FUJITSU

FUJITSU Software NetCOBOL V11.0

ユーザーズガイド

Windows(64)

B1WD-3192-01Z0(00)
2013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システムで動作するアプリケーションを開発するCOBOL開発システムです。

NetCOBOLシリーズについて

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

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

登録商標について

- Microsoft、Windows、Windows Server、Windows Vista、Visual C++、Visual Basic、ActiveXは、米国Microsoft Corporationの米国およびその他の国における登録商標または商標です。
- UNIXは、米国およびその他の国におけるオープン・グループの登録商標です。
- OracleとJavaは、Oracle Corporationおよびその子会社、関連会社の米国およびその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。
- HP、HP-UXは、米国Hewlett-Packard Companyの商標です。
- 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) 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 Foundation Microsoft(R) Windows Server(R) 2008 R2 Standard Microsoft(R) Windows Server(R) 2008 R2 Enterprise Microsoft(R) Windows Server(R) 2008 R2 Datacenter	Windows Server 2008 R2 または Windows Server 2008
Microsoft(R) Windows Server(R) 2008 for Itanium-Based Systems	Windows Server 2008(Itanium)
Microsoft(R) Windows Server(R) 2008 Foundation Microsoft(R) Windows Server(R) 2008 Standard Microsoft(R) Windows Server(R) 2008 Standard without Hyper-V(TM) Microsoft(R) Windows Server(R) 2008 Enterprise Microsoft(R) Windows Server(R) 2008 Enterprise without Hyper-V(TM) Microsoft(R) Windows Server(R) 2008 Datacenter Microsoft(R) Windows Server(R) 2008 Datacenter without Hyper-V(TM)	Windows Server 2008
Microsoft(R) Windows Server(R) 2003, Standard Edition	Windows Server 2003

正式名称	略称
Microsoft(R) Windows Server(R) 2003, Enterprise Edition Microsoft(R) Windows Server(R) 2003 R2, Standard Edition Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition	
Microsoft(R) Windows Server(R) 2003, Standard x64 Edition Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition	Windows Server 2003 または Windows Server 2003(x64)
Microsoft(R) Windows Server(R) 2003, Enterprise Edition for Itanium-based Systems Microsoft(R) Windows Server(R) 2003, Datacenter Edition for Itanium-based Systems	Windows Server 2003(Itanium)
Windows(R) 8 Windows(R) 8 Pro Windows(R) 8 Enterprise	Windows 8 または Windows 8(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)
Windows Vista(R) Home Basic Windows Vista(R) Home Premium Windows Vista(R) Business Windows Vista(R) Enterprise Windows Vista(R) Ultimate	Windows Vista
Microsoft(R) Windows(R) XP Home Edition Operating System Microsoft(R) Windows(R) XP Professional Operating System	Windows XP
Microsoft(R) Visual C++(R) development system	Visual C++
Microsoft(R) Visual Basic(R) programming system	Visual Basic
Oracle Solaris 10 Oracle Solaris 11	Solaris
PowerSORT PowerSORT Server PowerSORT Workstation	PowerSORT

・ 以下をすべて指す場合は、「32ビットWindows」と表記します。

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

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

  ― Windows Server 2012

  ― Windows Server 2008 R2

  ― Windows 8(x64)

  ― Windows 7(x64)

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

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。

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 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M 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。

本書の位置付け

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

マニュアル名称 (注1)	製品名	使用目的
FORM ユーザーズガイド FORMヘルプ (注2) PowerFORMヘルプ (注2)	FORM FORMオーバレイオプション(オプション製品)	画面帳票定義体およびフォームオーバレイパターンの作成
MeFt ユーザーズガイド	MeFt	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 (コンポーネント) (注3)	画面帳票定義体を使用したプログラムのWeb環境でのリモート実行
Jアダプタクラスジェネレータ ユーザーズガイド	Jアダプタクラスジェネレータ(コンポーネント) (注3)	COBOLプログラムからJavaのクラスを利用するためのアダプタクラスの生成
NetCOBOL Studio ユーザーズガイド	64ビットWindowsで動作するNetCOBOLシリーズ製品	NetCOBOL Studioを使用したCOBOLアプリケーションの開発
PowerSORT Serverユーザーズガイド	PowerSORT	PowerSORTを使用した整列併合を行うプログラムの実行

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

注2 ： ヘルプまたはオンラインマニュアルは、各製品の中にあります。

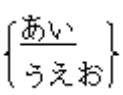
注3 ： NetCOBOLシリーズに同梱されています。

注意

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

本書で使用する書体と記号

書体および記号	意味
［参照］	参照先を示します。
→	操作結果を示します。

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

その他の注意事項

- 本書に記載されている画面は、使用されているシステムにより、実際の画面と異なる場合がありますので注意してください。
- 本書では、“COBOL文法書”で“原始プログラム”と記述されている用語を“ソースプログラム”と記述しています。
- OSIV/MSP、OSIV/XSPなどのOSIV系システムを総称して、“OSIV系システム”と記述しています。

輸出管理規制について

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

2013年8月

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

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
- 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

また、サーバサイドアプリケーション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
- バッチタイプアプリケーションの開発・運用

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

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。

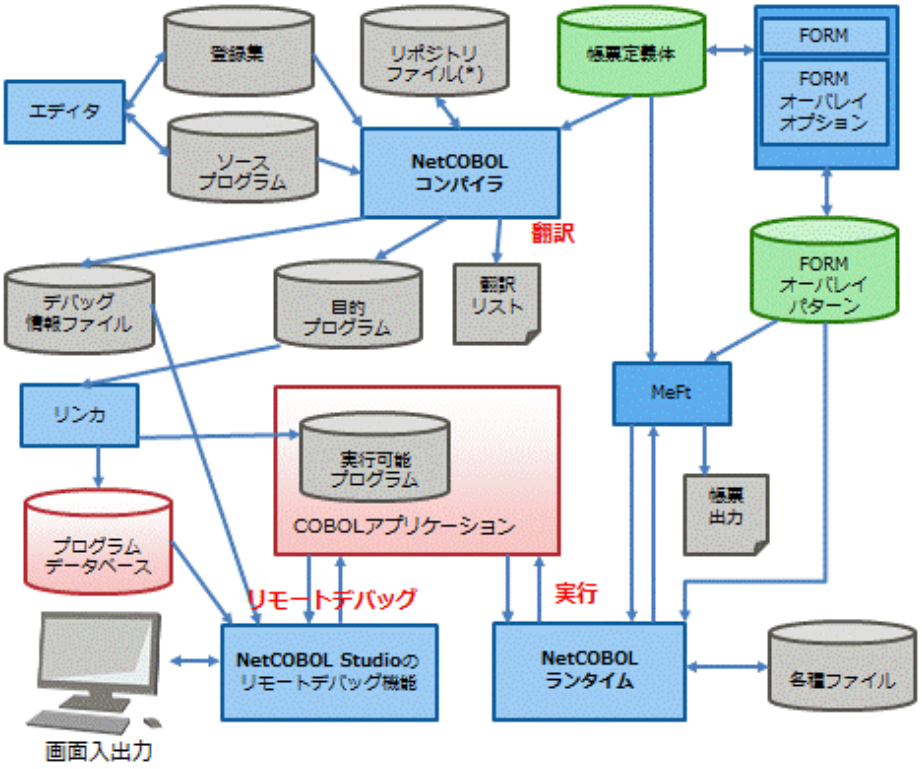
- エラーの種別
- エラー発生箇所
- エラー発生箇所への呼出経路
- システムの種別とバージョン情報
- コマンドライン
- プロセスの環境変数
- COBOL実行環境情報
- システム上のプロセス一覧
- プロセス内のモジュール一覧
- プロセス内のスレッド情報
- スレッドごとのスタック情報

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。

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のシステムで設定する。

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。

設定時	環境変数名	設定する内容	条件	
共通	PATH	COBOLコンパイラのインストール先フォルダ	NetCOBOLを使用する場合	必須
		COBOLランタイムシステムのインストール先フォルダ		
		MeFtのインストール先フォルダ	帳票定義体を使用する場合	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	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1)	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_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

設定時	環境変数名	設定する内容	条件	
リンク	LIB	COBOLランタイムシステム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	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
リモート開発	COB_RDENV_X64	リモートビルド時にユーザ固有設定を行うバッチファイル(注2)	リモートビルド時にユーザ固有の環境を設定する場合	任意

- 注1
  - 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。
  - 環境変数名がNetCOBOLでサポートする環境変数名と重なる名前を指定してはいけません。
  - 環境変数COB_登録集名には、フォルダは1つしか指定できません。

例

COPY文の記述例: COPY TEST OF TESTLIB1

```
COB_TESTLIB1=C:¥COPYLIB
```

- 注2
  詳細は“NetCOBOL Studio ユーザーズガイド”を参照してください。

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COB_COBCOPY
- 環境変数COB_REPIN

例

```
COB_COBCOPY=C:¥COPY1;"C:¥Program Files¥NetCOBOL"
```

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
```

環境変数SMED_SUFFIXの指定がない場合の検索順序

1. 拡張子(PMD)
2. 拡張子(SMD)

注意

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。

環境変数COB_LIBSUFFIXの指定がない場合の検索順序

1. 拡張子CBL
2. 拡張子COB
3. 拡張子COBOL

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 COBOLコマンド”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)"
```

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。

-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
FORM	プログラムが出力する帳票の定義および帳票設計ツール(PowerFORM)の提供
FORMオーバレイオプション	プログラムが出力するオーバレイパターンの定義
MeFt	画面や帳票の入出力
MeFt/Web	Webブラウザを使用した画面や帳票の入出力
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	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。

FORM

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

FORMオーバレイオプション

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

MeFt

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

MeFt/Web

MeFt/Webは、Webサーバ上で動作するMeFt経由の画面や帳票の入出力を行うプログラムのディスプレイ装置やプリンタ装置に対する入出力を、Webブラウザに対して行うことができます。

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

PowerSORT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-STEPCOUNTER
- SIMPLIA/TF-EXCOUNTER

NetCOBOLシリーズ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。

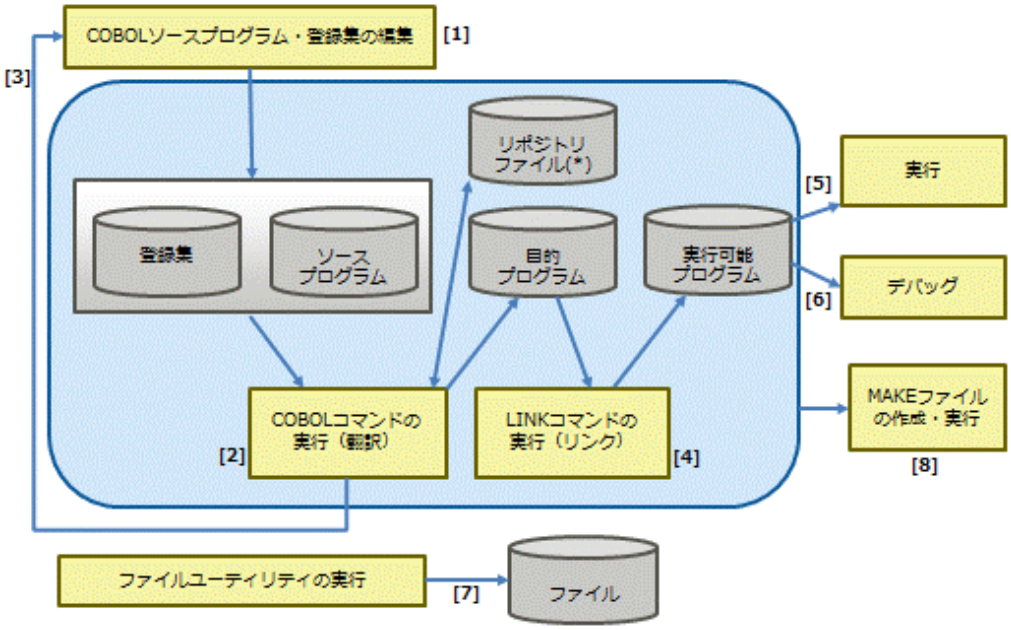
1.2.3 資源一覧

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	任意	smd pmd

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
		MeFt/Web		
FORMオーバレイパターン	フォームオーバレイパターン情報	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	コンパイラ	任意	exe
ダイナミックリンクライブラリ				
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
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	デバッガ	ソース名.svd	svd

1.3 プログラム開発



＊：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]

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]

COBOLコマンドを実行し、プログラムの翻訳を行います。
オブジェクト指向プログラム開発を行う場合、開発資源にリポジトリファイルが加わるため、資源の依存関係を把握しながらCOBOLコマンドを実行し、プログラムの翻訳を行います。[参照]“第3章 プログラムを翻訳する”

[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]

LINKコマンドを実行し、プログラムのリンクを行います。[参照]“第4章 プログラムをリンクする”

[5]

COBOLアプリケーションを実行します。[参照]“第5章 プログラムを実行する”

[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、NetCOBOL Studioのデバッグ機能を使ってデバッグを行います。[参照]“NetCOBOL Studio ユーザーズガイド”

[7]

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”

[8]

MAKEファイルの内容については、“16.2.6 MAKEファイル”または例題プログラムに添付されているMAKEファイルを参照してください。

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。

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。

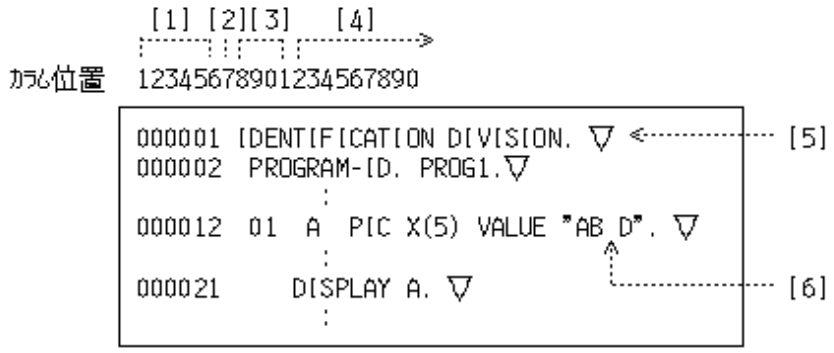
エディタの起動

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。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”

参考

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.3.47 SRF（正書法の種類）”

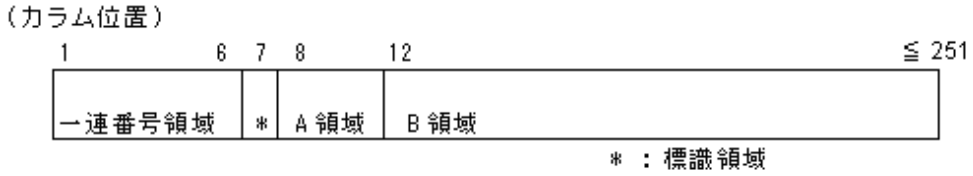
以下にそれぞれの形式について説明します。なお、各行の区切りの改行文字は、行の一部とはみなされません。

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。

可変形式

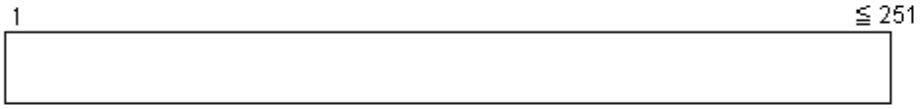
可変形式では、COBOLソースプログラムの1行の長さを、251バイト以下の任意のバイト数で記述します。



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コマンドに指定しますが、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。

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、COBOL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¥Sample01¥Sample1.COBとします。

1. NetCOBOLコマンドプロンプトは、〔スタート〕>〔すべてのアプリ〕> 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。(注1)
2. COBOLコマンドを指定して実行します。

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

   ー -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¥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。

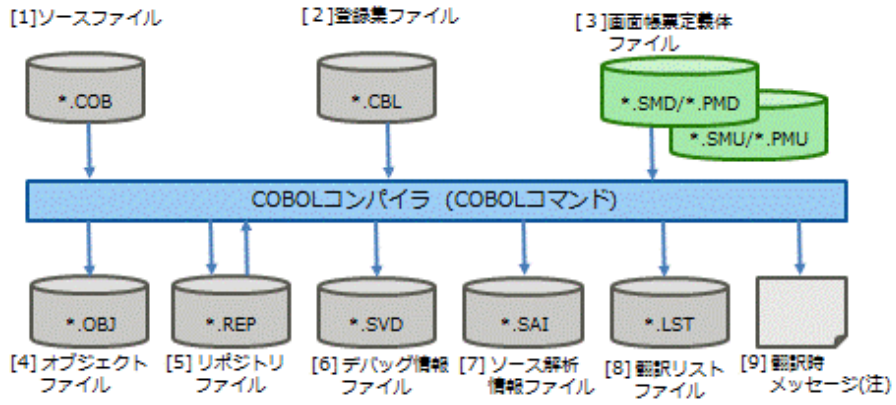
注1)Windows 8を使って説明しています。

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。COBOLコンパイラは、以下のファイルを使用します。

- ソースファイル(*.COB)
- 登録集ファイル(*.CBL)
- 画面帳票定義体ファイル(*.SMD/*.PMD/*.SMU/*.PMU)
- オブジェクトファイル(*.OBJ)
- リポジトリファイル(*.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、コマンドプロンプトに翻訳時メッセージが出力されます。翻訳メッセージの詳細は、“B.1 診断メッセージリスト”を参照してください。

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)	-I	COB_COBCOPY COB_LIBSUFFIX COB_登録集名
[3]	画面帳票定義体	画面帳票定義体名 .SMD (注4) .PMD (注4)(注5) .SMU (注5)(注10) .PMU (注5)(注10)	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)	-m	SMED_SUFFIX
[4]	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 -do	―
[5]	クラス情報(注7)	クラス名.REP	入力	リポジトリ段落を指定したソースプログラムを翻訳する場合	-R -dr	COB_REPIN
[6]	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 -Dt	―
[7]	ソース解析情報(注8)	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 -ds	―
[8]	翻訳リスト(注9)	ソースファイル名 .LST	出力	翻訳リストを出力する場合	-P -dp	―
[9]	翻訳時メッセージ	―		コマンドプロンプト上に表示する	―	―

注1

COBOLソースファイル名に、以下の拡張子を使用してはなりません。

― 拡張子OBJ

- SVD
- REP
- SAI
- LST

注2

拡張子は、環境変数COB_LIBSUFFIXを使って任意の文字列に変更できます。環境変数COB_LIBSUFFIXを指定しない場合、以下の順で検索します。[参照]“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. 翻訳コマンドの-Iオプション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]“J.1.12 -I（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.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。

IN/OF 登録集名の指定があるCOPY文を記述した場合

1. 登録集名に関連付けたフォルダを検索します。登録集名にフォルダを関連付けていない場合、エラーとなります。
2. 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

拡張子は、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]“1.2.1 環境変数の設定”

注5

拡張子PMD、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、“J.1.14 -m（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

リポジトリファイルの詳細は、“16.2.4.2.1 リポジトリファイル”を参照してください。

注8

ソース解析情報ファイルは、ソースを解析するための情報です。

注9

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10

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。

- -Pオプションを指定して翻訳を行う。[参照]“J.1.16 -P（各種翻訳リストの出力および出力先の指定）”

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ソースプログラムの翻訳は、cobolコマンドを使用します。翻訳コマンドを使った翻訳操作については、“3.4 翻訳コマンド”で説明しています。

注意

- 1つのファイル中に複数の翻訳単位を含むソースプログラムを翻訳する場合、翻訳オプションNAMEを指定してください。[参照]“A.3.28 NAME(翻訳単位ごとのオブジェクトファイルの出力の可否)”
- 1つのCOBOL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”

3.4 翻訳コマンド

翻訳コマンド(cobol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 COBOL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>cobol -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。

例2 : 複数のソースファイルを翻訳する(複数翻訳)

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

test1.cobは主プログラム、test2.cobは服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。

第4章 プログラムをリンクする

本章では、必要な資源、プログラム構造、手順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¥Sample01¥Sample1.OBJとします。

NetCOBOL コマンドプロンプト

```
c:¥COBOL¥Samples¥Sample01>LINK /OUT:Sample1.EXE Sample1.OBJ F4AGCIMP.LIB LIBCMT.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¥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。

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"

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)
- ダイナミックリンクライブラリ(*.DLL)
- 実行可能ファイル(*.EXE)
- プログラムデータベース(*.PDB)

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
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。
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。
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。
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。
実行可能プログラム	オブジェクトファイル名.EXE	出力	リンクが成功した場合に作成されます。

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
	または 任意の名前.EXE		
プログラムデータベース	オブジェクトファイル名.PDB または 任意の名前.PDB	出力	リンクオプション/DEBUGを指定した場合に作成されます。このファイルは、診断機能、NetCOBOL Studioのデバッグ機能、Visual C++のデバッグ機能などが使用します。

注意

アプリケーションのデバッグを容易にするために、リンク時にはリンクオプション/DEBUGを指定することを推奨します。

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、Windows(x64)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。

注意

64ビットの目的プログラムと32ビットの目的プログラムを混在させてリンクを行うことはできません。
[参照]“4.4.4 注意事項”

4.2.4 インポートライブラリ

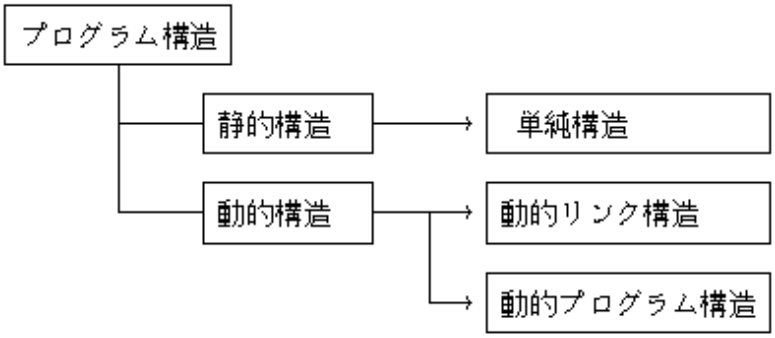
インポートライブラリは、アプリケーション実行時に、実行中のアプリケーションとDLLとの間にダイナミックリンクを確立するための情報を含んでいます。動的リンク構造で実行可能ファイルを作成する場合には、必ずインポートライブラリが必要となります。必要なライブラリが実行時にどこにあるかを探すための情報をインポートライブラリから実行可能プログラムにコピーします。つまり、インポートライブラリはアプリケーションとDLLとの仲立ちの役目をするものです。

4.3 プログラム構造

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

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

図4.1 プログラム構造

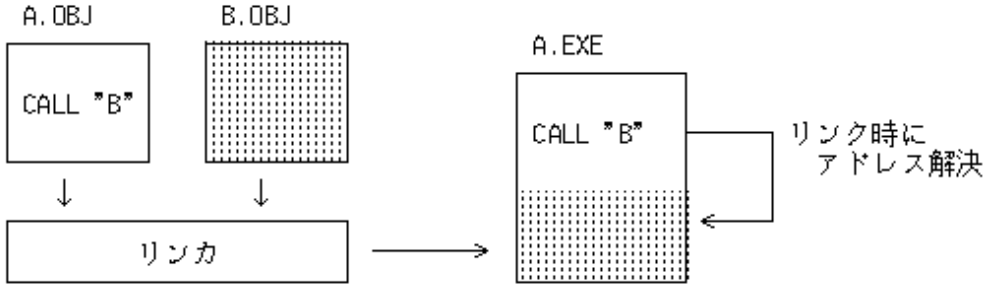


4.3.1 静的構造

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

単純構造

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



4.3.2 動的構造

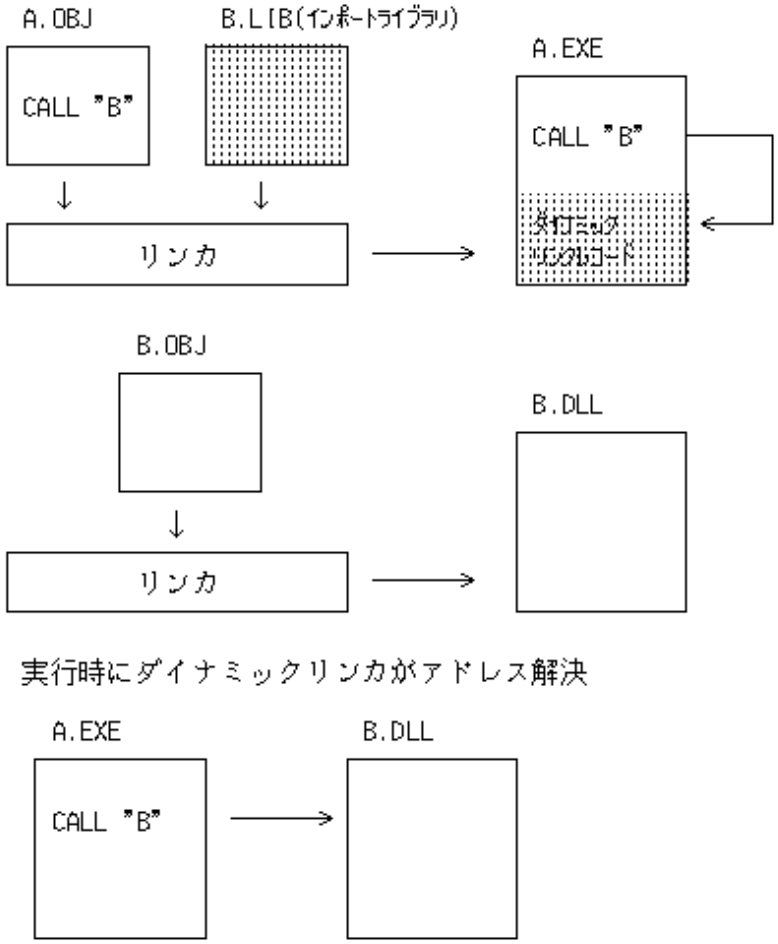
動的構造とは、実行時に必要となるモジュールを、リンカによっていくつかのDLLとし、プログラムの実行時に必要なDLLを仮想記憶上にロードし、実行させる構造です。

DLLに含まれるモジュールは、そのモジュールを利用するすべてのプログラムによって共用されるため、ファイルを格納するディスク空間、および複数のプログラムを実行したときの仮想記憶の節約となります。また、プログラムを変更した場合、変更したプログラムを含むDLLだけを再リンクすればよいため、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カが行います。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、“10.1.3 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.3.11 DLOAD(プログラム構造の指定)”を参照してください。

表4.2 CALL文の書き方/翻訳オプションとプログラム構造の関係

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

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

4.3.4 プログラム構造とCANCEL文の関係

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

プログラム構造とCANCEL文の関係を以下に示します。

表4.3 プログラム構造とCANCEL文の関係

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

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

4.4 リンク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、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。

4.4.1 LINKコマンド

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

コマンドの形式の詳細については、“J.2 LINKコマンド”を参照してください。

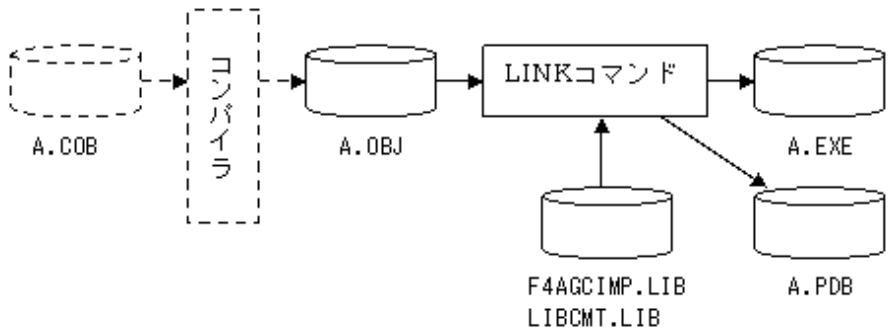
LINKコマンドを使う場合には、以下に示すライブラリを指定する必要があります。

- F4AGCIMP.LIB
- LIBCMT.LIB
  LIBCMT.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。

4.4.2 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。

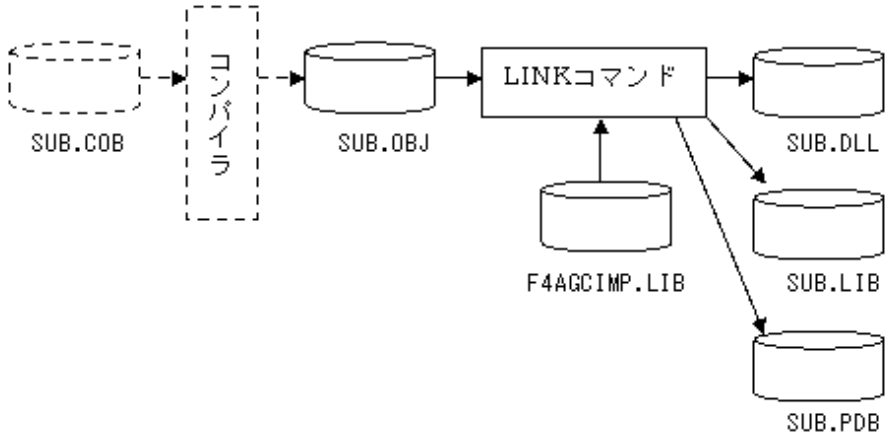
例1:1つの目的プログラムから実行可能プログラムを作成する場合



```
LINK A.OBJ F4AGCIMP.LIB LIBCMT.LIB /DEBUG /OUT:A.EXE
```

/DEBUG：デバッグ情報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

例2:1つの目的プログラムからDLLを作成する場合

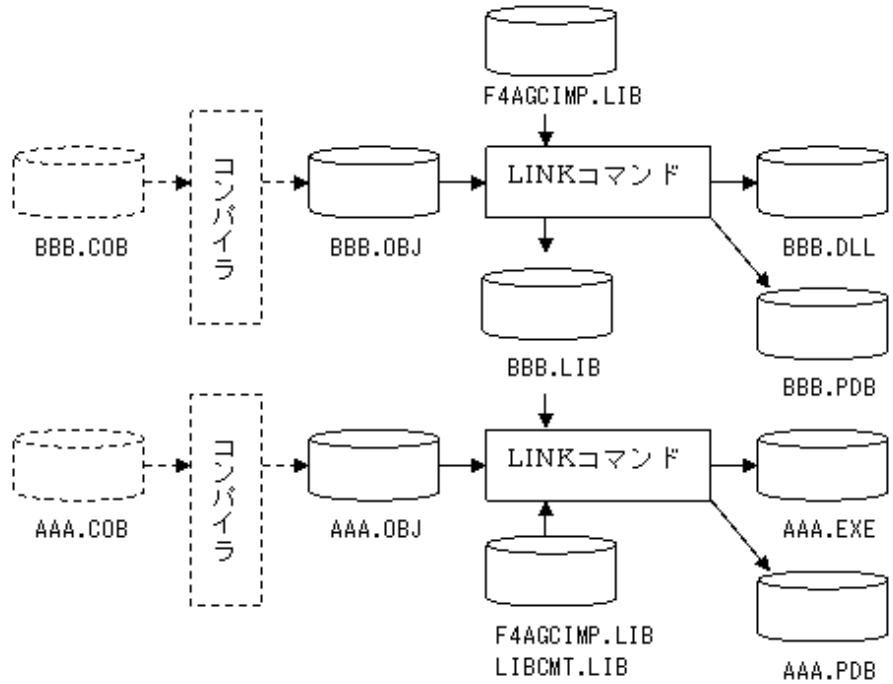



```
LINK SUB.OBJ F4AGCIMP.LIB /DLL /NOENTRY /DEBUG /OUT:SUB.DLL
```

/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
/NOENTRY：DLLを作成するときに指定（DLLからC関数を呼び出さない場合のみ）
/DEBUG　：デバッグ情報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

DLLからC関数を呼び出す場合には、/NOENTRYを指定せずにLIBCMT.LIBを指定します。

例3: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 F4AGCIMP.LIB /DLL /NOENTRY /DEBUG /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
/NOENTRY：DLLを作成するときに指定（DLLからC関数を呼び出さない場合のみ）
/DEBUG　：デバッグ情報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

DLLからC関数を呼び出す場合には、/NOENTRYを指定せずにLIBCMT.LIBを指定します。

・ 実行可能プログラムの作成

```
LINK AAA.OBJ F4AGCIMP.LIB LIBCMT.LIB BBB.LIB /DEBUG /OUT:AAA.EXE
```

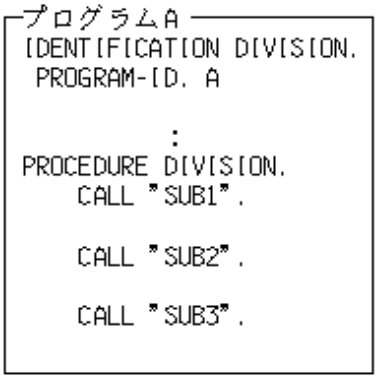
/DEBUG：デバッグ情報を作成するときに指定
/OUT　：メイン出力ファイルの名前を指定

4.4.3 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



SUB1.DLLを作成する

```
LINK SUB1.OBJ F4AGCIMP.LIB /DLL /NOENTRY /DEBUG /OUT:SUB1.DLL
```

SUB1.OBJ　　：プログラムSUB1のオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ

LINKすると、SUB1.DLL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 F4AGCIMP.LIB /DLL /NOENTRY /DEBUG /OUT:SUB2.DLL
```

SUB2.OBJ　　：プログラムSUB2のオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ

LINKすると、SUB2.DLL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 F4AGCIMP.LIB /DLL /NOENTRY /DEBUG /OUT:SUB3.DLL
```

SUB3.OBJ　　：プログラムSUB3のオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ

LINKすると、SUB3.DLLとインポートライブラリSUB3.LIBが作成されます。

SUB1.DLL、SUB2.DLLおよび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、SUB2.DLLおよびSUB3.DLLのインポートライブラリが結合されたインポートライブラリSUB123.LIBが作成されます。

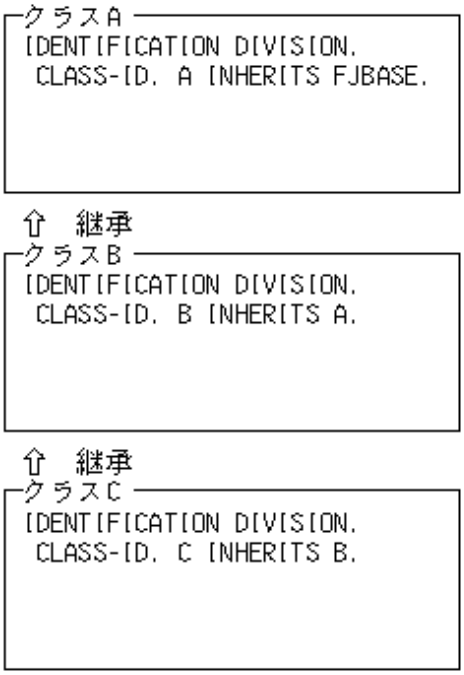
作成されたインポートライブラリを使用して、プログラムAをLINKします。

プログラムAをLINKする

```
LINK A.OBJ F4AGCIMP.LIB LIBCMT.LIB SUB123.LIB /DEBUG /OUT:A.EXE
```

A.OBJ　　：プログラムA のオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ
LIBCMT.LIB　：C ランタイムライブラリ
SUB123.LIB　：SUB1.DLL、SUB2.DLLおよび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 F4AGCIMP.LIB /DLL /NOENTRY /DEBUG /OUT:A.DLL
```

A.OBJ　　：クラスA のオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ

LINKすると、A.DLL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 F4AGCIMP.LIB A.LIB /DLL /NOENTRY /DEBUG /OUT:B.DLL
```

B.OBJ　　：クラスB のオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ
A.LIB　　：クラスA のインポートライブラリ

LINKすると、B.DLL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。

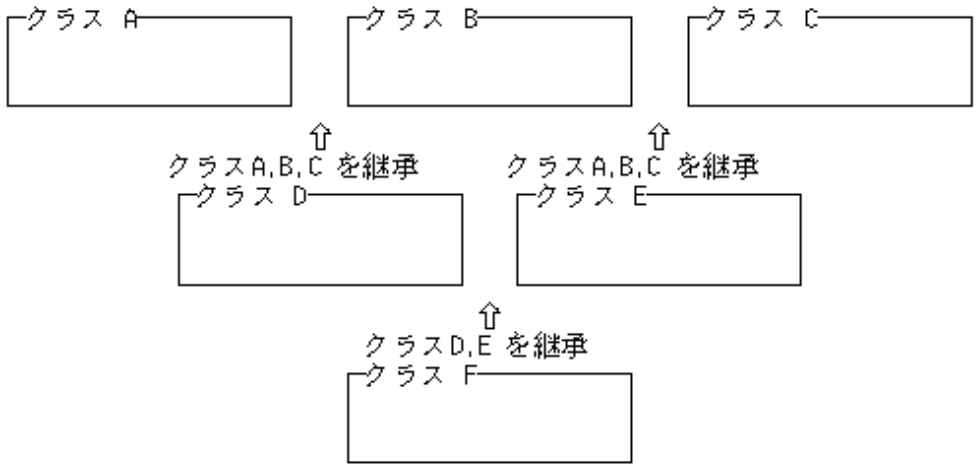
クラスCのDLLを作成する

```
LINK C.OBJ F4AGCIMP.LIB AB.LIB /DLL /NOENTRY /DEBUG /OUT:C.DLL
```

C.OBJ　　：クラスC のオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ
AB.LIB　　：クラスA とクラスB のインポートライブラリを１つに結合したもの

LINKすると、C.DLL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 F4AGCIMP.LIB /DLL /NOENTRY /DEBUG /OUT:A.DLL
```

A.OBJ　　：クラスA のオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ

LINKすると、A.DLL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 F4AGCIMP.LIB /DLL /NOENTRY /DEBUG /OUT:B.DLL
```

B.OBJ　　：クラスB のオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ

LINKすると、B.DLL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 F4AGCIMP.LIB /DLL /NOENTRY /DEBUG /OUT:C.DLL
```

C.OBJ　　：クラスC のオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ

LINKすると、C.DLL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 F4AGCIMP.LIB ABC.LIB /DLL /NOENTRY /DEBUG /OUT:D.DLL
```

D.OBJ　　：クラスD のオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ
ABC.LIB　　：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１つに結合したもの

LINKすると、D.DLL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 F4AGCIMP.LIB ABC.LIB /DLL /NOENTRY /DEBUG /OUT:E.DLL
```

E.OBJ　　：クラス Eのオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ
ABC.LIB　　：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１つに結合したもの

LINKすると、E.DLL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 F4AGCIMP.LIB ABCDE.LIB /DLL /NOENTRY /DEBUG /OUT:F.DLL
```

F.OBJ　　：クラスFのオブジェクトファイル
F4AGCIMP.LIB：COBOLランタイムシステムのインポートライブラリ
ABCDE.LIB　：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１つに結合したもの

注意

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

4.4.4 注意事項

64ビットの目的プログラムと32ビットの目的プログラムを混在させてリンクを行うことはできません。混在している場合は、リンク時に以下のリンクメッセージが出力されます。

単純構造の場合

リンクメッセージLNK1112が出力されます。

例

```
link /out:a.exe a.obj b.obj f4agcimp.lib libcmt.lib
--------------------------------------------------------------------------------
b.obj : fatal error LNK1112: モジュールのコンピューターの種類 'X86' は対象コンピューターの種類 'x64' と競合しています。
```

a.obj ： 64ビットの目的プログラム
b.obj ： 32ビットの目的プログラム

動的リンク構造の場合

リンクメッセージLNK2001が出力されます。

例

```
link /out:a.exe a.obj b.lib f4agcimp.lib libcmt.lib
--------------------------------------------------------------------------------
a.obj : error LNK2001: 外部シンボル "B" は未解決です。
```

a.obj ： 64ビットの目的プログラム
b.lib ： 32ビットのインポートライブラリ

第5章 プログラムを実行する

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

5.1 サンプルプログラムの実行

リンクが終わったら、作成した実行可能ファイルを実行してみましょう。

ここでは、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¥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せんが、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¥Sample01)に実行用の初期化ファイル(COBOL85.CBR)を作成します。
4. 必要な実行環境情報を“設定”し、“適用”します。
   → 実行用の初期化ファイル(C:¥COBOL¥Samples¥Sample01¥COBOL85.CBR)の内容が変更されます。
5. 〔ファイル〕メニューから“閉じる”を選択します。
6. 〔ファイル〕メニューから“終了”を選択します。
   → 実行環境設定ツールを終了します。

では、実際に実行可能ファイルを実行してみましょう。

1. NetCOBOLコマンドプロンプト上で、Sample1.EXEを実行します。

```
c:¥COBOL¥Samples¥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。

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Windows(x64)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。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

   各ウィンドウシステムで使用されているアプリケーションの実行方法を使用して、COBOLプログラムを実行します。

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となる情報です。

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

   ― システムのコントロールパネルで設定する

     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と便利です。

   ― 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。

   ― テキストエディタで設定する。

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。

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

エントリ情報を設定する方法には、以下があります。

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。

参考

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。

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。
  - 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。
  - 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。
- 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。

5.4 プログラムの実行

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

例えば、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

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
- エディタを使用しての参照および更新

・ 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［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

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。

-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、ABC.INIが指定されました(PROG1.EXEが主プログラムの場合)。

参考

主プログラムが多言語(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と、オペレータの入力待ちなしにCOBOLプログラムを実行することができます。

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。

― 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

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。

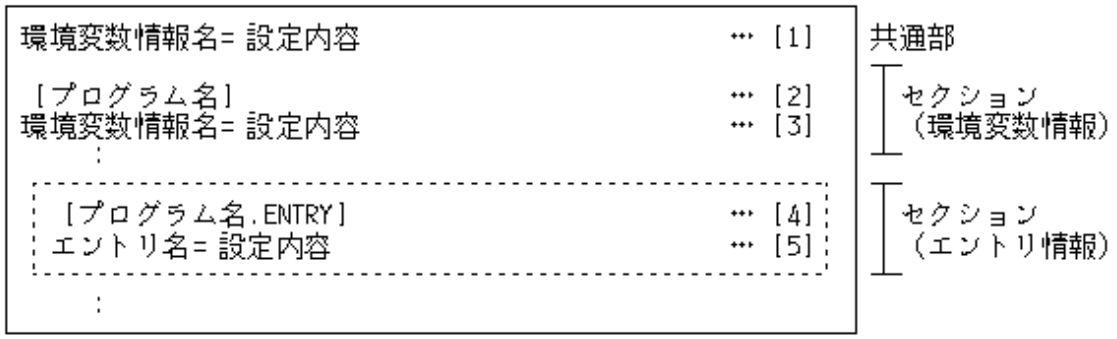
なお、実行用の初期化ファイルがなくても、プログラムは実行できます。

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(注1)に“.ENTRY”を付加した名前を指定します。このセクションは、1つのプログラム名に対して1つだけ記述できます。

[5] プログラムごとのエントリ情報を記述します。(注2)エントリ情報の指定形式については、“5.6 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。

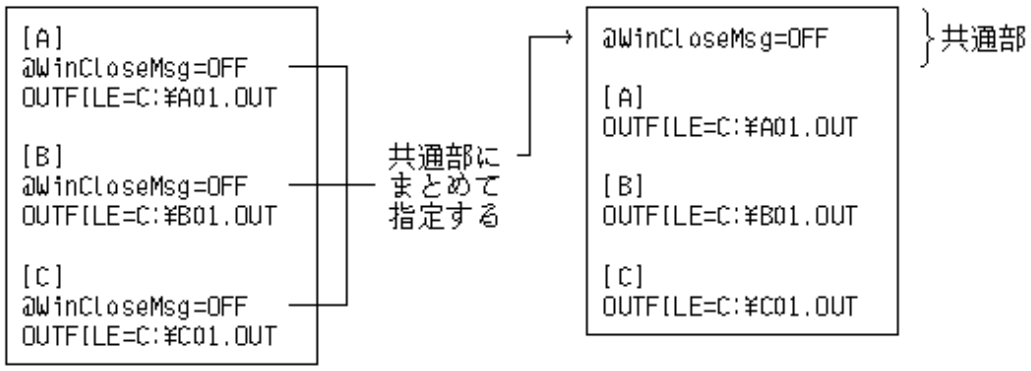
注1 : 他言語プログラムからCOBOLプログラムをJMPCINT2およびJMPCINT3を使用して呼び出す場合、最初に呼び出すCOBOLプログラムのプログラム名を指定してください。COBOLの主プログラムおよび実行環境については、“10.1.2 COBOLの言語間の環境”を参照してください。

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          ------
@CnslWinSize=(80,24)                ↑
@CnslBufLine=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]                       ↑
A=X.DLL                           PROG1のエントリ情報
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。

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

例

検索順序の例

上記では、実行用の初期化ファイルの検索順序は次のようになります。

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

ただし、JMPCINTC/JMPCINTBで実行用の初期化ファイルを指定した場合またはコマンドラインで実行用の初期化ファイルを指定した場合には、上記の検索は行われずに指定したファイルが有効になります。[参照]“5.5.5.1 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	―

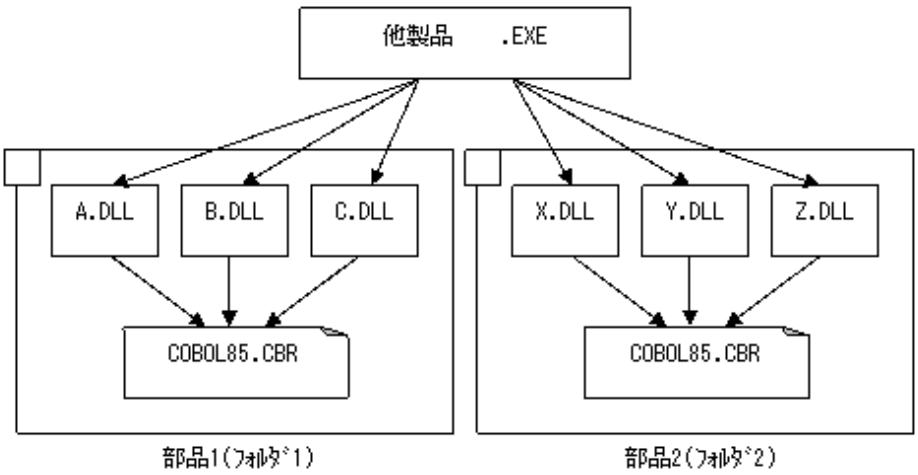
5.5.4 DLL配下にある実行用の初期化ファイルを使用する

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エントリオブジェクトを結合する処理が必要になります。

注意

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

DLLエントリオブジェクトの結合

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

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

- F4AGCBDM.OBJ ： COBOL のDLL を作成する場合に指定します。
- F4AGMLDM.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 F4AGCBDM.OBJ F4AGCIMP.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
F4AGCBDM.OBJ：DLL エントリオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ
KERNEL32.LIB：Windows関数のインポートライブラリ

注意

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

COBOLと他言語を結合してDLLを作成する場合

```
LINK CPROG.OBJ COBSUB.OBJ F4AGMLDM.OBJ F4AGCIMP.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
F4AGMLDM.OBJ：DLL エントリオブジェクトファイル
F4AGCIMP.LIB：COBOL ランタイムシステムのインポートライブラリ
KERNEL32.LIB：Windows関数のインポートライブラリ
Cランタイムライブラリ：
　　C プログラムを/MT オプションで翻訳した場合はLIBCMT.LIBを指定してください。
　　C プログラムを/MD オプションで翻訳した場合はMSVCRT.LIBを指定してください。

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。

```
int JMPCINTC(int 0, void &param)
```

param

0	エントリ数
4	未使用域（注１）
8 16	実行用の初期化ファイル名文字列の先頭アドレス（注２）

注1 ： 0で初期化してください。

注2 ： 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

Imports文、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。

Declare 句への記述

```
Imports System.Runtime.InteropServices
        :
Private Declare Sub JMPCINT2 LIB "F4AGPRCT.DLL" ()
Private Declare Sub JMPCINT3 LIB "F4AGPRCT.DLL" ()
Private Declare Function JMPCINTB LIB "F4AGPRCT.DLL"_  (注1)
  (ByVal A As Integer, <MarshalAs(UnmanagedType.LPStr)> ByVal D As String) As Integer
```

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
Dim ANS As Integer

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。)

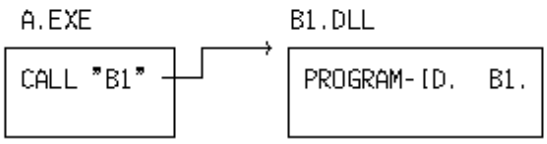
JMPCINTBの返却値	意味
0	正常終了
-1	JMPCINT2の未呼出し／機能コードの指定誤り／作業領域不足
1	多重呼出し (JMPCINTBがすでに呼び出されている場合を意味します。実行用の初期化ファイル名は、最初に実行されたJMPCINTBで指定されたものが有効となります。)

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。

マルチスレッドで利用する場合は、“18.10.3 動的プログラム構造”を参照してください。

5.6.2 エントリ情報の形式

5.6.2.1 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”

注意

DLL名の拡張子は、“DLL”でなければなりません。

5.6.2.2 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”

5.6.3 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、“5.6.2.1 副プログラム名の指定形式”、“5.6.2.2 二次入口点名の指定形式”および“16.3.2.4.4 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。

エントリ情報ファイル名の指定

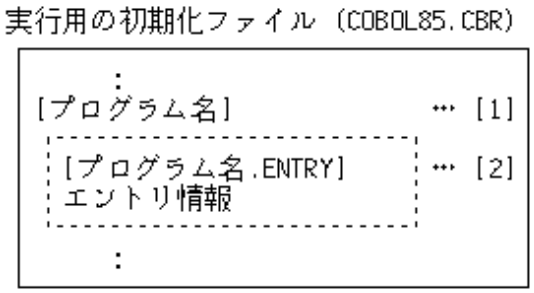
エントリ情報ファイルを使用する場合、環境変数情報@CBR_ENTRYFILEにエントリ情報ファイル名を指定します。[参照]“C.2.25 @CBR_ENTRYFILE(エントリ情報ファイルの指定)”

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
```

5.6.4 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



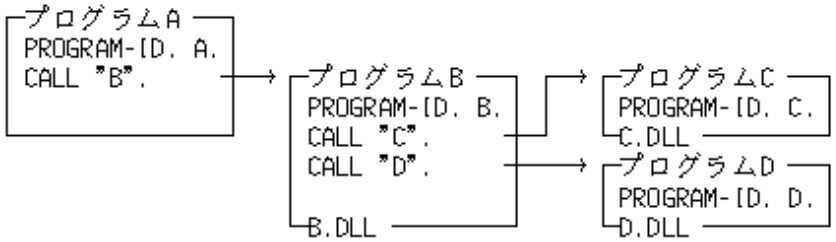
[1] 主プログラムのプログラム名(PROGRAM-ID段落に指定した名前)をセクション名として指定します。

[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。

5.6.5 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
```

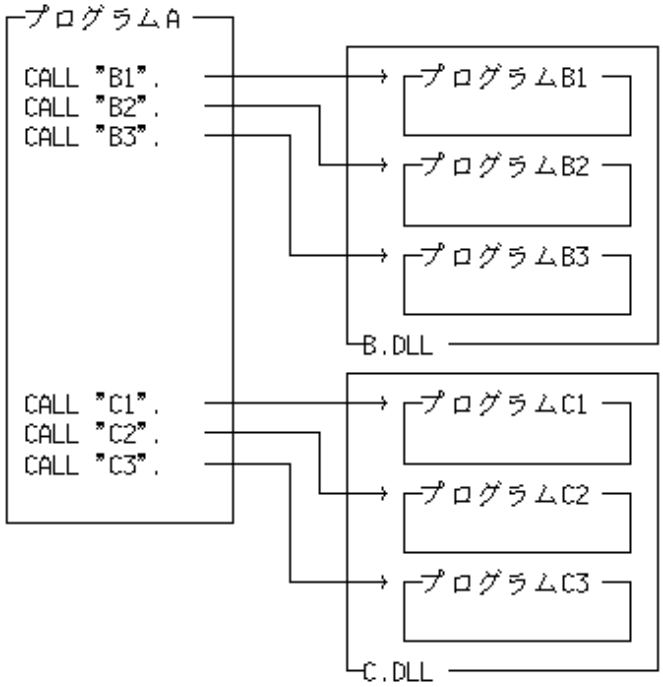
(2) 実行用の初期化ファイルに指定する場合

```
[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

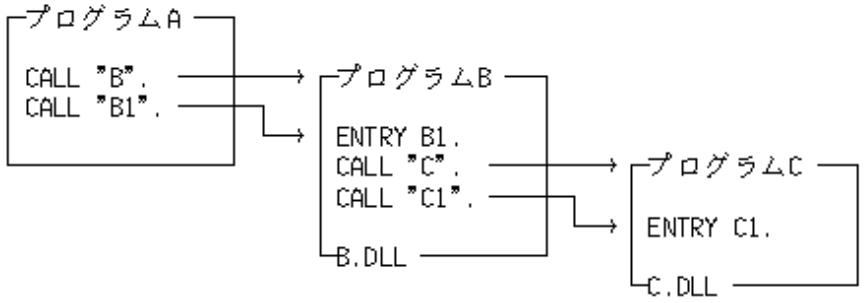
```
[A.ENTRY]
B1=B.DLL
B2=B.DLL
B3=B.DLL
C1=C.DLL
C2=C.DLL
C3=C.DLL
```

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。

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
```

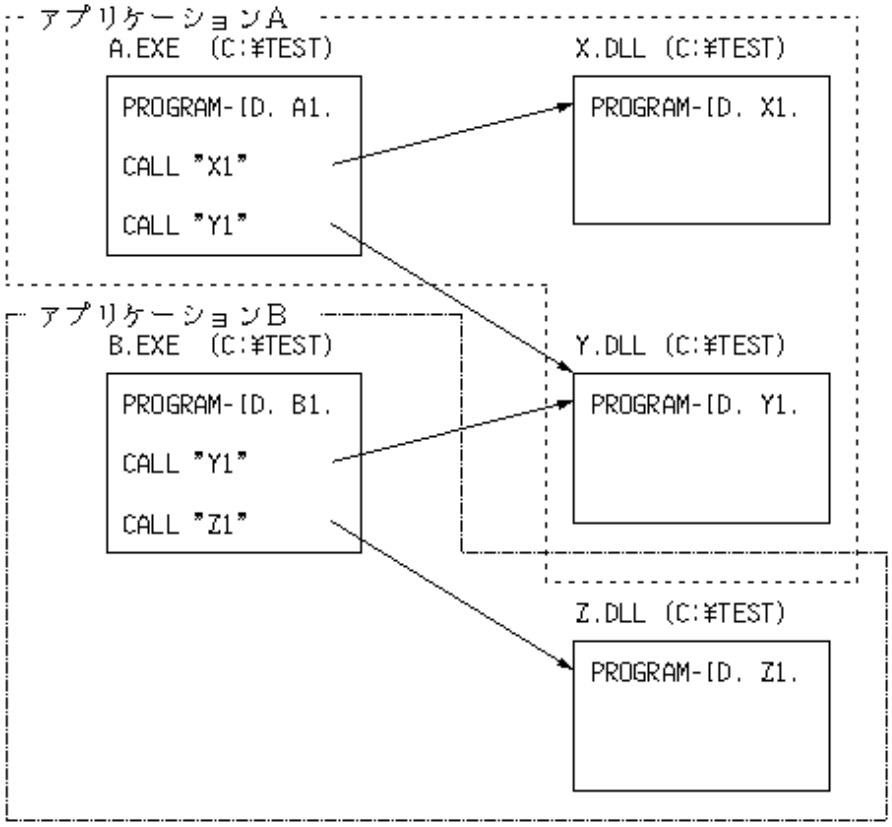
(2) 実行用の初期化ファイルに指定する場合

```
[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。

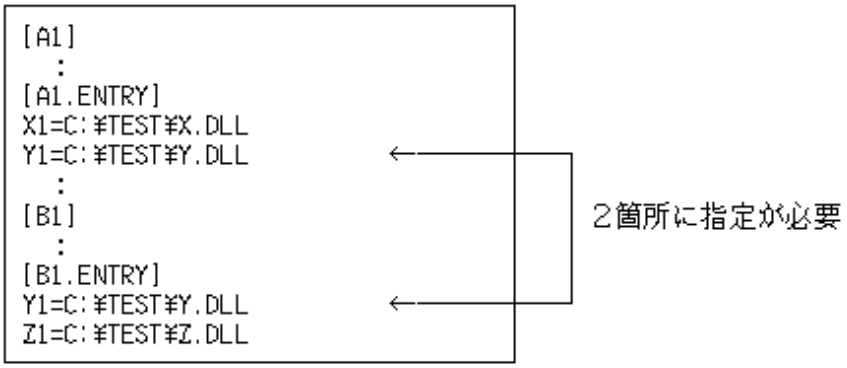


エントリ情報ファイル (C:¥TEST¥FILE.ENT)

```
[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）



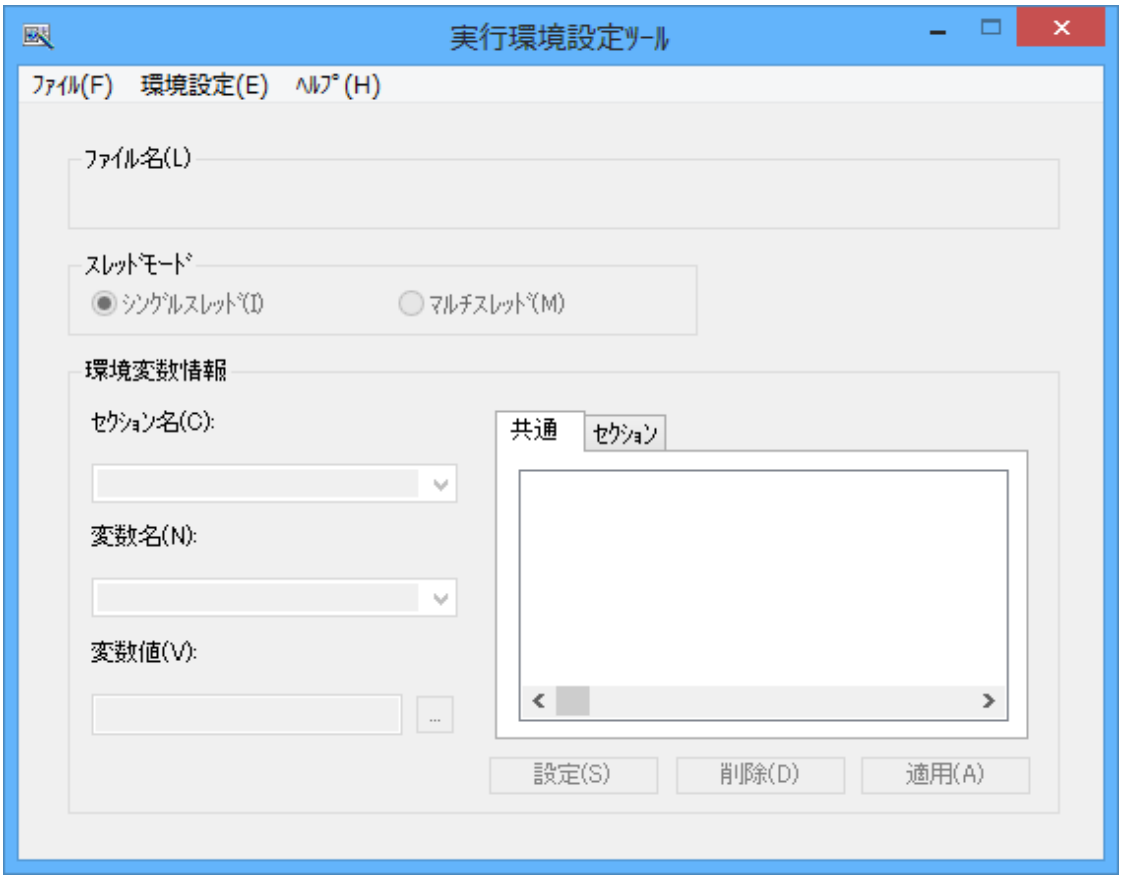
アプリケーション単位でセクションを指定し、セクションの下にそのアプリケーションに対するエントリ情報を指定します。つまり、複数のアプリケーションで共通で使用するDLLは、それぞれのセクションの下で定義します。

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るため、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

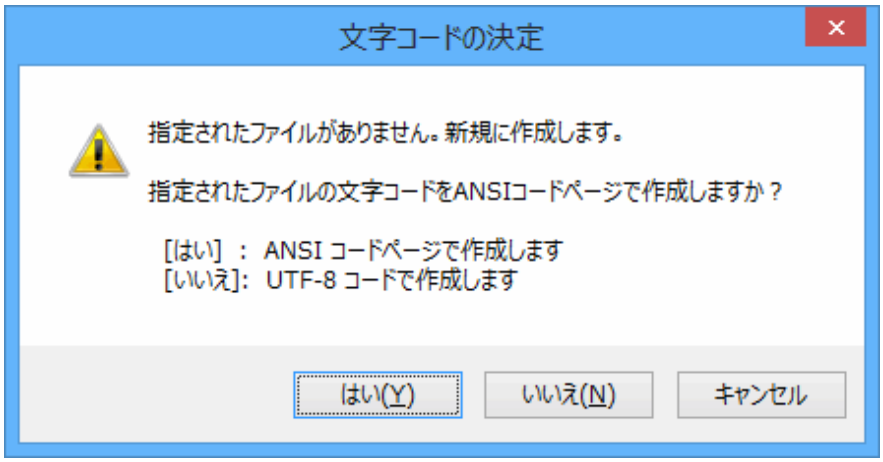
環境変数情報の設定は、実行環境設定ツールのセクション名、変数名、変数値のエディットボックスを使用して行います。

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オープンします。
   BOM付きUTF-8のファイルをオープンした時は、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.2.25 @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

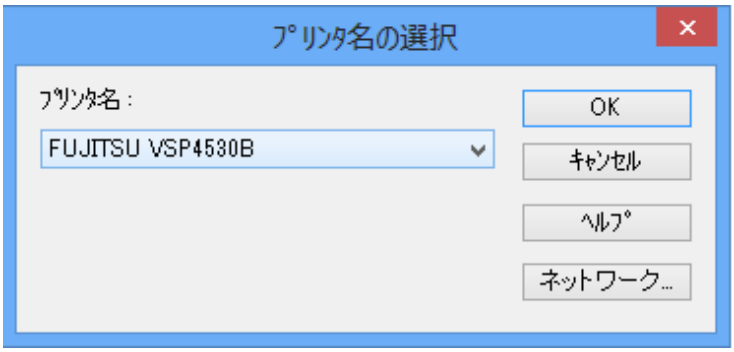
〔適用〕ボタンをクリックすると、リストボックス内の情報が実行用の初期化ファイルに保存されます。

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個数 | nor]
エラー検出時の処理実行回数の指定、およびCHECK機能の抑制指定	[c回数 | {noc | nocb | noci | nocn | nocp}]
外部スイッチの値の指定	[s値]
PowerSORTが使用するメモリ容量の最大値を指定	[smsize値k]

それぞれのオプションについて説明します。

[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.1.8 -Dr（TRACE機能を使用する指定）"

"A.3.54 TRACE（TRACE機能の使用の可否）"

[c回数 | {noc | nocb | noci | 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、以下のとお通りです。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)
- 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.1.5 -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"オンラインマニュアル"をお読みください。

このオプションは、翻訳オプションSMSIZE()および特殊レジスタSORT-CORE-SIZEに指定する値の意味と等価ですが、同時に指定された場合の優先度は、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(4M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”

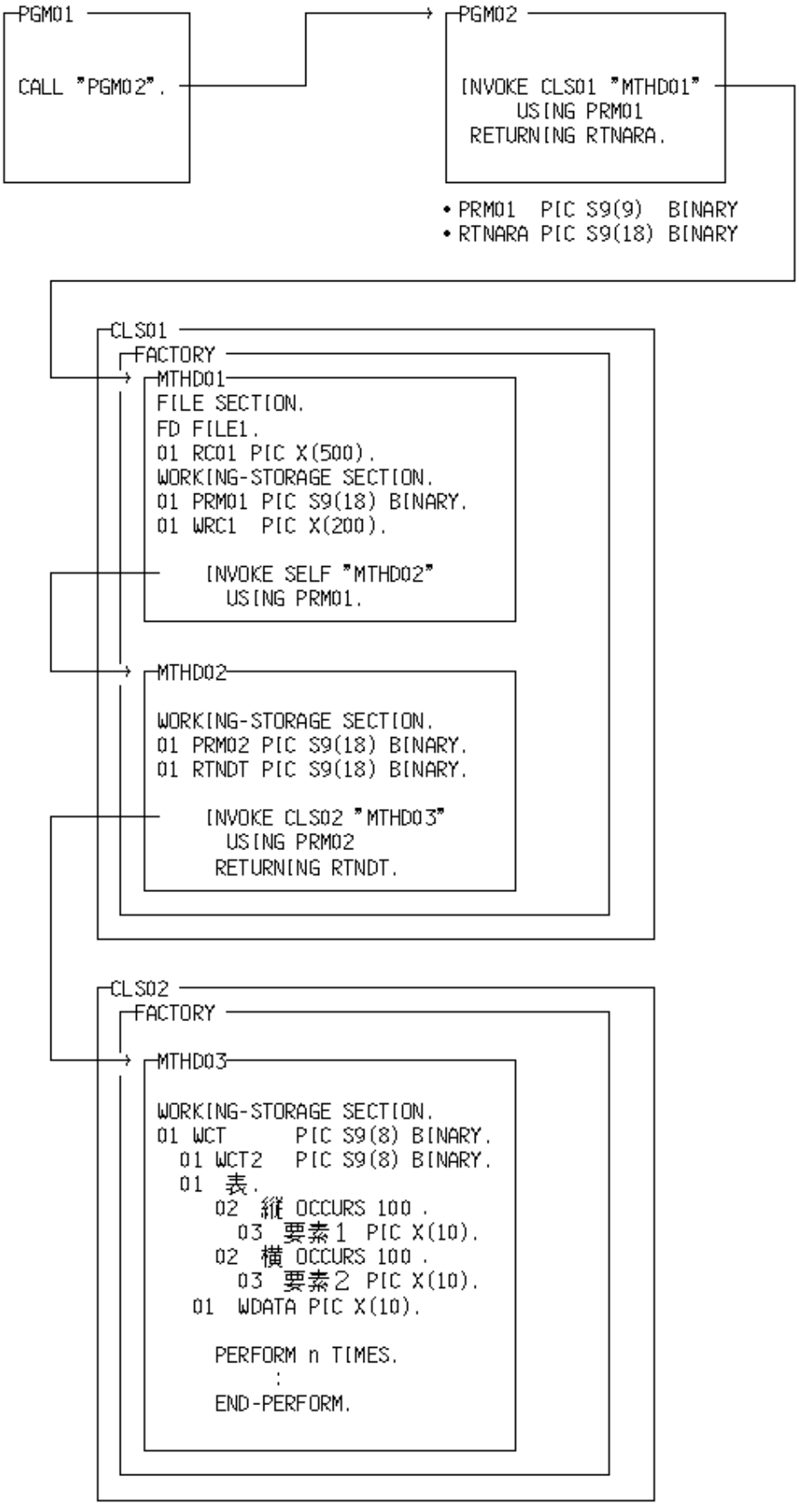
使用するスタックサイズの求め方

トータルスタックサイズ　≒　（α1 ＋α2 … ＋αn）＋（β1 ＋β2 …　＋　βm）

αn　：　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、…）
βm　：　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。

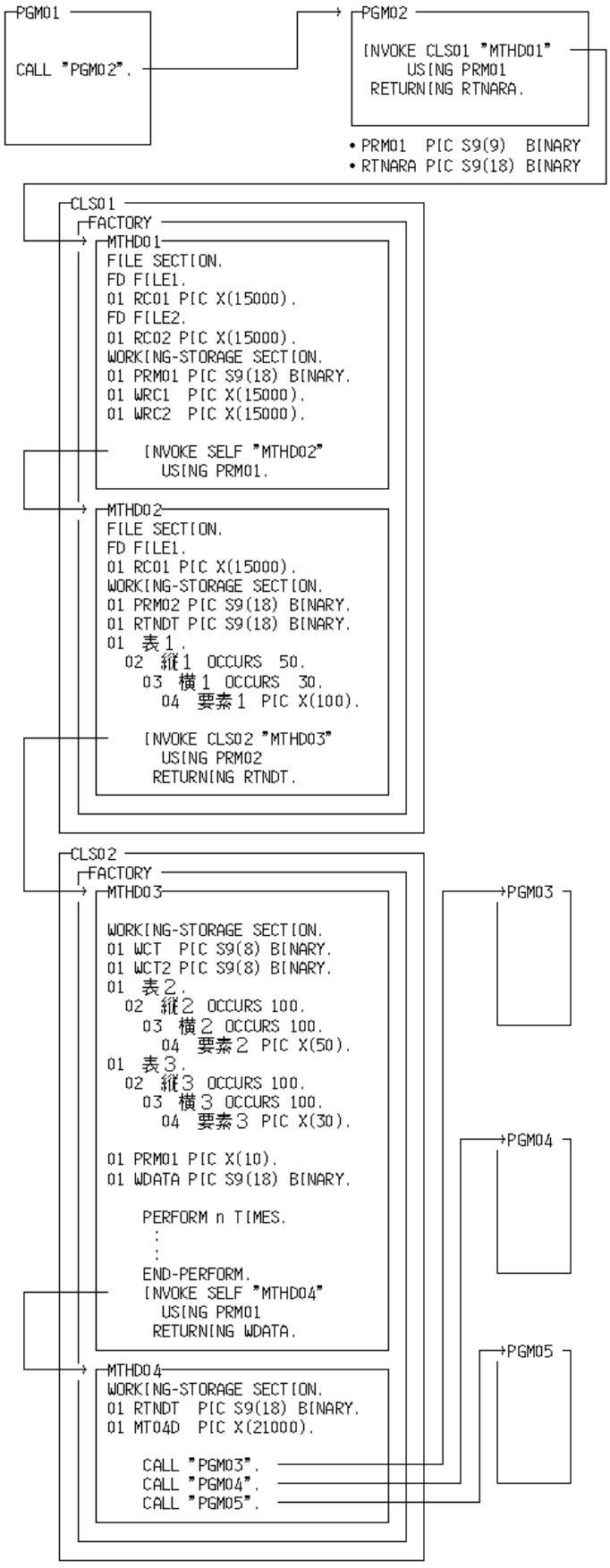
（例題1）オーバフローしないケース



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
```

スタックのオーバフローを回避する方法

オーバフローを回避するためには、リンク時にスタックサイズを拡張する/STACKオプションを指定します。

```
/STACK: reserve [, commit]
```

reserve

仮想メモリ内に確保するスタックの総サイズを指定します。

デフォルト値は4M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い。
- 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)の数が多いなど。

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、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”

“19.2 CHECK機能”

“19.5 メモリチェック機能”

5.9.3 32ビットWindowsで動作するアプリケーションの呼び出しについて

本製品で作成したアプリケーションから、32ビットWindowsで動作するアプリケーションを、動的プログラム構造で呼び出すことはできません。

同様に、32ビットアプリケーションで動作するアプリケーションから、本製品で作成したアプリケーションを呼び出すことはできません。

呼び出した場合、JMP0015I-U(詳細コード 0xc1)の実行時エラーメッセージを出力します。

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

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。

ー 文字種の切り換えにシフトコードを使用しません。

ー 1バイトで表現される英数字・カナについてはJIS8コード系と同じ値を持ちます。

ー JIS漢字は4つの領域に分散しますが、演算により規則的に対応しますし、文字のコード値の大小関係もJIS漢字と一致します。

なお、シフトJISにはJIS漢字に含まれない文字を追加するための領域があり、その領域に追加した文字の違いにより、いくつかの変種があります。例えば富士通により78JIS固有の文字やOASYS記号等を追加したもの(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。

ー UTF-8、UTF-16、UTF-32など、複数のエンコードを持っています。

ー 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。

ー 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。

ー 文字種の切り換えにシフトコードを使用します。

ー 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。

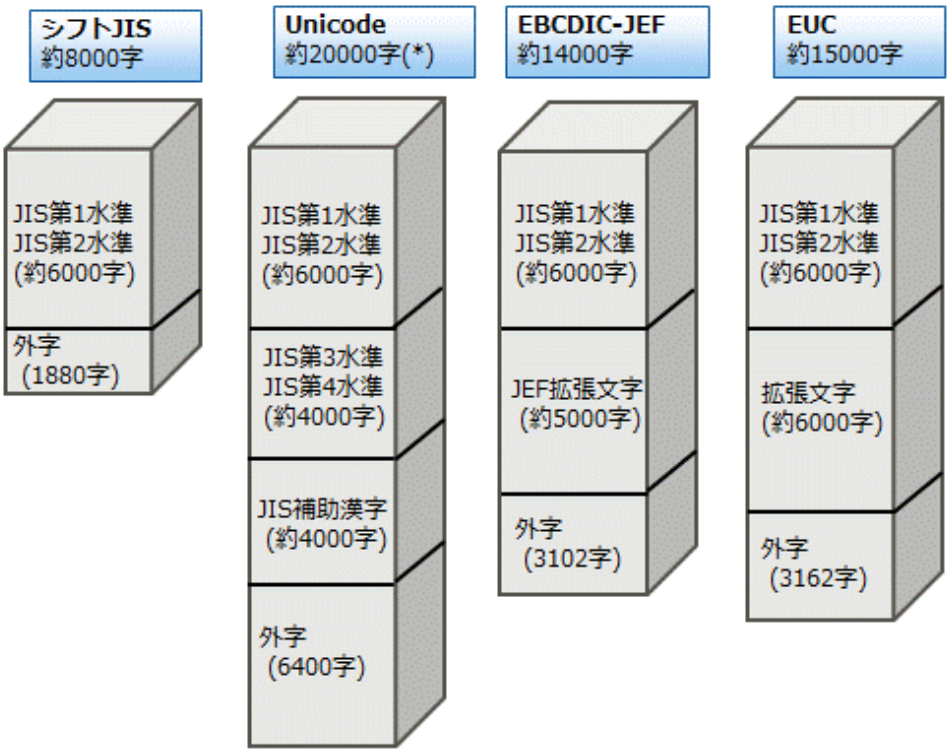
ー 文字は1～3バイトで表現されるが、1バイト目で文字種の判定が可能です。

ー 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。

ー JIS漢字のすべてが1つの領域に含まれます。

ー 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		EBCIDIC+JEF	EBCIDIC+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(x64)	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(x64)		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。

注意

Windows 8およびWindows Server 2012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、

```
基本項目のENCODING指定 > 集団項目のENCODING指定 > ファイル記述項のENCODING指定
```

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.3.13 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

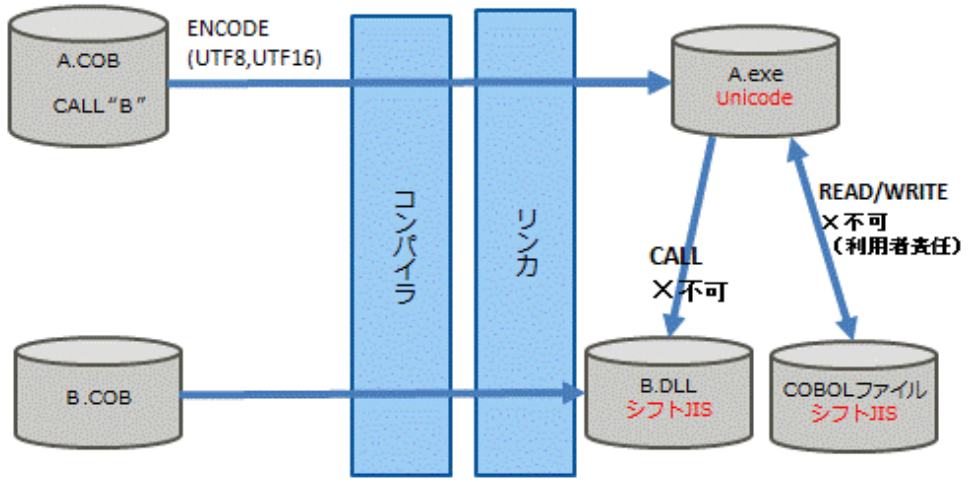
実行時コード系とは、NetCOBOLが作成した目的プログラムの属性で、“6.2.5 資源”で説明する実行時資源のコード系を決定する要因になります。実行時コード系は、翻訳オプションENCODEの指定に応じて決まります。

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	シフトJIS	シフトJIS	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。

なお、NetCOBOLでは実行単位内で実行時コード系は混在できません。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
シフトJIS	SCS(SJIS)または省略
Unicode(UTF-8)	SCS(UTF8)

翻訳オプションSCSの指定方法は、"A.3.41 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。

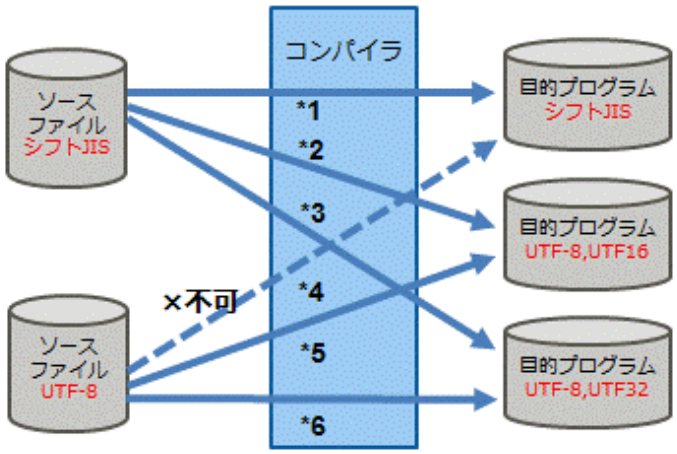
翻訳オプションを省略した場合は、SCS(SJIS)、ENCODE(SJIS,SJIS)が指定されたとみなされます。

	ENCODE(SJIS,SJIS)	ENCODE(UTF-8,UTF16)	ENCODE(UTF-8,UTF32)
SCS(SJIS)	○(省略時)	○	○

	ENCODE(SJIS,SJIS)	ENCODE(UTF-8,UTF16)	ENCODE(UTF-8,UTF32)
SCS(UTF8)	×	○	○

○: 組合せ可能

×: 組合せ不可。



*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。

実行時コード系がシフトJISの場合

資源のコード系はすべてシフトJISです。

	資源	入出力	コード系
翻訳時	翻訳リスト	出力	シフトJIS
リンク時	モジュール定義ファイル	入力	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	シフトJIS
	キー定義ファイル	入力	シフトJIS
	COBOLファイル(順/行順/相対/索引)	入出力	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	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	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	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	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	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) (注5)
	TRACE情報ファイル 汎用ログファイル	出力	
	メッセージを出力するファイル COUNT情報ファイル	出力	シフトJIS または Unicode(UTF-8) (注4)(注5)

注1:“A.3.41 SCS(ソースファイルのコード系)”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変換することができます。[参照]“C.2.6 @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L 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 代替文字

NetCOBOLでは、エンコードの変換処理において、変換元のデータ中に変換先のエンコードに対応する文字が存在しない場合、当該コードの変換結果を代替文字に置き換えます。代替文字は、REPLACNG CHARACTER句を使用して任意の代替文字を指定できます。

なお、REPLACING CHARACTER句が省略された場合、アンダースコアが指定されたものとみなします。

6.3.4 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 実行時の注意点

6.4.1 スクリーン機能

スクリーン操作機能では、UTF-32エンコードのデータ項目は使用できません。

6.4.2 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ース名に、Unicodeに対応したフォントを指定してください。

指定方法は、“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のオンラインマニュアル”および、“8.1.13 フォントテーブル”を参照してください。

TRACE機能、COUNT機能が出力するファイル

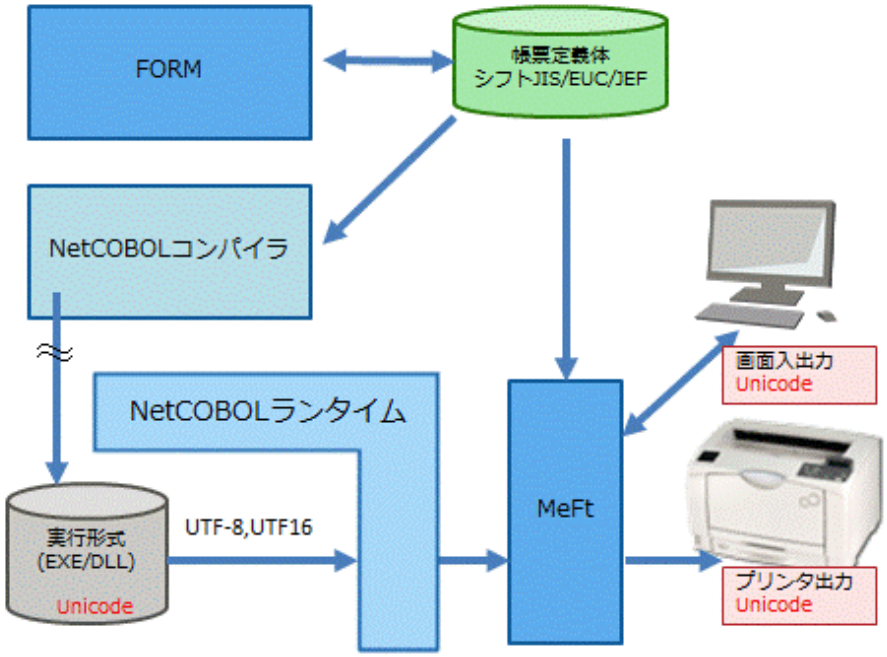
動作モードがUnicodeの場合、TRACE機能およびCOUNT機能が出力するファイルのコード系は、以下になります。

- 既存ファイルに追加書きする場合は、既存ファイルのコード系になります(COUNT機能のみ)。
- 新規ファイルに出力する場合は、UTF-8になります。

6.5 関連製品連携

6.5.1 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。

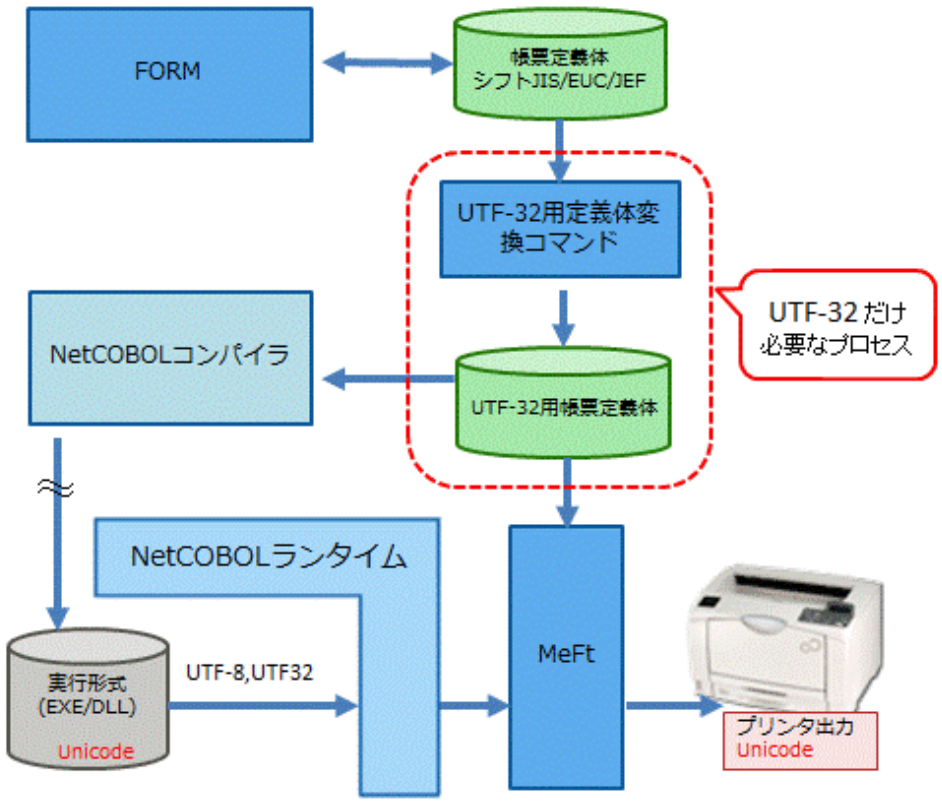
エンコードUTF-16のサロゲートペアについて

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のマニュアルをご確認ください。

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

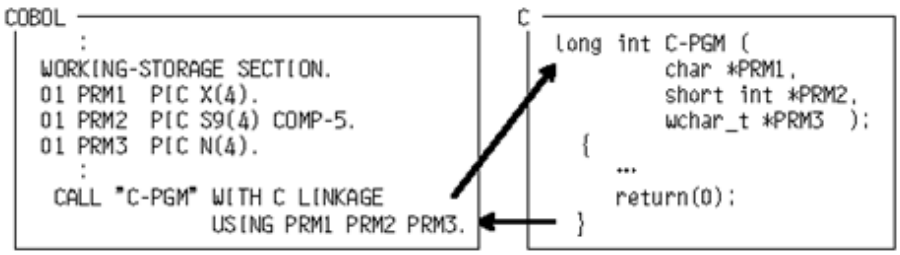
画面定義体は、エンコードUTF-32の場合には使用できません。

6.5.2 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型との間でのデータ送受もできます。

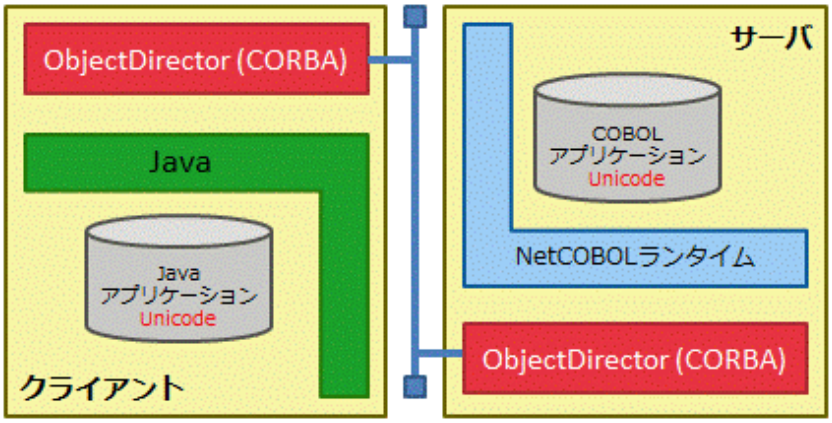


Windowsシステム関数の呼出しもできます。WindowsはUnicodeで動作するシステム関数(関数名の末尾がW)を用意しています。この関数との間で文字データを送受する場合は、日本語項目を使用してください。

```
      :
    WORKING-STORAGE SECTION.
    01 M-TITLE   PIC N(20).
    01 M-MSG     PIC N(20).
    01 RET-CODE  PIC S9(9) COMP-5.
         :
   ** パラメタの設定
        MOVE NC"COBOL Unicodeアプリ" & NX"0000" TO M-TITLE.
        MOVE NC"メッセージの表示"    & NX"0000" TO M-MSG.
   **
   ** システム関数の呼出し
        CALL "MessageBoxW" WITH STDCALL USING BY VALUE     0
                                              BY REFERENCE M-MSG
                                              BY REFERENCE M-TITLE
                                              BY VALUE     1
                                     RETURNING RET-CODE.
```

Java

Javaと連携する場合、一般的にはCORBAを利用しますが、Jアダプタクラスを利用してCOBOLから直接Javaのクラスを呼び出すこともできます。いずれの場合もUTF-8またはUTF-16を使用してください。



6.5.3 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。

6.5.4 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。

6.5.5 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)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。

6.5.6 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を指定できません。

6.5.7 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

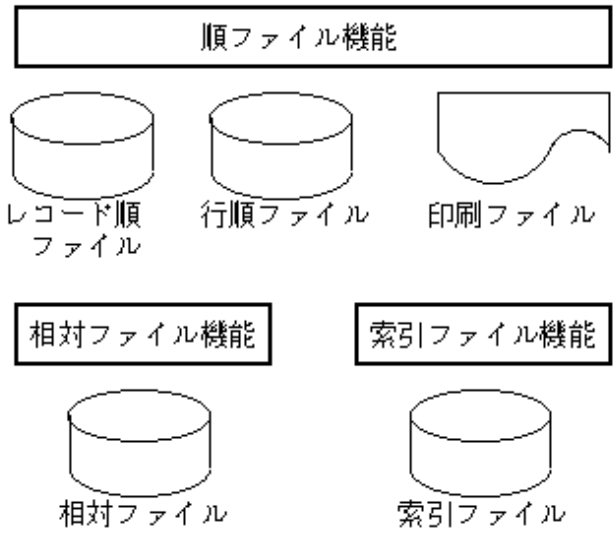
本章では、ファイルからデータを読み込んだり、ファイルにデータを書き出したりする処理、COBOLファイルユーティリティの使い方について説明します。

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、以下のファイルを処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

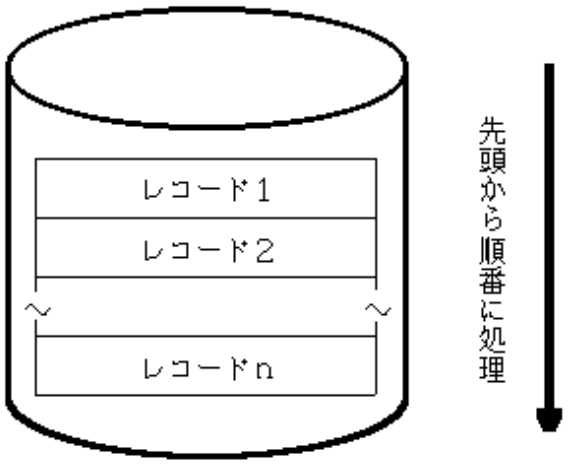
	ファイルの種類				
	レコード順ファイル	行順ファイル	印刷ファイル	相対ファイル	索引ファイル
レコードの処理	レコードが格納されている順番			相対レコード番号	レコードキーの値
使用できる媒体	ハードディスク(*) フロッピーディスク	ハードディスク(*) フロッピーディスク	印刷装置	ハードディスク(*) フロッピーディスク	ハードディスク(*) フロッピーディスク
利用例	データ退避 作業ファイル	テキストファイル	データの印刷	作業ファイル	マスタファイル

＊ ： 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。

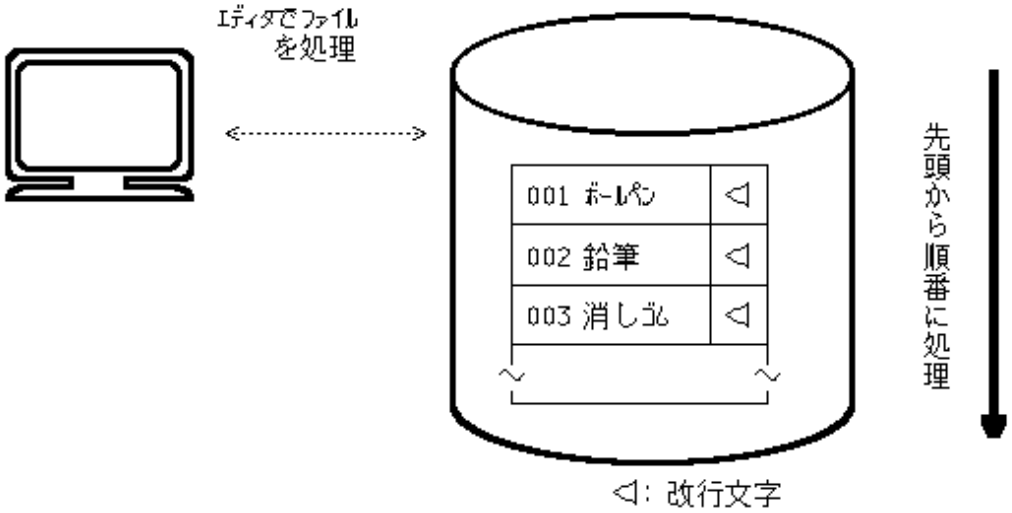
レコード順ファイル

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を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。

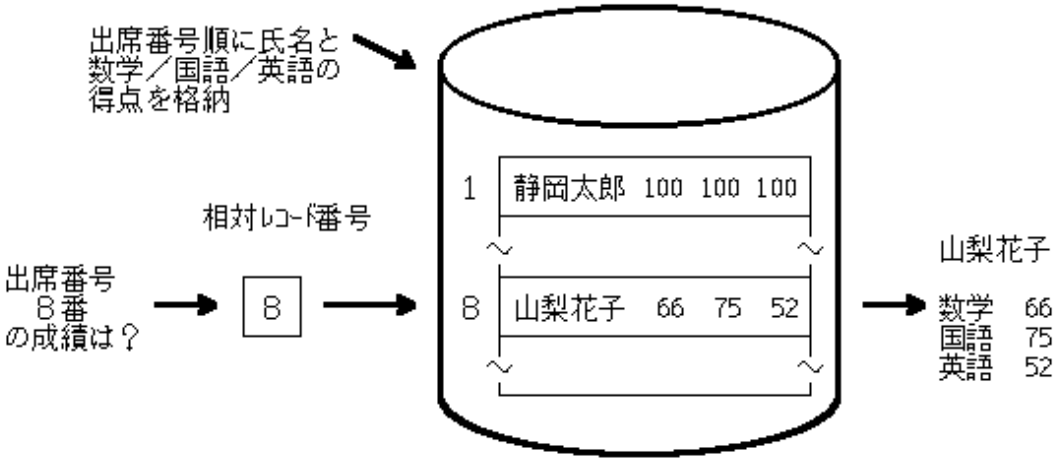
・ 行順ファイルの文字コードは、ひとつの表現形式で統一しなければなりません。

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。

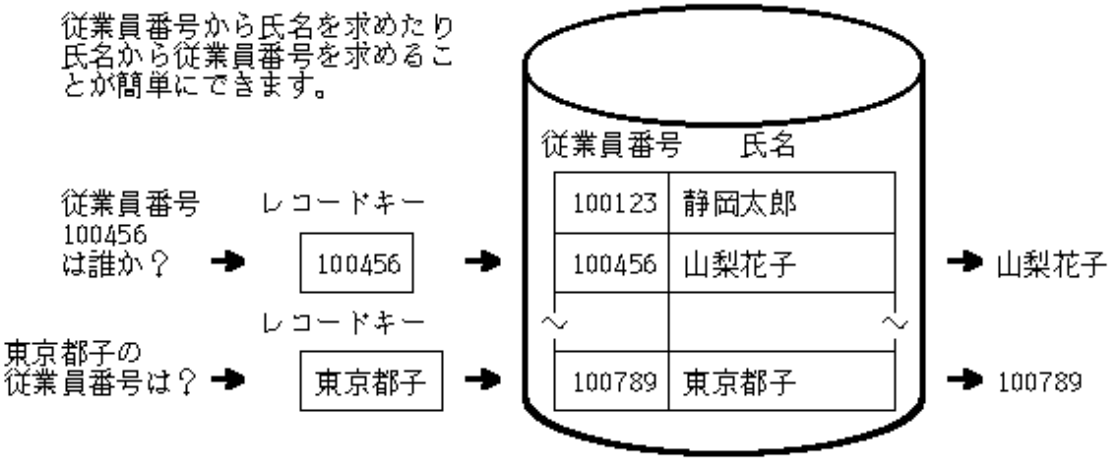
相対ファイル

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なります。可変長レコード形式は、必要な大きさで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)出力先がファイル以外の場合は、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を関連付ける方法が異なります。ファイル参照子に何を指定するかは、実際の入出力媒体のファイル名がいつ決まるかにより、以下のように決定することをおすすめします。

- COBOLプログラム作成時に実際の入出力媒体のファイル名が決定し、その後変更されない場合には、ファイル識別名定数または文字列DISKを指定します。
- COBOLプログラム作成時に実際の入出力媒体のファイル名が決定しなかったり、毎回のプログラム実行時にファイル名を決定したい場合には、ファイル識別名を指定します。
- プログラムの中でファイル名を決定したい場合には、データ名を指定します。
- プログラムの終了時には必要のない、一時的なファイルである場合には、文字列DISKを指定します。

注意

文字列DISKを指定した場合、プログラムの終了時に、使用したファイルが削除されるわけではありません。

また、実際の入出力媒体のファイル名には、以下の文字を含むことができます。

空白、「+」、「,」、「;」、「=」、「[」、「]」、「(」「)」、「'」

なお、コンマ(,)を含む場合には、ファイル名を二重引用符(")で囲む必要があります。

参考

レコード順ファイルおよび行順ファイルでは、ファイルを高速に処理することができます。指定方法については、"7.7.4 ファイルの高速処理"を参照してください。

SELECT句およびASSIGN句の記述例を"表7.3 SELECT句およびASSIGN句の記述例"に示します。

表7.3 SELECT句およびASSIGN句の記述例

ファイル参照子の種類	記述例	備考
ファイル識別名	SELECT ファイル１ ASSIGN TO INFILE	プログラム実行時に実際の入出力媒体と結び付ける必要があります。
データ名	SELECT ファイル２ ASSIGN TO データ名	データ名はデータ部の作業場所節で定義する必要があります。
ファイル識別名定数	SELECT ファイル３ ASSIGN TO "C.DAT"	
文字列DISK	SELECT DATA1 ASSIGN TO DISK	ファイル名を絶対パス名で指定することはできません。

ファイル編成

ORGANIZATION句にSEQUENTIALを指定します。なお、ORGANIZATION句を省略した場合には、SEQUENTIALを指定したものとみなされます。

7.2.2 レコード順ファイルのレコードの定義

ここでは、レコード形式とレコード長およびレコードの構成について説明します。

レコード形式とレコード長

レコード順ファイルのレコード形式には、固定長レコード形式と可変長レコード形式があります。固定長レコード形式のレコード長は、RECORD句に指定した値、またはRECORD句が省略された場合はレコード記述項の最大値となります。可変長レコード形式では、レコードを書き出したときのレコードの長さが、そのレコードのレコード長になります。書き出すレコードの長さは、RECORD句の“DEPENDING ON データ名”に記述したデータ名に設定することができます。また利用者は、このデータ名を使って、レコードの入力時にレコード長を得ることもできます。

レコードの構成

レコード中のデータの属性、位置および大きさは、レコード記述項で定義します。

例

固定長レコード形式のレコードの定義例

データ1 (100バイト)	データ2 (100バイト)

```
FD  データ保存用ファイル.
01  データレコード.
    02  データ１  PIC X(100).
    02  データ２  PIC X(100).
```

可変長レコード形式のレコードの定義例

データ1 (100バイト)	データ2 (1～100バイト)

```
FD  データ保存用ファイル
    RECORD IS VARYING IN SIZE FROM 101 TO 200 CHARACTERS
                        DEPENDING ON レコード長.
01  データレコード.
    02  データ１  PIC X(100).
    02  データ２.
        03  PIC X OCCURS 1 TO 100 TIMES DEPENDING ON  長さ.
      :
WORKING-STORAGE SECTION.
01  レコード長     PIC 9(3) BINARY.
01  長さ           PIC 9(3) BINARY.
  :
```

注意

OCCURS句を指定した可変長データ項目を含む複数のデータ項目から構成される可変長レコード形式の場合、以下の注意が必要です。

レコードの書出し時

レコード項目にデータを転記するとき、最初の可変長データ項目から順にデータおよびデータ長を転記しなければなりません。データを転記する順番によっては、意図しない転記結果となる場合があります。

レコードの読込み時

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.
```

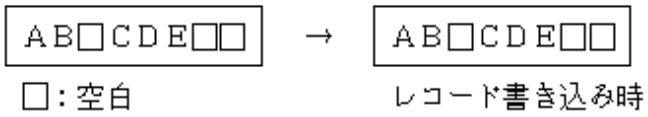
注意

－ 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。

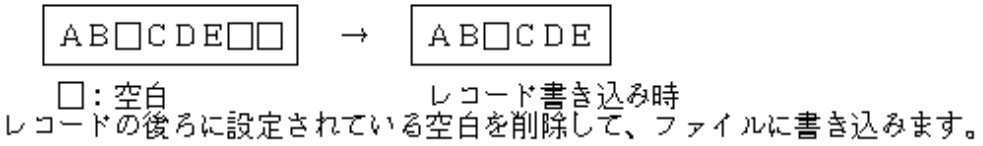
ー レコードの書出しでは、レコードの領域の内容と改行文字が書き出されます。

ー レコード内の後置空白を取り除いてレコードを書き出す場合は、環境変数情報@CBR_TRAILING_BLANK_RECORDに文字列“REMOVE”を指定します。[参照]“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

ー ファイル管理記述項のSELECT句にOPTIONALを指定した場合、OPEN文実行時にファイルが存在していなくても、OPEN文は成功となり、最初のREAD文の実行でファイル終了条件が成立します。ファイル終了条件については、“7.6 入出力エラー処理”を参照してください。

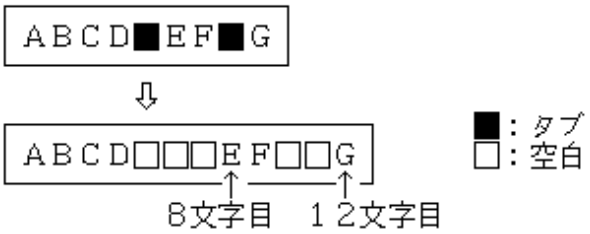
ー 読み込んだレコードの大きさがレコード長より大きいときには、1回のREAD文の実行でレコード長と同じ長さのデータがレコード領域に設定されます。次のREAD文の実行では、同じレコードのデータの続きから、データがレコード領域に設定されます。設定するデータがレコード長より小さい場合には、レコード領域の残りの領域に、空白が設定されます。

レコード内のタブの扱い

読み込んだレコードにタブが存在した場合、以下のように空白が設定されます。

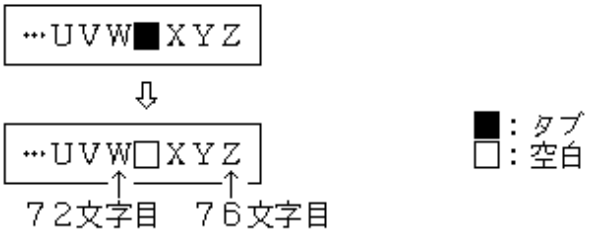
72文字以内にタブが存在する場合

先頭の文字位置を1として、8、12、16、20、24、28、32、36、40、44、48、52、56、60、64、68、72の文字位置にタブの次の文字が配置されるように、空白が設定されます。



72文字を超えた位置にタブが存在する場合

1文字の空白が設定されます。



レコード内の制御文字の扱い

読み込むレコードに制御文字が含まれている場合の動作は、以下のとおりです。

制御文字	動作
0x0C(改頁)	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。

ADVANCING指定		出力されるデータ
なし		{レコードデータ}0x0D 0x0A
BEFORE	n LINES (*1)	{レコードデータ}0x0D 0x0A … 0x0A (n個)
	PAGE	{レコードデータ}0x0D 0x0C
AFTER	n LINES (*1)	0x0A … 0x0A {レコードデータ}0x0D (n個)
	PAGE (*2)	0x0C {レコードデータ}0x0D

*1:行数に0を指定した場合、レコードの後ろに0x0D(復帰)のみを付加したレコードが出力されます。0x0A(改行)は出力されません。

*2:OPEN OUTPUT直後のWRITE文では、0x0C(改頁)は出力されません。

注意

ADVANCING指定のWRITE文で書き出されたレコードを読み込む場合、特に前後のレコードの制御文字が連続している場合は、意図したレコード区切りで読み込まれないことがあります。

- 以下の連続する制御文字は、1つのレコード区切り文字として扱います。
  - 0x0D(復帰)に続き、0x0A(改行)や0x0C(改頁)が存在する
  - 0x0A(改行)や0x0C(改頁)に続き、0x0D(復帰)が存在する
- 0x0A(改行)や0x0C(改頁)が連続する場合、1つの制御文字として扱います。
- レコードの先頭にある0x0A(改行)や0x0C(改頁)は、読み飛ばされます。

制御文字が連続しない場合の動作については、“レコード内の制御文字の扱い”を参照してください。

Unicodeの行順ファイルを参照する場合

Unicodeアプリケーションにおいて行順ファイルを参照する場合、ファイルの先頭に付加されているBOM(Byte Order Mark)と呼ばれる識別コードの扱いを選択することができます。

詳細は、“C.2.28 @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)”を参照してください。

7.4 相対ファイルの使い方

ここでは、相対ファイルについて以下の項目を説明します。

- ファイルの定義方法
- レコードの定義方法
- ファイルの処理方法

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.   プログラム名.
 ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
   FILE-CONTROL.
```

```
    SELECT  ファイル名
      ASSIGN TO  ファイル参照子
      ORGANIZATION IS RELATIVE
     [ACCESS MODE  IS  アクセス形態]
     [RELATIVE KEY IS  相対レコード番号格納域].
DATA DIVISION.
 FILE SECTION.
 FD  ファイル名
    [RECORD  レコードの大きさ].
 01  レコード名.
       レコード記述項
 WORKING-STORAGE SECTION.
[01  相対レコード番号格納域     PIC 9(5) BINARY.]
PROCEDURE DIVISION.
    OPEN   オープンモード  ファイル名.
   [MOVE   相対レコード番号  TO  相対レコード番号格納域.]
   [READ     ファイル名.]
   [REWRITE  レコード名.]
   [DELETE   ファイル名.]
   [START    ファイル名.]
   [WRITE    レコード名.]
    CLOSE    ファイル名.
END PROGRAM  プログラム名.
```

7.4.1 相対ファイルの定義

ここでは、COBOLプログラムで相対ファイルを使うときに必要なファイルの定義について説明します。

ファイル名とファイル参照子

相対ファイルでは、レコード順ファイルと同様に、ファイル名とファイル参照子を指定します。指定方法については、“7.2.1 レコード順ファイルの定義”を参照してください。

ファイル編成

ORGANIZATION句にRELATIVEを指定します。

アクセス形態

ACCESS MODE句に以下のアクセス形態のどれかを指定します。

順呼出し法(SEQUENTIAL)

ファイルの先頭またはある相対レコード番号のレコードから、相対レコード番号の昇順にレコードを処理することができる、順呼出しの処理を行うことができます。

乱呼出し法(RANDOM)

ある相対レコード番号のレコードだけを単独に処理することができる、乱呼出しの処理を行うことができます。

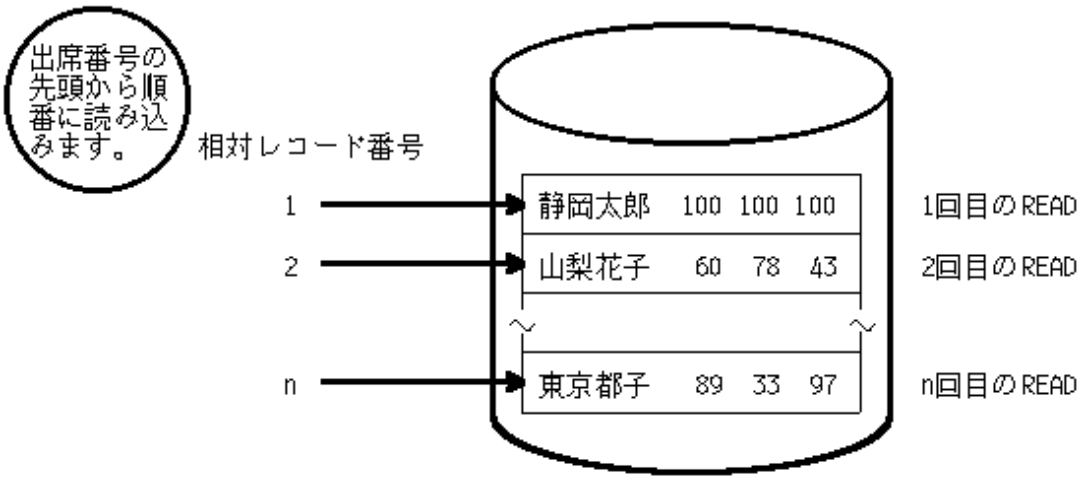
動的呼出し法(DYNAMIC)

順呼出しと乱呼出しの両方の処理を行うことができます。

以下に、レコードの参照処理を例に、順呼出しの場合と乱呼出しの場合での処理の違いを示します。

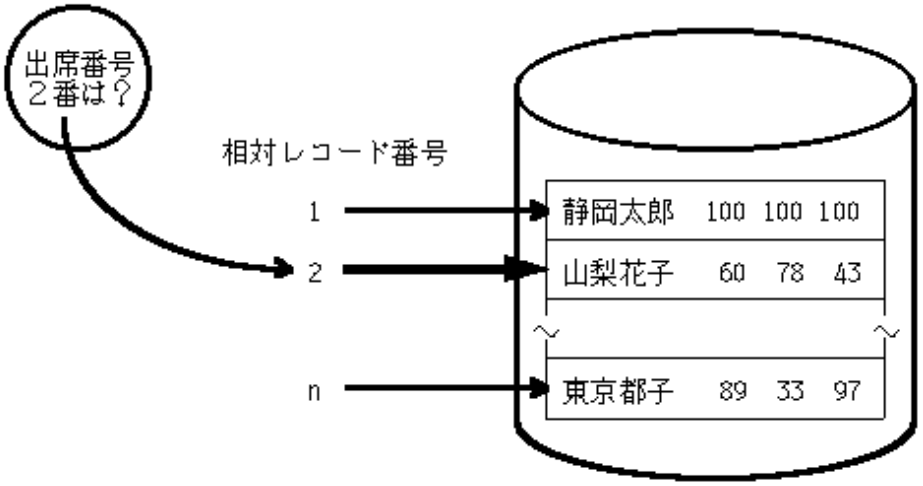
順呼出し

出席番号順に格納されているレコードを先頭から順番に処理する場合



乱呼出し

出席番号順に格納されているレコードの、ある出席番号の者のデータを処理する場合



相対レコード番号

相対レコード番号を設定するデータ名を、RELATIVE KEY句に指定します。ただし、順呼出しの場合は、この句を省略することができます。このデータ名には、レコードの入力時には入力したレコードの相対レコード番号が設定され、出力時には書き出すレコードの相対レコード番号を利用者が設定します。ただし、レコードの出力を順呼出しでアクセスする場合、利用者が設定した相対レコード番号は無視されます。なお、このデータ名は、符号なし整数項目として作業場所節に定義する必要があります。

7.4.2 相対ファイルのレコードの定義

ここでは、レコード形式とレコード長およびレコードの構成について説明します。

レコード形式とレコード長

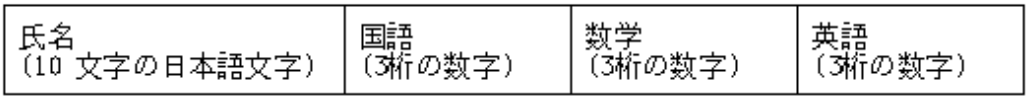
相対ファイルのレコード形式とレコード長の説明は、レコード順ファイルの説明と同じです。“7.2.2 レコード順ファイルのレコードの定義”を参照してください。

レコードの構成

レコード中のデータの属性、位置および大きさは、レコード記述項で定義します。なお、相対レコード番号を設定するための領域を定義する必要はありません。

例

固定長レコード形式のレコードの定義例




```
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。

参考

相対レコード番号は、RELATIVE KEY句に指定したデータ名に設定します。

```
例：MOVE  1  TO  データ名.
```

7.5 索引ファイルの使い方

ここでは、索引ファイルについて以下の項目を説明します。

- ファイルの定義方法
- レコードの定義方法
- ファイルの処理方法

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.   プログラム名.
 ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
   FILE-CONTROL.
     SELECT  ファイル名
       ASSIGN TO  ファイル参照子
       ORGANIZATION IS INDEXED
      [ACCESS MODE  IS  アクセス形態]
       RECORD KEY   IS  主キー名１[主キー名ｎ] … [WITH DUPLICATES]
     [[ALTERNATE RECORD KEY IS  副キー名１[副キー名ｎ] …
                                          [WITH DUPLICATES]] … ].
 DATA DIVISION.
  FILE SECTION.
  FD  ファイル名
     [RECORD  レコードの大きさ].
  01  レコード名.
      02  主キー名１  ～.
     [02  主キー名ｎ  ～.]
     [02  副キー名１  ～.]
     [02  副キー名ｎ  ～.]
     [02  キー以外のデータ  ～.]
 PROCEDURE DIVISION.
     OPEN   オープンモード  ファイル名.
    [MOVE     主キーの値 TO 主キー名ｎ.]
    [MOVE     副キーの値 TO 副キー名ｎ.]
    [READ     ファイル名.]
    [REWRITE  レコード名.]
    [DELETE   ファイル名.]
    [START    ファイル名.]
    [WRITE    レコード名.]
     CLOSE    ファイル名.
 END PROGRAM  プログラム名.
```

7.5.1 索引ファイルの定義

ここでは、COBOLプログラムで索引ファイルを使うときに必要なファイルの定義について説明します。

ファイル名とファイル参照子

索引ファイルでは、レコード順ファイルと同様に、ファイル名とファイル参照子を指定します。指定方法については、“7.2.1 レコード順ファイルの定義”を参照してください。

ファイル編成

ORGANIZATION句にINDEXEDを指定します。

アクセス形態

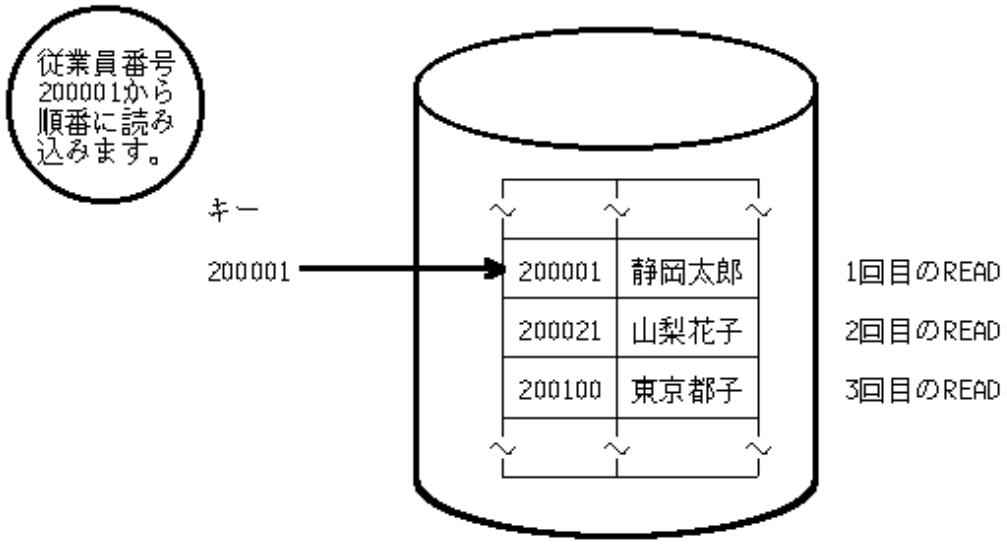
ACCESS MODE句に以下のアクセス形態のどれかを指定します。

- 順呼出し法(SEQUENTIAL): ファイルの先頭またはある特定の値のキーを持つレコードから、キーの昇順にレコードを処理することができる、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。

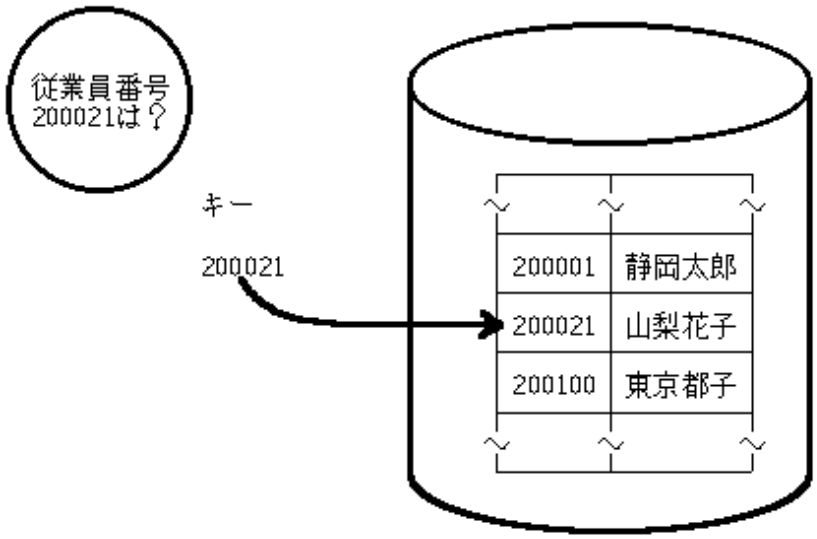
順呼出し

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。

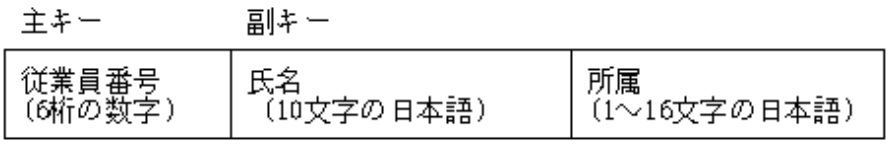
レコードの構成

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。

- 既存のファイルを処理する場合、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




```
       :
     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.
```

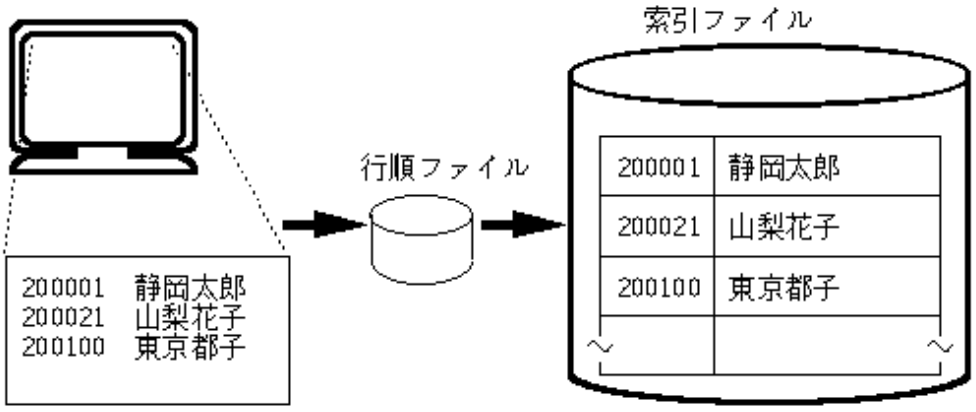
注意

- 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。

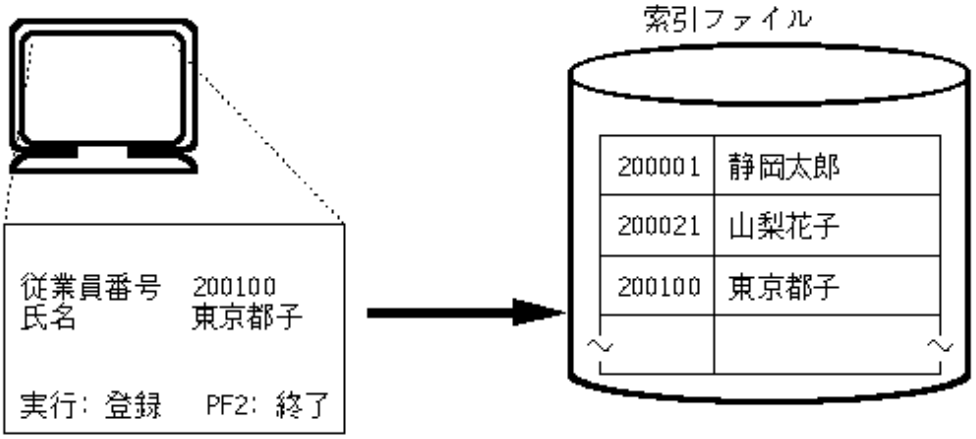
参考

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(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すが、主キーの内容を変更することはできません。

削除〔順/乱/動的〕

レコードの削除では、ファイルを入出力モードで開いて、ファイルのレコードを削除します。順呼出しの場合、直前のREAD文により読み込まれたレコードが削除されます。乱呼出しの場合、設定したキー値を持つ主キーのレコードが削除されます。

順呼出し

```
OPEN I-O  ファイル名.
MOVE  キーの値 TO キー名.
START  ファイル名.
READ  ファイル名 [NEXT] ～.
DELETE ファイル名  ～.
CLOSE  ファイル名.
```

乱呼出し

```
OPEN I-O  ファイル名.
MOVE  キーの値 TO キー名.
DELETE ファイル名  ～.
CLOSE  ファイル名.
```

注意

乱呼出し法または動的呼出し法で、指定したキーの値を持つレコードが存在しないときに無効キー条件が成立します。無効キー条件については、“7.6 入出力エラー処理”を参照してください。

挿入〔乱/動的〕

レコードの挿入では、ファイルを入出力モードで開いて、ファイルにレコードを挿入します。レコードの挿入される位置は、主キーの値によって決定されます。

```
OPEN I-O  ファイル名.
  レコードの編集処理
MOVE  キーの値 TO キー名.
WRITE  レコード名  ～.
CLOSE  ファイル名.
```

注意

ファイル管理記述項のRECORD KEY句またはALTERNATE RECORD KEY句にDUPLICATES指定がない場合、指定したキーの値を持つレコードがすでに存在するとき、無効キー条件が成立します。無効キー条件については、“7.6 入出力エラー処理”を参照してください。

7.6 入出力エラー処理

ここでは、入出力エラーの検出方法および入出力エラーが発生したときの実行結果について説明します。入出力エラーの検出方法には、次の4つがあります。

- AT END指定(ファイル終了条件発生の検出)
- INVALID KEY指定(無効キー条件発生の検出)
- FILE STATUS句(入出力状態のチェックによる入出力エラーの検出)
- 誤り処理手続き(入出力エラーの検出)

7.6.1 AT END指定

ファイル中のレコードを順番に読み、すべてのレコードを読み終わったときに、次に読み込むレコードが存在しないとファイル終了状態になります。これを、ファイル終了条件の発生といいます。ファイル終了条件の発生を検出するには、READ文にAT END指定を記述します。AT END指定には、ファイル終了条件が発生したときに行う処理を記述することができます。

例

AT END指定のREAD文の記述例

```
READ   順ファイル AT END
                  GO TO  ファイルの終了処理
END-READ.
```

ファイルの最後のレコードを読んだ次のREAD文の実行で、ファイル終了条件が発生し、GO TO文が実行されます。

7.6.2 INVALID KEY指定

索引ファイルまたは相対ファイルの入出力処理で、指定したキーや相対レコード番号を持つレコードが存在しなかった場合、入出力エラーとなります。これを、無効キー条件の発生といいます。無効キー条件の発生を検出するには、READ文、WRITE文、REWRITE文、START文およびDELETE文にINVALID KEY指定を記述します。INVALID KEY指定には、無効キー条件が発生したときに行う処理を記述することができます。

例

INVALID KEY指定のREAD文の記述例

```
MOVE  "Z" TO   主キー.
READ   索引ファイル INVALID KEY
                  GO TO  無効キーの処理
END-READ.
```

主キーの値に“Z”を持つレコードが存在しないとき、無効キー条件が発生し、GO TO文が実行されます。

7.6.3 FILE STATUS句

ファイル管理記述項にFILE STATUS句を記述すると、入出力文実行時に、FILE STATUS句に指定したデータ名に入出力状態が通知されます。入出力文の後に、このデータ名の内容(入出力状態値)をチェックする文(IF文またはEVALUATE文)を記述することによって、プログラムで入出力文の結果に応じた処理手続きを実行することができます。ただし、入出力文の後に入出力状態値をチェックしない場合、入出力エラーが発生してもプログラムの後続処理の実行が続けられる場合があるため、その後の動作は保証されません。

通知される入出力状態値については、“付録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ので、オープンに失敗した場合にはGO TO文が実行されます。

7.6.4 誤り処理手続き

手続き部の宣言節部分で、USE AFTER ERROR/EXCEPTION文を記述することにより、誤り処理手続きを指定することができます。誤り処理手続きを記述すると、入出力エラー発生時に誤り処理手続きに記述した処理が実行されます。誤り処理を実行後、入出力エラーが発生した入出力文の直後の文に制御が渡るので、入出力文の直後には、入出力エラーが発生したファイルの処理の制御を指示する文を記述する必要があります。ここでの入出力エラーとは、入出力状態の成功の分類に含まれるもの以外の条件が発生したことを示します。

以下の場合には、誤り処理手続きに制御は渡りません。

- ファイル終了条件が発生したREAD文にAT END指定がある場合
- 無効キー条件が発生した入出力文にINVALID KEY指定がある場合
- ファイルが開かれる前に入出力文を実行した場合(オープンモードの指定をした場合)

以下の場合、誤り処理手続き中から手続き中へGO TO文を使って分岐してください。

- 誤り処理手続きの中で、入出力エラーが発生したファイルに対して入出力文を実行した場合
- 誤り処理手続きが終了する前に、その誤り処理手続きが再び実行された場合

```
PROCEDURE DIVISION.
  DECLARATIVES.
    誤り処理 SECTION.
      USE AFTER ERROR PROCEDURE ON  ファイル.
        MOVE  エラー発生  TO  ファイルの状態.          … [1]
  END DECLARATIVES.
      :
      OPEN  INPUT  ファイル.
      IF   ファイルの状態 =  エラー発生                 … [2]
        THEN GO TO   オープン失敗の処理.
      :
```

ファイルのオープンに失敗すると、誤り処理手続き([1]のMOVE文)が実行され、OPEN文の直後の文([2]のIF文)に制御が渡ります。

7.6.5 入出力エラーが発生したときの実行結果

入出力エラーが発生したときの実行結果は、AT END指定の有無、INVALID KEY指定の有無、FILE STATUS句の有無および誤り処理手続きの有無によって異なります。入出力エラーが発生したときの実行結果を“表7.4 入出力エラーが発生したときの実行結果”に示します。ここでの入出力エラーとは、入出力状態の成功の分類に含まれるもの以外の条件が発生したことを示します。

表7.4 入出力エラーが発生したときの実行結果

誤り処理手続き	有	有	有	有	無	無	無	無
FILE STATUS 句	有	無	有	無	有	無	有	無
AT END指定 または INVALID KEY 指定	有	有	無	無	有	有	無	無
ファイル終了条件発生時 または 無効キー条件発生時の処理	1	1	2	2	1	1	3	5
その他の入出力エラー発生時の処理	2	2	2	2	4	5	4	5

1 ： AT END指定／INVALID KEY 指定に記述した文が実行されます。

2 ： 誤り処理手続きが実行された後、エラーが発生した入出力文の直後の文から処理が続行されます。

3 ： エラーが発生した入出力文の直後の文から処理が続行されます。

4 ： Iレベルメッセージが出力されて、エラーが発生した入出力文の直後の文から処理が続行されます。

5 ： Uレベルメッセージが出力されて、プログラムが異常終了します。

参考

- 有効な誤り処理手続きがある場合、環境変数情報@CBR_FILE_USE_MESSAGE=YESを指定すると、Iレベルメッセージが出力されます。

```
@CBR_FILE_USE_MESSAGE=YES
```

- 環境変数情報の指定誤りを示す入出力エラーが発生した場合、Uレベルの実行時メッセージを出力させて、診断機能を利用してアプリケーション実行時に設定されていた環境変数および実行環境情報を確認する方法もあります。

  診断機能で診断レポートを出力するには、“表7.4 入出力エラーが発生したときの実行結果”の組合せの通り、誤り処理手続きとFILE STATUS句の指定が“無”でなければなりません。この場合は、プログラムのデバッグを目的として、一時的にプログラムを修正する必要があります。

  診断機能を利用したプログラムのデバッグについては、“20.1 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

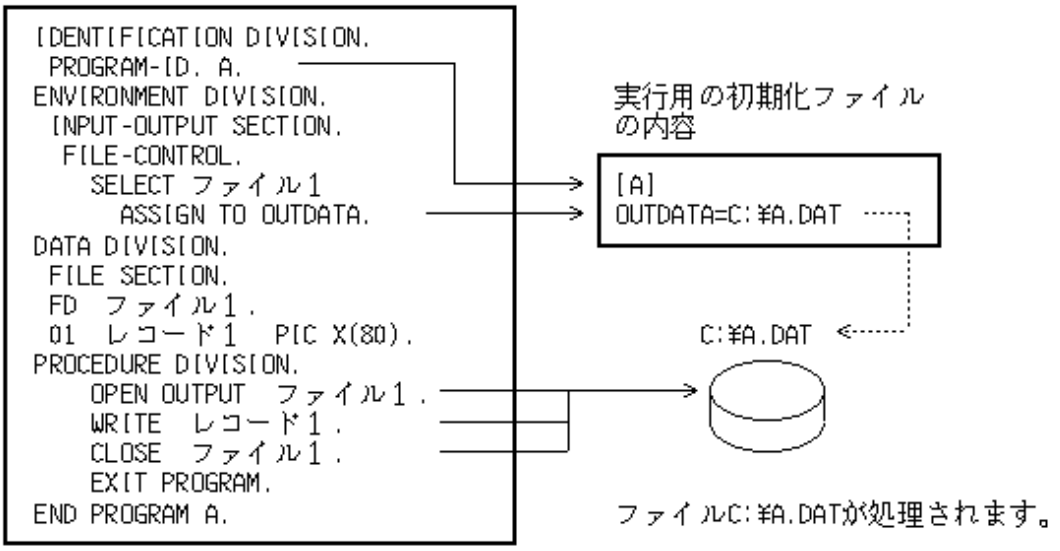
プログラムの実行時に入出力処理の対象となるファイルの決定方法は、ファイル管理記述項のASSIGN句の記述内容によって異なります。ASSIGN句の記述内容とファイルの関係を以下に示します。

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 実行環境情報の設定”を参照してください。

例

実行用の初期化ファイルを使った環境変数情報を設定する場合の例



注意

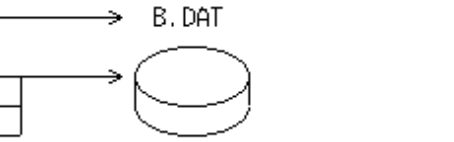
- ASSIGN句に英小文字のファイル識別名を指定した場合、以下の注意が必要です。
  - 翻訳オプションNOALPHALを指定して翻訳すると、翻訳エラーとなります。
  - 環境変数情報を設定するときには、英大文字で指定してください。
- 環境変数情報の内容が空白の場合、ファイルの割当てエラーとなります。

ASSIGN句にデータ名を記述した場合

データ名に設定されたファイル名のファイルに対して入出力処理が行われます。

例

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. B.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT ファイル1
      ASSIGN TO  データ名1 .
DATA DIVISION.
 FILE SECTION.
 FD  ファイル1 .
 01  レコード1   PIC X(80).
 WORKING-STORAGE SECTION.
 01  データ名1  PIC X(30).
PROCEDURE DIVISION.
    MOVE "B.DAT" TO  データ名1 .
    OPEN OUTPUT  ファイル1 .
    WRITE  レコード1 .
    CLOSE  ファイル1 .
    EXIT PROGRAM.
END PROGRAM B.
```



☆プログラムB を実行すると、
ファイル B.DATが処理されます。

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

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. C.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT ファイル1
      ASSIGN TO "C:¥C.DAT".
DATA DIVISION.
 FILE SECTION.
 FD  ファイル1.
 01  レコード1   PIC X(80).
PROCEDURE DIVISION.
    OPEN OUTPUT  ファイル1.
    WRITE  レコード1.
    CLOSE  ファイル1.
    EXIT PROGRAM.
END PROGRAM C.
```

☆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

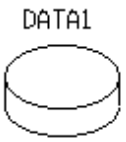
プログラムに記述されたファイル名が相対パス名の場合、カレントフォルダを先頭に付加したファイルが入出力処理の対象となります。

ASSIGN句に文字列DISKを記述した場合

SELECT句に記述したファイル名のファイルに対して入出力処理が行われます。

例

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. D.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT DATA1
      ASSIGN TO DISK.
DATA DIVISION.
 FILE SECTION.
 FD  DATA1.
 01  レコード１   PIC X(80).
PROCEDURE DIVISION.
    OPEN OUTPUT  DATA1.
    WRITE  レコード１.
    CLOSE  DATA1.
    EXIT PROGRAM.
END PROGRAM D.
```



☆プログラムD を実行すると、
ファイル DATA1が処理されます。

注意

ファイルはカレントフォルダが対象となります。

7.7.2 ファイルの排他制御

ファイル処理では、ファイル自体を排他モードにしたり、使用中のレコードを排他状態にすることにより、他からのアクセスを不可能にすることができます。これをファイルの排他制御といいます。

ファイルの排他制御は、COBOLプログラム、COBOLファイルアクセスルーチンまたはCOBOLファイルユーティリティを使用したアクセスに対して有効です。他言語や各種ツールからのアクセスでは、ファイルの排他制御は無効となり、同時にアクセスした場合の動作は保証されません。

ここでは、ファイル処理とファイルの排他制御の関係について説明します。

7.7.2.1 ファイルを排他モードにする方法

ファイルを排他モードで開くと、他の使用者はそのファイルをアクセスすることができません。

次の場合、ファイルは排他モードで開かれます。

- ファイル管理記述項のLOCK MODE句にEXCLUSIVEを指定したファイルに対してOPEN文を実行します。
- ファイル管理記述項のLOCK MODE句を指定しないファイルに対してINPUTモード以外のOPEN文を実行します。
- WITH LOCK指定のOPEN文を実行します。
- OUTPUTモードのOPEN文を実行します。

上記組合せによる、ファイルのモードの状態を下表に示します。

表7.5 LOCK MODE句が指定されていない場合

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

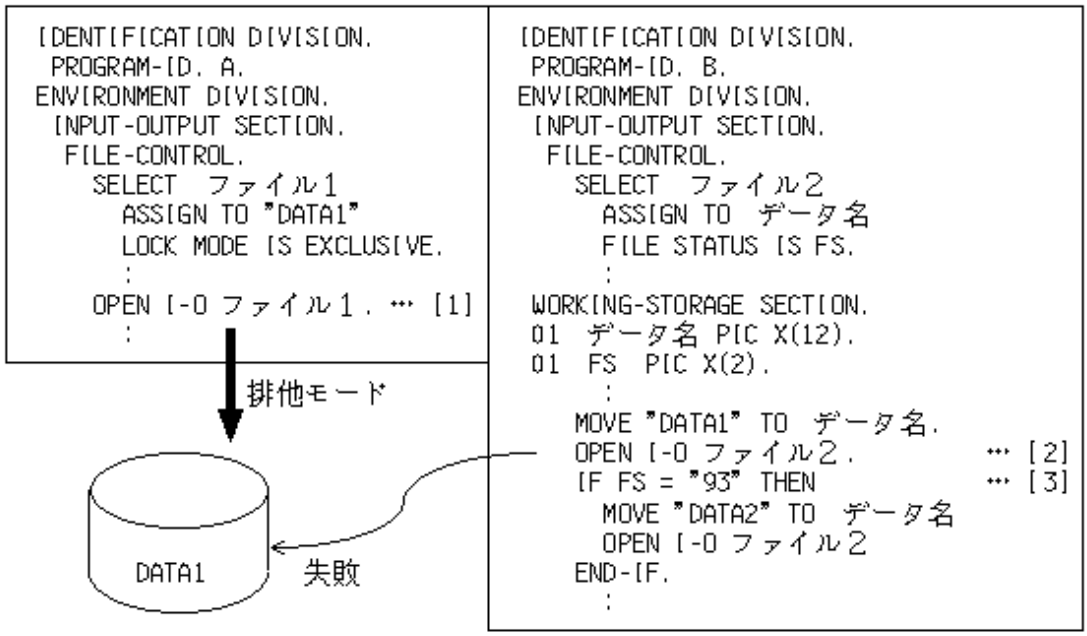
表7.6 LOCK MODE句にEXCLUSIVEが指定されている場合

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

表7.7 LOCK MODE句にAUTOMATICまたはMANUALが指定されている場合

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

例



[1] 排他モードでファイル(DATA1)を開きます。

[2] プログラムAですでに排他モードで使用されているファイル(DATA1)に対して、OPEN文を実行しても失敗となります。

[3] FILE STATUS句に指定したデータ名に入出力状態値“93”(ファイルの排他によるエラー発生)が設定されます。

7.7.2.2 レコードを排他状態にする方法

任意のレコードを排他状態にすると、他の利用者はそのレコードを処理することができません。任意のレコードを排他状態にするためには、まず、ファイルを共用モードで開きます。

次の場合、ファイルは共用モードで開かれます。

- ファイル管理記述項のLOCK MODE句にAUTOMATICまたはMANUALを指定したファイルに対して、OUTPUTモード以外のWITH LOCK指定なしのOPEN文を実行します。
- LOCK MODE句を指定しないファイルに対してINPUTモードのOPEN文を実行します。

共用モードで開かれたファイルは、他の利用者と共用して使用することができます。ただし、すでに他の利用者がそのファイルを排他モードで使用しているとき、OPEN文は失敗となります。共用モードで開かれたファイルのレコードは、排他処理を指定した入出力文の実行により排他状態となります。

次の場合、レコードが排他状態となります。

- ファイル管理記述項のLOCK MODE句にAUTOMATICを指定したファイルを入出力モードで開き、WITH NO LOCK指定なしのREAD文を実行します。
- ファイル管理記述項のLOCK MODE句にMANUALを指定したファイルを入出力モードで開き、WITH LOCK指定ありのREAD文を実行します。

上記組合せによる、レコードの状態を下表に示します。

表7.8 レコードの状態(排他/共用)

LOCK MODE 句の記述	AUTOMATIC			MANUAL		
READ文の記述	記述なし	WITH LOCK	WITH NO LOCK	記述なし	WITH LOCK	WITH NO LOCK
レコードの排他状態	する	する	しない	しない	する	しない

また、次の場合、レコードの排他状態が解除されます。

LOCK MODE句にAUTOMATICを指定したファイルの場合

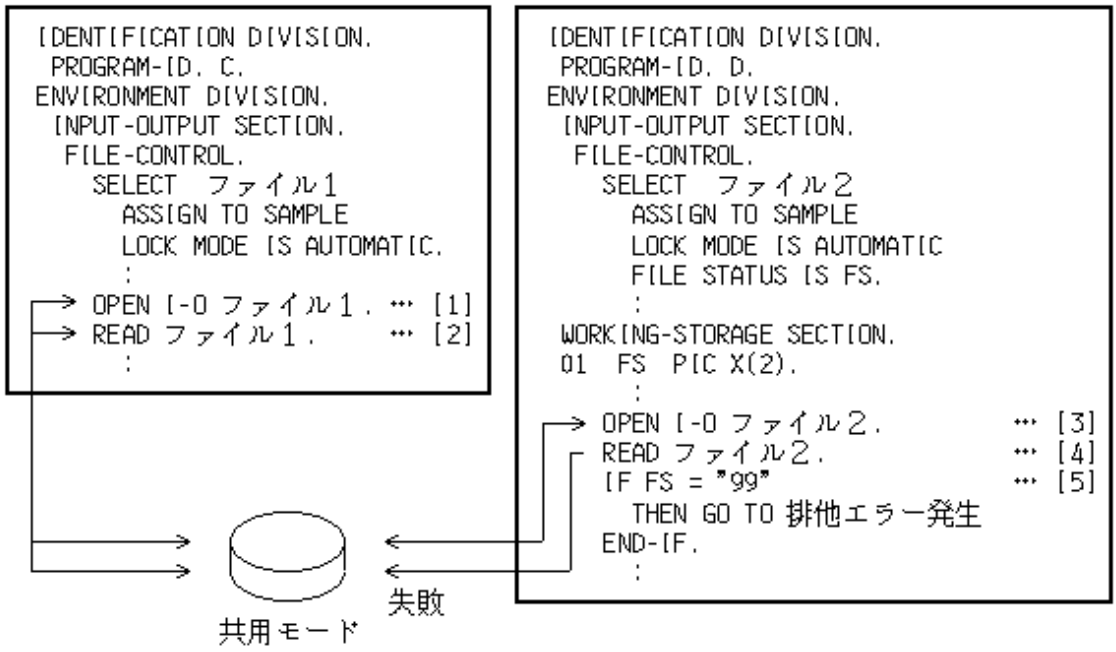
- READ文/REWRITE文/WRITE文/DELETE文/START文を実行します。
- UNLOCK文を実行します。
- CLOSE文を実行します。

LOCK MODE句にMANUALを指定したファイルの場合

- UNLOCK文を実行します。
- CLOSE文を実行します。

例

レコードの排他処理



[1] 共用モードでファイルを開きます。

[2] READ文の実行により、ファイルの先頭レコードは排他状態となります。

[3] 共用モードでファイルを開きます。

[4] 排他状態となっているレコードに対するREAD文は失敗となります。

[5] FILE STATUS句に指定したデータ名に入出力状態値“99”(レコードの排他によるエラー発生)が設定されます。

7.7.3 ファイル処理の結果

ファイル処理を行うことにより、新しいファイルが作られたり、ファイル中の内容が書き換えられたりします。ここでは、ファイル処理を行ったときのファイルの状態について説明します。

ファイルの創成処理を行ったとき

創成処理では、OPEN文の実行により、新しいファイルが作られます。このとき、同じファイル名のファイルがすでに存在している場合、そのファイルは新しく作り直され、元の内容は失われます。

ファイルの拡張処理を行ったとき

既存ファイルに対しての拡張処理では、WRITE文の実行によってファイルが拡張されます。プログラム実行時に存在しないファイル(不定ファイル)に対しての拡張処理では、OPEN文の実行により、新しいファイルが作られます。

レコードの参照処理を行ったとき

参照処理では、参照したファイルの内容は変更されません。不定ファイルに対しての参照処理では、最初のREAD文でファイル終了条件が発生します。

レコードの更新/削除/挿入処理を行ったとき

既存ファイルに対しての更新/削除/挿入処理では、REWRITE文/DELETE文/WRITE文の実行により、ファイルの内容が変更されます。不定ファイルに対しての更新/削除/挿入処理では、OPEN文の実行により、新しいファイルが作られます。ただし、このファイルにはデータが存在しないので、最初のREAD文でファイル終了条件が発生します。

注意

- CLOSE文を実行しないでプログラムを終了すると、そのファイルは強制的に閉じられます。これを強制クローズといいます。強制クローズは、以下の場合に行なわれます。
  - STOP RUN文の実行
  - 主プログラムでのEXIT PROGRAM文の実行
  - 外部プログラムに対するCANCEL文の実行
  - 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.2.75 ファイル識別名（プログラムで使用するファイルの指定）”を参照してください。

