NetCOBOL V10.5

使用手引書

Solaris

J2S2-1420-03Z0(00)
2013年1月

まえがき

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/

商標について

- UNIXは、米国およびその他の国におけるオープン・グループの登録商標です。
- X Window Systemは、オープン・グループの商標です。
- 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。
- Microsoft、Windows、Windows ServerおよびWindows Vistaは、米国Microsoft Corporationの米国およびその他の国における商標または登録商標です。
- C-ISAMは、米国Informix Software,Inc.の米国での登録商標です。
- その他の会社名または製品名は、それぞれ各社の商標または登録商標です。

製品の呼び名について

本書では、製品の名称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
Oracle Solaris 10 Oracle Solaris 11	Solaris
Microsoft(R) Windows(R) XP Home Edition Operating System Microsoft(R) Windows(R) XP Professional Operating System	Windows XP
Windows Vista(R) Home Basic Windows Vista(R) Home Premium Windows Vista(R) Business Windows Vista(R) Enterprise Windows Vista(R) Ultimate	Windows Vista
Windows(R) 7 Home Premium Windows(R) 7 Professional Windows(R) 7 Enterprise Windows(R) 7 Ultimate	Windows 7
Windows(R) 8 Windows(R) 8 Pro Windows(R) 8 Enterprise	Windows 8
Microsoft(R) Windows Server(R) 2003, Standard Edition Microsoft(R) Windows Server(R) 2003, Enterprise Edition Microsoft(R) Windows Server(R) 2003 R2, Standard Edition Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition Microsoft(R) Windows Server(R) 2003, Standard x64 Edition Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition	Windows Server 2003

正式名称	略称
Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition	
Microsoft(R) Windows Server(R) 2008 Foundation Microsoft(R) Windows Server(R) 2008 Standard Microsoft(R) Windows Server(R) 2008 Standard without Hyper-V(TM) Microsoft(R) Windows Server(R) 2008 Enterprise Microsoft(R) Windows Server(R) 2008 Enterprise without Hyper-V(TM) Microsoft(R) Windows Server(R) 2008 Datacenter Microsoft(R) Windows Server(R) 2008 Datacenter without Hyper-V(TM) Microsoft(R) Windows Server(R) 2008 R2 Foundation Microsoft(R) Windows Server(R) 2008 R2 Standard Microsoft(R) Windows Server(R) 2008 R2 Enterprise Microsoft(R) Windows Server(R) 2008 R2 Datacenter	Windows Server 2008
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

- Solaris上で動作する製品を「Solaris版の製品」と表記します。
- 以下をすべて指す場合は、「Windows」と表記します。
  - Windows XP
  - Windows Vista
  - Windows 7
  - Windows 8
  - Windows Server 2003
  - Windows Server 2008
  - Windows Server 2012

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、COBOL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。

NetCOBOLが出力するメッセージについては、“メッセージ説明書”をお読みください。

Sun日本語COBOLからの移行上の注意については、“リリース情報”の“互換情報”をお読みください。

本書の読者

本書は、NetCOBOLを使用してCOBOLプログラムを開発する方を対象に書かれています。本書を読むためには、以下の知識が必要です。

- COBOLの文法に関する基本的な知識
- ご使用になるOSに関する基本的な知識

本書の構成

本書の構成と内容は、以下のとおりです。

第1章 概要

この製品の動作環境および機能について説明しています。

第2章 プログラムの書き方

COBOLプログラムの書き方について説明しています。

第3章 プログラムの翻訳とリンク

COBOLプログラムを翻訳・リンクする方法について説明しています。

第4章 プログラムの実行

翻訳・リンクしたCOBOLプログラムを実行する方法について説明しています。

第5章 プログラムのデバッグ

この製品のデバッグ機能について説明しています。

第6章 ファイル処理

ファイルを使用する方法について説明しています。

第7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8章 画面を使った入出力

画面を使ってデータのやりとりを行う方法について説明しています。

第9章 サブプログラムを呼び出す～プログラム間連絡機能～

プログラムからプログラムを呼び出す方法について説明しています。

第10章 ACCEPT文およびDISPLAY文の使い方

ACCEPT文およびDISPLAY文を使った機能として、小入出力機能、コマンド行引数の操作機能、および環境変数の操作機能の使い方について説明しています。

第11章 SORT文およびMERGE文の使い方～整列併合機能～

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

第12章 システムプログラムを記述するための機能

システムプログラムを記述する場合に有効となる機能について説明しています。

第13章 オブジェクト指向プログラミングとは

オブジェクト指向プログラミングの概念について説明しています。

第14章 オブジェクト指向プログラミング機能～基本的な使い方～

オブジェクト指向プログラミング機能の基本的な使い方について説明しています。

第15章 オブジェクト指向プログラミング機能～さらに進んだ使い方～

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

第16章 オブジェクト指向プログラムの開発と実行

オブジェクト指向プログラムの開発方法および実行について説明しています。

第17章 マルチスレッド

マルチスレッドプログラムについて説明しています。

第18章 Unicode

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

第19章 通信機能

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

第20章 Web連携

COBOLが提供するWeb連携機能の概要について説明しています。

第21章 CORBAアプリケーション

COBOLインタフェースのCORBAのサーバ/クライアントアプリケーションの概要について説明しています。

第22章 プロジェクトマネージャの使い方

プロジェクトマネージャの使い方について説明しています。

第23章 対話型デバッガの使い方

対話型デバッガの使い方について説明しています。

第24章 SCREEN-DESIGNERの使い方

SCREEN-DESIGNERの使い方について説明しています。

第25章 ファイルユーティリティ

ファイルユーティリティの使い方について説明しています。

第26章 分散開発支援機能

Windows(R)版製品を使用してUNIX系システムのアプリケーションの開発を行う場合の分散開発について説明しています。

第27章 CSV形式データの操作

CSV形式データの操作について説明しています。

付録A 翻訳オプション

COBOLコンパイラに与える翻訳オプションについて説明しています。

付録B 入出力状態一覧

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

付録C 広域最適化

COBOLコンパイラが翻訳時に行う最適化の内容について説明しています。

付録D 組込み関数の使用

COBOLで使用できる組込み関数について説明しています。

付録E 環境変数一覧

この製品で使用する環境変数の一覧を記述しています。

付録F 特殊な定数の書き方

システム固有の定数の書き方について説明しています。

付録G COBOLが提供するサブルーチン

COBOLが提供するサブルーチンについて説明しています。

付録H オブジェクト指向と従来機能の組合せ

オブジェクト指向と従来機能の組合せについて説明しています。

付録I データベース連携

この製品をデータベースと連携して使用する場合の注意事項について説明しています。

付録J 日本語コード系

この製品で日本語処理を行う場合のコード系について説明しています。

付録K SCCSの利用方法

COBOL開発でSCCS(Source Code Control System)を利用する方法について説明しています。

付録L ldコマンド

COBOLコンパイラが生成した再配置可能プログラムを結合するときのldコマンドの入力形式および使い方について説明しています。

付録M makeコマンドの活用

COBOL開発でmakeコマンドを活用する方法について説明しています。

付録N Windowsを利用したUNIX版COBOLアプリケーションの作成手順

Windows(R)を利用してUNIX版COBOLアプリケーションを作成するための手順について説明します。

付録O OSIV系システムとの機能比較

OSIV系システムとこのシステムのCOBOLの機能比較について説明しています。

付録P COBOL G移行支援オプション

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

付録Q セキュリティ

セキュリティについて概要を説明しています。

付録R COBOLプログラムの作成技法

効率よいCOBOLプログラムの作成技法について説明しています。

注意事項

本書の情報は、プログラミングサービス情報です。COBOLを使用してプログラムを開発する際に使用することができます。

表記上の規約

本書は次の表記の規約で書かれています。

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。
$	Bourne shellのプロンプトを示します。
%	C shellのプロンプトを示します。
…	この記号の直前の項目を、繰り返して指定できることを示します。
あいうえお	プログラム例中で、可変文字列を示します。可変文字列は、実際にはほかの文字列に置き換えます。 例： PROGRAM-ID. **プログラム名**. → PROGRAM-ID. SAMPLE1.
{あい / うえお} または {あい |うえお}	{ }で囲まれた文字列の1つを選択することを示します。省略した場合、“_”(アンダーライン)の文字列が選択されたものとして扱われます。
[あいうえお]	[ ]で囲まれた文字列は省略できることを示します。

その他の注意事項

- 本書では、“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。

- 本書では、“Interstage Application Server”を総称して、“Interstage”として記述しています。なお、本書で記述している“Interstage”は、V4以前では“INTERSTAGE”に置き換えてお読みください。
- 本書では、“Interstage Charset Manager”の“標準コード変換”を、“標準コード変換”と記述しています。
- 本書では、OSIV/MSP、OSIV/XSPなどのOSIV系システムを総称して、“OSIV系システム”として記述しています。

お願い

- 本書を無断で他に転載しないようお願いします。
- 本書は予告なしに変更されることがあります。

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ください。

2013年1月

Copyright 2000-2013 FUJITSU LIMITED

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言語方式および言語の正確さと機能について、いかなる保証を与えるものでもなく、またそれに関連して、いかなる責任を負うものでもない。

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

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

目　次

第1章 概要.....1
1.1 機能.....1
1.1.1 COBOLの機能.....1
1.1.2 この製品が提供するプログラムおよびユーティリティ.....1
1.1.3 COBOLと連携機能.....2
1.2 開発環境.....3
1.2.1 関連製品.....4
1.3 資源一覧.....5

第2章 プログラムの書き方.....7
2.1 プログラムの作成と編集.....7
2.1.1 ソースプログラムの作成.....7
2.1.2 登録集原文の作成.....8
2.2 プログラムの形式.....8
2.3 翻訳指示文.....9

第3章 プログラムの翻訳とリンク.....10
3.1 翻訳とリンク.....10
3.1.1 翻訳時に設定する環境変数.....10
3.1.1.1 COBOLOPTS（翻訳オプションの指定）.....10
3.1.1.2 COBCOPY（登録集ファイルの格納ディレクトリの指定）.....10
3.1.1.3 COB_COPYNAME（登録集原文の検索条件の指定）.....10
3.1.1.4 COB_LIBSUFFIX（登録集ファイル名の拡張子の指定）.....11
3.1.1.5 SMED_SUFFIX（画面帳票定義体ファイル名の拡張子の指定）.....11
3.1.1.6 FORMLIB（画面帳票定義体ファイルの格納ディレクトリの指定）.....11
3.1.1.7 FFD_SUFFIX（ファイル定義体ファイル名の拡張子の指定）.....11
3.1.1.8 FILELIB（ファイル定義体ファイルの格納ディレクトリの指定）.....11
3.1.1.9 COB_REPIN（リポジトリファイルの入力先ディレクトリの指定）.....11
3.1.2 基本操作.....11
3.1.3 登録集（COPY文）を使ったプログラムの翻訳方法.....12
3.1.4 副プログラムを呼び出すプログラムの翻訳・リンク方法.....14
3.1.5 対話型デバッガを使う場合のプログラムの翻訳方法.....15
3.1.6 COBOLコンパイラが使用するファイル.....15
3.1.7 COBOLコンパイラが出力する情報.....17
3.1.7.1 診断メッセージ.....18
3.1.7.2 オプション情報リスト、翻訳単位統計情報リスト.....18
3.1.7.3 相互参照リスト.....19
3.1.7.4 ソースプログラムリスト.....20
3.1.7.5 目的プログラムリスト.....21
3.1.7.6 データエリアに関するリスト.....22
3.2 実行可能プログラムの構造.....28
3.2.1 概要.....28
3.2.2 結合の種類とプログラム構造.....29
3.2.3 単純構造の実行可能プログラムの作成方法.....32
3.2.4 動的リンク構造の実行可能プログラムの作成方法.....33
3.2.5 動的プログラム構造の実行可能プログラムの作成方法.....34
3.3 cobolコマンド.....34
3.3.1 翻訳に関するオプション.....35
3.3.1.1 -c（翻訳だけを行う指定）.....36
3.3.1.2 -Dc（COUNT機能 を使用する指定）.....36
3.3.1.3 -Dk（CHECK機能を使用する指定）.....36
3.3.1.4 -Dr（TRACE機能を使用する指定）.....37
3.3.1.5 -dr（リポジトリファイルの入出力先ディレクトリの指定）.....37
3.3.1.6 -ds（ソース解析情報ファイルの出力ディレクトリの指定）.....37
3.3.1.7 -Dt（対話型デバッガを使用する指定）.....37

3.3.1.8 -dd（デバッグ情報ファイルのディレクトリの指定）......38
3.3.1.9 -do（オブジェクトファイルのディレクトリの指定）......38
3.3.1.10 -dp（翻訳リストファイルのディレクトリの指定）......38
3.3.1.11 -f（ファイル定義体ファイルのディレクトリの指定）......39
3.3.1.12 -I（登録集ファイルのディレクトリの指定）......39
3.3.1.13 -i（オプションファイルの指定）......39
3.3.1.14 -M（主プログラムを翻訳するときの指定）......39
3.3.1.15 -m（画面帳票定義体ファイルのディレクトリの指定）......39
3.3.1.16 -P（翻訳リストのファイル名の指定）......40
3.3.1.17 -R（リポジトリファイルの入力先ディレクトリの指定）......40
3.3.1.18 -Tm（マルチスレッドモデルのプログラムを翻訳するときの指定）......40
3.3.1.19 -WC（翻訳オプションの指定）......40
3.3.2 リンクに関するオプション......41
3.3.2.1 -dy/-dn（結合モードの指定）......41
3.3.2.2 -G（共用オブジェクトプログラムを生成する指定）......41
3.3.2.3 -L（ライブラリサーチパス名を追加する指定）......41
3.3.2.4 -l（リンクする副プログラムまたはライブラリの指定）......42
3.3.2.5 -Ns（C言語で記述したプログラムから呼び出されるプログラムの指定）......42
3.3.2.6 -o（オブジェクトファイルの指定）......42
3.3.2.7 -pc（スクリーン操作機能を使うプログラムをリンクする指定）......42
3.3.2.8 -pi（C-ISAMを使うプログラムをリンクする指定）......42
3.3.2.9 -pm（画面定義体を使うプログラムをリンクする指定）......42
3.3.2.10 -Tm（マルチスレッドモデルのプログラムをリンクする指定）......42
3.3.2.11 -Wl（リンクオプションの指定）......42
3.3.3 cobolコマンドの復帰値......43

第4章 プログラムの実行......44
4.1 実行環境の設定......44
4.1.1 実行環境......44
4.1.2 実行環境の設定方法......48
4.1.2.1 シェルの初期化ファイルに設定する方法......49
4.1.2.2 環境変数設定コマンドで設定する方法......49
4.1.2.3 実行用の初期化ファイルに設定する方法......49
4.1.2.3.1 実行用の初期化ファイルの内容......49
4.1.2.3.2 実行用の初期化ファイルの検索順序について......50
4.1.2.3.3 ライブラリ格納ディレクトリの実行用の初期化ファイルの使用について......51
4.1.2.3.4 実行用の初期化ファイルの情報を表示する方法......52
4.1.2.4 コマンド行で設定する方法......52
4.1.3 環境変数による接続製品の指定......52
4.1.4 副プログラムのエントリ情報......54
4.1.4.1 副プログラム名の指定形式......54
4.1.4.2 二次入口点名の指定形式......56
4.2 実行操作......56
4.2.1 プログラムの実行形式......57
4.2.2 実行時オプションを指定する......58
4.2.2.1 [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）......58
4.2.2.2 [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）.58
4.2.2.3 [s値]（外部スイッチの値の指定）......59
4.2.2.4 [smsize値k]（PowerSORTが使用するメモリ容量を指定）......59
4.2.3 実行用の初期化ファイルを指定する......59
4.2.4 OSIV系形式の実行時パラメタを指定する......59
4.3 実行時メッセージの出力方法の指定......60
4.3.1 実行時メッセージの重大度指定......60
4.3.2 実行時メッセージのファイル出力......61
4.3.3 実行時メッセージのSyslog出力......62
4.3.4 実行時メッセージのInterstage Business Application Serverの汎用ログへの出力......62
4.4 終了ステータス......63

