

# FUJITSU Software

## NetCOBOL V12.0

# ユーザーズガイド

Windows

B1WD-3444-01Z0(00)  
2017年8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://www.fujitsu.com/jp/software/cobol/>

## 登録商標について

- Microsoft、Windows、Windows Server、Visual C++、Visual Basic、SQL Server、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。
- Btrieve および Pervasive は Actian Corporation の登録商標です。Pervasive Software、Pervasive.SQL、Pervasive PSQL はActian Corporation の商標です。
- 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 Server(R) 2016 Datacenter Microsoft(R) Windows Server(R) 2016 Standard Microsoft(R) Windows Server(R) 2016 Essentials	Windows Server 2016
Microsoft(R) Windows Server(R) 2012 R2 Datacenter Microsoft(R) Windows Server(R) 2012 R2 Standard Microsoft(R) Windows Server(R) 2012 R2 Essentials Microsoft(R) Windows Server(R) 2012 R2 Foundation	Windows Server 2012 R2
Microsoft(R) Windows Server(R) 2012 Datacenter Microsoft(R) Windows Server(R) 2012 Standard Microsoft(R) Windows Server(R) 2012 Essentials Microsoft(R) Windows Server(R) 2012 Foundation	Windows Server 2012
Microsoft(R) Windows Server(R) 2008 R2 Datacenter Microsoft(R) Windows Server(R) 2008 R2 Enterprise Microsoft(R) Windows Server(R) 2008 R2 Standard Microsoft(R) Windows Server(R) 2008 R2 Foundation	Windows Server 2008 R2

正式名称	略称
Windows(R) 10 Education Windows(R) 10 Home Windows(R) 10 Pro Windows(R) 10 Enterprise	Windows 10 または Windows 10(x64)
Windows(R) 8.1 Windows(R) 8.1 Pro Windows(R) 8.1 Enterprise	Windows 8.1 または Windows 8.1(x64)
Windows(R) 7 Home Premium Windows(R) 7 Professional Windows(R) 7 Enterprise Windows(R) 7 Ultimate	Windows 7 または Windows 7(x64)
Microsoft(R) Visual C++(R) development system	Visual C++
Microsoft(R) Visual Basic(R) programming system	Visual Basic
Red Hat(R) Enterprise Linux(R) 6(for Intel64) Red Hat(R) Enterprise Linux(R) 7(for Intel64)	Linux(64)
Red Hat(R) Enterprise Linux(R) 5(for x86) Red Hat(R) Enterprise Linux(R) 6(for x86)	Linux
Oracle Solaris 10 Oracle Solaris 11	Solaris
PowerSORT PowerSORT Server PowerSORT Workstation	PowerSORT
Microsoft(R) Internet Information Services	IIS

- 以下をすべて指す場合は、「Windows」または「Windows(32)」と表記します。

- Windows Server 2016
- Windows Server 2012 R2
- Windows Server 2012
- Windows Server 2008 R2
- Windows 10
- Windows 8.1
- Windows 7

- 以下をすべて指す場合は、「Windows(64)」と表記します。

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

— Windows 7(x64)

- Solarisシステムで動作し、32ビットCOBOLアプリケーションを開発・運用するシステムを、「Solaris 32bit版 NetCOBOL」と表記します。Solaris 32bit版 NetCOBOLが動作するOracle Solarisを「Solaris」または「Solaris(32)」と表記します。
- Linuxで動作し、32ビットCOBOLアプリケーションを開発・運用するシステムを、「Linux 32bit版 NetCOBOL」と表記します。
- Windows(64)システムで動作し、64ビットCOBOLアプリケーションを開発・運用するシステムを、「Windows 64bit版 NetCOBOL」と表記します。

## 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部 NetCOBOLとは

#### 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部 プログラムの編集から翻訳・リンク・実行まで

#### 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3部 アプリケーションの開発と運用

#### 第6章 文字コード

NetCOBOLの文字コード系の扱い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 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16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17章 オブジェクト指向プログラミング機能

オブジェクト指向プログラミングについて説明しています。

## 第4部 サーバサイドアプリケーションの開発と運用

### 第18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19章 Web連携

各種WWWサーバの特徴とCOBOL Webサブルーチンの使用方法について説明しています。

### 第20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5部 テスト支援機能／デバッグ支援機能

### 第21章 テスト支援機能

NetCOBOLのテスト支援機能について説明しています。

### 第22章 デバッグ支援機能

NetCOBOLのデバッグ支援機能について説明しています。

## 第6部 プロジェクトマネージャを利用した開発

### 第23章 プロジェクト管理機能

プロジェクトマネージャを使用した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24章 対話型デバッグの使い方

NetCOBOLの対話型デバッグの使い方について説明しています。

### 第25章 ツールの操作

プロジェクトマネージャから使用できるツールについて説明しています。

## 第26章 Webアプリケーション開発支援機能の利用

プロジェクトマネージャを使用したWebアプリケーション開発について説明しています。

## 第27章 オブジェクト指向プログラミングの開発

プロジェクトマネージャを使用したオブジェクト指向プログラミング開発について説明しています。

## 第28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。

## 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 翻訳リスト

COBOLコンパイラが出力する翻訳リストについて説明しています。

## 付録C 環境変数情報

環境変数情報について説明しています。

## 付録D 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E 定量制限

NetCOBOLの定量制限について説明しています。

## 付録F 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G 特殊な定数および利用者語の書き方

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H 関数

NetCOBOLで使用できる関数について説明しています。

## 付録I サブルーチン

NetCOBOLが提供するサブルーチンについて説明しています。

## 付録J コマンド

NetCOBOLが提供するコマンドについて説明しています。

## 付録K 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L 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M 文字コードの留意点

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 Webアプリケーション

COBOL Webサブルーチンを利用するために基本的な事柄、Webサブルーチンの使用時のQ/A集、CGIからISAPIへの移行方法、注意事項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)	コンポーネント名	使用目的
PowerCOBOL ユーザーズガイド PowerCOBOL リファレンス	PowerCOBOL	GUI画面を持つCOBOLアプリケーションの設計および実行
FORM ユーザーズガイド FORMヘルプ (注2) PowerFORMヘルプ (注2)	FORM PowerFORM FORMオーバーレイオプション	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
MeFt ユーザーズガイド	MeFt	画面帳票定義体を使用したプログラムの実行
MeFt/Web ユーザーズガイド	MeFt/Web	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ユーザーズガイド	Jアダプタクラスジェネレータ	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ユーザーズガイド	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 Server ユーザーズガイド	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。

## 注意

各製品に含まれるマニュアル一覧は、“ソフトウェア説明書”を参照してください。

“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>あいうえお</u>	プログラム例中で、可変文字列を示します。可変文字列は、実際には他の文字列に置き換えます。  例： PROGRAM-ID. <u>プログラム名</u> . → PROGRAM-ID. SAMPLE1.
<u>あ</u> または { <u>あ</u>   <u>うえお</u> }	{ }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。

## 輸出管理について

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

2017年8月

Copyright 1992-2017 FUJITSU LIMITED

All Rights Reserved, Copyright(C) Microsoft Corporation. 1992-2017

## 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部 NetCOBOLとは.....	1
第1章 NetCOBOLの概要.....	2
1.1 NetCOBOLの機能.....	2
1.1.1 COBOLの機能.....	2
1.1.2 NetCOBOLが提供するプログラムおよびユーティリティ.....	2
1.2 開発環境.....	5
1.2.1 環境変数の設定.....	5
1.2.2 関連製品.....	8
1.2.3 資源一覧.....	9
1.3 プログラム開発.....	10
1.3.1 NetCOBOL Studioを利用した開発.....	10
1.3.2 プロジェクトマネージャを利用した開発.....	11
1.3.3 開発環境を利用しない開発.....	13
1.4 リモート開発.....	13
第2部 プログラムの編集から翻訳・リンク・実行まで.....	14
第2章 プログラムを作成・編集する.....	15
2.1 プログラムの作成.....	15
2.1.1 COBOLソースプログラムの作成/編集.....	15
2.1.2 登録集原文の作成/編集.....	16
2.1.3 プログラムの形式.....	16
2.1.4 翻訳指示文.....	17
第3章 プログラムを翻訳する.....	18
3.1 サンプルプログラムの翻訳.....	18
3.2 翻訳に必要な資源.....	18
3.3 翻訳の手順.....	21
3.4 翻訳コマンド.....	22
第4章 プログラムをリンクする.....	23
4.1 サンプルプログラムのリンク.....	23
4.2 リンクに必要な資源.....	23
4.2.1 リンクコマンドが使用するファイル.....	23
4.2.2 実行可能ファイル.....	25
4.2.3 DLL (ダイナミックリンクライブラリ).....	25
4.2.4 インポートライブラリ.....	25
4.3 プログラム構造.....	25
4.3.1 静的構造.....	26
4.3.2 動的構造.....	26
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.....	27
4.3.4 プログラム構造とCANCEL文の関係.....	27
4.4 モジュール定義ファイル.....	28
4.4.1 モジュール定義ファイルの内容.....	28
4.4.2 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.....	29
4.5 リンクコマンドを使ったリンク操作.....	29
4.5.1 LINKコマンド.....	29
4.5.2 リンクコマンドの使用例.....	29
4.5.3 インポートライブラリの結合.....	31
第5章 プログラムを実行する.....	35
5.1 サンプルプログラムの実行.....	35
5.2 実行の手順.....	36
5.3 実行環境情報の設定.....	36
5.3.1 実行環境情報の種類.....	36
5.3.2 実行環境情報の設定方法.....	36

5.4 プログラムの実行	37
5.4.1 コマンドラインから実行する	38
5.4.2 バッチファイルを使用する	39
5.4.3 サービス配下での注意事項(実行時)	40
5.5 実行用の初期化ファイル	41
5.5.1 実行用の初期化ファイルとは	41
5.5.2 実行用の初期化ファイルの内容	41
5.5.3 実行用の初期化ファイルの検索順序	43
5.5.4 DLL配下にある実行用の初期化ファイルを使用する	44
5.5.5 主プログラムが他言語の場合の実行用の初期化ファイル名の指定方法	46
5.5.5.1 主プログラムがCプログラムの場合	46
5.5.5.2 主プログラムがVisual Basicプログラムの場合	46
5.6 エントリ情報	47
5.6.1 エントリ情報とは	47
5.6.2 エントリ情報の指定方法	48
5.6.3 エントリ情報の形式	48
5.6.3.1 副プログラム名の指定形式	48
5.6.3.2 二次入口点名の指定形式	48
5.6.4 エントリ情報ファイルの内容	49
5.6.5 エントリ情報ファイル名の指定	49
5.6.6 実行用の初期化ファイルに含まれるエントリ情報の内容	49
5.6.7 エントリ情報の設定例	50
5.7 実行環境設定ツール	54
5.7.1 環境変数情報の設定	55
5.7.1.1 実行用の初期化ファイルのオープン	55
5.7.1.2 環境変数情報の追加	56
5.7.1.3 環境変数情報の変更	56
5.7.1.4 環境変数情報の取消し	56
5.7.1.5 実行用の初期化ファイルのクローズ	56
5.7.2 エントリ情報の設定	57
5.7.3 実行用の初期化ファイルへの保存	57
5.7.4 プリンタの設定	57
5.7.5 実行環境設定ツールの終了	57
5.8 実行時オプション	57
5.9 注意事項	59
5.9.1 COBOLプログラムの実行時にスタックオーバフローが発生する場合	59
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	62
<b>第3部 アプリケーションの開発と運用</b>	<b>64</b>
<b>第6章 文字コード</b>	<b>65</b>
6.1 文字コードの概念	65
6.2 文字コードの指定	67
6.2.1 文字データのエンコード	67
6.2.2 エンコードの指定	68
6.2.3 翻訳オプションによるエンコードの指定	69
6.2.4 実行時コード系	69
6.2.5 資源	70
6.3 言語要素	73
6.3.1 異なるエンコード間の比較	73
6.3.2 異なるエンコード間の転記	73
6.3.3 字類条件	74
6.4 UTF-32エンコードを使用できない機能	74
6.5 実行時の注意点	74
6.5.1 フォントについて	74
6.6 関連製品連携	75
6.6.1 FORM、MeFt	75
6.6.2 他言語間結合	76

6.6.3 他のファイルシステム	77
6.6.4 プリコンパイラ	78
6.6.5 リモートデータベースアクセス(ODBC)	78
6.6.6 Interstage Business Application Server	78
6.6.7 Interstage Application Server	78
6.6.8 Interstage Job Workload Server	78
<b>第7章 ファイルの処理</b>	<b>79</b>
7.1 ファイルの種類	79
7.1.1 ファイルの種類と特徴	79
7.1.2 レコードの設計	82
7.1.2.1 レコード形式	82
7.1.2.2 索引ファイルのレコードキー	82
7.1.3 ファイルの処理方法	82
7.1.4 Unicodeデータの扱い	83
7.2 レコード順ファイルの使い方	84
7.2.1 レコード順ファイルの定義	84
7.2.2 レコード順ファイルのレコードの定義	85
7.2.3 レコード順ファイルの処理	87
7.3 行順ファイルの使い方	88
7.3.1 行順ファイルの定義	88
7.3.2 行順ファイルのレコードの定義	88
7.3.3 行順ファイルの処理	89
7.4 相対ファイルの使い方	92
7.4.1 相対ファイルの定義	93
7.4.2 相対ファイルのレコードの定義	94
7.4.3 相対ファイルの処理	95
7.5 索引ファイルの使い方	98
7.5.1 索引ファイルの定義	98
7.5.2 索引ファイルのレコードの定義	100
7.5.3 索引ファイルの処理	101
7.6 入出力エラー処理	105
7.6.1 AT END指定	105
7.6.2 INVALID KEY指定	105
7.6.3 FILE STATUS句	106
7.6.4 誤り処理手続き	106
7.6.5 入出力エラーが発生したときの実行結果	107
7.7 ファイル処理の実行	107
7.7.1 ファイルの割当て	107
7.7.2 ファイルの排他制御	112
7.7.2.1 ファイルを排他モードにする方法	112
7.7.2.2 レコードを排他状態にする方法	113
7.7.3 ファイル処理の結果	115
7.7.4 ファイルの高速処理	115
7.7.5 ファイル追加書き	119
7.7.6 ファイルの連結	119
7.7.7 ダミーファイル	120
7.7.8 注意事項	121
7.8 COBOLファイルユーティリティ	123
7.8.1 COBOLファイルユーティリティとは	123
7.8.2 COBOLファイルユーティリティの使い方	124
7.8.3 COBOLファイルユーティリティの機能	126
7.8.3.1 ファイルの創成	126
7.8.3.2 ファイルの拡張	126
7.8.3.3 レコードの表示	128
7.8.3.4 レコードの編集	129
7.8.3.5 レコードの整列	131

7.8.3.6	ファイルの操作	131
7.8.3.7	ファイルの印刷	132
7.8.3.8	ファイルの構造の変換	132
7.8.3.9	索引ファイルの操作	132
7.9	他のファイルシステムの使用法	134
7.9.1	Btrieveファイル	137
7.9.1.1	ファイル環境の指定	137
7.9.1.2	外部10進項目のデータ形式変換	138
7.9.1.3	注意事項	140
7.9.2	PowerRDBconnector	140
7.9.2.1	ファイル環境の指定	140
7.9.2.2	注意事項	141
7.9.3	外部ファイルハンドラ	141
7.10	索引ファイルの復旧	143
7.11	COBOLファイルアクセスルーチン	143
<b>第8章</b>	<b>印刷処理</b>	<b>144</b>
8.1	印刷方法の種類	144
8.1.1	各印刷方法の概要	145
8.1.2	印刷ファイル/表示ファイルの決定方法	147
8.1.3	印字文字	148
8.1.4	帳票設計について	152
8.1.5	印字文字の配置座標	153
8.1.6	印刷不可能な領域について	154
8.1.7	フォームオーバーレイパターン	155
8.1.8	FCB	155
8.1.9	I制御レコード/S制御レコード	157
8.1.10	帳票定義体	164
8.1.11	特殊レジスタ	165
8.1.12	印刷情報ファイル	166
8.1.13	フォントテーブル	171
8.1.14	Unicodeの印刷について	172
8.1.15	サービス配下の注意事項(印刷時)	173
8.2	行単位のデータを印刷する方法	174
8.2.1	概要	174
8.2.2	プログラムの記述	174
8.2.3	プログラムの翻訳・リンク	176
8.2.4	プログラムの実行	176
8.3	フォームオーバーレイおよびFCBを使う方法	182
8.3.1	概要	183
8.3.2	プログラムの記述	183
8.3.3	プログラムの翻訳・リンク	185
8.3.4	プログラムの実行	185
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	185
8.3.4.2	FCBを使うプログラム	186
8.4	帳票定義体を使う印刷ファイルの使い方	187
8.4.1	概要	187
8.4.1.1	帳票のパーティション	188
8.4.1.2	帳票の電子化	190
8.4.2	プログラムの記述	192
8.4.3	プログラムの翻訳・リンク	195
8.4.4	プログラムの実行	195
8.5	表示ファイル(帳票印刷)の使い方	196
8.5.1	概要	196
8.5.2	作業手順	197
8.5.3	帳票定義体の作成	197
8.5.4	プログラムの記述	198

8.5.5 プログラムの翻訳・リンク	201
8.5.6 プリンタ情報ファイルの作成	201
8.5.7 プログラムの実行	201
8.6 電子帳票出力機能を使う方法	202
8.6.1 電子帳票出力機能の概要	202
8.6.2 帳票を電子化する方法	202
8.6.3 電子帳票の出力例	206
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)	206
8.6.5 実行時エラーについて	209
<b>第9章 画面を使った入出力</b>	<b>210</b>
9.1 画面を使った入出力の種類	210
9.2 表示ファイル(画面入出力)の使い方	210
9.2.1 概要	210
9.2.2 動作環境	211
9.2.3 作業手順	212
9.2.4 画面定義体の作成	212
9.2.5 プログラムの記述	213
9.2.6 プログラムの翻訳・リンク	217
9.2.7 ウィンドウ情報ファイルの作成	217
9.2.8 プログラムの実行	217
9.3 スクリーン操作機能の使い方	218
9.3.1 概要	218
9.3.2 スクリーンウィンドウ	218
9.3.3 キー定義ファイルの利用	219
9.3.4 プログラムの記述	221
9.3.5 プログラムの翻訳・リンク	223
9.3.6 プログラムの実行	223
9.3.7 Unicodeデータの扱い	224
<b>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～</b>	<b>225</b>
10.1 呼出し関係の概要	225
10.1.1 呼出し関係の形態	225
10.1.2 各種呼出し規約の違い	225
10.1.3 呼出し規約と指定形式	226
10.1.4 COBOLの言語間の環境	228
10.1.5 動的プログラム構造	231
10.1.5.1 動的プログラム構造の特徴	231
10.1.5.2 副プログラムのエントリ情報	232
10.1.5.3 注意事項	232
10.2 COBOLプログラムからCOBOLプログラムを呼び出す	235
10.2.1 呼出し方法	235
10.2.2 二次入口点	235
10.2.3 制御の復帰とプログラムの終了	235
10.2.4 パラメタの受渡し方法	235
10.2.5 データの共用	237
10.2.5.1 外部データ使用時の注意事項	238
10.2.5.2 外部ファイル使用時の注意事項	238
10.2.6 復帰コード	239
10.2.7 内部プログラム	240
10.2.8 注意事項	241
10.3 C言語プログラムとの結合	242
10.3.1 COBOLプログラムからCプログラムを呼び出す方法	243
10.3.1.1 呼出し方法	243
10.3.1.2 パラメタの受渡し方法	243
10.3.1.3 復帰コード(関数値)	245
10.3.2 CプログラムからCOBOLプログラムを呼び出す方法	248
10.3.2.1 呼出し方法	249

10.3.2.2	パラメタの受渡し方法	249
10.3.2.3	復帰コード(関数値)	249
10.3.3	データ型の対応	251
10.3.4	プログラムの翻訳	254
10.3.5	プログラムのリンク	254
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	255
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	258
10.3.6	プログラムの実行	261
10.4	Visual BasicからCOBOLプログラムを呼び出す	261
10.4.1	呼出し方法	261
10.4.2	データ型の対応	261
10.4.3	Visual BasicからCOBOLの呼出し例	261
<b>第11章</b>	<b>ACCEPT文およびDISPLAY文の使い方</b>	<b>264</b>
11.1	小入出力機能	264
11.1.1	概要	264
11.1.2	入出力先の種類と指定方法	264
11.1.3	Unicodeデータの扱い	265
11.1.4	コンソールウィンドウを使ったデータの入出力	266
11.1.4.1	COBOLのコンソールウィンドウ	266
11.1.4.2	コマンドプロンプト	267
11.1.4.3	システムのコンソールウィンドウ	267
11.1.4.4	プログラムの記述	268
11.1.4.5	プログラムの翻訳・リンク	268
11.1.4.6	プログラムの実行	269
11.1.5	メッセージボックスにメッセージを出力する方法	269
11.1.5.1	メッセージボックス	269
11.1.5.2	プログラムの記述	269
11.1.5.3	プログラムの翻訳・リンク	270
11.1.5.4	プログラムの実行	270
11.1.6	ファイルを使うプログラム	270
11.1.6.1	プログラムの記述	271
11.1.6.2	プログラムの翻訳・リンク	272
11.1.6.3	プログラムの実行	273
11.1.6.4	DISPLAY文のファイル出力拡張機能	273
11.1.6.5	ACCEPT文のファイル入力拡張機能	274
11.1.7	現在の日付および時刻の入力	276
11.1.7.1	プログラムの記述	276
11.1.7.2	プログラムの翻訳・リンク	277
11.1.7.3	プログラムの実行	277
11.1.8	任意の日付の入力	277
11.1.8.1	プログラムの記述	277
11.1.8.2	プログラムの翻訳・リンク	278
11.1.8.3	プログラムの実行	278
11.1.9	Interstage Business Application Serverの汎用ログを使うプログラム	278
11.1.9.1	プログラムの記述	279
11.1.9.2	プログラムの翻訳・リンク	279
11.1.9.3	プログラムの実行	279
11.1.10	イベントログを使うプログラム	280
11.1.10.1	プログラムの記述	280
11.1.10.2	プログラムの翻訳・リンク	280
11.1.10.3	プログラムの実行	280
11.2	コマンド行引数の取出し	282
11.2.1	概要	282
11.2.2	プログラムの記述	283
11.2.3	プログラムの翻訳・リンク	284
11.2.4	プログラムの実行	285

11.3 環境変数の操作機能.....	285
11.3.1 概要.....	285
11.3.2 プログラムの記述.....	285
11.3.3 プログラムの翻訳・リンク.....	286
11.3.4 プログラムの実行.....	286
<b>第12章 SORT文およびMERGE文の使い方～整列併合機能～.....</b>	<b>287</b>
12.1 ソート・マージ処理の概要.....	287
12.2 ソートの使い方.....	288
12.2.1 ソート処理の種類.....	288
12.2.2 プログラムの記述.....	288
12.2.3 プログラムの翻訳・リンク.....	291
12.2.4 プログラムの実行.....	291
12.3 マージの使い方.....	291
12.3.1 マージ処理の種類.....	291
12.3.2 プログラムの記述.....	292
12.3.3 プログラムの翻訳・リンク.....	294
12.3.4 プログラムの実行.....	294
<b>第13章 CSV形式データの操作.....</b>	<b>295</b>
13.1 CSV形式データとは.....	295
13.2 CSV形式データの作成 (STRING文).....	296
13.2.1 基本操作.....	296
13.2.2 処理異常の検出.....	297
13.3 CSV形式データの分解 (UNSTRING文).....	297
13.3.1 基本操作.....	297
13.3.2 処理異常の検出.....	298
13.4 CSV形式のバリエーション.....	299
<b>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～.....</b>	<b>301</b>
14.1 SD機能の種類.....	301
14.2 ポインタ付けの使い方.....	301
14.3 ADDR関数およびLENG関数の使い方.....	302
14.4 終了条件なしのPERFORM文の使い方.....	303
<b>第15章 リモートデータベースアクセス.....</b>	<b>304</b>
15.1 プリコンパイラを使用したアクセス.....	304
15.2 ODBC経由によるアクセス.....	304
15.2.1 概要.....	304
15.2.1.1 COBOLプログラムの構成.....	305
15.2.1.2 埋込みSQL文による操作.....	306
15.2.2 コネクション操作.....	307
15.2.2.1 コネクションを接続する.....	307
15.2.2.2 コネクションを切断する.....	308
15.2.2.3 コネクションを変更する.....	308
15.2.3 データ操作.....	309
15.2.3.1 サンプルデータベース.....	309
15.2.3.2 データの参照.....	311
15.2.3.2.1 表の全行を参照する.....	311
15.2.3.2.2 条件を指定して参照する.....	313
15.2.3.2.3 1つの行を参照する.....	314
15.2.3.2.4 表を関連付けてデータを参照する.....	314
15.2.3.3 データの更新.....	316
15.2.3.4 データの削除.....	316
15.2.3.5 データの挿入.....	316
15.2.3.6 動的SQL.....	317
15.2.3.7 可変長文字列の使用.....	320
15.2.3.8 複数コネクションでのカーソル操作.....	321

15.2.4 高度なデータ操作.....	322
15.2.4.1 高度なデータ操作を可能とするホスト変数.....	322
15.2.4.1.1 複数行指定ホスト変数.....	323
15.2.4.1.2 表指定ホスト変数.....	327
15.2.4.2 動的SQL文で使用方法.....	329
15.2.4.3 SQLERRDによる処理行数の確認方法.....	329
15.2.4.4 FOR句による処理行数制御.....	330
15.2.4.5 スクロール可能なカーソルを使用したデータの取得.....	332
15.2.5 ストアドプロシージャの呼出し.....	340
15.2.5.1 ストアドプロシージャとは.....	340
15.2.5.2 ストアドプロシージャの呼出し例.....	341
15.2.6 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.....	341
15.2.6.1 サンプルデータベース.....	341
15.2.6.2 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.....	342
15.2.6.3 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.....	343
15.2.7 プログラムの翻訳・リンク.....	345
15.2.8 プログラムの実行.....	345
15.2.8.1 実行環境の構築.....	345
15.2.8.1.1 実行環境情報の設定.....	346
15.2.8.1.2 ODBC情報ファイルの作成.....	346
15.2.8.2 ODBC情報設定ツールの使い方.....	351
15.2.8.3 ODBC情報ファイルの設定内容の最大長.....	353
15.2.8.4 連携ソフトウェアおよびハードウェア環境の整備.....	353
15.2.9 埋込みSQL文のキーワード一覧.....	354
15.2.10 SQL文と指定可能なホスト変数.....	360
15.2.11 ODBCで扱うデータとの対応.....	361
15.2.12 SQLSTATE/SQLCODE/SQLMSG.....	364
15.2.13 ODBCドライバ使用時の注意事項.....	365
15.2.13.1 SQL文の文法上の制限事項.....	365
15.2.13.2 埋込みSQL文の実行時の注意事項.....	366
15.2.13.3 各ODBCドライバ固有の留意事項.....	367
15.2.13.4 埋込みSQL文の実行時の定量制限.....	369
15.2.14 デッドロック出口.....	370
15.2.14.1 デッドロック出口スケジュールの概要.....	370
15.2.14.2 注意事項.....	370
<b>第16章 通信機能.....</b>	<b>372</b>
16.1 通信の種類.....	372
16.2 簡易アプリ間通信機能の使い方.....	372
16.2.1 概要.....	372
16.2.2 作業手順.....	373
16.2.3 サーバ操作ウィンドウ.....	375
16.2.3.1 [通信システム環境設定]ダイアログ.....	375
16.2.3.2 簡易アプリ間通信サーバウィンドウ.....	375
16.2.3.2.1 論理宛先創成.....	376
16.2.3.2.2 論理宛先削除.....	376
16.2.3.2.3 論理宛先のモード変更.....	376
16.2.3.2.4 情報表示.....	377
16.2.3.2.5 ログ取得.....	377
16.2.3.2.6 サーバ終了.....	379
16.2.3.3 サーバ定義ファイルを利用した起動.....	380
16.2.4 クライアント操作ウィンドウ.....	383
16.2.4.1 論理宛先定義ファイル作成ユーティリティ.....	383
16.2.5 メモリ資源の見積り.....	385
16.2.6 注意事項.....	386
16.3 簡易アプリ間通信のサブルーチン.....	387
<b>第17章 オブジェクト指向プログラミング機能.....</b>	<b>388</b>

17.1 基本的な使い方.....	388
17.1.1 ソース定義.....	388
17.1.1.1 クラス定義.....	388
17.1.1.2 ファクトリ定義.....	389
17.1.1.3 オブジェクト定義.....	391
17.1.1.4 メソッド定義.....	392
17.1.2 オブジェクトインスタンスの操作.....	394
17.1.2.1 メソッドの呼出し.....	394
17.1.2.1.1 オブジェクト参照項目.....	395
17.1.2.1.2 INVOKE文.....	396
17.1.2.1.3 パラメタの指定.....	397
17.1.2.2 オブジェクトの寿命.....	399
17.1.3 継承.....	400
17.1.3.1 継承の概念と実現.....	401
17.1.3.2 FJBASEクラス.....	403
17.1.3.3 メソッドの上書き.....	404
17.1.4 適合.....	405
17.1.4.1 適合の概念.....	405
17.1.4.2 オブジェクト参照項目と適合チェック.....	407
17.1.4.3 翻訳時の適合チェックと実行時の適合チェック.....	408
17.1.4.3.1 代入時の適合チェック.....	408
17.1.4.3.2 メソッド呼出し時の適合チェック.....	408
17.1.5 リポジトリ.....	409
17.1.5.1 リポジトリファイルの概要.....	409
17.1.5.1.1 クラスブラウザへの入力.....	409
17.1.5.2 継承の実現.....	409
17.1.5.2.1 適合チェックの実現.....	410
17.1.5.3 リポジトリファイル更新の影響.....	411
17.1.6 メソッドの束縛.....	411
17.1.6.1 メソッドの静的束縛.....	412
17.1.6.2 メソッドの動的束縛と多態.....	412
17.1.6.3 定義済みオブジェクト一意名SUPER.....	413
17.1.6.4 定義済みオブジェクト一意名SELF.....	414
17.1.7 メソッドのPROTOTYPE宣言.....	416
17.1.8 多重継承.....	417
17.1.9 行内呼出し.....	419
17.1.10 オブジェクト指定子.....	420
17.1.11 PROPERTY句.....	421
17.1.12 初期化処理メソッドと終了処理メソッド.....	423
17.1.13 間接参照クラス.....	425
17.1.14 相互参照クラス.....	426
17.1.14.1 相互参照パターン.....	426
17.1.14.2 相互参照クラスの翻訳.....	429
17.1.14.3 相互参照クラスのリンク.....	430
17.1.14.4 相互参照クラスの実行.....	432
17.2 オブジェクト指向プログラミングの開発.....	432
17.2.1 オブジェクト指向プログラミングで使用する資源.....	432
17.2.2 開発手順.....	433
17.2.3 使用するクラスの選定.....	434
17.2.3.1 プログラム構造.....	435
17.2.3.1.1 翻訳単位とリンク単位.....	435
17.2.3.1.2 プログラム構造の概要.....	435
17.2.3.2 翻訳処理.....	438
17.2.3.2.1 リポジトリファイル.....	438
17.2.3.3 リンク処理.....	442
17.2.3.3.1 インポートライブラリ.....	442
17.2.4 クラスの公開.....	446

17.3 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～	447
17.3.1 例外処理	447
17.3.1.1 概要	447
17.3.1.2 例外オブジェクト	447
17.3.1.3 RAISE文の動作	448
17.3.1.4 RAISING指定のEXIT文の動作	449
17.3.2 動的プログラム構造	450
17.3.2.1 動的プログラム構造の特徴	450
17.3.2.2 クラスの動的プログラム構造	450
17.3.2.3 メソッドの動的プログラム構造	451
17.3.2.4 動的プログラム構造の作成から実行	452
17.3.2.4.1 翻訳の方法	452
17.3.2.4.2 リンクの方法	453
17.3.2.4.3 DLLの構成とファイル名	453
17.3.2.4.4 クラスとメソッドのエントリ情報	453
17.3.3 メモリに関するチューニング	456
17.3.3.1 概要	456
17.3.3.2 使用メモリの節約	457
17.3.3.3 実行性能の向上	458
17.3.3.4 メモリのチューニングに関する実行環境情報	459
17.3.3.4.1 クラス情報	459
17.3.3.4.2 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)	459
17.3.4 Visual C++プログラムとの連携	459
17.3.4.1 概要	459
17.3.4.2 Visual C++連携の方法	459
17.3.4.3 Visual C++連携の概要	459
17.3.4.3.1 COBOLおよびVisual C++でのクラスの対応	460
17.3.4.3.2 処理の概要	460
17.3.4.3.3 インタフェースプログラムの仕組み	460
17.3.4.4 Visual C++連携のプログラム手順	462
17.3.4.4.1 Visual C++で定義されているクラスを調べる	462
17.3.4.4.2 COBOL側での定義	462
17.3.4.4.3 Visual C++側での定義	463
17.3.4.5 COBOLからの利用	463
17.3.4.6 サンプルプログラム	464
17.3.5 オブジェクトの永続化	466
17.3.5.1 オブジェクトの永続化とは	466
17.3.5.2 概要	466
17.3.5.3 クラス構造の例	467
17.3.5.4 索引ファイルとオブジェクトの対応	468
17.3.5.4.1 クラスとファイルの対応	468
17.3.5.4.2 索引ファイルの定義	470
17.3.5.5 オブジェクトの保存/復元	470
17.3.5.5.1 索引ファイル操作クラス	470
17.3.5.5.2 保存するオブジェクトのメソッドの追加	471
17.3.5.5.3 処理の流れ	472
17.3.6 特殊クラス	473
17.3.6.1 *COB-BINDTABLEクラス	473
17.3.6.1.1 概要	473
17.3.6.1.2 *COB-BINDTABLEクラスの説明	475
17.3.6.1.3 *COB-BINDTABLEクラスのファクトリメソッド	475
17.3.6.1.4 *COB-BINDTABLEクラスのオブジェクトメソッド	476
17.3.7 ANY LENGTH句を使用したプログラミング	481
17.3.7.1 文字列を扱うクラス	481
17.3.7.2 ANY LENGTH句の使用	482
17.4 オブジェクト指向と従来機能の組合せ	483
17.4.1 クラス定義で使用できない機能	483

17.4.2 分離翻訳されるメソッド定義で使用できない機能.....	484
<b>第4部 サーバサイドアプリケーションの開発と運用.....</b>	<b>485</b>
<b>第18章 サーバ・タイプのアプリケーション.....</b>	<b>486</b>
18.1 バックグラウンド処理.....	486
18.1.1 画面による入出力操作の抑止方法.....	486
18.1.2 コマンドプロンプトウィンドウの使用法.....	487
18.1.3 サービス配下で動作するプログラム.....	487
18.2 Web連携.....	488
18.3 イベントログ.....	489
18.3.1 実行時メッセージをイベントログに出力する機能.....	489
18.3.2 利用者定義の情報をイベントログに出力する機能.....	489
<b>第19章 Web連携.....</b>	<b>491</b>
19.1 概要.....	491
19.1.1 インターネット/イントラネット環境の活用.....	491
19.1.2 Webアプリケーション.....	492
19.1.3 Web連携の利用方法.....	493
19.1.4 各Web連携機能の特徴.....	493
19.1.5 各Web連携機能の詳細情報.....	494
19.2 MeFt/Webを利用したアプリケーション.....	494
19.2.1 MeFt/Web利用の目的.....	494
19.2.2 MeFt/Webの利用方法.....	494
19.3 COBOL Webサブルーチンを利用したアプリケーションの概要.....	495
19.3.1 Webアプリケーションが利用できるインタフェースの種類.....	495
19.3.2 Webアプリケーションの処理の流れ.....	495
19.3.3 CGIとISAPIの相違点.....	496
19.3.4 COBOL Webサブルーチンの特徴.....	497
19.3.5 COBOL Webサブルーチン.....	498
19.4 CGIサブルーチンを使用したアプリケーションの作成と実行.....	498
19.4.1 CGIの概要.....	498
19.4.2 CGIの処理の流れ.....	498
19.4.3 COBOL CGIサブルーチンとは.....	499
19.4.4 必要なもの.....	499
19.4.5 作成するもの.....	499
19.4.5.1 CGIサブルーチンを使用したCOBOLプログラム.....	499
19.4.5.1.1 メインプログラム.....	500
19.4.5.1.2 初期プログラム.....	500
19.4.5.1.3 業務プログラム.....	500
19.4.5.1.4 終了プログラム.....	502
19.4.5.2 呼出し用ページ.....	503
19.4.5.3 結果出力用ページ.....	503
19.4.6 作成する際の注意点.....	503
19.4.7 実行手順.....	504
19.4.7.1 翻訳およびリンク.....	504
19.4.7.2 CGIサブルーチンの環境変数設定.....	505
19.4.7.3 CGIアプリケーションの実行.....	506
19.5 ISAPIサブルーチンを使用したアプリケーションの作成と実行.....	506
19.5.1 ISAPIの概要.....	506
19.5.2 ISAの構成.....	507
19.5.3 COBOL ISAPIサブルーチンとは.....	507
19.5.4 必要なもの.....	507
19.5.5 作成するもの.....	508
19.5.5.1 GetExtensionVersion.....	508
19.5.5.2 HttpExtensionProc.....	509
19.5.5.3 TerminateExtension.....	511
19.5.5.4 呼出し用ページ.....	512

19.5.5.5 結果出力用ページ	512
19.5.6 ISAを作成する際の注意点	512
19.5.7 実行手順	513
19.5.7.1 翻訳およびリンク	513
19.5.7.2 ISAPIサブルーチンの環境変数設定	514
19.5.7.3 ISAPIアプリケーション (ISA) の実行	515
19.6 WebアプリケーションにおけるUnicodeの利用	515
19.6.1 Unicodeの利用	515
19.6.2 Webブラウザ	516
19.6.3 Webブラウザからの入力コード系	516
19.6.4 Webブラウザへの出力コード系	516
19.6.5 翻訳オプション	517
19.6.6 使用可能なサブルーチン	517
19.6.7 呼出し用ページに使用できるHTML文書のコード系	517
19.6.8 結果出力用ページに使用できるHTML文書のコード系	518
19.6.9 ログファイルのコード系	518
19.6.10 コード系の異なるCOBOLアプリケーションの混在	518
19.7 COBOL Webサブルーチン	518
19.7.1 Webサブルーチンの提供するクラス	518
19.7.2 IISの設定方法	519
19.8 動作確認	525
19.8.1 ログ情報を参照する	525
19.8.2 対話型デバッガで動作確認する	525
19.8.2.1 CGIアプリケーションの動作確認	525
19.8.2.2 ISAPIアプリケーションの動作確認	526
19.8.2.2.1 デバッガを起動する	526
19.8.2.2.2 デバッグを行う	527
19.8.2.2.3 デバッグを終了する	527
19.8.2.2.4 Webアプリケーションを入れ替える	527
19.8.3 NetCOBOL Studioのデバッグ機能で動作確認する	528
19.8.3.1 CGIアプリケーションの動作確認	528
19.8.3.2 ISAPIアプリケーションの動作確認	528
19.8.3.2.1 デバッグを開始する	529
19.8.3.2.2 デバッグを行う	529
19.8.3.2.3 デバッグを終了する	530
19.8.3.2.4 Webアプリケーションを入れ替える	530
19.8.4 Webサブルーチンが検出したエラーを参照する	530
19.8.5 実行中のデータを表示形式で参照する	531
<b>第20章 マルチスレッド</b>	<b>532</b>
20.1 概要	532
20.1.1 特徴	532
20.1.2 機能範囲	532
20.2 マルチスレッドのメリット	532
20.2.1 スレッドとは	532
20.2.2 マルチスレッドプログラムとは	533
20.2.3 マルチスレッドの効果	534
20.3 マルチスレッドプログラムの基本動作	535
20.3.1 実行環境と実行単位	535
20.3.2 マルチスレッドプログラムのデータの扱い	537
20.3.2.1 プログラム定義に宣言されたデータ	538
20.3.2.2 ファクトリオブジェクトとオブジェクトインスタンス	539
20.3.2.3 メソッド定義に宣言されたデータ	541
20.3.2.4 スレッド間共有外部データと外部ファイル	541
20.3.3 プログラムの実行とスレッドモード	542
20.4 スレッド間の資源の共有	543
20.4.1 競合状態	543

20.4.2 資源の共有.....	545
20.4.2.1 スレッド間共有外部データと外部ファイル.....	545
20.4.2.2 ファクトリオブジェクト.....	546
20.4.2.3 オブジェクトインスタンス.....	548
20.5 基本的な使い方.....	549
20.5.1 入出力機能の利用.....	550
20.5.2 リモートデータベースアクセス(ODBC)の利用.....	552
20.5.3 プリコンパイラの利用によるSymfoware連携.....	552
20.5.4 DISPLAY文およびACCEPT文の利用.....	553
20.5.4.1 小入出力機能について.....	553
20.5.4.2 コマンド行引数および環境変数の操作機能について.....	556
20.5.4.2.1 コマンド行引数の操作機能.....	556
20.5.4.2.2 環境変数の操作機能.....	556
20.5.5 スクリーン機能の利用.....	556
20.6 少し進んだ使い方.....	556
20.6.1 入出力機能の利用.....	556
20.6.1.1 スレッド間共有外部ファイル.....	557
20.6.1.2 ファクトリオブジェクト内に定義したファイル.....	559
20.6.1.3 オブジェクト内に定義したファイル.....	561
20.6.2 リモートデータベースアクセス(ODBC)の利用.....	563
20.6.2.1 コネクションをスレッド間で共有する.....	563
20.6.2.2 注意事項.....	566
20.6.3 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.....	566
20.6.3.1 概要.....	566
20.6.3.2 起動方法.....	566
20.6.3.3 パラメタの受渡し方法.....	566
20.6.3.4 復帰コード(関数値).....	567
20.6.3.5 翻訳とリンク.....	568
20.6.4 スレッド間で実行単位の資源を引き継ぐ方法.....	571
20.6.4.1 概要.....	571
20.6.4.2 利用方法.....	573
20.6.4.3 注意事項.....	573
20.6.4.4 翻訳とリンク.....	574
20.7 翻訳から実行までの方法.....	575
20.7.1 翻訳とリンク.....	575
20.7.1.1 COBOLプログラムだけでDLLを作成する場合.....	575
20.7.1.2 COBOLプログラムとCプログラムでDLLを作成する場合.....	576
20.7.2 実行.....	576
20.7.2.1 実行環境情報の設定.....	576
20.7.2.2 マルチスレッドモードでだけ有効な実行環境情報.....	579
20.8 マルチスレッドプログラムのデバッグ方法.....	579
20.8.1 TRACE機能.....	579
20.8.2 CHECK機能.....	581
20.8.3 COUNT機能.....	581
20.8.4 対話型デバッガ.....	582
20.8.5 NetCOBOL Studioのデバッグ機能.....	582
20.9 スレッド同期制御サブルーチン.....	582
20.10 注意事項.....	582
20.10.1 COM連携機能.....	582
20.10.1.1 COMのマルチスレッド環境.....	582
20.10.1.2 COBOLの初期化方法.....	582
20.10.1.3 COM連携で使用するオブジェクトの解放について.....	583
20.10.1.4 COMサーバの寿命について.....	583
20.10.2 オブジェクト指向プログラミング機能.....	583
20.10.3 簡易アプリ間通信機能.....	583
20.10.4 印刷機能.....	584
20.10.5 動的プログラム構造.....	584

第5部 テスト支援機能／デバッグ支援機能.....	585
第21章 テスト支援機能.....	586
21.1 NetCOBOL Studioのデバッグ機能.....	588
21.2 対話型デバッグ機能.....	588
21.3 CHECK機能.....	588
21.3.1 デバッグ作業の流れ.....	588
21.3.2 出力メッセージ.....	589
21.3.2.1 メッセージの内容.....	589
21.3.2.2 メッセージの出力回数.....	592
21.3.3 CHECK機能の使用例.....	592
21.3.4 注意事項.....	596
21.4 TRACE機能.....	597
21.4.1 デバッグ作業の流れ.....	597
21.4.2 トレース情報.....	598
21.4.3 注意事項.....	600
21.5 COUNT機能.....	601
21.5.1 デバッグ作業の流れ.....	601
21.5.2 COUNT情報.....	602
21.5.3 COUNT機能を使用したプログラムのデバッグ.....	605
21.5.4 注意事項.....	605
21.6 メモリチェック機能.....	606
21.6.1 デバッグ作業の流れ.....	606
21.6.2 出力メッセージ.....	606
21.6.2.1 メッセージの内容.....	607
21.6.3 診断レポート.....	607
21.6.4 プログラムの特定.....	608
21.6.5 破壊箇所の特定制.....	608
21.6.5.1 指定方法.....	608
21.6.6 注意事項.....	609
第22章 デバッグ支援機能.....	610
22.1 診断機能.....	610
22.1.1 診断機能の概要.....	610
22.1.2 診断機能が使用する資源.....	611
22.1.2.1 対象プログラム.....	611
22.1.2.2 プログラムの翻訳・リンク方法と出力情報との関係.....	611
22.1.2.3 プログラムの翻訳・リンク.....	613
22.1.2.4 デバッグ情報ファイルの格納先.....	614
22.1.3 診断機能の起動.....	614
22.1.3.1 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.....	614
22.1.3.2 応答しなくなったアプリケーションに対する起動.....	615
22.1.3.3 領域破壊が発生するアプリケーションに対する起動.....	615
22.1.3.4 起動パラメタ.....	615
22.1.4 診断レポート.....	617
22.1.4.1 診断レポートの出力先.....	617
22.1.4.2 診断レポートの出力情報.....	618
22.1.4.2.1 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.....	618
22.1.4.2.2 領域破壊監視の場合.....	624
22.1.4.3 診断レポートの見方.....	626
22.1.4.3.1 アプリケーションエラー.....	626
22.1.4.3.2 実行時メッセージ.....	628
22.1.4.3.3 応答なし.....	631
22.1.4.3.4 領域破壊監視.....	631
22.1.5 ダンプ.....	634
22.1.5.1 ダンプとは.....	634
22.1.5.2 ダンプの出力先.....	634
22.1.5.3 ダンプファイル数の管理.....	635

22.1.6 注意事項.....	635
22.2 翻訳リストとデバッグツールを使ったデバッグ.....	636
22.2.1 デバッグ作業の流れ.....	636
22.2.2 障害発生箇所の特定方法(プログラムデータベースファイル使用時).....	638
22.2.3 障害発生箇所の特定方法(リンクマップファイル使用時).....	643
22.2.4 データの参照方法.....	647
<b>第6部 プロジェクトマネージャを利用した開発.....</b>	<b>650</b>
<b>第23章 プロジェクト管理機能.....</b>	<b>651</b>
23.1 プロジェクト管理機能とは.....	651
23.2 プロジェクト管理に必要な資源.....	651
23.3 プロジェクト管理の手順.....	654
23.4 プロジェクトマネージャウィンドウ.....	655
23.4.1 プロジェクトファイルの作成.....	656
23.4.2 翻訳オプションの設定.....	657
23.4.3 リンクオプションの設定.....	657
23.4.3.1 [リンクオプション]ダイアログ.....	657
23.4.4 ファイルの登録.....	658
23.4.4.1 最終ターゲットファイルの登録.....	658
23.4.4.2 COBOLソースファイルの登録.....	659
23.4.4.3 主プログラムの指定.....	659
23.4.4.4 登録集ファイルの登録.....	660
23.4.4.5 各種定義体ファイルの登録.....	660
23.4.4.6 その他必要なファイルの登録.....	660
23.4.4.7 フォルダの作成.....	661
23.4.4.8 プロジェクトファイルの保存.....	661
23.4.5 プロジェクトのビルド・リビルド.....	662
23.4.5.1 デバッグモジュールの作成方法.....	662
23.4.6 翻訳エラーの修正.....	663
23.4.7 アプリケーションプログラムの実行.....	663
23.4.8 プリコンパイラの登録とプリコンパイルの実行.....	664
23.4.8.1 資産とツール.....	664
23.4.8.2 プロジェクト管理.....	665
23.4.8.3 プロジェクト管理の手順.....	665
23.4.8.4 [プリコンパイラ設定]ダイアログ.....	667
23.4.8.5 選択項目と対象プリコンパイラ.....	670
23.4.8.6 インクルードファイル.....	670
23.4.8.7 多段プリコンパイル.....	671
23.4.8.8 注意事項.....	671
23.4.9 実行可能プログラムのバージョン設定機能.....	671
23.4.10 リストビュー.....	672
23.5 モジュール定義ファイル.....	674
23.5.1 モジュール定義ファイルの生成方法.....	674
23.6 MAKEファイル.....	675
23.6.1 MAKEファイルの用途.....	675
23.6.2 プロジェクトマネージャのMAKEファイル.....	676
23.6.3 MAKEファイルとプロジェクトファイル.....	676
23.6.4 コマンドファイル.....	677
<b>第24章 対話型デバッガの使い方.....</b>	<b>678</b>
24.1 デバッガの概要.....	678
24.2 デバッグの準備.....	679
24.2.1 被デバッグプログラム.....	679
24.2.2 プログラムの翻訳.....	680
24.2.3 プログラムのリンク.....	680
24.3 デバッガの環境変数.....	680
24.4 デバッガの起動.....	680

24.4.1 起動方法.....	681
24.4.1.1 一般的なプログラムに対するデバグ起動方法.....	681
24.4.1.2 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グ起動方法.....	681
24.4.1.3 ジャストインタイムデバグ機能を使用したデバグ起動方法.....	681
24.4.2 デバグに必要な資源やパラメタの指定.....	681
24.4.2.1 [デバグを開始する]ダイアログを使ったデバグ指定.....	681
24.4.2.2 起動コマンドを使ったデバグ指定.....	684
24.5 デバグの操作.....	685
24.5.1 デバグの画面.....	685
24.5.1.1 メインウィンドウ.....	685
24.5.1.2 ソースファイルウィンドウ.....	687
24.5.1.3 プログラム一覧ウィンドウ.....	687
24.5.1.4 監視ウィンドウ.....	688
24.5.1.5 ラインコマンド入力ウィンドウ.....	688
24.5.1.6 呼出し経路ウィンドウ.....	689
24.5.1.7 スレッド一覧ウィンドウ.....	689
24.5.2 デバグの機能.....	689
24.5.2.1 デバグ機能の使い方.....	689
24.5.2.1.1 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.....	689
24.5.2.1.2 プログラムの実行順序を変更する.....	690
24.5.2.1.3 データの変更箇所を調べる.....	690
24.5.2.1.4 実行経路をたどる.....	691
24.5.2.1.5 プログラムの呼出し経路をたどる.....	691
24.5.2.1.6 デバグの状態を知る.....	692
24.5.2.2 デバグ操作の自動化.....	692
24.5.2.3 その他の補助的な機能.....	693
24.6 プログラムの特徴に合わせたデバグ方法.....	696
24.6.1 動的構造に対するデバグ方法.....	696
24.6.2 例外ハンドラで処理されるシグナルのデバグ方法.....	696
24.6.3 オブジェクト指向に対応したデバグ機能.....	697
24.6.4 マルチスレッドでのデバグの対応.....	698
24.6.4.1 デバグ機能のマルチスレッドでの基本的な動作.....	698
24.6.4.2 デバグ機能のマルチスレッド特有な動作.....	698
24.6.4.3 マルチスレッドのデバグを補助する機能.....	698
24.6.4.4 マルチスレッドでのデバグ操作の自動化.....	699
24.6.5 デバグのUnicode対応.....	699
24.6.5.1 Unicodeプログラムのデバグ方法.....	699
24.6.5.2 Unicodeを扱う上での注意事項.....	699
24.7 注意事項.....	700
24.8 デバグのコマンド一覧.....	702
24.9 対話型リモートデバグの使い方.....	703
24.9.1 リモートデバグの概要.....	703
24.9.1.1 リモートデバグ導入による効果.....	704
24.9.1.2 リモートデバグ時の資産の格納場所.....	704
24.9.1.3 リモートデバグ時の被デバグプログラムの作成方法.....	705
24.9.2 リモートデバグの開始手順.....	705
24.9.2.1 一般的なプログラムに対するリモートデバグ起動方法.....	705
24.9.2.2 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グ起動方法.....	706
24.9.2.3 ジャストインタイムデバグ機能を使用したリモートデバグ起動方法.....	707
24.9.3 リモートデバグコネクタの使い方.....	707
24.9.3.1 サーバ側のリモートデバグコネクタの使い方.....	708
24.9.3.2 サーバ側のリモートデバグコネクタの起動方法.....	708
24.9.3.3 接続制限ファイルの形式.....	709
24.9.3.4 クライアント側のリモートデバグコネクタの使い方.....	710
24.9.3.4.1 クライアント側のリモートデバグコネクタの起動方法.....	710
24.9.3.4.2 [リモートデバグコネクタ]ダイアログの設定方法.....	710
24.9.3.5 リモートデバグを許可するコンピュータを制限する.....	711

24.9.4 リモートデバッガの環境変数.....	713
24.9.4.1 @SVD_COMPATIBILITY (互換性の指定).....	713
24.9.5 リモートデバッガの起動.....	714
24.9.5.1 [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.....	714
24.9.5.2 起動コマンドを使ったリモートデバッグ指定.....	718
24.9.6 リモートデバッガの操作.....	720
24.9.7 リモートデバッガの注意事項.....	720
24.9.7.1 Win32/Win64リモートデバッガの注意事項.....	720
24.9.7.2 Solaris/Linuxリモートデバッガの注意事項.....	720
<b>第25章 ツールの操作.....</b>	<b>722</b>
25.1 WINCOBコマンド.....	723
25.1.1 WINCOBウィンドウ.....	723
25.1.2 WINCOBを使用した翻訳の手順.....	725
25.1.2.1 1つのソースファイルを翻訳するときの手順.....	725
25.1.2.2 複数のソースファイルを翻訳するときの手順.....	726
25.1.3 サンプルプログラムの翻訳.....	726
25.1.4 [翻訳オプション]ダイアログ.....	729
25.1.4.1 翻訳オプションの追加.....	729
25.1.4.2 翻訳オプションの変更.....	730
25.1.4.3 翻訳オプションの取消し.....	730
25.1.5 [登録集名の設定]ダイアログ.....	730
25.1.5.1 登録集名の追加.....	731
25.1.5.2 登録集名の変更.....	731
25.1.5.3 登録集名の取消し.....	731
25.1.5.4 登録集名の設定の終了.....	731
25.1.6 設定したオプションの格納.....	731
25.2 WINLINKコマンド.....	732
25.2.1 WINLINKウィンドウ.....	732
25.2.1.1 WINLINK[リンク]ウィンドウ.....	732
25.2.1.1.1 ターゲット名の指定.....	733
25.2.1.1.2 ファイルの指定.....	733
25.2.1.1.3 インポートライブラリの作成.....	733
25.2.1.1.4 リンクオプションの設定.....	733
25.2.1.1.5 モジュール定義ファイルの生成.....	734
25.2.1.1.6 リンクの開始と終了.....	735
25.2.1.2 WINLINK[ライブラリモジュール結合]ウィンドウ.....	735
25.2.1.2.1 ライブラリ名の指定.....	736
25.2.1.2.2 ファイルの指定.....	736
25.2.1.2.3 ライブラリ結合の開始と終了.....	736
25.2.2 WINLINKを使用したリンクの手順.....	737
25.2.2.1 WINLINKが使用するファイル.....	737
25.2.2.2 1つの目的プログラムをリンクするときの手順.....	739
25.2.2.3 DLLを作成するときの手順.....	739
25.2.2.4 単純構造の実行可能プログラムを作成するときの手順.....	740
25.2.2.5 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.....	740
25.2.2.6 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.....	741
25.2.2.7 ライブラリを作成するときの手順.....	741
25.2.3 サンプルプログラムのリンク.....	741
25.3 WINEXECコマンド.....	742
25.3.1 WINEXECウィンドウ.....	742
25.3.1.1 ファイル名の指定.....	743
25.3.1.2 実行の開始と終了.....	743
25.3.2 WINEXECを使用した実行の手順.....	743
25.3.3 サンプルプログラムの実行.....	744
<b>第26章 Webアプリケーション開発支援機能の利用.....</b>	<b>747</b>
26.1 Webアプリケーション開発支援機能の概要.....	747

26.2 Webアプリケーション開発支援機能を使用した作業の流れ.....	749
26.3 Webアプリケーションウィザード.....	749
26.3.1 Webアプリケーション.....	750
26.3.2 Webアプリケーションウィザードの使用法.....	752
26.3.2.1 Webアプリケーションウィザードの起動.....	752
26.3.2.2 Webアプリケーションウィザードの画面と操作.....	753
26.3.3 Webアプリケーション雛形の構成.....	755
26.3.3.1 生成するプログラム資源.....	755
26.3.3.2 Webアプリケーション雛形の違い.....	755
26.3.3.3 生成するその他の資源.....	757
26.3.3.4 その他のWebアプリケーションウィザードの使用法.....	758
26.4 Webパラメタ取得プログラム生成.....	759
26.4.1 Webパラメタ取得プログラム.....	760
26.4.2 Webパラメタ取得プログラム生成ツールの使用法.....	762
26.4.3 Webパラメタ取得プログラムの生成規則.....	763
26.4.3.1 呼出し用ページ.....	763
26.4.3.2 データ入力用登録集.....	764
26.4.3.3 生成するプログラム.....	766
26.5 結果ページ出力プログラム生成.....	766
26.5.1 結果ページ出力プログラム.....	767
26.5.2 結果ページ出力プログラム生成ツールの使用法.....	769
26.5.3 結果ページ出力プログラムの生成規則.....	771
26.5.3.1 結果出力用ページ.....	771
26.5.3.2 データ出力用登録集.....	771
26.5.3.3 生成する出力プログラム.....	772
26.6 Webアプリケーション開発のヒント.....	773
26.6.1 プロジェクトマネージャからのHTML文書の操作.....	773
26.6.2 エディタテンプレートの活用.....	773
<b>第27章 オブジェクト指向プログラミングの開発.....</b>	<b>775</b>
27.1 オブジェクト指向プログラミングで使用する資源.....	775
27.2 開発手順.....	776
27.3 プロジェクト管理.....	776
27.3.1 プロジェクト管理のメリット.....	776
27.3.2 プログラム構造.....	777
27.3.3 翻訳処理.....	777
27.3.4 リンク処理.....	779
27.3.5 相互参照クラス.....	780
27.3.5.1 相互参照クラスのプロジェクト管理.....	780
27.3.5.2 相互参照クラスの翻訳処理.....	780
27.3.5.3 相互参照クラスのリンク処理.....	781
27.4 クラスの公開.....	782
27.5 クラスブラウザ.....	784
27.5.1 概要.....	784
27.5.2 使用する資源.....	784
27.5.3 クラス情報データベースの生成.....	785
27.5.4 オプション設定ウィンドウ.....	785
27.5.5 データベース生成.....	786
27.5.6 クラスブラウザの表示/操作手順.....	786
27.5.7 クラスブラウザウィンドウ.....	786
27.5.7.1 表示可能情報および表示方法.....	786
27.5.7.2 継承関係の表示.....	787
27.5.7.3 メソッド一覧の表示.....	787
27.5.7.4 インタフェースの表示.....	787
27.5.7.5 テンプレート展開機能.....	787
27.5.7.6 リポジトリ段落のクラス指定子.....	787
27.5.7.7 オブジェクトプロパティのテンプレート展開.....	788

27.5.7.8 INVOKE文のテンプレート展開.....	788
27.5.8 クラスの検索条件の入力.....	788
27.6 プロジェクトブラウザ.....	789
27.6.1 概要.....	789
27.6.2 使用する資源.....	789
27.6.3 プロジェクト情報データベース生成.....	791
27.6.4 プロジェクトブラウザの表示/操作手順.....	791
27.6.5 プロジェクトブラウザウィンドウ.....	791
27.6.5.1 表示可能情報および表示方法.....	791
27.6.5.2 継承の表示.....	792
27.6.5.3 クラス参照の表示.....	792
27.6.5.4 呼出しプログラムの表示.....	792
27.6.5.5 メソッド一覧の表示.....	792
27.6.5.6 二次入口点一覧の表示.....	792
27.6.5.7 インタフェースの表示.....	793
27.6.5.8 COBOLソースファイル名表示.....	793
27.6.5.9 エディタ連携機能.....	793
27.6.5.10 テンプレート展開機能.....	793
27.6.5.11 リポジトリ段落のクラス指定子.....	794
27.6.5.12 オブジェクトプロパティのテンプレート展開.....	794
27.6.5.13 CALL文の自動生成.....	794
27.6.5.14 INVOKE文のテンプレート展開.....	794
<b>第28章 COM機能.....</b>	<b>796</b>
28.1 COM機能を利用したアプリケーション.....	796
28.1.1 一般的なシステム形態.....	796
28.1.2 COMを利用したアプリケーション形態.....	796
28.1.3 COM機能を利用するアプリケーションの作成.....	798
28.1.4 開発環境と資源.....	798
28.2 COMクライアント機能.....	800
28.2.1 概要.....	800
28.2.2 アーリバインドとレイトバインド.....	801
28.2.2.1 アーリバインドとレイトバインドの違い.....	801
28.2.2.2 アーリバインドのメリットとデメリット.....	802
28.2.2.3 バインド方法選択の基準.....	803
28.2.2.4 .NET言語で作成されたCOMサーバとの連携.....	803
28.2.3 *COMクラス.....	803
28.2.3.1 プログラミング例.....	804
28.2.4 *COMレイトバインドクラス.....	807
28.2.4.1 *COMレイトバインドクラスのファクトリメソッド.....	807
28.2.4.2 *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.....	808
28.2.5 *COMアーリバインドクラス.....	808
28.2.5.1 *COMアーリバインドクラスのファクトリメソッド.....	808
28.2.5.2 *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.....	809
28.2.6 名前付きパラメタ.....	809
28.2.7 パラメタの省略.....	809
28.2.8 COM機能の例外処理.....	810
28.2.8.1 例外の種類.....	810
28.2.8.2 例外発生時に通知される情報.....	810
28.2.8.3 例外処理の流れ.....	811
28.2.8.4 *COM-EXCEPTIONクラスの使用例.....	812
28.2.9 オブジェクト指定子.....	813
28.3 COMサーバ機能.....	814
28.3.1 COMサーバとは.....	815
28.3.2 COBOLで作成するCOMサーバの特徴.....	815
28.3.3 COMサーバの作成.....	815
28.3.4 COMサーバの登録と削除.....	816

28.3.5 COMサーバの利用	817
28.3.6 例外の通知	817
28.3.7 既存プログラムのCOMサーバ化	819
28.3.8 注意事項	821
28.4 DCOMとMTS	821
28.4.1 MTSの概要	821
28.4.2 MTS環境への登録方法	822
28.4.3 クライアントマシンへのインストール	824
28.4.4 MTSのトランザクション管理	825
28.4.4.1 COMサーバからのトランザクション制御	825
28.4.4.2 オブジェクトコンテキストオブジェクトのメソッド	826
28.4.4.3 COMクライアントからのトランザクション制御	828
28.4.4.4 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	829
28.4.4.5 注意事項	830
28.4.5 MTSのトランザクション処理	831
28.4.5.1 トランザクションの基本的な考え方	831
28.4.5.2 トランザクションを制御するための追加処理	831
28.4.5.3 MTSを使ったトランザクションのプログラム例	832
28.4.5.4 トランザクション範囲	836
28.4.5.5 MTS配下での動作上の注意	839
28.5 COM連携で扱うデータ型の対応	840
28.6 COM連携のためのその他の特殊クラス	842
28.6.1 *COM-ARRAYクラス	842
28.6.1.1 *COM-ARRAYクラスのファクトリメソッド	842
28.6.1.2 *COM-ARRAYクラスのオブジェクトメソッド	843
28.6.1.3 *COM-ARRAYの型指定子	843
28.6.1.4 *COM-ARRAYクラスの使用例	844
28.6.1.5 バリエーション型データの扱い	846
28.6.1.6 *COM-ARRAYクラスで扱うデータ型	847
28.6.2 *COM-EXCEPTIONクラス	847
28.6.2.1 *COM-EXCEPTIONクラスのメソッド	848
28.6.3 特殊クラスの注意事項	850
28.7 COMサーバおよびCOMクライアントのデバッグ	851
28.7.1 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	851
28.7.2 COMクライアントのデバッグ方法	852
28.7.3 COMサーバのデバッグ方法	852
28.7.3.1 デバッガを起動する	853
28.7.3.2 デバッグを行う	853
28.7.4 COMクライアントとCOMサーバを通じたデバッグ方法	853
28.8 開発環境	854
28.8.1 COMサーバ名の設定	854
28.8.2 COMサーバ作成機能の使い方	855
28.9 COM関連の用語の説明	857
付録A 翻訳オプション	859
A.1 翻訳オプションの指定方法と優先順位	859
A.2 翻訳オプションの指定形式	860
A.3 翻訳オプション一覧	861
A.3.1 AIMLIB (サブスキーマ定義ファイルのフォルダの指定)	863
A.3.2 ALPHAL (英小文字の扱い)	864
A.3.3 ASCOMP5 (2進項目の解釈の指定)	864
A.3.4 BINARY (2進項目の扱い)	865
A.3.5 CHECK (CHECK機能の使用の可否)	866
A.3.6 CONF (規格の違いによるメッセージの出力の可否)	867
A.3.7 CONVCHAR (コンパイラが使用するコード変換ライブラリ)	868
A.3.8 COPY (登録集原文の表示)	869
A.3.9 COUNT (COUNT機能の使用の可否)	869

A.3.10 CREATE(創成ファイルの指定)	869
A.3.11 CURRENCY(通貨編集用文字の扱い)	870
A.3.12 DLOAD(プログラム構造の指定)	870
A.3.13 DNTB(DISPLAY-OF関数およびNATIONAL-OF関数における後置空白の扱い)	870
A.3.14 DUPCHAR(重複文字の扱い)	871
A.3.15 ENCODE(データ項目のエンコードの指定)	871
A.3.16 EQUALS(SORT文での同一キーデータの処理方法)	873
A.3.17 FILELIT(ファイル識別名定数の扱い)	873
A.3.18 FLAG(診断メッセージのレベル)	873
A.3.19 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)	874
A.3.20 FORMEXT(画面帳票定義体ファイルの拡張子の指定)	875
A.3.21 FORMLIB(画面帳票定義体ファイルのフォルダの指定)	875
A.3.22 GEN(FCOMおよびUWAの表示)	876
A.3.23 INITVALUE(作業場所節でのVALUE句なし項目の扱い)	876
A.3.24 LANGLVL(ANSI COBOL規格の指定)	876
A.3.25 LIB(登録集ファイルのフォルダの指定)	877
A.3.26 LIBEXT(登録集ファイルの拡張子の指定)	877
A.3.27 LINECOUNT(翻訳リストの1ページあたりの行数)	878
A.3.28 LINESIZE(翻訳リストの1行あたりの文字数)	878
A.3.29 LIST(目的プログラムリストの出力の可否)	879
A.3.30 MAIN(主プログラム/副プログラムの指定)	879
A.3.31 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)	880
A.3.32 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)	880
A.3.33 MF(Micro Focus互換モードの指定)	880
A.3.34 MODE(ACCEPT文の動作の指定)	881
A.3.35 NAME(翻訳単位ごとのオブジェクトファイルの出力の可否)	881
A.3.36 NCW(日本語利用者語の文字集合の指定)	882
A.3.37 NSP(日本語項目における空白の扱い)	882
A.3.38 NSPCOMP(日本語空白の比較方法の指定)	883
A.3.39 NUMBER(ソースプログラムの一連番号領域の指定)	884
A.3.40 OBJECT(目的プログラムの出力の可否)	884
A.3.41 OPTIMIZE(広域最適化の扱い)	885
A.3.42 PRINT(各種翻訳リストの出力の可否および出力先の指定)	885
A.3.43 QUOTE/APOST(表意定数QUOTEの扱い)	885
A.3.44 RCS(実行時コード系の指定)	886
A.3.45 REP(リポジットファイルの入出力先フォルダの指定)	887
A.3.46 REPIN(リポジットファイルの入力先フォルダの指定)	887
A.3.47 RSV(予約語の種類)	888
A.3.48 SAI(ソース解析情報ファイルの出力の可否)	889
A.3.49 SCS(ソースファイルのコード系)	889
A.3.50 SDS(符号付き10進項目の符号の整形の可否)	890
A.3.51 SHREXT(マルチスレッドプログラムの外部属性に関する扱い)	890
A.3.52 SMSIZE(PowerSORTが使用するメモリ容量を指定)	890
A.3.53 SOURCE(ソースプログラムリストの出力の可否)	891
A.3.54 SQLGRP(SQLのホスト変数定義の拡張)	891
A.3.55 SRF(正書法の種類)	892
A.3.56 SSIN(ACCEPT文のデータの入力先)	892
A.3.57 SSOUT(DISPLAY文のデータの出力先)	893
A.3.58 STD1(英数字の文字の大小順序の指定)	893
A.3.59 TAB(タブの扱い)	894
A.3.60 TEST(デバッグ機能の使用の可否)	894
A.3.61 THREAD(マルチスレッドプログラム作成の指定)	895
A.3.62 TRACE(TRACE機能の使用の可否)	895
A.3.63 TRUNC(桁落とし処理の可否)	896
A.3.64 XREF(相互参照リストの出力の可否)	897
A.3.65 ZWB(符号付き外部10進項目と英数字項目の比較)	897
A.4 プログラム定義にだけ指定可能な翻訳オプション	898

A.5 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	898
A.6 V10以降より追加された翻訳オプション	899
<b>付録B 翻訳リスト</b>	<b>900</b>
B.1 診断メッセージリスト	900
B.2 オプション情報リスト、翻訳単位統計情報リスト	900
B.3 相互参照リスト	901
B.4 ソースプログラムリスト	902
B.5 目的プログラムリスト	903
B.6 データエリアに関するリスト	905
B.6.1 データマップリスト	905
B.6.2 プログラム制御情報リスト	909
B.6.3 セクションサイズリスト	912
<b>付録C 環境変数情報</b>	<b>913</b>
C.1 環境変数情報の優先順位	913
C.2 環境変数情報一覧	913
C.2.1 @AllFileExclusive (ファイルの排他処理の指定)	916
C.2.2 @CBR_ATTACH_TOOL (プログラムからデバッグ機能を起動する指定)	916
C.2.3 @CBR_CBRFILE (実行用の初期化ファイルの指定)	919
C.2.4 @CBR_CBRINFO (簡略化した動作状態を出力する指定)	919
C.2.5 @CBR_CIINF (論理宛先定義ファイルの指定)	919
C.2.6 @CBR_ClassInfFile (クラス情報ファイルの指定)	919
C.2.7 @CBR_CODE_SET (ファイルのコード系の指定)	920
C.2.8 @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)	921
C.2.9 @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)	921
C.2.10 @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)	921
C.2.11 @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)	922
C.2.12 @CBR_CONSOLE (コンソールウィンドウの種別の指定)	922
C.2.13 @CBR_CONVERT_CHARACTER (コード変換ライブラリの指定)	922
C.2.14 @CBR_CSV_OVERFLOW_MESSAGE (CSV形式データ操作時のメッセージ抑止指定)	923
C.2.15 @CBR_CSV_TYPE (生成するCSV形式のバリエーション)	923
C.2.16 @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)	924
C.2.17 @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)	924
C.2.18 @CBR_DISPLAY_CONSOLE_OUTPUT (DISPLAY UPON CONSOLEのイベントログ出力指定)	925
C.2.19 @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)	925
C.2.20 @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)	926
C.2.21 @CBR_DISPLAY_SYSERR_OUTPUT (DISPLAY UPON SYSERRのイベントログ出力指定)	926
C.2.22 @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)	926
C.2.23 @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)	927
C.2.24 @CBR_DISPLAY_SYSOUT_OUTPUT (DISPLAY UPON SYSOUTのイベントログ出力指定)	928
C.2.25 @CBR_DocumentName_xxxx (I制御レコードによる文書名の指定)	928
C.2.26 @CBR_ENTRYFILE (エントリ情報ファイルの指定)	928
C.2.27 @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)	929
C.2.28 @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)	929
C.2.29 @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)	929
C.2.30 @CBR_FILE_LFS_ACCESS (COBOLファイルのサイズを拡張する指定)	930
C.2.31 @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)	930
C.2.32 @CBR_FILE_USE_MESSAGE (入出力エラーの実行時メッセージの出力)	931

C.2.33 @CBR_FUNCTION_NATIONAL (NATIONAL関数の変換モードの指定)	931
C.2.34 @CBR_InstanceBlock (オブジェクトインスタンスの獲得方法の指定)	932
C.2.35 @CBR_JOBDATE (任意の日付を取得)	932
C.2.36 @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)	932
C.2.37 @CBR_MESSAGE (実行時メッセージの出力先の指定)	934
C.2.38 @CBR_MESS_LEVEL_CONSOLE (実行時メッセージの重大度指定)	935
C.2.39 @CBR_MESS_LEVEL_EVENTLOG (実行時メッセージの重大度指定)	936
C.2.40 @CBR_MEMORY_CHECK (メモリチェック機能を使って検査を行う指定)	937
C.2.41 @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)	937
C.2.42 @CBR_PrinterANK_Size (ANK文字サイズの指定)	938
C.2.43 @CBR_PrintFontTable (印刷ファイルで使用するフォントテーブルの指定)	939
C.2.44 @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)	939
C.2.45 @CBR_PrintTextPosition (文字配置座標の計算方法の指定)	939
C.2.46 @CBR_SCR_KEYDEFFILE (スクリーン操作のキー定義ファイルの指定)	940
C.2.47 @CBR_SCREEN_POSITION (スクリーン画面の表示位置の指定)	940
C.2.48 @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)	941
C.2.49 @CBR_SYMFOWARE_THREAD (Symfoware連携でマルチスレッド動作可能にする指定)	941
C.2.50 @CBR_SYSERR_EXTEND (SYSERR出力情報の拡張の指定)	941
C.2.51 @CBR_TextAlign (文字行内配置時の上端/下端合わせの指定)	941
C.2.52 @CBR_THREAD_MODE (スレッドモードの指定)	941
C.2.53 @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)	942
C.2.54 @CBR_TRACE_FILE (トレース情報の出力ファイルの指定)	942
C.2.55 @CBR_TRACE_PROCESS_MODE (TRACEファイルのプロセス毎の出力指定)	942
C.2.56 @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)	942
C.2.57 @CnslBufLine (コンソールウィンドウのバッファ数の指定)	943
C.2.58 @CnslFont (コンソールウィンドウのフォントの指定)	943
C.2.59 @CnslWinSize (コンソールウィンドウの大きさの指定)	943
C.2.60 @DefaultFCB_Name (デフォルトFCB名の指定)	944
C.2.61 @ExitSessionMSG (COBOLアプリケーション起動中のWindows終了指定)	944
C.2.62 @GOPT (実行時オプションの指定)	945
C.2.63 @IconDLL (アイコンリソースのDLL名の指定)	945
C.2.64 @IconName (アイコンリソースの識別名の指定)	946
C.2.65 @MessOutFile (メッセージを出力するファイルの指定)	946
C.2.66 @MGPRM (OSIV系形式の実行時パラメタの指定)	946
C.2.67 @NoMessage (実行時メッセージおよびSYSERRの出力抑止指定)	947
C.2.68 @ODBC_Inf (ODBC情報ファイルの指定)	947
C.2.69 @PrinterFontName (印刷ファイルで使用するフォントの指定)	948
C.2.70 @PRN_FormName_xxx (用紙名の指定)	948
C.2.71 @ScrnFont (スクリーン操作で使用するフォントの指定)	949
C.2.72 @ScrnSize (スクリーン操作の論理画面の大きさの指定)	949
C.2.73 @ShowIcon (NetCOBOLのアイコン表示の抑止指定)	949
C.2.74 @SVD_COMPATIBILITY (デバッグの互換性の指定)	950
C.2.75 @SVD_ENV (デバッグ環境の指定)	950
C.2.76 @SVD_PERFORM_CHECK (デバッグでPERFORMレベルの監視を行う指定)	952
C.2.77 @WinCloseMsg (ウィンドウを閉じるときのメッセージ表示の指定)	952
C.2.78 ファイル識別名 (プログラムで使用するファイルの指定)	952
C.2.79 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の指定)	953
C.2.80 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)	954
C.2.81 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)	955
C.2.82 FCBxxxx (FCB制御文の指定)	957
C.2.83 FOVLDIR (フォームオーバーレイパターンフォルダの指定)	957
C.2.84 FOVLTYPE (フォームオーバーレイパターンファイル名の形式の指定)	958
C.2.85 FOVLNAME (フォームオーバーレイパターンファイル名の指定)	958
C.2.86 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)	958
C.2.87 SYSCOUNT (COUNT情報の出力ファイルの指定)	958
C.2.88 SYSINのアクセス名 (小入出力機能の入力ファイルの指定)	959

C.2.89 SYSOUTのアクセス名(小入出力機能の出力ファイルの指定)	959
付録D 入出力状態一覧	961
付録E 定量制限	964
E.1 正書法	964
E.2 中核のデータ部	964
E.3 中核の手続き部	964
E.4 順ファイル	965
E.5 印刷ファイル	965
E.6 行順ファイル	966
E.7 相対ファイル	966
E.8 索引ファイル	966
E.9 表示ファイル	967
E.10 プログラム間連絡	967
E.11 整列併合	968
E.12 原始文操作	968
E.13 オブジェクト指向プログラミング	968
E.14 ODBC情報ファイル	968
E.15 埋込みSQL文	968
E.16 実行時パラメタ	969
付録F 広域最適化	970
F.1 最適化の項目	970
F.2 共通式の除去	970
F.3 不変式の移動	971
F.4 誘導変数の最適化	971
F.5 PERFORM文の最適化	971
F.6 隣接転記の統合	972
F.7 無駄な代入の除去	972
F.8 広域最適化での注意事項	972
付録G 特殊な定数および利用者語の書き方	974
G.1 プログラム名定数	974
G.2 原文名定数	974
G.3 ファイル識別名定数	974
G.4 外部名を指定するための定数	975
G.5 外部名を指定するための利用者語	975
G.6 外部名の規則	976
付録H 関数	977
H.1 組込み関数	977
H.1.1 組込み関数一覧	977
H.1.2 組込み関数の型と記述の関係	978
H.1.3 引数の型によって決定される関数の型	980
H.1.4 組込み関数の便利な使い方	980
H.2 索引ファイル復旧関数	983
H.2.1 索引ファイル復旧関数 (CFURCOV)	983
H.2.2 索引ファイル簡易復旧関数 (CFURCOVS)	984
H.2.3 注意事項	985
H.2.4 COBOLから呼び出す場合の使用例	985
H.2.5 メッセージの内容とコード	987
H.3 COBOLファイルユーティリティ関数	988
H.3.1 ファイル変換関数 (COB_FILE_CONVERT)	989
H.3.2 ファイルロード関数 (COB_FILE_LOAD)	992
H.3.3 ファイルアンロード関数 (COB_FILE_UNLOAD)	995
H.3.4 ファイル整列関数 (COB_FILE_SORT)	996
H.3.5 ファイル再編成関数 (COB_FILE_REORGANIZE)	999

H.3.6 ファイルコピー関数 (COB_FILE_COPY).....	999
H.3.7 ファイル移動関数 (COB_FILE_MOVE).....	1000
H.3.8 ファイル削除関数 (COB_FILE_DELETE).....	1001
H.4 COBOLファイルアクセスルーチン.....	1001
H.4.1 COBOLファイルアクセスルーチンとは.....	1001
H.4.2 API関数一覧.....	1002
H.4.3 API関数で使用する構造体.....	1003
H.4.4 準備するもの.....	1003
H.4.5 環境設定.....	1003
H.4.6 使い方.....	1003
H.4.7 ファイルのオープン (cobfa_open()).....	1004
H.4.8 ファイルのオープン (cobfa_openW()).....	1009
H.4.9 ファイルのクローズ (cobfa_close()).....	1009
H.4.10 レコードの読み込み (cobfa_rdkey()).....	1010
H.4.11 レコードの読み込み (cobfa_rdnex()).....	1013
H.4.12 レコードの読み込み (cobfa_rdnex()).....	1014
H.4.13 レコードの書き出し (cobfa_wrkey()).....	1016
H.4.14 レコードの書き出し (cobfa_wrnex()).....	1018
H.4.15 レコードの書き出し (cobfa_wrnex()).....	1019
H.4.16 レコードの削除 (cobfa_delcurr()).....	1020
H.4.17 レコードの削除 (cobfa_delkey()).....	1022
H.4.18 レコードの削除 (cobfa_delrec()).....	1023
H.4.19 レコードの書き換え (cobfa_rewcurr()).....	1024
H.4.20 レコードの書き換え (cobfa_rewkey()).....	1025
H.4.21 レコードの書き換え (cobfa_rewrec()).....	1027
H.4.22 レコードの位置決め (cobfa_stkey()).....	1028
H.4.23 レコードの位置決め (cobfa_strec()).....	1030
H.4.24 レコードロックの解除 (cobfa_release()).....	1032
H.4.25 ファイル情報の取得 (cobfa_indexinfo()).....	1033
H.4.26 エラー番号の取得 (cobfa_erno()).....	1034
H.4.27 入出力状態の取得 (cobfa_stat()).....	1035
H.4.28 読み込みレコード長の取得 (cobfa_reclen()).....	1035
H.4.29 相対レコード番号の取得 (cobfa_recnun()).....	1036
H.4.30 ファイルアクセスの排他ロック (LOCK_cobfa()).....	1036
H.4.31 ファイルアクセスの排他ロック解除 (UNLOCK_cobfa()).....	1038
H.4.32 fa_keydesc構造体.....	1038
H.4.33 fa_keylist構造体.....	1041
H.4.34 fa_dictinfo構造体.....	1043
H.4.35 ファイルの機能.....	1044
H.4.36 エラー番号.....	1045
H.4.37 入出力状態.....	1047
H.4.38 COBOLファイルアクセスルーチンの制限事項.....	1047
H.4.39 COBOLファイルアクセスルーチンの留意事項.....	1047
付録I サブルーチン.....	1050
I.1 サブルーチン一覧.....	1050
I.2 COBOLプログラムから呼び出すサブブルーチン.....	1052
I.2.1 COB_ALLOC_MEMORY (メモリ割当てサブブルーチン).....	1052
I.2.2 COB_DEADLOCK_EXIT (デッドロック出口スケジューラサブブルーチン).....	1053
I.2.3 COB_EXIT_PROCESS (プロセス終了サブブルーチン).....	1054
I.2.4 COB_FREE_MEMORY (メモリ解放サブブルーチン).....	1056
I.2.5 COB_GET_PROCESSID (プロセスID取得サブブルーチン).....	1057
I.2.6 COB_GET_THREADID (スレッドID取得サブブルーチン).....	1058
I.2.7 COB_LOCK_DATA (データロック獲得サブブルーチン).....	1058
I.2.8 COB_LOCK_OBJECT (オブジェクトロック獲得サブブルーチン).....	1059
I.2.9 COB_REPORT_EVENT (イベントログ出力サブブルーチン).....	1060
I.2.10 COB_UNLOCK_DATA (データロック解放サブブルーチン).....	1061

I.2.11 COB_UNLOCK_OBJECT(オブジェクトロック解放サブルーチン).....	1062
I.2.12 COBCL_CLOSE(サーバとの接続の切断).....	1062
I.2.13 COBCL_OPEN(サーバとの接続の確立).....	1063
I.2.14 COBCL_READ(サーバからのメッセージの読み込み).....	1064
I.2.15 COBCL_WRITE(サーバへのメッセージの書き込み).....	1066
I.2.16 JMPBGWDH(ウィンドウハンドル獲得サブルーチン).....	1068
I.2.17 JMPBWINS(インスタンスハンドル獲得サブルーチン).....	1069
I.2.18 #NATLETOBE(ビッグエンディアン変換サブルーチン).....	1070
I.2.19 #NATBETOLE(リトルエンディアン変換サブルーチン).....	1070
I.3 他言語連携で使用するサブルーチン.....	1071
I.3.1 COB_GETRUNIT(COBOL実行単位ハンドル取得サブルーチン).....	1071
I.3.2 COB_SETRUNIT(COBOL実行単位ハンドル設定サブルーチン).....	1071
I.3.3 JMPCINT2(実行単位の開始サブルーチン).....	1072
I.3.4 JMPCINT3(実行単位の終了サブルーチン).....	1073
I.3.5 JMPCINT4(実行環境の閉鎖サブルーチン).....	1074
I.4 COBOL Webサブルーチン.....	1075
I.4.1 COBW3_ALTER_SESSION_TIMEOUT(セッションのタイムアウト時間の変更).....	1075
I.4.2 COBW3_CHECK_VALUE_X, COBW3_CHECK_VALUE_N(Webパラメタ中のVALUEの検索).....	1076
I.4.3 COBW3_CNV_DEL(登録した変換データの削除).....	1076
I.4.4 COBW3_CNV_INIT(登録した変換データをすべて初期化).....	1077
I.4.5 COBW3_CNV_SET(結果出力用ページ内に指定された変換データを登録).....	1078
I.4.6 COBW3_DEL_CNV_X, COBW3_DEL_CNV_N(結果出力用ページ内の変換データの削除).....	1079
I.4.7 COBW3_DEL_COOKIE(登録したCookieデータの削除).....	1080
I.4.8 COBW3_DEL_COOKIE_X, COBW3_DEL_COOKIE_N(Cookieデータの削除).....	1080
I.4.9 COBW3_DEL_REPEAT_X, COBW3_DEL_REPEAT_N(結果出力用ページ内の繰返し変換データの削除).....	1081
I.4.10 COBW3_DEL_UPLOADEDFILE(アップロード済みファイルの削除).....	1082
I.4.11 COBW3_END_SESSION(セッションの終了).....	1082
I.4.12 COBW3_FREE(Web実行環境の資源の解放).....	1083
I.4.13 COBW3_INIT(作業環境の設定およびWebパラメタの獲得).....	1083
I.4.14 COBW3_INIT_CNV(結果出力用ページ内の変換データの初期化).....	1084
I.4.15 COBW3_INIT_COOKIE(登録済みの全Cookieデータの初期化).....	1084
I.4.16 COBW3_INIT_REPEAT(結果出力用ページ内の繰返し変換データの初期化).....	1084
I.4.17 COBW3_GEN_UPLOADFILE_X, COBW3_GEN_UPLOADFILE_N(アップロードされたファイルをサーバ上に生成).....	1085
I.4.18 COBW3_GET_AUTHORIZE(認証情報の取得).....	1086
I.4.19 COBW3_GET_COOKIE, COBW3_GET_COOKIE_N(リクエストCookieデータから任意の名前の検索).....	1086
I.4.20 COBW3_GET_COOKIE_XX, COBW3_GET_COOKIE_NX, COBW3_GET_COOKIE_XN, COBW3_GET_COOKIE_NN(WebパラメタからCookieデータの取得).....	1088
I.4.21 COBW3_GET_REQUEST_INFO(リクエストに関する情報の取得).....	1089
I.4.22 COBW3_GET_SESSION_DATA(セッションデータの取得).....	1090
I.4.23 COBW3_GET_SESSION_INFO(セッション情報の取得).....	1091
I.4.24 COBW3_GET_UPLOADFILE_INFO_X, COBW3_GET_UPLOADFILE_INFO_N(アップロードファイル情報の取得).....	1092
I.4.25 COBW3_GET_VALUE_XX, COBW3_GET_VALUE_NX, COBW3_GET_VALUE_XN, COBW3_GET_VALUE_NN (Webパラメタ中のNAMEの検索およびVALUEの取得).....	1094
I.4.26 COBW3_NAME, COBW3_NAMEN(Webパラメタから任意の名前を検索).....	1095
I.4.27 COBW3_PUT_HEAD(ヘッダの出力).....	1096
I.4.28 COBW3_PUT_HTML(結果出力用ページの出力).....	1097
I.4.29 COBW3_PUT_TEXT(データの出力(改行コードを付加)).....	1100
I.4.30 COBW3_RECEIVE_HEADER(HTTPヘッダの取得).....	1101
I.4.31 COBW3_SET_CNV_XX, COBW3_SET_CNV_NX, COBW3_SET_CNV_XN, COBW3_SET_CNV_NN(結果出力用ペ ージ内の変換データの登録).....	1102
I.4.32 COBW3_SET_COOKIE(ブラウザへ送信するCookieデータの登録).....	1104
I.4.33 COBW3_SET_COOKIE_XX, COBW3_SET_COOKIE_NX, COBW3_SET_COOKIE_XN, COBW3_SET_COOKIE_NN (Cookieデータの登録).....	1106
I.4.34 COBW3_SET_REPEAT_XX, COBW3_SET_REPEAT_NX, COBW3_SET_REPEAT_XN, COBW3_SET_REPEAT_NN (結果出力用ページ内の繰返し変換データの登録).....	1109
I.4.35 COBW3_SET_SESSION_DATA(セッションデータの登録).....	1111
I.4.36 COBW3_START_SESSION(セッションの開始).....	1112

I.4.37 COBW3_SYSTEM(システムコマンドの実行)	1113
I.4.38 COBW3_VALUE(Webパラメタから任意の値を検索)	1113
I.5 メモリ関連サブルーチンの使い方	1114
I.5.1 シングルスレッドモードのプログラム	1114
I.5.2 マルチスレッドモードのプログラム	1116
I.6 スレッド同期制御サブルーチンの使い方	1119
I.6.1 データロックサブルーチン	1119
I.6.2 オブジェクトロックサブルーチン	1119
I.6.3 スレッド同期制御サブルーチンのエラーコード	1119
I.7 簡易アプリ間通信のサブルーチンの使い方	1120
I.7.1 簡易アプリ間通信のサブルーチンの注意事項	1122
I.7.2 簡易アプリ間通信のサブルーチンのエラーコード	1122
I.8 COBOL Webサブルーチンの使い方	1125
I.8.1 COBOL Webサブルーチンの機能	1125
I.8.2 Webサブルーチンのインタフェース	1130
I.8.3 エラー処理	1132
付録J コマンド	1139
J.1 COBOL32コマンド	1139
J.1.1 COBOL32コマンドの復帰コード	1140
J.1.2 翻訳コマンドのオプション一覧	1140
J.1.3 -A(サブスキーマ定義ファイルのフォルダの指定)	1141
J.1.4 -Dc(COUNT機能を使用する指定)	1141
J.1.5 -dd(デバッグ情報ファイルのフォルダの指定)	1142
J.1.6 -Dk(CHECK機能を使用する指定)	1142
J.1.7 -do(オブジェクトファイルのフォルダの指定)	1143
J.1.8 -dp(翻訳リストファイルのフォルダの指定)	1143
J.1.9 -Dr(TRACE機能を使用する指定)	1144
J.1.10 -dr(リポジトリファイルの入出力先フォルダの指定)	1144
J.1.11 -ds(ソース解析情報ファイルのフォルダの指定)	1144
J.1.12 -Dt(デバッグ機能および診断機能を使用する指定)	1145
J.1.13 -I(登録集ファイルのフォルダの指定)	1145
J.1.14 -i(オプションファイルの指定)	1146
J.1.15 -M(主プログラムを翻訳するときの指定)	1146
J.1.16 -m(画面帳票定義体ファイルのフォルダの指定)	1146
J.1.17 -O(広域最適化を適用する指定)	1146
J.1.18 -P(各種翻訳リストの出力および出力先の指定)	1147
J.1.19 -R(リポジトリファイルの入力先フォルダの指定)	1147
J.1.20 -v(各種情報を出力する指定)	1147
J.1.21 -WC(翻訳オプションの指定)	1147
J.2 LINKコマンド	1148
J.2.1 LINKコマンドのオプション	1149
J.3 DUMPBINコマンド	1150
J.4 COBOLファイルユーティリティコマンド	1151
J.4.1 ファイル変換コマンド(cobfconv)	1151
J.4.2 ファイルロードコマンド(cobfload)	1155
J.4.3 ファイルアンロードコマンド(cobfulod)	1157
J.4.4 ファイル表示コマンド(cobfbrws)	1158
J.4.5 ファイル整列コマンド(cobfsort)	1161
J.4.6 ファイル属性コマンド(cobfattr)	1165
J.4.7 ファイル復旧コマンド(cobfrcov)	1166
J.4.8 ファイル再編成コマンド(cobfreog)	1167
J.5 INSDBINFコマンド	1167
J.6 UTF-32用定義体変換コマンド	1170
付録K OSIV系システムとの機能比較	1174
K.1 機能比較一覧	1174
K.2 プログラムの動作確認における注意事項	1178

K.2.1 プログラムの起動時にOSIV系システムのパラメタを渡す.....	1179
K.2.2 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.....	1179
K.2.3 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.....	1180
付録L COBOL G移行支援オプション.....	1181
付録M 文字コードの留意点.....	1182
M.1 文字コードの概説.....	1182
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.....	1182
M.1.2 文字種の混在方式による分類.....	1183
M.2 文字コード変換.....	1184
M.3 シフトJISへのコード変換時の問題.....	1186
M.4 他システムからの移行上の注意.....	1189
M.4.1 日本語空白と英数字空白の文字コード.....	1189
M.4.2 JIS非漢字の負号について.....	1192
M.5 文字コードに関するコーディング上の留意点.....	1192
M.5.1 日本語16進文字定数.....	1193
M.5.2 表意定数.....	1193
M.5.3 文字の泣き別れ.....	1193
M.5.4 日本語文字定数.....	1193
M.5.5 項目の再定義.....	1193
M.5.6 集団項目転記.....	1194
M.5.7 空白づめ.....	1194
M.5.8 集団項目比較.....	1194
M.5.9 大小比較.....	1195
M.5.10 プログラムの作成・編集.....	1195
M.5.11 半角カナ.....	1195
M.5.12 小入出力.....	1195
M.5.13 COBOLファイル.....	1196
M.6 Windowsシステムにおけるシステムロケール.....	1196
付録N セキュリティ.....	1198
N.1 資源の保護.....	1198
N.2 アプリケーション作成のための指針.....	1198
N.3 インターネット接続対象外の通信機能.....	1199
N.4 Web連携機能のセキュリティ.....	1199
N.5 リモート開発デバッグ.....	1199
N.6 リモート開発機能.....	1199
付録O Webアプリケーション.....	1200
O.1 Web上で動作するCOBOLアプリケーションをはじめて作成する方のために.....	1200
O.1.1 Webアプリケーションとは.....	1200
O.1.2 HTTPの基礎.....	1200
O.1.3 HTMLの基礎.....	1202
O.1.4 Webブラウザ操作上の注意点と対策.....	1210
O.2 Q&A集.....	1211
O.2.1 使えない文字はある？.....	1211
O.2.2 HTML文書のテキスト入力フィールドに、半角と全角文字を混在させて入力してもいい？.....	1211
O.2.3 Webブラウザにエラーメッセージが表示されたのはなぜ？.....	1212
O.2.4 部外者からデータを保護することは必要？.....	1212
O.2.5 Webアプリケーションでファイルの入出力が正しく行われぬ.....	1212
O.2.6 Webアプリケーションで表示内容を動的に変える手段は？.....	1212
O.2.7 Webアプリケーションを起動しても正しく動作しない。どうして？.....	1212
O.2.8 Webブラウザに次のメッセージが表示された。どうして？.....	1212
O.2.9 WebブラウザにContent-typeの宣言が表示される。どうして？.....	1213
O.2.10 Webアプリケーションを実行するとWebブラウザに“ファイルをディスクに保存しますか”のメッセージが表示された。どうして？.....	1213
O.2.11 Webアプリケーション作成時にCOBOLプログラムで注意することは？.....	1213

O.2.12 Webサーバを起動させる時に必要な環境変数の設定はどうしたらいい？	1213
O.2.13 COBOLデバッガは使えない？	1213
O.2.14 動作確認のためにデバッガを起動しても、クライアントにデバッガの画面が表示されない。どうして？	1213
O.2.15 WebブラウザにWebサーバからステータスコード(Status-code)が表示された。意味は？	1214
O.2.16 タイムアウト時間を制御することはできる？	1214
O.2.17 フレームを使ったHTML文書をWebアプリケーションで使うことはできる？	1214
O.2.18 COBW3_CHECK_VALUE_Xなどで<INPUT>タグで指定したVALUEの値が正しく検索されないのは？	1214
O.2.19 絶対パスで指定したファイルがアクセスできない。どうして？	1214
O.2.20 Webブラウザに「文書にデータが含まれません。」のエラーが表示された。どうして？	1214
O.2.21 COBOLデバッガは起動されたが、デバッグが開始されない。どうして？	1214
O.2.22 サーバに「F3BIPRCT.DLLが見つかりません。」のエラーが表示される。どうして？	1214
O.2.23 ネットワーク環境の資源はアクセスできない？	1215
O.2.24 サーバから応答がない場合は？	1215
O.2.25 COBOLデバッガが起動されない。どうして？	1215
O.2.26 NetCOBOL Studioでデバッグが開始されない。どうして？	1215
O.2.27 Cookieデータに漢字を使ってもいい？	1215
O.2.28 セッション管理機能を使用したWebアプリケーションを二重に起動すると、正しく動作しないことがある。どうして？	1215
O.2.29 クライアント(Webブラウザ)側が何らかの理由で強制終了した場合、サーバ(Webサーバ)側のWebアプリケーションの動きはどうなる？	1216
O.2.30 バッチファイルからCGIアプリケーションを起動する場合、指定したコマンドをWebブラウザに表示させないためには？	1216
O.2.31 ログファイルと重要度を指定しているのに、ログファイルが出力されない。どうして？	1216
O.3 WebアプリケーションをCOBOLで作成する場合の概念図	1217
O.3.1 CGIアプリケーションをCOBOLで作成する場合の概念図	1217
O.3.2 ISAをCOBOLで作成する場合の概念図	1217
O.4 CGIからISAPIへの移行の手引き	1218
O.4.1 CGIからISAPIへの移行	1218
O.4.2 アプリケーションの形式	1219
O.4.3 翻訳、リンク方法	1219
O.4.4 実行単位	1220
O.4.5 モジュール構成	1220
O.4.6 サーバとのインタフェース領域	1221
O.4.7 復帰コード	1221
O.4.8 環境変数操作	1222
O.4.9 CGI環境変数の操作	1223
O.5 CGIサブルーチン V1.0からの移行時の注意事項について	1224
O.5.1 COBOLプログラムおよび関連資産の修正	1224
O.5.2 COBOLプログラムの翻訳およびリンク	1227
索引	1228

# 第1部 NetCOBOLとは

---

---

第1章 NetCOBOLの概要.....	2
----------------------	---

# 第1章 NetCOBOLの概要

本章では、本製品の機能および動作環境を説明します。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の機能を説明します。

### 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ファイルユーティリティ	COBOLファイルの処理
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
NetCOBOL Studio	COBOLアプリケーションの開発支援
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
プロジェクトマネージャ	COBOLアプリケーションの開発支援
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
COBOL対話型デバッガ	COBOLアプリケーションのデバッグ
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
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

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
- ・ 翻訳時メッセージの出力
- ・ 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。

## 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

## NetCOBOL Studio

NetCOBOL Studioは、COBOLプログラムを開発するための統合開発環境です。COBOLソースファイルの編集からCOBOLプログラムの翻訳・デバッグおよび実行までの一連の開発作業を効率良く行うことができます。NetCOBOL Studioの詳細は、“NetCOBOL Studio ユーザーズガイド”を参照してください。

## 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。

## 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。

## 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

## 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。

## 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

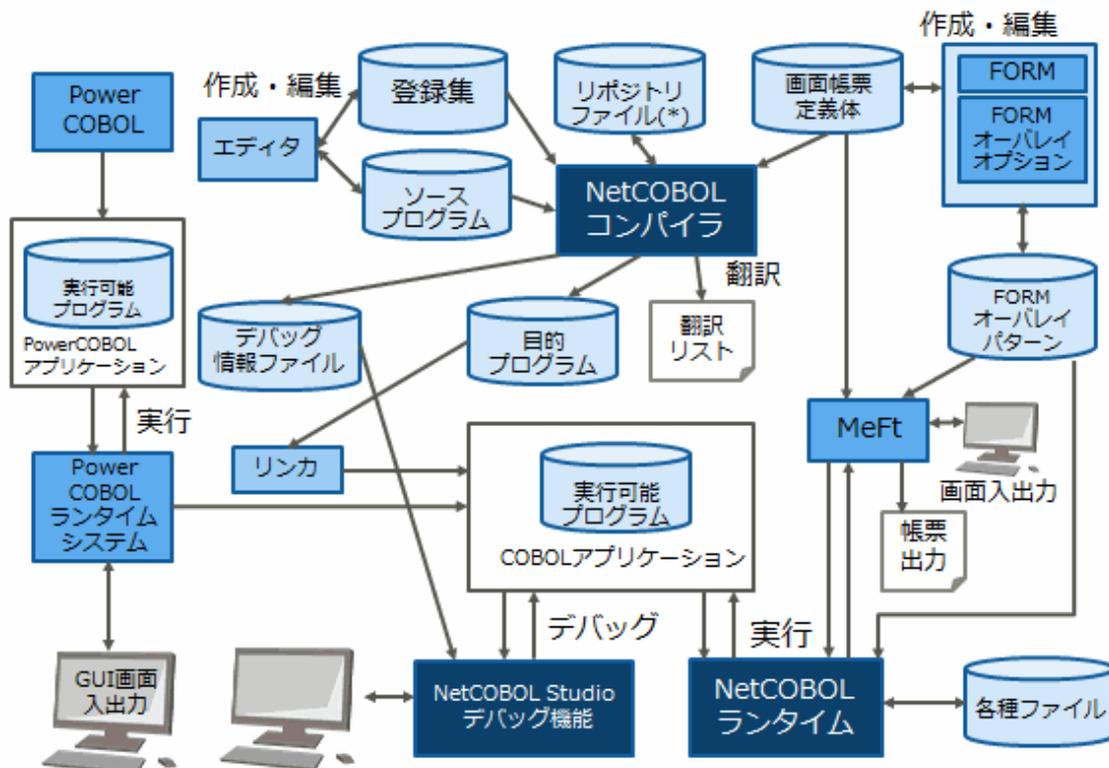
## クラスブラウザ

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

## 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。



\*: オブジェクト指向プログラム開発を行う場合に必要です。

### 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からシステムを選択して設定する。

SETコマンドによる設定方法は、コマンドプロンプトからSETコマンドのヘルプを、コントロールパネルから設定する方法は、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
	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
	COBMSG_FORMAT	翻訳コマンドが標準エラー出力に表示する診断メッセージの形式(1~4の数値)。(注3)	診断メッセージの形式を変更する場合(注4)	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
  - 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”は使用できません。
  - 環境変数名には、NetCOBOLが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。
  - 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。



COPY文の記述例: COPY TEST OF TESTLIB1

```
COB_TESTLIB1=C:¥COPYLIB
```

- 注3  
環境変数COBMSG\_FORMATに指定した値によって、翻訳コマンドが標準エラー出力に表示する診断メッセージの形式は以下のようになります。

指定する値	メッセージの形式
1	ファイル名 行番号: メッセージ番号 メッセージ内容 (デフォルト)
2	ファイル名(行番号): メッセージ番号 メッセージ内容
3	ファイル名:行番号:メッセージ番号 メッセージ内容
4	ファイル名 行番号 メッセージ番号 メッセージ内容

- 注4  
NetCOBOL開発環境(NetCOBOL Studio、COBOLビルダ)を利用する場合に指定すると、正しく表示およびタグジャンプができなくなる場合があります。

### 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.3.20 FORMEXT(画面帳票定義体ファイルの拡張子の指定)”
- “A.3.26 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、“[J.1.2 翻訳コマンドのオプション一覧](#)”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.1 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。



### 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、-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オーバーレイオプション	プログラムが出力するオーバー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
Jアダプタクラスジェネレータ	Javaクラスを呼び出すCOBOLクラス(アダプタクラス)を生成するツール
PowerSORT Server	ビジネス分野向けの高性能なソートマージツール
SIMPLIA/COBOL支援キット	アプリケーション開発・保守作業を支援するツール群

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オーバーレイオプション

FORMオーバーレイオプションは、COBOLプログラムが印刷するオーバー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、対話的にオーバー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。

## Jアダプタクラスジェネレータ

Jアダプタクラスジェネレータは、Javaクラスを呼び出すCOBOLクラス(アダプタクラス)を生成するツールです。生成したアダプタクラスを使用することにより、COBOLからJavaのクラスライブラリを利用できるようになります。

## PowerSORT Server

PowerSORT Serverは、ビジネス分野向けの高性能なソート・マージプログラムです。事務処理において取り扱う大量のデータを、高度なソート技法により短時間でソート処理できます。また、NetCOBOLシリーズに同梱されるPowerSORTはマルチ運用配下で各種性能向上を実現、多様化するユーザのバッチ処理に対応できるよう設計/作成されたサーバ専用製品です。

## SIMPLIA/COBOL支援キット

SIMPLIA(SIMple development & maintenance support Program Libraries for Application system)は、アプリケーション開発・保守作業を支援するツール群の総称です。SIMPLIA製品を導入することにより、作業効率アップ・生産性/品質向上などの効果が期待できます。

SIMPLIA/COBOL支援キットには、以下の製品が含まれています。

- SIMPLIA/TF-LINDA
- SIMPLIA/TF-MDPORT
- SIMPLIA/MF-STEP COUNTER
- SIMPLIA/TF-EXCOUNTER
- SIMPLIA/DF-COBDOC
- SIMPLIA/VF-FILECOMP

NetCOBOLシリーズ製品への同梱の有無については、“NetCOBOL ソフトウェア説明書”を参照してください。

## 1.2.3 資源一覧

本製品での資源の一覧を下表に示します。

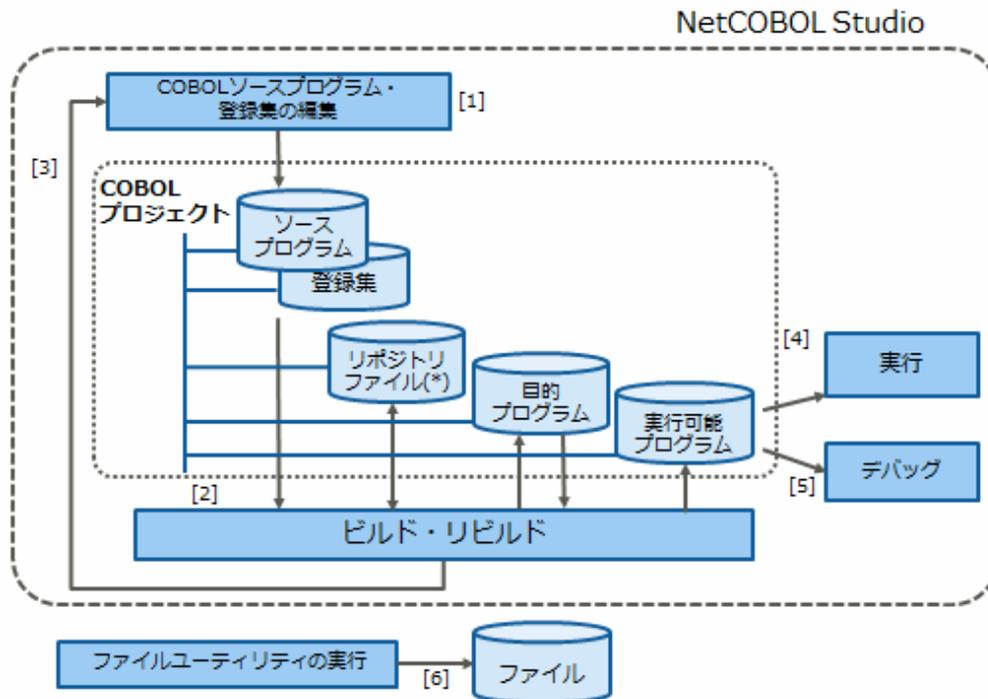
資源名	内容	主なコンポーネント	ファイル名命名規約	推奨拡張子
ソースプログラム	COBOLソースプログラム	コンパイラ	任意	cob cobol
登録集	登録集原文	コンパイラ	任意	cbl
画面帳票定義体	画面および帳票の定義情報	FORM MeFt MeFt/Web	任意	smd pmd pxd
FORMオーバーレイパターン	フォームオーバーレイパターン情報	FORM MeFt MeFt/Web	任意	ovd
リポジトリファイル	クラス関連情報	コンパイラ	クラス名.rep	rep
ソース解析情報ファイル	ソース解析情報	コンパイラ	ソース名.sai	sai
目的プログラム	目的プログラム	コンパイラ	ソース名.obj	obj
実行可能プログラム	実行形式プログラム	ランタイムシステム	任意	exe
ダイナミックリンクライブラリ	実行形式プログラム	ランタイムシステム	任意	dll
翻訳リスト	翻訳リスト情報	コンパイラ	任意	lst
テキスト	COBOL行順ファイルなど	ランタイムシステム	任意	txt
順	COBOL順ファイル	ランタイムシステム	任意	seq
相対	COBOL相対ファイル	ランタイムシステム	任意	rel

資源名	内容	主なコンポーネント	ファイル名命名規約	推奨拡張子
索引	COBOL索引ファイル	ランタイムシステム	任意	idx
実行用の初期化ファイル	COBOLの実行時に設定する環境変数の定義	ランタイムシステム	COBOL85.CBR (任意)	CBR
印刷情報	プリンタ種別などの印刷フォーマット定義	ランタイムシステム	任意	—
フォントテーブル	印刷ファイルで使用する書体番号の定義	ランタイムシステム	任意	—
FCB	1ページ分の行数、行間隔、開始行などの定義	ランタイムシステム	任意(4文字)	—
クラス情報	実行時に獲得するオブジェクトインスタンス数などの指定	ランタイムシステム	任意	—
トレース情報(最新)	トレース機能で出力される実行経路情報	ランタイムシステム	実行形式名.trc	trc
トレース情報(一世代前)	トレース情報の一世代前の情報	ランタイムシステム	実行形式名.tro	tro
COUNT情報	COUNT機能による統計情報	ランタイムシステム	任意	—
ウィンドウ情報	画面(表示ファイル)の各種ウィンドウ情報	MeFt/Web	任意	—
プリンタ情報	帳票印刷時での各種プリンタ情報	MeFt MeFt/Web	任意	—
デバッグ情報	デバッグ機能用のデバッグ情報	デバッグ	ソース名.svd	svd

## 1.3 プログラム開発

### 1.3.1 NetCOBOL Studioを利用した開発

NetCOBOL Studio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。詳細は、“NetCOBOL Studio ユーザーズガイド”の“チュートリアル”を参照してください。



\* : オブジェクト指向プログラム開発を行う場合に必要です。

[1]

エディタを使って、COBOLソースプログラムを記述します。ソースプログラム、登録集はプロジェクトに登録します。

[2]

ビルドまたはリビルドを行います。これにより、プログラムの翻訳およびリンクが実行されます。

[3]

翻訳エラーが発生した場合、エラーの発生したCOBOLの文を修正します。

[4]

COBOLアプリケーションを実行します。

[5]

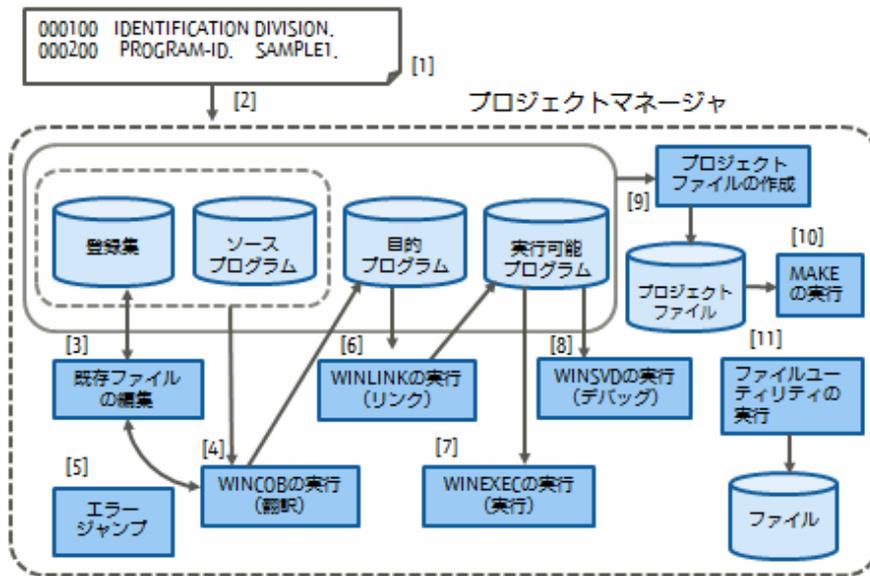
プログラムが期待どおり動かない場合やプログラムの単体テストを行う場合、NetCOBOL Studioのデバッグ機能を使ってデバッグします。

[6]

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”](#)

### 1.3.2 プロジェクトマネージャを利用した開発

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]“[25.1.2 WINCOBを使用した翻訳の手順](#)”
- [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]“[23.4.6 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]“[25.2.2 WINLINKを使用したリンクの手順](#)”
- [7]  
WINEXECを実行し、プログラムを実行します。[参照]“[25.3.2 WINEXECを使用した実行の手順](#)”
- [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24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23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]“[23.4.5 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](#)”



## 参考

オブジェクト指向プログラム開発の手順は“[第27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。

### 1.3.3 開発環境を利用しない開発

---

コマンドプロンプト上から翻訳、リンク、実行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。

- “[第3章 プログラムを翻訳する](#)”
- “[第4章 プログラムをリンクする](#)”
- “[第5章 プログラムを実行する](#)”

## 1.4 リモート開発

---

「クライアント側」のNetCOBOL Studioと「サーバ側」のNetCOBOLを使用してリモート開発を行うと、COBOLアプリケーションを効率よく開発することができます。

### クライアント側

NetCOBOL Studioを含むNetCOBOL製品がインストールされていることが必要です。

開発者は、クライアント側のNetCOBOL Studioを使用して開発します。NetCOBOL Studioは、必要に応じてサーバ側に接続し、サーバ側で翻訳などの開発作業を行います。作業の結果は、NetCOBOL Studioが表示します。

### サーバ側

NetCOBOL製品がインストールされていることが必要です。

### リモート開発のメリット

COBOLアプリケーションの多くは、高価なサーバマシンで運用されます。これらのCOBOLアプリケーションを同様のシステム上で開発する場合、以下の問題があります。

- これらのシステムではGUIベースの環境が用意されていない場合が多く、コマンドラインを用いて開発作業を行う必要があります。
- 複数の開発者がサーバマシンを共有する必要があります。

一方、Windowsシステムは個人用端末として広く普及しており、これを利用するとGUIベースの環境を開発者が占有することができます。

リモート開発では、開発作業をなるべくWindowsシステムで行うようにすることで、上記の問題点を解決します。

- GUIベースの開発環境を用いて、開発を効率よく行うことができます。
- ソースの編集など可能な作業をクライアント側で行うことにより、貴重なサーバマシンの負荷を減らします。

リモート開発の詳細は、“[NetCOBOL Studio ユーザーズガイド](#)”を参照してください。

## 第2部 プログラムの編集から翻訳・リンク・実行まで

ここでは、コマンドを使用した翻訳・リンク・実行の方法を説明します。

NetCOBOL Studioを使用したCOBOLプログラムの編集からビルド・実行・デバッグ方法の説明は、“NetCOBOL Studioユーザーズガイド”を参照してください。

また、プロジェクトマネージャを使用したCOBOLプログラムの翻訳・実行・デバッグ方法の説明は、以下を参照してください。

- “25.1.2 WINCOBを使用した翻訳の手順”
- “25.2.2 WINLINKを使用したリンクの手順”
- “25.3.2 WINEXECを使用した実行の手順”

第2章 プログラムを作成・編集する.....	15
第3章 プログラムを翻訳する.....	18
第4章 プログラムをリンクする.....	23
第5章 プログラムを実行する.....	35

## 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。

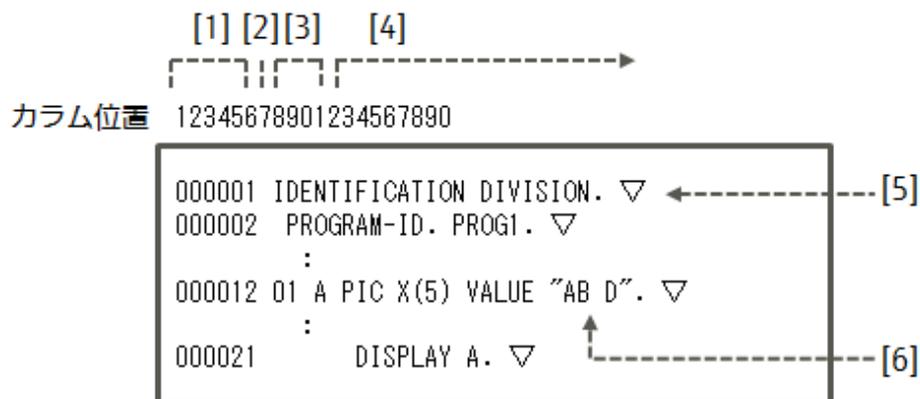
##### エディタの起動

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.3.26 LIBEXT (登録集ファイルの拡張子の指定)”



### 参考

画面帳票定義体は、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.3.55 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)



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、参考にしてください。

### 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]“[25.1.4 \[翻訳オプション\]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、NetCOBOLコマンドプロンプト上で、COBOL32コマンドを使ってSample1のプログラムを翻訳します。以降の説明では、サンプルプログラムの格納先ファイルをC:\COBOL\Samples\COBOL\Sample01\Sample1.cobとします。

1. NetCOBOLコマンドプロンプトは、[スタート]>お使いのNetCOBOL製品名>[NetCOBOLコマンドプロンプト]を選択すると、表示されます。
2. COBOL32コマンドを指定して実行します。

```
c:\COBOL\Samples\COBOL\Sample01>COBOL32 -WC, "MAIN(WINMAIN)" Sample1.cob
c:\COBOL\Samples\COBOL\Sample01>
```

— -WC,"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使用する指定。

3. 翻訳が終了したら、サンプルプログラムの格納先(ここでは、C:\COBOL\Samples\COBOL\Sample01)に、目的プログラム(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。



#### 参考

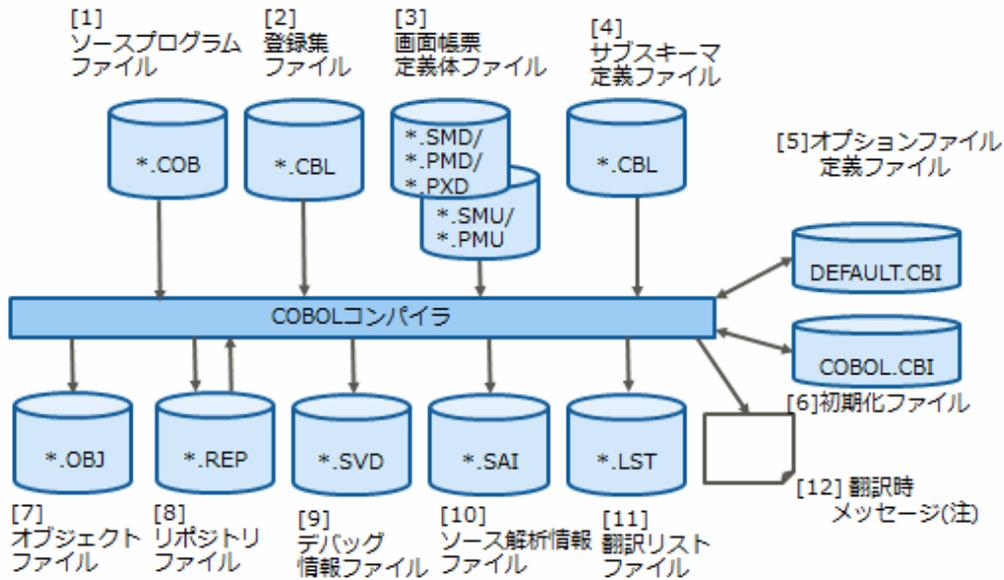
- NetCOBOL Studioを利用したサンプルプログラムのビルド手順については、“NetCOBOL Studioユーザーズガイド”の“チュートリアル”または“NetCOBOL 入門ガイド”を参照してください。
- プロジェクトマネージャを利用したサンプルプログラムの翻訳手順については、“[25.1.3 サンプルプログラムの翻訳](#)”を参照してください。

### 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。

- 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/\*.\*.SMU/\*.\*.PMU)
- 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)
- リポトリファイル(\*.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)

図3.1 COBOLコンパイラが使用するファイルの関係



注：翻訳終了時に出力されます。翻訳メッセージの詳細は、“[診断メッセージ](#)”を参照してください。

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 .PMD .PXD .SMU .PMU (注4)(注5)(注12)	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	FORMLIB SMED_SUFFIX
[4]	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
[5]	オプション文字列	DEFAULT.CBI または 任意(注7)	入力  出力	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  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	—	—

	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]	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	—	—
[7]	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	—
[8]	クラス情報 (注8)	クラス名.REP	入出力	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
[9]	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	—
[10]	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	—
[11]	翻訳リスト(注11)	ソースファイル名 .LST	出力	翻訳リストを出力する場合	PRINT	—
[12]	翻訳時メッセージ	—	—	翻訳終了後にウィンドウ上に出力される	—	—

#### 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。

- 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.3.26 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.3.25 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

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。

#### NetCOBOL Studioを利用する場合

1. [登録集名]ページで指定したフォルダを検索します。[参照]“NetCOBOL Studioユーザーズガイド”の“登録集名の設定”
2. [登録集名]ページに指定がない場合、環境変数COB\_登録集名に指定したフォルダを検索します。

#### プロジェクトマネージャを利用する場合

1. [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]“登録集名の設定”
2. [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。

#### 開発環境を利用しない場合

1. 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または環境変数SMED\_SUFFIXを使って任意の文字列に変更できます。[参照]“A.3.20 FORMEXT (画面帳票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、PMUおよびSMU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.3.21 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。

#### 注7

オプションファイルの作成方法および使用方法は、“J.1.14 -i (オプションファイルの指定)”を参照してください。

#### 注8

リポジットファイルの詳細は、“17.2.3.2.1 リポジットファイル”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、“27.6.2 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、“NetCOBOL ユーザーズガイド (OSIV分散開発・プロジェクトマネージャ編)”または“NetCOBOL ユーザーズガイド (OSIV分散開発・NetCOBOL Studio編)”を参照してください。

#### 注11

翻訳リストの種類と詳細は、“付録B 翻訳リスト”を参照してください。

#### 注12

拡張子SMUおよびPMUは、UTF-32用の定義体です。[参照]“エンコードUTF-32の利用について”

### 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を指定して翻訳する。[参照]“A.3.42 PRINT (各種翻訳リストの出力の可否および出力先の指定)”
- -Pオプションを指定して翻訳する。[参照]“J.1.18 -P (各種翻訳リストの出力および出力先の指定)”
- プロジェクトマネージャを利用する場合、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、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の詳細は、“B.1 診断メッセージリスト”を参照してください。

## 3.3 翻訳の手順

---

COBOLソースプログラムを翻訳するには、以下の方法があります。

- 翻訳コマンド(COBOL32コマンド)を使用する。[参照]“3.4 翻訳コマンド”

- NetCOBOL Studioを使用する。[参照]“NetCOBOL Studioユーザーズガイド”の“COBOLプログラムのビルド”
- WINCOBウィンドウを使用する。[参照]“25.1.2 WINCOBを使用した翻訳の手順”

## 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.3.35 NAME(翻訳単位ごとのオブジェクトファイルの出力の可否)”

## 3.4 翻訳コマンド

翻訳コマンド(COBOL32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。また、翻訳コマンドの指定形式と翻訳オプションの一覧は、“J.1 COBOL32コマンド”を参照してください。

### 例

例1 : 1つのソースファイルを翻訳する

```
c:¥COBOL>COBOL32 -M test.cob
```

test.cobは主プログラムとして翻訳され、目的プログラムtest.objが作成されます。

[補足]WINCOBウィンドウを使用して翻訳する方法は、“25.1.2.1 1つのソースファイルを翻訳するときの手順”を参照してください。

例2 : 複数のソースファイルを翻訳する(複数翻訳)

```
c:¥COBOL>COBOL32 -M -O test1.cob test2.cob
```

test1.cobは主プログラム、test2.cobは副プログラムとして翻訳され、tes1.objとtest2.objが作成されます。

最適化(-O)は両方のプログラムに適用されます。

[補足]WINCOBウィンドウを使用して翻訳する方法は、“25.1.2.2 複数のソースファイルを翻訳するときの手順”を参照してください。

## 注意

1つのCOBOL32コマンドに複数のソースファイルを指定して翻訳することを、複数翻訳といいます。複数翻訳では、複数のソースファイルに対して、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”

## 第4章 プログラムをリンクする

本章では、サンプルプログラムのリンク手順を説明した後、必要な資源、プログラム構造、リンクコマンド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、LINKコマンドを使って、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、目的プログラムの格納先ファイルをC:\¥COBOL¥Samples¥COBOL¥Sample01¥Sample1.OBJとします。

NetCOBOL コマンドプロンプト

```
C:\¥COBOL¥Samples¥COBOL¥Sample01>LINK /OUT:Sample1.EXE Sample1.OBJ F3BICIMP.LIB MSVCRT.LIB
```

リンクが終了したら、サンプルプログラムの格納先(ここでは、C:\¥COBOL¥Samples¥COBOL¥Sample01)に実行可能ファイル(Sample1.EXE)が作成されているか確認してください。



#### 参考

- NetCOBOL Studioを利用したサンプルプログラムのビルド手順については、“NetCOBOL Studioユーザーズガイド”の“チュートリアル”を参照してください。
- プロジェクトマネージャを利用したサンプルプログラムのリンク手順については、“25.2.3 サンプルプログラムのリンク”を参照してください。

### 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、リンクコマンドを使って行います。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”

WINLINKを使用した目的プログラムのリンクは、“25.2.2 WINLINKを使用したリンクの手順”を参照してください。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。

#### 4.2.1 リンクコマンドが使用するファイル

リンクコマンドは、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)

リンクコマンドが使用するファイルを下表に示します。

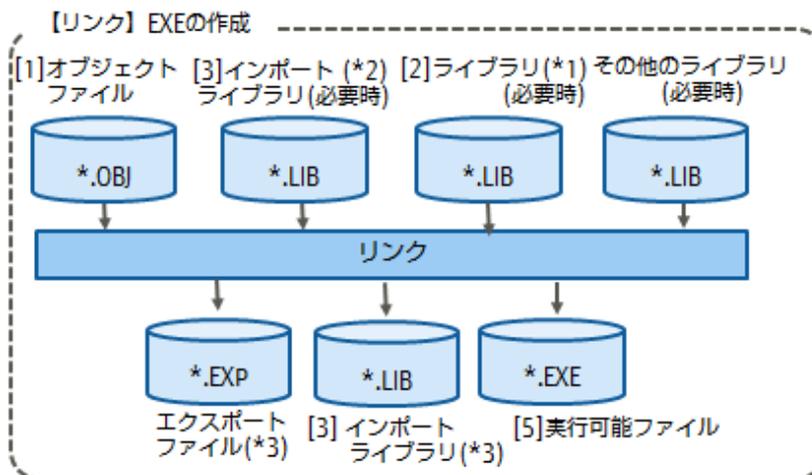
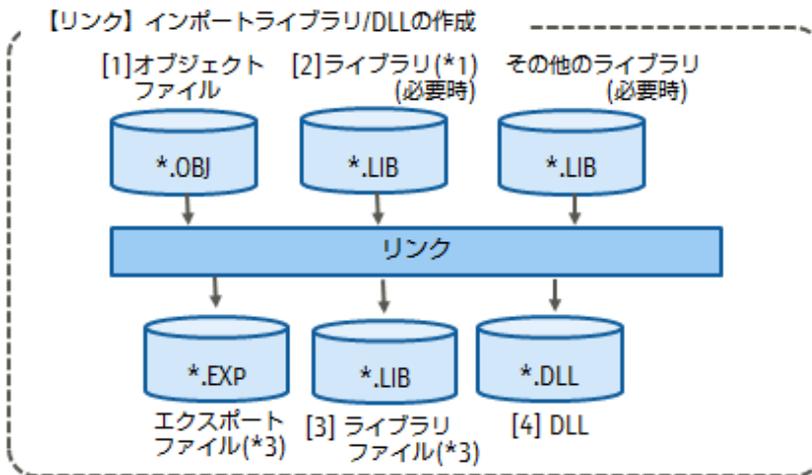
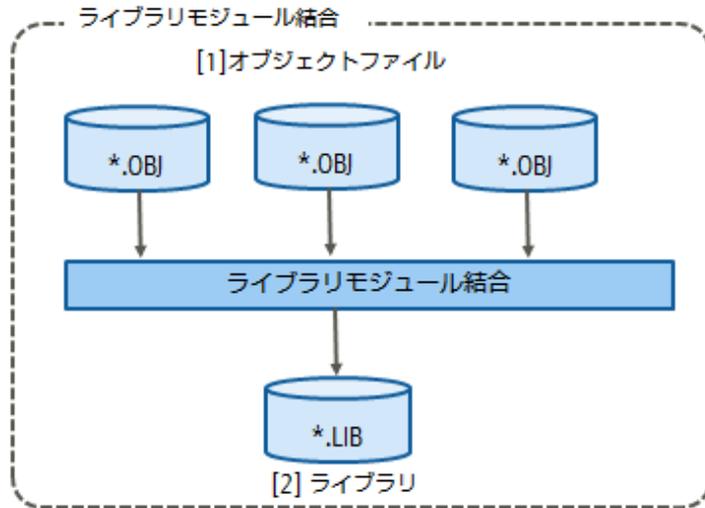


表4.1 リンクコマンドが使用するファイル

	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)(*1)	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。

	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
		オブジェクトファイル名.LIB または 任意の名前.LIB	出力	LIBコマンドを使用することで、1つ以上の目的プログラムを結合して1つのライブラリを作成することができます。また、WINLINK[ライブラリモジュール結合]ウィンドウを使用して作成することもできます。詳細は、“ <a href="#">25.2.2.7 ライブラリを作成するときの手順</a> ”を参照してください。
[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	DLLの作成時に、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。

\*1: LIBコマンドまたはWINLINK[ライブラリモジュール結合]ウィンドウを使用して作成しておきます。LIBコマンドを使用することで、1つ以上の目的プログラムを結合して1つのライブラリを作成することができます。WINLINK [ライブラリモジュール結合]ウィンドウの使用方法は、“[25.2.1.2 WINLINK\[ライブラリモジュール結合\]ウィンドウ](#)”を参照してください。

\*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。



### 注意

リンクオプション「/DEBUG」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、NetCOBOL Studioのデバッグ機能、対話型デバッガ、クラッシュダンプ、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

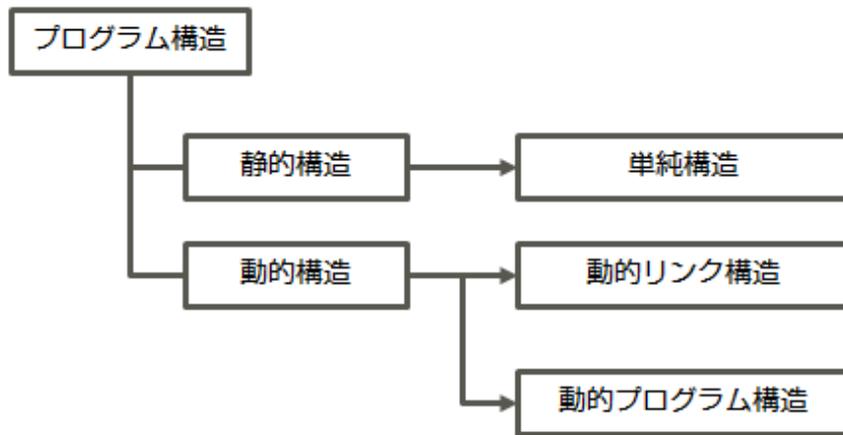
インポートライブラリは、アプリケーション実行時に、実行中のアプリケーションとDLLとの間にダイナミックリンクを確立するための情報を含んでいます。動的リンク構造で実行可能ファイルを作成する場合には、必ずインポートライブラリが必要となります。必要なライブラリが実行時にどこにあるかを探すための情報をインポートライブラリから実行可能プログラムにコピーします。つまり、インポートライブラリはアプリケーションとDLLとの仲立ちの役目をするものです。

## 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、副プログラムといえます。

図4.1 プログラム構造

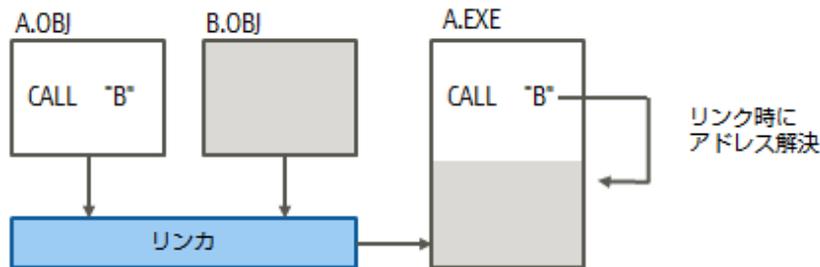


### 4.3.1 静的構造

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

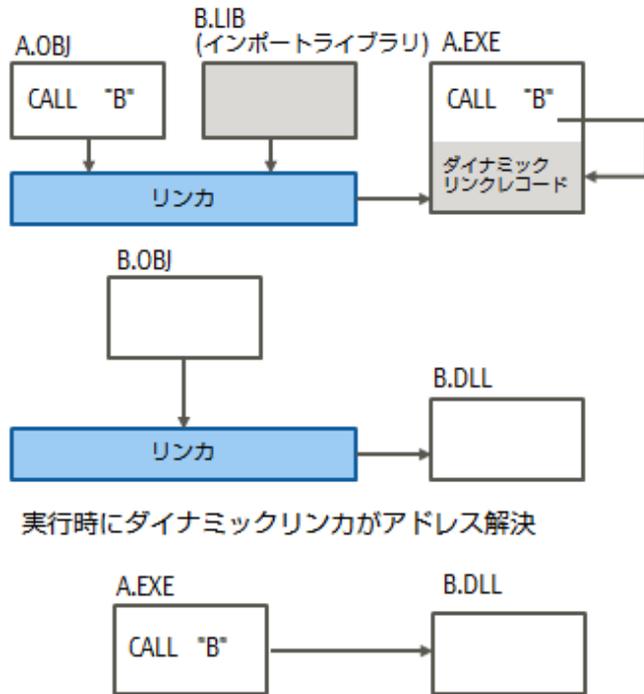
動的構造とは、実行時に必要となるモジュールを、リンカによっていくつかのDLLとし、プログラムの実行時に必要なDLLを仮想記憶上にロードし、実行させる構造です。

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。



### 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.3.12 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 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。

### 4.4.1 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。詳細は、“25.2.1.1.5 モジュール定義ファイルの生成”を参照してください。

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。

```

-----
;
;  INSATSU.DLL 用モジュール定義ファイル
;
-----
;  LIBRARY : ダイナミックリンクライブラリの名前を指定します。
LIBRARY "INSATSU"
;  EXPORTS : プログラムIDとENTRYの名前を指定します。
;      注) WINLINKが作成したモジュール定義ファイルはオブジェクト名が指定されます。
;          正しい名前に変更して使用してください。
EXPORTS
        INSATSU
    
```

## 4.4.2 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

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、LINKコマンドに以下を指定します。

```
/DEF:モジュール定義ファイル名
```

モジュール定義ファイルは1つしか LINKコマンドに指定できません。

## 4.5 リンクコマンドを使ったリンク操作

リンクコマンドは、目的プログラムをリンクして、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。

### 4.5.1 LINKコマンド

LINKコマンドは、リンク操作をNetCOBOL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、“[J.2 LINKコマンド](#)”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
- MSVCRT.LIB

MSVCRT.LIBは、実行可能プログラム(EXE)を作成する場合または、DLLからC関数を呼び出す場合に指定します。

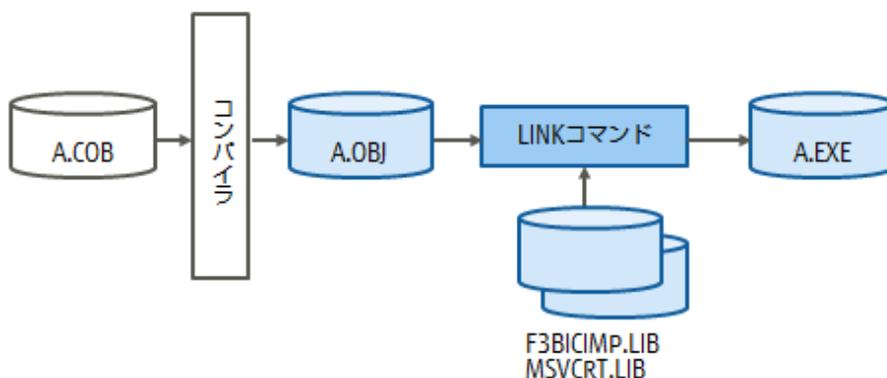


V11.0.0より使用するリンクカのバージョンが変わります。これに伴い、CランタイムライブラリはLIBC.LIBからMSVCRT.LIBに変わります。

### 4.5.2 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。

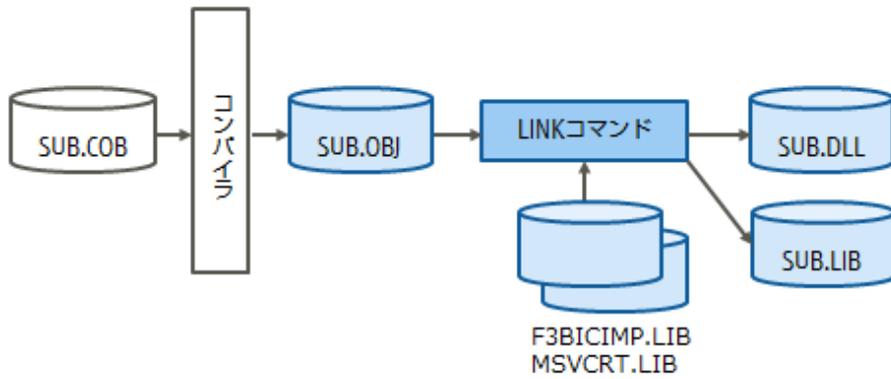
1つの目的プログラムをリンクする場合



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

/OUT: メイン出力ファイルの名前を指定

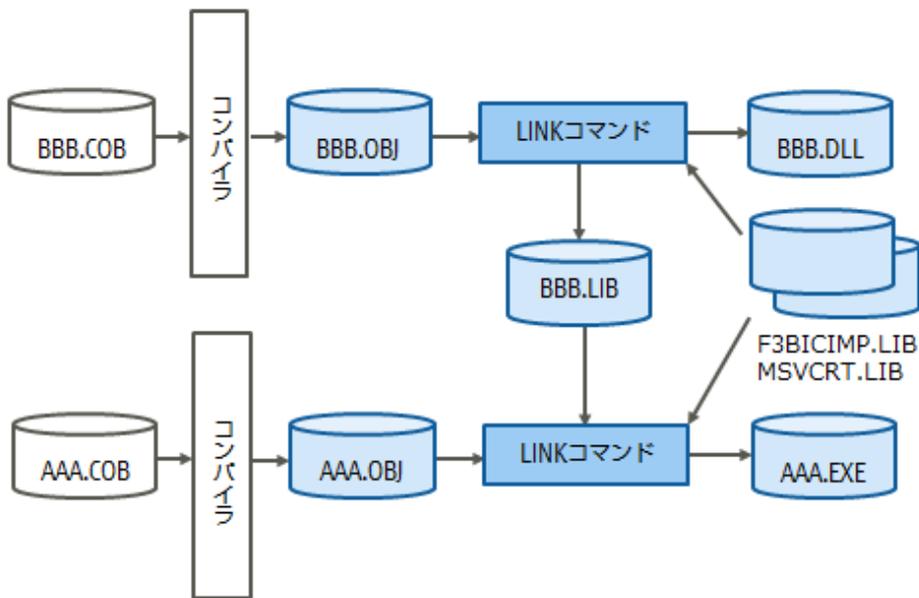
DLLを作成する場合



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

/OUT : メイン出力ファイルの名前を指定

## 4.5.3 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 MSVCRT.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  
MSVCRT.LIB : C ランタイムライブラリ

リンクすると、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 MSVCRT.LIB /DLL /OUT:SUB2.DLL
```

SUB2.OBJ : プログラムSUB2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  
MSVCRT.LIB : C ランタイムライブラリ

リンクすると、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 F3BICIMP.LIB MSVCRT.LIB /DLL /OUT:SUB3.DLL
```

SUB3.OBJ : プログラムSUB3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  
MSVCRT.LIB : C ランタイムライブラリ

リンクすると、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。

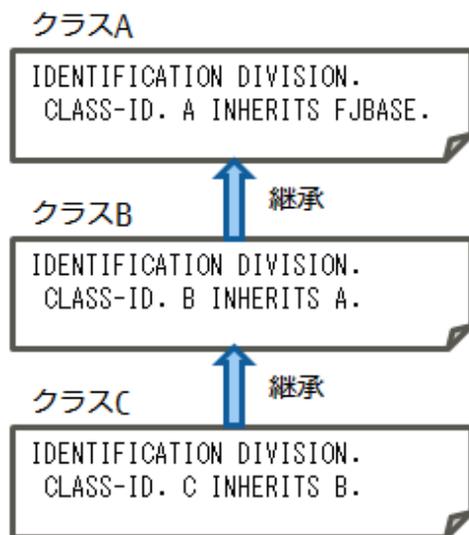
作成されたインポートライブラリを使用して、プログラムAをリンクします。

プログラムAをリンクする

```
LINK A.OBJ F3BICIMP.LIB MSVCRT.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  
F3BICIMP.LIB : COBOL ランタイムシステムのインポートライブラリ  
MSVCRT.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 MSVCRT.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  
MSVCRT.LIB : C ランタイムライブラリ

リンクすると、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 MSVCRT.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  
MSVCRT.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

リンクすると、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。

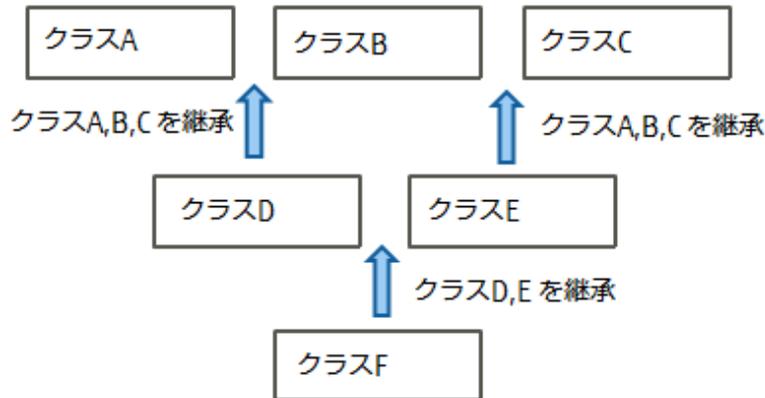
クラスCのDLLを作成する

```
LINK C.OBJ F3BICIMP.LIB MSVCRT.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  
MSVCRT.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

リンクすると、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 MSVCRT.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  
MSVCRT.LIB : C ランタイムライブラリ

リンクすると、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 MSVCRT.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  
MSVCRT.LIB : C ランタイムライブラリ

リンクすると、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 MSVCRT.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  
MSVCRT.LIB : C ランタイムライブラリ

リンクすると、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 MSVCRT.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  
MSVCRT.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

リンクすると、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 MSVCRT.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  
MSVCRT.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

リンクすると、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 MSVCRT.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  
MSVCRT.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。

## 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、NetCOBOLコマンドプロンプト上で、例題1のプログラムを実行します。

以降の説明では、実行可能ファイルをC:\¥COBOL¥Samples¥COBOL¥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. “実行環境設定ツール”を起動します。  
→ [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¥COBOL¥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¥COBOL¥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. NetCOBOLコマンドプロンプト上で、Sample1.EXEを実行します。

```
C:\¥COBOL¥Samples¥COBOL¥Sample01>Sample1.EXE
```

2. 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。

コンソール: **SAMPLE1**

```
アルファベットを1文字（小文字）入力してください。=>
```

→ 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。

```
アルファベットを1文字（小文字）入力してください。=>      c
cobol
```

実行後、「コンソールウィンドウを閉じます。」というメッセージボックスが表示されます。

3. 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られます。

以上でサンプルプログラムの実行は終わりです。



#### 参考

- NetCOBOL Studioを利用したサンプルプログラムの実行手順については、“NetCOBOL Studioユーザズガイド”の“チュートリアル”を参照してください。

- ・ プロジェクトマネージャを利用したサンプルプログラムの実行手順については、“[25.3.3 サンプルプログラムの実行](#)”を参照してください。
- 

## 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。

### 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。

### 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。

## 5.3 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

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。

環境変数情報の詳細は、“[付録C 環境変数情報](#)”を参照してください。

#### エントリ情報

動的プログラム構造のアプリケーションを実行する場合に必要な情報です。

エントリ情報の詳細は、“[5.6 エントリ情報](#)”を参照してください。

### 5.3.2 実行環境情報の設定方法

---

#### 環境変数情報を設定する

環境変数情報を設定する方法には、以下があります。

##### a. 環境変数に設定する

- － コントロールパネルからシステムを選択して設定する

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便利です。

- － SETコマンドで設定する

コマンドプロンプトから設定した環境変数情報は、そのコマンドプロンプトから起動されたプログラムに対して環境変数情報が有効になります。

バッチファイルの中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になります。

それぞれの設定方法は、WindowsのヘルプおよびコマンドプロンプトからSETコマンドのヘルプを参照してください。

##### b. 実行用の初期化ファイルに設定する

実行用の初期化ファイルに設定された環境変数情報は、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。

- 一 実行環境設定ツールで設定する。  
実行環境設定ツールの詳細は、“5.7 実行環境設定ツール”を参照してください。
- 一 テキストエディタで設定する。

## 参考

実行用の初期化ファイルと環境変数に設定した環境変数情報の設定が重複する場合、実行用の初期化ファイルに設定した値が有効になります。

## エントリ情報を設定する

“5.6.2 エントリ情報の指定方法”を参照してください。

## 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。
  - 一 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。
  - 一 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。

## 5.4 プログラムの実行

COBOLプログラムを実行するには、以下の方法があります。

- ・ コマンドプロンプトのコマンドラインから実行する。(\*)
- ・ 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。(\*)
- ・ NetCOBOL Studioを使用して、COBOLプログラムを実行する。[参照]“NetCOBOL Studioユーザーズガイド”の“COBOLプログラムの実行”
- ・ WINEXECウィンドウを使用して、COBOLプログラムを実行する。[参照]“25.3.2 WINEXECを使用した実行の手順”

\*: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。

## 注意

COBOLプログラムの実行時、実行用の初期化ファイルはCOBOLランタイムシステムがアクセスしています。COBOLプログラムの実行が終了するまで、実行用の初期化ファイルに対して以下の操作を行わないでください。

- ・ 他のプログラムでの参照および更新
- ・ エディタを使用するの参照および更新
- ・ 複写

上記の操作を行った場合、実行用の初期化ファイルの情報が有効にならない場合があります。

## 5.4.1 コマンドラインから実行する

実行可能ファイルをコマンド名として起動します。以下に指定形式を示します。

```
実行可能ファイル名 [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。

### コマンドの引数

引数は、空白で区切って指定します。引数に空白を含めたい場合には、その引数を二重引用符(“)で囲んでください。

### 実行時パラメタ

以下の場合、実行時パラメタを指定します。

- ACCEPT文/DISPLAY文でコマンド行引数を操作する  
[参照]“[11.2 コマンド行引数の取出し](#)”
- OSIV系形式で実行時パラメタを使用する  
[参照]“[OSIV系形式の実行時パラメタの指定方法](#)”

これらは、プログラムの記述に従って判断されます。

### 例

```
PROG1.exe A B C,D
```

引用として“A”、“B”、“C,D”の3つの引数が指定されました。

```
PROG1.exe "A B C,D"
```

引数として、“A B C,D”という1つの引数が指定されました。

### -CBR 実行用の初期化ファイル名

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。

### 例

```
PROG1.exe -CBR ABC.INI
```

実行用の初期化ファイル名として、カレントフォルダのABC.INIが指定されました。(PROG1.exeが主プログラムの場合)

### 参考

主プログラムが他言語(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、“[5.5.5 主プログラムが他言語の場合の実行用の初期化ファイル名の指定方法](#)”を参照してください。

### -CBL 実行時オプション

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。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詳細は、“[5.8 実行時オプション](#)”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。

実行時オプションは、環境変数情報@GOPT(実行時オプションの指定)でも指定できます。



## 参考

コマンドラインからの指定と環境変数情報@GOPTでの指定が重複した場合、コマンドラインに指定された値が有効になります。

### 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。詳細は、“[K.2.1 プログラムの起動時にOSIV系システムのパラメタを渡す](#)”を参照してください。



## 例

```
$ PROG1.exe "ABCDE"
```

OSIV系形式の実行時パラメタとして、ABCDEが指定されます。

OSIV系形式の実行時パラメタは、環境変数情報@MGPRM(OSIV系形式の実行時パラメタの指定)でも指定できます。



## 参考

コマンドラインからの指定と環境変数情報@MGPRMでの指定が重複した場合、コマンドラインに指定された値が有効になります。

## 5.4.2 バッチファイルを使用する

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、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。

表5.1 オペレータの入力待ちなしにCOBOLプログラムを実行するための環境変数情報

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。

バッチファイルからプログラムを起動した場合、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。



## 例

### 環境変数情報を実行用の初期化ファイルに設定する場合

実行用の初期化ファイル(COBOL85.CBR)に環境変数情報を設定します。

実行用の初期化ファイル(COBOL85.CBR)

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

バッチファイル(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
```



## 例

### 環境変数情報を環境変数に設定する場合

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。

バッチファイル(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、“メッセージ集”の“第4章 実行時メッセージ”を参照してください。

## 5.4.3 サービス配下での注意事項(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。

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、特別な指定は必要ありません。

## 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、「[表5.1 オペレータの入力待ちなしにCOBOLプログラムを実行するための環境変数情報](#)」を参照してください。

なお、サービス配下での印刷機能の使用時の注意点は、「[8.1.15 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。



### 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。

## 5.5 実行用の初期化ファイル

### 5.5.1 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、「[5.5.4 DLL配下にある実行用の初期化ファイルを使用する](#)」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、「[5.5.5 主プログラムが他言語の場合の実行用の初期化ファイル名の指定方法](#)」を参照してください。
- ・ 環境変数情報で指定する場合は、「[C.2.3 @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、「[5.4.1 コマンドラインから実行する](#)」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。

### 5.5.2 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。



## 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、“付録C 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、“付録C 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<sup>(注1)</sup>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、“5.6 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。

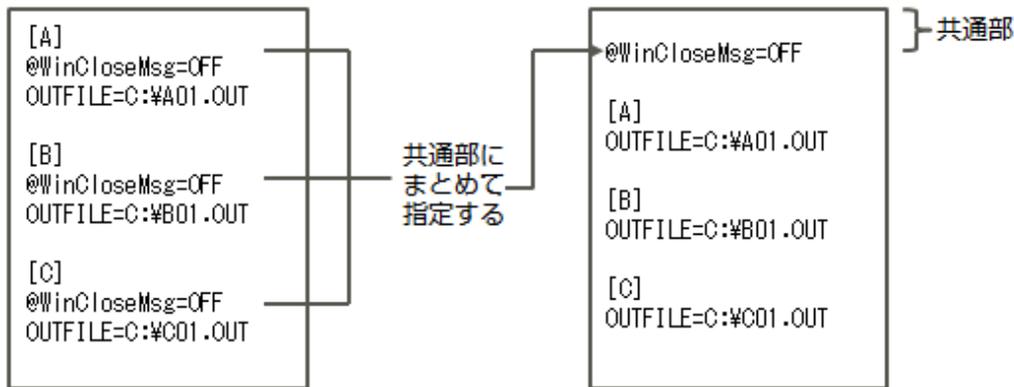
注1：他言語プログラムからCOBOLプログラムをJMPCINT2およびJMPCINT3を使用して呼び出す場合、最初に呼び出すCOBOLプログラムのプログラム名を指定してください。COBOLの主プログラムおよび実行環境については、“10.1.4 COBOLの言語間の環境”を参照してください。

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、“5.6 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      -----
@CnsIWinSize=(80, 24)          ↑
@CnsIBufLine=100              共通部（環境変数情報）
@WinCloseMsg=ON                ↓
@IconName=COB85EXE            -----
                                ↑
[PROG1]                        PROG1の環境変数情報
INFILE=C:¥FILE¥INFILE.TXT     ↓
                                -----
                                ↑
[PROG1.ENTRY]                  PROG1のエントリ情報
A=X.DLL                        ↓
B=Y.DLL                        -----

```

### 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。

## 5.5.3 実行用の初期化ファイルの検索順序

実行用の初期化ファイルの検索順序を以下に示します。

- EXEファイルのフォルダ配下のCOBOL85.CBR
- 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
- 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]“5.5.5.1 主プログラムがCプログラムの場合”、“5.5.5.2 主プログラムがVisual Basicプログラムの場合”、“5.4.1 コマンドラインから実行する”

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]“C.2.4 @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	—

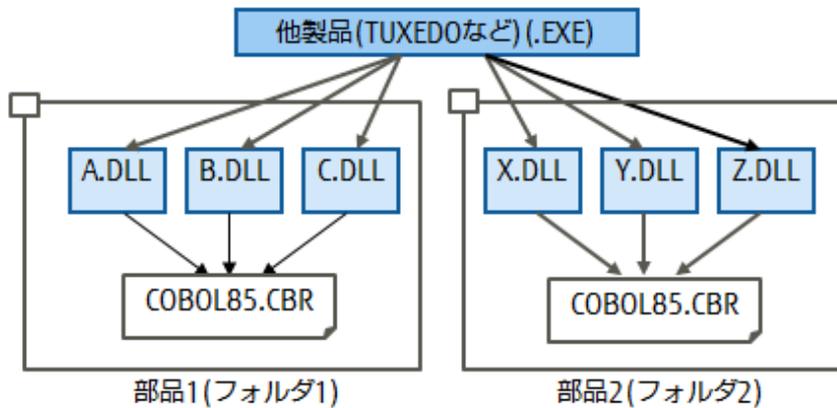
## 5.5.4 DLL配下にある実行用の初期化ファイルを使用する

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트리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트리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트리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트리オブジェクトの結合

DLLエン트리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트리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:COBDMAIN /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트리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

リンクオプション”/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 : MSVCRT.LIBを指定してください

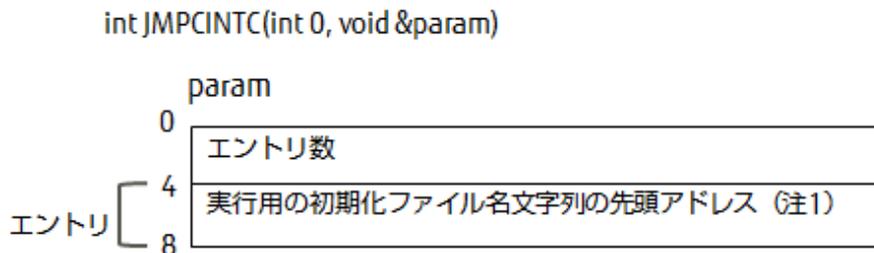
## 5.5.5 主プログラムが他言語の場合の実行用の初期化ファイル名の指定方法

ここでは、プログラムの主プログラムが他言語(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。

### 5.5.5.1 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。

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。



エントリ数: 現在は1となる

注1 : 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。)

### 5.5.5.2 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 "F3B1PRCT.DLL" ()
Private Declare Sub JMPCINT3 LIB "F3B1PRCT.DLL" ()
Private Declare Function JMPCINTB LIB "F3B1PRCT.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。)

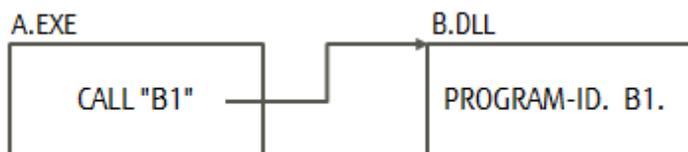
JMPCINTBの返却値	意味
0	正常終了
-1	JMPCINT2の未呼出し/機能コードの指定誤り/作業領域不足
1	多重呼出し (JMPCINTBがすでに呼び出されている場合を意味します。実行用の初期化ファイル名は、最初に実行されたJMPCINTBで指定されたものが有効となります。)

## 5.6 エントリ情報

### 5.6.1 エントリ情報とは

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だけ必要です。呼び出すプログラムが格納されているDLLを特定するために使用されます。

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、エントリ情報は必要ありません。たとえば、呼び出すエントリ名が“B1”で、DLL名が“B1.DLL”の場合、エントリ情報は不要です。



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。

## 5.6.2 エントリ情報の指定方法

エントリ情報の指定法には、以下の2つがあります。

- a. エントリ情報ファイルに設定する
- b. 実行用の初期化ファイルに設定する

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。



- 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。
- エントリ情報ファイルと実行用の初期化ファイルでエントリ情報の設定が重複する場合は、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。

## 5.6.3 エントリ情報の形式

### 5.6.3.1 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

指定の例は、以下を参照してください。

- “例1：DLLが1つの副プログラムで構成されている場合”
- “例2：DLLが複数の副プログラムで構成されている場合”



DLL名の拡張子は、“DLL”でなければなりません。

### 5.6.3.2 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。

指定の例は、以下を参照してください。

- “例3：二次入口名を指定して呼び出す場合”

## 5.6.4 エントリ情報ファイルの内容

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、“[5.6.3.1 副プログラム名の指定形式](#)”、“[5.6.3.2 二次入口点名の指定形式](#)”および“[17.3.2.4.4 クラスとメソッドのエントリ情報](#)”を参照してください。

### 注意

- ・ 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。
- ・ 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。

## 5.6.5 エントリ情報ファイル名の指定

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ます。[参照]“[C.2.26 @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が「C:¥TEST¥FILE.ENT」の場合

```
@CBR_ENTRYFILE = C:¥TEST¥FILE.ENT
```

## 5.6.6 実行用の初期化ファイルに含まれるエントリ情報の内容

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
:
:
```

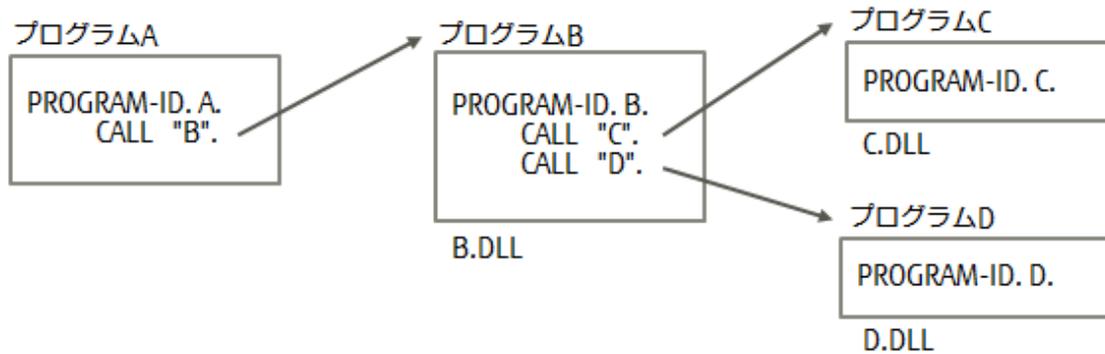
[1] 主プログラムのプログラム名(PROGRAM-ID段落に指定した名前)をセクション名として指定します。

[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。

## 5.6.7 エントリ情報の設定例

例1 : DLLが1つの副プログラムで構成されている場合

【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
```

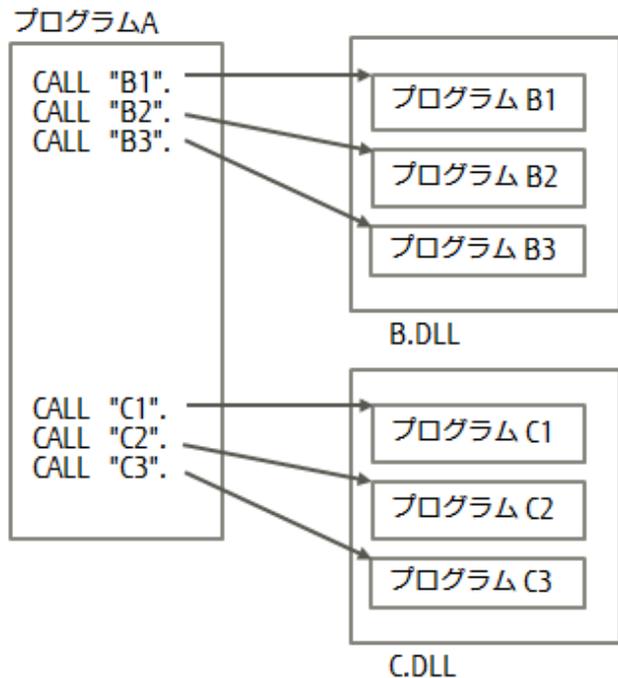
(2) 実行用の初期化ファイルに指定する場合

```
[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。

## 例2 : DLLが複数の副プログラムで構成されている場合

【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
```

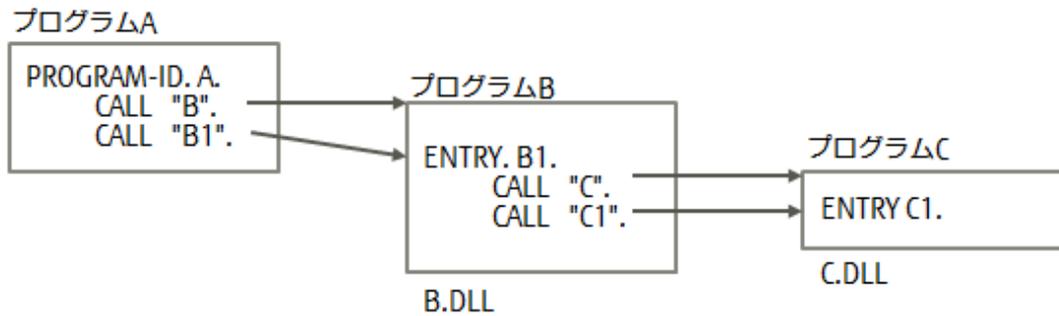
DLL内の呼び出されたすべての副プログラムに対してCANCEL文が実行されたときに、DLLは仮想記憶上から削除されます。

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とき、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文を実行する場合は、“[20.10.5 動的プログラム構造](#)”を参照してください。

### 例3：二次入口名を指定して呼び出す場合

【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
B1=B
C1=C
```

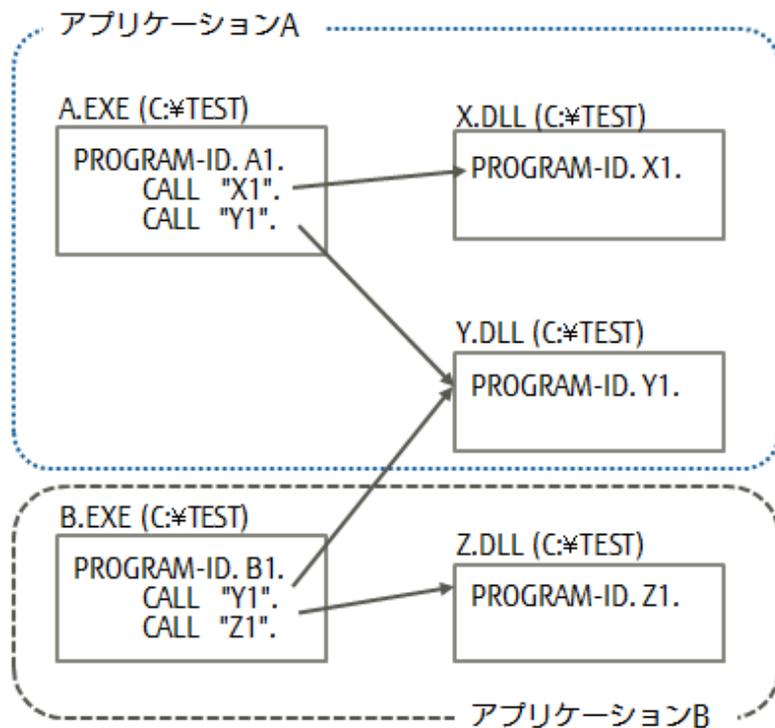
(2) 実行用の初期化ファイルに指定する場合

```
[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。

### 例4：2つのアプリケーションでDLLを共有している場合

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(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。

#### 実行用の初期化ファイル(C:¥TEST¥COBOL85.CBR)

```
[A1]
:
[A1.ENTRY]
X1=C:¥TEST¥X.DLL
Y1=C:¥TEST¥Y.DLL
:
[B1]
:
[B1.ENTRY]
Y1=C:¥TEST¥Y.DLL
Z1=C:¥TEST¥Z.DLL
```

2箇所に指定が必要

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。

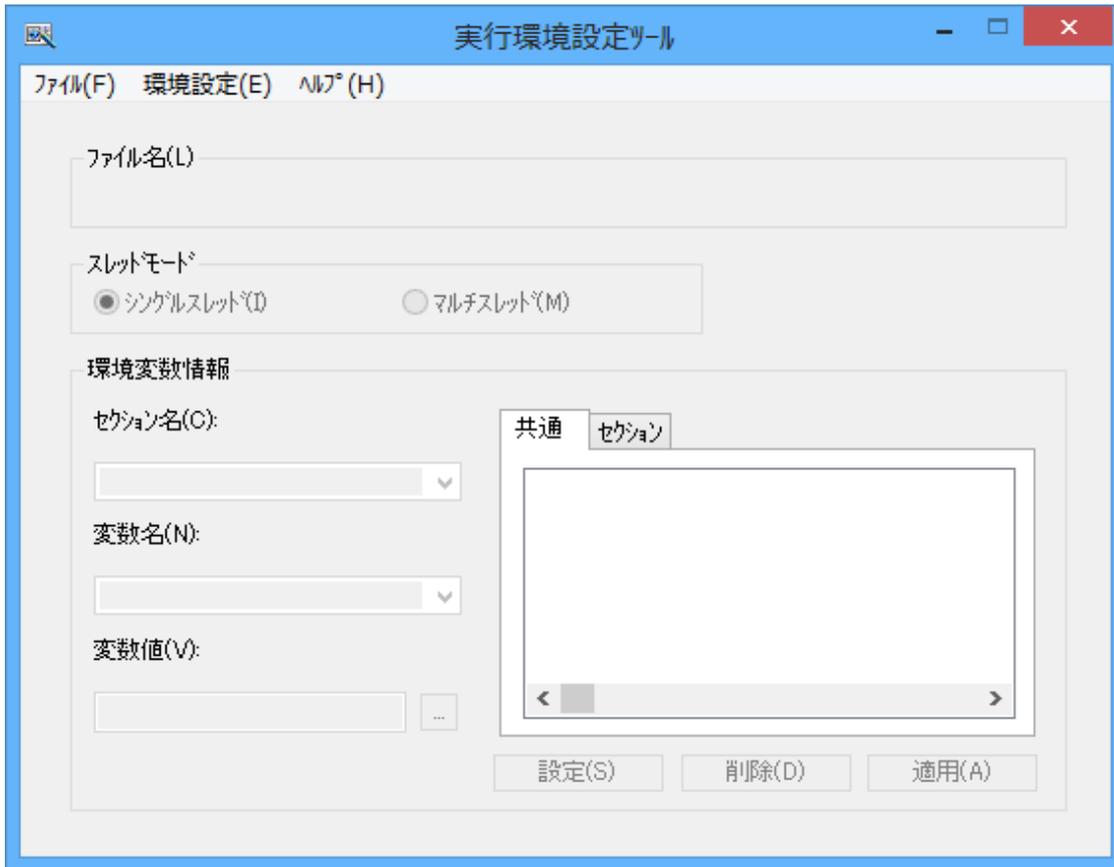
## 5.7 実行環境設定ツール

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、“5.5.3 実行用の初期化ファイルの検索順序”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。

設定する個々の実行環境情報の指定形式の詳細は、“付録C 環境変数情報”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。

## 5.7.1 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。

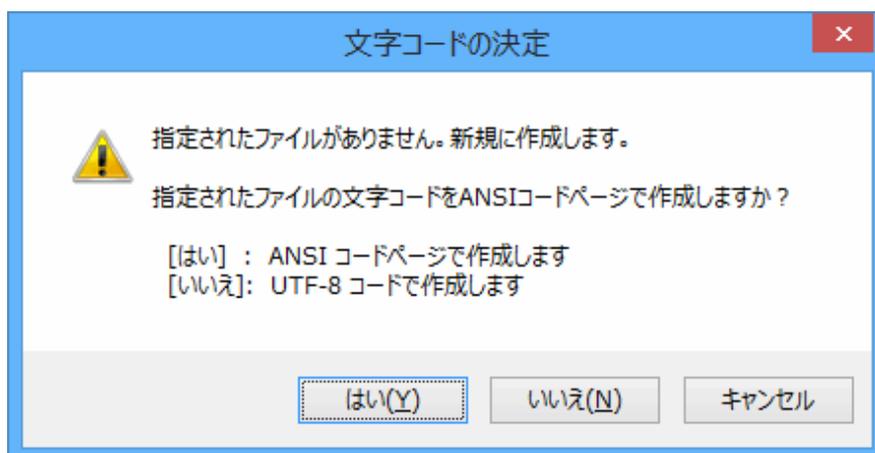
ここでは、環境変数情報を設定するときの操作方法について説明します。

### 5.7.1.1 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。

### 5.7.1.2 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。

### 5.7.1.3 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。

### 5.7.1.4 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。

### 5.7.1.5 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。

## 5.7.2 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、“C.2.26 @CBR\_ENTRYFILE(エントリ情報ファイルの指定)”を参照してください。

## 5.7.3 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。

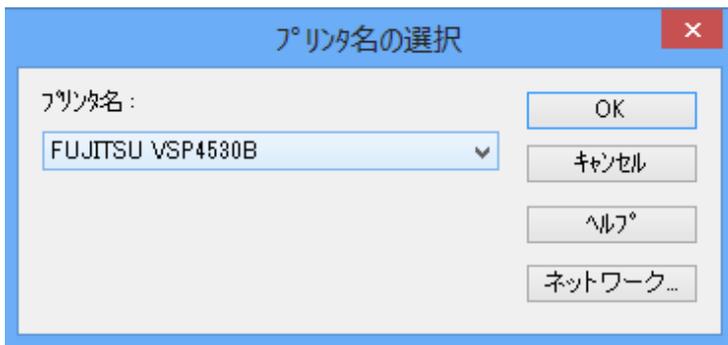


注意

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。

## 5.7.4 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。



## 5.7.5 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。

## 5.8 実行時オプション

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

“5.4.1 コマンドラインから実行する”

“C.2.62 @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5.2 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 <code>個数</code>   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 <code>回数</code>   { noc   nocb   noci   nocl   nocn   nocp } ]
外部スイッチの値の指定	[ s <code>値</code> ]
PowerSORTが使用するメモリ容量の最大値を指定	[ smsize <code>値k</code> ]

### [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

“J.1.9 -Dr (TRACE機能を使用する指定)”

“A.3.62 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

“J.1.6 -Dk (CHECK機能を使用する指定)”

“A.3.5 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の最大値を指定)

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のユーザーズガイドをお読みください。

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度は、特殊レジスタSORT-CORE-SIZEが一番高く、以降、実行時オプションsmsize、翻訳オプションSMSIZE()の順で低くなります。

## 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

.....  
“J.2 LINKコマンド”

.....  
“J.2.1 LINKコマンドのオプション”  
.....

#### 使用するスタックサイズの求め方

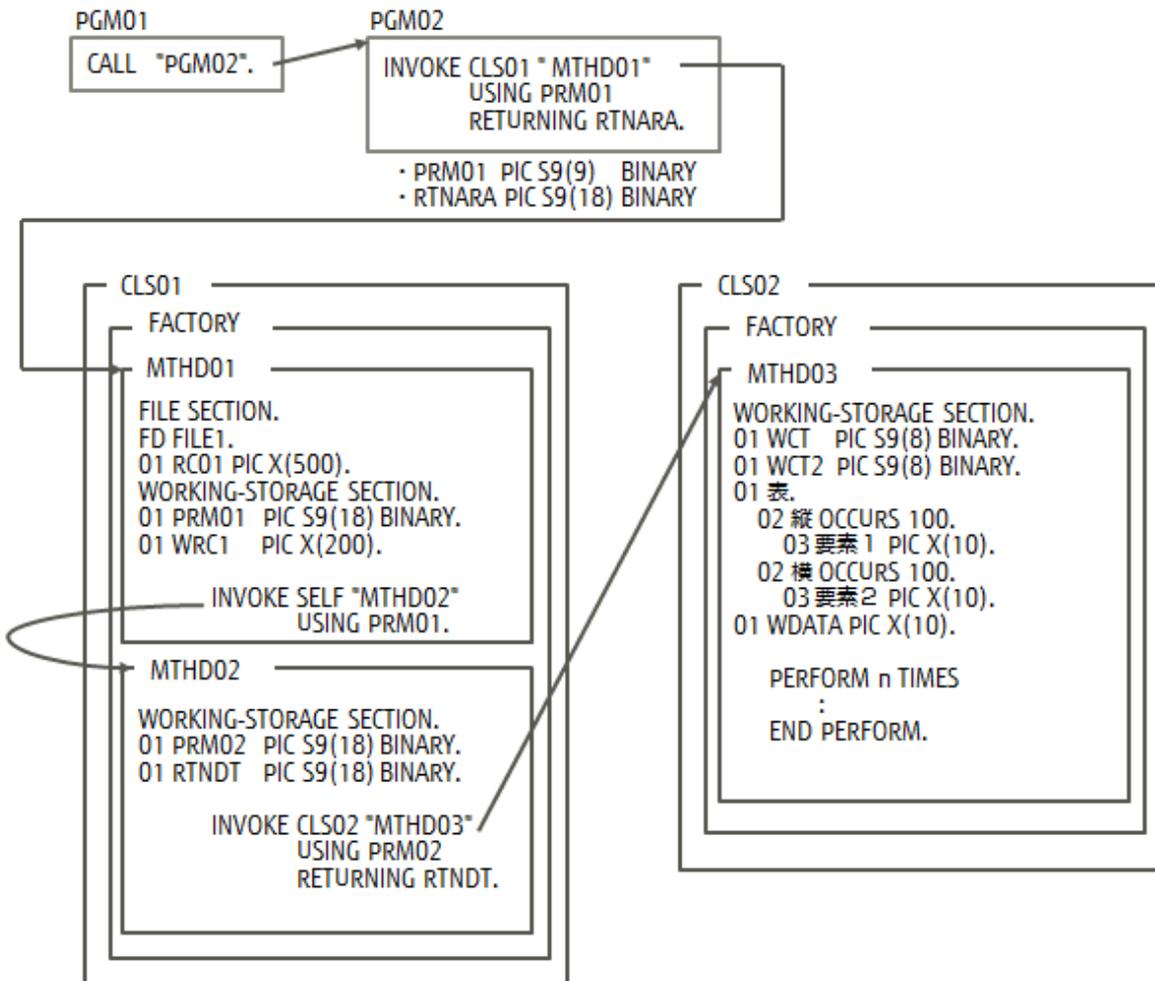
$$\text{トータルスタックサイズ} \doteq (\alpha_1 + \alpha_2 \cdots + \alpha_n) + (\beta_1 + \beta_2 \cdots + \beta_m)$$

$\alpha_n$  : 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、...)

$\beta_m$  : 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、“[B.6 データエリアに関するリスト](#)”を参照してください。

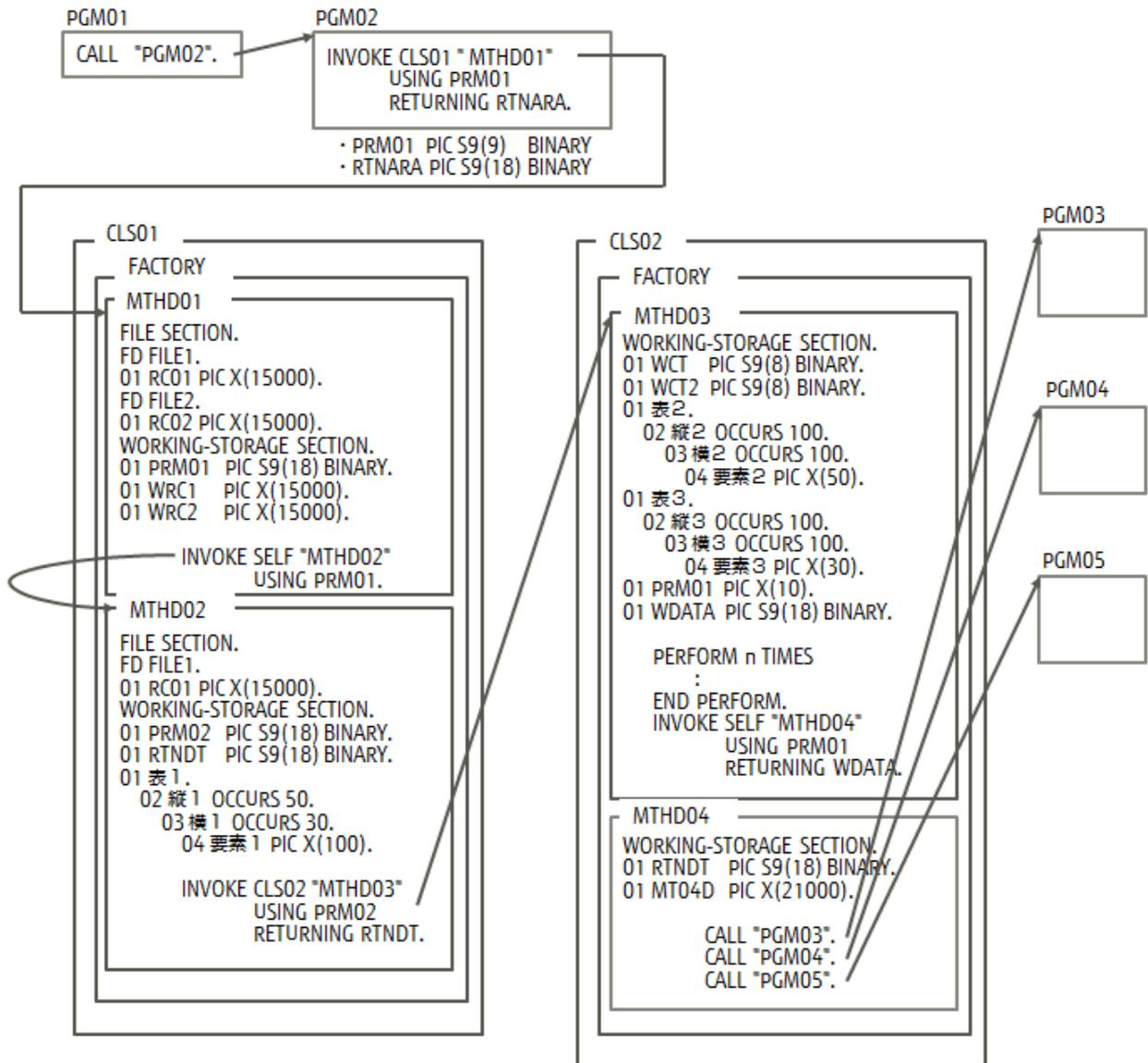
(例題1)オーバーフローしないケース



例題1でのスタック使用状況

PGM (プログラム定義)	
PGM01	176バイト
PGM02	200バイト
CLS (クラス定義)	
CLS01	
METHOD (1)	1,064バイト
METHOD (2)	376バイト
CLS02	
METHOD (3)	2,520バイト
-----	
計	4,360バイト

(例題2)オーバフローするケース



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

### スタックのオーバーフローを回避する方法

オーバー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。  
→ NetCOBOL Studioのデバッグ機能、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

.....  
“NetCOBOL Studio ユーザーズガイド”

“第24章 対話型デバッガの使い方”

“21.3 CHECK機能”

“21.6 メモリチェック機能”  
.....

## 第3部 アプリケーションの開発と運用

---

---

第6章 文字コード.....	65
第7章 ファイルの処理.....	79
第8章 印刷処理.....	144
第9章 画面を使った入出力.....	210
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～.....	225
第11章 ACCEPT文およびDISPLAY文の使い方.....	264
第12章 SORT文およびMERGE文の使い方～整列併合機能～.....	287
第13章 CSV形式データの操作.....	295
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～.....	301
第15章 リモートデータベースアクセス.....	304
第16章 通信機能.....	372
第17章 オブジェクト指向プログラミング機能.....	388

## 第6章 文字コード

文字コードは、計算機で文字を表現する仕組みです。

### 6.1 文字コードの概念

一般に広く利用されている文字コードには様々なものがありますが、ここではCOBOLで使用できる文字コードを説明します。

#### ・ シフトJIS

PCで広く利用されているコード系です。英数字・カナ文字(JIS8)、日本語文字(JIS漢字)を混在させる方法で、次のような特徴を持ちます。

- 文字種の切り換えにシフトコードを使用しません。
- 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。
- 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記号等を追加したもの(R90)や、またMicrosoft社により別の文字を追加しているもの(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。

#### ・ Unicode

世界中の文字を表現することを目的にユニコードコンソーシアムにより作成された文字コードです。

次のような特徴を持ちます。

- UTF-8、UTF-16、UTF-32など、複数のエンコードを持っています。
- エンコードによって、1文字の長さは1～6バイトになります。
- UTF-16、UTF-32には、データを上位バイトから並べるビッグエンディアンと、下位バイトから並べるリトルエンディアンがあります。

#### ・ EBCDIC-JEF

富士通のメインフレーム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)を表現でき、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。
- EBCDICは1バイト、JEFは2バイトで表現されます。

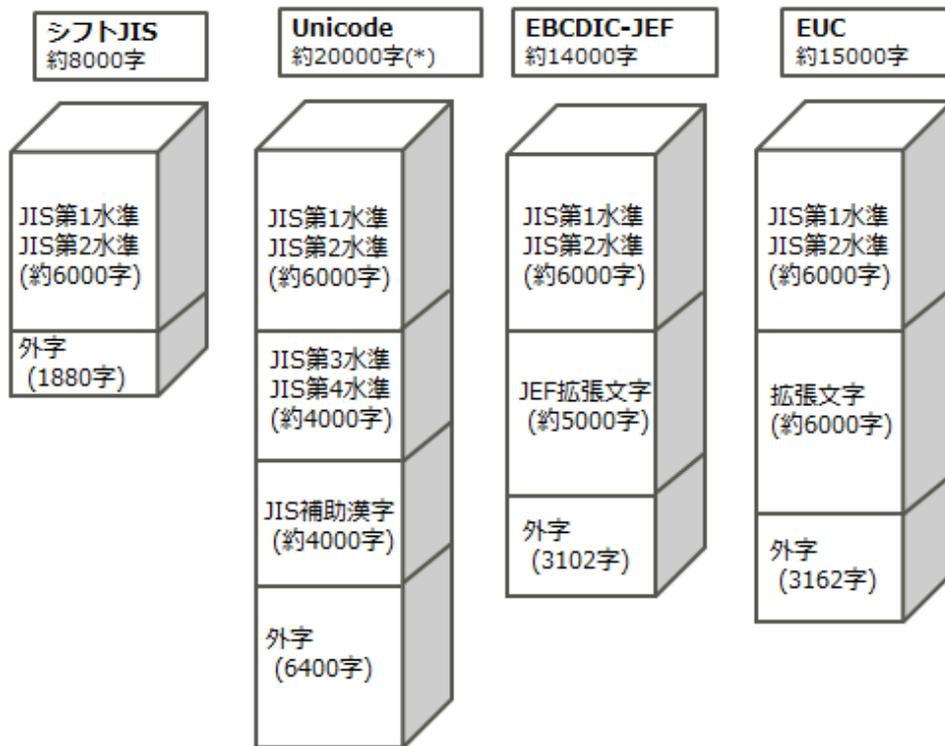
#### ・ EUC

UNIX系のシステムで広く利用されていました。(近年は利用が減っています。)

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)を表現でき、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つの領域に含まれます。
- 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。この領域にシフトJIS(R90)、EBCDIC-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。



\*:JIS2004(Windows Server 2008およびWindows 7以降において標準で採用されている文字集合)と標準で使用できる外字

OSがサポートするコード系とNetCOBOLがサポートするコード系の対応表を以下に示します。

OS		OSがサポートする 文字コード	富士通COBOLがサポートする 文字コード
メインフレーム		EBCDIC+JEF	EBCDIC+JEF
UNIX系	Solaris	EUC シフトJIS Unicode(UTF-8)	EUC(*1) シフトJIS Unicode
	Linux	Unicode(UTF-8)	EUC(*4) Unicode
	Linux(64)	Unicode(UTF-8)	Unicode シフトJIS(*3)
Windows(32)		シフトJIS Unicode	シフトJIS Unicode EBCDIC+JEF(*2)
Windows(64)		シフトJIS Unicode	シフトJIS Unicode

\*1: 外字については、内部コードにCOBOL-EUCと呼ばれる特殊な形式を採用しています。

\*2: NetCOBOL JEFオプション(別売り)が必要です。

\*3: 翻訳オプションRCSの指定が必要です。

\*4: 旧OSの互換用です。

以降では、本製品(Windowsシステム上で動作するNetCOBOL)がサポートしている文字コードを前提に説明します。

## 6.2 文字コードの指定

### 6.2.1 文字データのエンコード

Windowsシステムでサポートしている文字コードは、シフトJISとUnicodeです。

COBOLでは、文字を扱うためのデータ型として英数字項目 (PIC X)と日本語項目 (PIC N)があります。それぞれの項目のエンコードを下表に示します。

シフトJISの場合

項目のレベル	種類	字類	エンコード
基本項目	英数字項目	英数字	シフトJIS
	日本語項目	日本語	シフトJIS
集団項目	-	英数字	シフトJIS

Unicodeの場合

項目のレベル	種類	字類	エンコード
基本項目	英数字項目	英数字	UTF-8
	日本語項目	日本語	UTF-16 UTF-32(*1)
集団項目	-	英数字	UTF-8

\*1: UTF-32は、NetCOBOL V11以降でサポートしています。

シフトJISは、英数字・カナ文字(JIS8)と日本語文字(JIS第1,2水準漢字)を混在させる文字コードで、英数字・カナ文字は英数字項目を、日本語文字は日本語項目を使用します。なお、推奨する使い方ではありませんが、英数字項目中に日本語文字を格納することも可能です。いずれもエンコードはシフトJISになります。

Unicodeには、複数のエンコードがあります。

NetCOBOLでは、英数字項目のエンコードにはUTF-8を、日本語項目のエンコードにはUTF-16またはUTF-32を採用しています。

UTF-8では、1文字を格納するために必要な領域長は1~4バイトで変化します。ASCIIの範囲であれば1バイトですが、ギリシャ文字や一部記号類は2バイト、日本語文字は3バイト、一部のJIS第4水準漢字は4バイトとなります。

UTF-16では、1文字を格納するために必要な領域長は2バイトまたは4バイトです。ISO/IEC 10646で0面(BMPと呼ぶ)に配置された文字は2バイト、2面に配置された文字(JIS第4水準漢字の一部)は4バイトとなります。後者はサロゲートペアと呼ばれます。

UTF-32では、1文字を格納するために必要な領域長は4バイトです。ISO/IEC 10646で0面(BMP)に配置された文字も4バイトで表現します。なお、UTF-16およびUTF-32では、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です。



Windows 8.1以降およびWindows Server 2012以降では、IVS(Ideographic Variation Sequence)と呼ばれる表現形式を使用できますが、本COBOLコンパイラおよびランタイムシステムでは、IVSは使用できません。

## 6.2.2 エンコードの指定

データ項目のエンコードは、ENCODING句によって決定します。

```
01 データ名 1 PIC X(nn) [ENCODING IS 符号系名 1].
01 データ名 2 PIC N(nn) [ENCODING IS 符号系名 2].
```

ENCODING句には符号系名を指定します。この符号系名がエンコードを表します。NetCOBOLでは、以下のエンコードに対して符号系名を定義できます。

文字コード	字類	エンコード	補足
シフトJIS	英数字	SJIS	シフトJIS
	日本語	SJIS	シフトJIS
Unicode	英数字	UTF8	UTF-8
	日本語	UTF16	UTF-16
		UTF16BE	UTF-16 ビッグエンディアン
		UTF16LE	UTF-16 リトルエンディアン
		UTF32	UTF-32
		UTF32BE	UTF-32 ビッグエンディアン
		UTF32LE	UTF-32 リトルエンディアン

※ SJISのエンコード方式は日本工業規格JIS X 0208附属書1の規則に従います。

UTF8、UTF16、UTF16LE、UTF16BE、UTF32、UTF32LE、またはUTF32BEのエンコード方式は国際規格ISO/IEC 10646の規則に従います。

例えば、Unicodeアプリケーションの場合に、英数字項目はUTF-8、日本語項目はUTF-16LEとUTF-32LEを混在させて使用したい場合は、以下のとおり定義します。

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
  ALPHABET
    SJ  FOR ALPHANUMERIC IS SJIS
    U8  FOR ALPHANUMERIC IS UTF8    ...[1]
    U16L FOR NATIONAL    IS UTF16LE ...[2]
    U32L FOR NATIONAL    IS UTF32LE ...[3]
.
WORKING-STORAGE SECTION.
01 DATA1 PIC X(10) ENCODING IS U8.    *> エンコードはUTF8になります
01 DATA2 PIC N(10) ENCODING IS U16L. *> エンコードはUTF16LEになります
01 DATA3 PIC N(10) ENCODING IS U32L. *> エンコードはUTF32LEになります
```

[1] エンコードUTF8に対して符号系名"U8"を定義します。

[2] エンコードUTF16LEに対して符号系名"U16L"を定義します。

[3] エンコードUTF32LEに対して符号系名"U32L"を定義します。

ENCODING句は、ファイル記述項や集団項目にも指定できます。

この場合、ENCODING句を明に指定していない項目に有効となります。

```
01 A ENCODING IS U8 OR U32L.
02 A1 PIC X(10).          *> エンコードはUTF8です。
02 A2 PIC N(10) ENCODING IS U16L. *> エンコードはUTF16LEです。
02 A3 PIC N(10).          *> エンコードはUTF32LEです。
```

また、異なるエンコードが指定された場合、強さの関係は、

となります。



### 注意

以下のとおり、翻訳単位内でシフトJISとUnicodeを混在させて使用することはできません。

```
01 A.
  02 A1  PIC X(10) ENCODING IS SJ.
  02 A2  PIC N(10) ENCODING IS U16L.
```

## 6.2.3 翻訳オプションによるエンコードの指定

前述したとおり、ENCODING句は省略できます。ENCODING句が省略された場合、翻訳オプションENCODEによって指定されたエンコードが有効になります。翻訳オプションENCODEの詳細は、“[A.3.15 ENCODE\(データ項目のエンコードの指定\)](#)”を参照してください。

Windows系システムでは、翻訳オプションENCODEの省略値はENCODE(SJIS,SJIS)です。よって、ENCODING句と翻訳オプションの両方の指定が省略された場合、データ項目のエンコードはシフトJISになります。これは、旧バージョンと互換性があることを意味します。

### 翻訳オプションENCODE指定なし、または、ENCODE(SJIS,SJIS)指定の場合

```
01 A.
  02 A1  PIC X(10).          *> エンコードはSJISです。
  02 A2  PIC N(10).          *> エンコードはSJISです。
```

### 翻訳オプションENCODE(UTF8,UTF16)指定の場合

```
01 A.
  02 A1  PIC X(10).          *> エンコードはUTF8です。
  02 A2  PIC N(10).          *> エンコードはUTF16LEです。
```

### 翻訳オプションENCODE(UTF8,UTF32)指定の場合

```
01 A.
  02 A1  PIC X(10).          *> エンコードはUTF8です。
  02 A2  PIC N(10).          *> エンコードはUTF32LEです。
```

なお、翻訳オプションENCODEとENCODING句の強さの関係は、

明示指定されたENCODING句 > 翻訳オプションENCODE

となります。



### 注意

NetCOBOL V11より前のバージョンでは、翻訳オプションRCSによってアプリケーションの実行時コード系とデータ項目のエンコードを指定していました。翻訳オプションRCSは互換動作のために引き続き指定できますが、V11以降は翻訳オプションENCODEを使用してください。

## 6.2.4 実行時コード系

実行時コード系とは、NetCOBOLが作成した目的プログラムの属性で、“[6.2.5 資源](#)”で説明する実行時資源のコード系を決定する要因になります。実行時コード系は、翻訳オプションENCODEの指定に応じて決まります。

翻訳オプションENCODEと実行時コード系の関係を以下に示します。

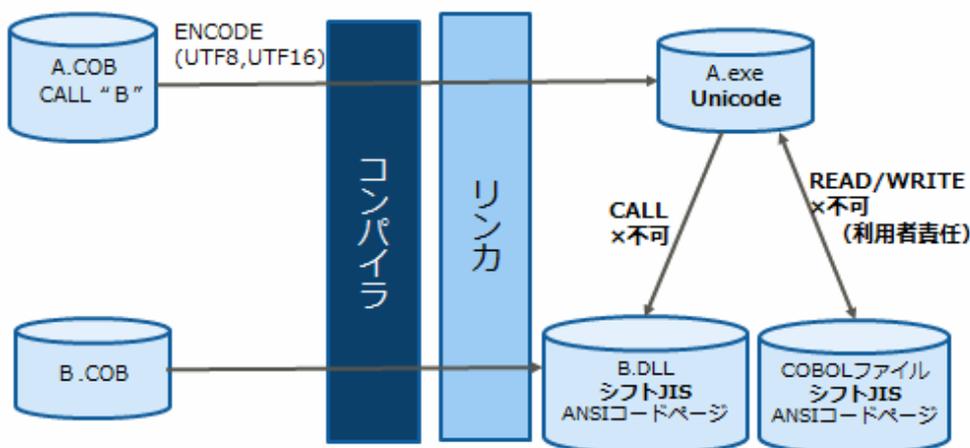
翻訳オプションENCODE	データ項目のエンコード		実行時コード系
	英数字項目	日本語項目	
指定なし	ASCII または シフトJIS	シフトJIS	ANSIコードページ または シフトJIS
ENCODE(SJIS,SJIS)	シフトJIS	シフトJIS	Unicode (注)
ENCODE(UTF8,UTF16[,LE/BE])	UTF-8	UTF-16	Unicode
ENCODE(UTF8,UTF32[,LE/BE])	UTF-8	UTF-32	Unicode

注: 翻訳オプションRCS(SJIS)を明示指定している場合は、シフトJISになります。

## 注意

ENCODEオプションを省略した場合、RCSオプションが有効になります。“A.3.44 RCS (実行時コード系の指定)”を参照してください。

なお、NetCOBOLでは実行単位内で実行時コード系(UnicodeとANSIコードページ/シフトJIS)は混在できません。例えば、以下の図のように実行時コード系が異なるDLLを呼び出した場合、実行時エラーとなります。



## 6.2.5 資源

ソースファイルおよび登録集などの翻訳資源はANSIコードページ、シフトJISまたはUnicode(UTF-8)で作成してください。

翻訳資源のコード系は翻訳オプションSCSで指定します。

## 注意

翻訳資源のコード系と翻訳オプションSCSのコード系を一致させてください。

翻訳資源のコード系	翻訳オプションSCSの指定方法
ANSIコードページ	SCS(ACP)または省略
シフトJIS	SCS(SJIS)または省略
Unicode(UTF-8)	SCS(UTF8)

- システムロケールが日本語の場合、SCS(ACP)とSCS(SJIS)は同義です。システムロケールに依存せず、翻訳資源のコード系をシフトJIS固定で扱う場合、SCS(SJIS)を指定します。

- ・ システムロケールが日本語以外の動作については、“[M.6 Windowsシステムにおけるシステムロケール](#)”を参照してください。
- ・ 翻訳オプションSCSの指定方法は、“[A.3.49 SCS \(ソースファイルのコード系\)](#)”を参照してください。

翻訳オプションSCSと翻訳オプションENCODEの組合せの可否を以下に示します。

	ENCODE(SJIS,SJIS)	ENCODE(UTF-8,UTF16)	ENCODE(UTF-8,UTF32)
<b>SCS(ACP)</b>	○	○	○
<b>SCS(SJIS)</b>	○(省略時)	○	○
<b>SCS(UTF8)</b>	×	○	○

○: 組合せ可能

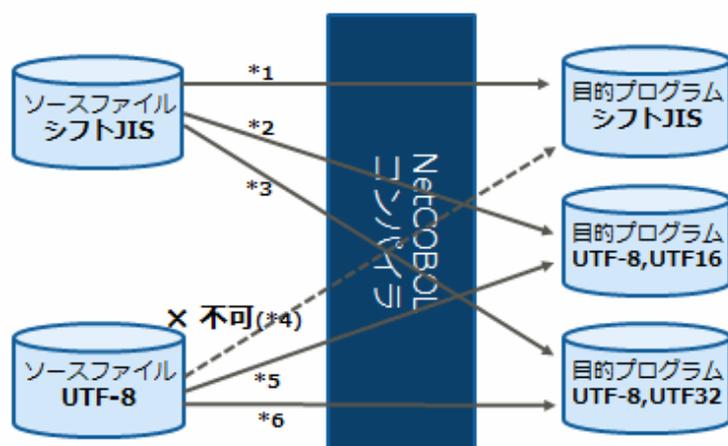
×: 組合せ不可。

### 注意

ENCODEオプションを省略した場合、RCSオプションが有効となります。

	RCS(ACP)	RCS(SJIS)	RCS(UTF16)
<b>SCS(ACP)</b>	○(省略時)	○	○
<b>SCS(SJIS)</b>	○	○	○
<b>SCS(UTF8)</b>	×	×	○

システムロケールが日本語以外の場合、翻訳オプションSCSと翻訳オプションRCS/ENCODEの組合せ可否については、“[M.6 Windowsシステムにおけるシステムロケール](#)”を参照してください。



\*1: 翻訳オプションは不要です。

\*2: 翻訳オプションSCS(SJIS),ENCODE(UTF8,UTF16[,LE/BE])を指定してください。

\*3: 翻訳オプションSCS(SJIS),ENCODE(UTF8,UTF32[,LE/BE])を指定してください。

\*4: 翻訳オプションSCS(UTF8),ENCODE(SJIS,SJIS)は指定できません。翻訳時エラーとなります。

\*5: 翻訳オプションSCS(UTF8),ENCODE(UTF8,UTF16[,LE/BE])を指定してください。

\*6: 翻訳オプションSCS(UTF8),ENCODE(UTF8,UTF32[,LE/BE])を指定してください。

### 実行時コード系がANSIコードページまたはシフトJISの場合

資源のコード系はすべてANSIコードページまたはシフトJISです。

	資源	入出力	コード系
翻訳時	翻訳リスト	出力	ANSIコードページまたはシフトJIS
リンク時	モジュール定義ファイル	入力	ANSIコードページまたはシフトJIS
実行時	実行用初期化ファイル エントリ情報ファイル クラス情報ファイル ODBC情報ファイル 印刷情報ファイル プリンタ情報ファイル ウィンドウ情報ファイル	入力	ANSIコードページまたはシフトJIS
	キー定義ファイル	入力	ANSIコードページまたはシフトJIS
	COBOLファイル(順/行順/相対/索引)	入出力	ANSIコードページまたはシフトJIS
	小入出力ファイル TRACE情報ファイル 汎用ログファイル	出力	ANSIコードページまたはシフトJIS
	メッセージを出力するファイル COUNT情報ファイル	出力	ANSIコードページまたはシフトJIS

### 実行時コード系がUnicodeの場合

資源のコード系は、以下のとおりです。

	資源	入出力	コード系
翻訳時	翻訳リスト	出力	ANSIコードページ、シフトJIS または Unicode(UTF-8) (注1)
リンク時	モジュール定義ファイル	入力	ANSIコードページシフトJIS または Unicode(UTF-8)
実行時	実行用初期化ファイル エントリ情報ファイル クラス情報ファイル ODBC情報ファイル 印刷情報ファイル プリンタ情報ファイル	入力	ANSIコードページ、シフトJIS または Unicode(UTF-8) (注2)
	ウィンドウ情報ファイル	入力	シフトJIS
	キー定義ファイル	入力	Unicode(UTF-8)
	COBOLファイル(順/行順/相対/索引)	入出力	Unicode(UTF-8、UTF-16 または UTF-32) (注3)
	小入出力ファイル	入出力	Unicode(UTF-8)
	TRACE情報ファイル 汎用ログファイル	出力	(注5)
	メッセージを出力するファイル COUNT情報ファイル	出力	ANSIコードページ、シフトJIS または Unicode(UTF-8) (注4)(注5)

注1: 翻訳オプションSCSによって指定されたコード系で出力されます。

注2: 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3: データ項目のエンコードによって決定します。[参照]“7.1.4 Unicodeデータの扱い”

注4: 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。[参照]“C.2.65 @MessOutFile(メッセージを出力するファイルの指定)”

注5: ファイルに入出力するコード系を指定することができます。[参照]“C.2.7 @CBR\_CODE\_SET(ファイルのコード系の指定)”



## 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なしでも入力できます。

## 6.3 言語要素

ここでは、文字コードに関する言語要素を解説します。なお、文字コードに関するコーディング上の注意点を“付録M 文字コードの留意点”にて解説していますので、本項と併せてお読みください。

本項のプログラム例では、環境部で以下の符号系名が宣言されているとみなします。

```
ALPHABET
U8   FOR ALPHANUMERIC IS UTF8
U16L FOR NATIONAL     IS UTF16LE
U16B FOR NATIONAL     IS UTF16BE
U32L FOR NATIONAL     IS UTF32LE
U32B FOR NATIONAL     IS UTF32BE.
```

### 6.3.1 異なるエンコード間の比較

COBOL文法書で定義されている「比較の規則」に従って、エンコードが異なるデータ項目どうしの比較は翻訳時にチェックされます。

```
WORKING-STORAGE SECTION.
77 DATA1 PIC N(10) VALUE SPACE ENCODING IS U16L.  *> UTF-16LE
77 DATA2 PIC N(10) VALUE SPACE ENCODING IS U32L.  *> UTF-32LE
:
IF DATA1 = DATA2 THEN DISPLAY "OK!!".           *> 翻訳時エラー
```

このような場合、作業用データ項目を用いて比較対象同士のエンコードを合わせてください。なお、作業用データ項目に値を転記する際、後述するMOVE文(書き方3)を利用すると便利です。

```
77 TEMP PIC N(10) VALUE SPACE ENCODING IS U32L.  *> 作業用データ項目を定義
:
MOVE CONV DATA1 TO TEMP.                         *> UTF-16LE → UTF-32LE
IF TEMP = DATA2 THEN DISPLAY "OK!!".             *> UTF-32LEどうしの比較
```

### 6.3.2 異なるエンコード間の転記

COBOL文法書で定義されている「転記の規則」に従って、エンコードが異なるデータ項目どうしの転記は翻訳時にチェックされます。

```
WORKING-STORAGE SECTION.
77 DATA1 PIC N(10) VALUE SPACE ENCODING IS U16L.  *> UTF-16LE
77 DATA2 PIC N(10) VALUE SPACE ENCODING IS U32L.  *> UTF-32LE
:
MOVE DATA1 TO DATA2.                             *> 翻訳時エラー
```

このような場合、MOVE文(書き方3)を使用します。

MOVE文(書き方3)は、送り出し側項目を受け取り側項目のエンコードに変換してから転記を実行します。

```
MOVE CONV DATA1 TO DATA2.    *> DATA1をUTF-32LEに変換してからDATA2に転記
```

このとき、エンコードが正しく変換できたか、変換したデータの長さが受け取り側項目の長さを超えてないか、をプログラム中で判定できるよう、特殊レジスタCONV-STATUS、CONV-SIZEを用意しています。

変換元のデータに不完全な文字や変換元のエンコードと異なるデータが現れた場合などは、特殊レジスタCONV-STATUSにエラーを通知します。

MOVE文(書き方3)では、異なる字類(英数字と日本語)間の転記も可能です。詳細については“COBOL文法書 6.4.28 MOVE文”を参照してください。

### 6.3.3 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、まだ旧バージョンの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。それらシステムと連携して業務を構築する場合、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。

NetCOBOLでは、データ項目に格納されている文字の範囲を判定するために、以下の2つの字類条件を用意しています。

字類条件	条件
BMP	データ項目の内容がISO/IEC 10646-1のBMP(Basic Multilingual Plane)で規定されている文字の場合に真となります。
UNICODE1	データ項目の内容がUNICODE1.1(ISO/IEC 10646-1:1993)で規定されている文字の場合に真となります。

これら字類条件を用いて要所で検査することでシステムの安定性が確保できます。

たとえば、JIS2004固有文字が含まれていないかは、以下のとおり検査します。

```
WORKING-STORAGE SECTION.  
77 DATA1 PIC N(5) ENCODING IS U32L.  
   :  
   IF DATA1 IS NOT UNICODE1 THEN  
       DISPLAY "JIS2004固有文字または不当なコードが含まれています."  
   END-IF.
```

## 6.4 UTF-32エンコードを使用できない機能

以下の機能では、UTF-32エンコードのデータ項目は使用できません。翻訳エラーとなります。

- ・ スクリーン機能
- ・ 報告書作成機能

## 6.5 実行時の注意点

### 6.5.1 フォントについて

実行時コード系がUnicodeの場合、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、“[C.2.58 @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、“C.2.71 @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イス名に、Unicodeに対応したフォントを指定してください。

指定方法は、“C.2.69 @PrinterFontName(印刷ファイルで使用するフォントの指定)”および、“8.1.13 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

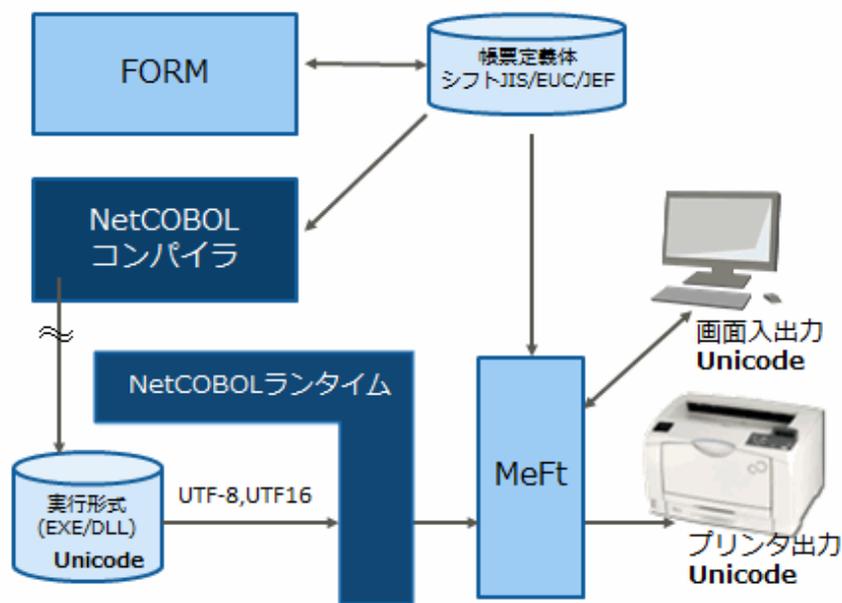
プリンタ情報ファイルおよびフォントテーブルに、Unicodeに対応したフォントを指定してください。

指定方法は、“MeFtユーザーズガイド”および、“8.1.13 フォントテーブル”を参照してください。

## 6.6 関連製品連携

### 6.6.1 FORM、MeFt

FORMおよびPowerFORMを使用して作成した画面帳票定義体を使用できます。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可能です。また、実行時にMeFtが自動で実行時のコード系に変換するため、定義体のコード系は、シフトJIS、EUC、EBCDIC-JEFのどれでも利用できます。



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。“8.5 表示ファイル(帳票印刷)の使い方”および“第9章 画面を使った入出力”を参照してください。

#### 英数字項目に格納できる文字について

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のみ格納できます。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、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 "PRT".  
:  
FILE SECTION.  
FD IO-FILE.  
  COPY 帳票定義体名 OF XMDLIB.  
(01 帳票レコード名.          ) (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。

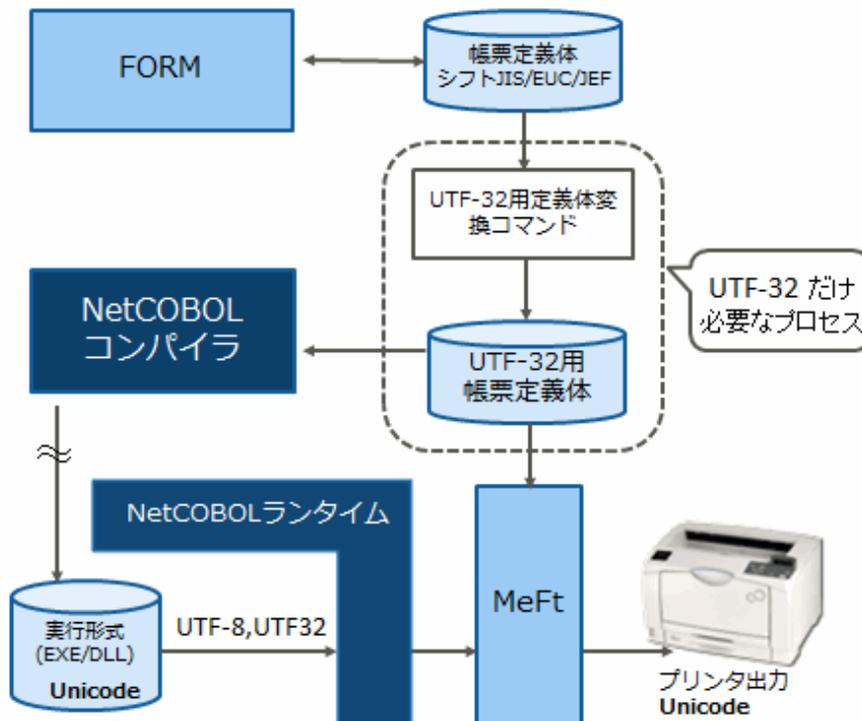
## エンコードUTF-16のサロゲートペアについて

入出力するデータのエンコードが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ユーザーズガイド”をご確認ください。

## エンコードUTF-32の利用について

帳票定義体はエンコードがUTF-32の場合でも使用できます。

出力する日本語項目のエンコードがUTF-32の場合、UTF-32用定義体変換コマンドを使用してUTF-32用の帳票定義体を作成する必要があります。



UTF-32用定義体変換コマンドの詳細については、“[J.6 UTF-32用定義体変換コマンド](#)”を参照してください。

なお、出力した帳票の確認やエンハンスなどで帳票定義体を更新する局面がありますが、UTF-32用帳票定義体はFORMおよびPowerFORMを用いて直接更新することはできません。このような場合、変換元の帳票定義体を更新してから、再度、UTF-32用定義体変換コマンドで変換してください。

### 注意

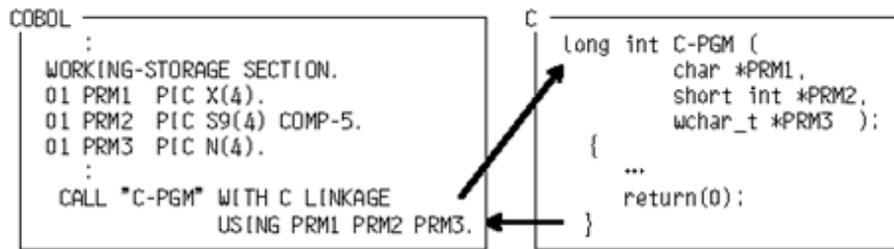
画面定義体は、エンコードUTF-32の場合には使用できません。

## 6.6.2 他言語間結合

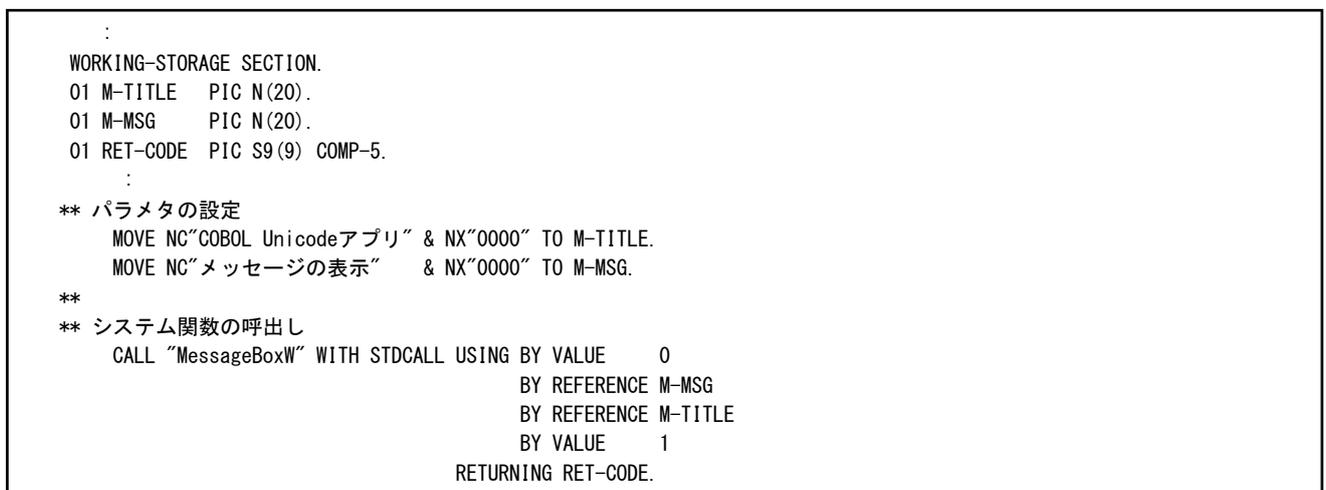
他言語間は他言語とインタフェース(文字コード)を合わせる必要があります。多くの他言語はUTF-32をサポートしていないため、実行時コード系がUnicodeの場合、UTF-8またはUTF-16を使用して結合します。

## 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型との間でデータの送受もできます。

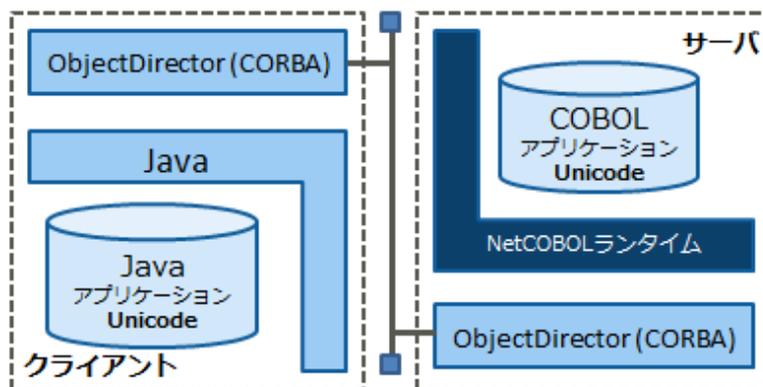


Windowsシステム関数の呼出しもできます。WindowsはUnicodeで動作するシステム関数(関数名の末尾がW)を用意しています。この関数との間で文字データを送受する場合は、日本語項目を使用してください。



## Java

Javaと連携する場合、Jアダプタクラスを利用しないときは、UTF-8またはUTF-16を使用してください。



Jアダプタクラスを利用する場合は、“Jアダプタクラスジェネレータ ユーザーズガイド”を参照してください。

## 6.6.3 他のファイルシステム

### Btrieveファイル

実行時コード系がUnicodeの場合、以下の点に注意してください。

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)にリトルエンディアンの日本語項目が含まれる場合、その項目はリトルエンディアンのままで比較処理されます。その結果、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。

## 6.6.4 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、NetCOBOLがサポートするエンコードが利用できるかどうかは、プリコンパイラの仕様に依存します。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。

なお、Symfoware Serverについては、NetCOBOLがサポートするエンコードを全てサポートしています。使用方法については、Symfoware Serverのマニュアルを参照してください。

また、他社のプリコンパイラについても、ALPHABET句やENCODING句を明示指定せずに、翻訳オプションENCODEによってデータ項目のエンコードを制御することで、連携が可能になる場合があります。

## 6.6.5 リモートデータベースアクセス(ODBC)

---

エンコードUTF-32の日本語項目を使用する場合は、実行時にUnicodeデータ型(UTF-16)に変換してマッピングします。

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]“15.2.11 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)またはSCS(ACP)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。