```
ファイル識別名=［パス名］ファイル名,BSAM
```

プログラム中のファイル参照子にデータ名を指定する場合

プログラム中のデータ名の指定時に、割り当てるファイルのファイル名に続き“,BSAM”を指定します。
ドライブ名を省略した場合、カレントドライブとみなされます。

```
MOVE  "［パス名］ファイル名,BSAM"  TO  データ名.
```

プログラム中のファイル参照子にファイル識別名定数を指定する場合

プログラム中のファイル識別名定数の指定時に、割り当てるファイルのファイル名に続き“,BSAM”を指定します。

```
ASSIGN TO  "［パス名］ファイル名,BSAM"
```

注意

- レコードの更新(REWRITE文)は、できません。レコードの更新を行った場合は、実行時にエラーとなります(レコード順ファイルだけ)。
- ファイル共用する場合には、以下の注意が必要です。

  他プロセス間でのファイル共用は、すべてのファイルが共用モードで、かつINPUT指定で開かれている必要があります。INPUT指定以外で開いたファイルがある場合、動作は保証されません。また、同一プロセス内はファイル共用できません。同一プロセス内でファイル共用した場合、動作は保証されません。

  なお、OPENモードがINPUT指定以外の場合は、常に排他モードでOPENします。このため、他のプログラムからアクセスした際、OPENエラーになる場合があります。[参照]“ファイル共用時の注意事項”
- ファイル参照子にDISKを指定した場合、この機能は使用できません。
- 行順ファイルの場合、読み込んだレコードにタブが存在していても、そのタブコードを空白に置き換えません。また、制御文字(0x0C(改頁)、0x0D(復帰)、0x1A(データ終了記号))が含まれていても、レコードの区切り文字やファイルの終端として扱いません。ファイルの高速処理を指定しない場合のタブや制御文字の扱いについては、“7.3.3 行順ファイルの処理”の“レコード内のタブの扱い”および“レコード内の制御文字の扱い”を参照してください。
- レコードの書込み(WRITE文)におけるADVANCING指定は有効となりません。指定した場合は、ADVANCING指定のないWRITE文と同じ結果となります。

一括指定

ファイルの高速処理を一括して有効とする指定方法について説明します。

使い方

ファイルの高速処理を有効とする場合、環境変数情報@CBR_FILE_SEQUENTIAL_ACCESSに“BSAM”を指定します。

```
@CBR_FILE_SEQUENTIAL_ACCESS = BSAM
```

本環境変数は、以下のファイルに対して有効になります。

ファイル編成	• レコード順ファイル • 行順ファイル
ASSIGN句の指定	• ファイル識別名 • ファイル識別名定数 • データ名 • DISK

ただし、以下の機能を指定したファイルに対しては、有効になりません。

機能	ファイル機能名
ファイルの追加書き(注)	MOD
ファイルの連結(注)	CONCAT
ダミーファイル	DUMMY
他のファイルシステム	BTRV
	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。

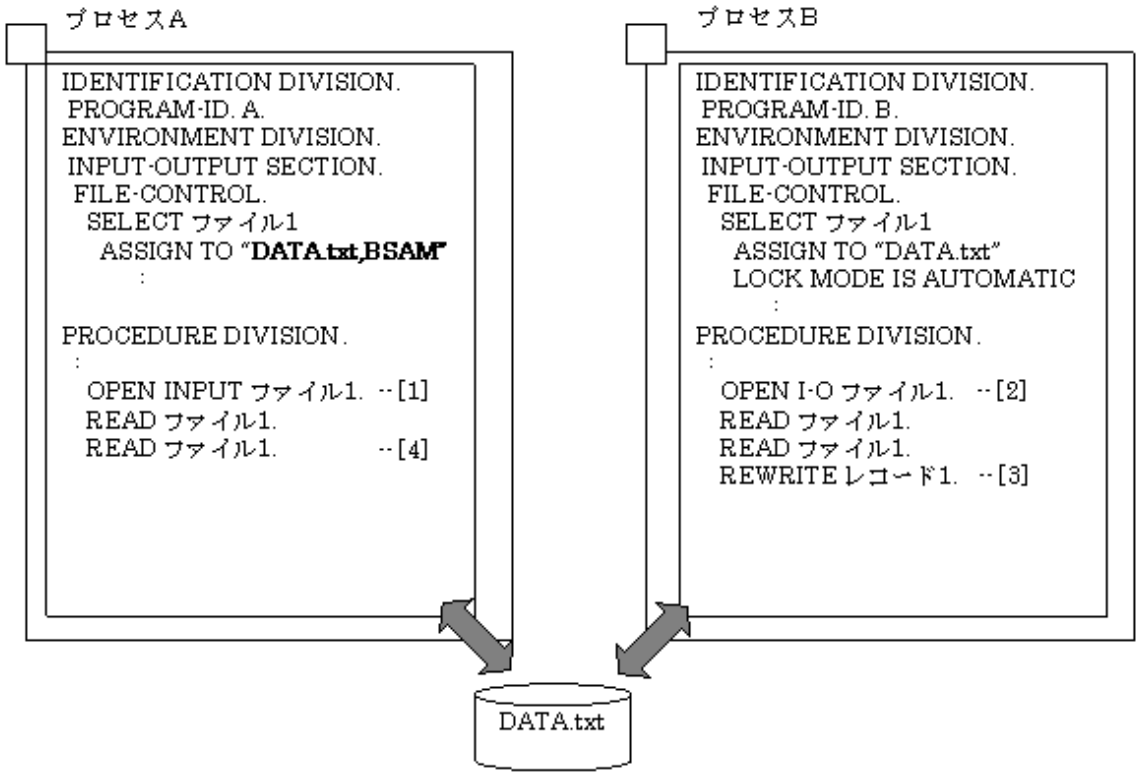
[参照]“ファイル共用時の注意事項”

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。



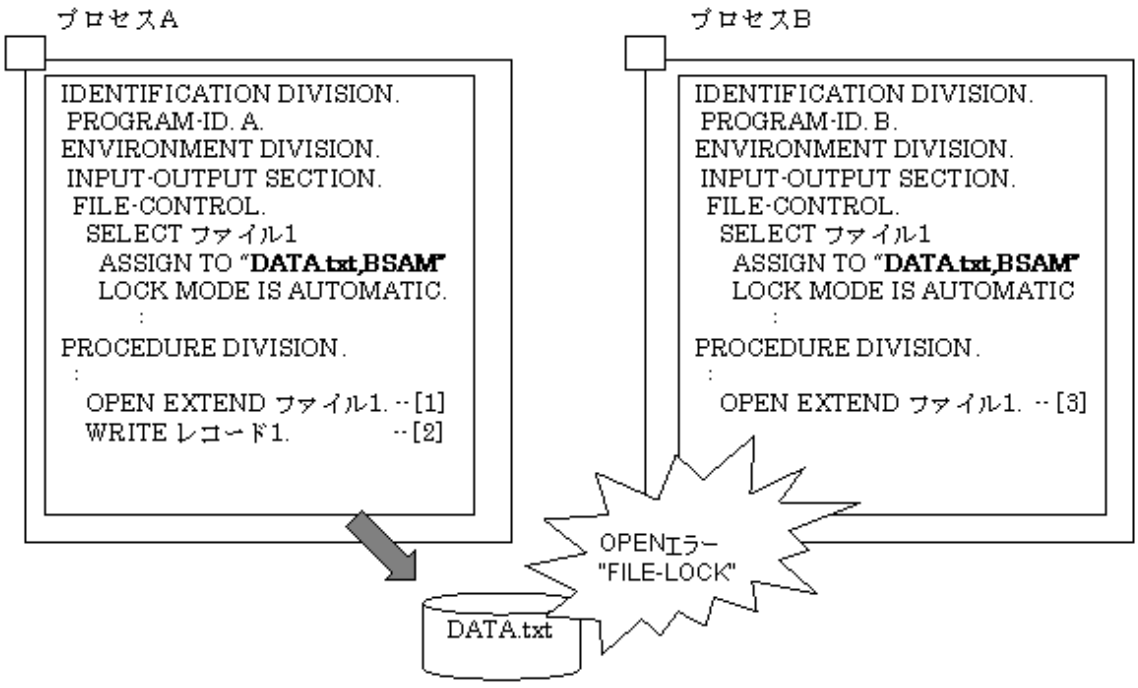
[1] プロセスA：共用モード、INPUT指定でファイルを開きます。(ファイルの高速処理)

[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。

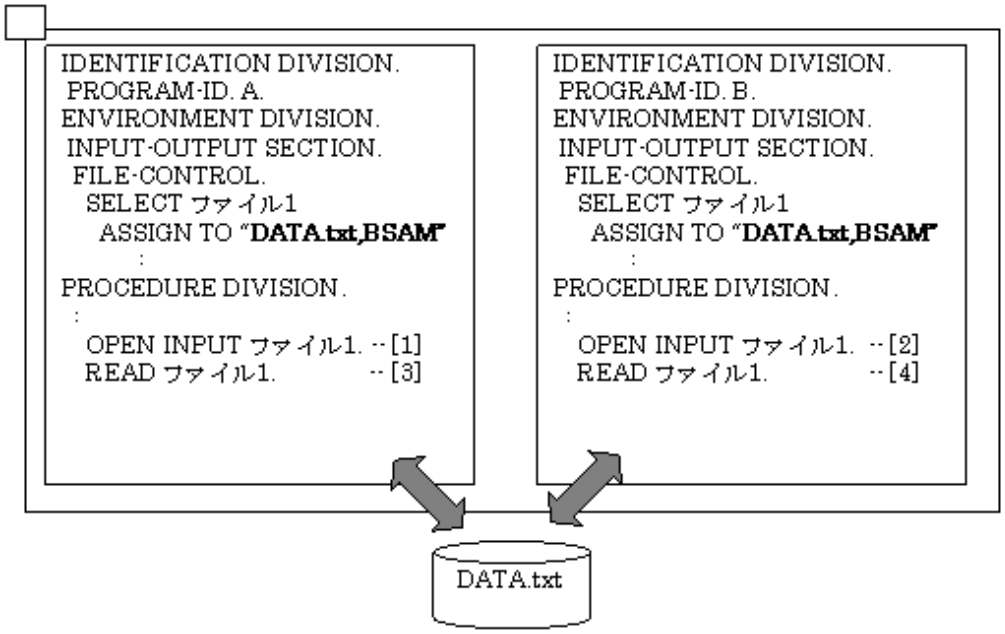


[1] プロセスA：共用モード、EXTEND指定でファイルを開きます。

[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用する場合、本機能は使用できません。



[1] プログラムA：共用モード、INPUT指定でファイルを開きます。

[2] プログラムB：共用モード、INPUT指定でファイルを開きます。

[3] プログラムA：1件目のレコードを読み込みます。

[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。

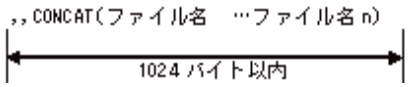
使い方

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。



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止し、トラブル発生時にダミーファイルとしての扱いを外して、ログファイルを出力するという使い方があります。

また、プログラム開発途中では入力ファイルがない場合があります。ダミーファイルとすることで、空のファイルなどの不要なファイルを用意する手間が省け、作業の効率を向上させることができます。

ここでは、ダミーファイルの使い方と機能範囲について説明します。

使い方

ファイル識別名に、割り当てるファイルのファイル名に続き、“,DUMMY”を指定します。

ファイル名の指定は任意です。省略してもかまいません。

```
ファイル識別名=[ファイル名],DUMMY
```

注意

- 文字列“DUMMY”の前にコンマ(,)が必要です。コンマ(,)がない場合、文字列“DUMMY”をファイル名とみなして、通常のファイルと同じ動作をします。エラーにはならないことに注意してください。
- ファイル名の指定の有無による動作の違いはありません。指定したファイル名は意味を持ちません。指定したファイルが存在した場合も、既存ファイルに対する操作は行われません。

機能範囲

ダミーファイルを使用する場合、有効な機能範囲を以下に示します。

表7.9 ダミーファイルの機能範囲

ファイル編成	レコード順ファイル 行順ファイル 相対ファイル 索引ファイル		
オープンモード	OUTPUT EXTEND I-O INPUT		
入出力文	OPEN文	入出力文の実行が成功します。	
	CLOSE文		
	WRITE文		
	START文		
	UNLOCK文		
	READ文	順呼出し	ファイル終了条件が発生します。
		乱呼出し	無効キー条件が発生します。
	REWRITE文 DELETE文	順呼出し	先行するREAD文が不成功になるため、実行順序誤りになります。
		乱呼出し	無効キー条件が発生します。

7.7.8 注意事項

ファイル識別名に指定できる文字列のバイト数

ファイル識別名には、1024バイト以内の文字列を指定してください。

文字列が1024バイトを超えた場合は、1024バイトまでの文字列を有効とみなして処理します。

ただし、ファイル連結機能の指定の途中で文字列が1024バイトを超えた場合は、OPEN文実行時にエラーになります。

同時に指定可能なファイル機能の組合せ

ファイル機能は、以下の3つの種別に分類することができます。

種別	ファイル機能名	機能
(1) アクセス種別	BSAM	ファイルの高速処理
	DUMMY	ダミーファイル
(2) ファイルシステム種別	BTRV	Btrieveファイル
	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 / 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ファイルを処理するためのユーティリティです。具体的には以下の機能を持ちます。

- 各種テキストエディタを使って作成したデータからCOBOLファイルを作成する。
- COBOLファイルに対する以下の操作を行う。
  - 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。

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	-	-
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ったり、レコードの挿入や削除を行うことができなかったりします。[参照]“表7.2 ファイルの種類と処理”
- 本ユーティリティのウィンドウを使い、出力ファイルに指定したファイルが既に存在していた場合、そのファイルを上書きするかどうか確認を求めます。しかし、コマンドや関数を使用した場合、上書き確認をせず、エラーとします。
- 本ユーティリティは、ファイルの高速処理に対応していません。ファイル名に続き「,BSAM」を指定した場合、エラーとします。

- 本ユーティリティは、操作対象となるファイルがCOBOLプログラムや他のユーティリティからアクセスされている場合、エラーとします。このため、同じファイルに対するユーティリティの同時実行はできません。なお、索引ファイルの復旧処理では、同時実行した場合の動作は保証されません。直前の操作が完了したことを確認してから再度実行してください。

7.8.2 COBOLファイルユーティリティの使い方

ここでは、COBOLファイルユーティリティの使い方を、操作手順に従って説明していきます。

1. 環境設定

   ー COBOLランタイムシステムが格納されているパス名を環境変数PATHに設定してください。

   ー 環境変数TMPまたはTEMPに、作業用一時ファイルを作成するフォルダを、ドライブ名から始まるパス名で設定してください。本ユーティリティで、〔復旧〕コマンドを実行した場合、指定されたフォルダに処理するファイルと同じ大きさの一時的な作業用のファイル(UTYnnnn.TMP(nnnnは英数字を示す))を作成します。

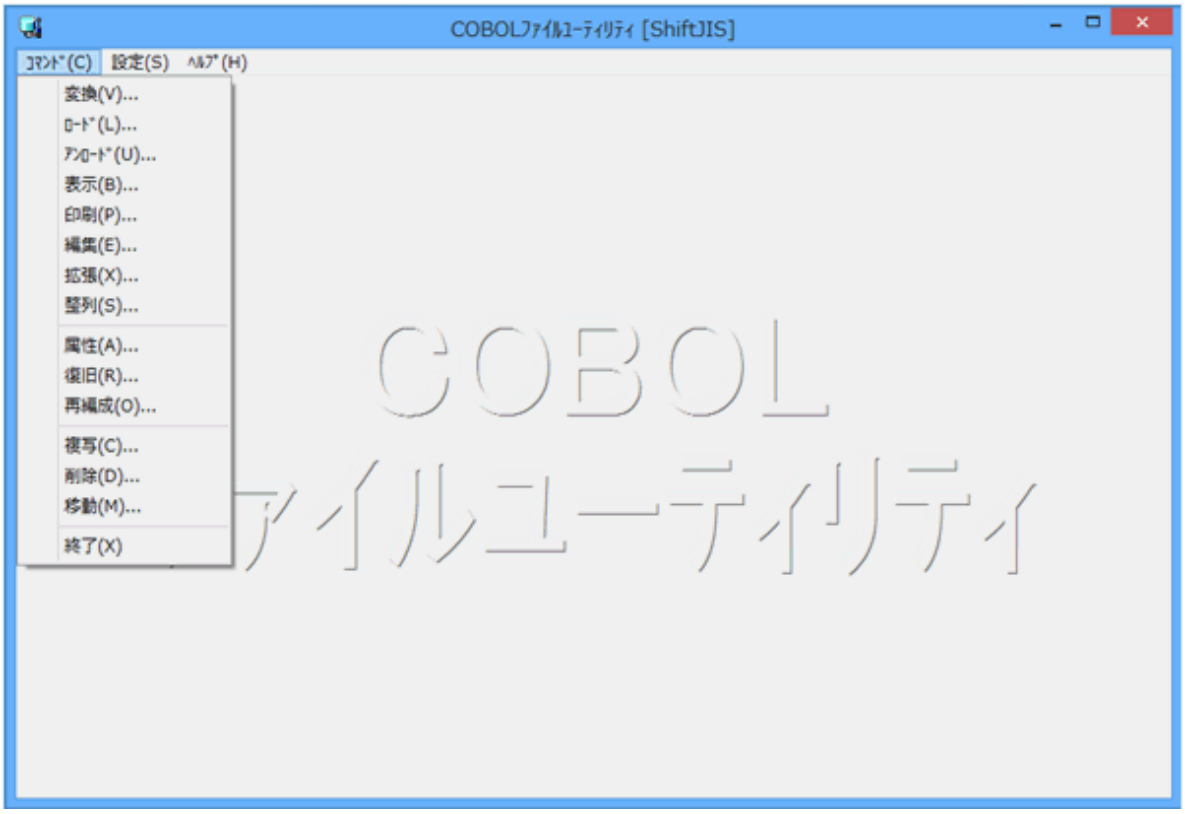
   ー 環境変数BSORT_TMPDIRに整列併合用ファイルを作成するフォルダを、ドライブ名から始まるパス名で設定してください。本ユーティリティで、〔整列〕コマンドを実行した場合、指定されたフォルダに一時的な作業用のファイルを作成します。環境変数BSORT_TMPDIRを設定していない場合の作業用ファイルについては、“12.2.4 プログラムの実行”を参照してください。

2. COBOLファイルユーティリティの起動

   〔スタート〕>〔すべてのアプリ〕> お使いのNetCOBOL製品名 >〔COBOLファイルユーティリティ〕を起動します。

3. 処理の選択

   〔コマンド〕メニューから、実行するコマンドを選択します。選択を行うと、各コマンドを実行するためのダイアログボックスが表示されます。
   各機能については、“7.8.3 COBOLファイルユーティリティの機能”を参照してください。



コマンド名	機能
変換	テキストファイルから可変長レコード形式のレコード順ファイルを作成します。または、可変長レコード形式のレコード順ファイルからテキストファイルを作成します。

コマンド名	機能
ロード	可変長レコード形式のレコード順ファイルから、任意のCOBOLファイルを作成します。また、可変長レコード形式のレコード順ファイルの全レコードを、任意の属性のファイルに拡張(レコードの追加)することができます。
アンロード	任意のCOBOLファイルから、可変長レコード形式のレコード順ファイルを作成します。
表示	任意のCOBOLファイルのレコードを1件ずつ、画面に表示します。
印刷	任意のCOBOLファイルのレコードを印刷します。
編集	任意のCOBOLファイルのレコードを1件ずつ、画面に表示して、レコード内容を編集することができます。
拡張	任意のCOBOLファイルにレコードを1件ずつ、追加することができます。
整列	任意のCOBOLファイルのレコードを整列し、その整列結果を可変長レコード形式のレコード順ファイルに格納します。
属性	索引ファイルの属性情報(レコード長など)を画面に表示します。
復旧	アクセスできなくなった索引ファイルを復旧します。
再編成	索引ファイルの未使用空間を削除し、ファイルサイズを小さくします。
複写	ファイルを複写します。
削除	ファイルを削除します。
移動	ファイルを別の場所に移動します。

4. COBOLファイルユーティリティの終了

   〔コマンド〕メニューから“終了”を選択します。

7.8.3 COBOLファイルユーティリティの機能

COBOLファイルユーティリティは、以下に示す機能を備えています。

- ファイルの創成
- ファイルの拡張
- レコードの表示
- レコードの編集
- レコードの整列
- ファイルの操作(複写、削除、移動)
- ファイルの印刷
- ファイル構造の変換
- 索引ファイルの操作(属性情報の表示、復旧、再編成)

これらの機能については、以下で説明します。なお、『』内は機能を使用するときのコマンド(〔〕:コマンド名、[]:省略可能)を示します。各コマンドのダイアログの操作方法については、ヘルプを参照してください。

7.8.3.1 ファイルの創成

各種テキストエディタを使って作成したデータを入力として、COBOLファイルを新規に作成することができます。『〔変換〕[+〔ロード〕]』

図7.1 操作手順

注意

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

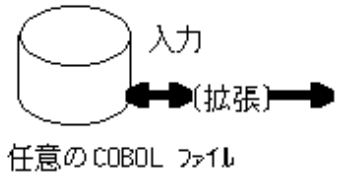
7.8.3.2 ファイルの拡張

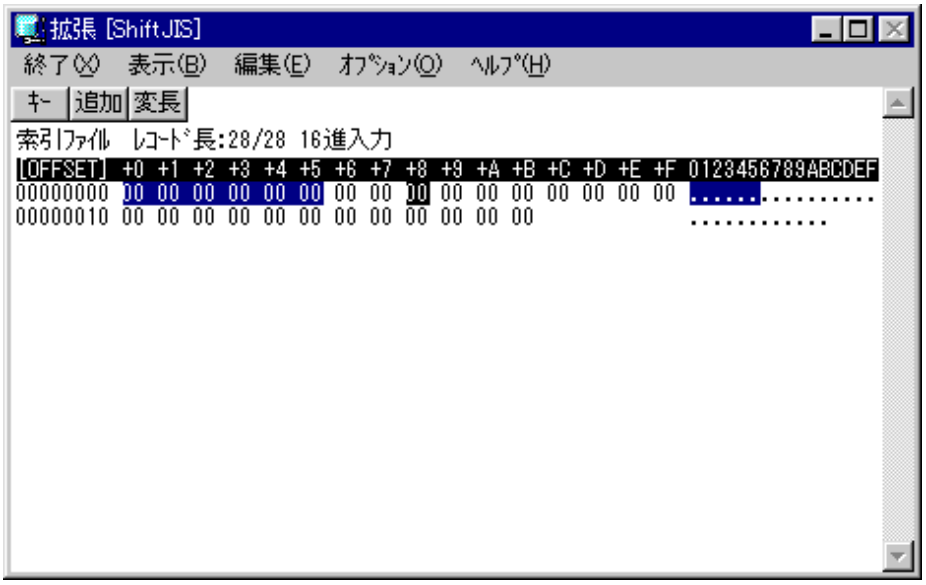
レコードの追加および他ファイルからのレコードの追加を行うことができます。

レコードの追加

既存のCOBOLファイルにレコードを1件ずつ追加することができます。『〔拡張〕』

図7.2 操作手順





→ レコードを1件ずつ追加します。

メニューバー	終了	拡張ウィンドウを終了します。
	表示	次に表示するレコードをどのように指示するかを選択します。
	編集	表示中のレコードを操作します。
	オプション	レコード操作画面の入力方法(16進入力/文字入力)の指定と、フォントの指定を行います。
	ヘルプ	拡張ウィンドウのヘルプを表示します。
ボタン	キー	読込み順序を変更する場合に選択します。読込み順序を変更すると、画面に表示されるレコードの順番は、選択したレコードキーの順番に依存するようになります。
	追加	画面に表示しているレコードをファイルに追加する場合に選択します。
	変長	画面に表示しているレコードの長さを変更する場合に選択します。
レコード操作画面(上記画面の場合)	索引ファイル	表示中のファイルの属性を示しています。
	レコード長:28/28	表示中のレコードのレコード長/最大レコード長を示しています。
	16進入力	レコード操作画面へのデータ入力方法を示しています。
	[OFFSET]の下の文字列および“+0 +1 … +E +F”	画面に表示されているレコードデータ内の位置のオフセットを16進数で示しています。上記画面でカーソルの位置(レコードデータ内のオフセット)は、カーソル行の“+8”とカーソルの左の“00000000”を加えた0x00000008となります。
	“+0 +1 … +E +F”の下の2個の文字列	レコードデータを1バイトごとに16進数で表示しています。
	“0123456789ABCDEF”の下の文字列	レコードデータを1バイト表記の文字で表示しています。

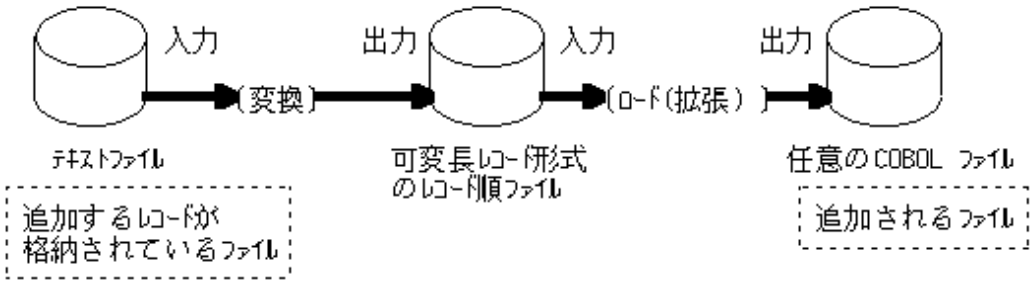
注意

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

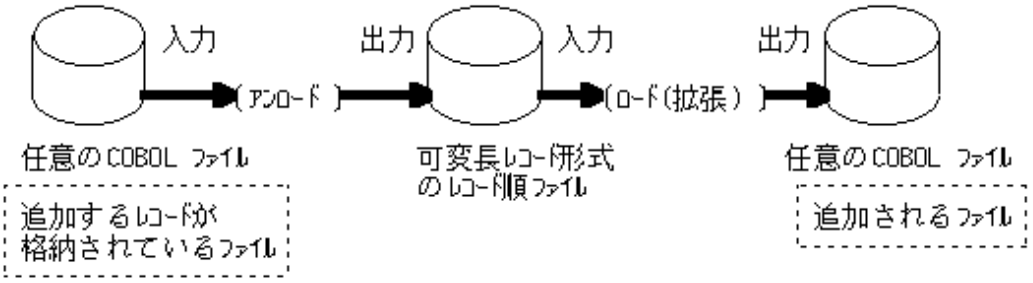
他ファイルからのレコードの追加

既存のCOBOLファイルに他のファイルの内容(レコード)を追加することができます。『[〔変換〕+]〔ロード(拡張)〕または[〔アンロード〕+]〔ロード(拡張)〕』

図7.3 操作手順



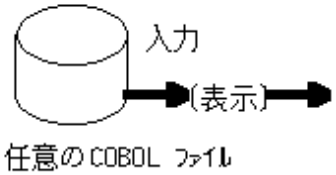
または、

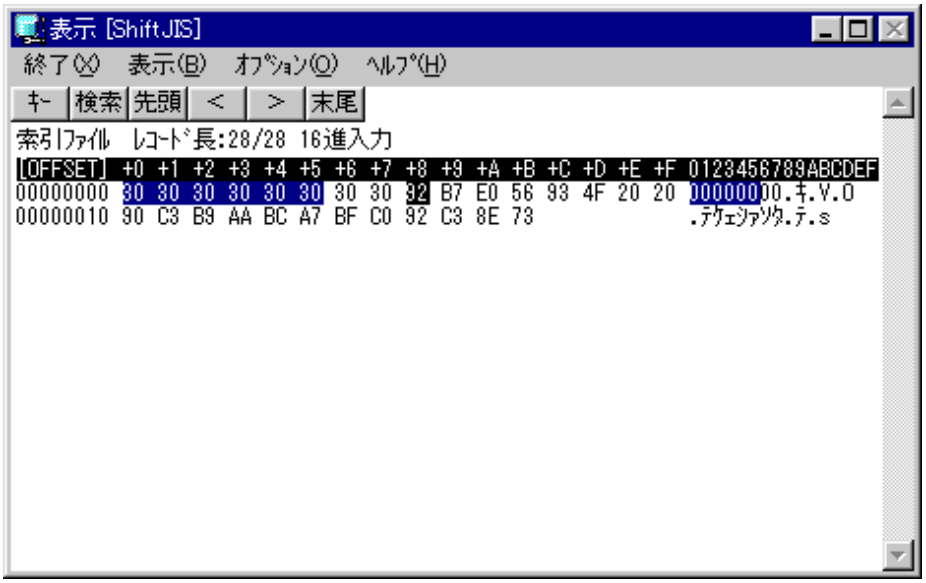


7.8.3.3 レコードの表示

COBOLファイルのレコードの内容を表示することができます。『〔表示〕』

図7.4 操作手順





→ ファイルの内容をレコード単位に文字と16進数で表示します。

メニューバー	終了	表示ウィンドウを終了します。
	表示	次に表示するレコードをどのように指示するかを選択します。
	オプション	レコード操作画面の入力方法(16進入力/文字入力)の指定と、フォントの指定を行います。
	ヘルプ	表示ウィンドウのヘルプを表示します。

ボタン	キー	読込み順序を変更する場合に選択します。読込み順序を変更すると、画面に表示されるレコードの順番は、選択したレコードキーの順番に依存するようになります。
	検索	画面に表示するレコードを直接指示する場合に選択します。
	先頭	ファイル内で論理的に先頭に格納されているレコードを表示する場合に選択します。
	<	現在表示しているレコードより論理的に前に格納されているレコードを表示する場合に選択します。
	>	現在表示しているレコードより論理的に後ろに格納されているレコードを表示する場合に選択します。
	末尾	ファイル内で論理的に末尾に格納されているレコードを表示する場合に選択します。
レコード操作画面(上記画面の場合)	索引ファイル	表示中のファイルの属性を示しています。
	レコード長:28/28	表示中のレコードのレコード長/最大レコード長を示しています。
	16進入力	レコード操作画面へのデータ入力方法を示しています。
	[OFFSET]の下の文字列および“+0 +1 … +E +F”	画面に表示されているレコードデータ内の位置のオフセットを16進数で示しています。上記画面でカーソルの位置(レコードデータ内のオフセット)は、カーソル行の“+8”とカーソルの左の“00000000”を加えた0x00000008となります。
	“+0 +1 … +E +F”の下の2個の文字列	レコードデータを1バイトごとに16進数で表示しています。
	“0123456789ABCDEF”の下の文字列	レコードデータを1バイト表記の文字で表示しています。

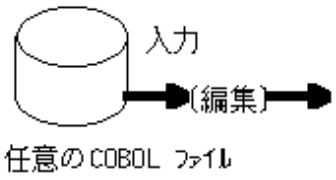
注意

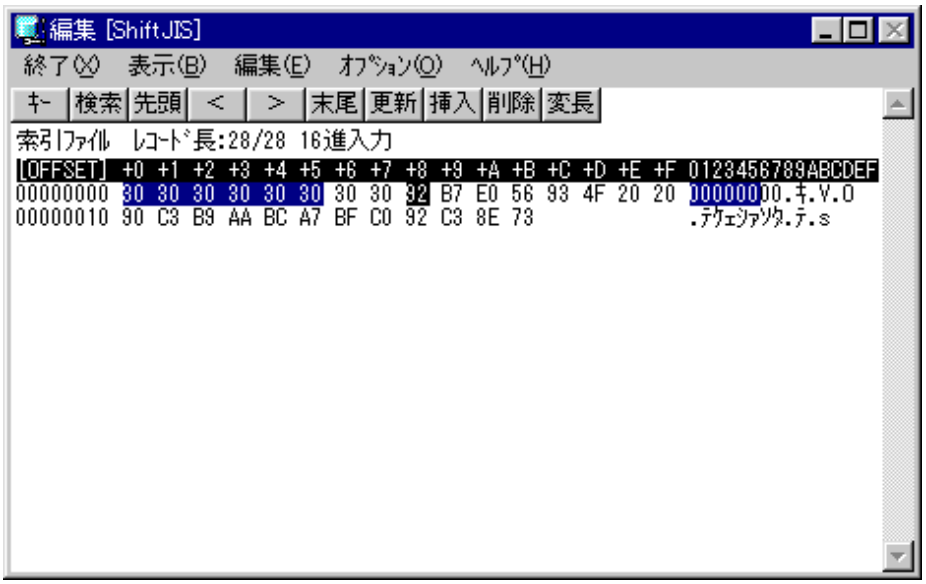
日本語文字などの表示できない文字は、ピリオドに置き換えられて表示されます。
詳細は、ヘルプを参照してください。

7.8.3.4 レコードの編集

COBOLファイルのレコードを更新、挿入および削除することができます。『〔編集〕』

図7.5 操作手順





→ ファイルの内容をレコード単位に編集します。

文字と16進数で表示されるレコードを編集し、挿入/削除/更新を行います。

メニューバー	終了	編集ウィンドウを終了します。
	表示	次に表示するレコードをどのように指示するかを選択します。
	編集	表示中のレコードを操作します。
	オプション	レコード操作画面の入力方法(16進入力/文字入力)の指定と、フォントの指定を行います。
	ヘルプ	編集ウィンドウのヘルプを表示します。
ボタン	キー	読込み順序を変更する場合に選択します。読込み順序を変更すると、画面に表示されるレコードの順番は、選択したレコードキーの順番に依存するようになります。
	検索	画面に表示するレコードを直接指示する場合に選択します。
	先頭	ファイル内で論理的に先頭に格納されているレコードを表示する場合に選択します。
	<	現在表示しているレコードより論理的に前に格納されているレコードを表示する場合に選択します。
	>	現在表示しているレコードより論理的に後ろに格納されているレコードを表示する場合に選択します。
	末尾	ファイル内で論理的に末尾に格納されているレコードを表示する場合に選択します。
	更新	画面に表示しているレコードを更新する場合に選択します。
	挿入	画面に表示しているレコードをファイルに追加する場合に選択します。
	削除	画面に表示しているレコードを削除する場合に選択します。
	変長	画面に表示しているレコードの長さを変更する場合に選択します。
レコード操作画面(上記画面の場合)	索引ファイル	表示中のファイルの属性を示しています。
	レコード長:28/28	表示中のレコードのレコード長/最大レコード長を示しています。
	16進入力	レコード操作画面へのデータ入力方法を示しています。
	[OFFSET]の下の文字列および“+0 +1 … +E +F”	画面に表示されているレコードデータ内の位置のオフセットを16進数で示しています。上記画面でカーソルの位置(レコードデータ内のオフセット)は、カーソル行の“+8”とカーソルの左の“00000000”を加えた0x00000008となります。

	“+0 +1 … +E +F”の下の2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。

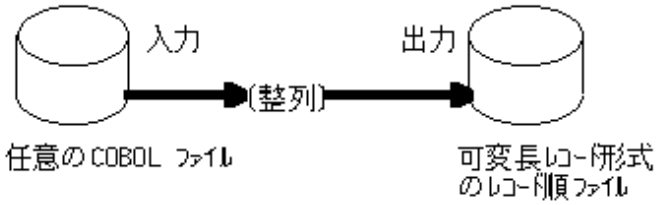
注意

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〕』

図7.6 操作手順



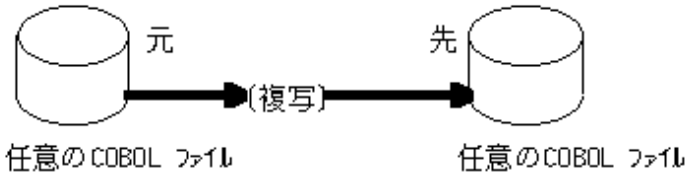
7.8.3.6 ファイルの操作

COBOLファイルを複写、削除および移動することができます。

ファイルの複写

COBOLファイルを複写します。『〔複写〕』

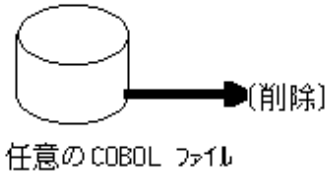
図7.7 操作手順



ファイルの削除

COBOLファイルを削除します。『〔削除〕』

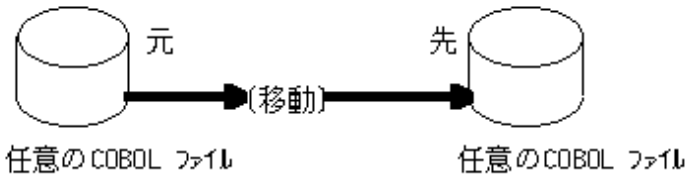
図7.8 操作手順



ファイルの移動

COBOLファイルを移動します。『〔移動〕』

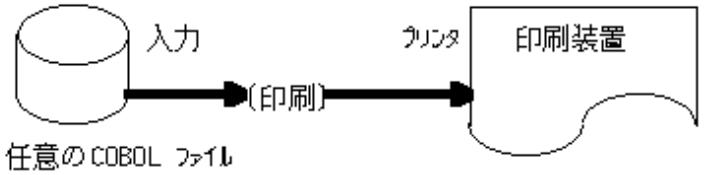
図7.9 操作手順



7.8.3.7 ファイルの印刷

COBOLファイルの内容を印刷することができます。『〔印刷〕』

図7.10 操作手順



7.8.3.8 ファイルの構造の変換

COBOLファイルの構造を、別の種類のファイル構造に変換します。『〔ロード〕、〔アンロード〕』

図7.11 操作手順

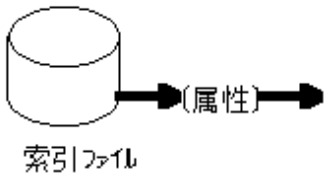
7.8.3.9 索引ファイルの操作

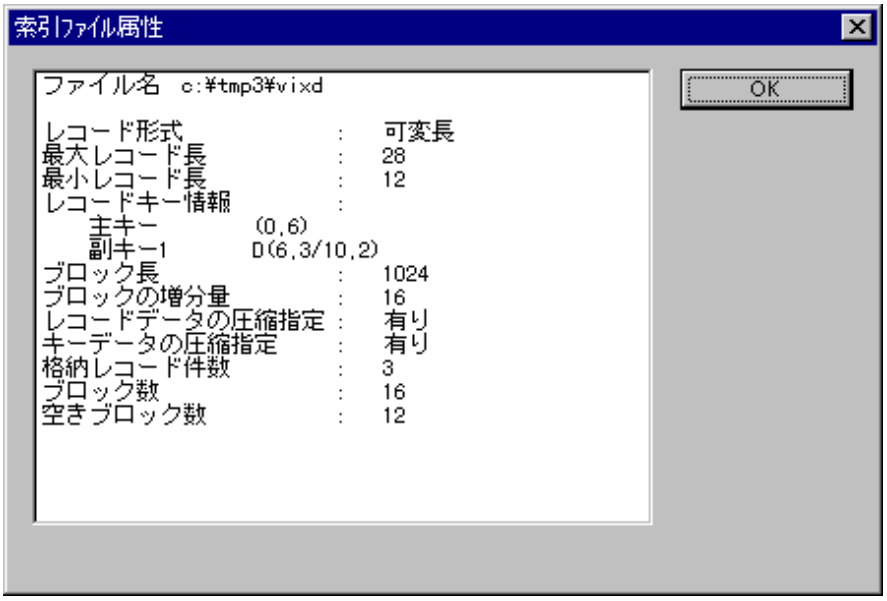
索引ファイルに対して、以下の操作を行うことができます。

属性情報の表示

索引ファイルの属性情報を表示します。『〔属性〕』

図7.12 操作手順





ファイル名	属性を表示している索引ファイル名を示しています。
レコード形式	レコード形式を示しています。(固定長/可変長)
最大レコード長	固定長の場合、レコード長を示しています。可変長の場合、最大レコード長を示しています。
最小レコード長	固定長の場合、表示されません。可変長の場合、最小レコード長を示しています。
レコードキー情報	索引ファイルのキー情報を示しています。(注)
ブロック長	索引ファイルの1ブロックの大きさを示しています。
ブロックの増分量	索引ファイルの増分単位をブロックの数で示しています。
レコードデータの圧縮指定	格納されているレコードデータが圧縮されているかどうかを示しています。
キーデータの圧縮指定	格納されているキーデータが圧縮されているかどうかを示しています。
格納レコード件数	索引ファイル中に格納されているレコードの件数を示しています。
ブロック数	索引ファイルを構成するブロックの数を示しています。
空きブロック数	索引ファイルを構成するブロックのうち、未使用なブロックの数を示しています。

注 ： キー情報の形式を以下に示します。

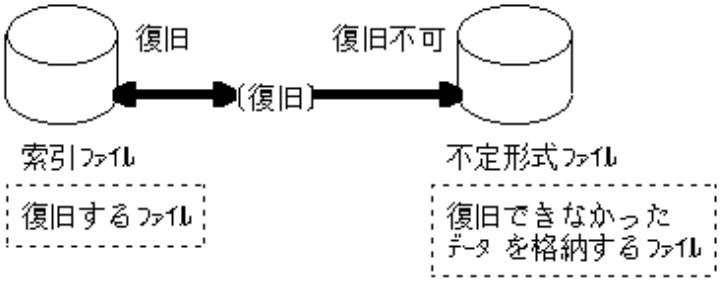
[D](オフセット,長さ[,{N|N32})[/オフセット,長さ[,{N|N32}]]…)

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〕』

図7.13 操作手順



再編成

索引ファイルの未使用空間を削除し、ファイルサイズを小さくします。『〔再編成〕』

図7.14 操作手順

7.9 他のファイルシステムの使用法

NetCOBOLでは、製品に同梱されているCOBOLファイルシステム以外にも、ファイル名に特定の文字列を指定することで、他のファイルシステムを使用することができます。

利用できるファイルシステムを以下に示します。

Btrieve

Btrieveは、Pervasive Software社のレコードマネジメントシステムです。

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の使用方法、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ファイルシステムおよびBtrieve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。

各ファイルの定量制限は、“付録E 定量制限”を参照してください。

表7.10 各ファイルシステムの機能差

分類	項目		COBOLファイルシステム	Btrieve(注1)
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	○	○
ファイル管理記述項	SELECT句	OPTIONAL指定	○	○
	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2)
	LOCK MODE句	AUTOMATIC	○	○
		EXCLUSIVE	○	○
		MANUAL	○	○
	レコードキーの項目	英数字	○	○
		日本語	○	○
		符号なし外部10進数	○	○
		符号付き外部10進数	－	○(注3)
		符号なし内部10進数	○	○(注3)
		符号付き内部10進数	－	○
		符号なし2進数(BINARY)	○	○
		符号付き2進数(BINARY)	－	○
		符号なし2進数(COMP-5)	－	○
		符号付き2進数(COMP-5)	－	○
文	READ文	WITH LOCK指定	○	○
	START文	キー全体を指定	○	○
		キーの一部を指定	○	○

分類	項目		COBOLファイルシステム	Btrieve(注1)
	DELETE文	キー値が重複しているレコードを削除	○	○(注4)
	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5)
		COMMIT指示	－	○(注5)
		ROLLBACK指示	－	○(注5)
	リカバリ		○(注6)	○(注6)

○ ： サポート
－ ： 非サポート

- 注1
  Btrieveについては、“Pervasive PSQL V10 SP1”の情報をもとにしています。
- 注2
  Btrieveは、FILE STATUS=02を返却しません。
- 注3
  BtrieveファイルのNUMERICタイプは、COBOLではSEPARATE指定のない符号付き外部10進項目のデータ型に相当しますが、NUMERICの内部形式は88コンソーシアム形式と呼ばれる内部形式となっており、NetCOBOLの内部形式とは異なります。そのため、Btrieveファイルを使用して、SEPARATE指定のない符号付き外部10進項目を入力または出力項目として扱う場合、書込み前および読込み後にデータを変換する必要があります。データ変換の詳細については、“7.9.1.2 外部10進項目のデータ形式変換”を参照してください。
- 注4
  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5
  Btrieve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。
- 注6
  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。

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。

注意

日本語項目のエンディアンにリトルエンディアンを選択し、かつ、索引編成のBtrieveファイルで、索引キー(基本項目、集団項目のいずれの場合も)に日本語項目が含まれる場合、日本語項目に格納されたデータはリトルエンディアンのままで比較処理されるため、意図したレコードに位置づけられない可能性があります。

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、書込み前および読込み後に行う必要があります。

BtrieveのNUMERICタイプのデータの内部形式については、"Btrieve"または"Pervasive.SQL"のマニュアルをお持ちの方は、こちらを参照してください。お持ちでない方は、技術員(SE)に連絡してください。

記述形式

- 88コンソーシアム形式からNetCOBOLの内部形式への変換

```
CALL "#DEC88TOFJ" USING [BY REFERENCE]   一意名
```

- NetCOBOLの内部形式から88コンソーシアム形式への変換

```
CALL "#DECFJTO88" 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ファイルへの書込み
     OPEN  OUTPUT ファイル.
     CALL  "#DECFJTO88" 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 "#DEC88TOFJ" 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 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。

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

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

ここでは、索引ファイル復旧関数および索引ファイル簡易復旧関数について説明します。

7.10.1 索引ファイル復旧関数(CFURCOV)

ここでは、索引ファイル復旧関数について説明します。

機能

索引ファイル復旧関数は、正常に閉じることができなかったために使用できない状態にある索引ファイルを使用可能にするため、ファイルの先頭から正常な部分を再生し、異常部分を別ファイルに出力します。つまり、COBOLファイルユーティリティの〔復旧〕コマンドと同じ機能を持っています。
再度使用できない状態にある索引ファイルを開いた場合は、入出力状態に“39”または“90”が返却されます。

記述形式

```
#include "f4agfutc.h"  /* 関数宣言 */
    signed __int64 CFURCOV( char *ixdfilename,  char *blkdatname,  char *message);
```

ixdfilename ：復旧するファイルの名前
blkdatname ： 復旧できなかったデータを格納するファイルの名前
message ： 索引ファイル復旧関数の結果（メッセージ）を格納する領域

関数名

CFURCOV

第一パラメタ(ixdfilename)

復旧する索引ファイルの名前(文字列)が格納されている領域のアドレスを指定します。文字列の終端位置には、NULL文字(0x00)または空白文字(0x20)を設定する必要があります。また、ファイル名に空白またはコンマ(,)を含む場合は、ファイル名を二重引用符で囲む必要があります。(例:”¥”C:¥¥ixd file¥”¥0”)

第二パラメタ(blkdatname)

復旧不可能であったレコードを書き込むファイルの名前(文字列)が格納されている領域のアドレスを指定します。文字列の終端位置には、NULL文字(0x00)または空白文字(0x20)を設定する必要があります。また、ファイル名に空白またはコンマ(,)を含む場合は、ファイル名を二重引用符で囲む必要があります。なお、復旧不可能であったレコードを書き込むファイルが必要なければ、0を指定します。
既に存在するファイルの名前を指定した場合、エラーとなります。

第三パラメタ(message)

索引ファイル復旧関数の実行結果を示すメッセージを格納する領域のアドレスを指定します。格納する領域は、呼出し元で確保(512バイト必要)してください。
なお、メッセージが必要なければ、0を指定します。

復帰値

索引ファイル復旧関数の結果として、メッセージに対応するコードを返却します。
コードの値とメッセージの内容については、“表7.11 索引ファイル復旧関数(簡易復旧関数)が返却するメッセージの内容とコード”を参照してください。

例

```
#include "f4agfutc.h"
void callcobfrcov(void)
{
    char ixdfilename[512] = "c:¥¥ixdfile¥0";
    char blkdatname[512] = "c:¥¥blkdat¥0";
    char message[512];
    CFURCOV(ixdfilename,blkdatname,message);
    return;
}
```

7.10.2 索引ファイル簡易復旧関数（CFURCOVS）

ここでは、索引ファイル簡易復旧関数について説明します。

機能

索引ファイル簡易復旧関数は、再度使用できない状態にある索引ファイルを使用可能にするため、索引ファイル中のフラグをリセットします。索引ファイル復旧関数とは異なり、索引ファイル中の異常な箇所について修復しないため、復旧後の索引ファイルに対するアクセスで、データの矛盾でエラーになる場合があります。
再度使用できない状態にある索引ファイルを開いた場合は、入出力状態に“39”または“90”が返却されます。

記述形式

```
#include "f4agfutc.h"  /* 関数宣言 */
    signed __int64 CFURCOVS( char *ixdfilename,  char *message);
```

ixdfilename ： 復旧するファイルの名前
message　　： 索引ファイル復旧関数の結果（メッセージ）を格納する領域

関数名

CFURCOVS

第一パラメタ(ixdfilename)

復旧する索引ファイルの名前(文字列)が格納されている領域のアドレスを指定します。文字列の終端位置には、NULL文字(0x00)または空白文字(0x20)を設定する必要があります。また、ファイル名に空白またはコンマ(,)を含む場合は、ファイル名を二重引用符で囲む必要があります。

第二パラメタ(message)

索引ファイル簡易復旧関数の実行結果を示すメッセージを格納する領域のアドレスを指定します。格納する領域は、呼出し元で確保(512バイト必要)してください。
なお、メッセージが必要なければ、0を指定します。

復帰値

索引ファイル簡易復旧関数の結果として、メッセージに対応するコードを返却します。コードの値とメッセージの内容については、“表7.11 索引ファイル復旧関数(簡易復旧関数)が返却するメッセージの内容とコード”を参照してください。

例

```
#include "f4agfutc.h"
void callcobfrcovs(void)
{
    char ixdfilename[512] = "c:¥¥ixdfile¥0";
    char message[512];
    CFURCOVS(ixdfilename,message);
    return;
}
```

7.10.3 注意事項

ここでは、索引ファイルの復旧を行うために必要な作業について説明します。

作成時

Cプログラムから呼び出す場合には、以下のファイルをインクルードしてください。

- f4agfutc.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、F4AGFUTC.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。

f4agfutc.dll、f4agfuty.dll、f4agfrm.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=F4AGFUTC.DLL
CFURCOVS=F4AGFUTC.DLL
```

7.10.4 COBOLから呼び出す場合の使用例

ここで示す例は、オープンの状態により復旧を行うプログラムの例です。

例

例1)索引ファイル復旧関数に対して、復旧できなかったデータを格納するファイルおよびメッセージについて必要でない場合の呼出しを行っています。

```
000010 IDENTIFICATION DIVISION.
000020  PROGRAM-ID.  SAGYOU1.
000030 ENVIRONMENT DIVISION.
000040  CONFIGURATION SECTION.
000050  SPECIAL-NAMES.
000060     ENVIRONMENT-NAME  ENV-NAME
000070     ENVIRONMENT-VALUE ENV-VALUE.
000080  INPUT-OUTPUT SECTION.
000090   FILE-CONTROL.
000100     SELECT IXDFILE ASSIGN TO FILE1
000110       ORGANIZATION IS INDEXED
000120       RECORD KEY   IS IXDFILE-RECKEY
000130       FILE STATUS  IS FSDATA.
000140 DATA DIVISION.
000150  FILE SECTION.
000160  FD  IXDFILE.
000170  01  IXDFILE-REC.
000180      03  IXDFILE-RECKEY PIC X(8).
000190      03  IXDFILE-DATA   PIC X(20).
000200  WORKING-STORAGE SECTION.
000210  77  IXDFILE-NAME PIC X(512).
000220  77  BLKFILE-NON  PIC S9(9) COMP-5 VALUE 0.
000230  77  MESSAGE-NON  PIC S9(9) COMP-5 VALUE 0.
000240  77  FSDATA       PIC XX.
000250  77  OPEN-FSDATA  PIC XX.
000260 PROCEDURE DIVISION.
000270     OPEN I-O IXDFILE.
000280     MOVE FSDATA TO OPEN-FSDATA.
000290     CLOSE IXDFILE.
000300         EVALUATE OPEN-FSDATA
000310             WHEN "00"
000320               GO TO 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.
                 :
000560     CLOSE IXDFILE.
000570 GYOUMU-TEISHI.
000580     EXIT PROGRAM.
```

7.10.5 メッセージの内容とコード

“表7.11 索引ファイル復旧関数(簡易復旧関数)が返却するメッセージの内容とコード”に、索引ファイルの復旧で出力されるメッセージの内容とコードについて示します。

表7.11 索引ファイル復旧関数(簡易復旧関数)が返却するメッセージの内容とコード

コード10進	メッセージの内容
0	レコードをn件復旧しました。(注1)
1	レコードをn件復旧しました。復旧できないレコードがn件存在しました。(注2)
10	復旧すべきファイルが存在しません。
11	復旧すべきファイルが索引ファイルではありません。
12	復旧すべきファイルへのアクセス権がありません。
13	復旧不可データ出力ファイルが既に存在します。(注2)
14	復旧すべきファイルを他プロセスがアクセスしています。
15	復旧すべきファイルを他プロセスがリカバリしています。
16	ファイルの書込み領域が不足しました。(注2)
128	メモリ領域が不足しました。
131	指定された索引ファイル中にレコードが存在しませんでした。(注2)
132	ファイル情報に矛盾がありました。
136	復旧すべきファイル名に誤りがあります。
137	復旧不可データ出力ファイル名に誤りがあります。(注2)

注1 ： 簡易復旧関数ではメッセージだけが返却されません。
注2 ： 簡易復旧関数では返却されません。

7.11 COBOLファイルアクセスルーチン

COBOLファイルアクセスルーチンは、C言語からCOBOLの各編成のファイルをアクセスするためのAPI(Application Program Interface)関数群です。COBOLファイルをアクセスする64ビット(x64)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.3 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。

注意

印刷ファイルで出力するデータは、表示用(USAGE IS DISPLAY)のデータ項目で定義されていなければなりません。データ中にバイナリの不正なコードが含まれる場合、正しく印字されないことがあります。

8.1.1 各印刷方法の概要

印刷ファイルには、FORMAT句なし印刷ファイルとFORMAT句付き印刷ファイルがあります。

FORMAT句なし印刷ファイルは、行単位のデータを印刷したり、行単位のデータをフォームオーバ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	○	○	○	×
	フォームオーバレイパターンと合成した印刷ができる	×	○	○	○ (注)
	帳票定義体を使った帳票印刷ができる	×	×	○	○
利点	プログラムの記述が簡単	◎	○	○	◎
	帳票形式の印刷が簡単	○	◎	◎	◎
	他システムで作成した既存の帳票定義体を使用できる	×	×	◎	◎
	プログラム中で各種印刷情報を指示することができる	×	◎	◎	○
用途	帳票を印刷する	○	◎	◎	◎

◎ ： 使用可能であり適している
○ ： 使用可能である
× ： 使用不可能

注：帳票定義体にオーバ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レイパターン名を指定した場合だけ印刷可能です。詳細は、“MeFtのオンラインマニュアル”を参照してください。

表8.2 関連製品

使用するファイルの種類	[1]	[2]	[3]	[4]
FORM	―	○(*1)	○	○
FORM オーバレイオプション	―	○	○	○
PowerFORM (*2)	―	○	○	○
MeFt	―	―	○	○
MeFt/Web (*3)	―	―	○	○
Interstage List Works (*4)	○	○	○	○
Interstage List Creator Enterprise Edition (*5)	―	―	○	○

*1 ： オーバレイを作成するために使用します。
*2 ： FORMに同梱されています。

*3 ： NetCOBOL Enterprise EditionまたはNetCOBOL Standard Edition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ListWORKSと記述します。

*5 ： PDF変換機能を使用して、帳票をPDFファイル出力する場合に必要となります。

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。

フォームオーバレイパターンについては“8.1.7 フォームオーバレイパターン”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。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。 ・ 日本語項目:12ポイントで印字します。 ・ 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。

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。
  - ー デバイスフォントがアウトラインフォントである場合
  - ー 環境変数情報に“MS 明朝”や“MS ゴシック”などのTrueTypeフォントを指定した場合
- 上記表の漢字ROMカートリッジは、7ポイント/9ポイント/12ポイントすべてのカートリッジを装着した場合を想定しています。
- 他社のプリンタ装置を使用する場合、デバイスフォントの扱いは各社プリンタ装置および各社プリンタドライバに依存します。プリンタ装置の取扱い説明書などを参照してください。

印字文字の書体

明朝体/明朝体半角/ゴシック体/ゴシック体半角/ゴシックDP/書体番号を指定できます。

指定方法

指定方法の詳細は、“COBOL文法書”を参照してください。

指定方法	書体
MODE-1/ MODE-2/ MODE-3	明朝体
呼び名と関連付ける機能名で指定	明朝体／ゴシック体 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。 ・ 日本語項目:明朝体 ・ 英数字項目:ゴシック体

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ー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ース名の指定がない場合は、書体番号の値にかかわらず、すべて明朝体(通常の書体)で印字されます。
フォントテーブルの詳細および指定方法については以下を参照してください。

- “C.2.42 @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.2.77 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.2.78 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にフォントフェー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ース名の文字書体で印字されます。フォントフェー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のオンラインマニュアル”を参照してください。

注意

“MS P明朝”や“MS Pゴシック”など、プロポーショナ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	
印字モード名(PRINTING MODE句のPITCH指定)	0.01～24.00cpiの文字間隔を0.01cpi単位で指定します。 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。 ・ 日本語項目:6.00cpi ・ 英数字項目:10.00cpi

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

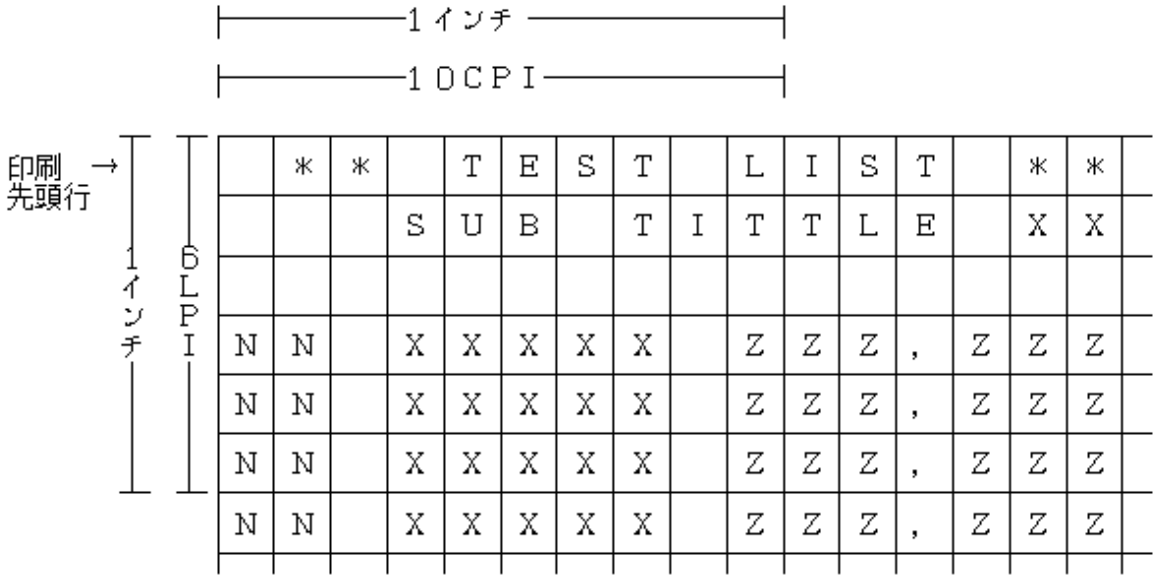
COBOLの帳票印刷には、行と桁の概念が不可欠です。特にFORMAT句なし印刷ファイルのように行レコード主体で帳票印刷を行う場合、実際にプログラムを作成する前にきめ細かい帳票設計が必要です。

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 CPI.
          :
DATA DIVISION.
          :
01 印刷レコード PIC X(80)
               CHARACTER TYPE IS PM1.
          :
PROCEDURE DIVISION.
          :
    OPEN OUTPUT 印刷ファイル.
    MOVE ″  **  TEST LIST  **″ TO 印刷データ.
    WRITE 印刷レコード FROM 印刷データ
                    AFTER ADVANCING PAGE.
           :
    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.2.44 @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。

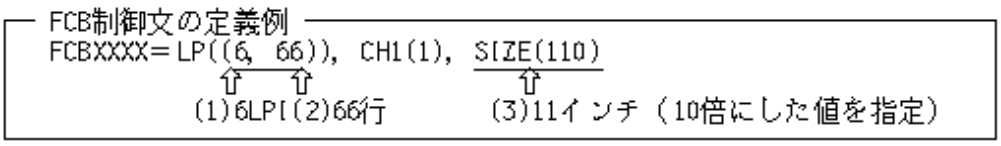
用紙内で印字可能な行数

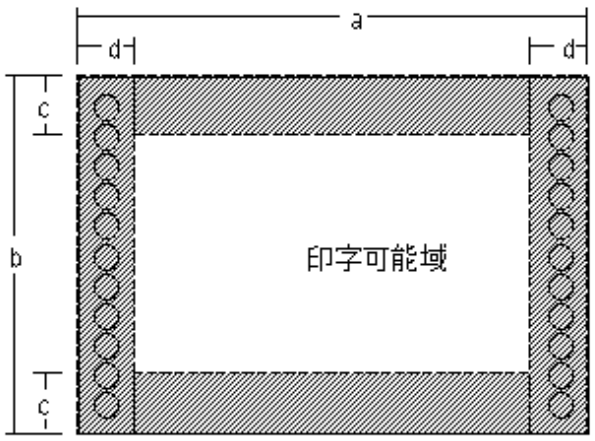
用紙内に印字可能な行数は、FCB制御文の定義により決定します。

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)





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い説明書などを参照し、印字不可能域を考慮した帳票設計を行う必要があります。このため、上記の印字可能文字数および行数はあくまでも目安であり、すべてがこの限りではありませんので注意してください。

8.1.7 フォームオーバレイパターン

フォームオーバレイパターンは、あらかじめ罫線や見出し文字など帳票の固定部分を設定するときに使用します。1ページ分の出力データとフォームオーバ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。

フォームオーバレイパターンは、FORMオーバ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、それぞれチャネル1からチャネル12に割り当てた行番号にスキップすることを意味します。通常、チャネル1(CHANNEL-01)は、印字可能な最初の行を識別するために使用されます。

SIZE情報

SIZEオペランドでは、帳票の縦方向のサイズを1/10インチ単位で指定します。指定可能な値は、35～159(3.5インチ～15.9インチ)です。省略した場合、110(11インチ)が有効になります。

FORM情報

FORMオペランドでは、帳票を定型サイズで指定します。定型サイズでは、PORT(ポートレート:向きが縦)とLAND(ランドスケープ:向きが横)の用紙の向きごとに、縦方向のサイズが一意に決定します。

デフォルトFCB名の指定方法については、“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。

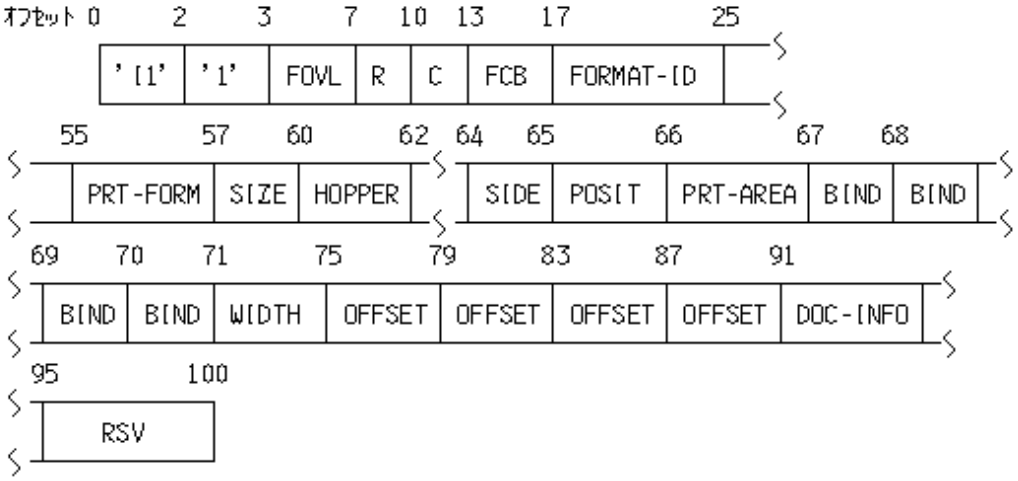
このように、COBOLプログラムで帳票印刷を行う場合、実際に使用する物理用紙サイズに対応したFCBの指定が必ず必要です。物理用紙サイズとFCBの指定が合致していない場合、意図した設計どおりの印刷結果が得られないことがありますので十分注意してください。

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るので、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 "I1".
    03  形式                   PIC X(1) VALUE "1" .
    03  オーバレイ名           PIC X(4).              → FOVL
    03  オーバレイ焼付け回数   PIC 9(3).              → R
    03  複写数                 PIC 9(3).              → C
    03  ＦＣＢ名               PIC X(4).              → FCB
    03  帳票定義体名           PIC X(8).              → FORMAT-ID
    03                          PIC X(30).
    03  印刷形式               PIC X(2).              → PRT-FORM
    03  用紙サイズ             PIC X(3).              → SIZE
    03  用紙供給口             PIC X(2).              → HOPPER
    03                          PIC X(2).
    03  印刷面                 PIC X.                 → SIDE
    03  印刷面位置付け         PIC X.                 → POSIT
    03  印字禁止域             PIC X.                 → PRT-AREA
    03  とじしろ方向.                                 → BIND
```

```
    04                  PIC X    OCCURS 4 TIMES.
  03  印刷位置情報.
    04  とじしろ幅       PIC 9(4).              → WIDTH
    04  印刷原点位置.
      05                PIC 9(4) OCCURS 4 TIMES.    → OFFSET
  03  文書名識別情報     PIC X(4).                   → DOC-INFO
  03                    PIC X(5) VALUE SPACE.       → RSV
```

以降の説明中で“ファイルオープン時の指定”とある場合、そのファイルがオープンされたときのシステムが持つ印刷環境のことを指します。

FORMAT句付き印刷ファイルの場合、帳票定義体の指定値が有効になります。帳票定義体の指定がない場合は、プリンタ情報ファイルに指定された情報が有効になります。プリンタ情報ファイルの詳細は、“MeFtのオンラインマニュアル”を参照してください。

FOVL (フォームオーバレイパターン名)

使用するフォームオーバレイパターン名を指定します。オーバレイグループが指定された場合、先頭のオーバレイパターンに対して単一オーバ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 (フォームオーバレイパターン焼付け回数)

フォームオーバレイパターンの焼付け回数(0～255)を指定します。
ただし、本システムでは複写数と同じ値が指定されたものと扱われます。

C (複写数)

ページ単位の複写数(0～255)を指定します。0を指定するとファイルオープン時の指定が有効となります。

注意

両面印刷指定時は、複写数の指定は有効となりません。1が指定されたものとみなします。また、複写数の指定はプリンタドライバが複写機能を持っている場合だけ有効となります。

FCB (FCB名)

FCB名を指定します。このフィールドが空白の場合、環境変数情報@DefaultFCB_Nameに指定されたFCB名が有効となります。なお、環境変数情報の指定については、“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。

注意

帳票定義体名の指定は、FORMAT句付き印刷ファイルの場合だけ有効となります。使用方法については、“8.4.2 プログラムの記述”を参照してください。

PRT-FORM (印刷形式)

印刷形式を指定します。設定可能な値を以下に示します。

- "P"（ポートレートモード）
- "L"（ランドスケープモード）
- "LP"（ラインプリンタモード）
- "PZ"（縮小印刷のポートレートモード）
- "LZ"（縮小印刷のランドスケープモード）

このフィールドが空白の場合、用紙サイズの指定(SIZE)にも空白が指定されていなければなりません。この場合、ファイルオープン時の指定が有効となります。

注意

- FORMAT句なし印刷ファイルの場合、"PZ"、"LZ"指定は、80%縮刷をサポートしているプリンタおよびプリンタドライバに対してだけ有効となります。80%縮刷をサポートしていないプリンタおよびプリンタドライバでは、"PZ"、"LZ"の縮小指定は無視され、"P"、"L"が指定されたものとみなします。また、"LP"指定は、連続用紙(ストックフォーム)サイズで設計された帳票を、A4用紙を横向きに使用したときのサイズに文字間および行間を縮小して印刷します。この場合、文字サイズ自身は縮小されませんので、文字の大きさや形態(平体や倍角など)によっては、隣接する文字同士が重なって印刷されることがあります。

  なお、"PZ"、"LZ"、"LP"指定で、フォームオーバレイは縮小されません。

- プリンタから供給される用紙の印刷形式(用紙の方向)は、本フィールドの指定により決定します。本フィールドの指定を省略した場合、以下の解釈になります。

  - FORMAT句なし印刷ファイル

    プリンタドライバでの設定値

  - FORMAT句付き印刷ファイル

    帳票定義体での設定値→プリンタ情報ファイルでの設定値

  なお、実際に使用される印刷形式に合わせて、FCB制御文を定義・指定する必要があります。詳細は、“8.1.8 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のオンラインマニュアル”を参照してください。

- プリンタから供給される用紙サイズ(用紙の種類)は、本フィールドの指定により決定します。本フィールドの指定を省略した場合、以下の解釈になります。

  - FORMAT句なし印刷ファイル

    プリンタドライバでの設定値

  - FORMAT句付き印刷ファイル

    帳票定義体での設定値→プリンタ情報ファイルでの設定値→プリンタドライバでの設定値

なお、実際に使用される用紙サイズに合わせて、FCB制御文を定義・指定する必要があります。詳細は、“8.1.8 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。

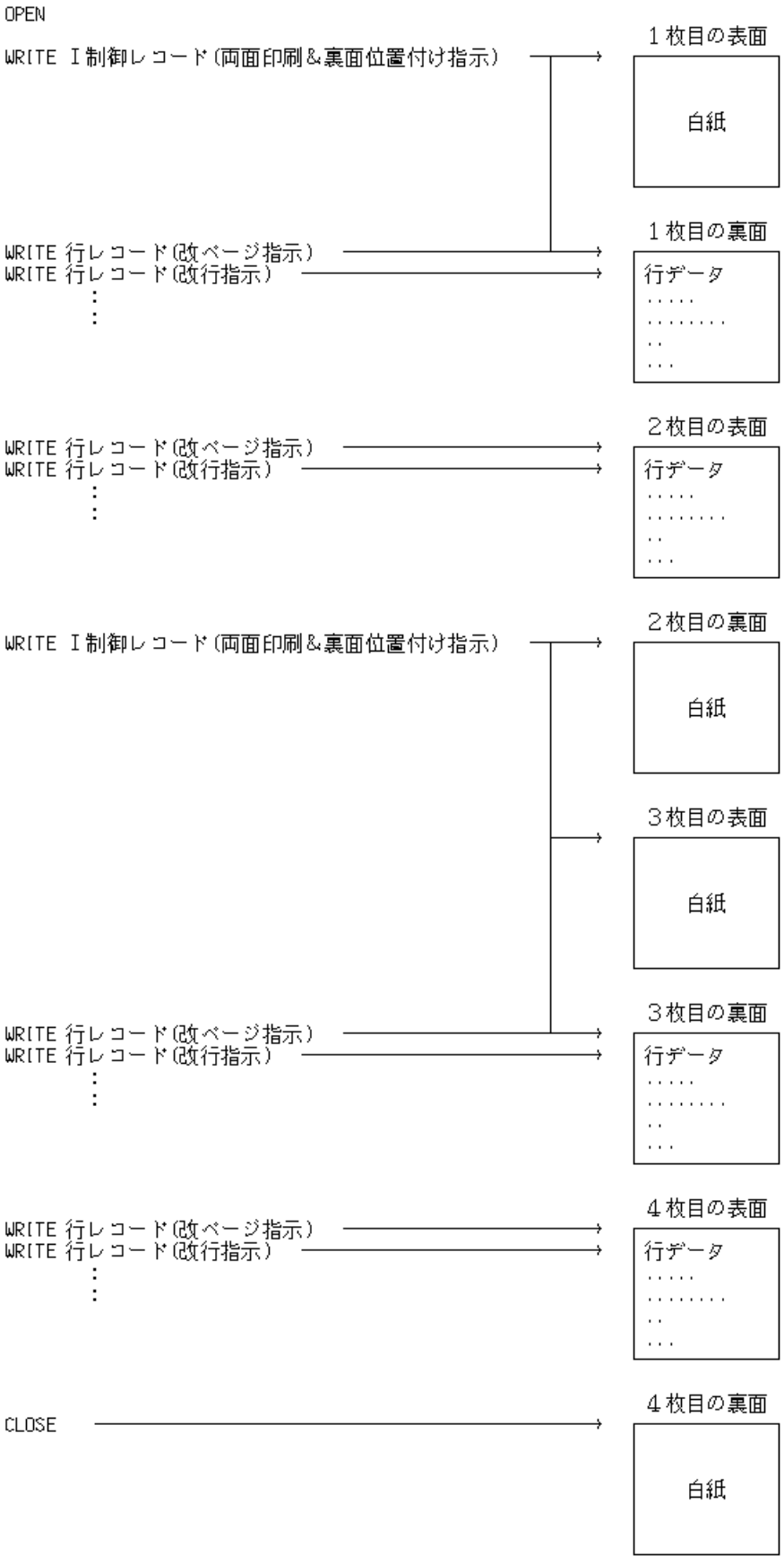
- "F"(表面位置付け)
- "B"(裏面位置付け)

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。

ー 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

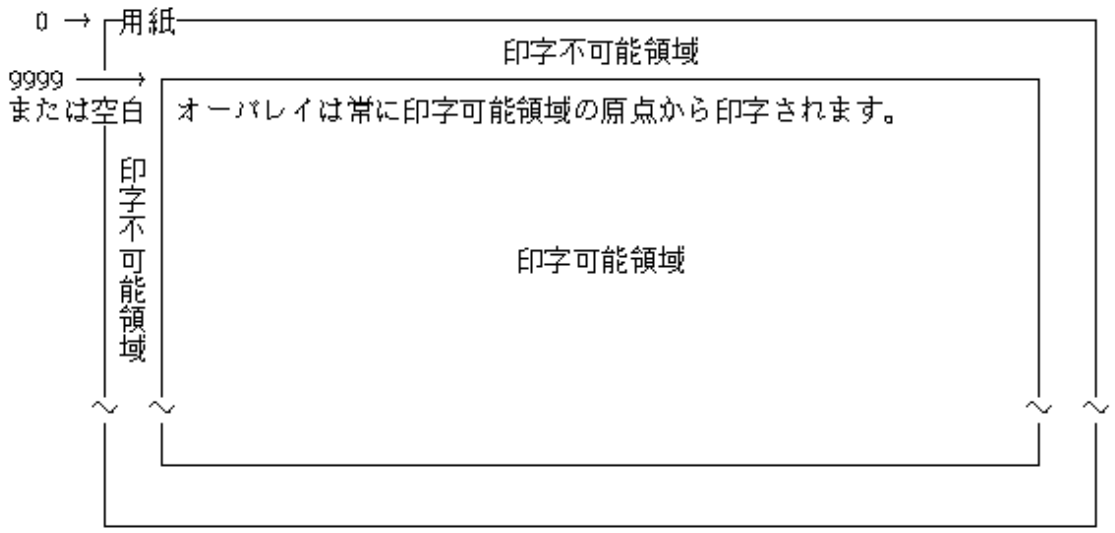
とじしろ方向の指定は、左とじ指定("L")および上とじ指定("U")だけ有効となります。したがって、右とじ指定("R")および下とじ指定("D")は無視されます。右方向および下方向にとじしろ幅を設ける場合、アプリケーションで意識する必要があります。

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。

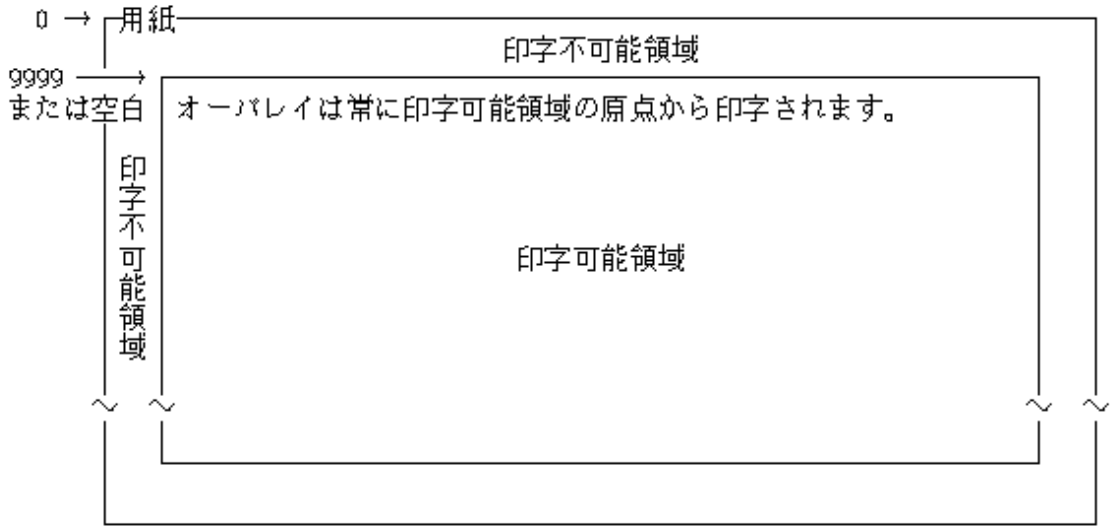
OFFSET (印刷原点位置)

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.2.24 @CBR_DocumentName_xxxx(I制御レコードによる文書名の指定)”を参照してください。

注意

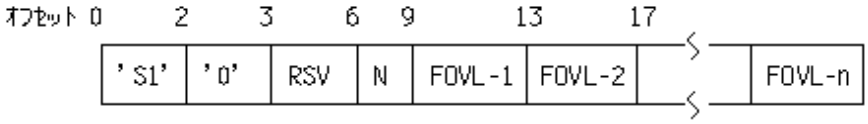
DOC-INFOフィールドの指定は、FORMAT句付き印刷ファイルには指定できません。FORMAT句付き印刷ファイルを利用して、プリントマネージャに任意の文書名を表示させる場合、本フィールドに空白を指定し、プリンタ情報ファイルで文書名を指定します。詳細は、“MeFtのオンラインマニュアル”を参照してください。

RSV (システム使用領域)

システムの使用する領域です。空白を設定しておきます。

S制御レコード

S制御レコードの形式を以下に示します。




```
01  S制御レコード.
    03  識別子              PIC X(2) VALUE "S1".
    03  形式                PIC X(1) VALUE "0" .
    03                       PIC X(3) VALUE SPACE.      → RSV
    03  個数                PIC 9(3).                  → N
    03  オーバレイ名１      PIC X(4).                  → FOVL-1
      :
    03  オーバレイ名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 (フォームオーバレイパターン名の個数)

指定するフォームオーバレイパターン名の数を指定します。

FOVL-n (フォームオーバレイパターン名)

フォームオーバ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レイパターン名は、次のI制御レコードまたはS制御レコードが出力されるまで有効です。

S制御レコードが有効となる範囲は、そのレコードが出力されてから、次のS制御レコードまたはI制御レコードが出力されるまでです。

注意

制御レコードの各フィールドに指定された値に誤りがあると、FILE STATUS句または誤り手続きの指定に関係なく、プログラムの実行を中断し、終了処理を行います。

8.1.10 帳票定義体

FORMを使って帳票を設計すると、帳票定義体が作成されます。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、フォームオーバレイパターンを取り込むこともできます。

帳票定義体を使って帳票の印刷を行う場合は、MeFtが必要です。MeFtを使用する場合、MeFtが使用するプリンタ情報ファイルが必要となります。プリンタ情報ファイルの詳細は、“MeFtのオンラインマニュアル”を参照してください。

帳票定義体の作成方法については“FORM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。

注意

NetCOBOLで使う帳票定義体の拡張子を除いたファイル名は、英字で始まる8文字以内の英数字となるようにしてください。

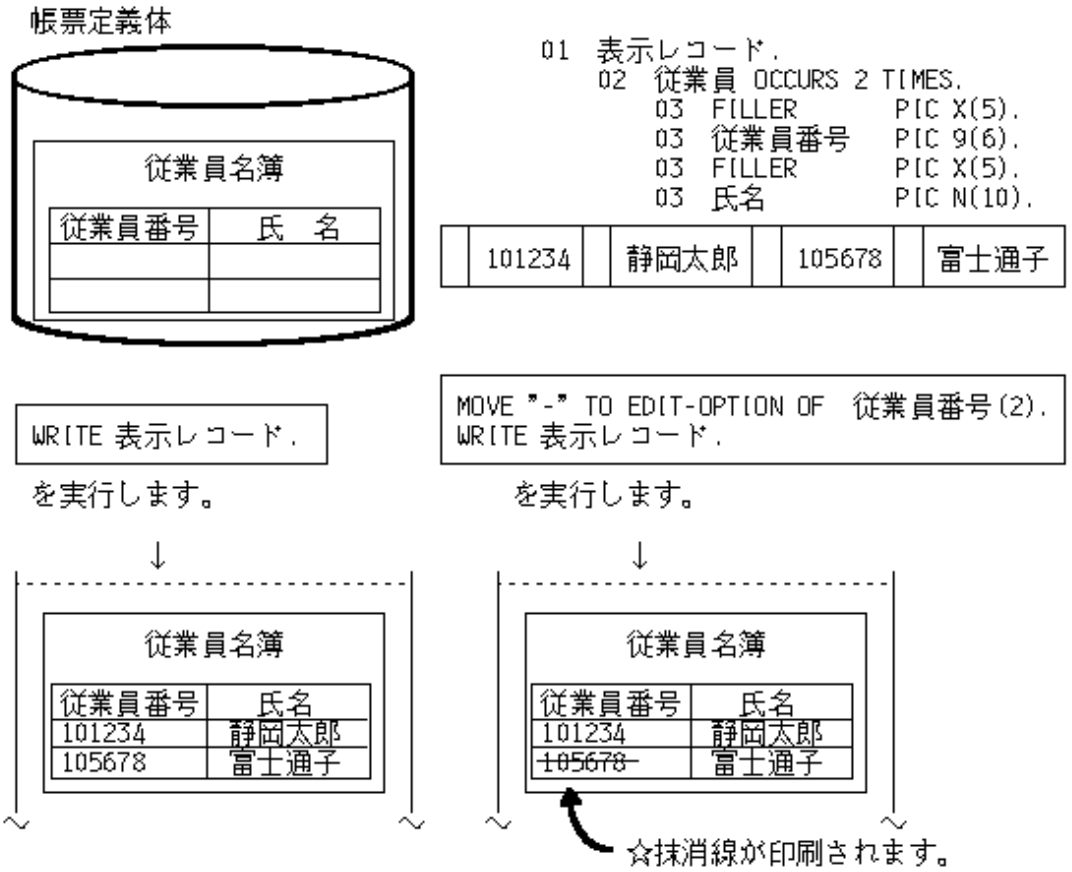
8.1.11 特殊レジスタ

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のオンラインマニュアル”を参照してください。



注意

項目制御部なしを指定した帳票定義体では、特殊レジスタを使用することはできません。
また、1つのプログラム中で項目制御部なしを指定した帳票定義体と項目制御部を指定した帳票定義体を混在して使うことはできません。

EDIT-OPTION2およびEDIT-OPTION3は5バイトの項目制御部を指定した帳票定義体のみで使用することができます。ただし、EDIT-OPTION2およびEDIT-OPTION3の指示を有効にするためには、MeFt V7.2以降が必要です。このとき、プリンタ情報ファイルに

PRTITEMCTL(項目制御部拡張指定)で拡張の指定が必要です。プリンタ情報ファイルの指定の詳細は“MeFtのオンラインマニュアル”を参照してください。

8.1.12 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。

注意

- 印刷情報ファイルに指定する絶対パスのファイル名および絶対パス名は、二重引用符(")で囲まないでください。
- 行の先頭に;(セミコロン)がある場合、セミコロンから改行までの間は、コメントとして認識されます。

書式および設定情報

```
--------------------------------------------------------------------------------------------------

[PrintInformation]                                          ← セクション名(固定、かつ必須)

    TextAlign=      { TOP    }                              ← 上端／下端合わせ指定(省略可能)
                    { BOTTOM }

    DocumentName = 文書名                                   ← 文書指定(省略可能)

    OverlayPrintOffset =  { VALID   }                       ← オーバレイ原点移動機能を有効または無効
                          { INVALID }                         にする指定(省略可能)

    STREAM=         { LW }                                  ← 帳票の出力方式(電子帳票出力／プリンタ
                    { PR }                                    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:                }                ← 出力プリンタ名を指定(省略可能)
                    { COM:                 }
                    { PRTNAME:プリンタ名   }

    FOVLDIR=オーバレイファイルのフォルダ名                  ← フォームオーバレイパターンファイルを格納
                                                              したフォルダ名(省略可能)

    FOVLTYPE=オーバレイファイルの形式                       ← フォームオーバレイパターンファイルのプレ
                                                              フィクス(形式)名(省略可能)

    FOVLNAME=オーバレイファイル名                           ← フォームオーバレイパターンファイルのファ
                                                              イル名(省略可能)

    OVD_SUFFIX=オーバレイファイル(拡張子)名                 ← フォームオーバレイパターンファイルのサ
                                                              フィクス(拡張子)名(省略可能)

--------------------------------------------------------------------------------------------------
```

セクション名(必須)

セクション名“[PrintInformation]”は、印刷情報ファイルを識別するための名前で、常に固定です。

TextAlign

行内に配置する印字文字の位置を指定します。印字する文字を文字セルの左上端を基準点として行内に上端合わせで配置する(TOP)か、左下端を基準点として行内に下端合わせで配置する(BOTTOM)かを指定します。なお、本指定は、環境変数情報@CBR_TextAlignの指定よりも優先されます。[参照]“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.2.24 @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.2.40 @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レイに対して有効または無効にする指定）”

参考

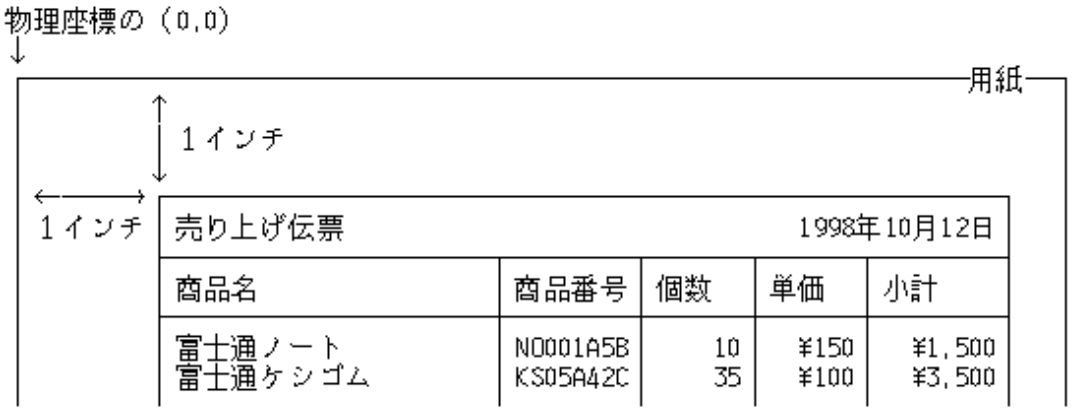
本環境変数情報にVALIDを指定することで、I制御レコードの同機能を行レコードおよびフォームオーバレイの両方に対して有効にすることができます。
以下に、原点オフセットを設けない通常の印刷結果と、上方向および左方向にそれぞれ1インチの原点オフセットを設けた場合の印刷結果を例にとり、VALID/INVALID指定時(または省略時)の印刷結果の相違について図示します。

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。

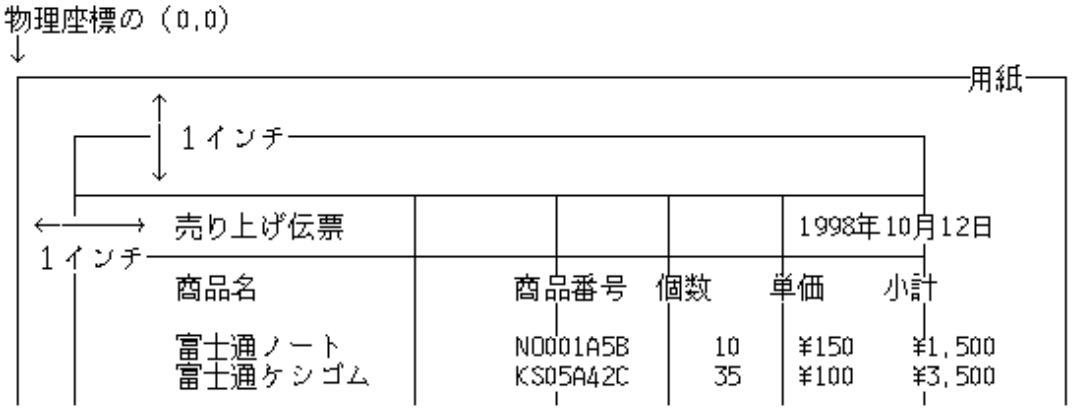
－ 印刷原点位置を指定しない場合の印刷結果

ー X,Y座標にそれぞれ1インチの原点オフセットを設けた場合の印刷結果

- VALID指定の場合



- INVALID指定の場合



STREAM

FORMAT句なし印刷ファイルを使用して出力する帳票の出力方式を指定します。通常のプリンタ出力(紙印刷)を行う場合“PR”を指定します。ListWORKSと連携して電子帳票出力を行う場合“LW”を指定します。本指定を省略した場合、“PR”が指定されたものとみなされます。
なお、電子帳票出力を行う場合、ListWORKSが必要になります。電子帳票出力の詳細は、“8.6 電子帳票出力機能を使う方法”、“ListWORKSのオンラインマニュアル”および“ListWORKSのヘルプ”を参照してください。

STREAMENV

ListWORKSと連携して電子帳票出力を行う場合(STREAM=LWを指定した場合)、電子帳票出力に関する静的な情報を定義した電子帳票情報ファイルのファイル名を絶対パスで指定します。電子帳票情報ファイルの指定は、省略することも可能です。省略した場合、ListWORKSが保持する省略時の解釈に従って動作します。
なお、電子帳票情報ファイルの定義は、ListWORKSによって規定されています。詳しくは、“ListWORKSのオンラインマニュアル”および“ListWORKSのヘルプ”を参照してください。

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。

ー 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.2.78 ファイル識別名（プログラムで使用するプリンタおよび各種パラメタの指定）”を参照してください。

なお、ListWORKS連携により電子帳票出力機能を使用する場合、“LPTn:”および“COMn:”は指定できません。この場合、“PRTNAME:プリンタ名”を使用し、プリンタ名には、ListWORKSで電子化・登録するときに使用するListWORKS仮想プリンタ名またはデータ転送コネクタ名（以降、総称して電子保存装置と呼びます）を指定する必要があります。詳細は、“8.6 電子帳票出力機能を使う方法”を参照してください。

FOVLDIR

フォームオーバ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.2.80 FOVLDIR（フォームオーバレイパターンのフォルダの指定）”の指定よりも優先されます。

FOVLTYPE

フォームオーバレイパターンのファイル名の先頭4文字が“KOL5”(省略値)以外の場合、形式に、ファイル名の先頭4文字を指定します。ファイル名に形式を持たない場合は、文字列“None”を指定してください。

例

フォームオーバレイパターンファイルが“C:¥FOVLDATA¥KOL2OVD1.OVD”の場合

```
FOVLDIR=C:¥FOVLDATA
FOVLTYPE=KOL2
FOVLNAME=OVD1 または I/S制御のFOVLフィールドで “OVD1” を指定してWRITE
```

なお、本指定は、実行環境変数“C.2.81 FOVLTYPE（フォームオーバレイパターンのファイル名の形式の指定）”の指定よりも優先されます。

FOVLNAME

フォームオーバ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。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

フォームオーバレイパターンファイルが“C:¥FOVLDATA¥KOL2TEST.OVL”の場合

```
FOVLDIR=C:¥FOVLDATA
FOVLTYPE=KOL2
FOVLNAME=TEST かつ （I/S制御レコードのWRITEなし
              または I/S制御のFOVLフィールドに空白を指定してWRITE）
OVD_SUFFIX=OVL
```

なお、本指定は、実行環境変数“C.2.82 FOVLNAME（フォームオーバレイパターンのファイル名の指定）”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

フォームオーバレイパターンファイルが“C:¥FOVLDATA¥KOL5ABCD”の場合

```
FOVLDIR=C:¥FOVLDATA
FOVLTYPE=KOL5
FOVLNAME=ABCD または I/S制御のFOVLフィールドで “ABCD” を指定してWRITE
OVD_SUFFIX=None
```

なお、本指定は、実行環境変数“C.2.83 OVD_SUFFIX(フォームオーバレイパターンのファイルの拡張子の指定)”の指定よりも優先されます。

注意

FOVLDIR、FOVLTYPE、FOVLNAMEおよびOVD_SUFFIXの指定は、PrintWalker/OVLオプションによるフォームオーバレイ印刷およびListWORKS連携によるフォームオーバ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

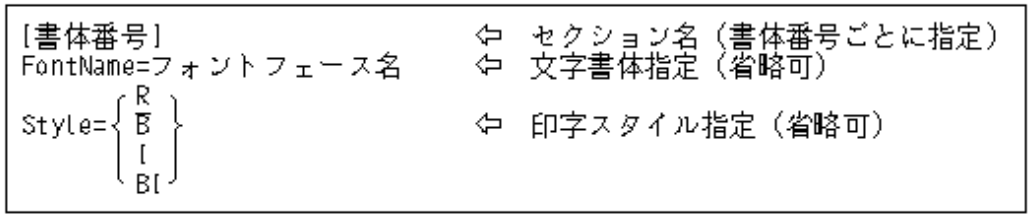
フォントテーブルとは、印刷ファイルを利用して帳票出力を行うときの書体情報を定義するテキスト形式のファイルのことです。

書体情報には、印字する文字のフォントフェー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



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

フォントフェース名には、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(通常の書体)で印字されます。

注意

- 印字に使用するフォントフェース名は、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。
- フォントフェース名は、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.2.42 @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.2.77 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.2.78 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 NC"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 NC"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

ー FORMAT句なし印刷ファイルの場合

- ASSIGN句の指定

  SELECT 印刷ファイル ASSIGN TO S-PRTF.

- 環境変数情報の指定

  PRTF=PRTNAME:FUJITSU VSP4620A

ー FORMAT句付き印刷ファイルの場合

- プリンタ情報ファイルの指定

  PRTDRV FUJITSU VSP4620A

サービスがシステムアカウントまたは特定のユーザアカウントでシステムに乗り込むかは、各サービスの仕様を確認してください。

なお、サービスによっては、コントロールパネルのサービスのスタートアップの設定により、特定のユーザアカウントでログインするように設定できる場合があります。この場合、使用するプリンタドライバをインストールしたときのユーザアカウントを設定することで、印刷できることがあります。

・ サービス配下で印刷ファイルを使用する場合、プリンタ用紙サイズは、以下に設定した情報が有効になります。

ー サービス配下の場合は、プリンタのプロパティに設定された情報

ー 通常のアプリ動作の場合は、プリンタの印刷設定に設定された情報

8.2 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します。
	ASSIGN句	ファイル参照子	PRINTER 、PRINTER-n 、アクセス名、ローカルプリンタポート名(LPTn:) 、シリアルポート名(COMn:) またはプリンタ名を指定します。PRINTER を指定すると、通常使うプリンタに設定されているプリンタに出力されます。
任意	ORGANIZATION句	ファイル編成を示す文字列	SEQUENTIALを指定します。
	FILE STATUS句	データ名	作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、入出力処理の実行結果が設定されます。(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

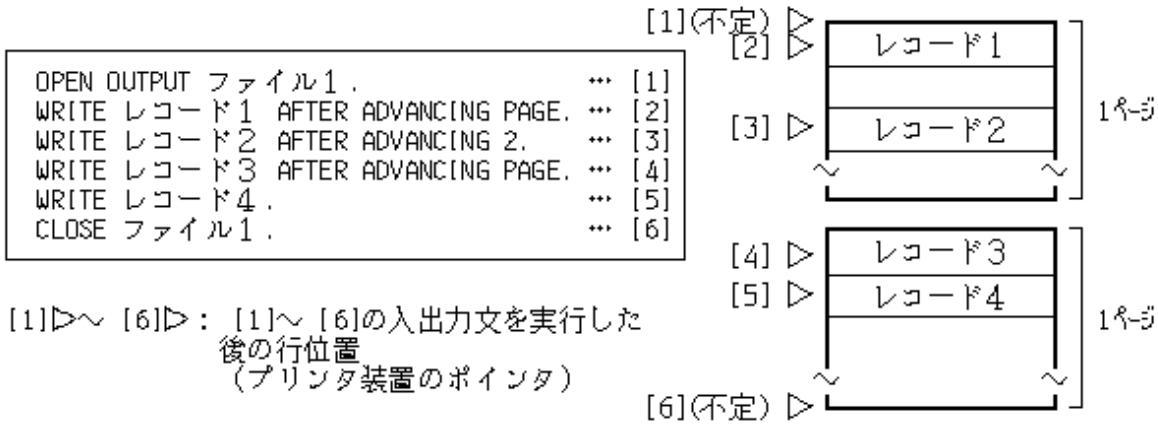
OPEN文およびCLOSE文

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

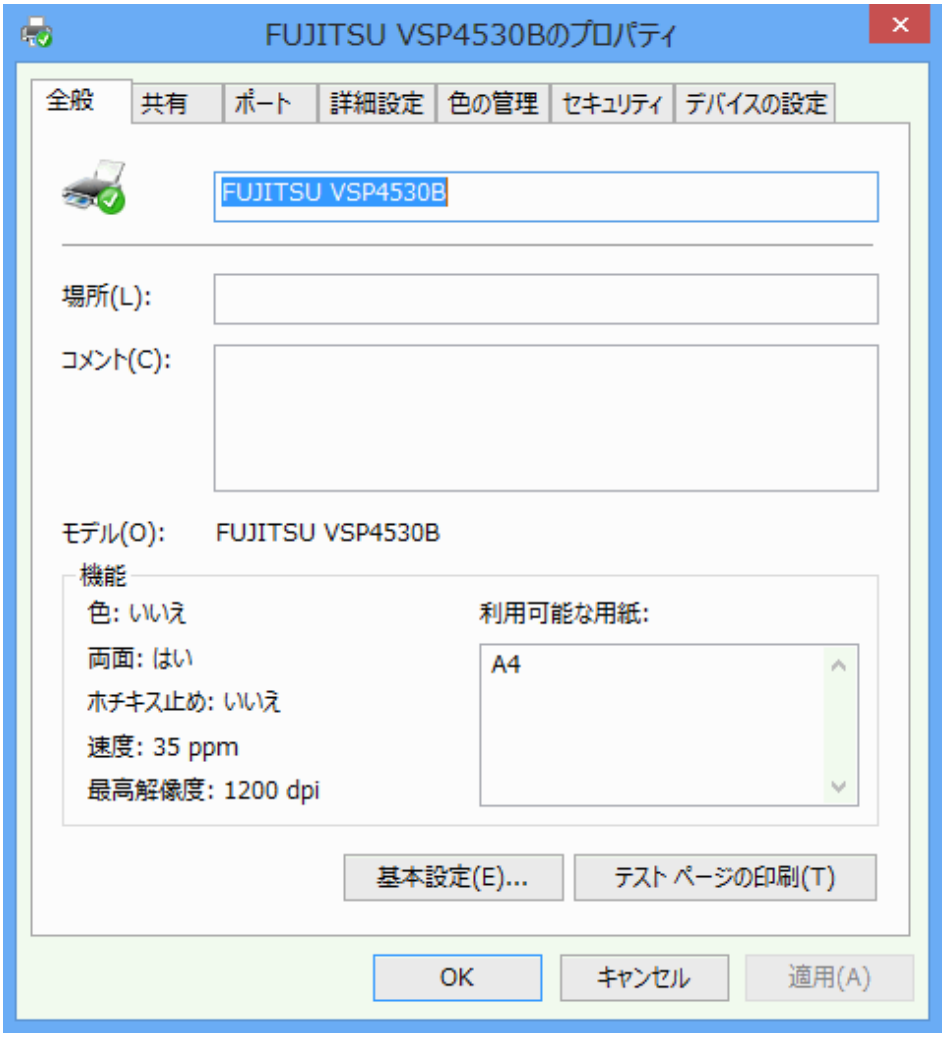
印刷装置の指定は、以下の2種類の方法で設定することができます。

- 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 VSP4620A
```

ネットワーク上のプリンタに接続している場合は、プリンタの名前に¥¥サーバ名¥の指定が必要です。

例

```
PRTNAME:¥¥PRTSVR¥FUJITSU VSP4620A
```

参考

プリンタ名の取得は、実行環境設定ツールの〔環境設定〕メニューから“プリンタ名”を選択することによって表示される〔プリンタ名の選択〕ダイアログを利用すると便利です。

注意

プリンタのプロパティにネットワークプリンタの名前は表示されませんので、必ず〔プリンタ名の選択〕ダイアログで指定するようにしてください。

ローカルプリンタポート名/シリアルポート名

〔プロパティ〕ダイアログの〔ポート〕シートの印刷先のポートのリストボックスに表示されている名前を指定します。

例

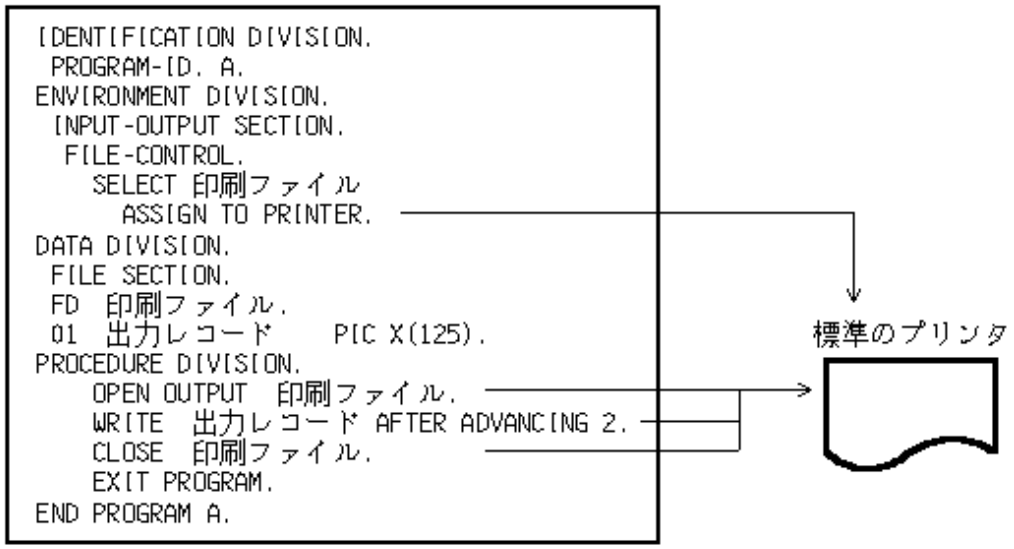
"LPT1:"

プログラム例

ASSIGN句に文字列PRINTERを記述した場合

出力先は、標準のプリンタになります。

標準のプリンタとは、プリンタの設定で“通常使うプリンタに設定”がチェックされているプリンタのことです。



なお、特定のポートに接続されているプリンタ装置に出力する場合は、以降で説明する方法で行ってください。

ASSIGN句にファイル識別名を記述した場合

ファイル識別名を環境変数情報名として、出力先のプリンタ名またはローカルプリンタポート名/シリアルポート名を設定します。環境変数情報の設定方法は、“5.3 実行環境情報の設定”を参照してください。

なお、ファイル識別名にプリンタが割り当てられていない場合、ファイルの割当てエラーとなります。

図8.1 プリンタ名を指定した場合の例

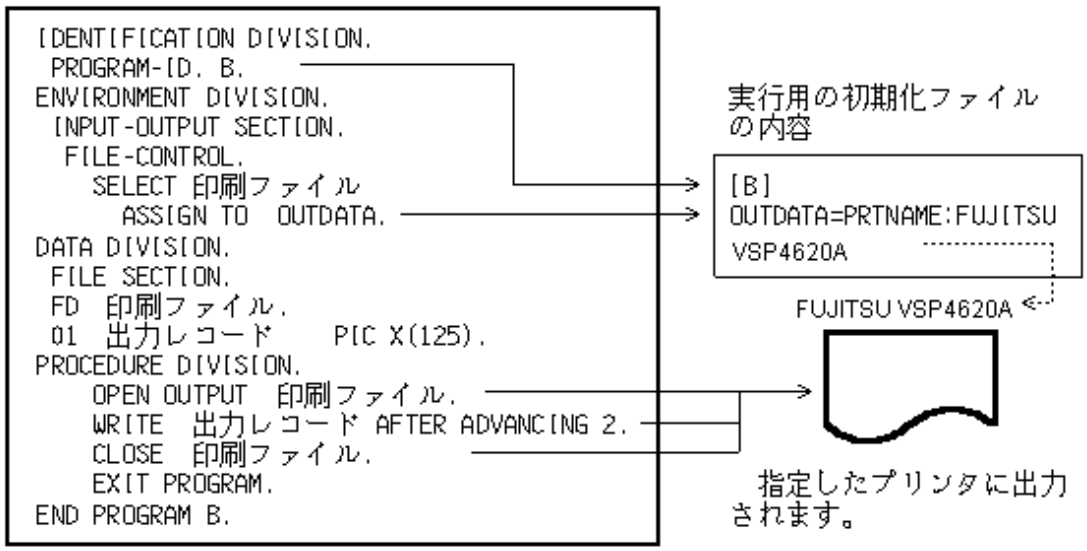
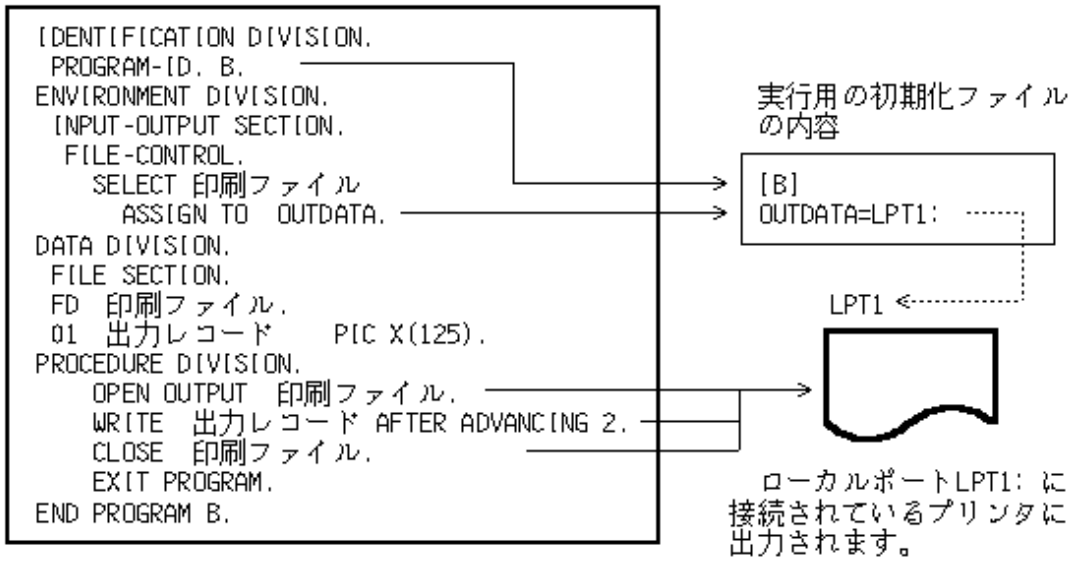


図8.2 ローカルポート名を指定した場合の例



ASSIGN句にファイル識別名定数を記述した場合

ファイル識別名定数に直接記述されたプリンタまたはファイル識別名定数に記述したローカルポート/シリアルポートに接続されているプリンタが出力先になります。

図8.3 プリンタ名を指定した場合の例

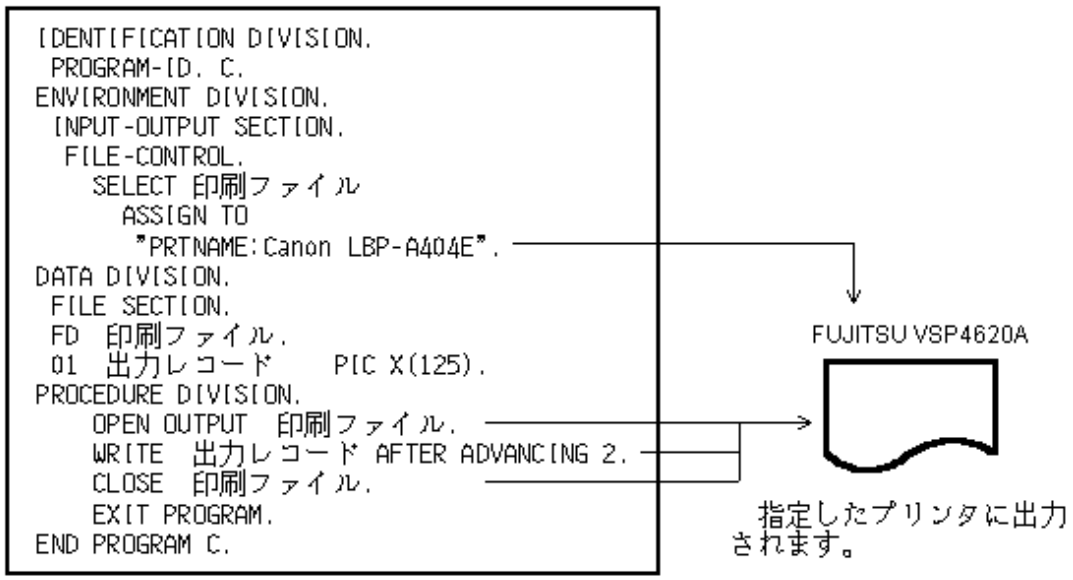
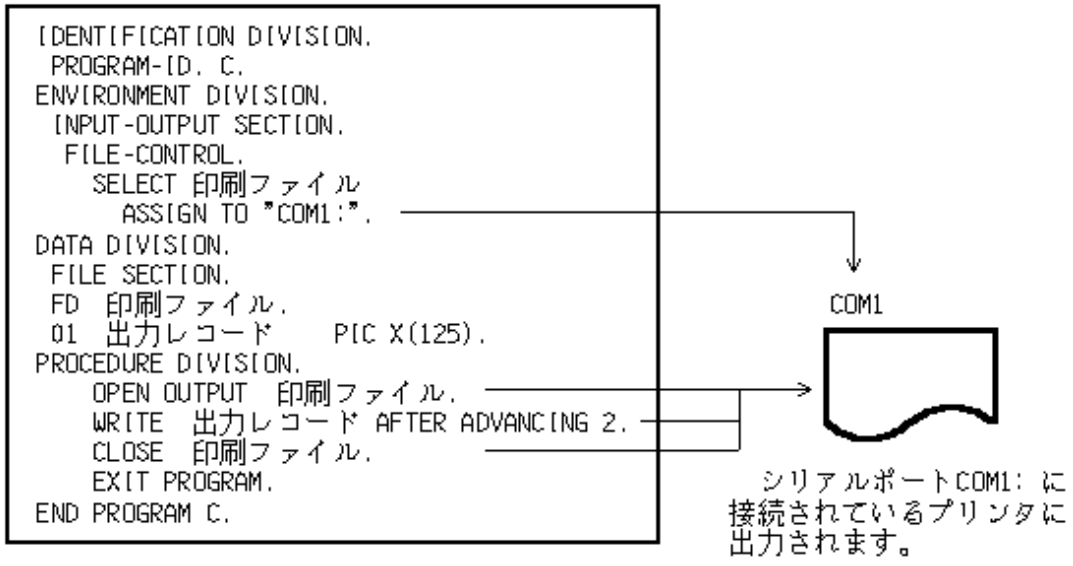


図8.4 シリアルポート名を指定した場合の例



ASSIGN句にデータ名を記述した場合

データ名に直接記述されたプリンタまたはデータ名に設定されたローカルポート/シリアルポートに接続されているプリンタが出力先になります。データ名の内容が空白の場合、ファイルの割当てエラーとなります。

図8.5 プリンタ名を指定した場合の例

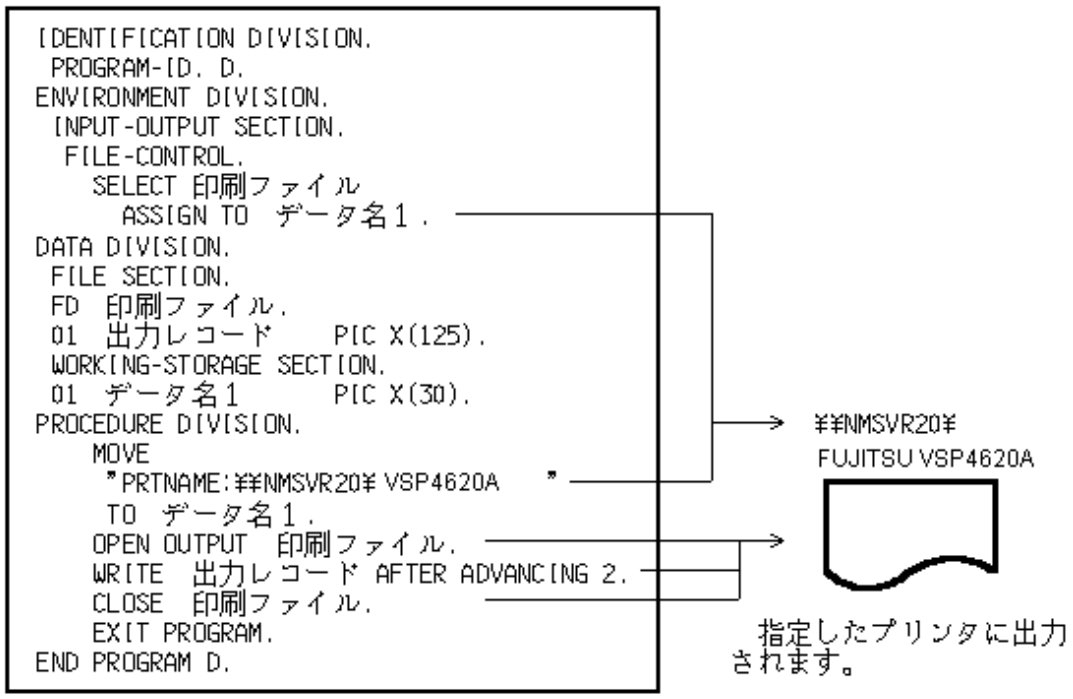
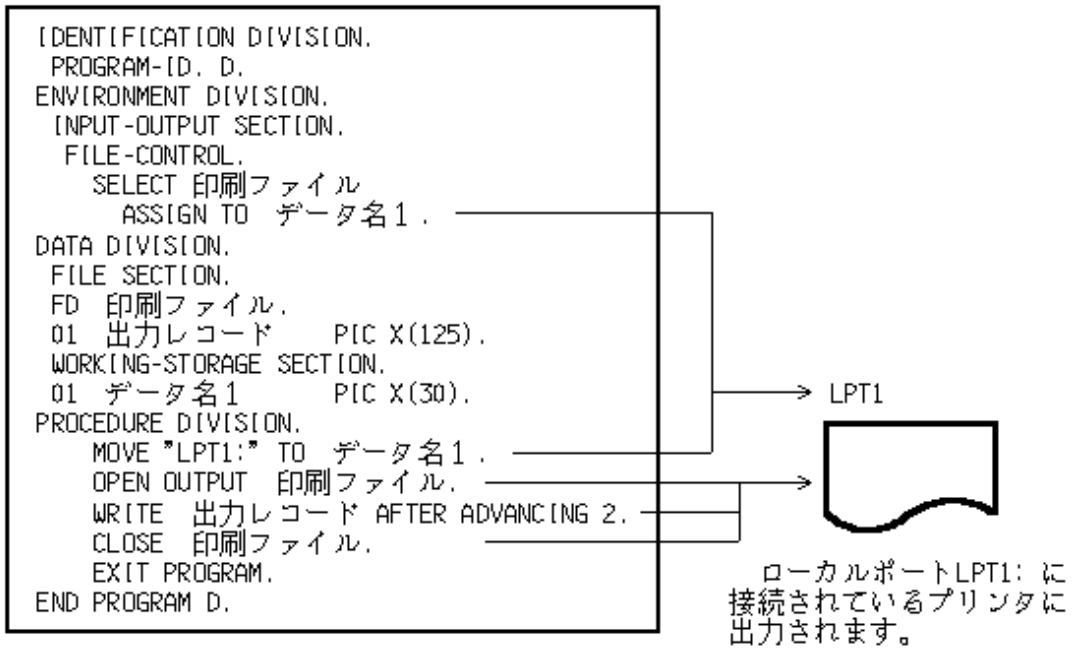


図8.6 ローカルポート名を指定した場合の例



ASSIGN句に文字列PRINTER-nを記述した場合

ファイル識別名を記述した場合と同様に、文字列PRINTER-n(n=1～9)を環境変数情報名として、出力先のプリンタ名またはローカルプリンタポート名/シリアルポート名を設定します。

環境変数情報の設定方法については、“5.3 実行環境情報の設定”を参照してください。

なお、PRINTER-nにプリンタが割り当てられていない場合、ファイルの割当てエラーとなります。以下に、実行用の初期化ファイルを使った場合の例を示します。

図8.7 プリンタ名を指定した場合の例

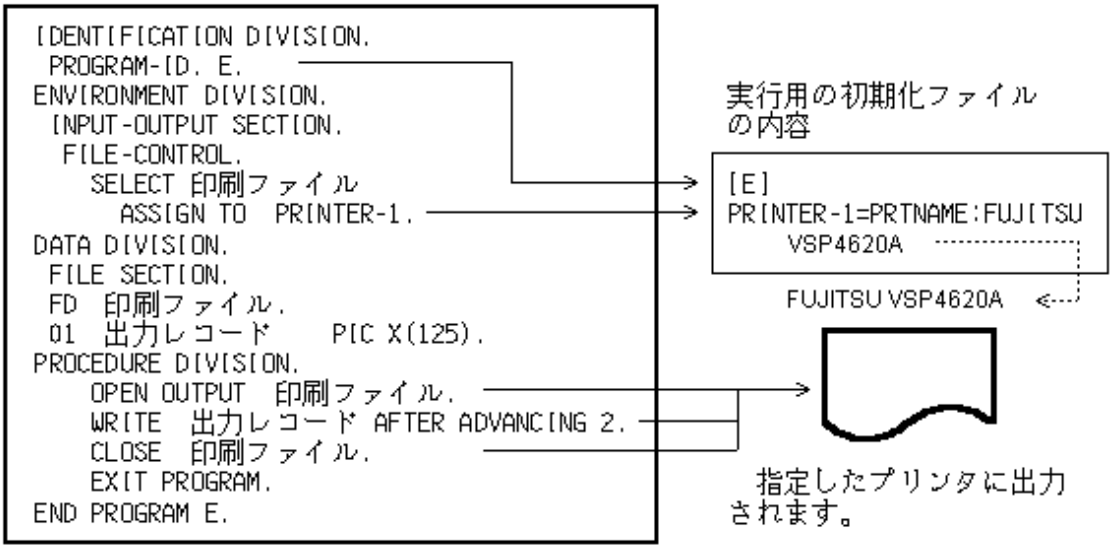
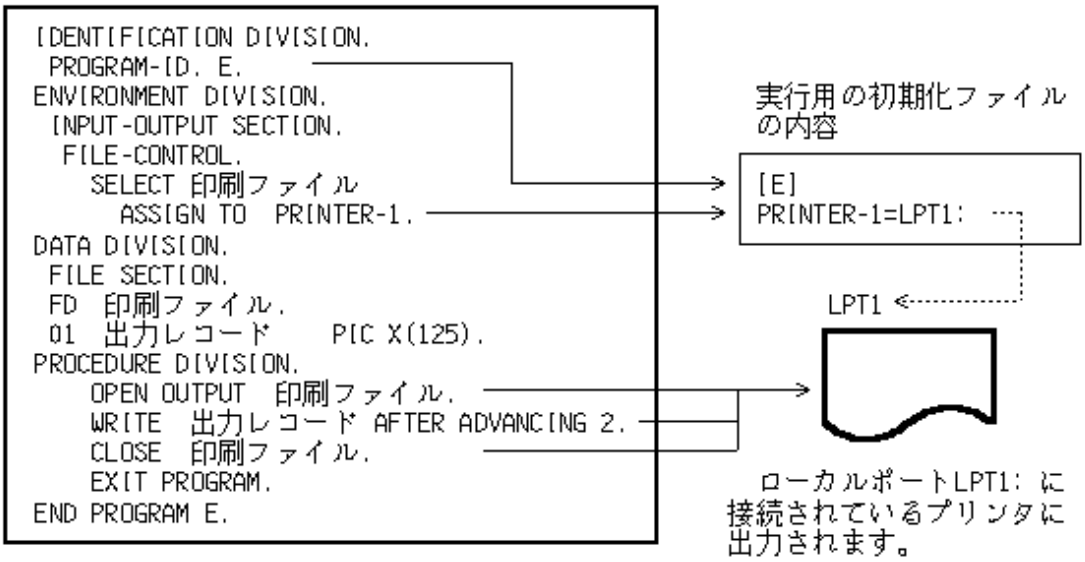


図8.8 ローカルポート名を指定した場合の例

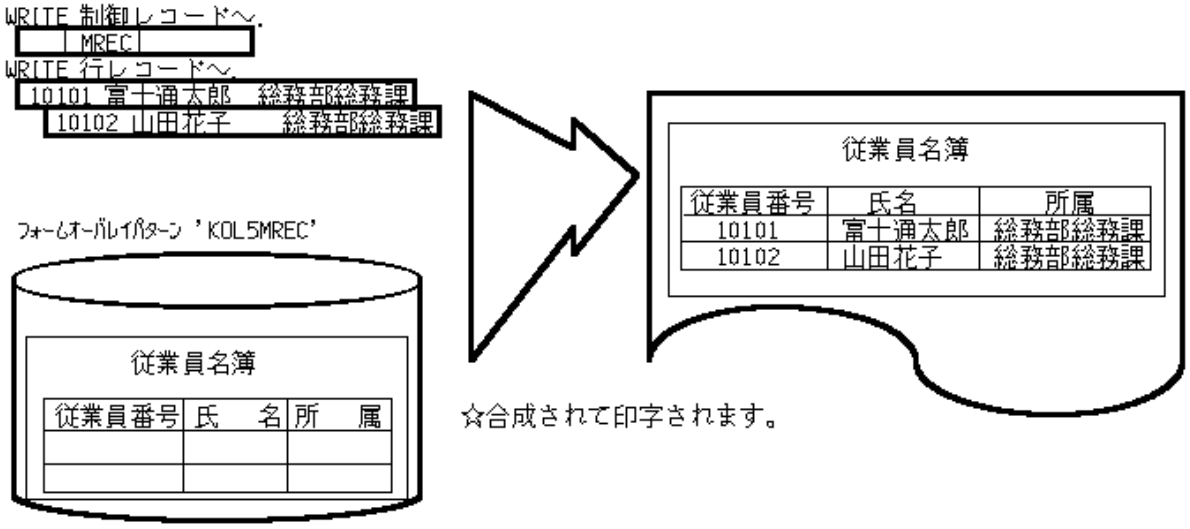


8.3 フォームオーバレイおよびFCBを使う方法

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

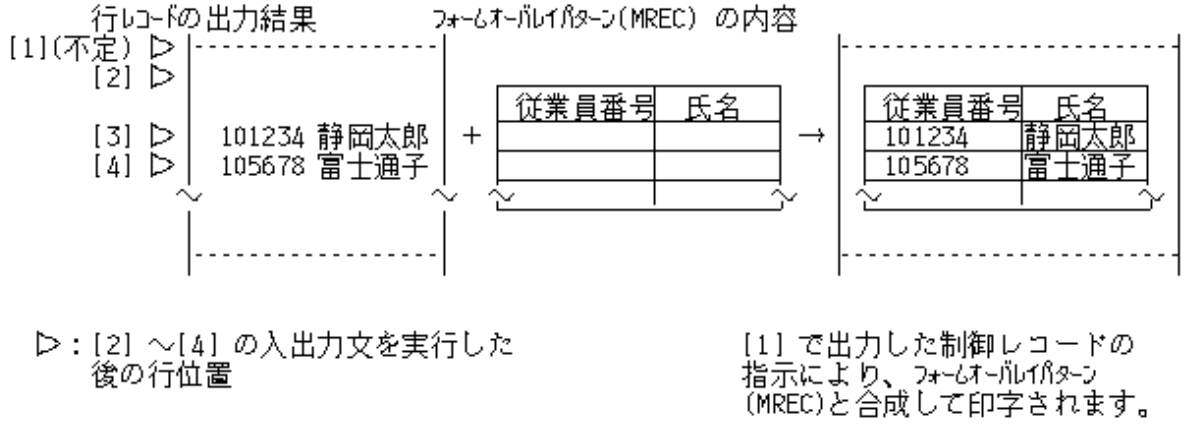
OPEN文およびCLOSE文

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レイパターンと合成して印字するには、フォームオーバ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、フォームオーバレイパターン名に空白を設定した制御レコードを出力します。制御レコードを出力すると、その後に出力するページの形式が制御レコードの内容のとおり設定されます。ただし、制御レコードを出力すると、現在のページには行レコードを出力できなくなるので、制御レコード出力直後の行レコードの出力には、AFTER ADVANCING PAGEを指定する必要があります。

```
    :
FILE SECTION.
FD  ファイル１.
    :
01  制御レコード.
    :
    02  FOVL       PIC X(4).
    :
    MOVE  "MREC"    TO FOVL.
    WRITE  制御レコード AFTER ADVANCING  呼び名.          … [1]
    MOVE  SPACE TO   行レコード.
    WRITE  行レコード   AFTER ADVANCING  PAGE.           … [2]
    MOVE  101234    TO   従業員番号.
    MOVE  NC"静岡太郎" TO  氏名.
    WRITE  行レコード   AFTER ADVANCING  2.              … [3]
    MOVE  105678    TO   従業員番号.
    MOVE  NC"富士通子" TO  氏名.
    WRITE  行レコード   AFTER ADVANCING  1.              … [4]
        :
```



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、フォームオーバレイパターンの焼付けは行われません。[参照]“C.2.80 FOVLDIR（フォームオーバ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.2.83 OVD_SUFFIX(フォームオーバ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.2.81 FOVLTYPE(フォームオーバ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。

フォームオーバレイパターンの出力

フォームオーバ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

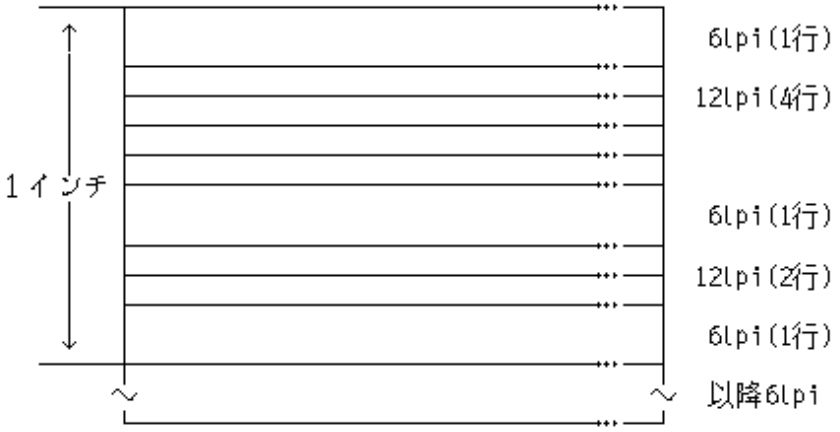
FCB制御文の例

- 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

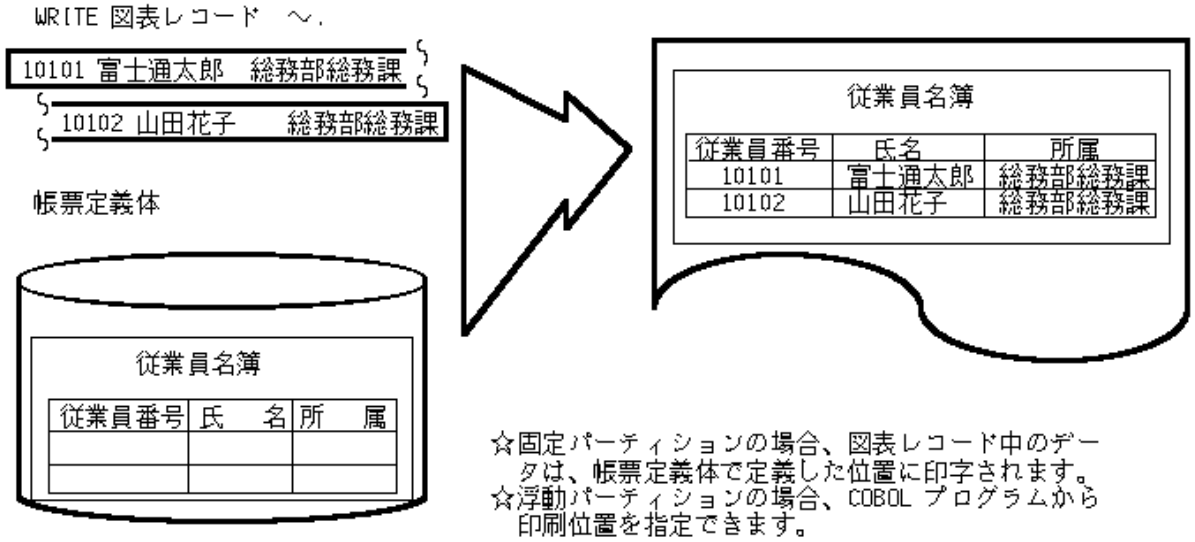
ここでは、FORMAT句付き印刷ファイルでパーティション形式の帳票定義体を使う方法について説明します。

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、通常の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。



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

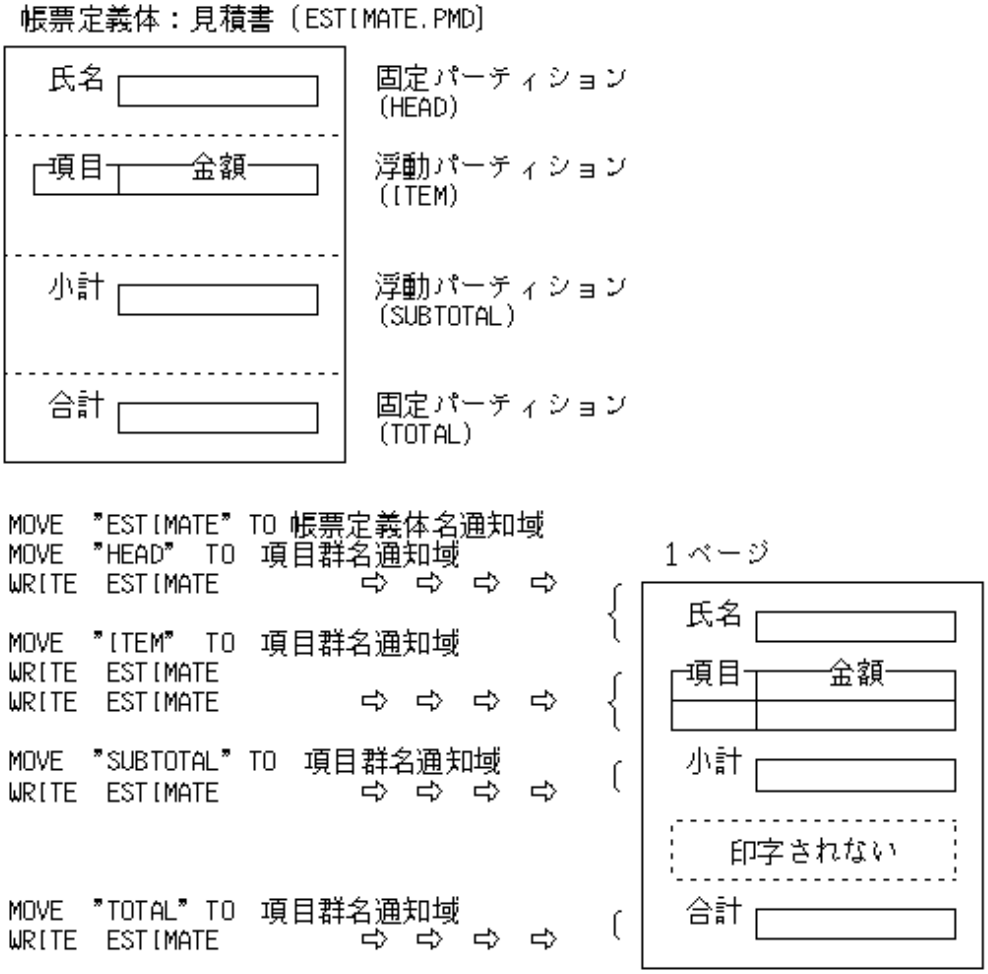
固定パーティションとはページ内での印刷位置が固定のパーティションで、帳票定義体で指定された位置に印刷されます。

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。



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。

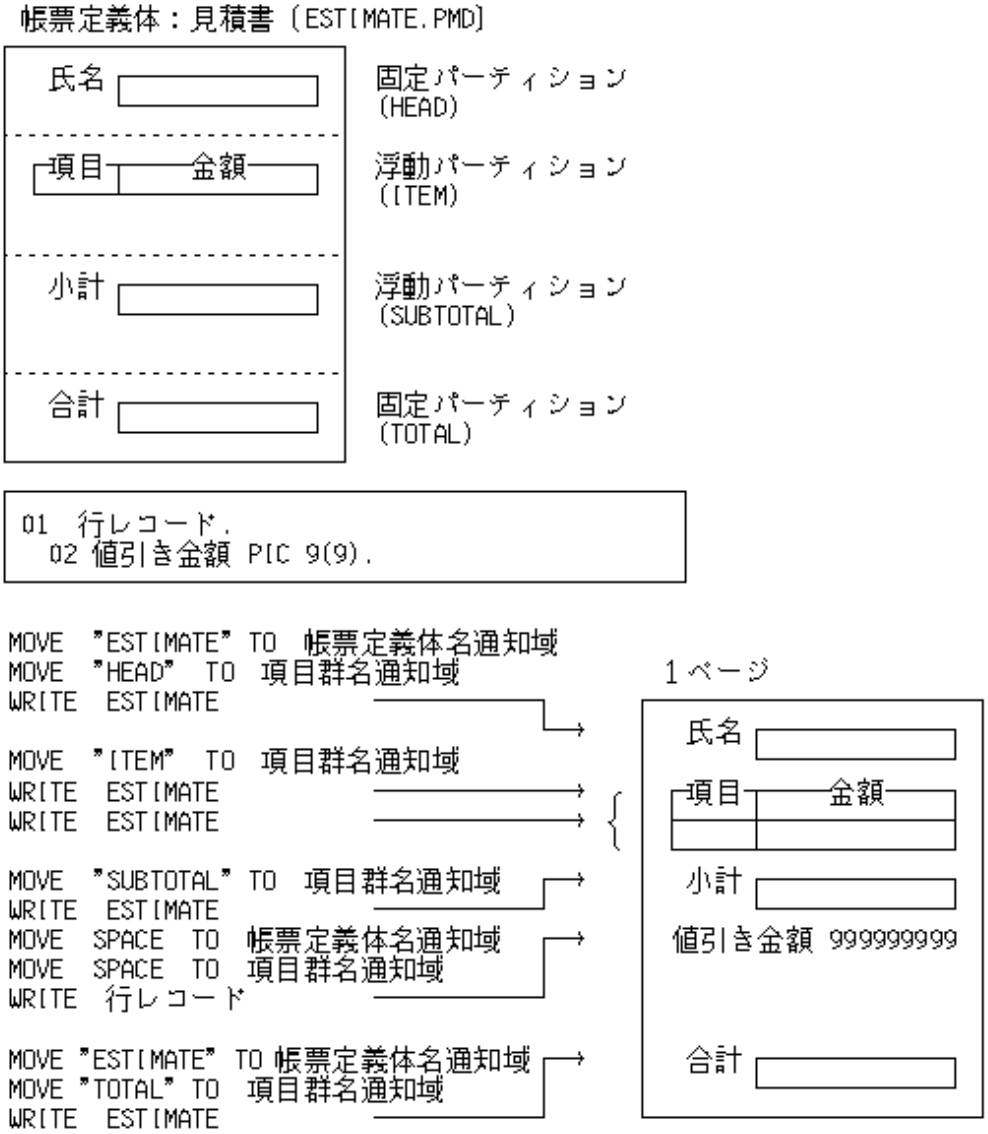
パーティションと行レコードの組合せ

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。



注意

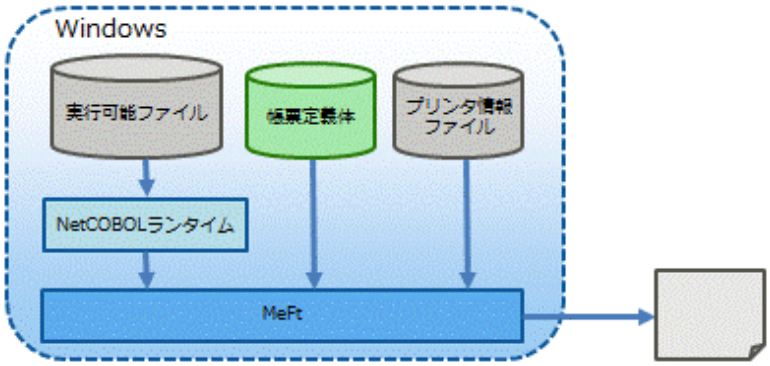
行レコードまたは浮動パーティションを固定パーティションの印刷位置に出力した場合には、その位置に印刷する固定パーティションは、そのページ中では印刷できません。そのページは改ページされて、固定パーティションは次のページの印刷位置に印刷されます。

