラムで必要な環境変数情報は、システム環境変数に設定してください。

## プリンタ情報

FORMAT句ありなし印刷ファイルおよび表示ファイル(宛て先PRT)を使用したCOBOLプログラムをサービス配下で実行する場合、そのファイルの出力先への割当ては明にプリンタ名を指定してください。出力先への指定にデフォルトのプリンタ、ローカルプリンタポート名(LPTn:)および通信ポート名(COMn:)を指定した場合、印刷されないことがあります(ファイルの割当てエラー)。詳細は、“[8.1.11 サービス配下の注意事項\(印刷時\)](#)”を参照してください。

## ネットワークドライブ情報

ネットワークドライブ情報は参照できません。ネットワーク接続されている環境のファイルなどをアクセスする場合には、UNC指定を使用してください。



### 例

.....  
ファイル識別名 = ¥¥FILESERVER¥¥FILE¥WORK. DAT  
.....

## ODBC情報

ユーザデータソースは参照できません。データソースをシステムデータソースとして定義してください。詳細は、“[22.2.8.4 連携ソフトウェアおよびハードウェア環境の整備](#)”を参照してください。

## 画面の入出力機能

サービスの多くは、画面による入出力を行いません。画面による入出力を行わないサービスから呼び出されるCOBOLプログラムが画面の入出力を行うと、不可視の画面が入力待ちの状態になり、あたかもシステムが応答しなくなったような状態に見えます。このような場合は、COBOLの画面による入出力機能を抑止する設定を行う必要があります。詳細は、“[23.1.1 画面による入出力操作の抑止方法](#)”を参照してください。

## プログラムのデバッグ

サービスから呼び出されるCOBOLプログラムをデバッグする場合は、プログラムから対話型デバッガを起動する方法を使用します。詳細は、“[20.4.1.2 Interstageなどのサーバ環境で動作するプログラムに対するデバッガ起動方法](#)”を参照してください。

## サービス

COBOLプログラムを呼び出すサービスを作成する場合は、サービス配下で動作するアプリケーションがユーザごとの情報をシステムの情報として参照できるように作成する必要があります。

# 23.2 Web連携

COBOLが提供するWeb連携機能を利用することで、インターネット/イントラネットを活用した基幹業務システムをCOBOLで構築することができます。

インターネット/イントラネット環境では、一般にPerl、java、Visual C++、Visual Basicなどの言語を利用して構築される例が広く知られています。COBOL Web連携機能では、これらの言語に加えCOBOLを利用したインターネット/イントラネットアプリケーションを構築することが可能となります。

これにより、膨大なCOBOL既存資産やCOBOL技術者のノウハウを有効活用し、先端技術・急速に普及するネットワーク環境に対応することができます。

COBOLが提供するWeb連携機能は、大きく分けて以下の2つのカテゴリがあります。

### 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を使用したアプリケーション開発に必要なパラメタ解析や処理結果出力などの複雑な処理を意識しないで、COBOLの知識だけでアプリケーションを作成できます(ISAPI、CGI、ほか)。

### MeFt/Webの利用

Windows上でMeFtを使用したCOBOLアプリケーションまたはクライアント/サーバ環境のMeFt/NET-SV(NET連携機能)を使用したCOBOLアプリケーションを再翻訳しないで、そのままイントラネット環境の基幹業務として運用できます。

詳細は、“[NetCOBOL Web連携ガイド](#)”および“[COBOL Webサブルーチン使用手引書](#)”を参照してください。





注意

Webアプリケーションは、WebサーバあるいはMeFt/Webが提供するサービス配下で動作するアプリケーションとなります。このため、プログラムの作成時には、“[23.1.3 サービス配下で動作するプログラム](#)”を参照し、注意事項を確認してください。

## 23.3 イベントログ

イベントログは、Windowsが提供する機能です。

各アプリケーションがエラーなどの情報をイベントログに記録することにより、システムの管理者はアプリケーションの状態を集中的に管理(イベントの監視)できるようになり、トラブルシューティングの効率が向上します。

ここでは、実行時メッセージおよび利用者が定義する任意の情報をイベントログに出力する機能について説明します。なお、イベントの監視はイベントビューアやSystemwalker/CentricMGRで行います。使用方法については、各製品のマニュアルおよびオンラインヘルプを参照してください。

### 23.3.1 実行時メッセージをイベントログに出力する機能

COBOLランタイムシステムが出力する実行時メッセージをイベントログに出力することができます。イベントログの各項目に対しては、ソース名はNetCOBOL、イベントIDはメッセージ番号、種類は重大度コード、説明には実行時メッセージを出力します。ただし、種類についてはレベルの区分がシステムとCOBOLの実行時メッセージとで一致しないため、次のように対応付けています。

表23.1 実行時メッセージの重大度コードとイベントログの種類の対応

| 重大度コード            | イベントログの種類    |
|-------------------|--------------|
| I (INFORMATION)   | 警告 (WARNING) |
| W (WARNING)       |              |
| E (ERROR)         |              |
| U (UNRECOVERABLE) | エラー (ERROR)  |

実行時メッセージをイベントログに出力する機能を使用するためには、環境変数情報@CBR\_MESSAGEの指定が必要です。詳細は、“[5.4.1.33 @CBR\\_MESSAGE \(実行時メッセージの出力先の指定\)](#)”を参照してください。

また、実行時メッセージの重大度コードの意味については“メッセージ説明書”の“第3章 実行時メッセージ”を参照してください。



注意

ネットワーク上の他のコンピュータ(Windows)に出力することができます。この場合、出力先のコンピュータに、必ずCOBOLをインストールしてください。

### 23.3.2 利用者定義の情報をイベントログに出力する機能

イベントログ出力サブルーチン(COB\_REPORT\_EVENT)を呼び出して、利用者がCOBOLソースプログラム中に定義した文字列などの情報を、イベントログに出力することができます。利用者がサブルーチンに対して指定できる項目と範囲は下表のとおりです。なお、イベントログ出力サブルーチンのインタフェースについては、“[G.1.3 イベントログ出力サブルーチン](#)”を参照してください。

イベントログの指定可能項目

| 項目     | 内容                                                                                                                                                           |
|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ソース    | イベントIDと共にイベントを識別するために使用します。一般的にソースはアプリケーションの名前にする場合がありますが、COBOLのインタフェースでは256バイト以内の任意の文字列を指定できます。ただし、出力可能な文字列の最大長は、システムの定量化に依存します。指定省略時はNetCOBOL Application。 |
| イベントID | ソースと共にイベントを識別するために使用します。0～999までの値が指定可能です。                                                                                                                    |
| 種類     | イベントの種類として情報／警告／エラーを指定します。                                                                                                                                   |
| 説明     | イベントの説明文を最大1024バイトまで指定できます。指定を省略した場合、説明に対する情報は出力されません。                                                                                                       |

| 項目  | 内容                                                                                            |
|-----|-----------------------------------------------------------------------------------------------|
| データ | データ域を出力する場合、出力するデータ域のアドレスと長さを指定します。出力可能なデータ域の最大長はシステムの定量制限に依存します。指定を省略した場合、データに対する情報は出力されません。 |

## 注意

- ネットワーク上の他のコンピュータ(Windows)に出力することができます。  
出力先コンピュータは、実行時メッセージのイベントログ出力先と同じです。実行時メッセージ同様、出力先のコンピュータに、必ずCOBOLをインストールしてください。  
詳細は、“[23.3.1 実行時メッセージをイベントログに出力する機能](#)”を参照してください。
- ソースにデフォルト以外の任意の文字列を出力する場合、出力先コンピュータのレジストリに情報を設定する必要があります。本機能のためのレジストリ情報の設定および削除は“イベントログ出力サブルーチン用レジストリキーの追加・削除”ツールを使用してください。なお、レジストリ情報の設定および削除は、Administratorsグループのユーザなどレジストリキーのアクセス権(値の照会・値の設定・サブキーの作成・削除)のあるユーザが行ってください。



## 第24章 マルチスレッド

本章では、マルチスレッドプログラムについて説明します。

マルチスレッドプログラムを作成することにより、マルチスレッド環境下でCOBOLプログラムを実行させることができるようになります。

なお、マルチスレッド機能は、サーバ向け運用環境製品固有の機能です。

### 24.1 概要

Web連携や分散オブジェクトなどの機能を利用して、COBOLアプリケーションをサーバアプリケーションとして使用する場合、多くのクライアントからの実行要求により、サーバの負荷は増加します。このような運用形態に対して、マルチスレッドを適用することにより、サーバのメモリやハードディスクなどの資源の使用量を減らすことができ、さらに、実行性能も向上させることができます。

#### 24.1.1 特徴

COBOLの既存資産をマルチスレッド環境下で利用可能

COBOLの既存資産は、基本的にプログラムを再翻訳するだけで、マルチスレッド環境下で利用できるようになります。

スレッド間でデータやファイルを共有可能

スレッド間でデータやファイルなどの資源を共有するマルチスレッドの特性を活かしたプログラムも作成できます。一般的に、このようなプログラムを作成する場合は、複数のスレッドが資源を同時にアクセスしないように(以降、スレッドの同期制御と呼びます)、プログラムの作成者が複雑なプログラミングをする必要があります。しかし、COBOLでは、COBOLランタイムシステムが複雑なスレッドの同期制御を自動的にを行い、さらに、スレッドの同期制御を行うためのサブルーチンを提供しているため、プログラムを簡単に作成することができます。

マルチスレッドでのデバッグ支援

NetCOBOLの提供するTRACE機能、CHECK機能、COUNT機能および対話型デバッガにより、マルチスレッドプログラムをデバッグできます。また、デバッグ補助のために、プロセスID/スレッドIDの取得サブルーチンを提供しています。



#### 注意

COBOLでは、スレッドを起動する機能を提供していません。

#### 24.1.2 機能範囲

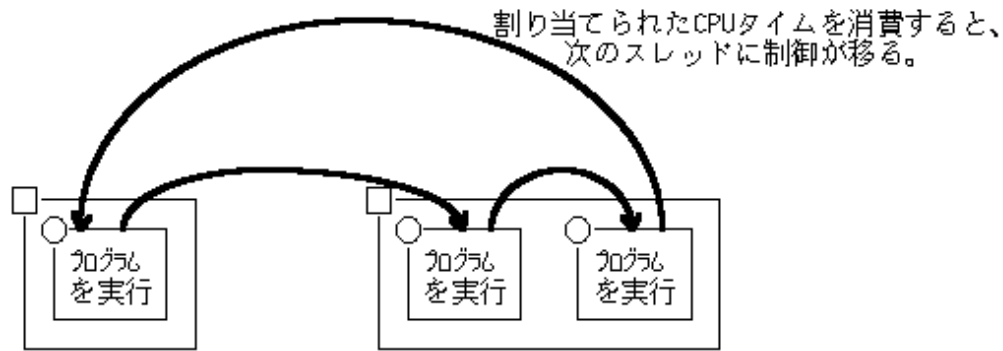
マルチスレッド環境では、COBOLの機能をすべて利用することができます。ただし、関連製品を利用する機能については、その関連製品の対応状況を確認の上、使用してください。[参照]“NetCOBOL ソフトウェア説明書”

## 24.2 マルチスレッドのメリット

### 24.2.1 スレッドとは

スレッドとは、オペレーティングシステムによってスケジュール管理される最小の実行単位です。オペレーティングシステムは、実行するスレッドおよびその実行時期をスケジュールし、CPUタイムを割り当てます。割り当てられたCPUタイムを消費すると、次のスレッドにCPUタイムを割り当てます。

スレッドはプロセスの中に存在し、スレッドによってプログラムが実行されます。プロセスは、メモリ上にあるプログラムのコード、データ、オープンされているファイル、動的に割り当てられたメモリなどの資源から構成され、少なくとも1つのスレッドが存在します。



→ : 実行制御の流れを示す。

□ : プロセスを示す。

○ □ : スレッドを示す。

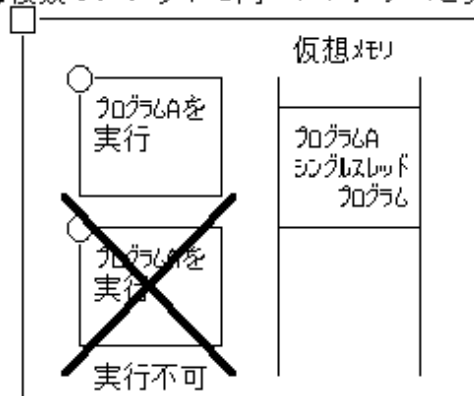
以降、プロセスとスレッドをこの図で表記します。

## 24.2.2 マルチスレッドプログラムとは

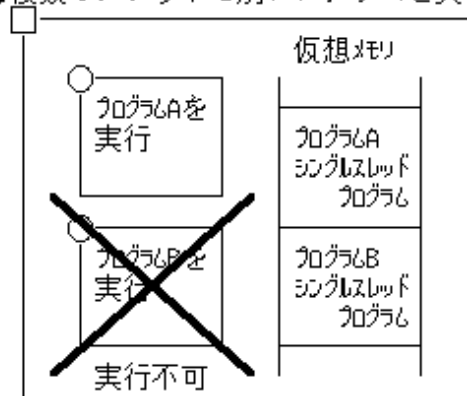
シングルスレッドプログラムは、プロセス内の1つのスレッドでしかCOBOLプログラムを実行できません。このため、プロセス内の複数のスレッドで、同じプログラムを実行することも、異なるプログラムを実行することもできません(下図参照)。しかし、マルチスレッドプログラムにすることによって、プロセス内の複数のスレッド、すなわち、マルチスレッドでCOBOLプログラムを実行できるようになります。

COBOLでは、翻訳オプションTHREAD(SINGLE)で翻訳したプログラムはシングルスレッドプログラムとなり、翻訳オプションTHREAD(MULTI)で翻訳したプログラムはマルチスレッドプログラムとなります。

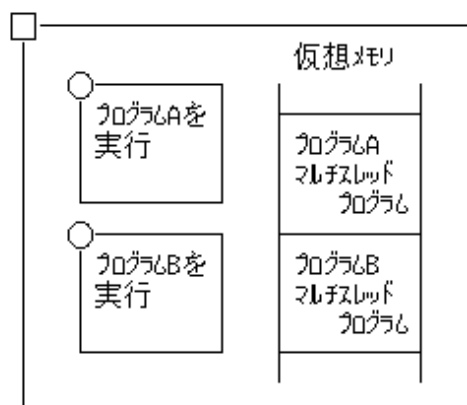
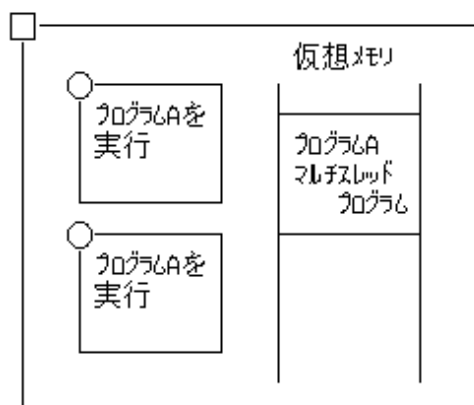
●複数のスレッドで同一プログラムを実行



●複数のスレッドで別プログラムを実行



マルチスレッドプログラムに  
することにより実行できる。



## 24.2.3 マルチスレッドの効果

サーバから起動されるWeb連携や分散オブジェクトなどの機能を利用したサーバアプリケーションをマルチスレッドプログラムにすることによって、以下の効果が得られ、高性能なサーバアプリケーションになります。

### 高速なスタートアップ

シングルスレッドプログラムを実行するためには、プロセスを起動する必要があります。プロセスを起動するには、プロセス空間を獲得し、実行可能ファイルと必要ならDLLをディスクから読み込み、そこにロードする必要があります。しかし、スレッドだけを起動するのであれば、このような処理は不要になるため、起動時間が短縮されます。

### メモリ節約と容易な資源共有

プロセス内のすべてのスレッドで、プロセス空間は共有されるため、メモリの節約とともにスレッド間で資源の共有が可能となります。

以下に、WebサーバでCGIとISAPIを利用した場合を例にとりマルチスレッドプログラムの効果を説明します。

図24.1 シングルスレッドプログラム(CGIアプリケーション)

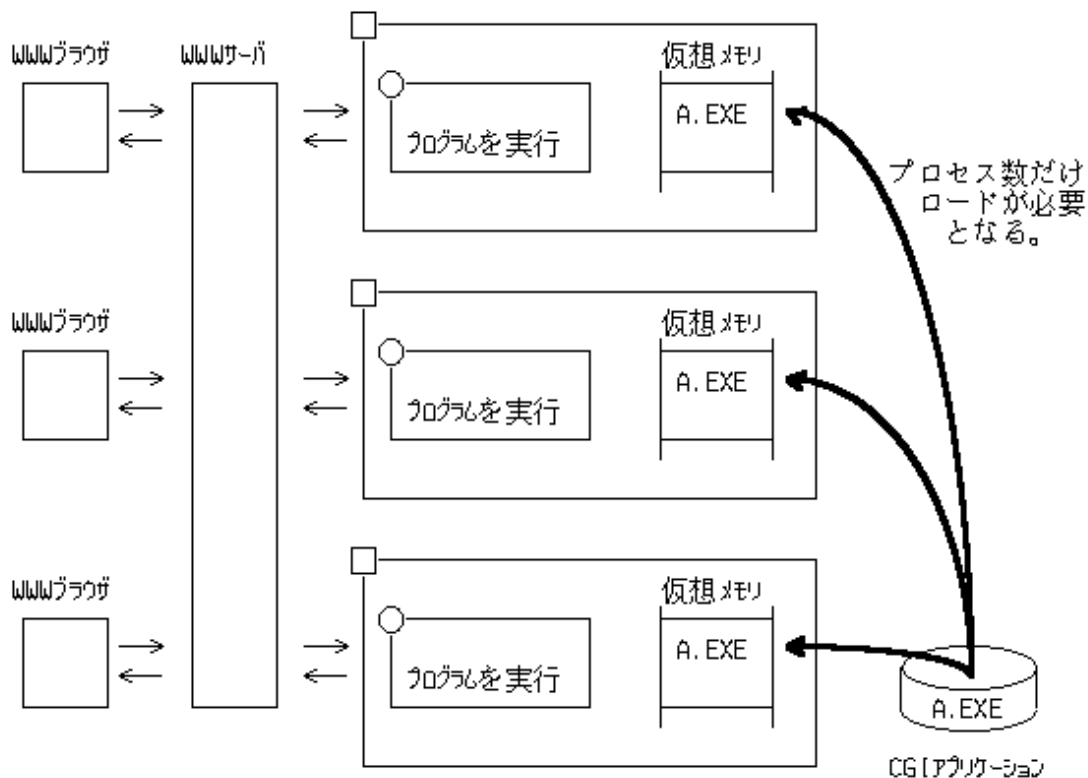
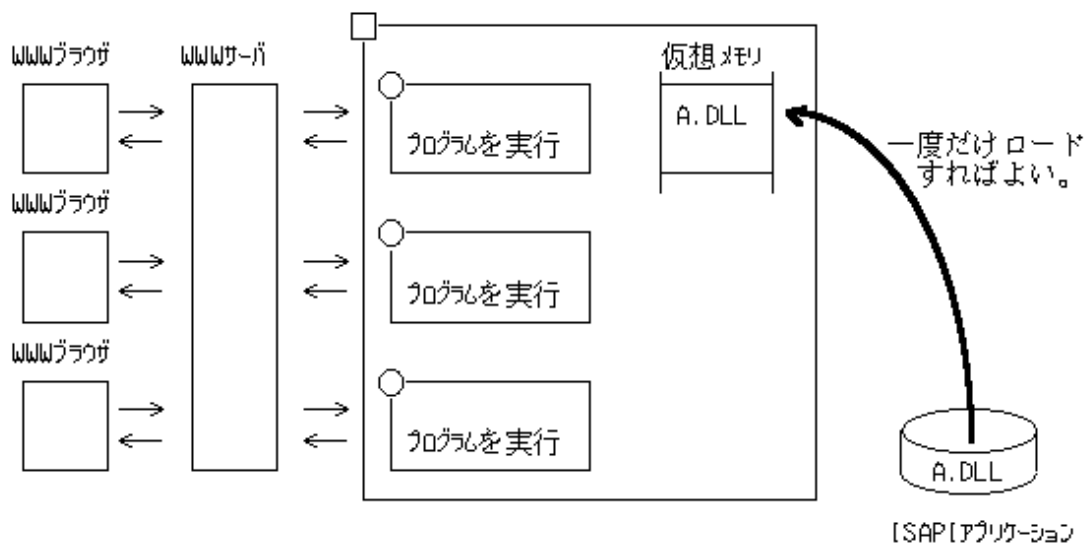


図24.2 マルチスレッドプログラム(ISAPIアプリケーション)



CGI(Common Gateway Interface)を利用した場合、クライアントから要求を受信すると、WebサーバはEXEファイルであるCGIアプリケーションを新しいプロセスとして起動します。プロセスを起動するためには、プロセス空間の獲得、ロードモジュールのロード、データの初期化などの処理が必要であり、オーバーヘッドになります。

これに対して、ISAPI(Internet Server Application Programming Interface)を利用した場合、サーバは新しいプロセスを起動するのではなく、プロセスのスレッドによってISAPIアプリケーションを実行します。ISAPIアプリケーションは、DLLであり、最初に呼び出されたとき

に、プロセス空間にロードされ、通常はその後常駐します。このため、スタートアップのオーバーヘッドがなくなるとともに、必要なメモリの量も削減されます。

## 24.3 マルチスレッドプログラムの基本動作

### 24.3.1 実行環境と実行単位

“10.1.4 COBOLの言語間の環境”で説明したように、COBOLには実行環境と実行単位があります。実行環境はプロセスごとに存在し、実行単位はスレッドごとに存在します。

#### 実行環境

マルチスレッドの実行環境は、プロセスでCOBOLプログラムが初めて呼び出されたときに開設され、プロセスの終了時またはJMPCINT4が呼び出されたときに閉鎖されます。実行環境の開設時には、シングルスレッド同様、COBOLプログラムが実行するために必要となる実行用の初期化ファイルの情報などが取り込まれます。実行環境の閉鎖時には、プロセス単位で管理される資源の解放が行われます。プロセス単位で管理される資源には、以下があります。

- ・ ファクトリオブジェクト
- ・ オブジェクトインスタンス
- ・ システムのコンソールウィンドウ
- ・ 小入出力機能で使用するファイル
- ・ スレッド間共有外部データ/外部ファイル(マルチスレッド固有の機能である翻訳オプションSHREXTを指定して翻訳された外部データと外部ファイル)

#### 実行単位

マルチスレッドの実行単位の開始と終了のタイミングはシングルスレッドと同じです。しかし、COBOLプログラムが複数のスレッドで実行されるため、1つのプロセスに同時に複数の実行単位が存在することになります。実行単位の終了時には、スレッド単位で管理される資源の解放が行われます。スレッド単位で管理される資源には、プログラム定義に宣言されたデータ(スレッド間共有外部データ/外部ファイルは除きます)、COBOLのコンソールウィンドウ、スクリーンウィンドウなどがあります。



#### 注意

JMPCINT2を呼び出した場合は、必ずJMPCINT3を呼び出してください。JMPCINT3が呼び出されない場合、COBOLの実行単位が終了しないため、スレッドで利用された資源が解放されず、メモリリークの原因になります。

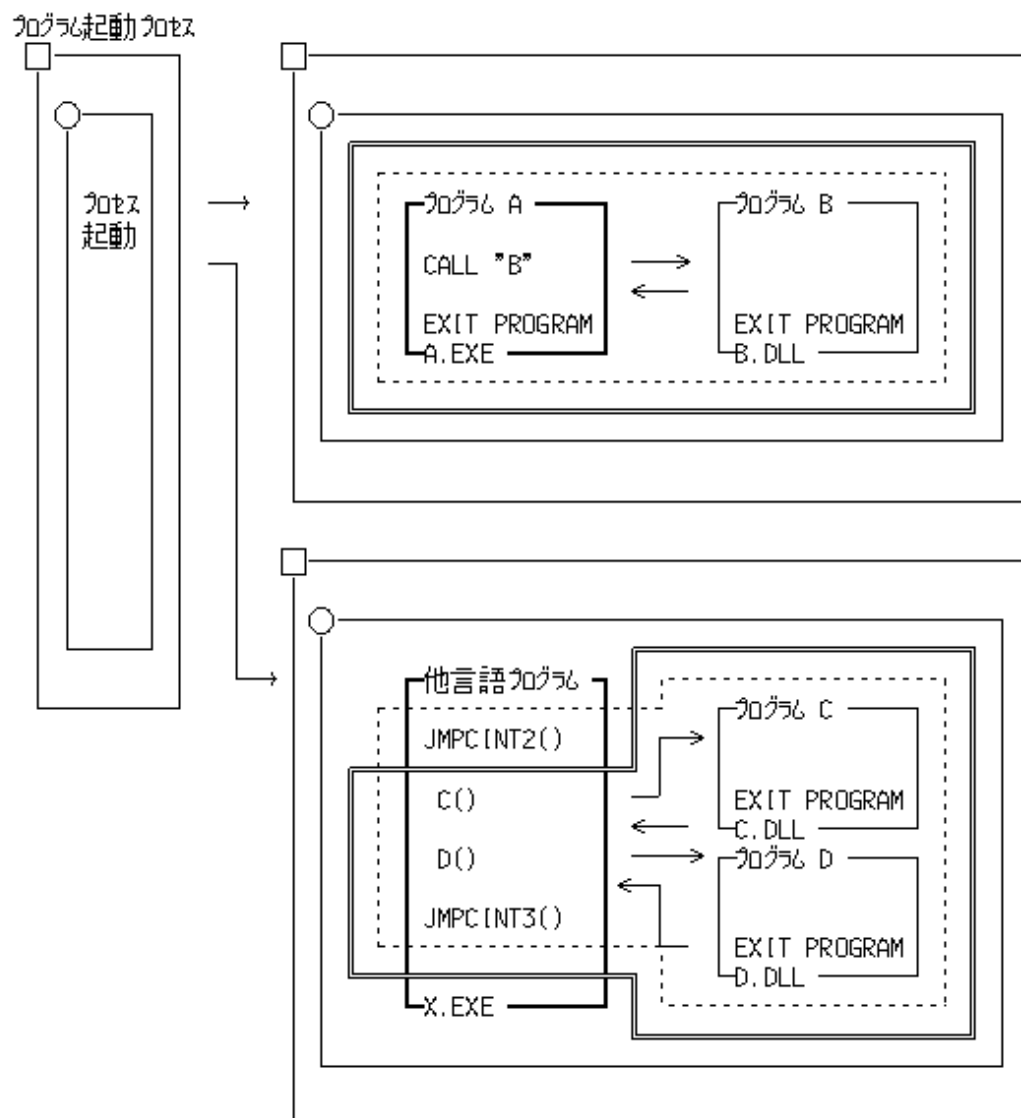
#### JMPCINT4

プロセス終了前に実行環境を閉鎖するためのサブルーチンとして、JMPCINT4を提供しています。このサブルーチンを他言語プログラムから呼び出すことにより、実行環境を閉鎖することができます。このサブルーチンは、プロセス内のすべての実行単位が終了してから呼び出してください。COBOLプログラムの実行中に呼び出されると、実行環境が閉鎖されるため、実行中のCOBOLプログラムは異常終了しますので注意してください。JMPCINT4の呼び出し形式については、“G.2 他言語連携で使用するサブルーチン”を参照してください。

以下に、シングルスレッドプログラムとマルチスレッドプログラムの実行環境と実行単位の間を関係を示します。シングルスレッドプログラムは、プロセスが起動され、そのプロセスのスレッドによって実行されます。それに対して、マルチスレッドプログラムは、プロセス内の別のスレッドが起動され、そのスレッドによって実行されます。

#### シングルスレッドプログラム

シングルスレッドプログラムでは、プロセス内の1つのスレッドだけしかCOBOLプログラムを実行できないため、プロセス内に実行単位は1つしか存在しません。また、実行環境は実行単位の終了時に閉鎖されます。

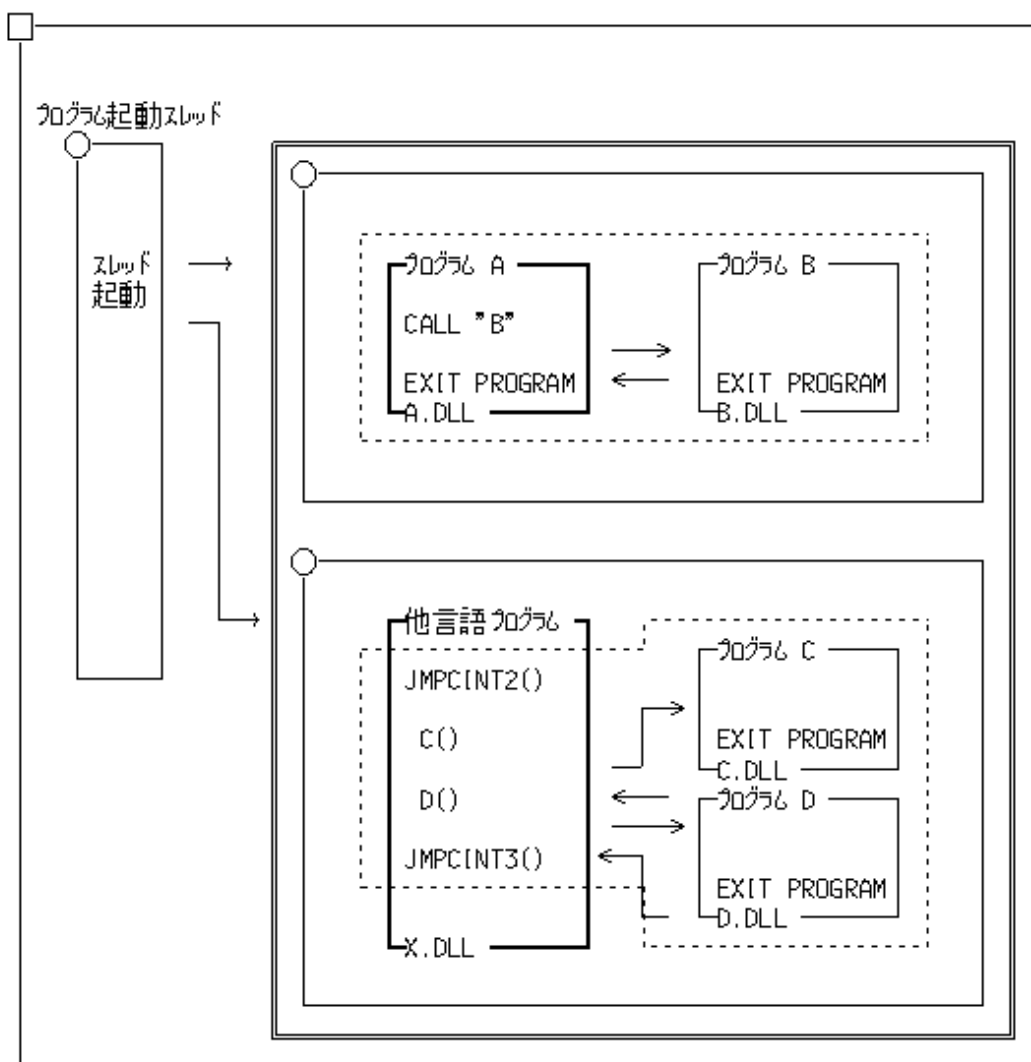


□ : COBOL の主プログラムを示す。 □ : COBOL の実行単位を示す。

□ : COBOL の実行環境を示す。

## マルチスレッドプログラム

マルチスレッドプログラムでは、プロセス内の複数のスレッドで同時にCOBOLプログラムを実行できるため、プロセス内に複数の実行単位が存在できます。実行環境はスレッドがすべて消滅し、プロセスが終了するときに閉鎖されます。



□ : COBOL の主プログラムを示す。 □ : COBOL の実行単位を示す。

□ : COBOL の実行環境を示す。

## 24.3.2 マルチスレッドプログラムのデータの扱い

ここでは、マルチスレッドプログラムでのデータ領域の管理のされ方について説明します。

マルチスレッドプログラムには、プロセス(実行環境)、スレッド(実行単位)および呼出し(呼び出されてから復帰まで)単位で確保/管理されるデータがあります。

プロセス単位で確保/管理されるデータ

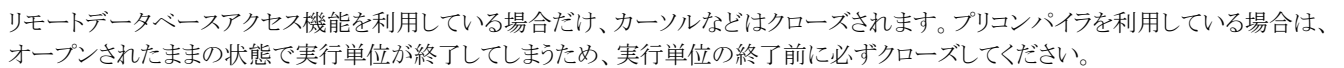
- スレッド間共有外部データとスレッド間共有外部ファイル(注1)
- ファクトリオブジェクト
- オブジェクトインスタンス

ー プログラム定義に宣言されたデータ(注2)

- 一 メソッド定義に宣言されたデータ

注2：スレッド間共有外部データとスレッド間共有外部ファイルを除きます。

プログラム定義に宣言されたデータはスレッドごとに確保されます。この領域は、実行単位の開始時に確保され、実行単位の終了時に解放されます。実行単位の終了時、クローズされていないファイル、カーソルなどは強制的にクローズされます。



|                                                                                                                                                                                                                                                                      |                                                                                                                                                                                                          |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> IDENTIFICATION DIVISION. PROGRAM-ID. A. ; DATA DIVISION. FILE SECTION. FD FILE1 EXTERNAL. 01 REC1 PIC X(80). FD FILE2. 01 REC2 PIC X(80). ; WORKING-STORAGE SECTION. 01 DATA1 PIC X(5) EXTERNAL. 01 DATA2 PIC X(5). ; PROCEDURE DIVISION. ; CALL "B". ; </pre> | <pre> IDENTIFICATION DIVISION. PROGRAM-ID. B. ; DATA DIVISION. FILE SECTION. FD FILE1 EXTERNAL. 01 REC1 PIC X(80). ; WORKING-STORAGE SECTION. 01 DATA1 PIC X(5) EXTERNAL. ; PROCEDURE DIVISION. ; </pre> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

図から分かるように、シングルスレッドプログラムでプロセスごとに確保されていたデータが、マルチスレッドプログラムではスレッドごとに確保されます。



図24.4 シングルスレッドプログラム

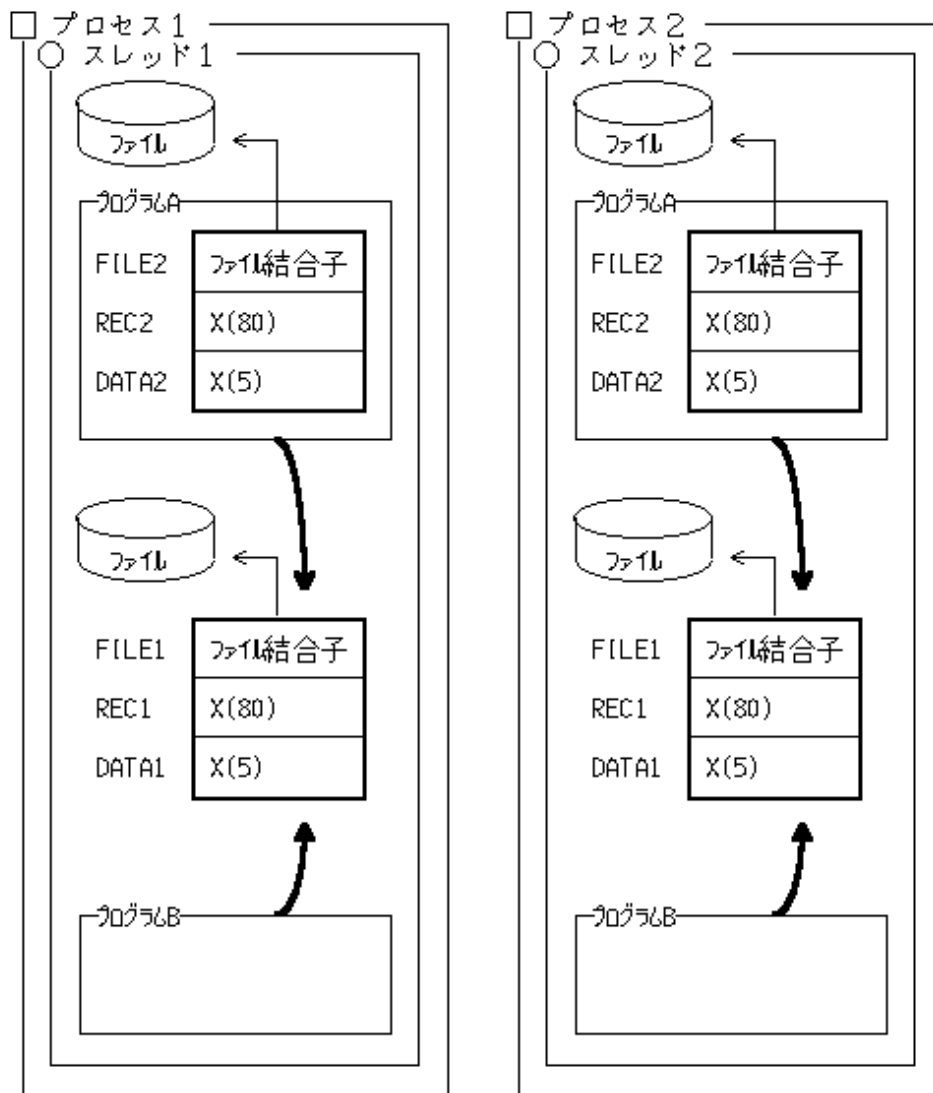
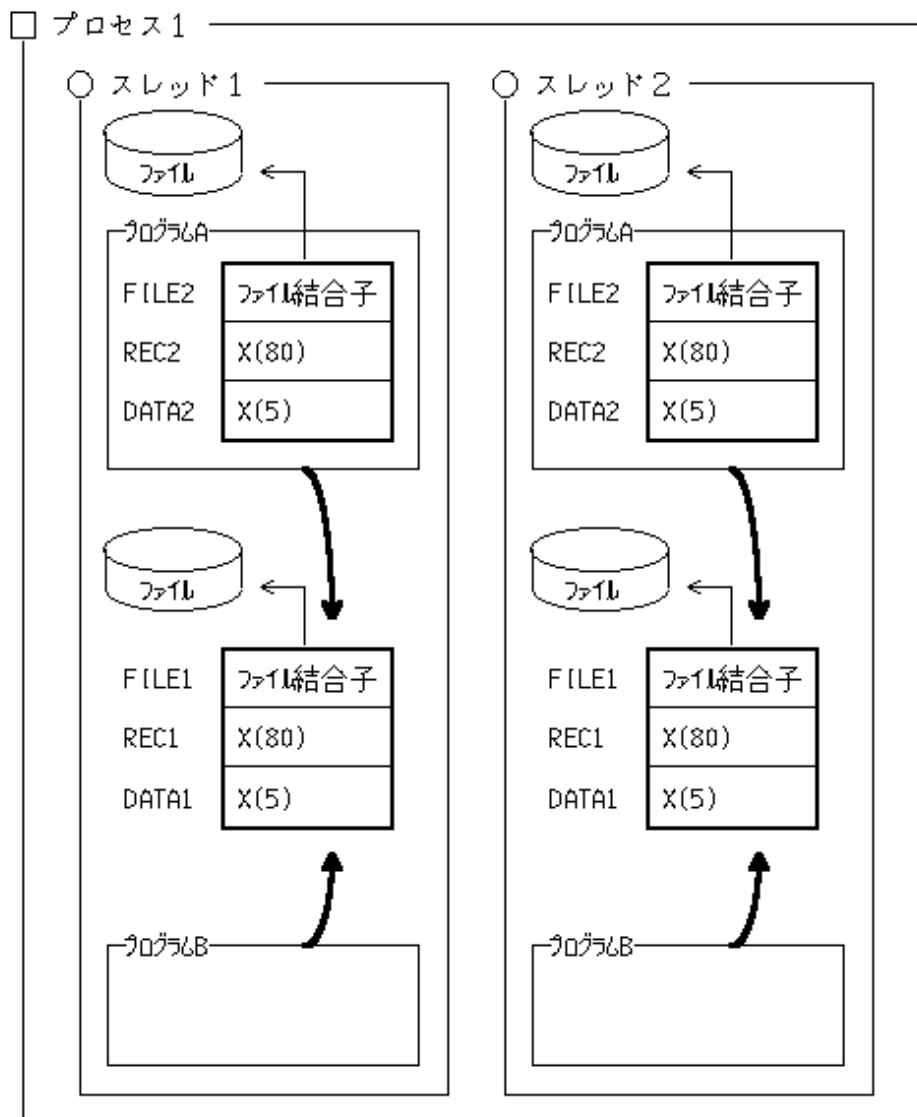


図24.5 マルチスレッドプログラム



### 24.3.2.2 ファクトリオブジェクトとオブジェクトインスタンス

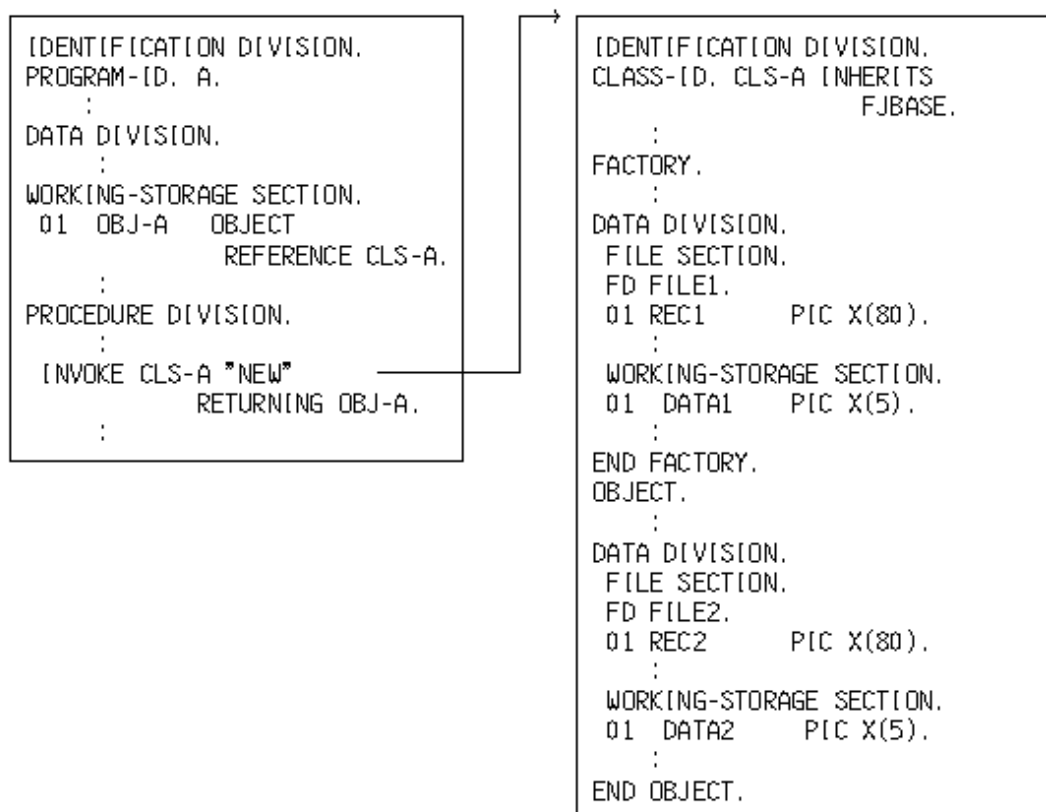
ファクトリオブジェクトとオブジェクトインスタンスはプロセスで管理されます。

各クラスのファクトリオブジェクトはプロセスに1つだけ存在するため、マルチスレッドプログラムでは、つねにスレッド間で共有されます。  
[参照]“[24.4.2.2 ファクトリオブジェクト](#)”

オブジェクトインスタンスもファクトリオブジェクトを介することによってスレッド間で共有することができます。[参照]“[24.4.2.3 オブジェクトインスタンス](#)”

実行環境の終了時に、ファクトリオブジェクトと未解放のオブジェクトインスタンスは解放されます。

## ファクトリ定義とオブジェクト定義に宣言されたデータとファイル



以下の図は、上記のプログラムが2つ起動された場合を表しています。シングルスレッドプログラムでは2つのプロセスが起動され、マルチスレッドプログラムでは2つのスレッドが起動されています。

図から分かるように、マルチスレッドプログラムでは、シングルスレッドプログラムと同じくオブジェクトはプロセスで管理されます。このため、マルチスレッドプログラムでは、ファクトリオブジェクトがつねにスレッド間で共有されることになります。

図24.6 シングルスレッドプログラム

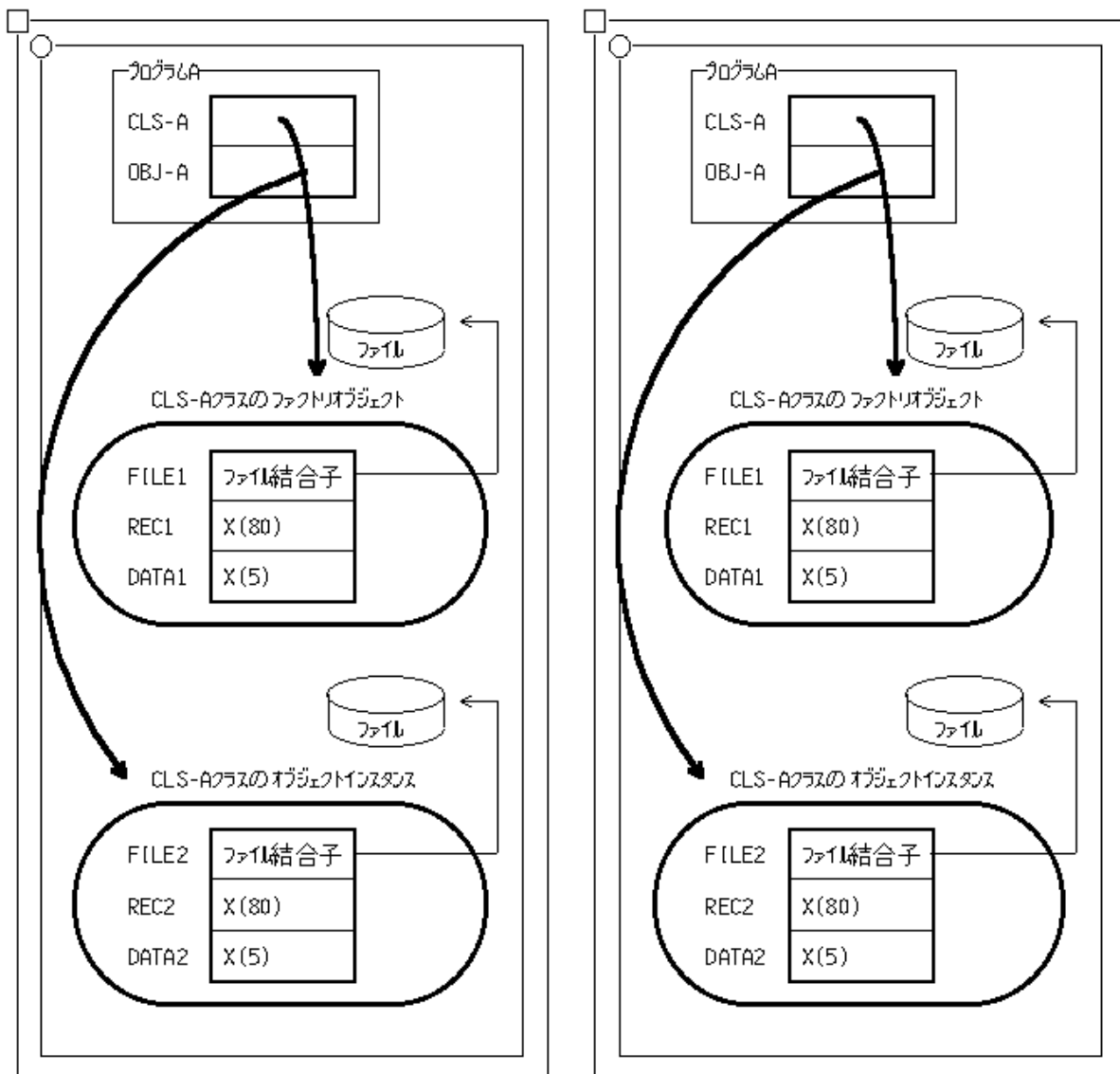
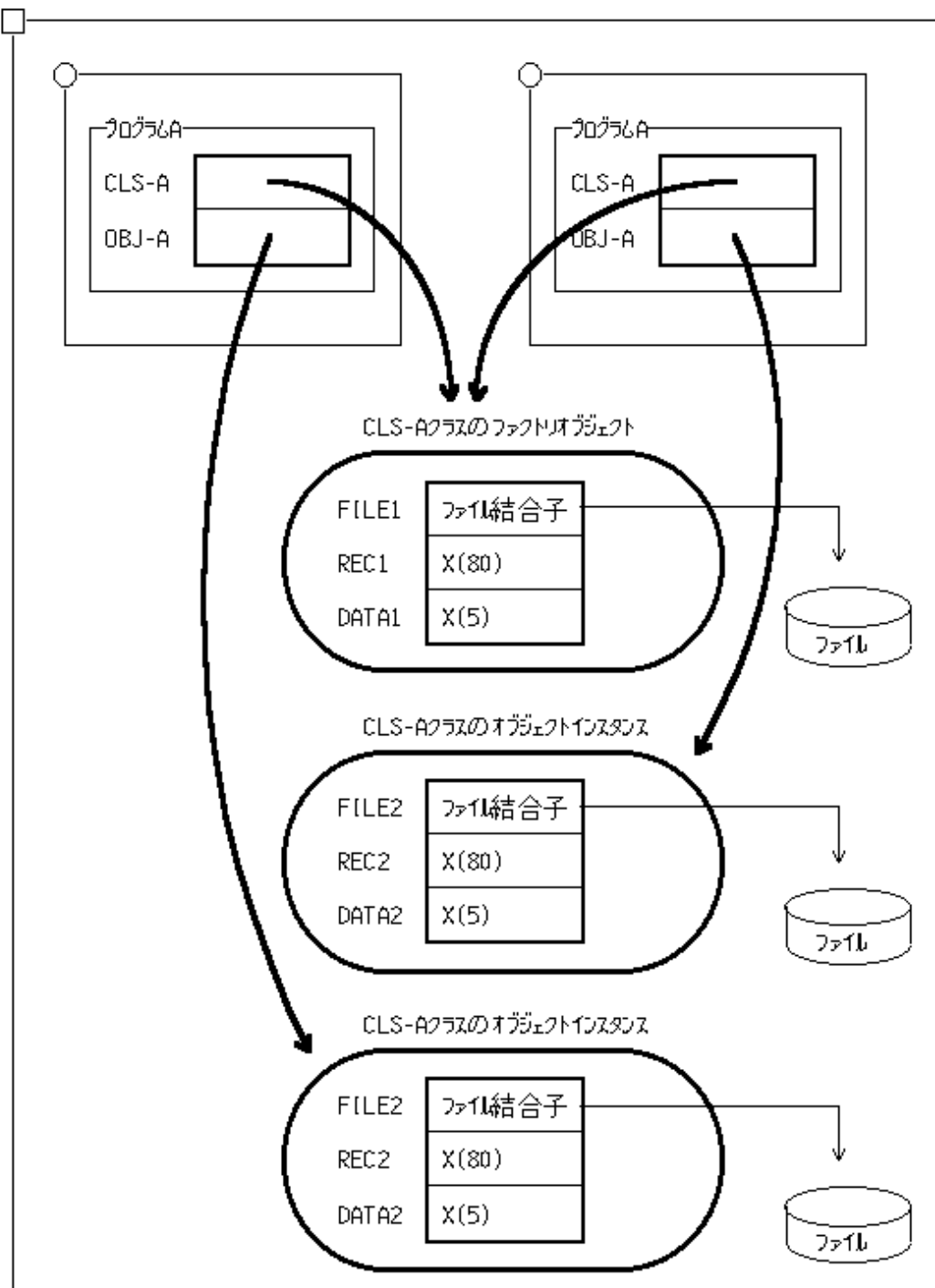


図24.7 マルチスレッドプログラム



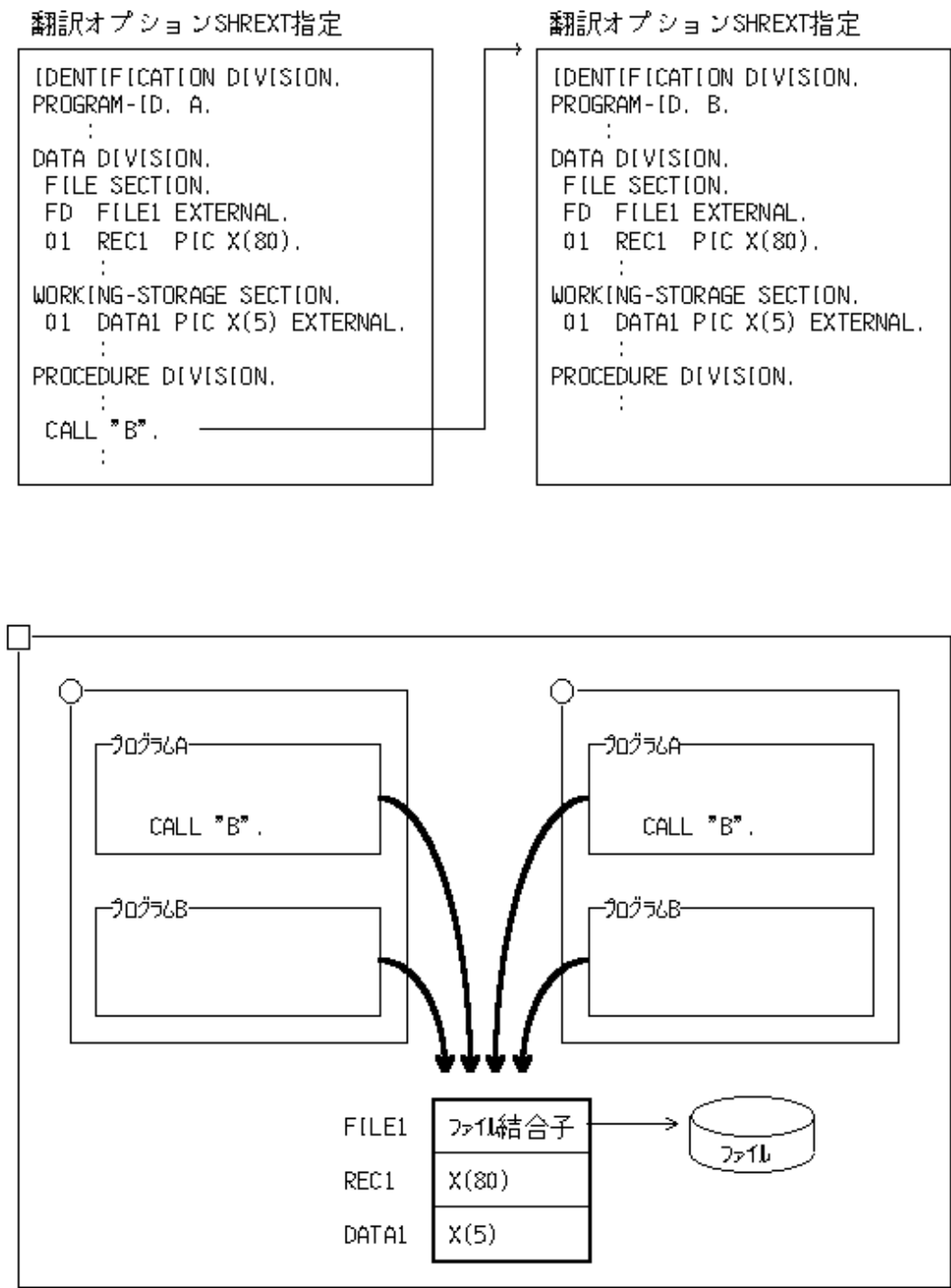
### 24.3.2.3 メソッド定義に宣言されたデータ

メソッド定義に宣言されたデータの割り付けは、シングルスレッドプログラムと変わりありません。つまり、メソッドの呼び出し時に確保され、そのメソッドの呼び出し元に戻るときに解放されます。メソッドの呼び出し元に戻るとき、クローズされていないファイルは強制的にクローズされます。

### 24.3.2.4 スレッド間共有外部データと外部ファイル

データ記述項またはファイル記述項にEXTERNAL句を指定し、翻訳オプションTHREAD(MULTI)のほかに翻訳オプションSHREXTを指定して翻訳することにより、スレッド間で共通のデータ領域を使用することができます。

以下に、データとファイルをスレッド間で共有した場合のデータ領域の持ち方を示します。この図は、マルチスレッドプログラムが2つのスレッドで実行されているところを表しています。



## 24.3.3 プログラムの実行とスレッドモード

COBOLプログラムが動作するプロセスは、スレッドモードを持ちます。スレッドモードには、シングルスレッドモードとマルチスレッドモードがあります。シングルスレッドモードのときは、プロセス内の1つのスレッドだけがCOBOLプログラムを実行できます。マルチスレッドモードのときは、プロセス内の複数のスレッドがCOBOLプログラムを実行することができます。このため、マルチスレッドモードでは、翻訳オプションTHREAD(SINGLE)で翻訳したプログラムを実行することはできません。

スレッドモードはプロセスで最初に実行されるCOBOLプログラムによって決まります。そのプログラムが翻訳オプションTHREAD(SINGLE)で翻訳したプログラムならシングルスレッドモードになり、翻訳オプションTHREAD(MULTI)で翻訳したプログラムならマルチスレッドモードになります。

### マルチスレッドでのプログラムの実行

マルチスレッドでプログラムを実行させる場合は、実行環境内のすべてのプログラムを翻訳オプションTHREAD(MULTI)で翻訳してください。

### シングル/マルチの共通部品の作成

翻訳オプションTHREAD(MULTI)で翻訳したプログラムはシングルスレッドモードでも動作できます。このため、翻訳オプションTHREAD(MULTI)で翻訳することにより、シングルスレッドでもマルチスレッドでも動作する共通部品を作成することができます。ただし、翻訳オプションTHREAD(MULTI)で翻訳されたプログラムをシングルスレッドで実行すると、翻訳オプションTHREAD(SINGLE)で翻訳されたプログラムよりも、実行性能が劣化します。また、後述の“[注意事項](#)”を必ずお読みください。

### シングルスレッドモードでの強制実行

プロセスで最初に実行されるCOBOLプログラムが、翻訳オプションTHREAD(MULTI)で翻訳したプログラムでも、環境変数情報@CBR\_THREAD\_MODE=SINGLEを指定することによって強制的にシングルスレッドモードで実行させることができます。指定方法については、“[24.7.2.2 実行環境情報の指定形式](#)”を参照してください。

### 注意事項

マルチスレッドプログラムをシングルスレッドモードで実行する場合、このプログラムをCANCEL文で削除するには注意が必要です。以下に例を示します。

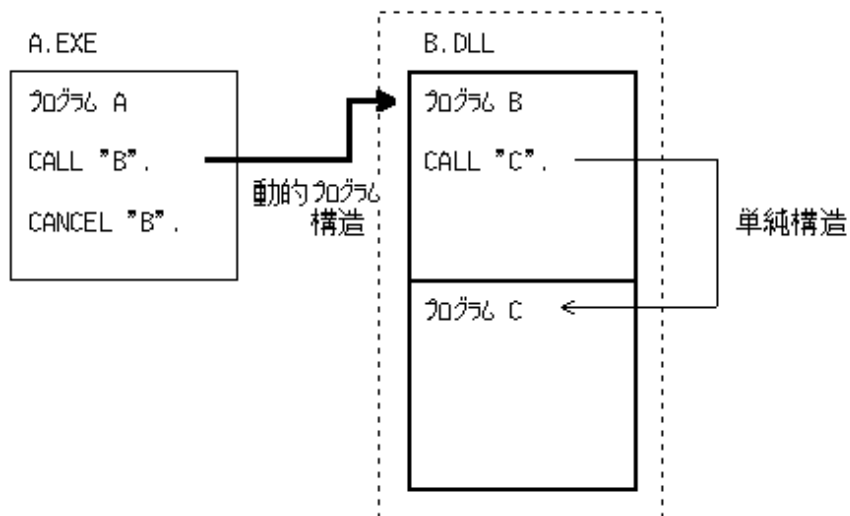


#### 例

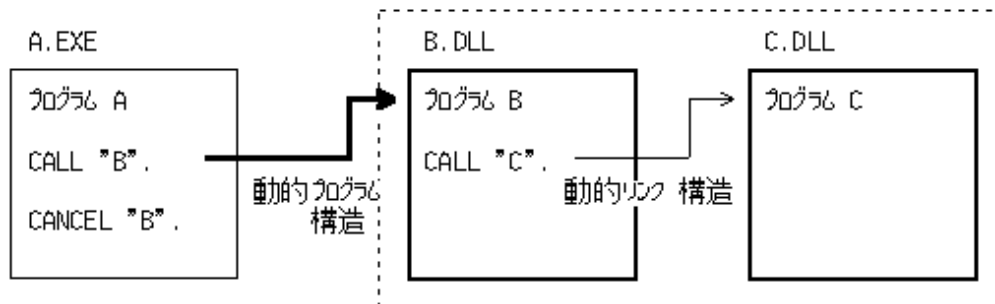
プログラムBとプログラムCを翻訳オプションTHREAD(MULTI)で翻訳します。

プログラムBとプログラムCの結合を単純構造(下図(a))または動的リンク構造(下図(b))とした場合、「CANCEL ”B”」の実行により、プログラムBが仮想メモリから削除されると、プログラムCも削除されるため、このCANCEL文は使用できません。この場合、プログラムBとプログラムCの結合を動的プログラム構造(下図(c))に変更することにより、CANCEL文を使用できるようになります。

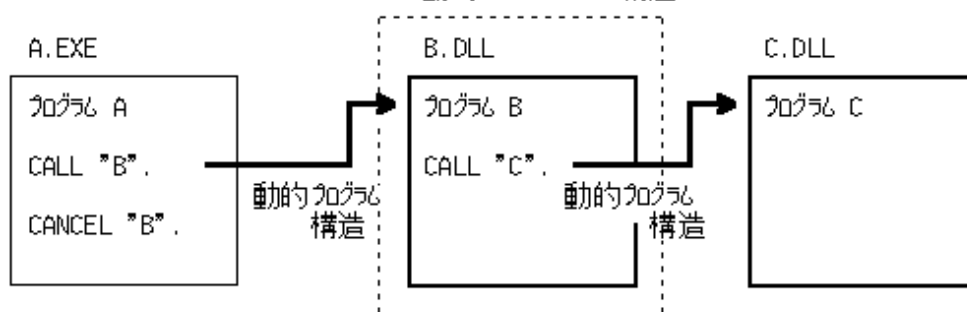
(a) プログラム B とプログラム C が単純構造



(b) プログラム B とプログラム C が動的リンク構造



(c) プログラム B とプログラム C が動的プログラム構造



□ : 翻訳オプション  
THREAD(MULTI) の有効な  
プログラム

□ : 「CANCEL "B"」の実行により  
仮想メモリから削除される  
プログラム

## 24.4 スレッド間の資源の共有

COBOLのマルチスレッドプログラムでは、スレッド間でデータやファイルなどの資源を共有するマルチスレッドの特性を活かしたプログラムを簡単に作成できます。



## 24.4.1 競合状態

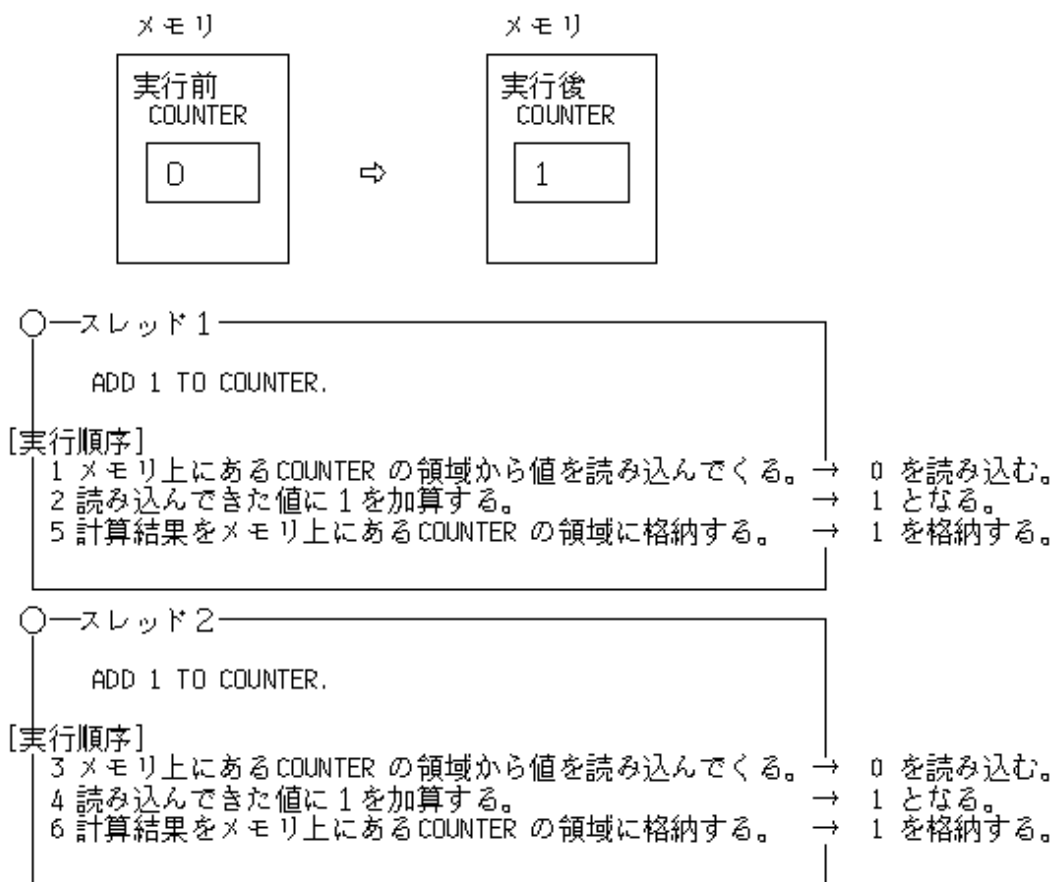
ここでは、スレッド間で資源を共有する場合に、一般的に発生する競合状態について説明します。

システムは、実行するスレッドを強制的に切り換えるため、スレッドの実行順序は予測できません。このため、スレッド間で資源を共有する場合、スレッドの実行順序が、プログラムの結果に影響を与える場合があります。これを「競合状態」と呼びます。競合状態を例で説明します。

スレッド間で共有するCOUNTERというデータに対して、“ADD 1 TO COUNTER”の文を実行する2つのスレッドがあったとします。“ADD 1 TO COUNTER”は1文ですが、コンパイラによっていくつかの機械語に展開され、システムは次のような順番で実行します。

1. メモリ上にあるCOUNTERの領域から値を読み込んでくる。
2. 読み込んできた値に1を加算する。
3. 計算結果をメモリ上にあるCOUNTERの領域に格納する。

このため、スレッド1とスレッド2の実行順序が以下のように切り替わった場合、COUNTERの値は2とならず、1となってしまいます。



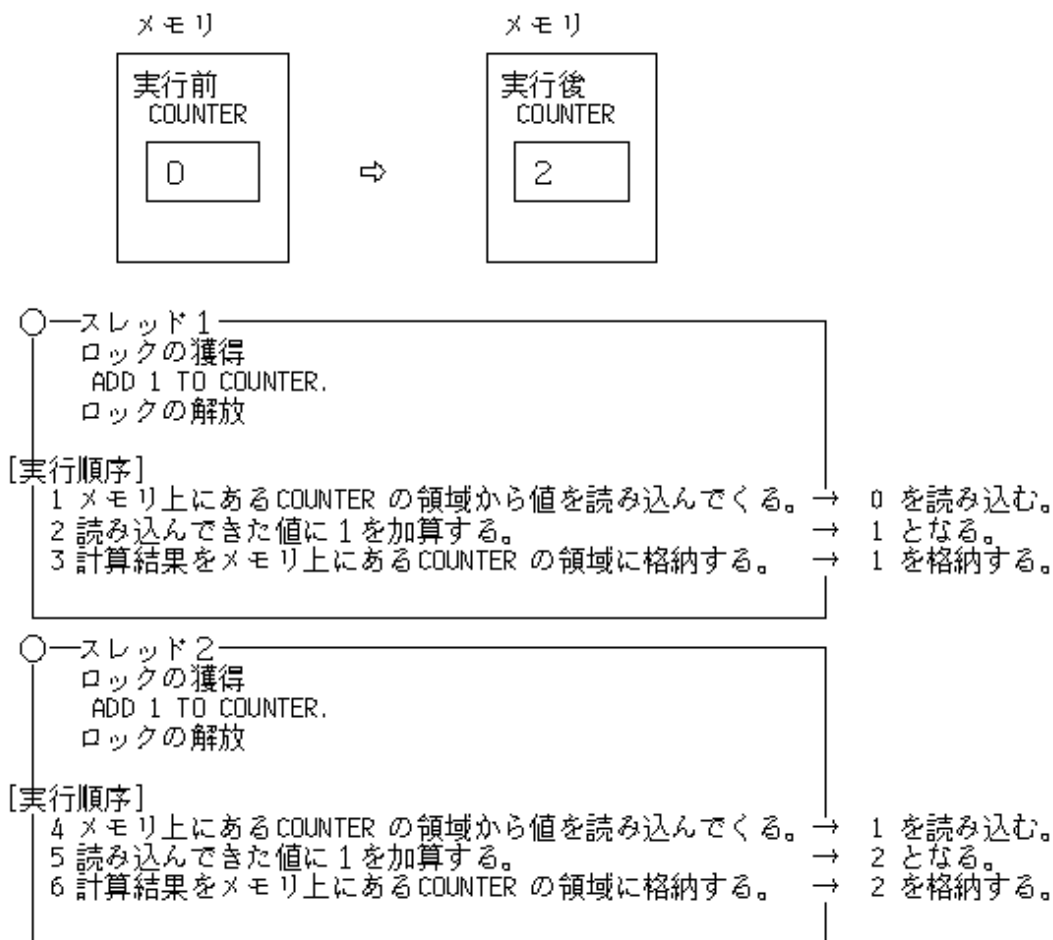
このような競合状態が発生するのは、次の条件のときです。もちろん、同一データをすべてのスレッドが参照するだけなら、同時にデータをアクセスしても問題ありません。

- ・ 同一データを更新するスレッドと参照するスレッドが同時にデータをアクセスした場合
- ・ 同一データを更新するスレッドと更新するスレッドが同時にデータをアクセスした場合

同一データに複数のスレッドが同時にアクセスしないようにするため、スレッド間で同期制御が必要となります。

これを行うための機構として、ロックがあります。プログラムを実行するために獲得しなければならない権利があり、これを「ロック」と呼びます。ロックを獲得できるスレッドは1つだけであり、ロックを獲得したスレッドだけが実行されます。ロックを獲得しようとしたほかのスレッドは、ロックを獲得しているスレッドがロックを解放するまで待つことになります。

上記の例に、ロックを使用することによって、スレッド1の実行後にスレッド2が実行されるため、COUNTERの値は2となります(この例では、スレッド1が先にロックを獲得したとします)。



## 24.4.2 資源の共有

COBOLでは、スレッド間で共有することができる資源として次のものがあります。

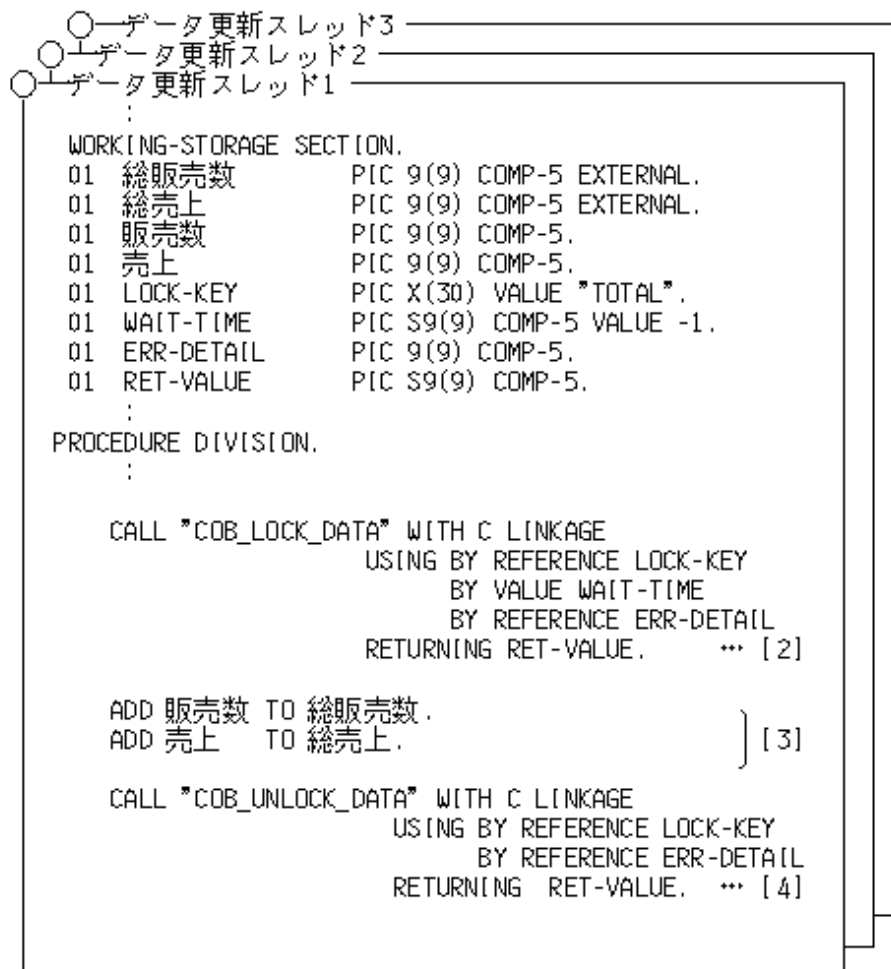
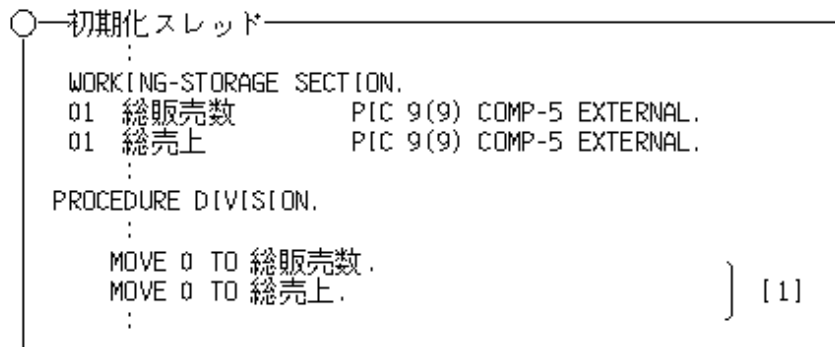
- ・ スレッド間共有外部データとスレッド間共有外部ファイル
- ・ ファクトリオブジェクト
- ・ オブジェクトインスタンス

このうち、スレッド間共有外部ファイルについては単一の入出力文単位で、ファクトリオブジェクトについてはファクトリオブジェクト単位で、COBOLランタイムシステムが自動的にスレッドの同期制御を行います。また、プログラムから直接スレッドの同期制御をするためのサブルーチンを提供しています。このサブルーチンについては、“[24.9 スレッド同期制御サブルーチン](#)”を参照してください。

### 24.4.2.1 スレッド間共有外部データと外部ファイル

データ記述項またはファイル記述項にEXTERNAL句を指定し、THREAD(MULTI)のほかに翻訳オプションSHREXTを指定して翻訳することにより、スレッド間で共通のデータ領域を使用することができます。

外部データをスレッド間で共有する例を以下に示します。外部ファイルをスレッド間で共有する場合は、“[24.6.1.1 スレッド間共有外部ファイル](#)”を参照してください。



[1] 初期化スレッドは、実行環境で最初に1回だけ起動されます。初期化スレッドで、共有データを初期化しています。

[2] 共有データを複数のスレッドで同時に更新しないようにするため、データロックサブルーチンを使用して、“TOTAL”というデータ名に対応するロックキーに対してロックを獲得しています。データロックサブルーチンについては、“[24.9.1 データロックサブルーチン](#)”を参照してください。

[3] 共有データに値を加算しています。この処理は、[2]でロックを獲得したスレッドにより実行されます。

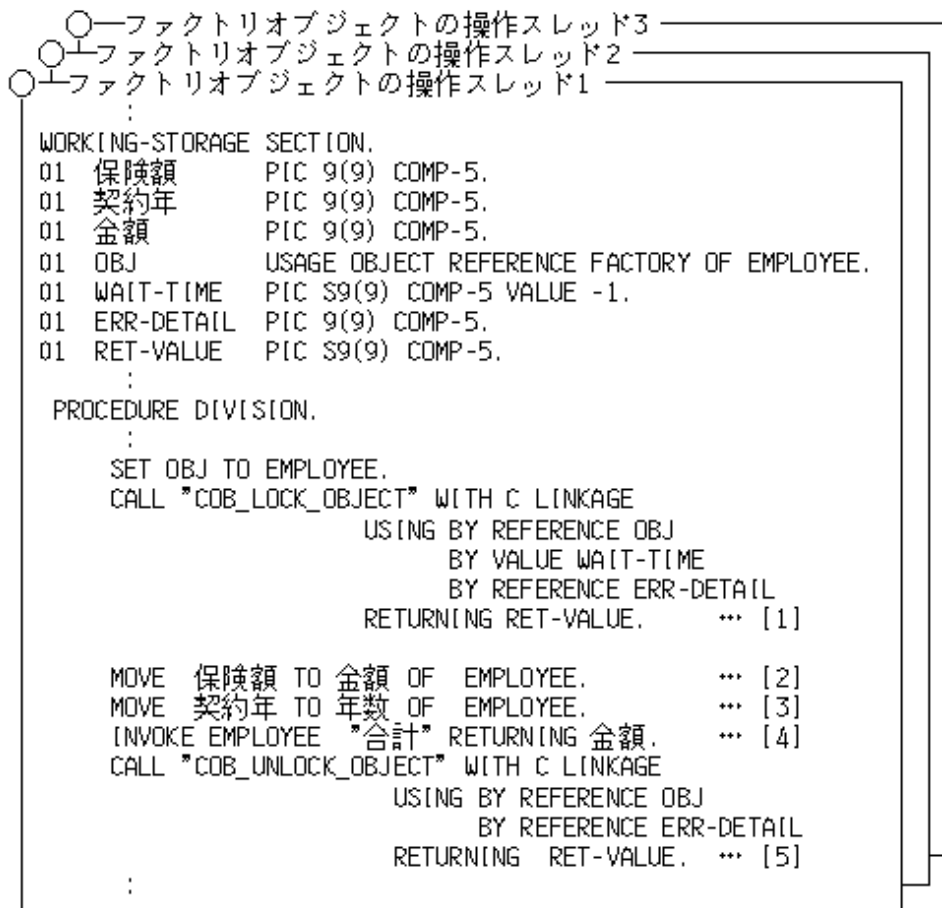
[4] “TOTAL”というデータ名に対応するロックキーに対して獲得したロックを解放しています。ロックの解放により、別のスレッドで、[2]と同様の処理でロックを獲得できます。

## 24.4.2.2 ファクトリオブジェクト

ファクトリオブジェクトはスレッド間で共有されます。このため、ファクトリデータを利用して、データやファイルをスレッド間で共有することができます。

COBOLランタイムシステムは、複数のスレッドからファクトリデータへのアクセスが同時に起こらないように、スレッドの同期制御を自動的行います(詳細については、後述の“COBOLランタイムシステムによる同期制御”参照)。このため、ファクトリメソッドの1回の呼び出しで処理が完結するような場合は、プログラムでスレッドの同期制御を行う必要はありません。

しかし、下の例のようにメソッドの複数回の呼び出しで1つの処理が完結するような場合は、オブジェクトロックサブルーチンを使用して、スレッドの同期制御を行う必要があります。オブジェクトロックサブルーチンは、オブジェクト単位で同期制御を行うことができます。オブジェクトのロックを獲得したスレッドは、ロックを解放するまで、そのオブジェクトを所有できます。



[1] EMPLOYEEクラスのファクトリデータを複数のスレッドで同時に更新しないようにするため、オブジェクトロックサブルーチンを使用して、ファクトリオブジェクトのロックを獲得します。オブジェクトロックサブルーチンについては、“[24.9.2 オブジェクトロックサブルーチン](#)”を参照してください。

[2] EMPLOYEEクラスのファクトリオブジェクトのプロパティメソッドを呼び出し、保険額をファクトリデータに設定しています。

[3] EMPLOYEEクラスのファクトリオブジェクトのプロパティメソッドを呼び出し、契約年をファクトリデータに設定しています。

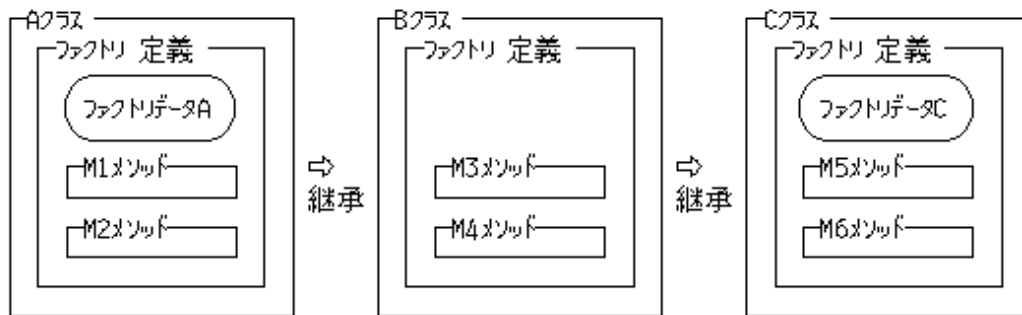
[4] EMPLOYEEクラスのファクトリオブジェクトの合計メソッドを呼び出し、金額を求めています。[2]～[4]までの処理は、[1]でロックを獲得したスレッドにより実行されます。

[5] ファクトリオブジェクトのロックを解放しています。ロックの解放により、別のスレッドで、[1]と同様の処理でロックを獲得できます。

## COBOLランタイムシステムによる同期制御

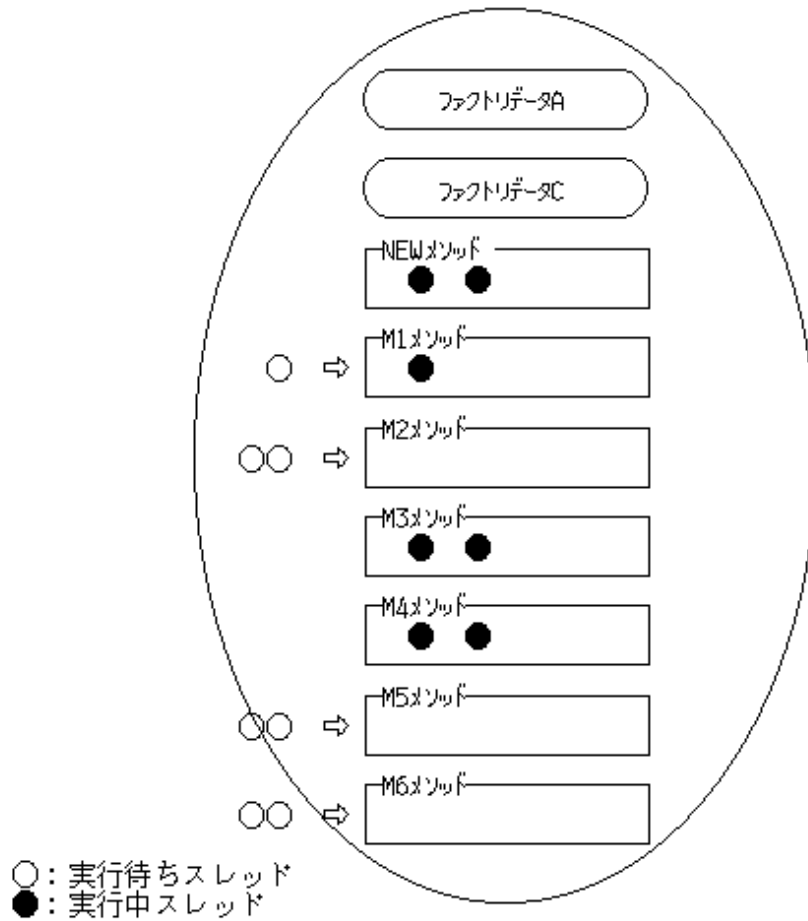
ここでは、ファクトリデータに対してCOBOLランタイムシステムが自動的に行っているスレッドの同期制御の動作について説明します。

COBOLランタイムシステムは、ファクトリデータが明示定義されているクラスのファクトリメソッドが、同一ファクトリオブジェクト上で、同時に1つのスレッドしか実行されないように制御します。この動作を下図のような継承関係にあるCクラスを例にとって動作を説明します。



Cクラスのファクトリオブジェクトは、AクラスとCクラスで明示定義されたファクトリデータを持ちます。このため、Aクラスで明示定義されているメソッド(M1メソッドとM2メソッド)とCクラスで明示定義されているメソッド(M5メソッドとM6メソッド)は、同時に1つのスレッドでしか実行されません。そのほかのメソッドは、同時に複数のスレッドで実行されます。

### Cクラスのファクトリオブジェクト



上の例は、1つのスレッドがM1メソッドを実行しています。このため、M1メソッドを実行する別スレッド、M2メソッド、M5メソッドおよびM6メソッドを実行するスレッドは、すべて実行待ち状態となります。ほかのメソッドは、同時に複数のスレッドで実行されます。このように、ファクトリデータのアクセスはCOBOLランタイムシステムによって自動的に同期制御されるため、メソッドの1回の呼び出しで処理が完結するような場合は、プログラムでスレッドの同期制御を行う必要はありません。

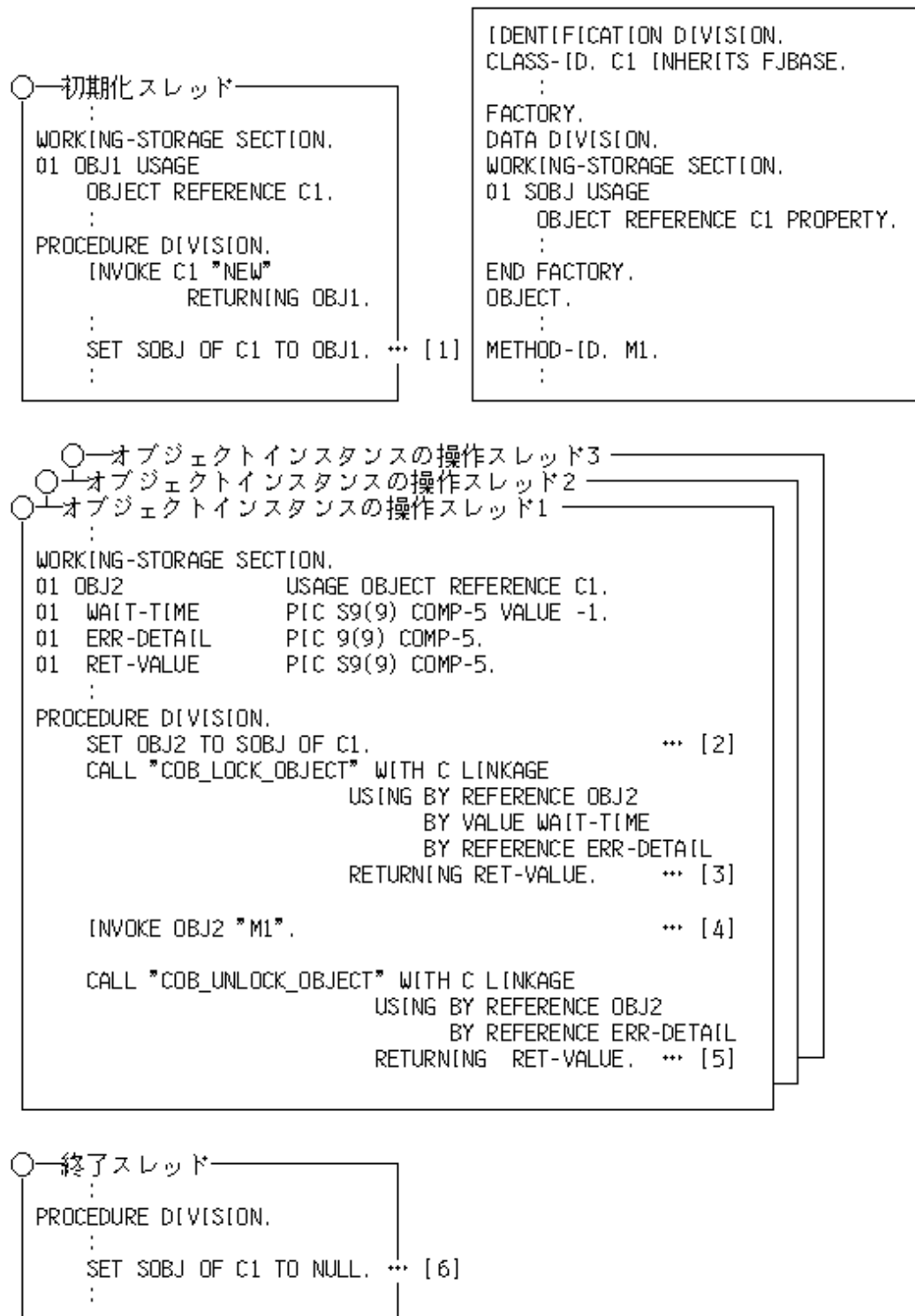
#### 参考

上記の同期制御は、ファクトリオブジェクト単位で行われます。したがって、継承しているクラスのファクトリオブジェクトでどのメソッドが実行されていようと、自クラスのファクトリオブジェクトの同期制御には影響しません。

### 24.4.2.3 オブジェクトインスタンス

ファクトリデータを介して、スレッドからスレッドへオブジェクトインスタンスのオブジェクト参照を受け渡すことにより、オブジェクトデータをスレッド間で共有することができます。COBOLランタイムシステムは、オブジェクトインスタンスに対して、同期制御を行いません。このため、オブジェクトデータを持つ場合は、オブジェクトロックサブルーチンを使用してオブジェクト単位で同期制御を行う必要があります。

ファクトリデータを介して、オブジェクトインスタンスをスレッド間で共有する例を以下に示します。



[1] 初期化スレッドは、実行環境で最初に1回だけ起動されます。初期化スレッドで、C1クラスのファクトリオブジェクトのプロパティメソッドを呼び出し、C1クラスのオブジェクトインスタンスをファクトリデータに設定しています。

[2] C1クラスのファクトリオブジェクトのプロパティメソッドを呼び出し、ファクトリデータから[1]で設定されたC1クラスのオブジェクトインスタンスを取得します。

[3] C1クラスのオブジェクトインスタンスが複数のスレッドで同時に使用されないようにするため、オブジェクトロックサブルーチンを使用して、オブジェクトインスタンスのロックを獲得します。もちろん、オブジェクトデータを持たない、またはオブジェクトデータを参照するだけであるなら、ロックする必要はありません。オブジェクトロックサブルーチンについては、“[24.9.2 オブジェクトロックサブルーチン](#)”を参照してください。

[4] C1クラスのオブジェクトインスタンスのM1メソッドを呼び出し、処理を行っています。この処理は、[3]でロックを獲得したスレッドにより実行されます。

[5] オブジェクトインスタンスのロックを解放しています。ロックの解放により、別のスレッドが[3]でロックを獲得できます。

[6] 終了スレッドは、実行環境で最後に1回だけ起動されます。終了スレッドで、C1クラスのファクトリオブジェクトのプロパティメソッドを呼び出し、ファクトリデータに設定されているオブジェクトインスタンスを削除しています。

## 24.5 基本的な使い方

---

“[24.3.2 マルチスレッドプログラムのデータの扱い](#)”から分かるように、オブジェクト指向のファクトリオブジェクトにデータまたはファイルを持たないシングルスレッドプログラムであれば、翻訳オプションTHREAD(MULTI)を指定して翻訳するだけで、マルチスレッドで実行できるプログラムになります。このため、既存のプログラムは、再翻訳するだけでマルチスレッド環境へ簡単に移行できます(翻訳とリンクについては、“[24.7.1 翻訳とリンク](#)”を参照)。

ファクトリオブジェクトにファクトリデータやファイルを持つ場合は、“[24.4 スレッド間の資源の共有](#)”を必ずお読みください。

シングルスレッドプログラムからマルチスレッドプログラムに移行するときに注意が必要となる機能について、以下に説明します。

### 24.5.1 入出力機能の利用

---

ここでは、入出力機能を利用してファイル操作を行うマルチスレッドプログラムを開発する方法について説明します。

#### マルチスレッドプログラムへの移行

シングルスレッドプログラムをマルチスレッドプログラムに移行するには、プログラムを修正する必要はありません。翻訳オプションTHREAD(MULTI)を指定してプログラムを再翻訳し、リンクするだけで、そのままマルチスレッドプログラムとして利用することができます。

以降では、同一ファイルを共有するプログラムについて説明します。

また、同一プログラムを実行し、スレッドごと、別々のファイルを操作する方法について説明します。

#### 同一ファイルを共有する

外部媒体上のファイルとプログラムとの関係付けは、ファイル結合子を通して行われます。ファイル結合子には、内部属性と外部属性を持つ2つのファイル結合子があります。内部属性/外部属性に関係なく、スレッド間で異なるファイル結合子を操作して、ファイルを共有することができます。

ここでは、内部属性を持つファイル結合子を操作して同一ファイルを共有する方法について説明します。

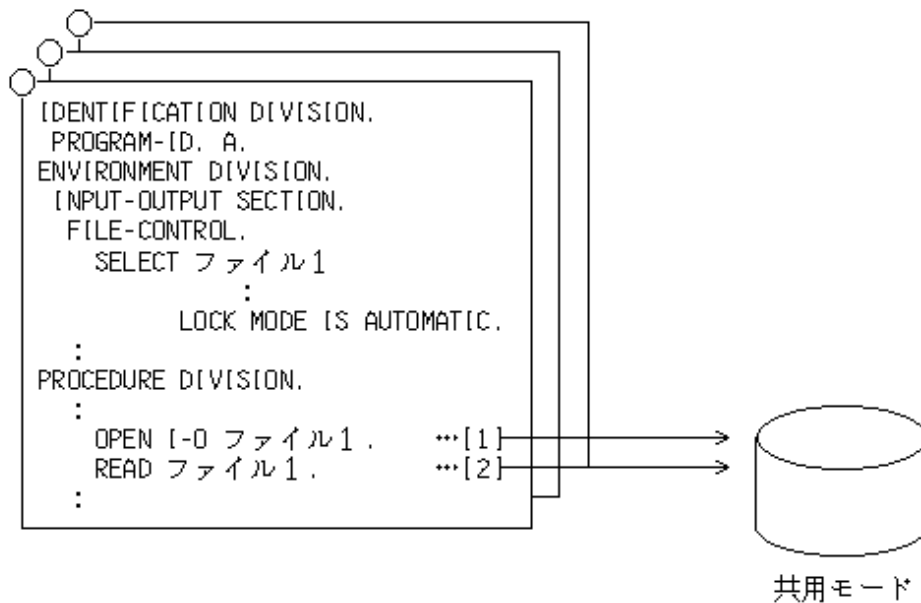
##### プログラム内に定義したファイル/メソッド内に定義したファイル/オブジェクト内に定義したファイル

プログラム内/メソッド内/オブジェクト内のファイル記述項に定義したファイルが内部属性を持つファイル結合子の場合、同一のファイルを割り当てることにより、スレッド間でファイルを共有することができます。

オブジェクト内のファイル記述項に定義したファイルの場合、別々のオブジェクトインスタンスのオブジェクト参照子を操作することで、プログラム内に定義したファイルと同様にファイルを共有することができます。

以下に、プログラム内に定義したファイルの共有処理を示します。





[1] 共用モードでファイルを開きます。

[2] レコードを読み込みます。

シングルスレッドプログラムと同様に、同一ファイル进行操作する場合は、ファイルの排他制御に従って処理してください。ファイルの排他制御については、“[7.7.2 ファイルの排他制御](#)”を参照してください。



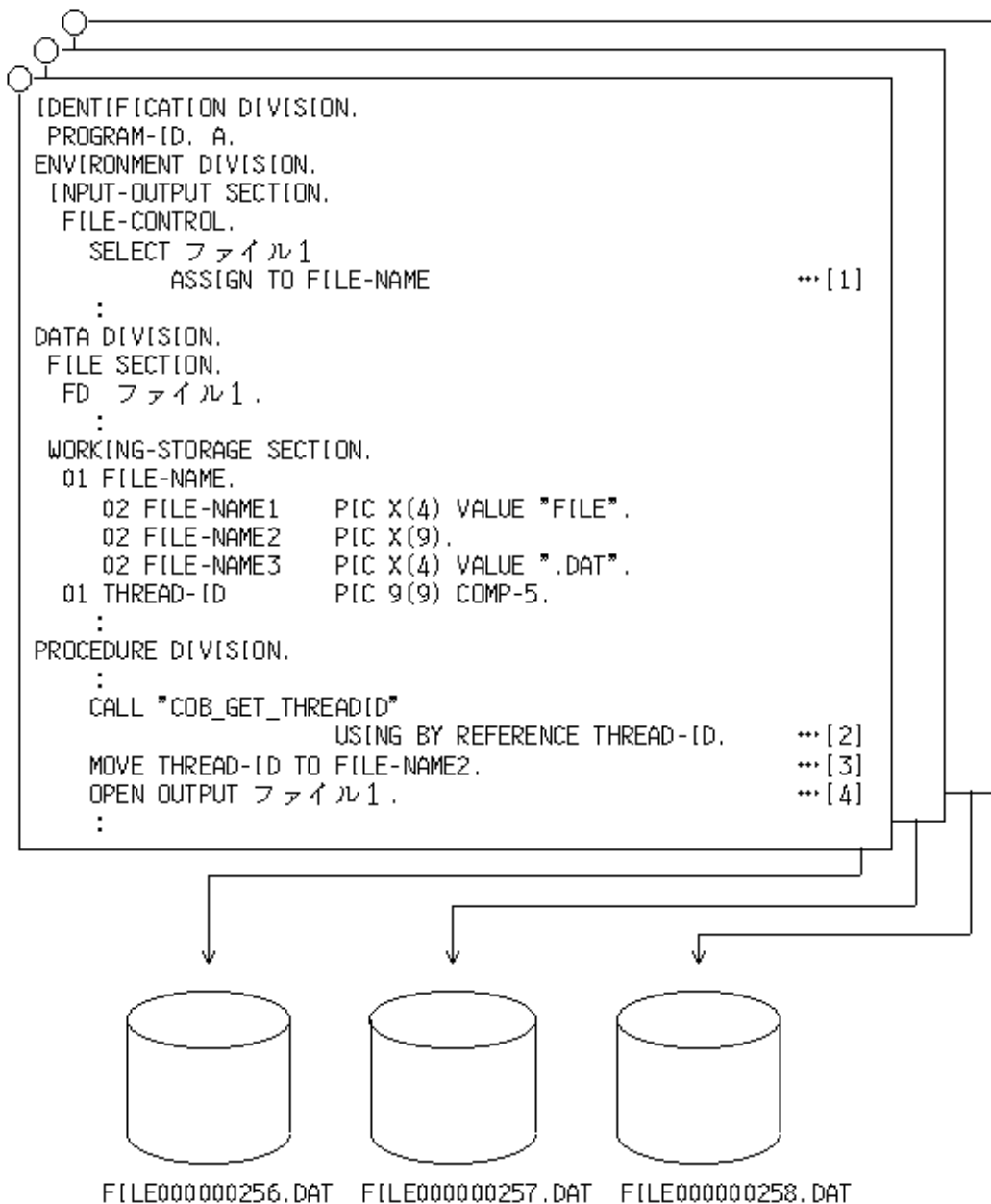
### 注意

ファクトリオブジェクト内のファイル記述項に定義したファイルは、スレッド間で共有されます。ファクトリオブジェクト内に定義したファイルについては、“[24.6 少し進んだ使い方](#)”を参照してください。

## 同一プログラムを実行し、スレッドごと、別々のファイル进行操作する

同一プログラムをマルチスレッドプログラムとして動作させる場合、基本的にはスレッド間で同一のファイル进行操作することになります。

ここでは、同一プログラムを実行して、スレッドごと、別々のファイル进行操作する方法について説明します。



- [1] ASSIGN句にデータ名を記述する。
- [2] スレッド取得サブルーチンを呼び出し、スレッドIDを取得する。
- [3] [2]で取得したスレッドIDをデータ名に設定する。
- [4] OPEN文(OUTPUTモード)を実行し、ファイルを創成する。

上記のように、スレッド取得サブルーチンを使用して取得したスレッドIDをファイル名とすることで、同一プログラムを実行して、スレッドごと、別々のファイルを操作することができます。スレッド取得サブルーチンについては、“[G.1.5 スレッドID取得サブルーチン](#)”を参照してください。



## 注意

スレッドIDは、プログラムを実行するたびに变化します。このため、スレッドIDをファイル名とする場合は、一時的な作業ファイルとして利用してください。

スレッド終了時に不用となったファイルをプログラム中から削除する場合には、COBOLファイルユーティリティ関数の削除機能を使用してください。COBOLファイルユーティリティ関数の削除機能については、“[7.8.5.8 ファイル削除関数](#)”を参照してください。

## 注意事項

1つのOPEN文で複数のファイルを指定したマルチスレッドプログラムを多重動作させる場合、ファイルの指定順序によってはデッドロック状態となる可能性があります。マルチスレッドプログラムを多重動作させる場合には、ファイルごとにOPEN文を記述するか、1つのOPEN文に記述するファイル名の指定順序を同じにしてください。

## 24.5.2 リモートデータベースアクセス(ODBC)の利用

ここでは、リモートデータベースアクセス(ODBC)を行う既存のプログラムを、マルチスレッドプログラムに移行する方法について説明します。

### マルチスレッドプログラムへの移行

シングルスレッドプログラムをマルチスレッドプログラムに移行するには、プログラムを修正する必要はありません。翻訳オプションTHREAD(MULTI)を指定して再翻訳し、リンクを行ってください。

プログラムの実行時に使用するODBC情報ファイルのコネクション有効範囲には、スレッドを指定します。[参照]“[22.2.8.1.2 ODBC情報ファイルの作成](#)”

ODBC情報ファイルは、ODBC情報設定ツールを使用して変更してください。[参照]“[22.2.8.2 ODBC情報設定ツールの使い方](#)”

コネクション有効範囲にスレッドを指定すると、スレッドごとのコネクションが保証されます。ただし、クラス定義に埋込みSQL文が記述されているアプリケーション(オブジェクト指向機能使用時)は、この環境では動作しません。[参照]“[マルチスレッドプログラムへの移行\(オブジェクト指向機能使用時\)](#)”

### マルチスレッドプログラムへの移行(オブジェクト指向機能使用時)

オブジェクト指向機能を利用したシングルスレッドプログラムをマルチスレッドプログラムに移行するには、プログラムを修正する必要はありません。翻訳オプションTHREAD(MULTI)を指定して再翻訳し、リンクを行ってください。

プログラムの実行時に使用するODBC情報ファイルのコネクション有効範囲には、オブジェクトインスタンスを指定します。[参照]“[22.2.8.1.2 ODBC情報ファイルの作成](#)”

ODBC情報ファイルは、ODBC情報設定ツールを使用して変更してください。[参照]“[22.2.8.2 ODBC情報設定ツールの使い方](#)”

コネクション有効範囲にオブジェクトインスタンスを指定すると、オブジェクトインスタンスごとのコネクションが保証されます。実行時は、プログラム定義に埋込みSQL文が記述されているアプリケーションは、この環境では動作しません。オブジェクトインスタンス単位でデータベースの接続から切断までの処理が完結していなければなりません。

## 注意事項

- 複数のスレッドが同一のデータベース表にアクセスする場合、データベースがトランザクションの一貫性を保証するために、スレッドを待ち状態にしたり、エラーを返すことがあります。

トランザクションの管理については、データベースのマニュアルを参照してください。なお、トランザクションの管理は、ODBC情報ファイルのカーソルの同時実行の指定値により、動作が異なる場合があります。[参照]“[22.2.8.1.2 ODBC情報ファイルの作成](#)”

- 一部のODBCドライバでは、データソースの設定で、マルチスレッド(スレッドセーフティ)の動作を指定できる場合があります。この場合、必ずマルチスレッド(スレッドセーフティ)を有効にしてください。
- 一部のODBCドライバでは、マルチスレッドサポートされていない場合があります。この場合、マルチスレッドプログラムは正常に動作しません。

## 24.5.3 プリコンパイラの利用によるSymfoware連携

Symfoware RDBにアクセスするマルチスレッドのプログラムは、プリコンパイラを使用して作成することができます。プリコンパイラの使用方法については、“Symfoware Server RDBユーザーズガイド 応用プログラム開発編”を参照してください。

### プログラムの記述

埋込みSQL文を記述し、データベースにアクセスするCOBOLプログラムを作成してください。マルチスレッド固有の記述は特にありません。ただし、セッションを意識したプログラムを作成する場合は、SQL拡張インタフェース(セッションの作成、破棄、開始、終了)を使用する必要があります。SQL拡張インタフェースの詳細については、“Symfoware Server RDBユーザーズガイド 応用プログラム開発編”および“Symfoware Server SQLリファレンスガイド”を参照してください。

### プログラムの翻訳・リンク

sqlcobolコマンドにより翻訳・リンクを行う場合、オプション-Tを指定してください。



#### 注意

プリコンパイル・翻訳とリンクを別に行う場合は、以下を行ってください。

- ・プリコンパイルオプション-Tを指定ください。
- ・翻訳・リンクについては“[24.7 翻訳から実行までの方法](#)”を参照してください。なお、リンク時には、Symfowareが提供するF3CWDRVM.LIBを結合してください。

### プログラムの実行

プリコンパイラを利用したSymfoware連携のマルチスレッドプログラムを動作可能にするためには、以下の環境変数情報を設定する必要があります。ただし、SQL拡張インタフェースを使用して、セッションを意識したマルチスレッドプログラムを動作させる場合は設定する必要はありません。

```
@CBR_SYMFOWARE_THREAD=MULTI
```

[参照]“[24.7.2.2 実行環境情報の指定形式](#)”の“@CBR\_SYMFOWARE\_THREAD (プリコンパイラを利用したSymfoware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)”



#### 注意

プリコンパイラを利用したSymfoware連携を行うマルチスレッドプログラム開発時の注意事項について説明します。

- ・プリコンパイル時に、マルチスレッドオプションを指定して作成したアプリケーションは、シングルスレッドで運用することはできません。

## 24.5.4 DISPLAY文およびACCEPT文の利用

ここでは、マルチスレッド環境でのDISPLAY文およびACCEPT文の使用方法について説明します。

### 24.5.4.1 小入出力機能について

マルチスレッドモードでは、COBOLのコンソールウィンドウはスレッドごとに別々に表示されます。これに対して、システムのコンソールウィンドウ(コマンドプロンプトウィンドウを含む)およびファイルを使用した小入出力では、プロセスで1つのウィンドウおよびファイルを共有します。

| 入出力先        | DISPLAY     |            | ACCEPT      |            |
|-------------|-------------|------------|-------------|------------|
|             | シングルスレッドモード | マルチスレッドモード | シングルスレッドモード | マルチスレッドモード |
| COBOLのコンソール | プロセス        | スレッドごと     | プロセス        | スレッドごと     |
| システムのコンソール  | プロセス        | プロセス       | プロセス        | プロセス       |

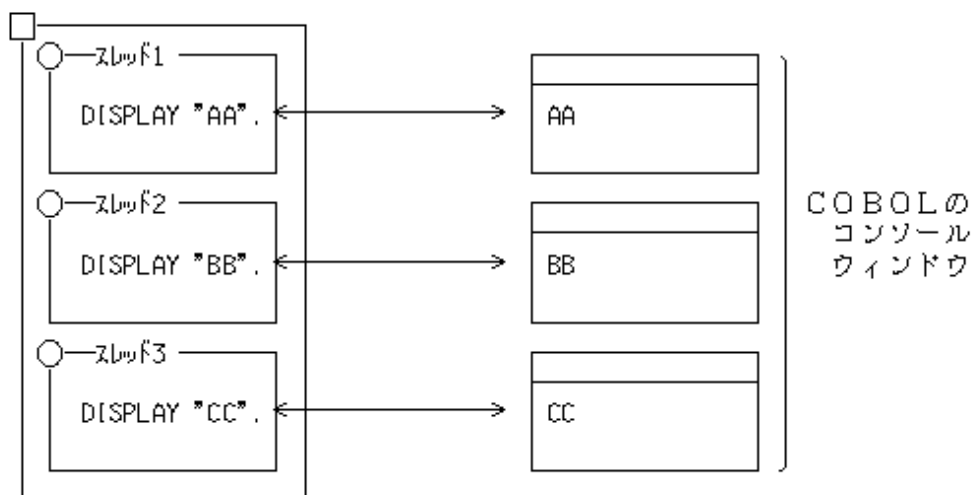
| 入出力先 | DISPLAY     |            | ACCEPT      |            |
|------|-------------|------------|-------------|------------|
|      | シングルスレッドモード | マルチスレッドモード | シングルスレッドモード | マルチスレッドモード |
| ファイル | プロセス        | プロセス       | プロセス        | プロセス       |



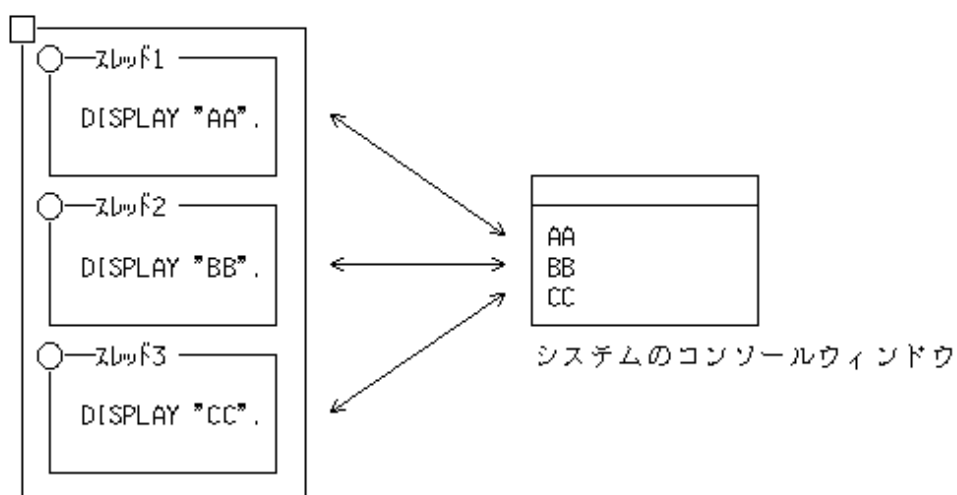
## 例

入出力の例

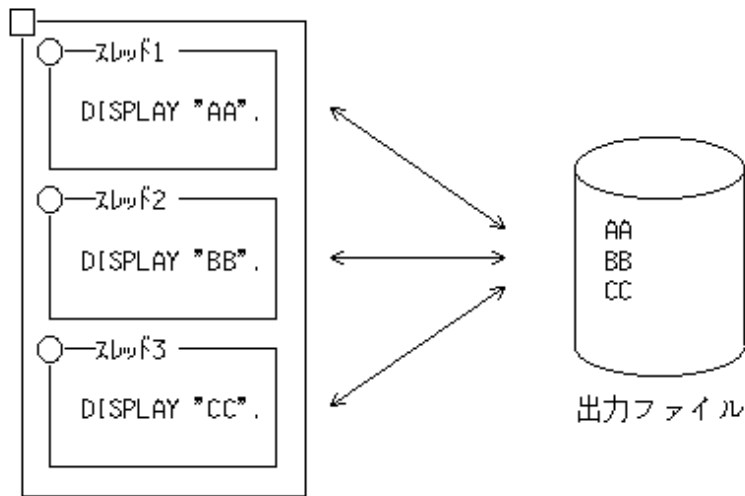
- COBOLのコンソールウィンドウを使用する場合



- システムのコンソールウィンドウを使用する場合



- ・ ファイルを使用する場合



システムのコンソールウィンドウおよびファイルを使用した小入出力を行う場合、DISPLAY文およびACCEPT文単位でデータの入出力は同期制御されます。

しかし、文単位での同期制御は行いますが、各文の実行順序については、システムのスレッド制御の順序に依存するため、結果が実行のたびに異なることがあります。

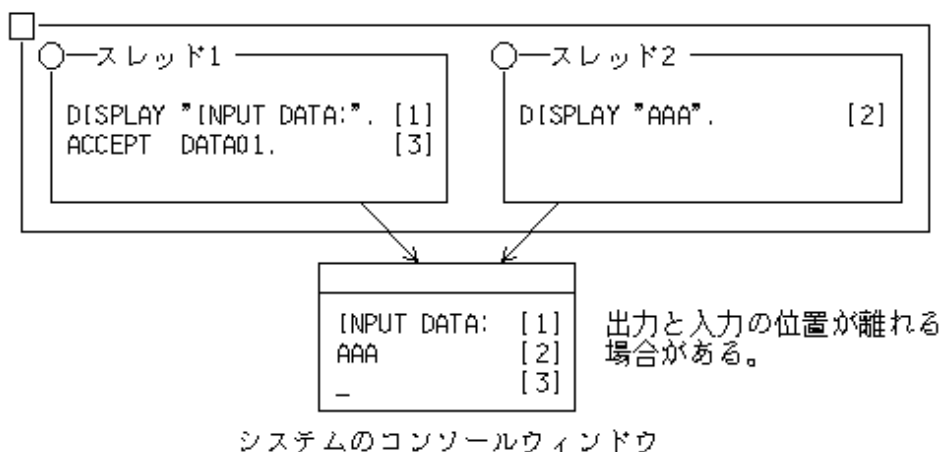
実行順序の同期制御を行いたい場合、たとえば、システムのコンソールウィンドウを使用する際に、DISPLAY文の直後にACCEPT文を実行したい場合は、スレッド同期制御サブルーチンを使用することで、DISPLAY文の出力の直後にACCEPT文を実行できます。[参照]“24.9 スレッド同期制御サブルーチン”



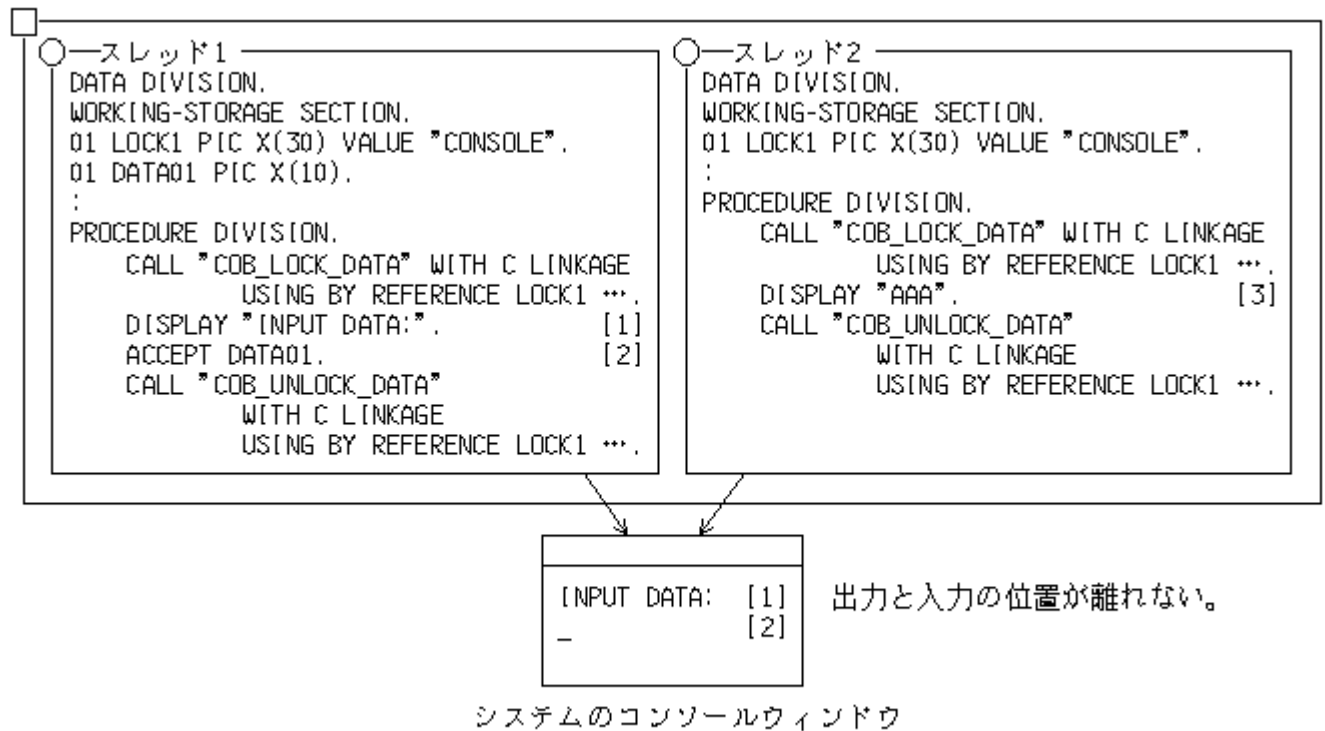
## 例

複数の入出力文の同期制御([1]～[3];実行順序)

- ・ スレッド同期制御サブルーチンを使用しない場合



- ・ スレッド同期制御サブルーチンを使用した場合



## 注意

- ・ DISPLAY文およびACCEPT文で、コンソール画面とファイルのどちらを使用するか、また、ファイルを使用する場合はどのファイル識別名が有効となるかは、最初に動作したDISPLAY文およびACCEPT文を含むオブジェクトの翻訳オプションに依存します。このため、翻訳時の翻訳オプションは可能な限りそろえてください。
- ・ マルチスレッドモードでは、COBOLコンソール画面の「×(閉じる)」ボタンを押下すると、プロセスを終了します。
- ・ マルチスレッド環境において、システムの標準出力と標準エラー出力をリダイレクションの指定により同じファイルに出力した場合、出力されるファイルの内容は保証されません。ランタイムシステムが出力する実行時メッセージおよび機能名SYSERRに対応付けられたDISPLAY文の結果を任意のファイルに出力する場合は、環境変数情報@MessOutFileに出力するファイルを指定してください。

## 24.5.4.2 コマンド行引数および環境変数の操作機能について

### 24.5.4.2.1 コマンド行引数の操作機能

コマンド行引数の操作機能で使用する引数の位置は、スレッドごとに持ちます。このため、複数のスレッドでコマンド行引数操作を行っても、別スレッドで行っている操作には影響しません。

```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
 ARGUMENT-NUMBER IS ARGNUM . . . [1]
 ARGUMENT-VALUE IS ARGVAL. . . . [2]
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DATA01 PIC X(10).