## 6.6.6 Interstage Business Application Server

---

Interstage Business Application Serverでは、サーバアプリケーションのパラメタの定義をCOBOL登録集に記述しますが、パラメタとして記述する基本項目にENCODING句を指定できません。指定した場合、COBOL実行基盤インターフェースの生成に失敗します。また、COBOL実行基盤インターフェースソースファイルの翻訳には、翻訳オプションENCODEを指定して日本語項目のエンコードをUTF-32にすることはできません。

## 6.6.7 Interstage Application Server

---

CORBAサービスおよびイベントサービスのアプリケーションを作成する場合、翻訳オプションENCODEを指定して日本語項目のエンコードをUTF-32にすることはできません。

## 6.6.8 Interstage Job Workload Server

---

Interstage Job Workload Serverでは、バッチアプリケーションのパラメタの定義をCOBOL登録集に記述しますが、パラメタとして記述する基本項目にENCODING句を指定できません。指定した場合、COBOL実行基盤インターフェースの生成に失敗します。また、COBOL実行基盤インターフェースソースファイルの翻訳では、翻訳オプションENCODEを指定して日本語項目のエンコードをUTF-32にすることはできません。

# 第7章 ファイルの処理

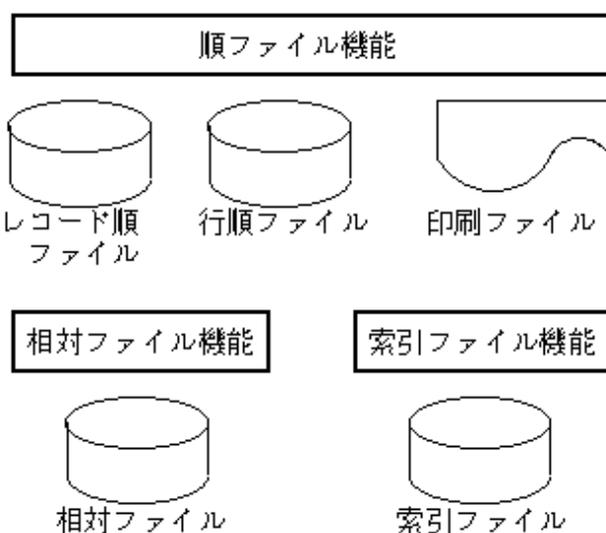
本章では、ファイルからデータを読み込んだり、ファイルにデータを書き出したりする処理、COBOLファイルユーティリティの使い方および他のファイルシステムの使用法について説明します。

## 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

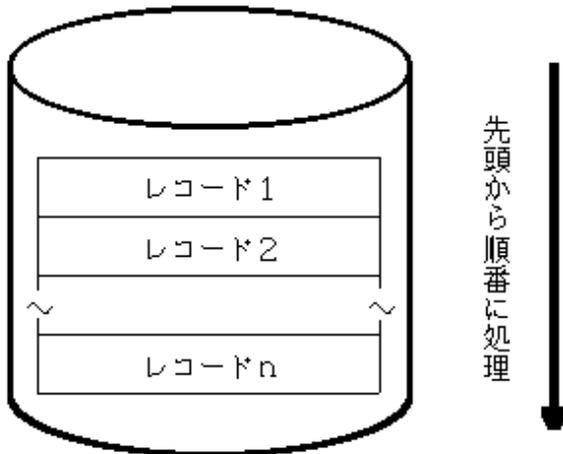
	ファイルの種類				
	レコード順ファイル	行順ファイル	印刷ファイル	相対ファイル	索引ファイル
レコードの処理	レコードが格納されている順番			相対レコード番号	レコードキーの値
使用できる媒体	ハードディスク(*) フロッピーディスク	ハードディスク(*) フロッピーディスク	印刷装置	ハードディスク(*) フロッピーディスク	ハードディスク(*) フロッピーディスク
利用例	データ退避 作業ファイル	テキストファイル	データの印刷	作業ファイル	マスタファイル

\* : 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

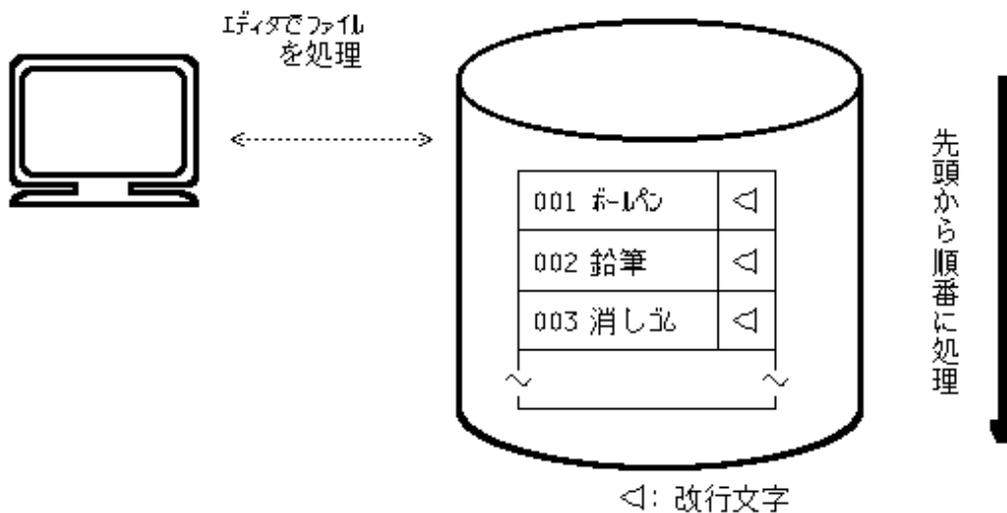
レコード順ファイルでは、ファイルの先頭レコードからファイルに書き出した順番でレコードを読んだり、更新したりできます。レコード順ファイルは、最も簡単に扱うことのできるファイルで、データを順次蓄積する場合や、大量データを保存する場合などに効果的です。



## 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。

- ・ 行順ファイルの文字コードは、ひとつの表現形式で統一しなければなりません。

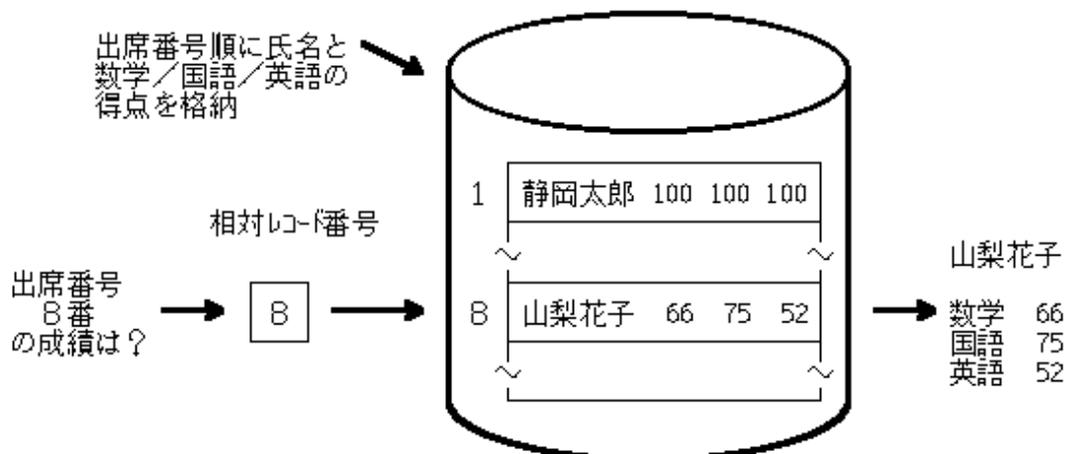
## 印刷ファイル

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。

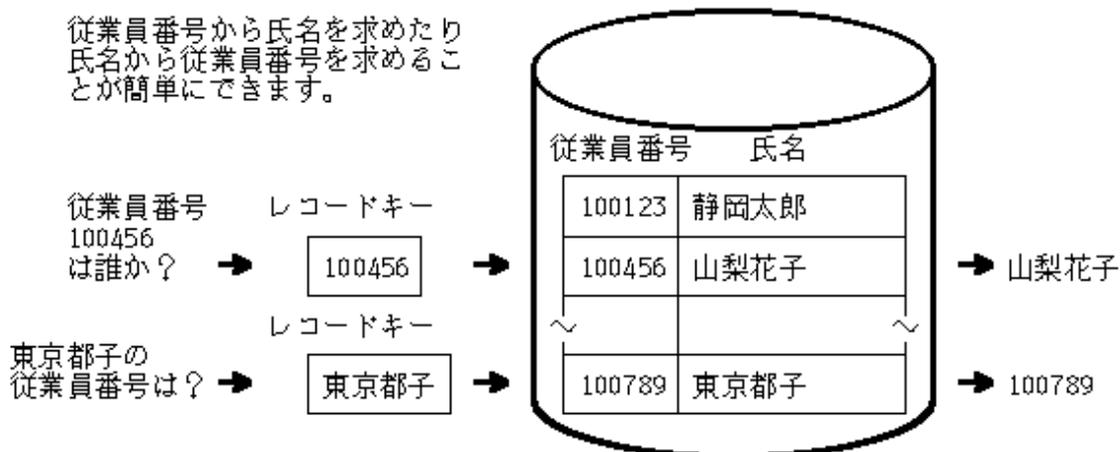
## 相対ファイル

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します。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
ファイルの創成	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					
		創成	拡張	挿入	参照	更新	削除
レコード順ファイル	順呼出し	○	○	×	○	○	×
行順ファイル	順呼出し	○	○	×	○	×	×
印刷ファイル	順呼出し	○	○(注)	×	×	×	×
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

注) 創成と同じ動作になります。

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。

## 7.1.4 Unicodeデータの扱い

COBOLファイルでは、基本的にエンコードは何も加工せずに入出力しますが、Unicodeの場合、印刷ファイルおよび表示ファイル(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。

以下にUnicodeの場合のCOBOLファイルに関する注意点をまとめます。

### ファイル識別名

ファイル識別名にデータ名を指定している場合で、かつ、そのデータ名が複数のエンコードを含む集団項目だった場合、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 (UTF-8とUTF-16が混在)
  02 F-PATH PIC X(11) ENCODING IS U8.
  02 F-NAME PIC N(4) ENCODING IS U16L VALUE NC"ファイル".
  :
  MOVE "d:¥cobfile¥" 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エンコードを統一してください。

### 行順ファイル

行順ファイルは、各種テキストエディタによる表示・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エンコード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エンコードを統一しなければなりません。以下の例のように字類またはエンコードが混在する場合、翻訳時エラーが出力されるため、用途に合わせて字類およびエンコード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 (UTF-16とUTF-32が混在)
01 OUT-REC.
  02 REC-ID PIC N(4) ENCODING IS U16L.
  02 REC-DATA PIC N(20) ENCODING IS U32L.
```

## 索引ファイル

索引キーに集団項目を指定し、かつ、その集団項目がリトルエンディアンの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) ENCODING IS U8.  
        03 KEY2      PIC N(8) ENCODING IS U16L.  
    02 IX-DATA      PIC X(80).
```

日本語項目には、データがUTF-16のリトルエンディアン、つまり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。

## 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と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 INFIL	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“DEPENDENT 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  
DEPENDENT 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 DEPENDENT 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.
```

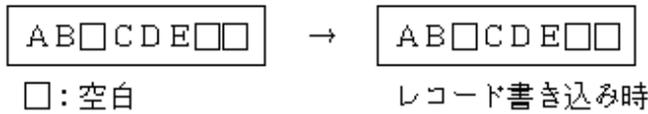


### 注意

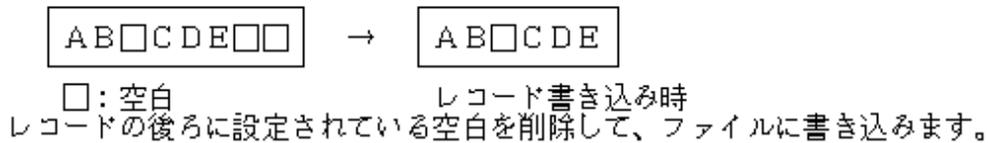
- すでに存在するファイルに対して創成処理を行った場合、そのファイルは新しく作り直され、元の内容は失われます。

- 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。
- 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]“C.2.56 @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]“C.2.56 @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

- 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。

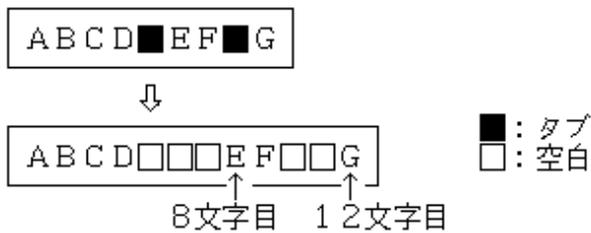
- 読み込んだレコードの大きさがレコード長より大きいときには、1回のREAD文の実行でレコード長と同じ長さのデータがレコード領域に設定されます。次のREAD文の実行では、同じレコードのデータの続きから、データがレコード領域に設定されます。設定するデータがレコード長より小さい場合には、レコード領域の残りの領域に、空白が設定されます。

## 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。

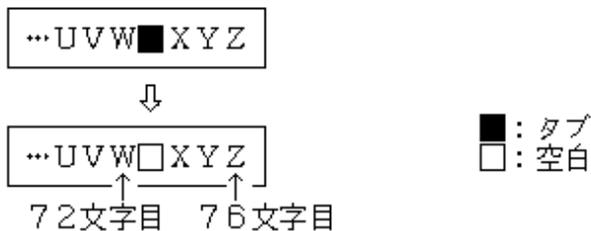
### 72文字以内にタブが存在する場合

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(改頁)	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。



```

    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。

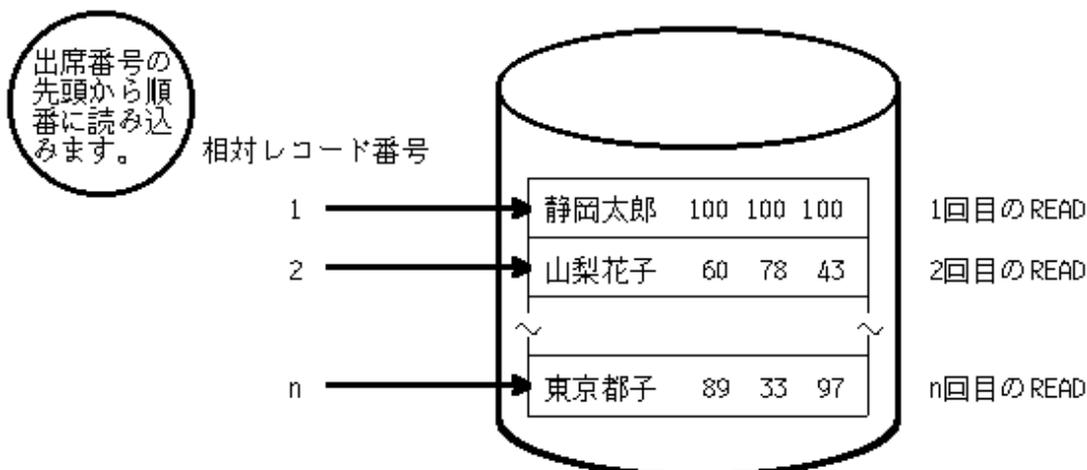
#### 動的呼出し法(DYNAMIC)

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。

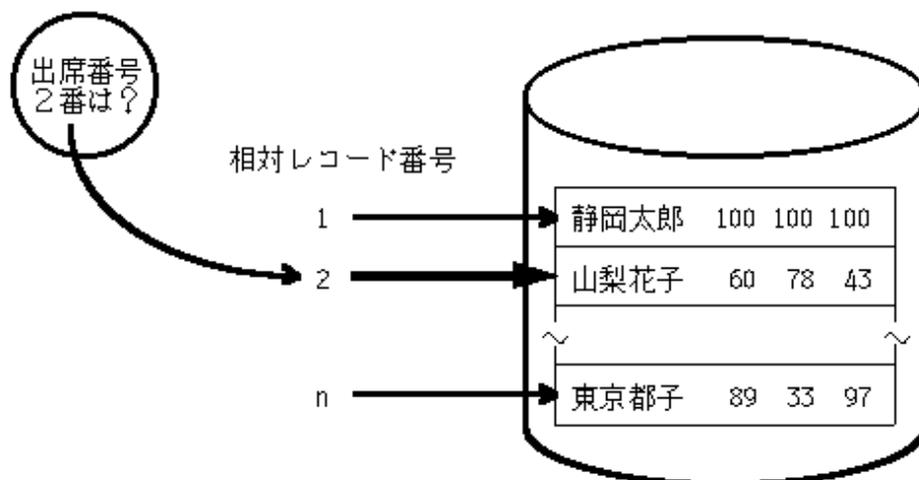
#### 順呼出し

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).  
   02 数学      PIC 9(3).  
   02 英語      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.
```

## 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〔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。



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

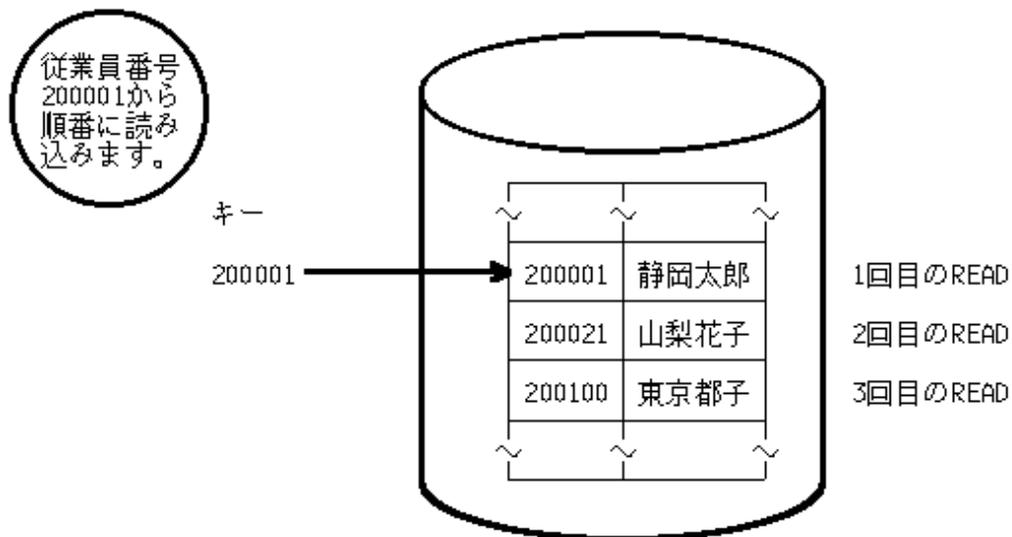
ACCESS MODE句に以下のアクセス形態のどれかを指定します。

- ・ 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。

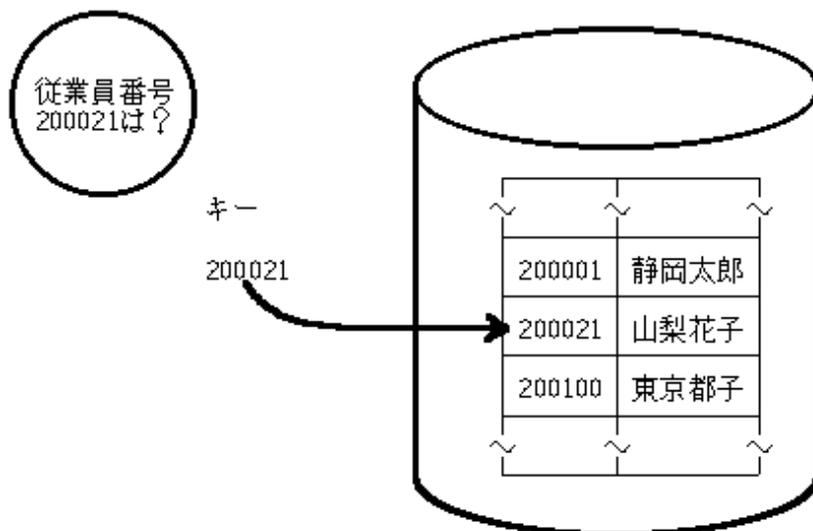
### 順呼出し

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.

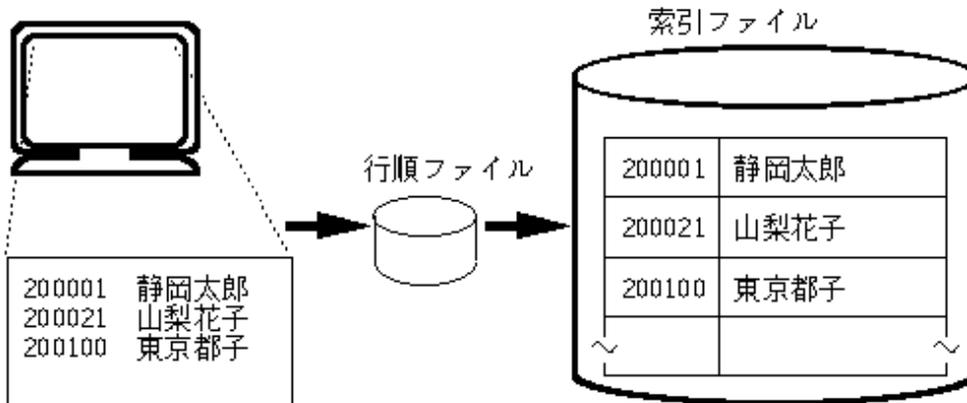
## 注意

- 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。

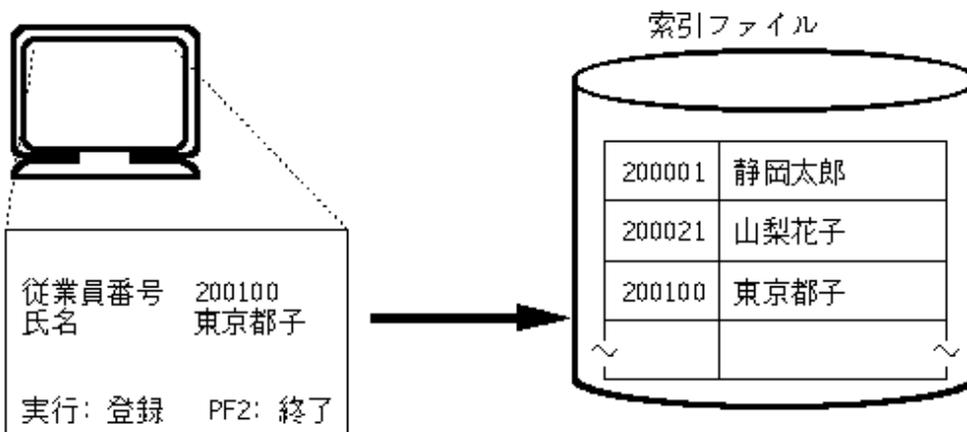
## 参考

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。

- 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。



- 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。
- 副キーの内容を変更することはできませんが、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検出)
- INVALID KEY指定(無効キー条件発生検出)
- FILE STATUS句(入出力状態のチェックによる入出力エラー検出)
- 誤り処理手続き(入出力エラー検出)

### 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D 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ため、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、“22.1 診断機能”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

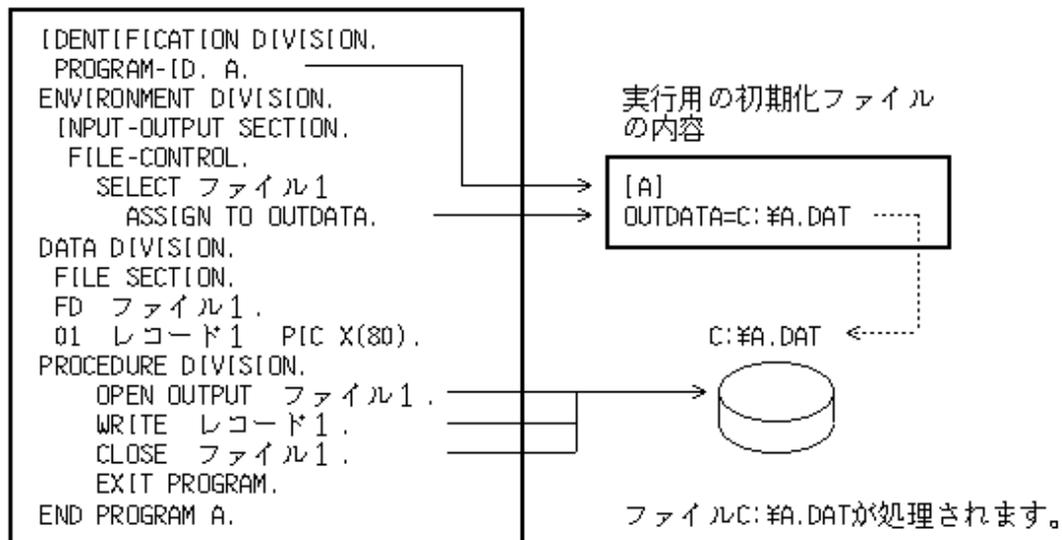
プログラムの実行時に入出力処理の対象となるファイルの決定方法は、ファイル管理記述項のASSIGN句の記述内容によって異なります。ASSIGN句の記述内容とファイルの関係を以下に示します。

#### 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、“5.3.2 実行環境情報の設定方法”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



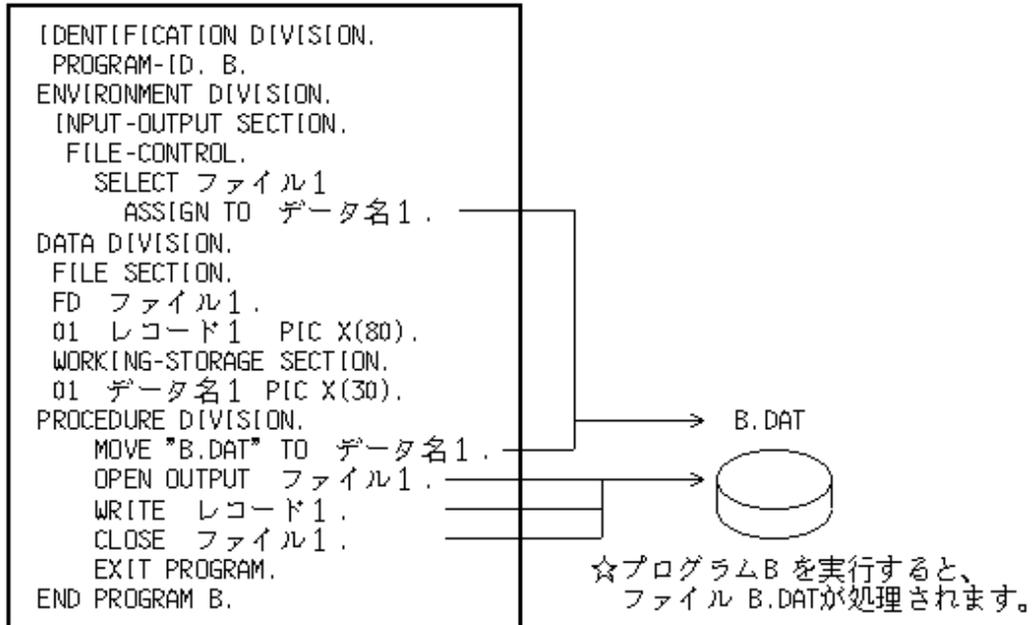
## 注意

- 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

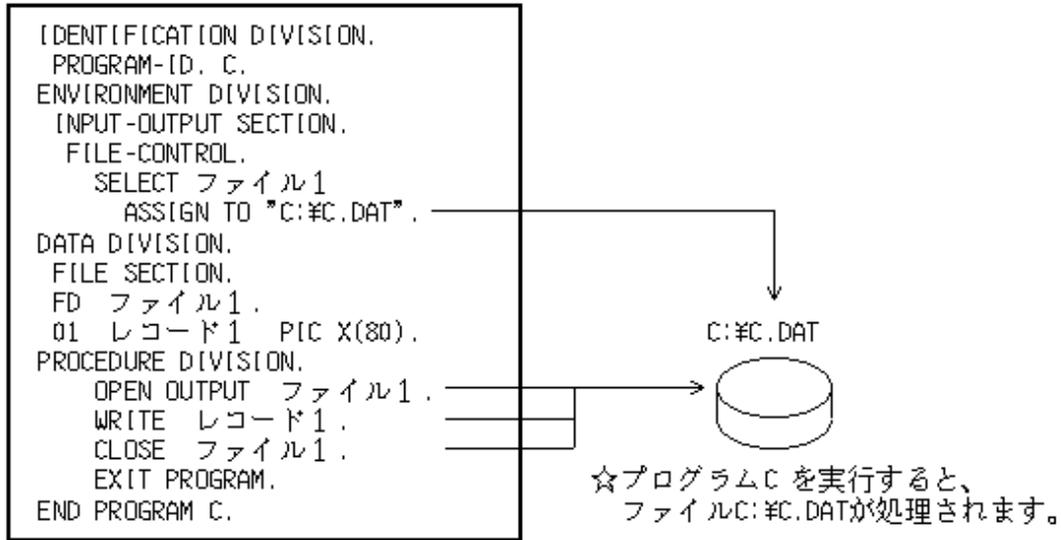
- プログラムに記述されたファイル名が相対パス名の場合、カレントフォルダを先頭に付加したファイルが入出力処理の対象となります。
- データ名の内容が空白の場合、ファイルの割当てエラーとなります。

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。

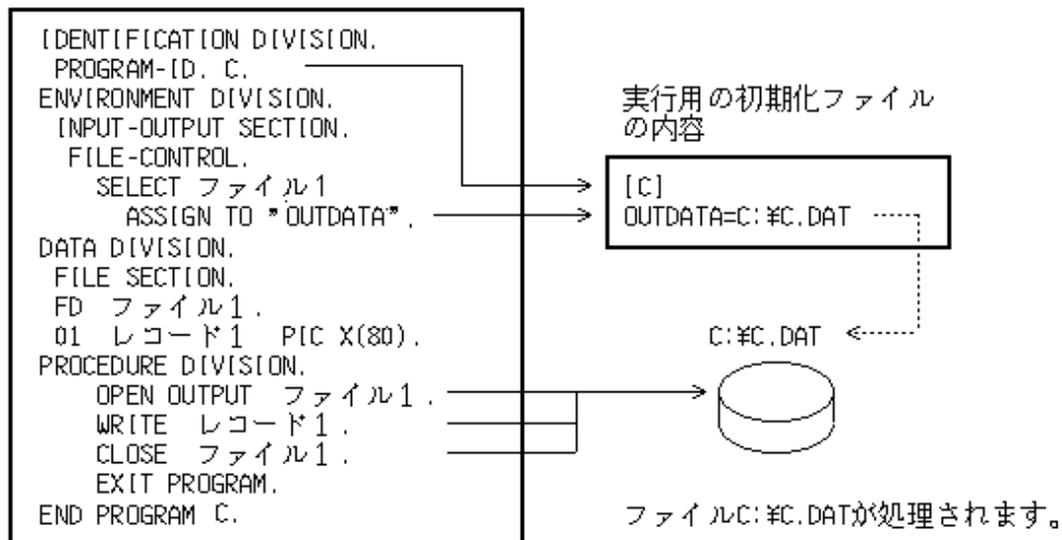
#### 翻訳オプションFILELIT(ENV)指定時

ファイル識別名定数を環境変数名とし、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。



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



プログラムCを実行すると、ファイルC:¥C.DATが処理されます。

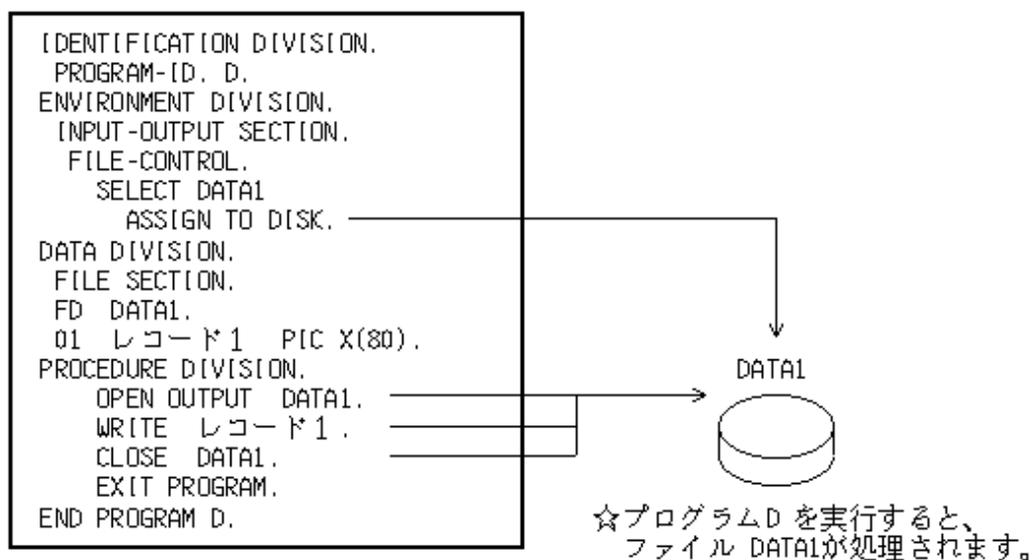
### 注意

翻訳オプションFILELIT(ENV)が有効な場合、環境変数名として使用できない記号類をファイル識別名定数に指定しないでください。

### 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





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

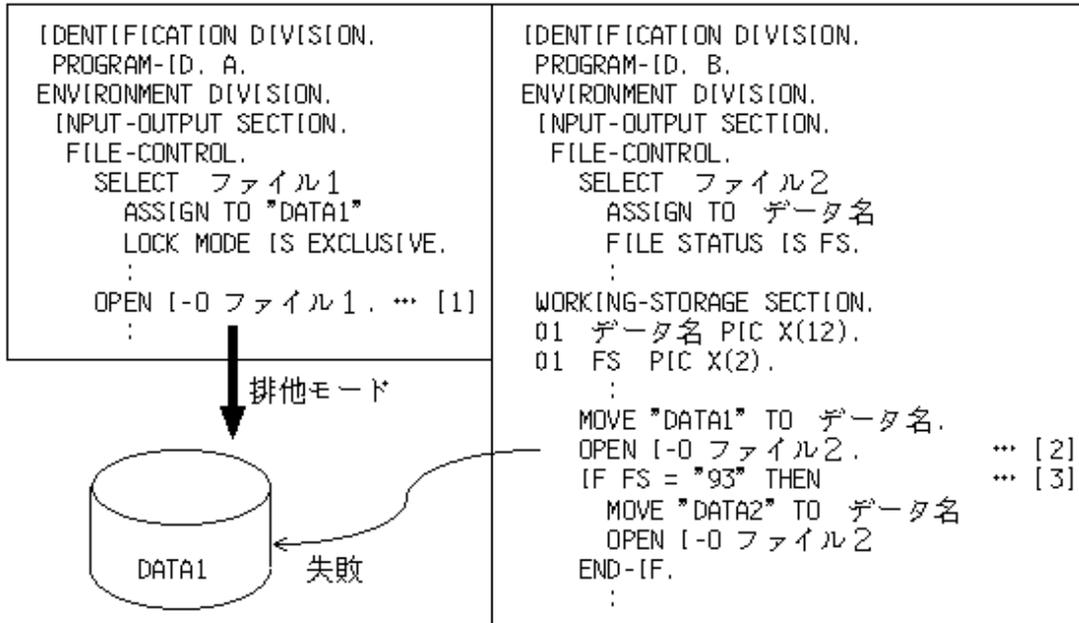
OPEN文のモード	入力INPUT		入出力I-O		拡張EXTEND		出力OUTPUT	
OPEN文のWITH LOCK 指定	なし	あり	なし	あり	なし	あり	なし	あり
ファイルのモード	共用	排他	排他	排他	排他	排他	排他	排他

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



- [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		
	記述なし	WITH LOCK	WITH NO LOCK	記述なし	WITH LOCK	WITH NO LOCK
READ文の記述						
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。

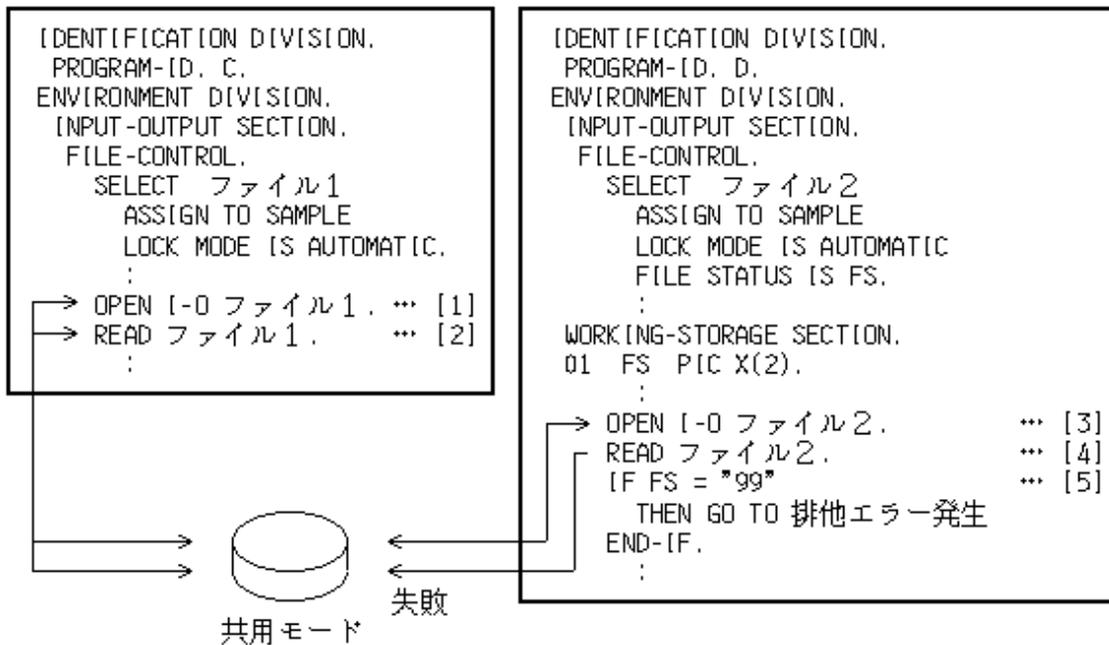
LOCK MODE句にMANUALを指定したファイルの場合

- 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

既存ファイルに対する拡張処理では、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)に対する拡張処理では、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。不定ファイルに対する参照処理では、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

既存ファイルに対する更新/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。不定ファイルに対する更新/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
  - JMPCINT3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、“[C.2.78 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	<ul style="list-style-type: none"><li>レコード順ファイル</li><li>行順ファイル</li></ul>
ASSIGN句の指定	<ul style="list-style-type: none"><li>ファイル識別名</li><li>ファイル識別名定数</li></ul>

	<ul style="list-style-type: none"> <li>・ データ名</li> <li>・ DISK</li> </ul>
--	--

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

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。

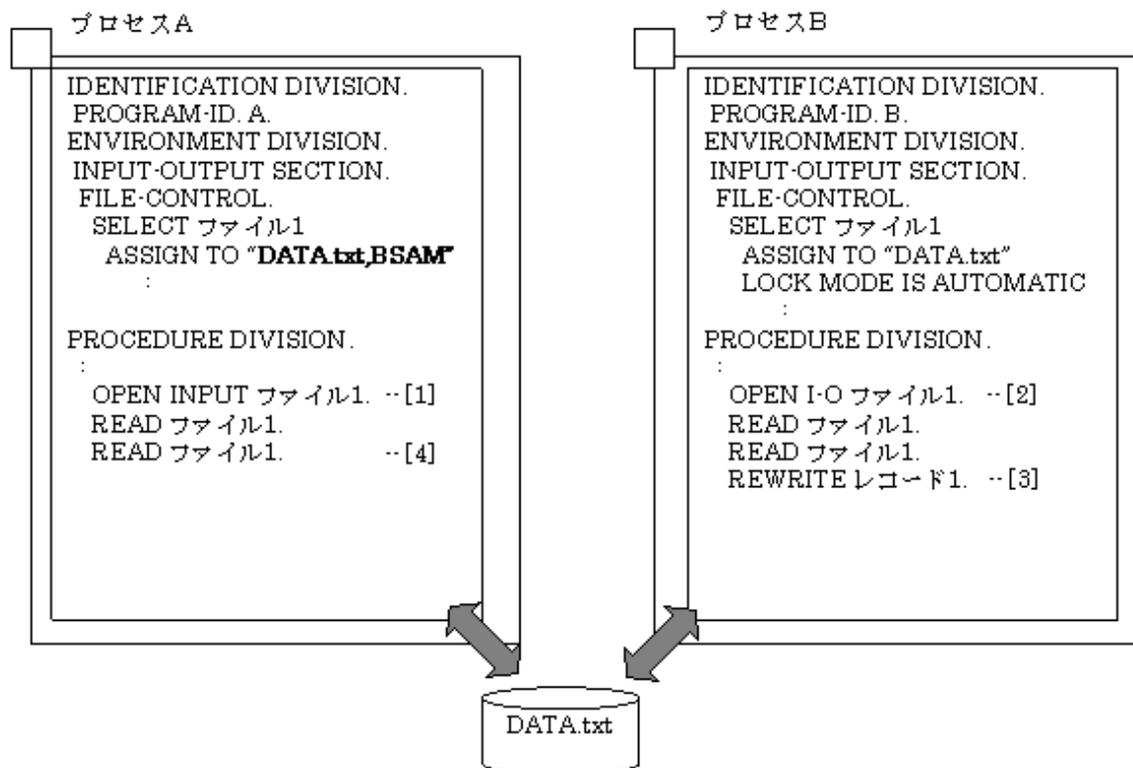
[参照]“ファイル共用時の注意事項”

### 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。

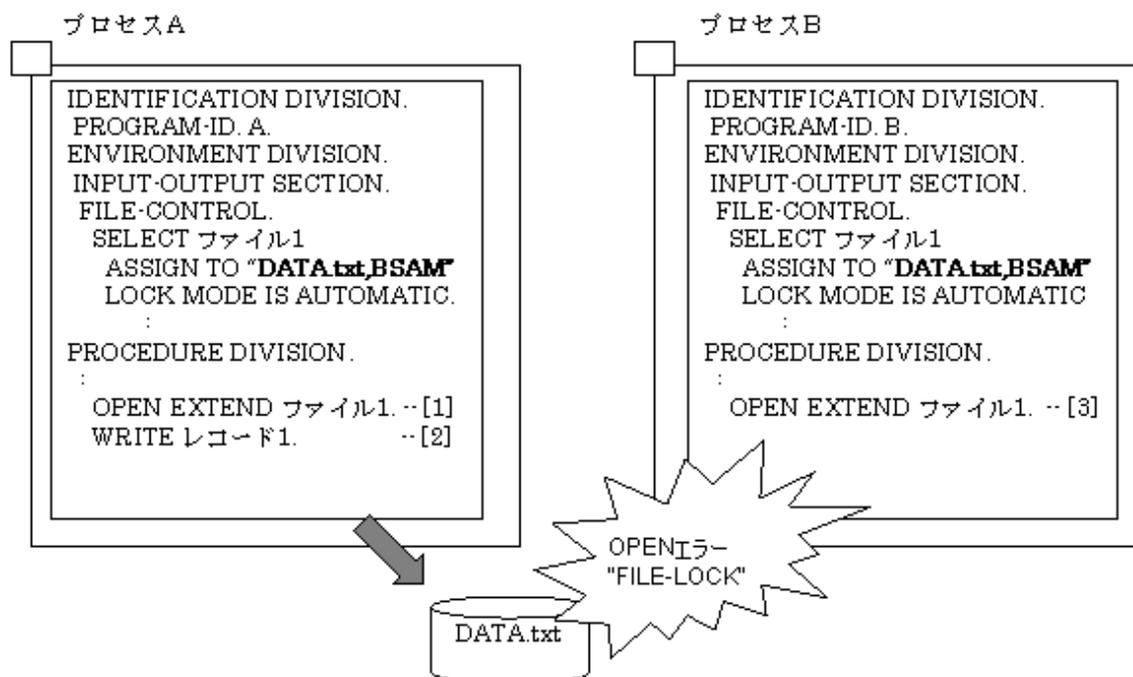
#### 例

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。



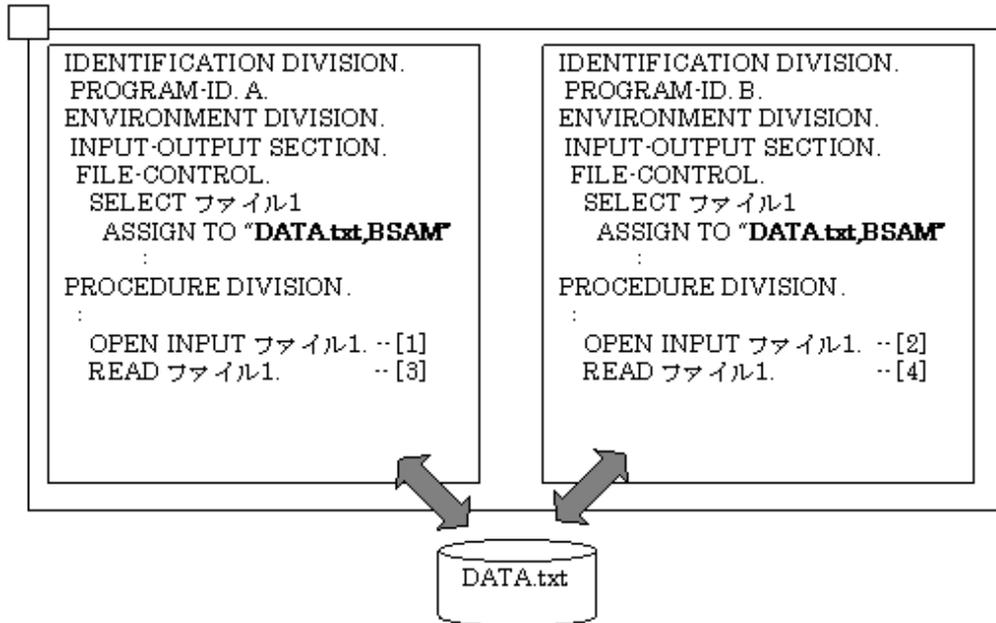
- [1] プロセスA：共用モード、INPUT指定でファイルを開きます。(ファイルの高速処理)
- [2] プロセスB：共用モード、I-O指定でファイルを開きます。
- [3] プロセスB：2件目のレコードを更新します。
- [4] プロセスA：2件目のレコードを読み込みます。ここで、更新前のデータが読み込まれる可能性があります。

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有する場合、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。
- [4] プログラムB：1件目のレコードを読み込みます。ここで、2件目以降のレコードデータが読み込まれる、または、ファイル終了条件が発生する場合があります。

## 7.7.5 ファイル追加書き

OPEN OUTPUT文の実行で、既に存在するファイルにレコードを追加することができます。以下に本機能の使い方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、“MOD”を指定します。

ファイル識別名=ファイル名,MOD



注意

COBOLファイルのレコード順ファイルのみに有効となります。他のファイル編成および他のファイルシステムに指定することはできません。

## 7.7.6 ファイルの連結

複数のファイルを連結して、レコードを参照、更新することができます。以下に本機能の使い方について説明します。

使い方

ファイル識別名に、“CONCAT(連結するファイル名の並び)”を指定します。

```
ファイル識別名=, CONCAT(ファイル名1 ファイル名2..)
```

## 注意

- ファイル名は半角空白で区切ります。
- ファイル名に空白文字を含む場合は、そのファイル名を二重引用符(")で囲んでください。
- COBOLファイルのレコード順ファイルのみに有効となります。他のファイル編成および他のファイルシステムに指定することはできません。
- OUTPUT指定またはEXTEND指定のOPEN文は実行時にエラーになります。
- 同一ファイルが複数指定された場合、別ファイルを指定した場合と同じ動作となります。
- ファイル識別名に1024バイトを超える文字列を指定することはできません。したがって、連結可能なファイル数は、ファイル連結機能で指定するファイル名の長さに依存します。  
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。

```
,, CONCAT(ファイル名 ...ファイル名 n)  
└──────────────────────────────────┘  
1024 バイト以内
```

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。

## 7.7.7 ダミーファイル

ダミーファイルは、実体が存在しない架空のファイルです。

入出力文を実行する対象がダミーファイルの場合、物理的なファイル操作は行われません。例えば、OUTPUTモードのOPEN文を実行した場合、通常はOPEN文が成功するとファイルが生成されて書込み可能な状態になりますが、ダミーファイルを指定した場合、OPEN文は成功しますが、ファイルは生成されません。

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制し、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。
  - － (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
  - － 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ファイル进行处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。
  - － ファイルの複写/移動/削除
  - － ファイル構造の変換
  - － 索引ファイルの再編成/復旧/属性の表示
- ・ COBOLファイルのレコードの操作(表示/編集/整列など)を行う。

これらの操作は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。

コマンドを使用する方法は、“[J.4 COBOLファイルユーティリティコマンド](#)”を、関数を使用する場合は“[H.3 COBOLファイルユーティリティ関数](#)”を参照してください。

以下は、COBOLファイルユーティリティの操作一覧と、NetCOBOLが提供するファイルユーティリティ(GUI)のコマンドメニュー、コマンド、および関数の対応表です。

操作		ファイルユーティリティ(GUI)のコマンドメニュー	コマンド	関数
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。	変換	ファイル変換コマンド (cobfconv)	ファイル変換関数
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。	ロード	ファイルロードコマンド (cobfload)	ファイルロード関数
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。	アンロード	ファイルアンロードコマンド (cobfulod)	ファイルアンロード関数
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。	表示	ファイル表示コマンド (cobfbrws)	-
印刷	任意のCOBOLファイルのレコードを印刷します。	印刷	-	-
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。	編集	-	-

操作		ファイルユーティリティ(GUI)のコマンドメニュー	コマンド	関数
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。	拡張	-	-
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。	整列	ファイル整列コマンド(cobfsort)	ファイル整列関数
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。	属性	ファイル属性コマンド(cobfattr)	-
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。	復旧	ファイル復旧コマンド(cobfrcov)	-
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	ファイル再編成コマンド(cobfreog)	ファイル再編成関数
複写	ファイルを複写します。	複写	-	ファイルコピー関数
削除	ファイルを削除します。	削除	-	ファイル移動関数
移動	ファイルを別の場所に移動します。	移動	-	ファイル削除関数

## 注意

- 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

- 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。
- 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。
- 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。

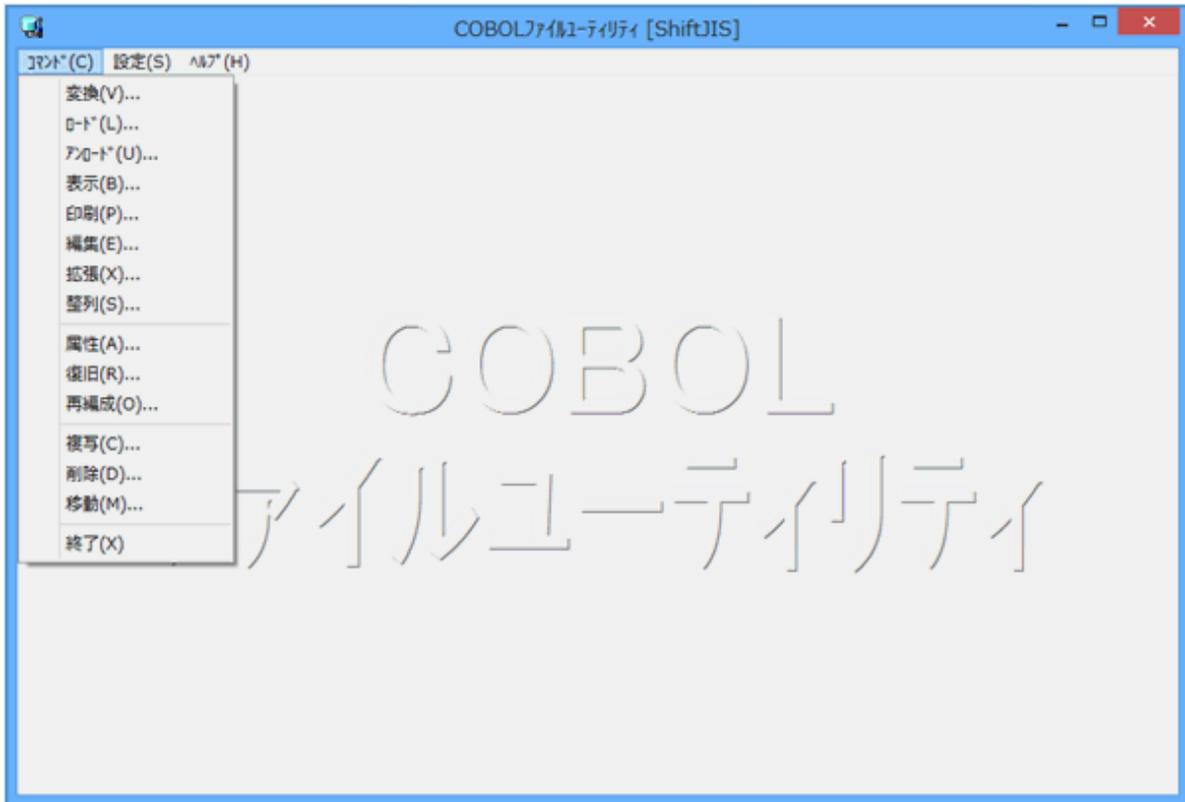
## 2. COBOLファイルユーティリティの起動

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。
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。

コマンド名	機能
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

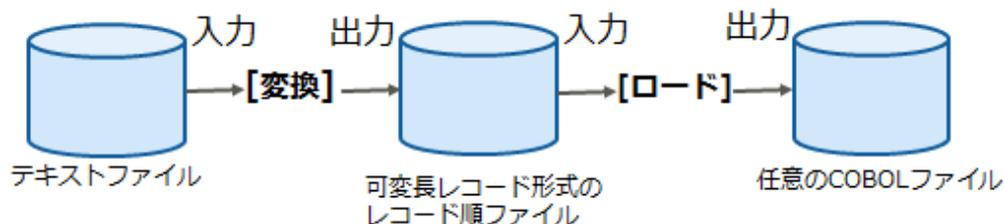
COBOLファイルユーティリティは、以下に示す機能を備えています。

- ・ 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]』



#### 注意

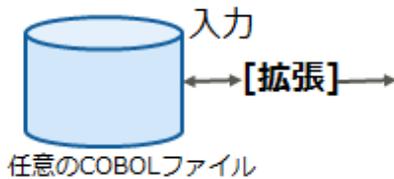
拡張指定の[ロード]コマンドでエラーとなった場合、バックアップのチェックボックスを選択していないと、既存ファイルの内容は失われます。

#### 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]』



→レコードを1件ずつ追加します。

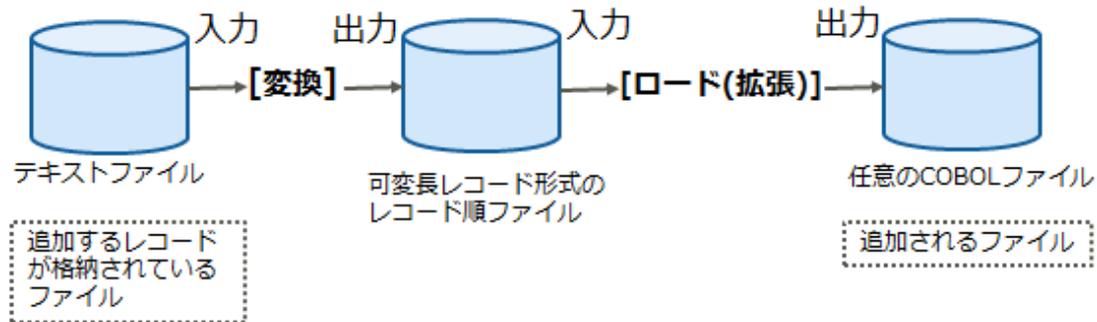
メニューバー	終了	拡張ウィンドウを終了します。
	表示	次に表示するレコードをどのように指示するかを選択します。
	編集	表示中のレコードを操作します。
	オプション	レコード操作画面の入力方法(16進入力/文字入力)の指定と、フォントの指定を行います。
	ヘルプ	拡張ウィンドウのヘルプを表示します。
ボタン	キー	読み込み順序を変更する場合に選択します。読み込み順序を変更すると、画面に表示されるレコードの順番は、選択したレコードキーの順番に依存するようになります。
	追加	画面に表示しているレコードをファイルに追加する場合に選択します。
	変長	画面に表示しているレコードの長さを変更する場合に選択します。
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:46/46	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

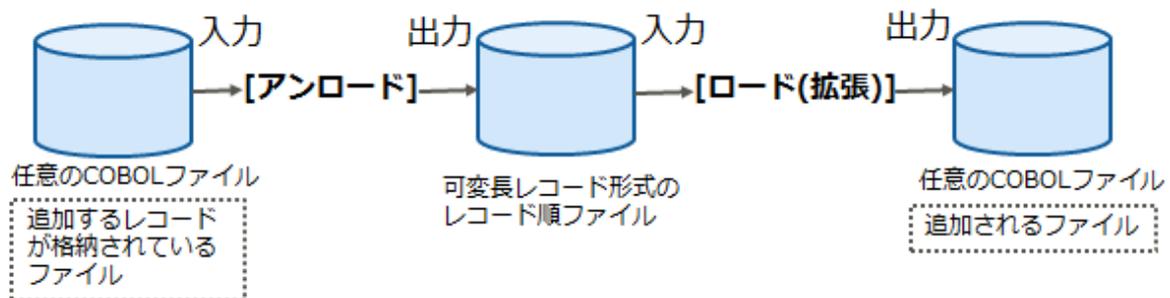
日本語文字などの表示できない文字は、ピリオドに置き換えられて表示されます。詳細は、ヘルプを参照してください。

## 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)]]』

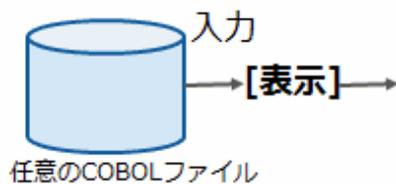


または、



## 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]』



→ 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:46/46	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。

## 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]』





→ 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。
	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:46/46	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。

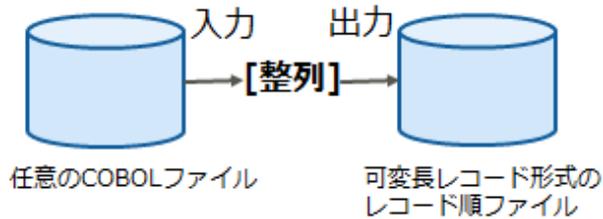
“0123456789ABCDEF”の 下の文字列	レコードデータを1バイト表記の文字で表示しています。
------------------------------	----------------------------

### 注意

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]』

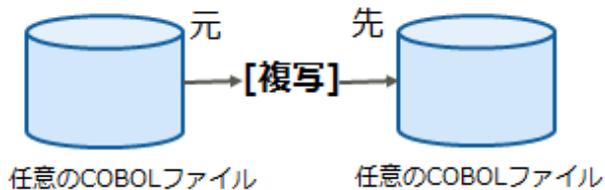


#### 7.8.3.6 ファイルの操作

COBOLファイルを複写、削除および移動することができます。

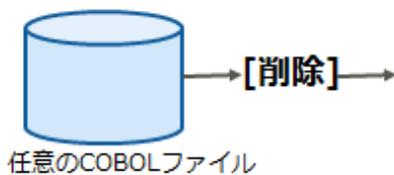
##### ファイルの複写

COBOLファイルを複写します。『[複写]』



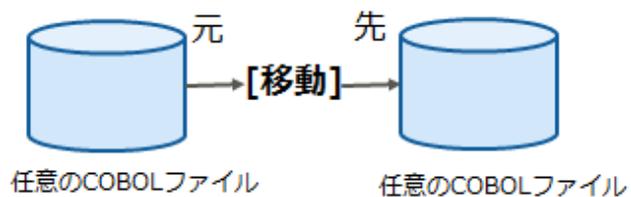
##### ファイルの削除

COBOLファイルを削除します。『[削除]』



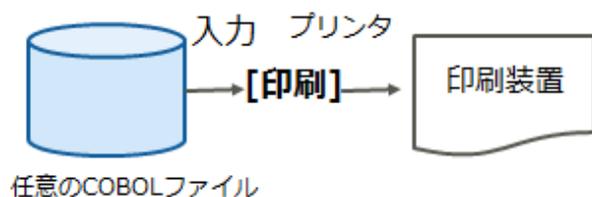
##### ファイルの移動

COBOLファイルを移動します。『[移動]』



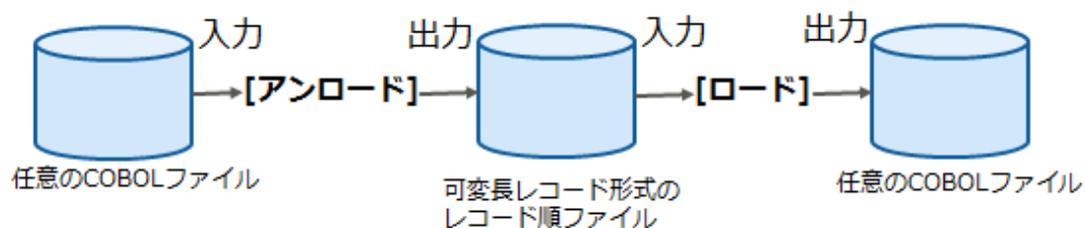
### 7.8.3.7 ファイルの印刷

COBOLファイルの内容を印刷することができます。『[印刷]』



### 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]』

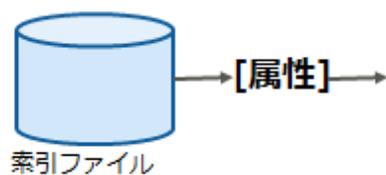


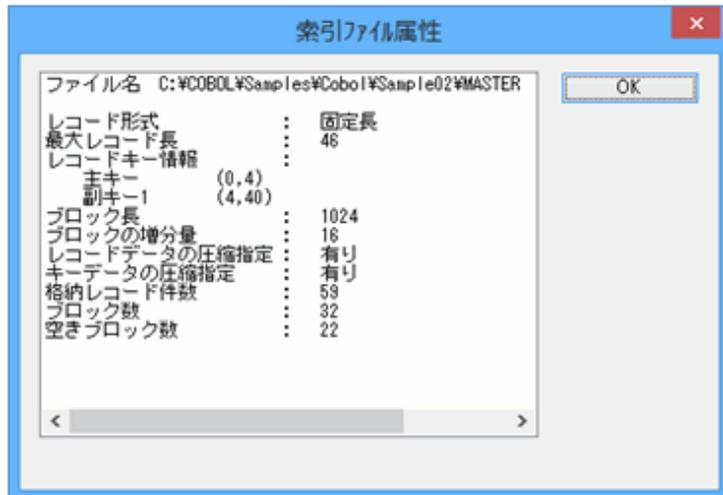
### 7.8.3.9 索引ファイルの操作

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]』





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。

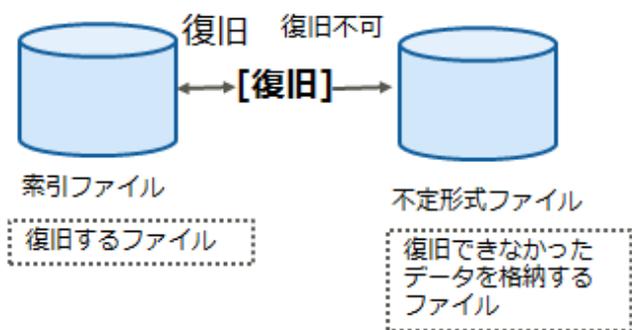
注：キー情報の形式を以下に示します。

[D] (オフセット, 長さ[, {N|N32}]) [/オフセット, 長さ[, {N|N32}]]...

- 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。
- N32 : UTF-32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。

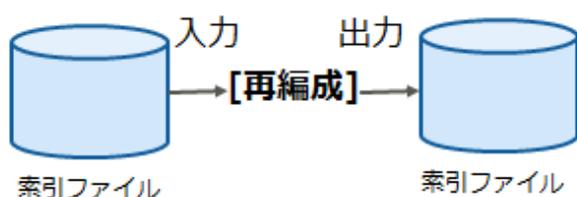
## 復旧

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]』



## 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]』



## 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、Actian社のレコードマネジメントシステムです。

### 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、“[15.2.14 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。

COBOLファイルシステム、BtrieveおよびPowerRDBconnector経由のデータベース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。

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)	PowerRDBconnector SQL Server(注1)	PowerRDBconnector Oracle(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10)	128G	32T	無制限
		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10)	—	—	—
		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	—	—	—
		索引ファイル	約1.7G(注10)	128G	32T	無制限
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	8,060	32,760
		可変長レコード形式	32,760	32,760	8,060	32,760
	レコードの最小長(バイト数)	レコード順	1	4	1	1
		行順	0	—	—	—
		相対	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	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)	—(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)	○(注3)
	LOCK MODE句	AUTOMATIC	○	○	—(注4)	—(注4)
		EXCLUSIVE	○	○	—(注4)	—(注4)
		MANUAL	○	○	—(注4)	—(注4)
	RECORD KEY句	指定できるデータの最大個数	254(注5)	119	16	32
		指定できるデータ総長の最大(バイト数)	254	255	255	データ・ブロックのサイズに依存
	ALTERNATE RECORD KEY句	指定できるデータの最大個数	254(注5)	119	16	32
		指定できるデータ総長の最大(バイト数)	254	255	255	データ・ブロックのサイズに依存
	レコードキーの項目	英数字	○	○	○	○
日本語		○	○	○	○	
符号なし外部10進数		○	○	○	○	

分類	項目		COBOLファイルシステム	Btrieve(注1)	PowerRDBconnector SQL Server(注1)	PowerRDBconnector Oracle(注1)
		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	○	○	○	○
	トランザクション管理	トランザクション開始指示	—	○ (注8)	○ (注8)	○ (注8)
		COMMIT指示	—	○ (注8)	○ (注8)	○ (注8)
		ROLLBACK指示	—	○ (注8)	○ (注8)	○ (注8)
	リカバリ		○ (注9)	○ (注9)	○ (注9)	○ (注9)

○ : サポート  
 — : 非サポート

- 注1  
Btrieveについては、“Actian PSQL V11”の情報をもとにしています。  
PowerRDBconnectorについては、“PowerRDBconnector”の情報をもとにしています。
- 注2  
PowerRDBconnectorでは、ファイルが存在しない場合、ファイルなしのエラーとなります。自動生成は行いません。
- 注3  
BtrieveおよびPowerRDBconnectorは、FILE STATUS=02を返却しません。また、PowerRDBconnector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  
エラーとなりませんが、PowerRDBconnectorでは、LOCK MODE句で指定したロック機構は動作しません。
- 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

読み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

BtrieveまたはPowerRDBconnector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。PowerRDBconnectorでトランザクション操作を行う方法については、“PowerRDBconnector 説明書”を参照してください。

• 注9

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。PowerRDBconnector経由でアクセスするデータベース(SQL Server、Oracle)で、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10

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。

### Unicodeデータの扱い

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。“7.1.4 Unicodeデータの扱い”を参考にしてください。



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。

#### 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。

```
ASSIGN TO "[パス名] ファイル名 { 指定なし | .BTRV } .
```

---

### 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。

---

MOVE "[パス名] ファイル名 { 指定なし  
                                  ,BTRV } " 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

---

ファイル識別名 = [パス名] ファイル名 { 指定なし  
  ,BTRV }

---

## 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、書込み前および読み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"Actian PSQL"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 "#DECFTO88" 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“#DECFJTO88”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“#DECFJTO88”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“#DECFJTO88”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“#DECFJTO88”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

```
      :  
FILE-CONTROL.  
  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  "#DECFJT088" 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 PowerRDBconnector

PowerRDBconnector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。PowerRDBconnector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、PowerRDBconnectorの使用方法について説明します。



### 参照

PowerRDBconnector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、“PowerRDBconnector”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、PowerRDBconnector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 { 指定なし
                                ,RDM } ".

```

## 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

PowerRDBconnector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 { 指定なし  
                             ,RDM } " TO データ名.
```

---

## ASSIGN句にDISKを記述した場合

PowerRDBconnector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 { 指定なし  
                                       ,RDM }
```

---

## 7.9.2.2 注意事項

- 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。

```
JMP03101-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=10'
```

- PowerRDBconnector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、“PowerRDBconnector説明書”を参照してください。

## 7.9.3 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。

### 実行環境で有効にする方法

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D 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

### 索引ファイル復旧関数 (CFURCOV)

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。

### 索引ファイル簡易復旧関数 (CFURCOVS)

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。

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の詳細は、“[H.2 索引ファイル復旧関数](#)”を参照してください。

## 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にアクセスするためのAPI(Application Program Interface)関数群です。COBOLファイルにアクセスする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は、“[H.4 COBOLファイルアクセスルーチン](#)”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るので、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。

印刷ファイルまたは表示ファイルを使用して、直接プリンタに出力することもできますが、帳票製品と連携することで多様な運用環境に対応することができます。

図8.1 List Works連携(電子帳票出力機能)  
アプリケーションサーバ/帳票出力サーバが同一筐体の場合

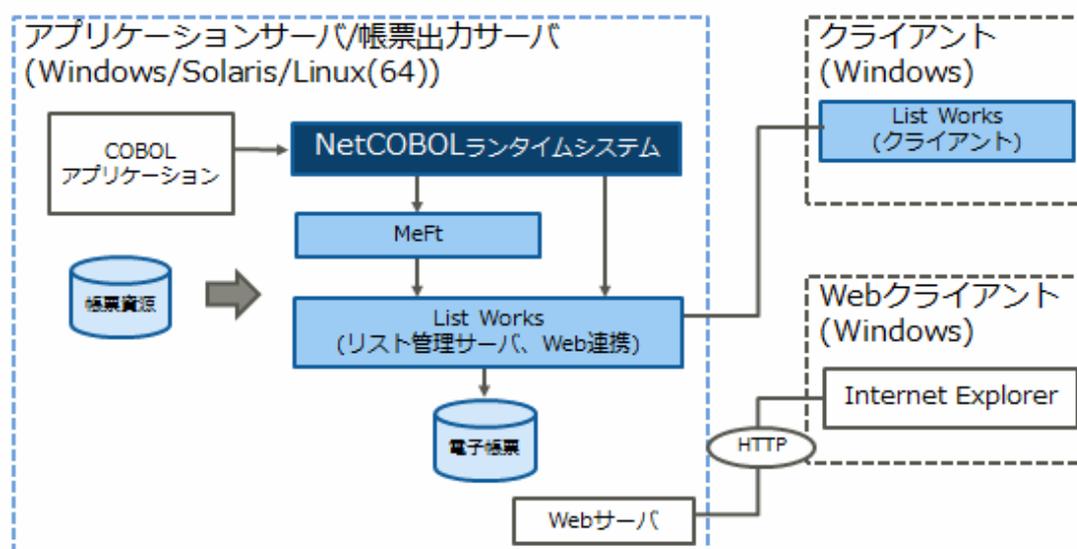


図8.2 List Works連携(電子帳票出力機能)  
アプリケーションサーバ/帳票出力サーバが別筐体の場合

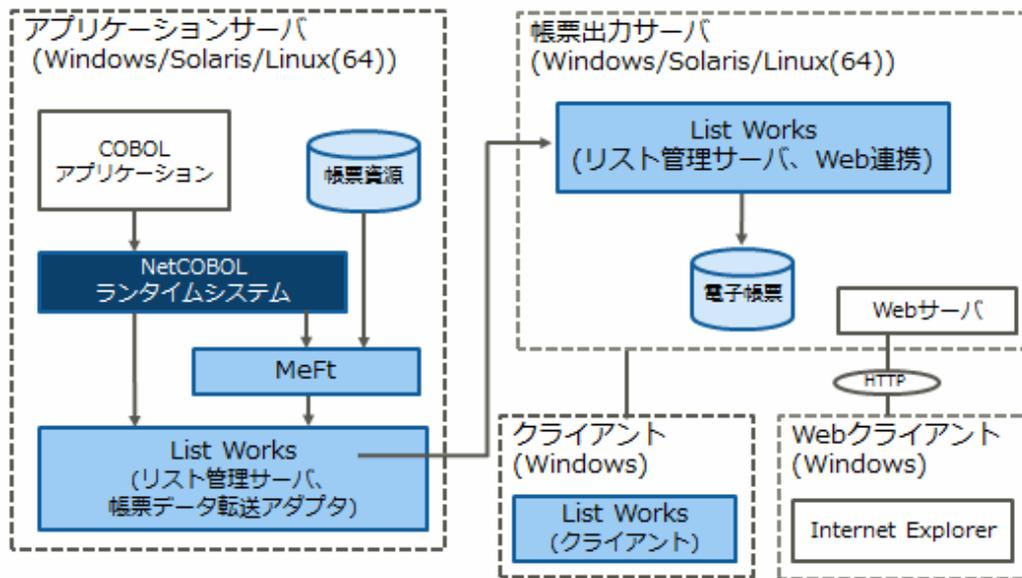
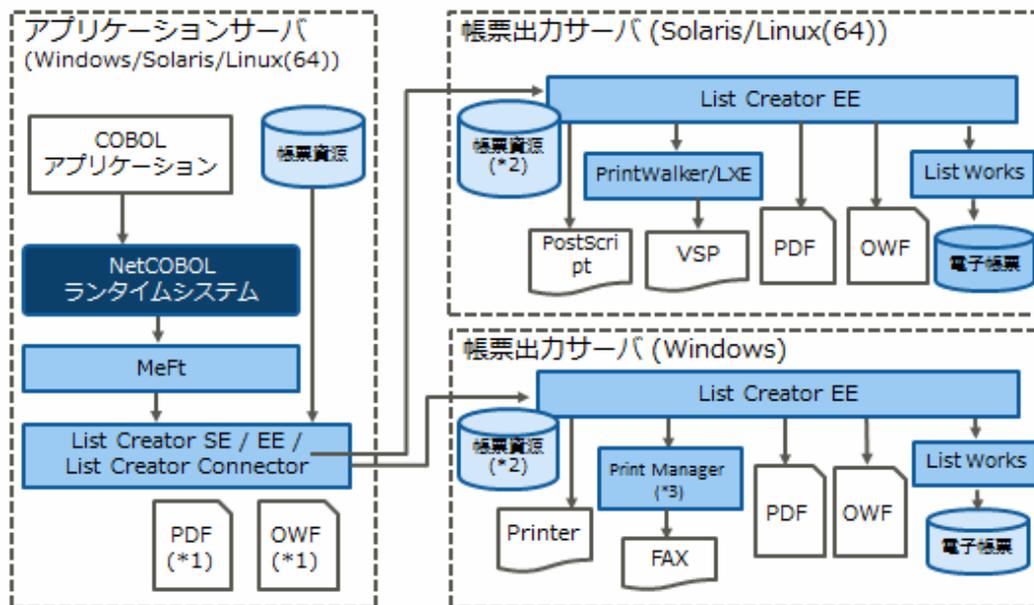


図8.3 List Creator連携(COBOLアプリケーション連携機能)



\*1) 帳票サーバで出力したPDF/OWFファイルをアプリケーションサーバへ転送(配置)することも可能です。

\*2) アプリケーションサーバから転送します。

\*3) Print Manager/FAX出力連携製品です。Windows(64)では未サポートです。

### 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
- ×

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ユーザーズガイド”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 オーバーレイオプション	—	○	○	○
PowerFORM (*2)	—	○	○	○
MeFt	—	—	○	○
MeFt/Web (*3)	—	—	○	○
Interstage List Works (*4)	○	○	○	○
Interstage List Creator EE (*5)	—	—	○	○

- \*1 : オーバー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 開発パッケージ、NetCOBOL Standard Edition開発パッケージ、NetCOBOL Enterprise Edition サーバ運用パッケージまたはNetCOBOL Standard Editionサーバ運用パッケージに同梱されています。
- \*4 : 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、Interstage List WorksをList Worksと記述します。
- \*5 : 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、Interstage List Creator EEの提供するPDF変換機能をe-DocGeneratorと記述します。

 参照

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”

- “List Works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“[8.1.7 フォームオーバーレイパターン](#)”を、FCBについては“[8.1.8 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“[8.1.10 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指定または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。

### 8.1.3 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)	3.0～300.0ポを0.1ポイント単位で指定できます。 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。

指定方法	文字サイズ
	<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@CBR_PrinterANK_Size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		FMPR
	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]“C.2.69 @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。
  - デバイスフォントがアウトラインフォントである場合
  - 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 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 書体の指定を省略した場合、以下の書体で印字します。

指定方法	書体
	<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 of 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。
  - “C.2.43 @CBR\_PrintFontTable (印刷ファイルで使用するフォントテーブルの指定)”
  - “C.2.80 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)”
  - “C.2.81 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)”
  - “8.1.13 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]“C.2.69 @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ユーザーズガイド”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、“8.1.13 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 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 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 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 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 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	印字文字の大きさと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	

指定方法	文字間隔
印字モード名(PRINTING MODE句のPITCH指定)	<p>0.01～24.00cpiの文字間隔を0.01cpi単位で指定します。</p> <p>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。</p> <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 : 印字文字の大きさと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ポ)	5	10	5	—	2.5	—	2.5	5
MODE-2 (9ポ)	8	16	8	—	4	—	4	8
MODE-3 (7ポ)	10	—	10	—	5	—	5	—
A (9ポ)	5	10	5	10	2.5	5	2.5	5
B (9ポ)	20/3	40/3	20/3	40/3	10/3	20/3	10/3	20/3
X-12P (12ポ)	5	10	5	10	2.5	5	2.5	5
X-9P (9ポ)	8	16	8	16	4	8	4	8
X-7P (7ポ)	10	20	10	20	5	10	5	10
C (9ポ)	7.5	15	7.5	15	3.75	7.5	3.75	7.5
D-12P (12ポ)	6	12	6	12	3	6	3	6
D-9P (9ポ)	6	12	6	12	3	6	3	6

## 8.1.4 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。

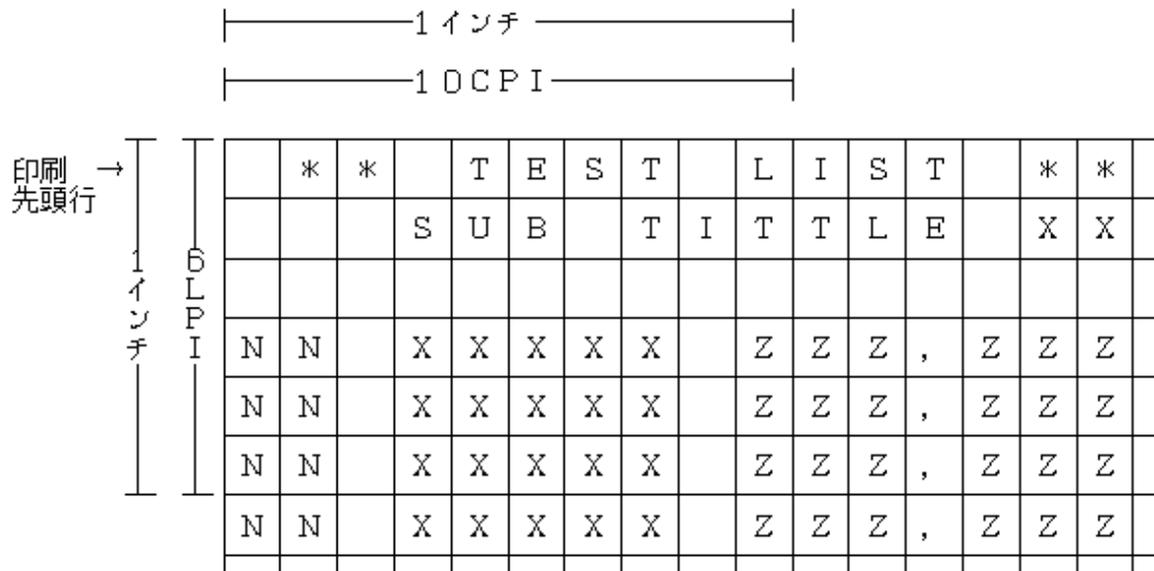
このため、実際にプログラミングに入る前に、スペーシングチャートのような設計用紙(または、FORMのようなグリッド表示可能なツール)を用いて、実際の印刷イメージで設計した後、この帳票設計をもとにプログラミングを行ってください。

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]



[FCB制御文]

```
LPI ( ( 6 ) ) , CH1 ( 1 ) , FORM ( ...
```

FCBについては、“8.1.8 FCB”を参照してください。

[COBOLプログラム]

```
SPECIAL-NAMES.
  PRINTING MODE PM1 IS FOR ALL
  AT PITCH 10.00 CPM.
  :
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.
  :
  CLOSE 印刷ファイル.
  STOP RUN.
```

### 8.1.5 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、“C.2.45 @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。

## 8.1.6 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

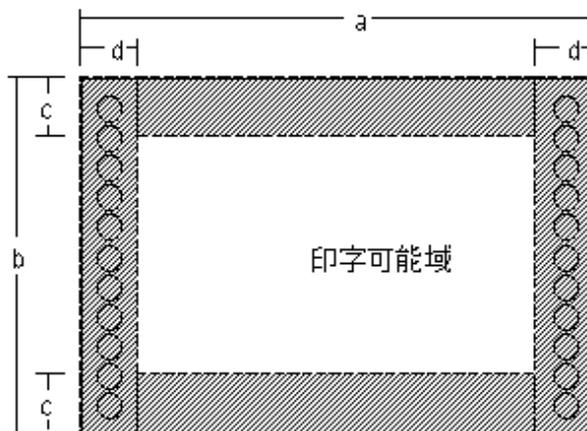
用紙内に印字可能な行数は、FCB制御文の定義により決定します。

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)およびチャン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たりします)。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。

### 8.1.7 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を使用します。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ユーザーズガイド”、“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。

### 8.1.8 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、それぞれチャンネル1からチャンネル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、チャンネル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、“[C.2.60 @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)
```

— 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、“[8.1.9 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のように行います。

```
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。

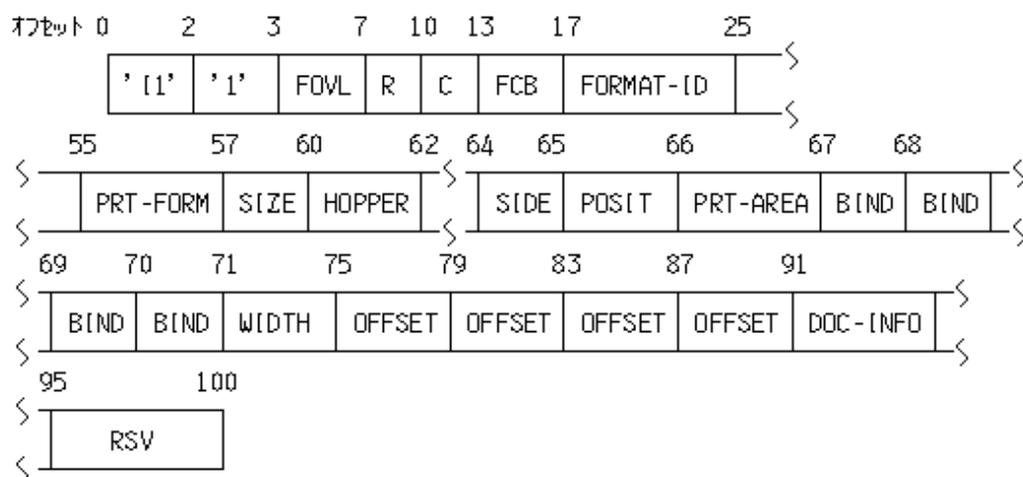
このように、COBOLプログラムで帳票印刷を行う場合、実際に使用する物理用紙サイズに対応したFCBの指定が必ず必要です。物理用紙サイズとFCBの指定が合致していない場合、意図した設計どおりの印刷結果が得られないことがありますので十分注意してください。

## 8.1.9 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 "I".	
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ユーザーズガイド”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。



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、“C.2.60 @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、“8.1.8 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。



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。



— 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

印刷情報ファイルの設定値→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、“[8.1.8 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、“[C.2.70 @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ユーザーズガイド](#)”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

印刷情報ファイルの設定値→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、“[8.1.8 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。

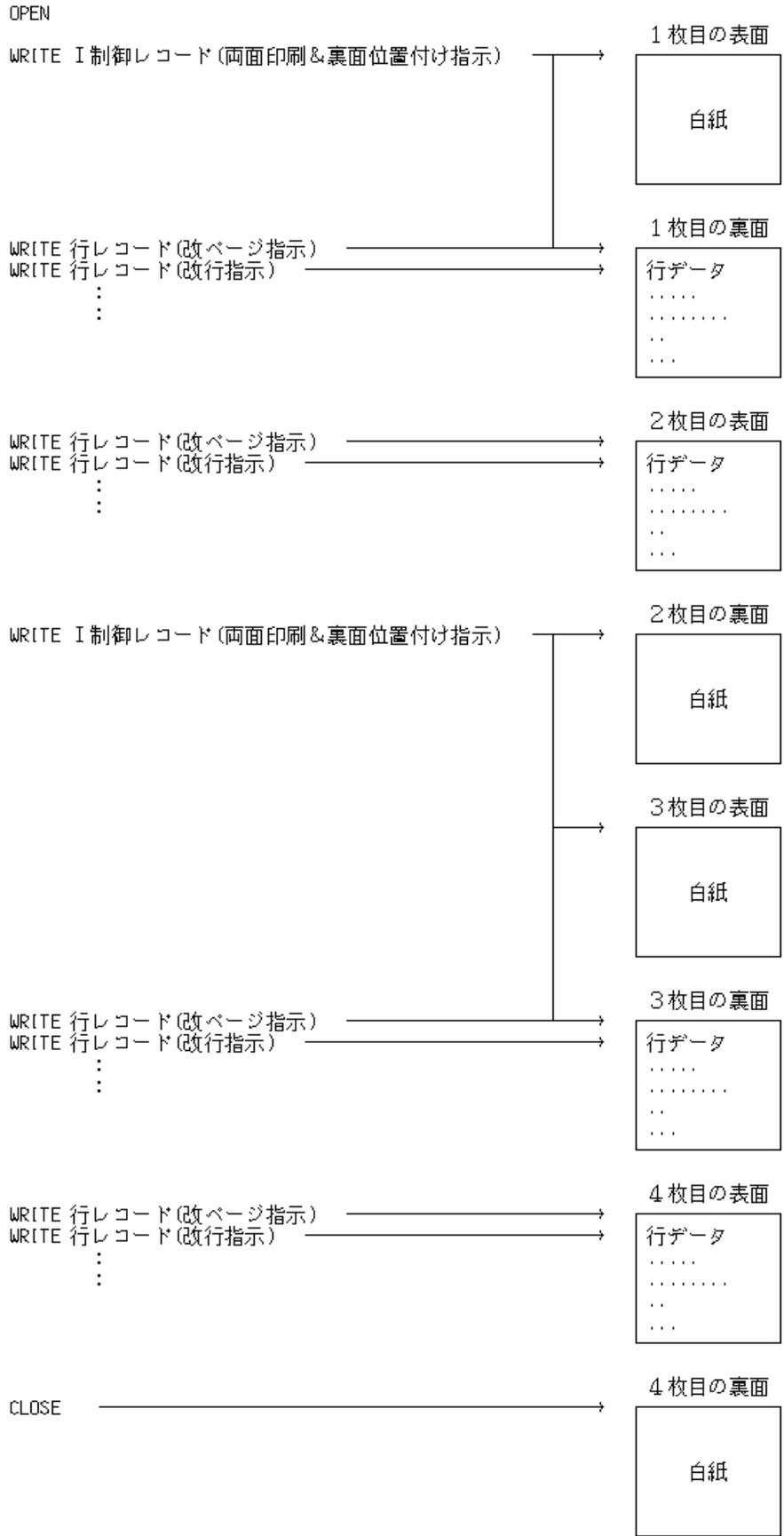
- "F"(表面位置付け)
- "B"(裏面位置付け)

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。



### 注意

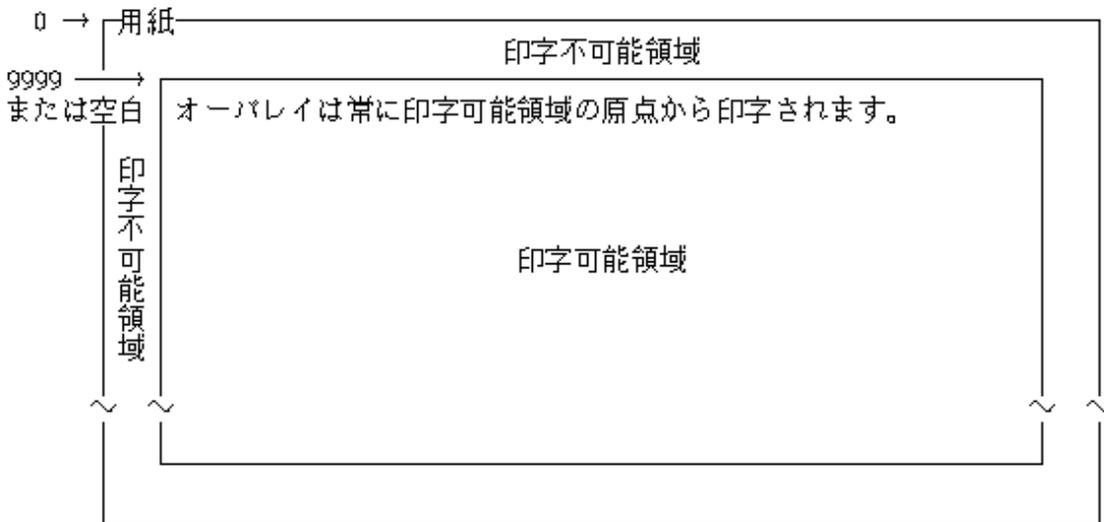
とじしろ方向の指定は、左とじ指定("L")および上とじ指定("U")だけ有効となります。したがって、右とじ指定("R")および下とじ指定("D")は無視されます。右方向および下方向にとじしろ幅を設ける場合、アプリケーションで意識する必要があります。

## 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、“8.1.6 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。

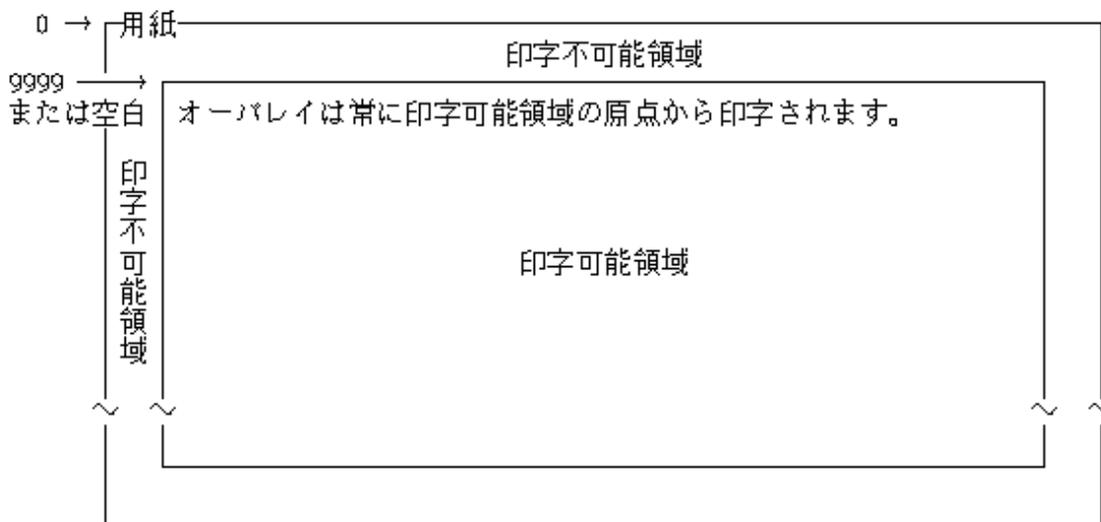
## OFFSET (印刷原点位置)

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、“8.1.6 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、“C.2.25 @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

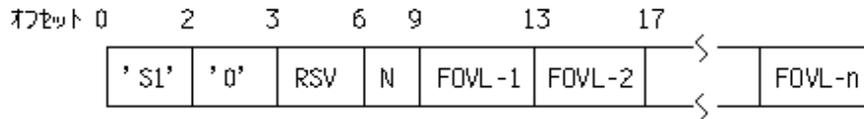
DOC-INFOフィールドの指定は、FORMAT句付き印刷ファイルには指定できません。FORMAT句付き印刷ファイルを利用して、プリントマネージャに任意の文書名を表示させる場合、本フィールドに空白を指定し、プリンタ情報ファイルで文書名を指定します。詳細は、“MeFtユーザーズガイド”を参照してください。

#### 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の数を指定します。

#### 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。



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。

## 8.1.10 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、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ユーザーズガイド”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ユーザーズガイド”、“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。



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。

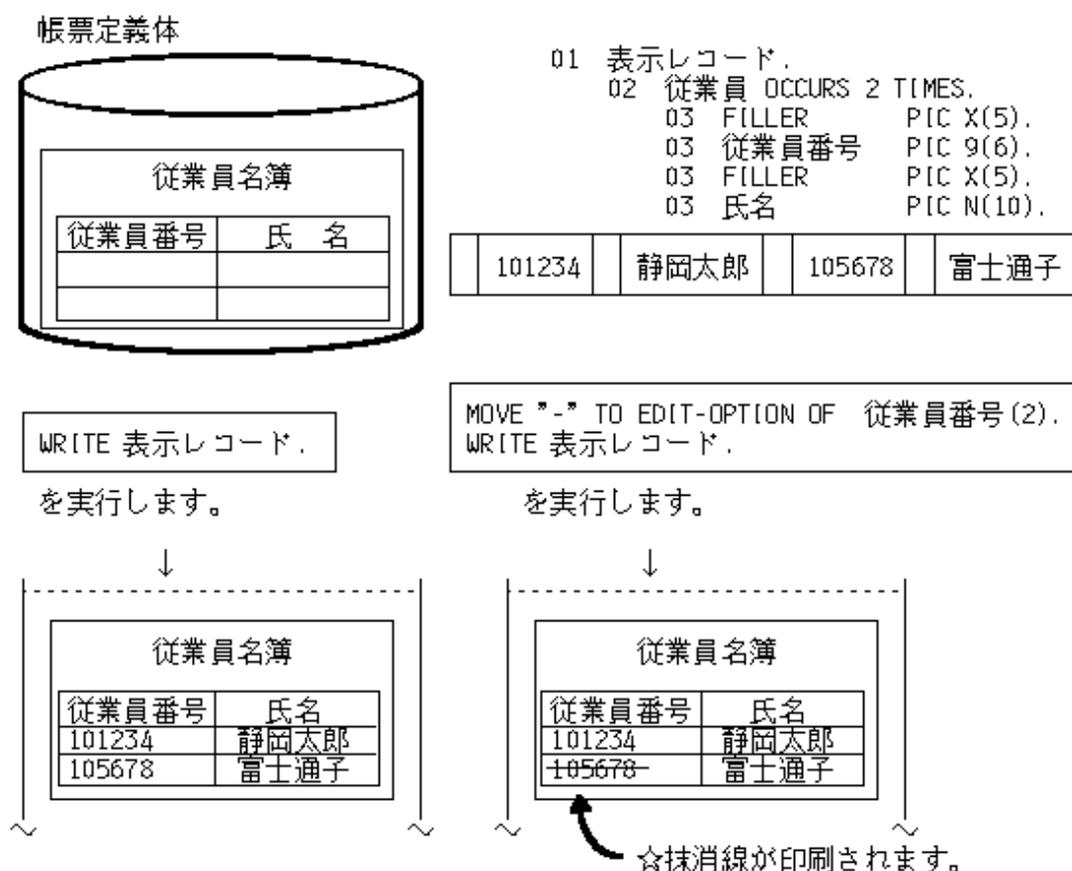
## 8.1.11 特殊レジスタ

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ユーザーズガイド”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パ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。このとき、プリンタ情報ファイルにPRITITEMCTL(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ユーザーズガイド”を参照してください。

## 8.1.12 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。



- ・ 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 BOTTOM }	← 上端/下端合わせ指定(省略可能)
DocumentName = 文書名	← 文書指定(省略可能)
OverlayPrintOffset = { VALID 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 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: COMn: PRTNAME:プリンタ名 }	← 出力プリンタ名を指定(省略可能)
PAPERSIZE={A3   A4   A5   B4   B5   LTR   任意}	デフォルト用紙サイズを指定します。(省略可能)
PRTFORM={ P   L   PZ   LZ   LP }	デフォルトの印刷形式を指定します。(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)
FOVLTYP=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す、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]“C.2.51 @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]“C.2.25 @CBR\_DocumentName\_xxxx (I制御レコードによる文書名の指定)”

## 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]“C.2.41 @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。

— 印刷原点位置を指定しない場合の印刷結果

論理座標の (0,0)

売り上げ伝票		1998年10月12日			
商品名	商品番号	個数	単価	小計	
富士通ノート	N0001A5B	10	¥150	¥1,500	
富士通ケシゴム	KS05A42C	35	¥100	¥3,500	

— 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。List Works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、List Works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”、“List Worksのオンラインマニュアル”および“List Worksのヘルプ”を参照してください。

## STREAMENV

List Works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、List Works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、List Worksによって規定されています。詳しくは、“List Worksのオンラインマニュアル”および“List Worksのヘルプ”を参照してください。

## PRTOUT

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。

PRTOUT指定は、省略することも可能です。PRTOUT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PRTOUT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PRTOUT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、“[C.2.81 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、List Works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、List Worksで電子化・登録するときに使用するList Works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。

## PAPERSIZE

デフォルト用紙サイズを指定します。指定可能なサイズは、A3/A4/A5/B4/B5/LTRです。

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。環境変数情報の指定方法については、“[C.2.70 @PRN\\_FormName\\_xxx\(用紙名の指定\)](#)”を参照してください。

なお、本指定が省略された場合、プリンタドライバの設定値が有効となります。

### 注意

デフォルト用紙サイズを変更する場合は、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、“[8.1.8 FCB](#)”を参照してください。

## PRTFORM

デフォルトの印刷形式を指定します。指定可能な形式は、P/L/PZ/LZ/LPです。

なお、本指定が省略された場合、プリンタドライバでの設定値が有効となります。

### 注意

- デフォルトの印刷形式を指定する場合は、用紙サイズも指定する必要があります。指定可能な用紙サイズと印刷形式の組合せについては、“[COBOL文法書](#)”の“付録F 制御レコード”の“表F.1 用紙サイズと印刷形式の組合せ”を参照してください。
- デフォルトの印刷形式を変更する場合は、指定した形式に合わせて、FCB制御文を定義・指定する必要があります。詳細は、“[8.1.8 FCB](#)”を参照してください。
- “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。

## 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“[C.2.83 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。

## FOVLTTYPE

フォームオーバーレイパターンファイル名の先頭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¥KOL2OVD1.OVD”の場合

```
FOVLDIR=C:¥FOVLDATA
FOVLTTYPE=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“C.2.84 FOVLTTYPE(フォームオーバーレイパターン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
FOVLTTYPE=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“C.2.85 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
FOVLTTYPE=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“C.2.86 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、FOVLTTYPE、FOVLNAMEおよびOVD\_SUFFIXの指定は、List Works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。

## 8.1.13 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={ R B I 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。
  - 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。

### 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

“C.2.43 @CBR\_PrintFontTable(印刷ファイルで使用するフォントテーブルの指定)”

“C.2.80 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)”

“C.2.81 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 Unicodeの印刷について

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、混在する場合でも問題なく動作します。ただし、ひとつの字類に対して複数のエンコードが混在する場合、たとえば、UTF-16とUTF-32の日本語項目が混在する場合は翻訳時エラーとなります。ひとつの字類はひとつのエンコードで統一してください。

```
FILE-CONTROL.
  SELECT OUT-FILE ASSIGN TO PRTFILE.
  :
FD OUT-FILE.                *> 翻訳時エラー (UTF-16とUTF-32が混在)
01 OUT-REC  PIC X(120).
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(10) ENCODING IS U16L.
  02 PRT-ADDR PIC N(20) ENCODING IS U32L.
  :
WRITE OUT-REC FROM PRT-DATA AFTER ADVANCING 1 LINE.
```

レコードの編集処理において、集団項目転記などにより、レコード中に字類と合わないコードの空白づめが行われる場合があります。この結果、シフトJISでは英数字と日本語のエンコードが同じため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として顕在化することがあります。

```
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 NG"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 NG"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。

## 8.1.15 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

- 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
```
- 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。
  - 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
  - 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

## 8.2 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なります。
	ASSIGN句	ファイル参照子	PRINTER、PRINTER-n、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す。(注)

注：設定される値については、“付録D 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

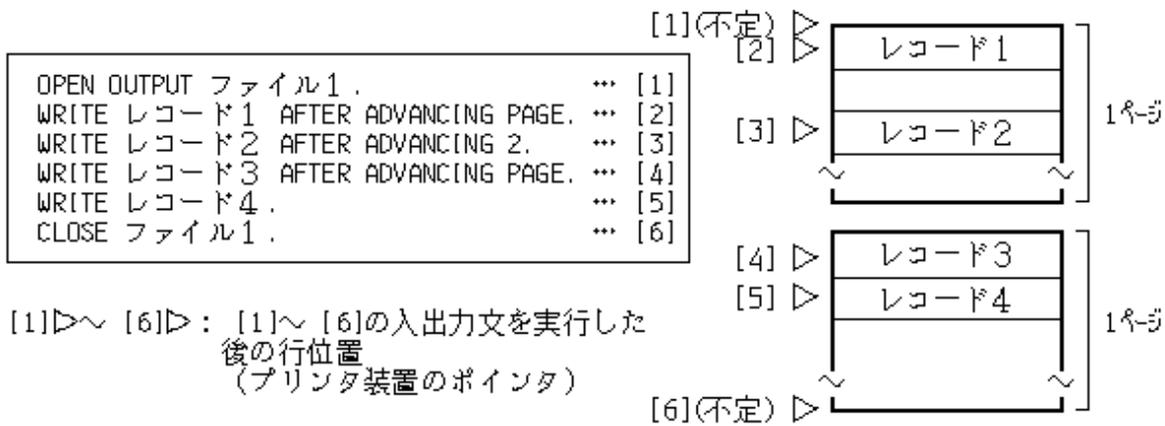
#### OPEN文およびCLOSE文

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。

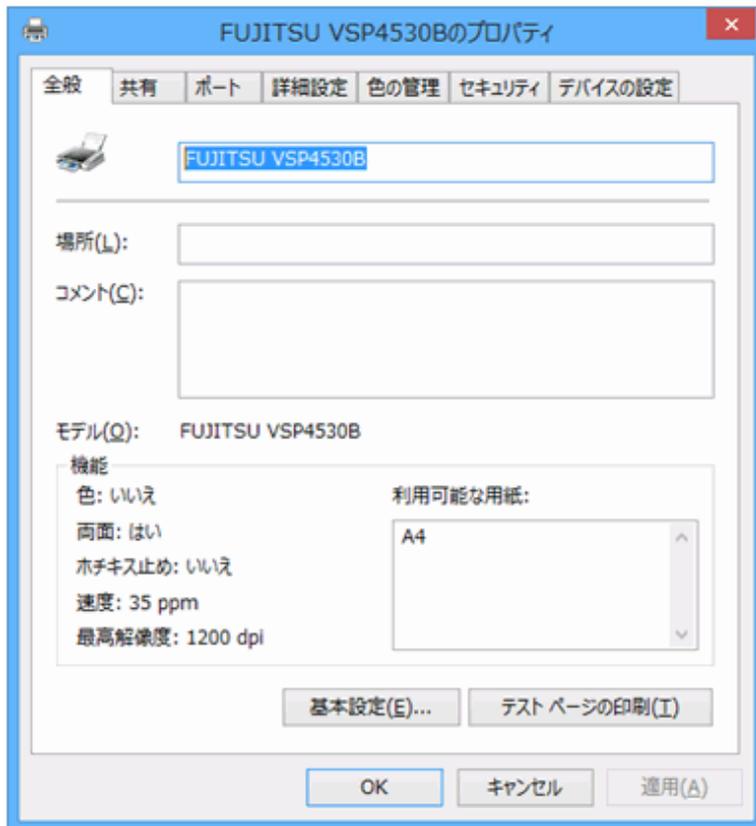
### 印刷装置の指定方法

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使用します。

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 VSP4530B

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 VSP4530B



### 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。



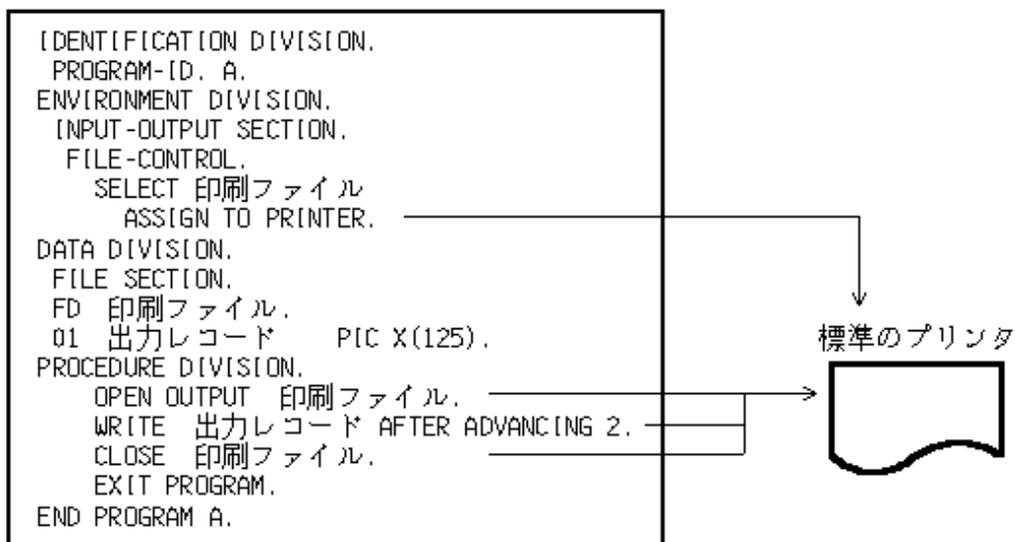
“LPT1:”

## 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8.4 プリンタ名を指定した場合の例

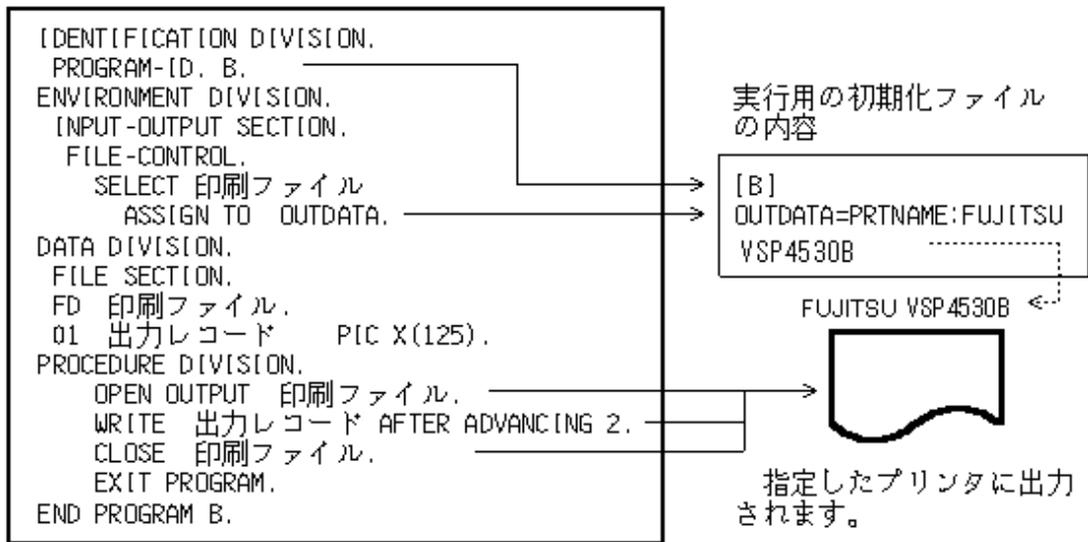
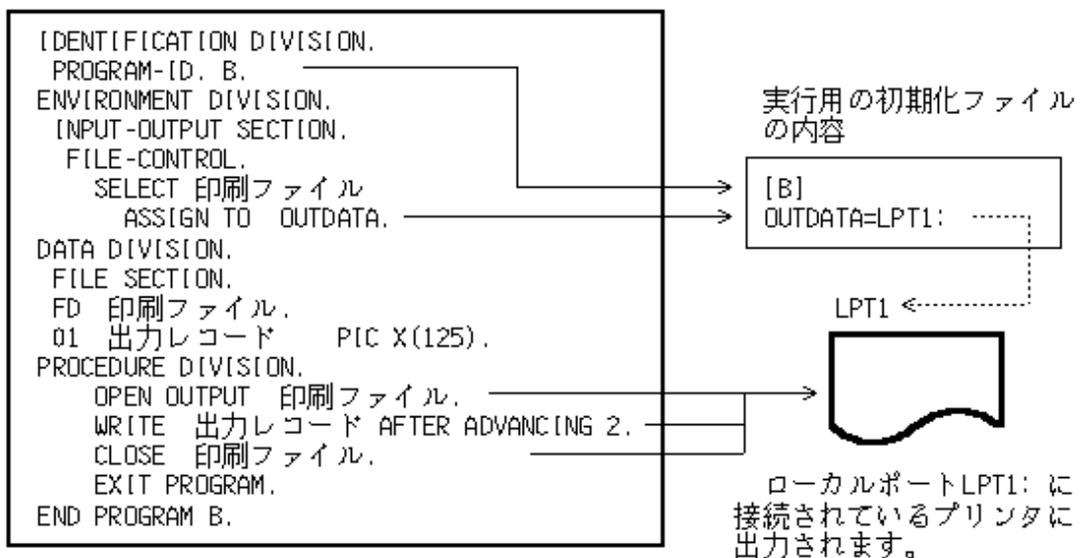


図8.5 ローカルポート名を指定した場合の例



#### 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8.6 プリンタ名を指定した場合の例

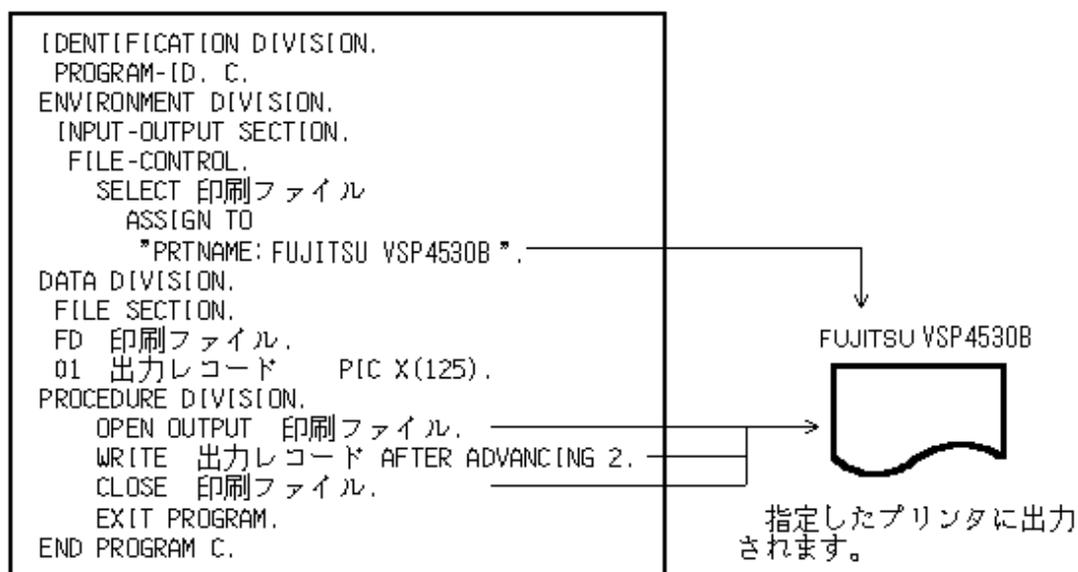
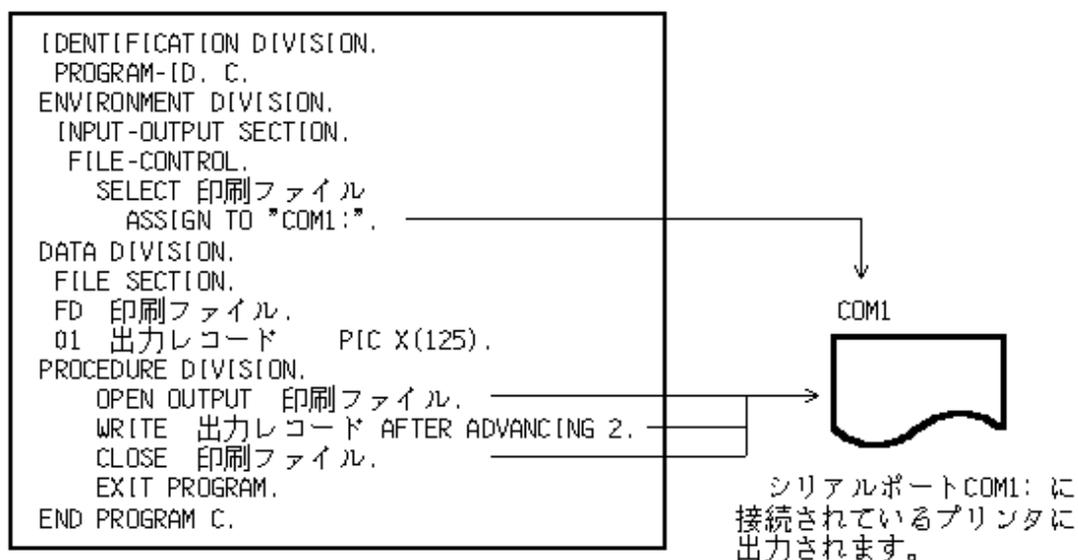


図8.7 シリアルポート名を指定した場合の例



#### 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8.8 プリンタ名を指定した場合の例

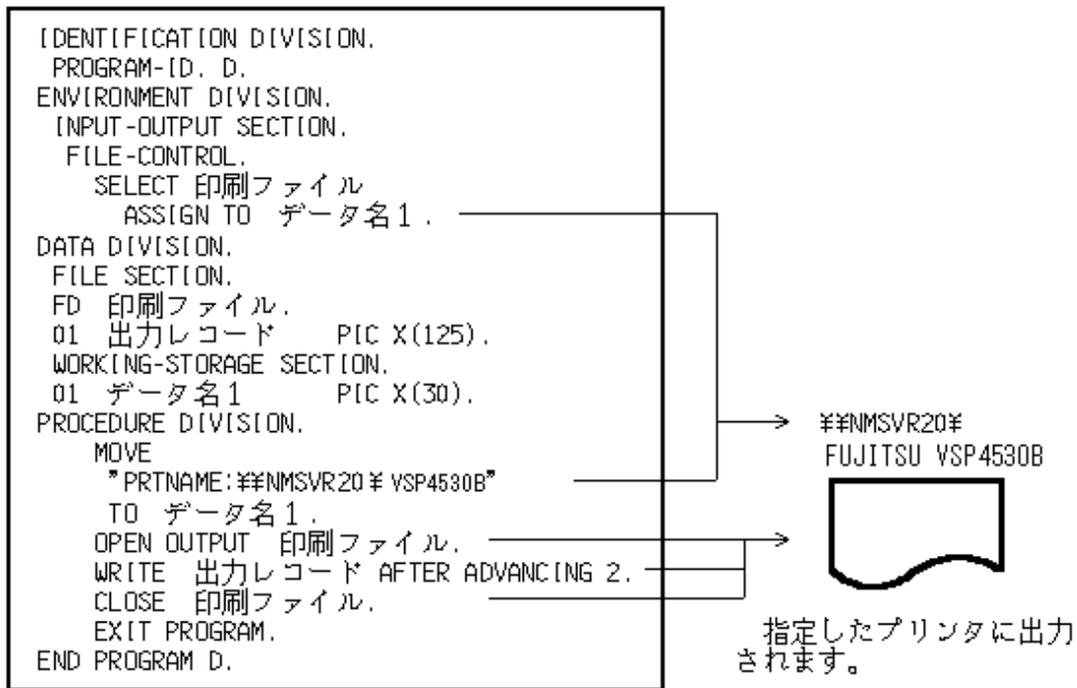
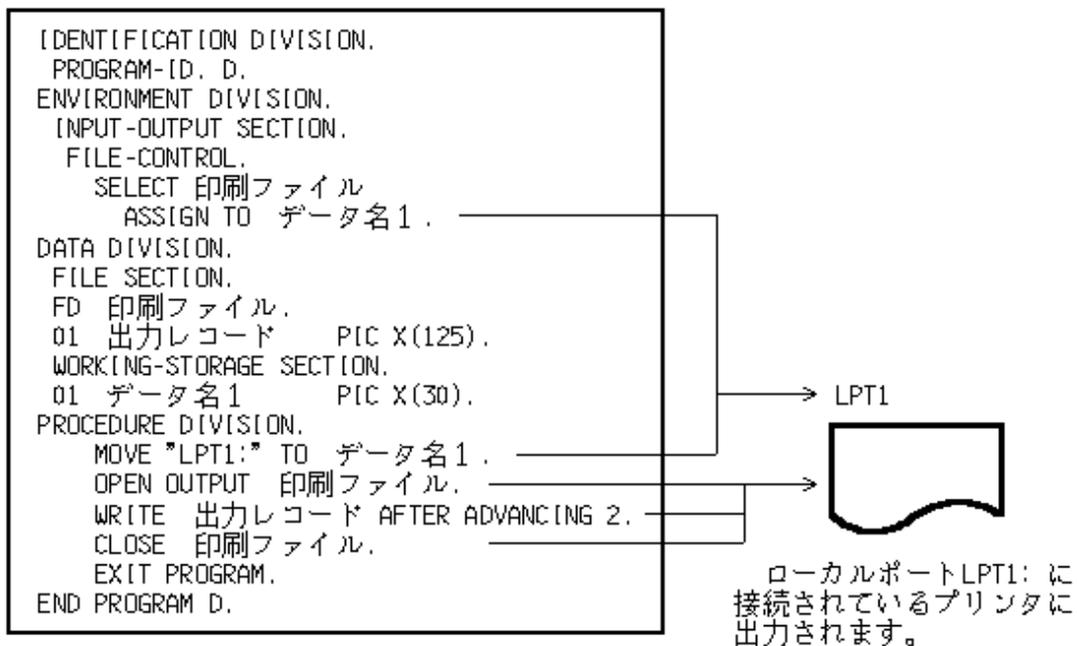


図8.9 ローカルポート名を指定した場合の例



ASSIGN句に文字列PRINTER-nまたは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)またはPRINTER\_n(n=1~99)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または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8.10 プリンタ名を指定した場合の例

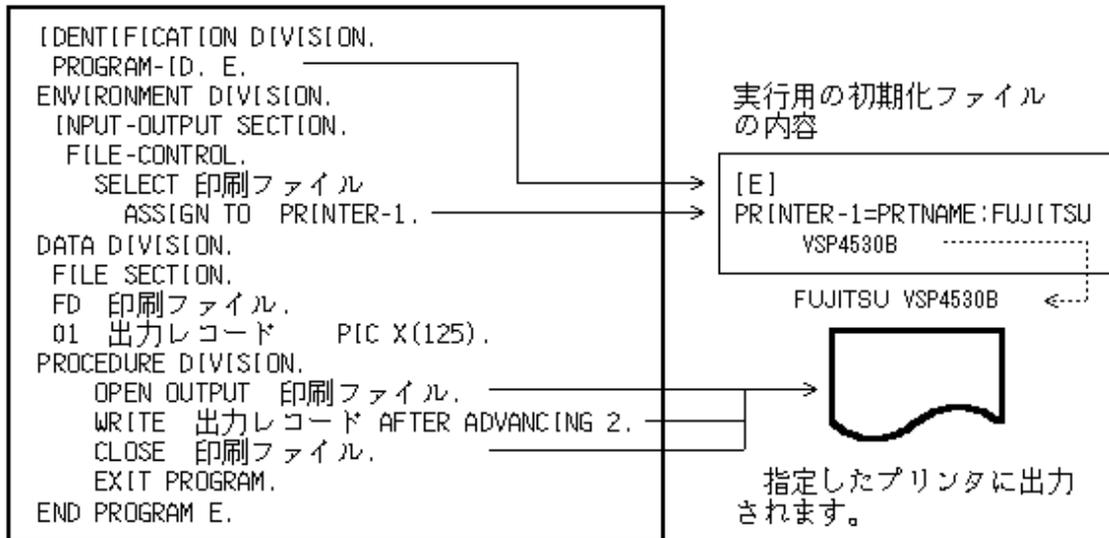
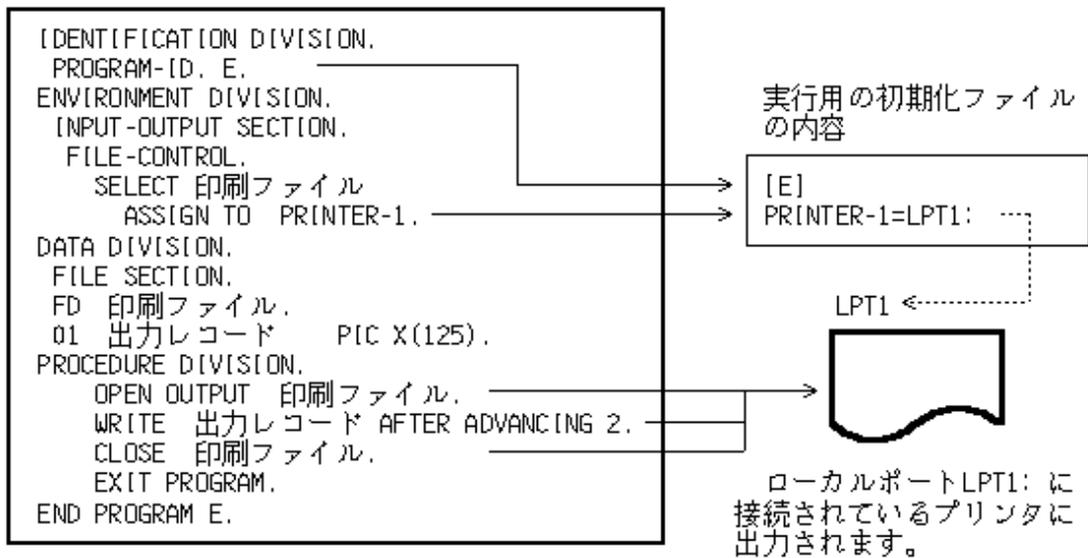


図8.11 ローカルポート名を指定した場合の例

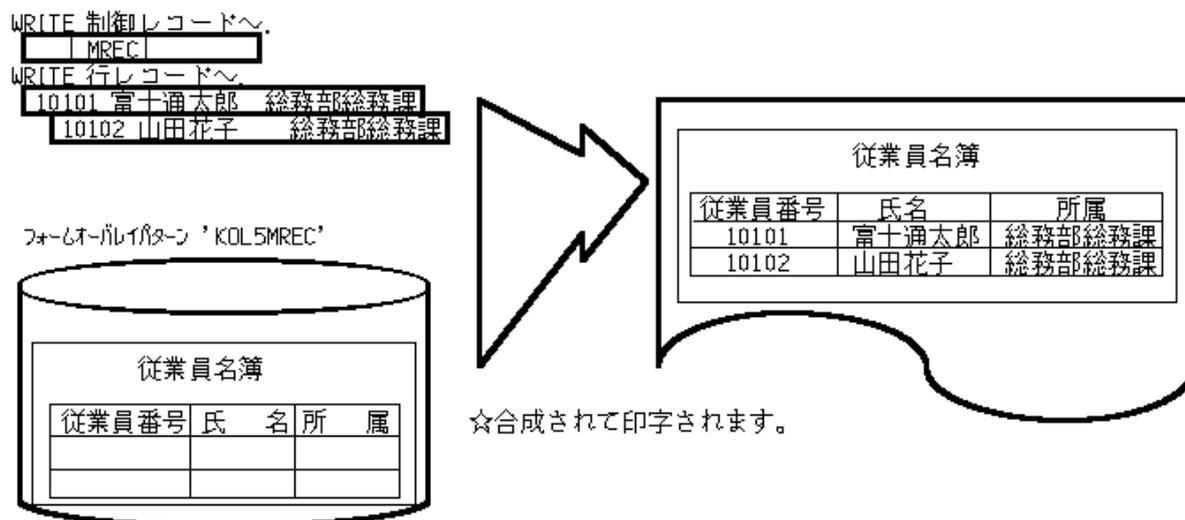


## 8.3 フォームオーバーレイおよびFCBを使う方法

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、“8.1.3 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用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、“8.1.3 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、“8.1.9 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]“C.2.83 FOVLDIR (フォームオーバーレイパターンのフォルダの指定)”、“8.1.12 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]“C.2.86 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)”、“8.1.12 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]“C.2.84 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)”、“8.1.12 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、“8.1.8 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

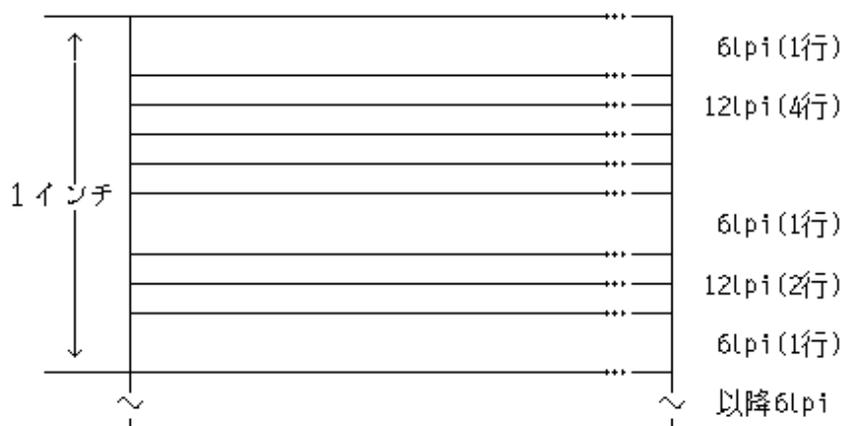
### FCB制御文の例

- ・ 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

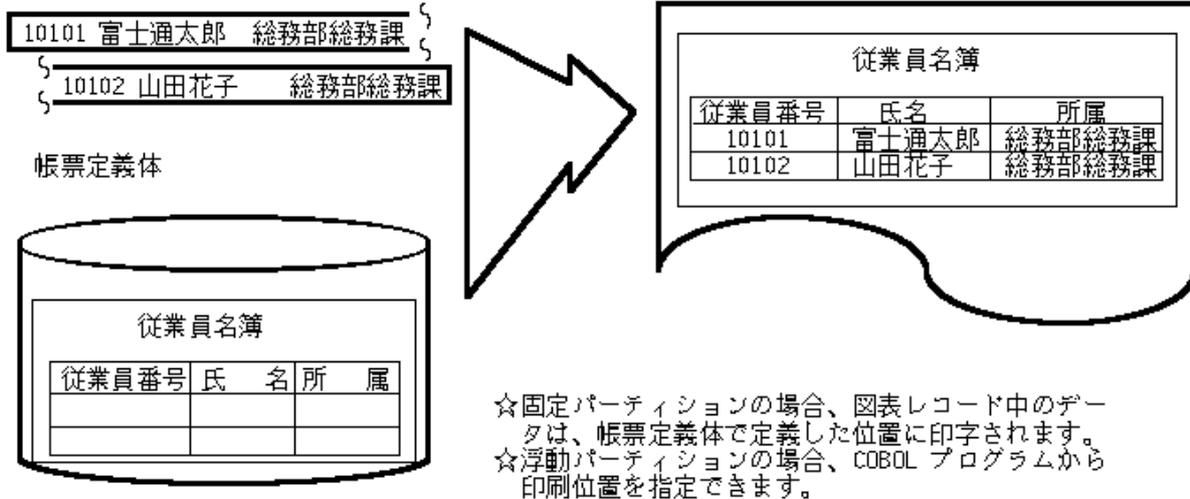
---

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“8.1.3 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 ～、



## 注意

- 以下の帳票定義体は、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。
  - 自由形式
  - タックシール形式
  - 段組みパーティションの改ブロック
  - フレームパーティションの改フレーム
  - 矩形領域の出力範囲指定
  - パーティション形式の下端情報設定
- 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。
- 日本語項目がエンコードUTF-32のデータを帳票出力する場合、帳票定義体をUTF-32用に変換してください。定義体の変換については“[J.6 UTF-32用定義体変換コマンド](#)”を参照してください。

### 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た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 <input type="text"/>	固定パーティション (HEAD)
項目 <input type="text"/> 金額 <input type="text"/>	浮動パーティション (ITEM)
小計 <input type="text"/>	浮動パーティション (SUBTOTAL)
合計 <input type="text"/>	固定パーティション (TOTAL)

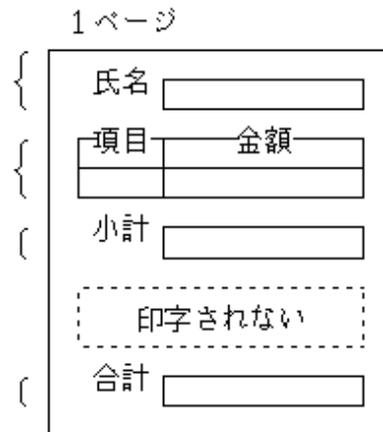
```

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      ⇨ ⇨ ⇨ ⇨
    
```



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 (HEAD)
項目 <input type="text"/> 金額 <input type="text"/>	浮動パーティション (ITEM)
小計 <input type="text"/>	浮動パーティション (SUBTOTAL)
合計 <input type="text"/>	固定パーティション (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 99999999
合計 <input type="text"/>



注意

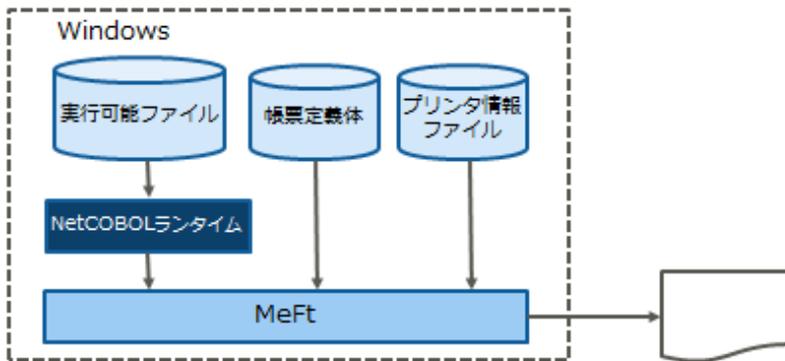
行レコードまたは浮動パーティションを固定パーティションの印刷位置に出力した場合には、その位置に印刷する固定パーティションは、そのページ中では印刷できません。そのページは改ページされて、固定パーティションは次のページの印刷位置に印刷されます。

### 8.4.1.2 帳票の電子化

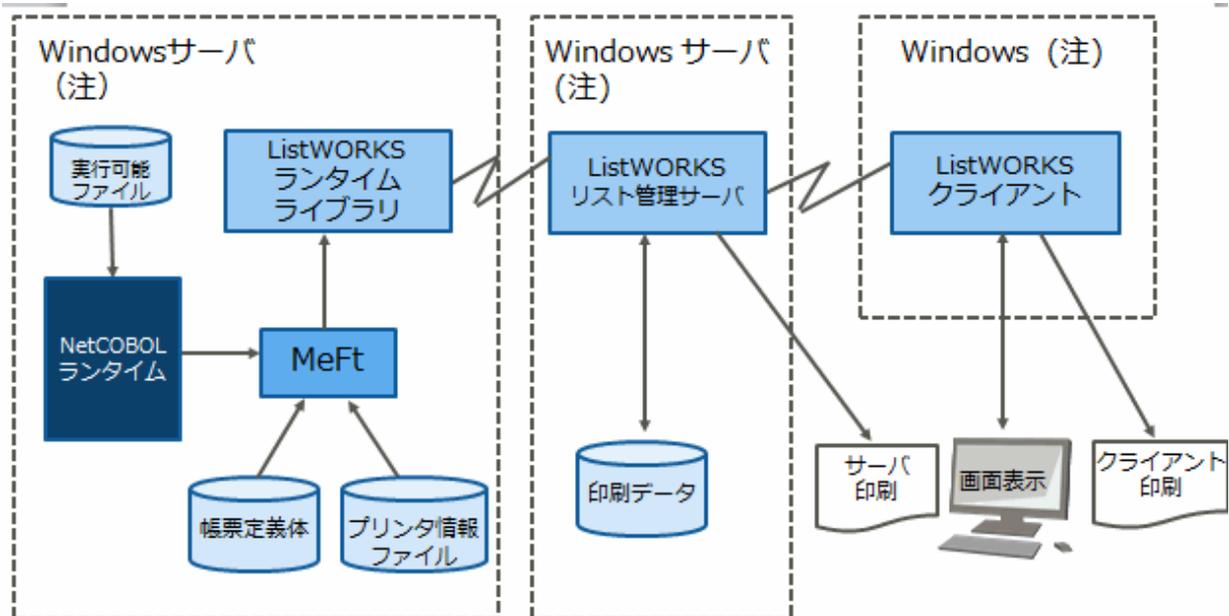
プリンタ情報ファイルに指定を追加することにより、MeFt経由で出力する帳票を電子化することができます。帳票を電子化する方法には、電子帳票保存とPDFファイル出力があります。それぞれの詳細は、“MeFtユーザーズガイド”および各連携製品のマニュアルを参照してください。

以下にそれぞれの関連図を示します。

- プリンタ出力の場合

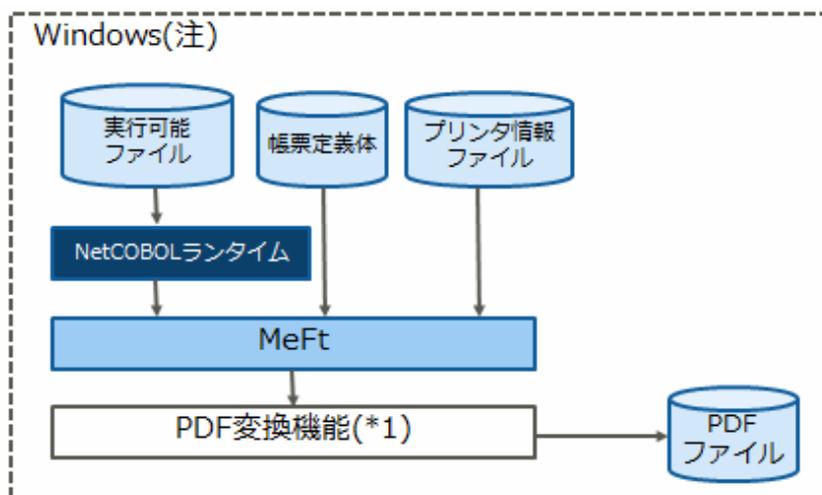


- 電子帳票保存の場合



Meftの出力帳票をList Worksで扱うことのできる電子帳票の形式にします。

- PDFファイル出力の場合



\*1 : Interstage List Creator Enterprise EditionのPDF変換機能

Interstage List Creator Enterprise EditionのPDF変換機能を使用して、MeFtの出力帳票をPDF(Portable Document Format)ファイルにします。

注 : List WorksおよびInterstage List Creator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なります。
	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) 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D 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。

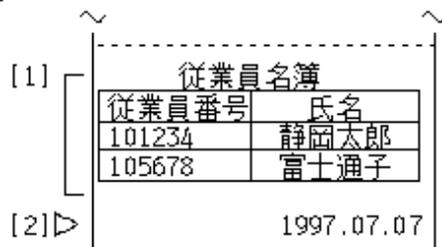
```

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]

```



#### 【出力結果】



#### 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.3.21 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.3.20 FORMEXT (画面帳票定義体ファイルの拡張子の指定)”

#### エンコードUTF-32のデータを帳票出力する場合

日本語項目がエンコードUTF-32のデータを帳票出力する場合、翻訳時のCOPY文の展開にUTF-32用帳票定義体が入力されるように、上記の翻訳オプションおよび環境変数の指定を確認してください。

なお、下記のいずれかを指定した場合に、日本語項目がエンコードUTF-32の帳票になります。

- 帳票定義体からレコード定義を取り込む場合、COPY文にエンコードUTF-32のENCODING句を指定する。
- ファイル記述項の配下のレコード定義として取り込む場合、ファイル記述項にエンコードUTF-32のENCODING句を指定する。
- 翻訳オプションENCODEで日本語項目にエンコードUTF-32を指定する。

強さの関係は、a.> b.> c.となります。

### 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ユーザーズガイド”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。

- 環境変数MEFTDIRに設定したフォルダ
- 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 PRTPFILE
```

[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。

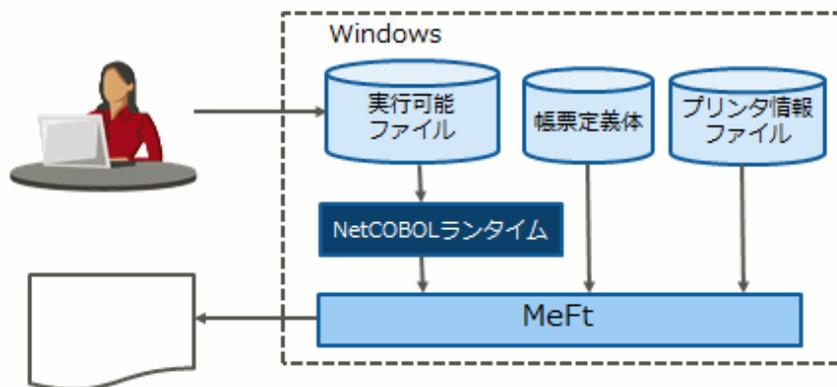
### 8.5.1 概要

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。

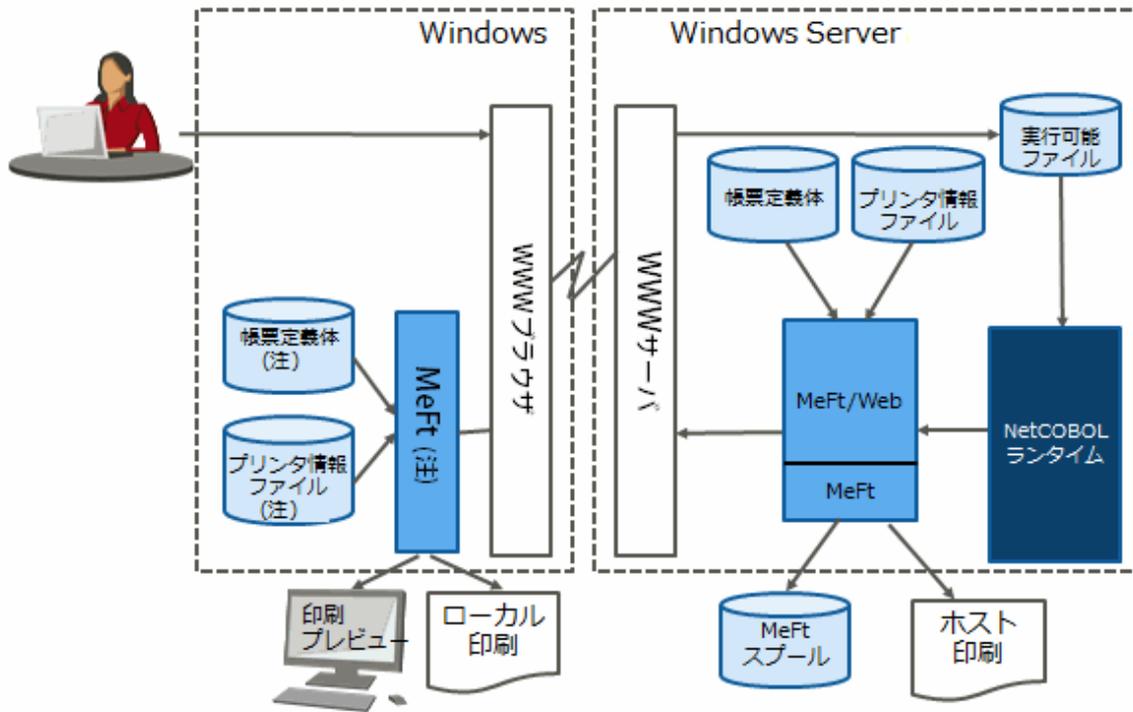
これらの関連図を以下に示します。

- 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(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：サーバから自動的にダウンロードされます。

MeFt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、プリンタ情報ファイルは実行時までには作成します。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の詳細な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。

### 注意

- ・ 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、以下に従ってください。
  - 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。
- ・ 日本語項目がエンコードUTF-32のデータを帳票出力する場合、帳票定義体をUTF-32用に変換してください。定義体の変換については“[J.6 UTF-32用定義体変換コマンド](#)”を参照してください。

## 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なります。
	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 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。(注1) 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、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 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D 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

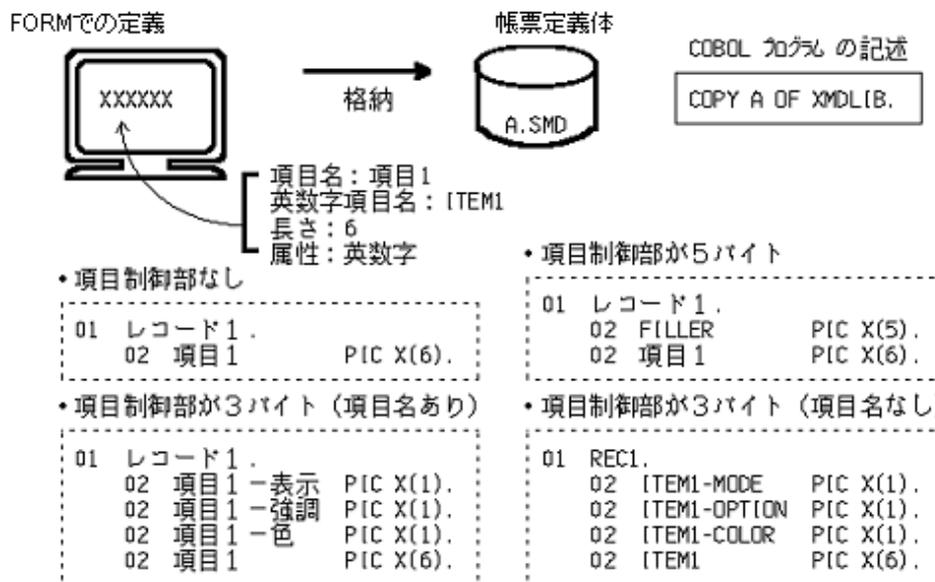
処理種別	値	制御情報	
プリンタ装置の制御	"CT"	改ページ	"PAGE"
パーティション出力	"PW"	nnn 行改行後出力	"Annn"

処理種別	値	制御情報	
		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、“8.1.11 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.3.21 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.3.20 FORMEXT (画面帳票定義体ファイルの拡張子の指定)”

#### エンコードUTF-32のデータを帳票出力する場合

日本語項目がエンコードUTF-32のデータを帳票出力する場合、翻訳時のCOPY文の展開にUTF-32用帳票定義体が入力されるように、上記の翻訳オプションおよび環境変数の指定を確認してください。

なお、下記のいずれかを指定した場合に、日本語項目がエンコードUTF-32の帳票になります。

- 帳票定義体からレコード定義を取り込む場合、COPY文にエンコードUTF-32のENCODING句を指定する。
- ファイル記述項の配下のレコード定義として取り込む場合、ファイル記述項にエンコードUTF-32のENCODING句を指定する。
- ENCODEオプションで日本語項目にエンコードUTF-32を指定する。

強さの関係は、a.>b.>c.となります。

### 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ユーザーズガイド”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。 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/Webを使用する場合

MeFt/Webを使用する場合には、“MeFt/Web ユーザーズガイド”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ユーザーズガイド”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、List Works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。

#### List Worksとは

List Works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

List Works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、List Works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List Worksに関する詳細は、“[List Worksのオンラインマニュアル](#)”および“[List Works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、List Works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List Works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、“[C.2.81 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または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List Works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List Works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][“C.2.44 @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:\PRAPL\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List Worksに向けて(帳票を電子化するため)、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、“[8.1.12 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、List Works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、“[8.1.12 印刷情報ファイル](#)”を、電子帳票情報ファイルの詳細は、“List Worksのオンラインマニュアル”および“List Works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List Worksの電子保存装置名を指定します。List Worksの仮想プリンタを使用する場合、List Works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、List Works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、“[8.1.12 印刷情報ファイル](#)”を、環境変数情報の指定の詳細は“[C.2.81 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。

## List Worksの準備・設定

List Works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、“List Worksのオンラインマニュアル”および“List Works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

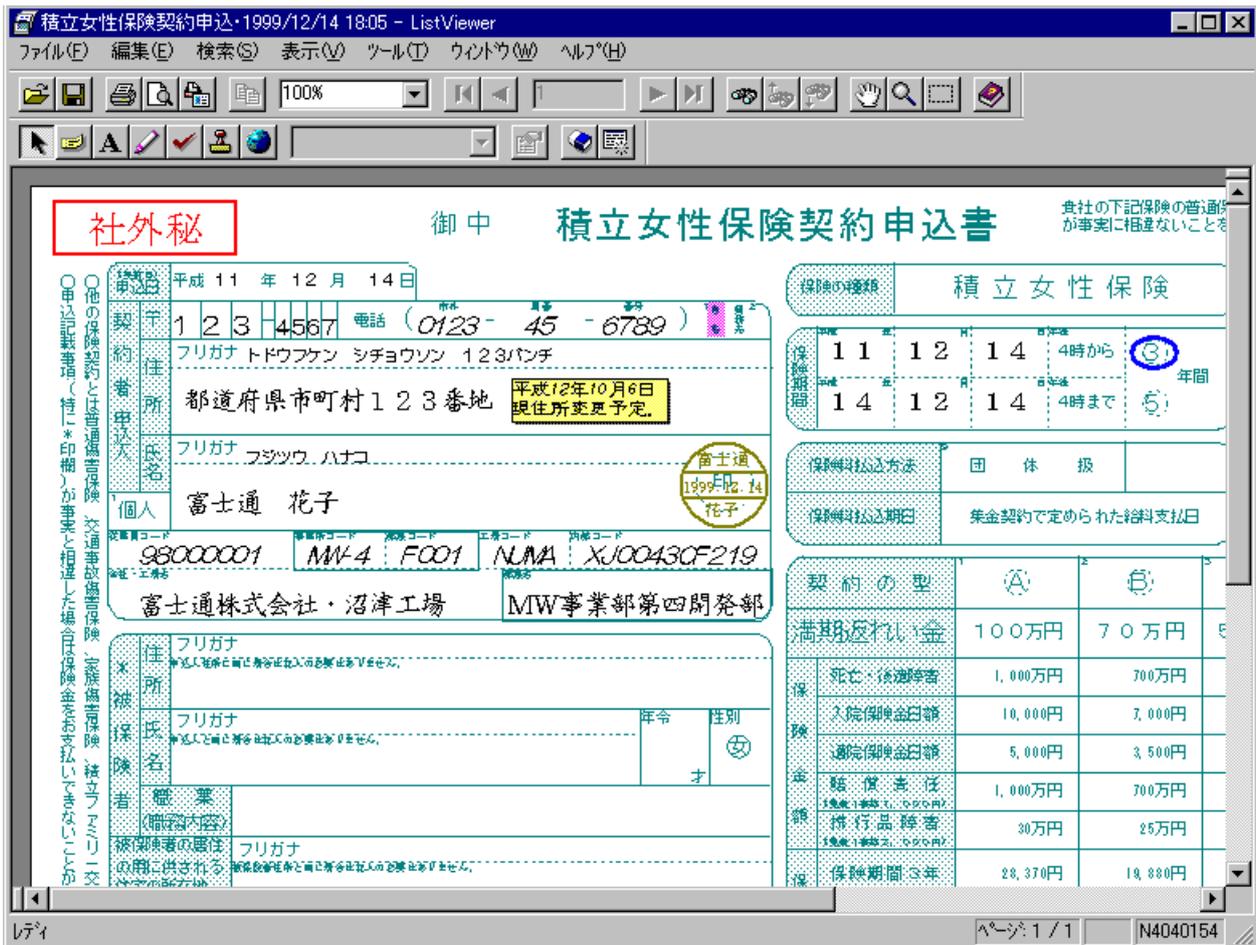
```
[COBOL ソースプログラムのASSIGN句の記述]
SELECT 印刷ファイル ASSIGN TO S-PRTFILE
[環境変数情報]
PRTFILE=PRTNAME:FUJITSU FMLBP, INF(C:\PRAPL\PRINTINF.TXT)
[印刷情報ファイル(C:\PRAPL\PRINTINF.TXT)の内容]
```

STREAM=LW  
 STREAMENV=C:¥LW.TXT  
 PRTOU=PRNAME: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、List Works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。PRTOUにList Works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。

### 8.6.3 電子帳票の出力例



### 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)

プリンタ(紙)出力時と電子帳票出力時の機能差について、以下に示します。

**実行環境情報を利用する機能**

各環境変数情報の詳細は、“付録C 環境変数情報”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
環境変数情報	@CBR_DocumentName_xxxx	○	×	印刷情報ファイルのDocumentNameパラメータを指定してください。

	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
	@CBR_OverlayPrintOffset	○	×	常に“VALID”が指定された場合と同じ状態になります。
	@CBR_PrinterANK_Size	○	○	
	@CBR_PrintFontTable	○	○	
	@CBR_PrintInfoFile	○	○	
	@CBR_PrintTextPosition	○	×	常に“TYPE2”が指定された場合と同じ状態になります。
	@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	○	○	
	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、“8.1.12 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	○	○	
	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	× (*2)	○	
	PRTOUT (*3)	○	○	
	PAPERSIZE	○	○	
	PRTFORM	○	○	
	FOVLDIR	○	×	
	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、List Works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	List Works
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。

List Worksを利用してFAX送付を行う場合、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、紙に印刷されるとき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	List Works
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(List Worksの制限事項)

List Works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。List Worksの制限事項およびその解除に関する詳細は、List Worksのマニュアルを参照してください。

### 8.6.5 実行時エラーについて

List Works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
```

List Works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。“Interstage List Worksメッセージ集”を参照し、“詳細コード”の説明から\$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。

#### 注意

CODE=2000が出力された場合、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バイトコード系が混在しています。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います)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/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。

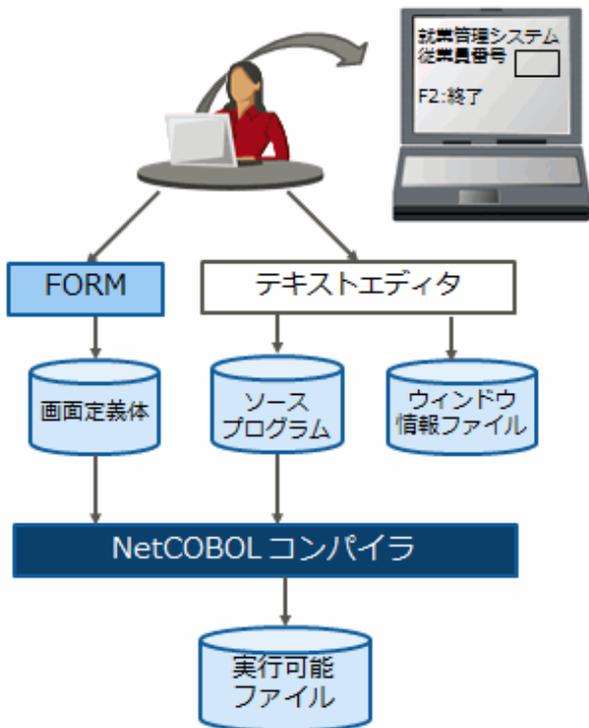
#### 9.2.1 概要

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

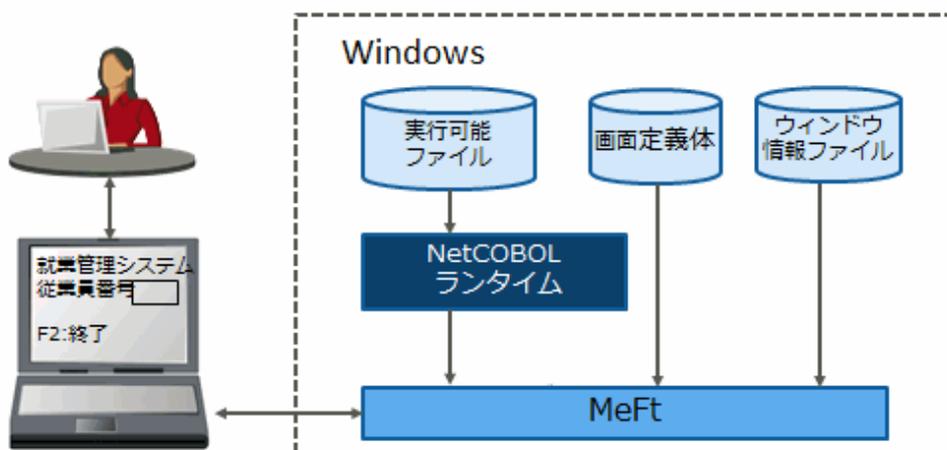


表示ファイル機能を実際に使用するときには、画面定義体およびMeFtが必要です。

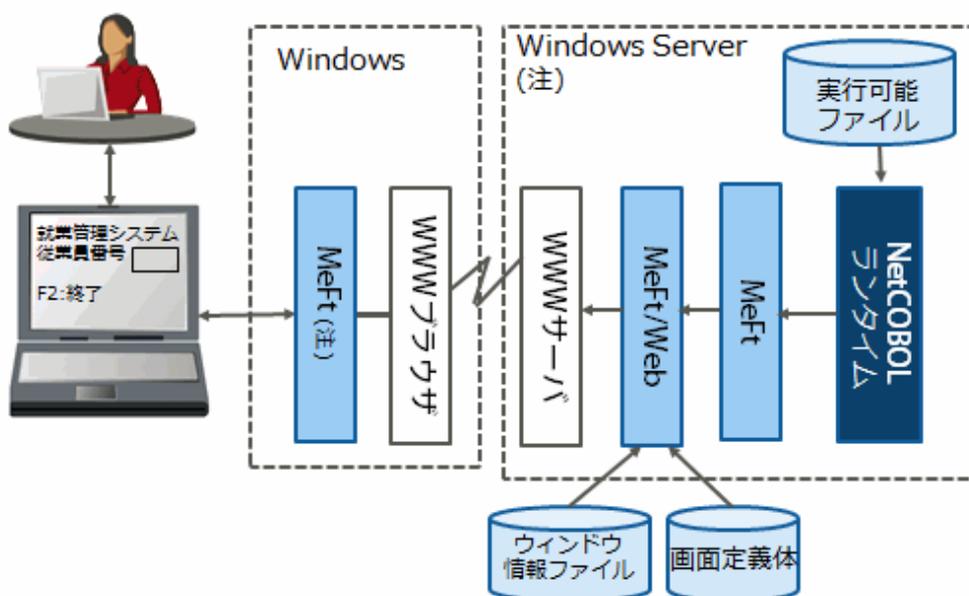
これらの関連図を以下に示します。

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(MeFt)



リモート環境で使用する場合 (MeFt/Web)



注：サーバから自動ダウンロードされます。

MeFt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に、ウィンドウ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。

情報の種類		指定する内容および用途
	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) 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D 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)	
	通常	制御	メニュー項目	アテンション情報
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"		
	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ユーザーズガイド”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

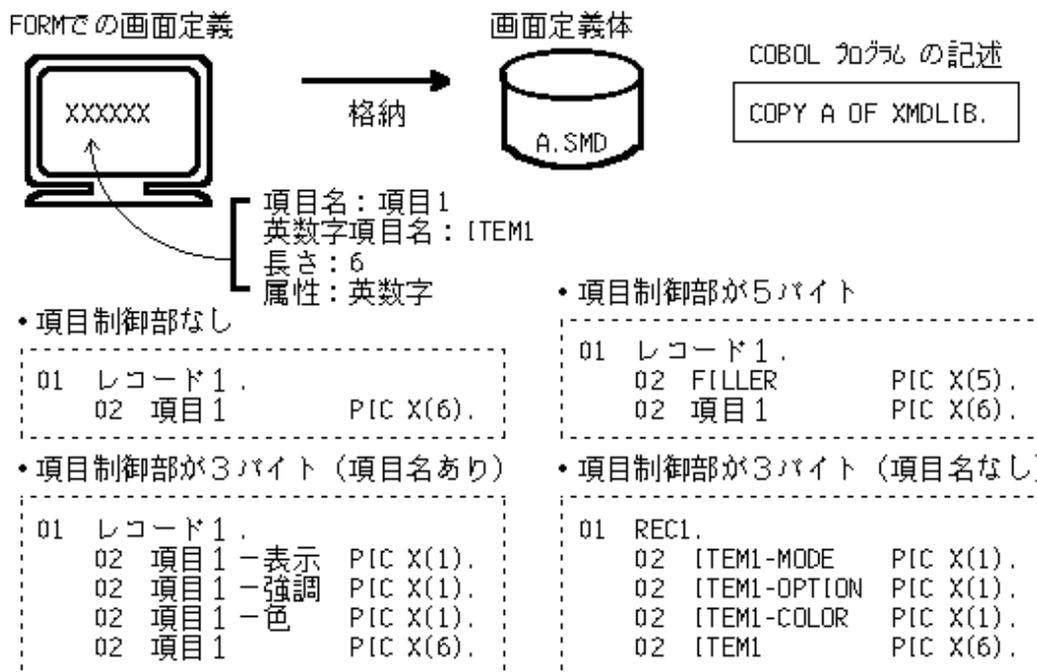
アテンション情報の値	アテンション種別
カスタマイズした値 (注1)	画面定義時に利用者が指定した値
"C000"	クリアキー
"E000"	実行キー (注2)
"E000"	データフルキー
"E000"	項目脱出キー

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、ウイ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ユーザーズガイド”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、“10.2.5.1 外部データ使用時の注意事項”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ユーザーズガイド”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ユーザーズガイド”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ユーザーズガイド”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.3.21 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.3.20 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細な内容や作成方法については、“MeFtユーザーズガイド”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以下に示します。

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。

- 環境変数MEFTDIRに設定したフォルダ
- カレントフォルダ

### MeFt/Webを使用する場合

MeFt/Webを使用する場合には、“MeFt/Webユーザーズガイド”を参照してください。

### 表示ファイルの入力中断

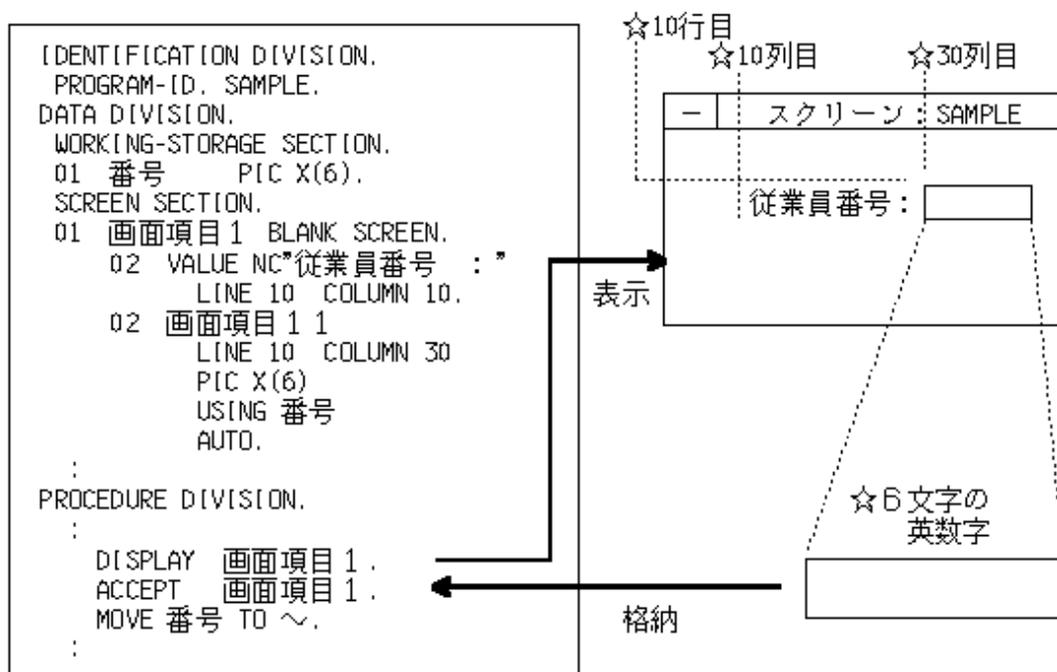
入力待ち状態の画面を、他のアプリケーションからMeFtの関数を直接呼び出して、入力中断することができます(MeFtを使用する場合)。詳細については、“12.16 JMPBGWDH(ウィンドウハンドル獲得サブルーチン)”を参照してください。

## 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。

- “C.2.72 @ScrnSize (スクリーン操作の論理画面の大きさの指定)”
- “C.2.77 @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 (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、“[C.2.46 @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 △	0 0 0 0 △
	"1"を入力	1 0 0 0 △	1 0 0 0 △
	"2"を入力	1 2 0 0 △	1 2 0 0 △
	". "を入力	1 2 0 0 △	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 △	□□00. 00 △
	"1"を入力	1□00. 00 △	□□10. 00 △
	"2"を入力	1 2 0 0. 00 △	□□12. 00 △
	"3"を入力	1 2 3 0. 00 △	□123. 00 △
	"4"を入力	1 2 3 4. 00 △	1 2 3 4. 00 △
	"5"を入力	1 2 3 4 5 0 0 △	1 2 3 4. 5 0 △
	"6"を入力	1 2 3 4 5 6 0 △	1 2 3 4. 5 6 △
	"ENTERキー"を入力	4 5 6 0. 0 0	1 2 3 4. 5 6

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となります。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、行の先頭にセミコロン(;)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.  
    [CURSOR 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"－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"－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)
"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	×	○	△	○	○	○

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
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

×

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] “C.2.46 @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。
  - F10キー(システムメニューの表示)
  - ALT+F4キー(アプリケーションの終了)
  - 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。
  - CTRL+ESC
  - ALT+ESC
  - CTRL+TAB
  - ALT+TAB

## 9.3.7 Unicodeデータの扱い

スクリーン操作機能の日本語項目に対してUTF-32は使用できません。

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。

## 注意

サロゲートペアの文字は使用できません。

## 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。

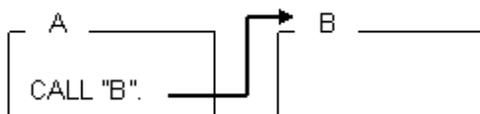
### 10.1 呼出し関係の概要

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

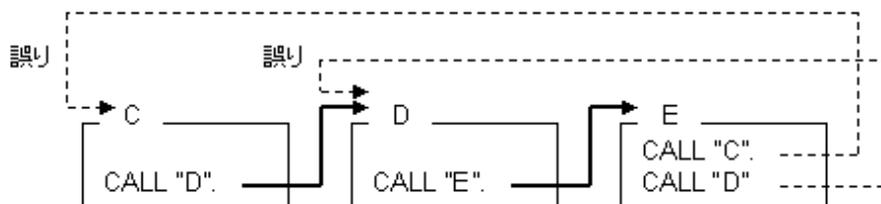
COBOLプログラムは、他のプログラムを呼び出したり、他のプログラムから呼び出されたりすることができます。

図10.1 呼出し関係の形態(a)



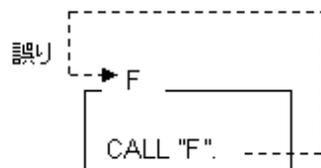
他のプログラムは、他言語で記述されたプログラムでもかまいません。ただし、COBOLプログラムを再帰的に呼び出すことはできません。

図10.2 呼出し関係の形態(b)



また、COBOLプログラム自身を呼び出すこともできません。

図10.3 呼出し関係の形態(c)



#### 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		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。
STDCALL呼出し規約			

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“10.3 C言語プログラムとの結合”に示します。

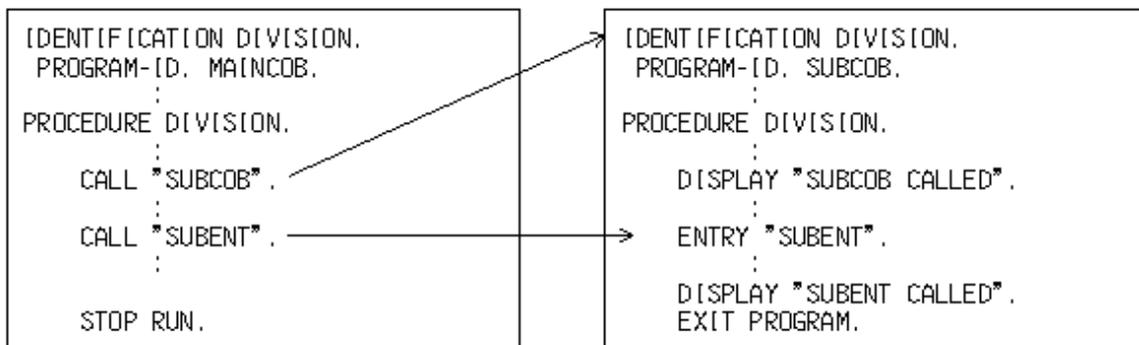
表10.2 呼出し規約と指定形式

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10.4 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10.5 COBOLプログラムからCプログラムを呼び出す場合

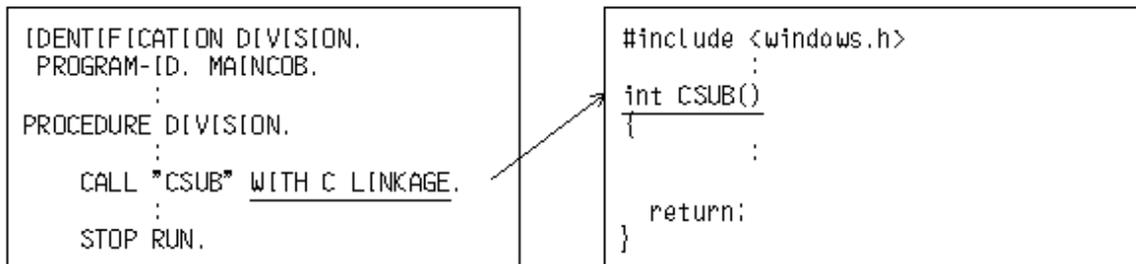
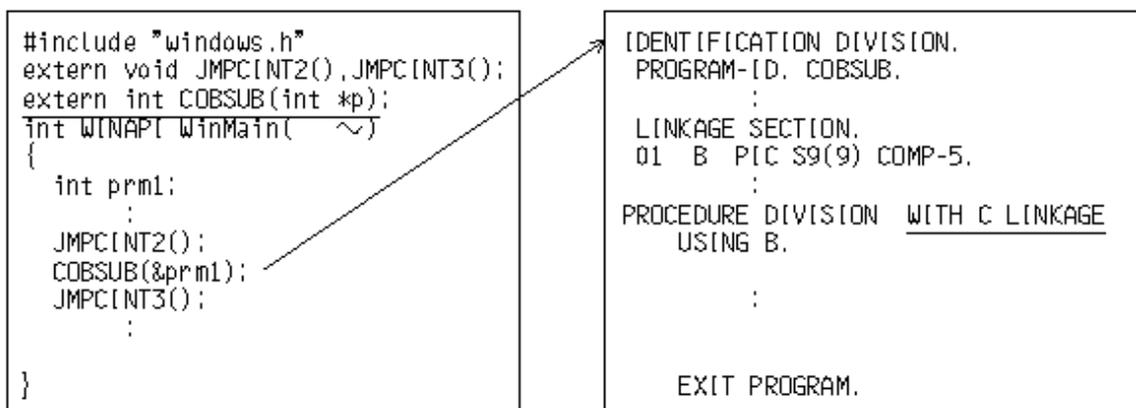
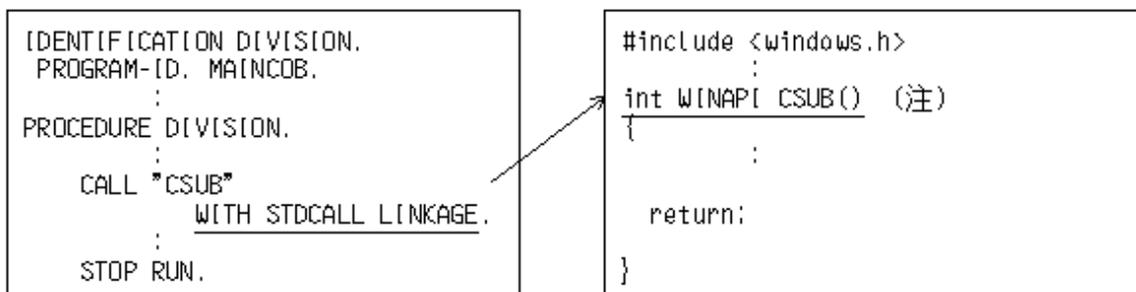


図10.6 CプログラムからCOBOLプログラムを呼び出す場合



## STDCALL呼出し規約

図10.7 COBOLプログラムからCプログラムを呼び出す場合

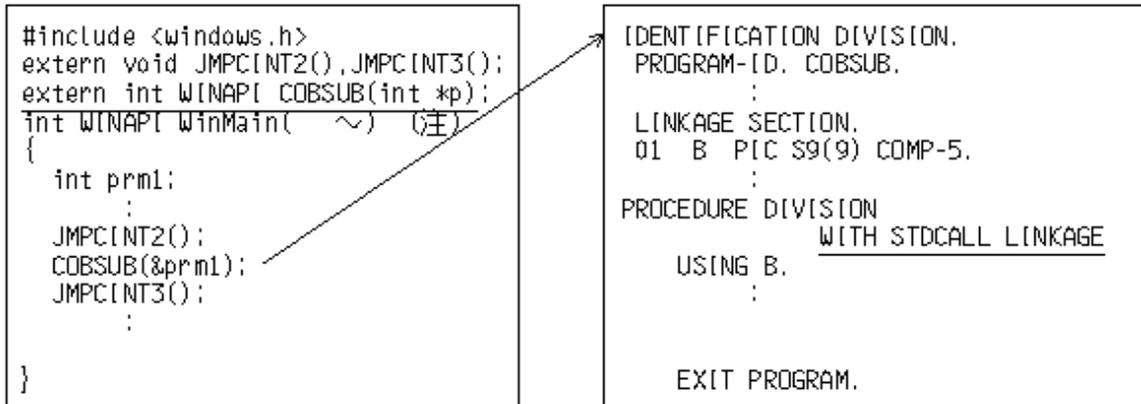


注：関数名の宣言が以下の場合、STDCALL呼出し規約となります。

- int WINAPI CSUB()
- int CALLBACK CSUB()
- int PASCAL CSUB()

- int far pascal CSUB()
- int \_stdcall CSUB()

図10.8 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、“20.3.1 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えます)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プログラムの呼び出しの後で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、“[I.3 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10.9 COBOLプログラムだけ

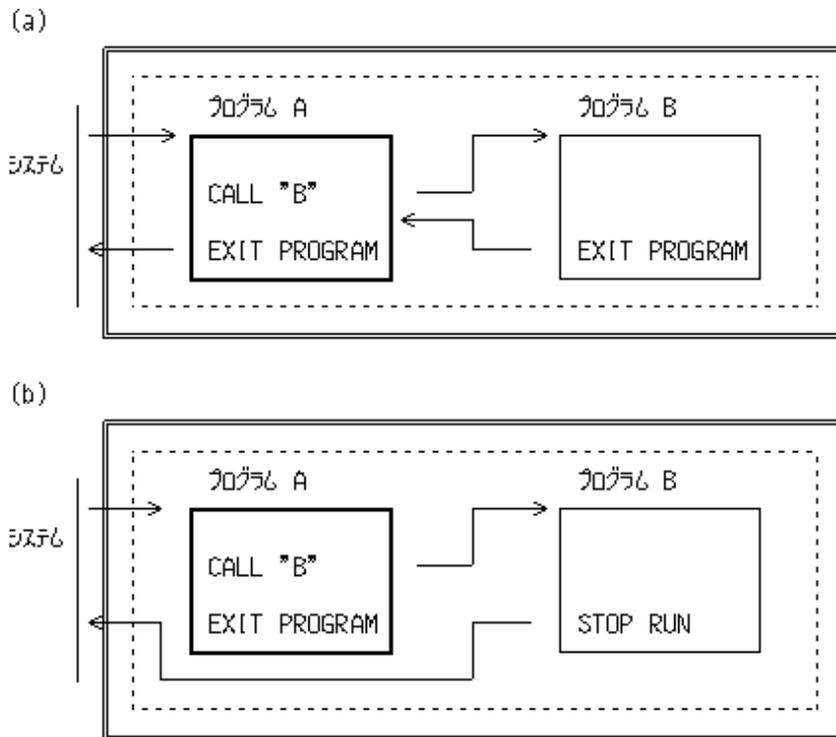
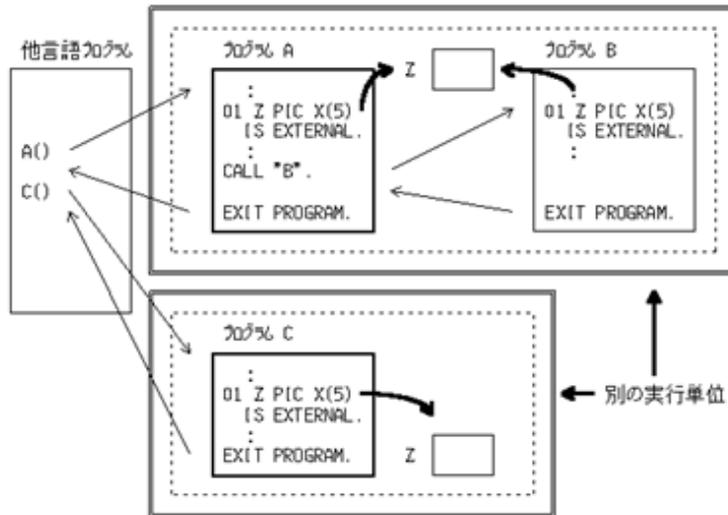
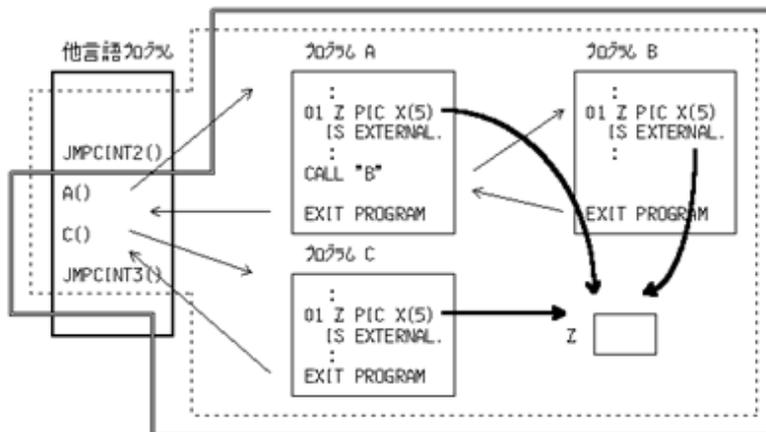


図10.10 他言語プログラムからCOBOLプログラムの呼出し

(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。

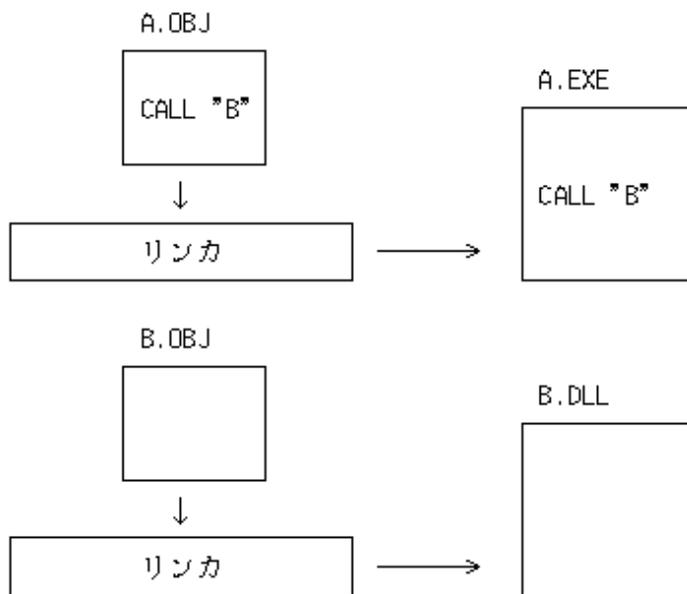
## 10.1.5 動的プログラム構造

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。

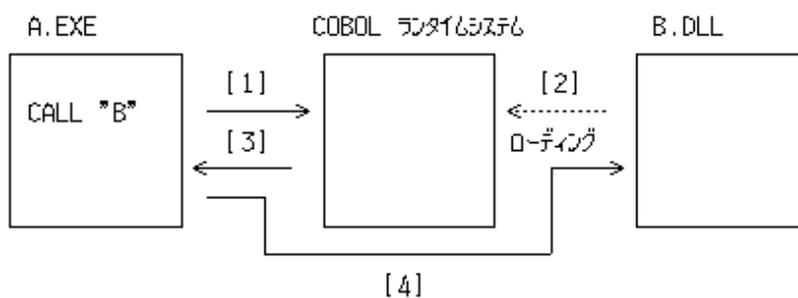
### 10.1.5.1 動的プログラム構造の特徴

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、“20.10.5 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、“5.6 エントリ情報”を参照してください。

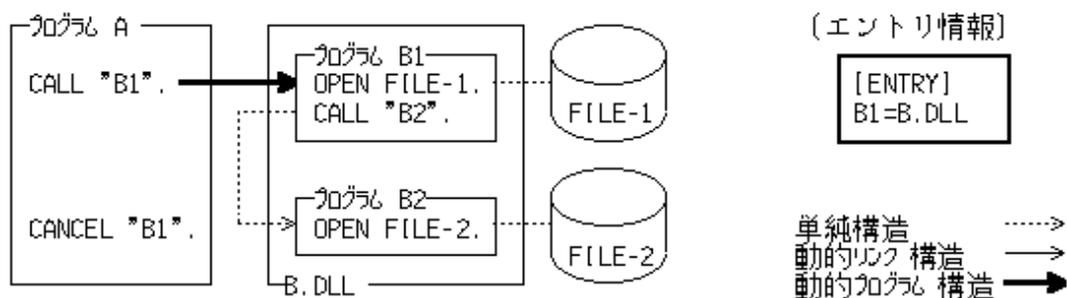
### 10.1.5.3 注意事項

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。

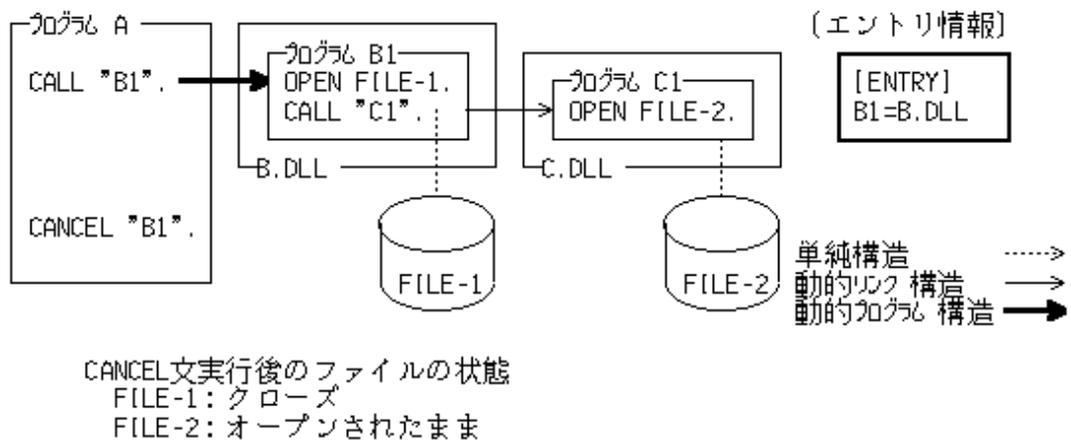
CANCEL文を実行した場合、DLLが仮想メモリから削除されることにより、CANCEL文に指定された副プログラムが呼び出している副プログラムの延長上で、オープンされたファイルおよびカーソルがオープンされたままになることがあります。この場合の動作については保証されませんので、副プログラムでオープンしたファイルおよびカーソルは、CANCEL文の実行前に必ずクローズしてください。

図10.11 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



CANCEL文実行後のファイルの状態  
 FILE-1: クローズ  
 FILE-2: オープンされたまま

図10.12 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



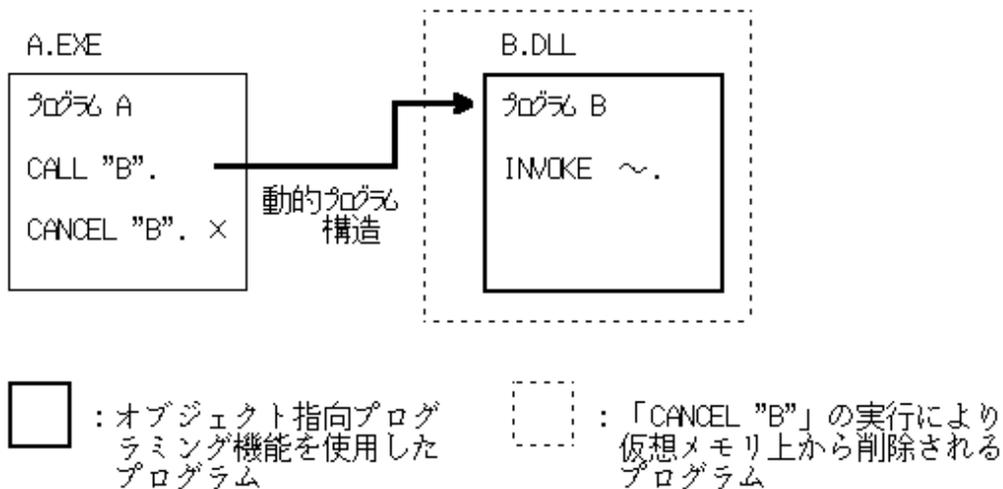
参考

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。

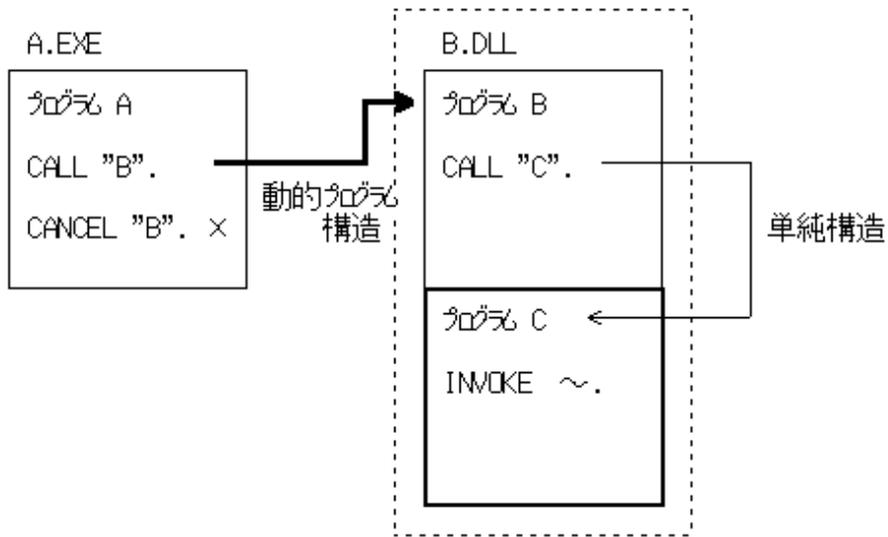


[例2] 以下の場合、CANCEL文の実行により、プログラムBが仮想メモリ上から削除されると、プログラムCも削除されるため、このCANCEL文は使用できません。

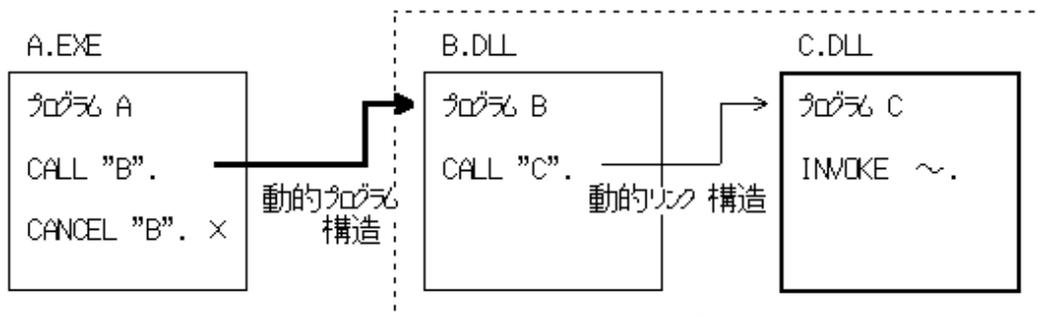
- プログラムBとオブジェクト指向プログラミング機能を使用したプログラムCが単純構造(下図(a))
- 動的リンク構造(下図(b))

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。

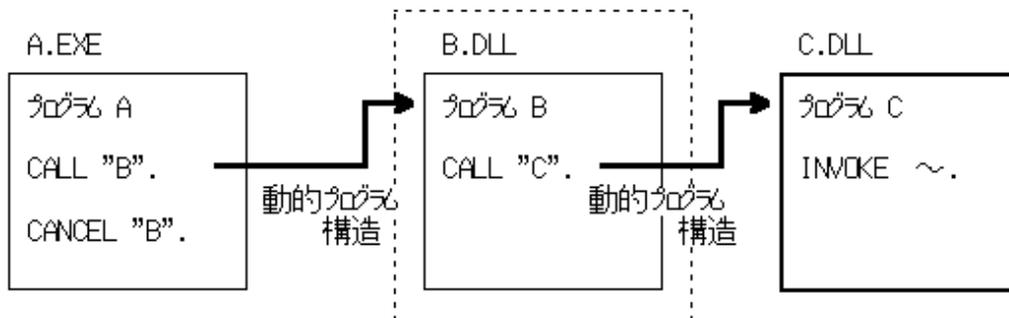
(a) プログラムBとプログラムCが単純構造



(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と同様に指定します。

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。

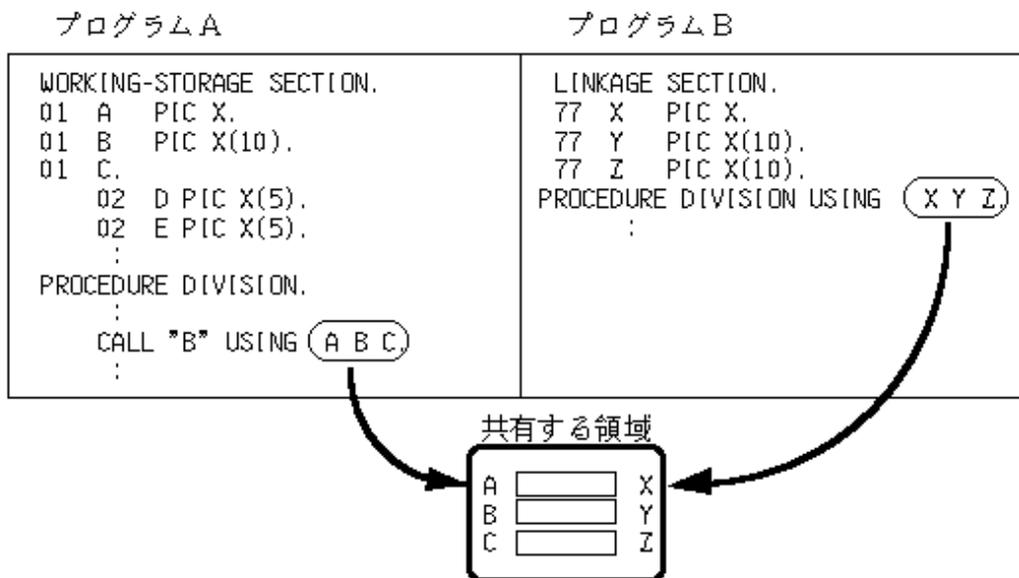
### 10.2.4 パラメタの受渡し方法

---

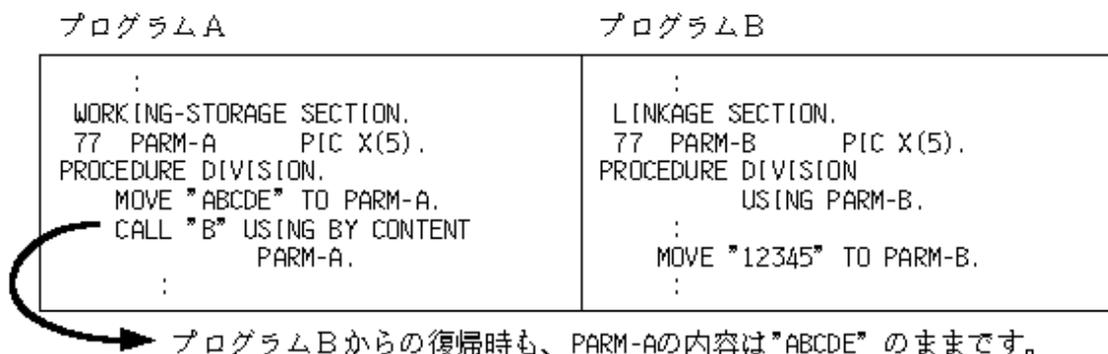
呼ぶプログラムと副プログラムの間で、パラメタを受け渡すことができます。

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、“21.3 CHECK機能”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、“21.3 CHECK機能”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。

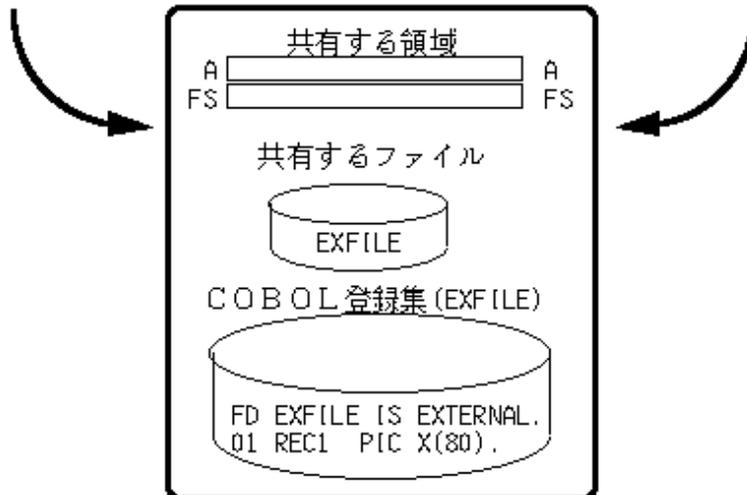
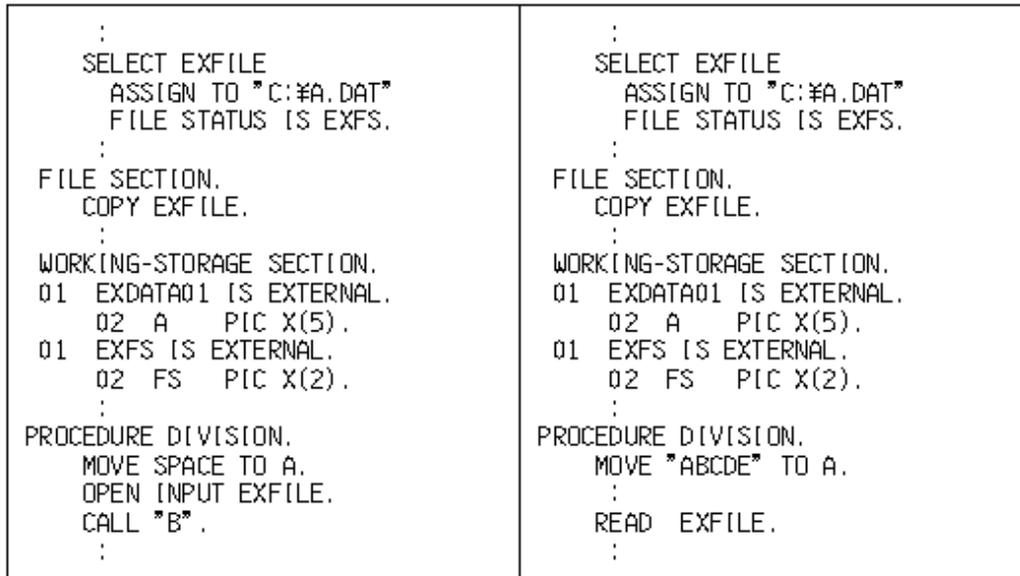
この機能を用いることにより、プログラム間で引数(パラメタの受渡しの機能)を使用しなくてもデータの受渡しが可能になります。ただし、領域の参照/設定の関係が分かりにくくなるため、利用には注意が必要です。

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

プログラムB



### 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、データを共有するプログラム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(通常的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。



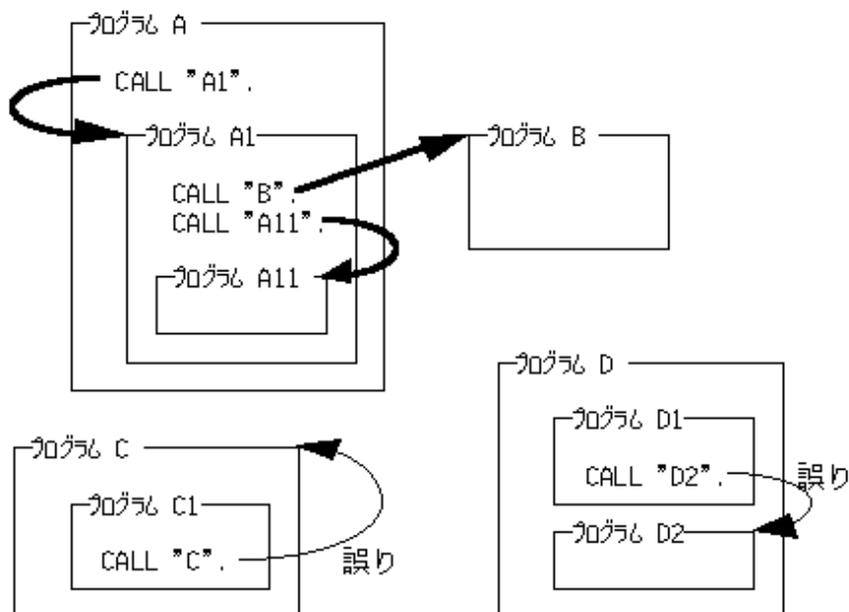
注意

COBOLプログラムでSTOP RUN文を使用する場合は、COBOLプログラムにはRETURNING指定を記述しないようにしてください。RETURNING指定を記述した場合、RETURNING指定に指定した復帰値の型によっては、EXIT PROGRAM文で復帰した場合(RETURNING指定の値)とSTOP RUN文実行時(特殊レジスタPROGRAM-STATUSの値)とで復帰値の型が異なり、正しく復帰値が取得できない場合があります。

## 10.2.7 内部プログラム

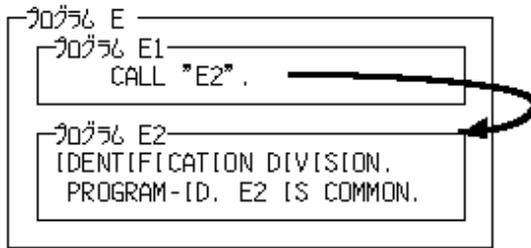
COBOLプログラムは、プログラムの構造の観点から、外部プログラムと内部プログラムに分類されます。他のプログラムに含まれない一番外側のプログラムを外部プログラムといいます。外部プログラムに直接的または間接的に含まれているプログラムを内部プログラムといいます。

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。



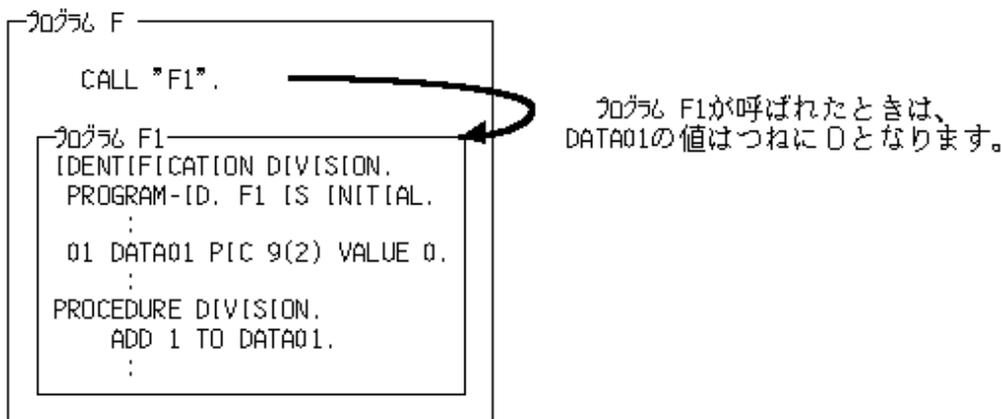
### 共通プログラム

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.3.2 ALPHAL(英小文字の扱い)”
  - 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。
  - 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.3.30 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 MSVCRT. 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 MSVCRT. 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 関数值 = 0;  
    // :  
    return(関数值);  
}
```

### 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

#### — 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.
```

#### — 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);  
}
```

#### — 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ータは、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ータは、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

#### — 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.