8.4.1.2 帳票の電子化

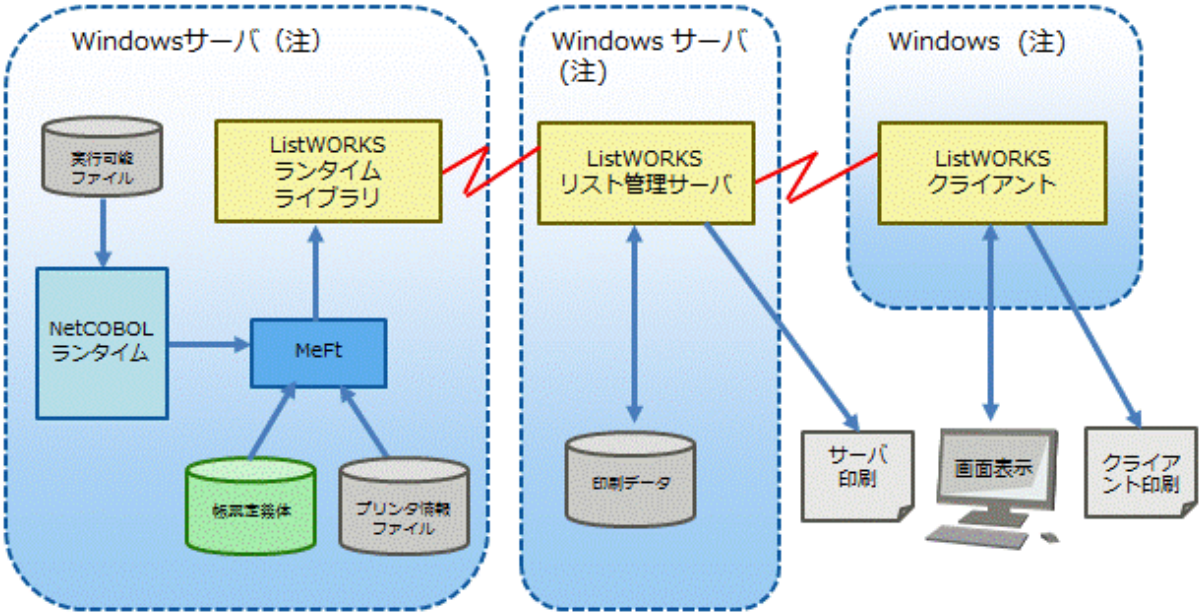
プリンタ情報ファイルに指定を追加することにより、MeFt経由で出力する帳票を電子化することができます。帳票を電子化する方法には、電子帳票保存とPDFファイル出力があります。それぞれの詳細は、“MeFtのオンラインマニュアル”および各連携製品のマニュアルを参照してください。

以下にそれぞれの関連図を示します。

・プリンタ出力の場合

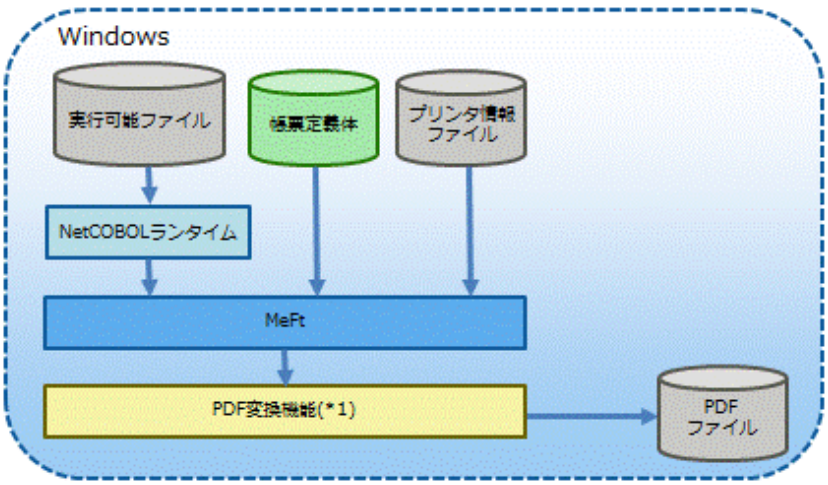


・電子帳票保存の場合



MeFtの出力帳票をListWORKS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 : ListWORKS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します。
	ASSIGN句	ファイル参照子	ファイル識別名、ファイル識別名定数またはデータ名のどれかを記述します。ファイル参照子は、実行時にMeFtの使用するプリンタ情報ファイルを割り当てるために使用します。
	FORMAT句	データ名	作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名は、帳票定義体名を設定するために使用します。
	GROUP句	データ名	作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名は、帳票定義体で定義した項目群名を設定するために使用します。
任意	FILE STATUS句	データ名	作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、入出力処理の実行結果が設定されます。(注) 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。

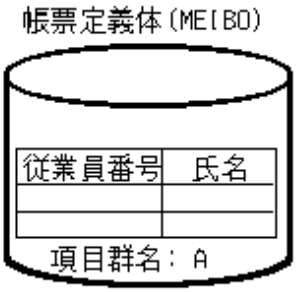
これらのページの形式は、ページの形式を変更する上記のWRITE文を実行しないかぎり、次のページへも引き継がれます。

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。

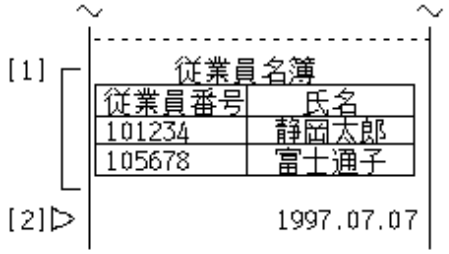
なお、制御レコードの出力の直後のWRITE文には、AFTER ADVANCING PAGEを指定します。

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-m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]“J.1.14 -m（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、環境変数 SMED_SUFFIXで、拡張子を指定します。[参照]“1.2.1 環境変数の設定”

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。

a. 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。

b. 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。

c. 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 ファイルの割当て”を参照してください。なお、プリンタ情報ファイルの内容および作成方法については、“MeFtのオンラインマニュアル”を参照してください。

参考

プリンタ情報ファイルを相対パス名で指定した場合、次の順序で検索されます。

1. 環境変数MEFTDIRに設定したフォルダ
2. カレントフォルダ

印刷ファイルで帳票定義体を使う場合の実行例を以下に示します。

例

〔COBOL プログラムのASSIGN句の記述内容〕

```
ASSIGN TO PRTFILE
```

〔実行用の初期化ファイルの内容〕

```
   :
PRTFILE=C:¥DIR1¥MEFPRC
   :
```

COBOL プログラムのASSIGN句に指定したファイル識別名に、プリンタ情報ファイルを割り当てます。

注意

FORMAT句付き印刷ファイルでフォームオーバレイパターンを使用する場合、フォームオーバ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。

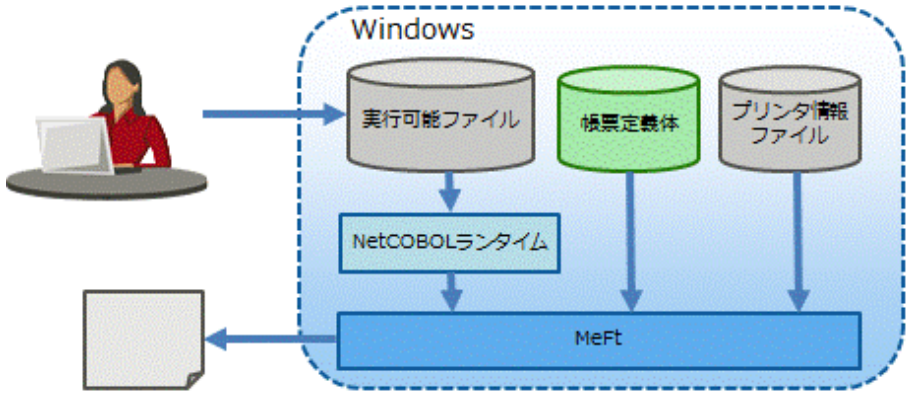
8.5.1 概要

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。

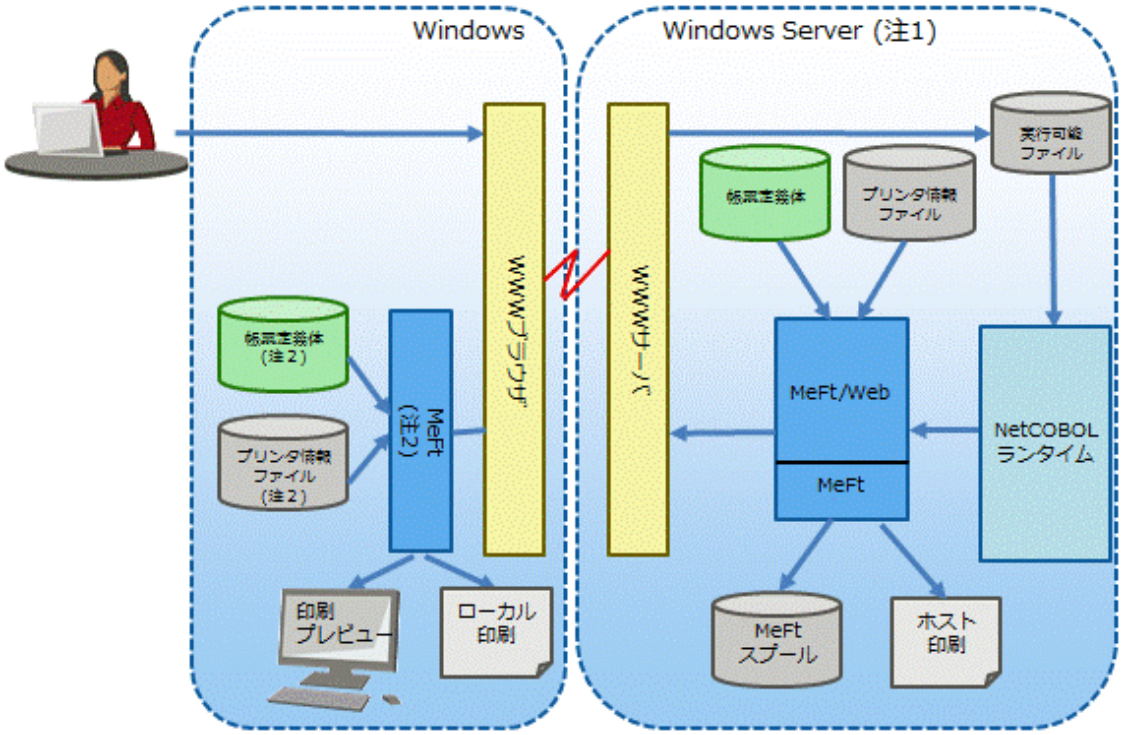
これらの関連図を以下に示します。

- 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1 ： Solaris、Linuxでも使用できます。ただし、そのシステムに対応したMeFtおよびMeFt/Webが必要になります。

注2 ： 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に、プリンタ情報ファイルは実行時までに作成します。以下に表示ファイル機能を使って帳票印刷を行うときの標準的な作業手順を示します。

1. FORMを使って帳票定義体を作成します。
2. テキストエディタを使ってCOBOLソースプログラムを作成します。
3. COBOLソースプログラムを翻訳・リンクし、実行可能プログラムを作成します。
4. テキストエディタを使ってプリンタ情報ファイルを作成します。
5. 実行可能プログラムを実行します。

8.5.3 帳票定義体の作成

ここでは、表示ファイル機能で使用するための帳票定義体を作成するときに設定する情報および注意事項について記述します。FORMの詳しい機能や使用方法については、“FORMヘルプ”を参照してください。

帳票定義体を作成するときに設定する情報を“表8.8 帳票定義体に設定する情報”に示します。

表8.8 帳票定義体に設定する情報

情報の種類		指定する内容および用途
必須	ファイル名	帳票定義体を格納するファイルの名前を指定します。
	定義サイズ	帳票の大きさを行数と桁数で指定します。
	定義体形式	形式を指定します。
	データ項目	印刷するデータを設定するためのデータ項目を指定します。ここで指定した項目名は、COBOLプログラムを記述するときにデータ名として使用されます。
	項目群	1回の印刷処理で印字する1つ以上の項目を1つの項目群としてまとめます。ここで指定した項目群名は、COBOL プログラムを記述するときに使用します。
任意	項目制御部 (注)	COBOL プログラム中で帳票定義体の定義内容を、特殊レジスタを使って変更したい場合、5バイトの項目制御部を指定します。

注 ： 項目制御部は、帳票定義体に定義したデータ項目に付加される情報で、入力処理と出力処理で“共用する(3バイト)”と“共用しない(5バイト)”または“なし”の3種類があります。COBOLプログラムで特殊レジスタを使用する場合、“共用しない(5バイト)”を指定する必要があります。

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します。
	ASSIGN句	ファイル参照子	“GS-ファイル識別名”の形式で指定します。このファイル識別名は、実行時に接続製品が使用するプリンタ情報ファイルのパス名を設定する環境変数情報となります。
	FORMAT句	データ名	作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名には、帳票印刷を行うとき、帳票定義体名を設定します。
	GROUP句	データ名	作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名には、帳票印刷を行うとき、出力の対象となる項目群名を設定します。
	SYMBOLIC 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。

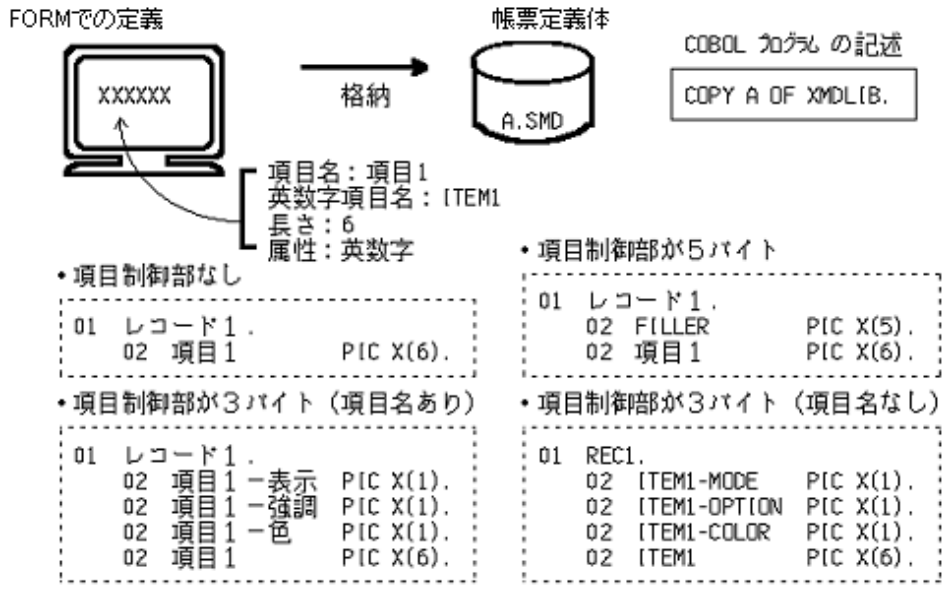
表8.10 入出力処理の種類と指定する値

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"
		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-m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]“J.1.14 -m（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、環境変数 SMED_SUFFIXで、拡張子を指定します。[参照]“1.2.1 環境変数の設定”

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。

a. 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。

b. 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。

c. 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のオンラインマニュアル”を参照してください。

プリンタ情報ファイルに設定する代表的な情報を“表8.11 プリンタ情報ファイルの設定情報”に示します。

表8.11 プリンタ情報ファイルの設定情報

情報の種類	指定する内容および用途
PRTDRV	出力するプリンタ装置のデバイス名を指定します。
MEDDIR	帳票定義体を格納したフォルダのパス名を設定します。
MEDSUF	帳票定義体を格納したファイルの拡張子を指定します。 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。

- 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

8.6 電子帳票出力機能を使う方法

ここでは、電子帳票出力機能の概要とFORMAT句なし印刷ファイルを使用して帳票を電子化する方法およびその注意事項について説明します。FORMAT句付き印刷ファイルおよび表示ファイル(帳票印刷)を使用した電子帳票出力機能については、“8.4.1.2 帳票の電子化”および“MeFtのオンラインマニュアル”を参照してください。

8.6.1 電子帳票出力機能の概要

COBOLでは、従来よりFORMAT句なし印刷ファイル、FORMAT句付き印刷ファイルおよび表示ファイル(宛て先PRT)機能を使用して帳票をプリンタ(紙)に印刷する機能を提供していました。これに加え、ListWORKSと連携することにより、従来のアプリケーションを変更することなく簡単な環境定義(印刷情報ファイルまたはプリンタ情報ファイル)だけで、帳票を電子化することが可能となります。

ListWORKSとは

ListWORKSとは、紙に出力して利用している帳票を電子化し、コンピュータの画面で参照することにより、帳票のデータを活用するシステムです。これにより、経費削減、情報提供のスピード化、信頼性の向上およびデータの有効利用を図ることが可能となります。

電子帳票ファイルの有効活用

ListWORKS連携により電子化した帳票を、紙に出力したイメージでコンピュータの画面にそのまま表示できます。電子化された帳票に対して、ListWORKSが提供する機能を利用することで以下の効果があります。

- 帳票を画面に表示して、付せん、メモ、チェックマークなどを記入できます。なお、電子化された帳票は、必要に応じていつでもプリンタ(紙)に印刷することもできます。
- 帳票は電子データであるため、紙より、迅速で正確にデータを検索できます。
- 表計算ソフト、ワープロソフトなど、ほかのアプリケーションとデータ連携できます。データの再入力をなくし、帳票業務の効率化を図ることができます。
- 電子化された帳票を、FAXや電子メールに添付して配信することができます。
- 紙での運用と比べて、仕分けや配送にかかっていた人件費や用紙コスト、保管費用などのコストを削減することができます。

なお、電子帳票およびListWORKSに関する詳細は、“ListWORKSのオンラインマニュアル”および“ListWORKSのヘルプ”を参照してください。

8.6.2 帳票を電子化する方法

ここでは、FORMAT句なし印刷ファイルを使用して帳票を電子化する方法について説明します。電子帳票出力機能を利用する場合、ListWORKSが必要となります。

プログラムの記述

既存アプリケーションを利用する場合

プログラム修正および再翻訳は一切必要ありません。既存アプリケーションに対して後述の環境定義(印刷情報ファイル)の指定を追加するだけで帳票を電子化することができます。

新規アプリケーションを作成する場合

帳票を電子化するための特別な記述は一切必要ありません。通常のプリンタ出力(紙)の場合と同様に、FORMAT句なし印刷ファイルの言語仕様に従ったプログラムを記述し、このアプリケーションに対する環境定義(印刷情報ファイル)の指定を追加するだけで帳票を電子化することができます。

印刷情報ファイル名の指定

印刷情報ファイル名の指定方法は、ASSIGN句の指定によって異なります。

ASSIGN句にファイル識別名を指定した場合

“PRTNAME:プリンタ名”の形式でListWORKSの電子保存装置名を指定した後に、“,,INF(印刷情報ファイルの絶対パス名)”を指定します。指定方法の詳細は、“C.2.78 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。

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)
```

プリンタの割当てを省略した場合、印刷情報ファイルのPRTOUT指定でプリンタの割当てを行う必要があります。

ASSIGN句にPRINTER-nを記述した場合

印刷情報ファイル名の指定方法は、ファイル識別名の場合と同じです。

“PRINTER-1”に仮想プリンタ名“FUJITSU ListWORKS Storage”を割り当てる場合

```
〔COBOL ソースプログラムのASSIGN句の記述〕
    SELECT  印刷ファイル ASSIGN TO PRINTER-1
〔環境変数情報〕
    PRINTER-1=PRTNAME:FUJITSU ListWORKS Storage,,INF(C:¥PRTAPL¥PRINTINF.TXT)
```

“PRINTER-2”に対するプリンタの割当てを省略した場合

```
〔COBOL ソースプログラムのASSIGN句の記述〕
    SELECT  印刷ファイル ASSIGN TO PRINTER-2
〔環境変数情報〕
    PRINTER-2=,,INF(C:¥PRTAPL¥PRINTINF.TXT)
```

注意

プリンタの割当てを省略した場合、印刷情報ファイルのPRTOUT指定でプリンタの割当てを行う必要があります。

ASSIGN句にデータ名を記述した場合

データ名領域に対して、“PRTNAME:プリンタ名”の形式でListWORKSの電子保存装置名を指定した後に、“,,INF(印刷情報ファイルの絶対パス名)”を指定したデータを設定します

例

データ名“PRTDATA”に仮想プリンタ名“FUJITSU ListWORKS Storage”を割り当てる場合

```
〔COBOL ソースプログラムのASSIGN句の記述〕
    SELECT  印刷ファイル ASSIGN TO PRTDATA
〔COBOL ソースプログラムのデータ記述項〕
    01 PRTDATA PIC X(128).
〔COBOL ソースプログラムの手続き部〕
    MOVE "PRTNAME:FUJITSU ListWORKS Storage,,INF(C:¥PRTAPL¥PRINTINF.TXT)"
         TO PRTDATA
```

データ名“PRTDATA”に対するプリンタの割当てを省略した場合

```
〔COBOL ソースプログラムのASSIGN句の記述〕
    SELECT  印刷ファイル ASSIGN TO PRTDATA
〔COBOL ソースプログラムのデータ記述項〕
    01 PRTDATA PIC X(128).
〔COBOL ソースプログラムの手続き部〕
    MOVE ",,INF(C:¥PRTAPL¥PRINTINF.TXT)" TO PRTDATA
```

注意

プリンタの割当てを省略した場合、印刷情報ファイルのPRTOUT指定でプリンタの割当てを行う必要があります。

ASSIGN句にファイル識別名定数を記述した場合

プログラム中に定数で、“PRTNAME:プリンタ名”の形式でListWORKSの電子保存装置名を指定した後に、“,,INF(印刷情報ファイルの絶対パス名)”を記述します。

例

仮想プリンタ名“FUJITSU ListWORKS Storage”を割り当てる場合

```
〔COBOL ソースプログラムのASSIGN句の記述〕
    SELECT  印刷ファイル ASSIGN
        TO "PRTNAME:FUJITSU ListWORKS Storage,,INF(C:¥PRTAPL¥PRINTINF.TXT)"
```

プリンタの割当てを省略した場合

```
〔COBOL ソースプログラムのASSIGN句の記述〕
    SELECT  印刷ファイル ASSIGN TO ",,INF(C:¥PRTAPL¥PRINTINF.TXT)"
```

注意

プリンタの割当てを省略した場合、印刷情報ファイルのPRTOUT指定でプリンタの割当てを行う必要があります。

ASSIGN句にPRINTERを記述した場合

環境変数情報@CBR_PrintInfoFileに印刷情報ファイルの絶対パス名を指定します。[参照]“C.2.43 @CBR_PrintInfoFile(ASSIGN句にPRINTERを指定したファイルに対して有効な印刷情報ファイルの指定)”

例

```
〔COBOL ソースプログラムのASSIGN句の記述〕
    SELECT  印刷ファイル ASSIGN TO PRINTER
〔環境変数情報〕
    @CBR_PrintInfoFile=C:¥PRTAPL¥PRINTINF.TXT
```

注意

電子帳票出力機能を利用する場合、ASSIGN TO PRINTER(通常使用するプリンタの設定)は意味を持ちません。印刷情報ファイルのPRTOUT指定に指定されたプリンタ名が有効となります。

印刷情報ファイルの定義

FORMAT句なし印刷ファイルの出力先をListWORKSに向ける(帳票を電子化する)ために、印刷情報ファイルに以下のパラメタを設定します。

STREAM=LW

帳票を電子化することを意味します。STREAM指定の詳細は、“8.1.12 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、ListWORKSが使用する電子帳票に関するさまざまな定義情報を含むファイルです。STREAMENV指定の詳細は、“8.1.12 印刷情報ファイル”を、電子帳票情報ファイルの詳細は、“ListWORKSのオンラインマニュアル”および“ListWORKSのヘルプ”を参照してください。

PRTOUT=PRTNAME:プリンタ名

プリンタ名にはListWORKSの電子保存装置名を指定します。ListWORKSの仮想プリンタを使用する場合、ListWORKSのインストール時に特に変更しない時の仮想プリンタ名は、“FUJITSU ListWORKS Storage”となります。

PRTOUT指定は、省略することもできます。PRTOUT指定を省略した場合、ASSIGN句の記述形式に従って、ListWORKSの電子保存装置名の指定が必要になります。両方指定された場合は、PRTOUTの指定が優先されます。PRTOUT指定の詳細は、“8.1.12 印刷情報ファイル”を、環境変数情報の指定の詳細は“C.2.78 ファイル識別名(プログラムで使用するプリンタおよび各種パラメタの指定)”を参照してください。

ListWORKSの準備・設定

ListWORKSのインストールおよび環境設定を行います。これらの詳細は、“ListWORKSのオンラインマニュアル”および“ListWORKSのヘルプ”を参照してください。

既存の帳票アプリケーションを電子化する場合の指定例

例

既存のプリンタ(紙)出力アプリケーションの指定例

```
〔COBOL ソースプログラムのASSIGN句の記述〕
    SELECT  印刷ファイル ASSIGN TO S-PRTFILE
〔環境変数情報〕
    PRTFILE=PRTNAME:FUJITSU FMLBP
〔印刷情報ファイルの記述〕
    なし
```

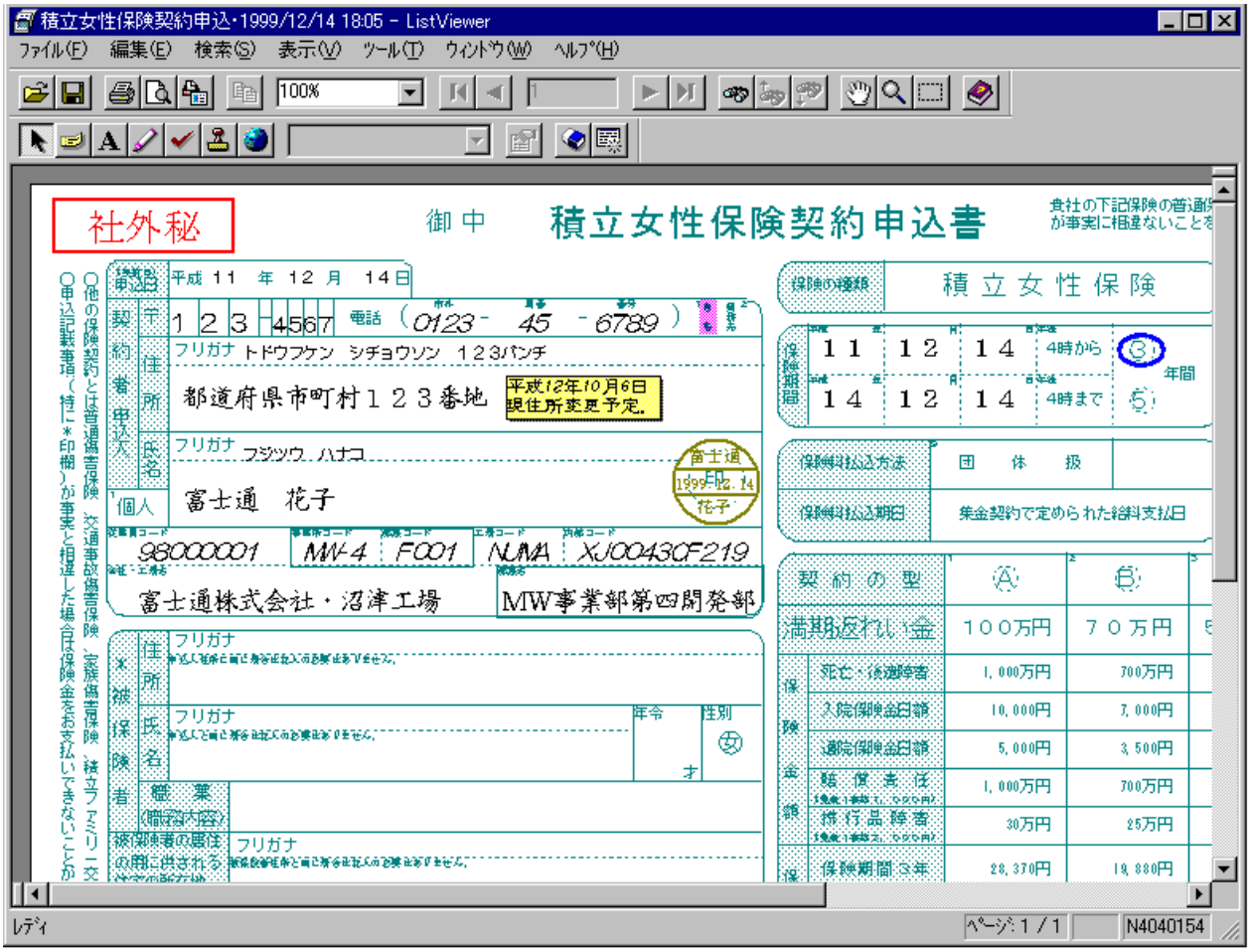
上記既存アプリケーションを電子化する指定例

```
〔COBOL ソースプログラムのASSIGN句の記述〕
    SELECT  印刷ファイル ASSIGN TO S-PRTFILE
〔環境変数情報〕
    PRTFILE=PRTNAME:FUJITSU FMLBP,,INF(C:¥PRTAPL¥PRINTINF.TXT)
〔印刷情報ファイル(C:¥PRTAPL¥PRINTINF.TXT)の内容〕
    STREAM=LW
    STREAMENV=C:¥LW.TXT
    PRTOUT=PRTNAME:FUJITSU ListWORKS Storage
```

[解説]

印刷情報ファイルに、STREAM=LWを指定することで、電子帳票出力を行うことが決定付けられます。STREAMENVに指定された電子帳票情報ファイルLW.TXTは、ListWORKSに渡され帳票を電子化する際のさまざまな定義情報として利用されます。PRTOUTにListWORKSの仮想プリンタ名を指定することで、ファイル識別名に指定されたFUJITSU FMLBPが無視され、帳票の出力先がFUJITSU ListWORKS Storageに向けられます。帳票の電子化に必要な定義はこれだけです。

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パラメタを指定してください。
	@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	○	○	
	OverlayPrintSPEC	○	×	常に“COBOL”が指定された場合と同じ状態になります。
	OverlayPrintOffset	○	×	常に“VALID”が指定された場合と同じ状態になります。
	STREAM (*1)	○	○ 必須	
	STREAMENV	× (*2)	○	
	PRTOUT (*3)	○	○	
	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、ListWORKSの電子保存装置名を“PRTNAME:プリンタ名”の 形式で指定します。

プログラムの指定関連

ページ属性に関する機能

COBOLの機能	プリンタ	ListWORKS
フォームオーバレイ	○ [a]	○ [a]
フォームオーバレイ焼き付け回数 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、単一オーバレイだけサポートしています。

[b] フォームオーバレイ焼き付け回数

プリンタ出力および電子帳票出力のどちらの場合でも、焼き付け回数は複写数と同じ値が採用されます。

[c] OSIV系システム固有またはプリンタ固有情報

OSIV系システム固有機能またはプリンタ固有情報であるため、プリンタ出力および電子帳票出力のどちらの場合も、意味を持たない指定です。

[d] 印刷形式

電子帳票出力時は、PZOOM/LZOOM/LP縮刷は有効になりません。PZOOMはポートレートモード、LZOOMおよびLP縮刷はランドスケープモードとみなされます。また、電子帳票出力時は1つのファイル内で複数の印刷形式をページ単位で混在させることができません。この場合、最初に指定された印刷形式が後続のページに対しても引き継がれます。

[e] 用紙サイズ指定

電子帳票出力時は1つのファイル内で複数の用紙サイズをページ単位で混在させることができません。この場合、最初に指定された用紙サイズが後続のページに対しても引き継がれます。
ListWORKSを利用してFAX送付を行う場合、Systemwalker/PrintMGRまたはInterstage Print Manager(以降PrintMGR)のFAX連携機能と連携する必要があります。PrintMGRのFAX連携機能で利用できる用紙サイズについては、“PrintMGRのオンラインマニュアル”を参照してください。

[f] 電子帳票では意味のない機能

これらの機能は、プリンタ(紙)に出力する場合に意味を持つ機能です。電子帳票出力時は、これらの指定は無視されます。

[g] とじしろ方向・とじしろ幅・印刷原点指定

電子帳票出力時の画面表示では、これらの指定は無視されます。本機能は、紙に印刷される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	ListWORKS
文字サイズ 3.0 ～ 300.0 ポ ： 0.1ポ単位		○	○
文字ピッチ 0.01 ～24.00 CPI ： 0.01CPI単位		○	○
書体	書体名	○	○
	書体番号 FONT-001 ～ FONT999	○	○
文字回転	横書き	○	○
	縦書き (反時計回り90度)	○	○
文字形態	全角／半角／平体／長体／倍角	○	○
	ボールド・イタリック	○	○
水平スキップ 0.01 ～ 24.00 CPI ： 0.01CPI単位		○	○

文字属性に関する機能差は、特にありません

その他(ListWORKSの制限事項)

ListWORKSの制限によりCOBOLの一部の機能が利用できないことがあります。ListWORKSの制限事項およびその解除に関する詳細は、“ListWORKSのオンラインマニュアル”および“ListWORKSのヘルプ”を参照してください。

8.6.5 実行時エラーについて

ListWORKSランタイムライブラリでエラーが発生すると、以下の実行時メッセージが出力されます。

```
JMP0362I-U ’ $2’ ファイルに対する’ $1’ 文の実行で，レコード生成処理のエラーが発生しました。 CODE=$3. $4
```

ListWORKSランタイムライブラリから通知されるエラー状態は、上記メッセージの$3に示されます。“ListWORKSのヘルプ”を参照し、“詳細コード”の“ListWORKSランタイムライブラリの内部コード”から$3に示されるエラーコードの意味を調べ、エラーの原因を取り除いた後、再度実行してください。

参考

以下に、“ListWORKSのヘルプ”に記載されていないエラーコードについて示します。

CODE	意味	対処
1052	COBOLプログラムの翻訳時に指定されたコード系(翻訳オプションENCODEまたは翻訳オプションRCS)が、現在使用しているListWORKSでは未サポートです。	現在使用しているListWORKSがサポートするコード系を調べ、COBOLプログラムの翻訳時に指定するコード系を変更してください。
2000	WRITE文に記述したレコードにおいて、同一基本項目内に1バイトコード系と2バイトコード系を混在してはなりません。	エラーが発生したWRITE文を調べ同一基本項目内に1バイトコード系と2バイトコード系が混在しないように修正してください。

第9章 画面を使った入出力

本章では、画面を表示したり、表示した画面からデータを入力したりする方法について説明します。

9.1 画面を使った入出力の種類

COBOLプログラムでは、ディスプレイ装置に画面を表示し、表示した画面からデータを入力する(以降の説明では、単に画面入出力といいます)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(注)	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	・ 表示ファイルの定義 ・ 表示レコードの定義 ・ OPEN文 ・ READ文 ・ WRITE文 ・ CLOSE文	・ 画面の定義 ・ ACCEPT文 ・ DISPLAY文
	関連製品	・ FORM(画面の定義) ・ MeFt ・ MeFt/Web(Web環境でのリモート入出力処理)	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

注:MeFt/Webを使用する場合、MeFtが必要です。

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。

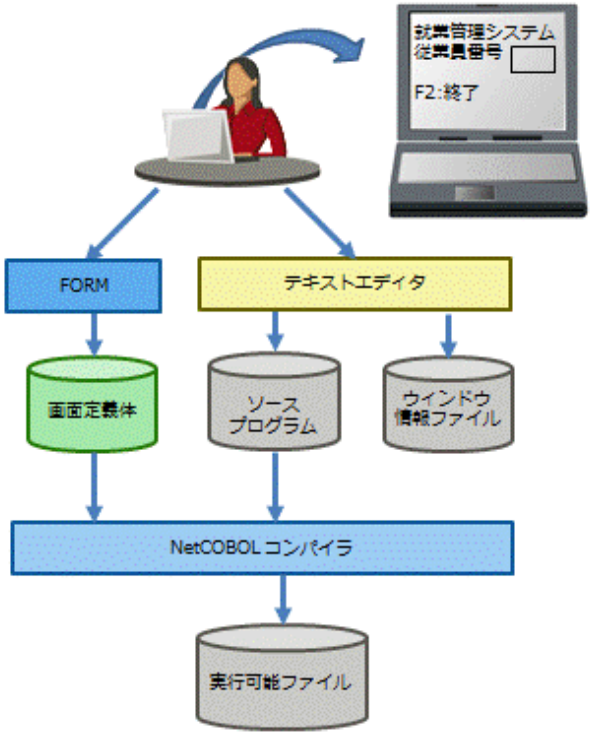
9.2.1 概要

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

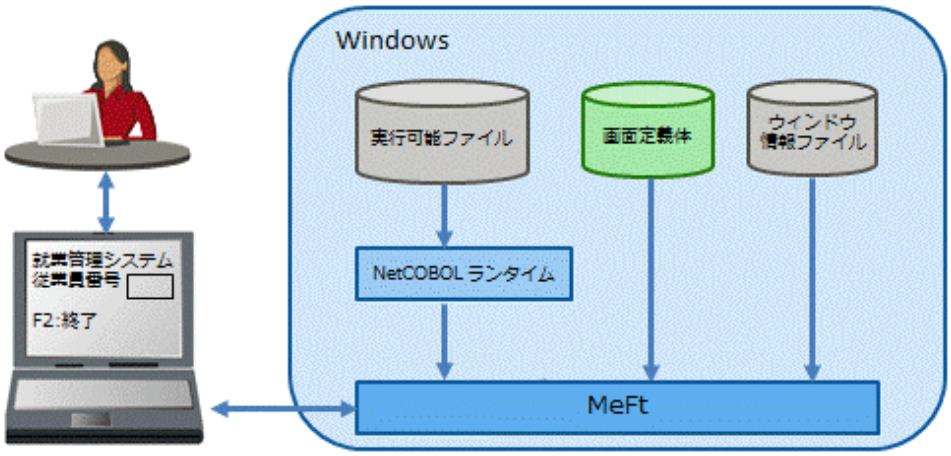


表示ファイル機能を実際に使用するときには、画面定義体、MeFtおよびMeFt/Webが必要です。

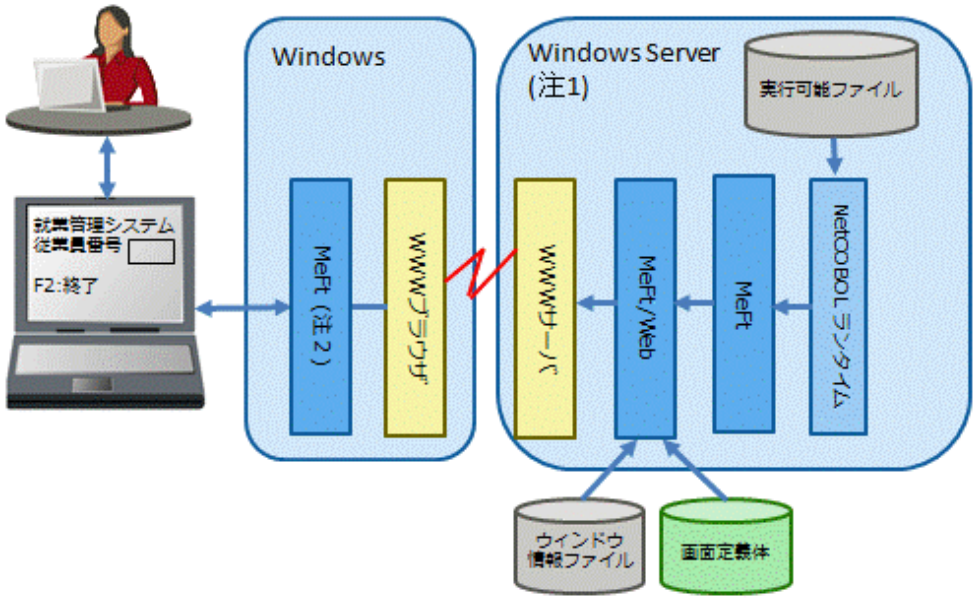
これらの関連図を以下に示します。

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1：Solaris、Linuxでも使用できます。ただし、MeFtおよびMeFt/Webが必要になります。

注2：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。

情報の種類		指定する内容および用途
	データ項目	画面入出力を行うための領域を指定します。ここで指定した項目名は、COBOL プログラムを記述するときにデータ名として使用されます。
	項目群	1回の入出力処理で表示またはデータ入力する1つ以上の項目を1つの項目群としてまとめます。ここで指定した項目群名は、COBOL プログラムを記述するときに使用します。
任意	項目制御部 (注)	COBOL プログラム中で画面定義体の定義内容を変更したい場合、5バイトの項目制御部を指定します。
	アテンション情報	COBOL プログラム中で入力キーを判定する場合、指定します。

注 ： 項目制御部は、画面定義体に定義したデータ項目に付加される情報で、入力処理と出力処理で“共用する(3バイト)”と“共用しない(5バイト)”または“なし”の3種類があります。COBOLプログラムで特殊レジスタを使用する場合、“共用しない(5バイト)”を指定することをおすすめします。

9.2.5 プログラムの記述

ここでは、表示ファイル機能を使うプログラムの記述内容について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.   プログラム名.
 ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
   FILE-CONTROL.
     SELECT  ファイル名
       ASSIGN TO  GS-ファイル識別名
       SYMBOLIC DESTINATION IS "DSP"
       FORMAT IS  画面定義体名通知域
       GROUP  IS  項目群名通知域
      [SELECTED FUNCTION IS アテンション種別通知域]
      [PROCESSING MODE IS 処理種別通知域]
      [UNIT CONTROL IS  特殊制御情報通知域]
      [FILE STATUS IS   入出力状態１  入出力状態２].
 DATA DIVISION.
  FILE SECTION.
  FD  ファイル名.
      COPY  画面定義体名 OF XMDLIB.
 (01  表示レコード名.         ) (注)
 (     02   データ名     ～.  )
  WORKING-STORAGE SECTION.
  01  画面定義体名通知域            PIC X(8).
  01  項目群名通知域                PIC X(8).
 [01  アテンション種別通知域        PIC X(4).]
 [01  処理種別通知域                PIC X(2).]
 [01  特殊制御情報通知域            PIC X(6).]
 [01  入出力状態１                  PIC X(2).]
 [01  入出力状態２                  PIC X(4).]
 PROCEDURE DIVISION.
     OPEN I-O   ファイル名.
    [MOVE   出力の指定       TO EDIT-MODE   OF  データ名.]
    [MOVE   強調の指定       TO EDIT-OPTION OF  データ名.]
    [MOVE   色               TO EDIT-COLOR  OF  データ名.]
    [MOVE   入力の指定       TO EDIT-STATUS OF  データ名.]
    [MOVE   カーソルの位置   TO EDIT-CURSOR OF  データ名.]
     MOVE   画面定義体名     TO 画面定義体名通知域.
     MOVE   項目群名         TO 項目群名通知域.
    [MOVE   処理種別         TO 処理種別通知域.]
    [MOVE   制御情報         TO 特殊制御情報通知域.]
     WRITE  表示レコード名.
     READ   ファイル名.
```

```
    CLOSE   ファイル名.
END PROGRAM   プログラム名.
```

注 : ()内はCOPY文の展開を表します。

環境部(ENVIRONMENT DIVISION)

表示ファイルを定義します。表示ファイルは、通常のファイルを定義するときと同様に、入出力節のファイル管理段落にファイル管理記述項を記述します。ファイル管理記述項に記述する内容を“表9.3 ファイル管理記述項に指定する情報”に示します。なお、これらの情報は、FORMで作成した画面定義体の定義内容とは関係なく値を決めることができます。

表9.3 ファイル管理記述項に指定する情報

	指定する場所	情報の種類	指定する内容および用途
必須	SELECT句	ファイル名	COBOL プログラム中で使用するファイル名を指定します。このファイル名は、COBOL の利用者語の規則に従った名前にします。
	ASSIGN句	ファイル参照子	“GS-ファイル識別名”の形式で指定します。このファイル識別名は、実行時に接続製品が使用するウィンドウ情報ファイルのパス名を設定する環境変数情報名となります。
	FORMAT句	データ名	作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名には、画面入出力を行うとき、画面定義体名を設定します。
	GROUP句	データ名	作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名には、画面入出力を行うとき、入出力の対象となる項目群名を設定します。
任意	SYMBOLIC DESTINATION句	画面の表示先	"DSP" を指定します。(この句の省略値は"DSP"のため省略することができます。)
	FILE STATUS 句	データ名	作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、入出力処理の実行結果が設定されます。(注1) 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)
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

	処理種別		制御情報 (注)	
	全画面消去出力	"CL"		
	メニュー項目の選択禁止	"PF"	メニュー項目	アテンション情報
	メニュー項目の選択禁止の解除	"PN"		
	端末制御出力	"CT"	ウィンドウ操作の種類	

注 ： 制御情報の値と詳細については、“MeFtのオンラインマニュアル”を参照してください。

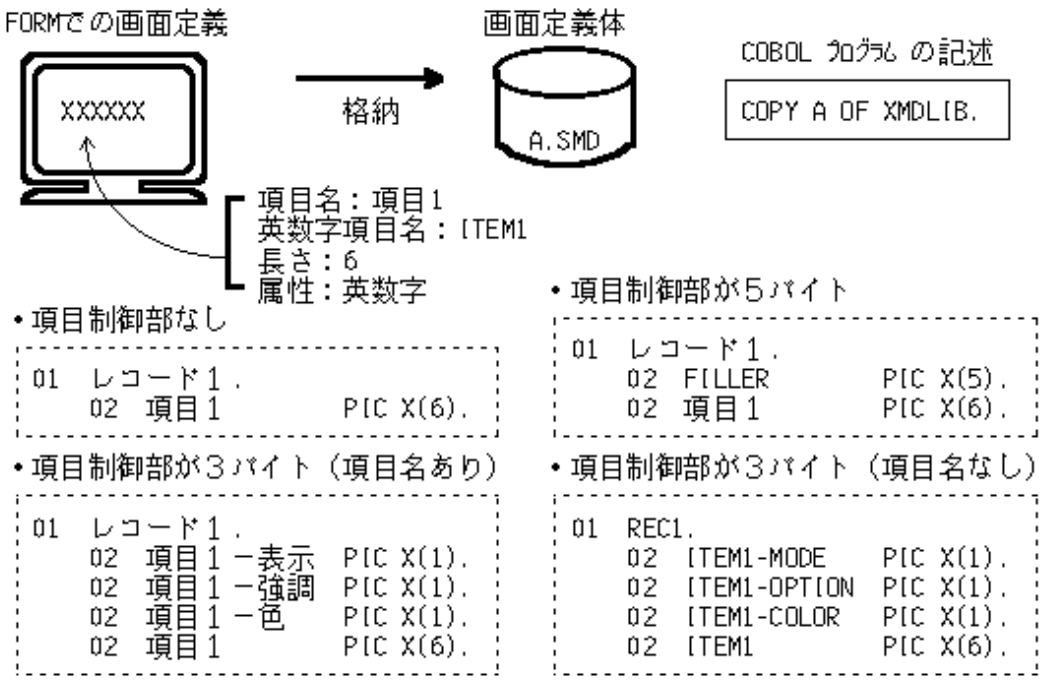
表9.5 アテンション情報の値とアテンション種別

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

注1 ： ファンクションキー、メニュー項目など。
注2 ： 実行キーの存在しないシステムでは、その代替キーを設定する必要があります。この情報は、ウィンドウ情報ファイルに設定します。指定方法および内容については、“MeFtのオンラインマニュアル”を参照してください。

データ部(DATA DIVISION)

データ部には、表示レコードの定義およびファイル管理記述項に指定したデータの定義を記述します。表示レコードに定義するレコード記述文は、IN/OF XMDLIBを指定したCOPY文を使って画面定義体から取り込むことができます。展開されるレコードの内容を以下に示します。



注意

表示ファイルに対してEXTERNAL句を指定する場合には、“10.2.5.2 外部ファイル使用時の注意事項”を必ずお読みください。

手続き部(PROCEDURE DIVISION)

画面入出力処理には、通常のファイル処理を行うときと同様に、入出力文を使います。

入出力は、次に示す順序で実行します。

1. I-O指定のOPEN文: 画面入出力処理の開始
2. READ文およびWRITE文: 画面入出力処理
3. CLOSE文: 画面入出力処理の終了

OPEN文およびCLOSE文

OPEN文は画面入出力処理の開始時に、CLOSE文は画面入出力処理の終了時に、それぞれ1回だけ実行します。

READ文またはWRITE文

画面を表示するときには表示レコードを指定したWRITE文を、画面からデータを読み込むときには表示ファイルを指定したREAD文を使います。WRITE文を実行する前には、FORMAT句に指定したデータ名に画面定義体名、GROUP句に指定したデータ名に項目群名を設定しておく必要があります。GROUP句に指定したデータ名に設定されている項目群に属するデータ項目が入出力の対象となります。
WRITE文またはREAD文の実行前に、特殊レジスタに値を設定することにより、表示レコード中のデータ項目の属性を変更することができます。各特殊レジスタに設定する値については、“MeFtのオンラインマニュアル”を参照してください。READ文の実行時に入力データに誤りがあり、画面定義体で再入力要求指示を定義している場合には、誤りがなくなるまで再入力のための画面表示と入力編集が繰り返し行われます。
誤りがない場合または再入力要求指示を定義していない場合は、入力編集結果の情報がCOBOLプログラムに通知されます。また、SELECTED FUNCTION句で指定したデータ名にアテンション情報が通知されます。
READ文の実行後、入力結果が特殊レジスタEDIT-STATUSに返却されます。設定される値については、“MeFtのオンラインマニュアル”を参照してください。

注意

画面定義体で定義された項目属性のうち、英数字日本語混在項目は、COBOLプログラム上では英数字項目として扱われます。また、この項目の内容をプログラム上で明に操作することはできません。すなわち、入力した英数字日本語混在項目は、そのまま英数字日本語混在項目として出力することしかできません。

特殊レジスタの使い方

特殊レジスタを使って、表示レコード中の項目制御部を持つデータ項目(画面定義体で定義したデータ項目)の属性を変更することができます。

表示ファイルの特殊レジスタには、次の5種類があります。

特殊レジスタ	用途
EDIT-MODE	出力処理の対象とする/しないなどを指定します。
EDIT-OPTION	強調、下線付き、反転表示などを指定します。
EDIT-COLOR	色を指定します。
EDIT-STATUS	入力処理の対象とする/しないなどを指定します。また、入力結果が通知されます。
EDIT-CURSOR	カーソル位置を指定します。

これらの特殊レジスタは、画面定義体で定義したデータ名で修飾して使います。たとえば、データ名Aの色属性の設定は、“EDIT-COLOR OF A”のように記述します。各特殊レジスタに設定する値については、“MeFtのオンラインマニュアル”を参照してください。

注意

- 1つのプログラム中で、項目制御部の長さ(“共用する(3バイト)”/“共用しない(5バイト)”/“なし”)が異なる複数の画面定義体を混在して使うことはできません。

・“共用する(3バイト)”の項目制御部を使用する場合、以下のような注意が必要です。
項目制御部を入力処理と出力処理で共用する3バイト系の画面帳票定義体では、画面帳票定義体に定義された各項目について、EDIT-MODEとEDIT-STATUS、EDIT-CURSORとEDIT-OPTIONは同じ記憶領域が使用されます。たとえば、データ名Aについて、“EDIT-MODE OF A”と“EDIT-STATUS OF A”、“EDIT-CURSOR OF A”と“EDIT-OPTION OF A”は、同じ記憶領域を使用します。

入出力エラー処理

入出力エラーの検出方法および入出力エラーが発生したときの実行結果については、“7.6 入出力エラー処理”を参照してください。

9.2.6 プログラムの翻訳・リンク

翻訳

翻訳オプション-m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] “J.1.14 -m（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、環境変数SMED_SUFFIXで拡張子を指定します。[参照] “1.2.1 環境変数の設定”

リンク

結合が必要なライブラリは特にありません。

9.2.7 ウィンドウ情報ファイルの作成

ここでは、表示ファイル機能を使って画面入出力処理を行うときのウィンドウ情報ファイルに設定する情報および注意事項について記述します。ウィンドウ情報ファイルの詳しい内容や作成方法については、“MeFtのオンラインマニュアル”を参照してください。

ウィンドウ情報ファイルに設定する情報を以下に示します。

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/Webを使用する場合

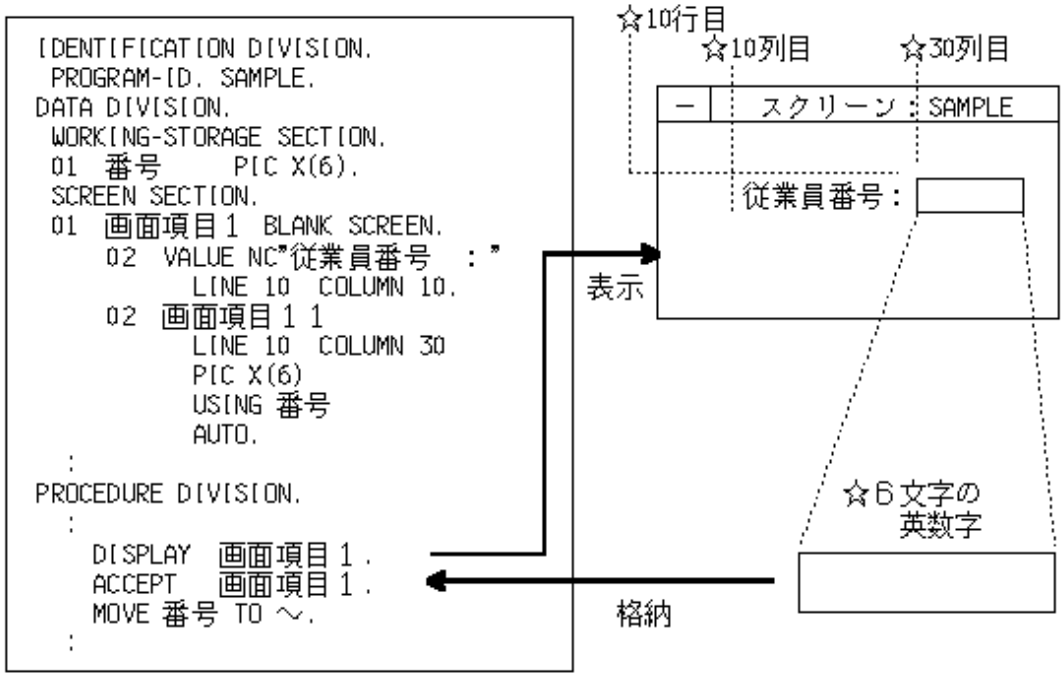
MeFt/Webを使用する場合には、“MeFt/Webユーザーズガイド”を参照してください。

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.2.74 @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	００００ △	００００ △
	”1”を入力	１０００ △	１０００ △
	”2”を入力	１２００ △	１２００ △
	”. ”を入力	１２００ △	００１２

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	１２００．００ △	□□１２．００ △
	”3”を入力	１２３０．００ △	□１２３．００ △
	”4”を入力	１２３４．００ △	１２３４．００ △
	”5”を入力	１２３４５００ △	１２３４．５０ △
	”6”を入力	１２３４５６０ △	１２３４．５６ △
	”ENTERキー”を入力	４５６０．００	１２３４．５６

上例は、TO指定で使用した例です。USING指定の場合またはFROM指定とTO指定が同時に指定されている場合、あらかじめ0で初期化されていないデータ項目では、0表示されている箇所が空白表示になることがあります。

キー定義のセクション名は固定です。

キー名の記述方法および指定可能なキーについては、製品に含まれるキー定義ファイルのサンプルを参照してください。

注意

- 状態キー2の値については、現在は'000'-'255'まで使用可能です。これ以上の値を有効なキーとして利用した場合は、CRT STATUS句に返却できる範囲を超えることになります。この場合の動作は、保証しません。
- “ENTER=2”を指定した場合、ENTERキーで画面からのデータ入力を終了させることはできません。この場合、別のキーを終了キーとして設定しておく必要があります。“ENTER=2”を指定しないかぎり、ENTERキーは終了キーとして使用できます。ENTERキーには、2(Tabキー)以外の値は割当てられません。
- キー定義ファイル内に[COBOL.KBD]セクションおよび[COBOL.ACCPMODE]セクションが複数ある場合、最初に現れたセクションが有効になります。セクション内の各キーの定義についても、同じキー名の複数の情報がある場合は、最初に現れたキー名の情報が有効になります。

- キー定義ファイル内の記述には、空白を含めてはなりません。
- キー定義ファイル内の記述は、大文字、小文字を区別して記述してください。
- キー定義ファイル内の記述で、行の先頭にセミコロン(;)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	×	○	△	○	○	○
その他	カーソルの自動スキップ	AUTO句	×	○	△	○	○	○
	一般的性質の指定	PICTURE句	×	○	○	○	×	○
	表現形式の指定	USAGE句	×	○	○	○	○	○
	定数項目の指定	VALUE句	○	×	×	×	×	○

○ ： 指定可能
△ ： 指定できるが有効にならない
× ： 指定できない
注 ： 集団は集団項目として定義し、基本は基本項目として定義してください。

手続き部(PROCEDURE DIVISION)

画面を表示するには、画面項目を指定したDISPLAY文を使います。DISPLAY文を実行すると、ディスプレイ装置に、指定した画面項目で定義された画面が表示されます。

画面からデータを入力するには、画面項目を指定したACCEPT文を使います。ACCEPT文を実行すると、ディスプレイ装置に表示された画面からデータを入力することができます。入力操作が終了すると、特殊名段落のCRT STATUS句に指定したデータ名に、画面入力状態が設定されるので、その値によって処理を切り分けることができます。

9.3.5 プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

9.3.6 プログラムの実行

1. ファンクションキーの利用者定義を行う場合、環境変数情報@CBR_SCR_KEYDEFFILEにキー定義ファイルを指定します。[参照]"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、ほかのプログラムを呼び出したり、ほかのプログラムから呼び出されたりすることができます(“呼出し関係の形態(a)”参照)。ほかのプログラムは、他言語で記述されたプログラムでも可能です。ただし、再帰属性を持たないCOBOLプログラムを再帰的に呼び出すことはできません(“呼出し関係の形態(b)”参照)。また、再帰属性を持たないCOBOLプログラム自身を呼び出すこともできません(“呼出し関係の形態(c)”参照)。

図10.1 呼出し関係の形態(a)

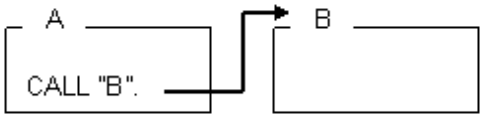


図10.2 呼出し関係の形態(b)

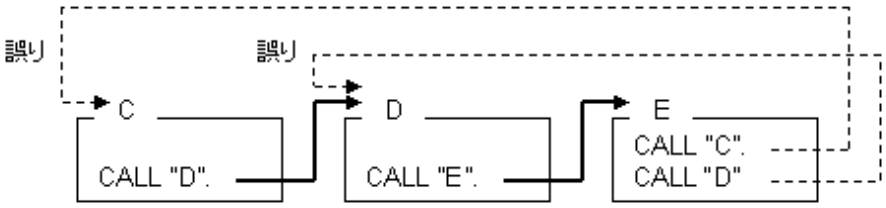
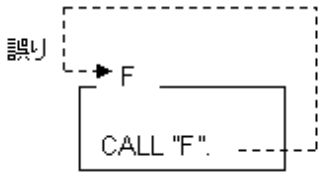


図10.3 呼出し関係の形態(c)



10.1.2 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、“18.3.1 実行環境と実行単位”を参照してください。

COBOLの実行単位とは、COBOLの主プログラムに制御が渡ってからCOBOLの主プログラムが呼出し元に復帰する(下図(a))か、STOP RUN文が実行される(下図(b))までをいいます。ただし、他言語で記述されたプログラム(以降、他言語プログラムといいます)からCOBOLプログラムを呼び出す場合は例外があります。

ここでいう、COBOLの主プログラムとは、COBOLの実行単位中で最初に制御が渡ったCOBOLプログラムをいいます。

他言語プログラムからCOBOLプログラムを呼び出す場合は、最初のCOBOLプログラムの呼出しの前にJMPCINT2を呼び出し、最後のCOBOLプログラムの呼び出しの後で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.2 他言語連携で使用するサブルーチン”を参照してください。

実行環境の開設と閉鎖時の処理

実行環境の開設時には、COBOLプログラムが実行するために必要となる実行用の初期化ファイルの情報などが取り込まれます。実行環境閉鎖時には、COBOLプログラムで使用された資源を解放します。このときに行われる処理としては、小入出力機能で使用されたファイルのクローズ、オープンされたままのファイルのクローズ、外部ファイルのクローズ、外部データの解放、ファクトリオブジェクトの解放、未解放のオブジェクトインスタンスの解放などがあります。たとえば、下図(c)のような使い方をした場合、プログラムAとプログラムBの実行環境と、プログラムCの実行環境は別になるので、プログラムAとプログラムBの外部データと、プログラムCの外部データはそれぞれ別の領域となります。また、このような使い方をした場合、以下の注意が必要となります。このため、下図(d)のような使い方をするようにしてください。

図10.4 COBOLプログラムだけ

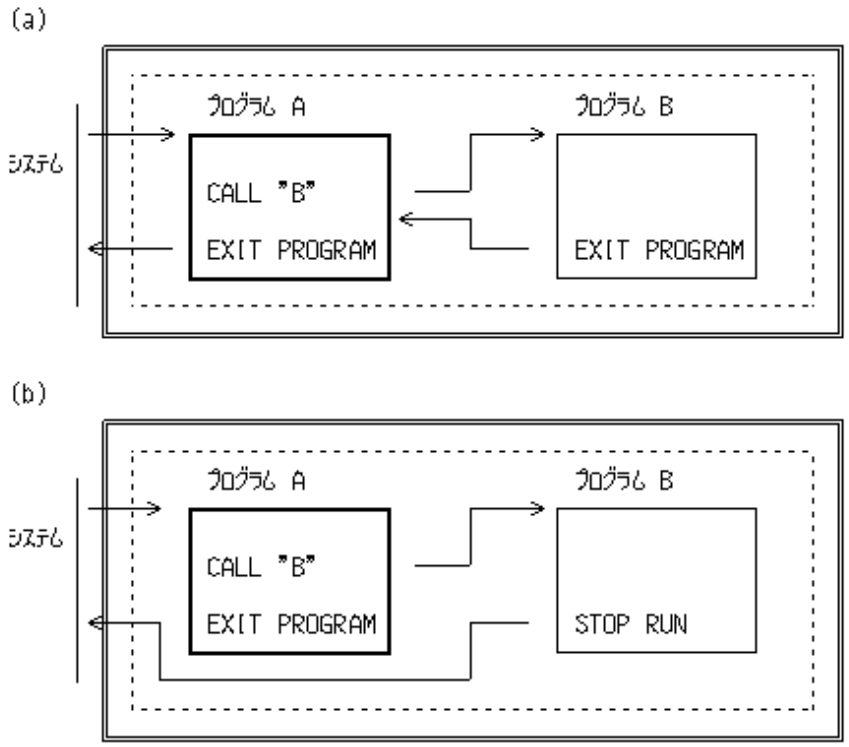
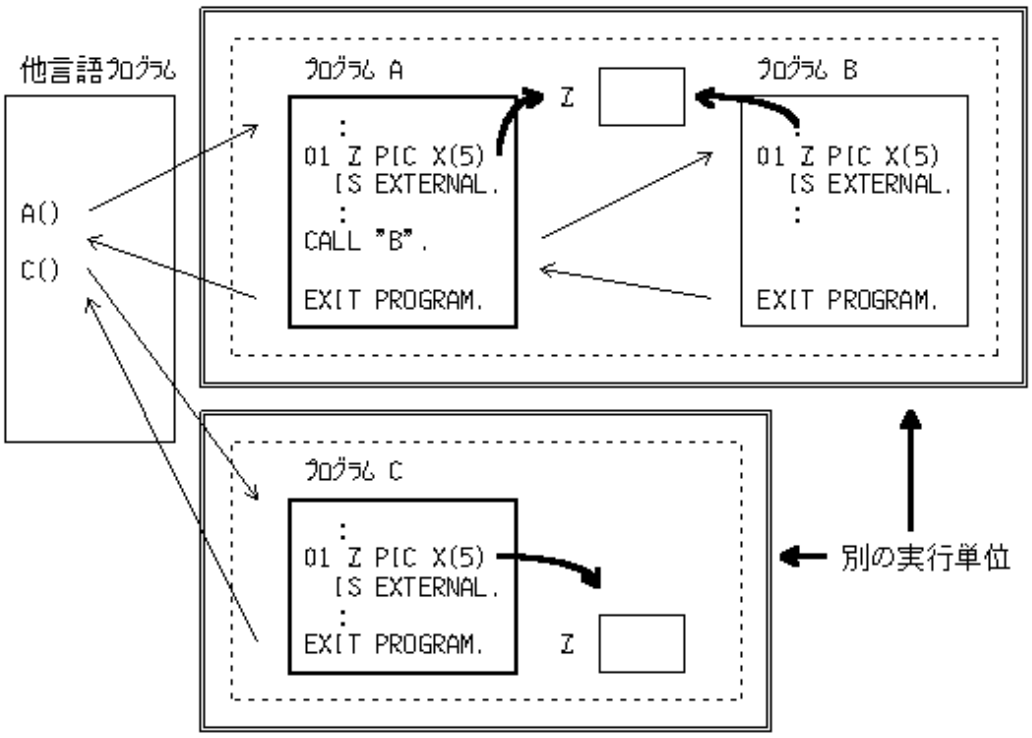
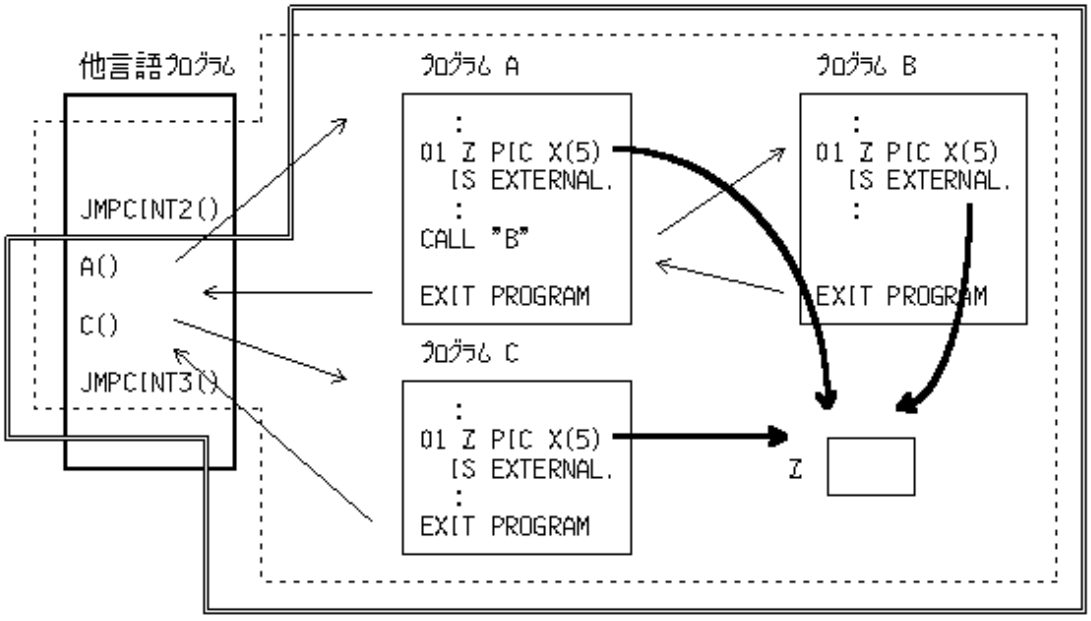


図10.5 他言語プログラムからCOBOLプログラムの呼出し

(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

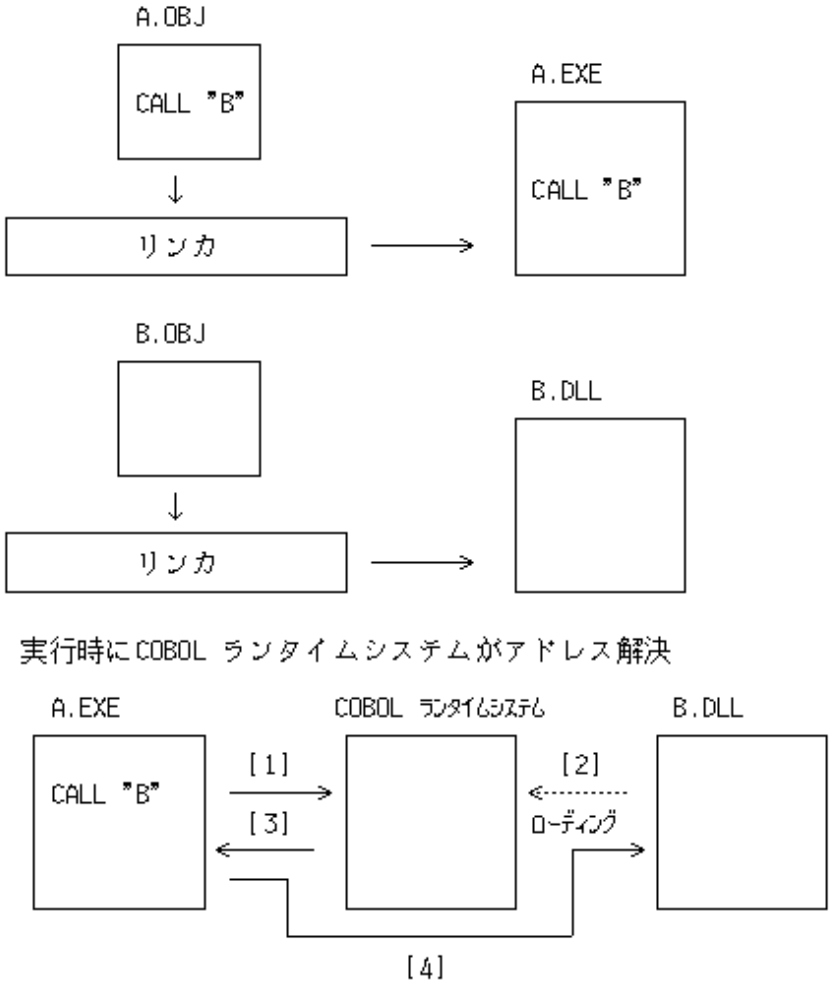
10.1.3 動的プログラム構造

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。

10.1.3.1 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、“18.10.3 動的プログラム構造”を参照してください。



[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(“10.1.3.3 注意事項”は必ずお読みください)。

10.1.3.2 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。

10.1.3.3 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10.6 キャンセルされる副プログラムが単純構造の副プログラムを呼び出している場合

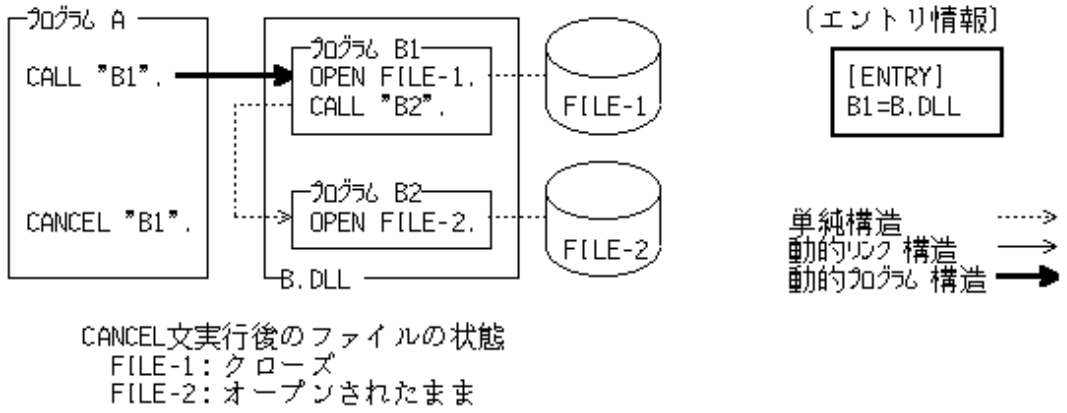
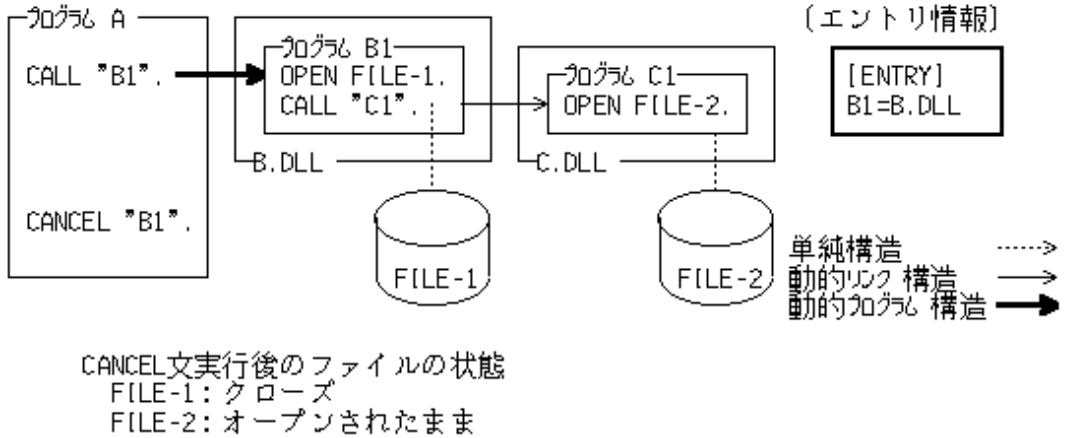


図10.7 キャンセルされる副プログラムが動的リンク構造の副プログラムを呼び出している場合



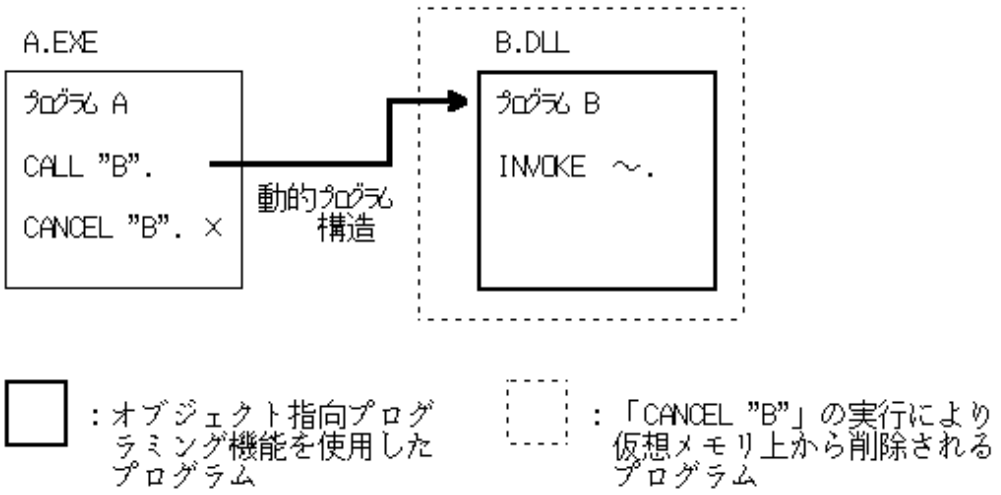
参考

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。



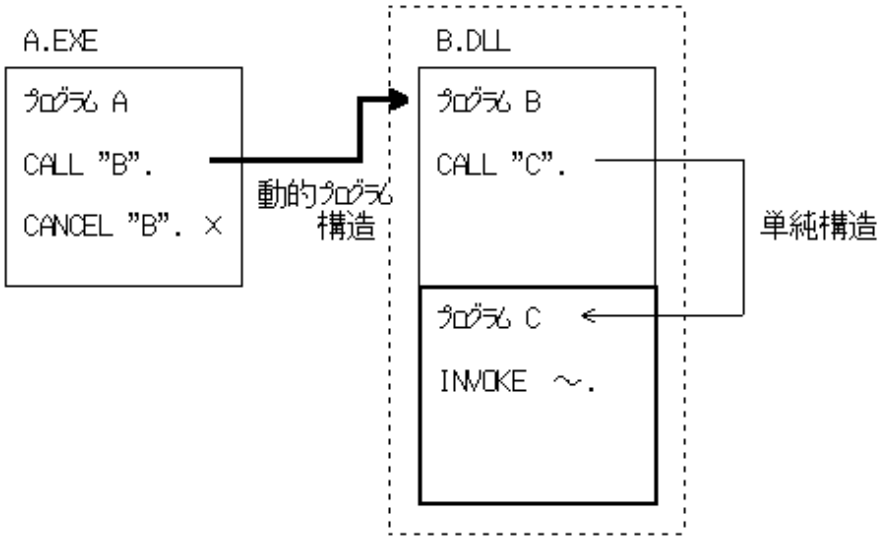
[例2] 以下の場合、CANCEL文の実行により、プログラムBが仮想メモリ上から削除されると、プログラムCも削除されるため、このCANCEL文は使用できません。

ー 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))

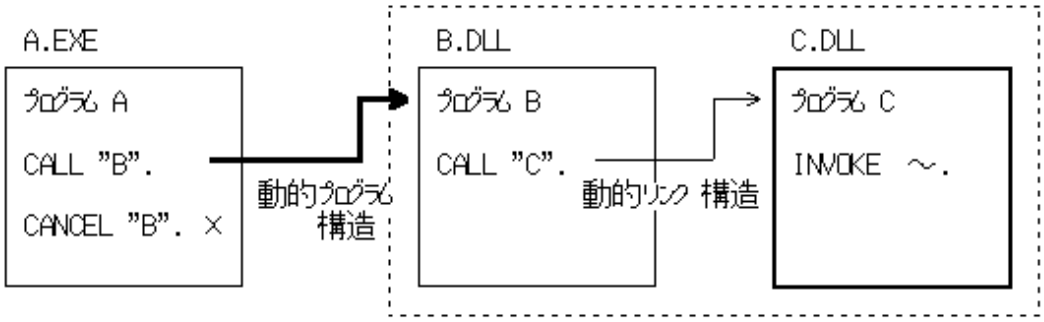
ー 動的リンク構造(下図(b))

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。

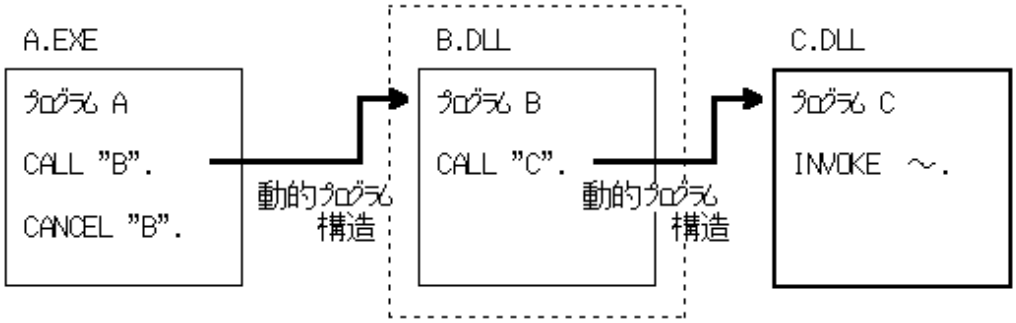
(a) プログラムＢとプログラムＣが単純構造



(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、手続きの途中に、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、手続きの途中に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

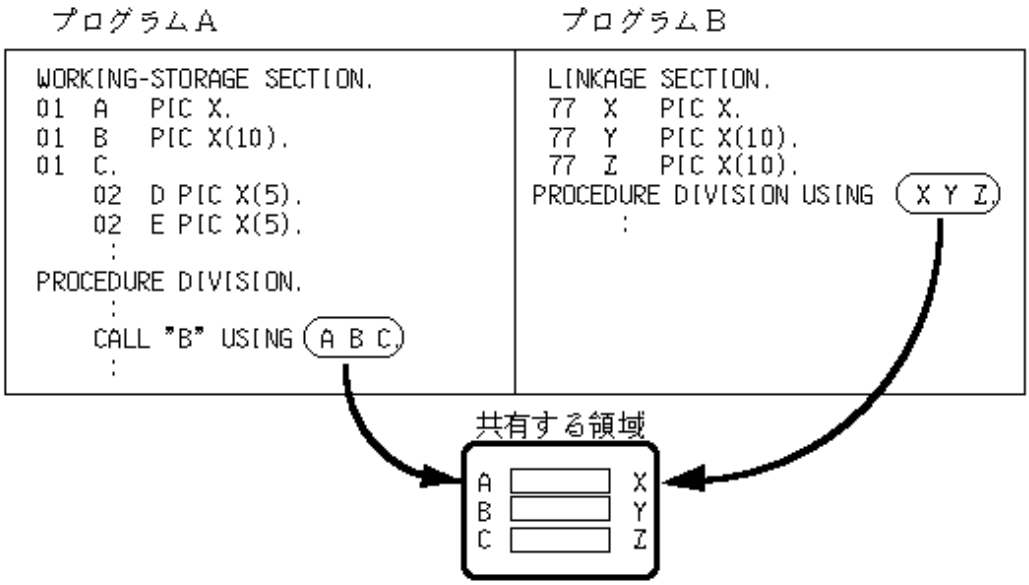
副プログラムから呼ぶプログラムに復帰するには、EXIT PROGRAM文を実行します。EXIT PROGRAM文を実行すると、呼ぶプログラムの実行したCALL文の直後に制御が戻ります。また、すべてのCOBOLプログラムの実行を終了させるには、STOP RUN文を実行します。STOP RUN文を実行すると、COBOLの主プログラムの呼出し元に制御が戻ります。

10.2.4 パラメタの受渡し方法

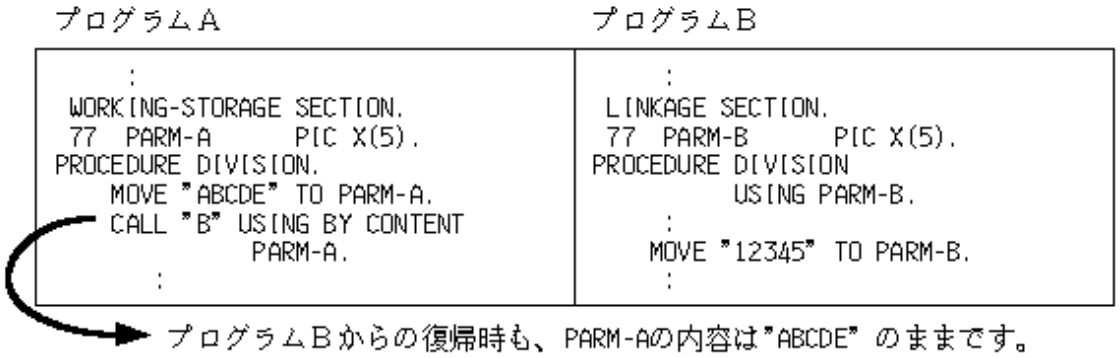
呼ぶプログラムと副プログラムの間で、パラメタを受け渡すことができます。

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。

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)

*1 ： プログラムの手続き部の見出しのUSING指定とENTRY文のUSING指定に記述したパラメタが異なる場合、誤った記述がチェックされない場合があります。

*2 ： プログラムの翻訳時に翻訳オプションCHECKの指定が必要です。内部プログラムを呼ぶCALL文は翻訳時にチェックされ、外部プログラムを呼ぶCALL文は実行時にチェックされます。詳しくは、“19.2 CHECK機能”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。

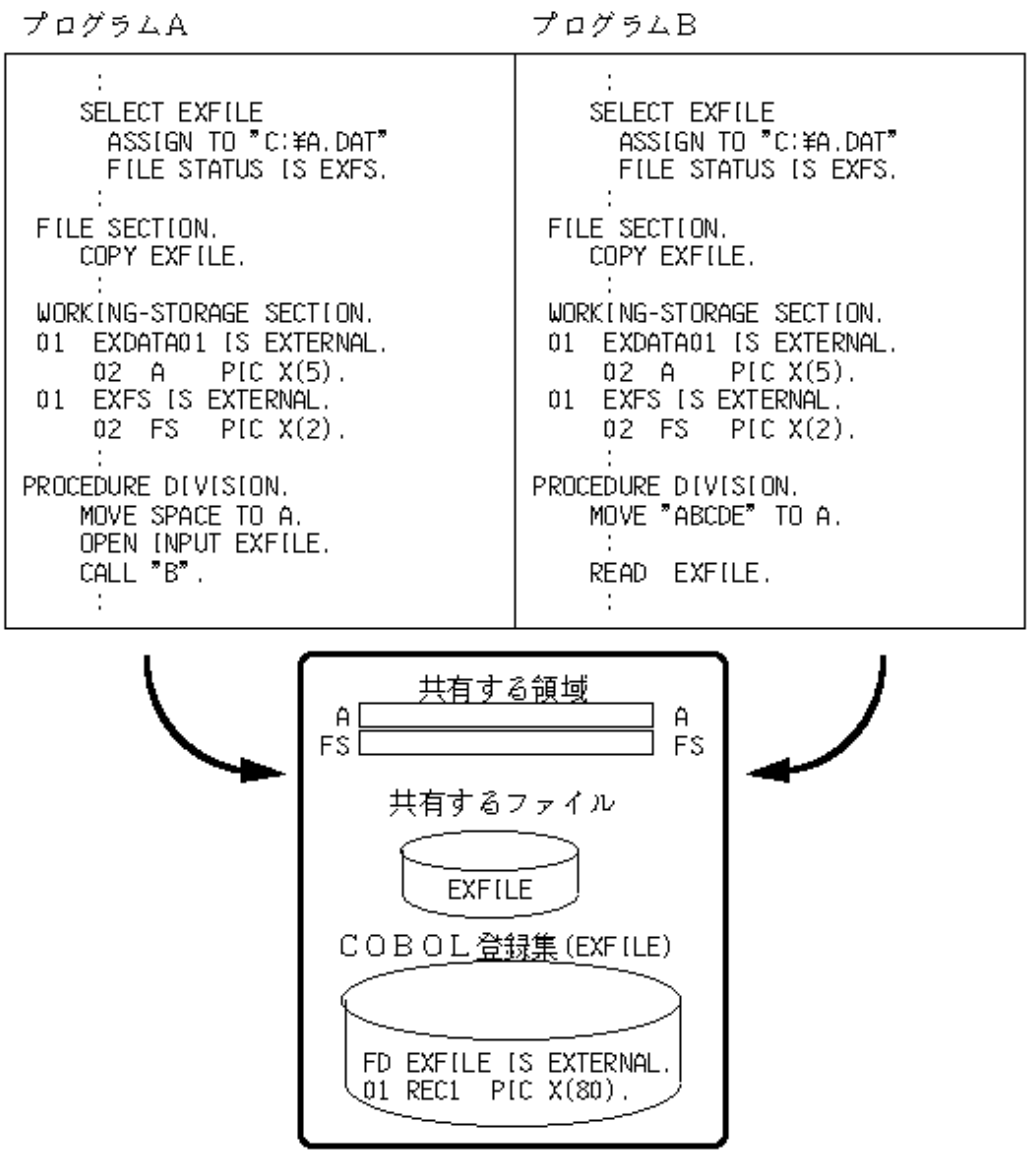
10.2.5 データの共用

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の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。



10.2.5.1 外部データ使用時の注意事項

外部データの同一性のチェックは、最大領域長および最小領域長(可変長データ項目)が対象になります。したがって、外部データが集団項目の場合などは、外部データを構成しているそれぞれのデータ項目の属性はチェックの対象ではないため、不用意に使用すると、データ例外や実行結果の異常などの原因になります。これを避けるためには、COBOL登録集を使用するなどして、データを共用するプログラムの間で同一のレコード構造になるように注意しなければなりません。

また、外部データのデータ領域は、その外部データが記述されたプログラムに一度でも制御が渡った時点で確保され、ランタイムシステムの実行単位の終了時に解放されます。すなわち、外部データのデータ領域はCANCEL文などによりプログラムを消去しても解放されません。このため、繰り返して呼び出されるプログラム内で外部データを使用する場合には、注意が必要です。

10.2.5.2 外部ファイル使用時の注意事項

- 外部ファイルは、複数のプログラムで共用できるファイルであり、OPEN文を実行したプログラムとは別のプログラムで入出力処理を行うことができます。
- 外部ファイルは、通常の外部属性を持たないファイルと同様にプログラムで扱うことができますが、外部ファイル特有の注意事項として、「共用するプログラム間でそれぞれ同一の属性で定義されていなければならない」ということがあります。複数のプログラムから1つのファイルを共用するのですから、必然的に属性の定義を一致させる必要があります。

- 外部ファイルの同一の属性とは、“COBOL文法書”の“4.3.1 ファイル管理段落(FILE-CONTROL)”の一般規則に示される項目の定義を一致させることをいいます。
- 外部ファイルの属性は、同一であるべき項目が多いため、できるだけCOBOL登録集の使用をおすすめします。なぜなら、これらの項目のチェックは実行時に行われるため、最終結合段階にエラーとなり、開発作業の手戻りが発生する可能性があるからです。
- 外部ファイルは、その外部ファイルの記述があるプログラムが一度でも実行されると、外部ファイルのレコード領域および制御用の領域はCANCEL文などによりプログラムを消去しても解放されません。これらの領域が解放されるのは、ランタイムシステムの実行単位の終了時です(通常の外部属性を持たないファイルの場合には、ファイルを記述したプログラムを消去した時点で解放されます)。このため、繰り返して呼び出されるプログラム内で外部ファイルを使用する場合には、注意が必要です。

10.2.6 復帰コード

副プログラムから呼ぶプログラムへ制御が戻るときに、RETURNING指定または特殊レジスタPROGRAM-STATUS(またはRETURN-CODE)を使用して、復帰コードを受け渡すことができます。

RETURNING指定は、利用者が定義した項目を使用して復帰コードを受け渡します。呼び出すプログラムのCALL文にRETURNING指定を記述し、副プログラムの手続き部の見出し(PROCEDURE DIVISION)にもRETURNING指定を記述します。RETURNING指定の有無、データの型および長さは、一致していなければなりません。

- プログラムA

```
*>      :
  WORKING-STORAGE SECTION.
  01 RTN-ITM  PIC S9(2) DISPLAY.
 PROCEDURE DIVISION.
*>      :
     CALL "B" RETURNING RTN-ITM.
     IF RTN-ITM NOT = 0 THEN
*>      :
```

- プログラムB

```
*>      :
  LINKAGE SECTION.
  01 RTN-CD  PIC S9(2) DISPLAY.
 PROCEDURE DIVISION RETURNING RTN-CD.
     IF エラー発生
       THEN
         MOVE 99 TO RTN-CD
       ELSE
         MOVE  0 TO RTN-CD
     END-IF.
```

注意

RETURNING指定が記述されたCALL文では、呼ぶプログラムの特殊レジスタPROGRAM-STATUSの値は変更されないことに注意してください。また、副プログラム側では、RETURNING指定に記述された項目には、値を設定しなければなりません。値が設定されていない場合、呼び出したプログラムのCALL文のRETURNING指定に記述された項目の値は、不定となります。

特殊レジスタPROGRAM-STATUSは、暗に“PIC S9(18) 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。

COBOLプログラムがSTOP RUNを実行した場合は、STOP RUN文を実行したプログラムの特殊レジスタPROGRAM-STATUSの値がCOBOLプログラムを呼び出したプログラムへの復帰値として返されます。特殊レジスタPROGRAM-STATUSを持たないメソッド手続きでSTOP RUN文を実行した場合は、復帰値として0が返されます。

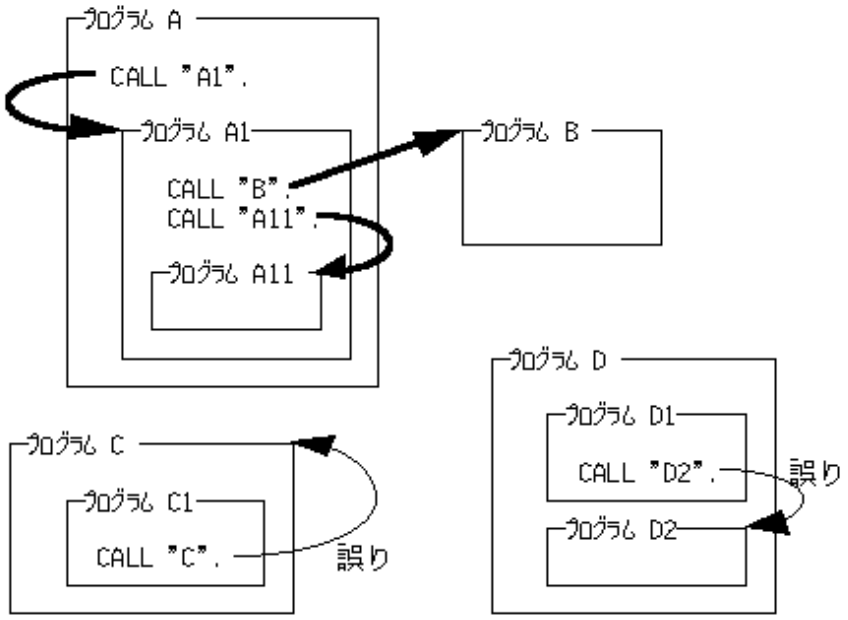
注意

COBOLプログラムでSTOP RUN文を使用する場合は、最初に呼び出される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

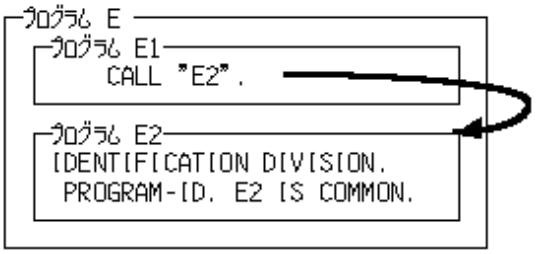
COBOLプログラムは、プログラムの構造の観点から、外部プログラムと内部プログラムに分類されます。他のプログラムに含まれない一番外側のプログラムを外部プログラムといいます。外部プログラムに直接的または間接的に含まれているプログラムを内部プログラムといいます。

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。



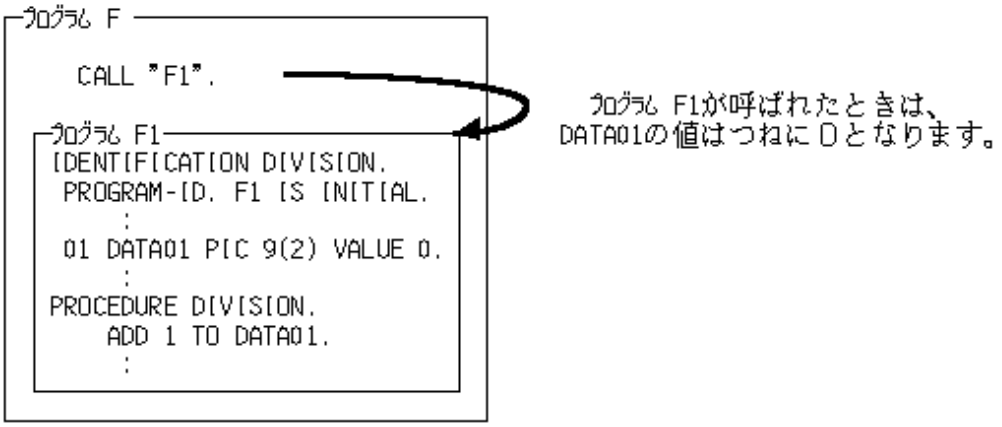
共通プログラム

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.3.1 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.3.24 MAIN(主プログラム/副プログラムの指定)"
- COBOLの実行単位内に同じプログラム名を持つ複数のプログラムが存在する場合は、動作を保証しません。
  仕様に反して、実行時に同じプログラム名を持つプログラムが呼び出された場合は、以下のように動作します。
  - 同じプログラム名だが、COBOLソースプログラムが異なる場合、JMP0032I-Uの実行時メッセージが出力されます。
  - 同じCOBOLソースプログラムだが、翻訳日付が異なる場合、JMP0032I-Uの実行時メッセージが出力されます。
  - 同一のプログラムが複数のDLLファイルに含まれている場合、前回呼び出された状態が保持できず、意図しない動作をすることがあります。

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。

- 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
            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 long int 関数名(
  構造体名        *仮引数1,
  short int       *仮引数2,
  char            *仮引数3)
{
  long int 関数値 = 0;
  //  :
  return(関数値);
}
```

10.3.1.1 呼出し方法

COBOLプログラムからCプログラムを呼び出す場合、COBOLのCALL文に関数名を指定します。呼ばれたCプログラムでreturn文を実行すると、COBOLのCALL文の直後に復帰します。

10.3.1.2 パラメタの受渡し方法

COBOLプログラムからCプログラムへパラメタを渡す場合には、CALL文のUSING指定にデータ名を記述します。Cプログラムに渡すパラメタの内容は、領域のアドレスまたはデータ名の内容となります。パラメタは、CALL文のUSING指定に記述します。以下にUSING指定の記述とパラメタの内容の関係を説明します。

BY REFERENCE データ名を指定した場合 … 領域のアドレス

COBOLプログラムがCプログラムに渡す実引数の値は、指定したデータ名の領域のアドレスとなります。Cプログラムでは、渡されるパラメタの属性に対応するデータ型を持つポインタを仮引数として宣言します。COBOLとCのデータ型の対応については、“表10.1 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"
                 USING BY REFERENCE PRM1
                       BY CONTENT   PRM2.
     DISPLAY "PRM1=" PRM1.
     DISPLAY "PRM2=" PRM2.
```

－ Cプログラム

```
#include <windows.h>

long long int SUBC(
  long int  *p1,
  long int  *p2)
{
  *p1 = *p1 +10;
  *p2 = *p2 +10;
  return(0);
}
```

ー 実行結果

```
PRM1=+000000020
PRM2=+000000010
```

上記のようなCOBOLプログラムからCプログラムを呼んだ結果は、BY REFERENCE指定のPRM1の内容が20になり、BY CONTENT指定のPRM2の内容は10のままとなります。これは、BY REFERENCE指定で受け渡すパラメタは、呼ばれたプログラムで値を変更した場合、呼ぶプログラムのデータを更新することを意味します。これに対して、BY CONTENT指定で受け渡すパラメタは、呼ばれたプログラムで値を変更しても呼ぶプログラムのデータの内容に影響を与えないことを意味します。上記の説明は、呼ばれるプログラムがCOBOLである場合も同じになります。

BY REFERENCE指定とBY VALUE指定の違い

ー COBOLプログラム

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.    MAINCOB.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
  01 PRM1   PIC S9(9) COMP-5.
  01 PRM2   PIC S9(9) COMP-5.
 PROCEDURE DIVISION.
     MOVE 10 TO  PRM1.
     MOVE 10 TO  PRM2.
     CALL "SUBC"
                 USING BY REFERENCE PRM1
                       BY VALUE     PRM2.
     DISPLAY "PRM1=" PRM1.
     DISPLAY "PRM2=" PRM2.
```

ー Cプログラム

```
#include <windows.h>

long long int SUBC(
  long int  *p1,
  long int   p2)
{
  *p1 = *p1 +10;
   p2 =  p2 +10;
  return(0);
}
```

ー 実行結果

```
PRM1=+000000020
PRM2=+000000010
```

BY REFERENCE指定がその項目のアドレスを渡す方法であるのに対して、BY VALUE指定は項目の値そのものを渡すことになります。したがって、BY VALUE指定もBY CONTENT指定と同じように、呼ばれるプログラムでその内容を変更しても呼ぶプログラムのデータの内容は変更することにはなりません。また、呼び出されるCプログラムでは、アドレスで受け取る場合と値で受け取る場合では、その記述に差異があるため、注意が必要です。

注意

BY指定を省略した場合の扱いは、BY REFERENCE指定となります。

10.3.1.3 復帰コード(関数値)

Cプログラムから復帰コード(関数値)を受け取るには、CALL文のRETURNING指定または特殊レジスタPROGRAM-STATUSを使います。

RETURNING指定

RETURNING指定に記述する項目の属性は、USING指定に記述する項目と同様にCのデータ型と対応がとれている必要があります。データ型の対応については、“10.3.3 データ型の対応”を参照してください。

- COBOLプログラム

```
  WORKING-STORAGE SECTION.
  01 AGRP.
     02  AITEM1 PIC X(10).
     02  AITEM2 PIC X(20).
  77 B          PIC S9(4) COMP-5.
  01 RTN-ITM    PIC S9(4) COMP-5.
 PROCEDURE DIVISION.
     CALL "C"
              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"
              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 long int型の関数として記述する必要があります。

- 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.
 PROCEDURE DIVISION.
     CALL "C"
              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 long int C(agrp *agrpp, short int *b)
{
  return(0);
}
```

long long int型以外のCプログラムの関数値を受け取る場合

long 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 long int型に対応します。したがって、long int型や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(18) 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(18) 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 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
                    USING データ名1 データ名2.
```

```
      :
    MOVE 0 TO PROGRAM-STATUS.
    EXIT PROGRAM.
END PROGRAM プログラム名.
```

10.3.2.1 呼出し方法

CプログラムからCOBOLプログラムを呼び出すには、Cの関数呼出しの形式でCOBOLのプログラム名を指定します。呼ばれたCOBOLプログラムでEXIT PROGRAM文を実行すると、Cプログラムの関数呼出しの直後に復帰します。

10.3.2.2 パラメタの受渡し方法

CプログラムからCOBOLプログラムへ引数を渡す場合には、Cの関数呼出しで実引数を指定します。CプログラムからCOBOLプログラムへ渡すことのできる実引数の値は、記憶領域のアドレスでなければなりません。COBOLプログラムでは、手続き部の見出しまたはENTRY文のUSING指定にデータ名を記述することにより、実引数に指定したアドレスにある領域の内容を受け取ります。実引数で指定したアドレスにある変数の宣言または定義でCONST型指定子を指定した場合、実引数に指定したアドレスにある領域の内容を変更してはいけません。

10.3.2.3 復帰コード(関数値)

手続き部の見出し(PROCEDURE DIVISION)のRETURNING指定の項目や特殊レジスタPROGRAM-STATUSに設定した値は、Cプログラムに関数値として渡ります。

RETURNING指定

手続き部の見出しのRETURNING指定に記述する項目は、Cプログラムのデータ型(下図の[1]および[2])と対応していなければなりません。データ型の対応については、“10.3.3 データ型の対応”を参照してください。

・ Cプログラム

```
C()
{
  typedef struct
  { char aitem1[10];
    char aitem2[20];
  }agrp;
  extern void JMPCINT2();
  extern void JMPCINT3();
  extern short int COB(agrp *agrpp, short int *b);  // [1]
  agrp      prm1;
  short int prm2;

  JMPCINT2();
  if (COB(&prm1, &prm2)==0){
    printf("Return Code = 0¥n");
  }
  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
                   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 long int型(下図の[3])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 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
                    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10.1 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10.1 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)	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
	long long int	77 K PIC S9(18) COMP-5 あるいは 77 K USAGE IS BINARY-DOUBLE SIGNED.	long long int K;	8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) 内部浮動小数点(倍精度)	float double	01 I COMP-1. 01 J COMP-2.	float I; double J;	4バイト 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, long intまたはlong 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

－ 10～18桁(BINARY(BYTE)オプション指定時は17～18桁) ： long 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"
              USING FLOAT.
```

－ Cプログラム

```
long long int C(float *h)
{
    :
  return(0);
}
```

・ **注3**

Cプログラムのshort int, long intまたはlong 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, USAGE IS BINARY-LONG SIGNEDまたはUSAGE IS BINARY-DOUBLE SIGNEDの項目を使用してください。

・ **注4**

集団項目を構造体として宣言するときには、その構造体に含まれる変数の記憶領域の境界に注意する必要があります。COBOLのデータ項目の記憶領域の境界調整については、“COBOL文法書”を参照してください。また、Cの変数の記憶領域の境界調整については、C言語のマニュアルを参照してください。

参考

・ 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" long long int 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.
```

ー 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" long long int 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.3.1 ALPHAL(英小文字の扱い)”

例

```
CALL "abc".
```

ー 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。

ー 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.
```

ー 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.
```

ー 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。

ー 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、F4AGCIMP.LIBが必要になります。
- COBOLプログラムとCプログラムを結合するときは、それぞれの開発環境で実行可能ファイルまたはDLLを作成することを推奨します。

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、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。

- Cプログラム

```
#include <windows.h>

extern void JMPCINT2(),JMPCINT3();
extern long long int  COBSUB(int *p);
int WINAPI WinMain( ～ )
{
    int prm1;
       :
    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  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 F4AGCIMP.LIB LIBCMT.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
F4AGCIMP.LIB　：　COBOLランタイムシステムのインポートライブラリ
LIBCMT.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 F4AGCIMP.LIB /DLL /NOENTRY /OUT:COBSUB.DLL
```

COBSUB.OBJ　　：　COBOLプログラムCOBSUBのオブジェクトファイル
F4AGCIMP.LIB　：　COBOLランタイムシステムのインポートライブラリ

－ 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 F4AGCIMP.LIB LIBCMT.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
F4AGCIMP.LIB　：　COBOLランタイムシステムのインポートライブラリ
LIBCMT.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();
long long int (far *COBSUB)(int *p);
int WINAPI WinMain(～)
{
  int prm1;
  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 F4AGCIMP.LIB /DLL /NOENTRY /OUT:COBSUB.DLL
```

COBSUB.OBJ　　：　COBOLプログラムCOBSUBのオブジェクトファイル
F4AGCIMP.LIB　：　COBOLランタイムシステムのインポートライブラリ

― 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 F4AGCIMP.LIB LIBCMT.LIB /OUT:Cprog.EXE
```

Cprog.obj　　:　Cプログラムのオブジェクトファイル
F4AGCIMP.LIB　:　COBOLランタイムシステムのインポートライブラリ
LIBCMT.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、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。

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。

・ 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)
long long int 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 F4AGCIMP.LIB LIBCMT.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
F4AGCIMP.LIB　:　COBOLランタイムシステムのインポートライブラリ
LIBCMT.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 LIBCMT.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
LIBCMT.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 F4AGCIMP.LIB LIBCMT.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
F4AGCIMP.LIB　：　COBOLランタイムシステムのインポートライブラリ
LIBCMT.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 LIBCMT.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
LIBCMT.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 F4AGCIMP.LIB LIBCMT.LIB /OUT:MAINCOB.EXE
```

MAINCOB.OBJ　：　COBOLプログラムMAINCOB のオブジェクトファイル
F4AGCIMP.LIB　：　COBOLランタイムシステムのインポートライブラリ
LIBCMT.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。
- 呼出しの途中に、JavaやC#で作成されたプログラムが存在する場合、STOP RUN文を実行してプログラムを終了してはいけません。

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。

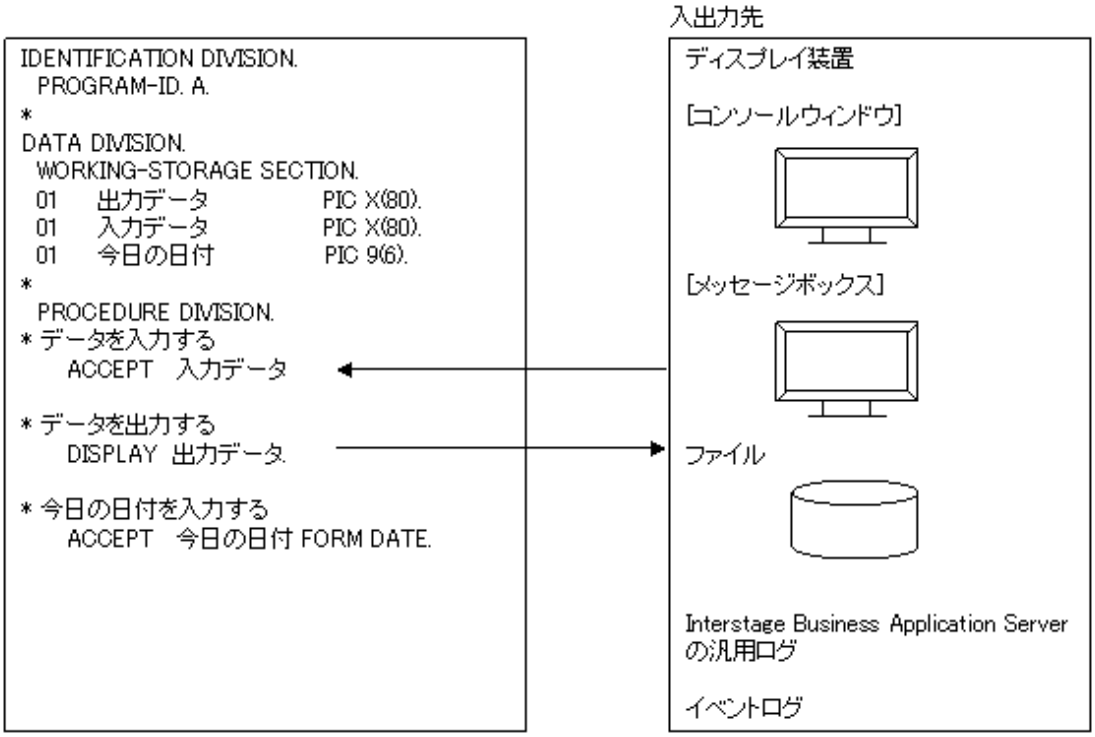
11.1 小入出力機能

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 ・機能名SYSINに対応付けた呼び名 ・機能名SYSOUTに対応付けた呼び名	なし	－	ディスプレイ装置（コンソールウィンドウ）(注2)
		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)
		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_SYSOUT_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。

注意

行順ファイルで作成するファイルは、レコード定義の字類が日本語で統一されている場合は、UTF-16となります。行順ファイルとして作成したファイルに対して小入出力機能を使用する、またはその逆の場合、コード系の違いに注意してください。

[参照]“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。

注意

- 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。

属性	環境変数情報	設定値	意味
		OFF	メッセージを表示しないで消去します。
ウィンドウの大きさ	@CnslWinSize	(m,n)	ウィンドウの大きさを、桁数(m) と行数(n)で指定します。
保持するデータの行数	@CnslBufLine	桁数	指定した行数のデータが保持され、ウィンドウの縦スクロールにより参照することができます。
コンソールウィンドウのフォント	@CnslFont	(フォント名,フォントサイズ)	コンソールウィンドウのフォントを指定できます。

注 ： この環境変数情報は、スクリーン操作機能で使用するウィンドウに対しても有効になります。

11.1.4.2 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り口関数をmain型(main関数)で作成して、環境変数情報@CBR_CONSOLE=SYSTEMを指定します。

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のと同時に、OPEN中ファイルのCLOSE処理などの資産の回収処理が行われない状態でCOBOLアプリケーションが強制的に終了します。

11.1.4.3 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〕ー“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のと同時に、OPEN中ファイルのCLOSE処理などの資産の回収処理が行われない状態でCOBOLアプリケーションが強制的に終了します。
- MAIN(WINMAIN)またはMAINオプションで翻訳したCOBOLアプリケーションでシステムのコンソール(コマンドプロンプトを除く)を使用する場合、およびWinMain型の他の言語からCOBOLを呼び出してシステムのコンソール(コマンドプロンプトを除く)を使用する場合、アプリケーションをコマンドプロンプトから起動すると正常に動作しないことがあります。このような場合は、STARTコマンドを使用してください。

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要求されるので、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）"

注意

- 翻訳オプションMAIN(MAIN)を指定した場合は、実行環境情報@CBR_CONSOLE=COBOLを指定してください。

・ 入出力先に仮想デバイス(NULなど)を使用することはできません。

11.1.6 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)

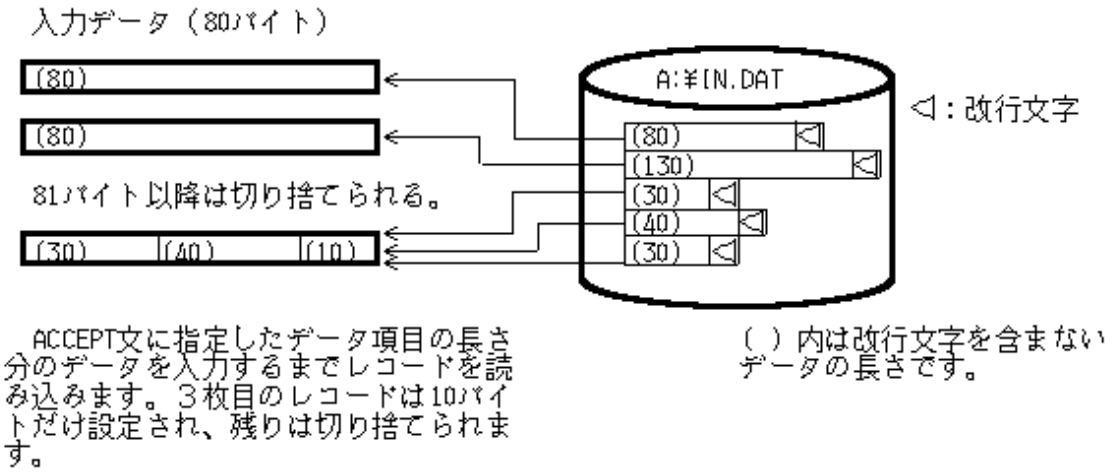
注 ： ACCEPT文では使用不可

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。

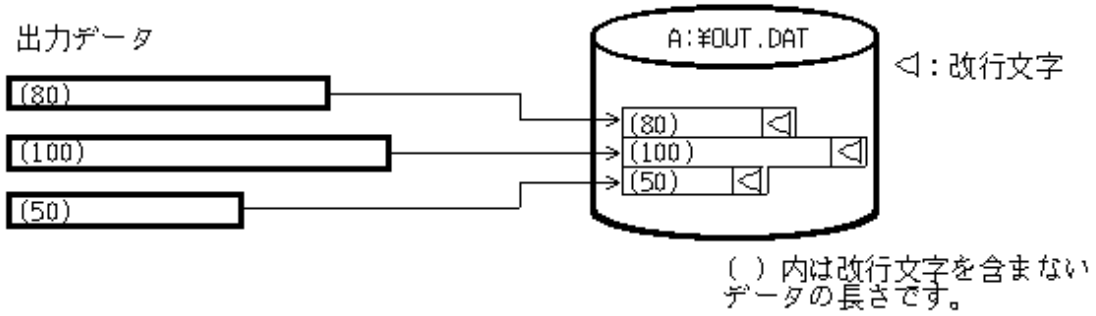
データの入力

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.3.48 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.3.49 SSOUT（DISPLAY文のデータの出力先）”

例

```
SSIN(INDATA),SSOUT(OUTDATA)
```

注意

- 翻訳オプションSSINに環境変数情報名としてSYSINを指定した場合、ACCEPT文のデータ入力先は、環境変数情報SYSINの設定にかかわらず、コンソールウィンドウとなります。

- 翻訳オプションSSOUTに環境変数情報名としてSYSOUTを指定した場合、DISPLAY文のデータ出力先は、環境変数情報SYSOUTの指定にかかわらず、コンソールウィンドウとなります。
- FROM指定またはUPON指定の記述に機能名CONSOLEに対応付けた呼び名を指定した場合、データの入出力先は、翻訳オプション(SSIN/SSOUT)の指定にかかわらず、コンソールウィンドウになります。

11.1.6.3 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とみなされます。
- コンマ(,) に続き“MOD”または“DUMMY”以外の文字列を指定した場合、最初のDISPLAY文実行時でエラーになります。

11.1.6.5 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

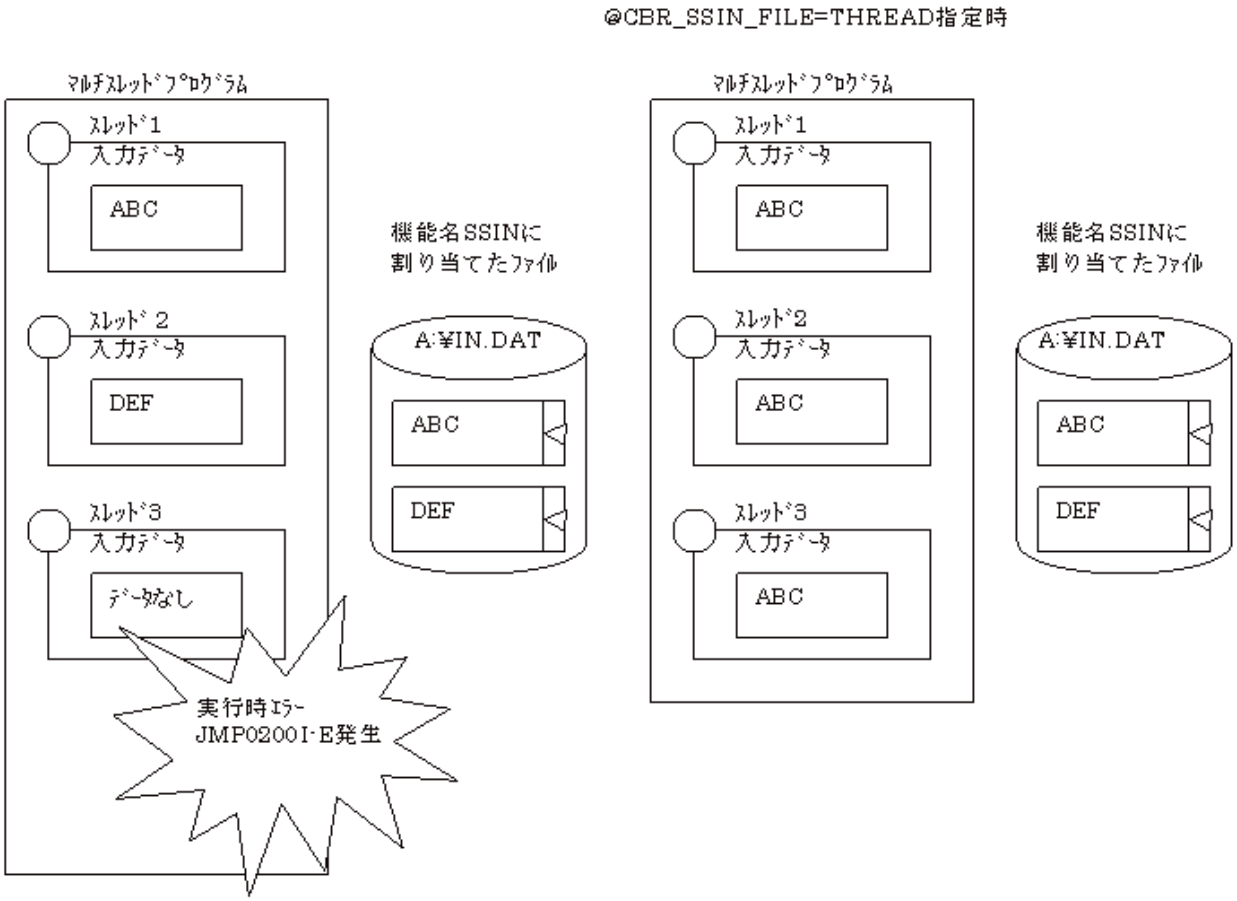
- ファイル名にコンマ(,)を含む場合、ファイル名を二重引用符(")で囲む必要があります。
- 文字列“DUMMY”の前にコンマ(,)が必要です。コンマ(,)がない場合、文字列“DUMMY”をファイル名とみなして、通常のファイルと同じ動作をします。
- ファイル名の指定の有無による動作の違いはありません。指定したファイル名は意味を持ちません。
- コンマ(,)に続き、“DUMMY”以外の文字列を指定した場合、最初のACCEPT文の実行でエラーになります。

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。



使い方

実行環境変数@CBR_SSIN_FILEに、“THREAD”を指定します。[参照]“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)”

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
```

注意

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)
```

注意

- 一度に出力できるデータは1024文字です。それ以降は出力されません。
- Interstage Business Application Serverの汎用ログと同時に使用することはできません。
- イベントログに出力する場合、他の出力先(コンソール、ファイル等)の指定は無効となり出力されません。
- イベントログ出力時のイベントソース名は“NetCOBOL Application x64”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 }
  ```

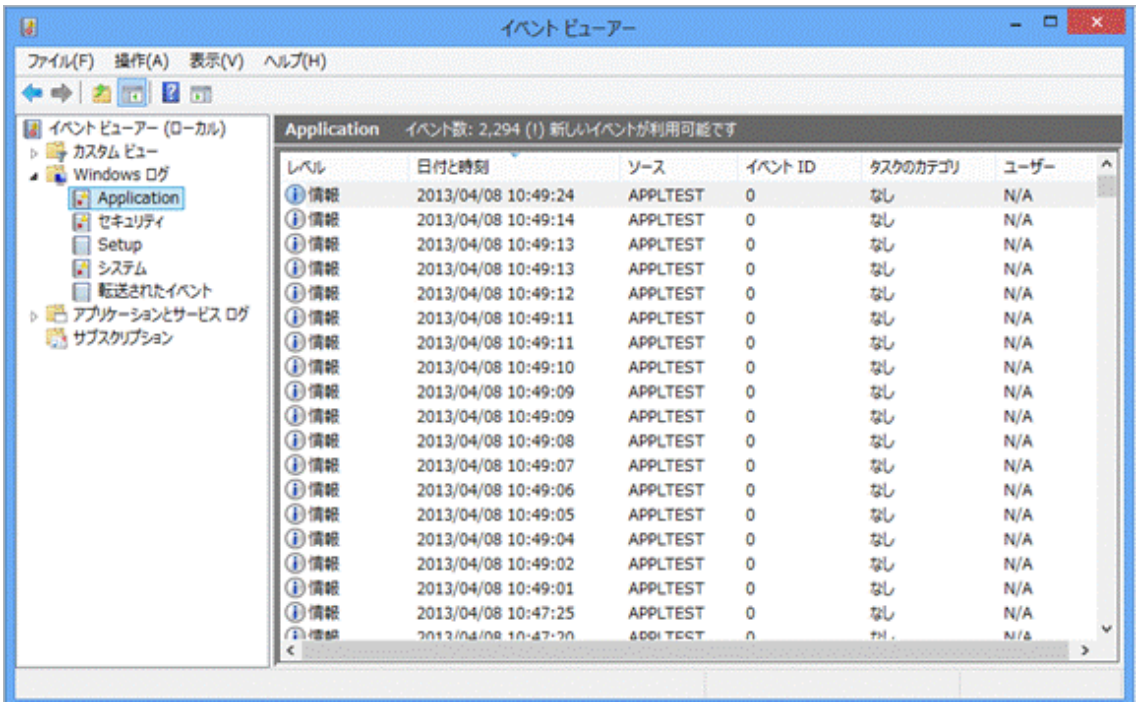
  レベルは以下を意味します。

  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“17.2.2 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。

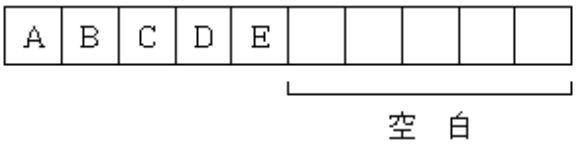
```
DISPLAY  5                UPON 呼び名１.              … [1]
ACCEPT   引数の値(5)      FROM 呼び名２               … [2]
    ON EXCEPTION MOVE  5 TO  誤り番号                 … [3]
                 GO TO 誤り処理
END-ACCEPT.
```

- 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。

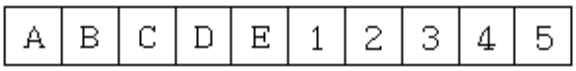
```
      :
01  個数    PIC 9.
01  引数    PIC X(10).
      :
    ACCEPT   個数 FROM 引数の番号.                    … [1]
    ACCEPT   引数 FROM 引数の値.                      … [2]
      :
```

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。



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。



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った場合、例外条件が発生します。例外条件が発生すると、DISPLAY文にON EXCEPTIONが指定されている場合にはそこに指定された文([6])が実行されます。

```
DISPLAY "TMP1"        UPON   呼名ー環境変数名.          … [1]
ACCEPT   ＴＭＰ１の値 FROM   呼名ー環境変数の値        … [2]
    ON EXCEPTION                                      … [3]
         MOVE   エラー発生 TO   ～
END-ACCEPT.
    :
DISPLAY "TMP2"        UPON   呼名ー環境変数名.          … [4]
DISPLAY  ＴＭＰ２の値 UPON   呼名ー環境変数の値        … [5]
    ON EXCEPTION                                      … [6]
         MOVE   エラー発生 TO   ～
END-DISPLAY.
```

注意

環境変数の値の長さを得ることはできません。

11.3.3 プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

11.3.4 プログラムの実行

通常のプログラムを実行するときと同様に実行してください。

注意

プログラムの実行中に変更した環境変数の値は、そのプログラムの実行しているプロセス内でだけ有効となります。

第12章　SORT文およびMERGE文の使い方～整列併合機能～

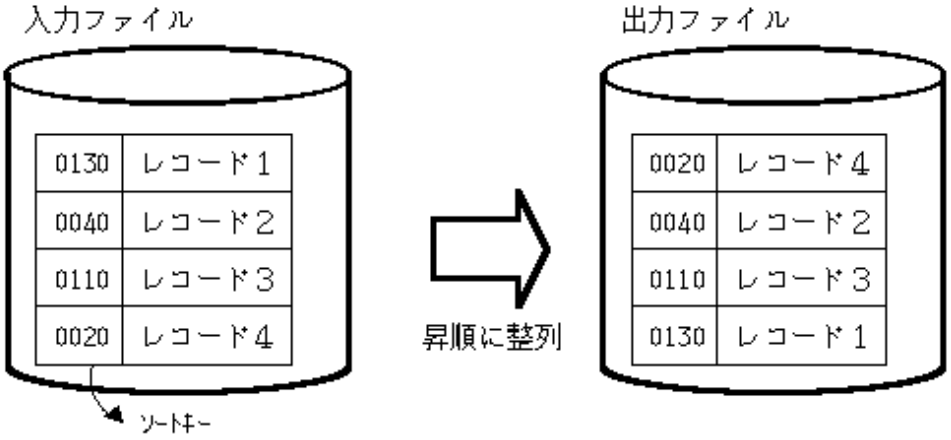
ファイルのレコードを一定の順序に並べ替えることをソート(整列)といい、複数のファイルを1つのファイルに並べ替えることをマージ(併合)といいます。本章では、ソート・マージ(整列・併合)機能について説明します。

12.1 ソート・マージ処理の概要

ここでは、ソート(整列)処理と、マージ(併合)処理の概要を説明します。

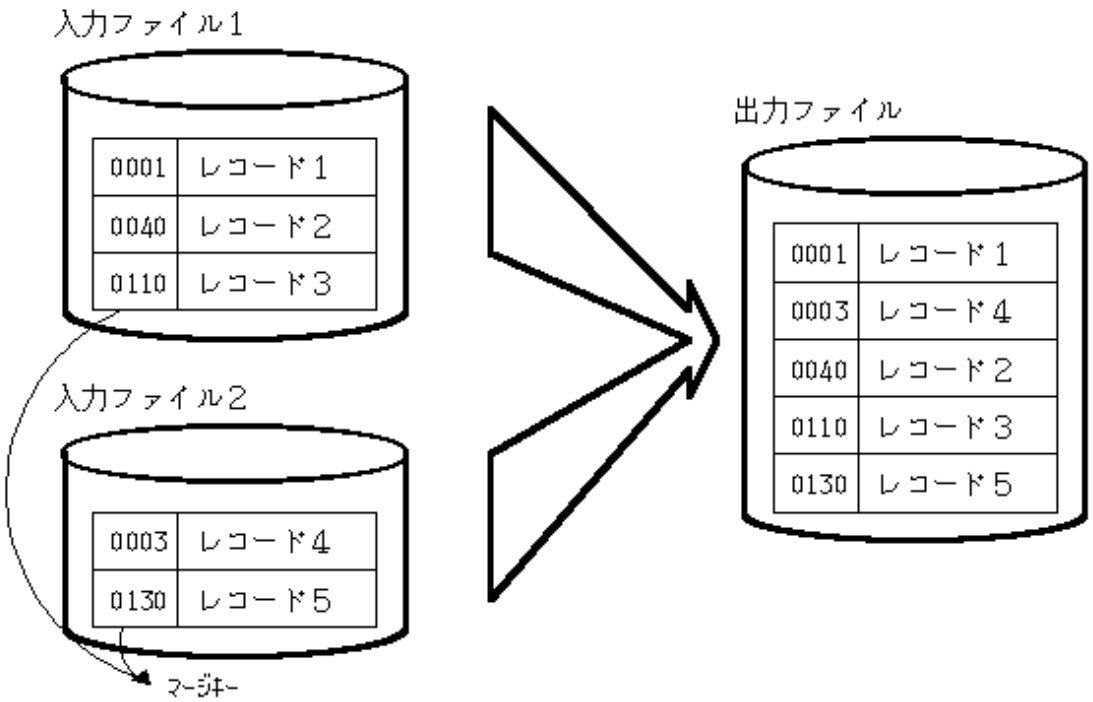
ソート

ソートとは、ファイル中のレコードの情報をキーとして、レコードを昇順または降順に並べ替える処理です。レコードの並べ替えは、プログラムのキー項目の属性に従って行われます。



マージ

マージとは、昇順または降順に整列(ソート)された複数のファイルを1つのファイルにまとめることです。



12.2 ソートの使い方

ここでは、ソート処理の種類、プログラムの書き方、翻訳・リンクおよび実行方法について説明します。

12.2.1 ソート処理の種類

ソート処理には、次の4種類の方法があります。

	方法	入力	出力
[1]	入力ファイルのレコードすべてを昇順または降順に出力ファイルに出力する方法	ファイル	ファイル
[2]	入力ファイルのレコードすべてを昇順または降順に出力データとして扱う方法	ファイル	レコード
[3]	特定のレコードまたはデータを昇順または降順に出力ファイルに出力する方法	レコード	ファイル
[4]	特定のレコードまたはデータを昇順または降順に出力データとして扱う方法	レコード	レコード

通常、入力ファイル(ソートするファイル)のレコードの内容を変更しないで、そのままソートする場合は、[1]または[2]を使います。
入力ファイルを使用しない場合やレコードの内容の変更を行う場合は、[3]または[4]を使います。

また、ソートしたレコードの内容を変更しないで、そのまま出力ファイルに書き出す場合は、[1]または[3]を使います。
出力ファイルを使用しない場合やレコードの内容を変更する場合は、[2]または[4]を使います。

12.2.2 プログラムの記述

ここでは、ソートを使うプログラムの記述内容について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.   プログラム名.
 ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
   INPUT-OUTPUT SECTION.
   FILE-CONTROL.
      SELECT  ソートファイル１ ASSIGN TO  SORTWK01.
```

```
    [SELECT  入力ファイル１   ASSIGN TO  ファイル参照子１   ～.]
    [SELECT  出力ファイル１   ASSIGN TO  ファイル参照子２   ～.]
DATA DIVISION.
 FILE SECTION.
 SD  ソートファイル１
    [RECORD  レコードの大きさ].
 01  ソートレコード１.
     02  ソートキー１   ～.
     02  データ１       ～.
 FD  入力ファイル１   ～.
 01  入力レコード１.
       レコード記述項
 FD  出力ファイル１   ～.
 01  出力レコード１.
       レコード記述項
PROCEDURE DIVISION.
    SORT   ソートファイル１ ON
        { ASCENDING   | DESCENDING }  KEY  ソートキー１
          { USING  入力ファイル１   | INPUT  PROCEDURE IS  入力手続き１}
          { GIVING 出力ファイル１   | OUTPUT PROCEDURE IS  出力手続き１} .
   入力手続き１ SECTION.
      OPEN INPUT   入力ファイル１.
   入力開始.
      READ   入力ファイル１ AT END GO TO 入力終了.
      MOVE   入力レコード１ TO ソートレコード１.
      RELEASE  ソートレコード１.
      GO TO  入力開始.
   入力終了.
      CLOSE  入力ファイル１.
   入力開始終了.
      EXIT.
   出力手続き１ SECTION.
      OPEN OUTPUT  出力ファイル１.
   出力開始.
      RETURN ソートファイル１ AT END GO TO 出力終了 END-RETURN.
      MOVE   ソートレコード１ TO 出力レコード１.
      WRITE  出力レコード１.
      GO TO  出力開始.
   出力終了.
      CLOSE  出力ファイル１.
   出力開始終了.
      EXIT.
END PROGRAM  プログラム名.
```

環境部(ENVIRONMENT DIVISION)

以下のファイルを定義します。

整列併合用ファイル

ソート処理を行うための作業ファイルを定義します。英字で始まる8文字以内の英数字を指定する必要があります。なお、ASSIGN句は注釈とみなされます。

同じプログラムでマージ処理を行う場合、整列併合用ファイルの定義は1つだけ行います。

入力ファイル

ソート処理で入力ファイルを使用するときには、通常のファイル処理を行うときと同様にファイルを定義します。

出力ファイル

ソート処理で出力ファイルを使用するときには、通常のファイル処理を行うときと同様にファイルを定義します。

データ部(DATA DIVISION)

環境部に定義したファイルのレコードを定義します。

手続き部(PROCEDURE DIVISION)

ソート処理には、SORT文を使います。ソート処理の入力と出力がファイルかレコードかによって、SORT文の記述内容が異なります。

入力がファイルの場合

“USING 入力ファイル名”を記述します。

入力がレコードの場合

“INPUT PROCEDURE 入力手続き名”を記述します。

出力がファイルの場合

“GIVING 出力ファイル名”を記述します。

出力がレコードの場合

“OUTPUT PROCEDURE 出力手続き名”を記述します。

INPUT PROCEDUREで指定した入力手続きでは、RELEASE文を使って、ソートするレコードを1件ずつ受け渡します。

OUTPUT PROCEDUREで指定した出力手続きでは、RETURN文を使って、ソート済みのレコードを1件ずつ受け取ります。

ソートキーは複数指定することができます。

ソート処理が終了すると、ソート処理の結果が特殊レジスタSORT-STATUSに設定されます。特殊レジスタSORT-STATUSは、COBOLコンパイラによって自動的に生成されるので、COBOLプログラムの中で定義する必要はありません。SORT文の実行後に特殊レジスタSORT-STATUSの値を検査することにより、ソート処理が異常終了しても、COBOLプログラムの実行を続行させることができます。また、SORT文で指定した入力手続きまたは出力手続きの中で、SORT-STATUSに16を設定することにより、ソート処理を終了させることもできます。

下表に、特殊レジスタSORT-STATUSに設定される値と意味を示します。

表12.1 SORT-STATUSに設定される値と意味

値	意味
0	正常終了
16	異常終了

注意

USING指定の入力ファイルおよびGIVING指定の出力ファイルを使用する場合、SORT文実行時にそれらのファイルが開かれた状態であってはいけません。

特殊レジスタSORT-CORE-SIZEを用いると、PowerSORTが使用するメモリ空間の容量を限定することができます。この特殊レジスタは、暗にPIC S9(8) COMP-5で定義された数字項目です。設定する値は、バイト単位の数値です。

この特殊レジスタは、翻訳オプションSMSIZE()および実行時オプションsmsizeに指定する値の意味と等価ですが、同時に指定された場合の優先順位は、特殊レジスタSORT-CORE-SIZEが一番強く、以降、実行時オプションsmsize、翻訳オプションSMSIZE()の順で弱くなります。

例

```
特殊レジスタ      MOVE 102400 TO SORT-CORE-SIZE   (102400=100キロです)
翻訳オプション    SMSIZE(500K)
実行時オプション  smsize300k
```

この場合、一番強い特殊レジスタSORT-CORE-SIZEの値 100キロバイトを優先します。

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.3.14 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。

同じプログラムでソート処理を行う場合、整列併合用ファイルの定義は1つだけ行います。

入力ファイル

マージ処理の入力ファイルは、通常のファイル処理を行うときと同様にファイルを定義します。

出力ファイル

マージ処理で出力ファイルを使用するときには、通常のファイル処理を行うときと同様にファイルを定義します。

データ部(DATA DIVISION)

環境部に定義したファイルのレコードを定義します。

手続き部(PROCEDURE DIVISION)

マージ処理には、MERGE文を使います。マージ処理の出力がファイルかレコードかによって、MERGE文の記述内容が異なります。

出力がファイルの場合

“GIVING 出力ファイル名”を記述します。

出力がレコードの場合

“OUTPUT PROCEDURE 出力手続き名”を記述します。

OUTPUT PROCEDUREで指定した出力手続きでは、RETURN文を使って、マージ済みのレコードを1件ずつ受け取ります。

マージキーは複数指定することができます。

マージ処理が終了すると、マージ処理の結果が特殊レジスタSORT-STATUSに設定されます。特殊レジスタSORT-STATUSは、COBOLコンパイラによって自動的に生成されるので、一般のデータとは異なり、COBOLプログラムの中で定義する必要はありません。MERGE文の実行後に特殊レジスタSORT-STATUSの値を検査することにより、マージ処理が異常終了しても、COBOLプログラムの実行を続行させることができます。また、MERGE文で指定した出力手続きの中で、特殊レジスタSORT-STATUSに16を設定することにより、マージ処理を終了させることができます。

下表に、特殊レジスタSORT-STATUSに設定される値と意味を示します。

表12.2 SORT-STATUSに設定される値と意味

値	意味
0	正常終了
16	異常終了

入力ファイルおよびGIVING指定の出力ファイルは、MERGE文実行時にそれらのファイルが開かれた状態であってはいけません。

特殊レジスタSORT-CORE-SIZEを用いると、PowerSORTが使用するメモリ空間の容量を限定することができます。この特殊レジスタは、暗にPIC S9(8) COMP-5で定義された数字項目です。設定する値は、バイト単位の数値です。

この特殊レジスタは、翻訳オプションSMSIZE()および実行時オプションsmsizeに指定する値の意味と等価ですが、同時に指定された場合の優先順位は、特殊レジスタSORT-CORE-SIZEが一番強く、以降、実行時オプションsmsize、翻訳オプションSMSIZE()の順で弱くなります。

例

```
特殊レジスタ       MOVE 102400 TO SORT-CORE-SIZE   (102400=100キロです)
翻訳オプション     SMSIZE(500K)
実行時オプション   smsize300k
```

この場合、一番強い特殊レジスタSORT-CORE-SIZEの値 100キロバイトを優先します。

注意

この特殊レジスタは、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

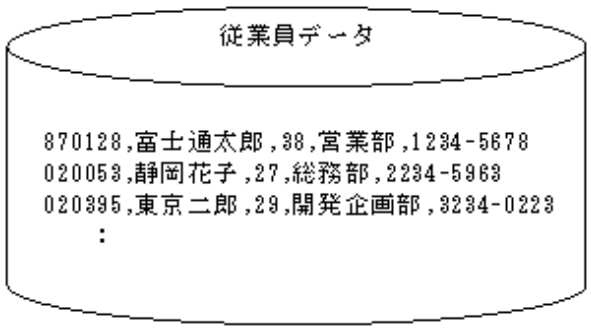
本章では、STRING文およびUNSTRING文を使用したCSV形式データの操作について説明します。

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います。

例えば、以下のようにテキストファイルで流通します。



これは、従業員番号、氏名、年齢、所属、内線をカンマで区切ったCSVデータです。

このデータをCOBOLプログラムで入力および編集する場合、行順ファイルとしてレコード単位に読み込み、カンマで区切られた文字列データを集団項目に従属する基本項目に分解して操作するのが一般的です。しかし、従来の言語仕様で実現するのは容易ではありませんでした。