```

```
PROCEDURE DIVISION.
 DISPLAY 3 UPON ARGNUM. . . . [1]
 ACCEPT DATA01 FROM ARGVAL. . . . [2]
```

[1] 引数の位置は、スレッドごとに持ちます。

[2] 引数の値は、プロセスで共通です。

#### 24.5.4.2.2 環境変数の操作機能

環境変数の操作機能で使用する環境変数名は、スレッドごとに持ちます。このため、特殊名段落のENVIRONMENT-NAMEに割り当てる環境変数名については、複数のスレッドで別々のものを使用しても問題ありません。ただし、環境変数値はプロセス内で共通のため、マルチスレッドモードで環境変数操作を行うと、別スレッドに影響があります。

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
 ENVIRONMENT-NAME IS ENVNAME . . . [1]
 ENVIRONMENT-VALUE IS ENVVAL. . . . [2]
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DATA01 PIC X(10).
PROCEDURE DIVISION.
 DISPLAY "ABC" UPON ENVNAME. . . . [1]
 ACCEPT DATA01 FROM ENVVAL. . . . [2]
```

[1] 環境変数名は、スレッドごとに持ちます。

[2] 環境変数値は、プロセスで共通です。

### 24.5.5 スクリーン機能の利用

マルチスレッドモードでのスクリーン機能は、スレッドごとにウィンドウを持ちます。

スクリーンウィンドウ上の「×(閉じる)」ボタンが押下された場合は、プロセスを終了します。

## 24.6 少し進んだ使い方

### 24.6.1 入出力機能の利用

基本的な使い方では、スレッド間で異なるファイル結合子を操作したファイルの共有について説明しました。

ここでは、スレッド間で同じファイル結合子を操作してファイルを共有する方法について説明します。

スレッド間で同じファイル結合子を操作するには、以下の方法があります。

- ・ スレッド間共有外部ファイル
- ・ ファクトリオブジェクト内に定義したファイル
- ・ オブジェクト内に定義したファイル



注意

RDMファイルは、スレッド間で同じファイル結合子を操作することはできません。

各ファイルの利用方法について、以下で説明します。

なお、同じファイル結合子を操作してファイルを共有する場合、スレッドの競合状態が発生します。スレッドの競合状態を防ぐために、スレッドの同期制御が必要になります。これについては、各ファイルの利用方法で説明します。





## 注意

同一ファイル結合子を使用して1つのファイルにアクセスする場合、入出力文の実行によりファイル位置指示子の状態が変化します。そのため、マルチスレッドで動作する場合、ファイル位置指示子の状態を意識したプログラムを設計する必要があります。

### 24.6.1.1 スレッド間共有外部ファイル

ファイル記述項にEXTERNAL句を指定した場合、ファイル結合子に外部属性が与えられます。外部属性を持つファイル結合子は、プログラム間で同じファイル結合子を共有することができます。

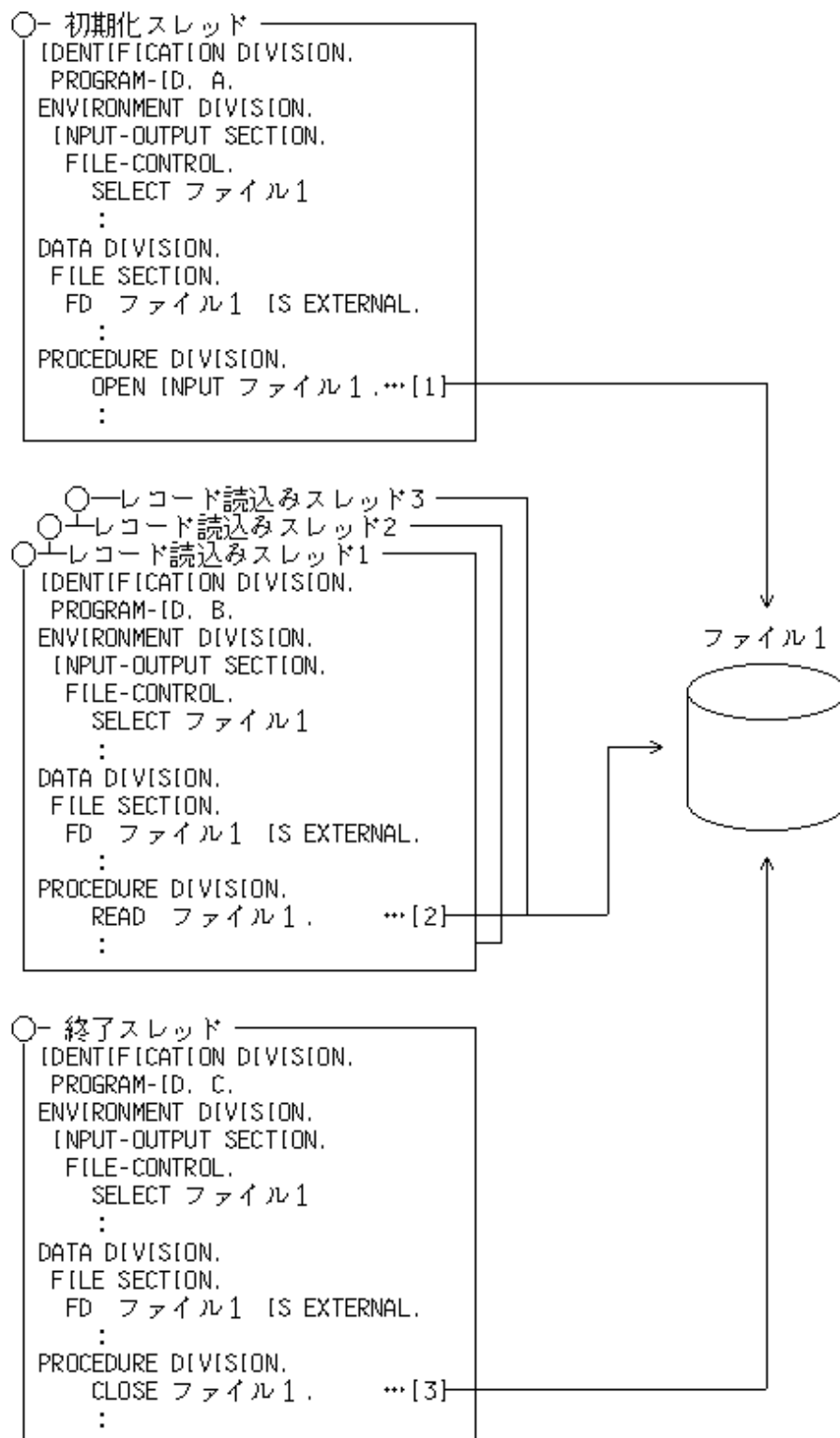
複数のスレッドの間で同じファイル結合子を共有する場合は、翻訳オプションTHREAD(MULTI)のほかに翻訳オプションSHREXTを指定します。

なお、EXTERNAL句は、メソッド内のファイル記述項にも指定することができます。



## 例

スレッド間共有外部ファイルを使用したプログラム例



[1] 初期化スレッドは、実行環境で最初に1回だけ起動されます。初期化スレッドでは、外部属性のファイル結合子を持つファイルを開きます。

[2] レコード読みスレッドは、複数(プログラム例では3つ)同時に起動されます。レコード読みスレッドでは、初期化スレッドによって開かれているファイルに対し、レコードを読み込みます。

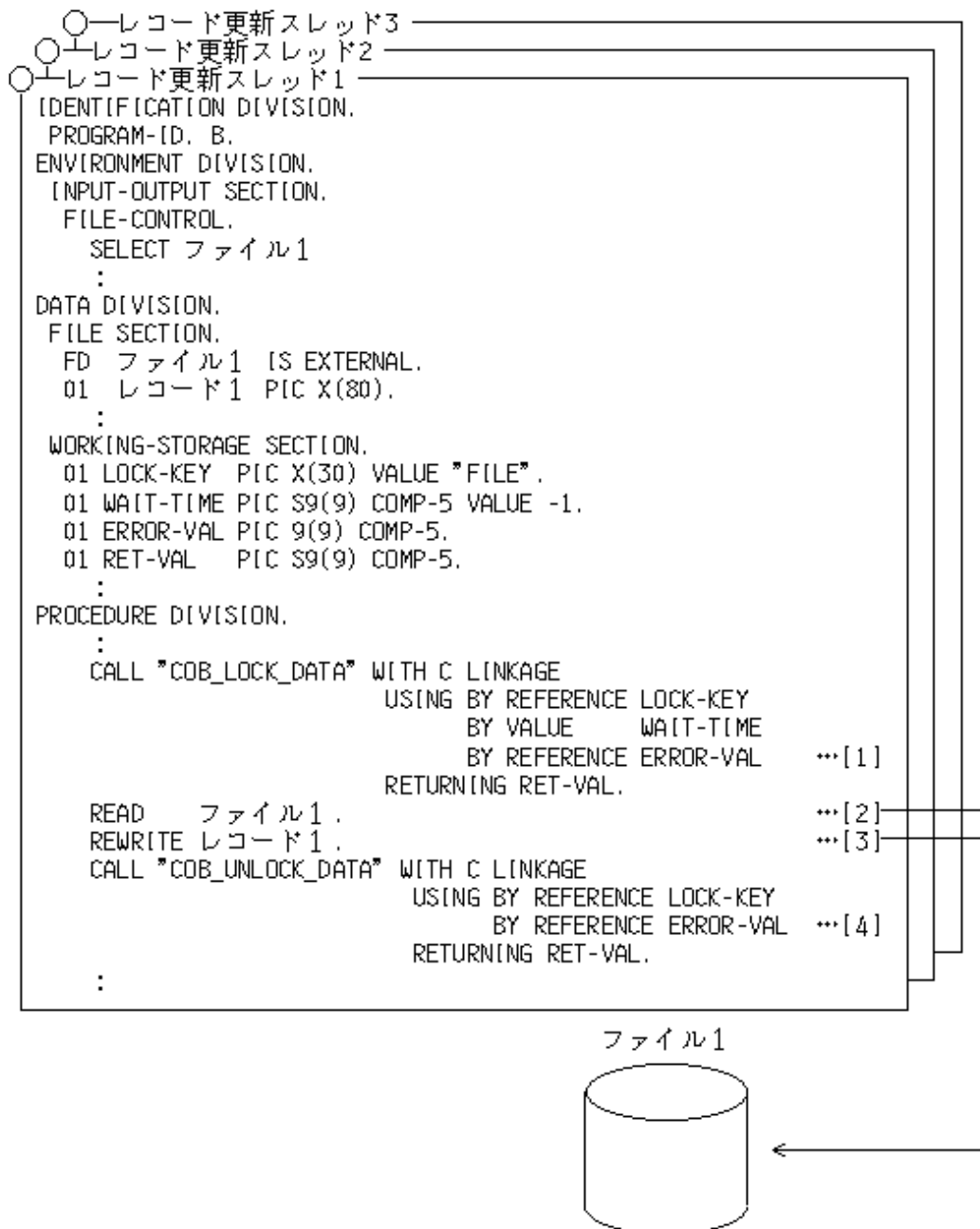
[3] 終了スレッドは、実行環境で最後に1回だけ起動されます。終了スレッドでは、初期化スレッドによって開かれているファイルを閉じます。

外部ファイルの場合、単一の入出力文については、COBOLランタイムシステムで自動的にスレッドの同期制御を行います。しかし、複数の入出力文の実行で、意図した処理を行う場合、スレッドの同期制御が必要になります。外部ファイルへの同期制御を行うには、データロックサブルーチン・データアンロックサブルーチンを使用します。このサブルーチンを使用することにより、[2]のREAD文と[3]のREWRITE文の間に、ほかのスレッドで別のREAD文が実行されることを防止することができます。



## 例

複数の入出力文に対するスレッドの同期制御のプログラム例



- [1] データロックサブルーチンにより、“FILE”というデータ名に対応するロックキーに対してロックを獲得します。
  - [2] ファイル1のレコードを読み込みます。
  - [3] [2]で読み込んだレコードを更新します。
  - [4] データロックサブルーチンにより、“FILE”というデータ名に対応するロックキーに対して獲得したロックを解放します。
- .....



#### 参照

.....

データロックサブルーチンについては、“[24.9.1 データロックサブルーチン](#)”を参照してください。

.....

### 24.6.1.2 ファクトリオブジェクト内に定義したファイル

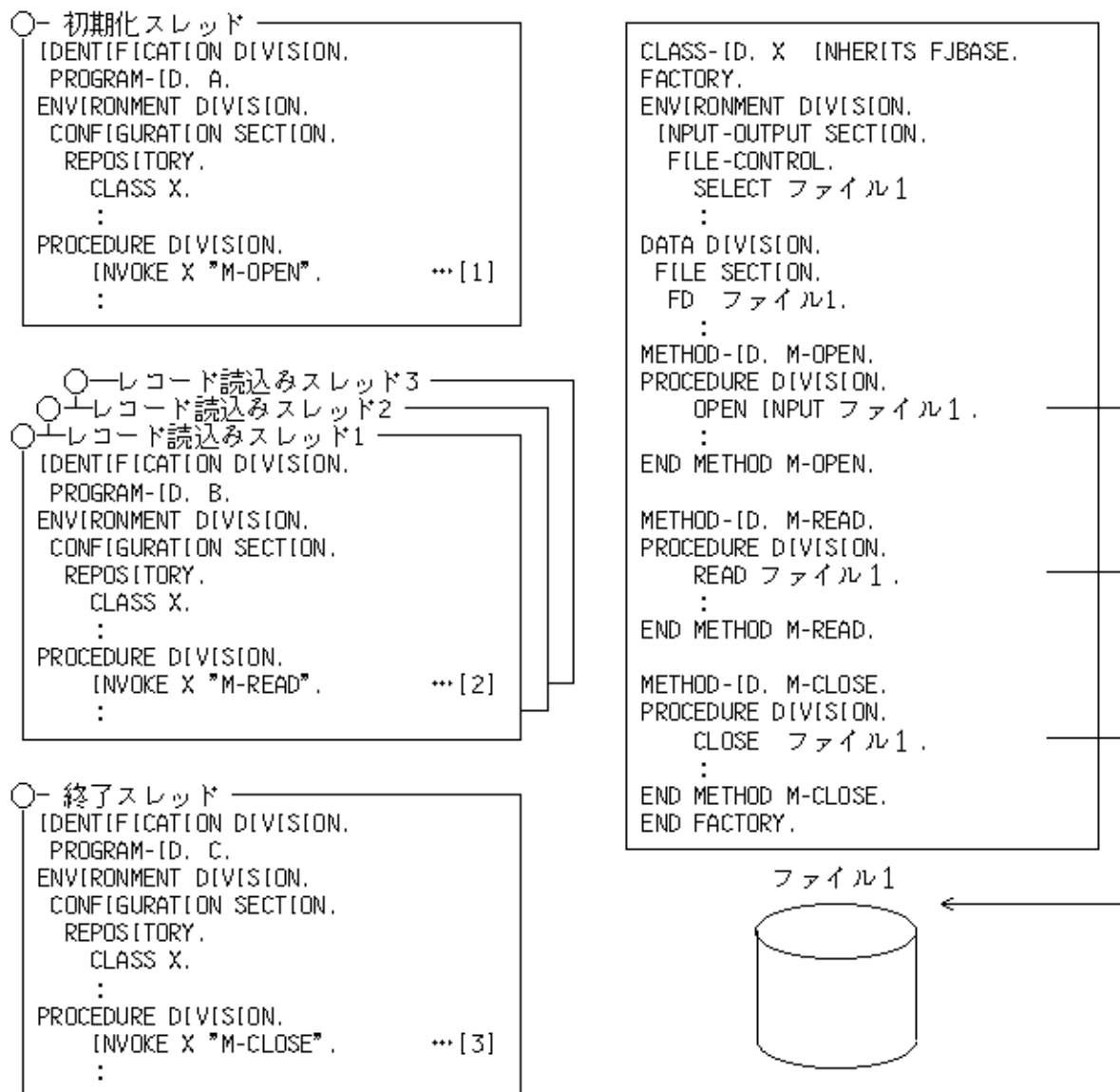
ファクトリオブジェクト内のファイル記述項に定義したファイルは、外部ファイルと同様に同じファイル結合子を共有することができます。



#### 例

.....

ファクトリオブジェクト内のファイルを使用したプログラム例



[1] 初期化スレッドは、実行環境で最初に1回だけ起動されます。初期化スレッドで、ファクトリメソッド“M-OPEN”を呼び出し、ファイルを開きます。

[2] レコード読みスレッドは、複数(プログラム例では3つ)同時に起動されます。ファクトリメソッド“M-READ”を呼び出し、初期化スレッドによって開かれているファイルに対し、レコードを読み込みます。

[3] 終了スレッドは、実行環境で最後に1回だけ起動されます。終了スレッドで、ファクトリメソッド“M-CLOSE”を呼び出し、初期化スレッドによって開かれているファイルを閉じます。

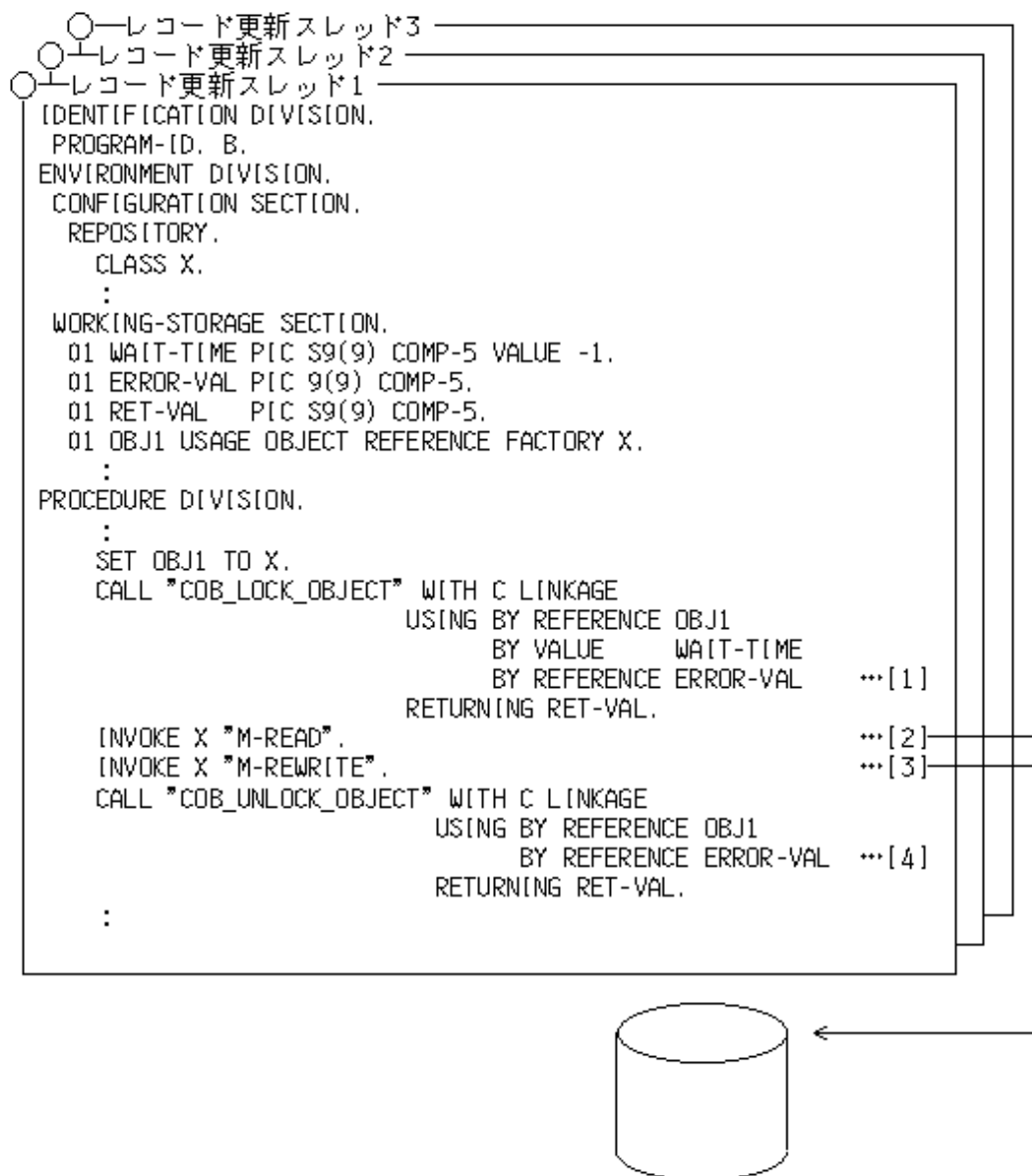
.....

ファクトリメソッドの1回の呼び出しで処理が完結する場合は、COBOLランタイムシステムで自動的にスレッドの同期制御を行います。しかし、ファクトリメソッドの複数回呼び出しで、意図した処理を行いたい場合は、スレッドの同期制御を行う必要があります。

#### 例

.....

ファクトリメソッドの複数回呼び出しに対するスレッドの同期制御のプログラム例



- [1] オブジェクトロックサブルーチンにより、ファクトリオブジェクトのロックを獲得します。
- [2] ファクトリメソッド"M-READ"を呼び出し、ファイル1のレコードを読み込みます。
- [3] ファクトリメソッド"M-REWRITE"を呼び出し、[2]で読み込んだレコードを更新します。
- [4] オブジェクトロックサブルーチンにより、ファクトリオブジェクトのロックを解放します。

### 参照

オブジェクトロックサブルーチンについては、“[24.9.2 オブジェクトロックサブルーチン](#)”を参照してください。

## 24.6.1.3 オブジェクト内に定義したファイル

オブジェクト内のファイル記述項に定義したファイルは、1つのオブジェクトインスタンスを共有することで、外部ファイルと同様に同じファイル結合子を共有することができます。

COBOLランタイムシステムは、オブジェクトインスタンスに対して、スレッドの同期制御は行いません。このため、1つのオブジェクトインスタンスを共有してオブジェクト内のファイル进行操作する場合、スレッドの同期制御を行う必要があります。

オブジェクト内のファイルに対するスレッド間の同期制御は、オブジェクトロックサブルーチンを使用して行います。



例

オブジェクト内のファイルを使用したプログラム例

○—初期化スレッド

```
IDENTIFICATION DIVISION.
PROGRAM-ID. A.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
CLASS X.
:
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ1 USAGE OBJECT REFERENCE X.
:
PROCEDURE DIVISION.
 INVOKE X "NEW" RETURNING OBJ1. ...[1]
 INVOKE OBJ1 "M-OPEN". ...[2]
 SET FOBJ OF X TO OBJ1. ...[3]
 :
```

○—レコード読みスレッド3

○—レコード読みスレッド2

○—レコード読みスレッド1

```
IDENTIFICATION DIVISION.
PROGRAM-ID. B.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
CLASS X.
:
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ2 USAGE OBJECT REFERENCE X.
01 WAIT-TIME PIC S9(9) COMP-5 VALUE -1.
01 ERR-VAL PIC 9(9) COMP-5.
01 RTN-VAL PIC S9(9) COMP-5.
:
PROCEDURE DIVISION.

 SET OBJ2 TO FOBJ OF X. ...[4]
 CALL "COB_LOCK_OBJECT"
 WITH C LINKAGE
 USING BY REFERENCE OBJ2
 BY VALUE WAIT-TIME
 BY REFERENCE ERR-VAL
 RETURNING RET-VAL. ...[5]
 INVOKE OBJ2 "M-READ". ...[6]
 CALL "COB_UNLOCK_OBJECT"
 WITH C LINKAGE
 USING BY REFERENCE OBJ2
 BY REFERENCE ERR-VAL
 RETURNING RET-VAL. ...[7]
 :
```

○—終了スレッド

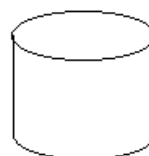
```
IDENTIFICATION DIVISION.
PROGRAM-ID. C.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
CLASS X.
:
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ3 USAGE OBJECT REFERENCE X.
:
PROCEDURE DIVISION.
 SET OBJ3 TO FOBJ OF X. ...[8]
 INVOKE OBJ3 "M-CLOSE". ...[9]
 :
```

```
CLASS-ID. X INHERITS FJBASE.
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 FOBJ USAGE OBJECT
 REFERENCE X PROPERTY.
:
END FACTORY.
OBJECT.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
 SELECT ファイル1
 :
DATA DIVISION.
FILE SECTION.
 FD ファイル1.
 :
METHOD-ID. M-OPEN.
PROCEDURE DIVISION.
 OPEN INPUT ファイル1.
 :
END METHOD M-OPEN.

METHOD-ID. M-READ.
PROCEDURE DIVISION.
 READ ファイル1.
 :
END METHOD M-READ.

METHOD-ID. M-CLOSE.
PROCEDURE DIVISION.
 CLOSE ファイル1.
 :
END METHOD M-CLOSE.
```

ファイル1





初期化スレッド([1]～[3])は、実行環境で最初に1回だけ起動されます。

- [1] クラス'X'のオブジェクトインスタンスを獲得します。
- [2] メソッド'M-OPEN'を呼び出し、ファイルを開きます。
- [3] PROPERTY句を指定したファクトリデータ'FOBJ'に、オブジェクトインスタンスを代入します。

レコード読み込みスレッド([4]～[7])は、複数(プログラム例では3つ)同時に起動されます。

- [4] オブジェクトインスタンス'OBJ2'にファクトリデータ'FOBJ'を代入します。これにより、プログラムAで使ったオブジェクトインスタンスを共有することができます。
- [5] オブジェクトロックサブルーチンにより、オブジェクトインスタンスのロックを獲得します。
- [6] メソッド'M-READ'を呼び出し、プログラム'A'で開いたファイルのレコードを読み込みます。
- [7] オブジェクトロックサブルーチンにより、オブジェクトインスタンスのロックを解放します。

終了スレッド([8]～[9])は、実行環境で最後に1回だけ起動されます。

- [8] オブジェクトインスタンス'OBJ3'にファクトリデータ'FOBJ'を代入します。これにより、プログラムAで使ったオブジェクトインスタンスを共有することができます。
- [9] メソッド'M-CLOSE'を呼び出し、初期化スレッドで開いたファイルを閉じます。



## 参照

オブジェクトロックサブルーチンについては、“[24.9.2 オブジェクトロックサブルーチン](#)”を参照してください。

## 24.6.2 リモートデータベースアクセス(ODBC)の利用

基本的な使い方では、シングルスレッドプログラムをそのままマルチスレッドプログラムに移行しました。

ここでは、マルチスレッドの利点を活かしたマルチスレッドプログラムを新規に構築する方法など、少し進んだ使い方について説明します。

### 24.6.2.1 コネクションをスレッド間で共有する

コネクションをスレッド間で共有することにより、負荷の大きい接続および切断処理をそれぞれ初期化スレッドおよび終了スレッドに分離したり、データ操作処理を複数のスレッドで行うことが可能になります。これにより、アプリケーション性能が向上します。ただし、コネクションをスレッド間で共有して利用する場合、以下の点に注意する必要があります。

複数のスレッドで1つのコネクションを利用する場合、トランザクションも共有することになります。これは、あるスレッドでトランザクション処理を行った場合、コネクションを共有するすべてのスレッドのデータ操作に影響することを意味します。データベース表を変更しないマルチスレッドプログラムを動作させる場合は問題ありませんが、データベース表を変更するマルチスレッドプログラムを動作させる場合は、同時に実行されるスレッドがそれぞれ別々のコネクションを利用してトランザクション処理を行うようにプログラミングする必要があります。

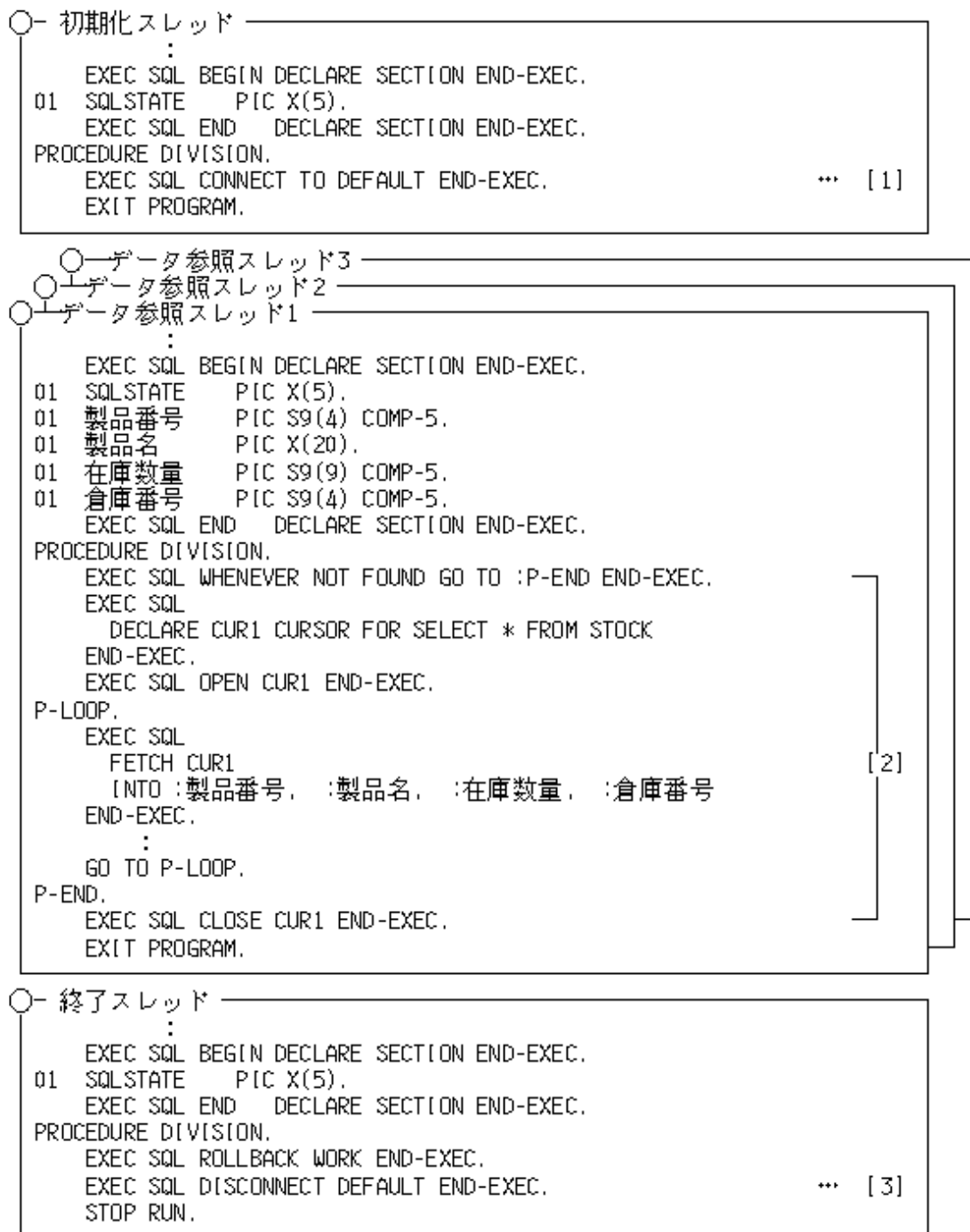
以降では、データベース表を変更しない(参照だけ行う)例、およびデータベース表を更新する例を説明します。

例では、サンプルデータベースのSTOCK表を使用しています。[参照]“[22.2.3.1 サンプルデータベース](#)”

### コネクション共有時のデータ参照

コネクションをスレッド間で共有して利用し、かつ、データベース表を変更しない(参照だけ行う)例を以下に示します。

図24.8 コネクション共有時のデータ参照例



[1] 初期化スレッドは、実行環境で最初に1回だけ起動されます。

初期化スレッドで、CONNECT文を実行し、サーバとのコネクションを接続します。

[2] データ参照スレッドは、クライアントからの要求ごとに起動されます。

データ参照スレッドで、カーソルを使用してSTOCK表からデータを1行ずつ取り出し、各列の値を対応するホスト変数の領域に設定します。取り出す行がなくなったとき、WHENEVER文で指定した手続き名“P-END”の位置に分岐し、データ参照スレッドを終了します。

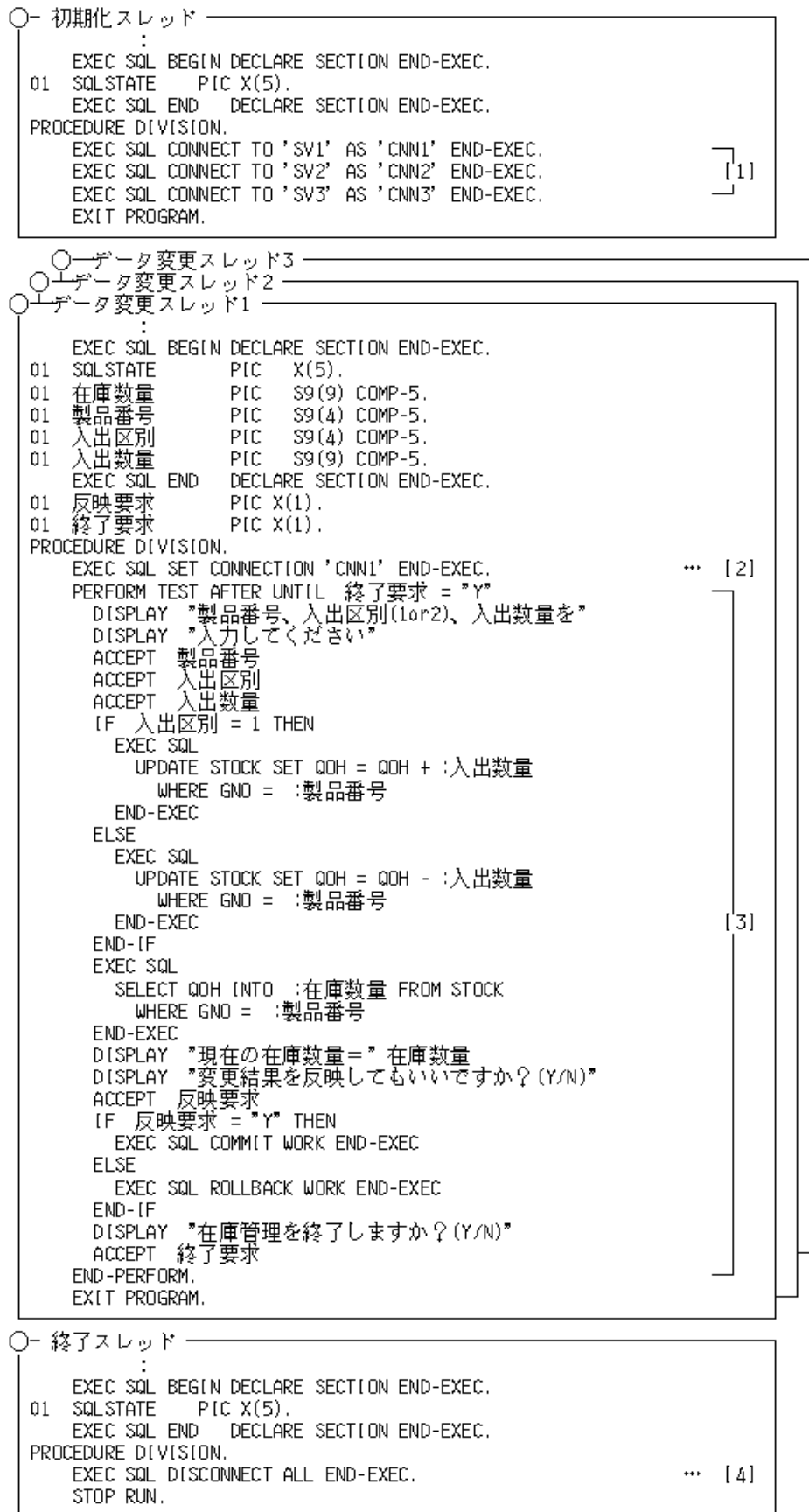
[3] 終了スレッドは、実行環境で最後に1回だけ起動されます。

終了スレッドで、ROLLBACK文を実行し、トランザクションを終了させ、DISCONNECT文を実行し、サーバとのコネクションを切断します。

## コネクション共有時のデータ更新

“[図24.9 コネクション共有時のデータ更新例](#)”は、コネクションをスレッド間で共有して利用し、かつ、データベース表を更新するCOBOLプログラム例です。

図24.9 コネクション共有時のデータ更新例



- [1] 初期化スレッドは、実行環境で最初に1回だけ起動されます。  
初期化スレッドで、データ変更スレッドを同時に起動する数(プログラム例では3つ)だけCONNECT文を実行し、サーバとのコネクションを接続します。
- [2] データ更新スレッドは、複数(プログラム例では3つ)同時に起動されます。  
データ更新スレッドで、SET CONNECTION文を実行し、利用する現コネクションを決定します。指定するコネクション名は、データ変更スレッドごとに異なるコネクション名(CNN1～CNN3)を指定します。
- [3] 製品番号、入出区別、入出数量の入力を要求し、在庫または出庫を切り分けて在庫数量を再計算します。次に再計算後の在庫数量を表示し、変更結果の反映有無を入力させます。一連の処理は、終了要求に”Y”が入力されるまで繰り返し行います。
- [4] 終了スレッドは、実行環境で最後に1回だけ起動されます。  
終了スレッドで、DISCONNECT文を実行し、すべてのサーバとのコネクションを切断します。

## 実行時の環境設定

コネクションをスレッド間で共有するには、ODBC情報ファイルのコネクション有効範囲にプロセスを指定します。[参照]“[22.2.8.1.2 ODBC情報ファイルの作成](#)”

ODBC情報ファイルは、ODBC情報設定ツールを使用して作成してください。[参照]“[22.2.8.2 ODBC情報設定ツールの使い方](#)”

コネクション有効範囲にプロセスを指定することにより、スレッド間でコネクションを共有して利用することができます。

## 24.6.2.2 注意事項

詳細については、“[注意事項](#)”を参照してください。

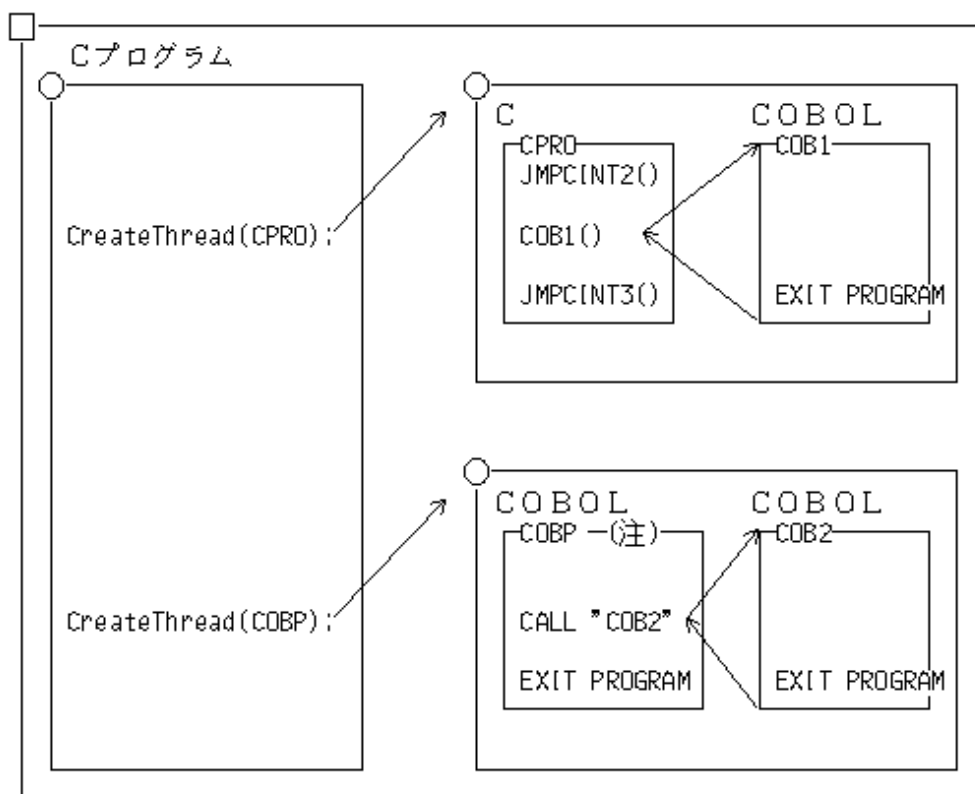
## 24.6.3 CプログラムからCOBOLプログラムをスレッドとして起動する方法

---

ここでは、CプログラムからCOBOLプログラムをスレッドとして起動する方法について説明します。

### 24.6.3.1 概要

CプログラムからCOBOLを呼び出す場合と違い、COBOLプログラムをスレッドとして起動する場合は、Windows関数を使用します。また、起動されたCOBOLプログラムのスレッドでEXIT PROGRAM文が実行されると、そのスレッドが終了するだけで呼出し元へは復帰しません。起動したCOBOLプログラムのスレッドからCOBOLプログラムを呼び出し、呼び出したCOBOLプログラムでEXIT PROGRAM文が実行されると呼出しの直後に復帰します。Cプログラムをスレッドとして起動した場合も同様です。



注:スレッドとして、直接起動されるプログラムは、STDCALL呼出し規約で宣言されている必要があります。

### 24.6.3.2 起動方法

CプログラムからCOBOLプログラムをスレッドとして起動するには、Windows関数“CreateThread”を使用します。スレッドを起動したCプログラムはスレッドの終了を待たずに次の処理を実行するので、スレッドとして起動されたCOBOLプログラムが終了する前にCプログラムが終了する場合があります。

Cプログラムで起動したスレッドの終了を待つには、Windows関数“WaitForSingleObject”を使用します。複数のスレッドの終了を待つには、Windows関数“WaitForMultipleObjects”を使用します。

スレッドとして起動されるCOBOLプログラムは、STDCALL呼出し規約でなければなりません。

### 24.6.3.3 パラメタの受渡し方法

Cプログラムからスレッドとして起動したCOBOLプログラムへ引数を渡す場合には、Windows関数“CreateThread”の第4引数に実引数を指定します。実引数は1つしか指定できません。Cプログラムからスレッドとして起動されたCOBOLプログラムへ渡すことのできる実引数の値は、記憶領域のアドレスでなければなりません。COBOLプログラムでは、手続き部の見出し(PROCEDURE DIVISION)またはENTRY文のUSING指定にデータ名を記述することにより、実引数に指定したアドレスにある領域の内容を受け取ります。実引数で指定したアドレスにある変数の宣言または定義でCONST型指定子を指定した場合、実引数に指定したアドレスにある領域の内容を変更してはいけません。

### 24.6.3.4 復帰コード(関数値)

手続き部の見出し(PROCEDURE DIVISION)のRETURNING指定の項目や特殊レジスタPROGRAM-STATUSに設定した値は、Windows関数“GetExitCodeThread”を使用して取り出します。

手続き部の見出しのRETURNING指定に記述する項目は、Cプログラムのデータ型(下図の[1]、[2]および[3])と対応していなければなりません。データ型の対応は、“[10.3.3 データ型の対応](#)”を参照してください。

- 関数C

```
extern int ...[1]
WINAPI COB(int *);
:
int WINAPI WinMain (~)
{
/* スレッドのハンドル */
HANDLE cobhnd;
/* スレッドID */
int cobthid;
/* COBOL プログラムに渡すパラメタ */
int cobprm;
/* 復帰コード */
int cobrcd; ...[2]
:
/* COBOL プログラム(COB)をスレッドとして起動 */
cobhnd = CreateThread(NULL,
 0,
 (LPTHREAD_START_ROUTINE) COB,
 &cobprm,
 0,
 &cobthid);
:
GetExitCodeThread(cobhnd, &cobrcd);
```

- COBOLプログラム(COB)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COB.
DATA DIVISION.
LINKAGE SECTION.
01 PRM PIC S9(9) COMP-5.
01 RTN-ITM PIC S9(9) COMP-5. ... [3]

PROCEDURE DIVISION WITH STDALL LINKAGE
 USING PRM
 RETURNING RTN-ITM.

:
MOVE 0 TO RTN-ITM.
IF エラー発生
 THEN MOVE 99 TO RTN-ITM
```



## 注意

手続き部の見出しにRETURNING指定を記述すると、特殊レジスタPROGRAM-STATUSに設定した値は、スレッドを起動したCプログラムには渡りません。

特殊レジスタPROGRAM-STATUSで関数値を渡す場合、スレッドを起動したCプログラムでは関数値をlong int型として受け取る必要があります。

- 関数C

```
extern int ...[1]
extern long int WINAPI
COB(int *);
:
int WINAPI WinMain (~)
{

HANDLE cobhnd;
int cobthid;
int cobprm;
```

```

long int cobrcd;
:
cobhnd = CreateThread(NULL,
 0,
 (LPTHREAD_START_ROUTINE) COB,
 &cobprm,
 0,
 &cobthid);
:
GetExitCodeThread(cobhnd, &cobrcd);

```

- COBOLプログラム(COB)

```

IDENTIFICATION DIVISION.
PROGRAM-ID. COB.
DATA DIVISION.
LINKAGE SECTION.
01 PRM PIC S9(9) COMP-5.
:
PROCEDURE DIVISION WITH STDALL LINKAGE
USING PRM.
:
IF エラー発生
THEN MOVE 99 TO PROGRAM-STATUS

```

### 24.6.3.5 翻訳とリンク

以下のプログラムを例に、翻訳とリンク方法について説明します。

- Cプログラム(CPROG.C)

当プログラムは、COBOLプログラムCOBTHD1、COBTHD2、COBTHD3をスレッドとして起動し、それぞれのCOBOLプログラムの復帰コードを獲得します。

```

#include <windows.h>
/* スレッドとして起動するCOBOL プログラムを宣言する */
/* 必ずSTDALL 呼出し規約で宣言してください */
extern int WINAPI COBTHD1(int *);
extern int WINAPI COBTHD2(int *);
extern int WINAPI COBTHD3(int *);
:
int WINAPI WinMain(~)
{
/* データ宣言 */
HANDLE cobhnd1;
HANDLE cobhnd23[2];
int cobrcd1;
int cobrcd23[2];
int cobtid1;
int cobtid23[2];
int cobprm1;
int cobprm23[2];
/* パラメタに1を設定してCOBTHD1を起動する */
cobprm1 = 1;
cobhnd1 = CreateThread(NULL,
 0,
 (LPTHREAD_START_ROUTINE) COBTHD1,
 &cobprm1,
 0,
 &cobtid1);
/* COBTHD1が終了するのを待つ */
WaitForSingleObject(cobhnd1, INFINITE);
/* COBTHD1の復帰コードを獲得する */

```



```

 GetExitCodeThread(cobhnd1, &cobrcd1);
/* パラメタ 2、3 を指定してCOBTHD2、COBTHD3を起動する */
cobprm23[0] = 2;
cobprm23[1] = 3;
cobhnd23[0] = CreateThread(NULL,
 0,
 (LPTHREAD_START_ROUTINE) COBTHD2,
 &cobprm23[0],
 0,
 &cobtid23[0]);
cobhnd23[1] = CreateThread(NULL,
 0,
 (LPTHREAD_START_ROUTINE) COBTHD3,
 &cobprm23[1],
 0,
 &cobtid23[1]);
/* COBTHD2、COBTHD3が終了するのを待つ */
WaitForMultipleObjects(2, cobhnd23, TRUE, INFINITE);
/* COBTHD2、COBTHD3の復帰コードを獲得する */
GetExitCodeThread(cobhnd23[0], &cobrcd23[0]);
GetExitCodeThread(cobhnd23[1], &cobrcd23[1]);
/* 起動したスレッドのハンドルをクローズする */
CloseHandle(cobhnd1);
CloseHandle(cobhnd23[0]);
CloseHandle(cobhnd23[1]);
return TRUE;
}

```

- COBOLプログラム(COBTHD1.COB)

当プログラムは、Cプログラム(CPROG.C)からスレッドとして起動され、パラメタが1なら復帰値0を返却し、1以外なら-1を返却します。

```

IDENTIFICATION DIVISION.
PROGRAM-ID. COBTHD1.
DATA DIVISION.
LINKAGE SECTION.
01 PRM1 PIC S9(9) COMP-5.
PROCEDURE DIVISION WITH STDALL LINKAGE USING PRM1.
 IF PRM1 = 1
 THEN MOVE 0 TO PROGRAM-STATUS
 ELSE MOVE -1 TO PROGRAM-STATUS
 END-IF.
EXIT PROGRAM.

```

- COBOLプログラム(COBTHD2.COB)

当プログラムは、Cプログラム(CPROG.C)からスレッドとして起動され、パラメタが2なら復帰値0を返却し、2以外なら-1を返却します。

```

IDENTIFICATION DIVISION.
PROGRAM-ID. COBTHD2.
DATA DIVISION.
LINKAGE SECTION.
01 PRM2 PIC S9(9) COMP-5.
PROCEDURE DIVISION WITH STDALL LINKAGE USING PRM2.
 IF PRM2 = 2
 THEN MOVE 0 TO PROGRAM-STATUS
 ELSE MOVE -1 TO PROGRAM-STATUS
 END-IF.
EXIT PROGRAM.

```

- COBOLプログラム(COBTHTD3.COB)

当プログラムは、Cプログラム(CPROG.C)からスレッドとして起動され、パラメタが3なら復帰値0を返却し、3以外なら-1を返却します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBTHTD3.
DATA DIVISION.
LINKAGE SECTION.
01 PRM3 PIC S9(9) COMP-5.
PROCEDURE DIVISION WITH STDCLL LINKAGE USING PRM3.
 IF PRM3 = 3
 THEN MOVE 0 TO PROGRAM-STATUS
 ELSE MOVE -1 TO PROGRAM-STATUS
 END-IF.
EXIT PROGRAM.
```



### 注意

COBOLプログラムは、必ずSTDCLL呼出し規約で宣言してください。

## 翻訳

Cプログラムを翻訳する

```
CL -c -MD CPROG.C
```

COBOLプログラムCOBTHTD1を翻訳する

```
COBOL32 -WC, "THREAD (MULTI)" COBTHTD1.COB
```

COBOLプログラムCOBTHTD2を翻訳する

```
COBOL32 -WC, "THREAD (MULTI)" COBTHTD2.COB
```

COBOLプログラムCOBTHTD3を翻訳する

```
COBOL32 -WC, "THREAD (MULTI)" COBTHTD3.COB
```



### 注意

COBOLプログラムを翻訳する場合は、必ず翻訳オプションTHREAD(MULTI)を指定してください。

## リンク

リンク方法については、“[5.3.2.3.4 DLL配下の実行用の初期化ファイルの使用について](#)”の“DLLエントリオブジェクトの結合”を参照してください。

COBOLプログラムCOBTHTD1を結合する

```
LINK COBTHTD1.OBJ F3BICBDM.OBJ F3BICIMP.LIB KERNEL32.LIB /DLL /ENTRY:COBDMAN /OUT:COBTHTD1.DLL
```

```
COBTHTD1.OBJ : COBOLプログラムCOBTHTD1のオブジェクト
F3BICBDM.OBJ : DLLエントリオブジェクト
F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ
KERNEL32.LIB : Windows関数のインポートライブラリ
```

COBOLプログラムCOBTHTD2を結合する

```
LINK COBTHTD2.OBJ F3BICBDM.OBJ F3BICIMP.LIB KERNEL32.LIB /DLL /ENTRY:COBDMAN /OUT:COBTHTD2.DLL
```

COBTHD2.OBJ : COBOLプログラムCOBTHD2のオブジェクト  
F3BICBDM.OBJ : DLLエントリオブジェクト  
F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
KERNEL32.LIB : Windows関数のインポートライブラリ

COBOLプログラムCOBTHD3を結合する

```
LINK COBTHD3.OBJ F3BICBDM.OBJ F3BICIMP.LIB KERNEL32.LIB /DLL /ENTRY:COBDMAIN /OUT:COBTHD3.DLL
```

COBTHD3.OBJ : COBOLプログラムのCOBTHD3のオブジェクト  
F3BICBDM.OBJ : DLLエントリオブジェクト  
F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
KERNEL32.LIB : Windows関数のインポートライブラリ

Cプログラムを結合する

```
LINK CPROG.OBJ COBTHD1.LIB COBTHD2.LIB COBTHD3.LIB /OUT:CPROG.EXE
```

CPROG.OBJ : CプログラムCPROGのオブジェクト  
COBTHD1.LIB : COBOLプログラムCOBTHD1のインポートライブラリ  
COBTHD2.LIB : COBOLプログラムCOBTHD2のインポートライブラリ  
COBTHD3.LIB : COBOLプログラムCOBTHD3のインポートライブラリ

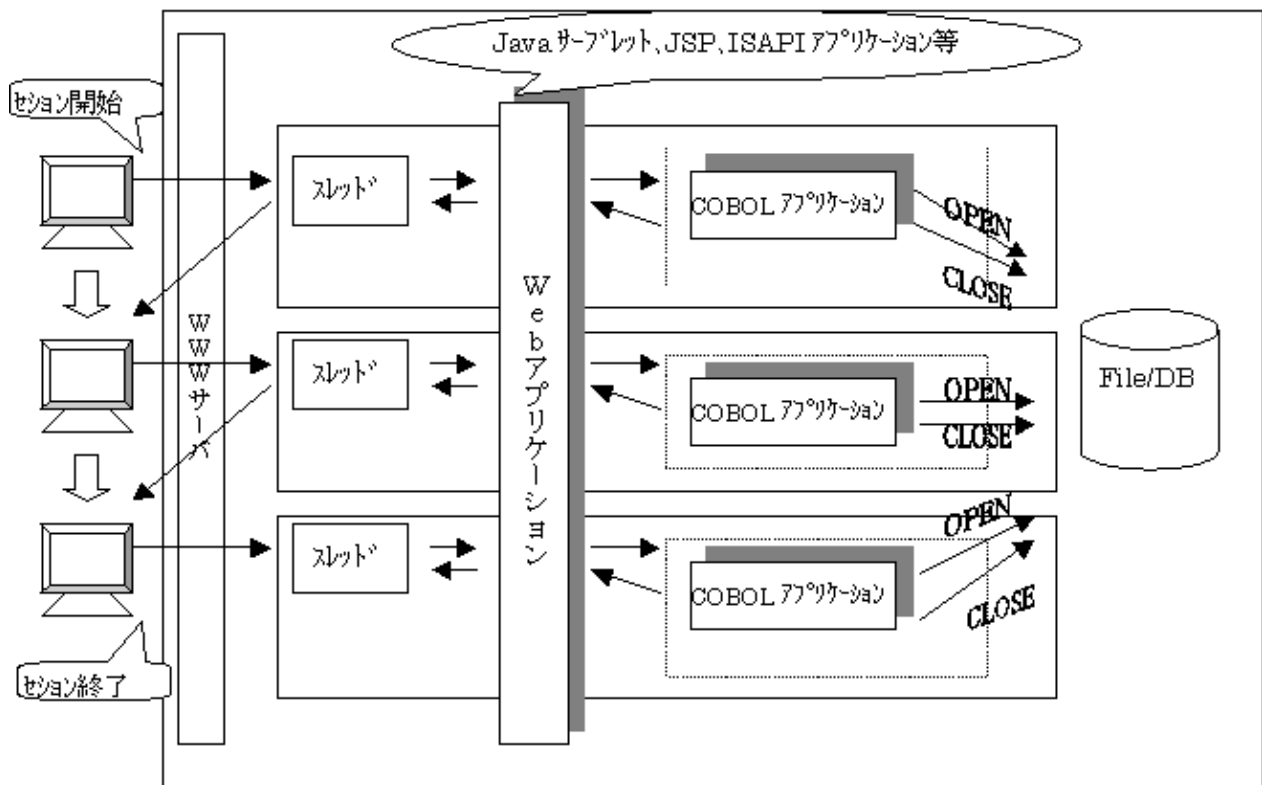
## 24.6.4 スレッド間で実行単位の資源を引き継ぐ方法

ここでは、複数スレッド間で実行単位の資源を引き継ぐ方法について説明します。

### 24.6.4.1 概要

COBOLのサーバアプリケーションをマルチスレッドで実行した場合、クライアントからサーバアプリケーションが呼び出された時にCOBOLの実行単位が開始され、クライアントへ復帰する時に実行単位が終了します。このため、COBOLランタイムシステムにより管理されるファイル結合子、DBカーソル、作業場所節に記述したデータなどの実行単位内で有効な資源は、実行単位が終了すると解放されてしまいます。たとえば、Webアプリケーションではクライアントからの呼び出しごとにCOBOLの実行単位の資源の生成と解放が繰り返されるため、複数のスレッドをまたがるセッション内では状態を保持することができません。

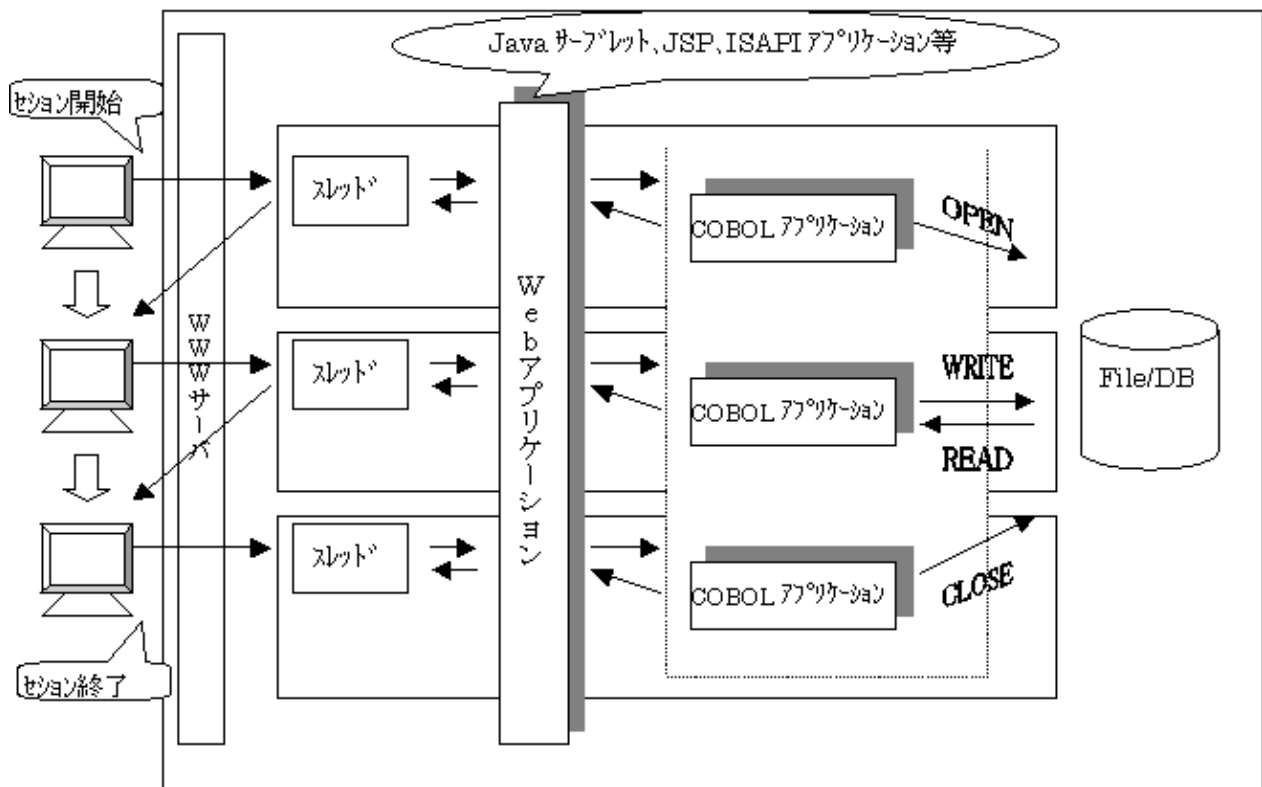
【実行単位を引き継がない場合】



複数のスレッドにまたがるセッション内で実行単位の資源を引き継ぐためには、クライアントから呼び出されるCOBOLアプリケーションは、毎回、同一のスレッドで実行されなければなりません。しかし、実際にはサーバアプリケーションを実行するスレッドは、同一のスレッドに固定されません。

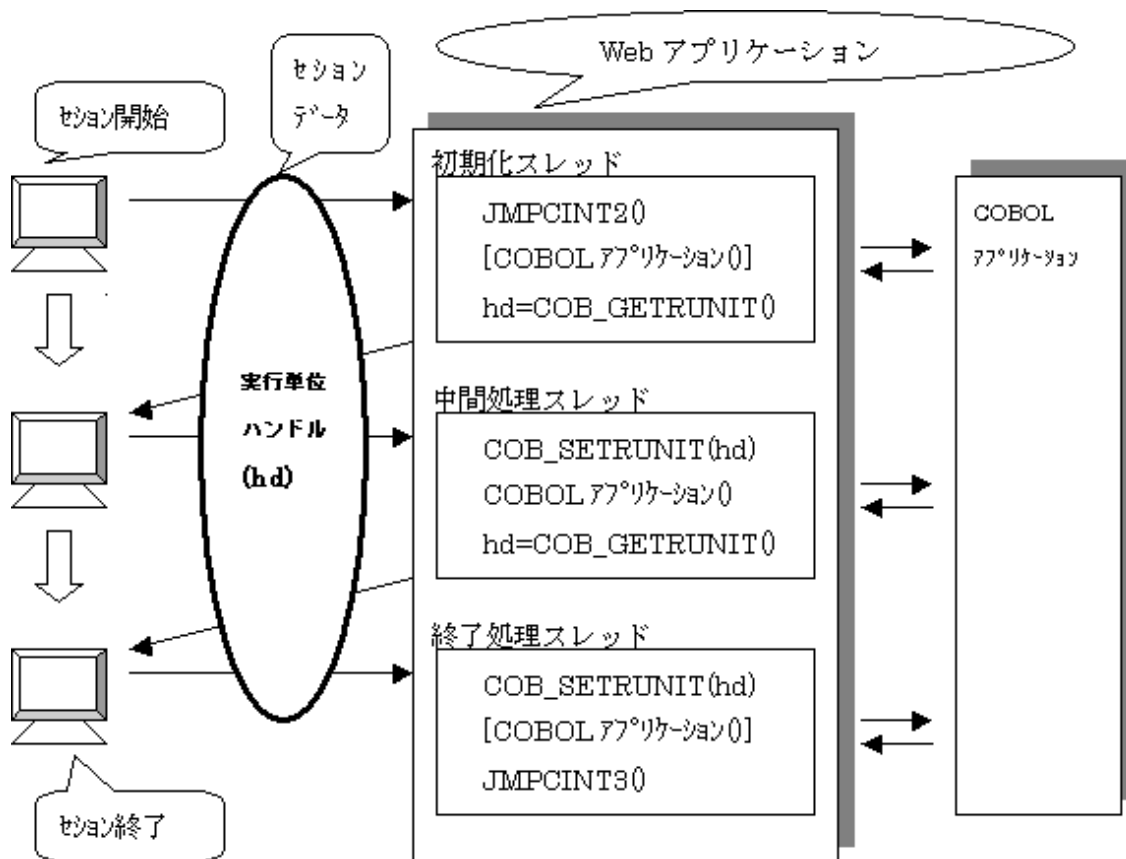
このような場合に、COBOLの実行単位のハンドルを返却するサブルーチンと、その実行単位のハンドルをCOBOLランタイムシステムに通知するサブルーチンを使用して、スレッド間で実行単位を引き継ぐことができます。引き継ぐ実行単位のハンドルとセッションの関連付けは、Cookieなどを用いた管理機構によって、COBOLアプリケーションの呼出し側で行う必要があります。

[実行単位を引き継ぐ場合]



#### 24.6.4.2 利用方法

下図のようにWebアプリケーションからCOBOLが提供するサブルーチン呼び出すことで、COBOLの実行単位のハンドルをスレッド間で持ち回ることができます。



別スレッドで開設された実行単位の資源(ファイル結合子、DBカーソル、作業場所節に記述したデータなど)を引き継ぐことにより、クライアントからの呼出しごとにオープン処理からクローズ処理までを行わなければならないファイル操作のオーバーヘッドなどが削減でき、処理の効率化も図ることができます。

## 参考

JavaプログラムからCOBOLプログラムの呼出しは、J Business Kit(以降、JBKといいます)との連携で簡単に実現できます。また、JBKのラッピング機能では、JavaのオブジェクトがCOBOLプログラムの実行単位を引き継ぎます。このため、同じJavaオブジェクトから呼び出されるCOBOLプログラムの間では、実行単位の資源を引き継いで利用することができます。

JBKは、Interstage Apworksに同梱されているコンポーネントです。

## 24.6.4.3 サブルーチンの使い方

### 24.6.4.3.1 COBOL実行単位ハンドル取得サブルーチン

COBOLの実行単位を識別するハンドルを取得する場合には、COB\_GETRUNITサブルーチンを使用します。

初期化スレッドの処理の最初に、JMPCINT2を呼び出して実行単位を開始する必要があります。

### 指定方法

呼出しの記述(C言語での呼出し方)

```

型宣言部 :
extern unsigned long COB_GETRUNIT(void);
データ宣言部 :
unsigned long hd; /* COBOLの実行単位ハンドル格納域 */
手続き部 :
hd = COB_GETRUNIT();

```

## インタフェース

### パラメタ

なし

### 復帰コード

正常時: COBOL実行単位のハンドル

異常時: 0

## 24.6.4.3.2 COBOL実行単位ハンドル設定サブルーチン

COBOLの実行単位のハンドルを呼出し元のスレッドに設定する場合には、COB\_SETRUNITサブルーチンを使用します。

終了スレッドの処理の最後に、JMPCINT3を呼び出して実行単位を終了する必要があります。

## 指定方法

呼出しの記述(C言語での呼出し方)

```
型宣言部 :
extern int COB_SETRUNIT(unsigned long hd);
データ宣言部 :
extern unsigned long hd;
/* COBOLの実行単位ハンドル格納域(COB_GETRUNITで取得) */
手続き部 :
COB_SETRUNIT(hd);
```

## インタフェース

### パラメタ

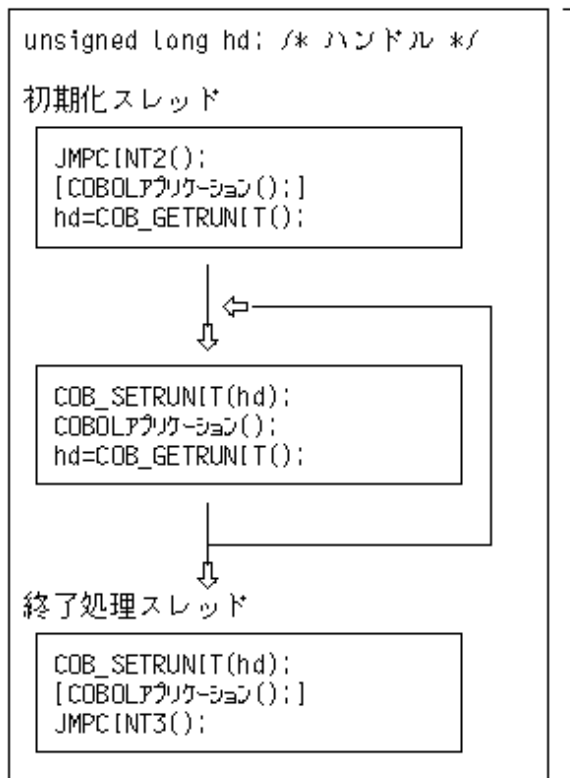
hd: COBOL実行単位のハンドル

### 復帰コード

正常時: 0

異常時:

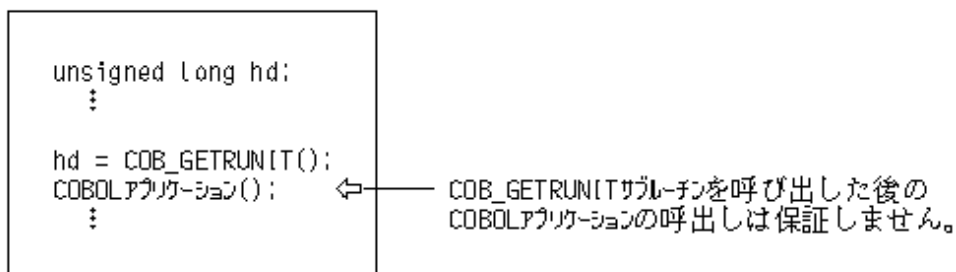
- -1 入力パラメタが正しくない
- -2 既にCOBOLの実行単位が存在する
- -99 その他のエラー(SEに連絡してください)



※[ ]内省略可能

#### 24.6.4.4 注意事項

- ・ タイムアウトなどのCOBOLアプリケーションが動作しているプロセス自身が終了しないような異常終了が発生した場合は、COB\_SETRUNITサブルーチンで実行単位のハンドル設定後、JMPCINT3を呼び出して実行単位の終了を行う必要があります。JMPCINT3の呼出しを行わない場合、実行単位の資源が解放されずに残るため、メモリリークの原因となります。
- ・ 複数のスレッドが同時に一つのCOBOLの実行単位を共有することはできません。必ず個々のCOBOLの実行単位が単一のスレッドだけで使用されるようにしてください。
- ・ COB\_GETRUNITサブルーチンを呼び出すと、呼び出したスレッドのCOBOLの実行単位をクリアします。このため、COB\_GETRUNITサブルーチンを呼び出した後に、同一スレッドからCOBOLアプリケーションを呼び出すことはできません。





- COB\_SETRUNITサブルーチンを呼び出すと、呼び出したスレッドのCOBOLの実行単位を上書きします。このため、COB\_SETRUNITサブルーチンを呼び出す前に、そのスレッドからCOBOLアプリケーションを呼び出すことはできません。

```

int rtncode;
extern unsigned long hd;
⋮

COBOLアプリケーション();
rtncode = COB_SETRUNIT(hd);
if (rtncode == -2) {
 エラ-処理
}
⋮

```

← COB\_SETRUNITサブルーチンを呼び出す前のCOBOLアプリケーションの呼出しはできません。この状態でCOB\_SETRUNITサブルーチンを呼び出すと復帰コード -2でエラーになります。

- COUNT情報中に出力されるプロセスID(PID=)およびスレッドID(TID=)は、COUNT機能が情報を出力するときのプロセスIDおよびスレッドIDです。COUNT情報の出力時期の詳細については、“[19.4.2 COUNT情報](#)”を参照してください。
- 当サブルーチンを利用する際には、以下の制限があります。
  - スクリーン操作機能  
スクリーン操作機能は使用できません。
  - 小入出力機能  
機能名CONSOLE、SYSINおよびSYSOUTで、COBOLコンソールウィンドウは使用できません。入出力先にシステムコンソールウィンドウまたはファイルを指定してください。
  - 表示ファイル  
表示ファイル(帳票印刷、画面入出力、非同期型メッセージ通信)は、スレッド間でファイル結合子を引き継ぐことができません。OPEN文からCLOSE文までを同一スレッドで行うようにしてください。
  - FORMAT句付き印刷ファイル  
FORMAT句付き印刷ファイルは、スレッド間でファイル結合子を引き継ぐことができません。OPEN文からCLOSE文までを同一スレッドで行うようにしてください。
  - 簡易アプリ間通信機能  
簡易アプリ間通信機能は、スレッド間でサーバ識別子を引き継ぐことができません。COBCLI\_OPENサブルーチンの呼出しからCOBCLI\_CLOSEサブルーチンの呼出しまでを同一スレッドで行うようにしてください。
  - プリコンパイラの利用によるSymfoware連携  
プリコンパイラを利用したSymfoware連携は、スレッド間でコネクションおよびカーソルを引き継ぐことができません。コネクションの接続から切断およびカーソルのオープンからクローズまでを同一スレッドで行うようにしてください。
  - RDMファイル  
RDMファイルは、スレッド間でファイル結合子を引き継ぐことができません。OPEN文からCLOSE文までを同一スレッドで行うようにしてください。

## 24.6.4.5 翻訳とリンク

### 翻訳

特別な指定は必要ありません。

### リンク

COB\_GETRUNITサブルーチンおよびCOB\_SETRUNITサブルーチンを使用する場合は、呼出し元のモジュールのリンク時にF3BICIMP.LIBをリンクしてください。

## 24.7 翻訳から実行までの方法

### 24.7.1 翻訳とリンク

ここでは、マルチスレッドプログラムの翻訳とリンク方法について説明します。

マルチスレッドプログラムの翻訳の手順で、シングルスレッドプログラムと異なるのは、翻訳オプションTHREAD(MULTI)が必要となることです。また、スレッド間共有外部データまたはスレッド間共有外部ファイルを使用する場合は、THREAD(MULTI)のほかに翻訳オプションSHREXTが必要となります。

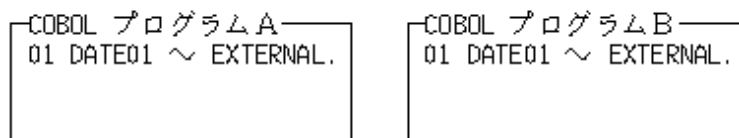
マルチスレッドプログラムのリンク手順は、シングルスレッドプログラムと同じです。

翻訳の方法は、“[第3章 プログラムを翻訳する](#)”を参照してください。

リンクの方法は、“[第4章 プログラムをリンクする](#)”を参照してください。

#### 24.7.1.1 COBOLプログラムだけでDLLを作成する場合

COBOLプログラムAとCOBOLプログラムBでCOB.DLLを作成します。



#### 翻訳

COBOLプログラムAを翻訳する

```
COBOL32 -WC, "THREAD (MULTI), SHREXT" A. COB
```

COBOLプログラムBを翻訳する

```
COBOL32 -WC, "THREAD (MULTI), SHREXT" B. COB
```



#### 注意

スレッド間共有外部データまたはスレッド間共有外部ファイルを使用する場合は、THREAD(MULTI)のほかに翻訳オプションSHREXTを指定してください。

#### リンク

リンク方法については、“[5.3.2.3.4 DLL配下の実行用の初期化ファイルの使用について](#)”の“DLLエントリオブジェクトの結合”を参照してください。

COBOLプログラムAとCOBOLプログラムBをリンクしてCOB.DLLを作成する

```
LINK A. OBJ B. OBJ F3BICBDM. OBJ F3BICIMP. LIB KERNEL32. LIB /DLL /ENTRY:COBDMAN /OUT:COB. DLL
```

A. OBJ : COBOL プログラムAのオブジェクトファイル  
B. OBJ : COBOL プログラムBのオブジェクトファイル  
F3BICBDM. OBJ : DLLのエントリオブジェクトファイル  
F3BICIMP. LIB : COBOLランタイムシステムのインポートライブラリ  
KERNEL32. LIB : Windows関数のインポートライブラリ

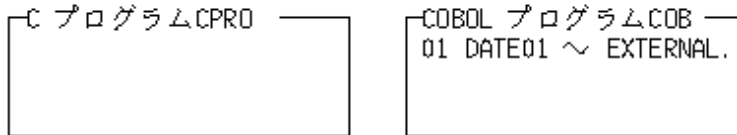


## 注意

LINKオプション/ENTRYに、必ずCOBDMANを指定してください。

### 24.7.1.2 COBOLプログラムとCプログラムでDLLを作成する場合

CプログラムCPROとCOBOLプログラムCOBでCCOB.DLLを作成します。



## 翻訳

CプログラムCPROを翻訳する

```
CL -c -MD CPRO.C
```

COBOLプログラムCOBを翻訳する

```
COBOL32 -WC, "THREAD (MULTI), SHREXT" COB.COB
```



## 注意

- ・ Cプログラムをマルチスレッド環境下で動作するように翻訳してください (Visual C++では翻訳オプション/MDが必要となります)。
- ・ COBOLプログラムでスレッド間共有外部データまたはスレッド間共有外部ファイルを使用する場合は、THREAD(MULTI)のほか、に翻訳オプションSHREXTを指定してください。

## リンク

リンク方法については、“5.3.2.3.4 DLL配下の実行用の初期化ファイルの使用について”の“DLLエントリオブジェクトの結合”を参照してください。

CプログラムCPROとCOBOLプログラムCOBをリンクしてCCOB.DLLを作成する

```
LINK CPRO.OBJ COB.OBJ F3BIMLDM.OBJ F3BICIMP.LIB KERNEL32.LIB MSVCRT.LIB /DLL /OUT:CCOB.DLL
```

CPRO.OBJ : CプログラムCPROのオブジェクトファイル  
 COB.COB : COBOLプログラムCOBのオブジェクトファイル  
 F3BIMLDM.OBJ : DLLエントリオブジェクト  
 F3BICIMP.LIB : COBOLランタイムシステムのインポートライブラリ  
 KERNEL32.LIB : Windows関数のインポートライブラリ  
 MSVCRT.LIB : Cランタイムライブラリ (翻訳オプション/MDで翻訳した場合、当ライブラリが必要となります)。



## 注意

プロジェクトマネージャウィンドウまたはWINLINKウィンドウでリンクを行う場合、[リンクオプション]ダイアログのDLLのエントリオブジェクトで“言語間結合用”を選択し、Cランタイムライブラリとして、“MSVCRT.LIB”を指定します。

# 24.7.2 実行

ここでは、マルチスレッドプログラムの実行の手順について説明します。

## 24.7.2.1 実行環境情報の設定

ここでは、実行環境情報の設定の手順について説明します。なお、シングルスレッドモードと重複する説明については、“[第5章 プログラムを実行する](#)”を参照してください。

### 実行用の初期化ファイルの内容

マルチスレッドモードで動作するCOBOLアプリケーションの使用する実行用の初期化ファイルの記述内容について説明します。

マルチスレッドモードでは、実行用の初期化ファイルの内容は、共通部だけで構成されます。セクションに記述された情報は無視されます。

以下に、実行用の初期化ファイルの内容の記述形式を示します。

|               |       |     |
|---------------|-------|-----|
| 実行環境情報名= 設定内容 | … [1] | 共通部 |
|---------------|-------|-----|

#### [1] 環境変数情報(共通部)

ここには、各プログラムに共通する環境変数情報を記述します。それぞれの環境変数情報の指定形式については、“[5.4.1 環境変数情報](#)”を参照してください。なお、1つの行に2個以上の環境変数情報を設定することはできません。ここで設定した実行環境情報は、アプリケーションの環境変数に反映されるため、アプリケーションが終了するまで有効となります。

なお、ここで設定された内容は、プロセス内で共有されるため、どのスレッドでも有効となります。

### 実行用の初期化ファイルの記述例

```
@MessOutFile=C:\MESSAGE.TXT
@CnsIWinSize=(80, 24)
@CnsIBufLine=100
@WinCloseMsg=ON
@IconName=COB85EXE
@CBR_ENTRYFILE=C:\TEST001.ENT
```

シングルスレッドアプリケーションをマルチスレッド環境に移行する場合、以下の注意が必要です。

- ・ セクションに記述された情報を共通部に指定してください。

例) COBOL85.CBR

```
OUTFILE=C:\OUTDATA.DAT

[A]
OUTFILEA=C:\A\DATAA.OUT
INFLEA=C:\A\MASTER.IN

[B]
OUTFILEB=C:\B\DATAB.OUT
INFLEB=C:\B\MASTER.IN
```

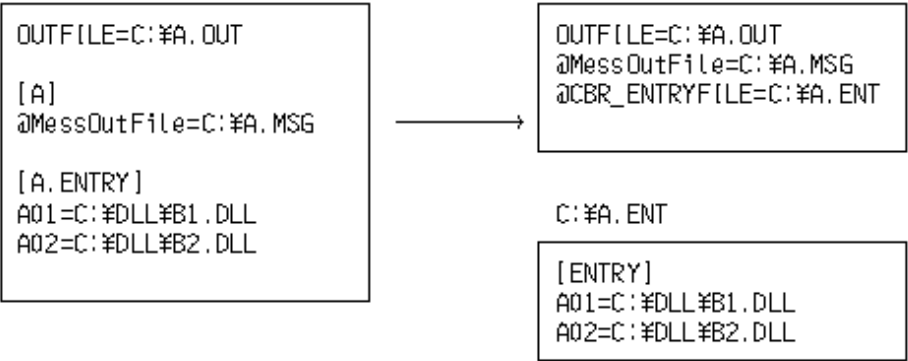


COBOL85.CBR

```
OUTFILE=C:\OUTDATA.DAT
OUTFILEA=C:\A\DATAA.OUT
INFLEA=C:\A\MASTER.IN
OUTFILEB=C:\B\DATAB.OUT
INFLEB=C:\B\MASTER.IN
```

- ・ エントリ情報が記述されている場合、エントリ情報ファイルに指定し、共通部の@CBR\_ENTRYFILEにファイル名を指定してください。

例) COBOL85.CBR



- ・ サーバ環境下では、環境変数の値としてファイル名に相対パスを指定しているものと、相対の基点がサーバによって異なる可能性があります。このような場合は、絶対パスでの指定が可能であれば絶対パスによるファイル名の指定を行ってください。

スレッドモードと実行用の初期化ファイル内の有効な記述についてまとめると、以下のようになります。

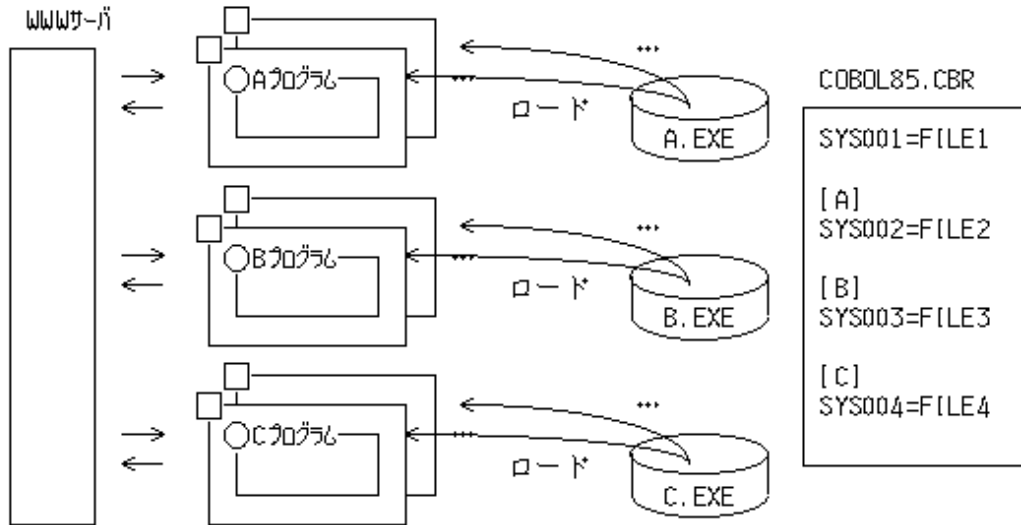
|       | シングルスレッドモード | マルチスレッドモード |
|-------|-------------|------------|
| セクション | 有効          | 無効         |
| 共通部   | 有効          | 有効         |



例

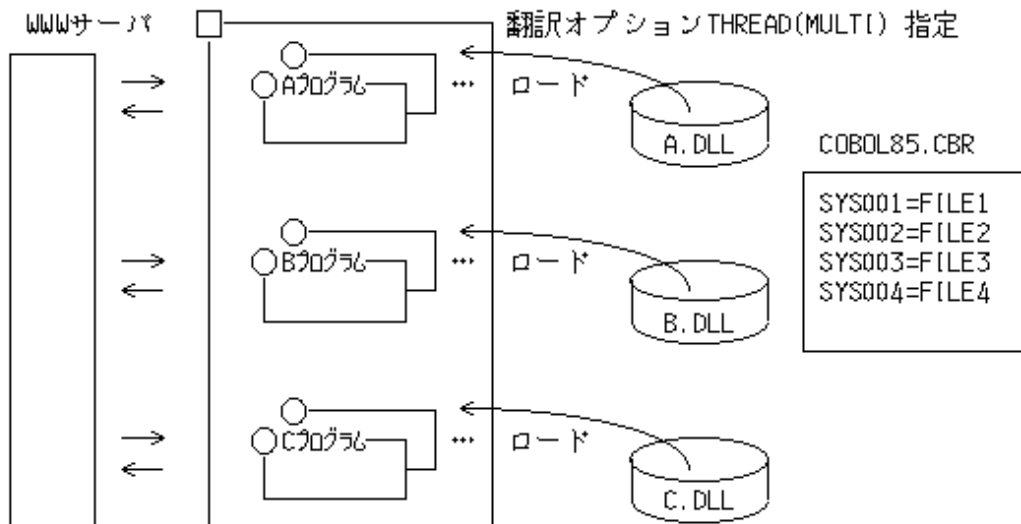
Webサーバで使用されているA、BおよびCのシングルスレッドプログラムをマルチスレッドプログラムに移行する例

## シングルスレッドプログラム



↓ 移行

## マルチスレッドプログラム



## サーバ環境でのDLLの格納位置のCOBOL85.CBRの使用について

実行用の初期化ファイルは、プロセスで1つ有効となります。

IISなどのサーバ制御プログラムから複数のCOBOLプログラム(DLL)を別々のプロセスで動作させることで、プロセスごとに異なる実行環境情報を使用することができます。

このとき、COBOLプログラムのDLLは同一のプロセスで動作するアプリケーションごとに1つのフォルダに格納し、共通部に必要な情報を記述するようにしてください。

同一の実行環境で動作するアプリケーションのCOBOLプログラム(DLL)が1つのフォルダにない場合、呼出し元のプログラムの処理によっては、どのフォルダのCOBOL85.CBRファイルが読み込まれるかわからなくなることがありますので注意してください。

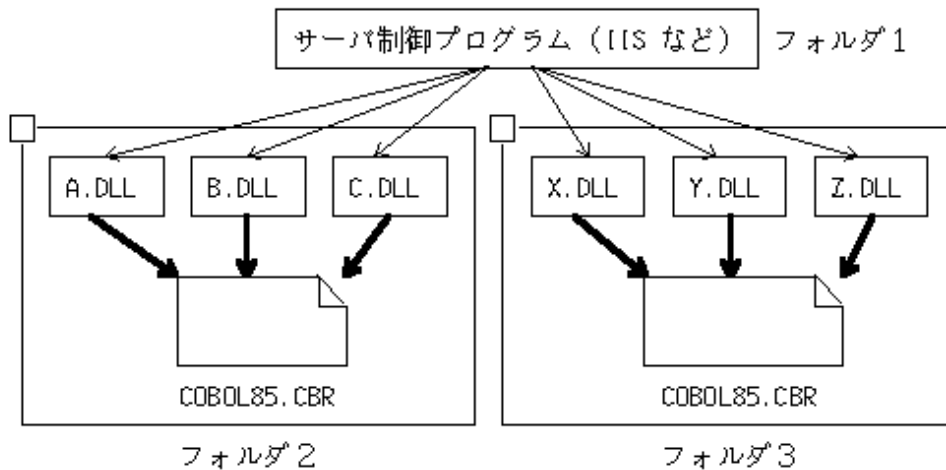
なお、DLLの格納位置からのCOBOL85.CBRファイルの読み込みを行うためには、DLLエントリオブジェクトをDLLに結合する必要があります。[参照]“[5.3.2.3.4 DLL配下の実行用の初期化ファイルの使用について](#)”



## 例

プロセス単位で同一フォルダのCOBOL85.CBRファイルを使用する

以下の例では、フォルダ1～3は別フォルダを指します。



### 24.7.2.2 実行環境情報の指定形式

ここでは、マルチスレッドモードでだけ有効な環境変数情報を説明します。

#### @CBR\_SYMFOWARE\_THREAD (Symfoware連携でマルチスレッド動作可能にする指定)

```
@CBR_SYMFOWARE_THREAD = MULTI
```

プリコンパイラを利用したSymfoware連携のマルチスレッドプログラムを動作可能にします。[参照]“[24.5.3 プリコンパイラの利用によるSymfoware連携](#)”

#### @CBR\_THREAD\_MODE (スレッドモードの指定)

```
@CBR_THREAD_MODE = SINGLE
```

マルチスレッド翻訳オプションで翻訳されたプログラムを、マルチスレッドモードではなくシングルスレッドモードで動作させます。指定しない場合、翻訳オプションで指定したモードで動作します。

この環境変数情報は、実行用の初期化ファイルに指定しても無効となります。直接、環境変数に設定してください。[参照]“[24.3.3 プログラムの実行とスレッドモード](#)”

#### @CBR\_THREAD\_TIMEOUT (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 = 待ち時間 (秒)
```

スレッド同期制御サブルーチンで無限待ちを指定した場合、待ち時間を変更する際に指定します。待ち時間は、0から最大32桁(秒)の数字で指定します。指定しない場合、無限待ちとなります。[参照]“[24.9 スレッド同期制御サブルーチン](#)”

#### @CBR\_SSIN\_FILE (スレッド単位に入力ファイルをオープンする指定)

```
@CBR_SSIN_FILE = THREAD
```

スレッド単位に入力ファイルをオープンする場合に指定します。[参照]“[11.1.5.5 ACCEPT文のファイル入力拡張機能](#)”

# 24.8 マルチスレッドプログラムのデバッグ方法

COBOLが提供する以下の機能は、マルチスレッドプログラムで用いても、基本的にデバッグ方法に変更はありません。

- ・ TRACE機能
- ・ CHECK機能
- ・ COUNT機能
- ・ 対話型デバッガ

ただし、マルチスレッドプログラムに特有する動作により、留意しなければならない事項がいくつかあります。

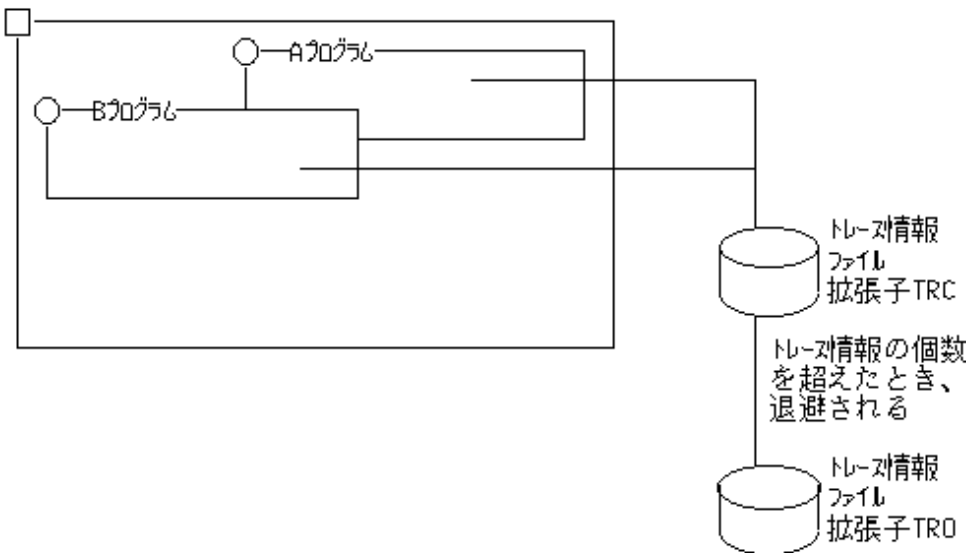
それぞれの機能に分けて、注意事項を説明します。

## 24.8.1 TRACE機能

トレース情報の内容に変更はありません。[参照]“19.3 TRACE機能の使い方”

ただし、各スレッドから採取されたトレース情報は、1つのファイル(拡張子TRC)に格納されます。

以下に図で示します。



トレース情報の見方を、以下の図で説明します。

NetCOBOL DEBUG INFORMATION    DATE 2007-05-01    TIME 20:29:52    PID=00000125

### TRACE INFORMATION

|     |    |   |                 |               |              |
|-----|----|---|-----------------|---------------|--------------|
| [1] | 1  | A | DATE 2007-05-01 | TIME 20:27:50 | TID=0000010C |
|     | 2  |   | 22.1            | TID=0000010C  |              |
|     | 3  |   | 23.1            | TID=0000010C  |              |
| [1] | 4  | B | DATE 2007-05-01 | TIME 20:27:50 | TID=0000011E |
|     | 5  |   | 22.1            | TID=0000011E  |              |
|     | 6  |   | 23.1            | TID=0000011E  |              |
|     | 7  |   | 24.1            | TID=0000011E  |              |
|     | 8  |   | 25.1            | TID=0000011E  |              |
| [2] | 9  |   | 26.1            | TID=0000011E  |              |
|     | 10 |   | 24.1            | TID=0000010C  |              |
|     | 11 |   | 25.1            | TID=0000010C  |              |



|     |    |                                                           |
|-----|----|-----------------------------------------------------------|
|     | 12 | 26.1 TID=0000010C                                         |
| [2] | 13 | THE INTERRUPTION WAS OCCURRED. PID=00000125, TID=0000011E |
|     | 14 | 27.1 TID=0000010C                                         |
|     | 15 | 28.1 TID=0000010C                                         |
| [3] | 16 | EXIT-THREAD TID=0000010C                                  |

#### [1] プログラムに割り当てられたスレッドID

プログラムAのスレッドIDは10Cです。プログラムBのスレッドIDは11Eです。

#### [2] 例外通知メッセージ

例外が発生したのは、スレッドIDが11EのプログラムBです。

#### [3] スレッド終了通知メッセージ

スレッドIDが終了したのは、スレッドIDが10CのプログラムAです。

上記の結果から、以下のことが分かります。

- ・ スレッドIDが10CのプログラムAは正常に動作した。
- ・ スレッドIDが11EのプログラムBは26行目の実行文で例外が発生した。



#### 参考

DISPLAY ... UPON SYSERRを使うと、トレース情報に任意のデータを出力することができます。

プログラムが使用するデータの遷移などを調べる場合に便利です。

なお、DISPLAY ... UPON SYSERRを使う場合には、スレッドIDも一緒に出力されるように、環境変数情報@CBR\_SYSERR\_EXTENDにYESを指定してください。[参照]“[5.4.1.46 @CBR\\_SYSERR\\_EXTEND \(SYSERR出力情報の拡張の指定\)](#)”



#### 注意

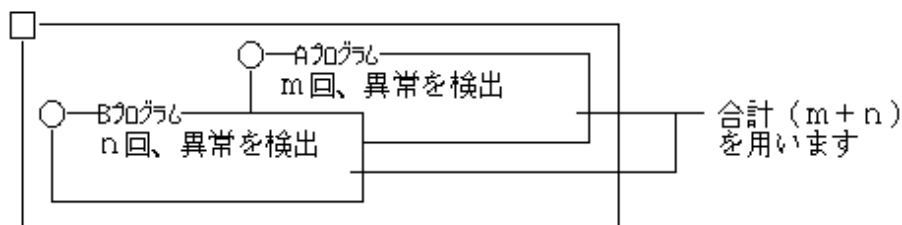
一度に多くのスレッドを実行する場合、大量のトレース情報が1つのファイルに書き込まれます。より多くのトレース情報が1つのファイルに出力されるように、トレース情報の個数(環境変数情報@GOPTのr指定)を調整してください。

## 24.8.2 CHECK機能

CHECK機能が有効な場合に、出力されるエラーメッセージの内容および検出方法に変更はありません。[参照]“[19.2 CHECK機能の使い方](#)”

ただし、出力メッセージの回数は、プロセス内で検出された回数の合計を用います。

以下の図に示します。



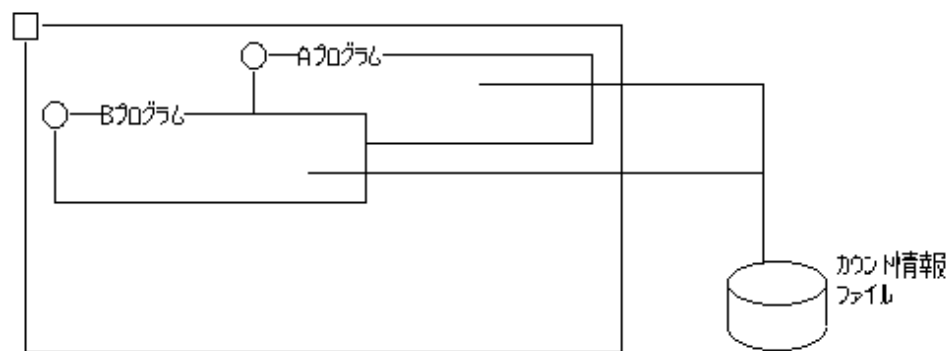
## 24.8.3 COUNT機能

カウント情報の内容に変更はありません。[参照]“[19.4 COUNT機能の使い方](#)”

ただし、以下のような動作となります。

- ・ 各スレッドから採取されたカウント情報は、1つのファイルに格納されます。
- ・ カウント情報が書き込まれる集計結果は、スレッド単位に出力します。プロセス全体で集計されません。

以下に図で示します。



出力されるカウント情報の見方を、以下の図で説明します。

```
NetCOBOL COUNT INFORMATION(END OF RUN UNIT) DATE 2007-05-01 TIME 21:38:59
 PID=000000B5 TID=000000D4
 [1]
STATEMENT EXECUTION COUNT PROGRAM-NAME : A
:
NetCOBOL COUNT INFORMATION(END OF RUN UNIT) DATE 2007-05-01 TIME 21:38:59
 PID=000000B5 TID=0000012A
 [1]
STATEMENT EXECUTION COUNT PROGRAM-NAME : B
:
```

[1] プログラムに割り当てられたスレッドID

プログラムAのスレッドIDは0D4です。プログラムBのスレッドIDは12Aです。

24.8.4 対話型デバグガ

詳細については、“[20.6.4 マルチスレッドでのデバグガの対応](#)”を参照してください。

24.9 スレッド同期制御サブルーチン

ここでは、スレッドの同期制御を行うためのサブルーチンについて説明します。スレッド同期制御サブルーチンには、データロックサブルーチンとオブジェクトロックサブルーチンがあります。

24.9.1 データロックサブルーチン

| サブルーチン名         | 機能              |
|-----------------|-----------------|
| COB_LOCK_DATA   | ロックキーに対するロックの獲得 |
| COB_UNLOCK_DATA | ロックキーに対するロックの解放 |

同一プロセス内のスレッド間で同期制御が必要となる場合に、当サブルーチンを使用して、互いに同じデータ名のロックキーに対してロックの獲得と解放を行うことで相互排他ロックが可能となります。このときに指定するデータ名は、プロセス内で一意にする必要があります。

COB\_LOCK\_DATAの呼出し時に、パラメタで指定されたデータ名に対応するロックキーが作成され、ロックを獲得します。指定されたデータ名に対応するロックキーが既に存在する場合は、そのロックキーに対してロックを獲得します。

ロックを獲得できるスレッドは1つだけであり、ロックを獲得したスレッドだけが実行されます。同じロックキーに対してロックを獲得しようとしたほかのスレッドは、ロックを獲得しているスレッドがロックを解放するまで待つことになります。

ロックは、COB\_UNLOCK\_DATAを呼び出すことによって解放されます。

## 動的プログラム構造による呼出し

動的プログラム構造で当サブルーチンを呼び出す場合、以下のようなエントリ情報が必要になります。エントリ情報の指定方法については、“[5.4.2 副プログラムのエントリ情報](#)”を参照してください。

```
[ENTRY]
COB_LOCK_DATA=F3BIEFNC.DLL
COB_UNLOCK_DATA=F3BIEFNC.DLL
```

### 24.9.1.1 COB\_LOCK\_DATA

#### 機能

指定されたデータ名に対応するロックキーに対してロックを獲得します。

#### 呼び出し形式

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LOCK-KEY PIC X(30).
01 WAIT-TIME PIC S9(9) COMP-5.
01 ERR-DETAIL PIC 9(9) COMP-5.
01 RET-VALUE PIC S9(9) COMP-5.

PROCEDURE DIVISION.
 CALL "COB_LOCK_DATA" WITH C LINKAGE
 USING BY REFERENCE LOCK-KEY
 BY VALUE WAIT-TIME
 BY REFERENCE ERR-DETAIL
 RETURNING RET-VALUE.
```

#### パラメタ

##### LOCK-KEY

ロックを獲得するロックキーの名前を30バイト以内で指定します。ロックキーの名前が30バイトに満たない場合は、末尾に空白が必要となります。

##### WAIT-TIME

ロックを獲得するまでの待ち時間(秒)を指定します。-1を指定すると無限待ちとなります。

無限待ちを指定した場合、環境変数情報@CBR\_THREAD\_TIMEOUTに待ち時間(秒)を指定することにより、待ち時間を変更できます。これは、デッドロックが発生した場合の箇所を特定したりする場合に利用します。

##### ERR-DETAIL

戻り値が-255の場合、Windowsのシステムエラーコードが返ります。

## 戻り値

### RET-VALUE

成功時は、0が返ります。また、スレッドモードがシングルスレッドの場合は、ロックが不要であるため、ロックを獲得せず、1を返します。動作としては問題ありません。  
失敗時は、負の値が返ります。詳細については、“[24.9.3 エラーコード](#)”を参照してください。

## 24.9.1.2 COB\_UNLOCK\_DATA

### 機能

指定されたデータ名に対応するロックキーに対してロックを解放します。

### 呼び出し形式

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LOCK-KEY PIC X(30).
01 ERR-DETAIL PIC 9(9) COMP-5.
01 RET-VALUE PIC S9(9) COMP-5.

PROCEDURE DIVISION.
 CALL "COB_UNLOCK_DATA" WITH C LINKAGE
 USING BY REFERENCE LOCK-KEY
 BY REFERENCE ERR-DETAIL
 RETURNING RET-VALUE.
```

### パラメタ

#### LOCK-KEY

ロックを解放するロックキーの名前を30バイト以内で指定します。ロックキーの名前が30バイトに満たない場合は、末尾に空白が必要となります。

#### ERR-DETAIL

戻り値が-255の場合、Windowsのシステムエラーコードが返ります。

## 戻り値

### RET-VALUE

成功時は、0が返ります。また、スレッドモードがシングルスレッドの場合は、ロックが不要であるため、ロックを解放せず、1を返します。動作としては問題ありません。  
失敗時は、負の値が返ります。詳細については、“[24.9.3 エラーコード](#)”を参照してください。

## 24.9.2 オブジェクトロックサブルーチン

| サブルーチン名           | 機能               |
|-------------------|------------------|
| COB_LOCK_OBJECT   | オブジェクトに対するロックの獲得 |
| COB_UNLOCK_OBJECT | オブジェクトに対するロックの解放 |

同一プロセス内のスレッド間でオブジェクトを共有している場合に、当サブルーチンを使用して、同一のオブジェクトに対してロックの獲得と解放を行うことで、相互排他ロックが可能となります。

COB\_LOCK\_OBJECTの呼び出し時に、オブジェクトを指定することにより、そのオブジェクトに対してロックを獲得します。

ロックを獲得できるスレッドは1つだけであり、ロックを獲得したスレッドだけがオブジェクトを所有できます。同じオブジェクトに対してロックを獲得しようとしたほかのスレッドは、ロックを獲得しているスレッドがロックを解放するまで待つことになります。

ロックは、COB\_UNLOCK\_OBJECTを呼び出すことによって解放されます。

## 動的プログラム構造による呼出し

動的プログラム構造で当サブルーチン呼び出す場合、以下のようなエントリ情報が必要になります。エントリ情報の指定方法については、“[5.4.2 副プログラムのエントリ情報](#)”を参照してください。

```
[ENTRY]
COB_LOCK_OBJECT=F3B1EFNC. DLL
COB_UNLOCK_OBJECT=F3B1EFNC. DLL
```

### 24.9.2.1 COB\_LOCK\_OBJECT

#### 機能

指定されたオブジェクトに対してロックを獲得します。

#### 呼び出し形式

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ OBJECT REFERENCE クラス名.
01 WAIT-TIME PIC S9(9) COMP-5.
01 ERR-DETAIL PIC 9(9) COMP-5.
01 RET-VALUE PIC S9(9) COMP-5.