4.5 注意事項....64
4.5.1 COBOLプログラムの実行時にスタックオーバーフローが発生する場合....64
4.5.2 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....69
4.5.3 英語環境下でのフォントについて....69

第5章 プログラムのデバッグ....70
5.1 デバッグ機能の種類....70
5.1.1 ステートメント番号....71
5.2 TRACE機能の使い方....71
5.2.1 デバッグ作業の流れ....72
5.2.2 トレース情報....72
5.2.3 注意事項....74
5.3 CHECK機能の使い方....75
5.3.1 デバッグ作業の流れ....76
5.3.2 出力メッセージ....76
5.3.3 CHECK機能の使用例....78
5.3.4 注意事項....82
5.4 COUNT機能の使い方....82
5.4.1 デバッグ作業の流れ....82
5.4.2 COUNT情報....83
5.4.3 COUNT機能を使用したプログラムのデバッグ....87
5.4.4 注意事項....88
5.5 メモリチェック機能の使い方....88
5.5.1 デバッグ作業の流れ....88
5.5.2 出力メッセージ....89
5.5.3 プログラムの特定....89
5.5.4 注意事項....90
5.6 異常終了時の障害発生箇所の特定方法....90
5.6.1 デバッグ作業の流れ....90
5.6.2 障害発生箇所の特定方法....91

第6章 ファイル処理....95
6.1 ファイルの種類....95
6.1.1 ファイルの種類と特徴....95
6.1.2 レコードの設計....98
6.1.2.1 レコード形式....98
6.1.2.2 索引ファイルのレコードキー....99
6.1.3 ファイルの処理方法....99
6.2 レコード順ファイルの使い方....100
6.2.1 レコード順ファイルの定義....100
6.2.2 レコード順ファイルのレコードの定義....101
6.2.3 レコード順ファイルの処理....102
6.3 行順ファイルの使い方....104
6.3.1 行順ファイルの定義....105
6.3.2 行順ファイルのレコードの定義....105
6.3.3 行順ファイルの処理....105
6.3.4 注意事項....109
6.4 相対ファイルの使い方....109
6.4.1 相対ファイルの定義....110
6.4.2 相対ファイルのレコードの定義....111
6.4.3 相対ファイルの処理....112
6.5 索引ファイルの使い方....116
6.5.1 索引ファイルの定義....116
6.5.2 索引ファイルのレコードの定義....118
6.5.3 索引ファイルの処理....119
6.6 入出力エラー処理....124
6.6.1 AT END指定....124
6.6.2 INVALID KEY指定....125

6.6.3 FILE STATUS句.....125
6.6.4 誤り処理手続き.....126
6.6.5 入出力エラーが発生したときの実行結果.....126
6.7 ファイル処理の実行.....127
6.7.1 ファイルの割当て.....127
6.7.2 ファイル処理の結果.....130
6.7.3 ファイルの排他制御.....131
6.7.3.1 ファイルを排他モードにする方法.....131
6.7.3.2 レコードを排他状態にする方法.....132
6.8 その他のファイル機能.....134
6.8.1 C-ISAMの使い方.....136
6.8.1.1 C-ISAMの指定.....136
6.8.1.2 プログラムの翻訳・リンク.....136
6.8.1.3 C-ISAMでの注意点.....137
6.8.2 大容量ファイル処理.....137
6.8.2.1 大容量ファイルアクセスの指定.....138
6.8.2.2 一括指定.....138
6.8.3 RDMファイル.....139
6.8.3.1 ファイル環境の指定.....139
6.8.3.2 注意事項.....140
6.8.4 性能向上のための機構.....140
6.8.4.1 ファイル処理の性能改善.....140
6.8.4.2 ファイルの高速処理.....141
6.8.5 ファイル書込みに関する機構.....144
6.8.5.1 クローズ時の書込み内容の即時反映.....144
6.8.5.2 行順ファイルの後置空白に関する指定.....145
6.8.6 COBOLファイルアクセスルーチン.....145
6.8.7 ファイル追加書き.....145
6.8.8 ファイルの連結.....146
6.8.9 ダミーファイル.....146
6.8.10 名前付きパイプ.....147
6.8.11 外部ファイルハンドラ.....148
6.8.12 注意事項.....150

第7章 印刷処理.....153
7.1 印刷方法の種類.....153
7.1.1 各印刷方法の概要.....153
7.1.2 印刷装置.....155
7.1.3 印字文字.....156
7.1.4 環境変数の設定.....160
7.1.5 印刷情報ファイル.....164
7.1.6 FCB.....169
7.1.7 フォームオーバレイパターン.....172
7.1.8 帳票定義体.....173
7.1.9 フォントテーブル.....173
7.1.10 他社プリンタ装置での注意事項.....176
7.1.11 特殊レジスタ.....178
7.1.12 印刷ファイル/表示ファイルの決定方法.....179
7.1.13 帳票設計について.....180
7.1.14 印刷不可能な領域について.....181
7.1.15 I制御レコード/S制御レコード.....182
7.1.16 Unicode(UTF-8)印刷について.....186
7.2 行単位のデータを印刷する方法.....187
7.2.1 概要.....187
7.2.2 プログラムの記述.....187
7.2.3 プログラムの翻訳・リンク.....189
7.2.4 プログラムの実行.....189

7.3 フォームオーバレイおよびFCBを使う方法....190
7.3.1 概要....190
7.3.2 プログラムの記述....191
7.3.3 プログラムの翻訳・リンク....193
7.3.4 プログラムの実行....193
7.4 帳票定義体を使う印刷ファイルの使い方....195
7.4.1 概要....195
7.4.1.1 帳票のパーティション....196
7.4.1.2 帳票の電子化....198
7.4.2 プログラムの記述....200
7.4.3 プログラムの翻訳・リンク....203
7.4.4 プログラムの実行....203
7.5 表示ファイル(帳票印刷)の使い方....204
7.5.1 概要....204
7.5.2 作業手順....206
7.5.3 帳票定義体の作成....206
7.5.4 プログラムの記述....207
7.5.5 プログラムの翻訳・リンク....209
7.5.6 プリンタ情報ファイルの作成....209
7.5.7 プログラムの実行....210
7.6 電子帳票出力機能を使う方法....211
7.6.1 電子帳票出力機能の概要....211
7.6.2 帳票を電子化する方法....211
7.6.2.1 プログラムの記述....211
7.6.2.2 印刷情報ファイルの定義....215
7.6.2.3 フォントの指定....215
7.6.2.4 ListWorksの準備・設定....216
7.6.2.5 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....216
7.6.3 電子帳票の出力例....217
7.6.4 プリンタ(紙)と電子帳票出力時の機能差(留意事項/制限事項)....217
7.6.4.1 環境変数を利用する機能....217
7.6.4.2 印刷情報ファイルを利用する機能....217
7.6.4.3 プログラムの指定関連....218
7.6.4.4 その他(ListWorksの制限事項)....219
7.6.5 実行時エラーについて....219

第8章 画面を使った入出力....221
8.1 画面を使った入出力の種類....221
8.2 表示ファイル機能(画面入出力)....221
8.2.1 概要....221
8.2.2 作業手順....223
8.2.3 画面定義体の作成....223
8.2.4 プログラムの記述....224
8.2.5 プログラムの翻訳・リンク....227
8.2.6 ウィンドウ情報ファイルの作成....228
8.2.7 プログラムの実行....228
8.3 スクリーン操作機能....229
8.3.1 概要....229
8.3.2 画面項目....229
8.3.3 キー定義ファイルの利用....230
8.3.4 プログラムの記述....232
8.3.5 プログラムの翻訳・リンク....233

第9章 サブプログラムを呼び出す～プログラム間連絡機能～....234
9.1 呼出し関係の形態....234
9.1.1 COBOLの言語間の環境....234
9.1.2 動的プログラム構造....237
9.1.2.1 動的プログラム構造の特徴....237

9.1.2.2 副プログラムのエントリ情報 238
9.1.2.3 注意事項 238
9.2 COBOLプログラムからCOBOLプログラムを呼び出す 241
9.2.1 呼出し方法 242
9.2.2 二次入口 242
9.2.3 制御の復帰とプログラムの終了 242
9.2.4 パラメタの受渡し 242
9.2.5 データの共用 244
9.2.5.1 外部データ使用時の注意事項 245
9.2.5.2 外部ファイル使用時の注意事項 245
9.2.6 復帰コード 245
9.2.7 内部プログラム 246
9.2.8 注意事項 248
9.3 C言語プログラムとのリンク 248
9.3.1 COBOLプログラムからCプログラムを呼び出す方法 248
9.3.1.1 呼出し方法 249
9.3.1.2 パラメタの受渡し方法 249
9.3.1.3 復帰コード(関数値) 251
9.3.2 CプログラムからCOBOLプログラムを呼び出す方法 252
9.3.2.1 呼出し方法 253
9.3.2.2 パラメタの受渡し方法 253
9.3.2.3 復帰コード(関数値) 253
9.3.3 データ型の対応 255
9.3.4 C言語プログラムとのデータの共用 256
9.3.5 翻訳・リンク方法 258
9.3.6 注意事項 260

第10章 ACCEPT文およびDISPLAY文の使い方 263
10.1 小入出力 263
10.1.1 概要 263
10.1.2 入出力先の種類と指定方法 263
10.1.3 システムの標準入出力(stdin/stdout)を使うプログラム 264
10.1.3.1 プログラムの記述 265
10.1.3.2 プログラムの翻訳・リンク 265
10.1.3.3 プログラムの実行 265
10.1.3.4 数字データの入力 265
10.1.4 Systemwalkerのコンソールを使うプログラム 266
10.1.4.1 プログラムの記述 266
10.1.4.2 プログラムの翻訳・リンク 266
10.1.4.3 プログラムの実行 266
10.1.4.4 数字データの入力 272
10.1.5 Interstage Business Application Serverの汎用ログを使うプログラム 272
10.1.5.1 プログラムの記述 272
10.1.5.2 プログラムの翻訳・リンク 273
10.1.5.3 プログラムの実行 273
10.1.6 システムの標準エラー出力(stderr)を使うプログラム 273
10.1.6.1 プログラムの記述 273
10.1.6.2 プログラムの翻訳・リンク 274
10.1.6.3 プログラムの実行 274
10.1.7 ファイルを使うプログラム 274
10.1.7.1 プログラムの記述 274
10.1.7.2 プログラムの翻訳・リンク 275
10.1.7.3 プログラムの実行 276
10.1.7.4 DISPLAY文のファイル出力拡張機能 276
10.1.7.5 ACCEPT文のファイル入力拡張機能 278
10.1.8 現在の日付および時刻の入力 279
10.1.8.1 プログラムの記述 279

10.1.8.2 プログラムの翻訳・リンク......280
10.1.8.3 プログラムの実行......280
10.1.9 任意の日付の入力......280
10.1.9.1 プログラムの記述......280
10.1.9.2 プログラムの翻訳・リンク......281
10.1.9.3 プログラムの実行......281
10.1.10 シスログを使うプログラム......281
10.1.10.1 プログラムの記述......281
10.1.10.2 プログラムの翻訳・リンク......282
10.1.10.3 プログラムの実行......282
10.2 コマンド行引数の取出し......283
10.2.1 概要......283
10.2.2 プログラムの記述......283
10.2.3 プログラムの翻訳・リンク......285
10.2.4 プログラムの実行......285
10.3 環境変数の操作機能......285
10.3.1 概要......285
10.3.2 プログラムの記述......286
10.3.3 プログラムの翻訳・リンク......287
10.3.4 プログラムの実行......287

第11章 SORT文およびMERGE文の使い方～整列併合機能～......288
11.1 ソート・マージ処理の概要......288
11.2 ソートの使い方......289
11.2.1 ソート処理の種類......289
11.2.2 プログラムの記述......289
11.2.3 プログラムの翻訳・リンク......292
11.2.4 プログラムの実行......292
11.3 マージの使い方......292
11.3.1 マージ処理の種類......292
11.3.2 プログラムの記述......293
11.3.3 プログラムの翻訳・リンク......295
11.3.4 プログラムの実行......295

第12章 システムプログラムを記述するための機能......296
12.1 SD機能の種類......296
12.2 ポインタ付けの使い方......296
12.2.1 概要......296
12.2.2 プログラムの記述......296
12.2.3 プログラムの翻訳・リンク......297
12.2.4 プログラムの実行......297
12.3 ADDR関数とLENG関数の使い方......297
12.3.1 概要......297
12.3.2 プログラムの記述......297
12.3.3 プログラムの翻訳・リンク......298
12.3.4 プログラムの実行......298
12.4 終了条件なしのPERFORM文の使い方......298
12.4.1 概要......298
12.4.2 プログラムの記述......298
12.4.3 プログラムの翻訳・リンク......299
12.4.4 プログラムの実行......299

第13章 オブジェクト指向プログラミングとは......300
13.1 オブジェクト指向の歴史的背景......300
13.1.1 ソフトウェア工学以前......300
13.1.2 構造化プログラミングの出現......300
13.1.3 モジュール化......300
13.1.4 抽象データ型......301

13.1.5 オブジェクト指向の誕生 302
13.2 オブジェクト指向の基本的な考え方 303
13.2.1 オブジェクトとクラス 303
13.2.2 オブジェクトインスタンスの作成・参照 304
13.2.3 メソッドの呼出し 305
13.2.4 継承 306
13.2.5 多態と動的束縛 309
13.3 オブジェクト指向のメリット 311

第14章 オブジェクト指向プログラミング機能～基本的な使い方～ 312
14.1 ソース定義 312
14.1.1 クラス定義 313
14.1.2 ファクトリ定義 314
14.1.3 オブジェクト定義 315
14.1.4 メソッド定義 317
14.2 オブジェクトインスタンスの操作 319
14.2.1 メソッドの呼出し 320
14.2.1.1 オブジェクト参照項目 320
14.2.1.2 INVOKE文 322
14.2.1.3 パラメタの指定 322
14.2.2 オブジェクトの寿命 324
14.3 継承 325
14.3.1 継承の概念と実現 326
14.3.2 FJBASEクラス 328
14.3.3 メソッドの上書き 330
14.4 適合 331
14.4.1 適合の概念 331
14.4.2 オブジェクト参照項目と適合チェック 333
14.4.3 翻訳時の適合チェックと実行時の適合チェック 334
14.4.3.1 代入時の適合チェック 334
14.4.3.2 メソッド呼出し時の適合チェック 334
14.5 リポジトリ 335
14.5.1 リポジトリファイルの概要 335
14.5.1.1 継承の実現 335
14.5.1.2 適合チェックの実現 336
14.5.2 リポジトリファイル更新の影響 336
14.6 メソッドの束縛 337
14.6.1 メソッドの静的束縛 338
14.6.2 メソッドの動的束縛と多態 338
14.6.3 定義済みオブジェクト一意名SUPER 341
14.6.4 定義済みオブジェクト一意名SELF 342
14.7 少し進んだ使い方 344
14.7.1 メソッドのPROTOTYPE宣言 344
14.7.2 多重継承 345
14.7.3 行内呼出し 346
14.7.4 オブジェクト指定子 347
14.7.5 PROPERTY句 349
14.7.6 初期化処理メソッドと終了処理メソッド 351
14.7.7 間接参照クラス 353
14.7.8 相互参照クラス 355
14.7.8.1 相互参照パターン 355
14.7.8.2 相互参照クラスの翻訳 358
14.7.8.3 相互参照クラスのリンク 359
14.7.8.4 相互参照クラスの実行 360