上記の従業員データを以下の集団項目へ転記する場合を考えます。

```
01 従業員.
  02 従業員番号  PIC 9(6).
  02 氏名        PIC N(10).
  02 年齢        PIC 9(2).
  02 所属        PIC N(10).
  02 内線        PIC X(10).
```

うちPERFORM文を用いてCSVデータを先頭から1文字ずつ検査しながら転記することもできますが、ここでは、UNSTRING文(書き方1)を使用した場合で考えます。

```
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。
  - ー 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。
  - ー 送出し側項目が符号つき数字項目の場合、SIGN句の指定に関わらず、符号を左端に付加します。また、小数部を含む場合、小数点文字を付加します。
- 送出し側データ中に区切り文字が含まれていた場合、データ全体を二重引用符で囲みます。
- 送出し側データ中に二重引用符が含まれていた場合、連続する2つの二重引用符に置き換え、データ全体を二重引用符で囲みます。
- 送出し側項目の字類が英数字または日本語の場合、後置空白は削除します。
- 送出し側項目が数字項目の場合、先行するゼロ列は削除します。ただし、値がゼロだった場合は、1けたのゼロを転記します。また、小数部を含む場合、後置ゼロは削除します。

・ TYPE指定に従い、データ全体を二重引用符で囲みます。
詳細は、“13.4 CSV形式のバリエーション”を参照してください。

STRING文の文法や仕様の詳細は、“COBOL文法書”を参照してください。

注意

・ 受取り側への転記処理は、STRING文が作用した部分しか実行されません。したがって、文字転記のような後続領域への空白づめは期待できません。
受取り側項目は、STRING文を実行する前に必ず初期化してください。

・ List Creatorでは、CSV形式データ中に2つの二重引用符が含まれていた場合でも、1つの二重引用符に置き換えません。

13.2.2 処理異常の検出

CSV形式データ作成時における異常とは、以下の状態を指します。

表13.1 CSV形式データ作成時における異常

(1)	送出し側項目に不正なデータが格納されている
(2)	受取り側項目が小さく、全てのデータが入り切らない
(3)	POINTER指定のデータ項目の値が1より小さい

STRING文では、ON OVERFLOW指定を記述することによって異常発生時の動作を記述することができます。このとき、正常に処理できたところまで、受取り側に格納されます。

例えば、前述の例で受取り側領域の大きさが20けたしか用意されてなかった場合、以下のようにON OVERFLOW指定を記述すると、異常を検知することができます。

```
WORKING-STORAGE SECTION.
77 従業員編集   PIC X(20).
01 従業員.
  02 従業員番号  PIC 9(6).  *> 870128      が格納されている状態
  02 氏名        PIC N(10). *> 富士通太郎        (以下同)
  02 年齢        PIC 9(2).  *> 38
  02 所属        PIC N(10). *> 営業部
  02 内線        PIC X(10). *> 1234-5678
       :
MOVE SPACE TO 従業員編集.
STRING 従業員 INTO 従業員編集 BY CSV-FORMAT
      ON OVERFLOW DISPLAY "編集に失敗しました。データ=" 従業員編集
END-STRING.
```

なお、ON OVERFLOW指定が記述されてない場合は、“表13.1 CSV形式データ作成時における異常”が発生した時、以下のように動作します。

異常(1),(3)の場合

実行時エラーを出力後、異常終了します。

異常(2)の場合

実行時エラーを出力後、STRING文の次の文へ制御を移します。

13.3 CSV形式データの分解(UNSTRING文)

ここでは、CSV形式データを分解する方法を説明します。

13.3.1 基本操作

CSV形式データを分解して、集団項目に従属する項目へ転記する場合、UNSTRING文(書き方2)を使用します。

```
WORKING-STORAGE SECTION.
77 従業員データ   PIC X(80)
                  VALUE "870128,富士通太郎,38,営業部,1234-5678".
01 従業員.
  02 従業員番号   PIC 9(6).
  02 氏名         PIC N(10).
  02 年齢         PIC 9(2).
  02 所属         PIC N(10).
  02 内線         PIC X(10).
       :
UNSTRING 従業員データ(1:FUNCTION STORED-CHAR-LENGTH(従業員データ))
         INTO 従業員 BY CSV-FORMAT.
```

上の例のUNSTRING文を実行すると、集団項目“従業員”に従属する各基本項目には、先頭から順番にカンマで分割されたデータが格納されます。

なお、CSV形式データの分解では、データを以下のとおり編集します。

- 送出し側項目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	・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。 ・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。
MODE-2	・送出し側データを二重引用符で囲みます。 ・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。
MODE-3	・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。 ・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。

バリエーション	内容
MODE-4	・ 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。 ・ 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。

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“C.2.14 @CBR_CSV_TYPE（生成するCSV形式のバリエーション）”で指定します。なお、省略時は、MODE-1が選択されたものとみなします。

以下に例を示します。

```
01 従業員.
  02 従業員番号  PIC 9(6)  VALUE 870128.
  02 氏名        PIC N(10) VALUE 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ときのパラメタが領域アドレスの場合、COBOLプログラム中では、ポインタ付けを使って、そのアドレスを持つ領域の内容を参照または更新することができます。

ADDR関数およびLENG関数

ADDR関数では、COBOLで定義したデータ項目のアドレスを得ることができます。また、LENG関数では、COBOLで定義したデータ項目および定数の長さをバイト数として得ることができます。たとえば、COBOLプログラムから他言語で記述されたプログラムを呼び出すときのパラメタとして、領域のアドレスや領域の長さを渡すことができます。

終了条件なしのPERFORM文

NetCOBOLでは、PERFORM文に終了条件を設定しない書き方ができます。たとえば、繰返し処理の中で行われる処理の結果を判定し、繰返し処理を終了することができます。

14.2 ポインタ付けの使い方

ここでは、ポインタ付けの使い方について説明します。

概要

ポインタ付けは、ある特定のアドレスを持つ領域を参照する場合に使います。ポインタ付けを行うためには、次のデータ項目が必要です。

- 基底場所節で定義したデータ項目(a)
- 属性がポインタデータ項目のデータ項目(b)

ポインタ付けは、通常、ポインタ修飾子(->)によって行われます。(a)は、(b)によって次のようにポインタ付けされます。これをポインタ修飾といいます。

```
(b)->(a)
```

この場合、(a)の内容は、(b)に設定されたアドレスを持つ領域の内容となります。

プログラムの記述

ここでは、ポインタ付けを使うプログラムの記述について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.   プログラム名.
 DATA DIVISION.
  BASED-STORAGE SECTION.
  01  データ名１      [BASED ON  ポインタ１].
  77  データ名２   ～ [BASED ON  ポインタ２].
  WORKING-STORAGE SECTION.
```

```
 01  ポインタ１  POINTER.
 LINKAGE SECTION.
 01  ポインタ２  POINTER.
PROCEDURE DIVISION USING   ポインタ２.
    MOVE  [ポインタ１->] データ名１  ～.
    IF    [ポインタ２->] データ名２  ～.
END PROGRAM  プログラム名.
```

環境部(ENVIRONMENT DIVISION)

とくに必要な記述はありません。

データ部(DATA DIVISION)

基底場所節で、アドレスを指定して参照または更新を行うためのデータ名を定義します。また、ファイル節、作業場所節、基底場所節および連絡節でアドレスを格納するためのデータ名(属性をポインタデータ項目(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

繰返し処理の中で終了条件を判定したい場合、通常のPERFORM文ではプログラムの記述が複雑になります。このような場合、終了条件を設定しないPERFORM文を使用すると、プログラムの記述が簡単になります。

プログラムの記述

ここでは、終了条件なしのPERFORM文を使うプログラムの記述について、COBOLの各部ごとに説明します。

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.   プログラム名.
 PROCEDURE DIVISION.
     PERFORM WITH NO LIMIT
        ～
       IF   ～
         EXIT PERFORM
       END-IF
        ～
     END-PERFORM.
 END PROGRAM  プログラム名.
```

環境部(ENVIRONMENT DIVISION)

とくに必要な記述はありません。

データ部(DATA DIVISION)

とくに必要な記述はありません。

手続き部(PROCEDURE DIVISION)

終了条件なしのPERFORM文には、WITH NO LIMITを指定します。PERFORM文による繰返し処理の中では、終了条件を判定し、繰返し処理を脱出する文を記述します。繰返し処理の中に繰返し処理を脱出する文がない場合、この繰返し処理は無限に繰り返されます(無限ループ)。

プログラムの翻訳・リンク

とくに必要な翻訳オプションおよび追加ライブラリはありません。

プログラムの実行

とくに必要な環境設定はありません。

第15章　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。

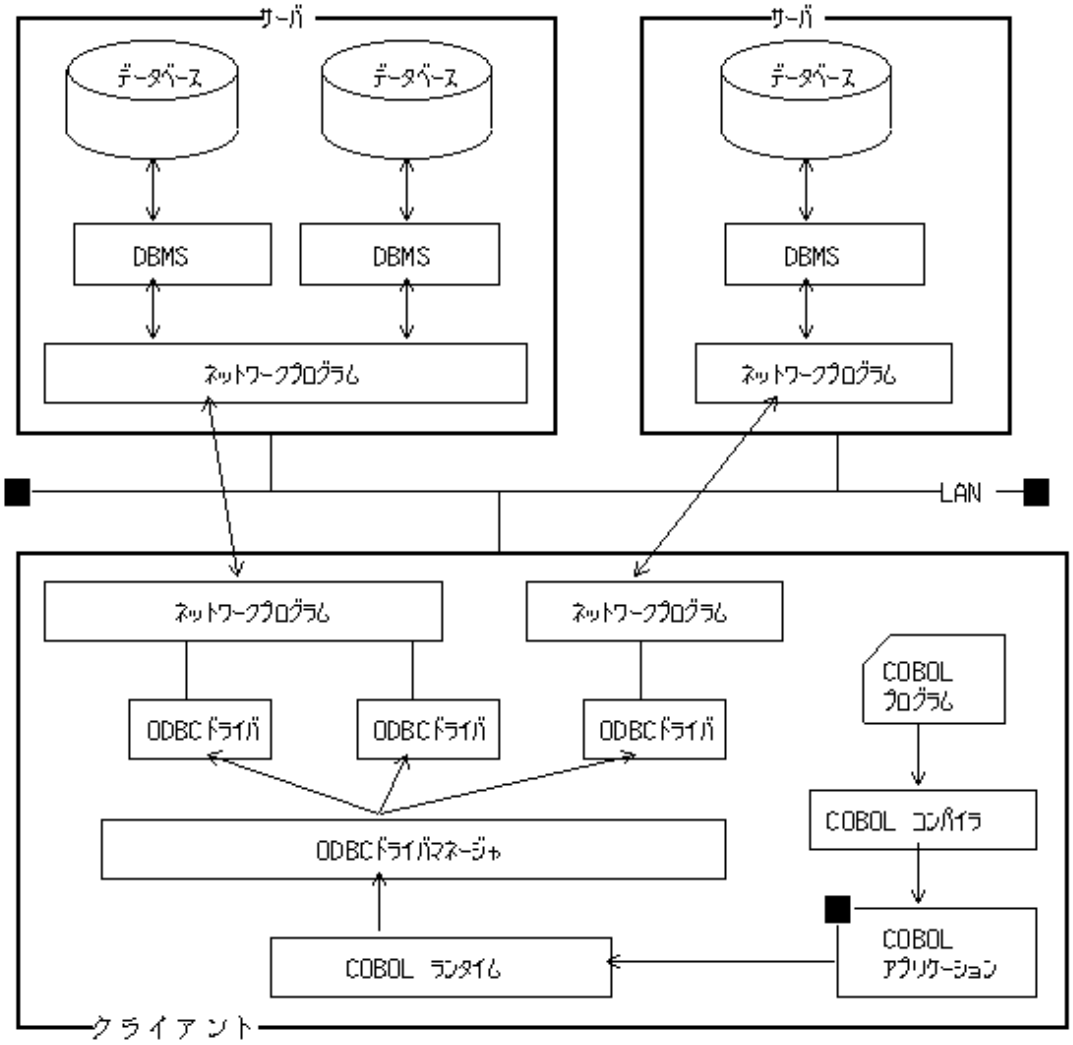
15.2.1 概要

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.                              ←特にSQL 固有の記述はありません。
           :
DATA DIVISION.
           :
 WORKING-STORAGE SECTION.
           :
     EXEC SQL BEGIN DECLARE SECTION END-EXEC.      ←宣言節内でSQLSTATEを宣言します。
 01  SQLSTATE    PIC X(5).                           また、必要なら、ホスト変数を定義します。
           :
     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。

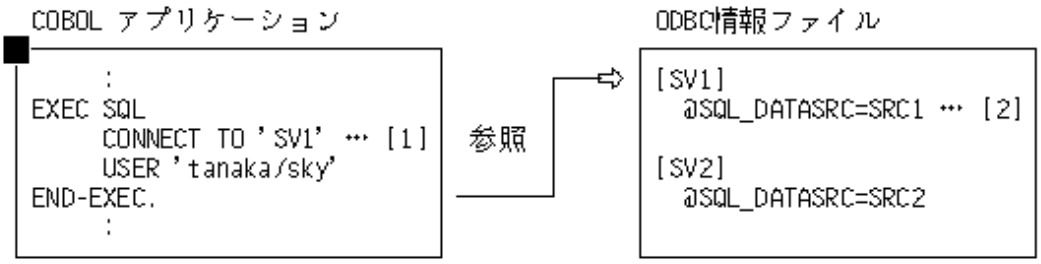
1. サーバの情報を登録します。
2. 以下のどちらかの方法で、サーバとコネクションを接続します。
   - サーバ名を指定して接続する
   - DEFAULTを指定して接続する

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ると、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。

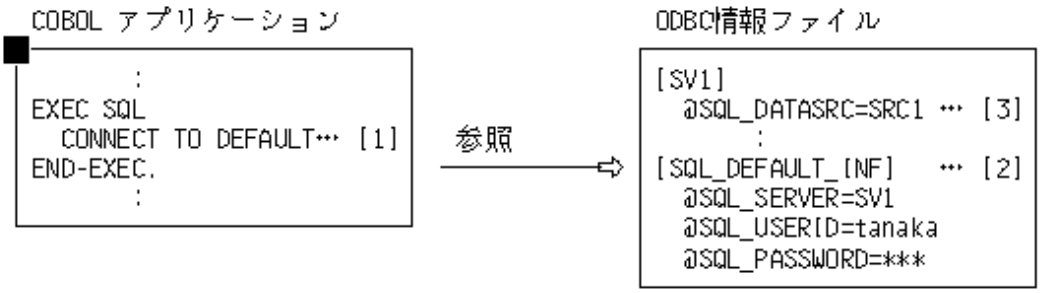
[2] CONNECT文の実行時に、ODBC情報ファイルの情報を参照してサーバとコネクションを接続します。

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ると、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、デフォルトコネクション情報を参照します。

[3] さらに、デフォルトコネクション情報のサーバ名の情報から、SV1の情報を参照してサーバとコネクションを接続します。

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ョンに接続します。

[2] SQLサーバ“SV1”に接続します。この接続の名前を“CNN1”とします。

[3] SQLサーバ“SV2”に接続します。この接続の名前を“CNN2”とします。

[4] SQLサーバ“SV3”に接続します。この接続の名前を“CNN3”とします。

[5] SQLサーバ“SV4”に接続します。この接続の名前を“CNN4”とします。最後に接続した“CNN4”が現在のコネクションになります。

[6] コネクション名“CNN4”を指定して、コネクションを切断します。

[7] コネクション名“CNN1”を指定し、コネクションを変更します。“CNN1”が現在のコネクションになります。

[8] コネクション“CNN1”中のデータの変更をすべて取り消します。

[9] 現在のコネクションを切断します。本例では現在のコネクションは“CNN1”です。そのため、“CNN1”が切断されます。

[10] DEFAULTを指定してコネクションを変更します。デフォルトコネクションが現在のコネクションになります。

[11] デフォルトコネクション中のデータの変更をすべて保存します。

[12] デフォルトコネクションを切断します。

[13] ALLを指定してDISCONNECT文を実行することにより、接続されているすべてのコネクションを切断します。

注意

CONNECT文により同時に接続できるコネクション数は、最大128 個です。
ただし、これは、ODBCドライバおよび関係する環境により、制限を受けることがあります。

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
- 複数コネクショ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”を指定しているので、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.
        :
```

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、サーバとのコネクションを接続します。

[4] OPEN文の実行により、[2]のカーソル宣言で指定した条件を満たす行からなる仮想的な表が作られます。ここで作られる仮想表は、STOCK表からQOH列の値が50以下である行を抽出し、さらにそこからGNO列、GOODS列およびQOH列を抜き出した形態をしています。一般に、このような仮想表の行の順番は不定です。

[5] FETCH文は、[4]で導出された仮想表の先頭から、データを1行ずつ取り出し、各列の値を対応するホスト変数の領域に設定します。特定の列について、値の昇順または降順で取り出したい場合は、[2]の問合せ式の後にORDER BY句を指定します。

[6] CLOSE文は、[4]で導出された仮想表を無効にします。再びOPEN文を実行しないかぎり、以降のSQL文でこの仮想表を参照することはできません。

[7] ROLLBACK文を実行し、トランザクションを終了します。

[8] DISCONNECT文を実行し、サーバとのコネクションを切断します。

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
111	…	2	61	…	60	63	…	FIFTH AVENUE N.Y USA
～	～	～	～	～	～	～	～	～

GNO	…	WHNO	COMPANYNO	…	OOH	CNO	…	ADDRESS
～	～	～	～	～	～	～	～	～
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                            [1]
         WHERE X1.GOODS = 'TELEVISION' AND
               X1.WHNO  = X2.WHNO                          ↓
     END-EXEC.                                             --
P-START.
     EXEC SQL CONNECT TO DEFAULT END-EXEC.
     EXEC SQL OPEN CUR4 END-EXEC.
P-LOOP.
     EXEC SQL
       FETCH CUR4 INTO  :製品名
     END-EXEC.
          :
     GO TO P-LOOP.
P-END.
     EXEC SQL CLOSE CUR4 END-EXEC.
     EXEC SQL ROLLBACK WORK END-EXEC.
     EXEC SQL DISCONNECT DEFAULT END-EXEC.
     STOP RUN.
```

[1] カーソル宣言文では、FROM句でSTOCK表に対して相関名(例では“X1”および“X2”)を指定し、あたかもそれが別々の表であるかのように扱います。

WHERE句の探索条件では、列名を相関名で修飾し、製品名TELEVISIONと同じ倉庫番号の倉庫に存在する製品に関する行を探索するように指定します。

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.
```

注意

複数の表を関連付けて得られる表に対して、データの更新を行うことはできません。

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.
```

注意

複数の表を関連付けて得られる表に対して、データの削除を行うことはできません。

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                                     …  [6]
         INTO :在庫表
     END-EXEC.
          :
     GO TO P-LOOP.
P-END.
```

```
     EXEC SQL CLOSE CUR8 END-EXEC.                          …  [7]
     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.                            …  [1]
     EXEC SQL CONNECT TO DEFAULT END-EXEC.
     EXEC SQL PREPARE STMIDT FROM :STMVAR END-EXEC.         …  [2]
     EXEC SQL EXECUTE STMIDT END-EXEC.                      …  [3]
          :
     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。

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する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。

注意

この機能はODBCを利用したリモートデータベースアクセス固有の機能です。分散開発など、Windowsプラットフォーム以外で動作するアプリケーションの開発時には、この機能を無効化する必要があります。翻訳オプションNOSQLGRPを指定して翻訳してください。
[参照]“A.3.46 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

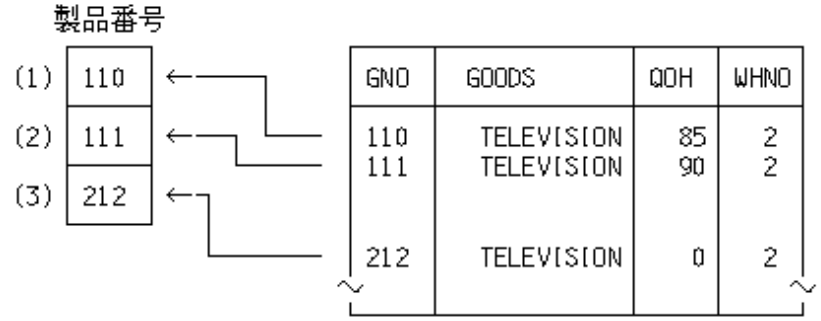
複数行指定ホスト変数を相手指定に使用すると、複数行指定ホスト変数の繰り返し回数と同じ件数のデータを参照することができます。

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                                   [2]
       WHERE GOODS = 'TELEVISION'                        ↓
     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。



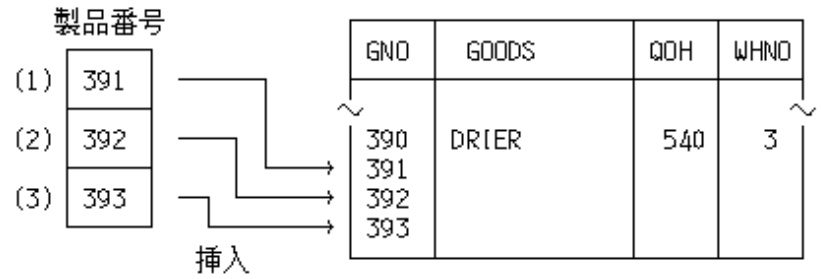
データの挿入

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.
      :
```

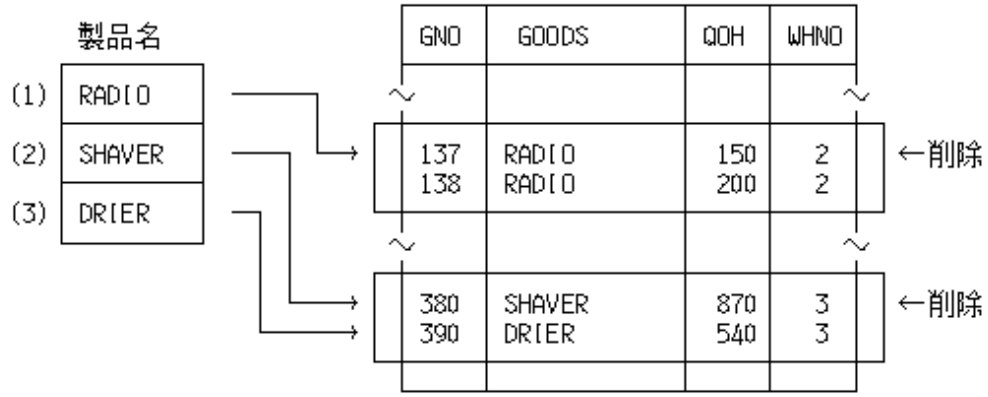
データの削除

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.
      :
```

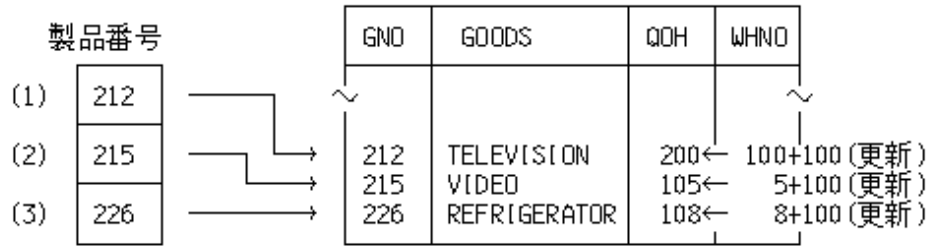
データの更新

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
```

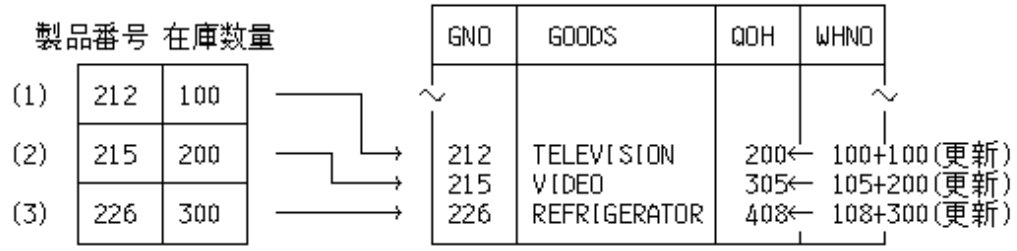
```
      WHERE GNO = :製品番号
    END-EXEC.
      :
```



また、複数行指定ホスト変数を問合わせ指定と設定句(SET)の値指定に同時に使用して、複数の条件に合致するデータを順番に更新することができます。

以下の例では、設定句および問合わせ指定に複数行指定ホスト変数を使用しています。これらの変数に格納された値は、それぞれの対応する列(GNO列およびQOH列)に対して、配列の先頭から順に使用されます。

```
    EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01  在庫表.
  02  製品番号  PIC S9(4) COMP-5 OCCURS 3 TIMES.
  02  在庫数量  PIC S9(9) COMP-5 OCCURS 3 TIMES.
01  SQLSTATE    PIC X(5).
    EXEC SQL END   DECLARE SECTION END-EXEC.
      :
PROCEDURE DIVISION.
      :
    MOVE 212 TO 製品番号(1).
    MOVE 215 TO 製品番号(2).
    MOVE 226 TO 製品番号(3).
    MOVE 100 TO 在庫数量(1).
    MOVE 200 TO 在庫数量(2).
    MOVE 300 TO 在庫数量(3).
    EXEC SQL
      UPDATE STOCK SET QOH = QOH + :在庫数量
      WHERE GNO = :製品番号
    END-EXEC.
      :
```



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文で使用する方法

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ド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                                       [3]
        FETCH CUR1 INTO :在庫表                          ↓
      END-EXEC.                                        ------
        :
```

[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).                        [1]
   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.                  … [2]
       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 NEXT CUR1 INTO :在庫表 END-EXEC.     … [5]
       EXEC SQL FETCH PRIOR CUR1 INTO :在庫表 END-EXEC.    … [6]
       EXEC SQL FETCH PRIOR CUR1 INTO :在庫表 END-EXEC.    … [7]
       EXEC SQL CLOSE CUR1 END-EXEC.                       … [8]
       EXEC SQL ROLLBACK WORK END-EXEC.                    … [9]
       EXEC SQL DISCONNECT DEFAULT END-EXEC.               … [10]
       STOP RUN.
```

[1] 表指定ホスト変数"在庫表"を定義します。

[2] カーソル"CUR1"をオープンします。

[3] 在庫表の1件目のデータを"在庫表"に取り出します。

[4] 在庫表の2件目のデータを"在庫表"に取り出します。

[5] 在庫表の3件目のデータを"在庫表"に取り出します。

[6] 在庫表の2件目のデータを"在庫表"に取り出します。

[7] 在庫表の1件目のデータを"在庫表"に取り出します。

[8] カーソル"CUR1"をクローズします。

[9] ROLLBACK文を実行し、トランザクションを終了します。

[10] DISCONNECT文を実行し、サーバとのコネクションを切断します。

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
	:		

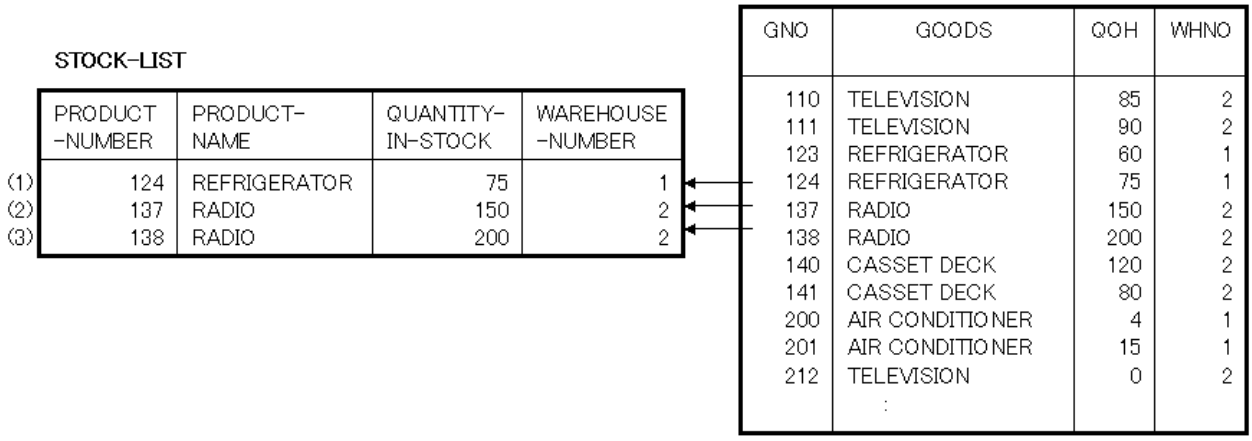
[5] 在庫表の7～9件目のデータを"在庫表"に取り出します。

STOCK-LIST

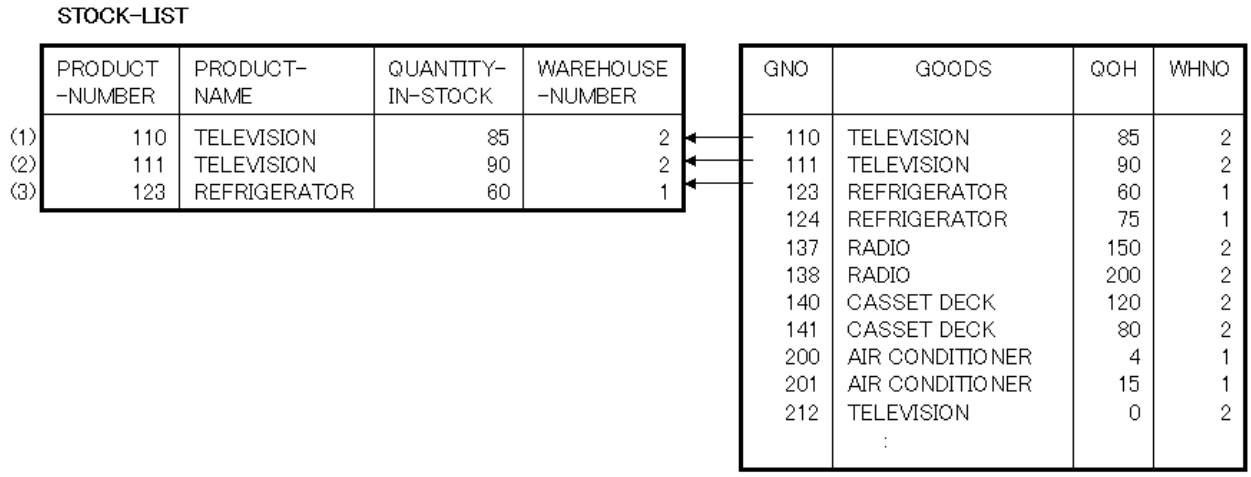
	PRODUCT-NUMBER	PRODUCT-NAME	QUANTITY-IN-STOCK	WAREHOUSE-NUMBER
(1)	140	CASSET DECK	120	2
(2)	141	CASSET DECK	80	2
(3)	200	AIR CONDITIONER	4	1

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



[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。



[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、サーバとのコネクションを切断します。

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).                             [1]
   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.                      … [2]
```

```
        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。

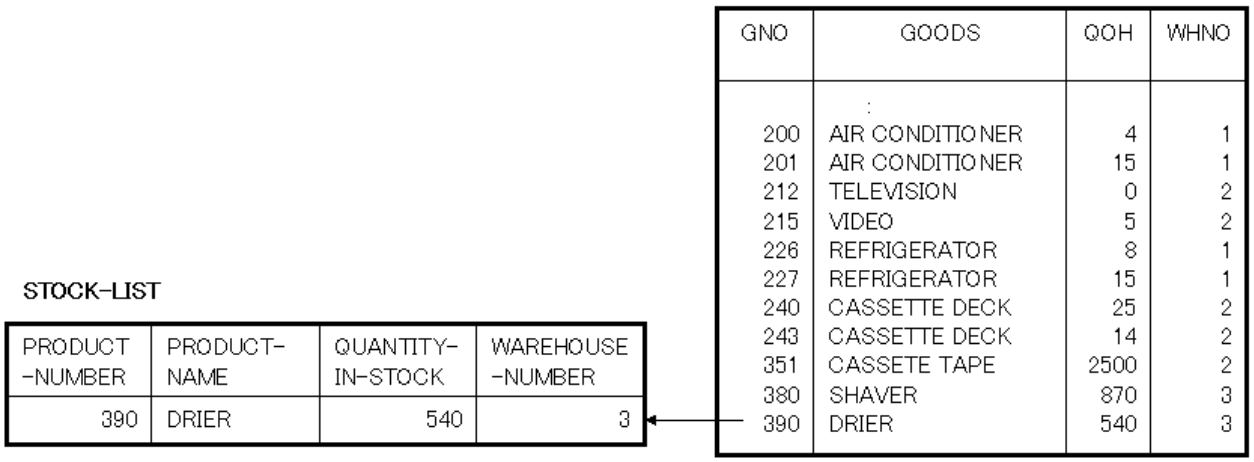
FETCH LASTプログラム例

```
     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).                                [1]
   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.                          … [2]
       EXEC SQL FETCH LAST CUR1 INTO :在庫表 END-EXEC.       … [3]
       EXEC SQL FETCH PRIOR CUR1 INTO :在庫表 END-EXEC.      … [4]
       EXEC SQL 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.
```

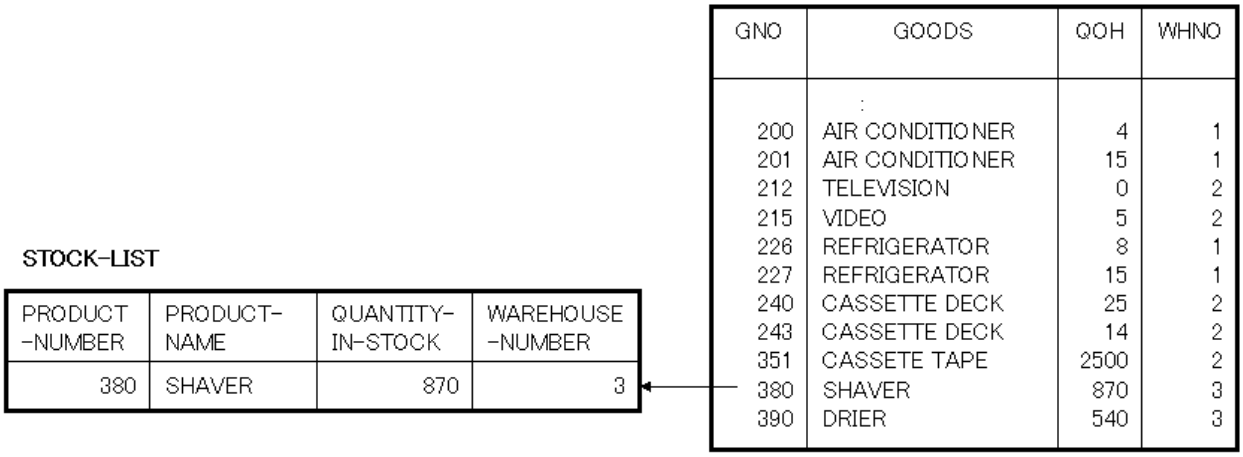
[1] 表指定ホスト変数"在庫表"を定義します。

[2] カーソル"CUR1"をオープンします。

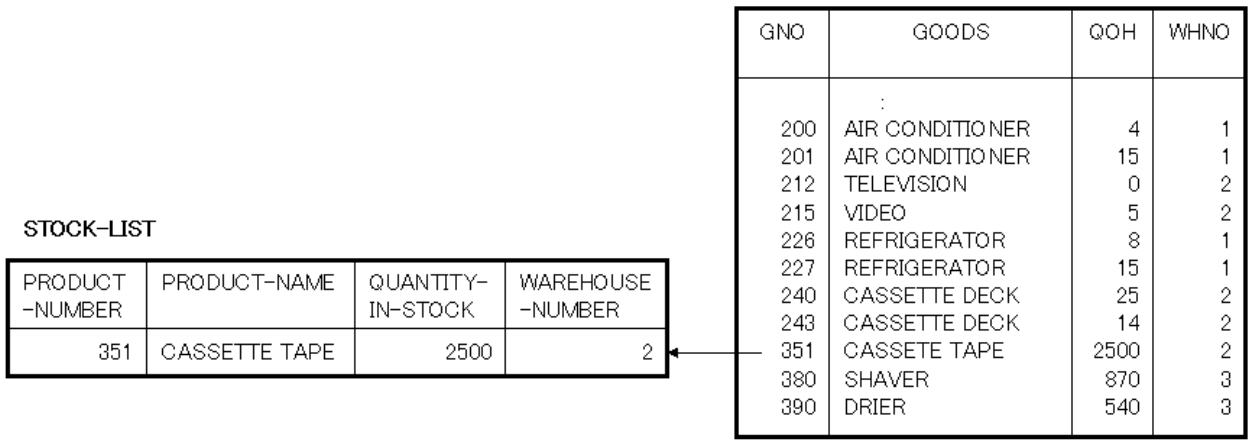
[3] 在庫表のGNOが390のデータを"在庫表"に取り出します。



[4] 在庫表のGNOが380のデータを"在庫表"に取り出します。



[5] 在庫表のGNOが351ののデータを"在庫表"に取り出します。



[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。

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。

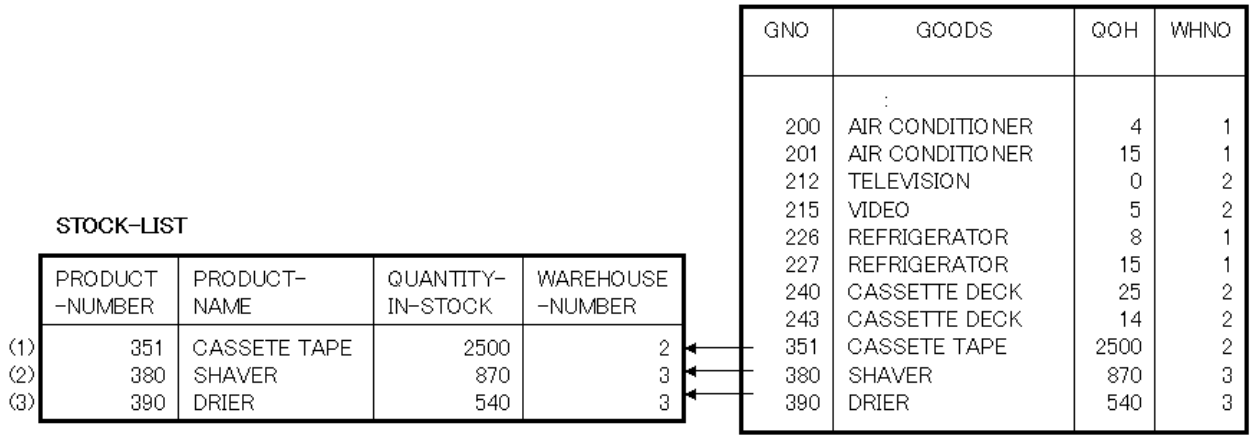
複数行指定ホスト変数を使用したFETCH LASTプログラム例

```
     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.
```

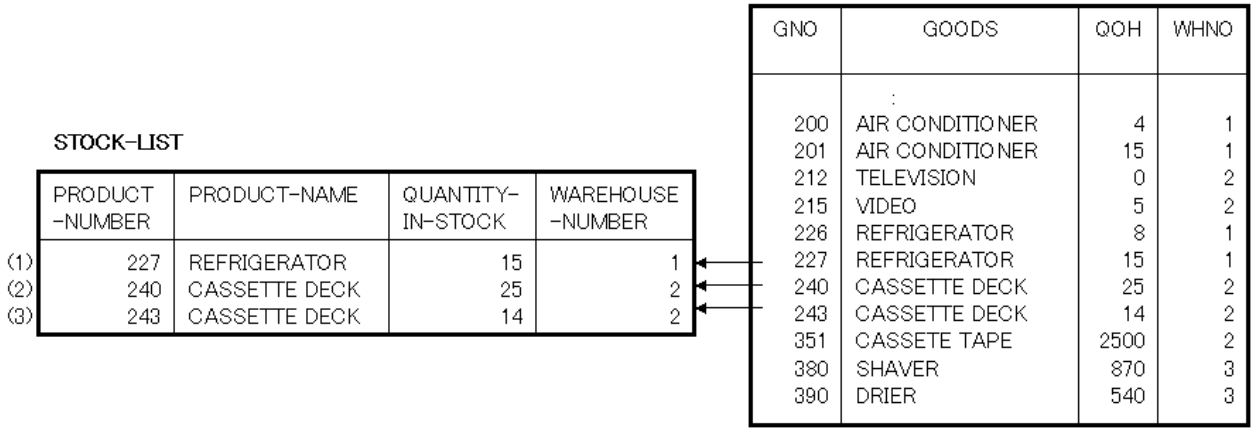
[1] 表指定ホスト変数"在庫表"を定義します。

[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。



[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。

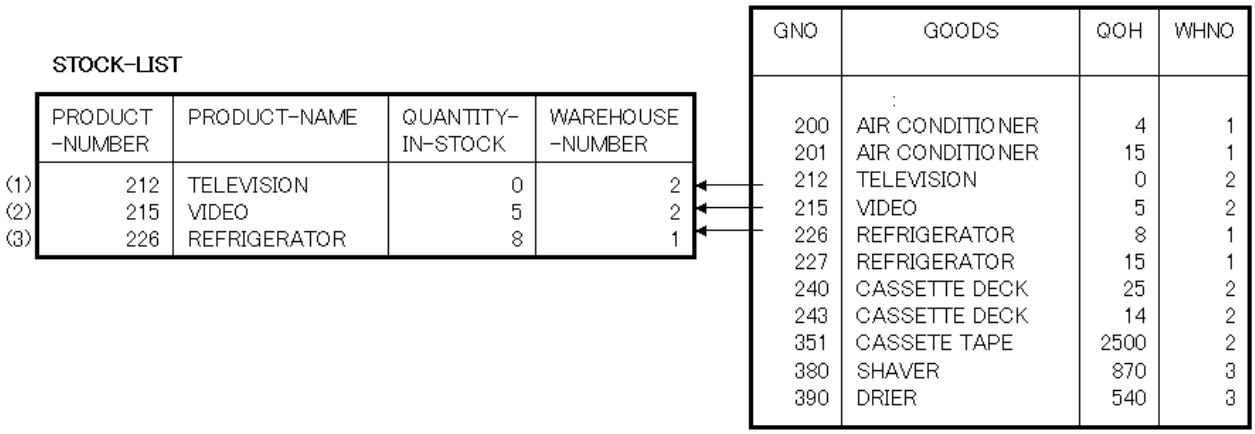


STOCK-LIST

	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	CASSETE 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	CASSETE 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、ストアドプロシージャ内の処理が軽い場合、パフォーマンスを発揮できない場合があります。その場合はストア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ストアドプロシージャを呼び出します。呼び出し後、ストアドプロシージャからの出力パラメ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16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).                                    [1]
  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.                      [2]
  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                     [3]
      ELSE
        COMPUTE 在庫数量 =  在庫数量 -  入出数量.
      EXEC SQL
        UPDATE STOCK SET QOH = :在庫数量 WHERE GNO = :製品番号
      END-EXEC.                                                       ↓
  END METHOD ZAIKO-KANRI.                                         ------
  END OBJECT.
END CLASS DBACCESS-CLASS.
```

[1] メソッドデータを定義します。

[2] 在庫管理メソッドのインタフェースを定義します。

[3] 在庫管理メソッドを定義します。このメソッドでは、該当する製品番号の在庫数量をSTOCK表(在庫表)より取り出し、入庫または出庫を切り分けて在庫数量を計算し、再び表に格納します。

データベースアクセスクラスのメソッドを呼び出すプログラムの例

```
          :
CONFIGURATION SECTION.
REPOSITORY.
    CLASS DBACCESS-CLASS.
DATA DIVISION.
WORKING-STORAGE SECTION.
    EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01  SQLSTATE      PIC   X(5).
    EXEC SQL END   DECLARE SECTION END-EXEC.
01  オブジェクト.
```

```
  02  DBACCESS-OBJECT  OBJECT REFERENCE DBACCESS-CLASS.
01  製品番号       PIC S9(4) COMP-5.
01  入出区別       PIC S9(4) COMP-5.
01  入出数量       PIC S9(9) COMP-5.
01  終了要求       PIC X(1).
PROCEDURE DIVISION.
    EXEC SQL CONNECT TO DEFAULT END-EXEC.
    INVOKE  DBACCESS-CLASS "NEW" RETURNING DBACCESS-OBJECT.          … [1]
    PERFORM TEST AFTER UNTIL  終了要求 = "Y"                          ------
      DISPLAY  "製品番号、入出区別(1or2)、入出数量を入力してください"   ↑
      ACCEPT  製品番号
      ACCEPT  入出区別
      ACCEPT  入出数量                                                [2]
      INVOKE  DBACCESS-OBJECT "ZAIKO-KANRI"
              USING   製品番号  入出区別  入出数量
      DISPLAY  "在庫管理を終了しますか？(Y/N)"
      ACCEPT  終了要求                                                ↓
    END-PERFORM.                                                      ------
    EXEC SQL COMMIT WORK END-EXEC.
    EXEC SQL DISCONNECT DEFAULT END-EXEC.
    STOP RUN.
```

[1] “NEW”メソッドによりオブジェクトインスタンスを生成します。

[2] 製品番号、入出区別、入出数量の入力を要求し、在庫管理メソッドを呼び出します。終了要求に“Y”が入力されるまで処理を繰り返し行います。

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、入庫または出庫を切り分けて在庫数量を再計算します。次に再計算後の在庫数量を表示し、変更結果の反映有無を入力させます。データ操作は、終了要求に"Y"が入力されるまで繰り返し行います。

データベースアクセスクラスのメソッド(在庫管理メソッド)を呼び出すプログラムの例

```
            :
  CONFIGURATION SECTION.
  REPOSITORY.
      CLASS DBACCESS-CLASS.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  OBJECT-COUNT   PIC  S9(1).
  01  オブジェクト.
    02  DBACCESS-OBJECT OCCURS 3 OBJECT REFERENCE DBACCESS-CLASS.
  PROCEDURE DIVISION.
      PERFORM WITH TEST AFTER VARYING OBJECT-COUNT FROM 1 BY 1          ------
            UNTIL OBJECT-COUNT = 3                                       ↑
        INVOKE  DBACCESS-CLASS "NEW"                                    [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.3.1 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

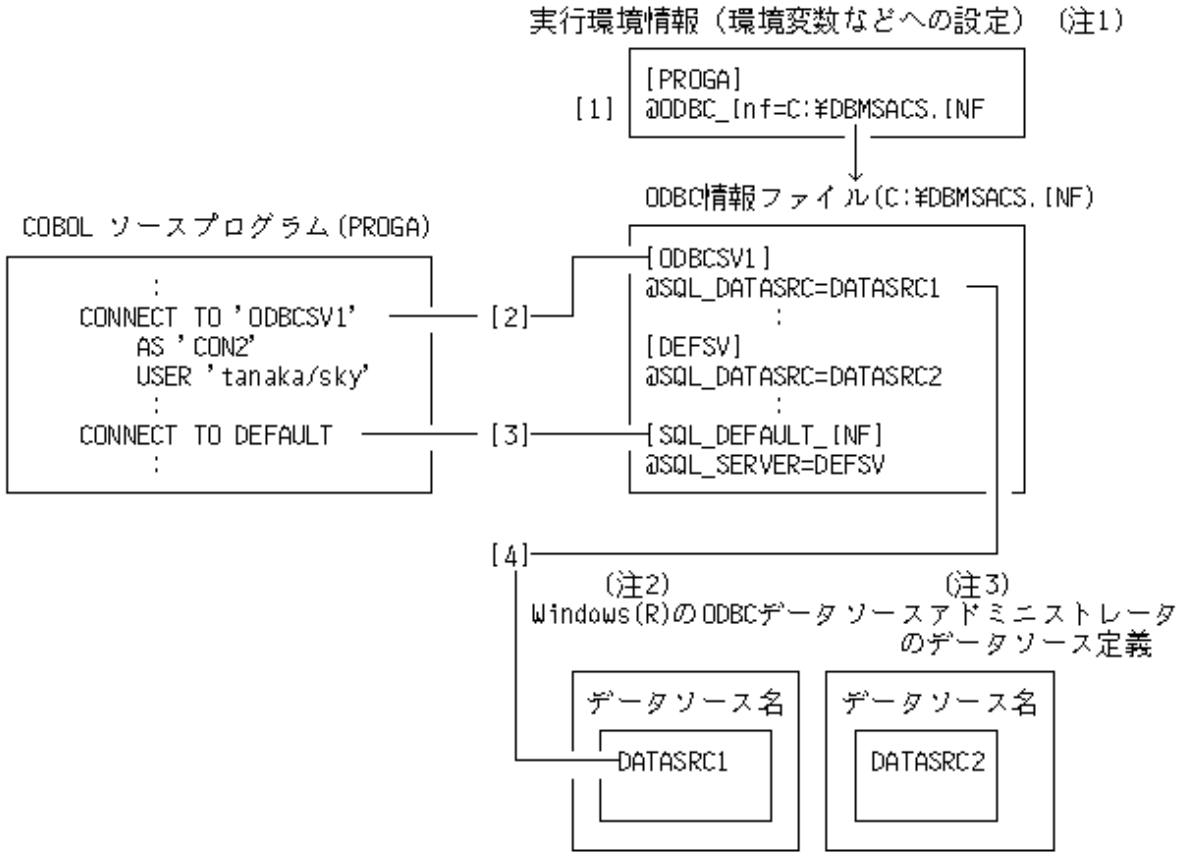
ここでは、プログラムを実行するために必要な実行環境の構築およびODBC情報設定ツールの使い方について説明します。

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、“5.3.2 実行環境情報の設定方法”を参照してください。

注2:データソースとは、ODBCドライバ、ネットワークシステムおよびデータベースなどの環境全般を総称したものです。

[1] ODBC情報ファイル名(C:¥DBMSACS.INF)を指定します。

[2] CONNECT文にサーバ名を記述した場合、ODBC情報ファイルにサーバ名を指定します。

[3] CONNECT文にDEFAULTを記述した場合、ODBC情報ファイルにデフォルトコネクション情報の定義を示す固定文字列を指定します。

[4] ODBC情報ファイルの各サーバのデータソース名の定義には、WindowsシステムのODBCデータソースアドミニストレータで定義したデータソース名を指定します。

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、以下の情報を環境変数情報に設定してください。[参照]“5.3.2 実行環境情報の設定方法”

@ODBC_Inf (ODBC情報ファイルの指定)

```
@ODBC_Inf=C:¥DBMSACS.INF
```

COBOLランタイムシステムがODBCを使用するために参照するファイル名を指定します。[参照]“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	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)
		FILE_DS	
@SQL_DATASRC	データソース名		WindowsシステムのODBCデータソースアドミニストレータで指定(追加)したデータソース名を指定します。
@SQL_USERID	ユーザID		データソースを操作するためのユーザIDを指定します。
@SQL_PASSWORD	パスワード		データソースを操作するためのパスワードを指定します。 このパスワードは、ODBC情報設定ツールを使用して、暗号化して設定してください。
@SQL_ACCESS_MODE	アクセスモード	READ_ONLY	データソースに対するアクセスモードを指定します。デフォルトは READ_WRITE です。 文字列READ_ONLYを指定した場合は読込み専用となり、文字列 READ_WRITE を指定した場合は読込みおよび書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。
		READ_WRITE	
@SQL_COMMIT_MODE	COMMITモード	MANUAL	データソースに対するCOMMITモード(トランザクションの動作)を指定します。デフォルトはMANUALです。 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。
		AUTO	
@SQL_CONCURRENCY	カーソルの同時実行	READ_ONLY	カーソルの同時実行を指定します。同時実行とは、複数のクライアント(複数のコネクションでも同様)が、同一データを同時に使用する機能を指します。

情報名	設定内容		備考
		LOCK	@SQL_CONCURRENCY が指定されない場合、デフォルトは READ_ONLYです。(注2)
		ROWVER	
		VALUES	
@SQL_CURSOR_TYPE	カーソルの種類	FORWARD_ONLY	カーソルの種類を指定します。 @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。

- 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、“15.2.13.3 各ODBCドライバ固有の留意事項”を参照してください。

- 注3

  指定値に対する動作は、“表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、“15.2.13.3 各ODBCドライバ固有の留意事項”を参照してください。

- 注4

  ODBCカーソルライブラリを使用する場合は、以下の注意事項があります。

  ― UPDATE文(位置付け)またはDELETE文(位置付け)を使用する場合は、カーソル宣言で値が一意な列を必ず1つ以上選択してください。

  ODBCカーソルライブラリは、UPDATE文(位置付け)およびDELETE文(位置付け)をそれぞれUPDATE文(探索)およびDELETE文(探索)にシミュレートして実行します。そのため、一意な値を持つ列が選択されていない場合、複数行に処理が影響する場合があります。

  ― ODBCカーソルライブラリを使用しない場合と比較して、実行性能が劣化する場合があります。

  ― カーソルの同時実行(@SQL_CONCURRENCY)に“VALUES”を、カーソルの種類(@SQL_CURSOR_TYPE)に“STATIC”を指定しなければなりません。

表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。

注意

ODBCドライバによって、指定に対する動作がエラーとなる場合があります。このような場合、READ_ONLYを指定してください。

表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例(セクション名)を指定します。

情報名	設定内容	備考
@SQL_SERVER	サーバ名	デフォルトコネクションを確立する対象のサーバ名を指定します。このサーバ名をもとにしてサーバごとの定義情報を検索し、サーバごとのデータソース名に対してコネクションを確立します。したがって、サーバごとの定義情報を記述しておく必要があります。
@SQL_USERID	ユーザID	デフォルトコネクションサーバに対応するデータソースを操作するためのユーザIDを指定します。
@SQL_PASSWORD	パスワード	デフォルトコネクションサーバに対応するデータソースを操作するためのパスワードを指定します。このパスワードは、ODBC情報設定ツールを使用して、暗号化して設定してください。

コネクション有効範囲に定義される内容

コネクションの有効範囲に関する情報の定義内容を“表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	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。 コネクションの有効範囲とは、接続したコネクションの利用可能範囲です。(注)
		THREAD	
		OBJECT_INSTANCE	

注：指定値に対する動作は、“表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、“18.5.2 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	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。 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。 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 ・ 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。 ・ 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。 ・ 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。
THREAD	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。 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。 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)
OBJECT_INSTANCE	接続したコネクションは、オブジェクトインスタンス内で利用可能です。

指定値	動作説明
	通常、オブジェクト指向機能を利用したシングルスレッドプログラムをマルチスレッドプログラムに移行する場合に指定します。オブジェクトインスタンス内で複数のコネクションを接続した場合、最後に実行されたCONNECT 文またはSET CONNECTION文で指定されたコネクションが、オブジェクトインスタンスの現コネクションになります。(注2)

注1:クラス定義(オブジェクト指向プログラミング機能)に記述した埋込みSQL文は動作しません。

注2:プログラム定義に記述した埋込みSQL文は動作しません。

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。

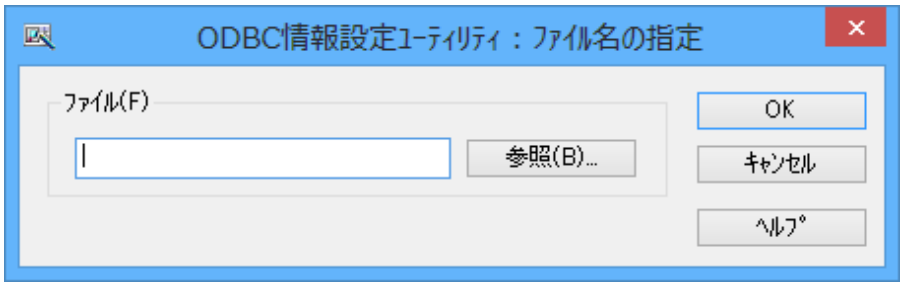
ODBC情報設定ツールには、次のような機能があります。

- 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。

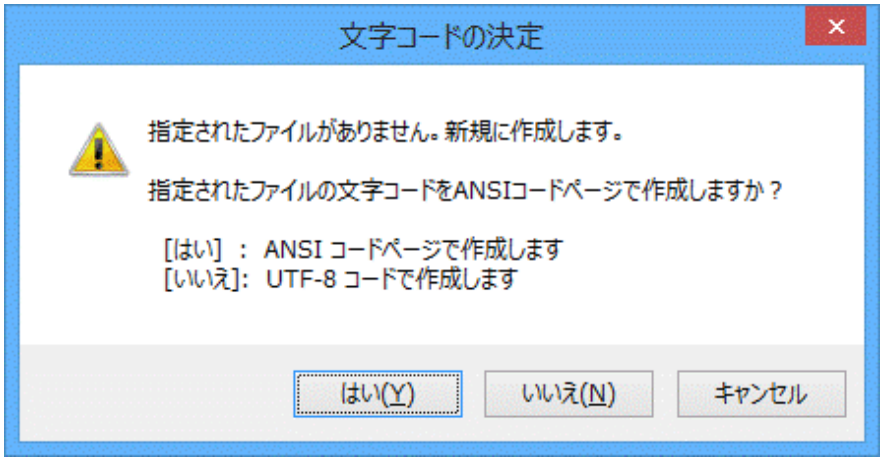
1. ODBC情報設定ツールの起動

   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

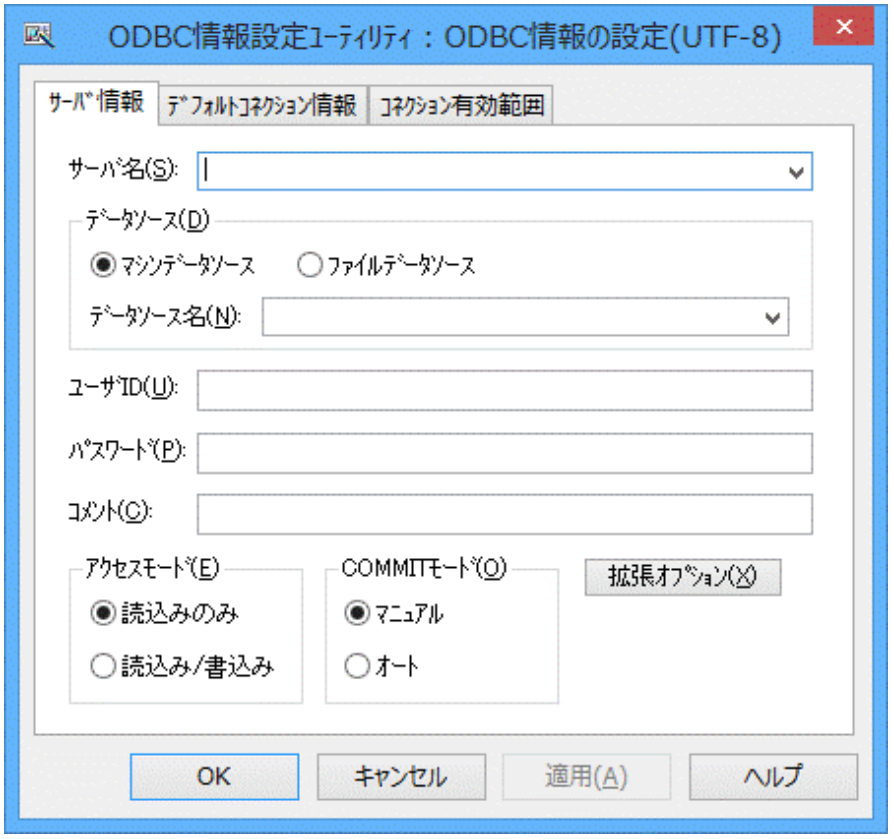
   情報を設定するODBC情報ファイルを指定します。
   指定できるファイルの文字コードは、ANSIコードページ(シフトJIS)とBOM付きUTF-8です。ファイルを新規に作成する際は、以下のダイアログにより、文字コードを選択してください。「はい」を指定するとシフトJISでファイルを作成します。「いいえ」を指定するとBOM付きUTF-8でファイルを作成します。



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オープンします。
BOM付きUTF-8のファイルをオープンした時は、ODBC情報設定ツールのタイトルに（UTF-8）の文字が表示されます。

3. サーバ情報の設定

CONNECT文またはデフォルトコネクション情報で、指定されたサーバ名に対する情報を設定します。



サーバ名を指定して、サーバ名毎の情報を設定します。サーバ名に指定した値は、[サーバ名]のセクションとして、ODBC情報ファイルに追加されます。
サーバ情報タグでは、サーバ名に指定した[サーバ名]セクション毎に以下の情報を設定します。

ODBC情報設定ユーティリティのサーバ情報		情報名
データソース	マシンデータソース/ファイルデータソースオプションボタン	@SQL_DATASRC_KIND
	データソース名	@SQL_DATASRC
ユーザID		@SQL_USERID
パスワード		@SQL_PASSWORD
アクセスモード		@SQL_ACCESS_MODE
COMMITモード		@SQL_COMMIT_MODE

[拡張オプション]ボタンで拡張オプションのダイアログが表示されます。

拡張オプションダイアログでは、サーバ名に指定した[サーバ名]セクション毎に以下の情報を設定します。

ODBC情報設定ユーティリティの拡張オプションダイアログ	情報名
カーソル同時実行	@SQL_CONCURRENCY
カーソルの種類	@SQL_CURSOR_TYPE
ODBCカーソルライブラリ	@SQL_ODBC_CURSORS
クエリータイムアウト時間	@SQL_QUERY_TIMEOUT

詳細は、“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。

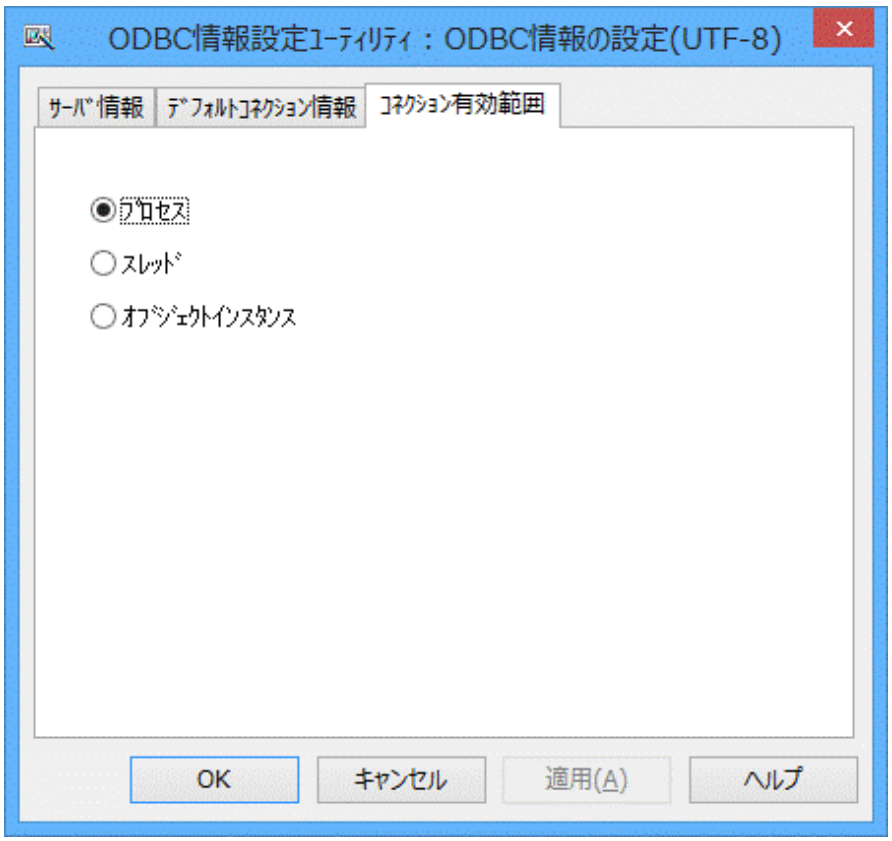
デフォルトコネクション情報タグでは、[SQL_DEFAULT_INFO]セクションに以下の情報を設定します。

ODBC情報設定ユーティリティのデフォルトコネクション情報	情報名
サーバ名	@SQL_SERVER
ユーザID	@SQL_USERID
パスワード	@SQL_PASSWORD

詳細は、“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。システムデータソースについては、ODBCデータソースアドミニストレータのオンラインヘルプを参照してください。
[参照]“17.1.3 サービス配下で動作するプログラム”

ODBCドライバに関係する環境の整備

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と安全です。

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

【G】	GET
	GLOBAL
	GO
	GOTO
	GRANT
	GROUP

【H】	HAVING
	HOUR

【I】	IDENTITY
	IGNORE
	IMMEDIATE
	IN
	INCLUDE
	INDEX
	INDICATOR
	INITIALLY
	INNER
	INPUT

	INSENSITIVE
	INSERT
	INTEGER
	INTERSECT
	INTERVAL
	INTO
	IS
	ISOLATION

【J】	JOIN

【K】	KEY

【L】	LANGUAGE
	LAST
	LEFT
	LENGTH
	LEVEL
	LIKE
	LIST
	LOCAL
	LOWER

【M】	MATCH
	MAX
	MIN
	MINUTE
	MODULE
	MONTH
	MUMPS

【N】	NAME
	NAMES
	NATIONAL
	NCHAR
	NEXT
	NONE
	NOT
	NULL
	NULLABLE
	NULLIF

	NUMERIC

【O】	OCTET_LENGTH
	OF
	OFF
	ON
	ONLY
	OPEN
	OPTION
	OR
	ORDER
	OUTER
	OUTPUT
	OVERLAPS

【P】	PARTIAL
	PASCAL
	PLI
	POSITION
	PRECISION
	PREPARE
	PRESERVE
	PREVIOUS
	PRIMARY
	PRIOR
	PRIVILEGES
	PROCEDURE
	PUBLIC

【R】	RELATIVE
	RESTRICT
	REVOKE
	RIGHT
	ROLLBACK
	ROWS

【S】	SCALE
	SCHEMA
	SCROLL
	SECOND
	SECTION

	SELECT
	SEQUENCE
	SET
	SIZE
	SMALLINT
	SOME
	SQL
	SQLCA
	SQLCODE
	SQLERARY
	SQLERRD
	SQLERROR
	SQLSTATE
	SQLWARNING
	START
	SUBSTRING
	SUM
	SYSTEM

【T】	TABLE
	TEMPORARY
	THEN
	TIME
	TIMESTAMP
	TIMEZONE_HOUR
	TIMEZONE_MINUTE
	TO
	TRANSACTION
	TRANSLATE
	TRANSLATION
	TRUE
	TYPE

【U】	UNION
	UNIQUE
	UNKNOWN
	UPDATE
	UPPER
	USAGE
	USER
	USING

【V】	VALUE
	VALUES
	VARCHAR
	VARIABLES
	VARYING
	VIEW

【W】	WHEN
	WHENEVER
	WHERE
	WITH
	WORK

【Y】	YEAR

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

		1行ずつ操作		複数行を同時に操作	
埋込みSQL文		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)	×	×	×	×

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
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。

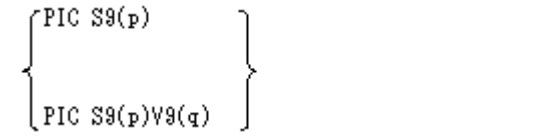
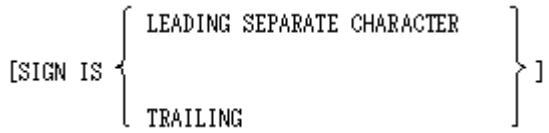
ODBCドライバがODBCのSQLデータ型をどのように扱うかについては、使用するODBCドライバのヘルプおよびマニュアルを参照してください。

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
	SQL_BIGINT (BIGINT)(注1)	PIC S9(18) BINARY または PIC S9(18) COMP-5
10進	SQL_DECIMAL (DECIMAL)	PIC S9(p) PACKED-DECIMAL または PIC S9(p)V9(q) PACKED-DECIMAL 1≦p≦31(注2) 1≦q p+q≦31(注2) ただし、最大精度はドライバの仕様に依存します。

ODBC SQLデータ型		COBOL での表現
	SQL_NUMERIC (NUMERIC)	  1≦p≦31(注2) 1≦q p＋q≦31(注2) ただし、最大精度はドライバの仕様に依存します。
内部浮動小数点	SQL_REAL (REAL)	COMP-1
	SQL_DOUBLE (FLOAT)	COMP-2

注1:このデータ型を使用する場合、31桁拡張演算モードで翻訳してください。

注2:18桁互換演算モードで翻訳する場合は、18桁まで使用できます。31桁まで使用する場合は、31桁拡張演算モードで翻訳してください。

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) 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) 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
	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. (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	プログラマの処理
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	プログラマの処理
99999	-999999999	同じコネクション名で接続しました。	同じコネクション名を持つ複数のコネクションの指定は許されません。それぞれコネクション名が一意になるように修正してください。
9999A	-999999990	コネクションの最大設定数を超えました。	COBOL システムで許されるコネクションの最大数を超えています。コネクション数を減らす対処を行ってください。 ただし、コネクションの最大数は、ODBCの環境により異なることがあります。ODBCの環境のマニュアルを参照して対処してください。
9999B	-999999800	指定したコネクションは存在しません。	コネクションがアクティブになっていないため、SQL 文が実行できない状態にあります。 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。

SQLSTATE	SQLCODE	SQLMSG	プログラマの処理
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ドライバに関係するソフトウェア/ハードウェア、およびデータベース管理システムの関連マニュアルも参照してください。
- COMMIT文またはROLLBACK文を使用してトランザクションを終了しなければ、動作を保証できない場合があります。また、DISCONNECT文でコネクションを切断する前には、必ずCOMMIT文またはROLLBACK文を記述してトランザクションを終了してください。
- SQL記述子域は使用できません。したがって、以下の埋込みSQL文または指定は記述できません。
  - 記述子名を指定したINTO句およびUSING句
  - ALLOCATE DESCRIPTOR 文、DEALLOCATE DESCRIPTOR 文、GET DESCRIPTOR 文、SET DESCRIPTOR 文、DESCRIBE文などの記述子域に関係する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)

ー 動的DELETE文(探索)

ー 動的DELETE文(位置付け)

- 動的カーソル宣言で使用する文識別子と同名の文識別子をEXECUTE文で指定することはできません。または、EXECUTE文に指定する文識別子と同名の文識別子を動的カーソル宣言で使用することはできません。
- 埋込みSQL文の文法の詳細は、各データベースおよびデータベース関連製品の仕様に従います。

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ドライバに関係するソフトウェア/ハードウェア、およびデータベース管理システムのマニュアルも参照してください。
- 埋込みSQL文の記述、ODBC情報ファイルの指定、データソースの設定により、ODBC環境のオプション値が変更されることがあります。この場合、埋込みSQL文を実行する際に各ODBCドライバにより通知メッセージが出力されます。埋込み例外宣言を使用する場合は、エラーが発生したものとして処理されますので、必要に応じてプログラムを変更してください。
- データ操作文以外の埋込みSQL文を動的に実行する場合、各ODBCドライバにより以下の現象が発生することがあります。データ操作文以外の埋込みSQL文を動的に実行しないようにしてください。

```
現象：
  SQLSTATE：02000
  SQLCODE  ：+100
  SQLMSG   ：データがありません。
```

- 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。

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

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。

Oracle(R) ODBCドライバ使用時の留意事項

CALL文で、以下を呼び出すことはできません。

- ファンクション
- パッケージ化されたストアドプロシージャ
- パッケージ化されたファンクション

Microsoft(R) SQL Server Native Client 10.0(TM) / Microsoft(R) SQL Native Client(TM)ODBCドライバ使用時の留意事項

- CONNECT文の実行時に、以下の現象(通知メッセージの出力)が発生しますが、正常に接続されています。
  埋込み例外宣言がCONNECT文より前に記述されている場合、エラーが発生したものとして処理されますので、埋込み例外宣言はCONNECT文より後ろに記述してください。

```
現象
  SQLSTATE : 01000
  SQLCODE  : +5701
  SQLMSG   : データベースのコンテキストを ' データベース名' に変更しました。
```

- カーソル系データ操作文のDELETE文(位置付け)、UPDATE文(位置付け)は特定の条件下でだけ使用可能です。[参照]"15.2.13.3 各ODBCドライバ固有の留意事項"の"特定の条件"
  なお、@SQL_CONCURRENCYを指定する場合の留意事項は、以下を参照してください。また、ODBC情報ファイルの作成、データソース(ODBCドライバ、データベース、データベース関連製品)のマニュアルを参照してください。[参照]"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。

  ー 表の作成時に任意の列に対して、PRIMARY KEY制約またはUNIQUE制約を付加します。

  ー 既存の表の任意の列に対して、一意なインデックスを作成します。(例:CREATE UNIQUE INDEX...)

  注意

  一意なインデックスの作成時に指定した任意の列には、重複値は許されません。つまり、任意の列の値は、それぞれ一意でなければなりません。

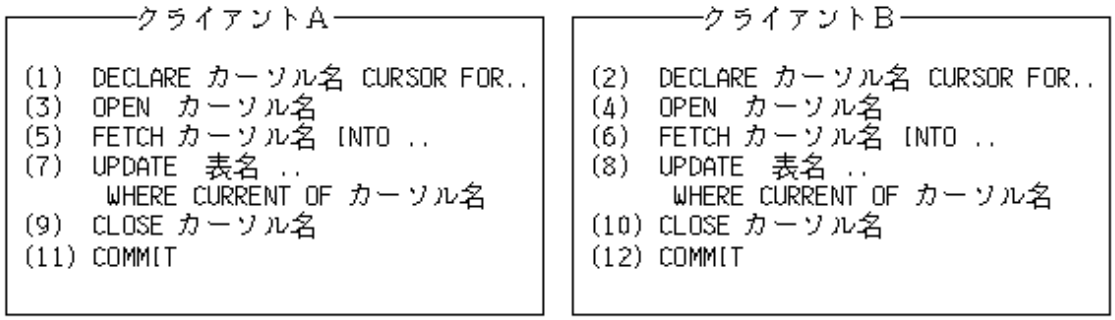
  既存の表の任意の列に対して、一意なインデックスを作成する場合、任意の列に重複値があるとエラーメッセージが返され、処理が失敗します。

- 一意なインデックスを作成しないで、@SQL_CONCURRENCYを指定した場合、カーソル系データ操作文のDELETE文(位置付け)、UPDATE文(位置付け)の実行時に以下の現象が発生し、処理が失敗する場合があります。

```
現象 :
  SQLSTATE : S1009
  SQLCODE  : +16929
  SQLMSG   : Cursor is read only.
```

- @SQL_CONCURRENCYを指定した場合の動作例を、以下に説明します。
  2つのクライアントA、Bで、最初にクライアントA、次にクライアントBの順番で同一サーバの同一表、同一行に対して、カーソル系

データ操作文のUPDATE文(位置付け)を行います。
このときにおけるロック、ロック解除のタイミングおよびクライアントの実行状態について説明します。



【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文を実行するようにしてください。
  また、これらのSQL文を実行可能にするためには、NetCOBOLおよびデータソース(ODBCドライバ、データベース、データベース関連製品)の環境が、特定の条件に合致している必要があります。[参照]“15.2.13.3 各ODBCドライバ固有の留意事項”の“特定の条件”

- 次に示すカーソル宣言で定義されたカーソルをオープンしようとすると、以下の現象が発生し、カーソルオープンが失敗します。

```
カーソル宣言
  DECLARE カーソル名 CURSOR FOR SELECT ... FOR UPDATE

現象
  SQLSTATE：37000
  SQLCODE  ：+1003
  SQLMSG   ：FOR UPDATE句はDECLARE CURSORでだけ許されます。
```

  したがって、上記のような条件下ではカーソル操作を実行することはできません。
  このカーソルオープンを成功させ、カーソル操作を実行可能にするためには、NetCOBOLおよびデータソース(ODBCドライバ、データベース、データベース関連製品)の環境が、特定の条件に合致している必要があります。[参照]“15.2.13.3 各ODBCドライバ固有の留意事項”の“特定の条件”

特定の条件

ODBC情報ファイルのサーバ情報に@SQL_CONCURRENCY(カーソルの同時実行)の値として、LOCK、ROWVERまたはVALUESを指定した場合です。

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ことがあります。

- 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

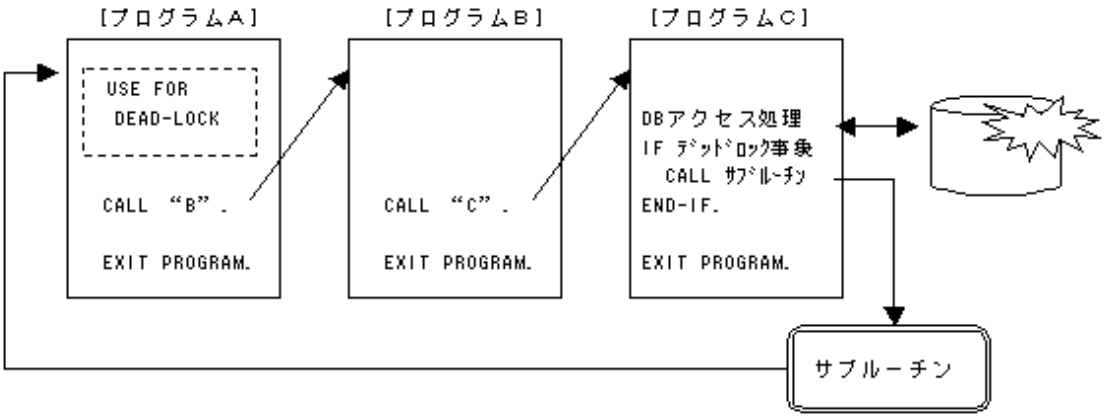
プログラムがデータベースをアクセスする時にアクセス競合が起こり、プログラム同士で占有解除を待つような状態が発生することをデッドロック状態といいます。デッドロック状態が発生すると、デッドロック事象がデータベースからプログラムに通知されます。

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.1.12 デッドロック出口スケジュールサブルーチン(COB_DEADLOCK_EXIT)”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　オブジェクト指向プログラミング機能

16.1 基本的な使い方

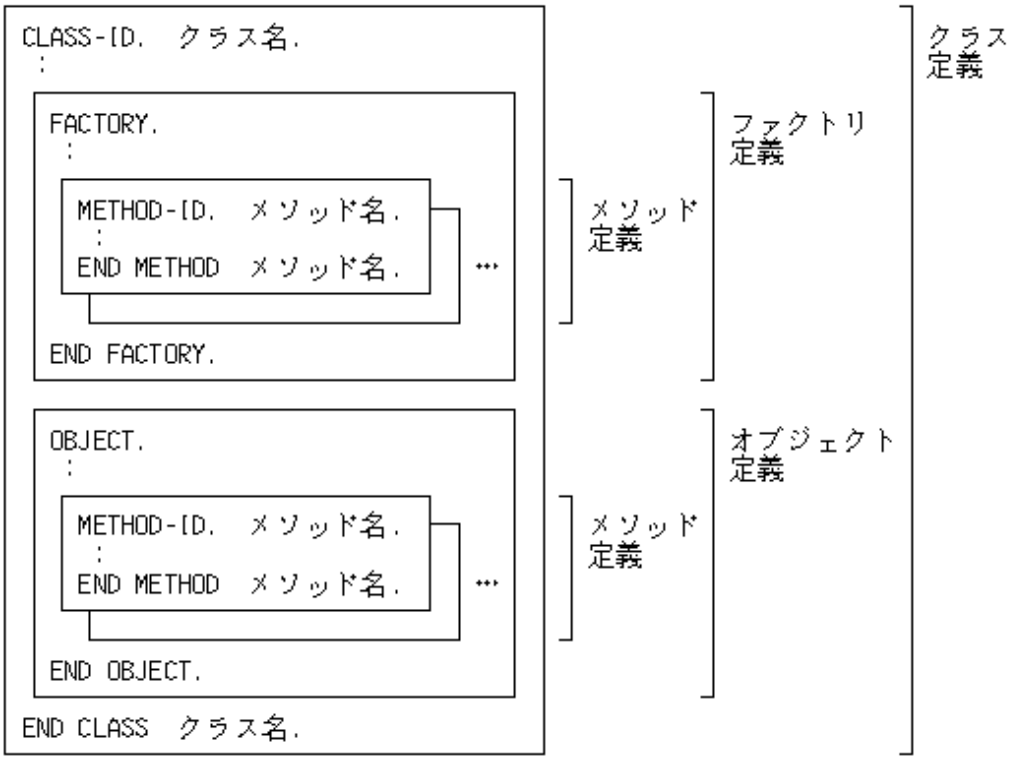
ここでは、オブジェクト指向プログラミングについて、具体的にコーディングレベルで説明します。

16.1.1 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。



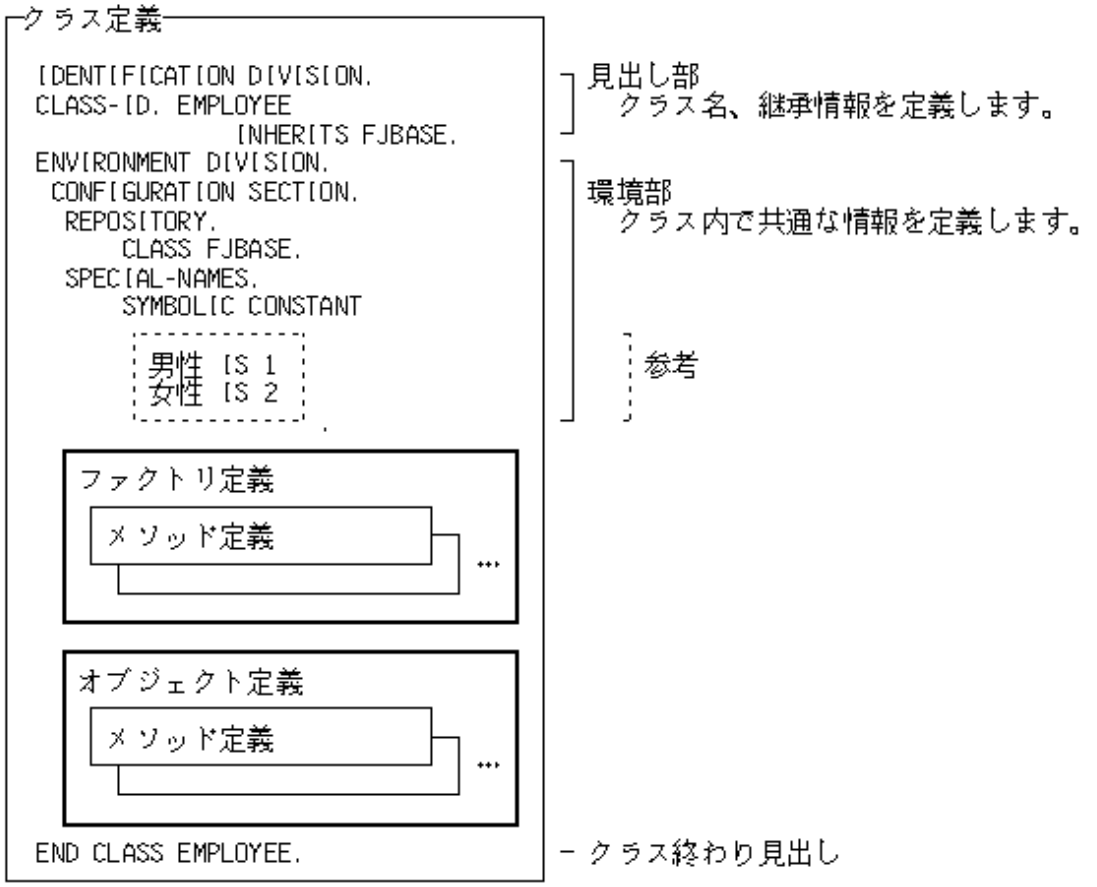
それぞれの定義について、具体的な定義方法や役割について説明します。

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。

16.1.1.1 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)。

16.1.1.2 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。

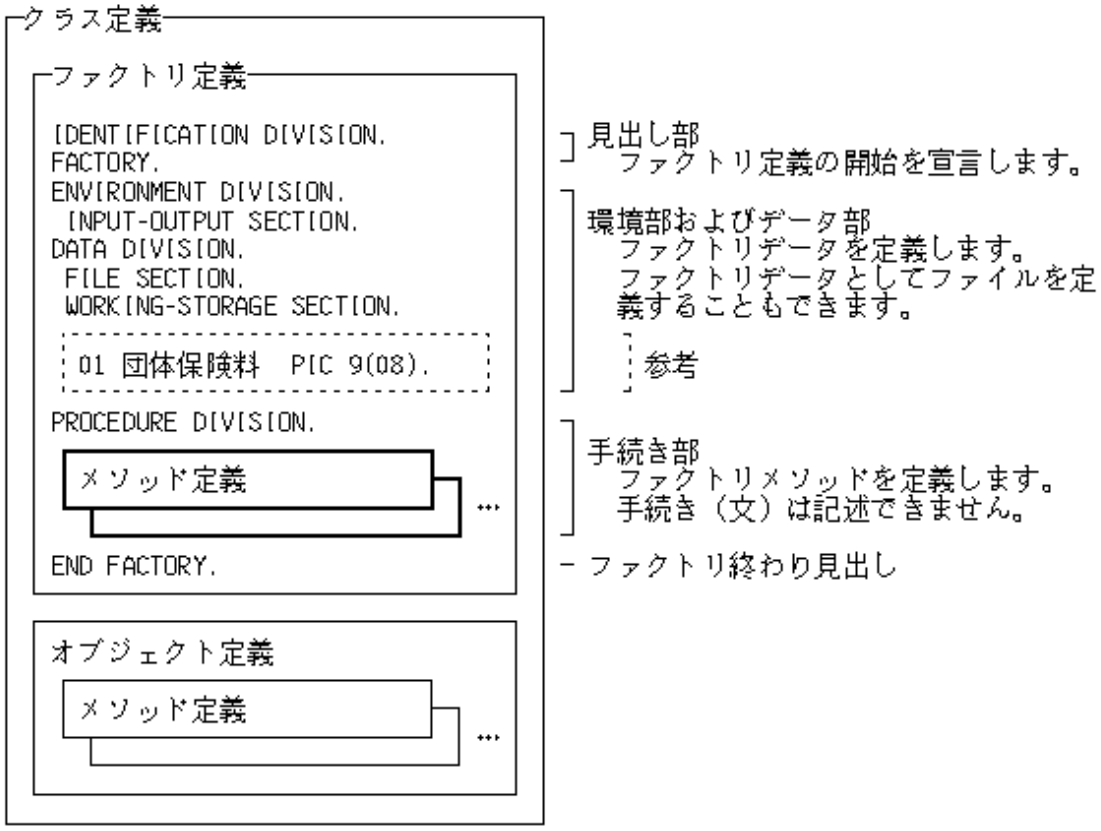
ファクトリ定義は、ファクトリ定義を表すための見出し部、ファクトリデータを定義するための環境部およびデータ部、ファクトリメソッドを定義するための手続き部から構成されます。

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)。

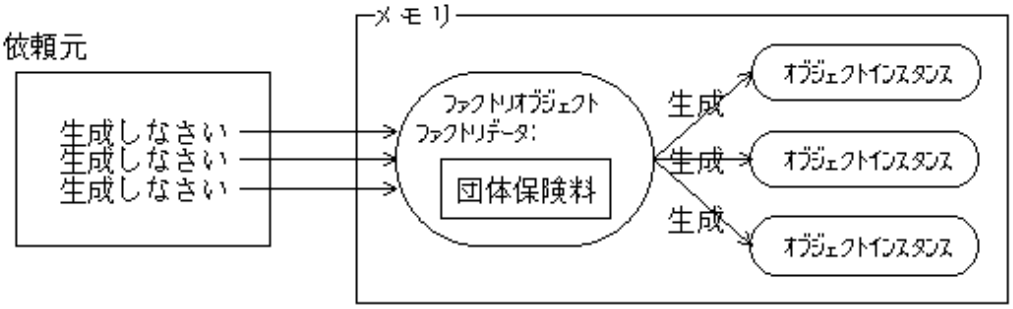
では、具体的にファクトリ定義の役割について説明します。

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、その名のとおりオブジェクトインスタンスを生成する「工場」なのです。

通常、この「生成処理」は、FJBASEクラス(注)を継承することによってクラスに組み込まれるため、「生成処理」を利用者がコーディングする必要はありません。では、他に何を定義するかというと、生成したオブジェクトインスタンスを初期化するメソッドや、生成された複数のオブジェクトインスタンスで共通に利用されるデータなどを定義しておきます。たとえば、上の例のように、ファクトリデータとして「団体保険料」を定義しておきます。これは、給与計算時に全従業員を対象に給与天引きする額であり、ファクトリメソッドを呼び出すことにより設定、参照することができます。クラス共通情報をファクトリデータとして保持することによって、額の増減などに容易に対応できるようになります。

注 ： 標準で提供。詳細は“16.1.3.2 FJBASEクラス”を参照してください。

16.1.1.3 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。

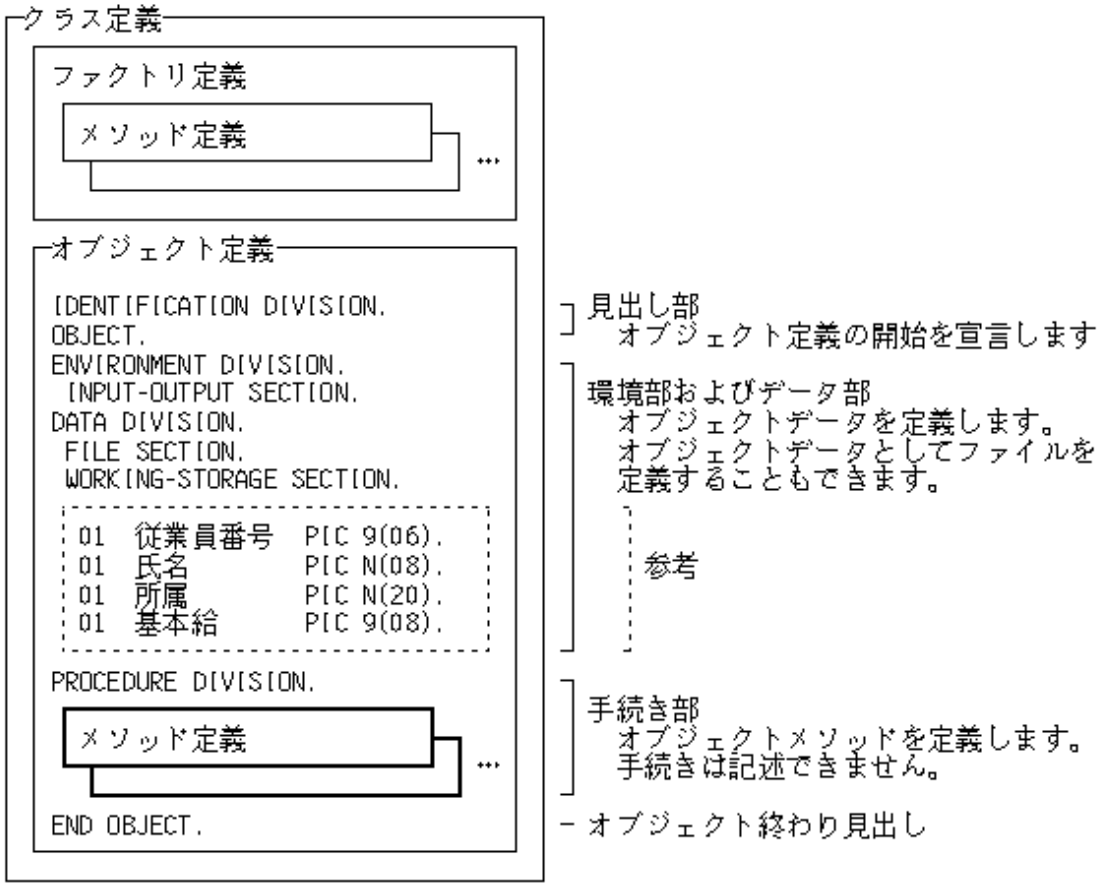
オブジェクト定義の構成はファクトリ定義と同じで、オブジェクト定義を表すための見出し部、オブジェクトデータを定義するための環境部およびデータ部、オブジェクトを定義するための手続き部からなります。

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。

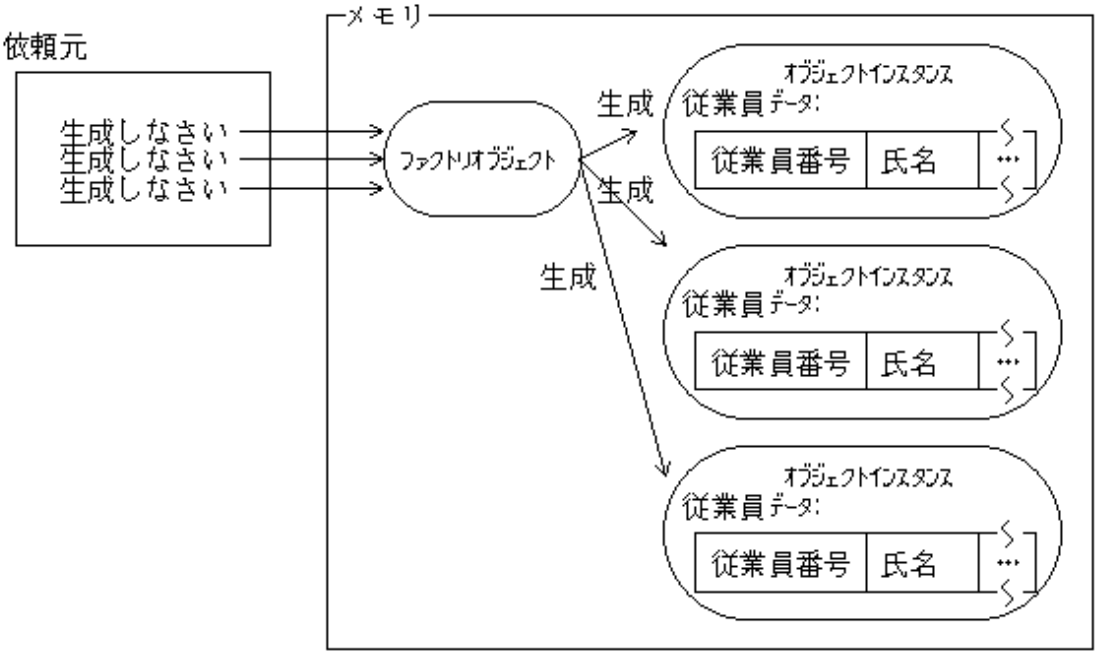


オブジェクト定義の環境部およびデータ部で定義されたデータは、オブジェクトメソッドでだけアクセスすることができます(上図の太線で囲まれている部分)。

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。

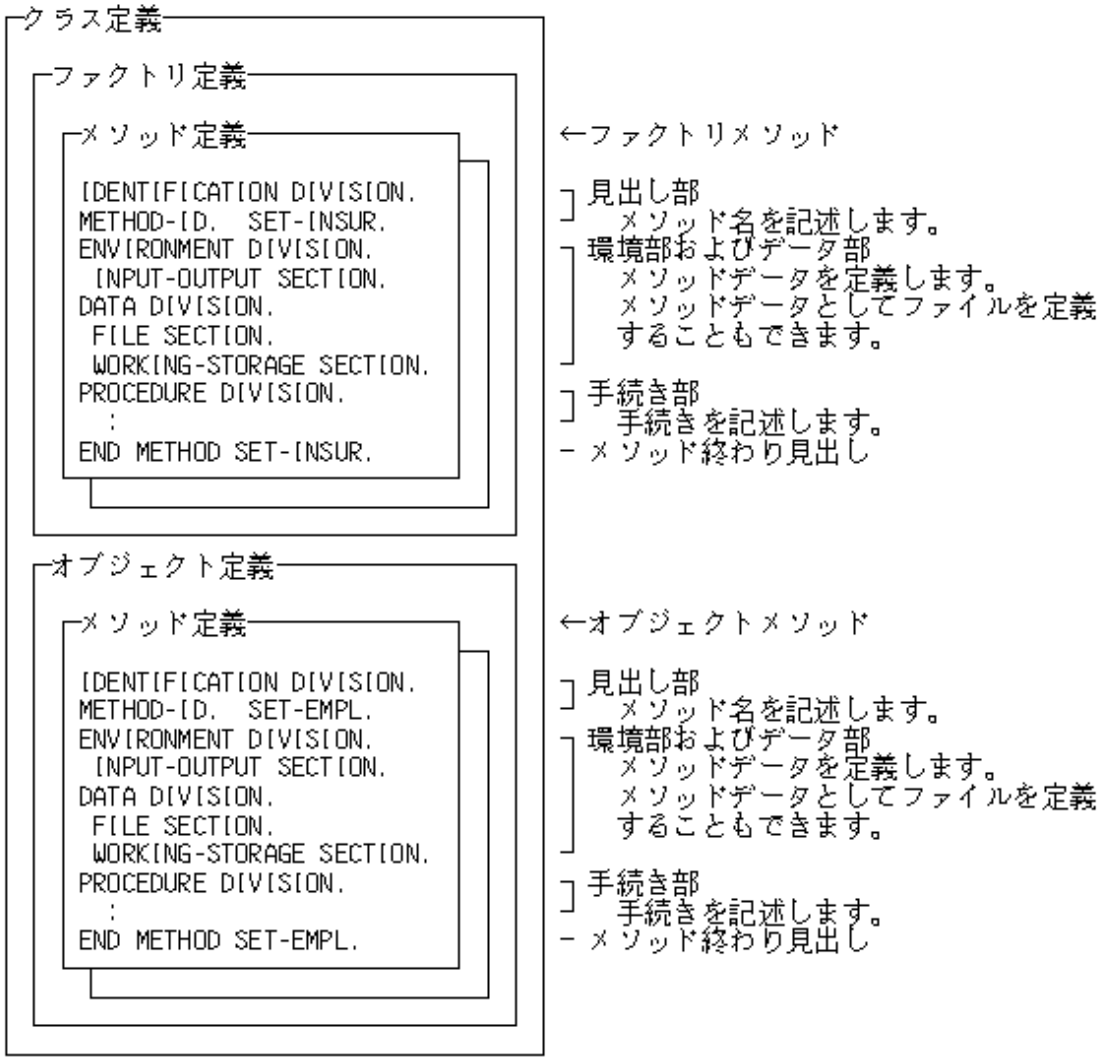


16.1.1.4 メソッド定義

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。

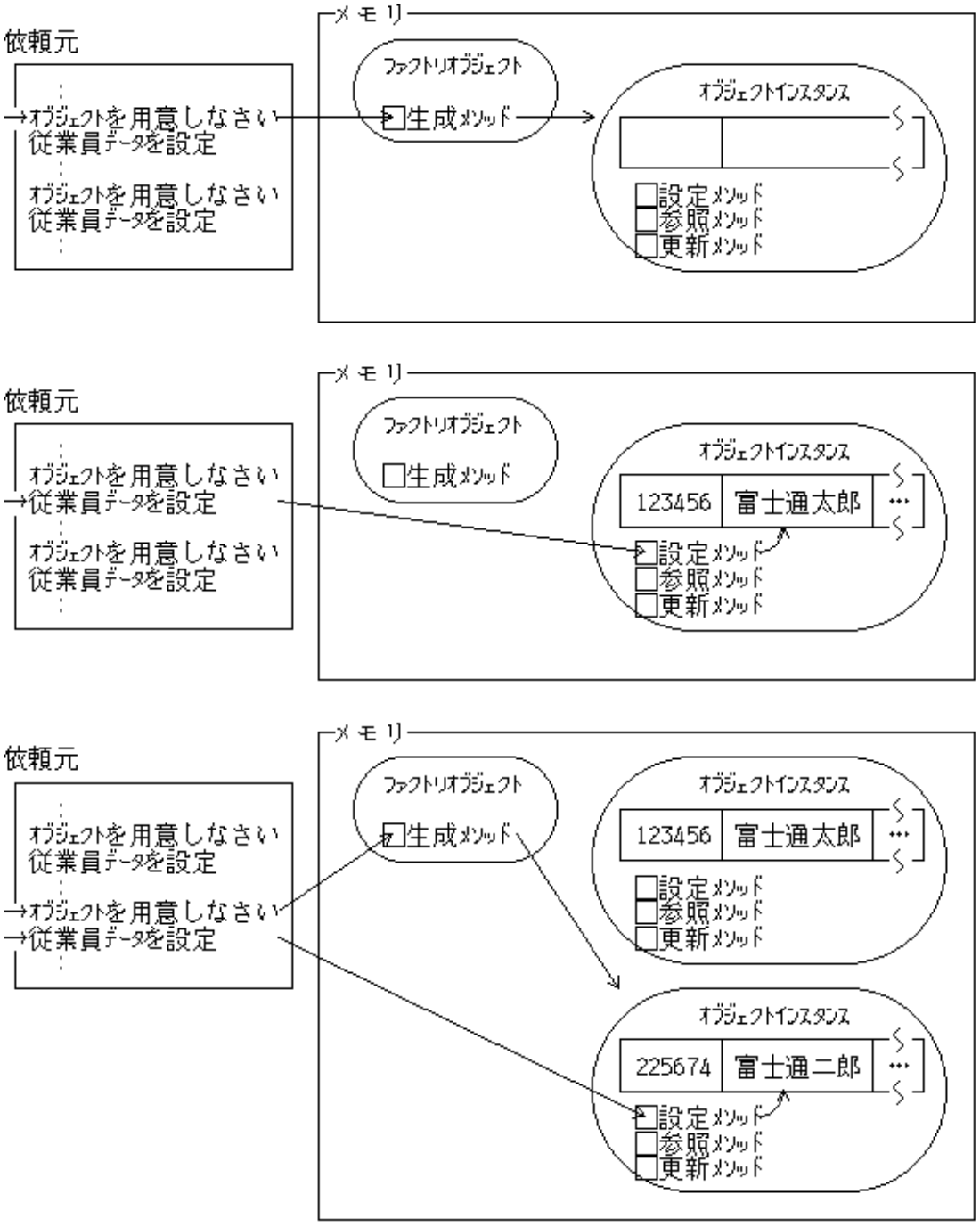


メソッド定義には、ファクトリメソッドとオブジェクトメソッドがあります。どちらも、メソッド内で定義されたデータは、そのメソッド内でだけアクセスすることができます。

では、メソッドの役割について説明します。

ファクトリデータおよびオブジェクトデータは、外部から隠蔽されています。これらのデータを操作(取出しや更新など)するためには、それぞれにメソッドを用意するしかありません。つまり、ファクトリデータは、ファクトリメソッドを介してしかアクセスできません。また、オブジェクトデータは、オブジェクトメソッドを介してしかアクセスできません。

メソッドの実行時イメージは、それぞれのオブジェクトインスタンス中にメソッド(手続き)が存在すると考えた方が分かりやすいでしょう。つまり、オブジェクトインスタンスの操作は以下のイメージとなります。



16.1.2 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。

16.1.2.1 メソッドの呼出し

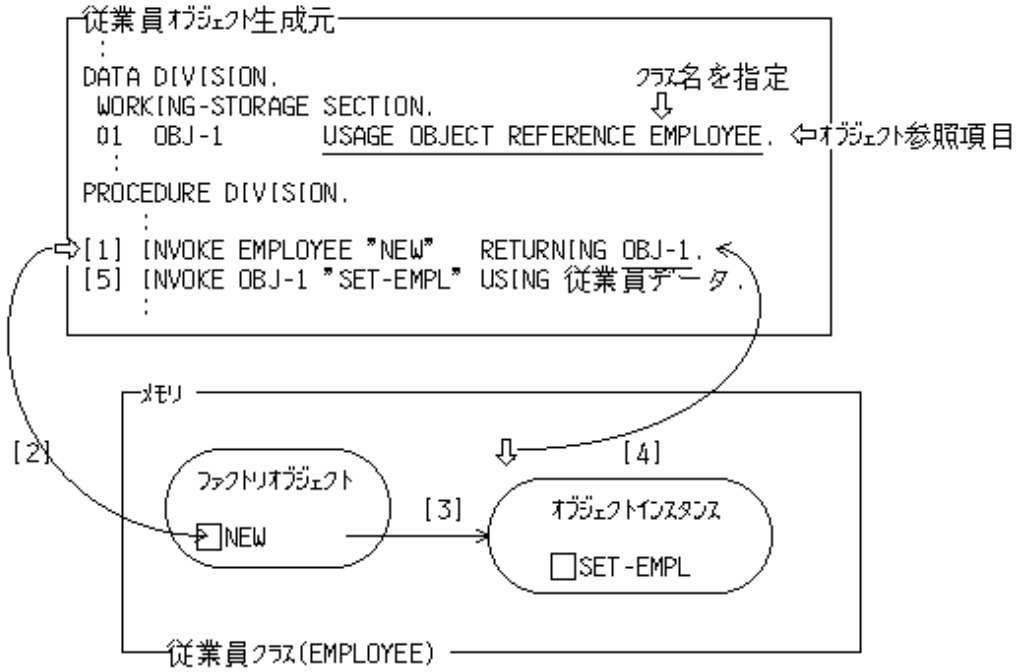
オブジェクトインスタンスを操作するためには、オブジェクトメソッドを呼び出す必要があります。このとき、「どのオブジェクトインスタンス」の「どのメソッド」を呼び出すかを指定します。「どのオブジェクトインスタンス」を表現するのに、オブジェクト参照項目と呼ばれるデータ項目を利用します。

16.1.2.1.1 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16.1 従業員オブジェクトインスタンスの生成

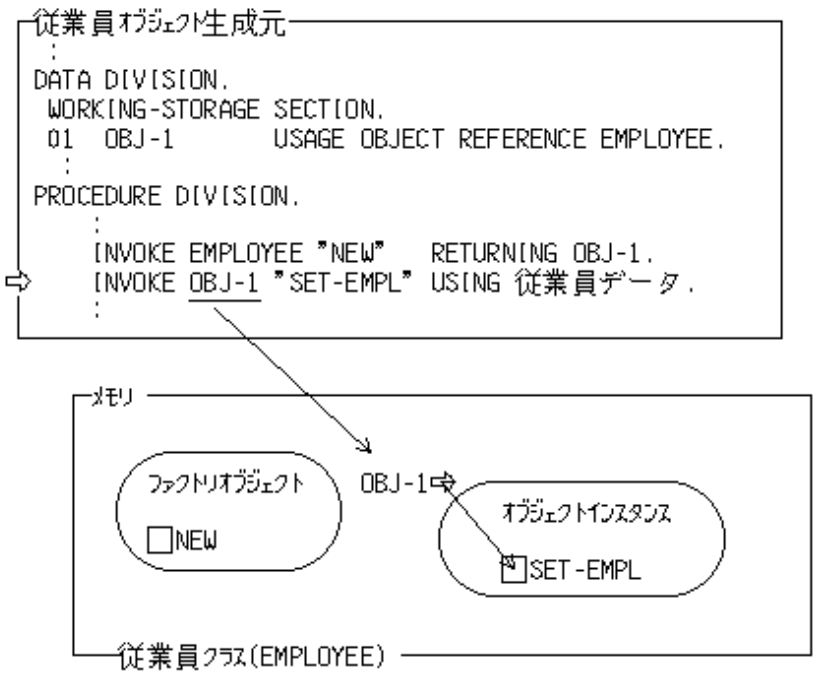


[1][2] ： NEW メソッドを呼び出す。

[3] ： NEW メソッドはオブジェクトインスタンスを生成する。

[4] ： オブジェクト参照を呼出し元へ返却する。

図16.2 従業員オブジェクトインスタンスへのデータ登録



上図のとおり、生成したオブジェクトインスタンスを操作する場合(オブジェクトメソッドを呼び出す場合)は、処理対象となるオブジェクトインスタンスを指しているオブジェクト参照項目を指定しなければなりません。

オブジェクト参照項目は、SET文を用いて他のオブジェクト参照項目に値を代入することができます(MOVE文による転記はできません)。また、IF文などにより、内容を比較することもできます。ただし、この場合、代入または比較されるのはオブジェクト参照データであり、オブジェクトインスタンスが代入または比較されるわけではないので、注意してください。

```
    :
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  OBJ-1       USAGE OBJECT REFERENCE EMPLOYEE.
 01  OBJ-X       USAGE OBJECT REFERENCE EMPLOYEE.
    :
PROCEDURE DIVISION.
    :
    INVOKE EMPLOYEE "NEW" RETURNING OBJ-1.
    SET OBJ-X TO OBJ-1.
    INVOKE OBJ-X "SET-EMPL" USING 従業員データ.
    :
```

注意

- 初期値はNULLで、VALUE句により初期値を与えることはできません。
- オブジェクト参照項目の内部形式については、意識しないようにコーディングしてください。COBOLシステムが管理するデータのため、無理に内容を変更した場合、正常に動作できなくなります。
- CALL文でオブジェクト参照項目を受け渡す場合、USAGE句が一致していないと正しく受け渡すことができません。

16.1.2.1.2 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16.1 従業員オブジェクトインスタンスの生成”のINVOKE文について説明します。

[1]は、オブジェクトインスタンスを生成するためにEMPLOYEEクラスのNEWメソッドを呼び出しています。ここでは、

- 「どのオブジェクト」 → ファクトリオブジェクトの、(注)
- 「どのメソッド」 → NEWメソッドを、
- 「どのようなパラメタ」 → OBJ-1(オブジェクト参照)を復帰値として

呼び出す。という意味になります。

注 ： 通常、ファクトリオブジェクトはクラス名で表現されます。

[5]は、[1]で生成したオブジェクトインスタンスに初期データとして従業員の情報を設定するためにSET-EMPLメソッドを呼び出しています。

このときのINVOKE文は、

- 「どのオブジェクト」 → OBJ-1で表されるオブジェクトインスタンスの、
- 「どのメソッド」 → SET-EMPLメソッドを、
- 「どのようなパラメタ」 → 従業員情報を入力として

呼び出す。という意味になります。

注意

INVOKE文にメソッド名を識別する一意名を指定した場合、メソッド名として有効となる文字列の最大は、指定された領域の先頭から255バイトまでです。256バイト以降の文字列は無視されます。また、このとき、文字列の後ろに埋められた空白も無視されます。

16.1.2.1.3 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。

```
呼出し元
      :
  INVOKE OBJ-1 "SET-EMPL"  USING 従業員データ.
      :

呼出し先
      :
>IDENTIFICATION DIVISION.
 METHOD-ID.     SET-EMPL.
 DATA DIVISION.
      :
  LINKAGE SECTION.
  01  従業員データ.
      02  従業員番号  PIC 9(06).
      02  氏名        PIC N(08).
      :
 PROCEDURE DIVISION  USING 従業員データ.
      :
```

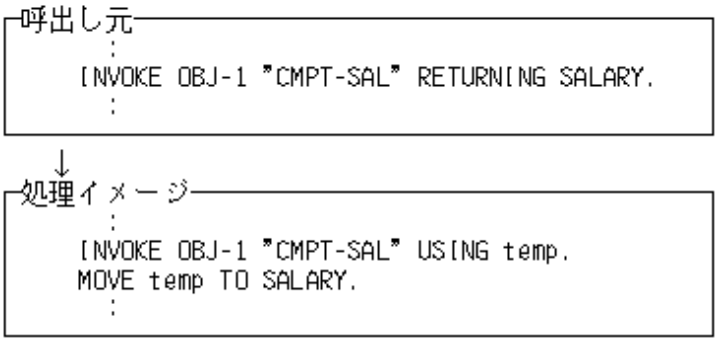
RETURNING指定

RETURNING指定は、呼出し先からの復帰値を受け取るために指定するもので、USING指定とは少し意味合いが異なります。また、USING指定が複数個のパラメタを指定できるのに対し、RETURNING指定は、1つだけ指定可能です。

```
呼出し元
      :
  INVOKE OBJ-1 "CMPT-SAL" RETURNING SALARY.
      :

呼出し先
      :
>IDENTIFICATION DIVISION.
 METHOD-ID.     CMPT-SAL.
 DATA DIVISION.
  LINKAGE SECTION.
  01  SALARY    PIC 9(08).
 PROCEDURE DIVISION  RETURNING SALARY.
       :
```

RETURNING指定は、復帰値であるため、呼出し元で設定された値を呼出し先で参照することはできません。つまり、一方通行の関係となります。

呼出し元の処理イメージは、下図のとおりです。



USING指定とRETURNING指定を同時に指定することもできます。それぞれの用途に合わせて利用してください。

16.1.2.2 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。

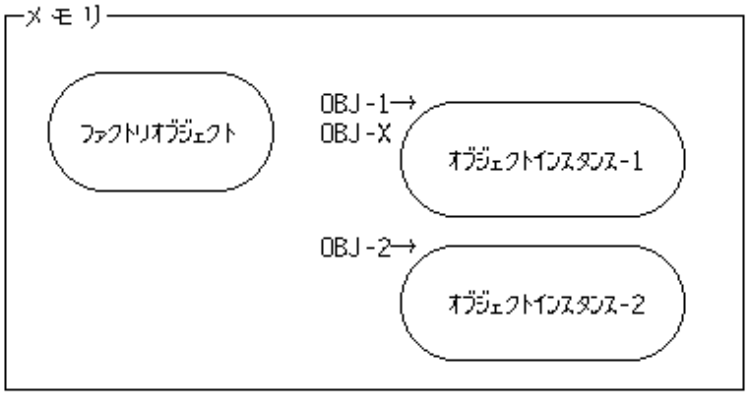
オブジェクトインスタンスの寿命

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。このような終り方をしない場合、COBOLランタイムシステムは、COBOLの実行環境閉鎖時に、残っているオブジェクトインスタンスを強制的にメモリ上から解放します。このとき、終了処理メソッド_FINALIZEは呼び出されません。また、マルチスレッドプログラムでは、メモリリークが発生するため注意が必要です。[参照]“18.10.1 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。

呼出し元

```
    :
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  OBJ-1       USAGE OBJECT REFERENCE EMPLOYEE.
 01  OBJ-2       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 EMPLOYEE "NEW" RETURNING OBJ-2.
      :
```



上図の状態で、以下の手続きが実行された場合

```
    :
   SET OBJ-2 TO NULL.                          ... [1]
   SET OBJ-1 TO NULL.                          ... [2]
   SET OBJ-X TO NULL.                          ... [3]
    :
```

[1]の実行により、OBJ-2だけで管理されていたオブジェクトインスタンス-2は、削除されます。

[2]の実行により、OBJ-1にNULLオブジェクトを代入しても、オブジェクトインスタンス-1はOBJ-Xによって管理されているため、削除されません。

[3]の実行により、さらにOBJ-XにNULLオブジェクトを代入すると、オブジェクトインスタンス-1を管理しているオブジェクト参照項目はなくなるため、削除されます。

ただし、ここでいう「削除」とは、アプリケーションから論理的に見えなくなるだけであり、メモリ上から解放されるわけではありません。メモリ上からの解放は、COBOLランタイムシステムが最適なタイミングで自動的に行います。これをガーベージコレクションと呼びます。

16.1.3 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。

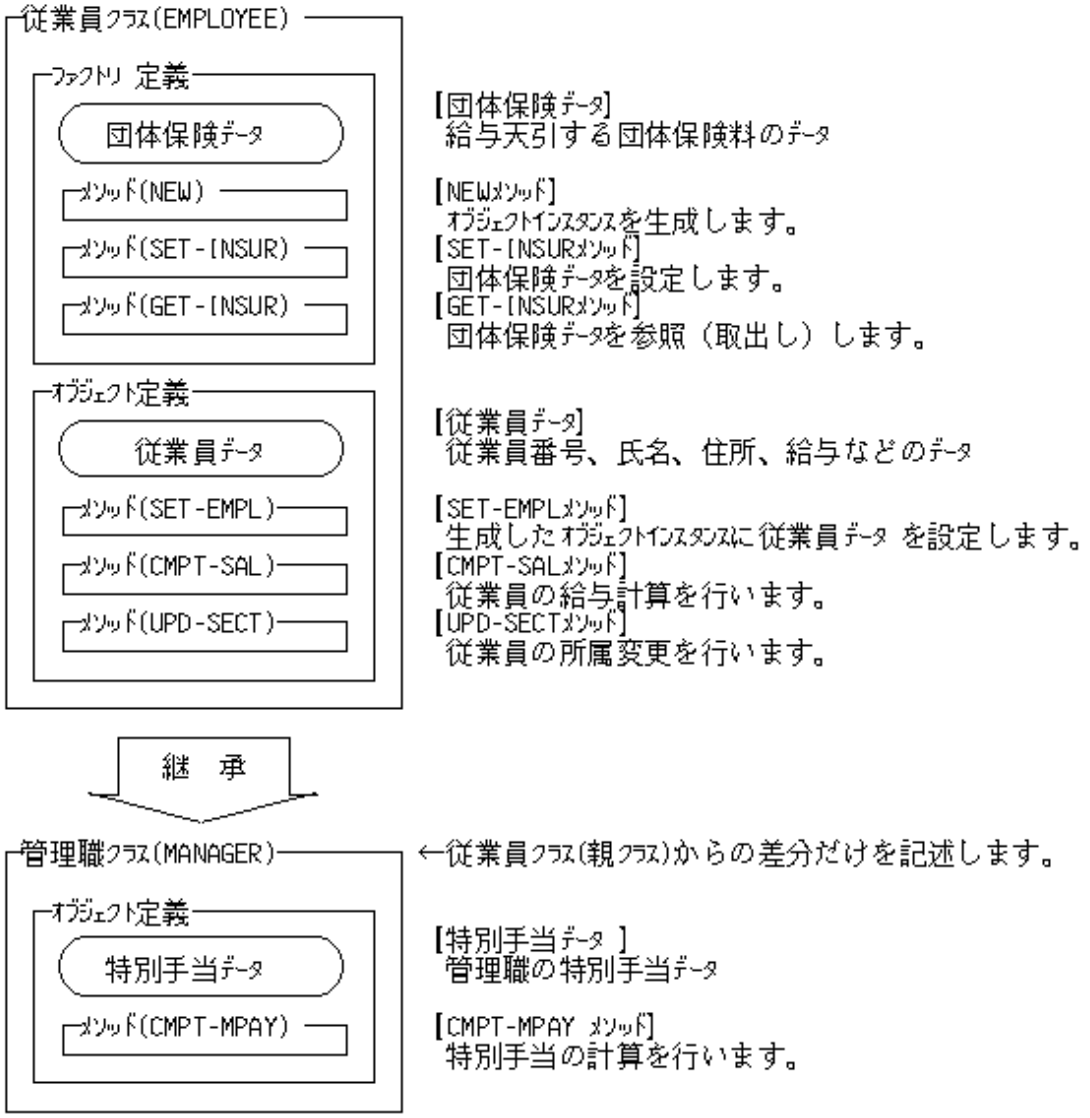
16.1.3.1 継承の概念と実現

継承の概念

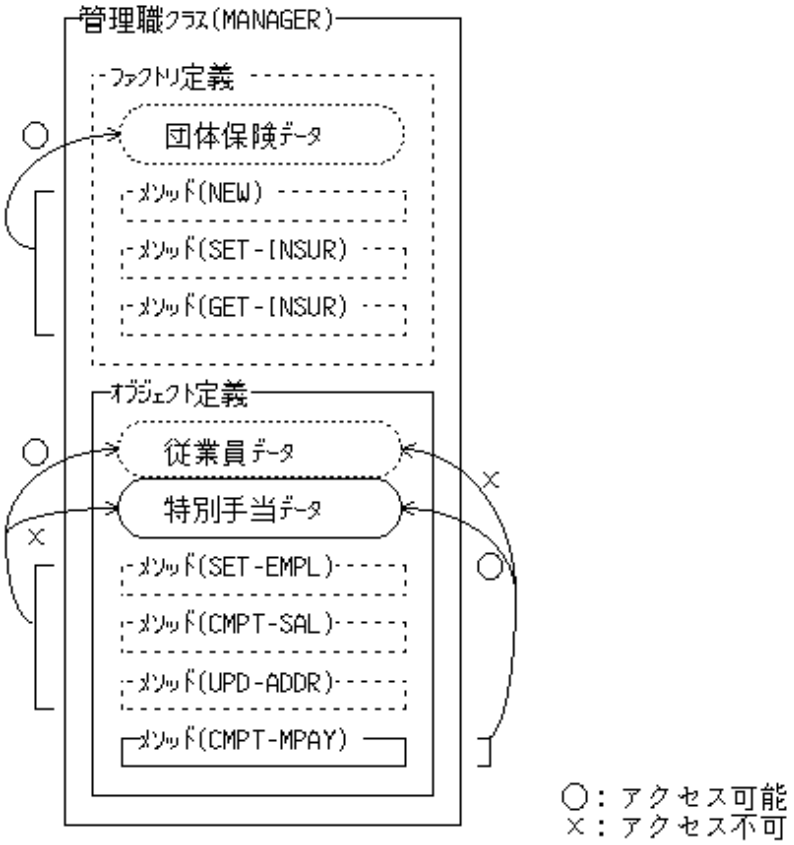
従来のプログラミング手法では、既存の部品(ルーチン)とよく似た機能を持った部品を作成する場合、既存のソースプログラムを複写し、複写したソースプログラムをベースにしてプログラミングする方法をとっていました。

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)を子クラス、継承されたクラスを親クラスと呼びます。上図では、従業員クラス(EMPLOYEE)が親クラスで、管理職クラス(MANAGER)が子クラスになります。

データのアクセス

暗黙定義されたデータ(団体保険データ、従業員データ)および明示定義されたデータ(特別手当データ)のアクセスについて説明します。

ファクトリデータおよびオブジェクトデータは、そのクラス定義で明示定義されたメソッドでだけアクセスすることができます。つまり、上図の場合、オブジェクトデータ(オブジェクトインスタンス)としては、従業員データと特別手当データの両方を持ちますが、明示定義された特別手当データは、明示定義されたオブジェクトメソッド(CMPT-MPAY)でだけアクセス可能です。逆に、暗黙定義された従業員データは、暗黙定義されたオブジェクトメソッド(SET-EMPL、CMPT-SALおよびUPD-SECT)でだけアクセス可能になります。

継承の定義方法

では、実際に継承を定義してみましょう。

継承は、クラス名段落(CLASS-ID)のINHERITS句に親クラス名を指定することで実現できます。このとき、環境部のリポジトリ段落に必ず親クラスを宣言してください。

管理職クラス

```
**------------------**
** クラス定義       **
**------------------**
```

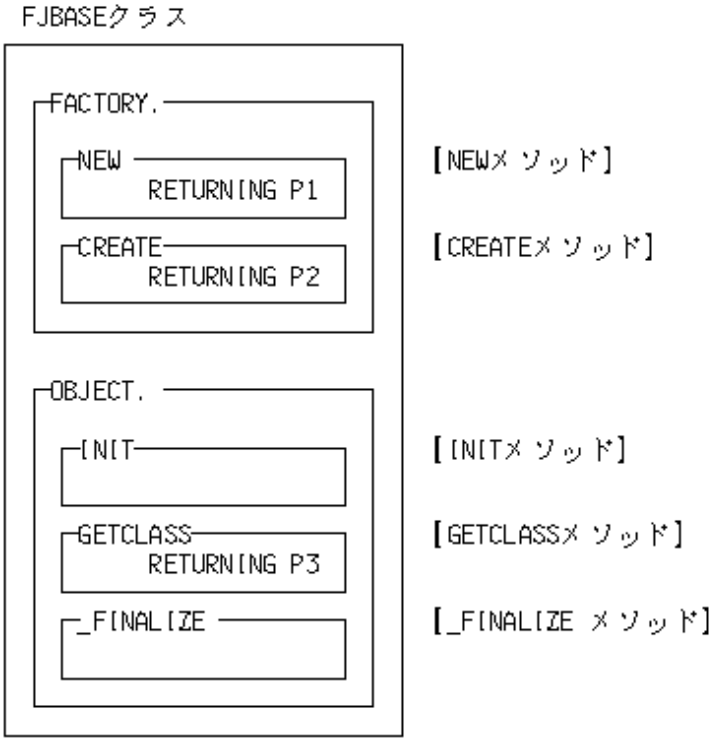
```
 IDENTIFICATION DIVISION.
 CLASS-ID. MANAGER
             INHERITS EMPLOYEE.          ←親クラスを指定します。
 ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
   REPOSITORY.
       CLASS EMPLOYEE.                   ←親クラスを宣言します。
**------------------**
** オブジェクト定義 **
**------------------**
 IDENTIFICATION DIVISION.
 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。

16.1.3.2 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。

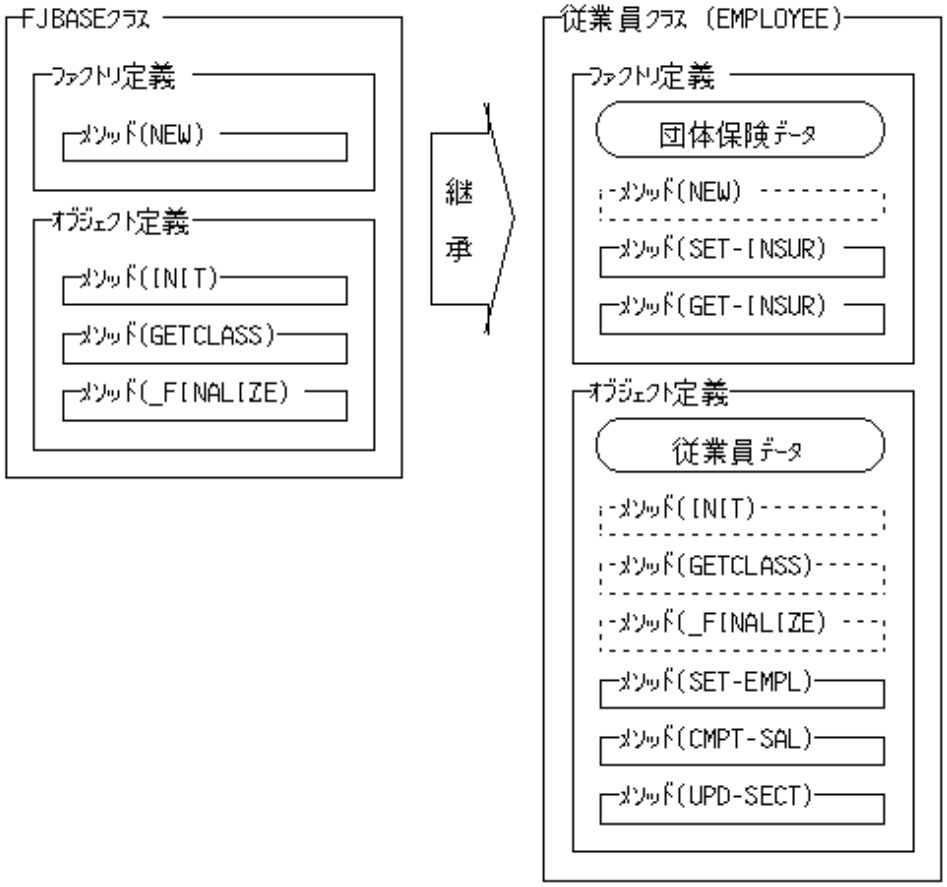


各メソッドの詳細については、“COBOL文法書”を参照してください。

INITメソッドおよび_FINALIZEメソッドについては、“16.1.12 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。



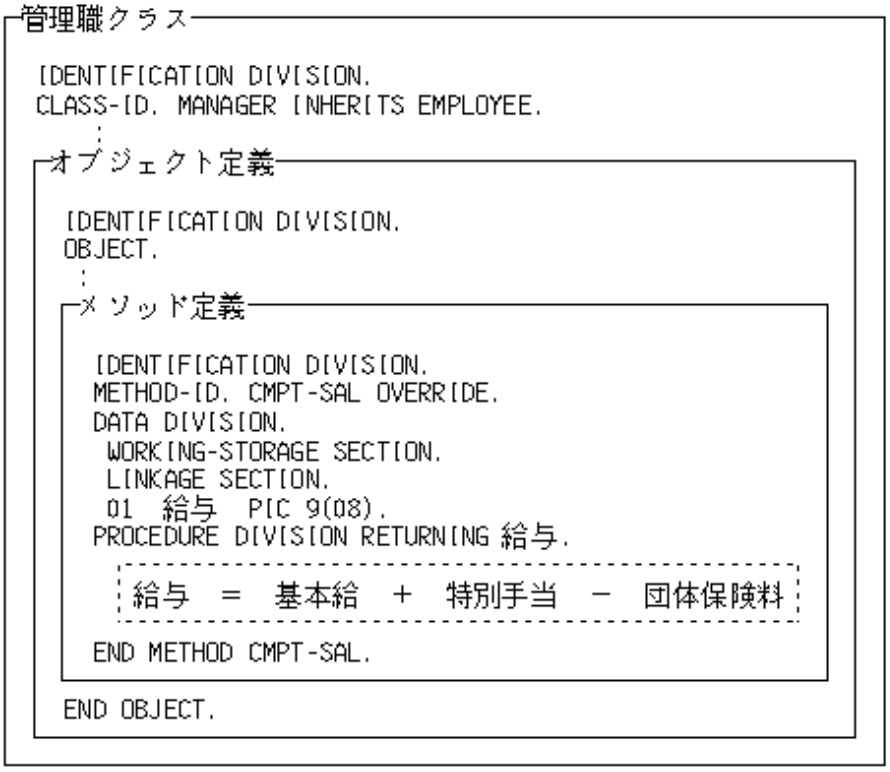
16.1.3.3 メソッドの上書き

クラスを継承する場合、「メソッド名やインタフェースは同じでよいが、処理を少し変更(追加)したい」ということが多くあります。このようなとき、継承をあきらめて新規に似たようなクラスを作成する必要はありません。OVERRIDE句を利用することによって、メソッドを上書きすることができます。

従業員クラスを継承した管理職クラスの場合を考えてみましょう。

管理職の場合、給与計算メソッド(CMPT-SAL)で、「特別手当を加算する」必要があります。つまり、従業員クラスから継承したCMPT-SALメソッドに処理を加える必要があります。

このような場合、OVERRIDE句を利用してメソッドを上書きすることができます。



メソッドの上書きは、直接の親クラスで明示または暗黙に定義されたメソッドに対して行うことができます。また、親クラスで上書きされているメソッドをさらに上書きすることも可能です。

ただし、インタフェース(パラメタ)は上書きされるメソッドと同じでなければなりません。

16.1.4 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。

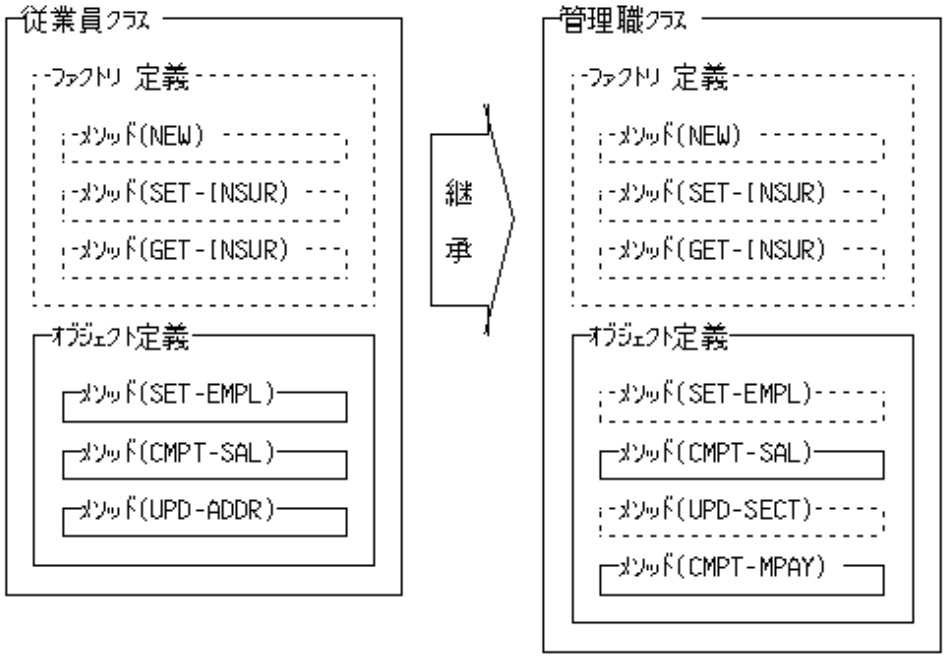
ここでは、適合の概念や規則について、具体例を用いて説明します。

16.1.4.1 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」ことになります。つまり、適合の関係は相互に成立するものではなく、単一方向に成立するものといえます。

この適合関係は、オブジェクト参照の代入時や、オブジェクト参照項目を使用したメソッド呼出し時などに意味を持ってきます。たとえば、代入(SET文)の場合、適合関係が成立するクラスのオブジェクト参照項目間の代入(管理職クラスのオブジェクト参照項目を従業員クラスのオブジェクト参照項目へ)はできますが、適合関係が成立しない場合、転記はできません(翻訳エラーとなります)。このように、適合関係をチェックすることを適合チェックと呼んでいます。

```
   :
 WORKING-STORAGE SECTION.
 01  OBJ-1       USAGE OBJECT REFERENCE EMPLOYEE.
 01  OBJ-2       USAGE OBJECT REFERENCE MANAGER.
   :
PROCEDURE DIVISION.
      :
    INVOKE EMPLOYEE "NEW" RETURNING OBJ-1.
    SET OBJ-2 TO OBJ-1.                          … [1]
      :
    INVOKE MANAGER "NEW" RETURNING OBJ-2.
    SET OBJ-1 TO OBJ-2.                          … [2]
      :
```

[1] 従業員クラスから管理職クラスへの適合関係は成立しないため、翻訳エラーになります。

[2] 管理職クラスから従業員クラスへの適合関係が成立するため、問題なくオブジェクト参照が転記されます。

メソッド呼出しの適合チェックの場合、メソッドのインタフェースに対してチェックが行われます。たとえば、INVOKE文に指定されたメソッドがクラス中に存在しない場合や、メソッドに渡すパラメタが異なる場合などにチェックされます。

```
   :
 WORKING-STORAGE SECTION.
 01  OBJ-1       USAGE OBJECT REFERENCE EMPLOYEE.
 01  OBJ-2       USAGE OBJECT REFERENCE MANAGER.
```

```
   :
PROCEDURE DIVISION.
       :
    INVOKE EMPLOYEE "NEW" RETURNING OBJ-1.
    INVOKE OBJ-1 "CMPT-MPAY" RETURNING 特別手当.   … [1]
       :
    INVOKE MANAGER "NEW" RETURNING OBJ-2.
    INVOKE OBJ-2 "CMPT-MPAY" USING 特別手当.       … [2]
       :
```

[1] 存在しないメソッドが指定されたためにエラーとなります。

[2] CMPT-MPAYメソッドへのパラメタが異なるためにエラーとなります。

では、なぜこのような適合という概念があるのか、従業員クラスと管理職クラスの関係を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。

16.1.4.2 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、“16.1.6.2 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いため、コーディング(目的プログラムができるまで)は容易ですが、実行時の適合チェックによって、手戻りの発生する可能性が大きくなります。

[2] これまでの例でも用いられた定義で、指定されたクラス(例では従業員クラス)のオブジェクト参照が格納されることを明示指定する定義です。この場合、従業員クラスまたは従業員クラスの子クラス(管理職クラス)のオブジェクト参照を格納することができます。

[3] 指定されたクラスのオブジェクト参照だけを格納する定義です。この場合、指定されたクラスに適合するクラスのオブジェクト参照を格納することはできなくなります。

また、[2]および[3]については、ファクトリオブジェクトまたはオブジェクトインスタンスによって指定が異なります。例では、オブジェクトインスタンスのオブジェクト参照を格納する指定で、ファクトリオブジェクトの場合は、以下のとおり定義します。

```
   :
 01  OBJ-2  USAGE OBJECT REFERENCE FACTORY OF EMPLOYEE.
 01  OBJ-3  USAGE OBJECT REFERENCE FACTORY OF EMPLOYEE ONLY.
   :
```

なお、ファクトリオブジェクトのオブジェクト参照項目については、これまで説明していませんでしたが、使い方はオブジェクトインスタンスの場合と同じです(下図を参照してください)。

```
   :
 WORKING-STORAGE SECTION.
 01  OBJ-F  USAGE OBJECT REFERENCE FACTORY OF EMPLOYEE.
 01  OBJ-1  USAGE OBJECT REFERENCE EMPLOYEE.
   :
PROCEDURE DIVISION.
       :
    SET OBJ-F TO EMPLOYEE.
```

```
    INVOKE OBJ-F "NEW" RETURNING OBJ-1.
      :
```

16.1.4.3 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。

16.1.4.3.1 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]は適合エラーになりません。

OBJ-3は、従業員クラスのオブジェクト参照だけ格納可能のため、[7]、[8]、[9]はどれも適合エラーとなります。適合チェックは翻訳時に行われます。

16.1.4.3.2 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タ不整合による障害が翻訳時に取り除けるというメリットを得ることができます。

16.1.5 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、“16.2.4.2.1 リポジトリファイル”を参照してください。

16.1.5.1 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。以下に、リポジトリファイルの利用方法を示します。

- 継承を実現するため、コンパイラへ入力する。
- 適合チェックを行うため、コンパイラへ入力する。

以下にそれぞれについて説明します。

16.1.5.2 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になりますが、翻訳時に、FJBASEクラスのリポジトリファイルを入力する必要はありません。