```

— 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);
}

```

— 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となります。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。

ただし、Cプログラムから内部浮動小数点項目(単精度と倍精度)の復帰コード(関数値)を受け取ることはできません。内部浮動小数点項目は必ずUSING指定に記述してください。

• 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

#### — 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  
*>      :
```

#### — Cプログラム

```
#include <windows.h>  
  
typedef struct  
{  
    char aitem1[10];  
    char aitem2[20];  
} agrp;  
  
long int C( agrp *agrpp, short int *b)  
{  
    return(0);  
}
```

### STDCALL呼出し規約の場合

#### — 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  
*>      :
```

#### — Cプログラム

```
#include <windows.h>  
  
typedef struct  
{  
    char aitem1[10];  
    char aitem2[20];  
} agrp;  
  
long int WINAPI C( agrp *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 JMPCINT2(), JMPCINT3();
  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。

ただし、Cプログラムに内部浮動小数点項目(単精度と倍精度)の関数値を渡すことはできません。内部浮動小数点項目(単精度と倍精度)はCの関数呼出しで実引数に指定してください。

- 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;

```

```

:
JMPCINT2();
if (COB(&prm1, &prm2)==0)
:
JMPCINT3();
}

```

- 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 JMPCINT2(), JMPCINT3();
extern long int COB(agrp *agrpp, short int *b);
agrp prm1;
short int prm2;
:
JMPCINT2();
if (COB(&prm1, &prm2)==0)
:
JMPCINT3();
}

```

- 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。

#### 注意

- 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) SIGN IS LEADING SEPARATE.	char C[6];	6バイト
		01 D PIC S9(9) SIGN IS TRAILING SEPARATE.	char D[10];	10バイト
2進 (注2)(注3)(注5)	unsigned char	01 E USAGE IS BINARY-CHAR UNSIGNED.	unsigned char E;	1バイト
	short int	01 F PIC S9(4) COMP-5.  あるいは 01 F USAGE IS BINARY-SHORT SIGNED.	short int F;	2バイト
	long int	77 G PIC S9(9) COMP-5.  あるいは 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. 02 H1 PIC S9(4) COMP-5. 02 H2 PIC X(4).	struct{ short int H1; char H2[4]; } 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。

- 1～4桁(BINARY(BYTE)オプション指定時は3～4桁) : short int
- 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。

— 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.
```

— 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。

・ 注5

Cプログラムから領域のアドレスを受け取る時は、ポインタデータ項目を使用してください。

 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 MSVCRT.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.3.2 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 JMPCINT2(), JMPCINT3();
extern int COBSUB(int *p);
int WINAPI WinMain( ~ )
{
    int prm1;
    :
    JMPCINT2();
    COBSUB (&prm1);
    JMPCINT3();
    :
}
```

- 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 MSVCRT.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  
MSVCRT.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)

#### 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 MSVCRT.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  
MSVCRT.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 MSVCRT.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  
 MSVCRT.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 JMPCINT2(), JMPCINT3();
int (far *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 MSVCRT.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  
 MSVCRT.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 MSVCRT.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  
 MSVCRT.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 JMPCINT2(), JMPCINT3();
extern int WINAPI COBSUB(int *p);
int WINAPI WinMain( ~ )
{
    int prm1=0;
    :
    JMPCINT2();
    COBSUB(&prm1);
    JMPCINT3();
    :
}
```

- COBOLプログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBSUB.
:
```

```
LINKAGE SECTION.
01 B PIC S9(9) COMP-5.
:
PROCEDURE DIVISION WITH STDCALL 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 MSVCRT.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
MSVCRT.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)
```

#### 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 MSVCRT.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
MSVCRT.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 MSVCRT.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
MSVCRT.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 JMPCINT2(), JMPCINT3();
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 MSVCRT.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
MSVCRT.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 MSVCRT.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
MSVCRT.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 MSVCRT.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  
MSVCRT.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)

#### 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 MSVCRT.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  
MSVCRT.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 MSVCRT.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  
MSVCRT.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 MSVCRT.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  
MSVCRT.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 MSVCRT.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  
MSVCRT.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 MSVCRT.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  
 MSVCRT.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)

#### 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 MSVCRT.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  
 MSVCRT.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 MSVCRT.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  
 MSVCRT.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 MSVCRT.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  
 MSVCRT.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

“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 MSVCRT.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  
 MSVCRT.LIB : Cランタイムライブラリ

## 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。

データ型	Visual Basic		COBOL
	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に戻します。

Visual Basicのコード

外部プロシージャへの参照の宣言

```
Private Declare Sub sample13 Lib "sample13.DLL" Alias "SAMPLE13" (ar%, br%, ByVal C As String)
Private Declare Sub JMPCINT2 Lib "f3biprct.dll" ()
Private Declare Sub JMPCINT3 Lib "f3biprct.dll" ()
```

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。

#### JMPCINT2の呼出し

```
Private Sub Form_Load()
    Call JMPCINT2
End Sub
```

#### JMPCINT3の呼出し

```
Private Sub Form_Unload(Cancel As Integer)
    Call JMPCINT3
End Sub
```

#### COBOLプログラムの呼出し

```
Private Sub Command1_Click()

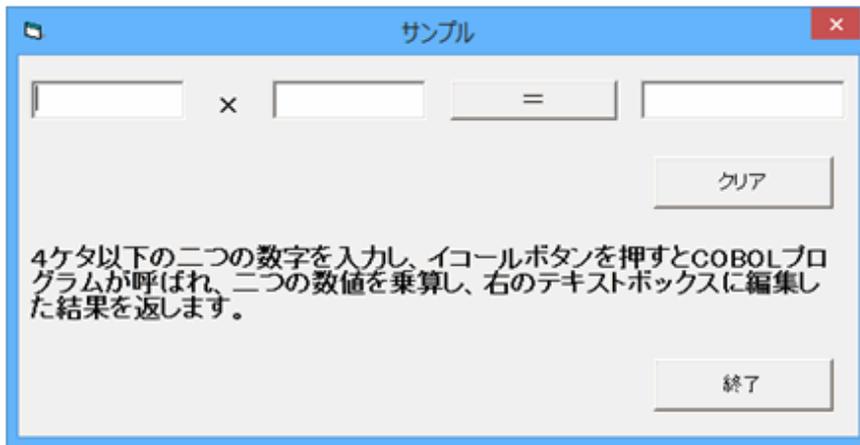
    Static C As String * 11
    a = Text1.Text
    ar% = Val(a)
    b = Text2.Text
    br% = Val(b)
    Call sample13(ar%, br%, C)
    Text3.Text = C

End Sub
```

#### 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 STDCLASS LINKAGE USING DATA1 DATA2 DATA3.
000120 COMPUTE WKDATA = DATA1 * DATA2
000130 MOVE WKDATA TO DATA3
000140 EXIT PROGRAM.
```

Visual Basicのフォーム



# 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。

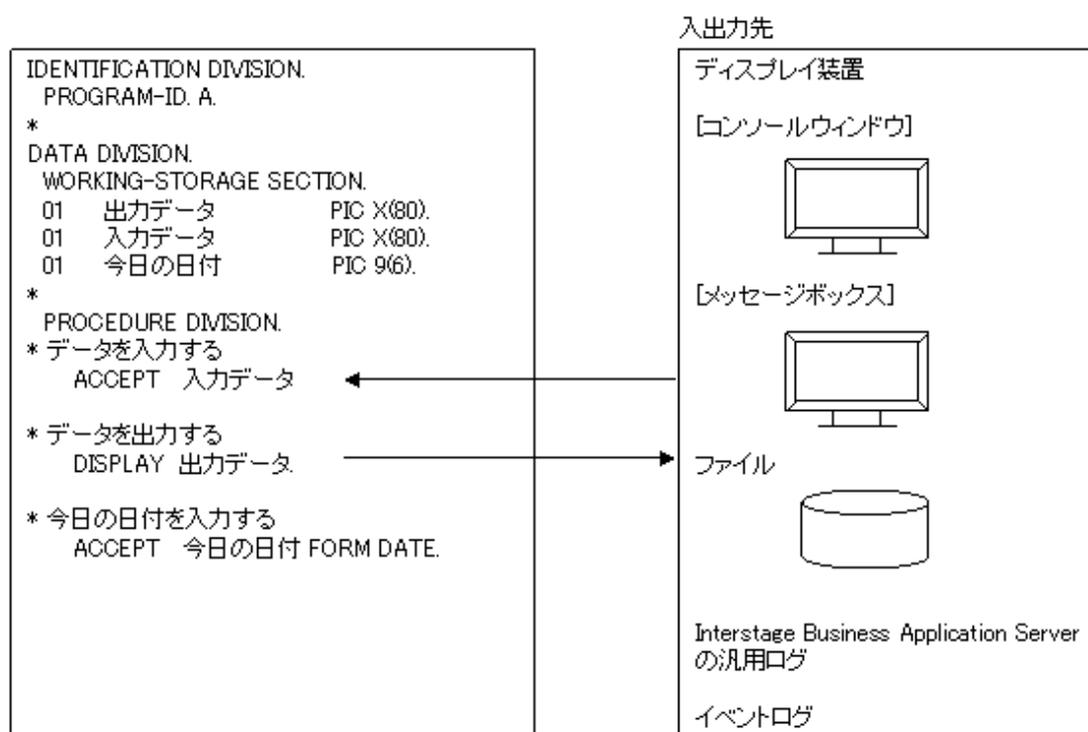
## 11.1 小入出力機能

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) 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_SYSOUT 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_SYSOUP OUTPUTにEVENTLOGを設定	イベントログ(注6)(注7)
[2]	機能名SYSERRに対応付けた呼び名	—	@MessOutFileに空白を設定	ディスプレイ装置(メッセージボックス、コマンドプロンプトまたはシステムのコンソール)(注3)
			@MessOutFileにファイル名を設定	ファイル
			@CBR_COMPOSER_SYSERR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5)
			@CBR_DISPLAY_SYSERR_OUTPUTにEVENTLOGを設定	イベントログ(注7)
[3]	機能名CONSOLEに対応付けた呼び名	—	—	ディスプレイ装置(コンソールウィンドウ)(注2)
			@CBR_COMPOSER_CONSOLE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)
			@CBR_DISPLAY_CONSOLE_OUTPUTに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 Unicodeデータの扱い

ACCEPT文およびDISPLAY文を使ってUnicodeのデータを入出力できます。ただし、作用対象が日本語を含む集団項目の場合には注意が必要です。

```

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、文字化けを起こします[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、[翻訳オプションENCODE](#)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。



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]COBOLファイルの“7.1.4 Unicodeデータの扱い”

## 11.1.4 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。

### 11.1.4.1 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 (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。

属性	環境変数情報	設定値	意味
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。

### 11.1.4.2 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力関数を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。



#### 注意

- 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。
  - 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
  - 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。

### 11.1.4.3 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。



#### 注意

- 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。



#### 例

```
START COBOL-APL.EXE
```

- システムのコンソールに出力される文字列はシステムコード系である必要があるため、実行時の出力文字列がUnicodeの場合、UnicodeからシフトJISへのコード変換が行われます。このとき、変換先(シフトJIS)に存在しないUnicodeの文字は、コード変換時に変換エラーが発生する場合があります。

#### 11.1.4.4 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。

#### 11.1.4.5 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。

## 11.1.4.6 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。

## 11.1.5 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。

### 11.1.5.1 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。

### 11.1.5.2 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。

### 11.1.5.3 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。

### 11.1.5.4 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]“[C.2.65 @MessOutFile\(メッセージを出力するファイルの指定\)](#)”



- 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。

## 11.1.6 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。

## 11.1.6.1 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)

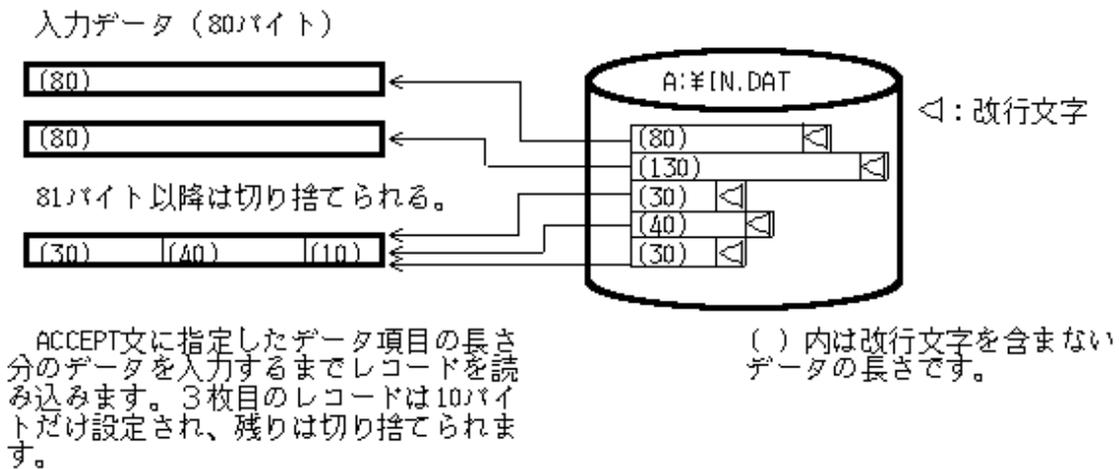
注：ACCEPT文では使用不可

### 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。

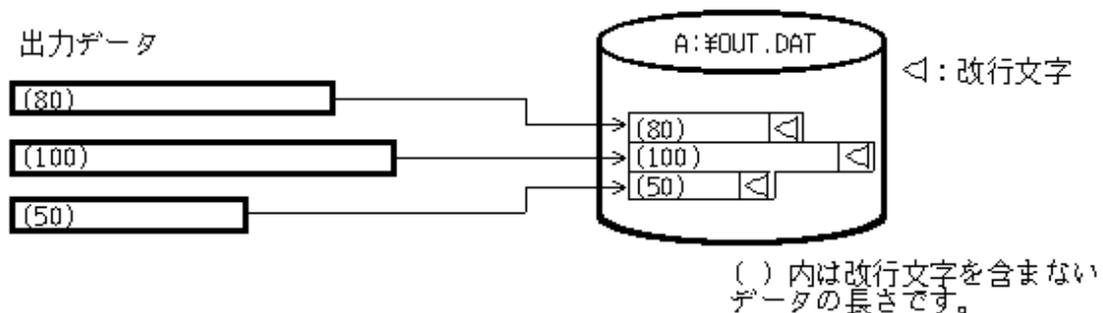
#### データの入力

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。



### 11.1.6.2 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.3.56 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.3.57 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となります。

### 11.1.6.3 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。

### 11.1.6.4 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らみなされます。
- 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。

## 11.1.6.5 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。

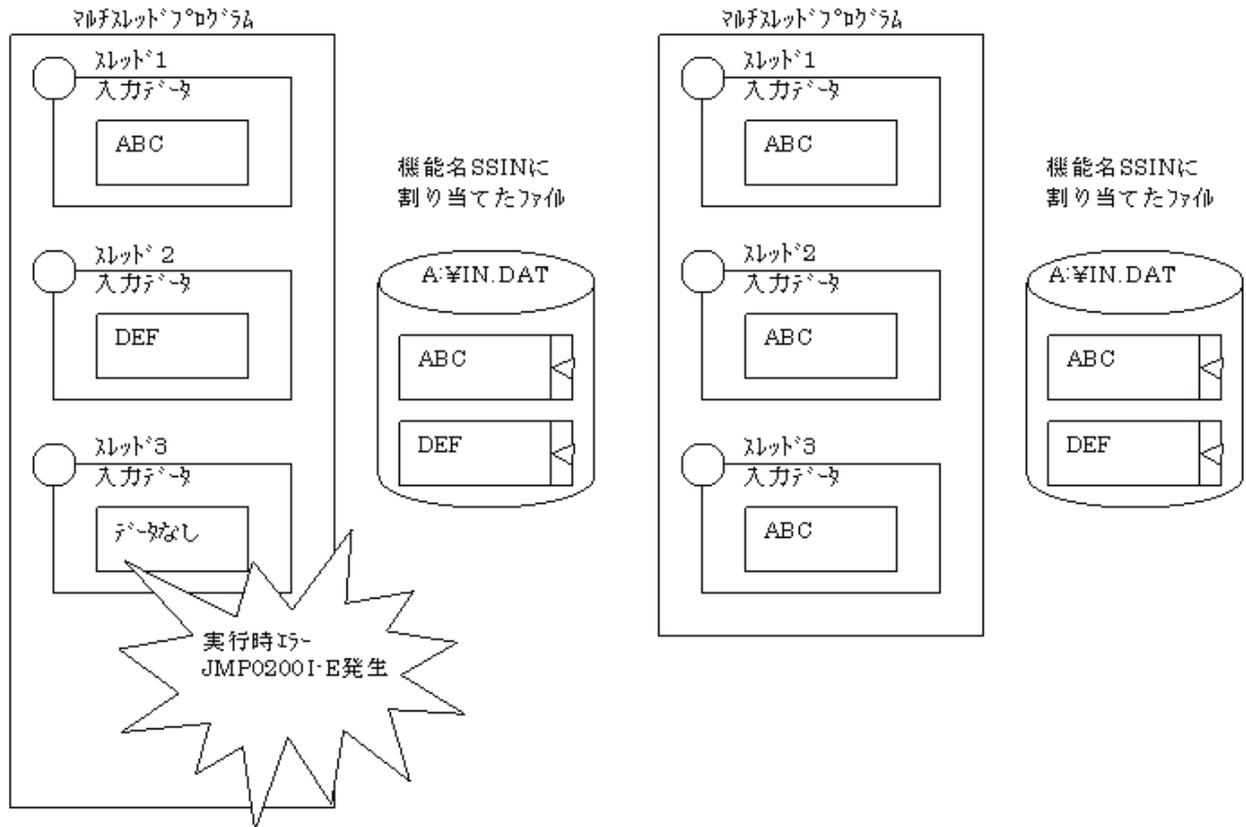
## スレッド単位でのファイルオープン

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。

@CBR\_SSIN\_FILE=THREAD指定時



## 使い方

実行環境変数C.2.48 @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)に、“THREAD”を指定します。



例

@CBR\_SSIN\_FILE=THREAD

## 11.1.7 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。

### 11.1.7.1 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。

### 11.1.7.2 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。

### 11.1.7.3 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、“[CURRENT-DATE関数を利用した西暦の取得](#)”を参照してください。

## 11.1.8 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。

### 11.1.8.1 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。

### 11.1.8.2 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

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

## 11.1.9 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。

### 11.1.9.1 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。

### 11.1.9.2 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
```



DISPLAY文の汎用ログへの出力レベルは5です。

## 11.1.10 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。

### 11.1.10.1 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。

### 11.1.10.2 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。

### 11.1.10.3 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)
```



- 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。

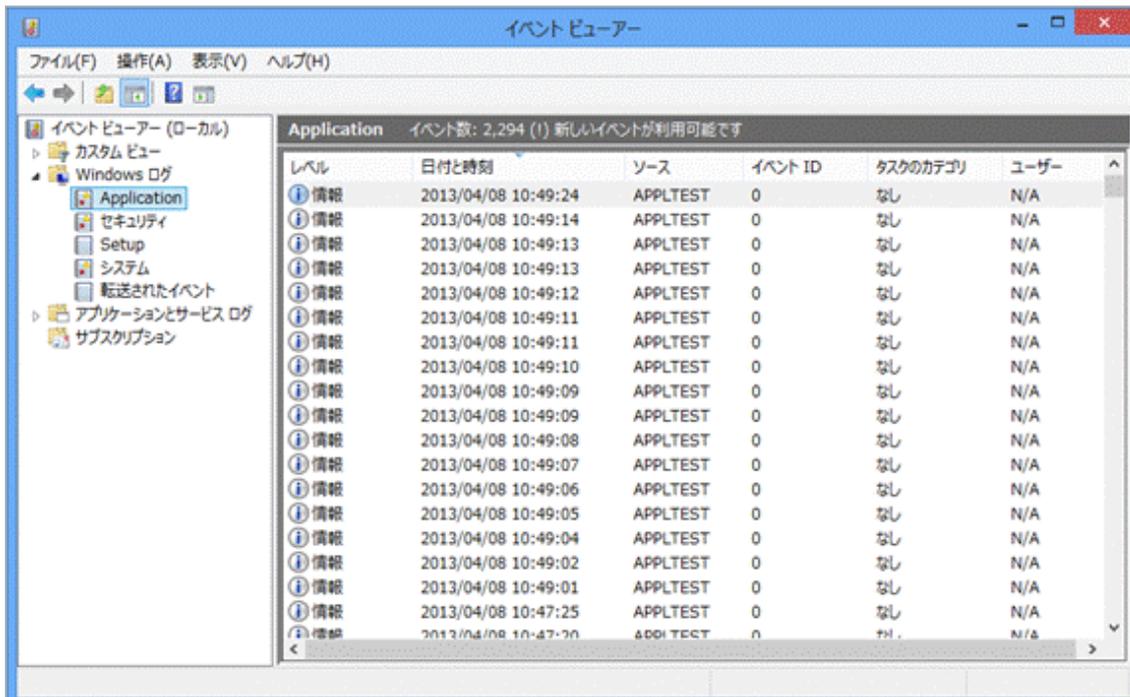
I : 情報イベント  
W : 警告イベント  
E : エラーイベント

- 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“[18.3.2 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



## 11.2 コマンド行引数の取出し

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。

### 11.2.1 概要

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.
```

### 注意

- 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。



### 注意

- ・ 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つた場合、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～

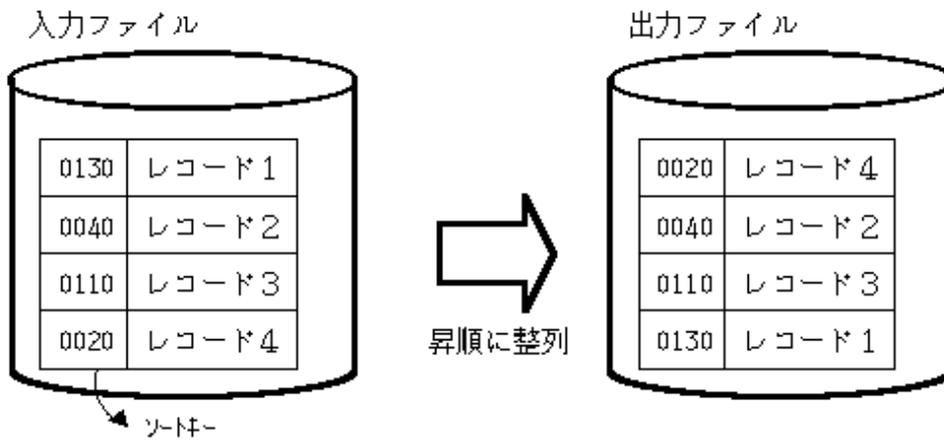
ファイルのレコードを一定の順序に並べ替えることをソート(整列)といい、複数のファイルを1つのファイルに並べ替えることをマージ(併合)といいます。本章では、ソート・マージ(整列・併合)機能について説明します。

### 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。

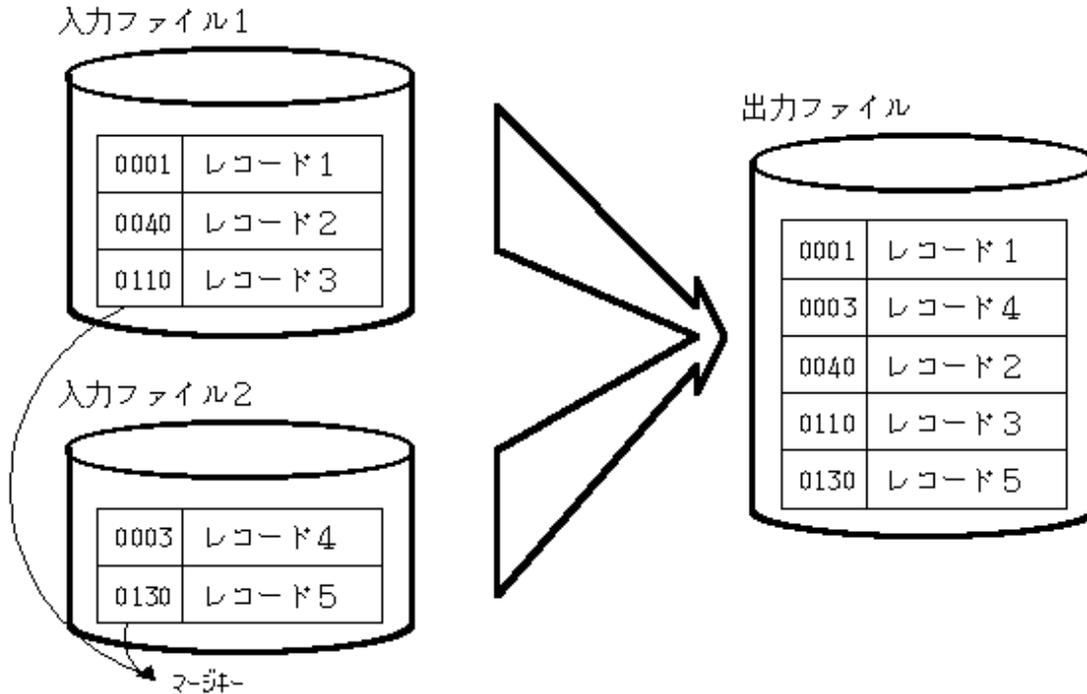
#### ソート

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文実行時にこれらのファイルが開かれた状態であってはけません。

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、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.3.16 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。

## 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

## 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。

### 注意

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

### 注意

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、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. 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。
2. プログラムの実行  
プログラムを実行します。

## 第13章 CSV形式データの操作

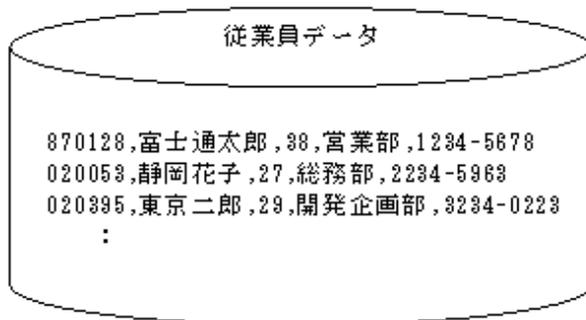
本章では、STRING文およびUNSTRING文を使用したCSV形式データの操作について説明します。

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きました。

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-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-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-R 年齢 所属-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。

- 送出し側項目を以下のように変換します。
  - 動作モードがUnicodeのとき、送出し側項目が日本語または英数字の場合、受取り側項目のエンコードに変換します。
  - 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指定が記述されていない場合は、“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。

- 送出し側項目を以下のように変換します。
  - 動作モードがUnicodeのとき、送出し側項目が日本語または英数字の場合、受取り側項目のエンコードに変換します。
  - 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こと、異常発生の原因となった送出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の仕様を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>送出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>送出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>送出し側データを二重引用符で囲みます。</li> <li>送出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、区切り文字のみを転記します。</li> </ul>
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> </ul>

バリエーション	内容
	<ul style="list-style-type: none"> <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@CBR\_CSV\_TYPE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 NC"富士通太郎".
02 所属 PIC N(10) VALUE NC"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時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、繰り返し処理の中で行われる処理の結果を判定し、繰り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(POINTER)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

繰り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文による繰り返し処理の中では、終了条件を判定し、繰り返し処理を脱出する文を記述します。繰り返し処理の中に繰り返し処理を脱出する文がない場合、この繰り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 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。

### 15.1 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。

### 15.2 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。

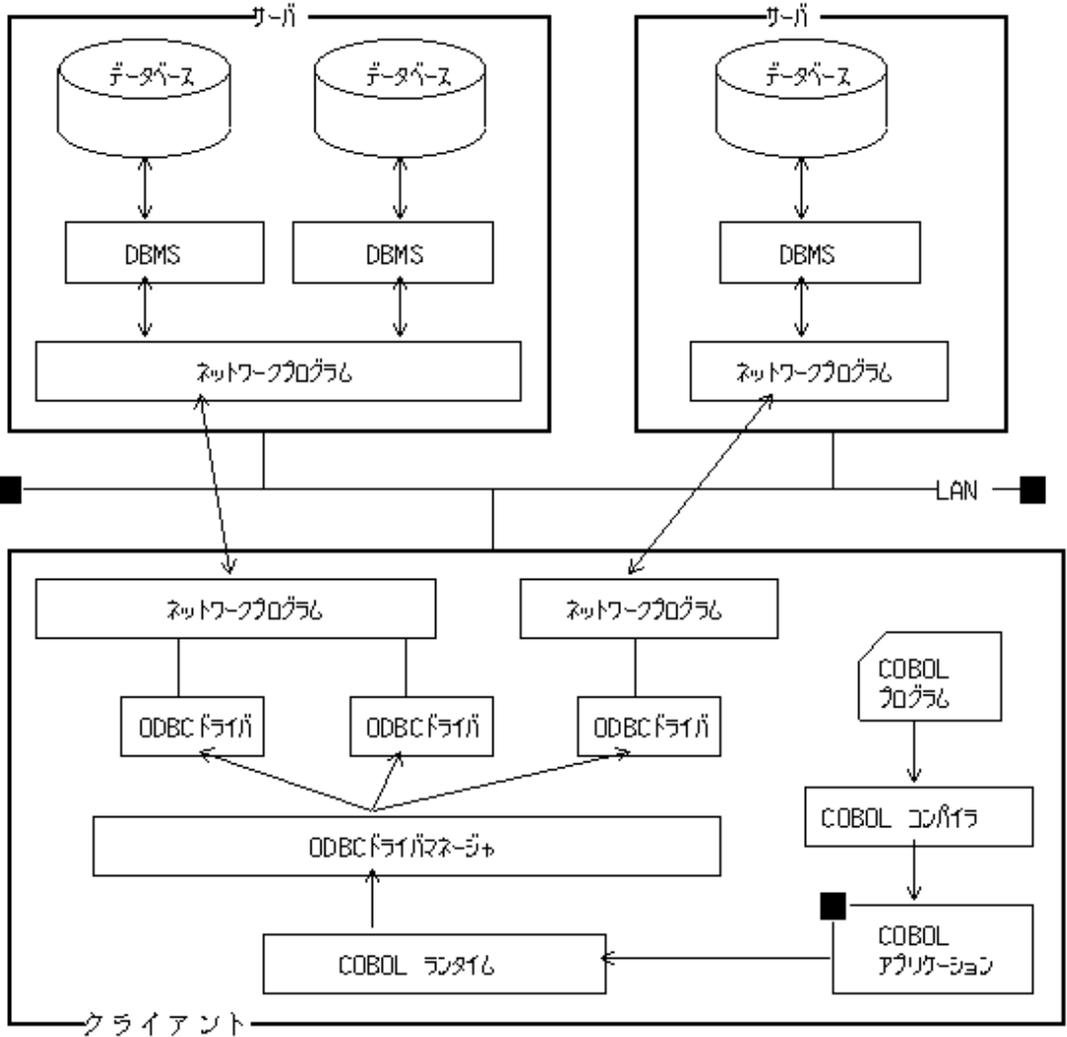
#### 15.2.1 概要

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15.1 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15.1 ODBCドライバを使用したデータベースアクセスの概要図



### 15.2.1.1 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. ----- 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。

## 15.2.1.2 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]“[15.2.2.1 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]“[15.2.2.3 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]“[15.2.3 データ操作](#)”、“[15.2.4 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“[15.2.11 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]“[15.2.5 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“[15.2.11 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なくてはなりません。[参照]“[15.2.2.2 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。

## 15.2.2 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。

### 15.2.2.1 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。

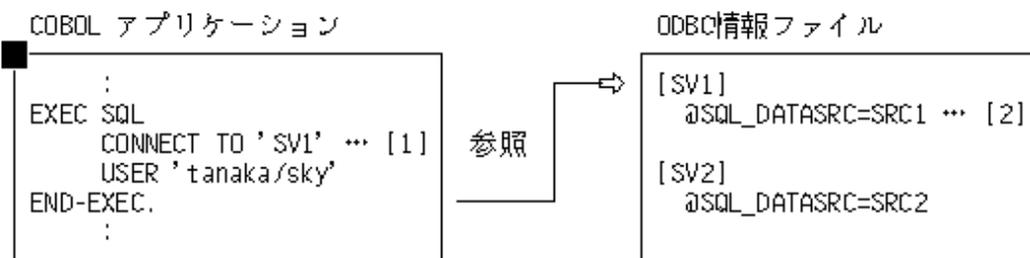
1. サーバの情報を登録します。
2. 以下のどちらかの方法で、サーバとコネクションを接続します。
  - サーバ名を指定して接続する
  - DEFAULTを指定して接続する

#### 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、“[15.2.8 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15.2 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。

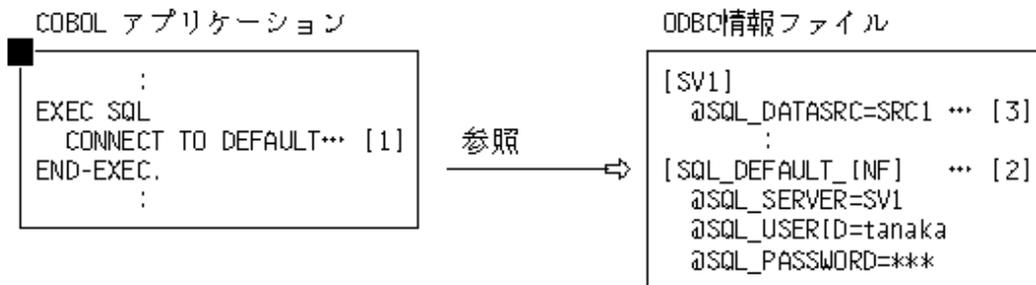
[2] CONNECT文の実行時に、ODBC情報ファイルの情報を参照してサーバとコネクションを接続します。

#### 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、“[15.2.8 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15.3 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、デフォルト接続情報を参照します。

[3] さらに、デフォルト接続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接続を接続します。

### 15.2.2.2 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。

### 15.2.2.3 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オンに接続します。
- [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”が現在のコネクシオンになります。
- [6] コネクシオン名“CNN4”を指定して、コネクシオンを切断します。
- [7] コネクシオン名“CNN1”を指定し、コネクシオンを変更します。“CNN1”が現在のコネクシオンになります。
- [8] コネクシオン“CNN1”中のデータの変更をすべて取り消します。
- [9] 現在のコネクシオンを切断します。本例では現在のコネクシオンは“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オンを変更します。デフォルトコネクシオンが現在のコネクシオンになります。
- [11] デフォルトコネクシオ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オンを切断します。
- [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オンを切断します。

## 注意

CONNECT文により同時に接続できるコネクシオン数は、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。

## 15.2.3 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
- 複数コネクシオンでカーソルを操作する

[参照]“[15.2.11 ODBCで扱うデータとの対応](#)”

### 15.2.3.1 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

### 15.2.3.2 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。

#### 15.2.3.2.1 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.	[1]
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.3.54 SQLGRP (SQLのホスト変数定義の拡張)”

### 15.2.3.2.2 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、サーバとの接続を接続します。

[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、サーバとの接続を切断します。

### 15.2.3.2.3 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。

### 15.2.3.2.4 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表に対して関連名(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、列名を関連名で修飾し、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。

### 15.2.3.3 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.
```



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。

### 15.2.3.4 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.
```



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。

### 15.2.3.5 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。

## 15.2.3.6 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。

## 15.2.3.7 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として値が設定されます。



## 注意

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。

### 15.2.3.8 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。

## 15.2.4 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、“[15.2.3.1 サンプルデータベース](#)”のSTOCK表(在庫表)を使用しています。

### 15.2.4.1 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。



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.3.54 SQLGRP\(SQLのホスト変数定義の拡張\)](#)”

### 15.2.4.1.1 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

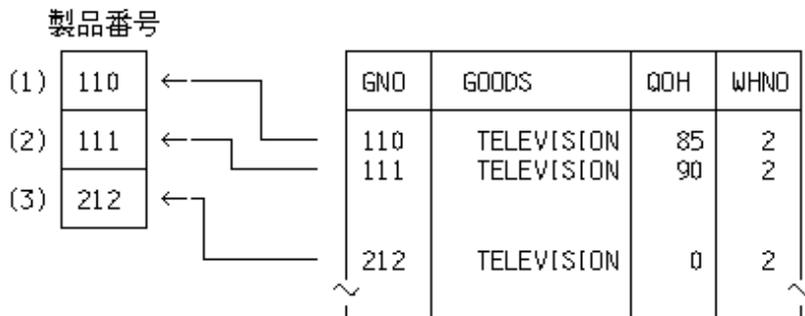
複数行指定ホスト変数を相手指定に使用すると、複数行指定ホスト変数の繰り返し回数と同じ件数のデータを参照することができます。

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'          [2]
  END-EXEC.                             ↓
:
-----
```

[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

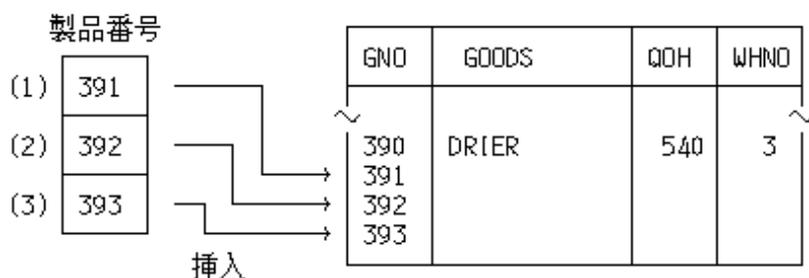
複数行指定ホスト変数を値指定に使用すると、複数行指定ホスト変数の繰り返し回数と同じ件数のデータを挿入することができます。

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

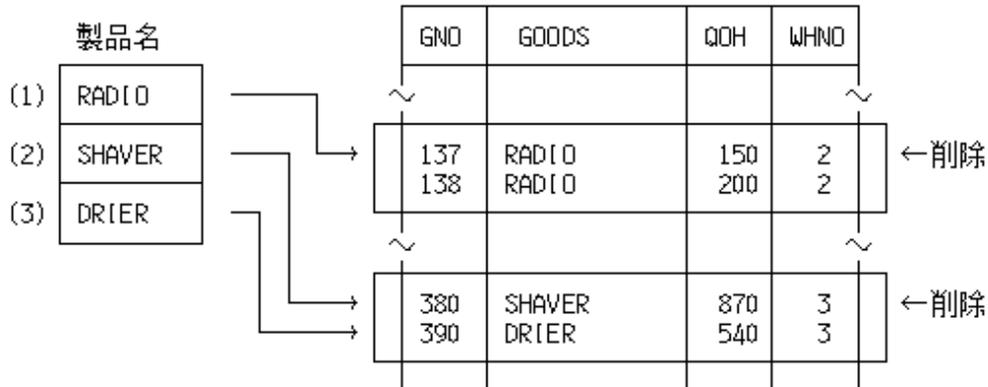
複数行指定ホスト変数を問い合わせ指定に使用すると、複数の条件に合致するデータを一度に削除できます。

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い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、設定句および問い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]“15.2.4.4 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.
:

```

### 15.2.4.1.2 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.             [1]
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
-----
↑
[2]

```

```
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

製品番号	製品名	在庫数量	倉庫番号
391	CASSETTE TAPE	100	1
392	SHAVER	200	2
393	DRIER	300	3

挿入

GNO	GOODS	QOH	WHNO
390	DRIER	540	3
391	CASSETTE TAPE	100	1
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。

## 15.2.4.2 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使用できる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、“[15.2.10 SQL文と指定可能なホスト変数](#)”を参照してください。

## 15.2.4.3 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プロシージャでの変更行は、SQLERRD(3)では保証されません。

### 15.2.4.4 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、“15.2.3.2.1 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.           [1]
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      ... [2]
END-EXEC.
:
EXEC SQL OPEN CUR1 END-EXEC.
:
EXEC SQL
    FOR :行数                                -----
    FETCH CUR1 INTO :在庫表                  ↑
END-EXEC.                                [3]
:                                          ↓
:                                          -----

```

- [1] 表指定ホスト変数“在庫数”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。

## 15.2.4.5 スクロール可能なカーソルを使用したデータの取得

スクロール可能なカーソルとは、カーソルのOPEN文で作成された結果を、順不同にアクセスすることができるカーソルのことを言います。NetCOBOLでは、以下の文が該当します。

FETCH文	動作説明
FETCH PRIOR文	最後にFETCHされた行の前のデータを取り出します。
FETCH FIRST文	カーソルを先頭行に位置付け、その行から後ろのデータを取り出します。
FETCH LAST文	カーソルを最終行に位置付け、その行から前のデータを取り出します。

ここでは、スクロール可能なカーソルを使用してデータを取得する方法について説明します。

### スクロール可能なカーソルを使用するため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、スクロール可能なカーソルを使用することができません。スクロール可能なカーソルを使用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、“[15.2.8.1.2 ODBC情報ファイルの作成](#)”を参照してください。

スクロール可能なカーソルを使用するため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 または STATIC または 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.
EXEC SQL FETCH NEXT CUR1 INTO :在庫表 END-EXEC.
EXEC SQL FETCH PRIOR CUR1 INTO :在庫表 END-EXEC.
EXEC SQL 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.
```

… [10]

- [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.             [1]  
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.                … [2]  
EXEC SQL FOR 3 FETCH NEXT CUR1 INTO :在庫表 END-EXEC. … [3]  
EXEC SQL FOR 3 FETCH NEXT CUR1 INTO :在庫表 END-EXEC. … [4]  
EXEC SQL FOR 3 FETCH NEXT CUR1 INTO :在庫表 END-EXEC. … [5]  
EXEC SQL FOR 3 FETCH PRIOR CUR1 INTO :在庫表 END-EXEC. … [6]  
EXEC SQL FOR 3 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1～3件目のデータを"在庫表"に取り出します。

STOCK-LIST

	PRODUCT- -NUMBER	PRODUCT- NAME	QUANTITY- IN-STOCK	WAREHOUSE -NUMBER
(1)	110	TELEVISION	85	2
(2)	111	TELEVISION	90	2
(3)	123	REFRIGERATOR	60	1

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
	:		

[4] 在庫表の4~6件目のデータを"在庫表"に取り出します。

STOCK-LIST

	PRODUCT- -NUMBER	PRODUCT- NAME	QUANTITY- IN-STOCK	WAREHOUSE -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
	:		

[5] 在庫表の7~9件目のデータを"在庫表"に取り出します。

STOCK-LIST

	PRODUCT- -NUMBER	PRODUCT-NAME	QUANTITY- IN-STOCK	WAREHOUSE -NUMBER
(1)	140	CASSET DECK	120	2
(2)	141	CASSET DECK	80	2
(3)	200	AIR CONDITIONER	4	1

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
	:		

[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				GNO	GOODS	QOH	WHNO	
(1)	124	REFRIGERATOR	75	1	124	REFRIGERATOR	75	1
(2)	137	RADIO	150	2	137	RADIO	150	2
(3)	138	RADIO	200	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				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。

## FETCH FIRST文によるデータ取得

"STOCK表"の先頭から2件のデータをそれぞれFETCH NEXT文で取り出し、FETCH FIRST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2件のデータを取り出し、次にFETCH FIRST文で1件のデータを取り出します。

### FETCH FIRST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.      ... [3]
EXEC SQL FETCH NEXT CUR1 INTO :在庫表 END-EXEC.      ... [4]
EXEC SQL FETCH FIRST CUR1 INTO :在庫表 END-EXEC.     ... [5]
EXEC SQL CLOSE CUR1 END-EXEC.                        ... [6]
EXEC SQL ROLLBACK WORK END-EXEC.                     ... [7]
EXEC SQL DISCONNECT DEFAULT END-EXEC.                ... [8]
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GNOが110のデータを"在庫表"に取り出します。

**STOCK-LIST**

PRODUCT- NUMBER	PRODUCT- NAME	QUANTITY- IN-STOCK	WAREHOUSE -NUMBER
110	TELEVISION	85	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
	:		

- [4] 在庫表のGNOが111のデータを"在庫表"に取り出します。

**STOCK-LIST**

PRODUCT- NUMBER	PRODUCT- NAME	QUANTITY- IN-STOCK	WAREHOUSE -NUMBER
111	TELEVISION	90	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
	:		

- [5] 在庫表のGNOが110のデータを"在庫表"に取り出します。

## STOCK-LIST

PRODUCT- -NUMBER	PRODUCT- -NAME	QUANTITY- IN-STOCK	WAREHOUSE -NUMBER
110	TELEVISION	85	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
	:		

- [6] カーソル"CUR1"をクローズします。
- [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。

## FETCH LAST文によるデータ取得

"STOCK表"の最終行をFETCH LAST文で取り出し、最終行から前の行をFETCH PRIOR文で取り出す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 LAST文で1件のデータを取り出し、次にFETCH PRIOR文で2件のデータを取り出します。

### FETCH LAST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 LAST CUR1 INTO :在庫表 END-EXEC.
EXEC SQL FETCH PRIOR CUR1 INTO :在庫表 END-EXEC.
EXEC SQL 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GNOが390のデータを"在庫表"に取り出します。

STOCK-LIST

PRODUCT- NUMBER	PRODUCT- NAME	QUANTITY- IN-STOCK	WAREHOUSE -NUMBER
390	DRIER	540	3

GNO	GOODS	QOH	WHNO
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

[4] 在庫表のGNOが380のデータを"在庫表"に取り出します。

STOCK-LIST

PRODUCT- NUMBER	PRODUCT- NAME	QUANTITY- IN-STOCK	WAREHOUSE -NUMBER
380	SHAVER	870	3

GNO	GOODS	QOH	WHNO
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

[5] 在庫表のGNOが351ののデータを"在庫表"に取り出します。

STOCK-LIST

PRODUCT- NUMBER	PRODUCT-NAME	QUANTITY- IN-STOCK	WAREHOUSE -NUMBER
351	CASSETTE TAPE	2500	2

GNO	GOODS	QOH	WHNO
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

[6] カーソル"CURI"をクローズします。

[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、サーバとの接続を切断します。

**複数行指定ホスト変数を使用したFETCH LAST文によるデータ取得**

"STOCK表"の最終行から3件のデータをFETCH LAST文で取り出し、そこから3件ずつ前の行のデータをFETCH PRIOR文で取り出す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、FETCH LAST文で3件のデータ取り出しを行い、次に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 LAST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. [1]
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. ... [2]
EXEC SQL FOR 3 FETCH LAST CUR1 INTO :在庫表 END-EXEC. ... [3]
EXEC SQL FOR 3 FETCH PRIOR CUR1 INTO :在庫表 END-EXEC. ... [4]
EXEC SQL FOR 3 FETCH PRIOR CUR1 INTO :在庫表 END-EXEC. ... [5]
EXEC SQL CLOSE CUR1 END-EXEC. ... [6]
EXEC SQL ROLLBACK WORK END-EXEC. ... [7]
EXEC SQL DISCONNECT DEFAULT END-EXEC. ... [8]
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GNOが351、380、390の3件のデータを"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- -NUMBER	PRODUCT- NAME	QUANTITY- IN-STOCK	WAREHOUSE -NUMBER
(1)	351	CASSETTE TAPE	2500	2
(2)	380	SHAVER	870	3
(3)	390	DRIER	540	3

GNO	GOODS	QOH	WHNO
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

[4] 在庫表のGNOが227、240、243の3件のデータを"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- -NUMBER	PRODUCT-NAME	QUANTITY- IN-STOCK	WAREHOUSE -NUMBER
(1)	227	REFRIGERATOR	15	1
(2)	240	CASSETTE DECK	25	2
(3)	243	CASSETTE DECK	14	2

GNO	GOODS	QOH	WHNO
	:		
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

[5] 在庫表のGNOが212、215、226の3件のデータを"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- -NUMBER	PRODUCT-NAME	QUANTITY- IN-STOCK	WAREHOUSE -NUMBER
(1)	212	TELEVISION	0	2
(2)	215	VIDEO	5	2
(3)	226	REFRIGERATOR	8	1

GNO	GOODS	QOH	WHNO
	:		
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

[6] カーソル"CUR1"をクローズします。

[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。

## 15.2.5 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。

### 15.2.5.1 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、パフォーマンスを發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。

### 15.2.5.2 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。

## 15.2.6 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17章 オブジェクト指向プログラミング機能](#)”

### 15.2.6.1 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、“[15.2.3.1 サンプルデータベース](#)”のSTOCK表(在庫表)を使用しています。

## 15.2.6.2 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ます。

### 15.2.6.3 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 [2]
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.

```

[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、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。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" [1]
  RETURNING DBACCESS-OBJECT(OBJECT-COUNT)
  INVOKE DBACCESS-OBJECT(OBJECT-COUNT) "ZAIKO-KANRI"
  ↓
END-PERFORM.
STOP RUN.

```

[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]“[15.2.8.1.2 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]“[15.2.8.2 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。

## 15.2.7 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小を区別する場合は、翻訳オプションNOALPHALを指定して翻訳します。[参照]“[A.3.2 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]“[15.2.9 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。

## 15.2.8 プログラムの実行

---

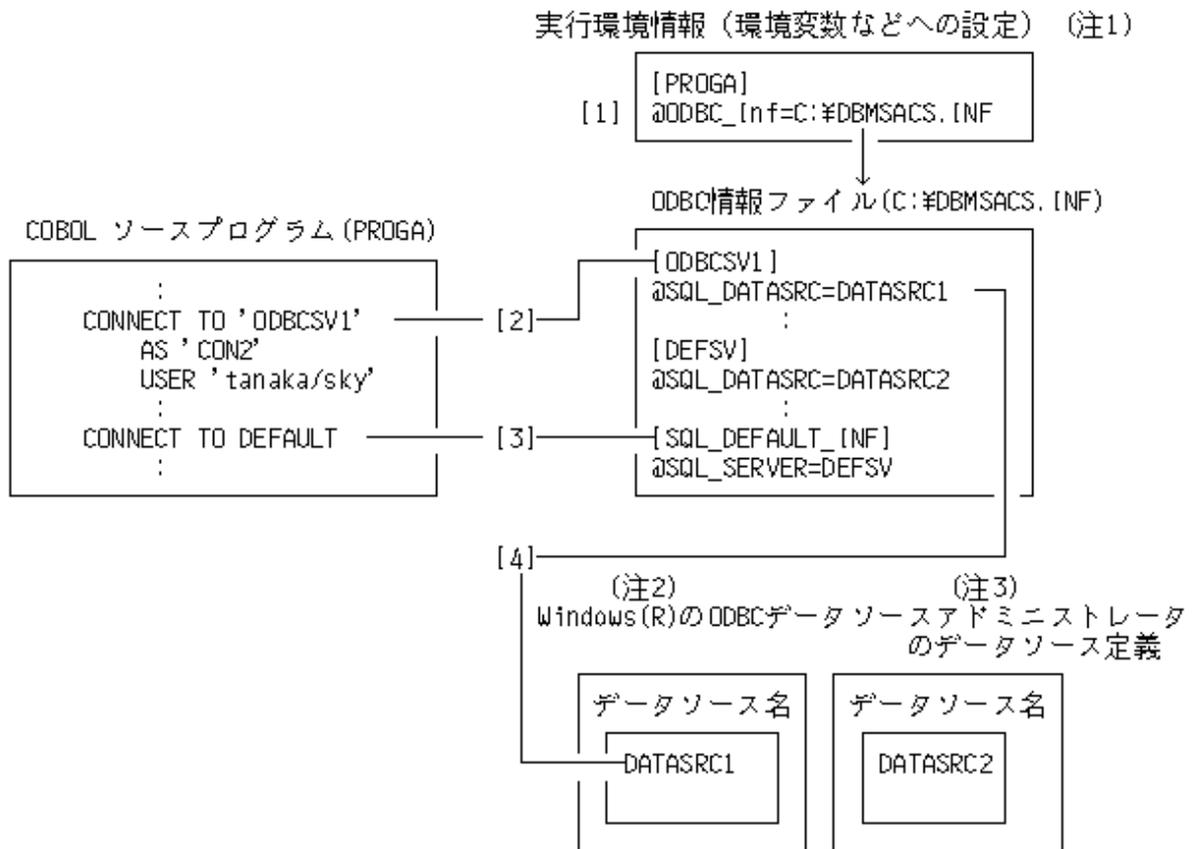
ここでは、プログラムを実行するために必要な実行環境の構築およびODBC情報設定ツールの使い方について説明します。

### 15.2.8.1 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15.4 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15.4 各情報ファイルの関連



注1: 詳細は、“5.3 実行環境情報の設定”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。

### 15.2.8.1.1 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、環境変数情報@ODBC\_Infを設定してください。[参照]“C.2.68 @ODBC\_Inf(ODBC情報ファイルの指定)”、“5.3.2 実行環境情報の設定方法”

```
@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]“15.2.8.1.2 ODBC情報ファイルの作成”

### 15.2.8.1.2 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]“15.2.8.2 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。

## 注意

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15.1 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 FILE_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。 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。 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。(注1)
@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。 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 READ_WRITE	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。 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。 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。
@SQL_COMMIT_MODE	COMMITモード	MANUAL AUTO	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。 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。 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。 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。

情報名	設定内容		備考
@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。 @SQL_CONCURRENCY が指定されない場合、デフォルトはREAD_ONLYです。(注2)
LOCK			
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。 @SQL_CURSOR_TYPEが指定されない場合、デフォルトはFORWARD_ONLYです。(注3) 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。 FORWARD_ONLY指定ではFETCH PRIOR文、FETCH FIRST文、およびFETCH LAST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。 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。 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。 @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。 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。
@SQL_ODBC_MARS	MARS機能の指定	OFF	SQL ServerのMARS機能(注5)を使用するかどうか指定します。MARS機能は、SQL Server Native Client 10.0/11.0 ODBCドライバを使用する場合にだけ使用することができます。OFFを指定した場合、MARS機能は有効になりません。ONを指定した場合にMARS機能が有効になります。デフォルトは、OFFです。
ON			

・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15.2 @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、“15.2.13 ODBCドライバ使用時の注意事項”を参照してください。

・注3

指定値に対する動作は、“表15.3 @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、“15.2.13 ODBCドライバ使用時の注意事項”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的な列を必ず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。

・ 注5

- － MARS(Multiple Active Result Set)機能についての詳細は、SQL Serverのマニュアルを参照してください。

表15.2 @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。



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15.3 @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15.4 デフォルトコネクション情報の定義内容”に示します。

表15.4 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。
@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15.5 コネクション有効範囲の定義内容”に示します。

表15.5 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	<table border="1"> <thead> <tr> <th>設定内容</th> <th>備考</th> </tr> </thead> <tbody> <tr> <td>PROCESS</td> <td>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。</td> </tr> <tr> <td>THREAD</td> <td>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)</td> </tr> <tr> <td>OBJECT_INSTANCE</td> <td></td> </tr> </tbody> </table>	設定内容	備考	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。	THREAD	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)	OBJECT_INSTANCE	
設定内容	備考									
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。									
THREAD	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)									
OBJECT_INSTANCE										

注: 指定値に対する動作は、“表15.6 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、“20.6.2 リモートデータベースアクセス(ODBC)の利用”および“15.2.6.3 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15.6 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> </ul>

指定値	動作説明
	<ul style="list-style-type: none"> <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	<p>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。</p> <p>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)</p>

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。

## 15.2.8.2 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となる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

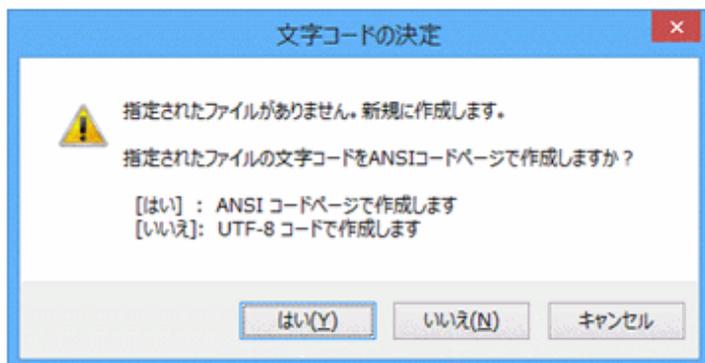
ODBC情報設定ツール(SQLODBCS.EXE)を起動します。

### 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開きます。BOM付きUTF-8のファイルを開いた時は、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、“15.2.8.1.2 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\_INF]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

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
ユーザID	@SQL_USERID
パスワード	@SQL_PASSWORD

詳細は、“15.2.8.1.2 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、“15.2.8.1.2 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。

### 15.2.8.3 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	-



#### 注意

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。

### 15.2.8.4 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。



#### 注意

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

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]“18.1.3 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。

## 15.2.9 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
	CASCADED

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

<b>【G】</b>	GET
	GLOBAL
	GO
	GOTO
	GRANT
	GROUP

<b>【H】</b>	HAVING
	HOUR

<b>【I】</b>	IDENTITY
	IGNORE
	IMMEDIATE
	IN
	INCLUDE
	INDEX
	INDICATOR
	INITIALLY
	INNER
	INPUT
	INSENSITIVE
	INSERT
	INTEGER
	INTERSECT
	INTERVAL
	INTO
	IS
ISOLATION	

<b>【J】</b>	JOIN
------------	------

<b>【K】</b>	KEY
------------	-----

<b>【L】</b>	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
-----	------

## 15.2.10 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15.7 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

## 15.2.11 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15.8 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 または PIC S9(4) COMP-5
	SQL_INTEGER (INTEGER)	PIC S9(9) BINARY または PIC S9(9) COMP-5
10進	SQL_DECIMAL (DECIMAL)	PIC S9(p) PACKED-DECIMAL または PIC S9(p)V9(q) PACKED-DECIMAL  $1 \leq p \leq 15$ $1 \leq q$ $p+q \leq 18$
	SQL_NUMERIC (NUMERIC)	$\left. \begin{array}{l} \text{PIC S9(p)} \\ \text{PIC S9(p)V9(q)} \end{array} \right\}$  [SIGN IS $\left. \begin{array}{l} \text{LEADING SEPARATE CHARACTER} \\ \text{TRAILING} \end{array} \right\}$ ]  $1 \leq p \leq 15$ $1 \leq q$ $p+q \leq 18$
内部浮動小数点	SQL_REAL (REAL)	COMP-1
	SQL_DOUBLE (FLOAT)	COMP-2

翻訳オプションENCODE(SJIS,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15.9 文字データとの対応表(翻訳オプションENCODE(SJIS,SJIS)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. 49 データ名-2 PIC S9(m) BINARY. (注2) 49 データ名-3 PIC X(n). (注1)  または 01 データ名-1. 49 データ名-2 PIC S9(m) COMP-5. (注2) 49 データ名-3 PIC X(n). (注1)

ODBC SQLデータ型		COBOL での表現
		m=4 または 9
		01 データ名-1. (注1) 49 データ名-2 PIC S9(m) BINARY. (注2) 49 データ名-3 PIC N(n). (注1)
		または
		01 データ名-1. (注1) 49 データ名-2 PIC S9(m) COMP-5. (注2) 49 データ名-3 PIC N(n). (注1)
		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ENCODE(UTF8,UTF16)またはENCODE(UTF8,UTF32)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15.10 文字データとの対応表(翻訳オプションENCODE(UTF8,UTF16)またはENCODE(UTF8,UTF32)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. 49 データ名-2 PIC S9(m) BINARY. (注2) 49 データ名-3 PIC X(n). (注1)
	SQL_WVARCHAR (WVARCHAR)	01 データ名-1. (注3) 49 データ名-2 PIC S9(m) BINARY. (注2) 49 データ名-3 PIC N(n). (注1)
		または
		01 データ名-1. 49 データ名-2 PIC S9(m) COMP-5. (注2) 49 データ名-3 PIC X(n). (注1)
		m=4 または 9
		01 データ名-1. (注3) 49 データ名-2 PIC S9(m) COMP-5. (注2) 49 データ名-3 PIC N(n). (注1)
		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。

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15.11 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。

## 15.2.12 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。 (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。 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過しました。	COBOLシステムで許されるコネクションの最大数を超過しています。コネクション数を減らす対処を行ってください。 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。

SQLSTATE	SQLCODE	SQLMSG	プログラムの処理
9999B	-999999800	指定した接続は存在しません。	接続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。 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。 提供元担当者に連絡してください。

## 15.2.13 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。

### 15.2.13.1 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]“15.2.11 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文で接続を切断する前には、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。
  - SELECT文(PREPARE/EXECUTE文の場合だけ)
  - UPDATE文(探索)
  - UPDATE文(位置付け)
  - DELETE文(探索)
  - DELETE文(位置付け)
  - INSERT文
  - 動的UPDATE文(探索)
  - 動的UPDATE文(位置付け)
  - 動的DELETE文(探索)
  - 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。

### 15.2.13.2 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、FETCH FIRST文、またはFETCH LAST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。

<p>現象 :</p> <p>SQLSTATE : S1106</p> <p>SQLCODE : +0</p> <p>SQLMSG : フェッチの型が範囲を超えています。</p>
---

### 15.2.13.3 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。

#### Symfoware(Postgres) Server ODBCドライバ使用時の留意事項

- カーソルの位置づけ更新はサポートされていません。
- 更新処理時のSQLERRD(3)による処理行数はサポートされていません。常に1となります。
- FETCH PRIORでカーソルの先頭に移動後に、FETCH NEXTを実行する動作は保証されません。

#### Oracle 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 SQL Server Native Client / Microsoft SQL Native Client 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。

<p>現象</p> <p>SQLSTATE : 01000</p> <p>SQLCODE : +5701</p> <p>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。</p>
--

- 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]“[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]“[15.2.8.1.2 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。
  - 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。
  - 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 .. WHERE CURRENT OF カーソル名	(8) UPDATE 表名 .. 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。
  - @SQL\_CONCURRENCY = READ\_ONLY
  - @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、または、MARS(Multiple Active Result Set)機能を有効にして実行してください。MARS機能は @SQL\_ODBC\_MARS=ON を指定することにより有効になります。(デフォルトはOFF)

なお、MARS機能は SQL Native Client ODBCドライバでのみ使用できます。MARS(Multiple Active Result Set)機能についての詳細は、SQL Serverのマニュアルを参照してください。

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]“[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]“[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。

### 15.2.13.4 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。

## 15.2.14 デッドロック出口

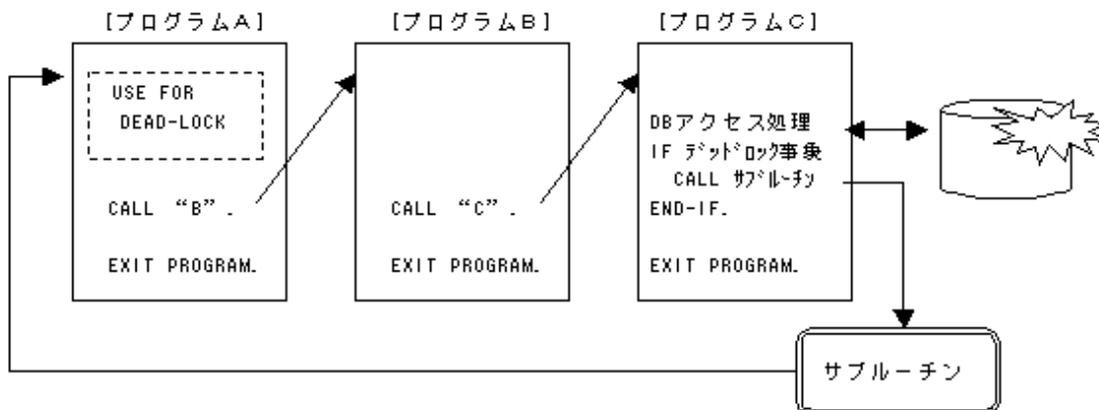
プログラムがデータベースをアクセスする時にアクセス競合が起こり、プログラム同士で占有解除を待つような状態が発生することをデッドロック状態といいます。デッドロック状態が発生すると、デッドロック事象がデータベースからプログラムに通知されます。

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。

### 15.2.14.1 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を呼び出します。

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。

サブルーチンを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、サブルーチンを呼び出したプログラムまたはその上位のプログラムで、サブルーチンを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、サブルーチンを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、“[I.2.2 COB\\_DEADLOCK\\_EXIT\(デッドロック出口スケジュールサブルーチン\)](#)”を参照してください。

### 15.2.14.2 注意事項

- 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。
- 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16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簡易アプリ間通信機能について説明します。

### 16.1 通信の種類

簡易アプリ間通信機能の特徴および用途を“表16.1 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していることを表しています。

表16.1 簡易アプリ間通信機能の特徴・用途

特徴・用途		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
	通信形態	[クライアント] ・ Windows(32) ・ HP-UX ・ Solaris(32) ・ Linux [サーバ] (注1) ・ Windows(32) ・ HP-UX ・ Solaris(32) ・ Linux
	プログラムの記述内容	・ 通信レコードの定義 ・ 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 — COBCI_OPEN — COBCI_READ — COBCI_WRITE — COBCI_CLOSE
	必要な関連製品	なし
用途	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も起動できます。

### 16.2 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。

#### 16.2.1 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。

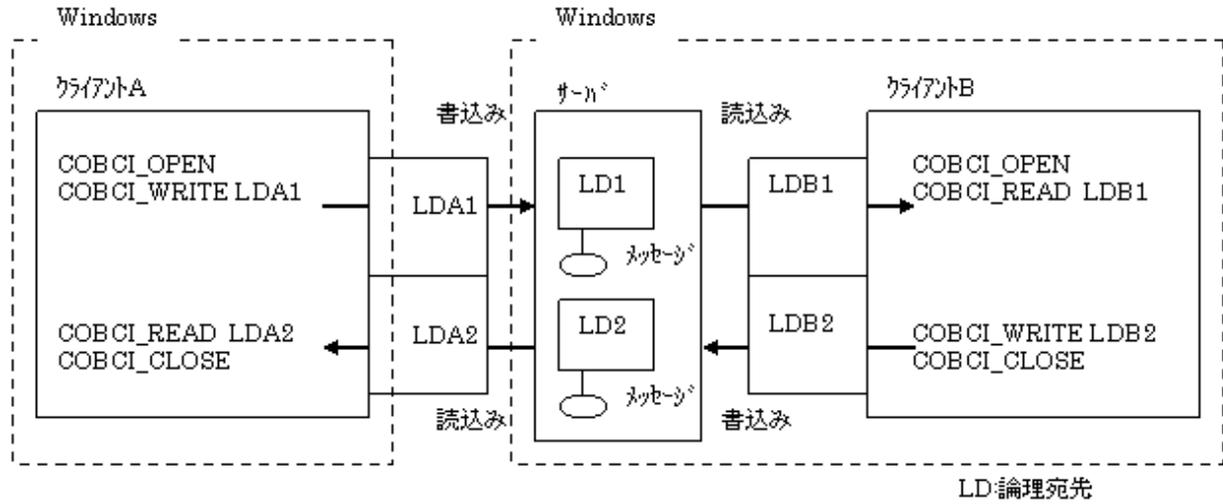
プログラム間のメッセージの受渡しは、メッセージを制御する通信システムプロセス(以降サーバと呼びます)の論理宛先を介して行われます。

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16.1 簡易アプリ間通信の概要図](#)”に示します。

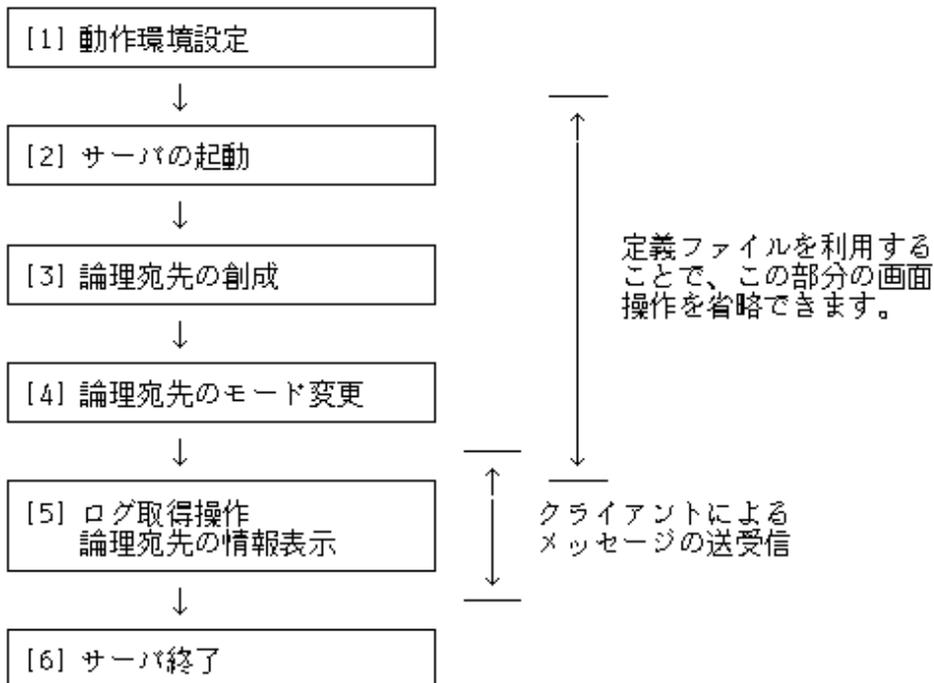
図16.1 簡易アプリ間通信の概要図



## 16.2.2 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ければ自由に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、“[16.2.3.1 \[通信システム環境設定\]ダイアログ](#)”および“[16.2.3.2 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。

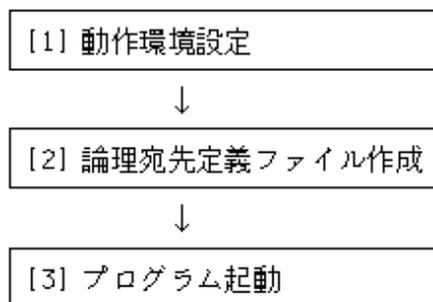
## [5] ログ取得操作/論理宛先の情報表示

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、“[16.2.3.2 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、“[16.2.4.1 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。

## 16.2.3 サーバ操作ウィンドウ

ここでは、サーバで操作するウィンドウについて、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ィンドウ

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、“[16.2.3.2 簡易アプリ間通信サーバウィンドウ](#)”の“ログ取得”を参照してください。

### 16.2.3.1 [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。

#### (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ものとみなします。

指定する値については、“[16.2.5 メモリ資源の見積り](#)”を参考にしてください。

#### (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ものとみなします。

指定する値については、“[16.2.5 メモリ資源の見積り](#)”を参考にしてください。

### 16.2.3.2 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。

### 16.2.3.2.1 論理宛先創成

[論理宛先]メニューから“創成”を選択します。[論理宛先の創成]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。

[創成]ボタンをクリックすると、続けて別の論理宛先を創成できます。

#### [論理宛先の創成]ダイアログ

##### 論理宛先

創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。

##### 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”の値が指定されたものとみなします。

##### 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が指定されたものとみなします。

### 16.2.3.2.2 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。

#### [論理宛先の削除]ダイアログ

##### 論理宛先

創成されている論理宛先名を表示します。

##### 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。

##### 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。

### 16.2.3.2.3 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]ダイアログ

##### 論理宛先

創成されている論理宛先名を表示します。

##### 読み

論理宛先の読み込み属性を表示します。○は読み込み可、×は読み込み不可を示します。

## 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。

## 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。

## 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。

### 16.2.3.2.4 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]ダイアログ

##### 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。

##### 論理宛先

創成されている論理宛先名を表示します。

##### 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。

##### 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。

##### 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。

##### 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。

##### 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。

##### 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。

### 16.2.3.2.5 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。
4. [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 ・ サーバの論理宛先名 ・ バッファ長 ・ 処理種別 ・ 監視時間

事象	付加情報
WRITE 要求	COBCL_WRITE の以下のパラメタ <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) 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) 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) 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) 詳細コード (detail: 10進表示) (注1)
通信エラー	Windows Sockets関数名 エラーコード (err:10進表示) (注1) エラー内容
API エラー	Windows関数名 エラーコード (err:10進表示) (注2) エラー内容
内部矛盾	エラー内容

注1: 詳細は、“[1.7.2 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。 詳細は、“ <a href="#">1.7.2 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。

#### 16.2.3.2.6 サーバ終了

[論理宛先]メニューから“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。

### 16.2.3.3 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”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イ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“99999999”です。

##### StartMode (サーバ起動時の画面表示の指定)

---

```
StartMode = { WINDOW }  
                  {  
                  ICON  
                  }
```

---

サーバ起動時の画面上での表示状態を指定します。

- WINDOW:ウイ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 =  $\left\{ \begin{array}{c} \underline{\text{SYSDEF}} \\ \text{CLOSEONLY} \end{array} \right\}$

---

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イ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 =  $\left\{ \begin{array}{c} \underline{\text{MANUAL}} \\ \text{AUTO} \end{array} \right\}$

---

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 =  $\left\{ \begin{array}{c} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$

---

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 =  $\left\{ \begin{array}{c} \underline{\text{ERROR}} \\ \text{TRACE} \end{array} \right\}$

---

[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

```
[SRVINP]
: 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=, ,
```

## 16.2.4 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。

### 16.2.4.1 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。[クライアント定義]ダイアログが表示されます。

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

##### 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。

##### 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。

##### 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。

##### 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16.2 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	以下の環境に対応しています。 <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	以下のIPv6 アドレスが使用できます。 <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> </ul>

項目	仕様
	<ul style="list-style-type: none"> <li>リンクローカルアドレス(LLA)</li> </ul> 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。 <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	以下のIPv6アドレス表記をサポートしています。 <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> <li>英字部の小文字表記</li> <li>英字部の大小文字混在表記</li> </ol> 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(プライベート拡張)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。

## 16.2.5 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。

$$MM \leq M$$

$$MW \leq WR + WW$$

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、“16.2.3.1 [通信システム環境設定]ダイアログ”および“16.2.3.3 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。

## 16.2.6 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込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イ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イ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

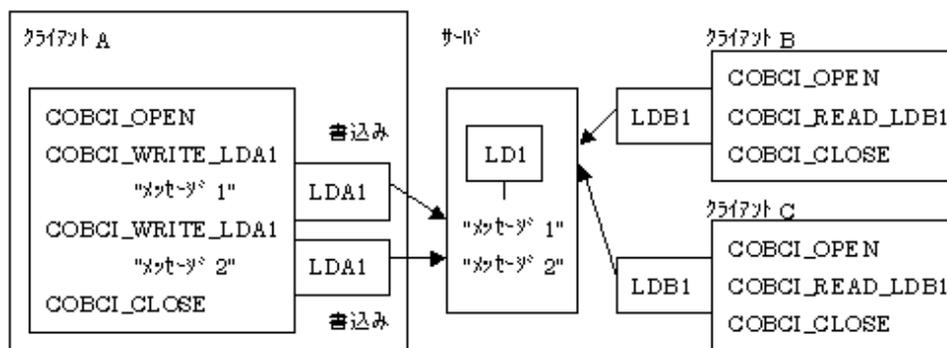
簡易アプリ間通信機能は、イントラネット環境下で使用してください。

### 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

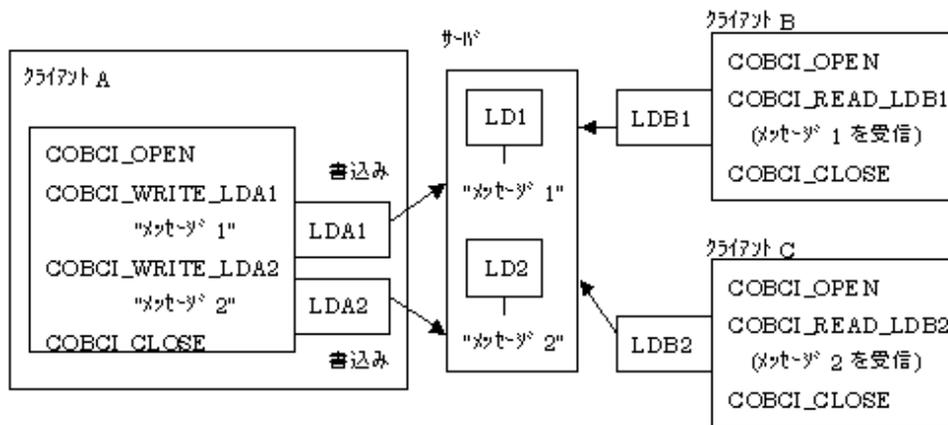
サーバに創成したひとつの論理宛先を複数のクライアントで共有し、複数のクライアントからメッセージの書き込み・読み込みを行った場合、以下の理由から、意図したクライアントにメッセージが伝わらない場合があります。

- 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」を受信する時があります。



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。



## 16.3 簡易アプリ間通信のサブルーチン

簡易アプリ間通信で使用するサブルーチンについては、“[1.7 簡易アプリ間通信のサブルーチンの使い方](#)”を参照してください。

# 第17章 オブジェクト指向プログラミング機能

## 17.1 基本的な使い方

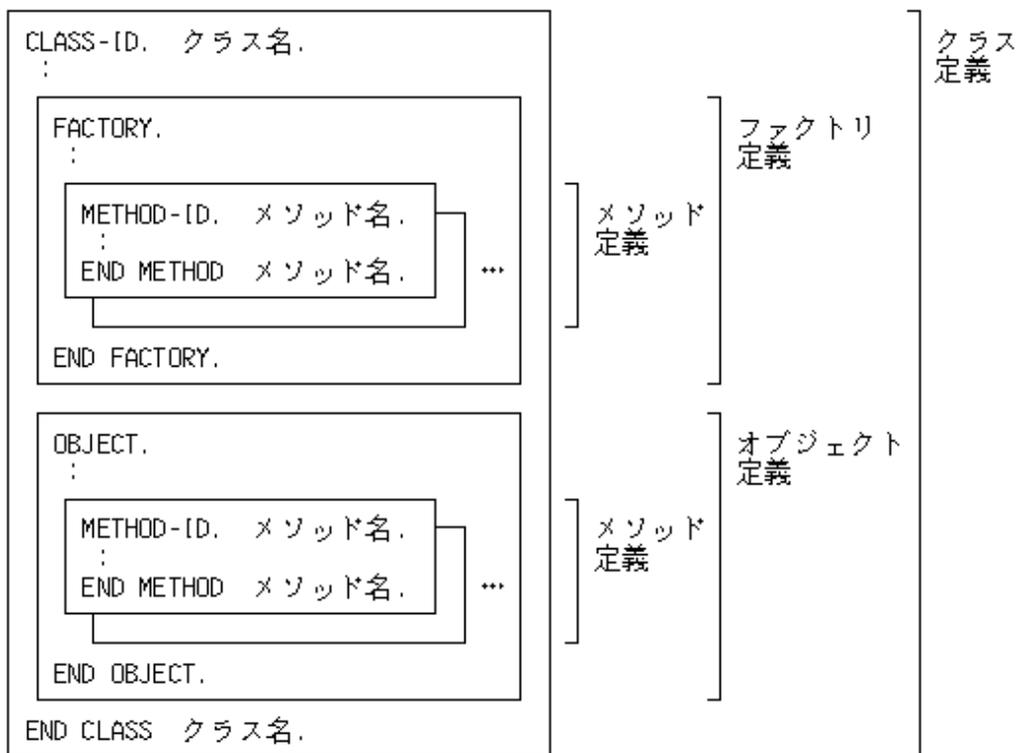
ここでは、オブジェクト指向プログラミングについて、具体的にコーディングレベルで説明します。

### 17.1.1 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。



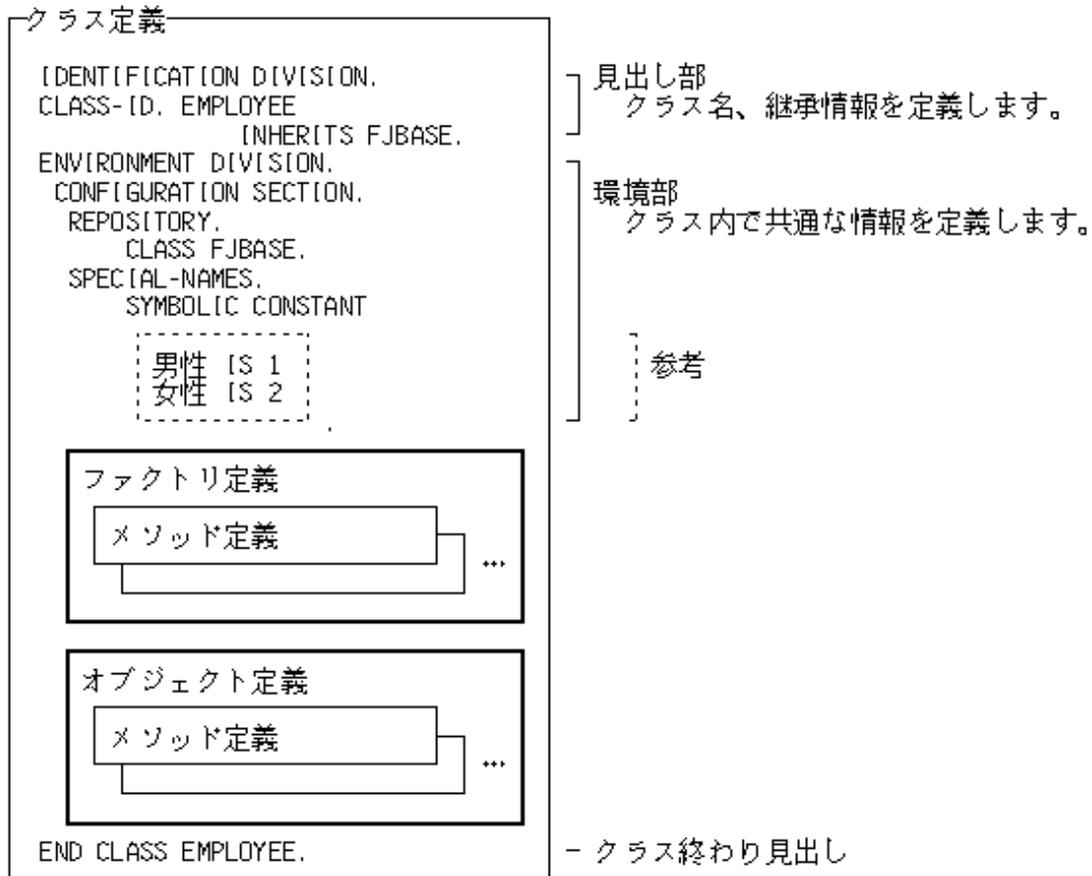
それぞれの定義について、具体的な定義方法や役割について説明します。

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。

#### 17.1.1.1 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)。

### 17.1.1.2 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。

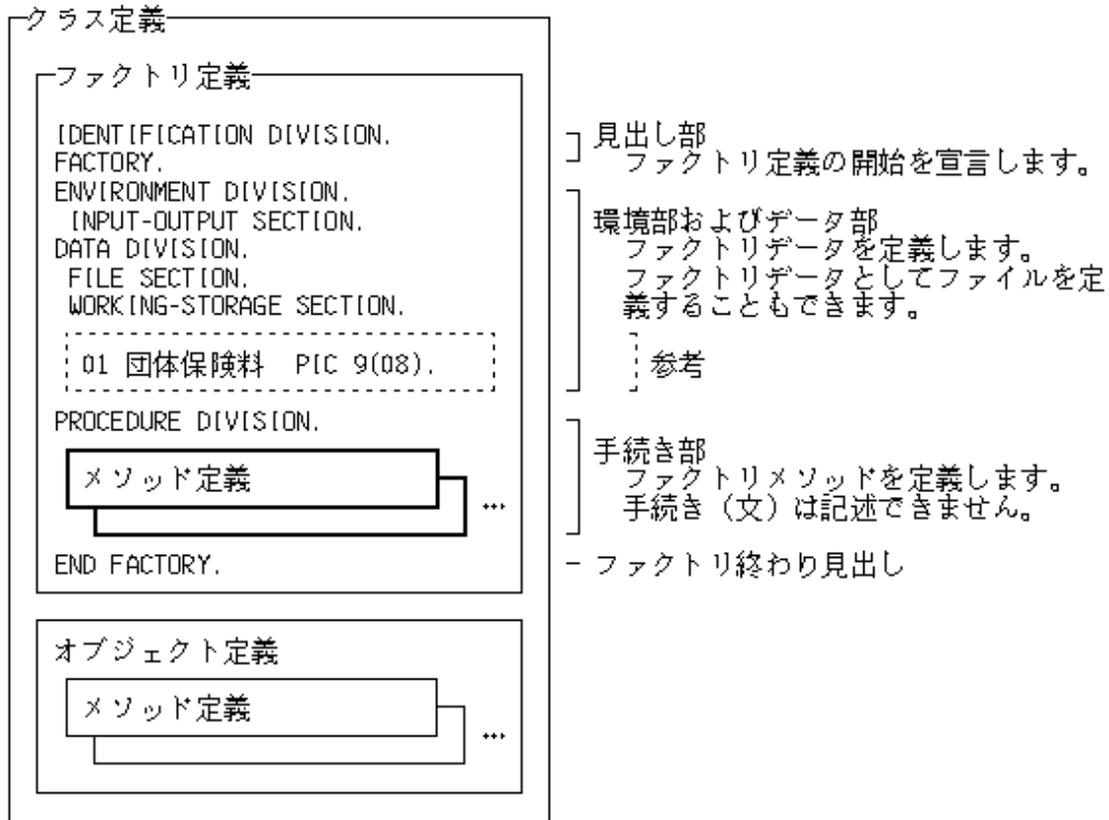
ファクトリ定義は、ファクトリ定義を表すための見出し部、ファクトリデータを定義するための環境部およびデータ部、ファクトリメソッドを定義するための手続き部から構成されます。

#### 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)。

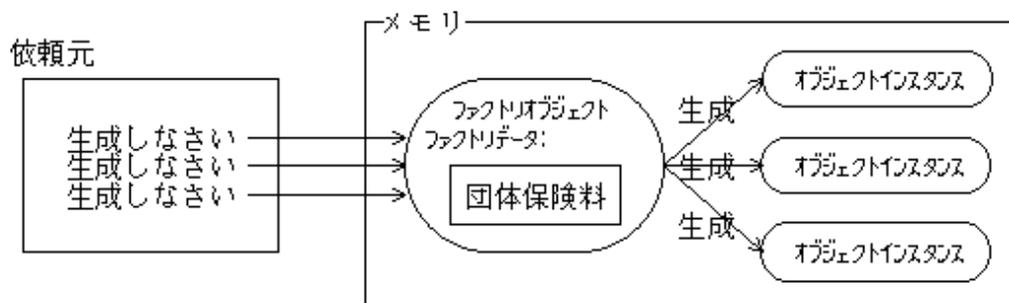
では、具体的にファクトリ定義の役割について説明します。

## 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、その名のとおりに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“17.1.3.2 FJBASEクラス”を参照してください。

### 17.1.1.3 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。

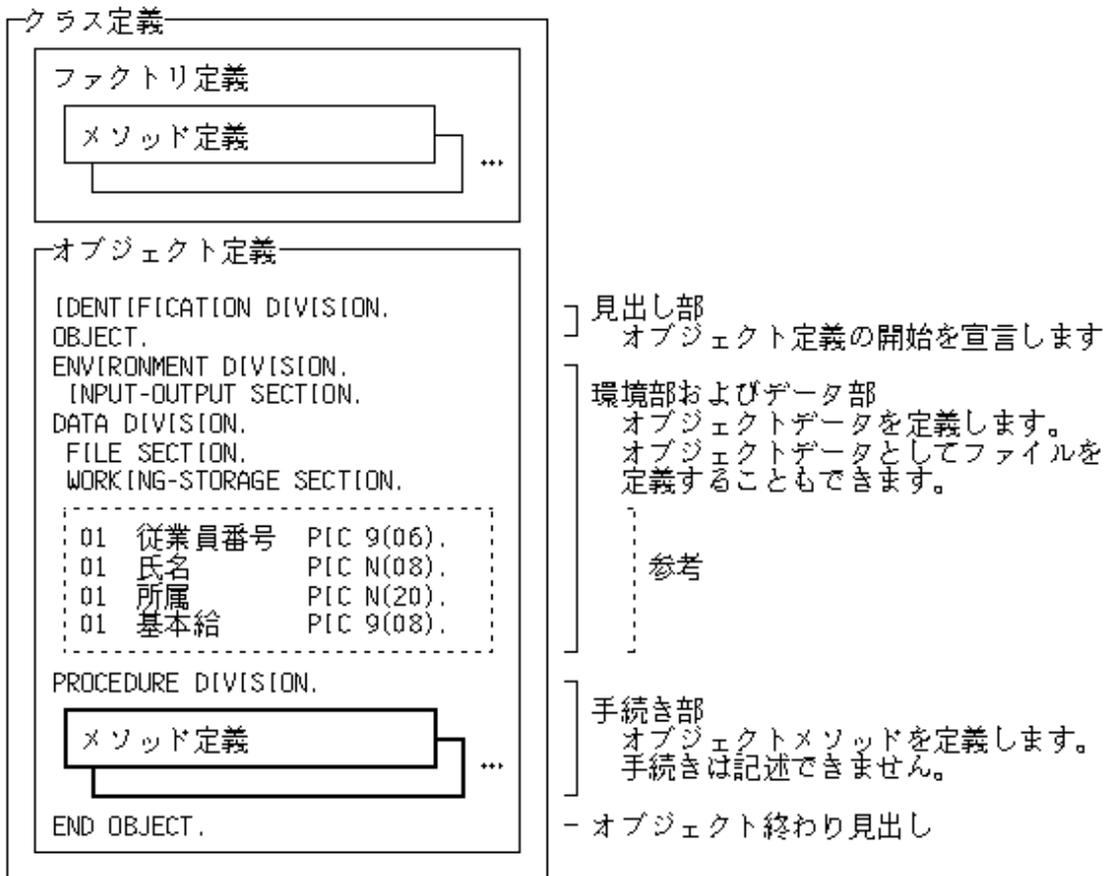
オブジェクト定義の構成はファクトリ定義と同じで、オブジェクト定義を表すための見出し部、オブジェクトデータを定義するための環境部およびデータ部、オブジェクトを定義するための手続き部からなります。

#### 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。

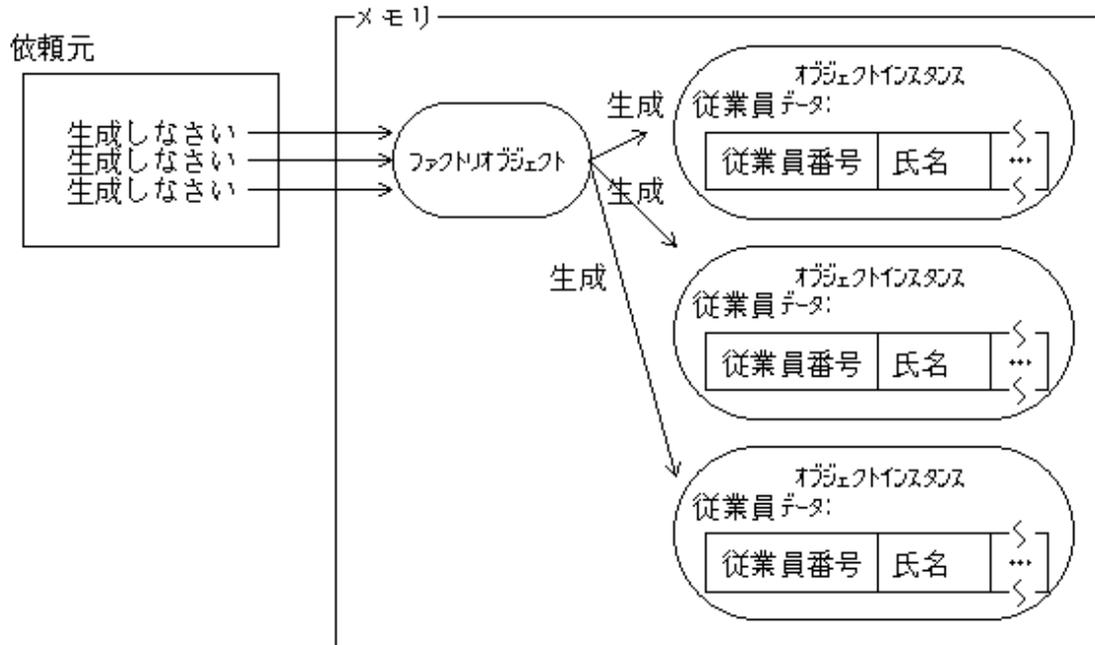


オブジェクト定義の環境部およびデータ部で定義されたデータは、オブジェクトメソッドでだけアクセスすることができます(上図の太線で囲まれている部分)。

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。

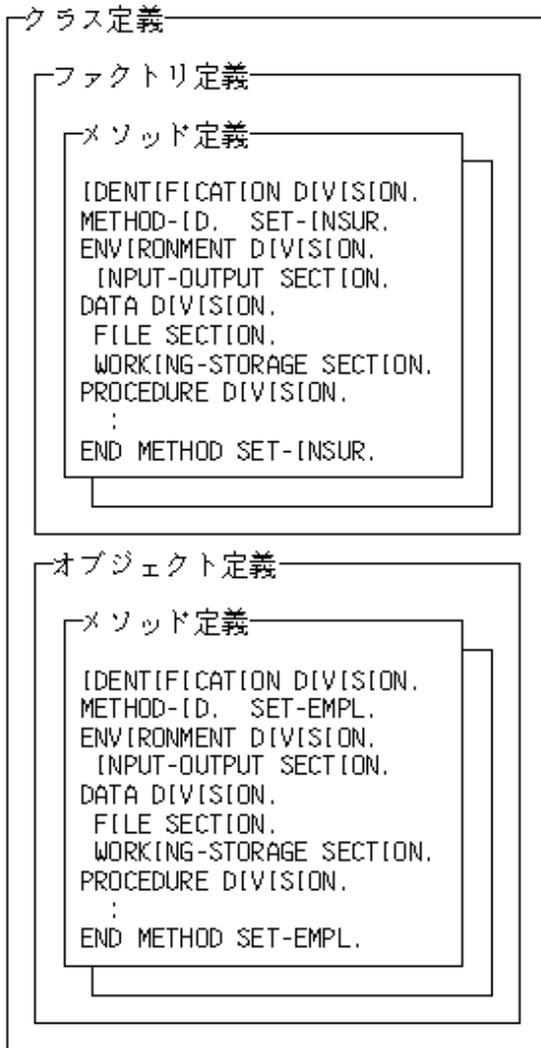


#### 17.1.1.4 メソッド定義

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。



←ファクトリメソッド

- 見出し部  
メソッド名を記述します。
- 環境部およびデータ部  
メソッドデータを定義します。  
メソッドデータとしてファイルを定義することもできます。
- 手続き部  
手続きを記述します。  
- メソッド終わり見出し

←オブジェクトメソッド

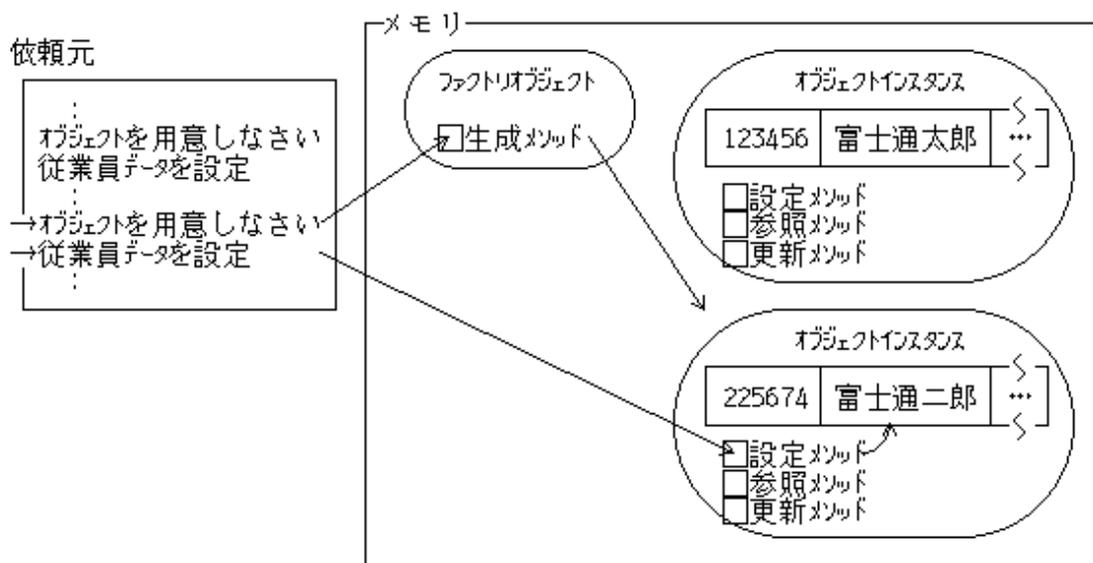
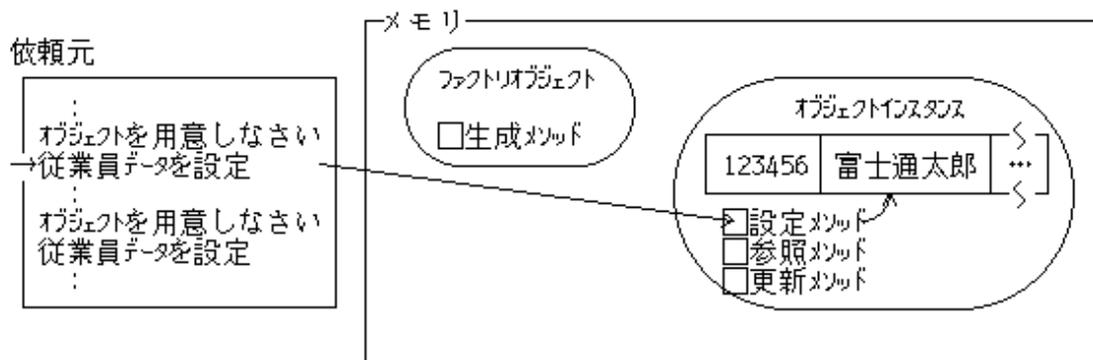
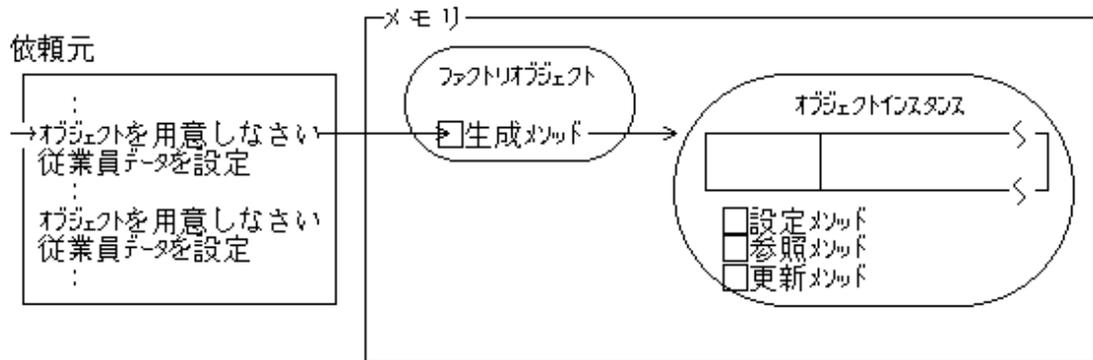
- 見出し部  
メソッド名を記述します。
- 環境部およびデータ部  
メソッドデータを定義します。  
メソッドデータとしてファイルを定義することもできます。
- 手続き部  
手続きを記述します。  
- メソッド終わり見出し

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。



## 17.1.2 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。

### 17.1.2.1 メソッドの呼出し

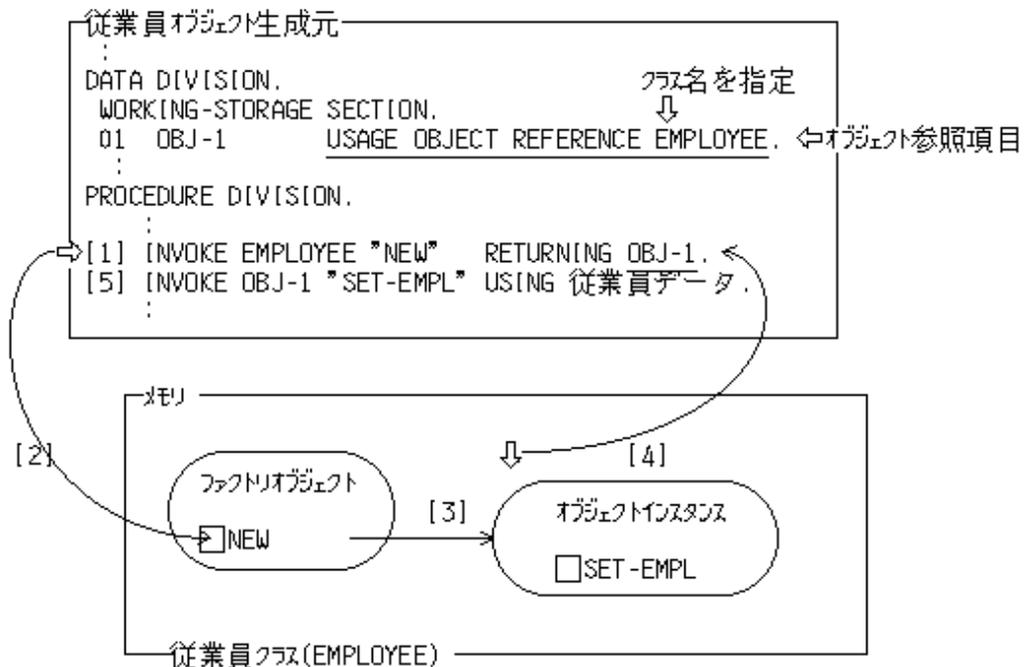
オブジェクトインスタンスを操作するためには、オブジェクトメソッドを呼び出す必要があります。このとき、「どのオブジェクトインスタンス」の「どのメソッド」を呼び出すかを指定します。「どのオブジェクトインスタンス」を表現するのに、オブジェクト参照項目と呼ばれるデータ項目を利用します。

### 17.1.2.1.1 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17.1 従業員オブジェクトインスタンスの生成

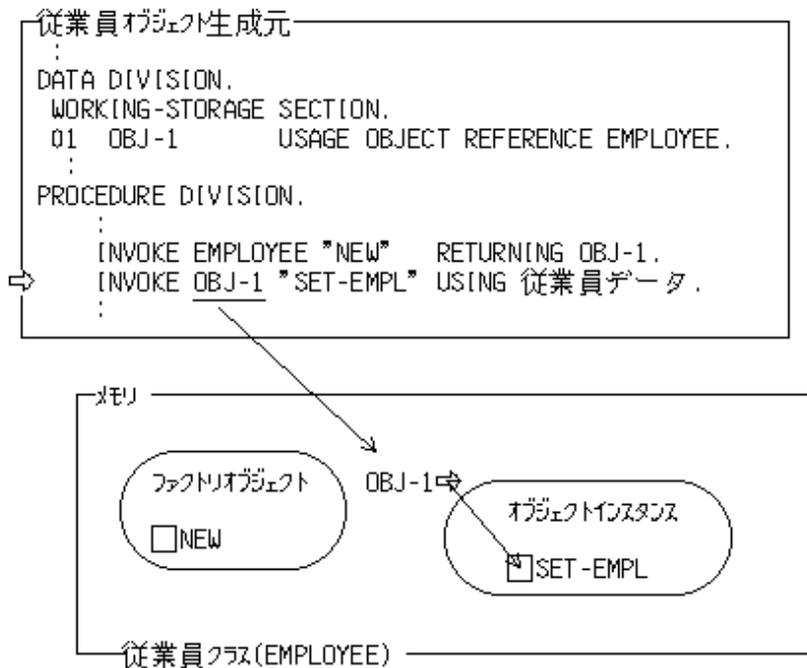


[1][2] : NEW メソッドを呼び出す。

[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17.2 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。

17.1.2.1.2 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17.1 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。

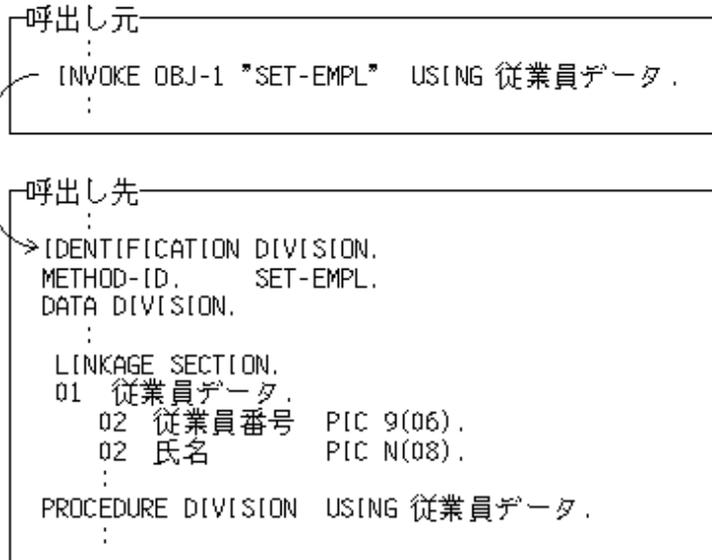
### 17.1.2.1.3 パラメタの指定

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。

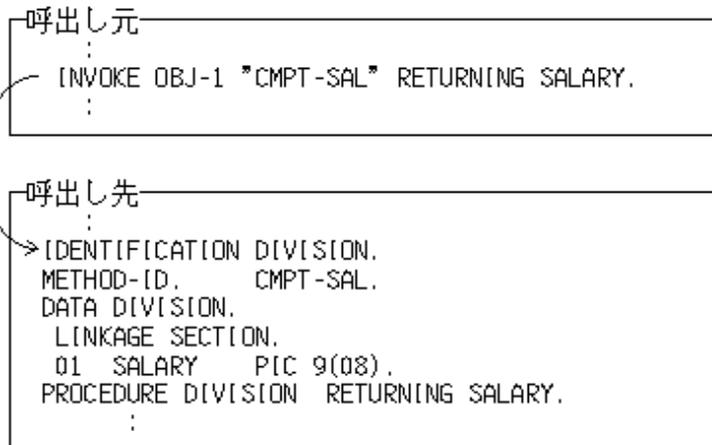
#### USING指定

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。



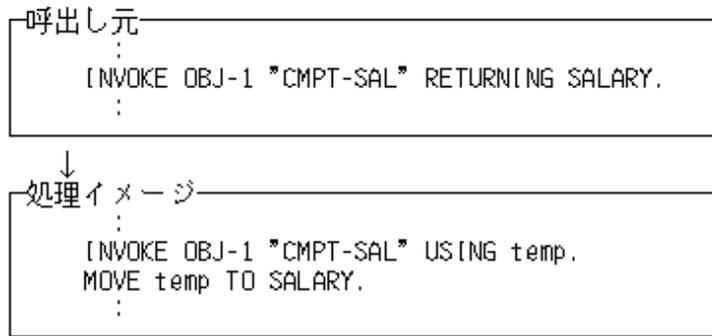
## RETURNING指定

RETURNING指定は、呼び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、呼び出し元で設定された値を呼び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。

呼び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。

### 17.1.2.2 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。

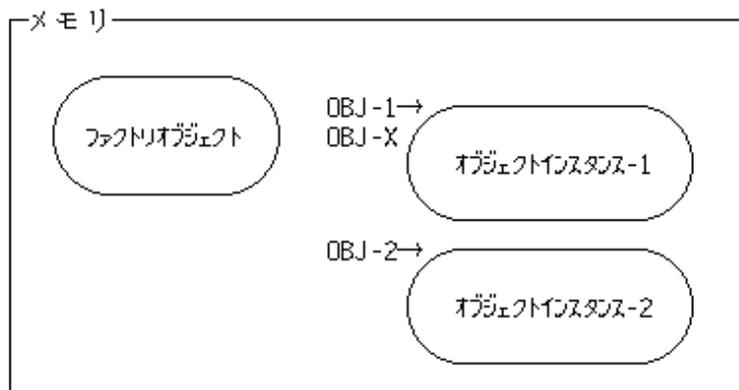
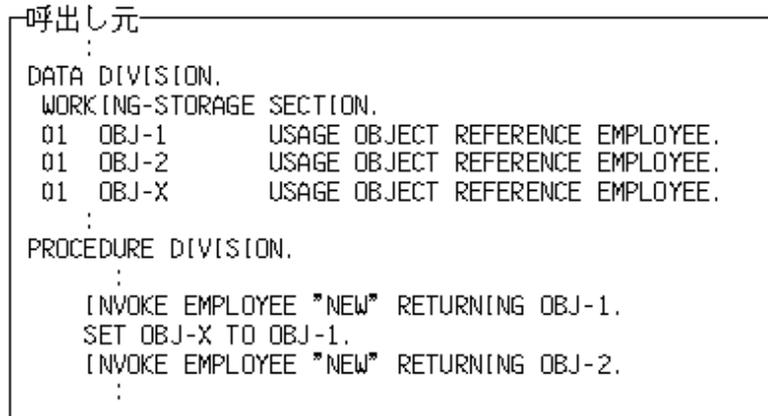
#### ファクトリオブジェクトの寿命

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。

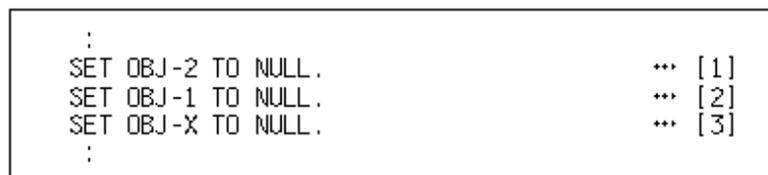
#### オブジェクトインスタンスの寿命

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。このような終了方をしない場合、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]“[20.10.2 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、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。

### 17.1.3 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。
- ・ システムの変更に対し、柔軟に対応できる。

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。

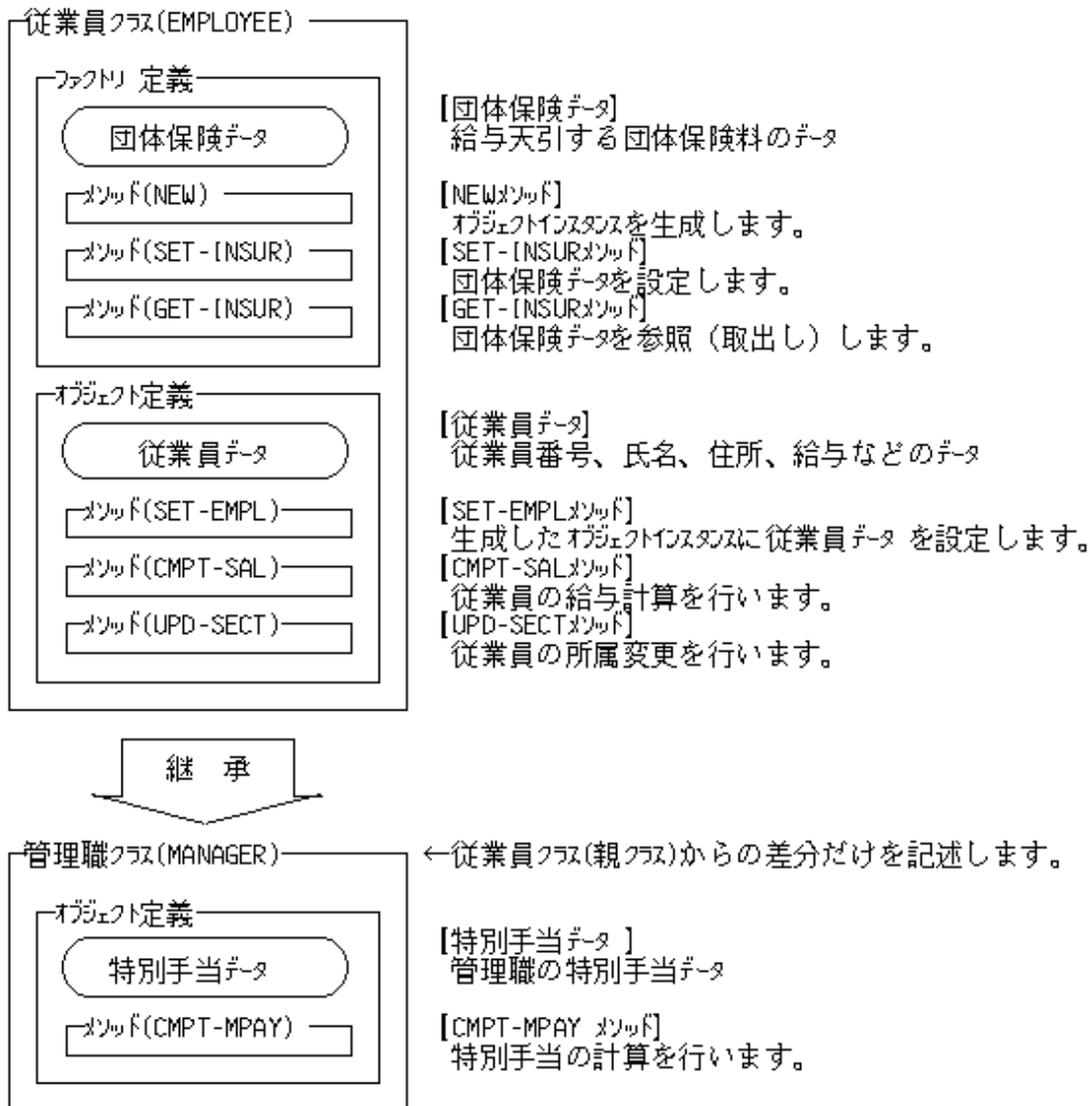
### 17.1.3.1 継承の概念と実現

#### 継承の概念

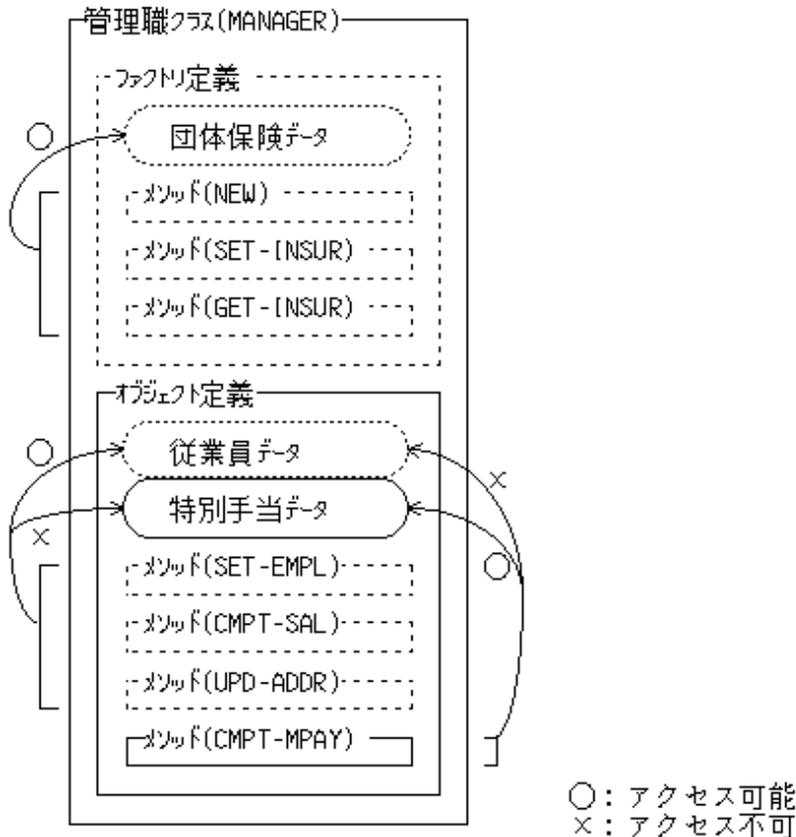
従来のプログラミング手法では、既存の部品(ルーチン)とよく似た機能を持った部品を作成する場合、既存のソースプログラムを複写し、複写したソースプログラムをベースにしてプログラミングする方法をとっていました。

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 subclasses、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 subclasses 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
      INHERITS EMPLOYEE.      ←親クラスを指定します。
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
  REPOSITORY.
    CLASS EMPLOYEE.          ←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.

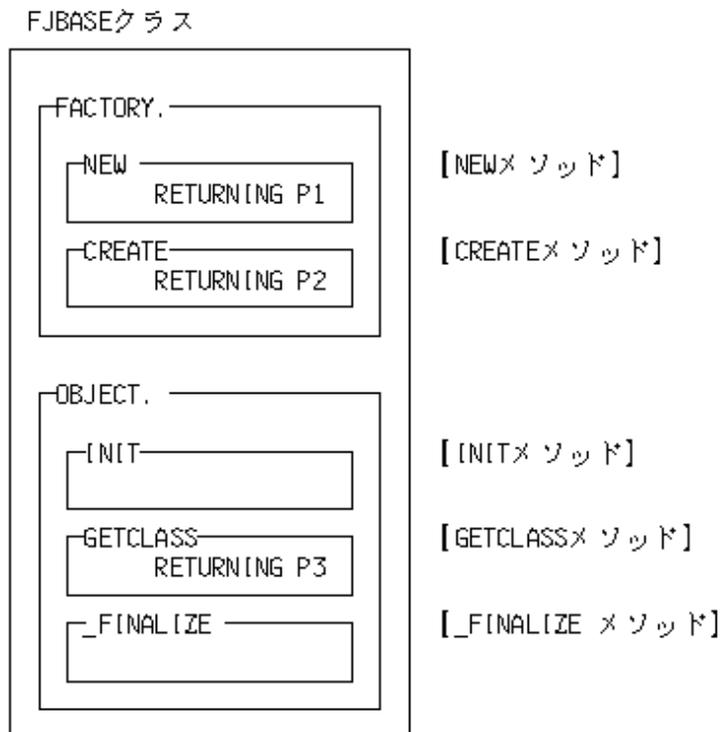
```

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。

### 17.1.3.2 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。

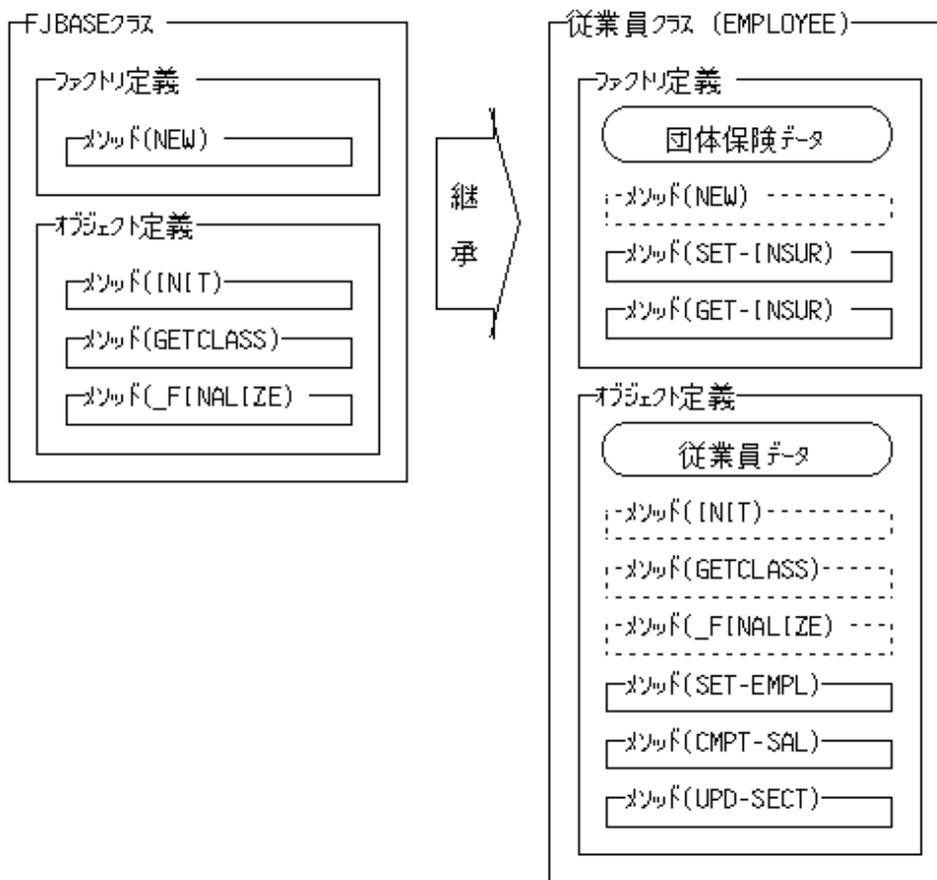


各メソッドの詳細については、“COBOL文法書”を参照してください。

INITメソッドおよび\_FINALIZEメソッドについては、“17.1.12 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。



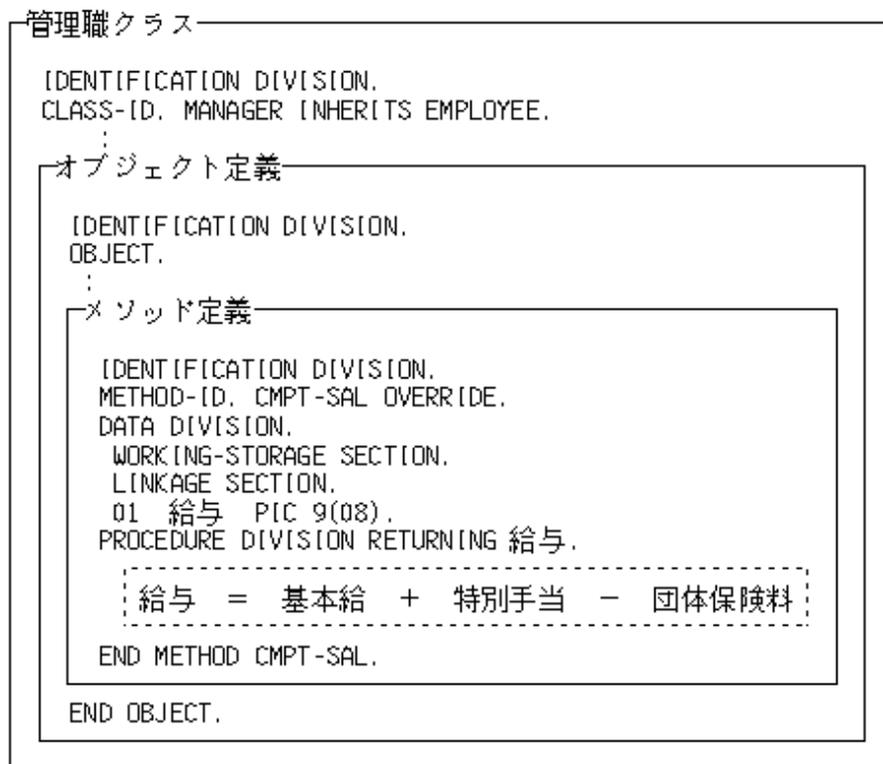
### 17.1.3.3 メソッドの上書き

クラスを継承する場合、「メソッド名やインター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。

## 17.1.4 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。

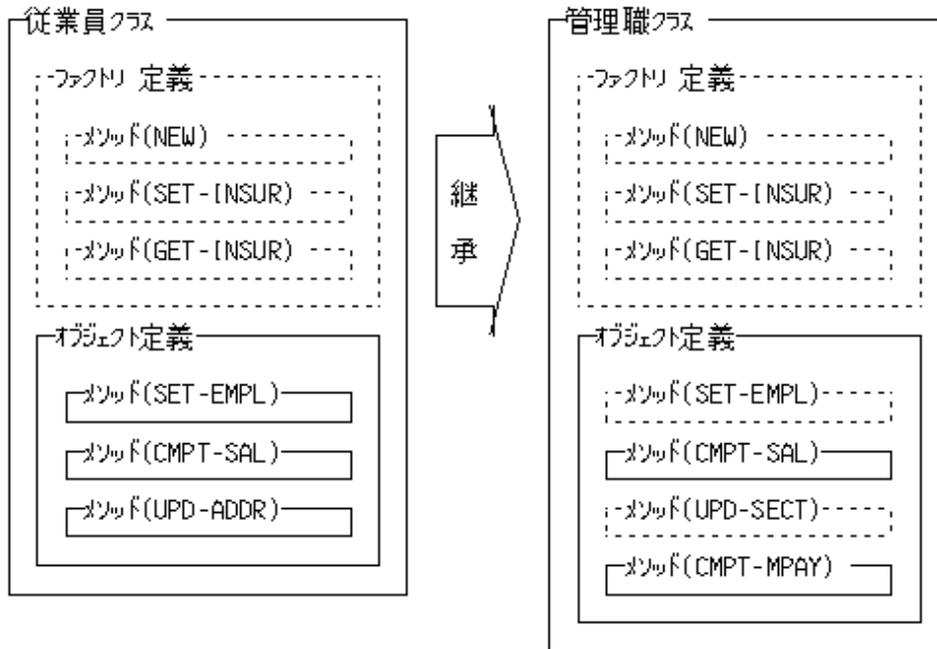
ここでは、適合の概念や規則について、具体例を用いて説明します。

### 17.1.4.1 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、メソッドのインター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ー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。

## 17.1.4.2 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、“[17.1.6.2 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.
```

```
:
```

### 17.1.4.3 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。

#### 17.1.4.3.1 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。

#### 17.1.4.3.2 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。

## 17.1.5 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、“[17.2.3.2.1 リポジトリファイル](#)”を参照してください。

### 17.1.5.1 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。

#### 17.1.5.1.1 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、“[27.6 プロジェクトブラウザ](#)”を参照してください。

### 17.1.5.2 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。

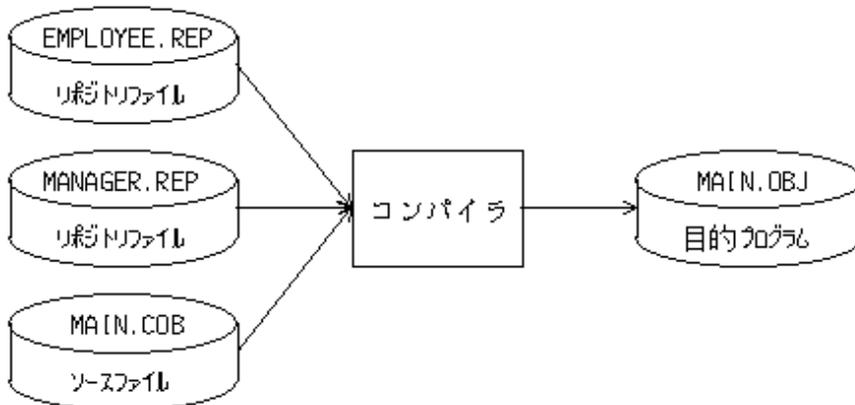
### 17.1.5.2.1 適合チェックの実現

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

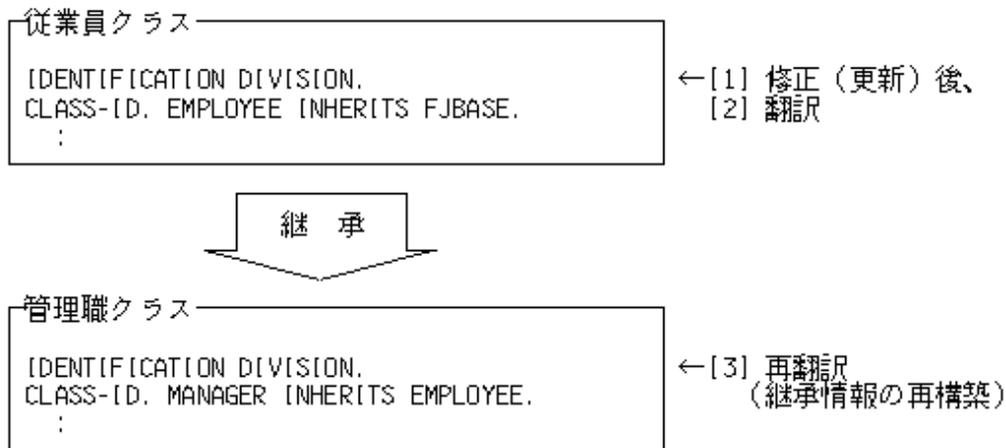
```
IDENTIFICATION DIVISION.  
PROGRAM-ID. MAIN.  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
REPOSITORY.  
    CLASS EMPLOYEE  
    CLASS MANAGER.  
:  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 OBJ-1      USAGE OBJECT REFERENCE EMPLOYEE.  
01 OBJ-2      USAGE OBJECT REFERENCE MANAGER.  
:  
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.  
:
```



### 17.1.5.3 リポジトリファイル更新の影響

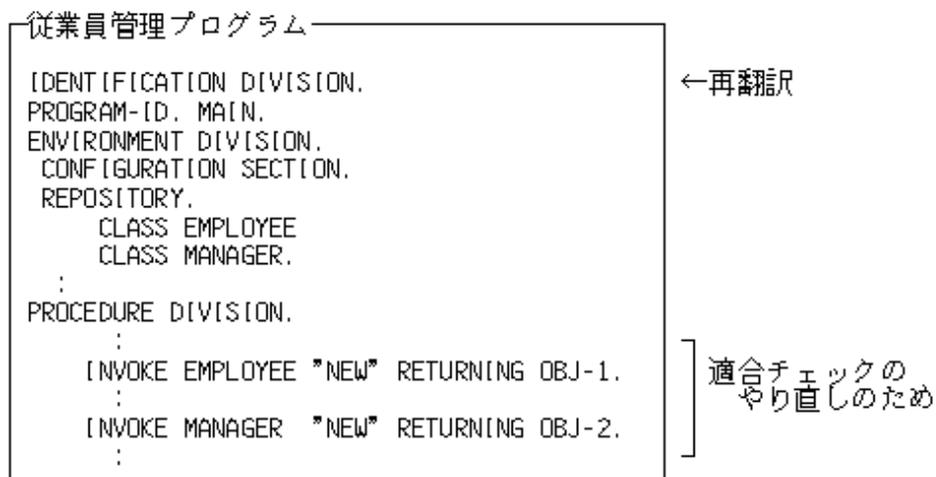
目的プログラムの生成後、親クラスや呼び出すクラスに修正が入った場合、どのように対応すればよいのでしょうか。

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られた場合、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27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。

### 17.1.6 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。

### 17.1.6.1 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、“[17.1.6.3 定義済みオブジェクト一意名SUPER](#)”を参照してください。

### 17.1.6.2 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。

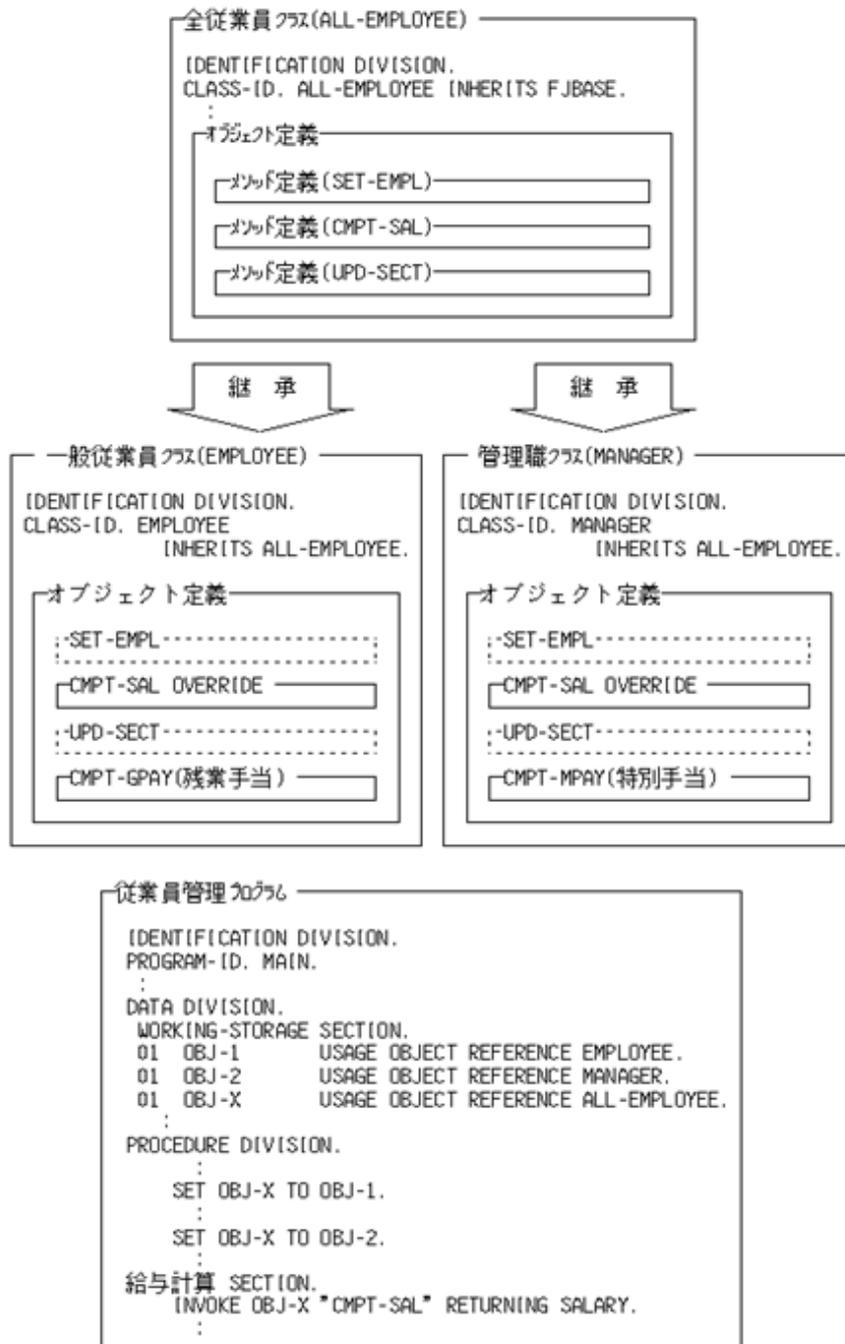
[2] OBJ-2には、従業員クラスに適合するクラスのオブジェクト参照が格納できるため、[1]の場合と同様に動的束縛になります。

[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、“[17.1.6.4 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。



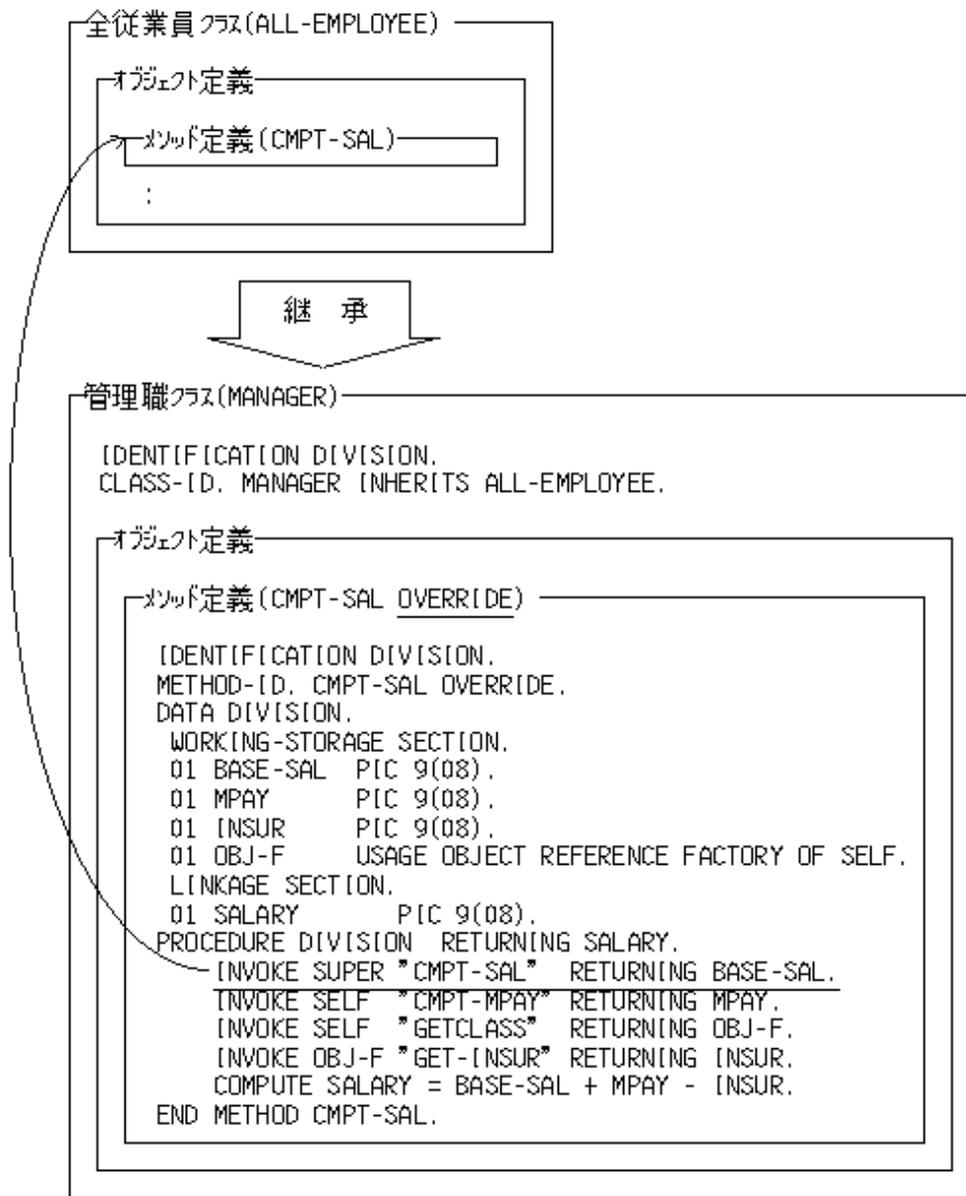
上図のようにオブジェクトを意識せず(一般従業員なのか、管理職なのか)に給与計算処理を行うことができます。つまり、1つのオブジェクト参照項目によって、複数の別定義オブジェクトを操作することができたこととなります。これを多態と呼びます。

### 17.1.6.3 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。



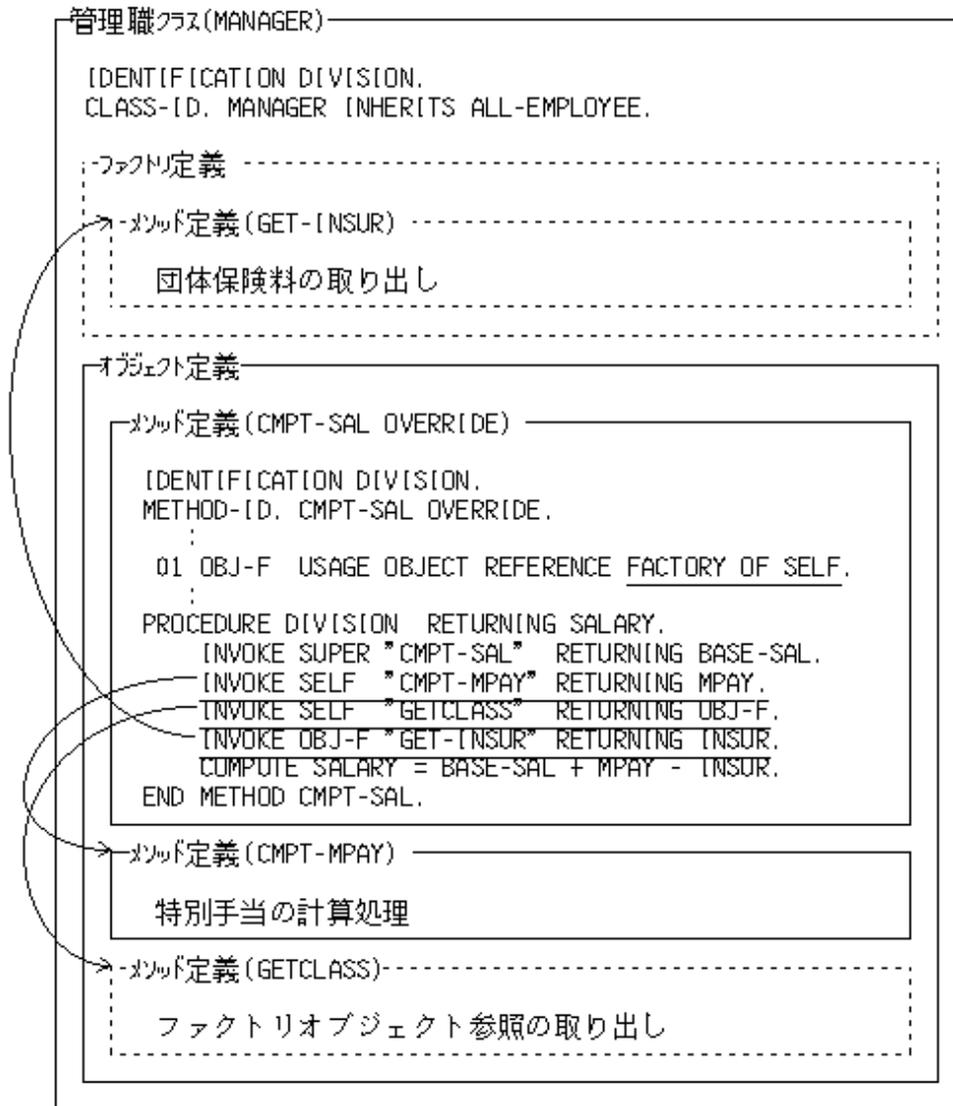
このように、親クラスを表現する場合に、定義済みオブジェクト一意名SUPERが利用されます。

#### 17.1.6.4 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。

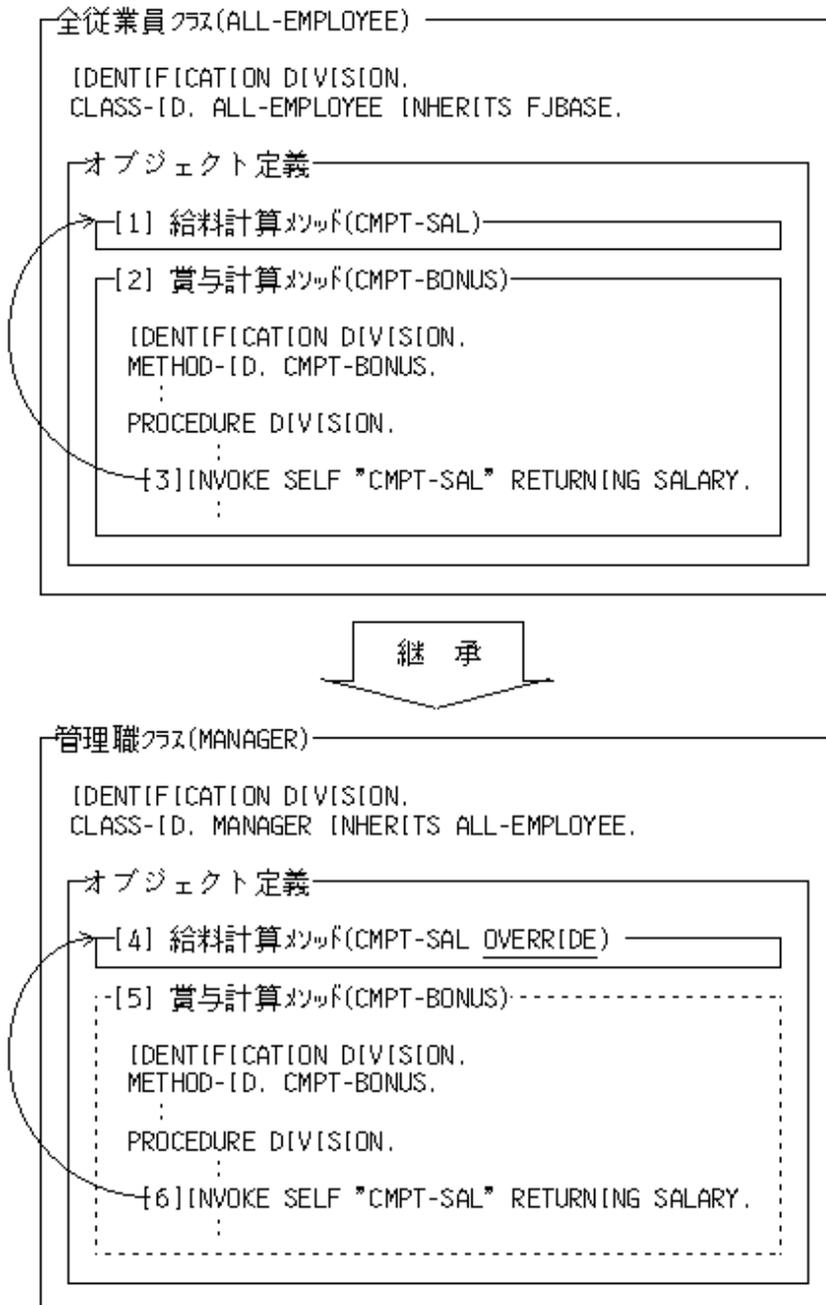
この定義済みオブジェクト一意名SELFの使用方法について説明します。

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、とときに定義済み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となります。

これも多態の1つの形態です。

### 17.1.7 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。

### 従業員クラス

```
IDENTIFICATION DIVISION.  
CLASS-ID. ALL-EMPLOYEE INHERITS FJBASE.  
:  
IDENTIFICATION DIVISION.  
METHOD-ID. UPD-SECT PROTOTYPE.  
DATA DIVISION.  
LINKAGE SECTION.  
01 NEW-SECT PIC N(20).  
PROCEDURE DIVISION USING NEW-SECT.  
END METHOD UPD-SECT.  
:
```

PROTOTYPE メソッド

インタフェースだけを記述  
します。

### UPD-SECT (分離されたメソッド)

```
IDENTIFICATION DIVISION.  
METHOD-ID. UPD-SECT OF ALL-EMPLOYEE.  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
REPOSITORY.  
CLASS ALL-EMPLOYEE.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
:  
LINKAGE SECTION.  
01 NEW-SECT PIC N(20).  
PROCEDURE DIVISION USING NEW-SECT.  
:  
END METHOD UPD-SECT.
```

→このメソッドを論理的に含  
むクラス名を指定します。

→同上

このメソッド内で使用する  
データを定義します。

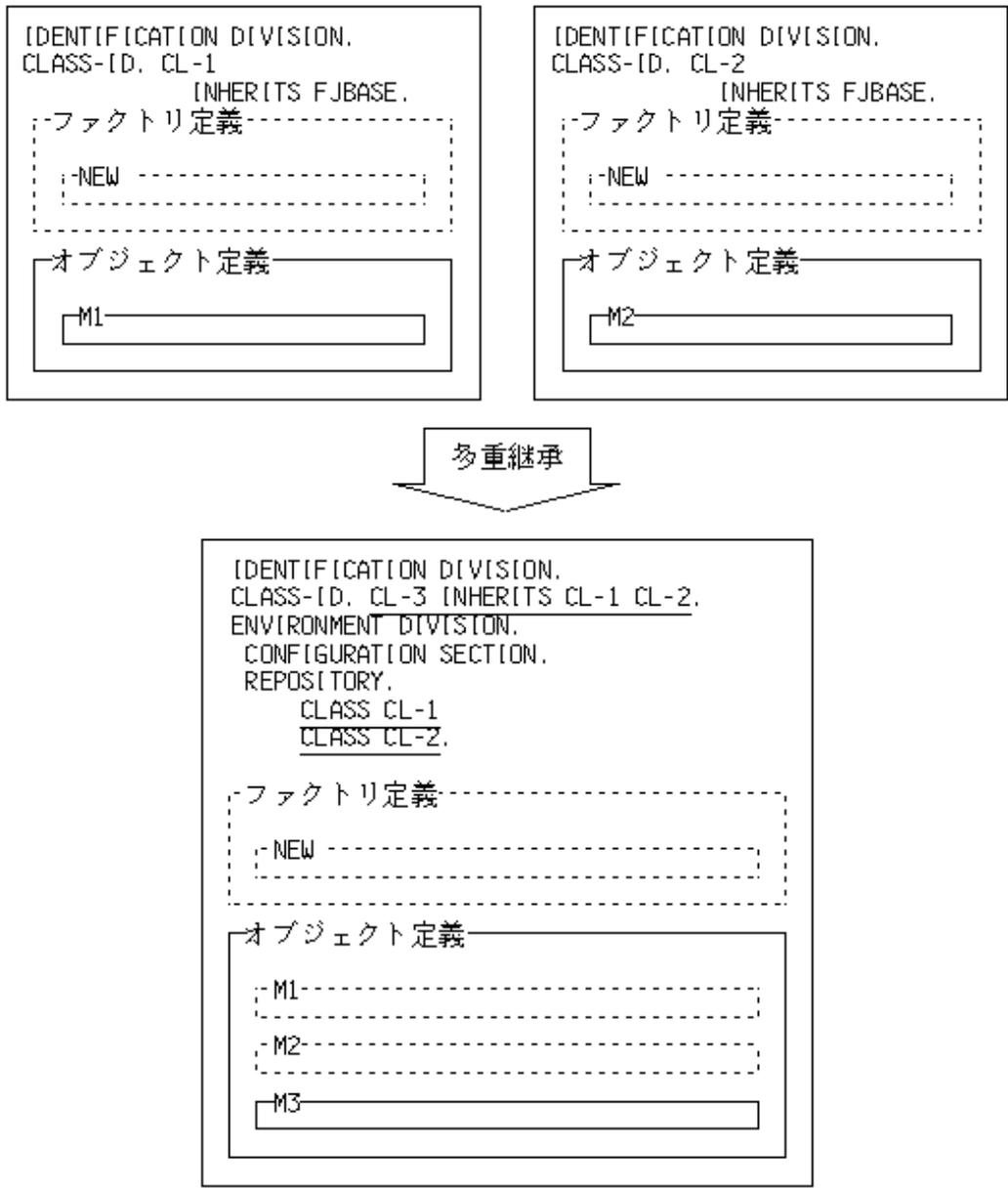
メソッドの手続きを記述し  
ます。

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。

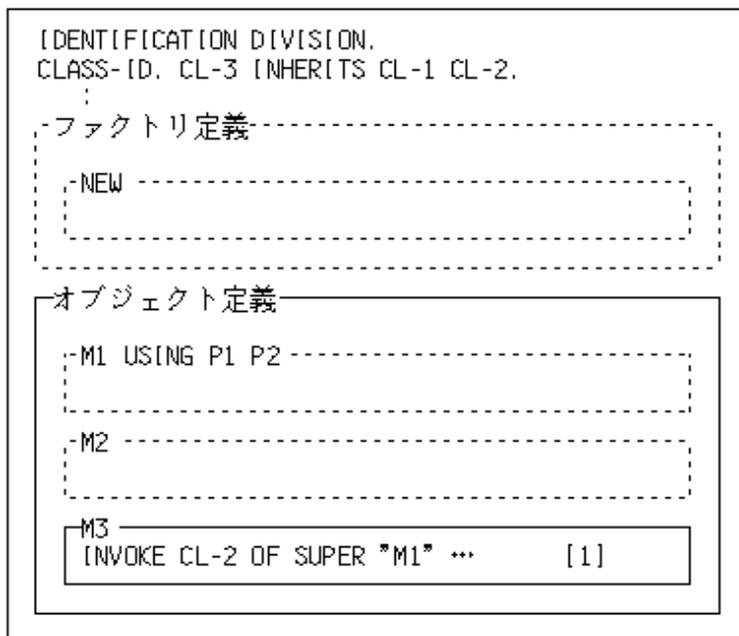
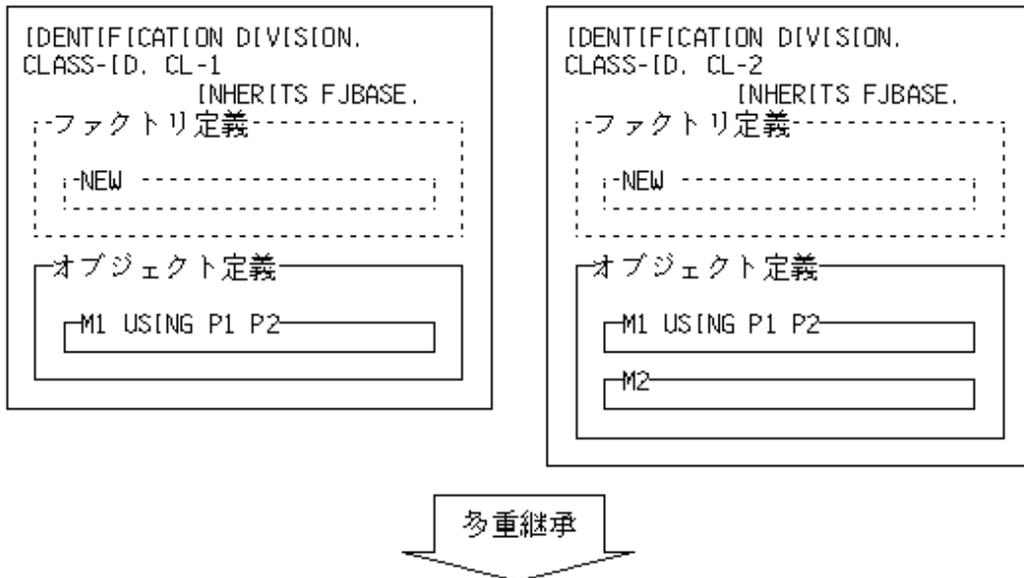
なお、分離されたメソッドの翻訳時に、そのメソッドを論理的に含むクラスのリポジトリファイルを入力する必要があるため、メソッドを翻訳するためには、先にクラス定義を翻訳しておく必要があります。詳細については、“[17.2 オブジェクト指向プログラミングの開発](#)”を参照してください。

## 17.1.8 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)。



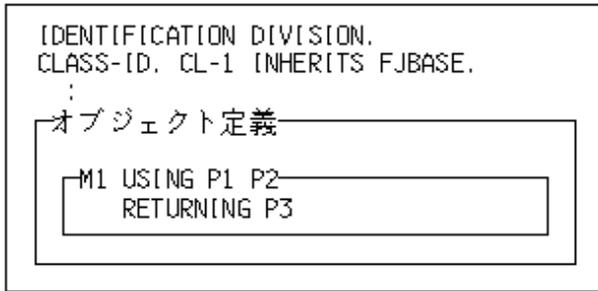
このように複数の親クラスで同名のメソッドが定義されていた場合、利用者が明にこのメソッドを上書きしない限り、引き継がれます。引き継がれるメソッドは、INHERITS句に指定されたクラス名の並びを左から検索して、最初にそのメソッドが見つかるクラスのメソッドになります。

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。

### 17.1.9 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。

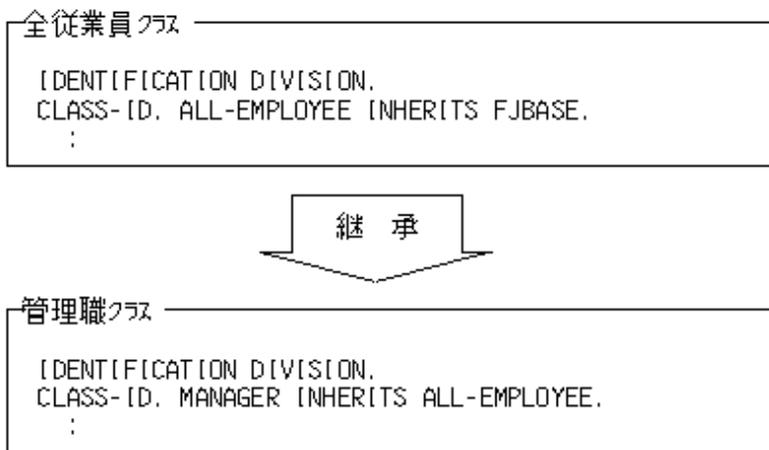
```

:
WORKING-STORAGE SECTION.
01 OBJ-1 USAGE OBJECT REFERENCE CL-1.
PROCEDURE DIVISION.
:
  IF OBJ-1 :: "M1"(P1 P2) = 0 THEN ...
:

```

### 17.1.10 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.  
    SET OBJ-2 TO OBJ-1.    ←エラー    ** [1]  
:  
    ** [2]
```

[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います。

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が指定されたのみならず適合チェックが行われるため、代入可能となります。

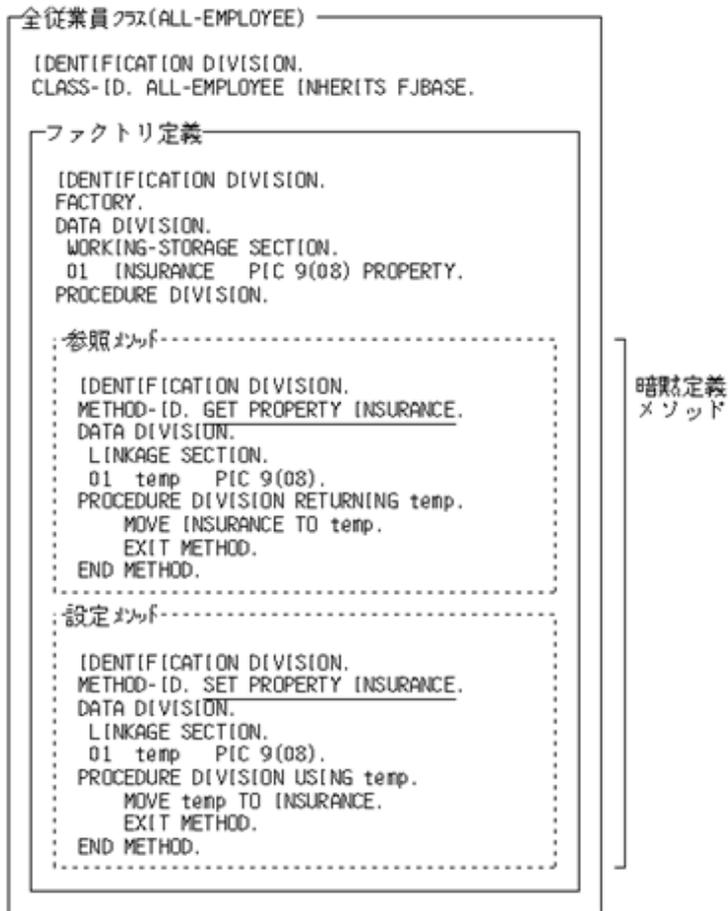
このように、オブジェクト指定子を用いた場合、翻訳時は指定されたクラス名で適合チェックが行われますが、実行時は実際に格納されているオブジェクト参照によって適合チェックが行われます。つまり、適合に違反している場合は、実行時にチェックアウトされます。

## 17.1.11 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。

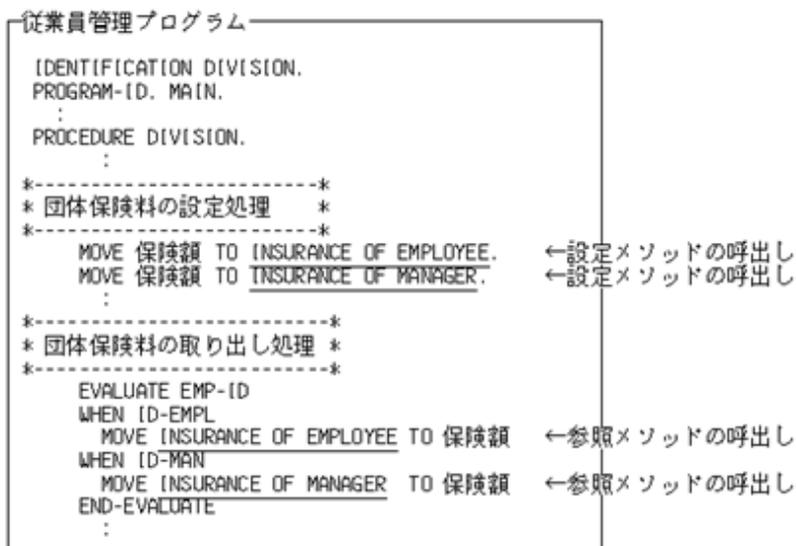
これは、PROPERTY句の指定によってデータを設定、参照するメソッドを自動生成することにより実現しています。

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。



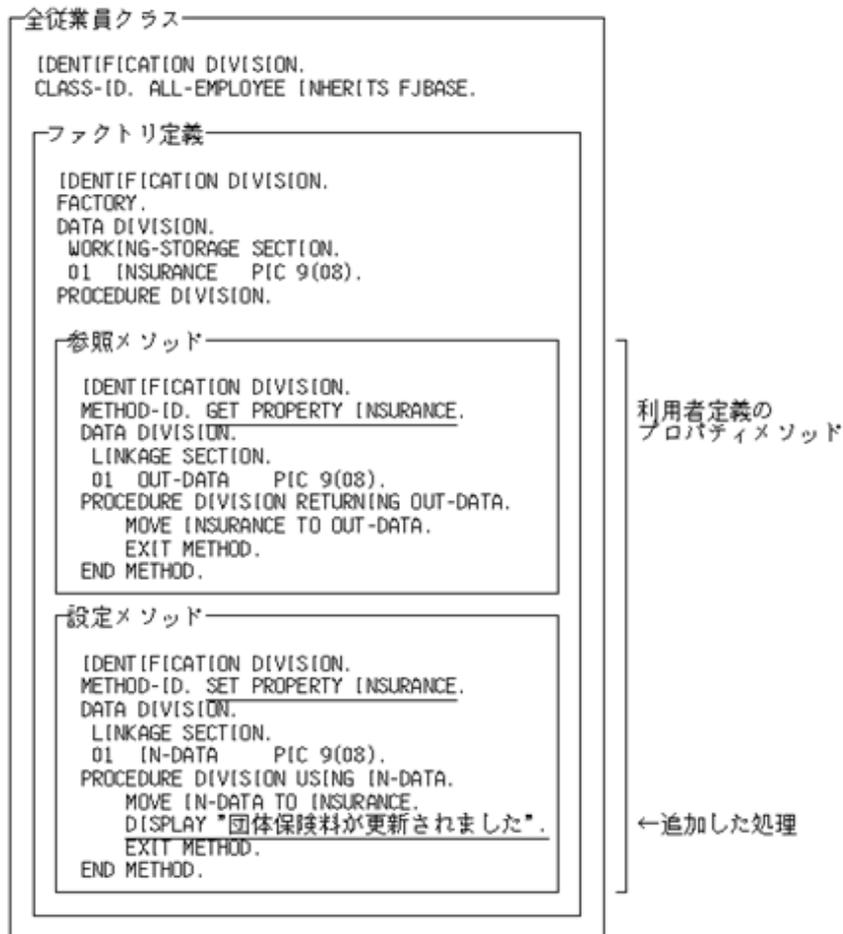
上図のようにPROPERTY句によって暗黙定義されるメソッドをプロパティメソッドと呼びます。

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。



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。

## 17.1.12 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尽きたことを自動的に判断して行います。[参照]“[17.1.2.2 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ときに行い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、“[17.1.3.3 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 I-F-SAMPLE.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 OBJREF USAGE OBJECT REFERENCE I-F-SAMPLE.  
PROCEDURE DIVISION.  
    INVOKE I-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. I-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。

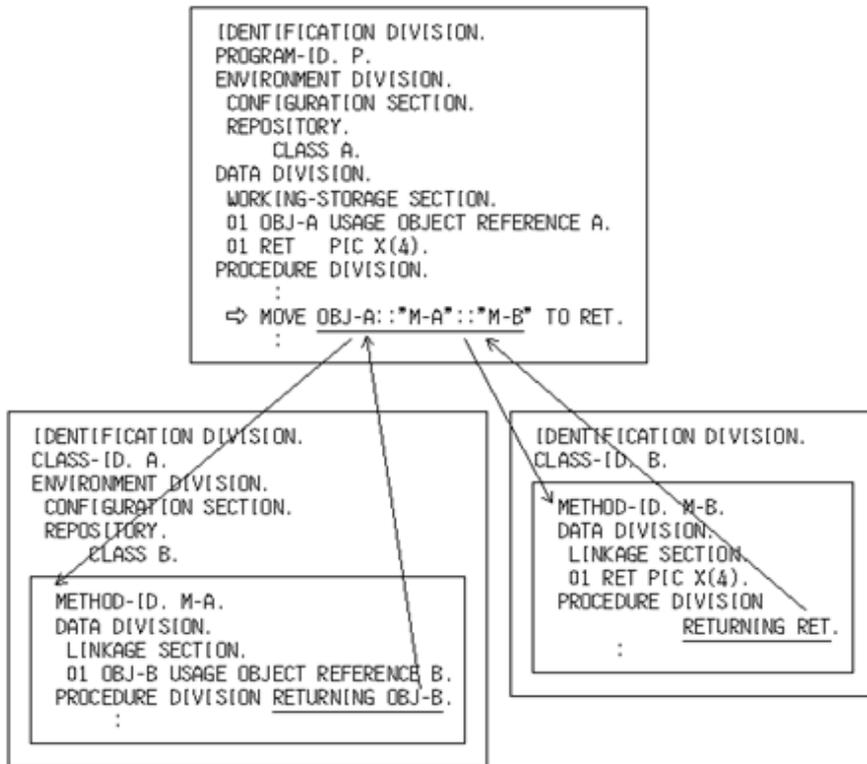
なお、\_FINALIZE を使用しないのであれば、従来どおり、ソースの一部を再翻訳してアプリケーションを構成することが可能です。

\_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。

### 17.1.13 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.
:

```

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。

## 17.1.14 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、“[27.4クラスの公開](#)”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。

### 17.1.14.1 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

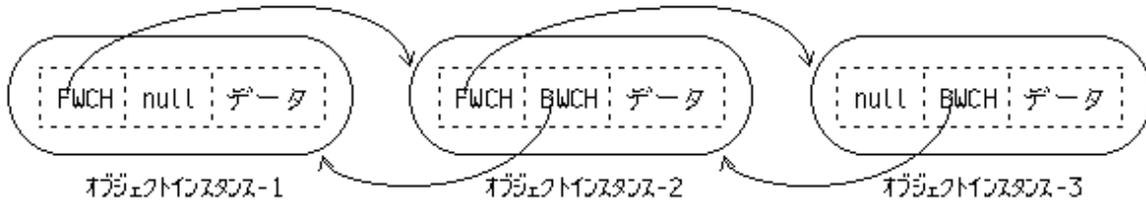
それぞれのパターンについて、以下に具体的に説明します。

#### 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。

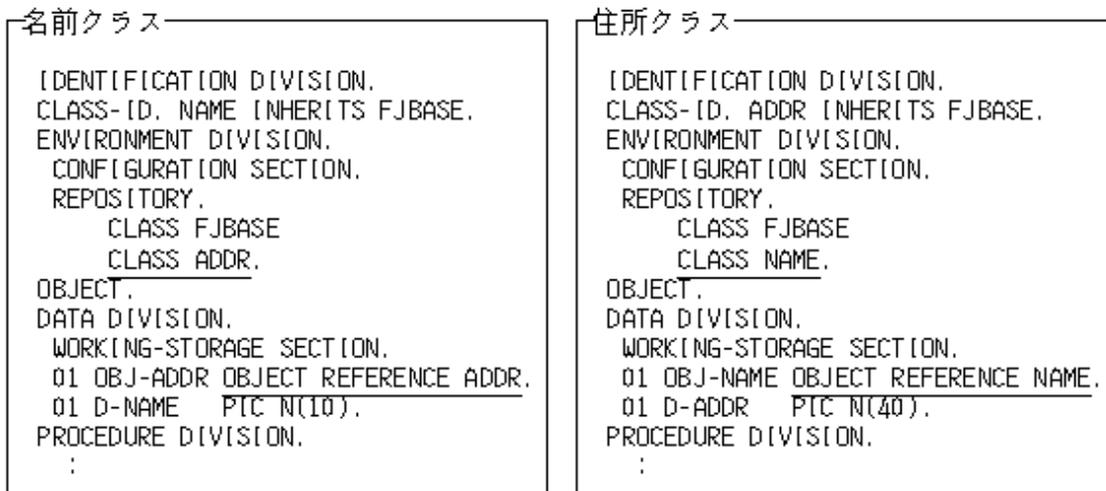
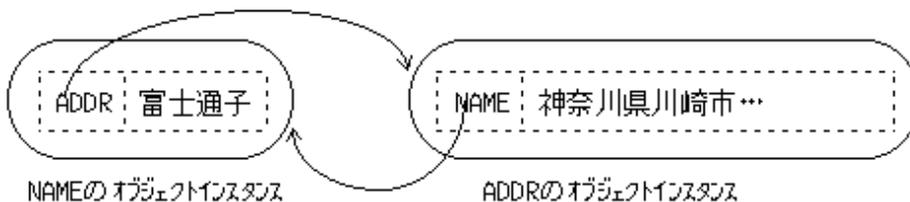


図17.3 実行時のオブジェクトインスタンスイメージ



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 PTC N(10),  
PROCEDURE DIVISION,  
:
```

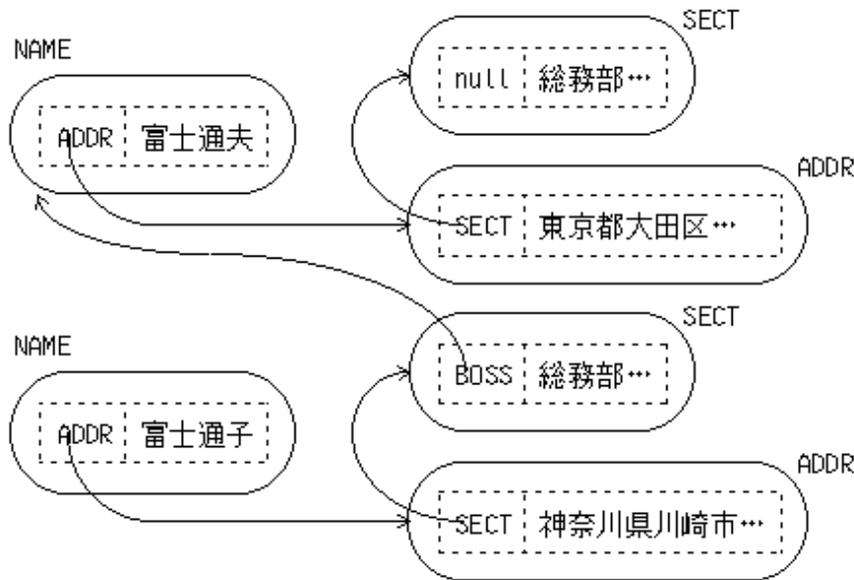
#### 住所クラス

```
[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 PTC 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 PTC N(20),  
PROCEDURE DIVISION,  
:
```

図17.4 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。

### 17.1.14.2 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。

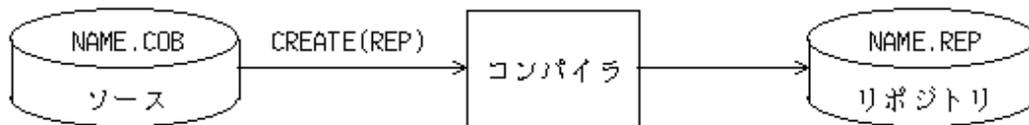
名前クラスを翻訳するためには、住所クラスのリポジトリファイルが必要です。住所クラスのリポジトリファイルを生成するには住所クラスを翻訳しなければなりません。その際、名前クラスのリポジトリファイルが必要になります。いわゆる、「タマゴが先か、ニワトリが先か」の状態に陥ってしまうわけです。

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

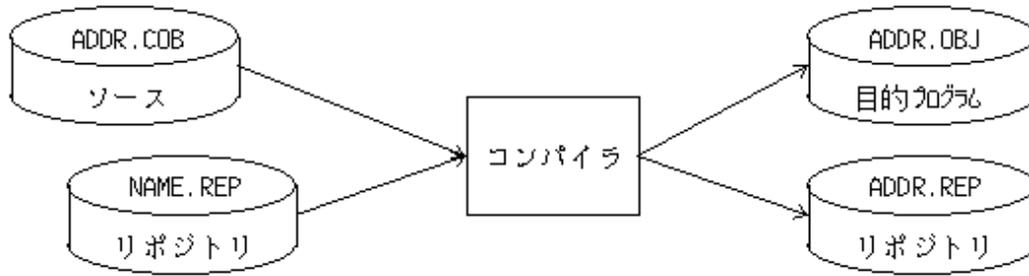
翻訳オプションCREATE(REP)を指定して名前クラスを翻訳します。

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。



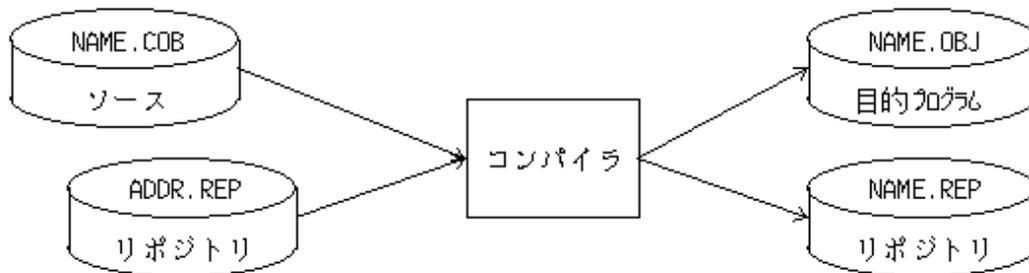
#### ステップ2：生成したリポジトリを利用して、住所クラスを翻訳

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。

### 17.1.14.3 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、通常の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"
EXPORTS _NAME_FACTORY
```

←ライブラリ名を指定  
←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。

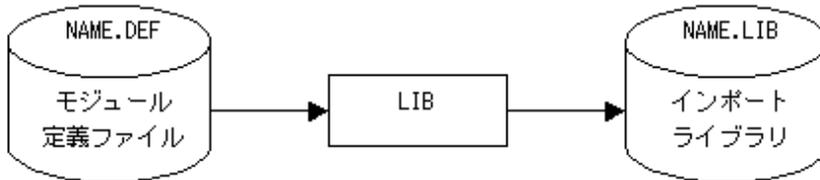
エントリ名の規約は、以下のとおりです。

- 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

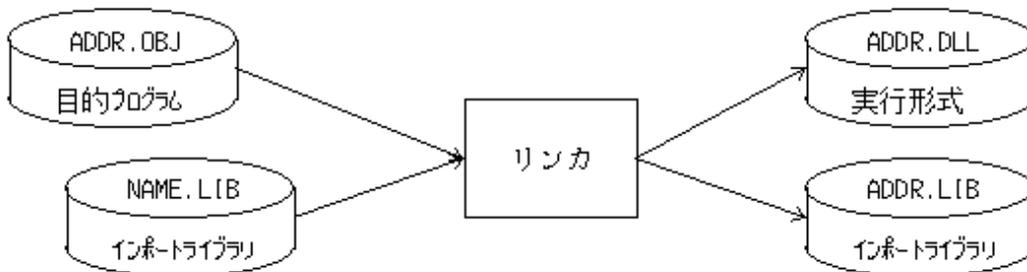
ステップ1で作成したモジュール定義ファイルを利用して、名前クラスのインポートライブラリを作成します。

```
LIB /OUT:NAME.LIB /DEF:NAME.DEF /MACHINE:IX86
```



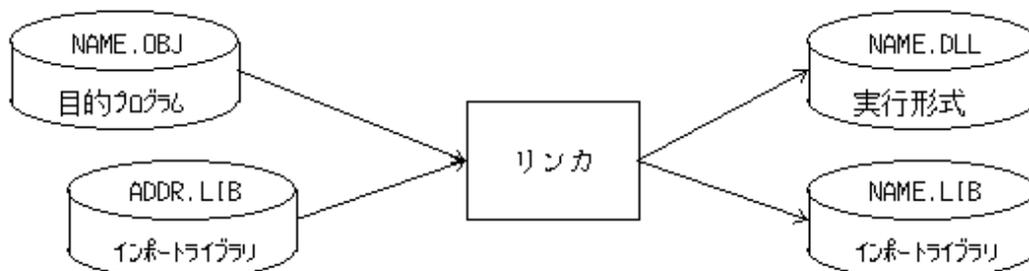
### ステップ3：住所クラスの実行形式を作成

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、自クラスの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。

## 17.1.14.4 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。

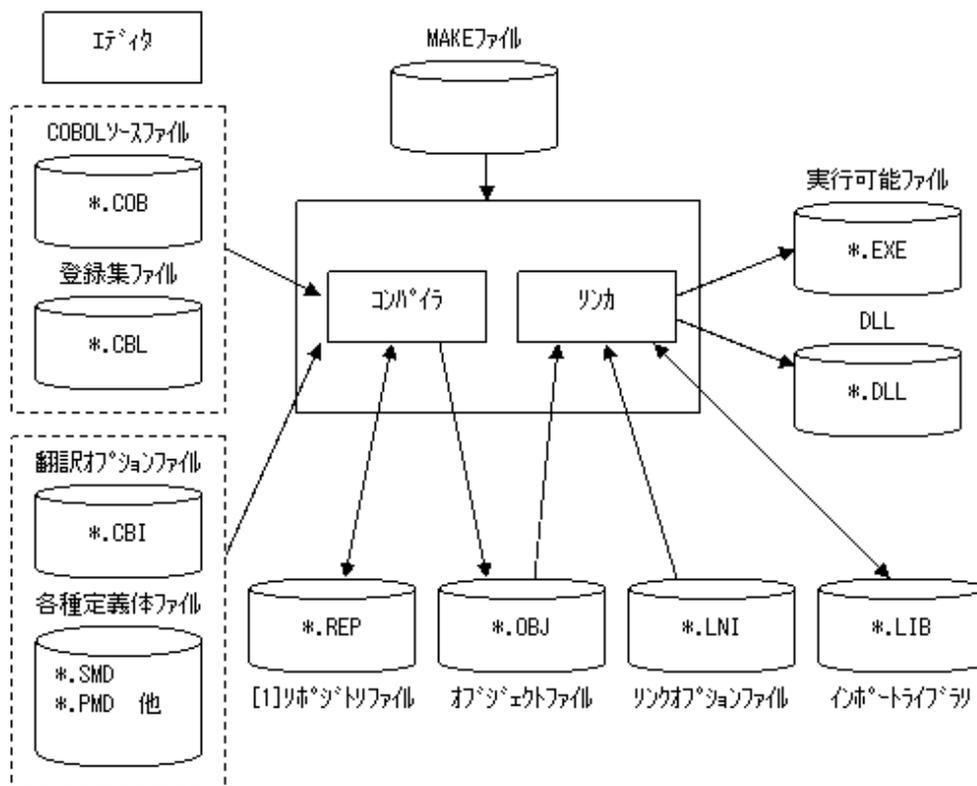
## 17.2 オブジェクト指向プログラミングの開発

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。

### 17.2.1 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。



プロジェクトマネージャのプロジェクト管理機能を使用した場合の必要な資源とツールの関係については、“[27.1 オブジェクト指向プログラミングで使用する資源](#)”を参照してください。

表17.1 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	継承および適合チェックが必要になるCOBOLソースファイルを翻訳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。

## 17.2.2 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. クラスの設計を行います。このとき、既存クラスから再利用できるクラスを選定します。  
[参照]“[17.2.3 使用するクラスを選定](#)”
2. ソースファイルを作成および編集します。
3. クラスの翻訳およびリンクを行います。クラスの翻訳およびリンクでは、資源の依存関係が複雑になるケースが多いため、MAKEファイルを使用することをおすすめします。MAKEファイルについては、Microsoft社のオンラインヘルプを参照してください。
4. プログラムをデバッグおよび実行します。
5. クラスを公開します。[参照]“[17.2.4 クラスの公開](#)”

なお、プロジェクトマネージャを使用したオブジェクト指向プログラミングの開発手順は、“[27.1 オブジェクト指向プログラミングで使用する資源](#)”を参照してください。

以降、オブジェクト指向プログラムの開発で特に注意が必要な項目について説明します。

## 17.2.3 使用するクラスの選定

---

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ーフェース

### 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ーフェース

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わからないと、思ったとお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ー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。

### 17.2.3.1 プログラム構造

#### 17.2.3.1.1 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間を下の図に示します。

図17.5 翻訳単位とリンク単位が一致する場合

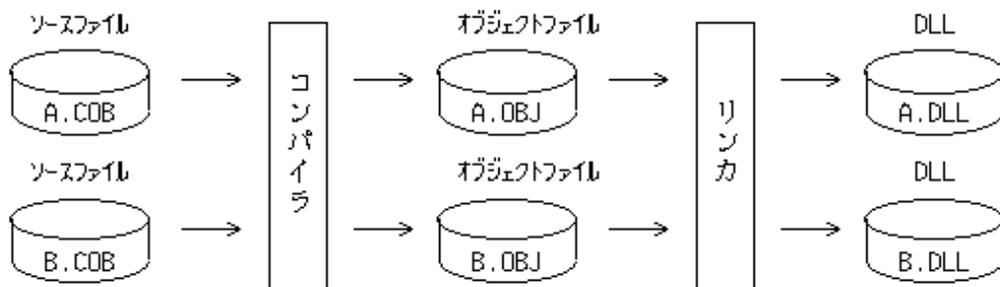
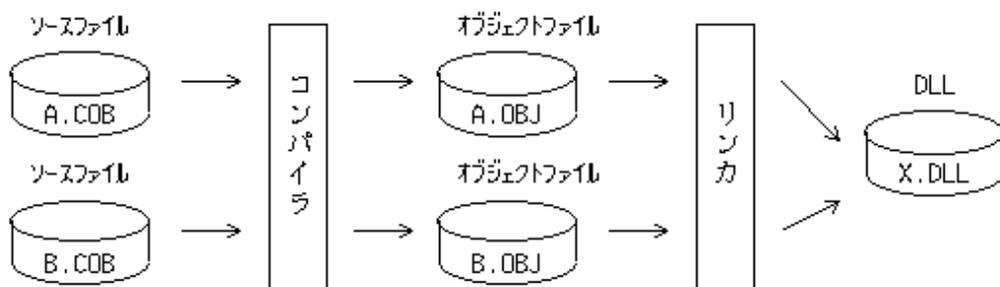


図17.6 翻訳単位とリンク単位が一致しない場合



#### 17.2.3.1.2 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、“[17.3.2 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

プロジェクトの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。

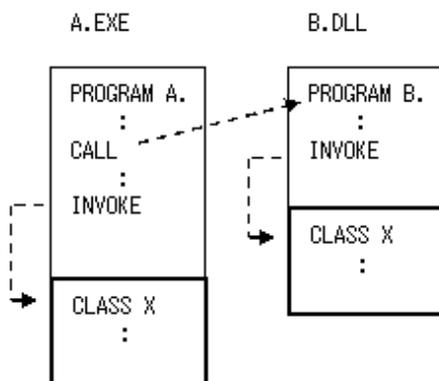
ここでの説明を参考にして、プログラム構造を設計してください。

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、“[17.2.3.3.1 インポートライブラリ](#)”を参照してください。

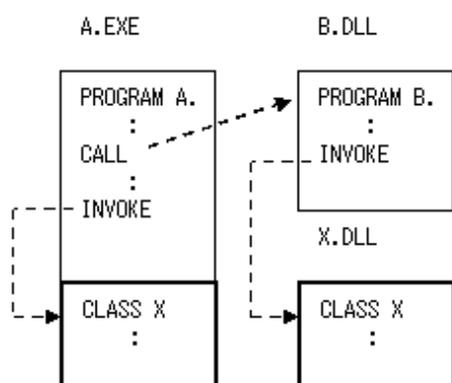
## 注意

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)

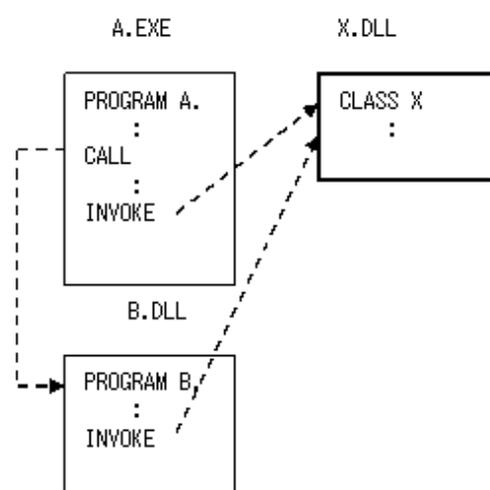
1. 静的構造により、同一クラスのオブジェクトが2つ存在する場合



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

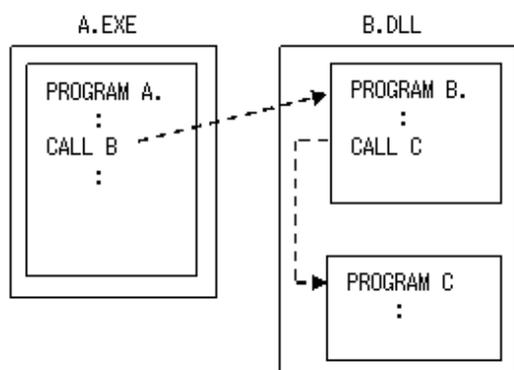


参考

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。



## 17.2.3.2 翻訳処理

ここでは、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で、特に意識する必要がある以下について説明します。

- ・ リポジトリファイルと翻訳の手順

なお、一般的な翻訳処理については、翻訳およびリンクの方法については、“[第3章 プログラムを翻訳する](#)”と“[第4章 プログラムをリンクする](#)”を参照してください。

### 17.2.3.2.1 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17.7 リポジトリファイルの出力

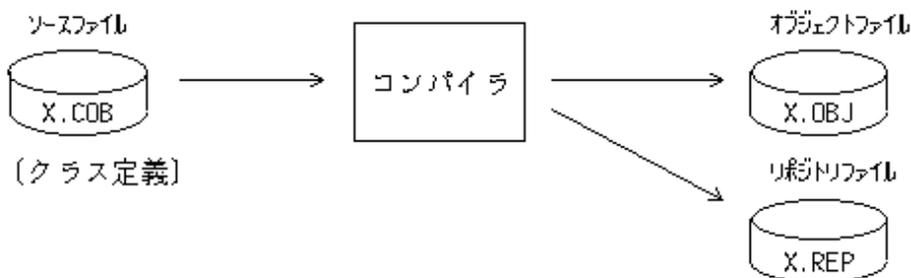
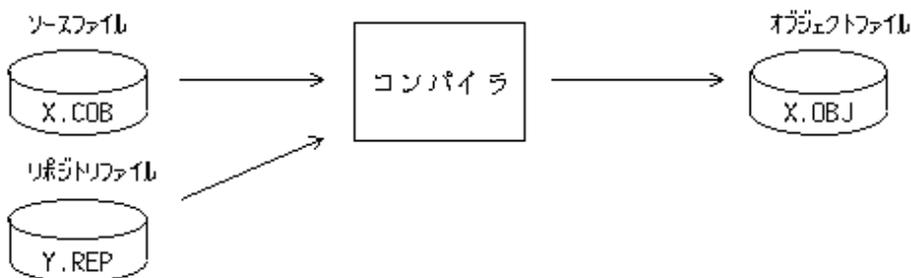


図17.8 リポジトリファイルの入力



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、“[27.5 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。

Sample15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なことがわかります。

“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なことがわかります。

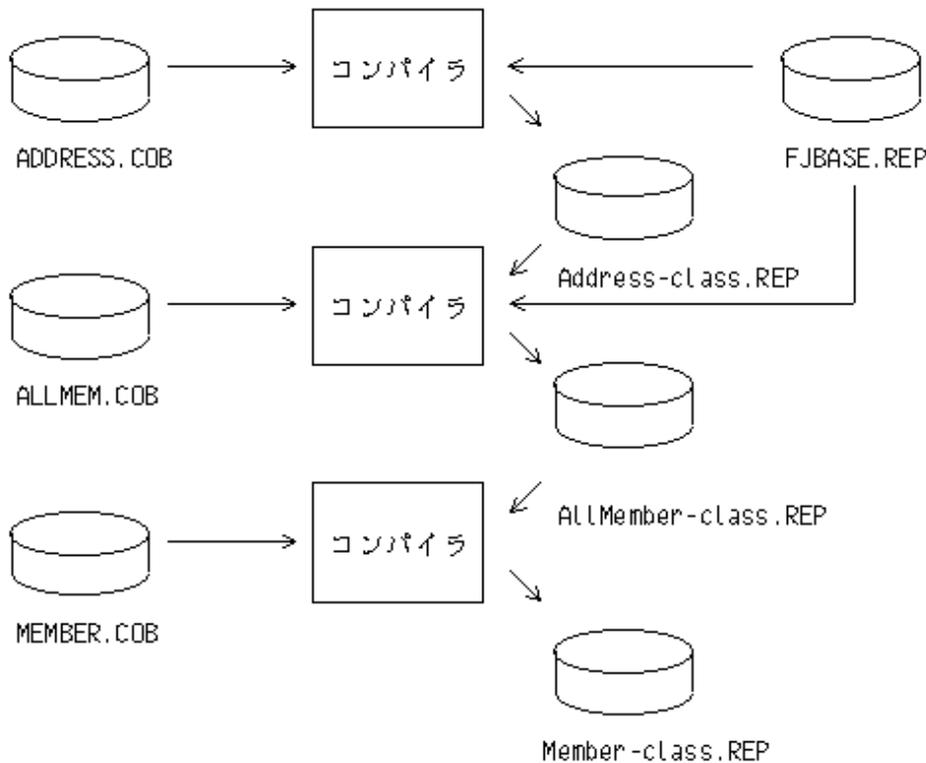
“ADDRESS.COB”のソースプログラムは、以下のように記述されています。

```
      :  
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なことがわかります。

以上を整理すると、資源の関係は以下ようになります。

図17.9 翻訳順序



“図17.9 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

### 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17.9 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フォルダ

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フォルダは、翻訳オプション-drの指定によって、下表のように決まります。  
[参照]“J.1.10 -dr(リポジトリファイルの入出力先フォルダの指定)”

-dr オペラント	フォルダ名
あり	-dr オペラントで指定したフォルダ
なし	COBOL ソースファイルが存在するフォルダ

### 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17.9 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フォルダ

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フォルダは、翻訳オプション-Rおよび-drの指定によって、下表のように順序付けられます。[参照]“[J.1.19 -R \(リポジトリファイルの入力先フォルダの指定\)](#)”、“[J.1.10 -dr \(リポジトリファイルの入出力先フォルダの指定\)](#)”

-R オペランド	-dr オペランド	検索フォルダの順序
あり	あり	(1) -R オペランドで指定したフォルダ (2) -dr オペランドで指定したフォルダ (3) カレントフォルダ
	なし	(1) -R オペランドで指定したフォルダ (2) カレントフォルダ
なし	あり	(1) -dr オペランドで指定したフォルダ (2) カレントフォルダ
	なし	(1) カレントフォルダ

## COBOL32コマンドで翻訳を行う例

以下に、allmem.cobをcobol32コマンドで翻訳した例を示します。cobol32コマンドの入力形式については、“[J.1 COBOL32コマンド](#)”を参照してください。

```
COBOL32 -c -R C:¥COBOL¥SAMPLES¥SAMPLE19 -dr C:¥COBOL¥SAMPLES¥SAMPLE19 allmem.cob
```

入力 : allmem.cob (ソースファイル)  
ADDRESS-CLASS.rep (リポジトリファイルC:¥COBOL¥SAMPLES¥SAMPLE19に格納)  
MEMBERMASTER-CLASS.rep

(リポジトリファイルC:¥COBOL¥SAMPLES¥SAMPLE19に格納)

出力 : allmem.o (オブジェクトファイル)  
ALLMEMBER-CLASS.rep

(リポジトリファイルC:¥COBOL¥SAMPLES¥SAMPLE19に格納)

オプション : -c (翻訳だけを行う指定)  
-R (リポジトリファイルの入力先フォルダ)  
-dr (リポジトリファイルの入出力先フォルダ)

## 相互参照クラスの翻訳

相互参照クラスの翻訳を行う場合、依存リポジトリファイルの作成にテクニックが必要になります。相互参照クラスの翻訳については“[17.1.14.2 相互参照クラスの翻訳](#)”を参照してください。

## 17.2.3.3 リンク処理

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、リンク処理について説明します。

### 17.2.3.3.1 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 [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	—
	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) [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。

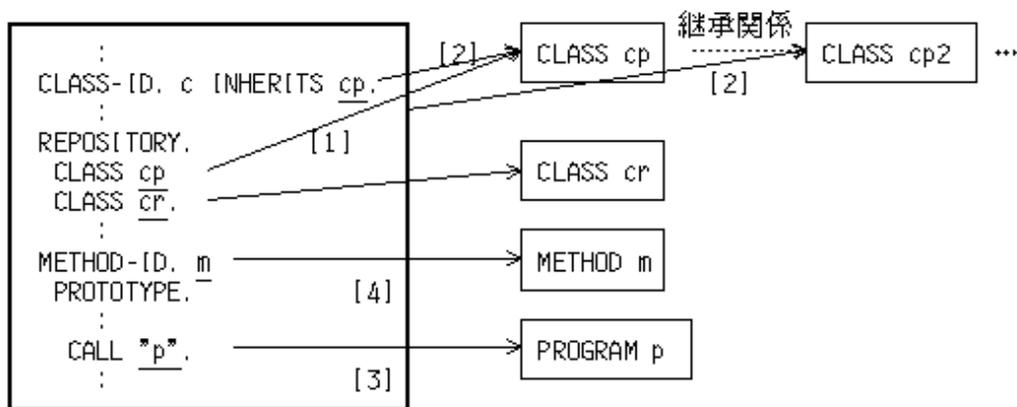
各定義間のリンク関係を下図に示します。

なお、図中の実線矢印がリンクを表します。

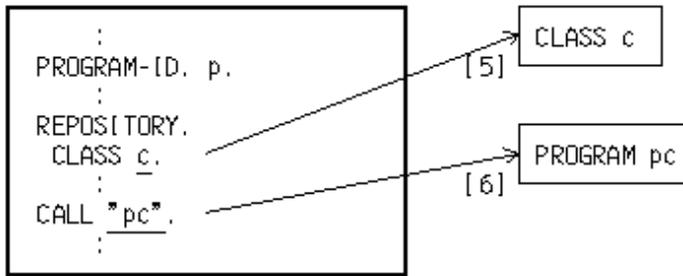
実線矢印で示されたリンク関係を、動的リンク構造で解決する場合にはインポートライブラリが必要になります。

各定義間のリンク

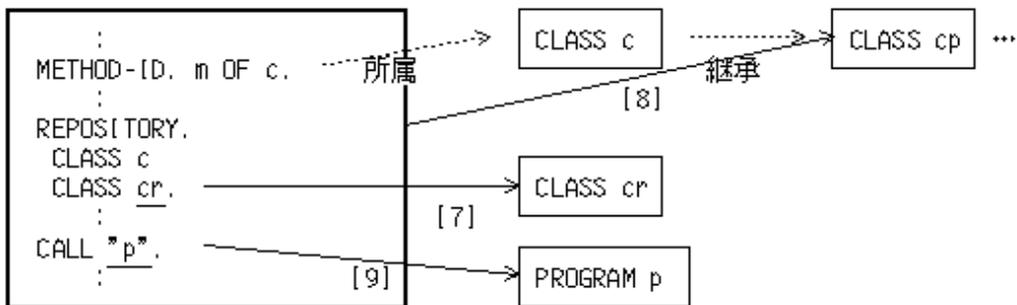
クラス定義のリンク



## 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



- ALLMEM.COB



• ADDRESS.COB

```

:
CLASS-ID. Address-class ...
:

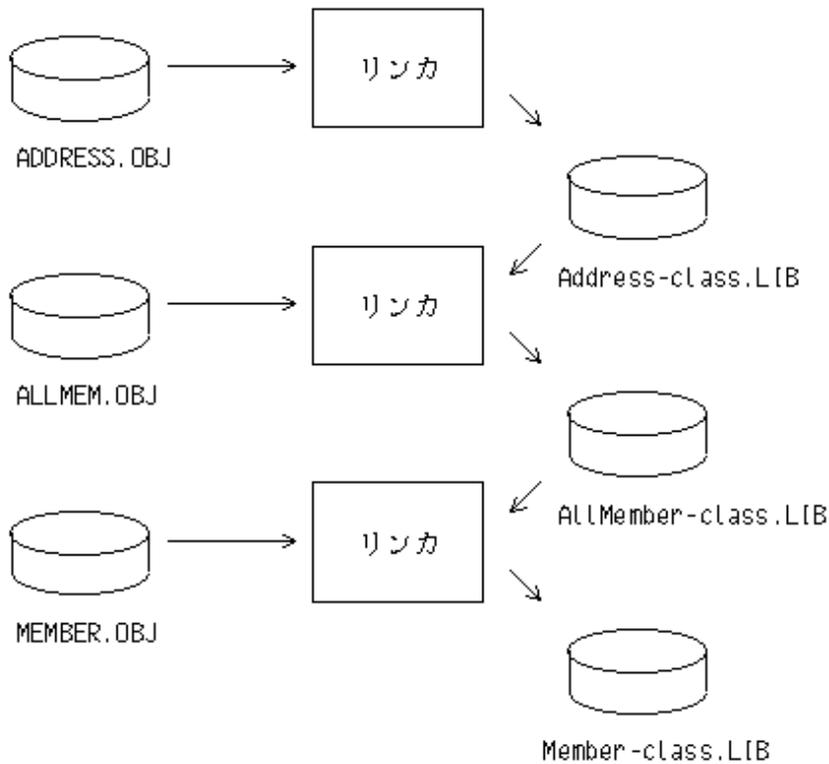
```

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に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/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17.10 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17.10 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17.10 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

```
LINK ADDRESS.OBJ F3BICIMP.LIB /NOENTRY /DLL /OUT : Address-class.DLL
```

入力 : ADDRESS.DLL  
出力 : Address-class.DLLとAddress-class.LIB

```
LINK ALLMEM.OBJ Address-class.LIB F3BICIMP.LIB /NOENTRY /DLL /OUT : AllMember-class.DLL
```

入力 : ALLMEM.OBJとAddress-class.LIB  
出力 : AllMember-class.DLLとAllMember-class.LIB

```
LINK MEMBER.OBJ AllMember-class.LIB F3BICIMP.LIB /NOENTRY /DLL /OUT : Member-class.DLL
```

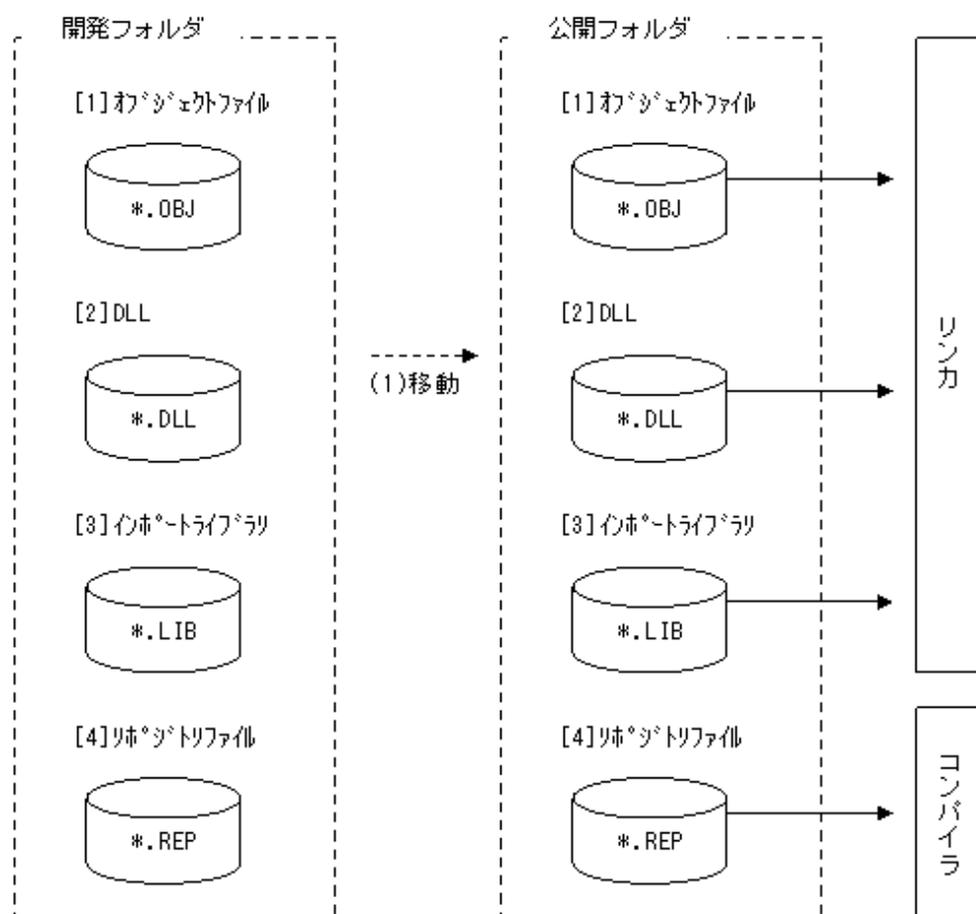
入力 : MEMBER.OBJとAllMember-class.LIB  
出力 : Member-class.DLLとMember-class.LIB

## 17.2.4 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。



なお、プロジェクトマネージャを使用した方法については、“[27.4 クラスの公開](#)”を参照してください。

### (1) 移動

クラスを再利用するために必要な資源を、開発フォルダから公開フォルダへ移動します。開発フォルダとは、開発中の各資源が格納されていたフォルダを指します。

公開フォルダとは、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フォルダを指します。

クラスを再利用するために必要な資源は、このフォルダ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17.3 公開資源”のようになります。

表17.3 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。
—	ドキュメント	クラスの機能、インタフェース(メソッド名、パラメタ、プロパティ名など)、必要な資源(リポジトリファイル名、共用オブジェクト名など)をクラスの利用者に伝えるために必要です。



## 参考

クラスを公開する場合のドキュメントでは、最低でも以下の内容を記述する必要があります。

- ・ 共用オブジェクトファイル名
- ・ 継承するクラスを含むクラス名とその機能概要
- ・ すべてのメソッド名とその機能概要
- ・ すべてのプロパティ名とその機能概要
- ・ メソッドまたはプロパティのインタフェース(パラメタ、復帰値)
- ・ リポジトリファイル名
- ・ その他の注意事項

## 17.3 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ます。

### 17.3.1 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。

#### 17.3.1.1 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。

#### 17.3.1.2 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

プロ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箇所にまとめて記述することができます。

## 注意

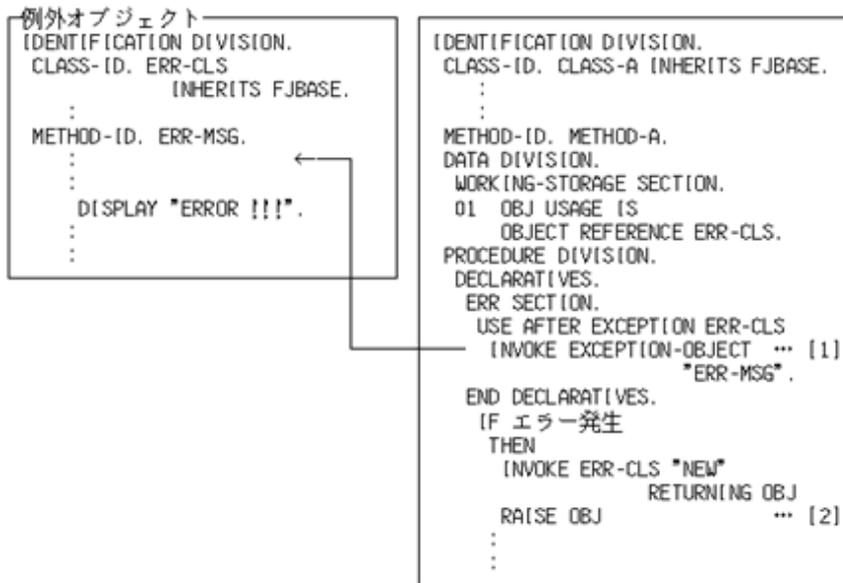
- 例外オブジェクトには、オブジェクトインスタンスを指定してください。
- USE文に記述するクラス名は、リポジトリ段落に宣言してください。
- 宣言節に例外オブジェクトに対するUSE文が複数記述されている場合、先頭の例外手続きが実行されます。以下のような例外条件が発生した場合、[1]の例外手続きが実行されます。
  - 例外オブジェクトとしてCLASS-Aのオブジェクトが指定されている。
  - CLASS-AがCLASS-BおよびCLASS-Cを継承している。

```
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。
  - 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。
  - 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。

### 17.3.1.3 RAISE文の動作

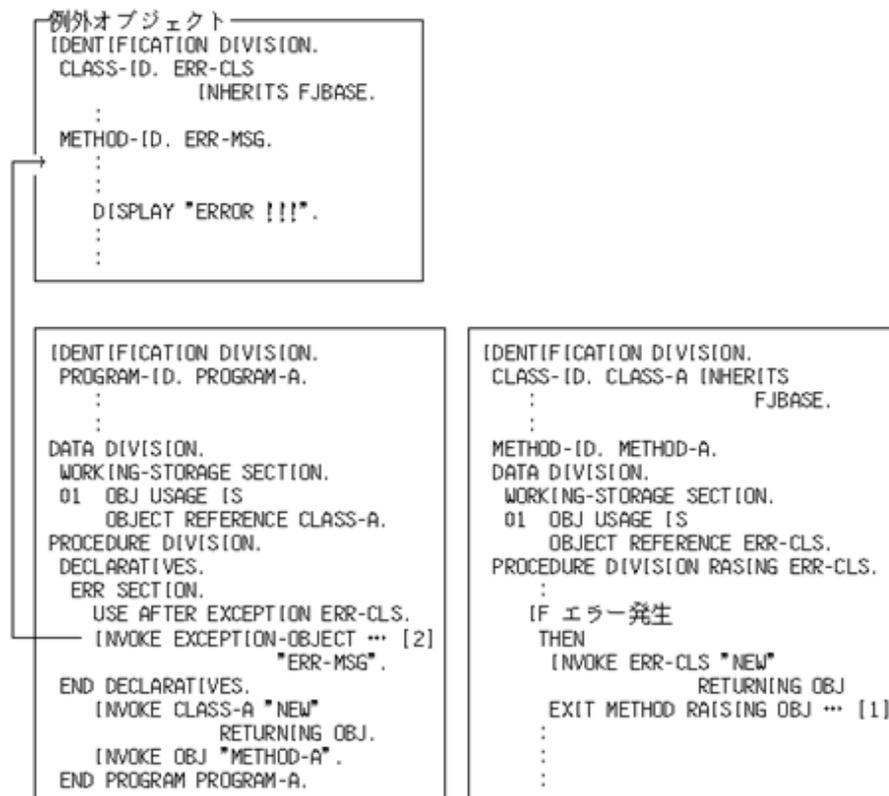
RAISE文は、手続きの途中で例外条件を発生させ、その手続きが属するプログラムまたはメソッドに記述された例外手続きを実行します。



[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。

### 17.3.1.4 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。

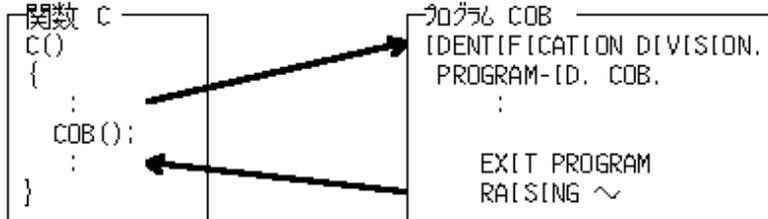
## 注意

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。

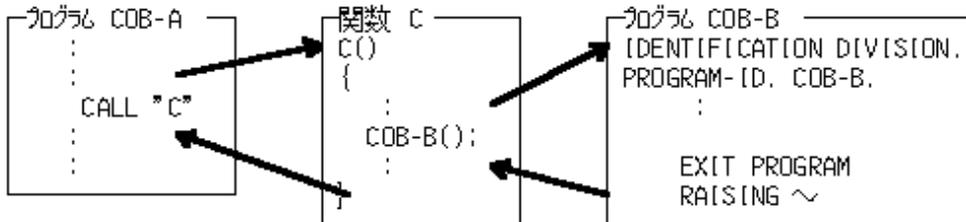
- ・ 主プログラム
- ・ 他言語プログラムから呼び出されたCOBOLプログラム(下図[1])

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。



## 17.3.2 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。

### 17.3.2.1 動的プログラム構造の特徴

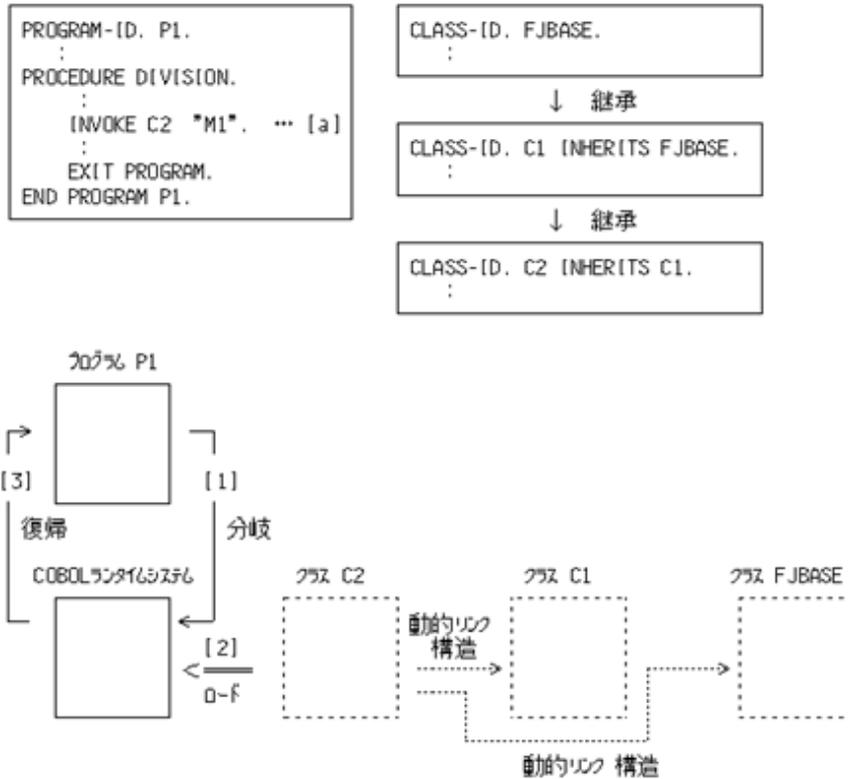
動的プログラム構造では、クラスおよびメソッドの仮想記憶上へのローディングは、実際に参照または呼び出されたときに、COBOLランタイムシステムによって行われます。

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。

### 17.3.2.2 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。

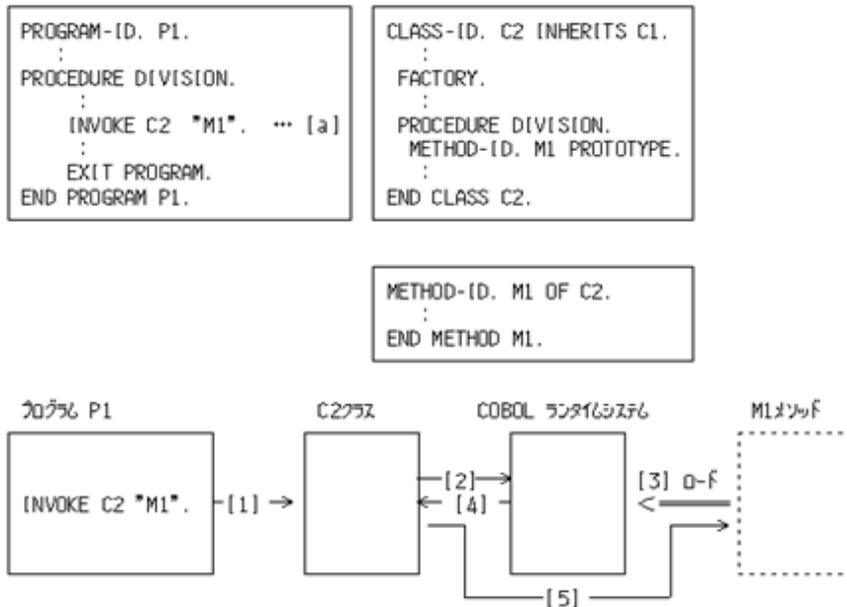
### 注意

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。

### 17.3.2.3 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。

### 17.3.2.4 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。

#### 17.3.2.4.1 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。

##### 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.3.12 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。

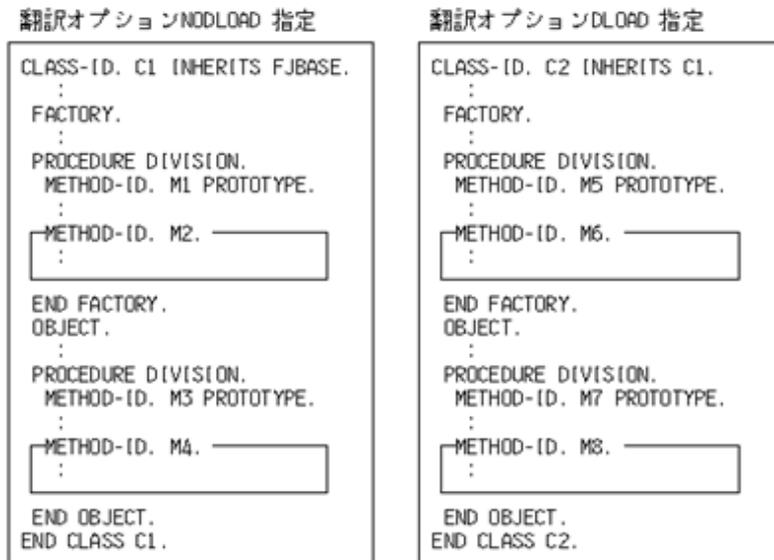
プロパティメソッドを動的プログラム構造にするには、利用者がプロパティメソッドを原型定義し、そのメソッドの原型定義を含むクラスの翻訳時に、翻訳オプションDLOADを指定します。[参照]“[A.3.12 DLOAD \(プログラム構造の指定\)](#)”



#### 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。



#### 17.3.2.4.2 リンクの方法

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。

##### 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。

#### 17.3.2.4.3 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。

#### 17.3.2.4.4 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となる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、“[C.2.26 @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、“[5.6 エントリ情報](#)”を参照してください。

## 注意

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

クラス名には使用するクラスのクラス名を指定し、DLLファイル名には使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、“[5.6 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、“[5.6 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.3.2 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17.4 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17.4 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 ・ 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、“17.1.4 適合”、“A.3.5 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。

## 17.3.3 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。

### 17.3.3.1 概要

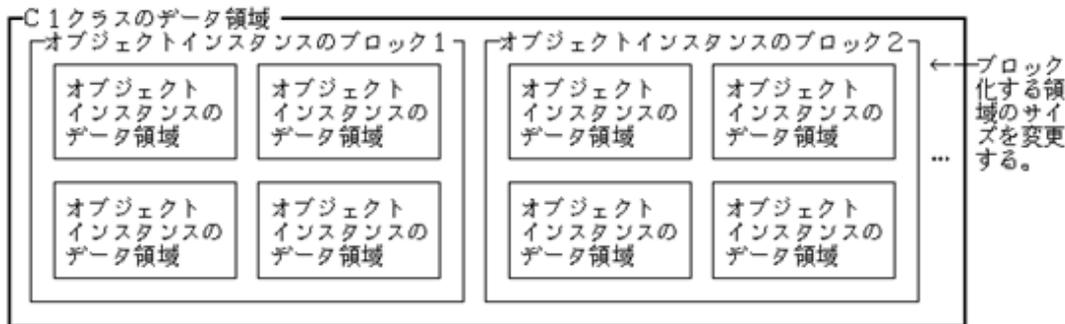
オブジェクト指向プログラムでは、NEWメソッドを実行することにより、オブジェクトインスタンスが生成されます。このとき、オブジェクトインスタンスのデータ部に記述したデータ項目およびオブジェクトインスタンスを動作させるために必要な作業域などが、オブジェクトインスタンスのデータ領域として獲得されます。

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。



### 17.3.3.2 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となります。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、“17.3.3.4 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
```

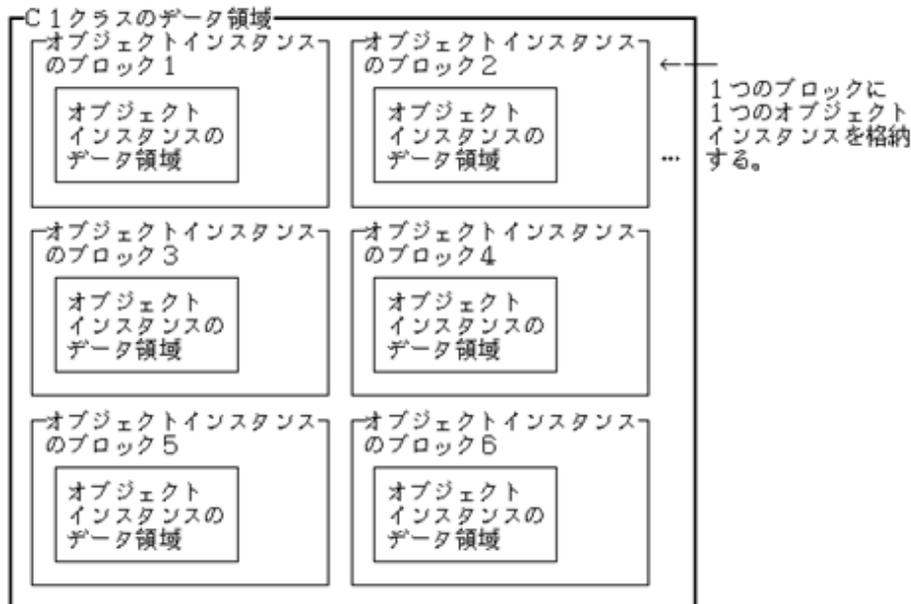
#### アプリケーションの実行単位に対する指定(環境変数情報)

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。



### 17.3.3.3 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、“[17.3.3.4 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。

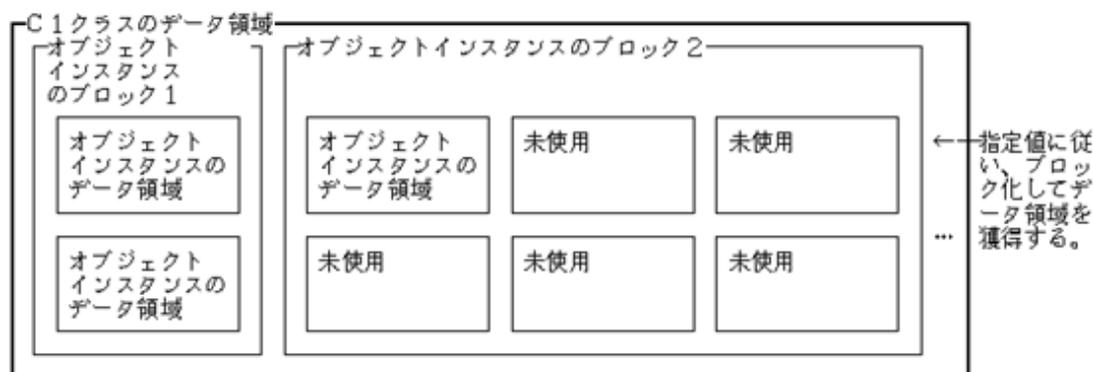
#### クラスごとの指定(クラス情報)

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る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。

### 17.3.3.4 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は、以下があります。

- “[C.2.6 @CBR\\_ClassInfFile](#) (クラス情報ファイルの指定)”
- “[C.2.34 @CBR\\_InstanceBlock](#) (オブジェクトインスタンスの獲得方法の指定)”

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。

#### 17.3.3.4.1 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

“[17.3.3.4.2 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、“[C.2.6 @CBR\\_ClassInfFile](#) (クラス情報ファイルの指定)”を参照してください。

#### 17.3.3.4.2 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] “[C.2.34 @CBR\\_InstanceBlock](#) (オブジェクトインスタンスの獲得方法の指定)”

## 17.3.4 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。

### 17.3.4.1 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。

### 17.3.4.2 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。

### 17.3.4.3 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。

### 17.3.4.3.1 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17.5 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17.5 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

### 17.3.4.3.2 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。

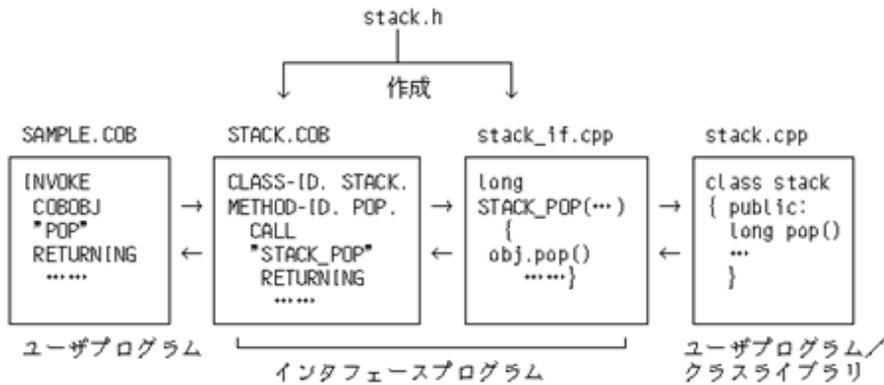
#### Visual C++側

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17.11 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17.11 インタフェースプログラムのイメージ



### 17.3.4.3.3 インタフェースプログラムの仕組み

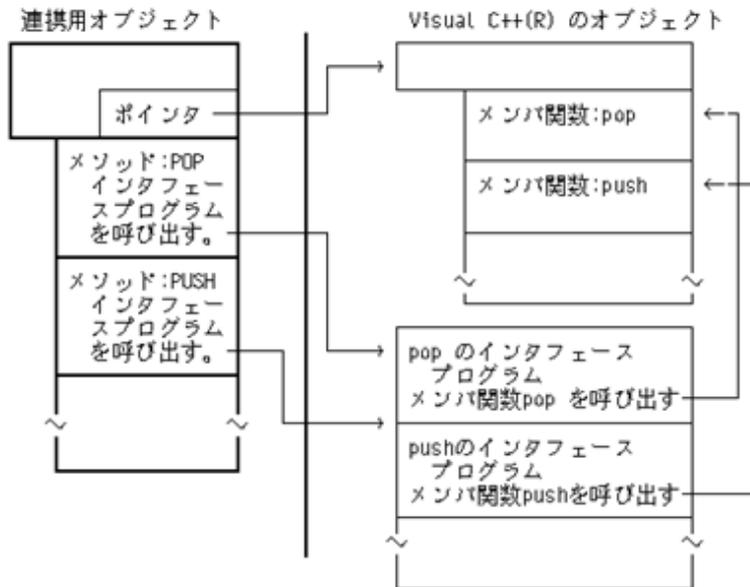
Visual C++のオブジェクトを操作する仕組みは以下ようになります。

#### 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。

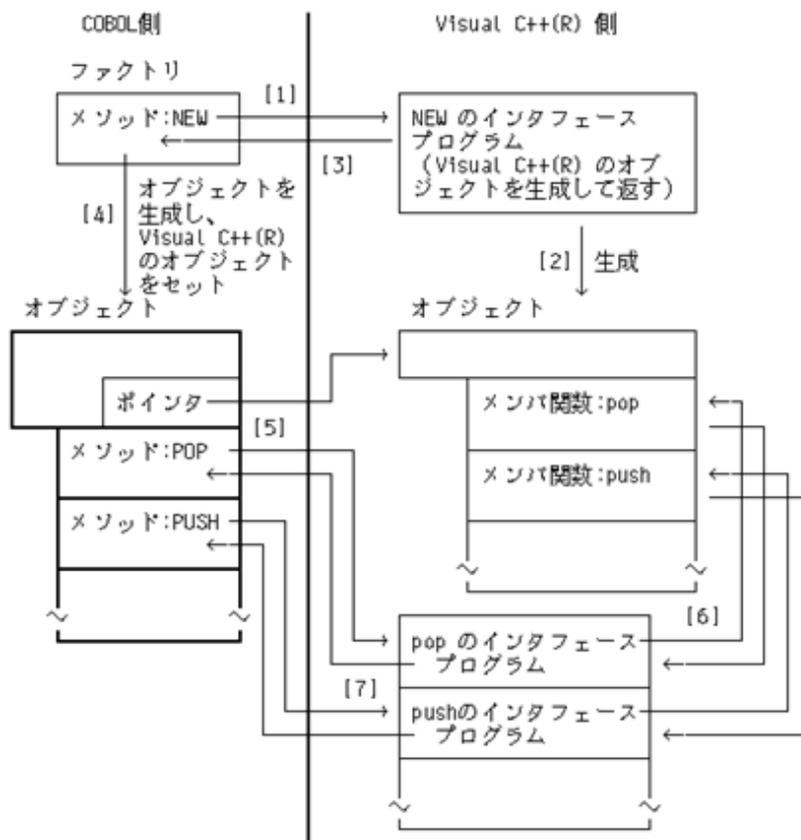
図17.12 インタフェースプログラムの構造



### 実行時の動き

実行時の制御の流れを“[図17.13 実行時の制御の流れ](#)”に示します。

図17.13 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])。

#### 17.3.4.4 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];  
};
```

##### 17.3.4.4.1 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

##### 17.3.4.4.2 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を参照します。

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。

### 17.3.4.4.3 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17.6 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 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	

### 17.3.4.5 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。



```

*>
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;
}
```

### 17.3.5 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。

#### 17.3.5.1 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。

#### 17.3.5.2 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17.14 永続オブジェクトの流れ](#)”)。

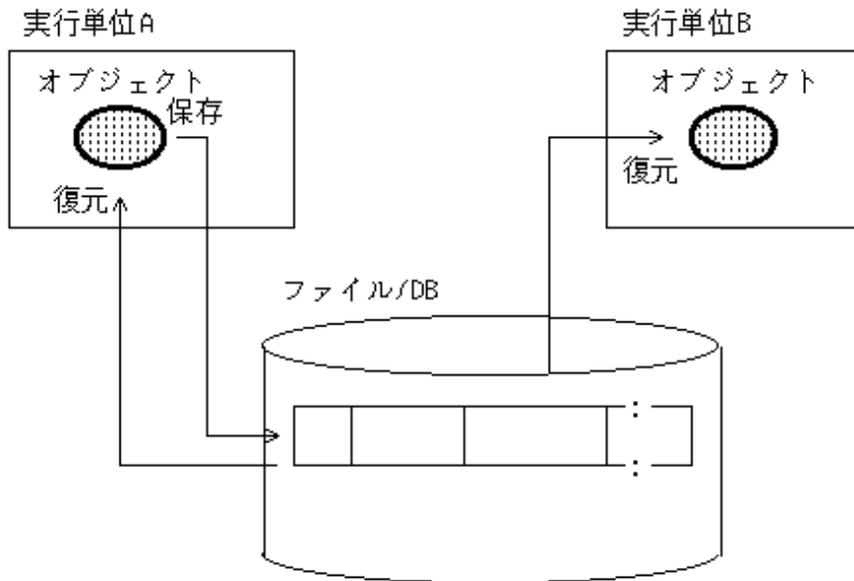
オブジェクトには適当な識別子を付け、その識別子を元に保存/復元を行います。



#### 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17.14 永続オブジェクトの流れ



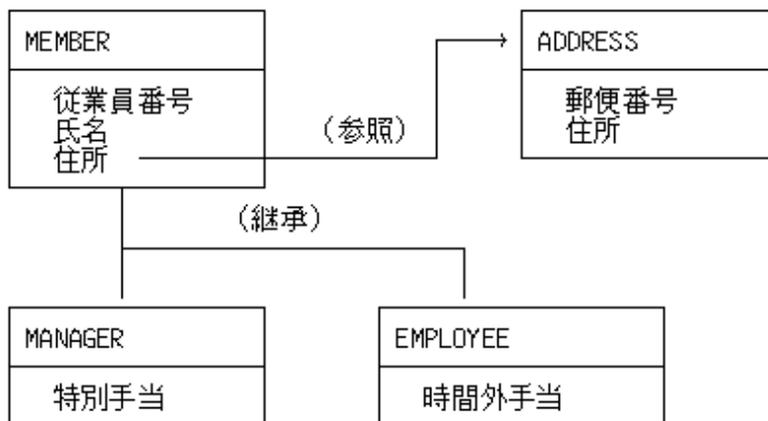
ここでは、索引ファイルを利用してオブジェクトの永続化を行う方法について説明します。

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]“15.2.6 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”

### 17.3.5.3 クラス構造の例

本節の説明で使用するクラス構造は、“図17.15 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17.15 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。

### 17.3.5.4 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。

#### 17.3.5.4.1 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

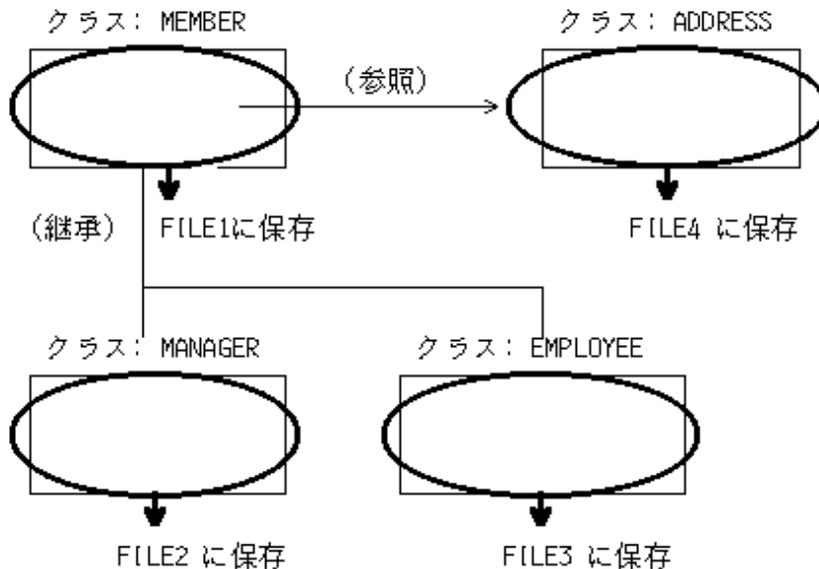
オブジェクトと保存ファイルの対応の基本は、クラスとファイルを一対一に対応付けることです。この例では、MEMBERをFILE1に、MANAGERをFILE2に、EMPLOYEEをFILE3に、ADDRESSをFILE4に保存することになります([参照]“[図17.16 クラスごとに異なるファイルに保存する場合](#)”)

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17.16 クラスごとに異なるファイルに保存する場合



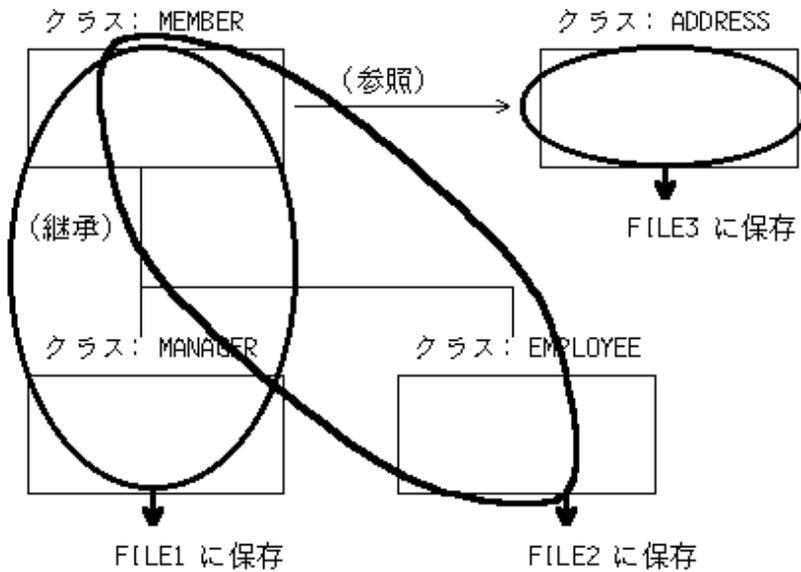
#### 親クラスのオブジェクトデータも含めて1つのファイルに保存する方法

MEMBERクラスが純粹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17.17 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17.17 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。

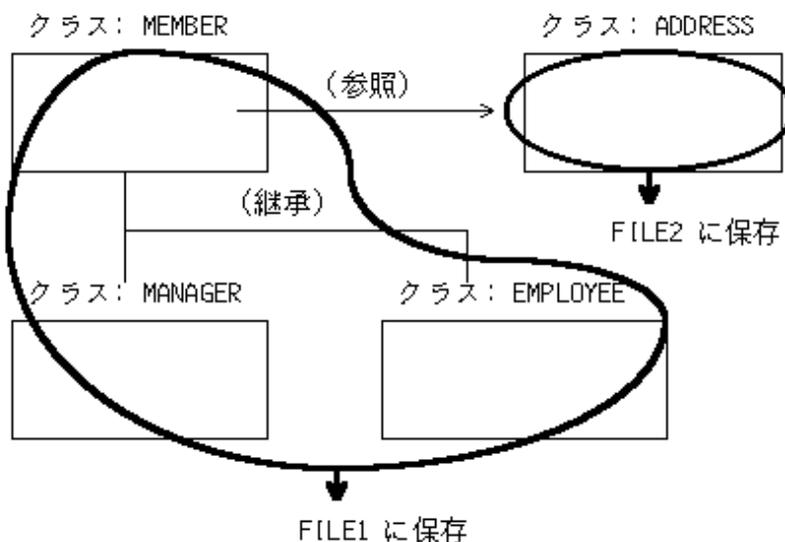
図17.17 親クラスのオブジェクトデータも含めて1つのファイルに保存する方法



### 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17.17 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17.18 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17.18 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。

#### 17.3.5.4.2 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17.19 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。

#### 17.3.5.5 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。

##### 17.3.5.5.1 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。

### 17.3.5.5.2 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

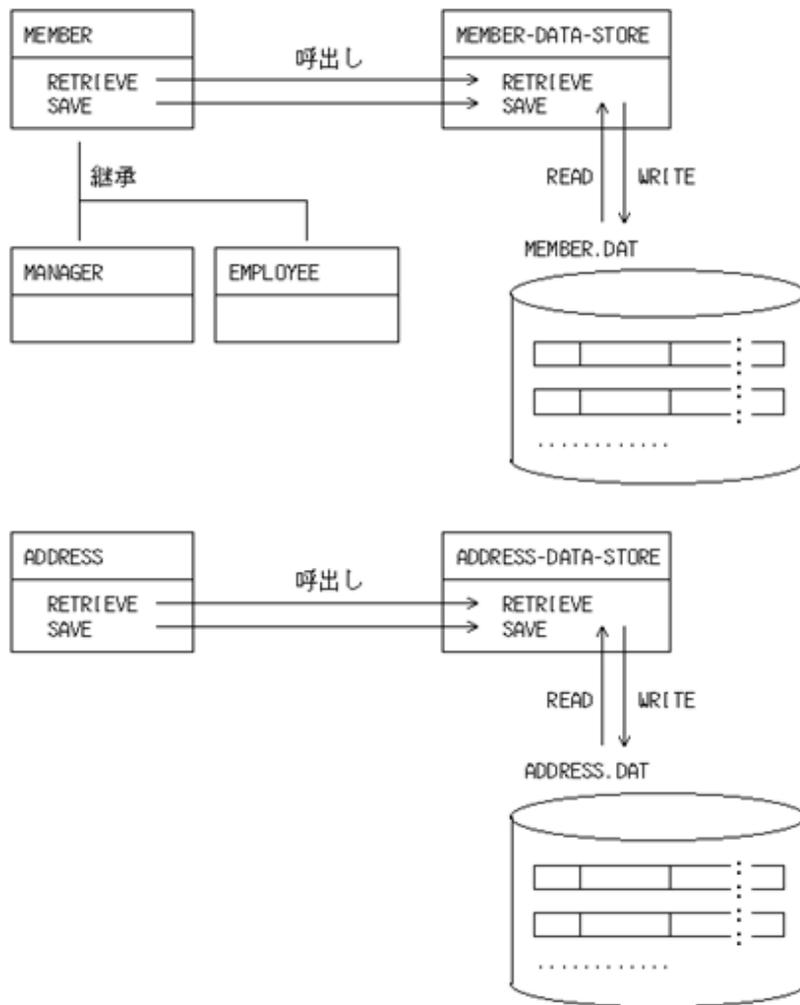
SAVEメソッドはオブジェクトを保存するメソッドです。実際には、自分自身を引数として、対応する索引ファイル操作クラスのSAVEメソッドを呼び出します。SAVEメソッドは、親クラスで定義すれば十分で、個々のクラスに定義する必要はありません。