第15章 オブジェクト指向プログラミング機能～さらに進んだ使い方～ 361
15.1 例外処理 361
15.1.1 概要 361

15.1.2 例外オブジェクト.....361
15.1.3 RAISE文の動作.....362
15.1.4 RAISING指定のEXIT文の動作.....362
15.2 C++プログラムとの連携.....364
15.2.1 概要.....364
15.2.2 C++連携の方法.....364
15.2.3 C++連携の概要.....364
15.2.3.1 COBOLおよびC++でのクラスの対応.....365
15.2.3.2 処理の概要.....365
15.2.3.3 インタフェースプログラムの仕組み.....365
15.2.4 C++連携のプログラム手順.....367
15.2.4.1 C++で定義されているクラスを調べる.....368
15.2.4.2 COBOL側での定義.....368
15.2.4.3 C++側での定義.....368
15.2.5 COBOLからの利用.....369
15.2.6 サンプルプログラム.....369
15.3 オブジェクトの永続化.....372
15.3.1 オブジェクトの永続化とは.....372
15.3.2 概要.....372
15.3.3 クラス構造の例.....372
15.3.4 索引ファイルとオブジェクトの対応.....373
15.3.4.1 クラスとファイルの対応.....373
15.3.4.2 索引ファイルの定義.....375
15.3.5 オブジェクトの保存/復元.....376
15.3.5.1 索引ファイル操作クラス.....376
15.3.5.2 保存するオブジェクトのメソッドの追加.....376
15.3.6 処理の流れ.....378
15.4 特殊クラス.....379
15.4.1 クラスライブラリ連携(*COB-BINDTABLE).....379
15.4.1.1 概要.....379
15.4.1.2 *COB-BINDTABLEクラスの説明.....381
15.4.1.2.1 *COB-BINDTABLEクラスのファクトリメソッド.....381
15.4.1.2.2 *COB-BINDTABLEクラスのオブジェクトメソッド.....381
15.5 ANY LENGTH句を使用したプログラミング.....386
15.5.1 文字列を扱うクラス.....386
15.5.2 ANY LENGTH句の使用.....387

第16章 オブジェクト指向プログラムの開発と実行.....389
16.1 オブジェクト指向プログラミングで使用する資源.....389
16.2 開発手順.....389
16.3 クラスの設計.....390
16.4 使用するクラスの選定.....390
16.5 プログラム構造.....391
16.5.1 翻訳単位とリンク単位.....391
16.5.2 プログラム構造の概要.....392
16.5.2.1 静的構造.....392
16.5.2.2 動的構造.....392
16.6 翻訳処理.....396
16.6.1 リポジトリファイルと翻訳の手順.....396
16.6.2 動的プログラム構造での翻訳処理.....401
16.7 リンク処理.....403
16.7.1 リンク関係とリンクの手順.....404
16.7.2 動的プログラム構造でのリンク処理.....407
16.7.3 共用オブジェクトファイルの構成とファイル名.....407
16.7.4 クラスとメソッドのエントリ情報.....408
16.8 クラスの公開.....411
16.9 実行時の注意事項.....411

16.9.1 スタックオーバーフロー......411
16.9.2 オブジェクトインスタンスのブロック化......411
16.9.2.1 概要......412
16.9.2.2 使用メモリの節約......412
16.9.2.3 実行性能の向上......413
16.9.2.4 メモリのチューニングに関する実行環境情報......414
16.9.2.4.1 環境変数......414
16.9.2.4.2 クラス情報......414

第17章 マルチスレッド......416
17.1 概要......416
17.1.1 特徴......416
17.2 マルチスレッドのメリット......416
17.2.1 スレッドとは......416
17.2.2 マルチスレッドモデルとプロセスモデル......417
17.2.3 マルチスレッドの効果......417
17.3 COBOLプログラムの動作......419
17.3.1 実行環境と実行単位......419
17.3.2 マルチスレッドモデルのプログラムのデータの扱い......422
17.3.2.1 プログラム定義に宣言されたデータ......423
17.3.2.2 ファクトリオブジェクトとオブジェクトインスタンス......425
17.3.2.3 メソッド定義に宣言されたデータ......428
17.3.2.4 スレッド間共有外部データと外部ファイル......429
17.4 スレッド間の資源の共有......429
17.4.1 資源の共有......430
17.4.2 競合状態......430
17.4.3 COBOLでの資源の共有......431
17.4.3.1 スレッド間共有外部データと外部ファイル......431
17.4.3.2 ファクトリオブジェクト......432
17.4.3.3 オブジェクトインスタンス......434
17.5 基本的な使い方......436
17.5.1 動的プログラム構造......436
17.5.2 入出力機能の利用......436
17.5.2.1 同一ファイルの共有......436
17.5.2.2 同一プログラムでの複数ファイルの操作......437
17.5.2.3 注意事項......439
17.5.3 印刷機能の利用......439
17.5.4 スクリーン操作機能の利用......439
17.5.5 DISPLAY文およびACCEPT文の利用......439
17.5.5.1 小入出力機能について......439
17.5.5.2 コマンド行引数および環境変数の操作機能について......442
17.5.5.2.1 コマンド行引数の操作機能......442
17.5.5.2.2 環境変数の操作機能......442
17.5.6 オブジェクト指向プログラミング機能の利用......442
17.5.7 簡易アプリ間通信機能の利用......442
17.5.8 連携機能の利用......443
17.5.8.1 Symfoware連携......443
17.5.8.1.1 プログラムの記述......443
17.5.8.1.2 プログラムの翻訳･リンク......443
17.5.8.1.3 プログラムの実行......443
17.5.9 多重動作ができないプログラムの呼出し......443
17.6 少し進んだ使い方......443
17.6.1 入出力機能の利用......443
17.6.1.1 スレッド間共有外部ファイル......444
17.6.1.2 ファクトリオブジェクト内に定義したファイル......447
17.6.1.3 オブジェクト内に定義したファイル......448
17.6.2 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......451

17.6.2.1 概要....451
17.6.2.2 起動方法....452
17.6.2.3 パラメタの受渡し方法....452
17.6.2.4 復帰コード(関数値)....452
17.6.2.5 翻訳とリンク....453
17.6.2.5.1 翻訳....454
17.6.2.5.2 リンク....455
17.6.3 スレッド間で実行単位の資源を引き継ぐ方法....455
17.6.3.1 概要....455
17.6.3.2 利用方法....456
17.6.3.3 サブルーチンの使い方....457
17.6.3.3.1 COBOL実行単位ハンドル取得サブルーチン....457
17.6.3.3.2 COBOL実行単位ハンドル設定サブルーチン....457
17.6.3.4 注意事項....458
17.7 翻訳から実行までの方法....459
17.7.1 翻訳とリンク....459
17.7.1.1 COBOLプログラムだけで共用オブジェクトプログラムを作成する場合....460
17.7.1.2 COBOLプログラムとCプログラムで共用オブジェクトプログラムを作成する場合....460
17.7.2 実行....461
17.7.2.1 実行用の初期化ファイル....461
17.7.2.2 実行環境変数の設定....461
17.7.2.2.1 実行環境変数の指定形式....461
17.7.3 マルチスレッドモデルとプロセスモデルの混在チェック....461
17.7.3.1 実行時チェック....461
17.7.3.2 プログラムのリンクチェック....462
17.8 マルチスレッドモデルのプログラムのデバッグ方法....462
17.8.1 マルチスレッドモデルのデバッグ....462
17.8.2 マルチスレッドモデルのデバッグ機能....462
17.8.2.1 TRACE機能....463
17.8.2.2 CHECK機能....464
17.8.2.3 COUNT機能....464
17.8.2.4 対話型デバッガ....465
17.8.2.5 障害発生箇所の特定方法....465
17.9 スレッド同期制御サブルーチン....465
17.9.1 データロックサブルーチン....466
17.9.1.1 COB_LOCK_DATA....466
17.9.1.2 COB_UNLOCK_DATA....467
17.9.2 オブジェクトロックサブルーチン....467
17.9.2.1 COB_LOCK_OBJECT....468
17.9.2.2 COB_UNLOCK_OBJECT....468
17.9.3 エラーコード....469

第18章 Unicode....470
18.1 Unicode概要....470
18.1.1 資源....470
18.1.2 表現形式....470
18.1.3 言語要素....471
18.1.4 コード系の混在....472
18.1.5 資産移行支援....473
18.2 Unicodeアプリケーションの作成....473
18.2.1 プログラムの作成、編集....473
18.2.2 翻訳、結合、実行....473
18.2.3 デバッグ....473
18.3 コーディング上の注意点....473
18.3.1 半角カナについて....473
18.3.2 文字定数....473
18.3.3 項目の再定義....474

18.3.4 転記....474
18.3.5 比較....475
18.3.6 ACCEPT/DISPLAY文....475
18.3.7 COBOLファイル....476
18.3.8 スクリーン機能....478
18.4 実行時の注意点....478
18.4.1 メッセージを出力するファイル....478
18.4.2 フォントについて....478
18.5 関連製品連携....478
18.5.1 FORM/MeFt....478
18.5.2 他言語間結合....479
18.5.3 他のファイルシステム....480
18.5.4 通信機能....480
18.5.5 Web連携....481

第19章 通信機能....482
19.1 通信の種類....482
19.2 表示ファイル機能(非同期型メッセージ通信)....482
19.2.1 動作環境....483
19.2.2 プログラムの記述....483
19.2.3 プログラムの翻訳・リンク....485
19.2.4 プログラムの実行....485
19.3 簡易アプリ間通信機能....485
19.3.1 概要....486
19.3.2 基本的な使い方....486
19.3.3 サービス名の定義....487
19.3.4 論理宛先定義ファイル....487
19.3.5 サーバの起動と論理宛先の創成....489
19.3.6 クライアントの処理....489
19.3.7 サーバの停止と論理宛先の削除....489
19.3.8 論理宛先の状態表示....489
19.3.9 ログ....490
19.3.10 コマンド....493
19.3.10.1 cobstrciコマンド....493
19.3.10.2 cobstpciコマンド....493
19.3.10.3 cobcrtldコマンド....494
19.3.10.4 cobdltldコマンド....494
19.3.10.5 cobactldコマンド....494
19.3.10.6 cobdacldコマンド....494
19.3.10.7 cobpurldコマンド....495
19.3.10.8 cobdspldコマンド....495
19.3.10.9 cobstrlgコマンド....495
19.3.10.10 cobstplgコマンド....496
19.3.11 関数....496
19.3.11.1 COBCI_OPEN関数....497
19.3.11.2 COBCI_CLOSE関数....498
19.3.11.3 COBCI_READ関数....499
19.3.11.4 COBCI_WRITE関数....500
19.3.12 関数のエラーコード....502
19.3.13 プログラムのリンクに関する注意事項....504

第20章 Web連携....505

第21章 CORBAアプリケーション....506
21.1 CORBAの概要....506
21.2 注意事項....506

第22章 プロジェクトマネージャの使い方....507

22.1 プロジェクトマネージャの概念....507
22.2 プロジェクトマネージャ....508
22.2.1 プロジェクトマネージャの使い方....509
22.2.1.1 設定(プロジェクト管理機能)....509
22.2.1.2 プロジェクト構成の定義....509
22.2.1.3 ソースプログラムの作成・変更....509
22.2.1.4 ソースプログラムの解析機能....510
22.2.1.5 依存調査....511
22.2.1.6 解析....511
22.2.1.7 ビルド....511
22.2.1.8 エラー修正....512
22.2.1.9 テスト機能....512
22.2.1.10 デバッグ機能....512
22.2.1.11 性能....512
22.2.1.12 実行....513
22.2.1.13 梱包....513
22.2.1.14 カスタマイズ....513
22.2.2 プロジェクトマネージャのリファレンス....513
22.2.2.1 プロジェクトマネージャのメニュー....513
22.2.2.2 プロジェクトマネージャのウィンドウおよびダイアログ・ボックス....515
22.2.3 カスタマイズ変数一覧....524
22.3 エディタ....524
22.3.1 エディタの使い方....525
22.3.1.1 テキストファイルの新規作成および編集....525
22.3.1.2 ツール連携....525
22.3.2 エディタのリファレンス....526
22.3.2.1 エディタのメニュー....526
22.3.2.2 エディタのウィンドウおよびダイアログ・ボックス....527
22.3.3 カスタマイズ変数一覧....529
22.4 メッセージ管理ツール....530
22.4.1 メッセージ管理ツールの使い方....530
22.4.1.1 メッセージファイルの新規作成および編集....530
22.4.1.2 メッセージの管理....531
22.4.1.3 ツール連携....531
22.4.2 メッセージ管理ツールのリファレンス....531
22.4.2.1 メッセージ管理ツールのメニュー....531
22.4.2.2 メッセージ管理ツールのウィンドウおよびダイアログ・ボックス....532
22.4.3 カスタマイズ変数一覧....534
22.5 ソース解析ツール....535
22.5.1 ソース解析ツールの使い方....536
22.5.1.1 ソースプログラムの解析....536
22.5.1.2 解析結果の管理....536
22.5.1.3 ツール連携....537
22.5.2 ソース解析ツールのリファレンス....537
22.5.2.1 ソース解析ツールのメニュー....537
22.5.2.2 ソース解析ツールのウィンドウおよびダイアログ・ボックス....538
22.5.3 カスタマイズ変数一覧....540
22.6 ビルダ....541
22.6.1 ビルダの使い方....542
22.6.1.1 翻訳およびリンク....542
22.6.1.2 翻訳結果の管理....542
22.6.1.3 ツール連携....542
22.6.2 ビルダのリファレンス....543
22.6.2.1 ビルダのメニュー....543
22.6.2.2 ビルダのウィンドウおよびダイアログ・ボックス....544
22.6.3 カスタマイズ変数一覧....546
22.7 翻訳オプション設定ツール....547

22.7.1 翻訳オプション設定ツールの使い方....547
22.7.1.1 翻訳オプション設定....547
22.7.2 翻訳オプション設定ツールのリファレンス....548
22.7.2.1 翻訳オプション設定ツールのメニュー....548
22.7.2.2 翻訳オプション設定ツールのウィンドウおよびダイアログ・ボックス....548
22.8 リンクオプション設定ツール....549
22.8.1 リンクオプション設定ツールの使い方....549
22.8.1.1 リンクオプション設定....550
22.8.2 リンクオプション設定ツールのリファレンス....550
22.8.2.1 リンクオプション設定ツールのメニュー....550
22.8.2.2 リンクオプション設定ツールのウィンドウおよびダイアログ・ボックス....550

第23章 対話型デバッガの使い方....552
23.1 デバッガの概要....552
23.1.1 デバッガの処理モード....552
23.2 デバッグの手順....552
23.2.1 デバッグ対象プログラム....553
23.2.2 翻訳....554
23.2.3 リンク....554
23.3 デバッガの起動....554
23.3.1 スクリーンモードでの起動....554
23.3.2 ラインモードでの起動....557
23.3.3 svdコマンド....557
23.3.4 注意事項....560
23.4 デバッガの操作....561
23.4.1 画面の構成....561
23.4.2 サブコマンドの入力....564
23.4.2.1 画面入力....564
23.4.2.2 キーボード入力....565
23.5 デバッガの機能....565
23.5.1 データの内容を確認しながらプログラムの誤りを検出する....565
23.5.2 プログラムの実行順序を変更する....566
23.5.3 データの変更箇所を調べる....566
23.5.4 実行経路をたどる....566
23.5.5 デバッガの状態を知る....567
23.5.6 デバッグ作業を自動化する....567
23.5.7 パラメタ領域を獲得する....568
23.5.8 デバッグ環境を変更する....569
23.5.9 テスト検証およびチューニングを行う....569
23.5.10 動的結合のプログラムをデバッグする....569
23.5.11 オブジェクト指向に対応したデバッグ機能....569
23.6 テスト網羅度測定機能....571
23.6.1 テスト網羅度測定機能の流れ....571
23.6.1.1 処理の流れ....571
23.6.1.2 資源の流れ....572
23.6.2 テスト網羅度測定機能の指定方法....574
23.6.3 テスト網羅度測定の機能分類....574
23.6.3.1 実行済み行の表示....574
23.6.3.2 テスト進捗の管理....575
23.6.3.3 未実行文の検索....576
23.6.3.4 テスト網羅度測定の状態変更....577
23.6.4 テスト進捗ファイルの管理....578
23.6.4.1 テスト進捗情報のマージ....578
23.6.4.2 テスト進捗情報のプログラム表示....579
23.6.5 テスト網羅度測定機能の注意事項....579
23.7 カバレージ機能....580
23.7.1 カバレージコマンドのオプション....581