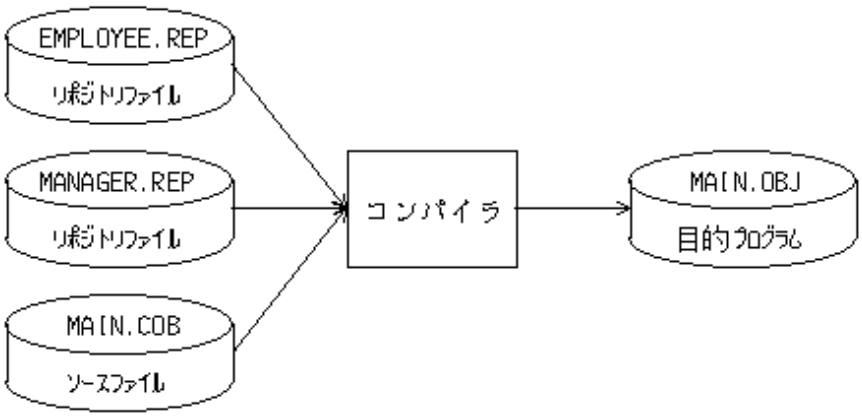
16.1.5.2.1 適合チェックの実現

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

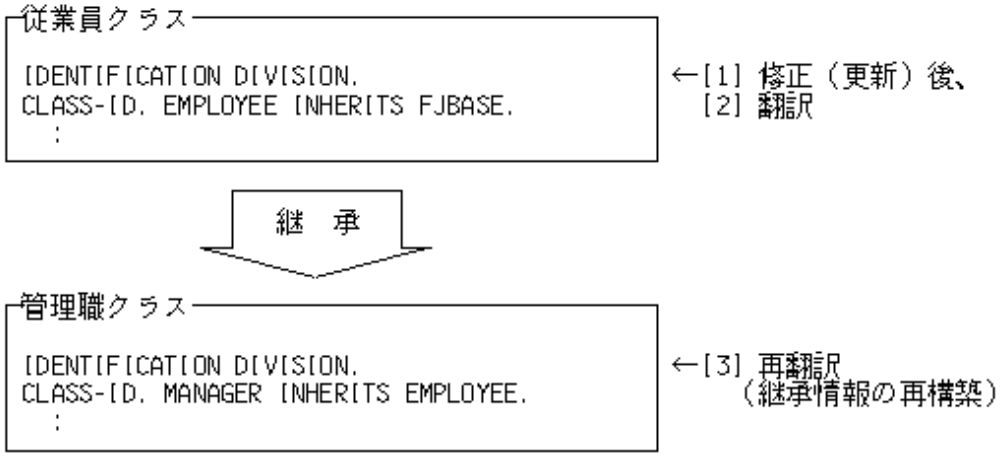
```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     CLASS EMPLOYEE
     CLASS MANAGER.
  :
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  OBJ-1       USAGE OBJECT REFERENCE EMPLOYEE.
 01  OBJ-2       USAGE OBJECT REFERENCE MANAGER.
  :
PROCEDURE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



16.1.5.3 リポジトリファイル更新の影響

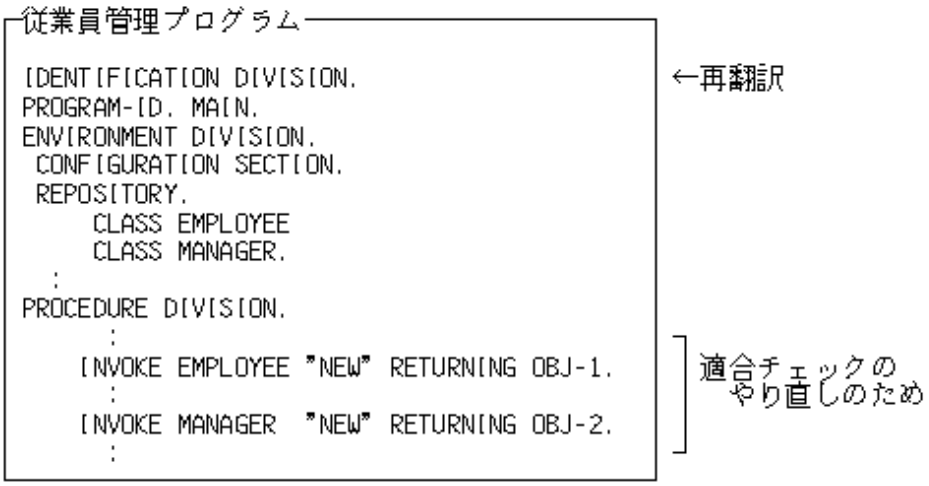
目的プログラムの生成後、親クラスや呼び出すクラスに修正が入った場合、どのように対応すればよいのでしょうか。

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、何も修正していなくても再翻訳が必要になります(継承情報の再構築のため)。

また、修正したクラスを呼び出しているプログラムやクラスについても再翻訳が必要です(適合チェックのやり直しのため)。



これらの再翻訳は、利用者が行わなければなりません。そのため、一度構築したクラス定義を修正する場合は、十分注意してください。

16.1.6 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。

16.1.6.1 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、“16.1.6.3 定義済みオブジェクト一意名SUPER”を参照してください。

16.1.6.2 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。

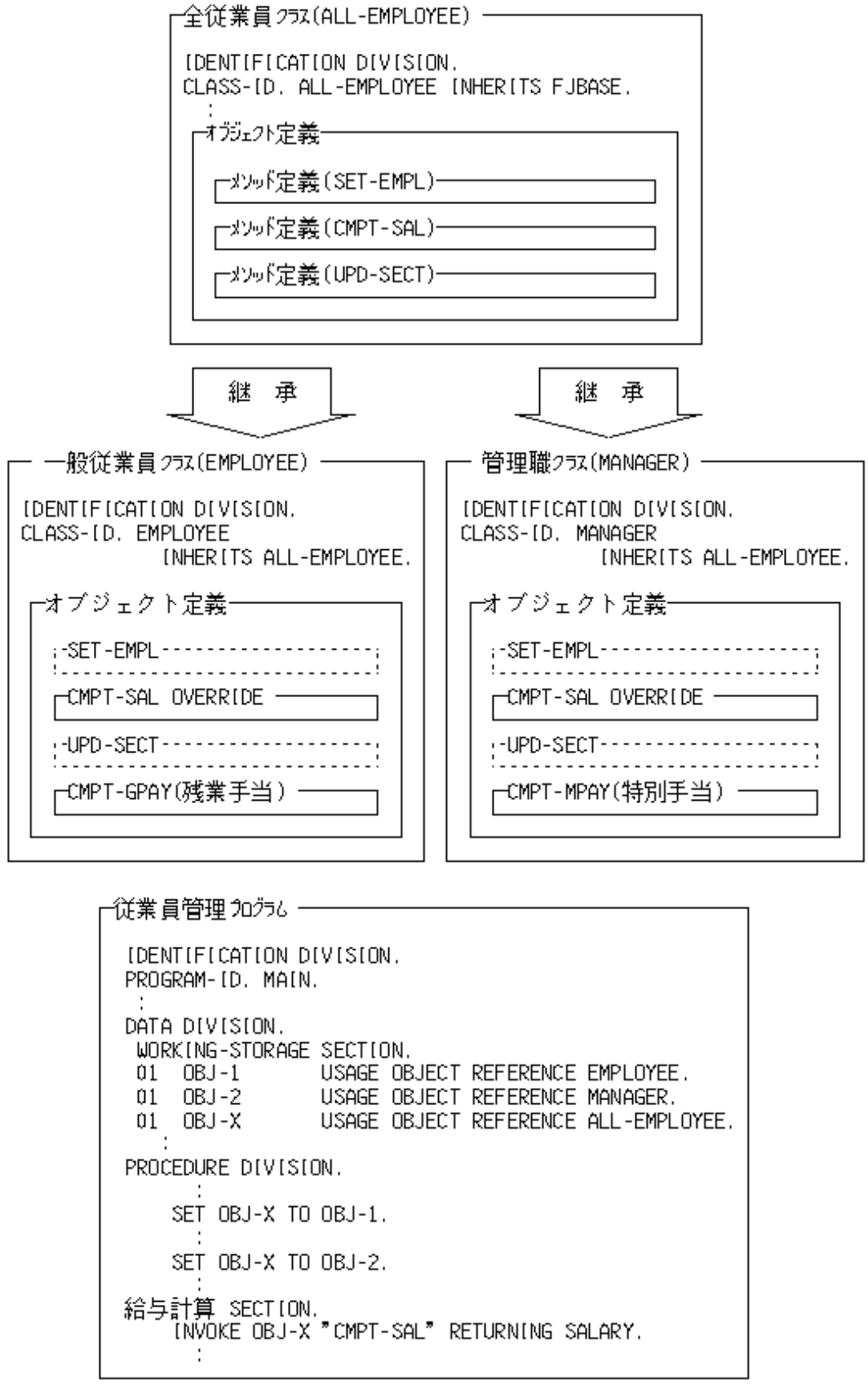
[2] OBJ-2には、従業員クラスに適合するクラスのオブジェクト参照が格納できるため、[1]の場合と同様に動的束縛になります。

[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、“16.1.6.4 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、共通のインタフェースを持つオブジェクトの処理時に利用することができます。

これまで、管理職クラスは従業員クラスの子クラスに位置付けられていましたが、実際には、一般従業員に固有な処理も必要になります(たとえば、給与計算処理での残業手当の加算など)。そのため、抽象化クラスとして、管理職を含めた全従業員対象のクラス(ALL-EMPLOYEEクラス)を作成します。この抽象化クラスの定義によって、多態を利用した共通処理が実現できます。



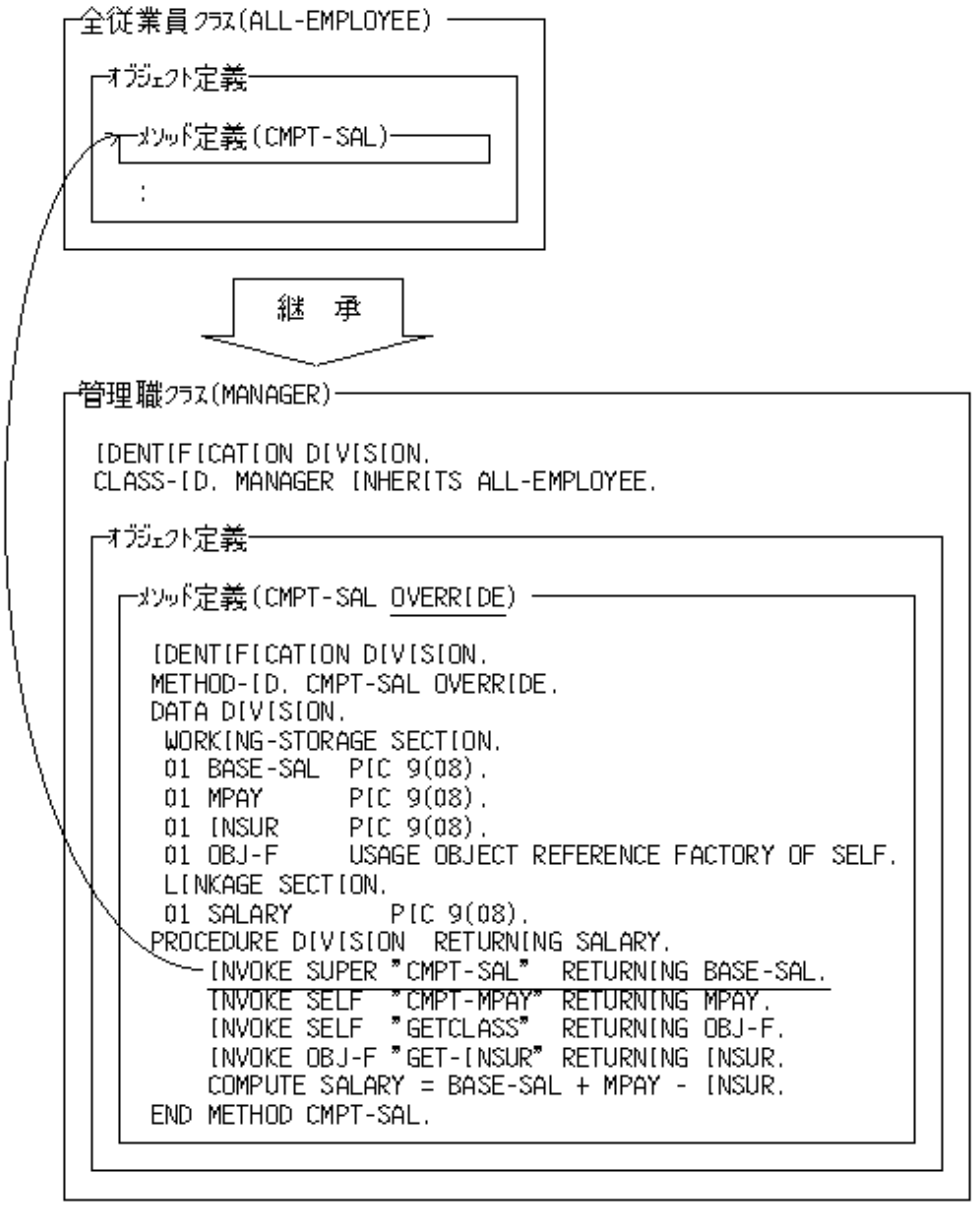
上図のようにオブジェクトを意識せず(一般従業員なのか、管理職なのか)に給与計算処理を行うことができます。つまり、1つのオブジェクト参照項目によって、複数の別定義オブジェクトを操作することができたことになります。これを多態と呼びます。

16.1.6.3 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。



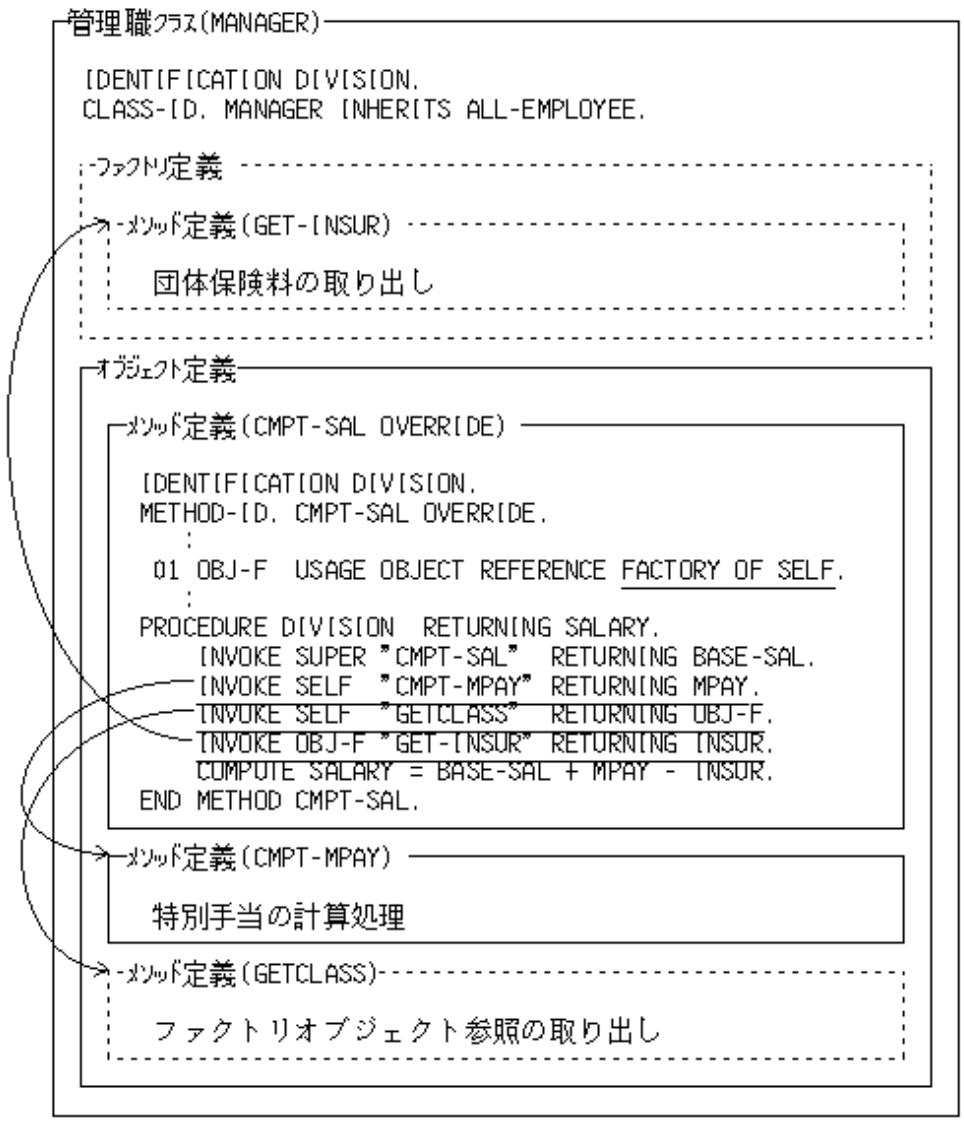
このように、親クラスを表現する場合に、定義済みオブジェクト一意名SUPERが利用されます。

16.1.6.4 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。

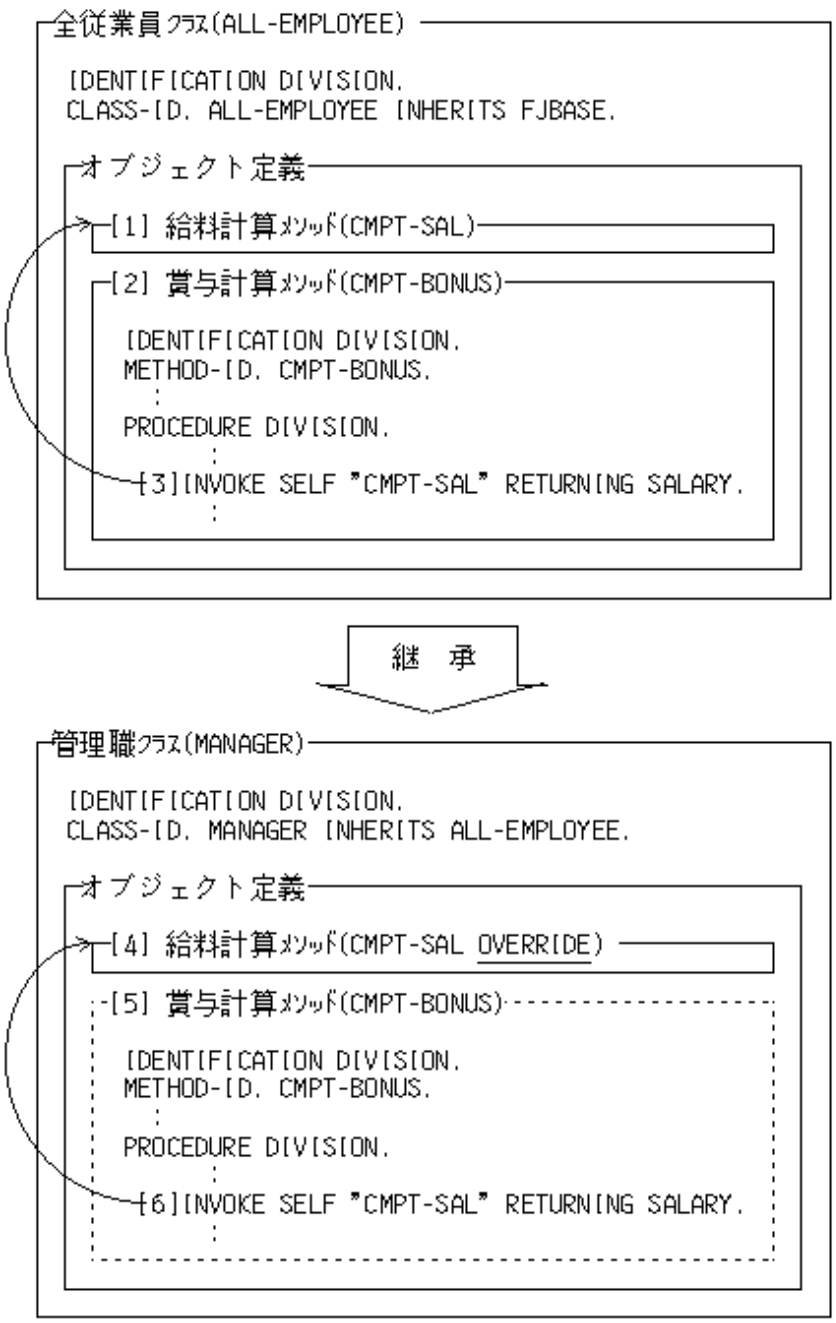
この定義済みオブジェクト一意名SELFの使用方法について説明します。

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、というときに定義済オブジェクト一意名SELFを利用します。



定義済みオブジェクト一意名SELFの使用時に注意しなければならないのは、呼び出すメソッドは、つねに実行時に決定する(動的束縛)ということです。つまり、上書きされたメソッドが存在する場合、実行中のオブジェクトによって呼び出されるメソッドが変わります。

たとえば、前述の例で全従業員クラス中に賞与(ボーナス)を計算するメソッドがあったとして、そのメソッド中で給料を求める処理が必要だった場合、定義済みオブジェクト一意名SELFを使用して以下のとおり記述することができます。



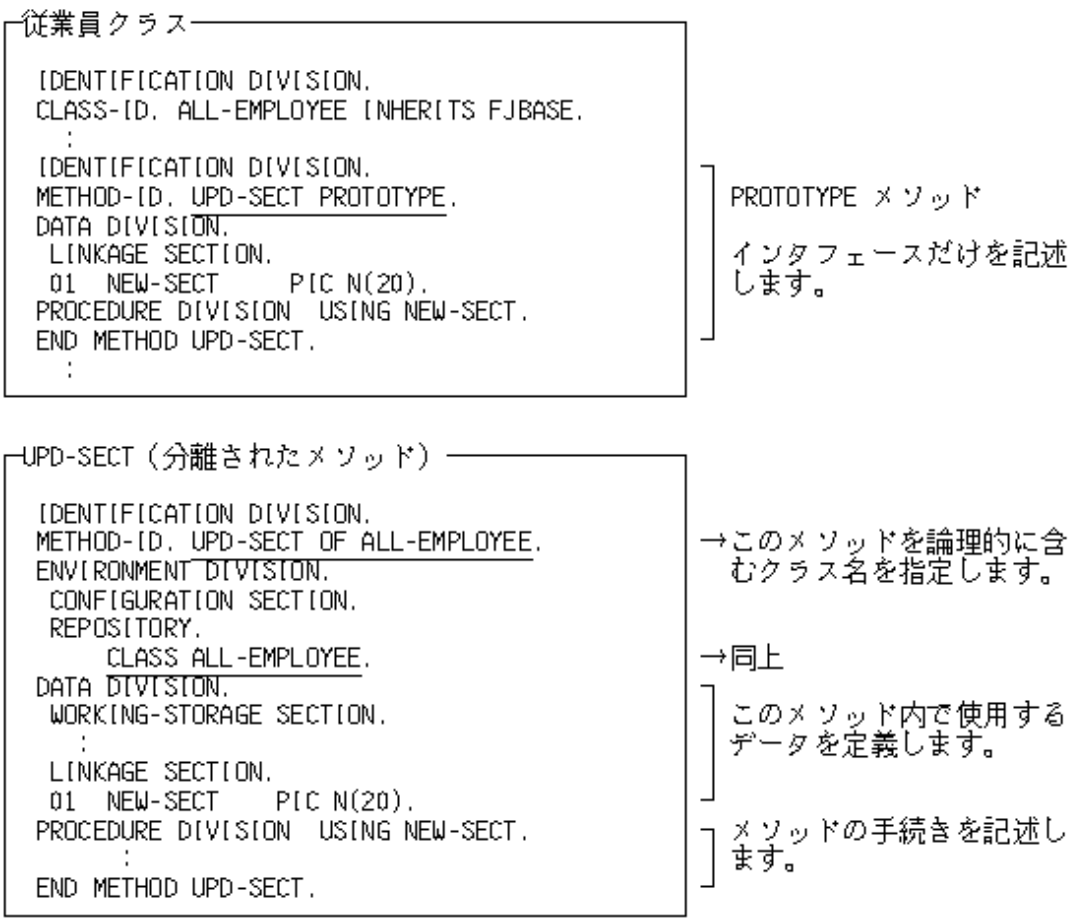
従業員クラスのオブジェクト参照によって賞与計算メソッド([2])が呼び出された場合、[3]のINVOKE文によって[1]の給料計算メソッドが呼び出されます。管理職クラスのオブジェクト参照によって賞与計算メソッド([5]暗黙定義メソッド)が呼び出された場合、[6]のINVOKE文によって[4]の給料計算メソッド(上書きメソッド)が呼び出されます。つまり、それぞれのクラスに適した給料計算ができることになります。

これも多態の1つの形態です。

16.1.7 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。

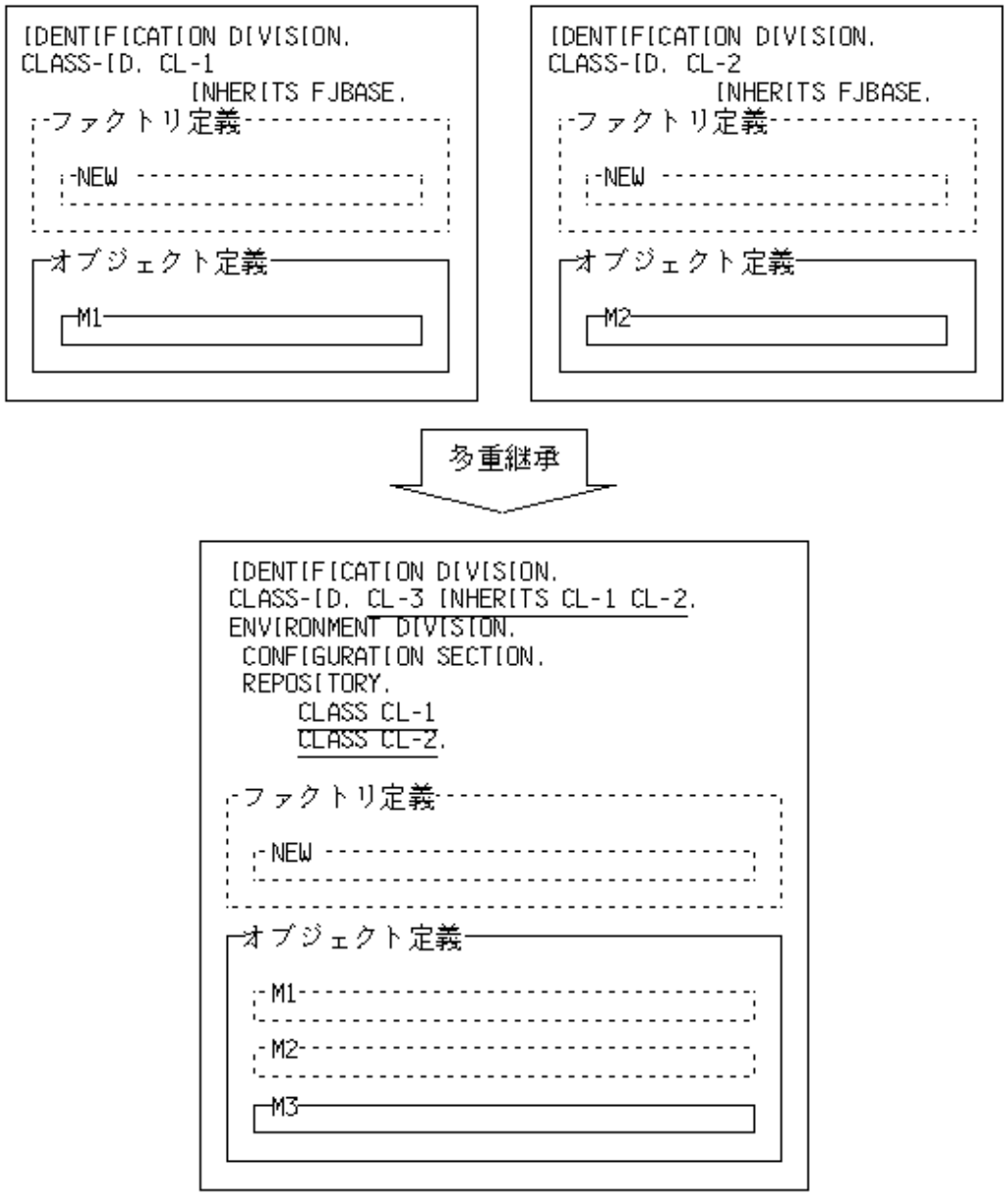


このようにメソッド定義を別翻訳単位にすることによって、1つのクラス定義を複数人で開発できるなどのメリットがあります。

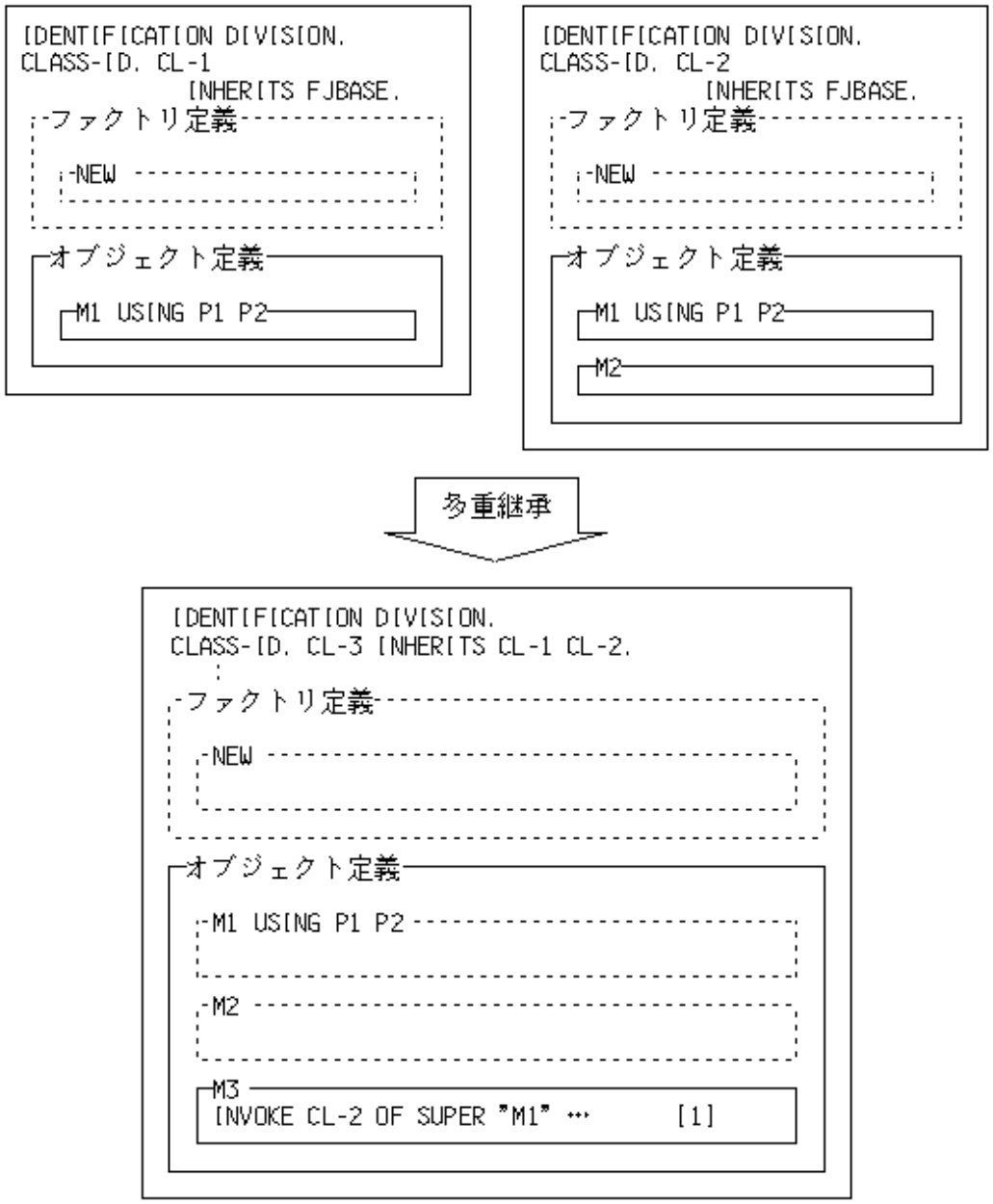
なお、分離されたメソッドの翻訳時に、そのメソッドを論理的に含むクラスのリポジトリファイルを入力する必要があるため、メソッドを翻訳するためには、先にクラス定義を翻訳しておく必要があります。詳細については、“16.2 オブジェクト指向プログラミングの開発”を参照してください。

16.1.8 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)。



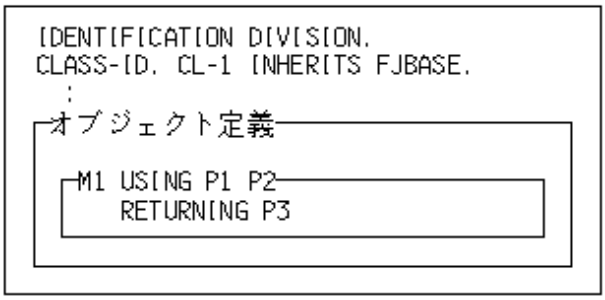
このように複数の親クラスで同名のメソッドが定義されていた場合、利用者が明にこのメソッドを上書きしない限り、引き継がれます。引き継がれるメソッドは、INHERITS句に指定されたクラス名の並びを左から検索して、最初にそのメソッドが見つかるクラスのメソッドになります。

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。

16.1.9 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。

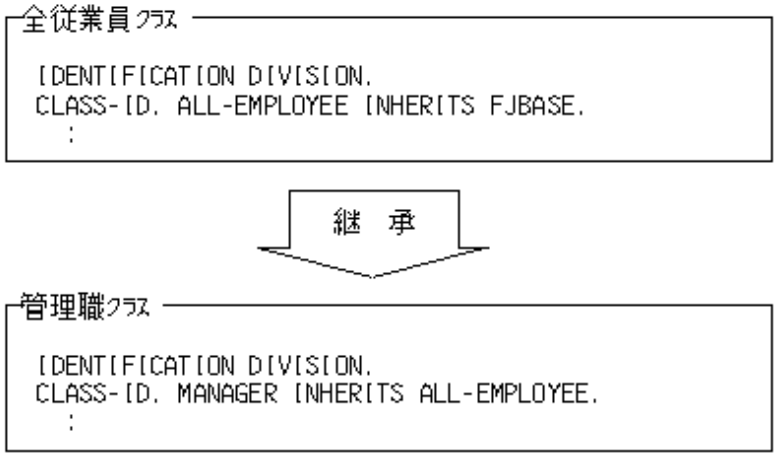
```
   :
 WORKING-STORAGE SECTION.
 01  OBJ-1 USAGE OBJECT REFERENCE CL-1.
PROCEDURE DIVISION.
      :
    INVOKE OBJ-1 "M1" USING P1 P2
                     RETURNING P3.
    IF P3 = 0 THEN …
      :
```

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 …
      :
```

16.1.10 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.                    ... [1]
    SET OBJ-2 TO OBJ-1.    ⇦エラー         ... [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うのです。

このような場合、オブジェクト指定子を利用することによって代入可能になります。

従業員管理プログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  OBJ-1   USAGE OBJECT REFERENCE ALL-EMPLOYEE.
 01  OBJ-2   USAGE OBJECT REFERENCE MANAGER.
PROCEDURE DIVISION.
      :
    INVOKE MANAGER "NEW" RETURNING OBJ-2.
      :
    SET OBJ-1 TO OBJ-2.
    SET OBJ-2 TO OBJ-1 AS MANAGER.
      :
```

上図のように記述すると、OBJ-1は、USAGE OBJECT REFERENCE句にMANAGERが指定されたとみなして適合チェックが行われるため、代入可能となります。

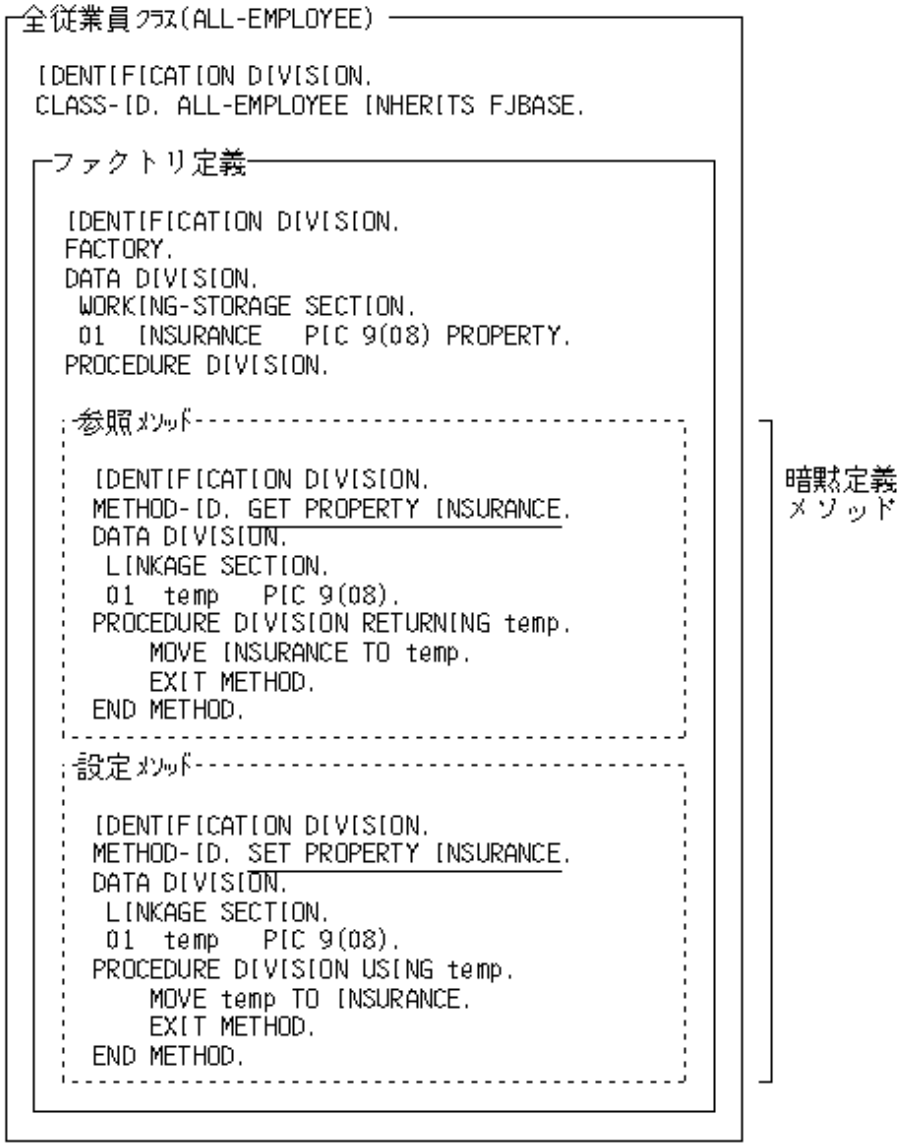
このように、オブジェクト指定子を用いた場合、翻訳時は指定されたクラス名で適合チェックが行われますが、実行時は実際に格納されているオブジェクト参照によって適合チェックが行われます。つまり、適合に違反している場合は、実行時にチェックアウトされます。

16.1.11 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。

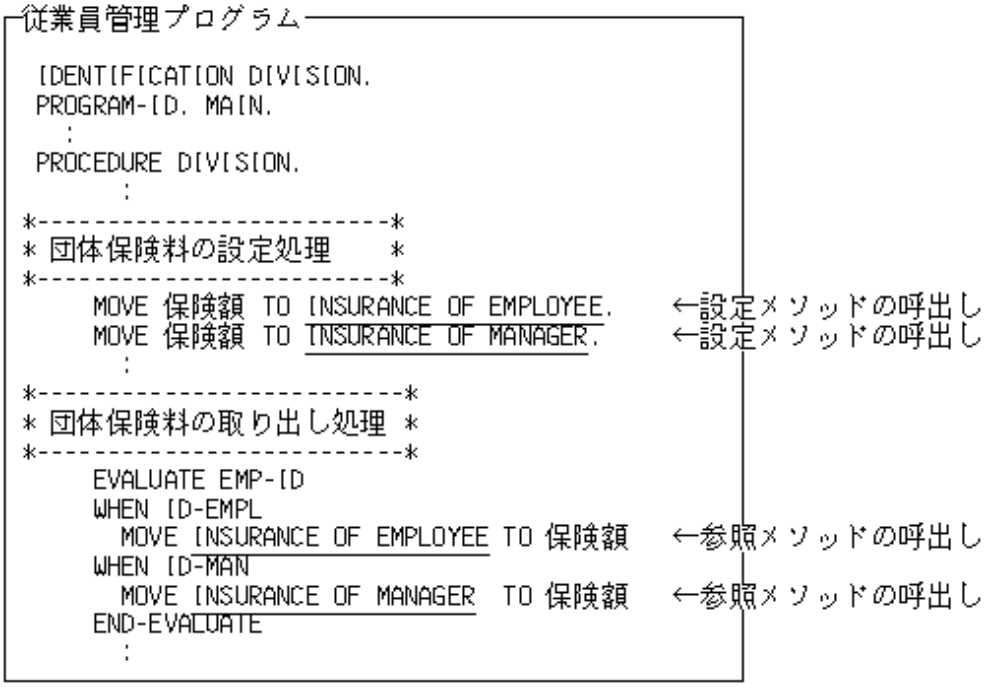
これは、PROPERTY句の指定によってデータを設定、参照するメソッドを自動生成することにより実現しています。

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。



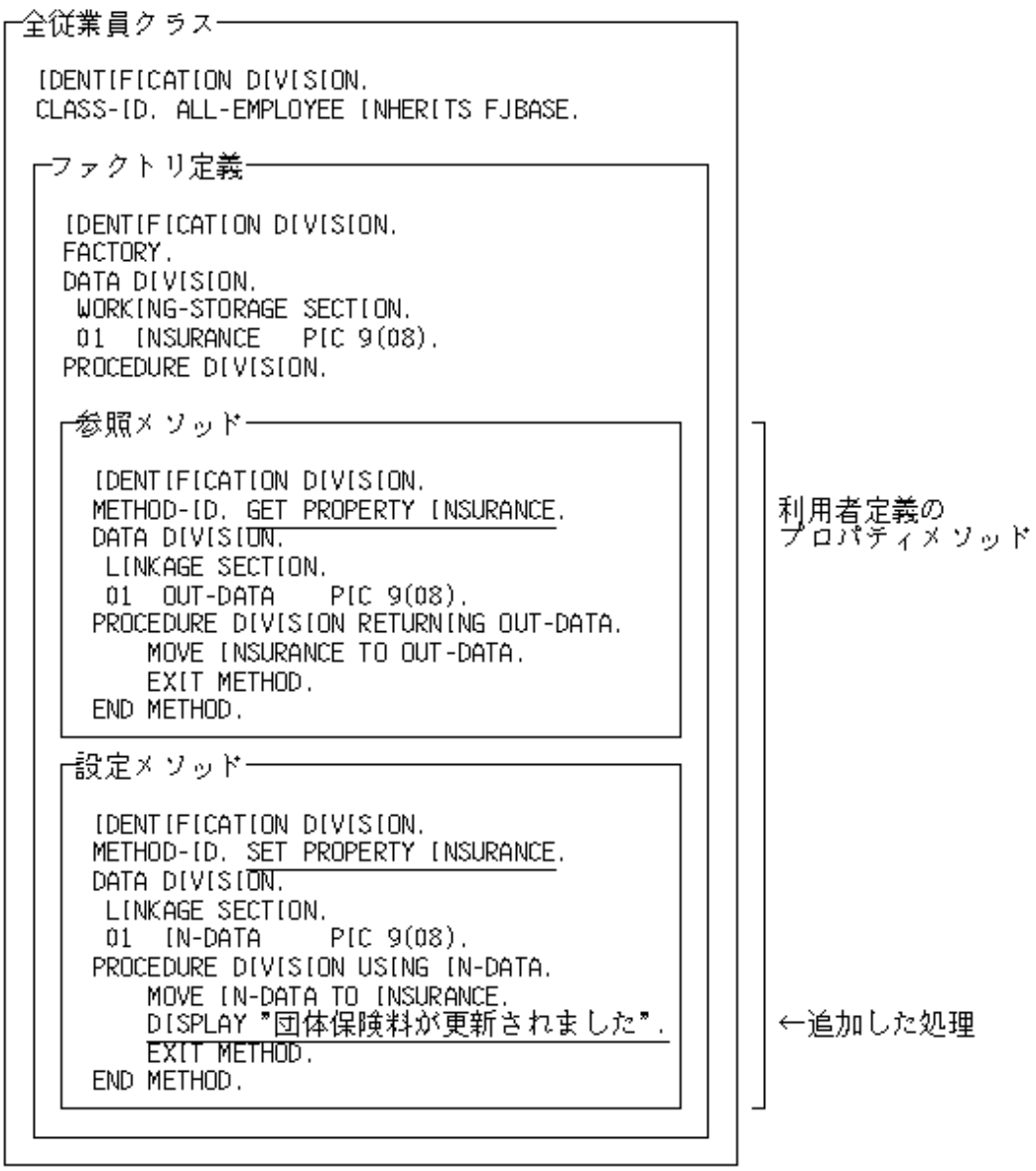
上図のようにPROPERTY句によって暗黙定義されるメソッドをプロパティメソッドと呼びます。

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。

16.1.12 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つきたことを自動的に判断して行います。[参照]“16.1.2.2 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、“16.1.3.3 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.      → OBJECT INSTANCE IS GENERATED を表示
    INVOKE OBJREF "XXX".                           → XXX IS INVOKED を表示
    SET OBJREF TO NULL.                            → OBJECT INSTANCE IS TERMINATEDを表示
END PROGRAM SAMPLE.
```

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

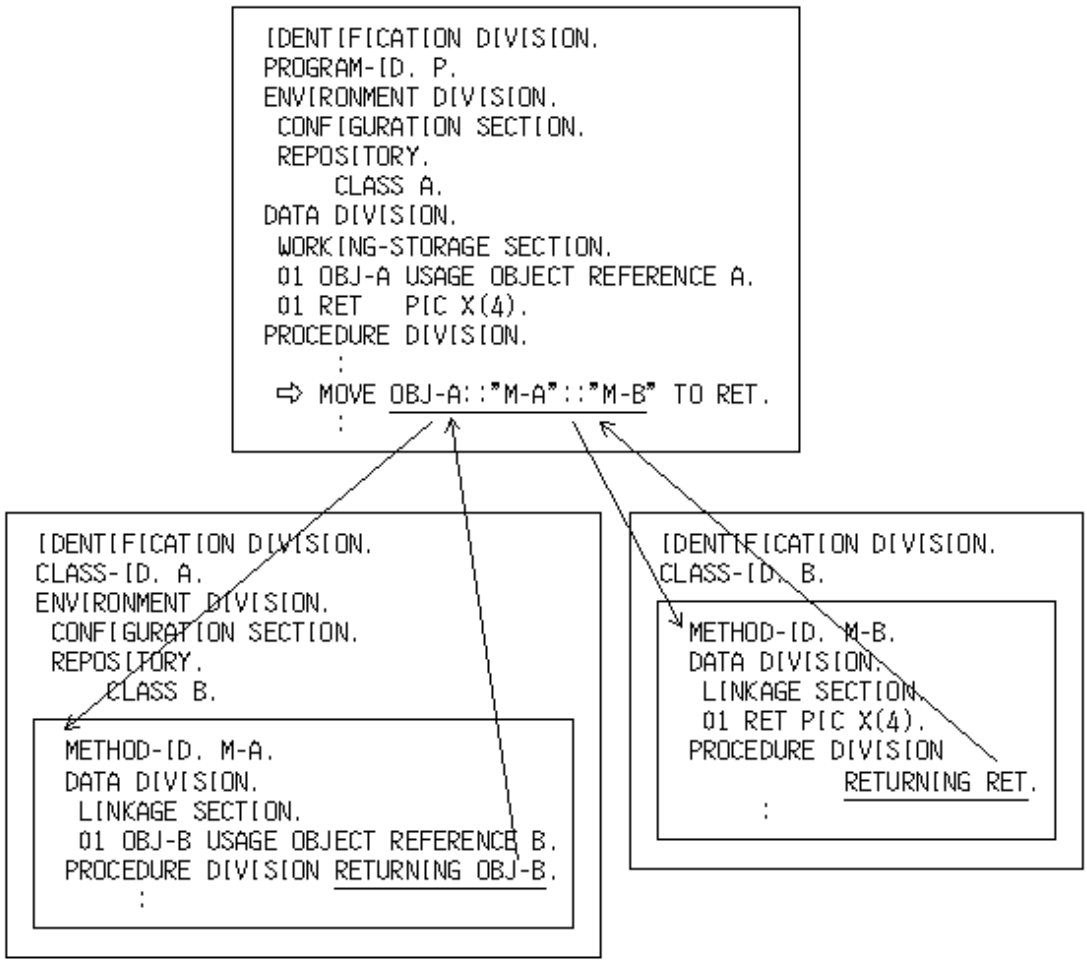
```
  IDENTIFICATION DIVISION.
  METHOD-ID. XXX.
  DATA DIVISION.
  PROCEDURE DIVISION.
    DISPLAY "XXX IS INVOKED".
  END METHOD XXX.
```

```
 END OBJECT.
```

16.1.13 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".            ┐生成した一時域を利用して
     MOVE temp ::"M-B" TO RET.            ┘入れ子を展開
       :
```

このとき、内部的に生成される一時域(上図temp)は、メソッドM-Aの復帰値と同じ属性がとられるため、クラスBのオブジェクト参照項目として定義されます。つまり、内部的に生成される一時域(暗黙に定義されたデータ項目)によってクラスBが参照されることになります。このようなクラスを間接参照クラスと呼び、明示的に参照されるクラスと同様、リポジトリ段落で宣言しなければなりません。つまり、プログラムPのリポジトリ段落にはクラスBの宣言が必要になります。

以上、行内呼出しの入れ子の場合を例に説明しましたが、このほかにも、

- オブジェクトプロパティの入れ子
- 復帰値に別のクラス(適合関係が成立するクラス)のオブジェクト参照項目が指定されたメソッドを呼び出す場合

などに間接参照クラスの宣言が必要となることがあります。行内呼出し、オブジェクトプロパティを含めて、復帰値がオブジェクト参照項目のメソッドを呼び出す場合には意識してコーディングしてください。

なお、間接参照クラスをリポジトリ段落で宣言しないで翻訳した場合、翻訳時にエラーメッセージが出力されるので、メッセージに従ってソースを修正してください。

16.1.14 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。

16.1.14.1 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が、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。

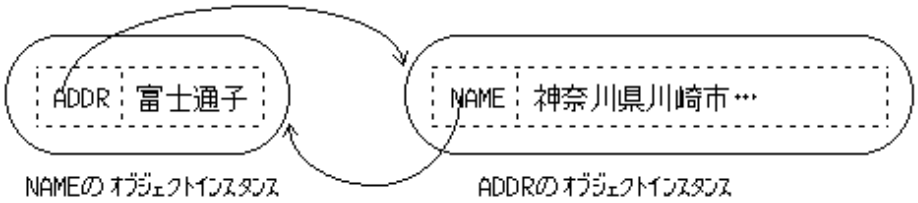
名前クラス

```
IDENTIFICATION DIVISION.
CLASS-ID. NAME INHERITS FJBASE.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     CLASS FJBASE
     CLASS ADDR.
OBJECT.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 OBJ-ADDR OBJECT REFERENCE ADDR.
 01 D-NAME   PIC N(10).
PROCEDURE DIVISION.
  :
```

住所クラス

```
IDENTIFICATION DIVISION.
CLASS-ID. ADDR INHERITS FJBASE.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     CLASS FJBASE
     CLASS NAME.
OBJECT.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 OBJ-NAME OBJECT REFERENCE NAME.
 01 D-ADDR   PIC N(40).
PROCEDURE DIVISION.
  :
```

図16.3 実行時のオブジェクトインスタンスイメージ



このようにオブジェクトインスタンスを相互結合しておくことによって、名前から住所を求めることも、逆に住所から名前を求めることも可能になります。

他クラスとの間接相互参照

他クラスのオブジェクトインスタンスと密接に関係するような構成で、間接的に相互参照関係が成立する場合があります。以下、

- 名前クラス(NAME)が住所クラスのオブジェクトインスタンスを、
- 住所クラス(ADDR)が所属クラスのオブジェクトインスタンスを、
- 所属クラス(SECT)が所属長の名前クラスのオブジェクトインスタンスを

保持する場合を例に説明します。

名前クラス

```
IDENTIFICATION DIVISION.
CLASS-ID. NAME INHERITS FJBASE.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     CLASS FJBASE
     CLASS ADDR.
OBJECT.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 OBJ-ADDR OBJECT REFERENCE ADDR.
 01 D-NAME   PIC N(10).
PROCEDURE DIVISION.
   :
```

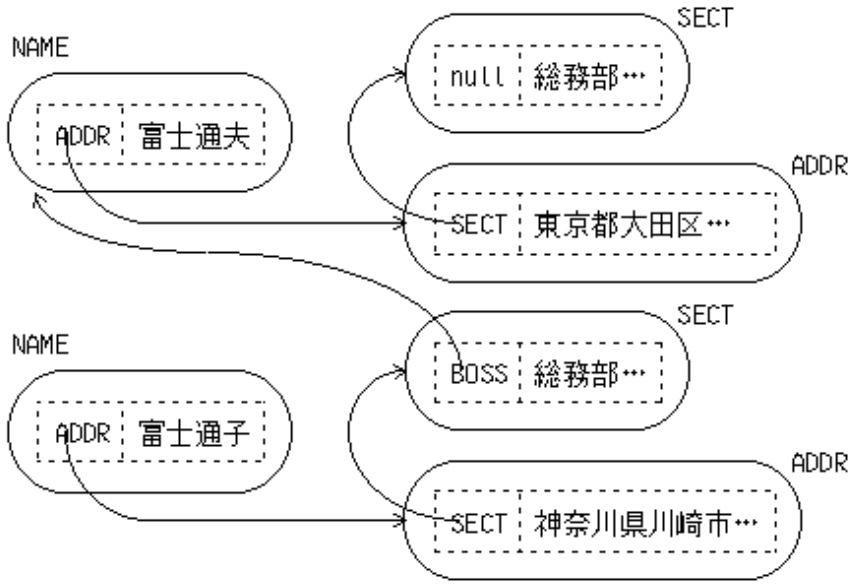
住所クラス

```
IDENTIFICATION DIVISION.
CLASS-ID. ADDR INHERITS FJBASE.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     CLASS FJBASE
     CLASS SECT.
OBJECT.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 OBJ-SECT OBJECT REFERENCE SECT.
 01 D-ADDR   PIC N(40).
PROCEDURE DIVISION.
   :
```

所属クラス

```
IDENTIFICATION DIVISION.
CLASS-ID. SECT INHERITS FJBASE.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     CLASS FJBASE
     CLASS NAME.
OBJECT.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 OBJ-BOSS OBJECT REFERENCE NAME.
 01 D-SECT   PIC N(20).
PROCEDURE DIVISION.
   :
```

図16.4 実行時のオブジェクトインスタンスイメージ



このように、オブジェクトインスタンスが複雑に結合し合うような場合、容易に間接相互参照関係が成立します。

16.1.14.2 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が、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。

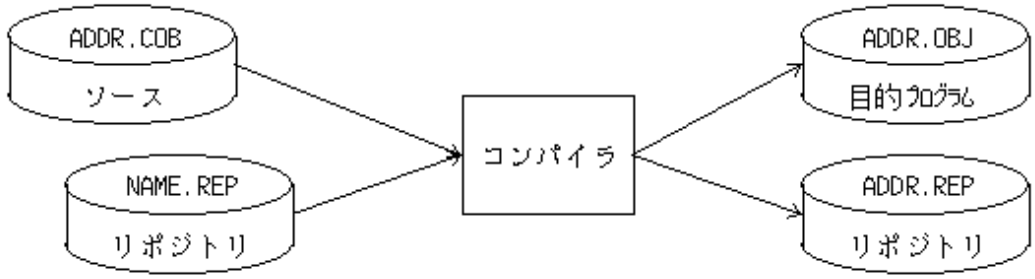
ステップ1 ： 名前クラスのリポジトリを生成

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。

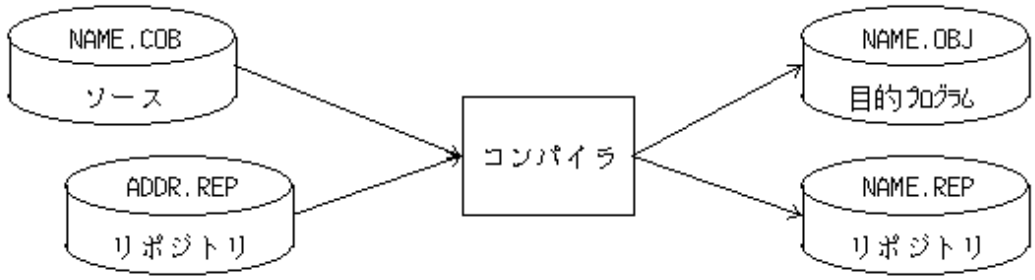
ステップ2 ： 生成したリポジトリを利用して、住所クラスを翻訳

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。

注意

CREATE(REP)オプションが指定された場合、コンパイラは手続き部の解析を行いません。このため、手続き部にエラーが存在してもメッセージは出力されません。後の目的プログラム生成時にエラーチェックされるため、その際に必要に応じて修正してください。

16.1.14.3 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が、その際、名前クラスのインポートライブラリが必要です。翻訳時と同様に、いわゆる、「タマゴが先か、ニワトリが先か」の状態に陥ってしまうわけです。

このような状態を回避するために、まず、いずれかのクラスのインポートライブラリを作成します。インポートライブラリは、リンク時に作成する方法とモジュール定義ファイルより作成する方法があるため、後者を利用します。

ステップ1 ： 名前クラスのモジュール定義ファイルを作成

エディタを利用して、名前クラスのモジュール定義ファイル(NAME.DEF)を作成します。モジュール定義ファイルは以下の形式です。

```
NAME.DEF
┌──────────────────────────────────┐
│ LIBRARY  "NAME"                  │ ←ライブラリ名を指定
│ EXPORTS   _NAME_FACTORY          │ ←エントリ名を指定
└──────────────────────────────────┘
```

LIBRARYにはライブラリ名を、EXPORTSにはエントリ名を指定します。

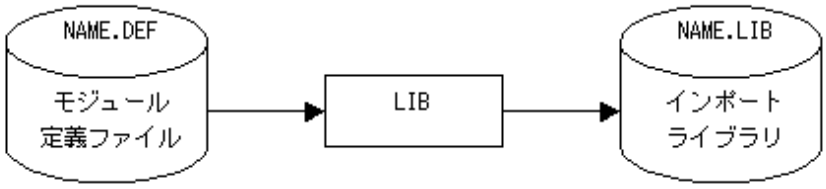
エントリ名の規約は、以下のとおりです。

- 先頭の1バイトは半角のアンダースコア ： 固定文字
- 続けてクラス名 ： 可変文字
- 続けて半角のアンダースコア ： 固定文字
- 最後にFACTORY ： 固定文字

ステップ2 ： 名前クラスのインポートライブラリを作成

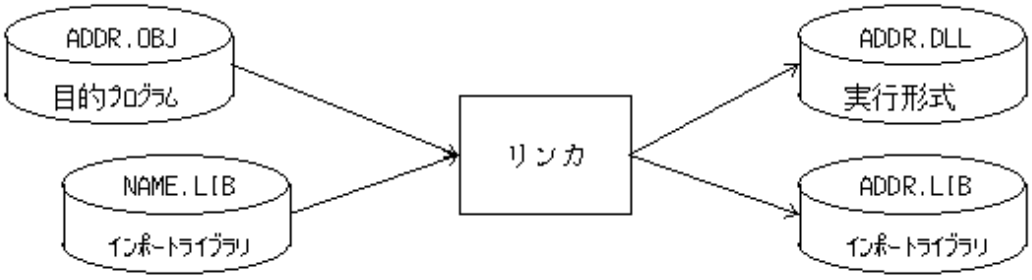
ステップ1で作成したモジュール定義ファイルを利用して、名前クラスのインポートライブラリを作成します。

```
LIB /OUT:NAME.LIB /DEF:NAME.DEF /MACHINE:x64
```



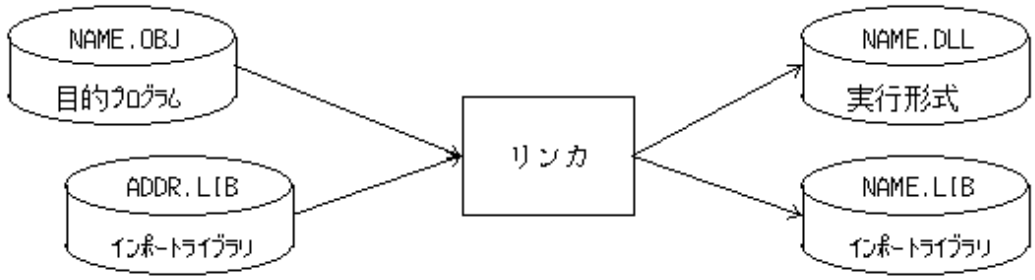
ステップ3 ： 住所クラスの実行形式を作成

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。

16.1.14.4 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。

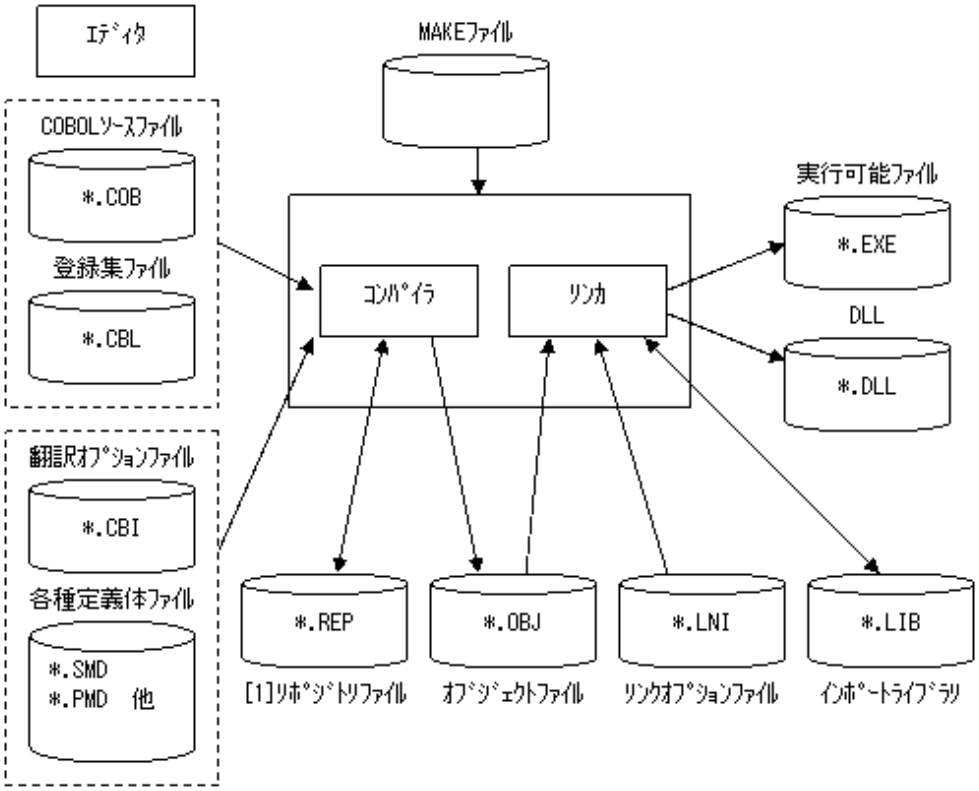
16.2 オブジェクト指向プログラミングの開発

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。

16.2.1 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。



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。

16.2.2 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。
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、NMAKEコマンドを使用することをおすすめします。NMAKEコマンドについては、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。

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。

16.2.3 クラスの設計

クラスを新しく設計する場合には、オブジェクト指向プログラムの特徴である“部品化”のメリットを十分に引き出すために以下の注意が必要です。

- 機能に汎用性を持たせる。
- クラス名、メソッド名から機能が類推できる。

また、クラス名、メソッド名の決定にあたっては、以下の点に注意が必要です。

- アプリケーションの処理過程で、利用あるいは継承するクラス名は、そのアプリケーションの中で一意となる必要があります。
- 先頭がアンダーバー“_”で始まるメソッド名は、利用者が作成するメソッド名として使用できません。

16.2.4 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フェース

クラス名とその機能

プログラミング作業は、何らかの目的を実現するために行います。

そのため、再利用するクラスもその目的のために使用できるものでなければなりません。

例

ある会社の従業員管理プログラムを作成するために従業員オブジェクトを作成するような場合、目的に対して無関係なクラス(たとえば、農場で牛の健康管理を行っているようなクラス)の流用は意味がありません。

このように、利用者は目的のプログラムを作成するために再利用するクラスとして、どのような機能を持った、どのような名前のクラスがあるかという情報を事前に入手していなければなりません。

クラスの持っているメソッド名およびその機能

オブジェクト指向プログラミングでは、クラスからオブジェクトを生成することも、そのオブジェクトを動作させることにも、メソッドの呼出しを行う必要があります。

既存のクラスを再利用したい場合、目的に合った処理を行うメソッドがない、またはあっても名前がわからなければ、利用者にとってはクラスが利用できないのと同じことになります。

例

従業員オブジェクトを使用して従業員の平均の月給を算出するようなプログラムを作成する場合には、従業員オブジェクトの中に従業員の月給を知るメソッドがあり、かつ、利用者はそのメソッドの名前がわからなければ、使用することはできません。

このように、利用者は、自分が使用するクラスに定義されているメソッドの名前および機能という情報も入手しておく必要があります。

各メソッドのインタフェース

自分の作成したいプログラムの目的に合ったクラスおよびメソッドが見つかっても、利用者はそのメソッドに対して、どのような値をどのような形式で渡すと、どのような値がどのような形式で返ってくるのかがわからないと、思った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のオブジェクトを検索する場合、検索メソッドに対して何の情報を渡せばよいのか(たとえば、名前や従業員番号など)、また、どのような情報が返ってくるのかといったインタフェース情報がわからなければ、そのメソッドを使用することはできません。

このように、利用者は、自分が使用するメソッドのインタフェースも理解している必要があります。

16.2.4.1 プログラム構造

16.2.4.1.1 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関係を下図に示します。

図16.5 翻訳単位とリンク単位が一致する場合

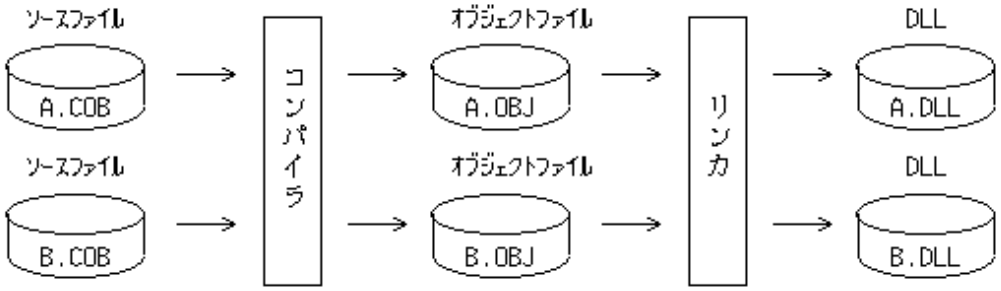
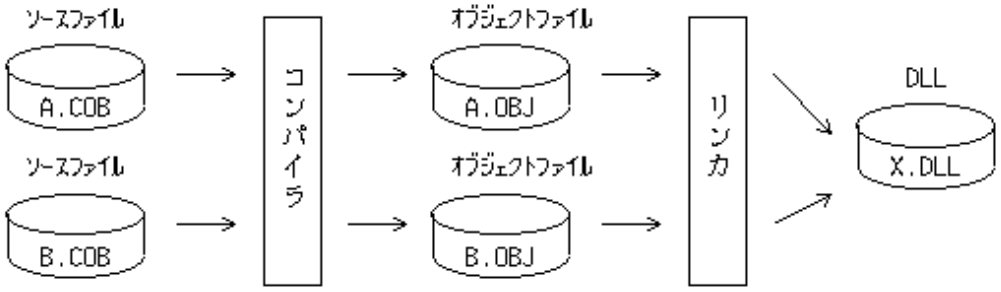


図16.6 翻訳単位とリンク単位が一致しない場合



16.2.4.1.2 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、“16.3.2 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すみます。

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)。

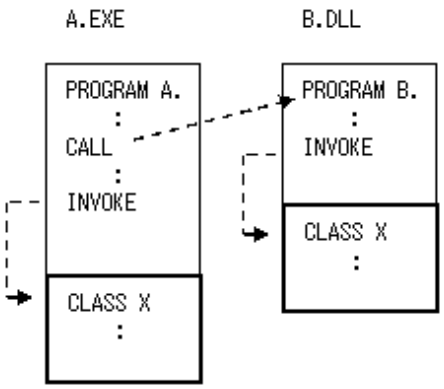
そのため、汎用的でリンク関係が多彩な、以下の場合に適しています。

- 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

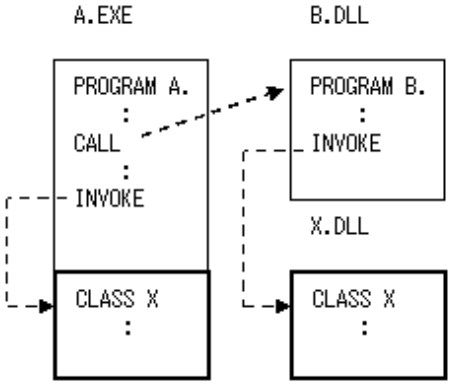
注意

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)

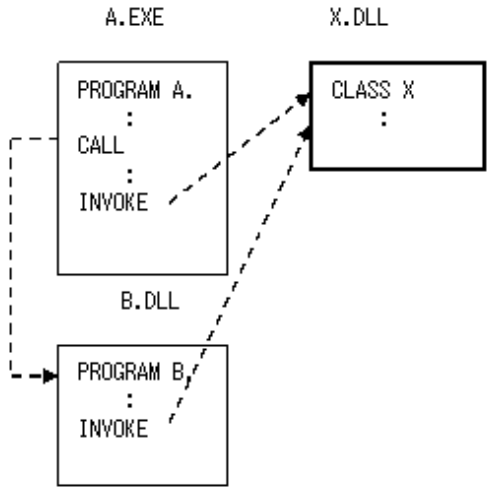
1. 静的構造により、同一クラスのオブジェクトが2つ存在する場合



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

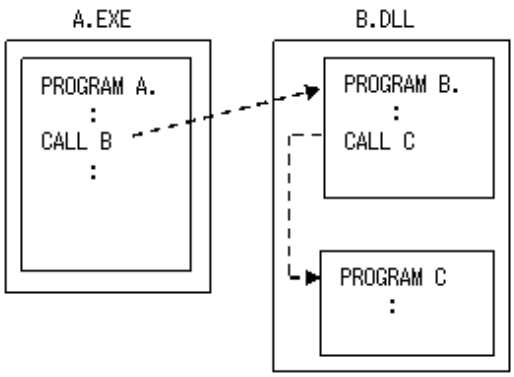


参考

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。



16.2.4.2 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。

16.2.4.2.1 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、クラスの情報を格納したファイルです。リポジトリファイルは、翻訳時に再利用するクラスの情報をコンパイラに通知するために使用します。

リポジトリファイルのファイル名は、“クラス外部名.REP”になります。

図16.7 リポジトリファイルの出力

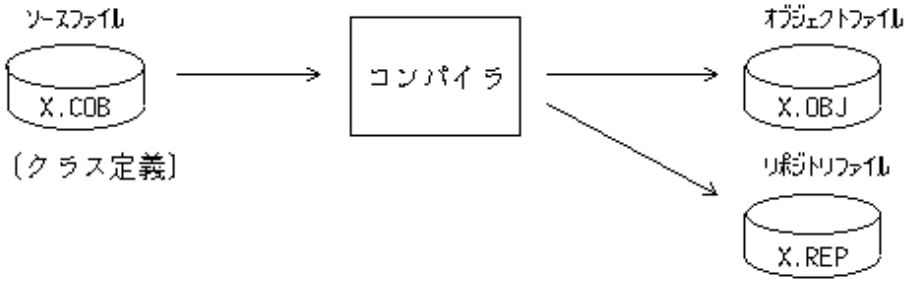
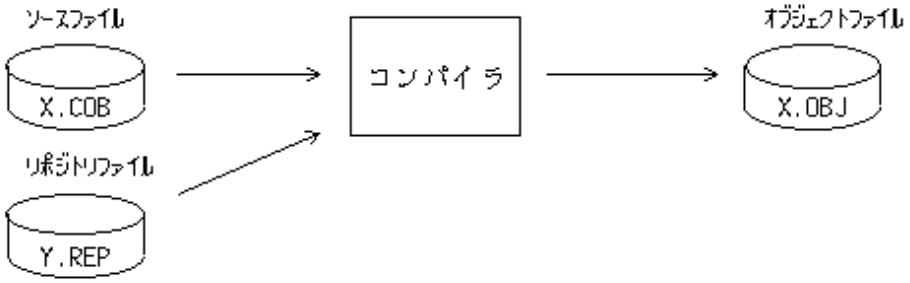


図16.8 リポジトリファイルの入力



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ければならないリポジトリファイル(クラス名)は以下のとおりです。

- 継承を利用する場合の直接の親クラス
- オブジェクト参照データ項目で指定したクラス
- 分離されたメソッドで、自メソッドのPROTOTYPE宣言が含まれるクラス

継承を利用する場合の直接の親クラス

例題15の“MEMBER.COB”を例にとってみます。

```
      :
CLASS-ID. Member-class INHERITS AllMember-class.
ENVIRONMENT DIVISION.                  [a]
 CONFIGURATION SECTION.
  REPOSITORY.
    CLASS AllMember-class.
     :          [b]
```

このプログラムでは、[a]が継承する直接の親クラス名になります。

このプログラムは、AllMember-classクラスを継承しているので、翻訳時にリポジトリファイルAllMember-class.REPが必要になります。

そのため、そのクラス名はリポジトリ段落に記述されていなければなりません(プログラム中の[b])。

このクラス名を元に、コンパイラは対応するリポジトリファイルを検索します。

オブジェクト参照変数で指定したクラス

“ALLMEM.COB”を例にとってみます。

```
      :
CLASS-ID. AllMember-class INHERITS FJBASE.
      :
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
  REPOSITORY.
      :
    CLASS Address-class.
     :          [b]
  OBJECT.
  DATA DIVISION.
   WORKING-STORAGE SECTION.
     :
   01  住所参照       OBJECT REFERENCE Address-class …
     :                                    [a]
```

このプログラムでは、[a]がオブジェクト参照変数で指定されているクラス名になります。

このプログラムは、Address-classクラスを参照しているので、翻訳時にリポジトリファイルAddress-class.REPが必要になります。

この場合にも、コンパイラがリポジトリファイルを検索するために、クラス名をリポジトリ段落に記述しなければなりません(プログラム中の[b])。

分離されたメソッドを作成する場合のメソッド定義されているクラス

“SALA_MEM.COB”を例にとってみます。

```
METHOD-ID. Salary-method  OF  Member-class.
ENVIRONMENT DIVISION.              [a]
 CONFIGURATION SECTION.
  REPOSITORY.
    CLASS Member-class.
     :        [b]
```

このプログラムでは、[a]がメソッドの所属しているクラス名になります。

このプログラムは、Member-classクラスの情報を参照するので、翻訳時にMember-class.REPというリポジトリファイルが必要になります。

この場合も、コンパイラが必要なリポジトリファイルを検索するために、クラス名をリポジトリ段落に記述する必要があります(プログラム中の[b])。

翻訳の手順

COBOLソースファイルを翻訳するときに、コンパイラは参照する必要があるリポジトリファイル(リポジトリ段落に記述されたクラスのリポジトリファイル)を検索します。このとき、必要なリポジトリファイルが存在しないと翻訳エラーとなります。

また、クラス定義を再翻訳すると、リポジトリファイルも更新されます。その場合、更新されたリポジトリファイルを翻訳時に入力しているソースファイルも、再度翻訳しなければなりません。

このように、リポジトリファイルの入力の関係により、翻訳順序に制約が生じます。

例題15の“MEMBER.COB”は、以下のように記述されています。

```
      :
CLASS-ID. Member-class INHERITS AllMember-class.
      :
  REPOSITORY.
    CLASS AllMember-class.
      :
```

このプログラムの内容から、翻訳時にはAllMember-classクラスのリポジトリファイルが必要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。

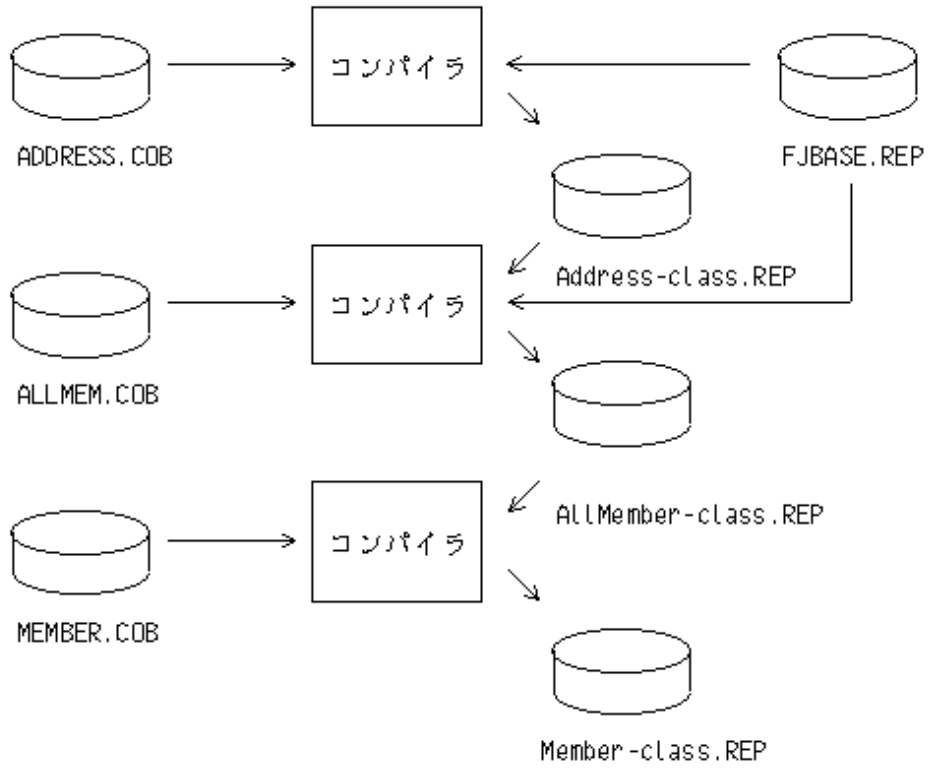
“ADDRESS.COB”のソースプログラムは、以下のように記述されています。

```
      :
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、資源の関係は以下のようになります。

図16.9 翻訳順序



“図16.9 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16.9 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.1.9 -dr(リポジトリファイルの入出力先フォルダの指定)”

-dr オペランド	フォルダ名
あり	-dr オペランド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16.9 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.1.17 -R(リポジトリファイルの入力先フォルダの指定)”、“J.1.9 -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

COBOLコマンドで翻訳を行う例

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

```
COBOL -c -R C:¥COBOL¥SAMPLES¥SAMPLE19 -dr C:¥COBOL¥SAMPLES¥SAMPLE19
```

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“16.1.14.2 相互参照クラスの翻訳”を参照してください。

16.2.4.3 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。

16.2.4.3.1 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16.1 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。

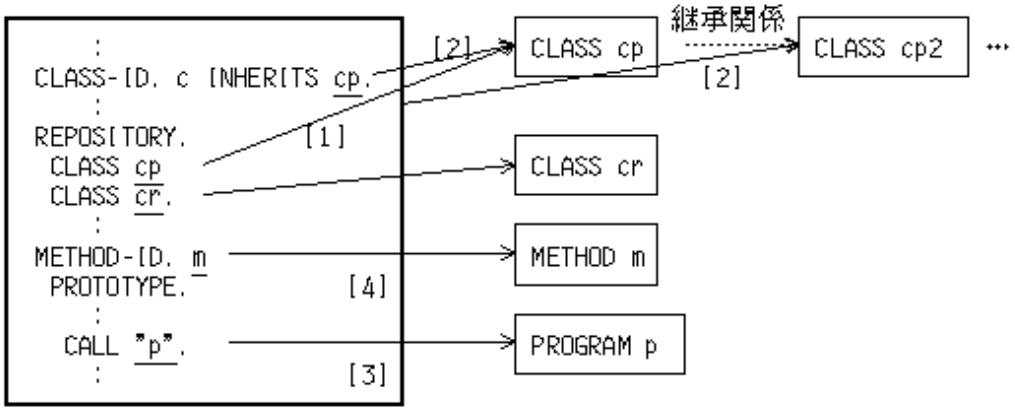
各定義間のリンク関係を下図に示します。

なお、図中の実線矢印がリンクを表します。

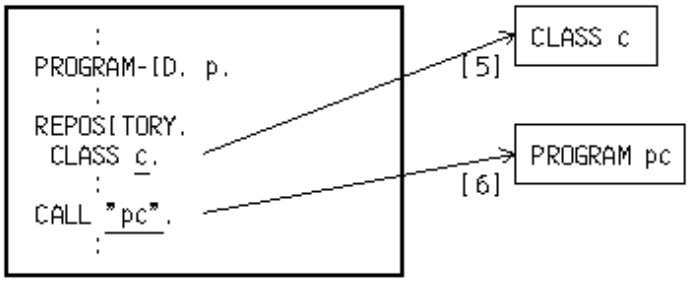
実線矢印で示されたリンク関係を、動的リンク構造で解決する場合にはインポートライブラリが必要になります。

各定義間のリンク

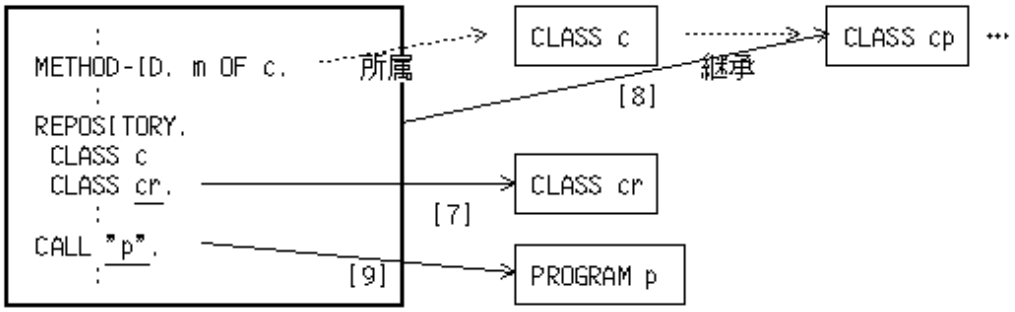
クラス定義のリンク



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

```
      :
CLASS-ID. Member-class INHERITS AllMember-class.
      :
  REPOSITORY.
    CLASS AllMember-class.
      :
```

・ ALLMEM.COB

```
      :
CLASS-ID. AllMember-class …
      :
  REPOSITORY.
      :
    CLASS Address-class.
      :
```

・ ADDRESS.COB

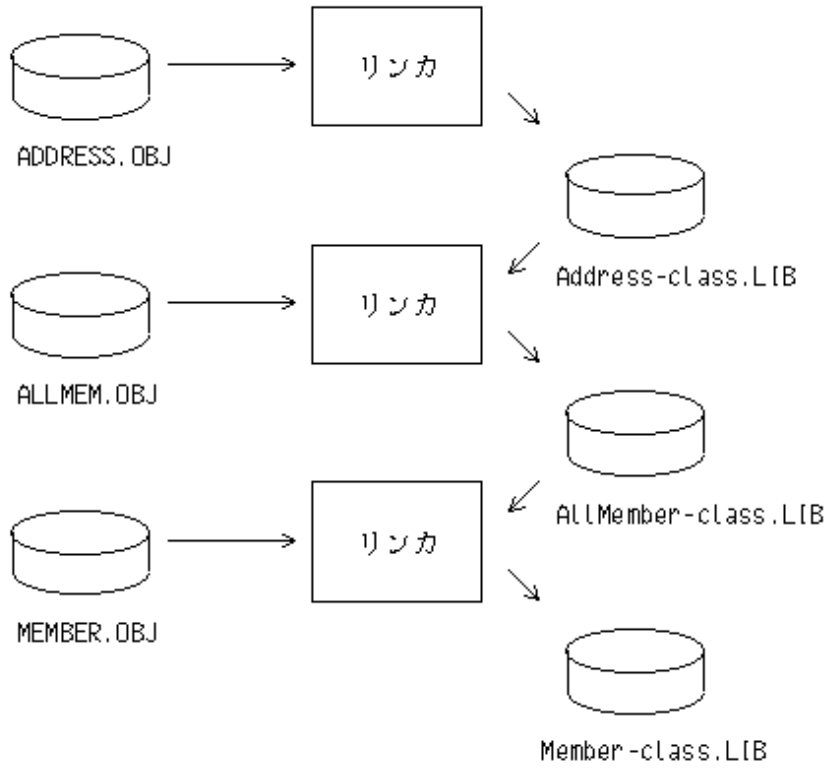
```
      :
CLASS-ID. Address-class …
      :
```

翻訳単位とリンク単位を同じにした場合、それぞれのリンク処理で必要なインポートライブラリおよび生成されるインポートライブラリは下表のようになります。

ソースファイル／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16.10 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16.10 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16.10 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    F4AGCIMP.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 F4AGCIMP.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 F4AGCIMP.LIB /NOENTRY /DLL /OUT : Member-class.DLL
```

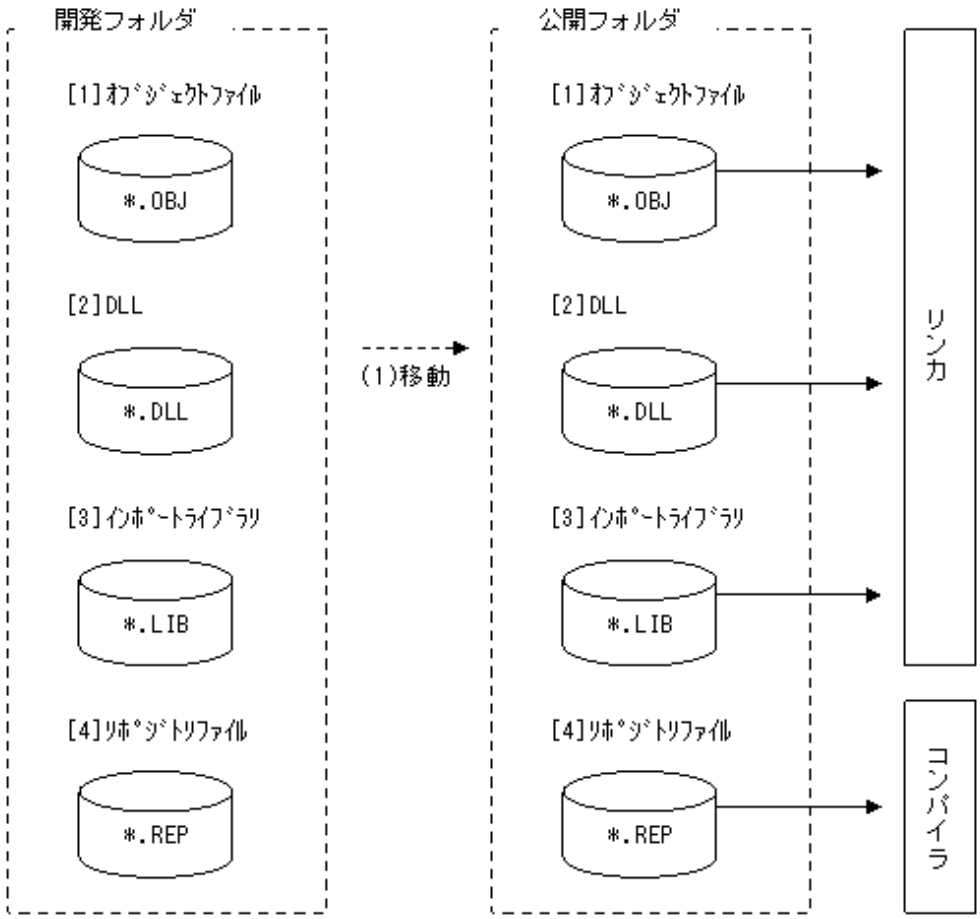
入力 ： MEMBER.OBJとAllMember－class.LIB
出力 ： Member-class.DLLとMember-class.LIB

16.2.5 クラスの公開

作成およびテストが完了したクラスは、新たな部品として、他プログラム開発の再利用の対象となります。その場合に、そのクラスは開発者以外からも利用可能でなければなりません。

作成したクラスのアクセス(再利用)を、開発者以外からも可能とすることを、「クラスの公開」と呼びます。

クラスを公開するための作業を、以下に説明します。



(1) 移動

クラスを再利用するために必要な資源を、開発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16.2 公開資源”のようになります。

表16.2 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

16.2.6 MAKEファイル

MAKEファイルとは、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のルールが記述してあるファイルのことです。

ここでは、MAKEファイルについて説明します。

MAKEファイルの用途

MAKEファイルは、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。
- 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、他の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。

例

MAKEファイルの例を以下に示します。

```
        :
COBOL_PATH=C:¥COBOL                                    … [1]
```

```
        :
SAMP1.OBJ : SAMP1.COB SAMP2.REP                                    … [2]
      $(COBOL_PATH)¥COBOL -M SAMP1.COB                             … [3]
        :
SAMP1.EXE : SAMP1.OBJ   … SAMP2.LIB                                … [4]
      $( COBOL_PATH)¥LINK  …                                       … [5]
      /OUT:SAMP1.EXE
      SAMP1.OBJ  SAMP2.LIB F4AGCIMP.LIB LIBCMT.LIB
        :
```

[1] コンパイラがインストールされているパス

[2] 翻訳を実行するための依存関係

[3] 翻訳コマンド

[4] リンクを実行するための依存関係

[5] リンクコマンド

16.3 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～

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

16.3.1 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。

16.3.1.1 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。

16.3.1.2 例外オブジェクト

例外オブジェクトは、エラー処理を1箇所でまとめて行いたいような場合に使用します。たとえば、誤ったデータが入力されたらメッセージを表示する処理を行うとします。

データエラーメッセージを表示する手続きを持つクラス(DATA-ERROR)を定義し、データに誤りがあった場合、これをオブジェクト化し、RAISE文またはRAISING指定のEXIT文にそのオブジェクトを指定します。

このとき指定したオブジェクトが例外オブジェクトとなり、「データに誤りが発生した」という例外条件になります。

例外オブジェクトが発生すると、例外オブジェクトのクラス名と継承関係を持つクラス名が宣言節部分のUSE文に記述された場合、その例外手続きが実行されます。上記の例でいうと、USE文にDATA-ERRORと記述するとその手続きが実行されます。例外手続きで例外オブジェクトを使用したい場合、EXCEPTION-OBJECTと記述することにより使用することができます。例外手続きが正常に終了した場合、例外条件が発生した直後の文に制御が移ります。

プログラムのどこで例外が発生しても、必ずこの例外手続きが実行されます。したがって、データ入力エラーに対する処理を1箇所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。

  ー 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。

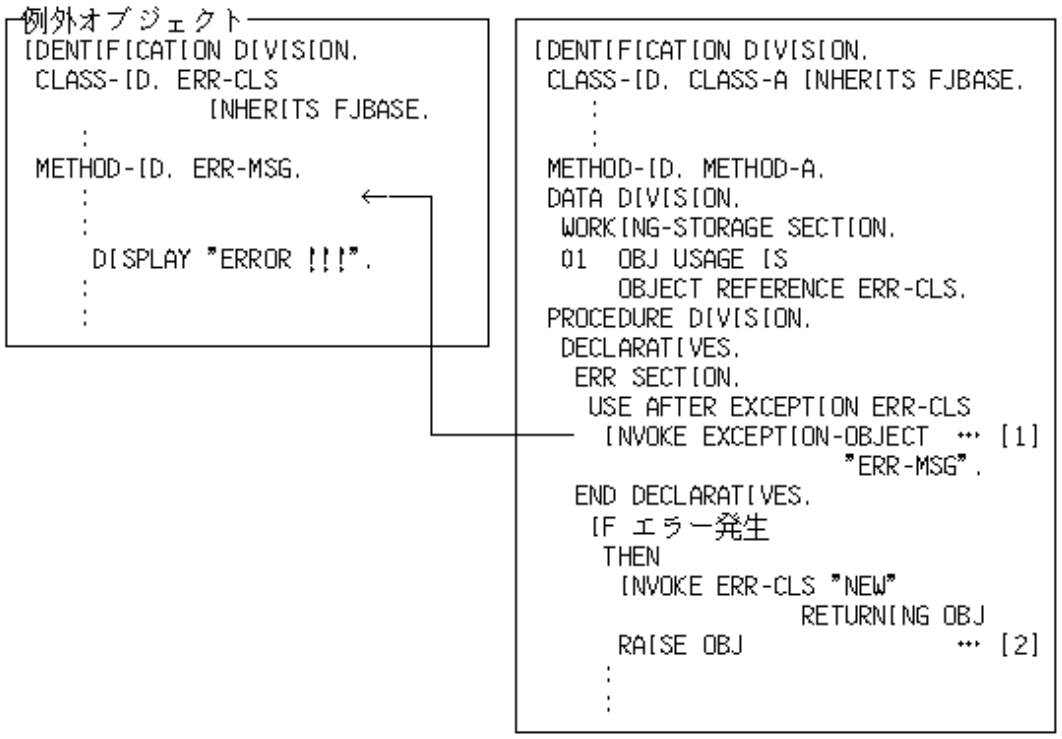
  － RAISE文
  例外条件を発生させた文の直後の文に制御が移ります。

  － 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。

16.3.1.3 RAISE文の動作

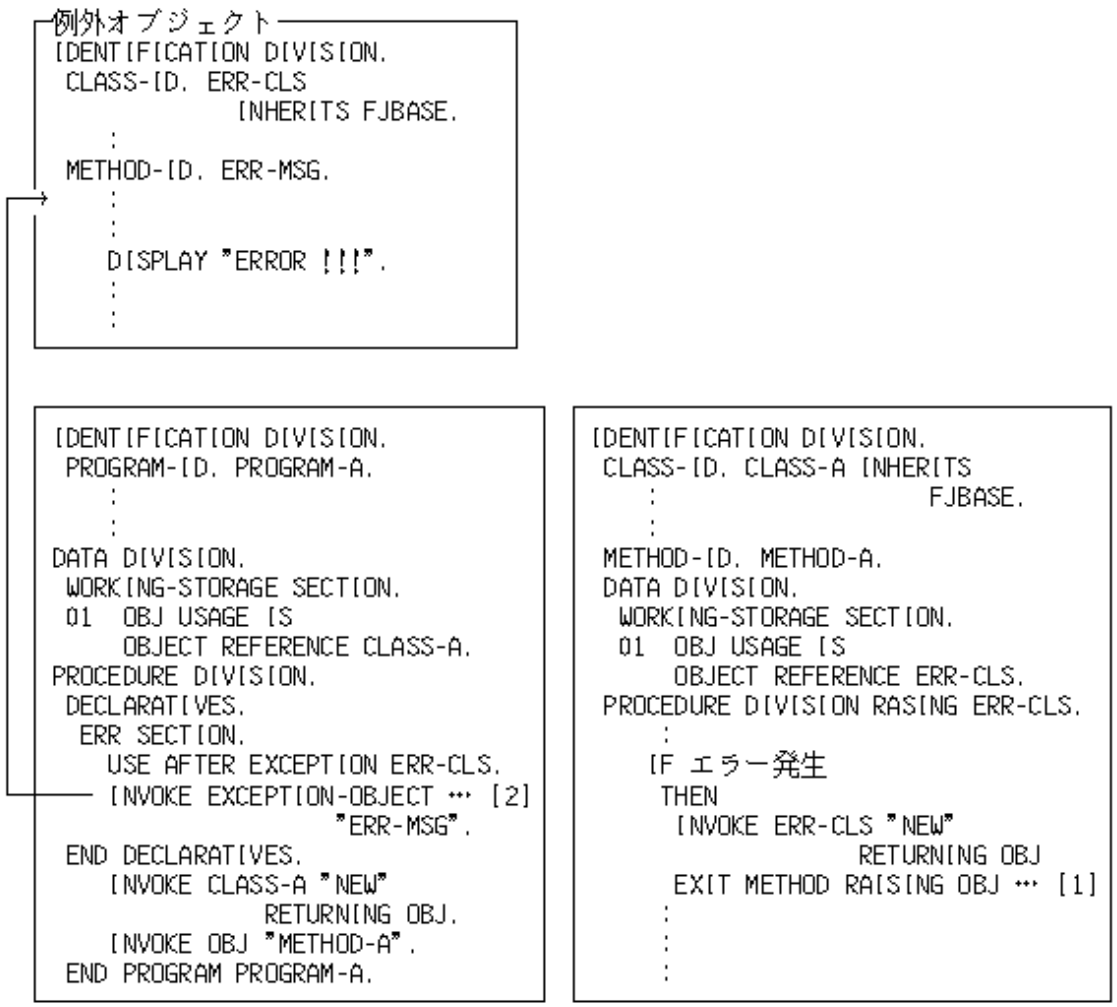
RAISE文は、手続きの途中で例外条件を発生させ、その手続きが属するプログラムまたはメソッドに記述された例外手続きを実行します。



[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。

16.3.1.4 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。

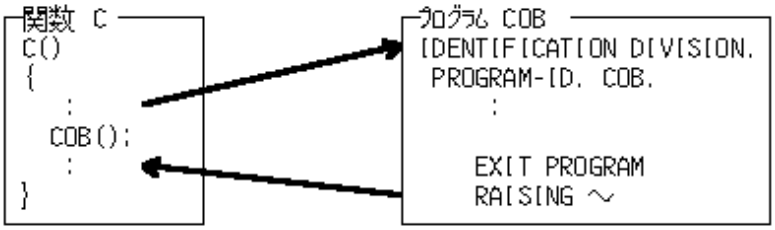
注意

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。

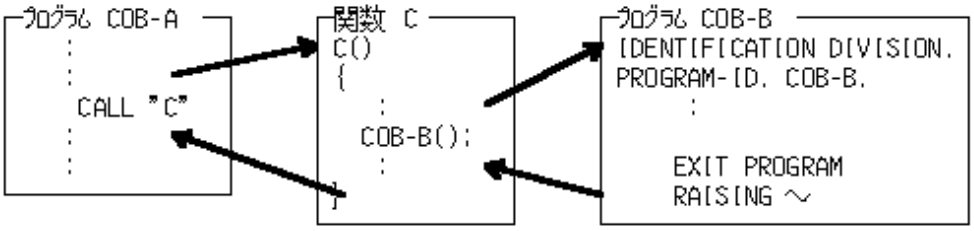
- 主プログラム
- 他言語プログラムから呼び出されたCOBOLプログラム(下図[1])

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。



16.3.2 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、“10.1.3 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、“10.1.3.3 注意事項”を必ずお読みください。

16.3.2.1 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。

16.3.2.2 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。

```
PROGRAM-[D. P1.
    :
PROCEDURE D[VIS[ON.
      :
    [NVOKE C2  "M1".   ... [a]
      :
    EX[T PROGRAM.
END PROGRAM P1.
```

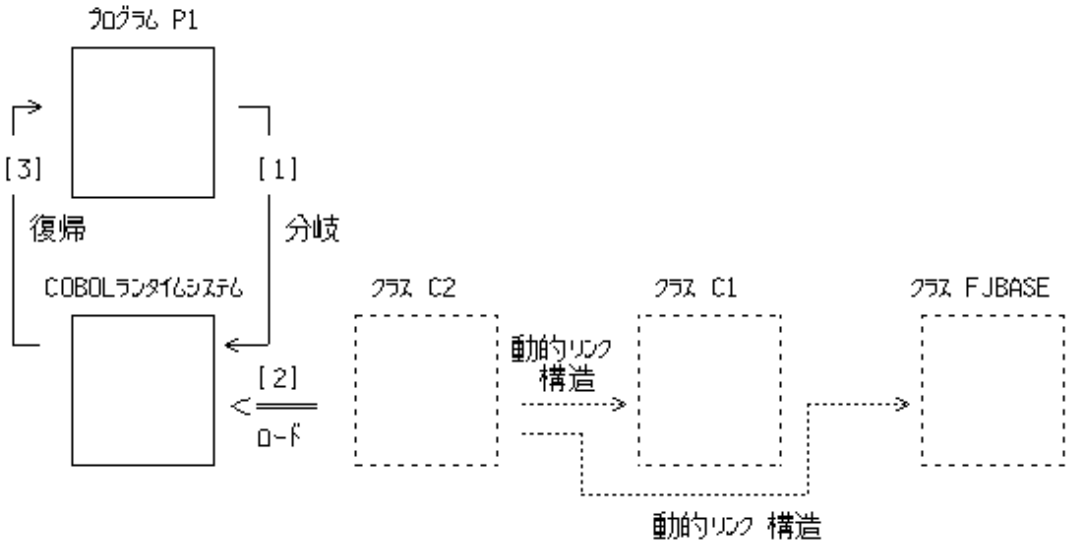
```
CLASS-[D. FJBASE.
    :
```

↓ 継承

```
CLASS-[D. C1 [NHER[TS FJBASE.
    :
```

↓ 継承

```
CLASS-[D. C2 [NHER[TS C1.
    :
```



[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。

16.3.2.3 メソッドの動的プログラム構造

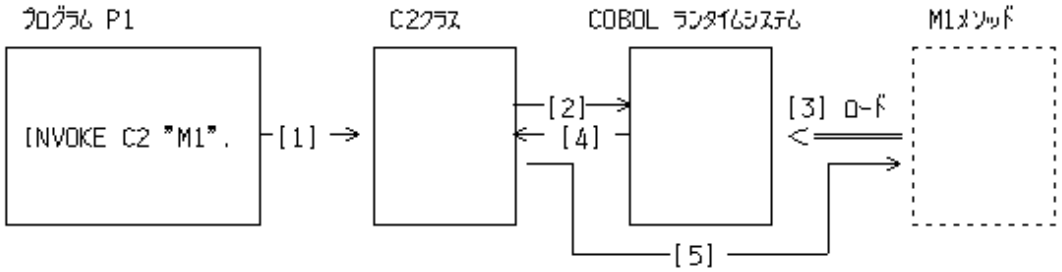
PROTOTYPE宣言により分離されたメソッドを動的プログラム構造にすると、アプリケーションで使用しているメソッドの仮想記憶上へのローディングは、メソッドが呼び出されたときに、COBOLランタイムシステムによって行われます。

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。

```
PROGRAM-ID. P1.
    :
PROCEDURE DIVISION.
      :
    INVOKE C2 "M1".   ... [a]
      :
    EXIT PROGRAM.
END PROGRAM P1.
```

```
CLASS-ID. C2 INHERITS C1.
    :
 FACTORY.
    :
 PROCEDURE DIVISION.
  METHOD-ID. M1 PROTOTYPE.
    :
END CLASS C2.
```

```
METHOD-ID. M1 OF C2.
    :
END METHOD M1.
```



[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。

16.3.2.4 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。

16.3.2.4.1 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。

翻訳の手順については、“第3章 プログラムを翻訳する”を参照してください。

クラスを動的プログラム構造にする場合

クラスを動的プログラム構造にするには、クラスを参照しているソースプログラムの翻訳時に、翻訳オプションDLOADを指定します。この指定により、このソースプログラムから参照されているクラスは、動的プログラム構造になります。[参照]“A.3.11 DLOAD(プログラム構造の指定)”

ただし、他のソースプログラムでそのクラスが動的リンク構造になっていると、実行可能ファイルの起動時にそのクラスはシステムによってロードされてしまいます。このため、COBOLの実行単位でプログラム構造を統一してください。

メソッドを動的プログラム構造にする場合

メソッドを動的プログラム構造にするには、動的プログラム構造にしたいメソッドを原型定義し、そのメソッドの原型定義を含むクラスの翻訳時に、翻訳オプションDLOADを指定します。このように、クラスの翻訳の方法によって、分離されたメソッドのプログラム構造が決まります。

プロパティメソッドを動的プログラム構造にするには、利用者がプロパティメソッドを原型定義し、そのメソッドの原型定義を含むクラスの翻訳時に、翻訳オプションDLOADを指定します。[参照]“A.3.11 DLOAD（プログラム構造の指定）”

例

以下のような継承関係のあるC1クラスとC2クラスがあり、C1クラスが翻訳オプションNODLOADで翻訳され、C2クラスが翻訳オプションDLOADで翻訳されているとします。

この場合、C2クラスで定義されているメソッドのうち、原型定義されているM5メソッドとM7メソッドが動的プログラム構造となります。M6メソッドとM8メソッドは原型定義されていないため、C2クラスが翻訳オプションDLOADで翻訳されても動的プログラム構造になりません。

C1クラスで定義されているメソッドは、C1クラスが翻訳オプションNODLOADで翻訳されているため動的プログラム構造にはなりません。

翻訳オプションNODLOAD 指定

```
CLASS-[D. C1 [NHERITS FJBASE.
    :
 FACTORY.
    :
 PROCEDURE D[V[S[ON.
  METHOD-[D. M1 PROTOTYPE.
    :
 METHOD-[D. M2.
    :
 END FACTORY.
 OBJECT.
    :
 PROCEDURE D[V[S[ON.
  METHOD-[D. M3 PROTOTYPE.
    :
 METHOD-[D. M4.
    :
 END OBJECT.
END CLASS C1.
```

翻訳オプションDLOAD 指定

```
CLASS-[D. C2 [NHERITS C1.
    :
 FACTORY.
    :
 PROCEDURE D[V[S[ON.
  METHOD-[D. M5 PROTOTYPE.
    :
 METHOD-[D. M6.
    :
 END FACTORY.
 OBJECT.
    :
 PROCEDURE D[V[S[ON.
  METHOD-[D. M7 PROTOTYPE.
    :
 METHOD-[D. M8.
    :
 END OBJECT.
END CLASS C2.
```

16.3.2.4.2 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。

リンクの手順については、“第4章 プログラムをリンクする”を参照してください。

プログラムの実行可能ファイルまたはDLLを作成する場合

インポートライブラリは不要です。

クラスのDLLを作成する場合

DLLを作成するクラスが直接および間接的に継承しているすべてのクラスのインポートライブラリが必要となります。

分離されたメソッドのDLLを作成する場合

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。

16.3.2.4.3 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。

16.3.2.4.4 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.2.25 @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。

副プログラムのエントリ情報は実行用の初期化ファイルに指定できますが、クラスおよびメソッドのエントリ情報は、実行用の初期化ファイルに指定できません。

エントリ情報の記述形式

ここでは、クラスおよびメソッドのエントリ情報の記述形式について説明します。

クラスのエントリ情報

クラスのエントリ情報とは、クラスとそのクラスが格納されているDLLを関連付けるための情報です。

以下にクラスのエントリ情報の記述形式について説明します。

```
[CLASS]                    … [1]
クラス名= DLL ファイル名    … [2]
        :
```

[1] クラスのエントリ情報の定義の開始を示すセクション名

セクション名は、“CLASS”の固定文字列です。このセクションは、1つのエントリ情報ファイルに1つしか記述できません。

[2] クラスのエントリ情報

クラス名には利用するクラスのクラス名を指定し、DLLファイル名には利用するクラスが格納されているDLLのファイル名を絶対パス名または相対パス名で指定します。DLLファイル名の拡張子は“DLL”でなければなりません。相対パス名で指定した場合のDLLの検索順序については、“5.6 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.3.1 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。

実行に必要となるエントリ情報

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16.3 実行に必要となるエントリ情報"に示すエントリ情報が必要となります。

表16.3 実行に必要となるエントリ情報

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、"16.1.4 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。

16.3.3 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。

16.3.3.1 概要

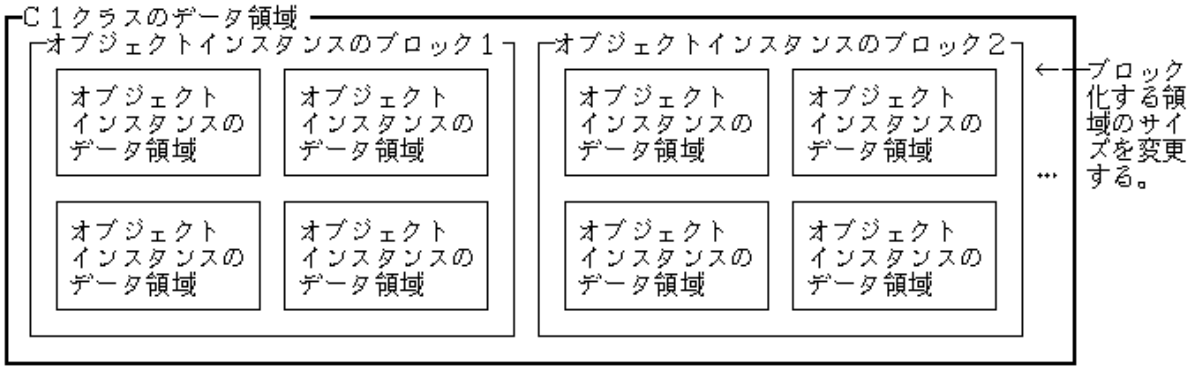
オブジェクト指向プログラムでは、NEWメソッドを実行することにより、オブジェクトインスタンスが生成されます。このとき、オブジェクトインスタンスのデータ部に記述したデータ項目およびオブジェクトインスタンスを動作させるために必要な作業域などが、オブジェクトインスタンスのデータ領域として獲得されます。

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。



16.3.3.2 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になります。したがって、オブジェクトインスタンスのデータ領域をブロック化しなければ、必要以上の領域を獲得することがなく、使用メモリを削減することができます。

オブジェクトインスタンスのデータ領域をブロック化しないで、メモリ優先で領域の獲得を行うためには、以下の指定をします。クラス情報および環境変数情報の指定の詳細については、“16.3.3.4 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
```

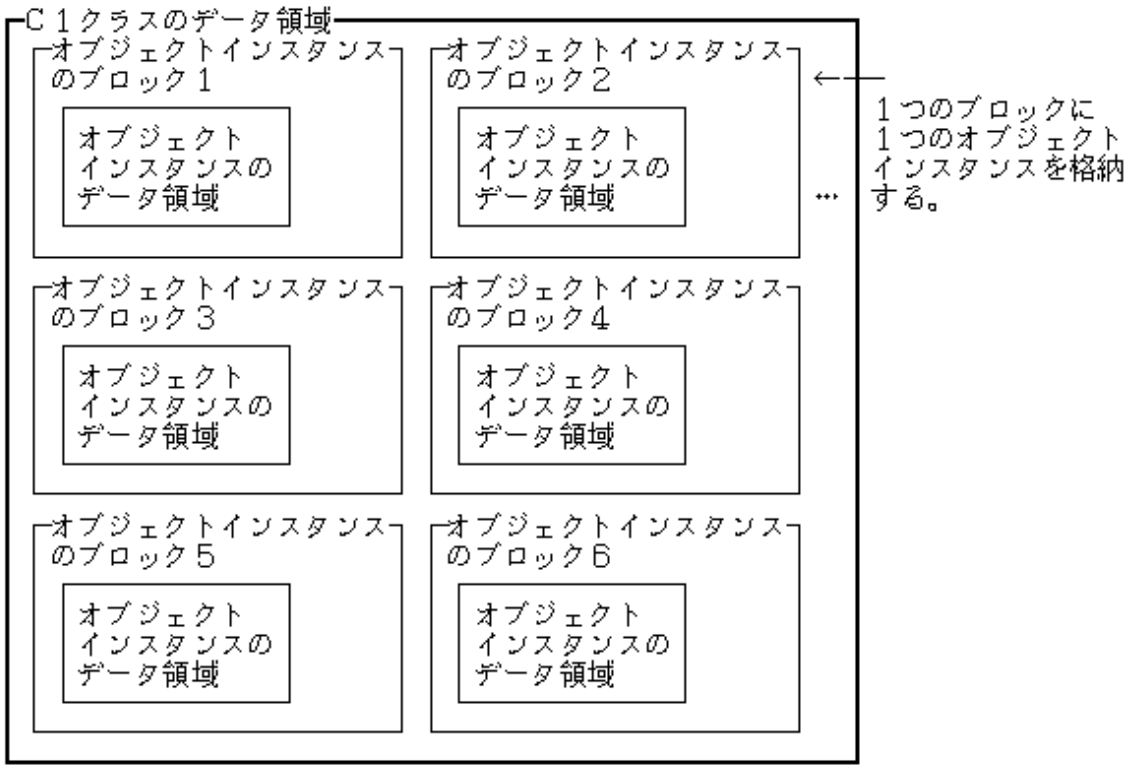
アプリケーションの実行単位に対する指定(環境変数情報)

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。



16.3.3.3 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、“16.3.3.4 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。

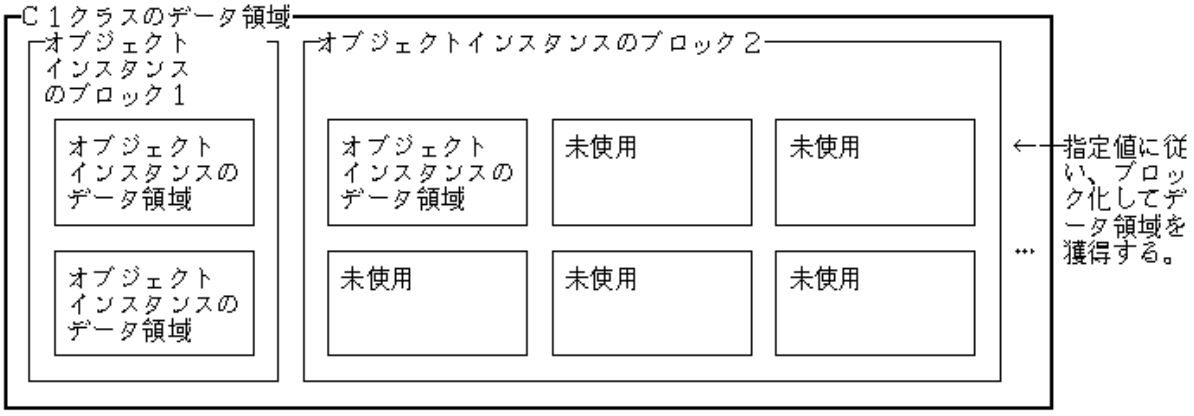
クラスごとの指定(クラス情報)

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。

16.3.3.4 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.2.5 @CBR_ClassInfFile（クラス情報ファイルの指定）”
- “C.2.33 @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。

16.3.3.4.1 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

“16.3.3.4.2 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.2.5 @CBR_ClassInfFile（クラス情報ファイルの指定）”を参照してください。

16.3.3.4.2 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.2.33 @CBR_InstanceBlock（オブジェクトインスタンスの獲得方法の指定）”

16.3.4 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。

16.3.4.1 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。

16.3.4.2 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することができるようになります。つまり、COBOLのINVOKE文によりVisual C++のメンバ関数を呼び出し、プロパティの参照/設定の構文でメンバ変数の参照/設定ができるようになります。

ここでは2.の連携方法について説明します。

16.3.4.3 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。

16.3.4.3.1 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16.4 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16.4 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

16.3.4.3.2 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。

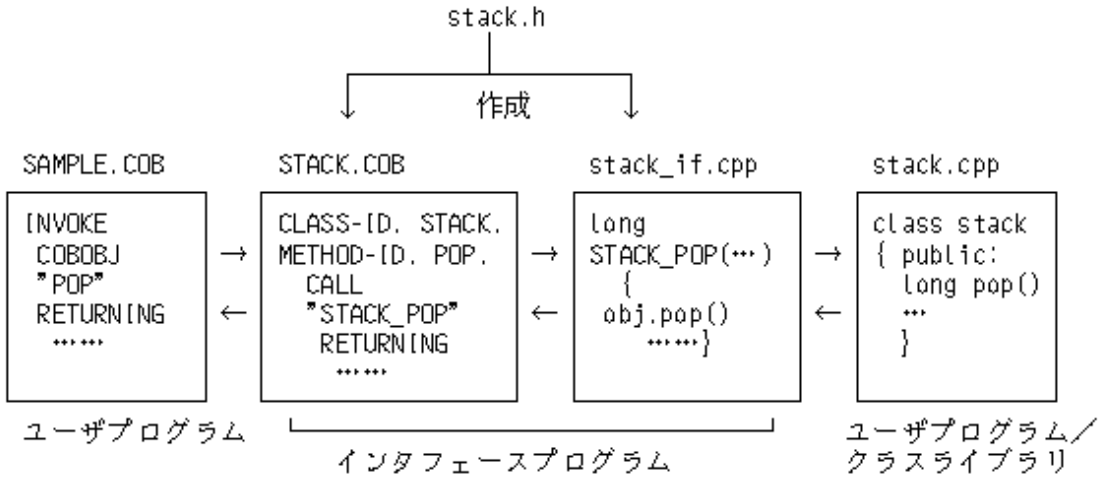
Visual C++側

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16.11 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16.11 インタフェースプログラムのイメージ



16.3.4.4.3.3 インタフェースプログラムの仕組み

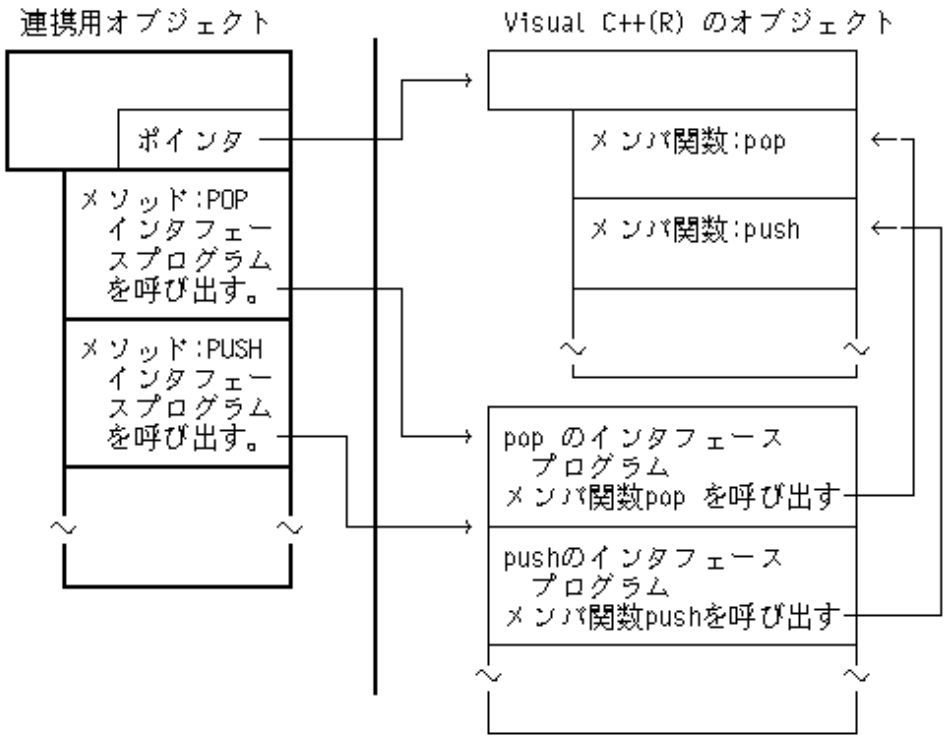
Visual C++のオブジェクトを操作する仕組みは以下のようになります。

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。

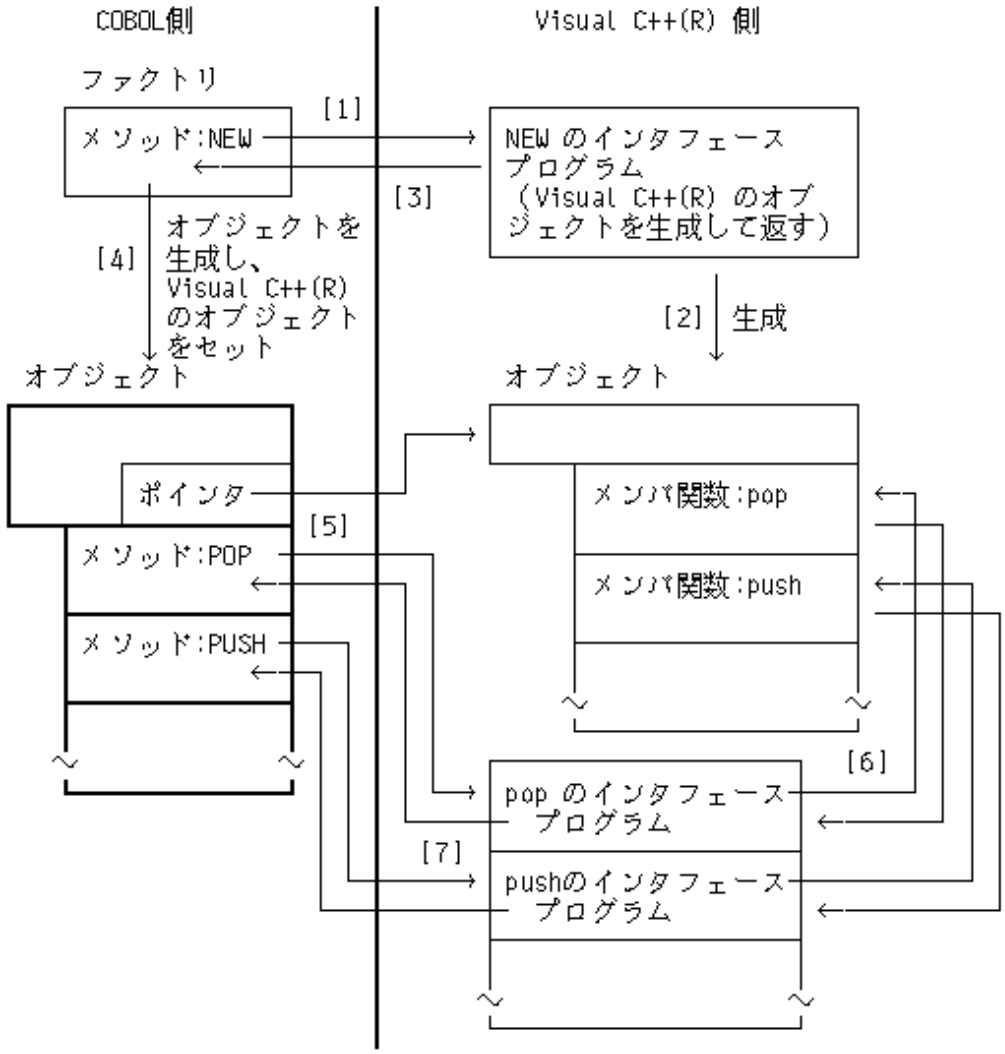
図16.12 インタフェースプログラムの構造



実行時の動き

実行時の制御の流れを“図16.13 実行時の制御の流れ”に示します。

図16.13 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])。

16.3.4.4 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];
} ;
```

16.3.4.4.1 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

16.3.4.4.2 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で参照します。

この例では、STACKとします。

プロパティ

プロパティとして、Visual C++側のメンバ変数に対応するものを定義します。

プロパティの参照および設定では、Visual C++側のメンバ変数参照/設定インタフェースプログラムを呼び出すメソッドを自分で定義します。

この例では、プロパティとしてPNTRを定義します。

また、Visual C++側のオブジェクトを保持するためのポインタ領域を用意します。

この例では、CPP-OBJ-POINTERという名前でポインタデータを定義します。

ファクトリメソッド

ファクトリメソッドとしてNEWメソッドを再定義します。NEWメソッドは以下の処理を行います。

- COBOL側で自クラスのオブジェクトを生成します。
- Visual C++のオブジェクト生成インタフェースプログラムを呼び出し、Visual C++側で生成するオブジェクトを取得します。このオブジェクトへのポインタを、Visual C++側のオブジェクトを保持するための領域CPP-OBJ-POINTERに保存します。

オブジェクトメソッド

オブジェクトメソッドとしてVisual C++側の定義と同じ名前のメソッドを定義します。

個々のメソッドは、対応するVisual C++側のメンバ関数呼出しインタフェースプログラムを呼び出します。

この例では、オブジェクトメソッドとしてPOPメソッド、PUSHメソッドを定義します。また、Visual C++側で生成したオブジェクトを削除するDELETE-OBJメソッドを定義します。DELETE-OBJメソッドは、オブジェクト削除インタフェースプログラムを呼び出します。

16.3.4.4.3 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16.5 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	

16.3.4.5 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。

COBOLからの使用例

```
 WORKING-STORAGE SECTION.
 01  STACKOBJ USAGE OBJECT REFERENCE STACK.
 01  POP-VALUE PIC S9(9) COMP-5.
PROCEDURE DIVISION.
    INVOKE STACK "NEW" RETURNING STACKOBJ.
    INVOKE STACKOBJ "PUSH" USING 10.
    INVOKE STACKOBJ "PUSH" USING 20.
    INVOKE STACKOBJ "PUSH" USING 30.
    INVOKE STACKOBJ "POP" RETURNING POP-VALUE.
    INVOKE STACKOBJ "DELETE-OBJ".
    SET STACKOBJ TO NULL.
      :
```

16.3.4.6 サンプルプログラム

ここで説明したインタフェースプログラムの定義の例を“Visual C++連携のCOBOLのクラス定義”および“Visual C++連携のVisual C++側のインタフェースプログラム”に示します。

Visual C++連携のCOBOLのクラス定義

```
*>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
*>  CLASS STACKを定義する
*>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
 IDENTIFICATION DIVISION.
 CLASS-ID.   STACK INHERITS FJBASE.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     CLASS FJBASE.
*>
*>  FACTORYの定義
*>    NEWメソッドを再定義する
 IDENTIFICATION DIVISION.
 FACTORY.
 PROCEDURE DIVISION.
 IDENTIFICATION DIVISION.
 METHOD-ID. NEW OVERRIDE.
 DATA                DIVISION.
 WORKING-STORAGE     SECTION.
  01 CPP-STACK USAGE IS POINTER.
 LINKAGE SECTION.
  01 STACKOBJ USAGE OBJECT REFERENCE  SELF.
 PROCEDURE DIVISION RETURNING STACKOBJ.
      INVOKE SUPER "NEW" RETURNING STACKOBJ.
      CALL "CPP_STACK_NEW" RETURNING CPP-STACK.
      INVOKE STACKOBJ "SET-CPP-OBJ-POINTER" USING CPP-STACK.
      EXIT METHOD.
 END METHOD NEW.
 END FACTORY.
*>
*>  OBJECTの定義
*>    オブジェクトデータとしC++側のオブジェクトを保持するデータ
*>    (CPP-OBJ-POINTER)を定義する
 IDENTIFICATION DIVISION.
 OBJECT.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
  01 CPP-OBJ-POINTER USAGE IS POINTER.
 PROCEDURE DIVISION.
*>
*>  SET-CPP-OBJ-POINTERメソッドの定義
*>
 IDENTIFICATION DIVISION.
 METHOD-ID. SET-CPP-OBJ-POINTER.
 DATA DIVISION.
 LINKAGE SECTION.
  01 USE-VALUE USAGE IS POINTER.
 PROCEDURE DIVISION USING USE-VALUE.
     MOVE CPP-OBJ-POINTER TO USE-VALUE.
     EXIT METHOD.
 END METHOD SET-CPP-OBJ-POINTER.
*>
*>  POPメソッドの定義
*>
 IDENTIFICATION DIVISION.
 METHOD-ID. POP.
 DATA DIVISION.
```

```
 LINKAGE SECTION.
  01 RET-VALUE PIC S9(9) COMP-5.
 PROCEDURE DIVISION RETURNING RET-VALUE.
     CALL "CPP_STACK_POP" USING CPP-OBJ-POINTER RETURNING RET-VALUE.
     EXIT METHOD.
 END METHOD POP.
*>
*>  PUSHメソッドの定義
*>
 IDENTIFICATION DIVISION.
 METHOD-ID. PUSH.
 DATA DIVISION.
 LINKAGE SECTION.
  01 SET-VALUE PIC S9(9) COMP-5.
 PROCEDURE DIVISION USING SET-VALUE.
     CALL "CPP_STACK_PUSH" USING CPP-OBJ-POINTER SET-VALUE.
     EXIT METHOD.
 END METHOD PUSH.
*>
*>  PNTRを参照するメソッドの定義
*>
 IDENTIFICATION DIVISION.
 METHOD-ID. GET PROPERTY PNTR.
 DATA DIVISION.
 LINKAGE SECTION.
  01 RET-VALUE PIC S9(9) COMP-5.
 PROCEDURE DIVISION RETURNING RET-VALUE.
     CALL "CPP_STACK_GET_PNTR" USING CPP-OBJ-POINTER RETURNING RET-VALUE.
     EXIT METHOD.
 END METHOD.
*>
*>  PNTRを設定メソッドの定義
*>
 IDENTIFICATION DIVISION.
 METHOD-ID. SET PROPERTY PNTR.
 DATA DIVISION.
 LINKAGE SECTION.
  01 SET-VALUE PIC S9(9) COMP-5.
 PROCEDURE DIVISION USING SET-VALUE.
     CALL "CPP_STACK_SET_PNTR" USING CPP-OBJ-POINTER SET-VALUE.
     EXIT METHOD.
 END METHOD.
*>
*>  DELETE-OBJメソッドの定義
*>
 IDENTIFICATION DIVISION.
 METHOD-ID. DELETE-OBJ.
 DATA DIVISION.
 LINKAGE SECTION.
 PROCEDURE DIVISION.
     CALL "CPP_STACK_DELETE_OBJ" USING CPP-OBJ-POINTER.
     EXIT METHOD.
 END METHOD DELETE-OBJ.
 END OBJECT.
 END CLASS STACK.
```

Visual C++連携のVisual C++側のインタフェースプログラム

```
#include "stack.h"

extern "C" stack* CPP_STACK_NEW() {
  return new stack;
}
```

```
extern "C" long int CPP_STACK_POP(stack** stk) {
  return (*stk)->pop();
}

extern "C" void CPP_STACK_PUSH(stack** stk, long int* value) {
  (*stk)->push(*value);
}

extern "C" long int CPP_STACK_GET_PNTR(stack** stk) {
  return (*stk)->pointer;
}

extern "C" void CPP_STACK_SET_PNTR(stack** stk, long int* value) {
  (*stk)->pointer = *value;
}

extern "C" void CPP_STACK_DELETE_OBJ(stack** stk) {
  delete *stk;
}
```

16.3.5 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。

16.3.5.1 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。

16.3.5.2 概要

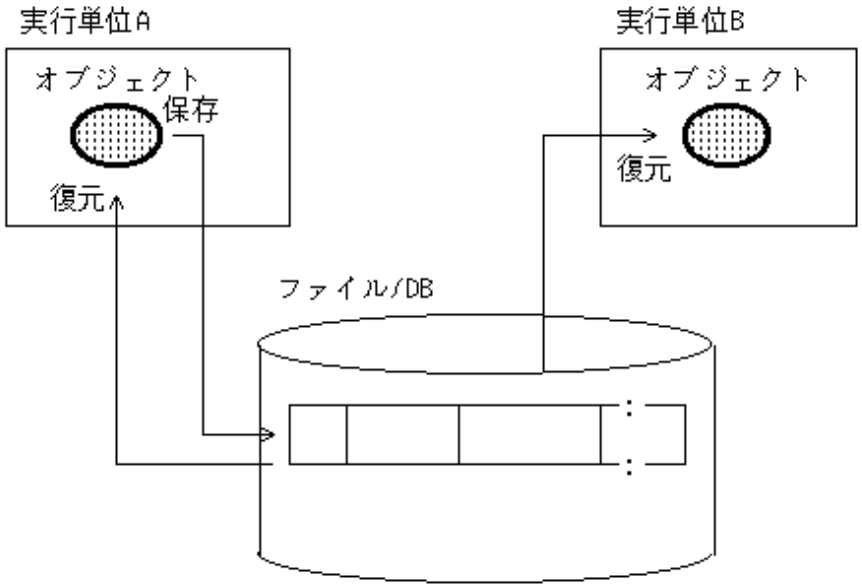
オブジェクトの永続化は、COBOLの実行中に生成したオブジェクトを外部記憶装置に保存し、別の実行単位で読み戻すことで実現します。保存のための記憶装置には、データベース、ファイルなどが用いられます([参照]"図16.14 永続オブジェクトの流れ")。

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16.14 永続オブジェクトの流れ



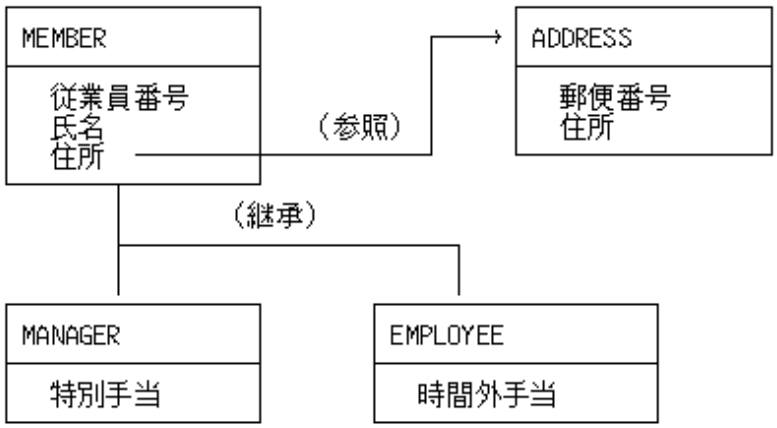
ここでは、索引ファイルを利用してオブジェクトの永続化を行う方法について説明します。

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”

16.3.5.3 クラス構造の例

本節の説明で使用するクラス構造は、“図16.15 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16.15 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。

16.3.5.4 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。

16.3.5.4.1 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

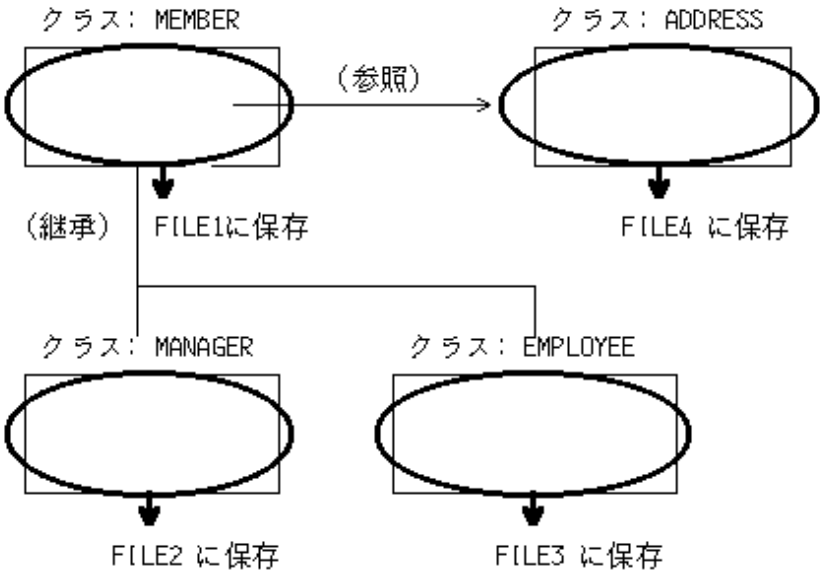
オブジェクトと保存ファイルの対応の基本は、クラスとファイルを一対一に対応付けることです。この例では、MEMBERをFILE1に、MANAGERをFILE2に、EMPLOYEEをFILE3に、ADDRESSをFILE4に保存することになります([参照]“図16.16 クラスごとに異なるファイルに保存する場合”)。

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16.16 クラスごとに異なるファイルに保存する場合



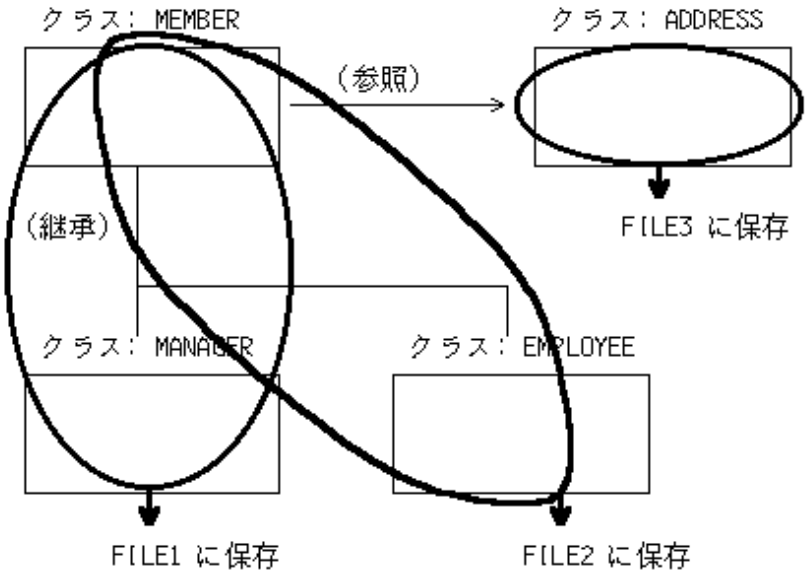
親クラスのオブジェクトデータも含めて1つのファイルに保存する方法

MEMBERクラスが純粋に抽象クラスとして定義されている場合、MEMBERクラスとして独立したオブジェクトは存在しません。この場合、MANAGERクラスの保存では、MEMBERクラスのオブジェクトデータとMANAGERクラスのオブジェクトデータとを同じファイルに保存した方が、クラスごとに保存ファイルを分ける方法に比べて処理が簡単になります。EMPLOYEEクラスについても同じように、MEMBERクラスのオブジェクトデータとEMPLOYEEクラスのオブジェクトデータを同じファイルに保存することができます。[参照]“図16.17 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のか判断できません。

この方法は、保存するクラスが1つしかない場合には有効な方法ですが、“図16.17 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。

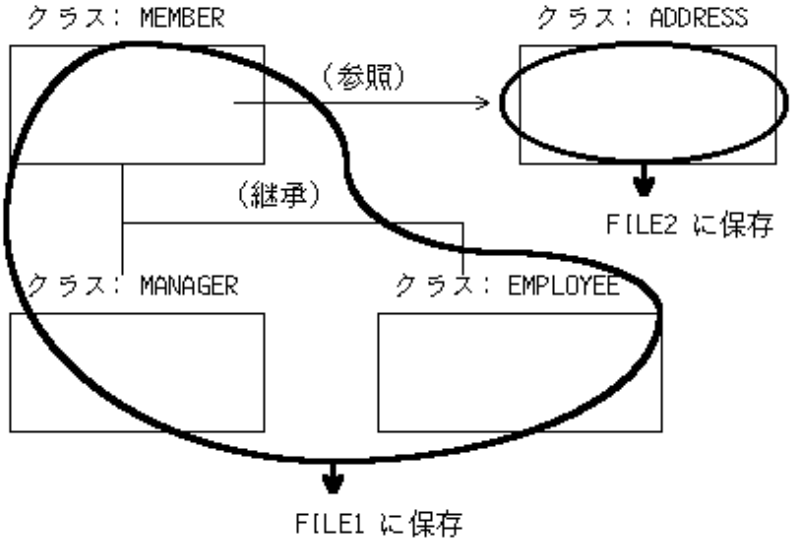
図16.17 親クラスのオブジェクトデータも含めて1つのファイルに保存する方法



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16.17 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16.18 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。

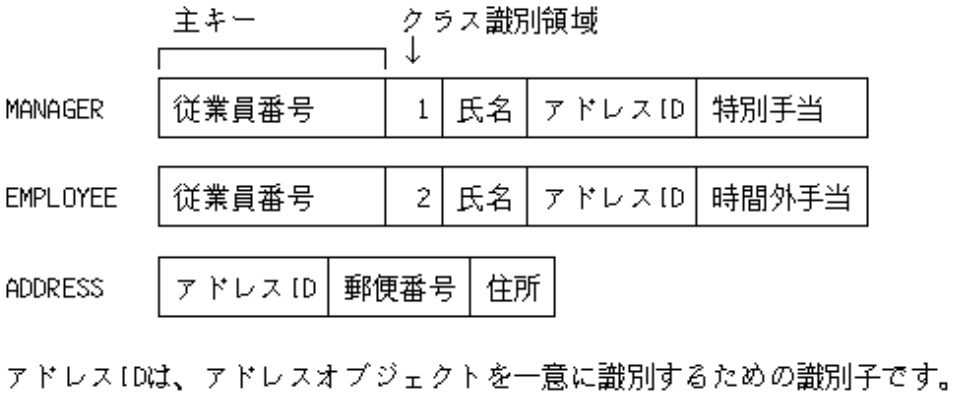
ここでは、“図16.18 同じ親クラスを持つクラスを同じファイルに保存する方法”に従って説明します。

16.3.5.4.2 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16.19 索引ファイル中のレコード



16.3.5.5 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。

16.3.5.5.1 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。

16.3.5.5.2 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

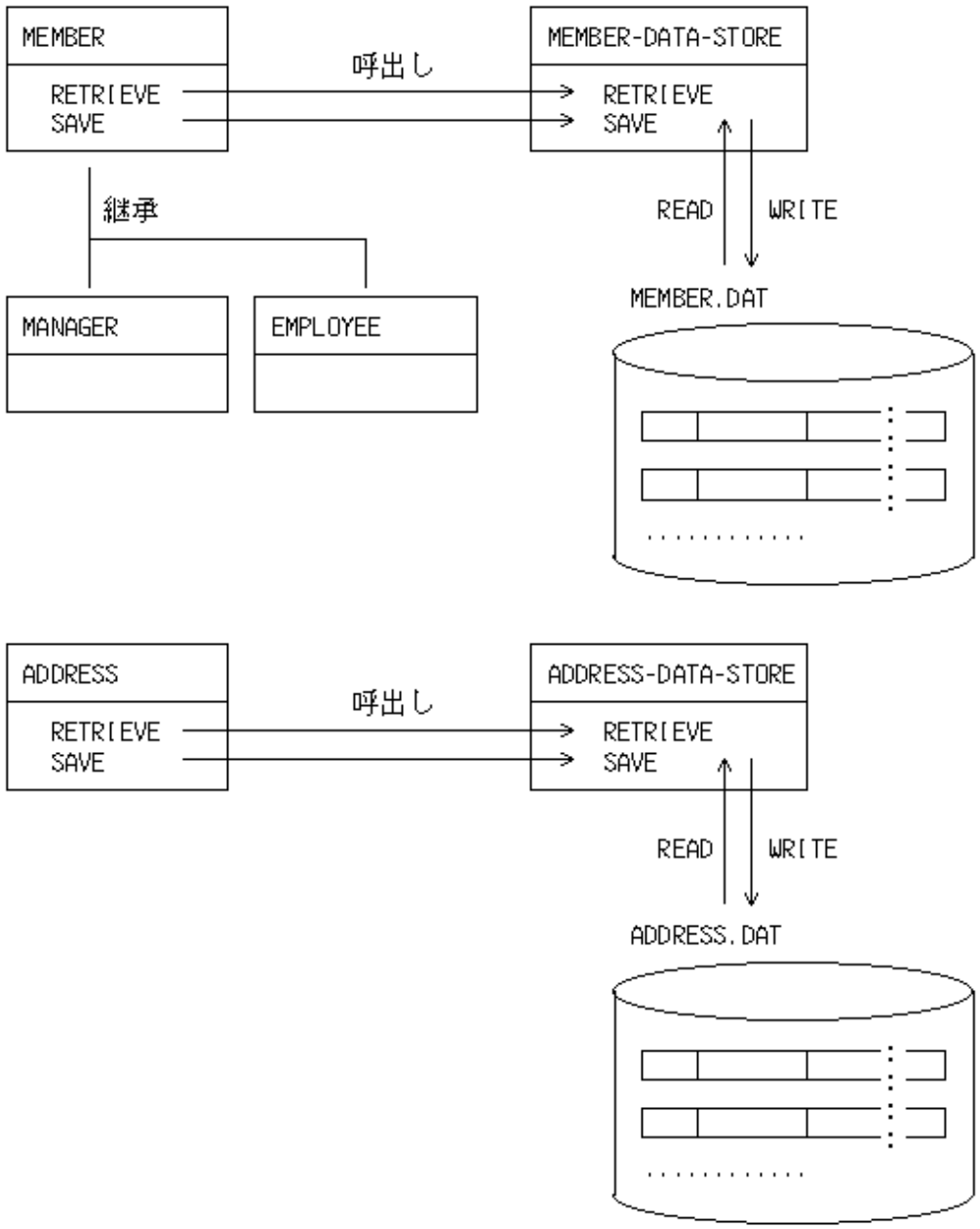
SAVEメソッドはオブジェクトを保存するメソッドです。実際には、自分自身を引数として、対応する索引ファイル操作クラスのSAVEメソッドを呼び出します。SAVEメソッドは、親クラスで定義すれば十分で、個々のクラスに定義する必要はありません。

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16.6 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16.19 索引ファイル中のレコード”に示します。

表16.6 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16.20 索引ファイル操作クラスとの関係



16.3.5.5.3 処理の流れ

保存/復元の処理の流れを“図16.20 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

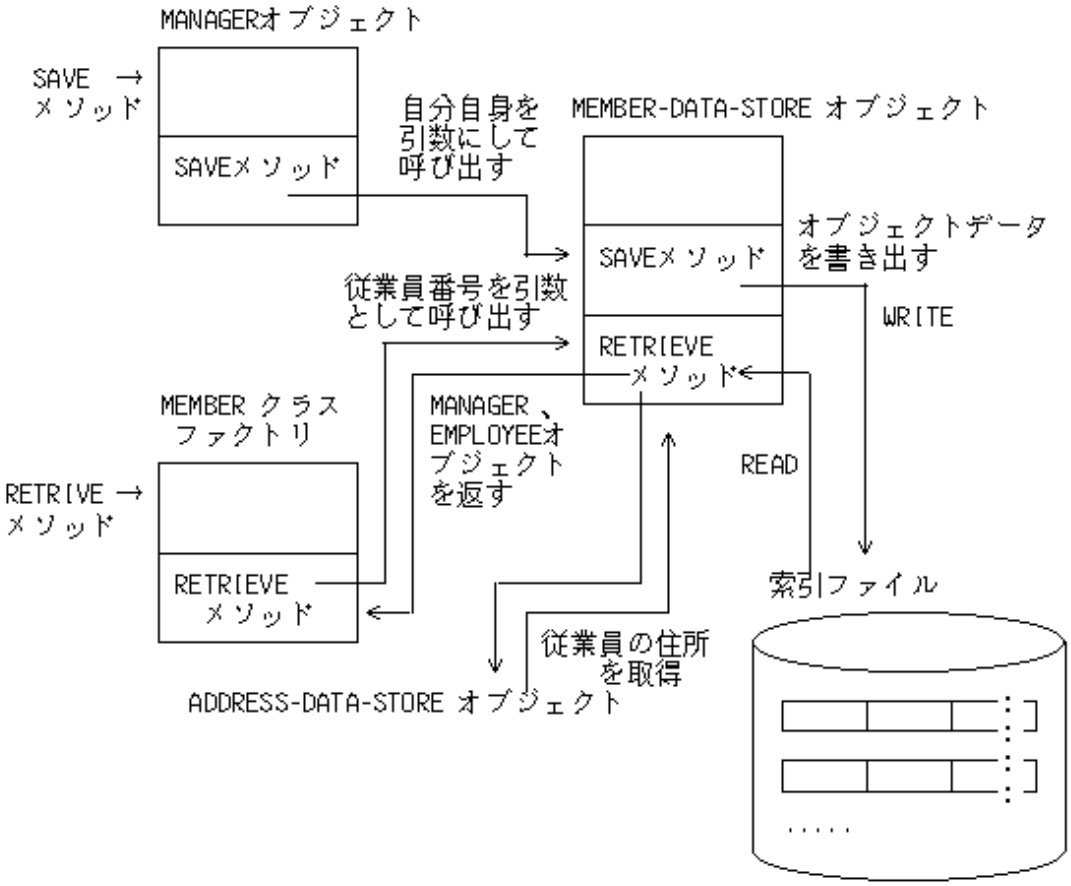
MANAGERオブジェクトに対してSAVEメソッドが呼ばれると、SAVEメソッドは自分自身を引数にして、MEMBER用の索引ファイル操作オブジェクト(MEMBER-DATA-STOREオブジェクト)のSAVEメソッドを呼び出します。MEMBER-DATA-STOREオブジェクトのSAVEメソッドは、引数のオブジェクトのクラスを調べ(この場合MANAGERクラスになる)、MANAGERクラス用のレコードにデータを転記し、索引ファイルに書き込みます。

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(ADDRESSオブジェクトの復元については“図16.21 保存/復元の処理の流れ”では省略)。

図16.21 保存/復元の処理の流れ



16.3.6 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。

16.3.6.1 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。

そうなると、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。変更に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れば、呼び出すメソッドに文字列の実際の長さを渡し、その長さで部分参照付けするような複雑な処理が必要になってしまいます。

たとえば、下の例のように名前とパスワードを認証するクラス(メソッド)を作成したい場合、文字列の最大長を決めておく必要があります。以下の例は、名前を日本語で10文字、パスワードを英数字で8文字として作成しています。

文字列を渡す側