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17.7 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17.20 索引ファイル操作クラスとの関係](#)”に示します。

表17.7 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17.20 索引ファイル操作クラスとの関係



### 17.3.5.5.3 処理の流れ

保存/復元の処理の流れを“[図17.20 索引ファイル操作クラスとの関係](#)”に従って説明します。

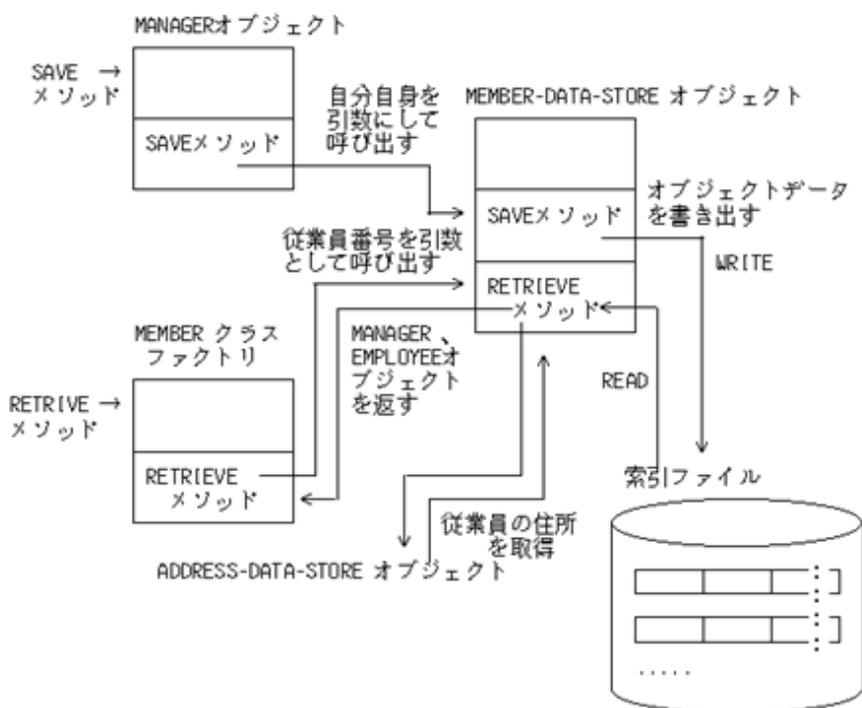
#### 保存

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(AADDRESSオブジェクトの復元については“[図17.21 保存/復元の処理の流れ](#)”では省略)。

図17.21 保存/復元の処理の流れ



## 17.3.6 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28章 COM機能](#)”を参照してください。

### 17.3.6.1 \*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。

#### 17.3.6.1.1 概要

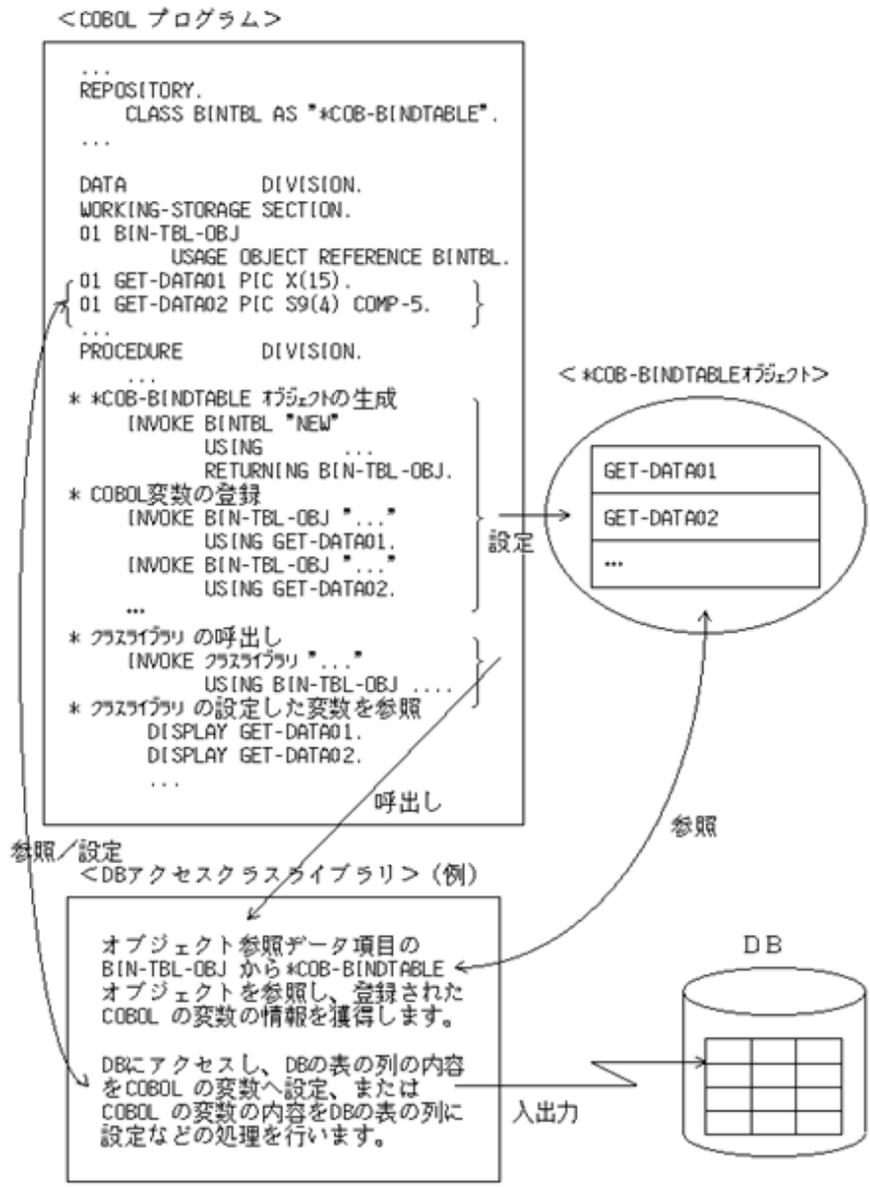
NetCOBOLのアプリケーションの作成のために、業務用の各種クラスライブラリが提供されます。各種クラスライブラリを利用することで、データベースのアクセスなどの業務用アプリケーションを簡単に作成することができます。

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。

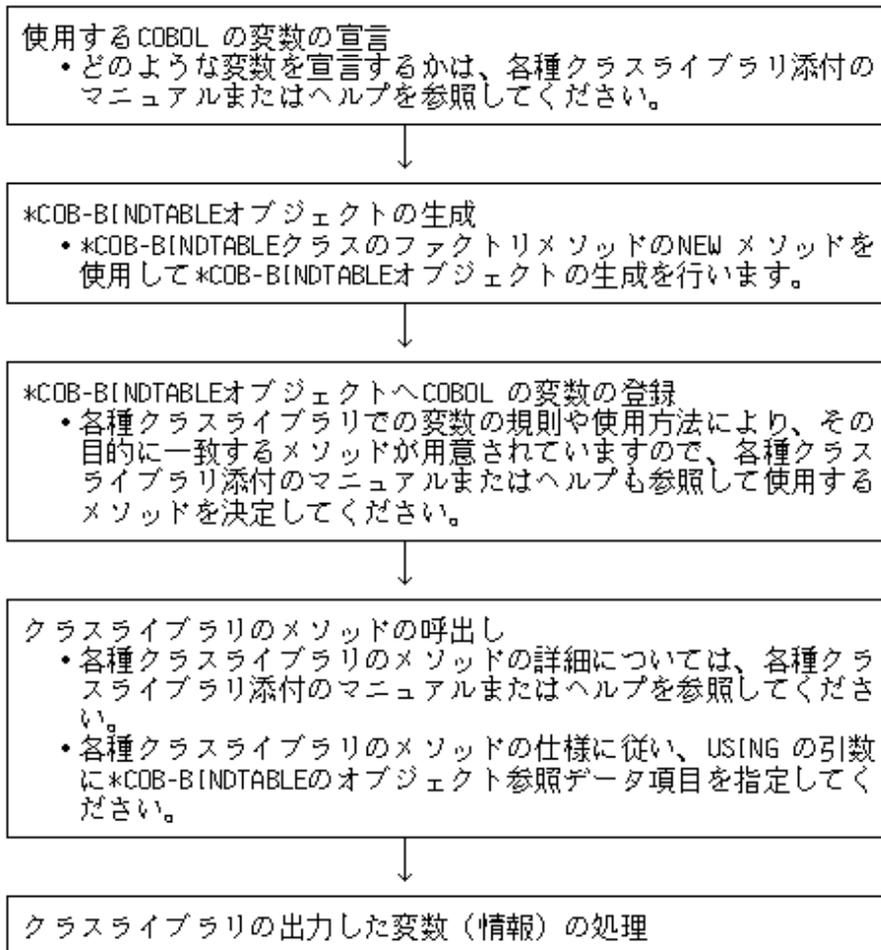
たとえば、データベースアクセスのためのクラスライブラリのデータ参照のメソッドを呼び出す場合、データベースから取り出したデータを受け取るためのCOBOL変数をアプリケーションから通知する必要があります。

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。



### 17.3.6.1.2 \*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。

### 17.3.6.1.3 \*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 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。 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 作成オブジェクト	USAGE OBJECT 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.
```

### 17.3.6.1.4 \*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 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。
	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 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。 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 標識変数	PIC S9(4) BINARY   COMP-5	
	RETURNING なし	—	

### 使用規則

1. 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。
2. 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。
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 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 可変長データ長さ部	PIC S9(4   9) BINARY   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。

メソッド名	引数	引数の型	意味
	USING 第2 可変長データデータ部	PIC X(p)   PIC N(q)	
	RETURNING 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.
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 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" USING GET-DATA1-LNG
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 可変長データ長さ部	PIC S9(4   9) BINARY   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。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 可変長データデータ部	PIC X(p)   PIC N(q)	
	USING 第3 標識変数	PIC S9(4) BINARY   COMP-5	

メソッド名	引数	引数の型	意味
	RETURNING 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 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 バイナリデータ長さ部	PIC 9(9) BINARY   COMP-5	
	USING 第3 バイナリデータデータ部	PIC X(p)   PIC N(q)	
	RETURNING 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 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 バイナリデータ長さ部	PIC 9(9) BINARY   COMP-5	
	USING 第3 バイナリデータデータ部	PIC X(p)   PIC N(q)	
	USING 第4 標識変数	PIC S9(4) BINARY   COMP-5	
	RETURNING 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.
```

## 17.3.7 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。

### 17.3.7.1 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すると、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るれば、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  "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。

### 17.3.7.2 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長さに決まる。
01 パスワード PIC X ANY LENGTH. |
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。

## 17.4 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。

### 17.4.1 クラス定義で使用できない機能

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で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。

機能	説明
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。
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 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。

## 17.4.2 分離翻訳されるメソッド定義で使用できない機能

分離翻訳されるメソッド定義中で使用できない機能を、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用できない機能を説明しています。“17.4.1 クラス定義で使用できない機能”に示した機能のうち、メソッド定義内で使用できない機能は、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	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。 <ul style="list-style-type: none"> <li>• ASSIGN句にPRINTER、PRINTER-n (nは1～9までの整数)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>

## 第4部 サーバサイドアプリケーションの開発と運用

---

---

第18章 サーバ・タイプのアプリケーション.....	486
第19章 Web連携.....	491
第20章 マルチスレッド.....	532

## 第18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。

### 18.1 バックグラウンド処理

#### 18.1.1 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
  - － 表示ファイル機能(画面入出力機能)
  - － スクリーン操作機能
  - － 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
  - － 実行時メッセージ
  - － NetCOBOLのアイコン
  - － 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。

- － 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、“[C.2.65 @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。
- － 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、“[C.2.37 @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、“[C.2.67 @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、“[C.2.73 @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制する方法

環境変数情報@WinCloseMsg=OFFを指定します。詳細は、“[C.2.77 @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制したい場合に指定します。

## 18.1.2 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、アイコンの状態から起動した場合は、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。

## 18.1.3 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、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プログラムに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、“[8.1.15 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、“[15.2.8.4 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、“[18.1.1 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、以下の方法を使用します。

- ・ プログラムから対話型デバッグを起動する方法  
詳細は、“[24.4.1.2 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](#)”を参照してください。
- ・ NetCOBOL Studioのアタッチデバッグ機能を使用する方法  
詳細は、“[NetCOBOL Studio ユーザーズガイド](#)”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。

## 18.2 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)。

#### MeFt/Webの利用

Windows上でMeFt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。表示ファイル機能の画面入出力(画面定義体)を使用して開発した業務アプリケーションは、MeFt/Web連携することにより、再翻訳することなく、そのままイントラネット環境の基幹業務に使用することができます。

詳細は、“[第19章 Web連携](#)”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、“[18.1.3 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。

## 18.3 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/Centric Manager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。

### 18.3.1 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18.1 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、“[C.2.37 @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“メッセージ集”の“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。

### 18.3.2 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、“[I.2.9 COB\\_REPORT\\_EVENT\(イベントログ出力サブルーチン\)](#)”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、システムの定量制限に依存します。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。

項目	内容
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。
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、“[18.3.1 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19章 Web連携

## 19.1 概要

アプリケーションが動作するシステムの形態には、ホスト集中型とクライアント/サーバ型に加え、ネットワークコンピューティング型があります。

ネットワークコンピューティング型は、インターネットの急速な普及に伴い、情報公開としての役割だけではなく、業務アプリケーションの動作環境として利用されています。

ネットワークコンピューティング環境では、ネットワークを通じての業務連携およびWebを使用した情報共有による業務効率の向上が可能となります。また、プラットフォーム無依存な操作性を持つWebブラウザの利用は、多種の業務連携を実現する上で欠かせないものといえます。

一方で、すでにホスト集中型およびクライアント/サーバ型のシステムでは、COBOLは業務システムの基幹系業務の領域での中心言語として多く使用されています。

このようなインターネット/イントラネット技術を基盤に、基幹系処理に適しているCOBOLを使用した業務アプリケーションを活用したいという要件は高まっています。

### 19.1.1 インターネット/イントラネット環境の活用

インターネット/イントラネット環境を有効に活用するために、Webの特色とCOBOLを使用する利点を以下に示します。

#### Webの特色

##### 遠隔地

ネットワーク上のあらゆる資産にアクセス可能となるため、遠隔地との接続も簡単に行えます。

##### コストの削減

Webブラウザの利用により、クライアントマシンのセットアップ・メンテナンス作業を軽減できます。

##### クライアントのシステム無依存

Webブラウザを使用するため、クライアントのシステムに依存しない業務システムを構築できます。モバイルコンピューティング環境でも利用可能です。

##### サーバの負荷分散

ネットワークで接続可能なサーバであれば、各種の業務システムを分散させることができます。

##### セキュリティ

Webの特色として、クライアントから容易に接続することが可能な反面、セキュリティの問題があります。システムを構築する場合には、Webサーバでクライアントが接続するドメインに対するアクセス制限を設ける、アプリケーションでユーザ制限を行う、といった自己防衛手段を講じる必要があります。

#### COBOLを使用する利点

通常、Web環境で動作するアプリケーションの開発言語を選択する場合、Webブラウザとのデータ入出力を可能とするために、以下の特徴を考慮します。

- ・ データ操作
- ・ ファイルやデータベースへのアクセス
- ・ 環境変数操作

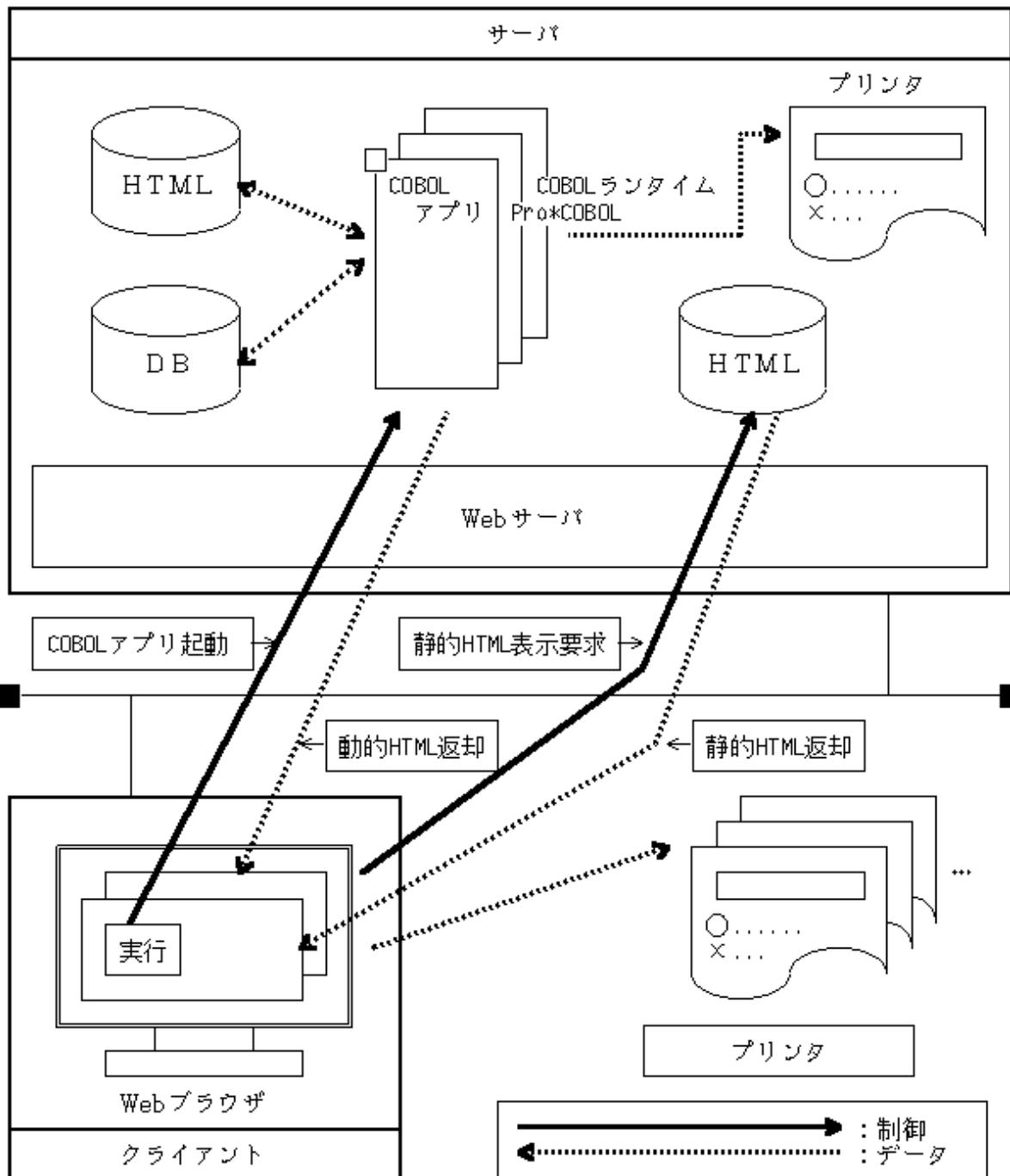
COBOLは、上記の特徴を満たすだけでなく、さらに以下の点からも効率のよい開発を行うことができます。

- ・ 今まで培った知識やシステム構築ノウハウの活用  
COBOLを使用したアプリケーションの開発により培われてきたノウハウを活用できるため、開発のスピードアップを図ることができます。
- ・ 既存資産の有効活用  
従来の環境での業務システムで使用していた資産を有効活用できます。

## 19.1.2 Webアプリケーション

HTML文書単独では静的な情報しか返せないため、更新される情報をリアルタイムに伝えられません。これに対し、CGIや各種Webサーバ固有のAPIを、Webブラウザから呼び出されるアプリケーションで利用し、Webブラウザから入力した情報に対して、更新される情報を動的に返却することにより、リアルタイムに情報を提供することができます。また、情報の返却だけでなく、業務アプリケーションには欠かせないファイル入出力やデータベースアクセスなどのデータ処理手続きをWebブラウザから依頼(実行)できます。

COBOLは、データ処理を中心とする事務処理言語であり、この特性を最大限に利用したインタラクティブな動的HTMLデータの作成が可能となります。



なお、Webアプリケーションは、Webサーバ上でサービス配下のアプリケーションとして動作します。アプリケーション作成および動作の留意点については、“18.1.3 サービス配下で動作するプログラム”を参照してください。

### 19.1.3 Web連携の利用方法

COBOLでインターネット/イントラネット環境の業務システムを構築するには、以下の方法があります。

#### MeFt/Webの利用

表示ファイル機能の画面入出力(画面定義体)を使用して開発した業務アプリケーションは、MeFt/Webと連携することにより、再翻訳することなく、そのままイントラネット環境の基幹業務に使用することができます。

#### COBOL Webサブルーチンを利用したアプリケーションの開発

Webブラウザとのデータ入出力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することなく、COBOLの知識だけでアプリケーションを作成できます。

### 19.1.4 各Web連携機能の特徴

COBOL Web連携の各機能における特徴を以下に示します。

特徴機能	連携機能		
	ISAPIサブルーチン	CGIサブルーチン	MeFt/Web
動作可能Webサーバ	IIS	CGIをサポートする全サーバ	IIS
Web利用技術	ISAPI	CGI	ActiveX技術
Webサーバが起動するファイルの形式	DLL	EXE	DLL/EXE
動作モデル	スレッド	プロセス	スレッド/プロセス
Webサーバ上で実行されるCOBOLアプリケーションのローディング機構	該当アプリに対する初回リクエスト時にロードされ、以後、次回リクエストに備えメモリ内に常駐。	該当アプリに対するリクエストごとにプロセスを起動し、処理終了後そのプロセスを破棄。	該当アプリケーションに対する初回リクエスト時にメモリにロードされ、表示ファイルによる一連の画面対話が行われている間(OPEN～CLOSE)メモリ内に常駐。
実行性能	マルチスレッド動作、DLLのプレロード機能等の仕組みにより、高速。	プロセス起動・消滅を毎リクエストごとに行うオーバーヘッドのため低速。	マルチスレッド動作により高速。
ユーザ/アプリケーション・インタフェース	HTML Webサブルーチン		表示ファイル
業務アプリケーション作成の容易性	Webサーバとのインタフェースは、COBOL言語仕様に則したWebサブルーチン内に隠蔽されているため、Webに関する特別難しい専門知識は不要。		基本的には、表示ファイルおよびMeFt/Webの知識があれば良い。
既存資産の活用性	既存資産のビジネスロジックは、ほとんどをそのまま有効活用することが可能。ただし、Webサーバとのインタフェースが必要となるため、Webサブルーチンを使用するための記述を追加し、再翻訳する必要がある。		プロセスベースのアプリであれば、ソース修正および再翻訳・再リンクとも不要。 スレッドベースであれば、再翻訳・再リンクが必要。
帳票印刷	サーバ側はアプリケーションによるきめ細かい帳票印刷が可能。ただし、クライアント側はブラウザの印刷機能またはそれに代わるダウンロードアプリケーションの印刷機能を使用。		サーバ、クライアントともアプリによるMeFtを使用した帳票印刷が可能。また、クライアント側で印刷プレビュー機能も利用可能。
利点	マルチスレッドで、高性能かつ資源消費の少ないアプリケーションを作成可能。	サーバに依存しない、独立性の高いアプリケーションを作成可能。	既存の表示ファイル機能を使用したアプリケーションを活用し、開発

特徴機能	連携機能		
	ISAPIサブルーチン	CGIサブルーチン	MeFt/Web
			期間を短縮。マルチスレッド化も容易。

## 19.1.5 各Web連携機能の詳細情報

ご使用になりたいCOBOL Web連携機能が決定したら、実際に業務アプリケーションの設計・開発を行います。

各Web連携機能の詳細は、以下を参照してください。

### COBOL ISAPIサブルーチンを使用する場合

COBOL ISAPIサブルーチンの詳細は、“[19.4 CGIサブルーチンを使用したアプリケーションの作成と実行](#)”を参照してください。

### COBOL CGIサブルーチンを使用する場合

COBOL CGIサブルーチンの詳細は、“[19.4 CGIサブルーチンを使用したアプリケーションの作成と実行](#)”を参照してください。

すでにCOBOL CGIサブルーチンを使用して作成されたアプリケーションを、COBOL ISAPIサブルーチンを使用したアプリケーションに移行する場合

CGIからISAPIへの移行については、“[0.4 CGIからISAPIへの移行の手引き](#)”を参照してください。

### MeFt/Webを使用する場合

表示ファイル機能の詳細は、“[9.2 表示ファイル\(画面入出力\)の使い方](#)”、“[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。

MeFt/Webの詳細は、“[MeFt/Web ユーザーズガイド](#)”を参照してください。

## 19.2 MeFt/Webを利用したアプリケーション

表示ファイル機能の画面入出力(画面定義体)を使用して開発した業務アプリケーションは、MeFt/Webと連携することにより、再翻訳することなく、そのままイントラネット環境の基幹業務に使用することができます。

### 19.2.1 MeFt/Web利用の目的

Web環境では、データ入力のためにHTMLを使用しますが、HTMLはデータタイプの不足など、業務システムでの入力処理に使用するには、機能が不足する部分もあります。MeFt/Webを使用すると、MeFtの入出力機能をそのまま利用することができるため、これらの機能不足を補うことが可能です。

また、以下のような利点も備えています。

#### 既存資産の活用とノウハウの継承

既存の表示ファイル(MeFt接続)を使用しているアプリケーションは、そのままWebサーバ上で動作します。

#### 資源の一元管理

Webブラウザを使用するため、クライアントアプリケーションの開発は不要になります。また、アプリケーションが使用する画面帳票定義体およびクライアント側で動作するMeFtもサーバ側に置くので、サーバでの資源の一元管理が可能です。

#### 帳票印刷機能の利用

サーバ、クライアントのどちらにでも帳票印刷ができます。また、従来の帳票印刷の方法に加えて、帳票のスプールおよびプレビューが可能になります。

#### Webブラウザ機能の活用

従来の画面機能に加えて、最小限の手直しで画面の3D化およびハイパーリンク機能を画面に追加することができます。

### 19.2.2 MeFt/Webの利用方法

アプリケーションは表示ファイル機能を利用するため、開発には従来のCOBOL開発環境を利用できます。表示ファイル機能としては、動作環境に依存せず、従来のMeFtを使用したアプリケーションと同様に開発できます。

詳細は、“[MeFt/Web ユーザーズガイド](#)”を参照してください。

## 19.3 COBOL Webサブルーチンを利用したアプリケーションの概要

通常、Webアプリケーションを開発/メンテナンスするには、Webブラウザとの間のインタフェースに関する知識が必要となります。しかし、COBOL Webサブルーチンを利用すれば、COBOLの知識だけでCOBOLプログラムとWebブラウザとの間でデータの入出力を行うことができます。

### 19.3.1 Webアプリケーションが利用できるインタフェースの種類

以下に示すように、Webサーバのアプリケーションインタフェースは多数存在します。

#### ISAPI(Internet Server API)

ISAPIは、Microsoftにより提唱されたWebサーバ上で動作する対話型アプリケーションのための拡張されたプログラミング・インタフェースです。ISAPIの実行コードは、IISなどのISAPI準拠のWebサーバに搭載されています。

COBOLでは、ISAPIおよびその実行コードをCOBOLの言語仕様に合わせてカプセル化し、IIS上で動作するISAPIアプリケーション(一般的には、ISAと呼ばれます)をより簡単に開発するための機能として、ISAPIサブルーチンを提供しています。

また、ISAPIは、Webサーバ上で動作する対話型アプリケーションのプログラミング・インタフェースの点ではCGIとよく似ていますが、CPUやメモリなどの使用資源量およびパフォーマンスの面で相違があります。

ISAPIとCGIの相違点については、“19.3.3 CGIとISAPIの相違点”を参照してください。

#### CGI(Common Gateway Interface)

CGIは、Webブラウザとのデータ入出力を行う最も標準的なインタフェースです。また、ほとんどのWebサーバはこのCGIをサポートしています。

COBOLでは、CGIを利用するアプリケーションをより簡単に開発するための機能として、COBOL CGIサブルーチンを提供しています。

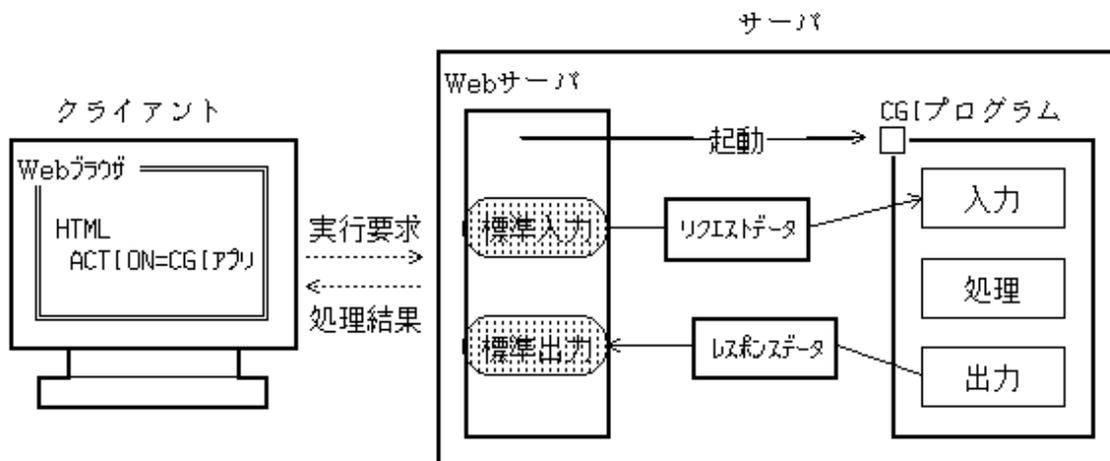
### 19.3.2 Webアプリケーションの処理の流れ

CGIを例に、WebブラウザとWebアプリケーションの処理の流れを説明します。

CGIは、Webブラウザとデータの入出力を行うために利用します。CGIとは、Webブラウザ(クライアント)からWebサーバ上のCGIプログラム(CGIスクリプトと呼ばれる)を動作させるための最も基本的なインタフェースであり、CGIプログラムの起動と標準入出力(環境変数経由によるデータ入力を含む)によるデータの入出力を提供しています。

CGIは、「データ入力(リクエスト)→処理→処理結果出力(レスポンス)」の流れで動作し、この流れを実現するのがCGIプログラムです。

ISAPIや他の各種APIも処理の流れは基本的に同じですが、CGIではデータの入出力が標準入出力を介して行われるのに対して、ISAPIや他の各種APIではデータ入出力専用のAPIが用意(APIの中に隠蔽)されています。この違いはCOBOL Webサブルーチンによって吸収されているため、COBOL Webサブルーチンを使用している限りCOBOLアプリケーション側で、このような違いを意識する必要はありません。



### 19.3.3 CGIとISAPIの相違点

CGIアプリケーションは、実行可能なモジュール形式で作成します。

Webサーバは、クライアント(Webブラウザ)からのリクエストごとに、CGIアプリケーションの実行プロセスを個別に生成します。したがって、サーバのCPUやメモリなどの資源を大量に消費します。

これに対し、ISAPIアプリケーション(ISA)は、ダイナミックリンクライブラリ形式(DLL)で作成します。

DLLは、Webサーバ自身のプロセスにロードされます。一度ロードされたDLLは、メモリ中に常駐したり、一定時間を経過するとアンロードされたりします。

CGIと他のアプリケーションインタフェースとの大きな違いは、以下の点です。

- CGIは、リクエストごとに個別のプロセスで実行される
- 他のアプリケーションインタフェースは、リクエストごとにWebサーバ自身のアドレス空間にスレッドを生成し、マルチスレッド環境で実行される

このため、CGIにおけるプロセスの生成/消滅、メモリ使用量、タスク切り替えなどのオーバーヘッドが解消され、パフォーマンスに優れたWebアプリケーションを構築することができます。

図19.1 ISAPIを使用した場合

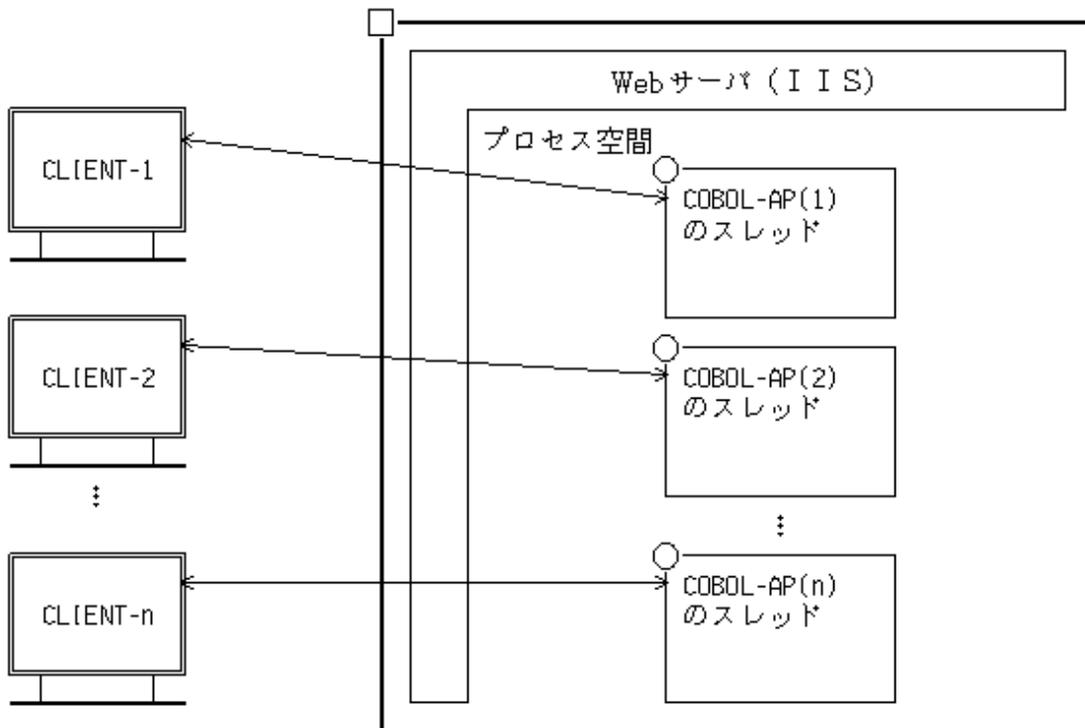
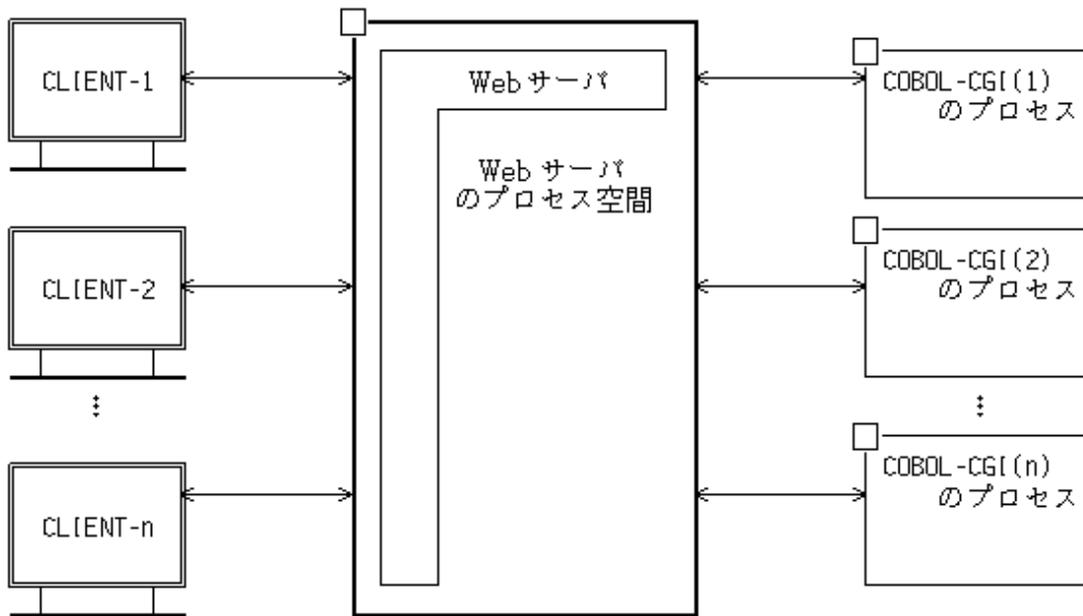


図19.2 CGIを使用した場合



### 19.3.4 COBOL Webサブルーチンの特徴

COBOL Webサブルーチンは、COBOLを使用してWebアプリケーションを作成するためのインタフェースと実行環境を提供します。CGIやISAPIなどの各種インタフェースを用いて構成されていますが、利用者はその違いをほとんど意識しないで利用できるため、PerlやC言語よりも簡単にWebアプリケーションを作成できます。

また、COBOL Webサブルーチンには次の特徴もあります。

- CALLインタフェースで実行できるためCOBOLで簡単に利用できます。
- HTMLの一部を変換して出力する機能を提供することで、プログラムとHTML文書を独立させることができ、生産性および保守性の高い開発が可能です。
- 利用者はCOBOL Webサブルーチンの提供するインタフェースを使用してプログラムを記述することで、さまざまなWebサーバ上で動作するWebアプリケーションを作成することができます。

COBOL Webサブルーチンでは次のサービスを提供しています。

- Webサーバからアプリケーション起動時に渡されるリクエストデータ(Webパラメタ)の受取りおよびCOBOLのデータ形式への変換
- 処理結果の出力(動的HTMLの出力機能など)
- ヘッダ(Content-Typeなど)の自動生成
- システムコマンドの実行機能
- リクエスト情報の取得
- Cookieデータの受取りおよび出力
- ファイルのアップロード

## 19.3.5 COBOL Webサブルーチン

### ISAPIの利用

COBOL ISAPIサブルーチンを使用することにより、IIS上で動作するISAPIを利用したWebアプリケーションをCOBOLで作成することができます。

COBOL ISAPIサブルーチンは、COBOL CGIサブルーチンと共通インタフェースであるため、CGIからISAPIへの移行も容易に行えます(ほとんど同じ仕様ですが、一部ISAPI固有の記述を追加する必要があります)。

### CGIの利用

COBOL CGIサブルーチンを使用することにより、さまざまなWebサーバ上で動作するCGIを利用したWebアプリケーションを作成することができます。

## 19.4 CGIサブルーチンを使用したアプリケーションの作成と実行

### 19.4.1 CGIの概要

CGI(Common Gateway Interface)とは、Webブラウザ(クライアント)とWebサーバとのやりとりに関する規格です。

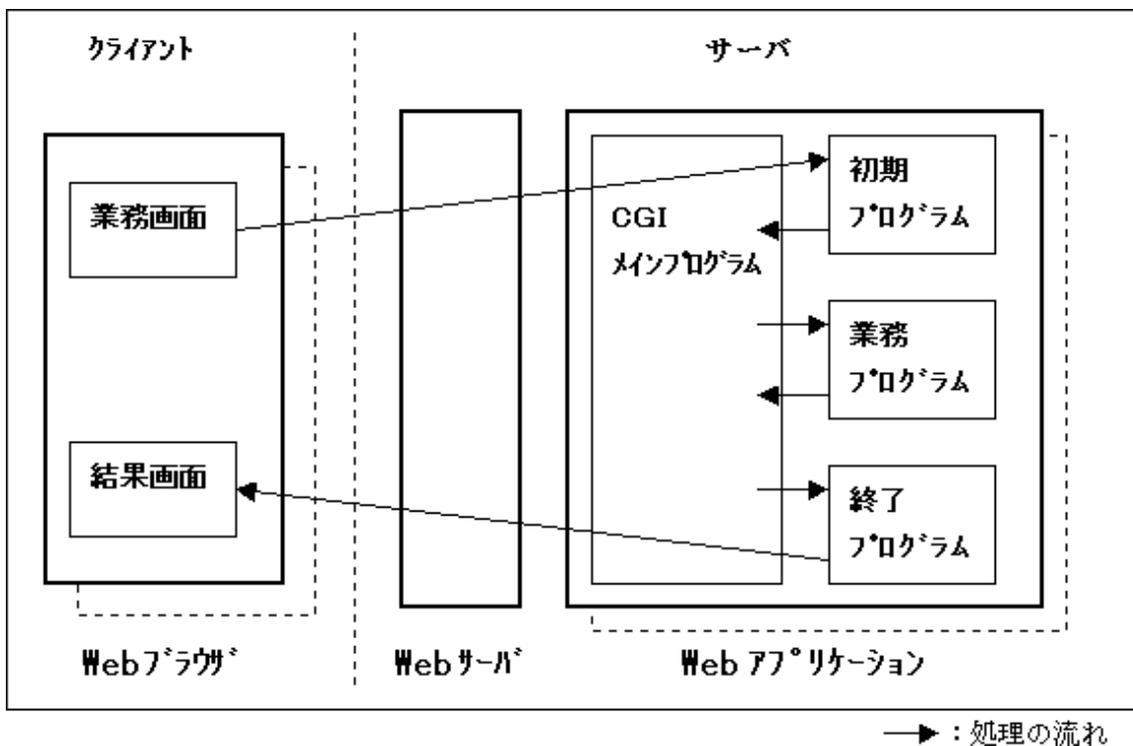
インターネット/イントラネットを利用して業務を構築する場合に、Webサーバ上のHTML文書(ホームページなど)だけでは、集計や検索などのデータ処理、ファイルアクセス、およびデータベースアクセスを実現できません。このようなサービスを実現するためには、外付けのCOBOLアプリケーションなどを利用する必要があります。

このCOBOLアプリケーションをWebアプリケーションと呼びます。

Webアプリケーションは、メインプログラム、初期プログラム、業務プログラム、および終了プログラムで構成されています。

### 19.4.2 CGIの処理の流れ

CGIを利用した場合の処理の流れを説明します。



Webブラウザ(クライアント)にURLを入力すると、あらかじめWebサーバに登録しているHTML文書の業務画面(以降、呼出し用ページと呼びます)が表示されます。

Webブラウザに表示された呼出し用ページに用意されたWebアプリケーション実行ボタンをクリックすると、Webサーバがプロセスを生成し、Webアプリケーションを起動します。また、呼出し用ページにテキスト入力項目などを用意した場合、利用者が入力した情報は、受渡しデータ(以降、Webパラメタと呼びます)としてWebサーバ側のWebアプリケーションに渡されます。

Webアプリケーションでは、以下の処理を行います。

- 初期処理を行う初期プログラムの実行
- Webパラメタの受け取り
- 手続きの実行
- 処理結果をHTML文書(以降、結果出力用ページと呼びます)としてWebブラウザに出力する業務プログラムの実行
- 終了処理を行う終了プログラムの実行

Webアプリケーションの処理が終了すると、プロセスも終了します。

### 19.4.3 COBOL CGIサブルーチンとは

---

COBOL CGIサブルーチン(以降、CGIサブルーチンと呼びます)は、CGIを使用したWebサブルーチンです。このサブルーチンを使用することで、WebアプリケーションをCOBOLによって簡単に作成することができます。

CGIを使用したWebアプリケーションは、Webサーバの種類を特定せずに動作させることができます。また、CGIサブルーチンはWebサブルーチンの共通機能で構成されているので、CGIを使用したWebアプリケーションは、わずかな修正で他のサブルーチンを使用したWebアプリケーションに移行することができます。

### 19.4.4 必要なもの

---

CGIサブルーチンを使用したWebアプリケーションの作成と実行に必要なものは以下のとおりです。

- Webサーバ  
Webアプリケーションを動作させるために必要です。
- NetCOBOL 開発パッケージ  
Webアプリケーションの翻訳、リンクおよび実行の際に必要です。なお、運用時にはNetCOBOL サーバ運用パッケージが必要です。



Webサーバは、Webアプリケーションを動作させるために実行権を設定する必要があります。インストールや設定方法については、運用するWebサーバのシステム管理者に相談してください。

### 19.4.5 作成するもの

---

CGIサブルーチンを使用してWebアプリケーションを作成するには、以下が必要です。

- CGIサブルーチンを使用したCOBOLプログラム
- HTML文書
  - － 呼出し用ページ(Webアプリケーション起動用)
  - － 結果出力用ページ(処理結果返却用、必要に応じて作成する)

#### 19.4.5.1 CGIサブルーチンを使用したCOBOLプログラム

COBOLプログラムを作成します。

なお、ここで作成するCOBOLプログラムは、画面入出力機能以外のCOBOLの機能をすべて利用できます。

プログラム作成上の注意については、“[19.4.6 作成する際の注意点](#)”を参照してください。

### 19.4.5.1.1 メインプログラム

メインプログラムは、各プログラムを呼び出すためのプログラムです。メインプログラムでは、それぞれのタイミングで初期プログラム、業務プログラムおよび終了プログラムを呼び出します。



例

#### メインプログラムのコーディング例

```
000010*-----*
000020*      CGI メインプログラム      *
000030*-----*
000040 IDENTIFICATION DIVISION.
000050 PROGRAM-ID.      CGI.
000060 DATA              DIVISION.
000070 PROCEDURE        DIVISION.
000080 MAIN-PROC.
000090     CALL "INIT".           *> 初期プログラム
000100*                               *> (ファイルのOPEN等)
000110     CALL "EXEC".         *> 業務プログラム
000120*
000130     CALL "TERM".        *> 終了プログラム
000140*                               *> (ファイルのCLOSE 等)
000150     STOP  RUN.
```

### 19.4.5.1.2 初期プログラム

初期プログラムは、Webアプリケーションで使用する資源の初期化を行うプログラムです。たとえばファイル入出力機能を使用している場合には、ファイルのオープン処理などを実行します。



例

#### 初期プログラムのコーディング例

```
000010*-----*
000020*  ・モジュール名   INIT      *
000030*  ・動作概要     CGI 初期プログラム *
000040*-----*
000050 IDENTIFICATION DIVISION.
000060 PROGRAM-ID.     INIT.
000070 DATA         DIVISION.
000080 PROCEDURE      DIVISION.
000090 INIT-PROC.
000100*
000110* ファイルのOPEN等、必要に応じて初期処理を行う
000120*      :
000200*
000210 INIT-END.
000220*
000230     EXIT PROGRAM.
```

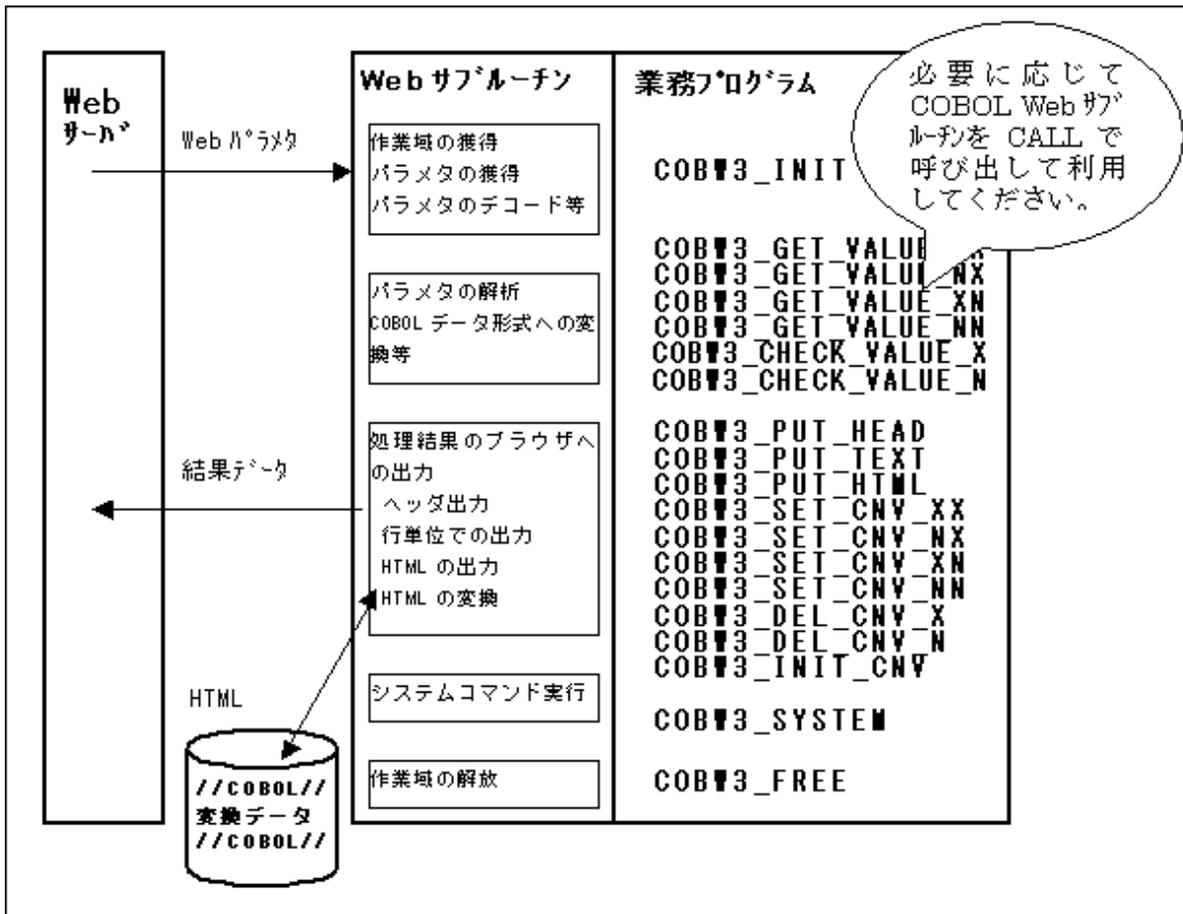


注意

初期プログラムでは、CGIサブルーチンを使用できません。CGIサブルーチンを使用した場合、動作は保証されません。

### 19.4.5.1.3 業務プログラム

業務プログラムとは、Webブラウザからリクエストデータを受け取り、レスポンスデータ(処理結果)をWebブラウザに出力するプログラムです。



Webブラウザから送られてきたリクエストデータ(Webパラメータ)を受け取り、CGIサブルーチンを使って入力データを処理します。処理終了後、実行結果をWebブラウザに出力します。

## 例

### 業務プログラムのコーディング例

```

000010*-----*
000020*   ・モジュール名   EXEC          *
000030*   ・動作概要     CGI 業務プログラム *
000040*-----*
000050 IDENTIFICATION  DIVISION.
000060 PROGRAM-ID       EXEC.
000070 ENVIRONMENT      DIVISION.
000080 CONFIGURATION     SECTION.
000090 INPUT-OUTPUT       SECTION.
000100 DATA             DIVISION.
000110 WORKING-STORAGE   SECTION.
000120 COPY COBW3.
000170*
000180 PROCEDURE         DIVISION.
000190*
000200 EXEC-PROC.
000210*
000220* CGIサブルーチン作業環境の設定および Webパラメータの獲得
000225   MOVE LOW-VALUE   TO   COBW3.
000230   CALL "COBW3_INIT" USING COBW3.
000240*
000250* 名前の取得
000260   MOVE "NAME1"      TO   COBW3-SEARCH-DATA.

```

```

000270 CALL "COBW3_GET_VALUE_XX" USING COBW3.
000280
000290* 結果出力用ページ中の変換データの登録
000300 MOVE "GET-NAME" TO COBW3-CNV-NAME.
000310 MOVE COBW3-GET-DATA TO COBW3-CNV-VALUE.
000320 CALL "COBW3_SET_CNV_XX" USING COBW3.
      :
000760* 結果出力用ページの出力
000770 MOVE "SAMPLE.HTML" TO COBW3-HTML-FILENAME.
000780 CALL "COBW3_PUT_HTML" USING COBW3.
      :
000810* CGIサブルーチン作業領域の解放
000820 CALL "COBW3_FREE" USING COBW3.
000830*
000840*
000850 EXEC-END.
000860*
000870 EXIT PROGRAM.

```

以下、CGIサブルーチンを利用する際の注意事項について説明します。

#### 見出し部

特にありません。

#### 環境部

特にありません。

#### データ部

作業場所節にCGIサブルーチンのインタフェースとなる登録集を、次に示すようにCOPY文で取り込んでください。

```

書き方
WORKING-STORAGE SECTION.
COPY COBW3.

```

なお、この登録集(COBW3.cb1)は、COBOLをインストールしたフォルダに格納されています。

#### 手続き部

CGIサブルーチンの環境設定およびWebパラメタの獲得のため、最初に“COBW3\_INIT”を呼び出す必要があります。また、CGIサブルーチンの処理を終了するとき(手続きの最後)には、必ず“COBW3\_FREE”を呼び出し、CGIサブルーチンの資源の解放をする必要があります。

その他のCGIサブルーチンは、必要に応じてCALL文で呼び出してください。

なお、Webパラメタの受け取り、参照方法および処理結果の出力方法の詳細は、“[1.8 COBOL Webサブルーチンの使い方](#)”を参照してください。

### 19.4.5.1.4 終了プログラム

終了プログラムは、Webアプリケーションの終了処理を行うプログラムです。

たとえば、ファイル入出力機能を使用している場合には、ファイルのクローズ処理などを行います。



#### 例

#### 終了プログラムのコーディング例

```

000010*-----*
000020*  ・モジュール名   TERM          *
000030*  ・動作概要     CGI 終了プログラム *
000040*-----*
000050 IDENTIFICATION DIVISION.
000060 PROGRAM-ID.     TERM.

```

```
000070 DATA          DIVISION.
000080 PROCEDURE       DIVISION.
000090 TERM-PROC.
000100*
000110* ファイルのCLOSE 等、必要に応じて終了処理を行う
          :
000200*
000210 TERM-END.
000220*
000230  EXIT PROGRAM.
```



## 注意

終了プログラムでは、CGIサブルーチンを使用できません。CGIサブルーチンを使用した場合、動作は保証されません。

### 19.4.5.2 呼出し用ページ

呼出し用ページとは、Webアプリケーションを起動する時に使うHTML文書です。CGIを使用するWebアプリケーションを起動するには、`<FORM>`タグや`<A>`タグなどを使用してください。たとえば、メインプログラムをWebアプリケーションの入口として作成する実行形式ファイルの場合、次のように指定します。

その他のHTML文書に関する内容については、“[O.1 Web上で動作するCOBOLアプリケーションをはじめて作成する方のために](#)”を参照してください。



## 例

メインプログラムCGI.cob で作成した実行形式CGI.exe をMETHOD="POST" で実行する場合

```
<FORM METHOD="POST" ACTION="CGI.exe">
```



## 注意

`<A>`タグを使用してWebアプリケーションを起動した場合、Webパラメタを受け渡すことはできません。

### 19.4.5.3 結果出力用ページ

結果出力用ページとは、Webアプリケーションの実行結果をWebブラウザに返却するためのHTML文書です。Webアプリケーションから直接レスポンスデータを返却することもできるため、必ずしも結果出力用ページを用意する必要はありません。ただし、結果出力用ページを用意することで、返却するデータとプログラムの独立性が高まり、Webアプリケーションの再翻訳なしで出力結果のレイアウト変更が可能になるなど、柔軟性、保守性を向上させることができます。

また、結果出力用ページの内容を動的に変換して出力することも可能です。

静的なHTML文書(処理の結果で出力内容が変化しないHTML文書)を出力する場合は、HTML作成ツールで作成したHTML文書そのまま利用できます。

動的なHTML文書(処理の結果によって出力結果が変動するHTML文書)を出力する場合は、結果出力用ページに“`//COBOL//`”で挟んだ項目名(変換名)を指定しておくことで、動的にデータを置き換えて出力できます。結果出力用ページ中の全データを置き換えの対象とするため、結果出力用ページの文字の変更やバックのカラーの指定などのタグを置き換えることも可能です。“[I.4.28 COBW3\\_PUT\\_HTML\(結果出力用ページの出力\)](#)”を参照してください。

結果出力用ページは、タイトル/本文/ヘッダなどを複数のHTML文書に分割して出力したり、プレーンテキストと組み合わせて出力したりすることもできます。

### 19.4.6 作成する際の注意点

以下の点に注意してください。

- Webアプリケーションで利用するCOBOLプログラムでは、画面操作に関する次の機能は利用しないでください。
  - 表示ファイル(画面操作機能)
  - スクリーン操作機能
  - 小入出力機能(ACCEPT文、DISPLAY文)  
(ただし、環境変数、日付および時間の操作機能は利用できます)
- CGIサブルーチンが提供する登録集(COBW3.cbl)を取り込むCOPY文にはREPLACINGを指定しないでください。また、このCOPY文が置き換えの対象となるようなREPLACE文は指定しないでください。
- データ部でデータを初期化(VALUE句など)した場合、運用環境によっては複数のリクエストにまたがった初期値の保障はされません。Webブラウザからのリクエストのたびに初期値を保障する必要がある場合、必ず実行される手続き部でデータを初期化してください。
- Webアプリケーションは、Webサーバが提供するサービス配下で動作するアプリケーションです。サービス配下で動作する場合の注意点として、“[18.1.3 サービス配下で動作するプログラム](#)”を参照してください。

## 19.4.7 実行手順

ここでは、翻訳、リンクおよび実行の基本的な方法を説明します。ここで説明していない翻訳オプションおよびリンクオプションの詳細は“[付録 A 翻訳オプション](#)”および“[J.2.1 LINKコマンドのオプション](#)”を参照してください。

Webアプリケーションの作成および実行は、以下の手順に従ってください。

- 作成したプログラムを翻訳およびリンクする。  
詳細は、“[19.4.7.1 翻訳およびリンク](#)”を参照してください。
- 環境を設定する。
  - Webサーバの環境設定を行う。  
詳細は、ご使用のWebサーバのマニュアルを参照してください。
  - CGIサブルーチンの環境設定を行う。  
詳細は、“[19.4.7.2 CGIサブルーチンの環境変数設定](#)”を参照してください。
- Webアプリケーションを実行する。  
Webサーバに登録した呼出し用ページ(HTML文書)を呼び出して、起動してください。  
詳細は、“[19.4.7.3 CGIアプリケーションの実行](#)”を参照してください。

### 19.4.7.1 翻訳およびリンク

COBOLで記述したメインプログラム、初期プログラム、業務プログラムおよび終了プログラムを以下の方法で翻訳およびリンクします。なお、プロジェクト管理機能を利用している場合、翻訳およびリンクはプロジェクトマネージャのビルドを使用してください。

#### COBOLソースプログラムを翻訳する

COBOLソースプログラムの翻訳は、WINCOBコマンドを実行してウィンドウ操作により行う方法と、翻訳コマンドを使用してNetCOBOLコマンドプロンプト上から指定する方法があります。ここでは、翻訳コマンドを使用して翻訳するときの例を示します。

WINCOBコマンドを使用した翻訳や翻訳コマンドの詳細は、“[25.1.2 WINCOBを使用した翻訳の手順](#)”および“[J.1 COBOL32コマンド](#)”を参照してください。

```
COBOL32 -M メインプログラム.COB
```

```
COBOL32 初期プログラム.COB
```

```
COBOL32 業務プログラム.COB
```

```
COBOL32 終了プログラム.COB
```

## COBOL目的プログラムをリンクする

COBOLソースプログラムを翻訳し作成した目的プログラムのリンクは、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用してNetCOBOLコマンドプロンプト上から指定する方法があります。ここでは、リンクコマンドを使用してリンクするときの例を示します。WINLINKコマンドを使用したリンクやリンクコマンドの詳細は、“25.2.2 WINLINKを使用したリンクの手順”および“J.2 LINKコマンド”を参照してください。

- 複数のオブジェクトファイルを1つの実行形式ファイル(.EXE)として作成(静的リンク)する場合

```
LINK メインプログラム.OBJ 初期プログラム.OBJ 業務プログラム.OBJ 終了プログラム.OBJ F3BICIMP.LIB MSVCRT.LIB  
F3BICWSR.LIB /OUT:実行形式名.EXE
```

- 複数のオブジェクトファイルを1つの実行形式ファイル(.EXE)と複数のダイナミックリンクライブラリ(.DLL)として作成(動的リンク)する場合

```
LINK 初期プログラム.OBJ F3BICIMP.LIB MSVCRT.LIB /DLL /OUT:初期プログラム.DLL
```

```
LINK 業務プログラム.OBJ F3BICIMP.LIB MSVCRT.LIB F3BICWSR.LIB /DLL /OUT:業務プログラム.DLL
```

```
LINK 終了プログラム.OBJ F3BICIMP.LIB MSVCRT.LIB /DLL /OUT:終了プログラム.DLL
```

```
LINK メインプログラム.OBJ 初期プログラム.LIB 業務プログラム.LIB 終了プログラム.LIB F3BICIMP.LIB MSVCRT.LIB /OUT:実行形式名.EXE
```



### 注意

CGIサブルーチンは下方互換がありません。このため、動作するCGIサブルーチンのバージョンレベルよりも新しいバージョンレベルの登録集(COBW3.cbl)は使用できません。また、業務プログラムで取り込んだ登録集のバージョンレベルより前のCGIサブルーチンをリンクしないでください。

## 19.4.7.2 CGIサブルーチンの環境変数設定

CGIサブルーチンを使用したWebアプリケーションを実行するためには、あらかじめ次の実行環境情報をシステム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。

### システム環境変数に設定すべきもの

#### PATH

Webアプリケーションが使用する動的リンクライブラリ(.DLL)のパスを指定します。

### システム環境変数または実行用初期化ファイルに設定するもの

#### @CBR\_ATTACH\_TOOL=TEST [起動パラメタ]

デバッグしたいプログラムからデバッガを起動することを指定します。デバッガを使用して、COBOLで作成したWebアプリケーションをデバッグする場合には、この環境変数を設定する必要があります。

詳細は、“19.8.2.1 CGIアプリケーションの動作確認”を参照してください。

#### @MessOutFile=ファイル名

COBOLランタイムシステムが出力する実行時メッセージを格納するファイル名を指定します。これにより、メッセージボックスが画面に出力されることを抑止します。ファイル名は、絶対パスまたは相対パスで指定できます。相対パスを指定した場合、カレントフォルダからの相対パスとなります。また、既に同名のファイルが存在する場合は、そのファイルにメッセージが追加書きされます。

#### @WinCloseMsg=OFF

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か(ON)、しないか(OFF)、を指定します。Webアプリケーションの実行には、確認メッセージを表示しない(OFF)を指定します。

#### @CBR\_CGI\_LOGFILE=ログファイル名

CGIサブルーチンが出力するログ情報を記録するファイルを指定します。障害時などの原因を調べるときに便利です。なお、ファイル名はサーバ上の絶対パスで指定してください。指定がない場合、ログは出力されません。

#### @CBR\_CGI\_SEVERITY=重要度

CGIサブルーチンが出力するログ情報の重要度を指定します。以下の値が指定できます。これ以外の値を指定した場合、または指定がない場合、重要度は0と見なします。

重要度	ログファイルに出力される内容
0	致命的なエラーだけ出力します。
1	0の出力内容に加えて、エラーを出力します。
2	1の出力内容に加えて、警告を出力します。
3	2の出力内容に加えて、CGIサブルーチンのトレースを出力します。

#### 注意

指定する数値が大きくなるに従ってアプリケーションは極端に遅くなりますので、運用時には必要なエラーだけを出力する値を使用してください。特に、障害原因の調査以外で3を指定することはおすすめしません。

他の実行環境情報および詳細は、“[付録C 環境変数情報](#)”を参照してください。

### 19.4.7.3 CGIアプリケーションの実行

作成したアプリケーションを仮想パスに対応する物理パスに配置し、用意した呼出し用ページを使用して起動します。

## 19.5 ISAPIサブルーチンを使用したアプリケーションの作成と実行

### 19.5.1 ISAPIの概要

ISAPI(Internet Server Application Programming Interface)とは、IISを拡張するための枠組みとそこで利用できるインタフェースです。

ISAPIを使用したアプリケーションであるInternet Server Application(以降、ISAと呼びます)はIISに組み込まれてスレッドとして動作するため、CGIに比べて高速に動作します。また、メモリなどの資源消費も少なくすみます。ただし、ISAはスレッドとして動作するため、アプリケーションはマルチスレッドで動作可能でなければなりません。また、資源の解放を忘れると、IISが終了するまでその資源を使用できなくなる可能性があるため、注意しなければなりません。

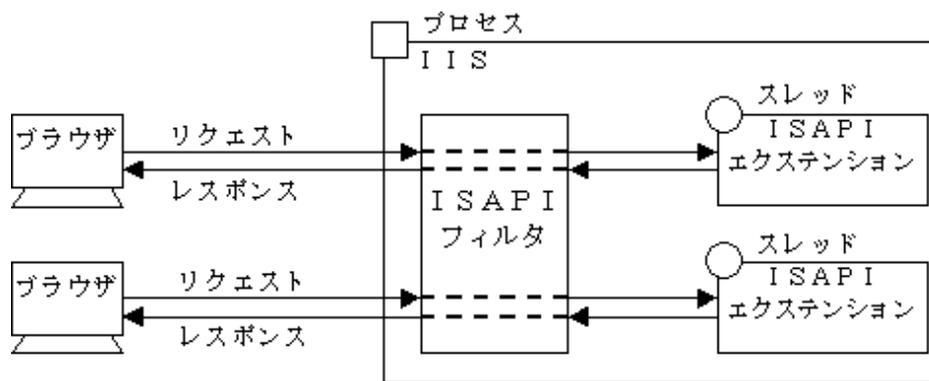
ISAPIで作成できるアプリケーションの形態は次の2つがあります。

#### ISAPIエクステンション

個々のWebアプリケーションを実装する単位です。リクエストごとに使用するアプリケーションを変えることが可能です。

#### ISAPIフィルタ

Webサイト全体に共通して呼び出されるWebアプリケーションです。ISAPIエクステンションの前後に実行され、アプリケーション全体に共通する処理を必要とする場合に使用します。



## 19.5.2 ISAの構成

ISAPIエクステンションの構成に関して紹介します。ただし、COBOL ISAPIサブルーチンではISAPIフィルタはサポートしていません。ISAPIフィルタについては書籍やIISのマニュアルを参照してください。

ISAPIエクステンションは、次の3つの関数をエクスポートしたDLLでなければなりません。なお、いずれの関数もマルチスレッドとして動作します。

### GetExtensionVersion

アプリケーションDLLがロードされたときに一度だけ呼び出される関数です。この関数が終了するまで、次のHttpExtensionProcは実行されません。

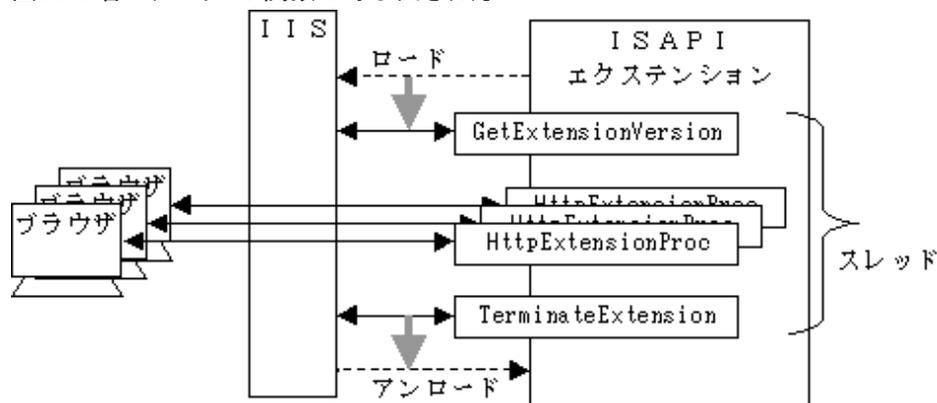
### HttpExtensionProc

リクエストごとに呼び出されるアプリケーション本体です。Webブラウザから送信されたデータを受け取り、HTML文書を送信するための機能がIISへのインタフェースとして提供されています。

### TerminateExtension

アプリケーションDLLがアンロードされるときに呼び出される関数です。

図19.3 各エクスポート関数の呼び出され方



## 19.5.3 COBOL ISAPIサブルーチンとは

COBOL ISAPIサブルーチン(以降、ISAPIサブルーチンと呼びます)は、ISAPIを使用したWebサブルーチンです。このサブルーチンを使用することで、ISAPIエクステンションをCOBOLによって簡単に作成できます。このサブルーチンを使用したWebアプリケーションはISAPIの特徴をそのまま受け継ぎます。また、ISAPIサブルーチンはCOBOL CGIサブルーチンに対して上位互換があるので、COBOL CGIサブルーチンで作成したWebアプリケーションは、わずかな修正でIIS配下のWebアプリケーションに移行することができます。

## 19.5.4 必要なもの

ISAPIサブルーチンを使用してISAを作成し、実行するにあたって、必要なものは次のとおりです。

- ・ IIS

- Webアプリケーションを動作させるために必要です。
- NetCOBOL 開発パッケージ  
Webアプリケーションの翻訳、リンクおよび実行の際に必要です。なお、運用時にはNetCOBOL サーバ運用パッケージが必要です。

## 注意

Webサーバは、Webアプリケーションを動作させるために実行権を設定する必要があります。インストールや設定方法については、運用するWebサーバのシステム管理者に相談してください。

## 19.5.5 作成するもの

ISAPIサブルーチンを使用してISAを作成する場合は、少なくとも以下のものを作成しなければなりません。

- 次の3つの入口名を持つプログラムとこれらをエクスポート関数として作成したISA(DLL)
  - GetExtensionVersion  
DLLのロード時に実行されます。初期化処理として使用できます。
  - HttpExtensionProc  
リクエストごとに実行されるアプリケーション本体です。ISAPIサブルーチンは、この入口名を持つプログラムだけで使用できます。
  - TerminateExtension(必要に応じて作成します)  
DLLのアンロード時に実行されます。終了処理として使用できます。

これらのプログラムの翻訳、リンクは“[19.5.7.1 翻訳およびリンク](#)”を参照してください。
- HTML文書
  - 呼出し用ページ(Webアプリケーション起動用)
  - 結果出力用ページ(処理結果出力用。必要に応じて作成する)

### 19.5.5.1 GetExtensionVersion

ISAがIISにロードされたときに一度だけ呼び出されます。この入口名を持つプログラムの処理が終わるまで、次のHttpExtensionProcが呼び出されることはありません。したがって、作成するISA共通の初期化処理を行うことができます。

たとえば、外部データの初期化などが行えます。ただし、後ほど説明するように、終了処理が行えるタイミングが不明確であるので、終了処理を必要とする処理は行わないでください。

この入口名を持つプログラムは必ず作成しなければなりません。

また、この入口名を持つプログラムを作成する際の注意事項について説明します。

#### 見出し部

プログラム名段落に次に示すようにプログラム名を記述してください。

```
PROGRAM-ID. "GetExtensionVersion".
```

## 注意

プログラム名および大文字/小文字を間違えないでください。間違えるとアプリケーションが動作しません。

#### 環境部

特になし

#### データ部

連絡節にIISとのインタフェースとなる登録集を次に示すようにCOPY文で取り込んでください。

なお、この登録集(ISAPIINF.cbl)は、COBOLをインストールしたフォルダに格納されています。

```
LINKAGE SECTION.  
COPY ISAPIINF.
```

#### 手続き部

IISからの呼出しインタフェースに適合させるために次のように記述してください。

```
PROCEDURE DIVISION WITH STDALL LINKAGE USING ISAPI-INFO.
```

#### ー 復帰コード

この関数の終了時にIISへの復帰コードを設定する必要があります。復帰コードは次の2種類があり、通常は正常終了を設定します。

復帰コード	意味
1	正常終了
0	エラー終了

正常終了を設定するには、次のように記述します。

```
MOVE 1 TO PROGRAM-STATUS.  
EXIT PROGRAM.
```

このプログラムのひな形は次のようになります。

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. "GetExtensionVersion".  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
LINKAGE SECTION.  
COPY ISAPIINF.  
PROCEDURE DIVISION WITH STDALL LINKAGE USING ISAPI-INFO.  
*  
* 必要に応じて処理を記述してください。  
*  
MOVE 1 TO PROGRAM-STATUS.  
EXIT PROGRAM.
```

### 注意

エラー終了を設定すると、次のHttpExtensionProcが呼び出されないので注意してください。

## 19.5.5.2 HttpExtensionProc

Webブラウザからのリクエストごとに呼び出されるWebアプリケーション本体です。ISAPIサブルーチンが利用できるのはこの入口名を持つプログラムだけです。この入口名を持つプログラムは必ず作成しなければなりません。

この入口名を持つプログラムを作成する際の注意事項について説明します。

#### 見出し部

プログラム名段落に次に示すようにプログラム名を記述してください。

```
PROGRAM-ID. "HttpExtensionProc".
```

### 注意

プログラム名および大文字/小文字を間違えないでください。間違えるとアプリケーションが動作しません。

## 環境部

特になし

## データ部

### 作業場所節

ISAPIサブルーチンとのインタフェースとなる登録集を次に示すようにCOPY文で取り込んでください。

なお、この登録集(COBW3.cbl)は、COBOLをインストールしたフォルダに格納されています。

```
WORKING-STORAGE SECTION.  
COPY COBW3.
```

### 連絡節

IISとのインタフェースとなる登録集を次に示すようにCOPY文で取り込んでください。

なお、この登録集(ISAPICTX.cbl)は、COBOLをインストールしたフォルダに格納されています。

```
LINKAGE SECTION.  
COPY ISAPICTX.
```

## 手続き部

IISからの呼出しインタフェースに適合させるために次に示すように記述してください。

```
PROCEDURE DIVISION WITH STDALL LINKAGE USING ISAPI-CTX-CNT.
```

また、上記の方法で取得したISAPI-CTX-CNTはISAPIサブルーチンがIISとのやりとりに必要としますので、次に示すようにそのポインタをCOBW3-CONTEXTに設定します。

```
MOVE LOW-VALUE TO COBW3.  
MOVE FUNCTION ADDR (ISAPI-CTX-CNT) TO COBW3-CONTEXT.
```

### ー 復帰コード

この関数の終了時にIISへの復帰コードを設定する必要があります。復帰コードは次の2種類があり、通常は正常終了を設定します。また、エラー終了を設定すると、サーバは該当アプリケーションでエラーが起きたと判断し、その旨のメッセージをIISがWebブラウザに送信します。

復帰コード	意味
1	正常終了
4	エラー終了

正常終了を設定するには次のように記述します。

```
MOVE 1 TO PROGRAM-STATUS.  
EXIT PROGRAM.
```

このプログラムのひな形は次のようになります。

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. "HttpExtensionProc".  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
COPY COBW3.  
LINKAGE SECTION.  
COPY ISAPICTX.  
PROCEDURE DIVISION WITH STDALL LINKAGE USING ISAPI-CTX-CNT.  
*  
MOVE LOW-VALUE TO COBW3.  
MOVE FUNCTION ADDR (ISAPI-CTX-CNT) TO COBW3-CONTEXT.  
*  
* 必要に応じて処理を記述してください。
```

\*

```
MOVE 1 TO PROGRAM-STATUS.  
EXIT PROGRAM.
```

### 19.5.5.3 TerminateExtension

DLLがアンロードされるときにIISから一度だけ呼び出されるので、Webアプリケーション共通の終了処理を行うことができます。ただし、DLLがアンロードされるタイミングは状況に応じて異なり、プロセス終了時か強制アンロードを行うまでアンロードされないこともあります。したがって、終了処理およびGetExtensionVersionにおいて初期化処理を記述する際はこの点を十分考慮してください。なお、この入口名を持つプログラムは省略することができます。

また、この入口名を持つプログラムを作成する際の注意事項について説明します。

#### 見出し部

プログラム名段落に次に示すようにプログラム名を記述してください。

```
PROGRAM-ID. "TerminateExtension".
```



プログラム名および大文字/小文字を間違えないでください。間違えるとこの関数が呼び出されません。

#### 環境部

特になし

#### データ部

連絡節にIISとのインタフェースとなる登録集を次に示すようにCOPY文で取り込んでください。

```
LINKAGE SECTION.  
COPY ISAPIFLG.
```

なお、この登録集(ISAPIFLG.cbl)は、COBOLをインストールしたフォルダに格納されています。

#### 手続き部

IISからの呼出しインタフェースに適合させるために次のように記述してください。

```
PROCEDURE DIVISION WITH STDALL LINKAGE USING ISAPI-FLAG.
```

#### ー 復帰コード

この関数の終了時にIISへの復帰コードを設定する必要があります。復帰コードは次の2種類あり、通常は正常終了を設定します。また、エラー終了を設定すると、サーバは該当アプリケーションでエラーが起きたと判断します。

復帰コード	意味
1	正常終了
0	エラー終了

正常終了を設定するには次のように記述します。

```
MOVE 1 TO PROGRAM-STATUS.  
EXIT PROGRAM.
```

このプログラムのひな形は次のようになります。

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. "TerminateExtension".  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
LINKAGE SECTION.  
COPY ISAPIFLG.
```

```
PROCEDURE DIVISION WITH STD CALL LINKAGE USING ISAPI-FLAG.
```

```
*
```

```
* 必要に応じて処理を記述してください。
```

```
*
```

```
MOVE 1 TO PROGRAM-STATUS.
```

```
EXIT PROGRAM.
```

#### 19.5.5.4 呼出し用ページ

呼出し用ページとは、Webアプリケーションを起動する時に使うHTML文書です。Webアプリケーションを起動するためには<FORM>タグや<A>タグなどを使用してください。

その他のHTML文書に関する内容については、“[O.1 Web上で動作するCOBOLアプリケーションをはじめて作成する方のために](#)”を参照してください。

- 作成したダイナミックリンクライブラリISAPI.dllをMETHOD="POST"で実行する場合

```
<FORM METHOD="POST" ACTION="ISAPI.dll">
```



注意

<A>タグを使用してWebアプリケーションを起動した場合、Webパラメタを受け渡すことはできません。

#### 19.5.5.5 結果出力用ページ

結果出力用ページとは、Webアプリケーションの実行結果をWebブラウザに返却するためのHTML文書です。Webアプリケーションから直接レスポンスデータを返却することもできるため、必ずしも結果出力用ページを用意する必要はありません。ただし、結果出力用ページを用意することで、返却するデータとプログラムの独立性が高まり、Webアプリケーションの再翻訳なしで出力結果のレイアウト変更が可能になるなど、柔軟性、保守性を向上させることができます。

また、結果出力用ページの内容を動的に変換して出力することも可能です。

静的なHTML文書(処理の結果で出力内容が変化しないHTML文書)を出力する場合は、HTML作成ツールで作成したHTML文書そのまま利用できます。

動的なHTML文書(処理の結果によって出力結果が変動するHTML文書)を出力する場合は、結果出力用ページに“//COBOL//”で挟んだ項目名(変換名)を指定しておくことで、動的にデータを置き換えて出力できます。HTML文書中の全データを置き換えの対象とするため、結果出力用ページの文字の変更やバックのカラーの指定などのタグを置き換えることも可能です。“[1.4.28 COBW3\\_PUT\\_HTML\(結果出力用ページの出力\)](#)”を参照してください。

結果出力用ページは、タイトル/本文/ヘッダなどを複数のHTML文書に分割して出力したり、プレーンテキストと組み合わせて出力したりすることもできます。

### 19.5.6 ISAを作成する際の注意点

以下の点に注意してください。

- Webアプリケーションで利用するCOBOLプログラムでは、画面操作に関する次の機能は利用しないでください。
  - 表示ファイル(画面操作機能)
  - スクリーン操作機能
  - 小入出力機能(ACCEPT文、DISPLAY文)  
(ただし、環境変数、日付および時間の操作機能は利用できます。)
- ISAPIサブルーチンは入口名:HttpExtensionProcのプログラム内でだけ使用可能です。
- ISAPIサブルーチンを使用する前に、必ず、パラメタで渡される集団項目(ISAPI-CTX-CNT)へのポインタを取得(FUNCTION ADDR)し、その値をCOBW3-CONTEXTに設定してください。
- COBW3およびISAPI-CTX-CNTとそのポインタデータはスレッド間で共有しないでください。IISが正常に動作しなくなる可能性があります。

- ISAPIサブルーチンが提供する登録集(COBW3.cbl、ISAPICTX.cbl、ISAPIINF.cbl、ISAPIFLG.cbl)を取り込むCOPY文では、REPLACING指定を行わないでください。また、COPY文が置換えの対象となるようなREPLACE文の指定は行わないでください。
- データ部でデータを初期化(VALUE句など)した場合、運用環境によっては複数のリクエストにまたがった初期値の保障はされません。Webブラウザからのリクエストのたびに初期値を保障する必要がある場合、必ず実行される手続き部でデータを初期化してください。
- Webアプリケーションは、Webサーバが提供するサービス配下で動作するアプリケーションです。サービス配下で動作する場合の注意点として、“18.1.3 サービス配下で動作するプログラム”を参照してください。

## 19.5.7 実行手順

翻訳、リンクおよび実行方法について、ここでは基本的な方法を説明します。ここで説明していない翻訳オプションおよびリンクオプションの詳細は、“付録A 翻訳オプション”および“J.2.1 LINKコマンドのオプション”を参照してください。

Webアプリケーションを実行するには、以下の手順に従ってください。

- 作成したプログラムを翻訳およびリンクする。  
詳細は、“19.5.7.1 翻訳およびリンク”を参照してください。
- 環境を設定する。
  - IISの環境設定を行う。  
詳細は、“19.7.2 IISの設定方法”を参照してください。
  - ISAPIサブルーチンの環境設定を行う。  
詳細は、“19.5.7.2 ISAPIサブルーチンの環境変数設定”を参照してください。
- Webアプリケーションを実行する。  
IISに登録した呼出し用ページ(HTML文書)を呼び出して、起動してください。  
詳細は、“19.5.7.3 ISAPIアプリケーション(ISA)の実行”を参照してください。

### 19.5.7.1 翻訳およびリンク

GetExtensionVersion、HttpExtensionProc、TerminateExtensionの入口名を持つプログラムを以下の方法で翻訳およびリンクします。なお、それぞれのファイルをここではVersion.cob、MainProc.cob、Terminate.cobとして説明します。なお、プロジェクト管理機能を利用している場合、翻訳およびリンクはプロジェクトマネージャのビルドを使用してください。

#### COBOLソースプログラムを翻訳する

COBOLソースプログラムの翻訳は、プロジェクトマネージャからWINCOBコマンドを実行してウィンドウ操作により行う方法と、翻訳コマンドを使用してNetCOBOLコマンドプロンプト上から指定する方法があります。ここでは、翻訳コマンドを使用して翻訳するときの例を示します。

WINCOBコマンドを使用した翻訳や翻訳コマンドの詳細は、“25.1.2 WINCOBを使用した翻訳の手順”および“J.1 COBOL32コマンド”を参照してください。

```
COBOL32 -WC, "ALPHAL (WORD), THREAD (MULTI)" Version.cob
COBOL32 -WC, "ALPHAL (WORD), THREAD (MULTI)" MainProc.cob
COBOL32 -WC, "ALPHAL (WORD), THREAD (MULTI)" Terminate.cob
```



翻訳オプションALPHAL(WORD)の代わりにNOALPHALも指定できます。エクスポート関数となるプログラムの入口名は大文字/小文字を区別する必要があるため、どちらかを必ず指定する必要があります。また、ISAはマルチスレッドアプリケーションであるため、翻訳オプションTHREAD(MULTI)も必ず指定してください。

## COBOL目的プログラムをリンクする(DLLの作成)

COBOLソースプログラムを翻訳し作成した目的プログラムのリンクは、プロジェクトマネージャから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用してNetCOBOLコマンドプロンプト上から指定する方法があります。ここでは、リンクコマンドを使用してリンクするときの例を示します。WINLINKコマンドを使用したリンクやリンクコマンドの詳細は、“[25.2.2 WINLINKを使用したリンクの手順](#)”および“[J.2 LINKコマンド](#)”を参照してください。

作成するDLLをISASMPL.dllとします。リンクを行う前に、次に示すモジュール定義ファイルを用意しておきます。ファイル名はここではISASMPL.defとして説明します。

```
LIBRARY ISASMPL
EXPORTS
    GetExtensionVersion
    HttpExtensionProc
    TerminateExtension
```

リンクは次のように行います。

```
LINK /DLL /OUT:ISASMPL.dll Version.obj MainProc.obj Terminate.obj F3BICBDM.obj F3BISAPI.lib F3BICIMP.lib
KERNEL32.LIB MSVCRT.LIB /DEF:ISASMPL.def /ENTRY:COBDMAN
```

リンク時にF3BICBDM.objをリンクし、/ENTRY:COBDMANを指定することで、初期化ファイル(COBOL85.CBR)をカレントフォルダとは関係なく、DLLと同じフォルダに配置することができます。IISなどのサーバ配下のアプリケーションのカレントフォルダは不定であるため、初期化ファイルをDLLと同じフォルダに配置できるのは非常に便利です。なお、詳細は、“[5.5.4 DLL配下にある実行用の初期化ファイルを使用する](#)”を参照してください。



### 注意

ISAPIサブルーチンは下方互換がありません。このため、動作するサブルーチンのバージョンレベルよりも新しいバージョンレベルの登録集(COBW3.cbl)は使用できません。また、Webアプリケーションで取り込んだ登録集のバージョンレベルより前のISAPIサブルーチンを結合してはなりません。

## 19.5.7.2 ISAPIサブルーチンの環境変数設定

ISAPIサブルーチンを使用したWebアプリケーションを実行するためには、あらかじめ次の実行環境情報を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。

なお、業務ごとに異なる実行用の初期化ファイルを使用する場合は、各業務アプリケーションを別々のフォルダに配置し、IIS(MMC)を使用して業務ごとに異なる仮想ディレクトリを割り当て、分離プロセスとして構成する必要があります。さらに、各アプリケーションにはF3BICBDM.objをリンクし、アプリケーションが格納されているフォルダの実行用の初期化ファイルを参照できるようにしておいてください。

### システム環境変数に設定すべきもの

PATH

Webアプリケーションが使用する動的リンクライブラリ(.DLL)のパスを指定します。

### システム環境変数または実行用初期化ファイルに設定するもの

@CBR\_ATTACH\_TOOL=TEST [起動パラメタ]

デバッグしたいプログラムからデバッガを起動することを指定します。デバッガを使用して、COBOLで作成したWebアプリケーションをデバッグする場合には、この環境変数を設定する必要があります。

詳細は、“[19.8.2.2 ISAPIアプリケーションの動作確認](#)”を参照してください。

@MessOutFile=ファイル名

COBOLランタイムシステムにより出力される実行時メッセージを格納するファイル名を指定します。これにより、メッセージボックスが画面に出力されることを抑止します。ファイル名の指定は、絶対パスを使用してください。また、既に同名のファイルが存在する場合は、そのファイルにメッセージが追加書きされます。

#### @WinCloseMsg=OFF

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か(ON)、しないか(OFF)、を指定します。Webアプリケーションの実行時は、確認メッセージを表示しない(OFF)を指定します。

#### @CBR\_ISAPI\_LOGFILE=ログファイル名

ISAPIサブルーチンが出力するログ情報を記録するファイルを指定します。障害時などの原因を調べるときに便利です。なお、この場合もファイル名はサーバ上の絶対パスで指定してください。指定がない場合、ログは出力されません。

#### @CBR\_ISAPI\_SEVERITY=重要度

ISAPIサブルーチンが出力するログ情報の重要度を指定します。以下の値が指定できます。これ以外の値を指定した場合、または指定がない場合、重要度は0とみなします。

他の実行環境情報および詳細は、“付録C 環境変数情報”を参照してください。

重要度	ログファイルに出力される内容
0	致命的なエラーだけ出力します。
1	0の出力内容に加えて、エラーを出力します。
2	1の出力内容に加えて、警告を出力します。
3	2の出力内容に加えて、ISAPIサブルーチンのトレースを出力します。

### 注意

指定する数値が大きくなるに従ってアプリケーションは極端に遅くなりますので、運用時には必要なエラーだけを出力する値を使用してください。特に、障害原因の調査以外で3を指定することはおすすめしません。

## 19.5.7.3 ISAPIアプリケーション(ISA)の実行

作成したWebアプリケーションを仮想ディレクトリに対応する物理パスに配置し、用意した呼出し用ページを使用して起動します。

### 参考

テストなどを行っているときWebアプリケーションの入れ替えが頻繁に起きます。また、運用環境においてもWebアプリケーションの機能拡張を行った場合、入れ替えが必要になります。しかし、一度Webアプリケーションを実行すると、IISにロードされた状態のままとなり、入れ替えることができません。

Webアプリケーションを入れ替える方法については、“19.8.2.2.4 Webアプリケーションを入れ替える”を参照してください。

## 19.6 WebアプリケーションにおけるUnicodeの利用

### 19.6.1 Unicodeの利用

NetCOBOL V7.0L10以降で提供するCOBOL Webサブルーチンは、すべてUnicodeをサポートしています。Unicodeで動作するWebアプリケーションは、文字数の増加により表現力が向上しますが、従来のシフトJISを使用したWebアプリケーションと異なる点があります。以下、Unicodeを使用したWebアプリケーションを作成、運用する上での注意事項について記述します。

なお、MeFt/WebでUnicodeを使用する場合は、“MeFt/Web ユーザーズガイド”を参照してください。

はじめに、シフトJISを使用したWebアプリケーションとUnicodeを使用したWebアプリケーションの作成および運用上の違いについて下表に示します。

項番	項目	シフトJIS	Unicode
1	Web ブラウザ	シフトJIS が表示できるもの 最近のWeb ブラウザならどれも可能	UTF-8またはUCS-2が表示でき、Web パラメタをUTF-8で送信できるWeb ブラ ウザ

項番	項目	シフトJIS	Unicode
2	Web ブラウザからの入力コード系	シフトJIS またはEUC(注)	UTF-8(注)
3	Webブラウザへの出力コード系	シフトJIS	UTF-8またはUCS-2
4	翻訳オプション	不要	必要
5	使用可能なサブルーチン	制限なし	一部制限あり
6	呼出し用ページに使用できるHTML文書のコード系	シフトJIS またはEUC	UTF-8またはUCS-2
7	結果出力用ページに使用できるHTML文書のコード系	シフトJIS	UTF-8、UCS-2またはシフトJIS
8	ログファイルのコード系	シフトJIS	UCS-2またはシフトJIS
9	コード系の異なるCOBOL アプリケーションの混在	基本的には不可 ただし、使用するサブルーチンやWeb サーバによっては、ある条件下で可能。	

注：COBOL WebサブルーチンがCOBOLアプリケーションのコード系に合わせて変換処理を行うため、COBOLアプリケーションで入力コード系を意識する必要はありません。

## 19.6.2 Webブラウザ

従来のコード系のWebアプリケーションは、日本語化されているWebブラウザであれば、問題なく使用することができました。しかし、UnicodeのWebアプリケーションは、UnicodeをサポートしているWebブラウザしか使用できません。特に古いWebブラウザはUnicodeをサポートしていないものが多いため、注意してください。また、最近のWebブラウザでもサポートの程度に差があります。COBOL WebサブルーチンでUnicodeを使用した場合、Webブラウザは次の条件を満たすものでなければなりません。

- UTF-8またはUCS-2のHTML文書が正しく表示できる。
- Webブラウザから送信するWebパラメタをUTF-8で送信できる。

## 19.6.3 Webブラウザからの入力コード系

シフトJISのWebアプリケーションは、Webブラウザからの入力コード系がシフトJISかEUCかを自動判別しています。したがって、呼出し用ページのHTML文書のコード系はシフトJISでもEUCでも問題ありませんでした。しかし、Unicodeを使用したWebアプリケーションはUTF-8しか認識できません。これは、UTF-8と他のコード系を自動的に判別できないこと、および一般的なWebブラウザがUnicodeのHTML文書を使用した時にWebパラメタのコード系をUTF-8として送信するためです。したがって、UnicodeのWebアプリケーションを起動する呼出し用ページは、Unicodeでなければなりません。

## 19.6.4 Webブラウザへの出力コード系

従来のシフトJISのWebアプリケーションでは、Webブラウザへの出力コード系をシフトJISとしていました。しかし、UnicodeのWebアプリケーションでは、UTF-8とUCS-2を選択できるようになっています。システム全体を通して半角英数字が多い場合はデータ量の少なくなるUTF-8、全角文字が多い場合はデータ量の少なくなるUCS-2を選択できます。このWebブラウザへの出力コード系は以下の環境変数情報で指定します。

@CBR\_WEB\_OUT\_CODE

Webブラウザへの出力コード系を指定します。

値	意味
UTF8	Web ブラウザへの出力コード系をUTF-8とします。
UCS2	Web ブラウザへの出力コード系をUCS-2リトルエンディアンとします。



- シフトJISのWebアプリケーションは出力コード系がシフトJIS固定のため、この環境変数に値を指定しても意味を持ちません。

- Webアプリケーションからこの環境変数情報を操作した場合の動作は不定です。

## 19.6.5 翻訳オプション

シフトJISのWebアプリケーションには必須の翻訳オプションはありませんが、UnicodeのWebアプリケーションには翻訳オプション”RCS(UTF16)”を指定しなければなりません。

## 19.6.6 使用可能なサブルーチン

シフトJISのWebアプリケーションでは、COBOLプログラム中で使用できるWebサブルーチンに制限はありません。しかし、UnicodeのWebアプリケーションでは、下表のWebサブルーチンは使用できません。代替サブルーチンを使用してください。なお、資産の移行性を高めるためには、シフトJISのWebアプリケーションでも代替サブルーチンの使用をおすすめします。

サブルーチンの詳細は“[1.8 COBOL Webサブルーチンの使い方](#)”を参照してください。

サブルーチン名	代替サブルーチン
<u>COBW3_NAME</u>	COBW3_GET_VALUE_XX COBW3_GET_VALUE_NX
COBW3_NAMEN	COBW3_GET_VALUE_XN COBW3_GET_VALUE_NN
COBW3_VALUE	COBW3_CHECK_VALUE_X COBW3_CHECK_VALUE_N
COBW3_CNV_SET	COBW3_SET_CNV_XX COBW3_SET_CNV_NX COBW3_SET_CNV_XN COBW3_SET_CNV_NN
COBW3_CNV_DEL	COBW3_DEL_CNV_X COBW3_DEL_CNV_N
COBW3_CNV_INIT	COBW3_INIT_CNV
COBW3_SET_COOKIE	COBW3_SET_COOKIE_XX COBW3_SET_COOKIE_NX COBW3_SET_COOKIE_XN COBW3_SET_COOKIE_NN
COBW3_DEL_COOKIE	COBW3_DEL_COOKIE_X COBW3_DEL_COOKIE_N
COBW3_GET_COOKIE	COBW3_GET_COOKIE_XX COBW3_GET_COOKIE_NX
COBW3_GET_COOKIEEN	COBW3_GET_COOKIE_XN COBW3_GET_COOKIE_NN

## 19.6.7 呼出し用ページに使用できるHTML文書のコード系

シフトJISのWebアプリケーションでは、呼出し用ページに使用できるHTML文書のコード系はシフトJISおよびEUCです。しかし、UnicodeのWebアプリケーションでは、呼出し用ページに使用できるHTML文書のコード系はUTF-8またはUCS-2(Webブラウザに依存します)でなければなりません。

## 19.6.8 結果出力用ページに使用できるHTML文書のコード系

---

シフトJISのWebアプリケーションでは、結果出力用ページに使用できるHTML文書のコード系はシフトJISだけでした。しかし、UnicodeのWebアプリケーションでは以下のコード系のHTML文書を扱えます。

- ・ シフトJIS
- ・ UTF-8
- ・ UCS-2

ただし、使用するHTMLファイルのコード系とWebブラウザへの出力コード系を統一させておく方が実行性能はよくなります。

## 19.6.9 ログファイルのコード系

---

Webサブルーチンが出力するログファイルのコード系は、以下により決定します。

新規にログファイルが作成される場合

実行環境で最初に動作したCOBOLアプリケーションのコード系で作成します。

すでに作成済みのログファイルにログ情報が追加される場合

既存ログファイルのコード系で作成します。

## 19.6.10 コード系の異なるCOBOLアプリケーションの混在

---

同一プロセス内ではコード系の異なるCOBOLアプリケーションは混在できません。したがって、IISでは、Webアプリケーションのコード系を混在することはできません。

ただし、IISでは分離プロセスを使用することで、同一Webサーバ内でコード系を混在できます。

また、CGIアプリケーションはそれぞれ独立したプロセスとして動作するため、同一サーバ内でコード系を混在させることができます。

COBOLアプリケーションにおけるコード系の混在に関しては“[第6章 文字コード](#)”を参照してください。

## 19.7 COBOL Webサブルーチン

---

Webサブルーチンの使い方と各サブルーチンの説明は、“[1.8 COBOL Webサブルーチンの使い方](#)”を参照してください。

### 19.7.1 Webサブルーチンの提供するクラス

---

Webサブルーチンでは、セッション管理機能で使用するクラスを提供しています。

#### COBW3-SESSION-ADAPTERクラス

セッションデータとして登録されたオブジェクトを管理するためのクラスです。セッションデータに登録するオブジェクトは、当クラスを継承するクラスから生成されなければなりません。また、当クラスに定義されているSWEEP-SESSIONメソッドを、OVERRIDE句を利用して上書きしたメソッド内で、タイムアウト処理を記述します。



注意

- ・ タイムアウト処理は、セッションデータにオブジェクト参照が登録されている場合だけ利用できます。
- ・ タイムアウト処理では、Webサブルーチンを呼び出すことはできません。

ファクトリメソッド

なし

## オブジェクトメソッド

### SWEEP-SESSION

セッションのタイムアウト発生時に呼び出されるメソッド。必要に応じて、ファイルのクローズやDBのコミット/ロールバック等の処理を記述するために利用します。

#### メソッドの引数

なし

#### メソッドの復帰情報

なし



### 注意

- セッションのタイムアウトが発生した場合、WebサブルーチンはSWEEP-SESSIONメソッドを呼び出し、セッションデータに登録されているオブジェクト参照にNULLオブジェクトを代入します。
- SWEEP-SESSIONメソッドが上書き定義されていない場合、SWEEP-SESSIONメソッドは何も処理を行いません。
- COBW3-SESSION-ADAPTERクラスは、FJBASEクラスを継承するクラスです。



### 例

- COBOLプログラム

```
IDENTIFICATION DIVISION.  
CLASS-ID. SESSDATA INHERITS COBW3-SESSION-ADAPTER.  
:  
IDENTIFICATION DIVISION.  
OBJECT.  
:  
IDENTIFICATION DIVISION.  
METHOD-ID. SWEEP-SESSION OVERRIDE.  
DATA DIVISION.  
:  
PROCEDURE DIVISION.  
:  
* 必要に応じてタイムアウト処理を記述する  
:  
END METHOD SWEEP-SESSION.  
:  
END OBJECT.  
END CLASS SESSDATA.
```

## 19.7.2 IISの設定方法

IISの設定は、呼出し用ページなどのHTML文書の参照やWebアプリケーションの実行のために必要な仮想ディレクトリの登録を行うだけです。ここでは、その手順について説明します。

より高度な設定や、設定についての詳細は、書籍やIISのオンラインマニュアルを参照してください。

なお、IISの設定には、インターネットサービスマネージャを使用します。以下の説明は、Webサブルーチンを動作させるためのIISの設定方法について、サンプルプログラム(WSESSION)を用いています。IISを設定する前に、インターネットインフォメーションサービスをインストールしてください。

デバッガを利用してデバッグする場合は、“[19.8.2.2 ISAPIアプリケーションの動作確認](#)”を行ってください。

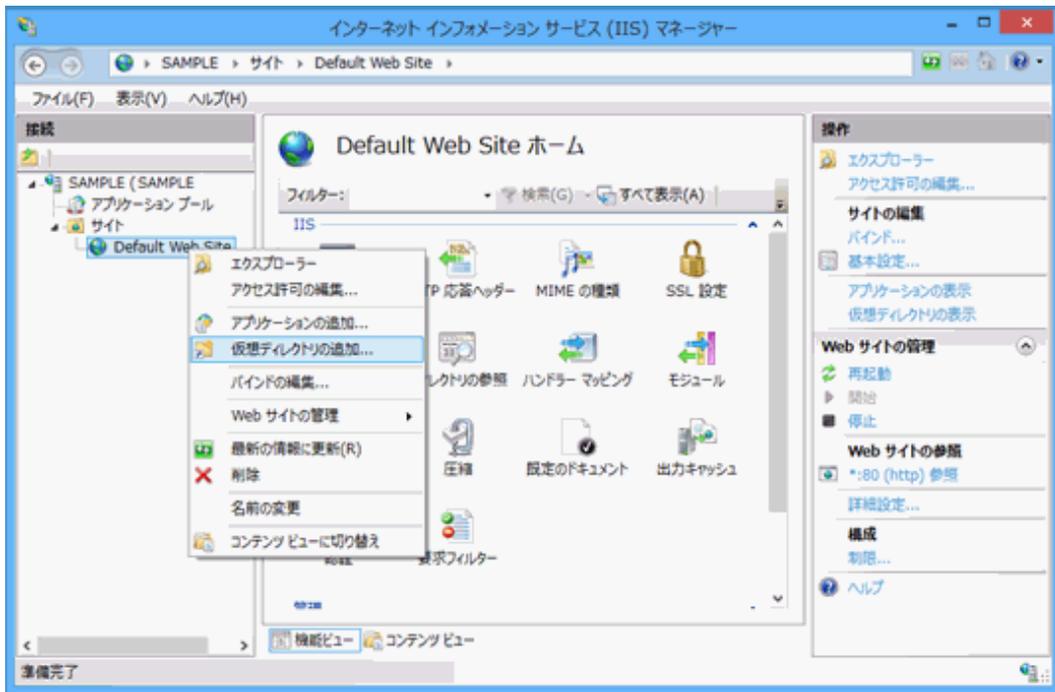
## 注意

IISで設定したCGI/ISAPIアプリケーションは、デフォルトでは、[IUSR\_<コンピュータ名>]のアカウントで実行されます。  
[IUSR\_<コンピュータ名>]のアカウントは、アクセス許可のないファイルへのアクセスやCOBW3\_SYSTEMサブルーチンによるシステムコマンドの実行に失敗することがあります。  
ディレクトリやファイルのセキュリティとアクセス許可を確認し、必要に応じて、適切なアクセス許可の付加やアカウントの変更を行ってください。

### IIS の設定手順(IIS 8.5の場合)

インターネット インフォメーション サービスをインストールしただけでは、Webサブルーチンは利用できません。以下の手順で、ISAPI拡張またはCGI機能を有効にしてください。

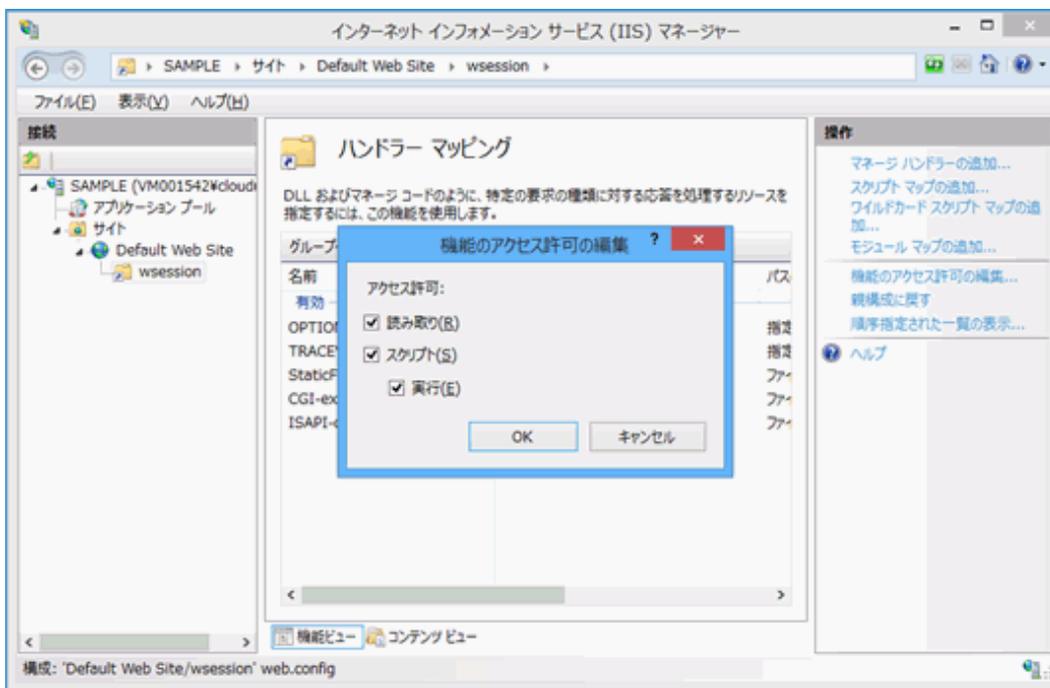
1. [コントロールパネル]の[プログラムと機能]を実行します。
2. [Windowsの機能の有効化または無効化]タスクを実行します。
3. [インターネット インフォメーション サービス]の[World Wide Web サービス]にある[アプリケーション開発機能]を選択します。  
[ISAPI拡張](CGIを利用する場合は、[CGI])を追加インストールしてください。
4. [インターネット インフォメーション サービス(IIS)マネージャー]を起動し、Webサイトに仮想ディレクトリを作成します。



5. 仮想ディレクトリにアクセスするためのエイリアスを指定します。任意の名称を指定してください。また、Webサイト上で公開するコンテンツを含む物理パスを指定してください。

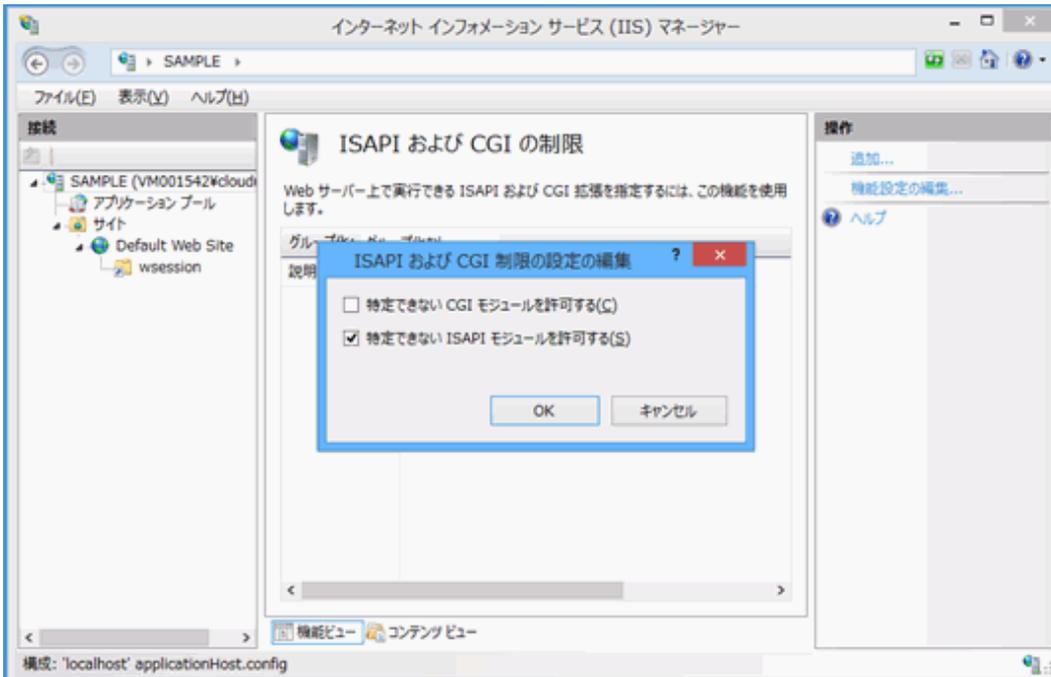


6. 作成した仮想ディレクトリを選択し、[ハンドラーマッピング]機能を開き、[機能のアクセス許可の編集]で[実行]にチェックします。

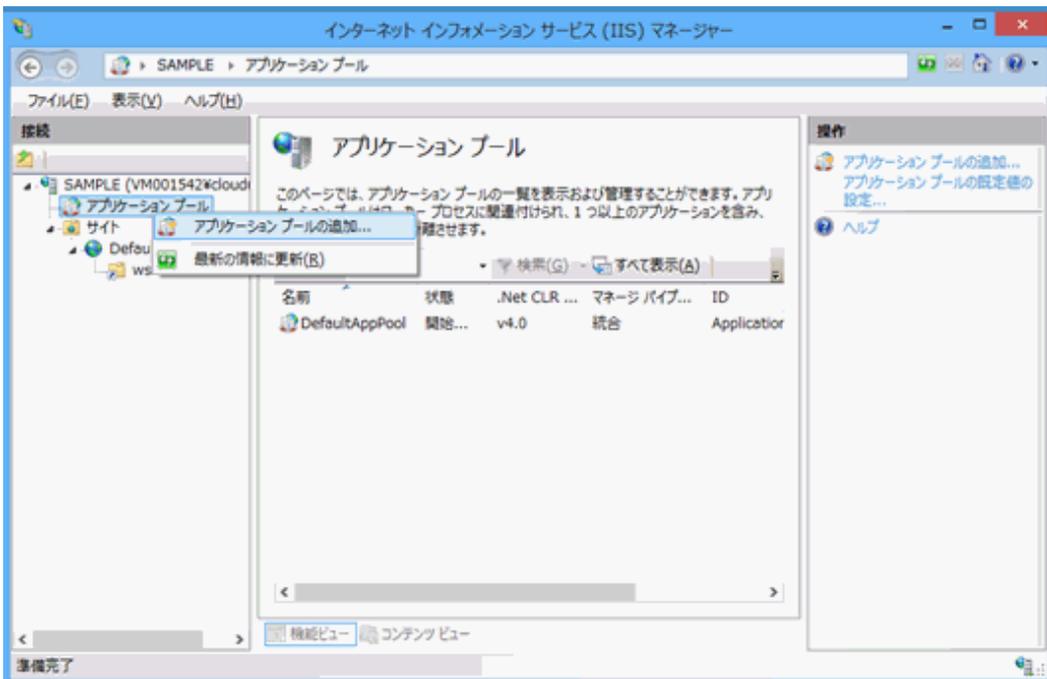


7. インターネットインフォメーションサービス(IIS)マネージャのルートにある[ISAPIおよびCGIの制限]機能を開き、[機能設定の編集]を選択します。

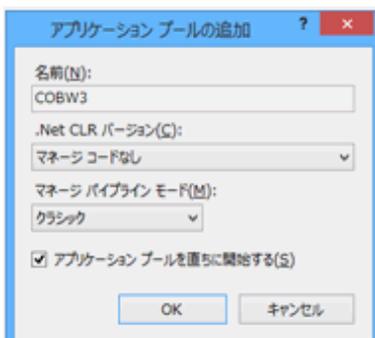
[特定できないISAPIモジュールを許可する]をチェックしてください。  
(CGIを利用する場合は、[特定できないCGIモジュールを許可する]をチェックしてください)



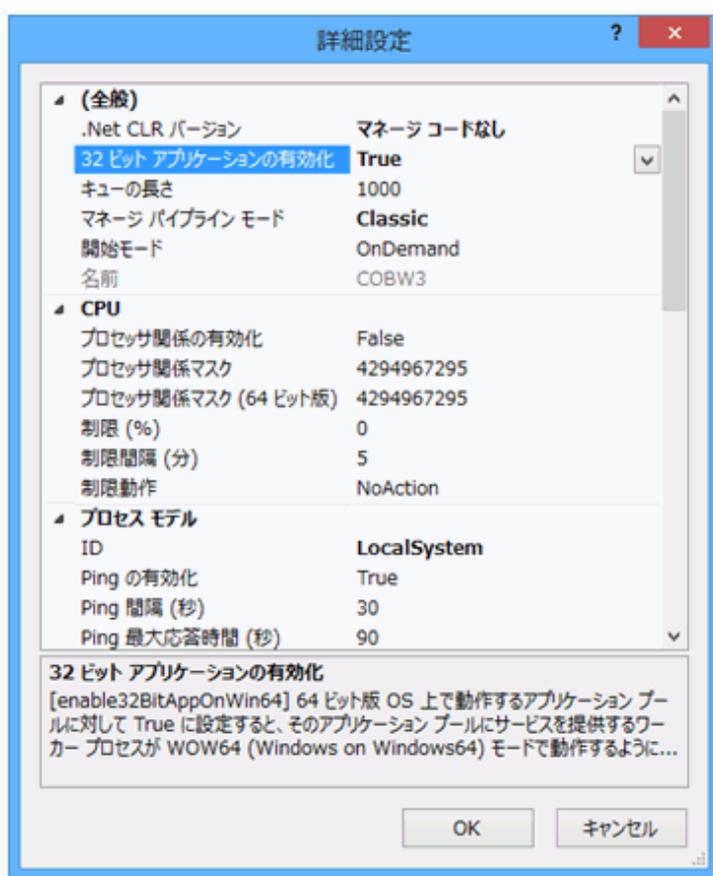
8. Webサブルーチンを動作させるためのアプリケーションプールを作成します。



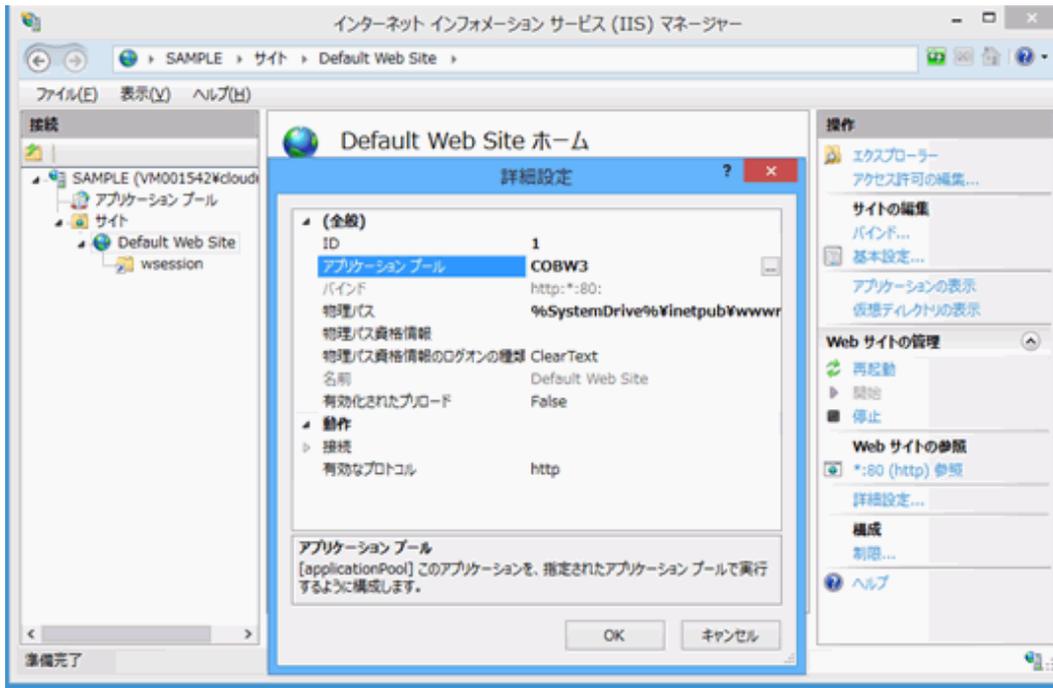
アプリケーションプールIDは任意の名称を指定できます。ここではCOBW3としています。  
[.Net CLRバージョン]は[マネージコードなし]を、[マネージパイプラインモード]は[クラシック]を選択してください。



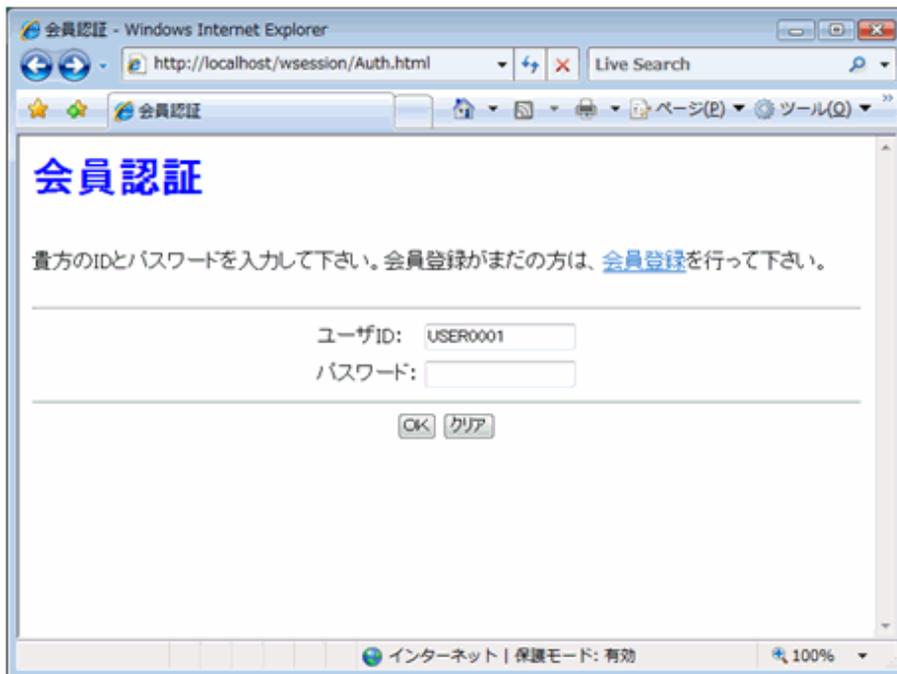
9. 作成したアプリケーションプールの[詳細設定]を開き、[プロセスモデル]の[ID]を選択し、プルダウンメニューから[LocalSystem]を選択してください。  
64ビットOSを使用している場合、CGIまたはISAPIアプリケーションをWOW64モードで動作させる必要があります。[32ビットアプリケーションの有効化]に[True]を設定してください。



10. “Default Web Site”の[詳細設定]を開き、[全般]の[アプリケーションプール]を選択し、プルダウンメニューから、新規に作成したアプリケーションプールIDを指定します。



11. 作成した仮想ディレクトリにブラウザからアクセスして動作を確認します。



## 注意

- NetCOBOLをアンインストールした後は、作成したアプリケーションプールを削除してください。
- アプリケーションプールはリサイクルできません。定期的な間隔による自動リサイクルや、アプリケーションプールタスクの操作によるリサイクルを行わないでください。リサイクルを行うと、動作中のISAPIアプリケーションのセッションが終了します。
- Webガーデンは使用できません。ワーカークロスの最大数に1を超える値を指定しないでください。

## 19.8 動作確認

作成したプログラムの動作確認(テスト時の確認または問題発生時の確認)するには、以下の方法があります。

- ・ ログ情報を参照する
- ・ 対話型デバッガで動作確認する
- ・ NetCOBOL Studioのアタッチデバッグ機能で動作確認する
- ・ Webサブルーチンが検出したエラーを参照する
- ・ 実行中のデータを表示形式で参照する

### 19.8.1 ログ情報を参照する

Webサブルーチンではログ情報をログファイルに出力するための機構を用意しています。このログ情報はアプリケーションを変更することなく、設定を変更することができるので運用時のデバッグに有効です。設定方法は、以下を参照してください。

- ・ “[19.4.7.2 CGIサブルーチンの環境変数設定](#)”
- ・ “[19.5.7.2 ISAPIサブルーチンの環境変数設定](#)”

なお、このログ情報はWebサブルーチンのログ情報であるため、アプリケーションそのもののトレースには役立ちませんが、動作確認の目安になります。ログファイルの構成は次のとおりです。

プロセスID	スレッドID	年-月-日	時:分:秒	重要度	メッセージ
--------	--------	-------	-------	-----	-------



#### 例

```
0000000188 0000000187 1999-02-02 11:48:07 02 COB-06310: COBW3: 指定された変換名は既に登録されています。既に登録されている変換情報を有効にします。
```

```
0000000188 0000000081 1999-02-02 11:48:10 01 COB-04470: COBW3: 指定された変換情報は登録されていないため、変更できませんでした。
```

なお、アプリケーションのトレース情報を採取する場合には、COBOLのTRACE機能を使用してください。TRACE機能の詳細は、“[21.4 TRACE機能](#)”を参照してください。

### 19.8.2 対話型デバッガで動作確認する

デバッガを使用してCOBOLで作成したWebアプリケーション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。

リモートデバッガを使用して、Webブラウザを起動しているクライアントからWebアプリケーションをリモートでデバッグすることも出来ます。リモートデバッガの使い方については、“[24.9 対話型リモートデバッガの使い方](#)”を参照してください。

#### 19.8.2.1 CGIアプリケーションの動作確認

デバッガを使用してCGIアプリケーションのデバッグを開始するには、以下の手順で行います。

1. Webアプリケーションを翻訳・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、Webアプリケーションを作成します。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“[24.2.2 プログラムの翻訳](#)”および“[24.2.3 プログラムのリンク](#)”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、あらかじめ次の実行環境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)に設定してください。システムの環境変数に設定した場合は、設定後システムを再起動してください。

```
@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)ことを指定します。

“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24.1 起動パラメタ](#)”を参照してください。

3. デバッグするWebアプリケーションが実行されるコンピュータにログオンします。
4. Webアプリケーションが起動されたとき、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。

以降の操作は、通常のデバッガを使用したデバッグと同じです。デバッガの使用方法については、“[24.5 デバッガの操作](#)”を参照してください。

## 注意

デバッガを使用してデバッグを行う場合は、Web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。したがって、Webブラウザによっては、デバッグ中にタイムアウトが発生することがあります。タイムアウトが設定できるWebブラウザは、適切な値に変更してください。また、タイムアウトが設定できるWebサーバは、適切な値に変更してください。

### 19.8.2.2 ISAPIアプリケーションの動作確認

デバッガを使用してデバッグする場合には、IISはWebアプリケーションを同じメモリ空間に読み込んで呼び出すため、デバッグするには以下の点を考慮する必要があります。

- Webアプリケーションが不正な動作をした場合、Webアプリケーションが異常になるばかりでなく、同時にロードされているほかのWebアプリケーションやIIS自身も異常になる可能性があります。デバッグ時には、まず、ほかのWebアプリケーションが何も組み込まれていない状態で検証し、次にいくつかのWebアプリケーションを混在させて確認することを推奨します。
- デバッガは、Webアプリケーションだけでなく、ほかのWebアプリケーションやIIS自身もデバッグの対象としています。よって、デバッガを終了させるとほかのWebアプリケーションやIIS自身も終了するため、IISを再起動する必要があります。

IISには、WebアプリケーションをIISのメモリ空間で動作させる方法(IISプロセス)とIISとは別のメモリ空間で動作させる方法(分離プロセスまたはプール)があります。Webアプリケーションを分離プロセスまたはプールにした場合、上記の問題がなくなるため、デバッグ作業が容易になります。

## 注意

- IIS 7.0以降では、COBOLで作成したWebアプリケーションを実行するために、アプリケーションプールを作成する必要があります。“[IISの設定手順 \(IIS 8.5の場合\)](#)”を参照し、アプリケーションプールを作成してください。
- Webアプリケーションで発生するアプリケーションエラーは、IIS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およびCOBOLランタイムシステムの診断機能は動作しません。

#### 19.8.2.2.1 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. Webアプリケーションを翻訳・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、Webアプリケーションを作成します。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、“[24.2.2 プログラムの翻訳](#)”および“[24.2.3 プログラムのリンク](#)”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、あらかじめ次の実行環境情報を設定しておく必要があります。実行環境情報の設定方法については“[19.5.7.2 ISAPIサブルーチンの環境変数設定](#)”を参照してください。

```
@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)ことを指定します。

“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24.1 起動パラメタ](#)”を参照してください。

3. デバッグするWebアプリケーションが実行されるコンピュータにログオンします。デバッグ中は、IISのサービスを停止・起動することがあるので、管理者権限でログオンしてください。
4. WebブラウザからWebアプリケーションを起動します。このとき、WebブラウザはデバッグするWebアプリケーションが実行されるコンピュータでなくてもかまいません。

5. Webアプリケーションが起動されたとき、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。

### 19.8.2.2.2 デバッグを行う

デバッグ操作は、通常のデバッガを使用したデバッグと同じです。デバッガの使用方法については、“[24.5 デバッガの操作](#)”を参照してください。ただし、デバッグを行う前に、以下の点に注意する必要があります。

- デバッガは、デバッグを開始したときにWebアプリケーションの入口で実行を自動的に中断し、デバッグ操作の機会を与えます。WebアプリケーションがWebブラウザに回答した後に次のリクエストによって実行されたときは、自動的に中断しないため、あらかじめ中断点を設定しておく必要があります。なお、次のリクエストによってロードされるWebアプリケーションで実行を中断させるには、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中断点の設定を指示します。詳細は、“[24.6.1 動的構造に対するデバッグ方法](#)”を参照してください。
- Webアプリケーションで発生するアプリケーションエラー(例外)は、IISが検知して処理するため、デバッガはアプリケーションエラーが発生したことを知るできません。デバッガでアプリケーションエラーを検知するには、デバッガの[環境設定]ダイアログの[動作]ページで、[初回のシグナル受信時に中断する]をチェックしてください。詳細は、“[24.6.2 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。
- Webブラウザに回答した後は、Webアプリケーション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から行ってください。

デバッグを行う際には、次の点にも注意をする必要があります。

- デバッガを使用してデバッグを行う場合は、Web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。したがって、Webブラウザによっては、デバッグ中にタイムアウトが発生することがあります。タイムアウトが設定できるWebブラウザは、適切な値に変更してください。また、IISの接続タイムアウトの値を適切な値に変更してください。

### 19.8.2.2.3 デバッグを終了する

デバッガを先に終了させる場合

Webアプリケーションが分離プロセスまたはプールの場合

デバッガを終了すると、Webアプリケーションのプロセスだけが終了します。したがって、IIS自身は終了しないため、IISを再起動する必要はありません。

WebアプリケーションがIISプロセスの場合

デバッガを終了すると、IIS自身も終了します。したがって、IISを再起動する必要があります。

Webアプリケーションを先に終了させる場合

Webアプリケーションが分離プロセスまたはプールの場合

インターネットサービスマネージャを使って、Webアプリケーションが属する仮想ディレクトリの[プロパティ]を開き、[アンロード]ボタンを押して、Webアプリケーションのプロセスを終了させます。なお、Webアプリケーションのプロセスを終了させる過程では、プロセスが終了するまでWebアプリケーションを実行させなければなりません。

WebアプリケーションがIISプロセスの場合

IISを停止させます。

IISは、インターネットサービスマネージャを使っても完全には停止しないため、サービスを開いて、IISのすべてのサービス(IIS Admin Serviceおよびそれに依存するサービス)を停止させます。なお、IISを停止させる過程では、プロセスが終了するまでWebアプリケーションを実行させなければなりません。

### 19.8.2.2.4 Webアプリケーションを入れ替える

Webアプリケーションの入れ替えは、以下の方法で行います。

Webアプリケーションが分離プロセスまたはプールの場合

Webアプリケーションのプロセスを終了させてからWebアプリケーションを入れ替えます。

Webアプリケーションのプロセスを終了させるには、インターネットサービスマネージャを使ってWebアプリケーションが属する仮想ディレクトリの[プロパティ]を開き、[アンロード]ボタンを押します。なお、Webアプリケーションのプロセスを終了させる過程では、プロセスが

終了するまでWebアプリケーションを実行させなければなりません。この方法でWebアプリケーションを入れ替えられない場合は、WebアプリケーションがIISプロセスの場合に示す方法で行ってください。

#### WebアプリケーションがIISプロセスの場合

IISを停止させてからWebアプリケーションを入れ替えて、再度IISを開始します。

IISは、インターネットサービスマネージャを使っても完全には停止しないため、サービスを開いて、IISのすべてのサービス(IIS Admin Serviceおよびそれに依存するサービス)を停止させます。なお、IISを停止させる過程では、プロセスが終了するまでWebアプリケーションを実行させなければなりません。

IISを開始するには、インターネットサービスマネージャを使って行うか、またはサービスを開いて、IISのすべてのサービス(IIS Admin Serviceおよびそれに依存するサービス)を開始します。

### 19.8.3 NetCOBOL Studioのデバッグ機能で動作確認する

NetCOBOL Studioのアタッチデバッグ機能を使用して、COBOLで作成したWebアプリケーションをデバッグできます。アタッチデバッグ機能の詳細は、“NetCOBOL Studio ユーザーズガイド”の“アタッチデバッグ”を参照してください。

#### 19.8.3.1 CGIアプリケーションの動作確認

NetCOBOL Studioのアタッチデバッグ機能を使用してCGIアプリケーションのデバッグを開始するには、以下の手順で行います。

1. Webアプリケーションを翻訳・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、Webアプリケーションを作成します。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“NetCOBOL Studio ユーザーズガイド”の“デバッグ機能”を参照してください。

2. 実行環境情報を設定します。

NetCOBOL Studioのアタッチデバッグ機能を使用するために、あらかじめ次の実行環境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)に設定してください。システムの環境変数に設定した場合は、設定後システムを再起動してください。

```
@CBR_ATTACH_TOOL=接続先/STUDIO [追加パスリスト]
```

アプリケーションの実行時にNetCOBOL Studioのアタッチデバッグ機能を使用することを指定します。

接続先、および追加パスリストについては、“C.2.2 @CBR\_ATTACH\_TOOL(プログラムからデバッグ機能を起動する指定)”を参照してください。

3. NetCOBOL Studioを起動して、アタッチ形式のデバッグを開始し、デバッグ待機状態にします。
4. Webアプリケーションが起動された時、待機状態のNetCOBOL Studioに自動的に接続し、デバッグが開始されます。

以降の操作は、通常のデバッグ機能を使用したデバッグと同じです。デバッグ機能の使用方法については、“NetCOBOL Studio ユーザーズガイド”を参照してください。



NetCOBOL Studioを使用してデバッグを行う場合は、Web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。したがって、Webブラウザによっては、デバッグ中にタイムアウトが発生することがあります。タイムアウトが設定できるWebブラウザは、適切な値に変更してください。また、タイムアウトが設定できるWebサーバは、適切な値に変更してください。

#### 19.8.3.2 ISAPIアプリケーションの動作確認

NetCOBOL Studioを使用してデバッグする場合には、IISはWebアプリケーションを同じメモリ空間に読み込んで呼び出すため、デバッグするには以下の点を考慮する必要があります。

- Webアプリケーションが不正な動作をした場合、Webアプリケーションが異常になるばかりでなく、同時にロードされているほかのWebアプリケーションやIIS自身も異常になる可能性があります。デバッグ時には、まず、ほかのWebアプリケーションが何も組み込まれていない状態で検証し、次にいくつかのWebアプリケーションを混在させて確認することを推奨します。
- NetCOBOL Studioのデバッグ機能は、Webアプリケーションだけでなく、ほかのWebアプリケーションやIIS自身もデバッグの対象としています。よって、デバッグ機能を終了させるとほかのWebアプリケーションやIIS自身も終了するため、IISを再起動する必要があります。

IISには、WebアプリケーションをIISのメモリ空間で動作させる方法(IISプロセス)とIISとは別のメモリ空間で動作させる方法(分離プロセスまたはプール)があります。Webアプリケーションを分離プロセスまたはプールにした場合、上記の問題がなくなるため、デバッグ作業が容易になります。

## 注意

- IIS 7.0以降では、COBOLで作成したWebアプリケーションを実行するために、アプリケーションプールを作成する必要があります。“[IISの設定手順 \(IIS 8.5の場合\)](#)”を参照し、アプリケーションプールを作成してください。
- Webアプリケーションで発生するアプリケーションエラーは、IIS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COBOLランタイムシステムの診断機能は動作しません。

### 19.8.3.2.1 デバッグを開始する

NetCOBOL Studioのアタッチデバッグ機能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. Webアプリケーションを翻訳・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、Webアプリケーションを作成します。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“[NetCOBOL Studio ユーザーズガイド](#)”を参照してください。

2. 実行環境情報を設定します。

NetCOBOL Studioのアタッチデバッグ機能を使用するために、あらかじめ次の実行環境情報を設定しておく必要があります。実行環境情報の設定方法については“[19.5.7.2 ISAPIサブルーチンの環境変数設定](#)”を参照してください。

```
@CBR_ATTACH_TOOL=接続先/STUDIO [追加パスリスト]
```

アプリケーションの実行時にNetCOBOL Studioのアタッチデバッグ機能を使用することを指定します。

接続先、および追加パスリストについては、“[C.2.2 @CBR\\_ATTACH\\_TOOL \(プログラムからデバッグ機能を起動する指定\)](#)”を参照してください。

3. NetCOBOL Studioを起動して、アタッチ形式のデバッグを開始し、デバッグ待機状態にします。
4. WebブラウザからWebアプリケーションを起動します。このとき、WebブラウザはデバッグするWebアプリケーションが実行されるコンピュータでなくてもかまいません。
5. Webアプリケーションが起動された時、待機状態のNetCOBOL Studioに自動的に接続し、デバッグが開始されます。

### 19.8.3.2.2 デバッグを行う

デバッグ操作は、通常のデバッグ機能を使用したデバッグと同じです。デバッグ機能の使用方法については、“[NetCOBOL Studio ユーザーズガイド](#)”を参照してください。ただし、デバッグを行う前に、以下の点に注意する必要があります。

- デバッグ機能は、自動的に中断しないため、あらかじめブレークポイントを設定しておく必要があります。ブレークポイントの設定方法については、“[NetCOBOL Studio ユーザーズガイド](#)”を参照してください。
- Webアプリケーションで発生するアプリケーションエラー(例外)は、IISが検知して処理するため、デバッグ機能はアプリケーションエラーが発生したことを知るできません。
- Webブラウザに応答した後は、Webアプリケーション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。[ウォッチ]ビューへのデータ項目追加などのデバッグ操作を行いたい場合は、NetCOBOL Studioのメニューから[実行]-[中断]を選択し、プログラムの実行を中断させてから行ってください。

デバッグを行う際には、次の点にも注意をする必要があります。

- NetCOBOL Studioを使用してデバッグを行う場合は、Web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。したがって、Webブラウザによっては、デバッグ中にタイムアウトが発生することがあります。タイムアウトが設定できるWebブラウザは、適切な値に変更してください。また、IISの接続タイムアウトの値を適切な値に変更してください。

### 19.8.3.2.3 デバッグを終了する

デバッグ機能を先に終了させる場合

Webアプリケーションが分離プロセスまたはプールの場合

デバッグ機能を終了すると、Webアプリケーションのプロセスだけが終了します。したがって、IIS自身は終了しないため、IISを再起動する必要はありません。

WebアプリケーションがIISプロセスの場合

デバッグ機能を終了すると、IIS自身も終了します。したがって、IISを再起動する必要があります。

Webアプリケーションを先に終了させる場合

Webアプリケーションが分離プロセスまたはプールの場合

インターネットサービスマネージャを使って、Webアプリケーションが属する仮想ディレクトリの[プロパティ]を開き、[アンロード]ボタンを押して、Webアプリケーションのプロセスを終了させます。なお、Webアプリケーションのプロセスを終了させる過程では、プロセスが終了するまでWebアプリケーションを実行させなければなりません。

WebアプリケーションがIISプロセスの場合

IISを停止させます。

IISは、インターネットサービスマネージャを使っても完全には停止しないため、サービスを開いて、IISのすべてのサービス(IIS Admin Serviceおよびそれに依存するサービス)を停止させます。なお、IISを停止させる過程では、プロセスが終了するまでWebアプリケーションを実行させなければなりません。

### 19.8.3.2.4 Webアプリケーションを入れ替える

Webアプリケーションの入れ替えは、以下の方法で行います。

Webアプリケーションが分離プロセスまたはプールの場合

Webアプリケーションのプロセスを終了させてからWebアプリケーションを入れ替えます。

Webアプリケーションのプロセスを終了させるには、インターネットサービスマネージャを使ってWebアプリケーションが属する仮想ディレクトリの[プロパティ]を開き、[アンロード]ボタンを押します。なお、Webアプリケーションのプロセスを終了させる過程では、プロセスが終了するまでWebアプリケーションを実行させなければなりません。この方法でWebアプリケーションを入れ替えられない場合は、WebアプリケーションがIISプロセスの場合に示す方法で行ってください。

WebアプリケーションがIISプロセスの場合

IISを停止させてからWebアプリケーションを入れ替えて、再度IISを開始します。

IISは、インターネットサービスマネージャを使っても完全には停止しないため、サービスを開いて、IISのすべてのサービス(IIS Admin Serviceおよびそれに依存するサービス)を停止させます。なお、IISを停止させる過程では、プロセスが終了するまでWebアプリケーションを実行させなければなりません。

IISを開始するには、インターネットサービスマネージャを使って行うか、またはサービスを開いて、IISのすべてのサービス(IIS Admin Serviceおよびそれに依存するサービス)を開始します。

## 19.8.4 Webサブルーチンが検出したエラーを参照する

Webサブルーチンが検出したエラーを参照する方法には、プログラム中の“COBW3\_INIT”を呼び出す直前で、デバッグモードの設定を行います。

```
000190      :  
          SET COBW3-DMODE-DBG TO TRUE.  
000200      CALL "COBW3_INIT" USING COBW3.  
          :
```

この設定により、Webサブルーチン内でエラーが検出された場合には、エラーメッセージがWebブラウザに表示されます。

また、ヘッダ出力については、COBW3-CONTENT-TYPE-NON以外が設定されていると、COBW3-CONTENT-TYPE-HTMLとみなしてCOBW3\_INITがContent-typeを宣言します。

## 参考

デバッグモードの設定は、プログラムを修正し、Webアプリケーションを再作成する必要があります。デバッグモードの設定により、Webブラウザに表示されるエラーメッセージはログ情報でも参照できます。したがって、アプリケーション変更なしにエラーメッセージを参照したい場合には、ログ情報の利用をおすすめします。

## 注意

Webアプリケーションの動作コード系がUnicodeの場合、デバッグモードの設定は無効になります。ログ情報を参照するなどの手段によって動作確認してください。

## 19.8.5 実行中のデータを表示形式で参照する

---

実行中のデータを表示形式で参照するには、COBW3\_PUT\_TEXTを使用します。参照したいデータをCOBW3\_PUT\_TEXTで指定して、Webブラウザで任意のデータを参照しながらデバッグする方法です。

## 注意

COBW3\_PUT\_TEXTによるデバッグでは、表示形式でないデータ(バイナリデータなど)は表示できません。また、翻訳/実行の繰り返し作業が頻繁に発生するため、効率が悪くなりがちです。

したがって、このような場合には実行中のデータを対話型デバッガで参照する方法を使ってデバッグすることをおすすめします。

## 第20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。

### 20.1 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。

#### 20.1.1 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扱い、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、対話型デバッグおよびNetCOBOL Studioのデバッグ機能により、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取得サブルーチン](#)/[スレッドID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。

#### 20.1.2 機能範囲

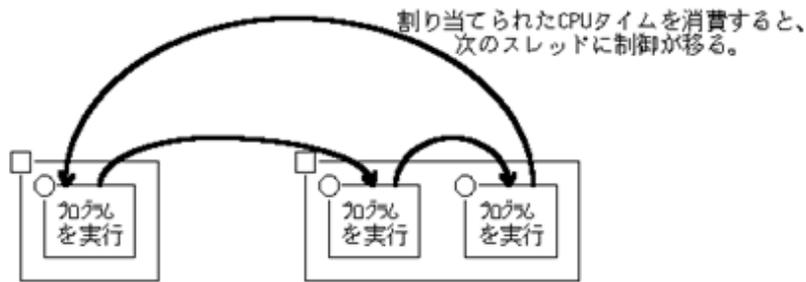
マルチスレッド環境では、COBOLの機能をすべて利用することができます。ただし、関連製品を利用する機能については、その関連製品の対応状況を確認の上、使用してください。[参照]“NetCOBOL ソフトウェア説明書”

## 20.2 マルチスレッドのメリット

### 20.2.1 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時間を割り当てます。割り当てられたCPU時間を消費すると、次のスレッドにCPU時間を割り当てます。

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。

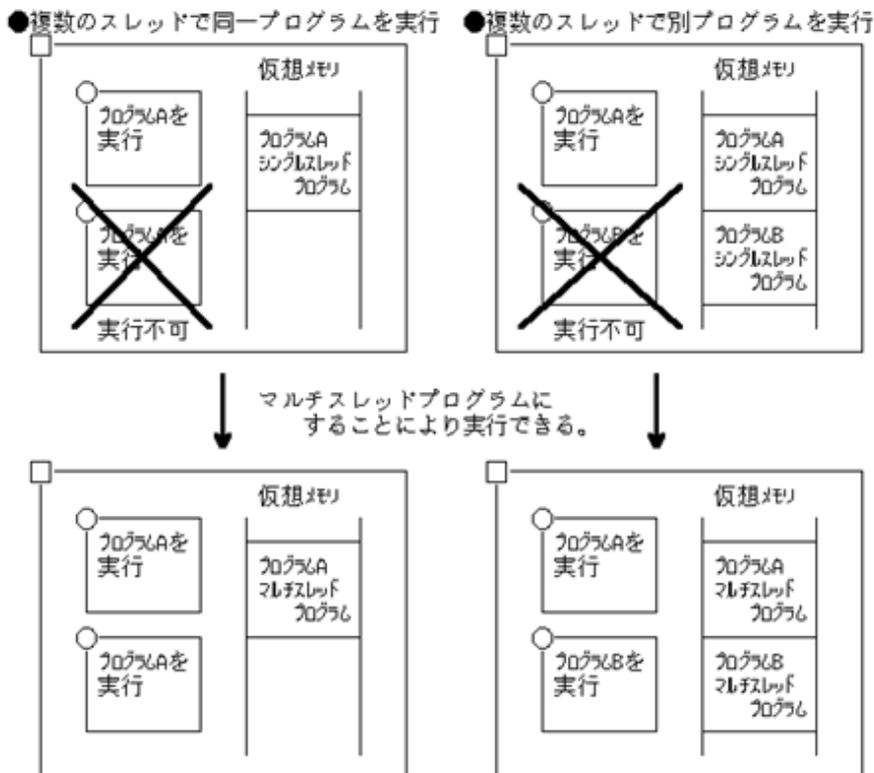
○ □ : スレッドを示す。

以降、プロセスとスレッドをこの図で表記します。

## 20.2.2 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。



## 20.2.3 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ってマルチスレッドプログラムの効果を説明します。

図20.1 シングルスレッドプログラム(CGIアプリケーション)

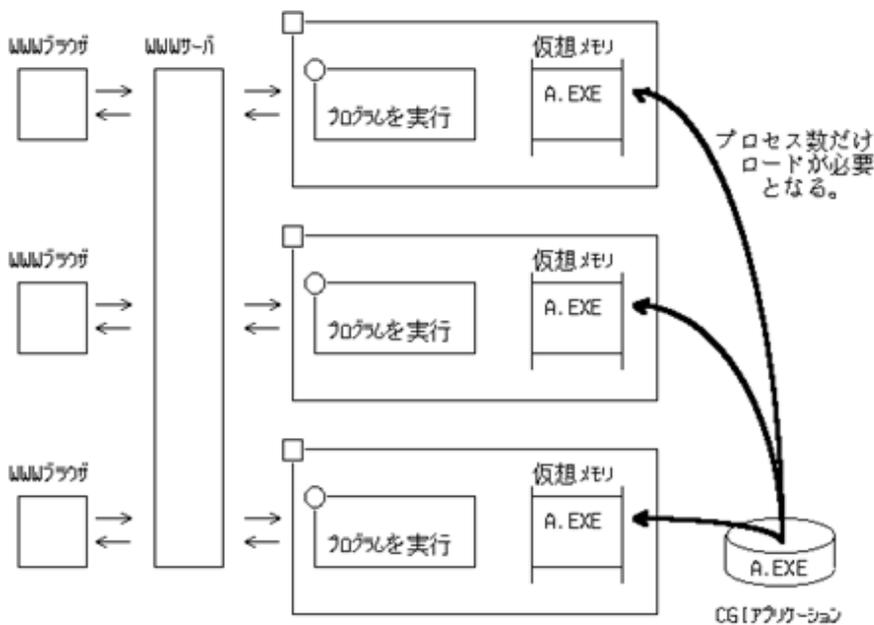
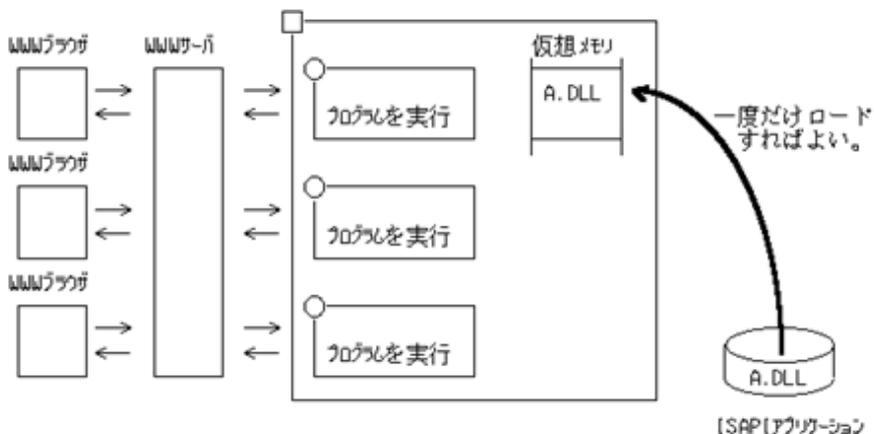


図20.2 マルチスレッドプログラム(ISAPIアプリケーション)



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、通常はその後も常駐します。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。

## 20.3 マルチスレッドプログラムの基本動作

### 20.3.1 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されるファイル
- ・ 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。

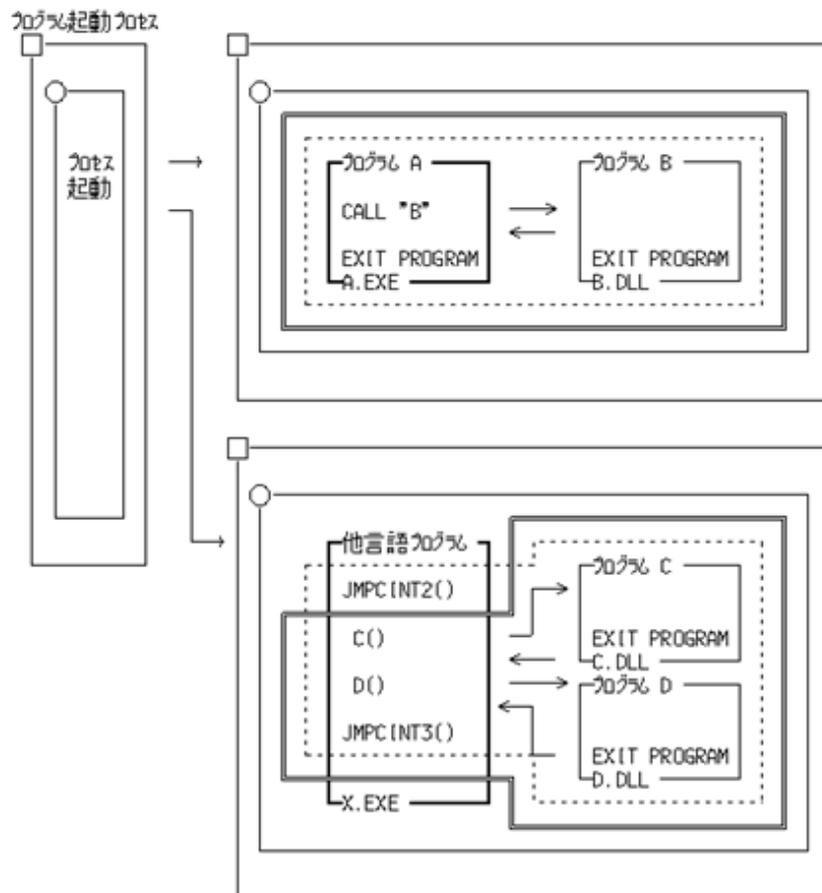
#### JMPCINT4

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、“1.3 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関係を図示します。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。

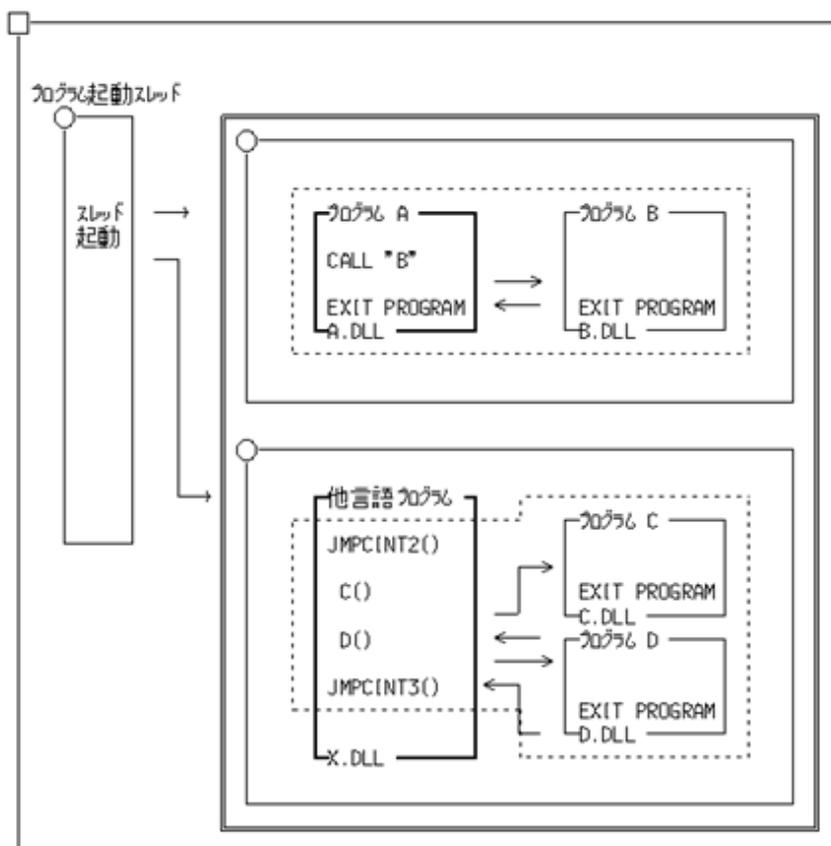


□ : COBOL の主プログラムを示す。 □ : COBOL の実行単位を示す。

□ : 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。

## 20.3.2 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

### スレッド単位で確保/管理されるデータ

- プログラム定義に宣言されたデータ(注2)

### 呼び出し単位で確保/管理されるデータ

- メソッド定義に宣言されたデータ

注1 : 翻訳オプションTHREAD(MULTI)のほかに翻訳オプションSHREXTを指定して翻訳されたCOBOLソースプログラム中のEXTERNAL句が指定されたデータまたはファイルを指します。

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。

### 20.3.2.1 プログラム定義に宣言されたデータ

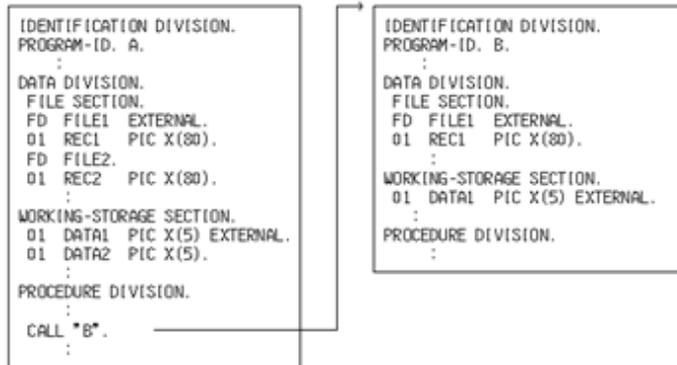
プログラム定義に宣言されたデータはスレッドごとに確保されます。この領域は、実行単位の開始時に確保され、実行単位の終了時に解放されます。実行単位の終了時、クローズされていないファイル、カーソルなどは強制的にクローズされます。



注意

リモートデータベースアクセス機能を利用している場合だけ、カーソルなどはクローズされます。プリコンパイラを利用している場合は、オープンされたままの状態で行単位の終了してしまうため、実行単位の終了前に必ずクローズしてください。

図20.3 プログラム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、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20.4 シングルスレッドプログラム

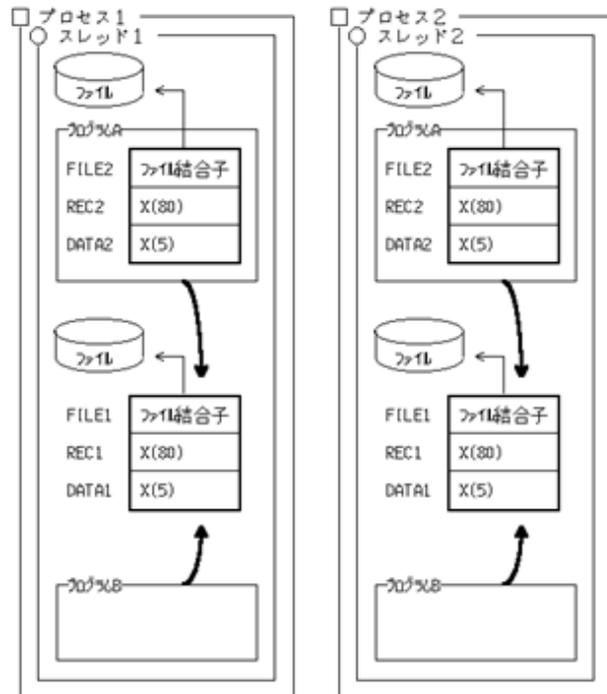
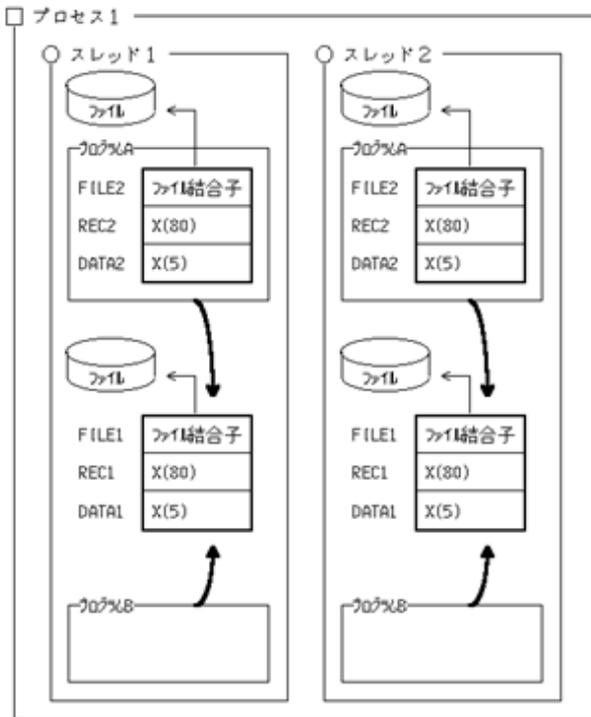


図20.5 マルチスレッドプログラム



### 20.3.2.2 ファクトリオブジェクトとオブジェクトインスタンス

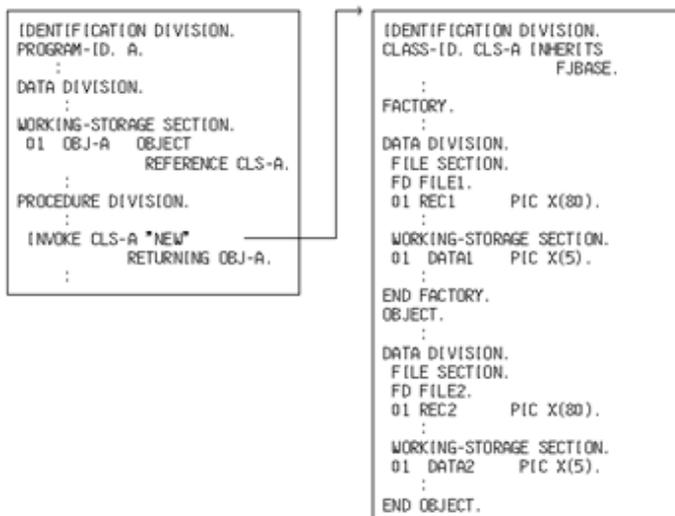
ファクトリオブジェクトとオブジェクトインスタンスはプロセスで管理されます。

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] “20.4.2.2 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] “20.4.2.3 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20.6 シングルスレッドプログラム

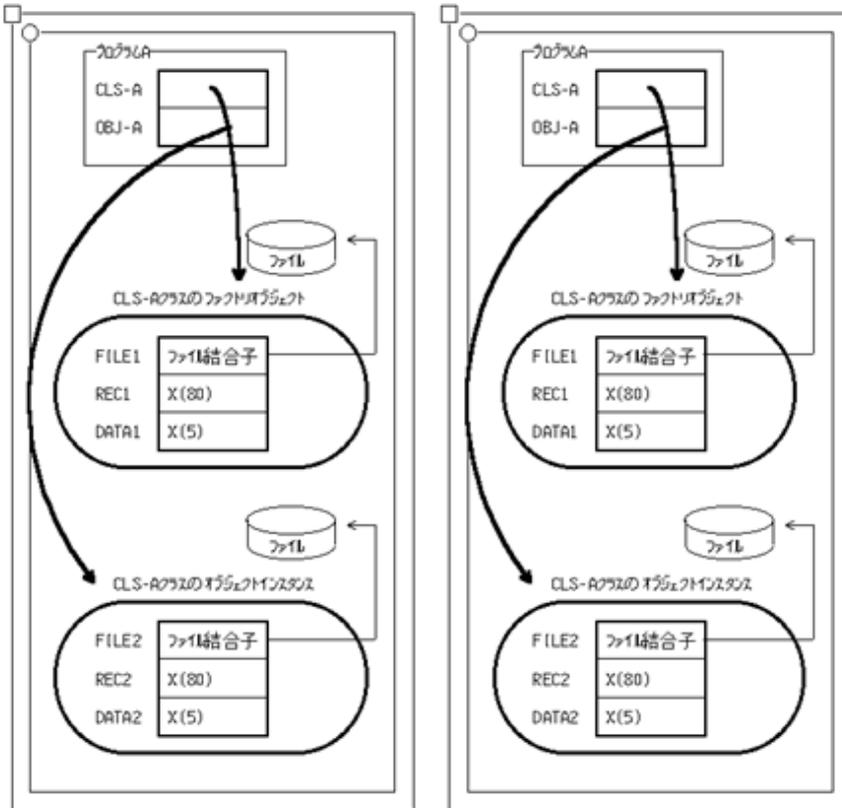
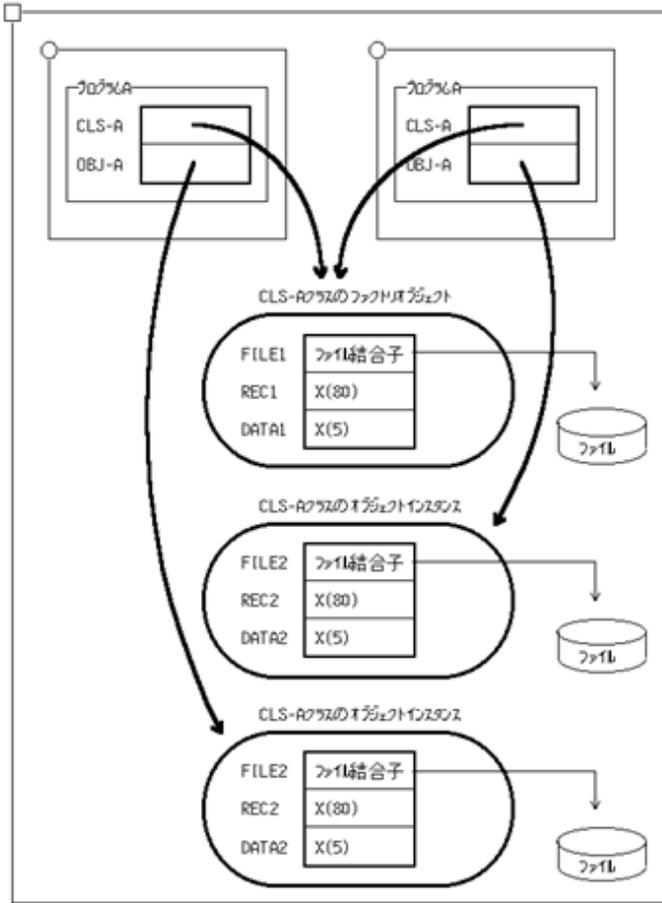


図20.7 マルチスレッドプログラム



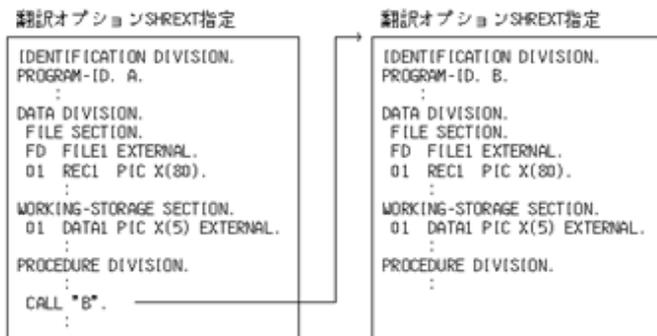
### 20.3.2.3 メソッド定義に宣言されたデータ

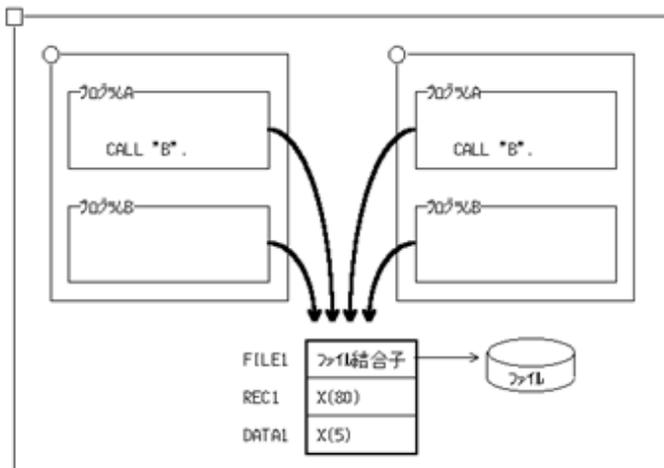
メソッド定義に宣言されたデータの割り付けは、シングルスレッドプログラムと変わりありません。つまり、メソッドの呼び出し時に確保され、そのメソッドの呼び出し元に戻るときに解放されます。メソッドの呼び出し元に戻るとき、クローズされていないファイルは強制的にクローズされます。

### 20.3.2.4 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。





### 20.3.3 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。

#### 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。

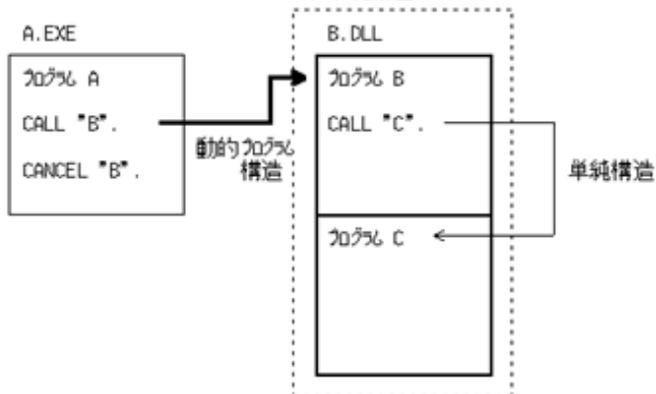


#### 例

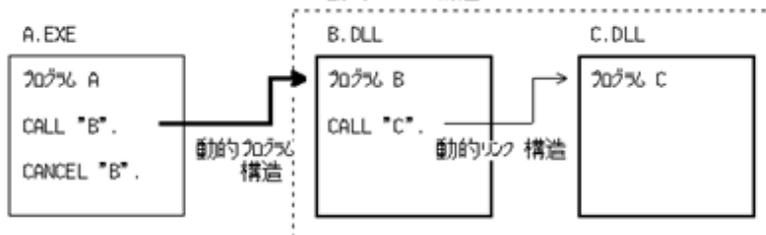
プログラムBとプログラムCを翻訳オプションTHREAD(MULTI)で翻訳します。

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。

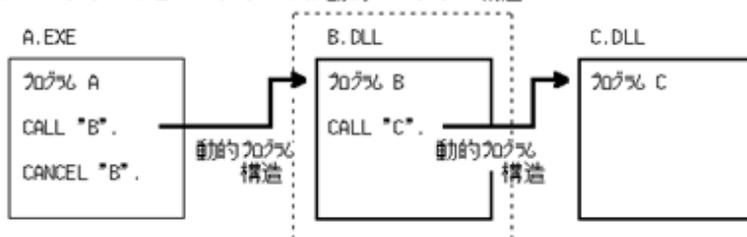
(a) プログラム B とプログラム C が単純構造



(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



□ : 選択オプション  
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

## 20.4 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。

### 20.4.1 競合状態

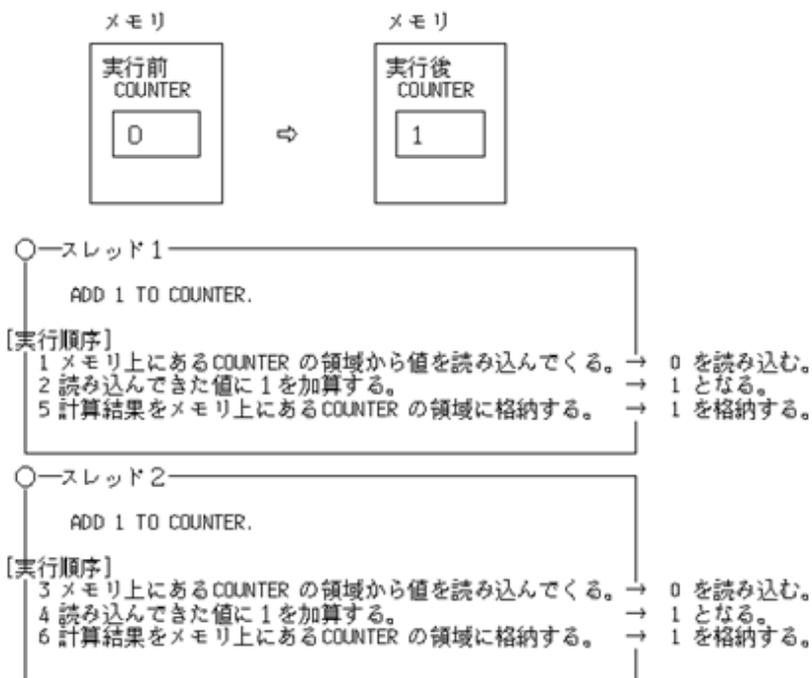
ここでは、スレッド間で資源を共有する場合に、一般的に発生する競合状態について説明します。

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。



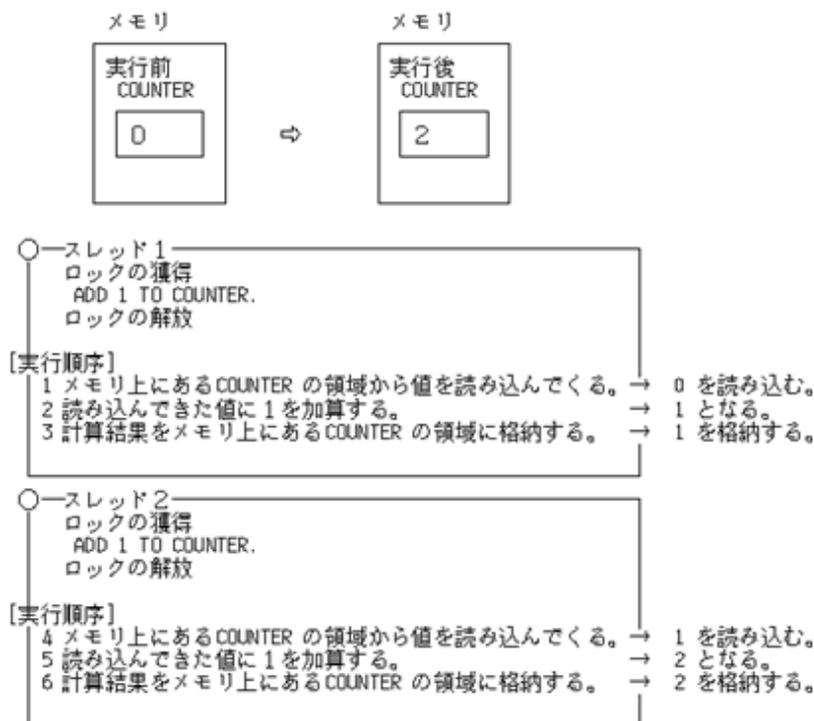
このような競合状態が発生するのは、次の条件のときです。もちろん、同一データをすべてのスレッドが参照するだけなら、同時にデータをアクセスしても問題ありません。

- 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)。



## 20.4.2 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、“[20.9 スレッド同期制御サブルーチン](#)”を参照してください。

### 20.4.2.1 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、“[20.6.1.1 スレッド間共有外部ファイル](#)”を参照してください。

```

○初期化スレッド
WORKING-STORAGE SECTION.
01 総販売数 PIC 9(9) COMP-5 EXTERNAL.
01 総売上 PIC 9(9) COMP-5 EXTERNAL.
PROCEDURE DIVISION.
MOVE 0 TO 総販売数.
MOVE 0 TO 総売上.
] [1]

```

```

○データ更新スレッド3
○データ更新スレッド2
○データ更新スレッド1
WORKING-STORAGE SECTION.
01 総販売数 PIC 9(9) COMP-5 EXTERNAL.
01 総売上 PIC 9(9) COMP-5 EXTERNAL.
01 販売数 PIC 9(9) COMP-5.
01 売上 PIC 9(9) COMP-5.
01 LOCK-KEY PIC X(30) VALUE "TOTAL".
01 WAIT-TIME PIC S9(9) COMP-5 VALUE -1.
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. ... [2]
ADD 販売数 TO 総販売数.
ADD 売上 TO 総売上.
] [3]
CALL "COB_UNLOCK_DATA" WITH C LINKAGE
USING BY REFERENCE LOCK-KEY
BY REFERENCE ERR-DETAIL
RETURNING RET-VALUE. ... [4]

```

- [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、“[I.2.7 COB\\_LOCK\\_DATA\(データロック獲得サブルーチン\)](#)”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。

## 20.4.2.2 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に行います(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。

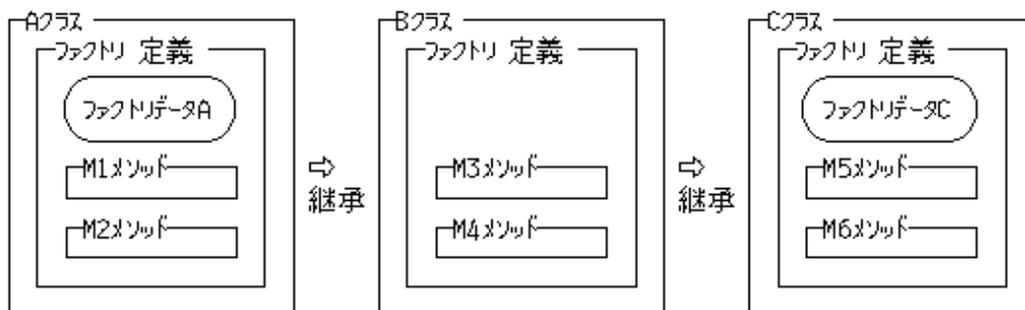
しかし、下の例のようにメソッドの複数回の呼び出しで1つの処理が完結するような場合は、オブジェクトロックサブルーチンを使用して、スレッドの同期制御を行う必要があります。オブジェクトロックサブルーチンは、オブジェクト単位で同期制御を行うことができます。オブジェクトのロックを獲得したスレッドは、ロックを解放するまで、そのオブジェクトを所有できます。



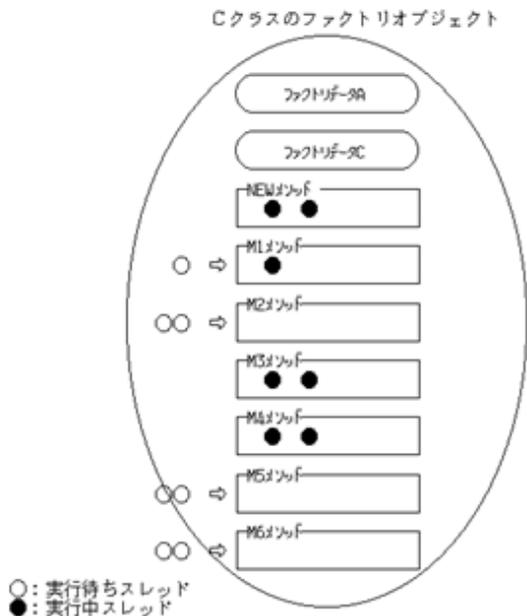
- [1] EMPLOYEEクラスのファクトリデータを複数のスレッドで同時に更新しないようにするため、オブジェクトロックサブルーチンを使用して、ファクトリオブジェクトのロックを獲得します。オブジェクトロックサブルーチンについては、“[1.2.8 COB\\_LOCK\\_OBJECT\(オブジェクトロック獲得サブルーチン\)](#)”を参照してください。
- [2] EMPLOYEEクラスのファクトリオブジェクトのプロパティメソッドを呼び出し、保険額をファクトリデータに設定しています。
- [3] EMPLOYEEクラスのファクトリオブジェクトのプロパティメソッドを呼び出し、契約年をファクトリ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もちます。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。



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。

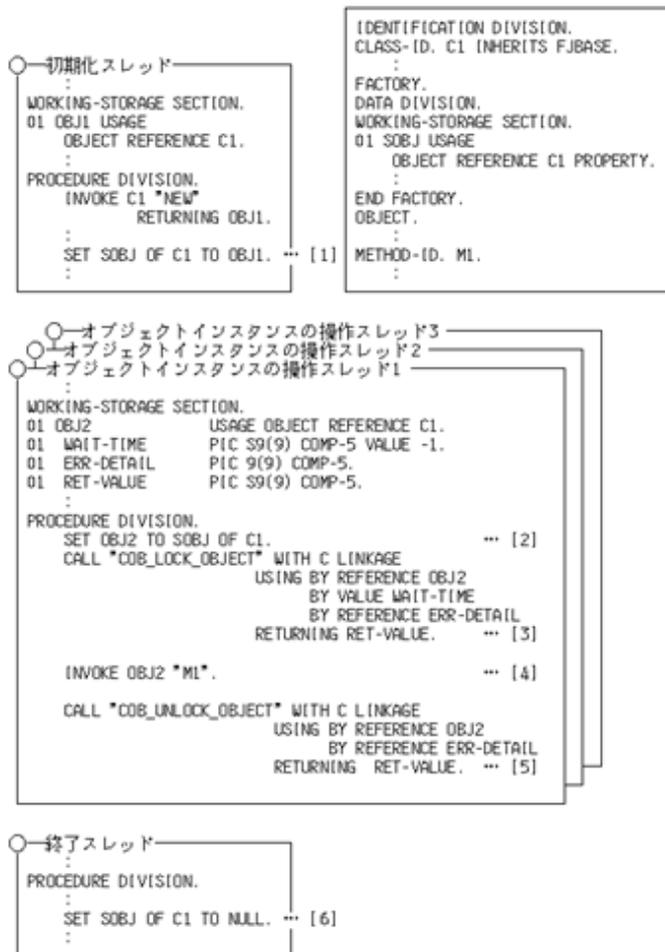
### 参考

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ると、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。

## 20.4.2.3 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、“[1.2.8 COB\\_LOCK\\_OBJECT\(オブジェクトロック獲得サブルーチン\)](#)”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。

## 20.5 基本的な使い方

“[20.3.2 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、“[20.7.1 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、“[20.4 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。

## 20.5.1 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、別々のファイル进行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使用して、ファイルを共有することができます。

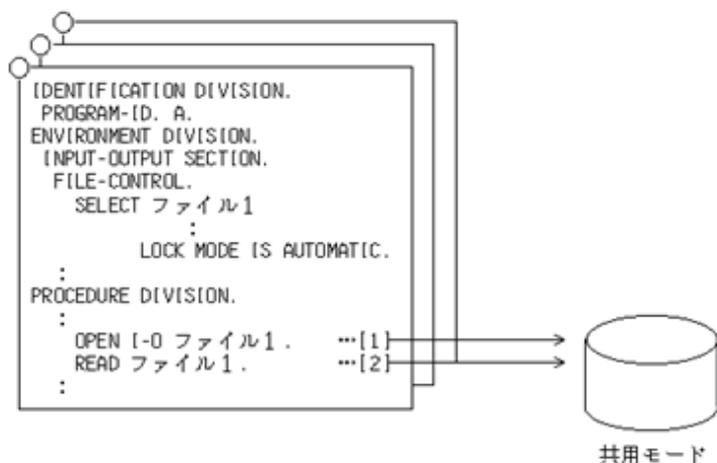
ここでは、内部属性を持つファイル結合子を使用して同一ファイルを共有する方法について説明します。

#### 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进行操作することで、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。

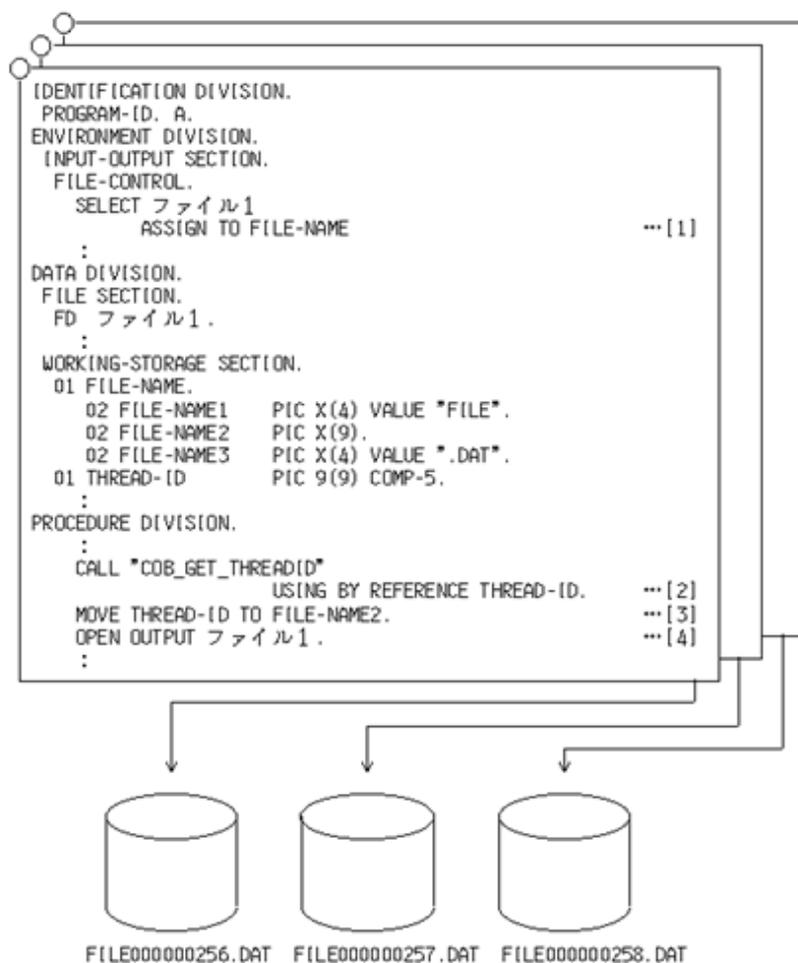
シングルスレッドプログラムと同様に、同一ファイル进行操作する場合は、ファイルの排他制御に従って処理してください。ファイルの排他制御については、“[7.7.2 ファイルの排他制御](#)”を参照してください。

### 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、“[20.6 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、ファイルを作成する。

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、別々のファイル进行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、“[1.2.6 COB\\_GET\\_THREADID\(スレッドID取得サブルーチン\)](#)”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変化します。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、“[H.3.8 ファイル削除関数 \(COB\\_FILE\\_DELETE\)](#)”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。

## 20.5.2 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]“[15.2.8.1.2 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]“[15.2.8.2 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]“[15.2.8.1.2 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]“[15.2.8.2 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]“[15.2.8.1.2 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。

## 20.5.3 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。



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“[20.7 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]“[C.2.49 @CBR\\_SYMFOWARE\\_THREAD\(Symfoware連携でマルチスレッド動作可能にする指定\)](#)”



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。

## 20.5.4 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。

### 20.5.4.1 小入出力機能について

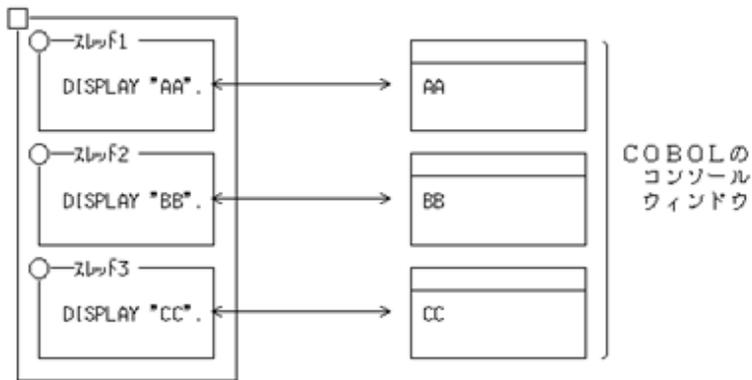
マルチスレッドモードでは、COBOLのコンソールウィンドウはスレッドごとに別々に表示されます。これに対して、システムのコンソールウィンドウ(コマンドプロンプトウィンドウを含む)およびファイルを使用した小入出力では、プロセスで1つのウィンドウおよびファイルを共有します。

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
ファイル	プロセス	プロセス	プロセス	プロセス

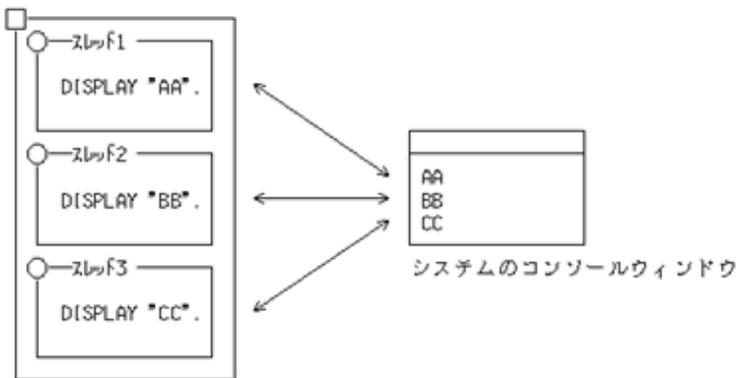


入出力の例

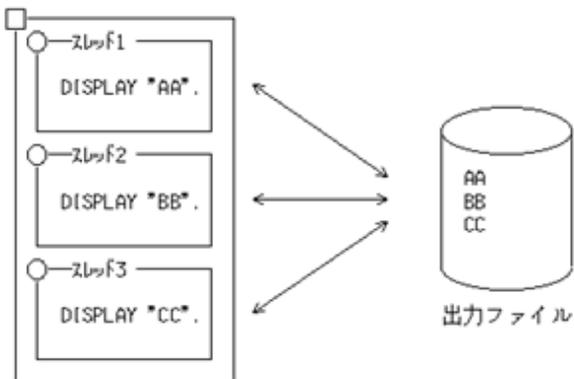
- COBOLのコンソールウィンドウを使用する場合



- 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。

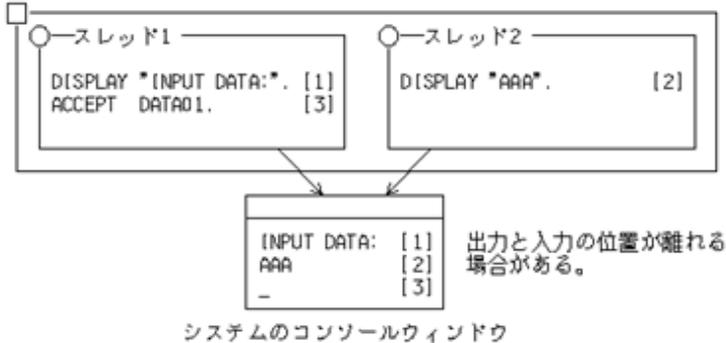
しかし、文単位での同期制御は行いますが、各文の実行順序については、システムのスレッド制御の順序に依存するため、結果が実行のたびに異なることがあります。

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]“[20.9 スレッド同期制御サブルーチン](#)”

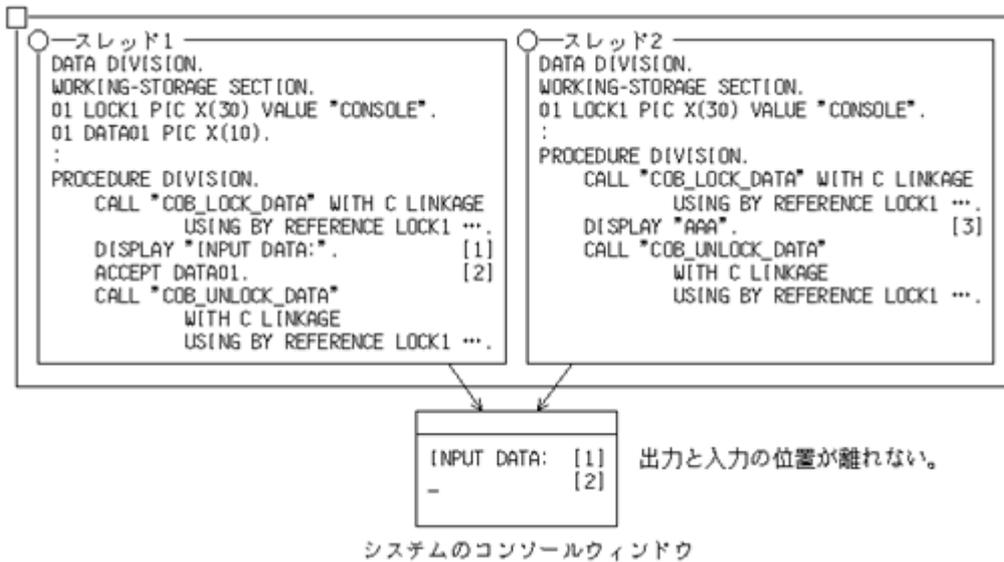
## 例

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。

## 20.5.4.2 コマンド行引数および環境変数の操作機能について

### 20.5.4.2.1 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。

### 20.5.4.2.2 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。

## 20.5.5 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。

## 20.6 少し進んだ使い方

### 20.6.1 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。

## 20.6.1.1 スレッド間共有外部ファイル

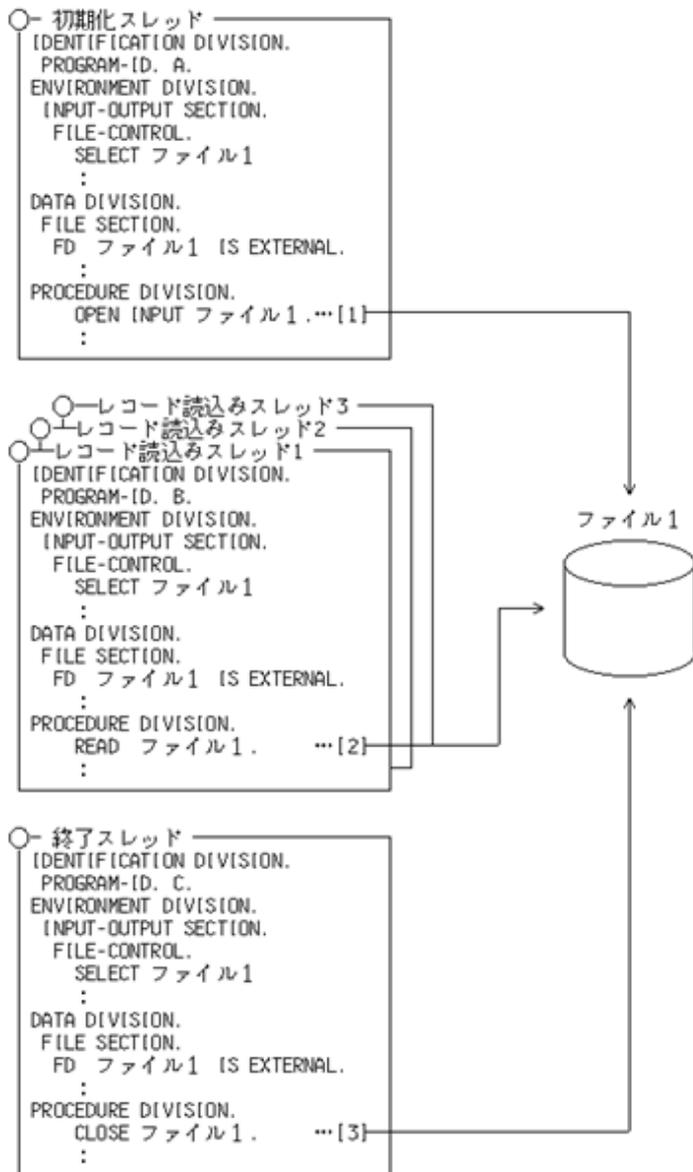
ファイル記述項にEXTERNAL句を指定した場合、ファイル結合子に外部属性が与えられます。外部属性を持つファイル結合子は、プログラム間で同じファイル結合子を共有することができます。

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

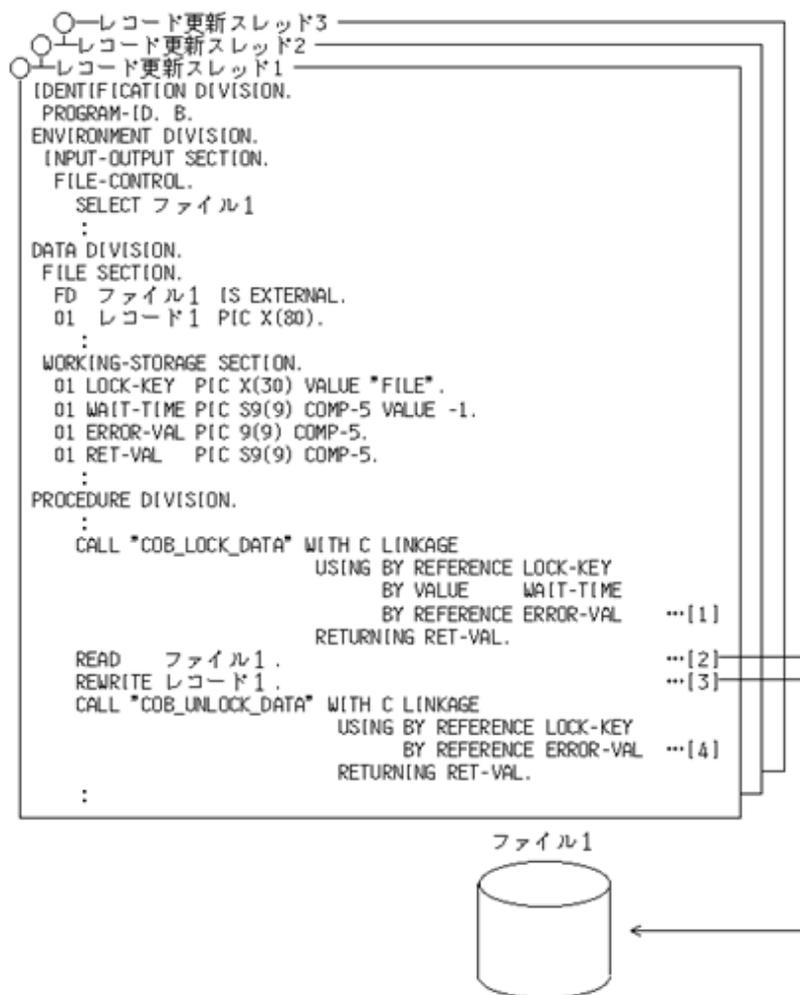


- [1] 初期化スレッドは、実行環境で最初に1回だけ起動されます。初期化スレッドでは、外部属性のファイル結合子を持つファイルを開きます。
- [2] レコード読みスレッドは、複数(プログラム例では3つ)同時に起動されます。レコード読みスレッドでは、初期化スレッドによって開かれているファイルに対し、レコードを読み込みます。
- [3] 終了スレッドは、実行環境で最後に1回だけ起動されます。終了スレッドでは、初期化スレッドによって開かれているファイルを閉じます。

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

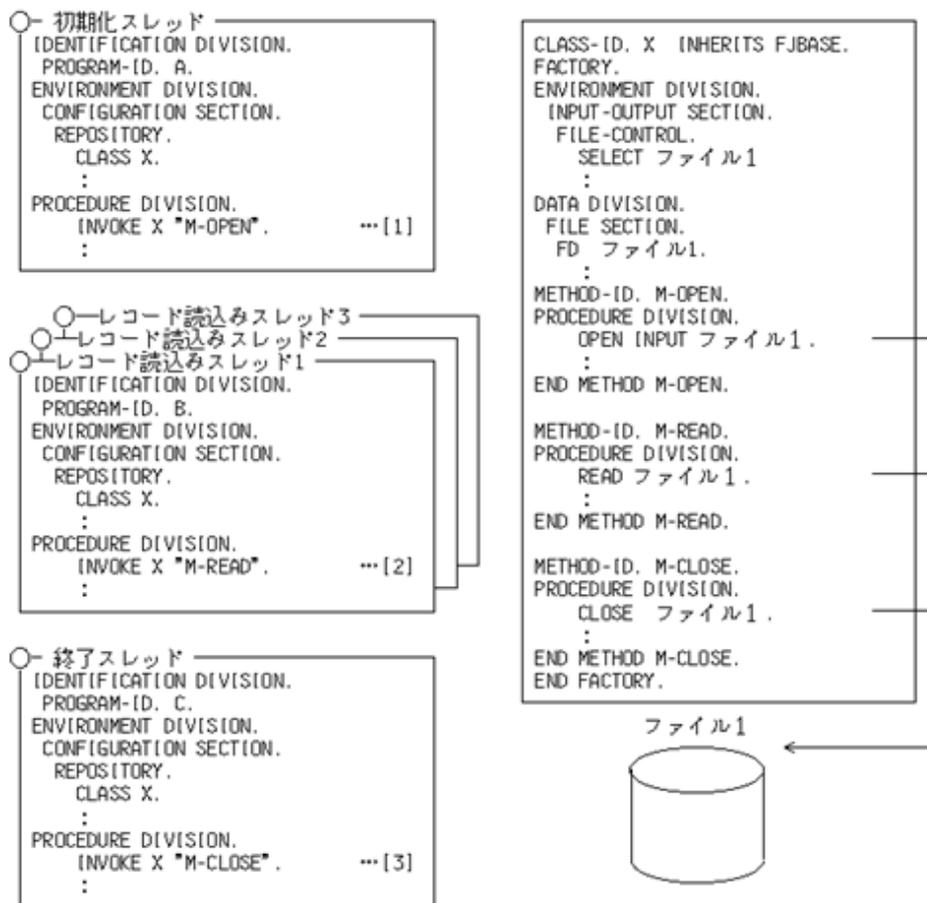
データロックサブルーチンについては、“[1.2.7 COB\\_LOCK\\_DATA\(データロック獲得サブルーチン\)](#)”を参照してください。

## 20.6.1.2 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] レコード読み込みスレッドは、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。

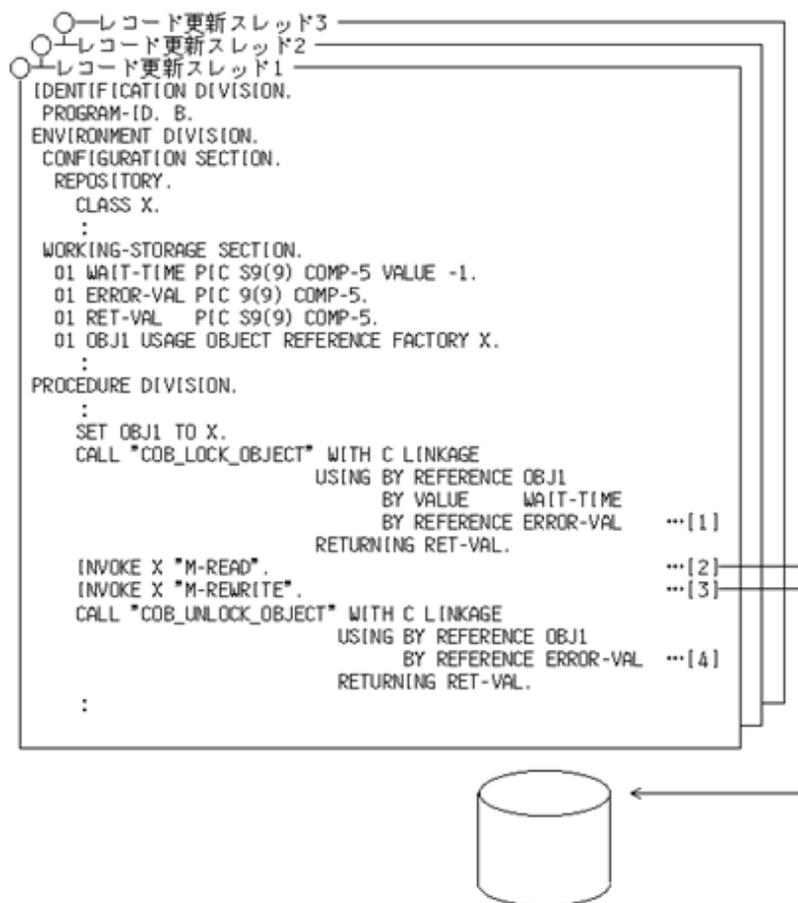
.....

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、“[I.2.8 COB\\_LOCK\\_OBJECT\(オブジェクトロック獲得サブルーチン\)](#)”を参照してください。

## 20.6.1.3 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を操作する場合、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]
:

```

```

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.

```

```

○ 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]
:

```

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、“[I.2.8 COB\\_LOCK\\_OBJECT\(オブジェクトロック獲得サブルーチン\)](#)”を参照してください。

## 20.6.2 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。

### 20.6.2.1 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。

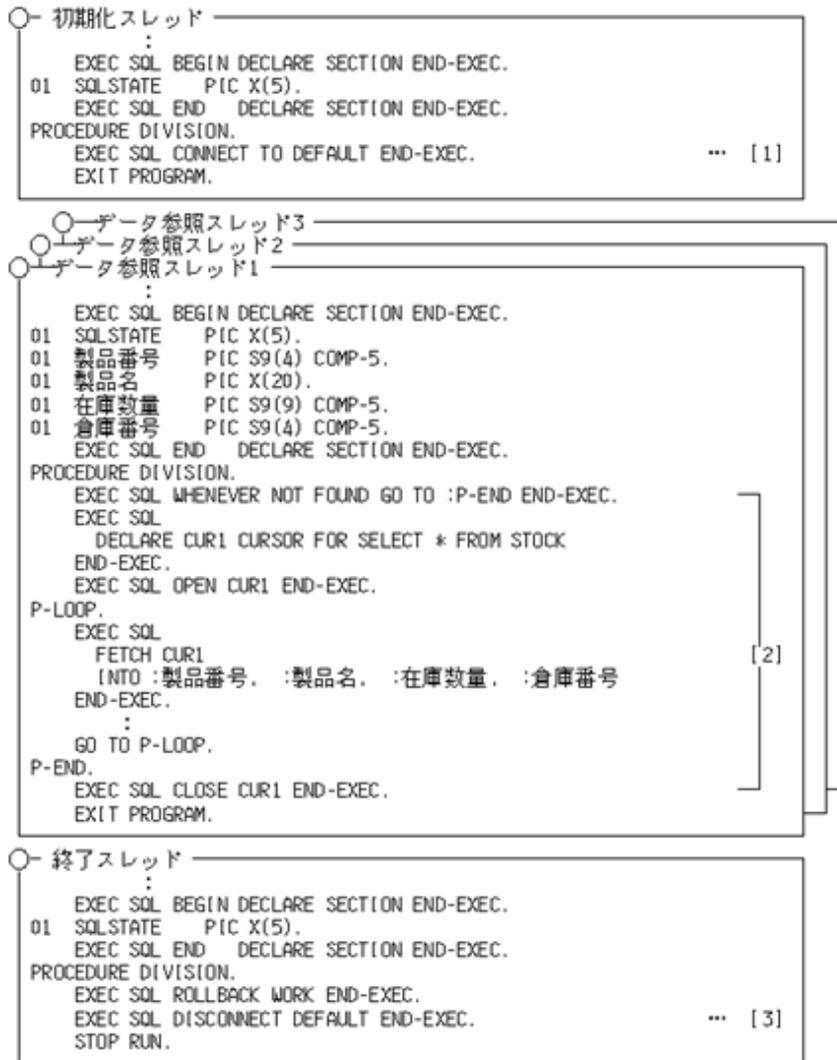
以降では、データベース表を変更しない(参照だけ行う)例、およびデータベース表を更新する例を説明します。

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]“[15.2.3.1 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20.8 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20.9 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20.9 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]“[15.2.8.1.2 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]“[15.2.8.2 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。

### 20.6.2.2 注意事項

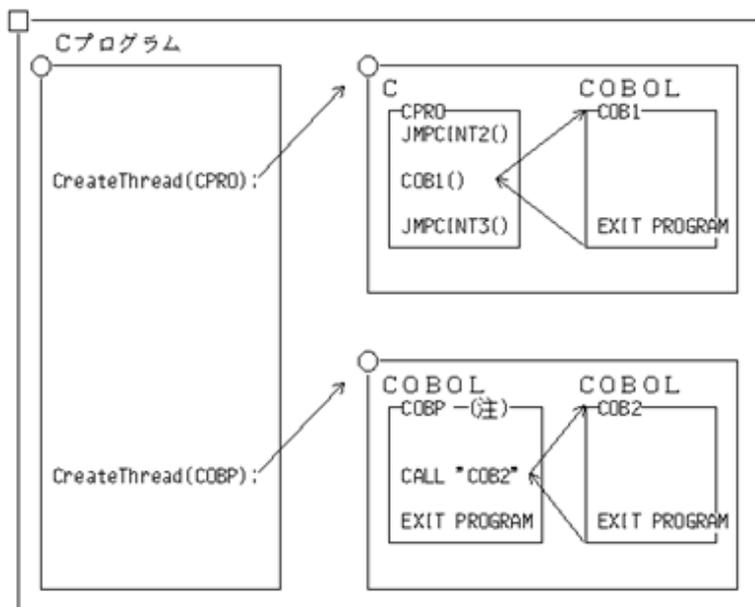
詳細については、“[20.5.2 リモートデータベースアクセス\(ODBC\)の利用](#)”の“[注意事項](#)”を参照してください。

## 20.6.3 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。

### 20.6.3.1 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。

### 20.6.3.2 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。

### 20.6.3.3 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。

### 20.6.3.4 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 STDCLL 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 STDCALL LINKAGE
USING PRM.
:
IF エラー発生
THEN MOVE 99 TO PROGRAM-STATUS
```

### 20.6.3.5 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STDCALL 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 STD CALL 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 STDCALL 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(COBTHD3.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. COBTHD3.
DATA DIVISION.
LINKAGE SECTION.
01 PRM3 PIC S9(9) COMP-5.
PROCEDURE DIVISION WITH STDCALL 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STDCALL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COBTHD1を翻訳する

```
COBOL32 -WC, "THREAD (MULTI)" COBTHD1.COB
```

### COBOLプログラムCOBTHD2を翻訳する

```
COBOL32 -WC, "THREAD (MULTI)" COBTHD2.COB
```

### COBOLプログラムCOBTHD3を翻訳する

```
COBOL32 -WC, "THREAD (MULTI)" COBTHD3.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、“[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COBTHD1を結合する

```
LINK COBTHD1.OBJ F3BICBDM.OBJ F3BICIMP.LIB KERNEL32.LIB /DLL /ENTRY:COBDMAIN /OUT:COBTHD1.DLL
```

COBTHD1.OBJ : COBOLプログラムCOBTHD1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COBTHD2を結合する

```
LINK COBTHD2.OBJ F3BICBDM.OBJ F3BICIMP.LIB KERNEL32.LIB /DLL /ENTRY:COBDMAIN /OUT:COBTHD2.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

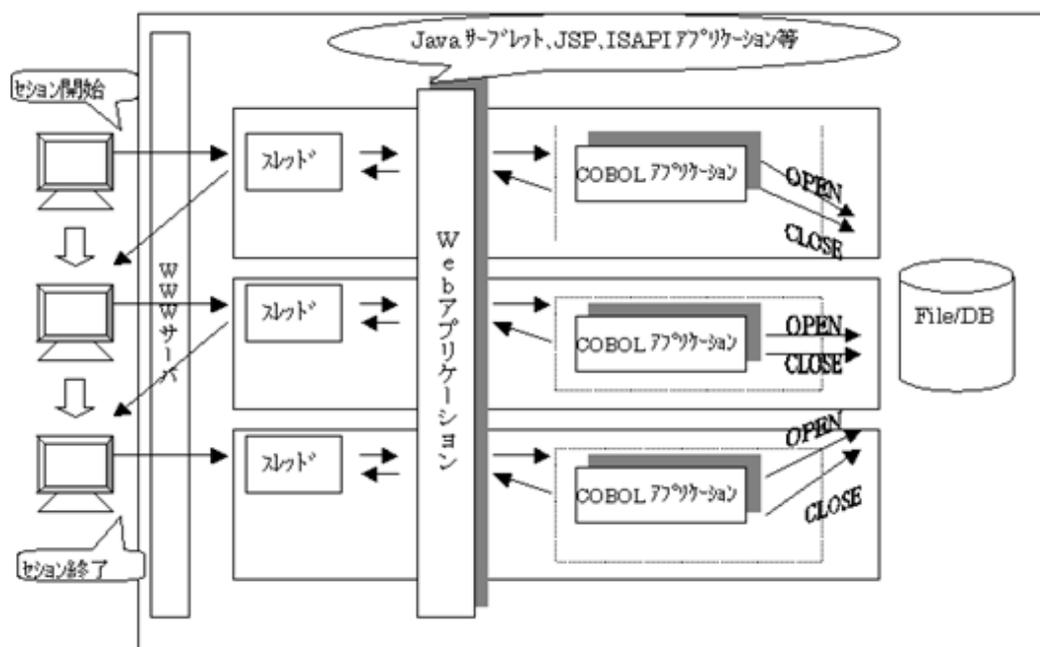
## 20.6.4 スレッド間で実行単位の資源を引き継ぐ方法

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。

### 20.6.4.1 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。

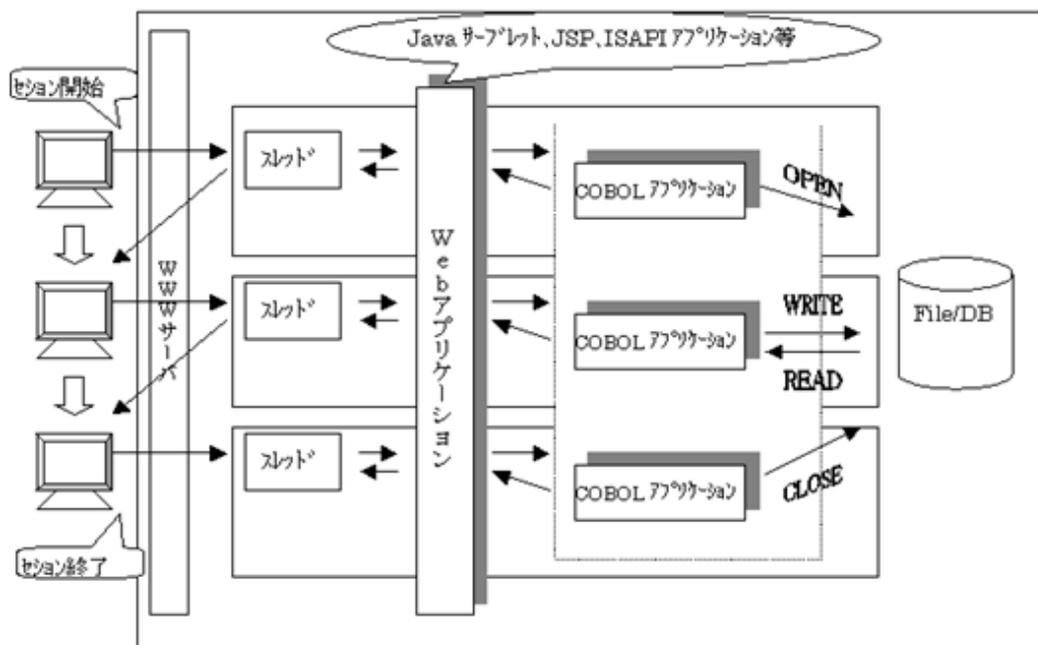
[実行単位を引き継がない場合]



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。

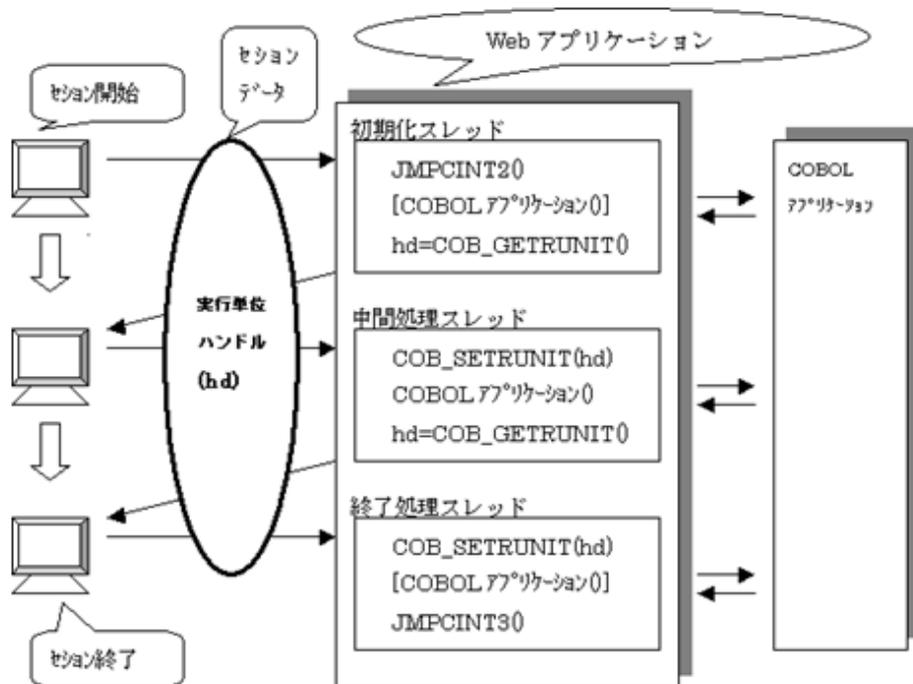
このような場合に、COBOLの実行単位のハンドルを返却するサブルーチンと、その実行単位のハンドルをCOBOLランタイムシステムに通知するサブルーチンを使用して、スレッド間で実行単位を引き継ぐことができます。引き継ぐ実行単位のハンドルとセッションの関連付けは、Cookieなどを用いた管理機構によって、COBOLアプリケーションの呼出し側で行う必要があります。

[実行単位を引き継ぐ場合]



## 20.6.4.2 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を呼び出すことで、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。

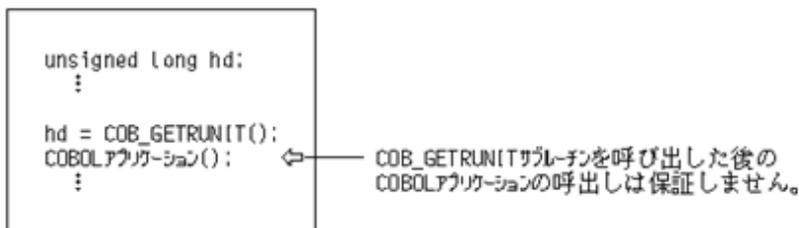


別スレッドで開設された実行単位の資源(ファイル結合子、DBカーソル、作業場所節に記述したデータなど)を引き継ぐことにより、クライアントからの呼出しごとにオープン処理からクローズ処理までを行わなければならなかったファイル操作のオーバーヘッドなどが削減でき、処理の効率化も図ることができます。

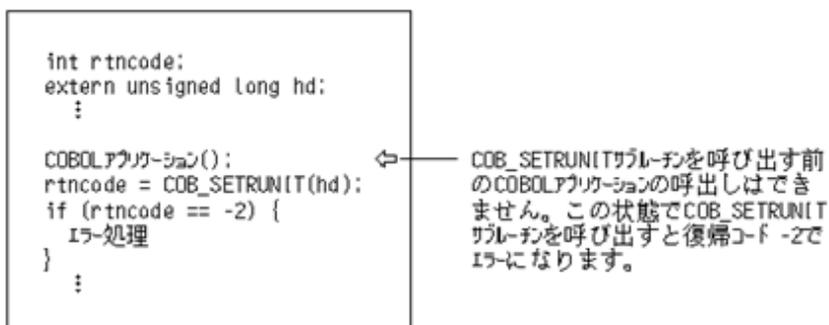
COBOLの実行単位のハンドルを返却するサブルーチンおよび実行単位のハンドルをCOBOLランタイムシステムに通知するサブルーチンの詳細は、“[1.3.1 COB\\_GETRUNIT\(COBOL実行単位ハンドル取得サブルーチン\)](#)”および“[1.3.2 COB\\_SETRUNIT\(COBOL実行単位ハンドル設定サブルーチン\)](#)”を参照してください。

## 20.6.4.3 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。



- 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、“[21.5.2 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。

## 20.6.4.4 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。

## 20.7 翻訳から実行までの方法

### 20.7.1 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。

#### 20.7.1.1 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、“[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:COBDMAIN /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

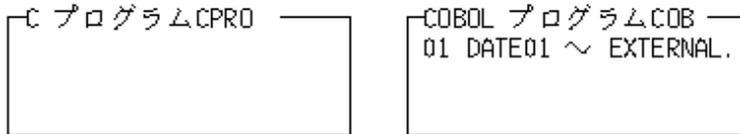


注意

LINKオプション/ENTRYに、必ずCOBDMAINを指定してください。

## 20.7.1.2 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、“[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

## 20.7.2 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。

### 20.7.2.1 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)

ここでは、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、“[付録C 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。

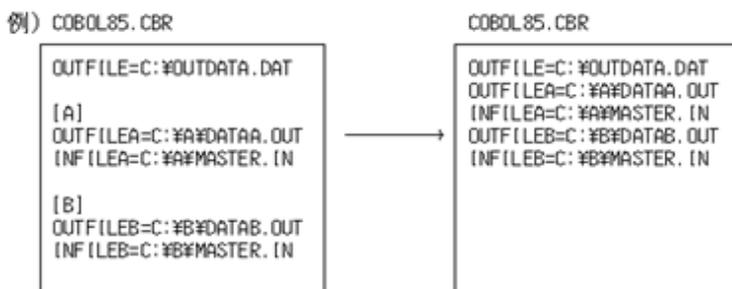
なお、ここで設定された内容は、プロセス内で共有されるため、どのスレッドでも有効となります。

## 実行用の初期化ファイルの記述例

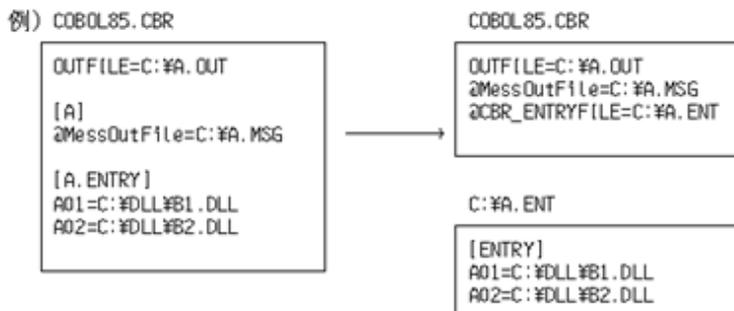
<pre>@MessOutFile=C:\MESSAGE.TXT @CnsIWinSize=(80, 24) @CnsIBufLine=100 @WinCloseMsg=ON @IconName=COB85EXE @CBR_ENTRYFILE=C:\TEST001.ENT</pre>
--

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。



- ・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。



- ・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。

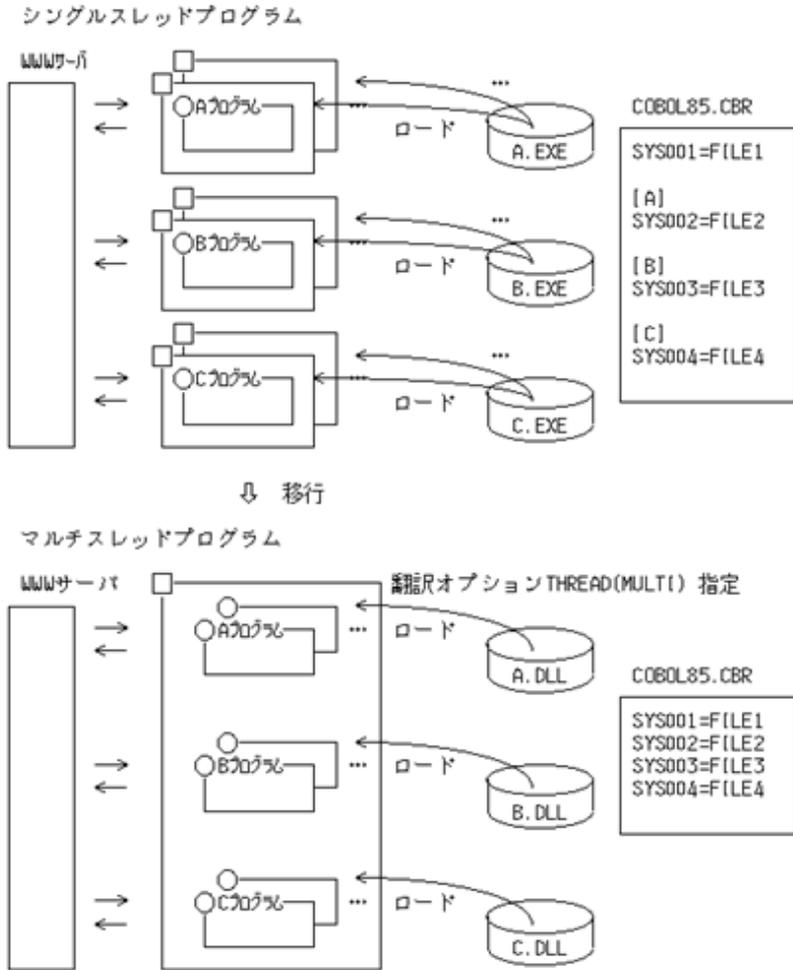
	シングルスレッドモード	マルチスレッドモード
セクション	有効	無効

	シングルスレッドモード	マルチスレッドモード
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



### 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。

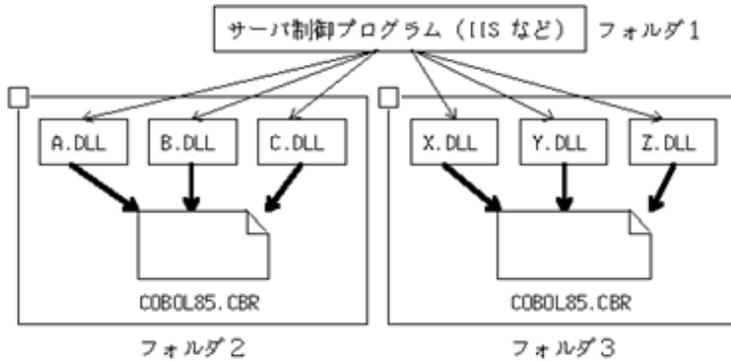
なお、DLLの格納位置からのCOBOL85.CBRファイルの読み込みを行うためには、DLLエントリオブジェクトをDLLに結合する必要があります。[参照]“5.5.4 DLL配下にある実行用の初期化ファイルを使用する”



## 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。



### 20.7.2.2 マルチスレッドモードでだけ有効な実行環境情報

以下は、マルチスレッドモードでだけ有効な環境変数情報です。

- “C.2.49 @CBR\_SYMFOWARE\_THREAD(Symfoware連携でマルチスレッド動作可能にする指定)”
- “C.2.52 @CBR\_THREAD\_MODE (スレッドモードの指定)”
- “C.2.53 @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)”
- “C.2.48 @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)”

## 20.8 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。

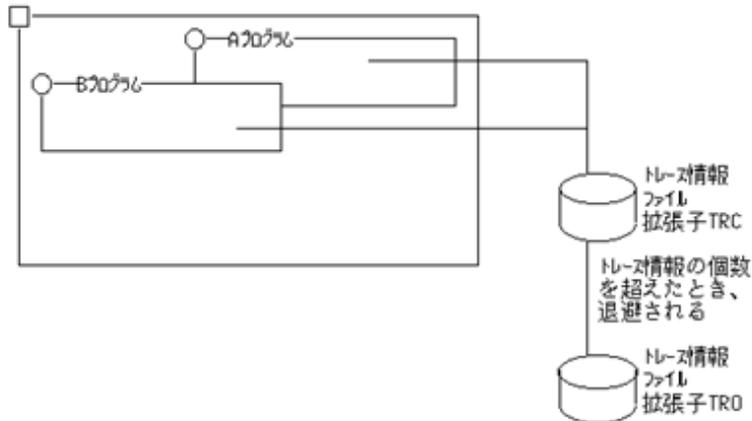
それぞれの機能に分けて、注意事項を説明します。

### 20.8.1 TRACE機能

トレース情報の内容に変更はありません。[参照]“21.4 TRACE機能”

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。



Trace information view is explained in the following diagram.