23.7.2 カバレージコマンド使用例....583
23.8 Interstageなどのサーバ環境で動作するプログラムのデバッグ....586
23.8.1 概要....586
23.8.2 指定方法....586
23.8.2.1 環境変数の指定....587
23.8.2.2 Xサーバへの接続許可....588
23.8.3 注意事項....589
23.9 サブコマンド機能....589
23.9.1 記述形式....590
23.9.2 共通オペランドの説明....591
23.9.2.1 プログラム名....591
23.9.2.2 プログラム修飾....592
23.9.2.3 行番号および行内識別番号....593
23.9.2.4 文識別番号....594
23.9.2.5 識別名....595
23.9.2.6 定数....596
23.9.2.7 式....596
23.9.2.8 文キーワード....596
23.9.2.9 再帰呼出しレベル....597
23.9.3 サブコマンドの説明....597
23.9.4 UNIXコマンド連携....617
23.10 注意事項....618

第24章 SCREEN-DESIGNERの使い方....621
24.1 概要....621
24.2 S-DESIGNの機能....621
24.3 S-DESIGNの操作概要....621
24.3.1 起動....621
24.3.2 画面設計....622
24.3.3 画面設計域の操作方法....622
24.3.4 スクリーン名の指定....624
24.3.5 登録集生成....624
24.4 関連コマンド....625

第25章 ファイルユーティリティ....626
25.1 概要....626
25.2 ファイルユーティリティの機能....626
25.2.1 機能概要....626
25.2.2 ファイルの操作方法....627
25.3 スクリーンモードの使い方....632
25.3.1 起動と機能選択....632
25.4 コマンドモードの使い方....634
25.4.1 変換....634
25.4.2 ロード....636
25.4.3 アンロード....638
25.4.4 表示....639
25.4.5 整列....641
25.4.6 属性....643
25.4.7 復旧....644
25.4.8 再編成....645

第26章 分散開発支援機能....646
26.1 分散開発の概要....646
26.1.1 分散開発のメリット....646
26.1.2 分散開発の機能範囲....646
26.2 分散開発支援機能....648

第27章 CSV形式データの操作....651

27.1 CSV形式データとは......651
27.2 CSV形式データの作成(STRING文)......652
27.2.1 基本操作......652
27.2.2 処理異常の検出......653
27.3 CSV形式データの分解(UNSTRING文)......653
27.3.1 基本操作......653
27.3.2 処理異常の検出......654
27.4 CSV形式のバリエーション......655
27.5 環境変数の設定......656
27.5.1 CBR_CSV_OVERFLOW_MESSAGE(CSV形式データ操作時のメッセージ抑止指定)......656
27.5.2 CBR_CSV_TYPE(生成するCSV形式のバリエーション)......656

付録A 翻訳オプション......658
A.1 翻訳オプション一覧......658
A.2 翻訳オプションの指定形式......659
A.2.1 ALPHAL(英小文字の扱い)......660
A.2.2 BINARY(2進項目の扱い)......660
A.2.3 CHECK(CHECK機能の使用の可否)......661
A.2.4 CODECHK(実行時のコード系チェックの指定)......662
A.2.5 CONF(規格の違いによるメッセージの出力の可否)......663
A.2.6 COPY(登録集原文の表示)......663
A.2.7 COUNT(COUNT機能の使用の可否)......664
A.2.8 CREATE(創成ファイルの指定)......664
A.2.9 CURRENCY(通貨編集用文字の扱い)......664
A.2.10 DLOAD(プログラム構造の指定)......665
A.2.11 DUPCHAR(重複文字の扱い)......665
A.2.12 EQUALS(SORT文での同一キーデータの処理方法)......666
A.2.13 FLAG(診断メッセージのレベル)......666
A.2.14 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)......666
A.2.15 INITVALUE(作業場所節でのVALUE句なし項目の扱い)......667
A.2.16 KANA(文字コードの扱い)......667
A.2.17 LALIGN(連絡節のデータ宣言の扱い)......668
A.2.18 LANGLVL(ANSI COBOL規格の指定)......669
A.2.19 LINECOUNT(翻訳リストの1ページあたりの行数)......669
A.2.20 LINESIZE(翻訳リストの1行あたりの文字数)......670
A.2.21 LIST(目的プログラムリストの出力の可否)......670
A.2.22 MAIN(主プログラム/副プログラムの指定)......670
A.2.23 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)......671
A.2.24 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)......671
A.2.25 MODE(ACCEPT文の動作の指定)......671
A.2.26 NAME(翻訳単位ごとのオブジェクトファイルの出力の可否)......671
A.2.27 NCW(日本語利用者語の文字集合の指定)......672
A.2.28 NSPCOMP(日本語空白の比較方法の指定)......672
A.2.29 NUMBER(ソースプログラムの一連番号領域の指定)......673
A.2.30 OBJECT(目的プログラムの出力の可否)......674
A.2.31 OPTIMIZE(広域最適化の扱い)......674
A.2.32 QUOTE/APOST(表意定数QUOTEの扱い)......675
A.2.33 RSV(予約語の種類)......675
A.2.34 SAI(ソース解析情報ファイルの出力の可否)......676
A.2.35 SDS(符号付き10進項目の符号の整形の可否)......676
A.2.36 SHREXT(マルチスレッドモデルのプログラムの外部属性に関する扱い)......676
A.2.37 SMSIZE(PowerSORTが使用するメモリ容量を指定)......677
A.2.38 SOURCE(ソースプログラムリストの出力の可否)......677
A.2.39 SRF(正書法の種類)......678
A.2.40 SSIN(ACCEPT文のデータの入力先)......678
A.2.41 SSOUT(DISPLAY文のデータの出力先)......679
A.2.42 STD1(英数字の文字の大小順序の指定)......679

A.2.43 TAB（タブの扱い）679
A.2.44 TEST（対話型デバッガの使用の可否）680
A.2.45 THREAD（マルチスレッドモデルのプログラム作成の指定）680
A.2.46 TRACE（TRACE機能の使用の可否）681
A.2.47 TRUNC（桁落とし処理の可否）681
A.2.48 XREF（相互参照リストの出力の可否）682
A.2.49 ZWB（符号付き外部10進項目と英数字項目の比較）682
A.3 プログラム定義にだけ指定可能な翻訳オプション683
A.4 メソッド原型定義と分離されたメソッド定義間での翻訳オプション683

付録B 入出力状態一覧684

付録C 広域最適化688
C.1 最適化の項目688
C.2 共通式の除去688
C.3 不変式の移動689
C.4 誘導変数の最適化689
C.5 PERFORM文の最適化689
C.6 隣接転記の統合690
C.7 無駄な代入の除去690
C.8 広域最適化での注意事項690

付録D 組込み関数の使用692
D.1 関数の型と記述の関係692
D.2 引数の型によって決定される関数の型694
D.3 CURRENT-DATE関数を利用した西暦の取得694
D.4 任意の基準日からの通日計算696
D.5 NATIONAL関数の変換モード696
D.5.1 CBR_FUNCTION_NATIONAL（NATIONAL関数の変換モードの指定）696
D.6 RANDOM関数を利用した疑似乱数列の生成697
D.7 組込み関数一覧698

付録E 環境変数一覧700

付録F 特殊な定数の書き方709
F.1 プログラム名定数709
F.2 原文名定数709
F.3 ファイル識別名定数709
F.4 外部名を指定するための定数709

付録G COBOLが提供するサブルーチン710
G.1 Web連携710
G.2 簡易アプリ間通信機能710
G.3 システム情報を取得するサブルーチン710
G.3.1 プロセスID取得サブルーチン710
G.3.2 スレッドID取得サブルーチン711
G.4 他言語連携で使用するサブルーチン711
G.4.1 実行単位の開始サブルーチン711
G.4.2 実行単位の終了サブルーチン712
G.4.3 実行環境の閉鎖サブルーチン713
G.5 日本語内部表現のコードセット（COBOL独自の16ビットワイドキャラクタ）操作714
G.5.1 mbston16s714
G.5.2 n16stombs716
G.6 ファイルアクセスルーチン718
G.7 メモリ割当てサブルーチン718
G.7.1 COB_ALLOC_MEMORY724
G.7.2 COB_FREE_MEMORY724
G.8 プロセス終了サブルーチン725
G.8.1 COB_EXIT_PROCESS725

付録H オブジェクト指向と従来機能の組合せ....727
H.1 クラス定義で使用できない機能....727
H.2 分離されたメソッド定義で使用できない機能....728

付録I データベース連携....729
I.1 機能概要....729
I.1.1 Oracle連携....729
I.1.2 Symfoware連携....731
I.2 埋込みSQL文のデバッグまでの流れ....731
I.2.1 埋込みSQL文のデバッグ方法....733
I.3 デッドロック出口....734
I.3.1 デッドロック出口スケジュールの概要....734
I.3.2 デッドロック出口スケジュールサブルーチン....734
I.3.3 注意事項....735

付録J 日本語コード系....736
J.1 日本語処理のコード系....736
J.1.1 概要....736
J.1.2 プログラムごとのコード系と実行時のコード系....736
J.1.3 コード系の一致チェック....737
J.2 日本語文字の種類と表現....738
J.2.1 日本語文字の種類....738
J.2.2 日本語文字の表現形式....738
J.3 他システムからの移行上の注意....739
J.3.1 日本語空白と英数字空白の文字コード....739
J.3.2 JIS非漢字の負号について....742
J.3.3 英数字項目のカナ文字の扱い....742

付録K SCCSの利用方法....744
K.1 プログラムの記述方法....744
K.2 履歴情報の参照方法....744

付録L ldコマンド....746
L.1 入力形式....746
L.2 ldコマンドの使い方....747
L.2.1 リンクをcobolコマンドで行う場合との比較....747
L.2.2 プログラム構造ごとのldコマンドの使い方....748

付録M makeコマンドの活用....750
M.1 makeコマンドについて....750
M.2 Makefileの記述方法....750
M.2.1 基本的な記述方法....750
M.2.2 COBOL資源の依存関係....750
M.2.3 クラスが相互に参照している場合の依存関係....751
M.2.4 Makefile作成支援コマンド....753
M.2.5 Makefileのサンプル....753

付録N Windowsを利用したUNIX版COBOLアプリケーションの作成手順....754
N.1 Windowsでの分散開発作業....754
N.2 WindowsからUNIXへのユーザ資産の移行....754
N.3 UNIXでの開発作業....755

付録O OSIV系システムとの機能比較....757

付録P COBOL G移行支援オプション....763

付録Q セキュリティ....764
Q.1 資源の保護....764
Q.2 アプリケーション作成のための指針....764

Q.3 インターネット接続対象外の通信機能....765
Q.4 Web連携機能のセキュリティ....765
Q.5 リモートデバッガ....765

付録R COBOLプログラムの作成技法....766
R.1 効率のよいプログラム....766
R.1.1 一般的な注意....766
R.1.2 データ項目の属性の選択....766
R.1.3 数字転記・数字比較・算術演算....767
R.1.4 英数字転記・英数字比較....768
R.1.5 入出力....768
R.1.6 プログラム間連絡....769
R.1.7 デバッグ....769
R.2 数字項目の標準規則....769
R.2.1 10進項目....769
R.2.2 2進項目....770
R.2.3 浮動小数点項目....770
R.2.4 数字項目の注意事項....771
R.3 注意事項....771

索引....772

第1章 概要

本章では、この製品の機能および動作環境について説明します。この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 機能

ここでは、この製品が持つ機能、この製品が提供する各種ユーティリティについて説明します。

1.1.1 COBOLの機能

この製品は、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
- システムプログラム記述向け(SD)機能
- スクリーン操作機能
- コマンド行引数の操作機能
- 環境変数の操作機能
- 報告書作成機能
- 組込み関数機能
- 浮動小数点数機能
- オブジェクト指向プログラミング機能

また、以下に示すサーバサイドアプリケーション(注)で有効となる機能を提供しています。

- マルチスレッド機能
- COBOL Webサブルーチン

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

1.1.2 この製品が提供するプログラムおよびユーティリティ

この製品は、プログラム開発を行うために、下表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。

表1.1 この製品が提供するプログラムおよびユーティリティ

名称	使用目的
COBOLコンパイラ	COBOLを使って記述したプログラムの翻訳
COBOLランタイムシステム	COBOLアプリケーションの実行
COBOL対話型デバッガ	COBOLアプリケーションのデバッグ
プロジェクトマネージャ	COBOLアプリケーションの開発支援

名称	使用目的
SCREEN-DESIGNER	スクリーン操作機能で使用する画面定義支援
簡易アプリ間通信	アプリケーション間でのデータのやりとり
COBOLファイルユーティリティ	COBOLファイルの処理

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。

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

- 翻訳リストの出力
- 規格仕様のチェック
- 広域最適化
- FORM(画面・帳票定義)との連携

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)を実行する場合に呼び出され、動作します。

COBOL対話型デバッガ

COBOL対話型デバッガは、COBOLアプリケーションをデバッグする場合に使用します。COBOL対話型デバッガは、以下のような機能を提供しています。デバッガの使用方法および機能の詳細については、"第23章 対話型デバッガの使い方"を参照してください。

- プログラムの実行の中断・再開
- データ域の内容の参照・変更
- データ域の値の変更監視
- プログラムの実行経路の表示

プロジェクトマネージャ

COBOLプログラムの翻訳、リンク、デバッグおよび実行を画面を使って会話的に行う機能を提供します。

SCREEN-DESIGNER

COBOLのスクリーン操作機能で使用する画面設計を会話的に行う機能を提供します。

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。

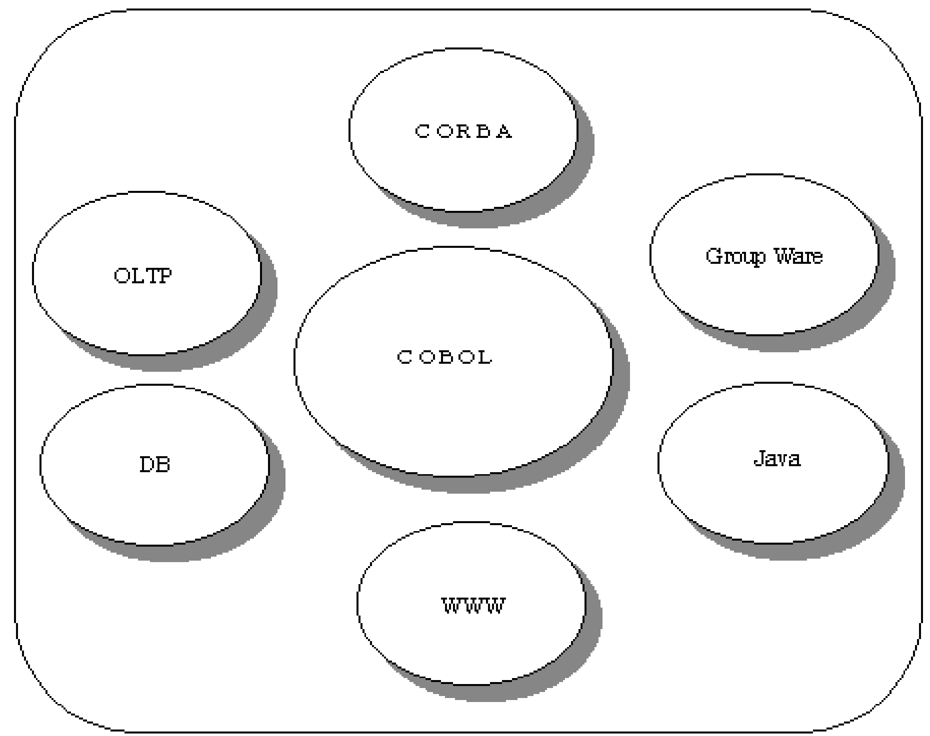
簡易アプリ間通信

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

1.1.3 COBOLと連携機能

この製品で作成したプログラムは、"図1.1 COBOLと関連プロダクト"に示す各種機能と連携して動作することができます。これらの連携機能とCOBOL本来の高信頼性によって、ネットワークインフラを利用した分散環境およびイントラネット環境、OLTPを含むDBアクセス処理に対応した柔軟な基幹業務システムを構築できます。