PROCEDURE DIVISION.
 CALL "COB_LOCK_OBJECT" WITH C LINKAGE
 USING BY REFERENCE OBJ
 BY VALUE WAIT-TIME
 BY REFERENCE ERR-DETAIL
 RETURNING RET-VALUE.
```

#### パラメタ

##### OBJ

ロックを獲得するオブジェクトのオブジェクト参照を指定します。

##### WAIT-TIME

ロックを獲得するまでの待ち時間(秒)を指定します。-1を指定すると無限待ちとなります。

無限待ちを指定した場合、環境変数情報@CBR\_THREAD\_TIMEOUTに待ち時間(秒)を指定することにより、待ち時間を変更できます。これは、デッドロックが発生した場合の箇所を特定したりする場合に利用します。

##### ERR-DETAIL

戻り値が-255の場合、Windowsのシステムエラーコードが返ります。

#### 戻り値

##### RET-VALUE

成功時は、0が返ります。また、スレッドモードがシングルスレッドの場合は、ロックが不要であるため、ロックを獲得せず、1を返します。動作としては問題ありません。

失敗時は、負の値が返ります。詳細については、“[24.9.3 エラーコード](#)”を参照してください。

### 24.9.2.2 COB\_UNLOCK\_OBJECT

#### 機能

指定されたオブジェクトに対してロックを解放します。

## 呼び出し形式

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ OBJECT REFERENCE クラス名.
01 ERR-DETAIL PIC 9(9) COMP-5.
01 RET-VALUE PIC S9(9) COMP-5.
PROCEDURE DIVISION.
 CALL "COB_UNLOCK_OBJECT" WITH C LINKAGE
 USING BY REFERENCE OBJ
 BY REFERENCE ERR-DETAIL
 RETURNING RET-VALUE.
```

## パラメタ

### OBJ

ロックを解放するオブジェクトのオブジェクト参照を指定します。

### ERR-DETAIL

戻り値が-255の場合、Windowsのシステムエラーコードが返ります。

## 戻り値

### RET-VALUE

成功時は、0が返ります。また、スレッドモードがシングルスレッドの場合は、ロックが不要であるため、ロックを獲得せず、1を返します。動作としては問題ありません。

失敗時は、負の値が返ります。詳細については、“[24.9.3 エラーコード](#)”を参照してください。

## 24.9.3 エラーコード

ここでは、スレッド同期制御サブルーチンのエラー発生時の戻り値について説明します。

表24.1 エラーコード一覧

| エラーコード | 意味と処置                                                                                                                                                                                                                                                     | 対象サブルーチン      |                 |                 |                   |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|-----------------|-----------------|-------------------|
|        |                                                                                                                                                                                                                                                           | COB_LOCK_DATA | COB_UNLOCK_DATA | COB_LOCK_OBJECT | COB_UNLOCK_OBJECT |
| -1     | COBOL の実行環境が開設されていません。COBOL の実行環境を開設してから使用してください。                                                                                                                                                                                                         | 通知される         | 通知される           | 通知される           | 通知される             |
| -2     | パラメタの指定が誤っています。以下である可能性があります。 <ul style="list-style-type: none"><li>ロックキーの名前の指定誤り ( COB_LOCK_DATA、COB_UNLOCK_DATA)</li><li>待ち時間の指定誤り ( COB_LOCK_DATA、COB_LOCK_OBJECT)</li><li>オブジェクト参照にNULL オブジェクトが指定されている ( COB_LOCK_OBJECT、COB_UNLOCK_OBJECT)</li></ul> | 通知される         | 通知される           | 通知される           | 通知される             |

| エラーコード | 意味と処置                                                                       | 対象サブルーチン      |                 |                 |                   |
|--------|-----------------------------------------------------------------------------|---------------|-----------------|-----------------|-------------------|
|        |                                                                             | COB_LOCK_DATA | COB_UNLOCK_DATA | COB_LOCK_OBJECT | COB_UNLOCK_OBJECT |
| -3     | あるスレッドのプログラムが異常終了したため、ロックが正しく解放されませんでした。異常終了したプログラムのエラーの原因を取り除き、再度実行してください。 | 通知される         | ×               | 通知される           | ×                 |
| -4     | ロックの獲得の待ち時間を経過しました。                                                         | 通知される         | ×               | 通知される           | ×                 |
| -5     | ロックを獲得していない、または別のスレッドが獲得したロックを解放しようとしていました。                                 | ×             | 通知される           | ×               | 通知される             |
| -255   | システムエラーが発生しました。この場合は、パラメタのERR-DETAILにWindowsのシステムエラーコードが設定されます。             | 通知される         | 通知される           | 通知される           | 通知される             |

## 24.10 注意事項

### 24.10.1 COM連携機能

ここでは、マルチスレッド環境下でCOM連携機能を使用する場合の注意事項を説明します。COM連携機能の詳細については、“[第26章 COM機能](#)”を参照してください。

COBOLランタイムシステムは、たとえば、COBOLでEXEファイルを作成して実行する場合のように、シングルスレッドモードで動作する場合にはCOMの動作環境を自動的に作成するようにしています。シングルスレッドモードで動作させるような使い方をする場合には、特に以下の説明を理解する必要はありません。

以下では、マルチスレッド環境下で動作するようなプログラムを作成する場合の注意事項を説明します。なお、一般にはCOMクライアント、COMサーバと呼ばれる方をしますが、この節ではほかの箇所との整合性のためCOMクライアント、COMサーバという呼び方をします。

#### 24.10.1.1 COMのマルチスレッド環境

COM連携機能のマルチスレッド環境下での動作の説明の前に、COMのマルチスレッドの考え方を簡単に説明します。詳細については、COM/DCOMについての市販の解説書などを参考にしてください。

COMのマルチスレッド対応では次のようなスレッドモデルを考えます。

##### STA(Single Thread Apartment)

COMサーバはそれが生成されたスレッド内だけで有効なモデル。すなわち、あるスレッドで作成したCOMサーバは、ほかのスレッドからは参照できません。このモデルの場合、COMサーバのメソッドの呼び出しは、スレッド内で同期制御されます。STAはスレッド単位でプロセス内にいくつでも作ることができます。

##### MTA(Multi Thread Apartment)

MTAはプロセス内に1つまたは0個(MTAを利用しない場合)だけ存在し、それぞれのスレッドでMTAを共有することで、複数のスレッドからCOMサーバを利用することができるモデル。このとき、COMサーバのメソッドの呼び出しは同期制御されないで、COMサーバは同期制御されなくても正しく動作するように作成されていなければなりません。

どちらのスレッドモデルになるかは、COM環境の初期化の方法に依存します。

#### 24.10.1.2 COBOLの初期化方法

COBOLのCOM環境の初期化は次のように行います。

表24.2 COM環境の初期化方法

| COBOL プログラムのスレッドモード |              | 初期化されるCOM環境のスレッドモデル |
|---------------------|--------------|---------------------|
| シングルスレッドモード         |              | STA                 |
| マルチスレッドモード          | そのスレッドは初期化済み | 初期化済みのモデルに従う        |
|                     | そのスレッドは未初期化  | MTA                 |

“[24.10.1.1 COMのマルチスレッド環境](#)”でも説明しましたが、COMがSTAで初期化されている場合、そのスレッド内で生成したCOMサーバは、ほかのスレッドで参照することはできないことに注意してください。

### 24.10.1.3 COM連携で使用するオブジェクトの解放について

COM連携で使用する特殊クラス\*COM、\*COM-ARRAY、\*COM-EXCEPTIONは、それぞれ内部的にCOMサーバ、COMの配列、COMの例外情報を保持しています。これらは、それを保持するオブジェクトが解放された時点で同時に解放されます。COM連携で使用するこれらのオブジェクトは、使用が終了した時点で必ず解放するようにしてください。オブジェクトは、オブジェクト参照項目にほかのオブジェクト参照(解放のためだけならNULLオブジェクト)をSETすることで解放されます。

特殊クラス\*COM、\*COM-ARRAY、\*COM-EXCEPTIONの詳細については、それぞれ“[26.2.3 \\*COMクラス](#)”、“[26.6.1 \\*COM-ARRAYクラス](#)”、“[26.6.2 \\*COM-EXCEPTIONクラス](#)”を参照してください。

### 24.10.1.4 COMサーバの寿命について

COM連携で使用するオブジェクトの寿命は、基本的に通常のオブジェクトの寿命と同じです。オブジェクトの寿命については、“[16.2.2 オブジェクトの寿命](#)”を参照してください。

ここでは、通常のオブジェクトとは異なる点を説明します。

通常のCOBOLオブジェクトは、COBOLプログラムの終了時に使用しているメモリが回収されます。COM連携で使用している特殊クラスオブジェクトは、COBOL領域内に取られているメモリは回収されますが、そのオブジェクトが保持しているCOMサーバやCOMの配列は解放されません(COMの例外情報は解放されます)。これは、COBOLシステムが解放処理をする時点で、COMの環境が残っている保証がないためです。COMサーバやCOMの配列を解放するためには、使用が終了した時点で必ず解放するようにしてください。

ただし、次のように特定の条件を満たす場合には終了時にCOMサーバ、COMの配列は自動的に解放されます。

- COBOLプログラムがシングルスレッドモードで動作しているときには、COBOLプログラムの終了時に、その時点で解放されていないCOMサーバ、COMの配列はすべて解放されます。
- COBOLプログラムがマルチスレッドモードで動作していて、COM環境がMTAで初期化されている場合には、JMPCINT4を呼び出すことで、すべてのCOMサーバ、COMの配列が解放されます。



JMPCINT4が呼ばれた時点で、STAで初期化したスレッドで作成したCOMサーバ、COMの配列が残っている場合にはエラーになります。これは、STAで初期化したスレッド内で生成されたCOMサーバ、COMの配列は、ほかのスレッドから参照できないという理由によります。

## 24.10.2 オブジェクト指向プログラミング機能

シングルスレッドプログラムでは、実行単位の終了と実行環境の閉鎖のタイミングが同じでした。このため、オブジェクト参照項目にNULLオブジェクトを転記してから実行単位を終了しなくても、実行単位の終了時に実行環境が閉鎖され、残っているオブジェクトインスタンスはメモリ上から解放されるため、特に問題は発生しませんでした。

しかし、マルチスレッドプログラムでは、実行単位の終了と実行環境の閉鎖のタイミングは異なります。このため、オブジェクト参照項目にNULLオブジェクトを転記しないで実行単位を終了すると、どこからも参照されないオブジェクトインスタンスがメモリ上に残ることになり、メモリ不足につながります。したがって、必ず、オブジェクト参照項目にNULLオブジェクトを転記してから実行単位を終了してください。マルチスレッドプログラムの実行単位と実行環境については、“[24.3.1 実行環境と実行単位](#)”を参照してください。

## 24.10.3 簡易アプリ間通信機能

マルチスレッドモードでは、簡易アプリ間通信機能の各サブルーチンで使用するサーバ識別子は、スレッド間で共用できません。



スレッドごとに一意のサーバ識別子を使用してください。

## 24.10.4 印刷機能

---

ここでは、スレッド間で同じファイル結合子を操作して印刷ファイルまたは表示ファイルを共有する場合の注意事項について説明します。

複数の入出力文の実行によって1つの帳票が生成されるような処理を行う場合、同じファイル結合子に対して複数のスレッドから入出力文の実行が行われると、入出力文の実行順序が非同期となるため、印刷データの出力順序が一定とならずに意図しない印刷結果を得ることがあります。このため、同一帳票に対する一連の処理開始から終了までの間、他スレッドの入出力文との競合を抑止する必要があります。

他スレッドとの競合状態を防ぐためには、一連の処理の前後でスレッドの同期制御を行います。

スレッド間で同じファイル結合子を持つファイルおよびスレッドの同期制御の詳細については、“[24.6.1 入出力機能の利用](#)”を参照してください。

## 24.10.5 動的プログラム構造

---

ここでは、動的プログラム構造によって呼び出された副プログラムに対して、CANCEL文を実行する場合の注意点について説明します。

- マルチスレッドモードで動作している場合も、CANCEL文によって、プログラムを初期状態にすることができます。しかし、CANCEL文に指定されたプログラムのDLLは仮想メモリから削除されません。これにより、実行中のプログラムに対して、ほかのスレッドがCANCEL文を実行しても、プログラムは正常に動作します。
- 翻訳オプションTHREAD(MULTI)で翻訳された副プログラムは、複数の副プログラムでDLLが構成されている場合でも、それぞれの副プログラムに対するCANCEL文で、初期状態になります。

## 第25章 Unicode

本章では、Unicodeで動作するCOBOLアプリケーションの作成方法について説明します。

### 25.1 Unicodeサポート概要

NetCOBOLでのUnicodeの実装について、まずは概要から説明します。

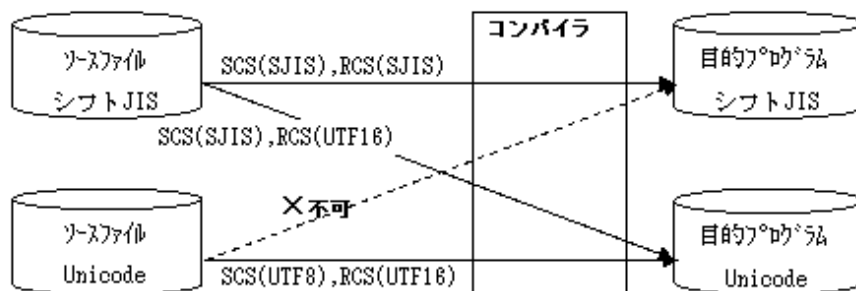
#### 25.1.1 資源

NetCOBOLでは、Unicodeデータを扱う場合、ソースプログラムなどの翻訳資源はシフトJISまたはUnicode(UTF-8)で作成することができます。

翻訳資源のコード系は翻訳オプションSCSで、実行時のコード系は翻訳オプションRCSで指定します。

組合せの可否は下表および下図のとおりです。省略値は、シフトJISです。

|           | RCS(SJIS) | RCS(UTF16) |
|-----------|-----------|------------|
| SCS(SJIS) | 組合せ可(省略値) | 組合せ可       |
| SCS(UTF8) | 組合せ不可     | 組合せ可       |



動作モードがUnicode時の各資源のコード系は、下表のとおりです。

|      | 資源                                                  | コード系                           |
|------|-----------------------------------------------------|--------------------------------|
| 翻訳時  | ソースファイル<br>登録集ファイル                                  | シフトJIS または Unicode(UTF-8) (注1) |
|      | オプションファイル                                           | 混在可能 (注3)                      |
|      | 翻訳リスト                                               | シフトJIS または Unicode(UTF-8) (注2) |
| リンク時 | モジュール定義ファイル                                         | シフトJIS                         |
| 実行時  | 実行用初期化ファイル<br>エントリ情報ファイル<br>クラス情報ファイル<br>ODBC情報ファイル | シフトJIS または Unicode(UTF-8) (注3) |
|      | 印刷情報ファイル<br>プリンタ情報ファイル<br>ウィンドウ情報ファイル               | シフトJIS                         |
|      | キー定義ファイル                                            | シフトJIS または Unicode(UTF-8)      |
|      | COBOLファイル(順/行順/相対/索引)                               | Unicode(UTF-8 または UTF-16) (注4) |
|      | 小入出力ファイル<br>TRACE情報ファイル                             | Unicode(UTF-8)                 |
|      |                                                     |                                |

|  | 資源                      | コード系 |
|--|-------------------------|------|
|  | COUNT情報ファイル<br>汎用ログファイル |      |

注1:“A.2.45 SCS(ソースファイルのコード系)”によって指定します。なお、ひとつの翻訳単位内で混在させることはできません。

注2:“A.2.45 SCS(ソースファイルのコード系)”で指定されたコード系で出力されます。

注3:BOMの有無でコード系を識別します。実行単位内で混在させることができます。

注4:レコード定義の字類によって決定します。[参照]“25.3.7 COBOLファイル”

## 参考

### BOMについて

Windows系システムでは、一般的にUnicodeのテキストファイルに表現形式を識別するためのBOM(Byte Order Mark)を付与します。このため、NetCOBOLコンパイラおよびランタイムシステムでもBOM付きを基本とします。ただし、UNIX系システムではBOMなしが一般的であるため、分散開発環境などで管理対象となるソースファイルや登録集ファイルは、BOMなしでも入力可能にしています。

## 25.1.2 表現形式

Windows Vistaを発端に、最近ではJIS X 0213:2004に対応した文字セット(以降、JIS2004と記述します)を利用する環境が整いつつあります。

このJIS2004で定義された文字を全て活用するにはUnicodeしか選択肢がありません。そして、Unicodeでは、一部の文字(300文字程度)が2バイトで表現できる範囲外(2面)にマップされたため、従来のUCS-2と呼ばれる表現形式ではこれらの文字が表現できなくなりました。そこで、UCS-2に代わってUTF-16と呼ばれる表現形式が利用されるようになっていきます。このUTF-16は、0面(いわゆるBMPと呼ばれる範囲)に配置された文字はUCS-2と互換性のある2バイトのコードで、2面に配置された文字はサロゲートペアと呼ばれる4バイトのコードで表現されます。

NetCOBOLでは、UTF-16表現で文字データを扱う場合、日本語項目を使用します。このとき、サロゲートペアで表現される文字は1文字につき2けたの宣言が必要となります。従来の、格納できる文字数=けた数が成立しないため、注意が必要です。また、Windows系システムで一般的に利用されているリトルエンディアンに加えて、ビッグエンディアンを選択することが可能になりました。指定方法については、“A.2.40 RCS(実行時コード系の指定)”を参照してください。

英数字項目の表現形式は従来どおりUTF-8です。BMPの範囲であれば1文字を格納するために最大で3けた用意すればよかったのですが、前述したサロゲートペアの文字を英数字項目に格納する場合、1文字につき4けたの宣言が必要となるため、注意してください。

なお、字類が数字のデータ項目についてはシフトJISと同じです。

以下に字類と表現形式をまとめます。

| 項目のレベル | 字類  | 表現形式         |
|--------|-----|--------------|
| 基本項目   | 英字  | ASCII        |
|        | 英数字 | ASCII(UTF-8) |
|        | 日本語 | UTF-16       |
| 集団項目   | 英数字 | ASCII(UTF-8) |

## 参考

日本語項目のエンコードがUTF-16になりますが、前述したとおり、BMPの範囲ではコードに互換性があるため、NetCOBOL V9.0以前に作成したUnicodeアプリケーションの互換性がなくなる心配はありません。ただし、Windows VistaなどのJIS2004対応されたシステムへ展開する際は、JIS2004対応に伴い一部文字で字形の変更がされた点、サロゲートペアの文字(1文字につき2けたを消費)が入力される可能性がある点を観点に、十分に検証することをおすすめします。



## 注意

### IVS文字の利用について

Windows 8およびWindows Server 2012では、IVS(Ideographic Variation Sequence)を利用した文字を使用できますが、本COBOLコンパイラおよびランタイムシステムでは、IVSを利用した文字は使用できません。

## 25.1.3 言語要素

Unicodeに関連する追加言語要素を説明します。

### 日本語文字定数

UTF-16では、ASCII文字(いわゆる半角の英数字)も1文字2バイトで表現されるため、シフトJISではできなかった日本語項目でのASCII文字のハンドリングができるようになります。これに伴い、日本語文字定数中にASCII文字が記述できるようになりました。

```
WORKING-STORAGE SECTION.
01 ADDR PIC N(40).
:
MOVE NC"神奈川県川崎市中原区上小田中4-1-1 コーポ富士通 B-5" TO ADDR.
```

### 組込み関数

UTF-16とUTF-8を相互に変換する組込み関数を提供します。それぞれの場合に応じて、以下の組込み関数を使用してください。

- UTF-16のデータをUTF-8へ変換する場合はDISPLAY-OF関数を使用
- UTF-8のデータをUTF-16へ変換する場合はNATIONAL-OF関数を使用

```
WORKING-STORAGE SECTION.
01 PIC-X PIC X(10).
01 PIC-N PIC N(06) VALUE NC"12ABあ亜".
:
MOVE FUNCTION DISPLAY-OF(PIC-N) TO PIC-X.
MOVE FUNCTION NATIONAL-OF(PIC-X) TO PIC-N.
```



## 注意

- 関数の結果の長さは、引数に指定された文字列の変換結果の長さになります。なお、引数の値が規定されたコードとして正しくない場合、結果は保証しません。
- NetCOBOL V9.0以前は、機能的に同じ関数としてUTF8-OF関数およびUCS2-OF関数を提供していました。これら関数はV10以降も引き続き使用できますが、今後はDISPLAY-OF関数およびNATIONAL-OF関数の使用を推奨します。

### 字類条件

Windows系システムが先行して、JIS2004を利用できる環境、つまり、Unicode3.2に対応した環境を整えつつあります。しかし、UNIX系システムをはじめとして、旧バージョンのUnicodeまでしか対応できてないシステムやミドルウェアが大勢を占める状況にあります。よって、それらと連携してシステムを構築する場合、互換をとれる範囲でデータを流通させる必要があります。

Interstage Charset Managerが提供するCharset Validator(以降、Validatorと呼びます)の機能を利用することで、データ項目に格納されている文字の範囲を検査することができます。前述したシステムやミドルウェアと連携する場合、要所に文字範囲の検査を入れることで、システムの安定性が保持されます。

Validatorは、サーバアプリケーションで文字範囲の検査を行うために、C言語およびJAVAのAPIを提供しています。COBOLアプリケーションでValidatorの文字範囲の検査を利用するためには、APIを呼び出すC言語のプログラムを作成し、COBOLプログラムからC言語プログラムをCALL文で呼び出します。

Validator機能については、Charset Managerのマニュアル“システム構築ガイド バリデーション編”を参照してください。

COBOLプログラムからC言語プログラムの呼出しについては、“C言語プログラムとの結合”を参照してください。

日本語項目の表現形式に採用したUTF-16では、サロゲートペアの文字に2けたの領域を必要とします。このため、1文字につき1けたで設計された従来系システムは、厳密には正しく動作しないことになります。ただし、サロゲートペアに該当する文字は、4000文

字を超えるJIS2004追加文字のうち、わずか300文字程度であり、当該文字を利用できなくとも、システム再設計するまで踏み込まずに動作させたいという局面は多いはずです。

NetCOBOLでは、データ項目に格納されている文字がBMP範囲内かを容易に検査するための字類条件(BMP)を用意しました。要所に検査を入れることで、従来資産をJIS2004が実装されたシステム上で安定して運用することができます。

```
WORKING-STORAGE SECTION.
01 PIC-N PIC N(10).
:
IF PIC-N IS NOT BMP THEN
 DISPLAY "サロゲートペアが含まれています。再入力してください。"
END-IF.
```

この字類条件BMPは、格納された文字数をカウントする場合や、部分参照や転記の際に“[文字の泣き別れ](#)”が発生していないかを確認する際にも利用できます。以下に文字数をカウントする例を示します。

```
WORKING-STORAGE SECTION.
01 PIC-N PIC N(10).
01 CHAR-NO PIC S9(4) BINARY VALUE ZERO.
01 CNT PIC 9(2).
:
PERFORM VARYING CNT FROM 1 BY 1 UNTIL CNT > 10
 IF PIC-N(CNT:1) IS NOT BMP THEN
 ADD 1 TO CNT
 END-IF
 ADD 1 TO CHAR-NO
END-PERFORM.
```

## エンディアン変換機能

NetCOBOLでは、日本語項目のエンディアンを選択することができます。リトルエンディアンとビッグエンディアンを相互に変換する機能を提供します。

### 記述形式

#### ビッグエンディアンからリトルエンディアンへ変換

```
CALL "#NATBETOLE" USING [BY REFERENCE] 一意名.
```

#### リトルエンディアンからビッグエンディアンへ変換

```
CALL "#NATLETOBE" USING [BY REFERENCE] 一意名.
```

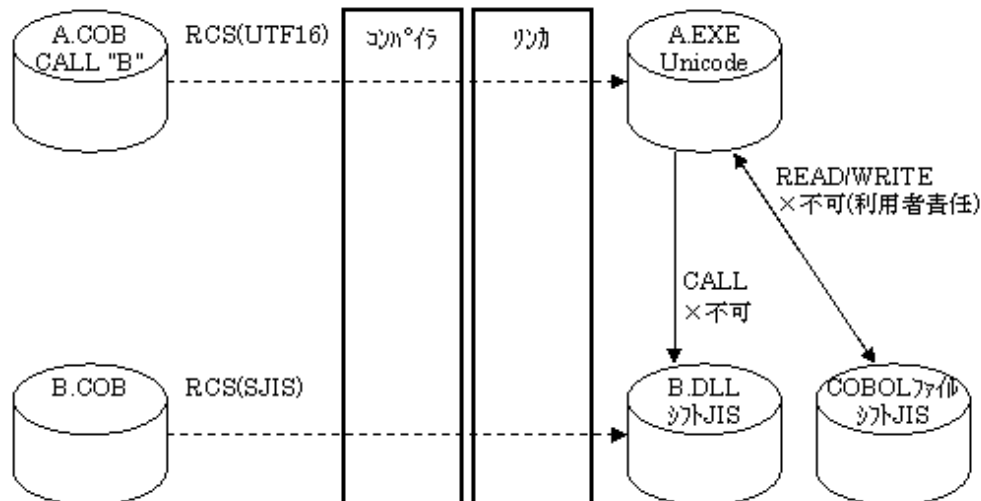
### 機能概要

- CALL文で呼び出すプログラム名が“#NATBETOLE”の場合、一意名または一意名中に含まれるすべての日本語項目および日本語編集項目をビッグエンディアンからリトルエンディアンに変換します。
- CALL文で呼び出すプログラム名が“#NATLETOBE”の場合、一意名または一意名中に含まれるすべての日本語項目および日本語編集項目をリトルエンディアンからビッグエンディアンに変換します。
- 一意名が基本項目の場合、その項目が日本語項目または日本語編集項目でなければ、何も変換されません。
- 一意名が集団項目であり、かつ、従属する項目に以下の項目が含まれている場合、その項目は変換の対象にはなりません。
  - 日本語項目または日本語編集項目ではない項目。
  - REDEFINES句が指定された項目および、その項目に従属する項目。
  - RENAMES句が指定された項目。
- 以下の場合、実行結果は保証されません。
  - CALL文で呼ぶプログラムが一意名指定であり、その一意名に#NATBETOLEまたは#NATLETOBEが設定されている。
  - 上記記述形式とは異なる書き方で呼び出している。
  - CALL文で呼び出すプログラム名が“#NATBETOLE”の場合で、一意名または一意名中に含まれる日本語項目および日本語編集項目のデータ形式がビッグエンディアンではない。

- CALL文で呼び出すプログラム名が“#NATLETOBE”の場合で、一意名または一意名中に含まれる日本語項目および日本語編集項目のデータ形式がリトルエンディアンではない。
- 一意名が集团項目であり、従属する項目にDEPENDING指定つきのOCCURS句を持つ項目が存在する。
- 一意名が定数節に定義されたデータ項目である。
- 一意名が部分参照付けされたデータ項目である。

## 25.1.4 コード系の混在

NetCOBOLでは実行単位内でコード系が混在することを許しません。以下の図のように、コード系の異なるDLLを呼び出した場合、実行時エラーとなります。



### 参考

コード系の混在は、1つの実行単位内で複数のコード系データが流れるため、動作異常の原因になります。また、デバッグの際の原因特定も困難を極め、将来的な拡張性も失われるため、NetCOBOLでは禁止しています。

## 25.1.5 資産移行支援

シフトJISで運用していたアプリケーションをUnicodeへ移行する場合、同時にCOBOLファイルの移行が必要になることがあります。NetCOBOLでは、関連製品であるSIMPLIA/TF-MDPORTを利用することによって、COBOLファイルのシフトJIS→Unicode変換を実現しました。詳細は、“SIMPLIA/TF-MDPORT オンラインマニュアル”を参照してください。

## 25.2 Unicodeアプリケーションの作成

ここでは、具体的にUnicodeアプリケーションを作成する手順を説明します。なお、実際にコーディングする際にはいくつか注意点があるので、“[25.3 コーディング上の注意点](#)”を一読の上、作業を開始してください。

### プログラムの作成、編集

ソースファイルや登録集ファイルはUnicode(UTF-8)に加えて、シフトJISでも作成できます。COBOL開発環境もしくは、お好みのエディタを利用してプログラムを作成、編集してください。

ただし、ソースファイルおよび登録集ファイルの物理ファイル名はフォルダ名を含めてシフトJISの範囲内で作成してください。

### 翻訳

ソースファイルおよび登録集ファイルのコード系を翻訳オプションSCSで、実行時の動作コードを翻訳オプションRCSで指定してください。

[参照]“[A.2.45 SCS\(ソースファイルのコード系\)](#)”、“[A.2.40 RCS\(実行時コード系の指定\)](#)”

## リンク

リンク時は特に注意すべき点はありません。Unicodeアプリケーションで特別に必要となるライブラリやリンクオプションはないため、シフトJISと同じ要領でリンクできます。

## 実行

実行時も特に注意すべき点はありません。シフトJISと同じ要領で実行するだけでUnicodeデータが扱えます。

## デバッグ

対話型デバッガを利用してUnicodeアプリケーションのデバッグを行うことができます。基本的には、シフトJISアプリケーションと同じ要領でデバッグできます。ただし、いくつか注意点があります。詳細は、“[20.6.5 デバッガのUnicode対応](#)”を参照してください。

# 25.3 コーディング上の注意点

ここでは、言語要素ごとにUnicode化の注意点やポイントを説明します。

## 25.3.1 利用者語

ソースファイルおよび登録集ファイルをUnicode(UTF-8)で作成することはできますが、以下の利用者語および文字定数にはシフトJIS範囲内の文字しか指定できません。

- ・ プログラム名
- ・ エントリ名
- ・ クラス名
- ・ メソッド名
- ・ プロパティ名
- ・ CALL 文のプログラム名
- ・ CANCEL 文のプログラム名
- ・ INVOKE 文のクラス名およびメソッド名

また、上記以外の利用者語についても、サロゲートペアの文字は使用できません。

## 25.3.2 文字定数

### 日本語16進文字定数

日本語16進文字定数はソース記述上、UTF-16のビッグエンディアンで記述してください。

```
WORKING-STORAGE SECTION.
01 NAME PIC N(3) VALUE NC"富士通".
:
IF NAME = NX"5BCC58EB901A" THEN DISPLAY "OK!!".
```

### 表意定数

表意定数の考え方はシフトJISと同じで、作用対象によって決まります。

たとえば、表意定数SPACEは、作用対象が英数字項目の場合はASCII空白(半角空白)となり、日本語項目の場合は全角空白になります。

## 25.3.3 項目の再定義

日本語項目を英数字項目で再定義(REDEFINES句)している、またはその逆の場合、注意が必要です。

```

WORKING-STORAGE SECTION.
01 PERSON.
 02 AGE PIC 9(3).
 02 NAME PIC N(8).
 02 NAME-X REDEFINES NAME PIC X(16).
01 TMP-NAME PIC X(16).
 :
 MOVE NAME-X TO TMP-NAME.
 DISPLAY TMP-NAME. ←文字化け

```

Unicodeでは、日本語項目(UTF-16)と英数字項目(UTF-8)の表現形式は全く異なります。このため、再定義によって同一のデータを別の字類で参照する場合、作用対象に合わせたデータ変換が必要になることがあります。データ変換には、NATIONAL-OF関数およびDISPLAY-OF関数を利用してください。

```

01 TMP-NAME PIC X(24).
 :
 MOVE FUNCTION DISPLAY-OF(NAME) TO TMP-NAME.
 DISPLAY TMP-NAME.

```

## 25.3.4 転記

### 集団項目転記

日本語項目を含む集団項目を転記に使用する場合、注意が必要です。

```

WORKING-STORAGE SECTION.
01 PERSON.
 02 AGE PIC 9(3).
 02 NAME PIC N(8).
01 TMP-AREA PIC X(80).
 :
 MOVE PERSON TO TMP-AREA.
 :
 MOVE TMP-AREA TO PERSON. ...[1]
 DISPLAY "DATA = " TMP-AREA. ...[2]

```

上記の例で、TMP-AREAを一時的な作業域として使う[1]のであれば特に問題ありませんが、直接データを参照する[2]場合、表現形式の異なるデータが混在していることから、正しく表示されません。このような場合、一時域を元(PERSON)のデータ構造に合わせてください。

### 空白づめ

COBOLでは、文字転記の際、受取り側項目が送出し側項目よりも大きい場合、受取り側項目の字類に合わせて空白づめを行います。Unicodeでは、日本語転記の際の空白づめにはシフトJISと同様に全角空白が使用されます。

```

WORKING-STORAGE SECTION.
01 PIC-N PIC N(10) VALUE NC"Fujitsu". ...[1]
01 PIC-X PIC X(10).
 :
 MOVE FUNCTION DISPLAY-OF(PIC-N) TO PIC-X. ...[2]
 IF PIC-X = "Fujitsu" THEN DISPLAY "OK!!". ...[3]

```

上記の例で、[3]の比較条件は真が成立するように見えますが、実際には偽が成立します。PIC-Nには3文字の全角空白がつけられる[1]ため、DISPLAY-OF関数の返却値にも全角空白が含まれます[2]。その状態で半角空白がつけられた文字定数と比較するため、空白のコードが異なり、条件式の結果は偽となります。上記の例の場合は、VALUE句の日本語定数[1]に明示的に半角空白をつけることで回避できます。

なお、文字比較についても、同様の注意が必要です。



## 文字の泣き別れ

日本語項目にサロゲートペアの文字を格納する際は、2けたの領域が必要となります。これは、表示などの際は1文字に見えますが、言語仕様上は2文字の扱いとなるため、転記や比較、部分参照などの局面で1文字が分割され、不正な文字データとなる可能性があります。このように、1文字が分割された状態を文字の泣き別れと呼びます。

このような不正データを作らないために、以下の注意が必要です。

- 十分な領域を用意する
- “[25.1.3 言語要素](#)”で説明した字類条件を利用してサロゲートペアを除外する、または、意識して文字列操作する

なお、英数字項目については、Unicodeに限らず、同様の注意が必要です。

## 25.3.5 比較

### 集団項目比較

集団項目比較の場合、事実上、字類の異なる項目どうしの比較が可能になります。Unicodeでは表現形式が異なるため、注意が必要です。

```
WORKING-STORAGE SECTION.
01 DATA-X.
 02 NAME PIC X(6) VALUE "日本". *> X"E697A5E69CAC"
01 DATA-N.
 02 NAME PIC N(2) VALUE NC"日本". *> X"65E5672C"
 :
 IF DATA-X = DATA-N THEN DISPLAY "OK??".
```

集団項目比較を行う場合、作用対象のデータ構造(宣言)は同じにしてください。

### 大小比較

NetCOBOLでは、日本語項目のエンディアンを選択することができますが、リトルエンディアンを選択(省略値)した場合、日本語項目を含む集団項目を大小比較する際に注意が必要です。

```
WORKING-STORAGE SECTION.
01 GRP.
 02 SMALL PIC N(1) VALUE NC"あ". *> X"4230"
01 LARGE PIC N(1) VALUE NC"A". *> X"21FF"
 :
 IF GRP < LARGE THEN DISPLAY "OK??". ...[1]
 IF SMALL < LARGE THEN DISPLAY "OK!!". ...[2]
```

[1]の場合、左辺は集団項目のため、左辺右辺共にエンディアンは変換せずに、リトルエンディアンのままの状態で大小比較します。その結果、左辺の方が大きくなり、真は成立しません。

[2]の場合は、左辺、右辺ともに日本語項目のため、コンパイラが双方ともビッグエンディアンに変換した後で大小比較するので正しい大小比較が行えます。

大小比較では、作用対象の字類およびデータ構造を合わせてください。

実際のコーディングでは、上記の例のような使い方はまれですが、同様なことが索引ファイルのキー、整列併合用ファイルのキー、SEARCH ALLのキー指定などにも当てはまります。これらの局面で日本語を含む集団項目を使用する場合には注意してください。

## 25.3.6 ACCEPT/DISPLAY文

### 小入出力

UnicodeのデータをACCEPT文、DISPLAY文を使用して入出力できますが、作用対象が日本語を含む集団項目の場合に注意が必要です。

```
WORKING-STORAGE SECTION.
01 PERSONAL-DATA.
 02 NAME PIC N(8).
```

```

02 TEL PIC 9(10).
:
DISPLAY PERSONAL-DATA. ...[1]
DISPLAY NAME TEL. ...[2]

```

Unicodeは字類によって表現形式が異なるため、日本語が含まれる集団項目をDISPLAY文で表示する場合、文字化けします[1]。このような場合は基本項目ごとに指定してください[2]。

また、日本語項目が含まれる集団項目を指定してACCEPT文によりデータを読み込む場合は、そのデータは、RCS翻訳オプションによって決定した実行時コードの英数字項目の文字コードで格納されます。

なお、ACCEPT文、DISPLAY文の入出力先をファイルにした場合、そのファイルの表現形式はUTF-8になります。



## 注意

行順ファイルで作成するファイルは、レコード定義の字類が日本語で統一されている場合は、UTF-16となります。行順ファイルとして作成したファイルに対して小入出力機能を使用する、またはその逆の場合、コード系の違いに注意してください。

[参照]“25.3.7 COBOLファイル”

## その他

システムコンソールを使用した小入出力機能では、シフトJIS範囲内の文字しか利用できません。

## 25.3.7 COBOLファイル

レコード順ファイル、相対ファイル、索引ファイル、行順ファイルおよび表示ファイル(ACM/APL)はUnicodeデータを何も加工せずに入出力しますが、印刷ファイルおよび表示ファイル(DSP/PRT)は各項目の字類に合わせて出力時にコード変換処理を行います。

以下、注意点をまとめます。

### ファイル識別名 (全ファイル)

ファイル識別名にデータ名を指定している場合で、かつ、そのデータ名が日本語項目を含む集団項目だった場合、Unicodeでは翻訳時エラーになります。

```

FILE-CONTROL.
 SELECT OUTFILE ASSIGN TO FILE-NAME.
:
01 FILE-NAME. →翻訳エラー
02 F-PATH PIC X(3).
02 F-NAME PIC N(4) VALUE NC"ファイル".
:
MOVE "C:¥" TO F-PATH.
OPEN OUTPUT OUTFILE.

```

これは、異なる表現形式の混在によるファイル名の文字化けを防ぐためです。ファイル識別名に集団項目を指定する場合は字類を英数字で統一してください。

### 行順ファイル

行順ファイルは、各種テキストエディタで表示・編集可能な形式であることが前提となるため、ひとつのファイルはひとつの表現形式で統一する必要があります。つまり、レコードを構成する各項目の字類を統一しなければなりません。NetCOBOLでは、レコード定義の字類が英数字で統一されている場合はUTF-8で、字類が日本語で統一されている場合はUTF-16で入出力します。以下の例のように字類が混在する場合、翻訳時エラーが出力されるため、用途に合わせて字類を統一してください。

```

FILE-CONTROL.
 SELECT OUTFILE ASSIGN TO "data.txt"
 ORGANIZATION IS LINE SEQUENTIAL.
:
FD OUTFILE. →翻訳時エラー
01 OUT-REC.

```

```
02 REC-ID PIC X(4).
02 REC-DATA PIC N(20).
```

作成された行順ファイルは、Unicodeを扱えるエディタで参照および更新ができます。

## 索引ファイル

索引キーに集団項目を指定し、かつ、その集団項目が日本語項目を含む場合、注意が必要です。

```
FILE-CONTROL.
 SELECT IX-FILE ASSIGN TO F-NAME
 ORGANIZATION IS INDEXED
 RECORD KEY IS IX-KEY.
 :
FD IX-FILE.
01 IX-REC.
 02 IX-KEY.
 03 KEY1 PIC X(4).
 03 KEY2 PIC N(8).
 02 IX-DATA PIC X(80).
```

日本語項目のエンディアンでリトルエンディアンを選択した場合、データは上位と下位バイトが逆転した形式で格納されます。これを集団項目で参照した場合、逆転した形式のまま処理されるため、START文などで意図したレコードに位置づけられない可能性があります。

このような場合、翻訳時に警告の意味でWレベルエラーを出力するので、必要に応じて、副レコードキーを利用するなどの修正をしてください。

## 印刷ファイル

印刷ファイルの場合、レコード内の字類の統一は不要です。各項目の字類に合わせてCOBOLランタイムシステムがコード変換するため、表現形式が混在しても問題なく動作します。

```
FILE-CONTROL.
 SELECT OUT-FILE ASSIGN TO PRTFILE.
 :
FILE SECTION.
FD OUT-FILE.
01 OUT-REC PIC X(80).
WORKING-STORAGE SECTION.
01 PRT-DATA CHARACTER TYPE IS MODE-1.
 02 PRT-NO PIC 9(4).
 02 PRT-ID PIC X(4).
 02 PRT-NAME PIC N(20).
 :
 WRITE OUT-REC FROM PRT-DATA AFTER ADVANCING 1 LINE.
```

ただし、日本語項目を含む集団項目を転記の受取り側項目に使用した場合、集団項目に字類と合わないコードの空白づめが行われます。これにより、シフトJISでは印刷可能であったものが、Unicodeでは文字化けなどの意図しない印刷結果になることがあります。

```
FILE-CONTROL.
 SELECT OUT-FILE ASSIGN TO PRTFILE.
 :
FILE SECTION.
FD OUT-FILE.
01 OUT-REC CHARACTER TYPE IS MODE-1.
 02 OUT-DATA PIC N(40).
WORKING-STORAGE SECTION.
01 PRT-DATA CHARACTER TYPE IS MODE-1.
 02 PRT-NO PIC N(4).
 02 PRT-ID PIC N(4).
 02 PRT-NAME PIC N(20).
 :
 MOVE PRT-DATA TO OUT-REC. ... [1]
```

```

WRITE OUT-REC AFTER ADVANCING 1 LINE.
*
MOVE NO"あいうえお" TO OUT-REC. ... [2]
WRITE OUT-REC AFTER ADVANCING 1 LINE.

```

受取り側項目が送出し側項目よりも大きい場合[1]、集団項目転記の規則により、半角空白が空白づめされます。このため、OUT-DATAには、UTF-16とUTF-8のデータが混在して格納されます。WRITE文を実行すると、OUT-DATAはUTF-16のデータとして扱われるため、UTF-8のデータが格納された部分の印刷結果は文字化けします。[2]の場合も同様です。

このような場合は、以下の例のように明示的に日本語項目を転記の対象に指定することで回避できます。

```

:
WRITE OUT-REC FROM PRT-DATA AFTER ADVANCING 1 LINE. ... [3]
*
MOVE NO"あいうえお" TO OUT-DATA.
WRITE OUT-REC AFTER ADVANCING 1 LINE. ... [4]

```

FROM句指定のWRITE文を使用した場合[3]は、FROM句に指定したデータ項目の字類に従って印字されます。また、転記の受取り側を日本語項目にした場合[4]は、全角空白が空白づめされるため、日本語項目にUTF-8のデータが混在することはありません。

## 画面帳票定義体 (FORMAT句付き印刷ファイル、表示ファイル(DSP/PRT))

画面帳票定義体に定義した英数字項目には、1バイトコードで表現される文字しか格納できません。画面帳票定義体に定義した英数字項目に、日本語文字(αなどの一部記号類、半角カナを含む)を格納して入出力を行うと、Unicodeでは意図した結果が得られません。

```

FILE-CONTROL.
 SELECT IO-FILE ASSIGN TO GS-PRTF
 SYMBOLIC DESTINATION IS "DSP"
:
FILE SECTION.
FD IO-FILE.
 COPY 画面帳票定義体名 OF XMDLIB.
(01 表示レコード名.) (注)
(02 DATA-1 PIC X(10).)
:
MOVE "ABCあいうエオ" TO DATA-1. ...[1]

```

注：()内はCOPY文の展開を表します。

[1]のように日本語文字が混在するデータを格納したい場合、画面帳票定義体には、英数字項目ではなく混在項目を定義します。定義した項目が英数字項目、混在項目のどちらでも、COPY文で展開されるデータの属性は英数字項目となりますが、実行時の扱いが異なるため、混在項目で定義されている場合のみ問題なく動作します。

## 25.3.8 スクリーン操作機能

スクリーン操作機能で全角文字を扱う場合は、日本語項目を使用してください。



サロゲートペアの文字は使用できません。

## 25.4 実行時の注意点

### 25.4.1 メッセージを出力するファイル

動作モードがUnicodeの場合、実行時メッセージを出力するファイル(環境変数情報@MessOutFileにより指定)のコード系は以下になります。[参照]“5.4.1.59 @MessOutFile(メッセージを出力するファイルの指定)”

- ・ 既存ファイルに追加書きする場合は、既存ファイルのコード系になります。

- ・ 新規ファイルに出力する場合は、UTF-8になります。

## 25.4.2 フォントについて

以下の機能を使用する場合、Unicodeに対応したフォントを使用してください。

### COBOLコンソールウィンドウ

環境変数情報@CnslFontに、Unicodeに対応したフォントを指定してください。

指定方法は、“[5.4.1.52 @CnslFont \(コンソールウィンドウのフォントの指定\)](#)”を参照してください。

### スクリーンウィンドウ

環境変数情報@ScrnFontに、Unicodeに対応したフォントを指定してください。

指定方法は、“[5.4.1.65 @ScrnFont \(スクリーン操作で使用するフォントの指定\)](#)”を参照してください。

### FORMAT句なし印刷ファイル

環境変数情報@PrinterFontNameおよびフォントテーブルに指定するフォントフェース名に、Unicodeに対応したフォントを指定してください。

指定方法は、“[5.4.1.63 @PrinterFontName \(印刷ファイルで使用するフォントの指定\)](#)”および、“[8.1.14 フォントテーブル](#)”を参照してください。

### FORMAT句付き印刷ファイルおよび表示ファイル

プリンタ情報ファイル、ウィンドウ情報ファイルおよびフォントテーブルに、Unicodeに対応したフォントを指定してください。

指定方法は、“MeFtのオンラインマニュアル”および、“[8.1.14 フォントテーブル](#)”を参照してください。

### TRACE機能、COUNT機能が出力するファイル

動作モードがUnicodeの場合、TRACE機能およびCOUNT機能が出力するファイルのコード系は、以下になります。

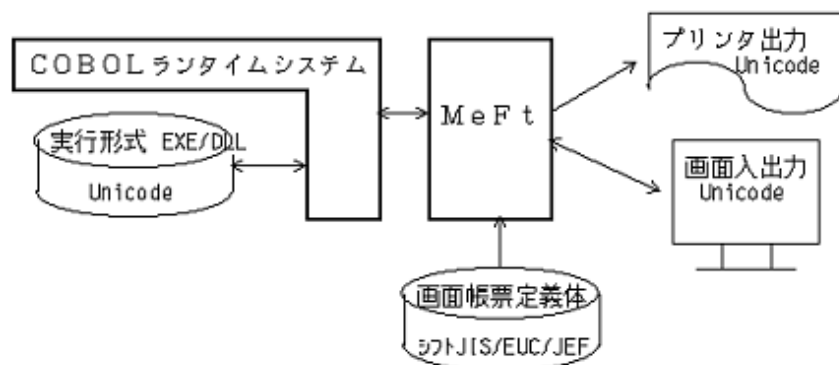
- ー 既存ファイルに追加書きする場合は、既存ファイルのコード系になります (COUNT機能のみ)。
- ー 新規ファイルに出力する場合は、UTF-8になります。

## 25.5 関連製品連携

### 25.5.1 FORM/MeFt

FORMを使用して作成した画面帳票定義体をUnicodeアプリケーションで使用することができます。定義体は既存のものを使用することも、新規に作成することもできます。また、実行時にMeFtが自動でUnicodeへコード変換するため、定義体のコード系は、シフトJIS、EUC、JEFのどれでも利用できます。

入出力するデータのコード系がUTF-16の場合、使用する製品のバージョンレベルによって、サロゲートペアの利用可否が異なります。MeFtのマニュアルでご確認ください。



FORMAT句付き印刷ファイル、表示ファイル(帳票印刷)、表示ファイル(画面入出力)の利用方法および機能範囲については、シフトJISと同じです。“[第8章 印刷処理](#)”および“[第9章 画面を使った入出力](#)”を参照してください。

## 25.5.2 プリコンパイラ

プリコンパイラを利用してリモートデータベースにアクセスする場合、Unicodeデータが利用できるかどうかは、プリコンパイラの仕様によって異なります。各データベースのプリコンパイラのマニュアルでご確認ください。

## 25.5.3 ODBC

埋込みSQL文を使用したリモートデータベースアクセス(ODBC)機能を利用して、データベースとUnicodeデータをやりとりすることができます。

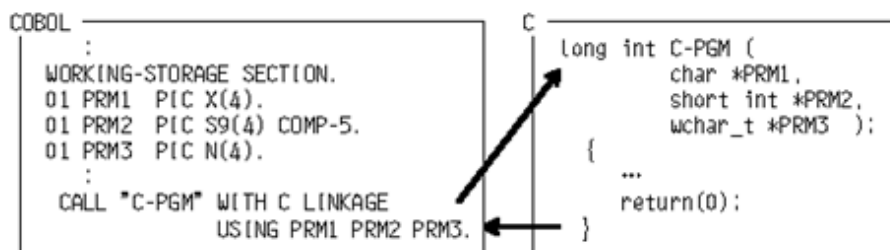
以下にUnicodeデータの入出力を行う場合の注意事項を示します。

- アクセス対象となるデータソース(データベース、ODBCドライバおよびその他ネットワークコンポーネント)がUnicodeデータの入出力をサポートしている必要があります。
- ホスト変数を使用してUnicodeデータの入出力を行う場合は、日本語項目のホスト変数を使用してください。COBOLでは、日本語項目をODBCで定義されているUnicodeデータ型にマッピングしています。[参照][“22.2.11 ODBCで扱うデータとの対応”](#)
- 英数字項目のホスト変数で扱うことができるデータは、英数字項目だけです。日本語文字を扱うことはできません。英数字項目のホスト変数を使用して、日本語文字の入出力を行った場合、入出力の結果は保証されません。
- 翻訳オプションSCS(SJIS)を指定している時、埋込みSQL文の値指定により、データベースのUnicodeデータ型列にUnicodeデータを入力する場合は、文字列定数または日本語文字列定数をシフトJISデータで記述してください。この場合、データソースがシフトJISデータをUnicodeデータに変換します。

## 25.5.4 他言語間結合

### C言語

C言語では、UTF-16のデータを扱うためにwchar\_t型をサポートしています。よって、COBOLの日本語項目とC言語のwchar\_t型との間でデータの送受ができます。また、ASCII文字の範囲であれば、COBOLの英数字項目とC言語のchar型との間でのデータ送受もできます。もちろん、数字系のデータはシフトJISと同じ要領で送受できます。



Windowsシステム関数の呼出しもできます。WindowsはUnicodeで動作するシステム関数(関数名の末尾がW)を用意しています。この関数との間で文字データを送受する場合は、日本語項目を使用してください。

```

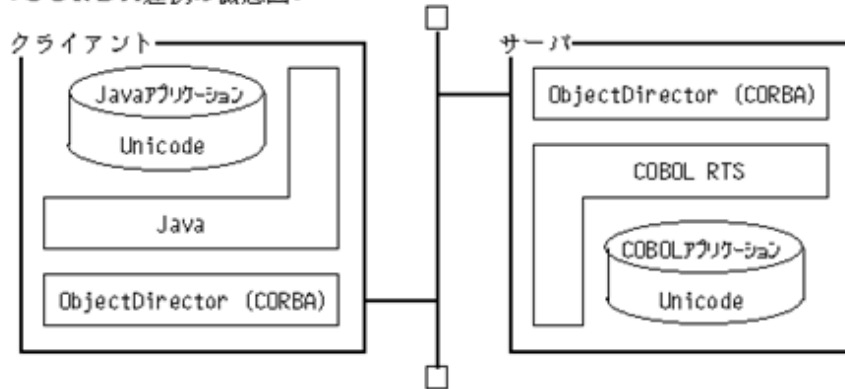
:
WORKING-STORAGE SECTION.
01 M-TITLE PIC N(20).
01 M-MSG PIC N(20).
01 RET-CODE PIC S9(9) COMP-5.
:
** パラメタの設定
 MOVE NC"COBOL Unicodeアプリ" & NX"0000" TO M-TITLE.
 MOVE NC"メッセージの表示" & NX"0000" TO M-MSG.
**
** システム関数の呼出し
 CALL "MessageBoxW" WITH STDALL USING BY VALUE 0
 BY REFERENCE M-MSG
```

BY REFERENCE M-TITLE  
BY VALUE 1  
RETURNING RET-CODE.

## Java

Javaと連携する場合、一般的にはCORBAを利用しますが、Jアダプタクラスを利用してCOBOLから直接Javaのクラスを呼び出すこともできます。いずれの場合もUnicodeデータを送受することができます。

<CORBA連携の概念図>



## 25.5.5 他のファイルシステム

### Btrieveファイル

BtrieveファイルはUnicodeアプリケーションでも使用することができます。使用方法については、以下の注意点を除いてCOBOLファイルと同じです。“[25.3.7 COBOLファイル](#)”を参考にしてください。



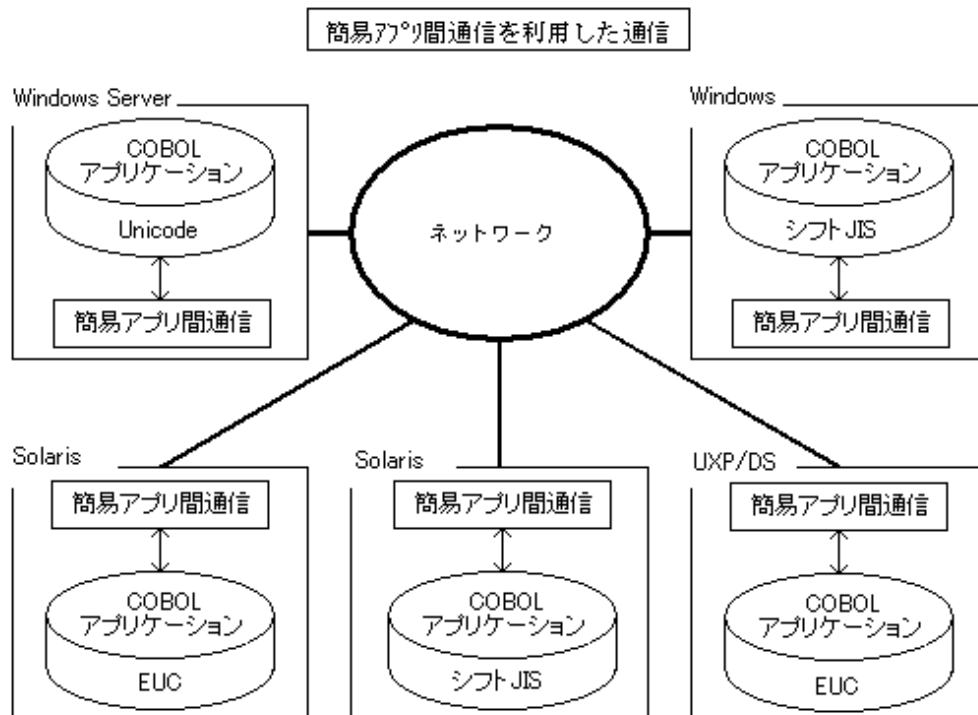
#### 注意

日本語項目のエンディアンにリトルエンディアンを選択し、かつ、索引編成のBtrieveファイルで、索引キー(基本項目、集団項目のいずれの場合も)に日本語項目が含まれる場合、日本語項目に格納されたデータはリトルエンディアンのままで比較処理されるため、意図したレコードに位置づけられない可能性があります。

## 25.5.6 通信機能

表示ファイル機能を利用した通信機能(ACM)、簡易アプリ間通信機能ともにUnicodeアプリケーションで使用することができます。

ただし、簡易アプリ間通信機能では、各関数のパラメタとして使用するサーバ名および論理宛て先名にはシフトJIS範囲の文字を使用してください。



なお、シフトJISのアプリケーションの場合も同様ですが、メッセージおよび転送データのコード系は利用者責任で相手先に合わせてください。

## 25.5.7 Web連携

NetCOBOLが提供するCOBOL WebサブルーチンはすべてUnicodeをサポートしています。これらを利用してUnicodeで動作するWebアプリケーションが作成できます。

詳細は、“COBOL Webサブルーチン使用手引書”の“WebアプリケーションにおけるUnicodeの利用”を参照してください。

## 25.5.8 分散オブジェクト

COMやCORBAを利用した分散オブジェクト環境でUnicodeを利用することもできます。

詳細は、“[第26章 COM機能](#)”および“[第27章 CORBAアプリケーション](#)”を参照してください。



## 第26章 COM機能

本章では、NetCOBOLでのCOM(Component Object Model)機能について説明します。

なお、DCOM機能を利用したCOMサーバ機能は、サーバ向け運用環境製品固有の機能です。

### 26.1 COM機能を利用したアプリケーション

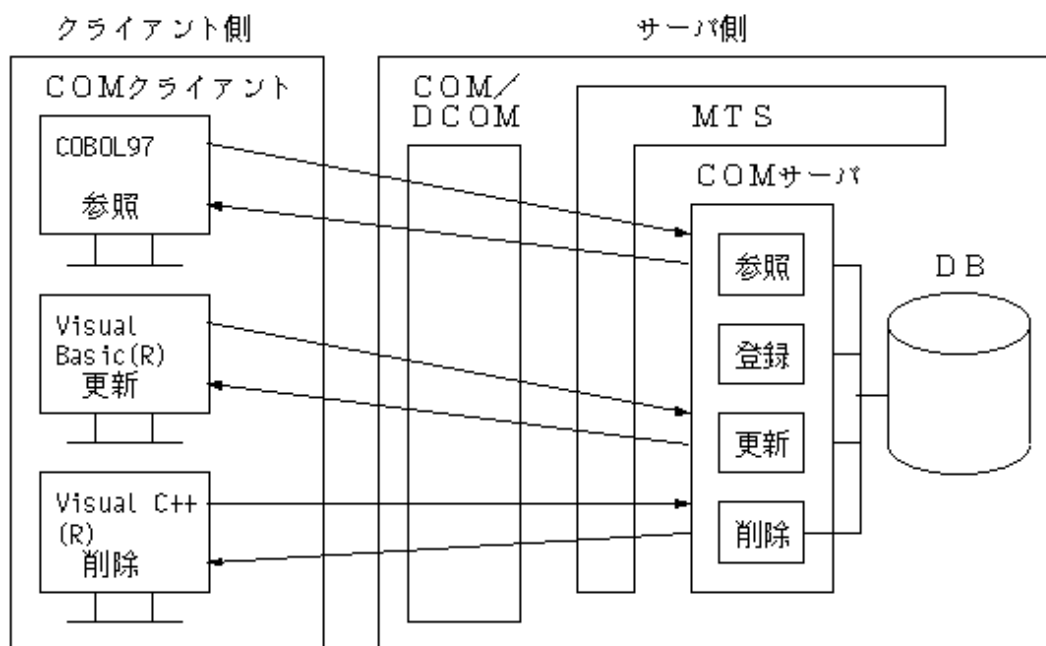
COMとは、ソフトウェアが他のソフトウェアにサービスを提供するための共通な定義です。サービスを提供する側をCOMサーバ、供給される側をCOMクライアントと呼びます。このCOMの共通定義に従ったアプリケーションを作成することで、クライアントやサーバの開発言語に依存しない、言語の特性を活かしたシステムを構築できるメリットがあります。このメリットを活かすことで、両方ともCOBOLであるシステムの組み合わせはもちろんのこと、画面入力するアプリケーション(COMクライアント)をVisual Basic、基幹業務のアプリケーション(COMサーバ)をCOBOLで作成するといったように、それぞれの開発に合った言語を自由に選択してシステムを構築できます。

またCOM機能を利用するアプリケーションは、同じコンピュータ(COM)上のシステムなのか、ネットワークで接続した別のコンピュータ(DCOM:Distributed COM)上のシステムなのかを意識する必要がありません。これは、OSの一部であるCOM/DCOMシステムがその違いを吸収しているためであり、システムの形態がアプリケーションに影響しないというメリットがあります。これにより、業務アプリケーションを部品(コンポーネント)として提供または利用できるようになり、さらに進んだ使い方としてWebサーバと組み合わせたシステムも構築可能になります。

#### 26.1.1 一般的なシステム形態

COM機能を利用した一般的なアプリケーションとして、複数のクライアントから画面入力し、サーバアプリケーションでデータの参照、登録、更新および削除を行い、結果をクライアント側に返却する問い合わせ業務を作成することができます。

図26.1 COMを利用した一般的なアプリケーションの形態



注意

MTS(Microsoft Transaction Server)はクライアントとサーバアプリケーションの間に入り、アプリケーションに代わってオブジェクト管理やトランザクション管理を行うため、基幹アプリケーションを作成する上で便利です。ただし、MTSはインタフェースをCOM機能に限定し

ているため、クライアントもサーバもCOM機能が必須となります。MTSを利用したアプリケーションの開発については“[26.4 DCOMとMTS](#)”を参照してください。

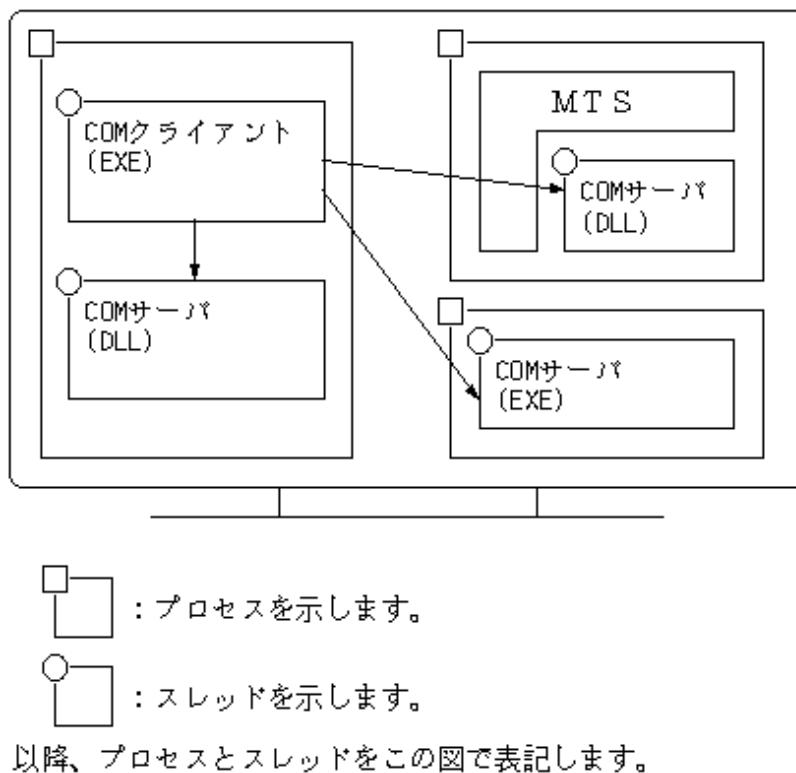
### 26.1.2 COMを利用したアプリケーション形態

COMサーバはCOMクライアントと同じコンピュータ上で動作させることはもちろん、ネットワークで接続された別のコンピュータ上でも動作させることもできます。この場合を特にDCOM(Distributed COM)と呼びます。

COMクライアントおよびCOMサーバを同一マシン上で動作させる場合、一般的に“[図26.2 同一マシン上での動作](#)”に示す運用が可能です。

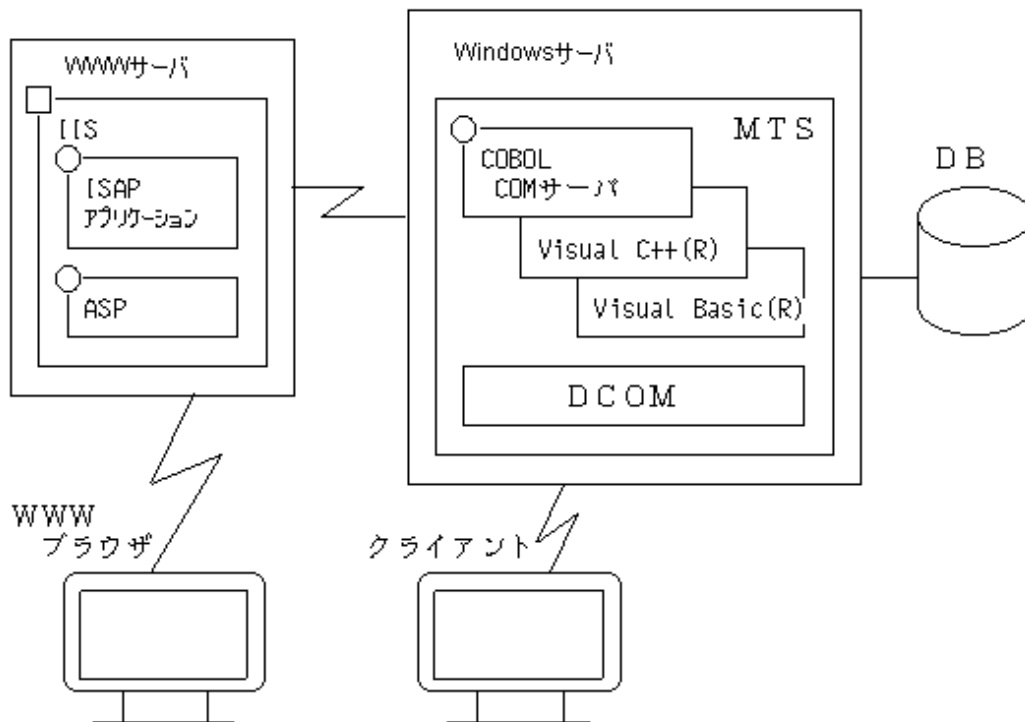
COMクライアントプログラムから、同じプロセス内にダイナミックリンクライブラリ(以降の説明では単にDLLといいます)で実装したCOMサーバをロードし連携することができます。また、MTS配下にインストールした、DLLで実装されたCOMサーバや実行可能プログラム(以降の説明では単にEXEといいます)で実装されたCOMサーバは、別プロセスとして連携することができます。

図26.2 同一マシン上での動作



COMクライアントとCOMサーバを別のマシンで動作させる場合、DCOMでは“[図26.3 DCOMを利用したシステム](#)”に示す運用が可能です。WebアプリケーションからCOMサーバを起動して、Webブラウザから利用したり、DCOMの機能を利用したりして、COMクライアントから直接COMサーバを操作することもできます。

図26.3 DCOMを利用したシステム



ブラウザ : Internet Explorer, Netscape Communicator, etc.  
IIS : Internet Information Server  
[SAP] : Internet Server API  
ASP : Active Server Page  
MTS : Microsoft Transaction Server

NetCOBOLでは、DLLで実装されたCOMサーバおよびCOMクライアントを作成することができます。

### 26.1.3 COM機能を利用するアプリケーションの作成

NetCOBOLのCOM機能を利用するアプリケーションは、COBOLのオブジェクト指向技術をベースに作成します。その理由のひとつは、COMオブジェクトの作成や利用方法が、オブジェクト指向プログラミングと高い親和性を持つためです。もうひとつの理由は、部品 (COMサーバ)として再利用するという考え方がオブジェクト指向的であるためです。ただし、オブジェクト指向プログラミングで説明している継承や多態といった複雑な知識は必要なく、クライアント側がCALL文からINVOKE文に、サーバ側がプログラム定義からクラス定義にそれぞれ書き方が変わるという感覚だけで簡単に作成できます。

オブジェクト指向技術については、“[第18章 オブジェクト指向プログラミング機能～さらに進んだ使い方～](#)”を参照してください。

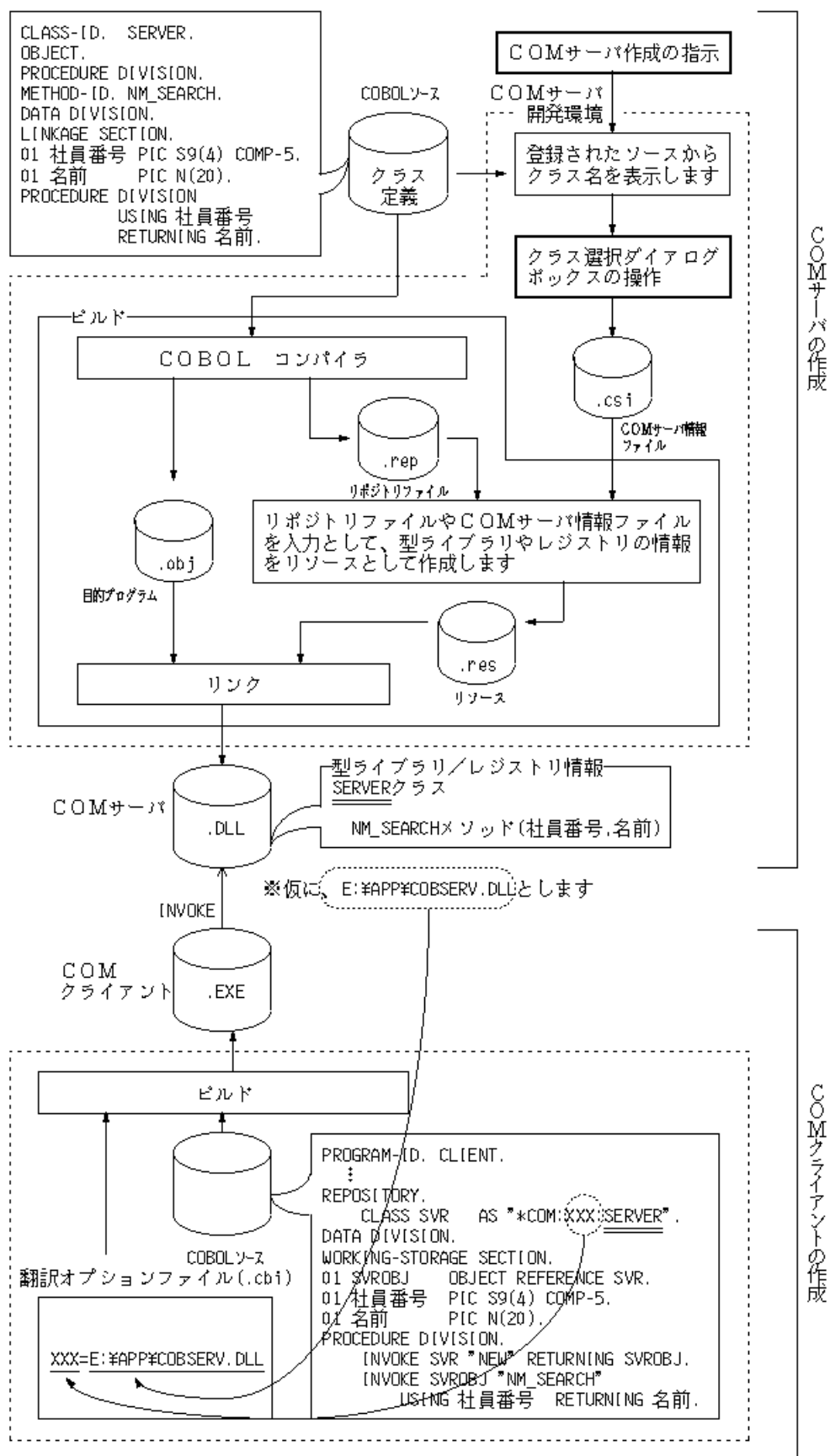
COM機能を利用するクライアントアプリケーションの作成方法については“[26.2 COMクライアント機能](#)”を参照してください。

COM機能を利用するサーバアプリケーションの作成方法については“[26.3 COMサーバ機能](#)”を参照してください。

### 26.1.4 開発環境と資源

COMサーバの開発環境および資源、COMクライアントの作成方法について説明します。

図26.4 開発環境と資源の関係



## COMサーバの開発環境および資源

COMサーバはプロジェクトをビルドすることによってDLLとして作成されます。プロジェクトの依存ファイルには、以下のファイルがあります。

- COBOLソースファイル(.cob)
- 目的プログラム(.obj)
- リポジトリファイル(.rep)
- COMサーバ情報ファイル(.csi)

COMクライアントが必要とする型ライブラリやレジストリ情報は、作成されたDLLの中にリソースとして含まれます。COMサーバの作成方法の詳細については“[26.8 開発環境](#)”を参照してください。

## COMクライアントの作成

COMクライアントを作成する場合、COMサーバとするクラスを特殊クラス化する点が、通常のオブジェクト指向プログラミングと異なります。

また、COMサーバをアーライバインド(注)する場合には、上記に示したように、型ライブラリの情報が含まれるファイルをオプションファイルに絶対パス名で指定しなければなりません。オプションファイルの設定方法については、“[26.8.1 COMサーバ名の設定](#)”を参照してください

注：アーライバインドについては、“[26.2.2 アーライバインドとレイバインド](#)”を参照してください。

## 26.2 COMクライアント機能

---

ここでは、オートメーションの機構を利用してCOMサーバと連携するCOMクライアントアプリケーションをCOBOLで作成する方法について説明します。

### 26.2.1 概要

---

NetCOBOLでは、COMサーバを利用する仕組みを提供しています。これをCOMクライアント機能と呼びます。COMクライアント機能を使うことで、NetCOBOLから、以下に示すようなCOMサーバの機能を持つ製品と簡単に連携できます。

- Microsoft Excel(以降、Excelと略記します)
- SymfoNET Object Director(以降、Object Directorと略記します)のOLEゲートウェイ
- Symfoware Serverのデータベースアクセスオブジェクト(以降、データベースアクセスオブジェクトと略記します)のCOMサーバ機能
- NetCOBOL、Visual BasicまたはVisual C++などのCOMサーバ作成機能を持つツールで作成したCOMサーバ

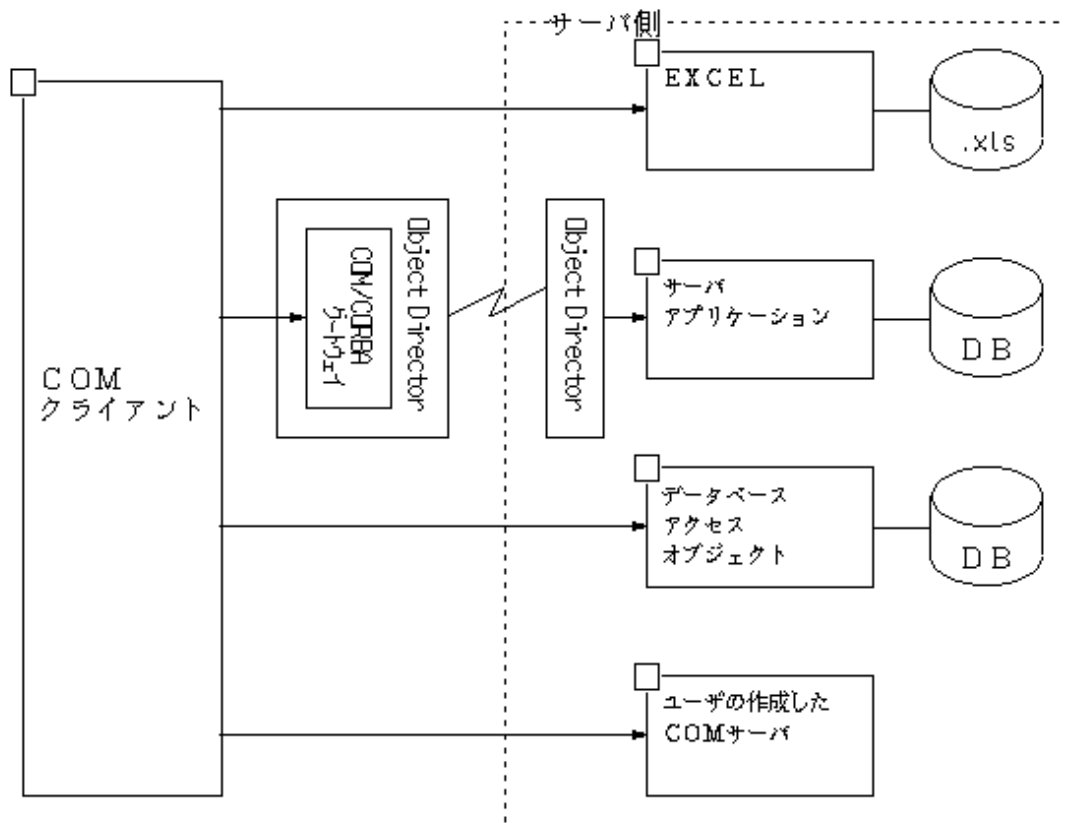


#### 参考

Visual Basicでは、ActiveX DLL作成機能でCOMサーバを作成します。この機能で定義したメソッド、プロパティをCOMクライアント機能で操作することができます。

COMクライアント機能の概要図を以下に示します。

図26.5 COMクライアント機能



COMクライアント機能では、COMサーバを特殊クラス\*COM(\*OLEも同義)として扱います。\*COMクラスはCOMサーバをCOBOLのオブジェクトとして扱うための特別なクラスです。\*COMについては、“[26.2.3 \\*COMクラス](#)”で詳しく説明します。

COMクライアント機能で扱うほとんどのデータ型は、COBOLの通常データ型に対応付けられます。ただし、配列やバリエーションのようなCOBOLのデータ型に直接対応付けられないデータ型については、それらを扱うための特殊クラスを用意しています。[参照]“[26.6.1 \\*COM-ARRAYクラス](#)”

## 26.2.2 アーリバインドとレイトバインド

バインドとはクライアント側がサーバ側のメソッドやインタフェースを確認する方法のことをいい、COMクライアントとCOMサーバとのバインド方法には、以下の2種類があります。

- ・アーリバインド
- ・レイトバインド

アーリバインドは、COMサーバが提供する型ライブラリの情報を翻訳時に参照する方法で、レイトバインドはCOMサーバの情報を実行時に問い合わせる方法です。

### 26.2.2.1 アーリバインドとレイトバインドの違い

アーリバインドとするかレイトバインドとするかは、REPOSITORY段落での特殊クラスの書き方によって決まります。

|                                            |           |
|--------------------------------------------|-----------|
| REPOSITORY.                                |           |
| CLASS LATE AS “*COM”                       | … レイトバインド |
| CLASS EARLY AS “*COM:COMSERV:APPLICATION”. | … アーリバインド |

レイトバインドは、“\*COM”とだけ記述します。

アーリバインドは、“\*COM:COMサーバ名:クラス名”と指定します。

このとき、COMサーバ名は任意の名前、クラス名は型ライブラリ上の中で識別されるクラス名です。COMサーバ名はユーザが任意に付けてよい名前ですが、型ライブラリが存在する場所と結びつけて、オプションファイルに格納されていなければなりません。[参照]“26.8.1 COMサーバ名の設定”

## 注意

- AS定数中のCOMサーバ名が小文字のとき、翻訳オプションALPHAL(ALL)を指定して翻訳すると、COMサーバ名は大文字として扱われるため、オプションファイル中のCOMサーバ名を小文字で設定した場合、意図した型ライブラリが読み込みません。AS定数中のCOMサーバ名を小文字として扱うには、ALPHAL(WORD)またはNOALPHALを指定して翻訳してください。[参照]“A.2.2 ALPHAL(英小文字の扱い)”
- ファクトリメソッド”NEW”を使用するアーリバインドクラスを定義する場合、クラス名にはcoclass名を指定してください。

アーリバインドとレイトバインドの違いについて以下の表に示します。

表26.1 アーリバインドとレイトバインドの違い

| 項目                  | レイトバインド                                        | アーリバインド                                            |
|---------------------|------------------------------------------------|----------------------------------------------------|
| REPOSITORY段落のクラス名指定 | 特殊クラスとして"*COM"("*OLE")を指定                      | 特殊クラスとして"*COM:COMサーバ名:クラス名"を指定                     |
| オブジェクトプロパティ         | 記述できない<br>INVOKE文で記述する                         | 記述できる(注1)                                          |
| 行内呼出し               | 記述できない<br>INVOKE文で記述する                         | 記述できる(注1)                                          |
| 定数パラメタ              | 記述できない<br>定数をデータ名に転記して指定する                     | 記述できる<br>直接INVOKE文のBY CONTENTや行内呼出しに定数の記述可能        |
| 型ライブラリの指定           | 指定の必要なし                                        | 必須<br>翻訳時に型ライブラリの絶対パス名を指定しなければならない                 |
| ファクトリメソッド           | CREATE-OBJECT<br>GET-OBJECT                    | NEW(注2)                                            |
| 実行性能                | アーリバインドより遅い                                    | レイトバインドより速い                                        |
| インタフェース変更の影響        | 影響が少ない<br>クライアントが使用しているインタフェースに影響がなければ再翻訳の必要なし | 影響が大きい<br>サーバにクライアントが使用しないメソッドが追加されてもクライアントの再翻訳が必要 |

### ・ 注1

オブジェクトプロパティが使用できるのは、呼び出すプロパティの型が以下に示すVT型の場合だけです。  
行内呼出しが使用できるのは、呼び出すメソッドの復帰値の型が以下に示すVT型の場合だけです。

- VT\_I2
- VT\_I4
- VT\_R4
- VT\_R8
- VT\_CY
- VT\_DISPATCH
- VT\_DATE

— VT\_BOOL

・ 注2

ファクトリメソッド”NEW”が使用できるのは、coclass名指定のアーリバインドクラスだけです。

## 26.2.2.2 アーリバインドのメリットとデメリット

### メリット

アーリバインドがレイトバインドよりも有利な点について、以下に示します。

- ・ 呼び出すメソッドがCOMサーバ内に存在するかどうかは翻訳時にわかるため、レイトバインドより実行性能が向上します。
- ・ パラメタの型が翻訳時にわかることで、クライアントのアプリケーションにとってサービス性や記述性が良くなります。具体的には、以下が可能になります。
  - パラメタの型一致検査を翻訳時にできるため、誤りを早期に検出できます。
  - オブジェクトプロパティを利用したプロパティ呼出しが記述できます。
  - 行内呼出しを使用してメソッドが呼べます。
  - メソッド呼出し時のパラメタに定数を指定できます。
- ・ 一般クラスと同じように”NEW”メソッドが使えるため、COMサーバ化(特殊クラス\*COM)するときの修正量が少なくなります。

### デメリット

アーリバインドがレイトバインドよりも不利な点について、以下に示します。

- ・ 型ライブラリを提供していないサーバとは連携できません。
- ・ クライアントの翻訳時にサーバの型ライブラリを指定する必要があります。そのためクライアント/サーバの同時開発であっても、サーバ開発が優先となります。
- ・ サーバ側にクライアント側で使用しないメソッドが追加された場合でも、クライアント側ではその型ライブラリを指定した再翻訳が必要になります。

## 26.2.2.3 バインド方法選択の基準

次のケースにあてはまるクライアントの開発には、性能や記述性のよいアーリバインドをおすすめします。ただし、サーバが型ライブラリを用意していることが前提となります。

- ・ サーバとクライアントを同じ部署で開発する、つまりサーバが変更されたタイミングでクライアントが再翻訳できるようなシステム開発の場合
- ・ サーバ側のクラスが少なく、複雑でない場合(特にCOBOLで作成するサーバ)

逆に次のようなクライアントの開発は、レイトバインドで作成することをおすすめします。

- ・ サーバのインタフェース変更が伝わりにくい、つまり他社が作った既存のCOMサーバに対する開発である場合
- ・ 多くのクラスまたは複雑なクラス構造を持つCOMサーバに対する開発である場合

## 26.2.2.4 .NET言語で作成されたCOMサーバとの連携

アーリバインドで連携する場合、対象クラスのインタフェースが、COMのデフォルトインタフェースとして公開されている必要があります。

COMサーバが.NET言語で作成されている場合、そのクラスに実装されているインタフェースと連携させるときは、COMサーバとして公開するクラスに以下のカスタム属性が付加されていなければなりません。

### System.Runtime.InteropServices.ClassInterface属性

COM公開するクラスに明示的にインタフェースを実装するような場合、この属性に `ClassInterface.None` を指定することで、クラスに対するクラスインタフェースを暗黙に生成しなくなります。

この属性を省略した場合は、暗黙的にクラスインタフェース(メンバを持たないディスパッチインタフェース)が公開され、CoClassのデフォルトインタフェースとなります。



## System.Runtime.InteropServices.ComDefaultInterface属性

この属性は、COM公開するクラスが複数のインタフェースを実装しているような場合に、既定のインタフェースを指定します。

(ComDefaultInterface属性は.NET Framework 2.0で追加された属性です。ClassInterface属性にClassInterface.Noneを指定した場合、実装されているインターフェースが単一であれば、この属性は省略しても問題ありません。)

これらの属性が指定されていない場合は、クラスに明示的にインタフェースを実装したとしても、メンバを持たない空のディスパッチインタフェースがデフォルトインタフェースとして公開されてしまうため、アールバインドでクラスのメソッドやプロパティを参照するとJMN55051Sの翻訳エラーが発生してしまいます。

このような場合、バインド方法をレイトバインドに変更するか、COMサーバを上記に従って作り直す必要があります。

.NET言語がNetCOBOL for .NETのV3.1以降の場合、上記のカスタム属性の付加方法の詳細は、“NetCOBOL for .NETの使用手引書”の“Windows版 NetCOBOLのCOM機能で利用する際の注意点”を参照してください。

## 26.2.3 \*COMクラス

\*COMクラスは、COMサーバを操作するためのクラスです。\*COMクラスを利用することで、COMサーバをCOBOLのオブジェクトであるかのように扱うことができます。

\*COMクラスにはアールバインドとレイトバインドがあるため、以降の説明では便宜上、アールバインドを指す場合には“\*COMアールバインドクラス”、レイトバインドを指す場合には“\*COMレイトバインドクラス”、両方を指す場合には“\*COMクラス”と呼びます。

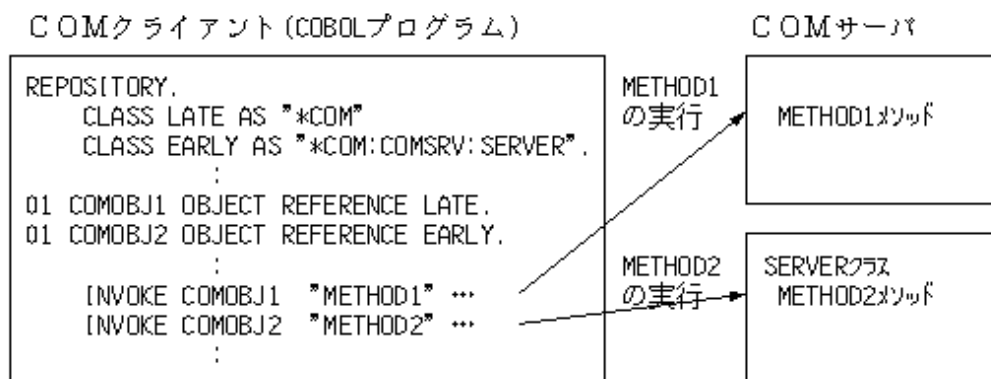


### 注意

\*COMレイトバインドクラスはV50以前では\*OLEクラスとして定義されていました。\*OLEは将来的にも利用可能ですが、今後はエンスされません。V60以降では\*COMを使用してください。

“図26.6 COMサーバの操作”で示すように、COBOLプログラムの中で\*COMクラスのオブジェクトのメソッドを呼び出すだけで、そのオブジェクトに関連付けられたCOMサーバのメソッドも呼び出せます。

図26.6 COMサーバの操作



### 26.2.3.1 プログラミング例

ここでは、オブジェクト指向を利用した簡単なアプリケーションを、COM機能を用いてクライアント/サーバアプリケーションとする例をレイトバインドおよびアールバインドに分けて説明します。

#### アプリケーションの概略

円(¥)を指定されたレートドル(\$)に変換する処理を部品化(EXCHANGEクラス)し、プログラムから部品(YENTODOLLメソッド)として呼び出します。

## プログラム定義

```
@OPTIONS MAIN(WINMAIN)
IDENTIFICATION DIVISION.
PROGRAM-ID. RATECHG.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS EXCHANGE.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 RATE PIC 999V99.
 01 W-RATE PIC S9(5) COMP-5.
 01 EXCHGOBJ OBJECT REFERENCE EXCHANGE.
 01 DOLLARS PIC S9(4)V9(2).
 01 ED-DOLLARS PIC ZZZ. 99.
PROCEDURE DIVISION.
 INVOKE EXCHANGE "NEW" RETURNING EXCHGOBJ.
 DISPLAY NG" 1ドルのレートを入力してください(999.99)".
 ACCEPT RATE.
* レートを小数部なしの5桁に変換する
 COMPUTE W-RATE = RATE * 100.
* 整数5桁に直されたレートをプロパティに設定する
 MOVE W-RATE TO EXRATE100 OF EXCHGOBJ.
 DISPLAY NG" 1999円をドルに変換する".
* 変換メソッドを呼ぶ
 INVOKE EXCHGOBJ "YENTODOLL" USING BY CONTENT 1999
 RETURNING DOLLARS.
* 編集して表示する
 MOVE DOLLARS TO ED-DOLLARS.
 DISPLAY ED-DOLLARS.
END PROGRAM RATECHG.
```

## クラス定義

```
IDENTIFICATION DIVISION.
CLASS-ID. EXCHANGE INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS FJBASE.
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 EXRATE100 PIC S9(5) COMP-5 PROPERTY.
PROCEDURE DIVISION.
METHOD-ID. YENTODOLL.
DATA DIVISION.
LINKAGE SECTION.
 01 YEN PIC S9(4) COMP-5.
 01 DOLLARS PIC S9(4)V9(2).
PROCEDURE DIVISION USING YEN
 RETURNING DOLLARS.
* レートを100で割り円をドルに換算する
 COMPUTE DOLLARS = YEN / (EXRATE100 / 100).
END METHOD YENTODOLL.
END OBJECT.
END CLASS EXCHANGE.
```

COM機能を利用する場合、プログラム定義をCOMクライアント、クラス定義をCOMサーバとして作成します。COMサーバをレイトバインドするか、アーリバインドするかによって、クライアントプログラムの書き方が異なります。

COMサーバについては“[26.3 COMサーバ機能](#)”で説明しますが、上記のクラス定義はプログラムマネージャーを用いて“COMサーバを作成する”という選択で構築を行えば、ソース記述そのままCOMサーバとなります。

## サーバをレイトバインドするクライアントプログラムの書き方

レイトバインドの場合、サーバとなるクラスを特殊化するために”\*COM”を指定します。またレイトバインドするクラスは、オブジェクト指向プログラミングの選択指定のないオブジェクト参照に対する扱いと同じように、INVOKE文のパラメタとしての定数指定やオブジェクトプロパティが記述できません。そのため次のような修正が必要になります。

### サーバをレイトバインドする場合の書き方

```
@OPTIONS MAIN(WINMAIN)
 IDENTIFICATION DIVISION.
 PROGRAM-ID. RATECHG.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
 CLASS EXCHANGE AS “*COM”. ...[1]
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 RATE PIC 999V99.
 01 W-RATE PIC S9(5) COMP-5.
 01 EXCHGOBJ OBJECT REFERENCE EXCHANGE.
 01 YEN PIC S9(4) COMP-5. ...[2]
 01 DOLLARS PIC S9(4)V9(2).
 01 ED-DOLLARS PIC ZZZ.99.
 01 PRG-ID PIC X(16) VALUE “COBSERV. EXCHANGE”. ...[3]
 PROCEDURE DIVISION.
 INVOKE EXCHANGE “CREATE-OBJECT” USING PRG-ID ...[4]
 RETURNING EXCHGOBJ.
 DISPLAY NC“ 1 ドルのレートを入力してください（999.99）”
 ACCEPT RATE.
 * レートを小数部なしの5桁に変換する
 COMPUTE W-RATE = RATE * 100.
 * 整数5桁に直されたレートをプロパティに設定する
 INVOKE EXCHANGE “SET-EXRATE100” USING W-RATE.
 DISPLAY NC“ 1999円をドルに変換する”.
 * 変換メソッドを呼ぶ
 MOVE 1999 TO YEN. ...[5]
 INVOKE EXCHGOBJ “YENTODOLL” USING YEN ...[6]
 RETURNING DOLLARS.
 * 編集して表示する
 MOVE DOLLARS TO ED-DOLLARS.
 DISPLAY ED-DOLLARS.
 END PROGRAM RATECHG.
```

[1] COMサーバとなるEXCHANGEクラスを特殊クラス化するため、“\*COM”を指定する。

[2] レイトバインドとしたCOMサーバのメソッドを呼び出すINVOKE文の引数に定数が指定できないため、定数を転記するためのデータ項目を宣言する。

[3] ProgIDを指定するためのデータ項目を宣言する。

[4] NEWメソッドの代わりに、CREATE-OBJECTメソッドを呼び出す。このとき、パラメタにProgIDを指定する。

[5] INVOKE文のパラメタに定数が指定できないためデータ項目に定数を転記する。

[6] INVOKE文のパラメタにデータ項目を指定する。

## サーバをアーリバインドするクライアントプログラムの書き方

アーリバインドの場合、サーバのクラスを特殊化するために”\*COM:COMサーバ名:クラス名”を指定します。翻訳時には、COMサーバ名と型の情報が格納されたファイル(型ライブラリ)を関連付けて登録しておく必要があります。

## サーバをアーライバインドする場合の書き方

```
@OPTIONS MAIN(WINMAIN)
IDENTIFICATION DIVISION.
PROGRAM-ID. RATECHG.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS EXCHANGE AS "*COM:COBSERV:EXCHANGE". ...[1]
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 RATE PIC 999V99.
 01 W-RATE PIC S9(5) COMP-5.
 01 EXCHGOBJ OBJECT REFERENCE EXCHANGE.
 01 DOLLARS PIC S9(4)V9(2).
 01 ED-DOLLARS PIC ZZZ.99.
PROCEDURE DIVISION.
 INVOKE EXCHANGE "NEW" RETURNING EXCHGOBJ.
 DISPLAY NC" 1 ドルのレートを入力してください (9 9 9 . 9 9) "
 ACCEPT RATE.
* レートを小数部なしの5桁に変換する
 COMPUTE W-RATE = RATE * 100.
* 整数5桁に直されたレートをプロパティに設定する
 MOVE W-RATE TO EXRATE100 OF EXCHGOBJ.
 DISPLAY " 1 9 9 9 円をドルに変換する".
* 変換メソッドを呼ぶ
 INVOKE EXCHGOBJ "YENTODOLL" USING BY CONTENT 1999
 RETURNING DOLLARS.
* 編集して表示する
 MOVE DOLLARS TO ED-DOLLARS.
 DISPLAY ED-DOLLARS.
END PROGRAM RATECHG.
```

[1] COMサーバとなるEXCHANGEクラスを特殊クラス化するため、“\*COM:COMサーバ名(任意):クラス名”を指定する。

アーライバインドするクライアントプログラムでは、プログラムのリポジトリ段落を変更するだけで簡単にCOMクライアントプログラムが作成できます。つまり、オブジェクト指向プログラミングで作成したプログラムやクラス定義をCOM機能を利用しない環境で動作保証しておけば、そのプログラム記述の一部を変更するだけで、COM機能を利用したクライアント/サーバアプリケーションとして動作することが可能になります。

## クライアントプログラムをVisual Basicとする場合

EXCHANGEクラスをサーバとするクライアントプログラムをVisual Basicで書いた例を以下に示します。このプログラムはACCEPT/DISPLAYがGUI形式である以外はCOBOLのクライアントプログラムと同じ機能を持ちます。COMの共通定義に従ったアプリケーションの開発はクライアントやサーバの開発言語に依存しないため、メソッドとその引数およびプロパティの情報がわかれば、クライアントを他言語でプログラミングすることもできます。

図26.7 Visual Basicの場合

Project1 - Client (コード)

(General)

(Declarations)

Dim exchangeobj As Object

Private Sub Form\_Load()  
Set exchangeobj = CreateObject("cobserv.EXCHANGE")  
End Sub

Private Sub ratesetbutton\_Click()  
exchangeobj.exrate100 = rate.Text \* 100  
End Sub

Private Sub exchangebutton\_Click()  
dollaroutput.Caption = exchangeobj.yentodoll(label1999.Caption)  
End Sub

Form1

レート (1ドルに対して) 

110

 円

1999 円= 18.17ドル

レート設定

変換

26.2.4 \*COMレイトバインドクラス

ここでは、\*COMレイトバインドクラスについて説明します。

26.2.4.1 \*COMレイトバインドクラスのファクトリメソッド

\*COMレイトバインドクラスのファクトリメソッドには、以下のメソッドがあります。

表26.2 \*COMレイトバインドクラスのファクトリメソッド

| メソッド名         | 引数                   | 意味／返却値                                                                                                             |
|---------------|----------------------|--------------------------------------------------------------------------------------------------------------------|
| CREATE-OBJECT | プログラムID (ProgID) (注) | プログラムIDで指定されたCOMサーバを起動し、起動したCOMサーバを制御する*COMクラスのオブジェクトを返却します。                                                       |
| GET-OBJECT    | ファイル名                | ファイル名に関連付けられたCOMサーバを起動し、起動したCOMサーバを制御する*COMクラスのオブジェクトを返却します。該当するCOMサーバがすでに起動されている場合、そのサーバを制御する*COMクラスのオブジェクトを返却します |

注: プログラムIDは、“アプリケーション.クラスタイプ”の形式の文字列です。プログラムIDの例を以下に示します。

- EXCEL.APPLICATION
- OASYS32.APPLICATION
- EXCEL.SHEET

- ・ CORBA.FACTORY

COMサーバのアプリケーション名およびクラスタイプ名については、各製品のマニュアルまたはヘルプを参照してください。

#### 26.2.4.2 \*COMレイトバインドクラスオブジェクトメソッド/プロパティ

\*COMレイトバインドクラスオブジェクトメソッドは、COMサーバに定義されているオブジェクトメソッドと同じものになります。また、COMサーバのプロパティは、以下の名前を持つメソッドとして、INVOKE文で呼び出します。

参照の場合

“GET-プロパティ名”

設定の場合

“SET-プロパティ名”

参照設定の場合

“SETREF-プロパティ名”



#### 参考

ほとんどの場合、プロパティは“設定”によって設定することができます。しかし、プロパティのデータ型がCOMサーバ(COMオブジェクト)の場合には、“参照設定”によってプロパティを設定しなければならないことがあります。“設定”“参照設定”のどちらで設定するかは、COMサーバのプロパティの属性に依存します。プロパティの属性が“propput”ならば“設定”、“propputref”ならば“参照設定”を行うようにします。プロパティの属性については、COMサーバの型情報やヘルプを参照してください。

COMサーバを\*COMレイトバインドクラスとして操作する場合の、メソッド/プロパティの関係を“表26.3 COM操作のメソッド/プロパティ”に示します。

表26.3 COM操作のメソッド/プロパティ

| COMサーバ | *COMクラス                                                                   |
|--------|---------------------------------------------------------------------------|
| メソッド名  | 同じメソッド名                                                                   |
| プロパティ名 | “GET-プロパティ名”メソッド(参照)<br>“SET-プロパティ名”メソッド(設定)<br>“SETREF-プロパティ名”メソッド(参照設定) |

各メソッドの引数の個数は、対応するCOMサーバのメソッドの引数の個数と同じになります。引数の型については、“26.5 COM連携で扱うデータ型の対応”で説明します。



#### 参考

プロパティの“設定”、“参照設定”または“参照”を“SET-プロパティ名”、“SETREF-プロパティ名”または“GET-プロパティ名”とすることで、COMサーバの本来のメソッド名と同じになる可能性があります。たとえば、COMサーバが“GET-M1”というメソッドと“M1”というプロパティを持っていた場合、\*COMクラスでは、GET-M1メソッドのメソッド名は“GET-M1”、M1プロパティの参照メソッド名も“GET-M1”になります。この場合、メソッドの先頭に“¥”を付加することで、メソッド呼出しを強制します。

```
INVOKE COMOBJ "¥GET-M1" ... GET-M1 メソッドを呼び出します。
INVOKE COMOBJ "¥¥GET-M1" ... ¥GET-M1メソッドを呼び出します。
INVOKE COMOBJ "GET-M1" ... M1 プロパティを参照します。
INVOKE COMOBJ "GET-¥M1" ... ¥M1プロパティを参照します。
```

#### 26.2.5 \*COMアーリバインドクラス

ここでは、\*COMアーリバインドクラスについて説明します。

### 26.2.5.1 \*COMアーライバインドクラスのファクトリメソッド

\*COMアーライバインドクラスのファクトリメソッドには、以下のメソッドがあります。

表26.4 \*COMアーライバインドクラスのファクトリメソッド

| メソッド名 | 引数 | 意味／返却値                                                      |
|-------|----|-------------------------------------------------------------|
| NEW   | なし | 指定されたクラス(型ライブラリから読み込み)のサーバを起動し、それを制御する*COMクラスのオブジェクトを返却します。 |

### 26.2.5.2 \*COMアーライバインドクラスのオブジェクトメソッド/プロパティ

\*COMアーライバインドクラスのオブジェクトメソッドは、COMサーバに定義されているオブジェクトメソッドと同じものになります。

COMサーバのプロパティをオブジェクトプロパティで呼び出す場合は、COMサーバのプロパティ名で呼び出します。INVOKE文でプロパティを呼び出す場合には、\*COMレイバインドクラスと同じ書き方で呼び出します。\*COMレイバインドクラスの書き方については、“26.2.4.2 \*COMレイバインドクラスのオブジェクトメソッド/プロパティ”を参照してください。



“26.2.4.2 \*COMレイバインドクラスのオブジェクトメソッド/プロパティ”で説明しているように、プロパティの設定の方法には、“設定”と“参照設定”がありますが、“参照設定”はINVOKE文によるSETREF-プロパティ名メソッドの呼出しだけで行うことができます。したがって、アーライバインドの場合、以下の記述ではプロパティの“設定”が行われます。

```
SET プロパティ名 OF COMオブジェクト TO 設定データ
```

### プロパティ指定子

他言語で作成されたCOMサーバには、COBOLの予約語と重なるプロパティ名を持つものがあります。COBOLの予約語と重なるプロパティ名は、そのままではプロパティ名としてプログラム中で使用できないため、リポジトリ段落のプロパティ指定子を用いて、プロパティ名を別の名前で扱うことができます。

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS WS AS “*COM:EXCEL:WORKSHEET”
 PROPERTY インデックス番号 AS “INDEX”.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WSOBJ OBJECT REFERENCE WS.
01 IDNO PIC S9(9) COMP-5.
PROCEDURE DIVISION.
 MOVE インデックス番号 OF WSOBJ TO IDNO
```

上記はWORKSHEETクラスの“INDEX”というプロパティに“インデックス番号”という名前を付けて、使用する例です。

プロパティ指定子には以下の使い方もあります。

- 長い名前のプロパティ名を、プロパティ指定子を用いて別の名前をつけることで記述性の悪さを解消する。

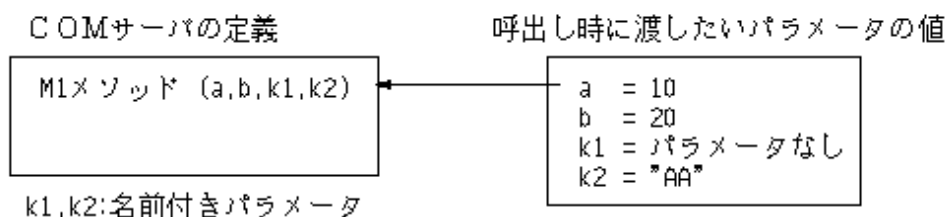
## 26.2.6 名前付きパラメタ

COMサーバにはパラメタを名前に対応付ける機能(名前付きパラメタ)を持つものもありますが、NetCOBOLで作成するCOMクライアントではパラメタを名前に対応付けることはできません。必ずサーバ側のパラメタの定義順に合わせてクライアント側のパラメタを定義し、必要に応じてパラメタの省略を使用してください。[参照]“26.2.7 パラメタの省略”

パラメタの定義順序については、対象となるCOMサーバのマニュアル、ヘルプまたは型ライブラリなどを参照してください。

## 26.2.7 パラメタの省略

パラメタを省略する場合、省略するパラメタにOMITTEDを指定します。



このような場合に、パラメータを省略する指定方法を以下に示します。

```
WORKING-STORAGE SECTION.
01 COMOBJ OBJECT REFERENCE COM.
01 VAL1 PIC S9(9) COMP-5 VALUE 10.
01 VAL2 PIC S9(9) COMP-5 VALUE 20.
01 VAL4 PIC X(10) VALUE "AA".
 :
PROCEDURE DIVISION.
 INVOKE COMOBJ "M1" USING VAL1 VAL2 OMITTED VAL4.
```

## 26.2.8 COM機能の例外処理

COMサーバはエラーが発生したとき、例外をCOMクライアントに通知します。NetCOBOLのCOMクライアント機能には、COMサーバから通知された例外を処理する機能があります。

ここでは、COM機能で発生するエラーと、エラー発生時の誤り処理手続きについて説明します。

### 26.2.8.1 例外の種類

COMサーバのメソッドを呼び出したときに発生するエラーは、以下の3つに分類できます。

- COBOLのシステムが検出するエラー
- COMのシステムが検出するエラー
- COMサーバが検出するエラー

| エラー分類              | 内容                                               | エラーの原因                                                                                                                                             |
|--------------------|--------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| COBOLのシステムが検出するエラー | COMサーバのメソッドを呼び出す前に、COBOLのシステムで検出するエラー            | <ul style="list-style-type: none"> <li>• CREATE-OBJECTメソッドなど、COMサーバのメソッドでないメソッド適合チェックでエラーが発生した</li> </ul>                                          |
| COMのシステムが検出するエラー   | COMサーバのメソッドを呼び出すときに、COMシステムやCOMサーバオブジェクトが検出するエラー | <ul style="list-style-type: none"> <li>• COMサーバが立ち上がっていない</li> <li>• COMサーバに指定したクラスやメソッドが存在しない</li> </ul>                                          |
| COMサーバが検出するエラー     | COMサーバのメソッドを実行しているときに、COMサーバが検出するCOMサーバ固有のエラー    | <ul style="list-style-type: none"> <li>• COMサーバのDBをアクセスしたときのパスワードが間違っている</li> <li>• COMサーバのメソッドが引数として0以上100以下の値を要求しているのに、引数として101を渡している</li> </ul> |

これらのエラーが発生したとき、COBOLのCOMクライアントアプリケーションは異常終了します。ただし、COMのシステムが検出するエラーと、COMサーバが検出するエラーについては、プログラム中に\*COM-EXCEPTIONクラスの例外処理を定義することによって、エラーを検出した場合の処理を定義することができます。

また、\*COM-EXCEPTIONクラスのメソッドを実行することにより、COMサーバで検出されたエラー情報を参照することができます。  
\*COM-EXCEPTIONクラスについては、“[26.6.2 \\*COM-EXCEPTIONクラス](#)”を参照してください。



## 26.2.8.2 例外発生時に通知される情報

ここでは、COM機能でエラーが発生したときに、COMのシステムまたはCOMサーバから通知されるエラー情報について説明します。

| 通知されるエラー情報の種類                            | 通知の有無        |               |
|------------------------------------------|--------------|---------------|
|                                          | COMのシステムから通知 | COMサーバから通知(注) |
| SCODE形式のエラーコード                           | ○            | ○             |
| COMサーバが定義した2バイトのエラーコード                   | —            | ○             |
| 発生したエラーに関する簡単な説明                         | —            | ○             |
| エラーを検出したCOMサーバの名前                        | —            | ○             |
| エラーに関する詳しい情報が入ったヘルプ<br>ファイル名とヘルプコンテキストID | —            | ○             |

注:COMサーバの仕様によっては通知されないエラー情報もあります。

SCODE形式のエラーコードとは、COMのシステムまたはCOMサーバが定義する4バイトのエラーコードです。コードの内容については、“メッセージ説明書”の“付録B SCODE形式のエラーコードの説明”を参照してください。

これらの情報は\*COM-EXCEPTIONクラスのメソッドを実行することによって参照できます。また、\*COM-EXCEPTIONクラスを利用した例外処理を定義しない場合は、プログラム終了時のエラーメッセージ中に表示されます。

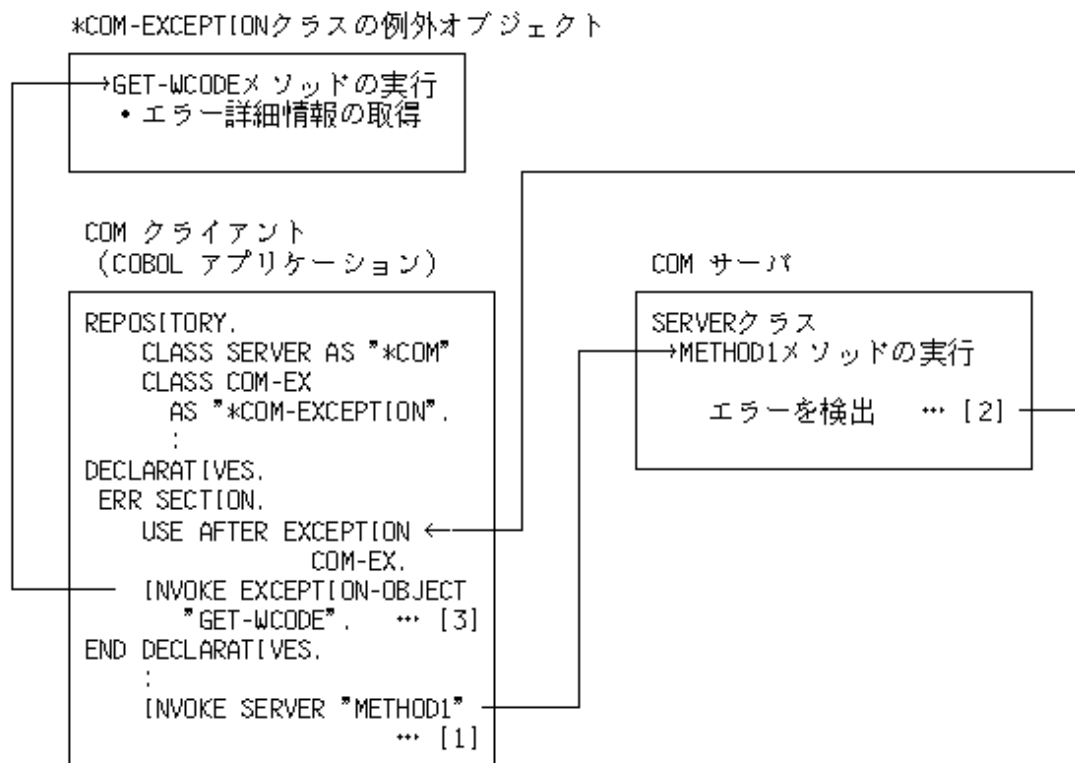
## 26.2.8.3 例外処理の流れ

COMクライアントには、\*COM-EXCEPTIONクラスのオブジェクトに対する例外処理手続きを記述します。

COMのシステムまたはCOMサーバでエラーを検出した場合、\*COM-EXCEPTIONクラスのオブジェクトを例外オブジェクトとする例外条件が発生し、COMクライアントに記述された例外処理が実行されます。

また、\*COM-EXCEPTIONクラスのメソッドを実行することで、COMのシステムやCOMサーバから通知されたエラーの詳細な情報を参照することもできます。

図26.8 例外処理の流れ



- [1] 特殊クラス\*COMを使用してCOMサーバのメソッドを呼び出します。
- [2] COMサーバのメソッド実行中にエラーが発生した場合、\*COM-EXCEPTIONクラスのオブジェクトが作成されます。
- [3] 例外処理手続きとして、例外オブジェクトのGET-WCODEメソッドを実行します。

例外オブジェクトによる例外条件の詳細については、“[18.1.2 例外オブジェクト](#)”を参照してください。

## 26.2.8.4 \*COM-EXCEPTIONクラスの使用例

### アプリケーションの概略

- ー レイトバインドでCOMサーバと連携します。
- ー COMサーバのプログラムIDを”COMSERVER.COMCLASS”とします。
- ー COMサーバのメソッドを実行した場合のエラー処理として、\*COM-EXCEPTIONクラスを使用します。
- ー COMサーバには、以下のメソッドがあるものとします。

#### PASSWORD\_CHECKメソッド

- 第一引数で指定された数値が、あらかじめ決められたCOMサーバのパスワードと一致するかをチェックします。
- パスワードの値が誤っている場合には、COMクライアントにエラーコード「0x0001」を通知してメソッドを終了します。

### \*COM-EXCEPTIONクラスを使用した例

```

IDENTIFICATION DIVISION.
PROGRAM-ID. PROGRAM-A.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.

```

```

CLASS COM AS "*COM"
CLASS COM-EXCEPTION AS "*COM-EXCEPTION".
:
DATA DIVISION.
WORKING-STORAGE SECTION.
01 COM-SV-OBJ OBJECT REFERENCE COM.
01 COM-ERROR-INFORMATION.
 02 COM-ER-TYPE PIC X(01).
 02 COM-WCODE PIC X(02).
01 COM-SERVER PIC X(20) VALUE "COMSERVER.COMCLASS".
01 METHOD-NAME PIC X(20) VALUE SPACE.
01 SERVER-PASSWORD PIC X(04).
:
PROCEDURE DIVISION.
DECLARATIVES.
ERR SECTION.
 USE AFTER EXCEPTION COM-EXCEPTION.
 INVOKE EXCEPTION-OBJECT "GET-ERROR-TYPE"
 RETURNING COM-ER-TYPE. ... [1]
 IF COM-ER-TYPE = "1" THEN
 DISPLAY "ERROR OCCURED IN " METHOD-NAME "METHOD" ... [2]
 SET COM-SV-OBJ TO NULL
 EXIT PROGRAM
 ELSE
 INVOKE EXCEPTION-OBJECT "GET-WCODE" ... [3]
 RETURNING COM-WCODE
 IF COM-WCODE NOT = X"0001" THEN ... [4]
 DISPLAY "ERROR OCCURED IN " METHOD-NAME "METHOD"
 SET COM-SV-OBJ TO NULL
 EXIT PROGRAM
 END-IF
 END-IF.
END DECLARATIVES.
:
MOVE SPACE TO COM-ER-TYPE.
MOVE "CREATE-OBJECT" TO METHOD-NAME.
INVOKE COM "CREATE-OBJECT" USING COM-SERVER ... [5]
 RETURNING COM-SV-OBJ.
:
PERFORM WITH TEST AFTER UNTIL COM-ER-TYPE = SPACE ... [6]
 DISPLAY "PLEASE INPUT SERVER-PASSWORD"
 ACCEPT SERVER-PASSWORD
 MOVE "PASSWORD_CHECK" TO METHOD-NAME
 MOVE SPACE TO COM-ER-TYPE
 MOVE X"0000" TO COM-WCODE
 INVOKE COM-SV-OBJ METHOD-NAME USING SERVER-PASSWORD
 IF COM-WCODE = X"0001" THEN ... [7]
 DISPLAY "PASSWORD ERROR! "
 END-IF
END-PERFORM.
SET COM-SV-OBJ TO NULL.
:
END PROGRAM PROGRAM-A.

```

[1] 例外発生時の処理を定義します。まず、例外オブジェクトのGET-ERROR-TYPEメソッドを実行し、発生したエラーの種別を判定します。

[2] エラー種別が「1(COMシステムがエラーを検出)」の場合は、COMサーバを解放して処理を終了します。

[3] エラー種別が「0(COMサーバがエラーを検出)」の場合は、GET-WCODEメソッドを呼び出して、COMサーバが設定したエラーコードを取得します。

[4] エラーコードが「0x0001(パスワード入力エラー)」以外の場合は、COMサーバを解放して処理を終了します。エラーコードが「0x0001(パスワード入力エラー)」の場合は、そのまま処理を継続します。

[5] CREATE-OBJECTメソッドを呼び出して、COMサーバオブジェクトを起動します。このメソッド呼び出し時にエラーが発生した場合は、エラー種別が必ず「1」になるので、[2]のエラー処理が実行されます。

[6] パスワードを入力し、その値を引数にしてCOMサーバオブジェクトのPASSWORD\_CHECKメソッドを呼び出します。このメソッド呼び出し時にCOMのシステムがエラーを検出した場合は[2]のエラー処理が、COMのサーバがエラーを検出した場合は[3]～[4]のエラー処理が実行されます。

[7] PASSWORD\_CHECKメソッドでパスワード入力エラーが発生した場合、正しいパスワードが入力されるまで[6] の処理を繰り返します。

## 26.2.9 オブジェクト指定子

COM機能においても、一般のクラスの場合と同様に、オブジェクト指定子が使用できます。

COMオブジェクト AS アーライバインド特殊クラス名

EXCELのような複合COMコンポーネントと連携するCOMクライアントプログラムでは、COMサーバとの間で、様々な種類のCOMオブジェクトを受け渡ししながら処理を進めていくことになります。

アーライバインドによりこの連携を実現する場合、COMサーバのプロパティまたはメソッドのパラメタ型がIDispatch\*型であれば、クライアントは、そのパラメタにより渡されるCOMオブジェクトを\*COMレイトバインド型でしか受け取れません。

開発者が全ての連携をアーライバインドにより実現することを考えた場合、この\*COMレイトバインド型で受け取ったCOMオブジェクトを使用したCOM連携も、アーライバインドによって実現したいと考える筈です。これを実現するためには、レイトバインド型に保持するCOMオブジェクトを、アーライバインド型に入れ替えなければなりません。

ところがこの組み合わせは、オブジェクトの「代入時の適合」の規則として許されていません。そのまま指定したのでは、翻訳時にエラーになってしまいます。

このような場合に、オブジェクト指定子を使用します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROGRAM-A.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS COM AS "*COM"
 CLASS EXCEL AS "*COM:EXCEL:APPLICATION"
 CLASS WORKBOOKS AS "*COM:EXCEL:WORKBOOKS"
 CLASS WORKBOOK AS "*COM:EXCEL:WORKBOOK"
 CLASS WORKSHEET AS "*COM:EXCEL:WORKSHEET"
 CLASS RANGE-C AS "*COM:EXCEL:RANGE".
DATA DIVISION.
WORKING-STORAGE SECTION.
01 汎用オブジェクト域 OBJECT REFERENCE COM.
01 ブック OBJECT REFERENCE WORKBOOK.
01 シート OBJECT REFERENCE WORKSHEET.
01 範囲 OBJECT REFERENCE RANGE-C.
01 セル OBJECT REFERENCE RANGE-C.
01 選択シート名 PIC X(20).
01 セル行位置 PIC S9(9) COMP-5.
01 セルカラム位置 PIC S9(9) COMP-5.
PROCEDURE DIVISION.
 :
* SHEETコレクションから、"SHEET1"のオブジェクトを選択します。
 MOVE "SHEET1" TO 選択シート名.
 INVOKE WORKSHEETS OF ブック "GET-ITEM"
 USING 選択シート名
 RETURNING 汎用オブジェクト域.
 ... [1]

* "*COM"型オブジェクト参照データに獲得した"SHEET1"のオブジェクトを、
* オブジェクト指定子を使用して、"*COM:EXCEL:SHEET"型オブジェクト参照
* データに設定します。
 SET シート TO 汎用オブジェクト域 AS WORKSHEET.
 ... [2]
```

```
* RANGEオブジェクトを取得した後、特定の位置のセルのオブジェクトを獲得します。
 INVOKE シート "GET-CELLS" RETURNING 範囲. ... [3]
 INVOKE 範囲 "GET-ITEM"
 USING セル行位置 セルカラム位置
 RETURNING セル.
 :
```

[1] 「WORKSHEETS OF ブック」の結果であるSHEETSオブジェクトにより、ITEMプロパティを呼び出し、WORKSHEETオブジェクトを獲得します。ITEMプロパティの返却値の型はIDispatch\*型なので、RETURNING指定には\*COMレイトバインド型のオブジェクト参照データを指定します。

[2] \*COMレイトバインド型に獲得したWORKSHEETオブジェクトを、オブジェクト指定子を使用して、WORKSHEETクラス指定のアーリーバインド型オブジェクト参照データに設定します。

[3] アーリーバインド型に入れ直したことにより、以降の連携もアーリーバインドで実現できます。

COM連携におけるオブジェクト指定子も、その使い方は、一般のクラスの場合と変わりありません。例えば上記例の[2]と[3]は、まとめて次のように書くことも可能です。

```
INVOKE 汎用オブジェクト域 AS WORKSHEET "GET-CELLS" RETURNING 範囲.
```

ただしオブジェクト指定子は、実行時にそれぞれに対して必ずチェック処理が走ります。従って、上記のような書き方を多用しなければならないのであれば、最初にアーリーバインド型オブジェクト参照データに入れ替えてしまい、以降のオブジェクト指定子の数を少なくした方が、実行性能的には有利です。

使い方については一般のクラスの場合と変わりませんが、チェックの規則は異なります。

COM連携におけるオブジェクト指定子の実行時チェックの規則は、次のようになります。

- ASに指定されたクラスが、dispinterface名指定のアーリーバインドクラスの場合、対象となるオブジェクトがそのdispinterfaceを実装している、またはNULLであれば適合。
- ASに指定されたクラスが、coclass名指定のアーリーバインドクラスの場合、対象となるオブジェクトが、そのcoclassのオブジェクトである、またはNULLであれば適合。

## 26.3 COMサーバ機能

ここでは、COBOLによるCOMサーバの作成方法や既存プログラムのCOMサーバ化などについて説明します。

### 26.3.1 COMサーバとは

COMサーバは、コンポーネントあるいはCOMコンポーネントとも呼ばれます。COMサーバは、操作方法をインタフェースという形で公開することで、他のプログラムから操作できます。インタフェースは言語無依存の形式で、どの言語で開発されたかを意識する必要がありません。このため、COBOLで作成したCOMサーバは、COBOLはもちろんVisual Basic、Visual Basic Scripting Edition、Visual C++などCOMクライアント機能を持つプログラムから利用できます。

### 26.3.2 COBOLで作成するCOMサーバの特徴

NetCOBOLで作成するCOMサーバには以下のような特徴があります。

- NetCOBOLのクラス定義をそのままCOMサーバにすることができます。言い換えれば、クラスとして定義されたものだけを、COMサーバにすることができます。
- NetCOBOLのCOMサーバは、DLLとして作成します。COMサーバは一般にはEXEでも作成可能ですが、NetCOBOLではEXEのCOMサーバは作成できません。

このほかの特徴を“表26.5 COBOLで作成したCOMサーバの特徴”にまとめます。

表26.5 COBOLで作成したCOMサーバの特徴

| 項目        | 特徴                               |
|-----------|----------------------------------|
| 公開インタフェース | IUnknown、IClassFactory、IDispatch |

| 項目         | 特徴        |
|------------|-----------|
| クラス種別      | coclassだけ |
| 実装型        | DLL       |
| 定義クラス数     | 複数        |
| 型ライブラリ/型情報 | あり        |
| DCOM       | 動作可(*1)   |
| MTS        | 動作可(*2)   |

\*1:リモートマシンにCOBOLランタイムシステムが必要です。

\*2:リモートで動作させる場合には、リモートマシンにCOBOLランタイムシステムが必要です。

インタフェース、クラス種別および型ライブラリなどについては、“[26.9 COM関連の用語の説明](#)”を参照してください。

### 26.3.3 COMサーバの作成

COMサーバの作成は、COBOLプロジェクトマネージャのCOMサーバ開発機能で行います。プロジェクトマネージャの操作方法については、“[26.8.2 COMサーバ作成機能の使い方](#)”を参照してください。

COMサーバは、クライアントへの例外通知処理を除けばCOMを意識したソース記述をまったく必要としません。COBOLのクラス定義で記述したメソッドおよびプロパティがそのままCOMサーバのメソッドとプロパティとなります。したがってオブジェクト指向プログラミングで動作実績のあるクラス定義であれば、そのままCOMサーバとして構築し、登録を行えばCOMサーバとして利用できます。



#### 注意

- ・ファクトリ定義のメソッドおよびプロパティは、COMサーバとして公開できません。
- ・COMサーバ名、クラス名、メソッド名、プロパティ名およびメソッド定義の連絡節のデータ名にハイフン(-)を使用しないでください。
- ・COMサーバ名、クラス名およびメソッド名に空白を使用しないでください。
- ・集団項目など、いくつかのデータ型はCOMサーバのメソッドのパラメタとして使用できません。使用できるデータ型については、“[26.5 COM連携で扱うデータ型の対応](#)”を参照してください。

#### COMサーバの作成手順

ここでは、“[クラス定義](#)”のEXCHANGEクラスをCOMサーバ化する方法について説明します。

1. プロジェクトマネージャを起動して、プロジェクトを作成し、ビルドが行える状態にします。ここでは、プロジェクト名をCOBSERV、DLL名をCOBSERV.DLLとして、DLLを構成するCOBOLプログラム(EXCHANGE.COB)を登録します。登録するCOBOLプログラムには、最低1つのクラス定義が必要です。登録の際には、ターゲットリポジトリの設定も必ず行ってください。

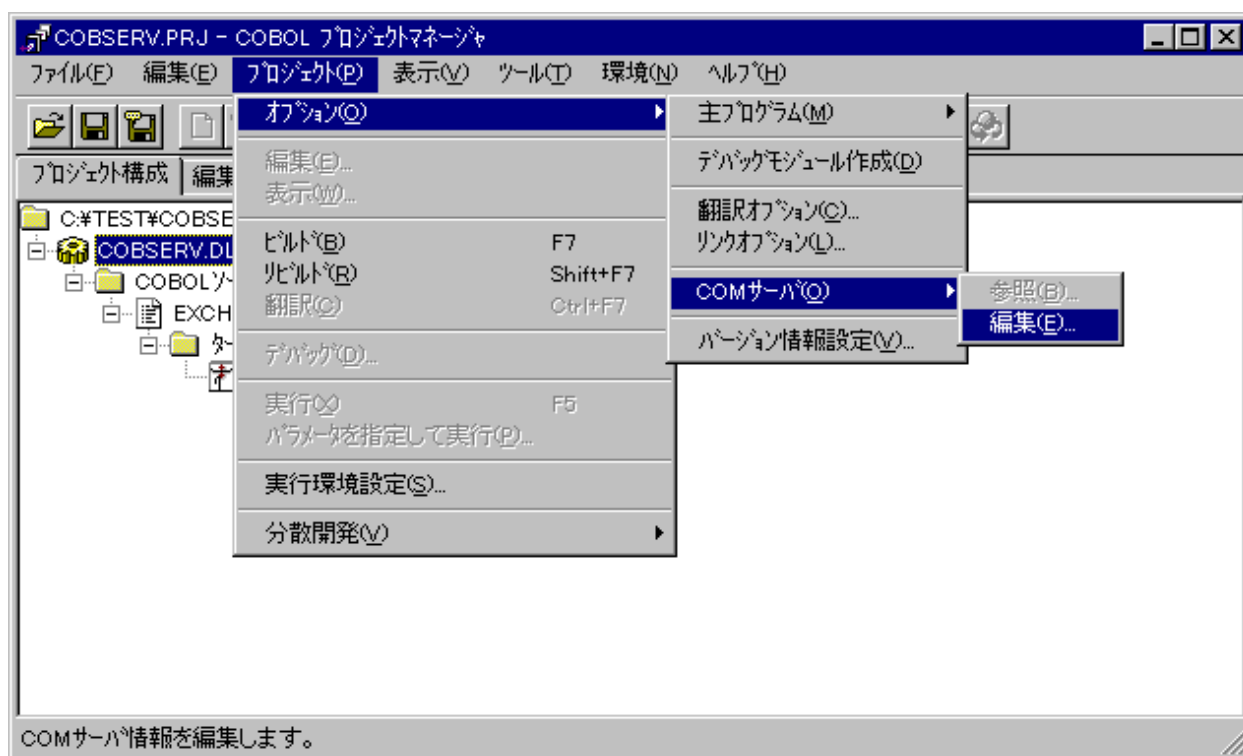
また、翻訳オプションとしてTHREAD(MULTI)を指定します。



#### 注意

COMサーバは、プロセスモードでもスレッドモードでも動作しますが、一般的にサーバタイプのアプリケーションには、マルチスレッドをおすすめします。スレッドモードは翻訳オプションTHREADで指定できます。スレッドモードについては、“[第24章 マルチスレッド](#)”を参照してください。

- プロジェクトマネージャに表示されたCOBSERV.DLLを選択し、[プロジェクト]－[オプション]メニューから“COMサーバ”の“編集”を選択します。



- サーバクラスの選択ページに表示されたCOBOLのクラスの一覧から、EXCHANGEクラスを選択し、サーバクラスに追加します。



サーバ情報、サーバクラス情報およびクラスIDについては、“[26.8.2 COMサーバ作成機能の使い方](#)”で説明します。

- ビルドを行い、COBSERV.DLLを作成します。

以上で、COMサーバの作成は完了です。

ここで作成されたCOMサーバには、EXCHANGEクラスの中に定義されたEXRATE100プロパティとYENTODOLLメソッドの情報が含まれます。このほかに、EXCHANGEクラスが継承しているFJBASEクラスの中に定義されたINITメソッドおよび\_FINALIZEメソッドの情報も含まれますが、この2つのメソッドはCOMサーバでは意味を持ちません。

## 26.3.4 COMサーバの登録と削除

作成したCOMサーバを利用するためには、システムのレジストリにCOMサーバを登録する必要があります。登録には、REGSVR32.EXE(注)を使用します。次のコマンドを入力することで、引数に指定したDLL(COMサーバ)がシステムのレジストリに登録されます。

注:REGSVR32.EXEは、Windowsがインストールされているフォルダの下のSYSTEMフォルダまたはSYSTEM32フォルダの下に格納されています。

### COMサーバの登録

COMサーバをレジストリに登録する場合、以下のコマンドを入力します。

```
REGSVR32. EXE [パス名¥]DLL名. DLL
```



例

COBSERV.DLLを登録する。

```
C:¥SAMPLE> REGSVR32. EXE COBSERV. DLL
```

### COMサーバの削除

登録したCOMサーバをレジストリから削除する場合、/Uスイッチを指定します。

```
REGSVR32. EXE /U [パス名¥]DLL名. DLL
```



例

COBSERV.DLLを削除する。

```
C:¥SAMPLE> REGSVR32. EXE /U COBSERV. DLL
```



注意

Windows Vista、Windows 7、またはWindows 8でREGSVR32.EXEを使用する場合、管理者として起動したコマンドプロンプトで使用してください。

## 26.3.5 COMサーバの利用

NetCOBOLで作成したCOMサーバは、COMサーバ作成機能の命名規則によって、“COBSERV.EXCHANGE”というProgIDが割り当てられます。このProgIDを使うことで、COMサーバはどんなCOMクライアントからも利用できます。

## 26.3.6 例外の通知

COMサーバでエラーが発生した場合、例外をCOMクライアントに通知します。ここでは、COMサーバで検出される例外の種類と、COMクライアントへの例外の通知方法について説明します。

### 例外の種類

COMサーバがCOMクライアントに通知する例外には、以下の3種類があります。



- Windowsシステムが検出する例外
- COBOLランタイムが検出する例外
  - a. ユーザが定義したプログラムに制御を渡す前に検出する例外
  - b. 利用者プログラム実行中にエラーが起こったときに発生する例外
- COMサーバ内で利用者プログラムが発生させる例外

Windowsシステムが検出する例外には、たとえば指定されたCOMサーバが存在しないなど、COMサーバに制御が渡る前に検出される例外があります。

COBOLランタイムシステムが検出する例外のa.は、クライアントから渡されたメソッドの引数の個数や型が違う場合など、利用者プログラムに制御を渡すための処理ができないときに発生します。COBOLランタイムシステムが検出する例外は、COMサーバ実行時に自動的にチェックされます。

COMサーバで利用者プログラムが例外が発生させる場合、RAISING指定のEXIT文を使用します。このとき、\*COM-EXCEPTIONクラスのオブジェクトを例外オブジェクトとして定義しておく必要があります。



参照

“18.1.4 RAISING指定のEXIT文の動作”

## 例外の通知方法

ここでは、“プログラム定義”および“クラス定義”を例にして説明します。

クラス定義をCOMサーバ、プログラム定義をCOMクライアントとしてアプリケーションを作成し、例外処理を追加したプログラムの例を以下に示します。

### COMサーバの例外処理

```

IDENTIFICATION DIVISION.
CLASS-ID. EXCHANGE INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS FJBASE
 CLASS COMEXCP AS “*COM-EXCEPTION”. ...[1]
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 EXRATE100 PIC S9(4) COMP-5 PROPERTY.
PROCEDURE DIVISION.
METHOD-ID. YENTODOLL.
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 EXCPOBJ OBJECT REFERENCE COMEXCP. ...[2]
 01 WCODE PIC S9(4) COMP-5 VALUE 1.
 01 ERR-MSG PIC N(100)
 VALUE NC“交換レートは正の数値でなければなりません。”.
LINKAGE SECTION.
 01 YEN PIC S9(4) COMP-5.
 01 DOLLARS PIC S9(4) V9(2).
PROCEDURE DIVISION USING YEN
 RETURNING DOLLARS
 RAISING COMEXCP. ...[3]
* レートが0以下ならば例外を返す。
 IF EXRATE100 <= 0 THEN -----
 INVOKE COMEXCP “NEW” RETURNING EXCPOBJ ↑
 INVOKE EXCPOBJ “SET-WCODE” USING WCODE [4]
 INVOKE EXCPOBJ “SET-DESCRIPTION” USING ERR-MSG
 EXIT METHOD RAISING EXCPOBJ ↓
 END-IF. -----

```

```
* レートを100で割り円をドルに換算する
 COMPUTE DOLLARS = YEN / (EXRATE100 / 100).
END METHOD YENTODOLL.
END OBJECT.
END CLASS EXCHANGE.
```

- [1] \*COM-EXCEPTIONクラスを使用することを宣言します。
- [2] \*COM-EXCEPTIONクラスのオブジェクトを定義します。
- [3] \*COM-EXCEPTIONクラスの例外を返却することを指定します。
- [4] 交換レートが0以下ならば例外を発生させる処理を行います。

#### COMクライアントの例外処理

```
@OPTIONS MAIN(WINMAIN)
IDENTIFICATION DIVISION.
PROGRAM-ID. RATECHG.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS EXCHANGE AS "*COM"
 CLASS COMEXCP AS "*COM-EXCEPTION". ...[1]
DATA DIVISION.
WORKING-STORAGE SECTION.
01 RATE PIC 999V99.
01 W-RATE PIC S9(5) COMP-5.
01 EXCHGOBJ OBJECT REFERENCE EXCHANGE.
01 YEN PIC S9(4) COMP-5.
01 DOLLARS PIC S9(4)V9(2).
01 ED-DOLLARS PIC ZZZ.99.
01 PRG-ID PIC X(20) VALUE "COBSERV. EXCHANGE".
01 COM-ERROR-INFORMATION.
 03 COM-ER-TYPE PIC X(01).
 03 COM-WCODE PIC X(02).
 03 COM-ERRMSG PIC N(100).
PROCEDURE DIVISION.
DECLARATIVES.
ERR SECTION.
 USE AFTER EXCEPTION COMEXCP.
 INVOKE EXCEPTION-OBJECT "GET-ERROR-TYPE"
 RETURNING COM-ER-TYPE.
 IF COM-ER-TYPE = "1" THEN
 DISPLAY "エラーが起きました。"
 ELSE
 INVOKE EXCEPTION-OBJECT "GET-WCODE"
 RETURNING COM-WCODE [2]
 IF COM-WCODE = X"0001" THEN
 INVOKE EXCEPTION-OBJECT "GET-DESCRIPTION"
 RETURNING COM-ERRMSG
 DISPLAY COM-ERRMSG
 ELSE
 DISPLAY NC"エラーが起きました"
 END-IF.
 END-IF.
 SET EXCHGOBJ TO NULL.
 EXIT PROGRAM.
END DECLARATIVES.
 INVOKE EXCHANGE "CREATE-OBJECT" USING PRG-ID
 RETURNING EXCHGOBJ.
 DISPLAY NC"1ドルのレートを入力してください(999.99)"
 ACCEPT RATE.
* レートを小数部なしの5桁に変換する
 COMPUTE W-RATE = RATE * 100.
```

```

* 整数5桁に直されたレートをプロパティに設定する
 INVOKE EXCHANGE "SET-EXRATE100" USING W-RATE.
 DISPLAY "1999円をドルに変換する".
* 変換メソッドを呼ぶ
 MOVE 1999 TO YEN.
 INVOKE EXCHGOBJ "YENTODOLL" USING YEN
 RETURNING DOLLARS.
* 編集して表示する
 MOVE DOLLARS TO ED-DOLLARS.
 DISPLAY ED-DOLLARS.
END PROGRAM RATECHG.