```

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] Program assigned thread ID

Program A's thread ID is 10C. Program B's thread ID is 11E.

[2] Exception message

The exception occurred in Program B with thread ID 11E.

[3] Thread end message

The thread ended in Program A with thread ID 10C.

From the above results, the following can be determined.

- Thread ID 10C of Program A operated normally.
- Thread ID 11E of Program B had an exception at line 26 of the execution text.

参考

DISPLAY ... UPON SYSERR is used to output arbitrary data to trace information.

It is convenient for checking data migration used by the program.

In addition, when using DISPLAY ... UPON SYSERR, the thread ID is also output together. Specify YES to the environment variable @CBR\_SYSERR\_EXTEND (SYSERR output information extension). [Reference] "C.2.50 @CBR\_SYSERR\_EXTEND (SYSERR output information extension)"

## 注意

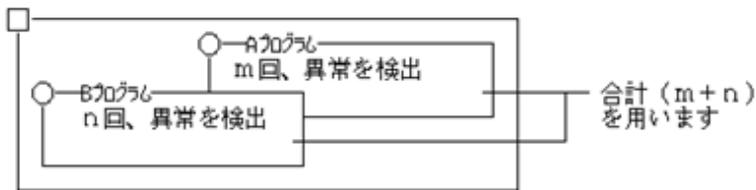
一度に多くのスレッドを実行する場合、大量のトレース情報が1つのファイルに書き込まれます。より多くのトレース情報が1つのファイルに出力されるように、トレース情報の個数(環境変数情報@GOPTのr指定)を調整してください。

### 20.8.2 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]“21.3 CHECK機能”

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。



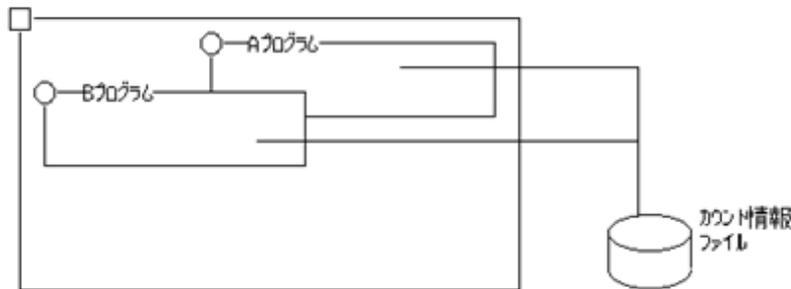
### 20.8.3 COUNT機能

カウント情報の内容に変更はありません。[参照]“21.5 COUNT機能”

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。

## 20.8.4 対話型デバッグ

詳細については、“[24.6.4 マルチスレッドでのデバッグの対応](#)”を参照してください。

## 20.8.5 NetCOBOL Studioのデバッグ機能

詳細は、“[NetCOBOL Studio ユーザーズガイド](#)”を参照してください。

## 20.9 スレッド同期制御サブルーチン

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。

詳細は、以下を参照してください。

- [I.6.1 データロックサブルーチン](#)
- [I.6.2 オブジェクトロックサブルーチン](#)

## 20.10 注意事項

### 20.10.1 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28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。

#### 20.10.1.1 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ため、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。

#### 20.10.1.2 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20.1 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

“20.10.1.1 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。

### 20.10.1.3 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“28.2.3 \*COMクラス”、“28.6.1 \*COM-ARRAYクラス”、“28.6.2 \*COM-EXCEPTIONクラス”を参照してください。

### 20.10.1.4 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、“17.1.2.2 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。

## 20.10.2 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、“20.3.1 実行環境と実行単位”を参照してください。

## 20.10.3 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。

## 20.10.4 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制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、“[20.6.1 入出力機能の利用](#)”を参照してください。
- MeFiをオープンした際に通知される表示ファイル識別子を別スレッドで使用してプリンタの入出力を行った場合、動作結果は保証されません。また、同様にプリンタのクローズを行った場合も、動作結果は保証されません。  
MeFiの印刷機能を利用する場合は、印刷機能をオープンしたスレッドでその帳票の入出力および、クローズを実行してください。  
COBOLプログラムからMeFiの印刷機能を利用する場合は、スレッド間で同じファイル結合子を操作してFORMAT句付き印刷ファイルまたは、表示ファイル(印刷)を共有する場合に、OPEN文を実行したスレッドと同じスレッドでCLOSE文が実行されるまでの間、別スレッドからファイル結合子を操作しないでください。
- 複数スレッドからのOLE2の印刷はしないでください。複数スレッドからのOLE2印刷を行った場合の動作結果は保証されません。

## 20.10.5 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5部 テスト支援機能／デバッグ支援機能

---

---

第21章 テスト支援機能.....	586
第22章 デバッグ支援機能.....	610

## 第21章 テスト支援機能

本製品で作成したプログラムをテストする手段として、以下の機能が用意されています。

- NetCOBOL Studioのデバッグ機能
- 対話型デバッガ機能
- 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)

### デバッグ機能を使用するために必要な翻訳オプション

機能名	使い方	翻訳オプション
NetCOBOL Studioの デバッグ機能	NetCOBOL Studioのデバッグ機能を使用して、作成したプログラムをデバッグします。 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。 <ul style="list-style-type: none"> <li>• データ変更での中断</li> </ul> <b>【用途】</b> プログラムを動作させながら、プログラムの論理的な誤りを検出したい場合	TEST
対話型デバッガ機能	対話型デバッガを使用して、作成したプログラムをデバッグします。 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。 <ul style="list-style-type: none"> <li>• データ変更での中断</li> <li>• 実行監視条件での中断</li> <li>• 特定実行条件での中断</li> <li>• バッチデバッグやデバッグ操作の再現</li> </ul> <b>【用途】</b> プログラムを動作させながら、プログラムの論理的な誤りを検出したい場合	TEST
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パラメタの数と長さが一致しているか</li> </ul> <b>【用途】</b> <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> </ul>	CHECK

機能名	使い方	翻訳オプション
	<ul style="list-style-type: none"> <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> </ul>	
TRACE機能	<p>次の情報を出力します。</p> <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</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> <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>	TRACE
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>破壊を検出したプログラム名またはメソッド名</li> <li>破壊を検出した場所(手続き部の開始/終了)</li> <li>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</li> </ul>	—

## 注意

TRACE機能とCOUNT機能は、同時に使用できません。

## 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]“[B.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。

## 21.1 NetCOBOL Studioのデバッグ機能

---

NetCOBOL Studioは、オープンソースの統合開発環境であるEclipseをベースに、COBOLプログラム開発支援機能を組み込んだCOBOL開発環境です。

NetCOBOL Studioのデバッグ機能を使用して、ブレークポイントの設定、プログラム中断時のデータ項目の内容確認など、プログラムの動作検証ができます。

詳細は、“NetCOBOL Studio ユーザーズガイド”の“デバッグ機能”を参照してください。

## 21.2 対話型デバッガ機能

---

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

詳細は、“[第24章 対話型デバッガの使い方](#)”を参照してください。

## 21.3 CHECK機能

---

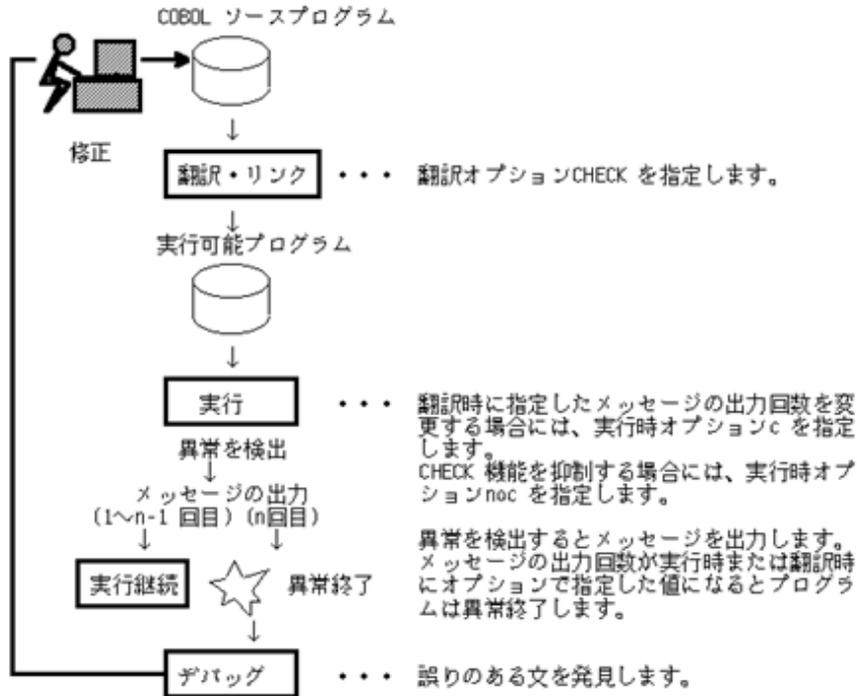
CHECK機能では、以下の検査を行い、異常を検出するとメッセージを出力し、異常終了します。そのため、プログラムの誤動作を防ぐことができます。

- ・ 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

### 21.3.1 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。



## 21.3.2 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、“メッセージ集”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もありません。

以下にCHECK機能のメッセージについて説明します。

### 21.3.2.1 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:xxxxxxx TID:xxxxxxx] 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:xxxxxxx TID:xxxxxxx] 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:xxxxxxx TID:xxxxxxx] 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: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=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:xxxxxxx TID:xxxxxxx] 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:xxxxxxx TID:xxxxxxx] '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21.1 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 (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。

## 21.3.2.2 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.3.5 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)
- nocp: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](#)”

## 21.3.3 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。

JMP08211-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。

JMP08221-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。

JMP08281-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。

JMP08291-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なければなりません。

14: JMN3414I-S 'SUB2' を呼ぶCALL文にはRETURNING指定を記述しなければなりません。プログラム' SUB2' のPROCEDURE DIVISIONにRETURNING指定があります。

15: JMN3508I-S 'SUB1' を呼ぶCALL文にはRETURNING指定を記述することはできません。プログラム' SUB1' のPROCEDURE DIVISIONにRETURNING指定がありません。

16: JMN3335I-S CALL文のUSING指定またはRETURNING指定に記述したパラメタ' P2' の長さは、プログラム' SUB1' のPROCEDURE DIVISIONのUSING指定またはRETURNING指定に記述したパラメタ' L1' の長さとも一致していなければなりません。

17: JMN33341-S CALL文のUSING指定またはRETURNING指定に記述したパラメタ'P3'の型は、プログラム'SUB3'のPROCEDURE DIVISIONのUSING指定またはRETURNING指定に記述したパラメタ'L-OR1'の型と一致していなければなりません。  
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文を実行するとき以下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.
```

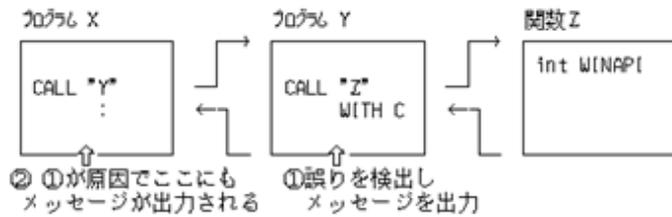
## 21.3.4 注意事項

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。



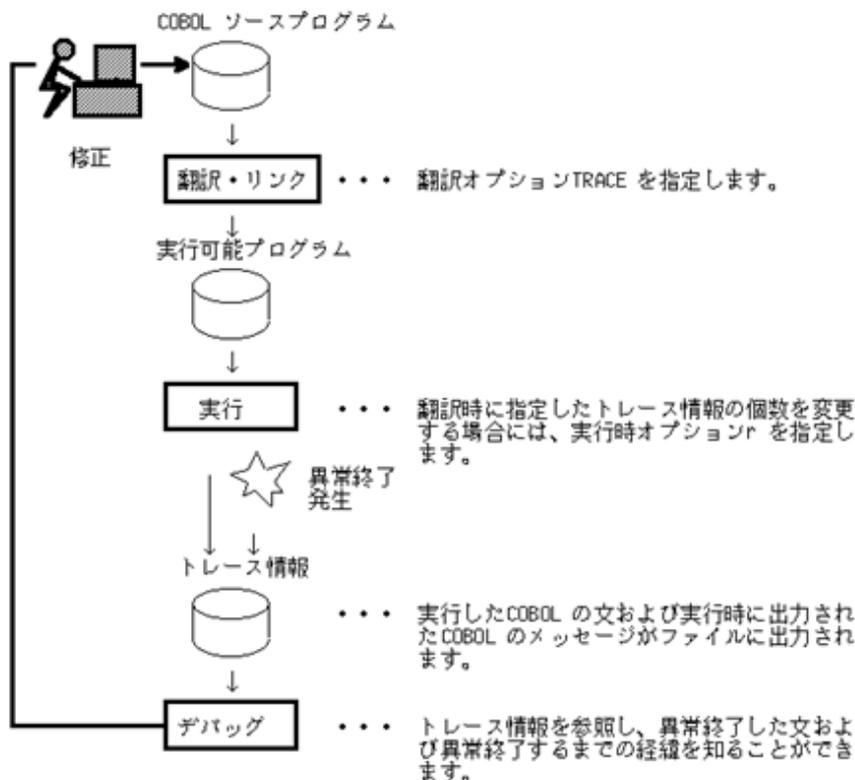
- CHECK(PRM)は、プログラム名として一意名を指定したCALL文によって内部プログラムを呼出す場合は検査しません。
- CHECK(PRM)で外部プログラム呼出しのパラメタを検査する場合、呼出し元および呼出し先のプログラムの両方をNetCOBOL V7.2L10以降のコンパイラで翻訳オプションCHECK(PRM)を指定し、翻訳する必要があります。他言語で作成されたプログラムを呼び出すCALL文、または、他言語で作成されたプログラムから呼び出された場合、パラメタの検査は行われません。
- CHECK(PRM)の外部プログラム呼出しの検査で、呼出し元のパラメタ個数と呼出し先のパラメタ個数の差が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。

## 21.4 TRACE機能

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。

### 21.4.1 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。



## 21.4.2 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.3.62 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”



### 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]“C.2.54 @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 xxxxxxxxxx xx xxxxxxxxxx. [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、“メッセージ集”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U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]“[C.2.55 @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
--------------------------------------

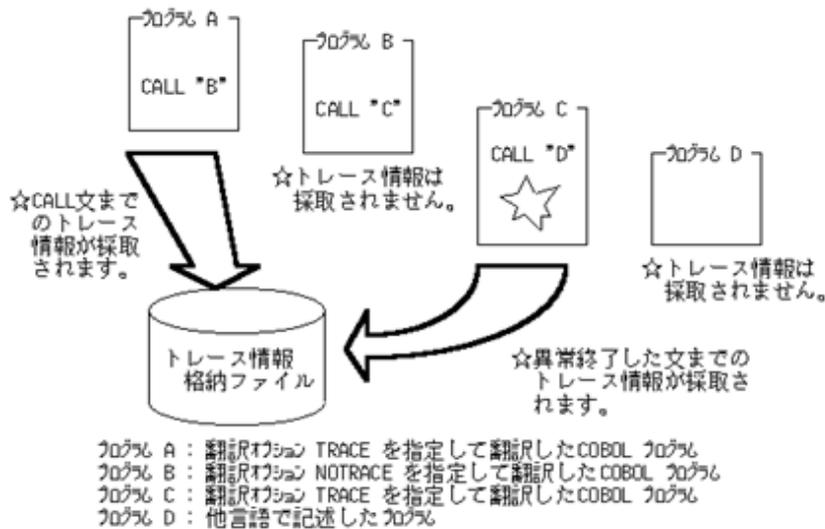
トレース情報ファイル名(一世代前の情報)

SAMPLE-00000EC4_20100112_104850. TRO
--------------------------------------

### 21.4.3 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。



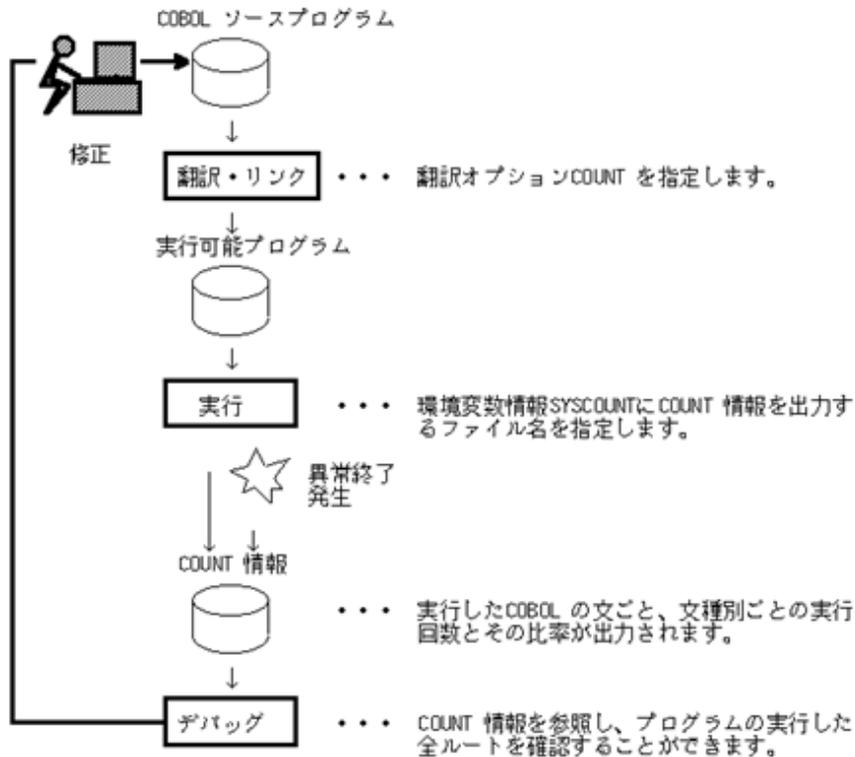
- TRACE機能では、トレース情報の採取など、COBOLプログラムで記述した以外の処理を行います。そのため、TRACE機能使用時には、プログラムのサイズが大きくなり、実行速度も遅くなります。TRACE機能は、デバッグ時にだけ使用し、デバッグ終了後は、翻訳オプションTRACEを指定しないで再翻訳してください。
- TRACE機能が有効な場合、プログラムの正常終了または異常終了にかかわらず、トレース情報は生成されます。
- トレース情報ファイルが存在している状態で、再度プログラムを実行した場合、元のトレース情報ファイルの内容は失われます。
- トレース情報ファイルが不要になった場合には、削除してください。
- NetCOBOL Studioのデバッグ機能または対話型デバッガと併用する場合、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。

## 21.5 COUNT機能

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。

### 21.5.1 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。



## 21.5.2 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.3.9 COUNT (COUNT機能の使用の可否)”、“C.2.87 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 (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 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]	[16]	[17]	[18]	[19]	[20]						
METHOD-NAME	ACTIVE VERB	TOTAL VERB	PERCENTAGE (%)	EXECUTION COUNT	PERCENTAGE (%)						
COUNT-METHOD	2	2	100.0000	2	100.0000						
-----											
	2	2	100.0000	2							

[21]

PROGRAM/CLASS/PROTOTYPE METHOD EXECUTION COUNT						
[22]	[23]	[24]	[25]	[26]	[27]	
PROGRAM/CLASS			PERCENTAGE		PERCENTAGE	

/METHOD-NAME	ACTIVE VERB	TOTAL VERB	(%)	EXECUTION COUNT	(%)
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方法名を表示します。

[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

### 21.5.3 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。

### 21.5.4 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
	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。

- デバッグ機能とCOUNT機能を同時に使用すると、COUNT情報が正しく出力されない場合があります。デバッグ機能を使用する場合は、COUNT機能を使用しないでください。

## 21.6 メモリチェック機能

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。

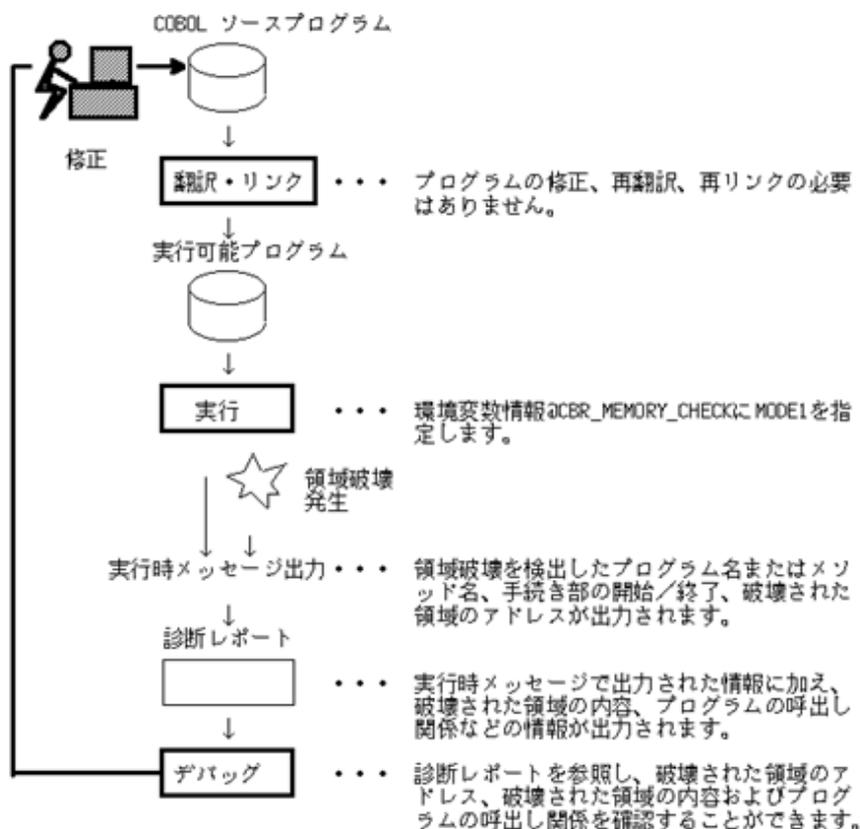
- JMP0009I-U ライブラリ作業域が確保できません。(注)
- JMP0010I-U ライブラリ作業域が破壊されています。
- アプリケーションエラー(アクセス違反)が発生した場合

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。“[5.9.2 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](#)”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]“[C.2.40 @CBR\\_MEMORY\\_CHECK\(メモリチェック機能を使って検査を行う指定\)](#)”

### 21.6.1 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。



### 21.6.2 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。

## 21.6.2.1 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。

## 21.6.3 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、“22.1 診断機能”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 41 AAAAAAAAAAAAAAAAAA
00DC11A8 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
00DC11B8 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
00DC11C8 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
00DC11D8 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
00DC11E8 41 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 00 .....
00DC1208 00 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 00 ..... TMEM...
00DC1228 80 00 00 00 00 98 11 DC 00 A8 12 DC 00 46 00 00 00 ..... F...
00DC1238 00 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 00 .....
00DC1258 00 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 00 .....
00DC1278 00 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 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=0013AAFO EIP=77EED7A2 ESP=0012E278 EBP=0012E2CC 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>

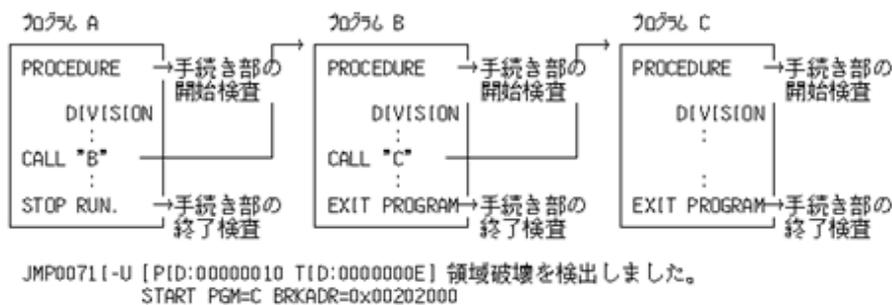
[ 1]-----

以降、省略

## 21.6.4 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。



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。

## 21.6.5 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。

### 21.6.5.1 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]“[22.1.3.3 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、“[22.1.4.3.4 領域破壊監視](#)”を参照してください。

## 21.6.6 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。

## 第22章 デバッグ支援機能

NetCOBOLには、次のデバッグ機能があります。

- ・ 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)

### デバッグ機能を使用するために必要な翻訳オプション

機能名	使い方	翻訳オプション
診断機能	次の情報を診断レポートに出力します。 <ul style="list-style-type: none"><li>・ エラーの種類(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> また、ダンプを出力します。 【用途】 <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	目的プログラムごとに次の情報を出力します。 <ul style="list-style-type: none"><li>・ オブジェクト内相対オフセット</li><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> 【用途】 <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>	LIST PRINT SOURCE COPY MAP

## 22.1 診断機能

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。

### 22.1.1 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。

- “表22.4 例外コードと例外名”に記述された例外
- 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ユーザーズガイド”を参照してください。

## 22.1.2 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。

### 22.1.2.1 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。

### 22.1.2.2 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22.1 COBOLプログラムと出力情報の関係”に示します。

表22.1 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

手順	詳細	[1]	[2]	[3]	[4]	[5]	[6]	
リンク	/DEBUG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	なし	あり	あり
		プログラム名+行番号	なし	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、[22.1.4.2.1 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22.1 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22.1 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22.1 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.3.29 LIST(目的プログラムリストの出力の可否)”、“A.3.42 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	XXXXXXXXXXXXXX	dc	"XXXXXXXX"
0000001B	XXXXXXXXXXXXXX	dc	"XXXXXXXX"
00000023	XXXXXXXXXXXXXX	dc	"XXXXXXXX"
0000002B	2B30303030	dc	"+0000"
00000030	2031	dc	" 1"
00000032	0000	db	0x0000
00000034		PROLOGUEAD:	
00000034	XXXXXXXX	db	0XXXXXXXXX : 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。

### 22.1.2.3 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が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
```

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”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”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、“[23.4.5.1 デバッグモジュールの作成方法](#)”を参照してください。

## 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”を指定しておくことをおすすめします。

### 22.1.2.4 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、“[22.1.2.2 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。

### 22.1.3 診断機能の起動

#### 22.1.3.1 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

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]“[C.2.36 @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 22.2 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
```

### 22.1.3.2 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22.2 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
```

### 22.1.3.3 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、“C.2.2 @CBR\_ATTACH\_TOOL (プログラムからデバッグ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、“21.6 メモリチェック機能”を参照してください。



注意

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。

### 22.1.3.4 起動パラメタ

診断機能の起動パラメタを“表22.2 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。

c. 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22.2 起動パラメタ

指定形式	a	b	c	内容
-d{YES   NO}	○	—	—	ダンプを出力するかどうかを指定します。 <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> 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。
-i{0   1   2}	○	○	—	出力内容を指定します。 <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> 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。
-o フォルダ名	○	—	—	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) 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。 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。 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。 標準の出力先フォルダについては、“ <a href="#">22.1.5.2 ダンプの出力先</a> ”を参照してください。
-r フォルダ名	○	○	○	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) 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。 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。 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。 標準の出力先フォルダについては、“ <a href="#">22.1.4.1 診断レポートの出力先</a> ”を参照してください。
-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。 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。 <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。 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。 アドレスは、先頭に'0x'を付けた16進数で指定します。 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“-l”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。

## 22.1.4 診断レポート

### 22.1.4.1 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。

```
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22.2 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22.3 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に対応

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“-I”を指定します。起動パラメタの詳細については、“表22.2 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。

## 注意

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“-I”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。

### 22.1.4.2 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。

#### 22.1.4.2.1 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22.4 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。

### — 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)

### — トップアドレス

スタックの最上位アドレス

### — 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@### または _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@### または _クラス名_SET_メソッド名@### または _クラス名_ENTRY_GET_メソッド名@### または _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 または 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には、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プログラムの場合には、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には、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
- 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22.4 例外コードと例外名”に示します。

表22.4 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み取りまたは書き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
	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。

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。

- 一 添字や指標の値が範囲外である。
- 一 部分参照で参照可能範囲外を参照する。
- 一 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]“21.3 CHECK機能”

## 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22.5 翻訳オプションと出力形式”に示します。

表22.5 翻訳オプションと出力形式

出力対象		NOOPTIMIZE指定	OPTIMIZE指定	
		NUMBER/NONNUMBER 指定共通	NUMBER指定	NONNUMBER指定
ソース ファイル 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)	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、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)

## 22.1.4.2.2 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します。

### 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

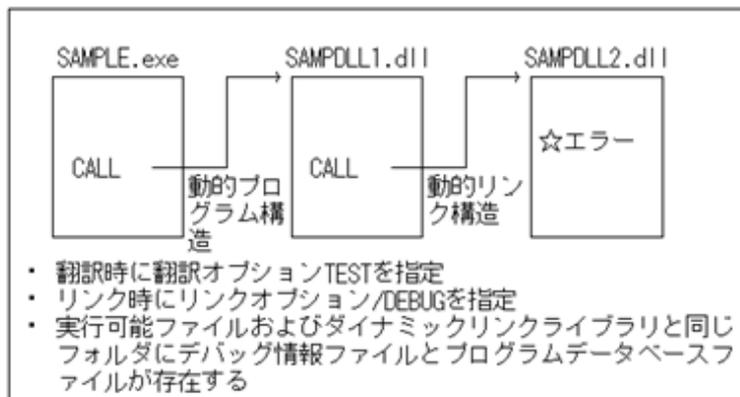
### 22.1.4.3 診断レポートの見方

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。

#### 22.1.4.3.1 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  
 : EDI=00405108 EIP=03BB125D ESP=0012FCD8 EBP=0012FD98 EFL=00010256  
 : GS=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

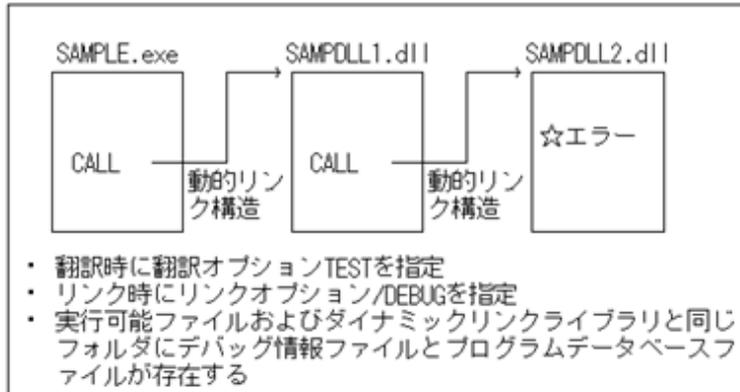
以降、省略

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。

## 22.1.4.3.2 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 : JMP0311I-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\Net\COBOL\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\Net\COBOL\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\Net\COBOL\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\F3B110.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\F3B110.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\F3B110.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

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、JMP031II-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には、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となります。

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。

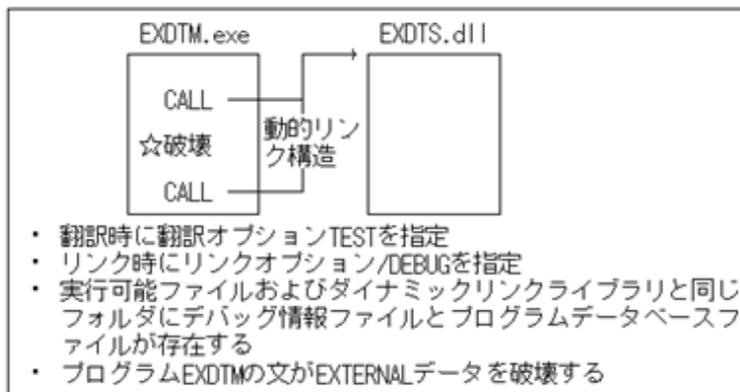
### 22.1.4.3.3 応答なし

1. <<スレッド情報>>に着目します。ここでは、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。

### 22.1.4.3.4 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 : JMP00711-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

1. 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。
2. 不当な書き込みかどうかは、実行時メッセージJMP00711-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JMP00711-Uが発生した場合の診断レポート(“[22.1.4 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JMP00711-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。

## 22.1.5 ダンプ

### 22.1.5.1 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UI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)などのデバッガ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。

### 22.1.5.2 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。

```
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22.2 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、ダンプ出力を抑止してください。起動パラメタの詳細については“[22.1.3.4 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。

- 以下のような場合、ダンプが出力されないことがあります。
  - COBOLプログラムがシステムによりローディングされている途中で異常終了した場合
  - 言語間結合で、COBOLプログラムが呼び出されず他言語で異常終了した場合
  - “表22.4 例外コードと例外名”に記述されている以外の例外が発生した場合

### 22.1.5.3 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。

## 22.1.6 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
  - Windows エラー報告

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。
  - Windows エラー報告
- 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ため、出力する行番号が正確でない場合があります。
- 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
  - “24.7 注意事項”のソースファイルの記述方法に関する項目
- 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、それらを読み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。

## 22.2 翻訳リストとデバッグツールを使ったデバッグ

---

ここでは、以下の翻訳リストとデバッグツールを使用して、異常終了時のデバッグを行う方法について説明します。

### 翻訳リスト

- － 目的プログラムリスト、ソースプログラムリスト  
障害発生箇所の特定に使用します。
- － データマップリスト  
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。

### デバッグツール

- － WinDbg (Debugging Tools for Windows - ネイティブ x86)  
Microsoft社が提供しているデバッグツールです。  
Debugging Tools for Windows - ネイティブ x86は、Microsoftのホームページで情報公開されています。
- － Windowsエラー報告  
クラッシュダンプはシステムのWER機能で採取します。

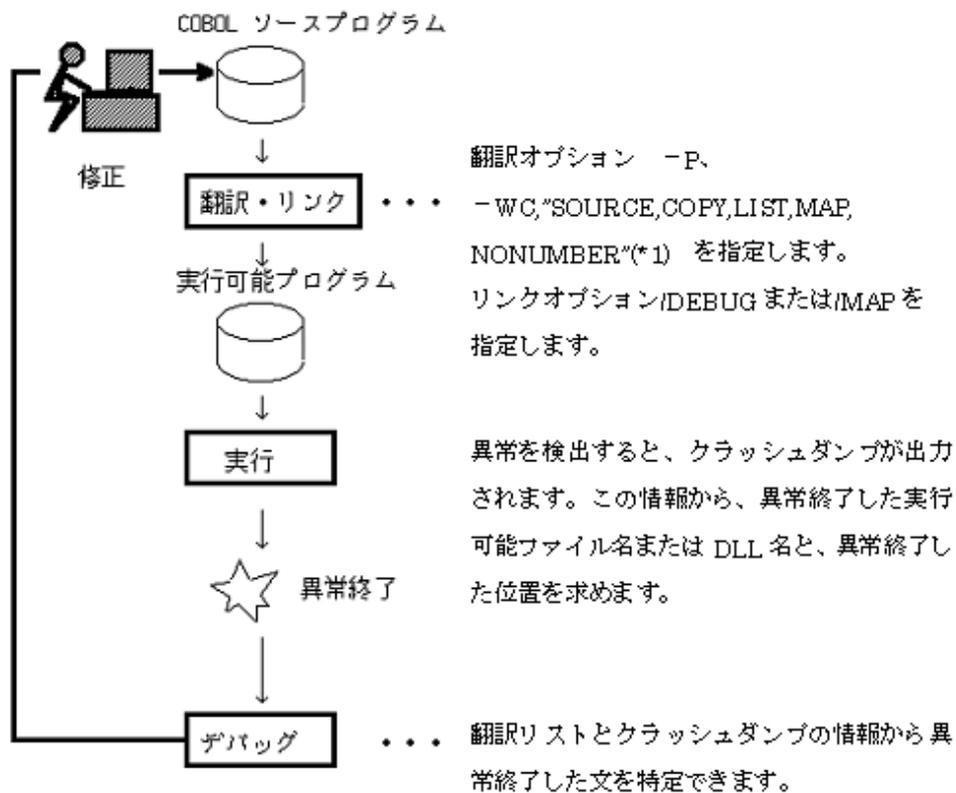
### その他のファイル

- － リンク時に出力される以下のファイルのどちらかひとつ。
  - プログラムデータベースファイル(.PDB) (/DEBUGオプション指定時に出力)
  - リンクマップファイル(.MAP) (/MAPオプション指定時に出力)
- － クラッシュダンプ
  - Windowsエラー報告機能により作成されたクラッシュダンプ

### 22.2.1 デバッグ作業の流れ

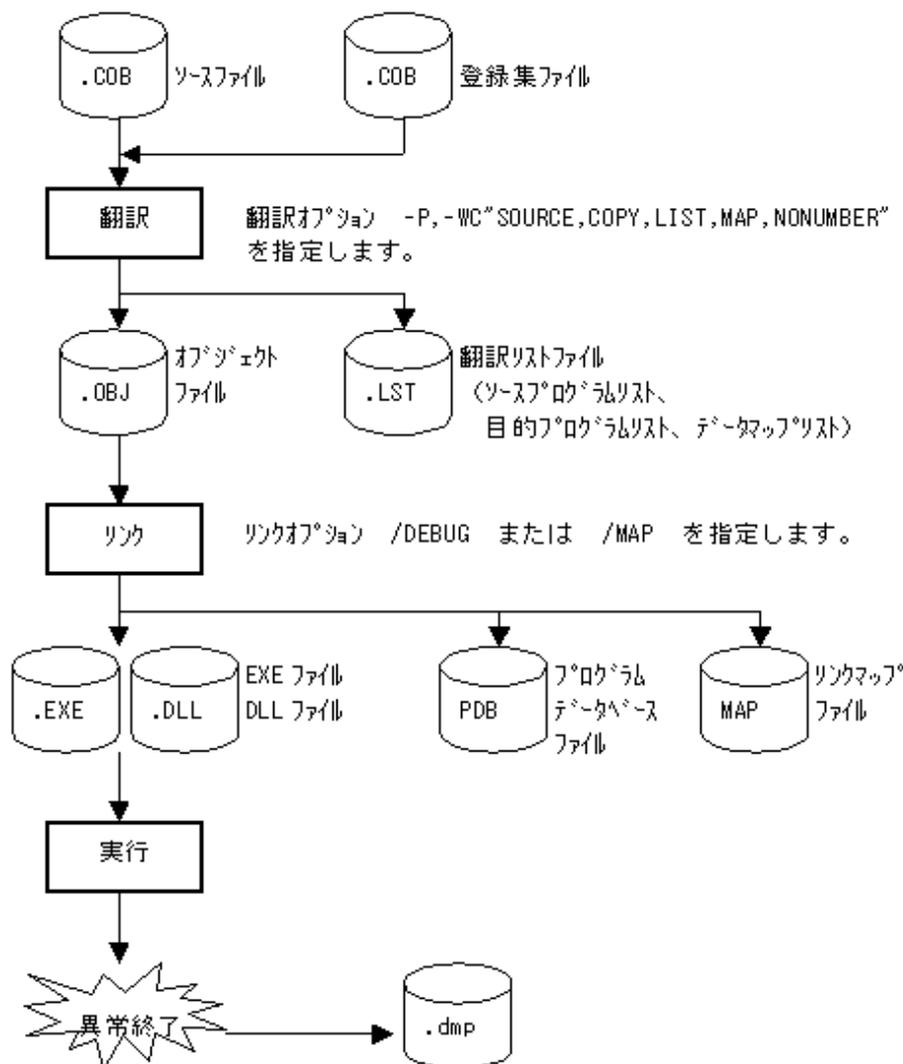
---

以下に翻訳リストとデバッグツールを使ったデバッグの流れを示します。



\*1: COBOLソースを翻訳する場合、NONUMBER(省略値)が有効な状態で行うことを推奨します。以降の説明は、NONUMBERが有効な状態で翻訳された場合を想定して記述されています。

以下にファイル関連図を示します。



## 注意

- クラッシュダンプを採取する方法については、“[22.1.5 ダンプ](#)”を参照してください。
- 異常終了時に参照できるデータは、異常終了したプログラムまたは異常終了したメソッドで使用するデータと、呼出し元のデータです。WinDbgの「Call Stack」に現れないプログラムおよびメソッドのデータは参照することはできません。
- 翻訳オプションのOPTIMIZEが有効な状態で翻訳されたプログラムまたはメソッドの場合は、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できない場合があります。
- データを確認するためには、データがレジスタに格納されている場合もあるため、アセンブラの知識が必要です。アセンブラの知識を使い、かつ、翻訳時に目的プログラムリストを出力しておくことにより、機械語命令でのデバッグを行うことができます。目的プログラムリストの形式については、“[B.5 目的プログラムリスト](#)”を参照してください。
- 内部プログラムは正しく「Call Stack」に出力されません。正しい呼出し関係を出力する必要がある場合は、NetCOBOLのデバッグ機能を使用して同じ状況を再現させてください。

### 22.2.2 障害発生箇所の特定方法(プログラムデータベースファイル使用時)

リンク時に“/DEBUG”オプションを指定してプログラムデータベースファイル(PDBファイル)を作成し、そのPDBファイルが参照できる状態(注)でプログラムが異常終了した場合の障害発生箇所の特定方法について説明します。

注:PDBファイルが参照できる状態とは、以下の状態をいいます。

- 開発環境の場合は、PDBファイルが、リンク時に出力されたフォルダにある状態をいいます。
- 運用環境の場合は、PDBファイルがEXEファイルまたはDLLファイルと同じフォルダにある状態をいいます。

ここでは、簡単な例を用いて説明します。

- LISTDBG.COB

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. LISTDBG.
000030*
000040 DATA DIVISION.
000050 WORKING-STORAGE SECTION.
000060 01 計算.
000070 02 被除数 PIC 9(2).
000080 02 除数 PIC 9(2).
000090 01 答え PIC 9(2).
000100 PROCEDURE DIVISION.
000120 MOVE 2 TO 被除数.
000130*
000140 CALL "DIVPROC" USING 計算 RETURNING 答え.
000150*
000160 DISPLAY "答え =" 答え.
000170 EXIT PROGRAM.
000180 END PROGRAM LISTDBG.
```

- DIVPROC.COB

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. DIVPROC.
000030*
000040 DATA DIVISION.
000050 WORKING-STORAGE SECTION.
000060 LINKAGE SECTION.
000070 01 引数.
000080 02 被除数 PIC 9(2).
000090 02 除数 PIC 9(2).
000100 01 答え PIC 9(2).
000110*
000120 PROCEDURE DIVISION USING 引数 RETURNING 答え.
000130*
000140 MOVE 0 TO 答え.
000150 COMPUTE 答え = 被除数 / 除数.
000160*
000190 EXIT PROGRAM.
000200 END PROGRAM DIVPROC.
```

- 使用するファイル

- クラッシュダンプ
- 翻訳オプションSOURCE、COPY、LISTを指定して出力した翻訳リスト
- プログラムデータベースファイル(.PDB)

- 使用するデバッグツール

- WinDbg (Debugging Tools for Windows - ネイティブ x86)

1. 以下の方法で、クラッシュダンプが調査対象のものか確認してください。

1. クラッシュダンプのファイル名を確認する。

Windowsエラー報告機能で採取したクラッシュダンプは、ファイル名がクラッシュしたアプリケーション名になっています。クラッシュダンプのファイル名が調査対象のアプリケーション名であるか確認します。

2. 発生時間を確認する。

クラッシュダンプの更新日時が調査対象のアプリケーションの異常終了した時間であるか確認します。

2. WinDbgからクラッシュダンプを開いて、例外発生箇所と理由を特定します。

1. WinDbgを起動します。

2. WinDbgの「File」メニューから「Open Crash Dump...」を選択し、調査対象のクラッシュダンプを開きます。

[開いた直後のWinDbgの出力]

```
Loading Dump File [C:\ProgramData\Fujitsu\NetCOBOL\COBSNAP\LISTDBG_1720_20140801-155635_cobsnap.dmp]
User Mini Dump File with Full Memory: Only application data is available

Symbol search path is: *** Invalid ***
*****
* Symbol loading may be unreliable without a symbol search path.          *
* Use .symfix to have the debugger choose a symbol path.                  *
* After setting your symbol path, use .reload to refresh symbol locations. *
*****
Executable search path is:
Windows 7 Version 7601 (Service Pack 1) MP (2 procs) Free x86 compatible
Product: WinNt, suite: SingleUserTS
Machine Name:
Debug session time: Fri Aug 1 15:56:35.000 2014 (UTC + 9:00)
System Uptime: 0 days 4:35:31.089
Process Uptime: 0 days 0:00:01.000
.....
*** ERROR: Symbol file could not be found. Defaulted to export symbols for ntdll.dll -

***** Symbol Loading Error Summary *****
Module name      Error
ntdll            The system cannot find the file specified

You can troubleshoot most symbol related issues by turning on symbol loading diagnostics (!sym noisy) and
repeating the command that caused symbols to be loaded.
You should also verify that your symbol search path (.sympath) is correct.
This dump file has an exception of interest stored in it.
The stored exception information can be accessed via .ecxr.
(1720.1108): Integer divide-by-zero - code c0000094 (first/second chance not available) ... [1]
*** WARNING: Unable to verify checksum for LISTDBG.exe
*** ERROR: Symbol file could not be found. Defaulted to export symbols for LISTDBG.exe -
eax=00000002 ebx=7ffdf000 ecx=00000000 edx=00000000 esi=00000000 edi=00406100
eip=00401516 esp=0012fd74 ebp=0012fe34 iopl=0         nv up ei ng nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010283
LISTDBG!DIVPROC+0x20e:                                     ... [3]
00401516 f7f9             idiv   eax,ecx                                           ... [2]
```

[1]から、例外の発生理由は“Integer divide-by-zero - code c0000094”(ゼロ除算)であることがわかります。

[2]から、例外の発生位置は“00401516”の“idiv eax,ecx”命令であることがわかります。

“idiv eax,ecx”命令は、符号付き割算を行う命令であり、edx:eaxレジスタの内容をecxレジスタの内容で割り、商をeaxに、余りをedxに設定します。

rコマンドにより、レジスタの内容を確認します。

[rコマンド出力結果]

```

0:000> r
eax=00000002 ebx=7ffdf000 ecx=00000000 edx=00000000 esi=00000000 edi=00406100
eip=00401516 esp=0012fd74 ebp=0012fe34 iopl=0         nv up ei ng nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010283
LISTDBG!DIVPROC+0x20e:
00401516 f7f9             idiv   eax,ecx

```

上記の出力から、“edx:eax=0000000000000002”を“ecx=00000000”で割算を行おうとしたが、ecxが0だったため、ゼロ除算が発生したことがわかります。

### 3. COBOLのソース行位置を特定する。

[3]の“LISTDBG!DIVPROC+0x20e”から、COBOLソース上の位置を求めます。

[3]の情報は、“モジュール名!シンボル名+オフセット”の形式で出力されています。この例では、以下のようになります。

```

モジュール名      : "LISTDBG"
シンボル名        : "DIVPROC"
シンボルからのオフセット : 0x20e

```

モジュール名は、拡張子が削除されています。拡張子を調べる場合は、lmコマンドを使用して確認します。

[lmコマンド出力結果]

```

0:000> lm v m LISTDBG
start  end      module name
00400000 0040a000 LISTDBG C (export symbols) LISTDBG.exe
Loaded symbol image file: LISTDBG.exe
Image path: C:\work\%tmp%\tmp\LISTDBG.exe          ... [4]
Image name: LISTDBG.exe                          ... [5]
Timestamp:      Fri Aug 01 15:56:24 2014 (53DB3A18)
Checksum:       00000000
ImageSize:     0000A000
File version:   0.0.0.0
Product version: 0.0.0.0
File flags:    0 (Mask 0)
File OS:       0 Unknown Base
File type:     0.0 Unknown
File date:     00000000.00000000
Translations:  0000.04b0 0000.04e4 0409.04b0 0409.04e4

```

[4]Image pathまたは[5]Image nameからファイルの拡張子を確認します。この例では、「LISTDBG.exe」であることがわかります。

シンボルは、COBOLソースの「PROGRAM-ID」に指定した名前や、「CLASS-ID」と「METHOD-ID」を組み合わせたものになっています。目的プログラムリスト中には、ラベル名として出力されています。

“LISTDBG!DIVPROC+0x20e”の場合、“LISTDBG.exe”を作成するために用意したLISTDBG.cobとDIVPROC.cobのうち、シンボル“DIVPROC”を含むDIVPROC.cobの目的プログラムリスト(DIVPROC.LST)から、“0x20e”の位置を求めます。

[目的プログラムリスト(DIVPROC.LST)]

```

BEGINNING OF DIVPROC
00000000 32          db      0x32
00000001 07          db      0x07
00000002 44495650524F43 dc      "DIVPROC"
00000009 08          db      0x08
0000000A 4E6574434F424F4C dc      "NetCOBOL"
00000012 07          db      0x07
00000013 5631302E352E30 dc      "V10.5.0"
0000001A 3230313430383031 dc      "20140801"
00000022 3131343033393030 dc      "11403900"
0000002A 2B30303030      dc      "+0000"
0000002F 30313031      dc      "0101"
00000033 00          db      0x00
00000034          PROLOGUEAD:

```

```

00000034 XXXXXXXX          db      0xFFFFFFFF : PLB. 1
00000038                  _DIVPROC:          ... [6]
00000038 55                  push   ebp          ... [7]
00000039 8BEC                mov    ebp, esp
0000003B 81ECC0000000        sub    esp, 000000C0
00000041 8BD7                mov    edx, edi
00000043 8BFD                mov    edi, ebp
00000045 81EF64000000        sub    edi, 00000064
0000004B B90E000000          mov    ecx, 0000000E
00000050 B800000000          mov    eax, 00000000
00000055 FC                  cld
00000056 F3AB                repe
00000058 8BFA                mov    edi, edx
0000005A 897DEC              mov    ss:ebp-14, edi
0000005D 8975F0              mov    ss:ebp-10, esi
00000060 895DF4              mov    ss:ebp-0C, ebx
00000063 6800000000          push  00000000
00000068 C7859CFFFFFF00000000 mov    ss:ebp+FFFFFF9C, 00000000
00000072 2EFF2534000000      jmp    cs:00000034
00000079 0000000000000000    db      0x0000000000000000

```

DIVPROC.cobの目的プログラムリスト(DIVPROC.LST)から“DIVPROC”を検索し、出力位置[6]を求めます。そのシンボルの行の番地からそのシンボルのオフセット位置の“00000038”(16進数)を求めます。

上で求めた番地(オフセット)とシンボルからのオフセット値“0x20e”を加算し、リスト内のオフセット値を求めます。

```
0x38+0x20e = 0x246
```

DIVPROC.cobの目的プログラムリスト(DIVPROC.LST)から番地が“0x246”を検索します。

[目的プログラムリストDIVPROC.LST]

```

--- 14 ---          MOVE
000001D0 B830300000          mov    eax, 00003030
000001D5 668945A0            mov    ss:ebp-60, ax : 答え
--- 15 ---          COMPUTE          ... [8]
000001D9 31D2                xor    edx, edx
000001DB 8B3D18000000        mov    edi, %GWA+00000018
000001E1 8A17                mov    dl, ds:edi : 被除数
000001E3 83E20F              and    edx, 0F
000001E6 31C9                xor    ecx, ecx
000001E8 8A4F01              mov    cl, ds:edi+01 : 被除数+1
000001EB 83E10F              and    ecx, 0F
000001EE 8D1492              lea   edx, ds:edx+edx*4
000001F1 8D1451              lea   edx, ds:ecx+edx*2
000001F4 8A4F01              mov    cl, ds:edi+01 : 被除数+1
000001F7 80E1F0              and    cl, F0
000001FA 80F950              cmp    cl, 50
000001FD 75XX                jne   GLB. 4
000001FF F7DA                neg    edx
00000201              GLB. 4
000001FE 02
00000201 31C0                xor    eax, eax
00000203 8B3D18000000        mov    edi, %GWA+00000018
00000209 8A4702              mov    al, ds:edi+02 : 除数
0000020C 83E00F              and    eax, 0F
0000020F 31C9                xor    ecx, ecx
00000211 8A4F03              mov    cl, ds:edi+03 : 除数+1
00000214 83E10F              and    ecx, 0F
00000217 8D0480              lea   eax, ds:eax+eax*4
0000021A 8D0441              lea   eax, ds:ecx+eax*2
0000021D 8A4F03              mov    cl, ds:edi+03 : 除数+1
00000220 80E1F0              and    cl, F0
00000223 80F950              cmp    cl, 50

```

```

00000226 75XX          jne    GLB. 5
00000228 F7D8          neg    eax
0000022A          GLB. 5
00000227 02
0000022A 66A30A000000      mov    %GWA+0000000A, ax : TRLP+0
00000230 66891512000000  mov    %GWA+00000012, dx : TRLP+0
00000237 0FBF0512000000  movsx  eax, %GWA+00000012 : TRLP+0
0000023E 99                cdq
0000023F 0FBF0D0A000000  movsx  ecx, %GWA+0000000A : TRLP+0
00000246 F7F9                idiv  ecx          ... [7]
00000248 668945DA          mov    ss:ebp-26, ax : TRLS+0
0000024C 0FBF45DA          movsx  eax, ss:ebp-26 : TRLS+0
00000250 B102                mov    cl, 02
00000252 B500                mov    ch, 00
00000254 8D7DA0            lea   edi, ss:ebp-60 : 答え
00000257 E8XXXXXXX          call  JMPZNCVZB_REG1

```

番地が“00000246”の位置([7]の命令)が「idiv ecx」となっており、[2]で求めた命令と一致します。このため、異常終了したのがこの命令位置であると特定できます。

この命令から上方向に“--- nnn ---”の形式で始まる行を見つけます。この例は、“--- 15 --- COMPUTE”が見つかります。([8])

nnnにあたる部分の“15”がCOBOLソースの行番号に対応します。

DIVPROC.LSTのソースプログラムリストから、行番号が“15”の位置を参照します。([9])

[ソースプログラムリスト (DIVPROC. LST)]

```

行番号 一連番号 A B
1 000010 IDENTIFICATION DIVISION.
2 000020 PROGRAM-ID. DIVPROC.
3 000030*
4 000040 DATA DIVISION.
5 000050 WORKING-STORAGE SECTION.
6 000060 LINKAGE SECTION.
7 000070 01 引数.
8 000080 02 被除数 PIC 9(2).
9 000090 02 除数 PIC 9(2).
10 000100 01 答え PIC 9(2).
11 000110*
12 000120 PROCEDURE DIVISION USING 引数 RETURNING 答え.
13 000130*
14 000140 MOVE 0 TO 答え.
15 000150 COMPUTE 答え = 被除数 / 除数.          ... [9]
16 000160*
17 000170 EXIT PROGRAM.
18 000180 END PROGRAM DIVPROC.

```

15行目のCOMPUTE文で異常終了したことが特定できました。

## 22.2.3 障害発生箇所の特定方法(リンクマップファイル使用時)

リンク時に“/MAP”を指定してリンクマップファイルを作成し、PDBファイルが参照できない状態でプログラムが異常終了した場合の障害発生箇所の特定方法について説明します。

- 使用するファイル
  - クラッシュダンプ
  - 翻訳オプションSOURCE、COPY、LISTを指定して出力した翻訳リスト
  - リンクマップファイル(.MAP)

- 使用するデバッグツール
  - WinDbg (Debugging Tools for Windows - ネイティブ x86)

使用する例は、“[22.2.2 障害発生箇所の特定方法\(プログラムデータベースファイル使用時\)](#)”と同じです。

クラッシュダンプは、リンク時に、「/DEBUG」ではなく、「/MAP」を指定して作成した実行ファイルの実行時に採取されたものです。

また、以下の手順1.と2.は、“[22.2.2 障害発生箇所の特定方法\(プログラムデータベースファイル使用時\)](#)”と同じです。

1. クラッシュダンプが調査対象のものかどうかを確認する。
2. クラッシュダンプから例外発生箇所と理由を特定する。
3. COBOLソース行位置を特定する。

クラッシュダンプとリンクマップリストからCOBOLソース行位置を求める方法を説明します。

- a. WinDbgのアドレスの情報からCOBOLソース行位置を求めます。

[WinDbg起動時出力結果]

```

Loading Dump File [C:\ProgramData\Fujitsu\NetCOBOL\COBSNAP\LISTDBG_1720_20140801-155635_cobsnap.dmp]
User Mini Dump File with Full Memory: Only application data is available

Symbol search path is: *** Invalid ***
*****
* Symbol loading may be unreliable without a symbol search path.          *
* Use .symfix to have the debugger choose a symbol path.                  *
* After setting your symbol path, use .reload to refresh symbol locations. *
*****
Executable search path is:
Windows 7 Version 7601 (Service Pack 1) MP (2 procs) Free x86 compatible
Product: WinNt, suite: SingleUserTS
Machine Name:
Debug session time: Fri Aug 1 15:56:35.000 2014 (UTC + 9:00)
System Uptime: 0 days 4:35:31.089
Process Uptime: 0 days 0:00:01.000
.....
*** ERROR: Symbol file could not be found. Defaulted to export symbols for ntdll.dll -

***** Symbol Loading Error Summary *****
Module name          Error
ntdll                The system cannot find the file specified

You can troubleshoot most symbol related issues by turning on symbol loading diagnostics (!sym noisy) and repeating
the command that caused symbols to be loaded.
You should also verify that your symbol search path (.sympath) is correct.
This dump file has an exception of interest stored in it.
The stored exception information can be accessed via .ecxr.
(1720.1108): Integer divide-by-zero - code c0000094 (first/second chance not available)
*** WARNING: Unable to verify checksum for LISTDBG.exe
*** ERROR: Symbol file could not be found. Defaulted to export symbols for LISTDBG.exe -
eax=00000002 ebx=7ffdf000 ecx=00000000 edx=00000000 esi=00000000 edi=00406100
eip=00401516 esp=0012fd74 ebp=0012fe34 iopl=0         nv up ei ng nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010283
LISTDBG!DIVPROC+0x20e:
00401516 f7f9             idiv   eax,ecx                ... [1]

```

異常終了した位置が“00401516”([1])であることがわかります。

- b. [1]のアドレス“00401516”からこのアドレスを含むモジュールを特定するため、モジュールの一覧をlmコマンドで出力します。

[lmコマンド出力結果]

```

0:000> lm
start      end          module name
002e0000 00324000    F3BIIO      (deferred)
00330000 00371000    F3BIFRM     (deferred)
00380000 003ca000    F3BIDBG     (deferred)
003d0000 003e5000    F3BIOLES    (deferred)
003f0000 003f5000    F3BIOVLD    (deferred)
00400000 0040a000    LISTDBG C (export symbols)  LISTDBG.exe      ... [2]
00420000 0045a000    F3BIOLER    (deferred)
00460000 004a2000    F3BISCRN    (deferred)
004b0000 00508000    F3BILPIO    (deferred)
00510000 0051e000    F3BISQL     (deferred)
00520000 00540000    F3BIPRIO    (deferred)
00540000 00551000    F3BISCLS    (deferred)
00560000 0056e000    F3BILANP    (deferred)
00570000 00588000    F3BIZCOM    (deferred)
005a0000 005ae000    F3BIAESV    (deferred)
005b0000 005dc000    F3BKATCH    (deferred)
005e0000 005f2000    F3BIEFNC    (deferred)
00600000 00636000    F3BIEFXH    (deferred)
00640000 00665000    F3BICICL    (deferred)
00670000 0067f000    F3BINDBC    (deferred)
006e0000 006f2000    F3BIFUNC    (deferred)
10000000 1006c000    F3BIPRCT    (deferred)
6c580000 6c6a2000    IMJP14K     (deferred)
6c6b0000 6c7c4000    IMJP14      (deferred)
71c90000 71ce1000    winspool    (deferred)
72170000 72177000    wsock32     (deferred)
72180000 7220e000    msvcp90     (deferred)
72410000 72494000    comctl32    (deferred)
72770000 72813000    msvcr90     (deferred)
74810000 74823000    dwmapi      (deferred)
74af0000 74b30000    uxtheme     (deferred)
750d0000 750d9000    version     (deferred)
75b40000 75bad000    sysfer      (deferred)
75bb0000 75bfc000    apphelp     (deferred)
75dc0000 75e0b000    KERNELBASE  (deferred)
76000000 76057000    shlwapi     (deferred)
76060000 7606a000    lpk         (deferred)
76150000 76d9a000    shell32     (deferred)
76da0000 76e74000    kernel32    (deferred)
76e80000 76f4c000    msctf       (deferred)
76f50000 76ff2000    rpcrt4      (deferred)
77000000 77006000    nsi         (deferred)
77010000 770bc000    msvcrt      (deferred)
770c0000 7715d000    usp10       (deferred)

```

[1]の“00401516”は、[2]の“00400000 0040a000 LISTDBG C (export symbols) LISTDBG.exe”に含まれるため、“LISTDBG.exe”モジュール内で異常終了していることが特定できます。

- c. リンク時に「/MAP」を指定して作成したリンクマップリストを使って、モジュールのどの部分が特定します。

[リンクマップリスト(LISTDBG .MAP)]

```

LISTDBG

Timestamp is 53db4524 (Fri Aug 01 16:43:32 2014)

Preferred load address is 00400000      ... [3]

Start      Length   Name                Class
0001:00000000 00003f1eH .text              CODE
~省略~

```

Address	Publics by Value	Rva+Base	Lib:Object
0001:00000000	\$LISTDBG.CODE	00401000	LISTDBG.OBJ
0001:00000000	_WinMain@16	00401000 f	LISTDBG.OBJ
0001:000002d0	\$DIVPROC.CODE	004012d0	DIVPROC.OBJ
0001:00000308	_DIVPROC	00401308 f	DIVPROC.OBJ ... [4]
0001:0000054a	_JMP1PLAN	0040154a f	f3bicimp:F3B1PRCT.dll
～省略～			

[2]のモジュールのローディングアドレス“400000”と、[3]の“Preferred load address is 00400000”のアドレスが同じになっているか確認します。

－ [2]のモジュールのローディングアドレスと[3]のアドレスが同じ場合

[1]のアドレス“00401516”をそのまま“Rva+Base”の値と比較して[1]のアドレスより小さい、かつ、一番近いアドレスを探します。

この例では、

[4] 0001:00000308	_DIVPROC	00401308 f	DIVPROC.OBJ
-------------------	----------	------------	-------------

が該当します。

次に、以下の計算で、シンボル相対オフセットを求めます。

$0x00401516 - 0x00401308 = 0x20e$
-----------------------------------

上記の結果から、以下の情報が求められます。

モジュール	: "LISTDBG.exe"
シンボル	: "_DIVPROC"
シンボルからのオフセット	: 0x20e

－ [2]のモジュールのローディングアドレスと[3]のアドレスが異なる場合

それぞれの相対オフセットを計算します。

1. 異常終了した位置の相対オフセットを求めます。

$0x00401516 - 0x00400000 = 0x1516$
------------------------------------

2. それぞれのシンボルの相対オフセットを求めます。

main	$0x00401000 - 0x00400000 = 0x1000$	
LISTDBG	$0x00401000 - 0x00400000 = 0x1000$	
DIVPROC	$0x00401308 - 0x00400000 = 0x1308$	... [5]
JMP1PLAN	$0x0040154a - 0x00400000 = 0x154a$	
:		

上記の中から、1-で求めたオフセットより小さく、かつ、一番近いオフセットを探します。

この例では、

[5] "DIVPROC 0x00401308 - 0x00400000 = 0x1308"
--

が該当します。

次に、以下の計算で、シンボル相対オフセットを求めます。

$0x1516 - 0x1308 = 0x20e$
---------------------------

上記の結果から、以下の情報が求められます。

モジュール	: "LISTDBG.exe"
シンボル	: "_DIVPROC"
シンボルからのオフセット	: 0x20e

これ以降の作業は、“22.2.2 障害発生箇所の特定方法(プログラムデータベースファイル使用時)”と同じです。

## 22.2.4 データの参照方法

異常終了した場所がCOBOLのプログラムまたはメソッドの場合、以下を使用して、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。

- 使用するファイルとデバッグツール
- クラッシュダンプ
- 翻訳オプションSOURCE、COPY、MAPを指定して出力した翻訳リスト
- WinDbg(Debugging Tools for Windows - ネイティブ x86)

### 注意

- 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. 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WinDbg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 Pablics 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 Pablics 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 Pablics 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	Publics 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
:		

## 2. 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“B.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、

- 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。
- 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。
- 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。

## 3. 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。

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、クラッシュダンプを開きます。
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[Virtual]欄に入力します。



### 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。

0040611a 30 30 00 00 00 00 00 47 43 42 05 48 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、[Virtual]欄に「00406030+20」と入力します。

[Memory]ウィンドウの表示

00406050 f8 6a 40 00 78 4c 49 41 01 00 00 00 60 fe 12 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、[Virtual]欄に「004061f8」と入力します。(+00000000は省略しています。)

[Memory]ウィンドウの表示

004061f8 31 30 00 00 30 1a 40 00 01 00 00 00 00 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[Virtual]欄に「EBP+FFFFFF9C」と入力します。

[Memory]ウィンドウの表示

0012fe54 14 2a c4 00 00 00 00 30 4c 49 56 3c 2a c4

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、[Virtual]欄に「004c2a14+24」と入力します。

[Memory]ウィンドウの表示

00c42a38 18 61 40 00 78 4c 49 41 01 00 00 00 5c fe 12

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、[Virtual]欄に「00406118」と入力します。(+00000000は省略しています。)

.....

# 第6部 プロジェクトマネージャを利用した開発

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。また、NetCOBOLが提供する各種ツールをプロジェクトマネージャから操作することができます。

なお、NetCOBOL V9.0L10からEclipseをベースとしたCOBOLプログラム開発環境「NetCOBOL Studio」を提供しています。NetCOBOL Studioの詳細は、“NetCOBOL Studioユーザーズガイド”を参照してください。

第23章 プロジェクト管理機能.....	651
第24章 対話型デバッガの使い方.....	678
第25章 ツールの操作.....	722
第26章 Webアプリケーション開発支援機能の利用.....	747
第27章 オブジェクト指向プログラミングの開発.....	775
第28章 COM機能.....	796

## 第23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。

### 23.1 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。

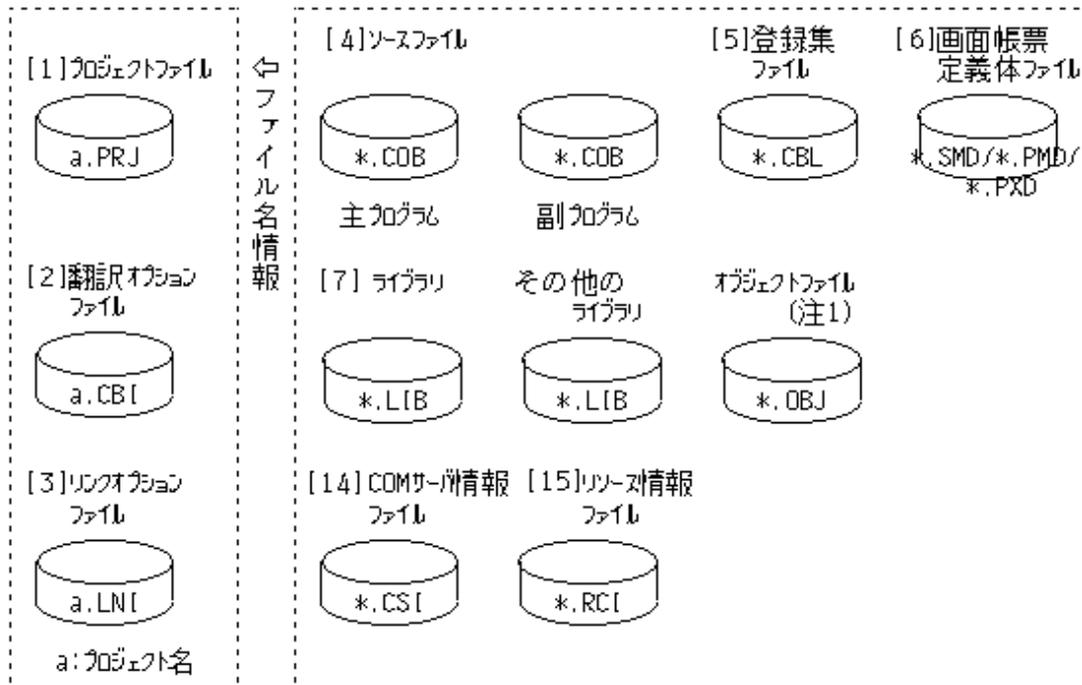
### 23.2 プロジェクト管理に必要な資源

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)
- 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。



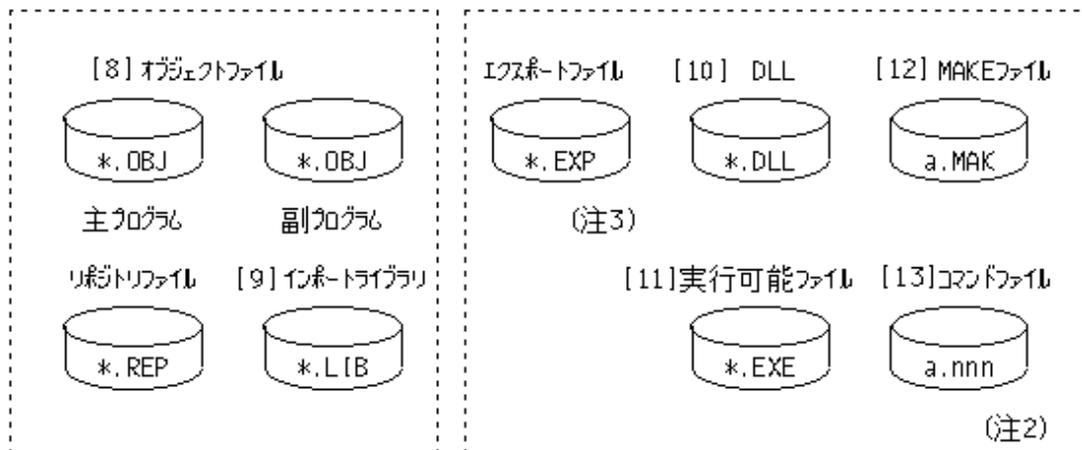
↑作成    ↓ビルド

↓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23.1 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 <ul style="list-style-type: none"> <li>・ ファイルの依存関係</li> <li>・ 最終ターゲット名</li> <li>・ COBOL ソースファイル名</li> <li>・ 登録集ファイル名</li> <li>・ 各種定義体ファイル名</li> <li>・ その他必要なファイル名</li> </ul>	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。
			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]	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。
[8]	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。
[9]	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。
[10]	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。
[11]	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。
[12]	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。
[13]	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。

	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]	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。
[15]	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、“[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: 画面帳票定義体を検索するフォルダについては、“[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。

## 23.3 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]“[23.4.1 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]“[23.4.2 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]“[23.4.3 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つりに登録します。[参照]“[23.4.4.1 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つりに登録します。[参照]“[23.4.4.7 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]“[23.4.4.2 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]“[23.4.4.3 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つりに登録します。[参照]“[23.4.4.7 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]“[23.4.4.4 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つりに登録します。[参照]“[23.4.4.7 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]“[23.4.4.5 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つりに登録します。[参照]“[23.4.4.7 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]“[23.4.4.6 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]“[23.4.5 プロジェクトのビルド・リビルド](#)”
- ビルド・リビルドの結果は、ビルダウイ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]“[23.4.6 翻訳エラーの修正](#)”

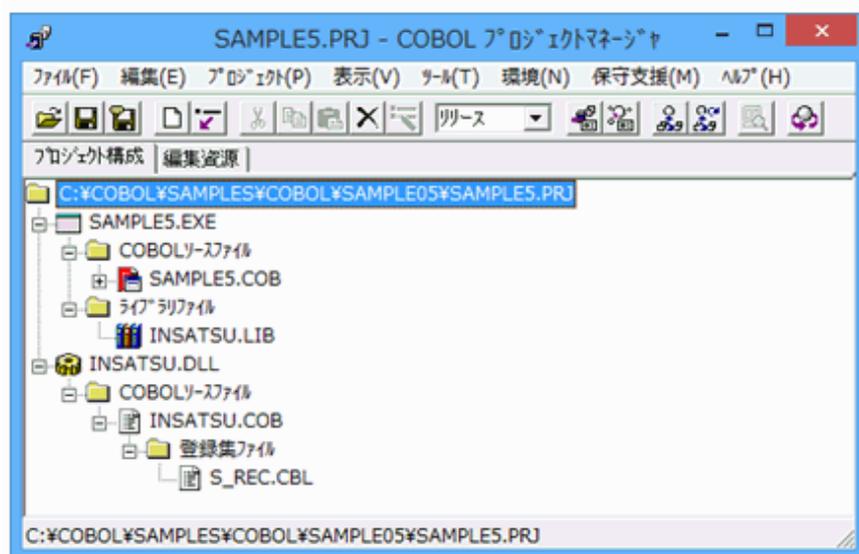
17. [プロジェクト]メニューから“実行”または“パラメタを指定して実行”を選択し、プログラムを実行します。[参照]“[23.4.7 アプリケーションプログラムの実行](#)”
18. [ファイル]メニューから“プロジェクトを閉じる”、“上書き保存”、“名前を付けて保存”または“終了”のどれかを指定することにより、プロジェクトファイルを更新します。[参照]“[23.4.4.8 プロジェクトファイルの保存](#)”

## 23.4 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用します。

### 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用します。

## 23.4.1 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の作成方法を説明します。

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. ファイル名のエディットボックスに、作成するプロジェクトファイルの名前を入力します。
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。

## 23.4.2 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]メニューから[オプション]>[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]“[25.1.4 \[翻訳オプション\]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]“[23.4.4.3 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.3.25 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の検索は、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対象になります。

## 23.4.3 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]メニューから、[オプション]>[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]“[23.4.3.1 \[リンクオプション\]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.3.60 TEST \(デバッグ機能の使用の可否\)](#)”

### 23.4.3.1 [リンクオプション]ダイアログ

リンクオプションを設定します。

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]“[J.2.1 LINKコマンドのオプション](#)”

表23.2 [リンクオプション]ダイアログ

項目	説明
コマンドライン	リンクオプションを指定します。
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。
/MAP	リンクオプション/MAP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。
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。
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。 V11以降は指定できません。
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、MSVCRT.LIB以外のライブラリをリンクする場合に指定します。
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。
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(COBOL単体用)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(言語間結合用)かを指定します。
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。

## 23.4.4 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



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。

### 23.4.4.1 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。

#### 23.4.4.2 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]“23.4.4.7 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]“23.4.4.3 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。

#### 23.4.4.3 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]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、“11.1.4 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。

#### 23.4.4.4 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リー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. 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、“登録集ファイル”フォルダを作成します。[参照]“23.4.4.7 フォルダの作成”
2. 登録集ファイルを登録します。

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]“23.4.4.2 COBOLソースファイルの登録”

#### 23.4.4.5 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
- ・ 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]“23.4.4.4 登録集ファイルの登録”

#### 23.4.4.6 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]“23.4.4.2 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。



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。



- ー インポートライブラリは、必ず“DLL名.LIB”という名前指定してください。
- ー 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。
- ー 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]“[23.5 モジュール定義ファイル](#)”

### 23.4.4.7 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23.3 フォルダの種類と指定可能な位置](#)”に示します。

表23.3 フォルダの種類と指定可能な位置

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、“[17.2.3.2.1 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、“[27.3.5.2 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、[編集]メニューから[フォルダ作成]を選択します。  
→ [フォルダの種類](#)が表示されます。
2. 作成するフォルダの種類を選択します。  
→ フォルダが作成されます。



#### 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。

### 23.4.4.8 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。

## 23.4.5 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りした場合、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。

### 23.4.5.1 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。



**注意**

- ・ 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せん(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。

## 23.4.6 翻訳エラーの修正

翻訳エラーが検出された場合、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(診断メッセージ)が表示されます。

翻訳エラーの修正は、ビルダのエラージャンプ機能を使用します。

ビルダのエラージャンプ機能とは、ビルダウィンドウに表示された翻訳時メッセージから、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機能です。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。ビルダのエラージャンプ機能を使うことで、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。

ビルダのエラージャンプ機能を使用して翻訳エラーを修正する場合、以下の操作を行います。

1. ビルダウィンドウに表示された翻訳時メッセージをダブルクリックします。  
→ COBOLソースプログラムの編集画面が表示され、翻訳エラーが検出された行にエラージャンプします。
2. 編集画面でCOBOLソースプログラムを修正します。

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。

### 注意

- 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、エラージャンプ機能は利用できません。

## 翻訳時メッセージをファイルに保存する

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。

### 注意

- 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.3.32 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)”“付録B 翻訳リスト”
- 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。

## 23.4.7 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]メニューから[パラメタを指定して実行]を選択し、[パラメタを指定して実行]ダイアログの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]メニューから[実行]を選択します。

### 注意

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]“5.7 実行環境設定ツール”

## 23.4.8 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、“[J.5 INSDBINF コマンド](#)”を参照してください。

### 23.4.8.1 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23.1 INSDBINFを使用しない場合

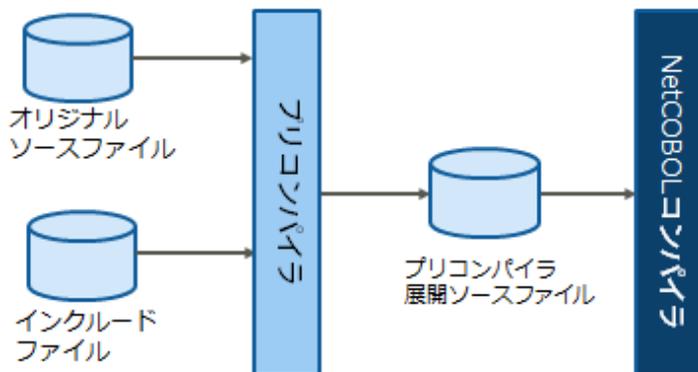
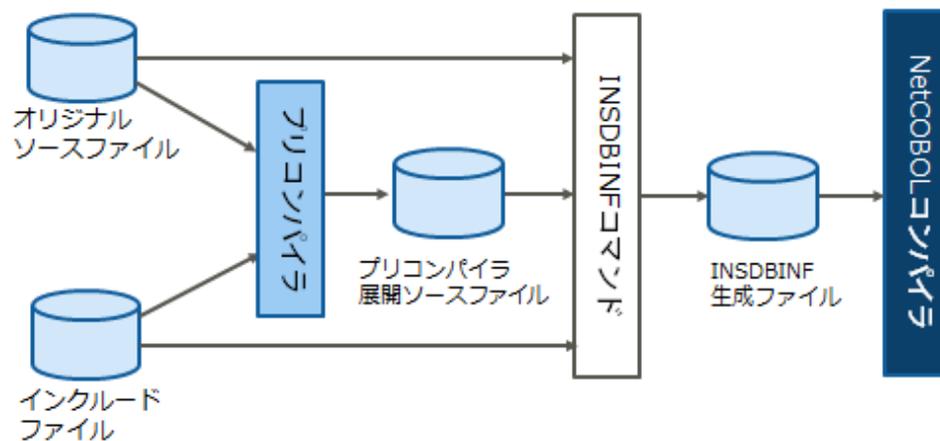


図23.2 INSDBINFを使用する場合



#### 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な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、“[23.4.8.6 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、“[J.5 INSDBINFコマンド](#)”を参照してください。

## 23.4.8.2 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、“[23.4.8.4 \[プリコンパイラ設定\]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。

## 23.4.8.3 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23.3 INSDBINFコマンドを使用しない場合

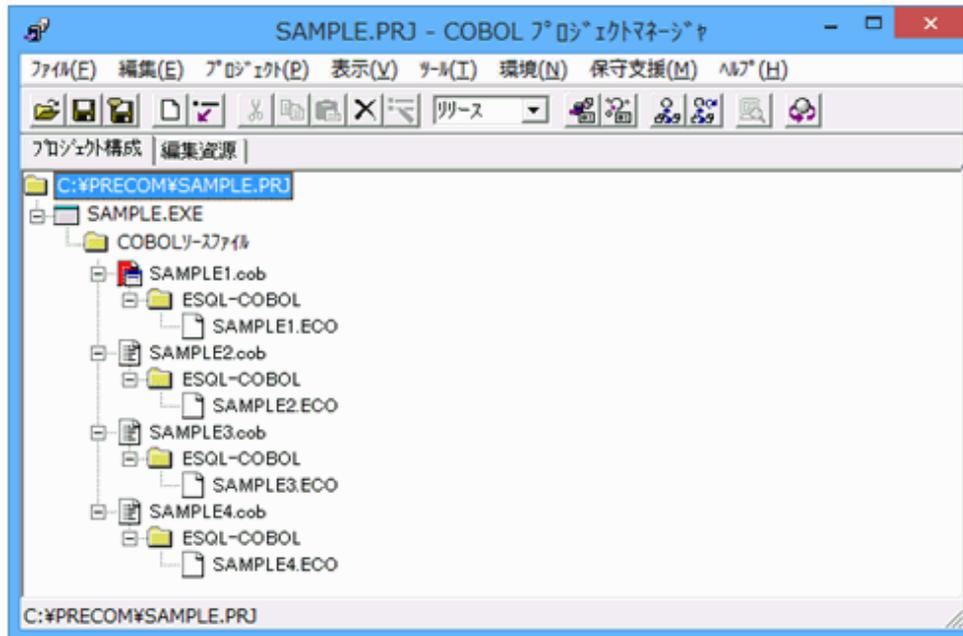
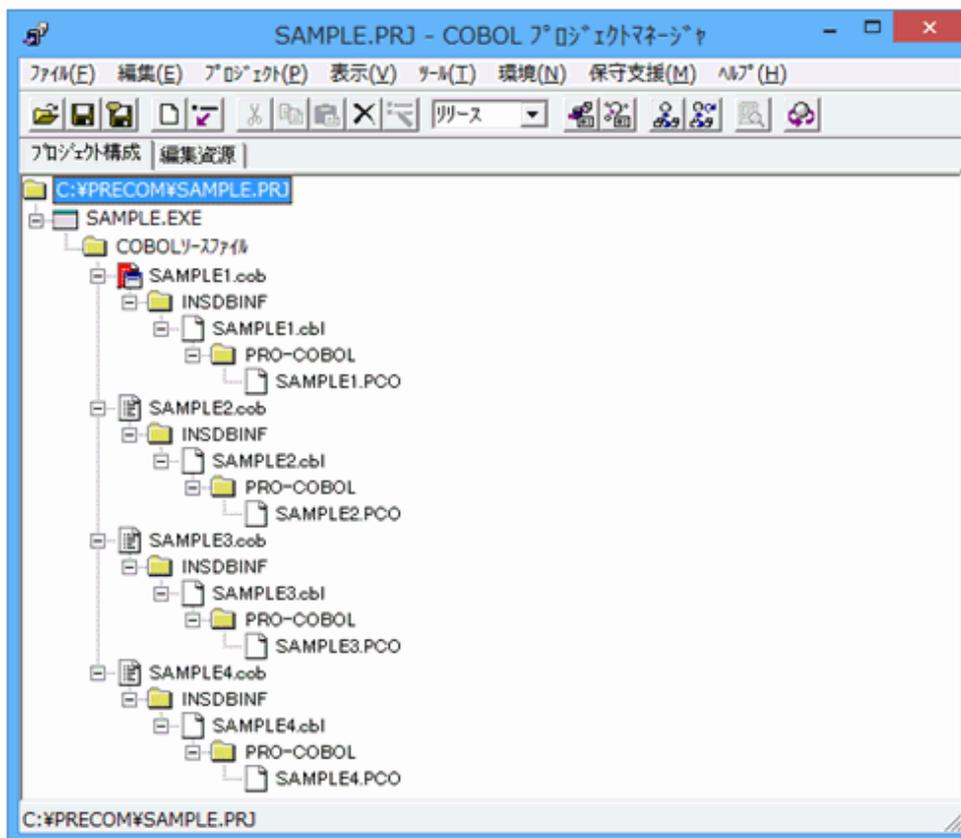


図23.4 INSDBINFコマンドを使用する場合



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、プリコンパイラの必要のない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。  
[プリコンパイラ設定]ダイアログが表示されます。

3. 必要な情報を登録します。

- コマンドページ  
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  
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  
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ページ  
INSDBINFコマンドを使用する場合、[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、“[23.4.8.4 \[プリコンパイラ設定\]ダイアログ](#)”を参照してください。

情報の登録後、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。

4. “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。

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。

#### 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。  
[プリコンパイラ設定]ダイアログが表示されます。

2. 必要な情報を変更します。

- コマンドページ  
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  
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  
出力ファイル名を更新します。
- INSDBINFページ  
INSDBINFコマンドを使用する場合、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、“[23.4.8.4 \[プリコンパイラ設定\]ダイアログ](#)”を参照してください。

情報の更新後、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。

3. “ビルド”または“リビルド”を実行します。

### 23.4.8.4 [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]ダイアログの詳細はヘルプを参照してください。

表23.4 [プリコンパイラ設定]ダイアログ(共通)

項目	説明
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。

表23.5 [プリコンパイラ設定]ダイアログの[コマンド]ページ

項目	説明
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、“マクロ”を参照してください。
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。

新規登録の場合と更新登録の場合は同じ手順で設定を行います。

表23.6 [プリコンパイラ設定]ダイアログの[入力]ページ

項目	説明
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。 更新登録の場合、表示される入力ファイル名は1つとなります。
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。 更新登録の場合、使用できません。
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つ選択しているとき使用可能です。
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。 更新登録の場合、使用できません。

表23.7 [プリコンパイラ設定]ダイアログの[出力]ページ

項目	説明
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。
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。

更新登録の場合、出力ファイル名の変更はできません。

表23.8 [プリコンパイラ設定]ダイアログの[INSDBINF]ページ

項目	説明
----	----

INSDBINFコマンドを使用する	プリコンパイル後に、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。
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。 新規登録の場合、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とき指定します。 更新登録の場合、[中間ファイル名]が指定されているときは、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。 新規登録の場合、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とき指定します。 更新登録の場合、[中間ファイル名]が指定されているときは、入力不可となります。入力不可の状態 で、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。
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。 更新登録の場合、[中間ファイル名]が指定されているときは、入力不可となります。

## 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。

### 23.4.8.5 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“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。

### 23.4.8.6 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。

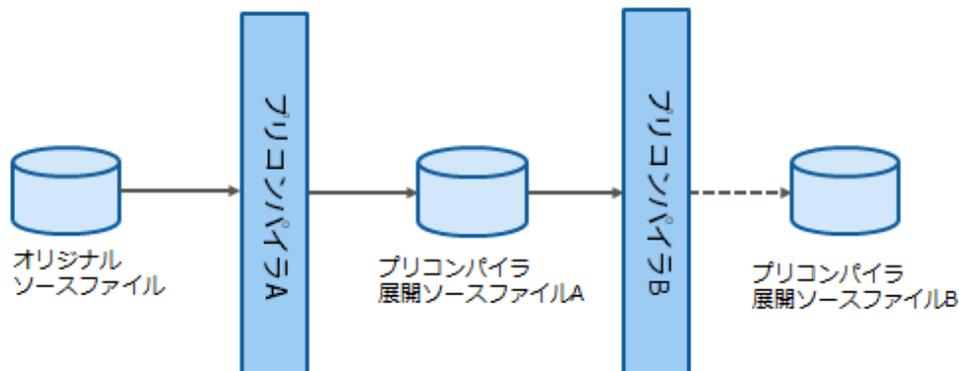
プロジェクトマネージャでは、COBOLの登録集と同様に扱います。

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]メニューから[フォルダ作成] > [インクルードファイル]を選択します。  
→ “インクルードファイル”フォルダが生成されます。
3. 生成されフォルダに、インクルードファイルを“追加”または“新規作成”します。

### 23.4.8.7 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。

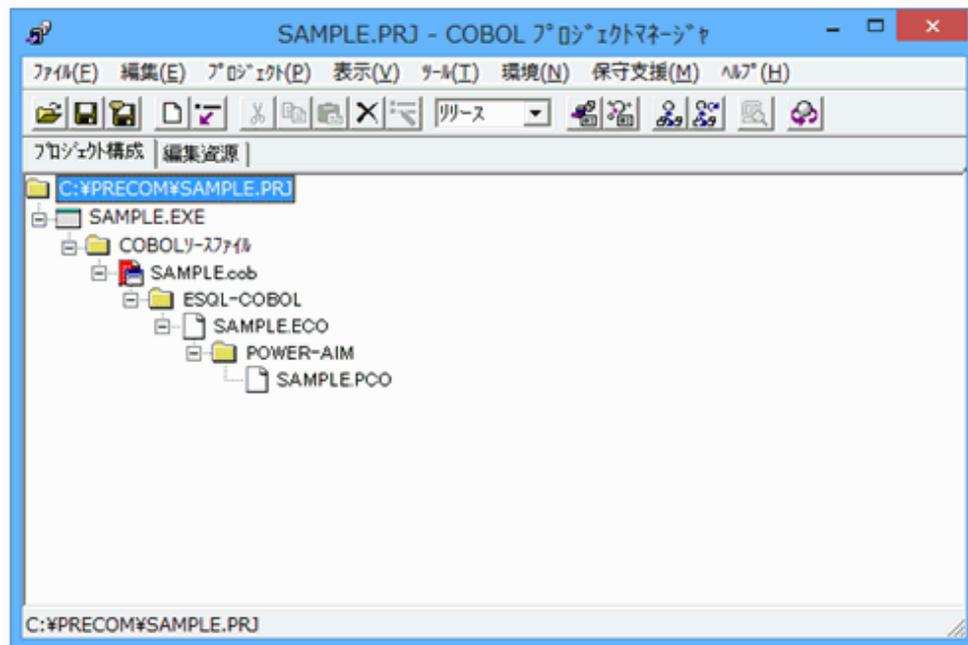


プロジェクトマネージャでは、あるプリコンパイラの入力ファイルを出力ファイルとする、新規のブロックを作成することにより、多段プリコンパイルが実現できます。

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“23.4.8.4 [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。



### 23.4.8.8 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。

### 23.4.9 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、[プロジェクト]メニューから[オプション]>[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。

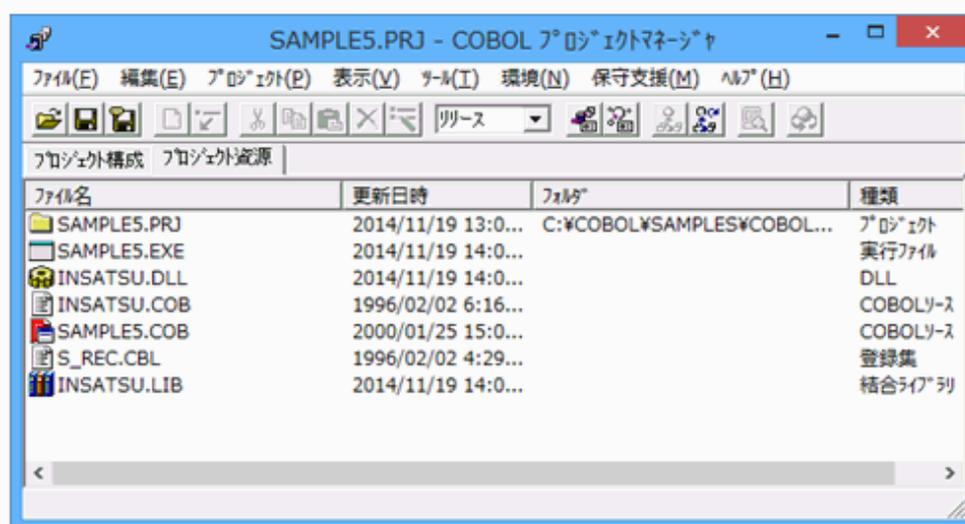
表23.9 [バージョン情報設定]ダイアログ

項目	説明
ファイルバージョン	メジャーバージョン、マイナーバージョン、リリース番号を指定します。それぞれの値は、スピンドルボタンで変更できます。
製品バージョン	メジャーバージョン、マイナーバージョン、リリース番号を指定します。それぞれの値は、スピンドルボタンで変更できます。
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。
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。

## 23.4.10 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。

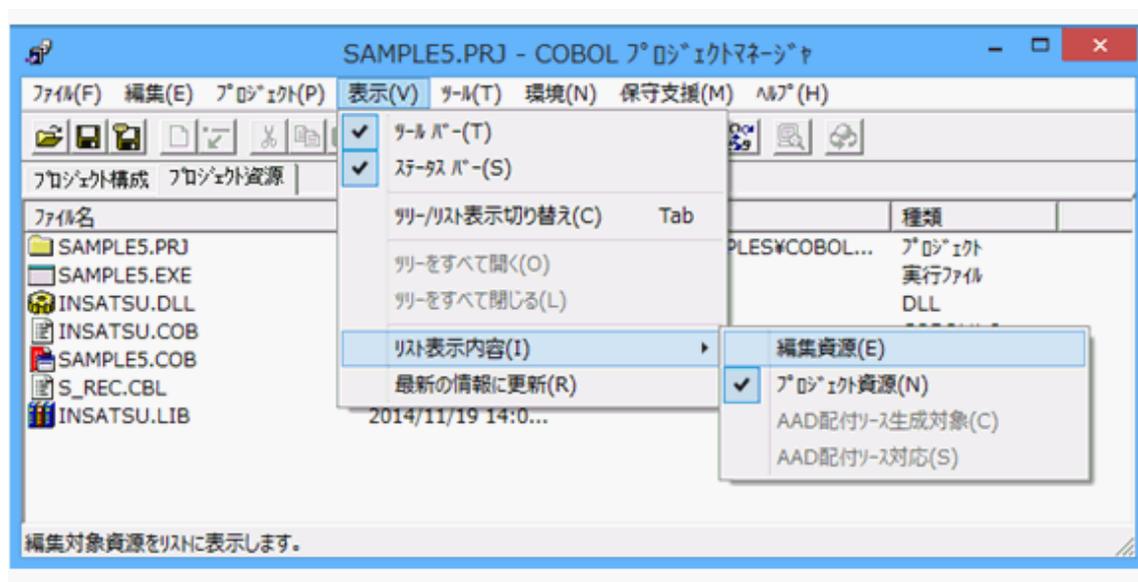
### AAD配付ソース生成対象

配布ソース生成時に使用します。詳細は、“NetCOBOL ユーザーズガイド(OSIV分散開発)プロジェクトマネージャ編”を参照してください。

### AAD配付ソース対応

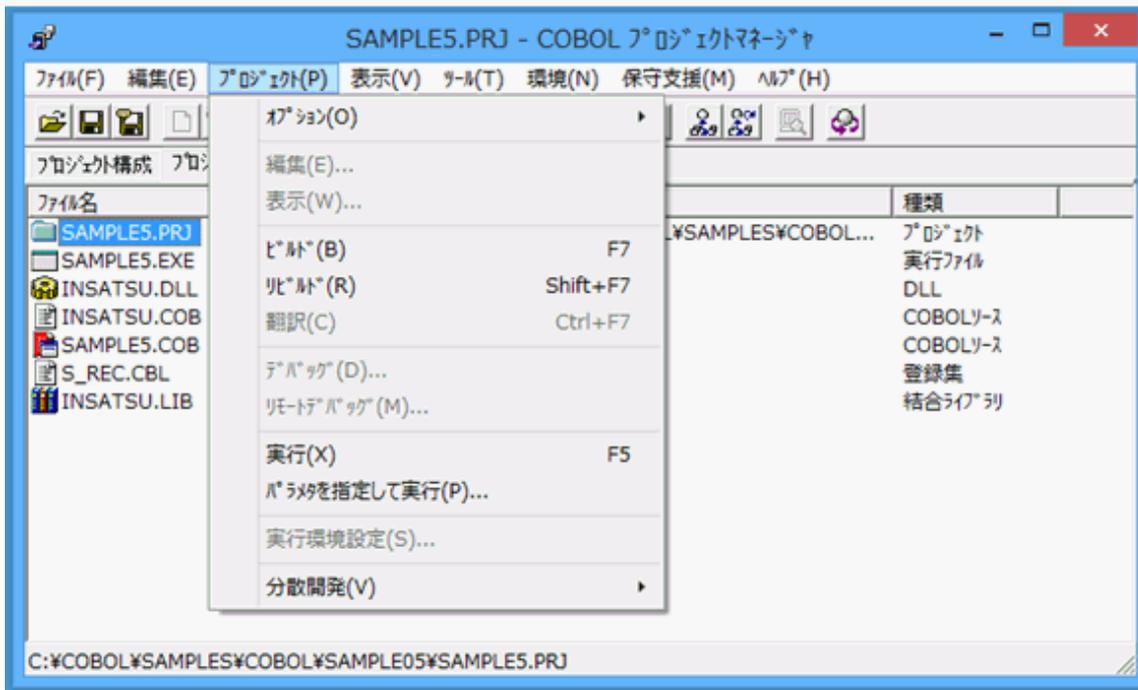
配付ソースの対応関係を表示します。詳細は、“NetCOBOL ユーザーズガイド(OSIV分散開発)プロジェクトマネージャ編”を参照してください。

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。

## 23.5 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。

### 23.5.1 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
  - 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、“[27.3.5.3 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。

## 23.6 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。

### 23.6.1 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。

## 23.6.2 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

## 23.6.3 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23.10 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23.10 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	<ul style="list-style-type: none"> <li>• ビルダ</li> <li>• NMAKEコマンド</li> </ul>	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	<ul style="list-style-type: none"> <li>• 実行可能ファイル</li> <li>• COBOL ソースファイル</li> <li>• オブジェクトファイル</li> <li>• リポトリファイル</li> <li>• 登録集ファイル</li> <li>• 各種定義体ファイル</li> </ul>

比較項目	MAKEファイル	プロジェクトファイル
		<ul style="list-style-type: none"> <li>ライブラリファイル</li> <li>モジュール定義ファイル</li> </ul>
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。

## 23.6.4 コマンドファイル

“23.6.2 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24章 対話型デバッグの使い方

本章では、対話型デバッグを使って作成したプログラムをデバッグする方法について説明しています。

### 24.1 デバッグの概要

デバッグでは、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。

デバッグは、デバッグの基本的な機能(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。

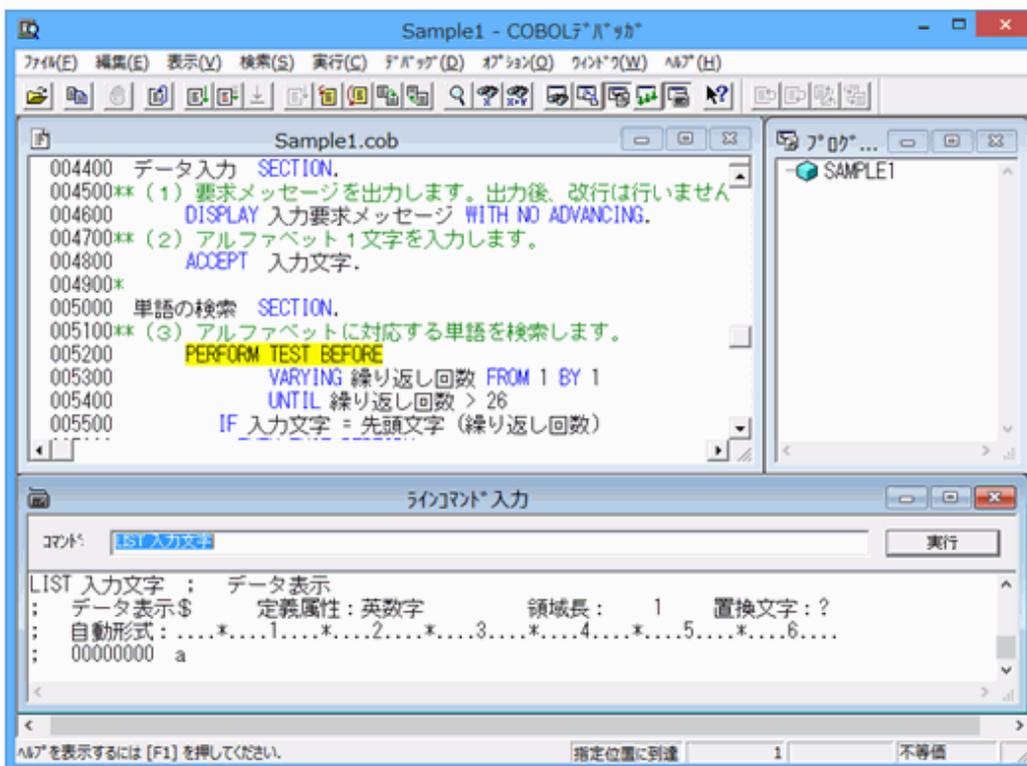
#### 特定実行条件での中断

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

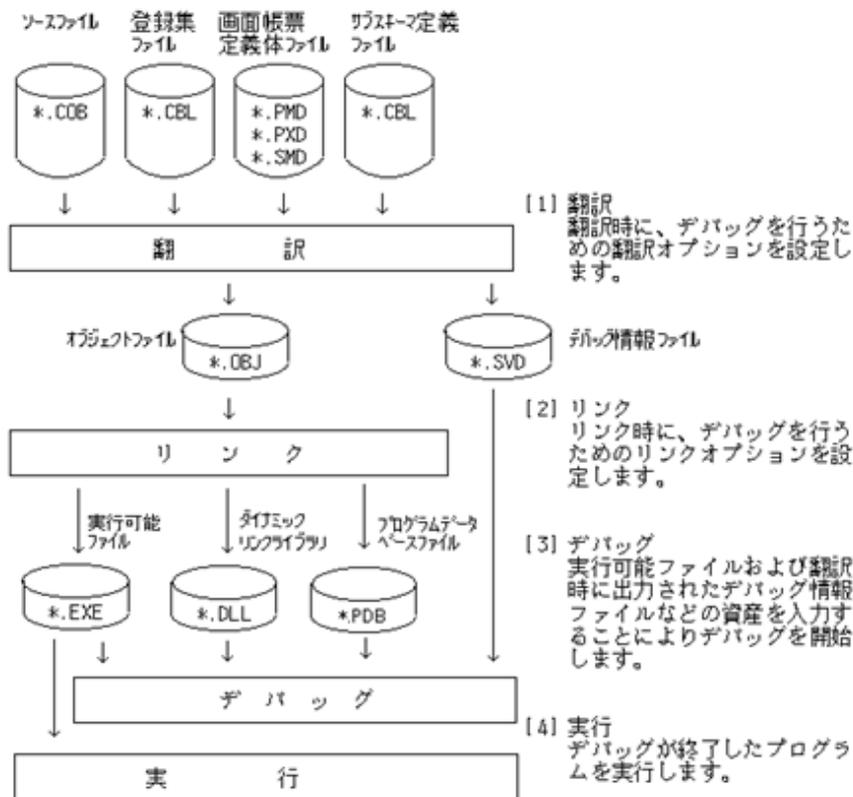
ファイルからコマンドを入力することにより、バッチデバッグを行うことができます。また、操作の履歴をファイルに保存し、これを入力として使用することにより、デバッグ操作を再現させることができます。

また、デバッグ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グについて説明します。対話型リモートデバッグについては、“24.9 対話型リモートデバッグの使い方”を参照してください。



## 24.2 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。

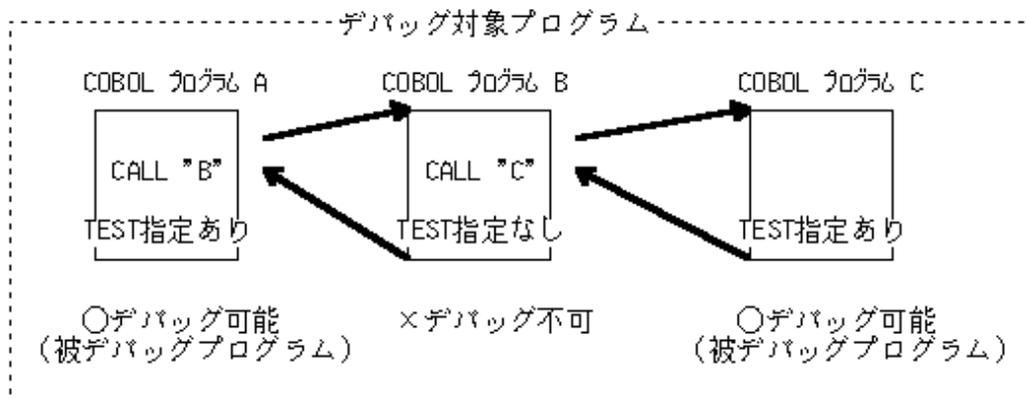


### 24.2.1 被デバッグプログラム

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。



## 24.2.2 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、“[23.4.5.1 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、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.3.60 TEST \(デバッグ機能の使用の可否\)](#)”

## 24.2.3 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”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”の指定形式を示します。

```
/DEBUG
```

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”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、“[23.4.5.1 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”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。



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”は指定しないでください。

## 24.3 デバッガの環境変数

デバッガが使用する環境変数を以下に示します。詳細は、それぞれの環境変数の説明を参照してください。

- [C.2.76 @SVD\\_PERFORM\\_CHECK](#) (デバッガでPERFORMレベルの監視を行う指定)
- [C.2.75 @SVD\\_ENV](#) (デバッグ環境の指定)
- [C.2.74 @SVD\\_COMPATIBILITY](#) (デバッガの互換性の指定)

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、“[24.5.2.2 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。

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。

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。

## 24.4 デバッガの起動

ここでは、デバッガ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。

## 24.4.1 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。

### 24.4.1.1 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]“[24.4.2.2 起動コマンドを使ったデバッグ指定](#)”

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。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。

### 24.4.1.2 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グ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グ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、“[24.9.2.2 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。

### 24.4.1.3 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。

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グを起動し、デバッグを開始することができます。[参照]“[C.2.36 @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、デバッグが起動され、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。

## 24.4.2 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、デバッグが必要とする資源(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。

### 24.4.2.1 [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グ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

### 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”

### 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。

このパラメタは、初期値としてpmd;pxd;smd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。

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、“[24.5.2.2 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。

## 24.4.2.2 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24.1 起動パラメタ](#)”に示します。

表24.1 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。

指定形式	内容
/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
```

## 24.5 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。

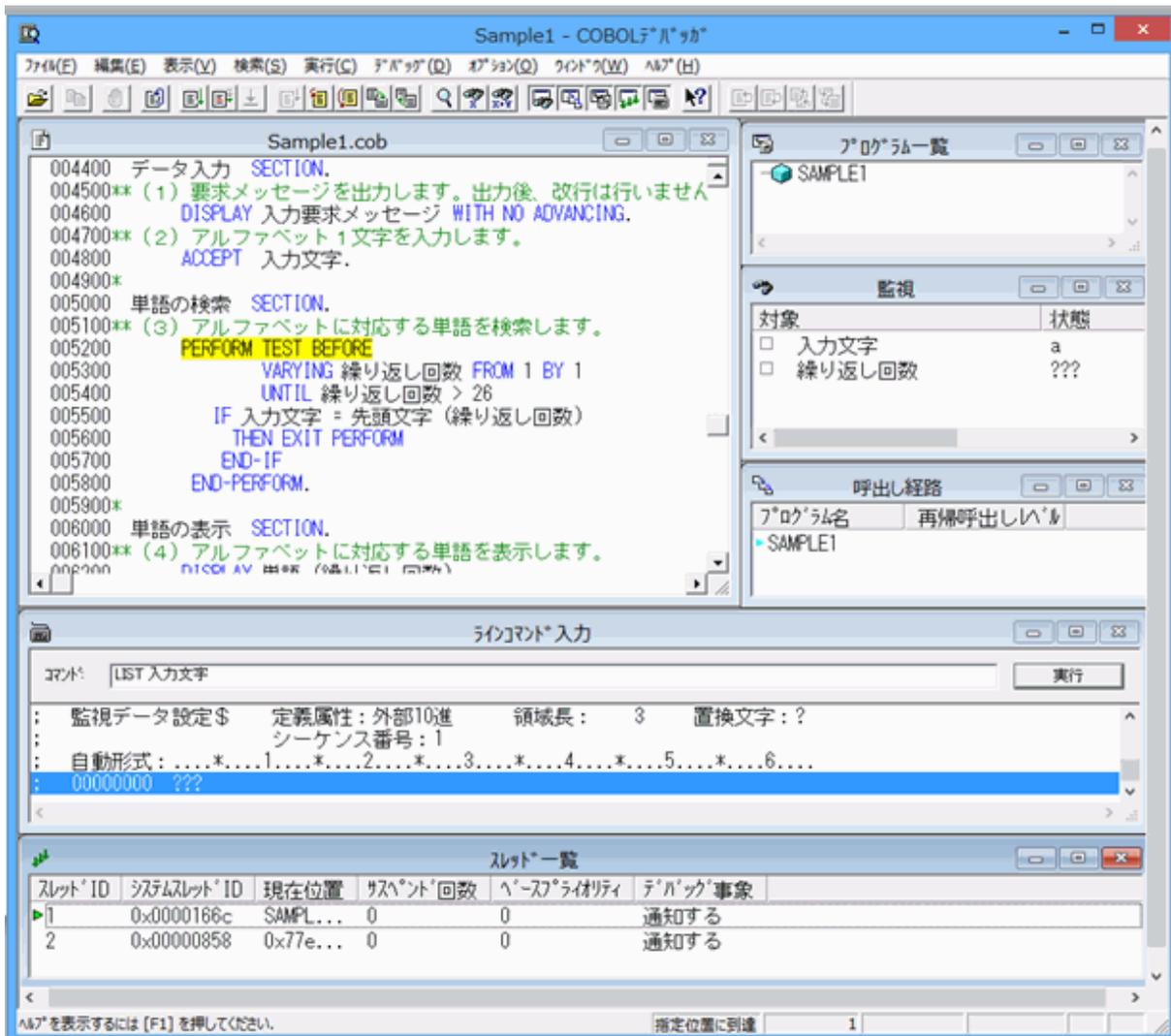
### 24.5.1 デバッガの画面

デバッガの画面構成を以下に示します。

#### 24.5.1.1 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

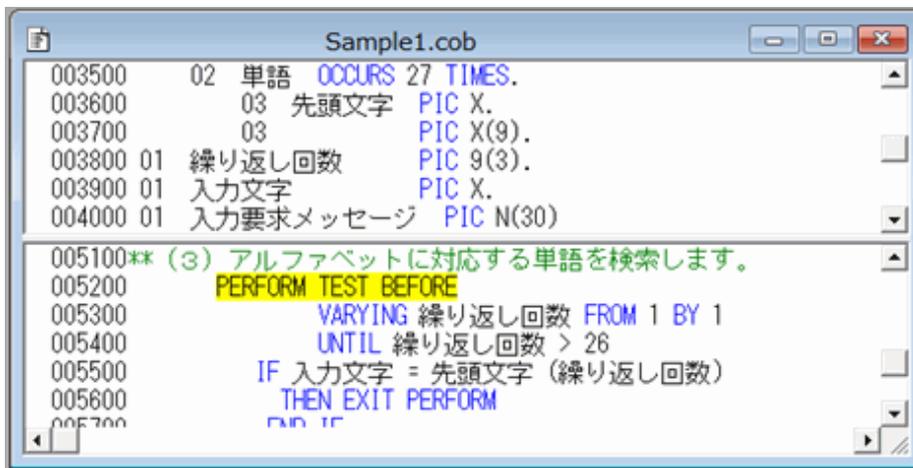
- ・ 実行が中断したときの中断事象
- ・ 現在の暗黙スレッドのスレッドID
- ・ ソースファイルウィンドウ内のカーソル位置
- ・ ラインコマンドにおける大文字/小文字の比較基準
- ・ トレース採取の状態
- ・ 履歴出力の状態

### 24.5.1.2 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、プログラム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。

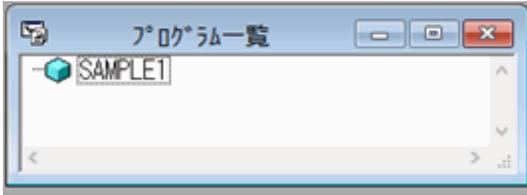
### 24.5.1.3 プログラム一覧ウィンドウ

プログラム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。

プログラム一覧ウィンドウは、[ウィンドウ]メニューの“プログラム一覧”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。



#### 24.5.1.4 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。

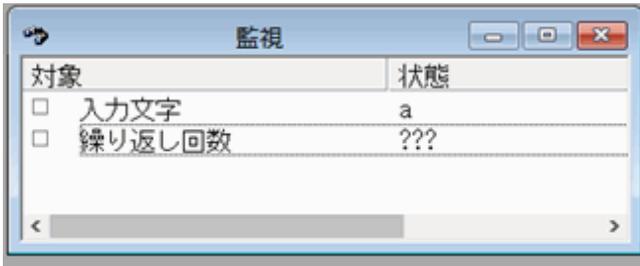
- 値の変更を監視しているデータ名およびその内容
- 条件の成立を監視している条件式およびその状態

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。

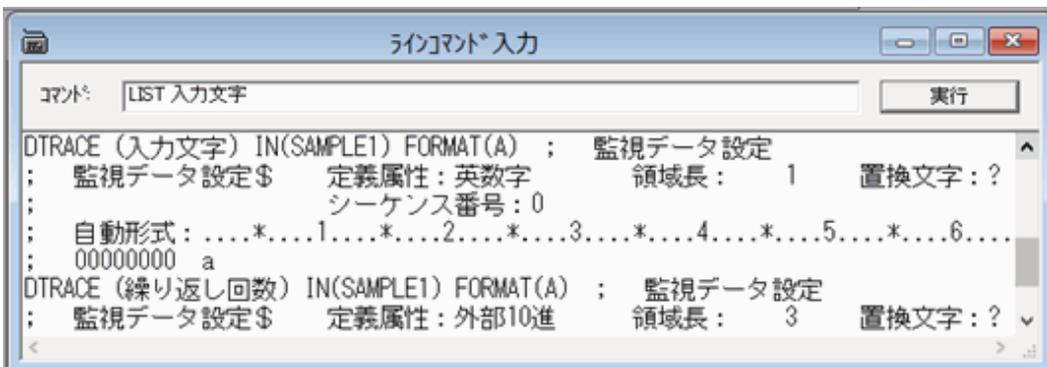


#### 24.5.1.5 ラインコマンド入力ウィンドウ

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。



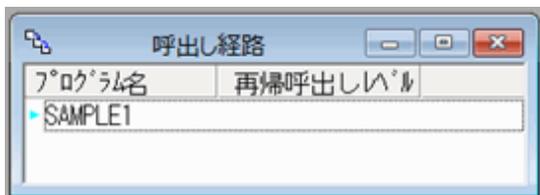
ラインコマンドの機能については、“[24.8 デバッガのコマンド一覧](#)”を参照してください。ラインコマンドの指定形式については、ヘルプを参照してください。

## 24.5.1.6 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。



## 24.5.1.7 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	優先度	デバッグ事象
1	0x0000166c	SAMPL...	0	0	通知する
2	0x00000858	0x77e...	0	0	通知する

### 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。

## 24.5.2 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。

### 24.5.2.1 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

#### 24.5.2.1.1 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 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]	

### 24.5.2.1.2 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 220行目で中断します]	
実行開始位置を SKIP 230	
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]	

### 24.5.2.1.3 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]

### 24.5.2.1.4 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. ←  
STOP RUN.  
:  
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+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]

### 24.5.2.1.5 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 PIC 9 VALUE 0.  
    :  
200 PROCEDURE DIVISION.  
    :  
250   CALL   "SUB" USING A.  
    :  
360 IDENTIFICATION DIVISION.  
350 PROGRAM-ID. SUB.  
    :  
410 LINKAGE SECTION.  
420 01 X PIC 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]	

### 24.5.2.1.6 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]	

### 24.5.2.2 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。

## 24.5.2.3 その他の補助的な機能

デバッガ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。

### 再デバッグ

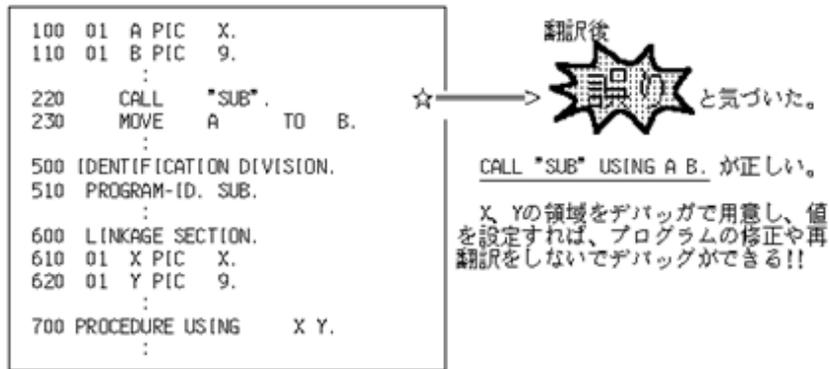
現在デバッグ中の利用者プログラムを強制的に終了させ、デバッガを終了しないで、同じプログラムのデバッグを再びスタートすることができます。

再デバッ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ッグ時に、呼出し元のプログラムから渡される連絡節のデータ領域を獲得できます。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ッグが可能で、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ッガによって解放されます。



## デバッグ環境の変更

デバッガでは、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、次回デバッガ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][“17.3.1 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][“24.6.2 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。

### ファイルへの書き込み

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。ファイルへの書き込みを使用すると、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】	【デバッグ】																										
<pre> : 100 LINKAGE SECTION. 110 COPY LINKCOPY. 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.           </pre>	<table border="1"> <thead> <tr> <th>メニューコマンド</th> <th>ラインコマンド</th> </tr> </thead> <tbody> <tr><td>[1] 中断点</td><td>BREAK 220</td></tr> <tr><td>[2] 実行</td><td>CONTINUE</td></tr> <tr><td>[3] (追加)</td><td>ASSIGNDATA LA INFL LINK</td></tr> <tr><td>[4] (追加)</td><td>ASSIGNDATA LB OUTFL LINK</td></tr> <tr><td>[5] (オフ)</td><td>OPENDATA INPUT LA</td></tr> <tr><td>[6] (オフ)</td><td>OPENDATA OUTPUT LB</td></tr> <tr><td>[7] (読み込み)</td><td>READDATA LA</td></tr> <tr><td>[8] [ ルートを確認するデバッグ操作 ]</td><td></td></tr> <tr><td>[9] (書き込み)</td><td>WRITEDATA LB</td></tr> <tr><td>[10] 実行</td><td>CONTINUE</td></tr> <tr><td>[11] (オン)</td><td>CLOSEDATA LA</td></tr> <tr><td>[12] (オン)</td><td>CLOSEDATA LB</td></tr> </tbody> </table> <p>( [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 )</p> <p>[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</p>	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] (オン)	CLOSEDATA LA	[12] (オン)	CLOSEDATA LB
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] (オン)	CLOSEDATA LA																										
[12] (オン)	CLOSEDATA LB																										
<pre> 【COPY句】 LINKCOPY 100 01 LINK. 110 02 A PIC 9. 120 02 B PIC 99.           </pre>																											
<p>【ファイルの内容】(COPY句) LINKCOPY</p> <table border="1"> <thead> <tr> <th></th> <th>INFL</th> <th>OUTFL</th> </tr> <tr> <th></th> <th>A B</th> <th>A B</th> </tr> </thead> <tbody> <tr> <td>(1)</td> <td>1</td> <td>1 11</td> </tr> <tr> <td>(2)</td> <td>2</td> <td>2 12</td> </tr> <tr> <td>⋮</td> <td>⋮</td> <td>⋮</td> </tr> <tr> <td>(n)</td> <td></td> <td></td> </tr> </tbody> </table>		INFL	OUTFL		A B	A B	(1)	1	1 11	(2)	2	2 12	⋮	⋮	⋮	(n)											
	INFL	OUTFL																									
	A B	A B																									
(1)	1	1 11																									
(2)	2	2 12																									
⋮	⋮	⋮																									
(n)																											

## 24.6 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。

### 24.6.1 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限り、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)
```

### 24.6.2 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。

## 24.6.3 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。

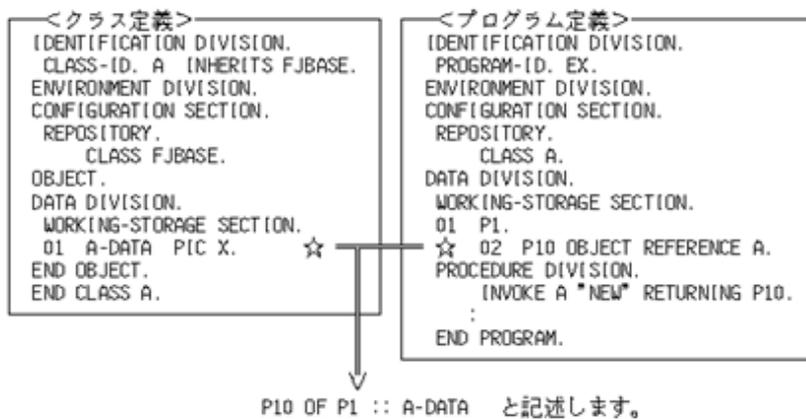
拡張した指定形式であるオブジェクト参照修飾子を用いた一意名を使用することにより、通常はプログラムから参照できないファクトリデータやオブジェクトデータを参照することができます。

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。

## 24.6.4 マルチスレッドでのデバッグの対応

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グの対応について説明します。

### 24.6.4.1 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。

デバッグ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。

### 24.6.4.2 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。

### 24.6.4.3 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なくなります。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。

### 24.6.4.4 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。

## 24.6.5 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。

### 24.6.5.1 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。

### 24.6.5.2 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。

## 24.7 注意事項

ここでは、対話型デバッグ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

デバッグは、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、デバッグ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

デバッグ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.3.38 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.3.39 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。

- ー 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。

- 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。

- ー 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。

- スタ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、システムが応答しなくなります。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つたとき、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つたとき、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。

- 一 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。
- 一 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。  
“24.2 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。
- 一 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。
- 一 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。

## 24.8 デバッグのコマンド一覧

デバッグのコマンド一覧を“表24.2 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24.2 デバッグのコマンド一覧

分類	コマンド	機能
実行	CONTINUE RUNTO SKIP RERUN	実行の再開 中断点指定実行 実行開始位置の変更 再デバッグ
中断点	BREAK DELETE	中断点の設定 中断点の解除
通過カウント点	COUNT DELCOUNT	通過カウント点の設定 通過カウント点の解除
データ	LIST SET LINKAGE ASSIGNDATA OPENDATA READDATA WRITEDATA CLOSEDATA	データ域の内容参照 データ域の内容変更 連絡節のデータ域の獲得 データファイルの割当て データファイルのオープン データファイルからの読み込み データファイルへの書き込み データファイルのクローズ
データ域の監視	MONITOR DELMON DTRACE DELDTR	データ域の変更時中断 データ域変更時中断の解除 データ域の変更の監視 データ域変更監視の解除
条件式の監視	DATACHK DELDCHK	条件式の成立の監視 条件式監視の解除
状態	ENV STATUS	デバッグ環境の設定/変更 デバッグ状態の表示

分類	コマンド	機能
	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

## 24.9 対話型リモートデバッグの使い方

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

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。

### 24.9.1 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(64)、Solaris(32)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(64)の場合、Win64リモートデバッグを使用
- サーバがSolaris(32)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。また、「Solaris(32)」は「Solaris」と表記します。

サーバがWindows(64)、SolarisまたはLinuxの場合には、必要に応じて以下のマニュアルの関連する項目を参照してください。

- Windows 64bit版 NetCOBOL の“NetCOBOL使用手引書(\*1)”
- Solaris 32bit版 NetCOBOL の“NetCOBOL使用手引書(\*1)”
- Linux 32bit版 NetCOBOLの“NetCOBOL 使用手引書”

\*1: NetCOBOL V11以降では、“NetCOBOL ユーザーズガイド”となります。



## 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。

### 24.9.1.1 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。

- 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。

- 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)
- 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。

### 24.9.1.2 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24.3 リモートデバッグ時の資産格納場所”に示します。

表24.3 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	○ (*5)	—
登録集原文	○ (*5)	—
画面帳票定義体	○ (*5)	○
サブスキーマ定義 (*4)	○ (*5)	—
コマンドファイル	○ (*5)	—
操作履歴ファイル	○ (*5)	—

\*1：サーバ側がWindowsまたはWindows(64)の場合

\*2：サーバ側がSolarisまたはLinuxの場合

\*3：サーバ側がWindows(64)の場合

\*4：サーバ側がWindowsの場合

\*5：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。

### 24.9.1.3 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“NetCOBOL ユーザーズガイド(\*1)”を参照してください。Win64リモートデバッグの場合は、“NetCOBOL Studio ユーザーズガイド”も合わせて参照してください。

\*1: NetCOBOL V10以前では、“NetCOBOL 使用手引書”となります。

## 24.9.2 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。

### 24.9.2.1 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(64)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、“[24.9.3.1 サーバ側のリモートデバッグコネクタの使い方](#)”および、“[24.9.3.5 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]“[24.9.5.2 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。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、リモートデバッガの[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]“[24.9.5 リモートデバッガの起動](#)”

4. リモートデバッガ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。

リモートデバッガの操作方法は通常のデバッガと同じです。使用方法については、“[24.9.6 リモートデバッガの操作](#)”を参照してください。

なお、サーバ側のリモートデバッガコネクタが起動している間は、リモートデバッガ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. リモートデバッガを終了させます。

[ファイル]メニューの“デバッガの終了”を選択することで、リモートデバッガを終了します。

6. サーバ側のリモートデバッガ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ガコネクタが終了します。

## 24.9.2.2 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ガ起動方法

デバッグしたいプログラムからリモートデバッガ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ガコネクタを起動します。

リモートデバッガコネクタは、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、“[24.9.3.4.1 クライアント側のリモートデバッガコネクタの起動方法](#)”および、“[24.9.3.5 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ガコネクタの[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]“[C.2.2 @CBR\\_ATTACH\\_TOOL\(プログラムからデバッグ機能を起動する指定\)](#)”

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]“[5.5.4 DLL配下にある実行用の初期化ファイルを使用する](#)”

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ガが起動します。

リモートデバッガが起動されると、クライアント側でリモート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。[参照]“[24.9.5 リモートデバッガの起動](#)”

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. リモートデバッガを使ってデバッグを行います。

リモートデバッガの操作方法是通常のデバッガと同じです。

操作方法については、“[24.5 デバッガの操作](#)”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ガ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. リモートデバッガを終了させます。

[ファイル]メニューの“デバッガの終了”を選択することで、リモートデバッガを終了します。

6. クライアント側のリモートデバッガ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ガコネクタのアイコンを右クリックして、表示されるメニューから“リモートデバッガコネクタの終了”を選択します。

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 ユーザーズガイド”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。

### 24.9.2.3 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ガ起動方法

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ガ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、“[24.9.2.2 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ガ起動方法](#)”と同じです。[参照]“[C.2.36 @CBR\\_JUSTINTIME\\_DEBUG \(異常終了時にデバッガまたは診断機能を使って調査を行う指定\)](#)”

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、リモートデバッガが起動され、リモート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。

## 24.9.3 リモートデバッガコネクタの使い方

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ガコネクタ

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。

“[24.9.2.1 一般的なプログラムに対するリモートデバッガ起動方法](#)”で使用します。

#### クライアント側のリモートデバッガコネクタ

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。

“[24.9.2.2 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ガ起動方法](#)”および、“[24.9.2.3 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ガ起動方法](#)”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、“[24.9.3.5 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。

### 24.9.3.1 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、“[24.9.2.1 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。

### 24.9.3.2 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

```
cobrd32 [ポート指定] [接続制限指定]
```

サーバがWindows(64)の場合

```
cobrd64 [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を処理した結果、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。
- リモートデバッグ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。
- リモートデバッグ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、リモートデバッグコネクタを起動する前に設定しておきます。
- リモートデバッグコネクタは、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グ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、リモートデバッグ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グコネクタの起動例

- ・ 複数の開発者が同時にデバッグを行う場合

```
cobrds32 -p 10001 -h client-1 ← 開発者-1用
cobrds32 -p 10002 -h client-2 ← 開発者-2用
cobrds32 -p 10003 -h client-3 ← 開発者-3用
```

- ー 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。
- ー -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。

## 24.9.3.3 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24.4 接続対象の指定形式(IPv4アドレスの場合)”および“表24.5 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24.4 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24.5 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、“[24.9.3.5 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、“[デバッグのIPv6対応](#)”を参照してください。



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。

### 24.9.3.4 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、“[24.9.2.2 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、“[24.9.2.3 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。

#### 24.9.3.4.1 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。



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 ユーザーズガイド](#)”を参照してください。

#### 24.9.3.4.2 [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

#### 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、“[24.9.3.5 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。

### 24.9.3.5 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24.6 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]“[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]“[24.9.3.3 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]“[24.9.3.4.2 \[リモートデバッグコネクタ\]ダイアログの設定方法](#)”

表24.6 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。 接続を許可するIPアドレスが、以下の場合 192.168.0.1	ALLOW=192.168.0.1 DENY=ALL	[接続を許可する対象]の設定 ・ [許可する対象]を選択します。 ・ エディットボックスに、以下の設定を行います。 192.168.0.1 [接続を拒否する対象]の設定 ・ [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。 接続を許可するIPアドレスが以下の場合 192.168.0.1 192.168.0.2 192.168.0.10	ALLOW=192.168.0.1 ¥ 192.168.0.2 192.168.0.10 DENY=ALL	[接続を許可する対象]の設定 ・ [許可する対象]を選択します。 ・ エディットボックスに、以下の設定を行います。 192.168.0.1 192.168.0.2 192.168.0.10 [接続を拒否する対象]の設定 ・ [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。 接続を許可するIPアドレスの範囲が以下の場合 下限 : 192.168.0.1 上限 : 192.168.0.10	ALLOW=192.168.0.1-192.168.0.10 DENY=ALL	[接続を許可する対象]の設定 ・ [許可する対象]を選択します。 ・ エディットボックスに、以下の設定を行います。 192.168.0.1-192.168.0.10 [接続を拒否する対象]の設定 ・ [すべての接続を拒否する]を選択します。
オクテッド単位に接続を許可する。 接続を許可するIPアドレスの範囲が以下の場合 下限 : 192.168.0.0 上限 : 192.168.0.255	ALLOW=192.168.0.* DENY=ALL	[接続を許可する対象]の設定 ・ [許可する対象]を選択します。 ・ エディットボックスに、以下の設定を行います。 192.168.0.* [接続を拒否する対象]の設定 ・ [すべての接続を拒否する]を選択します。
すべてのコンピュータからの接続を許可する。	ALLOW=ALL DENY=ALL	[接続を許可する対象]の設定 ・ [すべての接続を許可する]を選択します。 [接続を拒否する対象]の設定 ・ [すべての接続を拒否する]を選択します。

## 24.9.4 リモートデバッグの環境変数

リモートデバッグが使用する環境変数は、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通常のデバッグが使用する環境変数と同じです。

ここでは、SolarisリモートデバッグまたはLinuxリモートデバッグが使用する環境変数について説明します。

なお、通常のデバッグと同様に、SolarisリモートデバッグまたはLinuxリモートデバッグ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、“[24.5.2.2 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、“[24.3 デバッグの環境変数](#)”を参照してください。

### 24.9.4.1 @SVD\_COMPATIBILITY(互換性の指定)

SolarisリモートデバッグまたはLinuxリモート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、SolarisリモートデバッグまたはLinuxリモート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SolarisデバッグまたはLinux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。

#### Solarisリモートデバッグの場合

```
@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リモートデバッガまたはLinuxリモートデバッガの[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のリモートデバッガ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。

## 24.9.5 リモートデバッガの起動

ここでは、リモートデバッガ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。

### 24.9.5.1 [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、“[24.9.1.2 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

##### 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。

このパラメータは、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。

このパラメータは、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。

このパラメータは、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

### 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

### 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、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](#)”

### 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。

### 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。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。

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、Win32リモートデバッグまたはWin64リモートデバッグの場合は、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リモートデバッグまたはLinuxリモートデバッグの場合は、[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

### 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、“[24.5.2.2 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

### 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、“[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、“[24.9.2.2 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“[24.9.2.3 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、“[24.9.2.2 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“[24.9.2.3 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。

## 24.9.5.2 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、“24.9.2.2 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”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、“24.9.1.2 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24.7 起動パラメタ”に示します。

表24.7 起動パラメタ

指定形式	内容
-U 接続先	<p>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。</p> <pre>[IPアドレス ホスト名][:ポート番号]</pre> <p>IPアドレスは、IPv4またはIPv6の形式で指定します。</p> <p>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(初めの場合には、59998が指定されたと見なされます)。</p> <p>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。</p> <p>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。</p> <p>IPv6への対応仕様については、“デバッグのIPv6対応”を参照してください。</p> <p> <b>例</b></p> <p>.....</p> <p>[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</p> <p>[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</p>

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] 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] [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] server-1:2000</p> <p>[ポート番号省略の指定 (IPv4)] 192.168.0.1</p> <p>[ポート番号省略の指定 (IPv6)] fe80::1:23:456:789a</p> <p>.....</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	[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。 クライアント側のフォルダ構成に従って指定します。
-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。 クライアント側のフォルダ構成に従って指定します。
-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。 クライアント側のフォルダ構成に従って指定します。 本パラメタは、Win32リモートデバッグの場合のみ有効です。
-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。 クライアント側のフォルダ構成に従って指定します。
-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。
-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。 サーバ側のフォルダ構成に従って指定します。
-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。 クライアント側のフォルダ構成に従って指定します。
-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。 クライアント側のフォルダ構成に従って指定します。
-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。

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リモートデバッガまたはLinuxリモートデバッガの場合

— 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リモートデバッガまたはLinuxリモートデバッガの場合には、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。

## 24.9.6 リモートデバッガの操作

デバッガを開始した後の操作性は、通常のデバッガと同じです。[参照]“24.5 デバッガの操作”

ただし、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、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。

## 24.9.7 リモートデバッガの注意事項

ここでは、リモートデバッガ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。

### 24.9.7.1 Win32/Win64リモートデバッガの注意事項

ここでは、Win32リモートデバッガまたはWin64リモートデバッガ使用時の注意事項について説明します。

通常のデバッガと共通の注意事項については、“24.6.5.2 Unicodeを扱う上での注意事項”および、“24.7 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]“24.9.1.2 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。

### 24.9.7.2 Solaris/Linuxリモートデバッガの注意事項

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リモートデバッガまたはLinuxリモートデバッガは、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ガ起動方法の場合

サーバ側のリモートデバッガ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ガ起動方法の場合

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リモートデバッガまたはLinuxリモートデバッガでは、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

通常のリモートデバッガと共通の注意事項については、各OS版の“NetCOBOL ユーザーズガイド(\*1)”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]“[24.9.1.2 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。

\*1: NetCOBOL V10以前では、“NetCOBOL 使用手引書”となります。

## 第25章 ツールの操作

プロジェクトマネージャの[ツール]メニューから以下の操作ができます。

操作	説明	関連記事
[翻訳]	WINCOBを起動します。	“ <a href="#">25.1 WINCOBコマンド</a> ”
[リンク]	WINLINKを起動します。	“ <a href="#">25.2 WINLINKコマンド</a> ”
[デバッグ]	デバッグ種別で選択した対話型デバッグを起動します。	“ <a href="#">第24章 対話型デバッグの使い方</a> ”
[デバッグ種別] > [Win32]	Win32デバッグを起動することを指定します。	
[デバッグ種別] > [Win32リモート]	Win32リモートデバッグを起動することを指定します。	
[デバッグ種別] > [Solarisリモート]	Solarisリモートデバッグを起動することを指定します。	“ <a href="#">24.9 対話型リモートデバッグの使い方</a> ”
[デバッグ種別] > [Linuxリモート]	Linuxリモートデバッグを起動することを指定します。	
[リモートデバッグコネクタ]	リモートデバッグコネクタを起動します。	
[実行]	WINEXECを起動します。	“ <a href="#">25.3 WINEXECコマンド</a> ”
[実行環境設定ツール]	実行環境設定ツールを起動します。	“ <a href="#">5.7 実行環境設定ツール</a> ”
[ファイルユーティリティ]	COBOLファイルユーティリティを起動します。	“ <a href="#">7.8 COBOLファイルユーティリティ</a> ”
[イベントログ用レジストリキー]	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・削除を起動します。	“ <a href="#">1.2.9 COB_REPORT_EVENT(イベントログ出力サブルーチン)</a> ”
[エディタ]	エディタを起動します。	エディタのヘルプ
[クラスデータベース] > [生成]	クラス情報データベースを生成します。	“ <a href="#">27.5.3 クラス情報データベースの生成</a> ”
[クラスデータベース] > [オプション設定]	クラスデータベースのオプションを設定します。	“ <a href="#">27.5.4 オプション設定ウィンドウ</a> ”
[クラスブラウザ]	クラスブラウザを起動します。	“ <a href="#">27.5 クラスブラウザ</a> ”
[プロジェクトブラウザ]	プロジェクトブラウザを起動します。	“ <a href="#">27.6 プロジェクトブラウザ</a> ”
[Webパラメタ取得プログラム生成]	呼出し用ページからWebパラメタを取得するプログラムを自動生成します。	“ <a href="#">26.4 Webパラメタ取得プログラム生成</a> ”
[Web結果ページ出力プログラム生成]	結果出力用ページを出力するプログラムを自動生成します。	“ <a href="#">26.5 結果ページ出力プログラム生成</a> ”
[環境変数表示]	COBOLコンパイルに関する環境変数情報を表示します。	—
[複数プロジェクトビルド]	複数のプロジェクトを一括してビルドします。	プロジェクトマネージャのヘルプ
[IDL-COBOL変換] > [IDL->COBOL変換]	IDLソースファイルをCOBOL登録ファイルに変換します。	“ <a href="#">NetCOBOLユーザーズガイド(OSIV分散開発・プロジェクトマネージャ編)</a> の“IDL-COBOL変換機能”

操作	説明	関連記事
[IDL-COBOL変換] > [COBOL->IDL変換]	COBOL登録集ファイルをIDLソースファイルに変換します。	“NetCOBOLユーザーズガイド(OSIV分散開発・プロジェクトマネージャ編)”の“COBOL-IDL変換機能”

## 25.1 WINCOBコマンド

WINCOBコマンドはCOBOLコンパイラを起動するための簡易操作ビュー (WINCOBウィンドウ)を提供します。WINCOBウィンドウを使用することにより、COBOLコンパイラの起動を簡単な操作で行うことができます。

### 25.1.1 WINCOBウィンドウ

#### WINCOBウィンドウの起動

WINCOBウィンドウは、プロジェクトマネージャの[ツール]メニューから[翻訳]を選択して起動します。

- WINCOBウィンドウの[モード]メニューから[単数翻訳]を選択すると、単数翻訳ウィンドウが表示されます。(初期設定では単数翻訳が選択されています。)
- WINCOBウィンドウの[モード]メニューから[複数翻訳]を選択すると、複数翻訳ウィンドウが表示されます。

表25.1 WINCOBウィンドウ(単数翻訳ウィンドウ)

項目	説明
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。
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。 [参照]ボタンは、ファイルを参照するときに選択します。
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。

表25.2 WINCOBウィンドウ(複数翻訳ウィンドウ)

項目	説明
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。
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。
一覧	翻訳対象とするファイルを表示します。
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。
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。

項目	説明
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。

## ウィンドウの操作

- ・ [ファイル名の指定](#)
- ・ [翻訳オプションの設定](#)
- ・ [主プログラムの指定](#)
- ・ [翻訳の開始と終了](#)

## 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では異なります。

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。

## 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]“[25.1.4 \[翻訳オプション\]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。

## 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、“11.1.4 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

## 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]“23.4.6 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。

## 25.1.2 WINCOBを使用した翻訳の手順

ここでは、WINCOBを使って翻訳するときの手順について説明します。

### 25.1.2.1 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]“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]“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]“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]“登録集名の設定”
5. 翻訳を開始します。[参照]“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]“23.4.6 翻訳エラーの修正”
7. 翻訳を終了します。[参照]“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。

### 25.1.2.2 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]“[WINCOBウィンドウの起動](#)”
2. WINCOBウィンドウの[モード]メニューから[複数翻訳]を選択します。
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]“[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]“[翻訳オプションの設定](#)”
5. 登録集名を設定します。[参照]“[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](#)”

6. 主プログラムを指定します。[参照]“[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。

7. 翻訳を開始します。[参照]“[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。
8. 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]“[23.4.6 翻訳エラーの修正](#)”
9. 翻訳を終了します。[参照]“[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。

### 25.1.3 サンプルプログラムの翻訳

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格納されています。ここでは、WINCOBを使って、Sample01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¥COBOL¥Sample01¥Sample1.cobB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. [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。

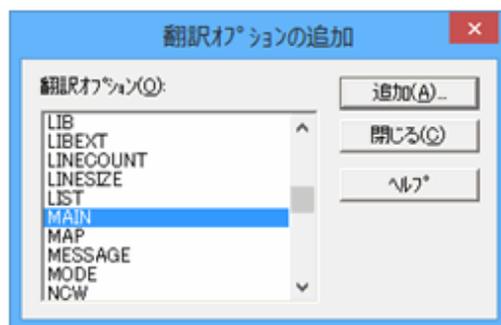
3. [参照]ボタンを使って、ソースファイルのエディットボックスにサンプルプログラムの格納先ファイル名を指定します。ファイル名は、エディットボックスに直接キー入力することもできます。



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。

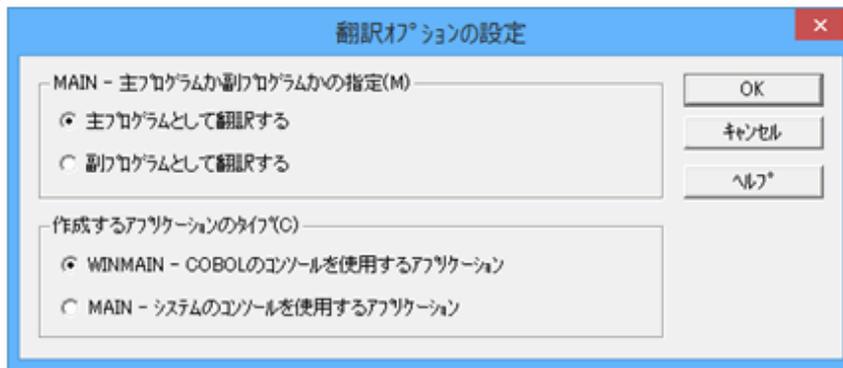


5. [追加]ボタンをクリックします。  
→ [翻訳オプションの追加]ダイアログが表示されます。



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]ダイアログが閉じられます。

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]ダイアログが閉じられます。
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]ダイアログが閉じられます。

10. WINCOBウィンドウの[翻訳]ボタンをクリックします。  
→ 翻訳が開始されます。
11. 翻訳が終了すると、ビルダウィ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ウインドウが閉じられます。  
サンプルプログラムの格納先(ここでは、C:\¥COBOL¥Samples¥COBOL¥Sample01)に、目的プログラム(Sample1.obj)が作成されているか確認してください。

## 25.1.4 [翻訳オプション]ダイアログ

[翻訳オプション]ダイアログは、翻訳オプションを設定するためのダイアログボックスです。

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アーリ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、“[28.8.1 COMサーバ名の設定](#)”を参照してください。

表25.3 [翻訳オプション]ダイアログ

項目	説明
オプションファイル	編集中のオプションファイル名が表示されます。
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。
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。
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。
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	ヘルプウインドウを開くときに選択します。

### 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。

#### 25.1.4.1 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]ダイアログが閉じられます。
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]ダイアログが閉じられます。
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。

#### 25.1.4.2 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]ダイアログが閉じられます。
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。

#### 25.1.4.3 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。  
→ 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。

### 25.1.5 [登録集名の設定]ダイアログ

[登録集名の設定]ダイアログは、登録集名を設定するためのダイアログボックスです。

表25.4 [登録集名の設定]ダイアログ

項目	設定
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。
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。
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。

#### 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.3.25 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。



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.3.2 ALPHAL \(英小文字の扱い\)](#)”

### 25.1.5.1 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. [登録集名]にCOBOLソースプログラムのCOPY文に記述した登録集名を、[登録集格納先]に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。

### 25.1.5.2 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]ダイアログが閉じられます。
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。

### 25.1.5.3 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。  
→ 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。

### 25.1.5.4 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。

## 25.1.6 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。



- 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]“[主プログラムの指定](#)”

## 25.2 WINLINKコマンド

WINLINKコマンドはリンクを起動するための簡易操作ビュー (WINLINKウィンドウ) を提供します。WINLINKウィンドウを使用することにより、リンクの起動を簡単な操作で行うことができます。

### 25.2.1 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したり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。

#### WINLINKウィンドウの起動

WINLINKウィンドウは、プロジェクトマネージャの[ツール]メニューから[リンク]を選択して起動します。

#### 25.2.1.1 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。

表25.5 WINLINK[リンク]ウィンドウ

項目	説明
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。
ファイル(T)	作成するファイル名を指定します。
種別	ターゲットの種類 (EXEまたはDLL) を選択します。
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。
一覧	リンクの対象となるファイルを表示します。
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。
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。

#### ウィンドウの操作

- “[25.2.1.1.1 ターゲット名の指定](#)”
- “[25.2.1.2.2 ファイルの指定](#)”

- “[25.2.1.1.3 インポートライブラリの作成](#)”
- “[25.2.1.1.4 リンクオプションの設定](#)”
- “[25.2.1.1.5 モジュール定義ファイルの生成](#)”
- “[25.2.1.1.6 リンクの開始と終了](#)”

### 25.2.1.1.1 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)]エディットボックスに指定します。

#### [追加]ボタンを使う方法

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、[ファイル(T)]のエディットボックスに、“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”が表示されます。

#### 直接入力する方法

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。

### 25.2.1.1.2 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。
  - オブジェクトファイル
  - ライブラリ
  - 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。

### 25.2.1.1.3 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。

### 25.2.1.1.4 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]“[J.2.1 LINKコマンドのオプション](#)”

表25.6 [リンクオプション]ダイアログ

項目	説明
リンクオプション	リンクオプションを指定します。
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を指定する時に選択します。 [マップ]ボタンをクリックすると、エディットボックスに/MAPが表示されます。
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、MSVCRT.LIB以外のライブラリをリンクする場合に指定します。
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(MSVCRT.LIB)に戻すときに選択します。
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(COBOL単体用)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(言語間結合用)かを選択します。
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。

### 25.2.1.1.5 モジュール定義ファイルの生成

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内容は、“4.4.1 モジュール定義ファイルの内容”を参照してください。

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が、WINLINK[リンク]ウィンドウを使用して、自動的に生成することもできます。

#### 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を自動生成する場合、[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。これにより、動的リンクライブラリファイルに対応したモジュール定義ファイルが自動生成され、動的リンクライブラリファイルの存在するフォルダ配下に登録されます。



#### 注意

- ・ モジュール定義ファイルは、“動的リンクライブラリ名. DEF”というファイル名で作成されます。
- ・ “動的リンクライブラリ名. DEF”がすでに存在する場合は生成しません。
- ・ 作成したモジュール定義ファイルのモジュール定義文"EXPORTS"にはソースファイル名もしくはオブジェクトファイル名が指定されます。エディタで正しい名前に変更してください。
- ・ 空白およびセミコロン(';')はリンクの外部シンボル名として使用できません。ファイル名にこれらの文字がある場合は、自動生成されたモジュール定義ファイルのEXPORTS文を必ず修正してください。

#### 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

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]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。

### 25.2.1.1.6 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。  
→ ビルダウイ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。  
→ ビルダウイ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、“NetCOBOL メッセージ集”の“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]ウインドウの[キャンセル]ボタンをクリックし、WINLINK[リンク]ウイ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] “NetCOBOL メッセージ集”の“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。

### 25.2.1.2 WINLINK[ライブラリモジュール結合]ウインドウ

WINLINK[ライブラリモジュール結合]ウインドウは、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。

表25.7 WINLINK[ライブラリモジュール結合]ウインドウ

項目	説明
コマンド	ウイ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]ウインドウのヘルプを表示します。
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。
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。
ファイル	[一覧]リストボックスに追加するファイルを指定します。
一覧	結合対象となるファイルを表示します。
参照(R)	ファイルを参照するときに選択します。
追加	[一覧]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	[一覧]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。
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]ウインドウを閉じます。
キャンセル	WINLINK[ライブラリモジュール結合]ウインドウを閉じます。
結合	ライブラリの作成を開始するときに選択します。

#### ウインドウの操作

- “25.2.1.2.1 ライブラリ名の指定”

- “25.2.1.2.2 ファイルの指定”
- “25.2.1.2.3 ライブラリ結合の開始と終了”

### 25.2.1.2.1 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。

#### [追加]ボタンを使う方法

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]リストボックスにオブジェクトファイル名が追加され、“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。

#### 直接入力する方法

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。

### 25.2.1.2.2 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]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]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.の操作を繰り返して行ってください。



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。

### 25.2.1.2.3 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。  
→ ビルダウィ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。  
→ ビルダウィ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て行ってください。  
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。

## 25.2.2 WINLINKを使用したリンクの手順

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NetCOBOL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、“[4.5 リンクコマンドを使ったリンク操作](#)”で説明しています。



注意

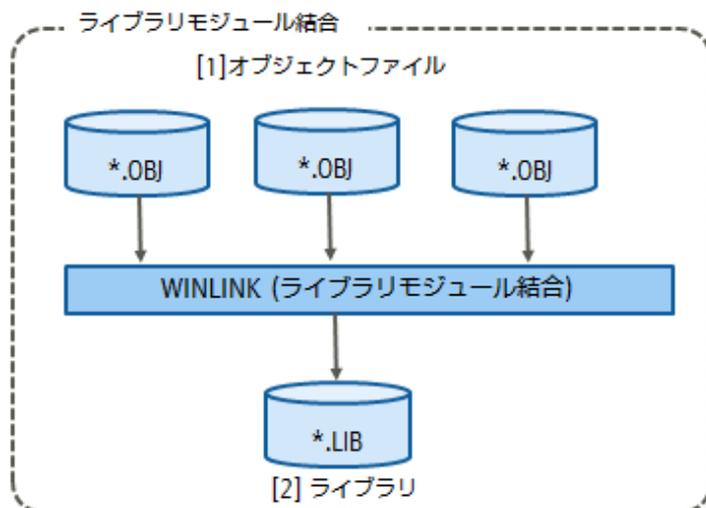
WINLINKを使ってリンクを行う場合、環境変数TMPに作業用のフォルダを設定しておく必要があります。

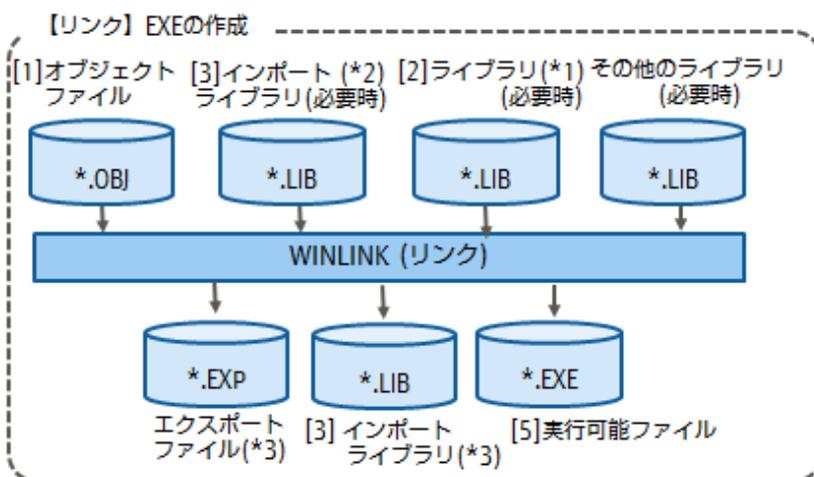
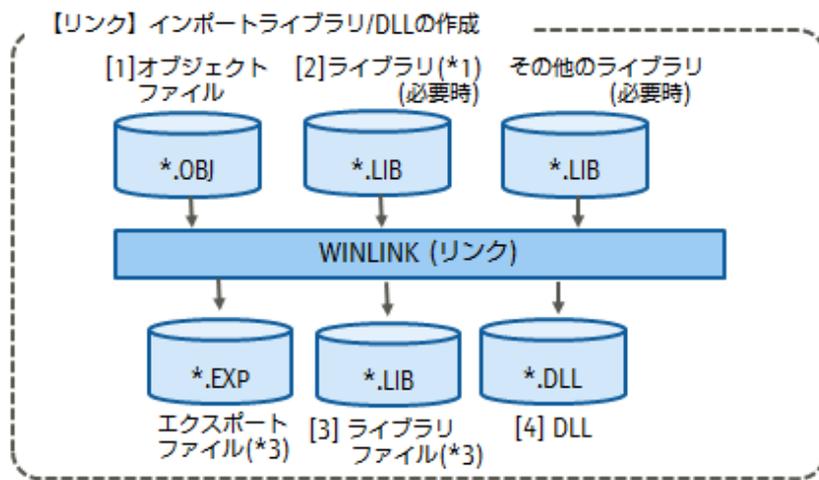
### 25.2.2.1 WINLINKが使用するファイル

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25.8 WINLINKが使用するファイル”に示します。

表25.8 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。
[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。

	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
[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」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、NetCOBOL Studioのデバッグ機能、対話型デバッガ、クラッシュダンプ、Visual C++のデバッグ機能などが使用します。

## 25.2.2.2 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]“[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]“[25.2.1.1.1 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)]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]“[25.2.1.1.2 ファイルの指定](#)”
5. リンクオプションを設定します。[参照]“[25.2.1.1.4 リンクオプションの設定](#)”
6. リンクを開始します。[参照]“[25.2.1.1.6 リンクの開始と終了](#)”
7. リンクを終了します。[参照]“[25.2.1.1.6 リンクの開始と終了](#)”

## 25.2.2.3 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]“[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]“[25.2.1.1.1 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]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)]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]リストボックスに指定します。[参照]“[25.2.1.1.2 ファイルの指定](#)”
5. リンクオプションを設定します。[参照]“[25.2.1.1.4 リンクオプションの設定](#)”
6. リンクを開始します。[参照]“[25.2.1.1.6 リンクの開始と終了](#)”
7. リンクを終了します。[参照]“[25.2.1.1.6 リンクの開始と終了](#)”

## 25.2.2.4 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]“[25.2.2.7 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]“[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]“[25.2.1.1.1 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]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)]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]リストボックスに指定します。[参照]“[25.2.1.1.2 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]“[25.2.1.1.4 リンクオプションの設定](#)”
7. リンクを開始します。[参照]“[25.2.1.1.6 リンクの開始と終了](#)”
8. リンクを終了します。[参照]“[25.2.1.1.6 リンクの開始と終了](#)”

## 25.2.2.5 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]“[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]“[25.2.1.1.1 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]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)]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]リストボックスに指定します。[参照]“[25.2.1.1.2 ファイルの指定](#)”
5. リンクオプションを設定します。[参照]“[25.2.1.1.4 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]“[25.2.1.1.6 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。



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]“[25.2.1.1.1 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]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)]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]リストボックスに指定します。[参照]“25.2.1.1.2 ファイルの指定”
10. リンクオプションを設定します。[参照]“25.2.1.1.4 リンクオプションの設定”
11. リンクを開始します。[参照]“25.2.1.1.6 リンクの開始と終了”
12. リンクを終了します。[参照]“25.2.1.1.6 リンクの開始と終了”

### 25.2.2.6 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]“25.2.2.5 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]“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]“25.2.1.1.1 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]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)]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]リストボックスに指定します。[参照]“25.2.1.1.2 ファイルの指定”
6. リンクオプションを設定します。[参照]“25.2.1.1.4 リンクオプションの設定”
7. リンクを開始します。[参照]“25.2.1.1.6 リンクの開始と終了”
8. リンクを終了します。[参照]“25.2.1.1.6 リンクの開始と終了”

### 25.2.2.7 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]“WINLINKウィンドウの起動”
2. [コマンド]メニューから[ライブラリモジュール結合]を選択して、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。
3. ライブラリ名を指定します。[参照]“25.2.1.2.1 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]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。

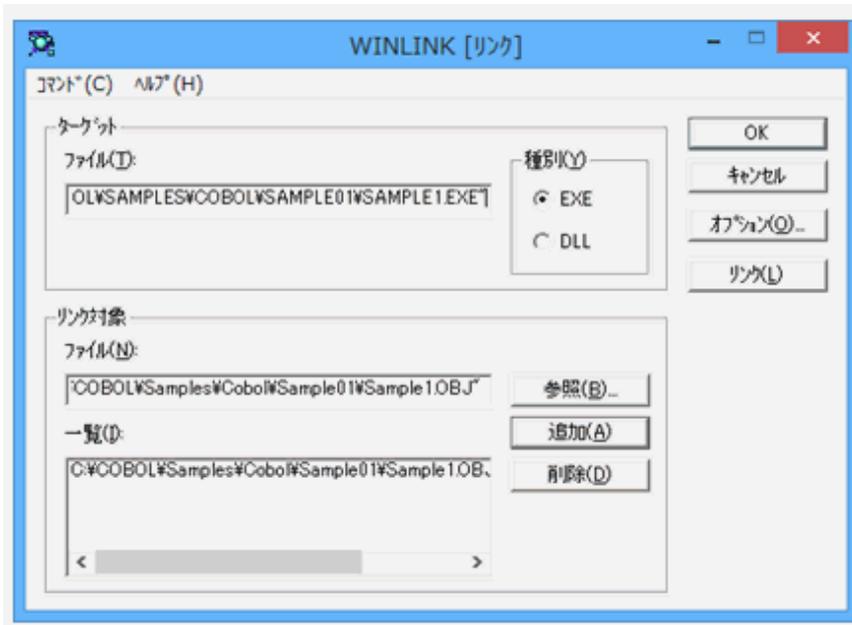
4. ライブラリ結合に必要なその他のファイル(ライブラリ)を[一覧]リストボックスに指定します。[参照]“25.2.1.1.2 ファイルの指定”
5. ライブラリ結合を開始します。[参照]“25.2.1.2.3 ライブラリ結合の開始と終了”
6. ライブラリ結合を終了します。[参照]“25.2.1.2.3 ライブラリ結合の開始と終了”

## 25.2.3 サンプルプログラムのリンク

ここでは、WINLINKを使って、Sample01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¥COBOL¥Sample01¥Sample1.OBJ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. [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。
3. 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)]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]リストボックスに、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。
4. 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。
  5. 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(ここでは、C:\COBOL\Samples\Cobol\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。

## 25.3 WINEXECコマンド

WINEXECコマンドはCOBOLアプリケーションを起動するための簡易操作ビュー(WINEXECウィンドウ)を提供します。WINEXECウィンドウを使用することにより、COBOLアプリケーションの起動を簡単な操作で行うことができます。

### 25.3.1 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。

#### WINEXECウィンドウの起動

WINEXECウィンドウは、プロジェクトマネージャの[ツール]メニューから[実行]を選択して起動します。

表25.9 WINEXECウィンドウ

項目	説明
ヘルプ	WINEXECウィンドウのヘルプを表示します。

項目	説明
コマンドライン	実行するファイルを指定します。
参照	ファイルを参照するときに選択します。
実行履歴	実行したファイルの履歴を表示します。
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。
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。

## ウィンドウの操作

- 25.3.1.1 ファイル名の指定
- 25.3.1.2 実行の開始と終了

### 25.3.1.1 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。

### 25.3.1.2 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]“5.7 実行環境設定ツール”)などで設定しておいてください。

## 25.3.2 WINEXECを使用した実行の手順

WINEXEC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

“WINCOB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

“25.3.1.1 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

“25.2.1.1.6 リンクの開始と終了”

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

“25.3.1.2 実行の開始と終了”

### 25.3.3 サンプルプログラムの実行

ここでは、WINEXECを使って、Sample01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¥COBOL¥Sample01¥SAMPLE1.EXEとします。

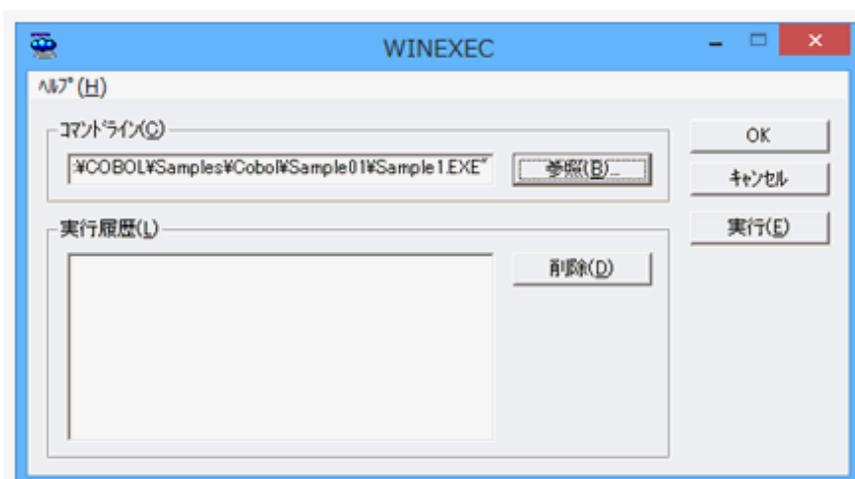
Sample01では、特に設定が必要な実行環境情報はありますが、実行環境情報を設定する場合は、プログラムの実行前に以下の操作を行います。

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. [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。[参照]“5.7 実行環境設定ツール”
3. [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。
4. [ファイル名]に、「C:\¥COBOL¥Samples¥COBOL¥Sample01¥COBOL85.CBR」を指定し、[開く]ボタンをクリックします。
5. [実行環境設定ツール]ダイアログで必要な実行環境情報を入力し、[適用]ボタンをクリックします。  
→ 実行用の初期化ファイル(C:\¥COBOL¥Samples¥COBOL¥Sample01¥COBOL85.CBR)の内容が変更されます。(この例では、特に入力する情報はありません。)
6. [ファイル]メニューから[閉じる]を選択します。
7. [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。

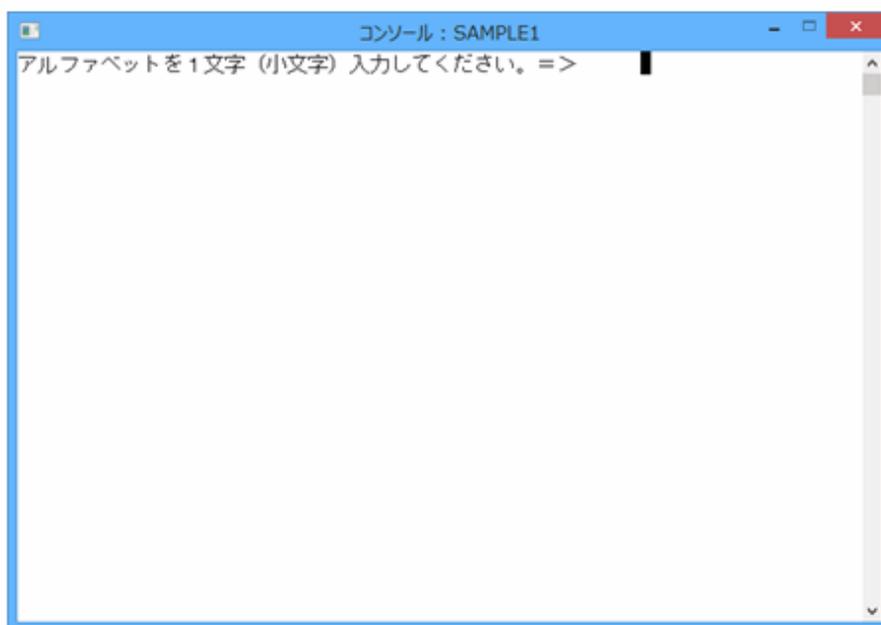
では、実際に実行可能ファイルを実行してみましょう。

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。

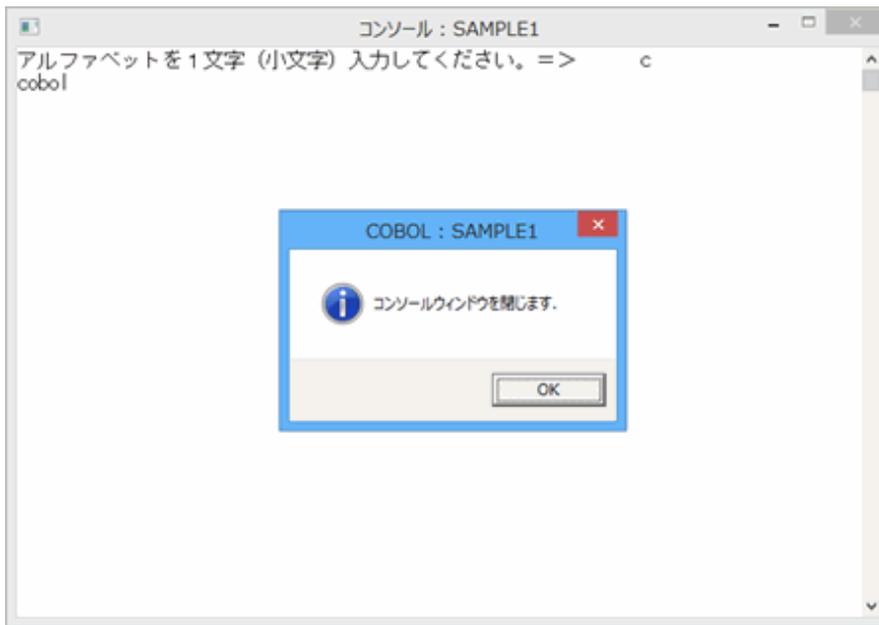
2. 実行するファイルの名前(SAMPLE1.EXE)をコマンドラインのエディットボックスに指定します。



3. [実行]ボタンをクリックします。
4. Sample01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。

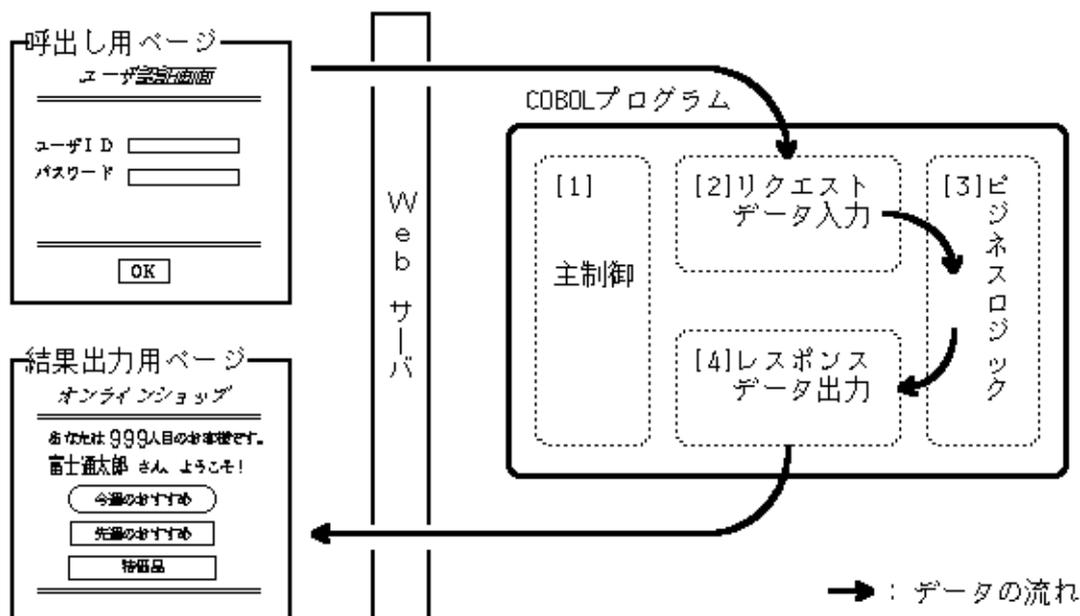


5. 実行結果を確認したら、メッセージボックスの[OK]ボタンをクリックします。  
→ メッセージボックスおよびコンソールウィンドウが閉じます。

## 第26章 Webアプリケーション開発支援機能の利用

### 26.1 Webアプリケーション開発支援機能の概要

Webアプリケーションは、しばしば複数のHTML文書とそれに対応付けられたプログラムから構成される大規模なアプリケーションとなります。しかし、ほとんどの場合において次のような単純な構成のプログラムが複数集まったものと見ることができます。



このため、Webアプリケーションの開発では次の作業を繰り返すことになります。

1. 呼出し用ページ(Webパラメタ)の作成
2. 呼出し用ページとCOBOLプログラムとの関連付け
3. セッション管理等のアプリケーションの制御ロジックの作成
4. ビジネスロジックの作成
5. 結果出力用ページ(変換名の指定を含む)の作成
6. 結果出力用ページとCOBOLプログラムとの関連付け

Webアプリケーション開発支援機能では、上記の2、3、6の作業を軽減するために、次の3つの支援機能を提供します。

#### Webアプリケーションウィザード

Webアプリケーション作成のためのプロジェクトと雛形となるプログラム(上の図の[1]、[2]、[4]に該当する部分)を生成します。

また、必要に応じて呼出し用ページと結果出力用ページの雛形を生成します。

#### Webパラメタ取得プログラム生成

呼出し用ページを入力し、このHTML文書に含まれる入力フォームからWebパラメタを入力するプログラム(上の図の[2]に該当する部分)を生成します。

#### Web結果ページ出カプログラム生成

結果出力用ページのHTML文書を入力し、このHTML文書に含まれる変換名の置き換え、HTML文書を出力するプログラム(上の図の[4]に該当する部分)を生成します。

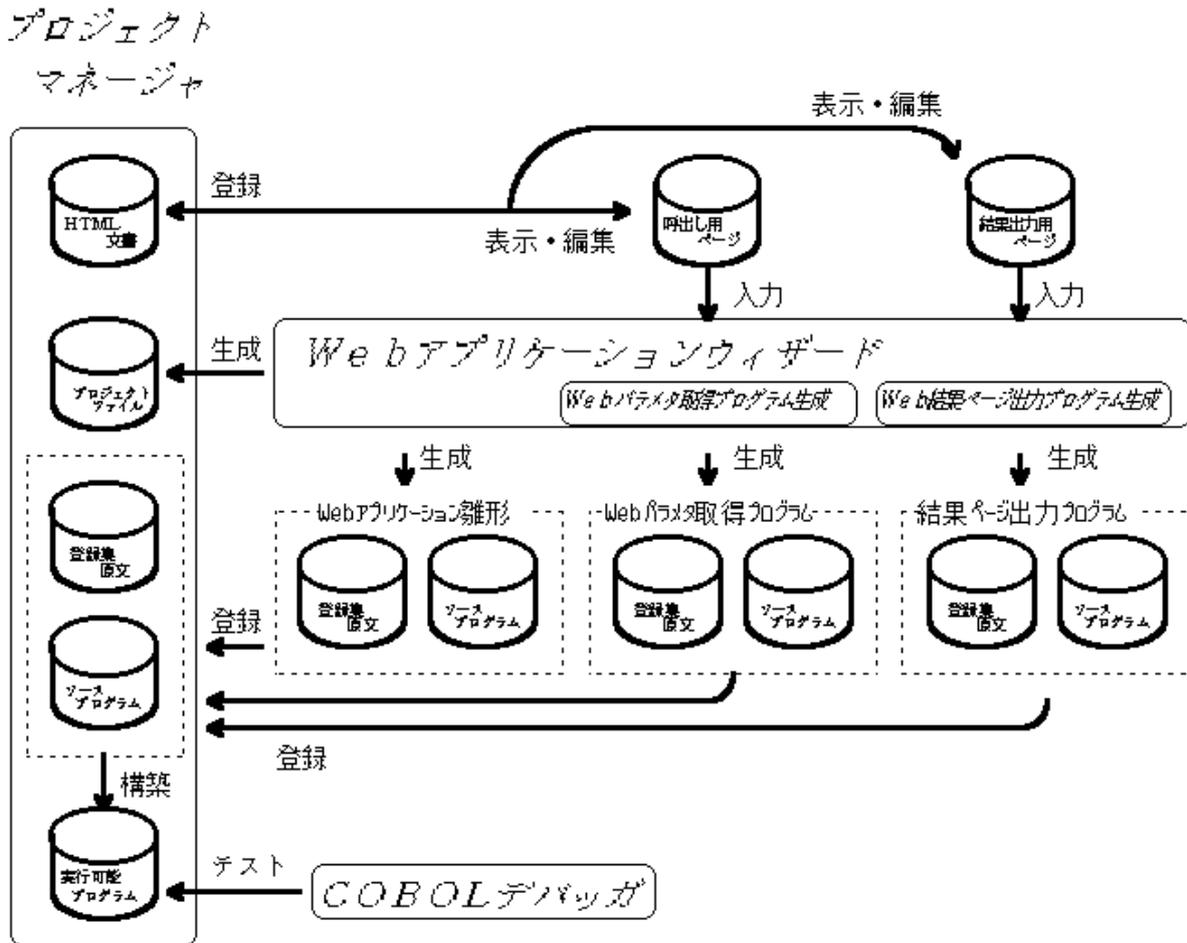
これらはプロジェクトマネージャの一部として動作し、アプリケーションの開発からテスト、資産管理についても従来のCOBOL開発環境と同等の機能を提供します。

## 注意

HTML文書もプログラム資産の一部としてプロジェクトに登録することができます。これによりプロジェクトマネージャからHTML文書の参照、編集などの操作が可能になりますが、これらはプロジェクト管理機能の依存関係チェックの対象にはなりません。

Webアプリケーション開発支援機能の概要を“図26.1 Webアプリケーション開発支援機能の概要”に示します。

図26.1 Webアプリケーション開発支援機能の概要



Webアプリケーション開発支援機能では、次に示すインターフェースを使用するWebアプリケーションの開発が可能です。

### CGI(Common Gateway Interface)

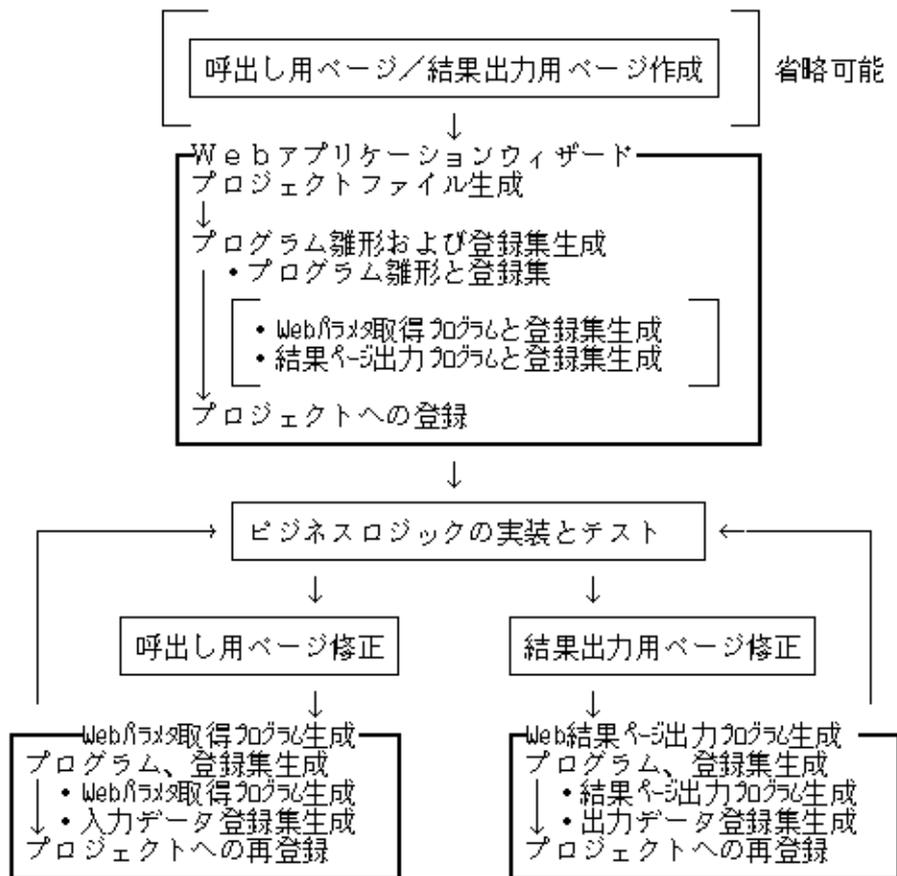
ほとんどのWebサーバが標準でサポートするインターフェースで、実行形式プログラム(EXE)で作成します。COBOLでCGIのWebアプリケーションを作成する場合には、COBOL CGIサブルーチンを使用します。詳細は、“19.4 CGIサブルーチンを使用したアプリケーションの作成と実行”を参照してください。

### ISAPI(Internet Server API)

Microsoft(R) Internet Information Server(IIS)などでサポートされるインターフェースで、ダイナミックリンクライブラリ(DLL)で作成します。COBOLでISAPIのWebアプリケーションを作成する場合には、COBOL ISAPIサブルーチンを使用します。詳細は、“19.5 ISAPIサブルーチンを使用したアプリケーションの作成と実行”を参照してください。

## 26.2 Webアプリケーション開発支援機能を使用した作業の流れ

Webアプリケーション開発支援機能を使用したアプリケーション開発は、次の手順で行います。



太線で囲んだ部分の処理が、Webアプリケーション開発支援機能が提供する機能で処理される部分です。

『ビジネスロジックの実装とテスト』の部分は通常のCOBOLアプリケーションの開発と同様の方法で行うことができます。

## 26.3 Webアプリケーションウィザード

Webアプリケーションウィザードは、画面から入力された情報をもとにWebアプリケーション開発用のプロジェクトとWebアプリケーションの雛形となるプログラムを生成するツールです。

以下のような情報を設定するだけで、ビルド可能なプロジェクトと基本的なWebアプリケーションの雛形を作成することができます。

- ・ プロジェクトファイル名と格納場所
- ・ 実行ファイル名
- ・ 使用するAPIの種類

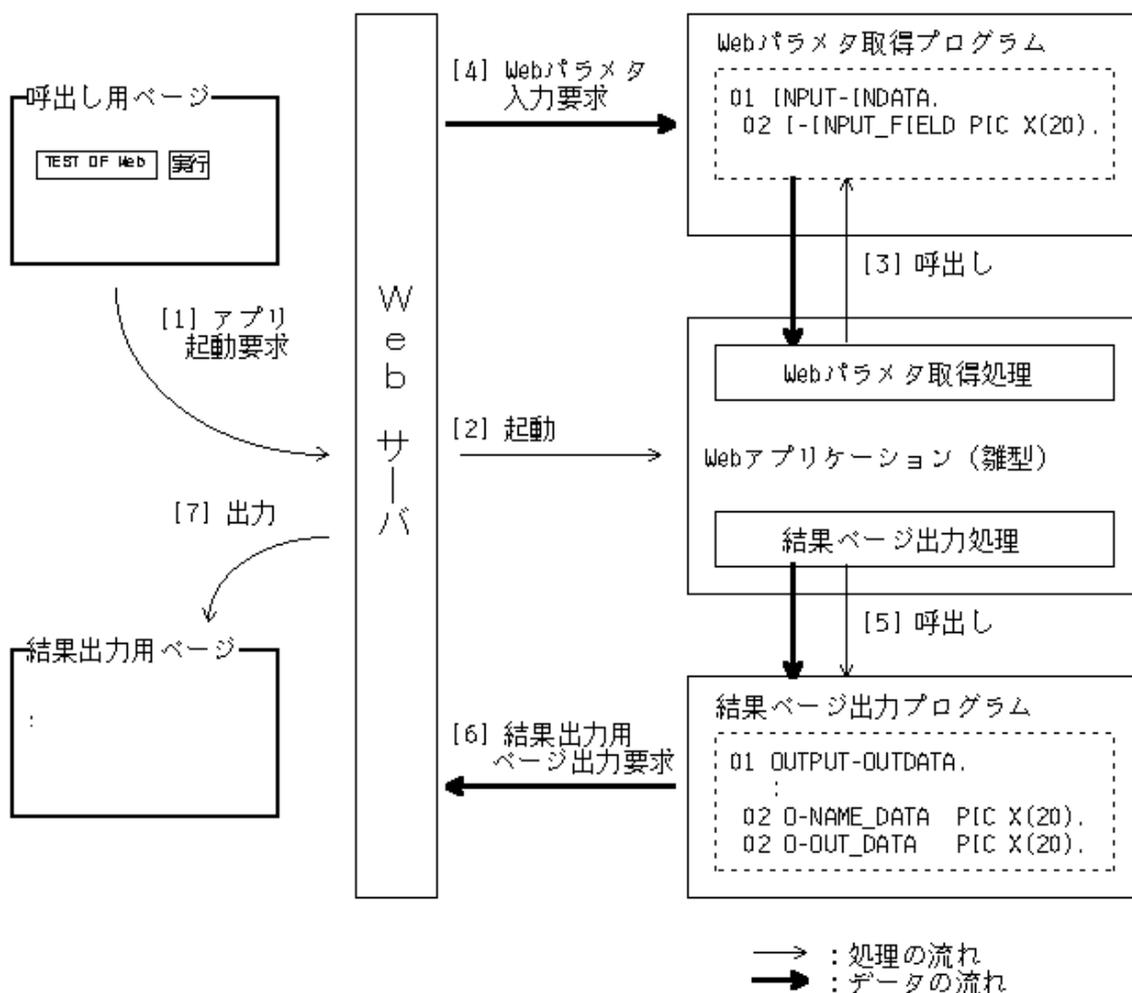
また、呼び出し用ページや結果出力用ページを指定して、データの入出力を行うプログラムを生成したり、新たに呼び出し用ページや結果出力用ページの雛形を生成したりすることもできます。

## 26.3.1 Webアプリケーション

ここではWebアプリケーションウィザードが生成するWebアプリケーションの雛形について説明します。

### Webアプリケーションの雛形

次のように動作するWebアプリケーションの雛形(呼び出し用ページおよび結果出力用ページの登録で“雛形を生成する”を選択した例です。詳細は、“26.3.2.2 Webアプリケーションウィザードの画面と操作”を参照してください)を生成します。



[1] 呼び出し用ページの入力フォームにデータを入力し、実行ボタンを押すとWebサーバにWebアプリケーションの起動要求が送られ、同時にデータが送信されます。

[2] WebサーバによってWebアプリケーションが起動されます。

[3] Webアプリケーションは必要な初期化処理などを行った後、Webパラメタ入力用のプログラムを呼び出します。

[4] Webパラメタ入力用のプログラムはWebサーバに要求を出して、呼び出し用ページから送信されたWebパラメタを取得します。

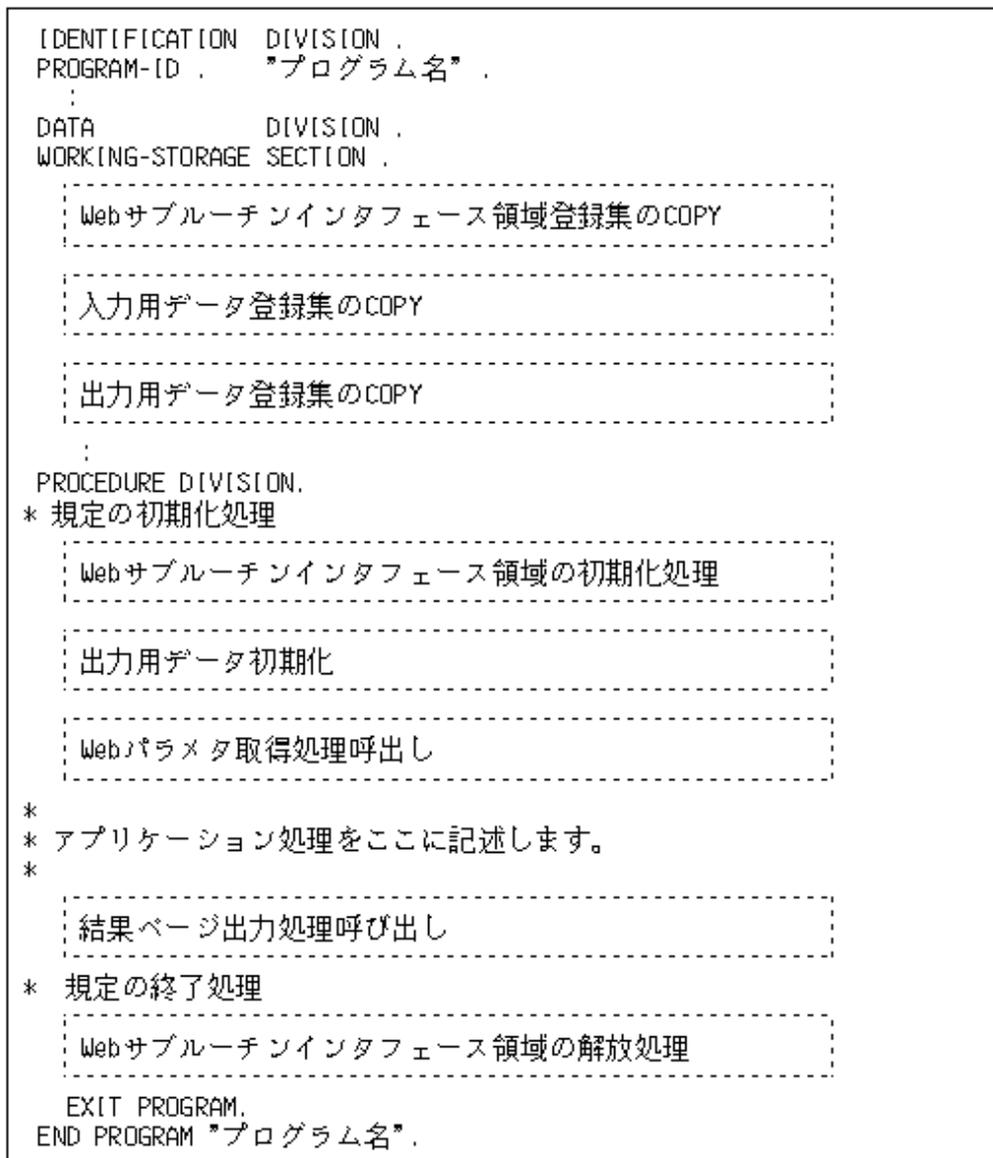
[5] Webアプリケーションは、結果ページ出力用のプログラムを呼び出します。

[6] 結果ページ出力用のプログラムは結果出力用ページ内に含まれる変換名を変換文字列で置き換えた上で、結果出力用ページの出力要求をWebサーバに送ります。

[7] Webブラウザに結果出力用ページが表示されます。

## プログラムの基本構造

Webアプリケーションの雛形は次の構造を持つプログラムです。



## Webアプリケーションの雛形の修正方法

Webアプリケーションの雛形には、アプリケーション固有処理は生成されていません。このため、このアプリケーションの雛形の動作は呼出し用ページからWebパラメタを受け取りますが、何もせず結果出力用ページを表示するだけです。

このWebアプリケーションの雛形を修正して、実際のアプリケーションにするためには、次のコメントの下にアプリケーション固有の処理を追加します。

```

*
* アプリケーション処理をここに記述します。
*

```

たとえば、呼出し用ページから受け取ったデータを結果出力用ページにそのまま表示したい場合は次の記述を追加します(生成された、結果ページ出力プログラムは、結果出力用ページに含まれる変換名を変換文字列で置き換える処理を含んでいます。詳細は、“[26.5.3.1 結果出力用ページ](#)”を参照してください)。

```
*
* アプリケーション処理をここに記述します。
*
  MOVE "TEST-DATA" TO O-NAME_DATA
  MOVE I-INPUT-FIELD TO O-OUT_DATA
*
*

```

結果ページ出力プログラムに渡すデータ項目

Webパラメタ取得プログラムから受け取ったデータ項目

これによりアプリケーションの動作は次のように変更されます。



なお、Webパラメタ取得プログラム、結果ページ出力プログラムおよびそれぞれの必要とする登録集は、Webパラメタ取得プログラム生成、Web結果ページ出力プログラム生成で生成されるものです。詳細は、“[26.4 Webパラメタ取得プログラム生成](#)”、“[26.5 結果ページ出力プログラム生成](#)”を参照してください。

## 26.3.2 Webアプリケーションウィザードの使用法

Webアプリケーションウィザードの使い方には、次の3つがあります。

1. Webアプリケーション開発用のプロジェクトとWebアプリケーションの雛形となるプログラムを生成する。
2. 既存のプロジェクトに新しいターゲットファイルとして、Webアプリケーションを追加する。
3. 既存のプロジェクトにすでに存在するターゲットファイルの構成要素にWebアプリケーションの雛形などを追加する。

ここでは最も一般的な使い方である1.について説明します。その他の使い方については、“[26.3.3 Webアプリケーション雛形の構成](#)”を参照してください。

### 26.3.2.1 Webアプリケーションウィザードの起動

プロジェクトマネージャの[ファイル]メニューから“Webアプリケーションウィザード”を選択します。



注意

プロジェクトマネージャが既存のプロジェクトを開いている場合、このプロジェクトは自動的に閉じられます。

## 26.3.2.2 Webアプリケーションウィザードの画面と操作

Webアプリケーションウィザードは3つの画面と1つのダイアログボックスを持ちます。ここでは、順番に操作を説明します。

### 画面1 プロジェクトの情報入力 (Webアプリケーションウィザード 1/3)

画面1ではプロジェクト関連の情報を設定します。

“プロジェクト名”と“格納フォルダ”を設定したら、[次へ]ボタンを押してください。

#### エディットボックス

##### プロジェクト名

生成するWebアプリケーション開発用のプロジェクトの名前を指定します。

##### 格納フォルダ

プロジェクトファイルおよびWebアプリケーションの雛形を構成するソースプログラム、登録集などを格納するフォルダを指定します。

#### ボタン

##### 参照

格納フォルダを選択するダイアログを開きます。

### 画面2 アプリケーションの情報入力 (Webアプリケーションウィザード 2/3)

画面2では、実行ファイル名、使用するWebサブルーチンの種類など、生成するアプリケーションの雛形についての基本情報を設定します。

必要な情報を設定したら、[次へ]ボタンを押してください。

#### エディットボックス

##### 実行ファイル名

ターゲットとしてプロジェクトに追加する実行ファイル名を指定します。

#### ラジオボタン

##### 使用するAPIの種類を選択してください

生成するWebアプリケーションの雛形で使用するWebサブルーチンのインタフェースを選択します。実行ファイル名に指定した実行形式によって選択できる種類が異なります。

EXE実行形式の場合は、“CGI”を選択することができます。DLL実行形式の場合は“ISAPI”を選択することができます。

#### コンボボックス

##### Webアプリケーションのコード系を選択してください

生成するアプリケーションの実行コード系を指定します。デフォルトはシフトJISです。使用するAPIの種類に“CGI”または“ISAPI”を選んでください。また、Unicodeを選択することができます。

### 画面3 ウィザードによるファイル生成方法の指定 (Webアプリケーションウィザード 3/3)

画面3では、ウィザードが生成するWebアプリケーションの雛形の生成方法を指定します。

必要な情報を指定して、[完了]ボタンを押すとWebアプリケーションの雛形を構成する各種のプログラム資産が生成され、プロジェクトに追加されます。

#### チェックボックス

##### プログラムの雛形を生成する

Webアプリケーションの雛形となるプログラムを生成するかどうかを指定します。デフォルトでチェックされています。この指定を変更すると不完全なプロジェクトができてしまうため、通常は変更しないでください。

##### セッションデータの退避処理を自動生成する

“プログラムの雛形を生成する”が有効で、Webアプリケーションの種類に“ISAPI”を選択している場合にチェック可能になります。セッションデータの退避処理については“[26.3.3 Webアプリケーション雛形の構成](#)”を参照してください。

## セッションデータの復元処理を自動生成する

“プログラムの雛形を生成する”が有効で、Webアプリケーションの種類に“ISAPI”を選択している場合にチェック可能になります。セッションデータの復元処理については“[26.3.3 Webアプリケーション雛形の構成](#)”を参照してください。

## 呼出し用ページからWebパラメタを取得するプログラムを自動生成する

呼出し用ページに含まれる入力フォームから、Webパラメタ取得プログラムおよび入力データ用登録集を生成するかどうかを指定します。“呼出し用ページをプロジェクトに登録する”が有効になっている場合にチェック可能になります。

### 注意

本機能を使用するためにはIEが必要です。

## 結果出力用ページを出力するプログラムを自動生成する

結果出力用ページに含まれる変換名の指定から、結果ページ出力プログラムおよび出力データ用登録集を生成するかどうかを指定します。“結果出力用ページをプロジェクトに登録する”が有効になっている場合にチェック可能になります。

## 呼出し用ページをプロジェクトに登録する

呼出し用ページをプロジェクトに登録するかどうかを指定します。登録するHTML文書に既存ファイルを使用するか、ウィザードの生成する雛形ファイルを使用するかは、ラジオボタンで選択します。

## 結果出力用ページをプロジェクトに登録する

結果出力用ページをプロジェクトに登録するかどうかを指定します。登録するHTML文書に既存ファイルを使用するか、ウィザードの生成する雛形ファイルを使用するかは、ラジオボタンで選択します。

## ISAPIアプリケーション用のモジュール定義ファイルを生成する

ISAPI用のモジュール定義ファイルを生成するかどうかを指定します。Webアプリケーションの種類に“ISAPI”を選択した場合にのみ、チェック可能になります。

## エディットボックス

### ファイル名(呼出し用ページをプロジェクトに登録する)

呼出し用ページに、“既存ファイルに登録する”を選んだ場合、そのファイル名を指定します。

### ファイル名(結果出力用ページをプロジェクトに登録する)

結果出力用ページに、“既存ファイルに登録する”を選んだ場合、そのファイル名を指定します。

## ボタン

### 生成ファイル詳細

[生成ファイル詳細]ダイアログを開きます。

### 参照(呼出し用ページをプロジェクトに登録する)

呼出し用ページに使用するファイル名を選択するダイアログを開きます。

### 参照(結果出力用ページをプロジェクトに登録する)

結果出力用ページに使用するファイル名を選択するダイアログを開きます。

## [生成ファイル詳細]ダイアログ

[生成ファイル詳細]ダイアログでは、ウィザードの生成するプログラム資源のファイル名などのデフォルトの指定を変更します。

## リストボックス

### 生成ファイル

ウィザードの生成するプログラム資源で、ファイル名などのデフォルトの指定を変更できるプログラム資源の一覧を表示します。変更したいものをこれらの中から選択します。

## エディットボックス

### プログラム名

リストボックスで選択したWebアプリケーションの雛形の新しい名前を指定します。APIの種類に“ISAPI”を選択した場合は指定できません。

### ファイル名

リストボックスで選択したプログラム資産のファイル名の新しい名前を指定します。

## コンボボックス

### HTML雛形ファイルのコード系

ウィザードが生成するHTML雛形ファイル(呼出し用ページ、結果出力用ページ、エラー出力ページ)の出力コード系を選択します。プログラム実行時のコード系に“Unicode”を選択した場合にだけ指定できます。

## 26.3.3 Webアプリケーション雛形の構成

### 26.3.3.1 生成するプログラム資源

Webアプリケーションウィザードは複数のプログラム資源を生成しますが、何を生成するかはウィザードの画面上での指定や使用するWebサブルーチンのインタフェースの選択に応じて異なります。

表26.1 Webサブルーチンの種類と生成されるプログラム資源の関係

	CGI	ISAPI
アプリケーション雛形	○	○
Webパラメタ取得プログラム	△	△
結果ページ出力プログラム	△	△
入力データ登録集	△	△
出力データ登録集	△	△
呼出し用ページ雛形	△	△
結果出力用ページ雛形	△	△
エラーページ出力プログラム	○	○
エラー出力ページ	○	○
モジュール定義ファイル	—	△

- : 常に生成
- △ : 選択により生成
- : 生成せず

### 26.3.3.2 Webアプリケーション雛形の違い

#### Webアプリケーション雛形のプログラム名とファイル名

Webアプリケーションウィザードによって生成されるアプリケーション雛形は基本的には、“26.3.1 Webアプリケーション”で説明した構造を持ちますが、使用するWebサブルーチンのインタフェースごとに、生成されるプログラムの名前や数が異なります。

表26.2 各Webサブルーチン使用時に生成される名前

API名	プログラム名	ファイル名	雛形
CGI	EXE名-MAIN	EXE名-MAIN.COB	○
ISAPI	GetExtensionVersion	DLL名-GetExtensionVersion.COB	
	HttpExtensionProc	DLL名-HttpExtensionProc.COB	○
	TerminateExtension	DLL名-TerminateExtension.COB	

インタフェースにISAPIを使用する場合はWebアプリケーションのプログラム名は固定になりますが、他のインタフェースを使用する場合は任意の名前を使用できるため、表中のEXE名あるいはDLL名の部分にウィザードで指定した“実行ファイル名”が使用されます。

なお、表の雛形の欄に○が示してあるプログラムが“26.3.1 Webアプリケーション”で説明した構造を持つプログラムです。



ウィザードで指定した“実行ファイル名”に空白またはCOBOLの利用者語に使用できない文字が含まれている場合はその文字は除いて使用します。

## Webアプリケーションの起動の指定

Webアプリケーションの起動は、呼出し用ページのFORMタグ中のACTION属性で指定します。たとえば、CGIのインタフェースを使用するSAMPLE.EXEというWebアプリケーションの場合は次のようになります。

```
<FORM METHOD="POST" ACTION="SAMPLE.EXE">
```

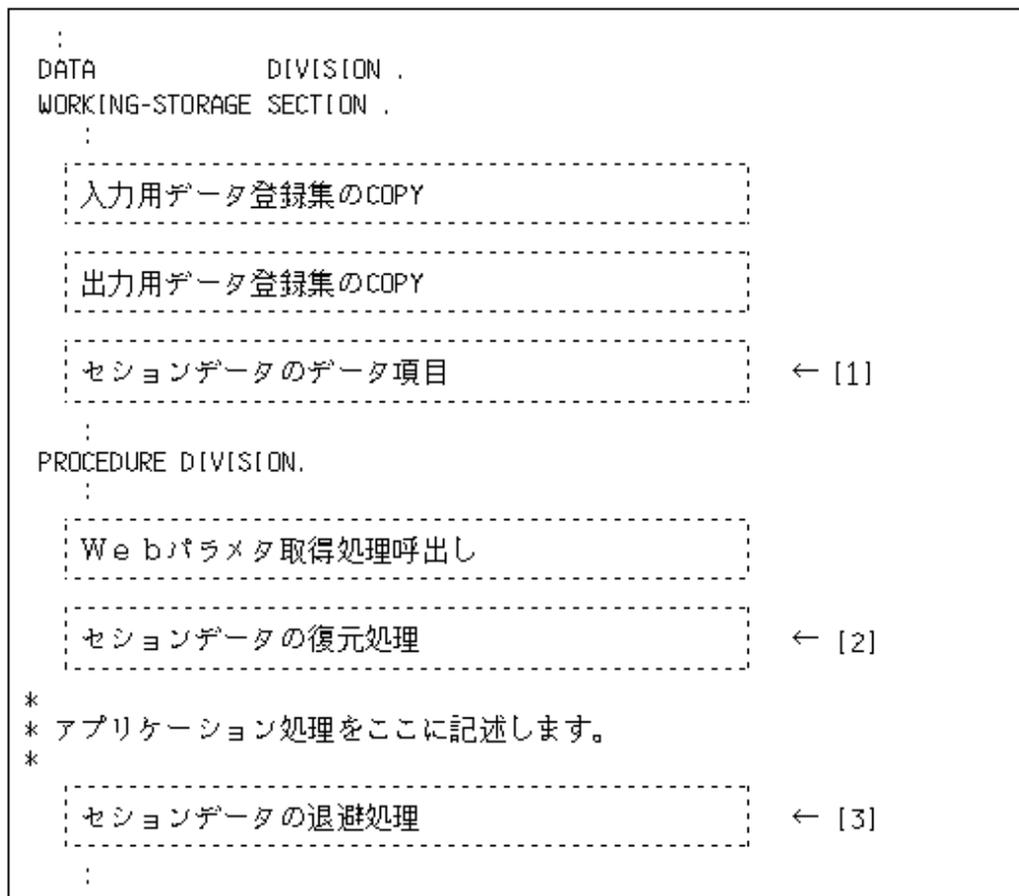
ウィザードの画面で“雛形ファイルを生成する”を選んだ場合は使用するインタフェースに応じた適切な記述をウィザードが生成しますが、“既存ファイルを登録する”を選択した場合は、このACTION属性に指定する値を次の規則に従って修正する必要があります。

API名	ACTION属性への指定
CGI	[URL名/]EXEファイル名
ISAPI	[URL名/]DLLファイル名

## セッションデータ退避/復元処理

Webアプリケーションウィザードの生成するアプリケーションの雛形は“26.3.1 Webアプリケーション”で説明した構造を持ちますが、使用するAPIにISAPIを選択した場合は、セッションデータの退避処理および復元処理を生成させることができます。

ウィザードで“セッションデータの退避処理を自動生成する”あるいは“セッションデータの復元処理を自動生成する”を選択した場合は次のような位置に作業用のデータ項目の雛形と処理を生成します。



なお、[1]の位置に生成されるセッションデータのデータ項目は、“-SESSION-DATA”というサフィックスを持つダミーの集団項目です。実際にセッションデータの復元/退避処理を有効にするには、対象となるデータ項目をこの集団項目に従属するデータ項目として定義する必要があります。また、[2]、[3]の位置に生成されるセッションデータの復元/退避では、セッション範囲を制御するサブルーチン呼出しを行います。セッション範囲を個別に制御する場合には、必要に応じてサブルーチン呼出し処理を修正してください。

### 26.3.3.3 生成するその他の資源

#### Webパラメタ取得プログラムと入力データ登録集

ウィザードの画面から“呼出し用ページからWebパラメタを取得するプログラムを自動生成する”を選択すると生成されます。生成されるプログラムおよびファイルはWebパラメタ取得プログラム生成ツールによって生成されるものと同じです。詳細は、“[26.4.3 Webパラメタ取得プログラムの生成規則](#)”を参照してください。

#### 結果ページ出力プログラムと出力データ登録集

ウィザードの画面から“結果出力ページを出力するプログラムを自動生成する”を選択すると生成されます。生成されるプログラムおよびファイルは結果ページ出力プログラム生成ツールによって生成されるものと同じです。

詳細は、“[26.5.3 結果ページ出力プログラムの生成規則](#)”を参照してください。

#### 注意

ウィザードが生成するWebパラメタ取得プログラムおよび結果ページ出力プログラムは、それぞれ[Webパラメタ取得プログラム生成]、[Web結果ページ出力プログラム生成]をデフォルト指定で実行した場合と同じ形式になります。生成内容の指定が必要な場合は、各機能を再度実行してください。

## エラーページ出力プログラムとエラー出力ページ

生成したアプリケーションの雛形でエラーを検出した場合に呼び出されるプログラムとエラー出力ページです。

以下の規則に従って生成されます。

種類	プログラム名	ファイル名
エラーページ出力プログラム	EXE/DLL名-ERRPAGE	EXE/DLL名-ERRPAGE.COB
エラー出力ページ	—	EXE/DLL名-ERRPAGE.HTML

エラー出力ページは、以下のコード系が使用されます。

- ・アプリケーションの実行時のコード系がシフトJISの場合、シフトJIS
- ・アプリケーションの実行時のコード系がUnicodeの場合、UCS-2とUTF-8のうちからウィザードで選択したもの

エラーページ出力プログラムは、エラー出力ページがアプリケーションと同じフォルダに存在するものとして処理を行います。

## 呼出し用ページ雛形と結果出力用ページ雛形

呼出し用ページ雛形と結果出力用ページ雛形は、ウィザードで以下を選択した場合に生成されます。

- ・“呼出し用ページをプロジェクトに登録する”と“雛形ファイルを生成する”を選んだ場合
- ・“結果出力用ページをプロジェクトに登録する”と“雛形ファイルを生成する”を選んだ場合

種類	ファイル名
呼出し用ページ雛形	EXE/DLL名-INPAGE.HTML
結果出力用ページ雛形	EXE/DLL名-OUTPAGE.HTML

呼出し用ページ雛形および結果出力用ページ雛形は、以下のコード系が使用されます。

- ・アプリケーションの実行時のコード系がシフトJISの場合、シフトJIS
- ・アプリケーションの実行時のコード系がUnicodeの場合、UCS-2とUTF-8のうちからウィザードで選択したもの

## モジュール定義ファイル

ウィザードで“ISAPIアプリケーション用のモジュール定義ファイルを生成する”を選んだ場合に生成します。

ISAPIアプリケーションに必要なモジュール定義ファイルをDLL名に“-MODULE.DEF”を付けた名前前で生成します。

### 26.3.3.4 その他のWebアプリケーションウィザードの使用方法

Webアプリケーションウィザードの使い方にはここまでに説明したWebアプリケーションの雛形を含む新規プロジェクトを生成する以外に2つの使い方があります。

- ・既存のプロジェクトにWebアプリケーションを追加する
- ・既存の実行形式ファイルにWebアプリケーションの情報を追加する

#### 既存のプロジェクトにWebアプリケーションを追加する

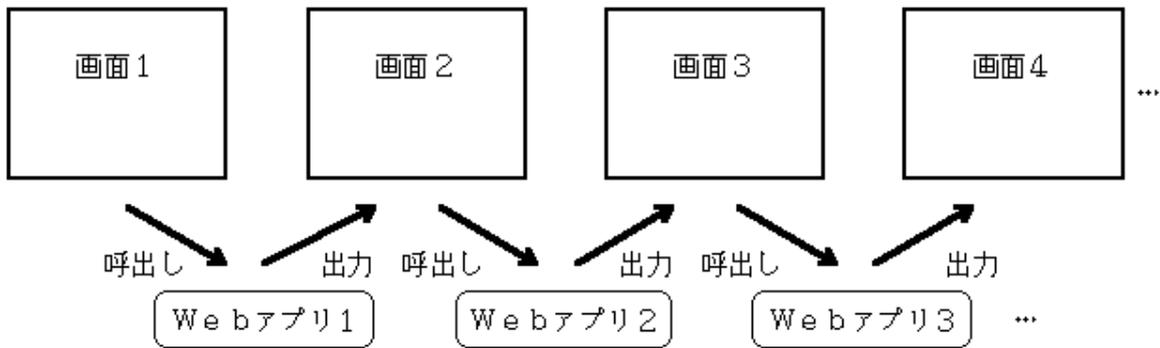
すでに1つ以上の実行形式ファイルを含むプロジェクトに新しい実行形式ファイルとしてWebアプリケーションの雛形を追加します。対象となるプロジェクトは通常の方法で作成したもの、またはWebアプリケーションウィザードによって生成したものです。

次の手順で起動します。

1. 対象となるプロジェクトを開き、プロジェクトマネージャのツリービュー上のプロジェクトファイルを選択します。
2. プロジェクトマネージャの[編集]メニューから、[Webアプリケーション定義] > [アプリケーション定義の追加]を選択します。

以降の操作は、“26.3.2.2 Webアプリケーションウィザードの画面と操作”と同じです。ただし、追加するアプリケーションの実行時のコード系は、プロジェクトにすでに存在する実行形式ファイルと同じものにしなければなりません。

この機能は次に示すような一連の流れを持つ複数の画面と複数のWebアプリケーションの開発を1つのプロジェクトで管理する場合にご使用ください。



### 既存の実行形式ファイルにWebアプリケーションの情報を追加する

プロジェクト中にすでに存在する実行形式ファイルの構成要素にWebアプリケーションの雛形をはじめとする各種ファイルを追加します。なお、対象となる実行形式ファイルはWebアプリケーションウィザードによって生成されたものであってはなりません。

次の手順で起動します。

1. プロジェクトを開き、プロジェクトマネージャのツリービュー上で対象となる実行形式ファイルを選択します。
2. プロジェクトマネージャの[編集]メニューから、[Webアプリケーション定義] > [雛形と情報の登録]を選択します。

以降の操作は、“[26.3.2.2 Webアプリケーションウィザードの画面と操作](#)”と同じです。ただし、対象となる実行形式ファイルの名前と実行時のコード系は変更できません。

この機能は、既存のアプリケーションをそのビジネスロジックの部分のみを残し、Webアプリケーション化する場合にご使用ください。

### Webアプリケーションウィザードを再実行する

Webアプリケーションウィザードによってプロジェクトに追加された、あるいはWebアプリケーション用の情報を追加された実行形式ファイルを対象にWebアプリケーションウィザードを再度実行することはできません(プロジェクト画面で異なるアイコンで表されます)。

このため、Webアプリケーションウィザードを再実行する必要がある場合には、不要な情報を一度削除する必要があります。

これは次の手順で行います。

1. プロジェクトマネージャのツリービュー上で対象となるWebアプリケーションの実行形式ファイルを選択します。
2. プロジェクトマネージャの[編集]メニューから、[Webアプリケーション定義] > [Webアプリケーションの定義解除]を選択します。

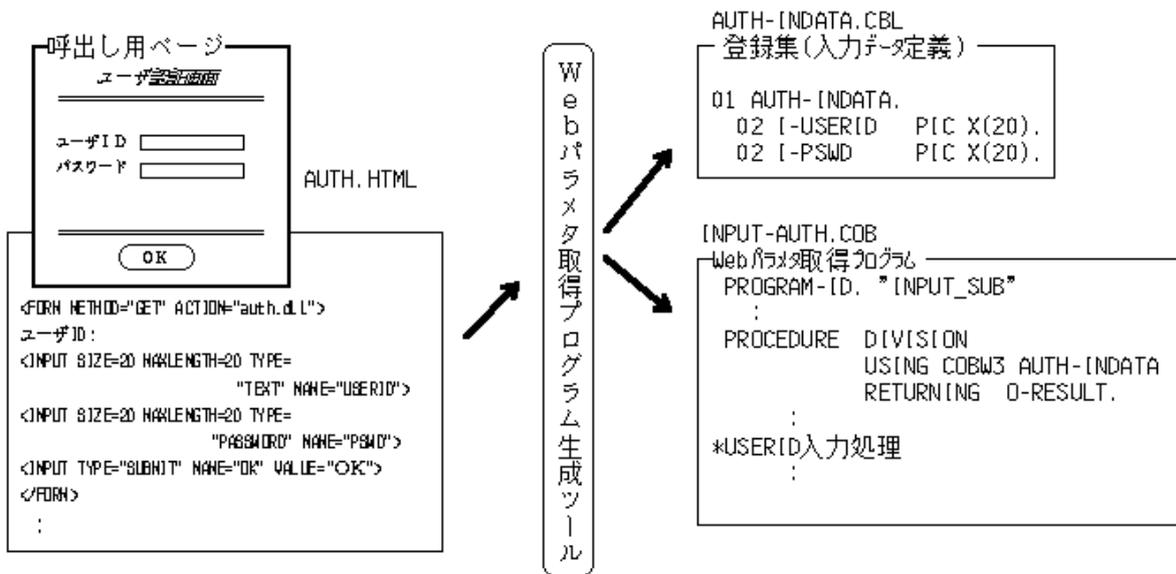


#### 注意

この機能はプロジェクトからWebアプリケーション固有の情報を削除しますが、生成したWebアプリケーションの雛形などのファイルの削除は行いません。不要なら手動で削除してください。

## 26.4 Webパラメタ取得プログラム生成

Webパラメタ取得プログラム生成ツールは、呼出し用ページに含まれる入力フォームからWebパラメタを取得するプログラムおよび入力データ用の登録集を生成します。



このツールは次のような場合に使用します。

- Webアプリケーションウィザードの実行時に“呼び出し用ページからWebパラメータを取得するプログラムを自動生成する”を選択しないで生成したWebアプリケーションにWebパラメータ取得プログラムを追加する。
- 呼び出し用ページを差し替えた場合や修正した場合、新しい呼び出し用ページに合わせてWebパラメータ取得プログラムを再生成する。

### 注意

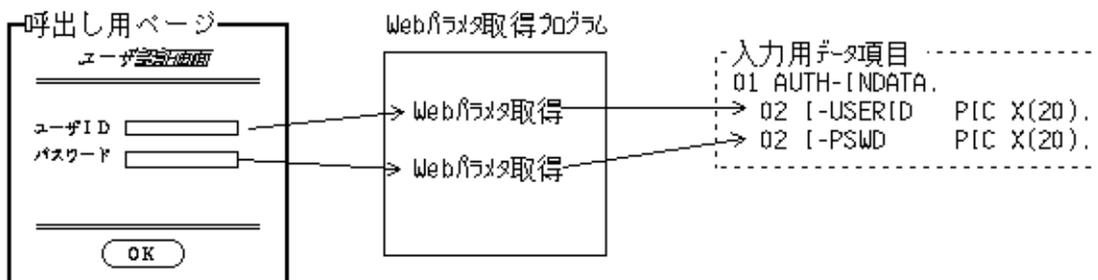
本機能を使用するためにはIEが必要です。

## 26.4.1 Webパラメータ取得プログラム

ここではWebパラメータ取得プログラム生成ツールの生成するプログラムについて説明します。

### プログラムの動作概要

Webパラメータ取得プログラムには、入力フォームの各入力項目に対応するデータ項目が宣言されており、入力フォームの各入力項目から値を取り出して、対応するデータ項目に設定します。



## プログラムの基本構造

Webパラメタ取得プログラムは、次のような構造を持つプログラムです。

```
IDENTIFICATION DIVISION .  
PROGRAM-ID . "プログラム名". ←[1]  
ENVIRONMENT DIVISION .  
DATA DIVISION .  
WORKING-STORAGE SECTION .  
LINKAGE SECTION .  
COPY COBW3.  
COPY 入力データ用登録集名. ←[2]  
01 0-RESULT PIC 9(9) .  
PROCEDURE DIVISION USING COBW3  
                        集団項目名 ←[3]  
                        RETURNING 0-RESULT .  
:  
----- HTMLデータの取得処理 ----- ←[4]  
:  
EXIT PROGRAM.  
END PROGRAM "プログラム名". ←[1]
```

[1] 生成されたプログラム名が、プログラム名段落とプログラム終わり見出しに展開されます。

[2] 生成された入力データ用の登録集ファイル名が展開されます。

[3] 入力フォームの各入力項目に対応するデータ項目は、1つの集団項目に従属する項目として生成されます。ここにはその集団項目名が展開されます。

[4] 入力フォームの各入力項目のデータを取得し、対応するデータ項目に設定する処理が展開されます。



注意

生成されるプログラム名、データ項目名、プログラムおよび登録集のファイル名などの詳細は、“[26.4.3 Webパラメタ取得プログラムの生成規則](#)”を参照してください。

## プログラムのインタフェースと使用方法

Webパラメタ取得プログラムは2つの引数と復帰値を持ち、Webアプリケーションからは次の形式で呼び出します。

```
CALL "Webパラメタ取得プログラム名"  
     USING データ名1 データ名2  
     RETURNING データ名3
```

引数および復帰値のインタフェースは次のとおりです。

- データ名1はWebサーバのAPIを使用するためのインタフェース領域です。登録集“COBW3.CBL”内に定義されているCOBW3を指定します。
- データ名2は入力データを格納する領域です。生成された入力データ用登録集内に定義されている01集団項目名を指定します。
- データ名3はWebパラメタ取得の成功/不成功を示す次の値を返却します。

0 : すべてのWebパラメタの取得に成功した。

0以外 : 何らかのエラーが発生した。

## 注意

複数の入力項目がある場合、ある入力項目に対するWebパラメタの取得に失敗しても、他の項目に対するWebパラメタの取得処理は続行されます。

この呼出しが正常に終了(復帰値の値が0)した場合は、入力フォームの各入力項目の値がデータ名2に従属する各データ項目に格納されています。

Webアプリケーションウィザードで生成したアプリケーション雛形のWebパラメタ取得プログラム部分を変更あるいは追加した場合はウィザードの生成したアプリケーションの雛形に次の修正が必要となります。

- アプリケーションの雛形が取り込む入力データ用の登録集名
- アプリケーションの雛形から呼び出すWebパラメタ取得プログラムの名前
- Webパラメタ取得プログラム呼出しの第2引数

以下にその例を示します。例では、呼出し用ページをSAMPLE.HTMLからAUTH.HTMLに変更して、Webパラメタ取得プログラムを再度生成したものとします。

## 例

- 修正前

```
:  
WORKING-STORAGE SECTION.  
:  
COPY "SAMPLE-INDATA.CBL".  
:  
PROCEDURE DIVISION.  
:  
CALL "INPUT-SAMPLE" USING COBW3 SAMPLE-INDATA  
RETURNING 0-RESULT.
```

- 修正後

```
:  
WORKING-STORAGE SECTION.  
:  
COPY "AUTH-INDATA.CBL".  
:  
PROCEDURE DIVISION.  
:  
CALL "INPUT-AUTH" USING COBW3 AUTH-INDATA  
RETURNING 0-RESULT.
```

## 26.4.2 Webパラメタ取得プログラム生成ツールの使用方法

ここでは、Webパラメタ取得プログラム生成ツールの起動方法と使い方を説明します。

### ツールの起動

まず、ツールの起動方法を説明します。

1. プロジェクトマネージャのツリービューから、Webパラメタ取得プログラム生成の対象とする呼出し用ページを選択します(HTML文書はプロジェクト内の“その他”のフォルダに含まれます)。
2. プロジェクトマネージャの[ツール]メニューから“Webパラメタ取得プログラム生成”を選択します。



## 注意

このプロジェクトには1つ以上のWebアプリケーションウィザードで追加した実行形式ファイルが含まれている必要があります。

## 画面と操作

Webパラメタ取得プログラム生成ツールを起動すると、Webパラメタ取得プログラムの生成に必要な情報を入力するための画面が表示されます。

必要な情報を指定して、[OK]ボタンを押すとツールが実行されます。

### エディットボックス

#### 呼出し用ページファイル名

ツールの起動時に指定した呼出し用ページのファイル名が表示されます。変更はできません。

#### 入力項目の最大文字列長

入力データを格納する領域の最大長を指定します。入力項目の字類が英数字の場合は1～1024の範囲で、日本語の場合は1～512の範囲で指定できます。デフォルトとして100が指定されています。

### リストボックス

#### 生成COBOLプログラム登録先

ツールの生成するWebパラメタ取得プログラムと登録集を構成要素に追加する対象となる実行形式ファイルを選択します。プロジェクト内に、Webアプリケーションウィザードにより追加した実行形式ファイルが複数登録されている場合だけ有効となります。

### ラジオボタン

#### 入力項目の字類

入力データを格納する領域を英数字データ項目として生成するか、日本語データ項目として生成するかを指定します。デフォルトでは英数字データ項目です。

#### 使用するAPIの種類

生成するWebパラメタ取得プログラムで使用するAPIの種類を表示します。変更はできません。

### ボタン

#### OK

指定した内容を有効にして、ツールを実行します。

#### キャンセル

処理を中断します。

#### ヘルプ

ヘルプを表示します。

## 実行結果

ツールの実行の結果、Webパラメタ取得プログラムと入力データ用の登録集が生成され、指定した実行形式ファイルの構成要素に追加されます。

## 26.4.3 Webパラメタ取得プログラムの生成規則

ここでは、Webパラメタ取得プログラムの生成規則の詳細について説明します。

### 26.4.3.1 呼出し用ページ

Webパラメタ取得プログラム生成ツールは、HTML3.2準拠の入力フォームを持つ呼出し用ページを入力として受け付けます。この呼出し用ページのコード系は次のどれかである必要があります。

- ・ シフトJIS

- UTF-8(Unicode)
- UCS-2(Unicode)

なお、入力可能な呼出し用ページのコード系は登録先のWebアプリケーションの実行時のコード系に依存します。WebアプリケーションがシフトJISで動作する場合はシフトJISの、Unicodeで動作する場合はUTF-8またはUCS-2のHTML文書を用意してください。

Webパラメタ取得プログラム生成ツールは、次の形式のFORMタグに含まれるフォームエレメントを調べます。

```
METHOD = "GET" | "POST"
ACTION = "起動プログラムの指定"
ENCTYPE= "エンコード方式"
</FORM>
```

## 注意

フォームエレメントにFILEタイプを使用する場合、FORMタグには必ず以下の属性を指定してください。

```
METHOD = "POST"
ENCTYPE = "multipart/form-data"
```

そして次のどれかの形式のフォームエレメントでNAME属性を持つものすべてを入力項目とみなして、対応する入力データ項目を生成します。

```
<INPUT
  TYPE = TEXT | CHECKBOX | RADIO | HIDDEN | PASSWORD | SUBMIT | FILE
  NAME = 名前
>

<SELECT
  NAME = 名前
  MULTIPLE = 複数指定
> </SELECT>

<TEXTAREA
  NAME = 名前
> </TEXTAREA>
```

## 注意

FILEタイプは、ISAPIおよびCGIがサポートしています。

なお、呼出し用ページに複数のFORMタグが存在する場合は、それらに含まれるすべてのフォームエレメントが対象となります。

## 注意

呼出し用ページには、上に挙げた形式以外のFORMタグやフォームエレメントを含めることができます。Webパラメタ取得プログラム生成ツールはそれらの指定を無視します。

### 26.4.3.2 データ入力用登録集

#### 命名規則

入力とした呼出し用ページの名前(ファイル名から拡張子を除いた部分)とフォームエレメントのNAME属性をもとに次の規則に従って、入力データ項目や登録集の名前が決まります。

## 登録集ファイル名

呼出し用ページの名前に“-INDATA.CBL”を付加した名前になります。長さが50文字を超える場合は50文字に収まるように呼出し用ページの名前の最後を切り詰めます。

## 集団項目名

呼出し用ページの名前に“-INDATA”を付加した名前になります。呼出し用ページの名前が全角文字で構成されている場合、この文字列も全角文字になります。

## 入力データ項目名

フォームエレメントのNAME属性で指定された入力項目名の先頭に“-I-”を付けた名前になります。入力項目名が全角文字で構成されている場合はこの文字列も全角文字になります。なお、カウンタ項目の場合は“-C-”が用いられます。当機能で処理可能なNAME属性文字列の上限は50文字です。

また、FILEタイプの場合、入力データ項目の従属項目として、クライアントパス名、クライアントファイル名、コンテンツタイプの3つが固定の名前で生成されます。



## 注意

集団項目名、入力データ項目名はCOBOLの利用者語の規則に従うために、次の場合は、もともになる呼出し用ページの名前やフォームエレメントのNAME属性を変更した上で名前名の生成が行われます。

- ・ 合計が30文字を超える場合は、もともになる名前を切り詰めます。
- ・ もともになる名前に半角文字、全角文字が混在している場合は、最初の文字と同じ種別の文字のみを有効とみなします。たとえば、“INPUT属性1”は“INPUT1”とみなされます。
- ・ もとの名前に含まれる空白またはCOBOLの利用者語に使用できない文字は無視します。
- ・ フォームエレメントのNAME属性に半角カナは使用できません。

## 入力データ項目の生成規則

生成される入力データ項目の属性は次のことから決定されます。

- ・ 入力項目がどんなフォームエレメントによるものか
- ・ フォームエレメントに含まれる属性に指定された値
- ・ 同種のフォームエレメントに同名の項目があるか
- ・ Webパラメタ取得プログラム生成ツールで指定したデータ項目の字類、最大長

以下に、各フォームエレメントに対して生成されるデータ項目の一般形式の一覧を示します。

タグ種別	同じ名前がない場合	同じ名前のある場合
INPUT TYPE指定なし	02 I-名前 PIC {X N} (最大長).	02 C-名前 PIC 9(4). 02 I-名前 OCCURS 重複数 PIC {X N} (最大長).
INPUT TYPE=TEXT		
INPUT TYPE=PUSSWORD		
INPUT TYPE=CHECKBOX	02 C-名前 PIC 9(4). 02 I-名前 PIC {X N} (最大長).	
INPUT TYPE=FILE	02 I-データ名. 03 F-CLIENT_PATH PIC X(128) 03 F-CLIENT_FILENAME PIC X(128) 03 F-CONTENT_TYPE PIC X(64). 03 F-FILE_SIZE PIC S9(18) COMP-5.	
INPUT TYPE=RADIO	02 I-名前 PIC {X N} (最大長).	
INPUT TYPE=SUBMIT		

タグ種別	同じ名前の項目がない場合	同じ名前の項目がある場合
INPUT TYPE=HIDDEN	02 I-名前 PIC {X N} (最大長).	02 C-名前 PIC 9(4). 02 I-名前 OCCURS 重複数 PIC {X N} (最大長).
TEXTAREA		
SELECT MULTIPLE指定なし		
SELECT MULTIPLE指定あり	02 C-名前 PIC 9(4). 02 I-名前 OCCURS OPTIONタグ数 PIC {X N} (最大長).	