```
PROGRAM-ID.   情報変更.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
 REPOSITORY.
     CLASS  認証クラス.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  名前文字列.
  02  名前          PIC N(10).
01  パスワード   PIC X(8).
01  認証結果     PIC X(2).
01  認証オブジェクト  USAGE OBJECT REFERENCE  認証クラス.
PROCEDURE DIVISION.
    INVOKE  認証クラス "NEW" RETURNING  認証オブジェクト.
    DISPLAY  NC"名前とパスワードを入力してください".
    ACCEPT  名前文字列.
    ACCEPT  パスワード.
    INVOKE  認証オブジェクト "認証メソッド" USING 名前  パスワード
                                          RETURNING   認証結果.
    IF  認証結果 = "OK" THEN
      CALL "情報変更処理"
    ELSE
      DISPLAY NC"変更する資格がありません"
    END-IF.
END PROGRAM 情報変更.
```

渡された文字列を扱う汎用クラス

```
CLASS-ID.   認証クラス INHERITS FJBASE.
 :
OBJECT.
PROCEDURE DIVISION.
METHOD-ID.  認証メソッド.
DATA DIVISION.
LINKAGE SECTION.
01  名前          PIC N(10).
01  パスワード   PIC X(8).
01  認証結果     PIC X(2).
PROCEDURE DIVISION USING  名前  パスワード
                  RETURNING  認証結果.
     EVALUATE 名前        ALSO  パスワード
     WHEN   NC"富士一夫" ALSO "A1"
     WHEN   NC"富士二郎" ALSO "XXXXYYY2"
       MOVE  "OK" TO  認証結果
     WHEN OTHER
       MOVE  "NG" TO  認証結果
     END-EVALUATE.
END METHOD  認証メソッド.
END OBJECT.
END CLASS   認証クラス.
```

この状態で、認証クラスを利用する処理が他にも発生し、パスワードの最大長を変更したくなった場合、まず、認証クラスを修正します。すると、それに引きずられる形で情報変更のプログラムの修正が必要になります。また、インタフェースが変更されるため、認証クラスを継承するクラスの再翻訳が必要となります。

16.3.6.2 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
01  パスワード   PIC X  ANY LENGTH.    |    きに、呼出し側の対応する項目の
01  認証結果     PIC X  ANY LENGTH.  ←    長さに決まる。
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。

16.4 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。

16.4.1 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。 ・ ALL 定数と数字項目または数字編集項目との関連付け ・ AUTHOR段落、INSTALLATION段落、DATE-WRITTEN段落、DATE-COMPILED 段落およびSECURITY段落 ・ RERUN句 ・ MULTIPLE FILE TYPE句 ・ LABEL RECORD句 ・ VALUE OF句 ・ DATA RECORDS句 ・ ALTER文 ・ ENTER文 ・ 手続き名-1を省略したGO TO文

機能	説明
	・ OPEN文のREVERSED指定 ・ 定数指定のSTOP文
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。
スクリーン操作機能	ファクトリ定義、オブジェクト定義およびメソッド定義では、スクリーン操作機能を使用できません。
CHARACTER TYPE句	ファクトリ定義およびオブジェクト定義のデータ部では、CHARACTER TYPE句を指定できません。ただし、メソッド定義のデータ部では指定できます。 また、メソッド原型定義の連絡節には、CHARACTER TYPE句を指定できません。
EXTERNAL句	ファクトリ定義およびオブジェクト定義のデータ部では、EXTERNAL句を指定できません。ただし、メソッド定義のデータ部では指定できます。
GLOBAL句	ファクトリ定義、オブジェクト定義およびメソッド定義のデータ部には、GLOBAL句を指定できません。ファクトリ定義およびオブジェクト定義のデータ部で宣言された名前は、すべて大域名として扱われます。
LINAGE句	ファクトリ定義およびオブジェクト定義で定義したファイルには、LINAGE句を指定できません。メソッド定義で定義したファイルには指定できます。ただし、EXTERNAL句を指定したファイルには指定できません。
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。

16.4.2 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。“16.4.1 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。 ・ ASSIGN句にPRINTER またはPRINTER-n (nは1～9までの整数) が指定されている。 ・ 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。 ・ FORMAT句付き印刷ファイルである。

第4部 サーバサイドアプリケーションの開発と運用

第17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。

17.1 バックグラウンド処理

17.1.1 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
  - 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.2.36 @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止する方法

   環境変数情報@WinCloseMsg=OFFを指定します。詳細は、“C.2.74 @WinCloseMsg（ウィンドウを閉じるときのメッセージ表示の指定）”を参照してください。

DISPLAY文でログ情報をコンソール画面に出力するような場合、COBOLアプリケーションの終了時に、COBOLがコンソール画面を閉じる旨を通知するメッセージを抑止したい場合に指定します。

17.1.2 コマンドプロンプトウィンドウの使用方法

コマンドプロンプトウィンドウを使用して、ACCEPT文の入力、DISPLAY文の出力および実行時メッセージの表示を行うことができます。

コマンドプロンプトウィンドウやバッチファイルから起動した場合は、起動したコマンドプロンプトウィンドウまたはバッチファイルが生成するウィンドウをコンソール画面として使用します。また、アイコンの状態で起動した場合は、起動と同時にコマンドプロンプトウィンドウが生成され、これをコンソール画面として使用します。

主プログラムがCOBOLの場合、翻訳オプションMAIN(MAIN)を指定します。主プログラムが他言語の場合、環境変数情報@CBR_CONSOLE=SYSTEMを指定します。

環境変数情報@CBR_CONSOLE=SYSTEMの指定は、主プログラムがCOBOLの場合で翻訳オプションMAIN(WINMAIN)を指定した場合、または、主プログラムが他言語でWinMain関数の場合、期待したとおりに動作しない場合があります。この場合、@CBR_CONSOLE=COBOL(省略値)の指定をおすすめします。また、環境変数情報@CBR_CONSOLE=SYSTEMを指定した場合の動作は、以下のとおりです。

1. 新しいコマンドプロンプトウィンドウが生成されます。
   起動したコマンドプロンプトウィンドウやバッチファイルが生成するウィンドウはコンソール画面として使用できません。コンソール機能の使用時に新しくコマンドプロンプトウィンドウを生成して、コンソール画面として使用します。
2. 入出力に失敗する場合、STARTコマンドを使用して回避してください。なお、コマンドプロンプトウィンドウへの入出力に失敗する場合、コマンドプロンプトウィンドウへの実行時メッセージの出力にも失敗するため、エラーの事象を確認できないことがあります。実行時メッセージをイベントログやファイルに出力することをおすすめします。

17.1.3 サービス配下で動作するプログラム

サービス配下のアプリケーション環境

サービスからCOBOLプログラムが呼び出される場合、プリンタ情報やネットワークドライブ情報などの取得に失敗して、正常に動作しない場合があります。

原因は、サービスから呼び出されたCOBOLプログラムがプリンタ情報などをシステムから取得する際、システムに設定されているユーザごとの設定情報を参照できないためと考えられます。

通常、サービスから呼び出されるCOBOLプログラムはシステムアカウントで動作します。このとき、システム上にはユーザごとの設定(たとえば、レジストリのHKEY_CURRENT_USERなどの情報)は構築されません。このため、COBOLプログラムはデフォルトプリンタなどのユーザごとの情報を取得することができません。ユーザごとの設定は、そのユーザがシステムにログオンしたときに構築されます。

サービスによっては、特定のユーザアカウントでCOBOLプログラムが動作するように設定することが可能な場合もあります。このときは、COBOLプログラムが動作するユーザアカウントと同じユーザであらかじめシステムにログオンしておくと、ユーザの設定が参照できます。また、COBOLプログラムがシステムアカウントで動作しても、ユーザごとの情報をシステムの情報として参照できるように作成されたサービスもあります。しかし、前述したようにサービスから呼び出されたCOBOLプログラムでは、ユーザごとの情報を参照できないのが一般的です。

プログラム作成時の注意事項

サービスから呼び出されるCOBOLプログラムを作成するときには、以下の注意が必要です。

環境変数

ユーザ環境変数に設定した環境変数情報は参照できません。COBOLプログラムで必要な環境変数情報は、システム環境変数に設定してください。

プリンタ情報

FORMAT句あり/なし印刷ファイルおよび表示ファイル(宛て先PRT)を使用したCOBOLプログラムをサービス配下で実行する場合、そのファイルの出力先への割当ては明にプリンタ名を指定してください。出力先への指定にデフォルトのプリンタ、ローカルプリンタポート名(LPTn:)および通信ポート名(COMn:)を指定した場合、印刷されないことがあります(ファイルの割当てエラー)。詳細は、“8.1.15 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、“17.1.1 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、NetCOBOL Studioのアタッチデバッグ機能を使用します。手順の詳細は、“NetCOBOL Studio ユーザーズ”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。

17.2 イベントログ

イベントログは、Windows(x64)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で行います。

17.2.1 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 x64、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17.1 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.2.36 @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“第3章 実行時メッセージ”を参照してください。

注意

ネットワーク上の他のコンピュータ(Windows(x64))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、必ず本製品(COBOLまたはCOBOLランタイムシステム)を必ずインストールしてください。

17.2.2 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.1.4 イベントログ出力サブルーチン(COB_REPORT_EVENT)”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多いですが、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 x64。
イベントID	ソースと共にイベントを識別するために使用します。0 ～999 までの値が指定可能です。
種類	イベントの種類として情報／警告／エラーを指定します。
説明	イベントの説明文を最大1024バイトまで指定できます。指定を省略した場合、説明に対する情報は出力されません。
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(x64))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。
  詳細は、“17.2.1 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18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。

18.1 概要

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。

18.1.1 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行い、さらに、スレッドの同期制御を行うためのサブルーチンを提供しているため、プログラムを簡単に作成することができます。

マルチスレッドでのデバッグ支援

NetCOBOLの提供するTRACE機能、CHECK機能、COUNT機能および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。

18.1.2 機能範囲

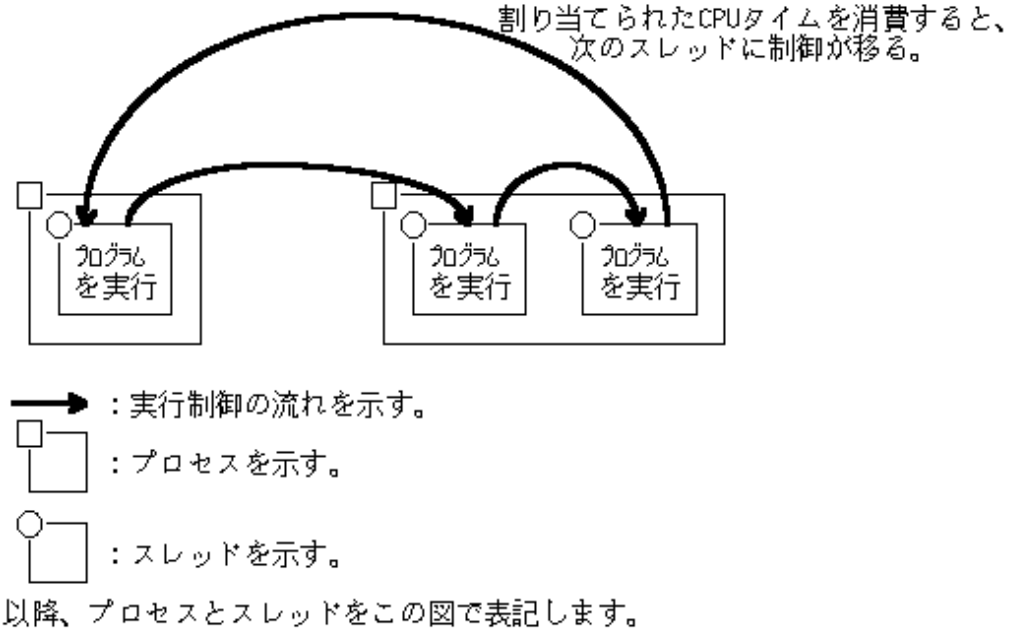
マルチスレッド環境では、COBOLの機能をすべて利用することができます。ただし、関連製品を利用する機能については、その関連製品の対応状況を確認の上、使用してください。[参照]“NetCOBOL ソフトウェア説明書”

18.2 マルチスレッドのメリット

18.2.1 スレッドとは

スレッドとは、オペレーティングシステムによってスケジュール管理される最小の実行単位です。オペレーティングシステムは、実行するスレッドおよびその実行時期をスケジュールし、CPUタイムを割り当てます。割り当てられたCPUタイムを消費すると、次のスレッドにCPUタイムを割り当てます。

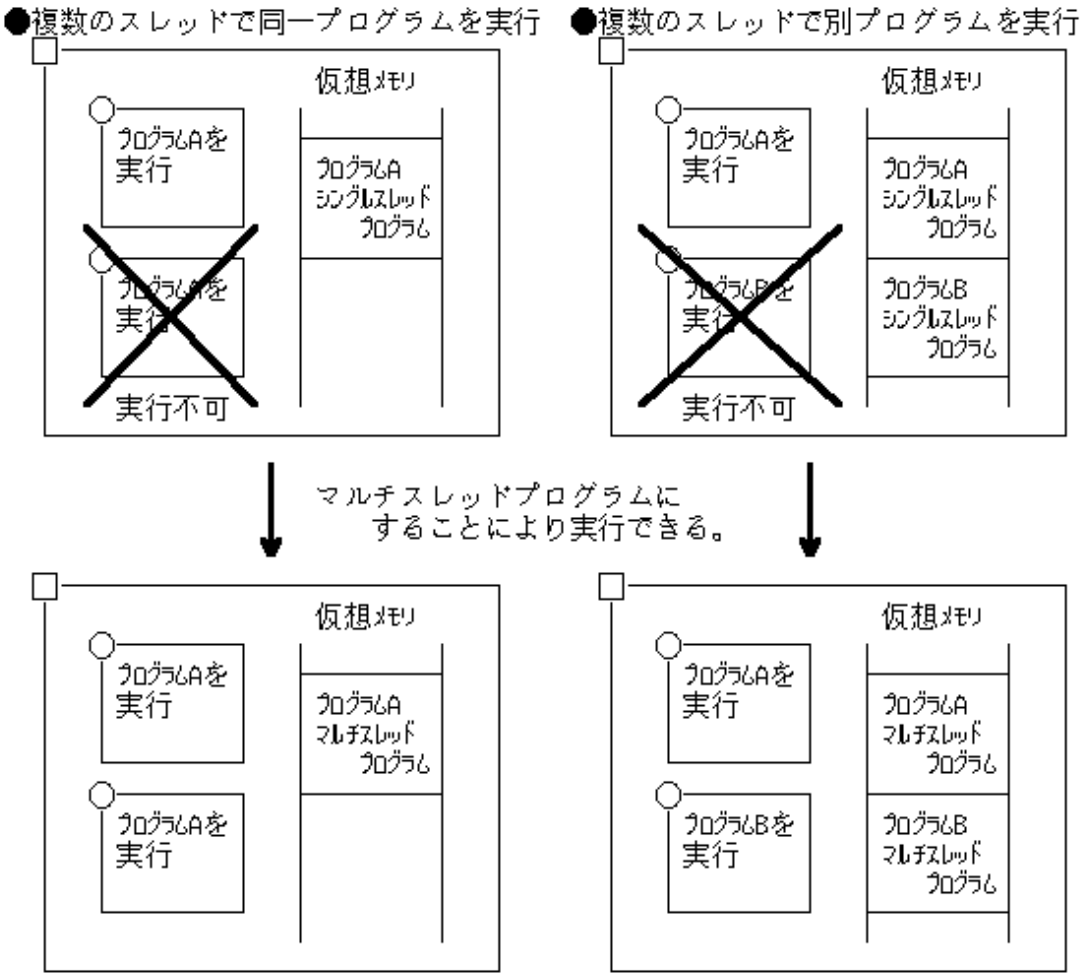
スレッドはプロセスの中に存在し、スレッドによってプログラムが実行されます。プロセスは、メモリ上にあるプログラムのコード、データ、オープンされているファイル、動的に割り当てられたメモリなどの資源から構成され、少なくとも1つのスレッドが存在します。



18.2.2 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。



18.2.3 マルチスレッドの効果

サーバから起動される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。

18.3 マルチスレッドプログラムの基本動作

18.3.1 実行環境と実行単位

“10.1.2 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。

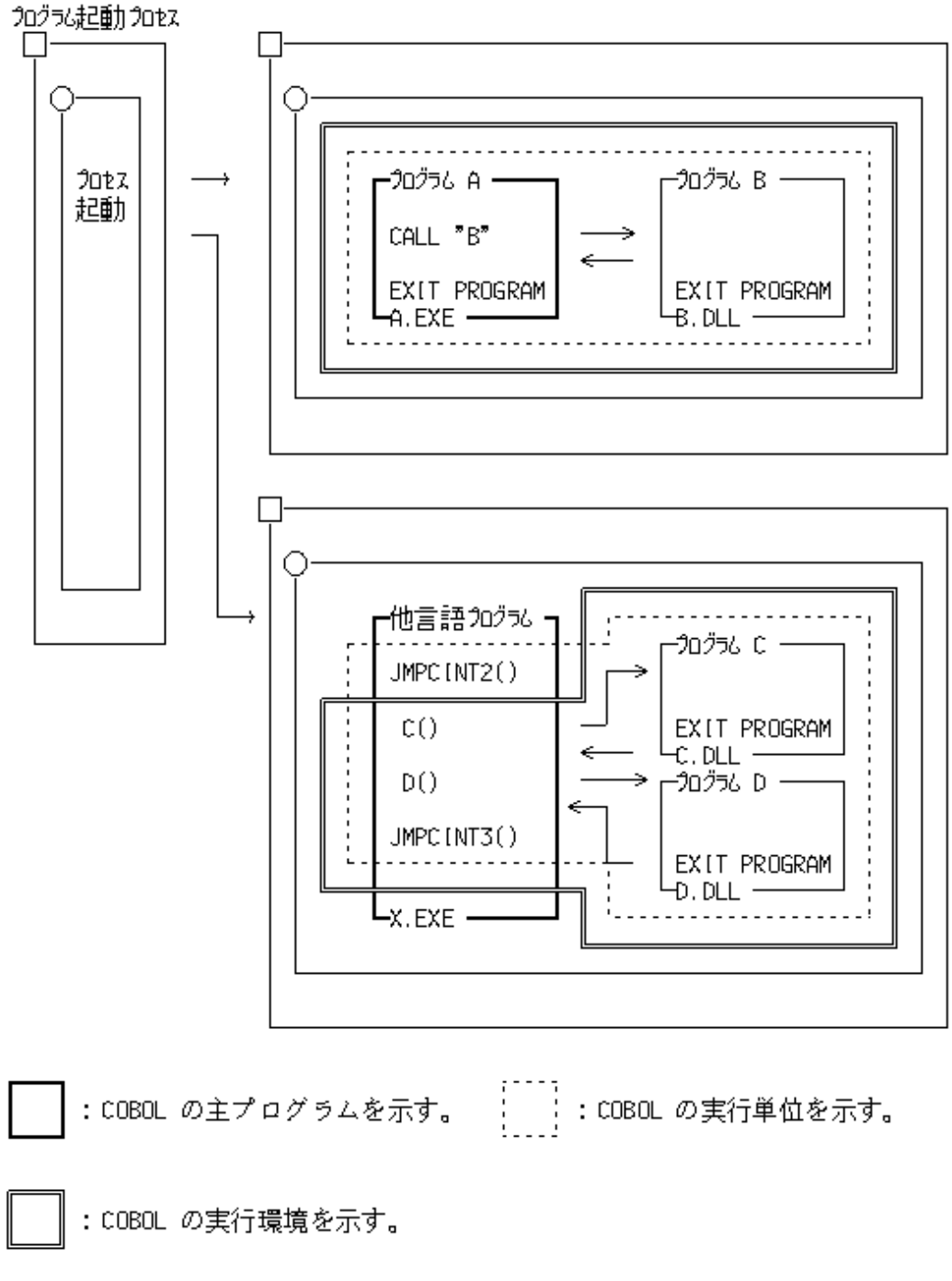
JMPCINT4

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、“I.2 他言語連携で使用するサブルーチン”を参照してください。

以下に、シングルスレッドプログラムとマルチスレッドプログラムの実行環境と実行単位の関係を図示します。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。

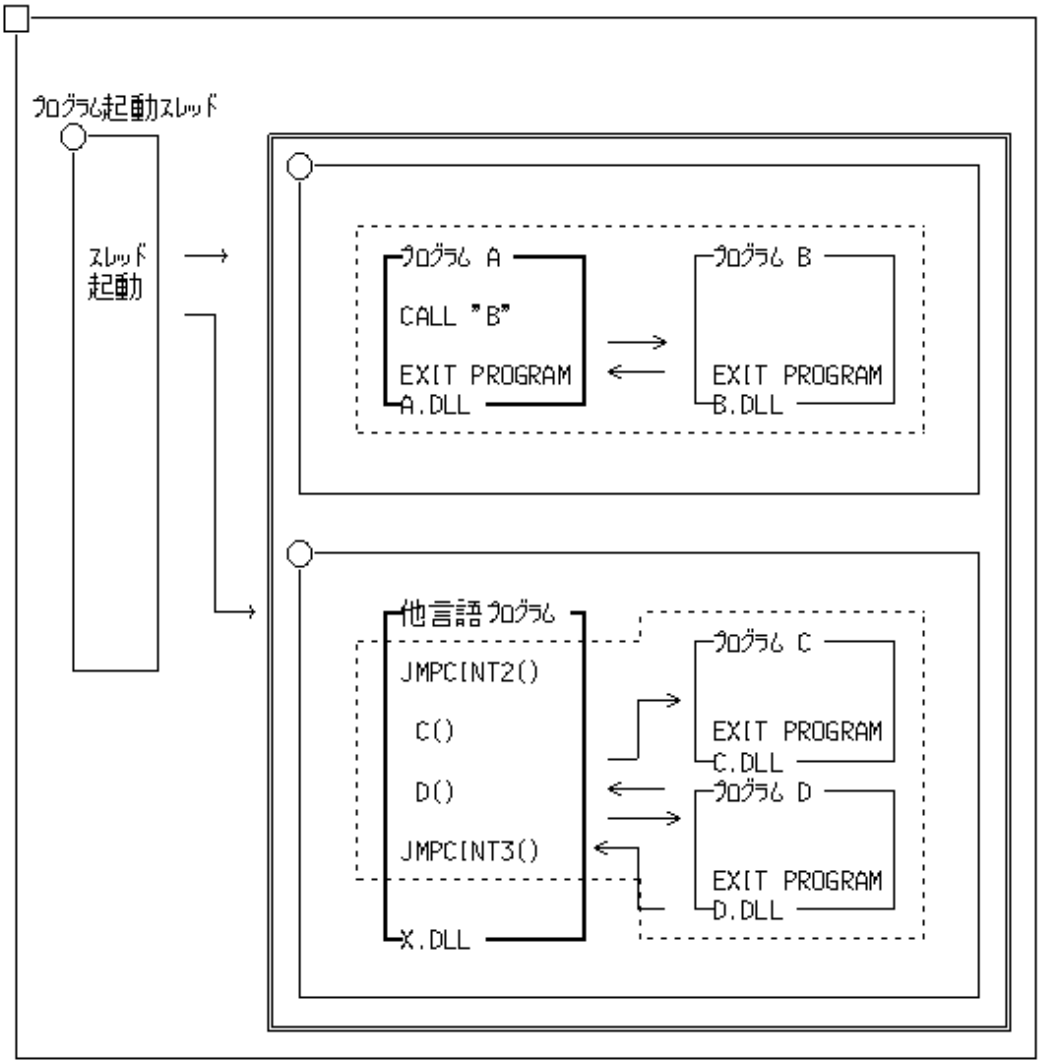


：COBOL の主プログラムを示す。　：COBOL の実行単位を示す。

：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。

18.3.2 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。

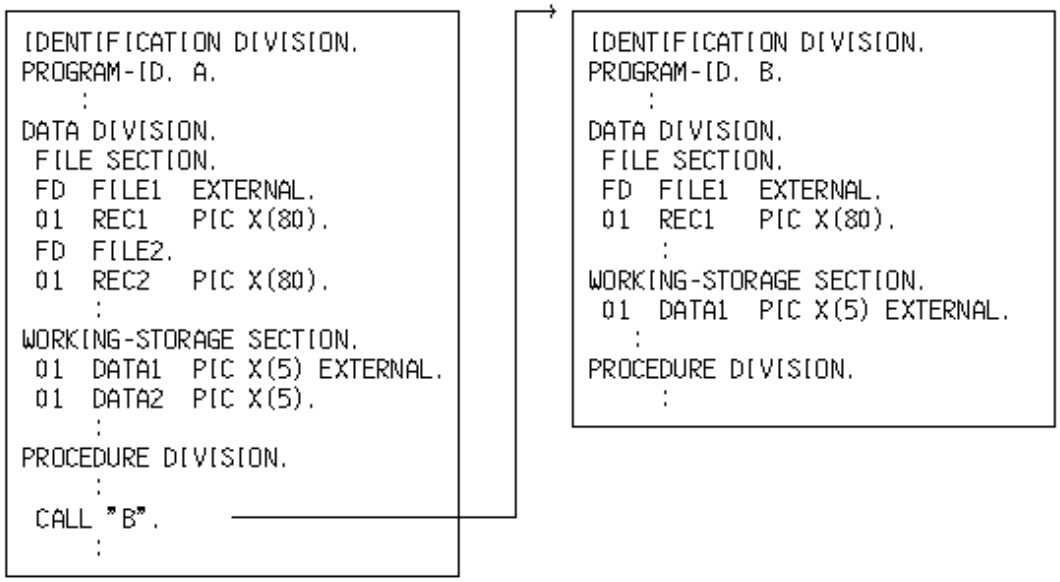
18.3.2.1 プログラム定義に宣言されたデータ

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実行単位が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18.1 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18.2 シングルスレッドプログラム

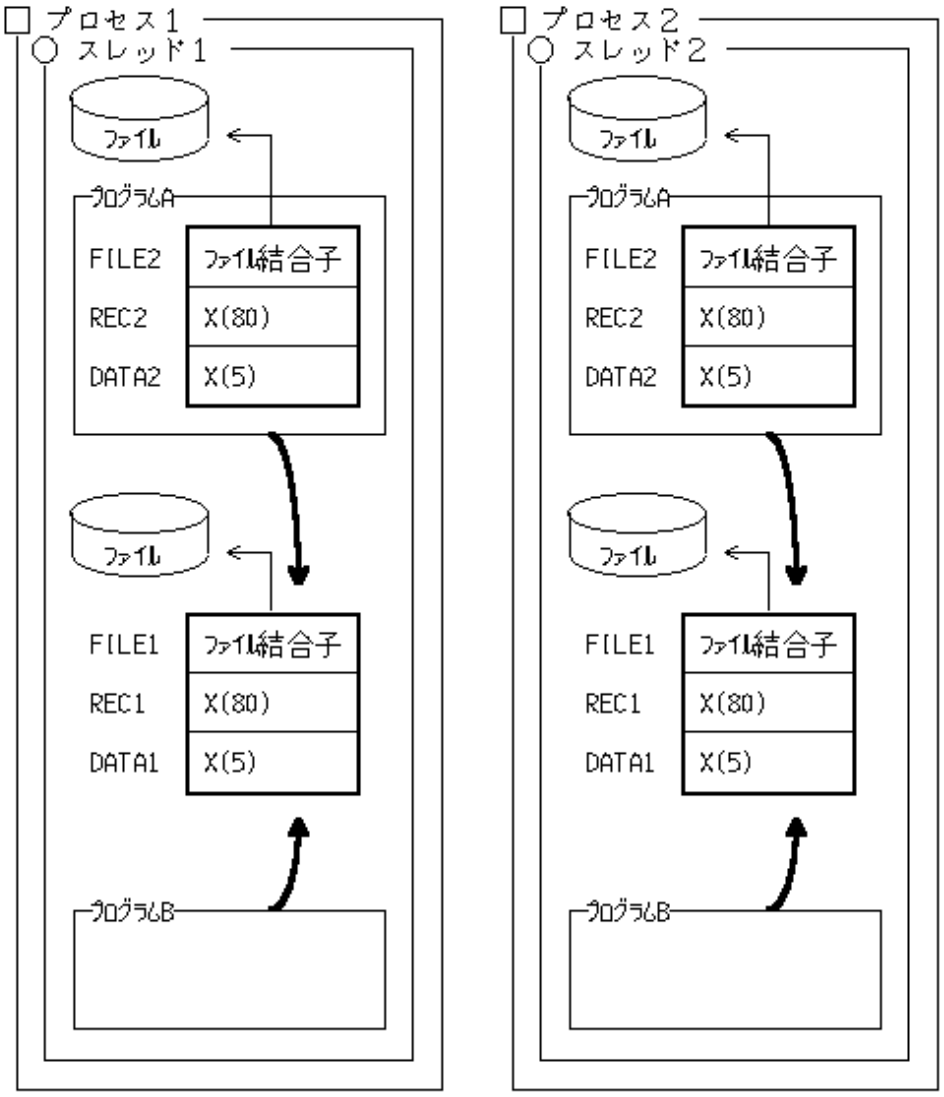
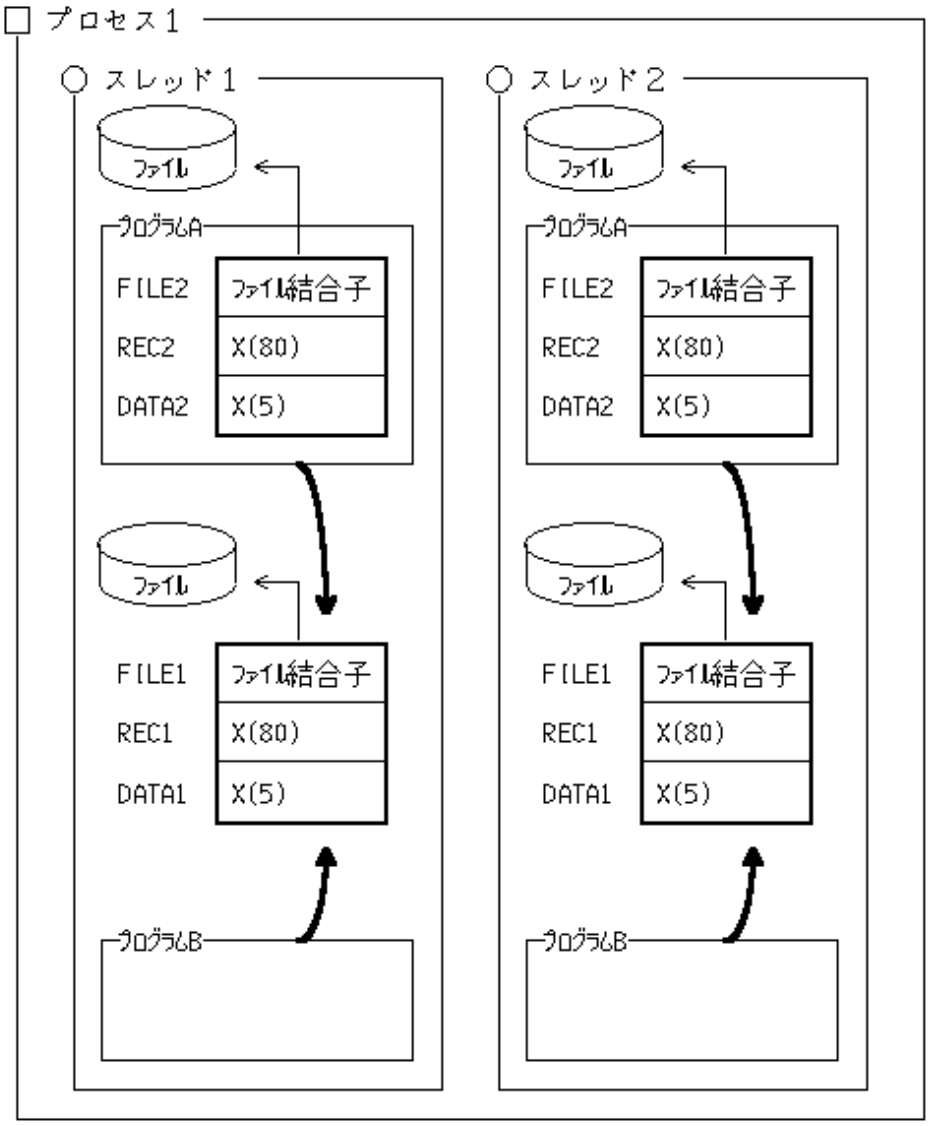


図18.3 マルチスレッドプログラム



18.3.2.2 ファクトリオブジェクトとオブジェクトインスタンス

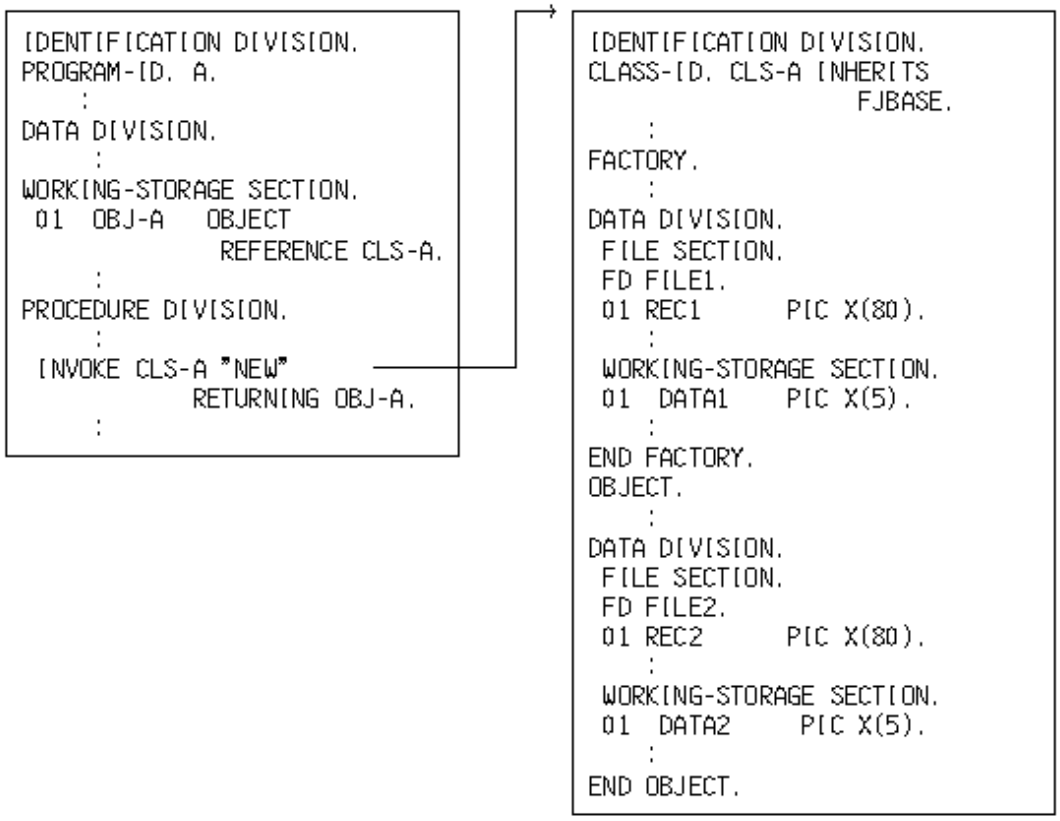
ファクトリオブジェクトとオブジェクトインスタンスはプロセスで管理されます。

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]“18.4.2.2 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]“18.4.2.3 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



```
IDENTIFICATION DIVISION.
PROGRAM-ID. A.
    :
DATA DIVISION.
     :
WORKING-STORAGE SECTION.
 01  OBJ-A   OBJECT
              REFERENCE CLS-A.
     :
PROCEDURE DIVISION.
     :
 INVOKE CLS-A "NEW"
            RETURNING OBJ-A.
     :
```

```
IDENTIFICATION DIVISION.
CLASS-ID. CLS-A INHERITS
                    FJBASE.
    :
FACTORY.
    :
DATA DIVISION.
 FILE SECTION.
 FD FILE1.
 01 REC1      PIC X(80).
    :
 WORKING-STORAGE SECTION.
 01  DATA1    PIC X(5).
    :
END FACTORY.
OBJECT.
    :
DATA DIVISION.
 FILE SECTION.
 FD FILE2.
 01 REC2      PIC X(80).
    :
 WORKING-STORAGE SECTION.
 01  DATA2    PIC X(5).
    :
END OBJECT.
```

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18.4 シングルスレッドプログラム

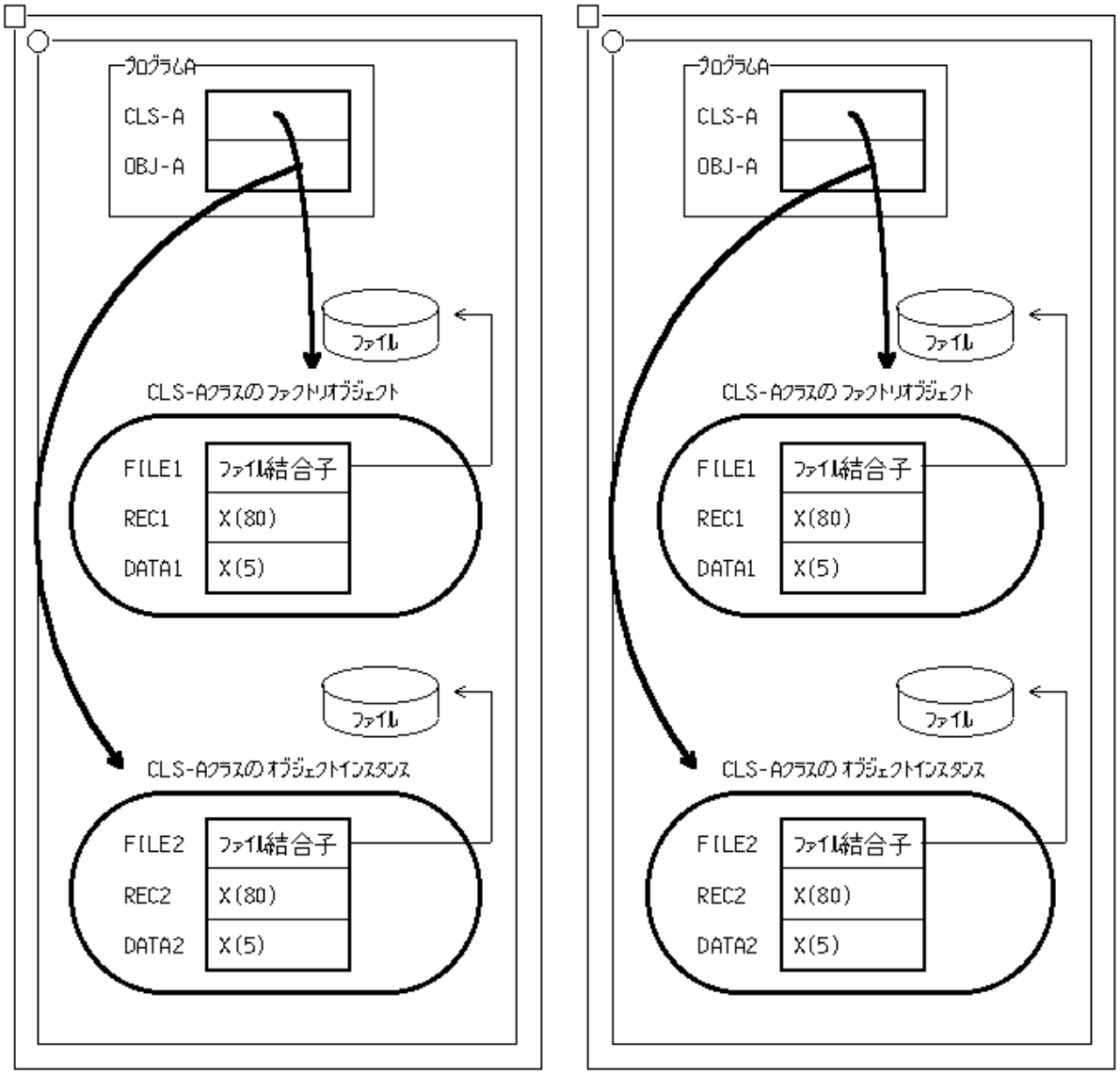
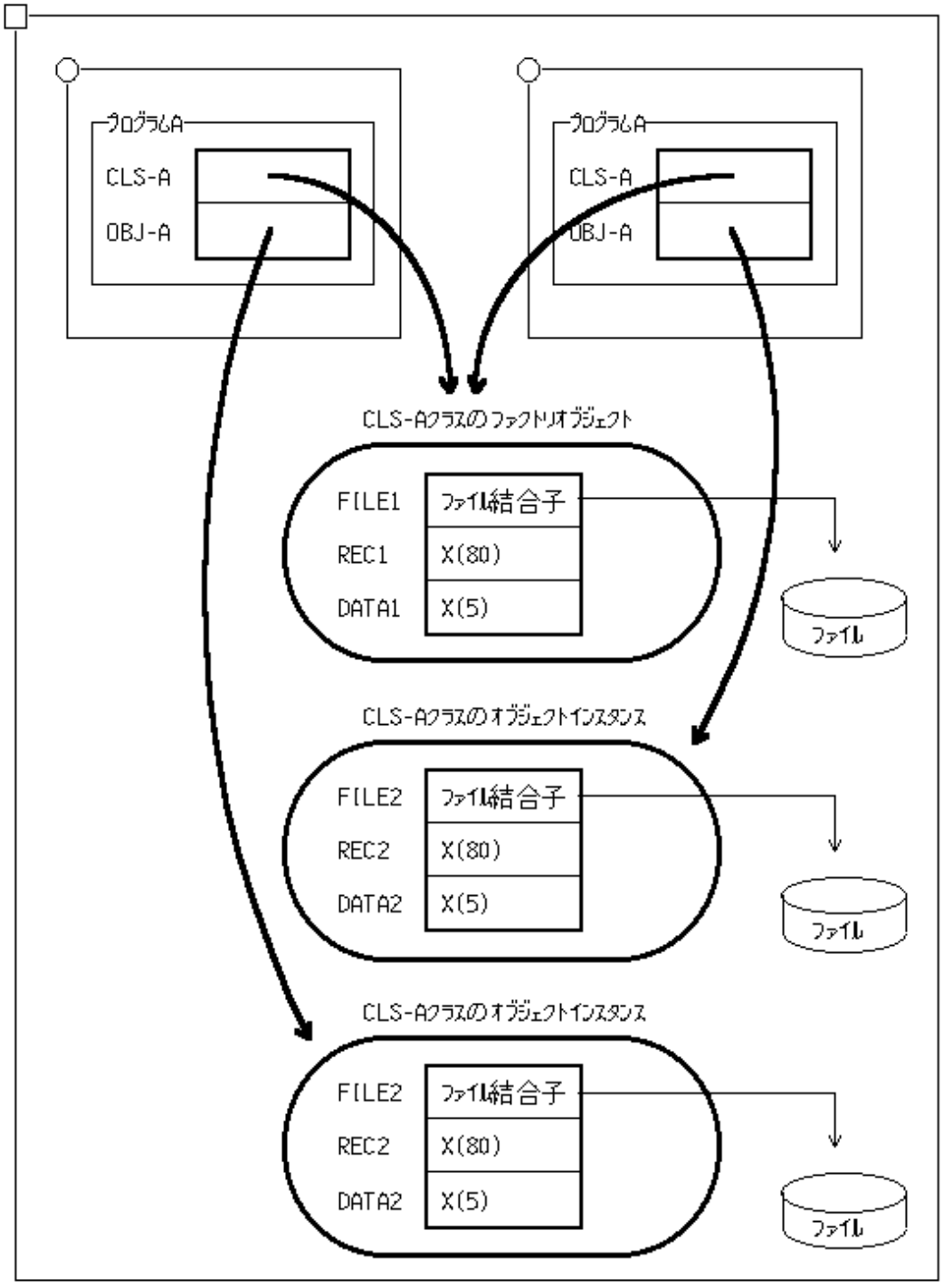


図18.5 マルチスレッドプログラム



18.3.2.3 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、そのメソッドの呼出し元に戻るときに解放されます。メソッドの呼出し元に戻るとき、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。

18.3.2.4 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。

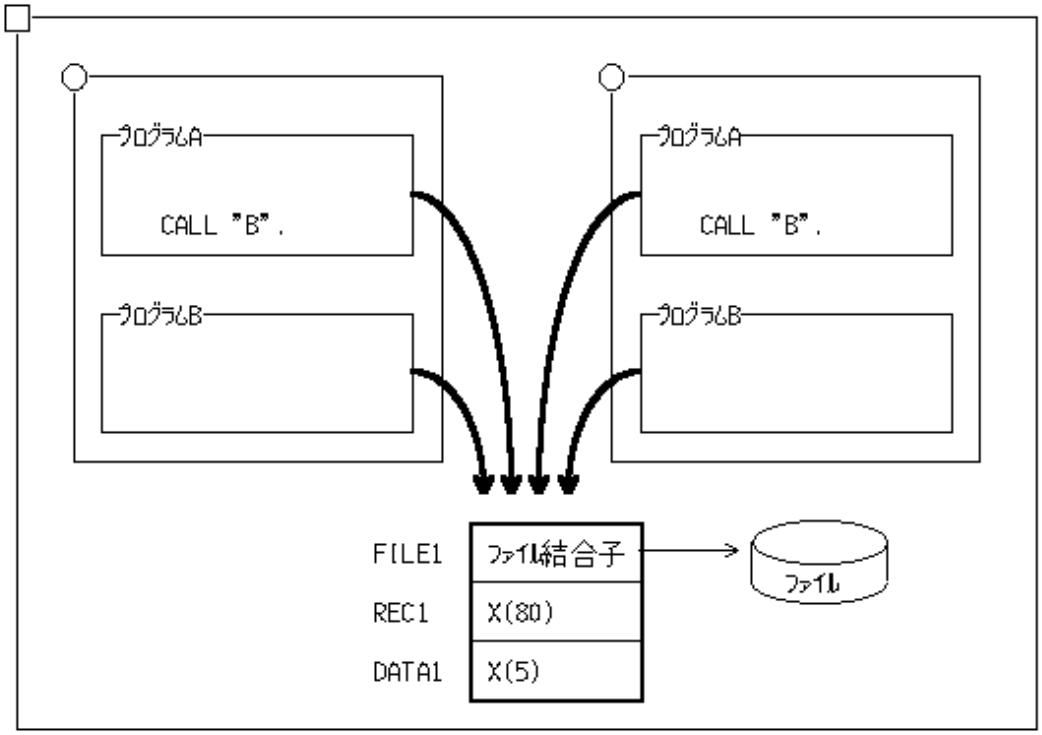
以下に、データとファイルをスレッド間で共有した場合のデータ領域の持ち方を示します。この図は、マルチスレッドプログラムが2つのスレッドで実行されているところを表しています。

翻訳オプションSHREXT指定

```
IDENTIFICATION DIVISION.
PROGRAM-ID. A.
     :
DATA DIVISION.
 FILE SECTION.
 FD  FILE1 EXTERNAL.
 01  REC1  PIC X(80).
     :
WORKING-STORAGE SECTION.
 01  DATA1 PIC X(5) EXTERNAL.
     :
PROCEDURE DIVISION.
     :
 CALL "B".
     :
```

翻訳オプションSHREXT指定

```
IDENTIFICATION DIVISION.
PROGRAM-ID. B.
     :
DATA DIVISION.
 FILE SECTION.
 FD  FILE1 EXTERNAL.
 01  REC1  PIC X(80).
     :
WORKING-STORAGE SECTION.
 01  DATA1 PIC X(5) EXTERNAL.
     :
PROCEDURE DIVISION.
     :
```



18.3.3 プログラムの実行とスレッドモード

COBOLプログラムが動作するプロセスは、スレッドモードを持ちます。スレッドモードには、シングルスレッドモードとマルチスレッドモードがあります。シングルスレッドモードのときは、プロセス内の1つのスレッドだけがCOBOLプログラムを実行できます。マルチスレッドモードのときは、プロセス内の複数のスレッドがCOBOLプログラムを実行することができます。このため、マルチスレッドモードでは、翻訳オプションTHREAD(SINGLE)で翻訳したプログラムを実行することはできません。

スレッドモードはプロセスで最初に実行されるCOBOLプログラムによって決まります。そのプログラムが翻訳オプションTHREAD(SINGLE)で翻訳したプログラムならシングルスレッドモードになり、翻訳オプションTHREAD(MULTI)で翻訳したプログラムならマルチスレッドモードになります。

マルチスレッドでのプログラムの実行

マルチスレッドでプログラムを実行させる場合は、実行環境内のすべてのプログラムを翻訳オプションTHREAD(MULTI)で翻訳してください。

シングル/マルチの共通部品の作成

翻訳オプションTHREAD(MULTI)で翻訳したプログラムはシングルスレッドモードでも動作できます。このため、翻訳オプションTHREAD(MULTI)で翻訳することにより、シングルスレッドでもマルチスレッドでも動作する共通部品を作成することができます。ただし、翻訳オプションTHREAD(MULTI)で翻訳されたプログラムをシングルスレッドで実行すると、翻訳オプションTHREAD(SINGLE)で翻訳されたプログラムよりも、実行性能が劣化します。また、後述の“注意事項”を必ずお読みください。

シングルスレッドモードでの強制実行

プロセスで最初に実行されるCOBOLプログラムが、翻訳オプションTHREAD(MULTI)で翻訳したプログラムでも、環境変数情報@CBR_THREAD_MODE=SINGLEを指定することによって強制的にシングルスレッドモードで実行させることができます。指定方法については、“C.2.52 @CBR_THREAD_MODE (スレッドモードの指定)”を参照してください。

注意事項

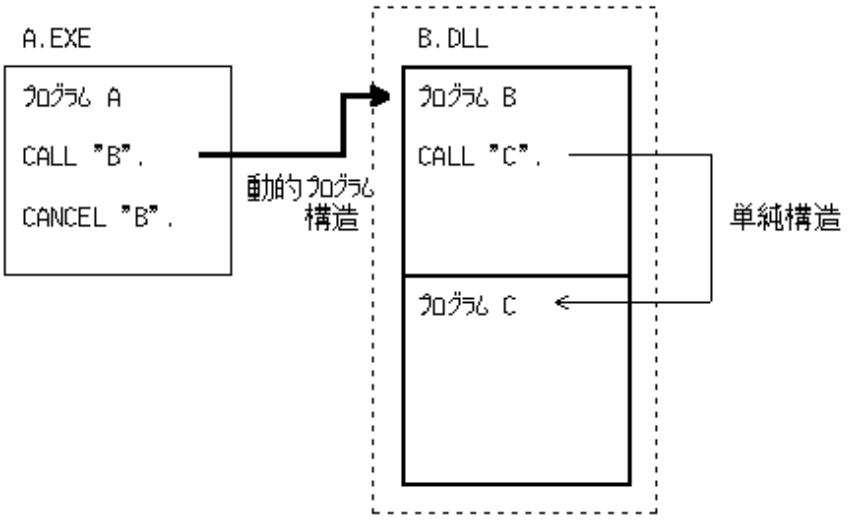
マルチスレッドプログラムをシングルスレッドモードで実行する場合、このプログラムをCANCEL文で削除するには注意が必要です。以下に例を示します。

例

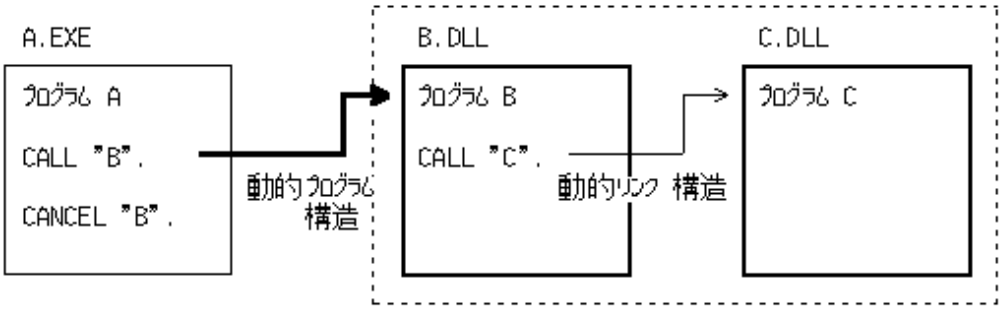
プログラムBとプログラムCを翻訳オプションTHREAD(MULTI)で翻訳します。

プログラムBとプログラムCの結合を単純構造(下図(a))または動的リンク構造(下図(b))とした場合、「CANCEL ”B”」の実行により、プログラムBが仮想メモリから削除されると、プログラムCも削除されるため、このCANCEL文は使用できません。この場合、プログラムBとプログラムCの結合を動的プログラム構造(下図(c))に変更することにより、CANCEL文を使用できるようになります。

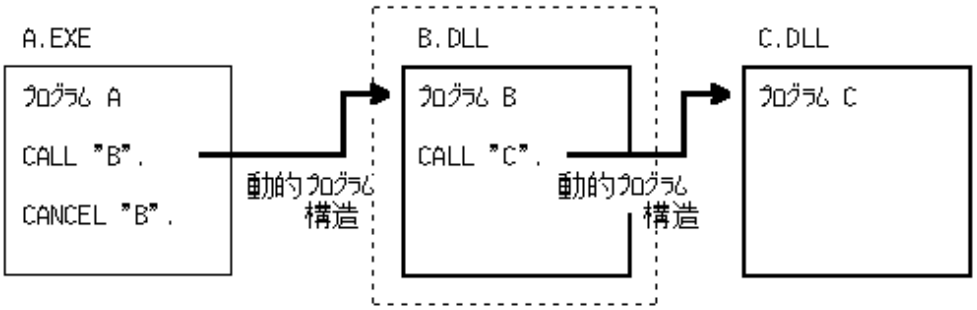
(a) プログラムＢとプログラムＣが単純構造



(b) プログラムＢとプログラムＣが動的リンク構造



(c) プログラムＢとプログラムＣが動的プログラム構造




：翻訳オプション THREAD(MULTI) の有効なプログラム


：「CANCEL "B"」の実行により仮想メモリから削除されるプログラム

18.4 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。

18.4.1 競合状態

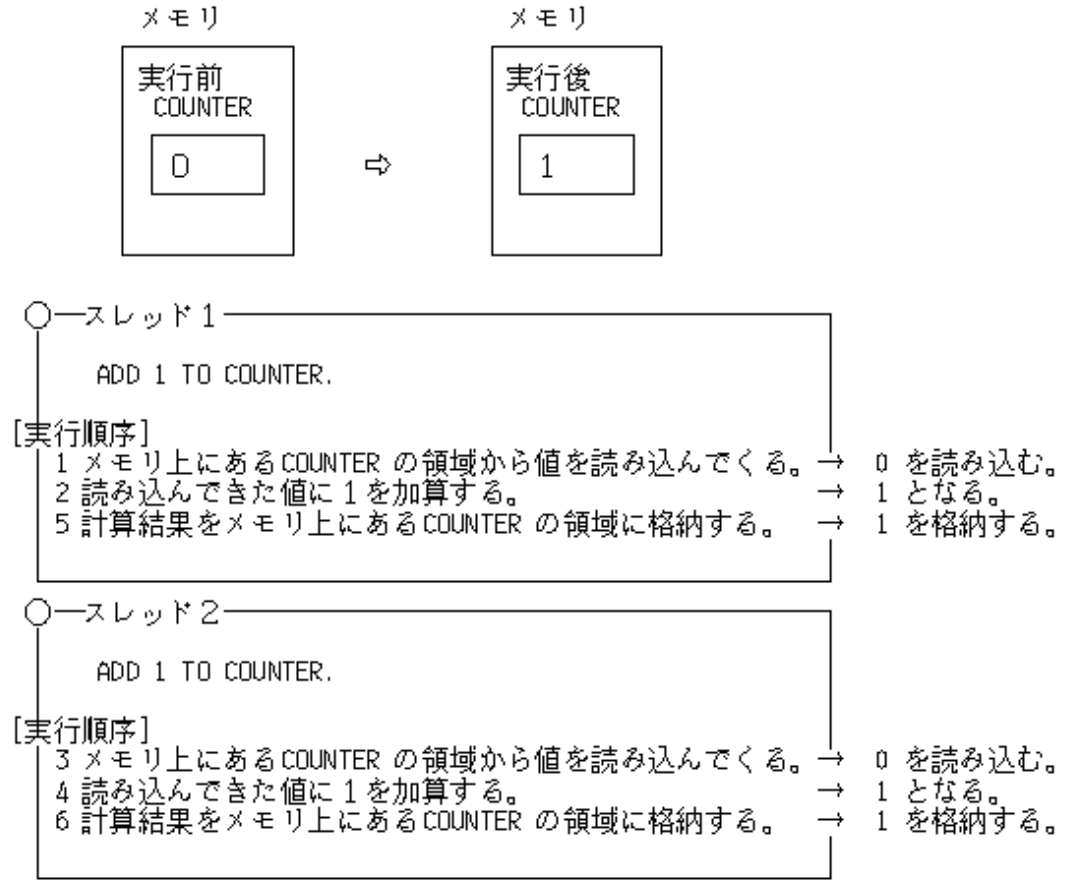
ここでは、スレッド間で資源を共有する場合に、一般的に発生する競合状態について説明します。

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。



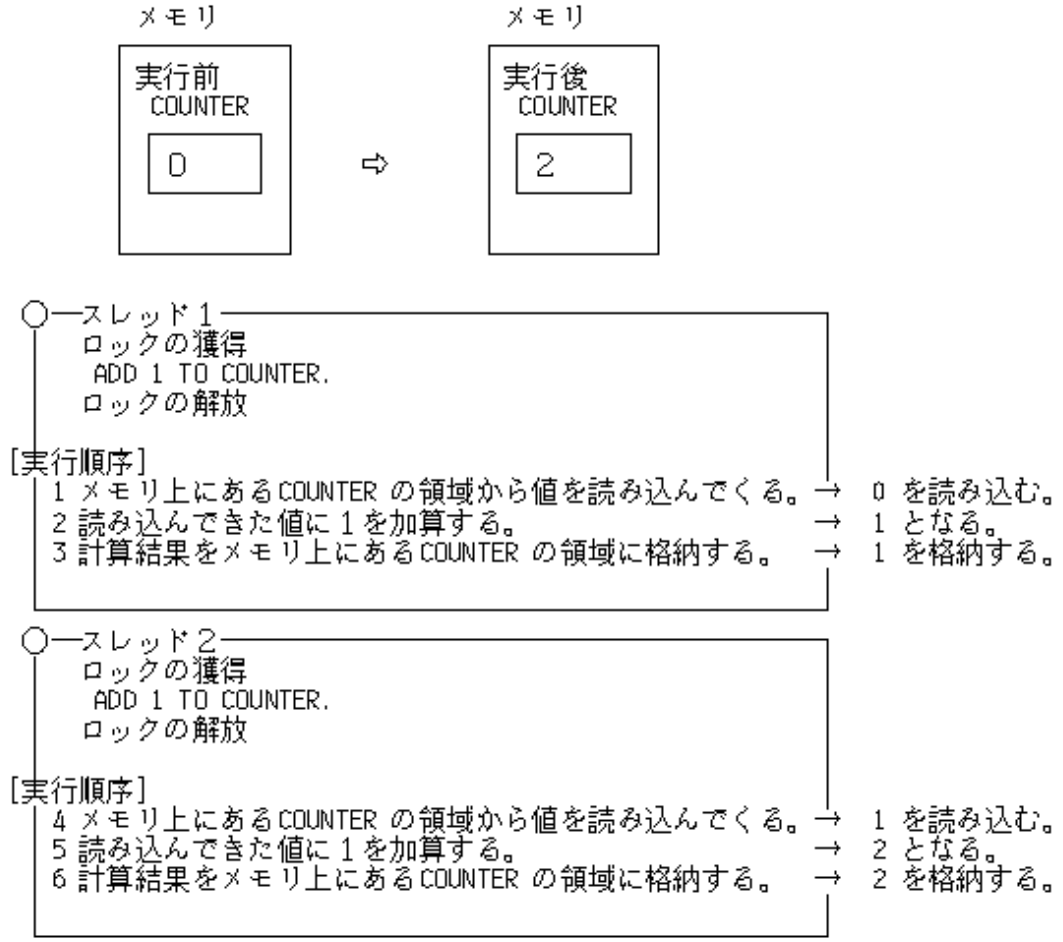
このような競合状態が発生するのは、次の条件のときです。もちろん、同一データをすべてのスレッドが参照するだけなら、同時にデータをアクセスしても問題ありません。

- 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)。



18.4.2 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。

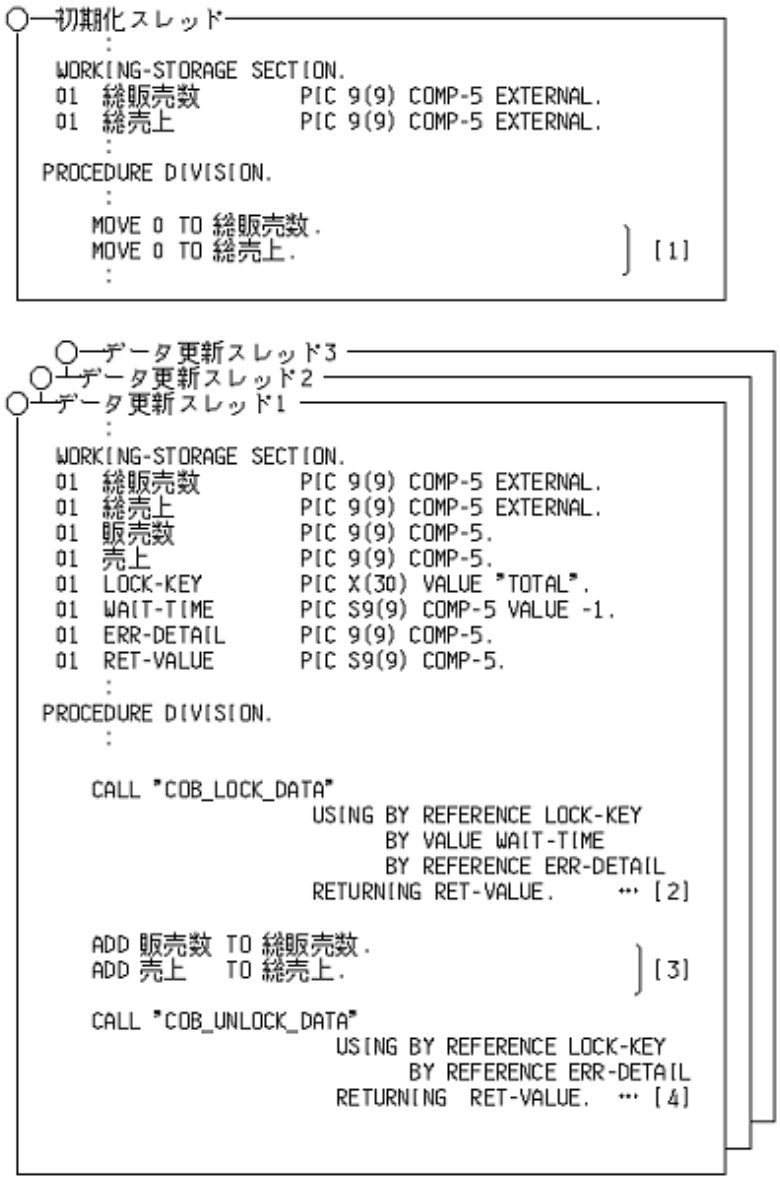
- スレッド間共有外部データとスレッド間共有外部ファイル
- ファクトリオブジェクト
- オブジェクトインスタンス

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、“18.9 スレッド同期制御サブルーチン”を参照してください。

18.4.2.1 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、“18.6.1.1 スレッド間共有外部ファイル”を参照してください。



[1] 初期化スレッドは、実行環境で最初に1回だけ起動されます。初期化スレッドで、共有データを初期化しています。

[2] 共有データを複数のスレッドで同時に更新しないようにするため、データロックサブルーチンを使用して、”TOTAL”というデータ名に対応するロックキーに対してロックを獲得しています。データロックサブルーチンについては、“I.1.8 データロック獲得サブルーチン(COB_LOCK_DATA)”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。

18.4.2.2 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、“I.1.10 オブジェクトロック獲得サブルーチン(COB_LOCK_OBJECT)”を参照してください。

[2] EMPLOYEEクラスのファクトリオブジェクトのプロパティメソッドを呼び出し、保険額をファクトリデータに設定しています。

[3] EMPLOYEEクラスのファクトリオブジェクトのプロパティメソッドを呼び出し、契約年をファクトリデータに設定しています。

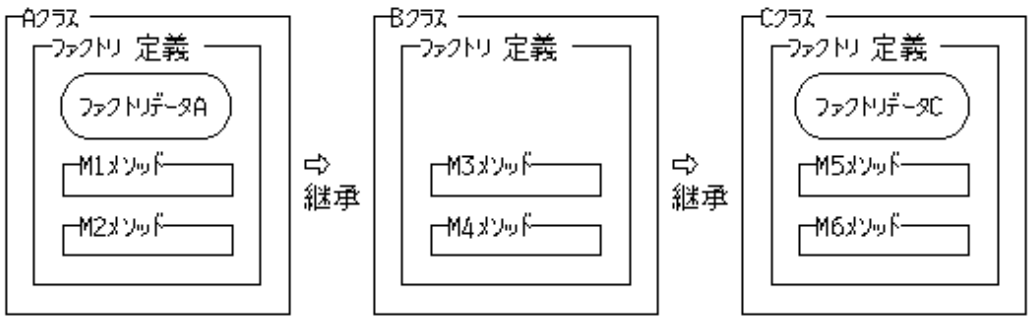
[4] EMPLOYEEクラスのファクトリオブジェクトの合計メソッドを呼び出し、金額を求めています。[2]～[4]までの処理は、[1]でロックを獲得したスレッドにより実行されます。

[5] ファクトリオブジェクトのロックを解放しています。ロックの解放により、別のスレッドで、[1]と同様の処理でロックを獲得できます。

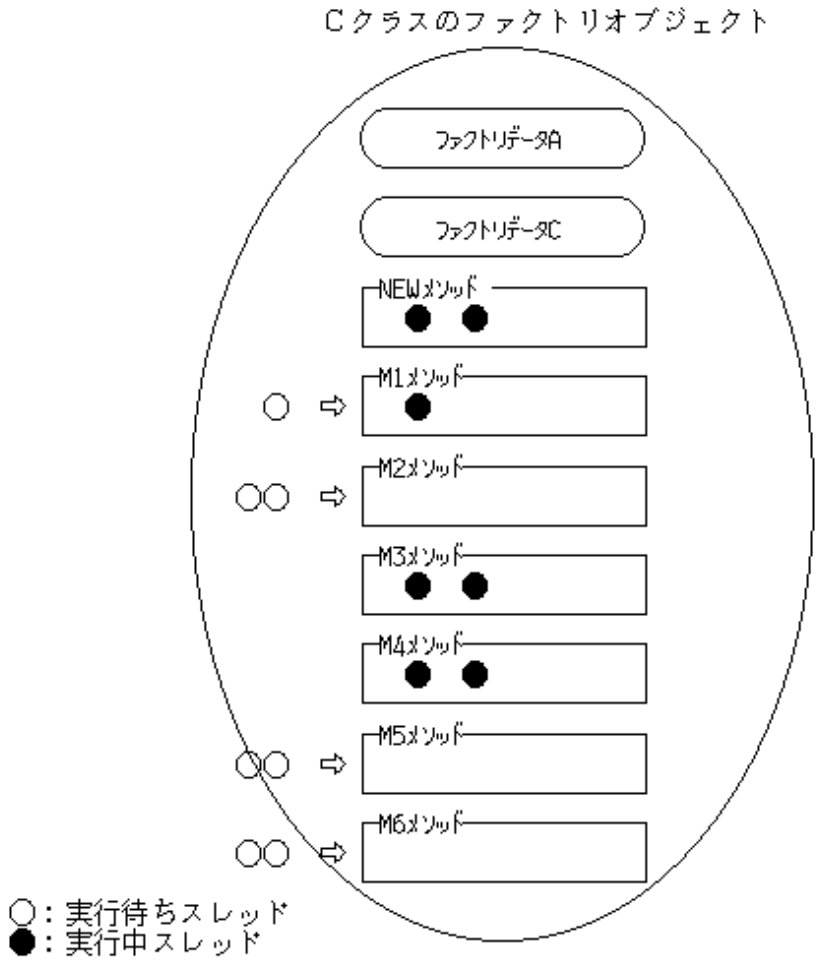
COBOLランタイムシステムによる同期制御

ここでは、ファクトリデータに対してCOBOLランタイムシステムが自動的に行っているスレッドの同期制御の動作について説明します。

COBOLランタイムシステムは、ファクトリデータが明示定義されているクラスのファクトリメソッドが、同一ファクトリオブジェクト上で、同時に1つのスレッドしか実行されないように制御します。この動作を下図のような継承関係にあるCクラスを例にとって動作を説明します。



Cクラスのファクトリオブジェクトは、AクラスとCクラスで明示定義されたファクトリデータを持ちます。このため、Aクラスで明示定義されているメソッド(M1メソッドとM2メソッド)とCクラスで明示定義されているメソッド(M5メソッドとM6メソッド)は、同時に1つのスレッドでしか実行されません。そのほかのメソッドは、同時に複数のスレッドで実行されます。



上の例は、1つのスレッドがM1メソッドを実行しています。このため、M1メソッドを実行する別スレッド、M2メソッド、M5メソッドおよびM6メソッドを実行するスレッドは、すべて実行待ち状態となります。ほかのメソッドは、同時に複数のスレッドで実行されます。このように、ファクトリデータのアクセスはCOBOLランタイムシステムによって自動的に同期制御されるため、メソッドの1回の呼び出しで処理が完結するような場合は、プログラムでスレッドの同期制御を行う必要はありません。

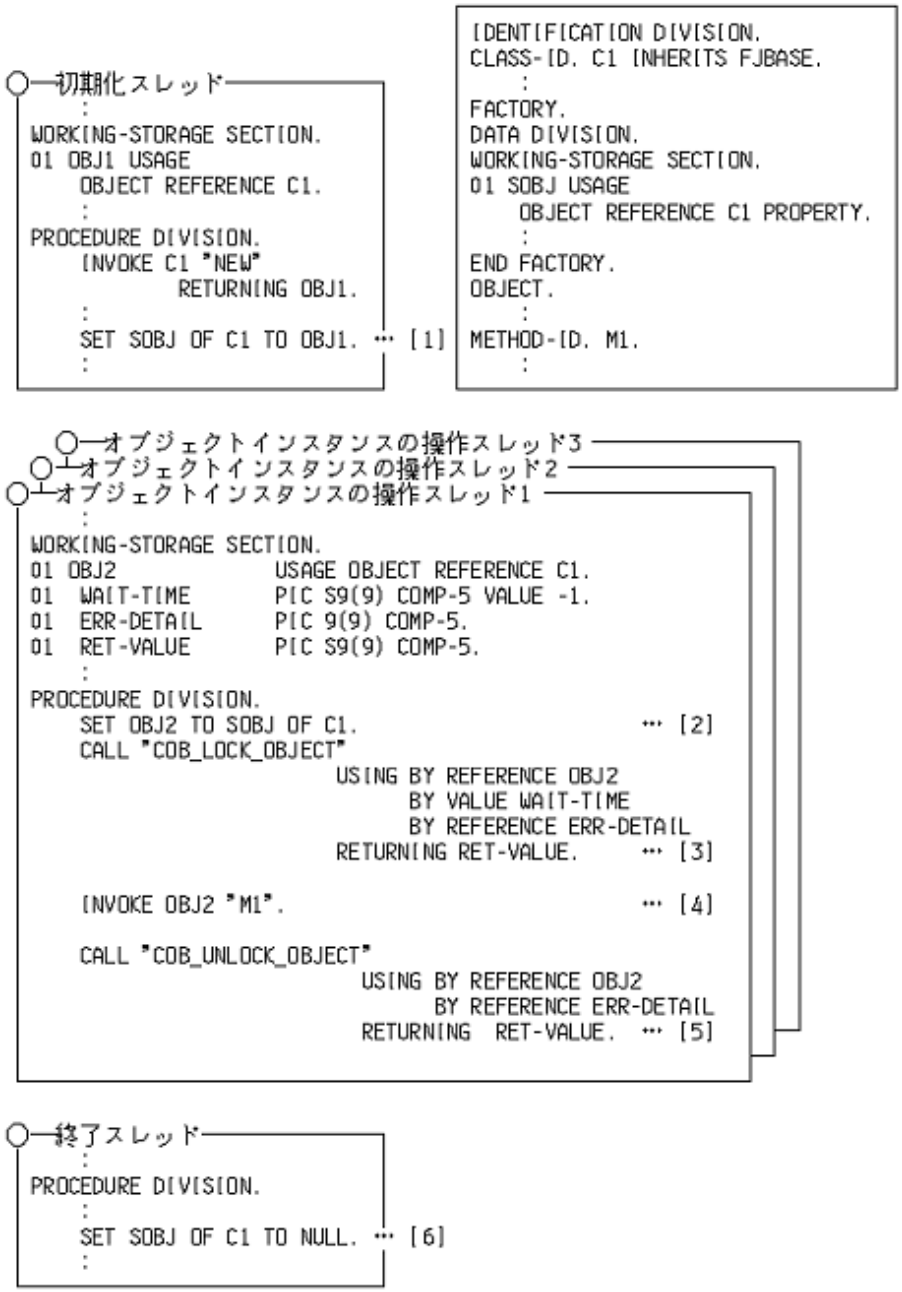
参考

上記の同期制御は、ファクトリオブジェクト単位で行われます。したがって、継承しているクラスのファクトリオブジェクトでどのメソッドが実行されていようと、自クラスのファクトリオブジェクトの同期制御には影響しません。

18.4.2.3 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、“I.1.10 オブジェクトロック獲得サブルーチン(COB_LOCK_OBJECT)”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。

18.5 基本的な使い方

“18.3.2 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、“18.7.1 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、“18.4 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。

18.5.1 入出力機能の利用

ここでは、入出力機能を利用してファイル操作を行うマルチスレッドプログラムを開発する方法について説明します。

マルチスレッドプログラムへの移行

シングルスレッドプログラムをマルチスレッドプログラムに移行するには、プログラムを修正する必要はありません。翻訳オプションTHREAD(MULTI)を指定してプログラムを再翻訳し、リンクするだけで、そのままマルチスレッドプログラムとして利用することができます。

以降では、同一ファイルを共有するプログラムについて説明します。

また、同一プログラムを実行し、スレッドごと、別々のファイルを操作する方法について説明します。

同一ファイルを共有する

外部媒体上のファイルとプログラムとの関係付けは、ファイル結合子を通して行われます。ファイル結合子には、内部属性と外部属性を持つ2つのファイル結合子があります。内部属性/外部属性に関係なく、スレッド間で異なるファイル結合子を操作して、ファイルを共有することができます。

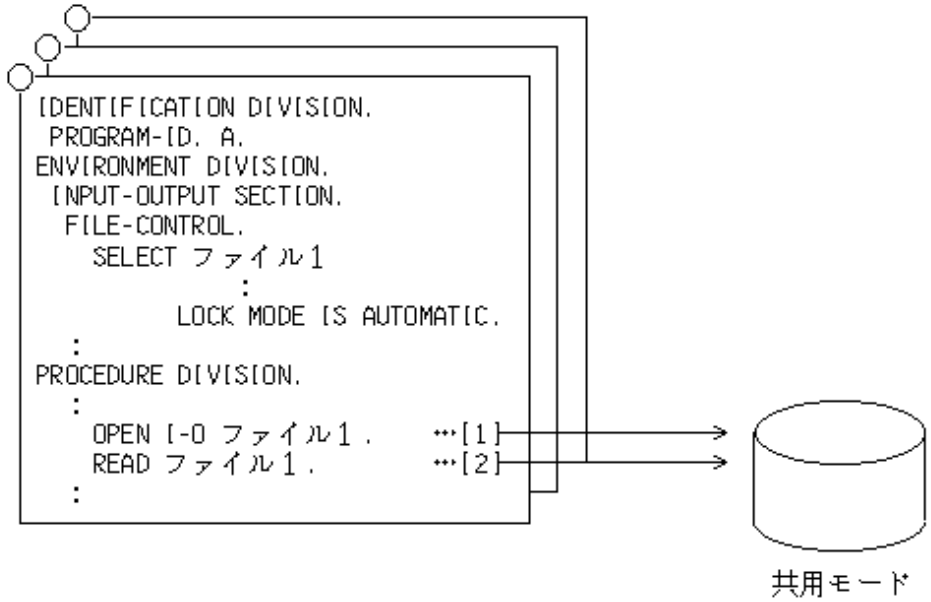
ここでは、内部属性を持つファイル結合子を操作して同一ファイルを共有する方法について説明します。

プログラム内に定義したファイル/メソッド内に定義したファイル/オブジェクト内に定義したファイル

プログラム内/メソッド内/オブジェクト内のファイル記述項に定義したファイルが内部属性を持つファイル結合子の場合、同一のファイルを割り当てることにより、スレッド間でファイルを共有することができます。

オブジェクト内のファイル記述項に定義したファイルの場合、別々のオブジェクトインスタンスのオブジェクト参照子を操作することで、プログラム内に定義したファイルと同様にファイルを共有することができます。

以下に、プログラム内に定義したファイルの共有処理を示します。



[1] 共用モードでファイルを開きます。

[2] レコードを読み込みます。

シングルスレッドプログラムと同様に、同一ファイルを操作する場合は、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。

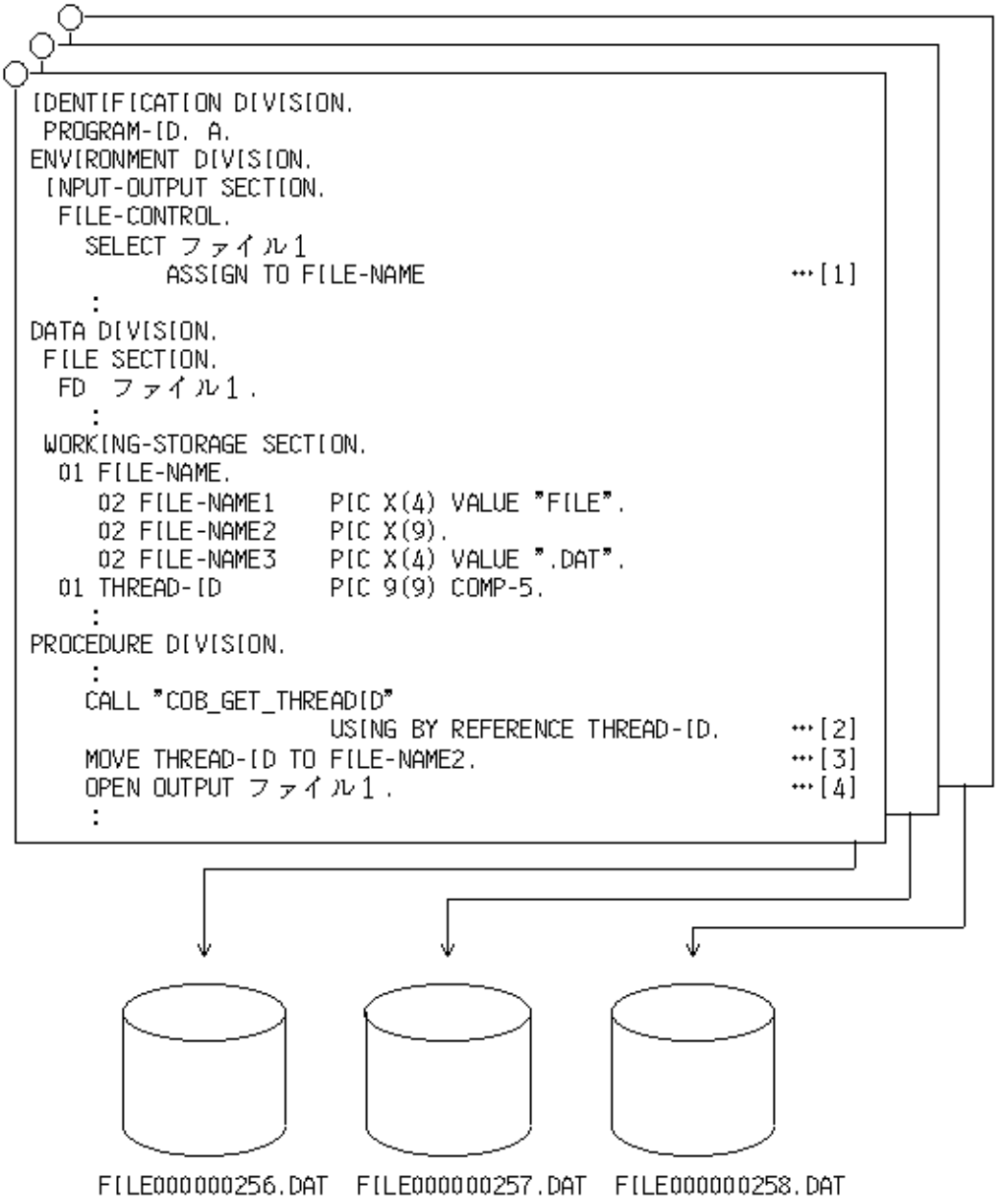
注意

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、“18.6 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、別々のファイルを操作する

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を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、別々のファイルを操作する方法について説明します。



[1] ASSIGN句にデータ名を記述する。

[2] スレッド取得サブルーチンを呼び出し、スレッドIDを取得する。

[3] [2]で取得したスレッドIDをデータ名に設定する。

[4] OPEN文(OUTPUTモード)を実行し、ファイルを創成する。

上記のように、スレッド取得サブルーチンを使用して取得したスレッドIDをファイル名とすることで、同一プログラムを実行して、スレッドごと、別々のファイルを操作することができます。スレッド取得サブルーチンについては、“I.1.3 スレッドID取得サブルーチン(COB_GET_THREADID)”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.2.8 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。

18.5.2 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。

18.5.3 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(セションの作成、破棄、開始、終了)を使用する必要があります。SQL拡張インタフェースの詳細については、“Symfoware Server RDBユーザーズガイド 応用プログラム開発編”および“Symfoware Server SQLリファレンスガイド”を参照してください。

プログラムの翻訳・リンク

sqlcobolコマンドにより翻訳・リンクを行う場合、オプション-Tを指定してください。

注意

プリコンパイル・翻訳とリンクを別に行う場合は、以下を行ってください。

- プリコンパイルオプション-Tを指定ください。
- 翻訳・リンクについては“18.7 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)”

注意

プリコンパイラを利用したSymfoware連携を行うマルチスレッドプログラム開発時の注意事項について説明します。

- プリコンパイル時に、マルチスレッドオプションを指定して作成したアプリケーションは、シングルスレッドで運用することはできません。

18.5.4 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。

18.5.4.1 小入出力機能について

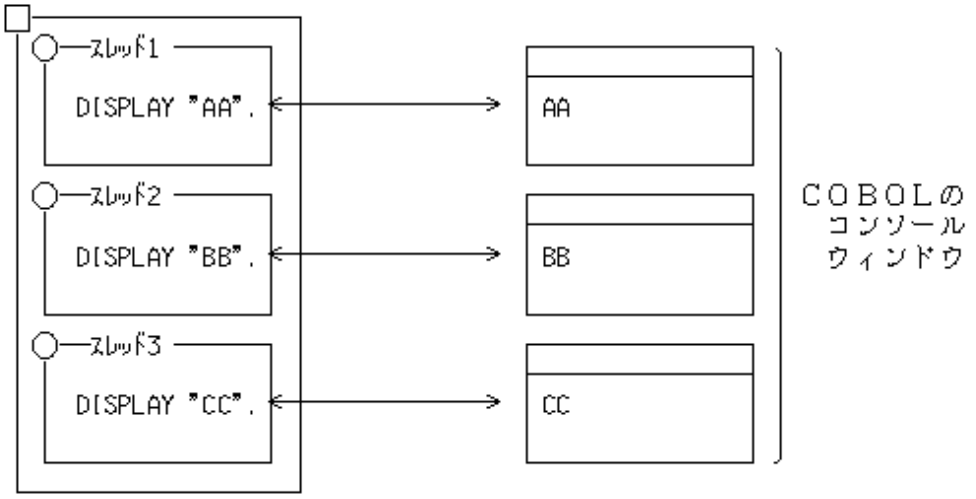
マルチスレッドモードでは、COBOLのコンソールウィンドウはスレッドごとに別々に表示されます。これに対して、システムのコンソールウィンドウ(コマンドプロンプトウィンドウを含む)およびファイルを使用した小入出力では、プロセスで1つのウィンドウおよびファイルを共有します。

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

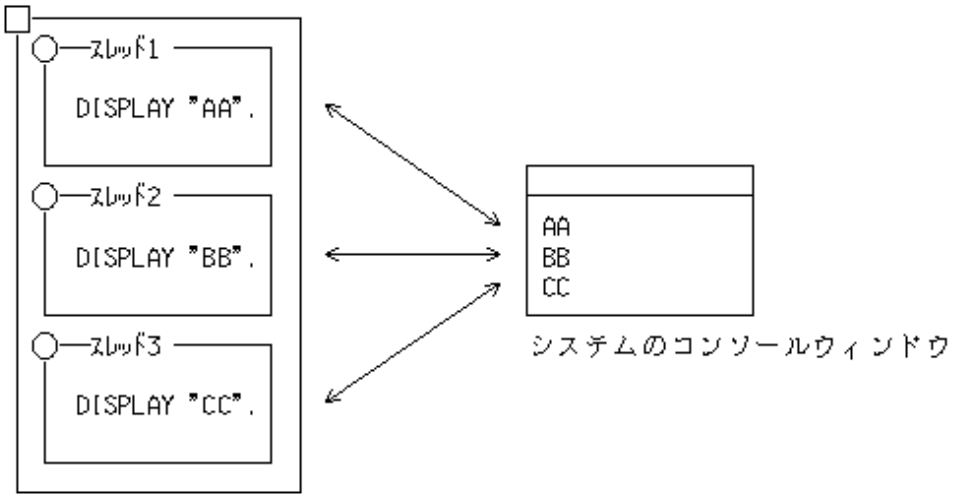
例

入出力の例

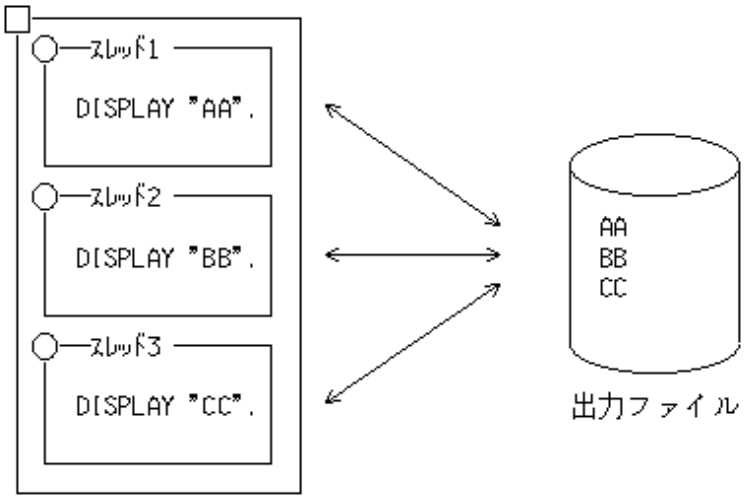
- COBOLのコンソールウィンドウを使用する場合



- システムのコンソールウィンドウを使用する場合



・ ファイルを使用する場合



システムのコンソールウィンドウおよびファイルを使用した小入出力を行う場合、DISPLAY文およびACCEPT文単位でデータの入出力は同期制御されます。

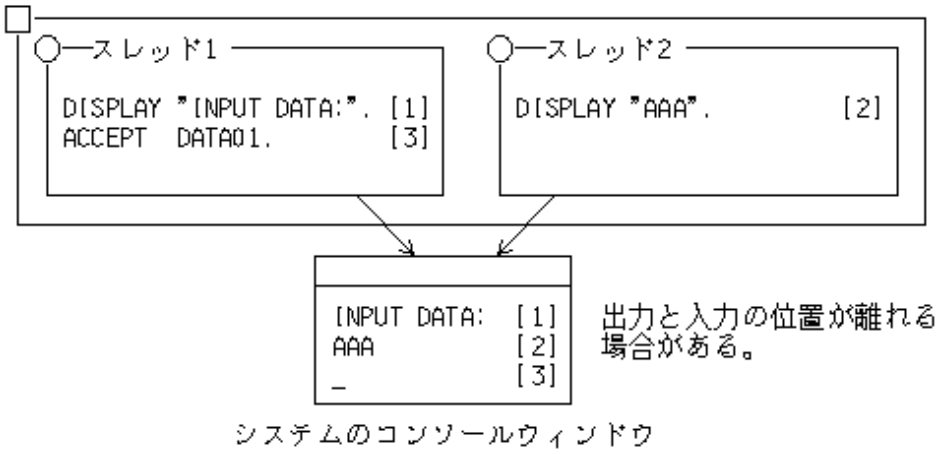
しかし、文単位での同期制御は行いますが、各文の実行順序については、システムのスレッド制御の順序に依存するため、結果が実行のたびに異なることがあります。

実行順序の同期制御を行いたい場合、たとえば、システムのコンソールウィンドウを使用する際に、DISPLAY文の直後にACCEPT文を実行したい場合は、スレッド同期制御サブルーチンを使用することで、DISPLAY文の出力の直後にACCEPT文を実行できます。[参照]“18.9 スレッド同期制御サブルーチン”

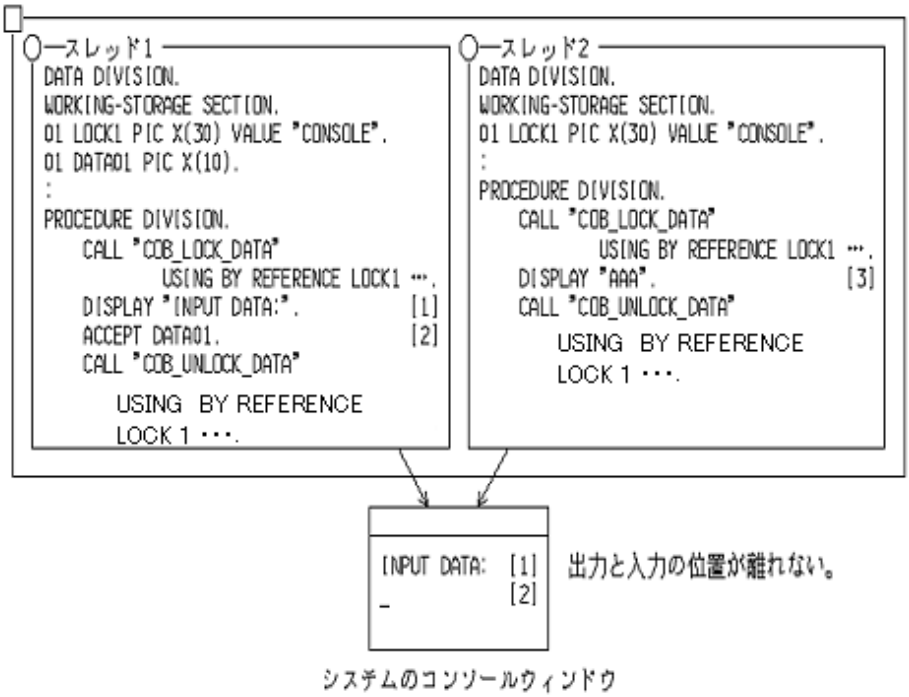
例

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。

18.5.4.2 コマンド行引数および環境変数の操作機能について

18.5.4.2.1 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。

18.5.4.2.2 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。

18.5.5 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。

18.6 少し進んだ使い方

18.6.1 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

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つのファイルをアクセスする場合、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。

18.6.1.1 スレッド間共有外部ファイル

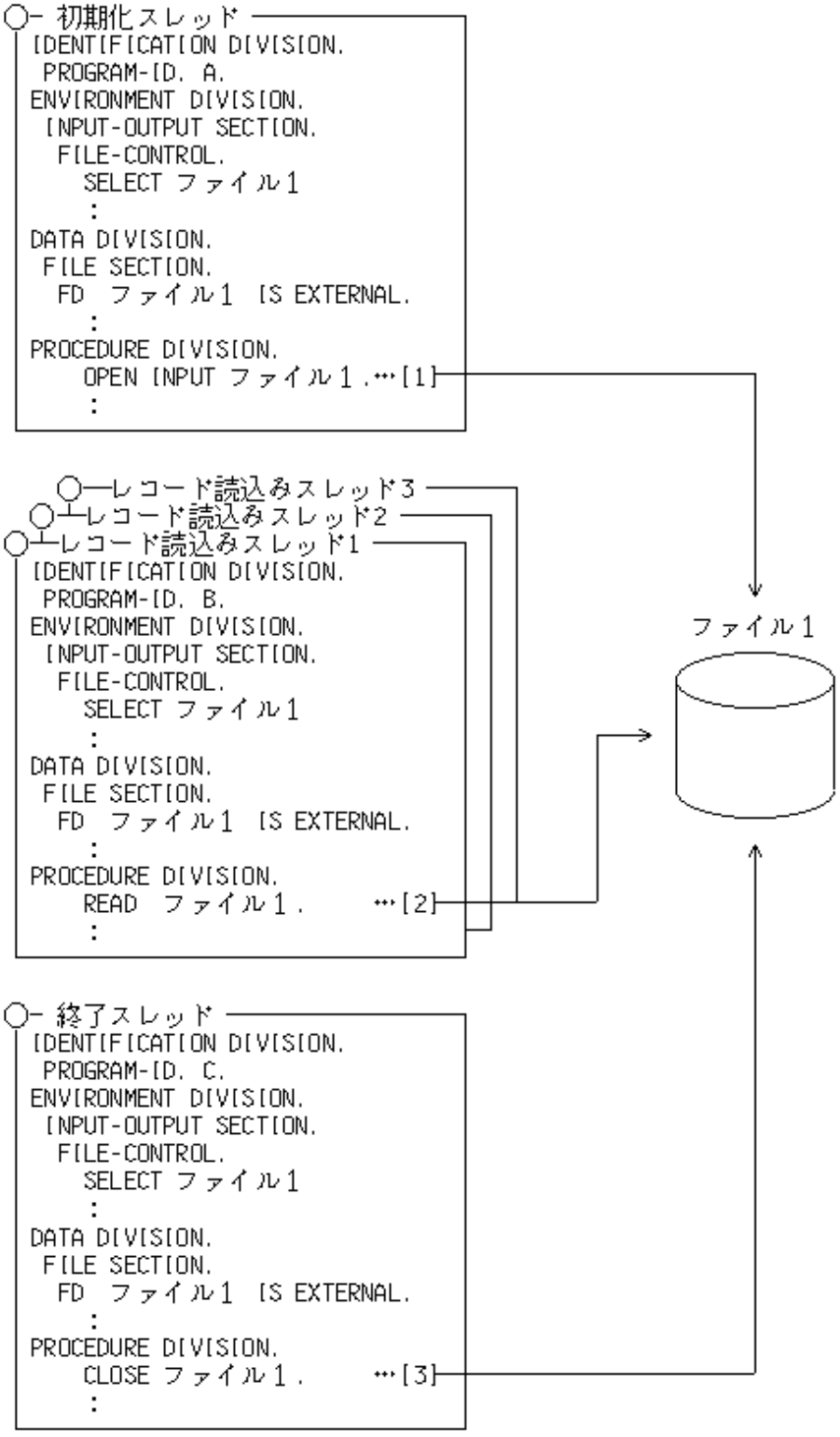
ファイル記述項にEXTERNAL句を指定した場合、ファイル結合子に外部属性が与えられます。外部属性を持つファイル結合子は、プログラム間で同じファイル結合子を共有することができます。

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。レコード読込みスレッドでは、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。

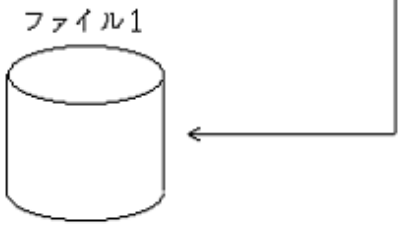
[3] 終了スレッドは、実行環境で最後に1回だけ起動されます。終了スレッドでは、初期化スレッドによって開かれているファイルを閉じます。

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

```
レコード更新スレッド3
レコード更新スレッド2
レコード更新スレッド1
IDENTIFICATION DIVISION.
 PROGRAM-ID. B.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT ファイル1
    :
DATA DIVISION.
 FILE SECTION.
  FD  ファイル1  IS EXTERNAL.
  01  レコード1  PIC X(80).
    :
 WORKING-STORAGE SECTION.
  01 LOCK-KEY  PIC X(30) VALUE "FILE".
  01 WAIT-TIME PIC S9(9) COMP-5 VALUE -1.
  01 ERROR-VAL PIC 9(9) COMP-5.
  01 RET-VAL   PIC S9(9) COMP-5.
    :
PROCEDURE DIVISION.
    :
    CALL "COB_LOCK_DATA"
                        USING BY REFERENCE LOCK-KEY
                              BY VALUE     WAIT-TIME
                              BY REFERENCE ERROR-VAL    …[1]
                        RETURNING RET-VAL.
    READ    ファイル1.                                  …[2]
    REWRITE レコード1.                                  …[3]
    CALL "COB_UNLOCK_DATA"
                          USING BY REFERENCE LOCK-KEY
                                BY REFERENCE ERROR-VAL  …[4]
                          RETURNING RET-VAL.
    :
```



[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

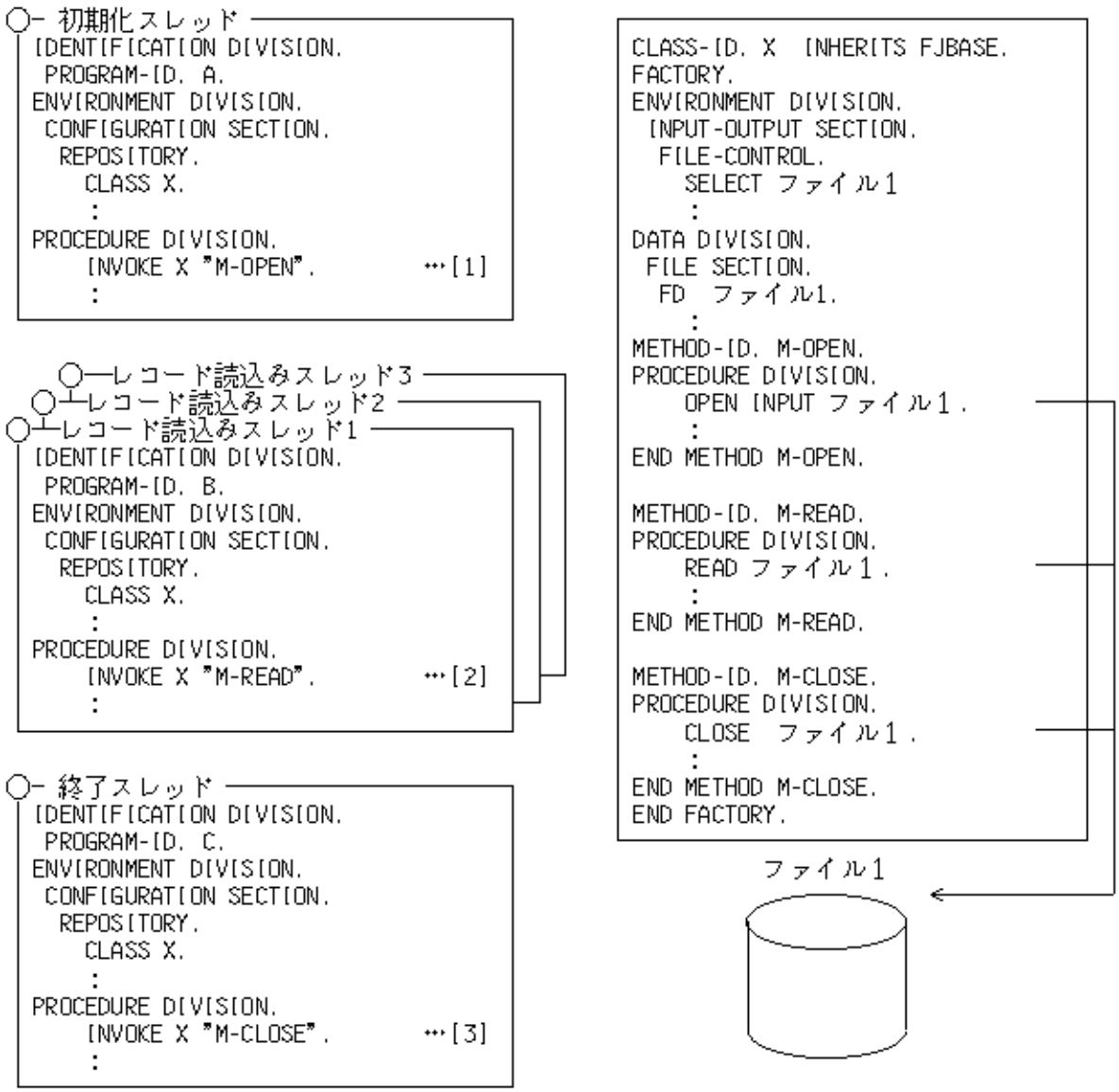
データロックサブルーチンについては、“I.1.8 データロック獲得サブルーチン(COB_LOCK_DATA)”を参照してください。

18.6.1.2 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。

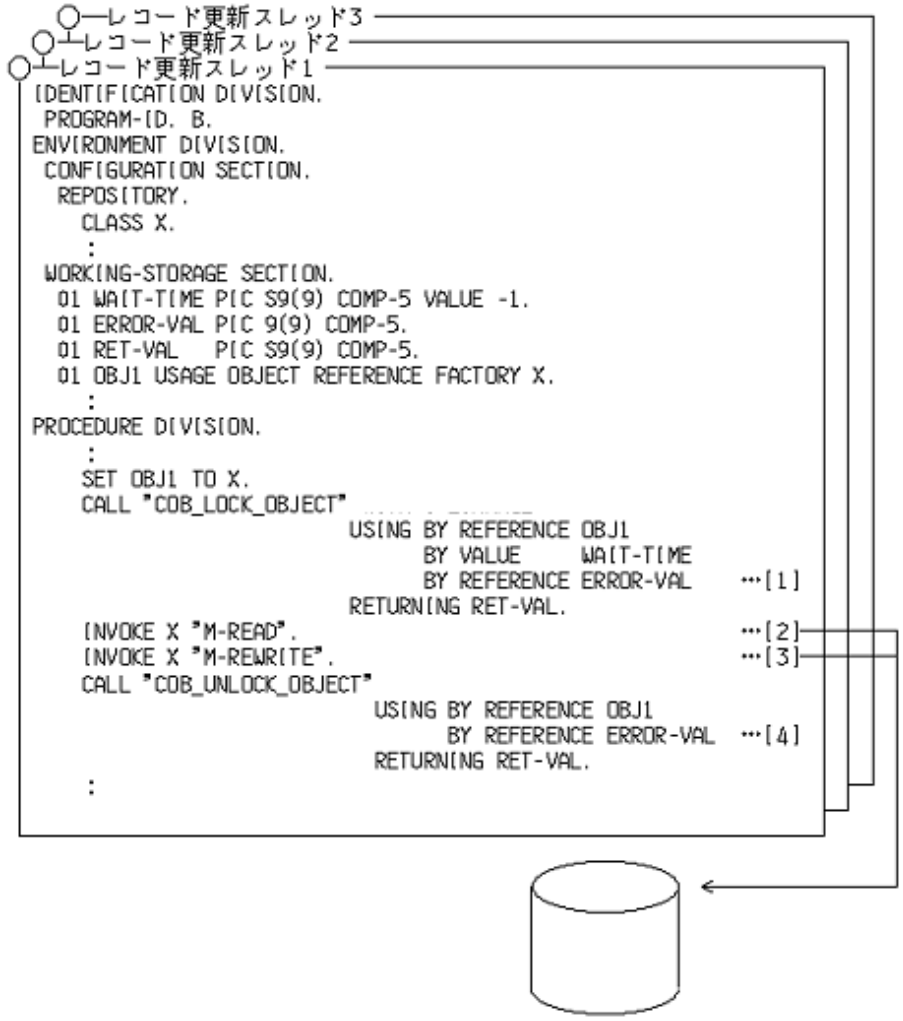
[2] レコード読込みスレッドは、複数(プログラム例では3つ)同時に起動されます。ファクトリメソッド'M-READ'を呼び出し、初期化スレッドによって開かれているファイルに対し、レコードを読み込みます。

[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出しに対するスレッドの同期制御のプログラム例




```
IDENTIFICATION DIVISION.
 PROGRAM-ID. B.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
  REPOSITORY.
    CLASS X.
    :
 WORKING-STORAGE SECTION.
  01 WAIT-TIME PIC S9(9) COMP-5 VALUE -1.
  01 ERROR-VAL PIC 9(9) COMP-5.
  01 RET-VAL   PIC S9(9) COMP-5.
  01 OBJ1 USAGE OBJECT REFERENCE FACTORY X.
    :
PROCEDURE DIVISION.
    :
    SET OBJ1 TO X.
    CALL "COB_LOCK_OBJECT"
                        USING BY REFERENCE OBJ1
                              BY VALUE     WAIT-TIME
                              BY REFERENCE ERROR-VAL    …[1]
                        RETURNING RET-VAL.
    INVOKE X "M-READ".                                  …[2]
    INVOKE X "M-REWRITE".                               …[3]
    CALL "COB_UNLOCK_OBJECT"
                          USING BY REFERENCE OBJ1
                                BY REFERENCE ERROR-VAL  …[4]
                          RETURNING RET-VAL.
    :
```

[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.1.10 オブジェクトロック獲得サブルーチン(COB_LOCK_OBJECT)”を参照してください。

18.6.1.3 オブジェクト内に定義したファイル

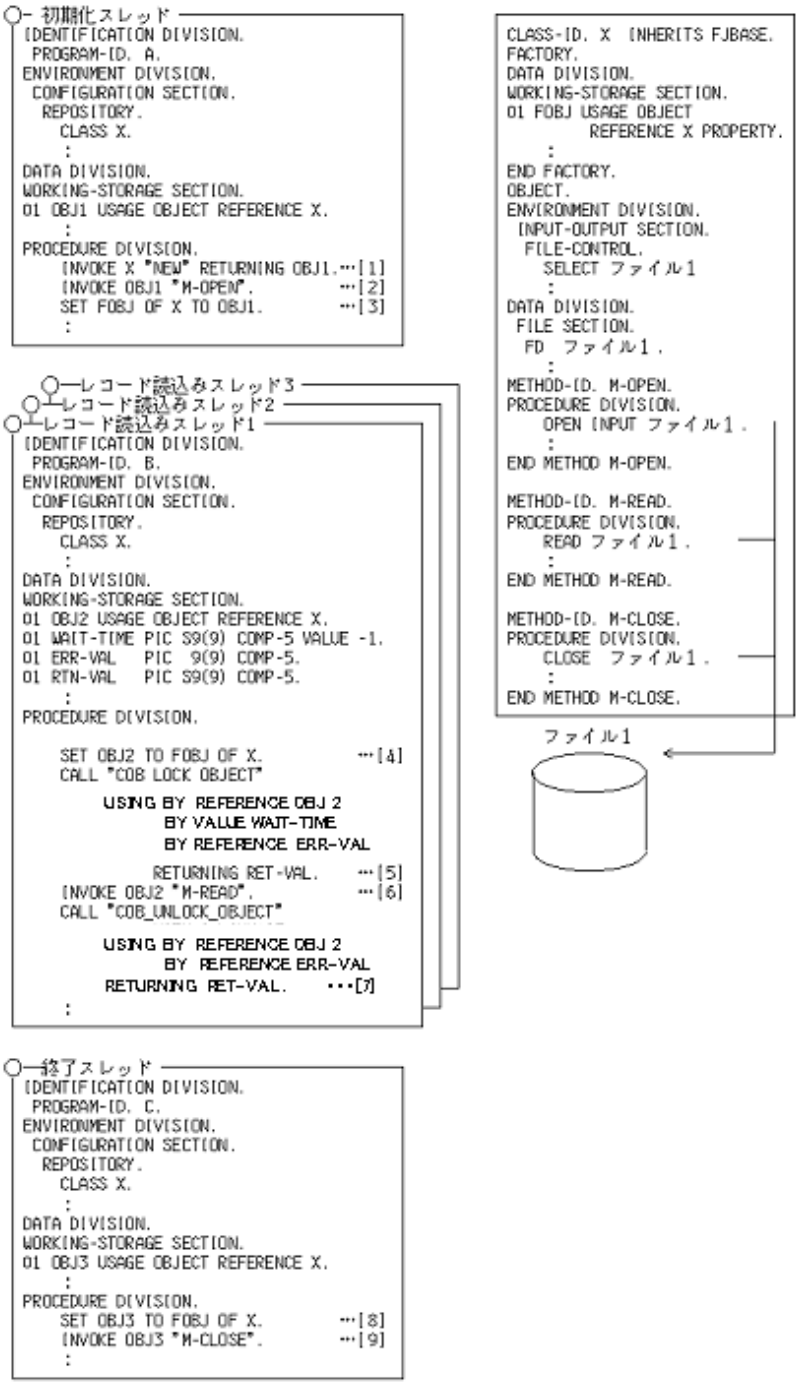
オブジェクト内のファイル記述項に定義したファイルは、1つのオブジェクトインスタンスを共有することで、外部ファイルと同様に同じファイル結合子を共有することができます。

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([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。

レコード読込みスレッド([4]～[7])は、複数(プログラム例では3つ)同時に起動されます。

[4] オブジェクトインスタンス'OBJ2'にファクトリデータ'FOBJ'を代入します。これにより、プログラムAで使用し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で使用し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.1.10 オブジェクトロック獲得サブルーチン(COB_LOCK_OBJECT)”を参照してください。

18.6.2 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。

18.6.2.1 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。

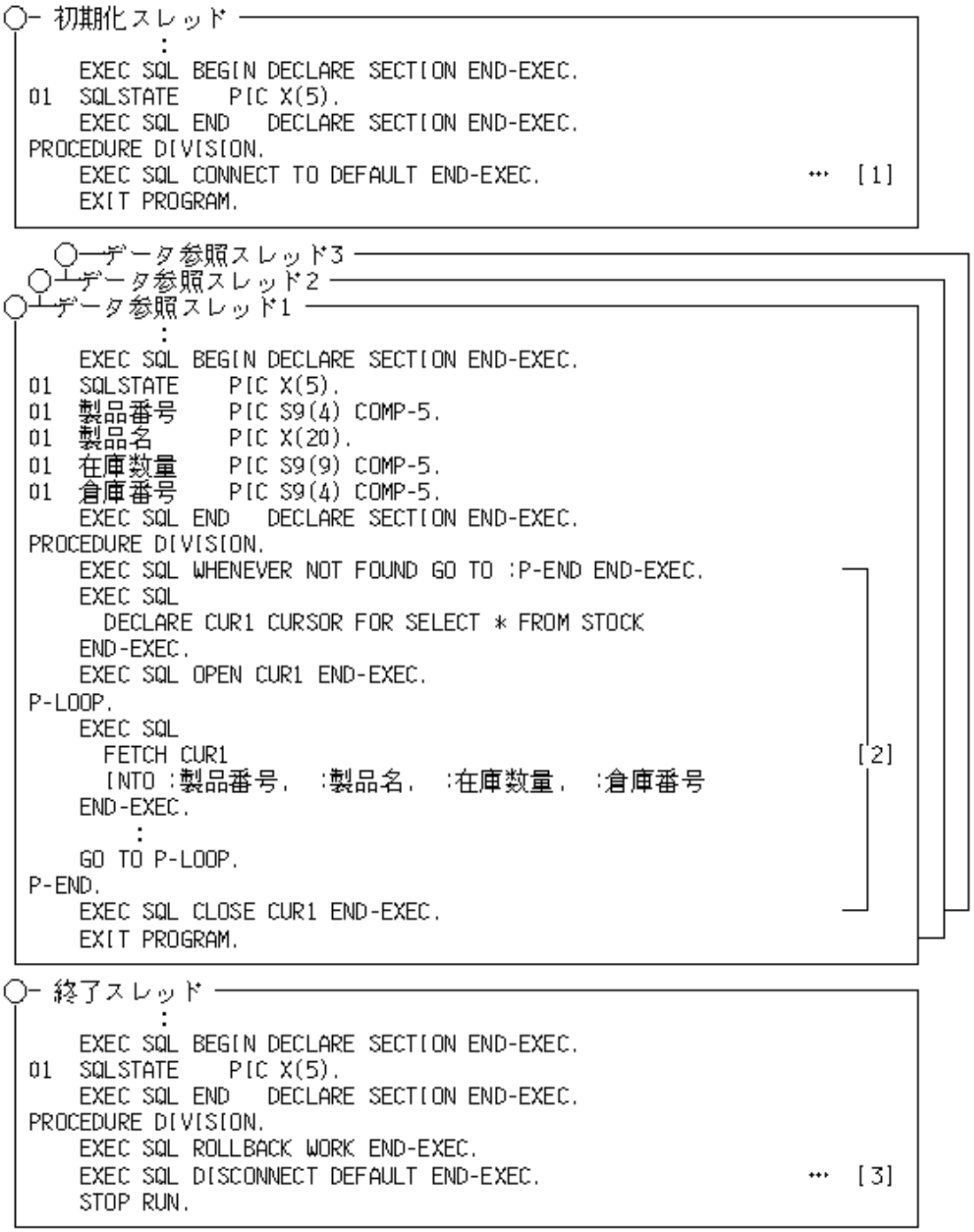
以降では、データベース表を変更しない(参照だけ行う)例、およびデータベース表を更新する例を説明します。

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18.6 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。

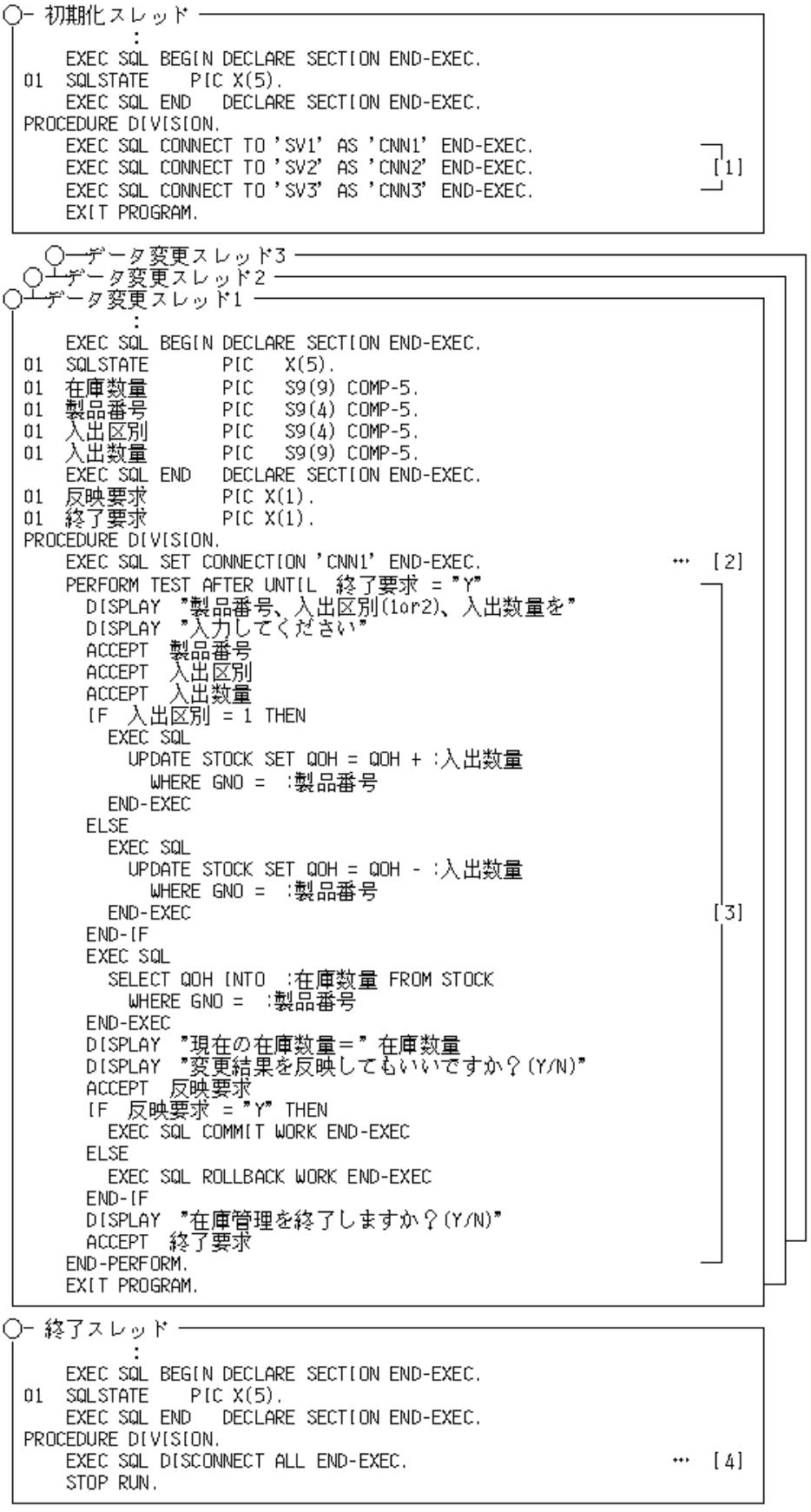
[2] データ参照スレッドは、クライアントからの要求ごとに起動されます。
データ参照スレッドで、カーソルを使用してSTOCK表からデータを1行ずつ取り出し、各列の値を対応するホスト変数の領域に設定します。取り出す行がなくなったとき、WHENEVER文で指定した手続き名“P-END”の位置に分岐し、データ参照スレッドを終了します。

[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18.7 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18.7 コネクション共有時のデータ更新例




```
初期化スレッド
        :
    EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01  SQLSTATE    PIC X(5).
    EXEC SQL END   DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
    EXEC SQL CONNECT TO 'SV1' AS 'CNN1' END-EXEC.                  ┐
    EXEC SQL CONNECT TO 'SV2' AS 'CNN2' END-EXEC.                   [1]
    EXEC SQL CONNECT TO 'SV3' AS 'CNN3' END-EXEC.                  ┘
    EXIT PROGRAM.

データ変更スレッド3
データ変更スレッド2
データ変更スレッド1
        :
    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.
01  入出数量       PIC   S9(9) COMP-5.
    EXEC SQL END   DECLARE SECTION END-EXEC.
01  反映要求       PIC X(1).
01  終了要求       PIC X(1).
PROCEDURE DIVISION.
    EXEC SQL SET CONNECTION 'CNN1' END-EXEC.                  …  [2]
    PERFORM TEST AFTER UNTIL  終了要求 = "Y"                       ┐
      DISPLAY  "製品番号、入出区別(1or2)、入出数量を"               │
      DISPLAY  "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                                                 [3]
      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.                                                  ┘
    EXIT PROGRAM.

終了スレッド
        :
    EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01  SQLSTATE    PIC X(5).
    EXEC SQL END   DECLARE SECTION END-EXEC.
PROCEDURE DIVISION.
    EXEC SQL DISCONNECT ALL END-EXEC.                         …  [4]
    STOP RUN.
```

[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、入庫または出庫を切り分けて在庫数量を再計算します。次に再計算後の在庫数量を表示し、変更結果の反映有無を入力させます。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。

18.6.2.2 注意事項

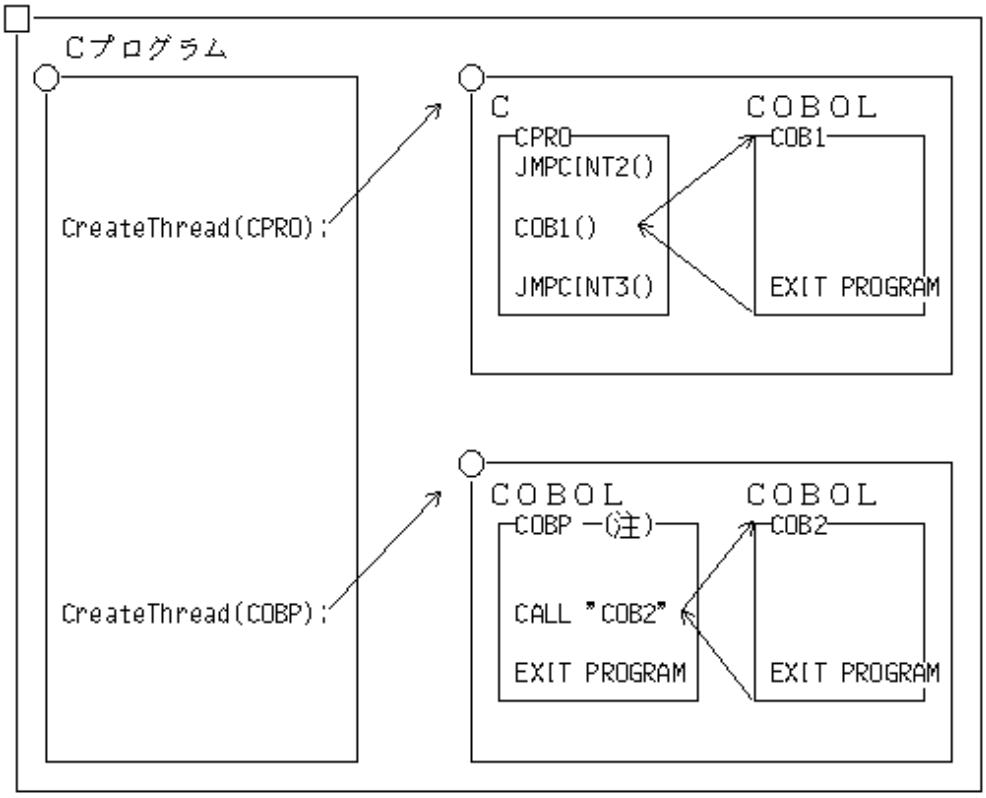
詳細については、"注意事項"を参照してください。

18.6.3 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。

18.6.3.1 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。



18.6.3.2 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。

18.6.3.3 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。

18.6.3.4 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]
        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
                  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 long int型として受け取る必要があります。

― 関数C

```
extern long long int
 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 USING PRM.