```

[1] \*COM-EXCEPTIONクラスを使用することを宣言します。

[2] 例外処理を定義します。



## 参考

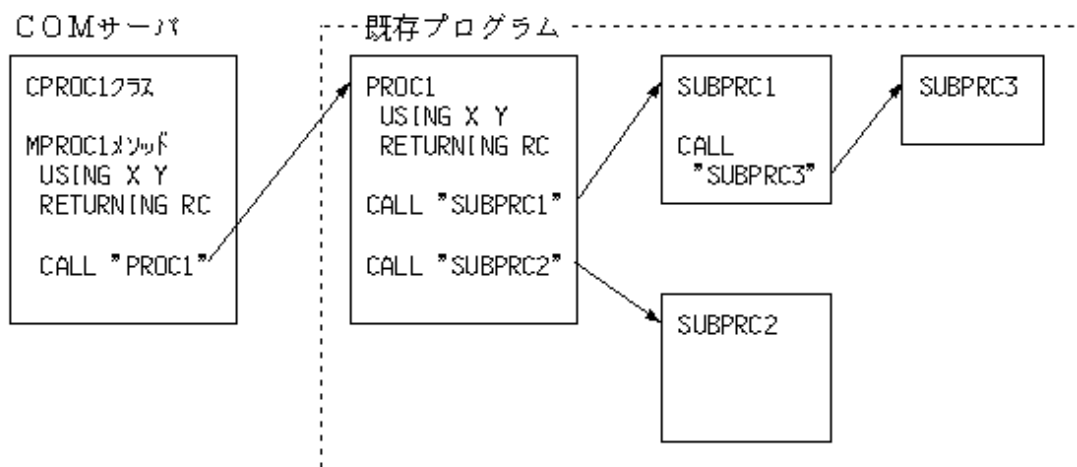
COMサーバプログラムでCOBOLのエラーを検出した場合、COBOLランタイムシステムは、WCODEが'1001'の例外を返却します。詳細については、“[26.6.2 \\*COM-EXCEPTIONクラス](#)”を参照してください。

## 26.3.7 既存プログラムのCOMサーバ化

既存の手続き型のプログラムをCOMサーバ化する方法について説明します。

この例では、PROC1は他のプログラムから呼び出されるいくつかのサブプログラムを持っています。このような既存プログラムをCOMサーバ化する場合、プログラムを呼び出すクラスを定義し、そのクラスをCOMサーバ化します。

図26.9 手続き型プログラムをCOMサーバ化する例



手続き型のプログラムを呼び出すメソッドの引数は、呼び出すプログラムの引数とデータ型および個数が同じになるように定義します。

以下にCOMサーバ化するクラス定義のプログラム例を示します。

手続き型プログラムを呼び出すクラス定義の例

```

IDENTIFICATION DIVISION.
CLASS-ID. CPROC1 INHERITS FJBASE. ...[1]
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS FJBASE.
OBJECT.

```

```

PROCEDURE DIVISION.
METHOD-ID. MPROC1. ...[2]
DATA DIVISION.
LINKAGE SECTION.
01 X ...
01 Y ...
01 RC ...
PROCEDURE DIVISION USING X Y RETURNING RC. ...[3]
* POC1を呼び出す
 CALL "PROC1" USING X Y RETURNING RC. ...[4]
END METHOD MPROC1.
END OBJECT.
END CLASS CPROC1.

```

- [1] PROC1を呼び出すクラスとして、CPROC1を定義します。
- [2] PROC1を呼び出すメソッドとして、MPROC1を定義します。
- [3] PROC1の呼び出し引数と同じ引数でメソッドを定義します。
- [4] PROC1を呼び出します。

この例のように既存の手続き型のプログラムでも、簡単なクラス定義でラッピングすればCOMサーバとすることができます。このサーバを呼び出すクライアントも、CALL文をCOMのオブジェクトを作成するINVOKE文とCOMオブジェクトを呼び出すINVOKE文に変更するだけです。



## 注意

CPROC1はクラスとして定義されているため、1つのプロセス内に同時に複数のオブジェクトインスタンスを起動できます。しかし、一般的には、PROC1やそれが呼び出しているサブプログラムは、同時に複数のクライアントから使用することはできません。つまり、ここで説明したCOMサーバ化の方法は、1クライアントに限る場合にだけ動作が保証されます。複数のクライアントから同時に使用できるようにするためには、PROC1やそれが呼び出すサブプログラム自身をクラス定義し直す必要があります。

## 26.3.8 注意事項

ここでは、NetCOBOLで作成したCOMサーバを動作させる場合の注意事項について説明します。

- COMサーバはDLLで作成されるため、クライアントプログラムのプロセスにロードされて実行します。このため、実行環境の初期化に関して、通常のCOBOLのDLLと同じ注意が必要になります。[参照]“[5.3.2 実行環境情報の設定方法](#)”
- COMクライアントがCOBOLで作成されている場合、COMサーバはクライアントと同じ文字コード(シフトJIS/Unicode)で動作するように作成しなければなりません。
- COMクライアントがCOBOLで作成されている場合、クライアントプログラムがシングルスレッドモードで動作するときは、COMサーバもシングルスレッドで動作します。
- COMクライアントがCOBOLで作成されている場合、COMクライアントが必要とする実行用初期化ファイルの情報とCOMサーバが必要とする実行用初期化ファイルの情報を一つのファイルにマージしておく必要があります。
- COMクライアントがCOBOLで作成されていない場合、COBOLの実行環境は、COBOLで作成したCOMサーバが最初にロードされた時に初期化されます。同じCOMクライアントから利用するCOMサーバのDLLは、すべて同じフォルダに置き、同じ場所に実行用初期化ファイルを置くようにします。このときの初期化については、“[5.3.2 実行環境情報の設定方法](#)”の“DLLの格納場所にあるCOBOL85.CBRの使用について”を参考にしてください。
- 別々のDLLに定義されている複数のCOMサーバを利用する場合、それらは同じプロセスにロードされるため、同じ文字コードで作成されていなければなりません。

## 26.4 DCOMとMTS

COMサーバは、ローカルマシン上だけでなく、ネットワークで接続された別のマシン上でも動作させることができます。この機能をDCOM(Distributed COM)と呼びます。COMサーバをリモートサーバマシン上で動作させる場合、COMサーバの作成時に特別なことをする必要はありません。ローカルで動作しているCOMサーバをそのままネットワーク環境にインストールできます。

DCOMは、単独でも動作できますが、ここではMTS(Microsoft Transaction Server)環境を利用した方法について説明します。DCOM単独の動作では、クライアントから起動要求があるたびにサーバ上にプロセスが開設されるため、性能上問題がありCOBOLでは推奨しません。

以下の順に説明します。

- ・ MTSの概要
- ・ MTS環境への登録方法
- ・ クライアントマシンへのインストール
- ・ MTSのトランザクション管理
- ・ MTSのトランザクション処理



MTSの詳細は、“Option Packのオンラインマニュアル”または“Microsoft Transaction Serverのヘルプ”を参照してください。

### 26.4.1 MTSの概要

MTSは、大きく2つに分類できます。

- ・ オブジェクトの管理
- ・ トランザクション制御

MTSに登録されるCOMサーバは、パッケージと呼ばれる単位で登録されます。

DLLで作成されたCOMサーバは、通常クライアントプログラムが動作しているプロセスにロードされますが、MTSに登録されたCOMサーバは、MTSの開設したプロセスにロードされます。このプロセスは、1つのパッケージに対して1つだけ開設され、そのパッケージに登録されたDLLはすべてそのプロセス内で動作します。一度パッケージのプロセスにロードされたDLLは、そのDLL内に定義されたCOMサーバが破棄された後もプロセスに留まったままになるため、起動が高速になります。

図26.10 パッケージとCOMサーバ(DLL)の関係

| MTS       |           |           |       |
|-----------|-----------|-----------|-------|
| パッケージA    | パッケージB    | パッケージC    |       |
| COMA1.DLL | COMB1.DLL | COMC1.DLL | ..... |
| COMA2.DLL |           | COMC2.DLL |       |
| ...       |           | ...       |       |

### 26.4.2 MTS環境への登録方法

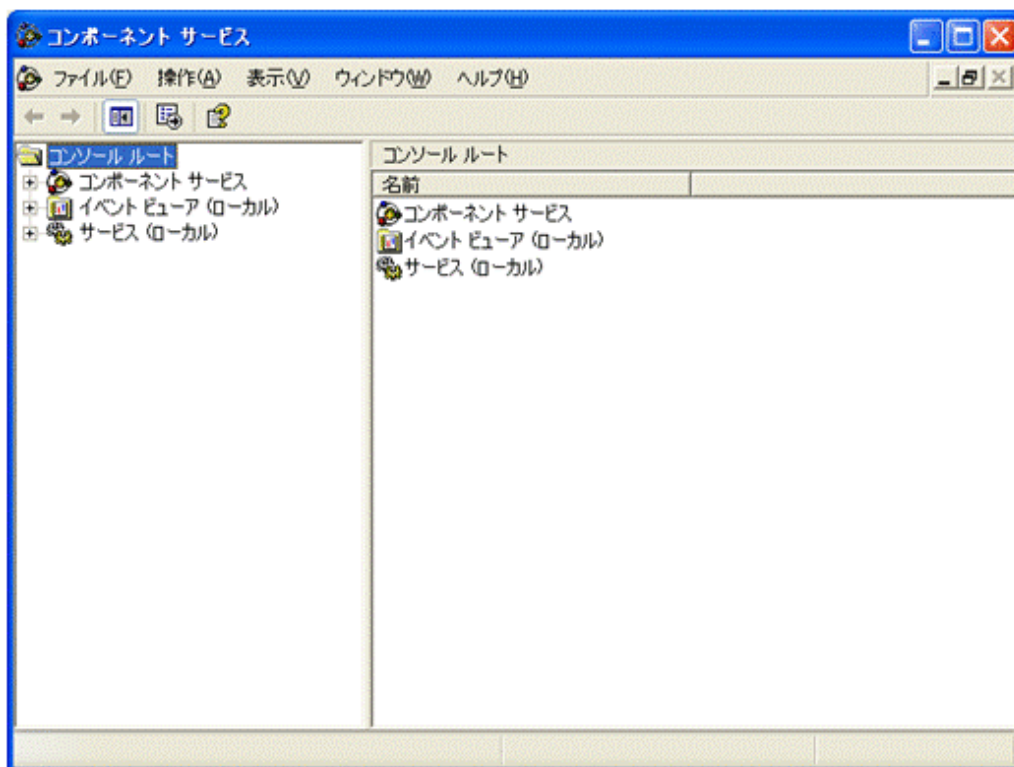
COMサーバをMTSに登録する方法について説明します。ここでは、Windows XPをもとに説明します。

その他のWindowsも同じ手順で行います。

1. コンポーネントサービスの起動

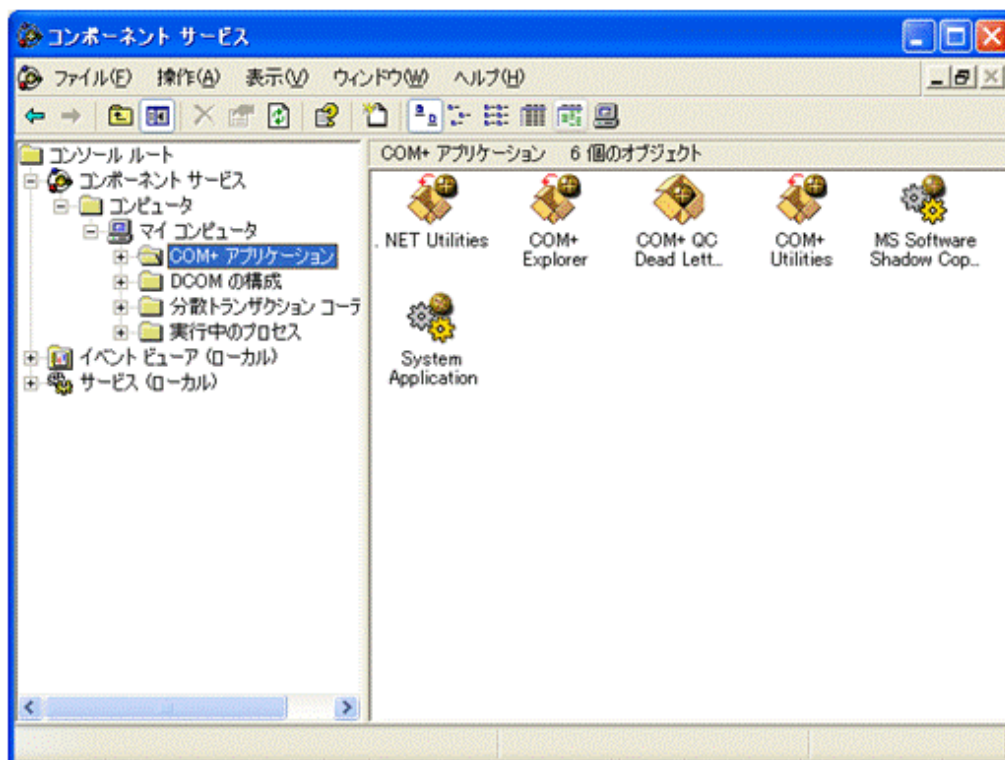
COMサーバをMTSに登録するには、コンポーネントサービスを起動します。

コンポーネントサービス画面は、[スタート]メニューから“設定”を選択し、“コントロールパネル”をクリックして、[管理ツール]アイコンをダブルクリックすると表示されます。コンポーネントサービスの初期画面を以下に示します。



## 2. 空アプリケーションの追加

コンポーネントサービス画面のコンソールルートから、“コンポーネントサービス”、“コンピュータ”、“マイコンピュータ”、“COM+アプリケーション”の順で開きます。“COM+アプリケーション”を選択して、[操作]メニューの“新規作成”の“アプリケーション”を選択します。



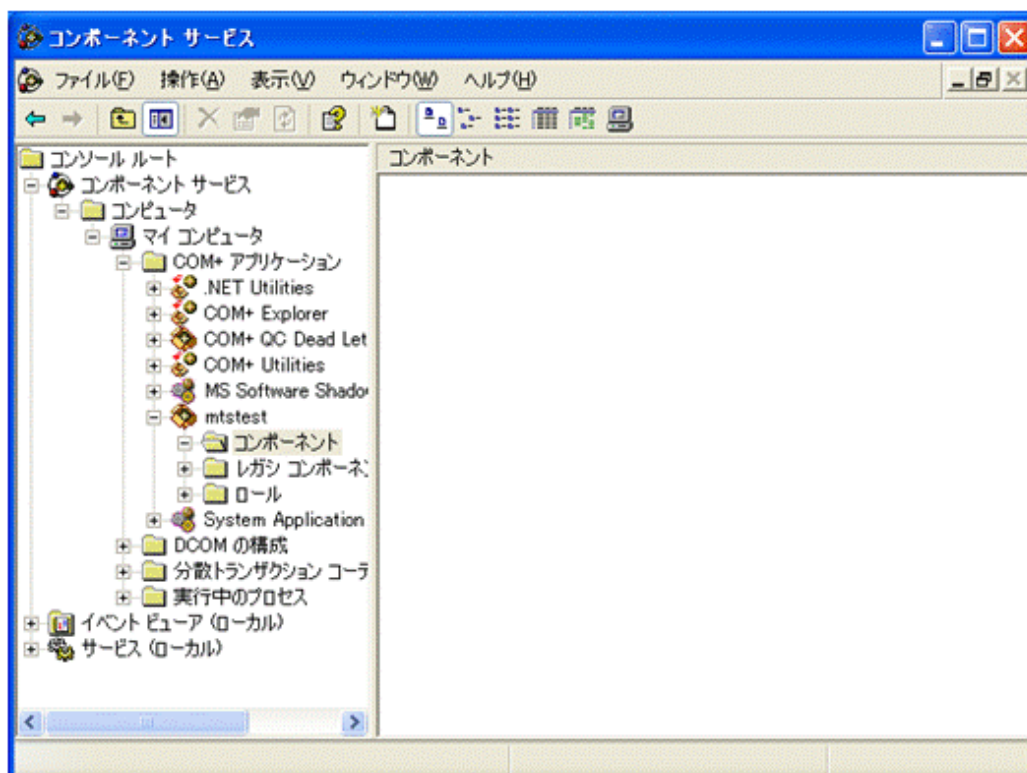


COM+ アプリケーション インストールウィザードが表示されたら、“次へ(N)”をクリックします。“新しいアプリケーションのインストールまたは作成”画面が表示されたら、“空のアプリケーションを作成する”をクリックします。新しいアプリケーション名の入力を求められたら、適当な名前を入力します。この例では“mtstest”とします。

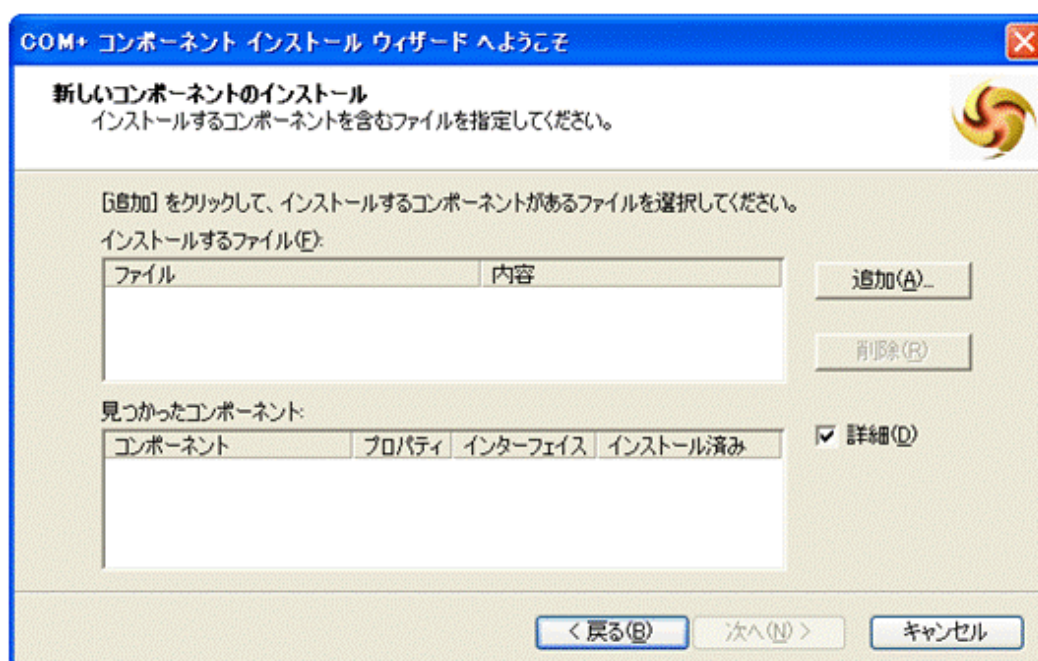
次に、アプリケーションIDの設定を行います。ここで入力するIDは、そのパッケージ内のCOMサーバが動作する時のアカウントを指定します。IDを指定して“次へ(N)”ボタンをクリックすると、完了画面が表示されて、空アプリケーションの作成が完了します。

### 3. コンポーネントの登録

2.で作成したアプリケーションを開いて、コンポーネントをクリックします。[操作]メニューの“新規作成”で、“コンポーネント”を選択します。

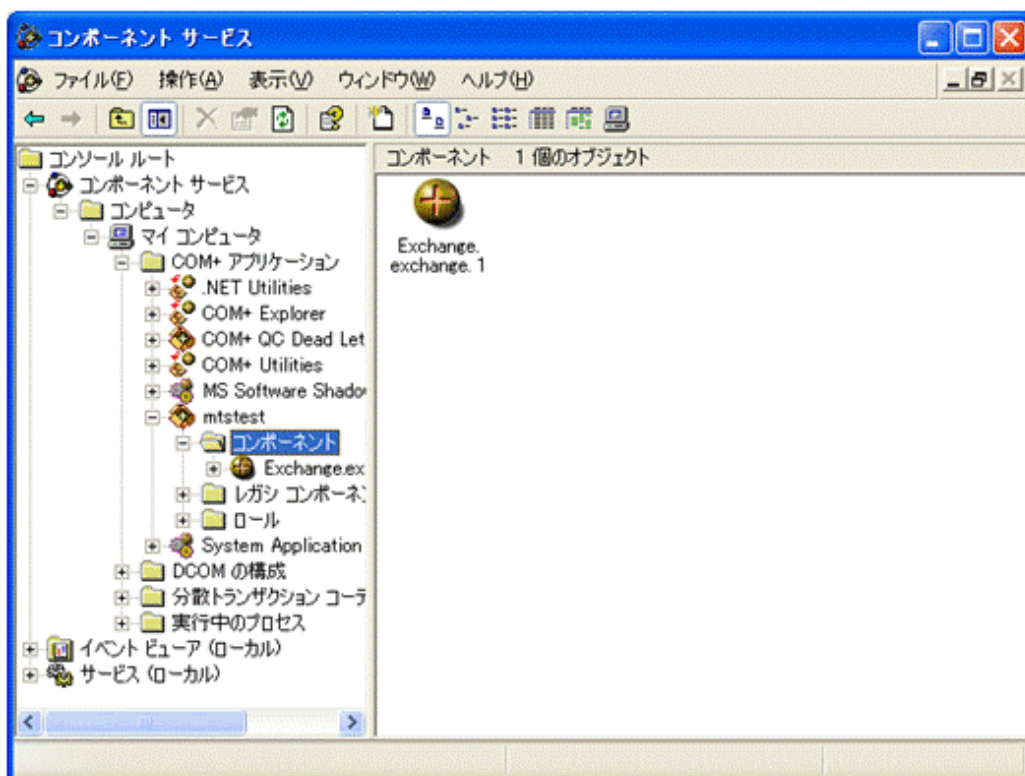


COM+ コンポーネントインストール ウィザードが表示されたら、“次へ(N)”をクリックし、“新しいコンポーネントをインストールする”をクリックします。インストールするファイルの選択ウィンドウが表示されたら、登録するDLLを指定します。このウィンドウが表示されずにコンポーネントのインストールウィンドウが表示されたら、[ファイルの追加]ボタンをクリックして登録するDLLを指定します。



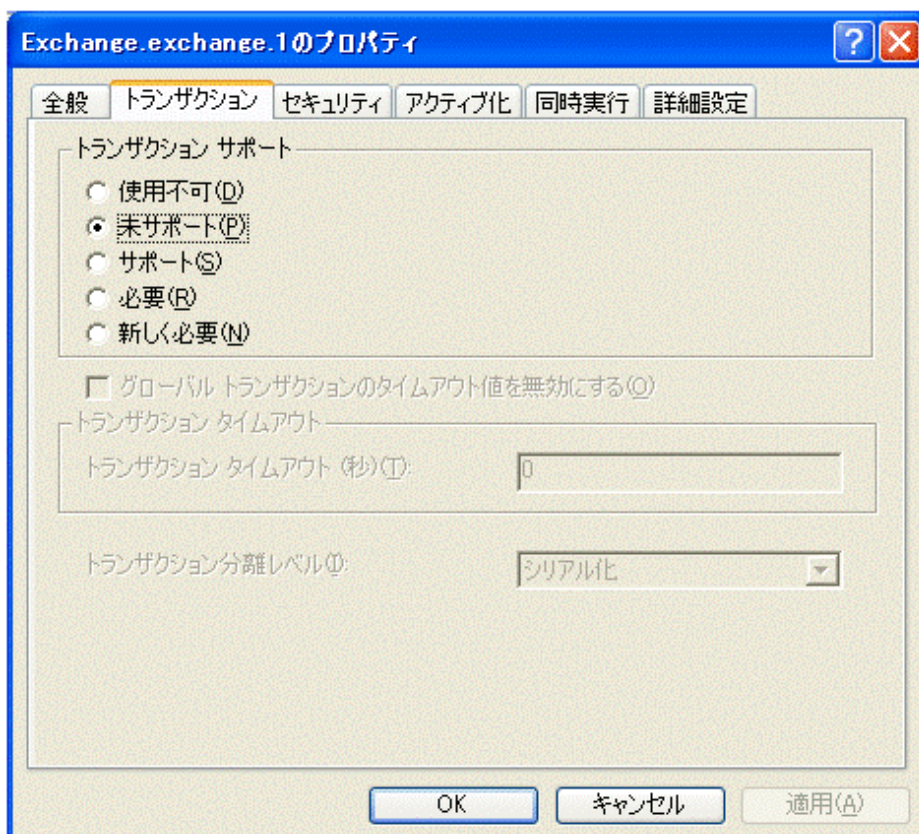


インストールが完了すると、MTSエクスプローラに追加されたCOMサーバが表示されます。



#### 4. プロパティの設定

登録されたコンポーネントのプロパティを設定します。プロパティを設定するコンポーネントをクリックして、[操作]メニューの“プロパティ”を選択すると、プロパティ設定ウィンドウが表示されます。



[全般] ページに設定する項目は特にありません。

[トランザクション] ページでは、トランザクションの制御方法を指定します。ここで登録するCOMサーバはトランザクションを行わないため、“未サポート”を選択します。

すべての設定が完了したら、[OK] ボタンをクリックします。これでCOMサーバのMTSへの登録が完了しました。

なお、プロパティ設定ウィンドウの[トランザクション] ページの情報は、以下のように対応しています。

| コンポーネントサービス | MTSエクスプローラ       |
|-------------|------------------|
| 必要          | トランザクションが必要      |
| 新しく必要       | 新しいトランザクションが必要   |
| サポート        | トランザクションをサポートする  |
| 未サポート       | トランザクションをサポートしない |
| 使用不可        | —                |

#### 5. レジストリの登録

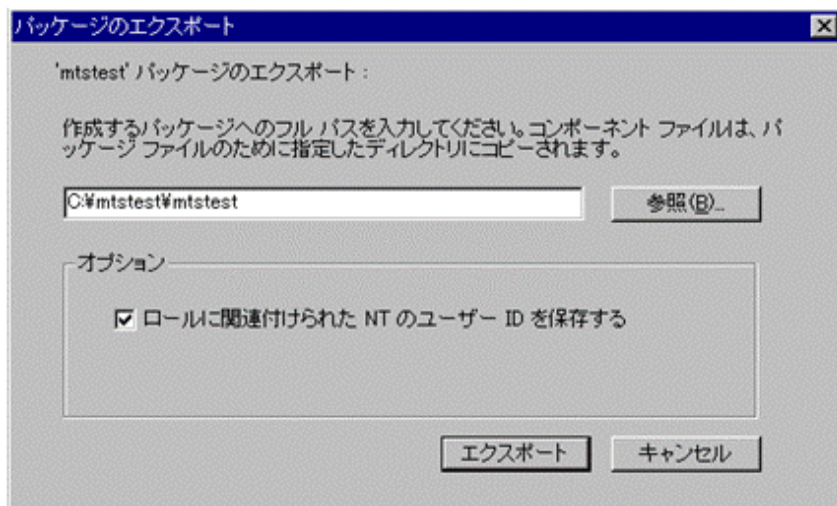
COMサーバをMTSに登録した場合、レジストリへの登録はMTSエクスプローラが行うため、REGSVR32.EXEを使って登録する必要はありません。

## 26.4.3 クライアントマシンへのインストール

MTSに登録したCOMサーバを別のマシン上にあるクライアントから利用する場合、クライアント側にサーバの情報をインストールする必要があります。クライアントにサーバの情報をインストールするインストーラの作成には、MTSエクスプローラのエクスポート機能を使用します。以下にインストーラの作成方法について説明します。

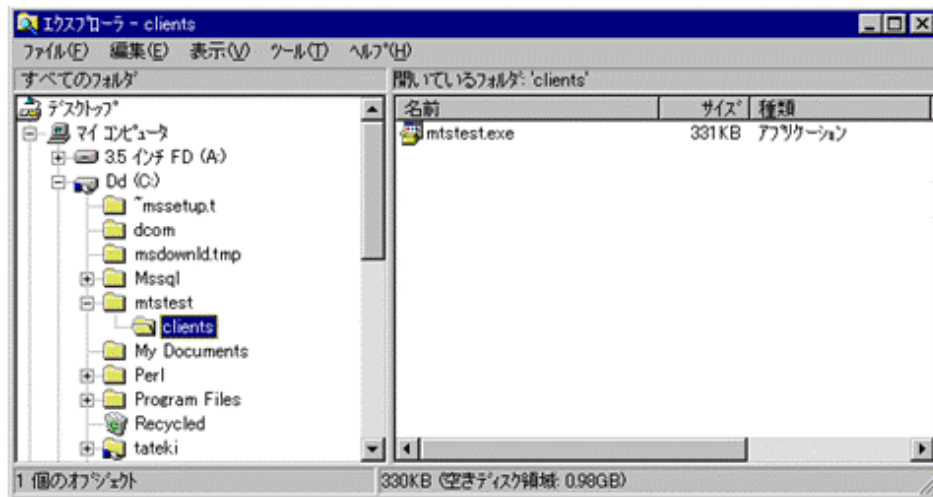
#### 1. パッケージのエクスポート

クライアント環境のインストーラは、パッケージ単位で作成します。インストーラを作成するパッケージをクリックして、[操作]メニューの“エクスポート”を選択すると、パッケージのエクスポートウィンドウが表示されます。



パッケージのエクスポートウィンドウで、インストーラを作成する場所を指定します。ここでは、C:\mtstest\mtstestと入力しています。[エクスポート] ボタンをクリックすると、mtstest.PAKファイルとclientsフォルダが作成され、同じフォルダにCOMサーバDLLがコピーされます。clientsフォルダにはEXE(mtstest.exe)ファイルが作成されます。

「アプリケーションをCOM+ 1.0形式で保存する」チェックボックスをチェック状態にしてください。



## 2. クライアントマシンでの実行

エクスポート機能を使って作成したEXEファイルを、クライアントプログラムを実行するマシン上で実行します。これで、サーバの情報がクライアント側にインストールされます。



### 注意

Windowsでコンポーネントサービスを使ってクライアント情報のインストールプログラムを作成する場合、作成されるファイルの形式はインストーラパッケージ(.MSI)になります。MSI(Microsoft Windows Installer)は、そのまま実行できます。

## 26.4.4 MTSのトランザクション管理

ここでは、MTSのトランザクション管理機能について説明します。

トランザクション管理を行う場合、コンポーネントのプロパティ設定ウィンドウ(“[26.4.2 MTS環境への登録方法](#)”の4.プロパティの設定)から、トランザクションの制御方法を指定します。制御方法には、以下の種類があります。

| プロパティ            | 意味                                                                         |
|------------------|----------------------------------------------------------------------------|
| トランザクションが必要      | トランザクション配下で起動された場合、その時点のトランザクションを継承して起動されます。そうでなければ新しくトランザクションを開設して起動されます。 |
| 新しいトランザクションが必要   | 常に新しいトランザクションを開設して起動されます。                                                  |
| トランザクションをサポートする  | トランザクション配下で起動された場合、その時点のトランザクションを継承して起動されます。そうでなければトランザクションとは無関係に起動されます。   |
| トランザクションをサポートしない | 常にトランザクションとは無関係に起動されます。                                                    |

COBOLアプリケーションからトランザクションを制御するには、以下の方法があります。

- ・COMサーバオブジェクトからオブジェクトが動作しているトランザクションを制御する。[参照]“[26.4.4.1 COMサーバからのトランザクション制御](#)”
- ・COMクライアントからCOMサーバが動作しているトランザクションを制御する。[参照]“[26.4.4.3 COMクライアントからのトランザクション制御](#)”

トランザクションの基本的な考え方については、“[26.4.5.1 トランザクションの基本的な考え方](#)”を参照してください。

### 26.4.4.1 COMサーバからのトランザクション制御

MTSで管理されるトランザクションを持ったCOMサーバから、そのオブジェクトが動作しているトランザクションを制御するには、オブジェクトコンテキストオブジェクトを使用します。

## オブジェクトコンテキストオブジェクト

オブジェクトコンテキストオブジェクトは、MTSが管理するオブジェクトの状態情報(コンテキスト)を管理するオブジェクトです。

オブジェクトコンテキストオブジェクトは、MTSが管理するCOMサーバクラスのオブジェクトを生成するごとに、必ず同時に作成されます。作成されたオブジェクトコンテキストオブジェクトは、対応するCOMサーバオブジェクトのトランザクションを制御します。

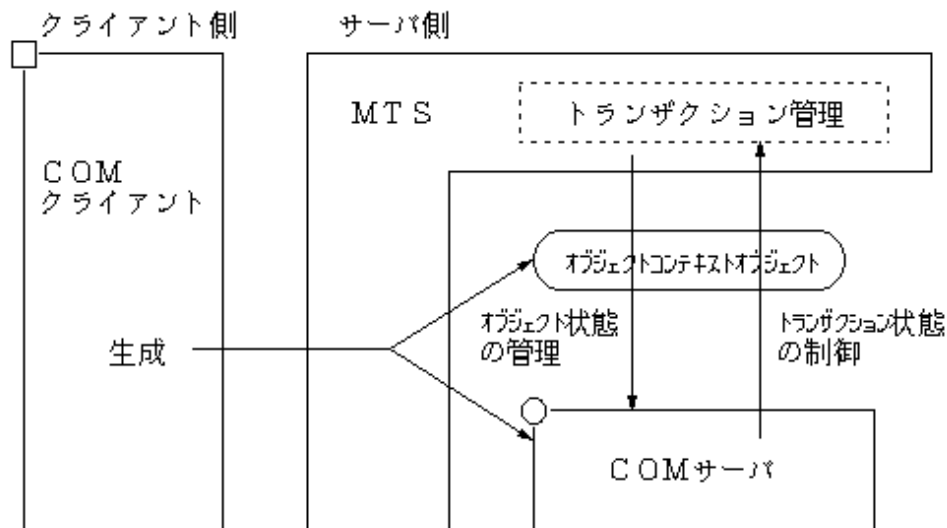
MTSはオブジェクトコンテキストオブジェクトのメソッドを呼び出すことで、以下の処理を行うことができます。

- ・ オブジェクトが実行されているトランザクションを完了または破棄する。
- ・ オブジェクトが実行されているトランザクションの状態を参照または更新する。
- ・ オブジェクトの状態情報を継承したオブジェクトを新しく生成する。



### 参考

オブジェクトコンテキストオブジェクトは、MTSのIObjectContextインターフェースを使用するためのオブジェクトです。



特殊クラス\*COMのファクトリメソッド“GETOBJECTCONTEXT”を呼び出すと、オブジェクトコンテキストオブジェクトのオブジェクト参照を取得できます。取得したオブジェクト参照を使って、オブジェクトコンテキストオブジェクトのメソッドを呼び出します。

### GETOBJECTCONTEXTメソッド

オブジェクトコンテキストオブジェクトのオブジェクト参照を取得し、返却します。引数はありません。

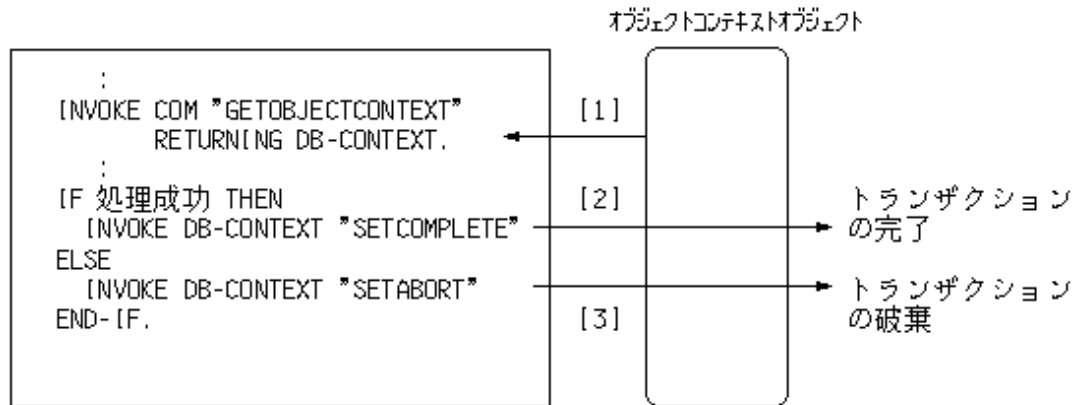


### 例

```
REPOSITORY.
 CLASS COM AS “*COM”.
 :
 01 DB-CONTEXT OBJECT REFERENCE COM.
 :
 INVOKE COM “GETOBJECTCONTEXT” RETURNING DB-CONTEXT.
```

オブジェクトコンテキストオブジェクトの使用例を以下に示します。





[1] オブジェクトコンテキストオブジェクトのオブジェクト参照(DB-CONTEXT)を取得します。

[2] SetCompleteメソッドを呼び出して、COMサーバオブジェクトが実行されているトランザクションを完了します。

[3] SetAbortメソッドを呼び出して、COMサーバオブジェクトが実行されているトランザクションを破棄します。

オブジェクトコンテキストオブジェクトのメソッドについては、“[26.4.4.2 オブジェクトコンテキストオブジェクトのメソッド](#)”を参照してください。

複数のCOMサーバのトランザクションを制御する方法については、“[26.4.5.4 トランザクション範囲](#)”を参照してください。

## 26.4.4.2 オブジェクトコンテキストオブジェクトのメソッド

オブジェクトコンテキストオブジェクトのメソッドには、以下のメソッドがあります。表中のSCODE値の意味については、“メッセージ説明書”の“付録B SCODE形式のエラーコードの説明”を参照してください。

表26.6 オブジェクトコンテキストオブジェクトのメソッド

| メソッド           | 引数   | 意味                                                                                                                                                                                                                                                                                                                      | 発生する例外SCODE値                                                                     |
|----------------|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|
| CreateInstance | X(n) | 引数で渡される文字列をProgIDとしてオブジェクトを生成して返却します。生成されるオブジェクトのトランザクションは、生成されるサーバのトランザクションプロパティの指定によって以下のようになります。 <ul style="list-style-type: none"> <li>「トランザクションが必要」または「トランザクションをサポートする」の場合には、トランザクションを引き継ぎます。</li> <li>「新しいトランザクションが必要」の場合には、新しいトランザクションが開設されます。</li> <li>「トランザクションをサポートしない」の場合には、トランザクションとは無関係に生成されます。</li> </ul> | 0x8004E005<br>0x80040154<br>0x8007000E<br>0x80020005<br>0x8002000E<br>0x8001FFFF |
| DisableCommit  | —    | オブジェクトが実行されているトランザクションの結果を破棄します。SetAbortとは異なり、クライアントに戻った時点でトランザクションの破棄は実行されません。トランザクションの破棄は、現在のトランザクション全体が完了する時まで実行されません。また、オブジェクトは状態を保持したまま、破棄されません。(注)                                                                                                                                                                | 0x8002000E<br>0x8001FFFF<br>0x8004E004                                           |
| EnableCommit   | —    | オブジェクトが実行されているトランザクションを完了します。SetCompleteとは異なり、クライアントに戻った時点でトランザクションの完了は実行されません。トランザクションの完了は、現在のトランザクション全体が完了する時まで実行されません。また、オブジェクトは状態を保持したまま、破棄されません。(注)                                                                                                                                                                | 0x8002000E<br>0x8001FFFF                                                         |
| IsCallerInRole | X(n) | 引数で渡された文字列が、直接の呼出し元のロールに入っているかどうかを調べ、入っていれば1、そうでなければ0をBITで返却します。                                                                                                                                                                                                                                                        | 0x8002000E<br>0x80020005                                                         |

| メソッド              | 引数 | 意味                                                                 | 発生する例外SCODE値             |
|-------------------|----|--------------------------------------------------------------------|--------------------------|
|                   |    |                                                                    | 0x8001FFFF<br>0x8004E00C |
| IsInTransaction   | —  | 現在、トランザクションを持っているかどうかを調べます。トランザクションを持っているれば1、そうでなければ0をBITで返却します。   | 0x8002000E               |
| IsSecurityEnabled | —  | カレントオブジェクトでセキュリティが有効になっているかどうかを調べます。有効になっていれば1、そうでなければ0をBITで返却します。 | 0x8002000E               |
| SetAbort          | —  | オブジェクトが実行されているトランザクションの結果を破棄します。オブジェクトは、クライアントに戻った時点で破棄されます。(注)    | 0x8002000E<br>0x8001FFFF |
| SetComplete       | —  | オブジェクトが実行されているトランザクションを完了します。オブジェクトは、クライアントに戻った時点で破棄されます。(注)       | 0x8002000E<br>0x8001FFFF |

注:トランザクションの完了/失敗、オブジェクトの保存/破棄には、以下の関係があります。

| メソッド          | トランザクション | オブジェクト | 実行のタイミング |
|---------------|----------|--------|----------|
| SetAbort      | 失敗       | 破棄     | 呼出し元に戻った |
| SetComplete   | 成功       | 破棄     | 呼出し元に戻った |
| DisableCommit | 失敗       | 保存     | 全体の完了時   |
| EnableCommit  | 成功       | 保存     | 全体の完了時   |

## 注意

同一メソッド内で、SetAbort、SetComplete、DisableCommit、EnableCommitメソッドの各メソッドが複数個呼び出された場合には、呼出し元に戻る直前に呼び出されたものが有効になります。また、これらのメソッドがどれも呼び出されなかった場合、EnableCommitメソッドが呼び出された場合と同じ状態になります。

SetCompleteおよびEnableCommitメソッドが成功するか失敗するかは、同じコンテキストを引き継いだオブジェクトでトランザクションが成功したか失敗したかに依存します。失敗した場合、SCODE値(0x8004E002)の例外が発生し、呼出し元に通知されます。

そのほかのメソッドの使い方については、“[26.4.5 MTSのトランザクション処理](#)”を参照してください。

## 26.4.4.3 COMクライアントからのトランザクション制御

COMクライアントからMTSのトランザクションを制御する場合、トランザクションコンテキストオブジェクトを使用します。

### トランザクションコンテキストオブジェクト

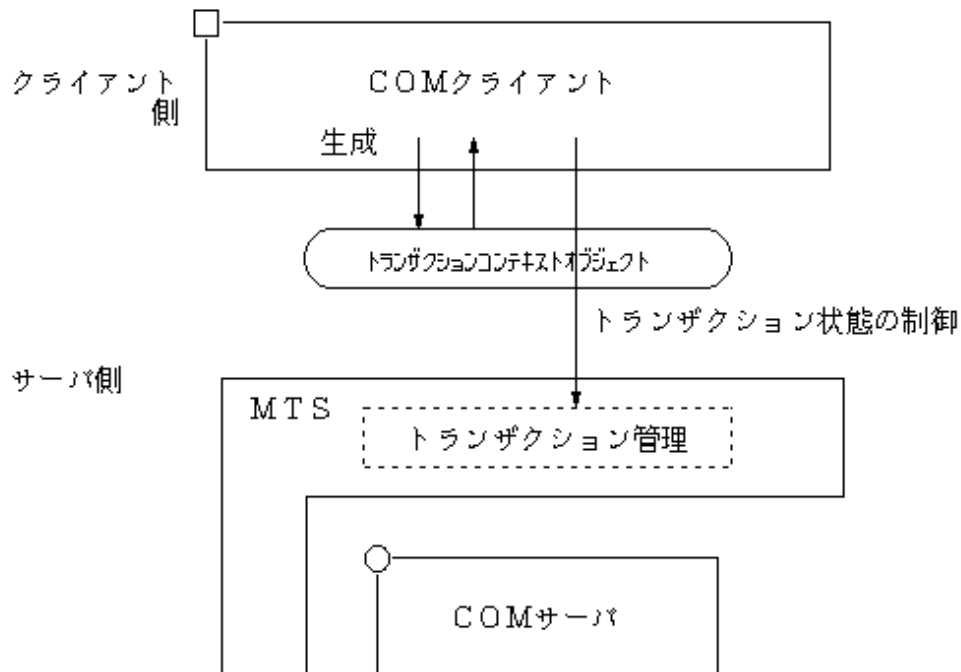
トランザクションコンテキストオブジェクトはトランザクションの状態情報(コンテキスト)を管理するオブジェクトです。このオブジェクトのメソッドを呼び出すことで、COMクライアントからCOMサーバのトランザクションを制御できます。

## 参考

トランザクションコンテキストオブジェクトは、MTSのITransactionContextEXインターフェースを使用するためのオブジェクトです。

## 注意

トランザクションコンテキストオブジェクトを使用する場合は、クライアントマシンにもMTSが必要です。



特殊クラス\*COMのファクトリメソッド“CREATE-OBJECT”に引数“TRANCONTEXT”を渡すと、トランザクションコンテキストオブジェクトを生成し、そのオブジェクト参照を取得できます。取得したオブジェクト参照を使用して、トランザクションコンテキストオブジェクトのメソッドを呼び出します。

CREATE-OBJECTメソッド  
 引数 : “TRANCONTEXT”  
 返却値 : \*COMクラスのオブジェクト参照



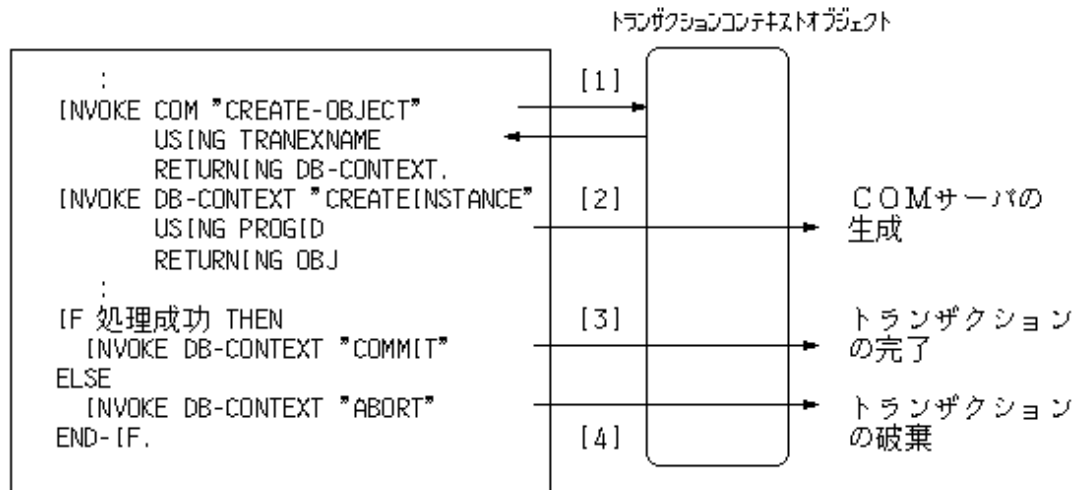
## 例

```

REPOSITORY.
 CLASS COM AS “*COM”.
 :
 01 DB-CONTEXT OBJECT REFERENCE COM.
 01 TRANEXNAME PIC X(11) VALUE “TRANCONTEXT”.
 :
 INVOKE COM “CREATE-OBJECT” USING TRANEXNAME
 RETURNING DB-CONTEXT.

```

トランザクションコンテキストオブジェクトの使用例を以下に示します。



- [1] トランザクションコンテキストオブジェクトを生成し、オブジェクト参照(DB-CONTEXT)を取得します。
- [2] COMサーバオブジェクトを生成します。ここで生成したオブジェクトのトランザクションは、[1]で生成したトランザクションコンテキストオブジェクトのメソッドで操作できます。
- [3] Commitメソッドを呼び出して、[2]で生成したCOMサーバのトランザクションを完了します。
- [4] Abortメソッドを呼び出して、[2]で生成したCOMサーバのトランザクションを破棄します。

トランザクションコンテキストオブジェクトのメソッドについては、“[26.4.4.4 トランザクションコンテキストオブジェクトのメソッド](#)”を参照してください。

複数のCOMサーバのトランザクションを制御する方法については、“[26.4.5.4 トランザクション範囲](#)”を参照してください。

## 26.4.4.4 トランザクションコンテキストオブジェクトのメソッド

トランザクションコンテキストオブジェクトのメソッドには、以下のメソッドがあります。表中のSCODE値の意味については、“メッセージ説明書”の“付録B SCODE形式のエラーコードの説明”を参照してください。

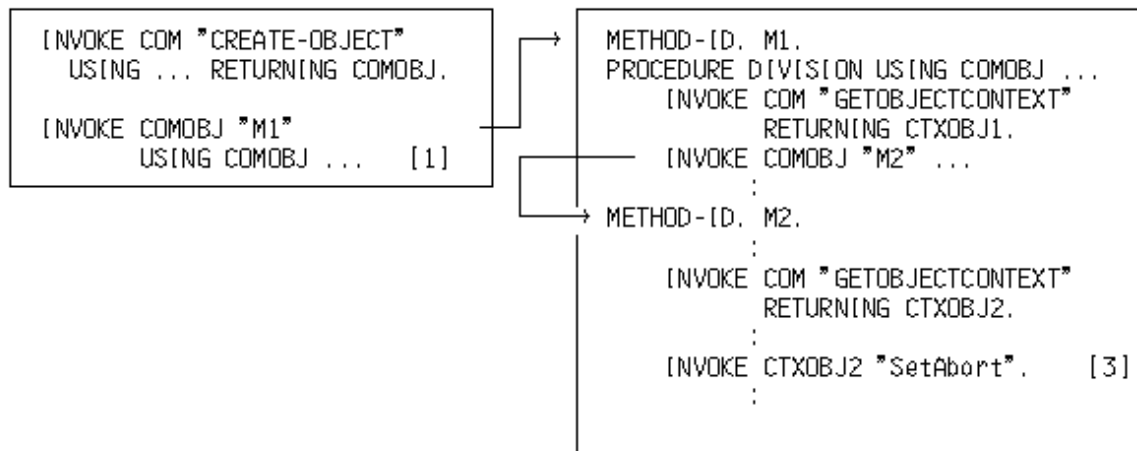
表26.7 トランザクションコンテキストオブジェクトのメソッド

| メソッド           | 引数   | 意味                                                                                                                                                                                                                                                              | 発生する例外SCODE値                                                                     |
|----------------|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|
| Abort          | —    | CreateInstanceメソッドで生成したサーバのトランザクションの結果を破棄します。(注1)                                                                                                                                                                                                               | 0x8002000E<br>0x80004005<br>0x8001FFFF                                           |
| Commit         | —    | CreateInstanceメソッドで生成したサーバのトランザクションの結果を反映します。ただし、サーバのどれかでSetAbortまたはDisableCommitが呼び出されている場合、トランザクションの結果は破棄されます。(注1)(注2)                                                                                                                                        | 0x8002000E<br>0x80004005<br>0x8001FFFF<br>0x8004E002                             |
| CreateInstance | X(n) | 引数で渡される文字列をProgIDとしてオブジェクトを生成して返却します。生成されるオブジェクトのトランザクションは、生成されるサーバのトランザクションプロパティの指定によって以下ようになります。 <ul style="list-style-type: none"> <li>「トランザクションが必要」または「トランザクションをサポートする」の場合には、トランザクションを引き継ぎます。</li> <li>「新しいトランザクションが必要」の場合には、新しいトランザクションが開設されます。</li> </ul> | 0x8004E005<br>0x80040154<br>0x8007000E<br>0x80020005<br>0x8002000E<br>0x8001FFFF |





## COMオブジェクトのメソッド呼出し



サーバ内で自分自身のメソッドを呼び出すため、COMサーバ自身のオブジェクトをサーバに渡します([1])。M1の中では、受け取ったオブジェクトでM2を呼び出します。このとき、M2の中でSetAbortメソッドを呼び出しているため、[2]のメソッド呼出しに戻った時点で、自分自身が破棄されます。これを回避するためには、[3]でSetAbortメソッドを呼び出す代わりに、DisableCommitメソッドを呼び出します。

## 26.4.5 MTSのトランザクション処理

### 26.4.5.1 トランザクションの基本的な考え方

MTSのトランザクション処理の基本的な考え方を以下に示します。

#### トランザクションの開始

COMサーバのトランザクションプロパティの指定を「トランザクションが必要」または「新しいトランザクションが必要」とすることで、サーバの起動と同時にトランザクションが開始されます。

あるいは、クライアント側でトランザクションコンテキストオブジェクトを生成することで、トランザクションを開始します。

#### コミット/ロールバック

トランザクションを持っているオブジェクトで、オブジェクトコンテキストオブジェクトを取得し、SetCompleteメソッドまたはSetAbortメソッドを呼び出すことで、トランザクションの結果を反映したり、破棄したりすることができます。また、トランザクションコンテキストオブジェクトを使用することで、変更反映/破棄がクライアント側でも制御できるようになります。

サーバ内で、オブジェクトコンテキストオブジェクトのSetCompleteメソッドまたはSetAbortメソッドを実行してクライアントプログラムに戻った場合、そのサーバのオブジェクトは使用できなくなります。

#### 原子性の保証

オブジェクトコンテキストオブジェクト、トランザクションコンテキストオブジェクトを使用し、サーバ起動時にトランザクションを引き継ぐことで、2フェーズコミットの制御ができるようになります。

### 26.4.5.2 トランザクションを制御するための追加処理

MTSでCOMサーバのトランザクションを制御するためには、COMサーバに以下の処理を追加します。

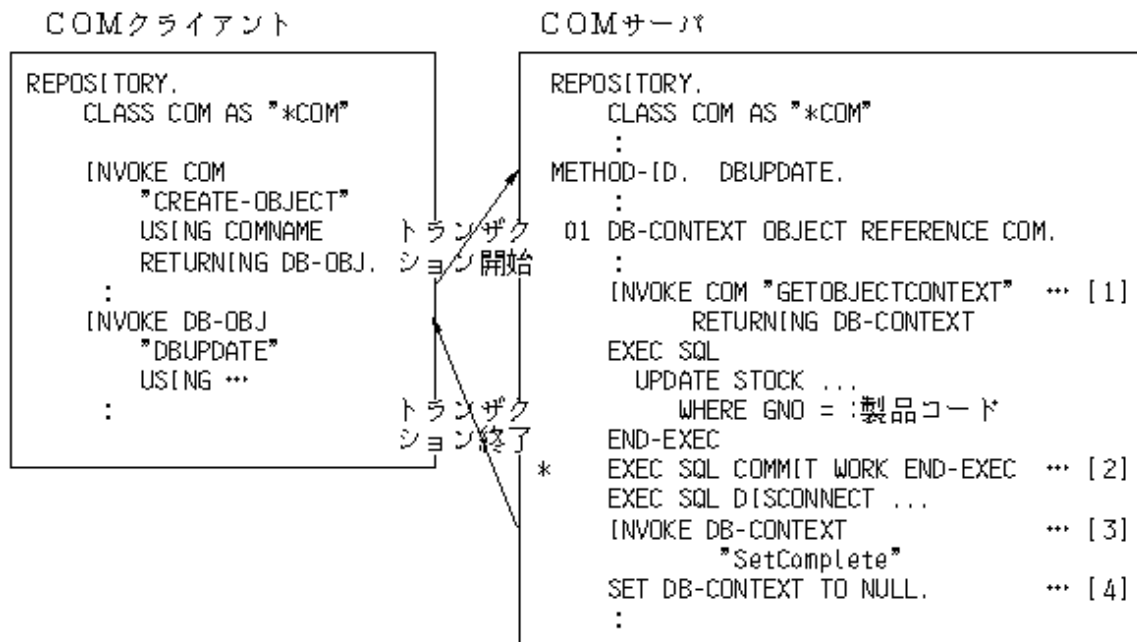
- オブジェクトコンテキストオブジェクトを取得する処理
- オブジェクトコンテキストオブジェクトを使用してSetCompleteメソッドまたはSetAbortメソッドを呼び出す処理

- ・ オブジェクトコンテキストオブジェクトを削除する処理

また、トランザクションの制御をMTSに任せるために、SQLのCOMMIT/ROLLBACK文があればコメントにします。

MTSの設定としては、対象となるCOMサーバのトランザクションプロパティで、「トランザクションが必要」あるいは「新しいトランザクションが必要」を指定します。

図26.11 MTS配下でのトランザクションを制御するための追加処理



[1] GETOBJECTCONTEXTメソッドでオブジェクトコンテキストオブジェクトを取得します。

[2] COMMIT文をコメント化します。

[3] オブジェクトコンテキストオブジェクトを使って、SetCompleteメソッドを呼び出し、トランザクションを終了します。

[4] オブジェクトコンテキストオブジェクトにNULLを設定し、オブジェクトコンテキストオブジェクトを削除します。

### 26.4.5.3 MTSを使ったトランザクションのプログラム例

ここでは、埋込みSQL文を使用したCOBOLプログラムを、COMクライアント/COMサーバ化し、MTSを使ってトランザクション制御を行うための修正方法について説明します。

#### アプリケーションの概略

- ・ DBACCESS1クラスは、DBREAD(データベースの検索)メソッドとDBUPDATE(データベースの更新)メソッドを持っています。
- ・ 使用するデータベースは“[22.2.3.1 サンプルデータベース](#)”を想定しています。

#### DBACCESS1クラスの定義

```

CLASS-ID. DBACCESS1 INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS FJBASE.
 OBJECT.
PROCEDURE DIVISION.
METHOD-ID. DBREAD.
DATA DIVISION.

```

```

WORKING-STORAGE SECTION.
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 SQLSTATE PIC X(5).
 EXEC SQL END DECLARE SECTION END-EXEC.
LINKAGE SECTION.
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 製品コード PIC S9(4) COMP-5.
01 製品名 PIC X(20).
01 在庫数 PIC S9(9) COMP-5.
01 倉庫番号 PIC S9(4) COMP-5.
 EXEC SQL END DECLARE SECTION END-EXEC.
01 CONNECT_STATE PIC S9(4) COMP-5.
PROCEDURE DIVISION USING 製品コード 製品名 在庫数 倉庫番号
 RETURNING CONNECT_STATE.
 EXEC SQL CONNECT TO DEFAULT END-EXEC.
 IF SQLSTATE = "00000" OR "01000" THEN
 MOVE 0 TO CONNECT_STATE
 EXEC SQL
 SELECT GOODS, QOH, WHNO INTO :製品名, :在庫数, :倉庫番号
 FROM STOCK WHERE GNO = :製品コード
 END-EXEC
 EXEC SQL DISCONNECT DEFAULT END-EXEC
 ELSE
 MOVE 1 TO CONNECT_STATE
 END-IF.
 EXIT METHOD.
END METHOD DBREAD.
*
METHOD-ID. DBUPDATE.
DATA DIVISION.
WORKING-STORAGE SECTION.
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 SQLSTATE PIC X(5).
 EXEC SQL END DECLARE SECTION END-EXEC.
LINKAGE SECTION.
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 製品コード PIC S9(4) COMP-5.
01 在庫数 PIC S9(9) COMP-5.
 EXEC SQL END DECLARE SECTION END-EXEC.
01 CONNECT_STATE PIC S9(4) COMP-5.
PROCEDURE DIVISION USING 製品コード 在庫数
 RETURNING CONNECT_STATE.
 EXEC SQL CONNECT TO DEFAULT END-EXEC.
 IF SQLSTATE = "00000" OR "01000" THEN
 MOVE 0 TO CONNECT_STATE
 EXEC SQL
 UPDATE STOCK SET QOH = :在庫数
 WHERE GNO = :製品コード
 END-EXEC
 EXEC SQL COMMIT WORK END-EXEC
 EXEC SQL DISCONNECT DEFAULT END-EXEC
 ELSE
 MOVE 1 TO CONNECT_STATE
 END-IF.
 EXIT METHOD.
END METHOD DBUPDATE.
END OBJECT.
END CLASS DBACCESS1.

```

まず、DBACCESS1クラスをCOMサーバ化します。COMサーバ作成時のDLL名は“SAMPLE.DLL”とします。

このCOMサーバを呼び出すCOMクライアントプログラムの例を以下に示します。

## クライアントプログラムの定義

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TESTCLIENT.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS COM AS "COM".
DATA DIVISION.
WORKING-STORAGE SECTION.
01 製品コード PIC S9(4) COMP-5.
01 製品名 PIC X(20).
01 在庫数 PIC S9(9) COMP-5.
01 倉庫番号 PIC S9(4) COMP-5.
01 ACCESS-MODE PIC 9(1).
01 DB-OBJECT OBJECT REFERENCE COM.
01 CONNECT-STAT PIC S9(4).
01 COMNAME PIC X(30) VALUE "SAMPLE.DBACCESS1".
PROCEDURE DIVISION.
 DISPLAY NC"サンプルプログラムの開始".
 INVOKE COM "CREATE-OBJECT" USING COMNAME
 RETURNING DB-OBJECT.
ACCESS-START.
 DISPLAY NC"参照なら0を更新なら1を終了なら9を入力.".
 ACCEPT ACCESS-MODE.
 IF ACCESS-MODE = 0 THEN
 DISPLAY NC"製品コードを入力してください.".
 ACCEPT 製品コード
 INVOKE DB-OBJECT "DBREAD"
 USING 製品コード 製品名 在庫数 倉庫番号
 RETURNING CONNECT-STAT
 IF CONNECT-STAT = 1 THEN
 DISPLAY NC"参照に失敗しました.".
 ELSE
 DISPLAY NC"製品コード=" 製品コード
 DISPLAY NC"製品名 =" 製品名
 DISPLAY NC"在庫数 =" 在庫数
 DISPLAY NC"倉庫番号 =" 倉庫番号
 END-IF
 ELSE
 IF ACCESS-MODE = 1 THEN
 DISPLAY NC"製品コードと新在庫数を入力してください.".
 ACCEPT 製品コード
 ACCEPT 在庫数
 INVOKE DB-OBJECT "DBUPDATE" USING 製品コード 在庫数
 RETURNING CONNECT-STAT
 IF CONNECT-STAT = 1 THEN
 DISPLAY NC"更新は失敗しました.".
 ELSE
 DISPLAY NC"更新は成功しました.".
 END-IF
 ELSE
 IF ACCESS-MODE = 9 THEN
 GO TO ACCESS-END
 END-IF
 END-IF
 END-IF.
 GO TO ACCESS-START.
ACCESS-END.
 SET DB-OBJECT TO NULL.
END PROGRAM TESTCLIENT.
```

次にCOMサーバ化したDBACCESS1クラスを、MTSを使ってトランザクション制御を行うように修正します。

## MTSでトランザクション制御を行うための修正

```

CLASS-ID. DBACCESS1 INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
 CLASS FJBASE
 CLASS COM AS "*COM".
OBJECT.
PROCEDURE DIVISION.
METHOD-ID. DBREAD.
DATA DIVISION.
WORKING-STORAGE SECTION.
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 SQLSTATE PIC X(5).
 EXEC SQL END DECLARE SECTION END-EXEC.
LINKAGE SECTION.
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 製品コード PIC S9(4) COMP-5.
01 製品名 PIC X(20).
01 在庫数 PIC S9(9) COMP-5.
01 倉庫番号 PIC S9(4) COMP-5.
 EXEC SQL END DECLARE SECTION END-EXEC.
01 CONNECT_STATE PIC S9(4) COMP-5.
PROCEDURE DIVISION USING 製品コード 製品名 在庫数 倉庫番号
 RETURNING CONNECT_STATE.
 EXEC SQL CONNECT TO DEFAULT END-EXEC.
 IF SQLSTATE = "00000" OR "01000" THEN
 MOVE 0 TO CONNECT_STATE
 EXEC SQL
 SELECT GOODS, QOH, WHNO INTO :製品名, :在庫数, :倉庫番号
 FROM STOCK WHERE GNO = :製品コード
 END-EXEC
 EXEC SQL DISCONNECT DEFAULT END-EXEC
 ELSE
 MOVE 1 TO CONNECT_STATE
 END-IF.
EXIT METHOD.
END METHOD DBREAD.
*
METHOD-ID. DBUPDATE.
DATA DIVISION.
WORKING-STORAGE SECTION.
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 SQLSTATE PIC X(5).
 EXEC SQL END DECLARE SECTION END-EXEC.
01 DB-CONTEXT OBJECT REFERENCE COM.
LINKAGE SECTION.
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 製品コード PIC S9(4) COMP-5.
01 在庫数 PIC S9(9) COMP-5.
 EXEC SQL END DECLARE SECTION END-EXEC.
01 CONNECT_STATE PIC S9(4) COMP-5.
PROCEDURE DIVISION USING 製品コード 在庫数
 RETURNING CONNECT_STATE.
 INVOKE COM "GETOBJECTCONTEXT" RETURNING DB-CONTEXT.
 EXEC SQL CONNECT TO DEFAULT END-EXEC.
 IF SQLSTATE = "00000" OR "01000" THEN
 MOVE 0 TO CONNECT_STATE
 EXEC SQL
 UPDATE STOCK SET QOH = :在庫数
 WHERE GNO = :製品コード
 END-EXEC

```

```

* EXEC SQL COMMIT WORK END-EXEC ...[3]
 EXEC SQL DISCONNECT DEFAULT END-EXEC
 INVOKE DB-CONTEXT "SetComplete" ...[4]
ELSE
 MOVE 1 TO CONNECT_STATE
 INVOKE DB-CONTEXT "SetAbort" ...[5]
END-IF.
EXIT METHOD.
END METHOD DBUPDATE.
END OBJECT.
END CLASS DBACCESS1.

```

[1] \*COMクラスを利用することを宣言します。

[2] オブジェクトコンテキストオブジェクトを取得します。

[3] COMMIT文をコメント化します。

[4] COMMIT文の代わりに、オブジェクトコンテキストオブジェクトを使って、SetCompleteメソッドを呼び出し、トランザクションを終了させます。

[5] オブジェクトコンテキストオブジェクトを使って、SetAbortメソッドを呼び出し、トランザクションの結果を破棄します。

COMサーバの修正が完了したら、COMサーバをMTSに登録します。MTSへの登録方法については、“[26.4.2 MTS環境への登録方法](#)”を参照してください。このとき、トランザクションのプロパティの設定で、「トランザクションが必要」または「新しいトランザクションが必要」のどちらかを選択してください。

#### 26.4.5.4 トランザクション範囲

トランザクション処理を行う複数のCOMサーバの実行には以下の方法があります。

- ・ 別々のトランザクションとして実行する方法
- ・ サーバ側で1つのトランザクションとして実行する方法
- ・ クライアント側で1つのトランザクションとして実行する方法

それぞれの方法について説明します。なお、以下の図の中で使用している太線は、トランザクションの範囲を表します。

##### 別々のトランザクションとして実行する方法

COMサーバをMTSに登録するとき、トランザクションのプロパティの設定で「新しいトランザクションが必要」を指定することで、そのCOMサーバのトランザクションは、他のトランザクションとは独立したものとなります。

図26.12 サーバからサーバを起動する場合

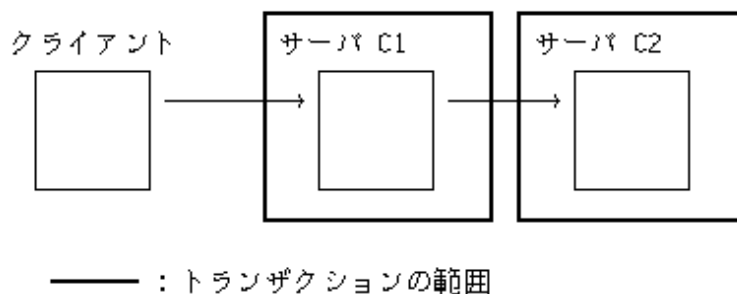
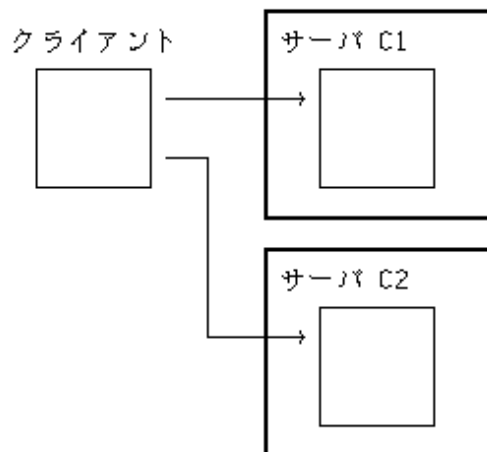


図26.13 クライアントからサーバを別々に起動する場合

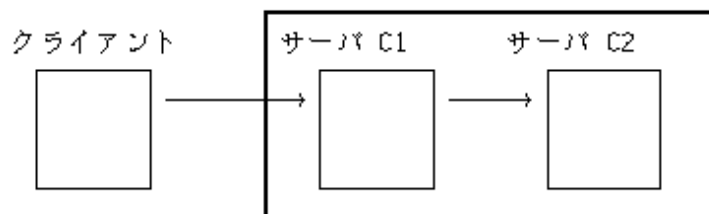


### サーバ側でまとめて一つのトランザクションとして実行する方法

サーバC1をMTSに登録するときは、トランザクションのプロパティの設定で「トランザクションが必要」または「新しいトランザクションが必要」のどちらかを指定します。サーバC2に登録するときは、「トランザクションが必要」または「トランザクションをサポートする」のどちらかを指定します。

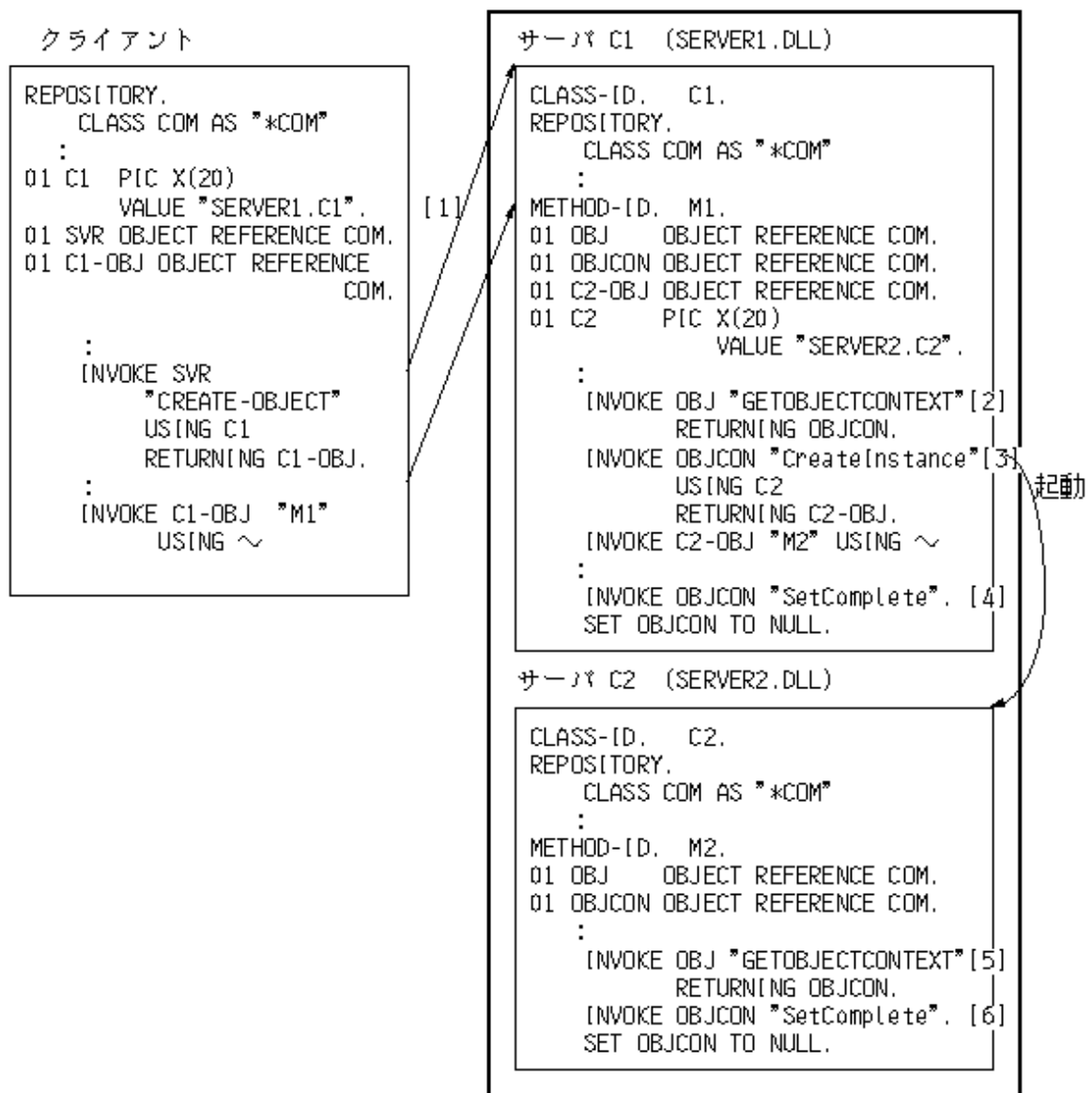
クライアントからサーバC1が起動されたとき、この時点ではトランザクションがまだないためサーバC1のトランザクションプロパティによって、新しいトランザクションが開設されます。次に、サーバC1の中からサーバC2を起動したとき、サーバC2のトランザクションプロパティによって、サーバC1のトランザクションが継承されます。

図26.14 トランザクションを継承してサーバを起動する場合



サーバ側で一つのトランザクションとして実行する場合のプログラム例を、以下に示します。





- [1] クライアントからサーバC1を起動した時に、サーバ側で新規にトランザクションが開設されます。
- [2] オブジェクトコンテキストオブジェクトを取得します。
- [3] オブジェクトコンテキストオブジェクトを使って、CreateInstanceメソッドを呼び出し、サーバC2を起動します。このときサーバC2はサーバC1のトランザクションを継承します。
- [4] オブジェクトコンテキストオブジェクトを使って、SetCompleteメソッドを呼び出し、トランザクションを終了します。
- [5] オブジェクトコンテキストオブジェクトを取得します。
- [6] オブジェクトコンテキストオブジェクトを使って、SetCompleteメソッドを呼び出し、トランザクションを終了します。

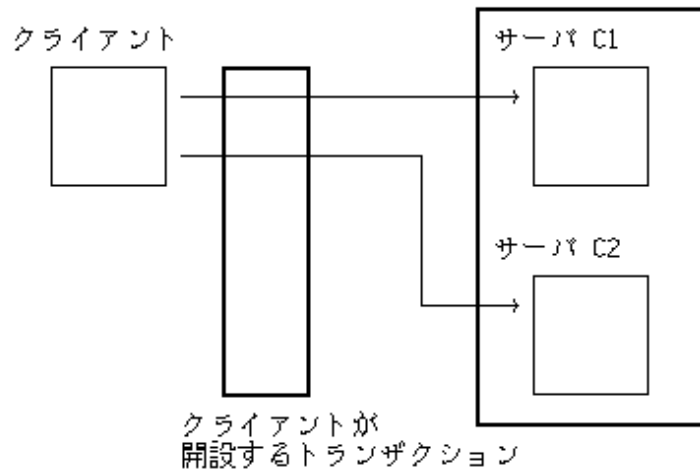
サーバC1、サーバC2を合わせたトランザクションは、両方のサーバでSetCompleteされたときに成功します。

### クライアント側で一つのトランザクションとして実行する方法

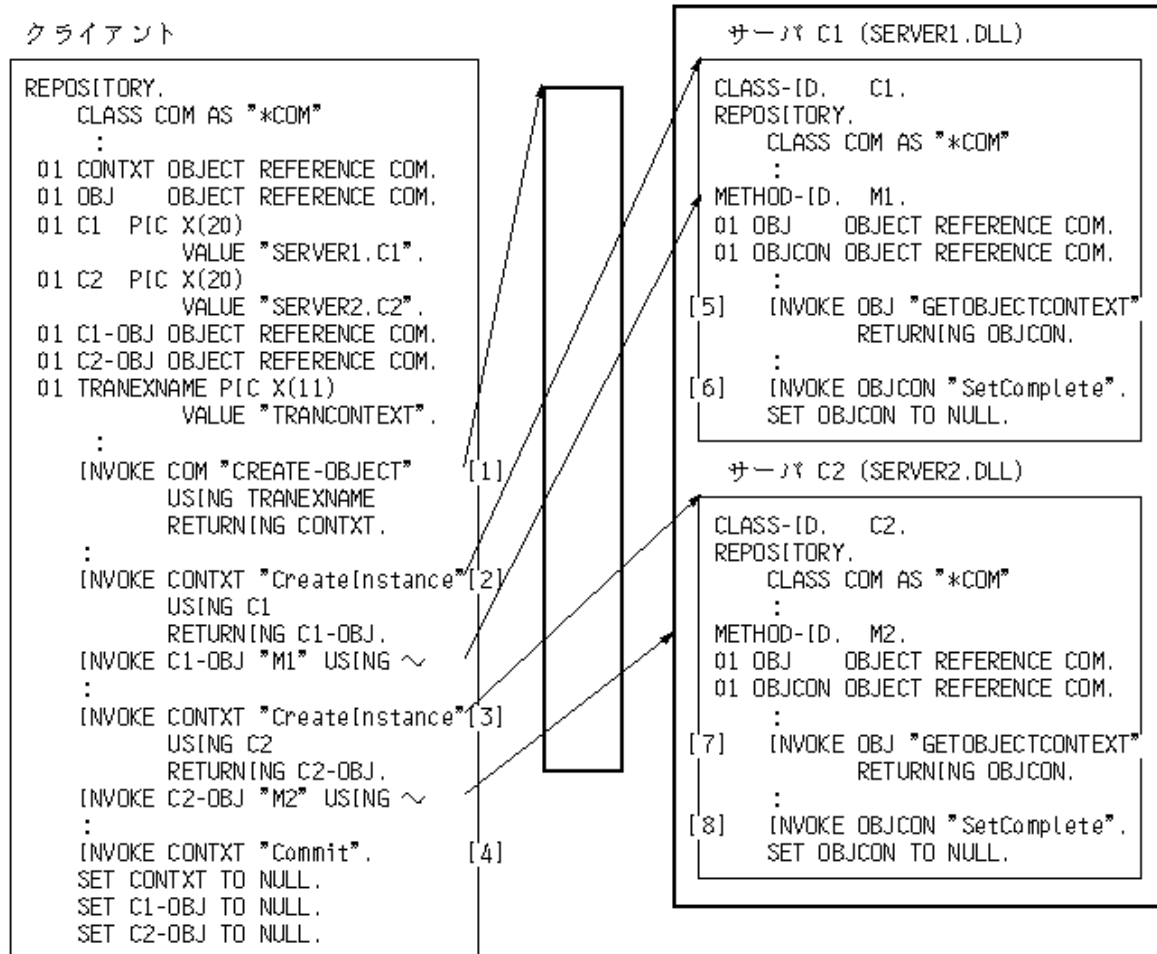
サーバC1およびサーバC2をMTSに登録するときに、トランザクションのプロパティの設定で「トランザクションが必要」または「トランザクションをサポートする」のどちらかを指定します。

この方法では、クライアントであらかじめトランザクションを開設してからサーバを起動します。サーバC1およびC2が起動されたとき、クライアントで開設しておいたトランザクションが継承されます。

図26.15 クライアントで1つのトランザクションとする場合



クライアント側で1つのトランザクションとして実行する場合のプログラム例を、以下に示します。



- [1] CREATE-OBJECTメソッドに引数“TRANCONTEXT”を渡して呼び出し、トランザクションコンテキストオブジェクトを生成します。
- [2] トランザクションコンテキストオブジェクトを使って、CreateInstanceメソッドを呼び出し、サーバC1を起動します。このとき、サーバC1はクライアントが生成したトランザクションを継承します。
- [3] トランザクションコンテキストオブジェクトを使って、CreateInstanceメソッドを呼び出し、サーバC2を起動します。このとき、サーバC2はクライアントが生成したトランザクションを継承します。
- [4] トランザクションコンテキストオブジェクトを使って、Commitメソッドを呼び出し、トランザクションを終了します。
- [5] オブジェクトコンテキストオブジェクトを取得します。
- [6] オブジェクトコンテキストオブジェクトを使って、SetCompleteメソッドを呼び出し、トランザクションを終了します。
- [7] オブジェクトコンテキストオブジェクトを取得します。
- [8] オブジェクトコンテキストオブジェクトを使って、SetCompleteメソッドを呼び出し、トランザクションを終了します。

クライアント、サーバC1およびサーバC2を合わせたトランザクションは、すべてSetCompleteされたときに成功します。

## 26.4.5.5 MTS配下での動作上の注意

## 全般的な注意

### COMサーバのスレッドモードについて

MTSに登録するサーバは、マルチスレッドで動作するように作成します。マルチスレッドで動作させるには、COMサーバの作成時に、翻訳オプションTHREAD(MULTI)を指定します。[参照]“[26.3.3 COMサーバの作成](#)”

### COBOLの実行環境の初期化について

MTSでは、COMサーバのDLLはパッケージと呼ばれる単位で管理され、同じパッケージに登録されたDLLは、同じプロセス内で実行されます。このときの実行用の初期化ファイルの作成方法については、“[5.3.2.3.2 実行用の初期化ファイルの内容](#)”および“[5.3.2.3.3 実行用の初期化ファイルの検索順序について](#)”を参考にしてください。

### DLLの編集について

MTSは、一度ロードしたDLLは次の使用に備えてプロセス内に留めておきます。このため、そのDLLを作り直そうとしたとき、“他から使用されているため書き込めません”というエラーになることがあります。この場合、MTSエクスプローラのウィンドウに表示されるツリー上の“マイコンピュータ”を右ボタンをクリックすると表示されるショートカットメニューから、“サーバプロセスのシャットダウン”を選択します。サーバプロセスをシャットダウンした後は、DLLの編集が可能になります。

### DLLのアンロードのタイミングについて

MTSに登録されたCOMサーバのDLLは、クライアントから使用されなくなってもシャットダウンされない限りアンロードされません。COMサーバのシャットダウンは、MTSエクスプローラのマイコンピュータの操作またはパッケージのプロパティの詳細設定で制御できます。

## トランザクション操作のためのメソッドの使い方に関する注意

### SetComplete/SetAbortメソッド実行後のオブジェクトの状態

サーバのオブジェクト内でSetCompleteメソッドまたはSetAbortメソッドを呼び出した後、呼出し元に復帰すると、そのサーバのオブジェクトは使用できなくなります。したがって、サーバから復帰する前に、データベースやファイルのクローズを行うか、EnableCommitメソッドまたはDisableCommitメソッドを呼び出してオブジェクトの状態を保存する必要があります。

ただし、MTSを使ってトランザクションを行うサーバの場合、オブジェクトの状態は保持しないようにしてください。

### EnableCommit/DisableCommitメソッドを使用する場合

サーバクラス内で、トランザクション制御を行う自分自身のメソッドを呼び出す場合、呼出し先のメソッドでSetCompleteメソッドまたはSetAbortメソッドを実行すると、復帰後に自分自身のオブジェクトが使用できなくなります。こうした問題を回避したい場合にだけ、EnableCommitメソッドまたはDisableCommitメソッドを使用してください。[参照]“[26.4.4.5 注意事項](#)”

### MTSトランザクション管理の対象

COMサーバのメソッドとして呼び出されたものでなければ、MTSのトランザクション管理の対象になりません。つまり、COMサーバクラス内でSELFオブジェクトによってメソッドを呼び出し、そこでトランザクションを操作しようとしても有効になりません。

## データベース操作に関する注意

### データベースへの接続

SetCompleteメソッドまたはSetAbortメソッドを使って、トランザクション制御を行うCOMサーバでデータベースアクセスを行う場合、サーバは複数回の呼出しにまたがってデータベースとの接続状態を維持することができません。このため、“[26.4.5.3 MTSを使ったトランザクションのプログラム例](#)”のCOMサーバでは、各メソッドごとにデータベース接続処理を行っています。

### データベースへの反映のタイミング

オブジェクトコンテキストオブジェクトのSetCompleteメソッドおよびSetAbortメソッド、トランザクションコンテキストオブジェクトのCommitメソッドおよびAbortメソッドを実行しても、すぐにはデータベースに反映されません。そのサーバから呼出し元へ復帰した時点で反映されます。

### ODBC情報の設定について

MTSを使ってトランザクションを行う場合、コネクション有効範囲は“オブジェクトインスタンス”にします。また、COMMITモードは“オート”を選択します。設定方法については、“[22.2.8.2 ODBC情報設定ツールの使い方](#)”を参照してください。

## 26.5 COM連携で扱うデータ型の対応

COM連携で扱うデータ型の対応は、“[図26.16 COBOLのデータ型とCOMのデータ型の対応](#)”のようになります。

### 注意

COMサーバに渡すUSINGパラメタは、メソッド呼出し前に必ず適切な値を設定するかまたは初期化してください。

### 参考

ほとんどのCOMサーバは、渡されたデータをCOMサーバの処理に必要な型に変換して処理を行います。したがって、データ型は厳密に一致させる必要はありません。また、COM連携では、復帰項目に値を設定するときに、可能なかぎり型変換を行います。

図26.16 COBOLのデータ型とCOMのデータ型の対応

| COBOLのデータ型<br><br>COMのデータ型<br>(VT型) | 数字       |            |             |       | 文字 |           |           |       |       |      | 浮動小数点 |       |       | ブール |    | ポインタ | 指標 | オブジェクト参照 | 集団項目 |   |   |
|-------------------------------------|----------|------------|-------------|-------|----|-----------|-----------|-------|-------|------|-------|-------|-------|-----|----|------|----|----------|------|---|---|
|                                     | 2進<br>*1 | 規格2進<br>*1 | 外部10進<br>*2 | 内部10進 | 英字 | 英数字<br>*3 | 日本語<br>*3 | 英数字編集 | 日本語編集 | 数字編集 | 外部    | 単精度内部 | 倍精度内部 | 内部  | 外部 |      |    |          |      |   |   |
| VT_UI1                              | △        |            |             |       | -  | □         | -         | □     | -     | □    | -     | □     | -     | □   |    | -    | △  | -        | -    |   |   |
| VT_I2                               |          |            |             |       | -  |           | -         |       | -     |      | -     |       | -     |     |    | -    |    | -        |      |   |   |
| VT_I4                               |          |            |             |       | -  |           | -         |       | -     |      | -     |       | -     |     |    | -    |    | -        |      |   |   |
| VT_CY                               |          |            |             |       | -  |           | -         |       | -     |      | -     |       | -     |     |    | -    |    | -        |      |   |   |
| VT_DECIMAL                          |          |            |             |       | -  |           | -         |       | -     |      | -     |       | -     |     |    | -    |    | -        |      |   |   |
| VT_R4                               | □        |            |             |       | -  | -         | -         | -     | -     | -    | ○     | -     | -     | -   | -  | -    | -  | -        | -    |   |   |
| VT_R8                               |          |            |             |       | -  | -         | -         | -     | -     | □    | ○     | -     | -     | -   | -  |      |    |          |      |   |   |
| VT_BSTR                             |          |            |             |       | *4 |           |           |       |       |      | -     | -     | -     | -   | -  | -    | -  | -        | -    |   |   |
| VT_DATE                             |          |            |             |       | -  | □         | -         | □     | -     | □    | -     | □     | □     | -   | -  | -    | -  | -        | -    | - | - |
| VT_BOOL                             |          |            |             |       | -  | □         | -         | -     | -     | -    | -     | -     | -     | -   | □  | □    | *5 | *5       | -    | - | - |
| VT_DISPATCH                         | -        | -          | -           | -     | -  | -         | -         | -     | -     | -    | -     | -     | -     | -   | -  | -    | -  | *6       | -    |   |   |
| VT_UNKNOWN                          | -        | -          | -           | -     | -  | -         | -         | -     | -     | -    | -     | -     | -     | -   | -  | -    | -  | □        | -    |   |   |
| VT_ERROR                            | □        |            |             |       | -  | □         | -         | □     | -     | □    | -     | -     | -     | -   | -  | -    | -  | -        | -    |   |   |
| VT_NULL                             | -        | -          | -           | -     | □  |           | -         | □     | -     | -    | -     | -     | -     | -   | -  | -    | -  | -        | -    |   |   |
| VT_ARRAY                            | -        | -          | -           | -     | -  | -         | -         | -     | -     | -    | -     | -     | -     | -   | -  | -    | -  | *6       | -    |   |   |
| VT_VARIANT                          | □        |            |             |       |    |           |           |       |       |      | -     | □     |       |     | -  | □    | -  |          |      |   |   |

- :  
変換可能です。

- :  
COMのデータからCOBOLのデータへの変換は可能です。  
COBOLのデータからCOMのデータへの変換は、アーリバインドでCOM連携している場合だけ可能です。  
ただし、変換可能とされている場合でも、変換できるかどうかはCOMのデータ変換関数の規則によるため、変換できない場合があります。  
例) 日付や数字を表すVT\_BSTR型から数字データへの変換は可能です。
- △:  
全体桁数／小数点桁数に応じて、以下の規則に従って変換します。
  - － 小数点桁数が0 のとき、全体桁数に応じて変換します。
    - 全体桁数が1 ～4 のとき、VT\_I2
    - 全体桁数が5 ～9 のとき、VT\_I4
  - － 全体桁数が18で、かつ小数点桁数が4 のとき、VT\_CY
  - － 上記以外の場合、VT\_DECIMAL
- \*1:  
int型2進整数データ(BINARY-CHAR, BINARY-SHORT, BINARY-LONG, BINARY-DOUBLE)は変換エラーとなります。  
また、翻訳オプションBINARY(BYTE)が指定されているとき、PICTURE句に指定した桁数が1～2、5～7または10～16桁のデータは変換エラーとなります。
- \*2:  
SIGN句にTRAILING以外を指定している場合、変換エラーとなります。
- \*3:  
ANY LENGTH句は指定できません。指定した場合、変換エラーとなります。
- \*4:  
変換されますが、文字列の右側の空白は削除されます。(COBOL→COM)  
また、受取り側の領域が返却された文字列の長さよりも長い場合、文字列は左詰めにされ、余った領域には空白が埋められます。  
受取り側の領域が返却された文字列よりも短い場合、返却値の文字列を左詰めし、入りきらなかった文字列は捨てられます。  
(COM→COBOL)
- \*5:  
1桁だけ変換可能です。
- \*6:  
COBOLのデータ型が特殊クラスのオブジェクト参照データである場合、変換可能です。  
COBOLのデータ型が特殊クラスのオブジェクト参照データ以外で、かつ、NULLオブジェクトである場合、COMのNULLオブジェクト(NULLインタフェース)に変換されます。NULLオブジェクト以外の場合、変換エラーとなります。

以下に、それぞれのVT型の意味について説明します。

表26.8 VT型の意味

| VT型        | COMの型       | 意味                            |
|------------|-------------|-------------------------------|
| VT_UI1     | 1バイト符号なし整数型 | 1バイト符号なし整数                    |
| VT_UI2     | 整数型         | 2バイト整数                        |
| VT_UI4     | 長整数型        | 4バイト整数                        |
| VT_CY      | 通貨型         | 通貨値                           |
| VT_DECIMAL | DECIMAL型    | 12バイトの整数データと小数点位置データで表される小数点数 |
| VT_R4      | 単精度浮動小数点数型  | IEEE形式4バイト実数                  |
| VT_R8      | 倍精度浮動小数点数型  | IEEE形式8バイト実数                  |
| VT_BSTR    | 文字列型        | 長さの情報を持つ文字列                   |
| VT_DATE    | 日付型         | 日付と時刻を表す倍精度数                  |

| VT型            | COMの型      | 意味                                |
|----------------|------------|-----------------------------------|
| VT_BOOL        | ブール型       | ブール値(True:0xFFFF,False:0x0000)    |
| VT_DISPATCH    | オブジェクト型    | COMオブジェクト(IDispatchインタフェースによる受渡し) |
| VT_UNKNOWN     | オブジェクト型    | COMオブジェクト(IUnknownインタフェースによる受渡し)  |
| VT_ERROR       | エラー型       | COMのエラーコード(SCODE)                 |
| VT_EMPTY       | 空値(省略パラメタ) | パラメタの省略                           |
| VT_NULL        | NULL型      | NULL値                             |
| VT_ARRAY(注1)   | 配列型        | 配列                                |
| VT_VARIANT(注2) | バリエーション型   | バリエーション                           |

注1:配列型は、NetCOBOLでは\*COM-ARRAYクラスのオブジェクトとして扱います。\*COM-ARRAYクラスについては、“[26.6.1 \\*COM-ARRAYクラス](#)”を参照してください。

注2:バリエーション型は、バリエーション型データを持つデータが保持しているデータの型として扱います。詳細については、“[26.6.1.5 バリエーション型データの扱い](#)”を参照してください。

注3:バリエーション型は、COM連携において頻繁に現れるデータ型ですが、特定のCOBOLデータがバリエーション型に変換可能か否かは、“[図26.16 COBOLのデータ型とCOMのデータ型の対応](#)”に示すとおり、あくまでもCOMの変換関数の規則によります。従って、バリエーション型データとのパラメタの受け渡しを、アトリビュートにより実現するソース記述に対しては、翻訳時に、念のために注意を促す目的で、Iレベルエラーメッセージを出力しています。

## 26.6 COM連携のためのその他の特殊クラス

### 26.6.1 \*COM-ARRAYクラス

COMの配列データ型は、配列のサイズ、保持する型の情報および配列の実体を持つデータ型です。このデータ型は、COBOLのデータ型に直接マッピングできないため、NetCOBOLでは配列データ型を扱うための特殊クラス\*COM-ARRAY(\*OLE-ARRAYも同義)を提供しています。



\*COM-ARRAYクラスの方法はレイトバインド形式で呼び出してください。

#### 26.6.1.1 \*COM-ARRAYクラスのファクトリメソッド

\*COM-ARRAYクラスの方法には、以下の方法があります。

- NEWメソッド

上記の方法の機能を“[表26.9 \\*COM-ARRAYクラスの方法](#)”に示します。

表26.9 \*COM-ARRAYクラスの方法

| メソッド名 | 引数                                  | 引数の型                                                              | 意味／返却値                                                                                                                                 |
|-------|-------------------------------------|-------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| NEW   | 型指定<br>次元数<br>1次元要素数<br>2次元要素数<br>: | S9(9) COMP-5<br>S9(9) COMP-5<br>S9(9) COMP-5<br>S9(9) COMP-5<br>: | NEWメソッドは*COM-ARRAYオブジェクトを生成して返却します。引数として、その配列が持つ要素の型、次元数、各次元のサイズを指定します。返却されるオブジェクトの要素は初期化されません。また、値が設定されていない要素を取り出した場合の動作については保証されません。 |

要素の型は整数で指定します。要素のデータ型とそれを表す数値の対応については、“[表26.11 データ型を表す数値](#)”を参照してください。

### 26.6.1.2 \*COM-ARRAYクラスのオブジェクトメソッド

\*COM-ARRAYクラスのオブジェクトメソッドには、以下のメソッドがあります。

- GET-DATAメソッド
- SET-DATAメソッド
- GET-TYPEメソッド
- GET-DIMENSIONメソッド
- GET-SIZEメソッド

上記のメソッドの機能を“表26.10 \*COM-ARRAYクラスのオブジェクトメソッド”に示します。

表26.10 \*COM-ARRAYクラスのオブジェクトメソッド

| メソッド名         | 引数                | 引数の型                    | 意味／返却値                                                            |
|---------------|-------------------|-------------------------|-------------------------------------------------------------------|
| GET-DATA      | 1次元添字<br>:        | S9(9) COMP-5<br>:       | 引数で指定された要素を返却します。返却値の型と配列の型が異なる場合は、可能なかぎりデータの型変換を行います。            |
| SET-DATA      | データ<br>1次元添字<br>: | 任意<br>S9(9) COMP-5<br>: | 引数で指定された配列の要素にデータを設定します。指定したデータの型と配列の型が異なる場合は、可能なかぎりデータの型変換を行います。 |
| GET-TYPE      | —                 | —                       | 配列の要素の型を整数で返却します。(注)                                              |
| GET-DIMENSION | —                 | —                       | 配列オブジェクトの次元数を整数S9(9) COMP-5で返却します。                                |
| GET-SIZE      | n                 | S9(9) COMP-5            | n で指定された次元の大きさを整数S9(9) COMP-5で返却します。n が配列の次元数を超えていてはなりません。        |

注: 返却される数値とそれに対応するデータ型については“表26.11 データ型を表す数値”を参照してください。

### 26.6.1.3 \*COM-ARRAYの型指定子

\*COM-ARRAYクラスのメソッドに指定する配列要素のデータ型は、整数で表現されます。“表26.11 データ型を表す数値”にその対応を示します。

表26.11 データ型を表す数値

| COMのデータ型      | 値  | COBOLのデータ型         |
|---------------|----|--------------------|
| 整数型           | 2  | S9(4) COMP-5       |
| 長整数型          | 3  | S9(4) COMP-5       |
| 単精度浮動小数点数型    | 4  | COMP-1             |
| 倍精度浮動小数点数型    | 5  | COMP-2             |
| 通貨型           | 6  | S9(14)V9(4) COMP-5 |
| 日付型           | 7  | COMP-2             |
| 文字列型          | 8  | X または N            |
| オブジェクト(COM) 型 | 9  | *COMオブジェクト         |
| ブール型          | 11 | 1(1) BIT           |
| バリエント型        | 12 | —                  |
| DECIMAL型      | 14 | 上記以外の数字型           |





## 注意

\*COM-ARRAYの型指定に、配列型(X'2000')は指定できません。

### 26.6.1.4 \*COM-ARRAYクラスの使用例

ここでは、\*COM-ARRAYクラスの使用方法を3つの例で説明します。



## 例

#### 同一要素型の配列を作成する例

同一要素型の配列を作成するプログラム例を以下に示します。この例では、要素の型が長整数(4バイト整数)の1次元3要素の配列を扱います。

```

REPOSITORY.
 CLASS COM-ARRAY AS "*COM-ARRAY".
 :
WORKING-STORAGE SECTION.
01 ARRAYOBJ OBJECT REFERENCE COM-ARRAY.
01 LONG-INT-TYPE PIC S9(9) COMP-5 VALUE 3.
01 ARRAY-DIMENSION PIC S9(9) COMP-5 VALUE 1.
01 AXIS-1 PIC S9(9) COMP-5 VALUE 3.
01 ELM-DATA PIC S9(9) COMP-5.
01 ELM-NO PIC S9(9) COMP-5.
:
PROCEDURE DIVISION.
* *COM-ARRAYクラスのオブジェクトを作成する。
* 型: 長整数(4バイト整数)、次元数:1、第1 次元の要素数:3
 INVOKE COM-ARRAY "NEW"
 USING LONG-INT-TYPE ARRAY-DIMENSION AXIS-1
 RETURNING ARRAYOBJ.
* 配列に第1 要素を設定する。
 MOVE 1 TO ELM-DATA.
 MOVE 1 TO ELM-NO.
 INVOKE ARRAYOBJ "SET-DATA" USING ELM-DATA ELM-NO.
* 配列に第2 要素を設定する。
 MOVE 10 TO ELM-DATA.
 MOVE 2 TO ELM-NO.
 INVOKE ARRAYOBJ "SET-DATA" USING ELM-DATA ELM-NO.
* 配列に第3 要素を設定する。
 MOVE 100 TO ELM-DATA.
 MOVE 3 TO ELM-NO.
 INVOKE ARRAYOBJ "SET-DATA" USING ELM-DATA ELM-NO.
 :

```

作成された配列は以下のようなCOMの配列になります。長整数型の配列でそれぞれ1、10、100の3つの要素を持ちます。

#### 長整数型配列

|     | 要素(1) | 要素(2) | 要素(3) |
|-----|-------|-------|-------|
| 1次元 | 1     | 10    | 100   |

このように作成した\*COM-ARRAYオブジェクトを引数として、配列を要求するCOMサーバに渡します。したがって、\*COM-ARRAYオブジェクトは、COMサーバが要求する構造になるように作成しなければなりません。



## 例

### 異なる要素型の配列を作成する例

\*COM-ARRAYオブジェクトは、要素の型が異なる配列を作成することもできます。要素の型が異なる配列の作成には、要素の型としてバリエント(値:12)を指定します。以下の例では、要素の型がバリエントの1次元3要素の配列を作成しています。

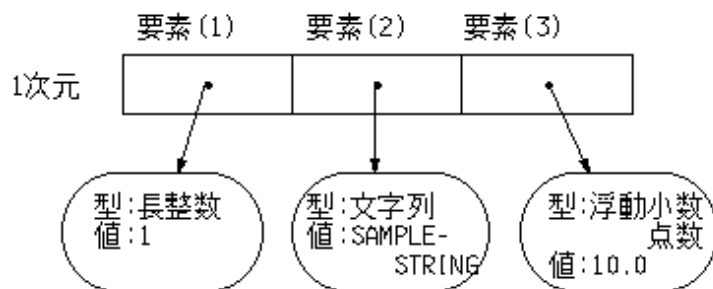
```

REPOSITORY.
 CLASS COM-ARRAY AS "COM-ARRAY".
 :
 WORKING-STORAGE SECTION.
 01 ARRAYOBJ OBJECT REFERENCE COM-ARRAY.
 01 VARIANT-TYPE PIC S9(9) COMP-5 VALUE 12.
 01 ARRAY-DIMENSION PIC S9(9) COMP-5 VALUE 1.
 01 AXIS-1 PIC S9(9) COMP-5 VALUE 3.
 01 LONGINT-DATA PIC S9(9) COMP-5 VALUE 1.
 01 STRING-DATA PIC X(20) VALUE "SAMPLE-STRING".
 01 FLOAT-DATA COMP-1 VALUE 10.0.
 01 ELM-NO PIC S9(9) COMP-5.
 :
 PROCEDURE DIVISION.
 * *COM-ARRAYクラスのオブジェクトを作成する。
 * 型: バリエント、次元数:1、第1 次元の要素数:3
 INVOKE COM-ARRAY "NEW"
 USING VARIANT-TYPE ARRAY-DIMENSION AXIS-1
 RETURNING ARRAYOBJ.
 * 配列に第1 要素を設定する。
 MOVE 1 TO ELM-NO.
 INVOKE ARRAYOBJ "SET-DATA" USING LONGINT-DATA ELM-NO.
 * 配列に第2 要素を設定する。
 MOVE 2 TO ELM-NO.
 INVOKE ARRAYOBJ "SET-DATA" USING STRING-DATA ELM-NO.
 * 配列に第3 要素を設定する。
 MOVE 3 TO ELM-NO.
 INVOKE ARRAYOBJ "SET-DATA" USING FLOAT-DATA ELM-NO.
 :

```

作成された配列は以下のようなCOMの配列になります。

### バリエント型の配列（要素の型が異なる配列）



COMのバリエント型の配列では、要素は型と値の組へのポインタとして保持される構造になっています。



## 例

### COMサーバから配列を受け取るクライアントプログラムの例

COMサーバから配列が返却される場合のクライアントプログラムの例を以下に示します。

クライアントでCOMの配列を受け取る場合、\*COM-ARRAY型の一意名を復帰項目に指定します。COMサーバが配列を返却するとき、NetCOBOLシステムが自動的に\*COM-ARRAYオブジェクトを生成してクライアントに返却するため、クライアントに指定する復帰項目の配列は初期化されている必要はありません。しかし、クライアントからCOMサーバに引数として配列を渡して、COMサーバに要素を書き込んでもらうような場合には、COMサーバが要求するような配列を定義して渡す必要があります。

```
REPOSITORY.
 CLASS COM AS "COM"
 CLASS COM-ARRAY AS "COM-ARRAY".
 :
WORKING-STORAGE SECTION.
01 COMOBJ OBJECT REFERENCE COM.
01 ARRAYOBJ OBJECT REFERENCE COM-ARRAY.
01 APPL PIC X(20) VALUE "SOME APPLICATION".
01 LONGINT-1 PIC S9(9) COMP-5.
01 LONGINT-2 PIC S9(9) COMP-5.
01 LONGINT-3 PIC S9(9) COMP-5.
01 ELM-NO PIC S9(9) COMP-5.
 :
PROCEDURE DIVISION.
* アプリケーションを起動しメソッドを実行し配列を受けとる。
 INVOKE COM "CREATE-OBJECT" USING APPL RETURNING COMOBJ.
 INVOKE COMOBJ "SOME-METHOD" USING ... RETURNING ARRAYOBJ.
* 配列の第1 要素を取り出す。
 MOVE 1 TO ELM-NO.
 INVOKE ARRAYOBJ "GET-DATA" USING ELM-NO RETURNING LONGINT-1.
* 配列の第2 要素を取り出す。
 MOVE 2 TO ELM-NO.
 INVOKE ARRAYOBJ "GET-DATA" USING ELM-NO RETURNING LONGINT-2.
* 配列の第3 要素を取り出す。
 MOVE 3 TO ELM-NO.
 INVOKE ARRAYOBJ "GET-DATA" USING ELM-NO RETURNING LONGINT-3.
 :
```

このプログラムでは、以下に示すような配列が返却されることを仮定しています。

長整数型配列

|     | 要素(1) | 要素(2) | 要素(3) |
|-----|-------|-------|-------|
| 1次元 | 1     | 10    | 100   |

.....

26.6.1.5 バリエーション型データの扱い

NetCOBOLでは、COMのバリエーション型(VT\_VARIANT)データをそのまま扱うことができません。バリエーション型データが保持しているデータを直接扱います。

以下にCOMクライアントでバリエーション型データを扱う場合について説明します。

COMサーバから返却されたバリエーション型データの扱い

クライアントに指定した復帰項目のデータ型に可能なかぎり変換します。

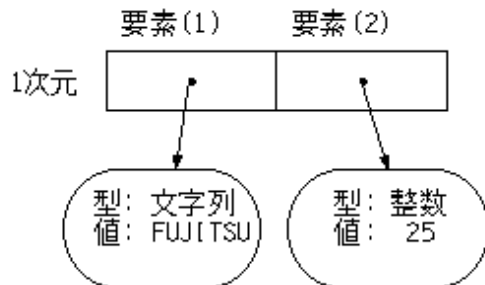
COMサーバに引数として渡すバリエーション型データの扱い

アローバインドの場合、引数に指定したデータを可能なかぎりバリエーション型に変換してCOMサーバに渡します。レイトバインドの場合、COMサーバにバリエーション型データを渡すことはできません。ただし、ほとんどのCOMサーバは渡されたデータを必要な型に変換して処理を行います。したがって、引数に適切な値を指定すれば、問題なく動作します。

## バリエント型データを要素とする配列のプログラム操作例

COMサーバが、以下に示すバリエント型データを返却する場合、それぞれの要素を別々に取り出すクライアントプログラムの例を以下に示します。

### 配列オブジェクト（バリエント型）



返却されるデータは、次元数1、要素数2のバリエント型配列で、第1要素が文字列、第2要素が整数です。

```
REPOSITORY.
 CLASS COM AS "*COM"
 CLASS COM-ARRAY AS "*COM-ARRAY".
 :
WORKING-STORAGE SECTION.
01 COMOBJ OBJECT REFERENCE COM.
01 ARRAYOBJ OBJECT REFERENCE COM-ARRAY.
01 PERSONAL-DATA.
 02 NAME PIC N(20).
 02 AGE PIC S9(4) COMP-5.
01 I PIC S9(9) COMP-5.
01 APPLI PIC X(20) VALUE "SOME. APPLICATION".