### 26.4.3.3 生成するプログラム

#### 命名規則

入力とした呼出し用ページの名前(ファイル名から拡張子を除いた部分)をもとに、次の規則に従って名前が決められます。

##### ファイル名

呼出し用ページの名前の先頭に“INPUT-”を、拡張子として“.COB”を付加した名前になります。ファイル名の長さが50文字を超える場合は50文字に収まるようにHTML文書名の最後を切り詰めます。

##### プログラム名

呼出し用ページの名前の先頭に“INPUT-”を付加した名前になります。



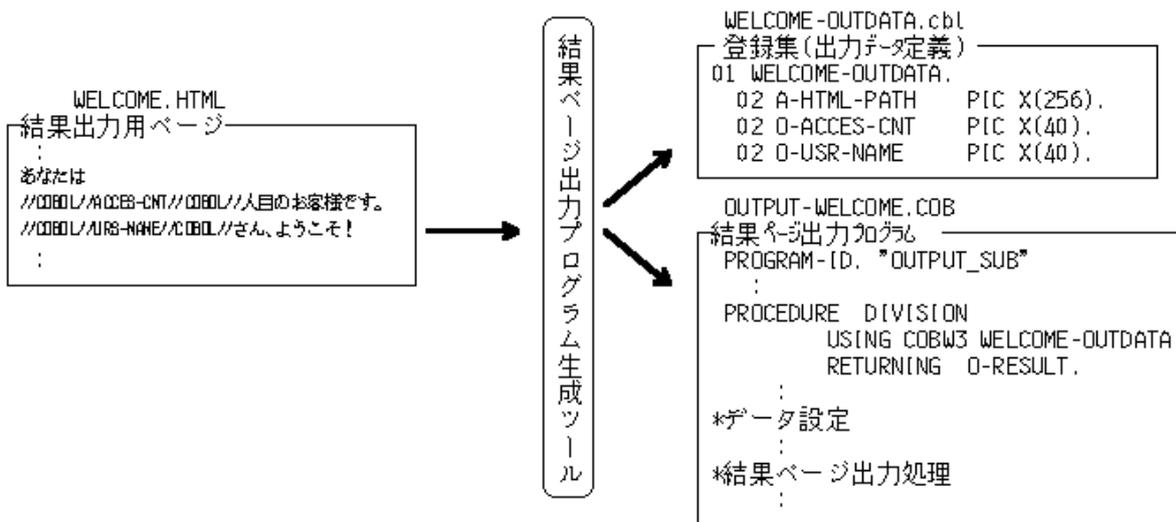
#### 注意

プログラム名はCOBOLの利用者語の規則に従うために、次の場合は、元になる呼出し用ページの名前を変更した上で名前の生成が行われます。

- ・ 合計が30文字を超える場合は、もともになる名前を切り詰めます。
- ・ もともになる名前に半角文字、全角文字が混ざっている場合は、最初の文字と同じ種別の文字だけが有効になります。たとえば、“INPUT 属性1”は“INPUT1”とみなします。
- ・ もとの名前に含まれる空白またはCOBOLの利用者語に使用できない文字は無視します。

## 26.5 結果ページ出力プログラム生成

結果ページ出力プログラム生成ツールは、結果出力用ページに含まれる変換名の指定を変換文字列で置き換え、結果出力用ページを出力するプログラムを生成します。



このツールは次のような場合に使用します。

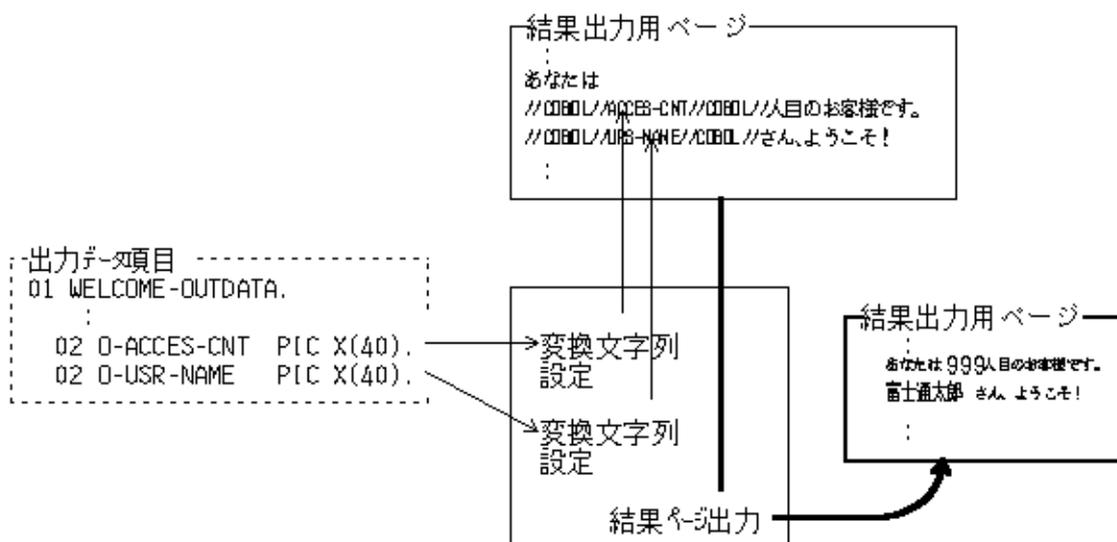
- Webアプリケーションウィザードの実行時に“結果出力用ページを出力するプログラムを自動生成する”を選択しないで生成したWebアプリケーションに結果ページ出力プログラムを追加する。
- 結果出力用ページを差し替えた場合や修正した場合、新しい結果出力用ページに合わせて結果ページ出力プログラムを再生成する。

## 26.5.1 結果ページ出力プログラム

ここでは結果ページ出力プログラム生成ツールによって生成されるプログラムについて説明します。

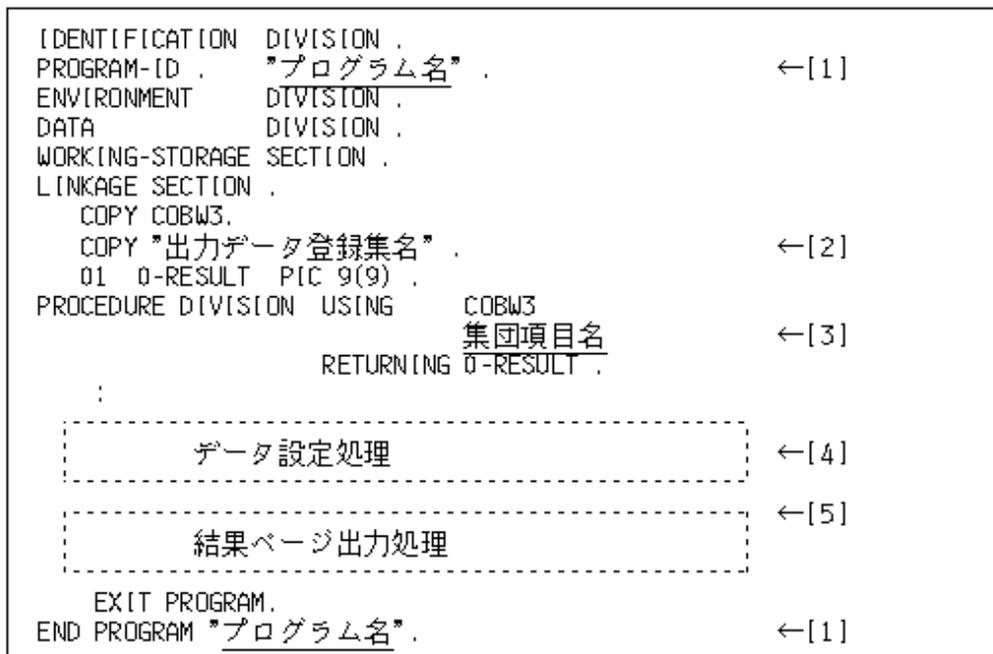
### プログラムの動作概要

結果ページ出力プログラムには、結果出力用ページ内に含まれる各変換名に対応するデータ項目が宣言されており、このデータ項目に設定されている変換文字列で変換名を置き換えて、出力します。



## プログラムの基本構造

結果ページ出力プログラムは次のような構造を持つプログラムです。



- [1] 生成されたプログラム名が、プログラム名段落とプログラム終わり見出しに展開されます。
- [2] 生成された出力データ用の登録集ファイル名が展開されます。
- [3] 各変換名の指定に対応するデータ項目は、1つの集団項目に従属する項目として生成されます。ここにはその集団項目名が展開されます。
- [4] 変換名の指定を各出力項目の変換文字列で置き換える処理が展開されます。
- [5] 結果出力用ページを出力(Webサーバに送信)する処理が展開されます。

### 注意

生成されるプログラム名、データ項目名、プログラムおよび登録集のファイル名などの詳細は、“[26.5.3 結果ページ出力プログラムの生成規則](#)”を参照してください。

## プログラムのインタフェースと使用方法

結果ページ出力プログラムは2つの引数と復帰値を持ち、Webアプリケーションからは次の形式で呼び出します。

```
[MOVE "結果出力用ページパス" TO A-HTML-PATH.]
CALL "結果ページ出力プログラム"
    SING データ名1 データ名2
    RETURNING データ名3
```

引数および復帰値のインタフェースは次のとおりです。

- データ名1はWebサーバのAPIを使用するためのインタフェース領域です。登録集“COBW3.CBL”内に定義されているCOBW3を指定します。

- データ名2は出力データを格納する領域です。生成された出力データ用登録集内に定義されている01集団項目名を指定します。最初の従属項目として、結果出力用ページの格納フォルダのパスを指定するデータ項目(A-HTML-PATH)を含みます。アプリケーションと異なるフォルダに結果出力用ページがある場合、このデータ項目に結果出力用ページの格納フォルダのパスを設定しておく必要があります。
- データ名3は処理の成功/不成功を示す次の値を返却します。
  - 0 : すべての処理が成功した。
  - 0以外 : 何らかのエラーが発生した。

Webアプリケーションウィザードで生成したアプリケーション雛形の結果ページ出力プログラム部分を変更あるいは追加した場合はウィザードの生成したアプリケーションの雛形に次の修正が必要となります。

- アプリケーションの雛形が取り込む出力データ用の登録集名
- アプリケーションの雛形から呼び出す結果ページ出力プログラムの名前
- 結果ページ出力プログラム呼出しの第2引数

以下にその例を示します。例では、結果出力用ページをSAMPLE.HTMLからWELCOME.HTMLに変更して、結果ページ出力プログラムを再度生成したものとします。



## 例

### 修正前

```

:
WORKING-STORAGE SECTION.
:
COPY "SAMPLE-OUTDATA. CBL".
:
PROCEDURE DIVISION.
:
CALL "OUTPUT-SAMPLE" USING COBW3 SAMPLE-OUTDATA
RETURNING 0-RESULT.

```

### 修正後

```

:
WORKING-STORAGE SECTION.
:
COPY "WELCOME-OUTDATA. CBL".
:
PROCEDURE DIVISION.
:
CALL "OUTPUT-WELCOME" USING COBW3 WELCOME-OUTDATA
RETURNING 0-RESULT.

```

## 26.5.2 結果ページ出力プログラム生成ツールの使用方法

ここでは、結果ページ出力プログラム生成ツールの起動方法と使い方を説明します。

### ツールの起動

まず、ツールの起動方法を説明します。

- プロジェクトマネージャのツリービューから、結果ページ出力プログラム生成ツールの入力とする結果出力用ページを選択します (HTML文書はプロジェクト内の“その他”のフォルダに含まれます)。
- プロジェクトマネージャの[ツール]メニューから“Web結果ページ出力プログラム生成”を選択します。



## 注意

このプロジェクトには少なくとも1つ以上のWebアプリケーションウィザードで追加した実行形式ファイルが含まれている必要があります。

## 画面と操作

結果ページ出力プログラム生成ツールを起動すると、結果ページ出力プログラムの生成に必要な情報を入力するための画面が表示されます。

必要な情報を指定して、[OK]ボタンを押すとツールが実行されます。

### エディットボックス

#### 結果出力用ページファイル名

ツールの起動時に指定した結果出力用ページのファイル名が表示されます。変更はできません。

#### 変換文字列の最大文字列長

変換文字列を格納する領域の最大長を指定します。変換文字列の字類が英数字の場合は1～1024の範囲で、日本語の場合は1～512の範囲で指定できます。デフォルトとして100が指定されています。

#### 繰返し変換データの最大設定数

繰返し変換データを対象とする場合は生成する変換データの繰返し数の最大値を指定します。1～1000までの範囲で指定することができます。デフォルトとして30が指定されています。

### チェックボックス

#### 繰返し変換データを処理対象とする

繰返し変換データのデータ定義および処理を生成するかどうかを指定します。APIの種類がISAPIまたはCGIの場合に指定できます。デフォルトの設定では繰返し変換データを処理しません。

### リストボックス

#### 生成COBOLプログラム登録先

ツールの生成した結果ページ出力プログラムと登録集を構成要素に追加する対象となる実行形式ファイルを選択します。プロジェクト内に、Webアプリケーションウィザードにより追加した実行形式ファイルが複数登録されている場合にのみ有効となります。

### ラジオボタン

#### 変換文字列の字類

変換文字列を格納する領域を英数字データ項目として生成するか、日本語データ項目として生成するかを指定します。デフォルトでは英数字データ項目です。

#### 作成するAPIの種類

生成する結果ページ出力プログラムで使用するAPIの種類を表示します。変更はできません。

### ボタン

#### OK

指定した内容を有効にしてツールを実行します。

#### キャンセル

処理を中断します。

#### ヘルプ

ヘルプを表示します。

## 実行結果

ツールの実行の結果、結果ページ出力プログラムと出力データ用の登録集が生成され、指定した実行形式ファイルの構成要素に追加されます。

## 26.5.3 結果ページ出力プログラムの生成規則

ここでは、結果ページ出力プログラムの生成規則の詳細について説明します。

### 26.5.3.1 結果出力用ページ

結果ページ出力プログラム生成ツールの入力とする結果出力用ページのコード系は次のどれかである必要があります。

- シフトJIS
- UTF-8(Unicode)
- UCS-2(Unicode)

なお、入力可能な結果出力用ページのコード系は登録先のWebアプリケーションの実行時のコード系に依存します。WebアプリケーションがシフトJISで動作する場合はシフトJISの、Unicodeで動作する場合はUTF-8またはUCS-2の結果出力用ページを用意してください。

結果ページ出力プログラム生成ツールの入力とする結果出力用ページには、ページ出力時に対応するデータ項目の値に置き換えられて表示される変換名の指定を記述することができます。変換名の指定には次の2つの書き方があります。

#### 書き方1 単純指定

```
//COBOL//変換名//COBOL// [...]
```

#### 書き方2 繰返し指定

```
//COBOL_REPEAT_START//  
:  
//COBOL_REPEAT//変換名//COBOL_REPEAT// [...]  
:  
//COBOL_REPEAT_END//
```

なお、書き方2の繰返し指定を持つ変換名の指定は、デフォルトでは処理の対象となりません。これは結果ページ出力プログラム生成ツールで“繰返し変換データを処理対象とする”を選択した場合にのみ処理されます。



#### 注意

結果出力用ページのファイル名は、結果ページ出力プログラム中に文字列の形式で展開されます。このとき、ファイル名がCOBOLの文字定数の上限を超える可能性があるため、結果出力用ページのファイル名は50文字以内にしてください。

### 26.5.3.2 データ出力用登録集

#### 命名規則

入力とした結果出力用ページの名前(ファイル名から拡張子を除いた部分)と変換名をもとに、次の規則に従って出力データ項目や登録集の名前が決められます。

##### 登録集ファイル名

結果出力用ページの名前に“-OUTDATA.CBL”を付加した名前になります。長さが50文字を超える場合は50文字に収まるように結果出力用ページの名前の最後を切り詰めます。

##### 集団項目名

結果出力用ページの名前に“-OUTDATA”を付加した名前になります。結果出力用ページの名前が全角文字で構成されている場合、この文字列も全角文字になります。

##### 出力データ項目名

変換名に“O-”を付けた名前になります。変換名が全角文字で構成されている場合、この文字列も全角文字になります。なお、繰返し指定を持つ変換名を対象とした場合は、繰返し対象の変換データを従属する集団項目“A-REPEAT\_DATA\_xx”と繰返し数を指定するデータ項目“A-REPEAT\_COUNT\_xx”(xxの部分は2けたの通番になります)が生成されます。

## 注意

集団項目名、出力データ項目名はCOBOLの利用者語の規則に従うために、次の場合は、もともになる結果出力用ページの名前や変換名を変更した上で名前の生成が行われます。

- 合計が30文字を超える場合は、もともになる名前を切り詰めます。
- もともになる名前に半角文字、全角文字が混ざっている場合は、最初の文字と同じ種別の文字だけが有効になります。たとえば、“OUTPUT名前1”は“OUTPUT1”とみなします。
- もとの名前に含まれる空白またはCOBOLの利用者語に使用できない文字は無視します。

## 出力データ項目の生成規則

生成される出力データ項目の属性は次のことから決定されます。

- 変換名の形式
- 結果ページ出力プログラム生成ツールで指定した項目の字類、最大長および最大繰返し数

表26.3 各変換名の形式に対して生成されるデータ項目の一般形式の一覧

	変換名の指定	生成データ項目の形式
単純指定	//COBOL//OUTPUT_DATA1//COBOL// //COBOL//OUTPUT_DATA2//COBOL// //COBOL//OUTPUT_DATA3//COBOL// :	02 O-OUTPUT_DATA1 PIC {X N} (指定長). 02 O-OUTPUT_DATA2 PIC {X N} (指定長). 02 O-OUTPUT_DATA3 PIC {X N} (指定長). :
繰返し指定	//COBOL_REPEAT//OUTPUT_DATA1//COBOL_REPEAT// //COBOL_REPEAT//OUTPUT_DATA2//COBOL_REPEAT// //COBOL_REPEAT//OUTPUT_DATA3//COBOL_REPEAT// :	02 A-REPEAT_COUNT_01 PIC 9(4). 02 A-REPEAT_DATA_01 OCCURS 最大繰返し数. 03 O-OUTPUT_DATA1 PIC {X N} (指定長). 03 O-OUTPUT_DATA2 PIC {X N} (指定長). 03 O-OUTPUT_DATA3 PIC {X N} (指定長). :

## 26.5.3.3 生成する出力プログラム

### 命名規則

入力とした結果出力用ページの名前(ファイル名から拡張子を除いた部分)をもとに、次の規則に従って名前が決められます。

#### ファイル名

結果出力用ページの名前の先頭に“OUTPUT-”を、拡張子として“COB”を付加した名前になります。ファイル名の長さが50文字を超える場合は50文字に収まるように結果出力用ページの名前の最後を切り詰めます。

#### プログラム名

結果出力用ページの名前の先頭に“OUTPUT-”を付加した名前になります。

## 注意

プログラム名はCOBOLの利用者語の規則に従うため、次の場合は、もともになる結果出力用ページの名前を変更した上で名前の生成が行われます。

- 合計が30文字を超える場合は、もともになる名前を切り詰めます。
- もともになる名前に半角文字、全角文字が混ざっている場合は、最初の文字と同じ種別の文字だけが有効になります。たとえば、“OUTPUT名前1”は“OUTPUT1”とみなします。
- もとの名前に含まれる空白またはCOBOLの利用者語に使用できない文字は無視します。

## 26.6 Webアプリケーション開発のヒント

---

### 26.6.1 プロジェクトマネージャからのHTML文書の操作

---

プロジェクトマネージャではプロジェクトファイルの配下に“その他”というフォルダを1つだけ作成することができます。

呼出し用ページや結果出力用ページなどのHTML文書をこのフォルダに登録すると、プログラム資産の一部としてプロジェクトマネージャ上から表示や編集ができます。

#### HTML文書の登録

HTML文書は、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、HTML文書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プロジェクトファイルの配下に“その他”フォルダが存在しない場合、フォルダを作成します。
2. “その他”フォルダを選択します。
3. [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。
4. HTML文書を選択します。  
→ フォルダの配下に選択したHTML文書が追加されます。

#### HTML文書の表示と編集

HTML文書は、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表示または編集することができます。

以下に、操作方法を説明します。

1. 表示または編集するHTML文書を選択します。
2. 表示する場合には、[プロジェクト]メニューから“表示”を選びます。  
→ Webブラウザが起動して、選択したHTML文書が表示されます。
3. 編集する場合には、[プロジェクト]メニューから“編集”を選びます。または、選択したHTML文書をダブルクリックします。  
→ HTMLエディタまたはテキストエディタが起動して、選択したHTML文書が編集可能になります。



#### 注意

HTML文書の編集時に起動されるエディタは、デフォルトではシステムでHTML文書に関連付けられているものになります。HTML文書に関連付けられているエディタが存在しない場合は通常のエディタが使用されます。またCOBOLエディタの設定を変更することによって、システムの関連付けとは別にHTMLエディタの関連付けを行うこともできます。

### 26.6.2 エディタテンプレートの活用

---

Webアプリケーション開発支援機能で提供されるウィザードやツールは、データの入出力の処理を生成します。しかし、Webアプリケーションで次のような処理をしたい場合、その処理を自分でWebアプリケーションの雛形に追加する必要があります。

- Cookieデータの処理をする
- Webサーバから固有の情報を取得する
- 認証情報を取得する

このような場合、エディタのテンプレート展開機能をお使いください。テンプレート展開機能は、目的別に分類された処理の雛形をソースの指定した位置に挿入する機能です。

テンプレート展開機能の使い方を以下に示します。

1. エディタでCOBOLソースを開き、処理を挿入したい行の直前にカーソルを位置づけます。
2. エディタの[編集]メニューから“テンプレート展開”を選びます。

3. “テンプレートの分類”のツリービューから挿入する処理のカテゴリを選び、“テンプレート名”のリストボックスから挿入する処理のテンプレート名を選びます。選択したテンプレートにより実際に挿入される処理が“プレビュー”に表示されています。
4. [展開]ボタンを押すとテンプレートで用意されている処理が展開されて、ソースに挿入されます。

#### 注意

この機能はプロジェクトマネージャの持つエディタを使用する場合だけ利用できます。

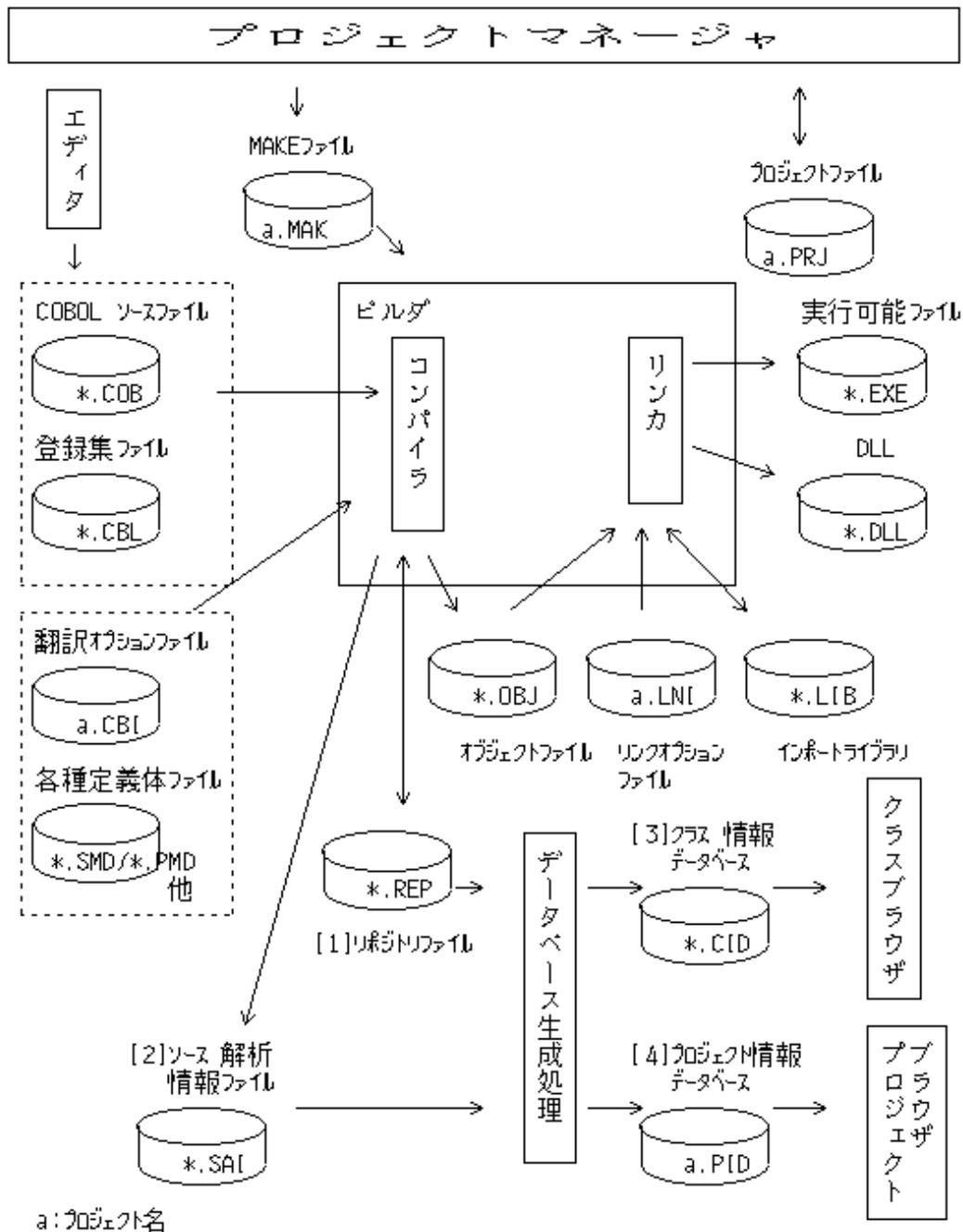
# 第27章 オブジェクト指向プログラミングの開発

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。

## 27.1 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、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た、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。



## 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。

## 27.2 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]“[17.1 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]“[17.2.3 使用するクラスの選定](#)”
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23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、ターゲットリポジティブファイル、依存リポジティブファイルに登録します。[参照]“[17.2.3.2.1 リポジティブファイル](#)”
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)に登録します。
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に登録します。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。[参照]“[17.2.3.3.1 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]“[23.4.5 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]“[23.4.7 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]“[27.4 クラスの公開](#)”

## 27.3 プロジェクト管理

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して、開発手順の説明を行います。

### 27.3.1 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“17.2.3.2.1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“17.2.3.3.1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。

## 27.3.2 プログラム構造

“17.2.3.1 プログラム構造”を参照してください。

- 翻訳単位とリンク単位
- プログラム構造の概要

## 27.3.3 翻訳処理

“17.2.3.2 翻訳処理”を参照してください。

なお、以下はプロジェクトマネージャを使用する場合の説明です。

### ターゲットリポジトリファイル生成フォルダ

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フォルダは、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.3.45 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	フォルダ名
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17.9 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.3.46 REPIN \(リポジトリファイルの入力先フォルダの指定\)](#)”、“[A.3.45 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 (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 (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 (2) プロジェクトファイルのあるパス
	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]メニューから[新規作成]を選択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]メニューから[追加]を選択して、コマンド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に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	○	○
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.3.45 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。

- ー 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。
- ー 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.3.46 REPIN \(リポジトリファイルの入力先フォルダの指定\)](#)”、“[A.3.45 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	ー	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

## 27.3.4 リンク処理

“[17.2.3.3 リンク処理](#)”を参照してください。

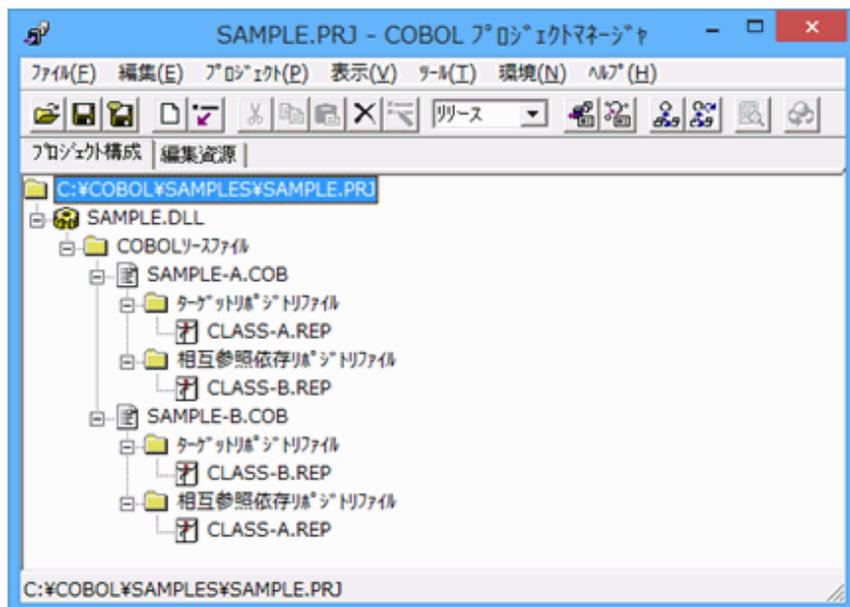
## 27.3.5 相互参照クラス

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、“[17.1.14 相互参照クラス](#)”を参照してください。

### 27.3.5.1 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23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、“[27.3.3 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、“[27.3.5.2 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、“[27.3.5.3 相互参照クラスのリンク処理](#)”を参照してください。

### 27.3.5.2 相互参照クラスの翻訳処理

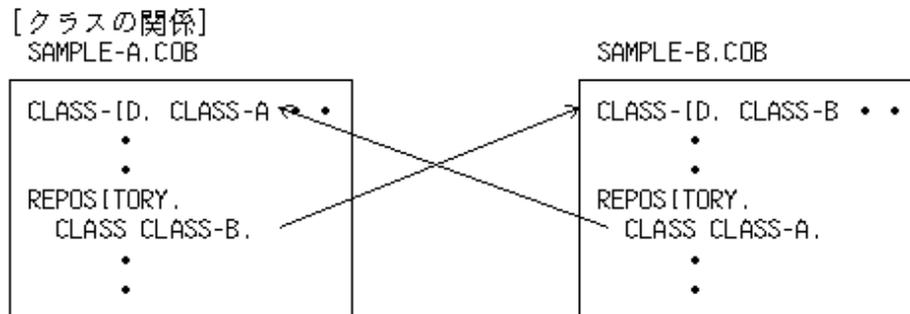
自クラス参照を除く相互参照クラスの翻訳処理では、“[17.1.14 相互参照クラス](#)”で説明したように、リポジトリファイルの依存関係に起因したデッドロック状態が発生します。

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、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。

## 27.3.5.3 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を指定してリンカを使用することにより、SAMPLE-B.DLLを作成します。
4. SAMPLE-B.LIBを指定してリンカ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、“[17.1.14.3 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。

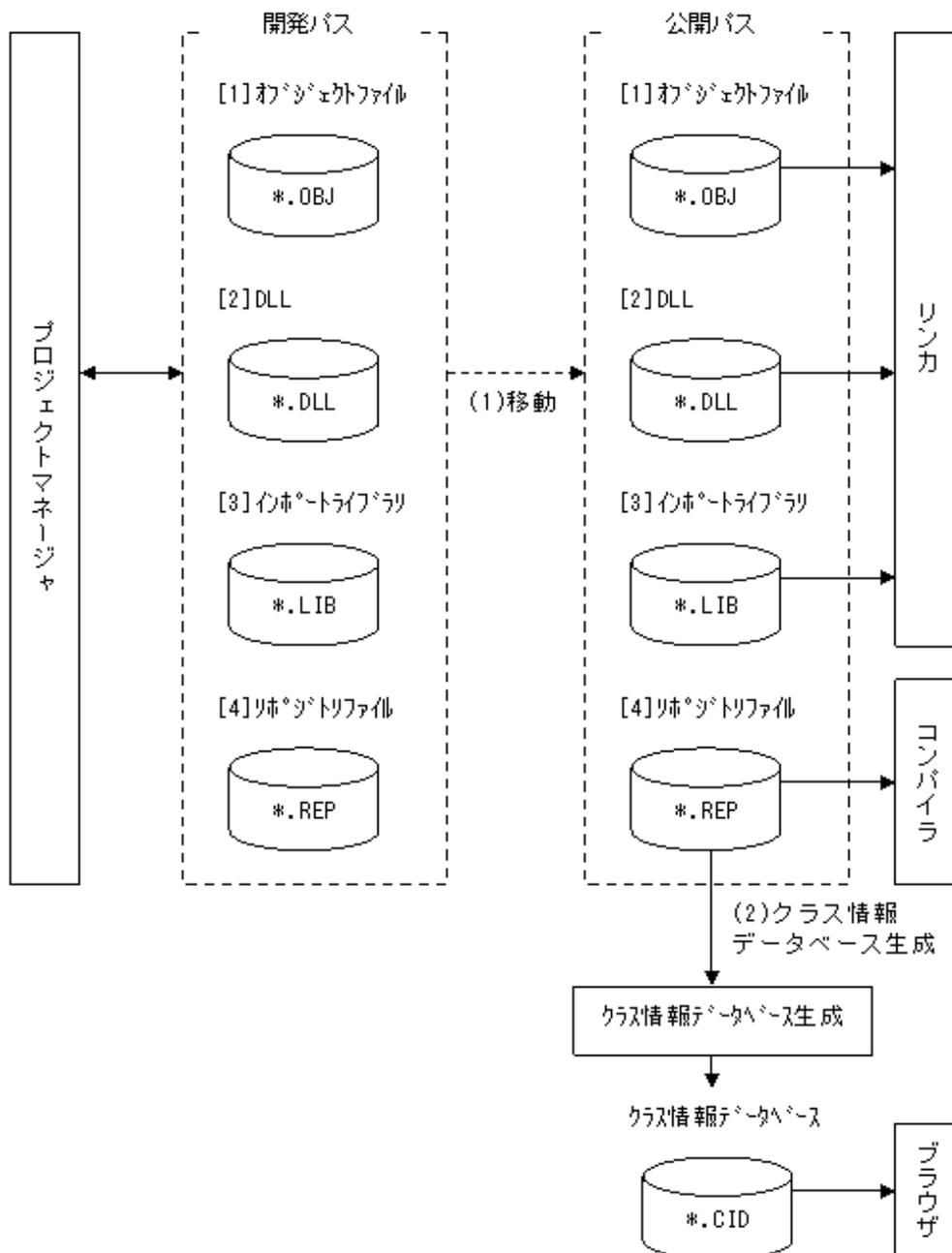
ここで説明しているのは前者です。

後者については、“[23.4.4.6 その他必要なファイルの登録](#)”を参照してください。

## 27.4 クラスの公開

---

ここでは、プロジェクトマネージャを使用して、クラスを公開する方法を説明します。



### (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27.1 公開資源”のようになります。

表27.1 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。

	資源名	用途
[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。 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、“[27.5 クラスブラウザ](#)”を参照してください。

## 27.5 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。

### 27.5.1 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。

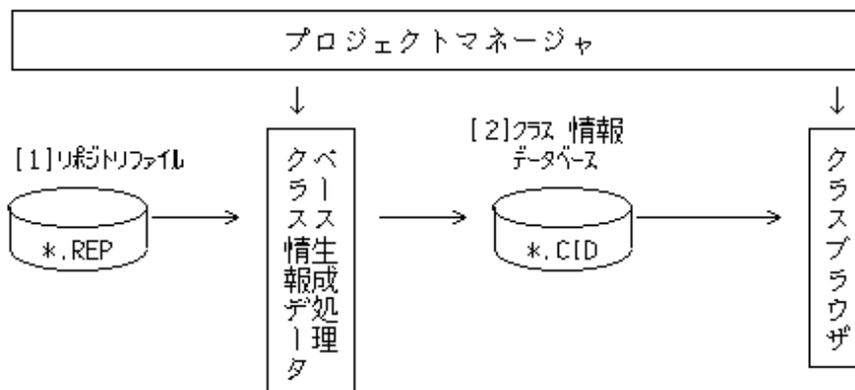
クラスブラウザは、これらの情報を利用者に提供するツールです。

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

### 27.5.2 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] リポジトリファイル

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する資源ですが、それ以外にクラスの情報を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、翻訳処理時にリポジ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ジトリの説明については、“17.2.3.2 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、リポジ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。

## 27.5.3 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. リポジトリファイル公開パス<sup>(a)</sup>に、リポジトリファイルが複写されていることを確認します。
2. リポジ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]“27.5.4 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]“27.5.5 データベース生成”



### 注意

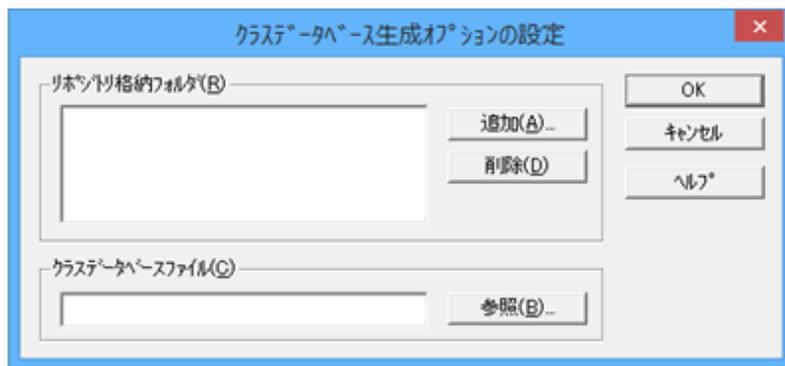
リポジトリファイル公開パスについては、“27.4 クラスの公開”を参照してください。

## 27.5.4 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。

- リポジトリファイル収集パス
- 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。



### 注意

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。

### リポジトリファイル収集パスの設定

リポジトリ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トリ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トリファイルが解析されます。

なお、リポジトリ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、リポジトリファイル収集パスの設定方法について説明します。

1. リポジトリ格納フォルダの[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。

## 27.5.5 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]メニューから[クラスデータベース]>[生成]を選択します。

## 27.5.6 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]“[27.5.7 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]“[27.5.7 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]“[27.5.7 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。

## 27.5.7 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。

### 27.5.7.1 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27.2 クラスブラウザの情報表示](#)”に示します。

表27.2 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

## 27.5.7.2 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。

## 27.5.7.3 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。

## 27.5.7.4 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。

## 27.5.7.5 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。

## 27.5.7.6 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。

## 27.5.7.7 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。

## 27.5.7.8 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、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。

## 27.5.8 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。

## 27.6 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。

### 27.6.1 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。

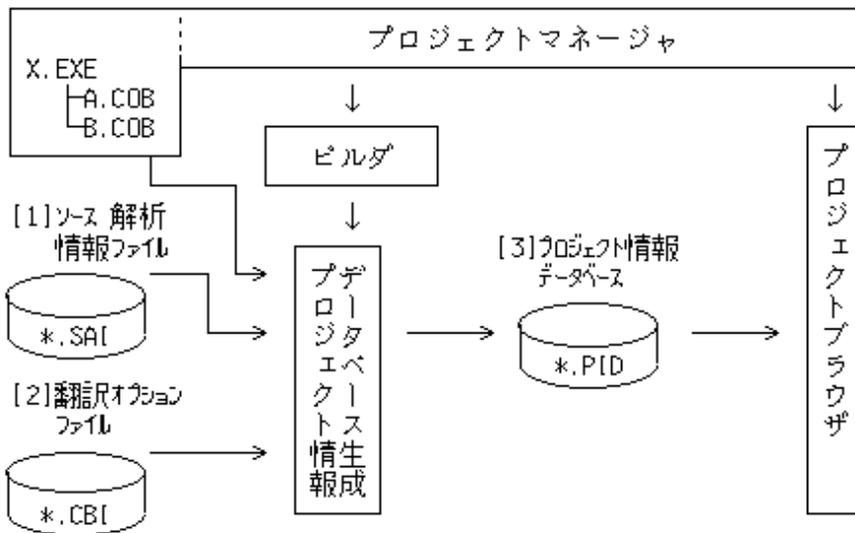
プロジェクトブラウザは、以下の機能を持っています。

- 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

### 27.6.2 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.3.48 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。



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、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。

## 27.6.3 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。



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。

## 27.6.4 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]“[27.6.5 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]“[27.6.5 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]“[27.6.5 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。

## 27.6.5 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。

### 27.6.5.1 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27.3 プロジェクトブラウザの情報表示](#)”に説明します。

表27.3 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
参照関係	クラス プログラム 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 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 メソッド	リストビューで表示
ファイル名	クラス プログラム メソッド	リストビューで表示

### 27.6.5.2 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。

### 27.6.5.3 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。

### 27.6.5.4 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。

### 27.6.5.5 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。

### 27.6.5.6 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。

### 27.6.5.7 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。

### 27.6.5.8 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。

### 27.6.5.9 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。

### 27.6.5.10 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。

### 27.6.5.11 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。

### 27.6.5.12 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。

### 27.6.5.13 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。

### 27.6.5.14 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。

## 第28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。

### 28.1 COM機能を利用したアプリケーション

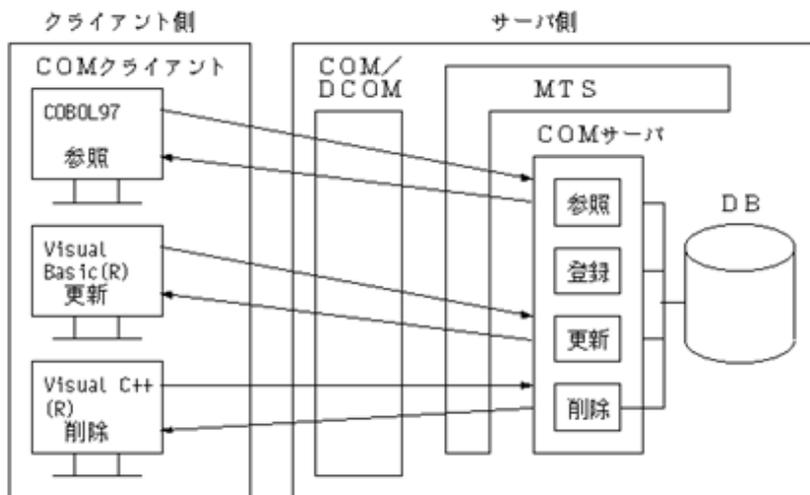
COMとは、ソフトウェアが他のソフトウェアにサービスを提供するための共通な定義です。サービスを提供する側をCOMサーバ、供給される側をCOMクライアントと呼びます。このCOMの共通定義に従ったアプリケーションを作成することで、クライアントやサーバの開発言語に依存しない、言語の特性を活かしたシステムを構築できるメリットがあります。このメリットを活かすことで、両方ともCOBOLであるシステムの組み合わせはもちろんのこと、画面入力するアプリケーション(COMクライアント)をVisual Basic、基幹業務のアプリケーション(COMサーバ)をCOBOLで作成するといったように、それぞれの開発に合った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。

#### 28.1.1 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28.1 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“[28.4 DCOMとMTS](#)”を参照してください。

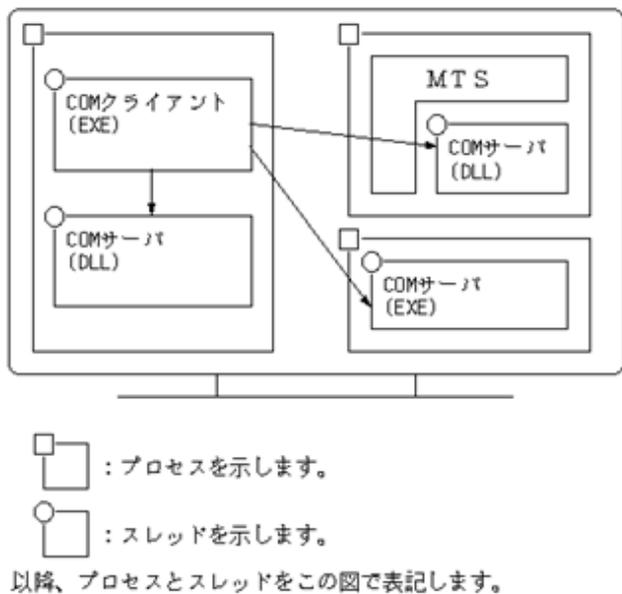
#### 28.1.2 COMを利用したアプリケーション形態

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28.2 同一マシン上での動作](#)”に示す運用が可能です。

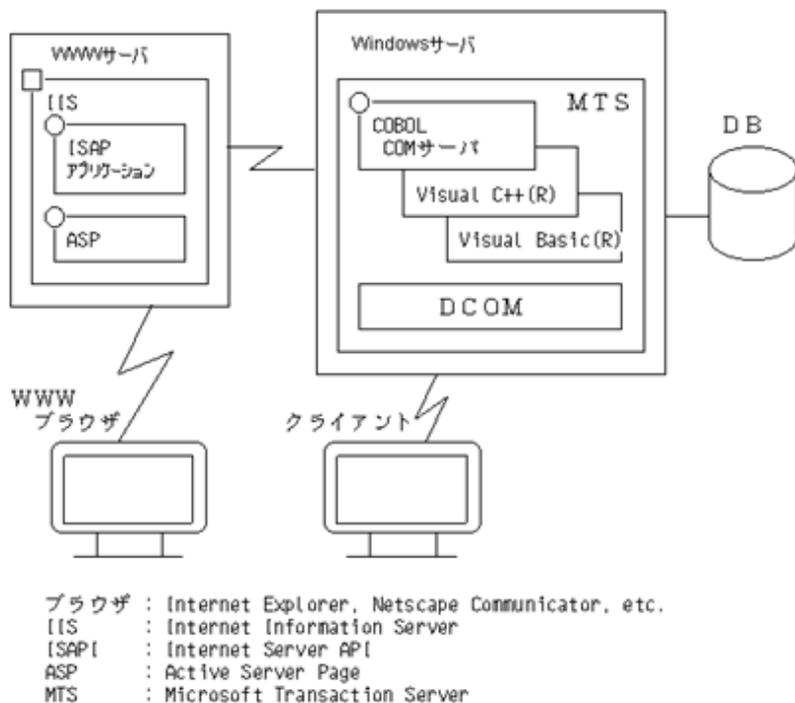
COMクライアントプログラムから、同じプロセス内にダイナミックリンクライブラリ(以降の説明では単にDLLといいます)で実装したCOMサーバをロードし連携することができます。また、MTS配下にインストールした、DLLで実装されたCOMサーバや実行可能プログラム(以降の説明では単にEXEといいます)で実装されたCOMサーバは、別プロセスとして連携することができます。

図28.2 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28.3 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28.3 DCOMを利用したシステム



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。

### 28.1.3 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17章 オブジェクト指向プログラミング機能](#)”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“[28.2 COMクライアント機能](#)”を参照してください。

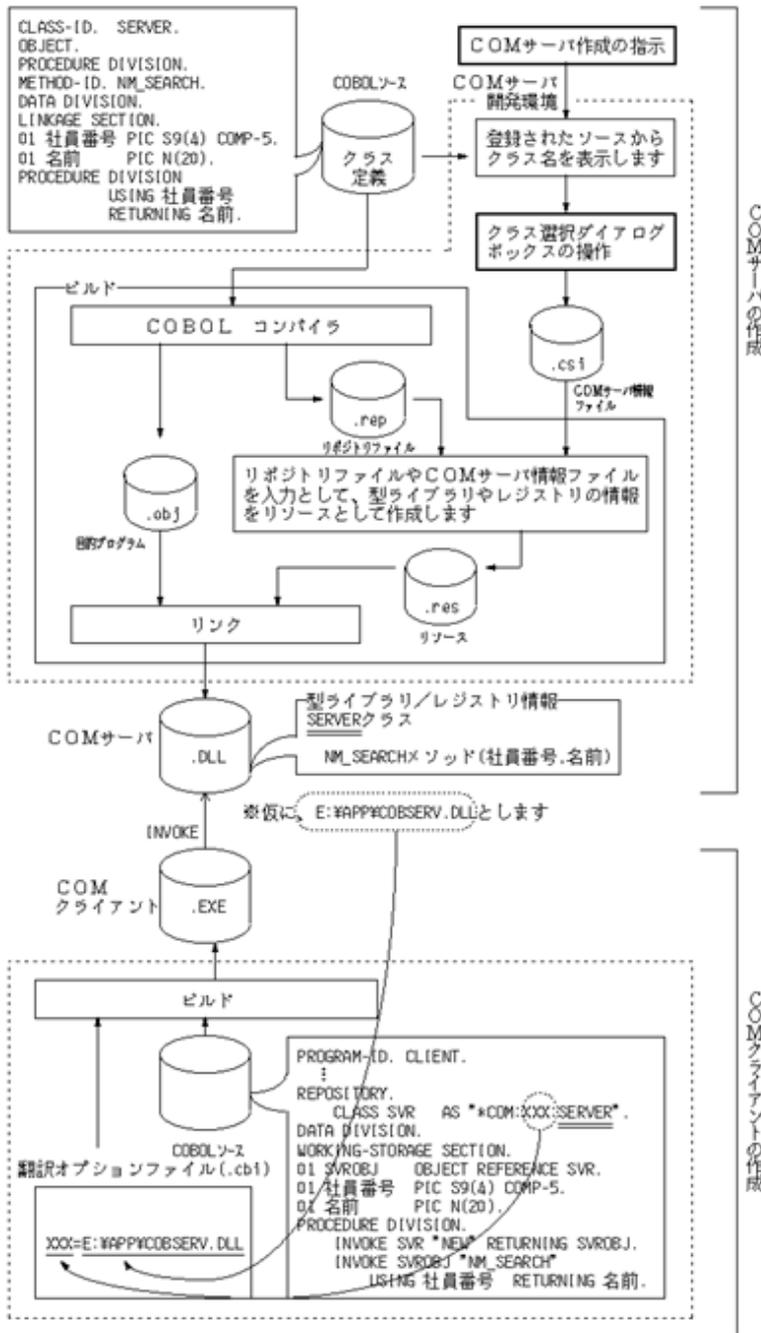
COM機能を利用するサーバアプリケーションの作成方法については“[28.3 COMサーバ機能](#)”を参照してください。

### 28.1.4 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28.4 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“28.8 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、“28.8.1 COMサーバ名の設定”を参照してください

注：アライバインドについては、“28.2.2 アライバインドとレイバインド”を参照してください。

## 28.2 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。

### 28.2.1 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

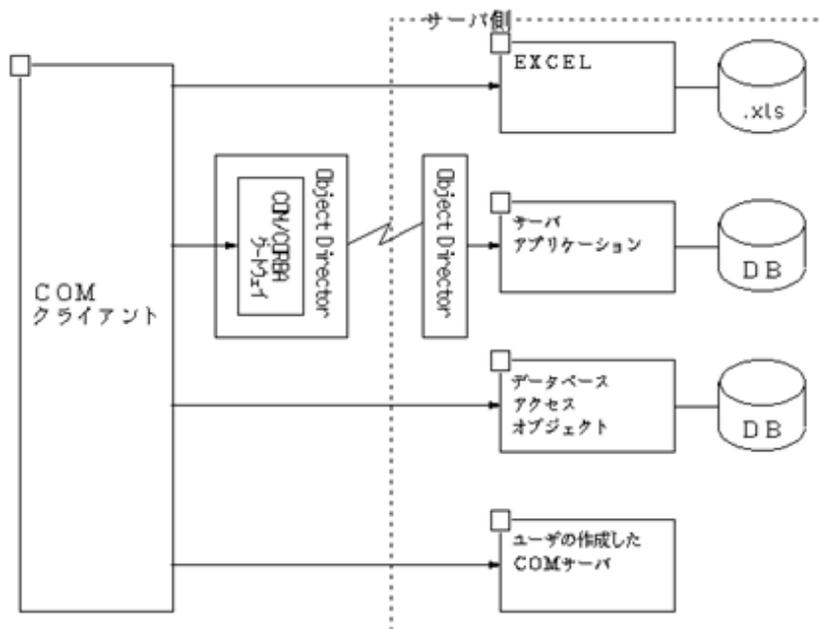


#### 参考

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28.5 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、“28.2.3 \*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]“28.6.1 \*COM-ARRAYクラス”

## 28.2.2 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。

### 28.2.2.1 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]“28.8.1 COMサーバ名の設定”



- ・ 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めません。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.3.2 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28.1 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 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 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 定数をデータ名に転記して指定する	記述できる 直接INVOKE文のBY CONTENTや行内呼出しに定数の記述可能

項目	レイトバインド	アーリバインド
型ライブラリの指定	指定の必要なし	必須 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 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 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 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。

## 28.2.2.2 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。

### 28.2.2.3 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

### 28.2.2.4 .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、アーリ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JMN5505I-S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。

## 28.2.3 \*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。

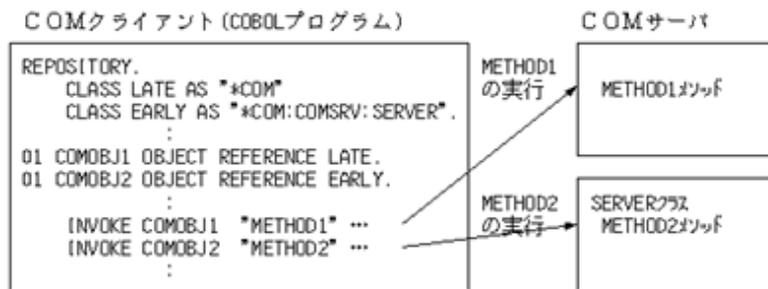
\*COMクラスにはアーリバインドとレイトバインドがあるため、以降の説明では便宜上、アーリバインドを指す場合には“\*COMアーリバインドクラス”、レイトバインドを指す場合には“\*COMレイトバインドクラス”、両方を指す場合には“\*COMクラス”と呼びます。



\*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28.6 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28.6 COMサーバの操作



### 28.2.3.1 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 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
MOVE W-RATE TO EXRATE100 OF EXCHGOBJ.
DISPLAY NC" 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“[28.3 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。

### サーバをアーリ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(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 "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.
```

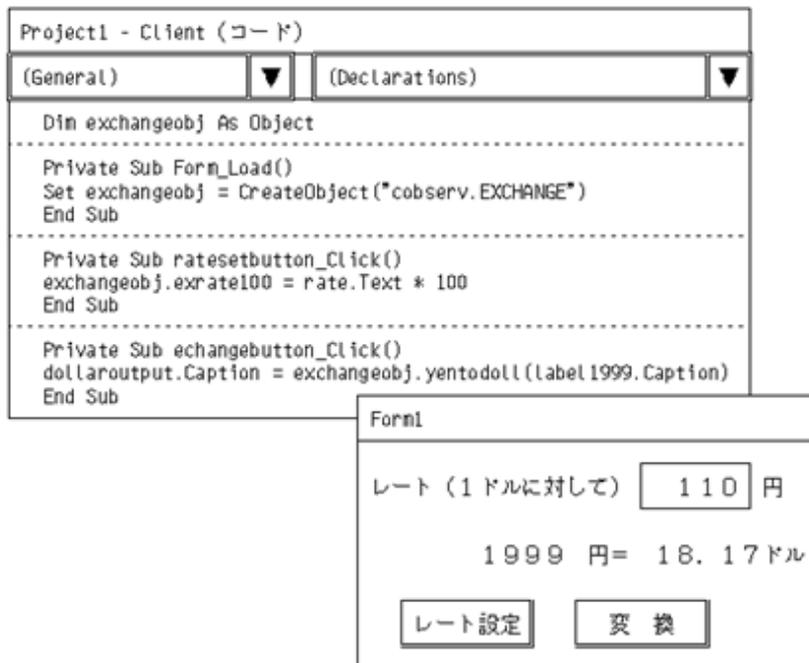
- [1] COMサーバとなるEXCHANGEクラスを特殊クラス化するため、“\*COM:COMサーバ名(任意):クラス名”を指定する。

アーリ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28.7 Visual Basicの場合



## 28.2.4 \*COMレートバインドクラス

ここでは、\*COMレートバインドクラスについて説明します。

### 28.2.4.1 \*COMレートバインドクラスのファクトリメソッド

\*COMレー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28.2 \*COMレー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。

## 28.2.4.2 \*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

\*COMレイト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。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28.3 COM操作のメソッド/プロパティ”に示します。

表28.3 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) “SET-プロパティ名”メソッド(設定) “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、“28.5 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。
```

## 28.2.5 \*COMアーライバインドクラス

ここでは、\*COMアーライバインドクラスについて説明します。

### 28.2.5.1 \*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28.4 \*COMアーリ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。

### 28.2.5.2 \*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

\*COMアーリ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レイトバインドクラスと同じ書き方で呼び出します。\*COMレイトバインドクラスの書き方については、“28.2.5.2 \*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”を参照してください。



“28.2.5.2 \*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”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、アーリバインドの場合、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、リポジット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。

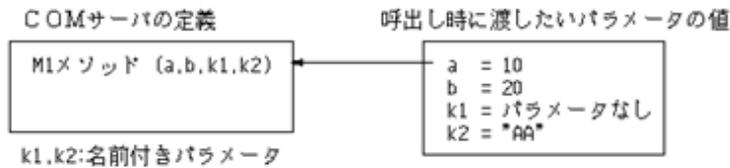
### 28.2.6 名前付きパラメタ

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]“28.2.7 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。

### 28.2.7 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.
  
```

## 28.2.8 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。

### 28.2.8.1 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	• 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
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	• COMサーバが立ち上がっていない • COMサーバに指定したクラスやメソッドが存在しない
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	• 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 • 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

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、“28.6.2 \*COM-EXCEPTIONクラス”を参照してください。

### 28.2.8.2 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 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、“メッセージ集”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。

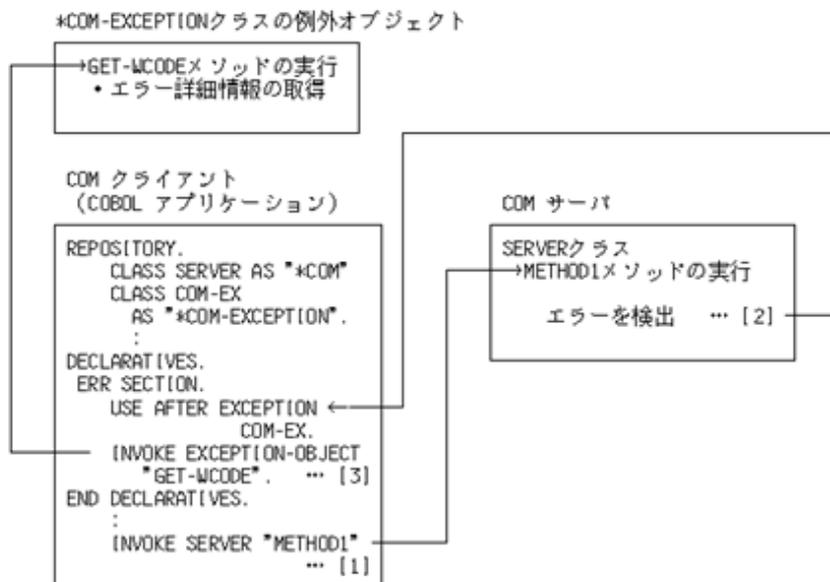
### 28.2.8.3 例外処理の流れ

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28.8 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、“17.3.1.2 例外オブジェクト”を参照してください。

## 28.2.8.4 \*COM-EXCEPTIONクラスの使用例

### アプリケーションの概略

- レイトバインドでCOMサーバと連携します。
- COMサーバのプログラムIDを”COMSERVER.COMCLASS”とします。
- 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。
- 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 ... [2]
        DISPLAY “ERROR OCCURED IN ” METHOD-NAME “METHOD”
        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。

## 28.2.9 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 アーリ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。

アーリ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リ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、アーリ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、アーリ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ぬのであれば、最初にアーリ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。

## 28.3 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。

## 28.3.1 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。

## 28.3.2 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28.5 COBOLで作成したCOMサーバの特徴”にまとめます。

表28.5 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
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、“28.9 COM関連の用語の説明”を参照してください。

## 28.3.3 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、“28.8.2 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、“28.5 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、“[28.2.3.1 プログラミング例](#)”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20章 マルチスレッド](#)”を参照してください。

2. 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。
3. 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、“[28.8.2 COMサーバ作成機能の使い方](#)”で説明します。
4. 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。

## 28.3.4 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

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。

## 28.3.5 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。

## 28.3.6 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を発生させる場合、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

“17.3.1.4 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

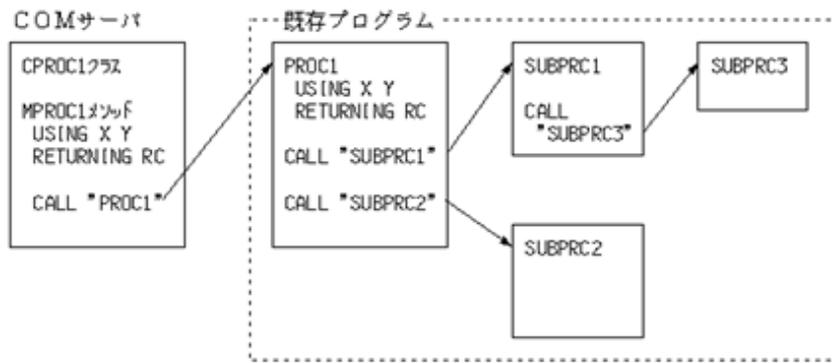
COMサーバプログラムでCOBOLのエラーを検出した場合、COBOLランタイムシステムは、WCODEが'1001'の例外を返却します。詳細については、“[28.6.2 \\*COM-EXCEPTIONクラス](#)”を参照してください。

## 28.3.7 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、PROCI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28.9 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。

## 28.3.8 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 実行環境情報の設定”
- 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 実行環境情報の設定”の“5.5.4 DLL配下にある実行用の初期化ファイルを使用する”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。

## 28.4 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



注意

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。

### 28.4.1 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28.10 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 COMA2.DLL ...	COMB1.DLL	COMC1.DLL COMC2.DLL ...	.....

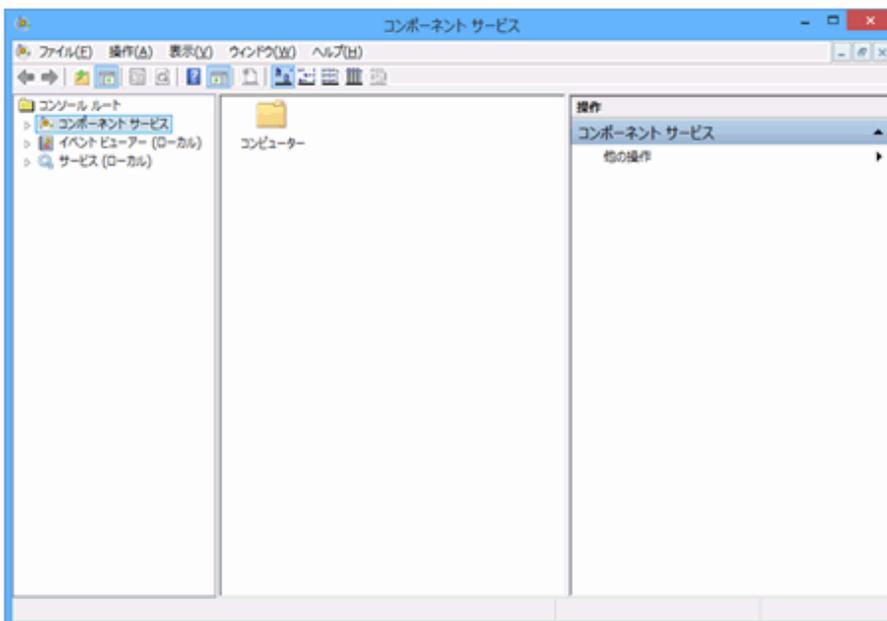
## 28.4.2 MTS環境への登録方法

COMサーバをMTSに登録する方法について説明します。

### 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、[コントロールパネル]>[管理ツール]>[コンポーネントサービス]を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

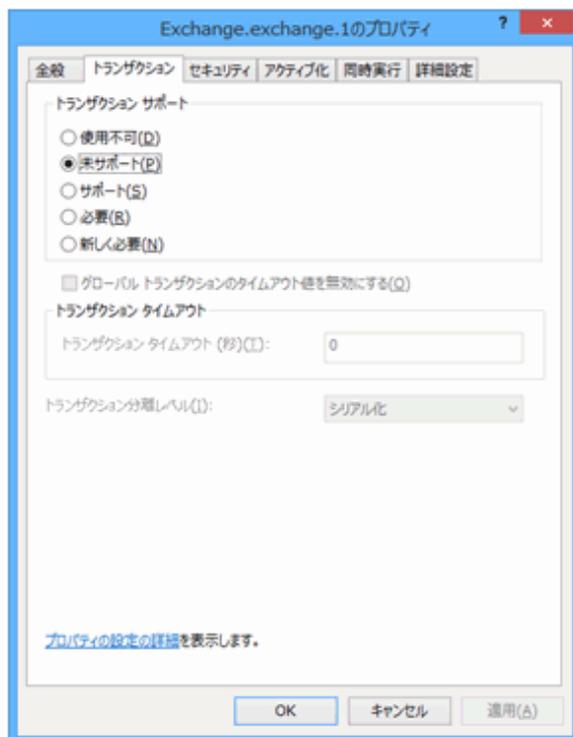
- a. コンポーネントサービス画面の[コンソールルート]から、[コンポーネントサービス]>[コンピューター]>[マイコンピューター]>[COM+アプリケーション]の順で開きます。  
[COM+アプリケーション]を選択して、[操作]メニューから[新規作成]>[アプリケーション]を選択します。  
→ [COM+ アプリケーション インストールウィザード]が表示されます。ウィザードの指示に従って設定を進めます。
- b. [新しいアプリケーションのインストールまたは作成]画面が表示されたら、“空のアプリケーションを作成する”をクリックします。  
→ [空のアプリケーションを作成]画面が表示されます。
- c. [新しいアプリケーション名の入力]に、適当な名前を入力します。この例では“mtstest”として、[次へ(N)]ボタンをクリックします。  
→ [アプリケーションIDの設定]画面が表示されます。
- d. アプリケーションIDの設定では、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。
- e. 空アプリケーションの作成が完了すると、完了画面が表示されます。

### 3. コンポーネントの登録

- a. 2.で作成したアプリケーションを開いて、コンポーネントを選択します。[操作]メニューから[新規作成]>[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]が表示されます。ウィザードの指示に従って設定を進めます。
- b. [コンポーネントのインポートまたはインストール]画面では、[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、[追加]ボタンをクリックして登録するDLLを指定します。
- c. 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(この例では、“Exchange.exchange.1”)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を選択して、[操作]メニューから[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。

### 参考

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
使用不可	—
未サポート	トランザクションをサポートしない
サポート	トランザクションをサポートする
必要	トランザクションが必要

コンポーネントサービス	MTSエクスプローラ
新しく必要	新しいトランザクションが必要

## 5. レジストリの登録

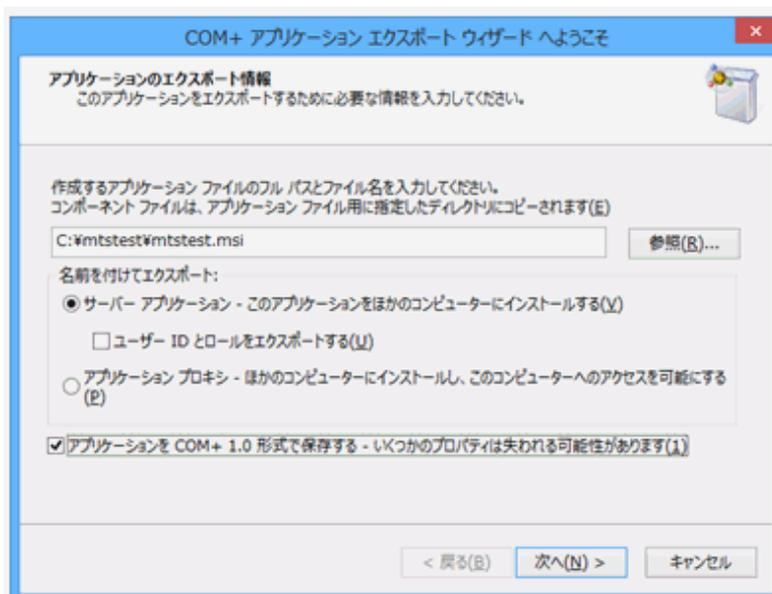
COMサーバをMTSに登録した場合、レジストリへの登録はMTSエクスプローラが行うため、REGSVR32.EXEを使って登録する必要はありません。

## 28.4.3 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

- a. 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]メニューから[エクスポート]を選択します。  
→ [COM+ アプリケーション エクスポート ウィザード]が表示されます。ウィザードの指示に従って設定を進めます。
- b. [アプリケーションのエクスポート情報]画面では以下を指定します。
  - インストーラを作成する場所を指定します。この例では、「C:\%mtstest%\%mtstest」と入力します。
  - [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ます。



ウィザードが完了すると、インストーラ(.msi)が作成されます。

### 2. クライアントマシンでの実行

作成したインストーラ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。

## 28.4.4 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(“28.4.2 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]“28.4.4.1 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]“28.4.4.3 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、“28.4.5.1 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。

### 28.4.4.1 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。

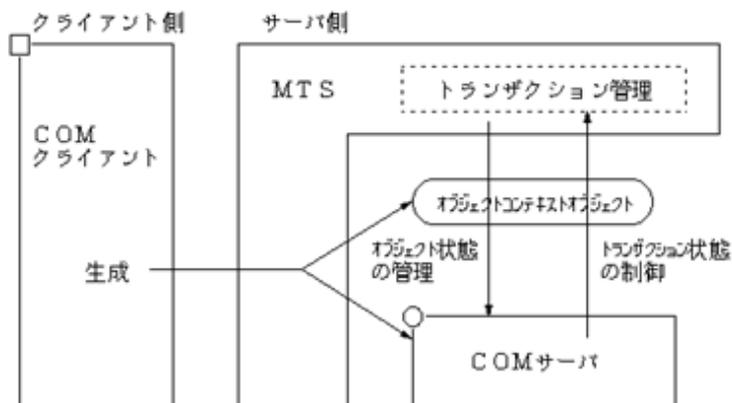
MTSはオブジェクトコンテキストオブジェクトのメソッドを呼び出すことで、以下の処理を行うことができます。

- ・ 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

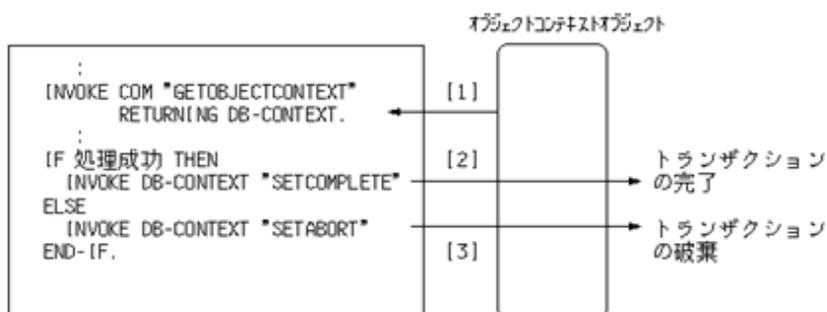
オブジェクトコンテキストオブジェクトのオブジェクト参照を取得し、返却します。引数はありません。



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、“[28.4.4.2 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、“[28.4.5.4 トランザクション範囲](#)”を参照してください。

## 28.4.4.2 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、“メッセージ集”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28.6 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 0x80040154 0x8007000E 0x80020005 0x8002000E 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。 <b>SetAbort</b> 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 0x8001FFFF 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。 <b>SetComplete</b> 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 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 0x80020005 0x8001FFFF 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 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 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、“28.4.5 MTSのトランザクション処理”を参照してください。

### 28.4.4.3 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。



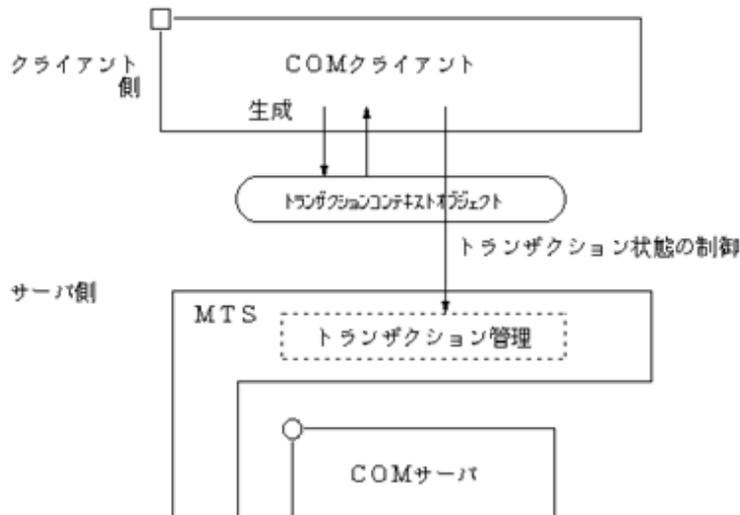
#### 参考

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”

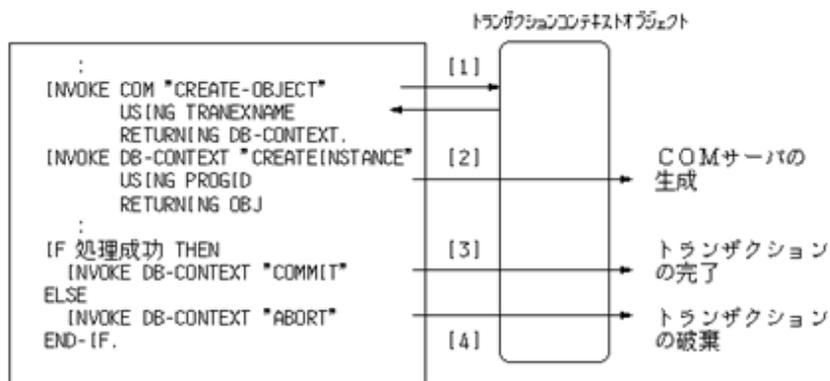
返却値 : \*COMクラスのオブジェクト参照



#### 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、“[28.4.4.4 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、“[28.4.5.4 トランザクション範囲](#)”を参照してください。

### 28.4.4.4 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、“メッセージ集”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28.7 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 0x80004005 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 0x80004005 0x8001FFFF 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> <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 0x80040154 0x8007000E 0x80020005 0x8002000E 0x8001FFFF

注1: Abortメソッド、Commitメソッドが呼び出された後に、トランザクションコンテキストオブジェクトに対して別の呼出しが行われた場合、新しいトランザクションが開始されます。

注2: Commitメソッドの実行の結果、トランザクションが破棄される場合、SCODE値(0x8004E002)の例外が発生します。

そのほかのメソッドの使い方については、“[28.4.5 MTSのトランザクション処理](#)”を参照してください。

## 28.4.4.5 注意事項

### SELFオブジェクトの扱いについて

MTSに登録されたCOMサーバのメソッド内でもSELFオブジェクトは利用できます。ただし、次の点に注意してください。

#### SELFオブジェクトでメソッドを呼び出す場合

このときのメソッド呼出しは、COMオブジェクトとしてではなく、COBOLのオブジェクトとしてのメソッド呼出しになるため、MTS環境からは連続した処理とみなされます。すなわち、SELFオブジェクトで呼び出すメソッドの最後にSetComplete/SetAbortメソッドを呼び出しても、その時点ではトランザクションは完了しません。トランザクションは、呼出し元に戻った時に完了します。

#### SELFオブジェクトをメソッドの引数として渡す場合

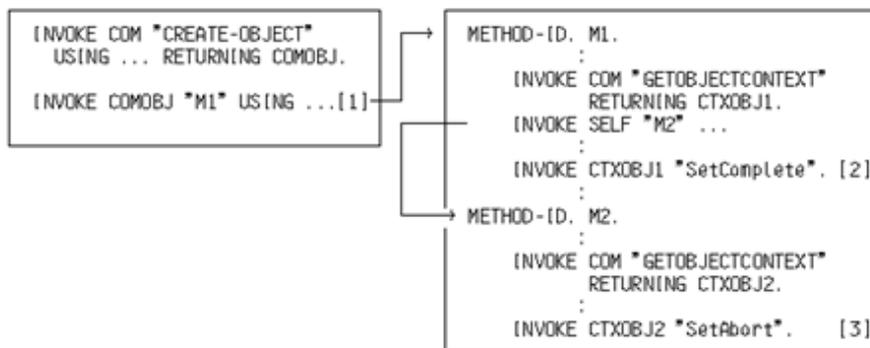
このオブジェクトは、COMオブジェクトではなく、COBOLのオブジェクトとして扱われます。受け取ったオブジェクトでメソッドを呼び出し、COBOLのオブジェクトでのメソッド呼出しになるため、MTS環境からは連続した処理とみなされます。

NetCOBOLでは、SELFオブジェクトをCOMサーバのオブジェクトとして扱う方法はありません。COMサーバ内のメソッドをCOMオブジェクトで呼び出すには、呼出し元で生成したCOMサーバのオブジェクトをメソッドの引数として受け取り、それを使用します。しかし、このときメソッド内でSetComplete/SetAbortメソッドを呼び出すと、メソッドから戻った時に、自分自身が破棄されてしまいます。このような場合、SetComplete/SetAbortメソッドの代わりに、EnableCommit/DisableCommitメソッドを使います。



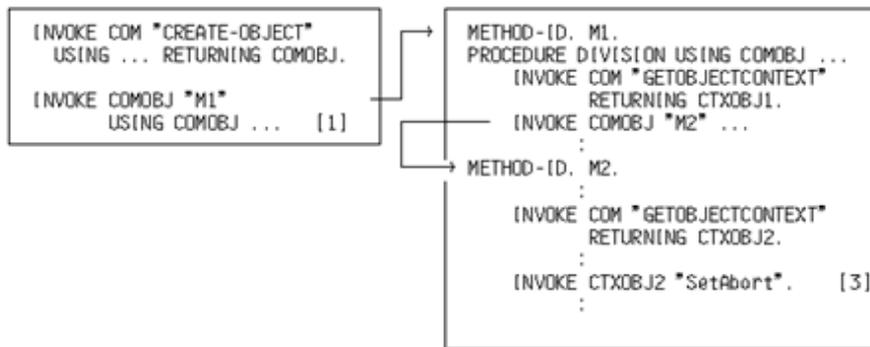
### 例

#### SELFオブジェクトのメソッド呼出し



M2の中で、SetAbortメソッドを呼び出していますが[3]、その後、呼出し元のM1でSetCompleteメソッドを呼び出しています。M2の呼出しはCOBOLのオブジェクトとしてのメソッド呼出しなので、MTSからは連続した処理とみなされ、最後に実行された[2]のSetCompleteメソッドが有効になります。トランザクションは[1]の呼出しに戻った時に完了します。

#### 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。

## 28.4.5 MTSのトランザクション処理

### 28.4.5.1 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。

### 28.4.5.2 トランザクションを制御するための追加処理

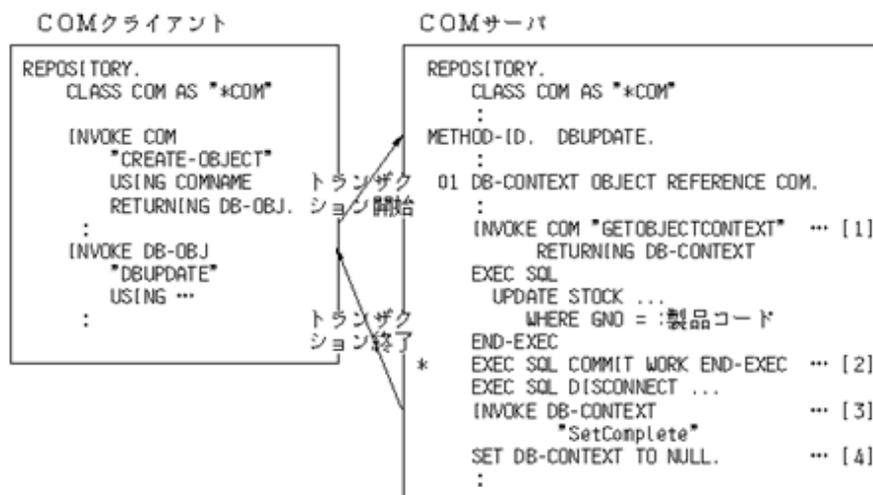
MTSでCOMサーバのトランザクションを制御するためには、COMサーバに以下の処理を追加します。

- 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28.11 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。

### 28.4.5.3 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“15.2.3.1 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. ...[2]
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、“28.4.2 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。

#### 28.4.5.4 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28.12 サーバからサーバを起動する場合

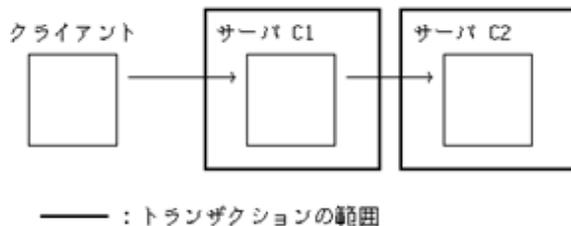
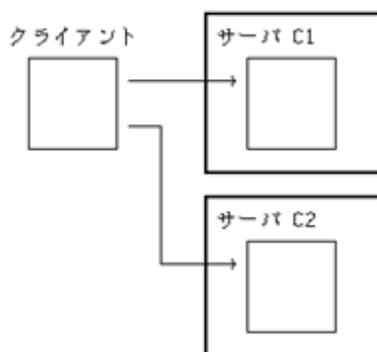


図28.13 クライアントからサーバを別々に起動する場合

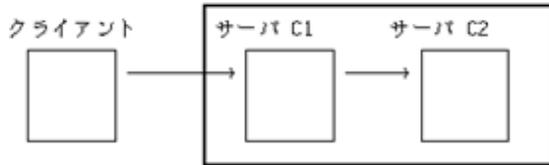


##### サーバ側でまとめて一つのトランザクションとして実行する方法

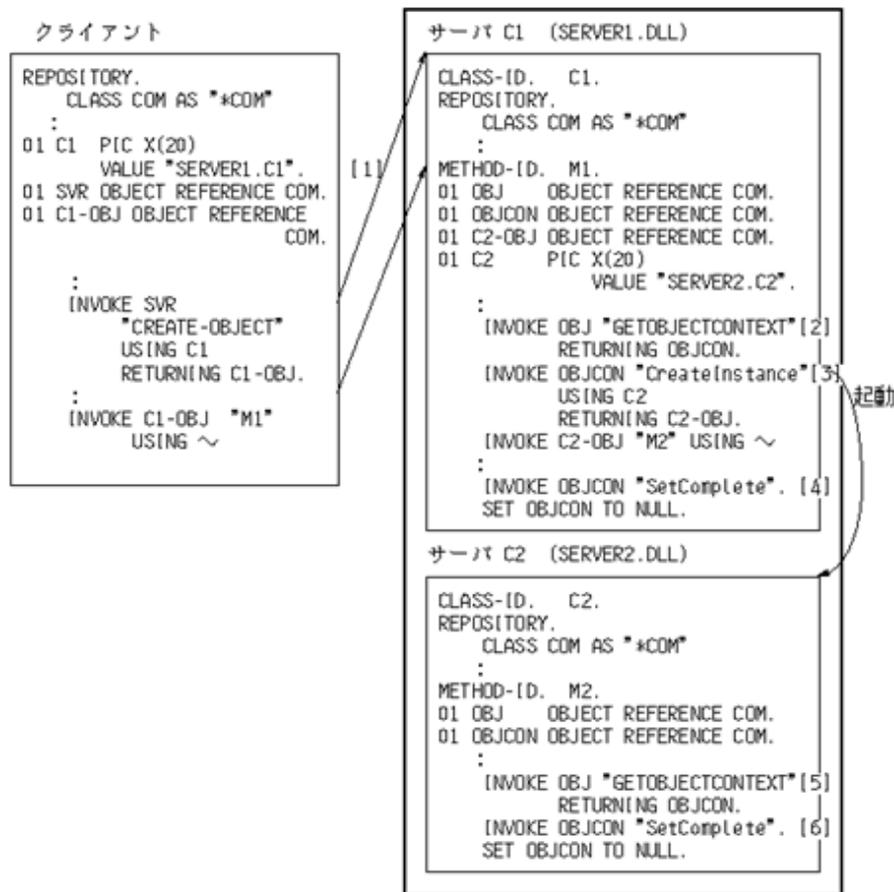
サーバC1をMTSに登録するときは、トランザクションのプロパティの設定で「トランザクションが必要」または「新しいトランザクションが必要」のどちらかを選択します。サーバC2に登録するときは、「トランザクションが必要」または「トランザクションをサポートする」のどちらかを選択します。

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28.14 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。

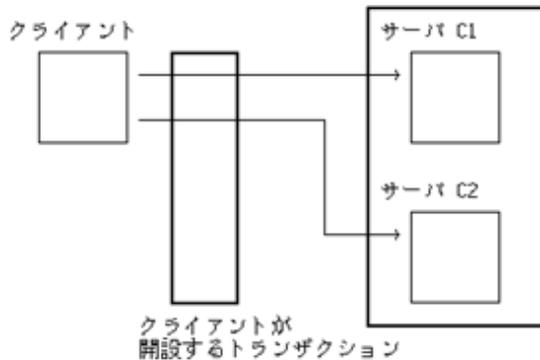
サーバC1、サーバC2を合わせたトランザクションは、両方のサーバでSetCompleteされたときに成功します。

## 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

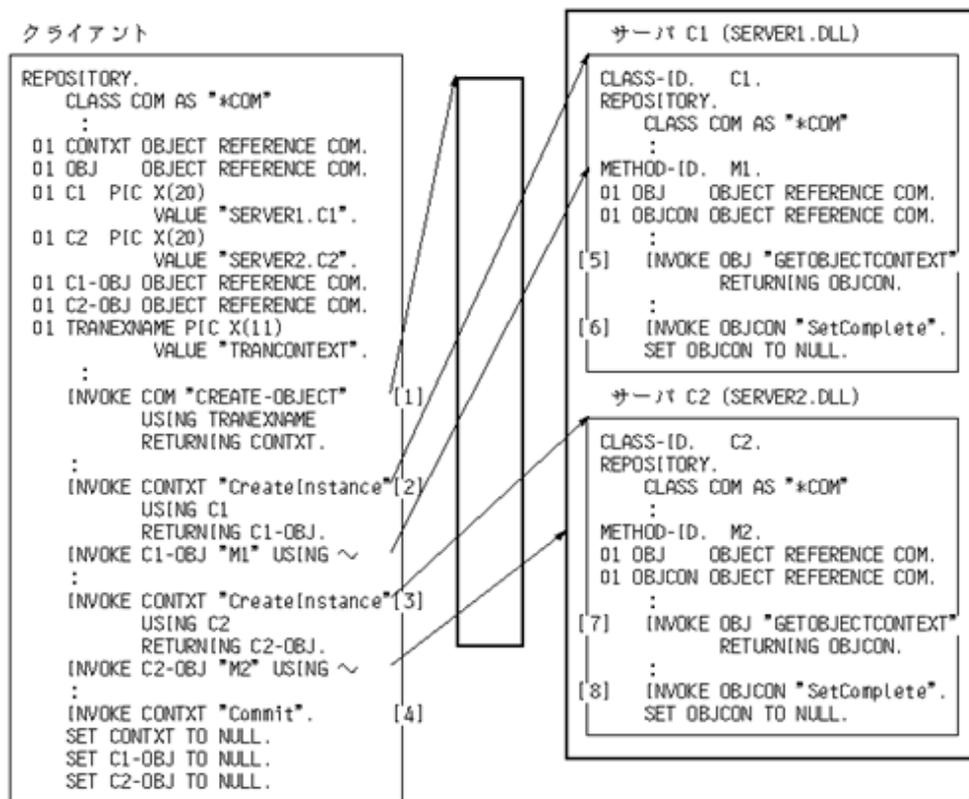
サーバC1およびサーバC2をMTSに登録するときに、トランザクションのプロパティの設定で「トランザクションが必要」または「トランザクションをサポートする」のどちらかを指定します。

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28.15 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。

## 28.4.5.5 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]“[28.3.3 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、“[5.5.2 実行用の初期化ファイルの内容](#)”および“[5.5.3 実行用の初期化ファイルの検索順序](#)”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、呼び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、呼び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]“[28.4.4.5 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び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、“[28.4.5.3 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、“15.2.8.2 ODBC情報設定ツールの使い方”を参照してください。

## 28.5 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28.16 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28.16 COBOLのデータ型とCOMのデータ型の対応

COBOLのデータ型	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	規格2進	外部10進	内部10進	英字	英数字	日本語	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				
COMのデータ型 (VT型)	*1	*1	*2			*3	*3											
VT_UI1					-		-		-							-		-
VT_I2					-		-		-							-		-
VT_I4			△		-		-		-							-	△	-
VT_CY					-	□	-	□	-	□		□		□		-		-
VT_DECIMAL					-		-		-							-		-
VT_R4					-		-		-		○					-		-
VT_R8					-		-		-		□	○				-		-
VT_BSTR				□							-	-	-			-	□	-
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28.8 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型	COMの型	意味
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_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、“[28.6.1 \\*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、“[28.6.1.5 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 28.16 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、アトリバインド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、レベルエラーメッセージを出力しています。

## 28.6 COM連携のためのその他の特殊クラス

### 28.6.1 \*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。



注意

\*COM-ARRAYクラスのメソッドはレイアウト形式で呼び出してください。

#### 28.6.1.1 \*COM-ARRAYクラスのファクトリメソッド

\*COM-ARRAY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。

- ・ NEWメソッド

上記のメソッドの機能を“[表28.9 \\*COM-ARRAYクラスのファクトリメソッド](#)”に示します。

表28.9 \*COM-ARRAYクラスのファクトリメソッド

メソッド名	引数	引数の型	意味/返却値
NEW	型指定 次元数 1次元要素数 2次元要素数 :	S9(9) COMP-5 S9(9) COMP-5 S9(9) COMP-5 S9(9) COMP-5 :	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28.11 データ型を表す数値”を参照してください。

### 28.6.1.2 \*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28.10 \*COM-ARRAYクラスのオブジェクトメソッド”に示します。

表28.10 \*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 :	S9(9) COMP-5 :	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 1次元添字 :	任意 S9(9) COMP-5 :	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過してはなりません。

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28.11 データ型を表す数値”を参照してください。

### 28.6.1.3 \*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28.11 データ型を表す数値”にその対応を示します。

表28.11 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。

### 28.6.1.4 \*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。

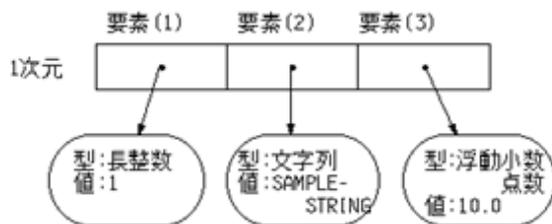
```

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

### 28.6.1.5 バリエント型データの扱い

NetCOBOLでは、COMのバリエント型(VT\_VARIANT)データをそのまま扱うことができません。バリエント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ント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ント型データの扱い

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。

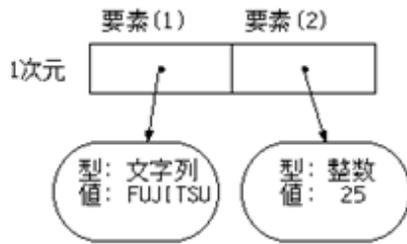
#### COMサーバに引数として渡すバリエント型データの扱い

アーリーバインドの場合、引数に指定したデータを可能なかぎりバリエント型に変換してCOMサーバに渡します。レイトバインドの場合、COMサーバにバリエント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 (バリエーション型)



返却されるデータは、次元数1、要素数2のバリエーション型配列で、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.
  :
```

#### 28.6.1.6 \*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]“[28.5 COM連携で扱うデータ型の対応](#)”

#### 28.6.2 \*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、“[28.2.8 COM機能の例外処理](#)”および“[28.3.6 例外の通知](#)”を参照してください。

## 28.6.2.1 \*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28.12 \*COM-EXCEPTIONクラスのメソッド”に示します。

表28.12 \*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) または 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。 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。
GET-SCODE	X(4) またはS9(9) 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) または 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。 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。

メソッド名	データ型(注1)	意味
GET-DESCRIPTION	X(n) または 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。 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。 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) または 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。 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。 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。 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) または 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。 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。 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) または 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。 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) または 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。 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) または 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) または 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) または 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) または 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、“メッセージ集”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、“28.2.4.1 \*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。

## 28.6.3 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、“[20.10.1 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ことがあります。

## 28.7 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、“[28.7.2 COMクライアントのデバッグ方法](#)”、“[28.7.3 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

### 28.7.1 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]
```

## アーリバインドの場合

```
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

レイトバインドの場合、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。アーリバインドの場合は、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。

## 28.7.2 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。

## 28.7.3 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。

### 28.7.3.1 デバッグを起動する

デバッグ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“[24.2 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グを起動するために、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グを起動する(TEST)を指定します。“TEST”に続けてデバッグ起動パラメタを指定できます。起動パラメタの詳細は、“[表24.1 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、デバッグが自動的に起動されます。デバッグが起動されたら、[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。

### 28.7.3.2 デバッグを行う

デバッグ操作は、デバッグを使用した通常のデバッグと同じです。デバッグの使用方法については、“[第24章 対話型デバッグの使い方](#)”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、デバッグのメニューから[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。
- デバッグ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、デバッグのメニューから[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。

## 28.7.4 COMクライアントとCOMサーバを通じたデバッグ方法

COMクライアントとCOMサーバの両方を通じ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グとCOMサーバ側のデバッグは連携しません。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サーバ側のデバッグは中断しません。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

## 28.8 開発環境

---

### 28.8.1 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]“25.1.4 [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

#### [COMサーバ名の設定]ダイアログ

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。

## 28.8.2 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。“27.2 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]メニューから、[オプション] > [COMサーバ] > [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。

- [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

#### 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/プロパティの返却値またはパラメタの属性に変更された。

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は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。

## 28.9 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タの管理を行います。
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、“[28.4 DCOMとMTS](#)”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.3 翻訳オプション一覧”から翻訳オプションを確認し、“A.1 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。

## A.1 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オプション以外のコマンドオプションによる指定
3. -WCオプションによる指定
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

“J.1.14 -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



“1.2.1 環境変数の設定”

2. 翻訳コマンドによる指定



“J.1.21 -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。



- 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
```

## A.2 翻訳オプションの指定形式

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

[翻訳オプション]ダイアログから指定できます。



V10以降より追加された翻訳オプションを[翻訳オプション]ダイアログから指定する場合は、[その他のオプション]エディットボックスから指定してください。詳細は、“A.6 V10以降より追加された翻訳オプション”を参照してください。

### STUDIO

NetCOBOL Studioの[翻訳オプション]ページから指定できます。

### -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、「+」、「,」、「:」、「=」、「[」、「]」、「(」、「)」、「'」



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。

### COBOL32コマンドを使用する場合

COBOL32コマンドを実行したフォルダです。

### NetCOBOL Studioからビルドする場合

プロジェクトフォルダがカレントフォルダです。



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.3 翻訳オプション一覧

### 翻訳時の資源に関するもの

- “A.3.1 AIMLIB (サブスキーマ定義ファイルのフォルダの指定)”
- “A.3.20 FORMEXT (画面帳票定義体ファイルの拡張子の指定)”
- “A.3.21 FORMLIB (画面帳票定義体ファイルのフォルダの指定)”
- “A.3.25 LIB (登録集ファイルのフォルダの指定)”
- “A.3.26 LIBEXT (登録集ファイルの拡張子の指定)”

- “A.3.45 REP (リポジトリファイルの入出力先フォルダの指定)”
- “A.3.46 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.3.8 COPY (登録集原文の表示)”
- “A.3.22 GEN (FCOMおよびUWAの表示)”
- “A.3.27 LINECOUNT (翻訳リストの1ページあたりの行数)”
- “A.3.28 LINESIZE (翻訳リストの1行あたりの文字数)”
- “A.3.29 LIST (目的プログラムリストの出力の可否)”
- “A.3.31 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.3.32 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.3.39 NUMBER (ソースプログラムの一連番号領域の指定)”
- “A.3.42 PRINT (各種翻訳リストの出力の可否および出力先の指定)”
- “A.3.53 SOURCE (ソースプログラムリストの出力の可否)”
- “A.3.64 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.3.6 CONF (規格の違いによるメッセージの出力の可否)”
- “A.3.18 FLAG (診断メッセージのレベル)”
- “A.3.19 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.3.2 ALPHAL (英小文字の扱い)”
- “A.3.4 BINARY (2進項目の扱い)”
- “A.3.11 CURRENCY (通貨編集用文字の扱い)”
- “A.3.17 FILELIT (ファイル識別名定数の扱い)”
- “A.3.14 DUPCHAR (重複文字の扱い)”
- “A.3.24 LANGLVL (ANSI COBOL規格の指定)”
- “A.3.23 INITVALUE (作業場所節でのVALUE句なし項目の扱い)”
- “A.3.33 MF (Micro Focus互換モードの指定)”
- “A.3.36 NCW (日本語利用者語の文字集合の指定)”
- “A.3.38 NSPCOMP (日本語空白の比較方法の指定)”
- “A.3.43 QUOTE/APOST (表意定数QUOTEの扱い)”
- “A.3.47 RSV (予約語の種類)”
- “A.3.49 SCS (ソースファイルのコード系)”
- “A.3.50 SDS (符号付き10進項目の符号の整形の可否)”
- “A.3.51 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.3.54 SQLGRP (SQLのホスト変数定義の拡張)”
- “A.3.55 SRF (正書法の種類)”

- “A.3.58 STD1 (英数字の文字の大小順序の指定)”
- “A.3.59 TAB (タブの扱い)”
- “A.3.65 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.3.48 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.3.3 ASCOMP5 (2進項目の解釈の指定)”
- “A.3.7 CONVCHAR (コンパイラが使用するコード変換ライブラリ)”
- “A.3.10 CREATE (創成ファイルの指定)”
- “A.3.12 DLOAD (プログラム構造の指定)”
- “A.3.15 ENCODE (データ項目のエンコードの指定)”
- “A.3.30 MAIN (主プログラム/副プログラムの指定)”
- “A.3.34 MODE (ACCEPT文の動作の指定)”
- “A.3.35 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.3.37 NSP (日本語項目における空白の扱い)”
- “A.3.40 OBJECT (目的プログラムの出力の可否)”
- “A.3.41 OPTIMIZE (広域最適化の扱い)”
- “A.3.44 RCS (実行時コード系の指定)”
- “A.3.61 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.3.13 DNTB (DISPLAY-OF関数およびNATIONAL-OF関数における後置空白の扱い)”
- “A.3.16 EQUALS (SORT文での同一キーデータの処理方法)”
- “A.3.63 TRUNC (桁落とし処理の可否)”

### 実行時の資源に関するもの

- “A.3.52 SMSIZE (PowerSORTが使用するメモリ容量を指定)”
- “A.3.56 SSIN (ACCEPT文のデータの入力先)”
- “A.3.57 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.3.5 CHECK (CHECK機能の使用の可否)”
- “A.3.9 COUNT (COUNT機能の使用の可否)”
- “A.3.60 TEST (デバッグ機能の使用の可否)”
- “A.3.62 TRACE (TRACE機能の使用の可否)”

## A.3.1 AIMLIB (サブスキーマ定義ファイルのフォルダの指定)

---

STUDIO, 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”、“J.1.3 -A(サブスキーマ定義ファイルのフォルダの指定)”

## A.3.2 ALPHAL(英小文字の扱い)

STUDIO, WINCOB, -WC, @

$$\left\{ \begin{array}{l} \underline{\text{ALPHAL}}[( \left\{ \begin{array}{l} \underline{\text{ALL}} \\ \underline{\text{WORD}} \end{array} \right\} )] \\ \underline{\text{NOALPHAL}} \end{array} \right\}$$

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.3.3 ASCOMP5(2進項目の解釈の指定)

STUDIO, WINCOB, -WC, @

ASCOMP5( { NONE  
ALL  
BINARY  
COMP } )

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。

## A.3.4 BINARY(2進項目の扱い)

STUDIO, WINCOB, -WC, @

BINARY( { WORD[, { MLBON  
MLBOFF } ] } ,  
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

- ・ クラス定義にBINARY(BYTE)を指定できません。
- ・ 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.3.5 CHECK(CHECK機能の使用の可否)

STUDIO, 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>と呼ばれる側が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>

- 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)
  - 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)
- 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。

## 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進項目が参照で使用される場合、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

“21.3 CHECK機能”

### A.3.6 CONF(規格の違いによるメッセージの出力の可否)

STUDIO, WINCOB, -WC, @

```

{
  CONF( { 68
        { 74
        { OBS
        }
      } )
  NOCONF
}

```

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LANGLVL(85)を指定した場合だけ意味を持ちます。

### 参照

“A.3.24 LANGLVL (ANSI COBOL規格の指定)”

### ポイント

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.3.7 CONVCHAR (コンパイラが使用するコード変換ライブラリ)

STUDIO, -WC

```
CONVCHAR( { ICONV } )  
          { FJICONV }
```

コンパイラが文字コード変換に使用するコード変換ライブラリを指定します。

- **ICONV**

NetCOBOL製品に組み込んでいるコード変換ライブラリを使用します。

- **FJICONV**

開発環境にインストールされたInterstage Charset Managerを使用します。Interstage Charset Managerでは、利用者が変換定義をカスタマイズすることができます。

### 注意

- 帳票定義体を読み込む際のコード変換には、本指定は有効になりません。Interstage Charset Managerがインストールされている環境ではInterstage Charset Managerを使用し、Interstage Charset Managerがインストールされていない環境では、NetCOBOL製品に組み込んでいるコード変換ライブラリを使用します。
- FJICONVを指定した場合、リモート開発時および運用時のコード変換ライブラリも、同じInterstage Charset Managerのコード変換ライブラリを使用してください。
  - “C.2.13 @CBR\_CONVERT\_CHARACTER (コード変換ライブラリの指定)”
- FJICONVを指定した場合、ソースファイルのコード系にANSIコードページ(SCS(ACP))は指定できません。シフトJIS(SCS(SJIS))またはUTF-8(SCS(UTF8))を指定してください。[参考]“A.3.49 SCS (ソースファイルのコード系)”

### A.3.8 COPY(登録集原文の表示)

---

STUDIO, 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。



“A.3.53 SOURCE(ソースプログラムリストの出力の可否)”

---



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.3.9 COUNT(COUNT機能の使用の可否)

---

STUDIO, 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。



- 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。
- 



“21.5 COUNT機能”

---

### A.3.10 CREATE(創成ファイルの指定)

---

WINCOB, -WC, @

---

CREATE [( { OBJ  
REP } )]

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.3.11 CURRENCY(通貨編集用文字の扱い)

STUDIO, WINCOB, -WC, @

CURRENCY ( { ¥  
\$ } )

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.3.12 DLOAD(プログラム構造の指定)

STUDIO, WINCOB, -WC, @

{ DLOAD  
NODLOAD }

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”

“20.10.5 動的プログラム構造”(マルチスレッドの動的プログラム構造)

### A.3.13 DNTB(DISPLAY-OF関数およびNATIONAL-OF関数における後置空白の扱い)

-WC, @

$$\left\{ \begin{array}{l} \text{DNTB} \\ \text{NODNTB} \end{array} \right\}$$

DISPLAY-OF関数およびNATIONAL-OF関数の引数-1に指定されたデータ項目について、後置空白を変換の対象にする(DNTB)か、しない(NODNTB)か、を指定します。

### A.3.14 DUPCHAR(重複文字の扱い)

STUDIO, WINCOB, -WC, @

$$\text{DUPCHAR} \left( \left\{ \begin{array}{l} \text{STD} \\ \text{EXT} \end{array} \right\} \right)$$

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)	- (X"EFBC8D")
DUPCHAR(STD)	- (X"E28892")



参考

“M.4.2 JIS非漢字の負号について”

### A.3.15 ENCODE(データ項目のエンコードの指定)

STUDIO, -WC, @

$$\text{ENCODE} \left( \left\{ \begin{array}{l} \text{SJIS} \\ \text{UTF8} \end{array} \right\} \left[ , \left\{ \begin{array}{l} \text{SJIS} \\ \left\{ \begin{array}{l} \text{UTF16} \\ \text{UTF32} \end{array} \right\} \left[ , \left\{ \begin{array}{l} \text{LE} \\ \text{BE} \end{array} \right\} \right] \right\} \right] \right\} \right)$$

英数字項目および日本語項目のエンコードを指定します。

第一サブオペランドには英数字項目のエンコード、第二サブオペランドには日本語項目のエンコード、第三サブオペランドには日本語項目のエンコードがUnicodeの場合のエンディアンを指定します。

- ・ ENCODE(SJIS,SJIS): 英数字項目のエンコードをSJIS、日本語項目のエンコードをSJISに指定します。
- ・ ENCODE(SJIS): 英数字項目のエンコードをSJIS、日本語項目のエンコードをSJISに指定します。
- ・ ENCODE(UTF8,UTF16): 英数字項目のエンコードをUTF8、日本語項目のエンコードをUTF16に指定します。

- ENCODE(UTF8): 英数字項目のエンコードをUTF8、日本語項目のエンコードをUTF16に指定します。
- ENCODE(UTF8,UTF32): 英数字項目のエンコードをUTF8、日本語項目のエンコードをUTF32に指定します。

日本語項目がUTF16またはUTF32の場合はエンディアンをリトルエンディアンにするかビッグエンディアンにするかを指定します

- ENCODE(UTF8,UTF16,LE): 日本語項目のエンディアンはリトルエンディアンにします。
- ENCODE(UTF8,UTF32,LE): 日本語項目のエンディアンはリトルエンディアンにします。
- ENCODE(UTF8,UTF16,BE): 日本語項目のエンディアンはビッグエンディアンにします。
- ENCODE(UTF8,UTF32,BE): 日本語項目のエンディアンはビッグエンディアンにします。

各サブオペランドを省略した場合は、以下となります。

- 日本語項目のエンコードを省略した場合、以下とみなします。
  - 英数字項目のエンコードがSJISの場合、日本語項目のエンコードはSJISとみなします。
  - 英数字項目のエンコードがUTF8の場合、日本語項目のエンコードはUTF16とみなします。
- 日本語項目のエンコードがUnicodeの場合で、エンディアンの指定を省略した場合、システムのエンディアンに従います。

翻訳オプションENCODEを指定した場合、実行時コード系はUnicodeになります。ただし、翻訳オプションRCSを指定した場合は、その指定が優先されます。

## 注意

- 翻訳オプションRCS(ACP)を明に指定した場合、ENCODEオプションは指定できません。
- 翻訳オプションRCS(SJIS)を明に指定した場合、ENCODE(SJIS,SJIS)以外の指定はできません。
- 翻訳オプションRCS(UTF16)を明に指定した場合、ENCODE(UTF8,UTF16)以外の指定はできません。
- 翻訳オプションENCODE(SJIS,SJIS)を明または暗に指定した場合は、ALPHABET句で以下に関連づけられた符号系名をENCODING句に指定できません。
  - UTF8
  - UTF16
  - UTF16LE
  - UTF16BE
  - UTF32
  - UTF32LE
  - UTF32BE
- 翻訳オプションENCODE(UTF8,UTF16)またはENCODE(UTF8,UTF32)を明または暗に指定した場合は、ALPHABET句でSJISに関連づけられた符号系名を、ENCODING句に指定できません。
- データ項目にENCODING句が明または暗に指定されている場合は、ENCODING句の指定が有効になります。
- ALPHABET句でUTF16またはUTF32に関連づけられた符号系名は、本オプションのエンディアンの指定に従います。

## ポイント

データ項目にENCODING句を指定した場合は、ENCODING句に指定したエンコードが有効となります。

ALPHABET句でUTF16またはUTF32を指定した場合のエンディアンは、本オプションのエンディアンの指定に従います。詳細は、“第6章 文字コード”を参照してください。

### A.3.16 EQUALS (SORT文での同一キーデータの処理方法)

STUDIO, WINCOB, -WC, @

$$\left\{ \begin{array}{l} \text{EQUALS} \\ \text{NOEQUALS} \end{array} \right\}$$

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。



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.3.17 FILELIT (ファイル識別名定数の扱い)

-WC, @

$$\text{FILELIT} \left( \left\{ \begin{array}{l} \text{LIT} \\ \text{ENV} \end{array} \right\} \right)$$

ファイル識別名定数の扱いを、ファイル識別名定数(FILELIT(LIT))とするか、ファイル識別名(FILELIT(ENV))とするか、を指定します。



翻訳オプションFILELIT(ENV)が有効な場合、環境変数名として使用できない記号類をファイル識別名定数に指定しないでください。

### A.3.18 FLAG (診断メッセージのレベル)

STUDIO, WINCOB, -WC, @

$$\text{FLAG} \left( \left\{ \begin{array}{l} \text{I} \\ \text{W} \\ \text{E} \end{array} \right\} \right)$$

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.3.6 CONF(規格の違いによるメッセージの出力の可否)”

## A.3.19 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)

STUDIO, WINCOB, -WC, @

```

{
  FLAGSW(
    [
      {
        STDM
        STDI
        STDH
      }
      ][,RPW]
    ]
    SIA
    GSW
    GSS
  )
}
NOFLAGSW

```

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
FLAGSW(GSS)	通常のホストアプリケーションの分散開発支援

### ポイント

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.3.20 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。
  - 翻訳オプションFORMEXT(複数指定の場合は指定された順)
  - 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。

- 拡張子(PMD)
- 拡張子(PXD)
- 拡張子(SMD)

日本語項目のエンコードがUTF-32の場合、UTF-32用の帳票定義体が必要です。この場合、PMDをPMU、SMDをSMUに読み替えて、1.PMU、2.SMUの順でファイルを検索します。その他の拡張子は無視されます。

UTF-32用の帳票定義体の作成については、“[J.6 UTF-32用定義体変換コマンド](#)”を参照してください。



## 参照

“1.2.1 環境変数の設定”

### A.3.21 FORMLIB(画面帳票定義体ファイルのフォルダの指定)

STUDIO, 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”

“J.1.16 -m(画面帳票定義体ファイルのフォルダの指定)”

### A.3.22 GEN(FCOMおよびUWAの表示)

STUDIO, WINCOB, -WC, @

{ GEN  
NOGEN }

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.3.23 INITVALUE(作業場所節でのVALUE句なし項目の扱い)

STUDIO, WINCOB, -WC, @

{ INITVALUE(xx)  
NOINITVALUE }

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.3.24 LANGLVL(ANSI COBOL規格の指定)

STUDIO, WINCOB, -WC, @

LANGLVL( { 85 } )  
                  { 74 }  
                  { 68 }

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.3.25 LIB (登録集ファイルのフォルダの指定)

STUDIO, 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”

“J.1.13 -I(登録集ファイルのフォルダの指定)”

### A.3.26 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.3.27 LINECOUNT (翻訳リストの1ページあたりの行数)

STUDIO, 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.3.28 LINESIZE (翻訳リストの1行あたりの文字数)

STUDIO, 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.3.29 LIST(目的プログラムリストの出力の可否)

STUDIO, WINCOB, -WC, @

```
{ LIST  
  NOLIST }
```

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、“[J.1.8 -dp\(翻訳リストファイルのフォルダの指定\)](#)”、“[J.1.18 -P\(各種翻訳リストの出力および出力先の指定\)](#)”および“[A.3.42 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.3.30 MAIN(主プログラム/副プログラムの指定)

WINCOB, -WC, @

```
{ MAIN[( { WINMAIN  
          MAIN } )] }  
  NOMAIN
```

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。
  - 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。
  - 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。
  - 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。

## ポイント

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]“11.1.4 コンソールウィンドウを使ったデータの入出力”、“C.2.12 @CBR\_CONSOLE(コンソールウィンドウの種別の指定)”

### A.3.31 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)

STUDIO, 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、“J.1.8 -dp(翻訳リストファイルのフォルダの指定)”、“J.1.18 -P(各種翻訳リストの出力および出力先の指定)”および“A.3.42 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.3.32 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)

STUDIO, 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.3.33 MF(Micro Focus互換モードの指定)

-WC, @

```
MF( { NONE  
     ALL  
     SYNONYM } )
```

---

Micro Focus互換モードの指定を行います。

- MF(NONE):

Micro Focus互換モードを使用しません。

- MF(ALL):

MF(SYNONYM)が指定されたとみなします。

- MF(SYNONYM):

Micro Focus同義語互換モードを指定します。

Micro Focus同義語互換モードを指定した場合、翻訳オプションRSVで指定した予約語に、“COBOL文法書”の“Micro Focus同義語互換モード”で示す予約語が追加されます。

### A.3.34 MODE (ACCEPT文の動作の指定)

---

STUDIO, WINCOB, -WC, @

---

MODE ( { STD  
CCVS } )

---

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。



注意

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.3.35 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, @

---

{ NAME  
NONAME }

---

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。



注意

- 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。
-



## 参考

- 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.3.36 NCW(日本語利用者語の文字集合の指定)

STUDIO, WINCOB, -WC, @

$$\text{NCW} \left( \begin{array}{c} \text{STD} \\ \text{SYS} \end{array} \right)$$

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.3.37 NSP(日本語項目における空白の扱い)

STUDIO, -WC, @

{  
  NSP  
  NONSP  
}

日本語項目における空白の扱いを日本語空白(NSP)とするか、英字空白(NONSP)とするかを指定します。

本オプションは、エンコードがUnicodeの日本語項目に対する後置空白および表意定数SPACEに作用します。日本語項目のエンコードがUnicode以外の場合は意味を持ちません。

### A.3.38 NSPCOMP(日本語空白の比較方法の指定)

STUDIO, 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、翻訳オプションによって決定されるエンコードの日本語空白はシフトJISまたはUTF16の場合には2バイトのANK空白、UTF32の場合には4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同士の比較
- ・ 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
- ・ 異なるエンコード方式の日本語項目を含む集団項目の比較
- ・ 翻訳オプションによって決定されるエンコードと異なるエンコード方式の日本語の比較

#### 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。
- ・ 部分参照付けされた集団項目は、新たな英数字項目として取り扱われるため、NSPCOMP(ASP)オプションの対象となりません。

#### 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、作用対象が集团項目のとき、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.3.39 NUMBER(ソースプログラムの一連番号領域の指定)

STUDIO, WINCOB, -WC, @

```
{ NUMBER }  
{ NONNUMBER }
```

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.3.55 SRF(正書法の種類)”

### A.3.40 OBJECT(目的プログラムの出力の可否)

STUDIO, WINCOB, -WC(F), @(F)

```
{ OBJECT [ (NetCOBOL StudioおよびWINCOBのみフォルダ指定可) ] }  
{ NOBJECT }
```

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。

NetCOBOL Studioおよび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.3.41 OPTIMIZE (広域最適化の扱い)

---

STUDIO, 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F 広域最適化”

“A.3.60 TEST (デバッグ機能の使用の可否)”

## A.3.42 PRINT (各種翻訳リストの出力の可否および出力先の指定)

---

STUDIO, 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.3.43 QUOTE/APOST (表意定数QUOTEの扱い)

---

STUDIO, WINCOB, -WC, @

{  
  QUOTE  
  APOST  
}

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.3.44 RCS(実行時コード系の指定)

STUDIO, WINCOB, -WC, @

```
RCS ( { ACP
      { SJIS
        { UTF16
          UCS2 } } , { LE
                     BE } ] )
```

実行時のコード系をANSIコードページにする(RCS(ACP))か、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。

## 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。
- V11以降では、翻訳オプションENCODEを指定することを推奨します。
- 翻訳オプションRCSを指定した場合、翻訳オプションENCODEは以下のように指定されたとみなされます。
  - RCS(ACP)を明に指定した場合、翻訳オプションENCODEは指定できません。
  - RCS(SJIS)を指定した場合、ENCODE(SJIS,SJIS)が指定されたとみなされます。
  - RCS(UTF16)またはRCS(UCS2)を指定した場合、ENCODE(UTF8,UTF16)が指定されたとみなされます。
- システムロケールが日本語の場合、RCS(ACP)とRCS(SJIS)は同義です。  
ただし、RCS(ACP)を明に指定した場合、英数字項目のエンコードはASCIIとして動作します。日本語項目の動作は保証されません。  
また、ENCODING句を指定することはできません。
- システムロケールが日本語以外の動作については、“[M.6 Windowsシステムにおけるシステムロケール](#)”を参照してください。

## 参考

“第6章 文字コード”

## A.3.45 REP(リポジットリファイルの入出力先フォルダの指定)

---

STUDIO, 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。



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
- 



“1.2.1 環境変数の設定”

“J.1.10 -dr(リポジットリファイルの入出力先フォルダの指定)”

“J.1.19 -R(リポジットリファイルの入力先フォルダの指定)”

“A.3.46 REPIN(リポジットリファイルの入力先フォルダの指定)”

---

## A.3.46 REPIN(リポジットリファイルの入力先フォルダの指定)

---

STUDIO, WINCOB

---

REPIN( フォルダ[; フォルダ ] ... )

---

リポジットリファイルの入力先フォルダを指定します。リポジットリ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。



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**

“J.1.10 -dr(リポジットリファイルの入出力先フォルダの指定)”

“J.1.19 -R(リポジットリファイルの入力先フォルダの指定)”

“A.3.45 REP(リポジットリファイルの入出力先フォルダの指定)”

## A.3.47 RSV(予約語の種類)

STUDIO, WINCOB, -WC, @

```

      ALL
      {
      V111
      V112
      V122
      V125
      V30
      V40
      V61
      V70
      V81
      V90
      V1050
      V1100
      VSR2
      VSR3
      }
RSV ( )
  
```

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(V1050)	NetCOBOL V10用
RSV(V1100)	NetCOBOL V11.0用
RSV(VSR2)	VS COBOLII REL2.0用
RSV(VSR3)	VS COBOLII REL3.0用

### A.3.48 SAI(ソース解析情報ファイルの出力の可否)

STUDIO, WINCOB, -WC, @ (F)

```

{   SAI [(NetCOBOL StudioおよびWINCOBのみフォルダ指定可)]   }
{   NOSAI                                                       }

```

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。

NetCOBOL Studioおよび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.3.49 SCS(ソースファイルのコード系)

-WC

```

SCS ( {   ACP
        {   SJIS
            UTF8
        }
    } )

```

COBOLソースファイルおよび登録集ファイルのコード系が、ANSIコードページである(SCS(ACP))か、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、[翻訳オプションENCODE\(UTF8\)](#)も指定してください。



参考

システムロケールが日本語の場合、SCS(ACP)とSCS(SJIS)は同義です。

システムロケールについては、“[M.6 Windowsシステムにおけるシステムロケール](#)”を参照してください。

### A.3.50 SDS (符号付き10進項目の符号の整形の可否)

---

STUDIO, WINCOB, -WC, @

---

{ SDS  
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.3.51 SHREXT (マルチスレッドプログラムの外部属性に関する扱い)

---

STUDIO, WINCOB, -WC, @

---

{ SHREXT  
NOSHREXT }

---

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.3.52 SMSIZE (PowerSORTが使用するメモリ容量を指定)

---

STUDIO, 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

- このオプションは、PowerSORTをインストールしている場合だけ有効になります。
- 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。

- 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。

## A.3.53 SOURCE (ソースプログラムリストの出力の可否)

STUDIO, 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、“[J.1.8 -dp \(翻訳リストファイルのフォルダの指定\)](#)”、“[J.1.18 -P \(各種翻訳リストの出力および出力先の指定\)](#)”および“[A.3.42 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.3.54 SQLGRP (SQLのホスト変数定義の拡張)

STUDIO, 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

“15.2.4 高度なデータ操作”

### A.3.55 SRF(正書法の種類)

STUDIO, WINCOB, -WC, @

```
SRF( { FIX  
      FREE  
      VAR } [, { FIX  
                FREE  
                VAR } ] )
```

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.3.56 SSIN(ACCEPT文のデータの入力先)

STUDIO, WINCOB, -WC, @

```
SSIN ( { 環境変数情報名  
        SYSIN  
        } )
```

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.3.57 SSOUT(DISPLAY文のデータの出力先)

STUDIO, WINCOB, -WC, @

SSOUT ( { 環境変数情報名 } )  
          { SYSOUT }

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.3.58 STD1(英数字の文字の大小順序の指定)

STUDIO, WINCOB, -WC, @

STD1( { ASCII  
JIS1  
JIS2 } )

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.3.59 TAB(タブの扱い)

STUDIO, WINCOB, -WC, @

TAB ( { 8  
4 } )

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.3.60 TEST(デバッグ機能の使用の可否)

STUDIO, WINCOB, -WC (F), @ (F)

{ TEST [(NetCOBOL StudioおよびWINCOBのみフォルダ指定可)]  
NOTEST } }

デバッグ機能(\*1)を使用する(TEST)か、使用しない(NOTEST)か、を指定します。

\*1: デバッグ機能は、以下を指します。

- ・ 対話型デバッガ
- ・ 診断機能
- ・ NetCOBOL Studioのデバッグ機能

TESTを指定すると、デバッガ機能で使用するデバッ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。NetCOBOL Studioおよび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.3.41 OPTIMIZE (広域最適化の扱い)”

## 参考

“3.2 翻訳に必要な資源”

“22.1 診断機能”

“第24章 対話型デバッガの使い方”

“24.2.3 プログラムのリンク”

“NetCOBOL Studio ユーザーズガイド”

## A.3.61 THREAD (マルチスレッドプログラム作成の指定)

STUDIO, WINCOB, -WC, @

---

THREAD (     {   MULTI    }  
                  {   SINGLE   }  
                  } )

---

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20章 マルチスレッド”

## A.3.62 TRACE (TRACE機能の使用の可否)

STUDIO, WINCOB, -WC, @

---

{   TRACE [(n)]  
  NOTRACE  
}

---

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.3.9 COUNT (COUNT機能の使用の可否)”

### 参考

“21.4 TRACE機能”

## A.3.63 TRUNC (桁落とし処理の可否)

STUDIO, WINCOB, -WC, @

{ TRUNC  
NOTRUNC }

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.3.41 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.3.64 XREF(相互参照リストの出力の可否)

STUDIO, 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、“[J.1.8 -dp\(翻訳リストファイルのフォルダの指定\)](#)”、“[J.1.18 -P\(各種翻訳リストの出力および出力先の指定\)](#)”および“[A.3.42 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.3.65 ZWB(符号付き外部10進項目と英数字項目の比較)

STUDIO, 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、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.4 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
- LANGLVL(74)またはLANGLVL(68)
- MAIN、MAIN(WINMAIN)またはMAIN(MAIN)

## A.5 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
- FILELIT
- DLOAD
- FLAG
- FORMEXT
- FORMLIB
- GEN
- LIB
- LINECOUNT
- LINESIZE
- LIST
- MAP
- MESSAGE
- MF
- NUMBER
- OBJECT
- PRINT
- QUOTE/APOST
- REP
- REPIN

- SAI
- SOURCE
- SRF
- TEST
- TRACE
- XREF

## A.6 V10以降より追加された翻訳オプション

---

プロジェクトマネージャおよびWINCOBの[翻訳オプション]ダイアログで、V10以降より追加された以下の翻訳オプションを指定する場合、[その他のオプション]エディットボックスに指定してください。

- DNTB
- CONVCHAR
- DUPCHAR
- ENCODE
- FILELIT
- MF
- NSP
- RSV(V81)、RSV(V90)、RSV(V1050)、RSV(V1100)
- SCS

## 付録B 翻訳リスト

翻訳時に出力する各種リストを翻訳リストといいます。

翻訳リストが出力されるファイルの指定は、“[J.1.18 -P\(各種翻訳リストの出力および出力先の指定\)](#)”を参照してください。

翻訳リストの種類		表示される内容	出力に必要な翻訳オプション(注)
診断メッセージリスト		プログラムの翻訳結果	なし
オプション情報リスト		翻訳時に確定した翻訳オプション	MESSAGE
翻訳単位統計情報リスト		プログラムのソースファイル名、翻訳日時、ソースプログラムのレコード数、目的プログラムの大きさなどの情報	MESSAGE
相互参照リスト		名標を参照する行の情報	XREF
ソースプログラムリスト		ソースプログラムの内容	SOURCE
目的プログラムリスト		機械語の文(オブジェクトコード)	LIST
データエリアに関するリスト	データマップリスト	目的プログラム内におけるデータ領域の割り付け情報およびデータの属性情報	MAP
	プログラム制御情報リスト	目的プログラム中に存在する各種作業域、データ領域の割り付け位置、および定数領域の情報	
	セクションサイズリスト	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、および実行に必要な領域の大きさ	

注：“[J.1.18 -P\(各種翻訳リストの出力および出力先の指定\)](#)”と合わせて指定が必要な翻訳オプションです。

### B.1 診断メッセージリスト

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。

#### 診断メッセージリストの出力形式

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、“メッセージ集”の“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。

### B.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が出力されます。

## オプション情報リストの出力形式

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	... [4]
	THREAD (SINGLE)		
BINARY (WORD, MLBON)	FLAG (I)	NCW (STD)	
:	:	:	

## 翻訳単位統計情報リストの出力形式

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。

## B.3 相互参照リスト

[翻訳オプションXREF](#)を指定すると、相互参照リストが出力されます。

### 相互参照リストの出力形式

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。

## B.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

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上に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。

# B.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

```

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 C705980000000000000000      mov   %GWA+00000098, 00000000 : OTHAR. 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, a1 : 除数+1
--- 13 ---              MOVE
000001A9 66A112000000      mov      ax, %COA+00000012 : +00
000001AF 66A3DC000000      mov      %GWA+000000DC, ax : 被除数
000001B5 A0DD000000      mov      a1, %GWA+000000DD : 被除数+1
000001BA 240F              and      a1, 0F
000001BC 0C30              or       a1, 30
000001BE A2DD000000      mov      %GWA+000000DD, a1 : 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+00000012 : "ABCD" .....
文字(日本語)	文字(日本語)  例 ..... mov ax, %COA+00000012 :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.3.41 OPTIMIZE(広域最適化の扱い)”

## B.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。

### B.6.1 データマップリスト

#### データマップリストの出力形式

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[12]	
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	エンコード
[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		SJIS
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		SJIS
22	COA+00000068		0	01	CEND	8	ALPHANUM	BCO. 000000		SJIS
23	COA+00000070		0	01	CRES	8	ALPHANUM	BCO. 000000		SJIS
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、“[B.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

— 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、“[22.2 翻訳リストとデバッグツールを使ったデバッグ](#)”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。

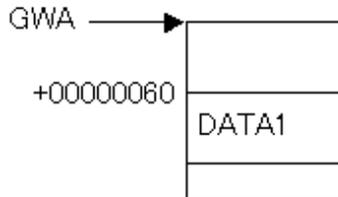
一 相対アドレス

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。

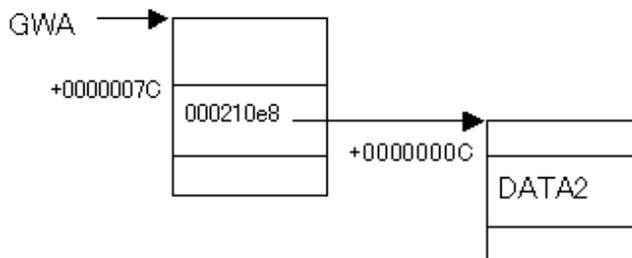
データ領域の参照方法は、オフセットが表示される場合と表示されない場合とで異なります。

- 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。

- 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)**

```

#### [12] エンコード方式

以下の項目に対してエンコードを表示します。

- ALPHANUM(英数字)
- AN-EDIT(英数字編集)

- NATIONAL(日本語)
- NAT-EDIT(日本語編集)

以下の値を表示します。

- SJIS : シフトJIS
- UTF8 : UTF-8
- UTF16LE : UTF-16 リトルエンディアン
- UTF16BE : UTF-16 ビッグエンディアン
- UTF32LE : UTF-32 リトルエンディアン
- UTF32BE : UTF-32 ビッグエンディアン

## B.6.2 プログラム制御情報リスト

### 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
.....	USESAVE	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
.....	SMBO	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
.....	BVOFI	0
.....	BVOII	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
.....	USESARE	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。

## B.6.3 セクションサイズリスト

### 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]
:	↓

[3] スタックサイズは、メソッド定義ごとに表示します。

## 付録C 環境変数情報

環境変数情報は、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です。

### C.1 環境変数情報の優先順位

環境変数情報の設定方法の詳細は、“5.3.2 実行環境情報の設定方法”を参照してください。

1. 環境変数の値
2. 実行用の初期化ファイルの値

環境変数情報の設定が重複した場合の優先順位は、2>1となります。

### C.2 環境変数情報一覧

#### 実行時オプションに関するもの

- “C.2.62 @GOPT(実行時オプションの指定)”
- “C.2.66 @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

- “C.2.3 @CBR\_CBRFILE(実行用の初期化ファイルの指定)”
- “C.2.4 @CBR\_CBRINFO(簡略化した動作状態を出力する指定)”
- “C.2.7 @CBR\_CODE\_SET(ファイルのコード系の指定)”
- “C.2.13 @CBR\_CONVERT\_CHARACTER(コード変換ライブラリの指定)”
- “C.2.26 @CBR\_ENTRYFILE(エントリ情報ファイルの指定)”
- “C.2.61 @ExitSessionMSG(COBOLアプリケーション起動中のWindows終了指定)”
- “C.2.63 @IconDLL(アイコンリソースのDLL名の指定)”
- “C.2.64 @IconName(アイコンリソースの識別名の指定)”
- “C.2.73 @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

- “C.2.47 @CBR\_SCREEN\_POSITION(スクリーン画面の表示位置の指定)”
- “C.2.46 @CBR\_SCR\_KEYDEFFILE(スクリーン操作のキー定義ファイルの指定)”
- “C.2.71 @ScrnFont(スクリーン操作で使用するフォントの指定)”
- “C.2.72 @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

- “C.2.12 @CBR\_CONSOLE(コンソールウィンドウの種別の指定)”
- “C.2.16 @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)”
- “C.2.17 @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)”
- “C.2.18 @CBR\_DISPLAY\_CONSOLE\_OUTPUT(DISPLAY UPON CONSOLEのイベントログ出力指定)”
- “C.2.19 @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)”

- “C.2.20 @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)”
- “C.2.21 @CBR\_DISPLAY\_SYSERR\_OUTPUT (DISPLAY UPON SYSERRのイベントログ出力指定)”
- “C.2.22 @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)”
- “C.2.23 @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)”
- “C.2.24 @CBR\_DISPLAY\_SYSOUT\_OUTPUT (DISPLAY UPON SYSOUTのイベントログ出力指定)”
- “C.2.35 @CBR\_JOBDATE (任意の日付を取得)”
- “C.2.57 @CnslBufLine (コンソールウィンドウのバッファ数の指定)”
- “C.2.58 @CnslFont (コンソールウィンドウのフォントの指定)”
- “C.2.59 @CnslWinSize (コンソールウィンドウの大きさの指定)”
- “C.2.77 @WinCloseMsg (ウィンドウを閉じるときのメッセージ表示の指定)”
- “C.2.8 @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)”
- “C.2.10 @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)”
- “C.2.11 @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)”
- “C.2.88 SYSINのアクセス名 (小入出力機能の入力ファイルの指定)”
- “C.2.89 SYSOUTのアクセス名 (小入出力機能の出力ファイルの指定)”

### メッセージに関するもの

- “C.2.37 @CBR\_MESSAGE (実行時メッセージの出力先の指定)”
- “C.2.38 @CBR\_MESS\_LEVEL\_CONSOLE (実行時メッセージの重大度指定)”
- “C.2.39 @CBR\_MESS\_LEVEL\_EVENTLOG (実行時メッセージの重大度指定)”
- “C.2.50 @CBR\_SYSERR\_EXTEND (SYSERR出力情報の拡張の指定)”
- “C.2.65 @MessOutFile (メッセージを出力するファイルの指定)”
- “C.2.67 @NoMessage (実行時メッセージおよびSYSERRの出力抑止指定)”
- “C.2.9 @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

- “C.2.1 @AllFileExclusive (ファイルの排他処理の指定)”
- “C.2.29 @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)”
- “C.2.30 @CBR\_FILE\_LFS\_ACCESS (COBOLファイルのサイズを拡張する指定)”
- “C.2.31 @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)”
- “C.2.32 @CBR\_FILE\_USE\_MESSAGE (入出力エラーの実行時メッセージの出力)”
- “C.2.27 @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)”
- “C.2.28 @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)”
- “C.2.56 @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)”
- “C.2.78 ファイル識別名 (プログラムで使用するファイルの指定)”

## 表示ファイルに関するもの

- “C.2.79 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の指定)”

## プリンタに関するもの

- “C.2.25 @CBR\_DocumentName\_xxxx (I制御レコードによる文書名の指定)”
- “C.2.41 @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)”
- “C.2.42 @CBR\_PrinterANK\_Size (ANK文字サイズの指定)”
- “C.2.43 @CBR\_PrintFontTable (印刷ファイルで使用するフォントテーブルの指定)”
- “C.2.44 @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)”
- “C.2.45 @CBR\_PrintTextPosition (文字配置座標の計算方法の指定)”
- “C.2.51 @CBR\_TextAlign (文字行内配置時の上端/下端合わせの指定)”
- “C.2.60 @DefaultFCB\_Name (デフォルトFCB名の指定)”
- “C.2.69 @PrinterFontName (印刷ファイルで使用するフォントの指定)”
- “C.2.70 @PRN\_FormName\_xxx (用紙名の指定)”
- “C.2.80 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)”
- “C.2.81 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)”
- “C.2.82 FCBxxxx (FCB制御文の指定)”
- “C.2.83 FOVLDIR (フォームオーバーレイパターンフォルダの指定)”
- “C.2.84 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)”
- “C.2.85 FOVLNAME (フォームオーバーレイパターンのファイル名の指定)”
- “C.2.86 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

- “C.2.2 @CBR\_ATTACH\_TOOL (プログラムからデバッグ機能を起動する指定)”
- “C.2.36 @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)”
- “C.2.40 @CBR\_MEMORY\_CHECK (メモリチェック機能を使って検査を行う指定)”
- “C.2.54 @CBR\_TRACE\_FILE (トレース情報の出力ファイルの指定)”
- “C.2.55 @CBR\_TRACE\_PROCESS\_MODE (TRACEファイルのプロセス毎の出力指定)”
- “C.2.76 @SVD\_PERFORM\_CHECK (デバッガでPERFORMレベルの監視を行う指定)”
- “C.2.75 @SVD\_ENV (デバッグ環境の指定)”
- “C.2.74 @SVD\_COMPATIBILITY (デバッガの互換性の指定)”
- “C.2.87 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

- “C.2.5 @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

- “C.2.68 @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

- “C.2.49 @CBR\_SYMFOWARE\_THREAD(Symfoware連携でマルチスレッド動作可能にする指定)”
- “C.2.52 @CBR\_THREAD\_MODE (スレッドモードの指定)”
- “C.2.53 @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)”
- “C.2.48 @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

- “C.2.6 @CBR\_ClassInfFile(クラス情報ファイルの指定)”
- “C.2.34 @CBR\_InstanceBlock(オブジェクトインスタンスの獲得方法の指定)”

## 組み関数に関するもの

- “C.2.33 @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

- “C.2.14 @CBR\_CSV\_OVERFLOW\_MESSAGE(CSV形式データ操作時のメッセージ抑止指定)”
- “C.2.15 @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。

## C.2.1 @AllFileExclusive(ファイルの排他処理の指定)

---

@AllFileExclusive =  $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$

---

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。

## C.2.2 @CBR\_ATTACH\_TOOL(プログラムからデバッグ機能を起動する指定)

---

@CBR\_ATTACH\_TOOL =  $\left\{ \begin{array}{l} \text{[接続先/JTEST [起動パラメタ]} \\ \text{SNAP 起動パラメタ} \\ \text{接続先/STUDIO [追加パスリスト]} \end{array} \right\}$

---

プログラムから以下のデバッグ機能を起動する場合に指定します。

- 対話型デバッグ
- 診断機能
- NetCOBOL Studioを使用したアタッチデバッグ

#### [接続先/TEST [起動パラメタ]

対話型デバッグを起動します。

デバッグの起動パラメタについては、“表24.1 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、リモートデバッグを使用します。リモートデバッグの起動パラメタについては、“表24.7 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ッ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。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。

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。

#### SNAP 起動パラメタ

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、“22.1.3.4 起動パラメタ”を参照してください。

#### 接続先/STUDIO 追加パスリスト

NetCOBOL Studioを使用してアタッチ形式のデバッグを行います。

接続先には、クライアント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。

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

— IPアドレスは、IPv4またはIPv6の形式で指定します。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。

— NetCOBOL Studioによるリモートデバッグを、ローカルPC上のアタッチ形式で行う場合は、localhostを指定します。

— 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。

追加パスリストには、起動するアプリケーションが動作し始めたときのカレントディレクトリと、起動するアプリケーションの格納ディレクトリは記述する必要はありません。以下の順序で、デバッグ情報のファイルが検索され、デバッグに利用されます。

1. 追加パスリストの指定順(各パスは、Windowsの場合は“;”で区切って記述してください。Solaris、Linuxの場合は“:”で区切って記述してください)
2. アプリケーションが動作し始めたときのカレントディレクトリ
3. 起動するアプリケーションの格納ディレクトリ

環境変数の設定後に、デバッグ対象のアプリケーションを実行すると、リモートデバッグとアプリケーションが接続され、デバッグが可能になります。詳細は、“NetCOBOL Studio ユーザーズガイド”の“アタッチデバッグ”を参照してください。



#### 例

#### 接続先の指定例

- 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
```

## デバッグのIPv6対応

デバッグ機能では、以下の範囲でIPv6アドレスに対応しています。

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。 <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	以下のIPv6 アドレスが使用できます。 <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> 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。 <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	以下のIPv6アドレス表記をサポートしています。 <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
	e. 英字部の小文字表記 f. 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(プライベート拡張)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。
- ・ 本環境変数は、デバッガなどでデバッグしている場合は無効です。

## C.2.3 @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]“5.5.3 実行用の初期化ファイルの検索順序”

## C.2.4 @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]“NetCOBOL メッセージ集”の“実行時メッセージ”

## C.2.5 @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。

[参照]“16.2 簡易アプリ間通信機能の使い方”

## C.2.6 @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、“17.3.3.4.1 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。

## C.2.7 @CBR\_CODE\_SET(ファイルのコード系の指定)

---

@CBR\_CODE\_SET =  $\left\{ \begin{array}{l} \text{ACP} \\ \text{SHIFT\_JIS} \\ \text{UTF-8} \end{array} \right\}$

---

ランタイムシステムが入出力するファイルの文字コード系を指定します。

- ・ 入力するファイル(注)
- ・ 出力するファイルを新規作成する

注) BOMの有無でコード系を識別します。

ファイルの文字コード系の省略値は、実行時コード系に応じて決まります。

- ・ 実行時コード系がANSIコードページの場合: ACP
- ・ 実行時コード系がシフトJISの場合: SHIFT\_JIS
- ・ 実行時コード系がUnicodeの場合: UTF-8

システムロケールが日本語の場合、ACPとSHIFT\_JISは同義です。

システムロケールが日本語以外の場合、SHIFT\_JISは指定しないでください。ファイルの文字コード系をUnicode以外にする場合は、ACPを指定してください。

本環境変数の指定は、以下のファイルに対して有効になります。

- ・ 小入出力ファイル
- ・ 実行時メッセージの出力ファイル
- ・ TRACE情報ファイル
- ・ COUNT情報ファイル
- ・ 汎用ログファイル

### 参照

- ・ “6.2.4 実行時コード系”
  - ・ “M.6 Windowsシステムにおけるシステムロケール”
-

## C.2.8 @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。



- 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。
- 

## C.2.9 @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。



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
- 

## C.2.10 @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。



- 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。

## C.2.11 @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。



- 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。

## C.2.12 @CBR\_CONSOLE (コンソールウィンドウの種別の指定)

@CBR\_CONSOLE =  $\left\{ \begin{array}{l} \text{SYSTEM} \\ \text{COBOL} \end{array} \right\}$

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。



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

## C.2.13 @CBR\_CONVERT\_CHARACTER (コード変換ライブラリの指定)

@CBR\_CONVERT\_CHARACTER =  $\left\{ \begin{array}{l} \text{ICONV} \\ \text{FJ\_ICONV} \\ \text{SYSTEM} \end{array} \right\}$

---

実行時に文字コード変換を行うときに使用するコード変換ライブラリを指定します。

本環境変数を省略した場合、実行環境で最初に動作したCOBOLプログラムの翻訳時に指定されたコード変換ライブラリが指定されたものとみなします。翻訳時のコード変換ライブラリの指定は、[翻訳オプションCONVCHAR](#)で指定します。なお、実行環境で最初に動作したCOBOLプログラムがV10.5以前のコンパイラで翻訳されている場合は、“ICONV”が指定されたものとみなします。

#### ICONV

NetCOBOL製品に組み込んでいるコード変換ライブラリを使用します。

#### FJ\_ICONV

運用環境にインストールされたInterstage Charset Managerのコード変換ライブラリを使用します。Interstage Charset Managerでは、利用者が変換定義をカスタマイズすることができます。

#### SYSTEM

システムAPIを使用します。エンコードUTF-32のコード変換はエラーになります。



#### 注意

プログラム資産転送時および翻訳時のコード変換ライブラリと同じコード変換ライブラリを指定してください。

- “[A.3.7 CONVCHAR\(コンパイラが使用するコード変換ライブラリ\)](#)”

---

## C.2.14 @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

---

## C.2.15 @CBR\_CSV\_TYPE (生成するCSV形式のバリエーション)

---

@CBR\_CSV\_TYPE =  $\left\{ \begin{array}{l} \text{MODE-1} \\ \text{MODE-2} \\ \text{MODE-3} \\ \text{MODE-4} \end{array} \right\}$

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。

## C.2.16 @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。

---

## C.2.17 @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。



### 注意

@CBR\_DISPLAY\_CONSOLE\_EVENTLOG\_SRCNAMEの指定がない場合、“NetCOBOL Application”となります。

---

## C.2.18 @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。

- 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。

## C.2.19 @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。

## C.2.20 @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。

## C.2.21 @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。

## C.2.22 @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。

## C.2.23 @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。

## C.2.24 @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。

- 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。

## C.2.25 @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]“8.1.9 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

## C.2.26 @CBR\_ENTRYFILE (エン트리情報ファイルの指定)

@CBR\_ENTRYFILE = エントリー情報ファイル名

---

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、“5.6 エントリ情報”または“17.3.2.4.4 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。

## C.2.27 @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]“7.9.3 外部ファイルハンドラ”

---



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。

---

## C.2.28 @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。



“7.9.3 外部ファイルハンドラ”

---

## C.2.29 @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。



“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。

### **C.2.30 @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方法”以下を参照してください。



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。

---

### **C.2.31 @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”

### C.2.32 @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。

### C.2.33 @CBR\_FUNCTION\_NATIONAL (NATIONAL関数の変換モードの指定)

@CBR\_FUNCTION\_NATIONAL = [ { MODE1  
MODE2 } ], { MODE3  
MODE4 } ]

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。

関数値がUnicodeの場合、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。



注意

本指定に誤りがある場合、省略したものとみなします。関数値がUnicode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C.1 関数値がSJISの場合

指定	変換方法
MODE1	「ー」(0xB0)を「-」(0x815C)に変換します。「`」(0x60)を「´」(0x8165)に変換します。
MODE2	「ー」(0xB0)を「一」(0x815B)に変換します。「`」(0x60)を「`」(0x814D)に変換します。

表C.2 関数値がUTF-16の場合

指定	変換方法
MODE1	「ー」(0xB0)を「-」(0x2015)に変換します。「`」(0x60)を「´」(0x2018)に変換します。
MODE2	「ー」(0xB0)を「一」(0x30FC)に変換します。「`」(0x60)を「`」(0xFF40)に変換します。

表C.3 関数値がUTF-32の場合

指定	変換方法
MODE1	「ー」(0xB0)を「-」(0x00002015)に変換します。「`」(0x60)を「´」(0x00002018)に変換します。
MODE2	「ー」(0xB0)を「一」(0x000030FC)に変換します。「`」(0x60)を「`」(0x0000FF40)に変換します。

MODE3とMODE4の違いを以下に示します。

表C.4 関数値がUTF-16の場合

指定	変換方法
MODE3	「-」(0x2D)を「ー」(0x2212)に変換します。「~」(0x7E)を「〜」(0x301C)に変換します。
MODE4	「-」(0x2D)を「-」(0xFF0D)に変換します。「~」(0x7E)を「~」(0xFF5E)に変換します。

表C.5 関数値がUTF-32の場合

指定	変換方法
MODE3	「-」(0x2D)を「-」(0x00002212)に変換します。「~」(0x7E)を「〜」(0x0000301C)に変換します。
MODE4	「-」(0x2D)を「-」(0x0000FF0D)に変換します。「~」(0x7E)を「~」(0x0000FF5E)に変換します。

### C.2.34 @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。

### C.2.35 @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。

### C.2.36 @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)

```
@CBR_JUSTINTIME_DEBUG = { [ { APLERR } ], { [接続先]/TEST [起動パラメタ] } }
                          { NO   { CBLERR } }, { SNAP [起動パラメタ] } }
                          { ALLERR }
```

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

調査手段にデバッグを使う。デバッグの起動パラメタについては、“表24.1 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、リモートデバッグを使用します。リモートデバッグの起動パラメタについては、“表24.7 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ッ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。

```
{ IPアドレス } [:ポート番号]
 { ホスト名 }
```

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が指定されたことと見なされます。

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_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、“[22.1.3.4 起動パラメタ](#)”を参照してください。



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、“[デバッガのIPv6対応](#)”を参照してください。

## C.2.37 @CBR\_MESSAGE (実行時メッセージの出力先の指定)

```
@CBR_MESSAGE = { CBLERROUT  
                  EVENTLOG[(コンピュータ名)]  
                  ALL[(コンピュータ名)] }
```

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力し、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

## C.2.38 @CBR\_MESS\_LEVEL\_CONSOLE(実行時メッセージの重大度指定)

```
@CBR_MESS_LEVEL_CONSOLE = { NO }  
                           { I  }  
                           { W  }  
                           { E  }  
                           { U  }
```

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。

## C.2.39 @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。

## C.2.40 @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、“[21.6 メモリチェック機能](#)”を参照してください。

## C.2.41 @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(VALID)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、“[8.1.12 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、“[8.1.9 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。

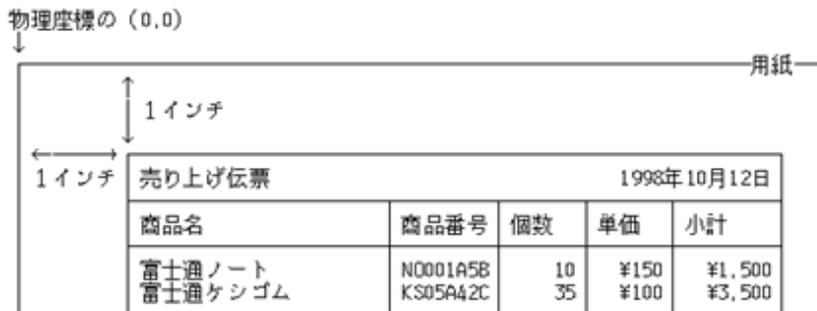
- 印刷原点位置を指定しない場合の印刷結果

論理座標の (0,0)

印刷不可能域	印刷不可能域		用紙		
	売上げ伝票		1998年10月12日		
	商品名	商品番号	個数	単価	小計
	富士通ノート	N0001A5B	10	¥150	¥1,500
	富士通ケシゴム	KS05A42C	35	¥100	¥3,500

- 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



## C.2.42 @CBR\_PrinterANK\_Size(ANK文字サイズの指定)

@CBR\_PrinterANK\_Size = { TYPE-M  
TYPE-PC  
TYPE-G }

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。

### C.2.43 @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、“[C.2.80 ファイル識別名\(プログラムで使用するプリンタ情報ファイルおよび各種パラメタの指定\)](#)”、“[C.2.81 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、“[8.1.3 印字文字](#)”の“印字文字の書体”および“印字文字のスタイル”、“[8.1.13 フォントテーブル](#)”を参照してください。

### C.2.44 @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、“[8.1.12 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
```

### C.2.45 @CBR\_PrintTextPosition(文字配置座標の計算方法の指定)

@CBR\_PrintTextPosition = { TYPE1 }  
{ TYPE2 }

---

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。



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。

---

### C.2.46 @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。

### C.2.47 @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。

## C.2.48 @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]“[11.1.6.5 ACCEPT文のファイル入力拡張機能](#)”

## C.2.49 @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]“[20.5.3 プリコンパイラの利用によるSymfoware連携](#)”

## C.2.50 @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。

## C.2.51 @CBR\_TextAlign(文字行内配置時の上端/下端合わせの指定)

---

-----  
@CBR\_TextAlign =  $\left\{ \begin{array}{l} \text{TOP} \\ \text{BOTTOM} \end{array} \right\}$   
-----

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。

## C.2.52 @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]“[20.3.3 プログラムの実行とスレッドモード](#)”

## C.2.53 @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]“1.6 スレッド同期制御サブルーチンの使い方”

## C.2.54 @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。



“21.4 TRACE機能”

---

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。

## C.2.55 @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
---

環境変数@CBR\_TRACE\_FILEを使用して、トレース情報の出力先を指定している場合のファイル名は、以下のようになります。

出力先ファイル名-プロセスID_日付(YYYYMMDD)_時間(HHMMSS).TRC or TRO
--

## C.2.56 @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 = { REMOVE }  
  { VALID }

---

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(VVALID)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。

---

## C.2.57 @CnslBufLine(コンソールウィンドウのバッファ数の指定)

---

$$\text{@CnslBufLine} = \left\{ \begin{array}{l} \text{バッファ行数} \\ \underline{100} \end{array} \right\}$$

---

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]“C.2.12 @CBR\_CONSOLE(コンソールウィンドウの種別の指定)”

---

## C.2.58 @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]“C.2.12 @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。

---

## C.2.59 @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]“C.2.12 @CBR\_CONSOLE(コンソールウィンドウの種別の指定)”

## C.2.60 @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)
```

## C.2.61 @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。

## C.2.62 @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”を参照してください。



例

@GOPT=r20 c20



参考

プログラムをコマンドから起動する場合、実行時オプションをコマンドの引数として指定できます。コマンドの引数に指定した実行時オプションと@GOPTの指定が重複する場合、コマンドの引数に指定した実行時オプションが有効になります。

## C.2.63 @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

“C.2.64 @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。

## C.2.64 @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]“C.2.63 @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。



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。

---

## C.2.65 @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。

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、“18.1.1 画面による入出力操作の抑止方法”を参照してください。



注意

- ・ 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。
- 

### Unicodeデータの扱い

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のコード系は以下になります。

- ・ 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。

## C.2.66 @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、“[K.2.1 プログラムの起動時にOSIV系システムのパラメタを渡す](#)”を参照してください。



#### 例

@MGPRM="ABCDE"

---



#### 参考

プログラムをコマンドから起動する場合、OSIV系形式の実行時パラメタをコマンドの引数として指定できます。コマンドの引数に指定した実行時パラメタと@MGPRMの指定が重複する場合、コマンドの引数に指定した実行時パラメタが有効になります。

---

## C.2.67 @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]“[C.2.77 @WinCloseMsg\(ウィンドウを閉じるときのメッセージ表示の指定\)](#)”

---

## C.2.68 @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接続(CONNECT文などで指定する)を確立するために使用されます。



#### 参照

“[15.2.8.1.2 ODBC情報ファイルの作成](#)”

---

## C.2.69 @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を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。

## C.2.70 @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。(Windows 10の場合)
  1. [コントロールパネル]>[デバイスとプリンターの表示]を選択し、[デバイスとプリンター]を表示させます。
  2. プリンターを選択して、画面右上の[プリントサーバープロパティ]をクリックします。  
→[プリントサーバーのプロパティ]ダイアログボックスが表示されます。
  3. [用紙]タブを選択し、[新しい用紙を作成する]にチェックを入れて、用紙を作成します。



### 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

環境変数情報名と用紙名の対応付け

@PRN\_FormName\_ABC=15x11

## C.2.71 @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。

## C.2.72 @ScrnSize(スクリーン操作の論理画面の大きさの指定)

@ScrnSize = { (桁数, 行数) }  
                  { (80,25) }

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。

## C.2.73 @ShowIcon(NetCOBOLのアイコン表示の抑止指定)

@ShowIcon = { YES }  
                  { NO }

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

- アイコン表示を抑制した場合、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、アイコン表示を抑制した場合、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。

## C.2.74 @SVD\_COMPATIBILITY (デバグの互換性の指定)

```
@SVD_COMPATIBILITY = { [COBOL97]バージョンレベル  
                      NetCOBOLバージョンレベル }
```

デバ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グの製品の種別とバージョンレベルを指定します。

デバグの製品種別とバージョンレベルは、デバグの[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。

## 参考

SolarisリモートデバグまたはLinuxリモートデバグの機能範囲を限定するための指定は、“[24.9.4.1 @SVD\\_COMPATIBILITY \(互換性の指定\)](#)”を参照してください。

## C.2.75 @SVD\_ENV (デバグ環境の指定)

$$\begin{array}{c}
 @SVD\_ENV = [ \left\{ \begin{array}{c} \text{DIFF} \\ \text{SAME}([ \left\{ \begin{array}{c} \text{ALL} \\ \text{WORD} \end{array} \right\} ]) \end{array} \right\} \parallel \left\{ \begin{array}{c} \text{BREAK} \\ \text{CONTINUE} \end{array} \right\} \\
 \\
 \left. \right\} [ \text{SIGNAL}([ \left\{ \begin{array}{c} \text{BREAK} \\ \text{CONTINUE} \end{array} \right\} ]) [ \text{THREADEND}([ \left\{ \begin{array}{c} \text{BREAK} \\ \text{CONTINUE} \end{array} \right\} ]) ]
 \end{array}$$

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、“[24.6.2 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、“[24.6.2 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。

## C.2.76 @SVD\_PERFORM\_CHECK(デバッガでPERFORMレベルの監視を行う指定)

---

-----

```
@SVD_PERFORM_CHECK = { YES }  
                      { NO  }
```

-----

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。

## C.2.77 @WinCloseMsg(ウィンドウを閉じるときのメッセージ表示の指定)

---

-----

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

-----

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。

## C.2.78 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。[参照]“7.9.2 PowerRDBconnector”

**EXFH**

外部ファイルハンドラ进行处理します。[参照]“7.9.3 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。

## C.2.79 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の指定)

-----

ファイル識別名 = [情報ファイル名]

-----

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。



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
```

環境変数情報

```
PRTFILE=C:¥WORK. PRC
```

## C.2.80 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、“[C.2.43 @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、“[8.1.3 印字文字](#)”の“印字文字の書体”および“印字文字のスタイル”、“[8.1.13 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
```

```
PRTFILE=C:¥WORK.PRC, , FONT (C:¥DEFAULT.FTB)
```

## C.2.81 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)

ファイル識別名 =  $\left\{ \begin{array}{l} \text{LPTn:} \\ \text{COMn:} \\ \text{PRTNAME:プリンタ名} \end{array} \right\} \quad [ , [ , \text{INF}(\text{xx}) ] [ , \text{FONT}(\text{yy}) ] ]$

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。

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、“8.1.12 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、“C.2.43 @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、“8.1.3 印字文字”の“印字文字の書体”および“印字文字のスタイル”、“8.1.13 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

---

## C.2.82 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、“[8.1.8 FCB](#)”を参照してください。

---

## C.2.83 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。

## C.2.84 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。

## C.2.85 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、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。

## C.2.86 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。

## C.2.87 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]“[21.5 COUNT機能](#)”

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。

## C.2.88 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出力先となるファイルの名前を指定します。[参照]“[A.3.56 SSIN \(ACCEPT文のデータの出力先\)](#)”

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。



### 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]“[11.1.6.5 ACCEPT文のファイル入力拡張機能](#)”

## C.2.89 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.3.57 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]“[11.1.6.4 DISPLAY文のファイル出力拡張機能](#)”

## 付録D 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D.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>
	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>	
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文またはCLOSE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。
	34	—	WRITE文またはCLOSE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同じでない。</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	—	<p>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。</p> <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。
	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。
	9E	MeFt	READ文による実行待ち状態が強制的に解除された。

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の通知コードを参照してください。“-”は詳細情報に0000が通知されます。

なお、MeFtの通知コードは、“NetCOBOLメッセージ集”の“MeFtのメッセージ”を参照してください。

## 付録E 定量制限

ここでは、COBOL処理系の定量制限を示します。なお、ここで示す制限は、論理的な値であり、プログラムが動作するための制限は、オペレーティングシステムおよび入出力管理システムに依存します。

### E.1 正書法

項目	値
可変形式の最大長(バイト数)	251
自由形式の最大長(バイト数)	251

### E.2 中核のデータ部

項目	値
データ項目の最大長(バイト数)	2147483647
英字項目、英数字項目の最大長(文字数)	2147483647
英数字編集項目の最大長(文字数)	2147483647
ブール項目の最大長(文字数)	2147483647
数字編集項目の最大長(文字数)	160
日本語項目の最大長(文字数)	1073741823(*1)
日本語編集項目の最大長(文字数)	1073741823(*1)
添字の最大値	2147483647
添字の次元数の最大値	7
OCCURS句の繰返し回数の最大値	2147483647
OCCURS句のKEY IS指定で指定するキー項目の長さの最大(バイト数)	256
1つの表要素に関するキーの総数の最大	2147483647
OCCURS句のINDEXED BY指定の指標名の個数の最大	60
暗黙のポインタ修飾の深さの最大	7

\*1 : エンコードUTF-32の場合、536870911となります。

### E.3 中核の手続き部

項目	値	
“ACCEPT 一意名-1 FROM 呼び名”の呼び名に機能名CONSOLEを対応付けた場合の一意名-1の大きさの最大(バイト数)	制限なし	
1つのINITIALIZE文の一意名の個数の最大	制限なし	
1つのINITIALIZE文の一意名で、添字付け、ポインタ付けまたは部分参照されたデータ項目または可変長項目の数	制限なし	
“INSPECT ~ CONVERTING 一意名-6 TO 一意名-7”の、一意名-6と一意名-7の大きさの最大(バイト数)	一意名-6と一意名-7が日本語/日本語編集項目以外の場合	256
	一意名-6と一意名-7が日本語/日本語編集項目の場合	制限なし
“PERFORM ~ 一意名-1/整数-1 TIMES ~”の一意名-1/整数-1の最大値	制限なし	

項目	値
“DISPLAY 一意名-1 UPON 呼び名”の呼び名に機能名 ENVIRONMENT-NAMEを対応付けた場合の一意名-1の大きさの最大(バイト数)	512
“DISPLAY 一意名-1 UPON 呼び名”の呼び名に機能名 ENVIRONMENT-VALUE を対応付けた場合の一意名-1の大きさの最大(バイト数)	2047
“ACCEPT 一意名-1 FROM 呼び名”の呼び名に機能名 ENVIRONMENT-VALUE を対応付けた場合の一意名-1の大きさの最大(バイト数)	2047

## E.4 順ファイル

項目	値		
ファイルの最大サイズ(バイト数)	COBOLファイルシステム	1G(*1)	
	COBOLファイルシステム(ファイルの高速処理)	システム制限まで	
	Btrieve(*2)	128G	
	PowerRDBconnector SQL Server(*3)	32T	
	PowerRDBconnector Oracle(*3)	無制限	
レコードの最大長(バイト数)	固定長レコード形式	COBOLファイルシステム	32,760
		Btrieve(*2)	32,760
		PowerRDBconnector SQL Server(*3)	8,060
		PowerRDBconnector Oracle(*3)	32,760
	可変長レコード形式	COBOLファイルシステム	32,760
		Btrieve(*2)	32,760
		PowerRDBconnector SQL Server(*3)	8,060
		PowerRDBconnector Oracle(*3)	32,760
レコードの最小長(バイト数)	COBOLファイルシステム	1	
	Btrieve(*2)	4	
	PowerRDBconnector SQL Server(*3)	1	
	PowerRDBconnector Oracle(*3)	1	

\*1 : 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。

\*2 : Btrieveについては、“Actian PSQL V11”の情報をもとにしています。

\*3 : PowerRDBconnectorについては、“PowerRDBconnector”の情報をもとにしています。

\*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。

## E.5 印刷ファイル

項目	値
レコードの最大長(バイト数)	32,760(*1)
レコードの最小長(バイト数)	1
LINAGE句で指定する論理ページの最大行数	制限なし

\*1 : エンコードUTF-32の帳票を使用する場合、65528となります。

## E.6 行順ファイル

項目		値
ファイルの最大サイズ(バイト数)	COBOLファイルシステム	1G(*1)
	COBOLファイルシステム(ファイルの高速処理)	システム制限まで
レコードの最大長(バイト数)		32760
レコードの最小長(バイト数)		0

\*1 : 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。

## E.7 相対ファイル

項目		値
ファイルの最大サイズ(バイト数)	COBOLファイルシステム	1G
レコードの最大長(バイト数)		32,760
レコードの最小長(バイト数)		1

## E.8 索引ファイル

項目		値	
ファイルの最大サイズ(バイト数)	COBOLファイルシステム	約1.7G(*1)	
	Btrieve(*2)	128G	
	PowerRDBconnector SQL Server(*3)	32T	
	PowerRDBconnector Oracle(*3)	無制限	
レコードの最大長(バイト数)	固定長レコード形式	COBOLファイルシステム	32,760
		Btrieve(*2)	32,760
		PowerRDBconnector SQL Server(*3)	8,060
		PowerRDBconnector Oracle(*3)	32,760
	可変長レコード形式	COBOLファイルシステム	32,760
		Btrieve(*2)	32,760
		PowerRDBconnector SQL Server(*3)	8,060
		PowerRDBconnector Oracle(*3)	32,760
レコードの最小長(バイト数)	COBOLファイルシステム	キーを構成する項目までの大きさ	
	Btrieve(*2)	キーを構成する項目までの大きさ	
	PowerRDBconnector SQL Server(*3)	キーを構成する項目までの大きさ	

項目	項目	値
	PowerRDBconnector Oracle(*3)	キーを構成する項目までの大きさ
RECORD KEY句に指定できるデータの最大個数	COBOLファイルシステム	254 (*5)
	Btrieve(*2)	119
	PowerRDBconnector SQL Server(*3)	16
	PowerRDBconnector Oracle(*3)	32
RECORD KEY句に指定できるデータ総長の最大(バイト数)	COBOLファイルシステム	254
	Btrieve(*2)	255
	PowerRDBconnector SQL Server(*3)	255
	PowerRDBconnector Oracle(*3)	データ・ブロックのサイズに依存
ALTERNATE RECORD KEY句に指定できるデータの最大個数	COBOLファイルシステム	254 (*5)
	Btrieve(*2)	119
	PowerRDBconnector SQL Server(*3)	16
	PowerRDBconnector Oracle(*3)	32
ALTERNATE RECORD KEY句に指定できるデータ総長の最大(バイト数)	COBOLファイルシステム	254
	Btrieve(*2)	255
	PowerRDBconnector SQL Server(*3)	255
	PowerRDBconnector Oracle(*3)	データ・ブロックのサイズに依存

\*1 : 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。

\*2 : Btrieveについては、“Actian PSQL V11”の情報をもとにしています。

\*3 : PowerRDBconnectorについては、“PowerRDBconnector”の情報をもとにしています。

\*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。

\*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。

## E.9 表示ファイル

項目	値
レコードの最大長(バイト数)	32767

## E.10 プログラム間連絡

項目	値
手続き部の見出しのパラメタの個数の最大	制限なし
CALL文のUSING 指定のパラメタの個数の最大	制限なし
ENTRY 文のUSING 指定のパラメタの個数の最大	制限なし
コマンド行文字の長さ(文字数)	260(引数は255)
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	指定された領域の先頭から255バイトまで(*1)

\*1 : 256バイト以降の文字列は無視されます。

## E.11 整列併合

項目	値
レコードの最大長(バイト数)	21484
SORT文およびMERGE 文で指定するキーデータ項目の個数の最大	64
SORT文およびMERGE 文で指定するキーデータ項目の長さの総和の最大(バイト数)	レコードの最大長
SORT文およびMERGE 文のキーデータ項目として指定する英字項目、英数字項目の最大長(文字数)	16382
SORT文およびMERGE 文のキーデータ項目として指定する日本語項目の最大長(文字数)	8191
SORT文およびMERGE 文のUSING で指定する入力ファイルの個数の最大	16

## E.12 原始文操作

項目	値
COPY文の原文語の長さの最大(文字数)	324
“COPY ~ JOINING 語-4”の語-4の長さの最大(文字数)	28
REPLACE 文の原文語の長さの最大	324

## E.13 オブジェクト指向プログラミング

項目	値
手続き部の見出しのパラメタの個数の最大	制限なし
INVOKE文のUSING 指定のパラメタの個数の最大	制限なし
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	指定された領域の先頭から255バイトまで(*1)

\*1 : 256バイト以降の文字列は無視されます

## E.14 ODBC情報ファイル

情報の種別	値	備考
ユーザIDの最大	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。
パスワードの最大	32バイト	
サーバ名の最大	32バイト	-
データソース名の最大	32バイト	-

## E.15 埋込みSQL文

以下は、実行時の埋込みSQL文の定量制限です。

項目	値
埋込みSQL文の最大長	16384バイト(*1)
クライアントとサーバ間で入出力できる領域長	制限なし(*2)
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	1024バイト(*3)

項目	値
CONNECT文により同時に接続できるコネクション数の最大	128個(*1)

\*1 : 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。

\*2 : 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ことがあります。

\*3 : 最大長を超えたメッセージ文字列は切り捨てられます。

## E.16 実行時パラメタ

---

項目	値
OSIV系形式の実行時パラメタの文字列の最大長(*1)	100バイト(*1)

\*1 : 実行環境変数情報@MGPRMで指定します。

## 付録F 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。

### F.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

### F.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]
```

## F.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10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10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.
```

## F.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]
```

## F.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。

## F.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]
```

## F.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。

## F.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.3.41 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



“A.3.63 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。



“A.3.63 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。



“A.3.60 TEST (デバッグ機能の使用の可否)”

- 広域最適化を行った場合、NetCOBOL Studioのデバッグ機能は使用できません。

## 付録G 特殊な定数および利用者語の書き方

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、各種定数および利用者語の書き方を説明します。

### G.1 プログラム名定数

プログラム名段落(PROGRAM-ID)、ENTRY文、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があります。詳細は、“G.6 外部名の規則”を参照してください。また、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。

### G.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

### G.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:][パス名]ファイル名 [.拡張子] } "  
" { ポート名 : } "  
-----

### 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

## G.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りません。また、使用できる文字に制限があります。詳細は“[G.6 外部名の規則](#)”を参照してください。

## G.5 外部名を指定するための利用者語

以下の利用者語は、AS指定を省略すると、外部名の一部として使用されます。

- プログラム名
- クラス名
- メソッド名
- プロパティ名

これらの利用者語は、使用できる文字に制限があります。詳細は“[G.6 外部名の規則](#)”を参照してください。

## G.6 外部名の規則

---

本システムでは、外部名としてシフトJISで表現できる文字しか使用できません。

また、以下の名前を使用することはできません。

- “WinMain”
- “main”
- “JMP”で始まる名前
- “COBW3\_”で始まる名前
- “COB\_”で始まる名前
- “COBOL\_”で始まる名前
- “NetCOBOL ユーザーズガイド”の“[付録I サブルーチン](#)”に記載されている名前

## 付録H 関数

NetCOBOLが提供する以下の関数を説明します。

- [H.1 組込み関数](#)
- [H.2 索引ファイル復旧関数](#)
- [H.3 COBOLファイルユーティリティ関数](#)
- [H.4 COBOLファイルアクセスルーチン](#)

### H.1 組込み関数

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。

#### H.1.1 組込み関数一覧

NetCOBOLで使用できる組込み関数の一覧を以下に示します。

表H.1 組込み関数一覧

分類	関数	用途	関数の型(注)
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
	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

分類	関数	用途	関数の型(注)
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
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

注: 関数の型が英数字または日本語のエンコードの値は、翻訳オプションRCSまたはENCODEに従います。

## H.1.2 組込み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、“[H.1.1 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-INTEG (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。

## H.1.3 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。

## H.1.4 組み込み関数の便利な使い方

- [CURRENT-DATE関数](#)を利用した西暦の取得
- [任意の基準日からの通日計算](#)
- [RANDOM関数](#)を利用した疑似乱数列の生成

## 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、“[11.1.8 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。

## 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。
```

## 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。

## H.2 索引ファイル復旧関数

### H.2.1 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つ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。

## 注意

データの状態によっては、ファイルに出力されない場合があります。

### 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H.2 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;
}
```

## H.2.2 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H.2 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;
}
```

## H.2.3 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、“[5.6 エントリ情報](#)”を参照してください。

```
[ENTRY]
CFURCOV=F3BIFUTC.DLL
CFURCOVS=F3BIFUTC.DLL
```

## H.2.4 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 GYOUMU-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 GYOUMU-KAISHI
000450 WHEN OTHER
000460 GO TO GYOUMU-TEISHI
000470 END-EVALUATE
000490 WHEN OTHER
000500 GO TO GYOUMU-TEISHI
000510 END-EVALUATE.
000520 GYOUMU-KAISHI.
000530 OPEN I-O IXDFILE.
:
000580 CLOSE IXDFILE.
000590 GYOUMU-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 GYOUMU-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 GYOUMU-KAISHI
000430         WHEN OTHER
000440             GO TO GYOUMU-TEISHI
000450         END-EVALUATE
000470     WHEN OTHER
000480         GO TO GYOUMU-TEISHI
000490     END-EVALUATE.
000500 GYOUMU-KAISHI.
000510     OPEN I-O IXDFILE.
000515     :
000560     CLOSE IXDFILE.
000570 GYOUMU-TEISHI.
000580     EXIT PROGRAM.

```

## H.2.5 メッセージの内容とコード

“表H.2 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H.2 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)

コード10進	メッセージの内容
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。

## H.3 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	COB_FILE_CONVERT
ロード	COB_FILE_LOAD
アンロード	COB_FILE_UNLOAD
整列	COB_FILE_SORT
再編成	COB_FILE_REORGANIZE
コピー	COB_FILE_COPY
移動	COB_FILE_MOVE
削除	COB_FILE_DELETE

### 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。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.
```



- 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、“5.6 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
```

## H.3.1 ファイル変換関数 (COB\_FILE\_CONVERT)

### 説明

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。

### 呼出し形式

```
CALL "COB_FILE_CONVERT" USING BY REFERENCE COBF-INF
```

### パラメタの説明

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、そのほかのデータ項目を16進形式とします。
- 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/UTF-32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。

また、テキストファイルの文字コードの扱いを以下に示します。

- ASCII/シフトJISコード系で動作する場合、ASCII/シフトJISコード体系として扱います。
- Unicode(UCS-2)コード系で動作する場合、UCS-2リトルエンディアンコード体系として扱います。
- Unicode(UTF-32)コード系で動作する場合、UTF-32リトルエンディアンコード体系として扱います。

### 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。

値	意味
"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 (UCS-2/ UTF-32)
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-TYPE-ALLUTF32	"A"	レコード中の全データをUTF-32形式とみなします。	—	—	○
COBF-CONVERT-TYPE-UTF32	"B"	指定長だけUTF-32形式とみなします。	—	—	○

## COBF-CONVERT-LEN

データ形式の長さ(UCS-2形式/UTF-32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
- COBF-CONVERT-TYPE-UTF32

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
- COBF-CONVERT-TYPE-ALLUTF32



### 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

特殊レジスタPROGRAM-STATUSを使用して受け取ります。

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。 エラーの詳細は、COBF-MESSAGEに文字列で設定されます。



## 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.

```

## H.3.2 ファイルロード関数 (COB\_FILE\_LOAD)

### 説明

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。

### 呼出し形式

```
CALL "COB_FILE_LOAD" USING BY REFERENCE COBF-INF.
```

### パラメタの説明

#### 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。

文字	意味
"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。

文字	意味
"F"	固定長
"V"	可変長

#### COBF-OUTPUT-RECLEN

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指定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。

文字	意味
"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形式またはUTF-32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。

文字	意味
"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。

文字	意味
"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	UCS-2形式のPIC Nの項目
COBF-EXT-KEY-TYPE-PICN32	3	UTF-32形式のPIC Nの項目

## 復帰コード

特殊レジスタPROGRAM-STATUSを使用して受け取ります。

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。 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

- 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 "1" 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.

```

### H.3.3 ファイルアンロード関数 (COB\_FILE\_UNLOAD)

#### 説明

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。

#### 呼出し形式

```
CALL "COB_FILE_UNLOAD" USING BY REFERENCE COBF-INF.
```

#### パラメタの説明

##### 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。

文字	意味
"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。

文字	意味
"F"	固定長
"V"	可変長

#### COBF-INPUT-RECLEN

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

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。 エラーの詳細は、COBF-MESSAGEに文字列で設定されます。



#### 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.
```

## H.3.4 ファイル整列関数 (COB\_FILE\_SORT)

#### 説明

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。

#### 呼出し形式

```
CALL "COB_FILE_SORT" USING BY REFERENCE COBF-INF.
```

#### パラメタの説明

##### 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。

文字	意味
"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。

文字	意味
"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。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() (注1)
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



```
MOVE "D" TO COBF-SORT-KEYSEQ (2).  
CALL "COB_FILE_SORT" USING BY REFERENCE COBF-INF.
```

### H.3.5 ファイル再編成関数 (COB\_FILE\_REORGANIZE)

#### 説明

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。

#### 呼出し形式

```
CALL "COB_FILE_REORGANIZE" USING BY REFERENCE COBF-INF.
```

#### パラメタの説明

##### COBF-INPUT-FILENAME

再編成を行う索引ファイルのパス名を指定します。

##### COBF-OUTPUT-FILENAME

再編成後の内容を出力する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。

#### 復帰コード

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。 エラーの詳細は、COBF-MESSAGEに文字列で設定されます。



#### 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.
```

### H.3.6 ファイルコピー関数 (COB\_FILE\_COPY)

#### 説明

ファイルを複写します。

#### 呼出し形式

```
CALL "COB_FILE_COPY" USING BY REFERENCE COBF-INF.
```

#### パラメタの説明

##### 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

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。 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

### 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.
```

## H.3.7 ファイル移動関数 (COB\_FILE\_MOVE)

### 説明

ファイルを移動します。

### 呼出し形式

```
CALL "COB_FILE_MOVE" USING BY REFERENCE COBF-INF.
```

### パラメタの説明

#### 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

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。 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

C:¥INFILEをF:¥OUTFILEに移動する

移動元ファイル名 : C:¥INFILE  
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.
```

## H.3.8 ファイル削除関数 (COB\_FILE\_DELETE)

### 説明

ファイルを削除します。

### 呼出し形式

```
CALL "COB_FILE_DELETE" USING BY REFERENCE COBF-INF.
```

### パラメタの説明

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

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。 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

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.
```

## H.4 COBOLファイルアクセスルーチン

### H.4.1 COBOLファイルアクセスルーチンとは

COBOLファイルアクセスルーチンは、COBOLファイルを操作するためのC言語用のAPI(Application Program Interface)関数群です。これらの関数は、COBOLランタイムシステムを呼び出すことによって、ファイルの操作を行います。

COBOLファイルアクセスルーチンを使用することにより、以下の操作が実現できます。

- COBOLアプリケーションで作成したファイルの読み込み/書換えなどの既存資産への入出力
- COBOLで扱う以下の編成のファイルの創成
  - 行順ファイル
  - レコード順ファイル
  - 相対ファイル
  - 索引ファイル
- COBOLアプリケーションとのファイル/レコードの排他/共用
- 既存の索引ファイルのファイル属性/レコードキー構成の解析

## H.4.2 API関数一覧

表H.3 入出力

関数名	機能の概要	関数の使用範囲 (注)
cobfa_close()	ファイルをクローズする	—
cobfa_delcurr()	順読みしたレコードを削除する	S
cobfa_delkey()	主レコードキーの値によって示されるレコードを削除する	RD
cobfa_delrec()	相対レコード番号によって示されるレコードを削除する	RD
cobfa_open()	ファイルをオープンする	—
cobfa_openW()	ファイルをオープンする	—
cobfa_release()	指定したファイルのレコードロックをすべて解除する	—
cobfa_rewcurr()	順読みしたレコードを書き換える	SD
cobfa_rewkey()	主レコードキーの値によって示されるレコードを書き換える	RD
cobfa_rewrec()	相対レコード番号によって示されるレコードを書き換える	RD
cobfa_rdkey()	任意のレコードキーの値によって示されるレコードを読み込む	RD
cobfa_rdnex()	レコードを順に読み込む	S
cobfa_rdnex()	相対レコード番号によって示されるレコードを読み込む	RD
cobfa_stkey()	任意のレコードキーの値によって示されるレコードに位置付ける	SD
cobfa_strec()	相対レコード番号によって示されるレコードに位置付ける	SD
cobfa_wrkey()	指定した主レコードキーの値を持つレコードを書き出す	RD
cobfa_wrnex()	レコードを順に書き出す	S
cobfa_wrnex()	指定した相対レコード番号を持つレコードを書き出す	RD

注: 関数の使用範囲を記号で示しています。各記号の意味は以下のとおりです。

- RD : オープンモードが乱呼出し、または動的呼出しで使用できる関数
- SD : オープンモードが順呼出し、または動的呼出しで使用できる関数
- S : オープンモードが順呼出しのときにだけ使用できる関数

表H.4 ファイルの情報を取得

関数名	機能の概要
cobfa_indexinfo()	索引ファイルが持つファイルの属性、またはレコードキーの構成を取得する

表H.5 入出力の状況を取得

関数名	機能の概要
cobfa_errno()	エラー番号を取得する
cobfa_reclen()	読み込んだレコードの長さを取得する
cobfa_recnun()	位置付けられたレコードの相対レコード番号を取得する
cobfa_stat()	入出力状態を取得する

表H.6 マルチスレッド環境下での排他制御

関数名	機能の概要
LOCK_cobfa()	他のスレッドのCOBOLファイルへのアクセスに対して排他ロックをかける
UNLOCK_cobfa()	他のスレッドのCOBOLファイルへのアクセスに対して排他ロックを解除する

### H.4.3 API関数で使用する構造体

いくつかのAPI関数は、以下に示す構造体型へのポインタ型を用います。

構造体名	機能の概要
struct fa_dictinfo	索引ファイルの属性を取得するための構造体
struct fa_keydesc	任意のレコードキーを指定するための構造体 指定した任意のレコードキーの構造を取得するためにも使用する
struct fa_keylist	すべてのレコードキーを指定するための構造体

### H.4.4 準備するもの

開発言語として32ビット対応Cコンパイラを用意します。Cコンパイラのインストールと、その動作環境の設定を行っておいてください。

### H.4.5 環境設定

NetCOBOLをインストールすると、COBOLファイルアクセスルーチンはNetCOBOLと同じフォルダに格納されます。

NetCOBOLをインストールした後、環境変数PATHと環境変数LIBにNetCOBOLのインストールフォルダが設定されていることを確認してください。設定されていない場合は、これらの環境変数にNetCOBOLのインストールフォルダを追加してください。

環境変数の設定は、“[1.2.1 環境変数の設定](#)”を参照してください。

### H.4.6 使い方

ここでは、Cソースプログラムの作成、翻訳、オブジェクトファイルのリンク、プログラムの実行について説明します。

#### Cソースプログラムの作成

COBOLファイルアクセスルーチンを用いたCソースプログラムをテキストエディタなどで作成します。当アクセスルーチンを使う上での注意事項については、“[H.4.39 COBOLファイルアクセスルーチンの留意事項](#)”を参照してください。

Cソースプログラムには以下の記述を入れ、ヘッダファイルをインクルードすることを明示します。

```
#include "f3bifcfa.h"
```

#### Cソースプログラムの翻訳

Cソースプログラムを翻訳します。

Cコンパイラに、インクルードファイルを検索するパスを指定する翻訳オプションを指定してください。この翻訳オプションに、NetCOBOLをインストールしたフォルダを指定します。

以下に、Visual C++での翻訳オプションを示します。

- `/I directory`  
(`directory`には、NetCOBOLをインストールしたフォルダを指定します)



例

`/I C:¥COBOL`

## オブジェクトファイルのリンク

オブジェクトファイルをリンクし、実行可能プログラムを作成します。

オブジェクトファイルをリンクするときは、NetCOBOLをインストールしたフォルダの中にある以下のファイルを結合します。

- F3BIFCFA.lib

## 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実行します。

このとき、特に考慮すべきことはありません。プログラムは、COBOLアプリケーションとファイル/レコードを排他/共用できます。

## H.4.7 ファイルのオープン (cobfa\_open())

```
long cobfa_open (
    const char      *fname, /* ファイル名 */
    long            openflgs, /* オープン属性 */
    const struct fa_keylist *keylist, /* レコードキーリスト */
    long            reclen /* レコード長 */
);
```

### 説明

ファイルをオープンします。

### パラメタの説明

#### ファイル名(fname)

ファイル名文字列。

#### ダミーファイル

COBOLアプリケーションのダミーファイルと同等の機能を実現させたい場合、ファイル名の末尾に“DUMMY”を付加するか、またはファイル名として“DUMMY”を指定します。  
詳細は、“[ダミーファイル](#)”を参照してください。

#### 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のアクセス性能は、使用する機能を限定することで高速化することができます。ファイルの高速処理を使用する場合、ファイル名の末尾に“BSAM”を付加します。

詳細は、“[ファイルの高速処理](#)”を参照してください。

#### オープン属性(openflgs)

指定値にはa.からi.までの9つのカテゴリがあり、これらをビットの論理和で結合して指定します。

詳細は、“[オープン属性の指定値](#)”を参照してください。

#### レコードキーリスト(keylist)

ファイル編成が索引ファイル(FA\_IDXFILE)である場合にだけ意味を持ちます。keylistはオープンするファイルの主レコードキー、副レコードキーの構成として有効になります。struct fa\_keylist型については“[H.4.33 fa\\_keylist構造体](#)”を参照してください。

索引ファイルのオープンで、keylistにNULLポインタを指定した場合、当関数は既存のファイルの索引構成とレコード形式、レコード長を認識してオープンします。このとき、レコード形式の指定とレコード長の指定は無効になります。

## レコード長(reclen)

レコード形式が固定長形式(FA\_FIXLEN)の場合、レコード長reclenを固定レコード長として扱います。

レコード形式が可変長形式(FA\_VARLEN)の場合、レコード長reclenを最大レコード長として扱います。

レコード長はFA\_NRECSIZE(32760)を超えてはいけません。

## 実行可能な条件

当関数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	○
	レコード順ファイル	○
	相対ファイル	○
	索引ファイル	○
オープンモード	INPUTモード	○
	OUTPUTモード	○
	I-Oモード	○
	EXTENDモード	○
呼出し法	順呼出し	○
	乱呼出し	○
	動的呼出し	○

○:当関数の実行が可能です。

ー:当関数の実行はできません。

## 復帰値

当関数の復帰値は、下表のようになります。

値	状態	意味
1以上	成功	当関数の実行が成功しました。入出力状態が、状況を示すコードを保持していることがあります。 この復帰値は、オープンに成功したファイルのファイルディスクリプタの値です。ただし、ファイルディスクリプタはファイルのオープン時にOSが返却したファイルハンドルの値ではありませんので、注意してください。
-1	失敗	当関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

## 発生するステータス

当関数の呼出しによって発生するステータスは、下表のようになります。

### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。
FA_ENOERR	5	不定ファイルが存在しなかったため、仮想的にファイルをオープンしました。または、オープンモードがINPUTモード(FA_INPUT)以外である場合、ファイルを新規に創成しました。

### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_EBADACC	90	指定したファイル編成、呼出し法、オープンモードの組合せでは実行することができません。
FA_EFNAME	35	ファイルが存在しません。
FA_EFLOCKED	93	ファイルはすでに排他でオープンされています。
FA_EFNAME	90	ファイル名が正しくありませんでした。または、ファイルへのアクセスが失敗しました。
FA_EFNAME	91	ファイル名を指定していません。
FA_EBADFLAG	39	指定したファイル編成やレコード形式などの属性と、既存ファイルの構成が異なります。
FA_EBADKEY	39	指定したレコードキーの情報と、既存の索引ファイルのキーの構成が異なります。
FA_EBADKEY	90	指定したレコードキーの情報が正しくありません。
FA_EBADLENG	39	指定したレコード長と、既存ファイルのレコード長が異なります。
FA_EBADFILE	90	指定した索引ファイルは、内部情報が破壊しています。または、ファイル編成が索引ファイルではありません。

注:これらは代表的なステータスです。これ以外のステータスについては、“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

## オープン属性の指定値

カテゴリb.からi.までは省略可能です。(\*は省略値)

### a. オープンモード

	記号定数	意味	対応するCOBOL構文
	FA_INPUT	INPUTモード	OPEN INPUT
	FA_OUTPUT	OUTPUTモード	OPEN OUTPUT
	FA_INOUT	I-Oモード	OPEN I-O
	FA_EXTEND	EXTENDモード	OPEN EXTEND

ファイル編成が行順ファイル(FA\_LSEQFILE)の場合、オープンモードにI-Oモード(FA\_INOUT)を指定することはできません。指定した場合はエラーになります。

### b. ファイル編成

	記号定数	意味	対応するCOBOL構文
*	FA_SEQFILE	レコード順ファイル	ORGANIZATION IS SEQUENTIAL
	FA_LSEQFILE	行順ファイル	ORGANIZATION IS LINE SEQUENTIAL
	FA_RELFILE	相対ファイル	ORGANIZATION IS RELATIVE
	FA_IDXFILE	索引ファイル	ORGANIZATION IS INDEXED

### c. レコード形式

	記号定数	意味	対応するCOBOL構文
*	FA_FIXLEN	固定長形式	RECORD CONTAINS integer CHARACTERS
	FA_VARLEN	可変長形式	RECORD IS VARYING IN SIZE

d. 呼出し法

	記号定数	意味	対応するCOBOL構文
*	FA_SEQACC	順呼出し	ACCESS MODE IS SEQUENTIAL
	FA_RNDACC	乱呼出し	ACCESS MODE IS RANDOM
	FA_DYNACC	動的呼出し	ACCESS MODE IS DYNAMIC

ファイル編成が行順ファイル(FA\_LSEQFILE)またはレコード順ファイル(FA\_SEQFILE)の場合、呼出し法に順呼出し(FA\_SEQACC)以外を指定することはできません。指定した場合はエラーになります。

e. ロックモード

	記号定数	意味	対応するCOBOL構文
	FA_AUTOLOCK	自動ロック	LOCK MODE IS AUTOMATIC
	FA_MANULOCK	手動ロック	LOCK MODE IS MANUAL
	FA_EXCLLOCK	排他ロック	LOCK MODE IS EXCLUSIVE または OPEN WITH LOCK

- ー オープンモードがOUTPUTモード(FA\_OUTPUT)の場合、ロックモードに排他ロック(FA\_EXCLLOCK)を指定したものと扱います。
- ー オープンモードがINPUTモード(FA\_INPUT)の場合、ロックモードに自動ロック(FA\_AUTOLOCK)または手動ロック(FA\_MANULOCK)を指定することはできません。指定した場合は無効になります。
- ー オープンモードがINPUTモード(FA\_INPUT)の場合、ロックモードのデフォルト値として、共用モードでファイルをオープンします。読み込み時は、レコードロックの指定が無効になります。
- ー オープンモードがINPUTモード(FA\_INPUT)以外の場合、ロックモードのデフォルト値は排他ロック(FA\_EXCLLOCK)になります。
- ー ファイル編成が行順ファイル(FA\_LSEQFILE)またはレコード順ファイル(FA\_SEQFILE)の場合、ロックモードに手動ロック(FA\_MANULOCK)を指定することはできません。指定した場合はエラーになります。

 参考

各API関数の実行とファイルの排他制御の関係について説明します。

ファイルの排他制御

- 排他モード

ファイルを排他モードで開くと、他の利用者はそのファイルにアクセスすることができません。

- 共用モード

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、ファイルのオープンは無効となります。

オープン属性の組合せ(a.オープンモードとe.ロックモード)によるファイルのモードの状態を、下表に示します。

		オープンモード			
		FA_INPUT	FA_OUTPUT	FA_INOUT	FA_EXTEND
ロックモード	指定無し	共用	排他	排他	排他
	FA_AUTOLOCK			共用	共用
	FA_MANULOCK				
	FA_EXCLLOCK	排他		排他	排他

## レコードの排他制御

オープンモードがI-Oモード(FA\_INOUT)の共用モードで開かれたファイルのレコードは、読み込み時のレコードロック指定により排他状態となります。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。読み込み時のレコードロック指定については、“[H.4.10レコードの読み込み\(cobfa\\_rdkey\(\)\)](#)”、“[H.4.11レコードの読み込み\(cobfa\\_rdnex\(\)\)](#)”または“[H.4.12レコードの読み込み\(cobfa\\_rdnex\(\)\)](#)”を参照してください。

また、次の場合、レコードの排他状態が解除されます。

- レコードロックの解除(cobfa\_release)
- ファイルのクローズ(cobfa\_close)

ロックモードが自動ロック(FA\_AUTOLOCK)の場合、以下の処理を実行した場合も同様となります。

- レコードの読み込み(cobfa\_rdkey/cobfa\_rdnex/cobfa\_rdnex())
- レコードの書き出し(cobfa\_wrkey/cobfa\_wrnex/cobfa\_wrnex())
- レコードの削除(cobfa\_delcurr/cobfa\_delkey/cobfa\_delkey())
- レコードの書換え(cobfa\_rewcurr/cobfa\_rewkey/cobfa\_rewkey())
- レコードの位置決め(cobfa\_stkey/cobfa\_stkey())

### f. 不定ファイル

	記号定数	意味	対応するCOBOL構文
*	FA_NOTOPT	不定ファイルでない	SELECT filename
	FA_OPTIONAL	不定ファイル	SELECT OPTIONAL filename

### g. 動作コード系

	記号定数	意味
*	FA_ASCII	ファイルはSJISまたはJIS8でエンコードした文字データを持つ
	FA_UCS2	ファイルはUCS-2でエンコードした文字データを持つ (注)
	FA_UTF8	ファイルはUTF-8でエンコードした文字データを持つ
	FA_UCS2BE	ファイルはUCS-2(ビッグエンディアン)でエンコードした文字データを持つ
	FA_UCS2LE	ファイルはUCS-2(リトルエンディアン)でエンコードした文字データを持つ
	FA_UTF32	ファイルはUTF-32でエンコードした文字データを持つ (注)
	FA_UTF32BE	ファイルはUTF-32(ビッグエンディアン)でエンコードした文字データを持つ
	FA_UTF32LE	ファイルはUTF-32(リトルエンディアン)でエンコードした文字データを持つ

注:エンディアンはシステムに依存し、リトルエンディアンとなります。

- 動作コード系は、行順ファイルを持つ文字データのエンコード種別を指定するものです。行順ファイルは、ファイルの構造がエンコード種別により異なるため、この指定が必要になります。
- 動作コード系は、ファイル編成が行順ファイル(FA\_LSEQFILE)のときだけ指定することができます。その他のファイル編成は、ファイルの構造がエンコード種別に依存しないため、動作コード系を指定する必要はありません。
- 動作モードがUnicodeのCOBOLアプリケーションで扱う行順ファイルで、レコードのデータ項目が日本語項目の場合にはFA\_UCS2、FA\_UCS2BE、FA\_UCS2LE、FA\_UTF32、FA\_UTF32BE、またはFA\_UTF32LEを、レコードのデータ項目が日本語項目以外の場合にはFA\_UTF8を指定します。
- 動作モードがUnicodeでないCOBOLアプリケーションで扱う行順ファイルにはFA\_ASCIIを指定します。

### h. キーパートフラグ使用指定

	記号定数	意味
	FA_USEKPFLAGS	struct fa_keypart型の構造体のkp_flagsメンバの指定値を有効とする

- キーパートフラグ使用指定は、レコードキーリストkeylistが包含するstruct fa\_keypart型の構造体のメンバkp\_flagsの指定値を有効とするときに設定します。struct fa\_keypart型については、“[H.4.32 fa\\_keydesc構造体](#)”を参照してください。
- キーパートフラグ使用指定は、ファイル編成が索引ファイル(FA\_IDXFILE)のときだけ指定することができます。
- 当指定は、動作モードがUnicodeのCOBOLアプリケーションで扱う索引ファイルを使用する場合に必要です。
- 当指定の省略時は、メンバkp\_flagsの指定値を無視します。

i. BSAM指定(高速処理およびファイルの最大サイズ拡張)

	記号定数	意味
	FA_BSAM	BSAM(ファイルの高速処理) 指定で操作する

- 「ファイルの高速処理」で作成されたCOBOLファイルを参照する場合や、「ファイルの高速処理」で参照されるファイルを作成する場合に指定します。  
ファイル処理を高速化させる場合、「ファイルの高速処理」を使用してください。  
詳細は、“[ファイルの高速処理](#)”を参照してください。
- ファイル編成がレコード順ファイル(FA\_SEQFILE)または、行順ファイル (FA\_LSEQFILE)の場合にのみ、利用できます。
- FA\_BSAMを指定することにより、「ファイルの高速処理」と同じファイル最大サイズの拡張、制限、注意事項が発生します。詳細は、以下を参照してください。  
- “[7.7.4 ファイルの高速処理](#)”
- FA\_BSAM指定と「ファイルの高速処理」を同時に指定した場合、「ファイルの高速処理」が有効となります。

## H.4.8 ファイルのオープン (cobfa\_openW())

```
long cobfa_openW (
    const short      *fname, /* ファイル名 */
    long             openflgs, /* オープン属性 */
    const struct fa_keylist *keylist, /* レコードキーリスト */
    long             reclen /* レコード長 */
);
```

### 説明

Unicode(UTF-16)のファイル名を指定して、ファイルをオープンします。

ファイル名fnameが指すファイルを、オープン属性openflgs、レコード長reclen、レコードキーリストkeylistの情報をもとにオープンします。  
指定するファイル名fnameのコード系を除き、本関数の機能や動作はcobfa\_open()と同等です。

### パラメタの説明

#### ファイル名(fname)

ファイル名文字列。Unicode(UTF-16)で指定します。

#### オープン属性(openflgs)、レコード長(reclen)、レコードキーリスト(keylist)

“[H.4.7 ファイルのオープン \(cobfa\\_open\(\)\)](#)”と同じです。

## H.4.9 ファイルのクローズ (cobfa\_close())

```
long cobfa_close (
    long fd /* ファイルディスクリプタ */
);
```

### 説明

ファイルディスクリプタfdが指すファイルをクローズします。

## パラメタの説明

### ファイルディスクリプタ(fd)

クローズするファイルのファイルディスクリプタを指定します。

### 実行可能な条件

当関数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	○
	レコード順ファイル	○
	相対ファイル	○
	索引ファイル	○
オープンモード	INPUTモード	○
	OUTPUTモード	○
	I-Oモード	○
	EXTENDモード	○
呼出し法	順呼出し	○
	乱呼出し	○
	動的呼出し	○

○:当関数の実行が可能です。

—:当関数の実行はできません。

### 復帰値

当関数の復帰値は、下表のようになります。

値	状態	説明
0	成功	当関数の実行が成功しました。
-1	失敗	当関数の実行が失敗しました。

### 発生するステータス

当関数の呼出しによって発生するステータスは、下表のようになります。

#### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

#### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_ENOTOPEN	42	不正なファイルディスクリプタを指定しています。

注:これらは代表的なステータスです。これ以外のステータスについては、“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

## H.4.10 レコードの読み込み (cobfa\_rdkey())

```
long cobfa_rdkey (  
    long          fd,          /* ファイルディスクリプタ */  
    long          readflgs,   /* 読み込み属性           */  
    char          *recarea,   /* レコード域             */  
)
```

```

const struct fa_keydesc *keydesc, /* レコードキー構成指定 */
long keynum /* レコードキー番号指定 */
);

```

## 説明

任意のレコードキーの値で示すレコードを読み込みます。(乱読み)

ファイルディスクリプタfdが示すファイルにおいて、レコード域recareaの任意のレコードキーの値でレコードを指定します。指定されたレコードを読み込み、レコード域recareaに格納します。

## パラメタの説明

### ファイルディスクリプタ(fd)

レコードを読み込むファイルのファイルディスクリプタを指定します。

### 読み込み属性(readflgs)

指定値には2つのカテゴリがあり、これらをビットの論理和で結合して指定します。これらは省略可能です。

詳細は、“[読み込み属性の指定値](#)”を参照してください。

### レコード域(recarea)

読み込んだレコードが格納されます。

### レコードキー構成指定(keydesc)

任意のレコードキーを指定します。struct fa\_keydesc型については、“[H.4.32 fa\\_keydesc構造体](#)”を参照してください。

NULLを指定した場合、任意のレコードキーの指定としてレコードキー番号指定keynumが有効になります。

### レコードキー番号指定(keynum)

レコードキー構成指定にNULLを指定した場合、有効になります。

主レコードキーを指定するには、1を指定します。副レコードキーを指定するには、レコードキー番号指定に2以上の値を指定します。この値は索引ファイルを創成したときの副レコードキーを宣言したときの並びの順番に対応しています。最初の副レコードキーなら2を、2番目の副レコードキーなら3を、それ以降もこれらと同様に指定します。

## 実行可能な条件

当該数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	—
	相対ファイル	—
	索引ファイル	○
オープンモード	INPUTモード	○
	OUTPUTモード	—
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	—
	乱呼出し	○
	動的呼出し	○

○:当該数の実行が可能です。

—:当該数の実行はできません。

## 復帰値

当該数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当関数の実行が成功しました。入出力状態が、状況を示すコードを保持していることがあります。
-1	失敗	当関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

#### 発生するステータス

当関数の呼出しによって発生するステータスは、下表のようになります。

#### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。
FA_ENOERR	2	読み込んだレコードの参照キーの値が、次に続くレコードの参照キーの値と同じです。

#### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_ENOTOPEN	47	指定したファイルは、INPUTモード以外、かつ、I-Oモード以外でオープンしてあります。または、不正なファイルディスクリプタを指定しています。
FA_EBADACC	90	当関数を実行することができないファイル編成または、呼出し法でファイルがオープンしてあります。
FA_EBADFLAG	90	当関数を実行することができない読み込みモードを指定しています。または、その他のフラグの指定が正しくありません。
FA_ENOREC	23	任意のレコードキーの値が示すレコードが存在しません。
FA_EBADKEY	90	指定したレコードキー構成またはレコードキー番号は存在しません。または、正しくありません。
FA_ELOCKED	99	主レコードキーの値で指定したレコードはロックされています。

注:これらは代表的なステータスです。これ以外のステータスについては、“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

#### 読み込み属性の指定値

これらは省略可能です。(\*は省略値)

##### a. 読み込みモード

	記号定数	意味
*	FA_EQUAL	指定したレコードキーの値に該当するレコードを読み込む

##### b. レコードロックフラグ

	記号定数	意味	対応するCOBOL構文
	FA_LOCK	ロックありで読み込む	READ WITH LOCK
	FA_NOLOCK	ロックなしで読み込む	READ WITH NO LOCK

レコードロックフラグのデフォルト値は、オープン時のロックモードの指定により異なります。ロックモードが自動ロック(FA\_AUTOLOCK)である場合はデフォルト値がロックあり(FA\_LOCK)となり、それ以外の場合のデフォルト値はロックなし(FA\_NOLOCK)となります。

## H.4.11 レコードの読み込み (cobfa\_rdnnext())

```
long cobfa_rdnnext (
    long fd,          /* ファイルディスクリプタ */
    long readflgs,   /* 読み込み属性          */
    char *recarea    /* レコード域            */
);
```

### 説明

レコードを順に読み込みます。(順読み)

ファイルディスクリプタfdが示すファイルにおいて、位置付けられているレコードの次(または前)のレコードを読み込み、レコード域recareaに格納します。

### パラメタの説明

#### ファイルディスクリプタ(fd)

レコードを読み込むファイルのファイルディスクリプタを指定します。

#### 読み込み属性(readflgs)

指定値には2つのカテゴリがあり、これらをビットの論理和で結合して指定します。これらは省略可能です。

詳細は、“[読み込み属性の指定値](#)”を参照してください。

#### レコード域(recarea)

読み込んだレコードが格納されます。

### 実行可能な条件

当関数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	○
	レコード順ファイル	○
	相対ファイル	○
	索引ファイル	○
オープンモード	INPUTモード	○
	OUTPUTモード	—
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	○
	乱呼出し	—
	動的呼出し	○

○: 当関数の実行が可能です。ただし、ファイルの位置付けが不定でないことが 必須条件となります。

—: 当関数の実行はできません。

### 復帰値

当関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当関数の実行が成功しました。入出力状態が、状況を示すコードを保持していることがあります。
-1	失敗	当関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

## 発生するステータス

当該関数の呼出しによって発生するステータスは、下表のようになります。

### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。
FA_ENOERR	2	読み込んだレコードの参照キーの値が、次に続くレコードの参照キーの値と同じです。

### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_ENOTOPEN	47	指定したファイルは、INPUTモード以外、かつ、I-Oモード以外でオープンしてあります。または、不正なファイルディスクリプタを指定しています。
FA_EBADACC	90	当該関数を実行することができない呼出し法で、ファイルがオープンしてあります。
FA_EENDFILE	10	ファイル終了条件が発生しました。
FA_EBADFLAG	90	当該関数を実行することができない読み込みモードを指定しています。または、その他のフラグの指定が正しくありません。
FA_ENOCURR	46	レコードへの位置付けが不定でした。
FA_ELOCKED	99	順読み込みによって位置付けようとしたレコードはすでにロックされています。

注:これらは代表的なステータスです。これ以外のステータスについては、“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

## 読み込み属性の指定値

これらは省略可能です。(\*は省略値)

### a. 読み込みモード

	記号定数	意味	対応するCOBOL構文
*	FA_NEXT	次のレコードを読み込む	READ NEXT RECORD
	FA_PREV	前のレコードを読み込む	READ PREVIOUS RECORD

ファイル編成が、行順ファイル(FA\_LSEQFILE)またはレコード順ファイル(FA\_SEQFILE)の場合、読み込みモードに前のレコード(FA\_PREV)を指定することはできません。

### b. レコードロックフラグ

	記号定数	意味	対応するCOBOL構文
	FA_LOCK	ロックありで読み込む	READ WITH LOCK
	FA_NOLOCK	ロックなしで読み込む	READ WITH NO LOCK

レコードロックフラグのデフォルト値は、オープン時のロックモードの指定により異なります。ロックモードが自動ロック(FA\_AUTOLOCK)の場合は、デフォルト値がロックあり(FA\_LOCK)となり、それ以外の場合のデフォルト値はロックなし(FA\_NOLOCK)となります。

## H.4.12 レコードの読み込み (cobfa\_rdrec())

```
long cobfa_rdrec (
    long          fd,          /* ファイルディスクリプタ */
```

```

long      readflgs, /* 読み属性          */
char      *recarea, /* レコード域        */
unsigned long recnum /* 相対レコード番号  */
);

```

## 説明

相対レコード番号が指すレコードを読み込みます。(乱読み)

ファイルディスクリプタfdが示すファイルにおいて、相対レコード番号recnumが指すレコードを読み込み、レコード域recareaに格納します。

## パラメタの説明

### ファイルディスクリプタ(fd)

レコードを読み込むファイルのファイルディスクリプタを指定します。

### 読み属性(readflgs)

指定値には2つのカテゴリがあり、これらをビットの論理和で結合して指定します。これらは省略可能です。

詳細は、“[読み属性の指定値](#)”を参照してください。

### レコード域(recarea)

読み込んだレコードが格納されます。

### 相対レコード番号(recnum)

読み込むレコードの相対レコード番号を指定します。

## 実行可能な条件

当該数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	—
	相対ファイル	○
	索引ファイル	—
オープンモード	INPUTモード	○
	OUTPUTモード	—
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	—
	乱呼出し	○
	動的呼出し	○

○:当該数の実行が可能です。

—:当該数の実行はできません。

## 復帰値

当該数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当該数の実行が成功しました。
-1	失敗	当該数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

## 発生するステータス

当関数の呼出しによって発生するステータスは、下表のようになります。

### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_ENOTOPEN	47	指定したファイルは、INPUTモード以外、かつ、I-Oモード以外でオープンしてあります。または、不正なファイルディスクリプタを指定しています。
FA_EBADACC	90	当関数を実行することができないファイル編成または呼出し法で、ファイルがオープンしてあります。
FA_EBADFLAG	90	当関数を実行することができない読み込みモードを指定しています。または、その他のフラグの指定が正しくありません。
FA_ENOREC	23	相対レコード番号が示すレコードが存在しません。
FA_ELOCKED	99	相対レコード番号で指定したレコードはロックされています。

注:これらは代表的なステータスです。これ以外のステータスについては、“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

## 読み込み属性の指定値

これらは省略可能です。(\*は省略値)