               :
     IF  エラー発生
      THEN MOVE 99 TO PROGRAM-STATUS
```

・ GetExitCodeThreadで復帰値を取り出す場合、復帰値として返せる値は32ビットの範囲内にする必要があります。

18.6.3.5 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>
extern int COBTHD1(int *);
extern int COBTHD2(int *);
extern int COBTHD3(int *);
               :
int WINAPI WinMain( ～  )
{
/*  データ宣言                                          */
  HANDLE cobhnd1;
  HANDLE cobhnd23[2];
  DWORD    cobrcd1;
  DWORD    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 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 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 USING PRM3.
    IF PRM3 = 3
     THEN MOVE  0 TO PROGRAM-STATUS
     ELSE MOVE -1 TO PROGRAM-STATUS
    END-IF.
    EXIT PROGRAM.
```

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

```
COBOL -WC,"THREAD(MULTI)" COBTHD1.COB
```

COBOLプログラムCOBTHD2を翻訳する

```
COBOL -WC,"THREAD(MULTI)" COBTHD2.COB
```

COBOLプログラムCOBTHD3を翻訳する

```
COBOL -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、“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”の“DLLエントリオブジェクトの結合”を参照してください。

COBOLプログラムCOBTHD1を結合する

```
LINK COBTHD1.OBJ F4AGCBDM.OBJ F4AGCIMP.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
F4AGCBDM.OBJ   :  DLLエントリオブジェクト
F4AGCIMP.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 F4AGCBDM.OBJ F4AGCIMP.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
F4AGCBDM.OBJ   :  DLLエントリオブジェクト
F4AGCIMP.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 F4AGCBDM.OBJ F4AGCIMP.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
F4AGCBDM.OBJ   :  DLLエントリオブジェクト
```

F4AGCIMP.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

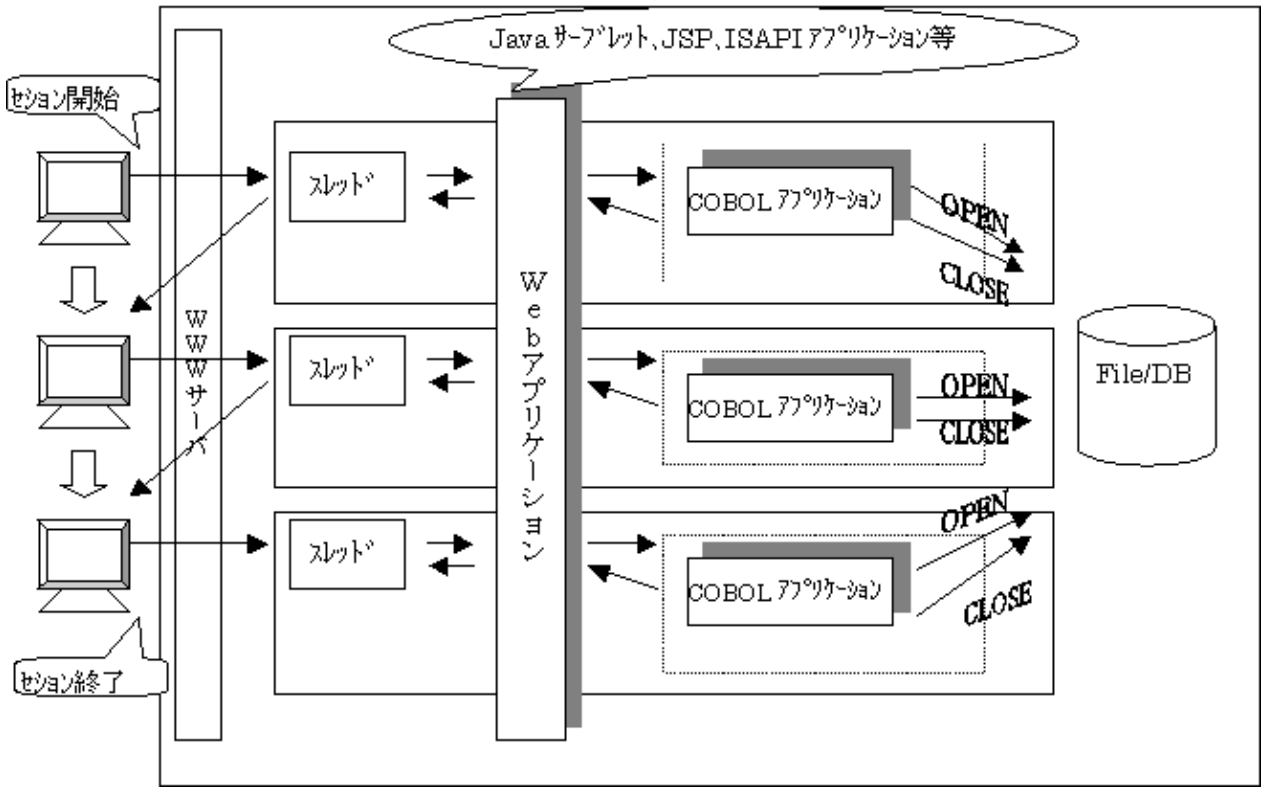
18.6.4 スレッド間で実行単位の資源を引き継ぐ方法

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。

18.6.4.1 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。

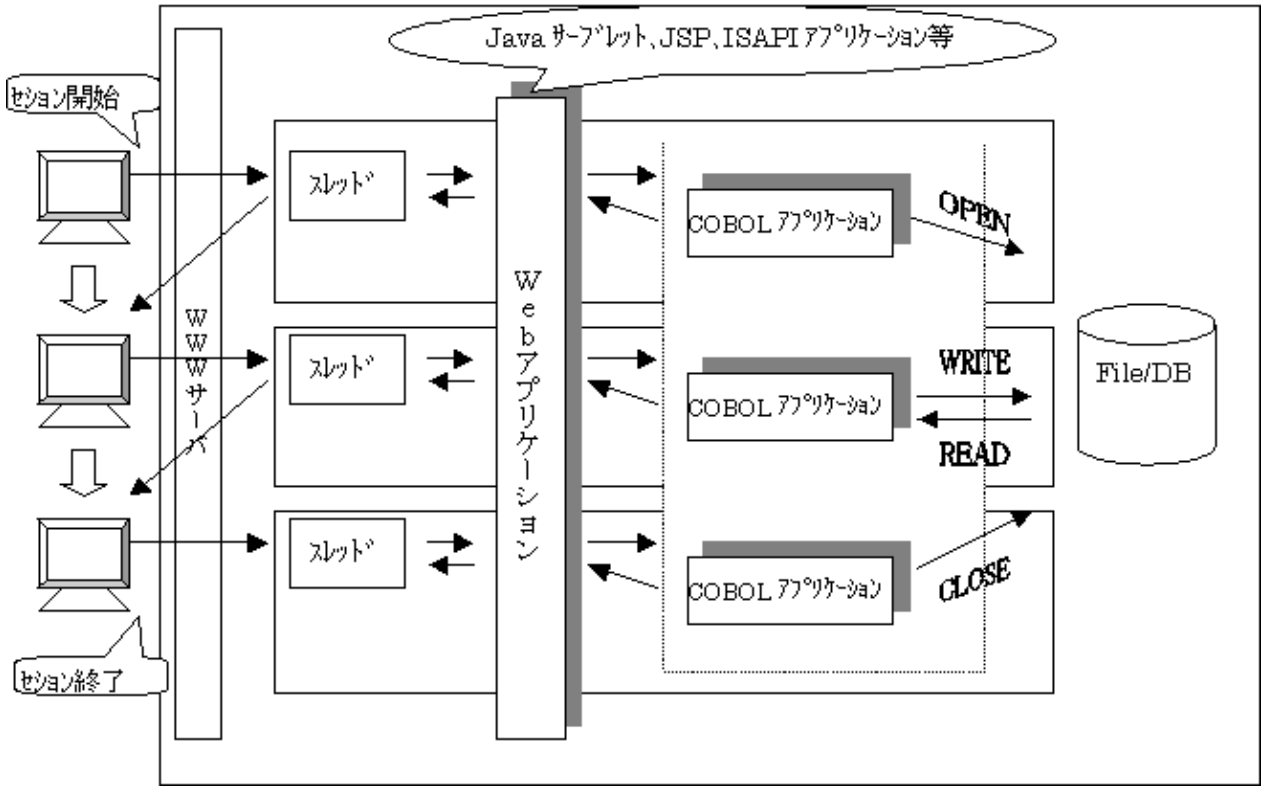
［実行単位を引き継がない場合］



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。

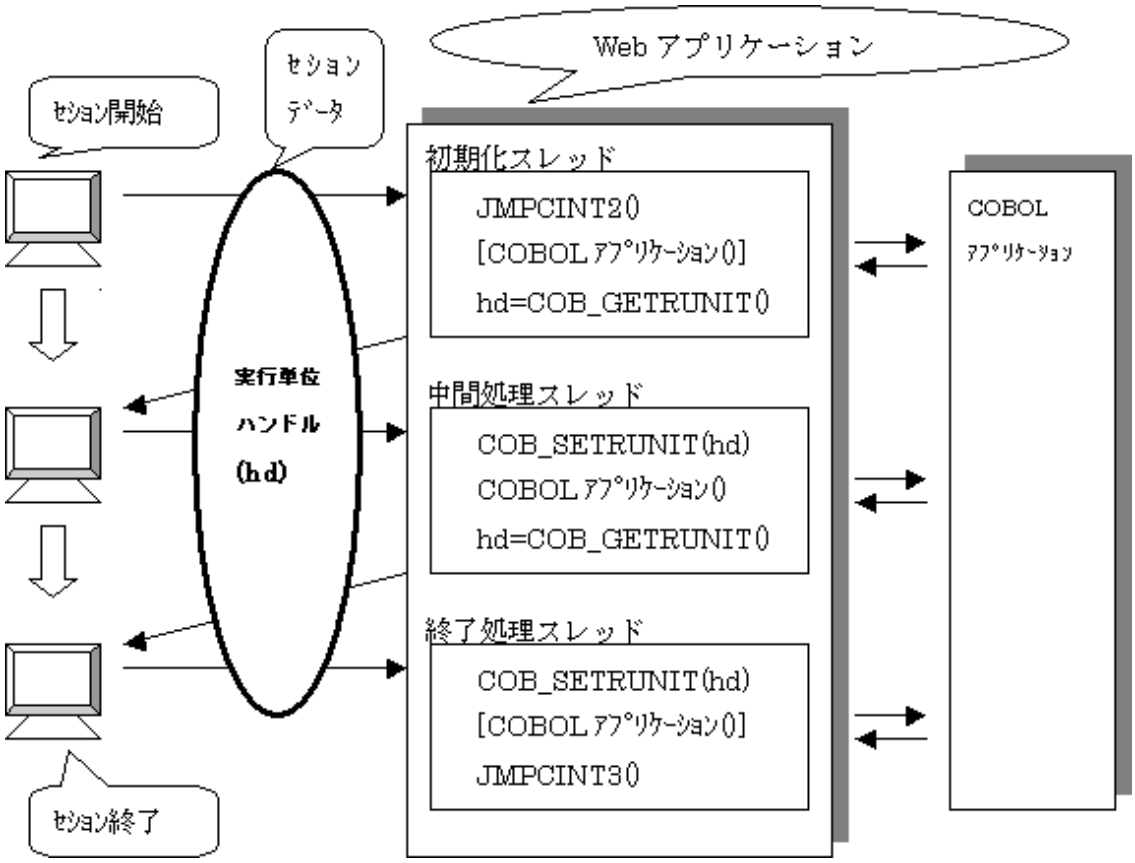
このような場合に、COBOLの実行単位のハンドルを返却するサブルーチンと、その実行単位のハンドルをCOBOLランタイムシステムに通知するサブルーチンを使用して、スレッド間で実行単位を引き継ぐことができます。引き継ぐ実行単位のハンドルとセションの関連付けは、Cookieなどを用いた管理機構によって、COBOLアプリケーションの呼出し側で行う必要があります。

[実行単位を引き継ぐ場合]



18.6.4.2 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。

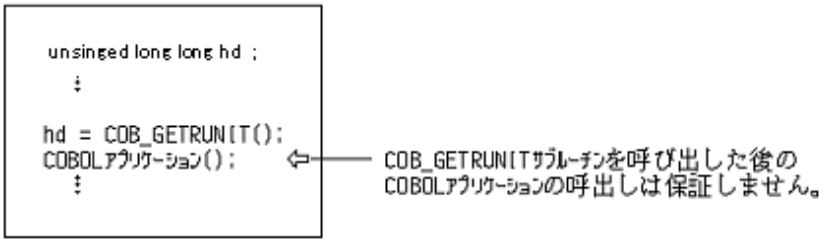


別スレッドで開設された実行単位の資源(ファイル結合子、DBカーソル、作業場所節に記述したデータなど)を引き継ぐことにより、クライアントからの呼出しごとにオープン処理からクローズ処理までを行わなければならなかったファイル操作のオーバヘッドなどが削減でき、処理の効率化も図ることができます。

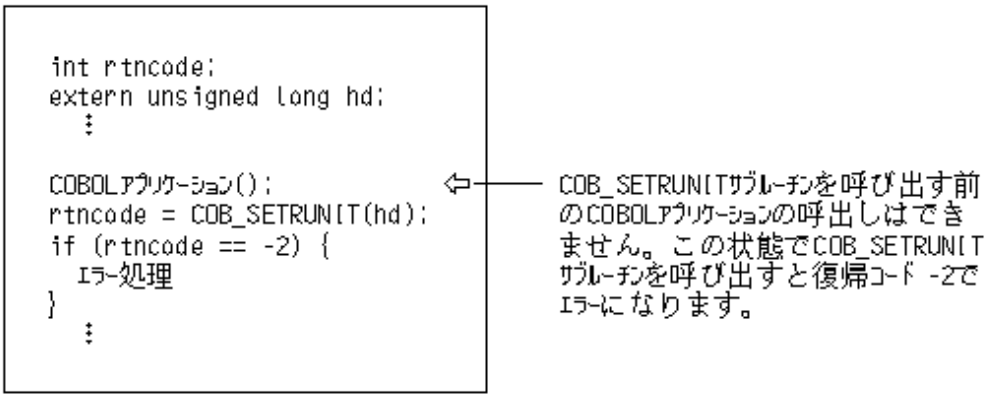
COBOLの実行単位のハンドルを返却するサブルーチンおよび実行単位のハンドルをCOBOLランタイムシステムに通知するサブルーチンの詳細は、“I.2.4 COBOL実行単位ハンドル取得サブルーチン(COB_GETRUNIT)”および“I.2.5 COBOL実行単位ハンドル設定サブルーチン(COB_SETRUNIT)”を参照してください。

18.6.4.3 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、“19.4 COUNT機能”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)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。

  ― 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。

18.6.4.4 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F4AGCIMP.LIBをリンクしてください。

18.7 翻訳から実行までの方法

18.7.1 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。

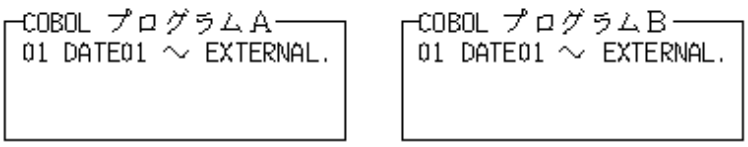
マルチスレッドプログラムのリンク手順は、シングルスレッドプログラムと同じです。

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。

18.7.1.1 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

```
COBOL -WC,"THREAD(MULTI),SHREXT" A.COB
```

COBOLプログラムBを翻訳する

```
COBOL -WC,"THREAD(MULTI),SHREXT" B.COB
```

注意

スレッド間共有外部データまたはスレッド間共有外部ファイルを使用する場合は、THREAD(MULTI)のほかに翻訳オプションSHREXTを指定してください。

リンク

リンク方法については、“5.5.4 DLL配下にある実行用の初期化ファイルを使用する”の“DLLエントリオブジェクトの結合”を参照してください。

COBOLプログラムAとCOBOLプログラムBをリンクしてCOB.DLLを作成する

```
LINK A.OBJ B.OBJ F4AGCBDM.OBJ F4AGCIMP.LIB KERNEL32.LIB /DLL /ENTRY:COBDMAIN /OUT:COB.DLL
```

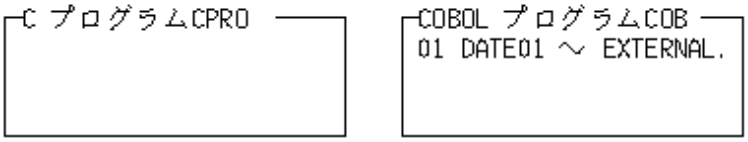
A.OBJ	:	COBOLプログラムAのオブジェクトファイル
B.OBJ	:	COBOLプログラムBのオブジェクトファイル
F4AGCBDM.OBJ	:	DLLのエントリオブジェクトファイル
F4AGCIMP.LIB	:	COBOLランタイムシステムのインポートライブラリ
KERNEL32.LIB	:	Windows関数のインポートライブラリ

注意

LINKオプション/ENTRYに、必ずCOBDMAINを指定してください。

18.7.1.2 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

```
COBOL -WC,"THREAD(MULTI),SHREXT" COB.COB
```

注意

- Cプログラムをマルチスレッド環境下で動作するように翻訳してください(Visual C++では翻訳オプション/MDが必要となります)。
- COBOLプログラムでスレッド間共有外部データまたはスレッド間共有外部ファイルを使用する場合は、THREAD(MULTI)のほかに翻訳オプションSHREXTを指定してください。

リンク

リンク方法については、“5.5.4 DLL配下にある実行用の初期化ファイルを使用する”の“DLLエントリオブジェクトの結合”を参照してください。

CプログラムCPROとCOBOLプログラムCOBをリンクしてCCOB.DLLを作成する

```
LINK CPRO.OBJ COB.OBJ F4AGMLDM.OBJ F4AGCIMP.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
F4AGMLDM.OBJ	:	DLLエントリオブジェクト
F4AGCIMP.LIB	:	COBOLランタイムシステムのインポートライブラリ
KERNEL32.LIB	:	Windows関数のインポートライブラリ
MSVCRT.LIB	:	Cランタイムライブラリ(翻訳オプション/MDで翻訳した場合、当ライブラリが必要となります)。

18.7.2 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。

18.7.2.1 実行環境情報の設定

ここでは、実行環境情報の設定の手順について説明します。なお、シングルスレッドモードと重複する説明については、“第5章 プログラムを実行する”を参照してください。

実行用の初期化ファイルの内容

マルチスレッドモードで動作するCOBOLアプリケーションの使用する実行用の初期化ファイルの記述内容について説明します。

マルチスレッドモードでは、実行用の初期化ファイルの内容は、共通部だけで構成されます。セクションに記述された情報は無視されます。

以下に、実行用の初期化ファイルの内容の記述形式を示します。

```
実行環境情報名= 設定内容                    … [1]    共通部
```

[1] 環境変数情報(共通部)

ここには、各プログラムに共通する環境変数情報を記述します。それぞれの環境変数情報の指定形式については、“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

```
@MessOutFile=C:¥MESSAGE.TXT
@CnslWinSize=(80,24)
@CnslBufLine=100
@WinCloseMsg=ON
@IconName=COB85EXE
@CBR_ENTRYFILE=C:¥TEST001.ENT
```

シングルスレッドアプリケーションをマルチスレッド環境に移行する場合、以下の注意が必要です。

- セクションに記述された情報を共通部に指定してください。

例) COBOL85.CBR

```
OUTF[LE=C:¥OUTDATA.DAT

[A]
OUTF[LEA=C:¥A¥DATAA.OUT
[NF[LEA=C:¥A¥MASTER.[N

[B]
OUTF[LEB=C:¥B¥DATAB.OUT
[NF[LEB=C:¥B¥MASTER.[N
```

→

COBOL85.CBR

```
OUTF[LE=C:¥OUTDATA.DAT
OUTF[LEA=C:¥A¥DATAA.OUT
[NF[LEA=C:¥A¥MASTER.[N
OUTF[LEB=C:¥B¥DATAB.OUT
[NF[LEB=C:¥B¥MASTER.[N
```

- エントリ情報が記述されている場合、エントリ情報ファイルに指定し、共通部の@CBR_ENTRYFILEにファイル名を指定してください。

例) COBOL85.CBR

```
OUTF[LE=C:¥A.OUT

[A]
@MessOutFile=C:¥A.MSG

[A.ENTRY]
A01=C:¥DLL¥B1.DLL
A02=C:¥DLL¥B2.DLL
```

→

COBOL85.CBR

```
OUTF[LE=C:¥A.OUT
@MessOutFile=C:¥A.MSG
@CBR_ENTRYF[LE=C:¥A.ENT
```

C:¥A.ENT

```
[ENTRY]
A01=C:¥DLL¥B1.DLL
A02=C:¥DLL¥B2.DLL
```

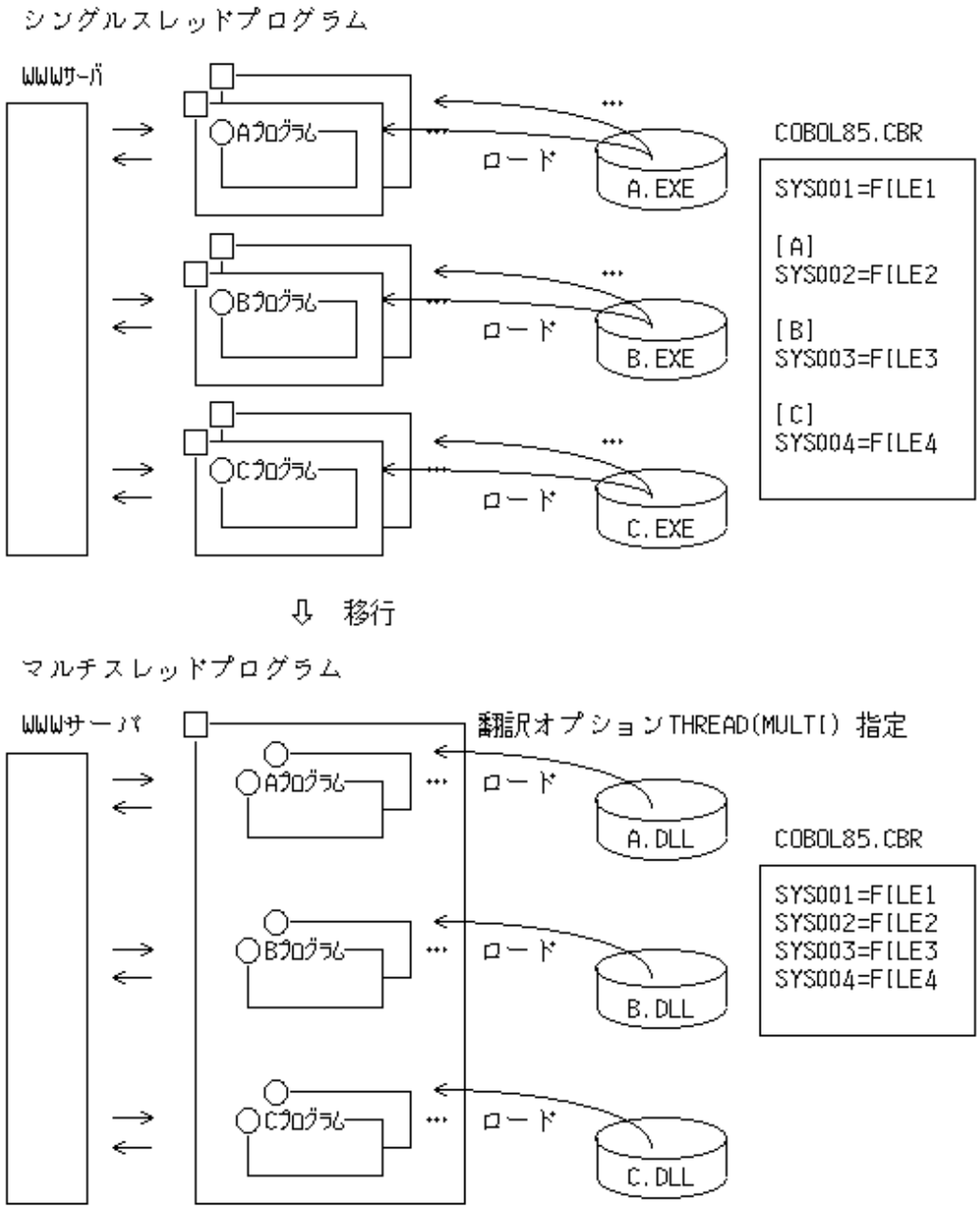
- サーバ環境下では、環境変数の値としてファイル名に相対パスを指定しているものがあると、相対の基点がサーバによって異なる可能性があります。このような場合は、絶対パスでの指定が可能であれば絶対パスによるファイル名の指定を行ってください。

スレッドモードと実行用の初期化ファイル内の有効な記述についてまとめると、以下のようになります。

	シングルスレッドモード	マルチスレッドモード
セクション	有効	無効
共通部	有効	有効

例

Webサーバで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。

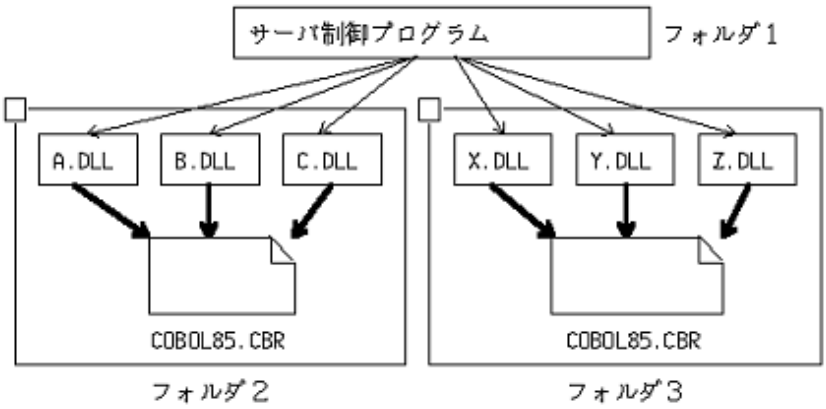
同一の実行環境で動作するアプリケーションのCOBOLプログラム(DLL)が1つのフォルダにない場合、呼出し元のプログラムの処理によっては、どのフォルダのCOBOL85.CBRファイルが読み込まれるかわからなくなることがありますので注意してください。

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。



18.7.2.2 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)"

18.8 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
- NetCOBOL Studioのデバッグ機能

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。

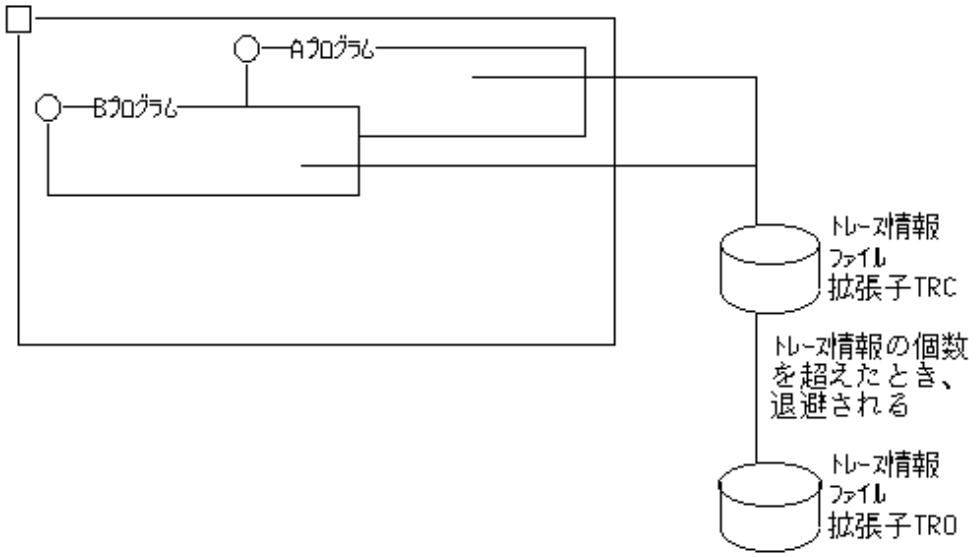
それぞれの機能に分けて、注意事項を説明します。

18.8.1 TRACE機能

トレース情報の内容に変更はありません。[参照]“19.3 TRACE機能”

ただし、各スレッドから採取されたトレース情報は、1つのファイル(拡張子TRC)に格納されます。

以下に図で示します。



トレース情報の見方を、以下の図で説明します。

```
NetCOBOL DEBUG INFORMATION   DATE 2007-05-01  TIME 20:29:52  PID=00000125

TRACE INFORMATION

  [1]    1  A    DATE 2007-05-01  TIME 20:27:50  TID=0000010C
         2              22.1 TID=0000010C
         3              23.1 TID=0000010C
  [1]    4  B    DATE 2007-05-01  TIME 20:27:50  TID=0000011E
         5              22.1 TID=0000011E
         6              23.1 TID=0000011E
         7              24.1 TID=0000011E
         8              25.1 TID=0000011E
  [2]'   9              26.1 TID=0000011E
        10              24.1 TID=0000010C
        11              25.1 TID=0000010C
        12              26.1 TID=0000010C
  [2]   13    THE INTERRUPTION WAS OCCURRED.PID=00000125,TID=0000011E
        14              27.1 TID=0000010C
        15              28.1 TID=0000010C
  [3]   16      EXIT-THREAD TID=0000010C
```

[1] プログラムに割り当てられたスレッドID

プログラムAのスレッドIDは10Cです。プログラムBのスレッドIDは11Eです。

[2] 例外通知メッセージ

例外が発生したのは、スレッドIDが11EのプログラムBです。

[3] スレッド終了通知メッセージ

スレッドIDが終了したのは、スレッドIDが10CのプログラムAです。

上記の結果から、以下のことが分かります。

- スレッドIDが10Cのプログラムは正常に動作した。
- スレッドIDが11Eのプログラムは26行目の実行文で例外が発生した。

参考

DISPLAY ... UPON SYSERRを使うと、トレース情報に任意のデータを出力することができます。

プログラムが使用するデータの遷移などを調べる場合に便利です。

なお、DISPLAY ... UPON SYSERRを使う場合には、スレッドIDも一緒に出力されるように、環境変数情報@CBR_SYSERR_EXTENDにYESを指定してください。[参照]"C.2.50 @CBR_SYSERR_EXTEND（SYSERR出力情報の拡張の指定）"

注意

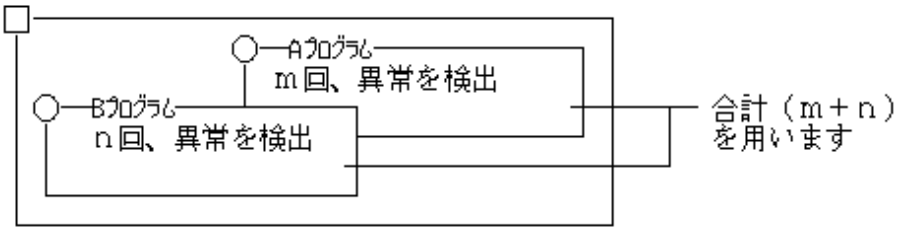
一度に多くのスレッドを実行する場合、大量のトレース情報が1つのファイルに書き込まれます。より多くのトレース情報が1つのファイルに出力されるように、トレース情報の個数(環境変数情報@GOPTのr指定)を調整してください。

18.8.2 CHECK機能

CHECK機能が有効な場合に、出力されるエラーメッセージの内容および検出方法に変更はありません。[参照]"19.2 CHECK機能"

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。



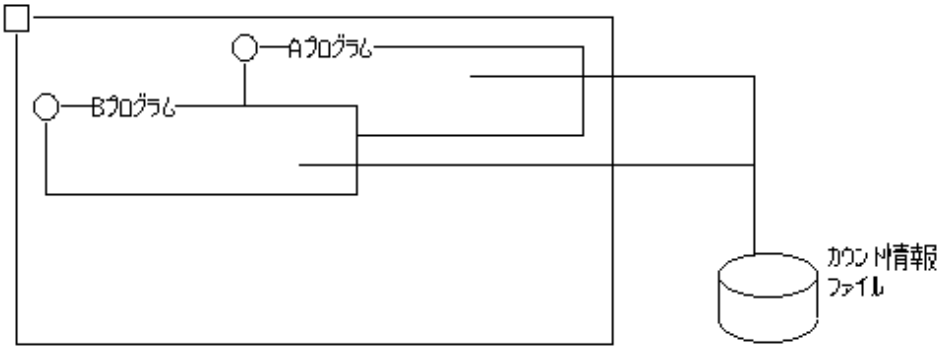
18.8.3 COUNT機能

カウント情報の内容に変更はありません。[参照]"19.4 COUNT機能"

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。

18.8.4 NetCOBOL Studioのデバッグ機能

デバッグにはNetCOBOL Studioのデバッグ機能を使用します。詳細は、“NetCOBOL Studio ユーザーズガイド”を参照してください。

18.9 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.4 データロックサブルーチンの使い方
- I.5 オブジェクトロックサブルーチンの使い方

18.10 注意事項

18.10.1 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、“18.3.1 実行環境と実行単位”を参照してください。

18.10.2 印刷機能

ここでは、スレッド間で同じファイル結合子を操作して印刷ファイルまたは表示ファイルを共有する場合の注意事項について説明します。

複数の入出力文の実行によって1つの帳票が生成されるような処理を行う場合、同じファイル結合子に対して複数のスレッドから入出力文の実行が行われると、入出力文の実行順序が非同期となるため、印刷データの出力順序が一定とならずに意図しない印刷結果を得ることがあります。このため、同一帳票に対する一連の処理開始から終了までの間、他スレッドの入出力文との競合を抑止する必要があります。

他スレッドとの競合状態を防ぐためには、一連の処理の前後でスレッドの同期制御を行います。

スレッド間で同じファイル結合子を持つファイルおよびスレッドの同期制御の詳細については、“18.6.1 入出力機能の利用”を参照してください。

18.10.3 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19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
- 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 Stiudioのデバッグ機能	デバッガの基本的な機能(中断点、データの表示/変更など)に加えて、以下の機能も持っています。 ・ データ変更での中断 【用途】 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。 ・ 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 ・ 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 ・ 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 ・ 数字項目参照時、属性形式と異なる値が入っていないか ・ 除算のとき、除数がゼロでないか ・ 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 ・ 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 【用途】 ・ 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 ・ 数値異常によるプログラムの誤動作を防ぎたい場合 ・ 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	CHECK
TRACE機能	次の情報を出力します。 ・ 実行した文のトレース結果 ・ 異常終了したときに実行した文の行番号 ・ 実行した文を含むプログラム名とプログラム属性情報 ・ 実行中に出力されたメッセージ 【用途】 ・ どの文で異常終了したのかを知りたい場合 ・ 異常終了までに実行した文の経路を知りたい場合 ・ 実行の途中で出力されたメッセージを確認したい場合	TRACE

機能名	使い方	翻訳オプション
COUNT機能	次の情報を出力します。 ・ プログラム上の各文の実行回数および全文の全実行回数に対する各文の実行比率 ・ プログラム上の文種別ごとの実行回数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 【用途】 ・ プログラムの実行した全ルートの走行を確認したい場合 ・ プログラムの効率化を図りたい場合	COUNT
メモリチェック機能	次の検査を行います。 ・ 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。 ― 破壊を検出したプログラム名またはメソッド名 ― 破壊を検出した場所(手続き部の開始／終了) ― 破壊された領域のアドレス 【用途】 ・ 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	―

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

[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。

19.1 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。

NetCOBOL Studioのデバッグ機能の詳細は、“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。

19.2 CHECK機能

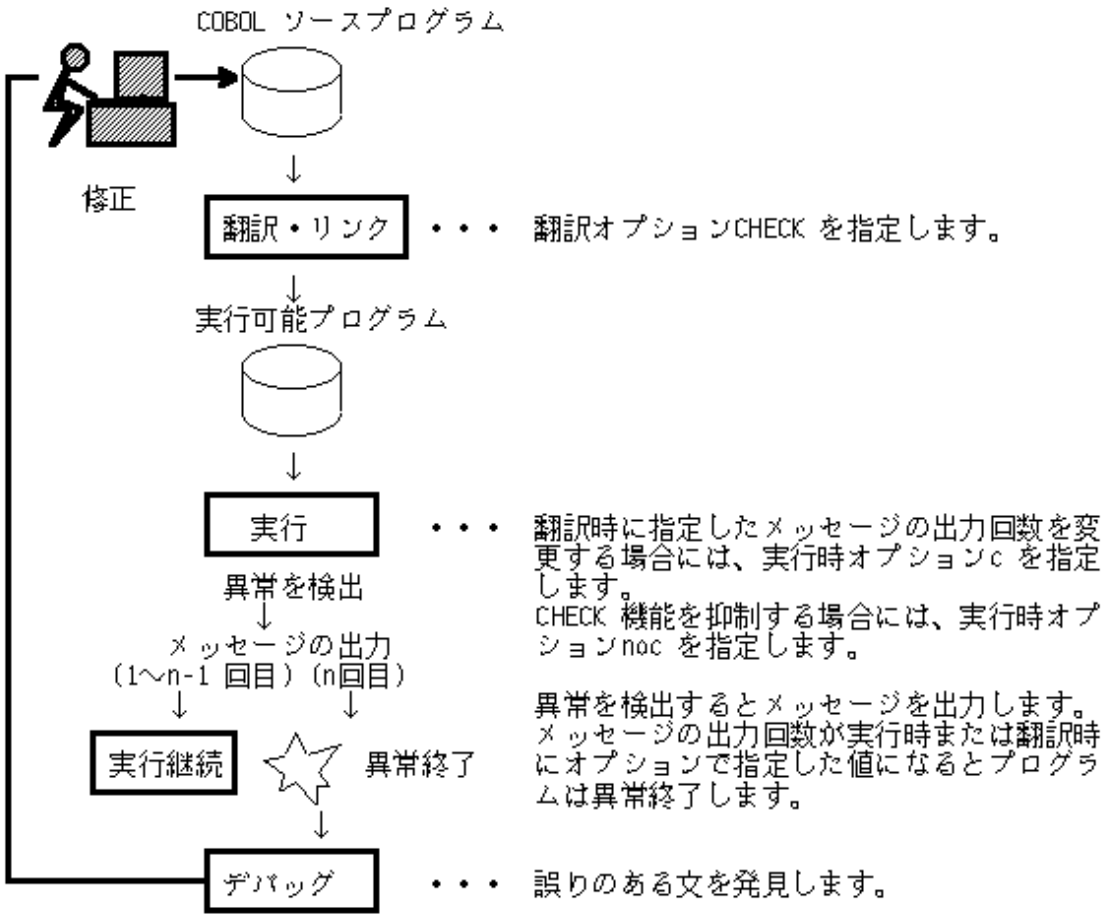
CHECK機能では、以下の検査を行い、異常を検出するとメッセージを出力し、異常終了します。そのため、プログラムの誤動作を防ぐことができます。

・ 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
- 内部プログラム呼出しのパラメタ
- 外部プログラム呼出しのパラメタ

19.2.1 デバッグ作業の流れ

以下にCHECK機能を使ったデバッグ作業の流れを示します。



19.2.2 出力メッセージ

CHECK機能では、検査によって異常を検出すると、メッセージを出力します。このメッセージは、通常メッセージボックスに出力されます。CHECK機能で出力されるメッセージの重大度コードは通常Eレベルですが、メッセージの出力回数が指定された回数になるとUレベルとなります。重大度コードについては、“メッセージ集”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をもちません。

以下にCHECK機能のメッセージについて説明します。

19.2.2.1 メッセージの内容

メソッド呼出しのパラメタ検査

JMP0810I-E/U
[PID:xxxxxxxx TID:xxxxxxxx] ’メソッド名’メソッドのUSING(またはRETURNING)指定のパラメタに誤りがあります.[’NUMBER’(またはPARAMETER=パラメタの順序番号)] PGM=プログラム名 LINE=ステートメント番号

〔対処〕

メッセージで指摘されたメソッドを正しいパラメタで呼び出すように、プログラムを修正し、再度実行してください。

添字および指標検査

JMP0820I-E/U
[PID:xxxxxxxx TID:xxxxxxxx] 添字または指標の値が範囲外を指しています.PGM=プログラム名.LINE=ステートメント番号.OPD=データ名(データ名の次元数)

〔対処〕

メッセージで指摘された添字または指標に正しい値が設定されるようにプログラムを修正してください。

部分参照検査

JMP0821I-E/U
[PID:xxxxxxxx TID:xxxxxxxx] 参照可能範囲外の部分参照を行っています.PGM=プログラム名.LINE=ステートメント番号.OPD=データ名.

〔対処〕

メッセージで指摘されたデータ名の部分参照を、その範囲を超えないようにプログラムを修正してください。

OCCURS DEPENDING ON句の目的語検査

JMP0822I-E/U
[PID:xxxxxxxx TID:xxxxxxxx] ODO句の目的語の値が許容範囲を超えています.PGM=プログラム名.LINE=ステートメント番号.OPD=データ名.ODO=ODO句の目的語の名前.

〔対処〕

メッセージで指摘されたOCCURS DEPENDING ON句の目的語に正しい値が設定されるようにプログラムを修正してください。

数字のデータ例外検査

JMP0828I-E/U
[PID:xxxxxxxx TID:xxxxxxxx] 属性と異なる形式のデータが格納されています.PGM=プログラム名.LINE=ステートメント番号.OPD=データ名.

〔対処〕

メッセージで指摘されたオペランドに正しい値を設定し、再度実行してください。

除数のゼロ検査

JMP0829I-E/U
[PID:xxxxxxxx TID:xxxxxxxx] 除数にゼロが指定されています.PGM=プログラム名.LINE=ステートメント番号.OPD=データ名

〔対処〕

メッセージで指摘されたオペランドに除数がゼロにならないように値を設定し、再度実行してください。

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:xxxxxxxx TID:xxxxxxxx] CALL文のパラメタが一致していません. '$1' PGM=プログラム名. LINE=ステートメント番号.

- $1:検出した誤りを示す文字列

〔対処〕

$1で指摘された内容をもとに、下表に示す処置を施してください。

表19.1 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。

19.2.2.2 メッセージの出力回数

メッセージの出力回数は、翻訳時に翻訳オプションCHECKに指定します。実行単位中にCHECKオプションを指定したCOBOLプログラムが複数存在する場合、最初に動作したプログラムの翻訳オプションに指定した出力回数が有効になります。[参照]"A.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)
- 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"

19.2.3 CHECK機能の使用例

ここでは、CHECK機能の使用例を示します。

添字および指標検査

プログラムA

```
000000 @OPTIONS CHECK(BOUND)
            :
000500  77  添字    PIC S9(4).
000600  01  表.
000700      02  表１   OCCURS 10 TIMES INDEXED BY  指標１.
000800          03  要素１   PIC 9(5).
            :
001100      MOVE  15     TO  添字.
001200      ADD    1     TO  要素１ (添字).
001300      SET  指標１   TO  0.
001400      SUBTRACT  1  FROM  要素１ (指標１).
            :
```

ADD/SUBTRACT文を実行するときに以下のメッセージが出力されます。

```
JMP0820I-E/U [PID:xxxxxxxx TID:xxxxxxxx]  添字または指標の値が範囲外を指しています. PGM=A. LINE=1200. OPD=要素１
JMP0820I-E/U [PID:xxxxxxxx TID:xxxxxxxx]  添字または指標の値が範囲外を指しています. PGM=A. LINE=1400. OPD=要素１
```

部分参照検査

プログラムA

```
000000 @OPTIONS CHECK(BOUND)
            :
```

```
000500  77  データ１          PIC X(12).
000600  77  データ２          PIC X(12).
000700  77  参照する長さ      PIC 9(4) BINARY.
              :
001100      MOVE  10    TO  参照する長さ.
001200      MOVE    データ１ (1:参照する長さ) TO  データ２ (4:参照する長さ).
              :
```

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

```
JMP0821I-E/U [PID:xxxxxxxx TID:xxxxxxxx]  参照可能範囲外の部分参照を行っています. PGM=A. LINE=1200. OPD= データ２.
```

OCCURS DEPENDING ON句の目的語検査

プログラムＡ

```
000000 @OPTIONS CHECK(BOUND)
              :
000050  77  添字    PIC S9(4).
000060  77  個数    PIC S9(4).
000070  01  表.
000080      02  表１  OCCURS 1 TO  10 TIMES DEPENDING ON  個数.
000090          03  要素     PIC X(5).
              :
000110      MOVE   5    TO  添字.
000120      MOVE  25    TO  個数.
000130      MOVE  "ABCDE"  TO  要素 (添字).
              :
```

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

```
JMP0822I-E/U [PID:xxxxxxxx TID:xxxxxxxx] ODO句の目的語の値が許容範囲を超えています. PGM=A. LINE=120. OPD=要素. ODO=個数.
```

数字のデータ例外検査

プログラムＡ

```
000000 @OPTIONS CHECK(NUMERIC)
              :
000050  01  文字    PIC X(4)    VALUE "ABCD".
000060  01  外部１０進  REDEFINES    文字    PIC S9(4).
000070  01  数字    PIC S9(4).
              :
000150      MOVE    外部１０進    TO    数字.
              :
```

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

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

除数のゼロ検査

プログラムＡ

```
000000 @OPTIONS CHECK(NUMERIC)
              :
000060  01  被除数  PIC S9(8)  BINARY  VALUE  1234.
000070  01  除数    PIC S9(4)  BINARY  VALUE  0.
000080  01  結果    PIC S9(4)  BINARY  VALUE  0.
              :
000150      COMPUTE  結果 =  被除数 /  除数.
              :
```

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

```
JMP0829I-E/U [PID:xxxxxxxx TID:xxxxxxxx]  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:xxxxxxxx TID:xxxxxxxx] '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.
```

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'の長さと一致していなければなりません.
17: JMN3334I-S  CALL文のUSING指定またはRETURNING指定に記述したパラメタ'P3'の型は, プログラム'SUB3'のPROCEDURE DIVISIONのUSING指定またはRETURNING指定に記述したパラメタ'L-OR1'の型と一致していなければなりません.
最大重大度コードは S で, 翻訳したプログラム数は 1 本です.
```

外部プログラム呼出しのパラメタの検査

プログラム呼出しにおいてパラメタ受渡しに誤りがあると、思わぬ所を参照したり、更新したりするため、プログラムを誤動作させてしまいます。

翻訳オプションCHECK(PRM)を指定して翻訳したCOBOLプログラムから、翻訳オプションCHECK(PRM)を指定して翻訳したCOBOLプログラムを呼び出した際、パラメタ個数および、それぞれのパラメタの長さが一致していない場合には、メッセージが出力されます。

```
000010 @OPTIONS CHECK(PRM)
000020 IDENTIFICATION  DIVISION.
000030 PROGRAM-ID.     A.
000040 DATA            DIVISION.
000050 WORKING-STORAGE SECTION.
000060 01  USE-PRM01   PIC 9(04).
000070 01  USE-PRM02   PIC 9(04).
000080 01  RET-PRM01   PIC 9(04).
000090 PROCEDURE       DIVISION.
000100     CALL 'B' USING USE-PRM01 USE-PRM02
000110              RETURNING RET-PRM01.
000120 END PROGRAM     A.
```

```
000000 @OPTIONS CHECK(PRM)
000010 IDENTIFICATION  DIVISION.
000020 PROGRAM-ID.     B.
000030 DATA            DIVISION.
000070 LINKAGE         SECTION.
000080 01  USE-PRM01   PIC 9(08).
000090 01  USE-PRM02   PIC 9(04).
000100 01  RET-PRM01   PIC 9(04).
000120 PROCEDURE       DIVISION USING USE-PRM01 USE-PRM02
```

```
000130                          RETURNING RET-PRM01.
000140 END PROGRAM     B.
```

プログラムAのCALL文を実行するときに以下のメッセージが出力されます。

```
JMP0812I-E/U [PID:xxxxxxxx TID:xxxxxxxx] CALL文のパラメタが一致していません. 'USING 1ST PARAMETER' PGM=A. LINE=10.
```

19.2.4 注意事項

ここでは、CHECK機能使用時の注意事項について説明します。

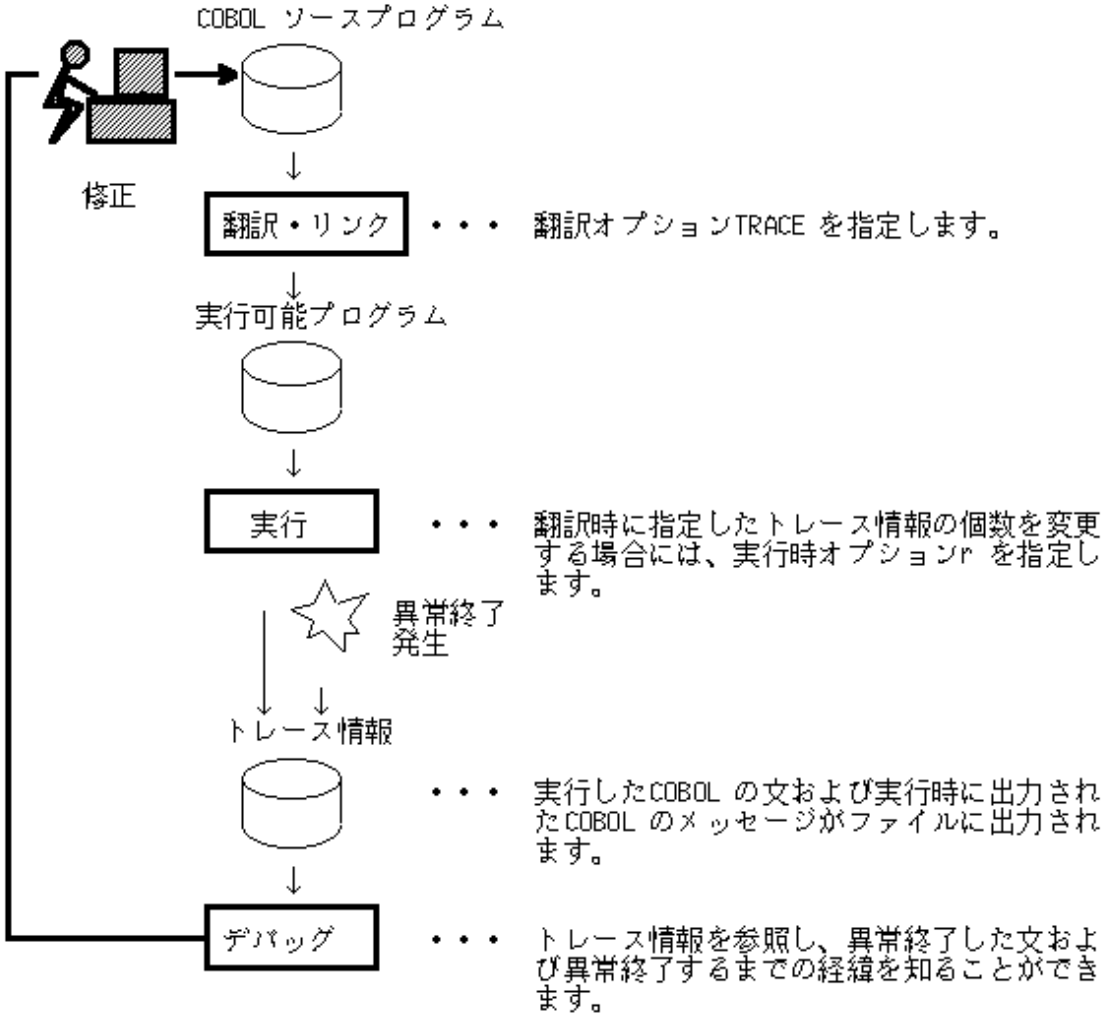
- CHECK機能による検査は必ず実施し、検出された情報を元に誤りを修正してください。検出された誤りが修正されない場合、メモリ破壊などの表面化しにくい重大なトラブルが発生することにつながります。検出された誤りが修正されていないアプリケーションの実行結果は保証できません。
- メッセージ出力回数を指定することにより、異常検出後も実行を継続することができますが、異常検出後の動作は保証されません。
- CHECK機能では、データ内容の検査など、COBOLプログラムで記述した以外の処理を行います。そのため、CHECK機能使用時には、プログラムのサイズが大きくなり、実行速度も遅くなります。CHECK機能は、デバッグ時にだけ使用し、デバッグ終了後は、翻訳オプションCHECKを指定しないで再翻訳してください。
- ON SIZE ERROR指定またはNOT ON SIZE ERROR指定の算術文では、ON SIZE ERRORの除数のゼロ検査が行われ、CHECK(NUMERIC)の除数のゼロ検査は行われません。
- 除数のゼロ検査を行った場合、メッセージの出力回数に関係なく、プログラムは異常終了します。
- CHECK(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)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8バイトのRETURNINGパラメタが指定されたものとして検査します。
- CHECK機能は、CHECKオプションを指定して翻訳したプログラムにだけ有効になります。
  複数のプログラムをリンクしている場合に特定のプログラムだけをCHECK機能の対象にするには、対象にするプログラムはCHECKオプションを指定して翻訳し、対象にしないプログラムはCHECKオプションを指定しないで翻訳してください。

19.3 TRACE機能

TRACE機能では、プログラムの異常終了時に、それまでに実行したCOBOLの文のトレース情報をファイルに出力します。出力されたトレース情報により、異常終了した文やそこまでの経緯を知ることができるので、デバッグ作業に役立ちます。ここでは、TRACE機能の使い方について説明します。

19.3.1 デバッグ作業の流れ

以下にTRACE機能を使ったデバッグ作業の流れを示します。



19.3.2 トレース情報

TRACE機能では、トレース情報として、異常終了するまでに実行したCOBOLの文をステートメント番号で出力します。

トレース情報の個数

翻訳時に個数を指定しない翻訳オプションTRACEを指定した場合、トレース情報は200個出力されます。個数を指定した翻訳オプションTRACEを指定した場合、指定した個数のトレース情報が出力されます。実行単位中にTRACEオプションを指定したCOBOLプログラムが複数存在する場合、最初に動作したプログラムの翻訳オプションに指定した個数が有効になります。[参照]"A.3.54 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 xxxxxxxxx xx xxxxxxxxx.  [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
```

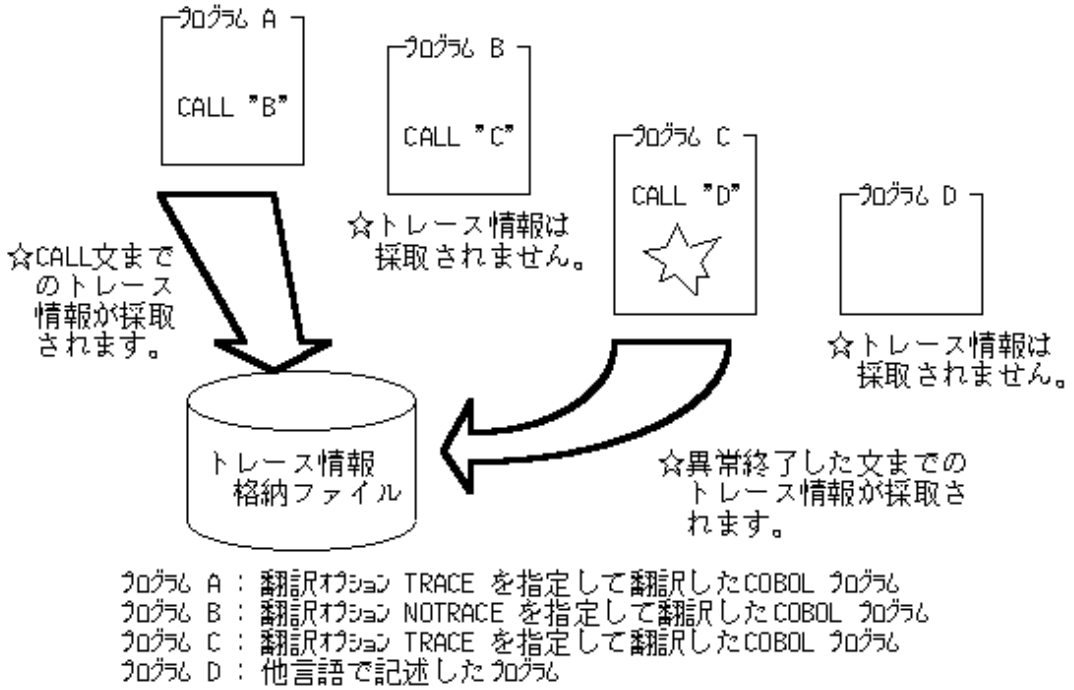
トレース情報ファイル名(一世代前の情報)

```
SAMPLE-00000EC4_20100112_104850.TRO
```

19.3.3 注意事項

ここでは、TRACE機能使用時の注意事項について説明します。

・ TRACE機能でトレース情報を採取できるのは、翻訳オプションTRACEを指定して翻訳したCOBOLプログラムだけです。



・ TRACE機能では、トレース情報の採取など、COBOLプログラムで記述した以外の処理を行います。そのため、TRACE機能使用時には、プログラムのサイズが大きくなり、実行速度も遅くなります。TRACE機能は、デバッグ時にだけ使用し、デバッグ終了後は、翻訳オプションTRACEを指定しないで再翻訳してください。

・ TRACE機能が有効な場合、プログラムの正常終了または異常終了にかかわらず、トレース情報は生成されます。

・ トレース情報ファイルが存在している状態で、再度プログラムを実行した場合、元のトレース情報ファイルの内容は失われます。

・ トレース情報ファイルが不要になった場合には、削除してください。

・ NetCOBOL Studioのデバッグ機能と併用する場合、例外通知メッセージが出力されないことがあります。

・ トレース情報ファイルには、プロトタイプ宣言されたメソッドであることが識別できる情報を出力しません。

  メソッドのステートメント番号を参照する場合、クラス名とメソッド名を参照して、プロトタイプ宣言により「分離されたメソッド」であるか、そうでないかを確認してください。「分離されたメソッド」の場合、ステートメント番号は、「分離されたメソッド」のソースファイルの行番号で表現します。クラス定義のソースファイルの行番号ではありません。

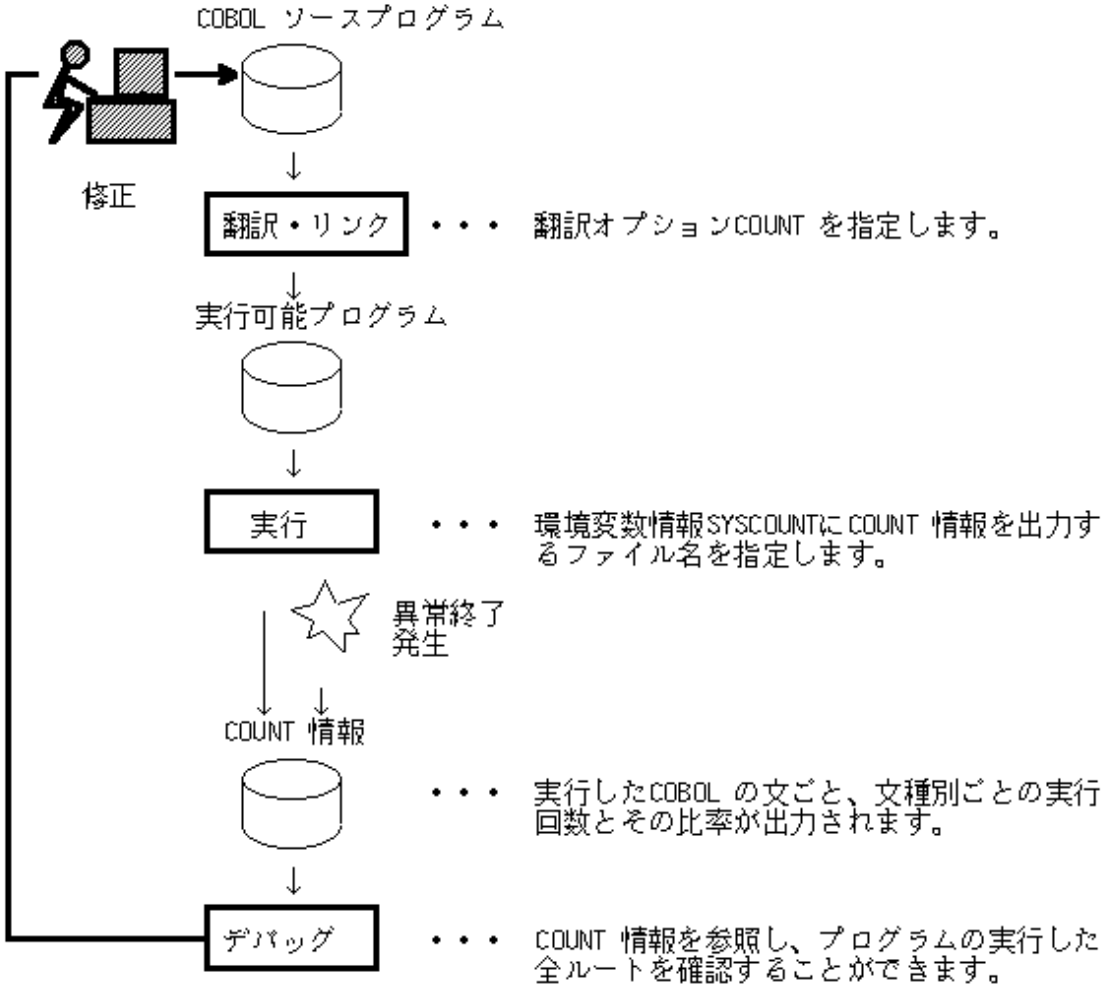
・ トレース情報ファイルは、実行可能ファイルのプロセス毎に出力されます。
  複数のプロセスから、同時に同じファイルへ出力することはできません。出力した場合は、実行時にエラーになります。
  同じ名前の実行可能プログラムを複数同時に実行する場合は、プロセス毎にトレース情報の出力ファイル名を変える必要があります。

19.4 COUNT機能

COUNT機能は、ソースプログラム上に書かれた各文の実行回数と全文の実行回数に対する各文の実行回数比率を表示する機能です。また、ソースプログラム中に書かれた文種別ごとの実行回数とその比率などを表示します。利用者は、COUNT機能により、各文の実行頻度を的確に把握し、プログラムの最適化に役立てることができます。

19.4.1 デバッグ作業の流れ

以下に、COUNT機能を使ったデバッグ作業の流れを示します。



19.4.2 COUNT情報

翻訳オプションCOUNTが有効な場合、環境変数情報SYSCOUNTに指定されたファイルに情報が出力されます。[参照]“A.3.8 COUNT（COUNT機能の使用の可否）”、“C.2.84 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]                                          [5]          [6]
      STATEMENT       [4]                          EXECUTION    PERCENTAGE
        NUMBER        PROCEDURE-NAME/VERB-ID         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
                                          [11]                        [13]
    [8]         [9]          [10]       PERCENTAGE [12]           PERCENTAGE
    VERB-ID     ACTIVE VERB  TOTAL VERB       (%)  EXECUTION COUNT      (%)
   ------------ ------------ ------------ ---------  ---------------- --------
   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)
                                          [11]                        [13]
    [8]         [9]          [10]       PERCENTAGE [12]           PERCENTAGE
    VERB-ID     ACTIVE VERB  TOTAL VERB       (%)  EXECUTION COUNT      (%)
   ------------ ------------ ------------ ---------  ---------------- --------
   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
                                          [18]                        [20]
    [15]        [16]         [17]       PERCENTAGE [19]           PERCENTAGE
    PROGRAM-NAME ACTIVE VERB TOTAL VERB       (%)  EXECUTION COUNT      (%)
   ------------ ------------ ------------ ---------  ---------------- --------
   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]                                        [5]          [6]
      STATEMENT       [4]                      EXECUTION    PERCENTAGE
        NUMBER        PROCEDURE-NAME/VERB-ID     COUNT           (%)
    ------------  -------------------------------  ----------------  --------
              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
                                          [11]                        [13]
    [8]         [9]          [10]       PERCENTAGE [12]           PERCENTAGE
    VERB-ID     ACTIVE VERB  TOTAL VERB       (%)  EXECUTION COUNT      (%)
   ------------ ------------ ------------ ---------  ---------------- --------
   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
```

```
                                    [18]                    [20]
 [15]        [16]         [17]      PERCENTAGE [19]         PERCENTAGE
METHOD-NAME  ACTIVE VERB  TOTAL VERB      (%)  EXECUTION COUNT      (%)
------------ ----------- ------------ --------  --------------- --------
COUNT-METHOD           2            2 100.0000               2 100.0000
------------ ----------- ------------ --------  --------------- --------
                       2            2 100.0000               2
[21]
PROGRAM/CLASS/PROTOTYPE METHOD EXECUTION COUNT
 [22]                               [25]                    [27]
 PROGRAM/CLASS [23]        [24]     PERCENTAGE [26]         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メソッド名を表示します。

[23] [16]を参照してください。

[24] [17]を参照してください。

[25] [18]を参照してください。

[26] 各翻訳単位の文実行回数を表示します。最後に合計を表示します。

[27] 各翻訳単位の全体に対する実行文数の比率を、以下で求め、表示します。

各翻訳単位の文実行回数÷全翻訳単位の文実行回数合計×100

19.4.3 COUNT機能を使用したプログラムのデバッグ

COUNT機能を利用して行うことのできるプログラムのデバッグ例を以下に示します。

プログラムの全ルート走行の確認

COUNT機能の出力リストには実行された文の実行回数が表示されるので、これを調べることにより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により、プログラム中で頻繁に使用される部分を探すことができます。こうした部分の文を適正化することにより、プログラム全体の効率化を図ることができます。

19.4.4 注意事項

ここでは、COUNT機能使用時の注意事項について説明します。

- COUNT機能では、COUNT情報の採取など、COBOLプログラムで記述した以外の処理を行います。そのため、COUNT機能使用時には、プログラムのサイズが大きくなり、実行速度も遅くなります。COUNT機能は、デバッグ時にだけ使用し、デバッグ終了後は、翻訳オプションCOUNTを指定しないで再翻訳してください。
- 異常終了時の出力ファイルには、異常終了の原因となった文も含まれて出力されています。
- CANCEL文実行時には、取り消されるプログラムのCOUNT情報を出力します。取り消されるプログラムからさらに呼ばれるプログラムがあるとき、そのプログラムのCOUNT情報は、呼ぶプログラムで出力されます。
- 出力ファイル名を定義するために、環境変数情報SYSCOUNTを指定しなければなりません。
- COUNT情報の出力ファイルを印刷する場合、各ページの先頭に見出しを出力するためには、以下の設定を行ってください。

```
使用するエディタ      : ワードパット
ページ設定            : 用紙サイズ      Ａ４
                        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。

19.5 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。

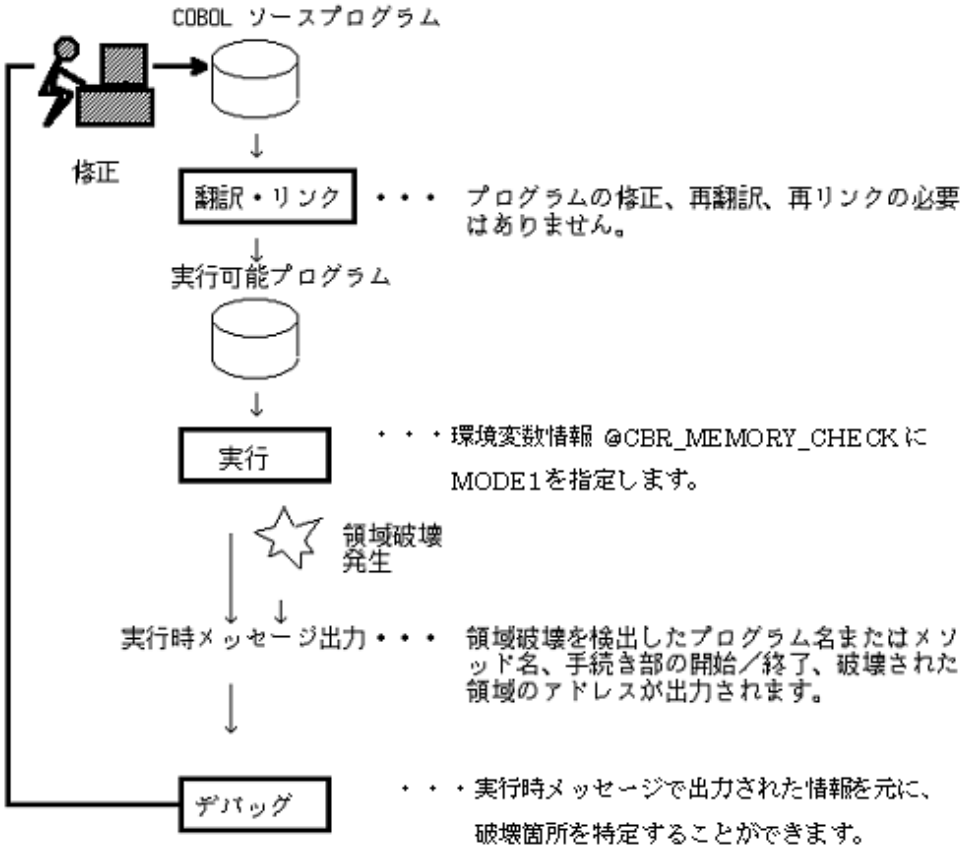
- JMP0009I-U ライブラリ作業域が確保できません.(注)
- JMP0010I-U ライブラリ作業域が破壊されています.
- アプリケーションエラー(アクセス違反)が発生した場合

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 注意事項”の“COBOLプログラムの実行時に仮想メモリ不足が発生する場合”を参照してください。

メモリチェック機能を使用する場合、環境変数情報@CBR_MEMORY_CHECK=MODE1を指定してください。[参照]“C.2.39 @CBR_MEMORY_CHECK（メモリチェック機能を使って検査を行う指定）”

19.5.1 デバッグ作業の流れ

以下に、メモリチェック機能を使ったデバッグ作業の流れを示します。



領域破壊を起こしたプログラムの特定方法は、“19.5.3 プログラムの特定”を参照してください。

19.5.2 出力メッセージ

メモリチェック機能では、領域破壊を検出すると以下のメッセージを出力します。メッセージは、通常メッセージボックスに出力されます。

メモリチェック機能のメッセージについて、以下に説明します。

19.5.2.1 メッセージの内容

プログラムまたはメソッドの手続き部の開始で領域破壊を検出した場合

JMP0071I-U
[PID:xxxxxxxx TID:xxxxxxxx] 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

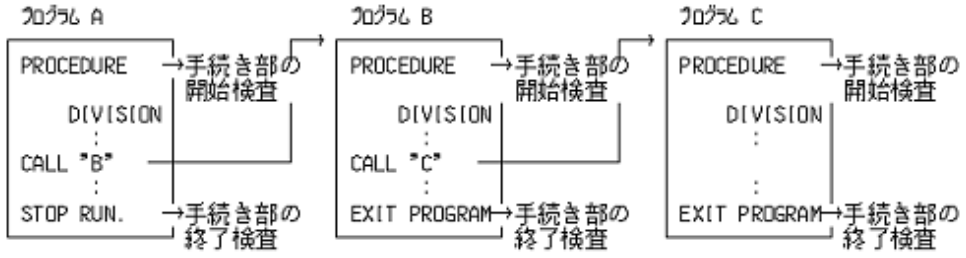
JMP0071I-U
[PID:xxxxxxxx TID:xxxxxxxx] 領域破壊を検出しました. END PGM=プログラム名 BRKADR=破壊された領域の先頭アドレス

メソッドで領域破壊を検出した場合、“PGM=プログラム名”は“CLASS=クラス名 METHOD=メソッド名”となります。

19.5.3 プログラムの特定

領域破壊を起こしたプログラムの特定方法を以下に示します。

次のような呼出し関係で、領域破壊のメッセージが出力されたとします。




```
JMP0071I-U [PID:00000010 TID:0000000E] 領域破壊を検出しました。
        START PGM=C BRKADR=0x00000000 00202000
```

メモリチェック機能は、プログラムの手続き部の開始/終了でランタイムシステム領域の検査を行います。この場合、COBOLプログラムAの手続き部の開始検査、COBOLプログラムBの手続き部の開始検査では領域破壊は検出されず、COBOLプログラムCの手続き部の開始検査で領域破壊が検出されたことになります。よって、COBOLプログラムBの手続き部の開始検査以降からCOBOLプログラムCを呼び出すまでに領域破壊が発生したことになります。

19.5.4 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。

第20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)
- 異常終了した文の特定とデータ参照（翻訳リストとデバッグツールを使ったデバッグ)

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。 ・ エラーの種別（例外コードまたは実行時メッセージ） ・ 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） ・ 呼出経路 ・ システム情報 ・ 環境変数 ・ 実行環境情報 ・ プロセス一覧 ・ モジュール一覧 ・ スレッド情報 また、ダンプを出力します。 【用途】 ・ 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 ・ 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 ・ 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	TEST
翻訳リストとデバッグツールを使ったデバッグ	次の翻訳リストを出力します。 ・ 目的プログラムリスト ・ データマップリスト ・ ソースプログラムリスト 【用途】 ・ アプリケーションを異常終了させた文の特定と異常終了時のデータ内容を参照したい場合	LIST -P SOURCE COPY MAP

20.1 診断機能

ここでは、診断機能の概要や使い方について説明します。

20.1.1 診断機能の概要

診断機能は、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を診断し、診断レポートを出力します。診断レポートには、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が、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されます。

また、以下の問題に対しては、診断レポートと合わせてダンプを出力します。

- スタックオーバーフロー例外(0xC00000FD)を除くアプリケーションエラー

- 以下のUレベル実行時メッセージ
  - JMP0009I-U
  - JMP0010I-U
  - JMP0370I-U

診断機能は、アプリケーションのデバッグに使用できるだけでなく、運用時にアプリケーションエラーや実行時メッセージが発生したときのトラブルシューティングにも使用することができます。

診断機能には、以下の特徴があります。

- 運用モジュールをそのまま対象としますので、本機能を使用するためにアプリケーションを作成し直す必要がありません。
- デバッグ情報ファイルおよびプログラムデータベースファイルがあると、COBOL言語レベルの情報を出力することができます。

20.1.2 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。

20.1.2.1 対象プログラム

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)

20.1.2.2 プログラムの作成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20.1 COBOLプログラムと出力情報の関係”に示します。

表20.1 COBOLプログラムと出力情報の関係

手順	詳細		組み合わせ条件					
			[1]	[2]	[3]	[4]	[5]	[6]
翻訳	TESTオプションの指定		なし	なし	あり	あり	あり	あり
リンク	/DEBUG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	あり	なし	なし	あり
	プログラムデータベースファイルの有無		なし	あり	なし	なし	あり	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	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	あり	あり

参照

EXPORT名やシンボル名の詳細については、“20.1.4.2 診断レポートの出力情報”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場合(条件[1])、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ます。したがって、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およびプログラムデータベースファイルを参照できる状態(条件[6])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20.1 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、診断機能がデバッグ情報ファイルやプログラムデータベース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20.1 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20.1 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-Pを指定して翻訳すると作成されます。[参照]“A.3.23 LIST(目的プログラムリストの出力の可否)”、“J.1.16 -P(各種翻訳リストの出力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
       BEGINNING OF ENTRY POINT CODE
[1] → 000000000000  3100
       000000000002  0800
       000000000004  53414D50444C4C32
                     "SAMPDLL2"
       00000000000C  08
       00000000000D  4E6574434F424F4C
                     "NetCOBOL"
       000000000015  07
       000000000016  5631302E322E30
                     "V10.2.0"
       00000000001D  3230313030323231
```

```
                    "20100221"
        000000000025  3133303631343030
                    "13061400"
        00000000002D  2B30303030
        000000000032  30313031
        000000000036  CCCC
                                GLB.1
                                SAMPDLL2:
[2] →   000000000038  48894C2408              mov       qword ptr [rsp+0x08],rcx
        00000000003D  4889542410              mov       qword ptr [rsp+0x10],rdx
        000000000042  4C89442418              mov       qword ptr [rsp+0x18],r8
        000000000047  4C894C2420              mov       qword ptr [rsp+0x20],r9
        00000000004C  55                      push      rbp
        00000000004D  4889E5                  mov       rbp,rsp

    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。

20.1.2.3 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、出力される情報量が限定されるため、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。[参照]“A.3.52 TEST（デバッグ機能および診断機能の使用の可否）”

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、NetCOBOL Studioのデバッグ機能では使用できません。

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。このオプションを指定すると、リンカはオブジェクトファイル内に出力された診断機能に必要な情報を、プログラムデータベースファイルに出力します。

20.1.2.4 デバッグ情報ファイルおよびプログラムデータベースファイルの格納先

COBOL言語レベルの情報で診断レポートを出力させるには、デバッグ情報ファイルおよびプログラムデータベースファイルをモジュール(実行可能ファイル、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、運用マシンに運用モジュールとともにデバッグ情報ファイルおよびプログラムデータベースファイルも一緒に格納してください。

なお、これらのファイルが存在しない場合には、診断機能はEXPORT名やEXPORT名相対オフセットなどのアセンブラレベルの情報を診断レポートに出力します。

診断機能の出力情報については、“20.1.2.2 プログラムの作成方法と出力情報との関係”を参照してください。

20.1.3 診断機能の起動

20.1.3.1 起動方法

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知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します。[参照]“C.2.35 @CBR_JUSTINTIME_DEBUG(異常終了時に診断機能を使って調査を行う指定)”

20.1.3.2 起動パラメタ

診断機能の動作は、起動パラメタを指定することによって制御することができます。起動パラメタを“表20.2 起動パラメタ”に示します。

起動パラメタは、環境変数情報@CBR_JUSTINTIME_DEBUGに指定します。起動パラメタの指定形式の例を以下に示します。例では、診断レポート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
```

表20.2 起動パラメタ

指定形式	内容
-d{YES｜NO}	ダンプを出力するかどうかを指定します。 ・ YES :ダンプを出力する ・ NO :ダンプを出力しない 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。 ・ 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 ・ 1 :環境変数の情報および初期化ファイル名を出力する ・ 0 :環境変数および初期化ファイル名を出力しない 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、“20.1.5.2 ダンプの出力先”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、“20.1.4.1 診断レポートの出力先”を参照してください。

指定形式	内容
-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。
-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。 ・ 0 :スタックのすべての内容を出力する ・ 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 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。

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。
- パラメタ名とオペランドの間は、空白を置いても置かなくても構いません。
- 複数個の起動パラメタを指定する場合には、起動パラメタと次の起動パラメタとの間に1つ以上の空白を置いて指定します。
- 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。

参考

起動パラメタ"-l"が指定されていない場合には、問題が発生したコンピュータにメッセージボックスで通知します。

20.1.4 診断レポート

20.1.4.1 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              : A94
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_A94_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20.2 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 x64、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、“表20.3 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の対応”のように対応付けています。

表20.3 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の対応

重大度コード	イベントログの種類
I(INFORMATION)	情報(INFORMATION)
W(WARNING)	警告(WARNING)
E(ERROR)	
U(UNRECOVERABLE)	エラー(ERROR)

事象発生通知の標準の出力先は、メッセージボックスです。出力先を変更する場合は、起動パラメタ“-l”を指定します。起動パラメタの詳細については、“表20.2 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“-l”を指定して、事象発生の通知先にイベントログを指定してください。

20.1.4.2 診断レポートの出力情報

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：

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20.4 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。

発生時間

エラーが発生した時間を出力します。

発生モジュール

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。

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

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

プログラムデータベースファイルが存在する場合に、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
```

翻訳情報

COBOLプログラムの場合に、プログラム翻訳時の情報を出力します。他言語プログラムの場合は、出力しません。

- コード系(ShiftJISまたはUnicode)
- オブジェクト形式(シングルスレッドまたはマルチスレッド)
- 最適化(OPTIMIZEまたはNOOPTIMIZE)

プログラム名

COBOLプログラムの場合は、外部プログラム名またはクラス名、および内部プログラム名またはメソッド名を出力します。他言語プログラムの場合は、関数名を出力します。

ソースファイル名

COBOLプログラムの場合に、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。他言語プログラムの場合は、出力しません。

行番号

COBOLプログラムの場合に、エラーが発生した文の行番号を出力します。他言語プログラムの場合は、出力しません。

補足情報

COBOLプログラムの場合で、かつエラー発生位置が宣言節手続き内の文であった場合に、宣言節への分岐元の文を出力します。エラー発生位置が宣言節手続き内の文でない場合には出力しません。

呼出経路

エラーが発生したプログラムまでの呼出経路を出力します。なお、デバッグ情報ファイルが存在しない場合は、内部プログラムの呼出し元は出力しません。

システム情報

コンピュータに関する一般的なシステム情報を出力します。

- コンピュータ名
- ユーザ名
- Windowsバージョン
- バージョンナンバー
- サービスパック

コマンドライン

アプリケーションの実行時に指定されたコマンド文字列を出力します。

環境変数

アプリケーションの実行時に設定されていた環境変数を出力します。

実行環境情報

COBOLの環境情報として以下の情報を出力します。なお、実行用の初期化ファイルが存在しない場合は、初期化ファイル名とプログラム名に“なし”と出力します。また、実行用の初期化ファイルが存在している場合でかつプログラムがマルチスレッドで動作している場合には、プログラム名に“なし”と出力します。

ランタイムシステム

COBOLランタイムシステムのバージョンとモジュールタイプを出力します。

実行モード

ランタイムシステムの実行時の情報を出力します。

- ー コード系(ShiftJISまたはUnicode)
- ー 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

- ー エントリ情報ファイル
- ー 論理宛先定義ファイル
- ー ODBC情報ファイル
- ー 印刷情報ファイル

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

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

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。

- 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20.4 例外コードと例外名”に示します。

表20.4 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取りまたは書込みをしようとしましたが、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ました。
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ました。
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で除算しようとしました。
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で除算しようとしました。
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ましたが、その演算は現在のマシンモードでは許可されていません。
C00000FD	EXCEPTION_STACK_OVERFLOW
	スタックオーバーフローが発生しました。

COBOLアプリケーションで発生しやすいアプリケーションエラーは、アクセス違反(EXCEPTION_ACCESS_VIOLATION)やゼロ除算(EXCEPTION_INT_DIVIDE_BY_ZERO)です。アクセス違反には、以下の原因が考えられます。

- 添字や指標の値が範囲外である。
- 部分参照で参照可能範囲外を参照する。
- プログラムの呼出し規約やパラメタの個数が一致していない。

これらはCHECK機能で簡単に検査することができるため、エラーが発生した場合はCHECK機能で確認してください。[参照]"19.2 CHECK機能"

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20.5 翻訳オプションと出力形式"に示します。

表20.5 翻訳オプションと出力形式

出力対象		NOOPTIMIZE指定	OPTIMIZE指定	
		NUMBER/NONUMBER指定共通	NUMBER指定	NONUMBER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)

20.1.5 ダンプ

20.1.5.1 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Uレベルの実行時メッセージが発生したときのメモリの内容をダンプファイルに出力します。

ダンプの出力を抑止する場合には、起動パラメタ“-d NO”を指定します。

ポイント

ダンプがないと、エラーの原因を特定するまでに時間を要し、問題が長期化する場合があります。ダンプの出力を有効にしておくことをおすすめします。

ダンプは、WinDbg(Microsoft社が提供しているデバッグツール)などのデバッ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。

20.1.5.2 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。

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
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。

```
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20.2 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“表20.2 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

ー 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

20.1.5.3 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。

20.1.6 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。

  ー Visual C++のジャストインタイムデバッガ

  ー 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NetCOBOL Studioのデバッグまたは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。

- 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。

  ー 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るため、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。

  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 PROGARM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 PROGARM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。

20.2 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

ー 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。

ー 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

ー WinDbg (Debugging Tools for Windows - ネイティブ x64)

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 x64は、Microsoftのホームページで情報公開されています。

ー 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

ー 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。

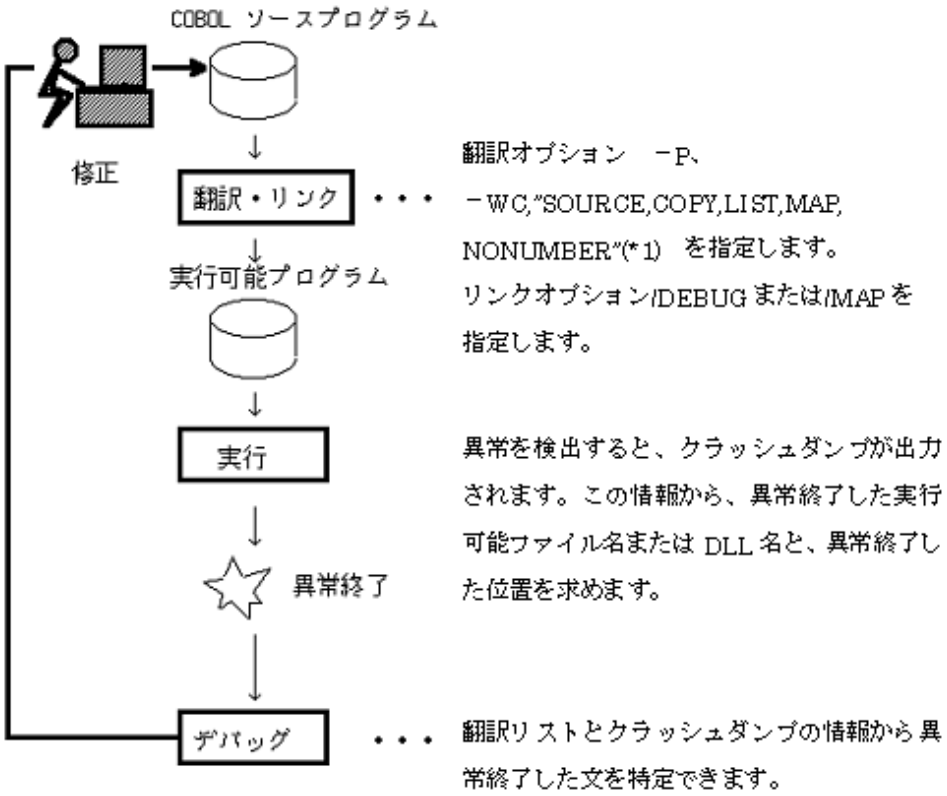
- プログラムデータベースファイル(.PDB)(/DEBUGオプション指定時に出力)
- リンクマップファイル(.MAP)(/MAPオプション指定時に出力)

ー 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

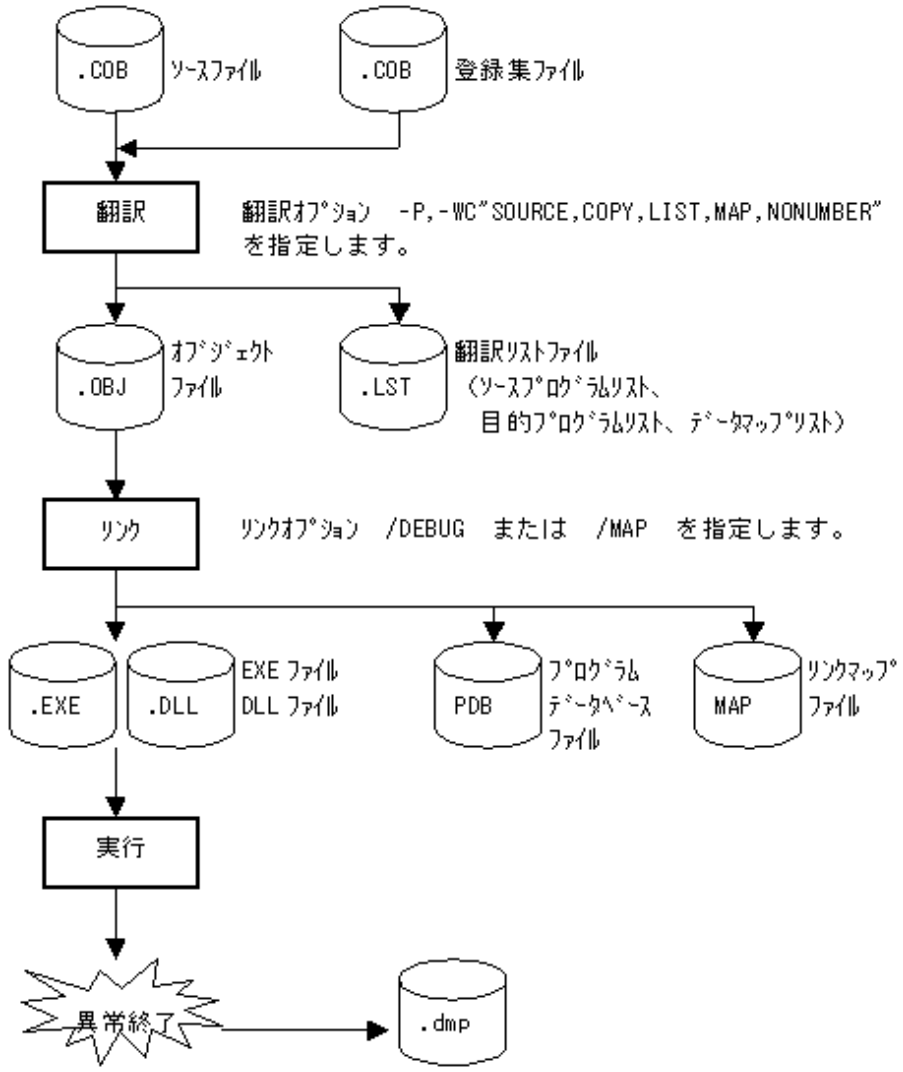
20.2.1 デバッグ作業の流れ

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、“20.1.5 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。WinDdbg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。

20.2.2 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 x64)

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。

   a. 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。

   b. 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。

   a. WinDbgを起動します。

   b. 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:¥werdump¥LISTDBG.exe.3584.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 Server 2008/Windows Vista SP1 Version 6001 (Service Pack 1) MP (2 procs) Free x64
Product: Server, suite: Enterprise TerminalServer
Machine Name:
Debug session time: Fri Dec  5 17:46:58.000 2008 (GMT+9)
System Uptime: 10 days 7:46:11.034
Process Uptime: 0 days 0:00:03.000
.....................................
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntdll.dll -
This dump file has an exception of interest stored in it.
The stored exception information can be accessed via .ecxr.
(e00.f34): Integer divide-by-zero - code c0000094 (first/second chance not …)  ... [1]
*** WARNING: Unable to verify checksum for LISTDBG.exe
LISTDBG!DIVPROC+0x431:                                                  ... [3]
00000001`40001b29 49f7fc          idiv    rax,r12                       ... [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“00000001`40001b29”の“idiv rax,r12”命令であることがわかります。

“idiv rax,r12”命令は、符号付き割算を行う命令であり、rdx:raxレジスタの内容をr12レジスタの内容で割り、商をraxに、余りをrdx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
rax=0000000000000002 rbx=00000000030354e0 rcx=0000000003035628
rdx=0000000000000000 rsi=0000000000000000 rdi=000000000012fbf0
rip=0000000140001b29 rsp=000000000012f9f0 rbp=000000000012fc60
 r8=0000000140024670  r9=00000000030354e0 r10=0000000000000a90
r11=000000000012fb2a r12=0000000000000000 r13=0000000000000002
r14=000000014001d1a0 r15=0000000003035240
iopl=0         nv up ei pl zr na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010244
LISTDBG!DIVPROC+0x431:
00000001`40001b29 49f7fc          idiv    rax,r12
```

上記の出力から、“rdx:rax=0000000000000002”を“r12=0000000000000000”で割算を行おうとしたが、r12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+0x431”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  : 0x431
```

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
00000001`40000000 00000001`40025000 LISTDBG C (private pdb symbols)  C:¥Work¥tel¥LISTDBG.pdb
    Loaded symbol image file: LISTDBG.exe
    Image path: C:¥Work¥tel¥LISTDBG.exe                                    ... [4]
    Image name: LISTDBG.exe                                                ... [5]
    Timestamp:        Fri Dec 05 17:46:44 2008 (4938EA74)
    CheckSum:         00000000
    ImageSize:        00025000
    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+0x431”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、“0x431”の位置を求めます。

[目的プログラムリスト(DIVPROC.LST)]

```
  番地         機械語          手続き名   アセンブラ形式命令


BEGINNING OF DIVPROC
BEGINNING OF ENTRY POINT CODE
000000000000  3000
000000000002  0700
000000000004  44495650524F43
              "DIVPROC"
00000000000B  08
00000000000C  4E6574434F424F4C
              "NetCOBOL"
000000000014  07
000000000015  5631302E322E30
              "V10.2.0"
00000000001C  3230313030323231
              "20100221"
000000000024  3137343132303030
              "17412000"
00000000002C  2B30303030
000000000031  30313031
```

```
000000000035  CCCCCC
                          GLB.1
                          DIVPROC:                                        ... [6]
000000000038  48894C2408              mov     qword ptr [rsp+0x08],rcx  ... [7]
00000000003D  4889542410              mov     qword ptr [rsp+0x10],rdx
000000000042  4C89442418              mov     qword ptr [rsp+0x18],r8
000000000047  4C894C2420              mov     qword ptr [rsp+0x20],r9
00000000004C  55                      push    rbp
00000000004D  4889E5                  mov     rbp,rsp
000000000050  57                      push    rdi
000000000051  56                      push    rsi
000000000052  53                      push    rbx
000000000053  4154                    push    r12
000000000055  4155                    push    r13
000000000057  4156                    push    r14
000000000059  4157                    push    r15
00000000005B  4881EC38020000          sub     rsp,0x00000238
000000000062  488DBC24E0000000        lea     rdi,[rsp+0x000000E0]
00000000006A  48C7C124000000          mov     rcx,0x00000024
000000000071  48C7C000000000          mov     rax,0x00000000
000000000078  FC                      cld
000000000079  F348AB                  rep stosq
00000000007C  488B4510                mov     rax,qword ptr [rbp+0x10]
000000000080  48898424E0010000        mov     qword ptr [rsp+0x000001E0],rax
000000000088  48B8A203000000000000    mov     rax,0x00000000000003A2  PROCEDURE_ENTRY
000000000092  48898424D8010000        mov     qword ptr [rsp+0x000001D8],rax
00000000009A  E9B9000000              jmp     000000000158                    GLB.2
END OF ENTRY POINT CODE   :
```

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。そのシンボルの次の行の番地からそのシンボルのオフセット位置[7]の“000000000038”(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“0x431”を加算し、リスト内のオフセット値を求めます。

```
0x38+0x431 = 0x469
```

DIVPROC.cobの目的プログラムリスト(DIVPROC.LST)から番地が“0x469”を検索します。

[目的プログラムリストDIVPROC.LST)]

```
--- 14 ---            MOVE
0000000003FE  66C78424F00100003030  mov     word ptr [rsp+0x000001F0],0x3030  答え
--- 15 ---            COMPUTE                                             ... [9]
000000000408  4C8BBB98010000        mov     r15,qword ptr [rbx+0x00000198]   BVA.1
00000000040F  410FB65701            movzx   edx,byte ptr [r15+0x01]          被除数+1
000000000414  490FB607              movzx   rax,byte ptr [r15]               被除数
000000000418  4883E00F              and     rax,0x0F
00000000041C  488D0480              lea     rax,[rax+rax*4]
000000000420  4883E20F              and     rdx,0x0F
000000000424  488D0442              lea     rax,[rdx+rax*2]
000000000428  66898424C6000000      mov     word ptr [rsp+0x000000C6],ax     TRLP+0
000000000430  410FB65703            movzx   edx,byte ptr [r15+0x03]          除数+1
000000000435  490FB64702            movzx   rax,byte ptr [r15+0x02]          除数
00000000043A  4883E00F              and     rax,0x0F
00000000043E  488D0480              lea     rax,[rax+rax*4]
000000000442  4883E20F              and     rdx,0x0F
000000000446  488D0442              lea     rax,[rdx+rax*2]
00000000044A  66898424D6000000      mov     word ptr [rsp+0x000000D6],ax     TRLP+0
000000000452  4C0FBFAC24C6000000    movsx   r13,word ptr [rsp+0x000000C6]    TRLP+0
00000000045B  4C0FBFA424D6000000    movsx   r12,word ptr [rsp+0x000000D6]    TRLP+0
000000000464  4C89E8                mov     rax,r13
000000000467  4899                  cqo
000000000469  49F7FC                idiv    r12                              ... [8]
00000000046C  4989C6                mov     r14,rax
```

```
00000000046F  4C8B9BA8010000          mov     r11,qword ptr [rbx+0x000001A8]   BCO.0
000000000476  4D8B6310                mov     r12,qword ptr [r11+0x10]         #
00000000047A  4C89F0                  mov     rax,r14
00000000047D  4C89F2                  mov     rdx,r14
000000000480  4C89F1                  mov     rcx,r14
000000000483  48F7D9                  neg     rcx
000000000486  4983FE00                cmp     r14,0x00
00000000048A  480F4CC1                cmovnge rax,rcx
00000000048E  4C39E0                  cmp     rax,r12
000000000491  7C08                    jl      00000000049B                     GLB.11
000000000493  4C89F0                  mov     rax,r14
000000000496  4899                    cqo
000000000498  49F7FC                  idiv    r12
                          GLB.11
00000000049B  4989D5                  mov     r13,rdx
00000000049E  664489AC24C6000000      mov     word ptr [rsp+0x000000C6],r13w   TRLP+0
0000000004A7  480FBF8424C6000000      movsx   rax,word ptr [rsp+0x000000C6]    TRLP+0
0000000004B0  4889C6                  mov     rsi,rax
0000000004B3  48C1FE3F                sar     rsi,0x3F
0000000004B7  4883FE00                cmp     rsi,0x00
0000000004BB  790A                    jns     0000000004C7                     GLB.12
0000000004BD  48F7DE                  neg     rsi
0000000004C0  48F7D8                  neg     rax
0000000004C3  4883DE00                sbb     rsi,0x00
                          GLB.12
0000000004C7  488D8C24F0010000        lea     rcx,[rsp+0x000001F0]             答え
0000000004CF  48C7C202000000          mov     rdx,0x00000002
0000000004D6  4989F0                  mov     r8,rsi
0000000004D9  4989C1                  mov     r9,rax
0000000004DC  488BBBB8010000          mov     rdi,qword ptr [rbx+0x000001B8]
0000000004E3  FF5778                  call    qword ptr [rdi+0x78]
```

番地が“000000000469”の位置([8]の命令)が「idiv r12」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。([9])

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。([10])

[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.                    ... [10]
        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特定できました。

20.2.3 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 x64)

使用する例は、“20.2.2 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、“20.2.2 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:¥werdump¥LISTDBG.exe.3048.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 Server 2008/Windows Vista SP1 Version 6001 (Service Pack 1) MP (2 procs) Free x64
Product: Server, suite: Enterprise TerminalServer
Machine Name:
Debug session time: Mon Dec  8 09:25:24.000 2008 (GMT+9)
System Uptime: 12 days 23:24:36.422
Process Uptime: 0 days 0:00:03.000
.....................................
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntdll.dll -
This dump file has an exception of interest stored in it.
The stored exception information can be accessed via .ecxr.
(be8.504): 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 -
LISTDBG!DIVPROC+0x431:
00000001`400019a9 49f7fc          idiv    rax,r12                           ... [1]
```

異常終了した位置が“00000001`400019a9”([1])であることがわかります。

b. [1]のアドレス“00000001`400019a9”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
00000000`00020000 00000000`00030000  F4AGLANP  (deferred)
00000000`00190000 00000000`001c5000  F4AGSQL   (deferred)
00000000`001d0000 00000000`00203000  F4AGAESV  (deferred)
00000000`00210000 00000000`00249000  F4AGEFNC  (deferred)
00000000`006e0000 00000000`00754000  F4AGIO    (deferred)
00000000`00760000 00000000`007a4000  F4AGFRM   (deferred)
00000000`007b0000 00000000`00841000  F4AGLPIO  (deferred)
00000000`00850000 00000000`008b1000  F4AGDBG   (deferred)
00000000`008c0000 00000000`00904000  F4AGPRIO  (deferred)
00000000`00910000 00000000`0098b000  F4AGSCRN  (deferred)
00000000`00990000 00000000`00a11000  F4AGOLER  (deferred)
00000000`00a20000 00000000`00a69000  F4AGOLES  (deferred)
00000000`00a70000 00000000`00abd000  F4AGARMV  (deferred)
00000000`00e40000 00000000`00e50000  F4AGFC64  (deferred)
00000000`10000000 00000000`10005000  F4AGOVLD  (deferred)
00000000`77130000 00000000`7725b000  kernel32  (deferred)
00000000`77260000 00000000`7732d000  user32    (deferred)
00000000`77330000 00000000`774b0000  ntdll     (export symbols)  ntdll.dll
00000001`40000000 00000001`40010000  LISTDBG C (export symbols)  LISTDBG.exe ... [2]
00000001`80000000 00000001`800ea000  F4AGPRCT  (deferred)
000007fe`f9680000 000007fe`f96d8000  winspool  (deferred)
000007fe`faf40000 000007fe`fafe0000  comctl32  (deferred)
000007fe`fc180000 000007fe`fc379000  comctl32_7fefc180000  (deferred)
000007fe`fca70000 000007fe`fca7b000  version   (deferred)
000007fe`fda80000 000007fe`fdc58000  ole32     (deferred)
000007fe`fdc60000 000007fe`fdc77000  imagehlp  (deferred)
000007fe`fdc80000 000007f 0x140000000 =e`fdce3000  gdi32   (deferred)
000007fe`fdde0000 000007fe`fdee1000  msctf     (deferred)
000007fe`fdef0000 000007fe`fdf7c000  comdlg32  (deferred)
000007fe`fdf80000 000007fe`fdfad000  imm32     (deferred)
000007fe`fe0b0000 000007fe`fe0bd000  lpk       (deferred)
000007fe`fe0c0000 000007fe`fe133000  shlwapi   (deferred)
000007fe`fe1a0000 000007fe`fe23c000  msvcrt    (deferred)
000007fe`fe240000 000007fe`fee92000  shell32   (deferred)
000007fe`ff080000 000007fe`ff1bf000  rpcrt4    (deferred)
000007fe`ff340000 000007fe`ff413000  oleaut32  (deferred)
000007fe`ff480000 000007fe`ff588000  advapi32  (deferred)
000007fe`ff590000 000007fe`ff62a000  usp10     (deferred)
```

[1]の“00000001`400019a9”は、[2]の“00000001`40000000 00000001`40010000

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か特定します。

[リンクマップリスト(LISTDBG.MAP)]

```
LISTDBG
 Timestamp is 493c694d (Mon Dec 08 09:24:45 2008)
 Preferred load address is 0000000140000000                              ... [3]
 Start         Length     Name                   Class
 0001:00000000 000060f2H .text                   CODE
  ～ 省略 ～
 0005:00000000 000005d8H .rodata                 DATA
  Address         Publics by Value          Rva+Base           Lib:Object
 0000:00000000    ___safe_se_handler_count  0000000000000000   <absolute>
```

```
0000:00000000    ___safe_se_handler_table  0000000000000000    <absolute>
0000:00000000    __ImageBase               0000000140000000    <linker-defined>
0001:00000000    main                      0000000140001000 f  LISTDBG.obj
0001:00000050    LISTDBG                   0000000140001050 f  LISTDBG.obj
0001:00000578    DIVPROC                   0000000140001578 f  DIVPROC.obj  ... [4]
0001:00000a50    JMP1PLAN                  0000000140001a50 f  f4agcimp:F4AGPRCT.dll
 ～ 省略 ～
```

[2]のモジュールのローディングアドレス“00000001`40000000”と、[3]の“Preferred load address is 0000000140000000”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“00000001`400019a9”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] “0001:00000580 DIVPROC 0000000140001580 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。

```
0x1400019a9 - 0x140001578 = 0x431
```

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: 0x431
```

－ [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。

```
0x1400019a9 - 0x1400000000 = 0x19a9
```

2. それぞれのシンボルの相対オフセットを求めます。

```
main      0x140001000 - 0x140000000 = 0x1000
LISTDBG   0x140001050 - 0x140000000 = 0x1050
DIVPROC   0x140001578 - 0x140000000 = 0x1578   ... [5]
JMP1PLAN  0x140001a50 - 0x140000000 = 0x1a50
      :
```

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  0x140001578 - 0x1400000000 = 0x1578”
```

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。

```
0x19a9 - 0x1578 = 0x431
```

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: 0x431
```

これ以降の作業は、“20.2.2 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。

20.2.4 異常終了時のデータの参照方法

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 x64)

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と、呼び出し元のデータです。「Call Stack」に現れないプログラムまたはメソッドのデータは参照することはできません。

データ域の説明

基本となるデータ域は次の4つです。

- スタックデータ
- ヒープデータ
- .data
- .rdata

また、メソッド定義の場合は、以下のデータ域も使用します。

- .rodata
- オブジェクトデータまたはファクトリデータが使用するヒープデータ

データ域名	表記(注)	説明
スタックデータ	stac	スタックデータは、メソッド定義の作業場所節(WORKING-STORAGE SECTION)に記述したデータおよびコンパイラの生成する作業用・管理用のデータを格納する領域です。 プログラム・メソッドが実行している間だけ使うことができる領域です。 LINK.EXEのオプション(/STACK)でサイズを指定(変更)します。 高い(大きい)アドレスから、低い(小さい)アドレスに拡張されます。 “rsp”レジスタで使用領域と未使用領域の境界を管理します。 COBOLの手続き実行時は“rsp”レジスタを基準に参照します。
ヒープデータ	heap	ヒープデータは、プログラム定義の作業場所節(WORKING-STORAGE SECTION)に記述したデータおよびコンパイラの生成する作業用・管理用のデータを格納する領域です。 COBOLの手続き実行時は“rbx”レジスタを基準に参照します。
.data	data	.dataは読み込み・書き込み可能なデータ域です。コンパイラの生成する作業・管理用のデータを格納する領域です。
.rdata	rdat	.rdataは、読み込み専用のデータ域です。定数節に記述したデータおよびコンパイラの生成する読み込み専用のデータなどの手続き実行中に不変なデータを格納する領域です。
.rodata	―	オブジェクト定義またはファクトリ定義内で共通の読込み専用データを格納する領域です。

データ域名	表記(注)	説明
オブジェクトデータまたはファクトリデータが使用するヒープデータ	hea2	オブジェクト定義またはファクトリ定義内で共通のデータを格納する領域です。

注: “B.6.1 データマップリスト”のデータマップリストの出力形式で説明される「番地」部分に表示される時の表記です。

それぞれのデータ域の先頭は、以下の方法で求めます。

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。
3. 翻訳オプションMAPを指定して出力した翻訳リストを用意し、“プログラム制御情報リスト”を参照します。

```
[プログラム制御情報リスト(DIVPROC.LST)]
  ～省略～
    ** HGWA **
      * HGCB *
heap+00000000          HGCB FIXED AREA                  72

      * HGMB *
heap+00000048          LIA FIXED AREA                  256
     ........          IWA1 AREA                         0
     ........            LINAGE COUNTER                  0
     ........            OTHER AREA                      0
     ........            ALTINX                          0
     ........            PCT                             0
heap+00000148          LCB AREA                         80
heap+00000198          HGMB POINTERS AREA               72
     ........            VPA                             0
     ........            PSA                             0
heap+00000198            BVA                             8
     ........            BEA                             0
     ........            FMBE                            0
heap+000001A0            CONTROL ADDRESS TABLE           64              ... [1]
     ........            MUTEX HANDLE AREA                0
  ～省略～

    ** STK **
      * SCB *
stac+00000000          ABIA                             72
stac+00000050          SCB FIXED AREA                  112
stac+000000C0          TL 1ST AREA                      32
stac+000000E0          LCB AREA                         80
stac+00000130          SGM POINTERS AREA                 8
     ........            VPA                             0
     ........            PSA                             0
     ........            BVA                             0
stac+00000130            BHG                             8              ... [2]
     ........            BOD                             0
  ～省略～
```

4. CONTROL ADDRESS TABLE(制御情報テーブル)([1])の内容を表示させ、それぞれのデータ域の先頭アドレスを求めます。

   まず、heap域の先頭アドレスを求めます。

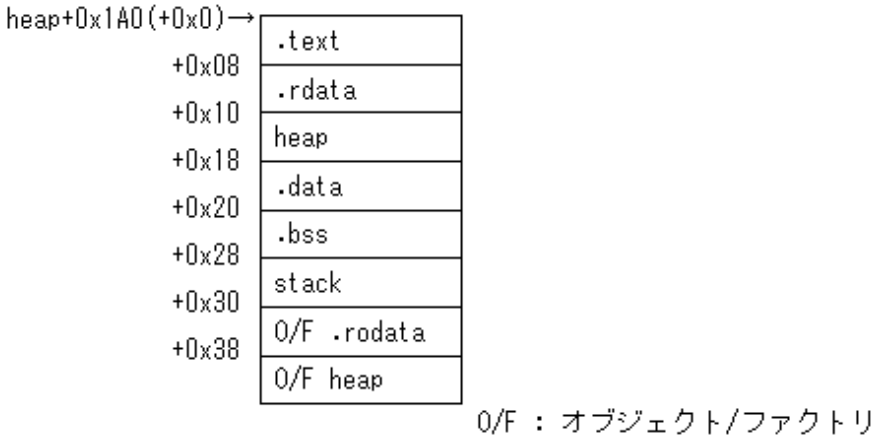
   ー heap域の先頭アドレスの求め方

   手続き実行時は、“rbx”レジスタにheap域の先頭アドレスが設定されます。手続き実行時でない場合、“rbx”レジスタにheap域の先頭アドレスは設定されていません。
   「“rbx”レジスタの値＝heap域先頭アドレス」かどうかは、以下の方法で確認できます。

- [2]のBHG(stac+00000130)に設定されている値と同じかどうかを確認する。
  stac”は“rsp”に設定されているため、“rsp+130”のメモリの内容と“rbx”レジスタの内容をWinDbgで表示させ、比較します。

```
[rbxレジスタの値]
0:000> r rbx
rbx=00000000030354e0

[rsp+00000130のメモリの内容]
0:000> dq rsp+130 L(1)
00000000`0012fb20  00000000`030354e0
```

heap域の先頭アドレスが求められたら、CONTROL ADDRESS TABLE(制御情報テーブル)の内容を表示させます。
CONTROL ADDRESS TABLE(制御情報テーブル)の構造は、以下のようになっています。




```
[CONTROL ADDRESS TABLEのメモリの内容]
0:000> dq rbx+1a0 L(8)
00000000`03035680  00000001`400016c0 00000001`400243a0
                   ~~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~~
                   .text             .rdata
00000000`03035690  00000000`030354e0 00000001`4001d1a0
                   ~~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~~
                   heap              .data
00000000`030356a0  :
```

例えば、.textの先頭アドレスから、メモリの内容を表示させる場合

```
0:000> db 1`400016c0 L(40)
00000001`400016c0  30 00 07 00 44 49 56 50-52 4f 43 08 4e 65 74 43  0...DIVPROC.NetC
00000001`400016d0  4f 42 4f 4c 07 56 31 30-2e 32 2e 30 32 30 31 30  OBOL.V10.2.02010
00000001`400016e0  31 32 30 35 31 37 34 31-32 30 30 30 2b 30 30 30  120517412000+000
00000001`400016f0  30 30 31 30 31 cc cc cc-48 89 4c 24 08 48 89 54  00101...H.L$.H.T
```

異常終了時のデータの参照

ここでは、“20.2.2 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)”で求めたプログラムのデータを参照します。

[異常終了したCOBOLソース行位置]

```
      15    000150      COMPUTE 答え = 被除数 / 除数.
```

異常終了したCOBOLソースの行位置で使用しているデータを参照します。

1. 翻訳オプションMAPを指定して出力した翻訳リストを用意し、“データマップリスト”を参照します。データマップリストの形式は“B.6.1 データマップリスト”を参照してください。

   [データマップリスト(DIVPROC.LST)]

```
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
7  [heap+00000198]+00000000    0 01 引数             4 GROUP-F  BVA.000001
8  [heap+00000198]+00000000    0 02 被除数           2 EXT-DEC  BVA.000001 ... [3]
9  [heap+00000198]+00000002    2 02 除数             2 EXT-DEC  BVA.000001 ... [4]
10  stac+000001F0              0 01 答え             2 EXT-DEC  BST.000000
```

2. それぞれのデータの内容を参照します。

   ー 被除数([heap+00000198]+00000000)のデータの内容

   [heap+00000198]の内容

```
0:000> dq rbx+198 L(1)
00000000`03035678  00000000`03035240
```

   [heap+00000198]+00000000の内容

```
0:000> db 3035240 L(2)
00000000`03035240  30 32                                           02
```

   上記の出力から、“被除数”には、外部10進で“02”が入っていることがわかります。

   ー 除数([heap+00000198]+00000002)のデータの内容

   [heap+00000198]の内容は、“被除数”と同じ。

```
[heap+00000198]+00000002の内容
0:000> db 3035240+2 L(2)
00000000`03035242  00 00
```

   上記の出力から、本来“除数”には、外部10進で値が設定されていなければなりませんが、値を設定していないため、X"0000"が入っており、ゼロだと評価されました。

この確認結果から、ゼロ除算を行ったことがわかります。

20.2.5 異常終了時の呼び出し元のデータ参照と呼び出し箇所の特定

ここでは、呼び出し元のデータ参照と呼び出し箇所の特定について、簡単な例を用いて説明します。

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、LIST、MAPを指定して出力した翻訳リスト
- WinDbg(ebugging Tools for Windows - ネイティブ x64)

呼び出し元のデータ参照

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。
3. WinDbgで呼び出し経路(「Call Stack」)の情報を表示させます。

```
0:000> k
Child-SP          RetAddr           Call Site
00000000`0012f9f0 00000001`40001438 LISTDBG!DIVPROC+0x431        ← 異常終了箇所
00000000`0012fc70 00000001`40001054 LISTDBG!LISTDBG+0x3a8        ← 呼び出し元
00000000`0012fef0 00000001`40001f2b LISTDBG!main+0x14
00000000`0012ff20 00000000`7715495d LISTDBG!__tmainCRTStartup+0x15b
00000000`0012ff60 00000000`77358791 kernel32!BaseThreadInitThunk+0xd
00000000`0012ff90 00000000`00000000 ntdll!RtlUserThreadStart+0x21
```

異常終了した箇所と呼び出し元プログラムがLISTDBGであることがわかります。

4. 呼び出し元プログラムであるLISTDBGのデータを参照します。翻訳オプションMAPを指定して出力した翻訳リスト(LISTDB.LST)を参照します。

[データマップリスト(LISTDBG.LST)]

```
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
6    heap+000001E0            0 01 計算             4 GROUP-F     BHG.000000
7    heap+000001E0            0 02 被除数           2 EXT-DEC     BHG.000000
8    heap+000001E2            2 02 除数             2 EXT-DEC     BHG.000000
9    heap+000001E8            0 01 答え             2 EXT-DEC     BHG.000000
```

[プログラム制御情報リスト(LISTDBG.LST)]

```
heap+00000198         CONTROL ADDRESS TABLE            64
      :
stac+00000120         BHG                               8
```

“20.2.4 異常終了時のデータの参照方法”と同じ方法でそれぞれのデータの内容を参照します。

5. 呼び出し経路から呼び出し元の“Child-SP”(この例では、000000000012fc70)をスタックのアドレスとして、スタックからheapアドレス＝BHG(この例では、stac+00000120)を求めます。

[BHG(heapアドレス)の内容]

```
0:000> dq 12fc70+120 L(1)
00000000`0012fd90  00000000`03035060
```

[CONTROL ADDRESS TABLEのheap]

```
0:000> dq 3035060+198+10 L(1)
00000000`03035208  00000000`03035060
```

6. それぞれのデータの内容を参照します。

   ー 被除数(heap+000001E0)のデータの内容

```
0:000> db 3035060+1e0 L(2)
00000000`03035240  30 32                                          02
```

   ー 除数(heap+000001E2)のデータの内容

```
0:000> db 3035060+1e2 L(2)
00000000`03035242  00 00                                          ..
```

   上記の出力から、本来“除数”には、外部10進で値が設定されていなければなりませんが、値を設定していないため、X"0000"が入っており、ゼロを呼び出し先に渡したことがわかります。

呼び出し箇所の特定

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。
3. WinDbgで呼び出し経路(「Call Stack」)の情報を表示させます。呼び出し経路(「Call Stack」)の情報から、呼び出し元の呼び出し位置が以下であることがわかります。

```
“LISTDBG!LISTDBG+0x3a8”
```

4. 翻訳オプションLISTを指定して出力した翻訳リストを用意し、“目的プログラムリスト”を参照します。
5. “目的プログラムリスト”から、LISTDBGのオフセットを調べます。

```
                        LISTDBG:
000000000050  48894C2408              mov       qword ptr [rsp+0x08],rcx
```

6. LISTDBGのオフセットと呼び出し位置のオフセット“0x3a8”を加算します。

```
0x3a8 + 0x50 = 0x3f8
```

7. 呼び出し箇所を特定します。

計算で求めた“0x3f8”の位置を目的プログラムリスト(LISTDBG.LST)から調べます。

```
--- 13 ---            CALL
0000000003DE  488D83E0010000    lea     rax,[rbx+0x000001E0]            計算
0000000003E5  48890424          mov     qword ptr [rsp],rax             PARM-1B8
0000000003E9  4C8BBBB0010000    mov     r15,qword ptr [rbx+0x000001B0]  BGW.0
0000000003F0  488B0C24          mov     rcx,qword ptr [rsp]
0000000003F4  41FF5758          call    qword ptr [r15+0x58]            DIVPROC
0000000003F8  48898424C0000000  mov     qword ptr [rsp+0x000000C0],rax  TRLP+0
000000000400  0FB78424C0000000  movzx   eax,word ptr [rsp+0x000000C0]   TRLP+0
000000000408  668983E8010000    mov     word ptr [rbx+0x000001E8],ax    答え
00000000040F  4C8BB3A0010000    mov     r14,qword ptr [rbx+0x000001A0]  BC0.0
000000000416  410FB64621        movzx   eax,byte ptr [r14+0x21]         X"00"
00000000041B  0FB68B70010000    movzx   ecx,byte ptr [rbx+0x00000170]   LCBC+0
000000000422  39C1              cmp     ecx,eax
```

“0x3f8”の位置は戻り位置のため、呼び出した命令は、その前の“0x3F4”のcall命令です。

8. ソースプログラム上の位置を特定します。

目的プログラム上の呼び出した命令位置から上方向に、行の情報を検索します。この場合、“--- 13 --- CALL”が見つかります。ソースプログラムリスト上でこのCALL文を調べると以下のようになっています。

```
[ソースプログラムリスト(LISTDBG.LST)]
        13   000130     CALL "DIVPROC" USING 計算 RETURNING 答え.
```

このCALL文では、DIVPROCの呼び出し時に、引数として“計算”に含まれる“被除数”と“除数”を渡しています。呼び出し元プログラム“LISTDBG”で“除数”に値を設定していないため、ゼロ除算が発生したことがわかります。

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. NetCOBOL Studioの〔翻訳オプション〕ページによる指定

   参照

   “NetCOBOL Studio ユーザーズガイド”

2. -WCオプションによる指定
3. -WCオプション以外のコマンドオプションによる指定

   参照

   “J.1 COBOLコマンド”

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、-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。

-WCオプション

-WCオプションの指定には、以下の方法があります。

1. 環境変数COB_OPTIONSによる指定

   参照

   “1.2.1 環境変数の設定”

2. 翻訳コマンドによる指定

   参照

   “J.1.19 -WC（翻訳オプションの指定）”

指定の強さは、2.＞1.です。

-WCオプションは複数指定できます。指定されたすべての翻訳オプションが有効になります。

ただし、同じ種類の翻訳オプションが複数指定された場合、指定の強さが強く、最後に指定された翻訳オプションが有効になります。

例

- 翻訳オプションMESSAGEと翻訳オプションNOOPTIMIZEが有効になります。

```
set COB_OPTIONS=-WC,"MESSAGE,OPTIMIZE"
cobol -WC,"NOOPTIMIZE" a.cob
```

- 翻訳オプションMESSAGEと翻訳オプションCHECK(NUMERIC)が有効になります。

```
cobol -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。

STUDIO

NetCOBOL Studioの〔翻訳オプション〕ページで指定できます。

-WC

コマンドオプション-WCで指定できます。

@

翻訳指示文で指定できます。

上記以外に、“(@)”と表示されている場合は、ソースファイルに複数の翻訳単位が記述されていない場合だけ、翻訳指示文での指定が可能になります。

フォルダ名およびファイル名の指定方法

翻訳オプションに指定するフォルダ名およびファイル名は、絶対パス名または相対パス名で指定します。以下の文字を含む場合、フォルダ名またはファイル名を二重引用符(")で囲む必要があります。

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

注意

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

COBOL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、カレントフォルダはプロジェクトフォルダです。

注意

実行時のカレントフォルダは、アプリケーションの実行環境や動作内容に依存します。

A.3 翻訳オプション一覧

翻訳時の資源に関するもの

- “A.3.17 FORMLIB(画面帳票定義体ファイルのフォルダの指定)”
- “A.3.20 LIB(登録集ファイルのフォルダの指定)”

- “A.3.37 REP（リポジトリファイルの入出力先フォルダの指定）”
- “A.3.38 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.3.7 COPY（登録集原文の表示）”
- “A.3.21 LINECOUNT（翻訳リストの1ページあたりの行数）”
- “A.3.22 LINESIZE（翻訳リストの1行あたりの文字数）”
- “A.3.23 LIST（目的プログラムリストの出力の可否）”
- “A.3.25 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.3.26 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.3.31 NUMBER（ソースプログラムの一連番号領域の指定）”
- “A.3.34 PRINT（各種翻訳リストの出力の可否および出力先の指定）”
- “A.3.45 SOURCE（ソースプログラムリストの出力の可否）”
- “A.3.56 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.3.15 FLAG（診断メッセージのレベル）”
- “A.3.16 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.3.1 ALPHAL（英小文字の扱い）”
- “A.3.4 BINARY（2進項目の扱い）”
- “A.3.10 CURRENCY（通貨編集用文字の扱い）”
- “A.3.12 DUPCHAR（重複文字の扱い）”
- “A.3.19 LANGLVL（ANSI COBOL規格の指定）”
- “A.3.18 INITVALUE（作業場所節でのVALUE句なし項目の扱い）”
- “A.3.29 NCW（日本語利用者語の文字集合の指定）”
- “A.3.30 NSPCOMP（日本語空白の比較方法の指定）”
- “A.3.35 QUOTE/APOST（表意定数QUOTEの扱い）”
- “A.3.39 RSV（予約語の種類）”
- “A.3.41 SCS（ソースファイルのコード系）”
- “A.3.42 SDS（符号付き10進項目の符号の整形の可否）”
- “A.3.43 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.3.46 SQLGRP（SQLのホスト変数定義の拡張）”
- “A.3.47 SRF（正書法の種類）”
- “A.3.50 STD1（英数字の文字の大小順序の指定）”
- “A.3.51 TAB（タブの扱い）”
- “A.3.57 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.3.40 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.3.2 ARITHMETIC（演算モードの指定）”
- “A.3.3 ASCOMP5（2進項目の解釈の指定）”
- “A.3.9 CREATE（創成ファイルの指定）”
- “A.3.11 DLOAD（プログラム構造の指定）”
- “A.3.13 ENCODE(データ項目のエンコードの指定)”
- “A.3.24 MAIN（主プログラム/副プログラムの指定）”
- “A.3.27 MODE（ACCEPT文の動作の指定）”
- “A.3.28 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.3.32 OBJECT（目的プログラムの出力の可否）”
- “A.3.33 OPTIMIZE（広域最適化の扱い）”
- “A.3.36 RCS（実行時コード系の指定）”
- “A.3.53 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.3.14 EQUALS（SORT文での同一キーデータの処理方法）”
- “A.3.55 TRUNC（桁落とし処理の可否）”

実行時の資源に関するもの

- “A.3.44 SMSIZE（PowerSORTが使用するメモリ容量を指定）”
- “A.3.48 SSIN（ACCEPT文のデータの入力先）”
- “A.3.49 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.3.8 COUNT（COUNT機能の使用の可否）”
- “A.3.52 TEST（デバッグ機能および診断機能の使用の可否）”
- “A.3.54 TRACE（TRACE機能の使用の可否）”

A.3.1 ALPHAL（英小文字の扱い）

STUDIO,-WC,@

```
{ ALPHAL[( { ALL  } )] }
{          { WORD }    }
{ NOALPHAL             }
```

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.3.2 ARITHMETIC(演算モードの指定)

STUDIO,-WC,@

ARITHMETIC({ 18 | 31 [, INF] })

演算モードとして、18桁互換演算モードを使用する(ARITHMETIC(18))か、31桁拡張演算モードを使用する(ARITHMETIC(31))か、を指定します。

ARITHMETIC(18)	18桁互換演算モードを使用します。
ARITHMETIC(31)	31桁拡張演算モードを使用します。
ARITHMETIC(31,INF)	31桁拡張演算モードを使用します。また、以下の箇所に対して、Iレベルの診断メッセージを出力します。 ・ 18桁互換演算モードの時、翻訳メッセージ“JMN3024I-W 中間結果のけた数が30けたを超えています. 中間結果のけた数を30けたにして処理を続けます. ”が出力される箇所 ・ 中間結果の属性(固定小数点であるか、浮動小数点であるか)が、18桁互換演算モードとは異なる箇所

注意

- ARITHMETIC(31)を指定する場合、翻訳オプションBINARYは、BINARY(WORD, MLBON)のみ指定できます。
- NetCOBOL V10.5以前や他システムとの互換性を重視する場合、ARITHMETIC(18)を指定してください。
- 演算モードの詳細は、“COBOL文法書”の“1.7 演算モード”を参照してください。

A.3.3 ASCOMP5(2進項目の解釈の指定)

STUDIO,-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,-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

- 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。
- 31桁拡張演算モード(ARITHMETIC(31))を使用する場合、BINARY(WORD,MLBON)のみ指定できます。

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
19 ～ 28 (注)	19 ～ 28	-	12
29 ～ 31 (注)	29 ～ 31	-	16

注 ： 31桁演算モードの場合

A.3.5 CHECK（CHECK機能の使用の可否）

STUDIO,-WC,@

{ CHECK[([n] [,ALL] [,BOUND] [,ICONF] [,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、NUMERIC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(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)	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。 ・ USING指定のパラメタの個数の一致 ・ RETURNING指定のパラメタの有無の一致 ・ 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） ・ 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 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。

	・ 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) ・ RETURING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
  - 次の組込み関数の引数
    - FUNCTION ADDR
    - FUNCTION LENG
    - FUNCTION LENGTH
  - 英数字項目または集団項目から、外部10進項目または内部10進項目のオブジェクトプロパティへの転記

参考

“19.2 CHECK機能”

A.3.6 CONF(規格の違いによるメッセージの出力の可否)

STUDIO,-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.3.19 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 COPY(登録集原文の表示)

STUDIO,-WC,@

{ COPY / NOCOPY }

ソースプログラムリスト内に、COPY文によって組み込まれる登録集原文を表示する(COPY)か、しない(NOCOPY)か、を指定します。

参照

“A.3.45 SOURCE(ソースプログラムリストの出力の可否)”

注意

COPYは、翻訳オプションSOURCEを指定した場合だけ意味を持ちます。

A.3.8 COUNT(COUNT機能の使用の可否)

STUDIO,-WC,@

{ COUNT / NOCOUNT }

COUNT機能を使用する(COUNT)か、使用しない(NOCOUNT)か、を指定します。

注意

- COUNTを指定すると、COUNT情報を出力するための処理が目的プログラム中に組み込まれるため、実行性能が低下します。デバッグ終了時には、NOCOUNTを指定して再翻訳してください。
- COUNTは、翻訳オプションTRACEまたは-Drオプションと同時に指定できません。同時に指定された場合、後に指定されたオプションが有効となります。

参考

“19.4 COUNT機能”

A.3.9 CREATE(創成ファイルの指定)

-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.3.10 CURRENCY(通貨編集用文字の扱い)

STUDIO,-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.3.11 DLOAD(プログラム構造の指定)

STUDIO,-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"

"10.1.3 動的プログラム構造"

"16.3.2 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.3.12 DUPCHAR(重複文字の扱い)

STUDIO,-WC,@

```
DUPCHAR ( { STD } )
          { EXT }
```

ソースファイルがUnicodeの場合、コンパイラが付加／置換する以下のJIS非漢字の負号を、拡張文字(DUPCHAR(EXT))とするか、標準文字(DUPCHAR(STD))とするか、を指定します。

- 3バイト項目制御部を指定した帳票定義体
- COPY文の書き方2と3の日本語利用者語

表A.1 JIS非漢字の負号の扱い

オプションの指定	文字コード(UTF-8)
DUPCHAR(EXT)	－(X"EFBC8D")
DUPCHAR(STD)	-(X"E28892")

参考

"L.4.2 JIS非漢字の負号について"

A.3.13 ENCODE(データ項目のエンコードの指定)

STUDIO,-WC,@

```
ENCODE ( { SJIS } [, { SJIS { UTF16 } [, { LE } ] } ])
         { UTF8 }      { UTF32 }       { BE }
```

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(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.3.14 EQUALS(SORT文での同一キーデータの処理方法)

STUDIO,-WC,@

{ EQUALS / NOEQUALS }

実行時に、SORT文の入力中に同一キーを持つレコードが複数個存在する場合、それらに関して、SORT文の出力レコードの順序をSORT文の入力レコードの順序と同じにすることを保証する(EQUALS)か、しない(NOEQUALS)か、を指定します。

NOEQUALSを指定すると、SORT文で同一キーを持つレコードの出力順序は規定されません。

注意

EQUALSを指定すると、整列操作で入力順序を保証するための特別な処理が行われるために実行性能が低下します。

A.3.15 FLAG(診断メッセージのレベル)

STUDIO,-WC,@

FLAG({ I / W / E })

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

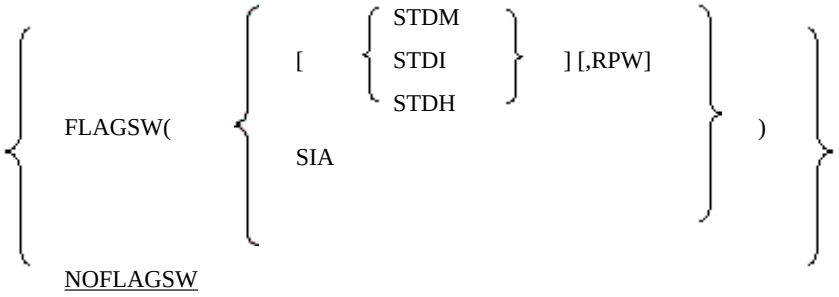
翻訳オプションCONFによる指摘メッセージは、FLAGの指定に関係なく表示されます。

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.3.16 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,-WC,@



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

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.3.17 FORMLIB（画面帳票定義体ファイルのフォルダの指定）

STUDIO

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.1.14 -m（画面帳票定義体ファイルのフォルダの指定）”

A.3.18 INITVALUE（作業場所節でのVALUE句なし項目の扱い）

STUDIO,-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.3.19 LANGLVL（ANSI COBOL規格の指定）

STUDIO,-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

オプション	規格
LANGLVL(68)	'68 ANS COBOL

A.3.20 LIB（登録集ファイルのフォルダの指定）

STUDIO

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.1.12 -I（登録集ファイルのフォルダの指定）”

A.3.21 LINECOUNT（翻訳リストの1ページあたりの行数）

STUDIO,-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.3.22 LINESIZE（翻訳リストの1行あたりの文字数）

STUDIO,-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.3.23 LIST(目的プログラムリストの出力の可否)

STUDIO,-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.1.7 -dp(翻訳リストファイルのフォルダの指定)"、"J.1.16 -P(各種翻訳リストの出力および出力先の指定)"および"A.3.34 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.3.24 MAIN(主プログラム/副プログラムの指定)

-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。
- 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

"5.4 プログラムの実行"

A.3.25 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,-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.1.7 -dp(翻訳リストファイルのフォルダの指定)"、"J.1.16 -P(各種翻訳リストの出力および出力先の指定)"および"A.3.34 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.3.26 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,-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.3.27 MODE(ACCEPT文の動作の指定)

STUDIO,-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.3.28 NAME(翻訳単位ごとのオブジェクトファイルの出力の可否)

-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。

参考

- 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.3.29 NCW(日本語利用者語の文字集合の指定)

STUDIO,-WC,@

NCW ({ STD / SYS })

利用者語に指定できる日本語文字集合をシステム共通な日本語文字集合とする(NCW(STD))か、計算機の日本語文字集合とする(NCW(SYS))か、を指定します。

STDを指定すると、次の日本語文字集合が日本語利用者語として利用できます。

・ JIS第一水準

・ JIS第二水準

・ JIS非漢字(以下の文字)

－ ０、１、…、９

－ Ａ、Ｂ、…、Ｚ

－ ａ、ｂ、…、ｚ

－ ぁ、あ、ぃ、い、…、ん

－ ァ、ア、ィ、イ、…、ン、ヴ、ヵ、ヶ

－ **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.3.30 NSPCOMP（日本語空白の比較方法の指定）

STUDIO,-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。

参考

OSIV系システムのコード系では、日本語空白がANK空白の2文字分と同じ値を持っていますが、Windowsのコード系では、同じ値を持たないため、OSIV系システムで使用していたCOBOLプログラムを本システムで動作させる場合には、通常、ソースプログラムの修正が必要となります。

翻訳オプションNSPCOMP(ASP)を指定することにより、比較対象の空白が等価に処理されるため、前述の条件に合う比較は、ソースプログラムを修正しなくてもOSIV系システムと同じ動作が期待できます。

ただし、空白を等価に扱う処理は、格納されている文字データから判断するため、作用対象が集団項目のとき、従属する項目属性に合わせた処理はしません。このため、例えば、従属する英数字項目中に文字以外のデータが設定されていた場合などは誤動作する可能性がありますので、注意が必要です。

参照

“L.4.1 日本語空白と英数字空白の文字コード”

A.3.31 NUMBER(ソースプログラムの一連番号領域の指定)

STUDIO,-WC,@

{ NUMBER / NONUMBER }

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(NONUMBER)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。
NONUMBER	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、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。NONUMBERを有効にして翻訳してください。

参照

“A.3.47 SRF(正書法の種類)”

A.3.32 OBJECT(目的プログラムの出力の可否)

STUDIO,-WC,@

{ OBJECT [(NetCOBOL Studioのみフォルダ指定可)] | NOOBJECT }

目的プログラムを出力するか(OBJECT)、しないか(NOOBJECT)を指定します。

目的プログラムは、ソースプログラムと同じフォルダに格納されます。

NetCOBOL Studio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.3.33 OPTIMIZE(広域最適化の扱い)

STUDIO,-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OPTIMIZEを指定した場合、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、NetCOBOL Studioのデバッグ機能では使用できません。デバッグ機能を使用する場合は、NOOPTIMIZEを選択してください。
- OPTIMIZEオプションが有効な状態で翻訳を行った場合、CPUの特性上デバッグが難しいため、デフォルトのNOOPTIMIZEの状態で使用することを推奨します。実行性能などの問題により、OPTIMIZEオプションを有効にする必要がある場合は、このシステム上でプログラムの正当性を十分検討してから使用してください。

参考

“付録F 広域最適化”

“A.3.52 TEST(デバッグ機能および診断機能の使用の可否)”

A.3.34 PRINT（各種翻訳リストの出力の可否および出力先の指定）

STUDIO

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.3.35 QUOTE/APOST（表意定数QUOTEの扱い）

STUDIO,-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.3.36 RCS（実行時コード系の指定）

STUDIO,-WC,@

RCS ({ SJIS / { UTF16 / UCS2 } [, { LE / BE }] })

実行時のコード系をシフトJISにする(RCS(SJIS))か、Unicodeにする(RCS(UTF16)またはRCS(UCS2))か、を指定します。

RCS(UTF16,LE) または RCS(UCS2,LE)

Unicode環境でのエンディアンをリトルエンディアンとします。

RCS(UTF16,BE) または RCS(UCS2,BE)

Unicode環境でのエンディアンをビッグエンディアンとします。

RCS(UTF16)とRCS(UCS2)は同義です。

注意

- 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(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。

参考

“第6章 文字コード”

A.3.37 REP(リポジトリファイルの入出力先フォルダの指定)

STUDIO

REP(フォルダ)

リポジトリファイルは、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、リポジトリファイルの入力先フォルダとしても使用されます。

参考

オプションの指定が重複した場合、入力先フォルダは以下の順に検索されます。

1. -Rオプション
2. 翻訳オプションREPIN
3. 環境変数COB_REPIN
4. -drオプション
5. 翻訳オプションREP
6. カレントフォルダ

参照

“1.2.1 環境変数の設定”

“J.1.9 -dr(リポジトリファイルの入出力先フォルダの指定)”

“J.1.17 -R(リポジトリファイルの入力先フォルダの指定)”

“A.3.38 REPIN(リポジトリファイルの入力先フォルダの指定)”

A.3.38 REPIN(リポジトリファイルの入力先フォルダの指定)

STUDIO

REPIN(フォルダ[; フォルダ] ...)