PROCEDURE DIVISION.
* アプリケーションを起動してメソッドを実行し、配列を受け取る。
 INVOKE COM "CREATE-OBJECT" USING APPLI RETURNING COMOBJ.
 INVOKE COMOBJ "SOME-METHOD" USING ... RETURNING ARRAYOBJ.
* 配列の第1 要素を処理する。
 MOVE 1 TO I.
 INVOKE ARRAYOBJ "GET-DATA"
 USING I
 RETURNING NAME OF PERSONAL-DATA.
* 配列の第2 要素を処理する。
 ADD 1 TO I.
 INVOKE ARRAYOBJ "GET-DATA"
 USING I
 RETURNING AGE OF PERSONAL-DATA.
 :
```

### 26.6.1.6 \*COM-ARRAYクラスで扱うデータ型

\*COM-ARRAYクラスのSET-DATAメソッドは、第1パラメタに指定されたCOBOLデータを、配列のVT型に合わせて変換し、配列に指定します。また、GET-DATAメソッドは配列のデータをVT型からCOBOLのデータに変換して返却します。

この変換は、\*COMクラスで扱うデータの型変換と同じ規則で行われます。[参照]“[26.5 COM連携で扱うデータ型の対応](#)”

## 26.6.2 \*COM-EXCEPTIONクラス

\*COM-EXCEPTIONクラスは、COM機能で例外処理を行うために利用する例外オブジェクトです。

COMクライアントは、\*COM-EXCEPTION(\*OLE-EXCEPTIONも同義)クラスのオブジェクトを例外オブジェクトとすることで、COMサーバからの例外を受け取ることができます。

COMサーバでは、\*COM-EXCEPTIONクラスのオブジェクトに対して例外を発生させることで、COMクライアントに例外を通知することができます。

\*COM-EXCEPTIONクラスの使い方については、“[26.2.8 COM機能の例外処理](#)”および“[26.3.6 例外の通知](#)”を参照してください。

### 26.6.2.1 \*COM-EXCEPTIONクラスのメソッド

\*COM-EXCEPTIONクラスのメソッドについて説明します。

\*COM-EXCEPTIONクラスには、例外が発生した原因を知るために、以下のオブジェクトメソッドがあります。

- GET-ERROR-TYPEメソッド
- GET-WCODEメソッド
- GET-SCODEメソッド
- GET-SCODE-TEXTメソッド
- GET-DESCRIPTIONメソッド
- GET-DESCRIPTION-LENGTHメソッド
- GET-SOURCEメソッド
- GET-SOURCE-LENGTHメソッド
- GET-HELPPFILEメソッド
- GET-HELPPFILE-LENGTHメソッド
- GET-HELPCONTEXTメソッド
- SET-WCODEメソッド
- SET-DESCRIPTIONメソッド
- SET-SOURCEメソッド
- SET-HELPPFILEメソッド
- SET-HELPCONTEXTメソッド

\*COM-EXCEPTIONクラスのメソッドが返却するデータ型および意味を“[表26.12 \\*COM-EXCEPTIONクラスのメソッド](#)”に示します。

表26.12 \*COM-EXCEPTIONクラスのメソッド

| メソッド名          | データ型(注1)                    | 意味                                                                                                                                                                                                                                                                                                                    |
|----------------|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GET-ERROR-TYPE | X(1)                        | エラーがCOM のシステムで検出されたか、COM サーバで検出されたかを区別するコードを返却します。 <ul style="list-style-type: none"><li>• 0 : COM サーバがエラーを検出しました。他のメソッドを実行することにより、COM サーバから通知されたエラーに関する詳細な情報を参照することができます。</li><li>• 1 : COM のシステムがエラーを検出しました。GET-SCODE メソッド、GET-SCODE-TEXTメソッドを実行することにより、エラーの内容を参照することができます。それ以外のメソッドを実行すると0 または空白が返却されます。</li></ul> |
| GET-WCODE      | X(2)<br>または<br>S9(4) COMP-5 | COMサーバから通知された2バイトのエラーコードを返却します。<br>このコードはCOM サーバが独自に定義しますので、エラーコードの内容と対処方法については、COM サーバのマニュアルまたはオンラインヘルプを参照してください。                                                                                                                                                                                                    |

| メソッド名                  | データ型(注1)                    | 意味                                                                                                                                                               |
|------------------------|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GET-SCODE              | X(4)<br>またはS9(9)<br>COMP-5  | COM のシステムまたはCOM サーバから通知された4バイトのエラーコードを返却します。(注2)                                                                                                                 |
| GET-SCODE-TEXT         | X(n)<br>または<br>N(n)         | COM のシステムまたはCOM サーバから通知された4バイトのエラーコード(GET-SCODEメソッドが返却する値)を、システム関数によってエラーメッセージ文字列に変換したものを返却します。<br>システム関数による文字列の変換に失敗した場合や、エラーコードがCOM サーバが独自に設定したものの場合は空白を返却します。 |
| GET-DESCRIPTION        | X(n)<br>または<br>N(n)         | COM サーバから通知された、発生したエラーに関する簡単な説明を返却します。<br>COM サーバがエラーに関する説明を通知しなかった場合は、空白を返却します。                                                                                 |
| GET-DESCRIPTION-LENGTH | S9(9) COMP-5                | GET-DESCRIPTION メソッドで返却する文字列の長さを、バイト数で返却します。<br>COM サーバがエラーに関する説明を通知しなかった場合は、0 を返却します。                                                                           |
| GET-SOURCE             | X(n)<br>または<br>N(n)         | COM サーバから通知された、エラーを検出したCOM サーバ名を返却します。<br>COM サーバがCOM サーバ名を通知しなかった場合は、空白を返却します。<br>COM サーバ名は一般にプログラムIDの形式で通知されます。(注3)                                            |
| GET-SOURCE-LENGTH      | S9(9) COMP-5                | GET-SOURCEメソッドで返却する文字列の長さを、バイト数で返却します。<br>COM サーバがCOM サーバ名を通知しなかった場合は、0 を返却します。                                                                                  |
| GET-HELPPFILE          | X(n)<br>または<br>N(n)         | COM サーバから通知された、エラーに関する詳細な情報が入ったヘルプファイル名を返却します。<br>COM サーバがヘルプファイル名を通知しなかった場合は、空白を返却します。                                                                          |
| GET-HELPPFILE-LENGTH   | S9(9) COMP-5                | GET-HELPPFILEメソッドが返却する文字列の長さを、バイト数で返却します。<br>COM サーバがヘルプファイル名を通知しなかった場合は、0 を返却します。                                                                               |
| GET-HELPCONTEXT        | X(4)<br>または<br>S9(9) COMP-5 | COM サーバから通知された、エラーに関する詳細な情報の、ヘルプファイル内でのコンテキストIDを返却します。<br>COM サーバがコンテキストIDを通知しなかった場合は、0 (0x00000000)を返却します。                                                      |
| SET-WCODE              | X(2)<br>または<br>S9(4) COMP-5 | COM クライアントに通知する2バイトのラーコードを指定します。<br>このコードはCOM サーバが独自に定義するものですから、エラーコードの内容と対処方法については、COM サーバの利用者に公開してください。                                                        |
| SET-DESCRIPTION        | X(n)<br>または<br>N(n)         | COM クライアントに通知する、発生したラーに関する簡単な説明を指定します。既定値は空文字列です。                                                                                                                |
| SET-SOURCE             | X(n)<br>または<br>N(n)         | COM クライアントに通知する、エラーを検出したCOM サーバ名を指定します。既定値は空文字列です。                                                                                                               |
| SET-HELPPFILE          | X(n)<br>または<br>N(n)         | COM クライアントに通知する、エラーに関する詳細な情報が入ったヘルプファイル名を指定します。既定値は空文字列です。                                                                                                       |
| SET-HELPCONTEXT        | X(4)<br>または<br>S9(9) COMP-5 | COM クライアントに通知する、ヘルプファイル内でのコンテキストIDを指定します。既定値は、0 (0x00000000)です。                                                                                                  |

注1: 'GET-'で始まるメソッドの場合には、返却されるデータの型を、'SET-'で始まるメソッドの場合には、引数に指定するデータの型を表します。

注2: エラーコードの意味およびプログラムの対処については、“メッセージ説明書”の“付録B SCODE形式のエラーコードの説明”を参照してください。

注3: プログラムIDの形式については、“26.2.4.1 \*COMレイトバインドクラスのファクトリメソッド”を参照してください。

## 注意

- エラーコードやヘルプコンテキストIDが返却されるメソッドの復帰項目には、英数字項目またはCOMP-5形式の数字項目が指定できます。復帰項目の内容を10進形式で参照したい場合には数字項目を、16進形式で参照したい場合には英数字項目を指定してください。
- GET-SCODE-TEXTメソッドが返却するメッセージ文字列は、システムによって異なる場合があります。アプリケーション中で発生したエラーの現象を判定したい場合は、GET-SCODEが返却するコードを使用してください。
- COMサーバプログラムの中でCOBOLのエラーを検出した場合、COBOLランタイムシステムは、以下を返却します。

```
GET-ERROR-TYPE : '0'
GET-WCODE : 1001
GET-DESCRIPTION : “COM例外オブジェクト以外の例外が発生しました。
 エラーを検出したクラス名”
```

GET-WCODEメソッド、GET-ERROR-TYPEメソッド、GET-DESCRIPTIONメソッドおよびGET-DESCRIPTION-LENGTHメソッドで取得できる値以外は、意味を持ちません。また、WCODEの値'1001'は予約されているため、SET-WCODEメソッドに指定しないでください。

## 26.6.3 特殊クラスの注意事項

ここでは、特殊クラス(\*COM、\*COM-ARRAY、\*COM-EXCEPTION)を利用する場合の注意事項について説明します。

- 特殊クラスのメソッド呼出しの引数はすべて一意名で指定します。
- 特殊クラスのファクトリオブジェクトを取り出すことはできません。以下の書き方はエラーとなります。

```
SET 一意名 TO 特殊クラス.
```

- 特殊クラスのオブジェクトは、選択指定のないオブジェクト参照項目として扱うことはできません。
- 特殊クラスは継承できません。
- \*COMおよび\*COM-ARRAY型のオブジェクトデータは、データ定義だけではNULLオブジェクトの状態になっています。このため、この状態で返却用の引数としてCOMサーバに渡すと、VT\_EMPTYが渡されることになり、返却値の設定でエラーになります。返却用の引数として渡すときには、オブジェクトに値を設定してから渡してください。

返却用の引数を使用したプログラム例を、以下に示します。

```
REPOSITORY.
 CLASS COM AS “*COM”
 CLASS COM-ARRAY AS “*COM-ARRAY”.
:
WORKING-STORAGE SECTION.
01 COMOBJ OBJECT REFERENCE COM.
01 RETOBJ OBJECT REFERENCE COM.
01 ARRAYOBJ OBJECT REFERENCE COM-ARRAY.
01 TYPE-INT PIC S9(9) COMP-5 VALUE 2.
01 ARY-DIM PIC S9(9) COMP-5 VALUE 1.
01 ARY-SIZE PIC S9(9) COMP-5 VALUE 1.
01 APPLI PIC X(20) VALUE “SOME APPLICATION”.
:
PROCEDURE DIVISION.
 INVOKE COM “CREATE-OBJECT” USING APPLI RETURNING COMOBJ. ... [1]
```

```

SET RETOBJ TO COMOBJ. ... [2]
INVOKE COM-ARRAY "NEW" USING TYPE-INT ARY-DIM ARY-SIZE ... [3]
 RETURNING ARRAYOBJ.
INVOKE COMOBJ "SOME-METHOD" USING RETOBJ ARRAYOBJ. ... [4]
:

```

[1] SOME.APPLICATIONオブジェクトを特殊クラス\*COMのオブジェクトとして作成します。

[2] [1]で作成したCOMOBJを代入することで返却用のRETOBJを作成します。

[3] 返却用の\*COM-ARRAYのオブジェクトARRAYOBJを、要素型が整数型、次元数が1、要素数が1として作成します。

[4] [2]、[3]で作成した特殊クラスのオブジェクトを返却用引数として指定し、メソッドを呼び出します。このとき、引数に指定したオブジェクトの値はサーバによって置き換えられます。

- \*COMおよび\*COM-ARRAYクラスのオブジェクトの使用が終了したら、オブジェクトを解放することをおすすめします。特に、マルチスレッドモードで動作している場合には、COBOLのプロセスが終了してもオブジェクトが自動的に解放されない場合があります。詳細については、“[24.10.1 COM連携機能](#)”を参照してください。
- COMの例外オブジェクト\*COM-EXCEPTIONを使用するにあたって、エラー種別の判定には、GET-SCODE-TEXTメソッドの返却値ではなくGET-SCODEメソッドの返却値を使用してください。GET-SCODE-TEXTメソッドの返却値はOS、システムの地域設定(ロケール)などによって異なる場合があります。

## 26.7 COMサーバおよびCOMクライアントのデバッグ

ここでは、COMサーバおよびCOMクライアントのデバッグ方法について説明します。

問題の発生条件を限定するため、COM機能を使わないプログラムへの変更が可能な場合は、通常のCOBOLプログラムに修正してデバッグすることをおすすめします。プログラム変更ができない以下のような場合は、“[26.7.2 COMクライアントのデバッグ方法](#)”、“[26.7.3 COMサーバのデバッグ方法](#)”に示す方法でデバッグを行ってください。

- COM機能を使わないプログラムへの変更が困難な場合
- COMサーバとして動作させているときにしか問題が発生しない場合
- MTS配下で実行させなければならない場合

### 26.7.1 通常のCOBOLプログラムとしてデバッグする場合の変更点について

COMサーバは、COBOLのクラス定義から作成します。このため、COBOLで作成したCOMサーバおよびCOMクライアントのプログラムは、COM機能を使わずに、通常のCOBOLプログラムとしてコンパイルおよびリンクを行って実行させることができます。

COMサーバおよびCOMクライアントとして作成したプログラムを、COM機能を使わずにコンパイルおよびリンクする場合には、次の点に注意してください。

#### サーバプログラム

サーバプログラム側では特に注意することはありません。クラス定義がそのまま使えます。

ただし、MTS配下で動作させるプログラムを通常のCOBOLプログラムとして利用する場合、オブジェクトコンテキストオブジェクトが取得できません。このため、オブジェクトコンテキストオブジェクトを使ってCreateInstanceメソッドを呼び出している箇所は、一般クラスのNEWメソッドを呼び出すように書き換えてください。また、そのほかのオブジェクトコンテキストオブジェクトを使ってメソッドを呼び出している箇所はコメントにしてください。

#### クライアントプログラム

##### プログラムの修正

クライアントプログラムを一部修正する必要があります。以下のクライアントプログラムを例にして修正方法を説明します。

##### レイトバインドの場合

```

REPOSITORY.
 CLASS DBACCESS1 AS "COM". ... [1]
:

```



```

01 COMOBJ OBJECT REFERENCE DBACCESS1. ... [2]
01 PROGID PIC X(16) VALUE "SAMPLE.DBACCESS".
01 PROP PIC 9(10).
:
 INVOKE DBACCESS1 "CREATE-OBJECT" USING PROGID ... [3]
 RETURNING COMOBJ.
:
 INVOKE COMOBJ "GET-PROP" RETURNING PROP. ... [4]

```

#### アーライバインドの場合

```

REPOSITORY.
 CLASS DBACCESS1 AS "*COM:SAMPLE:DBACCESS". ... [5]
:
01 COMOBJ OBJECT REFERENCE DBACCESS1. ... [6]
01 PROP PIC 9(10).
:
 INVOKE DBACCESS1 "NEW" RETURNING COMOBJ. ... [7]
:
 MOVE PROP OF COMOBJ TO PROP. ... [8]

```

#### REPOSITORY段落の変更

REPOSITORY段落の特殊クラスの定義を、COMサーバからCOBOLのクラスを参照するように変更します。つまり、[1]および[5]を以下のように書き換えます。

```

REPOSITORY.
 CLASS DBACCESS1.

```

#### サーバ参照変数定義の変更

COMサーバを扱うデータの定義を、\*COMクラスからDBACCESS1クラスに変更します。[2]および[6]は、DBACCESS1クラスのデータとして定義されているため、変更する必要はありません。

#### 引数型のチェック

COMサーバのメソッド呼出しでは、引数の型をCOMが可能な限り調整します。しかし、COBOLのクラスのメソッド呼出し時は、完全に一致していなければならないので、必要があれば変更します。

#### オブジェクトの起動部分の変更

レイバインドの場合、COMサーバの起動には\*COMのCREATE-OBJECTメソッドを使いますが、COBOLのクラスのオブジェクトの生成には、NEWメソッドを使うため、[3]を変更します。アーライバインドの場合は、NEWメソッドでサーバを起動するため、[7]を変更する必要はありません。

#### プロパティの参照/設定

GET-プロパティ名/SET-プロパティ名をメソッド名として指定したINVOKE文でプロパティの参照/設定を行っている箇所は、オブジェクトプロパティによる参照/設定に変更します。

### 翻訳時の注意

クライアントプログラムを翻訳する場合、COMサーバクラスのリポジトリファイルを指定する必要があります。

### リンク時の注意

クライアントプログラムのリンクでは、COMサーバの目的プログラム(.OBJ)をリンクするか、もしくは、COMサーバとして作成したDLLのインポートライブラリをリンクします。

## 26.7.2 COMクライアントのデバッグ方法

COBOLで作成したCOMクライアントのデバッグを開始する場合には特別な注意はありません。通常のデバッグと同じように開始できます。



## 注意

デバッガを使用してデバッグを行う場合は、アプリケーションの動作時間が通常の場合よりも長くなります。そのため、COMサーバがタイムアウトしないようにする必要があります。たとえば、MTSの場合にはトランザクションタイムアウトを無効にします。

### 26.7.3 COMサーバのデバッグ方法

COBOLで作成したCOMサーバをデバッグするには、デバッグしたいプログラムからデバッガを起動する方法を使用します。



## 注意

MTSの場合には、COMサーバで発生するアプリケーションエラーは、MTSが検知して処理するため、アプリケーションエラーに対するジャストインタイムデバッグ機能とランタイムシステムの診断機能は、機能しません。

#### 26.7.3.1 デバッガを起動する

デバッガを使用してデバッグを開始するには、以下の手順で行います。

1. COMサーバ(COBOLプログラム)を翻訳・リンクします。

デバッグを行うための翻訳オプションおよびリンクオプションを指定して、COMサーバを作成します。デバッグを行うための翻訳およびリンクの方法については“[20.2 デバッグの準備](#)”を参照してください。

2. 実行環境情報を設定します。

デバッグしたいプログラムからデバッガを起動するために、あらかじめ次の環境変数情報をシステムの環境変数または実行用の初期化ファイル(COBOL85.CBR)に設定しておく必要があります。

```
@CBR_ATTACH_TOOL=TEST [起動パラメタ]
```

アプリケーションの実行時にデバッガを起動する(TEST)を指定します。“TEST”に続けてデバッガ起動パラメタを指定できます。起動パラメタについては、“[表20.7 起動パラメタ](#)”を参照してください。

3. デバッグするCOMサーバが実行されるコンピュータにログオンします。
4. COMクライアントを起動し、COMサーバを呼び出す処理を実行します。
5. COMサーバが最初に呼び出されたとき、デバッガが自動的に起動されます。デバッガが起動されたら、[デバッグを開始する]ダイアログから、デバッグ情報ファイル格納フォルダと必要な情報を指定してデバッグを開始します。

#### 26.7.3.2 デバッグを行う

デバッグ操作は、デバッガを使用した通常のデバッグと同じです。デバッガの使用方法については、“[第20章 対話型デバッガの使い方](#)”を参照してください。



## 注意

- COMクライアントに制御が戻った後は、COMサーバは次のリクエストを待っているためプログラム実行中になっています。中断点の設定などのデバッグ操作を行いたい場合は、デバッガのメニューから[実行中断]コマンドを選択し、プログラムの実行を中断させてください。
- デバッガを使用してデバッグを行う場合は、アプリケーションの動作時間が通常の場合よりも長くなります。そのため、COMクライアントがデバッグ中にタイムアウトしないようにする必要があります。たとえば、COMクライアントもデバッグし、COMクライアントがサーバの処理を呼び出した時に、デバッガのメニューから[実行中断]コマンドを選択し、COMクライアントの実行を中断させます。この操作で、COMクライアントのタイムアウトを防ぐことができます。

## 26.7.4 COMクライアントとCOMサーバを通したデバッグ方法

COMクライアントとCOMサーバの両方を通したデバッグを行う場合には、COMクライアントのデバッグ方法、COMサーバのデバッグ方法で説明した方法を組み合わせることでデバッグできます。



### 注意

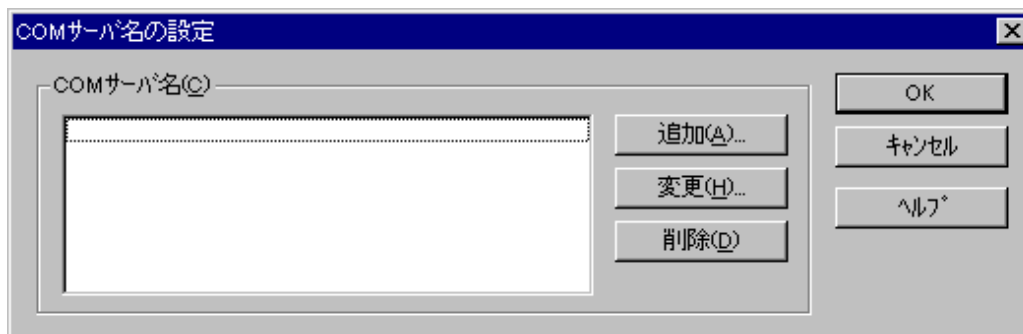
COMクライアント側のデバッガとCOMサーバ側のデバッガは連携しません。そのため、COMクライアント側のCOMサーバを呼び出す命令をステップインで実行してもCOMサーバ側のデバッガは中断しません。COMサーバ側で中断するためにはあらかじめ適切な位置に中断点を設定しておく必要があります

## 26.8 開発環境

### 26.8.1 COMサーバ名の設定

アーライバインドでCOBOLソースプログラムを翻訳する場合、COMサーバ名と型ライブラリファイル名を関連付ける必要があります。COMサーバ名の設定は[翻訳オプション]ダイアログの[COMサーバ名]ボタンをクリックしたときに表示される、[COMサーバ名の設定]ダイアログで行います。[参照]“3.5 [翻訳オプション]ダイアログ”  
[COMサーバ名の設定]ダイアログから、以下の操作を行うことができます。

- COMサーバ名の追加
- COMサーバの型ライブラリファイル名の変更
- COMサーバ名の取消し



#### リストボックス

##### COMサーバ名

すでに設定されているCOMサーバ名が表示されます。

#### ボタン

##### OK

COMサーバ名の設定が終了したときに選択します。

##### キャンセル

[COMサーバ名の設定]ダイアログを開く前の状態に戻し、ダイアログを閉じます。

##### ヘルプ

ヘルプウィンドウを開くときに選択します。

##### 追加

COMサーバ名の追加を行うときに選択します。

##### 変更

リストボックスの型ライブラリファイル名を変更するときに選択します。

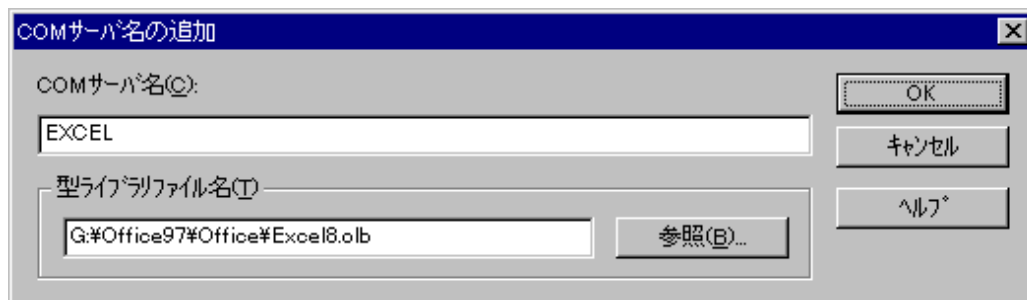
## 削除

リストボックスからCOMサーバ名を削除するときに選択します。

## COMサーバ名の追加

COMサーバ名のリストボックスに表示されていないCOMサーバ名を指定する場合、以下の操作を行います。

1. [COMサーバ名の設定]ダイアログの[追加]ボタンをクリックします。  
→ [COMサーバ名の追加]ダイアログが表示されます。
2. COBOLソースプログラムに記述したCOMサーバ名と型ライブラリファイル名を指定します。  
型ライブラリファイルは、一般に、そのCOMサーバ機能を提供しているTLB/DLLファイルあるいはOLBファイルなどで提供されます。どのような形で提供されているかは、利用するCOMサーバの説明書あるいはヘルプなどで確認してください。NetCOBOLで開発したCOMサーバの場合、COMサーバ機能を提供しているDLLファイルに組み込まれます。



3. [OK]ボタンをクリックします。  
指定した内容を取り消したい場合は、[キャンセル]ボタンをクリックします。→ [COMサーバ名の追加]ダイアログが閉じます。[OK]ボタンをクリックした場合、[COMサーバ名の設定]ダイアログのCOMサーバ名のリストボックスに、指定したCOMサーバ名と型ライブラリファイル名が表示されます。



## 注意

[OK]ボタンをクリックしないでダイアログを閉じた場合、指定は無効となります。

## COMサーバの型ライブラリファイル名の変更

COMサーバ名に対応付ける型ライブラリファイル名を変更する場合、以下の操作を行います。

1. COMサーバ名のリストボックスから、変更したい型ライブラリファイル名を選択し、[変更]ボタンをクリックします。  
→ [型ライブラリファイル名の変更]ダイアログが表示されます。
2. 内容を変更します。変更が終わったら、[OK]ボタンをクリックします。  
→ [型ライブラリファイル名の変更]ダイアログが閉じます。
3. [COMサーバ名の設定]ダイアログのCOMサーバ名のリストボックスに、変更した内容が表示されます。

## COMサーバ名の取消し

COMサーバ名のリストボックスに表示されているCOMサーバ名の指定を取り消す場合、以下の操作を行います。

1. COMサーバ名のリストボックスから、取り消したいCOMサーバ名を選択し、[削除]ボタンをクリックします。
2. COMサーバ名のリストボックスから、選択したCOMサーバ名が削除されます。

## COMサーバ名の設定の終了

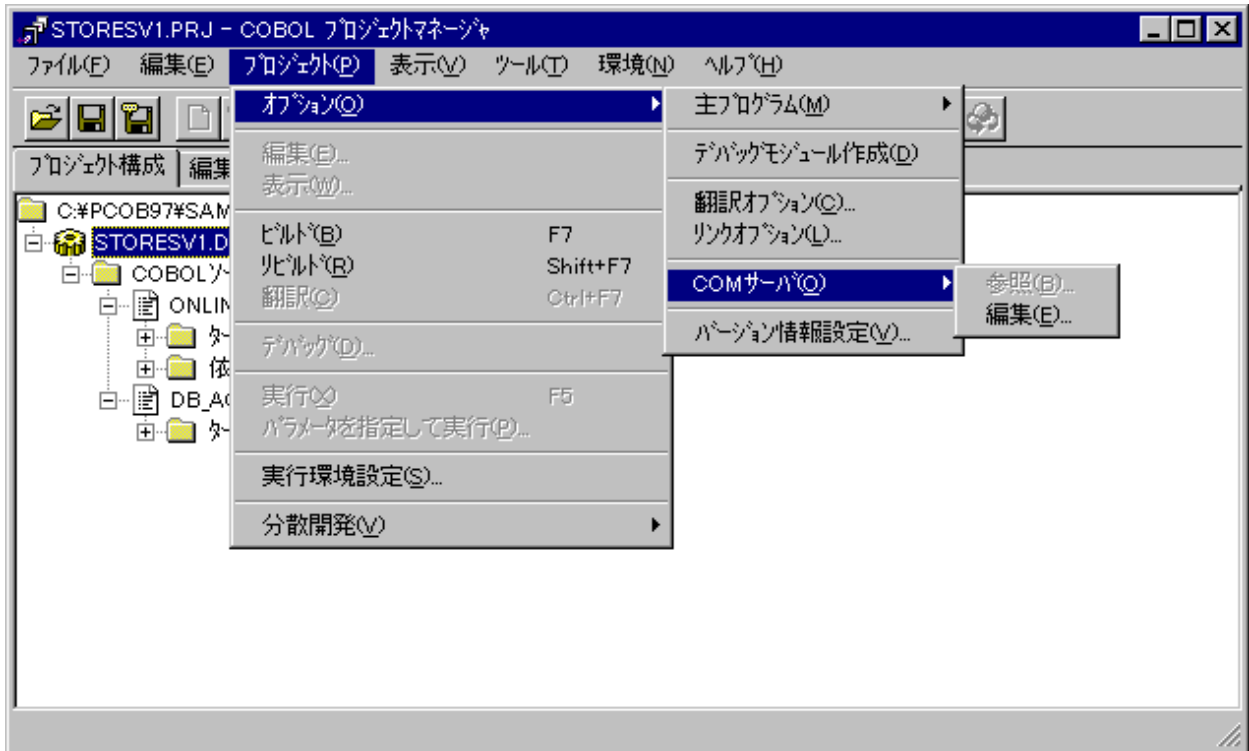
COMサーバ名の設定が終了したら、[COMサーバ名の設定]ダイアログの[OK]ボタンをクリックします。  
→ [翻訳オプション]ダイアログに戻ります。

## 26.8.2 COMサーバ作成機能の使い方

ここでは、プロジェクトマネージャでCOMサーバを作成する方法について説明します。

COMサーバの作成は、プロジェクト管理機能でビルド・リビルドが行える状態になっていることが前提です。“17.2 開発手順”の1.～5.までの処理を完了させてから、以下のCOMサーバの作成処理を行ってください。

COMサーバの作成処理を行うには、プロジェクトマネージャに表示されているDLLを選択した後、[プロジェクト]-[オプション]メニューから“COMサーバ”の“編集”を選択してください。[編集]ダイアログが表示されます。



### 〔参照〕ダイアログ

〔参照〕ダイアログは、COMサーバの登録情報を参照する場合に利用します。〔参照〕ダイアログでは、すべての入力が不可になります。COMサーバを開発しない場合やCOMについての知識がない場合はこちらを選択してください。

### 〔編集〕ダイアログ

〔編集〕ダイアログは、COMサーバアプリケーションを作成する場合に利用します。



#### ボタン

##### OK

COMサーバの設定が終了したときに選択します。このボタンをクリックすると、サーバクラスの選択ページ、サーバ情報ページ、サーバクラス情報ページ、クラスIDページの各情報が保存されます。

##### キャンセル

〔編集〕ダイアログが表示される前の状態に戻して、ダイアログを閉じます。

##### ヘルプ

ヘルプウィンドウを表示するときに選択します。

#### サーバクラスの選択ページ

COMサーバに登録するクラスを設定します。



## リストボックス

### COBOLクラス

プロジェクトに登録されているクラス名が表示されます。プロジェクトツリーの画面でターゲットリポジトリファイル名が正しく登録されていることを確認してください。ターゲットリポジトリファイル名が正しく登録されていないとクラス名が正しく表示されません。

### サーバクラス

サーバクラス名が表示されます。

## ボタン

### 追加

サーバクラスを追加するときに選択します。

### 削除

サーバクラスを削除するときに選択します。

## サーバ情報ページ

COMサーバのバージョン情報および説明文を設定します。

#### エディットボックス

##### メジャー

メジャーバージョンを指定します。スピンボタンで変更することができます。メジャーバージョンは、既存のCOMサーバのインタフェースを変更した場合だけ変更します。それ以外の場合は、マイナバージョンを変更してください。

##### マイナ

マイナバージョンを指定します。スピンボタンで変更することができます。

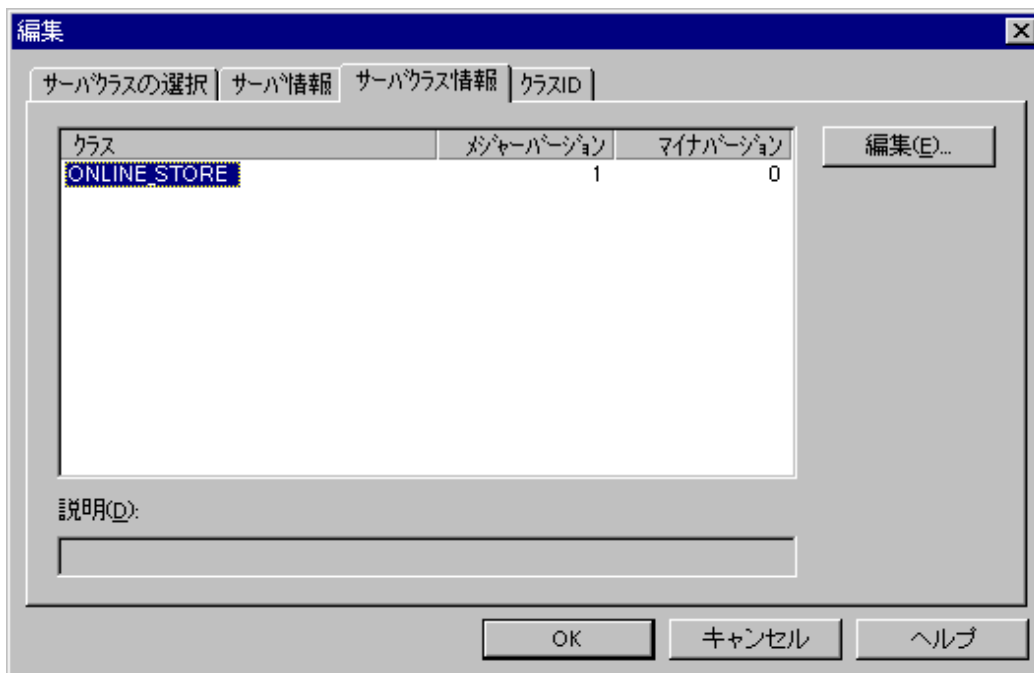
##### 説明

COMサーバに関する説明文を記述します。

#### サーバクラス情報ページ

COMサーバに登録する各クラスに対してバージョン情報を設定します。





#### リストビュー

サーバクラスのクラス名とメジャーバージョンおよびマイナバージョンが表示されます。

#### エディットボックス

##### 説明

クラスに関する説明文が表示されます。入力できません。

#### ボタン

##### 変更

サーバクラス情報を編集するときに選択します。編集を行うクラスをリストビューの中から選択し、[編集]ボタンをクリックすると、[サーバクラス情報の編集]ダイアログが表示されます。



[サーバクラス情報の編集]ダイアログでは、サーバ情報ページと同じように、バージョン情報および説明文の設定を各クラスに対して行うことができます。

#### クラスIDページ

クラスIDページは、既存のCOMサーバのインタフェースに変更がある場合だけ利用します。インタフェースの変更とは、以下の場合を意味します。

- サーバクラスが削除された。サーバクラスが保持するメソッド/プロパティが追加または削除された。
- サーバクラスが保持するメソッド/プロパティの返却値またはパラメタ数に変更された。

- ・ サーバクラスが保持するメソッド/プロパティのパラメタの並びが変更された。
- ・ サーバクラスが保持するメソッド/プロパティの返却値またはパラメタの属性が変更された。

クラスIDは、クラスを識別するために全世界で一意になるように決められたIDなので、上記のインタフェースの変更に当てはまらない場合、クラスIDを変更してはなりません。クラスIDを変更すると、既存のクラスを利用できなくなります。



#### リストビュー

クラス名とそのクラスのクラスIDを表示します。新規に登録したサーバクラス名は、リストビューには表示されません。

#### ボタン

##### 更新

クラスIDを更新する場合に選択します。クラスIDを更新するクラスをリストビューの中から選択し、[更新]ボタンをクリックすると、クラスIDを更新してよいかどうかを尋ねるメッセージが表示されます。クラスIDを更新してよければ、[OK]ボタンをクリックしてください。

## 26.9 COM関連の用語の説明

### プログラムID(ProgID)とクラスID(CLSID)

COMサーバには、全世界で一意になる128ビットのIDが割り当てられます。このIDをクラスID(CLSID)と呼びます。COMサーバを厳密に識別するためには、CLSIDを指定する必要があります。しかし、CLSIDは意味のない数字の並びであるため、何であるかわからないとか間違いやすいという問題があります。

そのため、COMサーバには、CLSIDとは別に意味のある文字列によるIDが割り当てられています。これをプログラムID(ProgID)と呼びます。通常、クライアントプログラムを作成するときは、CLSIDは使わずに、このProgIDを使用してCOMサーバを起動します。たとえば、\*COMクラスを利用してCOMサーバを起動する場合、引数にはCLSIDではなくProgIDを指定します。

ProgIDには以下のようなものがあります。

```
EXCEL. APPLICATION. 8 (Excel98のProgID)
WORD. DOCUMENT. 8 (Word98のProgID)
```

最後の数字はバージョンを表します。互換の問題などで、特定のバージョンのサーバを利用したい場合を除いて、一般にはProgIDはバージョン部分を除いた形式で表されます。この形式をバージョン独立形式と呼びます。

ProgIDはユニークであることが保証されていませんが、衝突しないような工夫をすることで、問題を回避しています。つまり、ProgIDは注意深く決める必要があります。

COBOLのCOMサーバ作成機能は、次のルールでProgIDを割り当てます。

ProgID = “DLL名. クラス名. バージョン番号”

このため、たとえばEXCELとは全く関係がなくてもEXCEL.APPLICATIONというProgIDを持つCOMサーバを作成することが可能になります。しかし、このCOMサーバがインストールされると、EXCELを利用している他のシステムは誤動作をするようになります。そのような事態を避けるために、ProgIDは一意になるように決める必要があります。そのための工夫のひとつとして、DLL名に製品名やベンダ名を含めるなどの方法があります。

## COMサーバのインタフェース

COMのサーバとクライアントは、インタフェースと呼ばれるプログラムを介して連携します。インタフェースにはいくつかの種類があり、それぞれの用途に沿って使い分けれます。COMサーバは通常複数のインタフェースを持っています。COMサーバがどのようなインタフェースを持っているかが分かれば、何に使えるかが分かります。

COBOLのCOMサーバ作成機能では、次のインタフェースを作成します。これらのインタフェースは、COMサーバ作成機能が作成するので、利用者が特に意識する必要はありません。

| インタフェース       | 機能                                |
|---------------|-----------------------------------|
| IUnknown      | 他のインタフェースの取得と、サーバの参照カウントの管理を行います。 |
| IClassFactory | COMサーバを作成します。                     |
| IDispatch     | メソッドの実行、プロパティの参照/設定を行います。         |

COMクライアント機能では、IDispatchインタフェースを操作します。

## 型ライブラリと型情報

COMサーバが公開しているインタフェース、メソッド、プロパティ、引数などの情報を型情報と呼びます。型情報はいくつかをまとめて型ライブラリという形で公開されます。この情報は、クライアントプログラムのアールバインド処理や、オブジェクトの定義情報を表示するツールで使用されます。通常、型ライブラリは、そのCOMサーバを提供しているDLL/EXEファイルまたはそのCOMサーバに関連付けられたTBLファイル/OLBファイル内に格納されています。

NetCOBOLでは、型ライブラリおよび型情報は、COMサーバ作成機能が自動的に作成し、DLLファイルに取り込みます。

## coclassと非coclass

COMサーバにはCOMクライアントから直接起動可能なものと、COMサーバの中でしか起動できないものがあります。たとえば、人事データベースを参照する場合、人物データをオブジェクトで管理するには、検索結果をオブジェクトとして(すなわち、COMサーバとして)返す必要があります。このようなシステムでは、人物データを表すオブジェクトは、データベースを検索した結果としてだけ存在するもので、COMクライアントから直接起動しても意味がありません。一方データベース検索サーバは、COMクライアントから起動できるようになっていなければなりません。このように、COMクライアントから直接起動できるCOMサーバをcoclass、直接起動できないCOMサーバを非coclassと呼びます。COBOLのCOMサーバ作成機能では、coclassだけを作成することができます。

## COMサーバの実装形態

COMサーバは、ダイナミックリンクライブラリ(DLL)で実装する方法と実行可能ファイル(EXE)で実装する方法があります。COBOLのCOMサーバ作成機能では、DLLで実装する方法だけをサポートしています。1つのDLLには、複数のクラスを定義することができます。

## COMとDCOM

COMサーバは、クライアントプログラムが動作するマシン(ローカルマシン)上で動作させることはもちろん、ネットワーク上の別のマシン(サーバマシン)上で動作させることも可能です。後者の連携形態をDCOMと呼びます。COBOLの提供するCOM機能は、両方の連携形態をサポートします。

また、COBOLで作成したCOMサーバは、トランザクション管理を行うMicrosoft Transaction Server(MTS)配下でも動作可能です。MTSについては、“[26.4 DCOMとMTS](#)”を参照してください。

## 第27章 CORBAアプリケーション

プロジェクトマネージャを使用してCORBAアプリケーションを開発する方法について説明します。

### 27.1 CORBAアプリケーションの開発

#### 27.1.1 CORBAアプリケーションとは

CORBAアプリケーションは、CORBAに準拠したアプリケーションのことであり、OSや開発言語、プラットフォームに依存しない相互接続を実現します。

CORBAアプリケーションの中でも、データベース管理システムが提供する、データベースと連携してトランザクション処理を行うアプリケーションのことを、トランザクションアプリケーションと呼びます。

グローバルサーバ上で動作するCORBAアプリケーションの内、Interstageスタイルで開発したアプリケーションのことを、本マニュアルでは、AIMApplicationDirectorアプリケーション(以降、AADアプリケーション)と呼びます。

CORBA アプリケーションを作成、実行するには、ObjectDirector、Interstage、Interstage Application ServerまたはInterstage/ AIMApplicationDirectorが必要です。

ObjectDirectorについては、“ObjectDirector プログラミングガイド”を参照してください。

Interstageについては、“Interstage 分散アプリケーション作成ガイド(CORBAサービス編)”および“Interstage 分散アプリケーション作成ガイド(コンポーネントトランザクションサービス編)”を参照してください。

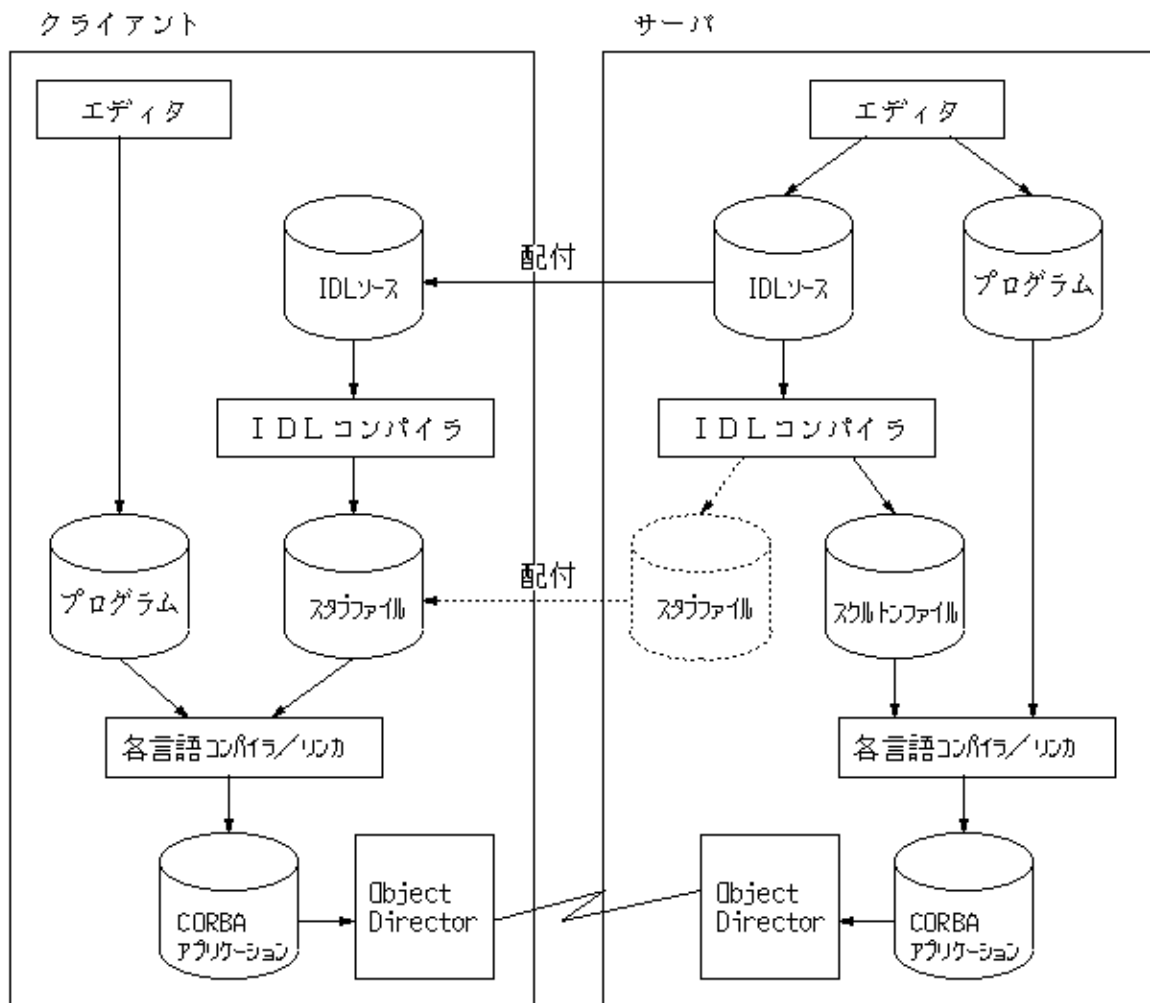
Interstage Application Serverについては、“Interstage Application Server 分散アプリケーション作成ガイド(CORBAサービス編)”および“Interstage Application Server 分散アプリケーション作成ガイド(コンポーネントトランザクションサービス編)”を参照してください。

Interstage/AIMApplicationDirectorについては、“Interstage システム開発手引書(AIM連携)”を参照してください。

#### 27.1.2 概要

CORBAアプリケーションを開発する場合の資産の流れを“[図27.1 CORBAアプリケーションの作成](#)”に示します。NetCOBOLの開発環境では、これら一連の作業を支援します。

図27.1 CORBAアプリケーションの作成



通常、クライアントのCORBAアプリケーションを作成するために、サーバはIDLソースファイルを配信しますが、クライアントとサーバの両方をCOBOLで開発する場合、サーバ作成時に作成したスタブファイルを利用できます(“図27.1 CORBAアプリケーションの作成”の点線)。これにより、IDLソースファイルの配信やクライアントでのIDLコンパイルを省略できます。

### 27.1.3 プロジェクト管理

プロジェクトマネージャでは、以下の処理を自動で行うことにより、CORBAアプリケーションの作成を支援します。

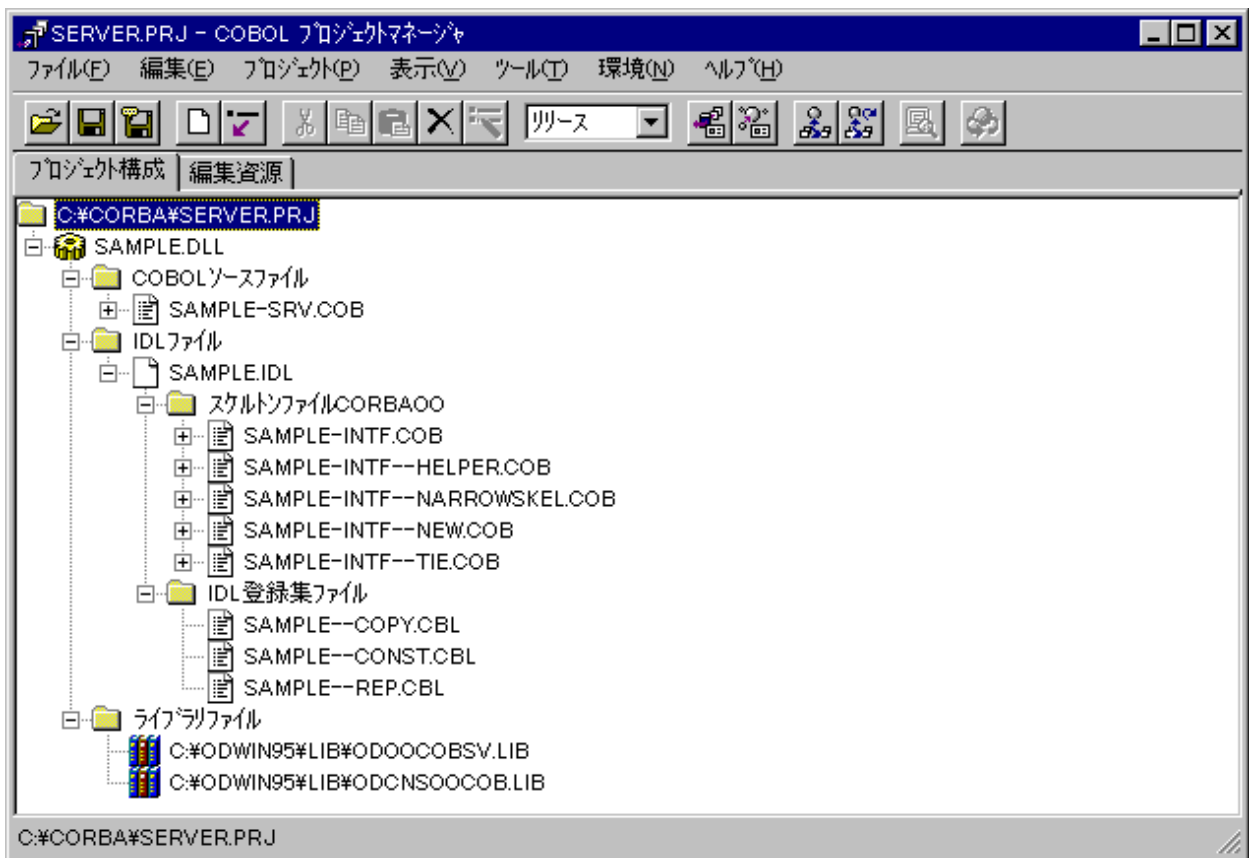
1. 利用者がIDLソースファイルを指定することにより、生成されるスケルトンファイルまたはスタブファイルのファイル名を類推し、プロジェクトの依存関係に登録します。
2. IDLソースファイルをIDLコンパイラを使用して翻訳し、COBOLインタフェースによるスケルトンファイルまたはスタブファイルを生成します。
3. スケルトンファイルまたはスタブファイルをCOBOLコンパイラで翻訳し、利用者アプリケーションとリンクします。

#### 27.1.3.1 サーバアプリケーションの作成

サーバアプリケーションを作成する場合、IDLソースファイルの運用方法によってプロジェクトの構成が変わります。

## IDLソースファイルを登録する場合

まず、IDLソースファイルをプロジェクトに登録するパターンです。この場合、プロジェクト管理機能がIDLコンパイラの起動を制御するため、利用者はIDLコンパイルの要、不要を特に意識することなく、サーバプログラムをビルドすることができます。ひとつのサーバアプリケーションに対して、ひとつのIDLソースファイルを作成するような運用形態に適しています。

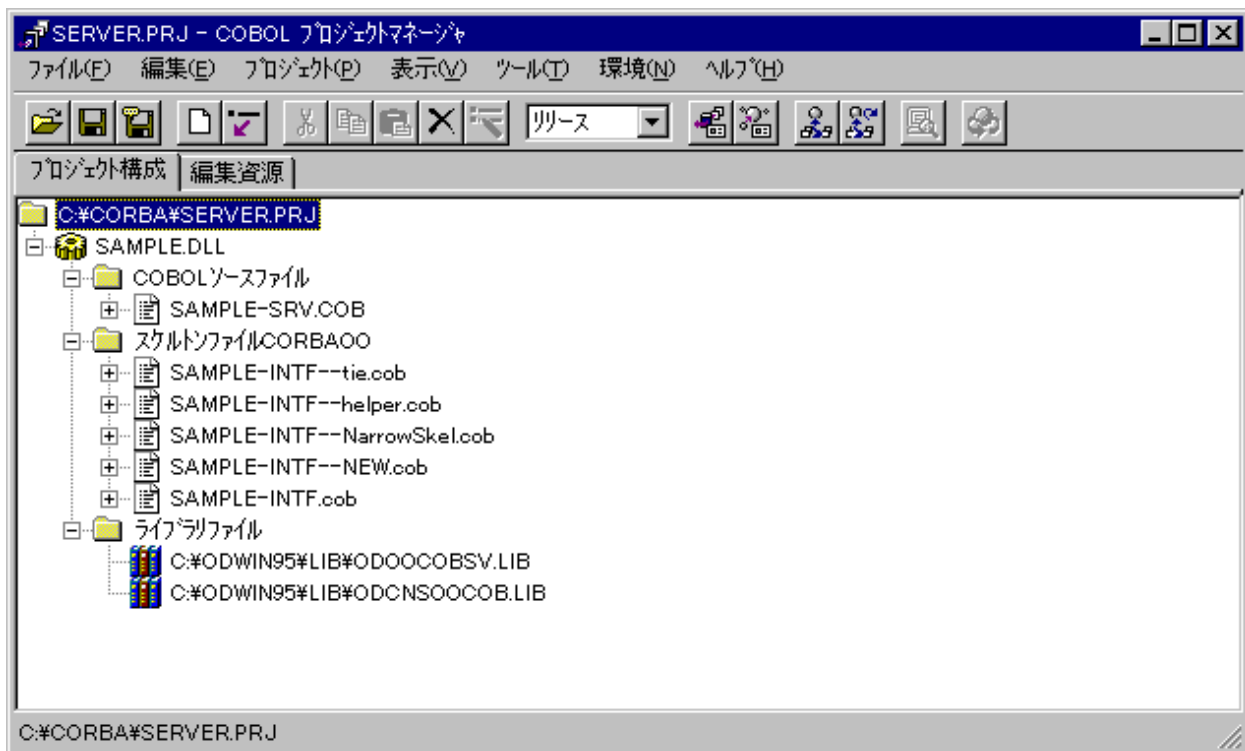


この場合、プロジェクトの作成は、以下の手順で行います。

1. プロジェクトに最終ターゲットファイルおよびソースファイルを登録します。
2. 最終ターゲットファイルを選択し、[編集]-[フォルダ作成]メニューから“IDLファイル”を選択して、IDLファイルのフォルダを登録します。
3. IDLファイルのフォルダに対して必要なIDLソースファイルを[編集]メニューの“新規作成”または“追加”でツリーに登録します。
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”を選択します。
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。[参照]“[27.1.4 \[Interstage\]ダイアログ](#)”
7. 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。
8. その他の必要なファイルを登録します。
9. “ビルド”または“リビルド”を実行します。

## IDLソースファイルを登録しない場合

次に、IDLソースファイルをプロジェクトに登録しないパターンです。この場合、利用者はプロジェクト外で必要に応じてIDLコンパイルを実行しなければなりません。複数のサーバアプリケーションを、ひとつのIDLソースファイルで管理するような運用形態に適しています。



この場合、プロジェクトの作成は、以下の手順で行います。

1. プロジェクトに最終ターゲットファイルおよびソースファイルを登録します。
2. 最終ターゲットファイルを選択して、[編集]メニューから“Interstage”を選択します。
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およびその他の必要な情報を登録します。[参照][“27.1.4 \[Interstage\] ダイアログ”](#)
4. 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。
5. その他の必要なファイルを登録します。
6. “ビルド”または“リビルド”を実行します。

### 27.1.3.2 クライアントアプリケーションの作成

クライアントアプリケーションは、サーバアプリケーションと同じ要領で作成できます。

クライアント側でIDLソースをもとにスタブファイルを生成する場合は、“[27.1.3.1 サーバアプリケーションの作成](#)”の“IDLソースファイルを登録する場合”を参照してください。また、サーバ側からスタブファイルが提供される場合は、“[27.1.3.1 サーバアプリケーションの作成](#)”の“IDLソースファイルを登録しない場合”を参照してください。このとき、説明中の“スケルトンファイル”は“スタブファイル”に読み替えてください。

### 27.1.4 [Interstage]ダイアログ

[Interstage]ダイアログを使用することにより、CORBAアプリケーションのプロジェクトの依存関係を生成することができます。CORBAアプリケーションを作成する場合は、必ずこのダイアログを使用してください。

[Interstage]ダイアログは、[編集]メニューから“Interstage”を選択することにより表示されます。

## IDLソースファイルを登録する場合

### ラジオボタン

#### COBOL

IDLソースファイルをCOBOLインタフェース(プログラムインタフェース)でIDLコンパイルすることを指定します。

#### オブジェクト指向COBOL

IDLソースファイルをオブジェクト指向COBOLインタフェース(クラスインタフェース)でIDLコンパイルすることを指定します。

#### CORBA

CORBAアプリケーションを作成することを指定します。これを選択すると、IDLcコマンドを使用してIDLソースファイルをIDLコンパイルします。

#### トランザクション

トランザクションアプリケーションを作成することを指定します。これを選択すると、tdcコマンドを使用してIDLソースファイルをIDLコンパイルします。

#### グローバルサーバ

AADアプリケーションを作成することを指定します。これを選択すると、aadidlcコマンドを使用してIDLソースファイルをIDLコンパイルします。

#### サーバ

サーバアプリケーションを作成することを指定します。

#### クライアント

クライアントアプリケーションを作成することを指定します。

#### シングルスレッド

シングルスレッドプログラムを作成することを指定します。

#### マルチスレッド

マルチスレッドプログラムを作成することを指定します。

### チェックボックス

#### IDL翻訳オプションの設定だけを行う

プロジェクトの依存関係を変更しないで、IDL翻訳オプションの設定だけを行う場合に指定します。

### エディットボックス

#### IDL翻訳オプション

IDLコンパイル時の翻訳オプションを指定します。



## ボタン

### OK

Interstageに関する情報の設定が終了したときに押します。このボタンが押されると、[IDL翻訳オプションの設定だけを行う]を指定した場合はプロジェクトの依存関係を更新せずIDL翻訳オプションだけを設定します。

### キャンセル

[Interstage]ダイアログが表示される前の状態に戻して、ダイアログを終了します。

### ヘルプ

ヘルプウィンドウを開くときに選択します。

## IDLソースファイルを登録しない場合

## ラジオボタン

### COBOL

スケルトンファイルまたはスタブファイルがCOBOLインタフェース(プログラムインタフェース)でIDLコンパイルされていることを指定します。

### オブジェクト指向COBOL

スケルトンファイルまたはスタブファイルがオブジェクト指向COBOLインタフェース(クラスインタフェース)でIDLコンパイルされていることを指定します。

### CORBA

CORBAアプリケーションを作成することを指定します。

### トランザクション

トランザクションアプリケーションを作成することを指定します。

### グローバルサーバ

AADアプリケーションを作成することを指定します。

#### サーバ

サーバアプリケーションを作成することを指定します。

#### クライアント

クライアントアプリケーションを作成することを指定します。

#### シングルスレッド

シングルスレッドプログラムを作成することを指定します。

#### マルチスレッド

マルチスレッドプログラムを作成することを指定します。

#### エディットボックス

##### スケルトン/スタブファイル名

リストボックスに追加するスケルトンファイル名またはスタブファイル名を指定します。

#### リストボックス

##### 一覧

取り込む対象となるスケルトンファイル名またはスタブファイル名を表示します。

#### ボタン

##### OK

Interstageに関する情報の設定が終了したときに押します。

##### キャンセル

[Interstage]ダイアログが表示される前の状態に戻して、ダイアログを終了します。

##### ヘルプ

ヘルプウィンドウを開くときに選択します。

##### 参照

ファイルを参照するときに選択します。

##### 追加

リストボックスにファイルを追加するときに選択します。

##### 削除

リストボックスからファイルを削除するときに選択します。

## 27.1.5 ターゲットオブジェクトファイル

---

ターゲットオブジェクトファイルは、プログラムインタフェースによるCORBAアプリケーションを作成する場合に、スケルトンファイルまたはスタブファイルを翻訳した結果、生成される目的プログラムです。

プログラムインタフェースを使用した場合に作成されるスケルトンファイルおよびスタブファイルには、ひとつのソースファイルの中に複数のプログラムが定義される場合があります。この場合、プロジェクトマネージャは、それぞれのプログラムごとに目的プログラムを生成します。このときのターゲットオブジェクトファイル名は、プログラム名.OBJになります。

なお、“[27.1.4 \[Interstage\]ダイアログ](#)”を使用することにより、プロジェクトにターゲットオブジェクトファイル名を自動的に登録することができます。

## 27.1.6 注意事項

---

プロジェクトマネージャを使用してCORBAアプリケーションを作成する場合の注意事項を以下に示します。

- 本機能は、CORBAのCOBOLインタフェースおよびオブジェクト指向COBOLインタフェースを使用できます。他の言語によるインタフェースは使用できません。

- IDLソースファイルの依存関係をプロジェクトに登録した後に、IDLソースファイル名を変更したり、IDLソースファイルの内容を変更したりした場合は、[Interstage]ダイアログを再起動して依存関係を登録し直す必要があります。この場合、以前の依存関係はすべて削除され、新しい依存関係が登録されます。
- 複数の最終ターゲットファイルに同一ファイル名のIDLソースファイルを登録する場合(たとえば、IDLソースファイルに複数のインタフェースを定義し、インタフェースごとに複数の最終ターゲットファイルを作成するような場合)、最終ターゲットファイルごとに必要なスケルトンファイルまたはスタブファイルだけを残して、不要なスケルトンファイルまたはスタブファイルを削除してください。削除しないと翻訳エラーが発生する場合があります。
- 継承関係を記述したIDLソースファイルを登録する場合、継承元のスケルトンファイルを削除してください。削除しない場合、リンクエラーが発生することがあります。
- プロジェクトにIDLソースファイルの依存関係を自動登録する機能は、IDLコンパイラが作成するスケルトンファイルおよびスタブファイルを解析することにより実現しています。このため、プロジェクトにCORBAアプリケーションを登録するとき、IDLコンパイラの処理でエラーが発生した場合、IDLソースファイルの依存関係の自動登録は行いません。
- COBOLインタフェースによるCORBAサーバアプリケーションを作成する場合、最終ターゲットファイル名(DLL名)に対応したスケルトンファイルだけをプロジェクトに自動で登録します。
- [Interstage]ダイアログを使用すると、必要な翻訳オプションが自動で設定されます。この翻訳オプションがないと翻訳エラーが発生します。
- [Interstage]ダイアログを使用すると、最終ターゲットファイル配下に必要なインポートライブラリが自動で設定されます。このインポートライブラリがないとリンクエラーが発生します。
- トランザクションアプリケーションで例外を処理する場合、最終ターゲットファイル配下に、以下のインポートライブラリを登録する必要があります。

|                                               |
|-----------------------------------------------|
| Interstageインストールフォルダ¥odwin¥lib¥ODC0BCBLSV.LIB |
|-----------------------------------------------|

- エディタを使用して、プロジェクトに登録したスケルトンファイルやスタブファイルを参照できます。ただし、編集はできません。

## 27.2 AADアプリケーション

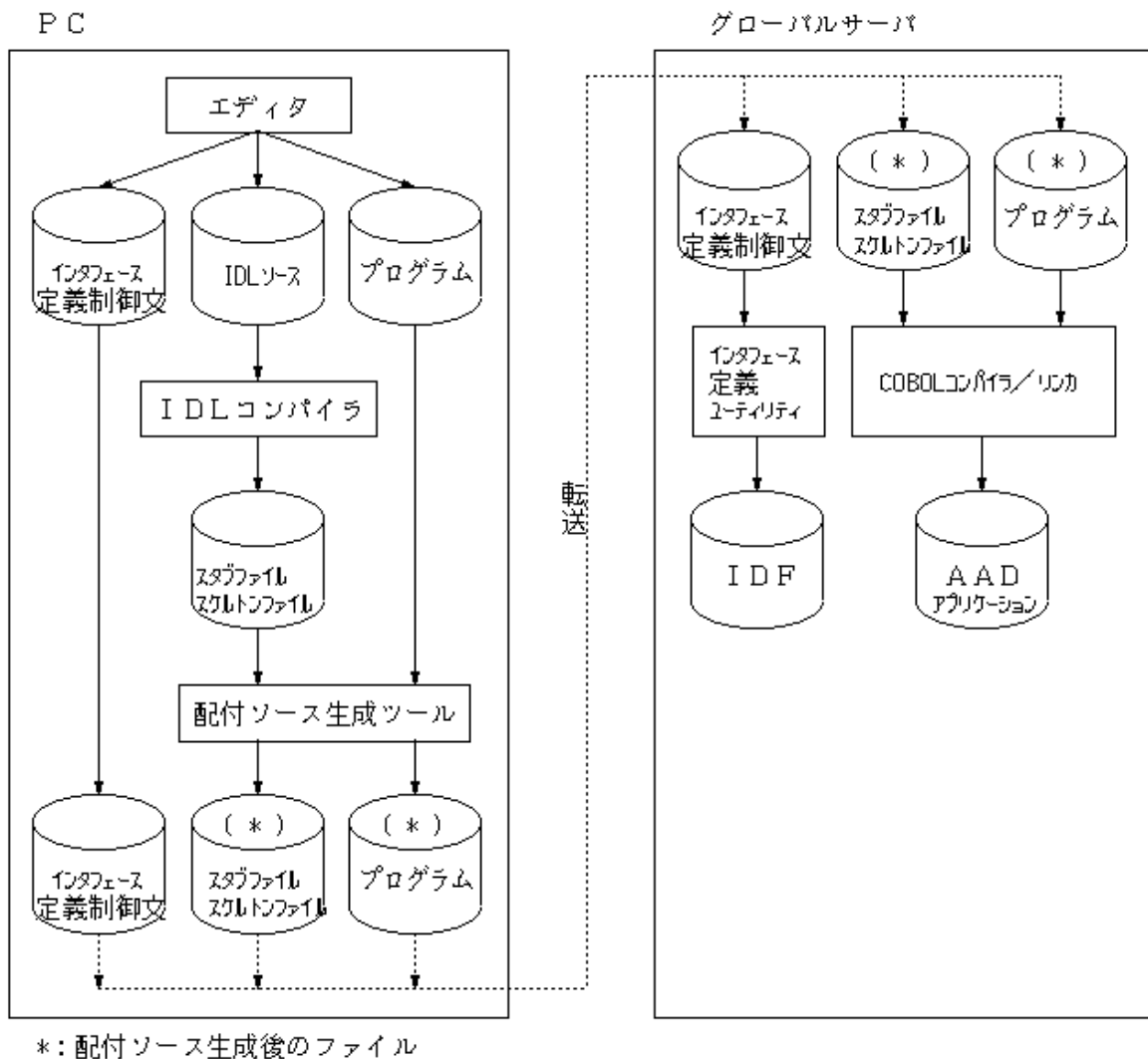
ここでは、グローバルサーバ上で動作するAADアプリケーションの作成について説明します。

### 27.2.1 概要

AADアプリケーションはグローバルサーバ上で動作するCORBAアプリケーションの内、Interstageスタイルで開発したアプリケーションです。AADアプリケーションを開発する場合の資産の流れを“[図27.2 AADアプリケーションの作成](#)”に示します。

AADアプリケーション開発の詳細については、“Interstageシステム開発手引書(AIM連携)”を参照してください。

図27.2 AADアプリケーションの作成



プロジェクトマネージャは以下の処理を行うことにより、AADアプリケーションの作成を支援します。

- ・ IDLコンパイラを使用することでIDLソースファイルを翻訳し、スケルトンファイルまたはスタブファイルをプロジェクトに登録します。
- ・ 配付ツールを使用することで、PC上の資源をグローバルサーバで使用可能な資源(これを配付ソースと呼ぶ)に変換し、配付ソースをプロジェクトに登録します。
- ・ PC上のファイルをグローバルサーバ上に転送します。
- ・ グローバルサーバ上で翻訳、リンクするためのJCLおよびCLISTの雛型を生成します。
- ・ グローバルサーバ上のジョブを起動し、その結果を表示します。
- ・ プログラムの修正を容易にするため、配付ソース生成前のファイル名およびプログラム名と、配付ソース生成後の名前の対応を表示します。

## 27.2.2 AADアプリケーションの作成手順

AADアプリケーションの作成手順を、IDLソースファイルを登録する場合と登録しない場合とに分けて説明します。

### IDLソースファイルを登録する場合

IDLソースファイルをプロジェクトに登録するパターンです。AADアプリケーションの作成は、以下の手順で行います。

1. 最終ターゲットファイルを登録します。

最終ターゲットファイルをプロジェクトに登録します。最終ターゲットファイルは以下の単位で登録します。[参照]“[6.4.4.1 最終ターゲットファイルの登録](#)”

- スケルトンおよびオペレーション処理プログラム
- 出口プログラム
- スタブ

2. IDLソースファイルを登録します。

最終ターゲットファイルを選択し、[編集]-[フォルダ作成]メニューから“IDLファイル”を選択して、IDLファイルのフォルダを登録します。

IDLファイルのフォルダに対してIDLソースファイルを[編集]メニューの“新規作成”または“追加”でプロジェクトに登録します。



#### 注意

IDLソースファイルはプロジェクトファイルと同じフォルダに格納してください。

3. スケルトンファイルまたはスタブファイルを登録します。

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”を選択します。

表示された“[27.1.4 \[Interstage\]ダイアログ](#)”で“グローバルサーバ”を選択します。また、その他の必要な情報を設定し、[OK]ボタンをクリックします。

→ プロジェクトマネージャのツリーにスケルトンファイルまたはスタブファイルが登録されます。

4. COBOLソースファイルをプロジェクトに登録します。[参照]“[6.4.4.2 COBOLソースファイルの登録](#)”

5. その他の必要なファイルを登録します。[参照]“[6.4.4.6 その他必要なファイルの登録](#)”

6. 配付ソースを生成します。[参照]“[27.2.3 配付ソース生成](#)”

→ プロジェクトマネージャのツリーに、配付ソースファイルが登録されます。

7. インタフェース定義制御文を登録します。

最終ターゲットファイルを選択し、[編集]-[フォルダ作成]メニューから“その他”を選択して、その他のフォルダを登録します。その他のフォルダに対してインタフェース定義制御文を[編集]メニューの“追加”でプロジェクトに登録します。



#### 注意

インタフェース定義制御文の詳細については、“Interstageシステム開発手引書(AIM連携)”を参照してください。

8. 資産をグローバルサーバへ送信します。[参照]“[27.2.4 資産転送](#)”

9. ロードモジュールを作成するためのJCLまたはCLISTの雛型ファイルを生成します。[参照]“[27.2.5 ビルド用JCL/CLISTの生成](#)”

→ プロジェクトマネージャのツリーに、JCL/CLISTの雛型ファイルが登録されます。

10. グローバルサーバの環境に合わせて、JCL/CLISTの雛型ファイルを修正します。

11. JCL/CLISTをグローバルサーバ上へ転送します。[参照]“[27.2.4 資産転送](#)”

12. JCL/CLISTを起動し、ロードモジュールを作成します。[参照]“[27.2.6 ターゲットビルド](#)”

13. 配付ソース生成前のファイル名およびプログラム名と、配付ソース生成後の名前の対応を表示して、プログラムの修正をします。  
[参照]“[27.2.7 配付ソース対応表示](#)”

## IDLソースファイルを登録しない場合

IDLソースファイルをプロジェクトに登録しないパターンです。この場合、利用者はプロジェクト外で必要に応じてIDLコンパイルを実行しなければなりません。AADアプリケーションの作成は、以下の手順で行います。

1. 最終ターゲットファイルを登録します。

最終ターゲットファイルをプロジェクトに登録します。最終ターゲットファイルは以下の単位で登録します。[参照]“[6.4.4.1 最終ターゲットファイルの登録](#)”

- スケルトンおよびオペレーション処理プログラム
- 出口プログラム
- スタブ

2. スケルトンファイルまたはスタブファイルを登録します。

最終ターゲットファイルを選択し、[編集]メニューから“Interstage”を選択します。

表示された“[27.1.4 \[Interstage\]ダイアログ](#)”で“グローバルサーバ”を選択します。また、スケルトンファイルまたはスタブファイルを設定し、[OK]ボタンをクリックします。

→ プロジェクトマネージャのツリーにスケルトンファイルまたはスタブファイルが登録されます。

3. COBOLソースファイルをプロジェクトに登録します。[参照]“[6.4.4.2 COBOLソースファイルの登録](#)”

4. 以降の作業は、IDLソースファイルを登録する場合の5.以降と同じです。

## 27.2.3 配付ソース生成

---

配付ソース生成とは、PC上で作成した資源をグローバルサーバで使用するための資源に変換する操作です。

プロジェクトに登録している、次のファイルから配付ソースを生成することができます。

- COBOLソースファイル
- 登録集ファイル
- スケルトンファイルAADフォルダ配下のスケルトンファイル
- スタブファイルAADフォルダ配下のスタブファイル
- IDL登録集ファイルAADフォルダ配下のインタフェース定義用登録集ファイル

生成した配付ソースファイルは、プロジェクトファイルを格納したフォルダのサブフォルダに出力します。

配付ソースの詳細については、“[Interstageシステム開発手引書\(AIM連携\)](#)”を参照してください。

以下に、配付ソース生成の方法を説明します。

1. 配付ソース生成に必要なファイルを格納します。

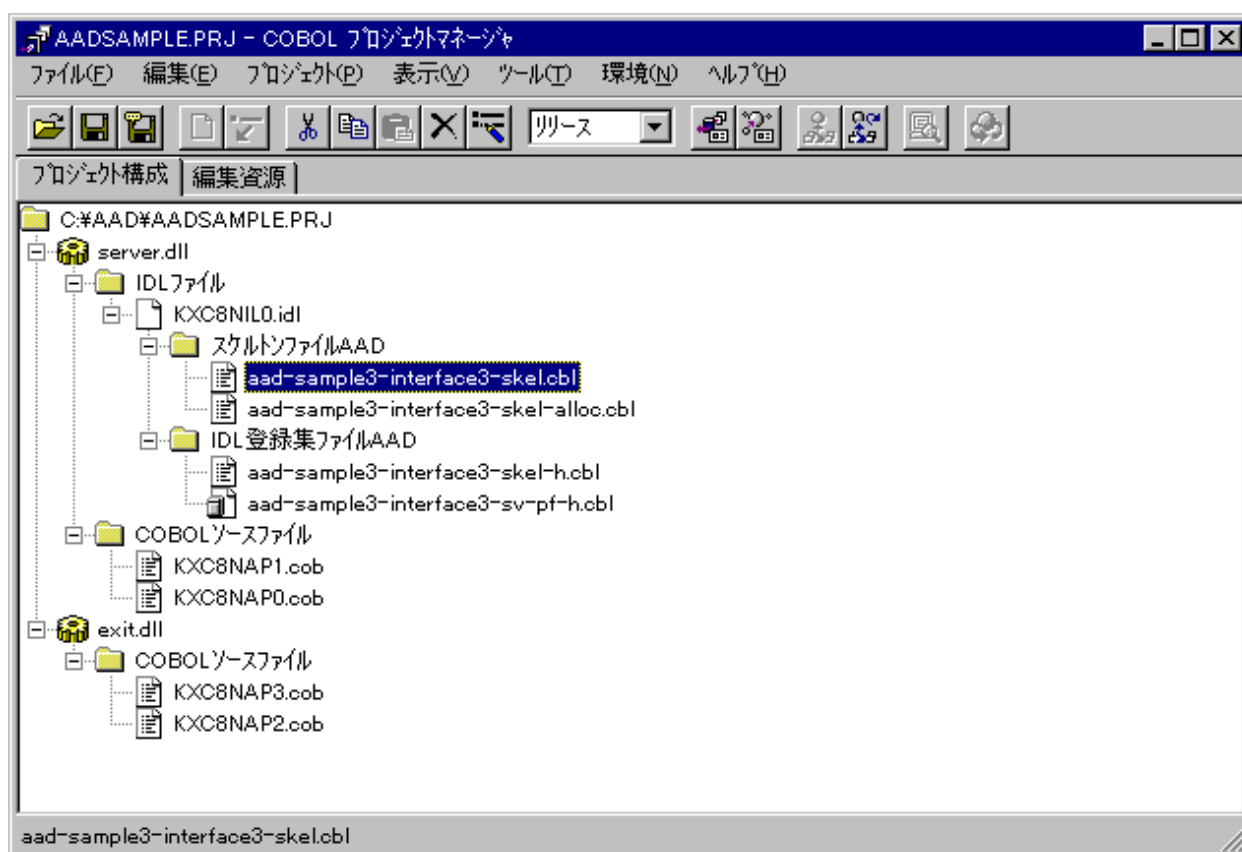
プロジェクトファイルと同じフォルダに、IDLソースファイル、スケルトンプレフィクス一覧およびスタブプレフィクス一覧を格納します。必要ならば、ユーザファイル名対応表およびユーザプログラム名対応表も格納します。

これらのファイルの詳細については、“[Interstageシステム開発手引書\(AIM連携\)](#)”を参照してください。

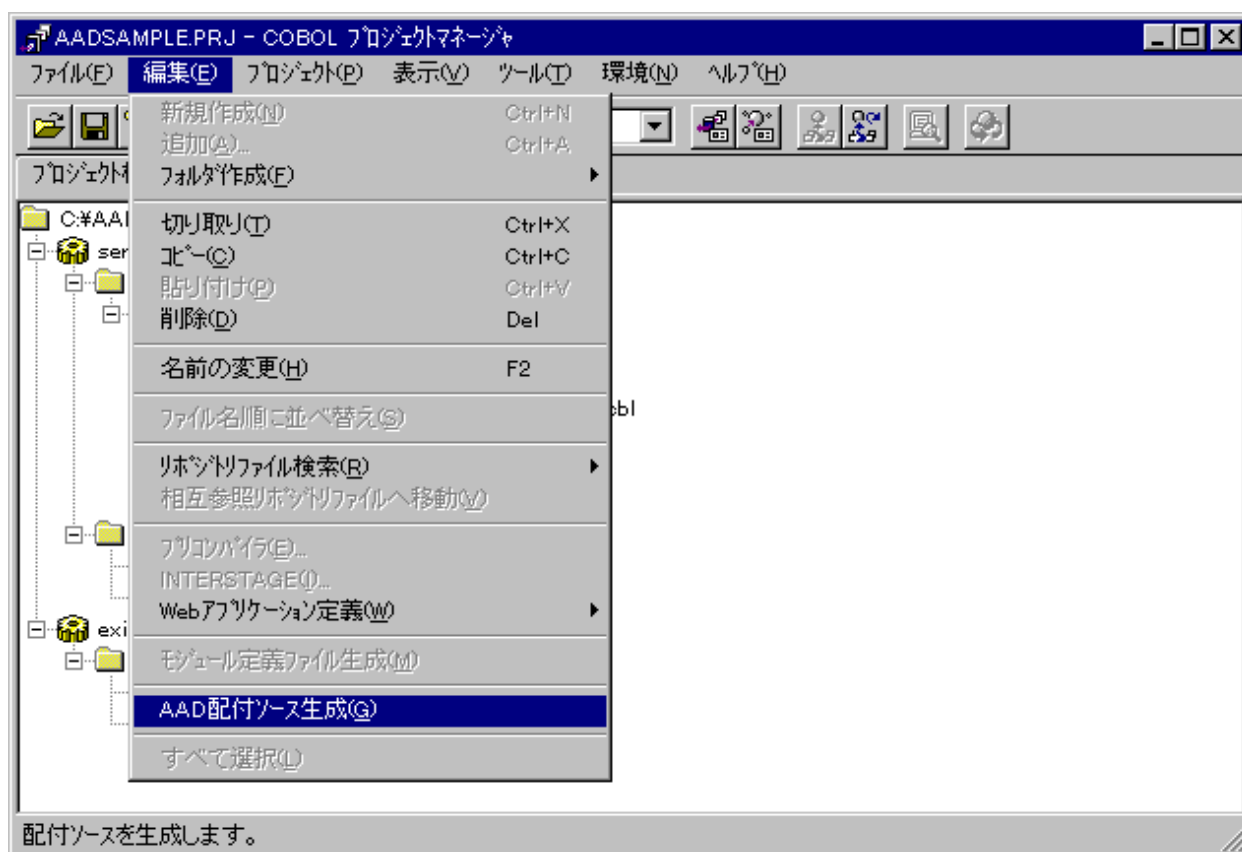
2. プロジェクトのプロパティを設定します。

プロジェクトを開き、[ファイル]メニューの“プロパティ”を選択します。表示された[プロパティ]ダイアログで、“[27.2.3.1 \[AAD配付ソース生成\]ページ](#)”を選択します。配付ソースファイルの出力先と、同じ名前のファイルが存在したときに上書きするかを指定します。

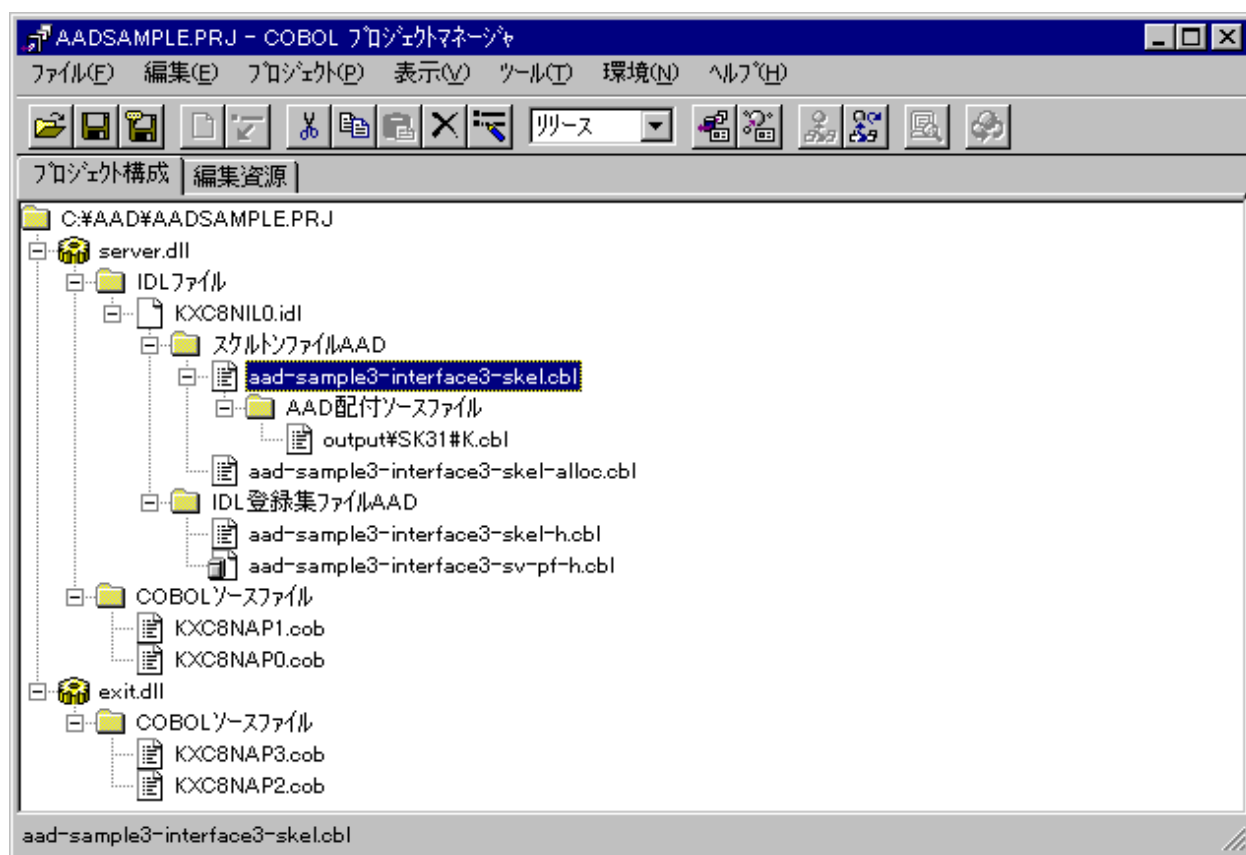
3. 生成するファイルを選択します。



4. [編集]メニューの“AAD配付ソース生成”を選択します。

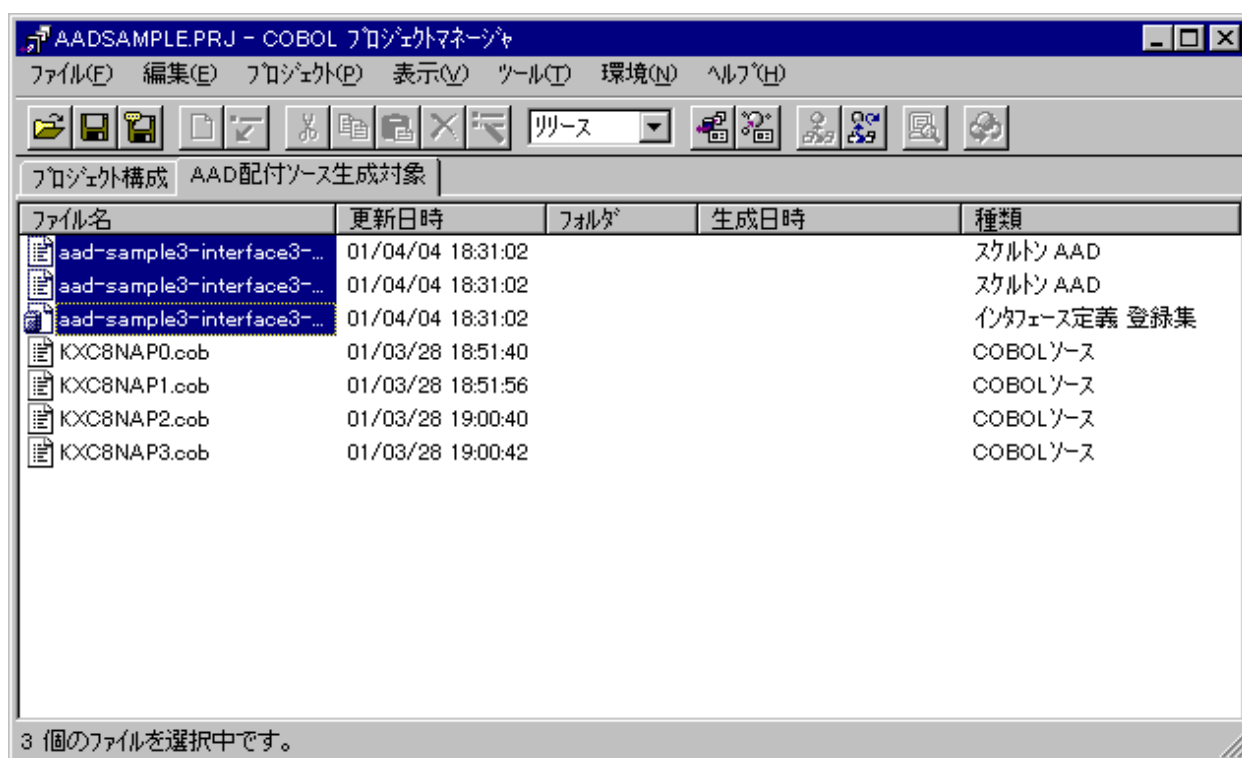


生成に成功すると、生成元のファイルの下にAAD配付ソースファイルフォルダと生成したAAD配付ソースファイルが登録されます。



## 複数ファイルを一括して配付ソース生成する方法

“6.4.10 リストビュー”を使用することによって、複数のファイルから一括して配付ソースを生成することもできます。





以下に手順を示します。

1. [表示]-[リスト表示内容]メニューから“AAD配付ソース生成対象”を選択します。これによって、リストビューに、生成対象ファイルだけが一覧表示されます。また、同時に、以下の情報も画面上に表示されます。

**ファイル名**

生成対象のファイル名です。

**更新日時**

生成対象ファイルの最終更新日時です。

**フォルダ**

生成対象ファイルが存在するフォルダです。

**生成日時**

最後に配付ソース生成を実行した日時です。

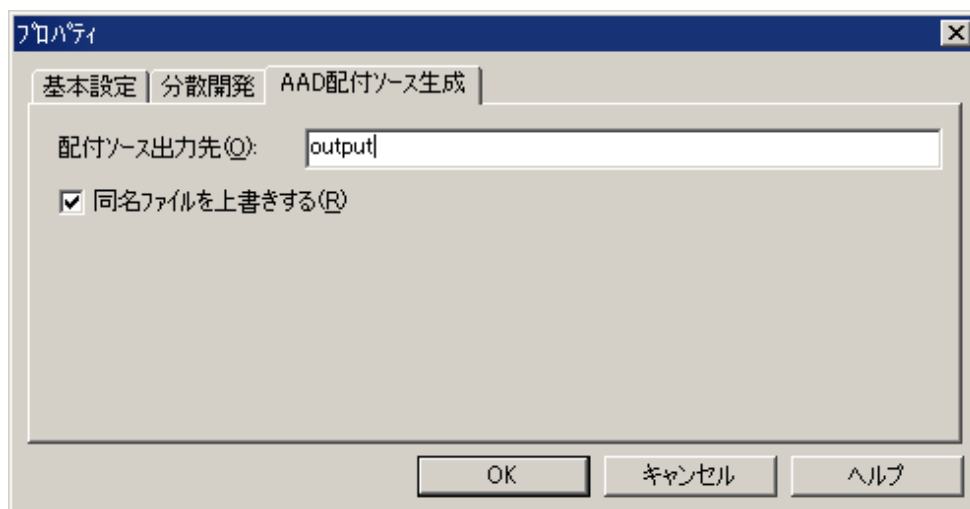
**種類**

生成対象ファイルの種類です。

2. リストビューに表示されたファイルのうち、配付ソース生成が必要なものを選択します。
3. [編集]メニューの“AAD配付ソース生成”を選択します。

### 27.2.3.1 [AAD配付ソース生成]ページ

配付ソース生成のしかたについて指定する場合は、[プロパティ]ダイアログの[AAD配付ソース生成]ページを使用します。



**エディットボックス**

**配付ソース出力先**

配付ソースの出力先フォルダ名を指定します。プロジェクトファイル格納フォルダのサブフォルダ名を指定します。フォルダがない場合は新規に作成されます。初期値は“output”です。

**チェックボックス**

**同名ファイルを上書きする**

配付ソース出力先に同じ名前のファイルがすでに存在するとき、上書きする場合に指定します。指定がない、かつ、同じ名前のファイルがすでに存在する場合、配付ソース生成はエラーになります。初期値はチェックされた状態(上書きを行う指定)です。

## 27.2.4 資産転送

プロジェクトに登録しているCOBOLソース、登録集、画面帳票定義体および[その他]フォルダのファイルをグローバルサーバの区分編成ファイルへ転送します。資産を転送するには、PowerGEM Plusをインストールする必要があります。

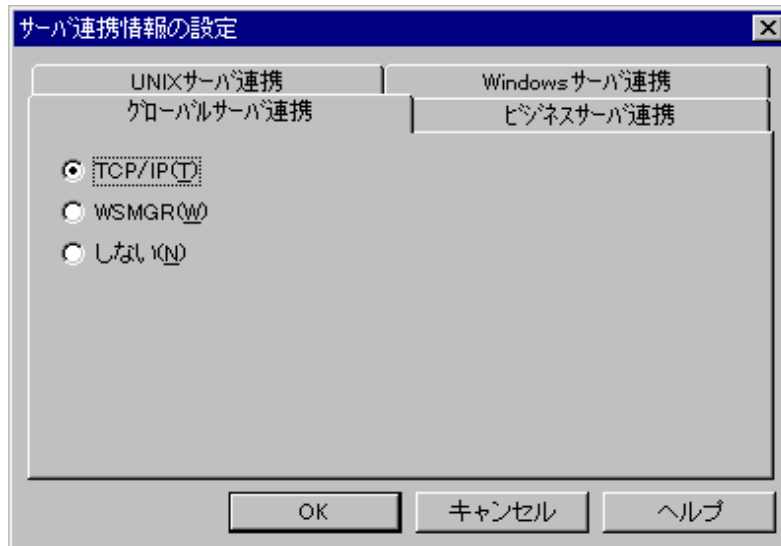
転送の手順を示します。

1. グローバルサーバとの接続方法を設定します。

PowerGEM Plus開発マネージャの[オプション]メニューの“環境設定”を選択して、[環境設定]ダイアログを表示します。

[サーバ連携]の、“ログオンユーザの環境”ボタンをクリックして、[サーバ連携情報の設定]ダイアログを表示します。

サーバ情報の設定の方法は、PowerGEM Plus開発マネージャのヘルプを参照してください。



2. グローバルサーバ連携情報を設定します。

プロジェクトマネージャを起動して、[プロジェクト]-[分散開発]メニューの“グローバルサーバ連携情報”を選択して、[グローバルサーバ連携情報]ダイアログを表示します。なお、グローバルサーバ連携情報として設定する項目は接続方法により異なります。

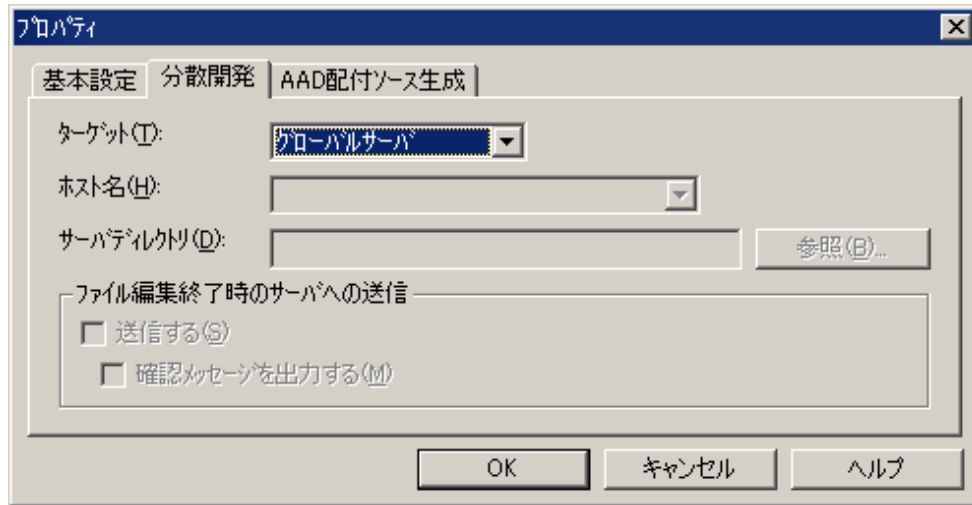
詳細については、PowerGEM Plus開発マネージャのヘルプを参照してください。



3. 転送する資産を含むプロジェクトを開きます。

4. プロジェクトのプロパティを設定します。

[ファイル]メニューの“プロパティ”を選択して、[プロパティ]ダイアログを表示します。ダイアログの[分散開発]ページで、“グローバルサーバ”を選択します。



5. 資産を転送するための情報を設定します。

[プロジェクト]-[分散開発]メニューの“送信”を選択して、[送信]ダイアログを表示します。ここでは転送する資産と、転送先の情報を設定します。[参照]“[27.2.4.1 \[送信\]ダイアログ](#)”

6. [送信]ボタンを押します。

指定した資産をグローバルサーバへ転送します。

### 27.2.4.1 [送信]ダイアログ

送信ダイアログを使用することにより、プロジェクトに登録しているファイルをグローバルサーバへ送信することができます。



## コンボボックス

### ファイルの種別

送信するファイルの種別を選択します。以下に示す種別があります。

#### ソース

- ツリー上で[COBOLソースファイル]フォルダに登録されているCOBOLソースファイル。ただし、[プリコンパイラ]フォルダ、[INSDBINF]フォルダ、[AAD配付ソースファイル]フォルダが下位階層に定義されていないCOBOLソースファイル。
- [プリコンパイラ]フォルダの最下位の階層に定義されたファイル。
- [AAD配付ソースファイル]フォルダに定義された配付ソース。

#### 登録集

- [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。[IDL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は対象になりません。
- [プリコンパイラ]フォルダの最下位の階層に定義されたCOBOLソースファイルの[インクルード]フォルダに定義されたインクルードファイル。
- [IDL登録集ファイルAAD]フォルダのインタフェース定義用登録集の配付ソース。

#### 画面帳票定義体

- [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。画面定義体ファイルの拡張子は、“.SMD”でなければなりません。[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以外のファイルは、対象になりません。

#### その他

- [その他]フォルダに登録されている任意のファイル。

### ホスト名

サーバのホスト名を選択します。[グローバルサーバ連携情報]ダイアログボックスで指定したホスト名の一覧を表示します。

### コード系

グローバルサーバのコード系を以下の種類から選択します。

[データの種別]に“テキスト”が選択された場合に有効になります。

- カナ文字EBCDIC
- 英小文字EBCDIC
- EBCDIC(ASCII)

## スタティックテキスト

### 最終送信日時

[ファイル種別]で選択中の種別のファイルを最後に送信した日時を表示します。送信していない場合は“なし”と表示します。

## エディットボックス

### ファイル名(送信元ファイル名指定用)

[ファイルの種別]で選択した種別の送信対象となるファイル名を指定します。以下に、ファイルの指定方法について説明します。

- ファイル名は絶対パス、または相対パスで指定します。相対パスで指定した場合、プロジェクトファイル格納フォルダからのパスになります。
- 複数ファイルの指定が可能です。
- 一文字の“\*”を指定すると、プロジェクトに登録しているファイルのうち、[ファイルの種別]で選択している種別に一致するすべてのファイルが対象になります。
- 一文字の“\*”以外のワイルドカードは指定できません。
- 送信元ファイル名の拡張子を外した部分が送信先の区分編成ファイルのメンバ名となります。そのため、ファイル名は区分編成ファイルのメンバ名として有効なものだけが指定できます。

## ファイル名(送信先ファイル名指定用)

グローバルサーバのファイル名を指定します。以下に、ファイルの指定方法について説明します。

- 送信先ファイルは区分編成ファイルを指定してください。区分編成ファイルのメンバ名を指定することはできません。
- 複数ファイルの指定はできません。
- ファイル名は、完全修飾名で指定します。ファイル名を引用符で囲む必要はありません。ただし、[グローバルサーバ連携情報]ダイアログボックスで[サーバファイル名付加文字列の付加]を指定した場合に、“ファイル名を引用符で囲まない”で指定すると、指定されたファイル名の先頭にサーバファイル名付加文字列が付加されます。
- 送信先のファイルが存在しない場合、区分編成ファイルが新規作成されます。

## ファイルパスワード

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

## VOL通番

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

## レコード長

送信先ファイルのレコード長を指定します。0を指定した場合は、省略値になります。詳細は、“[表27.1 指定値とグローバルサーバのファイル形式との関係](#)”を参照してください。

## ラジオボタン

### データの種別

送信するファイルの種別を“テキスト”または“バイナリ”から選択します。“テキスト”を選択した場合は、シフトJISから指定された送信先のコード系に変換して送信します。

### レコード形式

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

## チェックボックス

### 更新されたファイルのみ

ファイル種別ごとの最終送信日時より後に更新されたファイルだけを送信する場合、指定します。

### 上書き

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

## ボタン

### 参照(送信元ファイル名指定用)

プロジェクトに登録している[ファイルの参照]ダイアログボックスを表示します。[参照]“[27.2.4.2 \[参照\] \(送信元ファイル名指定用\) ダイアログ](#)”

### 参照(送信先ファイル名指定用)

グローバルサーバのファイル名を参照する[ファイルの参照]ダイアログボックスを表示します。[送信]ダイアログの“ファイル名(送信先ファイル名指定用)”への入力により、[ファイルの参照]ダイアログボックスの[ファイル一覧]は次のように表示されます。

- “指定名.%”を指定した場合、指定名で始まるファイルの一覧を表示します。ファイル名を省略した場合は、ユーザ識別名が指定名になります。
- ファイル名を引用符で囲まないで指定した場合、かつ、[グローバルサーバ連携情報]ダイアログボックスで“サーバファイル名付加文字列の付加”を指定していると、“サーバファイル名付加文字列”に指定した文字列がファイル名の先頭に付加されます。
- ファイル名を引用符で囲んで指定した場合、または、“サーバファイル名付加文字列の付加”を指定していない場合は、指定したファイル名が有効になります。

## 送信

指定した資産をグローバルサーバへ転送します。

## 終了

[送信]ダイアログを閉じます。

## ヘルプ

ヘルプウィンドウを開きます。



## 注意

- ダイアログで指定する[ファイルの種別]、[データの種別]とグローバルサーバのファイル形式には下の表の関係があります。

表27.1 指定値とグローバルサーバのファイル形式との関係

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

- ファイルはグローバルサーバの区分編成ファイルへ送信します。
- ファイルで使用しているタブは、空白文字に変換してグローバルサーバに格納します。空白文字の文字数は翻訳オプションTABの値になります。このファイルをグローバルサーバからPCへ受信してもタブは復元されません。
- 送信先のファイル名に“¥”が含まれている場合は、“¥”の代わりに“\$”を指定します。
- [送信]ダイアログボックスに指定した値はプロジェクトファイルに保存します。そのため、プロジェクトを開くときに読み取り専用で開いていると、この情報は保存されません。
- 画面定義体のコード情報は、カナ文字EBCDIC、英小文字EBCDICまたはEBCDIC(ASCII)のいずれかです。

## 27.2.4.2 [参照](送信元ファイル名指定用)ダイアログ

送信するファイルを選択します。

“27.2.4.1 [送信]ダイアログ”の“参照(送信元ファイル名指定用)”ボタンをクリックして表示します。プロジェクトに登録しているファイルのうち、[ファイルの種別]で選択した種別の対象ファイルの一覧を表示します。複数のファイルを選択することができます。

以下に、[参照]ダイアログについて説明します。



#### リストビュー

##### ファイル一覧

プロジェクトに登録しているファイル名の一覧です。ファイル名、ファイルの最終更新日時、パス名を表示します。パス名が相対パスの場合、プロジェクトファイルを格納するフォルダからのパスです。

#### ボタン

##### OK

ダイアログを閉じます。

##### キャンセル

ファイルの選択を破棄してダイアログを閉じます。

### 27.2.4.3 受信ダイアログ

グローバルサーバにある順編成ファイル、区分編成ファイルのメンバをPCに受信する場合は、[受信]ダイアログを使用します。

[プロジェクト]-[分散開発]メニューの“受信”を選択して、[受信]ダイアログを表示します。

## コンボボックス

### ホスト名

サーバのホスト名を選択します。[グローバルサーバ連携情報]ダイアログボックスで指定したホスト名の一覧を表示します。

### コード系

グローバルサーバのコード系を以下の種類から選択します。[データの種別]に“テキスト”が選択された場合に有効になります。

- カナ文字EBCDIC
- 英小文字EBCDIC
- EBCDIC(ASCII)

## エディットボックス

### ファイル名(受信元ファイル名指定用)

グローバルサーバのファイル名、メンバ名を指定します。以下に、指定方法を説明します。

- 区分編成ファイルの場合は、複数メンバの指定が可能です。
- 区分編成ファイルのメンバ名を指定する場合は、ファイル名に続いてメンバ名を()で囲んで指定します。メンバ名を複数指定する場合は、メンバ名を空白文字で区切って指定します。



例

PO.COBO(LMEM1)

PO.COBO(LMEM1 MEM2)

- 区分編成ファイルのメンバ名を省略した場合は、全メンバが処理の対象になります。
- 複数ファイルの指定はできません。
- ファイル名は、完全修飾名で指定します。ファイル名を引用符で囲む必要はありません。ただし、[グローバルサーバ連携情報]ダイアログボックスで“サーバファイル名付加文字列の付加”を指定した場合に、ファイル名を引用符で囲まないで指定すると、指定されたファイル名の先頭にサーバファイル名付加文字列が付加されます。



## ファイルパスワード

受信元のファイルがパスワードで保護されている場合、ファイルパスワードを指定します。

## VOL通番

受信元のファイルがカタログされていない場合、ボリューム通し番号を指定します。

## ファイル名(受信先ファイル名指定用)

受信先のフォルダ名またはファイル名を指定します。以下に、指定方法を説明します。

- フォルダ名、ファイル名は絶対パス、または相対パスで指定します。相対パスで指定した場合、プロジェクトファイル格納フォルダからのパスになります。
- 受信元ファイルに区分編成ファイルまたは区分編成ファイルのメンバ名を指定した場合は、フォルダを指定します。
- 受信元ファイルに順編成ファイルを指定した場合は、ファイル名を指定します。

## 拡張子

受信先のファイル名に付加する拡張子を指定します。受信先の「ファイル名」にフォルダを指定した場合に有効になります。

## ラジオボタン

### データの種別

受信先のファイルの種別を“テキスト”または“バイナリ”から選択します。“テキスト”を選択した場合は、指定したコード系からシフトJISに変換して受信します。

## チェックボックス

### 上書き

受信先の既存のファイルを上書きする場合に指定します。

## 固定長COBOLソースまたは登録集の受信:

固定長のCOBOLソースまたは登録集を受信する場合に指定します。指定した場合、固定長のCOBOLソースまたは登録集の受信となり、受信するファイルの中に全角文字があっても、プログラム識別番号域が正しい位置に補正されます。

## ボタン

### 参照(受信元ファイル名指定用)

グローバルサーバのファイル名とメンバを参照する「ファイル/メンバの参照」ダイアログボックスを表示します。

「受信」ダイアログのファイル名(受信元ファイル名指定用)への入力により、「ファイル/メンバの参照」ダイアログボックスの「ファイル一覧」は次のように表示されます。

- “指定名.%”を指定した場合、指定名で始まるファイルの一覧を表示します。
- ファイル名を省略した場合は、ユーザ識別名が指定名になります。
- 区分編成ファイルを指定すると、区分編成ファイルのメンバー一覧を表示します。
- ファイル名を引用符で囲まないで指定した場合、かつ、「グローバルサーバ連携情報」ダイアログボックスで“サーバファイル名付加文字列の付加”を指定していると、「サーバファイル名付加文字列」に指定した文字列がファイル名の先頭に付加されます。
- ファイル名を引用符で囲んで指定した場合、または、“サーバファイル名付加文字列の付加”を指定していない場合は、指定したファイル名が有効になります。

### 参照(受信先ファイル名指定用)

「フォルダの参照」ダイアログボックスを表示します。

## OK

ファイルを受信します。

## キャンセル

ファイルを受信しないでダイアログを閉じます。

## ヘルプ

ヘルプウィンドウを開きます。



### 注意

- 受信元が区分編成ファイルまたは区分編成ファイルのメンバの場合、区分編成ファイルのメンバ名が受信先のファイル名となります。また、[拡張子]が指定されていれば、メンバ名に拡張子を付加したものが受信先のファイル名となります。
- 受信元のファイル名またはメンバ名に“¥”が含まれている場合は、“¥”の代わりに“\$”を指定します。
- 受信元ファイルにGEMライブラリを指定することはできません。GEMライブラリのメンバをPCへ転送する必要がある場合には、PowerGEM Plusの[資産管理]を利用します。

## 27.2.5 ビルド用JCL/CLISTの生成

グローバルサーバに転送した資産を使用してロードモジュールを作成するための、JCLまたはCLISTファイルを作成します。制御文は、プロジェクトに登録しているCOBOLソースファイル名、翻訳オプションおよびファイルの送信情報を使用して生成します。

JCL/CLISTファイルを生成する手順を説明します。

1. プロジェクトのプロパティを設定します。

資産に登録しているプロジェクトを開き、[ファイル]メニューの“プロパティ”を選択して、[プロパティ]ダイアログを表示します。[基本設定]ページの“分散開発”で、“グローバルサーバ”を選択します。

2. 生成するための情報を設定します。

[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、“27.2.5.1 [ビルド用JCL/CLIST生成]ダイアログ”を表示します。ここでは生成するJCL/CLISTファイルのPC上でのファイル名、制御文の種別、ビルドを実行するシステム、リンクエディット処理の有無、アプリケーションの種類、および生成後の編集の有無を指定します。

3. JCL/CLISTファイルを生成します。

[OK]ボタンをクリックすると、雛型ファイルを生成します。生成に成功した場合、ツリーの[その他]フォルダにビルド制御文ファイルが登録されます。

ファイルは必要に応じて編集します。“27.2.5.1 [ビルド用JCL/CLIST生成]ダイアログ”で“生成後の編集”を指定した場合、自動的にエディタが起動して、ファイルを編集することができます。

生成されるファイルについては“27.2.5.2 JCL/CLISTファイル”を参照してください。

### 27.2.5.1 [ビルド用JCL/CLIST生成]ダイアログ

グローバルサーバで使用するビルド用JCL/CLISTの雛型ファイルを自動生成するためのダイアログです。

#### エディットボックス

##### ファイル名

生成するJCL/CLISTのファイル名を一つ入力します。

[種別]でJCLを選択した場合、拡張子は“.JCL”、CLISTならば“.CLIST”になります。入力したファイル名に拡張子がない場合や各種別で規定している拡張子でない場合は、自動的に拡張子を付加します。

#### ラジオボタン

##### JCL

JCLファイルを生成する場合に指定します。

##### CLIST

CLISTファイルを生成する場合に指定します。

##### MSP

グローバルサーバのシステムがMSPの場合に指定します。

##### XSP

グローバルサーバのシステムがXSPの場合に指定します。

#### チェックボックス

##### リンクエディット

JCL/CLISTにリンクエディット処理を展開する場合に指定します。

##### Interstage用のロードモジュールを生成する

AADアプリケーションを作成する場合に指定します。指定した場合、最終ターゲットファイル配下のファイルの登録状況によって、以下の制御文が生成されます。

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

AADアプリケーションのロードモジュールの詳細な作成方法については、“Interstageシステム開発手引書(AIM連携)”を参照してください。

##### 生成後の編集

JCL/CLISTファイルの生成後、自動的にエディタを起動して編集する場合に指定します。

#### ボタン

##### 参照

生成するJCL/CLISTのファイル名を選択するダイアログボックスを開きます。

##### OK

ファイルの生成を開始します。

##### キャンセル

ファイルを生成しないでダイアログを閉じます。

##### ヘルプ

ヘルプウィンドウを表示します。

## 27.2.5.2 JCL/CLISTファイル

ファイルに展開する情報と、生成後の編集について説明します。

### 展開する情報について

#### 翻訳対象ファイル

プロジェクトに登録しているファイルのうち、翻訳対象となるファイルは次のとおりです。

- ツリー上で[COBOLソースファイル]フォルダに登録されているCOBOLソースファイル。ただし、[プリコンパイラ]フォルダ、[INSDBINF]フォルダ、[AAD配付ソースファイル]フォルダが下位階層に定義されているものは対象になりません。
- [プリコンパイラ]フォルダの最下位の階層に登録されたファイル。
- [AAD配付ソースファイル]フォルダに登録された配付ソースファイル。

制御文では、プロジェクトに登録しているファイル名から、パス名および拡張子を除いたファイル名を使用します。そのため、ファイル名は区分編成ファイルのメンバ名として正しい名前でない限りなりません。

## ロードモジュール

ロードモジュールは、最終ターゲットファイルの単位で作成します。プロジェクトに複数の最終ターゲットファイルに登録している場合、すべてのロードモジュール生成制御文が、ひとつのJCL/CLISTファイルとして生成されます。制御文では、最終ターゲットファイル名からパス名および拡張子を除いたファイル名を、ロードモジュール名として使用します。そのため、ファイル名は区分編成ファイルのメンバ名として正しい名前でない限りなりません。

## ソース、登録集格納ファイル

雛型を生成する前に、“[27.2.4.1 \[送信\]ダイアログ](#)”でCOBOLソースファイルおよび登録集を送信した場合、ダイアログで設定した情報のうち、グローバルサーバのファイル名、ファイルパスワードおよびボリューム通し番号が制御文に展開されます。

## 翻訳オプション

プロジェクトに設定している翻訳オプションをグローバルサーバでの翻訳オプションに変換したものを制御文に展開します。設定した翻訳オプションのうち、グローバルサーバと互換を持つオプションが展開されます。

展開するオプションの一覧を以下に示します。

- CONF
- COPY
- DLOAD
- EQUALS
- FLAG
- FLAGSW
- LANGLVL
- LIB(注1)
- LINECOUNT
- LINESIZE
- LIST
- MESSAGE
- NCW
- NUMBER
- OBJECT(注2)
- OPTIMIZE
- QUOTE/APOST
- SDS
- SOURCE
- TRUNC
- XREF
- ZWB

注1：[送信]ダイアログを使用して登録集をグローバルサーバへ送信していない、かつ、翻訳オプション“LIB”が設定されている場合、登録集格納ファイル名が“SAMPLE.COBOL”で展開されます。

注2：翻訳オプション“NOOBJECT”が設定されている場合、オブジェクトモジュールの出力先の制御文と、リンクエディット制御文を展開しません。翻訳オプション“OBJECT”が設定されている場合、オブジェクトモジュールの出力先の制御文を展開しますが、翻訳オプションには“OBJECT”を生成しません。

## リンクオプション

プロジェクトにリンクオプションが設定してあっても、制御文には展開しません。

## AADアプリケーションのロードモジュール

AADアプリケーションのロードモジュールを作成する制御文は、以下のように展開されます。

- Interstageシステム提供のCOBOL登録集の指定“CORBA DD文(XSPの場合、CORBA FD文)”が追加されます。
- リンクエディット処理の生成を指定した場合、リンクオプションに“DYNAMIC”、“NCAL(XSPの場合はNOCALL)”が展開されます。
- スケルトンおよびオペレーション処理プログラムのロードモジュールを生成する場合、翻訳オプション“PSF”が追加されます。また、プロジェクトにIDLソースファイルを登録している場合、スケルトンのオブジェクトモジュールが初めに結合されます。なお、IDLソースファイルを登録していない場合は、スケルトンのオブジェクトモジュールが最初に結合されるように編集する必要があります。
- Interstageの出口プログラムのロードモジュールと、スタブのロードモジュールを生成する場合、翻訳オプション“PSF”および“NAME”が追加されます。

## 生成後の編集について

以下の情報は雛型生成後に編集する必要があります。

### SAMPLE.COBOL(登録集格納ファイル名)

グローバルサーバの登録集格納ファイル名です。登録集を、“27.2.4.1 [送信]ダイアログ”を使用して送信していない、かつ、翻訳オプションに“LIB”を指定している場合に展開されます。実際の登録集格納ファイル名に変更します。

### SAMPLE.COBOL(ソース格納ファイル名)

グローバルサーバのソース格納ファイル名です。ソースファイルを、“27.2.4.1 [送信]ダイアログ”を使用して送信していない場合に展開されます。実際のソース格納ファイル名に変更します。

### SAMPLE.OBJ

CLISTの場合、オブジェクトモジュールの出力先です。適切なファイル名に変更します。

### SAMPLE.LOAD

ロードモジュールの格納ファイル名です。適切なファイル名に変更します。

### ENTRY1

入口名です。実際の入口名に変更します。

以下に、生成される雛形を示します。なお、雛型の中に展開するファイル名やオプションなどの名前の長さによって、折り返し方が図と違う場合があります。

- MSPのJCL

```
//JOB1 JOB
//COMP1 EXEC PGM=JMNO00, REGION=2048K, ...[1]
// PARM=(翻訳オプション...) ...[2]
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=&&SYSUT1, UNIT=SYSDA, SPACE=(460, (700, 100))
//SYSUT2 DD DSN=&&SYSUT2, UNIT=SYSDA, SPACE=(460, (700, 100))
//SYSUT3 DD DSN=&&SYSUT3, UNIT=SYSDA, SPACE=(460, (700, 100))
//SYSUT4 DD DSN=&&SYSUT4, UNIT=SYSDA, SPACE=(460, (700, 100))
//SYSLIN DD DSN=&&LOADSET(ソースファイル名), UNIT=SYSDA, DISP=(NEW, PASS), ...[3]
// SPACE=(80, (500, 100, 30)), DCB=BLKSIZE=800
//CORBA DD DSN=A1M1. AADCPLIB, DISP=SHR ...[4]
```

```
//SYSLIB DD DSN=登録集格納ファイル名, UNIT=SYSDA, DISP=SHR, ...[5]
 PASSWORD=パスワード, VOL=SER=VOL通番
//SYSIN DD DSN=ソース格納ファイル名(ソースファイル名), UNIT=SYSDA, DISP=SHR, ...[6]
 PASSWORD=パスワード, VOL=SER=VOL通番

/*
//LKED1 EXEC PGM=JQAL, REGION=2048K, ...[7]
// PARM=(リンクエディットオプション...) ...[8]
//SYSLIB DD DSN=SYS1.COBLIB, DISP=SHR
//OBJ DD DSN=&&LOADSET, UNIT=SYSDA, DISP=(OLD, DELETE)
//SYSLMOD DD DSN=SAMPLE.LOAD, UNIT=SYSDA, DISP=OLD ...[9]
//SYSUT1 DD UNIT=SYSDA, SPACE=(1024, (50, 20))
//SYSPRINT DD SYSOUT=*
//SYSLIN DD * ...[10]
 INCLUDE OBJ(ソースファイル名)
 ENTRY ENTRY1 ...[11]
 NAME ロードモジュール名(R) ...[12]

/*
//
```

• XSPのJCL

```
¥ JOB JOB1
¥COMP1 EX COBOL, RSIZE=2048 ...[1]
¥ PARA 翻訳オプション... ...[2]
¥ FD U01=DA, VOL=WORK, BLK=(460*250, 100)
¥ FD U02=DA, VOL=WORK, BLK=(460*500, 100)
¥ FD U03=DA, VOL=WORK, BLK=(460*500, 100)
¥ FD U04=DA, VOL=WORK, BLK=(460*700, 100)
¥ FD LIST=DA, VOL=WORK, BLK=(255*900, 500), SOUT=A
¥ FD CORBA=DA, FILE=AIM1.AADCPLIB ...[4]
¥ FD RLIB=DA, VOL=WORK, FILE=(/, AD), BLK=(80*500, 100), C ...[3]
 MEMBER=ソースファイル名, DRTY=(5, BLK), FCB=BLKSIZE=3200, DISP=CONT
¥ FD MLIB=DA, FILE=登録集格納ファイル名, C ...[5]
 PSW=パスワード, VOL=VOL通番
¥ FD SLIB=DA, FILE=ソース格納ファイル名, MEMBER=ソースファイル名, C ...[6]
 PSW=パスワード, VOL=VOL通番

¥*
¥LKED EX LIED, RSIZE=512 ...[7]
¥ PARA リンクエディットオプション... ...[8]
¥ SW RLIB=RLIB, DISP=DLT ...[13]
¥ FD U01=DA, VOL=WORK, TRK=(10, 2)
¥ FD LIST=DA, VOL=WORK, BLK=(255*900, 100), SOUT=A
¥ FD ALIB=DA, FILE=C, ALIB
¥ FD ELIB=DA, FILE=SAMPLE.LOAD, DISP=LOCK ...[9]
¥ FD COIN=* ...[10]
 INCLUDE RLIB(ソースファイル名)
 NAME ロードモジュール名(R), ENT=ENTRY1 ...[11] [12]
¥ JEND
```

• MSPのCLIST

```
PROC 0
CONTROL LIST
CRTFILE SAMPLE.OBJ PARTITIONED(10) BLOCKS SPACE(500 100) -
 FORMAT(FIXED) RLENGTH(80) BLENGTH(3200)
ALLOC F(CORBA) DA(' AIM1.AADCPLIB') SHR ...[4]
COBOL ' ソース格納ファイル名(ファイル名)' - ...[1] [6]
 OBJECT(SAMPLE.OBJ(ファイル名)) - ...[3]
 LIB(' 登録集格納ファイル名') - ...[5]
 PRINT(*) -
 翻訳オプション... ...[2]
LINK (SAMPLE.OBJ(ファイル名)) - ...[7]
LOAD (SAMPLE.LOAD(ロードモジュール名)) - ...[9] [12]
```

```

PRINT(*) -
NOLIBDD -
COBLIB -
ENT(ENTRY1) - ...[11] (注)
リンクエディットオプション... ...[8]
FREE DA(SAMPLE.OBJ)
DLTFILE SAMPLE.OBJ
END

```

注：XSPの場合、次のようになります。

```

ENTRY(ENTRY1) - ...[11]
REP -

```

[1]

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

[2]

翻訳オプションです。

[3]

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

- MSP