### a. 読み込みモード

	記号定数	意味
*	FA_EQUAL	指定した相対レコード番号のレコードを読み込む

### b. レコードロックフラグ

	記号定数	意味	対応するCOBOL構文
	FA_LOCK	ロックありで読み込む	READ WITH LOCK
	FA_NOLOCK	ロックなしで読み込む	READ WITH NO LOCK

レコードロックフラグのデフォルト値は、オープン時のロックモードの指定により異なります。ロックモードが自動ロック(FA\_AUTOLOCK)の場合は、デフォルト値がロックあり(FA\_LOCK)となり、それ以外の場合のデフォルト値はロックなし(FA\_NOLOCK)となります。

## H.4.13 レコードの書出し (cobfa\_wrkey())

```
long cobfa_wrkey (  
    long    fd,          /* ファイルディスクリプタ */  
    const char *recarea, /* レコード域             */  
    long    reclen      /* 書出しレコード長       */  
);
```

### 説明

主レコードキーの値で指定したレコードを書き出します。(乱書出し)

ファイルディスクリプタfdが示すファイルにおいて、レコード域recareaが持つ主レコードキーの値が指すレコードを、レコード域が持つ内容で書き出します。

## パラメタの説明

### ファイルディスクリプタ(fd)

レコードを書き出すファイルのファイルディスクリプタを指定します。

### レコード域(recarea)

書き出すデータを格納します。

### 書出しレコード長(reclen)

書出すレコードのレコード長を指定します。ファイルが可変長形式であるときだけ有効になります。

## 実行可能な条件

当関数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	—
	相対ファイル	—
	索引ファイル	○
オープンモード	INPUTモード	—
	OUTPUTモード	○
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	—
	乱呼出し	○
	動的呼出し	○

○:当関数の実行が可能です。

—:当関数の実行はできません。

## 復帰値

当関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当関数の実行が成功しました。
-1	失敗	当関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

## 発生するステータス

当関数の呼出しによって発生するステータスは、下表のようになります。

### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_EDUPL	22	書出ししよ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ます。しかし、主レコードキーまたは副レコードキーは重複を許可していません。
FA_EBADLENG	44	指定した書出しレコード長の値が指定可能な範囲を超えています。

エラー番号	入出力状態	説明
FA_ENOTOPEN	48	指定したファイルは、OUTPUTモード以外、かつ、I-Oモード以外でオープンしてあります。または、不正なファイルディスクリプタを指定しています。

注:これらは代表的なステータスです。これ以外のステータスについては“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

## H.4.14 レコードの書出し (cobfa\_wrnext())

```
long cobfa_wrnext (
    long      fd,          /* ファイルディスクリプタ */
    const char *recarea,  /* レコード域              */
    long      reclen     /* 書出しレコード長        */
);
```

### 説明

レコードを順に書き出します。(順書出し)

ファイルディスクリプタfdが示すファイルにおいて、レコード域recareaが持つ内容で、レコードを順に書き出します。

### パラメタの説明

#### ファイルディスクリプタ(fd)

レコードを書き出すファイルのファイルディスクリプタを指定します。

#### レコード域(recarea)

書き出すデータを格納します。

#### 書出しレコード長(reclen)

書出すレコードのレコード長を指定します。ファイルが可変長形式であるときだけ有効になります。

### 実行可能な条件

当該数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	○
	レコード順ファイル	○
	相対ファイル	○
	索引ファイル	○
オープンモード	INPUTモード	—
	OUTPUTモード	○
	I-Oモード	—
	EXTENDモード	○
呼出し法	順呼出し	○
	乱呼出し	—
	動的呼出し	—

○:当該数の実行が可能です。

—:当該数の実行はできません。

### 復帰値

当該数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当該数の実行が成功しました。
-1	失敗	当該数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

#### 発生するステータス

当該数の呼出しによって発生するステータスは、下表のようになります。

#### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

#### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_EDUPL	22	書き出そ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ます。しかし、主レコードキーまたは副レコードキーは重複を許可していません。
FA_EBADLENG	44	指定した書出しレコード長の値が指定可能な範囲を超えています。
FA_ENOTOPEN	48	指定したファイルは、OUTPUTモード以外、かつ、EXTENDモード以外でオープンしてあります。または、不正なファイルディスクリプタを指定しています。

注:これらは代表的なステータスです。これ以外のステータスについては、“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

## H.4.15 レコードの書出し (cobfa\_wrrec())

```
long cobfa_wrrec (
    long      fd,          /* ファイルディスクリプタ */
    const char *recarea,  /* レコード域              */
    long      reclen,     /* 書出しレコード長        */
    unsigned long recnum  /* 相対レコード番号        */
);
```

#### 説明

相対レコード番号で指定したレコードを書き出します。(乱書出し)

ファイルディスクリプタfdが示すファイルで、相対レコード番号recnumが指すレコードを、レコード域recareaが持つ内容で書き出します。

#### パラメタの説明

##### ファイルディスクリプタ(fd)

レコードを書き出すファイルのファイルディスクリプタを指定します。

##### レコード域(recarea)

書き出すデータを格納します。

##### 書出しレコード長(reclen)

書出すレコードのレコード長を指定します。ファイルが可変長形式であるときだけ有効になります。

##### 相対レコード番号(recnum)

相対レコード番号を指定します。

#### 実行可能な条件

当該数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	—
	相対ファイル	○
	索引ファイル	—
オープンモード	INPUTモード	—
	OUTPUTモード	○
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	—
	乱呼出し	○
	動的呼出し	○

○: 当該関数の実行が可能です。  
 —: 当該関数の実行はできません。

#### 復帰値

当該関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当該関数の実行が成功しました。
-1	失敗	当該関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

#### 発生するステータス

当該関数の呼出しによって発生するステータスは、下表のようになります。

##### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

##### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_EDUPL	22	すでに存在する相対レコード番号を使って書出しを行おうとしました。
FA_EBADLENG	44	指定した書出しレコード長の値が指定可能な範囲を超えています。
FA_ENOTOPEN	48	指定したファイルは、OUTPUTモード以外、かつ、I-Oモード以外でオープンしてあります。または、不正なファイルディスクリプタを指定しています。

注: これらは代表的なステータスです。これ以外のステータスについては、“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

## H.4.16 レコードの削除 (cobfa\_delcurr())

```
long cobfa_delcurr (
    long fd /* ファイルディスクリプタ */
);
```

#### 説明

順読み込みしたレコードを削除します。(順削除)

ファイルディスクリプタfdが示すファイルにおいて、順読込みによって位置付けられているレコード(カレントレコード)を削除します。

#### パラメタの説明

##### ファイルディスクリプタ(fd)

レコードを削除するファイルのファイルディスクリプタを指定します。

#### 実行可能な条件

当関数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	—
	相対ファイル	○
	索引ファイル	○
オープンモード	INPUTモード	—
	OUTPUTモード	—
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	○
	乱呼出し	—
	動的呼出し	—

○:当関数の実行が可能です。ただし、レコードが順読込みによって位置付けられていることが必須条件となります。  
—:当関数の実行はできません。

#### 復帰値

当関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当関数の実行が成功しました。
-1	失敗	当関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

#### 発生するステータス

当関数の呼出しによって発生するステータスは、下表のようになります。

##### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

##### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_ENOTOPEN	49	指定したファイルは、I-Oモード以外でオープンしてあります。または、不正なファイルディスクリプタを指定しています。
FA_EBADACC	90	当関数を実行することができないファイル編成または、呼出し法でファイルがオープンしてあります。
FA_ELCKED	99	順読込みによって位置付けられているレコードはロックされています。
FA_ENOCURR	43	指定したファイルでの順読込みが成功していませんでした。

注:これらは代表的なステータスです。これ以外のステータスについては、“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

## H.4.17 レコードの削除 (cobfa\_delkey())

```
long cobfa_delkey (  
    long      fd,      /* ファイルディスクリプタ */  
    const char *recarea /* レコード域          */  
);
```

### 説明

主レコードキーの値が示すレコードを削除します。(乱削除)

ファイルディスクリプタfdが示すファイルにおいて、レコード域recareaの内容が持つ主レコードキーの値が指すレコードを削除します。

### パラメタの説明

#### ファイルディスクリプタ(fd)

レコードを削除するファイルのファイルディスクリプタを指定します。

#### レコード域(recarea)

主レコードキーを含みます。

### 実行可能な条件

当該数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	—
	相対ファイル	—
	索引ファイル	○
オープンモード	INPUTモード	—
	OUTPUTモード	—
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	—
	乱呼出し	○
	動的呼出し	○

○:当該数の実行が可能です。

—:当該数の実行はできません。

### 復帰値

当該数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当該数の実行が成功しました。
-1	失敗	当該数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

### 発生するステータス

当該数の呼出しによって発生するステータスは、下表のようになります。

#### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

#### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_ENOTOPEN	49	指定したファイルは、I-Oモード以外でオープンしてあります。または、不正なファイルディスクリプタを指定しています。
FA_EBADACC	90	当関数を実行することができないファイル編成または、呼出し法でファイルがオープンしてあります。
FA_ELOCKED	99	主レコードキーの値で指定したレコードはロックされています。
FA_ENOREC	23	主レコードキーの値で指定したレコードは存在しません。

注:これらは代表的なステータスです。これ以外のステータスについては、“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

## H.4.18 レコードの削除 (cobfa\_delrec())

```
long cobfa_delrec (
    long          fd,      /* ファイルディスクリプタ */
    unsigned long recnum /* 相対レコード番号      */
);
```

#### 説明

相対レコード番号が指すレコードを削除します。(乱削除)

ファイルディスクリプタfdが示すファイルにおいて、相対レコード番号recnumが指すレコードを削除します。

#### パラメタの説明

##### ファイルディスクリプタ(fd)

レコードを削除するファイルのファイルディスクリプタを指定します。

##### 相対レコード番号(recnum)

削除するレコードの相対レコード番号を指定します。

#### 実行可能な条件

当関数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	—
	相対ファイル	○
	索引ファイル	—
オープンモード	INPUTモード	—
	OUTPUTモード	—
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	—
	乱呼出し	○
	動的呼出し	○

- : 当該関数の実行が可能です。
- －: 当該関数の実行はできません。

#### 復帰値

当該関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当該関数の実行が成功しました。
-1	失敗	当該関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

#### 発生するステータス

当該関数の呼出しによって発生するステータスは、下表のようになります。

##### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

##### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_ENOTOPEN	49	指定したファイルは、I-Oモード以外でオープンしてあります。または、不正なファイルディスクリプタを指定しています。
FA_EBADACC	90	当該関数を実行することができないファイル編成または、呼出し法でファイルがオープンしてあります。
FA_ELOCKED	99	相対レコード番号で指定したレコードはロックされています。
FA_ENOREC	23	相対レコード番号で指定したレコードは存在しません。

注: これらは代表的なステータスです。これ以外のステータスについては、“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

## H.4.19 レコードの書換え (cobfa\_rewcurr())

```
long cobfa_rewcurr (
    long      fd,          /* ファイルディスクリプタ */
    const char *recarea,  /* レコード域              */
    long      reclen     /* 書換えレコード長        */
);
```

### 説明

順読込みしたレコードを書き換えます。(順更新)

ファイルディスクリプタfdが示すファイルにおいて、順読込みによって位置付けられているレコード(カレントレコード)を、レコード域recareaの内容で書き換えます。

#### パラメタの説明

##### ファイルディスクリプタ(fd)

レコードを書き換えるファイルのファイルディスクリプタを指定します。

##### レコード域(recarea)

レコードの内容を格納します。

##### 書換えレコード長(reclen)

書換えレコードのレコード長を指定します。ファイルが可変長形式であるときだけ有効になります。ファイル編成がレコード順ファイル(FA\_SEQFILE)のときは元のレコード長を変えることはできません。

## 実行可能な条件

当関数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	○
	相対ファイル	○
	索引ファイル	○
オープンモード	INPUTモード	—
	OUTPUTモード	—
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	○
	乱呼出し	—
	動的呼出し	—

○: 当関数の実行が可能です。ただし、レコードが順読込みによって位置付けられていることが必須条件となります。  
—: 当関数の実行はできません。

## 復帰値

当関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当関数の実行が成功しました。
-1	失敗	当関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

## 発生するステータス

当関数の呼出しによって発生するステータスは、下表のようになります。

### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_EDUPL	22	書き換えよ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ます。しかし、主レコードキーまたは副レコードキーは重複を許可していません。
FA_EBADLENG	44	指定した書換えレコード長の値が指定可能な範囲を超えています。

注: この表にあるステータス以外に発生するものは、“[H.4.16レコードの削除 \(cobfa\\_delcurr\(\)\)](#)”の発生するステータスを参照してください。

## H.4.20 レコードの書換え (cobfa\_rewkey())

```
long cobfa_rewkey (  
    long      fd,          /* ファイルディスクリプタ */  
    const char *recarea,  /* レコード域              */
```

```
long    reclen /* 書換えレコード長 */
);
```

## 説明

主レコードキーの値が指すレコードを書き換えます。(乱更新)

ファイルディスクリプタfdが示すファイルにおいて、レコード域recareaの中に持つ主レコードキーの値が指すレコードを、レコード域recareaの内容で書き換えます。

## パラメタの説明

### ファイルディスクリプタ(fd)

レコードを書き換えるファイルのファイルディスクリプタを指定します。

### レコード域(recarea)

レコードの内容を格納します。主レコードキーを含みます。

### 書換えレコード長(reclen)

書換えレコードのレコード長を指定します。ファイルが可変長形式であるときだけ有効になります。

## 実行可能な条件

当関数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	—
	相対ファイル	—
	索引ファイル	○
オープンモード	INPUTモード	—
	OUTPUTモード	—
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	—
	乱呼出し	○
	動的呼出し	○

○:当関数の実行が可能です。

—:当関数の実行はできません。

## 復帰値

当関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当関数の実行が成功しました。
-1	失敗	当関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

## 発生するステータス

当関数の呼出しによって発生するステータスは、下表のようになります。

### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_EDUPL	22	重複を許可していないレコードキーに対して、すでに存在するレコードキーの値を持つレコード内容で書き換えようとした。
FA_EBADLENG	44	指定した書換えレコード長の値が指定可能な範囲を超えています。

注:この表にあるステータス以外に発生するものは、“H.4.17 レコードの削除 (cobfa\_delkey())”の発生するステータスを参照してください。

## H.4.21 レコードの書換え (cobfa\_rewrec())

```
long cobfa_rewrec (
    long      fd,          /* ファイルディスクリプタ */
    const char *recrea,   /* レコード域              */
    long      reclen,     /* 書換えレコード長        */
    unsigned long recnum  /* 相対レコード番号        */
);
```

**説明**

相対レコード番号が指すレコードを書き換えます。(乱更新)

ファイルディスクリプタfdが示すファイルにおいて、相対レコード番号recnumのレコードを、レコード域recreaの内容で書き換えます。

**パラメタの説明**

**ファイルディスクリプタ(fd)**

レコードを書き換えるファイルのファイルディスクリプタを指定します。

**レコード域(recrea)**

レコードの内容を格納します。

**書換えレコード長(reclen)**

書換えレコードのレコード長を指定します。ファイルが可変長形式であるときだけ有効になります。

**相対レコード番号(recnum)**

相対レコード番号を指定します。

**実行可能な条件**

当該関数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	—
	相対ファイル	○
	索引ファイル	—
オープンモード	INPUTモード	—
	OUTPUTモード	—
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	—
	乱呼出し	○
	動的呼出し	○

- : 当関数の実行が可能です。
- －: 当関数の実行はできません。

#### 復帰値

当関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当関数の実行が成功しました。
-1	失敗	当関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

#### 発生するステータス

当関数の呼出しによって発生するステータスは、下表のようになります。

##### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

##### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_EBADLENG	44	指定した書換えレコード長の値が指定可能な範囲を超えています。

注: この表にあるステータス以外に発生するものは、“[H.4.18 レコードの削除 \(cobfa\\_delrec\(\)\)](#)”の発生するステータスを参照してください。

## H.4.22 レコードの位置決め (cobfa\_stkey())

```
long cobfa_stkey (
    long          fd,          /* ファイルディスクリプタ */
    long          stflgs,     /* 位置付け属性           */
    const char    *recarea,   /* レコード域             */
    const struct fa_keydesc *keydesc, /* レコードキー構成指定 */
    long          keynum,     /* レコードキー番号指定   */
    long          keyleng     /* 有効キー長             */
);
```

### 説明

任意のレコードキーの値が示すレコードに位置付けます。

ファイルディスクリプタfdが示すファイルにおいて、レコード域recareaの中に持つ任意のレコードキーの値に関連するレコードに位置付けを行います。また、指定した任意のレコードキーを、以降の順読込みでの参照キーとして宣言します。

#### パラメタの説明

##### ファイルディスクリプタ(fd)

レコードを位置付けるファイルのファイルディスクリプタを指定します。

##### 位置付け属性(stflgs)

指定値には2つのカテゴリがあり、これらをビットの論理和で結合して指定します。これらは省略可能です。(\*は省略値)

##### a. 位置付けモード

	記号定数	意味	対応するCOBOL構文
	FA_FIRST	先頭レコード	START FIRST RECORD
*	FA_EQUAL	レコードキーの値と等しいレコード	START KEY IS =

	記号定数	意味	対応するCOBOL構文
	FA_GREAT	レコードキーの値を超えるレコード	START KEY IS >
	FA_GTEQ	レコードキーの値以上のレコード	START KEY IS >=
	FA_LESS	レコードキーの値より小さいレコード	START KEY IS <
	FA_LTEQ	レコードキーの値以下のレコード	START KEY IS <=

b. 逆順読み込みフラグ

	記号定数	意味	対応するCOBOL構文
*	なし	順読み込み時に論理的に順方向に読み込む	—
	FA_REVORD	順読み込み時に論理的に逆方向に読み込む	START WITH REVERSED ORDER

- 逆順読み込みフラグのデフォルト値は、正順読み込みになります。
- 位置付けモードがレコードキーの値を超過(FA\_GREAT)またはレコードキーの値以上(FA\_GTEQ)の場合、逆順読み込みフラグ(FA\_REVORD)を指定することはできません。
- 逆順読み込みフラグの指定は、乱読み込み、位置付け、ファイルのクローズを行ったり、ファイル終了条件が発生したりすることで無効になります。

レコード域(recarea)

読み込んだレコードが格納されます。

レコードキー構成指定(keydesc)

任意のレコードキーを指定します。struct fa\_keydesc型については、“[H.4.32 fa\\_keydesc構造体](#)”を参照してください。

NULLを指定した場合、任意のレコードキーの指定としてレコードキー番号指定keynumが有効になります。

レコードキー番号指定(keynum)

レコードキー構成指定にNULLを指定した場合は、有効になります。

主レコードキーを指定するには、1を指定します。副レコードキーを指定するには、レコードキー番号指定に2以上の値を指定します。この値は索引ファイルを創成したときの副レコードキーを宣言したときの並びの順番に対応しています。最初の副レコードキーなら2を、2番目の副レコードキーなら3を、それ以降もこれらと同様に指定します。

有効キー長(keyleng)

有効な参照キーの長さを短くするために使います。有効な参照キーの長さを短くしない場合は、0を指定します。この場合、任意のレコードキーの全体が参照キーとなります。有効な参照キーの長さを短くする場合は、1以上の値を指定します。この値は、キーパートの並びを連続した先頭からのキーの長さをバイト単位で指定します。

実行可能な条件

関数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	—
	相対ファイル	—
	索引ファイル	○
オープンモード	INPUTモード	○
	OUTPUTモード	—
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	○

オープン属性	種別	実行の可否
	乱呼出し	—
	動的呼出し	○

○:当関数の実行が可能です。  
—:当関数の実行はできません。

#### 復帰値

当関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当関数の実行が成功しました。
-1	失敗	当関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

#### 発生するステータス

当関数の呼出しによって発生するステータスは、下表のようになります。

##### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

##### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_ENOTOPEN	47	指定したファイルは、INPUTモード以外、かつ、I-Oモード以外でオープンしてあります。または、不正なファイルディスクリプタを指定しています。
FA_EBADACC	90	当関数を実行することができないファイル編成または、呼出し法でファイルがオープンしてあります。
FA_EBADFLAG	90	当関数を実行することができない位置付けモードを指定しています。または、その他のフラグの指定が正しくありません。
FA_ENOREC	23	指定した条件に該当するレコードが存在しませんでした。
FA_EBADKEY	90	指定したレコードキー構成またはレコードキー番号は存在しません。または、正しくありません。

注:これらは代表的なステータスです。これ以外のステータスについては、“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

## H.4.23 レコードの位置決め (cobfa\_strec())

```
long cobfa_strec (
    long    fd,      /* ファイルディスクリプタ */
    long    stflgs, /* 位置付け属性          */
    unsigned long recnum /* 相対レコード番号    */
);
```

#### 説明

相対レコード番号が示すレコードに位置付けます。

ファイルディスクリプタfdが示すファイルにおいて、相対レコード番号recnumの値に関連するレコードに位置付けを行います。

#### パラメタの説明

##### ファイルディスクリプタ(fd)

レコードを位置付けるファイルのファイルディスクリプタを指定します。

### 位置付け属性(stflgs)

指定値は以下です。これは省略可能です。(\*は省略値)

#### 位置付けモード

	記号定数	意味	対応するCOBOL構文
*	FA_EQUAL	recnumと等しいレコード	START KEY IS =
	FA_GREAT	recnumを超えるレコード	START KEY IS >
	FA_GTEQ	recnum以上のレコード	START KEY IS >=
	FA_LESS	recnumより小さいレコード	START KEY IS <
	FA_LTEQ	recnum以下のレコード	START KEY IS <=

### 相対レコード番号(recnum)

位置付けるレコードの相対レコード番号を指定します。

### 実行可能な条件

当関数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	—
	相対ファイル	○
	索引ファイル	—
オープンモード	INPUTモード	○
	OUTPUTモード	—
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	○
	乱呼出し	—
	動的呼出し	○

○:当関数の実行が可能です。

—:当関数の実行はできません。

### 復帰値

当関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当関数の実行が成功しました。
-1	失敗	当関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

### 発生するステータス

当関数の呼出しによって発生するステータスは、下表のようになります。

#### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_ENOTOPEN	47	指定したファイルは、INPUTモード以外、かつ、I-Oモード以外でオープンしてあります。または、不正なファイルディスクリプタを指定しています。
FA_EBADACC	90	当関数を実行することができないファイル編成または、呼出し法でファイルがオープンしてあります。
FA_EBADFLAG	90	当関数を実行することができない位置付けモードを指定しています。または、その他のフラグの指定が正しくありません。
FA_ENOREC	23	指定した条件に該当するレコードが存在しませんでした。

注:これらは代表的なステータスです。これ以外のステータスについては、“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

## H.4.24 レコードロックの解除 (cobfa\_release())

```
long cobfa_release (
    long fd /* ファイルディスクリプタ */
);
```

**説明**

指定するファイルのすべてのレコードロックを解除します。

ファイルディスクリプタfdが示すファイルにおいて、すべてのレコードロックを解除します。

**パラメタの説明**

ファイルディスクリプタ(fd)

レコードロックを解除するファイルのファイルディスクリプタを指定します。

**実行可能な条件**

当関数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	○
	相対ファイル	○
	索引ファイル	○
オープンモード	INPUTモード	—
	OUTPUTモード	—
	I-Oモード	○
	EXTENDモード	—
呼出し法	順呼出し	○
	乱呼出し	○
	動的呼出し	○

○:当関数の実行が可能です。

—:当関数の実行はできません。

**復帰値**

当関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当関数の実行が成功しました。
-1	失敗	当関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

#### 発生するステータス

当関数の呼出しによって発生するステータスは、下表のようになります。

#### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

#### 関数の実行の失敗時(注)

エラー番号	入出力状態	説明
FA_ENOTOPEN	90	ファイルのオープン時に取得したものでない不正なファイルディスクリプタを指定しています。

注:これらは代表的なステータスです。これ以外のステータスについては、“[H.4.36 エラー番号](#)”と“[H.4.37 入出力状態](#)”を参照してください。

## H.4.25 ファイル情報の取得 (cobfa\_indexinfo())

```
long cobfa_indexinfo (
    long          fd,          /* ファイルディスクリプタ */
    struct fa_keydesc *buffer, /* 取得結果の格納域 */
    long          funcrcode /* 機能コード */
);
```

### 説明

索引ファイルの属性またはレコードキーの構成を取得します。

ファイルディスクリプタfdが示す索引ファイルに関する情報を、取得結果の格納域buffer内に設定します。

#### パラメタの説明

##### ファイルディスクリプタ(fd)

索引ファイルのファイルディスクリプタを指定します。

##### 取得結果の格納域(buffer)

索引ファイルに関する情報が格納されます。

##### 機能コード(funcrcode)

取得する情報の種類を指定します。

0を指定すると、取得結果の格納域にstruct fa\_dictinfo型でファイルの属性を格納します。struct fa\_dictinfo型については、“[H.4.34 fa\\_dictinfo構造体](#)”を参照してください。

1以上を指定すると、取得結果の格納域にstruct fa\_keydesc型で主レコードキーまたは副レコードキーの構成を格納します。機能コードの値に1を指定すると主レコードキーの構成を取得し、2以上では副レコードキーの構成を取得します。

この値は、索引ファイルを創成したときの副レコードキーを宣言したときの並びの順番に対応しています。最初の副レコードキーなら2を、2番目の副レコードキーなら3を、これ以降もこれらと同様に指定します。

struct fa\_keydesc型については、“[H.4.32 fa\\_keydesc構造体](#)”を参照してください。

#### 実行可能な条件

当関数の機能を実行することができるファイル編成、オープンモードおよび呼出し法は、それぞれ下表のとおりです。

オープン属性	種別	実行の可否
ファイル編成	行順ファイル	—
	レコード順ファイル	—
	相対ファイル	—
	索引ファイル	○
オープンモード	INPUTモード	○
	OUTPUTモード	○
	I-Oモード	○
	EXTENDモード	○
呼出し法	順呼出し	○
	乱呼出し	○
	動的呼出し	○

○: 当該関数の実行が可能です。  
 —: 当該関数の実行はできません。

#### 復帰値

当該関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	当該関数の実行が成功しました。
-1	失敗	当該関数の実行が失敗しました。エラー番号と入出力状態が、状況を示すコードを保持しています。

#### 発生するステータス

当該関数の呼出しによって発生するステータスは、下表のようになります。

##### 関数の実行の成功時

エラー番号	入出力状態	説明
FA_ENOERR	0	関数の実行は成功しました。

##### 関数の実行の失敗時

エラー番号	入出力状態	説明
FA_ENOTOPEN	90	不正なファイルディスクリプタを指定しています。
FA_EBADACC	90	指定したファイルは索引ファイルではありません。

## H.4.26 エラー番号の取得 (cobfa\_errno())

```
long cobfa_errno (
    void /* 引数なし */
);
```

#### 説明

エラー番号を返却します。

入出力のAPI関数またはファイル情報取得のAPI関数を実行した結果、起こったエラーを識別する番号を取得します。

#### パラメタの説明

ありません。

#### 実行可能な条件

常に呼出し可能。

#### 復帰値

当関数の復帰値は、下表のようになります。

復帰値	状態	説明
FA_ENOERR	成功	入出力機能の実行またはファイル情報取得の実行が成功したことを意味します。
FA_ENOERR以外	失敗	入出力機能の実行またはファイル情報取得の実行が失敗したことを意味します。(注)

注: 復帰値の詳細については、“[H.4.36 エラー番号](#)”を参照してください。

## H.4.27 入出力状態の取得 (cobfa\_stat())

```
long cobfa_stat (  
    void /* 引数なし */  
);
```

#### 説明

入出力状態を返却します。

入出力のAPI関数またはファイル情報取得のAPI関数を実行した結果の入出力状態を取得します。

#### パラメタの説明

ありません。

#### 実行可能な条件

常に呼出し可能。

#### 復帰値

入出力状態を返却します。入出力状態の種類とその詳細については、“[付録D 入出力状態一覧](#)”を参照してください。

## H.4.28 読み込みレコード長の取得 (cobfa\_reclen())

```
long cobfa_reclen (  
    void /* 引数なし */  
);
```

#### 説明

レコード長を返却します。

レコード形式が可変長であるファイルを扱うとき、入出力のAPI関数のうち、以下の関数の実行が成功したあと、読み込んだレコードの実際の長さを返却します。読み込み失敗や他の入出力機能の実行後は、不定な値を返却します。

- cobfa\_rdnnext()
- cobfa\_rdrec()
- cobfa\_rdkey()

#### パラメタの説明

ありません。

#### 実行可能な条件

常に呼出し可能。

## 復帰値

読み込んだレコードの長さ(バイト数)

### H.4.29 相対レコード番号の取得 (cobfa\_recnum())

```
unsigned long cobfa_recnum (  
    void /* 引数なし */  
);
```

#### 説明

相対レコード番号を返却します。

相対ファイルで、入出力のAPI関数のうち、以下の関数の実行が成功したあと、現在位置付けられているレコードの相対レコード番号を返却します。

- cobfa\_rdnex()
- cobfa\_rdnex()

読み込みが失敗した後や、相対ファイル以外のファイル編成での入出力機能の実行後には不定な値を返却します。

#### パラメタの説明

ありません。

#### 実行可能な条件

常に呼出し可能。

#### 復帰値

相対ファイルで、現在位置付けられている相対レコード番号

### H.4.30 ファイルアクセスの排他ロック (LOCK\_cobfa())

```
long LOCK_cobfa (  
    const unsigned long *timeout, /* 待ち時間 */  
    unsigned long *errcode /* エラーコード */  
);
```

#### 説明

マルチスレッド環境下でのCOBOLファイルアクセスに排他ロックをかけます。

マルチスレッド環境下では、複数のスレッドが同時にCOBOLランタイムシステムによるファイルアクセスを行わないように、排他制御する必要があります。

LOCK\_cobfa()は、他のスレッドが同時にCOBOLファイルへのアクセスを行わないように、COBOLランタイムシステムが行うファイルアクセスに対して排他ロックをかけます。マルチスレッド環境下でファイルへのアクセスを行う場合、競合による問題を発生させないために、この関数を呼び出す必要があります。

#### パラメタの説明

##### 待ち時間(timeout)

ミリ秒単位の値を持つ長整数型へのポインタを指定します。もし他のスレッドが先に排他ロックをかけているとき、当スレッドでの関数の呼出しからtimeoutまでの間に排他ロックが解除されないと、関数の実行は失敗します。NULLを指定した場合、待ち時間は無制限となります。

##### エラーコード(errcode)

当関数の復帰値が-1であるとき、エラーコード\*errcodeにWindowsシステムエラーコードが設定されます。errcodeにNULLを設定した場合、エラーコードは設定されません。

当関数の復帰値が-1でない場合の値は不定です。

#### 実行可能な条件

常に呼出し可能。

## 復帰値

関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	COBOLファイルアクセスの排他ロックが成功しました。
1		COBOLファイルアクセスの排他ロックが成功しました。排他ロックをしていた他のスレッドがロックを解除しないで終了したため、代わりに当スレッドがロック制御権を得ました。他のスレッドが異常な終了をしている可能性がありますので、処理を継続する際には十分な注意が必要です。
-1	失敗	システムエラーが発生しました。この場合、エラーコード*errcodeにWindowsシステムエラーコードが設定されます。
-2		待ち時間timeoutを超えました。排他ロックをかけませんでした。

## 使用方法

当関数は、入出力機能を持つAPI関数か、ファイルの情報を取得するAPI関数の前に呼びます。API関数呼出しに続けて、状況を取得するAPI関数を必要に応じて呼んだ後、“[H.4.31 ファイルアクセスの排他ロック解除 \(UNLOCK\\_cobfa\(\)\)](#)”を呼びます。

状況を取得する関数の呼出しは、入出力かファイル情報の取得のAPI関数呼出しとともに、必ず一回の排他ロック中に行ってください。数回の排他制御に分けて状況を取得すると、状況を示す値が他のスレッドで上書きされ、正しくない可能性があります。



### 例

#### 1つの入出力ごとに排他制御を行う例

```
long ret, fd, eno, stat, recnum;

ret = LOCK_cobfa ( NULL, NULL ); /* ファイルアクセスを排他ロックする */
if ( ret < 0 ) { ... /* エラー処理 */ }
fd = cobfa_open ( "file.rel", FA_INPUT | FA_RELFILE | FA_FIXED, 10 );
stat = cobfa_stat ( );
eno = cobfa_errno ( );
UNLOCK_cobfa ( NULL ); /* ファイルアクセスの排他ロックを解除する */
if ( eno == FA_EFNAME ) {
    printf ( "errno: %d, stat: %d\n", eno, stat );
    ...
}

.... /* ファイル入出力に関係のないさまざまな処理を行う */

ret = LOCK_cobfa ( NULL, NULL ); /* ファイルアクセスを排他ロックする */
if ( ret < 0 ) { ... /* エラー処理 */ }
cobfa_rdnxt ( fd, FA_NEXT, buff );
recnum = cobfa_recnum ( );
UNLOCK_cobfa ( NULL ); /* ファイルアクセスの排他ロックを解除する */
num = recnum;

.... /* ファイル入出力に関係のないさまざまな処理を行う */

ret = LOCK_cobfa ( NULL, NULL ); /* ファイルアクセスを排他ロックする */
if ( ret < 0 ) { ... /* エラー処理 */ }
cobfa_close ( fd );
UNLOCK_cobfa ( NULL ); /* ファイルアクセスの排他ロックを解除する */
```

また、最初にCOBOLファイルへのアクセスを始めてから最後にアクセスが終わるまでの間、排他制御をすることもできます。このように、排他制御を行う範囲は任意に広げることができます。



## 例

まとめて排他制御を行う例

```

long ret, fd;

ret = LOCK_cobfa ( NULL, NULL ); /* ファイルアクセスを排他ロックする */
if ( ret < 0 ) { ... /* エラー処理 */ }
fd = cobfa_open ( "file.seq", FA_OUTPUT | FA_SEQFILE | FA_FIXED, 10 );

cobfa_wrnext ( fd, buffer, 10 );
... /* その他、ファイルに対する入出力を行う */

cobfa_close ( fd );
UNLOCK_cobfa ( NULL ); /* ファイルアクセスの排他ロックを解除する */

```

## H.4.31 ファイルアクセスの排他ロック解除 (UNLOCK\_cobfa())

```

long UNLOCK_cobfa (
    unsigned long *errcode /* エラーコード */
);

```

### 説明

マルチスレッド環境下でのファイルアクセスに排他ロックを解除します。

マルチスレッド環境下で、COBOLランタイムシステムのファイルアクセスにかけていた排他ロックを解除します。

### パラメタの説明

#### エラーコード(errcode)

当関数の復帰値が-1であるとき、エラーコード\*errcodeにWindowsシステムエラーコードが設定されます。

errcodeにNULLを設定した場合、エラーコードは設定されません。

当関数の復帰値が-1でない場合の値は不定です。

### 実行可能な条件

常に呼び出し可能。

### 復帰値

関数の復帰値は、下表のようになります。

復帰値	状態	説明
0	成功	COBOLファイルアクセスの排他ロックの解除が成功しました。
-1	失敗	システムエラーが発生しました。この場合、エラーコード*errcodeにWindowsシステムエラーコードが設定されます。

## H.4.32 fa\_keydesc構造体

cobfa\_rdkey()とcobfa\_stkey()では、任意のレコードキーの選択をstruct fa\_keydesc型で指定できます。

cobfa\_indexinfo()は、オープンしてある索引ファイルの任意のレコードキーの構成を知ることができます。

ここでは、レコードキーの構成を与えるstruct fa\_keydesc型のメンバと、設定する/格納される値について説明します。

```

#define FA_NPARTS 254u /* max number of key parts */

struct fa_keydesc {
    long k_flags; /* flags (duplicatable or not) */
    long k_nparts; /* number of parts in key */
};

```

```

    struct fa_keypart k_part [FA_NPARTS]; /* each key part          */
};

```

- `k_flags`には、レコードキーの属性を示す以下の値が入ります。
  - `FA_DUPS` : このレコードキーは重複可能
  - `FA_NODUPS` : このレコードキーは重複を許可しない
- `k_nparts`には、レコードキーの中のパート(キーパート)の数が入ります。キーパート数の最小値は1で、最大値は`FA_NPARTS`(254)です。
- `k_part`は、個々のキーパートの情報を持ちます。struct `fa_keypart`型の配列で宣言しています。以下に説明します。

```

#define FA_NRECSIZE 32760u /* max number of bytes in a record */
#define FA_NKEYSIZE 254u /* max number of bytes in a key */

struct fa_keypart {
    short kp_start; /* starting byte of key part */
    short kp_leng; /* length in bytes */
    long kp_flags; /* flags (UCS-2 key part or not) */
};

```

- `kp_start`には、レコードの先頭位置を0とするバイト単位の変位を設定します。この変位の最大値は`FA_NRECSIZE-1`(32759)です。
- `kp_leng`には、キーパートの長さを設定します。この長さの最小値は1で、上位制限値は、`FA_NKEYSIZE`(254)です。変位と長さの和が`FA_NRECSIZE`(32760)を超えてはいけません。
- `kp_flags`には、以下の1つのカテゴリの情報を設定します。このメンバへの設定値は、ファイルのオープン(`cobfa_open()`)または`cobfa_openW()`のオープン属性引数にキーパートフラグ使用指定(`FA_USEKPFLAGS`)を指定したときだけ有効になります。

表H.7 キーパートのコード系種別

記号定数	説明
<code>FA_UCS2KPCODE</code>	キーパートのコード系種別はUCS-2(リトルエンディアン)である
<code>FA_UTF32KPCODE</code>	キーパートのコード系種別はUTF-32(リトルエンディアン)である
<code>FA_ANYKPCODE</code>	キーパートのコード系種別は、上記以外である

動作モードがUnicodeのCOBOLアプリケーションで扱う索引ファイルで、キーパートがUCS-2(リトルエンディアン)の日本語項目であるときは`FA_UCS2KPCODE`、キーパートがUTF-32(リトルエンディアン)の日本語項目であるときは`FA_UTF32KPCODE`を設定します。キーパートがUCS-2(リトルエンディアン)またはUTF-32(リトルエンディアン)の日本語項目でないときや動作モードがUnicodeでないときには、`FA_ANYKPCODE`を指定します。

レコードキーの構成の取得(`cobfa_indexinfo()`)で`kp_flags`を参照するときには、マスク値`FA_KPCODEMASK`を用いてキーパートのコード系種別を取り出してください。



## 例

### FA\_KPCODEMASKの使用例

```

#include "f3bifcfa.h"
#define GET_PRIM_KEY 1
struct fa_keydesc keydesc1;
:
ret = cobfa_indexinfo ( fd, &keydesc1, GET_PRIM_KEY );
for ( i = 0; i < keydesc1.k_nparts; i ++ ) {
    :
    switch ( keydesc1.k_part[i].kp_flags & FA_KPCODEMASK ) {
        case FA_UCS2KPCODE:
            :
            break;
        case FA_ANYKPCODE:
            :
            break;
    }
}

```

```

}
:
}

```

## 例

### 設定例(1)

- 主レコードキーは重複可能。キーパートは2つ。
- 最初のキーパートは先頭から5バイト目にあり、長さは3。
- 次のキーパートは先頭から11バイト目にあり、長さは5。

バイト目	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
変位	0-- 1-- 2-- 3-- 4-- 5-- 6-- 7-- 8-- 9-- 10- 11- 12- 13- 14- 15-															
	=====															
	-----					-----										
	最初のパート					次のパート										
	part-1					part-2										

### COBOLで記述した場合の参考例

```

:
000100 ENVIRONMENT DIVISION.
000200 CONFIGURATION SECTION.
000300 INPUT-OUTPUT SECTION.
000400 FILE-CONTROL.
000500     SELECT FILENAME-1 ASSIGN TO SYS006
000600     ORGANIZATION IS INDEXED
000700     RECORD KEY IS PART-1 PART-2 WITH DUPLICATES.
000800 DATA DIVISION.
000900 FILE SECTION.
001000 FD FILENAME-1.
001100     01 RECORD-1.
001200         02 FILLER PIC X(4).
001300         02 PART-1 PIC X(3).
001400         02 FILLER PIC X(3).
001500         02 PART-2 PIC X(5).
:

```

### Cソースプログラム例

```

#include "f3bifcfa.h"
struct fa_keydesc keydesc1;

keydesc1.k_flags = FA_DUPS;
keydesc1.k_nparts = 2;           /* number of key parts: 2 */
keydesc1.k_part[0].kp_start = 4; /* part_1: 5 - 1 == 4 */
keydesc1.k_part[0].kp_leng = 3;
keydesc1.k_part[0].kp_flags = FA_ANYKPCODE;
keydesc1.k_part[1].kp_start = 10; /* part_2: 11 - 1 == 10 */
keydesc1.k_part[1].kp_leng = 5;
keydesc1.k_part[1].kp_flags = FA_ANYKPCODE;

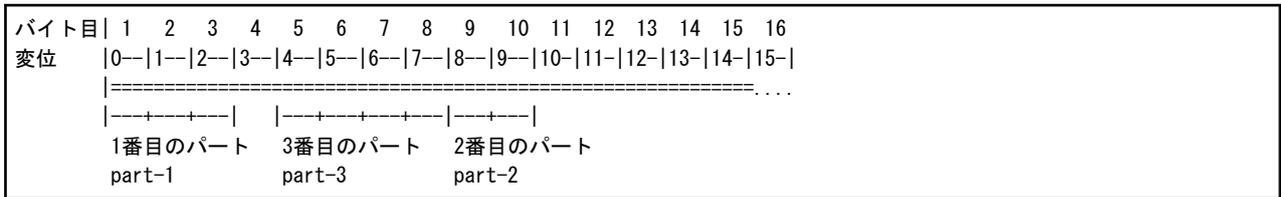
```

## 例

### 設定例(2)

- 主レコードキーは重複を許可しない。キーパートは3つ。

- 1番目のキーパートは先頭から1バイト目にあり、長さは3。
- 2番目のキーパートは先頭から9バイト目にあり、長さは2。
- 3番目のキーパートは先頭から5バイト目にあり、長さは4。



#### COBOLで記述した場合の参考例

```

:
000100 ENVIRONMENT DIVISION.
000200 CONFIGURATION SECTION.
000300 INPUT-OUTPUT SECTION.
000400 FILE-CONTROL.
000500     SELECT FILENAME-1 ASSIGN TO SYS006
000600     ORGANIZATION IS INDEXED
000700     RECORD KEY IS PART-1 PART-2 PART-3.
000800 DATA DIVISION.
000900 FILE SECTION.
001000 FD FILENAME-1.
001100 01 RECORD-1.
001200     02 PART-1 PIC X(3).
001300     02 FILLER PIC X(1).
001400     02 PART-3 PIC X(4).
001500     02 PART-2 PIC X(2).
:

```

#### Cソースプログラム例

```

#include "f3bifcfa.h"
struct fa_keydesc keydesc2;

keydesc2.k_flags = FA_NODUPS;
keydesc2.k_nparts = 3; /* number of key parts: 3 */
keydesc2.k_part[0].kp_start = 0; /* part_1: 1 - 1 == 0 */
keydesc2.k_part[0].kp_leng = 3;
keydesc2.k_part[0].kp_flags = FA_ANYKPCODE;
keydesc2.k_part[1].kp_start = 8; /* part_2: 9 - 1 == 8 */
keydesc2.k_part[1].kp_leng = 2;
keydesc2.k_part[1].kp_flags = FA_ANYKPCODE;
keydesc2.k_part[2].kp_start = 4; /* part_3: 5 - 1 == 4 */
keydesc2.k_part[2].kp_leng = 4;
keydesc2.k_part[2].kp_flags = FA_ANYKPCODE;

```

## H.4.33 fa\_keylist構造体

cobfa\_open()およびcobfa\_openW()では、オープンする索引ファイルのすべてのレコードキーの構成をstruct fa\_keylist型で指定します。ここでは、レコードキー全体の構成を指定するstruct fa\_keylist型のメンバに設定する値について説明します。

```

#define FA_NKEYS 126u /* max number of all keys */

struct fa_keylist {
    long kl_nkeys; /* number of keydesc */
    struct fa_keydesc *kl_key [FA_NKEYS]; /* keydesc address of each key */
};

```

- レコードキーの総数を示すkl\_nkeysには、レコードキーの総数を設定します。索引ファイルは主レコードキーを必ず含むので、レコードキーの総数は必ず1以上になります。なお、レコードキーの総数の最大値は、FA\_NKEYS(126)です。
- 個々のレコードキーの情報を持つkl\_keyは、struct fa\_keydesc型のポインタの配列で宣言しています。struct fa\_keydesc型については、“H.4.32 fa\_keydesc構造体”を参照してください。

なお、すべてのレコードキーが持つキーパート数の合計は、FA\_NALLPARTS(255)を超えてはいけません。また、すべてのレコードキーが持つ各キーパートの長さの合計がFA\_NALLKEYSIZE(255)を超えてはいけません。



## 例

### 設定例

- 索引ファイルのレコードキーの構成は、主レコードキーと副レコードキーが1つずつ、合計2個ある。
- 主レコードキーは重複を許可しない。
- 主レコードキーのキーパートは2つ。
- 最初のキーパートは先頭から1バイト目にあり、長さは4。
- 次のキーパートは先頭から7バイト目にあり、長さは2。
- 副レコードキーは重複可能。
- 副レコードキーのキーパートは1つ。
- キーパートは先頭から12バイト目にあり、長さは3。

バイト目	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
変位	0-- 1-- 2-- 3-- 4-- 5-- 6-- 7-- 8-- 9-- 10-- 11-- 12-- 13-- 14-- 15--															
	===== .....															
	-----				-----				-----							
	主キーの最初のパート				主キーの次のパート				副キーのパート							
	part-1				part-2				subpart							

### COBOLで記述した場合の参考例

```

:
000100 ENVIRONMENT DIVISION.
000200 CONFIGURATION SECTION.
000300 INPUT-OUTPUT SECTION.
000400 FILE-CONTROL.
000500     SELECT FILENAME-1 ASSIGN TO SYS006
000600     ORGANIZATION IS INDEXED
000700     RECORD KEY IS PART-1 PART-2
000800     ALTERNATE RECORD KEY IS SUBPART WITH DUPLICATES.
000900 DATA DIVISION.
001000 FILE SECTION.
001100 FD FILENAME-1.
001200 01 RECORD-1.
001300 02 PART-1 PIC X(4).
001400 02 FILLER PIC X(2).
001500 02 PART-2 PIC X(2).
001600 02 FILLER PIC X(3).
001700 02 SUBPART PIC X(3).
:

```

### Cソースプログラム例

```

#include "f3bifcfa.h"
struct fa_keylist keylist;          /* for all keys structure */
struct fa_keydesc keydesc1;        /* for prime record key */
struct fa_keydesc keydesc2;        /* for alternate record key */

keylist.kl_nkeys = 2;               /* number of keys: 2 (prim & alt) */

```

```

keylist.kl_key[0] = &keydesc1; /* prime key address */
keylist.kl_key[1] = &keydesc2; /* alternate key address */
keydesc1.k_flags = FA_NODUPS;

keydesc1.k_nparts = 2; /* number of key parts: 2 */
keydesc1.k_part[0].kp_start = 0; /* 1 - 1 == 0 */
keydesc1.k_part[0].kp_leng = 4;
keydesc1.k_part[0].kp_flags = FA_ANYKPCODE;
keydesc1.k_part[1].kp_start = 6; /* 7 - 1 == 6 */
keydesc1.k_part[1].kp_leng = 2;
keydesc1.k_part[1].kp_flags = FA_ANYKPCODE;

keydesc2.k_flags = FA_DUPS;
keydesc2.k_nparts = 1; /* number of key parts: 1 */
keydesc2.k_part[0].kp_start = 11; /* 12 - 1 == 11 */
keydesc2.k_part[0].kp_leng = 3;
keydesc2.k_part[0].kp_flags = FA_ANYKPCODE;

```

## H.4.34 fa\_dictinfo構造体

cobfa\_indexinfo()は、オープンしてある索引ファイルの情報をstruct fa\_dictinfo型で返却することができます。

ここでは、索引ファイルの情報を取得するstruct fa\_dictinfo型のメンバに格納される値について説明します。

```

struct fa_dictinfo {
    long di_nkeys; /* number of keys defined */
    long di_resize; /* max or fixed data record size */
    long di_idxsize; /* size of indexes */
    long di_flags; /* other flags (fixed or variable) */
};

```

- di\_nkeysには、索引ファイル中のレコードキーの総数が設定されます。
- ファイルのレコード長を示すdi\_resizeには、固定レコード長または最大レコード長が設定されます。なお、当アクセスルーチンでは、最小レコード長を取得することはできません。
- すべてのレコードキーの長さの合計を示すdi\_idxsizeには、レコードキーの総長が設定されます。
- ファイルの属性を示すdi\_flagsは、以下の1つのカテゴリの情報を持ちます。

表H.8 レコード形式

記号定数	説明
FA_FIXLEN	レコード形式が固定長であることを示す。
FA_VARLEN	レコード形式が可変長であることを示す。

指定した索引ファイルが上記のどの属性に該当するかは、di\_flagsの値と上記のどちらかの値との論理積を求めることによって知ることができます。



例

### Cソースプログラム例

```

#include "f3bifcfa.h"
struct fa_dictinfo di;
long fd, ret;
:
ret = cobfa_indexinfo(fd, &di, 0); /* to get indexed file info */
if (di.di_flags & FA_FIXLEN) {
    /* process for fixed length record type */
}

```

```
}
:
```

## H.4.35 ファイルの機能

ここでは、ファイルの機能について説明します。

### ダミーファイル

ダミーファイルは、実体が存在しない架空のファイルです。

API関数を実行する対象がダミーファイルの場合、物理的なファイル操作は行われません。例えば、OUTPUTモードのcobfa\_open関数を実行した場合、通常はオープンが成功するとファイルが生成されて書き込み可能な状態になりますが、ダミーファイルを指定した場合、オープンは成功しますが、ファイルは生成されません。

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

トラブル発生時のログファイルは、通常の運用時は不要なファイルです。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。

#### プログラム開発中で、入力ファイルがない場合

通常は、空のファイルを用意してテストを進めます。この場合、入力ファイルをダミーファイルにすれば、空の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。

#### 使用方法

cobfa\_open()またはcobfa\_openW()に指定するファイル名の末尾に“,DUMMY”を付加するか、またはファイル名を省略して“,DUMMY”だけを指定します。

```
fname = " [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”をファイル名としてみなして、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。

#### ダミーファイルに対する動作

入出力機能の各API関数でダミーファイルを使用する場合、ダミーファイルに対する動作は以下のとおりです。

入出力機能	API関数	復帰値	状態	エラー番号	入出力状態
ファイルのオープン	cobfa_open()	1以上	成功	FA_ENOERR	0
	cobfa_openW()				
ファイルのクローズ	cobfa_close()	0	成功	FA_ENOERR	0
レコードの書き出し	cobfa_wrnext()				
レコードの位置決め	cobfa_wrrec()				
レコードロックの解除	cobfa_wrkey()				
	cobfa_strec()				
	cobfa_stkey()				
	cobfa_release()				
レコードの読み込み	cobfa_rdnex()	-1	失敗	FA_EENDFILE	10

入出力機能	API関数	復帰値	状態	エラー番号	入出力状態
	cobfa_rdrec()	-1	失敗	FA_ENOREC	23
	cobfa_rdkey()				
レコードの削除 レコードの書換え	cobfa_delcurr()	-1	失敗	FA_ENOCURR	43
	cobfa_rewcurr()				
	cobfa_delrec()	-1	失敗	FA_ENOREC	23
	cobfa_delkey()				
	cobfa_rewrec()				
	cobfa_rewkey()				

## 注意

ファイルアクセスルーチンでは、索引ファイルのオープンでレコードキーの指定を省略することができます。この場合、既存ファイルの索引構成とレコード形式、レコード長を認識してオープンします。

ダミーファイル機能を指定した場合も同様に、レコードキーの指定を省略することができます。ただし、ファイル情報の取得を行うcobfa\_indexinfo関数は失敗します。エラー番号はFA\_EUNDEFKEY、入出力状態は90になります。

## 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のアクセス性能は、使用する機能を限定することで高速化することができます。本機能は、以下のAPI関数で有効となります。

- cobfa\_wrnext (レコードの順書出し)
- cobfa\_rdnex (レコードの順読込み)

### 使用方法

cobfa\_open関数またはcobfa\_openW関数に指定するファイル名に続いて“,BSAM”を指定します。

```
fname = "ファイル名,BSAM";
```

## 注意

- ファイル編成がレコード順ファイル(FA\_SEQFILE)または行順ファイル(FA\_LSEQFILE)の場合に有効となります。これ以外のファイル編成を指定した場合、cobfa\_open関数またはcobfa\_openW関数の実行が失敗します。
- レコードの更新はできません。オープンモードにI-Oモード(FA\_INOUT)を指定した場合、cobfa\_open関数またはcobfa\_openW関数の実行が失敗します。
- 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、すべてのプロセスで、そのファイルが共用モードでかつINPUTモード(FA\_INPUT)でオープンされている必要があります。INPUTモード以外でオープンしたファイルがある場合、動作は保証されません。
  - 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。
- ファイルの高速処理を指定した場合、行順ファイル(FA\_LSEQFILE)から読み込んだレコードにタブが含まれていても、そのタブ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。

## H.4.36 エラー番号

エラー番号はAPI関数のエラーの種別を返却します。これには入出力状態だけでは表現できないCOBOLの翻訳エラーに相当する情報も含まれます。詳細については、“[H.4.26 エラー番号の取得 \(cobfa\\_ernno\(\)\)](#)”を参照してください。

以下にエラー番号とその意味について説明します。

エラー番号	説明
FA_ENOERR(0)	入出力機能の実行またはファイル情報取得の実行が成功したことを意味します。
FA_ENOSPC(28)	ディスク容量が不足しています。
FA_EDUPL(100)	キーの重複に関するエラーです。以下のどちらかの状態です。 <ul style="list-style-type: none"> <li>・レコードキーに指定した値を持つレコードが、すでにファイルに存在します。レコードキーは重複を許していません。</li> <li>・相対レコード番号が既存のものと重複します。</li> </ul>
FA_ENOTOPEN(101)	ファイルのオープンに関するエラーです。以下のどちらかの状態です。 <ul style="list-style-type: none"> <li>・まだオープンしていないファイルです。</li> <li>・ファイルは、この機能を実行することができないオープンモードでオープンしています。</li> </ul>
FA_EBADARG(102)	引数に関するエラーです。以下のいずれかの状態です。 <ul style="list-style-type: none"> <li>・レコード域引数がNULLポインタです。</li> <li>・ファイル情報の取得関数で、機能番号が範囲外です。</li> <li>・ファイル情報の取得関数で、構造体ポインタがNULLです。</li> </ul>
FA_EBADKEY(103)	索引ファイルのレコードキー指定に関するエラーです。以下のいずれかの状態です。 <ul style="list-style-type: none"> <li>・与えたレコードキー構成リストに矛盾があります。</li> <li>・与えたレコードキー構成はファイルのキー構成と一致するものではありません。</li> <li>・与えたレコードキー番号はファイルが持つキーの数を超えています。</li> </ul>
FA_ETOOMANY(104)	OSまたは当アクセスルーチンの制限値を超える数のファイルのオープンを行おうとしました。
FA_EBADFILE(105)	ファイルの内部構成に関するエラーです。以下のいずれかの状態です。 <ul style="list-style-type: none"> <li>・ファイルの内部情報が正しくないか、破壊されています。</li> <li>・正しいファイル編成を指定していません。</li> <li>・動作コード系の指定と、行順ファイルのエンコード形式(シフトJIS、UCS-2、UTF-8)が一致していません。</li> </ul>
FA_ELOCKED(107)	レコードはすでにロックされています。
FA_EENDFILE(110)	ファイル終了条件が発生しました。
FA_ENOREC(111)	指定したレコードは存在しません。
FA_ENOCURR(112)	レコードへの位置付けが不定です。
FA_EFLOCKED(113)	ファイルはすでに排他オープンされています。
FA_EFNAME(114)	オープン時に与えたファイル名に関するエラーです。以下のいずれかの状態です。 <ul style="list-style-type: none"> <li>・ファイルが存在しません。</li> <li>・ファイルにはアクセスすることができません。</li> <li>・ファイル名がNULLポインタまたは空文字列です。</li> <li>・ファイル名の構成が正しくありません。</li> <li>・読み込み専用属性のファイルをINPUTモード以外のモードでオープンしようとしてしました。</li> </ul>
FA_EBADMEM(116)	機能を実行するために必要なメモリの獲得が失敗しました。
FA_EKEYSEQ(117)	キーの順序誤りまたは変更誤りです。以下のどちらかの状態です。 <ul style="list-style-type: none"> <li>・順書出で、主レコードキー値が昇順ではありません。</li> <li>・順書換えて、主レコードキー値を変更しようとしてしました。</li> </ul>

エラー番号	説明
FA_EBADACC(118)	実行不可能な組合せが発生しました。以下のいずれかの状態です。 <ul style="list-style-type: none"> <li>・ 呼出し法に違反する機能の実行を要求しました。</li> <li>・ このファイル編成では実行できない機能です。</li> <li>・ オープン時のフラグの組合せが正しくありません。</li> </ul>
FA_EBADFLAG(120)	フラグの指定値が正しくありません。以下のどちらかの状態です。 <ul style="list-style-type: none"> <li>・ オープンモード、読み込みモード、位置付けモードに使用できないモードを指定しています。</li> <li>・ その他、受け入れることができない値をフラグに指定しています。</li> </ul>
FA_EBADLENG(121)	長さに関するエラーです。以下のどちらかの状態です。 <ul style="list-style-type: none"> <li>・ レコード長がファイルの定量制限を超えています。“<a href="#">H.4.39 COBOLファイルアクセスルーチンの留意事項</a>”の“ファイル機能全般”を参照してください。</li> <li>・ 位置付け時の有効キー長が、ファイルが持つキーパートの長さの合計を超えています。</li> </ul>
FA_EUNDEFKEY(122)	索引ファイル情報を取得できません。 ダミーファイル機能を指定して、ファイルのオープン時にレコードキーの指定を省略した場合、索引ファイル情報は取得できません。 詳細は、“ <a href="#">ダミーファイル</a> ”を参照してください。
FA_EOTHER(999)	上記以外のエラーが発生しました。この場合、cobfa_stat()関数の復帰値を取得して状況を判断してください。cobfa_stat()関数については“ <a href="#">H.4.27 入出力状態の取得 (cobfa_stat())</a> ”を参照してください。

## H.4.37 入出力状態

当アクセスルーチンでは、入出力状態を取得するとき、cobfa\_stat()関数を呼び出します。cobfa\_stat()関数については“[H.4.27 入出力状態の取得 \(cobfa\\_stat\(\)\)](#)”を参照してください。

入出力状態の種類とその詳細は、“[付録D 入出力状態一覧](#)”を参照してください。

## H.4.38 COBOLファイルアクセスルーチンの制限事項

### 索引ファイル

動的呼出し法(FA\_DYNACC)でオープンしたファイルが、以下のどちらかの条件のとき、順読み込みで位置付けられているレコードに対して、順書換え/順削除を実行する方法はありません。

- ・ cobfa\_stkey()関数で、逆順読み込み(FA\_REVORD)を指定した場合
- ・ 主レコードキーの属性が重複可能(FA\_DUPS)である場合

## H.4.39 COBOLファイルアクセスルーチンの留意事項

### ファイル機能全般

- ・ アプリケーションプログラムでオープンしたファイルは、必ず、すべてをクローズしてから処理を終了してください。この操作を行わないとファイルの内容の破壊、システムリソースのリークなどの問題が発生します。
- ・ 同一プロセス内で、同一ファイルをオープンしても二重オープンエラー(入出力状態:41)は発生しません。この場合、別のファイルディスクリプタが割り当てられます。なお、同一プロセス内で同時にオープンできるファイルディスクリプタの最大数は1,024です。
- ・ API関数の実行時にエラーが複数重なる場合には、COBOLアプリケーションとは異なった入出力状態を返却する場合があります。
- ・ COBOLでは同じ入出力文を使っても異なる呼出し法を扱えますが、APIによる入出力では、呼出し法によって使用する関数が異なります。適切な関数を使用してください。
- ・ 入出力の状況(cobfa\_errno(), cobfa\_stat(), cobfa\_reclen(), cobfa\_reclen())の各関数の復帰値)はファイルごとに情報を保持しません。これらの値はプロセス単位で保持するので、実行した入出力機能の状況値が必要な場合は、次の入出力機能を実行する前までに値を変数などに保存してください。以前の入出力の状況の値は、次の入出力機能の実行によって上書きされます。

- レコードの読み込み/書出し/書換え/位置付けを行う場合、データ受渡し用のレコード域は、ファイルをオープンするときの最大/固定レコード長以上の大きさを持つ領域をあらかじめ確保しておく必要があります。
- COBOLでは翻訳時にエラーが検出できる場合でも、API関数では実行時にしかエラーを検出できないので注意してください。具体的には、呼出し法の不一致や正しくない索引キー指定などがこれに該当します。
- ファイルに関する定量制限は、以下を参照してください。
  - “E.4 順ファイル”、“E.6 行順ファイル”、“E.7 相対ファイル”、“E.8 索引ファイル”
  - “表7.10 各ファイルシステムの機能差”
- COBOLのデータ型とC言語のデータ型の対応については、以下を参照してください。
  - “10.3.3 データ型の対応”
 また、2進項目のCOMP-5とBINARYは内部表現が異なるため、扱いに注意してください。
- ファイルをオープンする際、指定するファイル名のコード系に注意してください。
  - cobfa\_open() : Shift-JIS
  - cobfa\_openW() : Unicode(UTF-16)
 上記と異なるコード系のファイル名を指定した場合、動作は保証されません。
- コンマ(,)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。

## 行順ファイル

- 0バイトの長さのレコードを書き出すことはできません。
- 本製品では、0x0D 0x0A(復帰+改行)を改行文字として扱い、レコードを書き出すときに付加します。
- 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、以下の通りです。

制御文字	意味	動作
0x0C	改頁	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。

- 読み込んだレコードにタブが存在した場合、そのタブコードを空白に置き換えます。詳細は、以下を参照してください。
  - “レコード内のタブの扱い”

## レコード順ファイル

- 印刷ファイルを扱えません。したがって、以下の仕様となります。
  - COBOL構文のLINAGE句に相当するものではありません。
  - COBOL構文のWRITE文の改行制御/ページ制御に相当するものではありません。

## 相対ファイル

- 以下の関数では、相対レコード番号を引数で明に指定する必要があります。
  - cobfa\_delrec()関数 : 乱削除
  - cobfa\_rewrec()関数 : 乱書換え
  - cobfa\_rdrec()関数 : 乱読み込み
  - cobfa\_strec()関数 : 位置決め
- 以下の関数でアクセスしたレコードの相対レコード番号は、cobfa\_recnum()関数で取得することができます。
  - cobfa\_rdnxt()関数 : 順読み込み

- `cobfa_wrnext()`関数：順書出し

## 索引ファイル

- 既存ファイルのオープン時に、レコードキー構成を指定しないでオープンすることができます。この場合、ファイルアクセスルーチンは既存ファイル内部のレコードキー構成を調査してオープンします。
- オープン中の索引ファイルの属性(レコード長、レコード形式)とレコードキー構成を調べる機能(`cobfa_indexinfo()`関数)があります。

## マルチスレッド

- 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。
- COBOLファイルアクセスの排他制御(排他ロック/排他ロック解除)には、必ず以下の関数を使ってください。詳細については、各API関数の説明を参照してください。

- [LOCK\\_cobfa\(\)](#)

- [UNLOCK\\_cobfa\(\)](#)

- マルチスレッド環境下では、入出力機能を持つAPI関数の呼出し前、またはファイルの情報を取得するAPI関数の呼出し前に、必ず排他制御を行ってください。排他制御を行わないと、競合によって以下のような問題が発生する可能性があります。

- ファイル入出力の不正な実行結果

- スレッドの異常終了

- ファイル内容の破壊

排他制御のロック・アンロックのタイミングについては、“[H.4.30 ファイルアクセスの排他ロック \(LOCK\\_cobfa\(\)\)](#)”の使い方の例を参照してください。

## Unicode

V60L10以降では、Unicodeを扱うために以下の追加/変更があります。

- `struct fa_keypart`型の構造体にメンバ`kp_flags`を追加しました。`struct fa_keypart`型については、“[H.4.32 fa\\_keydesc構造体](#)”を参照してください。
- ファイルのオープン(`cobfa_open()`関数)のオープン属性指数が次の2つのカテゴリを認識するようになりました。
  - 動作コード系
  - キーパートフラグ使用指定
- ファイルのオープンのオープン属性指数に与える索引ファイル(`FA_IDXFILE`)の定義値を変更しました。

Unicodeを扱う場合、以下に留意してください。

- 動作モードがUnicodeのCOBOLアプリケーションで扱う索引ファイルのキーパート長`kp_leng`は、バイト数を設定してください。
- 動作モードがUnicodeのCOBOLアプリケーションで扱う行順ファイルのレコード長には、バイト数を指定してください。

## その他

- V50L10以前のファイルアクセスルーチンを使って作成したアプリケーションやDLLなどは、正しく動作します。しかし、V60L10でヘッダファイルの内容を一部変更しているため、V50L10以前のファイルアクセスルーチン用に作成したものを修正する場合は、必ず関連するソースプログラムのすべてを翻訳しなおすようにしてください。

# 付録I 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。

## I.1 サブルーチン一覧

### ウィンドウハンドルの獲得

- ・ I.2.16 JMPBGWDH(ウィンドウハンドル獲得サブルーチン)

### インスタンスハンドルの獲得

- ・ I.2.17 JMPBWINS(インスタンスハンドル獲得サブルーチン)

### イベントログ出力

- ・ I.2.9 COB\_REPORT\_EVENT(イベントログ出力サブルーチン)

### マルチスレッドプログラムのデバッグ補助

- ・ I.2.5 COB\_GET\_PROCESSID(プロセスID取得サブルーチン)
- ・ I.2.6 COB\_GET\_THREADID(スレッドID取得サブルーチン)

### メモリ関連

- ・ I.2.1 COB\_ALLOC\_MEMORY (メモリ割当てサブルーチン)
- ・ I.2.4 COB\_FREE\_MEMORY(メモリ解放サブルーチン)

[参照]“I.5 メモリ関連サブルーチンの使い方”

### プロセスの強制終了

- ・ I.2.3 COB\_EXIT\_PROCESS(プロセス終了サブルーチン)

### スレッド同期制御

- ・ I.2.7 COB\_LOCK\_DATA(データロック獲得サブルーチン)
- ・ I.2.10 COB\_UNLOCK\_DATA(データロック解放サブルーチン)
- ・ I.2.8 COB\_LOCK\_OBJECT(オブジェクトロック獲得サブルーチン)
- ・ I.2.11 COB\_UNLOCK\_OBJECT(オブジェクトロック解放サブルーチン)

[参照]“I.6 スレッド同期制御サブルーチンの使い方”

### デッドロック出口スケジュール

- ・ I.2.2 COB\_DEADLOCK\_EXIT(デッドロック出口スケジュールサブルーチン)

### データのエンディアンを変換

- ・ I.2.18 #NATLETOBE(ビッグエンディアン変換サブルーチン)
- ・ I.2.19 #NATBETOLE(リトルエンディアン変換サブルーチン)

### 他言語連携で使用するサブルーチン

- ・ I.3.3 JMPCINT2(実行単位の開始サブルーチン)
- ・ I.3.4 JMPCINT3(実行単位の終了サブルーチン)

- ・ I.3.5 JMPCINT4(実行環境の閉鎖サブルーチン)
- ・ I.3.1 COB\_GETRUNIT(COBOL実行単位ハンドル取得サブルーチン)
- ・ I.3.2 COB\_SETRUNIT(COBOL実行単位ハンドル設定サブルーチン)

## 簡易アプリ間通信

- ・ I.2.13 COBCI\_OPEN(サーバとの接続の確立)
- ・ I.2.12 COBCI\_CLOSE(サーバとの接続の切断)
- ・ I.2.14 COBCI\_READ(サーバからのメッセージの読み込み)
- ・ I.2.15 COBCI\_WRITE(サーバへのメッセージの書き込み)

[参照]“I.7 簡易アプリ間通信のサブルーチンの使い方”

## Webサブルーチン

### 作業環境の設定およびWebパラメタの獲得

- ・ I.4.13 COBW3\_INIT(作業環境の設定およびWebパラメタの獲得)

### Webパラメタの操作

- ・ I.4.25 COBW3\_GET\_VALUE\_XX、COBW3\_GET\_VALUE\_NX、COBW3\_GET\_VALUE\_XN、COBW3\_GET\_VALUE\_NN (Webパラメタ中のNAMEの検索およびVALUEの取得)
- ・ I.4.2 COBW3\_CHECK\_VALUE\_X、COBW3\_CHECK\_VALUE\_N (Webパラメタ中のVALUEの検索)

### 処理結果の出力

- ・ I.4.27 COBW3\_PUT\_HEAD(ヘッダの出力)
- ・ I.4.28 COBW3\_PUT\_HTML(結果出力用ページの出力)
- ・ I.4.31 COBW3\_SET\_CNV\_XX、COBW3\_SET\_CNV\_NX、COBW3\_SET\_CNV\_XN、COBW3\_SET\_CNV\_NN (結果出力用ページ内の変換データの登録)
- ・ I.4.6 COBW3\_DEL\_CNV\_X、COBW3\_DEL\_CNV\_N (結果出力用ページ内の変換データの削除)
- ・ I.4.14 COBW3\_INIT\_CNV(結果出力用ページ内の変換データの初期化)
- ・ I.4.34 COBW3\_SET\_REPEAT\_XX、COBW3\_SET\_REPEAT\_NX、COBW3\_SET\_REPEAT\_XN、COBW3\_SET\_REPEAT\_NN (結果出力用ページ内の繰返し変換データの登録)
- ・ I.4.9 COBW3\_DEL\_REPEAT\_X、COBW3\_DEL\_REPEAT\_N (結果出力用ページ内の繰返し変換データの削除)
- ・ I.4.16 COBW3\_INIT\_REPEAT(結果出力用ページ内の繰返し変換データの初期化)
- ・ I.4.29 COBW3\_PUT\_TEXT(データの出力(改行コードを付加))

### システムコマンドの実行

- ・ I.4.37 COBW3\_SYSTEM(システムコマンドの実行)

### Web実行環境の資源の解放

- ・ I.4.12 COBW3\_FREE(Web実行環境の資源の解放)

### リクエスト情報の取得

- ・ I.4.30 COBW3\_RECEIVE\_HEADER(HTTPヘッダの取得)
- ・ I.4.21 COBW3\_GET\_REQUEST\_INFO(リクエストに関する情報の取得)
- ・ I.4.18 COBW3\_GET\_AUTHORIZE(認証情報の取得)

### Cookieデータ操作

- I.4.33 COBW3\_SET\_COOKIE\_XX、COBW3\_SET\_COOKIE\_NX、COBW3\_SET\_COOKIE\_XN、COBW3\_SET\_COOKIE\_NN (Cookieデータの登録)
- I.4.8 COBW3\_DEL\_COOKIE\_X、COBW3\_DEL\_COOKIE\_N (Cookieデータの削除)
- I.4.15 COBW3\_INIT\_COOKIE (登録済みの全Cookieデータの初期化)
- I.4.20 COBW3\_GET\_COOKIE\_XX、COBW3\_GET\_COOKIE\_NX、COBW3\_GET\_COOKIE\_XN、COBW3\_GET\_COOKIE\_NN (WebパラメタからCookieデータの取得)

#### ファイルアップロード

- I.4.24 COBW3\_GET\_UPLOADFILE\_INFO\_X、COBW3\_GET\_UPLOADFILE\_INFO\_N (アップロードファイル情報の取得)
- I.4.17 COBW3\_GEN\_UPLOADFILE\_X、COBW3\_GEN\_UPLOADFILE\_N (アップロードされたファイルをサーバ上に生成)
- I.4.10 COBW3\_DEL\_UPLOADEDFILE (アップロード済みファイルの削除)

#### セッション管理

- I.4.36 COBW3\_START\_SESSION (セッションの開始)
- I.4.11 COBW3\_END\_SESSION (セッションの終了)
- I.4.35 COBW3\_SET\_SESSION\_DATA (セッションデータの登録)
- I.4.22 COBW3\_GET\_SESSION\_DATA (セッションデータの取得)
- I.4.1 COBW3\_ALTER\_SESSION\_TIMEOUT (セッションのタイムアウト時間の変更)
- I.4.23 COBW3\_GET\_SESSION\_INFO (セッション情報の取得)

[参照]“I.8 COBOL Webサブルーチンの使い方

なお、以下のサブルーチンは互換を保証するために提供されています。

- I.4.26 COBW3\_NAME、COBW3\_NAMEN (Webパラメタから任意の名前を検索)
- I.4.38 COBW3\_VALUE (Webパラメタから任意の値を検索)
- I.4.5 COBW3\_CNV\_SET (結果出力用ページ内に指定された変換データを登録)
- I.4.3 COBW3\_CNV\_DEL (登録した変換データの削除)
- I.4.4 COBW3\_CNV\_INIT (登録した変換データをすべて初期化)
- I.4.32 COBW3\_SET\_COOKIE (ブラウザへ送信するCookieデータの登録)
- I.4.7 COBW3\_DEL\_COOKIE (登録したCookieデータの削除)
- I.4.19 COBW3\_GET\_COOKIE、COBW3\_GET\_COOKIEIEN (リクエストCookieデータから任意の名前の検索)

## I.2 COBOLプログラムから呼び出すサブルーチン

### I.2.1 COB\_ALLOC\_MEMORY (メモリ割当てサブルーチン)

#### 説明

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。

## 復帰コード

01 データ名-4 PIC S9(9) COMP-5.

復帰コードの値とエラーの原因は以下のとおりです。

値	意味
0	成功。
-1	失敗。パラメタの指定に誤りがあります。
-2	失敗。メモリが不足しています。
-3	失敗。実行単位が既に終了しています。

## 使い方

“[I.5 メモリ関連サブルーチンの使い方](#)”を参照してください。

## I.2.2 COB\_DEADLOCK\_EXIT(デッドロック出口スケジュールサブルーチン)

### 説明

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、“[15.2.14 デッドロック出口](#)”を参照してください。

### 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。

### 呼出し形式

CALL "COB\_DEADLOCK\_EXIT".

## パラメタの説明

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。

## 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。

## I.2.3 COB\_EXIT\_PROCESS(プロセス終了サブルーチン)

### 説明

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戻します。
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。

### 復帰コード

```
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。

## 使い方

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pl리케이션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

- Windows エラー報告



例

.....

診断レポートの例

## NetCOBOL 診断機能

### <<要約>>

アプリケーションが発生しました:  
アプリケーション : D:\APL\SAMPLE.exe (PID=000009A0)  
例外種別 : EXCEPTION\_NONCONTINUABLE\_EXCEPTION (C0000025)  
発生時間 : YYYY/MM/DD (H:MM:SS)  
発生モジュール : C:\WINDOWS\system32\kernel.dll  
作成時間 : YYYY/MM/DD (H:MM:SS)  
ファイルサイズ : 629760bytes

### <<問題箇所>>

バイト ID : 0000086C  
バイト : EAX=0012F9E0 EBX=0012FFB0 ECX=00000000 EDI=0056D6A0 ESI=0012FA5C  
EDI=00000000 EIP=7C98E9FA ESP=0012F9D4 EBP=0012FA44 EFL=00000202  
CS=001B SS=0023 DS=0023 ES=0023 FS=003B GS=0000  
ポインタ : 00000000 (オフ:00130000, オフ:00123000)  
命令 : jnb edi +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +a +b +c +d +e +f  
7C98E9EA 80 01 00 00 89 75 A4 89 4D AC EB 52 00 FC FF  
オフ -> 7C98E9FA E9 12 BF FD FF 83 A5 E4 FC FF FF 00 8D 85 D4 FC

モジュール : C:\WINDOWS\system32\kernel.dll  
アドレス : .text+0004DAFA  
EXPORT 相対位置 : RtlInitializeSListHead+00015AEA

### <呼出経路>

- [ 1 ]  
モジュール : C:\WINDOWS\system32\kernel.dll  
アドレス : .text+0000DAFA  
EXPORT 相対位置 : KiUserExceptionDispatcher+0000000E
- [ 2 ]  
モジュール : C:\Program Files\NetCOBOL\F381EFNC.dll  
アドレス : .text+0000130B  
EXPORT 相対位置 : COB\_EXIT\_PROCESS+0000005B
- [ 3 ]  
モジュール : D:\APL\SAMPLE.DLL.dll  
アドレス : .text+000001DE  
EXPORT 相対位置 : SAMPLE01+000001A2
- [ 4 ]  
モジュール : D:\APL\SAMPLE.exe  
アドレス : .text+000001C9  
オフ : 004011C9
- [ 5 ]  
モジュール : D:\APL\SAMPLE.exe  
アドレス : .text+0000000A  
オフ : 0040100A
- [ 6 ]  
モジュール : D:\APL\SAMPLE.exe  
アドレス : .text+0000031E  
オフ : 0040131E
- [ 7 ]  
モジュール : C:\WINDOWS\system32\kernel32.dll  
アドレス : .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。

## 1.2.4 COB\_FREE\_MEMORY(メモリ解放サブルーチン)

### 説明

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

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。

#### 復帰コード

01 データ名-2 PIC S9(9) COMP-5.

復帰コードの値と意味は以下のとおりです。

値	意味
0	成功。
-1	失敗。エラーの原因として以下が考えられます。 <ul style="list-style-type: none"><li>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。</li><li>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。</li></ul>

#### 使い方

“[I.5 メモリ関連サブルーチンの使い方](#)”を参照してください。



#### 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。

## I.2.5 COB\_GET\_PROCESSID(プロセスID取得サブルーチン)

#### 説明

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。

#### 呼出しの形式

```
CALL "COB_GET_PROCESSID" USING BY REFERENCE データ名.
```

#### パラメタの説明

01 データ名 PIC 9(9) COMP-5.

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、“[5.6 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=F3BIEFNC.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。

## I.2.6 COB\_GET\_THREADID(スレッドID取得サブルーチン)

### 説明

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。

### 呼出し形式

```
CALL "COB_GET_THREADID" USING BY REFERENCE データ名.
```

### パラメタの説明

```
01 データ名 PIC 9(9) COMP-5.
```

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。



- ・ 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、“5.6 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。

## I.2.7 COB\_LOCK\_DATA(データロック獲得サブルーチン)

### 説明

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。

### 呼出し形式

```
CALL "COB_LOCK_DATA" WITH C LINKAGE
      USING BY REFERENCE LOCK-KEY
            BY VALUE WAIT-TIME
            BY REFERENCE ERR-DETAIL
      RETURNING RET-VALUE.
```

### パラメタの説明

#### LOCK-KEY

```
01 LOCK-KEY PIC X(30).
```

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

```
01 WAIT-TIME PIC S9(9) COMP-5.
```

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

01	ERR-DETAIL	PIC 9(9)	COMP-5.
----	------------	----------	---------

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。

## 復帰コード

01	RET-VALUE	PIC S9(9)	COMP-5.
----	-----------	-----------	---------

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、“[I.6.3 スレッド同期制御サブルーチンのエラーコード](#)”を参照してください。

## 使い方

“[I.6.1 データロックサブルーチン](#)”を参照してください。

## I.2.8 COB\_LOCK\_OBJECT(オブジェクトロック獲得サブルーチン)

### 説明

指定されたオブジェクトに対してロックを獲得します。

### 呼出し形式

```
PROCEDURE DIVISION.  
    CALL "COB_LOCK_OBJECT" WITH C LINKAGE  
        USING BY REFERENCE OBJ  
              BY VALUE WAIT-TIME  
              BY REFERENCE ERR-DETAIL  
        RETURNING RET-VALUE.
```

### パラメタの説明

#### OBJ

01	OBJ	OBJECT REFERENCE	クラス名.
----	-----	------------------	-------

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

01	WAIT-TIME	PIC S9(9)	COMP-5.
----	-----------	-----------	---------

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

01	ERR-DETAIL	PIC 9(9)	COMP-5.
----	------------	----------	---------

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。

## 復帰コード

01	RET-VALUE	PIC S9(9)	COMP-5.
----	-----------	-----------	---------

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、“[I.6.3 スレッド同期制御サブルーチンのエラーコード](#)”を参照してください。

## 使い方

“[I.6.2 オブジェクトロックサブルーチン](#)”を参照してください。

## I.2.9 COB\_REPORT\_EVENT(イベントログ出力サブルーチン)

### 説明

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。

出力される情報については、“[18.3.2 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。

### 呼出し形式

```
INITIALIZE データ名-1.  
CALL "COB_REPORT_EVENT" USING データ名-1  
RETURNING データ名-2.
```

### パラメタの説明

```
WORKING-STORAGE SECTION.  
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).
```

#### 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.
```

- 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が返却される場合があります。
- 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

```
01 データ名-2 PIC S9(9) COMP-5.
```

復帰コードの値と意味は以下のとおりです。

値	意味
0	イベントログへの出力に成功しました。
1	イベント番号の指定に誤りがあります。
2	種類の指定に誤りがあります。
3	OSが、32ビットWindowsではありません。

値	意味
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。“メッセージ集”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、“18.3.2 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。
- 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、“5.6 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
```

## 1.2.10 COB\_UNLOCK\_DATA(データロック解放サブルーチン)

### 説明

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。

### 呼出し形式

```
CALL "COB_UNLOCK_DATA" WITH C LINKAGE
      USING BY REFERENCE LOCK-KEY
            BY REFERENCE ERR-DETAIL
      RETURNING RET-VALUE.
```

### パラメタの説明

#### LOCK-KEY

```
01 LOCK-KEY      PIC X(30).
```

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

```
01 ERR-DETAIL    PIC 9(9) COMP-5.
```

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。

### 復帰コード

```
01 RET-VALUE     PIC S9(9) COMP-5.
```

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、“[I.6.3 スレッド同期制御サブルーチンのエラーコード](#)”を参照してください。

#### 使い方

“[I.6.1 データロックサブルーチン](#)”を参照してください。

## I.2.11 COB\_UNLOCK\_OBJECT(オブジェクトロック解放サブルーチン)

#### 説明

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.
```

#### パラメタの説明

##### OBJ

```
01 OBJ          OBJECT REFERENCE  クラス名.
```

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

```
01 ERR-DETAIL   PIC 9(9)  COMP-5.
```

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。

#### 復帰コード

```
01 RET-VALUE    PIC S9(9) COMP-5.
```

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、“[I.6.3 スレッド同期制御サブルーチンのエラーコード](#)”を参照してください。

#### 使い方

“[I.6.2 オブジェクトロックサブルーチン](#)”を参照してください。

## I.2.12 COBCI\_CLOSE(サーバとの接続の切断)

簡易アプリ間通信で使用するサブルーチンです。

#### 説明

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
                    BY VALUE      UNUSED-REC.

```

:

#### パラメタの説明

##### 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。

詳細は“[I.7.2 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。

##### 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。

##### 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

サーバとの切断に成功した場合は、特殊レジスタPROGRAM-STATUSに0を返します。  
サーバとの切断に失敗した場合は、特殊レジスタPROGRAM-STATUSに-1を返します。

#### 使い方

“[I.7 簡易アプリ間通信のサブルーチンの使い方](#)”を参照してください。

## I.2.13 COBCI\_OPEN(サーバとの接続の確立)

簡易アプリ間通信で使用するサブルーチンです。

#### 説明

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.

```

:

## パラメタの説明

### 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。

詳細は“[1.7.2 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。

### 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。

## 使い方

“[1.7 簡易アプリ間通信のサブルーチンの使い方](#)”を参照してください。

## I.2.14 COBCI\_READ(サーバからのメッセージの読み込み)

簡易アプリ間通信で使用するサブルーチンです。

### 説明

サーバの論理宛先からメッセージを読み込みます。なお、読み込まれたメッセージは、サーバ上から削除されます。メッセージは、優先順位の高いものから読み込まれ、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。論理宛先にメッセージが存在しない場合、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  RECEIVETYPE-REC.
    05  BUFFERLEN   PIC S9(9)  COMP-5.
    05  MSGLEN     PIC S9(9)  COMP-5.
    05  MSGTYPE    PIC S9(9)  COMP-5.
    05  WALTTIME   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_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.

      :
```

## パラメタの説明

### 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。

詳細は“[1.7.2 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。

### 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。

## 使い方

“[1.7 簡易アプリ間通信のサブルーチンの使い方](#)”を参照してください。

## 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。

- 次に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。

## I.2.15 COBCI\_WRITE(サーバへのメッセージの書込み)

簡易アプリ間通信で使用するサブルーチンです。

### 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.
```

### パラメタの説明

#### 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。

詳細は“[I.7.2 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。

#### 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。

## 使い方

“[I.7 簡易アプリ間通信のサブルーチンの使い方](#)”を参照してください。



## 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。

## I.2.16 JMPBGWDH(ウィンドウハンドル獲得サブルーチン)

### 説明

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。

### 呼出しの形式

```
CALL "JMPBGWDH" WITH C LINKAGE USING データ名-1 データ名-2
```

### パラメタの説明

#### データ名-1

```
01 データ名-1 PIC X(8).
```

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。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。

#### データ名-2

```
01 データ名-2 PIC S9(9) COMP-5.
```

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。

### 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。

値	意味
0	ウィンドウハンドルの取得に成功しました。
-1	ウィンドウハンドルの取得に失敗しました。

### メッセージ

以下の場合、メッセージが出力されます。

- ー 処理対象とする種類のファイルではない。
- ー 指定されたファイルが存在しない。

### 注意

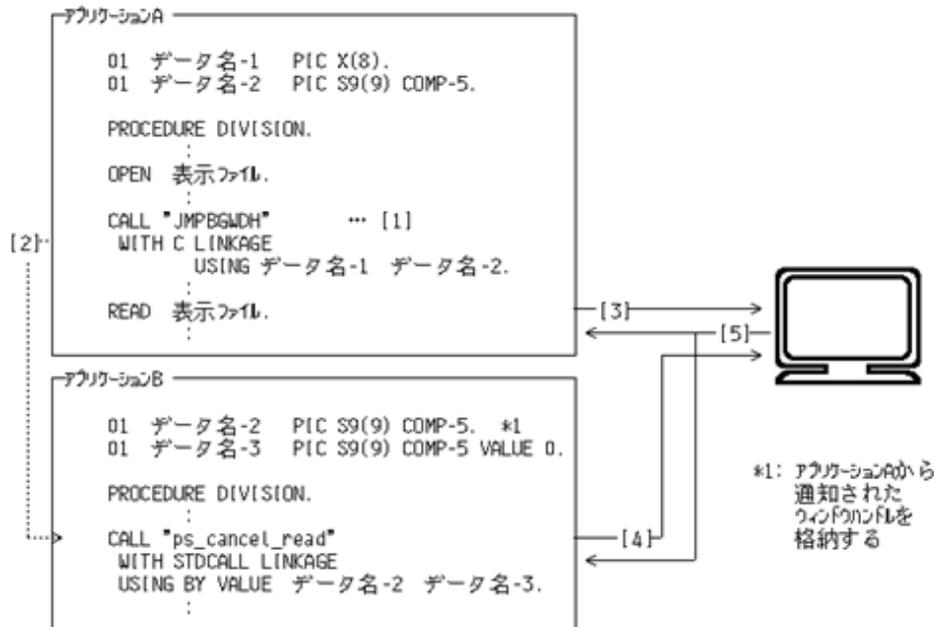
- ・ 表示ファイル単体テスト機能で動作する場合は、使用することはできません。
- ・ データ名-1に指定するファイル識別名は、実行単位内で単一のファイルに対応していなければなりません。複数のファイルに対応している場合は、実行結果は保証されません。
- ・ 動的プログラム構造で当サブルーチンを呼び出す場合、以下のようなエントリ情報が必要になります。エントリ情報の指定方法については、“[5.6 エントリ情報](#)”を参照してください。

```
[ENTRY]  
JMPBGWDH=F3BI1LNG.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ユーザーズガイド”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。

## I.2.17 JMPBWINS(インスタンスハンドル獲得サブルーチン)

### 説明

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。

### 呼出し形式

```
CALL "JMPBWINS" WITH C LINKAGE USING データ名.
```

### パラメタの説明

```
01 データ名.
02 データ名-1 PIC S9(9) COMP-5.
02 FILLER PIC X(12).
```

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、“5.6 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。

## I.2.18 #NATLETOBE(ビッグエンディアン変換サブルーチン)

### 説明

日本語項目に格納されているリトルエンディアンのデータを、ビッグエンディアンに変換します。

### 呼出し形式

```
CALL "#NATLETOBE" USING [BY REFERENCE] 一意名.
```

### パラメタの説明

一意名には、エンディアンを変換するデータ項目を指定します。集団項目が指定された場合、従属する日本語項目および日本語編集項目が変換対象になります。

### 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。

## 注意

- 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。
  - 一意名が集団項目であり、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。

## I.2.19 #NATBETOLE(リトルエンディアン変換サブルーチン)

### 説明

日本語項目に格納されているビッグエンディアンのデータを、リトルエンディアンに変換します。

### 呼出し形式

```
CALL "#NATBETOLE" USING [BY REFERENCE] 一意名.
```

### パラメタの説明

一意名には、エンディアンを変換するデータ項目を指定します。集団項目が指定された場合、従属する日本語項目および日本語編集項目が変換対象になります。

### 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。

## 注意

- 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。
  - 一意名が集団項目であり、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。

## I.3 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。

### I.3.1 COB\_GETRUNIT(COBOL実行単位ハンドル取得サブルーチン)

#### 説明

COBOLの実行単位を識別するハンドルを取得します。

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。

使用方法の詳細は、“[20.6.4 スレッド間で実行単位の資源を引き継ぐ方法](#)”を参照してください。

#### 呼出しの形式(C言語)

```
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();
```

#### パラメタの説明

呼出し時にパラメタは必要ありません。

#### 復帰コード

値	意味
COBOL実行単位のハンドル	成功。
0	失敗。

### I.3.2 COB\_SETRUNIT(COBOL実行単位ハンドル設定サブルーチン)

#### 説明

COBOLの実行単位のハンドルを呼出し元のスレッドに設定します。

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。

使用方法の詳細は、“[20.6.4 スレッド間で実行単位の資源を引き継ぐ方法](#)”を参照してください。

## 呼出しの形式(C言語)

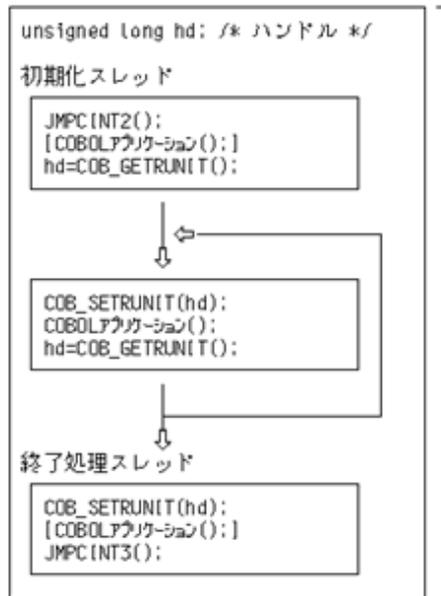
```
型宣言部 :
extern int COB_SETRUNIT(unsigned long hd);
データ宣言部 :
extern unsigned long hd;
/* COBOLの実行単位ハンドル格納域(COB_GETRUNITで取得) */
手続き部 :
COB_SETRUNIT(hd);
```

## パラメタの説明

hd:COBOL実行単位のハンドル

## 復帰コード

値	意味
0	成功
-1	失敗。入力パラメタが正しくない
-2	失敗。既にCOBOLの実行単位が存在する
-99	失敗。その他のエラー(SEに連絡してください)



※ [ ]内省略可能

## I.3.3 JMPCINT2(実行単位の開始サブルーチン)

### 説明

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。

### 呼出し形式(C言語)

```
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してください。

```
extern "C" void JMPCINT2(void);
```

### パラメタの説明

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

当サブルーチンを呼び出した場合、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“[20.3.1 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
}
```

## I.3.4 JMPCINT3(実行単位の終了サブルーチン)

### 説明

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。

### 呼出し形式(C言語)

```
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してください。

```
extern "C" void JMPCINT3(void);
```

### パラメタの説明

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

当サブルーチンを呼び出す前に、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“20.3.1 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
}
```

## I.3.5 JMPCINT4(実行環境の閉鎖サブルーチン)

### 説明

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。

### 呼出し形式(C言語)

```
型宣言部 :
extern void JMPCINT4(void);

手続き部 :
JMPCINT4();
```



### 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 JMPCINT4(void);
```

### パラメタの説明

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。

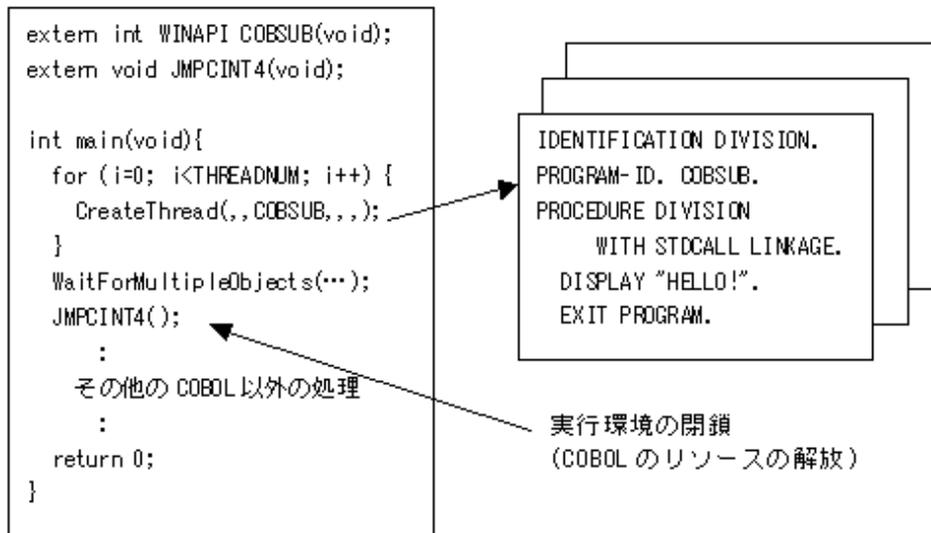
このため、当サブルーチンを呼び出す前に、JMPCINT2サブルーチン(実行単位の開始サブルーチン)を呼び出している場合は、必ずJMPCINT3サブルーチン(実行単位の閉鎖サブルーチン)を呼び出して実行単位の閉鎖を行ってください。

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“20.3.1 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



## I.4 COBOL Webサブルーチン

### I.4.1 COBW3\_ALTER\_SESSION\_TIMEOUT(セッションのタイムアウト時間の変更)

現行セッションのタイムアウト時間(秒)を変更します。

書き方

```
CALL "COBW3_ALTER_SESSION_TIMEOUT" USING COBW3.
```

呼出し時のデータ設定

COBW3-SESSION-TIMEOUT

セッションのタイムアウト時間(秒)を指定します。タイムアウト時間の計測は、“COBW3\_FREE”が呼び出された時点から、同一セッション内の次のスレッドで“COBW3\_INIT”が呼び出された時点までの最大待ち時間です。

値	意味
1~999999	指定した時間(秒)が経過した時点でタイムアウト処理が実行されます。

処理結果のデータ

なし



注意

- COBOL CGIサブルーチンでは当機能は使用できません。
- 当サブルーチンは、セッション開設中に呼び出してください。セッションの開始前およびセッションの終了後に当サブルーチンを呼び出した場合、セッションのタイムアウト時間は変更できません。

## I.4.2 COBW3\_CHECK\_VALUE\_X、COBW3\_CHECK\_VALUE\_N(Webパラメタ中のVALUEの検索)

“COBW3\_INIT”で獲得したWebパラメタから任意の値(VALUE)の検索を行います。このサブルーチンは、チェックボックスのある呼出し用ページから起動された場合に、Webブラウザでチェックされた項目を値(VALUE)で判定する場合などに使用します。各サブルーチンの意味は次のとおりです。

サブルーチン名	動作概要
COBW3_CHECK_VALUE_X	英数字文字列の値(VALUE)を検索します。
COBW3_CHECK_VALUE_N	日本語文字列の値(VALUE)を検索します。

### 書き方

```
CALL "COBW3_CHECK_VALUE_X" USING COBW3.
```

```
CALL "COBW3_CHECK_VALUE_N" USING COBW3.
```

### 呼出し時のデータ設定

COBW3-SEARCH-DATA

COBW3-SEARCH-DATA-N

検索する値(VALUE)を設定します。

COBW3\_CHECK\_VALUE\_Xの場合、COBW3-SEARCH-DATAに設定します。

COBW3\_CHECK\_VALUE\_Nの場合、COBW3-SEARCH-DATA-Nに設定します。

COBW3-SEARCH-LENGTH[省略可]

最後に有効な空白がある値(VALUE)を検索する場合、空白を含む値の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで検索します。
1~1024	指定された文字列の長さで検索します。

COBW3-NUMBER[省略可]

Webパラメタに同一のVALUEが複数存在する場合、検索対象のVALUE出現順序を設定します。

条件名	値	意味
COBW3-NUMBER-INIT	1	最初に一致する値を検索します。
—	2~9999	指定された出現順序の値を検索します。

### 処理結果のデータ

COBW3-SEARCH-FLAG

条件名	値	意味
COBW3-SEARCH-FLAG-NON	"0"	検索対象の値が存在しません。
COBW3-SEARCH-FLAG-EXIST	"1"	検索対象の値が存在しました。

## I.4.3 COBW3\_CNV\_DEL(登録した変換データの削除)

COBW3\_CNV\_SETで登録した変換データの削除を行います。

### 書き方

```
CALL "COBW3_CNV_DEL" USING COBW3.
```

#### 呼出し時のデータ設定

COBW3-CNV-NAME、COBW3-CNV-NAME-N

削除対象となる変換名を設定します。  
変換名が英数字の場合、COBW3-CNV-NAMEに設定します。  
変換名が日本語の場合、COBW3-CNV-NAME-Nに設定します。

COBW3-CNV-NAME-LENGTH[省略可]

最後に有効な空白がある変換名の場合、空白を含む値の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで登録します。ただし、COBW3-CNV-NAMEがすべて空白の場合、文字列長をゼロとして処理します。
1～30	指定された文字列の長さで登録します。

#### 処理結果のデータ

なし



#### 注意

本サブルーチンは互換を保証するために提供されています。

アプリケーションの動作コード系がシフトJISの場合にだけ、動作が保証されます。UnicodeのWebアプリケーションでは使用できないため、代替サブルーチンを使用してください。

なお、資産の移行性を高めるためには、シフトJISのWebアプリケーションでも代替サブルーチンの使用をおすすめします。

サブルーチン名	代替サブルーチン
COBW3_CNV_DEL	COBW3_DEL_CNV_X、COBW3_DEL_CNV_N

### I.4.4 COBW3\_CNV\_INIT (登録した変換データをすべて初期化)

COBW3\_CNV\_SETで登録した変換データをすべて初期化します。

#### 書き方

```
CALL "COBW3_CNV_INIT" USING COBW3.
```

#### 呼出し時のデータ設定

なし

#### 処理結果のデータ

なし



#### 注意

本サブルーチンは互換を保証するために提供されています。

アプリケーションの動作コード系がシフトJISの場合にだけ、動作が保証されます。UnicodeのWebアプリケーションでは使用できないため、代替サブルーチンを使用してください。

なお、資産の移行性を高めるためには、シフトJISのWebアプリケーションでも代替サブルーチンの使用をおすすめします。

サブルーチン名	代替サブルーチン
COBW3_CNV_INIT	COBW3_INIT_CNV

## I.4.5 COBW3\_CNV\_SET(結果出力用ページ内に指定された変換データを登録)

“COBW3\_PUT\_HTML”で出力する結果出力用ページ内に指定された変換データを登録します。

登録された情報はCOBW3\_PUT\_HTML実行時に参照され、登録されたデータに従って、結果出力用ページ内の変換が行われます。

### 注意

結果出力用ページ内に複数の異なる変換名が指定されている場合は、変換名ごとに“COBW3\_CNV\_SET”を呼び出して変換データを登録してください。

### 書き方

```
CALL "COBW3_CNV_SET" USING COBW3.
```

### 呼出し時のデータ設定

#### COBW3-CNV-NAME、COBW3-CNV-NAME-N

変換対象となる変換名を設定します。

変換名が英数字の場合、COBW3-CNV-NAMEに設定します。

変換名が日本語の場合、COBW3-CNV-NAME-Nに設定します。

#### COBW3-CNV-NAME-LENGTH[省略可]

最後に有効な空白がある変換名の場合、空白を含む値の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで登録します。ただし、COBW3-CNV-NAMEがすべて空白の場合、文字列長をゼロとして処理します。
1～30	指定された文字列の長さで登録します。

#### COBW3-CNV-VALUE、COBW3-CNV-VALUE-N

変換結果(変換文字列)を設定します。

変換文字列が英数字の場合、COBW3-CNV-VALUEに設定します。

変換文字列が日本語の場合、COBW3-CNV-VALUE-Nに設定します。

#### COBW3-CNV-VALUE-LENGTH[省略可]

最後に有効な空白がある変換名の場合、空白を含む値の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで登録します。ただし、COBW3-CNV-VALUEがすべて空白の場合、文字列長をゼロとして処理します。
1～1024	指定された文字列の長さで登録します。

#### COBW3-CNV-MODE[省略可]

変換種別を設定します。

条件名	値	意味
COBW3-CNV-MODE-ADDREP	LOW-VALUE	指定した変換名が登録されていない場合、変換データを追加します。
COBW3-CNV-MODE-REPLACE	"1"	変換データの置き換えを行います。指定した変換名が登録されていない場合、エラーを出力して変換を行いません。

条件名	値	意味
COBW3-CNV-MODE-ADD	"2"	変換データの追加を行います。指定した変換名が登録されている場合、エラーを出力して追加を行いません。

#### 処理結果のデータ

なし



#### 注意

本サブルーチンは互換を保証するために提供されています。

アプリケーションの動作コード系がシフトJISの場合にだけ、動作が保証されます。UnicodeのWebアプリケーションでは使用できないため、代替サブルーチンを使用してください。

なお、資産の移行性を高めるためには、シフトJISのWebアプリケーションでも代替サブルーチンの使用をおすすめします。

サブルーチン名	代替サブルーチン
COBW3_CNV_SET	COBW3_SET_CNV_XX、COBW3_SET_CNV_NX、COBW3_SET_CNV_XN、 COBW3_SET_CNV_NN

## I.4.6 COBW3\_DEL\_CNV\_X、COBW3\_DEL\_CNV\_N(結果出力用ページ内の変換データの削除)

COBW3\_SET\_CNV\_XXなどで登録した変換データの削除を行います。各サブルーチンの動作概要は次のとおりです。

サブルーチン名	動作概要
COBW3_DEL_CNV_X	英数字文字列の変換名に対して登録した変換データを削除します。
COBW3_DEL_CNV_N	日本語文字列の変換名に対して登録した変換データを削除します。

#### 書き方

```
CALL "COBW3_DEL_CNV_X" USING COBW3.
```

```
CALL "COBW3_DEL_CNV_N" USING COBW3.
```

#### 呼出し時のデータ設定

COBW3-CNV-NAME、COBW3-CNV-NAME-N

削除対象となる変換名を設定します。

COBW3\_DEL\_CNV\_Xの場合、COBW3-CNV-NAMEに設定します。

COBW3\_DEL\_CNV\_Nの場合、COBW3-CNV-NAME-Nに設定します。

COBW3-CNV-NAME-LENGTH[省略可]

最後に有効な空白がある変換名を指定する場合、空白を含む値の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで検索します。ただし、COBW3-CNV-NAMEまたはCOBW3-CNV-NAME-Nがすべて空白の場合、文字列長をゼロとして処理します。
1~30	指定された文字列の長さで検索します。

#### 処理結果のデータ

なし

## I.4.7 COBW3\_DEL\_COOKIE(登録したCookieデータの削除)

COBW3\_SET\_COOKIEで登録したCookieデータの削除を行います。



当サブルーチンは、COBW3\_PUT\_HTMLおよびCOBW3\_PUT\_TEXTを使用する前に使用してください。COBW3\_PUT\_HTMLまたはCOBW3\_PUT\_TEXTを一度でも使用した後に当サブルーチンを使用した場合、設定した情報は無効になります。

### 書き方

```
CALL "COBW3_DEL_COOKIE" USING COBW3.
```

### 呼出し時のデータ設定

COBW3-COOKIE-NAME、COBW3-COOKIE-NAME-N

削除するCookie名を設定します。

Cookie名が英数字の場合、COBW3-COOKIE-NAMEに設定します。

Cookie名が日本語の場合、COBW3-COOKIE-NAME-Nに設定します。

COBW3-COOKIE-NAME-LENGTH[省略可]

最後に有効な空白があるCookie名の場合、空白を含むCookie名の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで削除します。ただし、COBW3-COOKIE-NAME がすべて空白の場合、文字列長をゼロとして処理します。
1～64	指定された文字列の長さで削除します。

### 処理結果のデータ

なし



本サブルーチンは互換を保証するために提供されています。

アプリケーションの動作コード系がシフトJISの場合にだけ、動作が保証されます。UnicodeのWebアプリケーションでは使用できないため、代替サブルーチンを使用してください。

なお、資産の移行性を高めるためには、シフトJISのWebアプリケーションでも代替サブルーチンの使用をおすすめします。

サブルーチン名	代替サブルーチン
COBW3_DEL_COOKIE	COBW3_DEL_COOKIE_X、COBW3_DEL_COOKIE_N

## I.4.8 COBW3\_DEL\_COOKIE\_X、COBW3\_DEL\_COOKIE\_N(Cookieデータの削除)

COBW3\_SET\_COOKIE\_XXなどで登録したCookieデータを削除します。

各サブルーチンの動作概要は次のとおりです。

サブルーチン名	動作概要
COBW3_DEL_COOKIE_X	英数字文字列のCookie名に対応するCookieデータを削除します。
COBW3_DEL_COOKIE_N	日本語文字列のCookie名に対応するCookieデータを削除します。

## 注意

当サブルーチンは、COBW3\_PUT\_HTMLおよびCOBW3\_PUT\_TEXTを使用する前に使用してください。COBW3\_PUT\_HTMLまたはCOBW3\_PUT\_TEXTを一度でも使用した後に当サブルーチンを使用した場合、設定した情報は無効になります。

### 書き方

```
CALL "COBW3_DEL_COOKIE_X" USING COBW3.  
CALL "COBW3_DEL_COOKIE_N" USING COBW3.
```

### 呼出し時のデータ設定

COBW3-COOKIE-NAME  
COBW3-COOKIE-NAME-N

削除するCookie名を設定します。

COBW3\_DEL\_COOKIE\_Xの場合、COBW3-COOKIE-NAMEに設定します。

COBW3\_DEL\_COOKIE\_Nの場合、COBW3-COOKIE-NAME-Nに設定します。

COBW3-COOKIE-NAME-LENGTH[省略可]

最後に有効な空白があるCookie名を指定する場合、空白を含むCookie名の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで削除します。ただし、COBW3-COOKIE-NAMEまたはCOBW3-COOKIE-NAME-Nがすべて空白の場合、文字列長をゼロとして処理します。
1～64	指定された文字列の長さで削除します。

### 処理結果のデータ

なし

## I.4.9 COBW3\_DEL\_REPEAT\_X、COBW3\_DEL\_REPEAT\_N(結果出力用ページ内の繰返し変換データの削除)

COBW3\_SET\_REPEAT\_XXなどで登録した繰返し変換データの削除を行います。各サブルーチンの動作概要は次のとおりです。

サブルーチン名	動作概要
COBW3_DEL_REPEAT_X	英数字文字列の変換名に対して登録した繰返し変換データを削除します。
COBW3_DEL_REPEAT_N	日本語文字列の変換名に対して登録した繰返し変換データを削除します。

## 注意

指定された変換名に対して複数個の繰返し変換データが登録されている場合は、登録されているすべてのデータを削除します。

### 書き方

```
CALL "COBW3_DEL_REPEAT_X" USING COBW3.
```

```
CALL "COBW3_DEL_REPEAT_N" USING COBW3.
```

#### 呼出し時のデータ設定

COBW3-CNV-NAME  
COBW3-CNV-NAME-N

削除対象となる変換名を設定します。

COBW3\_DEL\_REPEAT\_Xの場合、COBW3-CNV-NAMEに設定します。

COBW3\_DEL\_REPEAT\_Nの場合、COBW3-CNV-NAME-Nに設定します。

COBW3-CNV-NAME-LENGTH[省略可]

最後に有効な空白がある変換名の場合、空白を含む値の文字列長(バイト長)を設定します。

値	意味
0	空白でない最後の文字までの長さで検索します。ただし、COBW3-CNV-NAMEまたはCOBW3-CNV-NAME-Nがすべて空白の場合、文字列長をゼロとして処理します。
1～30	指定された文字列の長さで検索します。

#### 処理結果のデータ

なし

### I.4.10 COBW3\_DEL\_UPLOADEDFILE(アップロード済みファイルの削除)

COBW3\_GEN\_UPLOADFILE\_Xなどで生成したアップロード済みのファイルを削除します。



同一リクエスト(スレッド)内で、COBW3\_GEN\_UPLOADFILE\_XまたはCOBW3\_GEN\_UPLOADFILE\_Nを使用して生成したファイルだけを削除できます。複数のリクエスト(ページ)にまたがるファイルの生成・削除はできません。

#### 書き方

```
CALL "COBW3_DEL_UPLOADEDFILE" USING COBW3.
```

#### 呼出し時のデータ設定

COBW3-UPLOADED-FILENAME

削除対象となるファイル名を設定します。

#### 処理結果のデータ

なし

### I.4.11 COBW3\_END\_SESSION(セッションの終了)

セッションを終了します。このとき、現行セッションの開始時に設定した情報を削除します。

#### 書き方

```
CALL "COBW3_END_SESSION" USING COBW3.
```

#### 呼出し時のデータ設定

なし

#### 処理結果のデータ

なし

## I.4.12 COBW3\_FREE(Web実行環境の資源の解放)

Webサブルーチンが獲得した資源を解放します。このため、当サブルーチンはWebアプリケーションの各接続の最後で必ず呼び出してください。

書き方

```
CALL "COBW3_FREE" USING COBW3.
```

呼出し時のデータ設定

なし

処理結果のデータ

なし



注意

- COBOL CGIサブルーチンでは当機能は使用できません。
- 当サブルーチンは、“COBW3\_PUT\_TEXT”、“COBW3\_PUT\_HTML”および“COBW3\_FREE”を使用する前に呼び出してください。これらのサブルーチンを使用した後に当サブルーチンを呼び出した場合、セッションは終了できません。既にセッションが終了されている場合、当サブルーチンを呼び出すとエラーが通知されます。

## I.4.13 COBW3\_INIT(作業環境の設定およびWebパラメタの獲得)

Webサブルーチン作業環境の設定を行います。また、Webパラメタを獲得し、Webサブルーチンが使用する作業領域に設定します。

書き方

```
CALL "COBW3_INIT" USING COBW3.
```

呼出し時のデータ設定

COBW3-CONTEXT[ISAPIサブルーチンの場合は必須]

IISディレクトタとのインタフェース領域へのポインタを指定します。指定した値を変更してはなりません。

COBW3-DMODE[省略可]

Webサブルーチンのエラーメッセージの出力を指定します。

なお、COBW3-DMODEは動作コード系がUnicodeの場合には無効です。

条件名	値	意味
COBW3-DMODE-NODBG	LOW-VALUE	エラーメッセージを出力しません。
COBW3-DMODE-DBG	"1"	ブラウザにエラーメッセージを出力します。

処理結果のデータ

なし



注意

- “COBW3\_INIT”を呼び出す前にインタフェース領域のCOBW3をLOW-VALUEで初期化してください。
- “COBW3\_INIT”を続けて2回以上呼び出すと、獲得したWebパラメタが初期化される場合があり、その後の動作は保証されません。したがって、“COBW3\_INIT”の呼び出しは、Webブラウザと接続するごとに他のWebサブルーチンを呼び出す前までの間に一度だけ実行されるようにプログラムを作成してください。

## I.4.14 COBW3\_INIT\_CNV(結果出力用ページ内の変換データの初期化)

COBW3\_SET\_CNV\_XXなどで登録した変換データをすべて初期化します。

書き方

```
CALL "COBW3_INIT_CNV" USING COBW3.
```

呼出し時のデータ設定

なし

処理結果のデータ

なし

## I.4.15 COBW3\_INIT\_COOKIE(登録済みの全Cookieデータの初期化)

COBW3\_SET\_COOKIE\_XXなどで登録したCookieデータをすべて初期化します。



当サブルーチンは、COBW3\_PUT\_HTMLおよびCOBW3\_PUT\_TEXTを使用する前に使用してください。COBW3\_PUT\_HTMLまたはCOBW3\_PUT\_TEXTを一度でも使用した後に当サブルーチンを使用した場合、設定した情報は無効になります。

書き方

```
CALL "COBW3_INIT_COOKIE" USING COBW3.
```

呼出し時のデータ設定

COBW3-COOKIE-INIT-MODE[省略可]

Cookieデータの初期化種別を設定します。

条件名	値	意味
COBW3-COOKIE-INIT-MODE-NORMAL	LOW-VALUE	すべてのCookieデータを初期化します。
COBW3-COOKIE-INIT-MODE-REQUEST	"1"	リクエストデータのCookieデータを初期値として初期化します。

処理結果のデータ

なし

## I.4.16 COBW3\_INIT\_REPEAT(結果出力用ページ内の繰返し変換データの初期化)

COBW3\_SET\_REPEAT\_XXなどで登録した繰返し変換データをすべて初期化します。

書き方

```
CALL "COBW3_INIT_REPEAT" USING COBW3.
```

呼出し時のデータ設定

なし

処理結果のデータ

なし

## I.4.17 COBW3\_GEN\_UPLOADFILE\_X、COBW3\_GEN\_UPLOADFILE\_N(アップロードされたファイルをサーバ上に生成)

アップロードされたファイルを、指定された名前で生成します。

各サブルーチンの動作概要は次のとおりです。

サブルーチン名	動作概要
COBW3_GEN_UPLOADFILE_X	英数字文字列の名前(NAME)を検索し、対応するアップロードされたファイルを指定された名前で生成します。
COBW3_GEN_UPLOADFILE_N	日本語文字列の名前(NAME)を検索し、対応するアップロードされたファイルを指定された名前で生成します。

### 注意

セキュリティ上危険なため、アップロードされたファイルを実行可能なファイルとして生成しないでください。また、システムが自動実行するようなファイルとして生成しないでください。

### 書き方

```
CALL "COBW3_GEN_UPLOADFILE_X" USING COBW3.
```

```
CALL "COBW3_GEN_UPLOADFILE_N" USING COBW3.
```

### 呼出し時のデータ設定

COBW3-SEARCH-DATA  
COBW3-SEARCH-DATA-N

検索するアップロードファイルの名前(NAME)を設定します(名前はファイル名ではなく、呼出し用ページのHTML文書のNAMEに指定した値)。

COBW3\_GEN\_UPLOADFILE\_Xの場合、COBW3-SEARCH-DATAに設定します。

COBW3\_GEN\_UPLOADFILE\_Nの場合、COBW3-SEARCH-DATA-Nに設定します。

### COBW3-SEARCH-LENGTH[省略可]

最後に有効な空白がある名前の場合、空白を含む名前の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで検索します。
1~1024	指定された文字列の長さで検索します。

### COBW3-NUMBER[省略可]

同一の名前(NAME)が複数存在する場合、検索対象の名前の出現順序を設定します。

条件名	値	意味
COBW3-NUMBER-INIT	1	最初に一致する名前を検索します。
—	2~9999	指定された出現順序の名前を検索します。

### COBW3-UPLOADED-FILENAME

サーバ上で実際に生成するファイル名を設定します。

### 処理結果のデータ

COBW3-SEARCH-FLAG

条件名	値	意味
COBW3-SEARCH-FLAG-NON	"0"	検索対象の名前が存在しません。または、ファイルのサイズが0です。
COBW3-SEARCH-FLAG-EXIST	"1"	検索対象の名前が存在しました。

## I.4.18 COBW3\_GET\_AUTHORIZE(認証情報の取得)

Webブラウザからの認証情報を取得します。認証要求に対して渡されたユーザID、パスワードおよびクライアントのIPアドレスを取得します。

書き方

```
CALL "COBW3_GET_AUTHORIZE" USING COBW3.
```

呼出し時のデータ設定

なし

処理結果のデータ

COBW3-USERID

取得したユーザIDが設定されます。

COBW3-USERID-LENGTH

ユーザIDの文字列長(バイト長)が設定されます。

COBW3-PASSWORD

取得したユーザパスワードが設定されます。

COBW3-PASSWORD-LENGTH

ユーザパスワードの文字列長(バイト長)が設定されます。

COBW3-IP-ADDRESS

取得したクライアントのIPアドレスが設定されます。

COBW3-IP-ADDRESS-LENGTH

IPアドレスの文字列長(バイト長)が設定されます。

COBW3-AUTH-TYPE

取得した認証タイプが設定されます。

COBW3-AUTH-TYPE-LENGTH

認証タイプの文字列長(バイト長)が設定されます。



注意

CGIサブルーチンおよびISAPIサブルーチンの場合

基本認証ではすべての情報を取得することができ、認証タイプには“Basic”が設定されます。匿名認証の場合、IPアドレス以外の情報は取得できません。Microsoft暗号化認証では、ユーザパスワード以外の情報すべてが取得可能で、認証タイプには“NTLM”が設定されます。

## I.4.19 COBW3\_GET\_COOKIE、COBW3\_GET\_COOKIEEN(リクエストCookieデータから任意の名前の検索)

“COBW3\_INIT”で獲得したリクエストCookieデータから任意の名前(NAME)の検索を行います。

また、名前(NAME)に対応する値(VALUE)を取得します。取得した値(VALUE)を格納する際のパディング文字列(埋草文字)は次のとおりです。

サブルーチン名	取得した値(VALUE)を格納する際のパディング文字列(埋草文字)
COBW3_GET_COOKIEの場合	半角空白
COBW3_GET_COOKIEENの場合	全角空白

#### 書き方

```
CALL "COBW3_GET_COOKIE" USING COBW3.
```

```
CALL "COBW3_GET_COOKIEEN" USING COBW3.
```

#### 呼出し時のデータ設定

##### COBW3-COOKIE-NAME、COBW3-COOKIE-NAME-N

検索するCookie名を設定します。

英数字文字列を検索する場合、COBW3-COOKIE-NAMEに設定します。

日本語文字列を検索する場合、COBW3-COOKIE-NAME-Nに設定します。

##### COBW3-COOKIE-NAME-LENGTH[省略可]

最後に有効な空白があるCookie名の場合、空白を含むCookie名の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで検索します。
1～64	指定された文字列の長さで検索します。

#### 処理結果のデータ

##### COBW3-SEARCH-FLAG

条件名	値	意味
COBW3-SEARCH-FLAG-NON	"0"	検索対象の名前が存在しません。
COBW3-SEARCH-FLAG-EXIST	"1"	検索対象の名前が存在しました。

##### COBW3-COOKIE-VALUE、COBW3-COOKIE-VALUE-N

検索対象のCookie名に対応するCookie値が設定されます。

COBW3\_GET\_COOKIEの場合、COBW3-COOKIE-VALUEに設定されます。

COBW3\_GET\_COOKIEENの場合、COBW3-COOKIE-VALUE-Nに設定されます。

##### COBW3-COOKIE-VALUE-LENGTH

検索対象のCookie名に対応するCookie値の文字列長(バイト長)が設定されます。



#### 注意

本サブルーチンは互換を保証するために提供されています。

アプリケーションの動作コード系がシフトJISの場合にだけ、動作が保証されます。UnicodeのWebアプリケーションでは使用できないため、代替サブルーチンを使用してください。

なお、資産の移行性を高めるためには、シフトJISのWebアプリケーションでも代替サブルーチンの使用をおすすめします。

サブルーチン名	代替サブルーチン
COBW3_GET_COOKIE	COBW3_GET_COOKIE_XX、COBW3_GET_COOKIE_NX
COBW3_GET_COOKIEEN	COBW3_GET_COOKIE_XN、COBW3_GET_COOKIE_NN

## I.4.20 COBW3\_GET\_COOKIE\_XX、COBW3\_GET\_COOKIE\_NX、COBW3\_GET\_COOKIE\_XN、COBW3\_GET\_COOKIE\_NN (WebパラメタからCookieデータの取得)

“COBW3\_INIT”で獲得したリクエストCookieデータから任意のCookie名の検索を行います。また、Cookie名に対応するCookie値を取得します。

各サブルーチンの動作概要および取得したCookie値を格納する際のパディング文字列(埋草文字)は、次のとおりです。

サブルーチン名	動作概要	取得したCookie値を格納する際のパディング文字列(埋草文字)
COBW3_GET_COOKIE_XX	英数字文字列のCookie名を検索し、対応するCookie値を英数字文字列として返します。	半角空白
COBW3_GET_COOKIE_NX	日本語文字列のCookie名を検索し、対応するCookie値を英数字文字列として返します。	半角空白
COBW3_GET_COOKIE_XN	英数字文字列のCookie名を検索し、対応するCookie値を日本語文字列として返します。	全角空白
COBW3_GET_COOKIE_NN	日本語文字列のCookie名を検索し、対応するCookie値を日本語文字列として返します。	全角空白

### 書き方

```
CALL "COBW3_GET_COOKIE_XX" USING COBW3.
```

```
CALL "COBW3_GET_COOKIE_NX" USING COBW3.
```

```
CALL "COBW3_GET_COOKIE_XN" USING COBW3.
```

```
CALL "COBW3_GET_COOKIE_NN" USING COBW3.
```

### 呼出し時のデータ設定

COBW3-COOKIE-NAME

COBW3-COOKIE-NAME-N

検索するCookie名を設定します。

COBW3\_GET\_COOKIE\_XXまたはCOBW3\_GET\_COOKIE\_XNの場合、COBW3-COOKIE-NAMEに設定します。

COBW3\_GET\_COOKIE\_NXまたはCOBW3\_GET\_COOKIE\_NNの場合、COBW3-COOKIE-NAME-Nに設定します。

COBW3-COOKIE-NAME-LENGTH[省略可]

最後に有効な空白があるCookie名を検索する場合、空白を含むCookie名の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで検索します。ただし、COBW3-COOKIE-NAMEまたはCOBW3-COOKIE-NAME-Nがすべて空白の場合、文字列長をゼロとして処理します。
1～64	指定された文字列の長さで検索します。

### 処理結果のデータ

COBW3-SEARCH-FLAG

条件名	値	意味
COBW3-SEARCH-FLAG-NON	"0"	検索対象の名前が存在しません。
COBW3-SEARCH-FLAG-EXIST	"1"	検索対象の名前が存在しました。

## COBW3-COOKIE-VALUE、COBW3-COOKIE-VALUE-N

検索対象のCookie名に対応するCookie値が設定されます。

COBW3\_GET\_COOKIE\_XXまたはCOBW3\_GET\_COOKIE\_NXの場合、COBW3-COOKIE-VALUEに設定されます。

COBW3\_GET\_COOKIE\_XNまたはCOBW3\_GET\_COOKIE\_NNの場合、COBW3-COOKIE-VALUE-Nに設定されます。

## COBW3-COOKIE-VALUE-LENGTH

検索対象のCookie名に対応するCookie値の文字列長(バイト長)が設定されます。

## I.4.21 COBW3\_GET\_REQUEST\_INFO(リクエストに関する情報の取得)

リクエストに関する各種情報を取得します。

### 書き方

```
CALL "COBW3_GET_REQUEST_INFO" USING COBW3.
```

### 呼出し時のデータ設定

#### COBW3-REQUEST-INFO-TYPE

取得する情報の種類を指定します。

条件名	値	意味
COBW3-URI	LOW-VALUE	リクエストURI
COBW3-URL	"1"	リクエストURL
COBW3-SERVER-TYPE (COBW3-LISTENERTYPE)	"2"	サーバの種類
COBW3-VIRTUALPATH	"3"	リクエストの仮想パス名
COBW3-PHYSICALPATH	"4"	リクエストの仮想パスに対応する物理パス名
COBW3-QUERYSTRING	"5"	Webパラメタ
COBW3-LANGUAGE	"6"	Webブラウザが受け取ることのできる言語のリスト
COBW3-ENCODING	"7"	Webブラウザが受け取ることのできるエンコーディングのリスト
COBW3-REQMIMETYPE	"8"	リクエストのMIMEタイプ
COBW3-REQUEST-METHOD	"9"	リクエストメソッド
COBW3-PATH-INFO	"A"	Webブラウザから送信される拡張パス名
COBW3-PATH-TRANSLATED	"B"	拡張パスに対応する物理パス名
COBW3-REMOTE-ADDR	"C"	クライアントのIPアドレス
COBW3-REMOTE-HOST	"D"	クライアントのホスト名
COBW3-GATEWAY-INTERFACE	"E"	サポートCGIのリビジョン
COBW3-SERVER-NAME	"F"	サーバのホスト名
COBW3-SERVER-PORT	"G"	サーバの接続ポート番号
COBW3-SERVER-PROTOCOL	"H"	サーバの使用しているプロトコル
COBW3-SERVER-PORT-SECURE	"I"	セキュア通信が行われている場合は"1"、それ以外は"0"を返す



CGIサブルーチンの場合

- リクエストのMIMEタイプは取得できません。
- クライアントのホスト名は、IPアドレスが返却される場合があります。
- セキュア通信情報の取得の可否は、Webサーバによって異なります。

#### ISAPIサブルーチンの場合

- リクエストのMIMEタイプは取得できません。

#### 処理結果のデータ

##### COBW3-REQUEST-INFO

取得したリクエスト情報が設定されます。

##### COBW3-REQUEST-INFO-LENGTH

取得したリクエスト情報の文字列長(バイト長)が設定されます。

## I.4.22 COBW3\_GET\_SESSION\_DATA(セッションデータの取得)

現行セッションに登録されているセッションデータを取得します。

#### 書き方

```
CALL "COBW3_GET_SESSION_DATA" USING COBW3 セッションデータ.
```

#### 呼出し時のデータ設定

##### COBW3-SESSION-DATA-SIZE

取得したセッションデータを格納する集団項目長(バイト長)を指定します。セッションを開始した時、セッションデータの形式にCOBW3-SESSION-DATA-GROUPITEMが設定されている場合だけ有効です。

値	意味
1~999999999	指定されたセッションデータの長さで取得します。セッションに登録されているセッションデータと異なる場合は、エラーになります。

#### セッションデータ

取得するセッションデータを格納するデータ項目を指定します。セッションを開始した時のセッションデータの形式に合わせて、集団項目またはオブジェクト参照項目を指定します。

#### 注意

- セッションデータにオブジェクト参照項目を登録している場合、当サブルーチンの呼び出しパラメタに指定するオブジェクト参照項目は、NULLオブジェクトで初期化されていなければなりません。NULLオブジェクトで初期化されていない場合、セッションデータは取得できません。
- なお、セッションデータにオブジェクト参照項目を指定する場合は、他の選択指定のないUSAGE OBJECT REFERENCE句が記述されたオブジェクト参照項目を指定してください。
- 取得するセッションデータがセッション開始時に設定したセッションデータの形式と異なる場合、Webアプリケーションの動作は保証できません。
- セッションデータに集団項目を指定する場合、従属する基本項目の構成を登録されているセッションデータの構成と一致させるために、登録集の利用をおすすめします。

#### 処理結果のデータ

なし

## 注意

- COBOL CGIサブルーチンでは当機能は使用できません。
- 当サブルーチンは、セッション開設中に呼び出してください。セッションの開始前およびセッションの終了後に当サブルーチンを呼び出した場合、セッションデータは取得できません。  
セッション開設中はセッションデータの形式を変更することはできません。  
“COBW3\_START\_SESSION”の呼び出し時に設定したセッションデータの形式に合わせたセッションデータを取得してください。

### I.4.23 COBW3\_GET\_SESSION\_INFO(セッション情報の取得)

現行セッションの情報を取得します。

#### 書き方

```
CALL "COBW3_GET_SESSION_INFO" USING COBW3.
```

#### 呼び出し時のデータ設定

なし

#### 処理結果のデータ

##### COBW3-SESSION-TIMEOUT

現行セッションのタイムアウト時間(秒)が設定されます。

##### COBW3-SESSION-ID

現行セッションのセッションIDを表わす数字が設定されます。

##### COBW3-SESSION-DATA-SIZE

現行セッションのセッションデータ長(バイト長)が設定されます。セッションを開始した時、セッションデータの形式にCOBW3-SESSION-DATA-GROUPITEMが設定されている場合だけ取得できます。セッションデータの形式にCOBW3-SESSION-DATA-OBJECTが設定されている場合は、ゼロが設定されます。

##### COBW3-SESSION-DATA-TYPE

現行セッションのセッションデータの形式が設定されます。

条件名	値	意味
COBW3-SESSION-DATA-GROUPITEM	"0"	セッションデータに集団項目を使用しています。
COBW3-SESSION-DATA-OBJECT	"1"	セッションデータにオブジェクト参照項目を使用しています。

##### COBW3-SESSION-STATUS

現行セッションの状態が設定されます。

条件名	値	意味
COBW3-SESSION-STATUS-NON	"0"	セッションは開始されていないか、終了しています。
COBW3-SESSION-STATUS-STARTED	"1"	セッションは開始されています。

## 注意

COBOL CGIサブルーチンでは当機能は使用できません。

## I.4.24 COBW3\_GET\_UPLOADFILE\_INFO\_X、 COBW3\_GET\_UPLOADFILE\_INFO\_N(アップロードファイル情報の取得)

アップロードされたファイルに関する情報を取得します。各サブルーチンの動作概要は次のとおりです。

サブルーチン名	動作概要
COBW3_GET_UPLOADFILE_INFO_X	英数字文字列の名前(NAME)を検索し、対応するアップロードされたファイルの情報を取得します。
COBW3_GET_UPLOADFILE_INFO_N	日本語文字列の名前(NAME)を検索し、対応するアップロードされたファイルの情報を取得します。

### 書き方

```
CALL "COBW3_GET_UPLOADFILE_INFO_X" USING COBW3.
```

```
CALL "COBW3_GET_UPLOADFILE_INFO_N" USING COBW3.
```

### 呼出し時のデータ設定

#### COBW3-SEARCH-DATA、COBW3-SEARCH-DATA-N

検索するアップロードファイルの名前(NAME)を設定します(名前はファイル名ではなく、呼出し用ページのHTML文書のNAMEに指定した値)。

COBW3\_GET\_UPLOADFILE\_INFO\_Xの場合、COBW3-SEARCH-DATAに設定します。

COBW3\_GET\_UPLOADFILE\_INFO\_Nの場合、COBW3-SEARCH-DATA-Nに設定します。

#### COBW3-SEARCH-LENGTH[省略可]

最後に有効な空白がある名前の場合、空白を含む名前の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで検索します。
1~1024	指定された文字列の長さで検索します。

#### COBW3-NUMBER[省略可]

同一の名前(NAME)が複数存在する場合、検索対象の名前の出現順序を設定します。

条件名	値	意味
COBW3-NUMBER-INIT	1	最初に一致する名前を検索します。
—	2~9999	指定された出現順序の名前を検索します。

### 処理結果のデータ

#### COBW3-SEARCH-FLAG

条件名	値	意味
COBW3-SEARCH-FLAG-NON	"0"	検索対象の名前が存在しません。または、ファイルのサイズが0です。
COBW3-SEARCH-FLAG-EXIST	"1"	検索対象の名前が存在しました。

#### COBW3-UPLD-CL-FILE-PATH

検索対象の名前(NAME)に対応するアップロードファイルのクライアントでのパス名が設定されます。

#### COBW3-UPLD-CL-FILE-PATH-LENGTH

検索対象の名前(NAME)に対応するアップロードファイルのクライアントでのパス名の文字列長(バイト長)が設定されます。

#### COBW3-UPLD-CL-FILE-NAME

検索対象の名前(NAME)に対応するアップロードファイルのクライアントでのファイル名が設定されます。

#### COBW3-UPLD-CL-FILE-NAME-LENGTH

検索対象の名前(NAME)に対応するアップロードファイルのクライアントでのファイル名の文字列長(バイト長)が設定されます。

#### COBW3-UPLD-CONTENT-TYPE

検索対象の名前(NAME)に対応するアップロードファイルのコンテンツタイプが文字列で設定されます。

#### COBW3-UPLD-CONTENT-TYPE-LENGTH

検索対象の名前(NAME)に対応するアップロードファイルのコンテンツタイプの文字列長(バイト長)が設定されます。

#### COBW3-UPLD-FILE-SIZE

検索対象の名前(NAME)に対応するアップロードファイルのファイルサイズ(バイト)が設定されます。



#### 注意

クライアントがUNIX系システムの場合、ファイル名に¥が含まれていると、ファイル名およびパス名を正しく受け取ることができません。



#### 例

- c.htm(呼出し用ページ)

```
      :  
<FORM METHOD="POST" ENCTYPE="multipart/form-data"  
      ACTION="sample/action.script">  
<P>  
氏名 : <INPUT TYPE="text" NAME="NAME1"><BR>  
送信ファイル1 : <INPUT TYPE="file" NAME="FILE1"><BR>  
<INPUT TYPE="submit" VALUE="送信">  
<INPUT TYPE="reset" VALUE="取消">  
</FORM>  
      :
```

- COBOLプログラム

```
      :  
* 名前の設定  
  MOVE "FILE1" TO COBW3-SEARCH-DATA.  
* アップロードファイル情報の取得  
  CALL "COBW3_GET_UPLOADFILE_INFO_X" USING COBW3.  
      :  
  IF COBW3-SEARCH-FLAG-EXIST THEN  
* 生成するファイル名の設定  
  MOVE "d.tmp" TO COBW3-UPLOADED-FILENAME  
* ファイルの生成  
  CALL "COBW3_GEN_UPLOADFILE_X" USING COBW3  
      :  
* ファイルデータの処理  
      :  
* 生成したファイルの削除  
  CALL "COBW3_DEL_UPLOADEDFILE" USING COBW3  
  END-IF.  
      :
```

## I.4.25 COBW3\_GET\_VALUE\_XX、COBW3\_GET\_VALUE\_NX、COBW3\_GET\_VALUE\_XN、COBW3\_GET\_VALUE\_NN(Webパラメタ中のNAMEの検索およびVALUEの取得)

“COBW3\_INIT”で獲得したWebパラメタから任意の名前(NAME)の検索を行い、対応する値(VALUE)を取得します。各サブルーチンの動作概要および取得した値(VALUE)を格納する際のパディング文字列(埋草文字)は、次のとおりです。

サブルーチン名	動作概要	取得した値(VALUE)を格納する際のパディング文字列(埋草文字)
COBW3_GET_VALUE_XX	英数字文字列の名前(NAME)を検索し、対応する値(VALUE)を英数字文字列として返します。	半角空白
COBW3_GET_VALUE_NX	日本語文字列の名前(NAME)を検索し、対応する値(VALUE)を英数字文字列として返します。	半角空白
COBW3_GET_VALUE_XN	英数字文字列の名前(NAME)を検索し、対応する値(VALUE)を日本語文字列として返します。	全角空白
COBW3_GET_VALUE_NN	日本語文字列の名前(NAME)を検索し、対応する値(VALUE)を日本語文字列として返します。	全角空白

### 書き方

```
CALL "COBW3_GET_VALUE_XX" USING COBW3.
```

```
CALL "COBW3_GET_VALUE_NX" USING COBW3.
```

```
CALL "COBW3_GET_VALUE_XN" USING COBW3.
```

```
CALL "COBW3_GET_VALUE_NN" USING COBW3.
```

### 呼出し時のデータ設定

COBW3-SEARCH-DATA

COBW3-SEARCH-DATA-N

検索する名前(NAME)を設定します。(名前は呼出し用ページのHTML文書のNAMEに指定した値)

COBW3\_GET\_VALUE\_XXまたはCOBW3\_GET\_VALUE\_XNの場合、COBW3-SEARCH-DATAに設定します。

COBW3\_GET\_VALUE\_NXまたはCOBW3\_GET\_VALUE\_NNの場合、COBW3-SEARCH-DATA-Nに設定します。

COBW3-SEARCH-LENGTH[省略可]

最後に有効な空白がある名前(NAME)を検索する場合、空白を含む名前の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで検索します。
1~1024	指定された文字列の長さで検索します。

COBW3-NUMBER[省略可]

Webパラメタに同一の名前(NAME)が複数存在する場合、検索対象の名前の出現順序を設定します。

条件名	値	意味
COBW3-NUMBER-INIT	1	最初に一致する名前を検索します。
—	2~9999	指定された出現順序の名前を検索します。

### 処理結果のデータ

COBW3-SEARCH-FLAG

条件名	値	意味
COBW3-SEARCH-FLAG-NON	"0"	検索対象の名前が存在しません。
COBW3-SEARCH-FLAG-EXIST	"1"	検索対象の名前が存在しました。

#### COBW3-GET-DATA、COBW3-GET-DATA-N

検索対象の名前(NAME)に対応する値(VALUE)が設定されます。  
COBW3\_GET\_VALUE\_XXまたはCOBW3\_GET\_VALUE\_NXの場合、COBW3-GET-DATAに設定されます。  
COBW3\_GET\_VALUE\_XNまたはCOBW3\_GET\_VALUE\_NNの場合、COBW3-GET-DATA-Nに設定されます。

#### COBW3-GET-LENGTH

検索対象の名前(NAME)に対応する値(VALUE)の文字列長(バイト長)が設定されます。

## I.4.26 COBW3\_NAME、COBW3\_NAMEN(Webパラメタから任意の名前を検索)

“COBW3\_INIT”で獲得したWebパラメタから任意の名前(NAME)を検索します。

また、名前(NAME)に対応する値(VALUE)を取得します。取得した値(VALUE)を格納する際のパディング文字列(埋草文字)は次のとおりです。

サブルーチン名	取得した値(VALUE)を格納する際のパディング文字列(埋草文字)
COBW3_NAME	半角空白
COBW3_NAMEN	全角空白

#### 書き方

```
CALL "COBW3_NAME" USING COBW3.
```

```
CALL "COBW3_NAMEN" USING COBW3.
```

#### 呼出し時のデータ設定

##### COBW3-SEARCH-DATA、COBW3-SEARCH-DATA-N

検索する名前(NAME)を設定します。(名前は呼出し用ページのHTML文書のNAMEに指定した値)  
英数字文字列を検索する場合、COBW3-SEARCH-DATAに設定します。  
日本語文字列を検索する場合、COBW3-SEARCH-DATA-Nに設定します。

##### COBW3-SEARCH-LENGTH[省略可]

最後に有効な空白がある名前(NAME)の場合、空白を含む名前の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで検索します。
1～1024	指定された文字列の長さで検索します。

##### COBW3-NUMBER[省略可]

Webパラメタに同一の名前(NAME)が複数存在する場合、検索対象の名前の出現順序を設定します。

条件名	値	意味
COBW3-NUMBER-INIT	1	最初に一致する名前を検索します。
—	2～9999	指定された出現順序の名前を検索します。

#### 処理結果のデータ

##### COBW3-SEARCH-FLAG

条件名	値	意味
COBW3-SEARCH-FLAG-NON	"0"	検索対象の名前が存在しません。
COBW3-SEARCH-FLAG-EXIST	"1"	検索対象の名前が存在しました。

#### COBW3-GET-DATA、COBW3-GET-DATA-N

検索対象のNAMEに対応するVALUEが設定されます。  
 COBW3\_NAMEの場合、COBW3-GET-DATAに設定されます。  
 COBW3\_NAMENの場合、COBW3-GET-DATA-Nに設定されます。

#### COBW3-GET-LENGTH

検索対象のNAMEに対応するVALUEの文字列長(バイト長)が設定されます。



本サブルーチンは互換を保証するために提供されています。

アプリケーションの動作コード系がシフトJISの場合にだけ、動作が保証されます。UnicodeのWebアプリケーションでは使用できないため、代替サブルーチンを使用してください。

なお、資産の移行性を高めるためには、シフトJISのWebアプリケーションでも代替サブルーチンの使用をおすすめします。

サブルーチン名	代替サブルーチン
COBW3_NAME	COBW3_GET_VALUE_XX、COBW3_GET_VALUE_NX
COBW3_NAMEN	COBW3_GET_VALUE_XN、COBW3_GET_VALUE_NN

## I.4.27 COBW3\_PUT\_HEAD(ヘッダの出力)

指定された任意のヘッダを出力します。

結果出力用ページのHTML文書を出力する場合、Webサブルーチンがヘッダをtext/htmlとして出力するため、当サブルーチンを使用してヘッダを出力する必要はありません。HTML文書以外のファイル(プレーンテキストなど)を出力する場合または任意のヘッダをWebサーバに出力する場合にだけ当サブルーチンを使用してください。



当サブルーチンは、COBW3\_PUT\_HTMLおよびCOBW3\_PUT\_TEXTを使用する前に使用してください。COBW3\_PUT\_HTMLまたはCOBW3\_PUT\_TEXTを一度でも使用した後に当サブルーチンを使用した場合、設定した情報はヘッダとして無効になります。

また、COBW3-CONTENT-TYPE、COBW3-STATUS-CODEに一度省略値以外の値を設定した後、値は変えないでください。

#### 書き方

```
CALL "COBW3_PUT_HEAD" USING COBW3.
```

#### 呼出し時のデータ設定

##### COBW3-PUT-HEAD

ヘッダに出力する文字列を設定します。

##### COBW3-PUT-HEAD-LENGTH[省略可]

最後に有効な空白がある文字列をヘッダに出力する場合、空白を含む文字列の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで出力します。ただし、COBW3-PUT-HEADがすべて空白の場合、文字列長をゼロ(改行だけ)として処理します。

値	意味
1～512	指定された文字列の長さで出力します。

#### COBW3-CONTENT-TYPE[省略可]

レスポンスデータのファイルタイプ(属性)(Content-type)を設定します。  
当サブルーチンが複数回呼び出された場合、最初の指定が有効になります。

条件名	値	意味
—	LOW-VALUE	HTML文書を出力します。
COBW3-CONTENT-TYPE-NON	HIGH-VALUE	Content-typeを出力しません。
COBW3-CONTENT-TYPE-HTML	"text/html"	HTML文書を出力します。
COBW3-CONTENT-TYPE-TEXT	"text/plain"	テキストファイルを出力します。
—	任意の文字列	任意の文字列をヘッダのContent-typeに指定します。

#### COBW3-STATUS-CODE[省略可]

ステータスコード(Status-code)を設定します。  
当サブルーチンが複数回呼び出された場合、最初の指定が有効になります。

条件名	値	意味
—	LOW-VALUE	正常終了のコード"200"をStatus-codeに出力します。
COBW3-STATUS-CODE-NON	HIGH-VALUE	Status-codeを出力しません。
COBW3-STATUS-CODE-200	"200"	正常終了のコード"200"をStatus-codeに出力します。
—	任意のコード	任意のStatus-codeを出力します。



#### 注意

COBW3-CONTENT-TYPE-NONおよびCOBW3-STATUS-CODE-NONは、極力指定しないでください。CGIなどでは、バッチファイルからContent-typeなどを指定することが可能なため、-NONの指定は有効です。しかし、ISAのようにDLLアプリケーションでは外部からの指定ができません。また、ISAで、-NONを指定してContent-typeやStatus-codeを指定しなかった場合の動作は不定です。

## I.4.28 COBW3\_PUT\_HTML(結果出力用ページの出力)

Webアプリケーション実行後に、結果出力用ページ(HTML文書)をWebブラウザに出力します。このとき、出力する結果出力用ページに“//COBOL//”または“//COBOL\_REPEAT//”で囲まれた変換名がある場合に、事前にCOBW3\_SET\_CNV\_XX、COBW3\_SET\_REPEAT\_XXなどで登録された変換文字列に変換します。

#### 書き方

```
CALL "COBW3_PUT_HTML" USING COBW3.
```

#### 呼出し時のデータ設定

##### COBW3-HTML-FILENAME

結果出力用ページのファイル名を設定します。

#### 処理結果のデータ

なし

## 結果出力用ページ内の変換名の指定方法

結果出力用ページ内の文字列で、Webアプリケーションの実行時に変換して出力したい箇所に、前後を“//COBOL//”で囲んだ文字列(変換名)を記述します。

```
... //COBOL//変換名//COBOL// ...
```

### 注意

以下のように“//COBOL//”は複数行にまたがって指定することはできません。

```
... //COBOL//変換名
変換名の続き//COBOL// ...
```

変換名の指定方法ごとの処理内容を以下に示します。

変換名の指定方法	処理内容
This is a //COBOL//NAME//COBOL// test.	変換名"NAME"が登録されていれば、“//COBOL//NAME//COBOL//”を登録されている変換文字列に置き換えます。
This is a //COBOL// test.	変換名の指定に誤りがあるため、エラーとなります。
This is a //COBOL test.	そのまま出力されます。
This is a //COBOL//NAME//COBOL// test.	変換名の指定に誤りがあるため、エラーとなります。
This is a //COBOL//NAME//COBOL	変換名の指定に誤りがあるため、エラーとなります。
This is a //COBOL////COBOL//	変換名の指定に誤りがあるため、エラーとなります。
This is a //cobol//NAME//cobol//.	そのまま出力されます。

### 例

— a.htm(結果出力用ページ)

```

:
<BODY>
  <BR>
  名前: //COBOL//GET-NAME//COBOL//
  <BR>
:

```

— COBOLプログラム

```

:
* 名前の設定
MOVE "GET-NAME" TO COBW3-CNV-NAME.
MOVE NC"富士通太郎" TO COBW3-CNV-VALUE-N.
* 結果出力用ページ内の変換データの登録
CALL "COBW3_SET_CNV_XN" USING COBW3.
:
* 結果出力用ページのファイル名の設定
MOVE "a.htm" TO COBW3-HTML-FILENAME.
* 結果出力用ページの出力
CALL "COBW3_PUT_HTML" USING COBW3.
:

```

## 一 Webブラウザ上の出力結果

```
      :  
名前：富士通太郎  
      :
```

### 結果出力用ページ内の繰返し範囲の指定方法

結果出力用ページ内でWebアプリケーションの実行時に繰返し出力したい箇所に“//COBOL\_REPEAT\_START//”、“//COBOL\_REPEAT\_END//”を使用して繰返し範囲を記述します。繰返し回数は、変換対象になる繰返し変換文字列の登録数と同じです。

#### 繰返し範囲開始

データを繰り返す範囲の開始位置に“//COBOL\_REPEAT\_START//”を記述します。

#### 繰返し項目

繰返し出力を行いたいデータを記述します。データの中でWebアプリケーションの実行時に変換して出力したい箇所には、変換名を記述します。

繰返しに応じて異なる変換文字列を出力する場合は、前後を“//COBOL\_REPEAT//”で囲んだ文字列(変換名)を記述します。繰返し時に、固定の変換文字列を出力する場合は、前後を“//COBOL//”で囲んだ文字列(変換名)を記述します。

#### 繰返し範囲終了

データを繰り返す範囲の終了位置に“//COBOL\_REPEAT\_END//”を記述します。

```
//COBOL_REPEAT_START//  
・ • //COBOL_REPEAT//変換名//COBOL_REPEAT// ・ •  
・ • //COBOL//変換名//COBOL// ・ •  
//COBOL_REPEAT_END//
```

## 注意

- “//COBOL\_REPEAT//”で囲んだ変換名に対応する変換文字列の登録は、必ず1回以上行ってください。
- 繰返し項目に“//COBOL\_REPEAT//”で囲んだ変換名が記述されない場合、繰返し回数は1回になります。
- “//COBOL//”および“//COBOL\_REPEAT//”は複数行にまたがって指定することはできません。
- 繰返し範囲の開始/終了を指示する“//COBOL\_REPEAT\_START//”または“//COBOL\_REPEAT\_END//”を含む行は、Webブラウザに出力するときに削除されます。したがって、表示に関わるタグ等を同じ行に記述しないでください。
- 繰返し範囲内に記述した繰返し変換文字列は、すべて同じ登録数にしてください。登録数が異なる場合、繰返し数に対して登録データが足りない変換名が存在するため、エラーになります。
- 繰返し範囲開始を示す記述がされていない場合、繰返し出力は行われずに、“//COBOL//”以外の記述はデータとみなして出力されます。
- 繰返し範囲終了を示す記述がされていない場合、結果出力用ページの末尾まで出力後、エラーになります。
- 変換名の指定方法による振る舞いは、結果出力用ページ内の変換名の指定方法と同じです。

## 例

- b.htm (結果出力用ページ)

```
<TABLE>  
<TR>  
  <TD>名前</TD>  
  <TD>年齢</TD>  
  <TD>会社</TD>  

```

```

//COBOL_REPEAT_START//
<TR>
  <TD>//COBOL_REPEAT//GET-NAME//COBOL_REPEAT//</TD>
  <TD>//COBOL_REPEAT//GET-AGE//COBOL_REPEAT//</TD>
  <TD>//COBOL//GET-OFFICE//COBOL//</TD>
</TR>
//COBOL_REPEAT_END//
</TABLE>

```

• COBOLプログラム

```

:
PERFORM UNTIL END-FLAG="END"
* 名前の設定
  MOVE "GET-NAME" TO COBW3-CNV-NAME
  MOVE 名前 TO COBW3-CNV-VALUE-N
* 結果出力用ページ内の繰返し変換データの登録
  CALL "COBW3_SET_REPEAT_XN" USING COBW3
* 年齢の設定
  MOVE "GET-AGE" TO COBW3-CNV-NAME
  MOVE 年齢 TO COBW3-CNV-VALUE
* 結果出力用ページ内の繰返し変換データの登録
  CALL "COBW3_SET_REPEAT_XX" USING COBW3
:
END-PERFORM.
:
* 会社名の設定
  MOVE "GET-OFFICE" TO COBW3-CNV-NAME.
  MOVE  "NC"富士通" TO COBW3-CNV-VALUE-N.
* 結果出力用ページ内の変換データの登録
  CALL "COBW3_SET_CNV_XN" USING COBW3.
:
* 結果出力用ページのファイル名の設定
  MOVE "b.htm" TO COBW3-HTML-FILENAME.
* 結果出力用ページの出力
  CALL "COBW3_PUT_HTML" USING COBW3.
:

```

• Webブラウザ上の出力結果

```

:

```

名前	年齢	会社
鈴木	35	富士通
佐藤	26	富士通
田中	23	富士通

```

:

```

## I.4.29 COBW3\_PUT\_TEXT(データの出力(改行コードを付加))

指定した文字列に改行コードを加えたデータを、クライアントのWebブラウザに出力します。

## 書き方

```
CALL "COBW3_PUT_TEXT" USING COBW3.
```

### 呼出し時のデータ設定

#### COBW3-PUT-STRING

出力する文字列を設定します。

#### COBW3-PUT-STRING-LENGTH[省略可]

最後に有効な空白がある文字列を出力する場合、空白を含む値の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで出力します。ただし、COBW3-PUT-STRINGがすべて空白の場合、文字列長をゼロ(改行だけ)として処理します。
1~1024	指定された文字列の長さで出力します。

### 処理結果のデータ

なし

### 文字列データ(改行コード含む)をWebブラウザに出力する方法

Webブラウザに“<CENTER>ご協力ありがとうございました。</CENTER>”を出力する場合の記述例を以下に示します。



#### 例

- COBOLプログラム

```
      :  
000360* 出力する文字列のセット  
000370  MOVE "<CENTER>ご協力ありがとうございました。</CENTER>"  
000380                                TO COBW3-PUT-STRING.  
000390* 文字列データの出力  
000400  CALL "COBW3_PUT_TEXT" USING COBW3.  
      :
```

- Webブラウザ上の出力結果

```
      :  
ご協力ありがとうございました。  
      :
```

## I.4.30 COBW3\_RECEIVE\_HEADER(HTTPヘッダの取得)

リクエストに対するHTTPヘッダを取得します。

## 書き方

```
CALL "COBW3_RECEIVE_HEADER" USING COBW3.
```

### 呼出し時のデータ設定

#### COBW3-HEADER-NAME

取得するヘッダ名を指定します。

#### COBW3-HEADER-NAME-LENGTH[省略可]

最後に有効な空白があるヘッダ名を検索する場合、空白を含むヘッダ名の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで検索します。
1～64	指定された文字列の長さで検索します。

### 注意

通常、末尾に空白を持つヘッダ名はないので、0を指定してください。

#### 処理結果のデータ

##### COBW3-HEADER-VALUE

リクエストのヘッダ名に対応するヘッダ値が設定されます。

##### COBW3-HEADER-VALUE-LENGTH

リクエストのヘッダ名に対応するヘッダ値の文字列長(バイト長)が設定されます。

## I.4.31 COBW3\_SET\_CNV\_XX、COBW3\_SET\_CNV\_NX、COBW3\_SET\_CNV\_XN、COBW3\_SET\_CNV\_NN(結果出力用ページ内の変換データの登録)

“COBW3\_PUT\_HTML”で出力する結果出力用ページ内に指定された“//COBOL//”で囲まれた変換名に対する変換文字列を登録します。

登録された情報はCOBW3\_PUT\_HTML実行時に参照され、登録された変換データに従って、結果出力用ページ内の変換が行われます。各サブルーチンの動作概要は次のとおりです。

サブルーチン名	動作概要
COBW3_SET_CNV_XX	英数字文字列の変換名に対応する変換文字列を英数字文字列として登録します。
COBW3_SET_CNV_NX	日本語文字列の変換名に対応する変換文字列を英数字文字列として登録します。
COBW3_SET_CNV_XN	英数字文字列の変換名に対応する変換文字列を日本語文字列として登録します。
COBW3_SET_CNV_NN	日本語文字列の変換名に対応する変換文字列を日本語文字列として登録します。

### 注意

結果出力用ページ内に複数の異なる変換名が指定されている場合は、変換名ごとに“COBW3\_SET\_CNV\_XX”などを呼び出して変換文字列を登録してください。

#### 書き方

```
CALL "COBW3_SET_CNV_XX" USING COBW3.
```

```
CALL "COBW3_SET_CNV_NX" USING COBW3.
```

```
CALL "COBW3_SET_CNV_XN" USING COBW3.
```

```
CALL "COBW3_SET_CNV_NN" USING COBW3.
```

## 呼出し時のデータ設定

COBW3-CNV-NAME

COBW3-CNV-NAME-N

変換対象となる変換名を設定します。

COBW3\_SET\_CNV\_XXまたはCOBW3\_SET\_CNV\_XNの場合、COBW3-CNV-NAMEに設定します。

COBW3\_SET\_CNV\_NXまたはCOBW3\_SET\_CNV\_NNの場合、COBW3-CNV-NAME-Nに設定します。

COBW3-CNV-NAME-LENGTH[省略可]

最後に有効な空白がある変換名を指定する場合、空白を含む値の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで検索します。ただし、COBW3-CNV-NAMEまたはCOBW3-CNV-NAME-Nがすべて空白の場合、文字列長をゼロとして処理します。
1～30	指定された文字列の長さで検索します。

COBW3-CNV-VALUE

COBW3-CNV-VALUE-N

変換結果(変換文字列)を設定します。

COBW3\_SET\_CNV\_XXまたはCOBW3\_SET\_CNV\_NXの場合、COBW3-CNV-VALUEに設定します。

COBW3\_SET\_CNV\_XNまたはCOBW3\_SET\_CNV\_NNの場合、COBW3-CNV-VALUE-Nに設定します。

COBW3-CNV-VALUE-LENGTH[省略可]

最後に有効な空白がある変換文字列を指定する場合、空白を含む値の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで登録します。ただし、COBW3-CNV-VALUE またはCOBW3-CNV-VALUE-N がすべて空白の場合文字列長をゼロとして処理します。
1～1024	指定された文字列の長さで登録します。

COBW3-CNV-MODE[省略可]

変換種別を設定します。

条件名	値	意味
COBW3-CNV-MODE- ADDREP	LOW-VALUE	指定した変換名に対する変換文字列が登録されていない場合、変換データを追加します。指定した変換名に対する変換文字列が登録されている場合は、変換データを置き換えます。
COBW3-CNV-MODE- REPLACE	"1"	変換データの置き換えを行います。指定した変換名に対する変換文字列が登録されていない場合、エラーを出力して変換を行いません。
COBW3-CNV-MODE-ADD	"2"	変換データの追加を行います。指定した変換名に対する変換文字列が登録されていない場合、エラーを出力して追加を行いません。

COBW3\_SANITIZE\_CNV[省略可]

変換データ内にクロスサイトスクリプティングを引き起こす可能性のある文字を検出した場合、自動的にサニタイジング(置換え)処理を施します。

クロスサイトスクリプティングの詳細は、“付録N セキュリティ”を参照してください。

COBW3\_SANITIZE\_CNVは、COBW3\_SET\_CNV\_XXまたはCOBW3\_SET\_CNV\_NX使用時に有効となります。ただし、動作コード系がUnicodeの場合のみ、COBW3\_SET\_CNV\_XNまたはCOBW3\_SET\_CNV\_NNでも有効となります。

条件名	値	意味
COBW3-SANITIZE-CNV-OFF	LOW-VALUE	サニタイジング処理を行いません。
COBW3-SANITIZE-CNV-ON	"1"	サニタイジング処理を行います。

### 注意

サニタイジング処理では、クロスサイトスクリプティングを引き起こす可能性のある5つの文字(&,<,>,”,’)を、以下のようなエスケープ文字へと置き換えます。

```
& → &amp;
< → &lt;
> → &gt;
” → &quot;
’ → &#39;
```

上記から、置換え前は1文字だったものが、エスケープ文字に置き換えた結果4〜6文字に増大します。このため変換データの内容によっては、置換え後の文字列がWebパラメタのVALUEの最大文字列長(1024バイト)を超えてしまう可能性があります。その場合、最大文字列長を超えた分のデータは、自動的に切り落とされます。

また、最大文字列長の関係で置換え後のエスケープ文字が途中で途切れてしまう場合、対象の文字は削除されるのみとなり、置換えは行われません。以下に例を示します。

#### [例]

変換データの領域が1021バイトで、1〜1020バイトにはサニタイジング処理されない文字が埋められており、末尾の文字のみ「&」の場合

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX& ←末尾文字が「&」
|                     1021バイト                     |
```

サニタイジング処理を行うと以下ようになります。

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX&amp;
|                     1025バイト                     |
```

しかし、置換え後のエスケープ文字列(&amp;)中で1024バイトを超えてしまったため、実際には以下のデータが設定されます。

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX ←末尾文字の「&」は削除
|                     1020バイト                     |
```

#### 処理結果のデータ

##### COBW3\_SANITIZED\_CNV\_FLAG

COBW3\_SANITIZED\_CNVを設定した場合、実際に置換え(サニタイジング)処理を行ったか判別する値が設定されます。COBW3\_SANITIZED\_CNV\_FLAGは、COBW3\_SET\_CNV\_XXまたはCOBW3\_SET\_CNV\_NX使用時に有効となります。ただし、動作コード系がUnicodeの場合のみ、COBW3\_SET\_CNV\_XNまたはCOBW3\_SET\_CNV\_NNでも有効となります。

条件名	値	意味
COBW3-SANITIZED-CNV-NON	"0"	サニタイジング処理を行いませんでした。
COBW3-SANITIZED-CNV-EXIST	"1"	サニタイジング処理を行いました。

## I.4.32 COBW3\_SET\_COOKIE(ブラウザへ送信するCookieデータの登録)

ブラウザへ送信するCookieデータの登録を行います。登録されたCookieデータはヘッダ出力時に使用されます。

#### 書き方

```
CALL "COBW3_SET_COOKIE" USING COBW3.
```

## 呼出し時のデータ設定

### COBW3-COOKIE-NAME、COBW3-COOKIE-NAME-N

登録するCookie名を設定します。

Cookie名が英数字の場合、COBW3-COOKIE-NAMEに設定します。

Cookie名が日本語の場合、COBW3-COOKIE-NAME-Nに設定します。

### COBW3-COOKIE-NAME-LENGTH[省略可]

最後に有効な空白があるCookie名の場合、空白を含むCookie名の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで登録します。ただし、COBW3-COOKIE-NAMEがすべて空白の場合、文字列長をゼロとして処理します。
1～64	指定された文字列の長さで登録します。

### COBW3-COOKIE-VALUE、COBW3-COOKIE-VALUE-N

Cookie値を設定します。

Cookie値が英数字の場合、COBW3-COOKIE-VALUEに設定します。

Cookie値が日本語の場合、COBW3-COOKIE-VALUE-Nに設定します。

### COBW3-COOKIE-VALUE-LENGTH[省略可]

最後に有効な空白があるCookie値の場合、空白を含むCookie値の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで登録します。ただし、COBW3-COOKIE-VALUEがすべて空白の場合、文字列長をゼロとして処理します。
1～1024	指定された文字列の長さで登録します。

### COBW3-COOKIE-EXPIRES[省略可]

Cookieの有効期限を指定する場合に使用します。この項目は、次の項目(年、月、日、時、分、秒)を持ちます。有効期限の指定はGMT(グリニッジ標準時刻)で行います。

- COBW3-COOKIE-EXPIRES-YEAR(年)(1582～9999)
- COBW3-COOKIE-EXPIRES-MONTH(月)(01～12)
- COBW3-COOKIE-EXPIRES-DAY(日)(01～31 ただし、末日は月によって異なります。)
- COBW3-COOKIE-EXPIRES-HOUR(時)(00～23)
- COBW3-COOKIE-EXPIRES-MIN(分)(00～59)
- COBW3-COOKIE-EXPIRES-SEC(秒)(00～59)

## 注意

有効期限が省略された場合、そのCookieデータはWebブラウザ終了時に削除されます。また、過去の時刻を指定することで、現在Webブラウザが持っているCookieデータを削除できます。

JST(日本標準時)とGMTの時差は9時間です。したがって、JSTから9を引いた値がGMTになります。

有効期限の指定には1582年10月15日以降の日付を使用してください。この日より前の日付を指定すると曜日の自動算出ができないため、有効期限が無効になります。

### COBW3-COOKIE-DOMAIN[省略可]

Cookieが有効となるドメインを指定します。

### COBW3-COOKIE-PATH[省略可]

Cookieが有効となる仮想ディレクトリを指定します。

### COBW3-COOKIE-SECURE[省略可]

セキュリティを指定します。セキュリティが有効な場合、SSLを使用している場合にだけWebブラウザからCookieデータが送信されます。

条件名	値	意味
<u>COBW3-COOKIE-SECURE-OFF</u>	LOW-VALUE	セキュリティを無効にします。
COBW3-COOKIE-SECURE-ON	"1"	セキュリティを有効にします。

### COBW3-COOKIE-MODE[省略可]

Cookieデータの登録種別を設定します。

条件名	値	意味
<u>COBW3-COOKIE-MODE-ADDREP</u>	LOW-VALUE	指定したCookie名が登録されていない場合、Cookieデータを追加します。
COBW3-COOKIE-MODE-REPLACE	"1"	Cookieデータの置き換えを行います。指定したCookie名が登録されていない場合、エラーを出力して置き換えを行いません。
COBW3-COOKIE-MODE-ADD	"2"	Cookieデータの追加を行います。指定したCookie名が登録されている場合、エラーを出力して追加を行いません。

### 処理結果のデータ

なし



- 当サブルーチンは、COBW3\_PUT\_HTMLおよびCOBW3\_PUT\_TEXTを使用する前に使用してください。COBW3\_PUT\_HTMLまたはCOBW3\_PUT\_TEXTを一度でも使用した後に当サブルーチンを使用した場合、設定したCOOKIEデータは無効になります。Webブラウザへ送信されるCookieの文字コードは、COBOLの動作コード系に関係なくシフトJISになります。ただし、CookieはCOBOL以外のアプリケーションでも送受信されることおよび、主としてキー情報などを送受信するのに使用されることから、ASCIIの範囲で設定することをおすすめします。
- 本サブルーチンは互換を保証するために提供されています。
- アプリケーションの動作コード系がシフトJISの場合にだけ、動作が保証されます。UnicodeのWebアプリケーションでは使用できないため、代替サブルーチンを使用してください。  
なお、資産の移行性を高めるためには、シフトJISのWebアプリケーションでも代替サブルーチンの使用をおすすめします。

サブルーチン名	代替サブルーチン
COBW3_SET_COOKIE_E	COBW3_SET_COOKIE_XX、COBW3_SET_COOKIE_NX、 COBW3_SET_COOKIE_XN、COBW3_SET_COOKIE_NN

## I.4.33 COBW3\_SET\_COOKIE\_XX、COBW3\_SET\_COOKIE\_NX、 COBW3\_SET\_COOKIE\_XN、COBW3\_SET\_COOKIE\_NN(Cookieデータの登録)

Webブラウザへ送信するCookieデータの登録を行います。登録されたCookieデータはヘッダ出力時に使用されます。

各サブルーチンの動作概要は次のとおりです。

サブルーチン名	動作概要
COBW3_SET_COOKIE_XX	英数字文字列のCookie名と、対応するCookie値を英数字文字列として登録します。
COBW3_SET_COOKIE_NX	日本語文字列のCookie名と、対応するCookie値を英数字文字列として登録します。
COBW3_SET_COOKIE_XN	英数字文字列のCookie名と、対応するCookie値を日本語文字列として登録します。
COBW3_SET_COOKIE_NN	日本語文字列のCookie名と、対応するCookie値を日本語文字列として登録します。

## 注意

当サブルーチンは、COBW3\_PUT\_HTMLおよびCOBW3\_PUT\_TEXTを使用する前に使用してください。COBW3\_PUT\_HTMLまたはCOBW3\_PUT\_TEXTを一度でも使用した後に当サブルーチンを使用した場合、設定した情報は無効になります。

Webブラウザへ送信されるCookieの文字コードは、COBOLの動作コード系に関係なくシフトJISになります。ただし、CookieはCOBOL以外のアプリケーションでも送受信されることおよび、主としてキー情報などを送受信するのに使用されることから、ASCIIの範囲で設定することをおすすめします。

## 書き方

```
CALL "COBW3_SET_COOKIE_XX" USING COBW3.
```

```
CALL "COBW3_SET_COOKIE_NX" USING COBW3.
```

```
CALL "COBW3_SET_COOKIE_XN" USING COBW3.
```

```
CALL "COBW3_SET_COOKIE_NN" USING COBW3.
```

## 呼出し時のデータ設定

COBW3-COOKIE-NAME

COBW3-COOKIE-NAME-N

登録するCookie名を設定します。

COBW3\_SET\_COOKIE\_XXまたはCOBW3\_SET\_COOKIE\_XNの場合、COBW3-COOKIE-NAMEに設定します。

COBW3\_SET\_COOKIE\_NXまたはCOBW3\_SET\_COOKIE\_NNの場合、COBW3-COOKIE-NAME-Nに設定します。

COBW3-COOKIE-NAME-LENGTH[省略可]

最後に有効な空白があるCookie名を指定する場合、空白を含むCookie名の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで登録します。ただし、COBW3-COOKIE-NAMEまたはCOBW3-COOKIE-NAME-Nがすべて空白の場合、文字列長をゼロとして処理します。
1~64	指定された文字列の長さで登録します。

COBW3-COOKIE-VALUE、COBW3-COOKIE-VALUE-N

Cookie値を設定します。

COBW3\_SET\_COOKIE\_XXまたはCOBW3\_SET\_COOKIE\_NXの場合、COBW3-COOKIE-VALUEに設定します。

COBW3\_SET\_COOKIE\_XNまたはCOBW3\_SET\_COOKIE\_NNの場合、COBW3-COOKIE-VALUE-Nに設定します。

COBW3-COOKIE-VALUE-LENGTH[省略可]

最後に有効な空白があるCookie値を指定する場合、空白を含むCookie値の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで登録します。ただし、COBW3-COOKIE-VALUEまたはCOBW3-COOKIE-VALUE-Nがすべて空白の場合、文字列長をゼロとして処理します。

値	意味
1～1024	指定された文字列の長さで登録します。

#### COBW3-COOKIE-EXPIRES[省略可]

Cookieの有効期限を指定する場合に使用します。この項目は、次の項目(年、月、日、時、分、秒)を持ちます。有効期限の指定はGMT(グリニッジ標準時刻)で行います。

- COBW3-COOKIE-EXPIRES-YEAR(年)(1582～9999)
- COBW3-COOKIE-EXPIRES-MONTH(月)(01～12)
- COBW3-COOKIE-EXPIRES-DAY(日)(01～31、ただし、末日は月によって異なります。)
- COBW3-COOKIE-EXPIRES-HOUR(時)(00～23)
- COBW3-COOKIE-EXPIRES-MIN(分)(00～59)
- COBW3-COOKIE-EXPIRES-SEC(秒)(00～59)

#### 注意

有効期限が省略された場合、そのCookieデータはWebブラウザ終了時に削除されます。また、過去の時刻を指定することで、現在Webブラウザが持っているCookieデータを削除できます。

JST(日本標準時)とGMTの時差は9時間です。したがって、JSTから9を引いた値がGMTになります。

有効期限の指定には1582年10月15日以降の日付を使用してください。この日より前の日付を指定すると曜日の自動算出ができないため、有効期限が無効になります。

#### COBW3-COOKIE-DOMAIN[省略可]

Cookieが有効となるドメインを指定します。

#### COBW3-COOKIE-PATH[省略可]

Cookieが有効となる仮想パスを指定します。

#### COBW3-COOKIE-SECURE[省略可]

セキュリティを指定します。セキュリティが有効な場合、SSLを使用している場合にだけWebブラウザからCookieデータが送信されます。

条件名	値	意味
COBW3-COOKIE-SECURE-OFF	LOW-VALUE	セキュリティを無効にします。
COBW3-COOKIE-SECURE-ON	"1"	セキュリティを有効にします。

#### COBW3-COOKIE-MODE[省略可]

Cookieデータの登録種別を設定します。

条件名	値	意味
COBW3-COOKIE-MODE-ADDREP	LOW-VALUE	指定したCookie名が登録されていない場合、Cookieデータを追加します。
COBW3-COOKIE-MODE-REPLACE	"1"	Cookieデータの置き換えを行います。指定したCookie名が登録されていない場合、エラーを出力して置き換えを行いません。
COBW3-COOKIE-MODE-ADD	"2"	Cookieデータの追加を行います。指定したCookie名が登録されている場合、エラーを出力して追加を行いません。

## 処理結果のデータ

なし

### I.4.34 COBW3\_SET\_REPEAT\_XX、COBW3\_SET\_REPEAT\_NX、COBW3\_SET\_REPEAT\_XN、COBW3\_SET\_REPEAT\_NN(結果出力用ページ内の繰返し変換データの登録)

“COBW3\_PUT\_HTML”で出力する結果出力用ページ内に指定された“//COBOL\_REPEAT//”で囲まれた変換名に対する繰返し変換データの登録を行います。

登録された情報はCOBW3\_PUT\_HTML実行時に参照され、同一の変換名に対して登録された繰返し変換文字列の登録数に応じて、結果出力用ページ内の繰返し範囲に指定された変換名が変換されます。

各サブルーチンの動作概要は次のとおりです。

サブルーチン名	動作概要
COBW3_SET_REPEAT_XX	英数字文字列の変換名に対応する変換文字列を英数字文字列として登録します。
COBW3_SET_REPEAT_NX	日本語文字列の変換名に対応する変換文字列を英数字文字列として登録します。
COBW3_SET_REPEAT_XN	英数字文字列の変換名に対応する変換文字列を日本語文字列として登録します。
COBW3_SET_REPEAT_NN	日本語文字列の変換名に対応する変換文字列を日本語文字列として登録します。



結果出力用ページ内に複数個の異なる変換名が指定されている場合は、変換名ごとに“COBW3\_SET\_REPEAT\_XX”などを呼び出して繰返し変換文字列を登録してください。

繰返し変換時に変換される変換文字列は、“COBW3\_SET\_REPEAT\_XX”などを呼び出して登録した順序に従います。

同じ繰返し範囲に含まれる変換名に対する繰返し変換文字列は、登録数が同じになるように登録してください。

#### 書き方

```
CALL "COBW3_SET_REPEAT_XX" USING COBW3.
```

```
CALL "COBW3_SET_REPEAT_NX" USING COBW3.
```

```
CALL "COBW3_SET_REPEAT_XN" USING COBW3.
```

```
CALL "COBW3_SET_REPEAT_NN" USING COBW3.
```

#### 呼出し時のデータ設定

COBW3-CNV-NAME

COBW3-CNV-NAME-N

変換対象となる変換名を設定します。

COBW3\_SET\_REPEAT\_XXまたはCOBW3\_SET\_REPEAT\_XNの場合、COBW3-CNV-NAMEに設定します。

COBW3\_SET\_REPEAT\_NXまたはCOBW3\_SET\_REPEAT\_NNの場合、COBW3-CNV-NAME-Nに設定します。

COBW3-CNV-NAME-LENGTH[省略可]

最後に有効な空白がある変換名の場合、空白を含む値の文字列長(バイト長)を設定します。

値	意味
0	空白でない最後の文字までの長さで検索します。ただし、COBW3-CNV-NAMEまたはCOBW3-CNV-NAME-Nがすべて空白の場合、文字列長をゼロとして処理します。
1～30	指定された文字列の長さで検索します。

COBW3-CNV-VALUE  
COBW3-CNV-VALUE-N

変換結果(変換文字列)を設定します。

COBW3\_SET\_REPEAT\_XXまたはCOBW3\_SET\_REPEAT\_NXの場合、COBW3-CNV-VALUEに設定します。

COBW3\_SET\_REPEAT\_XNまたはCOBW3\_SET\_REPEAT\_NNの場合、COBW3-CNV-VALUE-Nに設定します。

COBW3-CNV-VALUE-LENGTH[省略可]

最後に有効な空白がある変換文字列の場合、空白を含む値の文字列長(バイト長)を設定します。

値	意味
0	空白でない最後の文字までの長さで登録します。ただし、COBW3-CNV-VALUEまたはCOBW3-CNV-VALUE-Nがすべて空白の場合文字列長をゼロとして処理します。
1～1024	指定された文字列の長さで登録します。

 注意

変換結果に空白を指定したい場合には、変換文字列に空白、変換文字列長にゼロを指定して、繰返し変換データを登録してください。

COBW3\_SANITIZE\_CNV[省略可]

変換データ内にクロスサイトスクリプティングを引き起こす可能性のある文字を検出した場合、自動的にサニタイジング(置換え)処理を施します。

クロスサイトスクリプティングの詳細は、“付録N セキュリティ”を参照してください。

COBW3\_SANITIZE\_CNVは、COBW3\_SET\_REPEAT\_XXまたはCOBW3\_SET\_REPEAT\_NX使用時に有効となります。ただし、動作コード系がUnicodeの場合のみ、COBW3\_SET\_REPEAT\_XNまたはCOBW3\_SET\_REPEAT\_NNでも有効となります。

条件名	値	意味
COBW3-SANITIZE-CNV-OFF	LOW-VALUE	サニタイジング処理を行いません。
COBW3-SANITIZE-CNV-ON	"1"	サニタイジング処理を行います。

 注意

サニタイジング処理では、クロスサイトスクリプティングを引き起こす可能性のある5つの文字(&, <, >, “, ’)を、以下のようなエスケープ文字へと置き換えます。

&	→	&amp;
<	→	&lt;
>	→	&gt;
“	→	&quot;
’	→	&#39;

上記から、置換え前は1文字だったものが、エスケープ文字に置き換えた結果4～6文字に増大します。このため変換データの内容によっては、置換え後の文字列がWebパラメタのVALUEの最大文字列長(1024バイト)を超えてしまう可能性があります。その場合、最大文字列長を超えた分のデータは、自動的に切り落とされます。

また、最大文字列長の関係で置換え後のエスケープ文字が途中で途切れてしまう場合、対象の文字は削除されるのみとなり、置換えは行われません。以下に例を示します。

[例]

変換データの領域が1021バイトで、1～1020バイトにはサニタイジング処理されない文字が埋められており、末尾の文字のみ「&」の場合。

xx&	←末尾文字が「&」
	1021バイト

サニタイジング処理を行うと以下のようになります。

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX& ;
|               1025バイト               |
```

しかし、置換え後のエスケープ文字列(&amp;)中で1024バイトを超えてしまったため、実際には以下のデータが設定されます。

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX ←末尾文字の「&」は削除
|               1020バイト               |
```

#### 処理結果のデータ

##### COBW3\_SANITIZED\_CNV\_FLAG

COBW3\_SANITIZE\_CNVを設定した場合、実際に置換え(サニタイジング)処理を行ったか判別する値が設定されます。

COBW3\_SANITIZED\_CNV\_FLAGは、COBW3\_SET\_REPEAT\_XXまたはCOBW3\_SET\_REPEAT\_NX使用時に有効となります。ただし、動作コード系がUnicodeの場合のみ、COBW3\_SET\_REPEAT\_XNまたはCOBW3\_SET\_REPEAT\_NNでも有効となります。

条件名	値	意味
COBW3-SANITIZED-CNV-NON	"0"	サニタイジング処理を行いませんでした。
COBW3-SANITIZED-CNV-EXIST	"1"	サニタイジング処理を行いました。

### I.4.35 COBW3\_SET\_SESSION\_DATA(セッションデータの登録)

現行セッションで使用するセッションデータを登録します。登録したセッションデータは、同一セッション内で共有します。

#### 書き方

```
CALL "COBW3_SET_SESSION_DATA" USING COBW3 セッションデータ.
```

#### 呼出し時のデータ設定

##### COBW3-SESSION-DATA-SIZE

セッションデータ長(バイト長)を指定します。セッションを開始した時、セッションデータの形式にCOBW3-SESSION-DATA-GROUPITEMが設定されている場合だけ有効です。

値	意味
1~999999999	指定されたセッションデータの長さで登録します。

#### セッションデータ

登録するセッションデータを指定します。セッションを開始した時のセッションデータの形式に合わせて、集団項目またはオブジェクト参照項目を指定します。

#### 注意

- セッションデータに集団項目を指定する場合、従属する基本項目にオブジェクト参照項目およびポインタデータ項目を含む集団項目は指定できません。
- セッションデータにオブジェクト参照項目を指定する場合、オブジェクト参照項目が指すオブジェクトのクラスは、COBW3-SESSION-ADAPTERクラスを継承していなければいけません。
- セッションデータにオブジェクト参照項目を指定した場合、そのスレッドが終了する前に必ず、オブジェクト参照項目にNULLオブジェクトを代入してください。NULLオブジェクトを代入しない場合、セッションを終了した後に、どこからも参照されないオブジェクトがメモリ上に残ることがあり、メモリ不足につながります。
- セッションデータを登録すると、集団項目の場合はデータの内容が保持されます。オブジェクト参照項目の場合はオブジェクトが保持されます。

- 登録するセッションデータがセッション開始時に設定したセッションデータの形式と異なる場合、Webアプリケーションの動作は保証できません。

#### 処理結果のデータ

なし



#### 注意

- COBOL CGIサブルーチンでは当機能は使用できません。
- 当サブルーチンは、セッション開設中に呼び出してください。セッションの開始前およびセッションの終了後に当サブルーチンを呼び出した場合、セッションデータは登録できません。  
セッション開設中はセッションデータの形式を変更することはできません。“COBW3\_START\_SESSION”の呼び出し時に設定したセッションデータの形式に合わせたセッションデータを登録してください。

### I.4.36 COBW3\_START\_SESSION(セッションの開始)

セッションを開始します。このとき、開始するセッションのセッションデータの形式およびタイムアウトの時間設定も行います。

#### 書き方

```
CALL "COBW3_START_SESSION" USING COBW3.
```

#### 呼び出し時のデータ設定

##### COBW3-SESSION-DATA-TYPE

セッションデータの形式を指定します。

条件名	値	意味
COBW3-SESSION-DATA-GROUPITEM	"0"	セッションデータに集団項目を使用します。
COBW3-SESSION-DATA-OBJECT	"1"	セッションデータにオブジェクト参照項目を使用します。

##### COBW3-SESSION-TIMEOUT

セッションのタイムアウト時間(秒)を指定します。タイムアウト時間は、“COBW3\_FREE”が呼び出された時点から、同一セッション内の次のスレッドで“COBW3\_INIT”が呼び出された時点までの最大待ち時間です。

値	意味
1 ~ 999999	指定した時間(秒)が経過した時点でタイムアウト処理が実行されます。

#### 処理結果のデータ

なし



#### 注意

- COBOL CGIサブルーチンでは当機能は使用できません。
- 当サブルーチンは、“COBW3\_PUT\_TEXT”、“COBW3\_PUT\_HTML”および“COBW3\_FREE”を使用する前に呼び出してください。これらのサブルーチンを使用した後に当サブルーチンを呼び出した場合、セッションは開始されません。既にセッションが開始されている場合、当サブルーチンを呼び出すとエラーが通知されます。  
セッション開設中はセッションデータの形式を変更することはできません。変更する場合は、“COBW3\_END\_SESSION”を呼び出して一度セッションを終了してください。

## I.4.37 COBW3\_SYSTEM(システムコマンドの実行)

Webアプリケーションから任意のシステムコマンドを実行します。

書き方

```
CALL "COBW3_SYSTEM" USING COBW3.
```

呼出し時のデータ設定

COBW3-SYSTEMINFO

実行するシステムコマンドおよびそのパラメタを設定します。

処理結果のデータ

なし



注意

第三者が不正にシステムに侵入し、システムを破壊するなどの危険があるため、Webブラウザで入力されたコマンド文字をCOBW3\_SYSTEMでそのまま実行するなど、直接クライアント(Webブラウザ)からコマンドが実行できるような設計は行わないでください。

## I.4.38 COBW3\_VALUE(Webパラメタから任意の値を検索)

“COBW3\_INIT”で獲得したWebパラメタから任意の値(VALUE)の検索を行います。このサブルーチンは、チェックボックスのある呼出し用ページから起動された場合に、Webブラウザでチェックされた項目を値(VALUE)で判定する場合などに使用します。

書き方

```
CALL "COBW3_VALUE" USING COBW3.
```

呼出し時のデータ設定

COBW3-SEARCH-DATA、COBW3-SEARCH-DATA-N

検索するVALUEを設定します。

英数字文字列を検索する場合、COBW3-SEARCH-DATAに設定します。

日本語文字列を検索する場合、COBW3-SEARCH-DATA-Nに設定します。

COBW3-SEARCH-LENGTH[省略可]

最後に有効な空白がある値(VALUE)の場合、空白を含む値の文字列長(バイト長)を設定します。

値	意味
0	空白以外の最後の文字までの長さで検索します。
1 ~ 1024	指定された文字列の長さで検索します。

COBW3-NUMBER[省略可]

Webパラメタに同一のVALUEが複数存在する場合、検索対象のVALUEの出現順序を設定します。

条件名	値	意味
COBW3-NUMBER-INIT	1	最初に一致するVALUEを検索します。
—	2~9999	指定された出現順序のVALUEを検索します。

処理結果のデータ

COBW3-SEARCH-FLAG

条件名	値	意味
COBW3-SEARCH-FLAG-NON	"0"	検索対象のVALUEが存在しません。

条件名	値	意味
COBW3-SEARCH-FLAG-EXIST	"1"	検索対象のVALUEが存在しました。

## 注意

本サブルーチンは互換を保証するために提供されています。

アプリケーションの動作コード系がシフトJISの場合にだけ、動作が保証されます。UnicodeのWebアプリケーションでは使用できないため、代替サブルーチンを使用してください。

なお、資産の移行性を高めるためには、シフトJISのWebアプリケーションでも代替サブルーチンの使用をおすすめします。

サブルーチン名	代替サブルーチン
COBW3_VALUE	COBW3_CHECK_VALUE_X、COBW3_CHECK_VALUE_N

## 1.5 メモリ関連サブルーチンの使い方

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のようになります。

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、“[20.2.2 マルチスレッドプログラムとは](#)”を参照してください。

### 1.5.1 シングルスレッドモードのプログラム

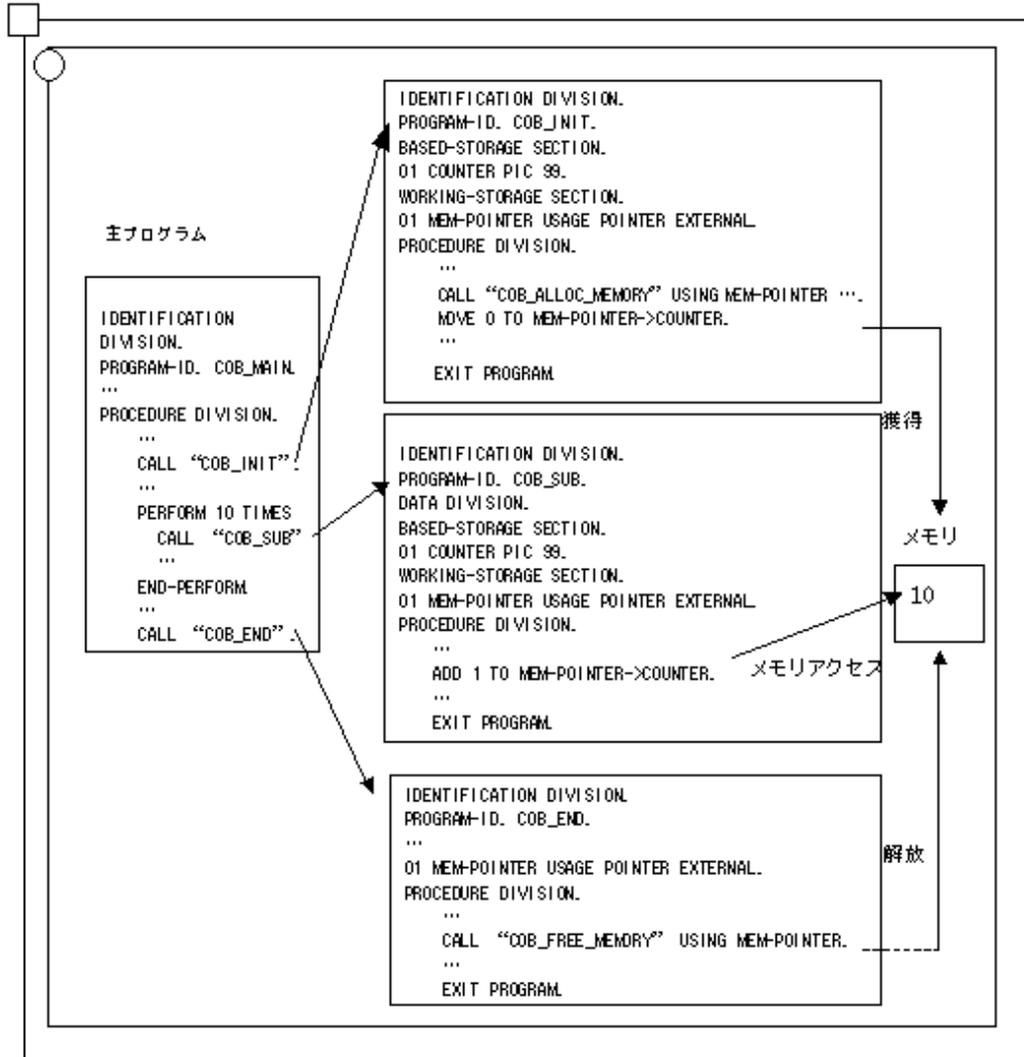
シングルスレッドモードのプログラムで、主プログラムがCOBOLの場合と他言語の場合について説明します。

#### 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I.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)



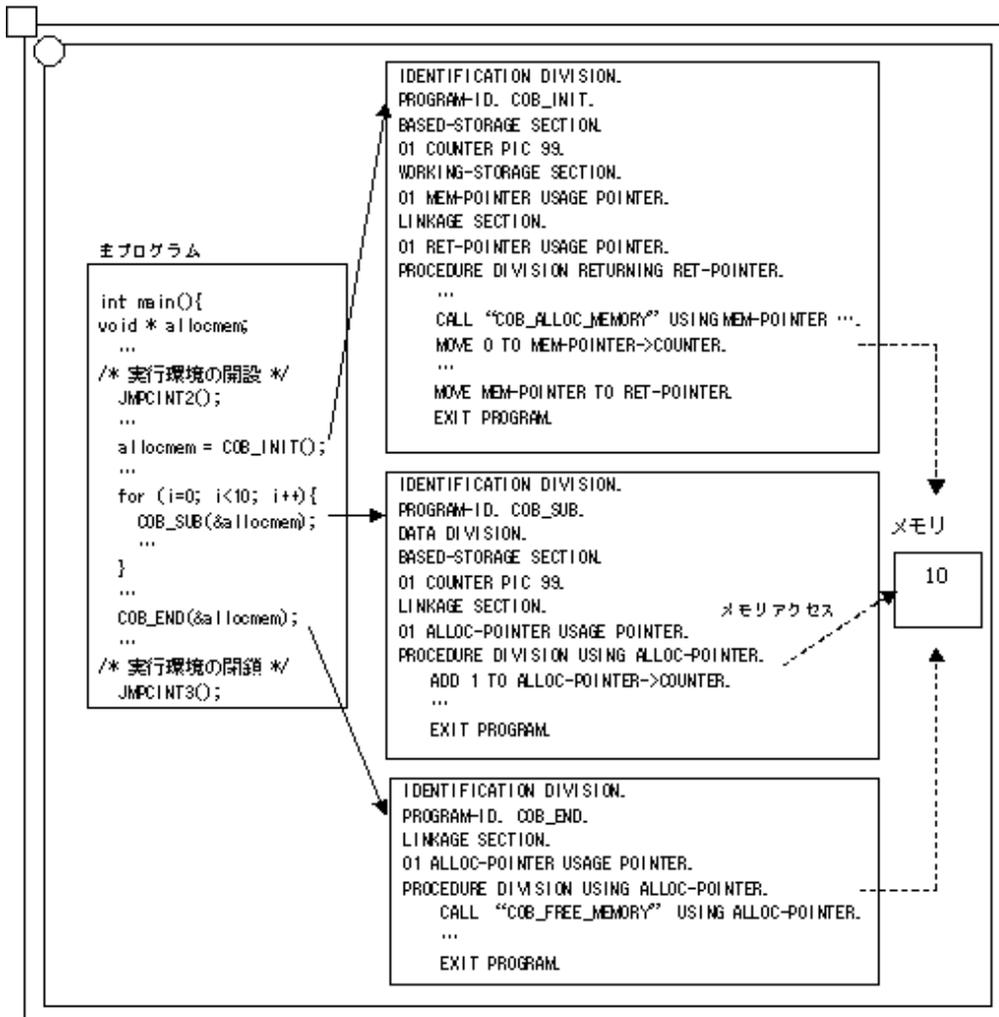
### 主プログラムが他言語の場合

主プログラムが他言語の場合、JMPCINT2およびJMPCINT3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I.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。

JMPCINT2およびJMPCINT3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。

JMPCINT2およびJMPCINT3の使用方法については、“[I.3 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1.2 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)



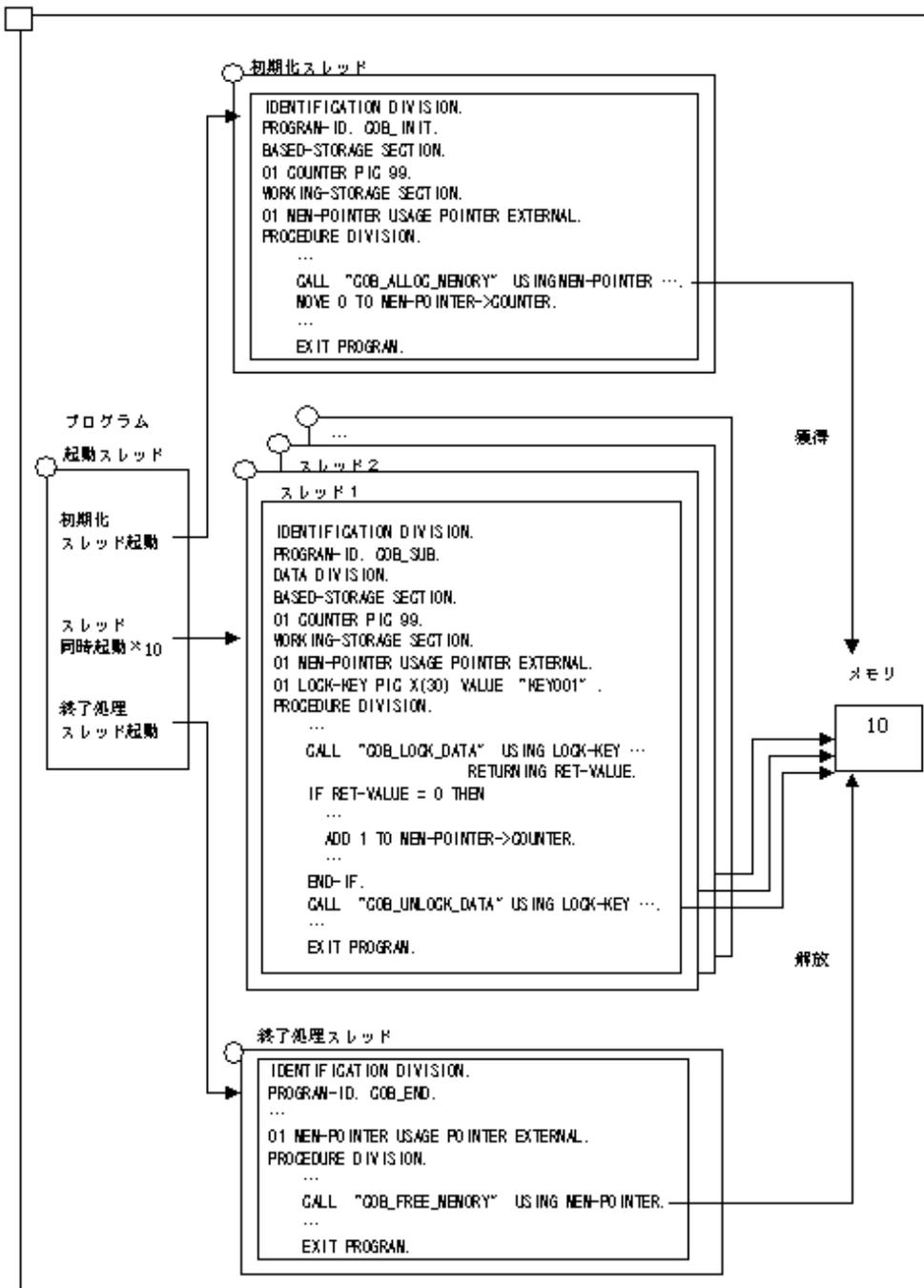
## 1.5.2 マルチスレッドモードのプログラム

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、「[20.4 スレッド間の資源の共有](#)」を参考にしてください。

図I.3 プロセス指定(マルチスレッドモード)

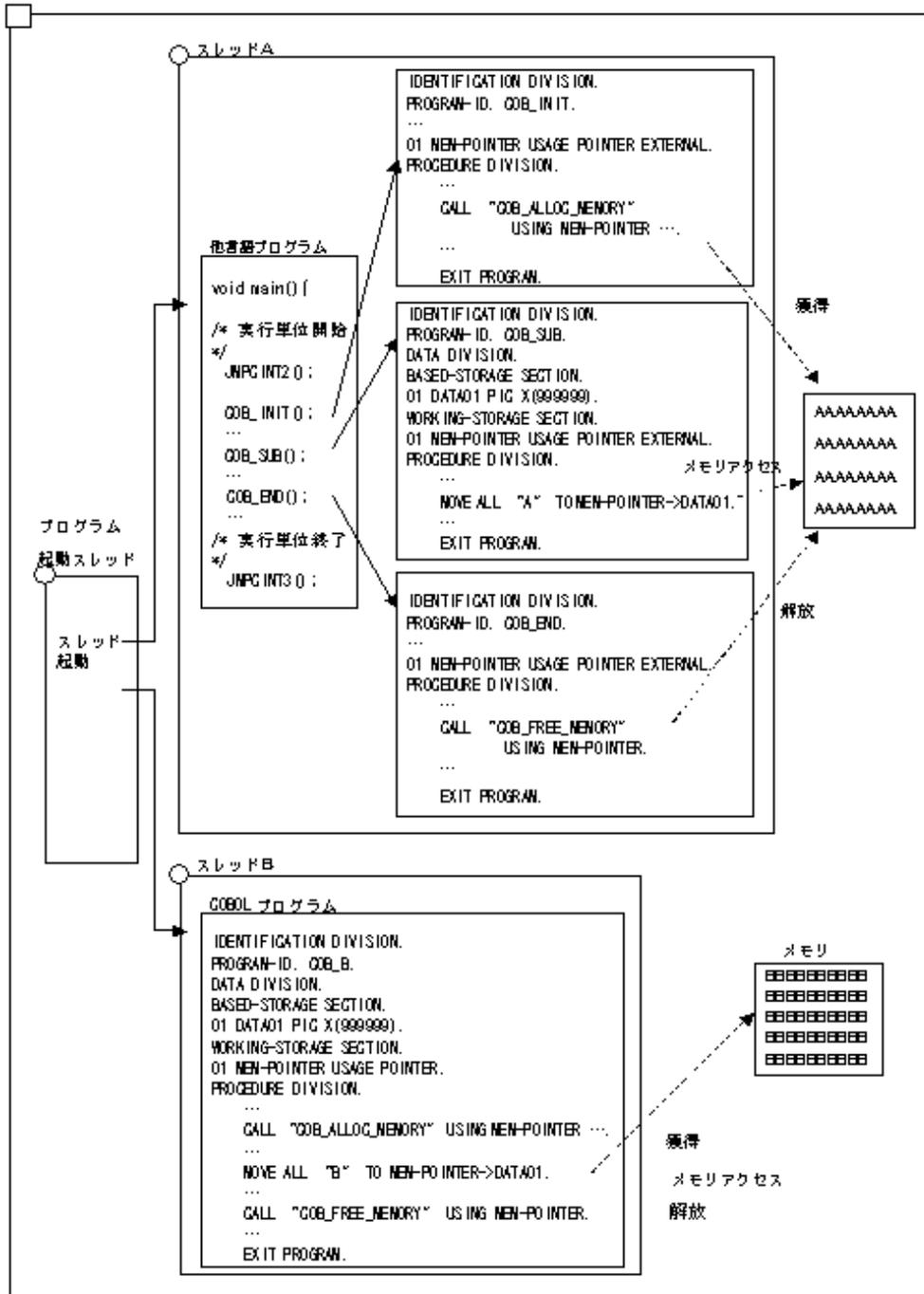


### スレッド指定

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I.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、「5.6 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

## I.6 スレッド同期制御サブルーチンの使い方

スレッド同期制御サブルーチンの使い方は、“[20.4.2 資源の共有](#)”も参照してください。

### I.6.1 データロックサブルーチン

サブルーチン名	機能
<a href="#">COB_LOCK_DATA</a>	ロックキーに対するロックの獲得
<a href="#">COB_UNLOCK_DATA</a>	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、“[5.6 エントリ情報](#)”を参照してください。

```
[ENTRY]
COB_LOCK_DATA=F3B1EFNC. DLL
COB_UNLOCK_DATA=F3B1EFNC. DLL
```

### I.6.2 オブジェクトロックサブルーチン

サブルーチン名	機能
<a href="#">COB_LOCK_OBJECT</a>	オブジェクトに対するロックの獲得
<a href="#">COB_UNLOCK_OBJECT</a>	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、“[5.6 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
```

### I.6.3 スレッド同期制御サブルーチンのエラーコード

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I.1 スレッド同期制御サブルーチンのエラーコード一覧

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。	○	○	○	○
-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>	○	○	○	○
-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。	○	×	○	×
-4	ロックの獲得の待ち時間を経過しました。	○	×	○	×
-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た。	×	○	×	○
-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

×:通知されない

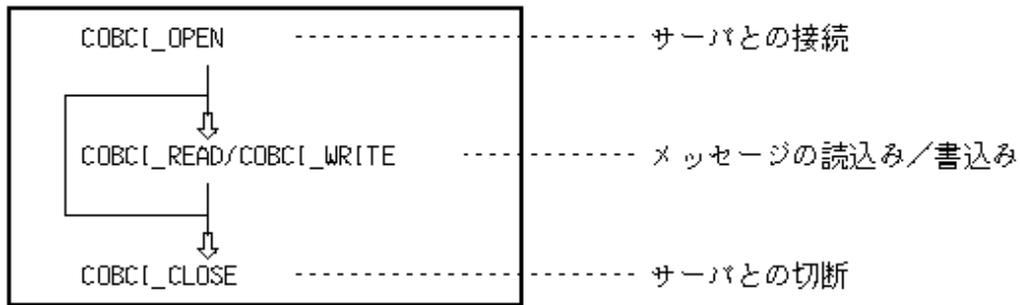
## I.7 簡易アプリ間通信のサブルーチンの使い方

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
COBCL_OPEN	サーバとの接続の確立
COBCL_CLOSE	サーバとの接続の切断
COBCL_READ	サーバからのメッセージの読み込み
COBCL_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.