図1.1 COBOLと関連プロダクト



CORBA

分散運用環境下で動作するプログラムを作成できます。
[関連製品] Interstage

OLTP

オンライントランザクション処理を行うプログラムを作成できます。
[関連製品] Interstage, PowerAIM, Oracle Tuxedo

DB

DBアクセスを行うプログラムを作成できます。
[関連製品] Symfoware, Oracle

WWW

イントラネット環境のプログラムを作成できます。
[関連製品] Interstage

Java

JavaシステムでCOBOL資産を活用できます。
[関連製品] J Business Kit

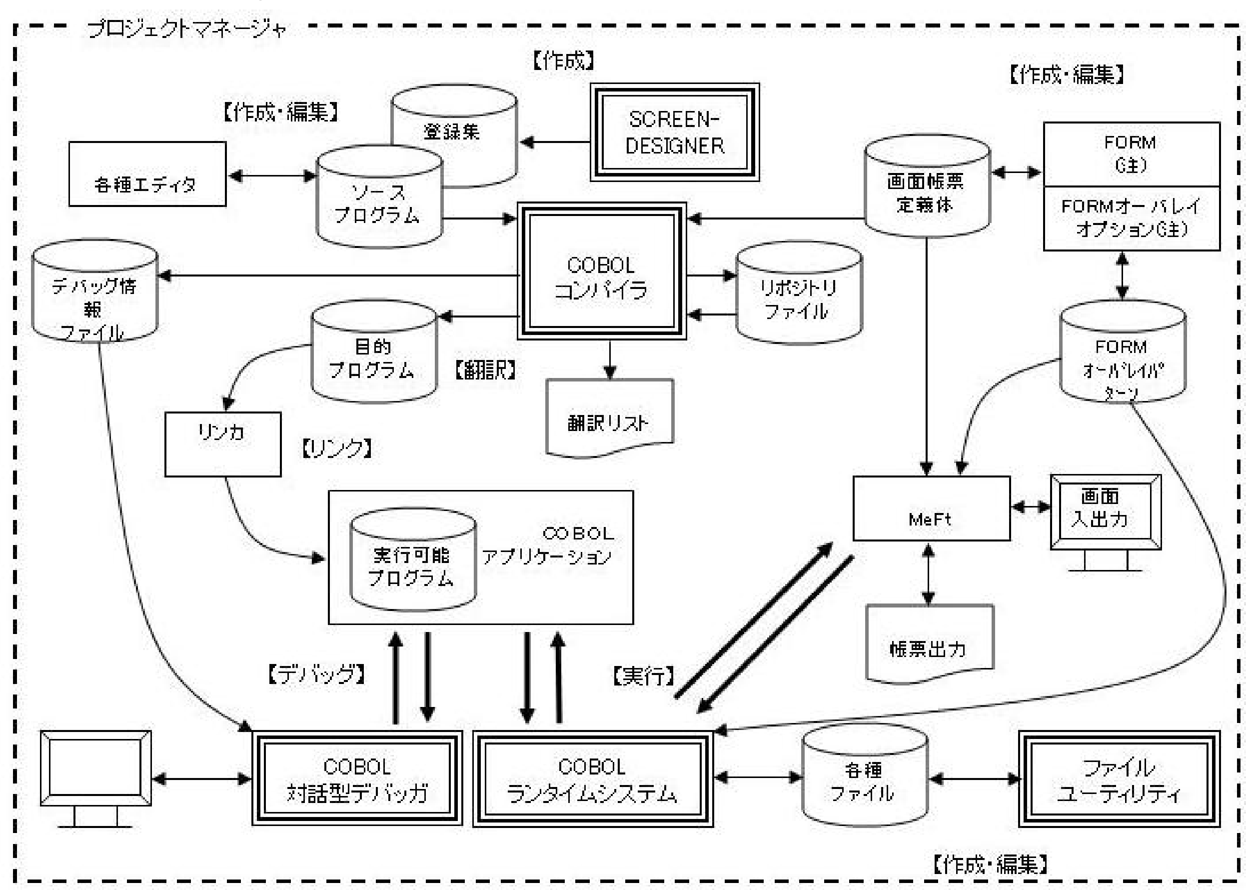
Group Ware

ワークフローを介したプログラムを作成できます。
[関連製品] TeamWARE Flow

1.2 開発環境

この製品を利用したプログラムの開発環境の概要を下図に示します。

図1.2 COBOLの開発環境



注:FORMおよびFORMオーバレイオプションは、Windows製品です。

1.2.1 関連製品

下表にこの製品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。

表1.2 関連製品

製品名	機能
FORM (注)	プログラムが出力する画面や帳票の定義および帳票設計ツール(PowerFORM)の提供
FORMオーバレイオプション (注)	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

注: Windows上で動作する開発系製品です。

FORM

FORMは、COBOLプログラムが表示･印刷する画面や帳票を設計するために使用します。利用者は、FORMを利用して、対話的に画面や帳票のレイアウトなどをデザインすることができます。

また、新しい帳票設計ツール(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ーへの使いやすさを配慮しています。

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使用されます。MeFtは、プログラムが発行する画面や帳票の入出力要求に対して、フォーマット編集を行います。

MeFt/Web

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

1.3 資源一覧

本製品での資源の一覧を下表に示します。

表1.3 資源一覧

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	雛形ファイル名
プロジェクト	プロジェクト情報	プロジェクトマネージャ	任意	prj	―
Make	Makeに関する依存関係などの記述	プロジェクトマネージャ	任意	mk	―
Makeルール	Make規則の記述 Makefileに組み込まれ利用される	プロジェクトマネージャ	任意	―	MakeRule
COBOLソース	COBOLソースプログラム	コンパイラ	任意	cob cobol	―
COBOL登録集	COBOL登録集原文	コンパイラ	任意	cbl	―
画面帳票定義体	画面および帳票の定義情報	FORM MeFt MeFt/Web	任意	smd pmd pxd	―
フォームオーバレイ	フォームオーバレイパターン情報	FORM MeFt	任意	ovd	―
ファイル定義体	ファイルの定義情報	FILE	任意	ffd	―
画面定義情報	スクリーン操作機能用画面の定義情報	スクリーンデザイナ	任意	FRM	―
HTML	ハイパーテキストマークアップ言語での記述情報	webサブルーチン	任意	html htm	―
翻訳オプション	COBOL翻訳オプション文字列	プロジェクトマネージャ	任意	cbi	―
リンクオプション	リンクオプション文字列	プロジェクトマネージャ	任意	lni	―
リポジトリ	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	―
オブジェクト	目的プログラム	コンパイラ	ソース名.o	o	―
共用オブジェクト	副プログラムの共用オブジェクトプログラム	コンパイラ	libライブラリ名.so	so	―
実行形式	実行形式プログラム	コンパイラ	任意 (a.out)	out exe	―
翻訳リスト	翻訳リスト情報	コンパイラ	任意	lst	―

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	雛形ファイル名
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	COBOL.CBR (任意)	CBR	−
キー定義	スクリーン操作機能用キー定義情報	ランタイムシステム	任意	−	keydeffile
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	−	prtinffile
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	−	fonttable
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)	−	FCB1
論理宛先定義	簡易アプリ間通信用の論理宛先定義	ランタイムシステム	任意	−	ciinf
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	任意	−	meftwrc
プリンタ情報	帳票印刷時での各種プリンタ情報	MeFt	任意	−	meftprc
デバッグ情報	対話型デバッガ用のデバッグ情報	デバッガ	ソース名.svd	svd	−
デバッガ動作環境	対話型デバッガの動作環境情報	デバッガ	cobdebug.ini	−	−
履歴 (コマンド)	自動デバッグ、バッチデバッグの入力情報	デバッガ	*.log	log	−
カバレージ	性能向上のためのチューニング情報	デバッガ	*.cvr	cvr	−
テスト進捗	テスト進捗および網羅性の情報	デバッガ	*.smp	smp	−

第2章 プログラムの書き方

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。

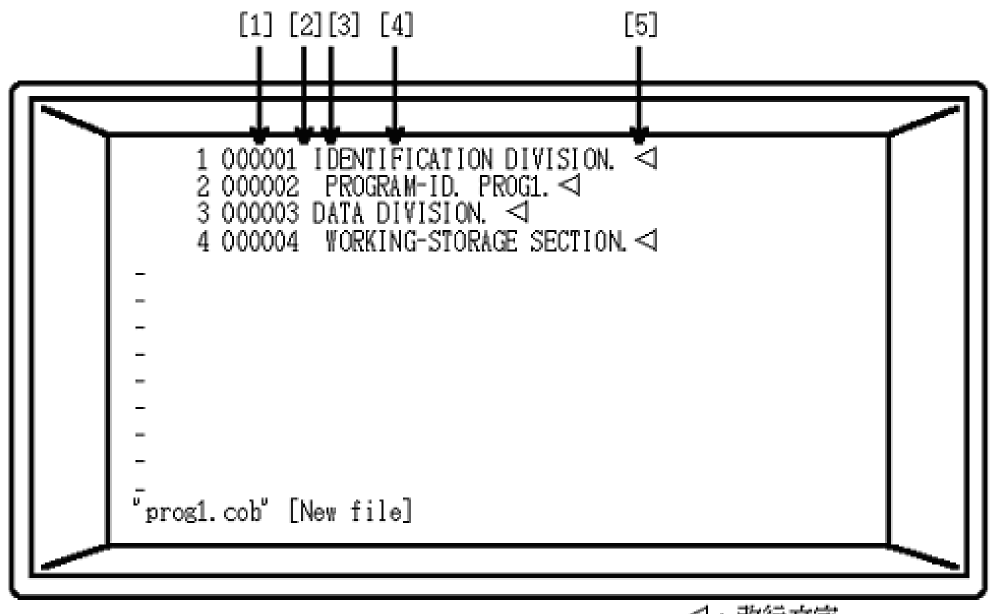
2.1 プログラムの作成と編集

COBOLソースプログラム(以降の説明では、ソースプログラムとよぶ場合があります)および登録集原文は、エディタを使用して作成することができます。

2.1.1 ソースプログラムの作成

ここでは、ソースプログラムを作成・編集する方法を説明します。

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に入力する必要があります。行番号やCOBOLの文および手続き名などは、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、行番号を指定します。行番号を指定しない(空白のままとする)ことも可能です。

[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。

[3] A領域(カラム8～11)

カラム8～11をA領域と呼びます。COBOLの部、節、段落およびプログラムの終わり見出しなどは、この領域から書きます。レベル番号が77や01のデータ項目もこの領域から書き始めます。

[4] B領域(カラム12以降)

カラム12以降をB領域と呼びます。通常、COBOLの文、注記項およびレベル番号が77や01でないデータ項目などは、この領域から書き始めます。

[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。

参考

ソースプログラムの文字定数には、TAB文字(ASCII X”09”)を指定することもできます。TAB文字は、文字定数中で1バイトを占めます。

2.1.2 登録集原文の作成

COBOLの原始文操作機能では、COPY文を使って、ソースプログラム中に登録集原文を取り込むことができます。登録集原文は、SCREEN-DESIGNERなどのユーティリティを利用したり、ソースプログラムを作成する場合と同様にviなどのエディタを利用したりして、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ソースプログラムの形式と同一にする必要はありません。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、ソースプログラムと同様、翻訳オプションで指定します。登録集原文を格納するファイル名は、通常、登録集原文名.cblにします。

参考

画面帳票定義体およびファイル定義体は、正書法のどの形式としても利用できます。

2.2 プログラムの形式

COBOLソースプログラムの各行は、改行文字で区切られます。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する必要があります。正書法には、固定形式、可変形式および自由形式の3つの形式があり、固定形式または自由形式を使用する場合は、翻訳時に翻訳オプションSRFで指定する必要があります。

参照

“A.2.39 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、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書きます。
以下に固定形式の正書法を示します。

可変形式

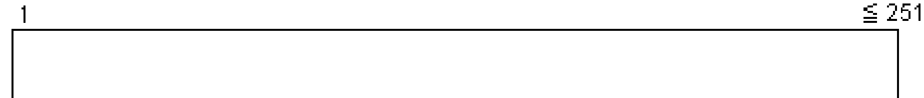
可変形式では、ソースプログラムの1行の長さを、251バイト以下の任意のバイト数で書くことができます。
以下に可変形式の正書法を示します。

（カラム位置）

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、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）



2.3 翻訳指示文

この製品は、翻訳指示文を使って、ソースプログラム中に翻訳オプションを記述することができます。

以下に、翻訳指示文の記述形式を示します。

```
@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.
000300  PROGRAM-ID. PROG1.
     :
008000 END PROGRAM PROG1.
```

注意

- @OPTIONS翻訳指示文の分離符としてタブ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、“付録A 翻訳オプション”を参照してください。

第3章 プログラムの翻訳とリンク

本章では、プログラムを翻訳・リンクし、実行可能プログラムを作成する方法について説明します。

3.1 翻訳とリンク

ソースプログラムは、翻訳およびリンクを行い、実行可能プログラムにします。ここでは、翻訳時に設定する環境変数、ソースプログラムの翻訳・リンクの基本操作、副プログラムのリンク方法およびCOBOLコンパイラが使用する資源について説明します。

3.1.1 翻訳時に設定する環境変数

毎回指定するオプションやファイル名などは、環境変数として設定することができます。なお、以下に示す環境変数は、ソースプログラムを翻訳する前に設定します。

- "3.1.1.1 COBOLOPTS（翻訳オプションの指定）"
- "3.1.1.2 COBCOPY（登録集ファイルの格納ディレクトリの指定）"
- "3.1.1.3 COB_COPYNAME（登録集原文の検索条件の指定）"
- "3.1.1.4 COB_LIBSUFFIX（登録集ファイル名の拡張子の指定）"
- "3.1.1.5 SMED_SUFFIX（画面帳票定義体ファイル名の拡張子の指定）"
- "3.1.1.6 FORMLIB（画面帳票定義体ファイルの格納ディレクトリの指定）"
- "3.1.1.7 FFD_SUFFIX（ファイル定義体ファイル名の拡張子の指定）"
- "3.1.1.8 FILELIB（ファイル定義体ファイルの格納ディレクトリの指定）"
- "3.1.1.9 COB_REPIN（リポジトリファイルの入力先ディレクトリの指定）"

3.1.1.1 COBOLOPTS（翻訳オプションの指定）

cobolコマンドのオプションを指定します。

3.1.1.2 COBCOPY（登録集ファイルの格納ディレクトリの指定）

登録集ファイルの格納ディレクトリ名を指定します。

ディレクトリを複数指定する場合、ディレクトリをコロンで区切って指定します。この場合、指定された順序でディレクトリが検索されます。

[関連オプション]"3.3.1.12 -I（登録集ファイルのディレクトリの指定）"

3.1.1.3 COB_COPYNAME（登録集原文の検索条件の指定）

登録集原文、ファイル定義体および画面帳票定義体を格納したファイルのファイル名を検索する方法を指定します。

ソースプログラム中のCOPY文に記述されたファイル名を、拡張子を含めてすべて英大文字または英小文字として検索を行いたい場合に、以下のどれかを指定します。

Upper	すべて大文字のファイル名を検索します。
Lower	すべて小文字のファイル名を検索します。
Default	"登録集原文名.英小文字の拡張子"の形式のファイル名を検索します。登録集原文名の英小文字/英大文字の扱いは、翻訳オプションALPHALの指定に依存します。

環境変数が指定されていない場合は、"Default"が指定されたものとみなされます。

なお、ソースプログラム中のCOPY文に定数指定でファイル名を記述した場合は、環境変数COB_COPYNAMEの指定は無視され、記述どおりの文字列で検索されます。

[関連オプション]"A.2.1 ALPHAL（英小文字の扱い）"