```

//SYSLIN DD DSN=&&LOADSET,UNIT=SYSDA,DISP=(NEW,PASS),
// SPACE=(80,(500,100)),DCB=BLKSIZE=800

```

- XSP

```

¥ FD RLIB=DA,VOL=WORK,FILE=(/,AD),BLK=(80*500,100),FCB=BLKSIZE=3200,DISP=CONT

```

[4]

AADアプリケーション用のCOBOL登録集の定義です。“[27.2.5.1 \[ビルド用JCL/CLIST生成\]ダイアログ](#)”の、“Interstage用のロードモジュールを生成する”を指定した場合に生成します。

[5]

グローバルサーバ上の登録集格納ファイルの定義です。“[27.2.4.1 \[送信\]ダイアログ](#)”の[ファイルの種別]で“登録集”を選択したときに設定した送信先ファイル名、パスワードおよびVOL通番を使用します。登録集を送信していない場合は、生成しません。ただし、翻訳オプションに“LIB”を指定した場合、登録集格納ファイル名を“SAMPLE.COBOL”として生成します。

[6]

グローバルサーバ上のソース格納ファイルの定義です。“[27.2.4.1 \[送信\]ダイアログ](#)”の[ファイルの種別]で“ソース”を選択して設定した送信先ファイル名、パスワードおよびVOL通番を使用します。ソースを送信していない場合、ソース格納ファイル名を“SAMPLE.COBOL”として生成します。  
“[27.2.4.1 \[送信\]ダイアログ](#)”で、“パスワード”および“VOL通番”を指定していない場合、パスワードとVOL通番を省略します。プロジェクトに登録しているCOBOLソースの、パス名と拡張子を除いた部分を区分編成ファイルのメンバ名として使用します。

[7]

リンクエディット部分です。“[27.2.5.1 \[ビルド用JCL/CLIST生成\]ダイアログ](#)”で、“リンクエディット”を指定した場合に出力します。

[8]

AADアプリケーションを作成するためのリンクオプションです。“[27.2.5.1 \[ビルド用JCL/CLIST生成\]ダイアログ](#)”で、“Interstage用のロードモジュールを生成する”を指定した場合に生成します。

[9]

ロードモジュールの格納ファイル名です。

[10]

スタブまたは出口プログラムのJCLの場合、次のようになります。

- MSP : 生成しません。
- XSP : ¥ FD COIN=/,SW=RLIB

[11]

入口名です。“27.2.5.1 [ビルド用JCL/CLIST生成]ダイアログ”で、“Interstage用のロードモジュールを生成する”を指定した場合には、生成しません。

[12]

ロードモジュール名です。スタブまたは出口プログラムの場合、生成しません。

[13]

スタブまたは出口プログラムの場合、生成しません。

## 27.2.6 ターゲットビルド

グローバルサーバへ送信した資産をビルドします。

以下に、ビルドの手順について説明します。

1. 生成したJCL/CLISTファイルをグローバルサーバ上へ送信します。  
送信の方法については、“27.2.4 資産転送”を参照してください。
2. [プロジェクト]-[分散開発]メニューで“ターゲットビルド”を選択します。

[ジョブの起動(グローバルサーバ)]ダイアログが表示されます。詳細はPowerGEM Plus開発マネージャのヘルプ、“ジョブ起動の定義(グローバルサーバ)”を参照してください。



3. [起動形式]ページの各項目を設定します。  
[ファイル名]で、送信したJCL/CLISTファイル名を指定します。  
[起動形式]で、“JCL”または“CLIST”を指定します。

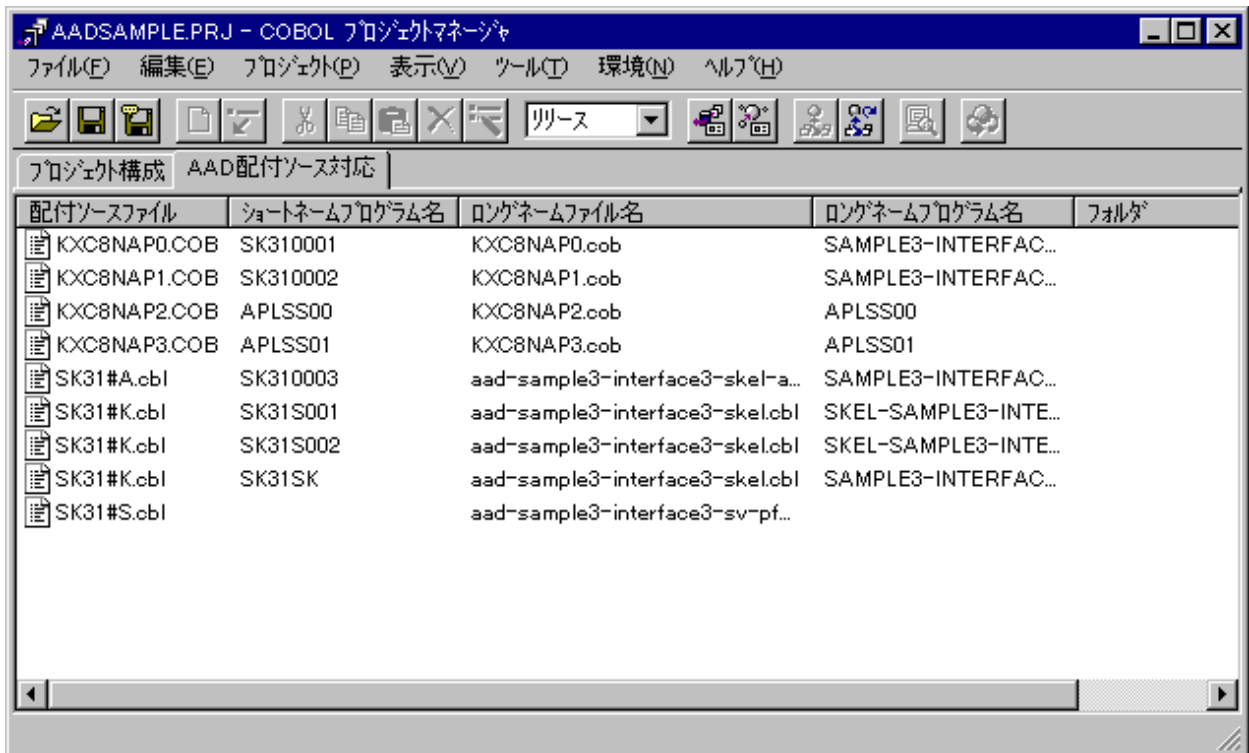
CLISTを使用する場合、[CLIST]ページで、[実行形式]の“同期”、および[結果表示]の“あり”を指定して[OK]ボタンをクリックすると、ビルド結果をエディタで確認することができます。



## 27.2.7 配付ソース対応表示

配付ソース生成前のロングネームと、生成後のショートネームの対応は、リストビューで表示することができます。

対応を表示するには、[表示]-[リスト表示内容]メニューの“AAD配付ソース対応”を選択します。



### 配付ソースファイル

配付ソースのファイル名です。

### ショートネームプログラム名

配付ソース生成で生成したプログラム名です。

### ロングネームファイル名

配付ソースの生成元のファイル名です。

### ロングネームプログラム名

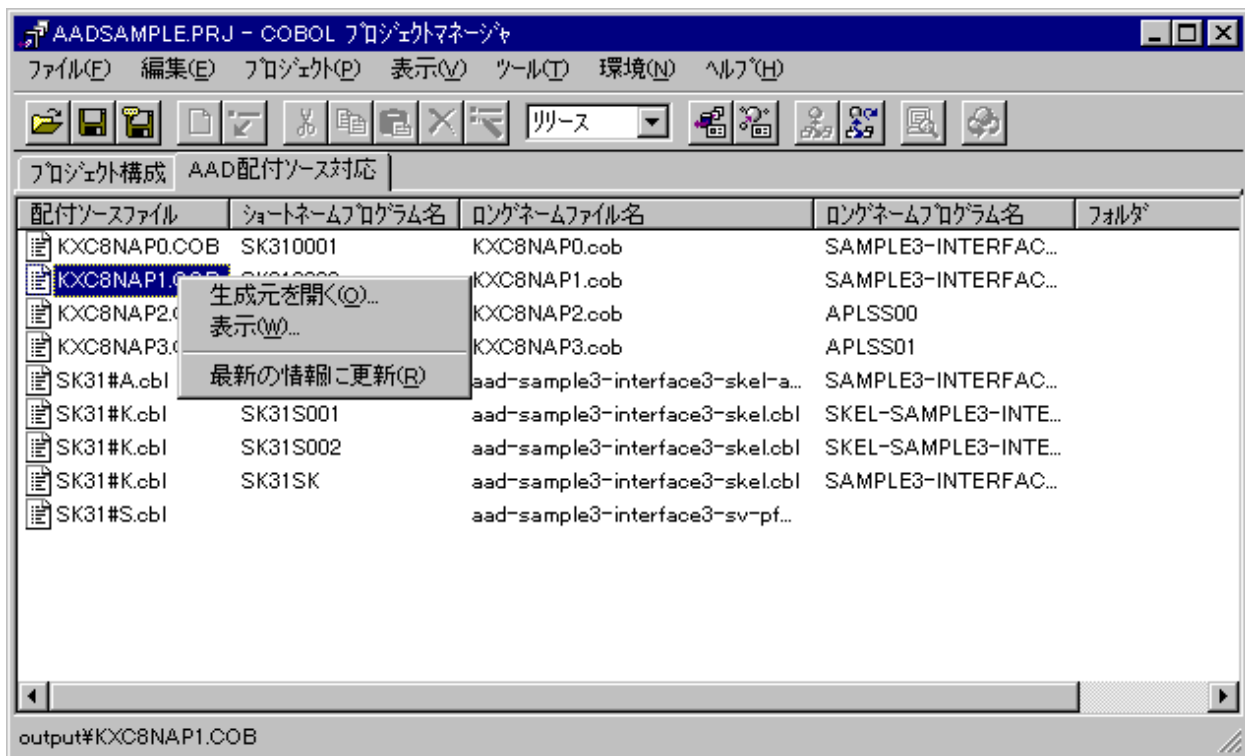
配付ソースの生成元のプログラム名です。

### フォルダ

生成元のファイルを格納しているフォルダです。ツリーに登録しているファイルにフォルダ名がある場合に表示します。

## ファイルの操作

ファイルを選択してマウスの右ボタンをクリックすると、メニューが表示されます。表示されたメニューを選択してファイル进行操作することができます。



配付ソースの生成元のファイルを開く場合、“生成元を開く”メニューを選択します。生成元のファイルがCOBOLソースファイルまたは登録集の場合、編集画面が開きます。それ以外のファイルの場合、表示状態で開きます。

ファイルをダブルクリック、またはファイルを選択して[Enter]キーを押しても同様です。

配付ソースを表示状態で開く場合、[表示]メニューを選択します。

## 27.3 IDL-COBOL変換機能

OSIV系システムのAIM応用プログラムをCORBA環境の応用プログラムとして利用可能にする製品に Interstage/ AIMApplicationDirector(以下、AADと呼びます)があります。

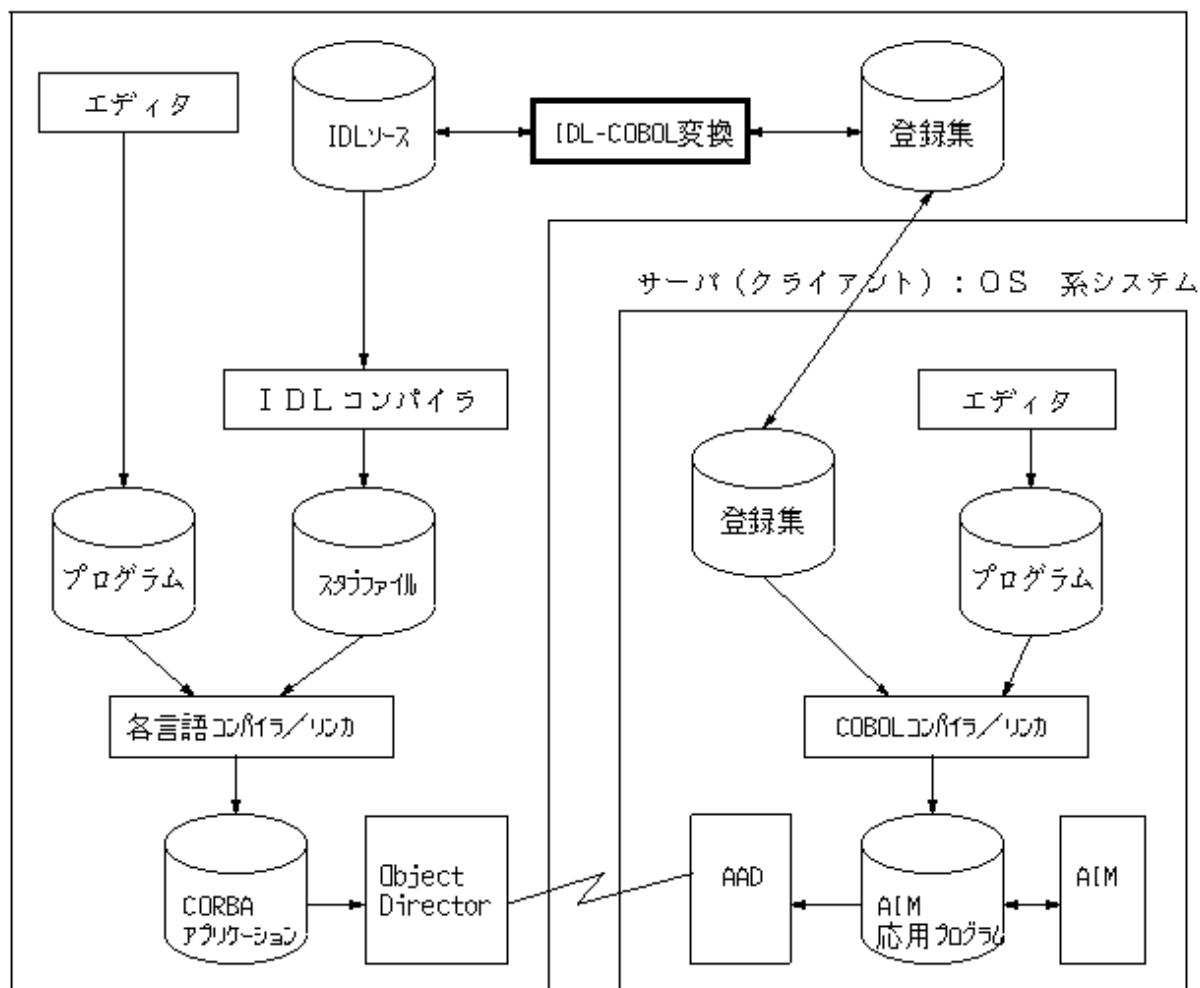
プロジェクトマネージャでは、以下の2つの機能によって、AADを利用して動作するAIM応用プログラムの開発を支援します。

- ・ IDLソースファイルから、AIM応用プログラムが使用するCOBOL登録集を生成する機能(IDL→COBOL変換機能)

- COBOL登録集から、IDLソースファイルを生成する機能(COBOL→IDL変換機能)

図27.3 OSIV系システム連携

クライアント (サーバ) : Windows



IDL-COBOL変換機能は、[ツール]メニューの“IDL-COBOL変換”から起動します。IDL-COBOL変換の詳細な使用方法については、ヘルプを参照してください。

# 付録A 翻訳オプション

ここでは、翻訳オプションについて説明します。

指定する翻訳オプションがわからないときには、“[A.1 翻訳オプション一覧](#)”から翻訳オプションを確認し、“[A.2 翻訳オプションの指定形式](#)”を参照してください。

## A.1 翻訳オプション一覧

### 翻訳時の資源に関するもの

- “[A.2.1 AIMLIB](#)(サブスキーマ定義ファイルのフォルダの指定)”
- “[A.2.14 FILEEXT](#)(ファイル定義体ファイルの拡張子の指定)”
- “[A.2.15 FILELIB](#)(ファイル定義体ファイルのフォルダの指定)”
- “[A.2.18 FORMEXT](#)(画面帳票定義体ファイルの拡張子の指定)”
- “[A.2.19 FORMLIB](#)(画面帳票定義体ファイルのフォルダの指定)”
- “[A.2.23 LIB](#)(登録集ファイルのフォルダの指定)”
- “[A.2.24 LIBEXT](#)(登録集ファイルの拡張子の指定)”
- “[A.2.41 REP](#)(リポジトリファイルの入出力先フォルダの指定)”
- “[A.2.42 REPIN](#)(リポジトリファイルの入力先フォルダの指定)”

### 翻訳リストに関するもの

- “[A.2.7 COPY](#)(登録集原文の表示)”
- “[A.2.20 GEN](#)(FCOMおよびUWAの表示)”
- “[A.2.25 LINECOUNT](#)(翻訳リストの1ページあたりの行数)”
- “[A.2.26 LINESIZE](#)(翻訳リストの1行あたりの文字数)”
- “[A.2.27 LIST](#)(目的プログラムリストの出力の可否)”
- “[A.2.29 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の出力の可否)”
- “[A.2.30 MESSAGE](#)(オプション情報リスト、翻訳単位統計情報リストの出力の可否)”
- “[A.2.35 NUMBER](#)(ソースプログラムの一連番号領域の指定)”
- “[A.2.38 PRINT](#)(各種翻訳リストの出力の可否および出力先の指定)”
- “[A.2.49 SOURCE](#)(ソースプログラムリストの出力の可否)”
- “[A.2.60 XREF](#)(相互参照リストの出力の可否)”

### 翻訳時メッセージに関するもの

- “[A.2.6 CONF](#)(規格の違いによるメッセージの出力の可否)”
- “[A.2.16 FLAG](#)(診断メッセージのレベル)”
- “[A.2.17 FLAGSW](#)(COBOL文法の言語要素に対しての指摘メッセージ表示の可否)”

### COBOLプログラムの解釈に関するもの

- “[A.2.2 ALPHAL](#)(英小文字の扱い)”
- “[A.2.4 BINARY](#)(2進項目の扱い)”
- “[A.2.10 CURRENCY](#)(通貨編集用文字の扱い)”

- “A.2.12 DUPCHAR(重複文字の扱い)”
- “A.2.22 LANGLVL(ANSI COBOL規格の指定)”
- “A.2.21 INITVALUE(作業場所節でのVALUE句なし項目の扱い)”
- “A.2.33 NCW(日本語利用者語の文字集合の指定)”
- “A.2.34 NSPCOMP(日本語空白の比較方法の指定)”
- “A.2.39 QUOTE/APOST(表意定数QUOTEの扱い)”
- “A.2.43 RSV(予約語の種類)”
- “A.2.45 SCS(ソースファイルのコード系)”
- “A.2.46 SDS(符号付き10進項目の符号の整形の可否)”
- “A.2.47 SHREXT(マルチスレッドプログラムの外部属性に関する扱い)”
- “A.2.50 SQLGRP(SQLのホスト変数定義の拡張)”
- “A.2.51 SRF(正書法の種類)”
- “A.2.54 STD1(英数字の文字の大小順序の指定)”
- “A.2.55 TAB(タブの扱い)”
- “A.2.61 ZWB(符号付き外部10進項目と英数字項目の比較)”

#### ソースプログラムの解析に関するもの

- “A.2.44 SAI(ソース解析情報ファイルの出力の可否)”

#### 目的プログラムの作成に関するもの

- “A.2.3 ASCOMP5(2進項目の解釈の指定)”
- “A.2.9 CREATE(創成ファイルの指定)”
- “A.2.11 DLOAD(プログラム構造の指定)”
- “A.2.28 MAIN(主プログラム/副プログラムの指定)”
- “A.2.31 MODE(ACCEPT文の動作の指定)”
- “A.2.32 NAME(翻訳単位ごとのオブジェクトファイルの出力の可否)”
- “A.2.36 OBJECT(目的プログラムの出力の可否)”
- “A.2.37 OPTIMIZE(広域最適化の扱い)”
- “A.2.40 RCS(実行時コード系の指定)”
- “A.2.57 THREAD(マルチスレッドプログラム作成の指定)”

#### 実行時の処理に関するもの

- “A.2.13 EQUALS(SORT文での同一キーデータの処理方法)”
- “A.2.59 TRUNC(桁落とし処理の可否)”

#### 実行時の資源に関するもの

- “A.2.48 SMSIZE(PowerSORTが使用するメモリ容量を指定)”
- “A.2.52 SSIN(ACCEPT文のデータの入力先)”
- “A.2.53 SSOUT(DISPLAY文のデータの出力先)”

## 実行時のデバッグ機能に関するもの

- ・ “A.2.5 CHECK (CHECK機能の使用の可否)”
- ・ “A.2.8 COUNT (COUNT機能の使用の可否)”
- ・ “A.2.56 TEST (対話型デバッガおよび診断機能の使用の可否)”
- ・ “A.2.58 TRACE (TRACE機能の使用の可否)”

## A.2 翻訳オプションの指定形式

---

### 翻訳オプションの指定方法と優先順位

翻訳オプションの指定方法には、以下の4種類があります。

1. オプションファイルによる指定
2. -WCオプションによる指定
3. -WCオプション以外のコマンドオプションによる指定



#### 参照

“3.8.1 オプション一覧”

4. ソースプログラム中の翻訳指示文(@OPTIONS)による指定



#### 参照

“2.1.4 翻訳指示文”

オプションの優先順位は、4.>3.>2.>1.となります。また、オプションファイルおよび-WCオプションの指定方法には複数あり、どの指定が有効になるかは、以下の規則に従います。

### オプションファイル

オプションファイルの指定には、以下の方法があります。

1. 環境変数COB\_OPTIONSの-iオプションによる指定



#### 参照

“1.2.1 環境変数の設定”

2. プロジェクトマネージャの[翻訳オプション]ダイアログ、またはNetCOBOL Studioの[翻訳オプション]ページによる指定
3. 翻訳コマンドの-iオプションによる指定



#### 参照

“3.8.1.13 -i (オプションファイルの指定)”

プロジェクトマネージャの[翻訳オプション]ダイアログでオプションを指定する場合、指定されたオプションはプロジェクトマネージャによってオプションファイルに格納され、ビルド時に自動的に取り込まれます。

指定の強さは3.>2.>1.となります。指定されたオプションファイルのうち、指定の強さが最も強いものが有効になります。また、翻訳コマンドで複数のオプションファイルが指定された場合、最後に指定されたものが有効になります。



## 例

- c:¥default.cbiが有効になり、f:¥default.cbiは無効になります。

```
set COB_OPTIONS=-if:¥default.cbi
cobol32 -ic:¥default.cbi a.cob
```

- f:¥default.cbiが有効になり、c:¥default.cbiは無効になります。

```
cobol32 -ic:¥default.cbi -if:¥default.cbi a.cob
```

## -WCオプション

-WCオプションの指定には、以下の方法があります。

1. 環境変数COB\_OPTIONSによる指定



### 参照

“1.2.1 環境変数の設定”

2. 翻訳コマンドによる指定



### 参照

“3.8.1.19 -WC (翻訳オプションの指定)”

指定の強さは、2.>1.です。

-WCオプションは複数指定できます。指定されたすべての翻訳オプションが有効になります。

ただし、同じ種類の翻訳オプションが複数指定された場合、指定の強さが強く、最後に指定された翻訳オプションが有効になります。



## 例

- 翻訳オプションMESSAGEと翻訳オプションNOOPTIMIZEが有効になります。

```
set COB_OPTIONS=-WC, "MESSAGE, OPTIMIZE"
cobol32 -WC, "NOOPTIMIZE " a.cob
```

- 翻訳オプションMESSAGEとCHECK(NUMERIC)が有効になります。

```
cobol32 -WC, "CHECK (ALL)" -WC, "MESSAGE" -WC, "CHECK (NUMERIC)" a.cob
```

## 翻訳オプションごとの指定方法

翻訳オプションの内容によっては、指定方法が限られる場合があります。以下のマークを参考にして指定してください。

### WINCOB

プロジェクトマネージャの[翻訳オプション]ダイアログ、またはNetCOBOL Studioの[翻訳オプション]ページで指定できます。

### -WC

コマンドオプション-WCで指定できます。

### @

翻訳指示文で指定できます。

(F)

この方法で指定する場合、フォルダ名は指定できません。

上記以外に、“(@)”と表示されている場合は、ソースファイルに複数の翻訳単位が記述されていない場合だけ、翻訳指示文での指定が可能になります。

## フォルダ名およびファイル名の指定方法

翻訳オプションに指定するフォルダ名およびファイル名は、絶対パス名または相対パス名で指定します。以下の文字を含む場合、フォルダ名またはファイル名を二重引用符(”)で囲む必要があります。

空白、「+」、「,」、「;」、「=」、「[」、「]」、「(」、「)」、「'」



注意

フォルダ名を指定した翻訳オプションは、-WCオプションに指定することはできません。

指定可能なフォルダ名の長さは、分離符であるセミコロンを含めて2048バイトまでです。

## カレントフォルダ

翻訳時のカレントフォルダは以下のようになります。

### WINCOBIによる単数翻訳の場合

翻訳対象となるソースファイルが格納されているフォルダをカレントフォルダとします。

### WINCOBIによる複数翻訳の場合

先頭に指定されているソースファイルが格納されているフォルダをカレントフォルダとします。

### プロジェクト管理機能を使用する場合

プロジェクトファイルが格納されているフォルダをカレントフォルダとします。



注意

実行時のカレントフォルダは、アプリケーションの実行環境や動作内容に依存します。

## A.2.1 AIMLIB(サブスキーマ定義ファイルのフォルダの指定)

WINCOB

AIMLIB( フォルダ[; フォルダ ] ... )

サブスキーマ段落にサブスキーマ定義ファイルを記述した場合、サブスキーマ定義ファイルのフォルダを指定します。

使用するサブスキーマ定義ファイルが複数のフォルダに存在する場合、フォルダをセミコロンで区切って複数指定します。フォルダを複数指定した場合、指定された順序でフォルダが検索されます。



参考

オプションの指定が重複した場合は、以下の順に検索します。

1. -Aオプション
2. 翻訳オプションAIMLIB



3. 環境変数COB\_AIMLIB



参照

“1.2.1 環境変数の設定”、“3.8.1.1 -A(サブスキーマ定義ファイルのフォルダの指定)”

## A.2.2 ALPHAL(英小文字の扱い)

WINCOB, -WC, @

$$\left\{ \begin{array}{l} \text{ALPHAL}[( \left\{ \begin{array}{l} \text{ALL} \\ \text{WORD} \end{array} \right\} )] \\ \text{NOALPHAL} \end{array} \right\}$$

ソースプログラム中の半角英小文字を半角英大文字と等価に扱う(ALPHAL)か、扱わない(NOALPHAL)か、を指定します。

COBOLの語については、“COBOL文法書”の“1.2.2 COBOLの語”を参照してください。

|              |                                                                                                 |
|--------------|-------------------------------------------------------------------------------------------------|
| ALPHAL(ALL)  | COBOLの語は、英小文字と英大文字が等価に扱われます。また、プログラム名定数、CALL文、CANCEL文、ENTRY文およびINVOKE文の定数中の英小文字も英大文字と等価に扱われます。  |
| ALPHAL(WORD) | COBOLの語は、英小文字と英大文字が等価に扱われます。プログラム名定数、CALL文、CANCEL文、ENTRY文およびINVOKE文の定数を含む、定数中の英小文字は英大文字と区別されます。 |
| NOALPHAL     | COBOLの語および定数中の英小文字は、英大文字と区別されます。                                                                |



参考

“10.3.4 プログラムの翻訳”

## A.2.3 ASCOMP5(2進項目の解釈の指定)

WINCOB, -WC, @

$$\text{ASCOMP5}( \left\{ \begin{array}{l} \text{NONE} \\ \text{ALL} \\ \text{BINARY} \\ \text{COMP} \end{array} \right\} )$$

2進項目の解釈を指定します。

|                 |                                                                                 |
|-----------------|---------------------------------------------------------------------------------|
| ASCOMP5(NONE)   | 宣言されたとおりに解釈します。                                                                 |
| ASCOMP5(ALL)    | USAGE BINARYおよびUSAGE COMP、USAGE COMPUTATIONALと宣言された項目はUSAGE COMP-5が指定されたとみなします。 |
| ASCOMP5(BINARY) | USAGE BINARYと宣言された項目はUSAGE COMP-5が指定されたとみなします。                                  |

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| ASCOMP5(COMP) | USAGE COMP、USAGE COMPUTATIONALと宣言された項目はUSAGE COMP-5が指定されたとみなします。 |
|---------------|------------------------------------------------------------------|

## 注意

- データの内部表現が変わるため、内部表現を意識したコーディングが含まれている場合には注意してください。
- PowerRW+を使用する場合、NONE以外は指定しないでください。

## A.2.4 BINARY(2進項目の扱い)

WINCOB, -WC, @

BINARY( { WORD[, { MLBON  
MLBOFF } 1 } )  
BYTE

2進データの基本項目が、桁数より求められるワード単位の領域長(2,4,8)に割り付けられる(BINARY(WORD))か、バイト単位の領域長(1~8)に割り付けられる(BINARY(BYTE))か、を指定します。なお、符号なし2進項目の最左端ビットの扱いも指定できます。

|                     |           |
|---------------------|-----------|
| BINARY(WORD,MLBON)  | 最左端ビットは符号 |
| BINARY(WORD,MLBOFF) | 最左端ビットは数値 |

## 注意

BINARY(BYTE)を指定した場合、最左端ビットは数値として扱われます。

## 参考

宣言した桁数と、割り当てられる領域長の関係は、下表のとおりです。

| PICの桁数  |         | 割り当てられる領域長   |              |
|---------|---------|--------------|--------------|
| 符号付き    | 符号なし    | BINARY(BYTE) | BINARY(WORD) |
| 1 ~ 2   | 1 ~ 2   | 1            | 2            |
| 3 ~ 4   | 3 ~ 4   | 2            | 2            |
| 5 ~ 6   | 5 ~ 7   | 3            | 4            |
| 7 ~ 9   | 8 ~ 9   | 4            | 4            |
| 10 ~ 11 | 10 ~ 12 | 5            | 8            |
| 12 ~ 14 | 13 ~ 14 | 6            | 8            |
| 15 ~ 16 | 15 ~ 16 | 7            | 8            |
| 17 ~ 18 | 17 ~ 18 | 8            | 8            |

## A.2.5 CHECK(CHECK機能の使用の可否)

WINCOB, -WC, @

```
{ CHECK[([n] [,ALL] [,BOUND] [,ICONF] [,LINKAGE] [,NUMERIC] [,PRM])) }
{ NOCHECK }
```

CHECK機能を使用する(CHECK)か、しない(NOCHECK)か、を指定します。

nには、メッセージを表示させる回数を0～999999の整数で指定します。省略した場合には、1が指定されたとみなします。

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CHECK(ALL)     | BOUND、ICONF、LINKAGE、NUMERICおよびPRMの検査を行います。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| CHECK(BOUND)   | 添字・指標および部分参照の範囲外検査を行います。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| CHECK(ICONF)   | 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を行います。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| CHECK(LINKAGE) | 以下のいずれかの場合に実行時メッセージを出力します。 <ul style="list-style-type: none"><li>・ CALL文の呼び出し側と呼ばれる側でリンケージ規約が合っていない場合</li><li>・ 呼ばれる側がSTDCALLリンケージで、パラメタの個数が合っていない場合</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| CHECK(NUMERIC) | 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)の検査を行います。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| CHECK(PRM)     | <p>翻訳時に、内部プログラムを呼び出すCALL文(CALL一意名を除く)のUSING指定またはRETURNING指定に記述されたデータ項目と内部プログラムのUSING指定またはRETURNING指定に記述されたデータ項目に対して以下の検査を行います。</p> <ul style="list-style-type: none"><li>・ USING指定のパラメタの個数の一致</li><li>・ RETURNING指定のパラメタの有無の一致</li><li>・ データ項目がオブジェクト参照以外の場合、データ項目の長さの一致(長さの検査は、翻訳時に長さが決定する場合のみ行う)</li><li>・ データ項目がオブジェクト参照の場合、USAGE OBJECT REFERENCE句に指定されたクラス名、FACTORY指定およびONLY指定の一致</li></ul> <p>実行時に、外部プログラムを呼び出すCALL文のUSING指定またはRETURNING指定に記述されたデータ項目と外部プログラムのUSING指定またはRETURNING指定に記述されたデータ項目に対して以下の検査を行います。</p> <ul style="list-style-type: none"><li>・ USING指定のパラメタの個数の一致、およびデータ項目の長さの一致(ただしUSING指定のパラメタの個数の不一致が4個以上の場合、誤りが検出されないことがあります)</li><li>・ RETURNING指定のパラメタの長さの一致(RETURNING指定がない場合、暗黙にPROGRAM-STATUSが受け渡されるため、長さ4バイトのデータ項目が指定されたものとみなします)</li></ul> <p>なお、実行時に長さが決定する場合は、翻訳時に記述した長さの最大値を使って、検査を行います。</p> |



### 注意

- ・ CHECK機能使用時には、n回目のメッセージが出力されるまで、プログラムの処理が続行されますが、領域破壊などによりプログラムが期待どおり動作しない場合があります。なお、nに0を指定した場合には、メッセージの表示回数に関係なく、プログラムの処理が続行されます。
- ・ CHECKを指定すると、上記の検査をするための処理が目的プログラム中に組み込まれるため、実行性能が低下します。デバッグ終了時には、NOCHECKを指定して再翻訳してください。

- ON SIZE ERROR指定またはNOT ON SIZE ERROR指定の算術文では、ON SIZE ERRORの除数のゼロ検査が行われ、CHECK(NUMERIC)の除数のゼロ検査は行われません。
- CHECK(NUMERIC)のデータ例外検査は、外部10進項目または内部10進項目が参照で 사용되는場合、および、英数字項目または集団項目から、外部10進項目または内部10進項目へ転記される場合に行われます。ただし次は、チェックの対象とはなりません。
  - 添字としてALLが指定されている表要素
  - SEARCH ALL文におけるキー項目(ただしキー項目に対する添字が1次元かつWHEN条件がひとつのみである場合は除く)
  - SORT/MERGE文におけるキー項目
  - SQL文中で使用されているホスト変数
  - CALL文、INVOKE文および行内呼び出しのBY REFERENCEパラメタ
  - 次の組み込み関数の引数
    - FUNCTION ADDR
    - FUNCTION LENG
    - FUNCTION LENGTH
  - 英数字項目または集団項目から、外部10進項目または内部10進項目のオブジェクトプロパティへの転記



## 参考

“19.2 CHECK機能の使い方”

## A.2.6 CONF(規格の違いによるメッセージの出力の可否)

WINCOB, -WC, @

$$\left\{ \begin{array}{l} \text{CONF} \left( \left\{ \begin{array}{l} 68 \\ 74 \\ \text{OBS} \end{array} \right\} \right) \\ \text{NOCONF} \end{array} \right\}$$

COBOLの旧規格と新規格の間の非互換を指摘させる(CONF)か、させない(NOCONF)か、を指定します。CONFを指定すると、非互換項目は、Iレベルの診断メッセージで指摘されます。

|           |                                                 |
|-----------|-------------------------------------------------|
| CONF(68)  | '68 ANS COBOLと'85 ANS COBOLとで意味の解釈が異なる項目を指摘します。 |
| CONF(74)  | '74 ANS COBOLと'85 ANS COBOLとで意味の解釈が異なる項目を指摘します。 |
| CONF(OBS) | 廃要素である言語仕様および機能を指摘します。                          |



## 注意

翻訳オプションCONF(68)および翻訳オプションCONF(74)は、翻訳オプションLANGVL(85)を指定した場合だけ意味を持ちます。



## 参照

“A.2.22 LANGLVL (ANSI COBOL規格の指定)”



## 参考

CONFは、従来の規格に従って作成したプログラムを、'85 ANS COBOLの規格に従うように変更する場合に有効です。

## A.2.7 COPY (登録集原文の表示)

WINCOB, -WC, @

{ COPY  
NOCOPY }

ソースプログラムリスト内に、COPY文によって組み込まれる登録集原文を表示する(COPY)か、しない(NOCOPY)か、を指定します。



## 参照

“A.2.49 SOURCE (ソースプログラムリストの出力の可否)”



## 注意

COPYは、翻訳オプションSOURCEを指定した場合だけ意味を持ちます。

## A.2.8 COUNT (COUNT機能の使用の可否)

WINCOB, -WC, @

{ COUNT  
NOCOUNT }

COUNT機能を使用する(COUNT)か、使用しない(NOCOUNT)か、を指定します。



## 注意

- COUNTを指定すると、COUNT情報を出力するための処理が目的プログラム中に組み込まれるため、実行性能が低下します。デバッグ終了時には、NOCOUNTを指定して再翻訳してください。
- COUNTは、翻訳オプションTRACEまたは-Drオプションと同時に指定できません。同時に指定された場合、後に指定されたオプションが有効となります。



参考

“19.4 COUNT機能の使い方”

## A.2.9 CREATE(創成ファイルの指定)

WINCOB, -WC, @

$$\text{CREATE} [ ( \left\{ \begin{array}{c} \text{OBJ} \\ \text{REP} \end{array} \right\} ) ]$$

オブジェクトの生成を目的に翻訳する(CREATE(OBJ))か、リポジトリの生成を目的に翻訳する(CREATE(REP))か、を指定します。  
CREATE(REP)が指定された場合、手続き部の解析は行われず、目的プログラムは生成されません。



注意

CREATE(REP)指定は、クラス定義の翻訳でだけ意味を持ちます。クラス定義以外の翻訳では、つねにCREATE(OBJ)とみなされます。

## A.2.10 CURRENCY(通貨編集用文字の扱い)

WINCOB, -WC, @

$$\text{CURRENCY} ( \left\{ \begin{array}{c} \text{¥} \\ \$ \end{array} \right\} )$$

通貨編集用文字として使用している文字に、¥を使用する(CURRENCY(¥))か、\$を使用する(CURRENCY(\$))か、を指定します。

## A.2.11 DLOAD(プログラム構造の指定)

WINCOB, -WC, @

$$\left\{ \begin{array}{c} \text{DLOAD} \\ \text{NODLOAD} \end{array} \right\}$$

プログラム構造を動的プログラム構造にする(DLOAD)か、しない(NODLOAD)か、を指定します。



参考

“4.3 プログラム構造”

“10.1.5 動的プログラム構造”

## A.2.12 DUPCHAR(重複文字の扱い)

WINCOB, -WC, @

DUPCHAR ( { STD  
EXT } )

ソースファイルがUnicodeの場合、コンパイラが付加／置換する以下のJIS非漢字の負号を、拡張文字(DUPCHAR(EXT))とするか、標準文字(DUPCHAR(STD))とするか、を指定します。

- ・ 3バイト項目制御部を指定した帳票定義体
- ・ COPY文の書き方2と3の日本語利用者語

表A.1 JIS非漢字の負号の扱い

| オプションの指定     | 文字コード(UTF-8)  |
|--------------|---------------|
| DUPCHAR(EXT) | ー (X"EFBC8D") |
| DUPCHAR(STD) | - (X"E28892") |



### 参考

“M.4.2 JIS非漢字の負号について”

## A.2.13 EQUALS(SORT文での同一キーデータの処理方法)

WINCOB, -WC, @

{ EQUALS  
NOEQUALS }

実行時に、SORT文の入力中に同一キーを持つレコードが複数個存在する場合、それらに関して、SORT文の出力レコードの順序をSORT文の入力レコードの順序と同じにすることを保証する(EQUALS)か、しない(NOEQUALS)か、を指定します。

NOEQUALSを指定すると、SORT文で同一キーを持つレコードの出力順序は規定されません。



### 注意

EQUALSを指定すると、整列操作で入力順序を保証するための特別な処理が行われるために実行性能が低下します。

## A.2.14 FILEEXT(ファイル定義体ファイルの拡張子の指定)

WINCOB

FILEEXT( 拡張子[, 拡張子] ... )

---

ファイル定義体ファイルの拡張子を指定します。拡張子には、任意の文字列を指定することができます。ファイル名に拡張子がない場合は、拡張子の代わりに、文字列“None”を指定します。

拡張子を複数指定した場合、指定された順序で検索が行われます。

#### 参考

---

- ・ オプションの指定が重複した場合、優先順位は以下のとおりです。
    1. 翻訳オプションFILEEXT(複数指定の場合は指定された順)
    2. 環境変数FFD\_SUFFIX(複数指定の場合は指定された順)
  - ・ 翻訳オプションFILEEXTおよび環境変数FFD\_SUFFIXの指定がない場合、拡張子(FFD)が検索されます。
- 

#### 参照

---

“1.2.1 環境変数の設定”

---

## A.2.15 FILELIB(ファイル定義体ファイルのフォルダの指定)

---

WINCOB

---

FILELIB( フォルダ[, フォルダ] ... )

---

IN/OF XFDLIB指定のCOPY文によりファイル定義体からレコード定義を取り込む場合、ファイル定義体ファイルのフォルダを指定します。

使用するファイル定義体ファイルが複数のフォルダに存在する場合、フォルダをセミコロンで区切って複数指定します。フォルダを複数指定した場合、指定された順序でフォルダが検索されます。

#### 参考

---

オプションの指定が重複した場合、検索は以下の順に行われます。

1. -fオプション
  2. 翻訳オプションFILELIB
  3. 環境変数FILELIB
  4. カレントフォルダ
- 

#### 参照

---

“1.2.1 環境変数の設定”、“3.8.1.11 -f(ファイル定義体ファイルのフォルダの指定)”

---

## A.2.16 FLAG(診断メッセージのレベル)

---

WINCOB, -WC, @



---


$$\text{FLAG}(\left\{ \begin{array}{c} \text{I} \\ \text{W} \\ \text{E} \end{array} \right\})$$


---

表示する診断メッセージを指定します。

| オプション   | 表示する診断メッセージ            |
|---------|------------------------|
| FLAG(I) | すべての診断メッセージを表示します。     |
| FLAG(W) | Wレベル以上の診断メッセージだけ表示します。 |
| FLAG(E) | Eレベル以上の診断メッセージだけ表示します。 |

### 注意

翻訳オプションCONFによる指摘メッセージは、FLAGの指定に関係なく表示されます。

### 参照

“A.2.6 CONF(規格の違いによるメッセージの出力の可否)”

## A.2.17 FLAGSW(COBOL文法の言語要素に対しての指摘メッセージ表示の可否)

WINCOB, -WC, @

---


$$\left\{ \begin{array}{l} \text{FLAGSW}(\left\{ \begin{array}{l} \left[ \left\{ \begin{array}{c} \text{STDM} \\ \text{STDI} \\ \text{STDH} \end{array} \right\} \right] [\text{RPW}] \\ \text{SIA} \\ \text{GWS} \\ \text{GSS} \end{array} \right\}) \\ \text{NOFLAGSW} \end{array} \right\}$$


---

COBOL文法の言語要素に対しての指摘メッセージを表示する(FLAGSW)か、しない(NOFLAGSW)か、を指定します。

以下に指定する言語要素を示します。

|              |                                          |
|--------------|------------------------------------------|
| FLAGSW(STDM) | '85 ANS COBOL規格の下位レベル外                   |
| FLAGSW(STDI) | '85 ANS COBOL規格の中位レベル外                   |
| FLAGSW(STDH) | '85 ANS COBOL規格の上位レベル外                   |
| FLAGSW(RPW)  | '85 ANS COBOL規格の報告書                      |
| FLAGSW(SIA)  | 富士通システム統合アーキテクチャ(SIA)の範囲外                |
| FLAGSW(GSW)  | GS-INTERSTAGEの使用を前提にしたホストアプリケーションの分散開発支援 |



## 参考

FLAGSW(SIA)は、他システムで動かすプログラムを作成するときに有効です。



## 注意

FLAGSWのサブオペランド{STDM|STDI|STDH}とRPWは、両方を同時に省略することはできません。

## A.2.18 FORMEXT(画面帳票定義体ファイルの拡張子の指定)

WINCOB

FORMEXT( 拡張子[, 拡張子 ] ... )

画面帳票定義体ファイルの拡張子を指定します。拡張子には、任意の文字列を指定することができます。ファイル名に拡張子がない場合は、拡張子の代わりに、文字列“None”を指定します。

拡張子を複数指定した場合、指定された順序で検索が行われます。



## 参考

- ・ オプションの指定が重複した場合、優先順位は以下のとおりです。
  1. 翻訳オプションFORMEXT(複数指定の場合は指定された順)
  2. 環境変数SMED\_SUFFIX(複数指定の場合は指定された順)
- ・ 翻訳オプションFORMEXTおよび環境変数SMED\_SUFFIXの指定がない場合、検索は以下の順に行われます。PXDは、「帳票クラスライブラリ」で使用される専用定義体の拡張子です。
  1. 拡張子(PMD)
  2. 拡張子(PXD)
  3. 拡張子(SMD)



## 参照

“1.2.1 環境変数の設定”

## A.2.19 FORMLIB(画面帳票定義体ファイルのフォルダの指定)

WINCOB

FORMLIB( フォルダ[; フォルダ ] ... )

IN/OF XMDLIB指定のCOPY文により画面帳票定義体からレコード定義を取り込む場合、画面帳票定義体ファイルのフォルダを指定します。

使用する画面帳票定義体ファイルが複数のフォルダに存在する場合、フォルダをセミコロンで区切って複数指定します。フォルダを複数指定した場合、指定された順序でフォルダが検索されます。



## 参考

オプションの指定が重複した場合、検索は以下の順に行われます。

1. -mオプション
2. 翻訳オプションFORMLIB
3. 環境変数FORMLIB
4. カレントフォルダ



## 参照

“1.2.1 環境変数の設定”

“3.8.1.15 -m(画面帳票定義体ファイルのフォルダの指定)”

## A.2.20 GEN(FCOMおよびUWAの表示)

WINCOB, -WC, @

|   |              |   |
|---|--------------|---|
| { | GEN          | } |
|   | <u>NOGEN</u> |   |

ソースプログラムリスト中に、AIM DBMSとの連絡領域(FCOM)およびAIM展開レコード域(UWA)を表示する(GEN)か、しない(NOGEN)か、を指定します。

## A.2.21 INITVALUE(作業場所節でのVALUE句なし項目の扱い)

WINCOB, -WC, @

|   |                    |   |
|---|--------------------|---|
| { | INITVALUE(xx)      | } |
|   | <u>NOINITVALUE</u> |   |

作業場所節データのVALUE句なし項目を指定値で初期化する(INITVALUE)か、しない(NOINITVALUE)か、を指定します。



## 注意

xxは、2桁の16進数を指定してください。xxは省略できません。

## A.2.22 LANGLVL(ANSI COBOL規格の指定)

WINCOB, -WC, @

---

LANGLVL(     $\left\{ \begin{array}{c} 85 \\ 74 \\ 68 \end{array} \right\}$     )

---

COBOLの旧規格と新規格との間で、ソースプログラムの解釈が異なる項目に対してどの規格に基づいて解釈するかを指定します。

| オプション       | 規格            |
|-------------|---------------|
| LANGLVL(85) | '85 ANS COBOL |
| LANGLVL(74) | '74 ANS COBOL |
| LANGLVL(68) | '68 ANS COBOL |

### A.2.23 LIB(登録集ファイルのフォルダの指定)

---

WINCOB

---

LIB( フォルダ[; フォルダ ] ... )

---

登録集機能(COPY文)を使用する場合、登録集ファイルのフォルダを指定します。使用する登録集ファイルが複数のフォルダに存在する場合、フォルダをセミコロンで区切って複数指定します。フォルダを複数指定した場合、指定された順序でフォルダが検索されます。



#### 参考

オプションの指定が重複した場合、検索は以下の順に行われます。

1. -Iオプション
  2. 翻訳オプションLIB
  3. 環境変数COB\_COBCOPY
  4. カレントフォルダ
- 



#### 参照

“1.2.1 環境変数の設定”

“3.8.1.12 -I(登録集ファイルのフォルダの指定)”

---

### A.2.24 LIBEXT(登録集ファイルの拡張子の指定)

---

WINCOB

---

LIBEXT( 拡張子[, 拡張子 ] ... )

---

登録集ファイルの拡張子を指定します。拡張子には、任意の文字列を指定することができます。ファイル名に拡張子がない場合は、拡張子の代わりに、文字列“None”を指定します。

拡張子を複数指定した場合、指定された順序で検索が行われます。

## 参考

- ・ オプションの指定が重複した場合、検索は以下の順に行われます。
  1. 翻訳オプションLIBEXT(複数指定の場合は指定された順序)
  2. 環境変数COB\_LIBSUFFIX(複数指定の場合は指定された順序)
- ・ 翻訳オプションLIBEXTおよび環境変数COB\_LIBSUFFIXの指定がない場合、検索は以下の順に行われます。
  1. 拡張子CBL
  2. 拡張子COB
  3. 拡張子COBOL

## 参照

“1.2.1 環境変数の設定”

## 注意

登録集ファイルの拡張子に“None”を利用することはできません。

### A.2.25 LINECOUNT(翻訳リストの1ページあたりの行数)

WINCOB, -WC, @

LINECOUNT(n)

翻訳リストの1ページあたりの行数を指定します。  
nは、3桁以内の整数を指定してください。  
本オプションを指定しなかった場合、LINECOUNT(0)が指定されたものとみなします。

## 注意

0から12までの値を指定すると、ページ替えのない出力となります。

### A.2.26 LINESIZE(翻訳リストの1行あたりの文字数)

WINCOB, -WC, @

LINESIZE(n)

翻訳リストの1行あたりの最大文字数(リスト上に表示されるA/N文字換算の値)を指定します。  
nは、0、80または120以上の3桁の整数を指定することができます。  
0を指定した場合、行の途中で改行せずにソースプログラムリストを出力します。  
本オプションを指定しなかった場合、LINESIZE(0)が指定されたものとみなします。

 注意

- ・ オプション情報リスト、診断メッセージリストおよび翻訳単位統計情報リストは、翻訳オプションLINESIZEに指定した最大文字数に関係なく固定の文字数(120)で出力されます。
- ・ 文字数として有効な最大の値は136です。翻訳オプションLINESIZEに136より大きい値を指定した場合、136として扱われます。

A.2.27 LIST(目的プログラムリストの出力の可否)

WINCOB, -WC, @

$$\left\{ \begin{array}{l} \text{LIST} \\ \text{NOLIST} \end{array} \right\}$$

目的プログラムリストを出力する(LIST)か、しない(NOLIST)か、を指定します。  
目的プログラムリストは、-Pオプションまたは翻訳オプションPRINTによって、翻訳リストの出力を有効にしている場合だけ出力されます。  
翻訳リストの出力については、“3.8.1.6 -dp(翻訳リストファイルのフォルダの指定)”、“3.8.1.17 -P(各種翻訳リストの出力および出力先の指定)”および“A.2.38 PRINT(各種翻訳リストの出力の可否および出力先の指定)”を参照してください。

A.2.28 MAIN(主プログラム/副プログラムの指定)

WINCOB, -WC, @

$$\left\{ \begin{array}{l} \text{MAIN}[( \left\{ \begin{array}{l} \text{WINMAIN} \\ \text{MAIN} \end{array} \right\} )] \\ \text{NOMAIN} \end{array} \right\}$$

COBOLソースプログラムが主プログラム(MAIN)か、副プログラム(NOMAIN)か、を指定します。  
MAINだけが指定された場合は、MAIN(WINMAIN)が指定されたものとみなします。

|               |                                                                                     |
|---------------|-------------------------------------------------------------------------------------|
| 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場合に指定します。 |
| 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)を使用する場合に指定します。      |

 注意

- ・ 主プログラムとなるCOBOLソースプログラムにMAINオプションを指定してください。
  - ー WINCOBの複数翻訳から翻訳する場合は、[ウィンドウ]ボタンまたは[コンソール]ボタンで指定してください。

- ー プロジェクトから翻訳する場合は、[プロジェクト]メニューから[オプション]-[主プログラム]を選択し、[ウィンドウ]または[コンソール]を指定してください。
- ー NetCOBOL Studioから翻訳する場合は、[依存]または[構造]ビューから主プログラムにするCOBOLプログラムを選択し、[ウィンドウ]または[コンソール]を指定してください。

いずれの場合も、[ウィンドウ]はMAIN(WINMAIN)、[コンソール]はMAIN(MAIN)の指定と同じ意味になります。

- ・ 通常は、MAIN(WINMAIN)を指定するようにしてください。システムのコンソールを指定する必要がある場合だけ、MAIN(MAIN)を指定するようにしてください。
- ・ 翻訳指示文(@OPTIONS)で指定されたMAINオプションは、翻訳指示文直後の翻訳単位にだけ有効となります。



## 参考

主プログラムが他言語のとき、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を使用したい場合、実行時の環境変数で指定します。



## 参照

“5.4 実行環境情報の指定形式”

## A.2.29 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の出力の可否)

WINCOB, -WC, @

```
{ MAP
 NOMAP }
```

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翻訳リストに出力する(MAP)か、しない(NOMAP)か、を指定します。

これらのリストは、-Pオプションまたは翻訳オプションPRINTによって、翻訳リストの出力を有効にしている場合だけ出力されます。

翻訳リストの出力については、“3.8.1.6 -dp(翻訳リストファイルのフォルダの指定)”、“3.8.1.17 -P(各種翻訳リストの出力および出力先の指定)”および“A.2.38 PRINT(各種翻訳リストの出力の可否および出力先の指定)”を参照してください。

## A.2.30 MESSAGE(オプション情報リスト、翻訳単位統計情報リストの出力の可否)

WINCOB, -WC, @

```
{ MESSAGE
 NOMESSAGE }
```

オプション情報リストおよび翻訳単位統計情報リストを出力する(MESSAGE)か、しない(NOMESSAGE)か、を指定します。

## A.2.31 MODE (ACCEPT文の動作の指定)

WINCOB, -WC, @

$$\text{MODE} \left( \begin{Bmatrix} \text{STD} \\ \text{CCVS} \end{Bmatrix} \right)$$

ACCEPT文の“ACCEPT 一意名 [FROM 呼び名]”の書き方で、受取り側項目に数字項目を指定したACCEPT文を実行する場合、受取り側項目に右詰めの数字転記を行う(MODE(STD))か、左詰めの文字転記を行う(MODE(CCVS))か、を指定します。



MODE(CCVS)を指定する場合、数字項目としては、外部10進項目だけがACCEPT文の受取り側項目として指定できます。

## A.2.32 NAME (翻訳単位ごとのオブジェクトファイルの出力の可否)

WINCOB, -WC, @

$$\begin{Bmatrix} \text{NAME} \\ \text{NONAME} \end{Bmatrix}$$

複数の翻訳単位(プログラム定義、クラス定義または分離されたメソッド定義)が記述された1つのソースファイルを翻訳する場合、翻訳単位ごとにオブジェクトファイルを出力する(NAME)か、1つにまとめて出力する(NONAME)か、を指定します。



- NONAMEオプションを指定して作成したオブジェクトファイルは、リンクコマンドで利用することはできません。
- プロジェクト管理機能では、NAMEオプションを使用することはできません。



- WINCOBでNAMEオプションを指定する場合は、その他の翻訳オプションのエディットボックスに直接“NAME”と入力してください。
- NAMEオプションを指定した場合、各オブジェクトファイルの名前は、“各翻訳単位のプログラムID名(クラスIDまたはメソッドID).OBJ”となります。

## A.2.33 NCW (日本語利用者語の文字集合の指定)

WINCOB, -WC, @



NCW ( { STD  
SYS } )

---

利用者語に指定できる日本語文字集合をシステム共通な日本語文字集合とする(NCW(STD))か、計算機の日本語文字集合とする(NCW(SYS))か、を指定します。

STDを指定すると、次の日本語文字集合が日本語利用者語として利用できます。

- ・ JIS第一水準
- ・ JIS第二水準
- ・ JIS非漢字(以下の文字)
  - 0、1、…、9
  - A、B、…、Z
  - a、b、…、z
  - あ、あ、い、い、…、ん
  - ア、ア、イ、イ、…、ン、ヴ、カ、ケ
  - ー (長音)、- (ハイフン)、- (負号)、々

SYSを指定すると、次の日本語文字集合が日本語利用者語として使用できます。

- ・ STD指定の文字集合
- ・ 拡張文字
- ・ 拡張非漢字
- ・ 利用者定義文字
- ・ JIS非漢字(以下の文字は使用不可)
  - 、 。 , . ・ : ; ? ! `
  - ° ` ^ \_ \_ / \ | ( )
  - [ ] { } 「 」 + = < >
  - ¥ \$ ¢ £ % # & \* @

## A.2.34 NSPCOMP(日本語空白の比較方法の指定)

---

WINCOB, -WC, @

---

NSPCOMP ( { NSP  
ASP } )

---

後述する比較において、日本語空白の扱いを、日本語空白として扱う(NSPCOMP(NSP))か、ANK空白とみなす(NSPCOMP(ASP))か、を指定します。日本語空白をANK空白とみなす場合には、日本語空白は2バイトのANK空白と等価に扱われます。

NSPCOMP(ASP)オプションは、以下の比較に対して有効となります。

- ・ 日本語項目を作用対象とする日本語文字比較
- ・ 集団項目を作用対象とする文字比較

以下の比較に対しては無効です。

- ・ 日本語項目を含まない集団項目どうしの比較
- ・ 明または暗に属性が表示用でない項目を含む集団項目の比較

## 注意

- ・ INSPECT文、STRING文、UNSTRING文および索引ファイルのキー操作で行われる文字比較および日本語文字比較では、NSPCOMP(ASP)オプションを指定しても日本語空白はANK空白と等価に扱われません。
- ・ NSPCOMP(ASP)が指定された場合、字類条件JAPANESEでANK空白が日本語として扱われます。

## 参考

OSIV系システムのコード系では、日本語空白がANK空白の2文字分と同じ値を持っていますが、Windowsのコード系では、同じ値を持たないため、OSIV系システムで使用していたCOBOLプログラムを本システムで動作させる場合には、通常、ソースプログラムの修正が必要となります。

翻訳オプションNSPCOMP(ASP)を指定することにより、比較対象の空白が等価に処理されるため、前述の条件に合う比較は、ソースプログラムを修正しなくてもOSIV系システムと同じ動作が期待できます。

ただし、空白を等価に扱う処理は、格納されている文字データから判断するため、作用対象が集団項目のとき、従属する項目属性に合わせた処理はしません。このため、例えば、従属する英数字項目中に文字以外のデータが設定されていた場合などは誤動作する可能性がありますので、注意が必要です。

## 参照

“M.4.1 日本語空白と英数字空白の文字コード”

## A.2.35 NUMBER(ソースプログラムの一連番号領域の指定)

WINCOB, -WC, @

$\left\{ \begin{array}{l} \text{NUMBER} \\ \text{NONNUMBER} \end{array} \right\}$

翻訳時および実行時の各種リストで、ソースプログラム中の各行を識別するための行情報の行番号に、ソースプログラムの一連番号領域の値を使用する(NUMBER)か、コンパイラが生成した値を使用する(NONNUMBER)か、を指定します。

|           |                                                                                                                               |
|-----------|-------------------------------------------------------------------------------------------------------------------------------|
| NUMBER    | 一連番号領域に数字以外の文字が含まれている場合および一連番号が昇順になっていない場合、その行の行番号は、直前の正しい一連番号に1を加えた値に変更されます。ただし、一連番号が降順となる場合、一意の補正された番号がCOPY修飾値と同じ形式で付加されます。 |
| NONNUMBER | 行番号は、1から1きざみに昇順に与えられます。                                                                                                       |

## 注意

- ・ NUMBERが指定されているときには、同一の一連番号が連続していても誤りではないものとみなされます。
- ・ NUMBERを指定した場合、ビルダのエラージャンプ機能は使用できません。

- NUMBERを指定した場合、翻訳オプションSRFにFREEは指定できません。翻訳オプションSRFにFREEを指定した場合、プログラムの動作は保証されません。
- プリコンパイラ連携時にNUMBERを指定できません。NONNUMBERを有効にして翻訳してください。



## 参照

“A.2.51 SRF(正書法の種類)”

## A.2.36 OBJECT(目的プログラムの出力の可否)

WINCOB, -WC (F), @ (F)

```
{ OBJECT [(WINCOBのみフォルダ指定可)]
 NOOBJECT }
```

目的プログラムを出力するか(OBJECT)、しないか(NOOBJECT)を指定します。

目的プログラムは、ソースプログラムと同じフォルダに格納されます。

WINCOBでは、格納先のフォルダを変更することができます。変更したい場合は、フォルダ名を指定してください。



## 参考

“3.2 翻訳に必要な資源”

## A.2.37 OPTIMIZE(広域最適化の扱い)

WINCOB, -WC, @

```
{ OPTIMIZE
 NOOPTIMIZE }
```

広域最適化された目的プログラムを作成する(OPTIMIZE)か、しない(NOOPTIMIZE)か、を指定します。



## 注意

TESTと同時に指定した場合、作成されるデバッグ情報ファイルは診断機能では使用できますが、対話型デバッガでは使用できません。



## 参考

“付録C 広域最適化”

“A.2.56 TEST(対話型デバッガおよび診断機能の使用の可否)”

## A.2.38 PRINT(各種翻訳リストの出力の可否および出力先の指定)

WINCOB

PRINT[( フォルダ )]

翻訳リストを出力する場合に指定します。

翻訳リストを出力する場合、通常、ソースプログラムと同じフォルダに格納されます。変更したい場合には、格納先を指定してください。

## A.2.39 QUOTE/APOST(表意定数QUOTEの扱い)

WINCOB, -WC, @

$$\left\{ \begin{array}{l} \underline{\text{QUOTE}} \\ \text{APOST} \end{array} \right\}$$

表意定数QUOTEおよびQUOTESとしてクォーテーションマーク(”)を使う(QUOTE)か、アポストロフィ(')を使う(APOST)か、を指定します。



### 注意

ソースプログラム中の引用符は、このオプションの指定に関係なく、クォーテーションマークとアポストロフィのどちらでも使用できます。ただし、左側の引用符と右側の引用符は、同じでなくてはなりません。

## A.2.40 RCS(実行時コード系の指定)

WINCOB, -WC, @

$$\text{RCS} \left( \left\{ \begin{array}{l} \underline{\text{SJIS}} \\ \left\{ \begin{array}{l} \text{UTF16} \\ \text{UCS2} \end{array} \right\} \end{array} \right\} \left[ , \left\{ \begin{array}{l} \underline{\text{LE}} \\ \text{BE} \end{array} \right\} \right] \right\} \right)$$

実行時のコード系をシフトJISにする(RCS(SJIS))か、Unicodeにする(RCS(UTF16)またはRCS(UCS2))か、を指定します。

RCS(UTF16,LE) または RCS(UCS2,LE)

Unicode環境でのエンディアンをリトルエンディアンとします。

RCS(UTF16,BE) または RCS(UCS2,BE)

Unicode環境でのエンディアンをビッグエンディアンとします。

RCS(UTF16)とRCS(UCS2)は同義ですが、RCS(UTF16)を使用することをおすすめします。



## 注意

実行時のコード系でUnicodeを指定した場合、分散開発支援機能を利用することはできません。



## 参考

“第25章 Unicode”

## A.2.41 REP(リポジットリファイルの入出力先フォルダの指定)

WINCOB

REP( フォルダ )

リポジットリファイルは、通常、ソースプログラムと同じフォルダに格納されます。

変更したい場合は、フォルダ名を指定してください。

指定されたフォルダは、リポジットリファイルの入力先フォルダとしても使用されます。



## 参考

オプションの指定が重複した場合、入力先フォルダは以下の順に検索されます。

1. -Rオプション
2. 翻訳オプションREPIN
3. 環境変数COB\_REPIN
4. -drオプション
5. 翻訳オプションREP
6. カレントフォルダ



## 参照

“1.2.1 環境変数の設定”

“3.8.1.8 -dr(リポジットリファイルの入出力先フォルダの指定)”

“3.8.1.18 -R(リポジットリファイルの入力先フォルダの指定)”

“A.2.42 REPIN(リポジットリファイルの入力先フォルダの指定)”

## A.2.42 REPIN(リポジットリファイルの入力先フォルダの指定)

WINCOB

REPIN( フォルダ[; フォルダ ] ... )

リポジットファイルの入力先フォルダを指定します。リポジットファイルが複数のフォルダに存在する場合、フォルダをセミコロンで区切って複数指定します。フォルダを複数指定した場合、指定された順番でフォルダが検索されます。



参考

オプションの指定が重複した場合、以下の順に検索されます。

- 1. -Rオプション
- 2. 翻訳オプションREPIN
- 3. 環境変数COB\_REPIN
- 4. -drオプション
- 5. 翻訳オプションREP
- 6. カレントフォルダ



参照

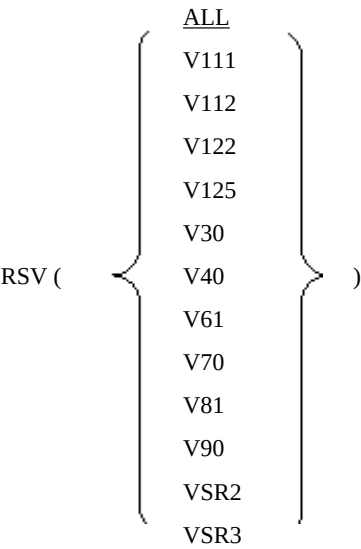
“3.8.1.8 -dr(リポジットファイルの入出力先フォルダの指定)”

“3.8.1.18 -R(リポジットファイルの入力先フォルダの指定)”

“A.2.41 REP(リポジットファイルの入出力先フォルダの指定)”

A.2.43 RSV(予約語の種類)

WINCOB, -WC, @



予約語の種類を指定します。

|           |                      |
|-----------|----------------------|
| RSV(ALL)  | 本製品用                 |
| RSV(V111) | OSIV COBOL85 V11L11用 |
| RSV(V112) | OSIV COBOL85 V11L20用 |

|           |                      |
|-----------|----------------------|
| RSV(V122) | OSIV COBOL85 V12L20用 |
| RSV(V125) | COBOL85 V12L50用      |
| RSV(V30)  | COBOL85 V30用         |
| RSV(V40)  | COBOL97 V40用         |
| RSV(V61)  | COBOL97 V61用         |
| RSV(V70)  | NetCOBOL V7.0用       |
| RSV(V81)  | NetCOBOL V8.0用       |
| RSV(V90)  | NetCOBOL V9.0用       |
| RSV(VSR2) | VS COBOLII REL2.0用   |
| RSV(VSR3) | VS COBOLII REL3.0用   |

## A.2.44 SAI(ソース解析情報ファイルの出力の可否)

WINCOB, -WC (F) , @ (F)

---

$$\left\{ \begin{array}{l} \text{SAI [(WINCOBのみフォルダ指定可)]} \\ \text{NOSAI} \end{array} \right\}$$

---

ソース解析情報ファイルを出力する(SAI)か、出力しない(NOSAI)か、を指定します。

ソース解析情報ファイルは、ソースプログラムと同じフォルダに格納されます。

WINCOBでは、格納先のフォルダを変更することができます。変更したい場合は、フォルダ名を指定してください。



参考

“3.2 翻訳に必要な資源”

---

## A.2.45 SCS(ソースファイルのコード系)

WINCOB, -WC

---

$$\text{SCS} \left( \left\{ \begin{array}{l} \text{SJIS} \\ \text{UTF8} \end{array} \right\} \right)$$

---

COBOLソースファイルおよび登録集ファイルのコード系が、シフトJISである(SCS(SJIS))か、UTF-8である(SCS(UTF8))か、を指定します。



注意

SCS(UTF8)を指定した場合は、RCS(UTF16)オプションも指定してください。

---



## 参照

“A.2.40 RCS (実行時コード系の指定)”

### A.2.46 SDS (符号付き10進項目の符号の整形の可否)

WINCOB, -WC, @

---

|   |            |   |
|---|------------|---|
| { | <u>SDS</u> | } |
| { | NOSDS      | } |

---

符号付き内部10進項目から符号付き内部10進項目への転記で、送出し側項目の符号をそのまま転記する(SDS)か、整形された符号を転記する(NOSDS)か、を指定します。

負号にはX‘B’およびX‘D’の2種類があり、その他は正号として扱われます。ここでいう整形された符号とは、送出し側項目の符号が正ならばX‘C’に、負ならばX‘D’に変換することです。

### A.2.47 SHREXT (マルチスレッドプログラムの外部属性に関する扱い)

WINCOB, -WC, @

---

|   |                 |   |
|---|-----------------|---|
| { | SHREXT          | } |
| { | <u>NOSHREXT</u> | } |

---

マルチスレッドとなるオブジェクト形式の場合(THREAD(MULTI)指定)に、外部属性(EXTERNAL指定)のデータおよびファイルをスレッド間で共有する(SHREXT)か、共有しない(NOSHREXT)か、を指定します。



## 注意

オブジェクト形式がシングルスレッド(THREAD(SINGLE))の場合、確定翻訳オプションにはSHREXTと表示されますが、NOSHREXTとして翻訳が行われます。

### A.2.48 SMSIZE (PowerSORTが使用するメモリ容量を指定)

WINCOB, -WC, @

---

SMSIZE(値K)

---

PowerSORTが使用するメモリ容量をキロバイト単位の数字で指定します。





## 注意

SORT文およびMERGE文から呼び出されるPowerSORTが使用するメモリ空間の容量を限定したい場合に指定します。指定する値は、キロバイト単位の数字です。指定された値を、PowerSORTのBSORT\_PRIME.memory\_sizeに設定します。指定された値が実際に有効になるかについては、PowerSORTの”オンラインマニュアル”をお読みください。

このオプションは、実行時オプションsmsizeおよび特殊レジスタSORT-CORE-SIZEに指定する値の意味と等価です。同時に指定された場合の優先順位は、以下のとおりです。

1. 特殊レジスタSORT-CORE-SIZE
2. 実行時オプションsmsize
3. 翻訳オプションSMSIZE()



## 例

```
特殊レジスタ MOVE 102400 TO SORT-CORE-SIZE
(102400=100キロです)
翻訳オプション SMSIZE (500K)
実行時オプション smsize300k
```

この場合、一番強い特殊レジスタSORT-CORE-SIZEの値 100キロバイトを優先します。



## 注意

本オプションは、別製品PowerSORTをインストールしている場合だけ有効になります。

## A.2.49 SOURCE(ソースプログラムリストの出力の可否)

WINCOB, -WC, @

```
{ SOURCE
 NOSOURCE }
```

ソースプログラムリストを出力する(SOURCE)か、しない(NOSOURCE)か、を指定します。

ソースプログラムリストは、-Pオプションまたは翻訳オプションPRINTによって、翻訳リストの出力を有効にしている場合だけ出力されます。

翻訳リストの出力については、“[3.8.1.6 -dp\(翻訳リストファイルのフォルダの指定\)](#)”、“[3.8.1.17 -P\(各種翻訳リストの出力および出力先の指定\)](#)”および“[A.2.38 PRINT\(各種翻訳リストの出力の可否および出力先の指定\)](#)”を参照してください。

## A.2.50 SQLGRP(SQLのホスト変数定義の拡張)

WINCOB, -WC, @

```
{ SQLGRP
 NOSQLGRP }
```

SQLのホスト変数の定義方法を拡張する(SQLGRP)か、しない(NOSQLGRP)か、を指定します。  
なお、NOSQLGRPを指定した場合、ホスト変数にREDEFINES句を指定することはできません。

### 注意

SQLGRP指定時は、データ項目がホスト変数として正しい形式を持つかどうかをデータ記述項だけでは決定できません。このため、ホスト変数定義に対する翻訳時の構文チェックに、以下の違いがあります。

- ・ 埋込みSQL文で参照しないホスト変数は、構文チェックの対象になりません。ただし、大域属性を持つ集団項目のホスト変数名についてのチェックは行われます。
- ・ 可変長文字列型のホスト変数の構文チェックが厳しくなりました。レベル番号49の項目を従属する集団項目は、可変長文字列型のホスト変数とみなします。

集団項目としてホスト変数を定義する機能を使用せず、従来どおりの構文チェックを行う場合には、NOSQLGRPを指定してください。

### 参照

“22.2.4 高度なデータ操作”

## A.2.51 SRF(正書法の種類)

WINCOB, -WC, @

$$\text{SRF}(\left\{ \begin{array}{c} \text{FIX} \\ \text{FREE} \\ \text{VAR} \end{array} \right\} [, \left\{ \begin{array}{c} \text{FIX} \\ \text{FREE} \\ \text{VAR} \end{array} \right\} ])$$

COBOLソースプログラムおよび登録集ファイルの正書法の種類を、固定形式にする(FIX)か、自由形式にする(FREE)か、可変形式にする(VAR)か、を指定します。

正書法の種類は、COBOLソースプログラム、登録集の順に指定します。

登録集の指定を省略した場合、COBOLソースプログラムに指定した正書法の種類となります。

### 注意

翻訳オプションSRFにFREEを指定した場合、翻訳オプションNUMBERは指定できません。翻訳オプションNUMBERを指定した場合、プログラムの動作は保証されません。

### 参考

ソースファイルと登録集ファイルの正書法が同じ場合、登録集ファイルの正書法の指定は省略することができます。

## A.2.52 SSIN(ACCEPT文のデータの入力先)

WINCOB, -WC, @

---

SSIN (      {    環境変数情報名    } )

---

小入出力機能のACCEPT文のデータの入力先を指定します。

|               |                                                    |
|---------------|----------------------------------------------------|
| SSIN(環境変数情報名) | データの入力先としてファイルを使用します。環境変数情報名には、実行時にファイルのパス名を設定します。 |
| SSIN(SYSIN)   | データの入力先としてコンソールウィンドウを使用します。                        |



#### 注意

環境変数情報名は英大文字(A～Z)で始まる8文字以内の英大文字および数字でなければなりません。また、環境変数情報名は、他のファイルで使用する環境変数情報名(ファイル識別名)と一致しないようにする必要があります。



#### 参考

“11.1 小入出力機能”

## A.2.53 SSOUT(DISPLAY文のデータの出力先)

WINCOB, -WC, @

---

SSOUT (      {    環境変数情報名    } )

---

小入出力機能のDISPLAY文のデータの出力先を指定します。

|                |                                                    |
|----------------|----------------------------------------------------|
| SSOUT(環境変数情報名) | データの出力先としてファイルを使用します。環境変数情報名には、実行時にファイルのパス名を設定します。 |
| SSOUT(SYSOUT)  | データの出力先としてコンソールウィンドウを使用します。                        |



#### 注意

環境変数情報名は英大文字(A～Z)で始まる8文字以内の英大文字または数字でなければなりません。また、環境変数情報名は、他のファイルで使用する環境変数情報名(ファイル識別名)と一致しないようにする必要があります。



#### 参考

“11.1 小入出力機能”

## A.2.54 STD1(英数字の文字の大小順序の指定)

WINCOB, -WC, @

$$\text{STD1}(\left\{ \begin{array}{c} \text{ASCII} \\ \text{JIS1} \\ \text{JIS2} \end{array} \right\})$$

ALPHABET句のEBCDIC指定で、英数字のコード(1バイト文字の標準コード)をASCII(ASCII)として取り扱うか、JIS8単位コード(JIS1)として取り扱うか、またはJIS7単位ローマ字コード(JIS2)として取り扱うか、を指定します。



注意

ALPHABET句でEBCDIC指定を記述した場合、このオプションの指定に応じて、以下に示す文字符号系を採用します。

|             |               |
|-------------|---------------|
| STD1(ASCII) | EBCDIC(ASCII) |
| STD1(JIS1)  | EBCDIC(カナ)    |
| STD1(JIS2)  | EBCDIC(英小)    |

## A.2.55 TAB(タブの扱い)

WINCOB, -WC, @

$$\text{TAB}(\left\{ \begin{array}{c} \underline{8} \\ 4 \end{array} \right\})$$

タブの扱いを4カラム単位にする(TAB(4))か、8カラム単位にする(TAB(8))か、を指定します。ただし、値としてのタブは、タブ値そのものです。

## A.2.56 TEST(対話型デバグおよび診断機能の使用の可否)

WINCOB, -WC (F), @ (F)

$$\left\{ \begin{array}{l} \text{TEST [(WINCOBのみフォルダ指定可)]} \\ \text{NOTEST} \end{array} \right\}$$

対話型デバグおよび診断機能の使用を可能にする(TEST)か、しない(NOTEST)か、を指定します。

TESTを指定すると、対話型デバグや診断機能で使用するデバグ情報ファイルが作成され、ソースプログラムと同じフォルダに格納されます。

WINCOBでは、格納先のフォルダを変更することができます。変更したい場合は、フォルダ名を指定してください。

また、リンク時(プロジェクト管理機能を除く)には、対話型デバッガや診断機能を使用するためのリンクオプションも設定する必要があります。

### 注意

OPTIMIZEと同時に指定した場合、作成されたデバッグ情報ファイルは診断機能では使用できますが、デバッガでは使用できません。

### 参照

“A.2.37 OPTIMIZE (広域最適化の扱い)”

### 参考

“3.2 翻訳に必要な資源”

“19.6 診断機能の使い方”

“第20章 対話型デバッガの使い方”

“20.2.3 プログラムのリンク”

## A.2.57 THREAD (マルチスレッドプログラム作成の指定)

WINCOB, -WC, @

$$\text{THREAD} ( \left\{ \begin{array}{l} \text{MULTI} \\ \text{SINGLE} \end{array} \right\} )$$

オブジェクトの形式をマルチスレッドとする(THREAD(MULTI))か、シングルスレッドとする(THREAD(SINGLE))か、を指定します。

### 注意

オブジェクトの形式でマルチスレッド(THREAD(MULTI))を指定した場合、分散開発支援機能を利用することはできません。

### 参考

“第24章 マルチスレッド”

## A.2.58 TRACE (TRACE機能の使用の可否)

WINCOB, -WC, @

$$\left\{ \begin{array}{l} \text{TRACE [(n)]} \\ \text{NOTRACE} \end{array} \right\}$$

TRACE機能を使用する(TRACE)か、しない(NOTRACE)か、を指定します。

nには、出力するトレース情報の個数を1～999999の整数で指定します。nが指定されない場合、出力するトレース情報の個数は200個になります。

 注意

- TRACEを指定すると、トレース情報を表示するための処理が目的プログラム中に組み込まれるため、実行性能が低下します。デバッグ終了時には、NOTRACEを指定して再翻訳してください。
- TRACEは、翻訳オプションCOUNTまたは-Dcオプションと同時に指定できません。同時に指定された場合、後に指定された方が有効となります。

 参照

“A.2.8 COUNT (COUNT機能の使用の可否)”

 参考

“19.3 TRACE機能の使い方”

## A.2.59 TRUNC (桁落とし処理の可否)

WINCOB, -WC, @

$\left\{ \begin{array}{l} \text{TRUNC} \\ \text{NOTRUNC} \end{array} \right\}$

2進項目を受取り側項目とする数字転記で、上位桁の桁落としに関する処理方法を指定します。

|         |                                                                                                                                                                                                |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TRUNC   | 結果の値が受取り側項目のPICTURE句の記述に従って、上位桁が桁落としされ、受取り側項目に格納されます。翻訳オプションOPTIMIZEを同時に指定した場合、最適化によって外部10進項目または内部10進項目から導入された変数に対しても上位の桁落としが行われます。なお、送出し側項目の整数部の桁数が、受取り側項目の整数部の桁数よりも大きい場合だけ、上記のような桁落としが行われます。 |
| NOTRUNC | 目的プログラムの実行速度を優先します。桁落としを行わないほうが速く実行できる場合には、桁落としは行われません。                                                                                                                                        |

 例

PICTURE 句の記述で、

- S999V9 (整数部3桁)をS99V99 (整数部2桁)に転記 : 桁落としあり
- S9V999 (整数部1桁)をS99V99 (整数部2桁)に転記 : 桁落としなし

 参照

“A.2.37 OPTIMIZE (広域最適化の扱い)”



## 注意

- NOTRUNCで、送出し側項目の整数部の桁数が、受取り側項目の整数部の桁数より大きい場合の結果は規定されません。
- NOTRUNCを指定する場合には、桁落としが行われなくても、受取り側項目にPICTURE句に記述した桁を超える値が格納されないように、プログラムを設計しなければなりません。
- NOTRUNCで桁落としを行うか行わないかの基準は、コンパイラによって異なります。したがって、NOTRUNCによって桁落としが行われないことを利用したプログラムは他システムへの互換が保証されないので注意してください。

## A.2.60 XREF(相互参照リストの出力の可否)

WINCOB, -WC, @

```

{
 XREF
 NOXREF
}

```

相互参照リストを翻訳リストに出力する(XREF)か、しない(NOXREF)か、を指定します。

相互参照リストは、-Pオプションまたは翻訳オプションPRINTによって、翻訳リストの出力を有効にしている場合だけ出力されます。

翻訳リストの出力については、“[3.8.1.6 -dp\(翻訳リストファイルのフォルダの指定\)](#)”、“[3.8.1.17 -P\(各種翻訳リストの出力および出力先の指定\)](#)”および“[A.2.38 PRINT\(各種翻訳リストの出力の可否および出力先の指定\)](#)”を参照してください。



## 注意

翻訳オプションXREFが指定されている場合で、翻訳の結果、最大重大度コードがSレベル以上の場合、相互参照リストの出力は抑止されます。

## A.2.61 ZWB(符号付き外部10進項目と英数字項目の比較)

WINCOB, -WC, @

```

{
 ZWB
 NOZWB
}

```

符号付き外部10進項目を英数字フィールドと比較するときに、外部10進項目の符号部を無視して比較する(ZWB)か、符号部を含めて比較する(NOZWB)か、を指定します。ここで、英数字とは、英数字項目、英字項目、英数字編集項目、数字編集項目、文字定数およびZERO以外の表意定数のことです。



## 例

```

77 ED PIC S9(3) VALUE +123.
77 AN PIC X(3) VALUE "123".

```

この場合、条件式 ED = AN の真偽は、以下のようになります。

- ZWB を指定した場合 : 真

- ・ NOZWB を指定した場合 : 偽
- 

## A.3 プログラム定義にだけ指定可能な翻訳オプション

---

以下に示す翻訳オプションは、プログラム定義の翻訳時にだけ指定できます。

- ・ BINARY(BYTE)
- ・ CONF(OBS)
- ・ NOFLAGSW以外のFLAGSW
- ・ LANTLRVL(74)またはLANTLRVL(68)
- ・ MAIN、MAIN(WINMAIN)またはMAIN(MAIN)

## A.4 メソッド原型定義と分離されたメソッド定義間での翻訳オプション

---

メソッド原型定義および分離されたメソッド定義のそれぞれの翻訳時に指定された翻訳オプションは、一致している必要があります。

ただし、以下の翻訳オプションについては、一致している必要はありません。

- ・ AIMLIB
- ・ CHECK
- ・ CONF
- ・ COPY
- ・ COUNT
- ・ DLOAD
- ・ FILEEXT
- ・ FILELIB
- ・ FLAG
- ・ FORMEXT
- ・ FORMLIB
- ・ GEN
- ・ LIB
- ・ LINECOUNT
- ・ LINESIZE
- ・ LIST
- ・ MAP
- ・ MESSAGE
- ・ NUMBER
- ・ OBJECT
- ・ PRINT
- ・ QUOTE/APOST
- ・ REP
- ・ REPIN



- SAI
- SOURCE
- SRF
- TEST
- TRACE
- XREF

## 付録B 入出力状態一覧

ここでは、入出力文を実行したときに、ファイル管理記述項のFILE STATUS句に記述したデータ名に設定される値(入出力状態値)と詳細情報の意味を、下表に示します。

表B.1 入出力状態値

| 分類            | 値  | 詳細   | 意味                                                                                                                                                                                                                                                                                                                                            |
|---------------|----|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 成功            | 00 | —    | 入出力文の実行が成功した。                                                                                                                                                                                                                                                                                                                                 |
|               | 02 | —    | 入出力文の実行は成功したが、以下の状態のどちらかである。 <ul style="list-style-type: none"> <li>• READ文の実行で、読み込んだレコードの参照キーの値が、次のレコードの参照キーの値と等しい。</li> <li>• WRITE文またはREWRITE文の実行で、書き出すレコードと同じレコードキーの値を持つレコードが、すでにファイル中に存在している。ただし、そのレコードキーは、重複した値が許されているので、誤りでない。</li> </ul>                                                                                              |
|               | 04 | —    | READ文の実行は成功したが、入力したレコードの長さが最大レコード長より長い。                                                                                                                                                                                                                                                                                                       |
|               |    | MeFt | 表示ファイルでREAD文の実行は成功したが、以下の状態のどれかである。 <ul style="list-style-type: none"> <li>• 全桁必須入力項目で入力エラーが発生した。</li> <li>• 入力必須項目で入力エラーが発生した。</li> <li>• データエラーが発生した。(日本語データエラー、ANK データエラー、数字項目構成データエラー、数字項目符号エラー、数字項目小数点エラー、リダンダンシチェックエラー)</li> <li>• データ項目に入力されたデータに誤りがある。</li> </ul>                                                                     |
|               |    | ACM  | READ文の実行は成功しましたが、入力したメッセージの長さが最大レコード長よりも長い。                                                                                                                                                                                                                                                                                                   |
|               | 05 | —    | OPTIONAL句指定のファイルで、入出力文の実行は成功したが、以下の状態のどれかである。 <ul style="list-style-type: none"> <li>• ファイルに対してINPUT/I-O/EXTENDモードのOPEN文を実行したが、ファイルが未創成状態だった。</li> <li>• ファイルに対してINPUT モードのOPEN文を実行したが、ファイルが存在しなかった。ファイルは生成されず、最初のREAD文実行時にファイル終了条件(入出力状態値="10")になる。</li> <li>• ファイルに対してI-O またはEXTENDモードのOPEN文を実行したが、ファイルが存在しなかった。このとき、ファイルは生成される。</li> </ul> |
|               | 07 | —    | 入出力文の実行は成功したが、以下の方法のどれかで参照したファイルは非リール/ユニット媒体上にあった。 <ul style="list-style-type: none"> <li>• NO REWIND 指定のOPEN文またはCLOSE文</li> <li>• REEL/UNIT 指定のCLOSE文</li> </ul>                                                                                                                                                                             |
|               | 0A | ACM  | READ文の実行は成功したが、指定した論理宛先に入力すべきメッセージが存在しなかった。                                                                                                                                                                                                                                                                                                   |
|               | 0B | ACM  | WRITE文の実行は成功したが、指定した論理宛先の最大格納メッセージ数を超えてメッセージを出力した。                                                                                                                                                                                                                                                                                            |
| ファイル終了条件(不成功) | 10 | —    | 順呼出しのREAD文でファイル終了条件になった。 <ul style="list-style-type: none"> <li>• ファイルの終わりに達した。</li> <li>• 存在しない不定入力ファイルに対して初めてのREAD文を実行した。存在しない不定入力ファイルとは、OPTIONAL句指定のファイルをINPUTモードで開いたが、そのファイルが未創成状態であることをいう。</li> </ul>                                                                                                                                    |
|               | 14 | —    | 順呼出しのREAD文でファイル終了条件になった。 <ul style="list-style-type: none"> <li>• 読み込んだレコードの相対レコード番号の有効桁が、そのファイルの相対キー項目の大きさより大きい。</li> </ul>                                                                                                                                                                                                                  |

| 分類          | 値  | 詳細  | 意味                                                                                                                                                                                                                                                                                        |
|-------------|----|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 無効キー条件（不成功） | 21 | —   | レコードキーの順序誤り。次の状態のどれかである。 <ul style="list-style-type: none"> <li>順呼出しにおいて、READ文とそれに続くREWRITE文との間で、主レコードキーの値が変更された。</li> <li>乱呼出しまたは動的呼出しにおいて、主キーにDUPLICATES指定の記述があるファイルで、READ文とそれに続くREWRITE文またはDELETE文との間で、主レコードキーの値が変更された。</li> <li>順呼出しにおいて、WRITE文の実行のときに主レコードキーの値が昇順になっていない。</li> </ul> |
|             | 22 | —   | WRITE文またはREWRITE文の実行時に、書こうとしたレコードの主レコードキーまたは副レコードキーの値が、すでにファイル中に存在している。ただし、主レコードキーまたは副レコードキーにDUPLICATES指定が記述されている場合を除く。                                                                                                                                                                   |
|             | 23 | —   | レコードが見つからない。 <ul style="list-style-type: none"> <li>START文または乱呼出しのREAD/REWRITE/DELETE文の実行で、指定されたキー値をもつレコードがファイル中に存在しない。</li> <li>相対レコード番号に0が指定された。</li> </ul>                                                                                                                             |
|             | 24 | —   | 次の状態のどれかである。 <ul style="list-style-type: none"> <li>WRITE文の実行で、領域不足が発生した。</li> <li>WRITE文の実行で、指定されたキーがキーレンジ外である。</li> <li>区域外書出し発生後、さらにWRITE文を実行しようとした。</li> </ul>                                                                                                                        |
| 永続誤り条件（不成功） | 30 | —   | 物理的なエラーが発生した。                                                                                                                                                                                                                                                                             |
|             |    | ACM | ACM がエラーを検出した。                                                                                                                                                                                                                                                                            |
|             | 34 | —   | WRITE文の実行で、領域不足が発生した。                                                                                                                                                                                                                                                                     |
|             | 35 | —   | OPTIONAL句指定のないファイルに対して、INPUT/I-O/EXTENDモードのOPEN文を実行したが、ファイルが未創成状態だった。                                                                                                                                                                                                                     |
|             | 37 | —   | 指定された機能は、未サポートである。                                                                                                                                                                                                                                                                        |
|             | 38 | —   | 以前にCLOSE LOCK文を実行したファイルに対してOPEN文を実行した。                                                                                                                                                                                                                                                    |
|             | 39 | —   | OPEN文の実行時に、プログラム中でそのファイルに指定した属性と矛盾するファイルが割り当てられた。                                                                                                                                                                                                                                         |
| 論理誤り条件（不成功） | 41 | —   | すでに開いたファイルに対して、OPEN文を実行した。                                                                                                                                                                                                                                                                |
|             | 42 | —   | 開いていないファイルに対して、CLOSE文を実行した。                                                                                                                                                                                                                                                               |
|             | 43 | —   | 順呼出しまたは主キーにDUPLICATES指定を記述したファイルに対するDELETE文またはREWRITE文の実行で、先行する入出力文が成功したREAD文でなかった。                                                                                                                                                                                                       |
|             | 44 | —   | 以下の状態のどちらかである。 <ul style="list-style-type: none"> <li>WRITE/REWRITE文実行時のレコード長が、プログラムの記述で決められた最大レコード長より長い、またはレコード長として誤った数値が指定された。</li> <li>REWRITE文実行時に、そのレコードの長さが書き換えるレコードの長さとは等しくない。</li> </ul>                                                                                          |
|             | 46 | —   | 順呼出しのREAD文の実行で、次のどちらかの理由でファイル位置指示子が不定である。 <ul style="list-style-type: none"> <li>先行するSTART文が不成功</li> <li>先行するREAD文が不成功（ファイル終了条件も含む）</li> </ul>                                                                                                                                            |
|             | 47 | —   | INPUT/I-Oモードで開かれていないファイルに対してREAD文またはSTART文を実行した。                                                                                                                                                                                                                                          |
|             | 48 | —   | OUTPUT/EXTEND（順・相対・索引）またはI-O（相対・索引）モードで開かれていないファイルに対して、WRITE文を実行した。                                                                                                                                                                                                                       |
|             | 49 | —   | I-Oモードで開かれていないファイルに対してREWRITE文またはDELETE文を実行した。                                                                                                                                                                                                                                            |

| 分類          | 値  | 詳細   | 意味                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------------|----|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| その他の誤り（不成功） | 90 | —    | 他のどれにも含まれないエラーである。以下のような状態が考えられる。 <ul style="list-style-type: none"> <li>ファイル情報が不完全またはその情報に誤りがある。</li> <li>OPEN/CLOSE文の実行時に、関数で何らかのエラーが発生した。</li> <li>以前にCLOSE文が入出力状態値90で不成功になったファイルに対して、入出力文を実行しようとした。</li> <li>主記憶などの資源が利用できない。</li> <li>正しく閉じられていないファイルに対して、OPEN文を実行した。</li> <li>区域外書き出しによる誤りの発生後、レコードを書き出そうとした。</li> <li>no-space状態発生後、レコードを書き出そうとした。</li> <li>テキストファイルのレコードに不当な文字がある。</li> <li>コードセットに変換できない文字がある。</li> <li>同一のファイルに対し、多数のアプリケーションからOPEN要求された。その結果、ロックテーブルに不足が発生した。</li> <li>必要な関連製品のローディングに失敗した。</li> <li>上記以外の誤りが存在する。その入出力動作に関してはそれ以上の情報がない。</li> <li>システムエラーが発生した。</li> </ul> |
|             |    | MeFt | MeFtがエラーを検出した。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|             |    | ACM  | ACM がエラーを検出した。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|             | 91 | —    | <ul style="list-style-type: none"> <li>ファイルが割り当てられていない。</li> <li>OPEN文実行時に、ファイル識別名と物理ファイル名の対応付けがない。</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|             | 92 | —    | 排他エラーが発生した。(RDM ファイル)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|             | 93 | —    | 排他エラーが発生した。(ファイルロック)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|             | 99 | —    | 排他エラーが発生した。(レコードロック)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|             |    | MeFt | システム異常が発生した。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|             |    | ACM  | システム障害が発生した。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|             | 9B | ACM  | 宛先異常が発生した。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|             | 9E | MeFt | READ文による実行待ち状態が強制的に解除された。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|             | 9F | ACM  | タイムオーバが発生した。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|             | 9G | ACM  | 論理宛先の最大メッセージ数オーバが発生した。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|             | 9H | ACM  | 論理宛先のファイル領域不足または記憶域不足が発生した。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

注:FORMAT句付き印刷ファイルおよび表示ファイルの場合は、入出力状態値と詳細情報が通知されます。詳細情報の“MeFt”はMeFtの通知コードを、“ACM”はSymfoWARE7000またはACM制御のエラーコードを参照してください。“-”は詳細情報に0000が通知されます。



## 参照

以下のマニュアルを参照してください。

- “MeFtのオンラインマニュアル”
- “RDB/7000 説明書”
- “SymfoWARE7000のRDB\_7000連携ヘルプ”

- “SymfoNET説明書 DB連携編”
- “ACM制御 説明書”



## 付録C 広域最適化

ここでは、コンパイラが行う広域最適化の内容および使用上の注意事項について記述します。

### C.1 最適化の項目

広域最適化では、1入口・1出口の手続き部を、記述順に実行されるような文の列(これを基本ブロックといいます)に分割します。そして、制御の移行およびデータの使用状態を解析し、主にループ(繰り返し実行される部分)に着目した最適化を行います。

以下に、最適化の項目を示します。

- ・ 共通式の除去
- ・ 不変式の移動
- ・ 誘導変数の最適化
- ・ PERFORM文の最適化
- ・ 隣接転記の統合
- ・ 無駄な代入の除去

### C.2 共通式の除去

以前に行われた演算や変換の結果が利用できる場合に、演算や変換を実行しないで、前の結果を保存しておいて、それを使用します。



#### 例

#### 例1

例1では、[1]と[2]の間で“添字1”の値が不変であれば、“項目1(添字1,添字2)”のアドレス計算式“項目1-22+添字1\*20+添字2\*2”(注)と、“項目2(添字1,添字3)”のアドレス計算式“項目2-22+添字1\*20+添字3\*2”で、(添字1\*20)の部分が共通となるので、[2]は[1]の結果を使用するように最適化されます。

注：項目1+(添字1-1)\*20+(添字2-1)\*2 = 項目1-22+添字1\*20+添字2\*2

```
77 添字 1 PIC S99 BINARY.
77 添字 2 PIC S99 BINARY.
77 添字 3 PIC S99 BINARY.
01 集団項目.
 02 項目 1 OCCURS 25 TIMES.
 03 項目 1 1 PIC XX OCCURS 10 TIMES.
 02 項目 2 OCCURS 35 TIMES.
 03 項目 2 1 PIC XX OCCURS 10 TIMES.
 :
 MOVE SPACE TO 項目 1 1 (添字 1, 添字 2). ... [1]
 :
 MOVE SPACE TO 項目 2 1 (添字 1, 添字 3). ... [2]
```

#### 例2

例2では、[1]と[2]の間で“数字1”と“数字2”の値が不変であれば、(数字1 \* 数字2)が共通となるので、[2]は[1]の結果を使用するように最適化されます。

```
77 計算結果 1 PIC S9(9) DISPLAY.
77 計算結果 2 PIC S9(9) DISPLAY.
77 数字 1 PIC S9(4) BINARY.
77 数字 2 PIC S9(4) BINARY.
 :
```

```

COMPUTE 計算結果 1 = 数字 1 * 数字 2. ... [1]
:
COMPUTE 計算結果 2 = 数字 1 * 数字 2. ... [2]

```

## C.3 不変式の移動

演算や変換がループ内にあり、ループ内外両方で行っても結果が変わらない場合、これをループ外に移動します。



例

ループ内で“外部10進項目”の値が不変であれば、“2進項目(添字)”と比較する時の外部10進数を2進数に変換する処理は、ループ外に移動されます。

```

77 添字 PIC S9(4) BINARY.
77 外部 1 0 進項目 PIC S9(7) DISPLAY.
01 集団項目.
 02 2 進項目 PIC S9(7) BINARY OCCURS 20 TIMES.
 :
 MOVE 1 TO 添字.
ループ開始.
 IF 2 進項目 (添字) = 外部 1 0 進項目 GO TO ループ終了.
 :
 ADD 1 TO 添字.
 IF 添字 IS <= 20 GO TO ループ開始.
ループ終了.

```

↑  
|  
ループ  
|  
↓

## C.4 誘導変数の最適化

ループ内で、定数または値が不変な項目によってだけ再帰的に定義される項目を誘導変数といいます。誘導変数を使用している部分式がある場合、新しい誘導変数を導入することにより、添字計算のための乗算を加算に変更します。



例

“添字”は誘導変数です(注1)。ここでは、新しい誘導変数(これをtとします)を導入し、“繰返し項目(添字)”のアドレス計算式“繰返し項目-10+添字\*10”(注2)の中の乗算(添字\*10)がtで置き換えられ、[2]の後に“ADD 10 TO t”が生成されます。さらに、ループ中で“添字”が他に使用されず、かつ、ループ中で計算した“添字”の値をループが出た後に使用していない場合、[3]は“IF t IS <= 200 GO TO ループ開始.”で置き換えられ、[2]は削除されます。

注1 : ループ内で定数により再帰的にだけ定義されています。

注2 : 繰返し項目+(添字-1)\*10 = 繰返し項目-10+添字\*10

```

77 添字 PIC S9(4) COMP-5.
01 集団項目.
 02 繰返し項目 PIC X(10) OCCURS 20 TIMES.
 :
ループ開始.
 IF 繰返し項目 (添字) = ... [1]
 :
 ADD 1 TO 添字. ... [2]
 IF 添字 IS <= 20 GO TO ループ開始. ... [3]

```

↑  
|  
ループ  
|  
↓

## C.5 PERFORM文の最適化

PERFORM文は、その復帰機構として、戻り先のアドレスを退避、設定および復元するために、いくつかの機械命令に展開されます。そこで、PERFORM文の出口に、そのPERFORM文以外で制御が渡る場合、復帰機構の機械命令のうちのいくつかが冗長となることがあります。これらの冗長な機械命令は削除されます。

## C.6 隣接転記の統合

複数の英数字転記文があり、領域の連続した項目が同じように領域の連続した項目に転記される場合、これらを1つの文にまとめます。



### 例

[1]と[2]の間で“基本項目A2”と“基本項目B2”の値が不変で、かつ、“基本項目B2”が参照されていないならば、[2]は削除されます。このとき、“基本項目A1”と“基本項目A2”を1つの領域とし、“基本項目B1”と“基本項目B2”を1つの領域として、まとめて転記が行われます。

```
02 基本項目 A 1 PIC X(32).
02 基本項目 A 2 PIC X(16).
:
02 基本項目 B 1 PIC X(32).
02 基本項目 B 2 PIC X(16).
:
MOVE 基本項目 A 1 TO 基本項目 B 1. ... [1]
:
MOVE 基本項目 A 2 TO 基本項目 B 2. ... [2]
```

## C.7 無駄な代入の除去

それ以降一度も明または暗に参照されないデータ項目への代入は、削除されます。

## C.8 広域最適化での注意事項

翻訳オプションOPTIMIZEが有効な場合、コンパイラは広域最適化を行った目的プログラムを生成します。詳細は、“[A.2.37 OPTIMIZE \(広域最適化の扱い\)](#)”を参照してください。

以下は、広域最適化での注意事項です。

### 連絡機能を使用する場合

呼ばれるプログラムに対し、“CALL "SUB" USING A,A.”や“CALL "SUB" USING A,B. (注)”のように、同時に指定された複数のパラメタにおいて、領域の一部または全部を共有しているものがあります。呼ばれるプログラムでその内容を書き換えていると、呼ばれるプログラムの最適化により、意図したとおりの結果が得られない場合があります。

注：ただし、AとBは領域の一部を共有します。

### 広域最適化が行われない場合

以下のプログラムでは、広域最適化は行われません。

- ・ 広域最適化の対象となる属性を持った項目および指標名が1つも定義されていないプログラム
- ・ 区分化機能を使用しているプログラム

以下のプログラムでは、広域最適化の効果は少なくなります。

- ・ 入出力操作を主とし、もともとCPUをあまり使わないプログラム
- ・ 数字項目を使わず、英数字項目ばかりを使うプログラム
- ・ 宣言部分から非宣言部分を参照しているプログラム
- ・ 非宣言部分から宣言部分を参照しているプログラム
- ・ 翻訳オプションTRUNCを指定しているプログラム





## 参照

“A.2.59 TRUNC (桁落とし処理の可否)”

## デバッグを行う場合

以下の注意が必要です。

- 広域最適化によって文の削除、移動および変更が行われるので、データ例外などのプログラム割込みの起こる回数や場所が変わることがあります。
- プログラム割込みなどで中断したとき、プログラム上の記述でデータ項目に値を設定していても、実際には設定されていないことがあります。
- 再帰的に定義される項目が内部10進項目または外部10進項目の場合、翻訳オプションNOTRUNCが指定されていると、意図したとおりにプログラムが動作しないことがあります。



## 参照

“A.2.59 TRUNC (桁落とし処理の可否)”

- 翻訳オプションTESTを同時に指定したプログラムのデバッグ情報ファイルは、対話型デバッガでは使用できません。



## 参照

“A.2.56 TEST (対話型デバッガおよび診断機能の使用の可否)”

## 付録D 組込み関数の使用

ここでは、組込み関数の用例や、記述の際の注意点について述べます。

### D.1 関数の型と記述の関係

関数はそれぞれに型を持っています。そして、その型の違いによってプログラム中に記述できる場所も異なります。関数と型の対応については、“表D.1 組込み関数一覧”を参照してください。

それぞれの型の関数の記述について、以下に説明します。なお、関数の呼出し形式に沿って書かれた記述のことを正しくは「関数一意名」と呼びますが、ここでは単に「関数」と呼んでいます。

#### 整数関数

整数関数は、算術式中にだけ記述できます。整数関数を算術式以外の場所、たとえばMOVE文の送出し側に記述することはできません。例ではCOMPUTE文で使用しています。



例

#### 通日計算

- COBOL プログラムの記述

```
DATA DIVISION.
WORKING-STORAGE SECTION.
 01 INT PIC S9(9) COMP-5.
 01 IN-YMD PIC 9(8).
 01 OUT-YMD PIC 9(8).
 01 OUT-YMD-ED PIC XXXX/XX/XX.
PROCEDURE DIVISION.
 MOVE 20021225 TO IN-YMD.
*年月日→通日計算
 COMPUTE INT = FUNCTION INTEGER-OF-DATE(IN-YMD).
 DISPLAY "2002年12月25日は、基準日から" INT "日目です. ".
*通日→年月日計算
 COMPUTE OUT-YMD = FUNCTION DATE-OF-INTEGER (INT).
 MOVE OUT-YMD TO OUT-YMD-ED.
 DISPLAY "基準日から" INT "日目は、" OUT-YMD-ED "です. ".
```

- 実行結果

```
2002年12月25日は、基準日から+000146821日目です.
基準日から+000146821日目は、2002/12/25です.
```

#### 数字関数

数字関数は、整数関数と同様、算術式中にだけ記述できます。数字関数を算術式以外の場所、たとえばMOVE文の送出し側に記述することはできません。

#### 英数字関数

英数字関数は、英数字項目が記述できるところに書くことができます。



例

#### UPPER-CASE関数

- ・ COBOL プログラムの記述

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LOWCASE PIC X(13) VALUE "fujitsu cobol".
PROCEDURE DIVISION.
 DISPLAY "変換前 : " LOWCASE.
 DISPLAY "変換後 : " FUNCTION UPPER-CASE (LOWCASE).
```

- ・ 実行結果

```
変換前 : fujitsu cobol
変換後 : FUJITSU COBOL
```

例では、大文字に変換した文字を直接DISPLAY文で表示していますが、MOVE文の送出し側に記述して作業域に転記することもできます。

## 日本語関数

日本語関数は、日本語項目が記述できるところに書くことができます。  
例では、変換した文字を一度転記してから表示します。



例

## NATIONAL関数

- ・ COBOL プログラムの記述

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 NCHAR PIC N(7).
01 CHAR PIC X(7) VALUE "FUJITSU".
PROCEDURE DIVISION.
*英数字を日本語文字に変換して表示
 MOVE FUNCTION NATIONAL (CHAR) TO NCHAR.
 DISPLAY "英数字 : " CHAR.
 DISPLAY "日本語 : " NCHAR.
```

- ・ 実行結果

```
英数字 : FUJITSU
日本語 : F U J I T S U
```

## ポインタデータ関数

ポインタデータ関数の記述については、“[14.3 ADDR関数およびLENG関数の使い方](#)”を参照してください。

## D.2 引数の型によって決定される関数の型

関数の中には、引数の型によって関数の型が決まるものがあります。  
最大値を求めるMAX関数を例に挙げて説明します。



例

- ・ COBOL プログラムの記述

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 C1 PIC X(10).
```

```

01 C2 PIC X(5).
01 C3 PIC X(5).
01 V1 PIC S9(3).
01 V2 PIC S9(3)V9(2).
01 V3 PIC S9(3).
01 MAXCHAR PIC X(10).
01 MAXVALUE PIC S9(3)V9(2).
PROCEDURE DIVISION.
 MOVE FUNCTION MAX(C1 C2 C3) TO MAXCHAR. ...[1]
 :
 COMPUTE MAXVALUE = FUNCTION MAX(V1 V2 V3). ...[2]

```

MAX関数は、最大値を求める関数で、関数の型は引数の型によって決まります。

[1]は引数の型が英数字であるため、関数の型は英数字となります。また、[2]は引数の型が数字であるため、関数の型は数字となります。

## D.3 CURRENT-DATE関数を利用した西暦の取得

小入出力機能を使った日付入力では、西暦の下2桁しか取得できませんが、CURRENT-DATE関数を利用すれば、4桁の西暦を得ることができます。



### 例

COBOL プログラムの記述

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01 TODAY.
 02 T-YEAR PIC X(4).
 02 PIC X(17).
PROCEDURE DIVISION.
 MOVE FUNCTION CURRENT-DATE TO TODAY.

```

転記後の変数YEARの内容が、4桁の西暦を示します。

環境変数情報“@CBR\_JOBDATE”に任意の日付を指定すると、CURRENT-DATE関数により指定された日付を受け取ることができます。



### 例

COBOL プログラムの記述

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01 TODAY.
 02 T-YEAR PIC X(4).
 02 T-MONTH PIC X(2).
 02 T-DAY PIC X(2).
 02 PIC X(13).
PROCEDURE DIVISION.
 MOVE FUNCTION CURRENT-DATE TO TODAY.