```

### 1.7.1 簡易アプリ間通信のサブルーチンの注意事項

- 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、“5.6 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。

### 1.7.2 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: COBCI\_OPEN
- C: COBCI\_CLOSE
- R: COBCI\_READ
- W: COBCI\_WRITE

表1.2 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	メッセージの読み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)	—			○	

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
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 ージの読み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超えて いたため、メッセージの書き込みに失敗 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	論理宛先が読み込み可能状態ではない ため、メッセージの読み込みに失敗しま した。	サーバの論理宛先を読み込み可能状態に 変更してください。			○	
	1538	論理宛先が書き込み可能状態ではない ため、メッセージの書き込みに失敗しま した。	サーバの論理宛先を書き込み可能状態に 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、 メッセージの読み込み/書き込みを行いま 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 超えたため、メッセージの書き込みに失 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超 えたため、メッセージの書き込みに失敗 しました。または、システム内最大待ち 命令数を超えたため、メッセージの読 込み/書き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。			○	○
	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。	○			

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
	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I.3 WindowsSockets関数のエラーコード一覧”を参照してください。

表I.3 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。
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。

WINSOCK.H内の表記	エラーコード	意味
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。

## I.8 COBOL Webサブルーチンの使い方

### I.8.1 COBOL Webサブルーチンの機能

COBOL Webサブルーチン(以降ではWebサブルーチンといいます)では次の機能を提供しています。

#### 基本機能

機能	サブルーチン名	用途
作業環境の設定およびWebパラメタの獲得	COBW3_INIT	Webサブルーチン作業環境の設定およびWebパラメタの獲得
Webパラメタの操作	COBW3_GET_VALUE_XX COBW3_GET_VALUE_NX COBW3_GET_VALUE_XN COBW3_GET_VALUE_NN	Webパラメタ中のNAMEの検索およびVALUEの取得
	COBW3_CHECK_VALUE_X COBW3_CHECK_VALUE_N	Webパラメタ中のVALUEの検索
処理結果の出力	COBW3_PUT_HEAD	ヘッダ出力
	COBW3_PUT_HTML	結果出力用ページの出力
	COBW3_SET_CNV_XX COBW3_SET_CNV_NX COBW3_SET_CNV_XN COBW3_SET_CNV_NN	結果出力用ページ内の変換データの登録
	COBW3_DEL_CNV_X COBW3_DEL_CNV_N	結果出力用ページ内の変換データの削除
	COBW3_INIT_CNV	結果出力用ページ内の変換データの初期化
	COBW3_SET_REPEAT_XX COBW3_SET_REPEAT_NX COBW3_SET_REPEAT_XN COBW3_SET_REPEAT_NN	結果出力用ページ内の繰返し変換データの登録
	COBW3_DEL_REPEAT_X	結果出力用ページ内の繰返し変換データの削除

機能	サブルーチン名	用途
	COBW3_DEL_REPEAT_N	
	COBW3_INIT_REPEAT	結果出力用ページ内の繰返し変換データの初期化
	COBW3_PUT_TEXT	データの出力(改行コードを付加)
システムコマンドの実行	COBW3_SYSTEM	指定されたシステムコマンドの実行
Web実行環境の資源の解放	COBW3_FREE	Webサブルーチンの獲得した資源を解放
リクエスト情報の取得	COBW3_RECEIVE_HEADER	HTTPヘッダの取得
	COBW3_GET_REQUEST_INFO	リクエストに関する情報の取得
	COBW3_GET_AUTHORIZE	認証情報の取得
Cookieデータ操作	COBW3_SET_COOKIE_XX	Cookieデータの登録
	COBW3_SET_COOKIE_NX	
	COBW3_SET_COOKIE_XN	
	COBW3_SET_COOKIE_NN	
	COBW3_DEL_COOKIE_X	登録済みのCookieデータの削除
	COBW3_DEL_COOKIE_N	
	COBW3_INIT_COOKIE	登録済みの全Cookieデータの初期化
COBW3_GET_COOKIE_XX	WebパラメタからのCookieデータの取得	
COBW3_GET_COOKIE_NX		
COBW3_GET_COOKIE_XN		
COBW3_GET_COOKIE_NN		

## 参考

COBW3\_XXXXX\_XXやCOBW3\_XXXXX\_Nなどのサブルーチンの意味は次のとおりです。

サブルーチン名	説明
COBW3_XXXXX_XX	NAME/VALUEパラメタ共に英数字項目とみなします
COBW3_XXXXX_NX	NAMEパラメタを日本語項目、VALUEパラメタを英数字項目とみなします
COBW3_XXXXX_XN	NAMEパラメタを英数字項目、VALUEパラメタを日本語項目とみなします
COBW3_XXXXX_NN	NAME/VALUEパラメタ共に日本語項目とみなします
COBW3_XXXXX_X	入力パラメタを英数字項目とみなします
COBW3_XXXXX_N	入力パラメタを日本語項目とみなします

## ファイルアップロード機能

ファイルアップロード機能では、クライアント側で作成したデータ(ファイル)を、サーバに転送することができます。

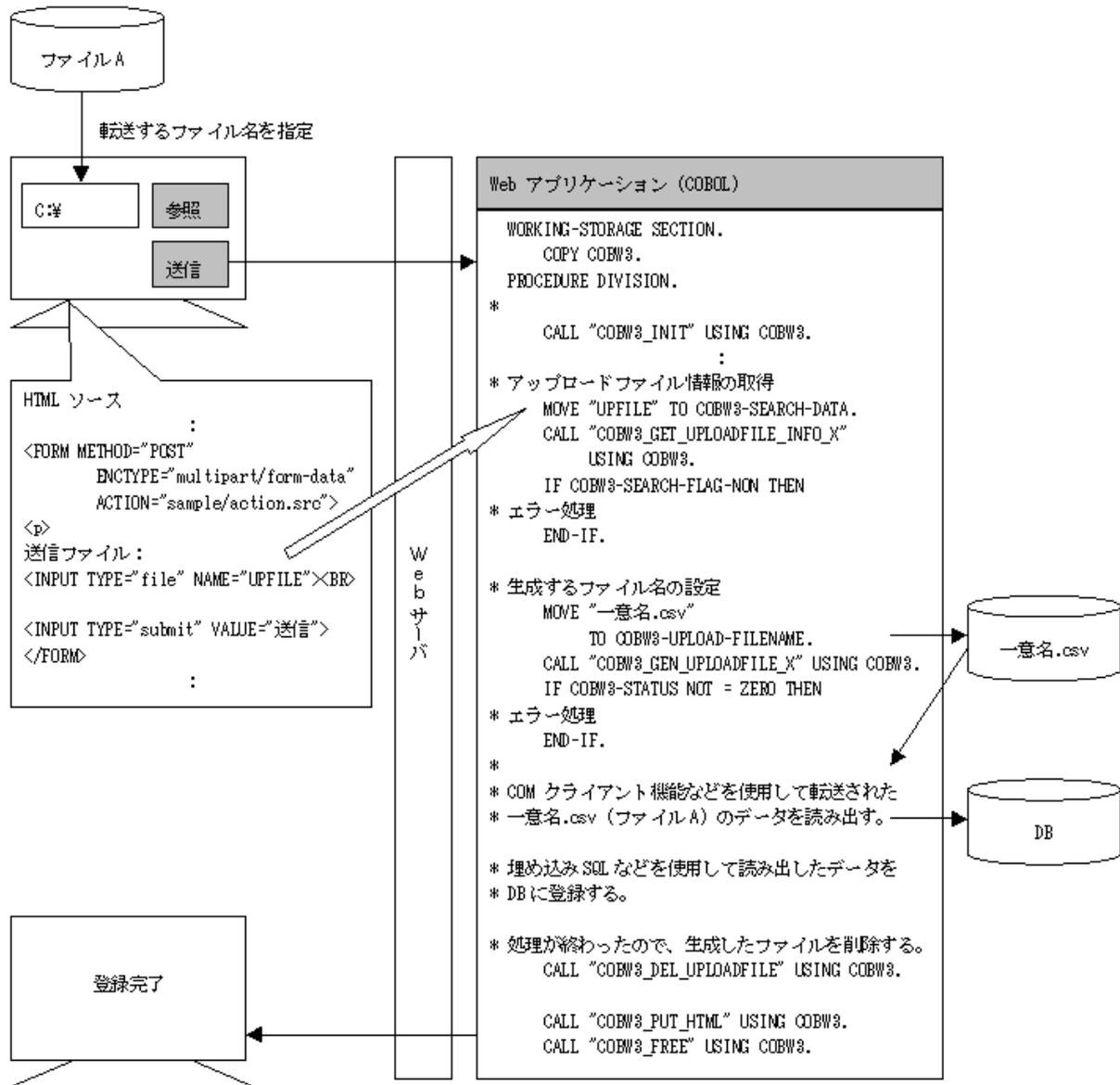
この機能は、以下のような場合に有効です。

- ・ オンラインで処理するとタイムアウトになるような大量のデータエントリをする場合
- ・ 従量制課金のモバイル環境を利用する場合
- ・ すでにクライアント側にデータエントリツールがあり、これらのツールのWeb化をするよりも、ツールが出力するデータ(ファイル)によるサーバ連携を望む場合

- 従来FTPを使用していたファイル転送処理に対し、セキュリティの関係でHTTPポートしかアクセスできなくなった場合

また、ファイルアップロード機能を使用した場合、ファイル転送とアプリケーションを関連付けることができるため、転送直後にデータのチェックおよび登録などの処理を行うことができます。

ファイルアップロード機能を利用した処理の概念図を以下に示します。



Webサブルーチンでは、このような業務プログラムを簡単に作成するために、以下の機能を提供しています。

機能	サブルーチン名	用途
ファイルアップロード	COBW3_GET_UPLOADFILE_INFO_X COBW3_GET_UPLOADFILE_INFO_N	アップロードファイルの情報の取得
	COBW3_GEN_UPLOADFILE_X COBW3_GEN_UPLOADFILE_N	アップロードされたファイルをサーバ上に生成
	COBW3_DEL_UPLOADFILE	アップロード済みファイルの削除

ファイルアップロードを行う場合、呼出し用ページに次の値を記述する必要があります。

タグ名	属性	値
FORM	METHOD	POST
	ENCTYPE	multipart/form-data
INPUT	TYPE	file

タグおよび属性の詳細は、“[O.1.3 HTMLの基礎](#)”およびHTMLについて解説している書籍やホームページなどを参照してください。

なお、上記のタグは同FORM内で他のTYPE("file"以外)のINPUTタグと併用することができます。これらのINPUTタグに対応するデータは、ファイルアップロード機能の使用の有無にかかわらず、COBW3\_GET\_VALUE\_XXなどのサブルーチンを使用して取得できます。

### 注意

ファイルアップロードのためのINPUTタグ(TYPE="file")に指定したNAME値は、他のTYPE("file"以外)のINPUTタグでは使用できません。この場合、Webサブルーチンの動作は保証されません。

### 例

```

:
<INPUT TYPE="text" NAME="FILE1">
<INPUT TYPE="file" NAME="FILE1">      →指定不可
:

```

## セッション管理機能

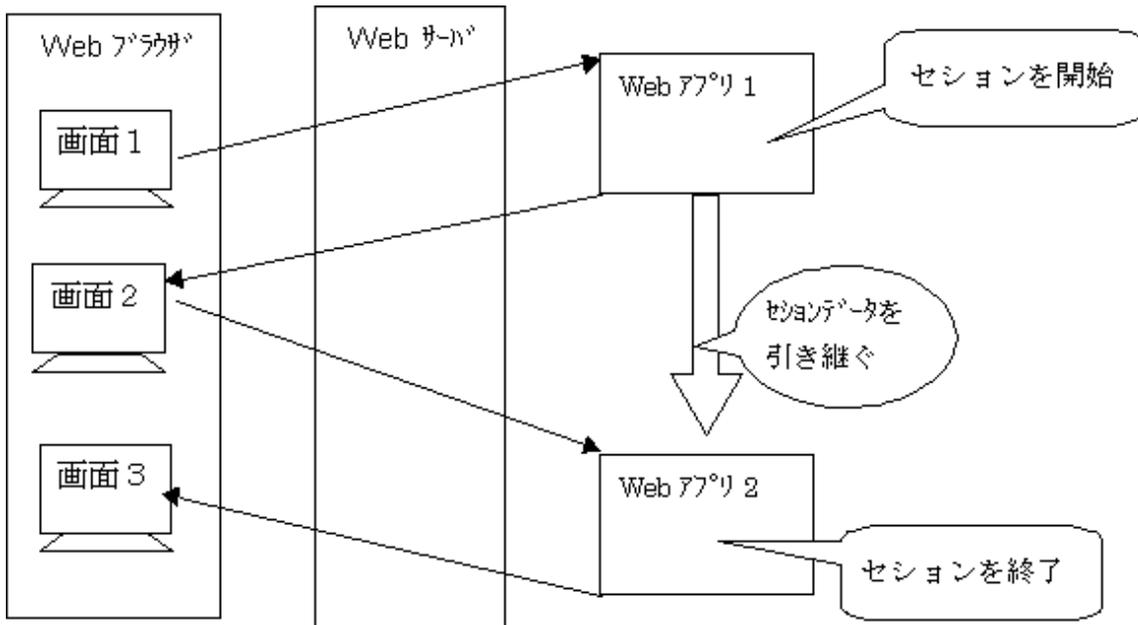
セッション管理機能を使用することで、特定のクライアント(Webブラウザ)から、複数回のリクエストにまたがった業務を行うWebアプリケーションを作成できます。

機能	サブルーチン名	用途
セッション管理	<a href="#">COBW3_START_SESSION</a>	セッションの開始
	<a href="#">COBW3_END_SESSION</a>	セッションの終了
	<a href="#">COBW3_SET_SESSION_DATA</a>	セッションデータの登録
	<a href="#">COBW3_GET_SESSION_DATA</a>	セッションデータの取得
	<a href="#">COBW3_ALTER_SESSION_TIMEOUT</a>	セッションのタイムアウト時間の変更
	<a href="#">COBW3_GET_SESSION_INFO</a>	現行セッション情報の取得

### 参考

セッション管理機能は、COBOL CGIサブルーチンでは利用できません。

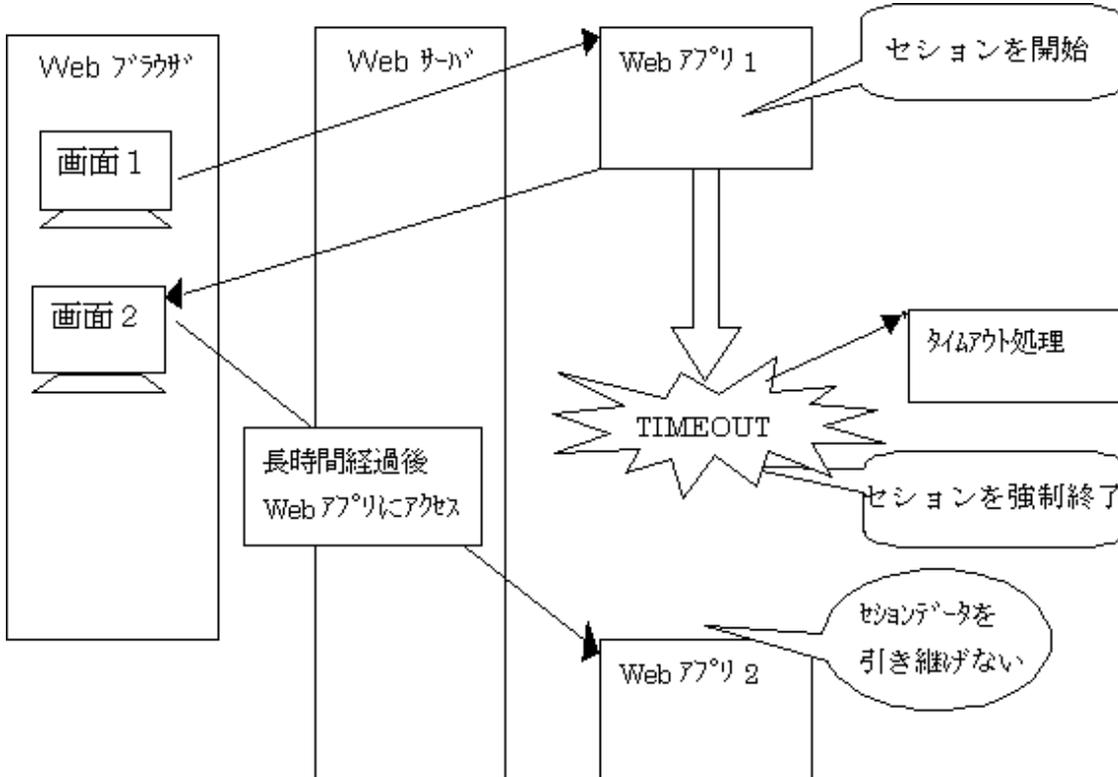
セッション管理機能を利用して、セッションを開設することにより、特定のクライアント(Webブラウザ)との接続状態を維持することができます。また、同一セッション内では、セッションデータを利用して前のページで入力されたデータを次のページに引き継ぐことができるため、複数の画面(ページ)が遷移する業務プログラムを簡単に作成できます。



セッションの開設中に、画面操作中の長時間にわたる離席や、Webブラウザを閉じてしまうなどの理由で業務が中断された場合、セッションのタイムアウトが発生します。

タイムアウトとは、同一セッション内でWebアプリケーションがWebブラウザにレスポンスを返してから、Webブラウザによって次のリクエストが発行されるまでの経過時間が規定時間に達することを言います。

タイムアウトが発生した時点で、セッション管理はセッションを強制終了します。このため、タイムアウトが発生すると、Webアプリケーションの資産はセッションが中断された状態のままになってしまいます。セッション管理機能では、このような状態に陥った場合の対応手段として、タイムアウト処理の登録機構を提供しています。



## 注意

セッションを使用したWebアプリケーションを、SUBMITボタン(INPUTタグのTYPE属性に"SUBMIT"を指定したボタン)のダブルクリックなどの操作によって二重に起動した場合、Webアプリケーションの動作は不定となります。このため、Webブラウザ上でJavaScriptなどを使用して二重起動を抑制してください。なお、Webブラウザを操作する上での一般的な注意点については、“[O.1.4 Webブラウザ操作上の注意点と対策](#)”を参照してください。

## I.8.2 Webサブルーチンのインタフェース

Webサブルーチンを利用する際のインタフェースについて説明します。

### 書き方

Webサブルーチンを呼び出す場合の書き方について説明します。

### 呼出し時のデータ設定

Webサブルーチンを呼び出す前に設定する情報(呼出しパラメタ)について説明します。

条件名が記述されているものは、SET文で値を設定してください。

条件名に下線がある場合は、その値が初期値であることを意味します。

なお、呼出し時に設定するデータの値は、Webサブルーチンを呼び出した前後で変化しません。

### 処理結果のデータ

Webサブルーチンの実行により得られる処理結果について説明します。

条件名が記述されているものは、条件名条件で判定してください。

また、Webサブルーチンでの異常および警告内容はCOBW3-STATUSで参照できます。

値	意味
0	正常終了
0 以外	エラー番号

エラー番号は、“[I.8.3 エラー処理](#)”を参照してください。

また、ISAPIサブルーチンの実行結果はPROGRAM-STATUSで参照できます。

値	意味
0	正常終了
-1	異常終了

## 参考

Webサブルーチンの実行が正常終了した場合、特殊レジスタPROGRAM-STATUSに0を返します。異常および警告がある場合には、特殊レジスタPROGRAM-STATUSに-1を返します。

## 注意

各サブルーチンを呼び出す前に、そのサブルーチンの呼出し時のデータ設定で公開されたデータ以外の領域を変更するような処理が行われた場合、動作およびその領域の内容は保証されません。

また、“COBW3\_INIT”を呼び出す前には、インタフェース領域のCOBW3をLOW-VALUEで初期化してください。

## 登録集

Webサブルーチンの呼出しパラメタおよび実行後の復帰パラメタのインタフェースとして登録集を提供しています。Webサブルーチン呼び出す場合には、作業場所節に以下の登録集をCOPY文で取り込んでください。

この登録集は、COBOL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

COBW3.cbl

### 書き方

次のように指定します。

```
WORKING-STORAGE SECTION.  
COPY COBW3.
```

## 注意

WebサブルーチンおよびWebサブルーチン固有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文の記述は行わないでください。

登録集を変更した場合、動作は保証されません。

## Webサブルーチンの呼出し

Webサブルーチンの環境設定およびWebパラメタの獲得のため、最初に“COBW3\_INIT”を呼び出す必要があります。また、Webサブルーチンの処理を終了するとき(手続きの最後)には、必ず“COBW3\_FREE”を呼び出し、Webサブルーチンの資源を解放する必要があります。

その他のサブルーチンについては必要に応じてCALL文で呼び出してください。

“COBW3\_INIT”を呼び出したプログラムとは異なるプログラムで、他のサブルーチン呼び出す場合、“COBW3\_INIT”を呼び出したプログラムで定義したインタフェース領域(COBW3)をプログラム間で持ち回り、その領域を使って他のサブルーチン呼び出してください。

## 例

親プログラムで“COBW3\_INIT”を呼び出し、子プログラムBで“COW3\_FREE”を呼び出す場合

- 親プログラム

```
IDENTIFICATION DIVISION.  
PROGRAM-ID      "HttpExtensionProc".  
DATA            DIVISION.  
WORKING-STORAGE SECTION.  
COPY COBW3.  
LINKAGE        SECTION.  
COPY ISAPICTX.  
PROCEDURE DIVISION WITH STDCALL LINKAGE USING ISAPI-CTX-CNT.  
MOVE LOW-VALUE TO COBW3.  
MOVE FUNCTION ADDR (ISAPI-CTX-CNT) TO COBW3-CONTEXT.  
CALL "COBW3_INIT" USING COBW3.  
:  
CALL "B"        USING COBW3.  
*  
EXIT PROGRAM.
```

- 子プログラム

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.    B.  
DATA          DIVISION.  
LINKAGE       SECTION.  
COPY COBW3.
```

```

PROCEDURE      DIVISION USING COBW3.
:
CALL "COBW3_FREE" USING COBW3.
EXIT PROGRAM.

```

## Webサブルーチンの注意事項

- WebサブルーチンはHTTP/1.0に対応しています。
- Webサブルーチンで使用するデータの定量制限は以下のとおりです。

項目	値
Webパラメタの最大長(注1)	1073741822バイト
WebパラメタのNAMEの最大文字列長	1024バイト
WebパラメタのVALUEの最大文字列長	1024バイト
COBW3-HTML-FILENAMEに指定する結果出力用ページ(ファイル)の最大レコード長(注2)	1024バイト
変換データを置き換えた後の結果出力用ページの1行の長さ	3072バイト

### 注1

WebパラメタはFORMタグのENCTYPEの指定により、データの形式が異なります。WebパラメタにはNAMEおよびVALUEの他、ファイルアップロードのデータなども含まれます。また、WebパラメタはWebサーバのあるマシンのメモリ上に獲得されるため、使用可能なメモリが少ない場合は、データのサイズが定量制限を超えていない場合でもデータを受け取れないことがあります。このような場合は、必要に応じて呼出し用ページにアップロード可能なファイルのサイズを表示したり、制限を超えるファイルをアップロードしないように注意書きを表示したりするなどの対処を行ってください。なお、Webパラメタのデータ形式の詳細は、HTMLについて解説している書籍またはホームページなどを参照してください。

### 注2

市販のHTMLエディタを使用して作成されたHTML文書は、1行のバイト数が1024バイトを超える可能性があります。この場合、メモ帳などのテキストエディタを使用して1024バイトを超えないように調整してください。

## 1.8.3 エラー処理

Webサブルーチン実行時に問題が検出された場合には、Webサブルーチンがエラー処理を行い、メッセージを出力します。

出力するメッセージの形式

```
COB-メッセージ番号 : COBW3 : メッセージ本文
```

COB-メッセージ番号

メッセージの通し番号を表示します。

メッセージ本文

エラーの内容を表示します。



### 参考

Webサブルーチンのログの設定を行っておくと、同様の内容がログに出力されます。

表1.4 出力されるメッセージ番号およびメッセージ本文

メッセージ番号	メッセージ本文
00100	COBW3-CONTEXTに値が設定されていないため、処理が続行できませんでした。

メッセージ番号	メッセージ本文
01100	何らかの理由により作業領域、オブジェクトが破壊されたと考えられます。領域の使い方およびWebサブルーチンのオブジェクトを点検してください。
01500	呼出し元のCOBOLアプリケーション動作コード系ではWebサブルーチンは正しく動作しません。
01501	呼出し元のアプリケーションの動作状態に誤りがあります。Webサブルーチンは正しく動作しません。
02050	検索文字列の文字列長に負の値が指定されました。0が指定されたものとみなして処理を続行します。
02051	検索文字列の文字列長に制限を超える値が指定されました。最大長が指定されたものとみなして処理を続行します。
03000	POST,GET以外でWebアプリケーションが呼び出されました。
03001	COBW3_INITが2回以上呼び出されました。2回目の呼び出しの前には、必ずCOBW3_FREEを呼び出してください。
03002	Webサブルーチンの作業領域を獲得できませんでした。
03003	GETで渡されたデータの長さが制限を超えました。最大長までを有効とします。
03020	POSTまたはGETで渡されたデータの獲得に失敗しました。
03021	POSTまたはGETで渡されたデータのサイズが制限を超えているため、データを獲得できませんでした。
03030	Webサブルーチンの作業領域の獲得に失敗しました。
03040	アップロードファイル情報を取得できませんでした。
03041	ヘッダ情報の取得に失敗したため、Webパラメタを処理することができません。
03042	POST以外のメソッドに対して、FORMタグのENCTYPEにmultipart/form-dataが指定されているため、Webパラメタおよびアップロードファイル情報を取得できませんでした。
03065	環境変数@CBR_WEB_OUT_CODEの設定に誤りがあります。変換コードの指定は行われなかったものとみなします。
03200	UTF-8からUCS-2への変換に失敗しました。
03201	UCS-2からUTF-8への変換に失敗しました。
03202	SJISからUCS-2への変換に失敗しました。
03203	SJISからUTF-8への変換に失敗しました。
03700	文字列として識別できない文字が指定されました。そのまま処理を行います。
03701	EUC/SJIS混在の文字列が存在しました。そのまま処理を行います。
03850	Cookieの取得に失敗しました。
03900	日本語コード変換時にコード変換エラーが発生しました。
03901	日本語コード変換時にコード変換(iconv_close)でエラーが発生しました。
03902	日本語コード変換ライブラリがロードできません。
03903	日本語コード変換関数(iconv)が呼び出せません。
04000	Webパラメタがありません。
04001	Webパラメタ長が0です。
04006	検索対象NAME/VALUEの出現回数値に0または負の値が指定されました。1が指定されたものとみなして処理を続行します。
04008	VALUEデータの長さが制限を超えました。最大長までを有効とします。
04400	変換名が指定されていません。変換名を確認してください。
04401	変換名の文字列長に負の値が指定されました。0が指定されたものとみなして処理を続行します。
04402	変換名の文字列長に制限を超える値が指定されました。最大長が指定されたものとみなして処理を続行します。

メッセージ番号	メッセージ本文
04450	指定された変換情報は登録されていないため、削除できませんでした。
04470	指定された変換情報は登録されていないため、変更できませんでした。
04480	指定された変換情報の削除に失敗しました。
05000	システムコマンドの実行に失敗しました。
05001	システムコマンドが指定されていません。
05500	ファイル名が指定されていません。ファイル名を確認してください。
05501	アップロードファイル情報がありません。
05510	アップロードファイルのクライアントでのパス名の取得に失敗しました。
05511	アップロードファイルのクライアントでのパス名の文字列長が制限を超えました。最大長までを有効にします。
05520	アップロードファイルのクライアントでのファイル名の取得に失敗しました。
05521	アップロードファイルのクライアントでのファイル名の文字列長が制限を超えました。最大長までを有効にします。
05530	アップロードファイルのコンテンツタイプの取得に失敗しました。
05531	アップロードファイルのコンテンツタイプの文字列長が制限を超えました。最大長までを有効にします。
05550	指定されたファイル名はすでに存在しているため、ファイルは生成しませんでした。
05551	ファイルの生成に失敗しました。
05552	サブルーチン内部でのデータ生成に失敗しました。生成したファイルをCOBW3_DEL_UPLOADFILEで削除することはできません。
05580	指定されたファイル名は同一スレッド内で生成されたファイルではありません。ファイルの削除は行いませんでした。
05581	指定されたファイルの削除に失敗しました。
06100	指定されたHTML文書ファイルのOPENでエラーが発生しました。
06101	指定されたHTML文書ファイルのREADで入出力エラーが発生しました。
06102	指定されたHTML文書ファイルのサイズは0です。
06105	HTML文書中の変換名に対応する変換情報が存在しません。変換データを登録してください。
06106	HTML文書の出力作業領域が不足したため出力に失敗しました。変換名に対応する変換データを確認してください。
06107	HTML文書中の変換名の指定形式に誤りがあるため、出力に失敗しました。
06108	指定された変換名の文字列長が長すぎます。変換名を確認してください。
06120	HTML文書をWWWサーバへ出力するのに失敗しました。
06130	HTML文書中の一行の文字数が制限を超えました。
06140	Webアプリケーションの動作コード系がSJISの場合、扱えるHTML文書ファイルの文字コードはSJISのみです。
06300	変換名が指定されていません。変換名を確認してください。
06301	変換名の文字列長に負の値が指定されました。0が指定されたものとみなして処理を続行します。
06302	変換名の文字列長に制限を超える値が指定されました。最大長が指定されたものとみなして処理を続行します。
06303	変換文字列が指定されていません。変換文字列を設定してください。
06304	変換文字列の文字列長に負の値が指定されました。0が指定されたものとみなして処理を続行します。
06305	変換文字列の文字列長に制限を超える値が指定されました。最大長が指定されたものとみなして処理を続行します。
06310	指定された変換名は既に登録されています。既に登録されている変換情報を有効とします。
06320	変換情報の登録作業領域が不足したため登録に失敗しました。変換情報を初期化した後、再度登録を行ってください。

メッセージ番号	メッセージ本文
06330	変換情報の削除に失敗しました。
06502	ブラウザへのデータ出力中に予期せぬエラーが発生しました。
06503	出力する文字列長に負の値が設定されています。0が設定されたものとみなして処理を続けます。
06504	出力する文字列長に制限を超える値が指定されています。最大長が指定されたものとみなして処理を続行します。
06700	既にCOBW3_PUT_HTMLまたはCOBW3_PUT_TEXTが呼び出されています。ヘッダの出力要求は無効です。
06702	COBW3-PUT-HEAD中にコロン(:)がありません。COBW3-PUT-HEADの指定は無視されます。
06704	COBW3-PUT-HEAD-LENGTHに負の値が指定されました。0が指定されたものとみなして処理を続行します。
06706	COBW3-PUT-HEAD-LENGTHに制限を超える値が指定されました。最大長が指定されたものとみなして処理を続行します。
06708	ヘッダとして出力する対象が指定されていません。
06710	既に出力されている値と異なる値がCOBW3-CONTENT-TYPEに指定されました。
06714	既に出力されている値と異なる値がCOBW3-STATUS-CODEに指定されました。
06718	既に指定されている値と異なる値がCOBW3-CONTENT-LENGTHに指定されました。
06730	Content-typeの宣言で予期せぬエラーが発生しました。
06734	Status-codeの宣言で予期せぬエラーが発生しました。
06738	Content-lengthの宣言で予期せぬエラーが発生しました。
06742	ヘッダの出力に失敗しました。
06746	ヘッダの出力を完了できませんでした。
07010	ユーザ名が取得できませんでした。
07011	ユーザ名の文字列長が制限を超えました。最大長までを有効にします。
07020	パスワードが取得できませんでした。
07021	パスワードの文字列長が制限を超えました。最大長までを有効にします。
07030	IPアドレスが取得できませんでした。
07031	IPアドレスの文字列長が制限を超えました。最大長までを有効にします。
07040	認証タイプが取得できませんでした。
07041	認証タイプの文字列長が制限を超えました。最大長までを有効にします。
07500	セッションは開始されていません。
07510	既にCOBW3_PUT_HTMLまたはCOBW3_PUT_TEXTが呼び出されているため、セッションは開始できませんでした。
07520	既にCOBW3_PUT_HTMLまたはCOBW3_PUT_TEXTが呼び出されているため、セッションは終了できませんでした。
07558	タイムアウト監視が開始できませんでした。
07562	何らかの理由でタイムアウト監視に失敗しました。
07566	タイムアウト監視の中断に失敗しました。
07570	何らかの理由でタイムアウト監視の準備が出来ませんでした。セッションは開始できませんでした。
07580	タイムアウト処理が中断できませんでした。
07584	Cookieデータから取得したセッションIDが正しくありません。
07588	セッション情報の管理域の獲得に失敗しました。
07590	タイムアウト監視の実行に失敗しました。

メッセージ番号	メッセージ本文
07600	既にセッションは開始されています。
07602	タイムアウト時間に0が指定されたため、セッションは開始できませんでした。
07604	タイムアウト時間に制限を超える値が指定されました。最大時間が指定されたものとみなして処理を続行します。
07606	セッションデータタイプに無効な値が指定されたため、セッションは開始できませんでした。
07608	セッション用のCookieデータの作成に失敗したため、セッションは開始できませんでした。
07610	セッションデータの大きさに0が指定されたため、セッションデータの登録は行えませんでした。
07614	セッションデータの大きさに制限を超える値が指定されました。最大値が指定されたものとみなして処理を続行します。
07618	セッションデータとして指定されたオブジェクトが正しくありません。
07650	指定されたセッションデータの大きさと登録されているセッションデータの大きさが異なるため、セッションデータの取得は行えませんでした。
07660	セッションデータにnull以外の値が設定されているため、セッションデータの取得を行えませんでした。
07670	セッションデータが登録されていません。
07680	タイムアウト時間に0が指定されたため、タイムアウト時間は変更できませんでした。
08004	COBW3-HEADER-NAME-LENGTHに負の値が指定されました。0が指定されたものとみなして処理を続行します。
08005	script-nameが取得できませんでした。
08006	COBW3-HEADER-NAME-LENGTHに制限を超える値が指定されました。最大長が指定されたものとみなして処理を続行します。
08008	COBW3-HEADER-NAMEが指定されていません。
08010	HTTPヘッダの取得に失敗しました。
08020	COBW3-HEADER-VALUEの長さが制限を超えました。最大長までを有効にします。
08024	COBW3-HEADER-NAMEに指定されたHTTPヘッダは見つかりませんでした。
08030	script-nameの文字列長が制限を超えました。最大長までを有効とします。
08101	キーワードが指定されていません。キーワードを設定してください。
08105	インタフェースファイル中に指定されたキーワードが見つかりませんでした。
08108	インタフェースファイル名が取得できませんでした。
08110	インタフェースファイルのOPENでエラーが発生しました。
08114	インタフェースファイルのREADでエラーが発生しました。
08118	インタフェースファイルのCLOSEでエラーが発生しました。
08130	キーワードに対応する値の長さが制限を超えました。値の最大長までを獲得します。
08150	Webサブルーチン内部でのデータ入出力でエラーが発生しました。
08500	既にCOBW3_PUT_HTMLまたはCOBW3_PUT_TEXTが呼び出されています。Cookieに対する操作は無効です。
08510	Cookie名が指定されていません。Cookie名を設定してください。
08512	Cookie名の文字列長に負の値が指定されました。0が指定されたものとみなして処理を続行します。
08514	Cookie名の文字列長に制限を超える値が指定されました。最大長が指定されたものとみなして処理を続行します。
08520	Cookie値が指定されていません。Cookie値を設定してください。
08522	Cookie値の文字列長に負の値が指定されました。0が指定されたものとみなして処理を続行します。
08524	Cookie値の文字列長に制限を超える値が指定されました。最大長が指定されたものとみなして処理を続行します。
08530	指定されたCookieデータは登録されていないため、変更できませんでした。

メッセージ番号	メッセージ本文
08532	指定されたCookie名は既に登録されています。既に登録されているCookieデータを有効とします。
08540	Cookieの有効期限として1582年10月15日以前の日付は無効です。有効期限は指定されなかったものとみなします。
08541	Cookieの有効期限における年の指定が正しくありません。有効期限は指定されなかったものとみなします。
08542	Cookieの有効期限における月の指定が正しくありません。有効期限は指定されなかったものとみなします。
08543	Cookieの有効期限における日の指定が正しくありません。有効期限は指定されなかったものとみなします。
08544	Cookieの有効期限における時の指定が正しくありません。有効期限は指定されなかったものとみなします。
08545	Cookieの有効期限における分の指定が正しくありません。有効期限は指定されなかったものとみなします。
08546	Cookieの有効期限における秒の指定が正しくありません。有効期限は指定されなかったものとみなします。
08550	Cookieのセキュリティに無効な値が指定されました。セキュリティは指定されなかったものとみなします。
08560	Cookieの登録種別に無効な値が指定されました。登録種別は指定されなかったものとみなします。
08570	Cookieデータの登録作業領域が不足したため登録に失敗しました。Cookie情報を初期化した後、再度登録を行ってください。
08580	Cookie情報のエンコードに失敗しました。
08610	指定されたCookieデータは登録されていないため、削除できませんでした。
08620	指定されたCookieデータの削除に失敗しました。
08710	Cookie情報の初期化種別に無効な値が指定されました。初期化種別は指定されなかったものとみなします。
08720	Cookie情報の登録作業域が不足したため、リクエストデータのCookie情報による初期化に失敗しました。
08810	Cookie値の長さが制限を超えました。最大長までを有効とします。
08820	クライアントからCookieデータが送信されませんでした。
09001	COBW3-REQUEST-INFO-TYPEに正しい値が指定されていません。
09010	リクエストの取得に失敗しました。
09020	COBW3-REQUEST-INFOの長さが制限を超えました。最大長までを有効にします。
09500	COBW3_CNV_SETはUnicode環境では使用できません。
09501	COBW3_CNV_DELはUnicode環境では使用できません。
09502	COBW3_NAMEはUnicode環境では使用できません。
09503	COBW3_NAMEN はUnicode環境では使用できません。
09504	COBW3_VALUE はUnicode環境では使用できません。
09505	COBW3_SET_COOKIEはUnicode環境では使用できません。
09506	COBW3_DEL_COOKIEはUnicode環境では使用できません。
09507	COBW3_GET_COOKIEはUnicode環境では使用できません。
09508	COBW3_GET_COOKIEEN はUnicode環境では使用できません。
20000	COBOLアプリケーションのロードに失敗しました。
20001	設定された環境設定ファイルが存在しません。
20002	実行するCOBOLアプリケーションが指定されていません。
20003	実行しようとしたCOBOLアプリケーション処理の開始アドレスの取得に失敗しました。
20004	ディレクタで原因不明のエラーが発生しました。
20005	ディレクタの環境設定処理が異常終了しました。
20006	メモリの割り当てに失敗しました。

メッセージ番号	メッセージ本文
20007	環境設定の記述に誤りがあります。
20008	環境設定のファイル名に誤りがあります。
20009	開放するCOBOLアプリケーションが指定されていません。
99999	Can't open message catalog.

## 付録J コマンド

ここでは、NetCOBOLが提供するコマンドおよびNetCOBOLから使用するコマンドを説明します。

### NetCOBOLが提供するコマンド

- “J.1 COBOL32コマンド”
- “J.4 COBOLファイルユーティリティコマンド”
- “J.5 INSDBINFコマンド”
- “J.6 UTF-32用定義体変換コマンド”

### NetCOBOLから使用するコマンド

- “J.2 LINKコマンド”
- “J.3 DUMPBINコマンド”

## J.1 COBOL32コマンド

COBOL32コマンドは、翻訳操作をNetCOBOL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 …

- 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。

### オペランドの説明

#### オプションの並び

翻訳オプションを指定します。

指定する内容は、“[J.1.2 翻訳コマンドのオプション一覧](#)”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.1 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。

### ポイント

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

#### 指定例

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

## 出力形式

COBOL32コマンドによるNetCOBOL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、NetCOBOL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。

### 通常の実出力形式

翻訳オプションMESSAGEを指定しない場合、通常の実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。

#### NetCOBOLコマンドプロンプト

```
c:¥COBOL>COBOL32 -M test.cob
** 診断メッセージ ** (SAMPLE1)
test.cob 8: JMN2503I-S 利用者語 'A' が定義されていません.
test.cob 8: JMN2557I-S DISPLAY文の書き方が不完全です.
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:¥COBOL>
```

### 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リストも出力されます。



標準出力に出力される翻訳メッセージの中にUnicode固有文字が含まれる場合、Unicode固有文字は“\_”に置き換えられて表示されます。

## J.1.1 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

## J.1.2 翻訳コマンドのオプション一覧

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

- “J.1.3 -A (サブスキーマ定義ファイルのフォルダの指定)”
- “J.1.10 -dr (リポジットファイルの入出力先フォルダの指定)”
- “J.1.13 -I (登録集ファイルのフォルダの指定)”
- “J.1.16 -m (画面帳票定義体ファイルのフォルダの指定)”
- “J.1.19 -R (リポジット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

- “J.1.8 -dp (翻訳リストファイルのフォルダの指定)”

- ・ “J.1.18 -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

- ・ “J.1.11 -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

- ・ “J.1.7 -do(オブジェクトファイルのフォルダの指定)”
- ・ “J.1.15 -M(主プログラムを翻訳するときの指定)”
- ・ “J.1.17 -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

- ・ “J.1.4 -Dc(COUNT機能を使用する指定)”
- ・ “J.1.5 -dd(デバッグ情報ファイルのフォルダの指定)”
- ・ “J.1.6 -Dk(CHECK機能を使用する指定)”
- ・ “J.1.9 -Dr(TRACE機能を使用する指定)”
- ・ “J.1.12 -Dt(デバッグ機能および診断機能を使用する指定)”

### その他

- ・ “J.1.14 -i(オプションファイルの指定)”
- ・ “J.1.20 -v(各種情報を出力する指定)”
- ・ “J.1.21 -WC(翻訳オプションの指定)”

## J.1.3 -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.3.1 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。

## J.1.4 -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、“21.5 COUNT機能”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.3.9 COUNT (COUNT機能の使用の可否)”

## J.1.5 -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

“J.1.12 -Dt(デバッグ機能および診断機能を使用する指定)”

“A.3.60 TEST(デバッグ機能の使用の可否)”

## J.1.6 -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、“21.3 CHECK機能”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.3.5 CHECK (CHECK機能の使用の可否)”

## J.1.7 -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。



-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。



“A.3.40 OBJECT(目的プログラムの出力の可否)”

## J.1.8 -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。



```
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。



```
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。



-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。



“J.1.18 -P(各種翻訳リストの出力および出力先の指定)”

“A.3.42 PRINT (各種翻訳リストの出力の可否および出力先の指定)”

## J.1.9 -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、“21.4 TRACE機能”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.3.62 TRACE (TRACE機能の使用の可否)”

## J.1.10 -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.3.45 REP (リポジトリファイルの入出力先フォルダの指定)”

## J.1.11 -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.3.48 SAI(ソース解析情報ファイルの出力の可否)”

## J.1.12 -Dt(デバッグ機能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、対話型デバッガ、NetCOBOL Studioのデバッグ機能および診断機能で使用するデバッ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。

対話型デバッガについては、“第24章 対話型デバッガの使い方”を参照してください。

NetCOBOL Studioのデバッグ機能については、“NetCOBOL Studio ユーザーズガイド”を参照してください。

診断機能については、“22.1 診断機能”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

“J.1.5 -dd(デバッグ情報ファイルのフォルダの指定)”

“A.3.60 TEST(デバッグ機能の使用の可否)”

## J.1.13 -I(登録集ファイルのフォルダの指定)

-I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]“1.2.1 環境変数の設定”



## 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.3.25 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。

## J.1.14 -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

## J.1.15 -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.3.30 MAIN(主プログラム/副プログラムの指定)”

## J.1.16 -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.3.21 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。

## J.1.17 -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F 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.3.41 OPTIMIZE (広域最適化の扱い)”

### J.1.18 -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

### J.1.19 -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。



## 参照

- -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.3.46 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。

### J.1.20 -v (各種情報を出力する指定)

-v

次の情報を標準出力に出力する場合に指定します。

- COBOL32コマンドのバージョン情報



## 参照

-vだけを指定し、これ以外のオプション、ソースファイル名、オブジェクトファイル名などを指定しなかった場合、COBOL32コマンドはバージョン情報のみを出力して正常終了します。このとき、COBOL32コマンドの復帰コードは0です。[参照]“J.1.1 COBOL32コマンドの復帰コード”

### J.1.21 -WC (翻訳オプションの指定)

-WC, “翻訳オプションの並び”

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,"翻訳オプションの並び"」の形式で指定することができます。翻訳オプションの並びには翻訳オプションを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。翻訳オプションの並びの長さは最大234バイトです。

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。

翻訳オプションの指定方法によるオプションの優先順位は、“[A.1 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
  - FORMEXT
  - LIB
  - PRINT
  - REP
  - REPIN
  - LIBEXT
  - 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

## J.2 LINKコマンド

ここでは、LINKコマンドの形式について説明します。なお、ここで説明している内容は、LINKコマンドの機能の一部です。詳細な説明は、Microsoft社の公式サイトを参照してください。

### 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
- MSVCRT.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。

## J.2.1 LINKコマンドのオプション

表J.1 LINKコマンドのオプション

指定形式	内容
/DEBUG(注1)	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。
/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します。
/DEF:モジュール定義ファイル名	モジュール定義ファイルを指定します。
/EXPORT:外部参照名	外部参照情報を生成します。
/OUT:filename	メイン出力ファイルの名前を指定します。
/STACK:スタックサイズ(注2)	スタックサイズの変更を指定します。
/MAP:filename	マップファイルを生成する場合に指定します。
/DYNAMICBASE:NO(注3)	アドレス空間ランダム化(ASLR)を無効にします(必須)
/NOENTRY	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からC関数を呼び出さない場合のみ)
/INCREMENTAL:NO(必須)	インクリメンタル・リンクを無効にします(注3)
/SUBSYSTEM:WINDOWS,バージョン番号 /SUBSYSTEM:CONSOLE,バージョン番号	サブシステムを指定します。 バージョン番号 例: 5.01や6.00など

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、“[22.1.2.3 プログラムの翻訳・リンク](#)”および“[24.2.3 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](#)”

注3: NetCOBOL開発環境(NetCOBOLコマンドプロンプト、NetCOBOL Studio、プロジェクトマネージャ)を使用しない場合、必ず指定してください。NetCOBOL開発環境では既定で指定します。



#### 注意

V11.0.0以降で使用するLINKコマンドに“/DEBUGTYPE”は指定できません。デバッグ情報は常に“.pdb”に出力されます。

## J.3 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J.2 DUMPBINコマンドのオプション”を参照してください。

表J.2 DUMPBINコマンドのオプション

指定形式	内容
/EXPORTS	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。
/IMPORTS	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。
/ALL	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。
/OUT:出力ファイル名	実行結果の出力先ファイル名を指定します。

### 参考

DUMPBINコマンドの使用法の説明は、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

## J.4 COBOLファイルユーティリティコマンド

COBOLファイルユーティリティコマンドは、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、コマンドで行います。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。

COBOLファイルユーティリティの詳細は、“7.8 COBOLファイルユーティリティ”を参照してください。



- 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。
- 戻り値  
COBOLファイルユーティリティコマンドの実行が成功した場合は0を返します。  
失敗した場合は-1を返します。  
コマンドの実行が失敗した場合は、出力されるメッセージに従って対処してください。

### J.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 { sjis  
    ebc dica  
    ebc dick  
    unicode[, { LE } ]  
    utf32[, { LE } ] }
```

sjis : ASCII/シフトJIS  
 ebcdica : JEF(EBCDIC/ASCII)  
 ebcdick : JEF(EBCDIC/KANA)  
 unicode : UCS-2  
 utf32 : UTF-32

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-32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。
- -kutf32を指定した場合、テキストファイルの文字コードはUTF-32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-32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UTF-32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。

## 注意

レコード順ファイルで、エンディアン(UCS-2またはUTF-32)が混在したレコードは指定できません。

## 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

---


$$-c \left\{ \begin{array}{c} b \\ t \end{array} \right\}, \left\{ \begin{array}{l} \text{allchar} \\ \text{alltxtbin} \\ " \left\{ \begin{array}{c} \text{c長さ} \\ \text{t長さ} \\ \text{k長さ} \end{array} \right\} [\dots]" \end{array} \right\}$$


---

-kunicodeまたは-kutf32を指定した場合

---


$$-c \left\{ \begin{array}{c} b \end{array} \right\} \left\{ \begin{array}{l} \text{allucs2} \\ \text{allutf8} \\ \text{allutf32} \end{array} \right\}$$


---

$$t \quad , \quad " \left\{ \begin{array}{l} u\text{長さ} \\ f\text{長さ} \\ t\text{長さ} \\ w\text{長さ} \end{array} \right\} [ \dots ] "$$

## 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進形式といいます。
- 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進形式といいます。
- 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/UTF-32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進形式といいます。

## 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。

### allxtbin

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。

### allutf32

レコード中の全データをUTF-32形式とみなします。

### "{u長さ|f長さ|t長さ|w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/UTF-32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  
w : UTF-32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 -ooutfile.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 -kebdica -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
```

## J.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  $\left\{ \begin{array}{c} S \\ R \\ I \end{array} \right\}$  ,  $\left\{ \begin{array}{c} f \\ v \end{array} \right\}$  ,レコード長,

"(オフセット,長さ[N],[N32]/ オフセット,長さ[N],[N32] …)[,D][;…]"

#### 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。

#### N32

レコードキーとするデータ項目がUTF-32(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	:	ixdfilename	
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 -oixdfilename -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。

## J.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。

$$-i \left\{ \begin{array}{c} S \\ R \\ I \end{array} \right\}, \left\{ \begin{array}{c} f \\ v \end{array} \right\}, \text{レコード長}$$

#### 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
```

## J.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
     { ebc dica
       { ebc dick
         { unicode[, { LE } ]
           { BE } ]
         { utf32[, { LE } ]
           { BE } ]
       }
     }

```

sjis : ASCII/シフトJIS

ebcdica : JEF(EBCDIC/ASCII)

ebcdick : JEF(EBCDIC/KANA)

unicode : UCS-2

utf32 : UTF-32

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UCS-2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。
- `-kutf32`を指定した場合、ファイルの文字コードをUTF-32コード体系として扱います。  
カンマ(,)に続き、UTF-32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。

---


$$-ps \left\{ \begin{array}{l} \text{格納順番} \\ r \text{ 相対レコード番号} \\ ic \text{ レコードキー値} \\ it \text{ レコードキー値} \\ in \text{ レコードキー値} \end{array} \right\}$$


---

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。

一 索引ファイルでは、以下のキーワード文字列に続き、レコードキー値を指定します。

"ic" : レコードキー値を文字形式で指定する場合に指定します。

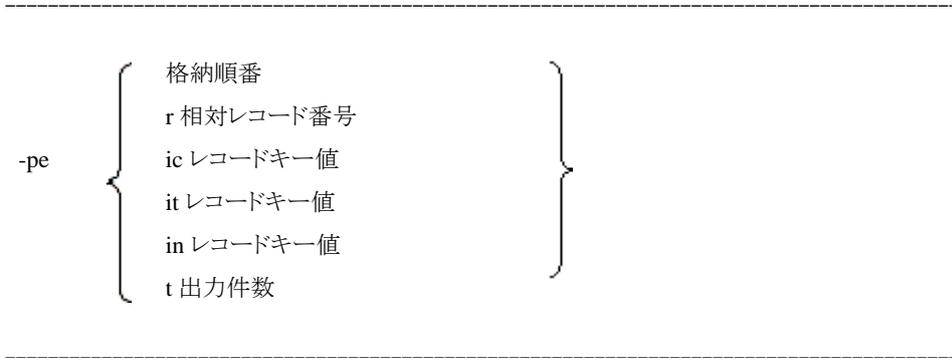
"it" : レコードキー値を16進形式で指定する場合に指定します。

"in" : すべて日本語項目のレコードキーに対し、レコードキー値を文字形式で指定する場合に指定します。

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。

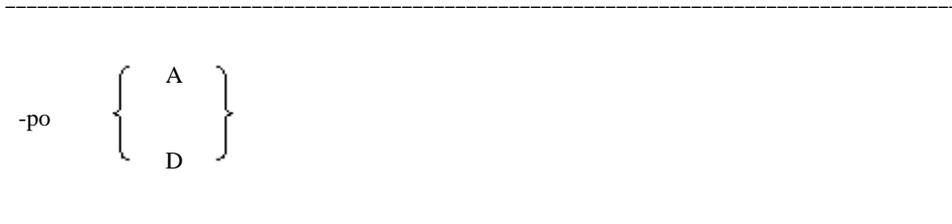


出力件数の指定以外は、開始位置と同じです。終了位置を出力件数で指定する場合は、キーワード文字“t”に続けて出力レコード件数を指定します。

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。



#### 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 : ixdfile  
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 -pk1 ixdfile
```



## 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は、英数字/日本語が混在するレコードキーに対しても指定可能です。  
ただし、Unicodeビッグエンディアン形式またはUTF-32形式を含むレコードキーに対して、指定することはできません。
- psinおよび-peinは、UTF-32形式を含むレコードキーに対して、指定することはできません。

## J.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 { sjis  
    { ebc dica  
    { ebc dick  
    { unicode[, { LE } ]  
    {          { BE } ]  
    { utf32[, { LE } ]  
    {         { BE } ] }  
-----
```

sjis : ASCII/シフトJIS

ebcdica : JEF(EBCDIC/ASCII)

ebcdick : JEF(EBCDIC/KANA)

unicode : UCS-2

utf32 : UTF-32

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

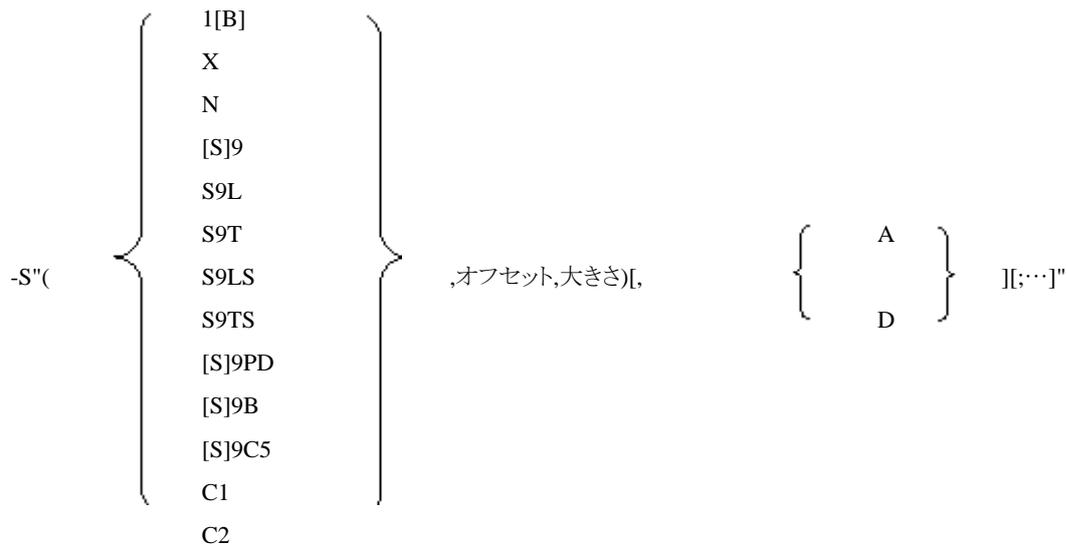
- -kオプションを省略した場合または-ksjisを指定した場合、整列するファイルの文字コードをASCII/シフトJISコード体系として扱います。
- -kebcdicaまたは-kebcdickを指定した場合、整列するファイルの文字コードをJEF(EBCDIC/KANA)コード体系またはJEF(EBCDIC/ASCII)コード体系として扱います。
- -kunicodeを指定した場合、整列するファイルの文字コードをUCS-2コード体系として扱います。カンマ(,)に続き、UCS-2コード体系のエンディアンを指定します。省略した場合には、LE(リトルエンディアン形式)が指定されたものとして扱います。
- -kutf32を指定した場合、ファイルの文字コードをUTF-32コード体系として扱います。カンマ(,)に続き、UTF-32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。



 **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

 **注意**

キー項目属性に”N”を指定した場合、データ項目の文字コードは-k指定に従います。異なる文字コードのキー項目を指定することはできません。

**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

- 相対ファイルを整理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
整理条件       : [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
                  整理順序               : 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
                  整理順序               : 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
  
```

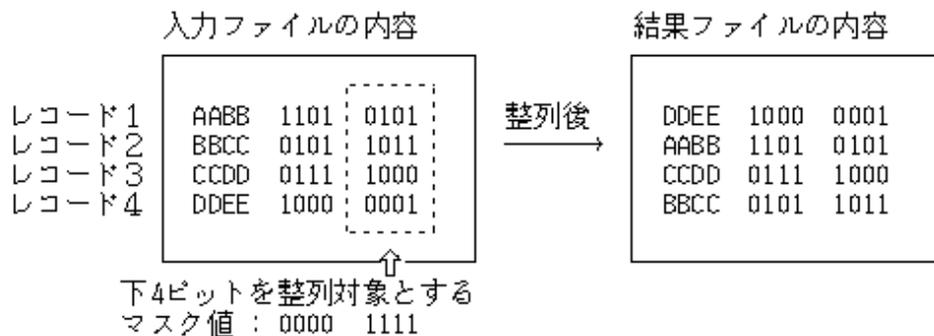
```
cobfsort -ooutfile -s"(N,0,2),A:(X,10,5),D"-iR,v,80 relfile
```

- レコード順ファイルを整理してレコード順ファイルに出力する

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.
  01 データ1 PIC X(4).
  01 データ2 PIC 1(8) BIT.
  
```



```

入力ファイル名 : infile
出力ファイル名 : outfile
整理条件       : [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
                  整理順序               : 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
  
```

```
cobfsort -ooutfile -s"(1B,4,15),A"-iS,f,5 infile
```

## J.4.6 ファイル属性コマンド(cobfattr)

索引ファイルの属性情報(レコード長、レコード形式、キー情報など)を表示します。

索引ファイル以外の属性情報は表示できません。

## 指定方法

```
cobfattr 入力ファイル名
```

## インタフェース

### 入力ファイル名

属性を表示する索引ファイルのパス名を指定します。



### 例

#### 指定例

- 索引ファイルの属性を表示する

入力ファイル名 : ixdfile

```
cobfattr ixdfile
```

## J.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。

### 未復旧データファイル名

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

未復旧データファイル名 : 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。
- データの状態によっては、未復旧データファイルに出力されない場合があります。

## J.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 -ooutfile 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。

## J.5 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。

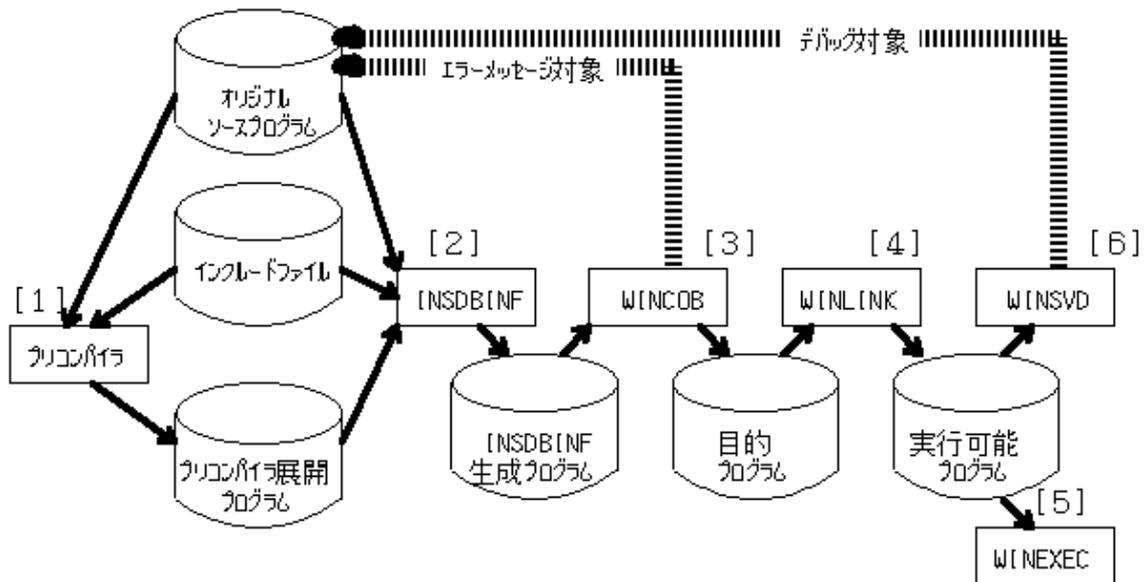
ファイル名制御情報(#FILE情報)は、オリジナルソースプログラムのファイル名や、プリコンパイラがインクルードしたファイルのファイル名を通知するための情報です。

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となります。

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NONNUMBER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、“[23.4.8 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、以下を参照してください。

表J.3 INSDBINFコマンドのオプション

指定形式	内容
-I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が存在する場合、-I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、-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、-I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。
-S拡張子[拡張子] …	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Noneを指定します。-S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。
-O出力ファイル名	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。-O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。
-C	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)。
-D	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)。
-U	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。

### 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。

## J.6 UTF-32用定義体変換コマンド

UTF-32用定義体変換コマンドは、FORMまたはPowerFORMを使用して作成した帳票定義体(.smd/.pmd)をUTF-32で扱う帳票定義体に変換するコマンドです。

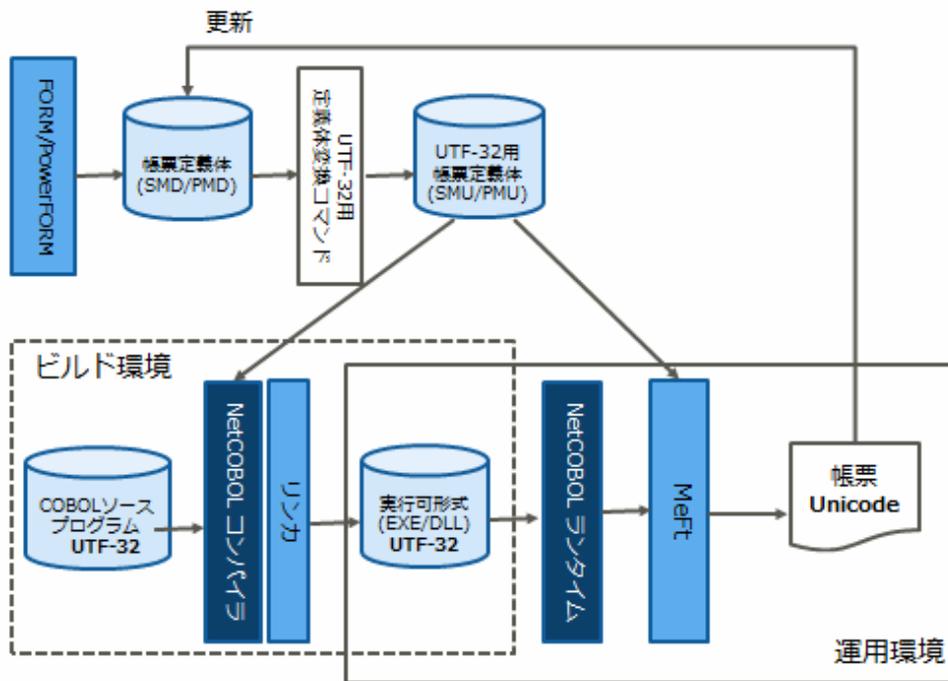
FORMまたはPowerFORMを使用して作成した帳票定義体は、日本語項目を1文字=2バイトで扱います。COBOLプログラムで日本語項目のエンコードをUTF-32とした場合は、1文字=4バイトで扱わなければプログラムと整合しません。UTF-32用定義体変換コマンドでは、帳票定義体に定義された日本語項目のデータ長の扱いをUTF-32用に変換した帳票定義体を出力します。

日本語項目のエンコードをUTF-32としたFORMAT句付き印刷ファイル/表示ファイル(PRT)を使用するCOBOLプログラムの翻訳では、UTF-32用定義体変換コマンドで変換した帳票定義体を入力にしてください。

UTF-32用定義体変換コマンドを使用した場合の帳票定義体を使用する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

UTF-32用定義体変換コマンドを使用した場合の帳票定義体を使用するプログラム開発の標準的な手順を以下に示します。



## 入力形式

コマンド	オペランド
CNVMED2UTF32	変換元ファイル名 [D 出力先ディレクトリ名] [C] [P] [Y]



## 注意

UTF-32用定義体変換コマンドは、FORM/PowerFORM V11以降のインストールディレクトリに格納されます。

## 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。

表J.4 UTF-32用定義体変換コマンドのオプション

指定形式	内容
変換元ファイル名	変換元の帳票定義体のファイル名を指定します。ファイル名にアスタリスクを指定して複数ファイルを指定することもできます。ただし、サブディレクトリのファイルは変換対象にはなりません。ファイル名に空白を使用する場合には""(ダブルクォーテーション)で囲んでください。
/D 出力先ディレクトリ	変換後の帳票定義体を出力するディレクトリを指定します。省略した場合は、変換元ファイルと同じディレクトリに出力します。
/C	変換時にエラーが発生しても処理を続行します。
/P	エラーメッセージを表示しません。
/Y	変換後のファイル名と同一ファイル名が存在する場合、上書きして出力します。

## 変換元ファイル

変換の対象となるファイルはFORMまたはPowerFORMで作成された帳票定義体です。以下の条件のいずれかに該当する場合、変換コマンドはエラーとなり、変換は失敗します。

- FORMまたはPowerFORMで作成された帳票定義体でないファイル

- 定義体版数が17版を超える帳票定義体
- 画面定義体
- 不完全な定義体
- UTF-32用定義体(変換後の定義体)

#### 出力ファイル

変換後の出力ファイルは、UTF-32用帳票定義体です。出力ファイルの拡張子は以下のようになります。

- 変換元ファイルがFORMで作成された帳票定義体の場合  
定義体名.SMD → 定義体名.SMU
- 変換元ファイルがPowerFORMで作成された帳票定義体の場合  
定義体名.PMD → 定義体名.PMU

#### 注意

- 出力された帳票定義体の拡張子は変更できません。
- 出力された帳票定義体は、日本語項目のエンコードをUTF-32としたCOBOLプログラムの翻訳・実行で使用します。
- 出力された帳票定義体を日本語項目のエンコードをUTF-32以外としたCOBOLプログラムの入力とした場合、翻訳エラーまたは実行時エラーになります。
- 出力された帳票定義体は、FORM/PowerFORMで更新できません。エンハンス等で帳票定義体の更新が必要な場合は、変換元の帳票定義体を更新し、UTF-32用定義体変換コマンドで再変換してください。

#### エラー一覧

メッセージは標準出力に表示します。

復帰値	メッセージ	対処方法
0	“入力ファイル”を正常に変換しました。	なし
-1	パラメタに誤りがあります。 使い方: CNVMED2UTF32.exe ファイル名 [/D 出力先ディレクトリ名] [/C] [/P] [/Y]	パラメタを確認してください。
-2	メモリが不足しています。	不要なアプリケーション等を終了させてから、再度実行してください。
-101	“入力ファイル”の読み込み時にエラーが発生しました。	入力ファイルを格納したデバイスの状態を確認してください。
-102	“入力ファイル”が存在しません。	ファイル名の指定を確認してください。
-103	“入力ファイル”へのアクセスが拒否されました。	入力ファイルのアクセス権を確認してください。
-104	“入力ファイル”は帳票定義体ではありません。	帳票定義体を指定してください。
-105	“入力ファイル”は未サポートの定義体のため変換できません。[詳細コード:??]	変換対象の条件に該当しているか確認してください。
-106	“入力ファイル”は画面定義体のため変換できません。	帳票定義体を指定してください。
-107	“入力ファイル”は未完成定義体のため変換できません。	FORM/PowerFORMで帳票定義体(完成定義体)にしてください。
-110	“入力ファイル”は項目のレコード領域長が65535を超えるため変換できません。	定義した項目のレコード領域長を調整してください。
-201	“出力ディレクトリ”が存在しません。	出力ディレクトリの指定を確認してください。

復帰値	メッセージ	対処方法
-202	出力先に、“出力ファイル名”が存在しているため出力できません。	出力ディレクトリのファイルを確認してください。上書きして出力する場合は、/Yオプションを指定してください。
-204	“出力ファイル”へのアクセスが拒否されました。	出力ファイルおよび出力ディレクトリのアクセス権を確認してください。
-205	“出力ファイル”の書き込み時にエラーが発生しました。	出力ディレクトリのディスク空き容量やデバイスの状態を確認してください。
-206	“出力ファイル”はパスが長すぎます。	出力ディレクトリを短いパスのディレクトリに変更してください。

## 付録K 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

### K.1 機能比較一覧

OSIV系システムのCOBOL85と本システムのNetCOBOLの機能比較を以下に示します。

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K.1 OSIV系システムと本システムの機能比較

機能分類		機能概要	比較	備考	
分類					
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 SHIFT-OUT		●	
		PROGRAM-STATUS RETURN-CODE		□	属性が異なるOSIV系システム： S9(4)BINARY 本システム：S9(9)COMP-5
		上記以外		○	
機能名	SYSPUNCH STACKER-01～12 CSP S01～02 SYSPCH BUSHU SOKAKU ON-YOMI KUN-YOMI		●		

機能分類		比較	備考		
分類	機能概要				
		SWITCH-8 SYSERR	△		
		CHANNEL02 ~12 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 日本語英数字定数 日本語連想定数 日本語ひらがな定数	×		
		16進文字定数 日本語16進定数 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: 翻訳オプション APOST/QUOTE に従う 本システム: 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 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 BINARY-SHORT BINARY-LONG BINARY-DOUBLE	△		
		上記以外	○		
式	算術式	2項演算子 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 正負条件 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			
		上記以外	○	
	基本命令	中核機能の文すべて	○	
順ファイル	環境定義	APPLY WRITE-ONLY句 MULTIPLE FILE TAPE句 RERUN句 PASSWORD句 RESERVE AREA句	●	
		ASSIGN句のデータ名指定 ASSIGN句のDISK指定 ASSIGN句のPRINTER 指定 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 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 RERUN句	●	
		ASSIGN句のデータ名指定 ASSIGN句のDISK指定 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 UNLOCK文	△	
		上記以外	○	
索引ファイル	環境定義	PASSWORD句 RERUN句	●	
		ASSIGN句のデータ名指定 ASSIGN句のDISK指定 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				
		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 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 SOKAKU ON-YOMI 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 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 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 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- 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	○		
	特殊レジスタ	○		
	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	CUURRENT-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“NetCOBOLユーザーズガイド(OSIV分散開発編)”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。

## K.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。

## K.2.1 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。

## K.2.2 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は、“第24章 対話型デバッガの使い方”を参照してください。

## K.2.3 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

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ユーザーズガイド”の“付録E FORM 画面帳票定義体 OS 別留意事項”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L 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  
NO } )

---

画面帳票定義体の項目名の展開を指定します。

#### ALL

画面帳票定義体をプログラミング名で展開します。

#### MED

画面帳票定義体をプログラミング名で展開します。

#### NO

画面帳票定義体を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M 文字コードの留意点

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
- 文字コードに関するコーディング上の留意点



### 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。

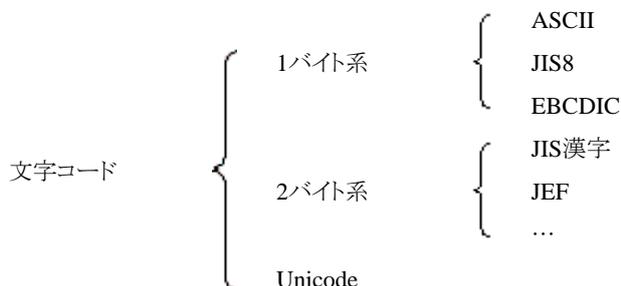
## M.1 文字コードの概説

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。

- バックスラッシュの代わりに“¥”を含む
- カナ文字を追加している

## EBCDIC

元々はIBM社の考案したコード系で、英大文字、数字、制御文字および少数の記号を含みます。文字の割り当ての一部が任意の割り当てを許すようになっており、使用する国、地域、用途などからさまざまな変種が存在します。日本国内では主に次のようなものが使われます。

### EBCDIC(カナ)

カナ文字と日本固有の記号を追加したもの

### EBCDIC(ASCII)

英小文字追加したもの

### EBCDIC(英小文字)

英小文字と日本固有の記号を追加したもの

2バイト系は日本語を表現するために十分な文字を表現するために用意されたものです。漢字以外にカタカナ、ひらがな、英字、数字、その他の記号を多数含みますが、総称して漢字コードと呼ばれます。

## JIS漢字

JIS X 0208で規定される漢字コードです。1978年に制定され、その後、2度の改定が行われていますが、1983年の改定は400近い文字についての変更が行われたため、この改定より前を78JIS、改定以降を83JISと言って区別する場合があります。

## JEF

78JISを元にEBCDICとの混在使用に適するように作られた富士通固有の漢字コードで、次の特徴を持ちます。

- JIS漢字に含まれない多くの漢字を含みます。
- 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。
- 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6章 文字コード](#)”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方法は文字種の判定の方法と使用可能な文字種によってさまざまな変種が存在します。

ここでは、3つに分けて説明します。

### シフトJIS (SJIS)

PCで広く利用されているコード系です。英数字・カナ文字(JIS8)、日本語文字(JIS漢字)を混在させる方法で、次のような特徴を持ちます。

- 文字種の切り換えにシフトコードを使用しません。
- 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。
- 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

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。

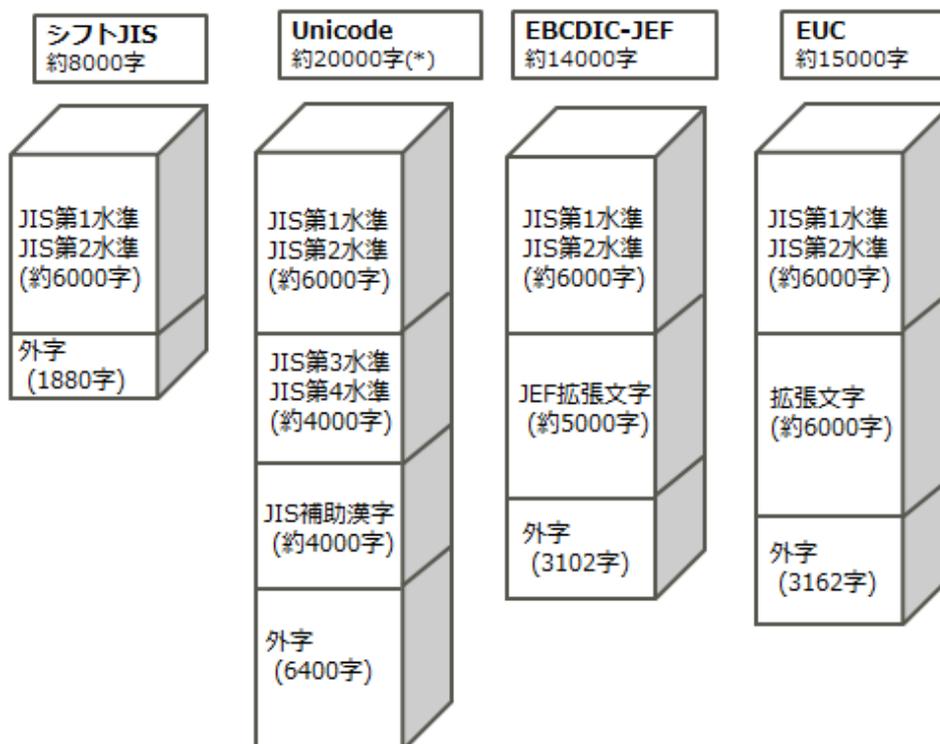
## EBCDIC-JEF

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。



## 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

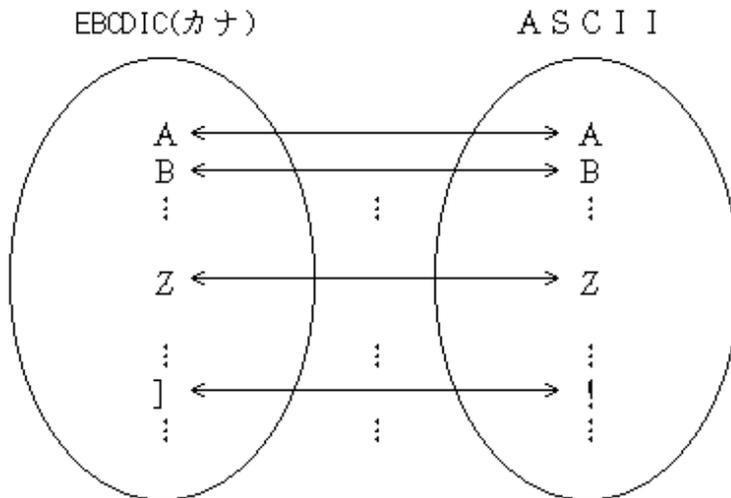
変換先のコード系に含まれる文字の種類が変換元のコード系に含まれる文字の種類より少ない場合、または単純な包含関係が成り立たない場合です。

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。

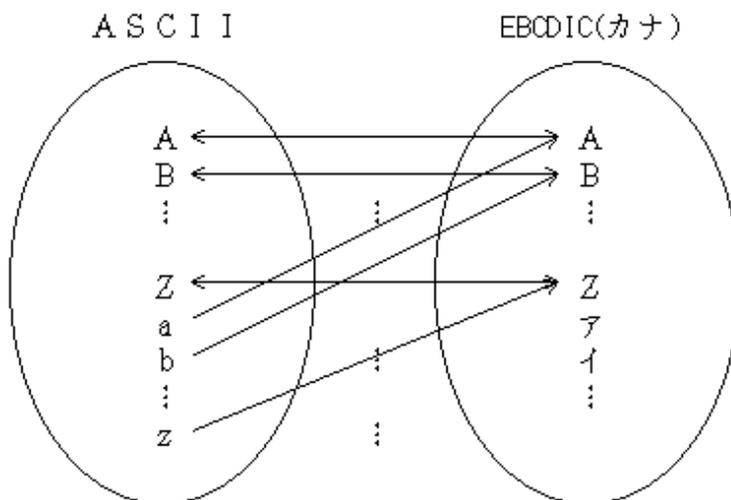
例えばEBCDICからASCIIへの変換をした場合、“!”が“]”に変換されますが、逆方向の変換をすることによって、元に戻すことができます。



#### 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。

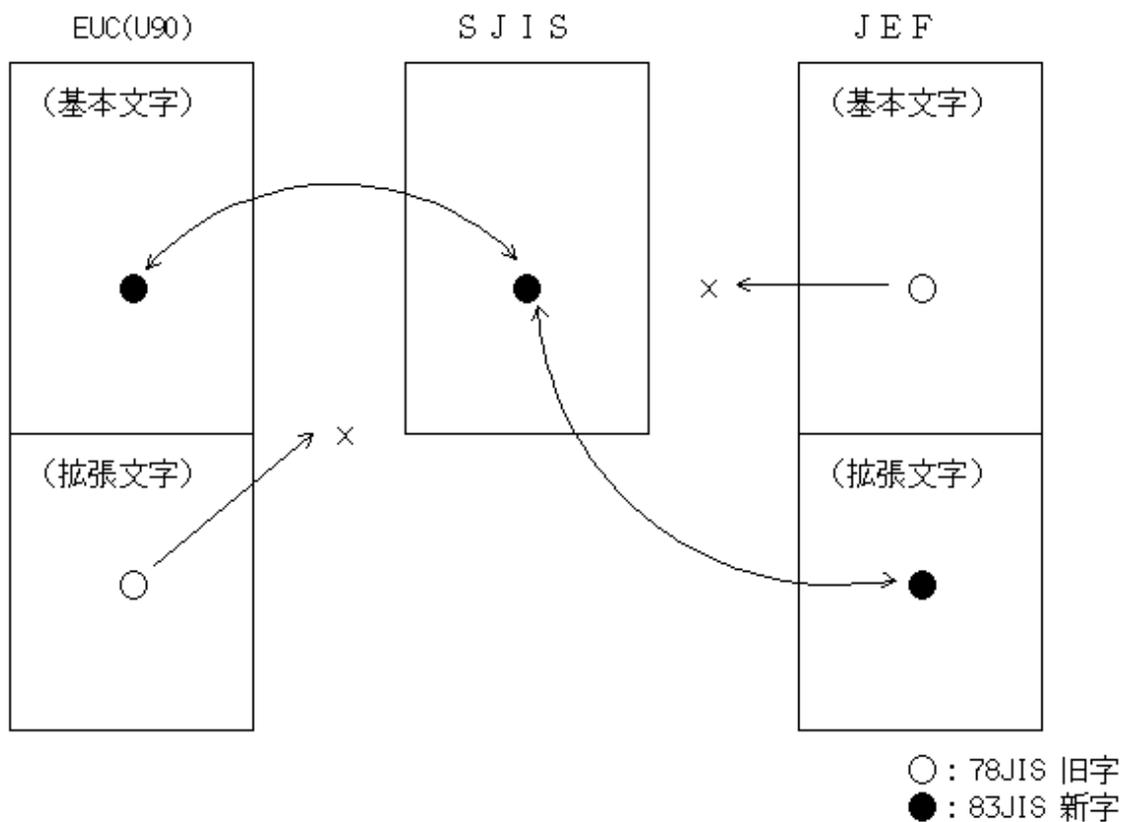
### EBCDICからシフトJISへの変換時の代替文字による変換

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：EBCDIC				変換先：SJIS	
コード	ASCII	英小文字	カナ	コード	字形
0x4F	“ ! ”	“   ”	“   ”	0x21	“ ! ”
0x4A	“ [ ”	“ 夫 ”	“ 夫 ”	0x5B	“ [ ”
0x5A	“ ] ”	“ ! ”	“ ! ”	0x5D	“ ] ”
0x5B	“ \$ ”	“ \$ ”	“ ¥ ”	0x23	“ \$ ”
0x5F	“ ^ ”	“ ~ ”	“ ~ ”	0x5E	“ ^ ”
0xA1	“ ~ ”	“ — ”	“ — ”	0x7E	“ — ”
0xE0	“ / ”	“ \$ ”		0x5C	“ ¥ ”

### 78JISと83JISで非互換のある文字に対する変換

JEFおよびEUC(U90)は78JIS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。

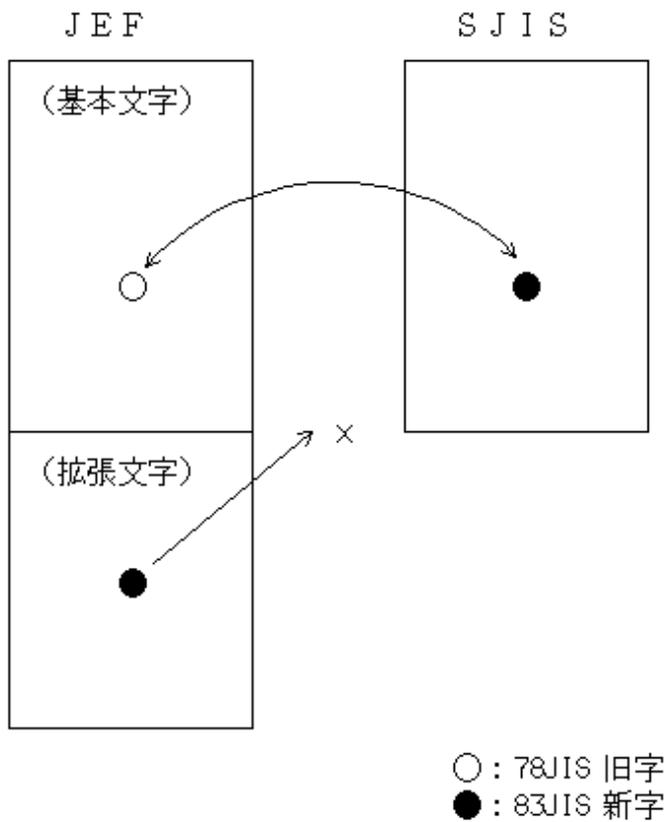


変換前のソースプログラム、データファイル等がEUC(U90)の場合、78JISの旧字体の文字は拡張文字セットに含まれるものであるため、このような変化が行われても問題となることはあまりありません。

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

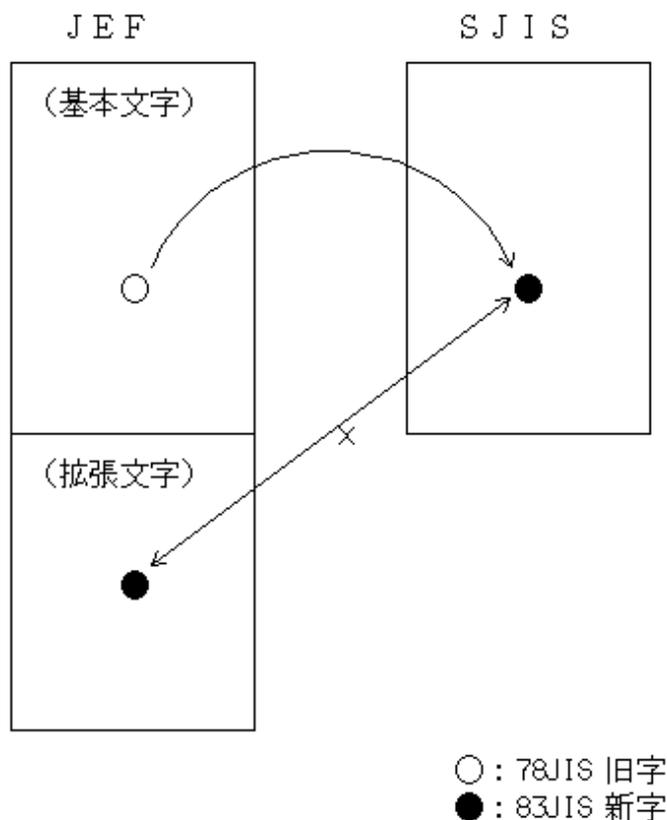
78JISによる旧字と83JISによる新字の違いを無視して、次のように変換を行います。



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います。このため、あまりお勧めできる方法ではありません。しかし、OSIVからのアプリケーションの移植などで字体の違いを強く意識しないような場合は、この方法での変換が効率的な解決方法になる場合もあります。

## 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" または NX"00003000"	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、EBCDIC-JEFコード系以外の場合は "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。翻訳オプションNSPCOMPについては、“[A.3.38 NSPCOMP\(日本語空白の比較方法の指定\)](#)”を参照してください。

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)や異なるエンコードの日本語項目が含まれる集団項目または翻訳オプションによって決定されるエンコードと異なる日本語項目の比較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 SPACE	文字定数	日本語定 数
				日本語項目あり		日本語項目なし		英数字 項目	日本語 項目			
				非表示項 目 あり	非表示項 目 なし	非表示項 目 あり	非表示項 目 なし					
作用対象1	データ項目	集団項目	日本語項目あり	非表示項目あり	非表示項目なし	非表示項目あり	非表示項目なし	英数字項目	日本語項目	表意定数SPACE	文字定数	日本語定数
			非表示項目なし	●	●	●	●	●	●	●	●	●

					作用対象2								
					データ項目				定数				
					集団項目				基本項目		表意定数 SPACE	文字定数	日本語定 数
					日本語項目あり		日本語項目なし		英数字 項目	日本語 項目			
					非表示項 目 あり	非表示項 目 なし	非表示項 目 あり	非表示項 目 なし					
		日本語項目なし	非表示項目あり	—	●	—	—	—	—	●			
			非表示項目なし	—	●	—	—	—	—	—	●		
基本項目	目	英数字項			●	—	—	—	ERR	—	—	ERR	
	目	日本語項	●	●	●	●	ERR	●	●	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

```

### エンコードが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

```

- ・ エンコードが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。

## M.5 文字コードに関するコーディング上の留意点

言語要素および機能ごとに、文字コードに関するコーディング上の留意点について説明します。

本項のプログラム例では、環境部で以下の符号系名が宣言されているとみなします。

```

ALPHABET.
  U8  FOR ALPHANUMERIC IS UTF8
  U16L FOR NATIONAL    IS UTF16LE

```

```
U16B FOR NATIONAL IS UTF16BE
U32L FOR NATIONAL IS UTF32LE
U32B FOR NATIONAL IS UTF32BE.
```

## M.5.1 日本語16進文字定数

日本語16進文字定数は、作用対象のエンコードに合わせて、日本語文字を16進数で記述します。

ただし、作用対象のエンコードがリトルエンディアンの場合、日本語16進文字定数はビッグエンディアンで記述してください。コンパイラが受取り側のエンディアンに合わせて変換した値を格納します。

```
WORKING-STORAGE SECTION.
01 DATA1 PIC N(3) ENCODING IS U16L VALUE NC"富士通".
01 DATA2 PIC N(3) ENCODING IS U32L VALUE NC"富士通".
:
IF DATA1 = NX"5BCC58EB901A" THEN DISPLAY "OK!!".
IF DATA2 = NX"00005BCC000058EB0000901A" THEN DISPLAY "OK!!".
```

## M.5.2 表意定数

表意定数は作用対象の字類によって決まります。

たとえば、表意定数SPACEは、作用対象の字類が英数字の場合はASCII空白(半角空白)となり、日本語の場合は日本語空白(全角空白)になります。コード値は、作用対象のエンコードに従います。

## M.5.3 文字の泣き別れ

英数字項目に日本語を格納する場合、シフトJISの場合は2桁/1文字、UTF-8の場合は2~4桁/1文字の領域が必要となります。転記や比較、部分参照などで1文字が分断され、不正な文字データとなる場合があります。このように1文字が分断された状態を文字の泣き別れと言います。コーディングの際は、泣き別れが発生しないよう、データの操作に注意が必要です。

日本語項目の場合、シフトJISでは桁数=文字数が成立するため泣き別れの考慮は不要でしたが、UTF-16ではサロゲートペアは2桁/1文字の領域に格納されるため、泣き別れの可能性があります。泣き別れのデータを作らないためには、以下の点に注意してください。

- ・ 十分な領域を用意する
- ・ 字類条件BMPやUNICODE1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、UTF-32はシフトJISと同様に桁数=文字数が成立するため、泣き別れの考慮は不要です。

## M.5.4 日本語文字定数

日本語文字定数には日本語文字を指定します。日本語文字の定義は作用対象のエンコードによって異なります。

たとえば、シフトJISでは日本語文字にASCII文字(いわゆる半角の英数字)は含まれませんが、Unicodeには含まれます。よって、作用対象のエンコードがUTF-16またはUTF-32の場合、全角と半角が混在する日本語文字定数を定義することができます。

```
MOVE NC"沼津市宮本140番地 コーポ富士通 B-5" TO ADDR. ...[1]
MOVE NC"沼津市宮本140番地 コーポ富士通 B-5" TO ADDR. ...[2]
```

上記の例で、[1]の場合、データ項目ADDRのエンコードがシフトJISでもUnicodeでも定義できますが、[2]の場合、シフトJISでは翻訳時エラーとなります。

## M.5.5 項目の再定義

シフトJISの場合、英数字項目と日本語項目のエンコードは同じため、日本語項目を英数字項目で再定義(REDEFINES句)する、またはその逆の使い方が可能でしたが、Unicodeの場合は注意が必要です。

```
WORKING-STORAGE SECTION.
01 PERSON.
02 AGE PIC 9(3).
02 NAME PIC N(8) ENCODING IS U16L.
02 NAME-X REDEFINES NAME PIC X(16) ENCODING IS U8.
```

```

01 TMP-NAME      PIC X(16).
:
MOVE NAME-X TO TMP-NAME.
DISPLAY TMP-NAME.      *> Unicodeでは文字化けします。

```

Unicodeでは、日本語項目と英数字項目のエンコードは異なります。このため、再定義によって同一のデータを異なるエンコードで参照する場合、文字化けなど意図しない動作をすることがあります。このような場合は、作用対象のエンコードに合わせてデータを変換してください。データ変換には、組み込み関数DISPLAY-OF/NATIONAL-OFを用いる方法と、MOVE文(書き方3)を用いる方法があります。

```

MOVE DISPLAY-OF(NAME) TO TMP-NAME.  *> DISPLAY-OF関数を使用、または、
MOVE CONV NAME TO TMP-NAME.        *> MOVE文のCONV指定を使用します。
DISPLAY TMP-NAME.                  *> Unicodeの場合でも文字化けしません。

```

## M.5.6 集団項目転記

シフトJISの場合、英数字項目と日本語項目のエンコードは同じため、日本語項目を含む集団項目を英数字項目に転記するような使い方が可能ですが、Unicodeの場合は注意が必要です。

```

WORKING-STORAGE SECTION.
01 PERSON.
02 AGE      PIC 9(3).
02 NAME     PIC N(8) ENCODING IS U16L.
01 TMP-AREA PIC X(80) ENCODING IS U8.
:
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。

## M.5.7 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。シフトJIS、Unicodeとも、日本語転記の空白づめには全角空白が使用されます。

Unicodeの場合、日本語項目に半角空白を格納できるため、注意が必要です。

```

WORKING-STORAGE SECTION.
01 PIC-N     PIC N(10) ENCODING IS U16L VALUE NC" Fujitsu".    ...[1]
01 PIC-X     PIC X(10) ENCODING IS U8.
:
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は見えますが、実際には偽が成立します。PIC-Nには3文字の全角空白が詰められる[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詰め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めることで回避できます。

なお、文字比較についても、同様な注意が必要です。

## M.5.8 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では英数字字類と日本語字類でエンコードが異なるため、注意が必要です。

```

WORKING-STORAGE SECTION.
01 DATA-X.
02 NAME     PIC X(6) ENCODING IS U8 VALUE "日本".    *> X"E697A5E69CAC"
01 DATA-N.

```

```

02 NAME PIC N(2) ENCODING IS U16L 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。

## M.5.9 大小比較

比較については、同じ字類どうしても、エンコードが異なる場合は翻訳時エラーとなります。エンコードを合わせて比較してください。

なお、Unicodeの場合、日本語項目を含む集団項目を大小比較に用いる場合に注意が必要です。

```

WORKING-STORAGE SECTION.
01 GRP.
  02 SMALL PIC N(1) ENCODING IS U16L VALUE NC"あ". *> X"3042"
  01 LARGE PIC N(1) ENCODING IS U16L VALUE NC"A". *> X"FF21"
:
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リトルエンディアンのまま、右辺は日本語項目のためビッグエンディアンに変換された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。

集団項目を大小比較に用いる場合は、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。

## M.5.10 プログラムの作成・編集

- 正書法では、1行の最大長は可変長形式および自由形式の場合は251バイト、固定長形式の場合は80バイトです。これは表示長ではなく物理長です。

Unicode(Utf-8)でプログラムを作成する場合、表示長より物理長が大きくなるため、1行の最大長を考慮してコーディングしてください。1行に収まらない場合は、継続行を利用するなどして行を分割してください。

1行の最大長を超えた場合、コンパイラは超えた部分を無視して翻訳を続行します。その結果、発生する現象のひとつとして以下を確認しています。

- 一 以下の翻訳時エラーが出力される。

```
cobol:ERROR: システムエラー' errno=0x016' が' iconv_error' で発生しました。
```

- 利用者語にはサロゲートペアの文字は使用できません。

## M.5.11 半角カナ

COBOL文字集合に半角カナは含まれません。このため、COBOLの語、たとえば利用者語などに半角カナは使用できません。COBOL文字集合に規定されている文字(たとえば全角カナなど)に修正してください。

## M.5.12 小入出力

ACCEPT文、DISPLAY文を使用してデータを入出力できますが、Unicodeにおいて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) ENCODING IS U16L.
  02 TEL PIC 9(10).
:

```

DISPLAY PERSONAL-DATA. ...[1]
DISPLAY NAME TEL. ...[2]

Unicodeは字類によってエンコードが異なるため、日本語が含まれる集団項目をDISPLAY文で表示する場合、文字化けが発生します[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]。

なお、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、データは実行時コード系の英数字項目のエンコードで格納されます。

また、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。

## M.5.13 COBOLファイル

COBOLファイルの留意事項は、以下を参照してください。

- “7.1.4 Unicodeデータの扱い”(COBOLファイル全般)
- “8.1.14 Unicodeの印刷について”(印刷ファイル)

## M.6 Windowsシステムにおけるシステムロケール

Windowsシステムにおけるシステムロケールは、Unicodeを用いないプログラムの既定の文字セット(文字、記号、および数値)とフォントを決定します。システムロケールの変更手順について、Windows 7を例に説明します。

1. [コントロールパネル]の[地域と言語]を選択します。
2. [管理] タブをクリックし、[Unicode 対応ではないプログラムの言語] の [システム ロケールの変更] をクリックします。
3. 言語を選択して、[OK] をクリックします。

設定を有効にする場合、システムの再起動が必要です。

NetCOBOLでは、翻訳オプションSCS(ACP)およびRCS(ACP)を指定した場合、この規定の文字セット(以降では、ANSIコードページと呼びます)が使用されます。

システムロケールが日本語の場合、ANSIコードページはシフトJISとなります。

システムロケールが日本語以外の環境で、COBOLアプリケーションを翻訳・実行する場合の注意事項を、翻訳オプションの指定ごとに説明します。

- SCSオプション
  - SCS(ACP)を指定、または、SCSオプションを省略した場合、ソースファイルの文字コードはANSIコードページとみなします。システムロケールを変更した場合、ソースファイルの文字コードの解釈も変更されます。
  - 日本語利用者語/日本語定数は、ソースファイルの文字コードがシフトJIS(SCS(ACP))かつシステムロケールが日本語、または、SCS(SJIS))の場合に使用できます。
- RCSオプション
  - RCS(ACP)を指定、または、RCS/ENCODEオプションを省略した場合、実行時コード系はANSIコードページになります。COBOLアプリケーションが入出力する資源の文字コードは、ANSIコードページです。  
システムロケールが日本語以外の環境では、英数字項目のエンコードはASCIIとして動作します。日本語項目の動作は保証されません。また、ENCODING句を指定することはできません。
  - RCS(SJIS)はシステムロケールが日本語以外の場合には指定できません。翻訳・実行ともにエラーとなります。
- ENCODEオプション

ENCODEオプションを指定した場合、実行時コード系はUnicodeとなります。システムロケールの変更は影響ありません。ただし、ENCODE(SJIS)オプションを指定する場合、ソースファイルの文字コードはシフトJISを指定します。特にシステムロケールが日本語以外の環境では、SCS(SJIS)を明示指定する必要があります。

システムロケールが日本語以外の場合、翻訳オプションSCSと翻訳オプションENCODEの組合せ可否について以下に示します。

	ENCODE(SJIS,SJIS)	ENCODE(UTF8,UTF16)	ENCODE(UTF8,UTF32)
<b>SCS(ACP)</b>	×	○	○
<b>SCS(SJIS)</b>	○	○	○
<b>SCS(UTF8)</b>	×	○	○

○:組合せ可能、×:組合せ不可

### 注意

ENCODEオプションを省略した場合、RCSオプションが有効となります。

	<b>RCS(ACP)</b>	<b>RCS(SJIS)</b>	<b>RCS(UTF16)</b>
<b>SCS(ACP)</b>	△(省略時)	×	○
<b>SCS(SJIS)</b>	△	×	○
<b>SCS(UTF8)</b>	×	×	○

△:組合せ可能。ただし、英数字項目のエンコードはASCIIとして動作。

## 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 ユーザーズガイド”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。

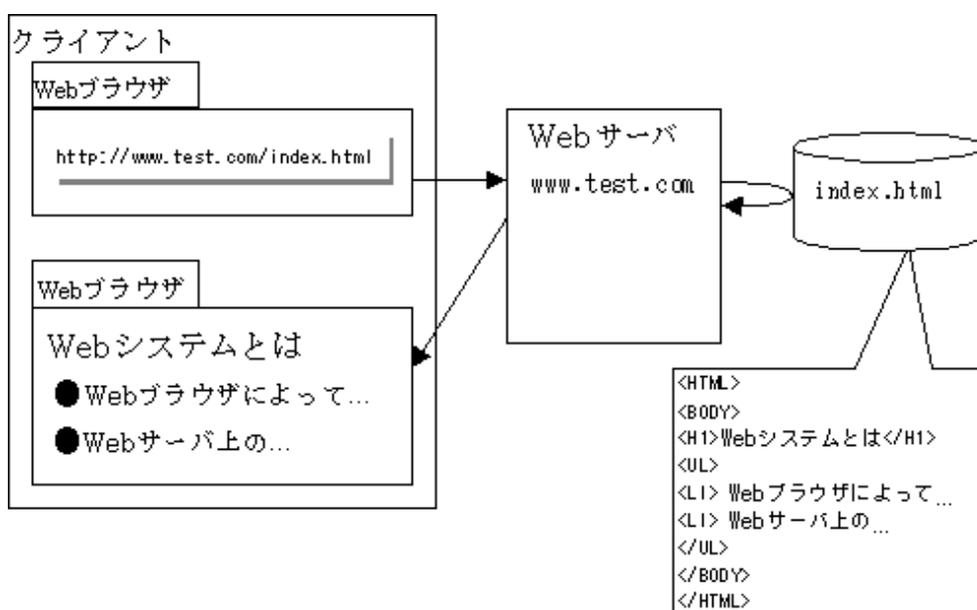
## 付録O Webアプリケーション

### 0.1 Web上で動作するCOBOLアプリケーションをはじめて作成する方のために

ここでは、Webを利用したアプリケーションをはじめて構築する方または、はじめにCOBOL Webサブルーチンを利用する方のために基本的な事柄について説明します。

#### 0.1.1 Webアプリケーションとは

インターネット/イントラネットなどで利用するWWW(World Wide Web)は、ネットワーク上のWebサーバとWebブラウザ(クライアント)の間をHTTP(HyperText Transfer Protocol)を使用して接続します。Webブラウザは、URL(Uniform Resource Locators)を指定すると、特定のWebサーバ上の、特定の資源にアクセスします。また、Webサーバは指定されたリソースをWebブラウザに送信します。通常はHTML(HyperText Markup Language)データを送信するので、Webブラウザはその内容を適切な形式で表示します。



このようにして、Webブラウザを使用して接続可能なサーバで公開されている情報を自由にアクセスすることが可能になります。現在では情報公開だけでなく、業務アプリケーションを実行するための環境としても利用されています。

HTML文書だけをサーバ上に置いた場合、静的な情報を公開することは可能です。しかし、静的な情報だけでは業務に欠かせないサーバ上のデータベースへのアクセスや、動的にHTML文書を生成するなどの処理ができません。

Webの主なインタフェースを以下に示します。

- CGI(Common Gateway Interface)
- ISAPI(Internet Server Application Programming Interface)

業務アプリケーションでは、これらインタフェースを利用してWebサーバ上のアプリケーションを作成する必要があります。このWebサーバ上で実行されるアプリケーションをWebアプリケーションと呼びます。

#### 0.1.2 HTTPの基礎

WebはHTTPの上に成り立っています。WebサーバとWebブラウザは、このHTTPを使用してデータの送受信を行います。WebブラウザからWebサーバへのデータの送信をリクエストと呼び、その逆をレスポンスと呼びます。通常は、リクエストに対してレスポンスが必ず存在し、リクエストで通信路を確保し、レスポンスが完了した時点で解放します。HTTPではリクエストとレスポンスは最初の行以外は同じ形式をしており、非常に簡単な構成になっています。

リクエストデータ

メソッド URL バージョン
HTTPヘッダ
メッセージボディ

#### レスポンスデータ

バージョン ステータスコード 理由
HTTPヘッダ
メッセージボディ

それぞれの意味は以下のとおりです。

#### メソッド

Webサーバへの要求内容を表し、GET、POST、HEADなどがあります。

#### URL

要求するリソースの場所を表します。

#### バージョン

使用するHTTPのバージョンを表します。

#### ステータスコード

3桁の数字でサーバおよびWebアプリケーションの処理状況を表します。

#### 理由

ステータスコードに対する理由を表します。

#### HTTPヘッダ

WebサーバからWebブラウザをコントロールしたり、Webブラウザの情報やメソッドに対する条件付けを行ったりするのに使用します。

#### メッセージボディ

WebサーバとWebブラウザ間で送受信されるデータを表します。

個々の詳細は、HTTPの仕様書または書籍などを参照してください。ここでは、メソッド、ステータスコードおよびHTTPヘッダのうち、Webアプリケーションが参照または設定するものについて簡単に説明します。

主なメソッドは、以下のとおりです。

#### GET

URLで指定した情報を取り出します。URLで指定したものがWebアプリケーションの場合、そのアプリケーションが出力するデータを受信します。

なお、フォームに入力したデータはアプリケーションに渡されますが、データ量には上限があります。

#### HEAD

機能的にはGETと同じですが、HTTPヘッダだけを受信する点が異なります。URLで指定した情報の属性などを調べるときに使用します。

#### POST

フォームに入力したデータをWebアプリケーションへ送信し、そのアプリケーションが出力するデータを受信するために使用します。ファイルのアップロードを行う場合も、このメソッドを指定します。送信できるデータ量に上限はありません。

主なステータスコードは以下のとおりです。

ステータスコード	意味
200	正常終了しました。
302	要求されたドキュメントは一時的に移動しました。移動先は"Location"ヘッダを参照してください。
303	要求されたドキュメントは別のURL に移動されました。移動先は"Location"ヘッダを参照してください。
400	構文エラーです。

ステータスコード	意味
401	認証に失敗しました。
403	指定されたURL に対するアクセスが禁止されています。
404	要求されたURL に対するリソースが存在しません。
410	Web ブラウザ、サーバ間の資源不一致です。
500	内部エラー。Web アプリケーションからの応答がない場合など。
502	Web サーバから不当なリクエストが返されました。
503	高負荷などの理由により、Web サーバへのアクセスができません。

主要なHTTPヘッダは以下のとおりです。

HTTPヘッダ	意味
Accept	Webブラウザが受け取ることのできるMIMEタイプを表します。
Accept-Charset	Webブラウザが受け取ることのできる文字コードセットを表します。
Accept-Encoding	Webブラウザが受け取ることのできるエンコーディング方式。通常はデータ圧縮の形式を表します。
Accept-Language	Webブラウザが受け取ることのできる言語の種類を表します。日本語はja、英語はenとなります。
Content-Encoding	レスポンスデータのエンコーディング方式を示します。
Content-Language	レスポンスデータの言語の種類を示します。
Content-Type	レスポンスデータのMIMEタイプを示します。
Date	リクエストデータおよびレスポンスデータの作成された日付を示します。
Expires	レスポンスデータの有効期限を示します。
Location	情報の正確な位置を示します。
Host	Webサーバが稼動しているサーバのホスト名。
Referer	リクエスト元となったURLを表します。
User-Agent	Webブラウザの情報を表します。

### 0.1.3 HTMLの基礎

HTML (HyperText Markup Language)はハイパーテキストを表現するための言語で、文章の見栄えやリンクなどをタグと呼ばれる特定のキーワードを使用して表現します。タグは“<”と“>”に囲まれた文字列で表されます。また、この文字列をタグ名といいます。ほとんどの場合、タグは開始タグと終了タグを持ち、それぞれ“<タグ名>”、“</タグ名>”となります。この開始タグと終了タグで囲まれた文字列がWebブラウザにとって特別な意味を持ちます。また、<タグ名>だけで意味を持つ単独タグと呼ばれるものもあります。また、タグは属性を持つ場合があります。ほとんどの場合、属性は属性名=属性値の形式で表現されますが、属性名だけのものもあります。属性を持つタグは<タグ名 属性名1=属性値 属性名2>のように表現されます。

ここでは、HTMLを記述するために最低限必要なタグと、Webアプリケーションを呼び出す際に必要なタグについてだけ紹介します。各タグの詳細や他のタグについては、HTMLについて解説している書籍やホームページなどを参照してください。

なお、現在使用されている代表的なWebブラウザはIEですが、独自のタグを定義していますので、すべてのタグがWebブラウザで使用できるとは限りません。また、共通のタグでも動作が異なるものがあります。さらに、Webブラウザのバージョンによってもサポート範囲や動作が異なります。これらについても詳しくは書籍やホームページなどを参照してください。

以下に主なタグの説明とそのタグで使用できる属性を示します。

<HTML>～</HTML>

#### 説明

HTML文書であることを表します。

## <HEAD>～</HEAD>

### 説明

HTML文書のヘッダであることを表します。ヘッダには<TITLE>を記述します。他には<BASE>、<SCRIPT>、<STYLE>、<META>、<LINK>、<OBJECT>、<NEXTID>、<ISINDEX>などのタグが記述できます。

## <TITLE>～</TITLE>

### 説明

HTML文書のタイトルを表します。通常、タイトルはページ上には表示されません。

## <BODY>～</BODY>

### 説明

HTML文章の本文を表します。この中で、さまざまなタグを使用して文章の体裁を整えます。

### 属性

属性名	意味
BGCOLOR= "色"	背景色を指定します。
BACKGROUND= "URL"	背景画像を指定します。
TEXT= "色"	テキストの色を指定します。
LINK= "色"	TEXTは通常テキスト、LINKはリンクテキスト、VLINKはキャッシュ済みリンクテキスト、ALINKはマウスクリックしている間のリンクテキストを表します。
VLINK= "色"	
ALINK= "色"	

## <Hn>～</Hn>

### 説明

章の見出しを指定します。nは章のレベルで、1から6まで指定できます。

### 属性

属性名	意味
ALIGN= "位置"	表示位置を指定します。位置にはleft、center、right が指定できます。

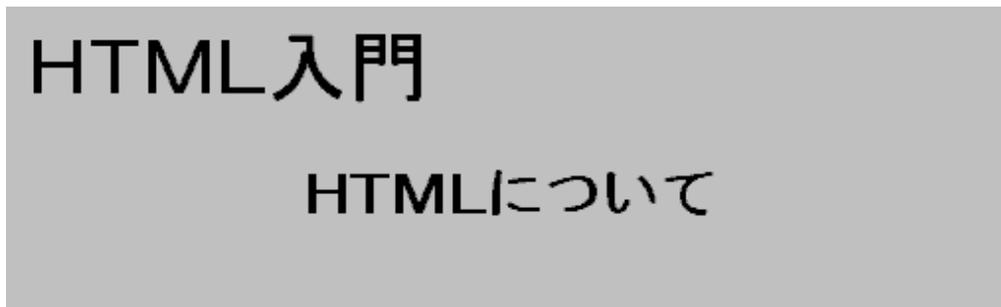


### 例

### 使用例

```
<H1>HTML入門</H1>  
<H2 ALIGN=center>HTMLについて</H2>
```

### 表示結果



## <P>~</P>

### 説明

段落を表します。<P>は省略可能ですが、属性を指定する場合は必須です。

### 属性

属性名	意味
ALIGN= "位置"	表示位置を指定します。位置にはleft、center、rightが指定できます。



### 例

#### 使用例

```
<P> 最初の段落です。
HTML文章では改行文字は意味を持ちません。
<P> 2つ目の段落。
段落を指定すると改行および1行あけて次の文章が記述されます。
<P ALIGN=right> 3つ目の段落。
属性の指定を行う場合は、終了タグを記述して下さい</P>
```

#### 表示結果

最初の段落です。HTML文章では改行文字は意味を持ちません。

2つ目の段落。段落を指定すると改行および1行あけて次の文章が記述されます。

3つ目の段落。属性の指定を行う場合は、終了タグを記述して下さい

## <HR>

### 説明

横線を引きます。

### 属性

属性名	意味
ALIGN= "位置"	表示位置を指定します。位置にはleft、center、rightが指定できます。
SIZE= "高さ"	横線の高さをピクセル単位で指定します。
WIDTH= "幅"	横線の幅をピクセル単位で指定します。また、ブラウザの表示幅に対する割合(%)での指定も可能です。
NOSHADA	横線を立体的な影を持たない線にします。



### 例

#### 使用例

```
最初の横線です。
<HR>
2つ目の横線です。
<HR SIZE=5 WIDTH=50% NOSHADA>
```

3つ目の横線です。  
<HR ALIGN=left SIZE=8 WIDTH=20%>

#### 表示結果

最初の横線です。

2つ目の横線です。

3つ目の横線です。

## <FORM>～</FORM>

### 説明

入力フォームを表します。フォーム中で<INPUT>や<SELECT>タグを使用することで入力部品を配置します。アクションを実行する際に<INPUT>や<SELECT>タグに対する入力データを渡すことができるので、Webアプリケーションの実行に使用されます。

### 属性

属性名	意味
ACTION= "アクション名"	実行 (submit) ボタンを押下したときに実行するアクションを指定します。通常はWeb アプリケーションを指定します。
METHOD= "メソッド名"	フォームに入力した値をWebアプリケーションに渡す方法を指定します。メソッド名がGETの場合は環境変数へ、POSTの場合は標準入力へ渡します。ファイルアップロードの機能を使用する場合は、POSTを指定します。
ENCTYPE= "エンコードタイプ名"	スクリプトにデータを受け渡す際のエンコード方法を指定します。通常は省略して構いませんが、ファイルアップロードの機能を使用する場合は、"multipart/form-data"を指定します。
TARGET= "ターゲット名"	アクションの実行結果を表示するウィンドウ(フレーム)を指定します。
NAME= "フォーム名"	フォームに名前を付けます。スクリプトから使用します。
onSubmit= "スクリプト"	実行ボタンを押下したときに実行されるスクリプトを指定します。
onReset= "スクリプト"	取消しボタンを押下したときに実行されるスクリプトを指定します。



### 例

#### 使用例

```
<FORM METHOD=POST ACTION=アプリ名">  
名前 : <INPUT TYPE=TEXT NAME="FULLNAME1" VALUE="Your name" SIZE=30><BR>  
パスワード : <INPUT TYPE=PASSWORD NAME="PASSWORD1" SIZE=30>  
  
<HR>  
<P>  
趣味 :  
<INPUT TYPE=CHECKBOX NAME="CHECK1" VALUE="音楽鑑賞" CHECKED>音楽鑑賞  
<INPUT TYPE=CHECKBOX NAME="CHECK1" VALUE="読書">読書  
<INPUT TYPE=CHECKBOX NAME="CHECK1" VALUE="スポーツ">スポーツ  
  
<P>  
性別 :  
<INPUT TYPE=radio NAME="RADIO1" VALUE="男性" CHECKED>男性  
<INPUT TYPE=radio NAME="RADIO1" VALUE="女性">女性
```

```
<P>
年齢 :
<SELECT NAME="DRDLIST1">
<OPTION VALUE="10代">10～19歳
<OPTION VALUE="20代" SELECTED> 20～29歳
<OPTION VALUE="30代">30～39歳
<OPTION VALUE="40代以降">40歳～
</SELECT>

<P>
職業 : <BR>
<SELECT NAME="LIST1" SIZE=3>
<OPTION VALUE="フリー">フリーター
<OPTION VALUE="会社員" SELECTED> 会社員
<OPTION VALUE="公務員">公務員
<OPTION VALUE="自営業">自営業
</SELECT>

<P>
コメント : <BR>
<TEXTAREA NAME="TEXTAREA1" COLS=25 ROWS=4></TEXTAREA>
<P>
<INPUT TYPE=submit VALUE="実行">
<INPUT TYPE=reset VALUE="取消">
</FORM>
</BODY>
</HTML>
```

表示結果

名前:

パスワード:

---

趣味:  音楽鑑賞  読書  スポーツ

性別:  男性  女性

年齢:

職業:

コメント:

<INPUT>

説明

入力フォームにおける各種フォーム部品を表示します。表示する部品はTYPE属性によって決まります。

共通属性

属性名	意味
TYPE= "タイプ名"	<p>タイプ名に応じて表示する内容が異なります。</p> <ul style="list-style-type: none"> <li>• TEXT (テキスト)</li> <li>• PASSWORD (パスワード)</li> <li>• CHECKBOX (チェックボックス)</li> <li>• RADIO (ラジオボタン)</li> <li>• HIDDEN (隠し領域)</li> <li>• BUTTON (ボタン)</li> <li>• SUBMIT (サブミットボタン)</li> <li>• RESET (リセットボタン)</li> <li>• FILE (ファイルアップロード)</li> </ul>

属性名	意味
NAME= "名前"	フォーム部品の名前を指定します

#### <INPUT TYPE=TEXT>

##### 説明

テキストを入力するためのフォーム部品です。

##### 属性

属性名	意味
SIZE= "幅"	入力領域の横幅を指定します。
MAXLENGTH= "長さ"	最大入力文字数を指定します。
onChange= "スクリプト"	入力領域が変更されたときに呼び出されるスクリプトを指定します。
onSelect= "スクリプト"	入力領域内の文字列が選択されたときに呼び出されるスクリプトを指定します。
onFocus= "スクリプト"	入力領域にフォーカスが移ったときに呼び出されるスクリプトを指定します。
onBlur= "スクリプト"	入力領域からフォーカスが外れるときに呼び出されるスクリプトを指定します。

#### <INPUT TYPE=PASSWORD>

##### 説明

パスワードを入力するためのフォーム部品です。入力文字が“\*”で表示される以外はTYPE=TEXTと同じです。

#### <INPUT TYPE=CHECKBOX>

##### 説明

複数選択を行うためのフォーム部品です。

##### 属性

属性名	意味
VALUE= "文字列"	この項目がチェックされたときに送信される値を指定します。
CHECKED	この項目の初期状態をチェックされた状態にします。
onClick= "スクリプト"	チェックされたときに呼び出されるスクリプトを指定します。

#### <INPUT TYPE=RADIO>

##### 説明

択一選択を行うためのフォーム部品です。

##### 属性

属性名	意味
VALUE= "文字列"	この項目がチェックされたときに送信される値を指定します。
CHECKED	この項目の初期状態をチェックされた状態にします。
onClick= "スクリプト"	チェックされたときに呼び出されるスクリプトを指定します。

#### <INPUT TYPE=HIDDEN>

##### 説明

隠しフィールドです。ブラウザには表示されません。ブラウザを介してWebアプリケーション間でデータを引き継ぐときなどに使用します。

##### 属性

属性名	意味
VALUE= "文字列"	フィールドの値を指定します。

## <INPUT TYPE=SUBMIT>

### 説明

FORMタグのACTION属性で指定したアクションを実行します。

### 属性

属性名	意味
VALUE= "文字列"	ボタンに表示する文字列を指定します。
onClick= "スクリプト"	チェックされたときに呼び出されるスクリプトを指定します。

## <INPUT TYPE=RESET>

### 説明

FORM内のすべての入力項目が初期値に戻ります。

### 属性

属性名	意味
VALUE= "文字列"	ボタンに表示する文字列を指定します。
onClick= "スクリプト"	チェックされたときに呼び出されるスクリプトを指定します。

## <INPUT TYPE=BUTTON>

### 説明

JavaScriptで使用します。したがって、JavaScript(またはJScript)が使用できないWebブラウザでは表示されません。

### 属性

属性名	意味
VALUE= "文字列"	ボタンに表示する文字列を指定します。
onClick= "スクリプト"	チェックされたときに呼び出されるスクリプトを指定します。

## <INPUT TYPE=FILE>

### 説明

クライアント側からWebサーバ側へアップロードするファイルを指定するためのフォーム部品です。



### 例

### 使用例

```
<FORM METHOD="POST" ACTION="sample/action. script" ENCTYPE="multipart/form-data">  
<P>  
送信ファイル1 : <INPUT TYPE="file" NAME="FILE1"><BR>  
送信ファイル2 : <INPUT TYPE="file" NAME="FILE2"><BR>  
<INPUT TYPE="submit" VALUE="送信">  
<INPUT TYPE="reset" VALUE="取消">  
</FORM>
```

### 表示結果

送信ファイル1:

送信ファイル2:

## <TEXTAREA>～</TEXTAREA>

### 説明

入力フォームにおける複数行入力フィールドを表示します。<TEXTAREA>～</TEXTAREA>までの間のテキストがフィールドに表示されます。

### 属性

属性名	意味
NAME= "文字列"	フィールドに名前を付けます。
ROWS= n	テキストエリアの行数を指定します。
COLS= n	テキストエリアの桁数を指定します。
onChange= "スクリプト"	入力領域が変更されたときに呼び出されるスクリプトを指定します。
onSelect= "スクリプト"	入力領域内の文字列が選択されたときに呼び出されるスクリプトを指定します。
onFocus= "スクリプト"	入力領域にフォーカスが移ったときに呼び出されるスクリプトを指定します。
onBlur= "スクリプト"	入力領域からフォーカスが外れるときに呼び出されるスクリプトを指定します。

## <SELECT>～</SELECT>

### 説明

入力フォームにおける選択フォーム部品を表示します。

### 属性

属性名	意味
NAME= "文字列"	フィールドに名前を付けます。
SIZE= n	選択フィールドの表示行数を指定します。
MULTIPLE	複数選択を可能にします。
onChange= "スクリプト"	入力領域が変更されたときに呼び出されるスクリプトを指定します。
onFocus= "スクリプト"	入力領域にフォーカスが移ったときに呼び出されるスクリプトを指定します。
onBlur= "スクリプト"	入力領域からフォーカスが外れるときに呼び出されるスクリプトを指定します。

## <OPTION>～</OPTION>

### 説明

SELECTタグにおける選択項目を表示します。</OPTION>は省略可能です。

### 属性

属性名	意味
SELECTED	初期状態として選択状態をとります。
VALUE= "文字列"	この項目がチェックされたときに送信される値を指定します。

## 0.1.4 Webブラウザ操作上の注意点と対策

クライアント・サーバシステムでは、クライアント側で利用者が勝手にアプリケーションの制御を変更することはできませんでしたが、Webブラウザを使用したインターネット/イントラネット型のシステムでは、利用者がWebブラウザの状態を容易に変更できるため、操作によってはサーバ上のWebアプリケーション(業務)の整合性が損なわれる恐れがあります。

一般的には、以下に示す操作が問題視されています。

- Webブラウザの「戻る」ボタンを使用する

→ すでに行った処理を再度実行できるため、登録や更新系の処理では、業務に矛盾を引き起こす場合があります。

- ダブルクリックなどの操作によって、SUBMITボタン(INPUTタグのINPUT属性に"SUBMIT"を指定したボタン)を複数回押ししてしまう  
→ 同じ処理が二重に実行されるため、登録や更新系の処理では、業務に矛盾を引き起こす場合があります。
- ブックマークやURLの直接入力によって、業務の途中から開始する  
→ 認証なしで業務を実行できるため、セキュリティ上の問題があります。また、正しい順序で業務を実行しないため、ロジック上の問題が発生する可能性があります。
- キャッシュされたページの参照を参照する  
→ Webブラウザのキャッシュ機能を使用することで、すでに参照されたページをサーバへのアクセスなしに参照できるため、第三者によって売上げデータや個人情報などの機密情報が参照される危険性があります。

上記の問題に対して、Webブラウザおよびサーバ上のWebアプリケーションがとれる対策の例を以下に示します。

利用者の操作	対策例	
	Webブラウザ	Webアプリケーション
Webブラウザの「戻る」ボタンを使用する。	JavaScriptなどを使用して、「戻る」ボタンをなくしたウィンドウを開く。	アプリケーションの実行順序を把握し、正しい順序で実行されていることを確認する。
ダブルクリックなどで、SUBMITボタンを複数回押ししてしまう。	JavaScriptなどを使用して、処理中かどうかを判断し、処理中の場合はSUBMITを無効にする。	二重に実行されても問題のない造りにしておく。
ブックマークやURLの直接入力によって、業務の途中から開始する。	JavaScriptなどを使用して、「ブックマーク」などをなくしたウィンドウを開く。	アプリケーションの実行順序を把握し、正しい順序で実行されていることを確認する。
キャッシュされたページを参照する。	Webブラウザのキャッシュを無効にする。	—

通常、業務の内容に応じて対策の内容も異なります。たとえば、機密度が低い業務では、キャッシュを無効にする必要はありません。また、上記以外にも業務内容によって問題となる操作がある場合は、それぞれの局面に応じた対策をとってください。

## O.2 Q&A集

ここでは、Webサブルーチンの使用時に起こり得る問題点などをQ&A方式でお答えします。

### O.2.1 使えない文字はある？

一般的にインターネット/イントラネットのデータ通信時には、特殊な文字の使用は制限されています。したがって、業務構築の際には以下に示すデータを取り扱わないように注意してください。

#### 外字

外字を使用すると、Webブラウザ(クライアント)によっては正しく表示できないことがあるため、使用しないでください。

#### 半角カナ

1バイトカナ(半角カナ)や機種依存文字(丸付き数字、ローマ数字など)は使用しないでください。

インターネット上では使用する機種によりさまざまなコード系が使用されるため、1バイトカナや機種依存文字を持たない機種もあり、正しく文字を表示することができない場合があります。

#### 識別できない文字

EUCとシフトJISで文字コードが同じ値となる文字が存在します。これらの文字はなるべく使用しないでください。とくに各NAMEまたはVALUEがこれらの文字だけで構成されている場合、正しく識別されません。なお、EUCとシフトJISで重なるコードはE0A1～FCFCです。

### O.2.2 HTML文書のテキスト入力フィールドに、半角と全角文字を混在させて入力してもいい？

半角文字、全角文字が混在しても問題はなりません。ただし、日本語項目ではなく英数字項目として扱ってください。

## 0.2.3 Webブラウザにエラーメッセージが表示されたのはなぜ？

以下の項目について確認してください。

- Webサーバの設定が正しいか確認してください。
- obj.confで指定したWebアプリケーションを起動するためのMIMEタイプとHTML文書のFORMタグ(ACTION)に書いた拡張子に対応するMIMEタイプが等しいか確認してください。
- プログラム名称とHTML文書の格納場所に間違いがないか確認してください。さらに、翻訳オプションが正しいか確認してください。

## 0.2.4 部外者からデータを保護することは必要？

クライアントとサーバ間の通信において、部外者によってデータを参照または変更される可能性があります。このようなシステムの運用に影響する行為を防止するために、必ずSSL(Secure Socket Layer)などの適切なネットワークのセキュリティを施してください。セキュリティ対策については、サーバ管理者とご相談ください。

## 0.2.5 Webアプリケーションでファイルの入出力が正しく行われたい。

ファイルに読み込みや書き込みを行う場合は、Webアプリケーションを実行するユーザIDに対して、ファイルとそのファイルが格納されているフォルダに読み込みや書き込みの権限を与えておく必要があります。

## 0.2.6 Webアプリケーションで表示内容を動的に変える手段は？

1. COBW3\_SET\_CNV\_XXなどを使用する

結果出力用ページの変換部分に//COBOL//変換名//COBOL//と記述しておきます。プログラムで、COBW3\_SET\_CNV\_XXなどを用いて変換名と置き換える文字列(変換文字列)を登録した後、COBW3\_PUT\_HTMLで変換文字列に置き換えることにより、動的にページを変更することができます。

2. COBW3\_PUT\_TEXTを使用する

通常、HTML文書などの出力用ファイルに記述する内容を、COBW3\_PUT\_TEXTに記述できます。したがって、プログラム中で条件ごとに異なるCOBW3\_PUT\_TEXTを用意して、COBW3\_PUT\_TEXTの中でデータ名を指定することにより、動的に出力するページを変更することができます。

3. 1つのページを2つ以上のファイルに分割する

出力用のファイルは2つ以上に分割できます。たとえば、出力するページの前半部分を記述したファイル名を指定してCOBW3\_PUT\_HTMLを呼び出します。ページの後半部分は条件に合わせて記述したファイルをいくつか用意して、プログラム中の条件によって出力するファイル名を変えて、再度COBW3\_PUT\_HTMLを呼び出すようにします。このように、ページの後半を動的に変えることができます。また、ページを部品化することにより、同じようなファイルをいくつも用意する必要がなくなります。

さらに、1～3の方法を組み合わせると、さまざまな場面に対応したページを作成することができます。

## 0.2.7 Webアプリケーションを起動しても正しく動作しない。どうして？

Webアプリケーションの先頭でCOBW3\_INITを指定しているか確認してください。その他、COBW3\_INITを指定している場合は、“[19.8 動作確認](#)”を参照して、エラー情報の収集やデバッグなどを行い、調査してください。

## 0.2.8 Webブラウザに次のメッセージが表示された。どうして？

```
Server Error
The server encountered an internal error or misconfiguration and was unable to complete your request.
```

または

```
指定されたWebアプリケーションは、Webヘッダを一部しか返していません。
```

Content-typeに誤りがあると考えられます。COBW3\_PUT\_HEADで指定したCOBW3-CONTENT-TYPEの値を見直してください。

#### HTML文書を出力する場合

COBW3\_PUT\_HEADを呼び出さない、COBW3-CONTENT-TYPEの値を省略(LOW-VALUE)する、またはCOBW3-CONTENT-TYPE-HTMLを指定してCOBW3\_PUT\_HEADを呼び出す。

#### テキストデータを出力する場合

COBW3-CONTENT-TYPE-TEXTを指定してCOBW3\_PUT\_HEADを呼び出す。

## 0.2.9 WebブラウザにContent-typeの宣言が表示される。どうして？

---

COBW3\_INITでCOBW3-DMODE-DBGが指定されている場合、デバッグ情報としてContent-typeなどのヘッダ情報がWebブラウザに表示されます。

COBW3-DMODEを指定しなければ、表示されません。

## 0.2.10 Webアプリケーションを実行するとWebブラウザに“ファイルをディスクに保存しますか”のメッセージが表示された。どうして？

---

WebアプリケーションでContent-typeの宣言に誤りがある場合、このようなメッセージが出力されることがあります。

Content-typeの宣言を正しく行っているか確認してください。

## 0.2.11 Webアプリケーション作成時にCOBOLプログラムで注意することは？

---

Webアプリケーションでは、COBOLのほとんどの機能をそのまま使用できますが、画面操作に関する次の機能は利用できません。注意してください。

- ・ 表示ファイル(画面操作機能)
- ・ スクリーン機能
- ・ 小入出力機能(ACCEPT文、DISPLAY文)  
(ただし、環境変数、日付および時間の操作機能は利用できます)

Webパラメタの受け取り、参照方法または処理結果の出力方法の詳細は、“[1.8 COBOL Webサブルーチンの使い方](#)”を参照してください。

## 0.2.12 Webサーバを起動させる時に必要な環境変数の設定はどうしたらいい？

---

Windows Server 2012では、[コントロールパネル]>[システム]から[システムの詳細設定]を選択します。表示された[システムのプロパティ]ダイアログボックスから、[詳細設定]タブを選択し、シートの[環境変数]ボタンをクリックします。[環境変数]ダイアログボックスが表示されるため、[システム環境変数]に必要な環境変数を追加し、その後システムをリブートしてください。

または、Webアプリケーション(CGIの場合は.exe、ISAPIの場合は.dll)と同じフォルダに、必要な実行環境情報を記述した実行用初期化ファイル(COBOL85.CBR)を配置してください。

## 0.2.13 COBOLデバッグは使えない？

---

Web環境でも、COBOLデバッグおよびNetCOBOL Studioのアタッチデバッグ機能でデバッグを行うことができます。

以下を参照してください。

- ・ “[19.8.2 対話型デバッグで動作確認する](#)”
- ・ “[19.8.3 NetCOBOL Studioのデバッグ機能で動作確認する](#)”

## 0.2.14 動作確認のためにデバッグを起動しても、クライアントにデバッグの画面が表示されない。どうして？

---

デバッグは、通常、サーバ側の画面に表示されます。よってクライアント側には表示されません。

## O.2.15 WebブラウザにWebサーバからステータスコード(Status-code)が表示された。意味は？

---

“O.1 Web上で動作するCOBOLアプリケーションをはじめて作成する方のために”のHTTPの基礎を参照してください。

## O.2.16 タイムアウト時間を制御することはできる？

---

Webサーバの機能です。Webサーバのマニュアルを参照してください。

## O.2.17 フレームを使ったHTML文書をWebアプリケーションで使うことはできる？

---

フレーム機能は、Webアプリケーションで特別な設定や処理を記述する必要はありません。

フレーム機能をサポートしたWebブラウザであれば、HTMLでフレームの記述をするだけで利用できます。

## O.2.18 COBW3\_CHECK\_VALUE\_Xなどで<INPUT>タグで指定したVALUEの値が正しく検索されないのは？

---

INPUTタグでNAMEを省略した場合、WebブラウザによってはWebパラメタにそのINPUTタグのVALUEの値が入りません。VALUEの値が必要となるINPUTタグには必ずNAMEを指定してください。

## O.2.19 絶対パスで指定したファイルがアクセスできない。どうして？

---

Webアプリケーションの実行時に使用するCOBOLファイルの格納先の指定は、使用するサーバマシンのドライブ構成に従って指定するように注意してください。

## O.2.20 Webブラウザに「文書にデータが含まれません。」のエラーが表示された。どうして？

---

Webアプリケーションから出力する結果出力用ページの<BODY>タグの前では、必ず1行改行してください。



例

.....  
<HTML> ↓  
<HEAD> ↓  
...  
</HEAD> ↓  
↓  
<BODY> ↓  
...  
</BODY> ↓  
</HTML> ↓  
.....

## O.2.21 COBOLデバグが起動されたが、デバグが開始されない。どうして？

---

[デバグを開始する]ダイアログの[デバグ情報]ページで、デバグ情報ファイル格納フォルダが正しく設定されているか確認してください。

## O.2.22 サーバに「F3BIPRCT.DLLが見つかりません。」のエラーが表示される。どうして？

---

COBOLランタイムシステムが、正しくインストールされているか確認してください。Webアプリケーションを動作させる場合は、ユーザ環境変数の指定は有効になりません。

あらかじめ、サーバマシンのシステム環境変数のPATHに、COBOLランタイムシステムのインストールフォルダを設定しておいてください。このとき、システム環境変数の設定を有効にするために、システムを再起動する必要があります。

## O.2.23 ネットワーク環境の資源はアクセスできない？

---

Webアプリケーションの実行時には、ユーザがログインしたときのネットワークドライブの構成は有効になりません。実行時に使用するネットワーク環境の資源は、UNC指定でアクセスしてください。詳細については、“[18.1.3 サービス配下で動作するプログラム](#)”を参照してください。

## O.2.24 サーバから応答がない場合は？

---

サーバ上でウィンドウまたはメッセージボックスの入力待ち状態になっている場合があります。

COBOLプログラムをWebサーバで実行する場合、以下の環境変数情報を実行用の初期化ファイルに記述するか、もしくはシステム環境変数に設定してください。

これら環境変数情報の指定により、ウィンドウまたはメッセージボックス表示が行われなくなるため、オペレータの入力待ちの状態を回避できます。

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。

また、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でください。

- ・ 表示ファイル(画面操作機能)
- ・ スクリーン操作機能
- ・ 小入出力機能(ACCEPT文、DISPLAY文)  
(ただし、環境変数、日付および時間の操作機能は利用できます)

詳細は、“[18.1.3 サービス配下で動作するプログラム](#)”を参照してください。

この他に、ODBCの設定(ODBCを使用している場合に)、不備がある場合もあります。これらについても確認してください。

## O.2.25 COBOLデバグが起動されない。どうして？

---

以下の点を確認してください。

- ・ 環境変数情報@CBR\_ATTACH\_TOOL=TESTが設定されていること
- ・ COBOL Tool Attaching Serviceが開始されていること

## O.2.26 NetCOBOL Studioでデバグが開始されない。どうして？

---

環境変数@CBR\_ATTACH\_TOOLが正しく設定されていることを確認してください。@CBR\_ATTACH\_TOOLの設定方法の詳細については、“[C.2.2 @CBR\\_ATTACH\\_TOOL\(プログラムからデバグ機能を起動する指定\)](#)”を参照してください。

## O.2.27 Cookieデータに漢字を使ってもいい？

---

Cookieデータに漢字を使用することは可能ですが、Cookieデータは他言語のアプリケーションも参照することを考慮してASCII文字を使用することをおすすめします。

## O.2.28 セッション管理機能を使用したWebアプリケーションを二重に起動すると、正しく動作しないことがある。どうして？

---

セッション管理機能を使用したWebアプリケーションは、二重に起動された場合、動作不定になります。これを回避するためには、WebブラウザでJavaScriptなどを利用して二重起動を行わないようにしてください。Webブラウザを操作する上での一般的な注意点については、“[O.1.4 Webブラウザ操作上の注意点と対策](#)”を参照してください。

## O.2.29 クライアント(Webブラウザ)側が何らかの理由で強制終了した場合、サーバ(Webサーバ)側のWebアプリケーションの動きはどうなる？

---

クライアント(Webブラウザ)が強制終了される原因として、以下の場合が考えられます。

- Webブラウザを終了してしまう
- ブラウザでタイムアウトが働いてしまう

通常、HTTPではWebブラウザからのリクエストによってWebサーバ内のWebアプリケーションプロセスとの間にセッションが張られ、これに対するレスポンスの送信が完了すると、そのセッションは直ちに切断されます(一問一答型)。このため、Webサーバで動作するWebアプリケーションは、基本的に一回のHTTPセッション内でファイルのOPEN→READ/WRITE→CLOSEなど、すべての処理を完結するよう構成されていなければなりません。

このようなWebアプリケーションに対して、前述のような強制終了が行われたとしても、WebブラウザとWebサーバ間のセッションが中断するだけで、それ自体はWebアプリケーションには直接影響を与えません。したがって、動作中のWebアプリケーションはそのコードの最後まで実行されます。ただし、HTTPのセッションが切断された後でCOBW3\_PUT\_HTMLやCOBW3\_PUT\_TEXT等によってレスポンスを送信するとエラーが通知されます。

なお、Webサブルーチンのセッション管理機能を使用して、複数のリクエストにまたがる連続した処理を行う場合には、タイムアウト時の異常終了手続きを記述して資源の回収処理を行うよう考慮が必要です。たとえば、初回リクエスト時にファイルのOPENを行い、最終リクエスト時にファイルのCLOSEを行うことを想定した場合、最終リクエストに到達する前にWebブラウザが強制されてしまうと、ファイルが開かれたままとなり、ファイル破壊の危険性が生じます。このため、タイムアウト時の異常終了処理手続きを作成し、その中でファイルをCLOSEするような考慮が必要です。

## O.2.30 バッチファイルからCGIアプリケーションを起動する場合、指定したコマンドをWebブラウザに表示させないためには？

---

バッチファイルの先頭に、@ECHO OFFを指定してください。

## O.2.31 ログファイルと重要度を指定しているのに、ログファイルが出力されない。どうして？

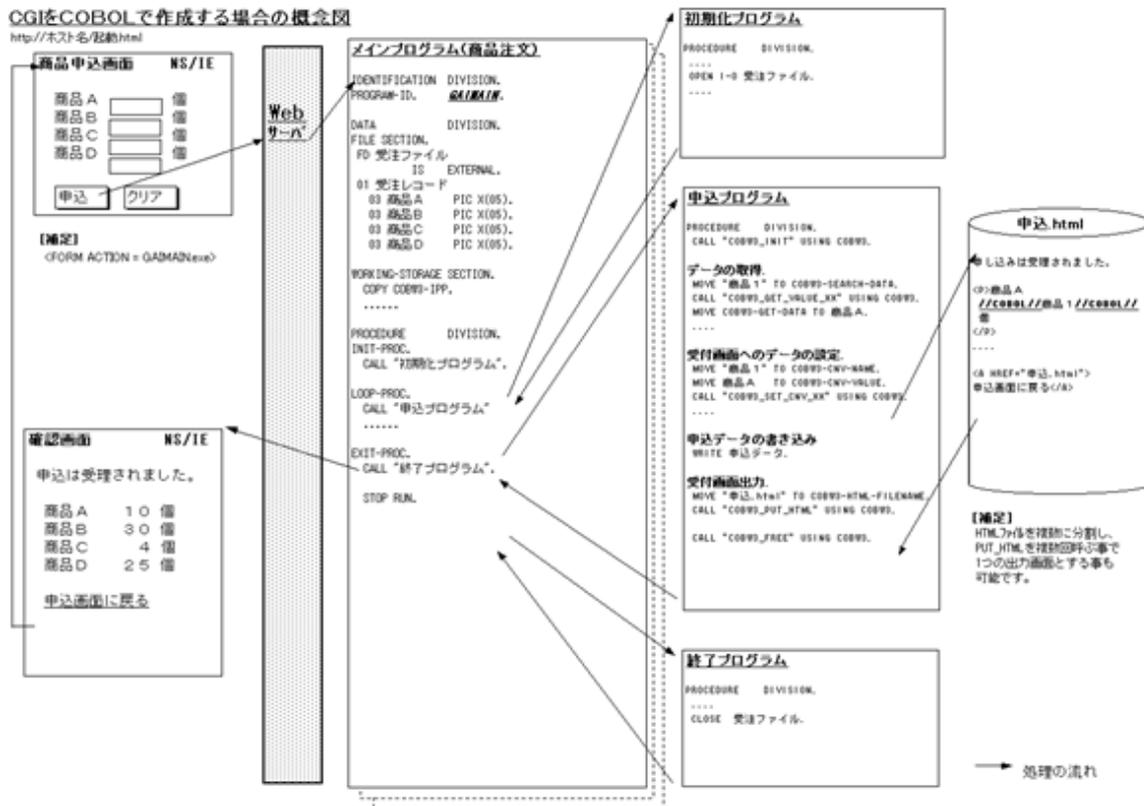
---

以下の項目について確認してください。

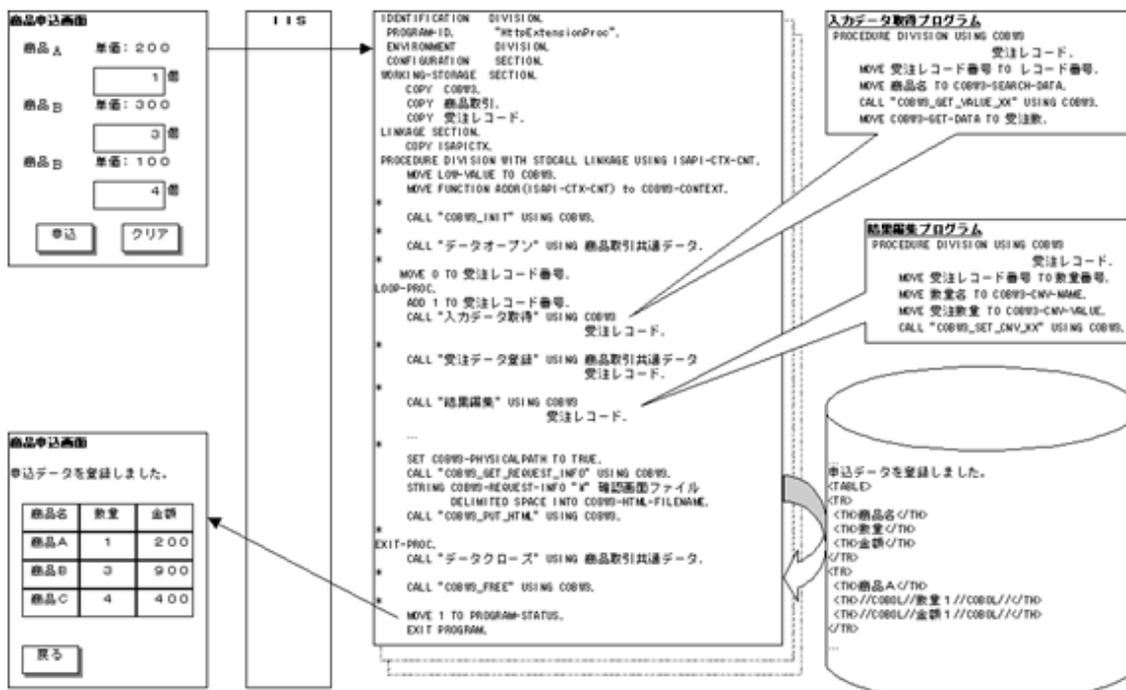
- ファイル名が正しいか
- サーバの実行権限とフォルダのアクセス権限に不整合がないか
- ディスクがいっぱいでないか
- 環境変数名が正しいか

## 0.3 WebアプリケーションをCOBOLで作成する場合の概念図

### 0.3.1 CGIアプリケーションをCOBOLで作成する場合の概念図



### 0.3.2 ISAをCOBOLで作成する場合の概念図



## O.4 CGIからISAPIへの移行の手引き

### O.4.1 CGIからISAPIへの移行

COBOL ISAPIサブルーチン(以後、ISAPIサブルーチンと呼びます)は、COBOL CGIサブルーチン(以後、CGIサブルーチンと呼びます)に対して、基本的にソースの上位互換を保証しています。このため、CGIサブルーチンで作成したWebアプリケーションを、ISAPIサブルーチンを使用したWebアプリケーションへ容易に移行することができます。ただし、ISAPI固有の機能などがあるため、若干のプログラム修正や、翻訳、リンク方法の変更が必要です。

以下に移行の際の注意点などについて記述します。

はじめに、COBOL CGIサブルーチンで作成したWebアプリケーション(以後、CGIアプリと呼びます)とCOBOL ISAPIサブルーチンで作成したWebアプリケーション(以後、ISAPIアプリと呼びます)の違いについて下表にまとめます。

項番	項目	CGIアプリ	ISAPIアプリ
1	アプリケーションの形式	主プログラム(EXE)	副プログラム(DLL)
2	実行単位	プロセス	スレッド
3	モジュール構成	自由	以下をエクスポート関数として用意する <ul style="list-style-type: none"> <li>• GetExtensionVersion</li> <li>• HttpExtensionProc</li> <li>• TerminateExtension</li> </ul> それ以外は自由
4	サーバとのインタフェース領域	不要	必須
5	復帰コード	不要または自由	必須であり、値の意味が決まっている
6	環境変数の操作	任意	制限あり
7	CGI 環境変数操作	提供サブルーチンまたは、 DISPLAY/ACCEPT文で操作可能	提供サブルーチンを使用することで操作可能

CGIからISAPIへ移行するには、上記の相違点を修正する必要があります。

また、以下のサブルーチンについては、互換のためISAPIサブルーチンでも提供しますが、今後、WebアプリケーションがシフトJISコード系で動作する環境でしか使用できません。ISAPIへの移行を機に代替サブルーチンへの修正をおすすめします。

サブルーチン名	代替サブルーチン
COBW3_NAME	COBW3_GET_VALUE_XX COBW3_GET_VALUE_NX
COBW3_NAMEN	COBW3_GET_VALUE_XN COBW3_GET_VALUE_NN
COBW3_VALUE	COBW3_CHECK_VALUE_X COBW3_CHECK_VALUE_N
COBW3_CNV_SET	COBW3_SET_CNV_XX COBW3_SET_CNV_NX COBW3_SET_CNV_XN COBW3_SET_CNV_NN
COBW3_CNV_DEL	COBW3_DEL_CNV_X COBW3_DEL_CNV_N
COBW3_CNV_INIT	COBW3_INIT_CNV

サブルーチン名	代替サブルーチン
COBW3_SET_COOKIE	COBW3_SET_COOKIE_XX COBW3_SET_COOKIE_NX COBW3_SET_COOKIE_XN COBW3_SET_COOKIE_NN
COBW3_DEL_COOKIE	COBW3_DEL_COOKIE_X COBW3_DEL_COOKIE_N
COBW3_GET_COOKIE	COBW3_GET_COOKIE_XX COBW3_GET_COOKIE_NX
COBW3_GET_COOKIEEN	COBW3_GET_COOKIE_XN COBW3_GET_COOKIE_NN

## 0.4.2 アプリケーションの形式

CGIとISAPIはアプリケーションの形式が異なるので、次の項目の変更が必要です。

### 呼出し用ページ

呼出し用ページに記述されているWebアプリケーションの拡張子をEXEからDLLに変更するか、場合によってはアプリケーション名そのものを変更してください。

たとえば、CGIアプリをCGI.EXE、ISAPIアプリをISAPI.DLLとすると、以下のような修正を行う必要があります。

- CGIアプリの呼出し用ページ

```
<FORM ACTION="/sample/CGI.EXE">
...
</FORM>
```

- ISAPIアプリ用に修正後の呼出し用ページ

```
<FORM ACTION="/sample/ISAPI.DLL">
...
</FORM>
```

## 0.4.3 翻訳、リンク方法

### 翻訳

CGIアプリの場合、プログラムを翻訳する時に必要なオプションはありません。強いていえば、主プログラムを翻訳する時に、翻訳オプション“MAIN”を指定するだけです。

ISAPIアプリの場合、DLLで作成する必要があるため、翻訳オプション“MAIN”を指定してはいけません。

また、ISAPIアプリは、次の翻訳オプションが必須となります。

ALPHAL(WORD)またはNOALPHAL

翻訳オプション“ALPHAL(WORD)”または“NOALPHAL”は、“0.4.5 モジュール構成”で説明する3つの入口名を持つプログラムに対して必ず指定してください。これは、入口名の大文字/小文字を区別するためです。

THREAD(MULTI)

ISAPIアプリは、マルチスレッドで動作するアプリケーションなので、すべてのソースプログラムに翻訳オプション“THREAD(MULTI)”を指定してください。

### リンク

CGIアプリは、オブジェクトファイルをリンクしてEXEファイルを作成します。



## 例

```
LINK メインプログラム.obj 初期プログラム.obj 業務プログラム.obj 終了プログラム.obj F3BICIMP.lib MSVCRT.LIB
F3BICWSR.lib /OUT: 実行形式名.exe
```

ISAPIアプリはDLLとして作成しなければなりません。また、CGIアプリとはリンクするインポートライブラリが異なります。



## 例

```
LINK /DLL 初期プログラム.obj 業務プログラム.obj 終了プログラム.obj F3BICBDM.obj F3BISAPI.lib F3BICIMP.lib
KERNEL32.lib MSVCRT.LIB /OUT:実行形式名.dll /DEF:モジュール定義.def /ENTRY:COBDMAN
```

なお、モジュール定義ファイル(モジュール定義.def)には、“[O.4.5 モジュール構成](#)”で述べる3つのエクスポート関数を記述します。具体的には次のようになります。

```
LIBRARY 実行形式名
EXPORTS
    GetExtensionVersion
    HttpExtensionProc
    TerminateExtensio
```

## O.4.4 実行単位

CGIアプリからISAPIアプリへの移行は、プロセスベースのアプリケーションからスレッドベースのアプリケーションへの移行になります。移行時に注意すべき点について説明します。

### 翻訳方法

マルチスレッドオブジェクトを生成するための翻訳オプション“`THREAD(MULTI)`”を指定する必要があります。

### 共有資源へのアクセス

CGIでは、CGIアプリ自身のプロセス内で1つの実行スレッド(シングルスレッド)として動作するため、同一プロセス内の複数のスレッド間で共有される資源は存在しませんでした。

ISAPIでは、IIS自身のプロセス内で複数のスレッド(マルチスレッド)としてISAPIアプリが動作するため、複数のスレッド間で資源を共有できます。

ISAPIアプリで、スレッド間共有資源を使用する場合、共有資源に対する競合状態が発生しないよう同期制御を行う必要があります。共有資源に対する同期制御は、その資源によって、COBOLランタイムシステムが自動的に制御するものと、COBOLが提供する“スレッド同期制御サブルーチン”を使用してISAPIアプリ自身が制御するものがあります。

詳細は、“[第20章 マルチスレッド](#)”の以下の節を参照してください。

- “[20.4 スレッド間の資源の共有](#)”
- “[20.9 スレッド同期制御サブルーチン](#)”

## O.4.5 モジュール構成

CGIアプリのモジュール構成は、主プログラムを必要とする以外、特に制限がありません。しかし、ISAPIアプリは次の3つのモジュールを定義しておく必要があります。

- `GetExtensionVersion`
- `HttpExtensionProc`
- `TerminateExtension`

この中で、CGIアプリの主プログラムに対応するものがHttpExtensionProcです。GetExtensionVersion、TerminateExtensionが対応するものはありません。GetExtensionVersionは初期化処理、TerminateExtensionは終了処理のための関数です。

GetExtensionVersion、HttpExtensionProc、TerminateExtensionの詳細は、“[GetExtensionVersion](#)”、“[HttpExtensionProc](#)”および“[TerminateExtension](#)”を参照してください。また、主プログラムのプログラム名をHttpExtensionProcにする必要があります。たとえば、次のように修正してください。

- CGIアプリの内容

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.     MAINPROG.  
ENVIRONMENT     DIVISION.
```

- ISAPIアプリ用に修正した内容

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.     "HttpExtensionProc".  
ENVIRONMENT     DIVISION.
```

## 0.4.6 サーバとのインタフェース領域

CGIアプリではWebサーバとのインタフェース(環境変数は除く)は特に意識する必要はありませんでした。しかし、ISAPIアプリのHttpExtensionProcは関数としてIISから呼ばれ、IISとのデータのやりとりを行うためのインタフェース(パラメタ)領域を受け取ります。このため、CGIアプリからISAPIアプリに移行する際は、連絡節と手続き部を修正する必要があります。

また、このインタフェース領域をCOBOL ISAPIサブルーチンで使用するため、COBOL ISAPIサブルーチンとのインタフェースであるCOBW3にそのアドレスを設定する必要があります。

たとえば、次のように修正してください。

- CGIアプリの内容

```
WORKING-STORAGE SECTION.  
COPY COBW3.  
PROCEDURE DIVISION.  
MOVE LOW-VALUE TO COBW3.
```

- ISAPIアプリ用に修正した内容

```
WORKING-STORAGE SECTION.  
COPY COBW3.  
LINKAGE SECTION.  
COPY ISAPICTX.  
PROCEDURE DIVISION WITH STDCALL USING ISAPI-CTX-CNT.  
MOVE LOW-VALUE TO COBW3.  
MOVE FUNCTION ADDR(ISAPI-CTX-CNT) TO COBW3-CONTEXT.
```

## 0.4.7 復帰コード

CGIアプリの主プログラムでは、復帰コードの使用は任意です。しかし、“[0.4.6 サーバとのインタフェース領域](#)”でも述べたように、ISAPIアプリのHttpExtensionProcは関数として呼び出され、IISはこの関数からの戻り値を期待しています。このため、ISAPIアプリでは、PROGRAM-STATUSに必ず復帰コードを設定する必要があります。

たとえば、次のように修正してください。

- CGIアプリの内容

```
END-PROC.  
STOP RUN.
```

- ISAPIアプリ用に修正した内容

```
END-PROC.  
MOVE 1 TO PROGRAM-STATUS.  
EXIT PROGRAM.
```

設定できる復帰コードは次の2種類です。通常は正常終了を設定します。また、エラー終了を設定すると、サーバは該当アプリケーションでエラーが起きたと判断し、その旨のメッセージをWebブラウザに送信します。

復帰コード	意味
1	正常終了
4	エラー終了

## O.4.8 環境変数操作

環境変数ブロックは、プロセスごとに1つ割り当てられるため、プロセスとして動作するCGIアプリでは、Webアプリケーションから環境変数を操作してもシステム上、何の問題もありませんでした。しかし、ISAPIアプリはIIS自身のプロセスで複数のスレッドとして動作するため、環境変数の内容を変更するとIISやほかのアプリケーションに影響を及ぼしかねません。また、環境変数の内容を参照する場合も、内容が不変であることは保証されないため、アプリケーションの動作が意図したものと異なる場合があります。したがって、CGIアプリで環境変数操作をしている場合は、環境変数を使用しないように変更する必要があります。

また、アプリケーションが呼び出されたときのカレントフォルダも固定である保証はないので、カレントフォルダを暗に使用する機能は注意が必要です。たとえば、カレントフォルダからの相対パス指定によるファイル操作が該当します。これに代わる手段として、次の方法があります。

### 絶対パスでファイルを指定する

- CGIアプリの内容

```
MOVE "RESPONSE. HTM" TO COBW3-HTML-FILENAME.
CALL "COBW3_PUT_HTML" USING COBW3.
```

- ISAPIアプリ用に修正した内容

```
MOVE "C:¥COBOL¥ISAPI¥RESPONSE. HTM" TO COBW3-HTML-FILENAME.
CALL "COBW3_PUT_HTML" USING COBW3.
```

### 仮想ディレクトリに対応する物理パスを利用してファイル位置を指定する

仮想ディレクトリに対応する物理パスを取得するには、COBW3\_GET\_REQUEST\_INFOを使用します。

- CGIアプリの内容

```
MOVE "RESPONSE. HTM" TO COBW3-HTML-FILENAME.
CALL "COBW3_PUT_HTML" USING COBW3.
```

- ISAPIアプリ用に修正した内容

```
SET COBW3-PHYSICALPATH TO TRUE.
CALL "COBW3_GET_REQUEST_INFO" USING COBW3.
MOVE COBW3-REQUEST-INFO TO COBW3-HTML-FILENAME.
MOVE "¥RESPONSE. HTM" TO
      COBW3-HTML-FILENAME (COBW3-REQUEST-INFO-LENGTH + 1:13).
CALL "COBW3_PUT_HTML" USING COBW3.
```

これに関連した内容として、実行用初期化ファイル(COBOL85.CBR)内のファイルの指定方法があります。CGIアプリでは絶対パス、相対パスの双方が指定可能ですが、ISAPIアプリでは相対パスは使用できません。また、代替手段もないため、絶対パスを指定するように実行用初期化ファイルを修正してください。

- CGIアプリのCOBOL85.CBR

```
[プログラム名]
アクセス名=SEQFILE.DAT
```

- ISAPIアプリ用に修正したCOBOL85.CBR

```
[プログラム名]
アクセス名=C:¥COBOL¥ISAPI¥SEQFILE.DAT
```

## O.4.9 CGI環境変数の操作

CGIアプリでは、COBOLの環境変数操作機能を使用して“CGI環境変数”を参照できますが、ISAPIアプリでは参照できません。このため、以下の3つのサブルーチンを使用します。

### COBW3\_RECEIVE\_HEADER

HTTPヘッダを取得します。

### COBW3\_GET\_REQUEST\_INFO

リクエストに関する各種情報を取得します。

### COBW3\_GET\_AUTHORIZE

認証情報を取得します。

また、CGIアプリでは、COBOL環境変数操作機能を使用してCookieデータを設定/参照できますが、ISAPIアプリでは、COBOL環境変数操作機能によってCookieデータを扱うことができません。このため、以下に示すCookieデータを扱うための専用サブルーチンを使用します。

### COBW3\_SET\_COOKIE\_XXなど

Cookieデータを登録します。

### COBW3\_DEL\_COOKIE\_Xなど

登録されたCookieデータを削除します。

### COBW3\_INIT\_COOKIE

クライアントに送信するCookieデータを初期化します。

### COBW3\_GET\_COOKIE\_XXなど

リクエストに含まれるCookieデータを取得します。

たとえば、HTTPヘッダから情報を取得する処理は次のように修正します。

- CGIアプリの内容

```
DISPLAY "HTTP_USER_AGENT" UPON 環境変数名.  
ACCEPT ブラウザ情報 FROM 環境変数値  
    ON EXCEPTION  
    MOVE "Error" TO ブラウザ情報  
END-ACCEPT.
```

- ISAPIアプリ用に修正した内容

```
MOVE "User-agent" TO COBW3-HEADER-NAME.  
CALL "COBW3_RECEIVE_HEADER" USING COBW3.  
IF COBW3-STATUS = ZERO THEN  
    MOVE COBW3-HEADER-VALUE TO ブラウザ情報  
ELSE  
    MOVE "Error" TO ブラウザ情報  
END-IF.
```

また、Cookieの操作は次のように修正します。

- CGIアプリの内容

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
    COPY COBW3.  
01 COOKIEDATA.  
    02          PIC X(19) VALUE "Sample+cookie+data=".  
    02 COOKIEVALUE PIC X(32).  
PROCEDURE DIVISION.  
    :  
    DISPLAY "HTTP_COOKIE" UPON 環境変数名.
```

```
ACCEPT COOKIEVALUE FROM 環境変数値
ON EXCEPTION
MOVE "Error" TO COOKIEVALUE
END-ACCEPT.
```

- ISAPIアプリ用に修正した内容

```
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY COBW3.
01 COOKIEVALUE PIC X(32).
PROCEDURE DIVISION.
:
MOVE "Sample cookie data" TO COBW3-COOKIE-NAME.
CALL "COBW3_GET_COOKIE" USING COBW3.
IF COBW3-STATUS = ZERO THEN
MOVE COBW3-COOKIE-VALUE TO COOKIEVALUE
ELSE
MOVE "Error" TO COOKIEVALUE
END-IF.
```



## 参考

環境変数操作でCookieデータを扱う場合は、ユーザがCookieデータのURLエンコードおよびURLデコードを行う必要があります。これを回避したい場合、Cookieデータには英数字だけを使用してください。

なお、半角空白をURLエンコードすると“+”になります。

## 0.5 CGIサブルーチン V1.0からの移行時の注意事項について

ここでは、COBOL97 V40L10で提供されているCGIサブルーチン1.0(以降、V1.0と呼びます)を使用したCOBOLアプリケーションをCGIサブルーチン2.0(以降、V2.0と呼びます)以降を使用したCOBOLアプリケーションに移行する方法について説明します。



## 参考

V2.0以降は、各種Webサーバが提供するAPI(ISAPIなど)に対応したWebサブルーチンとほぼ同様のインタフェースで実現されています。このため、V2.0以降を使用して作成したWebアプリケーションは、他のWebサブルーチンへの移植(たとえば、CGIからISAPIなど)が容易で、移植工数やテスト工数を大幅に削減しつつ、各Webサーバが提供するAPIがもたらす運用性能の向上などのメリットを得ることができます。

以上のことから、今後開発されるWebアプリケーション(新規/改造するもの)はV2.0以降を使用してください。また、V1.0で既に作成されたCOBOLアプリケーションもV2.0以降に移行することをおすすめします。

なお、V1.0を使用したCOBOLアプリケーションをそのまま使用する場合には、“[19.4 CGIサブルーチンを使用したアプリケーションの作成と実行](#)”を参照してください。

### 0.5.1 COBOLプログラムおよび関連資産の修正

次の手順に従ってソースプログラムを修正します。

#### 登録集の変更

V1.0のインタフェースである登録集(FJCOB-CGI.cbl)をV2.0以降の登録集(COBW3.cbl)に変更します。

変更前	V1.0	COPY FJCOB-CGI
変更後	V2.0以降	COPY COBW3

## サブルーチン名およびパラメタ名の変更

V2.0以降ではV1.0と同等の機能を提供していますが、各サブルーチンの名前とパラメタの名前に違いがあります。次の規則に従って変更してください。

- パラメタ名の修正(FJCOB-CGI-xxxxをCOBW3-xxxxに変更する)
- サブルーチン名の修正(COBW3\_CGI\_xxxをCOBW3\_xxxに変更する)
- CALL文のUSING指定の修正(FJCOB-CGIをCOBW3に変更する)



### 例

#### COBW3\_CGI\_NAMEの変更例

- 変更前

```
MOVE "NAME1" TO FJCOB-CGI-SEARCH-DATA.  
  
CALL "COBW3_CGI_NAME" USING FJCOB-CGI.  
  
IF FJCOB-CGI-SEARCH-FLAG-EXIST  
    MOVE FJCOB-CGI-GET-DATA TO データ  
  
END-IF.  
.....
```

- 変更後

```
MOVE "NAME1" TO COBW3-SEARCH-DATA.  
  
CALL "COBW3_NAME" USING COBW3.  
  
IF COBW3-SEARCH-FLAG-EXIST  
    MOVE COBW3-GET-DATA TO データ  
  
END-IF.  
.....
```



### 注意

“COBW3\_CGI\_xxx\_CNV”は“COBW3\_CNV\_xxx”に名前が変更されていますので注意してください。

V1.0とV2.0以降の各機能の対応は、下表を参照してください。

CGI サブルーチン 1.0	CGI サブルーチン 2.0以降
COBW3_CGI_INIT	COBW3_INIT
COBW3_CGI_NAME	COBW3_NAME (注)
COBW3_CGI_NAMEN	COBW3_NAMEN (注)
COBW3_CGI_VALUE	COBW3_VALUE_ (注)
COBW3_CGI_PUT_HTML	COBW3_PUT_HTML
COBW3_CGI_PUT_TEXT	COBW3_PUT_TEXT
COBW3_CGI_SET_CNV	COBW3_CNV_SET (注)
COBW3_CGI_SET_CNV2	

CGI サブルーチン 1.0	CGI サブルーチン 2.0以降
COBW3_CGI_DEL_CNV	COBW3_CNV_DEL (注)
COBW3_CGI_CNV_INIT	COBW3_CNV_INIT (注)
COBW3_CGI_SYSTEM	COBW3_SYSTEM
COBW3_CGI_FREE	COBW3_FREE

注：CGIサブルーチン3.0以降では、代替サブルーチンの利用をおすすめします。

## サブルーチンの非互換に関する変更

V2.0以降ではV1.0と同等の機能を提供していますが、一部のサブルーチンに非互換があるため次の対応が必要です。

- COBW3\_CGI\_INIT(呼出しパラメタの初期化)
- COBW3\_CGI\_PUT\_HTML(FJCOB-CGI-HTML-FSの削除および結果出力用ページファイルの変更)
- COBW3\_CGI\_SET\_CNV2(サブルーチンの変更)

## COBW3\_CGI\_INIT

V1.0のインタフェースである“FJCOB-CGI”はVALUE句によってLOW-VALUEで初期化されていましたが、V2.0以降のインタフェースである“COBW3”はVALUE句が指定されていないため、“COBW3\_INIT”を呼び出す前に“COBW3”にLOW-VALUEを転記し、初期化するように変更します。

- 変更前

```
CALL "COBW3_CGI_INIT" USING FJCOB-CGI.
```

- 変更後

```
MOVE LOW-VALUE TO COBW3.  
CALL "COBW3_INIT" USING COBW3.
```

また、V1.0の“COBW3\_CGI\_INIT”には“FJCOB-CGI-CONTENT-TYPE”を指定することにより、Content-typeを宣言する機能がありましたが、V2.0以降ではヘッダを出力する“COBW3\_PUT-HEAD”を使用する仕様に変更されています。このため、“COBW3\_INIT”の実行後に“COBW3\_PUT-HEAD”によってContent-typeを宣言するように修正してください。

- 変更前

```
SET FJCOB-CGI-CONTENT-TYPE-HTML TO TRUE.  
CALL "COBW3_CGI_INIT" USING FJCOB-CGI.
```

- 変更後

```
MOVE LOW-VALUE TO COBW3.  
CALL "COBW3_INIT" USING COBW3.  
SET COBW3-CONTENT-TYPE-HTML TO TRUE.  
CALL "COBW3_PUT_HEAD" USING COBW3.
```



## 参考

COBOL CGIサブルーチン2.0以降ではContent-typeなどのヘッダ出力機能が拡張されていますので、ヘッダ出力のタイミングを“COBW3\_INIT”を呼び出した後にすることが可能です。詳細は、“COBW3\_PUT-HEAD”を参照してください。

## COBW3\_CGI\_PUT\_HTML

V1.0では“COBW3\_CGI\_PUT\_HTML”を実行した場合、“FJCOB-CGI-HTML-FS”に指定した結果出力用ページとして使用するファイルの入出力状態を返却していましたが、V2.0以降の“COBW3\_PUT\_HTML”では入出力状態は返却されません。COBOLプログラムで入出力状態を参照する場合には、“COBW3-STATUS”を参照してください。

“COBW3\_PUT\_HTML”で出力する結果出力用ページに“//FJCOB//変換名//FJCOB//”の変換データが記述されている場合、キーワードが“//FJCOB//”から“//COBOL//”に変更されていますので、修正します。



## 例

### 変換名“NAME1”が記載されている場合の変更例

- 変更前

```
あなたの名前は、//FJCOB//NAME1//FJCOB// です。
```

- 変更後

```
あなたの名前は、//COBOL//NAME1//COBOL// です。
```

## COBW3\_CGI\_SET\_CNV2

変換名を登録するV1.0の“COBW3\_CGI\_SET\_CNV2”を使用している場合には、V2.0以降の“COBW3\_CNV\_SET”に変更します。



## 例

### 変換名“NAME1”を登録している場合の変更例

- 変更前

```
MOVE "NAME1" TO FJCOB-CGI-CNV-NAME.  
MOVE "名前" TO FJCOB-CGI-CNV-VALUE.  
SET FJCOB-CGI-CNV2-MODE-ADDREP TO TRUE.  
CALL "COBW3_CGI_SET_CNV2" USING FJCOB-CGI.
```

- 変更後

```
MOVE "NAME1" TO COBW3-CNV-NAME.  
MOVE "名前" TO COBW3-CNV-VALUE.  
SET COBW3-CNV-MODE-ADDREP TO TRUE.  
CALL "COBW3_CNV_SET" USING COBW3.
```

## 0.5.2 COBOLプログラムの翻訳およびリンク

ソースプログラムの修正後、プログラムの再翻訳を行います。

翻訳後はCOBOL CGIサブルーチン2.0以降をリンクします。



## 例

### a.cobを翻訳/リンクする例

```
cobol32 -M a.cob  
link /out:a.exe a.obj F3BICWSR.LIB F3BICIMP.LIB MSVCRT.LIB
```

# 索引

	[]		
-SRTnnnn.TMP.....		124	/WAITオプション.....
Cookieデータの削除.....		1080	39
	[数字]		@AllFileExclusive.....
1行の形式.....		15	916
1つの行を参照する.....		314	@CBR_ATTACH_TOOL.....
1つのソースファイルを翻訳する.....		725	505,514,525,526,528,529,853,916
1つの目的プログラムをリンクする.....		739	@CBR_CBRFILE.....
2進項目の扱い.....		865	919
88コンソーシアム形式.....		138	@CBR_CBRINFO.....
	[記号]		919
#.....		902	@CBR_CGI_LOGFILE.....
#DEC88TOFJ.....		139	506
#DECFJTO88.....		139	@CBR_CGI_SEVERITY.....
#FILE情報.....		1167	506
#LINE情報.....		1167	@CBR_CIINF.....
\$......		870	375,919
*COB-BINDTABLE.....		473	@CBR_CI_SRVINF.....
*COB-BINDTABLEクラスのオブジェクトメソッド.....		476	380
*COB-BINDTABLEクラスのファクトリメソッド.....		475	@CBR_ClassInfFile.....
*COM.....		801	919
*COM-ARRAYの型指定子.....		843	@CBR_CODE_SET.....
*COM-ARRAYクラス.....		842	920
*COM-EXCEPTIONクラス.....		847	@CBR_COMPOSER_CONSOLE.....
*COMアーライバインドクラス.....		808	279,921
*COMクラス.....		803	@CBR_COMPOSER_MESS.....
*COMレイトバインドクラス.....		807	921
*OLE.....		801	@CBR_COMPOSER_SYSERR.....
*OLE-ARRAY.....		842	279,921
*OLE-EXCEPTION.....		847	@CBR_COMPOSER_SYSOUT.....
->.....		301	279,922
-A.....		1141	922
-Dc.....		1141	@CBR_CONVERT_CHARACTER.....
-dd.....		1142	922
-Dk.....		1142	@CBR_CSV_OVERFLOW_MESSAGE.....
-do.....		1143	923
-dp.....		1143	@CBR_CSV_TYPE.....
-Dr.....		1144	923
-dr.....		1144	@CBR_DISPLAY_CONSOLE_EVENTLOG_SRCNAME.....
-ds.....		1144	924
-Dt.....		1145	@CBR_DISPLAY_CONSOLE_OUTPUT.....
-I.....		1145	925
-i.....		1146	@CBR_DISPLAY_SYSERR_EVENTLOG_LEVEL.....
-M.....		1146	925
-m.....		1146	@CBR_DISPLAY_SYSERR_EVENTLOG_SRCNAME.....
-O.....		1146	926
-P.....		1147	@CBR_DISPLAY_SYSERR_OUTPUT.....
-R.....		1147	926
-v.....		1147	@CBR_DISPLAY_SYSOUT_EVENTLOG_LEVEL.....
-WC.....		860,1147	926
.ENTRY.....		42	@CBR_DISPLAY_SYSOUT_EVENTLOG_SRCNAME.....
.METHOD.....		454	927
/DEBUG.....		614,680	@CBR_DISPLAY_SYSOUT_OUTPUT.....
/STACK.....		62	928
			@CBR_DisplayName_xxxx.....
			163,928
			@CBR_ENTRYFILE.....
			928
			@CBR_EXFH_API.....
			929
			@CBR_EXFH_LOAD.....
			929
			@CBR_FILE_BOM_READ.....
			929
			@CBR_FILE_LFS_ACCESS.....
			930
			@CBR_FILE_SEQUENTIAL_ACCESS.....
			930
			@CBR_FILE_USE_MESSAGE.....
			931
			@CBR_FUNCTION_NATIONAL.....
			931
			@CBR_InstanceBlock.....
			932
			@CBR_ISAPI_LOGFILE.....
			515
			@CBR_ISAPI_SEVERITY.....
			515
			@CBR_JOBDATE.....
			932
			@CBR_JUSTINTIME_DEBUG.....
			932
			@CBR_MEMORY_CHECK.....
			937
			@CBR_MESSAGE.....
			934
			@CBR_MESS_LEVEL_CONSOLE.....
			935
			@CBR_MESS_LEVEL_EVENTLOG.....
			936
			@CBR_OverlayPrintOffset.....
			937
			@CBR_PrinterANK_Size.....
			938
			@CBR_PrintFontTable.....
			939
			@CBR_PrintInfoFile.....
			939
			@CBR_PrintTextPosition.....
			939
			@CBR_SCREEN_POSITION.....
			940
			@CBR_SCR_KEYDEFFILE.....
			940
			@CBR_SSIN_FILE.....
			941
			@CBR_SYMFOWARE_THREAD.....
			941
			@CBR_SYSERR_EXTEND.....
			941

@CBR_TextAlign.....	941	AIM展開レコード域.....	876
@CBR_THREAD_MODE.....	941	ALL.....	866,888
@CBR_THREAD_TIMEOUT.....	942	ALPHABET句.....	894
@CBR_TRACE_FILE.....	598,942	ALPHAL.....	864
@CBR_TRAILING_BLANK_RECORD.....	942	ALTERNATE RECORD KEY句.....	100
@CBR_WEB_OUT_CODE.....	516	ANK空白.....	883
@CBR_DISPLAY_CONSOLE_EVENTLOG_LEVEL.....	924	ANK文字サイズ.....	938
@CBR_SSIN_FILE.....	276	ANSI COBOL規格.....	876
@CnslBufLine.....	943	ANY LENGTH句.....	481
@CnslFont.....	943	API関数で使用する構造体.....	1003
@CnslWinSize.....	943	APIの種類.....	753,763,770
@DefaultFCB_Name.....	155,944	APOST.....	885
@ExitSessionMSG.....	944	ARGUMENT-NUMBER.....	282,283
@GOPT.....	39,945	ARGUMENT-VALUE.....	282,283
@IconDLL.....	945	ASCII.....	894
@IconName.....	946	ASCOMP5.....	864
@MessOutFile.....	505,514,935,946	ASP.....	883
@MGPRM.....	39,946	ASSIGN句.....	107,108,109,111,178,179,180,181
@NoMessage.....	947	AT END指定.....	105
@ODBC_Inf.....	947	AUTO.....	381
@ODBC_Inf.....	346	A領域.....	15
@OPTIONS.....	17,859		
@PrinterFontName.....	149,948	[B]	
@PRN_FormName_xxx.....	159,948	B4.....	159
@ScrnFont.....	949	B5.....	159
@ScrnSize.....	218,949	BASED ON句.....	302
@ShowIcon.....	949	BEFORE指定.....	175
@SQL_CONCURRENCY.....	349	BINARY.....	865,898
@SVD_COMPATIBILITY.....	713,950	BIND.....	162
@SVD_ENV.....	950	BOM.....	73
@SVD_PERFORM_CHECK.....	952	BOTTOM.....	166,941
@WinCloseMsg.....	218,505,515,952	BOUND.....	866
_FINALIZEメソッド.....	404,423	BSAM.....	116,953
_GET_プロパティ名.....	454	BSAM指定.....	1009
_SET_プロパティ名.....	455	BSORT_TMPDIR.....	124,291
_stdcall.....	248	Btrieveファイル.....	77,137,953
_YEN.....	870	BTRV.....	138,953
		BUFFERLEN.....	1065
[A]		BY CONTENT.....	243
A.....	902	BY REFERENCE.....	243
A3.....	159	BY REFERENCE指定とBY CONTENT指定の違い.....	244,245
A4.....	159	BYTE.....	865
A5.....	159	BY VALUE.....	244
ABNORMAL END.....	604	B領域.....	15
ACCEPT文.....	264,282,285		
ACCEPT文のデータの入力先.....	892	[C]	
ACCEPT文の動作.....	881	c.....	58
ACCEPT文のファイル入力拡張機能.....	274	C.....	158
ACCESS MODE IS DYNAMIC.....	1007	CALLBACK.....	248
ACCESS MODE IS RANDOM.....	1007	CALL文.....	235,243,1120
ACCESS MODE IS SEQUENTIAL.....	1007	CALL文の自動生成.....	794
ACCESS MODE句.....	93,99	CANCEL PROGRAM.....	604
ADDRESS.....	468	CBI.....	651
ADDR関数.....	301,302	CBL.....	16,18,651
AFTER指定.....	175	CCVS.....	881
AIM DBMSとの連絡領域.....	876	CFURCOV.....	983
AIMLIB.....	863	CFURCOVS.....	984
		CGI.....	495,498,748

CGIアプリ.....	1218	COBOL Webサブルーチン.....	498
CGIサブルーチン.....	499	COBOLアプリケーション起動中のWindows終了.....	944
CHARACTER TYPE句.....	148,174	COBOLおよびVisual C++でのクラスの対応.....	460
CHECK.....	592,866	COBOLコマンド.....	1139
CHECK機能.....	58,581,586,588,866,1142	COBOLコンソールウィンドウ.....	74
CHECK機能の使用例.....	592	COBOLコンパイラ.....	3
CH情報.....	155	COBOL実行単位ハンドル取得サブルーチン.....	1071
CIM.....	141	COBOL実行単位ハンドル設定サブルーチン.....	1071
CLASS.....	454	COBOL診断機能.....	3
CLASS-ID.....	402	COBOLソースファイルの登録.....	659
CLOSEONLY.....	381	COBOLソースファイル名表示.....	793
CLOSE文.....	216	COBOLソースプログラムの作成.....	15
CLSID.....	857	COBOLソースプログラムを翻訳する.....	513
COB.....	16,18,651	COBOL対話型デバッグ.....	4
COBW3_GET_REQUEST_INFO.....	1089	COBOLの機能.....	2
COB_AIMLIB.....	7	COBOLの言語間の環境.....	228
COBCI_CLOSE.....	1062	COBOLのコンソールウィンドウ.....	266
COBCI_OPEN.....	1063	COBOLの主プログラム.....	228
COBCI_READ.....	1064	COBOLのデータ項目とCのデータ型との対応例.....	251
COBCISRV.LOG.....	382	COBOLファイル.....	123
COBCIU32.EXE.....	383	COBOLファイルアクセスルーチン.....	143
COBCI_WRITE.....	1066	COBOLファイルアクセスルーチンとは.....	1001
COB_COBCOPY.....	7	COBOLファイルアクセスルーチンの制限事項.....	1047
cobfa_close().....	1009	COBOLファイルアクセスルーチンの留意事項.....	1047
cobfa_delcurr().....	1020	COBOLファイルのサイズを拡張する指定.....	930
cobfa_delkey().....	1022	COBOLファイルユーティリティ.....	3,123
cobfa_delrec().....	1023	COBOLファイルユーティリティの起動.....	125
cobfa_errno().....	1034	COBOLファイルユーティリティの機能.....	126
cobfa_indexinfo().....	1033	COBOLファイルユーティリティの終了.....	126
cobfa_open().....	1004	COBOLファイルユーティリティの使い方.....	124
cobfa_openW().....	1009	COBOLプログラムからCOBOLプログラムを呼び出す..	226,235
cobfa_rdkey().....	1010	COBOLプログラムからCプログラムを呼び出す.....	227,243
cobfa_rdnex().....	1013	COBOLプログラム登録先.....	763
cobfa_rdnex().....	1014	COBOLプログラムの構成.....	305
cobfa_reclen().....	1035	COBOLプログラムの実行.....	36
cobfa_reclen().....	1036	COBOLプログラムを呼び出すCプログラムの結合.....	255
cobfa_release().....	1032	COBOL目的プログラム.....	737
cobfa_rewcurr().....	1024	COBOL目的プログラムをリンクする.....	514
cobfa_rewkey().....	1025	COBOL呼出し規約.....	225,226
cobfa_rewrec().....	1027	COBOLランタイムシステム.....	3
cobfa_stat().....	1035	COBOLランタイムシステムによる同期制御.....	547
cobfa_stkey().....	1028	COB_OPTIONS.....	8
cobfa_strec().....	1030	COB_REPIN.....	7
cobfa_wrkey().....	1016	COB_REPORT_EVENT.....	1060
cobfa_wrnext().....	1018	COBSVINI.INI.....	380
cobfa_wrrrec().....	1019	COB_UNLOCK_DATA.....	1061
COB_FREE_MEMORY.....	1056	COB_UNLOCK_OBJECT.....	1062
COB_GET_PROCESSID.....	1057	COBW3_ALTER_SESSION_TIMEOUT.....	1075
COB_GET_THREADID.....	1058	COBW3_CGI_INIT.....	1226
COB_LIBSUFFIX.....	7	COBW3_CGI_SET_CNV2.....	1227
COB_LOCK_DATA.....	1058	COBW3_CHECK_VALUE_N.....	1076
COB_LOCK_OBJECT.....	1059	COBW3_CHECK_VALUE_X.....	1076
COBOL.....	16	COBW3_CNV_DEL.....	1076
COBOL32コマンド.....	1139	COBW3_CNV_INIT.....	1077
COBOL85.CBR.....	41	COBW3_CNV_SET.....	1078
COBOL.CBI.....	18	COBW3_DEL_CNV_N.....	1079
COBOL G移行支援オプション.....	1181	COBW3_DEL_CNV_X.....	1079

COBW3_DEL_COOKIE.....	1080	COB_FILE_COPY.....	999
COBW3_DEL_COOKIE_N.....	1080	COB_FILE_DELETE.....	1001
COBW3_DEL_COOKIE_X.....	1080	COB_FILE_LOAD.....	992
COBW3_DEL_REPEAT_N.....	1081	COB_FILE_MOVE.....	1000
COBW3_DEL_REPEAT_X.....	1081	COB_FILE_REORGANIZE.....	999
COBW3_DEL_UPLOADEDFILE.....	1082	COB_FILE_SORT.....	996
COBW3_END_SESSION.....	1082	COB_FILE_UNLOAD.....	995
COBW3_FREE.....	1083	COB_GETRUNIT.....	1071
COBW3_GEN_UPLOADFILE_N.....	1085	COB_SETRUNIT.....	1071
COBW3_GEN_UPLOADFILE_X.....	1085	COB_登録集名.....	6
COBW3_GET_AUTHORIZE.....	1086	coclass.....	858
COBW3_GET_COOKIE.....	1086	COM.....	582,796
COBW3_GET_COOKIEEN.....	1086	commit.....	62
COBW3_GET_COOKIE_NN.....	1088	COMMIT文.....	306
COBW3_GET_COOKIE_NX.....	1088	COMMON.....	240
COBW3_GET_COOKIE_XN.....	1088	COMn.....	955
COBW3_GET_COOKIE_XX.....	1088	COMPANY表.....	311
COBW3_GET_SESSION_DATA.....	1090	COM機能の例外処理.....	810
COBW3_GET_SESSION_INFO.....	1091	COM機能を利用したアプリケーション.....	796
COBW3_GET_UPLOADFILE_INFO_N.....	1092	COMクライアントからのトランザクション制御.....	828
COBW3_GET_UPLOADFILE_INFO_X.....	1092	COMクライアント機能.....	800
COBW3_GET_VALUE_NN.....	1094	COMクライアントとCOMサーバを通じたデバッグ方法.....	853
COBW3_GET_VALUE_NX.....	1094	COMクライアントの作成.....	800
COBW3_GET_VALUE_XN.....	1094	COMクライアントのデバッグ方法.....	852
COBW3_GET_VALUE_XX.....	1094	COMサーバからのトランザクション制御.....	825
COBW3_INIT.....	1083	COMサーバ機能.....	814
COBW3_INIT_CNV.....	1084	COMサーバ作成機能の使い方.....	855
COBW3_INIT_COOKIE.....	1084	COMサーバ情報ファイル.....	651
COBW3_INIT_REPEAT.....	1084	COMサーバとは.....	815
COBW3_NAME.....	1095	COMサーバのインタフェース.....	857
COBW3_NAMEN.....	1095	COMサーバの削除.....	816
COBW3_PUT_HEAD.....	1096	COMサーバの作成.....	815
COBW3_PUT_HTML.....	1097	COMサーバの実装形態.....	858
COBW3_PUT_TEXT.....	1100	COMサーバのデバッグ方法.....	852
COBW3_RECEIVE_HEADER.....	1101	COMサーバの登録.....	816
COBW3-SESSION-ADAPTERクラス.....	518	COMサーバ名の設定.....	854
COBW3_SET_CNV_NN.....	1102	COMとDCOM.....	858
COBW3_SET_CNV_NX.....	1102	COMのマルチスレッド環境.....	582
COBW3_SET_CNV_XN.....	1102	COM連携機能.....	582
COBW3_SET_CNV_XX.....	1102	COM連携で扱うデータ型の対応.....	840
COBW3_SET_COOKIE.....	1104	COB_ALLOC_MEMORY.....	1052
COBW3_SET_COOKIE_NN.....	1106	CONF.....	867,898
COBW3_SET_COOKIE_NX.....	1106	CONNECT文.....	306,307
COBW3_SET_COOKIE_XN.....	1106	CONSOLE.....	269
COBW3_SET_COOKIE_XX.....	1106	CONST型指定子.....	249
COBW3_SET_REPEAT_NN.....	1109	CONTROL RECORD句.....	194
COBW3_SET_REPEAT_NX.....	1109	CONV-SIZE.....	74
COBW3_SET_REPEAT_XN.....	1109	CONV-STATUS.....	74
COBW3_SET_REPEAT_XX.....	1109	CONVCHAR.....	868
COBW3_SET_SESSION_DATA.....	1111	Cookieが有効となる仮想ディレクトリ.....	1105
COBW3_START_SESSION.....	1112	Cookieが有効となる仮想パス.....	1108
COBW3_SYSTEM.....	1113	Cookieが有効となるドメイン.....	1105,1108
COBW3_VALUE.....	1113	Cookieデータ.....	773
COBW3_CGI_PUT_HTML.....	1226	Cookieデータの削除.....	1080
COB_DEADLOCK_EXIT.....	1053,1070	Cookieデータの登録.....	1104,1106
COB_EXIT_PROCESS.....	1054	Cookieデータの操作.....	1106
COB_FILE_CONVERT.....	989	Cookieデータを削除.....	1080,1223

Cookieデータを取得.....	1223
Cookieデータを初期化.....	1223
Cookieデータをすべて初期化.....	1084
Cookieデータを登録.....	1223
Cookieの有効期限.....	1105,1108
Cookie名の検索.....	1088
COPY.....	869
COPY修飾値.....	902,903
COPY文.....	730,877
COUNT.....	602,869
COUNT機能.....	581,586,601,869,959,1141
COUNT情報.....	602
COUNT情報の出力形式.....	602
COUNT情報の出力ファイル.....	958
cpi.....	153
CREATE.....	869
CREATEメソッド.....	404
CRT STATUS句.....	221
CSI.....	651
CSV形式データ操作時のメッセージ抑止.....	923
CSV形式データとは.....	295
CSV形式データの作成.....	296
CSV形式データの操作.....	295
CSV形式データの分解.....	297
CSV形式のバリエーション.....	299
CTL.....	184
CURRENCY.....	870
CURRENT-DATE関数を利用した西暦の取得.....	981
CURSOR句.....	221
C言語プログラムとの結合.....	242
Cソースプログラムの作成.....	1003
Cソースプログラムの翻訳.....	1003
CプログラムからCOBOLプログラムを呼び出す.....	227,228,248
Cプログラムを呼び出すCOBOLプログラムの結合.....	258
C呼出し規約.....	225,227
C呼出し規約で結合する.....	255,258

[D]

D.....	902
DATE.....	277
DAY.....	277
DAY-OF-WEEK.....	277
DCOM.....	582,796
DCOMとMTS.....	821
Declare句.....	46
DEFAULT.CBI.....	18,731
DEFAULT指定.....	350
DELETE文.....	306,316
DISCONNECT文.....	307,308
DISK.....	85,111
DISPLAY-OF関数およびNATIONAL-OF関数における後置空白の扱い.....	870
DISPLAY文.....	264,282,285
DISPLAY文のデータの出力先.....	893
DISPLAY文のファイル出力拡張機能.....	273
DLL.....	23,25,651,737
DLLエントリオブジェクト.....	45

DLLを作成する.....	739
DLOAD.....	870
DNTB.....	870
DOC-INFO.....	163,928
DocumentName.....	167
DUMPBINコマンド.....	1150
DUMPBINコマンドのオプション.....	1150
DUPCHAR.....	871
DUPLICATES.....	100
DYNAMIC.....	93,99

[E]

EBCDIC指定.....	894
EDIT-COLOR.....	165
EDIT-CURSOR.....	216
EDIT-MODE.....	165
EDIT-OPTION.....	165
EMPLOYEE.....	468
ENCODE.....	871
END-EXEC.....	306
EndMode.....	381
END OF INITIAL PROGRAM.....	604
END OF RUN UNIT.....	604
ENTRY文.....	48,235
ENVIRONMENT-NAME.....	285
ENVIRONMENT-VALUE.....	285
ENVコマンド.....	693
EQUALS.....	291,873
ERROR.....	382
EVALUATE文.....	106
EXCEPTION文.....	106
EXCLUSIVE.....	112
EXE.....	23,651,737
EXEC SQL.....	306
EXECUTE IMMEDIATE文.....	318
EXIT PROGRAM文.....	235,249
EXP.....	23,651,737
EXTENDモード.....	1006
EXTERNAL句.....	237

[F]

F3BICIMP.LIB.....	29
f3bifrm.dll.....	985
f3bifutc.dll.....	985
f3bifutc.h.....	985
F3BIFUTC.LIB.....	985
f3bifuty.dll.....	985
FA_ASCII.....	1008
FA_AUTOLOCK.....	1007
fa_dictinfo構造体.....	1043
FA_DYNACC.....	1007
FA_EQUAL.....	1012,1016,1028,1031
FA_EXCLLOCK.....	1007
FA_EXTEND.....	1006
FA_FIRST.....	1028
FA_FIXLEN.....	1006
FA_GREAT.....	1029,1031
FA_GTEQ.....	1029,1031



ISAPIフィルタ.....	506
ISAを作成する際の注意点.....	512
I制御レコード.....	157
I制御レコードによる文書名.....	928

[J]

JIS1.....	894
JIS2.....	894
JIS7単位ローマ字コード.....	894
JIS8.....	1008
JIS8単位コード.....	894
JIS非漢字.....	882
JMPBGWDH.....	1068
JMPBWINS.....	1069
JMPCINT2.....	228,535,1072
JMPCINT3.....	535,1073
JMPCINT4.....	535,1074
JMPCINTB.....	46
JMPCINTC.....	46
Jアダプタクラスジェネレータ.....	9

[K]

KOL5.....	170,186,958
-----------	-------------

[L]

L.....	158,162
LAND.....	156
LANGVL.....	876,898
[LDINF]セクション.....	382
LDNAME-REC.....	1065
LENG関数.....	301,302
LIB.....	23,651,737,877
LIBEXT.....	877
LINECOUNT.....	878
LINE SEQUENTIAL.....	88
LINESIZE.....	878
LINKAGE.....	866
LINKコマンド.....	29,1148
LIST.....	879
List Works.....	168,202
LNI.....	651
LOCK_cobfa().....	1036
LOCK MODE IS AUTOMATIC.....	1007
LOCK MODE IS EXCLUSIVE.....	1007
LOCK MODE IS MANUAL.....	1007
LOCK MODE句.....	112
Log.....	381
LOG.....	684,716
LogFileName.....	382
LogType.....	381
long int型.....	247,248,250
LP.....	158
LPI.....	153
LPI情報.....	155
LPT1.....	975
LPT2.....	975
LPT3.....	975
LPTn.....	955

LST.....	18
LTR.....	159
LZ.....	158

[M]

MAIN.....	724,879,898
MAK.....	651
MAKE.....	651
MAKEファイル.....	651,675
MAKEファイルとプロジェクトファイルの比較.....	676
MAKEファイルの用途.....	675
MANAGER.....	468
MANUAL.....	381
MAP.....	880,905
MeFt.....	8,164,201,217,963
MeFt/Web.....	8,202,217,494,515
MEMBER.....	467
MERGE文.....	287,293
MESSAGE.....	880,900,1140
MF.....	880
Micro Focus互換モードの指定.....	880
Microsoft暗号化認証.....	1086
MLBOFF.....	865
MLBON.....	865
MODE.....	881,1181
MODE-1.....	148
MODE-2.....	148
MODE-3.....	148
MSGLEN.....	1065
MSGTYPE.....	1065
MTA.....	582
MTS.....	796
MTS環境への登録方法.....	822
MTSの概要.....	821
MTSのトランザクション管理.....	825
MTSのトランザクション処理.....	831

[N]

N.....	162,164
NAME.....	881
NATIONAL関数の変換モード.....	931
NCW.....	882
NetCOBOL Studio.....	4,10
NetCOBOL Studioのデバッグ機能.....	582
NetCOBOLのアイコン表示の抑止.....	949
NetCOBOLの概要.....	2
NetCOBOLの機能.....	2
NEWメソッド.....	404,475
noc.....	58
nocb.....	58
nocl.....	58
nocn.....	58
nocp.....	58
nor.....	58
NSP.....	882,883
NSPCOMP.....	700,883
NUMBER.....	587,700,884

NUMERIC.....	866
NUMERICタイプ.....	136

[O]

OBJ.....	18,23,651,737
OBJECT.....	884
OBS.....	868
OCCURS DEPENDING ON句の目的語検査.....	593
ODBC環境のセットアップ.....	353
ODBC経由によるアクセス.....	304
ODBC情報設定ツール.....	351
ODBC情報設定ツールの使い方.....	351
ODBC情報ファイル.....	351,947
ODBC情報ファイルの作成.....	346
ODBCで扱うデータとの対応.....	361
ODBCドライバ使用時の注意事項.....	365
ODBCドライバの導入.....	353
OFFSET.....	162
OPEN-DATA-FILE.....	470
OPEN EXTEND.....	1006
OPEN INPUT.....	1006
OPEN I-O.....	1006
OPEN OUTPUT.....	1006
OPEN WITH LOCK.....	1007
OPEN文.....	216
OPTIMIZE.....	885
OPTIONAL.....	87,90,96,103
ORDERS表.....	310
ORGANIZATION IS INDEXED.....	1006
ORGANIZATION IS LINE SEQUENTIAL.....	1006
ORGANIZATION IS RELATIVE.....	1006
ORGANIZATION IS SEQUENTIAL.....	1006
ORGANIZATION句.....	85,88,93,99
OSIV系形式の実行時パラメタ.....	946
OSIV系形式の実行時パラメタの指定方法.....	39
OSIV系システム.....	1174
OSIV系システム固有機能.....	1174
OSIV系システムとの機能比較.....	1174
OSIV系システムと本システムの機能比較.....	1174
OSIV系プログラム.....	1174
OUTPUTモード.....	1006
OVD.....	958
OVD_SUFFIX.....	170
OVD_SUFFIX.....	186,958
OverlayPrintOffset.....	167
OVERRIDE句.....	404

[P]

P.....	158,160,902
P1.....	159,160
P2.....	160
PAGE.....	175
PAPERSIZE.....	169
PASCAL.....	248
PATH.....	124,195,201,217,985
pdbファイル.....	25,739
PERFORM文の最適化.....	971
PGMNAME.....	1181

PMD.....	18,651,683,715
PMD.....	18
POINTER.....	302
PORT.....	156
POSIT.....	160
PowerCOBOL.....	8
PowerFORM.....	8
PowerRDBconnector.....	140
PowerSORT.....	291
PowerSORTが使用するメモリ容量.....	59,890
PRINT.....	21,885
PRINTER.....	178,939
PRINTER-n.....	181
PRINTER_n.....	181
PRINTING MODE句.....	148
PRIORITY.....	1067
PRJ.....	651
PRM.....	866
ProgID.....	857
PROGRAM-STATUS.....	239,247,250,1068,1122
PROPERTY句.....	421
PROTOTYPEメソッド.....	416
PRT-AREA.....	162
PRT-FORM.....	158
PRTFORM.....	169
PRTNAME.....	177,955
PRTOUT.....	168
PXD.....	18,651,683,715
PZ.....	158

[Q]

QUOTE.....	885
------------	-----

[R]

r.....	58,598
R.....	158,902
RAISE文の動作.....	448
RAISING指定のEXIT文の動作.....	449
RANDOM.....	93,99
RCI.....	651
RCS.....	886
RDM.....	141,953
RDMファイル.....	953
RDWR.....	382
READ NEXT RECORD.....	1014
READ PREVIOUS RECORD.....	1014
READ WITH LOCK.....	1012,1014,1016
READ WITH NO LOCK.....	1012,1014,1016
RECEIVETYPE-REC.....	1065
RECORD CONTAINS integer CHARACTERS.....	1006
RECORD IS VARYING IN SIZE.....	1006
RECORD KEY句.....	100
RECORD句.....	86
REGISTER-INDICATORメソッド.....	477
REGISTER-LARGE-DATA-INDICATORメソッド.....	480
REGISTER-LARGE-DATAメソッド.....	479
REGISTER-VARYING-INDICATORメソッド.....	478
REGISTER-VARYINGメソッド.....	477

REGISTERメソッド	476
RELATIVE	93
RELATIVE KEY句	94
REP	18,651,887
REPIN	887
reserve	62
RETRIEVE	471
RETURNING指定	239,245,249,398
ROLLBACK文	306
RPW	874
RSV	163,164,888

[S]

s	58
S	160,902
SAL	18,889
SAVE	471
SCS	889
SDS	890
SD機能	301
SD機能の種類	301
SELECTED FUNCTION句	216
SELECT filename	1008
SELECT OPTIONAL filename	1008
SELECT句	84
SELECT句およびASSIGN句の記述例	85
SELECT文	306,314
SENDTYPE-REC	1067
SEPARATE指定	139
SEQUENTIAL	85,93,99
SERVERHD-REC	1064
SERVERNAME-REC	1064
servicesファイル	374
SET CONNECTION文	306,308
SHREXT	890
SIA	874
SIDE	160
SIZE	159,948
SIZE情報	156
SJIS	1008
SMD	18,651,683,715
SMED_SUFFIX	7,21
smsize	59
SMSIZE	890
SMU	18
SORT-STATUS	290,293
SORT文	287,290
SORT文での同一キーデータの処理方法	873
SOURCE	891
SQL	304
SQLCODE	364
SQLERRD	329
SQLGRP	891
SQLMSG	364
SQLODBCS.EXE	351
SQLSTATE	364
SQL終了子	306

SQL先頭子	306
SQLのホスト変数定義の拡張	891
SRF	892
[SRVINP]セクション	380
SSIN	272,892,959
SSL	1212
SSOUT	272,893,959
STA	582
START FIRST RECORD	1028
START KEY IS <	1029,1031
START KEY IS <=	1029,1031
START KEY IS =	1028,1031
START KEY IS >	1029,1031
START KEY IS >=	1029,1031
StartMode	380
START WITH REVERSED	1029
STARTコマンド	39
STATUS-REC	1064
STD	881,882
STD1	893
STDCALL呼出し規約	225,227,261
STDCALL呼出し規約で結合する	256,259
STDH	874
STDI	874
STDM	874
STOCK表	310
STOP RUN文	235
STREAM	168
STREAMENV	168
STRING文	296
SVD	18,680
SWITCH-0	58
SWITCH-7	58
SWITCH-8	59
Symfoware連携でマルチスレッド動作可能にする	941
SYS	882
SYSCOUNT	602,958
SYSDEF	381
SYSERR	270
SYSERR出力情報の拡張	941
SYSINのアクセス名	959
SysMaxMsg	380
SysMaxWait	380
SYSOUTのアクセス名	959
S制御レコード	163

[T]

TAB	894
TAB文字	16
TCP/IPプロトコル	374
TerminateExtension	507,511,1220
TEST	613,680,894
TextAlign	166
fa_keylist構造体	1041
THREAD	895
TIME	277
TOP	166,941

TRACE.....	382,598,895
TRACE機能.....	58,579,586,597,895,1144
TRC.....	598
TRO.....	598
TRUNC.....	896
TYPE1.....	940
TYPE2.....	940
TYPE-G.....	938
TYPE-M.....	938
TYPE-PC.....	938

[U]

UCS-2.....	1008
Unicode.....	515,699,753,1008,1009,1049
Unicodeの印刷.....	172
Unicodeのぎょう順ファイル.....	92
Unicodeの行順ファイルの識別コード.....	929
UNLOCK_cobfa().....	1038
UNSTRING文.....	297
UNUSED-REC.....	1064
UPDATE文.....	306,316
UPON指定.....	264
URL.....	1201
URLエンコード.....	1224
USAGE IS DISPLAY.....	144
USAGE OBJECT REFERENCE句.....	395
USE AFTER ERROR.....	106
USING指定.....	397
USING指定の記述の違い.....	244
UTF-32用定義体変換コマンド.....	1170
UTF-8.....	1008
UWA.....	876

[V]

V1050.....	889
V1100.....	889
V111.....	888
V112.....	888
V122.....	888
V125.....	888
V30.....	888
V40.....	888
V61.....	888
V70.....	888
V81.....	889
V90.....	889
VAR.....	892
Visual Basic.....	261
Visual C++で定義されているクラスを調べる.....	462
Visual C++連携の概要.....	459
Visual C++連携のプログラム手順.....	462
Visual C++連携の方法.....	459
void型.....	248
VSR2.....	889
VSR3.....	889
VT型.....	841

[W]

WAITTIME.....	1065
Webアプリケーション.....	492,1200
Webアプリケーションウィザード.....	747,749,752,759
Webアプリケーション開発のヒント.....	773
Webアプリケーションの入れ替え.....	515
Webアプリケーションの情報を追加.....	759
Webアプリケーションの雛形.....	750
Webアプリケーションを入れ替える.....	527,530
Webアプリケーションを追加.....	758
Webサブルーチン.....	1075
Webサブルーチンの機能.....	1125
Webサブルーチンの種類.....	755
Webサーバ.....	498
Webパラメタ.....	499
WebパラメタからCookieデータの取得.....	1088
Webパラメタから任意の値を検索.....	1113
Webパラメタから任意の名前を検索.....	1095
Webパラメタ取得プログラム.....	754,757,760,763
Webパラメタ取得プログラム生成ツール.....	759,762
Webパラメタ中のNAMEの検索およびVALUEの取得.....	1094
Webパラメタ中のVALUEの検索.....	1076
Webパラメタの獲得.....	1083
Webブラウザ.....	498,516
Web連携.....	488,491,493
Web連携機能の特徴.....	493
WIDTH.....	162
WINAPI.....	248
WINCOBウィンドウ.....	723
WINCOBコマンド.....	4
WINDOW.....	381
Windowsの終了時の動作.....	381
WindowsソケットAPI.....	374
Windowsのコード系.....	883
WINEXECウィンドウ.....	742
WINEXECウィンドウの起動.....	742
WINEXECコマンド.....	4
WINLINKウィンドウ.....	732
WINLINKコマンド.....	4
WINLINK[ライブラリモジュール結合]ウィンドウ.....	735
WINLINK[リンク]ウィンドウ.....	732
WITH LOCK.....	112
WITH NO LIMIT.....	303
WORD.....	865

[X]

XMDLIB.....	193,200,215,875,1146
XREF.....	897,901

[Z]

ZWB.....	897
----------	-----

[あ]

アイコンリソースのDLL名.....	945
アイコンリソースの識別名.....	946
相手論理宛先.....	373
アクション.....	1205
アクセス形態.....	82,93,99

アクセス種別.....	953	印字モード名.....	148
アタッチデバッグ機能.....	528	インスタンスハンドル獲得サブルーチン.....	1069
アップロードされたファイルをサーバ上の生成.....	1085	インタフェースの表示.....	787,793
アップロード済みファイルの削除.....	1082	インタフェースプログラムの仕組み.....	460
アップロードファイル情報の取得.....	1092	インタフェース領域.....	1221
アテンション種別.....	215	インポートライブラリ.....	23,25,442,651,660,737
アテンション情報の値.....	215	インポートライブラリの作成.....	733
アプリケーションエラー.....	626	ウィンドウ情報ファイル.....	217
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.....	614	ウィンドウ情報ファイルの作成.....	217
アプリケーションプログラムの実行.....	663	ウィンドウ情報ファイルの設定情報.....	217
アプリケーションページ.....	682,714	ウィンドウハンドル獲得サブルーチン.....	1068
アポストロフィ.....	886	埋込みSQL文.....	306
誤り処理手続き.....	106	埋込みSQL文のキーワード一覧.....	354
暗黙スレッド.....	698	上書き保存.....	661
暗黙スレッドの切り替え.....	698	英小文字の扱い.....	864
アンロード.....	125	英数字の文字の大小順序.....	893
アーバインド.....	801	永続オブジェクト.....	466
アーバインドのメリットとデメリット.....	802	エクスポートファイル.....	23,651,737
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.....	932	エディタテンプレート.....	773
依存リポジトリ検索.....	779	エディタ連携機能.....	793
依存リポジトリファイル.....	441,777	エラー検出時の処理実行回数.....	58
依存リポジトリファイルの登録手順.....	778	エラージャンプ.....	663
一次入口点.....	235	エラー出力ページ.....	758
位置付けモード.....	1028,1031	エラー処理.....	1132
一連番号領域.....	15	エラー番号.....	1045
移動.....	126	エラー番号の取得.....	1034
イベントログ.....	489	エラーページ出力プログラム.....	758
イベントログ出力サブルーチン.....	1060	エラーログ.....	382
イベントログの指定可能項目.....	489	エンコード種別.....	1008
イベントログを使うプログラム.....	280	エントリ情報.....	36,47,453
インクルードファイル.....	670	エントリ情報の記述形式.....	454
印刷.....	125	エントリ情報のセクション名.....	42
印刷機能.....	584	エントリ情報の設定.....	57
印刷形式.....	158	エントリ情報ファイル.....	928
印刷原点位置.....	162	エントリ情報ファイルの内容.....	49
印刷情報ファイル.....	166,939	応答しなくなったアプリケーションに対する起動.....	615
印刷処理.....	144	応答なし.....	631
印刷装置の指定方法.....	176	同じ親クラスを持つクラスを同じファイルに保存する.....	469
印刷ファイル.....	81	オブジェクトインスタンス.....	539,548
印刷ファイルで使用するフォント.....	948	オブジェクトインスタンスの獲得方法.....	932
印刷ファイルの定義.....	174	オブジェクトインスタンスの格納数.....	459
印刷不可能な領域.....	154	オブジェクトインスタンスの寿命.....	399
印刷方法の種類.....	144	オブジェクトインスタンスの操作.....	394
印刷方法の特徴・利点・用途.....	146	オブジェクトコンテキストオブジェクト.....	825
印刷面.....	160	オブジェクト削除インタフェースプログラム.....	463
印刷面位置付け.....	160	オブジェクト参照項目.....	395,407
印字禁止域.....	162	オブジェクト指向と従来機能の組合せ.....	483
印字文字.....	148	オブジェクト指向に対応したデバッグ機能.....	697
印字文字の大きさ.....	148	オブジェクト指向プログラミング機能.....	388,447,583
印字文字の大きさ/形態と文字間隔の関係.....	152	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.....	341
印字文字の間隔.....	151	オブジェクト指向プログラミングで使用する資源.....	432,775
印字文字の形態.....	151	オブジェクト指向プログラミングで使用するファイル.....	433,776
印字文字の書体.....	149	オブジェクト指向プログラミングの開発.....	432,775
印字文字のスタイル.....	150	オブジェクト指定子.....	420,813
印字文字の配置座標.....	153	オブジェクト生成インタフェースプログラム.....	463
印字文字の方向.....	151	オブジェクト定義.....	391



行順ファイルの定義.....	88	継承関係の表示.....	787
行順ファイルのレコードの定義.....	88	継承の概念.....	401
強制クローズ.....	115	継承の実現.....	409
行単位のデータを印刷する方法.....	174	継承の定義方法.....	402
共通式の除去.....	970	継承の表示.....	792
共通プログラム.....	240	桁落とし処理.....	896
行内呼出し.....	419	結果出力用ページ.....	499,503,512,518,747,754
行内呼出しの入れ子.....	425	結果出力用ページ内に指定された変換データを登録.....	1078
行番号.....	15,587,902	結果出力用ページ内の繰返し変換データの削除.....	1081
行番号内動詞追番.....	588	結果出力用ページ内の繰返し変換データの初期化.....	1084
業務プログラム.....	500	結果出力用ページ内の繰返し変換データの登録.....	1109
共用モード.....	113	結果出力用ページ内の変換データの削除.....	1079
行レコード.....	184	結果出力用ページ内の変換データの初期化.....	1084
キー定義ファイル.....	219	結果出力用ページ内の変換データの登録.....	1102
キー定義ファイルの記述形式.....	219	結果出力用ページの出力.....	1097
キー入力 of の定義.....	219	結果出力用ページ雛形.....	758
キーパートフラグ使用指定.....	1008	結果出力用ページファイル名.....	770
クォーテーションマーク.....	886	結果ページ出力プログラム.....	747,754,757,767,771
組込み関数一覧.....	977	結果ページ出力プログラム生成ツール.....	766,769
クライアント.....	703	言語要素に対しての指摘メッセージ.....	874
クライアント操作ウィンドウ.....	383	現在位置表示機能.....	692
[クライアント定義]ダイアログ.....	383	現在の日付および時刻の入力.....	276
クライアントのIPアドレス.....	1086	原文名定数.....	974
クライアントマシンへのインストール.....	824	広域最適化.....	885,970,1146
クラス.....	462,518	公開資源.....	447,783
クラスID.....	857	公開パス.....	783
クラスごとに保存ファイルを分ける.....	468	公開フォルダ.....	447
クラス参照の表示.....	792	更新.....	87,96,103,115
クラス情報.....	459	更新項目.....	222
クラス情報データベース.....	785	高度なデータ操作.....	322
クラス情報データベースの生成.....	785	高度なデータ操作を可能とするホスト変数.....	322
クラス情報データベースファイル名の設定.....	786	子クラス.....	402
クラス情報ファイル.....	919	ゴシックDP.....	149
クラス定義.....	388	ゴシック体.....	149
クラス定義で使用できない機能.....	483	ゴシック体半角.....	149
クラスとファイルの対応.....	468	固定形式.....	16,892
クラスのエン트리情報.....	454	固定形式ページ.....	194
クラスの検索条件の入力.....	788	固定長形式.....	1006
クラスの公開.....	446,782	固定長レコード形式.....	82,86
クラスの動的プログラム構造.....	450	固定パーティション.....	188
クラスブラウザ.....	5,784	コネクション.....	306
クラスブラウザウィンドウ.....	786	コネクション操作.....	307
クラスブラウザの情報表示.....	786	コネクションの接続.....	306
クラスブラウザの表示.....	786	コネクションの切断.....	307
クラスブラウザへの入力.....	409	コネクションの変更.....	306
クラス名段落.....	402	コネクション有効範囲の指定値に対する動作説明.....	350
クラスを動的プログラム構造にする.....	452	コネクション有効範囲の定義内容.....	350
グラフィックデバイスインタフェース.....	186	コネクションを接続する.....	307
繰返し範囲の指定方法.....	1099	コネクションを切断する.....	308
繰返し変換データ.....	770	コネクションを変更する.....	308
繰返し変換データの最大設定数.....	770	コマンド行引数の操作機能.....	556
繰返し変換データの削除.....	1081	コマンド行引数の取出し.....	282
繰返し変換データの登録.....	1109	コマンドファイル.....	651,677,692
繰返し変換データをすべて初期化.....	1084	コマンドファイルの作成方法.....	693
グリッド表示可能なツール.....	152	コマンドファイル名.....	684,716
クロスサイトスクリプティング.....	1103,1110	コマンドプロンプト.....	29,267,1139
継承.....	400	コミット.....	831

コンソールウィンドウ.....	893
コンソールウィンドウの大きさ.....	943
コンソールウィンドウの種別.....	922
コンソールウィンドウの属性の変更.....	266
コンソールウィンドウのバッファ数.....	943
コンソールウィンドウのフォント.....	943
コンソールウィンドウを使ったデータの入出力.....	266
コンテキスト.....	825
コンパイラが使用するコード変換ライブラリ.....	868
コード系.....	753,763,771
コード変換ライブラリの指定.....	922

[さ]

在庫表.....	310
最左端ビットの扱い.....	865
最終ターゲットファイル.....	658
最終ターゲットファイルの登録.....	658
再デバッグ.....	693
再編成.....	125,134
作業環境の設定.....	1083
作業手順.....	197,212
索引ファイル.....	81,1006,1047,1049
索引ファイル簡易復旧関数.....	984
索引ファイル操作クラス.....	470
索引ファイルとオブジェクトの対応.....	468
索引ファイルの処理.....	101
索引ファイルの操作.....	132
索引ファイルの使い方.....	98
索引ファイルの定義.....	98
索引ファイルの復旧.....	143
索引ファイルのレコードの定義.....	100
索引ファイル復旧関数.....	983
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.....	987
削除.....	97,104,115,126,400
サスペンド/レジューム.....	699
サニタイジング.....	1103,1110
サブスキーマ定義ファイル.....	18,660,682,715
サブスキーマ定義ファイルのフォルダ.....	863,1141
サブプログラムを呼び出す.....	225
サブルーチン.....	1050
サブルーチン一覧.....	1050
参照.....	87,90,96,103,115
サンプルデータベース.....	341
サンプルプログラムの実行.....	35,744
サンプルプログラムの翻訳.....	18,726
サンプルプログラムのリンク.....	741
サーバ.....	703
サーバ起動時の画面表示.....	380
サーバ識別子.....	1064
サーバ終了.....	379
サーバ情報の定義内容.....	347
サーバ操作ウィンドウ.....	375
サーバ・タイプのアプリケーション.....	486
サーバ定義ファイルの記述内容.....	380
サーバディレクトリ.....	717
サーバ名.....	1064
サーバ連携情報ページ.....	716

サービス配下での注意事項.....	40
サービス配下の注意事項.....	173
識別できない文字.....	1211
シグナル.....	696
資源の解放.....	1083
システム環境変数.....	505,514
システムコマンドの実行.....	1113
システム最大格納メッセージ数.....	380
システム使用領域.....	163,164
システムのコンソールウィンドウ.....	267
定量制限.....	964
システムプログラムを記述するための機能.....	301
システム待ち命令数.....	380
実行.....	698
実行開始位置変更機能.....	690
実行可能ファイル.....	23,25,651,737
実行可能プログラムのバージョン設定機能.....	671
実行環境.....	228,535
実行環境情報.....	36
実行環境情報の種類.....	36
実行環境情報の設定.....	36,346,576
実行環境情報の設定方法.....	36
実行環境設定ツール.....	3,54
実行環境設定ツールの終了.....	57
実行環境の開設と閉鎖.....	229
実行環境の構築.....	345
実行環境の開鎖サブルーチン.....	1074
実行環境変数情報.....	913
実行環境変数情報の優先順位.....	913
実行監視条件での中断.....	678
実行経路をたどる.....	691
実行結果.....	770
実行時オプション.....	57,945
実行時コード系.....	886
実行に必要なとなるエントリ情報.....	455
実行時の適合チェック.....	408
実行時メッセージ.....	628
実行時メッセージおよびSYSERRの出力抑止.....	947
実行時メッセージの重大度.....	936
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.....	489
実行時メッセージの重大度指定.....	935
実行時メッセージの出力先.....	934
実行時メッセージをイベントログに出力する機能.....	489
実行性能の向上.....	458
実行単位.....	228,535
実行単位の開始サブルーチン.....	1072
実行単位の資源.....	573
実行単位の終了サブルーチン.....	1073
実行単位の資源を引き継ぐ方法.....	571
実行中断.....	693
実行の開始と終了.....	743
実行の手順.....	36
実行ファイル名.....	753
実行用初期化ファイル.....	505
実行用の初期化ファイル.....	41,107,110,919
実行用の初期化ファイルのオープン.....	55
実行用の初期化ファイルのクローズ.....	56

実行用の初期化ファイルへの保存.....	57	診断機能.....	610,1145
自動デバッグ.....	693	診断機能が使用する資源.....	611
自動ロック.....	1007	診断機能の概要.....	610
シフトJIS.....	753	診断機能の起動.....	614
ジャストインタイムデバッグ機能.....	681,707	診断メッセージ.....	21
自由形式.....	17,892	診断メッセージのレベル.....	873
集団項目名.....	765,771	診断メッセージリスト.....	900
終了.....	662	診断レポート.....	617
終了キー.....	219	診断レポートの出力先.....	617
終了条件なしのPERFORM文.....	301	診断レポートの出力情報.....	618
終了条件なしのPERFORM文の使い方.....	303	診断レポートの見方.....	626
終了処理メソッド.....	423	数字のデータ例外検査.....	593
終了プログラム.....	502	スクリーンウィンドウ.....	74,218
主供給口1.....	159,160	スクリーン画面の表示位置.....	940
主供給口2.....	160	スクリーン操作機能.....	210,218
主キー.....	82,100	スクリーン操作機能の使い方.....	218
縮小印刷のポートレートモード.....	158	スクリーン操作で使用するフォント.....	949
縮小印刷のランドスケープモード.....	158	スクリーン操作のキー定義ファイル.....	940
出力項目.....	222	スクリーン操作の論理画面の大きさ.....	949
出力コード系.....	516	スクリーン操作用のウィンドウの属性の変更.....	218
出力データ項目名.....	771	少し進んだ使い方.....	556
出力データ登録集.....	757	スタックオーバフロー.....	59
出力データ用登録集.....	754	スタックサイズの求め方.....	59
出力ファイルのコード系の指定.....	920	スタックのオーバフローを回避する.....	62
出力メッセージ.....	589	ステータスコード.....	1097,1201
手動ロック.....	1007	ステートメント番号.....	587
主プログラム.....	25,879,1146	ストアドプロシージャ.....	340
主プログラムが他言語の場合の実行用の初期化ファイル名の指 定方法.....	46	ストアドプロシージャの呼出し.....	306,340
主プログラムの指定.....	659,724	図表レコード.....	187
主レコードキー.....	82,100	スペーシングチャート.....	152
準備するもの.....	1003	スレッド.....	532
順呼出し.....	1007	スレッドID取得サブルーチン.....	1058
順呼出し法.....	93,99	スレッド一覧ウィンドウ.....	689
障害発生箇所の特定制法.....	638	スレッド間共有外部データ.....	575
使用可能なサブルーチン.....	517	スレッド間共有外部ファイル.....	575
条件を指定して参照する.....	313	スレッド間共有資源.....	1220
小数点位置固定入力.....	219	スレッド間の資源の共有.....	543
使用するクラスの選定.....	434	スレッド指定.....	1117
小入出力機能.....	264	スレッド単位でのファイルオープン.....	275
小入出力機能の出力ファイル.....	959	スレッド単位に入力ファイルを開く.....	941
小入出力機能の入出力先.....	264	スレッド同期制御サブルーチン.....	582
小入出力機能の入力ファイル.....	959	スレッド同期制御サブルーチンのエラーコード.....	1119
情報表示.....	377	スレッド同期制御サブルーチンの待ち時間.....	942
使用メモリの節約.....	457	スレッドの同期制御.....	532
初期化処理メソッド.....	423	スレッドモード.....	542,941
初期化ファイル.....	18	制御の復帰とプログラムの終了.....	235
初期化プログラム.....	241	制御レコード.....	183,184
初期プログラム.....	500	正書法.....	16
除数のゼロ検査.....	593	正書法の種類.....	892
書体番号.....	149	生成COBOLプログラム登録先.....	770
処理種別.....	1065	生成するCSV形式のバリエーションの指定.....	923
シリアルポート名.....	178,180	生成ファイル.....	754
字類条件.....	74	生成ファイル詳細.....	754
自論理宛先.....	372	静的構造.....	26,436
シングルスレッド.....	1220	整列.....	125,287
シングルスレッドモード.....	542,1114	整列併合機能.....	287
		整列併合用ファイル.....	289,293



デバッグの環境変数.....	680	データ変更での中断.....	678
デバッグの起動.....	680	データベース.....	304
デバッグの機能.....	689	データベース生成.....	786
デバッグのコマンド一覧.....	702	データマッピングリスト.....	880,905
デバッグの状態を知る.....	692	データ名.....	85,108,180
デバッグの操作.....	685	データロック獲得サブルーチン.....	1058,1061
デバッグを起動する.....	526	データロックサブルーチン.....	1119
デバッグを終了する.....	527	データを表示形式で参照.....	531
デバッグ環境の変更.....	694	データを保護.....	1212
デバッグ機能のマルチスレッドでの基本的な動作.....	698	同一表を関連付けて参照する.....	315
デバッグ機能のマルチスレッド特有な動作.....	698	動作環境.....	211
デバッグ作業の流れ.....	636	動作コード系.....	1008
デバッグ事象を通知する/通知しないの切り替え.....	699	動的SQL.....	317
デバッグしたいプログラムからのデバッグ起動方法.....	681,706	動的カーソル宣言.....	317
デバッグ情報ファイル.....	18,680	動的構造.....	26,436
デバッグ情報ファイルの格納先.....	614	動的パラメタ.....	319
デバッグ情報ファイルのフォルダ.....	1142	動的プログラム構造.....	27,231,450,584,696
デバッグ情報ページ.....	683,716	動的プログラム構造で結合する.....	256,257,259,260
デバッグ操作の再現.....	678	動的プログラム構造による呼出し.....	1119
デバッグ操作の自動化.....	692	動的プログラム構造の作成から実行.....	452
デバッグ対象プログラム.....	679	動的プログラム構造の実行可能プログラムを作成する.....	741
デバッグに必要な資源.....	681	動的プログラム構造の特徴.....	450
デバッグの準備.....	679	動的呼出し.....	1007
デバッグモジュールの作成方法.....	662	動的呼出し法.....	93,99
デバッグモード.....	530	動的リンク構造.....	26,696
デバッグを行う.....	527,529	動的リンク構造で結合する.....	255,257,259,260
デバッグを開始する.....	529	動的リンク構造の実行可能プログラムを作成する.....	740
[デバッグを開始する]ダイアログ.....	681,714	登録した変換データをすべて初期化.....	1077
デバッグを終了する.....	530	登録集.....	988,1131
デフォルトFCB名.....	944	登録集機能.....	877
デフォルトコネクション情報の定義内容.....	350	登録集原文.....	16
電子帳票.....	202	登録集原文の作成.....	16
電子帳票出力.....	168	登録集原文の表示.....	869
電子帳票出力機能.....	202,955	登録集ファイル.....	18,651
テンプレート展開機能.....	787,793	登録集ファイルの拡張子.....	877
データエリアに関するリスト.....	905	登録集ファイルの検索方法.....	20
データ型の対応.....	251,261	登録集ファイルの登録.....	660
データ監視機能.....	690	登録集ファイルのフォルダ.....	877,1145
データ監視/条件監視.....	698	登録集ファイル名.....	765,771
データ項目のエンコードの指定.....	871	登録集名の指定があるCOPY文.....	20
データ出力用登録集.....	771	登録集名の指定がないCOPY文.....	20
データ操作.....	306,309	登録集名の設定.....	730
データソース.....	346	[登録集名の設定]ダイアログ.....	730
データソースとの接続の確認.....	354	登録集名の追加.....	731
データソースの作成.....	353	[登録集名の追加]ダイアログ.....	731
データ入力方法の定義.....	219	登録集名の取消し.....	731
データ入力用登録集.....	764	登録集名の変更.....	731
データのアクセス.....	402	登録済みの全Cookieデータの初期化.....	1084
データの共用.....	237	特殊クラス.....	473
データの更新.....	316	特殊な定数および利用者語の書き方.....	974
データの削除.....	316	特殊レジスタ.....	165
データの参照.....	311	特殊レジスタの使い方.....	216
データの出力(改行コードを付加).....	1100	特徴.....	532
データの挿入.....	316	特定実行条件での中断.....	678
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.....	689	とじしろ幅.....	162
データのファイル入出力.....	695	とじしろ方向.....	162
データの変更箇所を調べる.....	690	ドライブ名.....	974,975

トランザクション.....	306
トランザクションコンテキストオブジェクト.....	828
トランザクション処理.....	306
トレース採取機能.....	691
トレース情報.....	525,598
トレース情報の格納先.....	598
トレース情報の個数.....	58,598
トレース情報の出力形式.....	599
トレース情報の出力ファイル.....	942
トレースログ.....	382

[な]

内部プログラム.....	240
内部プログラム呼出しのパラメタの検査.....	595
名前(NAME)の検索.....	1094
名前(NAME)を検索.....	1095
名前付きパラメタ.....	809
名前の有効範囲.....	241
名前を付けて保存.....	661
二次入口点.....	235
二次入口点一覧の表示.....	792
二次入口点名の指定形式.....	48
日本語空白.....	883
日本語空白の比較方法.....	883
日本語項目における空白の扱い.....	882
日本語利用者語の文字集合.....	882
入出力エラーが発生したときの実行結果.....	107
入出力エラーの実行時メッセージの出力.....	931
入出力状態.....	1047
入出力状態一覧.....	961
入出力状態の取得.....	1035
入出力状態のチェック.....	105
入出力処理の種類.....	199,214
入出力エラー処理.....	105
入力項目.....	222
入力項目の最大文字列長.....	763
入力項目の字類.....	763
入力コード系.....	516
入力データ項目名.....	765
入力データ登録集.....	757
入力データ用登録集.....	754
入力フォーム.....	1205
任意の供給口.....	160
任意の日付の入力.....	277
任意の日付を取得.....	932
認証情報.....	773,1223
認証情報の取得.....	1086
認証タイプ.....	1086
ネットワーク環境の資源.....	1215

[は]

排他ロック.....	1007
バイト単位の領域長.....	865
廃要素.....	868
バインド方法選択の基準.....	803
パス名.....	974,975
パスワード.....	1086
バックグラウンド処理.....	486

バッチデバッグ.....	678,693
バッチデバッグページ.....	683,716
バッチデバッグを行う.....	683,716
バッチファイルを使用する.....	39
パラメタ取得プログラム.....	747
パラメタの受渡し方法.....	235,243,249
パラメタの指定.....	397
パラメタの省略.....	809
パラメタを指定して実行.....	663
バリエーション型データ.....	846
半角カナ.....	1211
半角と全角文字が混在.....	1211
汎用ログ.....	278
汎用ログに出力.....	921
パーティション形式.....	187
ビッグエンディアン変換サブルーチン.....	1070
被デバッグプログラム.....	679
表意定数QUOTEの扱い.....	885
表示.....	125
標識領域.....	15
表指定ホスト変数.....	327
表示内容を動的に変える.....	1212
表示ファイル.....	75,147
表示ファイル(画面入出力)の使い方.....	210
表示ファイル(帳票印刷)の使い方.....	196
表示ファイル機能.....	210
表示ファイル単体テスト.....	5
表示ファイルの入力中断.....	217
標準のプリンタ.....	178
表の全行を参照する.....	311
表を関連付けてデータを参照する.....	314
ビルダ.....	675
ビルダのエラージャンプ機能.....	663
ビルド.....	651,662
ファイルからの読み込み.....	695
ファイル管理記述項に指定する情報.....	193,199,214
ファイル記述項に指定する情報.....	175,193
ファイル機能全般.....	1047
ファイル参照子.....	84
ファイル識別名.....	85,107,178,952,953,954,955
ファイル識別名定数.....	85,109,179,974
ファイル識別名定数の扱い.....	873
ファイルシステム.....	134
ファイルシステム種別.....	953
ファイル終了条件発生の検出.....	105
ファイル終了条件の発生.....	105
ファイル情報の取得.....	1033
ファイル処理の結果.....	115
ファイル処理の実行.....	107
ファイル追加書き.....	119
ファイルディスクリプタ.....	1005
ファイルの移動.....	131
ファイルの印刷.....	132
ファイルのオープン.....	1004,1009
ファイルの拡張.....	126
ファイルのクローズ.....	1009
ファイルの構造の変換.....	132

ファイルの高速処理.....	115,953,1045	複数翻訳ウィンドウ.....	723
ファイルの高速処理を一括して有効にする指定.....	930	副プログラム.....	25,879
ファイルの削除.....	131	副プログラム名の指定形式.....	48
ファイルの指定.....	733,736	副レコードキー.....	82,100
ファイルの種類.....	79	符号付き10進項目の符号の整形.....	890
ファイルの種類と処理.....	83	符号付き外部10進項目と英数字項目の比較.....	897
ファイルの種類と特徴.....	79	復帰コード.....	239,245,249,1221
ファイルの処理.....	79	復旧.....	125,133
ファイルの処理方法.....	82	物理パス.....	515,1222
ファイルの操作.....	131	不定形式ページ.....	194
ファイルの創成.....	126	不定ファイル.....	115,1008
ファイルの追加書き.....	273	浮動パーティション.....	188
ファイルの登録.....	658	部分参照検査.....	592
ファイルの排他処理.....	916	不変式の移動.....	971
ファイルの排他制御.....	83,112	ブラウザへ送信するCookieデータの登録.....	1104
ファイルの複写.....	131	[プリコンパイラ設定]ダイアログ.....	667
ファイルの連結.....	119	プリコンパイラの登録.....	664
ファイルの割当て.....	107	プリコンパイラを使用したアクセス.....	304
ファイルハンドル.....	1005	プリコンパイルの実行.....	664
ファイルへの書込み.....	695	プリンタ.....	955
ファイル編成.....	85,88,93,99,1006	プリンタ情報ファイル.....	201
ファイル名.....	84	プリンタ情報ファイルの作成.....	201
ファイル名の指定.....	724,743	プリンタ情報ファイルの設定情報.....	201
ファイルを使うプログラム.....	270	プリンタの設定.....	57
ファイルを排他モードにする方法.....	112	プリンタ名.....	177,179,180,181,182,955
ファクトリオブジェクト.....	390,539,546	[プリンタ名の選択]ダイアログ.....	57
ファクトリオブジェクトの寿命.....	399	フレーム.....	1214
ファクトリ定義.....	389	プログラムID.....	807,857
ファクトリメソッド.....	392,463,471	プログラム一覧ウィンドウ.....	687
ファンクションキーの利用者定義.....	219	プログラム開発.....	10
フォルダ.....	661	プログラムからデバッグ機能を起動.....	916
フォルダの作成.....	661	プログラム間連絡機能.....	225
フォルダの種類.....	661	プログラム構造.....	25,435
フォントテーブル.....	171	プログラム構造の概要.....	435
フォントテーブル名.....	955	プログラム構造の指定.....	870
フォーマット定義体.....	1180	プログラム資源.....	755
フォームエレメント.....	764	プログラム制御情報リスト.....	880,909
フォームオーバーレイおよびFCBを使う方法.....	182	プログラムの記述.....	213,221
フォームオーバーレイパターン.....	155,182	プログラムの形式.....	16
フォームオーバーレイパターンの出力.....	186	プログラムの作成.....	15
フォームオーバーレイパターンのファイルの拡張子.....	958	プログラムの実行.....	217,223,261,345,1004
フォームオーバーレイパターンのファイル名.....	958	プログラムの実行順序を変更する.....	690
フォームオーバーレイパターンのファイル名の形式.....	958	プログラムの実行とスレッドモード.....	542
フォームオーバーレイパターンのフォルダ.....	957	プログラムの特徴に合わせたデバッグ方法.....	696
フォームオーバーレイパターン名.....	158,164	プログラムの名前.....	755
フォームオーバーレイパターン名の個数.....	164	プログラムの雛形.....	753
フォームオーバーレイパターン焼付け回数.....	158	プログラムの翻訳.....	254,680
フォームオーバーレイパターンを使うプログラム.....	185	プログラムの翻訳・リンク.....	217,223,345,613
副供給口.....	160	プログラムの翻訳・リンク方法と出力情報との関係.....	611
副キー.....	82,100	プログラムの呼出し経路をたどる.....	691
復元.....	472	プログラムのリンク.....	254,680
複写.....	125	プログラム名定数.....	974
複写数.....	158	プログラム呼出し規約検査.....	594
複数行指定ホスト変数.....	323	プログラムを作成・編集する.....	15
複数コネクションでのカーソル操作.....	321	プログラムを実行する.....	35
複数のソースファイルを翻訳する.....	726	プログラムを翻訳する.....	18
複数翻訳.....	22	プログラムをリンクする.....	23