3.1.1.4 COB_LIBSUFFIX(登録集ファイル名の拡張子の指定)

登録集ファイル名の拡張子として設定する任意の文字列を指定します。

拡張子を複数指定する場合、拡張子をコンマで区切って指定します。この場合、指定された拡張子の順序でファイルが検索されます。

3.1.1.5 SMED_SUFFIX(画面帳票定義体ファイル名の拡張子の指定)

画面帳票定義体ファイル名の拡張子として設定する任意の文字列を指定します。

拡張子を複数指定する場合、拡張子をコンマで区切って指定します。この場合、指定された拡張子の順序でファイルが検索されます。

3.1.1.6 FORMLIB(画面帳票定義体ファイルの格納ディレクトリの指定)

画面帳票定義体ファイルの格納ディレクトリ名を指定します。

ディレクトリを複数指定する場合、ディレクトリをコロンで区切って指定します。この場合、指定された順序でディレクトリが検索されます。

[関連オプション]"3.3.1.15 -m (画面帳票定義体ファイルのディレクトリの指定)"

3.1.1.7 FFD_SUFFIX(ファイル定義体ファイル名の拡張子の指定)

ファイル定義体ファイル名の拡張子として設定する任意の文字列を指定します。

拡張子を複数指定する場合、拡張子をコンマで区切って指定します。この場合、指定された拡張子の順序でファイルが検索されます。

3.1.1.8 FILELIB(ファイル定義体ファイルの格納ディレクトリの指定)

ファイル定義体ファイルの格納ディレクトリ名を指定します。

ディレクトリを複数指定する場合、ディレクトリをコロンで区切って指定します。この場合、指定された順序でディレクトリが検索されます。

[関連オプション]"3.3.1.11 -f (ファイル定義体ファイルのディレクトリの指定)"

3.1.1.9 COB_REPIN(リポジトリファイルの入力先ディレクトリの指定)

リポジトリファイルの入力先ディレクトリ名を指定します。

ディレクトリを複数指定する場合、ディレクトリをコロンで区切って指定します。この場合、指定された順序でディレクトリが検索されます。

[関連オプション]"3.3.1.17 -R (リポジトリファイルの入力先ディレクトリの指定)"

3.1.2 基本操作

ソースプログラムを翻訳・リンクして実行可能プログラムを作成するには、次の方法があります。

- cobolコマンドで翻訳からリンクまでを一度に行う方法
- cobolコマンドで翻訳を行い、cobolコマンドでリンクを行う方法
- cobolコマンドで翻訳を行い、ldコマンドでリンクを行う方法
- プロジェクトマネージャを使う方法

ここでcobolコマンドおよびldコマンドで翻訳およびリンクを行う場合にmakeコマンドを使用するとさらに効率よく実行可能プログラムを作成することができます。makeコマンドについては、"付録M makeコマンドの活用"を参照してください。

以下に、cobolコマンドおよびldコマンドで翻訳およびリンクを行う方法について説明します。プロジェクトマネージャを使う方法については、"第22章 プロジェクトマネージャの使い方"を参照してください。

なお、cobolコマンドを使用する場合、以下の環境変数にCOBOLコンパイラの格納ディレクトリを設定しておく必要があります。

- PATH
- LD_LIBRARY_PATH

cobolコマンドで翻訳からリンクまでを一度に行う方法

cobolコマンドには、ソースプログラムを翻訳して再配置可能プログラムを生成し、生成した再配置可能プログラムを結合して実行可能プログラムを生成する機能があります。そのため、ldコマンドを使わずにcobolコマンドを実行するだけで、実行可能プログラムを作り出すことができます。さらに、利用者は、この製品が提供する各種ライブラリサブルーチンのリンクを意識する必要がないという利点もあります。

以下に、cobolコマンドを使って翻訳とリンクを行うときの例を示します。cobolコマンドのパラメタ形式については、"3.3 cobolコマンド"を参照してください。

```
$ cobol -dy -M -o P1 P1.cob
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

```
入力　　　:P1.cob(ソースファイル)
出力　　　:P1.o(オブジェクトファイル)　P1(実行可能ファイル)
オプション:-dy (動的結合の指定)　-M(主プログラムの指定)　-o(実行可能プログラムの出力先)
```

cobolコマンドで翻訳を行い、cobolコマンドでリンクを行う方法

cobolコマンドには、ソースプログラムの翻訳だけを行い、実行可能プログラムを生成しないで、再配置可能プログラムだけを生成する機能があります。また、翻訳を行わず、再配置可能プログラムおよび共用オブジェクトプログラムのリンクを行い、実行可能プログラムを生成する機能もあります。cobolコマンドを使ってリンクを行う場合、利用者は、cobolコマンドで翻訳・リンクを行う場合と同様に、この製品が提供する各種ライブラリサブルーチンを個々に指定する必要はありません。

以下に、cobolコマンドを使って翻訳を行い、cobolコマンドを使ってリンクを行うときの例を示します。

```
$ cobol -c -M P1.cob　　　←　(翻訳)
最大重大度コードはIで,翻訳したプログラム数は1本です.
$ cobol -dy -o P1 P1.o　　←　(リンク)
```

cobol コマンド(翻訳)

```
入力　　　:P1.cob(ソースファイル)
出力　　　:P1.o(オブジェクトファイル)
オプション:-M(主プログラムの指定)
　　　　　 -c(翻訳だけ行う指定)
```

cobol コマンド(リンク)

```
入力　　　:P1.o(オブジェクトファイル)
出力　　　:P1(実行可能ファイル)
オプション:-dy (動的結合の指定)
　　　　　 -o(実行可能プログラムの出力先)
```

cobolコマンドで翻訳を行い、ldコマンドでリンクを行う方法

cobolコマンドで作成した再配置可能プログラムを、ldコマンドを使って実行可能プログラムを生成することもできます。ldコマンドの使い方については、"付録L ldコマンド"を参照してください。

翻訳エラー発生時の注意点

- I, WおよびEレベルの翻訳エラー発生時にも、目的プログラムは作成されます。翻訳終了時には、エラーメッセージの内容を確認してください。
- Make実行時に、Makefile中に記述されたcobolコマンドがEレベル以上の翻訳エラーを発生した場合、Makeはその復帰コードからMakeの続行が不可能であると判断しエラー終了します。このため、目的プログラムが作成されない場合があるので注意してください。

3.1.3 登録集(COPY文)を使ったプログラムの翻訳方法

ここでは、COPY文を記述したソースプログラムを翻訳する方法について説明します。

COPY文を記述したソースプログラムを翻訳するときには、COPY文によって取り込む登録集原文を格納したファイル(以降の説明では登録集ファイルといいます)が必要となります。登録集ファイルのファイル名は、COPY文に指定した原文名または原文名定数によって決定されます。

COPY文に原文名を記述した場合、または原文名定数を相対パス名で記述した場合には、登録集ファイルの格納されているディレクトリをCOBOLコンパイラに指示する必要があります。

以下に、登録集ファイルの格納されているディレクトリの指定方法を、優先順位の高い順に示します。

ソースプログラムに、IN/OFなしのCOPY文を記述した場合

1. cobolコマンドのオペランドに登録集ファイルの格納されているディレクトリを指定した-Iオプションを指定します。
2. 環境変数COBCOPYに登録集ファイルの格納されているディレクトリを設定します。COBCOPYの詳細については、“3.1.1.2 COBCOPY（登録集ファイルの格納ディレクトリの指定）”を参照してください。

ソースプログラムに、IN/OFありのCOPY文を記述した場合

IN/OFで指定した登録集名を環境変数名とした環境変数に、登録集ファイルの格納されているディレクトリを設定します。この設定がない場合、翻訳時に翻訳エラーとなります。

環境変数COB_COPYNAMEを使用すると、登録集ファイル、画面帳票定義体およびファイル定義体ファイルの検索時の大文字、小文字、または従来どおりの指定を行うことができます。環境変数そのものが設定されていない場合は、従来どおり(Default)と同じです。COB_COPYNAMEの詳細については、“3.1.1.3 COB_COPYNAME（登録集原文の検索条件の指定）”を参照してください。

以下に、登録集を使ったプログラムを翻訳するときのcobolコマンドの実行例を示します。

ソースファイルと登録集ファイルが同じディレクトリに存在する場合

ソースプログラムの記述が COPY A.のとき

```
$ cobol -dy -M -o P1 P1.cob
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

```
入力      : P1.cob（ソースファイル）
            A.cbl （登録集ファイル）
出力      : P1.o（オブジェクトファイル）
            P1（実行可能ファイル）
オプション : -dy （動的結合の指定）
            -M（主プログラムの指定）
            -o（実行可能プログラムの出力先）
```

ソースファイルと登録集ファイルが異なるディレクトリに存在する場合

ソースプログラムの記述が COPY A.のとき

```
$ cobol -dy -M -o P1 -I/home/COBOL P1.cob
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

または、

```
$ COBCOPY=/home/COBOL ; export COBCOPY
$ cobol -dy -M -o P1 P1.cob
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

```
入力      : P1.cob（ソースファイル）
            /home/COBOL/A.cbl （登録集ファイル）
出力      : P1.o（オブジェクトファイル）
            P1（実行可能ファイル）
オプション : -dy （動的結合の指定）
            -M（主プログラムの指定）
            -o（実行可能プログラムの出力先）
            -I（登録集ファイルの入力先）
```

ソースプログラムの記述が COPY A OF B. の場合

```
$ B=/home/COBOL ; export B
$ cobol -M -o P1 P1.cob
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

```
入力      : P1.cob(ソースファイル)
            /home/COBOL/A.cbl (登録集ファイル)
出力      : P1.o(オブジェクトファイル)
            P1(実行可能ファイル)
オプション : -M(主プログラムの指定)
            -o(実行可能プログラムの出力先)
```

注意

ソースプログラムに英小文字の登録集原文名を記述した場合、翻訳オプションALPHAL/NOALPHALの指定によって取り込まれるファイルが異なります。

[例] ソースプログラムの記述がCOPY a.のとき

- 翻訳オプションALPHALが指定されている場合、取り込まれるファイル名はA.cbl
- 翻訳オプションNOALPHALが指定されている場合、取り込まれるファイル名はa.cbl

3.1.4 副プログラムを呼び出すプログラムの翻訳・リンク方法

ここでは、副プログラムを呼び出すプログラムと呼び出される副プログラムを翻訳・リンクして、実行可能プログラムを作成する方法について説明します。なお、以降の説明では副プログラムを呼び出すプログラムを主プログラムといい、呼び出される副プログラムを副プログラムといいます。

プログラムの結合の種類およびリンク方法の詳細については、"3.2.2 結合の種類とプログラム構造"を参照してください。

主プログラムを翻訳・リンクし、実行可能ファイルを作成するときには、まず、副プログラムを翻訳・リンクし、共用オブジェクトプログラムを作成しておく必要があります。共用オブジェクトプログラムは、主プログラムのリンクを行うときに必要となります。

以下に、静的結合の場合と動的結合の場合の実行可能プログラムの作成方法の例を説明します。

P1,P2,P3の3つのソースプログラムを翻訳・リンクします。なお、プログラム間の呼出し関係は、次のとおりです。

- P1は、P2とP3を呼び出します。
- P2,P3は、ほかのプログラムを呼び出しません。

静的結合の場合

```
$ cobol -c P2.cob P3.cob                                    [1]
最大重大度コードはI
最大重大度コードはI
最大重大度コードはIで,翻訳したプログラム数は2本です.
$ cobol -dn -M -o P1 P1.cob P2.o P3.o                       [2]
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

[1] 副プログラムを翻訳し、再配置可能プログラムを作成します。

```
入力      : P2.cob  P3.cob(ソースファイル)
出力      : P2.o  P3.o(オブジェクトファイル)
オプション : -c(翻訳だけ行う指定)
```

[2] 主プログラムを翻訳・リンクし、実行可能プログラムを作成します。

```
入力      : P1.cob(ソースファイル)
            P2.o  P3.o(オブジェクトファイル)
```

出力　　　：P1(実行可能ファイル)
オプション：-dn (静的結合の指定)
　　　　　　-M(主プログラムの指定)
　　　　　　-o(実行可能プログラムの出力先)

動的結合の場合

```
$ cobol -dy -G -o libP2.so P2.cob                                  [1]
最大重大度コードはIで,翻訳したプログラム数は1本です.
$ cobol -dy -G -o libP3.so P3.cob                                  [2]
最大重大度コードはIで,翻訳したプログラム数は1本です.
$ cobol -dy -M -o P1 -lP2 -lP3 P1.cob                              [3]
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

[1] [2]　副プログラムを翻訳･リンクし、共用オブジェクトファイルを作成します。

入力　　　：P2.cob　P3.cob(ソースファイル)
出力　　　：P2.o　P3.o(オブジェクトファイル)
　　　　　　libP2.so　libP3.so(共用オブジェクトファイル)
オプション：-dy (動的結合の指定)
　　　　　　-G(共用オブジェクトプログラムを作成する指定)
　　　　　　-o(共用オブジェクトプログラムの出力先)

[3]　主プログラムを翻訳･リンクし、実行可能ファイルを作成します。

入力　　　：P1.cob(ソースファイル)
　　　　　　libP2.so　libP3.so(共用オブジェクトファイル)
出力　　　：P1(実行可能ファイル)
オプション：-dy (動的結合の指定)
　　　　　　-M(主プログラムの指定)
　　　　　　-o(実行可能プログラムの出力先)
　　　　　　-l(リンクするライブラリ)

3.1.5 対話型デバッガを使う場合のプログラムの翻訳方法

対話型デバッガを使う場合、翻訳時およびリンク時に-Dtオプションを指定します。-Dtオプションを指定してプログラムを翻訳すると、デバッグ情報ファイルが作成されます。デバッグ情報ファイルは、対話型デバッガでプログラムをデバッグするときに必要です。

通常、デバッグ情報ファイルは、ソースファイルが格納されているディレクトリに格納されます。-ddオプションで格納先ディレクトリを指定することもできます。

デバッグ情報ファイルのファイル名は、ソースファイル名をピリオドで区切って、拡張子svdを付加した名前になります。

```
$ cobol -M -o P1 -Dt P1.cob
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

入力　　　：P1.cob(ソースファイル)
出力　　　：P1.o(オブジェクトファイル)
　　　　　　P1(実行可能ファイル)
　　　　　　P1.svd(デバッグ情報ファイル)
オプション：-M(主プログラムの指定)
　　　　　　-o(実行可能プログラムの出力先)
　　　　　　-Dt (対話型デバッガを使う指定)

3.1.6 COBOLコンパイラが使用するファイル

COBOLコンパイラは、以下のファイルを使用します。

なお、この製品が提供する各種ユーティリティを使用する場合には、“表3.1 cobolコマンドで使用するファイル”に書かれた推奨の拡張子を使用してください。

- ソースファイル(*.cob)
- オブジェクトファイル(*.o)

- 登録集ファイル(*.cbl)
- 画面帳票定義体ファイル(*.smd/*.pmd/*.pxd)
- ファイル定義体ファイル(*.ffd)
- 翻訳リストファイル(任意/*.lst)
- リポジトリファイル(*.rep)
- ソース解析情報ファイル(*.sai)
- 共用オブジェクトファイル(lib*.so)
- 実行可能ファイル(任意/a.out)
- デバッグ情報ファイル(*.svd)
- オプションファイル(任意)