```

プログラム実行時に環境変数情報“@CBR\_JOBDATE”に1999.09.01を設定します。

転記後の変数T-YEARに1999、変数T-MONTHに09、変数T-DAYに01が格納されます。

環境変数情報の指定形式については、“[11.1.7 任意の日付の入力](#)”を参照してください。



## 注意

例では、MOVE文の受取り側が集団項目であるため、集団項目転記が行われていますが、受取り側が数字項目であった場合は、数字転記が行われます。数字転記と集団項目転記では、桁よせの規則が異なるため、以下のように4桁の領域を準備しても、西暦は取得できません。

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77 LAG PIC 9(4).
PROCEDURE DIVISION.
 MOVE FUNCTION CURRENT-DATE TO LAG.
```

転記後の変数LAGの内容は、西暦ではなく、グリニッジ標準時からの進みまたは遅れ(CURRENT-DATE関数の関数値の、18～21桁)を示します。

## D.4 任意の基準日からの通日計算

通日計算の結果得られた値の差を取り、任意の基準日からの通日を知ることができます。  
例では、任意の基準日から現在の日付までの通日を計算し、その期間内での利息計算を行っています。



### 例

- COBOL プログラムの記述

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 TODAY PIC X(8).
01 TODAY-R REDEFINES TODAY PIC 9(8).
01 FROM-DAY PIC 9(8).
01 INT PIC 9(5).
01 PAY PIC 9(6).
01 EDT-PAY PIC ZZZ.ZZ9.
01 EDT-INT PIC ZZZZ9.
01 EDT1 PIC XXXX/XX/XX.
01 EDT2 PIC XXXX/XX/XX.
PROCEDURE DIVISION.
* 基準日を設定する
 ACCEPT FROM-DAY
 MOVE FROM-DAY TO EDT1
* 今日の日付を取得する
 MOVE FUNCTION CURRENT-DATE TO TODAY EDT2
* 2つの日付間の日数を計算する
 COMPUTE INT = (FUNCTION INTEGER-OF-DATE(TODAY-R))
 - (FUNCTION INTEGER-OF-DATE(FROM-DAY))
* 利息計算(例: 20日あたり133.3円固定)
 COMPUTE PAY = FUNCTION INTEGER-PART((INT / 20) * 133.3)

 MOVE PAY TO EDT-PAY.
 MOVE INT TO EDT-INT.
* 結果の表示
 DISPLAY "預入日(" EDT1 ")から " EDT-INT "日 経過しています。"
 DISPLAY EDT2 " 現在の利息合計は " EDT-PAY "円です。"
```

- 実行結果(基準日として"19920521"を入力した場合)

```
預入日(1992/05/21)から2272日 経過しています。
1998/08/10 現在の利息合計は 15,062円です。
```

# D.5 RANDOM関数を利用した疑似乱数列の生成

RANDOM関数の関数値として、疑似乱数を取得できます。  
このとき、関数値の範囲は  $0 \leq \text{関数値} < 1$  で、作用対象の小数部桁数に合わせて、桁よせされます。



## 例

### COBOLプログラムの記述

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 A PIC 9(08).
01 B PIC V9(07).
PROCEDURE DIVISION.
*
* 引数は省略できます。
 PERFORM 5 TIMES
 COMPUTE B = FUNCTION RANDOM
 DISPLAY B
 END-PERFORM.
*
* 同じ種子の場合、同じ疑似乱数の値が返却されます。
 MOVE 12345678 TO A.
 COMPUTE B = FUNCTION RANDOM(A).
 DISPLAY B.
 COMPUTE B = FUNCTION RANDOM(A).
 DISPLAY B.
*
 STOP RUN.
```



## 注意

- 引数の値(種子)が同じ場合は、同じ疑似乱数が返却されます。ただし、種子が異なる場合および種子を指定しない場合でも、同じ値が返却されることもあります。
- 関数値の範囲内であっても、返却されない疑似乱数はあります。

# D.6 組込み関数一覧

NetCOBOLで利用できる組込み関数の一覧を“表D.1 組込み関数一覧”に記述します。

表D.1 組込み関数一覧

| 分類  | 関数                 | 用途                | 関数の型        |
|-----|--------------------|-------------------|-------------|
| 長さ  | LENGTH             | データ項目や定数の長さを求めます。 | 整数          |
|     | LENG               | バイト数を求めます。        | 整数          |
|     | STORED-CHAR-LENGTH | 有効文字の長さを求めます。     | 整数          |
| 大きさ | MAX                | 最大値を求めます。         | 整数、数字または英数字 |
|     | MIN                | 最小値を求めます。         | 整数、数字または英数字 |
|     | ORD-MAX            | 最大値の順序位置を求めます。    | 整数          |
|     | ORD-MIN            | 最小値の順序位置を求めます。    | 整数          |
| 変換  | REVERSE            | 文字列の順序を逆にします。     | 英数字         |

| 分類   | 関数                 | 用途                                   | 関数の型    |
|------|--------------------|--------------------------------------|---------|
|      | LOWER-CASE         | 大文字を小文字に変換します。                       | 英数字     |
|      | UPPER-CASE         | 小文字を大文字に変換します。                       | 英数字     |
|      | NUMVAL             | 数字文字を数値に変換します。                       | 数字      |
|      | NUMVAL-C           | コンマや通貨記号のある数字文字を数値に変換します。            | 数字      |
|      | NATIONAL           | 日本語文字に変換します。                         | 日本語     |
|      | CAST-ALPHANUMERIC  | 項類および字類を英数字に変換します。                   | 英数字     |
|      | UCS2-OF            | エンコード方式をUCS-2に変換します。                 | 日本語     |
|      | UTF8-OF            | エンコード方式をUTF-8に変換します。                 | 英数字     |
|      | DISPLAY-OF         | 英数字文字に変換します。                         | 英数字     |
|      | NATIONAL-OF        | 日本語文字に変換します。                         | 日本語     |
| 文字操作 | CHAR               | プログラムの文字の大小順序において、指定した位置にある1文字を求めます。 | 英数字     |
|      | ORD                | プログラムの文字の大小順序において、指定した文字の順序位置を求めます。  | 整数      |
| 数値操作 | INTEGER            | 指定した値を超えない最大の整数を求めます。                | 整数      |
|      | INTEGER-PART       | 整数部を求めます。                            | 整数      |
|      | RANDOM             | 乱数を求めます。                             | 数字      |
| 金利計算 | ANNUITY            | 元金を1とし、利率と期間から均等払いの比率の近似値を求めます。      | 数字      |
|      | PRESENT-VALUE      | 減価率による現在の価格を求めます。                    | 数字      |
| 日付操作 | CURRENT-DATE       | 現在の日付、時刻、グリニッジ標準時からの時差を求めます。         | 英数字     |
|      | DATE-OF-INTEGER    | 通日に対応する年月日を求めます。                     | 整数      |
|      | DAY-OF-INTEGER     | 通日に対応する年日を求めます。                      | 整数      |
|      | INTEGER-OF-DATE    | 年月日に対応する通日を求めます。                     | 整数      |
|      | INTEGER-OF-DAY     | 年日に対応する通日を求めます。                      | 整数      |
|      | WHEN-COMPILED      | プログラムが翻訳された日時を求めます。                  | 英数字     |
| 算術計算 | SQRT               | 平方根の近似値を求めます。                        | 数字      |
|      | FACTORIAL          | 階乗を求めます。                             | 整数      |
|      | LOG                | 自然対数を求めます。                           | 数字      |
|      | LOG10              | 常用対数を求めます。                           | 数字      |
|      | MEAN               | 平均値を求めます。                            | 数字      |
|      | MEDIAN             | 中央値を求めます。                            | 数字      |
|      | MIDRANGE           | 最小値と最大値の平均値を求めます。                    | 数字      |
|      | RANGE              | 最大値と最小値の差を求めます。                      | 整数または数字 |
|      | STANDARD-DEVIATION | 標準偏差を求めます。                           | 数字      |
|      | MOD                | 指定した法での指定した値の値を求めます。                 | 整数      |
|      | REM                | 余りを求めます。                             | 数字      |
|      | SUM                | 和を求めます。                              | 整数または数字 |
|      | VARIANCE           | 分散を求めます。                             | 数字      |

| 分類   | 関数   | 用途            | 関数の型    |
|------|------|---------------|---------|
| 三角関数 | SIN  | 正弦の近似値を求めます。  | 数字      |
|      | COS  | 余弦の近似値を求めます。  | 数字      |
|      | TAN  | 正接の近似値を求めます。  | 数字      |
|      | ASIN | 逆正弦の近似値を求めます。 | 数字      |
|      | ACOS | 逆余弦の近似値を求めます。 | 数字      |
|      | ATAN | 逆正接の近似値を求めます。 | 数字      |
| ポインタ | ADDR | 先頭アドレスを求めます。  | ポインタデータ |



## 付録E 特殊な定数の書き方

ここでは、プログラム名やファイル名などのシステムで定められた名前を指定する、各種定数の書き方を説明します。

### E.1 プログラム名定数

プログラム名段落(PROGRAM-ID)、CALL文およびCANCEL文に定数指定でプログラム名を指定する場合、使用できる文字に制限はありません。ただし、定数の長さは60バイト以内でなければなりません。指定した文字がリンカの規則に従っているかどうかは、利用者が判断します。

プログラム名定数の長さは60バイト以内でなければなりません。

### E.2 原文名定数

COPY文に記述する原文名定数には、登録集原文を格納したファイルの名前を以下の形式で記述します。

"[ドライブ名:] [パス名] ファイル名 [.拡張子]"

#### ドライブ名

ドライブ名の指定をAからZまでの英字1文字で指定します。  
ドライブ名が省略された場合、カレントドライブとみなします。

#### パス名

ファイルの格納先を以下の形式で指定します。

[¥][フォルダ名 [¥ フォルダ名] … ]¥

パス名が省略された場合、翻訳オプションLIBで指定されたフォルダまたはカレントフォルダ中のファイルとみなします。なお、WINCOBウィンドウから翻訳を行う場合、カレントフォルダは、翻訳対象となるソースファイルが格納されているフォルダとなります。相対パス名が指定された場合、カレントフォルダが先頭に付加されます。

#### ファイル名

ファイルの名前を指定します。

#### 拡張子

ファイルが拡張子を持つ場合、指定します。  
拡張子は、“CBL”や“COB”以外でも可能です。

- C:¥COPY¥A.CBL
- A.CPY
- C:¥COPY¥A

### E.3 ファイル識別名定数

ファイル管理記述項のASSIGN句に指定するファイル識別名定数は、実行時に処理するファイルを以下の形式で指定します。

"[ { ドライブ名  
ポート名 } : ] [パス名] ファイル名 [.拡張子]"

---

### ドライブ名

ドライブ名の指定をAからZまでの英字1文字で指定します。AからZまでの英字1文字でない場合は、ポート名として扱われます。

ドライブ名が省略された場合、カレントドライブとみなします。

### ポート名

ポート名は順ファイルにだけ指定でき、ポート名を指定した場合、パス名およびファイル参照名の指定は無効となります。

ポート名でプリンタ装置を指定する場合、ポート名LPT1、LPT2またはLPT3を使用してください。

### パス名

ファイルの格納先を以下の形式で指定します。

`[¥][フォルダ名 [¥ フォルダ名] … ]¥`

パス名が省略された場合、ファイル名で示されるファイルはカレントフォルダ中のファイルとみなされます。

### ファイル名

ファイルの名前を指定します。

### 拡張子

ファイルの種類を区別するための、任意の文字列を指定します。

- A:¥COBOL¥A.DAT
- LPT1:
- B.TXT

## E.4 外部名を指定するための定数

---

見出し部で定義する以下の名前には、AS指定に定数を指定して外部名を付けることができます。AS指定を省略すると内部名と外部名は同じになります。

### プログラム名

`PROGRAM-ID. CODE-GET AS "XY1234".`

### クラス名

`CLASS-ID. 特殊処理 AS "SP-CLASS-001".`

### メソッド名

`METHOD-ID. 値 AS "VALUE".`

このAS指定に指定する定数は、最初の文字がアンダースコアで始まってはならないことを除けば、使用できる文字およびその構成規則に制限はありません。したがって、リンカの規則に従っているかどうかは、利用者が判断します。

## 付録F OSIV系システムとの機能比較

OSIV系システムに関する用語の定義を、以下に示します。

OSIV系システム

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

OSIV系プログラム

OSIV系システムで動作するプログラム

OSIV系システム固有機能

OSIV系システムでのみ利用できる機能

### F.1 機能比較一覧

OSIV系システムのCOBOL85と本システムのNetCOBOLの機能比較を“表F.1 OSIV系システムと本システムの機能比較”に示します。

表において、“比較”の記号は次の意味です。

- ： OSIV系システムと同様に使用することができます。
- ： 条件付きでOSIV系システムと同様に使用することができます。
- △： 本システム固有機能またはOSIV系システムとの非互換のため、OSIV系システムでは使用できません。
- ： 翻訳はできますが、実行時にその機能が有効となりません。
- ： 使用できますが、OSIV系システムと動作が異なります。
- ×： 本システムでは使用できません。

表F.1 OSIV系システムと本システムの機能比較

| 機能分類   |        |                                                                                                 | 比較 | 備考                                                            |
|--------|--------|-------------------------------------------------------------------------------------------------|----|---------------------------------------------------------------|
| 分類     | 機能概要   |                                                                                                 |    |                                                               |
| 文字集合   |        | プログラム中で使用可能な文字の種類すべて                                                                            | ○  |                                                               |
| コード系   |        | Unicode                                                                                         | △  |                                                               |
|        |        | システム依存                                                                                          | ○  |                                                               |
| COBOL語 | 利用者語   | 利用者語の種類すべて                                                                                      | ○  | ただし、_ (アンダースコア)の使用は、本システム固有の機能です。使用可能な日本語文字は、各システムのコード系に従います。 |
|        | 表意定数   | プログラム中で使用可能な表意定数すべて                                                                             | ○  |                                                               |
|        | 特殊レジスタ | SHIFT-IN<br>SHIFT-OUT                                                                           | ●  |                                                               |
|        |        | PROGRAM-STATUS<br>RETURN-CODE                                                                   | □  | 属性が異なるOSIV系システム：<br>S9(4)BINARY<br>本システム：S9(9)COMP-5          |
|        |        | 上記以外                                                                                            | ○  |                                                               |
|        | 機能名    | SYSPUNCH<br>STACKER-01～12<br>CSP<br>S01 ～02<br>SYSPCH<br>BUSHU<br>SOKAKU<br>ON-YOMI<br>KUN-YOMI | ●  |                                                               |

| 機能分類         |           |                                                                      |          | 比較 | 備考                                                             |  |
|--------------|-----------|----------------------------------------------------------------------|----------|----|----------------------------------------------------------------|--|
| 分類           |           | 機能概要                                                                 |          |    |                                                                |  |
|              |           | SWITCH-8<br>SYSERR                                                   |          | △  |                                                                |  |
|              |           | CHANNEL02 ～12<br>C02～12                                              |          | □  | FCB 制御文の指定が必要です。                                               |  |
|              |           | 上記以外                                                                 |          | ○  |                                                                |  |
|              | 定数        | 日本語項目定数<br>日本語英数字定数<br>日本語連想定数<br>日本語ひらがな定数                          |          | ×  |                                                                |  |
|              |           | 16進文字定数<br>日本語16進定数<br>日本語コード定数                                      |          | ■  | コード系の違いに注意が必要です。                                               |  |
|              |           | 上記以外                                                                 |          | ○  |                                                                |  |
|              | その他       | 定数の引用符指定                                                             |          | △  | OSIV系システム:<br>翻訳オプション<br>APOST/QUOTE に従う<br>本システム:<br>自動的に判定する |  |
|              | プログラムの書き方 | 正書法                                                                  | 自由形式     | △  |                                                                |  |
| 一連番号         |           |                                                                      | ○        |    |                                                                |  |
| 固定形式<br>可変形式 |           |                                                                      | ○        |    |                                                                |  |
| データ定義        | データ記述     | データ記述項に記述可能な句すべて                                                     |          | ○  |                                                                |  |
|              |           | 名前付き定数(78項目)                                                         |          | △  |                                                                |  |
|              |           | 型定義                                                                  |          | △  |                                                                |  |
|              |           | 型を参照するデータ定義                                                          |          | △  |                                                                |  |
|              | データ型      | BINARY-CHAR UNSIGNED<br>BINARY-SHORT<br>BINARY-LONG<br>BINARY-DOUBLE |          | △  |                                                                |  |
|              |           | 上記以外                                                                 |          | ○  |                                                                |  |
| 式            | 算術式       | 2項演算子<br>単項演算子                                                       |          | ○  |                                                                |  |
|              | 条件式       | 使用可能な比較演算子すべて                                                        |          | ○  |                                                                |  |
|              | 連結式       | 連結式の使用                                                               |          | △  |                                                                |  |
|              | 字類条件      | 使用可能な字類条件すべて                                                         |          | ○  | 個々の文字が実際に字類条件に合致するか否かはシステムのコード系に依存します。                         |  |
|              | その他の条件    | 条件名条件<br>正負条件<br>スイッチ状態条件                                            |          | ○  |                                                                |  |
| 中核機能         | 環境定義      | SUBSCHEMA-NAME段落                                                     |          | □  | デバッガを使って動作確認を行うことができます。(注1)                                    |  |
|              |           | ALPHABET句                                                            | EBCDIC指定 |    | △                                                              |  |
|              |           |                                                                      | 上記以外     |    | ○                                                              |  |

| 機能分類   |        |                                                                                  | 比較 | 備考                                                   |
|--------|--------|----------------------------------------------------------------------------------|----|------------------------------------------------------|
| 分類     |        | 機能概要                                                                             |    |                                                      |
| 順ファイル  |        | 上記以外                                                                             | ○  |                                                      |
|        | 基本命令   | 中核機能の文すべて                                                                        | ○  |                                                      |
|        | 環境定義   | APPLY WRITE-ONLY句<br>MULTIPLE FILE TAPE句<br>RERUN句<br>PASSWORD句<br>RESERVE AREA句 | ●  |                                                      |
|        |        | ASSIGN句のデータ名指定<br>ASSIGN句のDISK指定<br>ASSIGN句のPRINTER 指定<br>LOCK MODE句             | △  |                                                      |
|        |        | 上記以外                                                                             | ○  |                                                      |
|        | ファイル定義 | CODE-SET句                                                                        | ●  |                                                      |
|        |        | BLOCK CONTAINS句                                                                  | □  | 本システムでは、機能的に意味がありません。OSIV系システムで動作したプログラムは、そのまま動作します。 |
|        |        | 上記以外                                                                             | ○  |                                                      |
|        | 入出力文   | 入出力文のWITH LOCK指定<br>UNLOCK文                                                      | △  |                                                      |
|        |        | 上記以外                                                                             | ○  |                                                      |
|        | 制御レコード | フォームオーバーレイ                                                                       | □  | KOL2だけ。                                              |
| 行順ファイル |        | すべて                                                                              | △  |                                                      |
| 相対ファイル | 環境定義   | PASSWORD句<br>RERUN句                                                              | ●  |                                                      |
|        |        | ASSIGN句のデータ名指定<br>ASSIGN句のDISK指定<br>LOCK MODE句                                   | △  |                                                      |
|        |        | 上記以外                                                                             | ○  |                                                      |
|        | ファイル定義 | CODE-SET句                                                                        | ●  |                                                      |
|        |        | BLOCK CONTAINS句                                                                  | □  | 本システムでは、機能的に意味がありません。OSIV系システムで動作したプログラムは、そのまま動作します。 |
|        |        | 上記以外                                                                             | ○  |                                                      |
|        | 入出力文   | 入出力文のWITH LOCK 指定<br>UNLOCK文                                                     | △  |                                                      |
|        |        | 上記以外                                                                             | ○  |                                                      |
| 索引ファイル | 環境定義   | PASSWORD句<br>RERUN句                                                              | ●  |                                                      |
|        |        | ASSIGN句のデータ名指定<br>ASSIGN句のDISK指定<br>LOCK MODE句                                   | △  |                                                      |
|        |        | 1つのキーを複数のデータ項目で構成                                                                | □  | ESP IIIシステムおよびRDB II索引ファイルだけ。本システムでは条件付き。(注2)        |
|        |        | POSITIONING POINTER句                                                             | ×  |                                                      |

| 機能分類       |                       |                                               | 比較             | 備考                                                   |  |
|------------|-----------------------|-----------------------------------------------|----------------|------------------------------------------------------|--|
| 分類         |                       | 機能概要                                          |                |                                                      |  |
|            | ファイル定義                | 上記以外                                          | ○              |                                                      |  |
|            |                       | CODE-SET句                                     | ●              |                                                      |  |
|            |                       | BLOCK CONTAINS句                               | □              | 本システムでは、機能的に意味がありません。OSIV系システムで動作したプログラムは、そのまま動作します。 |  |
|            |                       | 上記以外                                          | ○              |                                                      |  |
|            | 入出力文                  | 入出力文のWITH LOCK 指定<br>UNLOCK文                  | △              |                                                      |  |
|            |                       | START文のPOSITIONING POINTER 指定                 | ×              | ESP IIIシステムだけ。                                       |  |
|            |                       | 上記以外                                          | ○              |                                                      |  |
| 整列併合機能     | 環境定義                  | ASSIGN句のファイル識別名定数                             | △              |                                                      |  |
|            | 呼び名                   | BUSHU<br>SOKAKU<br>ON-YOMI<br>KUN-YOMI        | ●              |                                                      |  |
|            |                       | 特殊レジスタ                                        | SORT-CORE-SIZE | ■                                                    |  |
|            |                       |                                               | SORT-MESSAGE   | ●                                                    |  |
|            |                       |                                               | 上記以外           | ○                                                    |  |
|            | その他                   | すべて                                           | ○              |                                                      |  |
| プログラム間連絡機能 | PROCEDURE<br>DIVISION | WITH指定                                        | △              |                                                      |  |
|            |                       | 日本語プログラム名                                     | △              |                                                      |  |
|            |                       | BY VALUE指定                                    | △              |                                                      |  |
|            |                       | RETURNING 指定                                  | △              |                                                      |  |
|            |                       | 上記以外                                          | ○              |                                                      |  |
|            | その他                   | すべて                                           | ○              |                                                      |  |
| 原始文操作      | COPY文                 | OF/IN のSYSDBDCT指定                             | ×              | ESP IIIシステムだけ。                                       |  |
|            |                       | OF/IN のXMDLIB、XFDLIB指定                        | △              |                                                      |  |
|            |                       | 登録集原文名定数                                      | △              |                                                      |  |
|            |                       | JOINING 指定だけの指定                               | △              |                                                      |  |
| 報告書作成      | ファイル定義                | BLOCK CONTAINS句<br>CODE 句                     | ●              |                                                      |  |
|            |                       | 上記以外                                          | ○              |                                                      |  |
|            | その他                   | すべて                                           | □              | 日本語項目の出力は正しく動作しません。                                  |  |
| 表示ファイル     | 環境定義                  | SYMBOLIC DESTINATION句<br>の"CMD"、"TRM"、"WST"指定 | □              | デバッガを使って動作確認を行うことができます。(注1)                          |  |
|            |                       | APPLY MULTICONVERSATION-<br>MODE句             | △              |                                                      |  |
|            |                       | PROCESSING TIME句                              | ●              |                                                      |  |
|            |                       | DESTINATION CONTROL句                          | ●              |                                                      |  |
|            |                       | MESSAGE SEQUENCE句                             | ●              |                                                      |  |

| 機能分類             |              |                | 比較     | 備考                                                |  |
|------------------|--------------|----------------|--------|---------------------------------------------------|--|
| 分類               |              | 機能概要           |        |                                                   |  |
|                  |              | 上記以外           | ○      |                                                   |  |
|                  | ファイル定義       | EXTERNAL句      | □      | OSIV系システムではINPUTまたはI-O指定のOPEN文で開かれるファイルには指定できません。 |  |
|                  | 入出力文         | すべて            | ○      |                                                   |  |
|                  | 特殊レジスタ       | すべて            | ○      |                                                   |  |
|                  |              | フォームオーバーレイ     | □      | KOL2だけ。                                           |  |
|                  |              | 画面帳票定義体        | □      | 使用できる機能範囲に注意してください。                               |  |
| デバッグ機能           |              | COUNT          | ○      |                                                   |  |
|                  |              | CHECK          | 指定なし   | ■                                                 |  |
|                  |              |                | EXTEND | ●                                                 |  |
|                  |              |                | 上記以外   | △                                                 |  |
|                  |              | TRACE          |        | △                                                 |  |
|                  |              | 上記以外           |        | ●                                                 |  |
| 区分化機能            |              | すべて            | ●      | OSIV系システムも翻訳だけ。                                   |  |
| 通信機能             |              | すべて            | ●      |                                                   |  |
| 拡張機能             | システム制御       | すべて            | ●      |                                                   |  |
|                  | ネットワークデータベース | すべて            | □      | デバッガを使って動作確認を行うことができます。(注1)                       |  |
|                  | AIM/RDB      | すべて            | ×      |                                                   |  |
|                  | SD機能         | すべて            | ○      |                                                   |  |
| 浮動小数点            |              | すべて            | ■      | OSIV系システムと内部表現が異なるため、演算結果が異なる場合があります。             |  |
| 組込み関数機能          |              | CUURENT-DATE関数 | ○      |                                                   |  |
|                  |              | 上記以外           | △      |                                                   |  |
| スクリーン操作機能        |              | すべて            | △      |                                                   |  |
| コマンド行引数／環境変数操作機能 |              | すべて            | △      |                                                   |  |
| オブジェクト指向機能       |              | すべて            | △      |                                                   |  |
| 翻訳オプションカスタマイズ機能  |              | すべて            | ×      |                                                   |  |
| ユーザテーラリング機能      |              | すべて            | ×      |                                                   |  |

注1：詳細は“OSIV分散開発の手引き”を参照してください。

注2：詳細は“7.5 索引ファイルの使い方”を参照してください。

## F.2 プログラムの動作確認における注意事項

共通の機能範囲内で作成したプログラムは、本システム上で動作確認を行うことができます。

しかし、機能によっては、プログラムの実行方法および実行結果が異なります。このような機能を使用しているOSIV系プログラムを本システム上で動作させるときの注意事項を、以下に説明します。

## プログラムの起動時にOSIV系システムのパラメタを渡す

本システム上でOSIV系システムのパラメタを渡すには、初期化ファイルを使って、パラメタの内容を指定します。  
初期化ファイルでOSIV系システムのパラメタを指定する方法は、“[5.3 実行環境情報の設定](#)”を参照してください。



### 例

COBOLプログラムの記述

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

- OSIV系システムでの入力コマンド

```
CALL 'X9999. A. LOAD (A)' 'ABCDE'
```

- OSIV系システムでのパラメタの内容

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 5 | A | B | C | D | E |
|---|---|---|---|---|---|

- 本システムでOSIV系システムと同様の結果を得るための初期化ファイルの内容

```
[A]
:
@MGPRM="ABCDE"
:
:
```



### 注意

本システムでは、OSIV系パラメタは、1つしか渡せません。

## OSIV系システム固有機能を使ったプログラムを本システム上で動作させる

OSIV系システム固有機能を使ったプログラムを本システム上で動作させるには、特別な操作が必要です。OSIV系システム固有機能を使ったプログラムを本システム上で動作させる方法を、以下に示します。

### 他の資源で代替する方法

通信機能を使ったプログラムを動作させる場合などは、順ファイルに対して入出力文を実行して動作を確認することができます。  
プログラムが意図したとおりに動作したかどうかは、順ファイルの内容で確認してください。

### 対話型デバッガを使って実行不可能な命令を迂回する方法

ファイル処理の対象となるファイルが存在しない場合や、副プログラムが存在しない場合などは、対話型デバッガを使って、入出力文およびCALL文を迂回することができます。

対話型デバッガの使用方法是、“[第20章 対話型デバッガの使い方](#)”を参照してください。



## 符号付き外部10進項目を比較対象にしているプログラムを翻訳時オプションNOZWBを指定して翻訳し動作させる

符号付き外部10進項目と英数字フィールド(英数字項目、英字項目、英数字編集項目、数字編集項目文字定数およびZERO以外の表意定数)を比較しているプログラムを、翻訳時オプションNOZWBを指定して翻訳した場合、OSIV系システムと本システムで、動作結果が異なる場合があります。

これは、データの内部表現形式が異なるためです。

翻訳時オプションZWBを指定して翻訳し、動作結果を確認してください。



### 例

COBOLプログラムの記述

```
IDENTIFICATION DIVISION.
:
WORKING-STORAGE SECTION.
:
01 A PIC S9(6) VALUE 200912.
01 B PIC 9(8) VALUE 20091201.
:
PROCEDURE DIVISION
 IF (A <= B(1:6))
:
:
```

- OSIV系システムでのデータの内部表現と比較結果  
A => F2F0F0F9F1C2 と B => F2F0F0F9F1F2 を比較  
比較結果 A < B
- 本システムでのデータの内部表現と比較結果  
A => 323030393142 と B => 323030393132 を比較  
比較結果 A > B



### 注意

- プログラム中で16進文字定数および日本語16進定数を使用する場合、コード系の考慮が必要です。
- 表示ファイル機能を使用する場合、COBOLソースプログラムを翻訳するときに、画面帳票定義体(本システム)またはフォーマット定義体(OSIV系システム)から取り込む表示レコードの展開形式が異なります。
- 機能名CHANNEL02～12およびC02～12を使用する場合、FCB制御文が必要となります。本システムで使用するFCB制御文の形式は、OSIV系システムのADJUSTで使用するFCB制御文の形式と同じです。
- フォームオーバーレイおよび画面帳票定義体は、動作させるOSによって、使用できる機能範囲が異なります。詳細は、FORMに添付されている文書の“OS別留意事項”(system.txt)を参照してください。
- 画面帳票定義体の移行については、“OSIV PSAM使用手引書”の“ホスト・ワークステーション連携”を参照してください。

## 付録G COBOLが提供するサブルーチン

ここでは、COBOLが提供するサブルーチンについて説明します。

### G.1 COBOLプログラムから呼び出すサブルーチン

ここでは、COBOLプログラムから呼び出すサブルーチンについて説明します。

#### G.1.1 ウィンドウハンドル獲得サブルーチン

サブルーチンJMPBGWDHを利用することによって、本システムで動作する表示ファイルの編集機能と連携して、MeFtのウィンドウハンドルを獲得することができます。

獲得したウィンドウハンドルをMeFtの関数の引数に指定することで、表示ファイルのREAD待ち状態の画面を、別プロセスのアプリケーションからの要求により、強制的に入力完了させることが可能となります。

指定方法

パラメタのデータ定義

|    |        |                   |
|----|--------|-------------------|
| 01 | データ名-1 | PIC X(8).         |
| 01 | データ名-2 | PIC S9(9) COMP-5. |

呼出しの記述

|      |                                 |        |        |
|------|---------------------------------|--------|--------|
| CALL | "JMPBGWDH" WITH C LINKAGE USING | データ名-1 | データ名-2 |
|------|---------------------------------|--------|--------|

インタフェース

- データ名-1には、ウィンドウハンドルを獲得する表示ファイルのファイル識別名を指定します。
- データ名-2には、サブルーチンによって通知されるウィンドウハンドルの格納域を指定します。
- データ名-1で指定した表示ファイルは、SYMBOLIC DESTINATION句で"DSP"を指定したものであり、すでにオープンされている必要があります。

メッセージ

以下の場合、メッセージが出力されます。

- 処理対象とする種類のファイルではない。
- 指定されたファイルが存在しない。

復帰コード

サブルーチンからの復帰コードは、特殊レジスタPROGRAM-STATUSを使用して受け取ります。

復帰値 0

ウィンドウハンドルの取得に成功しました。

復帰値-1

ウィンドウハンドルの取得に失敗しました。



注意

- ・ 表示ファイル単体テスト機能で動作する場合は、使用することはできません。
- ・ データ名-1に指定するファイル識別名は、実行単位内で単一のファイルに対応していなければなりません。複数のファイルに対応している場合は、実行結果は保証されません。

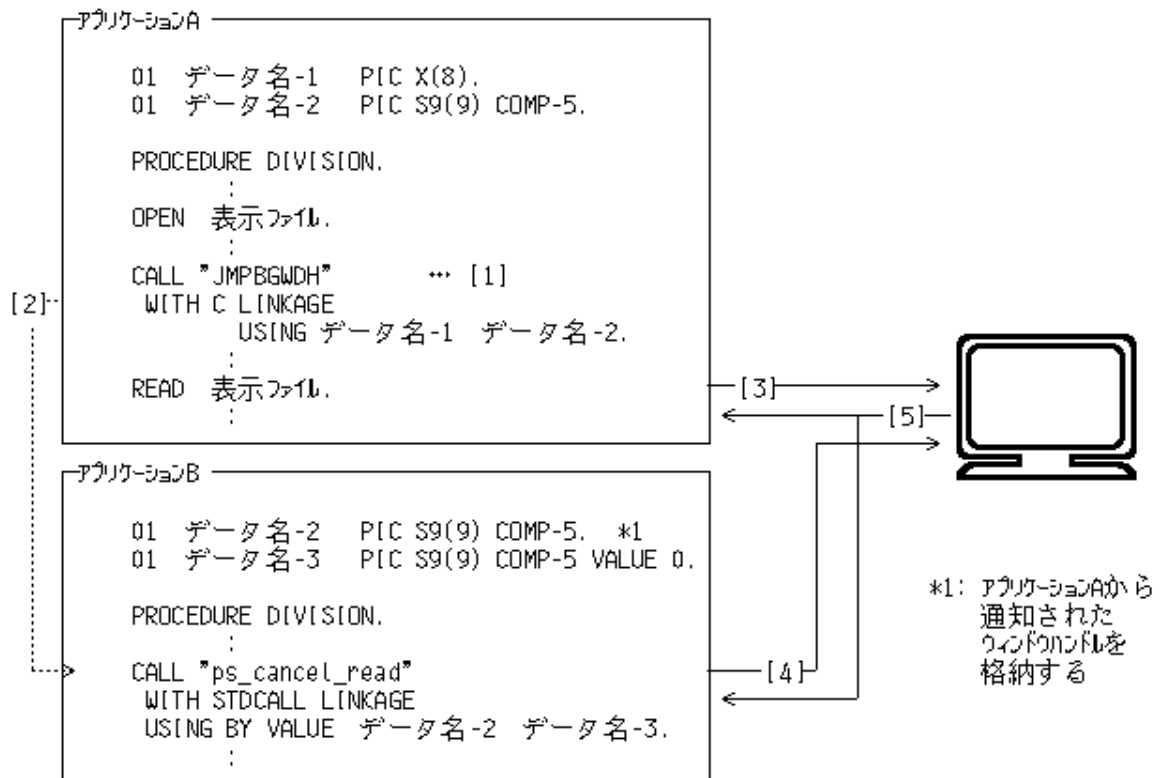
- 動的プログラム構造で当サブルーチンを呼び出す場合、以下のようなエントリ情報が必要になります。エントリ情報の指定方法については、“5.4.2 副プログラムのエントリ情報”を参照してください。

```
[ENTRY]
JMPBGWDH=F3BIILNG.DLL
```

- 動的リンク構造で当サブルーチンを呼び出す場合、呼出し元プログラムのリンク時に、F3BICIMP.LIBを結合してください。

## アプリケーションからの利用

ここでは、アプリケーションから利用するときの流れについて説明します。



- [1] ウィンドウハンドル獲得サブルーチンを使用して、ウィンドウハンドルを獲得します。
- [2] [1]で獲得したウィンドウハンドルをアプリケーションBに通知します。(プロセス間でのデータの受渡し)
- [3] 表示ファイルのREAD文を実行し、画面の入力待ち状態にします。
- [4] アプリケーションBでは、アプリケーションAから通知されたウィンドウハンドルをMeFtのps\_cancel\_read関数に指定し、画面の入力中断を指示します。ps\_cancel\_read関数については、“MeFtのオンラインマニュアル”を参照してください。
- [5] 画面の入力待ちを中断し、アプリケーションAには強制的に入力完了が行われたことを通知します。(注) アプリケーションBには、入力中断完了が通知されます。

注: 表示ファイルのREAD文の入出力状態値に“9E”、詳細情報に“9E5A”が通知されます。

MeFtのps\_cancel\_read関数の入口宣言は、WINAPIなので、STDCALL呼出し規約でCALL文を記述する必要があります。CALL文の呼出し規約については、“10.1.3 呼出し規約と指定形式”を参照してください。

## G.1.2 インスタンスハンドル獲得サブルーチン

サブルーチンJMPBWINSを利用することによって、Windowsで動作するCOBOLアプリケーションのインスタンスハンドルを獲得することができます。

## 指定方法

### パラメタのデータ定義

```
01 データ名.
02 データ名-1 PIC S9(9) COMP-5.
02 FILLER PIC X(12).
```

### 呼出しの記述

```
CALL "JMPBWINS" WITH C LINKAGE USING データ名.
```

## インタフェース

データ名には、サブルーチンによって通知されるインスタンスハンドルの格納域を指定します。インスタンスハンドルは、データ名-1に格納されます。



### 注意

- 動的プログラム構造で当サブルーチンを呼び出す場合、以下のようなエントリ情報が必要になります。エントリ情報の指定方法については、“[5.4.2 副プログラムのエントリ情報](#)”を参照してください。

```
[ENTRY]
JMPBWINS=F3BIPRCT.DLL
```

- 動的リンク構造で当サブルーチンを呼び出す場合、呼出し元プログラムのリンク時に、F3BICIMP.LIBを結合してください。

## G.1.3 イベントログ出力サブルーチン

サブルーチンCOB\_REPORT\_EVENTを利用することによって、利用者がCOBOLソースプログラム中に定義した文字列などの情報をイベントログに出力することができます。

## 指定方法

### パラメタのデータ定義

```
WORKING-STORAGE SECTION.
* USING 句に指定するデータ項目
01 データ名-1.
02 イベント番号 PIC 9(9) COMP-5.
02 種類 PIC 9(4) COMP-5.
02 FILLER PIC 9(4) COMP-5.
02 データ.
03 データ長 PIC 9(9) COMP-5.
03 データアドレス POINTER.
02 エラー詳細情報 PIC 9(9) COMP-5.
02 FILLER PIC 9(9) COMP-5.
02 ソース名 PIC X(256).
02 説明 PIC X(1024).

* RETURNING句に指定するデータ項目
01 データ名-2 PIC S9(9) COMP-5.

CONSTANT SECTION.
* 種類に格納する値
01 情報 PIC 9(4) COMP-5 VALUE 0.
01 警告 PIC 9(4) COMP-5 VALUE 1.
01 エラー PIC 9(4) COMP-5 VALUE 2.
```

## 呼出しの記述

```
INITIALIZE データ名-1.
CALL "COB_REPORT_EVENT" USING データ名-1
RETURNING データ名-2.
```

## インタフェース

- イベント番号には、イベントIDを0～999の値で指定します。
- 種類には0～2の値を指定します。情報の場合は0、警告の場合は1、エラーの場合は2を指定します。
- データ長には、出力するデータ領域の領域長を指定します。
- データアドレスには、出力するデータ領域の先頭アドレスを指定します。
- エラー詳細情報にはサブルーチンの処理が失敗した場合、エラーの詳細コードを返却する場合があります。
- ソース名には、スペースまたは出力先コンピュータに設定したレジストリキー名を指定します。スペースの場合、“NetCOBOL Application”になります。
- 説明には、文(文字列)を1024バイトの範囲で指定します。
- データ名-2には復帰コードを返却します。

## 復帰コード

サブルーチンからの復帰コードをデータ名-2に返却します。

### 復帰値 0

イベントログへの出力に成功しました。

### 復帰値 1

イベント番号の指定に誤りがあります。

### 復帰値 2

種類の指定に誤りがあります。

### 復帰値 3

OSが、32ビットWindowsではありません。

### 復帰値 99

上記以外の理由で失敗しました。データ名-1のエラー詳細情報に、システムのエラーコードが返却されます。“メッセージ説明書”の“付録A システムのエラーコードの説明”または“Visual C++のオンラインヘルプ”を参考にエラーの原因を取り除いてください。



## 注意

- データ名-1の初期化誤りによる誤動作を防止するために、データ名-1に値を設定する直前に、INITIALIZE文で初期化することをおすすめします。
- ネットワーク上の他のコンピュータ(32ビットWindows)に出力することができます。  
出力先コンピュータは実行時メッセージのイベントログ出力先と同じです。実行時メッセージ同様、出力先のコンピュータに必ず、COBOLをインストールしてください。  
詳細は、“[23.3.1 実行時メッセージをイベントログに出力する機能](#)”を参照してください。
- ソース名にデフォルト以外の任意の文字列を出力する場合、出力先コンピュータのレジストリに情報を設定する必要があります。本機能のためのレジストリ情報の設定および削除は“イベントログ出力サブルーチン用のレジストリキーの追加・削除”ツールを使用してください。  
なお、レジストリキー情報の設定および削除は、Administratorsグループのユーザなどレジストリキーのアクセス権(値の照会・値の設定・サブキーの作成・削除)のあるユーザが行ってください。  
NetCOBOLおよびNetCOBOL SNAPは、本製品により予約されているためソース名として使用できません。
- ソース名にはCOBOLのインタフェース上では最大256バイトまで指定できますが、システムの定量制限に依存します。

- ・ サブルーチンを呼び出すCOBOLプログラムの翻訳時にDLOADオプションを指定した場合、以下に示すエントリ情報が必要になります。エントリ情報の指定方法については、“5.4.2 副プログラムのエントリ情報”を参照してください。

```
[プログラム名. ENTRY]
COB_REPORT_EVENT=F3B1EFNC. DLL
```

## G.1.4 プロセスID取得サブルーチン

サブルーチンCOB\_GET\_PROCESSIDを利用することによって、本サブルーチンを呼び出しているプロセスのプロセスIDを取得することができます。

### 指定方法

#### パラメタのデータ定義

```
01 データ名 PIC 9(9) COMP-5.
```

#### 呼出しの記述

```
CALL "COB_GET_PROCESSID" USING BY REFERENCE データ名.
```

### インタフェース

データ名には、サブルーチンによって通知されるプロセスIDの格納域を指定します。



### 注意

- ・ サブルーチンを呼び出すCOBOLプログラムの翻訳時にDLOADオプションを指定した場合、以下のようなエントリ情報が必要になります。エントリ情報の指定方法については、“5.4.2 副プログラムのエントリ情報”を参照してください。

```
[プログラム名. ENTRY]
COB_GET_PROCESSID=F3B1EFNC. DLL
```

- ・ サブルーチンCOB\_GET\_PROCESSIDを呼び出した場合、特殊レジスタPROGRAM-STATUSが不定な値で更新されてしまいます。特殊レジスタPROGRAM-STATUSが更新されないようにするには、ダミーのデータ項目(PIC S9(9) COMP-5)をCALL文のRETURNING指定に記述してください。

## G.1.5 スレッドID取得サブルーチン

サブルーチンCOB\_GET\_THREADIDを利用することによって、本サブルーチンを呼び出しているスレッドのスレッドIDを取得することができます。

### 指定方法

#### パラメタのデータ定義

```
01 データ名 PIC 9(9) COMP-5.
```

#### 呼出しの記述

```
CALL "COB_GET_THREADID" USING BY REFERENCE データ名.
```

### インタフェース

データ名には、サブルーチンによって通知されるスレッドIDの格納域を指定します。



## 注意

- サブルーチンを呼び出すCOBOLプログラムの翻訳時にDLOADオプションを指定した場合、以下のようなエントリ情報が必要になります。エントリ情報の指定方法については、“5.4.2 副プログラムのエントリ情報”を参照してください。

```
[プログラム名. ENTRY]
COB_GET_THREADID=F3B1EFNC. DLL
```

- サブルーチンCOB\_GET\_THREADIDを呼び出した場合、特殊レジスタPROGRAM-STATUSが不定な値で更新されてしまいます。特殊レジスタPROGRAM-STATUSが更新されないようにするには、ダミーのデータ項目(PIC S9(9) COMP-5)をCALL文のRETURNING指定に記述してください。

## G.1.6 デッドロック出口スケジュールサブルーチン

NetCOBOLのデータベース機能(ODBC)またはプリコンパイラを利用してデータベースアクセスを行うプログラムにデッドロック事象が通知されたときに、USE FOR DEAD-LOCK文で記述したデッドロック出口に制御を戻す場合に使用します。デッドロック出口スケジュールについては、“22.2.14 デッドロック出口”を参照してください。

### 指定方法

#### パラメタのデータ定義

パラメタは必要ありません。

#### 呼出しの記述

```
CALL "COB_DEADLOCK_EXIT".
```

### 復帰コード

サブルーチンからの復帰コードはありません。

### 呼出し条件

データベースアクセスの実行によってSQLSTATEに復帰コードが通知されます。復帰コードにデッドロックを示す値が設定されているとき、デッドロック出口へプログラムの制御を戻すために呼び出します。



## 注意

- サブルーチンを呼び出したプログラムまたは上位のプログラムにデッドロック出口が記述されていない場合、デッドロック出口スケジュールは失敗し、実行時エラー(JMP0024I-U)を出力して異常終了します。
- サブルーチンを呼び出したプログラムからデッドロック出口を記述したプログラムの呼出しの間に他言語プログラムがある場合、他言語プログラムの回収処理は行われません。また、デッドロック出口で処理の再開を行う場合には、他言語プログラムに再入することになります。このため、他言語プログラムは再入可能かつ資源回収不要な構造である必要があります。
- サブルーチンをCOBOL以外の言語プログラムから呼び出した場合の動作は保証しません。
- 呼出し条件で示したデッドロック事象の発生の判断は、利用者の責任で判定処理を行う必要があります。
- デッドロック発生による対処を行う目的以外にサブルーチンを呼び出した場合の動作は保証しません。
- マルチスレッド環境での動作が可能です。

## G.1.7 メモリ割当てサブルーチン

COBOLでは、動的にメモリを割り当てる/解放する、以下のサブルーチンを提供しています。

| サブルーチン名          | 機能                  |
|------------------|---------------------|
| COB_ALLOC_MEMORY | 動的にメモリを割り当てる。       |
| COB_FREE_MEMORY  | 動的に割り当てられたメモリを解放する。 |

COB\_ALLOC\_MEMORYは、COBOLプログラムから動的にメモリの割当てを行うサブルーチンです。

このルーチンを使用して割り当てたメモリは、COB\_FREE\_MEMORYを呼び出すことで解放することができます。

COB\_FREE\_MEMORYを呼び出さない場合、メモリの解放のタイミングは以下ようになります。

| スレッドモード     | COB_ALLOC_MEMORYで指定するメモリの種別 | メモリ解放のタイミング |
|-------------|-----------------------------|-------------|
| シングルスレッドモード | —                           | 実行環境の閉鎖時    |
| マルチスレッドモード  | プロセス指定                      | 実行環境の閉鎖時    |
|             | スレッド指定                      | 実行単位の終了時    |

COB\_ALLOC\_MEMORYで指定するメモリの種別は、シングルスレッドモードのプログラムでは意味を持ちません。

マルチスレッドモードのプログラムでは、目的に応じてメモリの種別を選択してください。スレッド間でメモリを共有する場合、“プロセス指定”を選択します。メモリの使用範囲がスレッド内に閉じる場合、“スレッド指定”を選択します。

シングルスレッドモードとマルチスレッドモードについては、“[24.2.2 マルチスレッドプログラムとは](#)”を参照してください。

### シングルスレッドモードのプログラム

シングルスレッドモードのプログラムで、主プログラムがCOBOLの場合と他言語の場合について説明します。

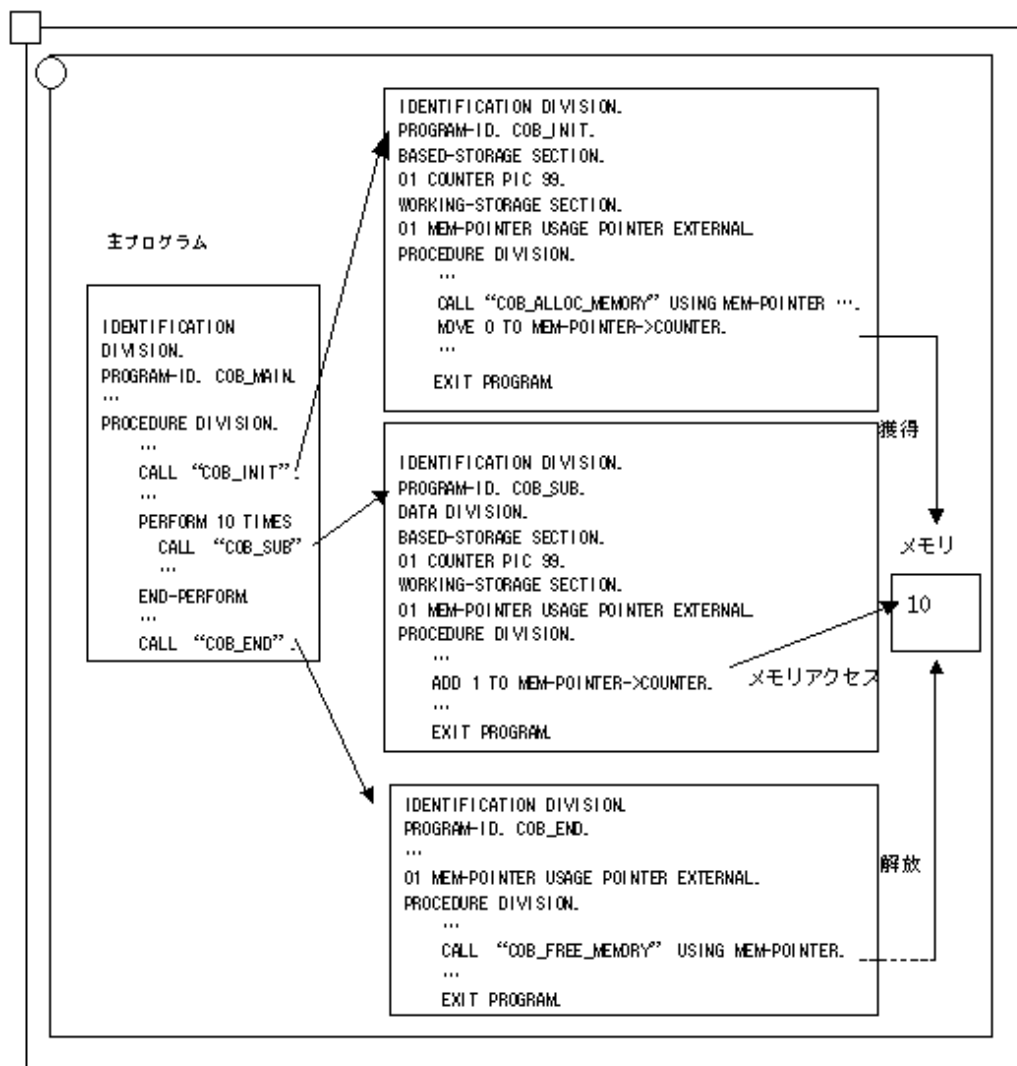
#### 主プログラムがCOBOLの場合

プログラムCOB\_INITで、サブルーチンCOB\_ALLOC\_MEMORYを呼び出してメモリを割り当てます。このとき、メモリの種別として「プロセス指定」と「スレッド指定」のどちらを選択しても動作に違いはありません。

割り当てたメモリは、同一実行単位上に存在するプログラムCOB\_SUBからアクセスできます。



図G.1 シングルスレッドモードにおけるメモリ割当て(主プログラムがCOBOL)



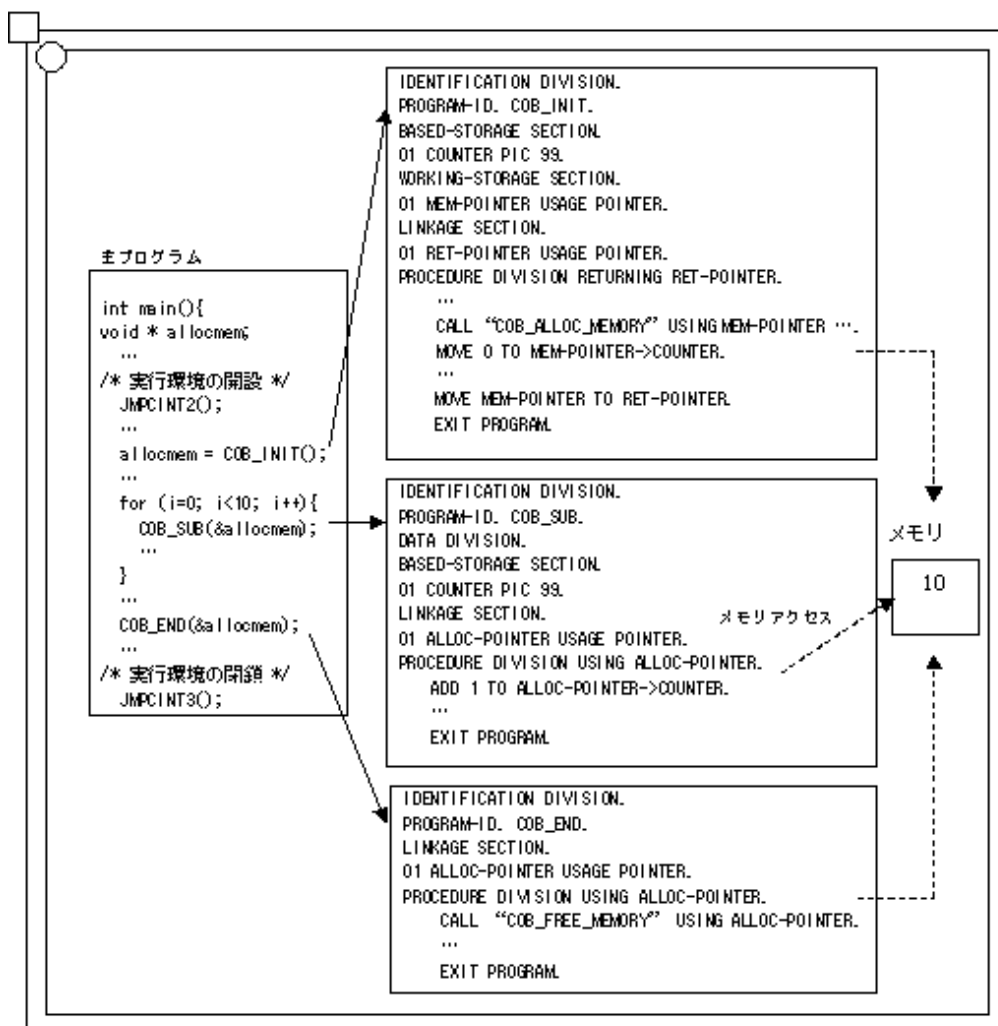
#### 主プログラムが他言語の場合

主プログラムが他言語の場合、JMPICINT2およびJMPICINT3を呼び出して複数のプログラムを同一の実行単位上で動作させることで、“図G.1 シングルスレッドモードにおけるメモリ割当て(主プログラムがCOBOL)”と同様の動きを実現することができます。

JMPICINT2およびJMPICINT3を呼び出さない場合、割り当てたメモリはプログラムCOB\_INITの終了時に解放されます。以降に呼び出されるCOB\_SUBからのメモリアクセスはできません。

JMPICINT2およびJMPICINT3の使用方法については、“G.2 他言語連携で使用するサブルーチン”を参照してください。

図G.2 シングルスレッドモードにおけるメモリ割当て(主プログラムが他言語)



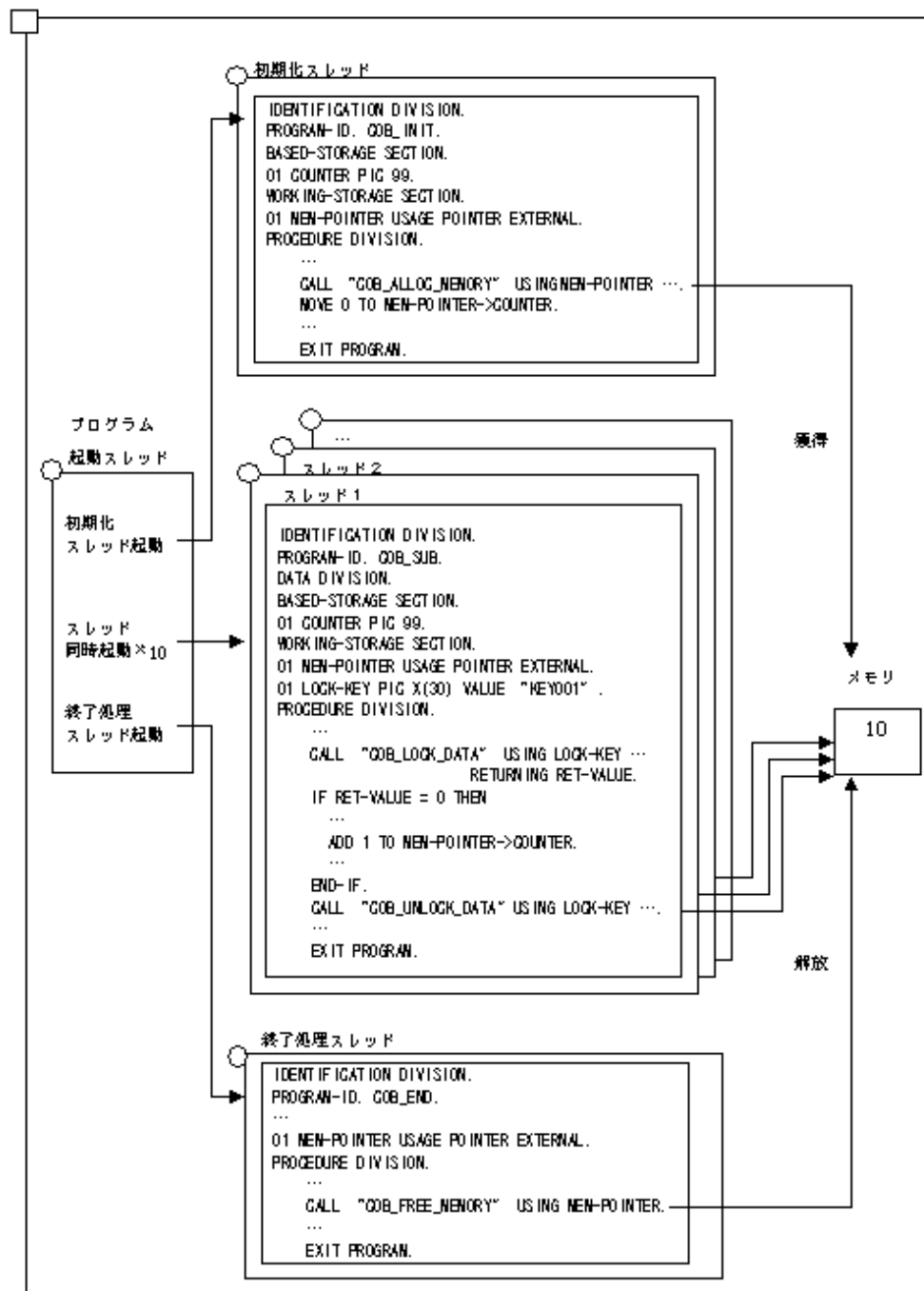
## マルチスレッドモードのプログラム

マルチスレッドモードのプログラムでは、メモリの種別として「プロセス指定」または「スレッド指定」を指定できます。ここでは、メモリの種別ごとの動作の違いについて説明します。

### プロセス指定

最初に呼び出される初期化スレッドのプログラムCOB\_INITで、COB\_ALLOC\_MEMORYを「プロセス指定」で呼び出して、メモリを割り当てます。ここで割り当てたメモリは、プロセスの終了まで解放されないため、スレッドの異なるプログラムCOB\_SUBからもアクセスすることができます。ただし、図のようにスレッド間でメモリを共有する場合、データロックサブルーチンによる同期制御が必要になります。スレッド間の同期制御についての詳細な説明および注意事項については、“[24.4 スレッド間の資源の共有](#)”を参考にしてください。

図G.3 プロセス指定(マルチスレッドモード)

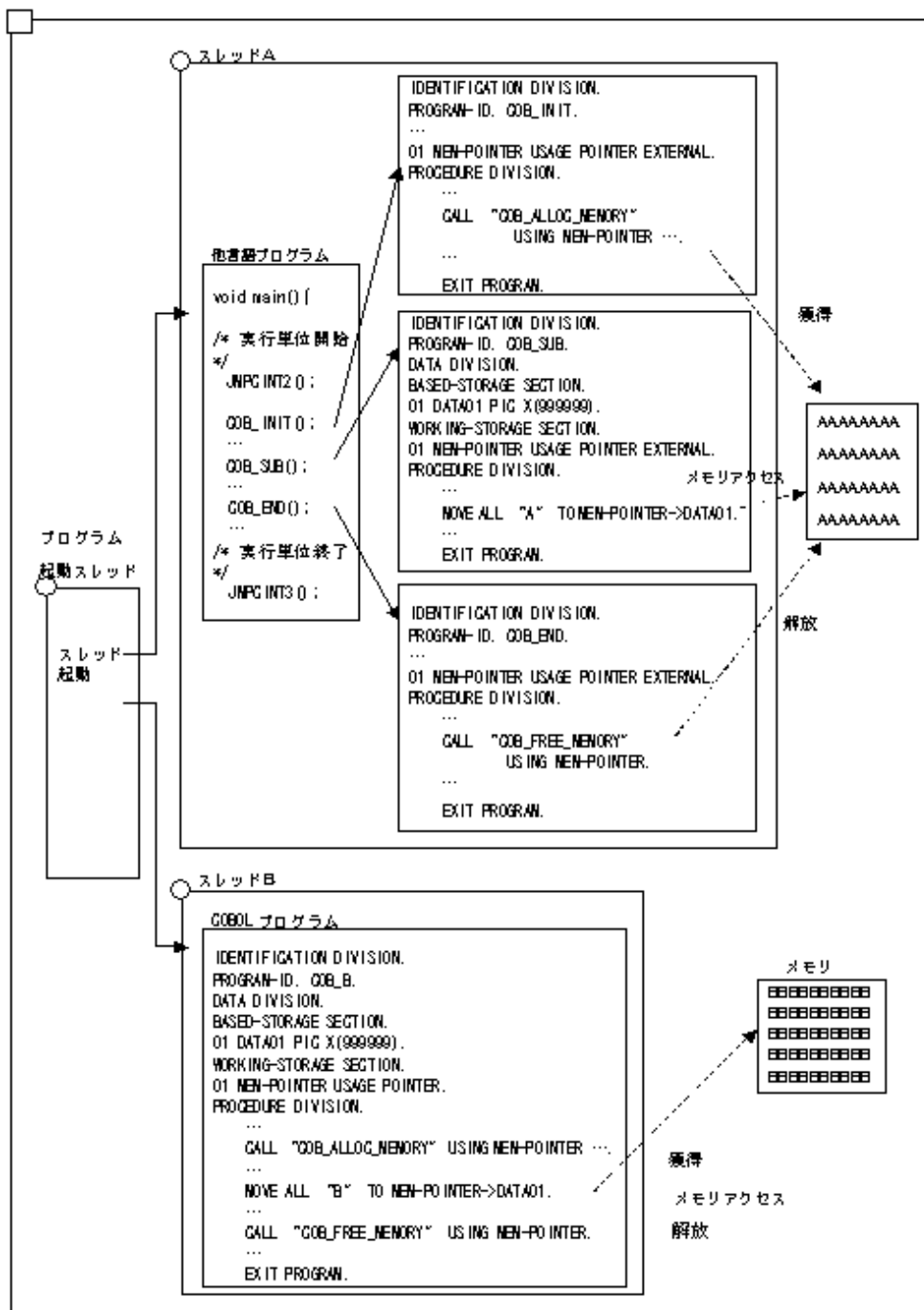


## スレッド指定

スレッドAのプログラムCOB\_INIT、スレッドBのプログラムCOB\_Bにおいて、それぞれCOB\_ALLOC\_MEMORYを「スレッド指定」で呼び出してメモリを割り当てます。

スレッドの主プログラムがCOBOLの場合には主プログラムの終了時、他言語の場合にはJMPCINT3の呼出し時にそれぞれ解放処理が行われます。「スレッド指定」は、メモリの使用範囲がスレッド内に閉じている場合に使用してください。

図G.4 スレッド指定(マルチスレッドモード)



## 注意

サブルーチンを呼び出すCOBOLプログラムの翻訳時にDLOADオプションを指定した場合、以下のようなエントリ情報が必要になります。エントリ情報の指定方法は、「[5.4.2 副プログラムのエントリ情報](#)」を参照してください。

[プログラム名. ENTRY]  
サブルーチン名=F3B1EFNC. DLL

## COB\_ALLOC\_MEMORY

サブルーチンCOB\_ALLOC\_MEMORYを利用することによって、動的にメモリを割り当てることができます。  
なお、割り当てられたメモリ域は初期化されません。

### 呼出し形式

```
CALL "COB_ALLOC_MEMORY" USING BY REFERENCE データ名－1
 BY VALUE データ名－2
 BY VALUE データ名－3
 RETURNING データ名－4
```

### パラメタの説明

#### データ名－1

01 データ名－1 USAGE POINTER.

サブルーチンによって割り当てられるメモリのアドレス格納域を指定します。

#### データ名－2

01 データ名－2 PIC 9(9) COMP-5.

割り当てるメモリのバイト数を指定します。

#### データ名－3

01 データ名－3 PIC 9(9) COMP-5.

メモリの種別を以下の値から指定します。

| 値 | 意味     |                                                              |
|---|--------|--------------------------------------------------------------|
| 0 | プロセス指定 | メモリをプロセス単位に割り当てます。解放ルーチンが呼び出されない場合、割り当てたメモリは実行環境の閉鎖時に解放されます。 |
| 1 | スレッド指定 | メモリをスレッド単位に割り当てます。解放ルーチンが呼び出されない場合、割り当てたメモリは実行単位の終了時に解放されます。 |

### 復帰値

#### データ名－4

01 データ名－4 PIC S9(9) COMP-5.

成功した時は0が返されます。失敗した場合の復帰値とエラーの原因は以下のとおりです。

| 値  | 意味               |
|----|------------------|
| -1 | パラメタの指定に誤りがあります。 |
| -2 | メモリが不足しています。     |
| -3 | 実行単位が既に終了しています。  |

## COB\_FREE\_MEMORY

サブルーチンCOB\_FREE\_MEMORYを利用することによって、動的に割り当てたメモリを解放することができます。

### 呼出し形式

```
CALL "COB_FREE_MEMORY" USING BY REFERENCE データ名－1
 RETURNING データ名－2
```

## パラメタの説明

### データ名-1

|                          |
|--------------------------|
| 01 データ名-1 USAGE POINTER. |
|--------------------------|

サブルーチンCOB\_ALLOC\_MEMORYによって割り当てられたメモリのアドレスを指定します。

## 復帰値

### データ名-2

|                             |
|-----------------------------|
| 01 データ名-2 PIC S9(9) COMP-5. |
|-----------------------------|

成功した時は0、失敗した時は-1が返されます。失敗した場合、エラーの原因として以下が考えられます。

- 指定したメモリが既に解放されている、または破壊されている。
- サブルーチンCOB\_ALLOC\_MEMORYを使用して割り当てたメモリでない。



## 注意

スレッド指定で割り当てたメモリを解放する場合、必ずメモリを割り当てた実行単位内で解放処理を行うようにしてください。COB\_ALLOC\_MEMORYを呼び出した実行単位とCOB\_FREE\_MEMORYを呼び出した実行単位が異なる場合、メモリの解放処理は失敗します。

## G.1.8 プロセス終了サブルーチン

COBOLでは、プロセスを強制的に終了させるサブルーチンを提供しています。

プロセスの終了方法には、以下の2つの方法があります。

### プロセスを正常終了する

プロセスを正常に終了させ、親プロセスに指定した値を返却します。  
サブプログラムから親プロセスに値を返却したい場合に使用すると便利です。

### プロセスを異常終了する

アプリケーションエラーを発生させ、プロセスを異常終了させます。

診断機能が有効な場合、出力される診断レポートから、エラーの発生箇所やプログラムの呼出し関係およびアプリケーションの状態を確認することができます。また、診断機能を無効とした場合、通常のアプリケーションエラーと同じように、以下の機能を使用することができます。

- Visual C++のジャストインタイムデバッガ
- ワトソン博士(Windows XPまたはWindows Server 2003の場合)
- Windows エラー報告(Windows 7、Windows 8の、Windows Vista、Windows Server 2008、およびWindows Server 2012の場合)



## 例

### 診断レポートの例

## NetCOBOL 診断機能

### <<要約>>

エラーが発生しました:

アプリケーション : D:\WPLKSAMPLE.exe (PID=000009A0)

例外種別 : EXCEPTION\_NONCONTINUABLE\_EXCEPTION (C0000025)

発生時間 : YYYY/MM/DD (HH:MM:SS)

発生ユーザ : C:\WINDOWS\system32\ntdll.dll

作成時間 : YYYY/MM/DD (HH:MM:SS)

ファイルサイズ : 629760bytes

### <<問題箇所>>

バイト ID : 00000B8C

レジスタ : EAX=0012F9E0 EBX=0012FFB0 ECX=00000000 EDI=0056D6A0 ESI=0012FA5C

EDI=00000000 EIP=7C98EAF8 ESP=0012F9D4 EBP=0012FA44 EPL=00000202

CS=001B SS=0023 DS=0023 ES=0023 FS=003B GS=0000

オペランド : 00000000 (byte: 00130000, 0x130000)

命令 : JNB +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +a +b +c +d +e +f

7C98EAF8 A0 01 00 00 89 75 A4 89 4D AC EB B2 00 FC FF

ターゲット -> 7C98EAF8 E9 12 BF FD FF 83 A5 E4 FC FF FF 00 8D 85 D4 FC

発生ユーザ : C:\WINDOWS\system32\ntdll.dll

アドレス相対位置 : .text+0004D9FA

EXPORT 相対位置 : RtlInitializeListHead+00015AE8

### <<呼出経路>>

- [ 1 ]-----  
発生ユーザ : C:\WINDOWS\system32\ntdll.dll  
アドレス相対位置 : .text+0000D9FA  
EXPORT 相対位置 : KiUserExceptionDispatcher+0000000E
- [ 2 ]-----  
発生ユーザ : C:\Program Files\NetCOBOL\F361\EFNC.dll  
アドレス相対位置 : .text+0000130B  
EXPORT 相対位置 : COB\_EXIT\_PROCESS+0000005B
- [ 3 ]-----  
発生ユーザ : D:\WPLKSAMPLE.dll  
アドレス相対位置 : .text+000001DE  
EXPORT 相対位置 : SAMPLED1+000001A2
- [ 4 ]-----  
発生ユーザ : D:\WPLKSAMPLE.exe  
アドレス相対位置 : .text+000001C9  
アドレス : 004011C9
- [ 5 ]-----  
発生ユーザ : D:\WPLKSAMPLE.exe  
アドレス相対位置 : .text+0000000A  
アドレス : 0040100A
- [ 6 ]-----  
発生ユーザ : D:\WPLKSAMPLE.exe  
アドレス相対位置 : .text+0000031E  
アドレス : 0040131E
- [ 7 ]-----  
発生ユーザ : C:\WINDOWS\system32\USER32.dll  
アドレス相対位置 : .text+00015FD7  
EXPORT 相対位置 : RegisterWaitForInputidle+00000049  
(以降、省略)



## 注意

サブルーチンCOB\_EXIT\_PROCESSは、アプリケーションにおいて重大な問題が検出されたときのみ、使用することをお勧めします。特にマルチスレッド環境では、当ルーチンは他のスレッドの終了を待たずに強制的にプロセスを終了させます。このため、他のスレッドで問題が発生する場合があります。

なお、サブルーチンCOB\_EXIT\_PROCESSの呼出しでは、ファイルの強制クローズが行われない場合があります。できる限り、オープン中のファイルはクローズしてから、当ルーチンを呼び出すようにしてください。

## COB\_EXIT\_PROCESS

サブルーチンCOB\_EXIT\_PROCESSを利用することによって、プロセスを強制的に終了させることができます。

## 呼出し形式

```
CALL "COB_EXIT_PROCESS" USING BY VALUE データ名-1
 BY VALUE データ名-2
 RETURNING データ名-3
```

## パラメタの説明

### データ名-1

01 データ名-1 PIC 9(9) COMP-5.

プロセスの終了方法を以下の値から指定します。

| 値 | 意味                                    |
|---|---------------------------------------|
| 0 | プロセスを正常に終了させ、データ名-2に指定した値を親プロセスに返します。 |
| 1 | アプリケーションエラーを発生させ、プロセスを異常終了します。        |

### データ名-2

01 データ名-2 PIC 9(9) COMP-5.

データ名-1に0を指定した場合、親プロセスに返却する値を指定します。データ名-1に1を指定した場合は、無効になります。返却できる値の範囲は、0から255です。これを超える値を入力した場合は、下位1バイトのみを有効にします。

## 復帰値

### データ名-3

01 データ名-3 PIC S9(9) COMP-5.

成功した時は値を返却しません。パラメタの指定に誤りがあった場合、-1を返却します。

## G.2 他言語連携で使用するサブルーチン

ここでは、他言語連携用にCOBOLが提供するサブルーチンについて説明します。

### G.2.1 実行単位の開始サブルーチン

他言語のプログラムから複数のCOBOLプログラムを呼び出す場合、同一の実行単位上でCOBOLプログラムを動作させる場合に使用します。

#### 指定方法

呼出しの記述(C言語での呼び出し方)

```
型宣言部 :
 extern void JMPCINT2(void);

手続き部 :
 JMPCINT2();
```



#### 注意

Visual C++用に翻訳するソースから呼び出す場合、型宣言部は以下のようになしてください。

```
extern "C" void JMPCINT2(void);
```

#### インタフェース

呼出し時にパラメタは必要ありません。



## 復帰コード

サブルーチンからの復帰コードはありません。

## 実行単位の開始サブルーチン使用時の注意事項

当サブルーチン呼び出した場合、実行単位の閉鎖時に、JMPCINT3サブルーチンを必ず呼び出してください。  
実行単位の開始については、“[10.1.4 COBOLの言語間の環境](#)”および“[24.3.1 実行環境と実行単位](#)”を参照してください。



### 例

```
extern void JMPCINT2(void);
extern void JMPCINT3(void);
extern int COBSUB1(void); ← COBOL プログラム
extern int COBSUB2(void); ← COBOL プログラム

int csub(void) {
 JMPCINT2(); ← 実行単位の開始
 COBSUB1(); ← COBOL プログラム
 COBSUB2(); ← COBOL プログラム
 JMPCINT3(); ← 実行単位の閉鎖
}
```

## G.2.2 実行単位の終了サブルーチン

実行単位の開始サブルーチンを使用した際に、実行単位を閉鎖させる場合に他言語のプログラムから使用します。

### 指定方法

呼出しの記述(C言語での呼び出し方)

```
型宣言部 :
extern void JMPCINT3(void);

手続き部 :
JMPCINT3();
```



### 注意

Visual C++用に翻訳するソースから呼び出す場合、型宣言部は以下のようになしてください。

```
extern "C" void JMPCINT3(void);
```

### インタフェース

呼出し時にパラメタは必要ありません。

## 復帰コード

サブルーチンからの復帰コードはありません。

## 実行単位の終了サブルーチン使用時の注意事項

当サブルーチン呼び出す前に、必ずJMPCINT2サブルーチン(実行単位の開始サブルーチン)を呼び出してください。  
実行単位の閉鎖については、“[10.1.4 COBOLの言語間の環境](#)”および“[24.3.1 実行環境と実行単位](#)”を参照してください。



### 例

```
extern void JMPCINT2(void);
extern void JMPCINT3(void);
extern int COBSUB1(void); ← COBOL プログラム
```

```
extern int COBSUB2(void); ← COBOL プログラム

int csub(void) {
 JMPCINT2(); ← 実行単位の開始
 COBSUB1(); ← COBOL プログラム
 COBSUB2(); ← COBOL プログラム
 JMPCINT3(); ← 実行単位の閉鎖
}
```

## G.2.3 実行環境の閉鎖サブルーチン

プロセス内のすべての実行単位が終了した状態で、他言語プログラムからJMPCINT4を呼び出すことにより、実行環境を閉鎖することができます。

### 指定方法

呼出しの記述(C言語での呼び出し方)

```
型宣言部 :
extern void JMPCINT4(void);

手続き部 :
JMPCINT4();
```



Visual C++用に翻訳するソースから呼び出す場合、型宣言部は以下のようになさってください。

```
extern "C" void JMPCINT4(void);
```

### インタフェース

呼出し時にパラメタは必要ありません。

### 復帰コード

サブルーチンからの復帰コードはありません。

### 実行環境の閉鎖サブルーチン使用時の注意事項

当サブルーチンを呼び出す前に、プロセスのすべてのスレッドでCOBOLプログラムの実行が終了している必要があります。COBOLプログラムの実行中に当サブルーチンが呼び出されると、実行中のCOBOLプログラムは異常終了します。

このため、当サブルーチンを呼び出す前に、JMPCINT2サブルーチン(実行単位の開始サブルーチン)を呼び出している場合は、必ずJMPCINT3サブルーチン(実行単位の閉鎖サブルーチン)を呼び出して実行単位の閉鎖を行ってください。

実行環境の閉鎖については、“[10.1.4 COBOLの言語間の環境](#)”および“[24.3.1 実行環境と実行単位](#)”を参照してください。

シングルスレッドプログラムでは、実行単位の終了時に実行環境も閉鎖します。このため、シングルスレッドプログラムで実行単位の終了後にJMPCINT4サブルーチンを呼び出しても、何もしないで復帰します。



## 例

```
extern int WINAPI COBSUB(void);
extern void JMPCINT4(void);

int main(void){
 for (i=0; i<THREADNUM; i++) {
 CreateThread(,,COBSUB,,,);
 }
 WaitForMultipleObjects(...);
 JMPCINT4();
 :
 その他の COBOL 以外の処理
 :
 return 0;
}
```

IDENTIFICATION DIVISION.  
PROGRAM-ID. COBSUB.  
PROCEDURE DIVISION  
 WITH STDALL LINKAGE.  
 DISPLAY "HELLO!".  
 EXIT PROGRAM.

実行環境の閉鎖  
(COBOL のリソースの解放)

## 付録H オブジェクト指向と従来機能の組合せ

COBOLのオブジェクト指向では、従来のCOBOLで使用していた機能であっても、クラス定義、メソッド定義の中では、使用できないものがあります。

ここでは、クラス定義および分離翻訳されるメソッド定義で使用できない機能について説明します。

### H.1 クラス定義で使えない機能

クラス定義、およびそれに含まれるファクトリ定義、オブジェクト定義、メソッド定義中で使用できない機能を、下表に示します。

| 機能                  | 説明                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ANSI'85 規格の廃要素      | 以下の機能は、ANSI'85 規格の廃要素です。これらの機能をクラス定義中使用することはできません。 <ul style="list-style-type: none"><li>• ALL 定数と数字項目または数字編集項目との関連付け</li><li>• AUTHOR段落、INSTALLATION段落、DATE-WRITTEN段落、DATE-COMPILED 段落およびSECURITY段落</li><li>• RERUN句</li><li>• MULTIPLE FILE TYPE句</li><li>• LABEL RECORD句</li><li>• VALUE OF句</li><li>• DATA RECORDS句</li><li>• ALTER文</li><li>• ENTER文</li><li>• 手続き名-1を省略したGO TO文</li><li>• OPEN文のREVERSED指定</li><li>• 定数指定のSTOP文</li></ul> |
| 定数による名前の指定          | CLASS-ID段落のクラス名およびMETHOD-ID 段落のメソッド名には、定数を指定できません。                                                                                                                                                                                                                                                                                                                                                                                              |
| 翻訳用計算機段落および実行用計算機段落 | クラス定義の環境部構成節には、翻訳用計算機段落および実行用計算機段落を指定できません。                                                                                                                                                                                                                                                                                                                                                                                                     |
| APPLY句              | ファクトリ定義、オブジェクト定義およびメソッド定義の環境部構成節入出力管理段落には、APPLY MULTICONVERSATION-MODE句およびAPPLY SAVED-AREA句を指定できません。                                                                                                                                                                                                                                                                                                                                            |
| 報告書作成機能             | ファクトリ定義、オブジェクト定義およびメソッド定義のデータ部には、報告書節を書くことはできません。                                                                                                                                                                                                                                                                                                                                                                                               |
| スクリーン操作機能           | ファクトリ定義、オブジェクト定義およびメソッド定義では、スクリーン操作機能を使用できません。                                                                                                                                                                                                                                                                                                                                                                                                  |
| CHARACTER TYPE句     | ファクトリ定義およびオブジェクト定義のデータ部では、CHARACTER TYPE句を指定できません。ただし、メソッド定義のデータ部では指定できます。また、メソッド原型定義の連絡節には、CHARACTER TYPE句を指定できません。                                                                                                                                                                                                                                                                                                                            |
| EXTERNAL句           | ファクトリ定義およびオブジェクト定義のデータ部では、EXTERNAL句を指定できません。ただし、メソッド定義のデータ部では指定できます。                                                                                                                                                                                                                                                                                                                                                                            |
| GLOBAL句             | ファクトリ定義、オブジェクト定義およびメソッド定義のデータ部には、GLOBAL句を指定できません。ファクトリ定義およびオブジェクト定義のデータ部で宣言された名前は、すべて大域名として扱われます。                                                                                                                                                                                                                                                                                                                                               |
| LINAGE句             | ファクトリ定義およびオブジェクト定義で定義したファイルには、LINAGE句を指定できません。メソッド定義で定義したファイルには指定できます。ただし、EXTERNAL句を指定したファイルには指定できません。                                                                                                                                                                                                                                                                                                                                          |

| 機能                       | 説明                                          |
|--------------------------|---------------------------------------------|
| PRINTING POSITION 句      | メソッド原型定義の連絡節には、PRINTING POSITION 句を指定できません。 |
| 特殊レジスタ<br>PROGRAM-STATUS | メソッド定義では、特殊レジスタPROGRAM-STATUSを使用できません。      |
| ENTRY文                   | メソッド定義の手続き部には、ENTRY 文を書くことはできません。           |
| ODBC                     | ファクトリ定義では、ODBCドライバ経由のデータベースアクセス機能は使用できません。  |

## H.2 分離翻訳されるメソッド定義で使えない機能

分離翻訳されるメソッド定義中で使えない機能を、下表に示します。

ここでは、分離翻訳されるメソッド定義内でだけ使えない機能を説明しています。“[H.1 クラス定義で使えない機能](#)”に示した機能のうち、メソッド定義内で使えない機能は、分離翻訳されるメソッド定義の場合でも使用できません。

| 機能                  | 説明                                                                                                                                                                                                                                                                                                                                |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 翻訳用計算機段落および実行用計算機段落 | 分離翻訳されるメソッド定義の環境部構成節には、翻訳用計算機段落および実行用計算機段落を指定できません。                                                                                                                                                                                                                                                                               |
| 特殊名段落               | 分離翻訳されるメソッド定義の環境部構成節には、特殊名段落を指定できません。                                                                                                                                                                                                                                                                                             |
| WRITE文のADVANCING指定  | <p>分離翻訳されるメソッド定義の手続き部で、ファクトリ定義またはオブジェクト定義で宣言されたファイルに対するWRITE文にADVANCING 指定を書く場合、以下の条件のどれかを満たしている必要があります。</p> <ul style="list-style-type: none"> <li>• ASSIGN句にPRINTER またはPRINTER-n (nは1～9までの整数) が指定されている。</li> <li>• ファイルを定義したソース単位に含まれるソース単位に、そのファイルに対するADVANCING 指定付きのWRITE文が指定されている。</li> <li>• FORMAT句付き印刷ファイルである。</li> </ul> |

## 付録I その他のコマンドの形式

ここでは、COBOLで提供している翻訳コマンド、リンクコマンド以外のコマンドについて説明します。

### I.1 DUMPBINコマンド

実行可能ファイル、DLL、オブジェクトファイル、インポートライブラリ、標準ライブラリ内の情報を調べるためには、DUMPBINコマンドが使用できます。

ここでは、DUMPBINコマンドの形式について説明します。

#### 入力形式

| コマンド    | オペランド            |
|---------|------------------|
| DUMPBIN | [オプションの並び] ファイル名 |

#### オペランドの説明

各オプションおよびファイル名の間には、1つ以上の空白が必要です。[]で囲んである部分は、省略できることを示します。なお、以降の説明で、ファイル名には、絶対パス名または相対パス名を指定することができます。

#### オプションの並び

DUMPBINコマンドのオプションを指定します。指定する内容については、“[表I.1 DUMPBINコマンドのオプション](#)”を参照してください。

表I.1 DUMPBINコマンドのオプション

| 内容                                                                                             | 指定形式         |
|------------------------------------------------------------------------------------------------|--------------|
| 実行可能ファイルまたはDLLからエクスポートされる定義をすべて表示します。この指定により、実行可能ファイルまたはDLLに含まれる外部参照名(プログラム名や関数名)を表示することができます。 | /EXPORTS     |
| 実行可能ファイルまたはDLLにインポートされる定義をすべて表示します。この指定により、実行可能ファイルやDLLから呼び出す外部参照名(プログラム名や関数名)を表示することができます。    | /IMPORTS     |
| コードの逆アセンブル以外の表示できる情報をすべて表示します。                                                                 | /ALL         |
| 実行結果の出力先ファイル名を指定します。                                                                           | /OUT:出力ファイル名 |



#### 参考

DUMPBINコマンドの使用方法的説明は、以下のようにコマンドだけを指定すれば参照することができます。

DUMPBIN

#### ファイル名

調査の対象となるファイル名を指定します。

指定可能なファイルを以下に示します。

- 実行可能ファイル
- DLL
- オブジェクトファイル
- インポートライブラリファイル
- 標準ライブラリファイル



## 例

呼び出すプログラム名が呼出し先のDLLにあるかどうか調べる場合

```
DUMPBIN /EXPORTS /OUT:出力ファイル名 DLL名
```

呼び出すプログラム名が呼出し先のDLLに対応するインポートライブラリにあるかどうか調べる場合

```
DUMPBIN /ALL /OUT:出力ファイル名 インポートライブラリファイル名
```

呼び出すプログラム名が実行可能ファイルにあるかどうか調べる場合

```
DUMPBIN /IMPORTS /OUT:出力ファイル名 実行可能ファイル名
```

## I.2 INSDBINFコマンド

これまでCOBOLコンパイラとプリコンパイラ連携では、以下のような問題がありました。

- ・コンパイラの出力する翻訳エラー検出行番号が、中間ファイル(プリコンパイル後のソースファイル)の行番号であるため、利用者は中間ファイルを参照しながらオリジナルソースプログラム(プリコンパイル前の、埋込みSQL文が書かれたCOBOLソースプログラム)の修正を行わなければならない。
- ・ビルダのエラージャンプ機能で、オリジナルソースプログラムに対してエラージャンプすることができない。
- ・オリジナルソースプログラムを被デバッグプログラムとして、デバッガでデバッグすることができない。

これらの問題を解決するため、INSDBINFコマンドでは行番号制御情報およびファイル名制御情報を埋め込んだ中間ファイルを生成します。

行番号制御情報(#LINE情報)は、プリコンパイルを行う前のソースの行番号を、それ以降に動作するコンパイラまたはプリコンパイラに通知するための情報です。

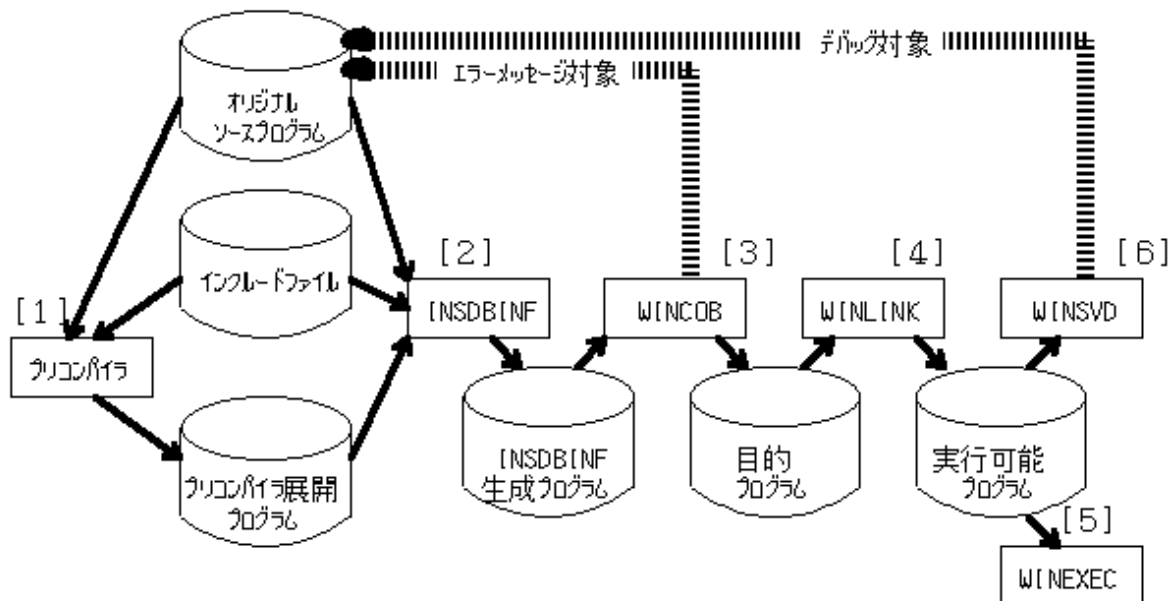
ファイル名制御情報(#FILE情報)は、オリジナルソースプログラムのファイル名や、プリコンパイラがインクルードしたファイルのファイル名を通知するための情報です。

COBOLコンパイラは、INSDBINFコマンドが生成した中間ファイルを入力ファイルとすることにより、行番号制御情報およびファイル名制御情報を参照できます。そして、それらの情報からオリジナルソースファイルとプリコンパイル後のソースファイル行番号の対応付けたオリジナルソースプログラムおよびインクルードファイルのファイル名の取得が可能となります。これにより、COBOLコンパイラとプリコンパイラの連携時のこれまでの問題が解決されることになります。

INSDBINFコマンドを使用した場合のデータベース連携プログラムの開発手順と、コマンドの形式について説明します。

### プログラム開発手順

INSDBINFコマンドを使用した場合のデータベース連携プログラム開発の標準的な手順を以下に示します。



[1] プリコンパイラで、埋込みSQL文が記述されたCOBOLソースプログラム(オリジナルソースプログラム)をプリコンパイルします。

[2] INSDBINFコマンドを実行します。

[3] WINCOBを実行し、プログラムを翻訳します。翻訳エラーが発生した場合、エラージャンプ機能を使って、エラーの発生したCOBOLの文を修正します。このとき、エラージャンプ機能によって表示されるのは、オリジナルソースプログラムです。



#### 注意

プログラム翻訳時、翻訳オプションNONUMBERを有効にして翻訳してください。

[4] WINLINKを実行し、プログラムのリンクを行います。

[5] WINEXECを実行し、プログラムを実行します。

[6] プログラムが期待どおり動かない場合や、プログラムの単体テストを行う場合、対話型デバッガを使ってデバッグを行います。このとき、オリジナルソースプログラムが被デバッグプログラムとなります。

なお、プロジェクトマネージャでは、INSDBINFコマンドを使用したプリコンパイルの実行を行うことができます。詳細については、“[6.4.8 プリコンパイラの登録とプリコンパイルの実行](#)”を参照してください。

## 入力形式

| コマンド     | オペランド                          |
|----------|--------------------------------|
| INSDBINF | [オプションの並び] 入力ファイル名1 [入力ファイル名2] |

## オペランドの説明

各オプションおよびファイル名の間には、1つ以上の空白が必要です。[]で囲んである部分は、省略できることを示します。なお、以降の説明で、ファイル名には、絶対パス名または相対パス名を指定することができます。

### オプションの並び

INSDBINFコマンドのオプションを指定します。指定する内容については、“[表1.2 INSDBINFコマンドのオプション](#)”を参照してください。



表I.2 INSDBINFコマンドのオプション

| 内容                                                                                                                                                                                                                                                   | 指定形式                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|
| プリコンパイラが処理するインクルードファイルが存在する場合、 <b>-I</b> オプションにインクルードファイルが存在するフォルダを指定します。インクルードファイルが複数のフォルダに存在する場合、 <b>-I</b> オプションを複数指定します。 <b>-I</b> オプションを複数指定した場合、指定した順序でフォルダが検索されます。インクルードファイルフォルダを省略した場合、または、 <b>-I</b> オプションを省略した場合は、カレントフォルダが検索されます。         | <b>-I</b> インクルードファイルフォルダ名 |
| プリコンパイラが処理するインクルードファイルの拡張子を指定します。拡張子を複数指定した場合、指定された順序で検索が行われます。ファイル名に拡張子がない場合は、拡張子の代わりに文字列 <b>None</b> を指定します。 <b>-S</b> オプションを省略した場合は、(1) 拡張子なし、(2) COB の順序で検索が行われます。                                                                               | <b>-S</b> 拡張子[,拡張子] …     |
| 当コマンドの生成プログラムの出力先ファイル名を指定します。 <b>-O</b> オプションを省略した場合、入力ファイル名1のファイル拡張子をCOB に置き換えたファイル名が指定されたとして扱われます。                                                                                                                                                 | <b>-O</b> 出力ファイル名         |
| ホスト変数の用途に指定されたBINARY、COMPおよびCOMPUTATIONALを、プリコンパイラがCOMP-5またはCOMPUTATIONAL-5に展開しない場合に指定します(Pro*COBOLの場合、comp5オプションにnoを指定して出力したプリコンパイラ展開プログラムを入力するときに指定します)。オプションを省略した場合、COMP-5またはCOMPUTATIONAL-5に展開されたソースとして扱われます(Pro*COBOLのcomp5オプションのデフォルト値はyesです)。 | <b>-C</b>                 |
| 埋込みSQL宣言節内の宣言のみをホスト変数として展開する場合に使用します(Pro*COBOLの場合、declare_sectionオプションにyesを指定して出力したプリコンパイラ展開プログラムを入力するときに指定します)。オプションを省略した場合、埋込みSQL宣言節外の宣言もホスト変数として扱われます(Pro*COBOLのdeclare_sectionオプションのデフォルト値はnoです)。                                                | <b>-D</b>                 |
| オリジナルソースプログラムのコード系がUnicode(UTF-8)の場合に指定します。オプションを省略した場合、シフトJISのソースプログラムとして扱われます。                                                                                                                                                                     | <b>-U</b>                 |

#### 入力ファイル名1

埋込みSQL文の記述されている、オリジナルソースプログラムのファイル名を指定します。

#### 入力ファイル名2

プリコンパイラの出力ファイルである、プリコンパイラ展開プログラムのファイル名を指定します。入力ファイル名2を省略した場合、入力ファイル名1のファイル拡張子を、CBLに置き換えたファイル名が指定されたとして扱われます。

### Oracleデータベース連携時の注意事項

- オリジナルソースプログラムで、ホスト変数のデータ型としてBINARY、COMPまたはCOMPUTATIONALを使用している場合、Pro\*COBOLはcomp5オプションの指定がyesならばCOMP-5に展開して出力します。当コマンドもそれにあわせて展開します。comp5オプションにnoを指定して展開したソースを入力する場合は、**-C**オプションを指定してください。ただし、バイトスワップシステムでは動作しないため、以下の警告メッセージが出力されます。

BINARY、COMPまたはCOMPUTATIONALで展開されたプログラムはバイトスワップのシステムでは動作しません。処理を続けます。

- Pro\*COBOLではdeclare\_section=yesの場合、comp5オプションは埋込みSQL宣言節内のBINARY、COMPまたはCOMPUTATIONALのデータ型に対してのみ有効となります。オリジナルソースプログラムで宣言節外にBINARY、COMPまたはCOMPUTATIONALのデータを定義し、declare\_sectionオプションをyesで展開したソースを入力する場合は、**-D**オプションを指定してください。当コマンドも埋込みSQL宣言節内のみをホスト変数として扱います。

- 以下のメッセージが出力される場合、入力ファイルの誤りまたはオプションの指定誤りが考えられます。

正しい行情報およびファイル名制御情報を出力することができませんでした。入力ファイルまたはオプションを確認してください。

- 当コマンドに入力するファイルは、プリコンパイラに入力するオリジナルソースプログラムと、そのオリジナルソースプログラムを入力としたプリコンパイラ展開プログラムでなければなりません。オリジナルソースプログラムを修正した場合は必ずプリコンパイルしてください。また、プリコンパイラ展開プログラムを修正しないでください。
  - NetCOBOLの予約語、またはPro\*COBOLの予約語およびキーワードが、利用者語として使用されています。予約語およびキーワードに関しては、“COBOL文法書”またはPro\*COBOLのマニュアルを参照してください。
  - Pro\*COBOLのオプションcomp5またはdeclare\_sectionと、当コマンドのオプション-Cまたは-Dの指定の対応がとれていません。Pro\*COBOLのオプションcomp5=noの場合は-Cオプションを、declare\_section=yesの場合は-Dオプションを指定してください。comp5=yesで-Cオプションが指定されている場合またはdeclare\_section=noで-Dオプションが指定されている場合は、それぞれのオプションを解除してください。
  - 上記項目に当てはまらない場合は、“NetCOBOL ソフトウェア説明書”の“留意事項”を参照してください。
- オリジナルソースプログラムにタブ文字を使用しないでください。タブ文字を使用している場合、当ツールがエラーを出力したり、デバッグ使用時に実行時異常を起こしたりする場合があります。この場合、オリジナルソースプログラム中のタブ文字を空白文字に変更し、再度プリコンパイルしてください。

## Symfoware(R)/RDBデータベース連携時の注意事項

Symfoware(R)/RDBデータベースと連携する場合は、行番号制御情報およびファイル名制御情報を埋め込むためのオプションがプリコンパイラ側に用意されていますので、そちらを使用してください。オプションの詳細については、“Symfoware(R) Server RDBユーザーズガイド”を参照してください。

## 動作確認しているプリコンパイラ

動作確認しているプリコンパイラと各プリコンパイラ使用時の留意事項などについては、“NetCOBOL ソフトウェア説明書”を参照してください。

## 付録J 移行上の注意事項

Windows 3.1の既存COBOL85アプリケーションを32ビットWindowsのNetCOBOLに移植する場合、移行のための作業が必要となります。

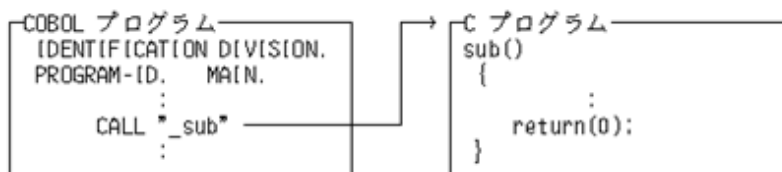
ここでは、このような場合に問題となる、ソースプログラムの互換性およびプロジェクトファイルの互換性について説明します。

### J.1 ソースプログラムの互換性について

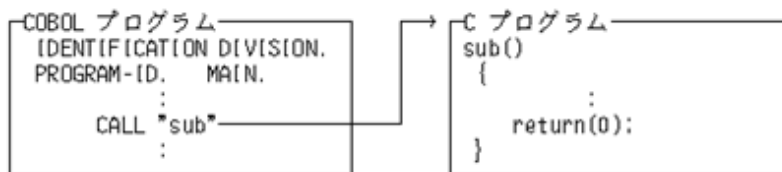
COBOLプログラムからCプログラムを呼ぶ場合、CプログラムからCOBOLプログラムを呼ぶ場合にソース修正が必要となります。COBOLプログラムから呼び出すCプログラムのプログラム名、およびCプログラムから呼ばれるCOBOLプログラムのプログラム名段落から先頭のアンダースコアを削除する必要があります。

COBOLプログラムからCプログラムを呼ぶ場合

Windows 3.1

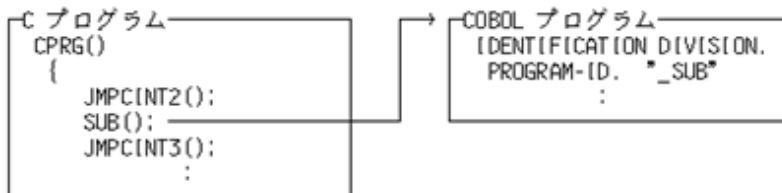


移行先のシステム

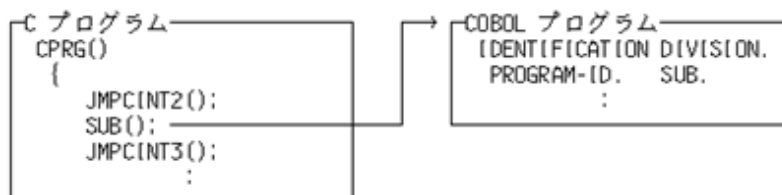


CプログラムからCOBOLプログラムを呼ぶ場合

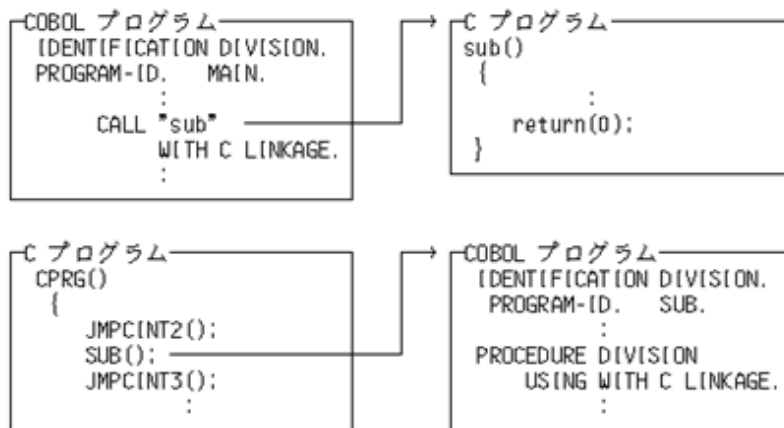
Windows 3.1



移行先のシステム



また、CALL文、ENTRY文またはPROCEDURE DIVISIONに呼出し規約の指定をすることにより、V20L10以降のWindows 3.1版COBOL85と資産を共有することができます。NetCOBOLがサポートしている呼出し規約については、“[10.3 C言語プログラムとの結合](#)”を参照してください。以下の例は、C呼出し規約を指定しています。



## J.2 プロジェクトファイルの互換性について

Windows 3.1版COBOL85で作成したプロジェクトファイルを利用して、COBOL97 V40以降のアプリケーションを構築する場合、以下の作業が必要となります。

### リンクオプションの再設定

Windows 3.1版のリンカと32ビットWindows版のリンカでは、リンクオプションに違いがあるため、COBOLプロジェクトマネージャからリンクオプションを設定し直す必要があります。

COBOL97 V40以降用に設定し直したリンクオプションは、Windows 3.1版のCOBOL85と共通であるリンクオプションファイル(.LNI)に、32ビットWindowsのCOBOL85およびCOBOL97用のセクションを新規に追加する形で格納されます。したがって、リンクオプションの再設定後はWindows 3.1版のCOBOL85と32ビットWindows版のCOBOL85およびCOBOL97とでリンクオプション情報を共用できます。

### モジュール定義ファイルの内容変更

Windows 3.1と32ビットWindowsとではモジュール定義ファイルの内容が異なります。

#### DLLを作成する場合のモジュール定義ファイルの変更

Windows 3.1版のモジュール定義ファイルをもとにして、32ビットWindows版のモジュール定義ファイルを作成する必要があります。32ビットWindowsのモジュール定義文として意味を持たないものはコメントにする(文の1カラム目にセミコロンを記述する)か、削除してください。32ビットWindowsのモジュール定義文については、“[4.6.1 モジュール定義ファイルの内容](#)”を参照してください。ただし、V40以降はPascal(またはSTDCALL)呼出し規約を指定したプログラム以外からDLLを作成する場合、モジュール定義ファイルを使用しなくても作成できます。そのため、資産共用などでモジュール定義ファイルが必要な場合を除き、プロジェクトファイルからモジュール定義ファイルの指定を削除すれば、モジュール定義ファイルは移行する必要はありません。

#### 実行可能ファイルを作成する場合のモジュール定義ファイルの変更

32ビットWindowsでは、実行可能ファイルを作成する場合にモジュール定義ファイルは必要ありません。プロジェクトのビルドまたはリビルドでは、実行可能ファイルに対するモジュール定義ファイルの指定は無視するので、修正の必要はありません。

### 資源が存在するドライブ名/フォルダ名の考慮

プロジェクトファイルに登録してある開発資源が存在するドライブ名などの見え方が、Windows 3.1からアクセスするときと32ビットWindowsからアクセスするときで異なる場合には、これら資源の情報をプロジェクトファイルに登録し直す必要があります。これを回避するためには、Windows 3.1からアクセスする時と32ビットWindowsからアクセスする時で、資源を格納するドライブ名やフォルダの見え方が同じになるようにします。

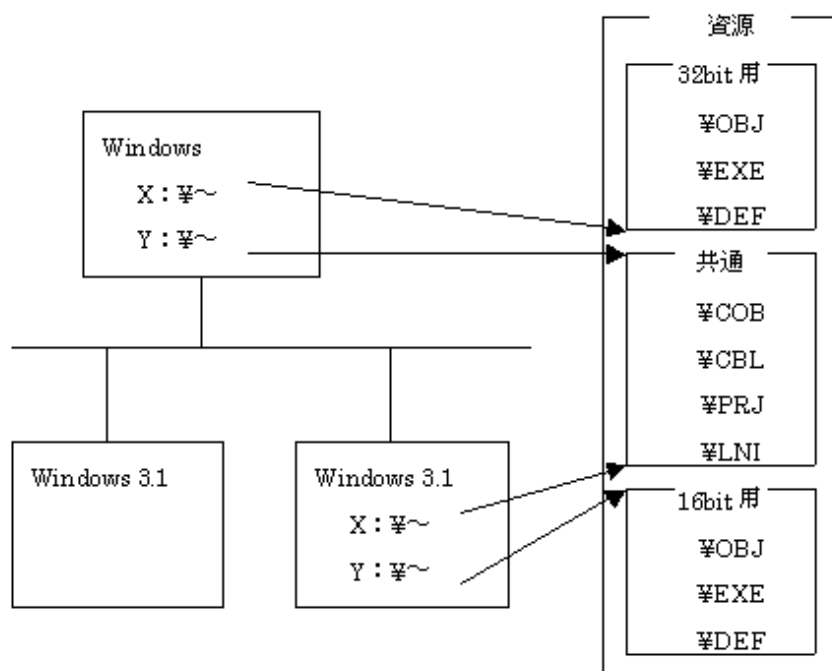


### 例

ソースプログラムなどのシステム間で共有できる資産は共有し、オブジェクトなどのシステム固有資産はシステムごとに管理する場合

- ・システム共通資産(実体は共通)はWindows 3.1からでも32ビットWindowsからでもXドライブの同じフォルダ名で参照できるようにネットワーク接続します。

- システム固有資産(実体は個別)はWindows 3.1からでも32ビットWindowsからでもYドライブの同じフォルダ名で参照できるようにネットワーク接続します。



## 付録K COBOL G移行支援オプション

SX/GシリーズおよびKシリーズのCOBOL G資産を活用するために、NetCOBOLシステムでは、移行支援オプションを用意しています。ただし、これらのオプションは、クラス定義およびメソッド定義には指定できません。

### 登録集の項目名の展開方法の指定

画面帳票定義体またはファイル定義体からCOPY文を使ってレコード定義を取り込むときに展開される項目名を、プログラミング名で展開するか、英数字項目名で展開するか、翻訳オプションPGMNAMEを使って指定します。

以下に、翻訳オプションの形式を示します。

---

PGMNAME( { ALL  
MED  
FFD  
NO } )

---

画面帳票定義体またはファイル定義体の項目名の展開を指定します。

#### ALL

画面帳票定義体、ファイル定義体ともにプログラミング名で展開します。

#### MED

画面帳票定義体だけプログラミング名で展開します。

#### FFD

ファイル定義体だけプログラミング名で展開します。

#### NO

画面帳票定義体、ファイル定義体ともに英数字項目名で展開します。プログラミング名での展開を指定した場合、項目定義にプログラミング名が存在しなければ、英数字項目名で展開します。

### COBOL GからNetCOBOLへの移行支援

NetCOBOLでは、次に示すCOBOL Gの機能を翻訳オプションで指示することにより、使用することができます。

- 名前を省略したクラス“LP”または“SP”のファイル識別名がファイル参照子に指定された場合、印刷ファイルの出力先を印刷装置とします。  
(NetCOBOLでは、ファイル参照子にPRINTERを指定した場合、印刷ファイルの出力先が印刷装置となります。)
- APPLY INDEXED-AREA句が記述できます。  
(NetCOBOLでは、翻訳エラーとなります。)

以下に、翻訳オプションMODEの指定形式を示します。

---

MODE(CBLG)

---

COBOL G からNetCOBOL への移行支援を行うことを指定します。

MODE(CBLG)を指定した場合の扱いを、以下に示します。

- 順ファイルのファイル参照子に、名前を省略したクラス“LP”または“SP”のファイル識別名が記述された場合、ファイル参照子にPRINTERが記述されたものとして扱います。

- APPLY INDEXED-AREA句が記述されている場合、この句を注釈として扱います。(翻訳エラーにはなりません。)

## 付録L XMLの利用

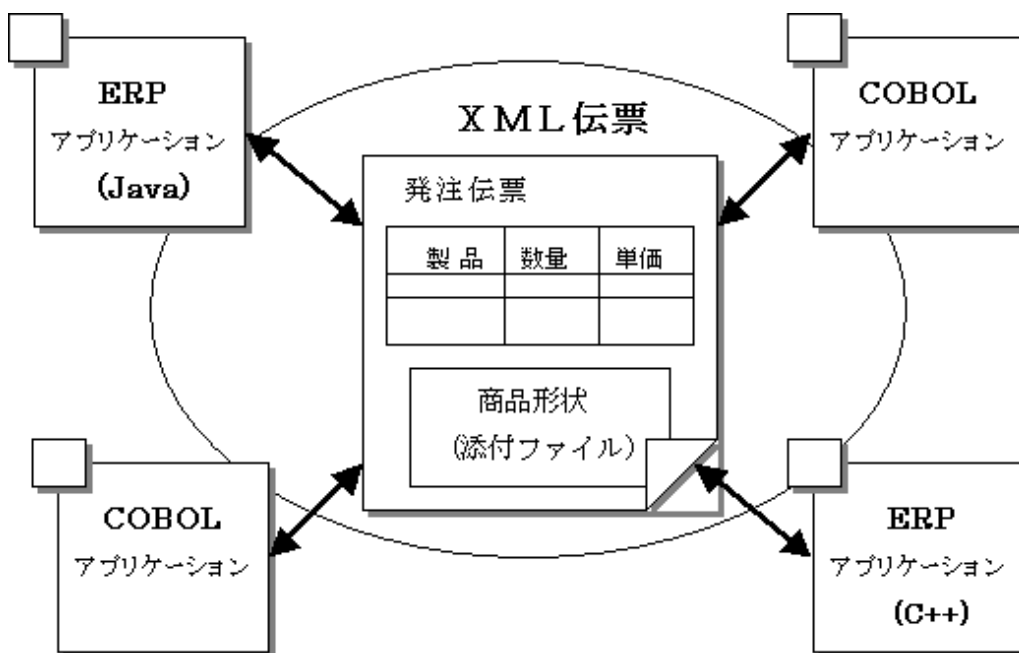
ここでは、XMLの概要とCOBOLアプリケーションからのXMLデータの利用方法および特徴について説明します。

### L.1 XMLの概要

ここでは、XMLの概要について説明します。

#### XMLとは

XMLとは、eXtensible Markup Language(拡張可能なマーク付け言語)の略で、利用者が自由にタグを定義して利用できる言語のことをいいます。もともと、SGML(Standard Generalized Markup Language)をインターネット向けに軽量化したもので、HTML(HyperText Markup Language)の決められたタグしか使えないという問題点を解決する目的でW3C(World Wide Web Consortium)によって規定されました。このため、HTMLのインターネットとの親和性が高いという特徴と、SGMLが持つ文書データを構造化できるという特徴を兼ね備えています。現在では単なるWebページ記述言語にとどまらず、EDI(Electronic Data Interchange)やERP(Enterprise Resource Planning)、EAI(Enterprise Application Integration)またはプロトコル記述などさまざまな方面に適用されつつあります。



#### XML文書の表現

XMLでは文書は要素から成り立っており、各要素タグを使って表現します。たとえば、次のように表現された文書をXML文書と呼びます。

```
<product>
 <name>FMV-DESKPOWER</name>
 <price>200000</price>
 <maker id="Fujitsu"/>
</product>
```

XMLでのタグには、開始タグと終了タグがあり、要素名が同じ開始タグと終了タグで囲まれたものが要素となります。通常、要素を表すには、開始タグに対応した終了タグが必ず必要です。例外として、要素の内容をもたない空要素タグがあります。

- ・ 開始タグ : "<"と">"で囲まれた文字列
- ・ 終了タグ : "</"と">"で囲まれた文字列
- ・ 空要素タグ : "<"と"/>"で囲まれた文字列

開始タグと空要素タグは属性を持つことができます。属性は次の形式をとります。



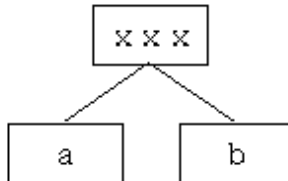
属性名=属性値 または 属性名

以上より、一般にXMLの要素は以下のように表現できます。

```
<要素名 属性名1=属性値 属性名2=属性値>要素の内容</要素名>
```

XMLでは要素の中に要素を記述することで、文書に階層構造を持たせることができます。

```
<xxx>
 <a>AAAA
 BBBB
</xxx>
```



XMLでは文書は必ず正しい階層構造を持たなければなりません。以下のような文書の場合は、XML文書とはみなされません。

```
<xxx>
 <a>AAAA
 BBBB
 CCCC

</xxx>
```

また、XMLでは基本的に大文字と小文字を区別します。したがって、以下のような文書もXML文書とはみなされません。

```
<xxx>
 <a>AAAA
</xxx>
```

これらのXML文書の表現の規則を守っていれば、XML文書としては正しいといえます。しかし、第三者間でXML文書をやりとりする場合に、各人が自由にタグを決めていたのでは、それぞれのXML文書をやりとりしてもタグの意味が分からないので役に立ちません。そこで、これらを回避するためにXML文書で使用するタグ名や出現位置や出現回数、要素内容の構造、属性など(これらを文書型[Document Type]といいます)を定義しておく必要があります。XMLでは、これらの文書型の定義をDTD(Document Type Definition)で行い、DTDに従ったXML文書のことを「検証済みXML文書」と呼んでいます。

ここまでの説明は、XMLに関する概要です。詳細については、W3CのホームページまたはXMLに関する書籍を参照してください。

## XMLの利点

クライアント-サーバ間、サーバ-サーバ間などでデータ交換をする際のデータ記述言語に、XMLを使用することで以下の利点が生じます。

### データの流通性の向上

XMLはタグを自由に定義できるので、内容の意味をタグで表現することができます。また、データの構造も自由に定義できます。したがって、XMLを使用してデータ交換を行う場合には、アプリケーション間でデータインタフェースを合意しやすくなり、データ交換を容易に行うことができます。

### 異種プラットフォーム・言語間の連携が可能

これまでのクライアント-サーバシステムでは、連携相手ごとに独自のフォーマットを使用してデータ交換を行っていました。このデータ交換のフォーマットにXMLを使用することで、プラットフォームおよび言語に依存しないデータ交換のシステムを構築できるようになります。

つまり、クライアントはPC、サーバはメインフレームという組合せや、クライアントアプリケーションはポータブルなJava、サーバアプリケーションにはビジネスロジックの記述に適したCOBOLというような組合せのシステムが容易に構築できます。

## 各種メディアに対応

XMLは、論理的な構造とその意味を定義するXMLデータと、XMLデータの表現形式の定義が独立しています。表現形式指定にはスタイルシートを使用します。このため、データの表現形式を変えるだけならばスタイルシートを更新するだけで済みます。したがって、スタイルシートを切り換えるだけで、同じXMLデータから各種形式の画面や帳票を出力することができます。

## L.2 NetCOBOL XMLオプションの利用

---

ここでは、COBOLアプリケーションからのXMLデータの利用方法および特徴について説明します。

COBOLアプリケーションからXMLデータを利用するための製品として、NetCOBOL XMLオプションを提供します。

### NetCOBOL XMLオプションを使用したCOBOLアプリケーションの利点

#### COBOLスキルの流用

COBOLプログラムでは、XMLのタグ付きデータ(XMLデータ)を意識しないで、集団項目のデータによるデータアクセスを可能にしています。このため、これまでのCOBOLスキルだけでも、XMLデータを処理するプログラムを作成することができます。

#### 既存ビジネスロジックの活用

COBOLプログラムからのXMLデータアクセスは集団項目のデータを利用するため、既存のビジネスロジック資産との親和性は高くなります。このため、DBなどの既存資産を活用して、XMLデータ処理アプリケーションを容易に構築することができます。

### NetCOBOL XMLオプションが提供する機能

#### 電子フォームアプリケーションのフレームワーク対応

Interstage Form CoordinatorまたはSystemwalker Formcoordinatorが提供する電子フォームアプリケーションのフレームワーク(Formcoordinator機能)をCOBOLアプリケーションから利用できます。

Formcoordinator機能を利用することにより、従来の帳票を有効に活用し、画面設計を行うことが可能となります。これにより、帳票入出力を行う業務などを効率良く開発することが可能となります。

従来、Webアプリケーションのフロント部分はJavaで構成されていましたが、本機能を利用することにより、COBOLでWebアプリケーションのフロント部分を作成することが可能となります。

#### MeFt XML入出力機能

MeFt XML入出力機能を利用することにより、FORMで作成した帳票定義体を使って、従来から利用している表示ファイルインターフェースでXMLデータファイルの入出力を行えます。XMLデータの構造は、帳票定義体に格納されますので、XMLを意識したデータ定義やプログラムは不要で、COBOLのREAD文、WRITE文を利用してXMLデータを入出力できます。

#### 他製品・他環境とのXMLデータの連携機能(XML COBOLマッピング機能)

XML COBOLマッピング機能を利用することにより、EAI連携機能などを持つ、他製品・他環境と、XMLデータを利用して連携するCOBOLアプリケーションを作成することが可能となるなど、XMLを基盤としたデータ連携を行う業務アプリケーションを効率良く開発することが可能となります。



#### 注意

NetCOBOL XMLオプションは、NetCOBOLのオプション製品として提供されています。NetCOBOL XMLオプションで提供される機能の詳細については、NetCOBOL XMLオプションに添付されているマニュアルをご参照ください。

## 付録M 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、文字コード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
- 他のシステムで動作していたアプリケーションのWindowsシステムへの移植

このような場合、Windowsともう一方のシステム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。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
- 文字コード変換
- シフトJISへのコード変換時の問題



### 注意

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。

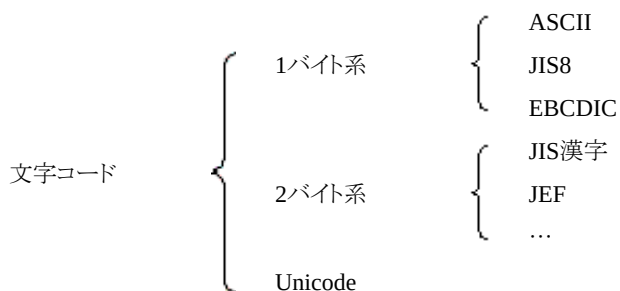
## M.1 文字コードの概説

文字コードにはいくつもの方式があり、かつ、分類の方法にも幾つかの種類があります。そのため、実際には、単純にあるコード系と別のコード系を比較することはできません。しかし、ここでは単純化のために次の分類に分けてコード系を説明します。

- 文字を表現するバイト数の違い
- 文字種の混在方式

### M.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。



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。

- ー バックスラッシュの代わりに“¥”を含む
- ー 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。

- ー JIS漢字に含まれない多くの漢字を含みます。
- ー 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ます。
- ー 83JISで追加された文字も含まれます。

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(次節で説明)が用いられます。

一方、Unicodeは世界中の文字を1つのコード系で網羅することを目指して設計されたので、ここまで説明してきた各コード系とまったく性質が異なります。詳細は、“[第25章 Unicode](#)”を参照してください。

## M.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是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、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。

- ー 文字種の切り換えにシフトコードを使用しません。
- ー 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ちます。
- ー JIS漢字は4つの領域に分散しますが、演算により規則的に対応しますし、文字のコード値の大小関係もJIS漢字と一致します。

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Microsoft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です。
- 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ちます。
- JIS漢字のすべてが1つの領域として含まれます。
- JISカナは1バイトのシフトコードを付けて表します。

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します。
- 各文字のコードはすべてEBCDICおよびJEFに完全に一致します。

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。

各コード系で使用可能な文字の範囲の概要を、図で示します。

| SJIS                                       | EUC(U90)                                   | JEF                                        |
|--------------------------------------------|--------------------------------------------|--------------------------------------------|
| <div>非漢字<br/>JIS 1/2 水準<br/>(約 6000)</div> | <div>非漢字<br/>JIS 1/2 水準<br/>(約 6000)</div> | <div>非漢字<br/>JIS 1/2 水準<br/>(約 6000)</div> |
| <div>利用者定義<br/>(約 2000)</div>              | <div>拡張 文字<br/>(約 6000)</div>              | <div>拡張 文字<br/>(約 5000)</div>              |
|                                            | <div>利用者定義<br/>(約 3000)</div>              | <div>利用者定義<br/>(約 3000)</div>              |
| 計：約8000字                                   | 計：約15000 字                                 | 計：約14000 字                                 |

## M.2 文字コード変換

コード変換と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ための方法です。コード交換の方法は2つあります。

### 変換の前後で文字が一致する変換

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

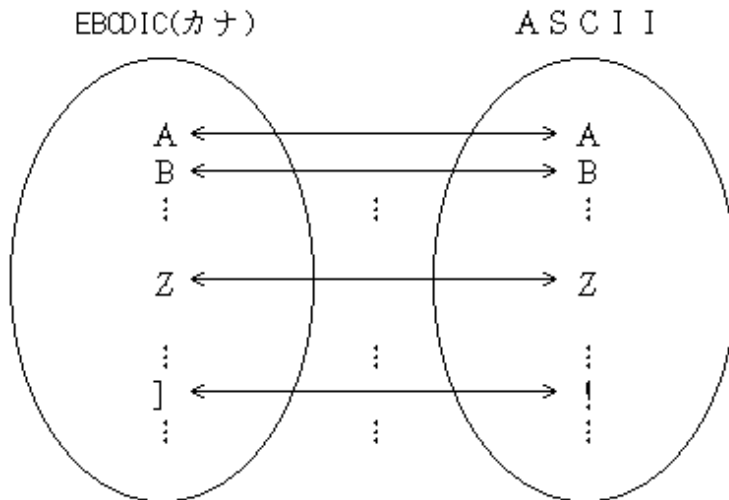
変換先のコード系に含まれる文字の種類が変換元のコード系に含まれる文字の種類より少ない場合、または単純な包含関係が成り立たない場合です。

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、さらに2つの方法があります。

#### 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ませんが、変換の前後で1対1の対応を維持できる場合です。

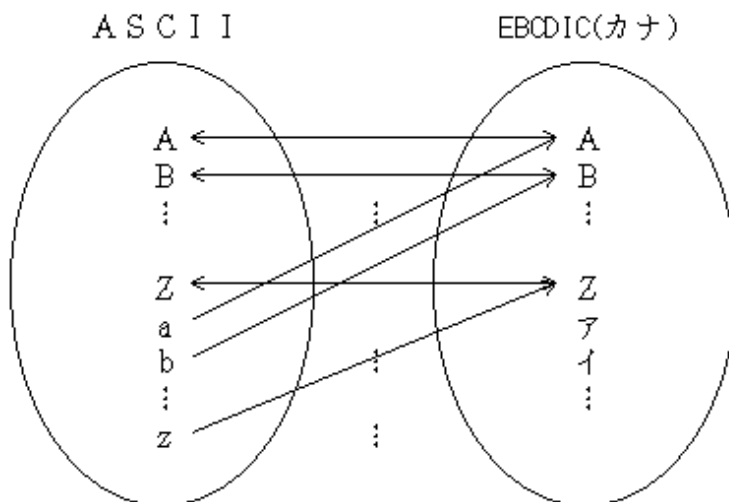
例えばEBCDICからASCIIへの変換をした場合、“!”が“]”に変換されますが、逆方向の変換をすることによって、元に戻すことができます。



#### 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。



## M.3 シフトJISへのコード変換時の問題

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のWindowsシステムへの移植する場合、ソースプログラムやデータファイルの一部をシフトJISにコードに変換する必要があります。

この際、シフトJISと元のソースプログラムやデータファイルで使われていたコード系とシフトJISの非互換から次のような問題が生じる場合があります。

- 翻訳時のエラーの発生
- 実行時の結果誤り、文字化け等

これは主に次のような場合が原因です。

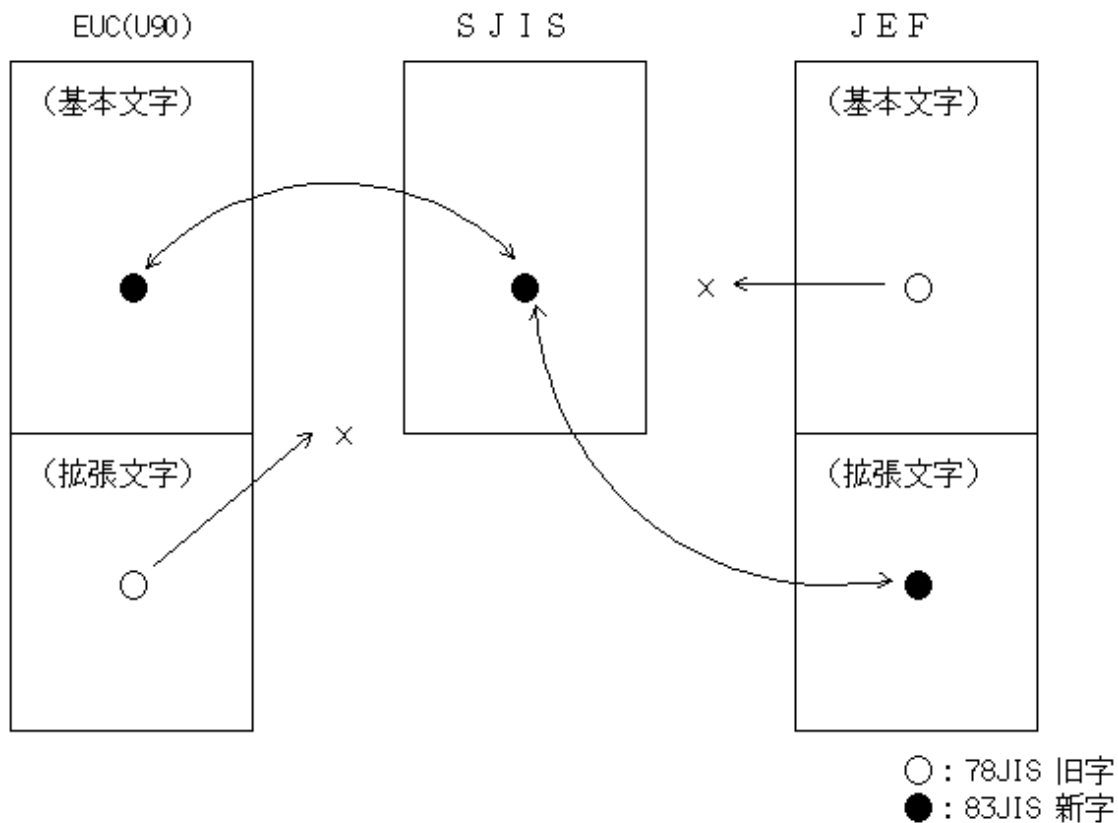
### EBCDICからシフトJISへの変換時の代替文字による変換

以下の一覧に示す文字の変換に関して、網かけされた部分は代替文字による変換が行われます。

| 変換元：E B C D I C |        |        |       | 変換先：S J I S |        |
|-----------------|--------|--------|-------|-------------|--------|
| コード             | ASCII  | 英小文字   | カナ    | コード         | 字形     |
| 0x4F            | “ ! ”  | “   ”  | “   ” | 0x21        | “ ! ”  |
| 0x4A            | “ [ ”  | “ & ”  | “ & ” | 0x5B        | “ [ ”  |
| 0x5A            | “ ] ”  | “ ! ”  | “ ! ” | 0x5D        | “ ] ”  |
| 0x5B            | “ \$ ” | “ \$ ” | “ ¥ ” | 0x23        | “ \$ ” |
| 0x5F            | “ ^ ”  | “ _ ”  | “ _ ” | 0x5E        | “ ^ ”  |
| 0xA1            | “ ~ ”  | “ — ”  | “ — ” | 0x7E        | “ — ”  |
| 0xE0            | “ / ”  | “ \$ ” |       | 0x5C        | “ ¥ ”  |

### 78SJISと83JISで非互換のある文字に対する変換

JEFおよびEUC(U90)は78SJISと83JISで非互換のある文字についてそれぞれの字形毎に別のコードが割り当てられていますが、シフトJISでは1つのコードしか割り当てられていません。このため、一般的にはJEFおよびEUC(U90)に含まれていた78JISの文字の情報は失われてしまいます。



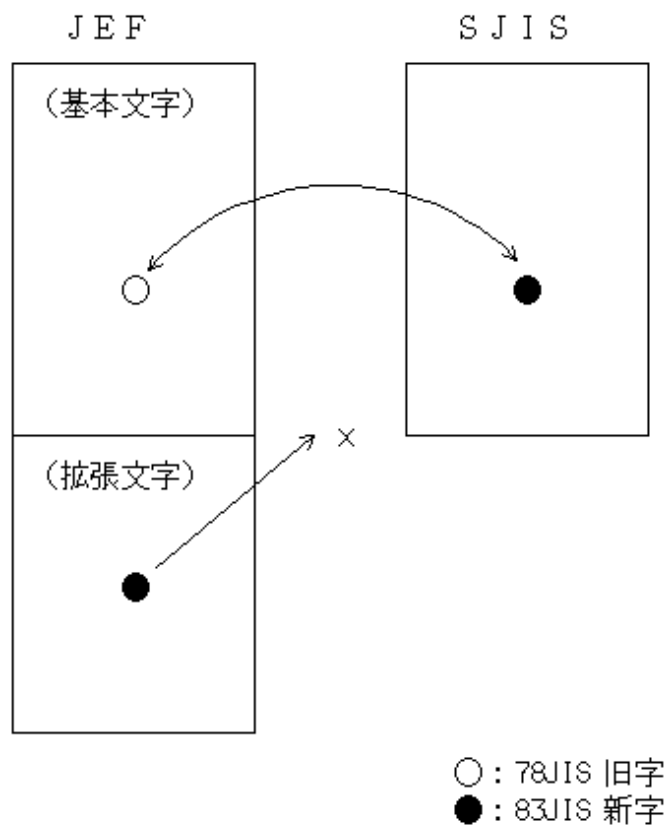
変換前のソースプログラム、データファイル等がEUC(U90)の場合、78JIS の旧字体の文字は拡張文字セットに含まれるものであるため、このような変化が行われても問題となることはあまりありません。

しかし、変換前のソースプログラム、データファイル等がJEFの場合、78JIS の旧字体の文字は基本文字セットに含まれるものであるため、より深刻な影響を被る可能性があります。そのような場合、コード変換の方法を変更し、次のいずれかの変換を行う必要があります。

#### 字形を無視した変換

78JISによる旧字と83JISによる新字の違いを無視して、次のように変換を行います。

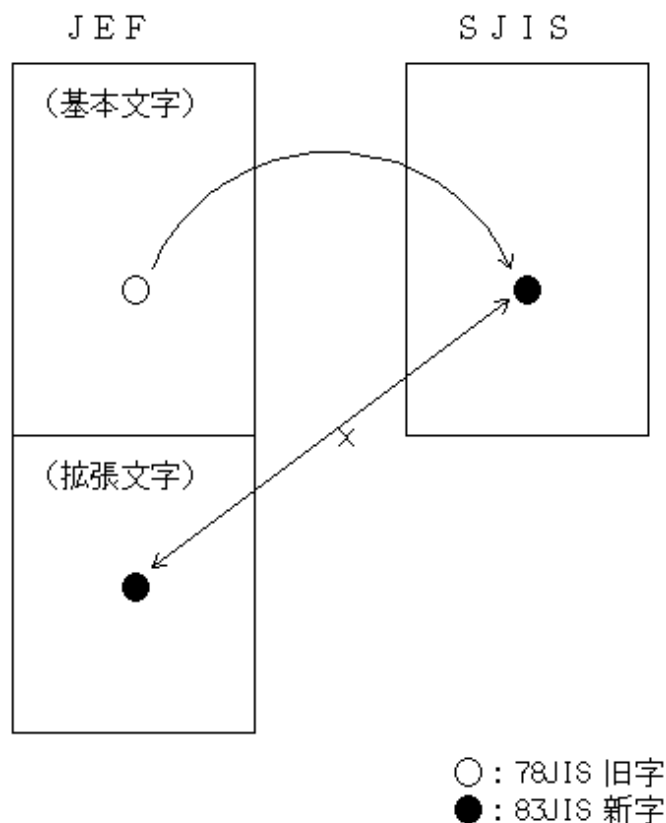




この方法をとる場合、PC上で表示される文字の字形は変換前と異なるものとなってしまいますが、OSIVアプリケーションの分散開発等の場合は、この方法がお勧めです。

#### 縮退を利用した変換

78JISによる旧字と83JISによる新字を縮退による変換を用いて変換します。



この方法をとる場合、78JISによる旧字と83JISによる新字の区別がつかなくなってしまう。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。

## M.4 他システムからの移行上の注意

### M.4.1 日本語空白と英数字空白の文字コード

EBCDICコード系からプログラムを移行してきた場合、空白の比較においてEBCDICコード系と同じ動作を期待できない場合があります。

EBCDICコード系では日本語の空白が英数字の空白2文字分と同じ値を持ちます。このことを前提とした比較は、他のコード系では同じ動作となりません。

空白の文字コードは、コード系によって以下のとおりです。

| コード系       | 英数字の空白 | 日本語の空白   | 主なシステム         |
|------------|--------|----------|----------------|
| EBCDIC/JEF | X"40"  | NX"4040" | OSIV系          |
| EUC        | X"20"  | NX"A1A1" | UNIX系          |
| シフトJIS     | X"20"  | NX"8140" | Windows系       |
| Unicode    | X"20"  | NX"3000" | Windows系、UNIX系 |



例

非互換が発生するプログラムの例