リポジトリファイルの入力先フォルダを指定します。リポジ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。

参考

オプションの指定が重複した場合、以下の順に検索されます。

1. -Rオプション
2. 翻訳オプションREPIN
3. 環境変数COB_REPIN
4. -drオプション
5. 翻訳オプションREP
6. カレントフォルダ

参照

“J.1.9 -dr(リポジトリファイルの入出力先フォルダの指定)”

“J.1.17 -R(リポジトリファイルの入力先フォルダの指定)”

“A.3.37 REP(リポジトリファイルの入出力先フォルダの指定)”

A.3.39 RSV(予約語の種類)

STUDIO,-WC,@

RSV (ALL | V111 | V112 | V122 | V125 | V30 | V40 | V61 | V70 | V81 | V90 | V1050 | VSR2 | VSR3)

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(VSR2)	VS COBOLII REL2.0用
RSV(VSR3)	VS COBOLII REL3.0用

A.3.40 SAI(ソース解析情報ファイルの出力の可否)

STUDIO,-WC,@

{ SAI [(NetCOBOL Studio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 SCS(ソースファイルのコード系)

-WC

SCS ({ 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、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。

参照

“A.3.13 ENCODE(データ項目のエンコードの指定)”

A.3.42 SDS(符号付き10進項目の符号の整形の可否)

STUDIO,-WC,@

{ SDS / NOSDS }

符号付き外部10進項目および符号付き内部10進項目への転記で、送出し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。

NOSDSを指定した場合、符号が負を意味する値なら負号に、符号が正を意味する値なら正号に変更します。値0に負号がつく場合、正号に変更します。

例

オプションによって動作が変わるプログラム

```
01  A PIC S9V9  VALUE -0.1
01  B PIC S9.
    :
MOVE A TO B.
```

SDSを指定した場合：-0.1の小数点以下が切り捨てられ、-0がBに格納されます。

NOSDSを指定した場合：-0.1の小数点以下が切り捨てられ-0となり、さらに符号の整形が行われ、+0がBに格納されます。

A.3.43 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,-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.3.44 SMSIZE(PowerSORTが使用するメモリ容量を指定)

STUDIO,-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

ー 特殊レジスタ

```
    MOVE 102400 TO SORT-CORE-SIZE
```

(102400=100キロです)

ー 翻訳オプション

```
  SMSIZE(500K)
```

ー 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.3.45 SOURCE(ソースプログラムリストの出力の可否)

STUDIO,-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.1.7 -dp(翻訳リストファイルのフォルダの指定)”、“J.1.16 -P(各種翻訳リストの出力および出力先の指定)”および“A.3.34 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.3.46 SQLGRP(SQLのホスト変数定義の拡張)

STUDIO,-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.3.47 SRF(正書法の種類)

STUDIO,-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。
- NetCOBOL Studioでは、翻訳オプションSRFにFREEは指定できません。

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.3.48 SSIN(ACCEPT文のデータの入力先)

STUDIO,-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.3.49 SSOUT(DISPLAY文のデータの出力先)

STUDIO,-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.3.50 STD1(英数字の文字の大小順序の指定)

STUDIO,-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.3.51 TAB(タブの扱い)

STUDIO,-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.3.52 TEST(デバッグ機能および診断機能の使用の可否)

STUDIO,-WC,@

{ TEST[(NetCOBOL Studioのみフォルダ指定可)] / NOTEST }

NetCOBOL Studioのデバッグ機能および診断機能の使用を可能にする(TEST)か、しない(NOTEST)か、を指定します。

TESTを指定すると、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。デバッグ機能および診断機能はこのデバッグ情報ファイルを使用します。

NetCOBOL Studio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。

リンク時には、デバッグ機能および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

TESTとOPTIMIZEを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、診断機能で使用することはできますが、NetCOBOL Studioのデバッグ機能で使用することはできません。

デバッグ機能を使用する場合は、OPTIMIZEを指定しないでください。

参照

"A.3.33 OPTIMIZE(広域最適化の扱い)"

参考

"3.2 翻訳に必要な資源"

"20.1 診断機能"

"NetCOBOL Studio ユーザーズガイド"

A.3.53 THREAD(マルチスレッドプログラム作成の指定)

STUDIO,-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。

参考

"第18章 マルチスレッド"

A.3.54 TRACE（TRACE機能の使用の可否）

STUDIO,-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.3.8 COUNT（COUNT機能の使用の可否）”

参考

“19.3 TRACE機能”

A.3.55 TRUNC（桁落とし処理の可否）

STUDIO,-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.3.33 OPTIMIZE(広域最適化の扱い)”

注意

- NOTRUNCで、送出し側項目の整数部の桁数が、受取り側項目の整数部の桁数より大きい場合の結果は規定されません。
- NOTRUNCを指定する場合には、桁落としが行われなくても、受取り側項目にPICTURE句に記述した桁を超える値が格納されないように、プログラムを設計しなければなりません。
- NOTRUNCで桁落としを行うか行わないかの基準は、コンパイラによって異なります。したがって、NOTRUNCによって桁落としが行われないことを利用したプログラムは他システムへの互換が保証されないので注意してください。

A.3.56 XREF(相互参照リストの出力の可否)

STUDIO,-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.1.7 -dp(翻訳リストファイルのフォルダの指定)”、“J.1.16 -P(各種翻訳リストの出力および出力先の指定)”およびA.3.34 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.3.57 ZWB(符号付き外部10進項目と英数字項目の比較)

STUDIO,-WC,@

{ ZWB / NOZWB }

符号付き外部10進項目を英数字フィールドと比較するときに、外部10進項目の符号部を無視して比較する(ZWB)か、符号部を含めて比較する(NOZWB)か、を指定します。ここで、英数字とは、英数字項目、英字項目、英数字編集項目、数字編集項目、文字定数およびZERO以外の表意定数のことです。

例

```
   77  ED    PIC  S9(3)  VALUE  +123.
   77  AN    PIC   X(3)  VALUE  "123".
```

この場合、条件式 ED = AN の真偽は、以下のようになります。

- ZWB を指定した場合 ： 真
- NOZWB を指定した場合 ： 偽

A.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。

- CHECK
- CONF
- COPY
- COUNT
- DLOAD
- FLAG
- FORMLIB
- LIB
- LINECOUNT
- LINESIZE
- LIST
- MAP
- MESSAGE
- NUMBER
- OBJECT

- PRINT
- QUOTE/APOST
- REP
- REPIN
- SAI
- SOURCE
- SRF
- TEST
- TRACE
- XREF

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.1.16 -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		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とデータの属性情報	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やデータ領域の割り付け位置や定数領域の情報	
	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.1.16 -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。

```
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“第2章 翻訳時メッセージ”を参照してください。

[3] 本製品のバージョンを表示します。

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

```
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

```
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

```
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

以下に、ソースプログラムリストの出力形式を示します。

```
    行番号  一連番号 A   B
      [1]                    [2]
        1     000100 IDENTIFICATION DIVISION.
        2     000200  PROGRAM-ID.  A.
        3     000300*
        4     000400 DATA DIVISION.
        5     000500 WORKING-STORAGE SECTION.
        6     000600     COPY  A1.
      1-1 C   000600 77  答え      PIC 9(2).
      1-2 C   000700 77  除数      PIC 9(2).
      1-3 C   000800 77  被除数    PIC 9(2).
        7     000900*
        8     001000 PROCEDURE DIVISION.
        9     001100*
```

```
      10      001200      MOVE 10 TO 被除数.
      11      001300      MOVE  0 TO 除数.
      12      001400*
      13      001500      COMPUTE 答え = 被除数 / 除数.
      14      001600*
      15      001700        EXIT PROGRAM.
      16      001800    END PROGRAM A.
```

[1] 行番号

行番号を次の形式で表示します

1. 翻訳オプションNUMBER有効時

```
[COPY修飾値-]利用者行番号
```

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

2. 翻訳オプションNONUMBER有効時

```
[COPY修飾値-]ソースファイル内相対番号
```

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

[2] ソースプログラム

ソースプログラムそのものを表示します。

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。

翻訳オプションNOOPTIMIZE有効時(デフォルト)の目的プログラムリスト

```
  [1]          [2]                    [3]
  番地         機械語       手続き名  アセンブラ形式命令
                              GLB.9  [4]
--- 10 --- [5]          MOVE [6]
0000000003E0  66C783E40100003130    mov       word ptr [rbx+0x000001E4],0x3031  被除数
--- 11 ---              MOVE
0000000003E9  66C783E20100003030    mov       word ptr [rbx+0x000001E2],0x3030 除数
--- 13 ---              COMPUTE
0000000003F2  0FB693E5010000        movzx     edx,byte ptr [rbx+0x000001E5] 被除数+1
0000000003F9  440FB683E4010000      movzx     r8d,byte ptr [rbx+0x000001E4] 被除数
000000000401  4183E00F              and       r8d,0x0F
000000000405  4F8D0480              lea       r8,[r8+r8*4]
000000000409  83E20F                and       edx,0x0F
00000000040C  4E8D3C42              lea       r15,[rdx+r8*2]
000000000410  0FB693E3010000        movzx     edx,byte ptr [rbx+0x000001E3]  除数+1
000000000417  440FB683E2010000      movzx     r8d,byte ptr [rbx+0x000001E2]  除数
00000000041F  4183E00F              and       r8d,0x0F
```

```
000000000423  4F8D0480              lea       r8,[r8+r8*4]
000000000427  83E20F                and       edx,0x0F
00000000042A  4E8D3442              lea       r14,[rdx+r8*2]
00000000042E  4C89F8                mov       rax,r15
000000000431  4899                  cqo
000000000433  49F7FE                idiv      r14
000000000436  4989C5                mov       r13,rax
000000000439  49C7C764000000        mov       r15,0x00000064
000000000440  4C89E8                mov       rax,r13
000000000443  4C89EA                mov       rdx,r13
000000000446  4C89E9                mov       rcx,r13
000000000449  48F7D9                neg       rcx
00000000044C  4983FD00              cmp       r13,0x00
000000000450  480F4CC1              cmovnge   rax,rcx
000000000454  4C39F8                cmp       rax,r15
000000000457  7C08                  jl        0x000000000461                GLB.10
000000000459  4C89E8                mov       rax,r13
00000000045C  4899                  cqo
00000000045E  49F7FF                idiv      r15
                          GLB.10
000000000461  4989D4                mov       r12,rdx
000000000464  664489A42496000000    mov       word ptr [rsp+0x00000096],r12w TRLP+0
00000000046D  480FBF842496000000    movsx     rax,word ptr [rsp+0x00000096]  TRLP+0
000000000476  4989C2                mov       r10,rax
000000000479  49C1FA3F              sar       r10,0x3F
00000000047D  4983FA00              cmp       r10,0x00
000000000481  790A                  jns       0x00000000048D                GLB.11
000000000483  49F7DA                neg       r10
000000000486  48F7D8                neg       rax
000000000489  4983DA00              sbb       r10,0x00
                          GLB.11
00000000048D  488DBBE0010000        lea       rdi,[rbx+0x000001E0]          答え
000000000494  48C7C602000000        mov       rsi,0x00000002
00000000049B  4C89D2                mov       rdx,r10
00000000049E  4889C1                mov       rcx,rax
0000000004A1  4C8B9BB0010000        mov       r11,qword ptr [rbx+0x000001B0]
0000000004A8  41FF5378              call      qword ptr [r11+0x78]
```

翻訳オプションOPTIMIZE有効時の目的プログラムリスト

```
  [1]         [2]                   [3]
  番地        機械語      手続き名  アセンブラ形式命令
                          GLB.9 [4]
--- 10 --- [5]        MOVE [6]
<SIMPLE STORE>
--- 11 ---            MOVE
<SIMPLE STORE>
--- 13 ---            COMPUTE
<CONSTANT FOLDING>
<CONSTANT FOLDING>
<REDUNDANT STORE>
<REDUNDANT STORE>
<REDUNDANT STORE>
--- 15 ---            EXIT PROGRAM
0000000003E0  49C7C700000000        mov       r15,0x00000000
0000000003E7  4489BC2408010000      mov       dword ptr [rsp+0x00000108],r15d   LCB+10
0000000003EF  49C7C70F000000        mov       r15,0x0000000F
0000000003F6  4489BC240C010000      mov       dword ptr [rsp+0x0000010C],r15d   LCB+10
0000000003FE  E929FFFFFF            jmp       0x00000000032C                    GLB.8
                                        BBK=00005(000)
                                        NEVER EXECUTED
                                        ............PX
```

[1] 番地

機械語のオブジェクト先頭からの相対オフセットを表しています。

[2] 機械語命令コード

機械語(オブジェクトコード)を表しています。

[3] アセンブラ形式の命令

機械語をx64アーキテクチャのアセンブリ言語に準じた形式で表しています。

[4] 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。

[5] 行番号

COBOLプログラムの行番号を表しています。

[6] 動詞名

COBOLプログラムの中に記述された動詞名を表しています。

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

```
   [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   heap+00000240                FD OUTFILE            LSAM       BHG.000000
 16 [heap+000001F8]+00000000     0 01 印刷レコード     60 ALPHANUM   BVA.000003    SJIS
 18   heap+000003C0               0 77 答え              8 EXT-DEC    BHG.000000
 19   heap+000003C8               0 77 除数              4 EXT-DEC    BHG.000000
 21   heap+00000040               0 01 CSTART            8 ALPHANUM   BCO.000000    SJIS
 22   heap-00000048               0 01 CEND              8 ALPHANUM   BCO.000000    SJIS
 23   heap-00000050               0 01 CRES              8 ALPHANUM   BCO.000000    SJIS
 25 [heap+000001E8]+00000000     0 01 被除数            2 EXT-DEC    BVA.000001
[11]**SUB1**
 48 [heap+000001F0]+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。

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
```

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。

[2] 番地、オフセット

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

```
セクション名+相対アドレス
```

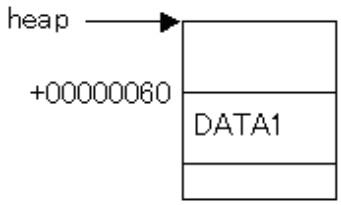
ー セクション名
以下のいずれかを表示します。

- rdat (.rodata)
- data (.data)
- stac (スタック)
- heap (ヒープ)
- hea2 (ヒープ)

ー 相対アドレス
セクションの先頭位置からの相対アドレスを表示します。

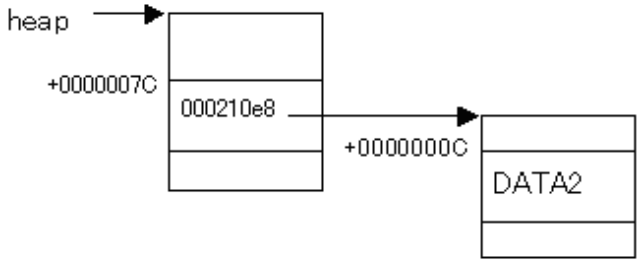
オフセットは、“[]”付きで表示された番地に対して表示します。
データ領域の参照方法は、オフセットが表示される場合と表示されない場合とで異なります。

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



heapの先頭アドレスに0x60を足した位置に、DATA1のデータ領域があることを示します。

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



heapの先頭アドレスに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

```
    番地              フィールド名                  長さ(10)
[1]
    ** GWA **
      * GCB *
data+00000000         GCB FIXED AREA                   8

      * GMB *
data+00000008         GMB POINTERS AREA              376
data+00000008           VNAL                         136
     ........           VNAS                           0
data+00000090           VNAO                         240
     ........           FARA                           0
     ........           METHOD ADDRESS AREA            0
     ........           BEA                            0
     ........           BEAD / BEAR                    0
     ........           FMBE                           0
     ........         GMB CONTROL BLOCKS 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
    ........            INTARFACE TABLE               0
    ........            METHOD COPY TABLE             0
    ........          INVOKE PARM INFO                0
    ........          AS MODIFY INFO                  0

   ** COA **
     * CCB *
rdat+00000000         CCB FIXED AREA                  4
    ........          STANDARD CONSTANT AREA          0

     * CMB *
    ........          CMB POINTERS AREA               0
    ........            BEAI                          0
    ........            BVAI                          0
    ........            IPA                           0
rdat+00000010         LITERAL AREA                   40
rdat+00000040         CONSTANT SECTION DATA          24
rdat+00000058         CMB CONTROL BLOCKS AREA       816
rdat+00000058           FMB2                        224
    ........            FMBI                          0
    ........            SMB1                          0
    ........            SMB2                          0
    ........            EDT                           0
    ........            EFT                           0
rdat+00000138           FMB2 ADD-PRM LIST            16
    ........            ERROR PROCEDURE               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 AREA                      0
    ........            SPECIAL REGISTERS             0
rdat+00000148           EPA CONTROL AREA            576
    ........            SCREEN CONTROL AREA           0
    ........            EXCEPTION PROC LIST           0
    ........            SQL INIT INFO                 0
    ........            OLE PARM INFO                 0
    ........            CALL PARM INFO                0
    ........            CALL PARM ADD INFO            0
    ........            UWA RECORD INFO               0
    ........            UWA INFO                      0
    ........            UWA RECORD LIST               0
    ........            RECORD INFO                   0
    ........          CALL PARM ADDRESS 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
    ........          INIT TABLE                      0

   ** HGWA **
     * HGCB *
```

```
heap+00000000          HGCB FIXED AREA                  72

      * HGMB *
heap+00000048          LIA FIXED AREA                  256
     ........          IWA1 AREA                         0
     ........            LINAGE COUNTER                  0
     ........            OTHER AREA                      0
     ........            ALTINX                          0
     ........            PCT                             0
heap+00000148          LCB AREA                        160
heap+000001E8          HGMB POINTERS AREA               88
     ........            VPA                             0
     ........            PSA                             0
heap+000001E8            BVA                           24
     ........            BEA                             0
     ........            FMBE                            0
heap+00000200            CONTROL ADDRESS TABLE           64
     ........            MUTEX HANDLE AREA               0
heap+00000240          HGMB CONTROL BLOCK AREA         384
heap+00000240            FMB1                          384
     ........            SMBO                            0
     ........            SPECIAL REGISTERS               0
     ........            S-A LIST                        0
     ........            DPA                             0
     ........            SQL CONTROL AREA                0
     ........            DMA                             0
     ........            SCREEN CONTROL AREA             0
     ........            METHOD INIT INFO                0
     ........          CFOR                              0

      * HGWS *
heap+000003C0          DATA AREA                        12

    ** STK **
      * SCB *
stac+00000000          ABIA                             72
stac+00000050          SCB FIXED AREA                  112
stac+000000C0          TL 1ST AREA                      32
stac+000000E0          LCB AREA                        144
stac+00000170          SGM POINTERS AREA                 8
     ........            VPA                             0
     ........            PSA                             0
     ........            BVA                             0
stac+00000170            BHG                             8
     ........            BOD                             0
stac+00000178          LIA VARIABLE AREA               160
     ........          SOR AREA                          0
     ........          TSG AREA                          0
     ........          TRG AREA                          0
     ........          SGM CONTROL BLOCKS AREA           0
     ........            FMB1                            0
     ........            SMBO                            0
     ........            SPECIAL REGISTER                0
     ........            MIA                             0
     ........            SQL CONTROL AREA                0
stac+00000218          IWA3 AREA                        56
     ........            ALTINX                          0
     ........            USESAVE                         0
stac+00000218            ENTSAVE                         8
     ........            PCT                             0
     ........            CONTENT                         0
     ........            USEOSAVE                        0
stac+00000220            RTNADDR                        16
```

```
     ........            RTNAREA                          0
     ........            LINAGE COUNTER                   0
     ........            OTHER AREA                       0
stac+00000230            PARM                            32
     ........            CALL PARM INFO                   0
     ........          DATA AREA                          0
stac+00000250          TL 2ND AREA                       32
     ........          PRESERVED REGISTER                 0
     ........          SCRATCH / REGISTER PARM            0
     ........          RTS RETURN AREA                    0
stac+00000278          LIA FIXED ADDR                     8
stac+00000280          CBL STRING                         8
stac+00000288          RESERVED REGISTER                 64
stac+000002C8          RETURN ADDRESS                     8
     ........          PARM AREA                          0

      ＊＊  定数領域  ＊＊
    番地               0 . . .  4 . . .  8 . . .  C . . .    0123456789ABCDEF
rdat+00000010          01000000 08000000 20000000 10000000   ........ ........
rdat+00000020          40000000 00000000 5359534F 55542020   @.......SYSOUT
rdat+00000030          80000000 00000000                     ........
```

[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

```
      ＊＊  目的プログラムの大きさ  ＊＊
[1]
    . ｔｅｘｔサイズ  =      10656 バイト
    . ｄａｔａサイズ  =        440 バイト
      ＊＊  実行に必要な領域の大きさ  ＊＊
[2]
    ヒープサイズ      =        976 バイト
    スタックサイズ    =       1168 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
     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.2.6 @CBR_CODE_SET(ファイルのコード系の指定）”
- “C.2.12 @CBR_CONVERT_CHARACTER（コード変換ライブラリの指定）
- “C.2.25 @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.2.11 @CBR_CONSOLE（コンソールウィンドウの種別の指定）”
- “C.2.15 @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.2.16 @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.2.17 @CBR_DISPLAY_CONSOLE_OUTPUT（DISPLAY UPON CONSOLEのイベントログ出力指定）”
- “C.2.18 @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.2.19 @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.2.20 @CBR_DISPLAY_SYSERR_OUTPUT（DISPLAY UPON SYSERRのイベントログ出力指定）”
- “C.2.21 @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.2.22 @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.2.23 @CBR_DISPLAY_SYSOUT_OUTPUT（DISPLAY UPON SYSOUTのイベントログ出力指定）”
- “C.2.34 @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.2.74 @WinCloseMsg（ウィンドウを閉じるときのメッセージ表示の指定）”
- “C.2.7 @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.2.9 @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.2.10 @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.2.85 SYSINのアクセス名（小入出力機能の入力ファイルの指定）”
- “C.2.86 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.2.36 @CBR_MESSAGE（実行時メッセージの出力先の指定）”
- “C.2.37 @CBR_MESS_LEVEL_CONSOLE（実行時メッセージの重大度指定）”
- “C.2.38 @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.2.8 @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.2.28 @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.2.29 @CBR_FILE_LFS_ACCESS（COBOLファイルのサイズを拡張する指定）”
- “C.2.30 @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.2.31 @CBR_FILE_USE_MESSAGE（入出力エラーの実行時メッセージの出力）”
- “C.2.26 @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.2.27 @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.2.75 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.2.45 @CBR_PSFILE_xxx（表示ファイルから使用する接続製品名（宛先ごとの指定））”
- “C.2.76 ファイル識別名（表示ファイルから使用する情報ファイルおよび接続製品名（ファイルごと）の指定）”

プリンタに関するもの

- “C.2.24 @CBR_DocumentName_xxxx（I制御レコードによる文書名の指定）”
- “C.2.40 @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レイに対して有効または無効にする指定）”
- “C.2.41 @CBR_PrinterANK_Size（ANK文字サイズの指定）”
- “C.2.42 @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.2.43 @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.2.44 @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.2.77 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.2.78 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.2.79 FCBxxxx（FCB制御文の指定）”
- “C.2.80 FOVLDIR（フォームオーバレイパターンのフォルダの指定）”
- “C.2.81 FOVLTYPE（フォームオーバレイパターンのファイル名の形式の指定）”
- “C.2.82 FOVLNAME（フォームオーバレイパターンのファイル名の指定）”
- “C.2.83 OVD_SUFFIX（フォームオーバ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（アタッチ形式のデバッグを行う指定）”
- “C.2.35 @CBR_JUSTINTIME_DEBUG（異常終了時に診断機能を使って調査を行う指定）”
- “C.2.39 @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.2.84 SYSCOUNT（COUNT情報の出力ファイルの指定）”

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.2.5 @CBR_ClassInfFile（クラス情報ファイルの指定）”
- “C.2.33 @CBR_InstanceBlock（オブジェクトインスタンスの獲得方法の指定）”

組込み関数に関するもの

- “C.2.32 @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.2.13 @CBR_CSV_OVERFLOW_MESSAGE（CSV形式データ操作時のメッセージ抑止指定）”
- “C.2.14 @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 = { YES / NO }

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

参考

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

C.2.2 @CBR_ATTACH_TOOL（アタッチ形式のデバッグを行う指定）

@CBR_ATTACH_TOOL = 接続先/STUDIO [追加パスリスト]

NetCOBOL Studioでアタッチ形式のデバッグを行う場合に指定します。

接続先

クライアント側のリモートデバッガ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 / ホスト名 / localhost } [:ポート番号]

サーバ側で実行したCOBOLアプリケーションからNetCOBOL Studioのリモートデバッグ機能に接続する場合は、IPアドレスまたはホスト名を指定します。

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。

IPアドレスは、IPv4またはIPv6の形式で指定します。IPアドレスの指定に関する詳細は、“NetCOBOL Studio ユーザーズガイド”を参照してください。

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。

STUDIO

NetCOBOL Studioでのデバッグを表す文字列として、常に指定します。

追加パスリスト

デバッグ情報ファイルの検索フォルダを指定します。デバッグ情報ファイルは、以下の順序で検索され、デバッグに利用されます。

1. 追加パスリストの指定順(複数のフォルダを指定する場合はセミコロン“;”で区切って記述してください)
2. COBOLプログラムが動作し始めたときのカレントフォルダ
3. 起動するCOBOLプログラムの格納フォルダ

注意

2.と3.を追加パスリストに記述する必要はありません。

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 実行用の初期化ファイル”

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_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、“16.3.3.4.1 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.2.6 @CBR_CODE_SET(ファイルのコード系の指定)

@CBR_CODE_SET = { SHIFT_JIS | UTF-8 }

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

実行時のコード系がシフトJISの場合、省略値はSHIFT_JISとなります。実行時のコード系がUnicodeの場合、省略値はUTF-8となります。

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

注)BOMの有無でコード系を識別します。

C.2.7 @CBR_COMPOSER_CONSOLE(Interstage Business Application Serverのログ定義ファイルで定義されている管理名の指定)

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

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

注意

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

C.2.8 @CBR_COMPOSER_MESS(実行時メッセージをInterstage Business Application Serverの汎用ログに出力する指定)

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

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

注意

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

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

C.2.9 @CBR_COMPOSER_SYSERR（Interstage Business Application Serverのログ定義ファイルで定義されている管理名の指定）

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

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

注意

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

C.2.10 @CBR_COMPOSER_SYSOUT（Interstage Business Application Serverのログ定義ファイルで定義されている管理名の指定）

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

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

注意

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

C.2.11 @CBR_CONSOLE(コンソールウィンドウの種別の指定)

@CBR_CONSOLE = { SYSTEM | COBOL }

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

SYSTEM

システムのコンソールを使用します。

COBOL

COBOLのコンソールウィンドウおよびメッセージボックスを使用します。

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

注意

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

C.2.12 @CBR_CONVERT_CHARACTER(コード変換ライブラリの指定)

@CBR_CONVERT_CHARACTER = { ICONV | FJ_ICONV }

実行時にランタイムシステムが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。

本指定を省略した場合、“ICONV”が指定されたものとみなします。

ICONV

NetCOBOLランタイムシステム内の処理でコード変換します。

FJ_ICONV

運用環境にインストールされたInterstage Charset Managerのコード変換関数を使用します。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。

C.2.13 @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.2.14 @CBR_CSV_TYPE（生成するCSV形式のバリエーション）

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

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

この指定は、TYPE指定を省略したSTRING文だけに有効です。

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

C.2.15 @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 = { I | W | E }

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.2.16 @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 x64"となります。

C.2.17 @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Windows(x64)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.2.18 @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 = { I | W | E }

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.2.19 @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 x64”となります。

C.2.20 @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Windows(x64)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.2.21 @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 = { I | W | E }

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.2.22 @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 x64"となります。

C.2.23 @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Windows(x64)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.2.24 @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.2.25 @CBR_ENTRYFILE(エントリ情報ファイルの指定)

@CBR_ENTRYFILE = エントリ情報ファイル名

実行用の初期化ファイル(COBOL85.CBR)のエントリ記述部以外からエントリ情報を指定したい場合、エントリ情報ファイルを作成し、本環境変数情報に指定します。なお、エントリ情報については、“5.6 エントリ情報”または“16.3.2.4.4 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.2.26 @CBR_EXFH_API(外部ファイルハンドラで結合するファイルシステムの入り口名の指定)

@CBR_EXFH_API = 入り口名

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

参照

[参照]“7.9.2 外部ファイルハンドラ”

注意

入り口名を指定する時、大文字と小文字を区別して記述してください。

C.2.27 @CBR_EXFH_LOAD（外部ファイルハンドラで結合するファイルシステムのDLL名の指定）

@CBR_EXFH_LOAD = DLL名

外部ファイルハンドラを使用する時、結合するファイルシステムのDLL名を指定します。

ファイルのパスには、絶対パス、相対パスのどちらも指定することができます。相対パスを用いた場合、カレントディレクトリからの相対パスとなります。

参照

“7.9.2 外部ファイルハンドラ”

C.2.28 @CBR_FILE_BOM_READ（Unicodeの行順ファイルを参照する時の識別コードの扱いの指定）

@CBR_FILE_BOM_READ = { CHECK | DATA | AUTO }

Unicodeの行順ファイルを参照する場合、ファイルに付加されている認識コードの扱いを指定します。

参照

“7.3.3 行順ファイルの処理”

CHECK

レコード定義の字類と一致したBOMが付加されている場合は、識別コードと認識し、READ文の実行でBOMを読み飛ばします。
レコード定義の字類と一致していないBOMが付加されている場合、またはBOMが付加されていない場合は、OPEN文の実行が失敗します。

DATA

BOMが付加されている場合は、BOMをレコードデータの一部として読み込みます。
BOMが付加されていない場合は、ファイルの先頭からレコードを読み込みます。

AUTO

レコード定義の字類と一致したBOMが付加されている場合は、識別コードと認識し、READ文の実行でBOMを読み飛ばします。
レコード定義の字類と一致していないBOMが付加されている場合は、OPEN文の実行が失敗します。
BOMが付加されていない場合は、ファイルの先頭からレコードを読み込みます。

C.2.29 @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、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.8 索引ファイル ”

注意

- 以下の環境で本環境変数を指定する場合、全ての処理で本環境変数を有効にしてください。

  ― 他のCOBOLプログラムとファイルを共有している

  ― 対象となるCOBOLファイルを、ツールおよび他製品からアクセスしている

  本環境変数が有効でない場合、COBOLファイルの排他制御の動作は保証されません。

- 相対ファイルに対して、本環境変数は有効になりません。本環境変数を指定しても、ファイルサイズは拡張されません。

C.2.30 @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.2.31 @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.2.32 @CBR_FUNCTION_NATIONAL(NATIONAL関数の変換モードの指定)

@CBR_FUNCTION_NATIONAL = [{ MODE-1 / MODE-2 }][, { MODE-3 / MODE-4 }]

NATIONAL関数の変換モードを指定します。

NATIONAL関数の変換モードには、第1オペランドにNetCOBOL V50L10(注)以前と同じ変換を行う(MODE1)か、視覚的により近い変換を行う(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。

注 ： 本製品とは別売の、32ビットWindowsで動作するNetCOBOLシリーズ製品

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「ー」(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「ー」(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「ー」(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「−」(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.2.33 @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.2.34 @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.2.35 @CBR_JUSTINTIME_DEBUG(異常終了時に診断機能を使って調査を行う指定)

@CBR_JUSTINTIME_DEBUG = { [{ APLERR / CBLERR / ALLERR }][,SNAP [起動パラメタ]] / NO }

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"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。

SNAP

調査手段に診断機能を使う指定として、起動パラメタを指定する時は、必ず指定する。

起動パラメタ

診断機能の起動パラメタを指定する。詳細は、“表20.2 起動パラメタ”を参照してください。

注意

NetCOBOL Studioのデバッグ機能などでデバッグしている場合は、無効です。

C.2.36 @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Windows(x64)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.2.37 @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.2.38 @CBR_MESS_LEVEL_EVENTLOG（実行時メッセージの重大度指定）

@CBR_MESS_LEVEL_EVENTLOG = { 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.2.39 @CBR_MEMORY_CHECK（メモリチェック機能を使って検査を行う指定）

@CBR_MEMORY_CHECK = MODE1

アプリケーション実行時、ランタイムシステム領域を、メモリチェック機能を使って検査します。メモリチェック機能の使い方については、“19.5 メモリチェック機能”を参照してください。

C.2.40 @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 }

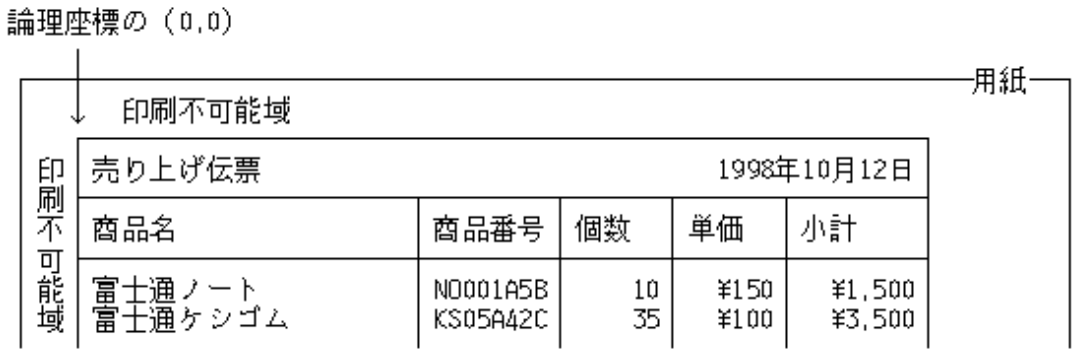
FORMAT句なし印刷ファイルで、I制御レコードに指定されたとじしろ方向(BIND)、とじしろ幅(WIDTH)および印刷原点位置(OFFSET)機能をフォームオーバレイに対しても有効にする(VALID)か、無効にする(INVALID)かを指定します。本指定の省略時は、”INVALID”が指定されたものとみなされます。なお、本指定の有効範囲は実行単位内です。ファイル単位に有効な指定については、“8.1.12 印刷情報ファイル”を参照してください。I制御レコードについては、“8.1.9 I制御レコード/S制御レコード”を参照してください。

参考

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。

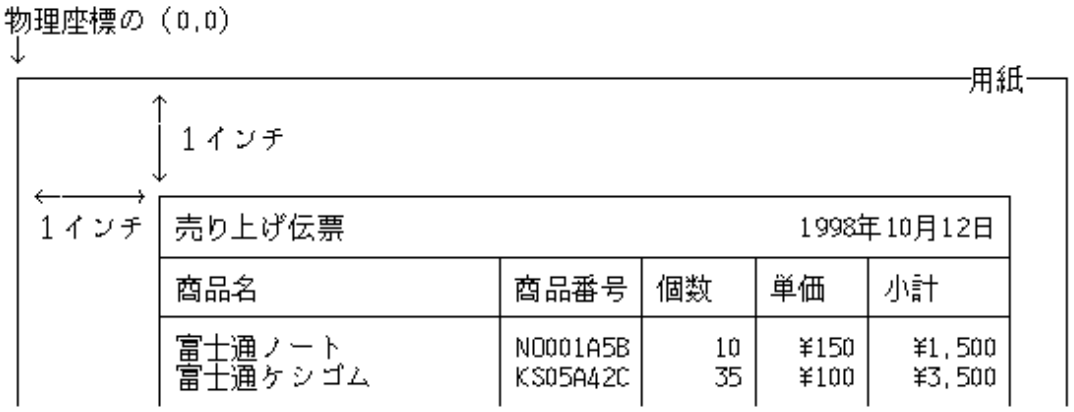
以下の図では、罫線=オーバレイ、文字=行レコードを示しています。

- 印刷原点位置を指定しない場合の印刷結果

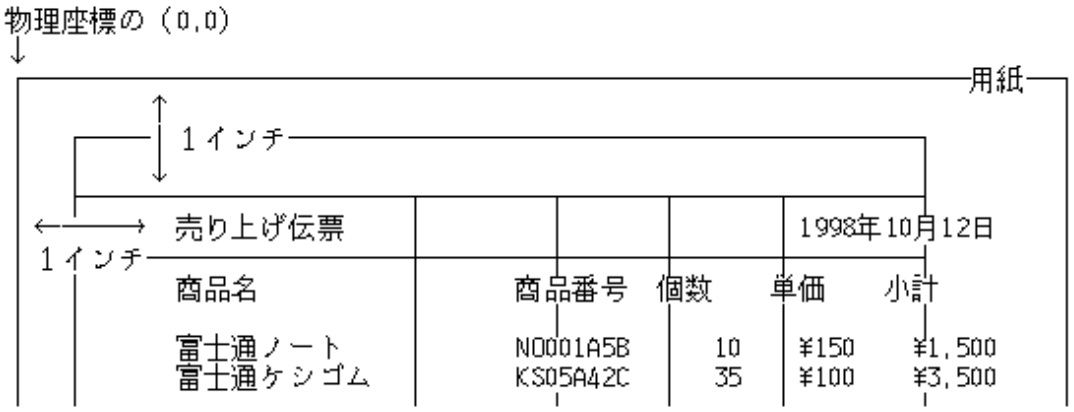


・ X,Y座標にそれぞれ1インチ原点オフセットを設けた場合の印刷結果

－ VALID指定の場合



－ INVALID指定の場合



C.2.41 @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.2.42 @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.2.77 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.2.78 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.2.43 @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.2.44 @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。

C.2.45 @CBR_PSFILE_xxx（表示ファイルから使用する接続製品名（宛先ごとの指定））

@CBR_PSFILE_xxx = 接続製品名

---

xxxには、表示ファイルのSYMBOLIC DESTINATION句に記述した、宛先名 PRT を指定します。

接続製品名には、実際に使用する接続製品を示す文字列を指定します。指定する文字列を以下に示します。

表C.6 接続製品名の指定形式

SYMBOLIC DESTINATION句の指定	使用する製品	接続製品名を示す文字列
PRT	MeFt	MEFT または 省略
	MeFt/NET	MEFTNET
DSP	MeFt	MEFT または 省略

注意

- ファイル識別名に接続製品名の指定がある表示ファイルでは、ここで指定する接続製品名は無効になります。[参照]“C.2.76 ファイル識別名（表示ファイルから使用する情報ファイルおよび接続製品名（ファイルごと）の指定）”

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 = { CENTER | SYSTEM | (x, y) }

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]“18.5.3 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 = { TOP | BOTTOM }

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]“18.3.3 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]“I.1.8 データロック獲得サブルーチン(COB_LOCK_DATA)”、“I.1.10 オブジェクトロック獲得サブルーチン(COB_LOCK_OBJECT)”

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。

参照

“19.3 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(VALID)かを指定します。

本指定を省略した場合、“VALID”が指定されたものとみなします。

注意

削除される空白はコード系に依存し、以下のようになります。

- コード系がUnicodeで字類が英数字の場合は、半角空白が削除されます。
- コード系がUnicodeで字類が日本語の場合は、全角空白が削除されます。
- コード系がUnicode以外の場合は、半角空白および全角空白が削除されます。

C.2.57 @CnslBufLine（コンソールウィンドウのバッファ数の指定）

@CnslBufLine = { バッファ行数 / 100 }

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.2.11 @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.2.11 @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.2.11 @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）"

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、“第17章 サーバ・タイプのアプリケーション”を参照してください。

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.2.74 @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コネクション(CONNECT文などで指定する)を確立するために使用されます。

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ース名を指定します。

注意

- フォントフェース名は、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。
- フォントフェース名の前後に、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ース名の一部とみなされます。
- フォントフェー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。
- フォントフェー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。
  - 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。
  - 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に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。
  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. 〔デバイスとプリンター〕のメニューバーから“プリントサーバー プロパティ”を選択し、〔用紙〕ページで作成します。

例

I制御レコードのSIZEフィールドの指定

```
ABC
```

xxx 部分置換後の環境変数情報名

```
@PRN_FormName_ABC
```

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

- アイコン表示を抑止した場合、システムのタスクマネージャ上に該当アプリケーション名が出力されなくなり、該当アプリケーションを強制終了する方法がなくなります。

  また、アイコン表示を抑止した場合、環境変数情報@ExitSessionMSGの指定は無効になります。この場合、COBOLプログラムの実行中にWindowsシステムを終了させると、システムの警告メッセージが出力されます。システムの警告メッセージが出力され、プログラムの強制終了を行った場合、強制終了したプログラムで行っていた入出力処理の結果は保証されません。

- マルチスレッドモードでの動作時には、翻訳オプションMAINを指定して翻訳されていない場合、アイコンは表示されません。

C.2.74 @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.2.75 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ファイルを処理します。[参照]“7.9.1 Btrieveファイル”

EXFH

外部ファイルハンドラを処理します。[参照]“7.9.2 外部ファイルハンドラ”

MOD

ファイルの追加書きを処理します。[参照]“7.7.5 ファイル追加書き”

CONCAT

ファイルの連結を処理します。[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.2.76 ファイル識別名（表示ファイルから使用する情報ファイルおよび接続製品名（ファイルごと）の指定）

ファイル識別名 = [情報ファイル名][,接続製品名]

表示ファイルを利用する場合のファイル識別名の指定方法について説明します。

ファイル識別名には、COBOLソースプログラムのASSIGN句に記述したファイル識別名を指定します。プログラム中でファイル識別名を英小文字で記述した場合でも、環境変数情報のファイル識別名は英大文字で指定してください。

情報ファイル名の指定は、MeFtを使用する場合は必須となりますが、MeFt以外を使用する場合は省略できます。情報ファイルには、実際に使用するMeFtのウィンドウ情報ファイル名またはプリンタ情報ファイル名を指定します。

接続製品名には、実際に使用する接続製品を示す文字列を指定します。接続製品を示す文字列については“表C.6 接続製品名の指定形式”を参照してください。接続製品名の指定を省略した場合、環境変数情報@CBR_PSFILE_xxxに指定された接続製品名が有効になります。[参照]“C.2.45 @CBR_PSFILE_xxx（表示ファイルから使用する接続製品名（宛先ごとの指定））”

接続製品名の指定が省略され、かつ環境変数情報@CBR_PSFILE_xxxの指定もない場合、表示ファイルの宛先指定に応じた省略値が採用されます。省略値については、“表C.6 接続製品名の指定形式”の“省略”と記述された欄を参照してください。

例

COBOLソースプログラムのASSIGN句の記述

```
ASSIGN TO GS-PRTFILE
```

COBOLソースプログラムのSYMBOLIC DESTINATION句の記述

```
SYMBOLIC DESTINATION IS "PRT"
```

実際に使用する情報ファイルの名前

```
C:¥WORK.PRC
```

実際に使用する接続製品

```
MeFt
```

環境変数情報

```
PRTFILE=C:¥WORK.PRC,MEFT
```

C.2.77 ファイル識別名（プログラムで使用するプリンタ情報ファイルおよび各種パラメタの指定）

ファイル識別名 = プリンタ情報ファイル名[,[,FONT(フォントテーブル名)]]

FORMAT句付き印刷ファイルを利用する場合のファイル識別名の指定方法について説明します。

ファイル識別名には、COBOLソースプログラムのASSIGN句に記述したファイル識別名を指定します。プログラム中でファイル識別名を英小文字で記述した場合でも、環境変数情報のファイル識別名は英大文字で指定してください。

プリンタ情報ファイルには、実際に使用するMeFtのプリンタ情報ファイル名を指定します。

フォントテーブル名には、フォントテーブルのファイル名を絶対パスで指定します。フォントテーブルには、PRINTING MODE句のFONT-nnn(書体番号)指定に対応するフォントフェース名や印字スタイルの情報を定義します。ここで指定されたフォントテーブルは、そのフ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.2.42 @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.PRC
```

環境変数情報

```
PRTFILE=C:¥WORK.PRC
```

例2

COBOL ソースプログラムのASSIGN句の記述

```
ASSIGN TO PRTFILE
```

実際に使用するプリンタ情報ファイルの名前

```
C:¥WORK.PRC
```

実際に使用するフォントテーブルの名前

```
C:¥DEFAULT.FTB
```

環境変数情報

```
PRTFILE=C:¥WORK.PRC,,FONT(C:¥DEFAULT.FTB)
```

C.2.78 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）

---

file-Identifier = { LPTn: / COMn: / PRTNAME:プリンタ名 } [,[,INF(xx)] [,FONT(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。

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ース名や印字スタイルの情報を定義します。ここで指定されたフォントテーブルは、そのフ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.2.42 @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.2.79 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.2.80 FOVLDIR（フォームオーバ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.2.81 FOVLTYPE（フォームオーバ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.2.82 FOVLNAME（フォームオーバ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
```

C.2.83 OVD_SUFFIX（フォームオーバ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.2.84 SYSCOUNT(COUNT情報の出力ファイルの指定)

SYSCOUNT = ファイル名[,MOD]

COUNT機能を使用する場合、COUNT情報の出力先となるファイルの名前を指定します。[参照]“19.4 COUNT機能”

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.2.85 SYSINのアクセス名(小入出力機能の入力ファイルの指定)

SYSIN のアクセス名 = ファイル名[,DUMMY]

SYSINのアクセス名には、翻訳オプションSSINに指定した環境変数情報名を指定し、ファイル名には、小入出力機能のACCEPT文を実行したときのデータの入力先となるファイルの名前を指定します。[参照]“A.3.48 SSIN(ACCEPT文のデータの入力先)”

ファイル名には、絶対パスと相対パスを指定できます。相対パスが指定された場合は、カレントフォルダからの相対パスになります。

例

COBOL ソースプログラムを翻訳するときの翻訳オプションSSINの指定

```
SSIN(INFILE)
```

ACCEPT文の入力先ファイルの名前

```
A:¥INDATA.TXT
```

環境変数情報

```
INFILE=A:¥INDATA.TXT
```

指定した文字列は以下のような意味を持ちます。

DUMMY

ダミーファイル機能を使用します。

[参照]“11.1.6.5 ACCEPT文のファイル入力拡張機能”

C.2.86 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.3.49 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、以下のどちらかの状態である。 ・ 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。 ・ 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、誤りではない。
	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も長い。
		MeFt	表示ファイルで、READ文の実行は成功したが、以下のどれかの状態である。 ・ 全桁必須入力項目で入力エラーが発生した。 ・ 入力必須項目で入力エラーが発生した。 ・ 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） ・ データ項目に入力されたデータに誤りがある。
	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。 ・ 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であった。 ・ 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" ）となる。 ・ 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。
	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。 ・ NO REWIND 指定のOPEN文またはCLOSE文 ・ REEL/UNIT 指定のCLOSE文
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となった。 ・ ファイルの終わりに達した。 ・ 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場合のことを指す。
	14	－	順呼出しのREAD文でファイル終了条件となった。 ・ 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。

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。 ・ 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。 ・ 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。 ・ 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。
	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。 ・ 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。 ・ 相対レコード番号に0が指定された。
	24	－	次の状態のどれかである。 ・ WRITE文またはCLOSE文の実行で、領域不足が発生した。 ・ WRITE文の実行で、指定されたキーがキーレンジ外である。 ・ 区域外書出し発生後、さらにWRITE文を実行しようとした。
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。 ・ 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か、またはレコード長として誤った数値が指定された。 ・ REWRITE文実行時に、そのレコードの長さは書き換えるレコードの長さと等しくない。
	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。 ・ 先行するSTART文が不成功。 ・ 先行するREAD文が不成功（ファイル終了条件も含む）。
	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文を実行しました。
	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文を実行しました。
	49	－	I-Oモードで開かれていないファイルに対してREWRITE文またはDELETE文を実行した。

分類	値	詳細	意味
その他の誤り（不成功）	90	－	他のどれにも含まれないエラーである。以下のような状態が考えられる。 ・ ファイル情報が不完全またはその情報に誤りがある。 ・ OPEN/CLOSE文の実行時に、関数で何らかのエラーが発生した。 ・ 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。 ・ 主記憶などの資源が利用できない。 ・ 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。 ・ 区域外書出しによる誤りの発生後、レコードを書き出そうとした。 ・ no-space状態発生後、レコードを書き出そうとした。 ・ テキストファイルのレコードに不当な文字がある。 ・ コードセットに変換できない文字がある。 ・ 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。 ・ 必要な関連製品のローディングに失敗した。 ・ 上記以外の誤りがある。その入出力動作に関してはそれ以上の情報がない。 ・ システムエラーが発生した。
		MeFt	MeFtがエラーを検出した。
	91	－	・ ファイルが割り当てられていない。 ・ 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。
	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。

注意

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。

[参照]“NetCOBOLメッセージ集”の“MeFtのメッセージ”

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)	256G
レコードの最大長（バイト数）	COBOLファイルシステム	32760
	Btrieve(*2)	32760
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

*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、“Pervasive PSQL V10 SP1”の情報をもとにしています。

E.5 印刷ファイル

項目	値
レコードの最大長（バイト数）	32760(*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

項目	値
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)	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)	256G
レコードの最大長(バイト数)	COBOLファイルシステム	32760
	Btrieve(*2)	32760
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
RECORD KEY句のデータ項目の個数の最大	COBOLファイルシステム	254 (*3)
	Btrieve(*2)	204 (*4)
RECORD KEY句のデータ項目の長さの総和の最大値(バイト数)	COBOLファイルシステム	254 (*5)
	Btrieve(*2)	255
ALTERNATE RECORD KEY句のデータ項目の個数の最大	COBOLファイルシステム	254 (*3)
	Btrieve(*2)	203 (*4)
1つのALTERNATE RECORD KEY句のデータ項目の長さの総和の最大値(バイト数)	COBOLファイルシステム	254 (*5)
	Btrieve(*2)	255
ALTERNATE RECORD KEY句の個数の最大	COBOLファイルシステム	125
	Btrieve(*2)	118

*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、“Pervasive PSQL V10 SP1”の情報をもとにしています。

*3 ： RECORD KEY句とALTERNATE RECORD KEY句のデータ項目の総数は、最大255個

*4 ： Btrieveでは、RECORD KEY句とALTERNATE RECORD KEY句に指定できるデータの個数の総和は、最大204個

*5 ： RECORD KEY句とALTERNATE RECORD KEY句のデータ項目の総長は、最大255バイト

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

項目	値
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）	32760
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)
CONNECT文により同時に接続できるコネクション数の最大(18.2.2.3)	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。(コマンド引数から指定する場合は？)

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１１    PIC XX OCCURS 10 TIMES.
    02  項目２ OCCURS 35 TIMES.
        03  項目２１    PIC XX OCCURS 10 TIMES.
       :
    MOVE SPACE TO  項目１１ (添字１, 添字２).              … [1]
       :
    MOVE SPACE TO  項目２１ (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.   |
       :                                                 |ループ
    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]     |
       :                                                 |ループ
    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”が参照されていなければ、[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Ａ１ PIC X(32).
02  基本項目Ａ２ PIC X(16).
 :
02  基本項目Ｂ１ PIC X(32).
02  基本項目Ｂ２ PIC X(16).
 :
MOVE  基本項目Ａ１ TO 基本項目Ｂ１.                    … [1]
 :
MOVE  基本項目Ａ２ TO 基本項目Ｂ２.                    … [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.3.33 OPTIMIZE（広域最適化の扱い）”を参照してください。

以下は、広域最適化での注意事項です。

連絡機能を使用する場合

呼ばれるプログラムに対し、“CALL "SUB" USING A,A.”や“CALL "SUB" USING A,B. (注)”のように、同時に指定された複数のパラメタにおいて、領域の一部または全部を共有しているものがあります。呼ばれるプログラムでその内容を書き換えていると、呼ばれるプログラムの最適化により、意図したとおりの結果が得られない場合があります。

注 ： ただし、AとBは領域の一部を共有します。

広域最適化が行われない場合

以下のプログラムでは、広域最適化は行われません。

- 広域最適化の対象となる属性を持った項目および指標名が1つも定義されていないプログラム
- 区分化機能を使用しているプログラム

以下のプログラムでは、広域最適化の効果は少なくなります。

- 入出力操作を主とし、もともとCPUをあまり使わないプログラム
- 数字項目を使わず、英数字項目ばかりを使うプログラム
- 宣言部分から非宣言部分を参照しているプログラム
- 非宣言部分から宣言部分を参照しているプログラム
- 翻訳オプションTRUNCを指定しているプログラム

参照

“A.3.55 TRUNC（桁落とし処理の可否）”

デバッグを行う場合

以下の注意が必要です。

- 広域最適化によって文の削除、移動および変更が行われるので、データ例外などのプログラム割込みの起こる回数や場所が変わることがあります。
- プログラム割込みなどで中断したとき、プログラム上の記述でデータ項目に値を設定していても、実際には設定されていないことがあります。
- 再帰的に定義される項目が内部10進項目または外部10進項目の場合、翻訳オプションNOTRUNCが指定されていると、意図したとおりにプログラムが動作しないことがあります。

参照

“A.3.55 TRUNC（桁落とし処理の可否）”

- 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　特殊な定数の書き方

ここでは、プログラム名やファイル名などのシステムで定められた名前を指定する、各種定数の書き方を説明します。

G.1 プログラム名定数

プログラム名段落(PROGRAM-ID)、CALL文およびCANCEL文に定数指定でプログラム名を指定する場合、使用できる文字に制限はありません。ただし、定数の長さは60バイト以内でなければなりません。指定した文字がリンカの規則に従っているかどうかは、利用者が判断します。

プログラム名定数の長さは60バイト以内でなければなりません。

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、カレントフォルダ中のファイルとみなします。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らないことを除けば、使用できる文字およびその構成規則に制限はありません。したがって、リンカの規則に従っているかどうかは、利用者が判断します。

付録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 COBOLファイルユーティリティ関数
- H.3 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
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
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

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-INTEGER (INT).
     MOVE OUT-YMD TO OUT-YMD-ED.
     DISPLAY "基準日から" INT "日目は，" OUT-YMD-ED "です．".
```

・ 実行結果

```
2002年12月25日は，基準日から+000146821日目です.
基準日から+000146821日目は，2002/12/25です.
```

数字関数

数字関数は、整数関数と同様、算術式中にだけ記述できます。数字関数を算術式以外の場所、たとえばMOVE文の送出し側に記述することはできません。

英数字関数

英数字関数は、英数字項目が記述できるところに書くことができます。

例

UPPER-CASE関数

・ COBOL プログラムの記述

```
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 LOWCASE       PIC X(13) VALUE "fujitsu cobol".
 PROCEDURE DIVISION.
     DISPLAY "変換前：" LOWCASE.
     DISPLAY "変換後：" FUNCTION UPPER-CASE(LOWCASE).
```

・ 実行結果

```
変換前：fujitsu cobol
変換後：FUJITSU COBOL
```

例では、大文字に変換した文字を直接DISPLAY文で表示していますが、MOVE文の送出し側に記述して作業域に転記することもできます。

日本語関数

日本語関数は、日本語項目が記述できるところに書くことができます。
例では、変換した文字を一度転記してから表示します。

例

NATIONAL関数

・ COBOL プログラムの記述

```
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 NCHAR  PIC N(7).
 01 CHAR   PIC X(7) VALUE "FUJITSU".
 PROCEDURE DIVISION.
*英数字を日本語文字に変換して表示
     MOVE FUNCTION NATIONAL(CHAR) TO NCHAR.
     DISPLAY "英数字 : " CHAR.
     DISPLAY "日本語 : " NCHAR.
```

・ 実行結果

```
英数字 : FUJITSU
日本語 : ＦＵＪＩＴＳＵ
```

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≦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 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.
```

注意

- COBOLファイルユーティリティ関数の登録集を取り込むCOPY文のREPLACING指定、またはこのCOPY文が置き換えの対象となるようなREPLACE文の記述はしないでください。
- 登録集を変更した場合、動作は保証されません。
- 各COBOLファイルユーティリティ関数を呼び出す前に、必ずインタフェース領域のCOBF-INFをLOW-VALUEで初期化してください。

リンク時

翻訳オプションDLOADを指定していないプログラムから利用する場合、F4AGFUTC.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=F4AGFUTC.DLL
COB_FILE_LOAD=F4AGFUTC.DLL
COB_FILE_UNLOAD=F4AGFUTC.DLL
COB_FILE_SORT=F4AGFUTC.DLL
COB_FILE_REORGANIZE=F4AGFUTC.DLL
COB_FILE_COPY=F4AGFUTC.DLL
COB_FILE_MOVE=F4AGFUTC.DLL
COB_FILE_DELETE=F4AGFUTC.DLL
```

H.2.1 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進形式といいます。
- Unicodeコード系で動作する場合、日本語項目および日本語編集項目をUCS-2形式/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。

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。	－	－	○

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-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

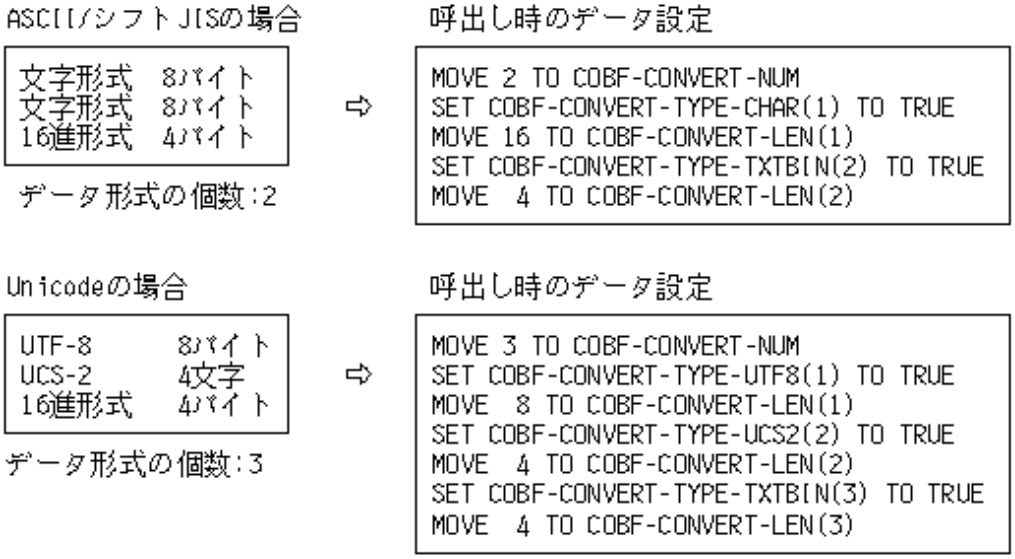
 例

[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。



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.2.2 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。

値	意味
-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 "I" TO COBF-OUTPUT-ATTR.
MOVE "V" TO COBF-OUTPUT-RECFM.
MOVE 80 TO COBF-OUTPUT-RECLEN.
MOVE 3 TO COBF-OUTPUT-KEYNUM.
MOVE 0 TO COBF-OUTPUT-OFFSET(1).
MOVE 5 TO COBF-OUTPUT-KEYLEN(1).
MOVE "C" TO COBF-OUTPUT-KEYCONT(1).
MOVE 5 TO COBF-OUTPUT-OFFSET(2).
MOVE 10 TO COBF-OUTPUT-KEYLEN(2).
MOVE "E" TO COBF-OUTPUT-KEYCONT(2).
MOVE "D" TO COBF-OUTPUT-KEYDUP(2).
MOVE 5 TO COBF-OUTPUT-OFFSET(3).
MOVE 5 TO COBF-OUTPUT-KEYLEN(3).
MOVE "E" TO COBF-OUTPUT-KEYCONT(3).
MOVE "C" TO COBF-LOAD-COND.
CALL "COB_FILE_LOAD" USING BY REFERENCE COBF-INF.
```

- レコード順ファイルの内容を相対ファイルに追加する(拡張)

```
入力ファイル名： C:¥INFILE
出力ファイル名： C:¥RELFILE
ファイル編成　： 相対ファイル
レコード形式　： 固定長
レコード長　　： 80バイト
```

```
MOVE LOW-VALUE TO COBF-INF.
MOVE "C:¥INFILE" TO COBF-INPUT-FILENAME.
MOVE "C:¥RELFILE" TO COBF-OUTPUT-FILENAME.
MOVE "R" TO COBF-OUTPUT-ATTR.
MOVE "F" TO COBF-OUTPUT-RECFM.
MOVE 80 TO COBF-OUTPUT-RECLEN.
```

```
MOVE "E" TO COBF-LOAD-COND
CALL "COB_FILE_LOAD" USING BY REFERENCE COBF-INF.
```

H.2.3 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.2.4 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
COBF-SORT-KEYATTR-PIC9C5	19	PIC 9() COMP-5
COBF-SORT-KEYATTR-PICS9C5	20	PIC S9() COMP-5
COBF-SORT-KEYATTR-PICC1	21	COMP-1
COBF-SORT-KEYATTR-PICC2	22	COMP-2

注1:文字コードは、翻訳オプションENCODEに従います。翻訳オプションENCODEと異なる文字コードが指定されたデータ項目をキーとした整列処理はできません。

COBF-SORT-KEYSEQ

整列順序を、以下の文字で指定します。

文字	意味
"A"	昇順にレコードを整列する
"D"	降順にレコードを整列する

復帰コード

特殊レジスタPROGRAM-STATUSを使用して受け取ります。

復帰値	意味
0	ファイルの整列に成功しました。
-1	ファイルの整列に失敗しました。 エラーの詳細は、COBF-MESSAGEに文字列で設定されます。

例

- 相対ファイルを整列し、結果をOUTFILEに出力する

```
入力ファイル名：  RELFILE
出力ファイル名：  OUTFILE
整列条件　　：　［第１キー］項目属性　　　：日本語
　　　　　　　　　　　　　　レコード内オフセット：0
　　　　　　　　　　　　　　大きさ　　　　：2
　　　　　　　　　　　　　　整列順序　　　：昇順
　　　　　　　　［第２キー］項目属性　　　：英数字
　　　　　　　　　　　　　　レコード内オフセット：10
　　　　　　　　　　　　　　大きさ　　　　：5
　　　　　　　　　　　　　　整列順序　　　：降順
ファイル属性　：　ファイル編成：相対ファイル
　　　　　　　　　レコード形式：可変長
　　　　　　　　　レコード長　：80
```

```
      MOVE LOW-VALUE TO COBF-INF.
      MOVE "RELFILE" TO COBF-INPUT-FILENAME.
      MOVE "R" TO COBF-INPUT-ATTR.
      MOVE "V" TO COBF-INPUT-RECFM.
      MOVE 80 TO COBF-INPUT-RECLEN.
      MOVE "OUTFILE" TO COBF-OUTPUT-FILENAME.
      MOVE 2 TO COBF-SORT-KEYNUM.
      SET  COBF-SORT-KEYATTR-PICN(1) TO TRUE.
      MOVE 0 TO COBF-SORT-OFFSET(1).
      MOVE 2 TO COBF-SORT-KEYLEN(1).
      MOVE "A" TO COBF-SORT-KEYSEQ(1).
      SET  COBF-SORT-KEYATTR-PICX(2) TO TRUE.
      MOVE 10 TO COBF-SORT-OFFSET(2).
      MOVE 5 TO COBF-SORT-KEYLEN(2).
      MOVE "D" TO COBF-SORT-KEYSEQ(2).
      CALL "COB_FILE_SORT" USING BY REFERENCE COBF-INF.
```

H.2.5　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。

復帰値	意味
-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.2.6　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.2.7 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:¥IXDFILE
移動先ファイル名　　: F:¥OUTFILE

```
MOVE LOW-VALUE TO COBF-INF.
MOVE "C:¥INFILE" TO COBF-INPUT-FILENAME.
MOVE "F:¥OUTFILE" TO COBF-OUTPUT-FILENAME.
CALL "COB_FILE_MOVE" USING BY REFERENCE COBF-INF.
```

H.2.8 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.3 COBOLファイルアクセスルーチン

H.3.1 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.3.2 API関数一覧

表H.2 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()	順読込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()	順読込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_rdnext()	レコードを順に読み込む	S
cobfa_rdrec()	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_wrnext()	レコードを順に書き出す	S
cobfa_wrrec()	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.3 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.4 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_recnum()	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.5 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.3.3 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.3.4 準備するもの

開発言語として64ビット(x64)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.3.5 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ランタイムシステムと同じフォルダに格納されます。

NetCOBOLランタイムシステムは、Windowsがインストールされているドライブの以下のフォルダにインストールされます。

NetCOBOLランタイムシステムのインストールフォルダ

```
¥Program Files¥Common Files¥Fujitsu¥NetCOBOL
```

環境変数PATHと環境変数LIBにNetCOBOLランタイムシステム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。設定されていなければ、これらの環境変数にNetCOBOLランタイムシステム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.3.6 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.3.39 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 "f4agfcfa.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ランタイムシステムのインストール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ランタイムシステムのインストールフォルダを指定します)

例

/I "C:¥Program Files¥Common Files¥Fujitsu¥NetCOBOL"

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ランタイムシステムのインストールフォルダにある、以下のファイルを結合します。

- F4AGFCFA.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.3.7 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.3.33 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。

エラー番号	入出力状態	説明
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っています。
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っています。
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っています。
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.3.36 エラー番号”と“H.3.37 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

- 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)を指定したものとして扱います。
- 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。
- 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。読込み時は、レコードロックの指定が無効になります。
- 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。
- 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失敗となります。

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。読込み時のレコードロック指定については、“H.3.10 レコードの読込み (cobfa_rdkey())”、“H.3.11 レコードの読込み (cobfa_rdnext())”または“H.3.12 レコードの読込み (cobfa_rdrec())”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。

- レコードの読込み(cobfa_rdkey/cobfa_rdnext/cobfa_rdrec)
- レコードの書出し(cobfa_wrkey/cobfa_wrnext/cobfa_wrrec)
- レコードの削除(cobfa_delcurr/cobfa_delkey/cobfa_delrec)
- レコードの書換え(cobfa_rewcurr/cobfa_rewkey/cobfa_rewrec)
- レコードの位置決め(cobfa_stkey/cobfa_strec)

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、行順ファイルが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.3.32 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.3.8 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.3.7 ファイルのオープン (cobfa_open())”と同じです。

H.3.9 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	○

オープン属性	種別	実行の可否
	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	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.3.36 エラー番号”と“H.3.37 入出力状態”を参照してください。

H.3.10 レコードの読込み (cobfa_rdkey())

```
long cobfa_rdkey (
       long                     fd,        /* ファイルディスクリプタ */
       long                     readflgs,  /* 読込み属性             */
       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。(乱読込み)

ファイルディスクリプタfdが指すファイルから、レコード域recareaに含まれる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が読み込まれ、レコード域recareaに格納されます。

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、“読込み属性の指定値(cobfa_rdkey)”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.3.32 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。

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、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。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.3.36 エラー番号”と“H.3.37 入出力状態”を参照してください。

読込み属性の指定値(cobfa_rdkey)

これらは省略可能です。(*は省略値)

a. 読込みモード

	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.3.11 レコードの読込み (cobfa_rdnext())

```
long cobfa_rdnext (
       long  fd,         /* ファイルディスクリプタ */
       long  readflgs,   /* 読込み属性             */
       char *recarea     /* レコード域             */
    );
```

説明

レコードを順に読み込みます。(順読込み)

ファイルディスクリプタfdが示すファイルから、位置付けられているレコードの次(または前)のレコードが読み込まれ、レコード域recareaに格納されます。

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

読込み属性(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、“読込み属性の指定値(cobfa_rdnext)”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。

エラー番号	入出力状態	説明
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	当関数を実行することができない読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.3.36 エラー番号”と“H.3.37 入出力状態”を参照してください。

読込み属性の指定値(cobfa_rdnext)

これらは省略可能です。(*は省略値)

a. 読込みモード

	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.3.12 レコードの読込み (cobfa_rdrec())

```
long cobfa_rdrec (
       long             fd,         /* ファイルディスクリプタ */
       long             readflgs,   /* 読込み属性             */
       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示すレコードを読み込みます。(乱読込み)

ファイルディスクリプタfdが示すファイルから、相対レコード番号recnumが指すレコードが読み込まれ、レコード域recareaに格納されます。

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

読込み属性(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、“読込み属性の指定値(cobfa_rdrec)”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、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読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.3.36 エラー番号”と“H.3.37 入出力状態”を参照してください。

読込み属性の指定値(cobfa_rdrec)

これらは省略可能です。(*は省略値)

a. 読込みモード

	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.3.13 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に従ってレコードを書き出す位置が決まります。(乱書出し)

主レコードキーはレコード域recareaに含まれます。主レコードキーの値に従ってレコードの書出し位置が決まり、レコード域recareaの内容が書き出されます。

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)

書き出すレコードを格納します。

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。
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.3.36 エラー番号”と“H.3.37 入出力状態”を参照してください。

H.3.14 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)

順書出しによって位置付けられているレコード位置に、レコード域recareaの内容が書き出されます。

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)

書き出すレコードを格納します。

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.3.36 エラー番号”と“H.3.37 入出力状態”を参照してください。

H.3.15 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

相対レコード番号に従って、レコードを書き出す位置が決まります。(乱書出し)

レコード域recareaの内容が書き出されます。

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)

書き出すレコードを格納します。

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	○

オープン属性	種別	実行の可否
	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	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.3.36 エラー番号”と“H.3.37 入出力状態”を参照してください。

H.3.16 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

順読込みしたレコードを削除します。(順削除)

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_ELOCKED	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.3.36 エラー番号”と“H.3.37 入出力状態”を参照してください。

H.3.17 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)

レコード域recareaに含まれる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	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.3.36 エラー番号”と“H.3.37 入出力状態”を参照してください。

H.3.18 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が示すファイルで、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。

レコード域(recarea)

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	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	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.3.36 エラー番号”と“H.3.37 入出力状態”を参照してください。

H.3.19 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)

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。

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.3.16 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.3.20 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)

主レコードキーはレコード域recareaに含まれます。主レコードキーの値が指すレコードが、レコード域recareaの内容に書き換えられます。

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	ー
	レコード順ファイル	ー
	相対ファイル	ー
	索引ファイル	○
オープンモード	INPUTモード	ー
	OUTPUTモード	ー
	I-Oモード	○
	EXTENDモード	ー
呼出し法	順呼出し	ー
	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。

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ました。
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.3.17 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.3.21 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    *recarea,  /* 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)

レコード域recareaの内容で書き換えられます。

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.3.18 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.3.22 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。

レコード域recareaに含まれる任意のレコードキーの値に従ってレコードの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 =
	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 >=

	記号定数	意味	対応するCOBOL構文
	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. 逆順読込みフラグ

	記号定数	意味	対応するCOBOL構文
*	なし	順読込み時に論理的に順方向に読み込む	―
	FA_REVORD	順読込み時に論理的に逆方向に読み込む	START WITH REVERSED ORDER

- 逆順読込みフラグのデフォルト値は、正順読込みになります。
- 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、逆順読込みフラグ(FA_REVORD)を指定することはできません。
- 逆順読込みフラグの指定は、乱読込み、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.3.32 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

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	―

オープン属性	種別	実行の可否
	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。

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.3.36 エラー番号”と“H.3.37 入出力状態”を参照してください。

H.3.23 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。

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.3.36 エラー番号”と“H.3.37 入出力状態”を参照してください。

H.3.24 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

ファイルに対して、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.3.36 エラー番号”と“H.3.37 入出力状態”を参照してください。

H.3.25 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               funccode  /* 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。

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(funccode)

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.3.34 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.3.32 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。

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

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ありません。

H.3.26 エラー番号の取得 (cobfa_errno())

```
long cobfa_errno (
      void  /* 引数なし */
    );
```

説明

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.3.36 エラー番号”を参照してください。

H.3.27 入出力状態の取得 (cobfa_stat())

```
long cobfa_stat (
      void  /* 引数なし */
    );
```

説明

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.3.28 読込みレコード長の取得 (cobfa_reclen())

```
long cobfa_reclen (
      void  /* 引数なし */
    );
```

説明

レコード形式が可変長であるファイルを扱うときに使用します。

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取得します。読込みに失敗した場合や他の入出力機能の実行後は、不定な値を返却します。

- cobfa_rdnext()
- 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.3.29 相対レコード番号の取得 (cobfa_recnum())

```
unsigned long cobfa_recnum (
                void  /* 引数なし */
              );
```

説明

相対ファイルを扱うときに使用します。

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取得します。

- cobfa_rdnext()
- cobfa_rdrec()

読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

現在位置付けられているレコードの相対レコード番号

H.3.30 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.3.31 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_rdnext ( 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.3.31 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、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.3.32 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

ー 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)   */
};
```

ー 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。

ー 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。

ー 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.6 キーパートのコード系種別

記号定数	説明
FA_UCS2KPCODE	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
FA_UTF32KPCODE	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
FA_ANYKPCODE	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。

FA_KPCODEMASKの使用例

```
#include "f4agfcfa.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 "f4agfcfa.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。

```
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-|
        |=============================================================....
        |---+---+---|   |---+---+---+---|---+---|
         1番目のパート   3番目のパート   2番目のパート
         part-1         part-3         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 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 "f4agfcfa.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.3.33 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 */
};
```

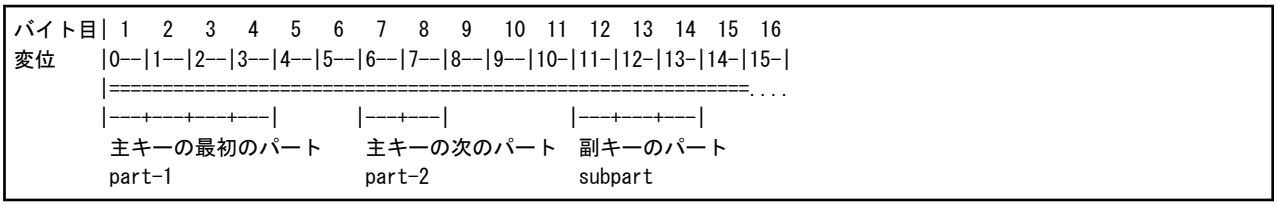
- レコードキーの総数を示すkl_nkeysには、レコードキーの総数を設定します。索引ファイルは主レコードキーを必ず含むので、レコードキーの総数は必ず1以上になります。なお、レコードキーの総数の最大値は、FA_NKEYS(126)です。
- 個々のレコードキーの情報を持つkl_keyは、struct fa_keydesc型のポインタの配列で宣言しています。struct fa_keydesc型については、“H.3.32 fa_keydesc構造体”を参照してください。

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。



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 "f4agfcfa.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.3.34 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_recsize;  /* 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_recsize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.7 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 "f4agfcfa.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.3.35 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”;
```

例

- 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()				
レコードの読込み	cobfa_rdnext()	-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_rdnext (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.3.36 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.3.26 エラー番号の取得 (cobfa_errno())”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。 ・ 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。 ・ 相対レコード番号が既存のものと重複します。
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。 ・ まだオープンしていないファイルです。 ・ 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。
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。 ・ レコード域引数がNULLポインタです。 ・ ファイル情報の取得関数で、機能番号が範囲外です。 ・ ファイル情報の取得関数で、構造体ポインタがNULLです。
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。 ・ 与えたレコードキー構成リストに矛盾があります。 ・ 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がありません。 ・ 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。
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。 ・ ファイルの内部情報が正しくないか、破壊されています。 ・ 正しいファイル編成を指定していません。 ・ 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。
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。 ・ ファイルが存在しません。 ・ ファイルにはアクセスすることができません。 ・ ファイル名がNULLポインタまたは空文字列です。 ・ ファイル名の構成が正しくありません。 ・ 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ました。
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。 ・ 順書出しで、主レコードキー値が昇順でありません。

エラー番号	説明
	・順書換えで、主レコードキー値を変更しようとしました。
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。 ・呼出し法に違反する機能の実行を要求しました。 ・このファイル編成では実行できない機能です。 ・オープン時のフラグの組合せが正しくありません。
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。 ・オープンモード、読込みモード、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。 ・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。
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。 ・レコード長がファイルの定量制限を超えています。“H.3.39 COBOLファイルアクセスルーチンの留意事項”の“ファイル機能全般”を参照してください。 ・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。
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、“ダミーファイル”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“H.3.27 入出力状態の取得 (cobfa_stat())”を参照してください。

H.3.37 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.3.27 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.3.38 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.3.39 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_recnum()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 ”
- 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。

  ― “7.3.3 行順ファイルの処理”

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_rdnext()関数 ： 順読込み
  - 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.3.30 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

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。

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。

COBOLプログラムから呼び出すサブルーチン

サブルーチン	サブルーチン名	用途
インスタンスハンドル獲得サブルーチン	JMPBWINS	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
プロセスID取得サブルーチン	COB_GET_PROCESSID	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
スレッドID取得サブルーチン	COB_GET_THREADID	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
イベントログ出力サブルーチン	COB_REPORT_EVENT	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
メモリ割当てサブルーチン(*1)	COB_ALLOC_MEMORY	動的にメモリを割り当てる
メモリを解放サブルーチン(*1)	COB_FREE_MEMORY	動的に割り当てられたメモリを解放する
プロセス終了サブルーチン	COB_EXIT_PROCESS	プロセスを強制的に終了させる
データロックサブルーチン(スレッド同期制御)(*2)	COB_LOCK_DATA	ロックキーに対するロックを獲得する
	COB_UNLOCK_DATA	ロックキーに対するロックを解放する
オブジェクトロックサブルーチン(スレッド同期制御)(*3)	COB_LOCK_OBJEC	オブジェクトに対するロックを獲得する
	COB_UNLOCK_OBJECT	オブジェクトに対するロックを解放する
デッドロック出口スケジュールサブルーチン	COB_DEADLOCK_EXIT	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する
エンディアン変換サブルーチン	#NATLETOBE	データのエンディアンを変換します。
	#NATBETOLE	

*1 : メモリ関連サブルーチンの使い方は、“I.3 メモリ関連サブルーチンの使い方”を参照してください。

*2 : データロックサブルーチンの使い方は、“I.4 データロックサブルーチンの使い方”を参照してください。

*3 : オブジェクトロックサブルーチンの使い方は、“I.5 オブジェクトロックサブルーチンの使い方”を参照してください。

他言語から呼び出すサブルーチン

サブルーチン	サブルーチン名	用途
実行単位の開始サブルーチン	JMPCINT2	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する
実行単位の終了サブルーチン	JMPCINT3	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する
実行環境の閉鎖サブルーチン	JMPCINT4	実行環境を閉鎖する場合に、他言語プログラムから使用する
COBOL実行単位ハンドル取得サブルーチン(スレッド同期制御)	COB_GETRUNIT	COBOLの実行単位を識別するハンドルを取得する

サブルーチン	サブルーチン名	用途
COBOL実行単位ハンドル設定サブルーチン (スレッド同期制御)	COB_SETRUNIT	COBOLの実行単位のハンドルを呼出し元のスレッドに設定する

I.1 COBOLプログラムから呼び出すサブルーチン

ここでは、COBOLプログラムから呼び出すサブルーチンについて説明します。

I.1.1 インスタンスハンドル獲得サブルーチン(JMPBWINS)

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" 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(18)  COMP-5.
    02  FILLER       PIC X(24).
```

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=F4AGPRCT.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、F4AGCIMP.LIBを結合してください。

I.1.2 プロセスID取得サブルーチン(COB_GET_PROCESSID)

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=F4AGEFNC.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.1.3 スレッドID取得サブルーチン(COB_GET_THREADID)

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。

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_THREADID=F4AGEFNC.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.1.4 イベントログ出力サブルーチン(COB_REPORT_EVENT)

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。

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 x64”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、Windows(x64)ではありません。
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(Windows(x64))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。
  詳細は、“17.2.1 実行時メッセージをイベントログに出力する機能”を参照してください。
- ソース名にデフォルト以外の任意の文字列を出力する場合、出力先コンピュータのレジストリに情報を設定する必要があります。本機能のためのレジストリ情報の設定および削除は“イベントログ出力サブルーチン用のレジストリキーの追加・削除”ツールを使用してください。

  なお、レジストリキー情報の設定および削除は、Administratorsグループのユーザなどレジストリキーのアクセス権(値の照会・値の設定・サブキーの作成・削除)のあるユーザが行ってください。

  NetCOBOLおよびNetCOBOL SNAPは、本製品により予約されているためソース名として使用できません。

- ソース名にはCOBOLのインタフェース上では最大256バイトまで指定できますが、システムの定量制限に依存します。
- サブルーチンを呼び出すCOBOLプログラムの翻訳時にDLOADオプションを指定した場合、以下に示すエントリ情報が必要になります。エントリ情報の指定方法については、“5.6 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=F4AGEFNC.DLL
```

I.1.5　メモリ割当てサブルーチン(COB_ALLOC_MEMORY)

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.3 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.1.6　メモリ解放サブルーチン(COB_FREE_MEMORY)

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。 ・ 指定したメモリが既に解放されている、または破壊されている。 ・ サブルーチンCOB_ALLOC_MEMORYを使用して割り当てたメモリでない。

使い方

“I.3 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.1.7　プロセス終了サブルーチン(COB_EXIT_PROCESS)

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返します。

値	意味
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、通常のアプリケーションエラーと同じように、以下の機能を使用することができます。

  － Visual C++のジャストインタイムデバッガ

  － Windowsエラー報告

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。

I.1.8　データロック獲得サブルーチン(COB_LOCK_DATA)

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"
                          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 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.4 データロックサブルーチンの使い方”を参照してください。

I.1.9　データロック解放サブルーチン(COB_UNLOCK_DATA)

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

```
PROCEDURE DIVISION.
    CALL   "COB_UNLOCK_DATA"
                          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 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.4 データロックサブルーチンの使い方”を参照してください。

I.1.10　オブジェクトロック獲得サブルーチン(COB_LOCK_OBJECT)

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"
                              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 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.5 オブジェクトロックサブルーチンの使い方”を参照してください。

I.1.11　オブジェクトロック解放サブルーチン(COB_UNLOCK_OBJECT)

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"
                              USING  BY REFERENCE OBJ
                                     BY REFERENCE ERR-DETAIL
                              RETURNING  RET-VALUE.
```

パラメタ

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 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.5 オブジェクトロックサブルーチンの使い方”を参照してください。

I.1.12　デッドロック出口スケジュールサブルーチン(COB_DEADLOCK_EXIT)

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.1.13　ビッグエンディアン変換サブルーチン(#NATLETOBE)

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.1.14　リトルエンディアン変換サブルーチン(#NATBETOLE)

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.2 他言語連携で使用するサブルーチン

ここでは、他言語連携用にCOBOLが提供するサブルーチンについて説明します。

I.2.1 実行単位の開始サブルーチン(JMPCINT2)

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、“10.1.2 COBOLの言語間の環境”および“18.3.1 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.2.2　実行単位の終了サブルーチン(JMPCINT3)

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、“10.1.2 COBOLの言語間の環境”および“18.3.1 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.2.3　実行環境の閉鎖サブルーチン(JMPCINT4)

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してください。

```
extern "C" void JMPCINT4(void);
```

パレメタの説明

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

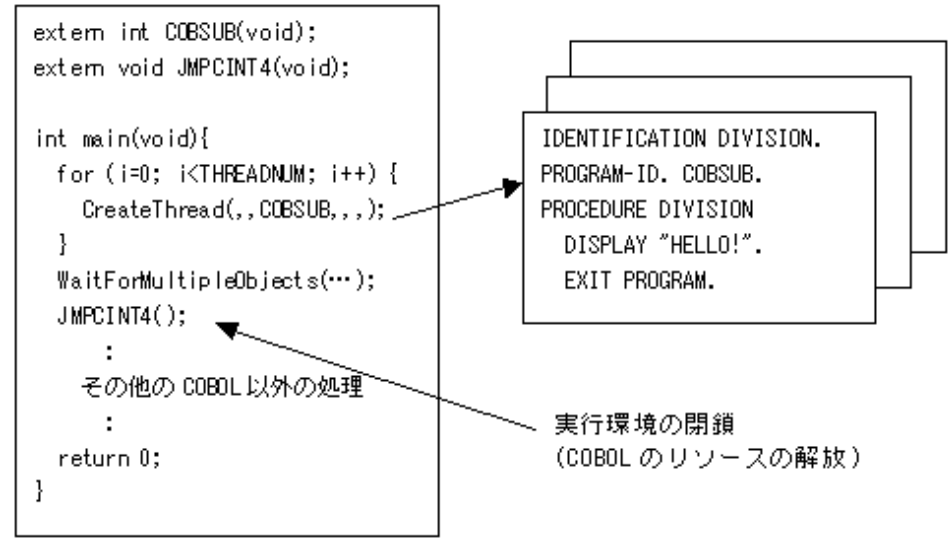
当サブルーチンを呼び出す前に、プロセスのすべてのスレッドでCOBOLプログラムの実行が終了している必要があります。COBOLプログラムの実行中に当サブルーチンが呼び出されると、実行中のCOBOLプログラムは異常終了します。

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、“10.1.2 COBOLの言語間の環境”および“18.3.1 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.2.4 COBOL実行単位ハンドル取得サブルーチン(COB_GETRUNIT)

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、“18.6.4 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 long COB_GETRUNIT(void);
```

```
データ宣言部：
  unsigned long 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.2.5　COBOL実行単位ハンドル設定サブルーチン(COB_SETRUNIT)

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、“18.6.4 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)

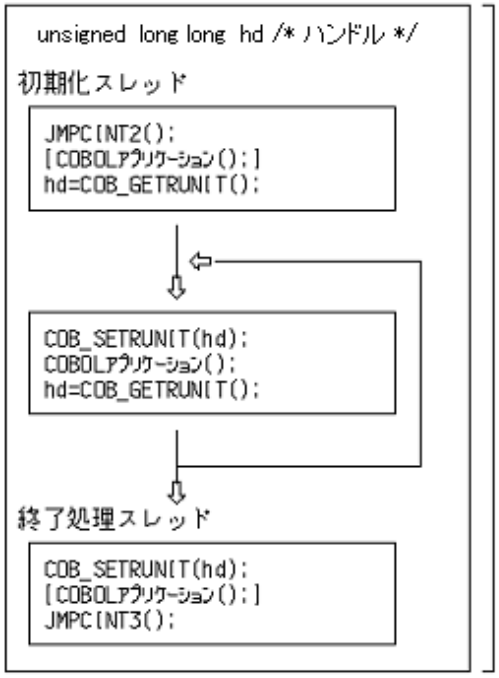
```
型宣言部：
  extern int COB_SETRUNIT(unsigned long long hd );
データ宣言部：
  extern unsigned long long hd;
                /* COBOLの実行単位ハンドル格納域(COB_GETRUNITで取得) */
手続き部：
  COB_SETRUNIT(hd);
```

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　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、“18.2.2 マルチスレッドプログラムとは”を参照してください。

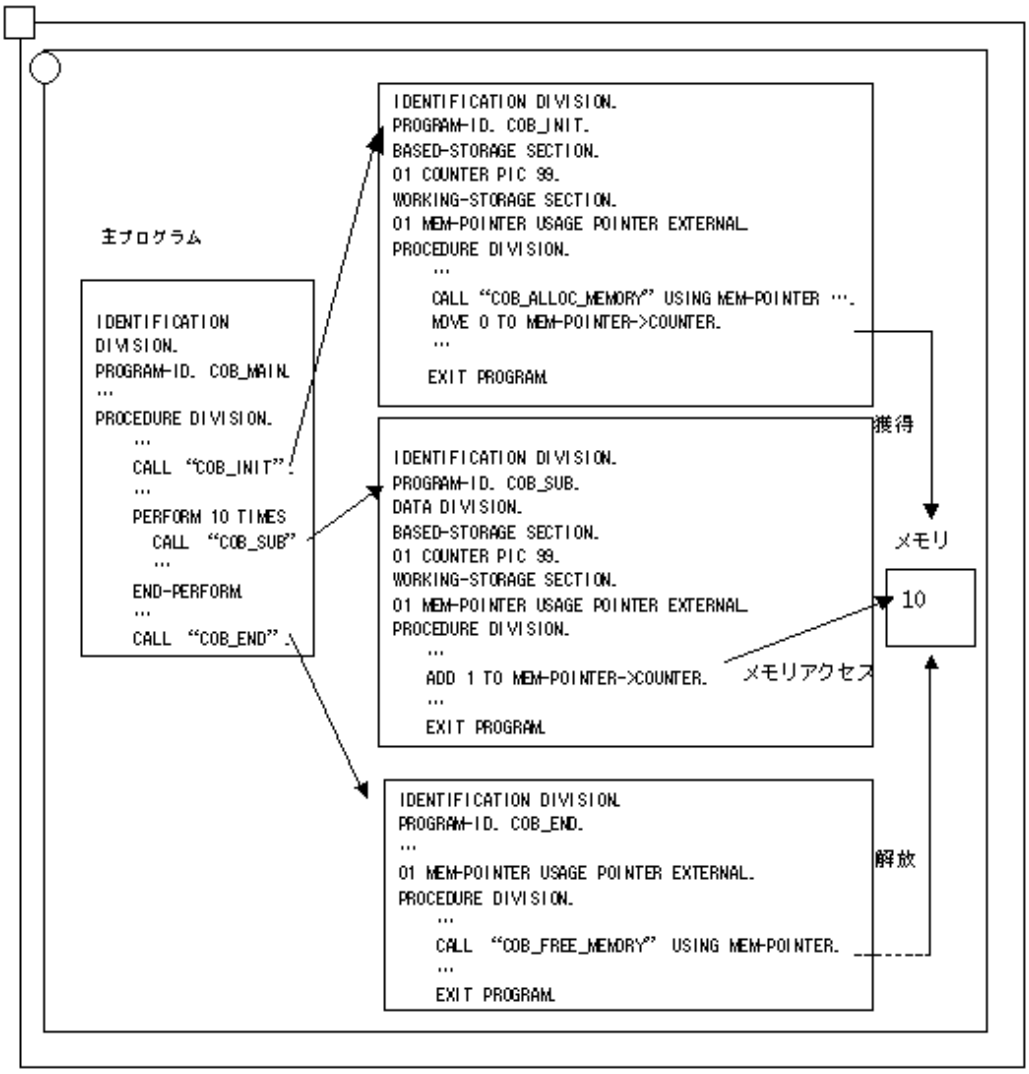
I.3.1　シングルスレッドモードのプログラム

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）



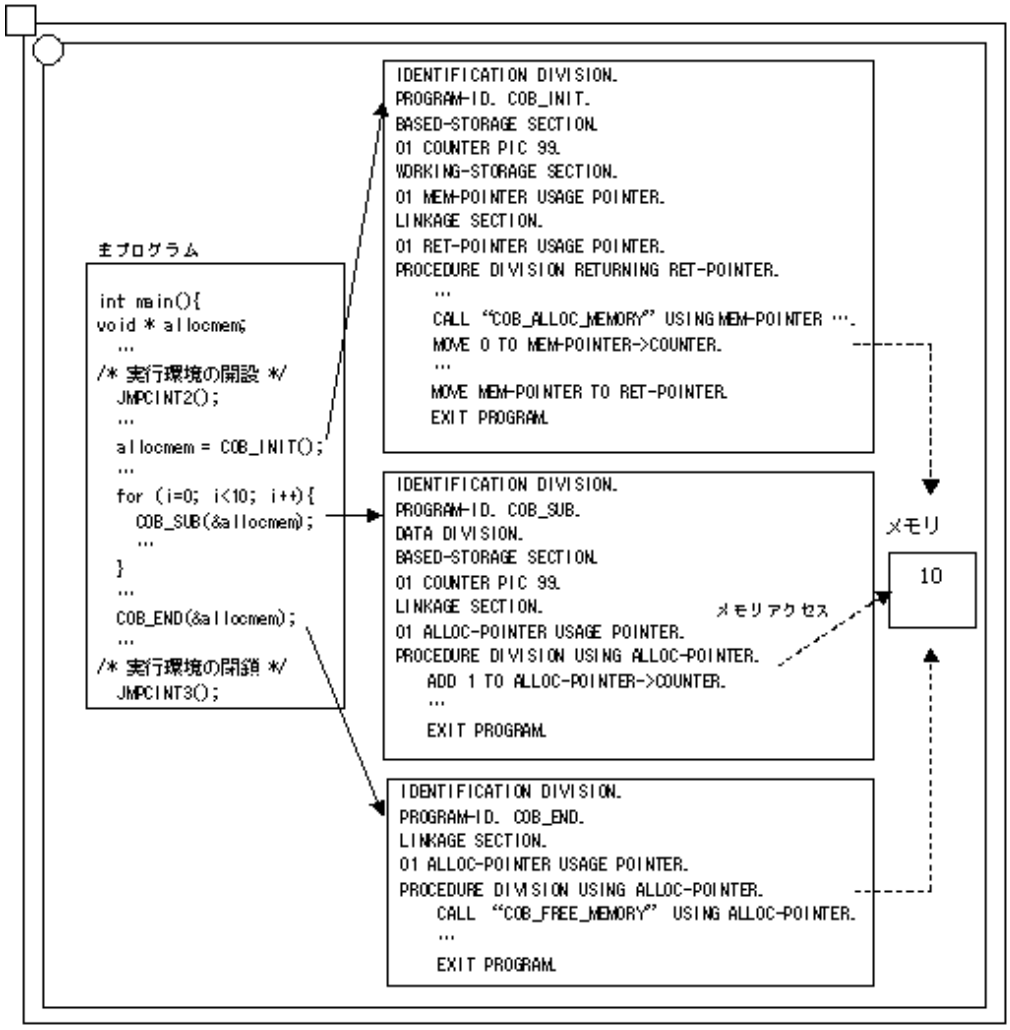
主プログラムが他言語の場合

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.2 他言語連携で使用するサブルーチン”を参照してください。

図I.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）



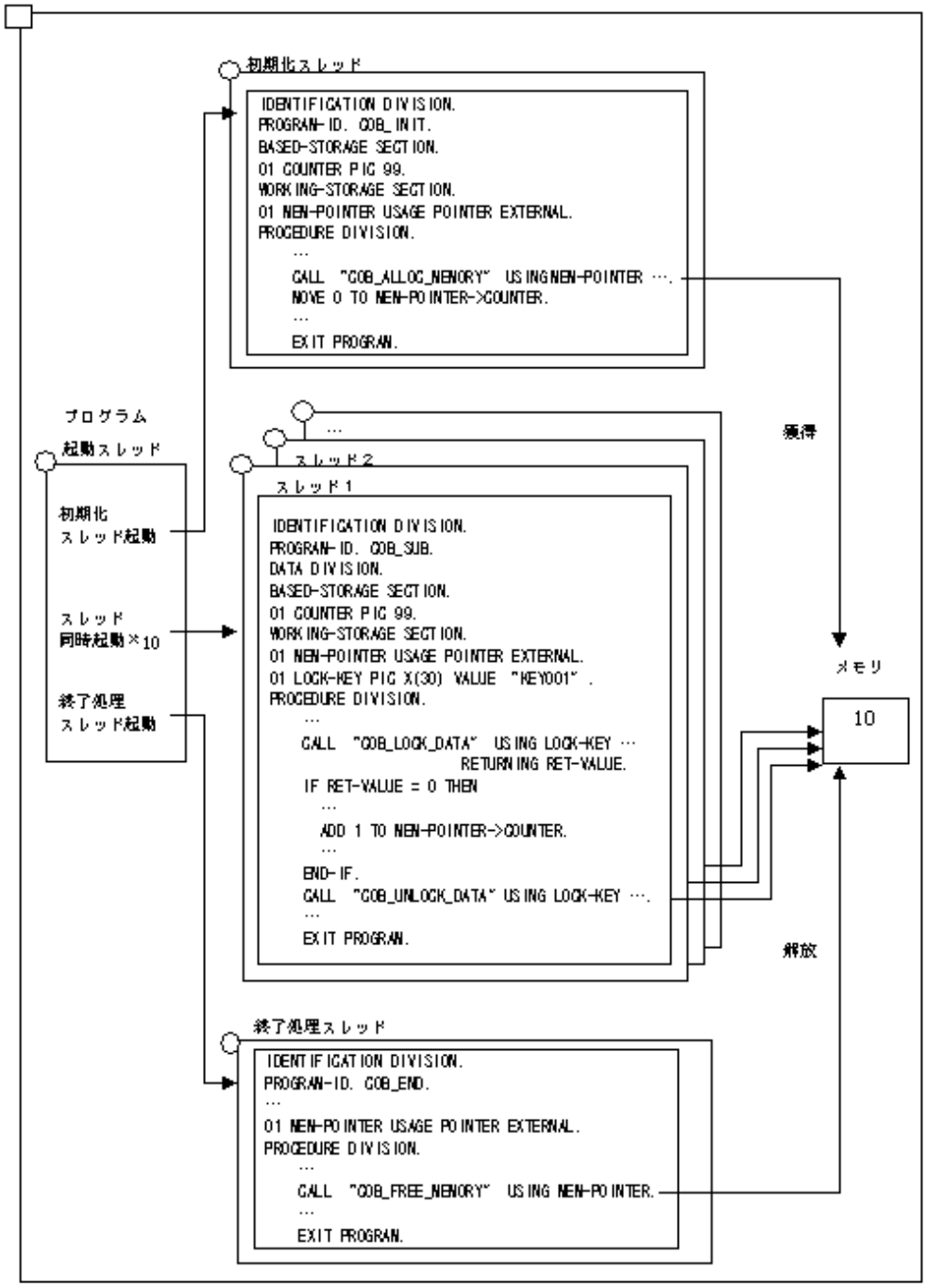
I.3.2 マルチスレッドモードのプログラム

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、“18.4 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)

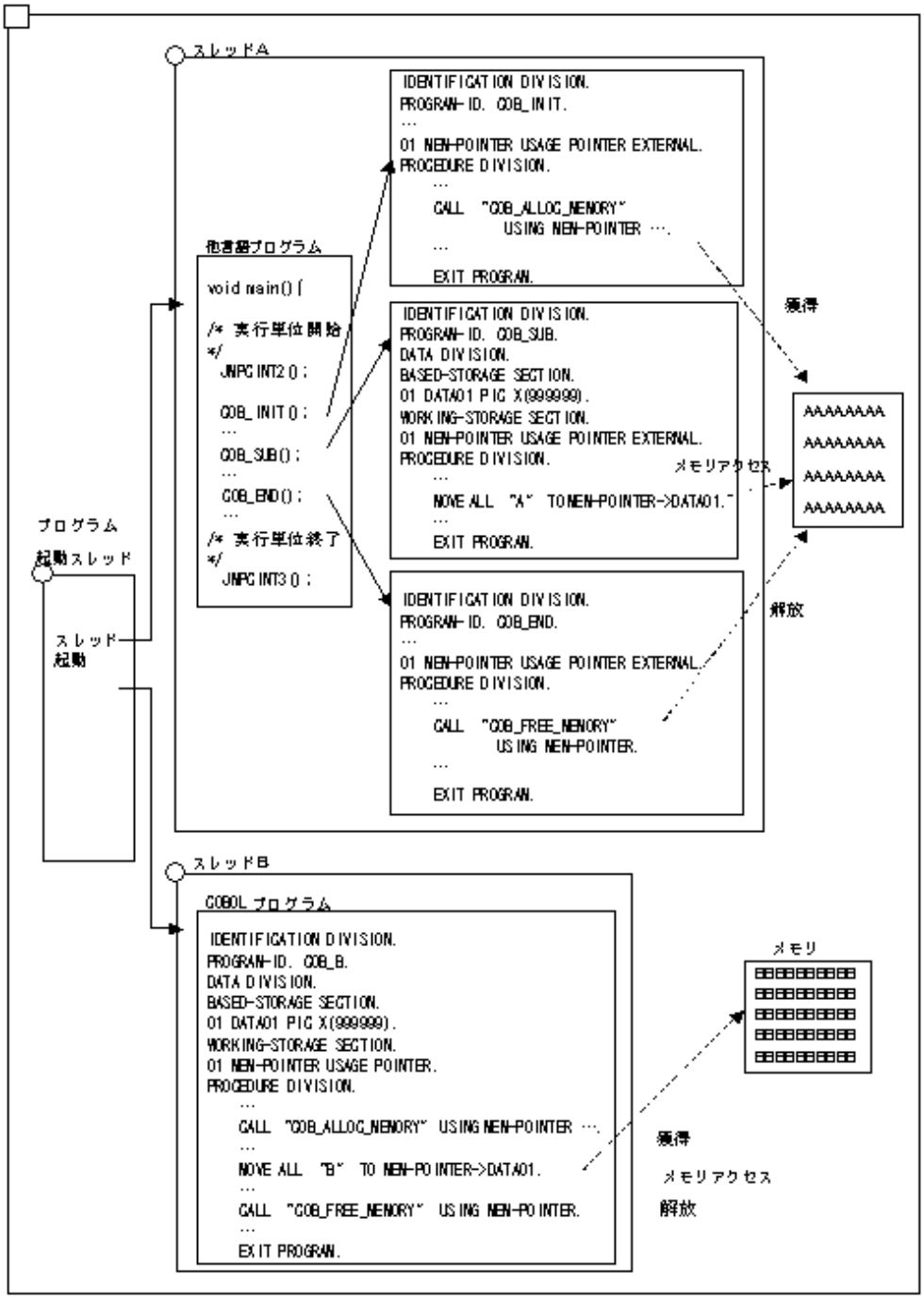


スレッド指定

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

サブルーチンを呼び出すCOBOLプログラムの翻訳時にDLOADオプションを指定した場合、以下のようなエントリ情報が必要になります。エントリ情報の指定方法は、“5.6 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=F4AGEFNC.DLL
```

I.4　データロックサブルーチンの使い方

サブルーチン名	機能
COB_LOCK_DATA	ロックキーに対するロックの獲得
COB_UNLOCK_DATA	ロックキーに対するロックの解放

同一プロセス内のスレッド間で同期制御が必要となる場合に、当サブルーチンを使用して、互いに同じデータ名のロックキーに対してロックの獲得と解放を行うことで相互排他ロックが可能となります。このときに指定するデータ名は、プロセス内で一意にする必要があります。

COB_LOCK_DATAの呼出し時に、パラメタで指定されたデータ名に対応するロックキーが作成され、ロックを獲得します。指定されたデータ名に対応するロックキーが既に存在する場合は、そのロックキーに対してロックを獲得します。

ロックを獲得できるスレッドは1つだけであり、ロックを獲得したスレッドだけが実行されます。同じロックキーに対してロックを獲得しようとしたほかのスレッドは、ロックを獲得しているスレッドがロックを解放するまで待つことになります。

ロックは、COB_UNLOCK_DATAを呼び出すことによって解放されます。

動的プログラム構造による呼出し

動的プログラム構造で当サブルーチンを呼び出す場合、以下のようなエントリ情報が必要になります。エントリ情報の指定方法については、“5.6 エントリ情報”を参照してください。

```
[ENTRY]
  COB_LOCK_DATA=F4AGEFNC.DLL
  COB_UNLOCK_DATA=F4AGEFNC.DLL
```

I.5　オブジェクトロックサブルーチンの使い方

サブルーチン名	機能
COB_LOCK_OBJECT	オブジェクトに対するロックの獲得
COB_UNLOCK_OBJECT	オブジェクトに対するロックの解放

同一プロセス内のスレッド間でオブジェクトを共有している場合に、当サブルーチンを使用して、同一のオブジェクトに対してロックの獲得と解放を行うことで、相互排他ロックが可能となります。

COB_LOCK_OBJECTの呼び出し時に、オブジェクトを指定することにより、そのオブジェクトに対してロックを獲得します。

ロックを獲得できるスレッドは1つだけであり、ロックを獲得したスレッドだけがオブジェクトを所有できます。同じオブジェクトに対してロックを獲得しようとしたほかのスレッドは、ロックを獲得しているスレッドがロックを解放するまで待つことになります。

ロックは、COB_UNLOCK_OBJECTを呼び出すことによって解放されます。

動的プログラム構造による呼出し

動的プログラム構造で当サブルーチンを呼び出す場合、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=F4AGEFNC.DLL
  COB_UNLOCK_OBJECT=F4AGEFNC.DLL
```

I.6　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 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。 ・ロックキーの名前の指定誤り（COB_LOCK_DATA、COB_UNLOCK_DATA） ・待ち時間の指定誤り（COB_LOCK_DATA、COB_LOCK_OBJECT） ・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）	○	○	○	○
-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ました。	×	○	×	○
-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

付録J　コマンド

ここでは、NetCOBOLで使用するコマンドを説明します。

コマンド		用途
cobolコマンド		COBOLソースプログラムを翻訳して、目的プログラムを生成します。
LINKコマンド		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。
COBOLファイルユーティリティコマンド	cobfconvコマンド （ファイル変換コマンド）	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への変換、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への変換を行います。
	cobfloadコマンド （ファイルロードコマンド）	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。
	cobfulodコマンド （ファイルアンロードコマンド）	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。
	cobfbrwsコマンド （ファイル表示コマンド）	任意のCOBOLファイルのレコードを1件ずつ表示します。
	cobfsortコマンド （ファイル整列コマンド）	任意のCOBOLファイルのレコードを整列して、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。
	cobfattrコマンド （ファイル属性コマンド）	索引ファイルの属性情報(レコード長など)を表示します。
	cobfrcovコマンド （ファイル復旧コマンド）	アクセスできなくなった索引ファイルを復旧します。
	cobfreogコマンド （ファイル再編成コマンド）	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。
INSDBINFコマンド		プリコンパイル後のソースファイルを入力して、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。この中間ファイルをCOBOLコンパイラの入力にすると、オリジナルソースに対してエラージャンプやデバッグができるようにます。
cobmkmfコマンド		カレントフォルダにあるソースファイル名を収集し、MAKEファイルを自動生成します。
CNVMED2UTF32コマンド （UTF-32用定義体変換コマンド）		FORMまたはPowerFORMで作成された帳票定義体(.smd/.pmd)を帳票データの日本語項目をUTF-32で扱う帳票定義体に変換します。

J.1　COBOLコマンド

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

入力形式

コマンド	オペランド
COBOL	［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

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

以下は、COBOLコマンドによる翻訳操作およびその出力例です。

通常の出力形式

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

NetCOBOLコマンドプロンプト

```
c:¥COBOL>cobol -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。

COBOLコマンドで翻訳を中断する場合、通常のDOSコマンドと同様の方法で行います。

J.1.1　COBOLコマンドの復帰コード

COBOL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

最大重大度コード	復帰コード
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.1.9 -dr（リポジトリファイルの入出力先フォルダの指定）”
- “J.1.12 -I（登録集ファイルのフォルダの指定）”
- “J.1.14 -m（画面帳票定義体ファイルのフォルダの指定）”
- “J.1.17 -R（リポジトリファイルの入力先フォルダの指定）”

翻訳リストに関するもの

- “J.1.7 -dp（翻訳リストファイルのフォルダの指定）”
- “J.1.16 -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.1.10 -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.1.6 -do（オブジェクトファイルのフォルダの指定）”
- “J.1.13 -M（主プログラムを翻訳するときの指定）”
- “J.1.15 -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.1.3 -Dc（COUNT機能を使用する指定）”
- “J.1.4 -dd（デバッグ情報ファイルのフォルダの指定）”
- “J.1.5 -Dk（CHECK機能を使用する指定）”
- “J.1.8 -Dr（TRACE機能を使用する指定）”
- “J.1.11 -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.1.18 -v（各種情報を出力する指定）”
- “J.1.19 -WC（翻訳オプションの指定）”

J.1.3　-Dc（COUNT機能を使用する指定）

```
-Dc
```

COUNT機能を使用する場合、-Dcオプションを指定します。

COUNT機能については、“19.4 COUNT機能”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.3.8 COUNT（COUNT機能の使用の可否）”

J.1.4　-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。

参照

“J.1.11 -Dt（デバッグ機能および診断機能を使用する指定）”

“A.3.52 TEST（デバッグ機能および診断機能の使用の可否）”

J.1.5　-Dk（CHECK機能を使用する指定）

```
-Dk
```

CHECK機能を使用する場合、-Dkオプションを指定します。

CHECK機能については、“19.2 CHECK機能”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.1.6　-do（オブジェクトファイルのフォルダの指定）

```
-doフォルダ
```

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

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

参照

“A.3.32 OBJECT（目的プログラムの出力の可否）”

J.1.7　-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オプションを指定した場合だけ意味を持ちます。

-dpオプションを指定した場合、翻訳リストファイルは以下のようになります。

-Pオプションにファイル名の指定がある場合

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

例

```
cobol -Pout.LST -dpd:¥test cobtest.cob
→  d:¥test¥out.LST
```

-Pオプションにファイル名の指定がない場合

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

例

```
cobol -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オプション出力規則に従って出力されます。

参照

“J.1.16 -P（各種翻訳リストの出力および出力先の指定）”

J.1.8　-Dr（TRACE機能を使用する指定）

```
-Dr
```

TRACE機能を使用する場合、-Drオプションを指定します。

TRACE機能については、“19.3 TRACE機能”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.3.54 TRACE(TRACE機能の使用の可否)”

J.1.9　-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、リポジトリファイルは、ソースファイルと同じフォルダに作成されます。

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。

J.1.10　-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。

参照

“A.3.40 SAI(ソース解析情報ファイルの出力の可否)”

J.1.11　-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、NetCOBOL Studioのデバッグ機能や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。

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、“20.1 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.1.4 -dd(デバッグ情報ファイルのフォルダの指定)”

“A.3.52 TEST（デバッグ機能および診断機能の使用の可否）”

J.1.12　-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。

参考

- -Iオプションは、環境変数COB_COBCOPYと同じ意味です。
- オプションの指定が重複した場合の検索順序は、以下の通りです。
  1. -Iオプション
  2. 環境変数COB_COBCOPY
  3. カレントフォルダ

J.1.13　-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.3.24 MAIN（主プログラム/副プログラムの指定）”

J.1.14　-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ファイル名および格納するフォルダ名に、Unicode固有文字を使用することはできません。
- 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

オプションの優先順位について、オプションの指定が重複した場合の検索順序は、以下の通りです。

1. -mオプション
2. カレントフォルダ

J.1.15　-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.3.33 OPTIMIZE（広域最適化の扱い）”

J.1.16　-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。

  ×：　COBOL -P ソースファイル名
  ○：　COBOL -P -WC,”OBJECT”　ソースファイル名

J.1.17　-R（リポジトリファイルの入力先フォルダの指定）

```
-Rフォルダ
```

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

参考

- -Rオプションは、環境変数COB_REPINと同じ意味です。

・ オプションの優先順位について、オプションの指定が重複した場合の入力先フォルダとしての検索順序は、以下の通りです。

1. -Rオプション
2. 環境変数COB_REPIN
3. -drオプション
4. カレントフォルダ

J.1.18 -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。

・ COBOLコマンドのバージョン情報

参考

-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、COBOL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、COBOLコマンドの復帰コードは0です。[参照]“J.1.1 COBOLコマンドの復帰コード”

J.1.19 -WC（翻訳オプションの指定）

```
-WC,"翻訳オプション"
```

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

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

翻訳オプションの指定方法によるオプションの優先順位は、“A.1 翻訳オプションの指定方法と優先順位”の“翻訳オプションの指定方法と優先順位”を参照してください。

ここでは、COBOLで提供している翻訳コマンド、リンクコマンド以外のコマンドについて説明します。

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

また、以下のファイルは、実行可能プログラムを作成する場合と、C関数を呼び出すDLLを作成する場合に指定してください。

- LIBCMT.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)	デバッグ情報を生成する場合に指定します。デバッグ情報はプログラムデータベース(PDB)に作成されます。
/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。
/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)。
/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。

注1 ： プログラムデータベース(PDB)を使用したデバッグ方法の詳細は、“20.1 診断機能”および“NetCOBOL Studio ユーザーズガイド”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”

J.2.2 /DUMPオプション

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、LINKコマンドの/DUMPオプションが使用できます。

ここでは、LINKコマンドの/DUMPオプションの形式について説明します。

入力形式

コマンド	オペランド
LINK	/DUMP［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

LINKコマンドの/DUMPオプション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.2 LINKコマンドの/DUMPオプションのオプション

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

LINKコマンドの/DUMPオプションの使用方法の説明は、以下のようにコマンドに/DUMPオプションだけを指定すれば参照することができます。

```
LINK /DUMP
```

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

```
LINK /DUMP /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

```
LINK /DUMP /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

```
LINK /DUMP /IMPORTS /OUT:出力ファイル名 実行可能ファイル名
```

J.3 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。

注意

各コマンドの入力・出力ファイル名に、Unicode固有の文字を指定することができます。ただし、各コマンドのメッセージは、シフトJIS範囲内の文字しか出力されません。Unicode固有の文字は、“?”に置き換えられて出力されます。

J.3.1　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。

```
                  sjis
                  ebcdica
                  ebcdick
-k  { 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、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

```
              ┌ allchar                                  ┐
    { b }     │ alltxtbin                                │
-c  {   }  ,  {          { c長さ }                        }
    { t }     │ "        { t長さ }    [;...]"             │
              └          { k長さ }                        ┘
```

-kunicodeまたは-kutf32を指定した場合

```
                allucs2
              ┌ allutf8                                  ┐
    { b }     │ allutf32                                 │
-c  {   }  ,  {           u長さ                           }
    { t }     │ "        { f長さ }    [;...]"             │
              └          { t長さ }                        ┘
                           w長さ
```

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。

alltxtbin

レコード中の全データを16進形式とみなします。

"{c長さ|t長さ|k長さ}[;…]"

レコード中にデータ形式が混在する場合、キーワード文字に続けて、そのデータ形式の長さ(日本語形式の場合は文字数)を指定します。それぞれのキーワード文字の意味を以下に示します。

c ： 文字形式
t ： 16進形式
k ： 日本語形式

ただし、日本語形式は、JEFコード体系の場合しか指定できません。

allucs2

レコード中の全データをUCS-2形式とみなします。

allutf8

レコード中の全データをUTF-8形式とみなします。

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

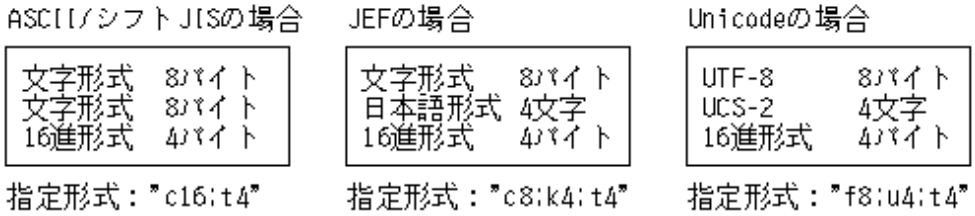
例

[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。



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 -kebcdica -ooutfile -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 -ooutfile -cb,"allucs2" infile.txt
```

J.3.2　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 { S / R / I } , { f / v } ,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   :  ixdfile
ファイル属性     :  ファイル編成       :  索引ファイル
                    レコード形式       :  可変長
                    レコード長         :  80
                    レコードキー情報   :  データ項目１（レコード内位置：0／長さ：5）
                                          データ項目２（レコード内位置：10／長さ：5）
                                          重複指定：あり
                                          副レコードキー（レコード内位置：5／長さ：5）
```

```
cobfload -oixdfile -dI,v,80,"(0,5/10,5),D;(5,5)" infile
```

- レコード順ファイルの内容を相対ファイルに追加する(拡張)

```
入力ファイル名   :  infile
出力ファイル名   :  relfile
ファイル属性     :  ファイル編成   :  相対ファイル
                    レコード形式   :  固定長
                    レコード長     :  80
```

```
cobfload -orelfile -e -dR,f,80 infile
```

注意

入力ファイルに出力ファイルの最大レコード長より大きいレコードが存在した場合、コマンド実行時にエラーとなります。

J.3.3　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 { S / R / I } , { f / v } ,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.3.4　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。

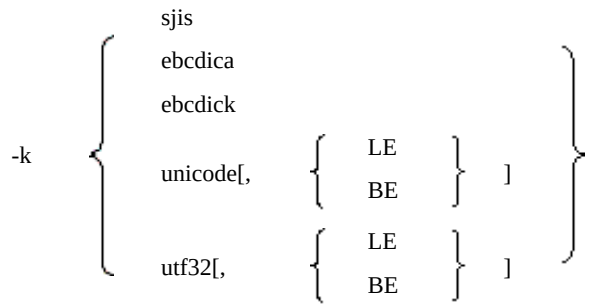
指定方法

```
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。



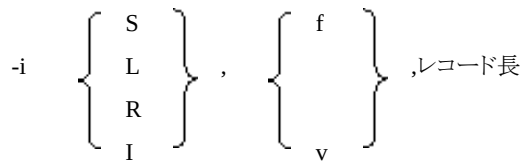
---

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。



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 { 格納順番 / r 相対レコード番号 / ic レコードキー値 / it レコードキー値 / in レコードキー値 }

ファイル編成によって指定する内容が異なります。

ー 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。

ー 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。

ー 索引ファイルでは、以下の3つのキーワード文字例に続き、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。

-pe { 格納順番 / r 相対レコード番号 / ic レコードキー値 / it レコードキー値 / in レコードキー値 / t 出力件数 }

出力件数の指定以外は、開始位置と同じです。終了位置を出力件数で指定する場合は、キーワード文字“t”に続けて出力レコード件数を指定します。

終了位置を省略した場合、ファイルの最後のレコードまで表示します。

表示順序

開始位置と終了位置の範囲に複数のレコードが存在する場合、それらのレコードの表示順序を以下の形式で指定します。

-po { A / D }

A

レコードを昇順に表示します。レコード順ファイルおよび行順ファイルの場合は、先に格納されたレコードから表示します。相対ファイルの場合は、相対レコード番号の小さいレコードから表示します。索引ファイルの場合は、レコードキー値が小さいレコードから表示します。

D

レコードを降順に表示します。相対ファイルの場合は、相対レコード番号の大きいレコードから表示します。索引ファイルの場合は、レコードキー値が大きいレコードから表示します。なお、レコード順ファイルおよび行順ファイルでは降順は指定できません。

表示順序を省略した場合、昇順を指定したとみなします。

検索キー番号

索引ファイルを表示する場合、検索キーになるレコードキーの番号を指定します。レコードキーの番号は、主レコードキーを0、最初の副レコードキーを1として、それ以降の副レコードキーを2以上の追番で表現した数字です。
検索キー番号を省略した場合、主レコードキーが指定されたとみなします。

入力ファイル名

表示するファイルのパス名を指定します。

例

指定例

- レコード順ファイルの内容を表示する

```
入力ファイル名          :  seqfile
ファイル属性            :  ファイル編成   :  レコード順ファイル
                           レコード形式   :  可変長
                           レコード長     :  80
開始位置（格納順番）    :  5
終了位置（出力件数）    :  10
```

```
cobfbrws -iS,v,80 -ps5 -pet10 seqfile
```

- 相対ファイルの内容を表示する

```
入力ファイル名                  :  relfile
文字コード                      :  JEF(EBCDIC/ASCII)
ファイル属性                    :  ファイル編成   :  相対ファイル
                                   レコード形式   :  固定長
                                   レコード長     :  50
開始位置（相対レコード番号）    :  20
終了位置（相対レコード番号）    :  10
表示順序                        :  降順
```

```
cobfbrws -kebcdica -iR,f,50 -psr20 -per10 -poD relfile
```

- 索引ファイルの内容を表示する

```
入力ファイル名                  :  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.3.5　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。

```
                 sjis
                 ebcdica
                 ebcdick
-k  { 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

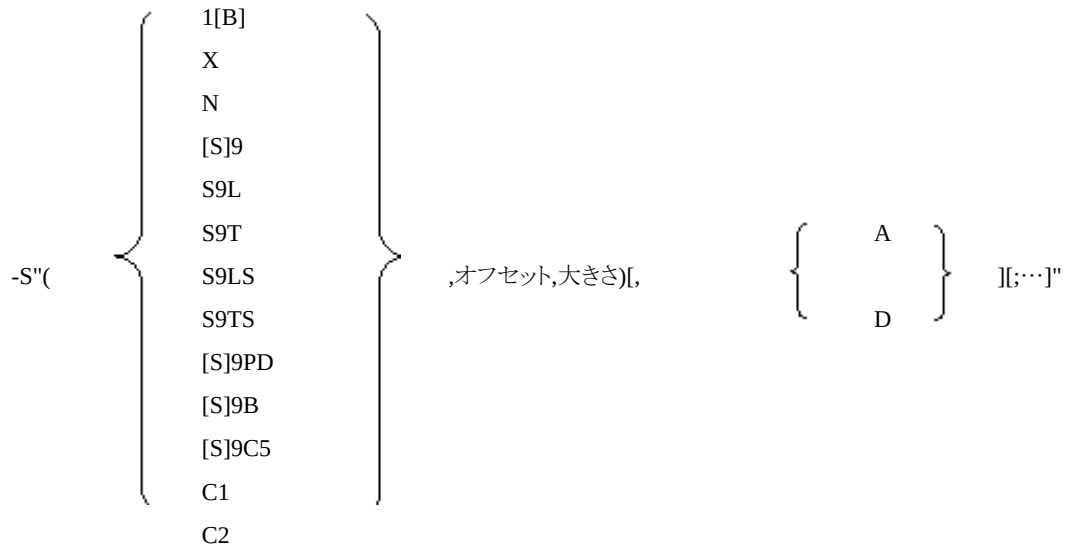
- -kオプションを省略した場合または-ksjisを指定した場合、整列するファイルの文字コードをASCII/シフトJISコード体系として扱います。
- -kebcdicaまたは-kebcdickを指定した場合、整列するファイルの文字コードをJEF(EBCDIC/KANA)コード体系またはJEF(EBCDIC/ASCII)コード体系として扱います。
- -kunicodeを指定した場合、整列するファイルの文字コードをUCS-2コード体系として扱います。
  カンマ(,)に続き、UCS-2コード体系のエンディアンを指定します。省略した場合には、LE(リトルエンディアン形式)が指定されたものとして扱います。
- -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

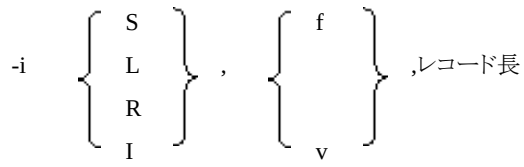
整列順序には、レコードをキーの項目属性に従って昇順に整列するか、降順に整列するかを指定します。

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。



ファイル編成

ファイル編成を以下の文字で指定します。

S ： レコード順ファイル
L ： 行順ファイル
R ： 相対ファイル
I ： 索引ファイル

レコード形式

レコード形式を以下の文字で指定します。ただし、ファイル編成が索引ファイルの場合は指定できません。

f ： 固定長
v ： 可変長

レコード長

レコード長を指定します。可変長の場合は、最大レコード長で指定します。ただし、ファイル編成が索引ファイルの場合は指定できません。

入力ファイル名

整列するファイルのパス名を指定します。

例

指定例

- 相対ファイルを整列してレコード順ファイルに出力する

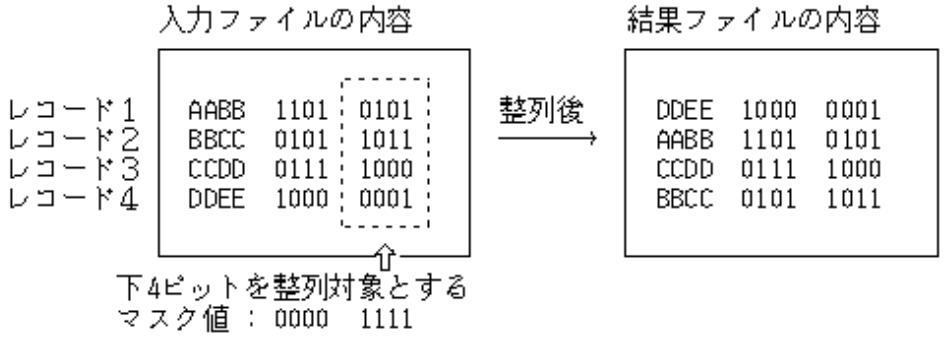
```
入力ファイル名： relfile
出力ファイル名： outfile
整列条件　　：　［第１キー］項目属性　　：日本語(N)
　　レコード内オフセット：0
　　大きさ　　：2
　　整列順序　　：昇順
　　［第２キー］項目属性　　：英数字
　　レコード内オフセット：10
　　大きさ　　：5
　　整列順序　　：降順
ファイル属性　：　ファイル編成：相対ファイル
　　レコード形式：可変長
　　レコード長　：80
```

```
cobfsort -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列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
整列条件       ： ［第１キー］項目属性              ：ブール(1B)
                  レコード内オフセット：4
                  マスク値(00001111)   ：15
                  整列順序             ：昇順
ファイル属性   ： ファイル編成：レコード順ファイル
                  レコード形式：固定長
                  レコード長   ：5
```

```
cobfsort -ooutfile -s"(1B,4,15),A" -iS,f,5  infile
```

J.3.6　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.3.7　ファイル復旧コマンド（cobfrcov）

プロセスの異常終了などで、クローズ処理が正常に行われなかった索引ファイルを、再度正常にアクセスできるように復旧します。ただし、データに異常があるために復旧できないレコードについては、それらのデータを未復旧データファイルとして可変長形式のレコード順ファイルに出力します。

指定方法

```
cobfrcov　復旧ファイル名　未復旧データファイル名
```

インタフェース

復旧ファイル名

復旧処理を行う索引ファイルのパス名を指定します。

未復旧ファイル名

復旧できないレコードのデータを出力するファイルのパス名を指定します。復旧できないデータがなければ、未復旧データファイルは作成されません。

例

指定例

- 索引ファイルを復旧する

  復旧ファイル名　：ixdfile
  未復旧ファイル名：seqfile

```
cobfrcov ixdfile seqfile
```

注意

同じ索引ファイルに対し、同時にファイル復旧コマンドを実行した場合の動作は保証されません。直前のコマンド実行の完了を確認後、実行してください。

J.3.8　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.4　INSDBINFコマンド

これまでCOBOLコンパイラとプリコンパイラ連携では、以下のような問題がありました。

- コンパイラの出力する翻訳エラー検出行番号が、中間ファイル(プリコンパイル後のソースファイル)の行番号であるため、利用者は中間ファイルを参照しながらオリジナルソースプログラム(プリコンパイル前の、埋込みSQL文が書かれたCOBOLソースプログラム)の修正を行わなければならない。
- ビルダのエラージャンプ機能で、オリジナルソースプログラムに対してエラージャンプすることができない。
- オリジナルソースプログラムを被デバッグプログラムとして、デバッガでデバッグすることができない。

これらの問題を解決するため、INSDBINFコマンドでは行番号制御情報およびファイル名制御情報を埋め込んだ中間ファイルを生成します。

行番号制御情報(#LINE情報)は、プリコンパイルを行う前のソースの行番号を、それ以降に動作するコンパイラまたはプリコンパイラに通知するための情報です。

ファイル名制御情報(#FILE情報)は、オリジナルソースプログラムのファイル名や、プリコンパイラがインクルードしたファイルのファイル名を通知するための情報です。

COBOLコンパイラは、INSDBINFコマンドが生成した中間ファイルを入力ファイルとすることにより、行番号制御情報およびファイル名制御情報を参照できます。そして、それらの情報からオリジナルソースファイルとプリコンパイル後のソースファイル行番号の対応付けたオリジナルソースプログラムおよびインクルードファイルのファイル名の取得が可能となります。これにより、COBOLコンパイラとプリコンパイラの連携時のこれまでの問題が解決されることになります。

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

指定形式	内容
	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。
```

ー 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。

ー 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。

ー 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。

ー 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、デバッガ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.5 cobmkmfコマンド

cobmkmfコマンドは、カレントフォルダにあるソースファイル名を収集し、MAKEファイルを自動生成します。作成されたMAKEファイルは、nmakeコマンドによって実行させることができ、COBOLソースプログラムの翻訳およびリンクを支援します。

入力形式

コマンド	オペランド
cobmkmf	[-w|-l] [MAIN=主プログラム名] [マクロ名=マクロ定義値]

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。

-w|-l

作成するMAKEファイルの用途を、実行可能プログラム(EXE)作成用とするか、ダイナミックリンクライブラリ(DLL)作成用とするかを選択します。

COBOLコンソールを使用する実行可能プログラム作成用とする場合、-wを指定します。

ダイナミックリンクライブラリ作成用とする場合、-lを指定します。

システムのコンソール(コマンドプロンプトウィンドウ)を使用する実行可能プログラム作成用とする場合は、指定を省略します。

MAIN=主プログラム名

実行可能プログラム作成用のMAKEファイルを生成する場合に、主プログラムとなるCOBOLソースファイル名を指定します。COBOLソースファイル名は、パス名を含まないファイル名で指定します。

マクロ名=マクロ定義値

“マクロ名=”で始まる行をMAKEファイル中から検索し、当該行を“マクロ名=マクロ定義値”に置き換えます。マクロ定義値に、空白などシステムのシェルで展開される文字列を使用する場合には、マクロ定義値をダブルクォーテーションで囲んで利用してくださ

い。また、マクロ定義値中にダブルクォーテーションを指定する場合は、ダブルクォーテーションを「¥"」(エスケープ文字指定)と記述してください。

以下のマクロ名が定義されており、置換可能です。

マクロ名	マクロ定義値
MAKFILE	MAKEファイル名
PROGRAM	作成する実行可能プログラム名またはダイナミックリンクライブラリ名
COBFLAGS	COBOL翻訳オプション
COBLDFLAGS	COBOL翻訳関連オプション
LDFLAGS	リンクオプション
LDLIBS	リンク時に結合するライブラリ(システムライブラリなど)

生成ファイル

cobmkmfコマンドが生成するファイルは以下のとおりです。

ファイル名	概要
Makefile	翻訳およびリンクを実行するためのコマンドの順序付けや実行のルールが記述されているファイルです。カレントフォルダにMakefileが存在している場合は、“Makefile.bak”というファイル名でバックアップファイルを作成します。
SRCLIST	依存関係にあるソースファイル名が記述されているファイルです。カレントフォルダにSRCLISTが存在している場合は、“SRCLIST.bak”というファイル名でバックアップファイルを作成します。

cobmkmfコマンドは、COBOLソースファイル名が以下の規則に従って命名されていることを前提にMAKEファイルの作成を行います。

- プログラム定義のソースファイルの場合
  プログラム名.cob
- クラス定義のソースファイルの場合
  クラス名.cob
- 分離されたメソッド定義のソースファイルの場合
  メソッド名.cob

また、ソースファイルの種類を以下に示す拡張子により認識します。認識したソースファイル名のリストを、SRCLISTファイルに出力します。そのため、事前に以下の規則に従って拡張子を変更しておく必要があります。

拡張子	ソースファイルの種類
.COB、.COBOL	COBOLソースプログラム
.CBL	登録集原文
.SMD、.PMD、.SMU、.PMU	画面帳票定義体
.REP	クラス情報
.LIB	ライブラリおよびインポートライブラリ

MAKEのターゲット

作成されたMAKEファイルでは、標準で以下のターゲットを利用できます。

ターゲット	機能
all	実行可能プログラムまたはダイナミックリンクライブラリを、翻訳およびリンクして作成します。
clean	すべてのオブジェクト、デバッグ情報等を削除します。
rebuild	nmake clean実行後、nmake allを実行します。

ターゲット	機能
srclist	カレントフォルダを検索し、SRCLISTを更新します。
depend	SRCLISTを元に依存関係を更新します。
opt	COBOLの最適化機能を使ってnmake allを実施します。
debug	COBOLのデバッグ機能を利用できるようにnmake allを実施します。
thread	COBOLのマルチスレッド機能を使ってnmake allを実施します。
trace	COBOLのトレース機能を利用できるようにnmake allを実施します。
check	COBOLのチェック機能を利用できるようにnmake allを実施します。
count	COBOLのカウント機能を利用できるようにnmake allを実施します。
list	COBOLのリスト出力機能を使ってnmake allを実施します。

cobmkmfの実施後に、ソースファイルの追加や依存関係に関わる修正を行った場合には、以下のいずれかの対処を行って、MAKEファイルを更新してください。

- cobmkmfを再度実施する。
- SRCLISTの再作成(nmake srclist)またはエディタで修正後、依存関係の再定義(nmake depend)を実施する。

例

cobmkmfコマンドの使用例

- COBOLコンソールを使用するMAIN01.EXEを作成するMAKEファイルを、COBOLの翻訳オプション-WC,"ALPHAL(WORD)"で作成したい場合(主プログラムのCOBOLソースファイル名をMAIN01.COBとします)

```
cobmkmf -w PROGRAM=MAIN01.EXE MAIN=MAIN01.COB "COBFLAGS=-WC,¥"ALPHAL(WORD)¥""
```

- システムのコンソールを使用するMAIN02.EXEを作成するMAKEファイルを、COBOLの翻訳オプション-WC,"ALPHAL(WORD)"で作成したい場合(主プログラムのCOBOLソースファイル名をMAIN02.COBとします)

```
cobmkmf PROGRAM=MAIN02.EXE MAIN=MAIN02.COB "COBFLAGS=-WC,¥"ALPHAL(WORD)¥""
```

- SUBLIB01.DLLを作成するMAKEファイルを、COBOLの翻訳オプション-WC,"ALPHAL(WORD)"で作成したい場合

```
cobmkmf PROGRAM=SUBLIB01.DLL "COBFLAGS=-WC,¥"ALPHAL(WORD)¥""
```

- 作成したMAKEファイルを使って、デバッグ形式のライブラリを作成する場合

```
nmake debug
```

- 作成したMAKEファイルを使って、スレッド形式ライブラリとして再構築する場合

```
nmake clean thread
```

注意事項

- cobmkmfは、カレントフォルダに1つの実行可能プログラムまたはダイナミックリンクライブラリを作成するために必要な資源だけが存在している事を前提にMAKEファイルを作成します。そのため、事前に不要なファイルの削除および必要なファイルをカレントフォルダにコピーしておく等の準備が必要です。
- cobmkmfでは、厳密な構文解析を実施していないため、書き方(登録集に依存関係を表わす語がある場合や構文エラーがある場合等)によっては、誤ったMAKEファイルを作成することがあります。また、SRCLISTもcobmkmfコマンドの実行時のカレントフォルダの状況に応じて作成されるものであり、修正が必要な場合があります。必ず、任意のテキストエディタで確認及び修正を実施してご利用ください。

・ 依存関係の調査では、COBOLソースファイル中から以下の構文の一部を認識して依存関係を調査します。

  ― プログラム名段落/クラス名段落/メソッド名段落

  ― プログラム終り見出し/クラス終り見出し/メソッド終り見出し

  ― リポジトリ段落のクラス指定子

  ― COPY文

  ― CALL文

  次のような場合には、依存関係が正しく検索されない場合がありますので、依存関係の正当性を確認し、適切な対処を実施してください。

  ― 1つのソースファイルに複数の翻訳単位が存在している場合

  ― ソース中で定義したプログラム名、クラス名と実際のファイル名が異なる場合

  ― 原始文操作機能により、上記の構文を構成する語が置き換わる場合

  ― 上記の構文を構成する語を継続行にまたがって記述した場合

  ― 上記の構文の途中から始まるCOPY文を含む場合

  ― 内部プログラムとそれを呼び出すCALL文が同一ファイル中に存在しない場合(COPY文を使用している)

・ cobmkmfは、内部でnmake dependを実施することで、依存関係定義を作成します。その際、依存関係に関連するエラーが存在した場合、警告メッセージが表示されます。

・ cobmkmfが作成したMAKEファイル中に再帰的な依存関係定義(相互参照や間接参照等)を含む場合、ソースを修正していなくても、nmakeコマンド実行時に、メイク順序の関係から一部のソースの翻訳およびリンクを実行する場合があります。

・ cobmkmfでは、1つの実行可能プログラムまたはダイナミックリンクライブラリを作成するMAKEファイルを作成することを前提としており、ライブラリ間で相互参照や間接参照となるMAKEファイルは自動生成できません。

・ nmakeコマンドは、翻訳およびリンクを実行するための環境変数が設定されている環境下で実行してください。

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。

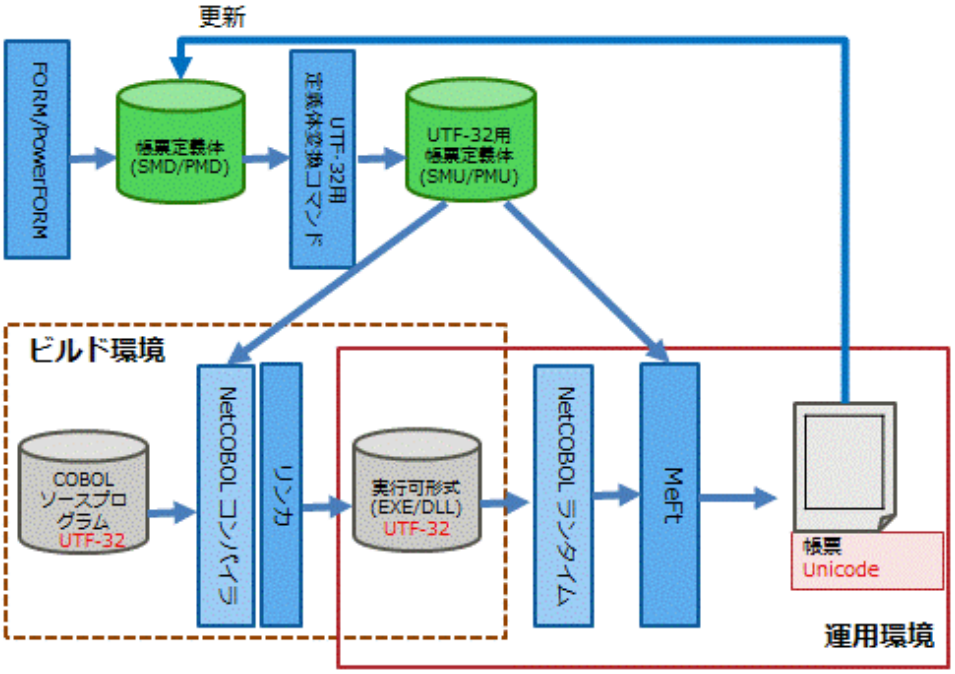
FORMまたはPowerFORMを使用して作成した帳票定義体は、日本語項目を1文字=2バイトで扱います。COBOLプログラムで日本語項目のエンコードをUTF-32とした場合は、1文字=4バイトで扱わなければプログラムと整合しません。UTF-32用定義体変換コマンドでは、帳票定義体に定義された日本語項目のデータ長の扱いをUTF-32用に変換した帳票定義体を出力します。

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。

復帰値	メッセージ	対処方法
-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。
-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			比較	備考
分類		機能概要		
文字集合		プログラム中で使用可能な文字の種類すべて	○	
コード系		Unicode	△	
		システム依存	○	
COBOL 語	利用者語	利用者語の種類すべて	○	ただし、__(アンダースコア)の使用は、本システム固有の機能です。使用可能な日本語文字は、各システムのコード系に従います。
	表意定数	プログラム中で使用可能な表意定数すべて	○	
	特殊レジスタ	SHIFT-IN 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(18)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
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		×	
		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)
		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	×	
表示ファイル	環境定義	SYMBOLIC DESTINATION句の"CMD"、"TRM"、"WST"、"DSP"、"APL"、"ACM" 指定	×	
		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	すべて		○	
	特殊レジスタ	すべて		○	
		フォームオーバレイ		□	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		×	
	AIM/RDB	すべて		×	
	SD機能	すべて		○	
浮動小数点		すべて		■	OSIV系システムと内部表現が異なるため、演算結果が異なる場合があります。
組込み関数機能		CUURENT-DATE関数		○	
		上記以外		△	
スクリーン操作機能		すべて		△	
コマンド行引数／環境変数操作機能		すべて		△	
オブジェクト指向機能		すべて		△	
翻訳オプションカスタマイズ機能		すべて		×	
ユーザテーラリング機能		すべて		×	

注：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

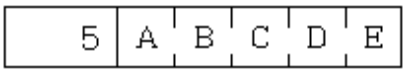
COBOLプログラムの記述

```
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



- 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。

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　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。

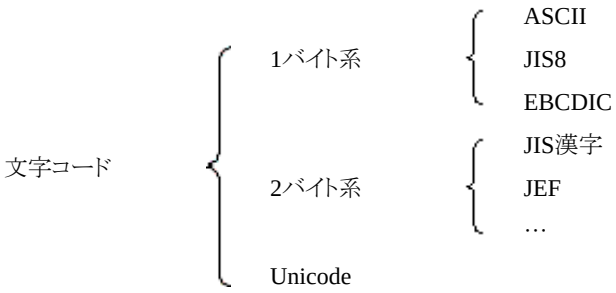
L.1　文字コードの概説

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

L.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。

L.1.2 文字種の混在方式による分類

1バイトで表現される英数字等の文字と2 バイトで表現される文字を混在して使用する方法は文字種の判定の方法と使用可能な文字種によってさまざまな変種が存在します。

ここでは、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。

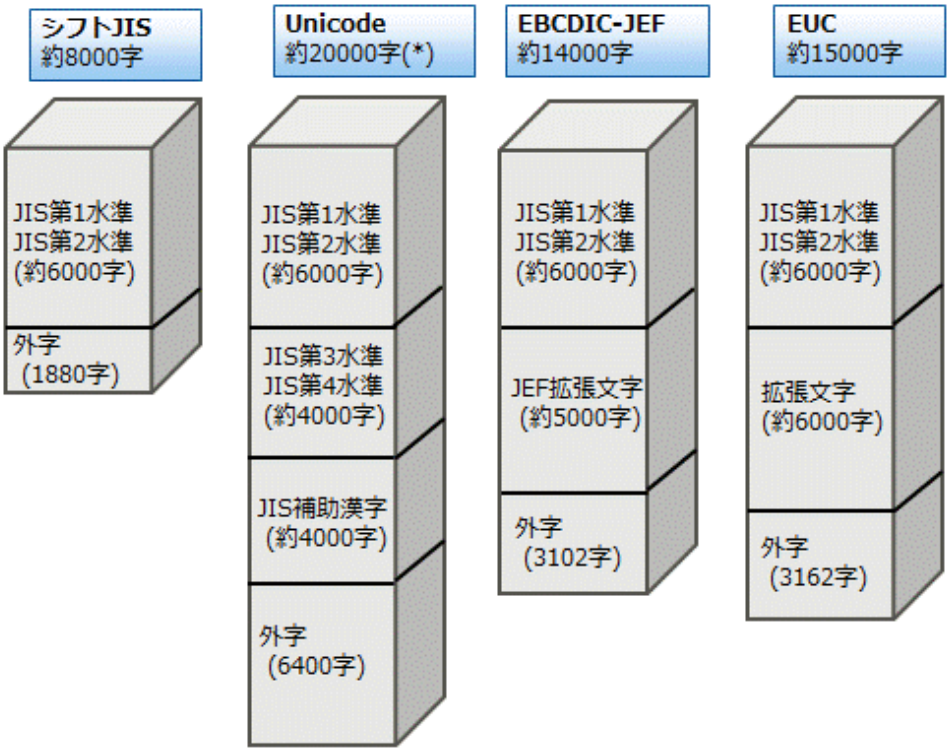
EBCIDIC-JEF

富士通のOSIVシリーズで使用されるコード系です。英数字・カナ文字(EBCDIC)、日本語文字(JEF)を混在させる方法で、次のような特徴を持ちます。

- 文字種の切り換えにシフトコードを使用します。
- 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。



L.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。

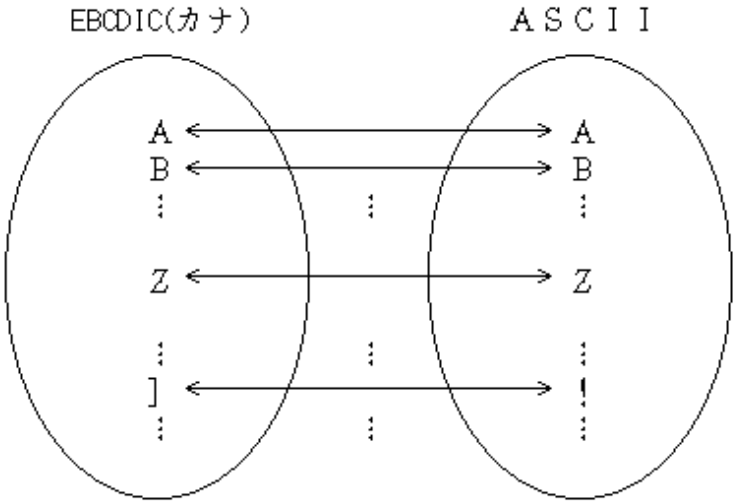
変換の前後で文字が一致しない変換

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。

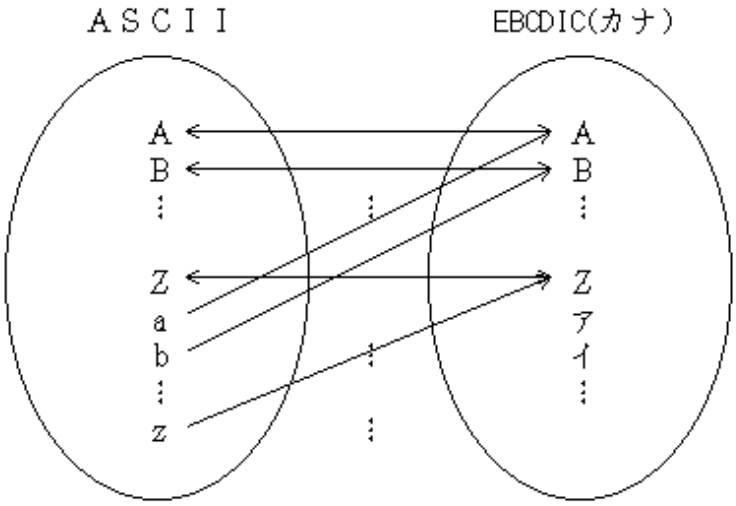
代替文字による変換

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。



L.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。

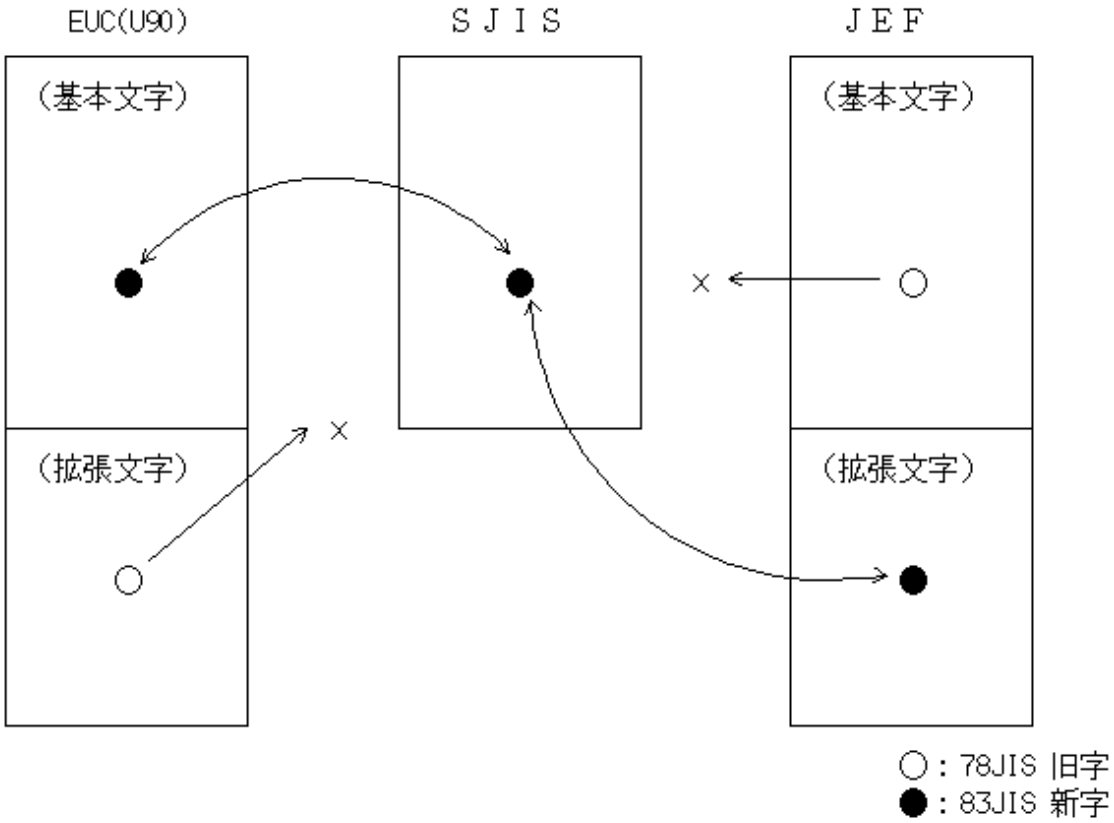
EBCDICからシフトJISへの変換時の代替文字による変換

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	“［”	“￠”	“￠”	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

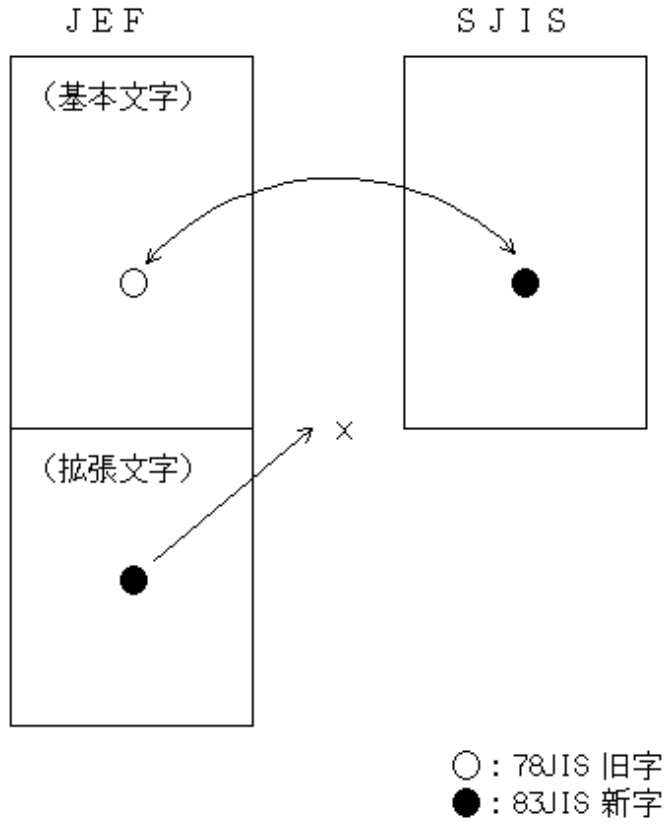
JEFおよびEUC(U90)は78JISと83JISで非互換のある文字についてそれぞれの字形毎に別のコードが割り当てられていますが、シフトJISでは1つのコードしか割り当てられていません。このため、一般的にはJEFおよびEUC(U90)に含まれていた78JISの文字の情報は失われてしまいます。



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

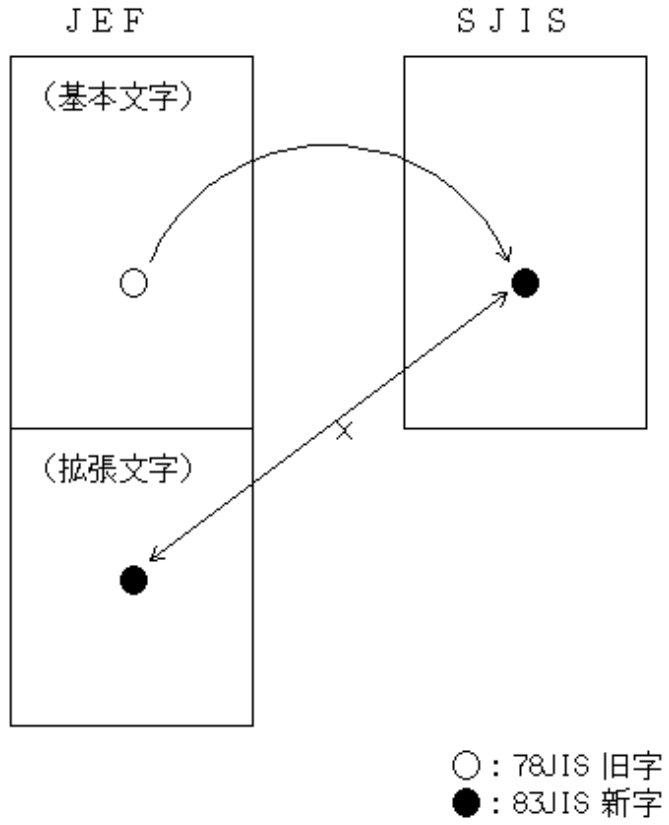
78JISによる旧字と83JISによる新字の違いを無視して、次のように変換を行います。



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。

L.4 他システムからの移行上の注意

L.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.3.30 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	日本語項目あり	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	―	●	―	―	―	●	―	―	●
	英数字項目			●	―	―	―	ERR	―	―	ERR
	日本語項目		●	●	●	●	ERR	●	●	ERR	●

		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					
	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 NC"†".
    :
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
```

L.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“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
MINUS SIGN	X"E28892"	X"2212"
FULLWIDTH HYPHEN-MINUS	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。

L.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.
```

L.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!!".
```

L.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。

L.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。

L.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。

L.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。
```

L.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。

L.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見えますが、実際には偽が成立します。PIC-Nには3文字の全角空白がつめられる[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つ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。

L.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。

L.5.9 大小比較

比較については、同じ字類どうしでも、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。

L.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。

  ー 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。

L.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。

L.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。

L.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)

付録M　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。

M.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。

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。

M.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、バッファオーバフロー(バッファオーバラン)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の作り込みを防止してください。バッファオーバ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。

M.3 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から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。

索　引

[A]

[B]

[C]

[D]

[J]

[K]

[L]

[M]

[N]

[O]

[P]

[Q]

[R]

[S]

[T]

[U]

[V]

[か]

[さ]

[た]

[な]

[は]

[ま]

[や]

[ら]

[わ]