表3.1 cobolコマンドで使用するファイル

ファイルの種類	ファイルの内容	ファイル名の形式	入出力	条件	関連オプション
ソースファイル	ソースプログラム	プログラム名.cob (注1)	入力	必須	―
オブジェクトファイル	目的プログラム(再配置可能プログラム)	ソースファイル名.o (注2)	入力	リンクを行う場合	―
			出力	翻訳が成功した場合に生成	-do
登録集ファイル	登録集原文	登録集原文名.cbl (注3)	入力	登録集原文を使ったソースプログラムを翻訳する場合	-I
画面帳票定義体ファイル	画面帳票定義体	画面帳票定義体名.smd 帳票定義体名.pmd 帳票定義体名.pxd (注4)	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	-m -pm
ファイル定義体ファイル	ファイル定義体	ファイル定義体名.ffd (注5)	入力	ファイル定義体を使ったソースプログラムを翻訳する場合	-f
翻訳リストファイル	翻訳リスト	任意 ソースファイル名.lst (注6)	出力	翻訳リストをファイルに出力	-P
リポジトリファイル	継承および適合チェックのためのクラス関連情報	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
			出力	クラス定義のソースプログラムを翻訳する場合	-dr
ソース解析情報ファイル	ソースをSIMPLIAなどで解析するための情報	ソースファイル名.sai	出力	ソース解析をする場合	-ds
共用オブジェクトファイル	副プログラムの共用オブジェクトプログラム	lib副プログラム名.so (注7)	入力	リンクを行う場合	-l
			出力	-Gオプションを指定してリンクを行った場合に生成	-G -o
実行可能ファイル	実行可能プログラム	任意 (省略時はa.out)	出力	-Gオプションを指定しないでリンクを行った場合に生成	-o
デバッグ情報ファイル	対話型デバッガ用デバッグ情報	ソースファイル名.svd (注8)	出力	-Dt オプションを指定して翻訳を行った場合に生成	-Dt -dd TEST
オプションファイル	翻訳オプションを示す文字列	任意	入力	翻訳オプションをファイルに格納して指定する場合	-i

- 注1
ファイル名は、任意の名前を使用できます。ただし、この製品は以下の拡張子を持つファイルを出力するため、ソースファイル名にこれらの拡張子を使用しないようにしてください。また、大文字のCOB,CBL,COBOLも拡張子として扱いません。
  - lst
  - rep
  - sai
- 注2
ソースファイルの名前が、拡張子cob,cblまたはcobolで終わる場合、拡張子cob,cblまたはcobolを拡張子oに変更したファイル名となります。それ以外の場合にはソースファイル名に拡張子oを付加したファイル名となります。
- 注3
拡張子cblは、環境変数COB_LIBSUFFIXを使って任意の文字列に変更することができます。文字列は以下のように設定します。なお、文字列Noneを設定した場合、拡張子なしとして扱われます。環境変数COB_LIBSUFFIXが指定されていない場合、拡張子がcbl,cob,cobolの順で登録集ファイルを検索します。

  [参照]"3.1.1.4 COB_LIBSUFFIX(登録集ファイル名の拡張子の指定)"

```
$ COB_LIBSUFFIX=文字列 ; export COB_LIBSUFFIX
```

- 注4
拡張子smdは、環境変数SMED_SUFFIXを使って任意の文字列に変更することができます。文字列は以下のように設定します。なお、文字列Noneを設定した場合、拡張子なしとして扱われます。環境変数SMED_SUFFIXが指定されていない場合、拡張子がpmd, pxd,smdの順で画面帳票定義体ファイルを検索します。

  [参照]"3.1.1.5 SMED_SUFFIX(画面帳票定義体ファイル名の拡張子の指定)"

```
$ SMED_SUFFIX=文字列 ; export SMED_SUFFIX
```

- 注5
拡張子ffdは、環境変数FFD_SUFFIXを使って任意の文字列に変更することができます。文字列は以下のように設定します。なお、文字列Noneを設定した場合、拡張子なしとして扱われます。

  [参照]"3.1.1.7 FFD_SUFFIX(ファイル定義体ファイル名の拡張子の指定)"

```
$ FFD_SUFFIX=文字列 ; export FFD_SUFFIX
```

- 注6
-Pオプションでファイル名の省略(-)を指定した場合、翻訳リストファイル名の形式は以下のようになります。
  - ソースファイルの名前が、拡張子cob,cblまたはcobolで終わる場合、拡張子cob,cblまたはcobolを拡張子lstに変更したファイル名となります。
  - 上記以外の場合にはソースファイル名に拡張子lstを付加したファイル名となります。
- 注7
リンク時に利用者が明に指定する必要があります。
- 注8
ソースファイルの名前が、拡張子cob,cblまたはcobolで終わる場合、拡張子cob,cblまたはcobolを拡張子svdに変更したファイル名となります。それ以外の場合にはソースファイル名に拡張子svdを付加したファイル名となります。

3.1.7 COBOLコンパイラが出力する情報

COBOLコンパイラが出力する情報には、以下があります。

- 診断メッセージ
- オプション情報リスト
- 翻訳単位統計情報リスト
- 相互参照リスト

- ソースプログラムリスト
- 目的プログラムリスト
- データエリアに関するリスト

3.1.7.1 診断メッセージ

COBOLコンパイラは、プログラムの翻訳結果を診断メッセージとして通知します。診断メッセージは、通常標準エラー出力先に出力されます。また、-Pオプションで出力先ファイル名を指定することもできます。

```
$ cobol -M -o P1 -PP1.lst P1.cob
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

```
入力：　P1.cob(ソースファイル)
出力：　P1.o(オブジェクトファイル)　P1(実行可能ファイル)
　　　　P1.lst (翻訳リストファイル)
```

診断メッセージリスト

```
＊＊　診断メッセージ　＊＊　(SAMPLE1)

JMN2503I-S　63　利用者語'A'が定義されていません.
```

3.1.7.2 オプション情報リスト、翻訳単位統計情報リスト

cobolコマンド実行時に有効となっている翻訳オプションを知りたい場合、-WCオプションで翻訳オプションMESSAGEを指定します。翻訳オプションMESSAGEを指定すると、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。これらのリストは通常標準エラー出力先に出力されます。また、-Pオプションで出力先ファイル名を指定することもできます。

```
$ cobol -M -o sample1 -Psample1.lst -WC,"MESSAGE,LINESIZE(80)" sample1.cob
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

```
入力：　sample1.cob (ソースファイル)
出力：　sample1.o (オブジェクトファイル)　sample1 (実行可能ファイル)
　　　　sample1.lst(翻訳リストファイル)
```

オプション情報リスト

```
＊＊　指定翻訳オプション　＊＊

MAIN,MESSAGE,LINESIZE(80)

＊＊　確定翻訳オプション　＊＊

  ALPHAL(ALL)              LANGLVL(85)              SDS
  BINARY(WORD,MLBON)       LINECOUNT(0)           NOSHREXT
NOCHECK                    LINESIZE(80)             SMSIZE(0)
  CODECHK                NOLIST                   NOSOURCE
NOCONF                     MAIN                     SRF(VAR,VAR)
NOCOPY                   NOMAP                      SSIN(SYSIN)
NOCOUNT                    MESSAGE                  SSOUT(SYSOUT)
  CREATE(OBJ)              MODE(STD)                STD1(JIS2)
  CURRENCY(¥)            NONAME                     TAB(8)
NODLOAD                    NCW(STD)               NOTEST
  DUPCHAR(STD)             NSPCOMP(NSP)             THREAD(SINGLE)
NOEQUALS                 NONUMBER                 NOTRACE
  FLAG(I)                  OBJECT                 NOTRUNC
NOFLAGSW                   OPTIMIZE               NOXREF
NOINITVALUE                QUOTE                    ZWB
  KANA(EUC)                RSV(ALL)
NOLALIGN                 NOSAI
```

翻訳単位統計情報リスト

```
**  プログラム特性リスト  **

   ファイル名 = sample1.cob
   翻訳日付   = 2007年09月27日(木) 12時58分13秒 (GMT+9.00)

     原始プログラムのレコード数                      =          65 レコード
     目的プログラムの大きさ(. ｔｅｘｔサイズ)        =        1692 バイト
     目的プログラムの大きさ(. ｄａｔａサイズ)        =         456 バイト
     制御レベル                                      =        0101 レベル
     翻訳に要した時間                                =        0.17 秒

  最大重大度コード = I
```

3.1.7.3 相互参照リスト

翻訳オプションXREFを指定すると、翻訳リストファイルに相互参照リストが出力されます。

相互参照リストの形式

```
      定義行  名標                      属性と参照行

         [1]  [2]                       [3]
           2  SAMPLE1
          44  データ入力
          38  繰り返し回数              53S 54R 55R 62R
          36  先頭文字                  55R
          35  単語                      62R
          50  単語の検索
          60  単語の表示
           6  単語一覧                  34D
```

[1] 定義行の行番号

行番号が次の形式で表示されます。

```
[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られます。

行番号

名前の定義された行番号が表示されます。暗黙定義されたものは、“*”が表示されます。

別翻訳単位定義記号

名前が別の翻訳単位中で定義されている場合、行番号に“#”が付加されて表示されます。

[2] 名標

ソースプログラムで定義されている名標が表示されます。名標が表示される領域は、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が表示されます。参照形態は、以下の記号で表示されます。

- 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

[参照]“A.2.48 XREF（相互参照リストの出力の可否）”

3.1.7.4 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。

ソースプログラムリストの出力形式

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

[1] 行番号

－ 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文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

ソースプログラムで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。

－ 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文に対して、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文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文による組み込み表示(“C”で表示)を行います。

[2] ソースプログラム

3.1.7.5 目的プログラムリスト

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

目的プログラムリストの出力形式

```
BEGINNING OF NONDECLARATIVE PROCEDURES
                                             BBK=00004(001)
                                             ............LX
                                             .........PN.PX
  [1]       [2]                         [4]
00000030 00000000
00000034 0000020C
0000020C 9207E018          add     %i7,+0x018,%o1
00000210 D227E014          st      %o1,[%i7+0x014]
00000214            GLB.3:   [3]
--- 12 ---              MOVE   [5]
00000214 3B00000C          sethi   %hi(0x00003000),%i5
00000218 BA176130          or      %i5,0x0130,%i5
0000021C FA37E054          sth     %i5,[%i7+0x054] :被除数
--- 13 ---              MOVE
00000220 3900000C          sethi   %hi(0x00003000),%i4
00000224 B8172030          or      %i4,0x0030,%i4
00000228 F837E052          sth     %i4,[%i7+0x052] :除数
--- 15 ---              COMPUTE
<SIMPLE STORE>
<REDUNDANT STORE>
<SIMPLE STORE>
<REDUNDANT STORE>
<REDUNDANT STORE>
<REDUNDANT STORE>
--- 17 ---              EXIT PROGRAM
0000022C 7FFFFFE9          call    GLB.1
00000230 01000000          nop
                                             BBK=00005(000)
                                             NEVER EXECUTED
                                             ............PX
END OF LISTDBG1
```

[1] オフセット

機械語の文のオブジェクト内相対オフセットを表しています。

[2] 機械語の命令コード

機械語の文(オブジェクトコード)を表しています。

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

[3] 手続き名と手続き番号

コンパイラが生成した手続き名と手続き番号を表しています。

[4] アセンブラ形式の命令

機械語の文をSPARCのアセンブリ言語に準じた形式で表しています。

[5] 動詞名と行番号

COBOLプログラムプログラム中に記述された動詞名と行番号を表しています。

3.1.7.6 データエリアに関するリスト

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

データエリアに関するリストには、以下があります。

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

データマップリストの出力形式

```
   [1]  [2]                   [3]  [4]    [5]         [6]     [7]     [8]    [9]
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
[10]**MAIN**
 15  l7+000000A8                  0 FD OUTFILE            LSAM       BG2.000000
 16 [l7+00000080]+00000000        0 01 印刷レコード     60 ALPHANUM   BVA.000002
 18  i7+00000068                  0 77 答え              8 EXT-DEC    BGW.000000
 19  i7+00000070                  0 77 除数              4 EXT-DEC    BGW.000000
 21  l6+00000018                  0 01 CSTART            8 ALPHANUM   BC0.000000
 22  l6+00000020                  0 01 CEND              8 ALPHANUM   BC0.000000
 23  l6+00000028                  0 01 CRES              8 ALPHANUM   BC0.000000
 25 [l7+0000007C]+00000000        0 01 被除数            2 EXT-DEC    BVA.000001
[11]**SUB1**
 48  i7+000000B0                  0 01 表示              8 EXT-DEC    BGW.000000
```

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については、"3.1.7.4 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。

```
レジスタ+相対アドレス
```

レジスタ

以下のいずれかを表示します。

- %l6 (.rodata)
- %i7 (.data)
- %l7 (.bss)
- %o6 (スタック)
- %i5 (ヒープ)
- %i4 (ヒープ)

相対アドレス

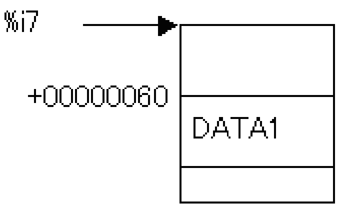
レジスタが示す位置からの相対アドレスを表示します。

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。

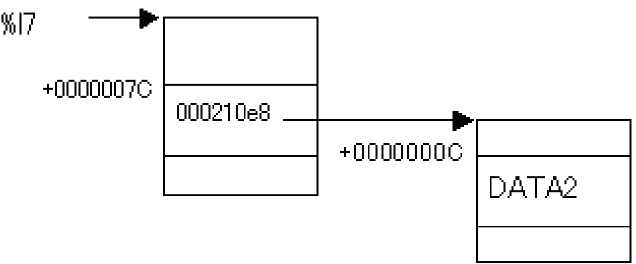
－ オフセットが表示されない場合(DATA1: i7+00000060)

%i7に格納されたアドレスに0x60を足した位置に、DATA1のデータ領域があることを示します。



－ オフセットが表示される場合(DATA2: [l7+0000007C]+0000000C)

%i7に格納されたアドレスに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
INDEX-NAME	指標名
INT-BOOLE	内部ブール
EXT-BOOLE	外部ブール
NATIONAL	日本語
NAT-EDIT	日本語編集
OBJ-REF	オブジェクト参照データ
POINTER	ポインタデータ

FD項目の場合は、ファイル種別と呼出し法を以下の記号で表示します。

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

[8] 基点

データ項目が割り付けられるベースレジスタとベース位置を表示します。

[9] 次元数

添字または指標付けの次元数を表示します。

[10]、[11] プログラム名

プログラムが入れ子の場合の区切りとしてプログラム名を表示します。

ただし、クラス定義の場合は、各定義の区切りを以下の形式で表示します。

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

プログラム制御情報リストの出力形式

```
    番地                フィールド名                長さ(10)
[1]
    ** GWA **
      * GCB *
  i7+00000000         GCB FIXED AREA                    8

      * GMB *
  i7+0000007C         GMB POINTERS AREA                36
  i7+0000007C           VNAL                           36
```

```
  ........          MUTEX HANDLE AREA                0
  ........          FMBE                             0
  ........          BEA                              0
  ........          BVA                              0
  ........          VPA                              0
  ........          PSA                              0
  ........          CONTROL ADDRESS TABLE            0
i7+00000008       LCB AREA                         116
  ........        TRG AREA                           0
  ........        TSG AREA                           0
  ........        LIA FIXED AREA                     0
  ........        IWA1 AREA                          0
  ........          ALTINX                           0
  ........          USESAVE                          0
  ........          PCT                              0
  ........          CONTENT                          0
  ........          USEOSAVE                         0
  ........          LINAGE COUNTER                   0
  ........          OTHER AREA                       0
i7+00000094         ENTSAVE                          4
  ........        GMB CONTROL BLOCKS AREA            0
  ........          FMB1                             0
  ........          DDCB                             0
  ........          SMBO                             0
  ........          SPECIAL REGISTERS                0
  ........          S-A LIST                         0
  ........          DPA                              0
  ........          DMA                              0
  ........          SCREEN CONTROL AREA              0
  ........          STRONG TYPE AREA                 0
  ........        FAI AREA                           0
  ........          ATTR INFO FIXED AREA             0
  ........          REUSE INFO TABLE                 0
  ........          METHOD TABLE                     0
  ........          INTERFACE TABLE                  0
  ........          METHOD COPY TABLE                0
  ........        IAI AREA                           0
  ........          ATTR INFO FIXED AREA             0
  ........          REUSE INFO TABLE                 0
  ........          METHOD TABLE                     0
  ........          INTERFACE TABLE                  0
  ........          METHOD COPY TABLE                0
  ........        CFOR                               0
  ........        INVOKE PARAM INFO                  0
  ........        AS MODIFY INFO                     0