```

WORKING-STORAGE SECTION.
01 GR01.
 02 DATA1 PIC N(1).
01 DATA2 PIC N(1).
 :
MOVE SPACE TO GR01. *> 英数字空白 (X"2020") を転記
MOVE SPACE TO DATA2. *> 日本語空白 (X"8140") を転記
 :
IF DATA1 = DATA2 THEN DISPLAY "EQUAL"
ELSE DISPLAY "NOT EQUAL".

```

上記のプログラムを実行すると、EBCDICコード系の場合は "EQUAL" が表示され、シフトJISコード系および動作モードがUnicodeの場合は "NOT EQUAL" が表示されます。

この場合、プログラムを修正して対応すべきですが、特定の条件下であれば、翻訳オプションNSPCOMP(ASP)を指定して日本語空白の比較方法を変更することによって、プログラムを修正せずに動作可能になります。

以下、注意事項と共に説明します。

## 使用する文や項目に関する条件

翻訳オプションNSPCOMPは、以下の文には作用しません。したがって、プログラム中の以下の文で日本語を扱わないことが条件です。

- INSPECT文
- STRING文
- UNSTRING文
- 索引ファイルのキー操作

2進項目などの非表示項目(USAGE DISPLAYではない項目)が含まれる集団項目に作用しない場合があります。注意してください。

NSPCOMPオプションが作用する比較について、下表にまとめます。

|               |             |             | 作用対象2       |             |             |             |           |           |               |      |           |
|---------------|-------------|-------------|-------------|-------------|-------------|-------------|-----------|-----------|---------------|------|-----------|
|               |             |             | データ項目       |             |             |             |           |           | 定数            |      |           |
|               |             |             | 集団項目        |             |             |             | 基本項目      |           | 表意定数<br>SPACE | 文字定数 | 日本語定<br>数 |
|               |             |             | 日本語項目あり     |             | 日本語項目なし     |             | 英数字項<br>目 | 日本語項<br>目 |               |      |           |
|               |             |             | 非表示項<br>目あり | 非表示項<br>目なし | 非表示項<br>目あり | 非表示項<br>目なし |           |           |               |      |           |
| 作用<br>対象<br>1 | 日本語項<br>目あり | 非表示項<br>目あり | —           | ●           | —           | —           | —         | ●         | —             | —    | ●         |
|               |             | 非表示項<br>目なし | ●           | ●           | ●           | ●           | ●         | ●         | ●             | ●    | ●         |
|               | 日本語項<br>目なし | 非表示項<br>目あり | —           | ●           | —           | —           | —         | ●         | —             | —    | ●         |
|               |             | 非表示項<br>目なし | —           | ●           | —           | —           | —         | ●         | —             | —    | ●         |
|               | 英数字項目       |             |             | ●           | —           | —           | —         | ERR       | —             | —    | ERR       |
|               | 日本語項目       |             | ●           | ●           | ●           | ●           | ERR       | ●         | ●             | ERR  | ●         |
|               | その他(2進項目など) |             |             | ●           | —           | —           | ERR       | ERR       | ERR           | ERR  | ERR       |

●：作用対象1、作用対象2共に、全角空白を半角空白2文字に変換してから比較します。

—：作用しません。

ERR：翻訳エラーになります。

## コード系共通の注意事項

- 日本語に泣き別れが発生するような部分参照を行っている場合は、NSPCOMP(ASP)を指定してもJEFと同じ結果にはなりません。

```
01 G1.
 02 G1-N PIC N(4) VALUE SPACE.
01 G2.
 02 G2-N PIC N(2) VALUE SPACE.
 :
IF G1(2:4) = G2 THEN DISPLAY "EQUAL" *> JEFではEQUAL
 ELSE DISPLAY "NOT EQUAL". *> NSPCOMPを指定してもNOT EQUAL
```

- NSPCOMPは、等価比較だけでなく、大小比較にも作用します。

```
01 G1.
 02 G1-N PIC N(1) VALUE SPACE. *> X"8140"
01 G2.
 02 G2-X PIC X(2) VALUE SPACE. *> X"2020"
 :
IF G1 > G2 THEN DISPLAY "OK?" *> NSPCOMP(NSP)ではTHEN
 ELSE DISPLAY "NG?". *> NSPCOMP(ASP)ではELSE
```

- 集団項目中に2進項目などの非表示項目が含まれていた場合、誤動作する危険があります。

```
01 G1.
 02 G1-B PIC S9(8) BINARY VALUE 33088. *> X"00008140"
 02 G1-X PIC X(2) VALUE SPACE.
01 G2.
 02 G2-B PIC S9(8) BINARY VALUE 8224. *> X"00002020"
 02 G2-N PIC N(1) VALUE SPACE.
 :
IF G1 = G2 THEN DISPLAY "OK?" *> NSPCOMP(ASP)ではTHEN
 ELSE DISPLAY "NG?". *> NSPCOMP(NSP)ではELSE
```

## 動作モードがUnicode固有の注意事項

- Unicodeの場合、日本語字類でX"2020"に該当する文字(短剣符†)が存在します。このため、データ中に短剣符が含まれる場合、誤動作する可能性があります。

```
01 N PIC N(1) VALUE NG"†".
 :
IF N = SPACE THEN DISPLAY "OK?" *> NSPCOMP(ASP)ではTHEN
 ELSE DISPLAY "NG?". *> NSPCOMP(NSP)ではELSE
```

- 動作モードがUnicodeの場合、英数字項目の表現形式はUTF-8となります。全角空白のUTF-16表現とUTF-8表現は異なるため、シフトJISでの動作と異なります。

```
01 G1.
 02 G1-X PIC X(8) VALUE "□□". *> □は全角空白を表す。
01 N PIC N(2) VALUE SPACE.
 :
IF G1 = N THEN DISPLAY "OK?" *> シフトJISではTHEN
 ELSE DISPLAY "NG?". *> UnicodeではELSE
```

## M.4.2 JIS非漢字の負号について

COPY文のDISJOINING/JOINING指定において日本語利用者語の場合に分離符とみなすのは、“JIS非漢字の負号”です。

Unicodeには、見た目では“負号”と識別できるコードが、2つ割り振られています。

|                               | UTF-8表現   | UTF16表現 |
|-------------------------------|-----------|---------|
| <b>MINUS SIGN</b>             | X"E28892" | X"2212" |
| <b>FULLWIDTH HYPHEN-MINUS</b> | X"EFBC8D" | X"FF0D" |

NetCOBOLにおいて、ソースおよび登録集をUTF-8で作成する場合、分離符として上記の「FULLWIDTH HYPHEN-MINUS」を採用しています。このため、日本語利用者語に「MINUS SIGN」を使用していた場合は、意図したとおり動作しません。

ただし、これは翻訳オプションDUPCHARによって変更することができます。

- DUPCHAR(STD) : MINUS SIGN
- DUPCHAR(EXT) : FULLWIDTH HYPHEN-MINUS(省略値)

上記の場合、DUPCHAR(STD)を指定して翻訳してください。

## 付録N セキュリティ

ネットワーク環境では、不正なアクセスによるシステムおよび資源の改ざんや破壊、情報の漏えいなどの危険があります。このため、システムの構築にあたっては、Webサーバのユーザ認証機能と暗号化通信機能を使用し、さらに、アプリケーションでユーザ制限を行うなど、自己防衛手段を講じる必要があります。

### N.1 資源の保護

プログラム、データに関する資源(データベースファイル、入出力ファイル等)およびプログラムの動作に必要な各種の定義・情報ファイルは、OSの機能やプログラムによるアクセス制限を行い、不正なアクセスや改ざんから保護してください。特に重要な資源は、信頼できる安全な業務セグメント(イントラネット環境)内に保持してください。

イントラネット環境の外部に配置するWebサーバ上で、COBOLが提供するWeb連携機能を使用する場合も同様です。データに関する重要な資源はイントラネット環境内に保持してください。

なお、Webサーバ上に配置した、プログラムおよびプログラムの動作に必要な各種の情報ファイルについても、OSの機能によるアクセス制限を行い、不正なアクセスや改ざんから保護してください。

### N.2 アプリケーション作成のための指針

セキュリティを考慮したアプリケーションを作成するための参考にしてください。

#### 事前確認と処理結果の通知

対話・応答を行う処理の場合、重要なデータへのアクセスや処理については、事前の確認および処理結果を通知して、誤った処理を検知できる設計を行ってください。また、ログを記録すると処理の解析に役立ちます。

#### 匿名性

ユーザの実名、実物を識別できるデータについては、特に漏えいの危険性を考慮してください。

#### インタフェースの検査

外部インタフェースについては、バッファオーバーフロー(バッファオーバーラン)やクロスサイトスクリプティングなどを考慮して、セキュリティホールへの作り込みを防止してください。バッファオーバーフローを防止するためには、外部インタフェースの入力データの長さ、型や属性などの検査が有効です。クロスサイトスクリプティングは、動的に生成されたページ中に意図しないタグが含まれないようにする事で防止できます。例えば、出力時にメタキャラクタをエスケープする方法があります。



#### 参考

##### クロスサイトスクリプティング

クロスサイトスクリプティングとは、入力データをプログラムでチェックせず出力データとしてHTMLに埋め込んでいる場合、入力データにJavaScriptなどのスクリプトコードが含まれると、そのHTMLを表示したクライアントでスクリプトが実行されるというものです。悪意のあるスクリプトコードが入力されることにより、Cookieデータの盗聴や改ざんが行われ、Cookieによる認証がパスされたり、セッションの乗っ取りが行われたりする危険があります。また、スクリプトコード以外にもHTMLタグを使って、意図していたものとは異なるHTMLを表示させられる危険もあります。

#### 繰り返し実行

同じ接続先からの一定時間内でのリクエスト数を制限するなどの考慮をしてください。

#### 監査ログの記録

WebサーバやOSの監査ログ機能、およびアプリケーションによるログ出力処理の作成などにより、セキュリティに関するイベントを記録して、セキュリティ侵害が発生した場合の分析や追跡方法を考慮してください。

#### セキュリティのためのルールの制定

セキュリティに関する脆弱な処理がない堅牢なアプリケーションを作成するためには、セキュリティ侵害の脅威から保護すべき重要な資源を特定し、資源のアクセスやインタフェース設計のために特定のルールを制定することが有効です。

## N.3 インターネット接続対象外の通信機能

---

本製品は、通信および通信を行う他製品と連携する機能(表示ファイル、COM連携機能、簡易アプリ間通信機能、Web連携機能)を提供しますが、Web連携機能を除く通信機能は、インターネットへのサービスを提供する用途のためには設計・製造されていません。インターネットに接続しない環境で使用するか、信頼できる安全な業務セグメント(イントラネット環境)内でセキュリティ侵害対策を構築した上で使用してください。

## N.4 Web連携機能のセキュリティ

---

Webサーバの認証機構および暗号化(SSL)通信機能を使用して、不正なアクセスや情報の漏洩、改ざんなどの危険を回避してください。また、Webサーバのアクセスログを採取して、不正なアクセスに対する調査・追跡ができるようにしてください。

詳細は、MeFt/Webを使用する場合は、“MeFt/Web 説明書”を参照してください。Webサブルーチンを使用する場合は、ご使用になるWebサーバのドキュメントを参照してください。

## N.5 リモート開発デバッグ

---

本製品は、ネットワーク上の別のコンピュータで動作するプログラムをデバッグすることができるリモートデバッグを提供しますが、リモートデバッグは、インターネット上で利用するようには設計・製造されていません。インターネットに接続しない環境で使用するか、信頼できる安全な業務セグメント(イントラネット環境)内でセキュリティ侵害対策を構築した上で使用してください。

## N.6 リモート開発機能

---

本製品は、ネットワーク上の別コンピュータでビルドを行うリモート開発機能を提供します。リモート開発機能はインターネット上で利用するように設計・製造されていません。信頼できる安全な業務セグメント(イントラネット環境)内でのみリモート開発機能を使用してください。

## 付録O COBOLプログラムの作成技法

ここでは、効率の良いプログラムを作成するためのテクニックについて説明します。

### O.1 効率のよいプログラム

ここでは、プログラムの実行時間を短縮するための細かい注意点を述べます。

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

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

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

#### O.1.1 一般的な注意

##### 作業場所節の項目

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

##### ループの最適化

- ループの中では、特に、COBOLの文では見かけ上わからない添字計算や、データの属性の変換など、目的コードの生成を極力抑えるような配慮が必要です。
  - ループの中で実行しなくてもいい文はループの外に出してください。
  - ループの中では、データ名による添字付けを避け、指標名による添字付けを用いてください。
  - ループの中で数字転記や数字比較などに用いる項目は、ループの外であらかじめ最適な属性の項目に移してください。

##### 複合条件の判定順序

- ANDだけ、またはORだけで結ばれた複合条件は、括弧がない限り左から右へ順に評価されます。以下の様にと書くと、平均実行時間を短縮することができます。
  - ORで結ばれている場合は、真になりやすい条件を先に書く
  - ANDで結ばれている場合は、偽になりやすい条件を先に書く

#### O.1.2 データ項目の属性の選択

##### 英数字項目と数字項目

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



## USAGE DISPLAYの数字項目(外部10進項目)

- 各文字位置には、文字"0"～"9"(16進表記でX"30"～X"39")が入ります。最後の文字の先頭4ビットは符号を表し、数値が正の場合はX"4"、負の場合はX"5"が入ります。
- 印字表示用として使用してください。演算や比較に使用したときの処理速度は、数字項目の中で最も遅く、使用領域も最も大きくなります。

## USAGE PACKED-DECIMALの数字項目(内部10進項目)

- 4ビットで1桁の数値を表し、最後の4ビットは符号を表します。数値が正の場合はX"C"、負の場合はX"D"、符号なしの場合はX"F"が入ります。
- 演算や比較に使用したときの処理速度は、外部10進項目よりは速く、2進項目よりは遅くなります。

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

- COMP-5の内部表現形式はシステムのエンディアンに従います。BINARYとCOMPは同義で、システムのエンディアンに従わず、常にビッグエンディアンの内部表現形式になります。
- 表示を目的としない演算、添字に適しています。演算や比較は外部10進項目および内部10進項目より速く、リトルエンディアン・システムではBINARYよりCOMP-5が速くなります。

## 数字項目の符号

- 数字項目には、その項目に値を転記する時に絶対値をとる必要がある場合を除き、PICTURE句でSを指定してください。符号をつける場合、SIGN LEADINGやSIGN SEPARATEは指定しないでください。
  - Sの指定がないと、転記する時に絶対値をとるための目的コードが生成されます。
  - SIGN LEADINGやSIGN SEPARATEの指定をすると、符号処理のための余分な命令が生成されます。

## O.1.3 数字転記・数字比較・算術演算

---

### 属性

- 数字転記、数字比較、演算では、作用対象のUSAGE句を統一してください。
- 数字転記、数字比較、加減算では、作用対象の小数部桁数を一致させてください。乗除算では、中間結果の小数部桁数と受取り側項目の小数部桁数を一致させてください。
  - 一致していないと、演算や比較のたびに、一致させるための変換や桁合せのための目的コードが生成されます。
  - 乗算 $C=B \times A$ では $dc=db+da$ を、除算 $C=B/A$ では $dc=db-da$ を満たすようにすると、桁合せは発生しません。(da,db,dcはそれぞれA,B,Cの小数部桁数を表します)
- 算術式では、属性が同じもの同士の演算が多くなるような順に、演算を行ってください。

### 桁数

- 桁数は、必要以上に大きくとらないでください。
  - 一般的に、演算や比較の時間は、桁数が多いほど長くなります。

### べき乗の指数

- べき乗の指数は、整数の定数が最も適しています。次に適しているのは、整数項目です。
  - 整数でない指数が指定されると、COBOLランタイムシステムによる浮動小数点演算となるので、効率は極めて悪くなります。

### ROUNDED指定

- ROUNDED指定の使用は、必要最小限にしてください。
  - ROUNDED指定をすると、演算結果が1桁余分に求められ、正負の判定と四捨五入を行う目的コードが生成されます。

## ON SIZE ERROR指定

- ・ ON SIZE ERROR指定の使用は、必要最小限にしてください。
  - ー ON SIZE ERROR指定すると、桁あふれを判定するために以下のような目的コードが生成されます。

### 演算結果が2進で求まる場合

絶対値をとるなどして受取り側項目に収まる最大値との比較

### 演算結果が内部10進で求まる場合

受取り側項目の文字位置を超える部分とゼロとの比較

## TRUNCオプション

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

## O.1.4 英数字転記・英数字比較

---

### 境界合せ

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

### 項目長

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

### 転記の統合

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

## O.1.5 入出力

---

### SAME RECORD AREA句

- ・ SAME RECORD AREA句は、2つ以上のファイルでレコード領域の内容を共有したい場合や、WRITE文の実行後もレコードを使用する必要がある場合に限って指定してください。
  - ー SAME RECORD AREA句が指定された物理順ファイルのREAD文およびWRITE文は、レコード領域とバッファ領域の間でレコードの転送を行うため、効率が悪くなります。

### ACCEPT文、DISPLAY文

- ・ ACCEPT文(DATE、DAY、TIME指定を除く)およびDISPLAY文は、少量のデータの入出力のみに用いてください。
  - ー これらの文は、READ文およびWRITE文よりも一般的に効率が悪くなります。

## OPEN文、CLOSE文

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

## O.1.6 プログラム間連絡

---

### 副プログラムの分割の基準

- 1つのシステムを多数のプログラムから構成する場合は、必要以上に小さい副プログラムに分割しないことをおすすめします。
  - ー 副プログラムの呼出しから復帰までに、静的構造の場合でも最低数10ステップの機械文が実行されます。したがって、小さい副プログラムでは、このステップ数が相対的に大きな比重を占めることになり、効率を悪化させてしまいます。副プログラムの手続き部が数百行以上あれば、効率は悪化しません。
  - ー 目的プログラムは初期化・終了ルーチンや作業領域などを必ず持っているので、小さい副プログラムに分割すると全体の領域が多く必要になります。

### 動的プログラム構造と動的リンク構造

- 動的プログラム構造(CALL一意名、またはDLOADオプションを指定して翻訳したCALL定数を用いるプログラム構造)は、非常に大きなシステムで、仮想記憶を節約するために副プログラムを削除する場合以外には使用しないでください。動的リンク構造で済むときは、動的リンク構造を使うことをおすすめします。
  - ー 動的プログラム構造では、副プログラムがローディングされた後も、副プログラムが既にローディングされているかどうかを調べるサーチ処理や、プログラム名のチェックが、呼出しのたびに行われます。そのため、オーバーヘッドは、静的プログラム構造より大きくなってしまいます。
  - ー 動的リンク構造では、副プログラムがローディングされた後は、呼出しは直接行われます。そのため、オーバーヘッドは、静的リンク構造の場合よりわずかに多い程度です。

## CANCEL文

- 動的プログラム構造を使う場合、CANCEL文は必要最小限に抑えてください。

### パラメタの個数

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

## O.1.7 デバッグ

---

- CHECK,COUNT,TRACEオプションを使用したデバッグを完了した後は、これらのオプションを取り除いて翻訳してください。
- CHECKオプションの指定によって、実行時間が2倍以上遅くなることがあります。運用時にCHECK(NUMERIC)を有効にする場合は、可能な限り10進項目を2進項目に変更すると、CHECKオプションによる性能劣化を防ぐことができます。

## O.2 数字項目の標準規則

---

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

### O.2.1 10進項目

---

#### 10進項目の入力

- 入力レコード中に10進項目が含まれている場合、誤った内容表現のデータが入らないように注意してください。正しいビットパターンが入っているかどうかを確かめるには、字類条件(IF ... IS NUMERIC)を使います。コンパイラは、READ文の実行時に、10進項目のビットパターンを検査しません。

- ・ 10進項目を含む集団項目へCORRESPONDING指定のない転記を行う場合、誤ったビットパターンが入らないよう注意してください。この場合も、コンパイラは検査を行いません。

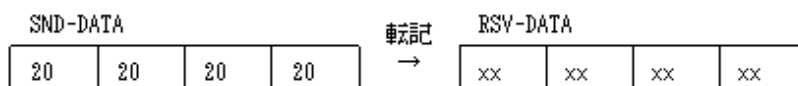
## 10進項目に誤ったビットパターンが入った場合

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

### 誤ったプログラム例

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

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



### 正しいプログラム例

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

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

## O.2.2 2進項目

### 2進項目の値の範囲

2進項目は、一般に、PICTURE句で示された値の範囲より大きい絶対値をもつ値を含むことができます。NOTRUNC指定の場合、2進項目への転記の際にPICTURE句に合わせた切り捨てが行われなかった結果、正の値が負の値になることもあります。  
[参照]“[A.2.59 TRUNC \(桁落とし処理の可否\)](#)”

### 2進項目のけた数の扱い

2進項目は、PICTURE句より大きい値を含むことができますが、これをDISPLAY文で参照した時は、PICTURE句で示された桁数だけ表示されます。  
ON SIZE ERROR指定、またはNOT ON SIZE ERROR指定が記述された演算文では、PICTURE句の指定を超えた値を格納しようとしているかどうかの判定が行われます。

一般に、コンパイラは、2進項目の値はPICTURE句で示された範囲内にあることを前提としてコンパイルを行うため、PICTURE句より大きい値を持つ2進項目を演算などに使用すると、場合によっては異常終了を起こすことがあります。

## O.2.3 浮動小数点項目

---

### 固定小数点への変換

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

## O.2.4 数字項目の注意事項

---

### 乗除算の混合時の小数部桁数

次のプログラムの[1]と[2]を比較します。

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

[1]の答はZ=333.00、[2]の答はZ=333.33となります。

この違いは、除算を行うタイミングによって発生します。

どちらも、除算の結果はXとZの小数部桁数の大きい方、すなわち小数第2位まで求められます。[1]では、X/Yが3.33と求められ、これが100倍されます。[2]ではX\*100の中間結果である1000がYで割られ、333.33が求まります。

よって精度のよい結果を求めるには、[2]のように、乗算を先に除算を後に行ってください。

### 絶対値がとられる転記

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

## O.3 注意事項

---

### 外部ブール項目のビットパターン

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

### レコード領域の参照

- ・ OPEN文実行前、またはCLOSE文実行後のファイルのレコード領域を参照してはいけません。

# 索引

|                                   |                                  |             |
|-----------------------------------|----------------------------------|-------------|
| [ ]                               | /DEBUGTYPE.....                  | 622,659     |
| -SRTnnnn.TMP.....                 | /STACK.....                      | 154         |
|                                   | @AllFileExclusive.....           | 95          |
| [数字]                              | @CBR_ATTACH_TOOL.....            | 96,966      |
| 10進項目.....                        | @CBR_CBRFILE.....                | 98          |
| 1行の形式.....                        | @CBR_CBRINFO.....                | 98          |
| 1つの行を参照する.....                    | @CBR_CIINF.....                  | 98,719      |
| 1つのソースファイルを翻訳する.....              | @CBR_CI_SRVINF.....              | 726         |
| 1つの目的プログラムをリンクする.....             | @CBR_ClassInfFile.....           | 567         |
| 2進項目.....                         | @CBR_COMPOSER_CONSOLE.....       | 98,431      |
| 2進項目の扱い.....                      | @CBR_COMPOSER_MESS.....          | 99          |
| 88コンソーシアム形式.....                  | @CBR_COMPOSER_SYERR.....         | 99,431      |
|                                   | @CBR_COMPOSER_SYSOUT.....        | 99,431      |
| [記号]                              | @CBR_CONSOLE.....                | 100         |
| #.....                            | @CBR_CSV_OVERFLOW_MESSAGE.....   | 100         |
| #DEC88TOFJ.....                   | @CBR_CSV_TYPE.....               | 100         |
| #DECFJTO88.....                   | @CBR_DocumentName_xxxx.....      | 105,308     |
| #FILE情報.....                      | @CBR_ENTRYFILE.....              | 105         |
| #LINE情報.....                      | @CBR_EXFH_API.....               | 106         |
| \$.....                           | @CBR_EXFH_LOAD.....              | 106         |
| *COB-BINDTABLE.....               | @CBR_FILE_BOM_READ.....          | 106         |
| *COB-BINDTABLEクラスのオブジェクトメソッド..... | @CBR_FILE_LFS_ACCESS.....        | 107         |
| *COB-BINDTABLEクラスのファクトリメソッド.....  | @CBR_FILE_SEQUENTIAL_ACCESS..... | 107         |
| *COM.....                         | @CBR_FILE_USE_MESSAGE.....       | 108         |
| *COM-ARRAYの型指定子.....              | @CBR_FUNCTION_NATIONAL.....      | 108         |
| *COM-ARRAYクラス.....                | @CBR_InstanceBlock.....          | 567         |
| *COM-EXCEPTIONクラス.....            | @CBR_JOBDATE.....                | 108         |
| *COMアーライバインドクラス.....              | @CBR_JUSTINTIME_DEBUG.....       | 109         |
| *COMクラス.....                      | @CBR_MEMORY_CHECK.....           | 113         |
| *COMレイトバインドクラス.....               | @CBR_MESSAGE.....                | 110         |
| *OLE.....                         | @CBR_MESS_LEVEL_CONSOLE.....     | 111         |
| *OLE-ARRAY.....                   | @CBR_MESS_LEVEL_EVENTLOG.....    | 112         |
| *OLE-EXCEPTION.....               | @CBR_OverlayPrintOffset.....     | 113         |
| ->.....                           | @CBR_OverlayPrintSPEC.....       | 115         |
| -A.....                           | @CBR_PrinterANK_Size.....        | 115         |
| -Dc.....                          | @CBR_PrintFontTable.....         | 116         |
| -dd.....                          | @CBR_PrintInfoFile.....          | 116         |
| -Dk.....                          | @CBR_PrintTextPosition.....      | 117         |
| -do.....                          | @CBR_PSFILE_ACM.....             | 716         |
| -dp.....                          | @CBR_PSFILE_DSP.....             | 368         |
| -Dr.....                          | @CBR_PSFILE_PRT.....             | 350         |
| -dr.....                          | @CBR_PSFILE_xxx.....             | 117         |
| -ds.....                          | @CBR_SCREEN_POSITION.....        | 118         |
| -Dt.....                          | @CBR_SCR_KEYDEFFILE.....         | 118         |
| -f.....                           | @CBR_SSIN_FILE.....              | 875         |
| -I.....                           | @CBR_SYMFOWARE_THREAD.....       | 875         |
| -i.....                           | @CBR_SYSERR_EXTEND.....          | 119         |
| -M.....                           | @CBR_TextAlign.....              | 119         |
| -m.....                           | @CBR_THREAD_MODE.....            | 875         |
| -O.....                           | @CBR_THREAD_TIMEOUT.....         | 875         |
| -P.....                           | @CBR_TRACE_FILE.....             | 119,607,608 |
| -R.....                           | @CBR_TRAILING_BLANK_RECORD.....  | 120         |
| -WC.....                          | @CBR_SSIN_FILE.....              | 428         |
| .ENTRY.....                       | @CnslBufLine.....                | 120         |
| .METHOD.....                      | @CnslFont.....                   | 120         |
| /DEBUG.....                       |                                  | 622,659     |

|                         |         |
|-------------------------|---------|
| @CnslWinSize.....       | 121     |
| @DefaultFCB_Name.....   | 121,300 |
| @ExitSessionMSG.....    | 121     |
| @GOPT.....              | 91,122  |
| @IconDLL.....           | 122     |
| @IconName.....          | 123     |
| @MessOutFile.....       | 112,123 |
| @MGPRM.....             | 91,123  |
| @NoMessage.....         | 124     |
| @ODBC_Inf.....          | 124,781 |
| @OPTIONS.....           | 14,1010 |
| @PrinterFontName.....   | 124,296 |
| @PRN_FormName_xxx.....  | 125,304 |
| @ScrnFont.....          | 126     |
| @ScrnSize.....          | 126,369 |
| @ShowIcon.....          | 126     |
| @SQL_CONCURRENCY.....   | 784     |
| @SVD_COMPATIBILITY..... | 661,700 |
| @SVD_ENV.....           | 660     |
| @SVD_PERFORM_CHECK..... | 660     |
| @WinCloseMsg.....       | 126,369 |
| _FINALIZEメソッド.....      | 485,507 |
| _GET_プロパティ名.....        | 563     |
| _SET_プロパティ名.....        | 563     |
| _stdcall.....           | 400     |
| _YEN.....               | 1018    |

#### [A]

|                            |             |
|----------------------------|-------------|
| A.....                     | 23          |
| A3.....                    | 304         |
| A4.....                    | 304         |
| A5.....                    | 304         |
| AADアプリケーション.....           | 976,983     |
| ABNORMAL END.....          | 613         |
| ACCEPT文.....               | 417,434,437 |
| ACCEPT文のデータの入力先.....       | 1038        |
| ACCEPT文の動作.....            | 1028        |
| ACCEPT文のファイル入力拡張機能.....    | 427         |
| ACCESS MODE句.....          | 209,214     |
| ACM.....                   | 117,1048    |
| ACM制御.....                 | 1048        |
| ACM通信機能.....               | 716         |
| ADDRESS.....               | 579         |
| ADDR関数.....                | 453,454     |
| AFTER指定.....               | 323         |
| AIM DBMSとの連絡領域.....        | 1023        |
| AIMLIB.....                | 1012        |
| AIM展開レコード域.....            | 1023        |
| ALL.....                   | 1015,1034   |
| ALPHABET句.....             | 1040        |
| ALPHAL.....                | 1013        |
| ALTERNATE RECORD KEY句..... | 215         |
| ANK空白.....                 | 1029        |
| ANK文字サイズ.....              | 115         |
| ANSI COBOL規格.....          | 1023        |
| ANY LENGTH句.....           | 592         |
| APL.....                   | 117         |

|                      |                             |
|----------------------|-----------------------------|
| APOST.....           | 1032                        |
| ARGUMENT-NUMBER..... | 434,435                     |
| ARGUMENT-VALUE.....  | 434,435                     |
| ASCII.....           | 1040                        |
| ASCOMP5.....         | 1013                        |
| ASP.....             | 1029                        |
| ASSIGN句.....         | 223,224,225,325,326,327,328 |
| AT END指定.....        | 221                         |
| AUTO.....            | 727                         |
| A領域.....             | 12                          |

#### [B]

|                                     |             |
|-------------------------------------|-------------|
| B4.....                             | 304         |
| B5.....                             | 304         |
| BASED ON句.....                      | 454         |
| BEFORE指定.....                       | 323         |
| BINARY.....                         | 1014,1044   |
| BIND.....                           | 307         |
| BOM.....                            | 887         |
| BOTH.....                           | 659         |
| BOTTOM.....                         | 119,312     |
| BOUND.....                          | 1015        |
| BSAM.....                           | 127,230     |
| BSORT_TMPDIR.....                   | 238,443,446 |
| Btrieveファイル.....                    | 127,280,899 |
| BTRV.....                           | 127,281     |
| BUFFERLEN.....                      | 738         |
| BY CONTENT.....                     | 395         |
| BY REFERENCE.....                   | 395         |
| BY REFERENCE指定とBY CONTENT指定の違い..... | 396,397     |
| BYTE.....                           | 1014        |
| BY VALUE.....                       | 396         |
| B領域.....                            | 12          |

#### [C]

|                      |                     |
|----------------------|---------------------|
| c.....               | 149                 |
| C.....               | 303                 |
| C++.....             | 459                 |
| CALLBACK.....        | 400                 |
| CALL文.....           | 387,395,734         |
| CALL文の自動生成.....      | 552                 |
| CANCEL PROGRAM.....  | 613                 |
| CBI.....             | 156                 |
| CBL.....             | 13,17,156           |
| CCVS.....            | 1028                |
| CFURCOV.....         | 288                 |
| CFURCOVS.....        | 288                 |
| CHARACTER TYPE句..... | 295,322             |
| CHECK.....           | 601,1015            |
| CHECK機能.....         | 48,149,595,877,1015 |
| CHECK機能の使用例.....     | 601                 |
| CHECK機能の使い方.....     | 597                 |
| CH情報.....            | 300                 |
| CIM.....             | 286                 |
| CLASS.....           | 562                 |
| CLASS-ID.....        | 483                 |
| CLOSEONLY.....       | 727                 |
| CLOSE文.....          | 366,715             |

|                                  |           |                                 |                         |
|----------------------------------|-----------|---------------------------------|-------------------------|
| CLSID.....                       | 974       | COBOLランタイムシステムによる同期制御.....      | 832                     |
| COB.....                         | 13,17,156 | COB_OPTIONS.....                | 6                       |
| COB_AIMLIB.....                  | 6         | COB_REPIN.....                  | 6                       |
| COBCI_CLOSE.....                 | 736       | COB_REPORT_EVENT.....           | 1072                    |
| COBCI_OPEN.....                  | 735       | COBSVINI.INI.....               | 726                     |
| COBCI_READ.....                  | 737       | COB_UNLOCK_DATA.....            | 880                     |
| COBCISRV.LOG.....                | 728       | COB_UNLOCK_OBJECT.....          | 881                     |
| COBCIU32.EXE.....                | 729       | COB_GETRUNIT.....               | 866                     |
| COBCI_WRITE.....                 | 739       | COB_SETRUNIT.....               | 867                     |
| COB_COBCOPY.....                 | 6         | COB_登録集名.....                   | 5                       |
| COB_FREE_MEMORY.....             | 1081      | coclass.....                    | 975                     |
| COB_GET_PROCESSID.....           | 1074      | COFF.....                       | 659                     |
| COB_GET_THREADID.....            | 1074      | COM.....                        | 883,901                 |
| COB_LIBSUFFIX.....               | 6         | commit.....                     | 154                     |
| COB_LOCK_DATA.....               | 879       | COMMIT文.....                    | 746                     |
| COB_LOCK_OBJECT.....             | 881       | COMMON.....                     | 392                     |
| COBOL.....                       | 13        | COMn.....                       | 130                     |
| COBOL32コマンド.....                 | 44        | COMPANY表.....                   | 751                     |
| COBOL32コマンドの復帰コード.....           | 46        | COM機能の例外処理.....                 | 916                     |
| COBOL85.CBR.....                 | 86        | COM機能を利用したアプリケーション.....         | 901                     |
| COBOL.CBI.....                   | 18        | COMクライアントからのトランザクション制御.....     | 938                     |
| COBOL G移行支援オプション.....            | 1098      | COMクライアント機能.....                | 905                     |
| COBOLアプリケーション起動中のWindows終了.....  | 121       | COMクライアントとCOMサーバを通じたデバッグ方法..... | 967                     |
| COBOLおよびVisual C++でのクラスの対応.....  | 569       | COMクライアントの作成.....               | 905                     |
| COBOLが提供するサブルーチン.....            | 1070      | COMクライアントのデバッグ方法.....           | 965                     |
| COBOLコンソールウィンドウ.....             | 897       | COMサーバからのトランザクション制御.....        | 935                     |
| COBOLコンパイラ.....                  | 2         | COMサーバ機能.....                   | 921                     |
| COBOL実行単位ハンドル取得サブルーチン.....       | 866       | COMサーバ作成機能の使い方.....             | 969                     |
| COBOL実行単位ハンドル設定サブルーチン.....       | 867       | COMサーバ情報ファイル.....               | 156                     |
| COBOL診断機能.....                   | 3         | COMサーバとは.....                   | 921                     |
| COBOLソースファイルの登録.....             | 167       | COMサーバのインタフェース.....             | 975                     |
| COBOLソースファイル名表示.....             | 551       | COMサーバの削除.....                  | 924                     |
| COBOLソースプログラムの作成.....            | 12        | COMサーバの作成.....                  | 922                     |
| COBOL対話型デバッグ.....                | 2         | COMサーバの実装形態.....                | 975                     |
| COBOLの機能.....                    | 1         | COMサーバのデバッグ方法.....              | 966                     |
| COBOLの言語間の環境.....                | 379       | COMサーバの登録.....                  | 924                     |
| COBOLのコンソールウィンドウ.....            | 419       | COMサーバ名の設定.....                 | 967                     |
| COBOLの主プログラム.....                | 379       | COMとDCOM.....                   | 975                     |
| COBOLのデータ項目とCのデータ型との対応例.....     | 403       | COMのマルチスレッド環境.....              | 883                     |
| COBOLファイル.....                   | 237       | COM連携機能.....                    | 883                     |
| COBOLファイルアクセスルーチン.....           | 292       | COM連携で扱うデータ型の対応.....            | 953                     |
| COBOLファイルのサイズを拡張する指定.....        | 107       | COB_ALLOC_MEMORY.....           | 1081                    |
| COBOLファイルユーティリティ.....            | 3,237     | CONF.....                       | 1016,1044               |
| COBOLファイルユーティリティの起動.....         | 238       | CONNECT文.....                   | 746,747                 |
| COBOLファイルユーティリティの機能.....         | 239       | CONSOLE.....                    | 421                     |
| COBOLファイルユーティリティの終了.....         | 239       | CONST型指定子.....                  | 401                     |
| COBOLファイルユーティリティの使い方.....        | 238       | CONTROL RECORD句.....            | 341                     |
| COBOLプログラムからCOBOLプログラムを呼び出す..... | 377,386   | COPY.....                       | 1017                    |
| COBOLプログラムからCプログラムを呼び出す.....     | 378,395   | COPY修飾値.....                    | 22                      |
| COBOLプログラムから呼び出すサブルーチン.....      | 1070      | COPY文.....                      | 40,1024                 |
| COBOLプログラムの構成.....               | 745       | CORBAアプリケーション.....              | 976                     |
| COBOLプログラムの作成技法.....             | 1116      | COUNT.....                      | 611,1017                |
| COBOLプログラムの実行.....               | 82        | COUNT機能.....                    | 48,133,595,610,877,1017 |
| COBOLプログラムを呼び出すCプログラムの結合.....    | 407       | COUNT機能の使い方.....                | 610                     |
| COBOL目的プログラム.....                | 61        | COUNT情報.....                    | 611                     |
| COBOL呼出し規約.....                  | 376,377   | COUNT情報の出力形式.....               | 611                     |
| COBOLランタイムシステム.....              | 2         | COUNT情報の出力ファイル.....             | 133                     |



|                               |             |
|-------------------------------|-------------|
| cpi.....                      | 311         |
| CRDB_XLIB_OVR.....            | 284         |
| CREATE.....                   | 1018        |
| CREATEメソッド.....               | 485         |
| CRT STATUS句.....              | 372         |
| CSI.....                      | 156         |
| CSV形式データ操作時のメッセージ抑止.....      | 100         |
| CSV形式データとは.....               | 447         |
| CSV形式データの作成.....              | 448         |
| CSV形式データの操作.....              | 447         |
| CSV形式データの分解.....              | 449         |
| CSV形式のバリエーション.....            | 451         |
| CTL.....                      | 331         |
| CURRENCY.....                 | 1018        |
| CURRENT-DATE関数を利用した西暦の取得..... | 1056        |
| CURSOR句.....                  | 372         |
| C言語プログラムとの結合.....             | 394         |
| CプログラムからCOBOLプログラムを呼び出す.....  | 378,379,400 |
| Cプログラムを呼び出すCOBOLプログラムの結合..... | 410         |
| C呼出し規約.....                   | 376,378     |
| C呼出し規約で結合する.....              | 407,410     |

## [D]

|                          |             |
|--------------------------|-------------|
| D.....                   | 23          |
| DATE.....                | 429         |
| DAY.....                 | 429         |
| DAY-OF-WEEK.....         | 429         |
| DCOM.....                | 883,902     |
| DCOMとMTS .....           | 928         |
| Declare句.....            | 147         |
| DEFAULT.CBI.....         | 18,42       |
| DEFAULT指定.....           | 784         |
| DELETE文.....             | 746,756     |
| DESTINATION-1句.....      | 715         |
| DISCONNECT文.....         | 747,748     |
| DISK.....                | 200,225     |
| DISPLAY文.....            | 417,434,437 |
| DISPLAY文のデータの出力先.....    | 1039        |
| DISPLAY文のファイル出力拡張機能..... | 426         |
| DLL.....                 | 56,58,156   |
| DLLエントリオブジェクト.....       | 90          |
| DLLを作成する.....            | 61          |
| DLOAD.....               | 1018        |
| DOC-INFO.....            | 105,308     |
| DocumentName.....        | 312         |
| DSP.....                 | 117         |
| DTD.....                 | 1101        |
| DUMPBINコマンド.....         | 1090        |
| DUMPBINコマンドのオプション.....   | 1090        |
| DUPCHAR.....             | 1019        |
| DUPLICATES.....          | 216         |
| DYNAMIC.....             | 209,214     |

## [E]

|                  |      |
|------------------|------|
| EBCDIC指定.....    | 1040 |
| EDIT-COLOR.....  | 310  |
| EDIT-CURSOR..... | 366  |
| EDIT-MODE.....   | 310  |

|                             |          |
|-----------------------------|----------|
| EDIT-OPTION.....            | 310      |
| EDSAC.....                  | 456      |
| EMPLOYEE.....               | 579      |
| END-EXEC.....               | 746      |
| EndMode.....                | 727      |
| END OF INITIAL PROGRAM..... | 613      |
| END OF RUN UNIT.....        | 613      |
| ENTRY文.....                 | 141,387  |
| ENVIRONMENT-NAME.....       | 437      |
| ENVIRONMENT-VALUE.....      | 437      |
| ENVコマンド.....                | 678      |
| EQUALS.....                 | 443,1019 |
| ERROR.....                  | 728      |
| EVALUATE文.....              | 221      |
| EXCEPTION文.....             | 222      |
| EXCLUSIVE.....              | 226      |
| EXE.....                    | 56,156   |
| EXEC SQL.....               | 746      |
| EXECUTE IMMEDIATE文.....     | 758      |
| EXIT PROGRAM文.....          | 387,401  |
| EXP.....                    | 56,156   |
| EXTERNAL句.....              | 389      |

## [F]

|                                                     |                |
|-----------------------------------------------------|----------------|
| F3BICIMP.LIB.....                                   | 71             |
| f3bifrm.dll.....                                    | 290            |
| f3bifutc.dll.....                                   | 290            |
| f3bifutc.h.....                                     | 289            |
| F3BIFUTC.LIB.....                                   | 289            |
| f3bifuty.dll.....                                   | 290            |
| far pascal.....                                     | 400            |
| FCB.....                                            | 300,303,329    |
| FCBxxxx.....                                        | 132            |
| FCB制御文.....                                         | 132,333        |
| FCB制御文の形式.....                                      | 300            |
| FCB名.....                                           | 303            |
| FCBを使うプログラム.....                                    | 333            |
| FCOM.....                                           | 1023           |
| FETCH PRIOR文によるデータの取得.....                          | 772            |
| FETCH文.....                                         | 746            |
| FFD.....                                            | 17,156,666,704 |
| FFD_SUFFIX.....                                     | 6,20           |
| FILEEXT.....                                        | 20,1019        |
| FILELIB.....                                        | 6,1020         |
| FILE STATUS句.....                                   | 221,1046       |
| FIX.....                                            | 1038           |
| FJBASEクラス.....                                      | 484            |
| FLAG.....                                           | 1020           |
| FLAGSW.....                                         | 1021,1044      |
| FORM.....                                           | 7,362          |
| FORM/MeFt.....                                      | 897            |
| FORMAT-ID.....                                      | 303            |
| FORMAT句付き印刷ファイル.....                                | 293,295,897    |
| FORMAT句なし印刷ファイル.....                                | 293,897        |
| FORMAT句なし印刷ファイル(行単位のデータを印刷する).....                  | 294            |
| FORMAT句なし印刷ファイル(フォームオーバーレイおよびFCB<br>を使用して印刷する)..... | 295            |

|                      |                     |
|----------------------|---------------------|
| FORMEXT.....         | 20,342,349,367,1022 |
| FORMLIB.....         | 6,342,349,367,1022  |
| FORMオーバーレイオプション..... | 7,299               |
| FORM情報.....          | 300                 |
| FORTRAN.....         | 456                 |
| FOR句による処理行数制御.....   | 770                 |
| FOVL.....            | 302                 |
| FOVLDIR.....         | 132,315,332         |
| FOVL-n.....          | 309                 |
| FOVLNAME.....        | 133,316             |
| FOVLTYPE.....        | 132,315,333         |
| FREE.....            | 1038                |
| FROM指定.....          | 417                 |

## [G]

|                   |      |
|-------------------|------|
| GDI.....          | 333  |
| GEN.....          | 1023 |
| GETCLASSメソッド..... | 485  |
| GLOBAL句.....      | 393  |
| GSS.....          | 1022 |
| GSW.....          | 1021 |

## [H]

|                |     |
|----------------|-----|
| HOPPER.....    | 304 |
| hostsファイル..... | 718 |

## [I]

|                                                          |         |
|----------------------------------------------------------|---------|
| ICON.....                                                | 727     |
| ICONF.....                                               | 1015    |
| IconMenu.....                                            | 727     |
| IDL-COBOL変換機能.....                                       | 1006    |
| IF文.....                                                 | 221     |
| INDEXED.....                                             | 214     |
| INHERITS句.....                                           | 483     |
| INITIAL.....                                             | 393     |
| INITVALUE.....                                           | 1023    |
| INITメソッド.....                                            | 485,507 |
| INSDBINFコマンド.....                                        | 1091    |
| INSDBINFコマンドのオプション.....                                  | 1093    |
| INSERT文.....                                             | 746,756 |
| InstanceBlockセクション.....                                  | 568     |
| Interstage Business Application Server.....              | 431     |
| Interstage Business Application Serverの汎用ログを使うプログラム..... | 431     |
| Interstageなどのサーバ環境で動作するプログラムに対するデバッグ起動方法.....            | 663,691 |
| INVALID KEY指定.....                                       | 221     |
| INVOKE文.....                                             | 476,477 |
| INVOKE文のテンプレート展開.....                                    | 526,552 |
| is_a関係.....                                              | 465     |
| I制御レコード.....                                             | 302     |
| I制御レコードによる文書名.....                                       | 105     |

## [J]

|                    |      |
|--------------------|------|
| JIS1.....          | 1040 |
| JIS2.....          | 1040 |
| JIS7単位ローマ字コード..... | 1040 |
| JIS8単位コード.....     | 1040 |

|               |         |
|---------------|---------|
| JIS非漢字.....   | 1029    |
| JMPBGWDH..... | 1070    |
| JMPBWINS..... | 1071    |
| JMPCINT2..... | 379,817 |
| JMPCINT3..... | 817     |
| JMPCINT4..... | 817     |
| JMPCINTB..... | 147     |
| JMPCINTC..... | 146     |

## [K]

|           |             |
|-----------|-------------|
| KOL5..... | 132,315,333 |
|-----------|-------------|

## [L]

|                      |             |
|----------------------|-------------|
| L.....               | 303,307     |
| LAND.....            | 300         |
| LANGLVL.....         | 1023,1044   |
| [LDINF]セクション.....    | 728         |
| LDNAME-REC.....      | 738         |
| LENG関数.....          | 453,454     |
| LIB.....             | 56,156,1024 |
| LIBC.LIB.....        | 71          |
| LIBEXT.....          | 1024        |
| LINECOUNT.....       | 1025        |
| LINE SEQUENTIAL..... | 203         |
| LINESIZE.....        | 1025        |
| LINKAGE.....         | 1015        |
| LINKコマンド.....        | 71          |
| LINKコマンドの形式.....     | 73          |
| LIST.....            | 1026        |
| ListWORKS.....       | 315,350     |
| LNI.....             | 156         |
| LOCK MODE句.....      | 226         |
| LOG.....             | 667,706     |
| Log.....             | 727         |
| LogFileName.....     | 728         |
| LogType.....         | 728         |
| long int型.....       | 399,400,402 |
| LP.....              | 303         |
| LPI.....             | 311         |
| LPI情報.....           | 300         |
| LPT1.....            | 1062        |
| LPT2.....            | 1062        |
| LPT3.....            | 1062        |
| LPTn.....            | 130         |
| LST.....             | 18          |
| LTR.....             | 304         |
| LZ.....              | 303         |

## [M]

|                             |              |
|-----------------------------|--------------|
| MAIN.....                   | 37,1026,1044 |
| MAK.....                    | 156          |
| MAKE.....                   | 156          |
| MAKEファイル.....               | 156,192      |
| MAKEファイルとプロジェクトファイルの比較..... | 193          |
| MAKEファイルの用途.....            | 192          |
| MANAGER.....                | 579          |
| MANUAL.....                 | 727          |
| MAP.....                    | 26,1027      |

|                     |                    |
|---------------------|--------------------|
| MeFt.....           | 7,309,349,367,1048 |
| MEFTNET.....        | 350,368            |
| MeFt/NET-SV.....    | 350,368            |
| MeFt/NET連携機能.....   | 350,368            |
| MeFt/Web.....       | 7,350,368          |
| MEMBER.....         | 578                |
| MERGE文.....         | 439,445            |
| MESSAGE.....        | 21,45,1027         |
| MLBOFF.....         | 1014               |
| MLBON.....          | 1014               |
| MODE.....           | 1028,1098          |
| MODE-1.....         | 295                |
| MODE-2.....         | 295                |
| MODE-3.....         | 295                |
| MSGLEN.....         | 738                |
| MSGTYPE.....        | 738                |
| MTA.....            | 883                |
| MTS.....            | 901                |
| MTS環境への登録方法.....    | 929                |
| MTSの概要.....         | 929                |
| MTSのトランザクション管理..... | 935                |
| MTSのトランザクション処理..... | 942                |

#### [N]

|                         |              |
|-------------------------|--------------|
| N.....                  | 307,309      |
| NAME.....               | 1028         |
| NATIONAL関数の変換モード.....   | 108          |
| NCW.....                | 1028         |
| NetCOBOLのアイコン表示の抑止..... | 126          |
| NetCOBOLの概要.....        | 1            |
| NetCOBOLの機能.....        | 1            |
| NEWメソッド.....            | 485,586      |
| noc.....                | 149          |
| nocb.....               | 149          |
| nocl.....               | 149          |
| nocn.....               | 149          |
| nocp.....               | 149          |
| nor.....                | 148          |
| NSP.....                | 1029         |
| NSPCOMP.....            | 685,1029     |
| NUMBER.....             | 597,686,1030 |
| NUMERIC.....            | 1015         |
| NUMERICタイプ.....         | 280          |

#### [O]

|                                 |             |
|---------------------------------|-------------|
| OBJ.....                        | 18,56,156   |
| OBJECT.....                     | 1031        |
| OBS.....                        | 1016        |
| OCCURS DEPENDING ON句の目的語検査..... | 602         |
| ODBC.....                       | 898         |
| ODBC環境のセットアップ.....              | 790         |
| ODBC経由によるアクセス.....              | 744         |
| ODBC情報設定ツール.....                | 786         |
| ODBC情報設定ツールの使い方.....            | 786         |
| ODBC情報ファイル.....                 | 124,781,786 |
| ODBC情報ファイルの作成.....              | 782         |
| ODBCで扱うデータとの対応.....             | 798         |

|                           |                 |
|---------------------------|-----------------|
| ODBCドライバ使用時の注意事項.....     | 801             |
| ODBCドライバの導入.....          | 790             |
| OFFSET.....               | 307             |
| OPEN-DATA-FILE.....       | 582             |
| OPEN文.....                | 366,715         |
| OPTIMIZE.....             | 1031            |
| OPTIONAL.....             | 202,205,212,219 |
| ORDERS表.....              | 750             |
| ORGANIZATION句.....        | 200,203,208,214 |
| OSIV系形式の実行時パラメタ.....      | 123             |
| OSIV系形式の実行時パラメタの指定方法..... | 91              |
| OSIV系システム.....            | 1063            |
| OSIV系システム固有機能.....        | 1063            |
| OSIV系システムとの機能比較.....      | 1063            |
| OSIV系システムと本システムの機能比較..... | 1063            |
| OSIV系プログラム.....           | 1063            |
| OVD.....                  | 133             |
| OVD_SUFFIX.....           | 133,333         |
| OVD_SUFFIX.....           | 316             |
| OverlayPrintOffset.....   | 313             |
| OverlayPrintSPEC.....     | 312             |
| OVERRIDE句.....            | 486             |

#### [P]

|                          |                             |
|--------------------------|-----------------------------|
| P.....                   | 23,303,304                  |
| P1.....                  | 304,305                     |
| P2.....                  | 304,305                     |
| PAGE.....                | 323                         |
| PASCAL.....              | 400                         |
| PATH.....                | 238,290,342,349,350,367,368 |
| pdbファイル.....             | 58                          |
| PERFORM文の最適化.....        | 1051                        |
| PGMNAME.....             | 1098                        |
| PMD.....                 | 17,156,666,704              |
| POINTER.....             | 454                         |
| PORT.....                | 300                         |
| POSIT.....               | 305                         |
| PowerCOBOL.....          | 7                           |
| PowerFORM.....           | 7                           |
| PowerSORT.....           | 443,446                     |
| PowerSORTが使用するメモリ容量..... | 149,1036                    |
| PRINT.....               | 21,1032                     |
| PRINTER.....             | 116,325                     |
| PRINTER-n.....           | 328                         |
| PRINTING MODE句.....      | 295                         |
| PRIORITY.....            | 740                         |
| PRJ.....                 | 156                         |
| PRM.....                 | 1015                        |
| ProgID.....              | 974                         |
| PROGRAM-STATUS.....      | 391,399,402,740,1070        |
| PROPERTY句.....           | 504                         |
| PROTOTYPEメソッド.....       | 500                         |
| PRT.....                 | 117                         |
| PRT-AREA.....            | 307                         |
| PRT-FORM.....            | 303                         |
| PRTNAME.....             | 130,324                     |
| PRTOUT.....              | 315                         |

PXD.....17,156,666,704  
PZ.....303

## [Q]

QUOTE.....1032

## [R]

R.....23,302  
r.....148,607  
RAISE文の動作.....556  
RAISING指定のEXIT文の動作.....556  
RANDOM.....209,214  
RCI.....156  
RCS.....1032  
RDM.....127,284  
RDMファイル.....127,283  
RDWR.....729  
RECEIVETYPE-REC.....738  
RECORD KEY句.....215  
RECORD句.....200  
REGISTER-INDICATORメソッド.....588  
REGISTER-LARGE-DATA-INDICATORメソッド.....591  
REGISTER-LARGE-DATAメソッド.....590  
REGISTER-VARYING-INDICATORメソッド.....589  
REGISTER-VARYINGメソッド.....588  
REGISTERメソッド.....587  
RELATIVE.....208  
RELATIVE KEY句.....210  
REP.....18,156,1033  
REPIN.....1033  
reserve.....154  
RETRIEVE.....582  
RETURNING指定.....391,397,401,479  
ROLLBACK文.....746  
RPW.....1021  
RSV.....308,309,1034

## [S]

S.....23,304  
s.....149  
SAI.....18,1035  
SAVE.....582  
SCS.....1035  
SDS.....1036  
SD機能.....453  
SD機能の種類.....453  
SELECTED FUNCTION句.....366  
SELECT句.....199  
SELECT句およびASSIGN句の記述例.....200  
SELECT文.....746,754  
SENDTYPE-REC.....740  
SEPARATE指定.....282  
SEQUENTIAL.....200,209,214  
SERVERHD-REC.....736  
SERVERNAME-REC.....736  
servicesファイル.....718  
SET CONNECTION文.....746,748  
SETコマンドで設定.....86

SHREXT.....1036  
SIA.....1021  
SIDE.....305  
SIZE.....125,304  
SIZE情報.....300  
Smalltalk.....459  
SMD.....17,156,666,704  
SMED\_SUFFIX.....6,20  
smsize.....149  
SMSIZE.....1036  
SORT-STATUS.....442,445  
SORT文.....439,442  
SORT文での同一キーデータの処理方法.....1019  
SOURCE.....1037  
SQL.....744  
SQLCODE.....800  
SQLERRD.....769  
SQLGRP.....1037  
SQLMSG.....800  
SQLODBCS.EXE.....786  
SQLSTATE.....800  
SQL終了子.....746  
SQL先頭子.....746  
SQLのホスト変数定義の拡張.....1037  
SRF.....1038  
[SRVINP]セクション.....726  
SSIN.....134,425,1038  
SSOUT.....134,425,1039  
STA.....883  
StartMode.....727  
STATUS-DETAIL.....736  
STATUS-ERROR.....736  
STATUS-REC.....736  
STD.....1028,1029  
STD1.....1040  
STDCALL呼出し規約.....376,378,414  
STDCALL呼出し規約で結合する.....408,412  
STDH.....1021  
STDI.....1021  
STDM.....1021  
STOCK表.....750  
STOP RUN文.....387  
STREAM.....315  
STREAMENV.....315  
STRING文.....448  
SVD.....18,659  
SWITCH-0.....149  
SWITCH-7.....149  
SWITCH-8.....149  
SYMBOLIC DESTINATION句.....117  
SymfoWARE7000.....1048  
SymfoWARE7000 for Windows NT.....713  
Symfoware連携でマルチスレッド動作可能にする.....875  
SYS.....1029  
SYSCOUNT.....133,611  
SYSDEF.....727  
SYSERR.....422

|                    |     |
|--------------------|-----|
| SYSERR出力情報の拡張..... | 119 |
| SYSINのアクセス名.....   | 134 |
| SysMaxMsg.....     | 726 |
| SysMaxWait.....    | 726 |
| SYSOUTのアクセス名.....  | 134 |
| S制御レコード.....       | 308 |

## [T]

|                  |                     |
|------------------|---------------------|
| TAB.....         | 1040                |
| TAB文字.....       | 13                  |
| TCP/IPプロトコル..... | 718                 |
| TEST.....        | 622,659,1040        |
| TextAlign.....   | 312                 |
| THREAD.....      | 1041                |
| TIME.....        | 429                 |
| TOP.....         | 119,312             |
| TRACE.....       | 607,728,1041        |
| TRACE機能.....     | 50,148,595,876,1041 |
| TRACE機能の使い方..... | 606                 |
| TRC.....         | 608                 |
| TRO.....         | 608                 |
| TRUNC.....       | 1042                |
| TYPE1.....       | 117                 |
| TYPE2.....       | 117                 |
| TYPE-G.....      | 116                 |
| TYPE-M.....      | 116                 |
| TYPE-PC.....     | 116                 |

## [U]

|                              |         |
|------------------------------|---------|
| Unicode.....                 | 685,886 |
| Unicodeアプリケーション.....         | 890     |
| Unicodeのぎょう順ファイル.....        | 207     |
| Unicodeの行順ファイルの識別コード.....    | 106     |
| UNSTRING文.....               | 449     |
| UNUSED-REC.....              | 736     |
| UPDATE文.....                 | 746,756 |
| UPON指定.....                  | 417     |
| USAGE IS DISPLAY.....        | 293     |
| USAGE OBJECT REFERENCE句..... | 476     |
| USE AFTER ERROR.....         | 222     |
| USING指定.....                 | 478     |
| USING指定の記述の違い.....           | 396     |
| UWA.....                     | 1023    |

## [V]

|                                |      |
|--------------------------------|------|
| V111.....                      | 1034 |
| V112.....                      | 1034 |
| V122.....                      | 1035 |
| V125.....                      | 1035 |
| V30.....                       | 1035 |
| V40.....                       | 1035 |
| V61.....                       | 1035 |
| V70.....                       | 1035 |
| V81.....                       | 1035 |
| V90.....                       | 1035 |
| VAR.....                       | 1038 |
| Visual Basic.....              | 414  |
| Visual C++で定義されているクラスを調べる..... | 573  |

|                           |      |
|---------------------------|------|
| Visual C++連携の概要.....      | 569  |
| Visual C++連携のプログラム手順..... | 572  |
| Visual C++連携の方法.....      | 568  |
| void型.....                | 400  |
| VSR2.....                 | 1035 |
| VSR3.....                 | 1035 |
| VT型.....                  | 954  |

## [W]

|                                 |      |
|---------------------------------|------|
| WAITTIME.....                   | 738  |
| Web連携.....                      | 810  |
| WIDTH.....                      | 307  |
| WINAPI.....                     | 400  |
| WINCOBウィンドウ.....                | 34   |
| WINCOBウィンドウの起動.....             | 34   |
| WINCOBコマンド.....                 | 3    |
| WINDOW.....                     | 727  |
| Windowsの終了時の動作.....             | 727  |
| WindowsソケットAPI.....             | 718  |
| Windowsのコード系.....               | 1030 |
| WINEXECウィンドウ.....               | 141  |
| WINEXECウィンドウの起動.....            | 141  |
| WINEXECコマンド.....                | 3    |
| WINLINKウィンドウ.....               | 64   |
| WINLINKウィンドウの起動.....            | 64   |
| WINLINKコマンド.....                | 3    |
| WINLINK[ライブラリモジュール結合]ウィンドウ..... | 68   |
| WINLINK[リンク]ウィンドウ.....          | 64   |
| WITH LOCK.....                  | 226  |
| WITH NO LIMIT.....              | 455  |
| WORD.....                       | 1014 |

## [X]

|               |                     |
|---------------|---------------------|
| XFDLIB.....   | 51,1020             |
| XLデータパイプ..... | 285                 |
| XMDLIB.....   | 52,341,348,365,1022 |
| XML.....      | 1100                |
| XREF.....     | 22,1043             |

## [Z]

|          |      |
|----------|------|
| ZWB..... | 1043 |
|----------|------|

## [あ]

|                                                   |             |
|---------------------------------------------------|-------------|
| アイコンリソースのDLL名.....                                | 122         |
| アイコンリソースの識別名.....                                 | 123         |
| 相手論理宛先.....                                       | 717         |
| アクセス形態.....                                       | 198,209,214 |
| アクセス種別.....                                       | 127         |
| 宛先.....                                           | 117         |
| アテンション種別.....                                     | 365         |
| アテンション情報の値.....                                   | 365         |
| アプリケーションエラー.....                                  | 635         |
| アプリケーションエラーやUレベルの実行時メッセージが発生するアプリケーションに対する起動..... | 622         |
| アプリケーションプログラムの実行.....                             | 173         |
| アプリケーションページ.....                                  | 663,702     |
| アポストロフィ.....                                      | 1032        |
| 誤り処理手続き.....                                      | 222         |

|                                 |                   |                                       |               |
|---------------------------------|-------------------|---------------------------------------|---------------|
| 暗黙スレッド.....                     | 683               | 英数字転記・英数字比較.....                      | 1118          |
| 暗黙スレッドの切り替え.....                | 683               | 英数字の文字の大小順序.....                      | 1040          |
| アンロード.....                      | 239               | 永続オブジェクト.....                         | 577           |
| アーバインド.....                     | 906               | エクスポートファイル.....                       | 56,156        |
| アーバインドのメリットとデメリット.....          | 908               | エディタ連携機能.....                         | 551           |
| 異常終了時にデバッグまたは診断機能を使って調査を行う..... | 109               | エラー検出時の処理実行回数.....                    | 149           |
| 依存リポジトリ検索.....                  | 537               | エラーコード.....                           | 882           |
| 依存リポジトリファイル.....                | 535               | エラージャンプ.....                          | 42            |
| 依存リポジトリファイルの登録手順.....           | 536               | エラーログ.....                            | 728           |
| 一次入口点.....                      | 387               | エントリ情報.....                           | 85,135,562    |
| 一連番号領域.....                     | 12                | エントリ情報の記述形式.....                      | 562           |
| 一般的な注意.....                     | 1116              | エントリ情報のセクション名.....                    | 87            |
| 移動.....                         | 239               | エントリ情報の設定.....                        | 146           |
| イベントログ.....                     | 811               | エントリ情報ファイル.....                       | 105,136       |
| イベントログ出力サブルーチン.....             | 1072              | 応答しなくなったアプリケーションに対する起動.....           | 623           |
| イベントログの指定可能項目.....              | 811               | 応答なし.....                             | 640           |
| イベントログを使うプログラム.....             | 432               | 同じ親クラスを持つクラスを同じファイルに保存する.....         | 580           |
| インクルードファイル.....                 | 185               | オブジェクト.....                           | 459           |
| 印刷.....                         | 239               | オブジェクトインスタンス.....                     | 822,834       |
| 印刷機能.....                       | 885               | オブジェクトインスタンスの獲得方法.....                | 567           |
| 印刷形式.....                       | 303               | オブジェクトインスタンスの格納数.....                 | 568           |
| 印刷原点位置.....                     | 307               | オブジェクトインスタンスの寿命.....                  | 480           |
| 印刷情報ファイル.....                   | 116,311           | オブジェクトインスタンスの操作.....                  | 475           |
| 印刷処理.....                       | 293               | オブジェクトコンテキストオブジェクト.....               | 936           |
| 印刷装置の指定方法.....                  | 324               | オブジェクト削除インタフェースプログラム.....             | 574           |
| 印刷ファイル.....                     | 197,895           | オブジェクト参照.....                         | 460           |
| 印刷ファイルで使用するフォント.....            | 124               | オブジェクト参照項目.....                       | 476,489       |
| 印刷ファイルの定義.....                  | 322               | オブジェクト指向.....                         | 458           |
| 印刷不可能な領域.....                   | 319               | オブジェクト指向と従来機能の組合せ.....                | 1088          |
| 印刷方法の種類.....                    | 293               | オブジェクト指向に対応したデバッグ機能.....              | 683           |
| 印刷方法の特徴・利点・用途.....              | 293               | オブジェクト指向のメリット.....                    | 467           |
| 印刷面.....                        | 305               | オブジェクト指向の歴史的背景.....                   | 456           |
| 印刷面位置付け.....                    | 305               | オブジェクト指向プログラミング.....                  | 456           |
| 印字禁止域.....                      | 307               | オブジェクト指向プログラミング機能.....                | 468,555,884   |
| 印字文字.....                       | 295               | オブジェクト指向プログラミング機能を使用したデータベースアクセス..... | 776           |
| 印字文字の大きさ.....                   | 295               | オブジェクト指向プログラミングで使用する資源.....           | 518           |
| 印字文字の大きさ/形態と文字間隔の関係.....        | 299               | オブジェクト指向プログラミングで使用するファイル.....         | 519           |
| 印字文字の間隔.....                    | 299               | オブジェクト指向プログラミングの開発.....               | 518           |
| 印字文字の形態.....                    | 298               | オブジェクト指向プログラム開発の手順.....               | 9             |
| 印字文字の書体.....                    | 297               | オブジェクト指向プログラムの開発環境.....               | 9             |
| 印字文字のスタイル.....                  | 298               | オブジェクト指定子.....                        | 503,920       |
| 印字文字の配置座標.....                  | 311               | オブジェクト生成インタフェースプログラム.....             | 574           |
| 印字文字の方向.....                    | 298               | オブジェクト定義.....                         | 471           |
| 印字モード名.....                     | 295               | オブジェクトの永続化.....                       | 577           |
| インスタンスハンドル獲得サブルーチン.....         | 1071              | オブジェクトの作成・参照.....                     | 460           |
| インタフェースの表示.....                 | 524,550           | オブジェクトの寿命.....                        | 480           |
| インタフェースプログラムの仕組み.....           | 570               | オブジェクトの保存/復元.....                     | 581           |
| インポートライブラリ.....                 | 56,58,156,170,538 | オブジェクトファイル.....                       | 18,56,156,169 |
| インポートライブラリの作成.....              | 66                | オブジェクトファイルのフォルダ.....                  | 49            |
| ウィンドウ情報ファイル.....                | 367               | オブジェクトファイル名.....                      | 73            |
| ウィンドウ情報ファイルの作成.....             | 367               | オブジェクトプロパティの入れ子.....                  | 510           |
| ウィンドウ情報ファイルの設定情報.....           | 367               | オブジェクトプロパティのテンプレート展開.....             | 526,552       |
| ウィンドウハンドル獲得サブルーチン.....          | 1070              | オブジェクトメソッド.....                       | 473,573,582   |
| 埋込みSQL文.....                    | 746               | オブジェクトロックサブルーチン.....                  | 880           |
| 埋込みSQL文のキーワード一覧.....            | 790               | オプション一覧.....                          | 46            |
| 上書き保存.....                      | 172               | オプション情報リスト.....                       | 21,1027       |
| 英小文字の扱い.....                    | 1013              |                                       |               |

|                                     |               |
|-------------------------------------|---------------|
| オプション設定ウィンドウ.....                   | 522           |
| オプションファイル.....                      | 18,42,52,1010 |
| オプションファイルの指定.....                   | 52            |
| 親クラス.....                           | 462,483       |
| 親クラスのオブジェクトデータも含めて1つのファイルに保存する..... | 579           |
| オーバフローしない.....                      | 151           |
| オーバフローする.....                       | 153           |

## [か]

|                                   |                         |
|-----------------------------------|-------------------------|
| 改行文字.....                         | 12                      |
| 開始プログラム名.....                     | 664,702                 |
| 会社表.....                          | 751                     |
| 開発環境.....                         | 4                       |
| 開発手順.....                         | 519                     |
| 開発パス.....                         | 553                     |
| 外部10進項目のデータ形式変換.....              | 281                     |
| 外部スイッチの値.....                     | 149                     |
| 外部属性.....                         | 389                     |
| 外部データ.....                        | 389,826,830             |
| 外部データ使用時の注意事項.....                | 390                     |
| 外部ファイル.....                       | 389,826,830             |
| 外部ファイル使用時の注意事項.....               | 390                     |
| 外部ファイルハンドラ.....                   | 286                     |
| 外部ファイルハンドラで結合するファイルシステムのDLL名..... | 106                     |
| 外部ファイルハンドラで結合するファイルシステムの入り口名..... | 106                     |
| 外部プログラム呼出しのパラメタの検査.....           | 605                     |
| 外部名を指定するための定数.....                | 1062                    |
| 概要.....                           | 813                     |
| 各種定義体ファイルの登録.....                 | 169                     |
| 各種翻訳リストの出力.....                   | 1032                    |
| 拡張.....                           | 202,205,211,218,229,239 |
| 仮想メモリ不足.....                      | 154                     |
| 型情報.....                          | 975                     |
| 型ライブラリ.....                       | 975                     |
| カプセル化.....                        | 458                     |
| 可変形式.....                         | 13,1038                 |
| 可変長レコード形式.....                    | 198,200                 |
| 画面項目.....                         | 373                     |
| 画面項目に指定するCOBOLの句.....             | 373                     |
| 画面帳票定義体.....                      | 13,896                  |
| 画面帳票定義体の検索順序.....                 | 20                      |
| 画面帳票定義体ファイル.....                  | 17,156,169              |
| 画面帳票定義体ファイルの拡張子.....              | 1022                    |
| 画面帳票定義体ファイルのフォルダ.....             | 52,1022                 |
| 画面定義体.....                        | 362                     |
| 画面定義体に設定する情報.....                 | 363                     |
| 画面定義体の作成.....                     | 362                     |
| 画面入出力状態の設定値.....                  | 372                     |
| 画面による入出力操作の抑止方法.....              | 808                     |
| 画面を使った入出力.....                    | 359                     |
| 画面を使った入出力の種類.....                 | 359                     |
| カレントフォルダ.....                     | 1012                    |
| 簡易アプリ間通信.....                     | 3                       |
| 簡易アプリ間通信機能.....                   | 712,717,884             |
| 簡易アプリ間通信機能の使い方.....               | 716                     |
| 簡易アプリ間通信サーバウィンドウ.....             | 720                     |

|                             |             |
|-----------------------------|-------------|
| 簡易アプリ間通信のサブルーチン.....        | 734         |
| 簡易アプリ間通信のサブルーチンのエラーコード..... | 740         |
| 環境変数情報.....                 | 85,92,567   |
| 環境変数情報の一覧.....              | 92          |
| 環境変数情報の設定.....              | 144         |
| 環境変数情報の追加.....              | 145         |
| 環境変数情報の取消し.....             | 145         |
| 環境変数情報の変更.....              | 145         |
| 環境変数の設定.....                | 4           |
| 環境変数の操作機能.....              | 437,844     |
| 監視ウィンドウ.....                | 672         |
| 監視時間.....                   | 738         |
| 関数値.....                    | 397,401     |
| 関数の型と記述の関係.....             | 1054        |
| 間接参照クラス.....                | 509         |
| 簡略化した動作状態を出力.....           | 98          |
| 関連製品.....                   | 7,294       |
| ガーベージコレクション.....            | 481         |
| 機械語.....                    | 456         |
| 規格の違いによるメッセージの出力.....       | 1016        |
| 既存プログラムのCOMサーバ化.....        | 927         |
| 起動コマンド.....                 | 668,708     |
| 起動パラメタ.....                 | 624,668,708 |
| 機能範囲.....                   | 813         |
| 基本的な使い方.....                | 836         |
| 基本ブロック.....                 | 1050        |
| 逆トレース.....                  | 684         |
| 逆トレース機能.....                | 676         |
| 行間隔.....                    | 311         |
| 競合状態.....                   | 829         |
| 行順ファイル.....                 | 196,894     |
| 行順ファイルの処理.....              | 204         |
| 行順ファイルの使い方.....             | 203         |
| 行順ファイルの定義.....              | 203         |
| 行順ファイルのレコードの定義.....         | 203         |
| 強制クローズ.....                 | 230         |
| 行単位のデータを印刷する方法.....         | 321         |
| 共通式の除去.....                 | 1050        |
| 共通プログラム.....                | 392         |
| 行内呼出し.....                  | 502         |
| 行内呼出しの入れ子.....              | 509         |
| 行番号.....                    | 12,22,597   |
| 行番号内動詞追番.....               | 597         |
| 共用モード.....                  | 228         |
| 行レコード.....                  | 331         |
| キー定義ファイル.....               | 369         |
| キー定義ファイルの記述形式.....          | 370         |
| キー入力定義.....                 | 370         |
| 空白づめ.....                   | 892         |
| クォーテーションマーク.....            | 1032        |
| 組込み関数.....                  | 888         |
| 組込み関数一覧.....                | 1058        |
| 組込み関数の使用.....               | 1054        |
| クライアント.....                 | 689         |
| クライアント操作ウィンドウ.....          | 729         |
| [クライアント定義]ダイアログ.....        | 730         |
| クライアントマシンへのインストール.....      | 934         |



|                          |                 |
|--------------------------|-----------------|
| クラス.....                 | 459,573         |
| クラスID.....               | 974             |
| クラスごとに保存ファイルを分ける.....    | 579             |
| クラス参照の表示.....            | 549             |
| クラス情報.....               | 568             |
| クラス情報データベース.....         | 521             |
| クラス情報データベースの生成.....      | 521             |
| クラス情報データベースファイル名の設定..... | 522             |
| クラス情報の取得.....            | 519             |
| クラス情報ファイル.....           | 567             |
| クラス定義.....               | 468             |
| クラス定義で使用できない機能.....      | 1088            |
| クラスとファイルの対応.....         | 579             |
| クラスのエントリ情報.....          | 562             |
| クラスの検索.....              | 519             |
| クラスの検索条件の入力.....         | 526             |
| クラスの公開.....              | 553             |
| クラスの動的プログラム構造.....       | 558             |
| クラスブラウザ.....             | 3,520           |
| クラスブラウザウィンドウ.....        | 523             |
| クラスブラウザの情報表示.....        | 523             |
| クラスブラウザの表示.....          | 523             |
| クラスブラウザへの入力.....         | 491             |
| クラス名段落.....              | 483             |
| クラスを動的プログラム構造にする.....    | 560             |
| グラフィックデバイスインタフェース.....   | 333             |
| グリッド表示可能なツール.....        | 318             |
| 継承.....                  | 462,481         |
| 継承関係の表示.....             | 524             |
| 継承の概念.....               | 482             |
| 継承の実現.....               | 491             |
| 継承の定義方法.....             | 483             |
| 継承の表示.....               | 549             |
| 桁落とし処理.....              | 1042            |
| 言語要素に対しての指摘メッセージ.....    | 1021            |
| 現在位置表示機能.....            | 677             |
| 現在の日付および時刻の入力.....       | 428             |
| 原文名定数.....               | 1061            |
| 広域最適化.....               | 53,1031,1050    |
| 公開資源.....                | 554             |
| 公開パス.....                | 553             |
| 更新.....                  | 202,212,219,229 |
| 更新項目.....                | 373             |
| 構造化定理.....               | 456             |
| 構造化プログラミング.....          | 456             |
| 高速オーバレイ.....             | 115             |
| 高速オーバレイ機能.....           | 115,312,313     |
| 高度なデータ操作.....            | 762             |
| 高度なデータ操作を可能とするホスト変数..... | 762             |
| 効率のよいプログラム.....          | 1116            |
| 子クラス.....                | 462,483         |
| ゴシックDP.....              | 297             |
| ゴシック体.....               | 297             |
| ゴシック体半角.....             | 297             |
| 固定形式.....                | 13,1038         |
| 固定形式ページ.....             | 341             |
| 固定長レコード形式.....           | 198,200         |

|                             |             |
|-----------------------------|-------------|
| 固定パーティション.....              | 335         |
| コネクション.....                 | 746         |
| コネクション操作.....               | 747         |
| コネクションの接続.....              | 746         |
| コネクションの切断.....              | 747         |
| コネクションの変更.....              | 746         |
| コネクション有効範囲の指定値に対する動作説明..... | 785         |
| コネクション有効範囲の定義内容.....        | 785         |
| コネクションを接続する.....            | 747         |
| コネクションを切断する.....            | 748         |
| コネクションを変更する.....            | 748         |
| コマンド行引数の操作機能.....           | 843         |
| コマンド行引数の取出し.....            | 434         |
| コマンドファイル.....               | 156,194,678 |
| コマンドファイルの作成方法.....          | 678         |
| コマンドファイル名.....              | 667,706     |
| コマンドプロンプト.....              | 44,71,419   |
| コマンドラインで設定.....             | 91          |
| コミット.....                   | 942         |
| コンソールウィンドウ.....             | 1039        |
| コンソールウィンドウの大きさ.....         | 121         |
| コンソールウィンドウの種別.....          | 100         |
| コンソールウィンドウの属性の変更.....       | 419         |
| コンソールウィンドウのバッファ数.....       | 120         |
| コンソールウィンドウのフォント.....        | 120         |
| コンソールウィンドウを使ったデータの入出力.....  | 419         |
| コンテキスト.....                 | 936         |
| コントロールパネルのシステムで設定.....      | 86          |
| コンパイラが出力する情報.....           | 20          |
| コンパイラが使用するファイル.....         | 17          |

[さ]

|                                   |                     |
|-----------------------------------|---------------------|
| 在庫表.....                          | 750                 |
| 最左端ビットの扱い.....                    | 1014                |
| 最終ターゲットファイル.....                  | 165                 |
| 最終ターゲットファイルの登録.....               | 165                 |
| 再デバッグ.....                        | 679                 |
| 再編成.....                          | 239,248             |
| 作業手順.....                         | 345,362             |
| 索引ファイル.....                       | 197,895             |
| 索引ファイル簡易復旧関数.....                 | 288                 |
| 索引ファイル操作クラス.....                  | 582                 |
| 索引ファイルとオブジェクトの対応.....             | 579                 |
| 索引ファイルの処理.....                    | 217                 |
| 索引ファイルの操作.....                    | 247                 |
| 索引ファイルの使い方.....                   | 213                 |
| 索引ファイルの定義.....                    | 214                 |
| 索引ファイルの復旧.....                    | 287                 |
| 索引ファイルのレコードの定義.....               | 216                 |
| 索引ファイル復旧関数.....                   | 288                 |
| 索引ファイル復旧関数(簡易復旧関数)が返却するメッセージ..... | 292                 |
| 削除.....                           | 212,220,229,239,481 |
| サスペンド/レジューム.....                  | 684                 |
| サブスキーマ定義ファイル.....                 | 17,169,665,703      |
| サブスキーマ定義ファイルのフォルダ.....            | 47,1012             |
| サブプログラムを呼び出す.....                 | 376                 |
| 参照.....                           | 202,205,211,219,229 |



|                                   |               |                          |             |
|-----------------------------------|---------------|--------------------------|-------------|
| サンプルデータベース.....                   | 776           | 実行単位の資源.....             | 866         |
| サンプルプログラムの実行.....                 | 80            | 実行単位の終了サブルーチン.....       | 1085        |
| サンプルプログラムの翻訳.....                 | 15            | 実行単位の資源を引き継ぐ方法.....      | 863         |
| サンプルプログラムのリンク.....                | 55            | 実行中断.....                | 679         |
| サーバ.....                          | 689           | 実行の開始と終了.....            | 143         |
| サーバ起動時の画面表示.....                  | 727           | 実行の手順.....               | 82          |
| サーバ識別子.....                       | 736           | 実行用の初期化ファイル.....         | 86,98,223   |
| サーバ終了.....                        | 726           | 実行用の初期化ファイルに設定.....      | 86          |
| サーバ情報の定義内容.....                   | 782           | 実行用の初期化ファイルのオープン.....    | 144         |
| サーバ操作ウィンドウ.....                   | 719           | 実行用の初期化ファイルのクローズ.....    | 145         |
| サーバ・タイプのアプリケーション.....             | 808           | 実行用の初期化ファイルへの保存.....     | 146         |
| サーバ定義ファイルの記述内容.....               | 726           | 実行用の初期化ファイル名の指定形式.....   | 146         |
| サーバディレクトリ.....                    | 707           | 実行用の初期化ファイル名の指定方法.....   | 92          |
| サーバ名.....                         | 736           | 自動デバッグ.....              | 678         |
| サーバ連携情報ページ.....                   | 706           | ジャストインタイムデバッグ機能.....     | 663,693     |
| サービス配下の注意事項.....                  | 84,317        | 自由形式.....                | 14,1038     |
| シグナル.....                         | 682           | 集団項目転記.....              | 892         |
| 資産移行支援.....                       | 890           | 集団項目比較.....              | 893         |
| システム最大格納メッセージ数.....               | 726           | 終了.....                  | 172         |
| システム使用領域.....                     | 308,309       | 終了キー.....                | 370         |
| システムのコンソールウィンドウ.....              | 420           | 終了条件なしのPERFORM文.....     | 453         |
| システムプログラムを記述するための機能.....          | 453           | 終了条件なしのPERFORM文の使い方..... | 455         |
| システム待ち命令数.....                    | 726           | 終了処理メソッド.....            | 507         |
| 実行.....                           | 684           | 主供給口1.....               | 304,305     |
| 実行開始位置変更機能.....                   | 675           | 主供給口2.....               | 304,305     |
| 実行可能ファイル.....                     | 56,58,156     | 主キー.....                 | 198,215     |
| 実行可能プログラムのバージョン設定機能.....          | 187           | 縮小印刷のポートレートモード.....      | 303         |
| 実行環境.....                         | 379,817       | 縮小印刷のランドスケープモード.....     | 303         |
| 実行環境情報.....                       | 85            | 出力項目.....                | 373         |
| 実行環境情報の指定形式.....                  | 92            | 出力メッセージ.....             | 598         |
| 実行環境情報の種類.....                    | 85            | 主プログラム.....              | 52,58,1026  |
| 実行環境情報の設定.....                    | 82,85,781,872 | 主プログラムの指定.....           | 37,168      |
| 実行環境情報の設定方法.....                  | 85            | 主レコードキー.....             | 198,215     |
| 実行環境設定ツール.....                    | 3,143         | 順呼出し法.....               | 209,214     |
| 実行環境設定ツールで設定.....                 | 91            | 障害発生箇所の特定方法.....         | 647         |
| 実行環境設定ツールの終了.....                 | 146           | 条件を指定して参照する.....         | 753         |
| 実行環境の開設と閉鎖.....                   | 380           | 小数点位置固定入力.....           | 369         |
| 実行環境の構築.....                      | 780           | 小入出力.....                | 893         |
| 実行環境の閉鎖サブルーチン.....                | 1086          | 小入出力機能.....              | 417         |
| 実行監視条件での中断.....                   | 657           | 小入出力機能の出力ファイル.....       | 134         |
| 実行経路をたどる.....                     | 676           | 小入出力機能の入出力先.....         | 417         |
| 実行時オプション.....                     | 122,148       | 小入出力機能の入力ファイル.....       | 134         |
| 実行時オプションの指定形式.....                | 148           | 情報隠蔽.....                | 458         |
| 実行時オプションの指定方法.....                | 92            | 情報表示.....                | 722         |
| 実行時コード系.....                      | 1032          | 使用メモリの節約.....            | 565         |
| 実行に必要なとなるエントリ情報.....              | 563           | 初期化処理メソッド.....           | 507         |
| 実行時の適合チェック.....                   | 490           | 初期化ファイル.....             | 18          |
| 実行時メッセージ.....                     | 636           | 初期化プログラム.....            | 393         |
| 実行時メッセージおよびSYSERRの出力抑止.....       | 124           | 除数のゼロ検査.....             | 602         |
| 実行時メッセージの重大度.....                 | 112           | 書体番号.....                | 297         |
| 実行時メッセージの重大度コードとイベントログの種類の対応..... | 811           | 処理種別.....                | 738         |
| 実行時メッセージの重大度指定.....               | 111           | 処理種別の種類.....             | 715         |
| 実行時メッセージの出力先.....                 | 110           | シリアルポート名.....            | 325,327     |
| 実行時メッセージをイベントログに出力する機能.....       | 811           | 字類条件.....                | 888         |
| 実行性能の向上.....                      | 566           | 自論理宛先.....               | 717         |
| 実行単位.....                         | 379,817       | シングルスレッドモード.....         | 827,1076    |
| 実行単位の開始サブルーチン.....                | 1084          | 診断機能.....                | 51,595,1040 |



|                             |                 |                                 |                    |
|-----------------------------|-----------------|---------------------------------|--------------------|
| 単純構造で結合する.....              | 407,409,411,412 | デバッグしたいプログラムからのデバッガ起動方法.....    | 663,691            |
| 単純構造の実行可能プログラムを作成する.....    | 62              | デバッグ情報ファイル.....                 | 18,659             |
| 単数翻訳ウィンドウ.....              | 34              | デバッグ情報ファイルの格納先.....             | 622                |
| ダンプとは.....                  | 643             | デバッグ情報ファイルのフォルダ.....            | 48                 |
| ダンプの出力先.....                | 643             | デバッグ情報ページ.....                  | 666,705            |
| ダンプファイル数の管理.....            | 644             | デバッグ操作の再現.....                  | 657                |
| ターゲットオブジェクトファイル.....        | 982             | デバッグ操作の自動化.....                 | 678                |
| ターゲット名の指定.....              | 65              | デバッグ対象プログラム.....                | 658                |
| ターゲットリポジトリ検索.....           | 537             | デバッグに必要な資源.....                 | 663                |
| ターゲットリポジトリファイル.....         | 534             | デバッグの準備.....                    | 658                |
| ターゲットリポジトリファイルの登録手順.....    | 534             | デバッグモジュールの作成方法.....             | 173                |
| 注意事項.....                   | 1121            | [デバッグを開始する]ダイアログ.....           | 663,701            |
| 注記行.....                    | 12              | デフォルトFCB名.....                  | 121                |
| 抽象クラス.....                  | 464             | デフォルトコネクション情報の定義内容.....         | 784                |
| 抽象データ型.....                 | 457             | 電子帳票.....                       | 350                |
| 中断点.....                    | 684             | 電子帳票出力.....                     | 315                |
| 注文表.....                    | 750             | 電子帳票出力機能.....                   | 130,350            |
| 帳票設計.....                   | 318             | テンプレート展開機能.....                 | 525,551            |
| 帳票定義体.....                  | 309,334,343     | データエリアに関するリスト.....              | 26                 |
| 帳票定義体に設定する情報.....           | 346             | データ型の対応.....                    | 403,414            |
| 帳票定義体の作成.....               | 346             | データ監視機能.....                    | 675                |
| 帳票定義体名.....                 | 303             | データ監視/条件監視.....                 | 684                |
| 帳票定義体を使う印刷ファイルの使い方.....     | 334             | データ項目の属性の選択.....                | 1116               |
| 帳票の電子化.....                 | 337             | データ操作.....                      | 746,749            |
| 重複文字の扱い.....                | 1019            | データソース.....                     | 781                |
| 追尾速度.....                   | 679             | データソースとの接続の確認.....              | 790                |
| 通過カウント点.....                | 684             | データソースの作成.....                  | 790                |
| 通貨編集用文字の扱い.....             | 1018            | データ入力方法の定義.....                 | 370                |
| 通信機能.....                   | 712             | データのアクセス.....                   | 483                |
| [通信システム環境設定]ダイアログ.....      | 719             | データの共用.....                     | 389                |
| 通信の種類.....                  | 712             | データの更新.....                     | 756                |
| 通信ポート名.....                 | 130             | データの削除.....                     | 756                |
| 定義行.....                    | 22              | データの参照.....                     | 751                |
| 定義済みオブジェクト一意名SELF.....      | 497             | データの挿入.....                     | 756                |
| 定義済みオブジェクト一意名SUPER.....     | 497             | データの内容を確認しながらプログラムの誤りを検出する..... | 674                |
| 定義体ページ.....                 | 665,704         | データのファイル入出力.....                | 680                |
| 定数項目.....                   | 373             | データの変更箇所を調べる.....               | 675                |
| 適合.....                     | 487             | データ変更での中断.....                  | 657                |
| 適合チェック.....                 | 488,489         | データベース.....                     | 744                |
| 適合チェックの実現.....              | 492             | データベース生成.....                   | 523                |
| 適合の概念.....                  | 487             | データマッピングリスト.....                | 26,1027            |
| デッドロック.....                 | 806             | データ名.....                       | 200,224,327        |
| デバッガの概要.....                | 657             | データロックサブルーチン.....               | 878                |
| デバッガの画面.....                | 669             | 同一表を関連付けて参照する.....              | 755                |
| デバッガの環境変数.....              | 659             | 動作環境.....                       | 360                |
| デバッガの起動.....                | 662             | 動的SQL.....                      | 757                |
| デバッガの機能.....                | 674             | 動的カーソル宣言.....                   | 757                |
| デバッガのコマンド一覧.....            | 687             | 動的構造.....                       | 59,529             |
| デバッガの状態を知る.....             | 677             | 動的束縛.....                       | 465                |
| デバッガの操作.....                | 669             | 動的パラメタ.....                     | 759                |
| デバッグ.....                   | 1119            | 動的プログラム構造.....                  | 60,382,558,682,885 |
| デバッグ環境の変更.....              | 679             | 動的プログラム構造で結合する.....             | 408,409,411,413    |
| デバッグ機能の概要.....              | 595             | 動的プログラム構造による呼出し.....            | 879,881            |
| デバッグ機能のマルチスレッドでの基本的な動作..... | 683             | 動的プログラム構造の作成から実行.....           | 560                |
| デバッグ機能のマルチスレッド特有な動作.....    | 683             | 動的プログラム構造の実行可能プログラムを作成する.....   | 63                 |
| デバッグ作業の流れ.....              | 645             | 動的プログラム構造の特徴.....               | 558                |
| デバッグ事象を通知する/通知しないの切り替え..... | 684             | 動的呼出し法.....                     | 209,214            |

|                             |                 |
|-----------------------------|-----------------|
| 動的リンク構造.....                | 59,682          |
| 動的リンク構造で結合する.....           | 407,409,411,412 |
| 動的リンク構造の実行可能プログラムを作成する..... | 62              |
| 登録集.....                    | 264             |
| 登録集機能.....                  | 1024            |
| 登録集原文.....                  | 13              |
| 登録集原文の作成.....               | 13              |
| 登録集原文の表示.....               | 1017            |
| 登録集ファイル.....                | 17,156          |
| 登録集ファイルの拡張子.....            | 1024            |
| 登録集ファイルの検索方法.....           | 20              |
| 登録集ファイルの登録.....             | 168             |
| 登録集ファイルのフォルダ.....           | 51,1024         |
| 登録集名の指定があるCOPY文.....        | 20              |
| 登録集名の指定がないCOPY文.....        | 20              |
| 登録集名の設定.....                | 40              |
| [登録集名の設定]ダイアログ.....         | 40              |
| 登録集名の追加.....                | 41              |
| [登録集名の追加]ダイアログ.....         | 41              |
| 登録集名の取消し.....               | 42              |
| 登録集名の変更.....                | 41              |
| 特殊クラス.....                  | 584             |
| 特殊な定数の書き方.....              | 1061            |
| 特殊レジスタ.....                 | 310             |
| 特殊レジスタの使い方.....             | 366             |
| 特徴.....                     | 813             |
| 特定実行条件での中断.....             | 657             |
| 特定の条件.....                  | 805             |
| とじしろ幅.....                  | 307             |
| とじしろ方向.....                 | 307             |
| ドライブ名.....                  | 1061,1062       |
| トランザクション.....               | 746             |
| トランザクションコンテキストオブジェクト.....   | 938             |
| トランザクション処理.....             | 746             |
| トレース採取機能.....               | 676             |
| トレース情報.....                 | 607             |
| トレース情報の格納先.....             | 607             |
| トレース情報の個数.....              | 148,607         |
| トレース情報の出力形式.....            | 608             |
| トレース情報の出力ファイル.....          | 119             |
| トレースログ.....                 | 728             |

## [な]

|                         |      |
|-------------------------|------|
| 内部プログラム.....            | 392  |
| 内部プログラム呼出しのパラメタの検査..... | 604  |
| 名前付きパラメタ.....           | 915  |
| 名前の有効範囲.....            | 393  |
| 名前を付けて保存.....           | 172  |
| 二次入口点.....              | 387  |
| 二次入口点一覧の表示.....         | 550  |
| 二次入口点名の指定形式.....        | 141  |
| 日本語16進文字定数.....         | 891  |
| 日本語空白.....              | 1029 |
| 日本語空白の比較方法.....         | 1029 |
| 日本語文字定数.....            | 888  |
| 日本語利用者語の文字集合.....       | 1028 |
| 入出力.....                | 1118 |

|                         |         |
|-------------------------|---------|
| 入出力エラーが発生したときの実行結果..... | 222     |
| 入出力エラーの実行時メッセージの出力..... | 108     |
| 入出力状態一覧.....            | 1046    |
| 入出力状態のチェック.....         | 220     |
| 入出力処理の種類.....           | 348,364 |
| 入出力エラー処理.....           | 220     |
| 入力項目.....               | 373     |
| 任意の供給口.....             | 304     |
| 任意の日付の入力.....           | 429     |
| 任意の日付を取得.....           | 108     |

## [は]

|                                  |                             |
|----------------------------------|-----------------------------|
| バイト単位の領域長.....                   | 1014                        |
| 配付ソース生成.....                     | 986                         |
| 廃要素.....                         | 1016                        |
| バインド方法選択の基準.....                 | 908                         |
| パス名.....                         | 1061,1062                   |
| バックグラウンド処理.....                  | 808                         |
| バッチデバッグ.....                     | 657,678                     |
| バッチデバッグページ.....                  | 667,705                     |
| バッチデバッグを行う.....                  | 667,706                     |
| バッチファイルを使用したCOBOLプログラムの実行方法..... | 83                          |
| パラメタの受渡し方法.....                  | 387,395,401                 |
| パラメタの指定.....                     | 478                         |
| パラメタの省略.....                     | 915                         |
| パラメタを指定して実行.....                 | 173                         |
| バリエーション型データ.....                 | 959                         |
| 汎用ログに出力.....                     | 99                          |
| 汎用ログ.....                        | 431                         |
| パーティション形式.....                   | 334                         |
| 被デバッグプログラム.....                  | 658                         |
| 表意定数.....                        | 891                         |
| 表意定数QUOTEの扱い.....                | 1032                        |
| 表示.....                          | 239                         |
| 標識領域.....                        | 12                          |
| 表指定ホスト変数.....                    | 767                         |
| 表示ファイル.....                      | 295,897                     |
| 表示ファイル(画面入出力)の使い方.....           | 359                         |
| 表示ファイル(帳票印刷)の使い方.....            | 343                         |
| 表示ファイル(非同期型メッセージ通信)の使い方.....     | 713                         |
| 表示ファイル機能.....                    | 359,713                     |
| 表示ファイル機能(非同期型メッセージ通信).....       | 712                         |
| 表示ファイル単体テスト.....                 | 3                           |
| 表示ファイルの入力中断.....                 | 368                         |
| 標準のプリンタ.....                     | 325                         |
| 表の全行を参照する.....                   | 751                         |
| 表を関連付けてデータを参照する.....             | 754                         |
| ビルダ.....                         | 192                         |
| ビルダのエラージャンプ機能.....               | 42                          |
| ビルダのエラージャンプ機能の使い方.....           | 44                          |
| ビルド.....                         | 156,172                     |
| ビルド用JCL/CLISTの生成.....            | 998                         |
| ファイルからの読込み.....                  | 680                         |
| ファイル管理記述項に指定する情報.....            | 340,347,364,714             |
| ファイル記述項に指定する情報.....              | 323,341                     |
| ファイル参照子.....                     | 199                         |
| ファイル識別名.....                     | 127,128,129,200,223,326,894 |

|                            |                  |                              |                         |
|----------------------------|------------------|------------------------------|-------------------------|
| ファイル識別名定数.....             | 200,224,326,1061 | フォームオーバーレイパターンの出力.....       | 333                     |
| ファイルシステム.....              | 277              | フォームオーバーレイパターンのファイルの拡張子..... | 133                     |
| ファイルシステム種別.....            | 127              | フォームオーバーレイパターンのファイル名.....    | 133                     |
| ファイル終了条件発生の検出.....         | 220              | フォームオーバーレイパターンのファイル名の形式..... | 132                     |
| ファイル終了条件の発生.....           | 221              | フォームオーバーレイパターンのフォルダ.....     | 132                     |
| ファイル処理の結果.....             | 229              | フォームオーバーレイパターン名.....         | 302,309                 |
| ファイル処理の実行.....             | 223              | フォームオーバーレイパターン名の個数.....      | 309                     |
| ファイル追加書き.....              | 234              | フォームオーバーレイパターン焼付け回数.....     | 302                     |
| ファイル定義体.....               | 13               | フォームオーバーレイパターンを使うプログラム.....  | 332                     |
| ファイル定義体の検索順序.....          | 20               | 副供給口.....                    | 304                     |
| ファイル定義体ファイル.....           | 17,51,156,169    | 副キー.....                     | 198,215                 |
| ファイル定義体ファイルの拡張子.....       | 1019             | 復元.....                      | 583                     |
| ファイル定義体ファイルのフォルダ.....      | 51,1020          | 複写.....                      | 239                     |
| ファイルの移動.....               | 246              | 複写数.....                     | 303                     |
| ファイルの印刷.....               | 246              | 複数行指定ホスト変数.....              | 763                     |
| ファイルの拡張.....               | 240              | 複数コネクションでのカーソル操作.....        | 761                     |
| ファイルの構造の変換.....            | 246              | 複数のソースファイルを翻訳する.....         | 33                      |
| ファイルの高速処理.....             | 127,230          | 複数翻訳ウィンドウ.....               | 35                      |
| ファイルの高速処理を一括して有効にする指定..... | 107              | 複数翻訳の指定.....                 | 35                      |
| ファイルの削除.....               | 245              | 副プログラム.....                  | 58,1026                 |
| ファイルの指定.....               | 66,69            | 副プログラム名の指定形式.....            | 138                     |
| ファイルの種類.....               | 195              | 副レコードキー.....                 | 198,215                 |
| ファイルの種類と処理.....            | 198              | 符号付き10進項目の符号の整形.....         | 1036                    |
| ファイルの種類と特徴.....            | 195              | 符号付き外部10進項目と英数字項目の比較.....    | 1043                    |
| ファイルの処理.....               | 195              | 復帰コード.....                   | 391,397,401             |
| ファイルの処理方法.....             | 198              | 復旧.....                      | 239,248                 |
| ファイルの操作.....               | 245              | 不定形式ページ.....                 | 341                     |
| ファイルの創成.....               | 239              | 不定ファイル.....                  | 229                     |
| ファイルの追加書き.....             | 426              | 浮動小数点項目.....                 | 1121                    |
| ファイルの登録.....               | 165              | 浮動パーティション.....               | 335                     |
| ファイルの内容の修正.....            | 173              | 部分参照検査.....                  | 601                     |
| ファイルの排他処理.....             | 95               | 不変式の移動.....                  | 1051                    |
| ファイルの排他制御.....             | 199,226          | ブリコンパイラ.....                 | 898                     |
| ファイルの複写.....               | 245              | [ブリコンパイラ設定]ダイアログ.....        | 177                     |
| ファイルの連結.....               | 234              | ブリコンパイラの登録.....              | 173                     |
| ファイルの割当て.....              | 223              | ブリコンパイラを使用したアクセス.....        | 744                     |
| ファイルへの書込み.....             | 681              | ブリコンパイルの実行.....              | 173                     |
| ファイル編成.....                | 200,203,208,214  | プリンタ.....                    | 129                     |
| ファイル名.....                 | 199              | プリンタ情報ファイル.....              | 349                     |
| ファイル名の指定.....              | 36,142           | プリンタ情報ファイルの作成.....           | 349                     |
| ファイルを使うプログラム.....          | 423              | プリンタ情報ファイルの設定情報.....         | 349                     |
| ファイルを排他モードにする方法.....       | 226              | プリンタの設定.....                 | 146                     |
| ファウンデーションクラス.....          | 7                | プリンタ名.....                   | 130,324,326,327,328,329 |
| ファクトリオブジェクト.....           | 470,822,832      | [プリンタ名の選択]ダイアログ.....         | 146                     |
| ファクトリオブジェクトの寿命.....        | 480              | プログラミングスタイル.....             | 456                     |
| ファクトリ定義.....               | 469              | プログラムID.....                 | 913,974                 |
| ファクトリメソッド.....             | 473,573,582      | プログラム一覧ウィンドウ.....            | 671                     |
| ファンクションキーの利用者定義.....       | 369              | プログラム開発の手順.....              | 7                       |
| フォルダ.....                  | 170              | プログラムからデバッグまたは診断機能を起動.....   | 96                      |
| フォルダの作成.....               | 170              | プログラム間結合.....                | 1119                    |
| フォルダの種類.....               | 170              | プログラム間連絡機能.....              | 376                     |
| フォントテーブル.....              | 320              | プログラム構造.....                 | 58,527                  |
| フォントテーブル名.....             | 130              | プログラム構造の概要.....              | 528                     |
| フォーマット定義体.....             | 1069             | プログラム構造の指定.....              | 1018                    |
| フォームオーバーレイ印刷方法.....        | 115              | プログラム制御情報リスト.....            | 29,1027                 |
| フォームオーバーレイおよびFCBを使う方法..... | 329              | プログラムの記述.....                | 363,372                 |
| フォームオーバーレイパターン.....        | 299,329          | プログラムの形式.....                | 13                      |



|                              |                 |                           |            |
|------------------------------|-----------------|---------------------------|------------|
| プログラムの作成.....                | 12              | 保存.....                   | 583        |
| プログラムの実行.....                | 367,374,413,780 | 保存するオブジェクトのメソッドの追加.....   | 582        |
| プログラムの実行順序を変更する.....         | 675             | 翻訳エラーの修正.....             | 42         |
| プログラムの実行とスレッドモード.....        | 827             | 翻訳オプション.....              | 1008,1044  |
| プログラムのデバッグ.....              | 595             | 翻訳オプション一覧.....            | 1008       |
| プログラムの特徴に合わせたデバッグ方法.....     | 682             | [翻訳オプション]ダイアログ.....       | 38,162     |
| プログラムの翻訳.....                | 406,659         | 翻訳オプションの指定.....           | 54         |
| プログラムの翻訳・リンク.....            | 367,374,621,780 | 翻訳オプションの指定形式.....         | 1010       |
| プログラムの翻訳・リンク方法と出力情報との関係..... | 620             | 翻訳オプションの設定.....           | 37,38,162  |
| プログラムの呼出し経路をたどる.....         | 677             | 翻訳オプションの追加.....           | 39         |
| プログラムのリンク.....               | 406,659         | 翻訳オプションの取消し.....          | 40         |
| プログラム名定数.....                | 1061            | 翻訳オプションの変更.....           | 40         |
| プログラム呼出し規約検査.....            | 603             | 翻訳オプションファイル.....          | 156,547    |
| プログラムを作成・編集する.....           | 12              | 翻訳コマンド.....               | 44         |
| プログラムを実行する.....              | 80              | 翻訳コマンドのオプション.....         | 46         |
| プログラムを翻訳する.....              | 15              | 翻訳指示文.....                | 14,34,1010 |
| プログラムをリンクする.....             | 55              | 翻訳時の適合チェック.....           | 490        |
| プロジェクト依存情報.....              | 547             | 翻訳時メッセージ.....             | 44         |
| プロジェクト管理.....                | 175,527,977     | 翻訳単位.....                 | 527        |
| プロジェクト管理機能.....              | 156             | 翻訳単位ごとのオブジェクトファイルの出力..... | 1028       |
| プロジェクト管理機能で使用するファイル.....     | 157             | 翻訳単位統計情報リスト.....          | 21,1027    |
| プロジェクト管理に必要な資源.....          | 156             | 翻訳単位の指定.....              | 35         |
| プロジェクト管理の手順.....             | 159             | 翻訳に必要な資源.....             | 17         |
| プロジェクト管理のメリット.....           | 527             | 翻訳の開始と終了.....             | 37         |
| プロジェクト情報データベース.....          | 547             | 翻訳の手順.....                | 33,533     |
| プロジェクト情報データベース生成.....        | 547             | 翻訳の方法.....                | 560        |
| プロジェクトファイル.....              | 156             | 翻訳リスト.....                | 21         |
| プロジェクトファイルの互換性.....          | 1096            | 翻訳リストの1行あたりの文字数.....      | 1025       |
| プロジェクトファイルの作成.....           | 162             | 翻訳リストの1ページあたりの行数.....     | 1025       |
| プロジェクトファイルの保存.....           | 172             | 翻訳リストの出力.....             | 53         |
| プロジェクトブラウザ.....              | 3,546           | 翻訳リストファイル.....            | 18,53      |
| プロジェクトブラウザウィンドウ.....         | 548             | 翻訳リストファイルのフォルダ.....       | 49         |
| プロジェクトブラウザの情報表示.....         | 548             | ポート名.....                 | 1062       |
| プロジェクトブラウザの表示.....           | 548             | ポートレートモード.....            | 303        |
| プロジェクトマネージャ.....             | 3,7,9           |                           |            |
| プロジェクトマネージャウィンドウ.....        | 160             |                           |            |
| プロジェクトを閉じる.....              | 172             |                           |            |
| プロセスID取得サブルーチン.....          | 1074            |                           |            |
| プロセス指定.....                  | 1078            |                           |            |
| プロセス終了サブルーチン.....            | 1082            |                           |            |
| ブロック化.....                   | 565             |                           |            |
| プロパティ.....                   | 573             |                           |            |
| プロパティ指定子.....                | 915             |                           |            |
| [プロパティ]ダイアログの[全般]シート.....    | 324             |                           |            |
| 文書名識別情報.....                 | 105,308         |                           |            |
| 分離されたメソッド.....               | 500             |                           |            |
| 分離翻訳されるメソッド定義で使用できない機能.....  | 1089            |                           |            |
| 併合.....                      | 439             |                           |            |
| 別表からの複数行の挿入.....             | 757             |                           |            |
| 別翻訳単位定義記号.....               | 22              |                           |            |
| 変換.....                      | 238             |                           |            |
| 編集.....                      | 239             |                           |            |
| ポインタ修飾子.....                 | 453             |                           |            |
| ポインタ付け.....                  | 453             |                           |            |
| ポインタ付けの使い方.....              | 453             |                           |            |
| ポインタデータ項目.....               | 454             |                           |            |
| ホストのコード系.....                | 1030            |                           |            |

[ま]

|                          |          |
|--------------------------|----------|
| マクロ.....                 | 184      |
| 待ち合わせ指示.....             | 738      |
| 待ち合わせなし指示.....           | 738      |
| マルチスレッドでのデバッガの対応.....    | 683      |
| マルチスレッドでのデバッグ操作の自動化..... | 684      |
| マルチスレッドの効果.....          | 815      |
| マルチスレッドのデバッグを補助する機能..... | 684      |
| マルチスレッドのメリット.....        | 813      |
| マルチスレッドプログラム.....        | 814      |
| マルチスレッドプログラム作成.....      | 1041     |
| マルチスレッドプログラムの外部属性.....   | 1036     |
| マルチスレッドプログラムの基本動作.....   | 817      |
| マルチスレッドプログラムへの移行.....    | 839      |
| マルチスレッドモード.....          | 827,1078 |
| マージ.....                 | 439      |
| マージ処理の種類.....            | 443      |
| マージの使い方.....             | 443      |
| 明朝体.....                 | 297      |
| 明朝体半角.....               | 297      |
| 無限ループ.....               | 455      |
| 無効キー条件の発生.....           | 221      |

|                             |            |
|-----------------------------|------------|
| 無効キー条件発生を検出.....            | 220        |
| 無駄な代入の除去.....               | 1052       |
| メイク.....                    | 156        |
| 名標.....                     | 22         |
| メインウィンドウ.....               | 669        |
| メソッド.....                   | 459        |
| メソッド一覧の表示.....              | 524,550    |
| メソッド定義.....                 | 473        |
| メソッドのPROTOTYPE宣言.....       | 500        |
| メソッドの上書き.....               | 486        |
| メソッドのエントリ情報.....            | 562        |
| メソッドの行内呼出し.....             | 502        |
| メソッドの静的束縛.....              | 494        |
| メソッドの束縛.....                | 493        |
| メソッドの動的束縛.....              | 494        |
| メソッドの動的プログラム構造.....         | 559        |
| メソッドの呼出し.....               | 460,475    |
| メソッド呼出し時の適合チェック.....        | 490        |
| メソッド呼出しのパラメタの検査.....        | 603        |
| メソッドを動的プログラム構造にする.....      | 561        |
| メッセージの優先順位.....             | 740        |
| メッセージボックス.....              | 422        |
| メッセージボックスにメッセージを出力する方法..... | 422        |
| メッセージを出力するファイル.....         | 123        |
| メモリ資源の見積り.....              | 732        |
| メモリチェック機能.....              | 595        |
| メモリチェック機能の使い方.....          | 615        |
| メモリチェック機能を使って検査を行う.....     | 113        |
| メモリに関するチューニング.....          | 565        |
| メモリ割当てサブルーチン.....           | 1075       |
| メンバ関数呼出しインタフェースプログラム.....   | 574        |
| メンバ変数参照/設定インタフェースプログラム..... | 574        |
| 目的プログラムの出力.....             | 1031       |
| 目的プログラムリスト.....             | 24,621     |
| 目的プログラムリストの出力の可否.....       | 1026       |
| 目的プログラムリストを使ったデバッグ.....     | 595,645    |
| 文字間隔.....                   | 311        |
| 文字行内配置時の上端/下端合わせ.....       | 119        |
| 文字コード系.....                 | 1103       |
| 文字の泣き別れ.....                | 893        |
| 文字配置座標の計算方法.....            | 117        |
| モジュール化.....                 | 456        |
| モジュール定義ファイル.....            | 70,170,191 |
| モジュール定義ファイルの自動生成.....       | 71         |
| モジュール定義ファイルの内容.....         | 70         |
| モジュール定義文.....               | 70         |
| 文字列を扱うクラス.....              | 592        |
| モード.....                    | 35         |

## [や]

|               |         |
|---------------|---------|
| 誘導変数の最適化..... | 1051    |
| 用紙供給口.....    | 304     |
| 用紙サイズ.....    | 125,304 |
| 用紙名.....      | 125     |
| 横書き.....      | 298     |
| 呼出し関係の概要..... | 376     |
| 呼出し関係の形態..... | 376     |

|                  |                 |
|------------------|-----------------|
| 呼出し規約.....       | 376             |
| 呼出し規約と指定形式.....  | 377             |
| 呼出し規約の違い.....    | 377             |
| 呼出し経路ウィンドウ.....  | 673             |
| 呼出し経路表示.....     | 677             |
| 呼出しプログラムの表示..... | 550             |
| 呼出し方法.....       | 387,395,401,414 |
| 呼び名.....         | 295             |
| 予約語の種類.....      | 1034            |

## [ら]

|                               |                 |
|-------------------------------|-----------------|
| ライブラリ.....                    | 56,156          |
| ライブラリ結合の開始と終了.....            | 69              |
| ライブラリ名.....                   | 73              |
| ライブラリ名の指定.....                | 69              |
| ライブラリを作成する.....               | 63              |
| ラインコマンド入力ウィンドウ.....           | 672             |
| ラインプリンタモード.....               | 303             |
| ランドスケープモード.....               | 303             |
| 乱呼出し法.....                    | 209,214         |
| リソース情報ファイル.....               | 156             |
| 資源の共有.....                    | 830             |
| リビルド.....                     | 156,172         |
| リポジトリ.....                    | 491             |
| リポジトリ検索.....                  | 536             |
| リポジトリ段落.....                  | 53              |
| リポジトリ段落のクラス指定子.....           | 525             |
| リポジトリファイル.....                | 18,156,521,531  |
| リポジトリファイル更新の影響.....           | 493             |
| リポジトリファイル収集パス.....            | 522             |
| リポジトリファイル収集パスの設定.....         | 522             |
| リポジトリファイルの概要.....             | 491             |
| リポジトリファイルの出力.....             | 531             |
| リポジトリファイルの入出力先フォルダ.....       | 50,1033         |
| リポジトリファイルの入力.....             | 531             |
| リポジトリファイルの入力先フォルダ.....        | 53,1033         |
| リモートデバッグ.....                 | 689,700,701,710 |
| リモートデバッグコネクタ.....             | 693,696         |
| リモートデバッグの注意事項.....            | 710,711         |
| リモートデバッグ時の資産の格納場所.....        | 690             |
| リモートデバッグの開始手順.....            | 690             |
| リモートデバッグを許可するコンピュータを制限する..... | 699             |
| リモートデータベースアクセス.....           | 744             |
| 領域破壊が発生するアプリケーションに対する起動.....  | 623             |
| 領域破壊監視.....                   | 640             |
| 領域破壊監視の場合.....                | 633             |
| 利用者定義の情報をイベントログに出力する機能.....   | 811             |
| 利用者定義のファンクションキー.....          | 373             |
| [リンクオプション]ダイアログ.....          | 66,163          |
| リンクオプションの設定.....              | 66,163          |
| リンクオプションファイル.....             | 156             |
| リンクコマンドの形式.....               | 73              |
| リンクコマンドの使用例.....              | 71              |
| リンクコマンドを使ったリンク操作.....         | 71              |
| リンク単位.....                    | 527             |
| リンクに必要な資源.....                | 56              |
| リンクの開始と終了.....                | 67              |

|                          |                 |
|--------------------------|-----------------|
| リンクの手順.....              | 61              |
| リンクの方法.....              | 561             |
| リンクメッセージ.....            | 78              |
| 隣接転記の統合.....             | 1052            |
| 例外オブジェクト.....            | 555             |
| 例外条件.....                | 438             |
| 例外処理.....                | 555             |
| 例外ハンドラ.....              | 682             |
| レイトバインド.....             | 906             |
| レコードキー.....              | 198             |
| レコード形式.....              | 198,200         |
| レコード順ファイル.....           | 195             |
| レコード順ファイルの処理.....        | 201             |
| レコード順ファイルの使い方.....       | 199             |
| レコード順ファイルの定義.....        | 199             |
| レコード順ファイルのレコードの定義.....   | 200             |
| レコード長.....               | 200             |
| レコードの構成.....             | 201,204,210,216 |
| レコードの整列.....             | 245             |
| レコードの設計.....             | 197             |
| レコードの追加.....             | 240             |
| レコードの表示.....             | 242             |
| レコードの編集.....             | 243             |
| レコードを排他状態にする方法.....      | 228             |
| 連携プログラムの構造.....          | 570             |
| 連絡節獲得.....               | 679             |
| ログ取得.....                | 723,727         |
| [ログ取得操作]ダイアログ.....       | 723             |
| ログ種別.....                | 728             |
| ログ定義ファイル.....            | 98              |
| ログ定義参照ファイル.....          | 99              |
| ログの出力形式.....             | 724             |
| ログファイル名.....             | 728             |
| ロック.....                 | 829             |
| ロックキー.....               | 879,880         |
| ロックを解放.....              | 880,881         |
| ロックを獲得.....              | 879,881         |
| 論理宛先削除.....              | 721             |
| [論理宛先情報]ダイアログ.....       | 722             |
| 論理宛先創成.....              | 720             |
| 論理宛先定義ファイル.....          | 98              |
| 論理宛先定義ファイル作成ユーティリティ..... | 729             |
| [論理宛先の削除]ダイアログ.....      | 721             |
| [論理宛先の創成]ダイアログ.....      | 720             |
| 論理宛先のモード変更.....          | 721             |
| [論理宛先のモード変更]ダイアログ.....   | 721             |
| 論理宛先名.....               | 738             |
| ローカルプリンタポート名.....        | 130,325         |
| ローカルポート名.....            | 326,328,329     |
| ロード.....                 | 239             |
| ロールバック.....              | 942             |
| [わ]                      |                 |
| ワード単位の領域長.....           | 1014            |