   * GWS *
i7+000000A0       DATA AREA                         18

 ** COA **
   * CCB *
l6+00000000       CCB FIXED AREA                     4
  ........        STANDARD CONSTANT AREA             0

   * CMB *
l6+00000004       CMB POINTERS AREA                  8
  ........          BEAI                             0
l6+00000004         BVAI                             8
  ........          IPA                              0
  ........          FARA                             0
l6+00000010       LITERAL AREA                       8
l6+00000018       CONSTANT SECTION DATA             24
l6+00000030       CMB CONTROL BLOCKS AREA          688
```

```
I6+00000030         FMB2                         148
  ........            FMBI                           0
  ........            SMB1                           0
  ........            SMB2                           0
  ........            EDT                            0
  ........            EFT                            0
I6+000000C4           FMB2 ADDR-PRM LIST             8
  ........            ERROR POROCEDURE               0
  ........            CALL PARM ADD INFO             0
  ........            CALL PARM INFO                 0
  ........            ALPHABETIC NAME                0
  ........            CLASS NAME                     0
  ........            CLASS TEST TABLE               0
  ........            TRANS-TABLE PROTO              0
  ........            CIPB                           0
  ........            DOG TABLE                      0
  ........            EXTERNAL DATA NAME             0
  ........            NATIONAL-MSG F-NAM             0
  ........            FLOW BLOCK INFO                0
  ........            SPECIAL REGISTERS              0
I6+000000CC           EPA CONTROL AREA             532
  ........            SIGN TABLE                     0
  ........            LIBRARY ADDR TABLE             0
  ........            SCREEN CONTROL AREA            0
  ........            EXCEPTION PROC LIST            0
  ........          FCM FMB1 OFFSET LIST             0
  ........          FCM OBJ-REF OFFSET LIST          0
  ........          ICM FMB1 OFFSET LIST             0
  ........          ICM OBJ-REF OFFSET LIST          0
  ........          MCM AREA/OBJ-REF LIST            0
  ........          CFOR OFFSET LIST                 0
  ........          CLASS NAME INFO                  0
  ........          AS MODIFY/PARAM INFO             0
  ........          METHOD NAME INFO                 0
  ........          CALL PARM INFO LIST              0

 ** GW2 **
   * GC2 *
I7+00000000         GMB2 POINTERS AREA               4

   * GM2 *
I7+00000004         LIA AREA                       120
I7+0000007C         GMB2 POINTERS AREA              24
  ........            FMBE                           0
  ........            BEA                            0
I7+0000007C           BVA                            8
  ........            VPA                            0
  ........            PSA                            0
I7+00000084           CONTROL AREA ADDR TBL         16
  ........            MUTEX HANDLE AREA              0
I7+00000094         IWA AREA                         4
  ........            ALTINX                         0
  ........            USESAVE                        0
  ........            PCT                            0
  ........            CONTENT                        0
  ........            USEOSAVE                       0
  ........            LINAGE COUNTER                 0
  ........            OTHER AREA                     0
I7+00000094           ENTSAVE                        4
I7+00000098         TRG AREA                        12
  ........          TSG AREA                         0
I7+000000A8         GMB2 CONTROL BLOCKS            208
I7+000000A8           FMB1                         208
```

```
   ........          DDCB                        0
   ........          SMBO                        0
   ........          SPECIAL REGISTERS           0
   ........          S-A LIST                    0
   ........          DPA                         0
   ........          DSA                         0
   ........          DMA1                        0
   ........          METHOD INIT INFO            0
   ........         CFOR                         0

    * GS2 *
   ........         DATA AREA                    0
    * TL2 *
   ........         TL 2ND AREA                  0
  ** STK **
    * SCB *
 o6+000000D0        SCB FIXED AREA              56
 o6+000000A0        TL 1ST AREA                 48
 o6+00000000        ABIA                        92
   ........         SGM POINTERS AREA            0
   ........          VPA                         0
   ........          PSA                         0
   ........          BVA                         0
   ........          BOD                         0
 o6+00000060        IWA3 AREA                   12

   ........          CONTENT                     0
   ........          RTNAREA                     0
   ........          USESAVE                     0
   ........          USEOSAVE                    0
   ........          OTHER AREA                  0
   ........          ENTSAVE                     0
 o6+00000060         PARM                        8
 o6+00000068         RTNADDR                     4
   ........          CALL PARM INFO              0
   ........         TRG AREA                     0
   ........         TSG AREA                     0
   ........         SOR AREA                     0
 o6+00000070        LIA VARIABLE AREA           48
   ........         LCB AREA                     0
   ........         SGM CONTROL BLOCKS AREA      0
   ........          FMB1                        0
   ........          SMBO                        0
   ........          SPECIAL REGISTER            0
   ........          DPA                         0
   ........          DMA                         0
   ........          MIA                         0

   ........         DATA AREA                    0

   ........         TL 2ND AREA                  0
 o6+00000108        LIA ADDRESS                 12
     ＊＊  定数領域  ＊＊
[2]
 番地               0 . . .  4 . . .  8 . . .  C . . .   0123456789ABCDEF

 l6+00000010        00000001 00000008                    ........
```

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

[2] 目的プログラム中に存在する定数領域を表示します。

セクションサイズリストの出力形式

```
      ＊＊  目的プログラムの大きさ  ＊＊
[1]
   ．ｔｅｘｔサイズ ＝      2284バイト
   ．ｄａｔａサイズ ＝       184バイト

      ＊＊  実行に必要な領域の大きさ  ＊＊
[2]
   ．ｂｓｓサイズ  ＝       376バイト
   ヒープサイズ   ＝         0バイト
   スタックサイズ  ＝       272バイト
```

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

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

```
      ＊＊  実行に必要な領域の大きさ  ＊＊

クラス名
    ．ｂｓｓサイズ  ＝       272バイト
    ヒープサイズ   ＝         0バイト

メソッド名
    スタックサイズ  ＝       384バイト                [3]
              :
```

[3] スタックサイズは、メソッド定義ごとに表示します。

3.2 実行可能プログラムの構造

ここでは、以下について説明します。

- 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の構造
- 結合の種類
- リンクによって生成される実行可能プログラムの構造
- 実行可能プログラムを作成するためのリンク方法

3.2.1 概要

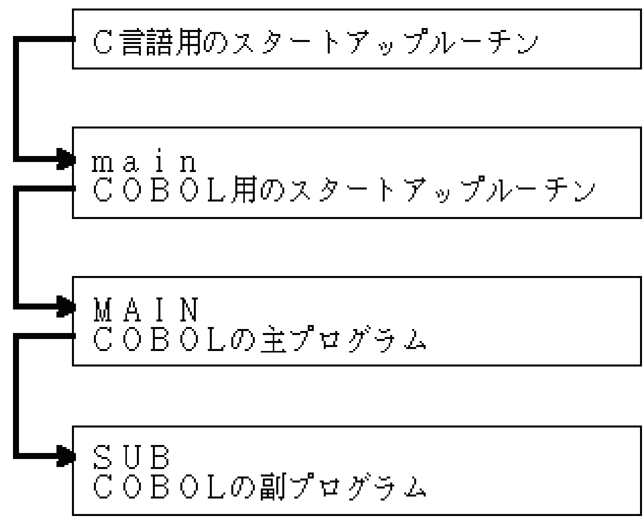
この製品の実行可能プログラムは、COBOLコンパイラによって生成された再配置可能プログラムに、以下のプログラムをリンクしたものです。

- C言語用のスタートアップルーチン
- COBOL用のスタートアップルーチン
- COBOL用のランタイムライブラリサブルーチン
- C言語のランタイムライブラリサブルーチン

cobolコマンドでリンクを行う場合には、これらのプログラムが自動的にリンクされます。しかし、ldコマンドでリンクを行う場合には、利用者がldコマンドに指定する必要があります。

リンクの完了した実行可能プログラムの構造の例を下図に示します。

図3.1 実行可能プログラムの構造



3.2.2 結合の種類とプログラム構造

結合の種類には、静的結合と動的結合があります。

静的結合

リンク時に、呼ぶプログラムと呼ばれるプログラムがすべて結合される方法です。

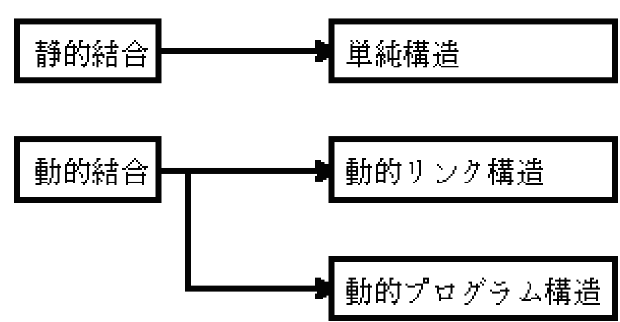
動的結合

実行時に、呼ぶプログラムに呼ばれるプログラムが結合される方法です。

それぞれのリンク方法によって作られるプログラム構造を"図3.2 結合の種類とプログラムの構造"に示し、各プログラム構造について説明します。

なお、説明中で、主プログラムとは、最初に動作するプログラム、副プログラムとは、主プログラムまたは副プログラムから呼ばれるプログラムをいいます。

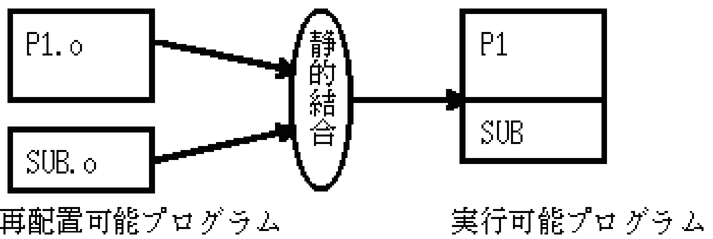
図3.2 結合の種類とプログラムの構造



単純構造

単純構造とは、1つ以上の再配置可能プログラムを静的結合によって1つの実行可能プログラムにしたものです。したがって、実行開始時に、主プログラムと副プログラムのすべてが仮想記憶上にローディングされ、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。

以下に単純構造の概要を示します。



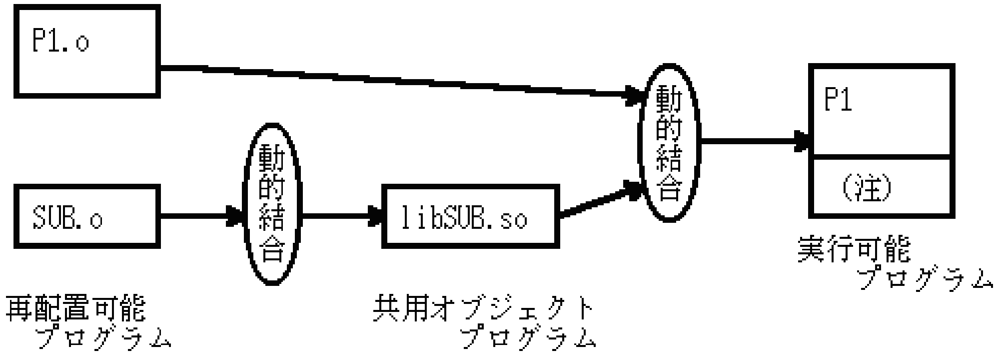
動的リンク構造

動的リンク構造とは、主プログラムの再配置可能プログラムと、副プログラムの共用オブジェクトプログラムを動的結合することによって、実行可能プログラムにしたものです。

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結合されません。副プログラムが必要になった時点でダイナミックリンカによって仮想記憶上にローディングされます。

ローディングは、動的結合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および環境変数LD_LIBRARY_PATHに設定されているパスリストを使って、システムのダイナミックリンカが行います。リンク後に副プログラムの格納ディレクトリを移動する場合には、環境変数LD_LIBRARY_PATHに移動後のパスを設定しておく必要があります。

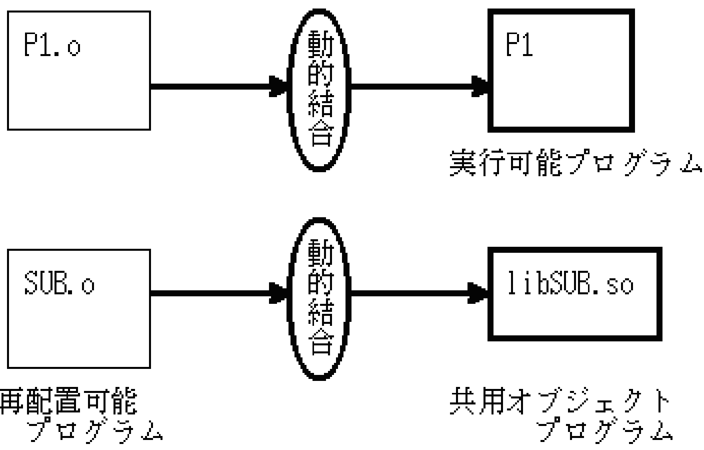
以下に動的リンク構造の概要を示します。



注) ダイナミックリンカが使用する副プログラムの情報

動的プログラム構造

動的プログラム構造では、動的結合によって、主プログラムの再配置可能プログラムだけを実行可能プログラムとします。そのため、動的リンク構造と違って、実行可能プログラム中にダイナミックリンカの副プログラム情報を含みません。副プログラムの共用オブジェクトプログラムは、実行時に初めて必要となります。動的プログラム構造では、副プログラムのローディングは、その副プログラムが呼ばれるときに、呼ぶプログラムがCOBOLのランタイムシステムに依頼して行われます。このとき、システムのローディング機能が用いられます。以下に動的プログラム構造の概要を示します。



プログラムの構造とCALL文の関係

プログラムの構造は、CALL文の書き方、翻訳時に指定するオプションおよび結合の種類で決定されます。“表3.2 プログラム構造とCALL文/翻訳オプションの種類の関係”に、プログラム構造とCALL文、翻訳オプションおよび結合の種類の関係を示します。なお、翻訳オプションDLOADについては、“付録A 翻訳オプション”を参照してください。

表3.2 プログラム構造とCALL文/翻訳オプションの種類の関係

		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共用オブジェクトプログラムが必要です。

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。

動的プログラム構造では、通常、エントリ情報が必要となります。エントリ情報の詳細については“4.1.4 副プログラムのエントリ情報”を参照してください。

プログラム構造と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。

“表3.3 プログラム構造とCANCEL文の関係”に、プログラム構造とCANCEL文の関係を示します。

表3.3 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、“9.2.7 内部プログラム”を参照してください。

3.2.3 単純構造の実行可能プログラムの作成方法

単純構造の実行可能プログラムの作成方法の例を以下に示します。

P1,P2,P3の3つのソースプログラムを翻訳・リンクします。なお、プログラム間の呼出し関係は、次のとおりです。

- P1は、P2とP3を呼び出します。
- P2,P3は、ほかのプログラムを呼び出しません。

cobolコマンドだけで実行可能プログラムを作成する方法

```
$ cobol -c P2.cob P3.cob                                        [1]
最大重大度コードはI
最大重大度コードはI
最大重大度コードはIで,翻訳したプログラム数は2本です.
$ cobol -dn -M -o P1 P1.cob P2.o P3.o                           [2]
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

[1] 副プログラムを翻訳し、再配置可能プログラムを作成します。

```
入力        : P2.cob
              P3.cob(ソースファイル)
出力        : P2.o
              P3.o(オブジェクトファイル)
オプション : -c(翻訳だけ行う指定)
```

[2] 主プログラムを翻訳・リンクし、実行可能プログラムを作成します。

```
入力        : P1.cob(ソースファイル)
              P2.o  P3.o(オブジェクトファイル)
出力        : P1(実行可能ファイル)
オプション : -dn (静的結合の指定)
              -M(主プログラムの指定)
              -o(実行可能プログラムの出力先)
```

副プログラムをアーカイブライブラリサブルーチンとしてリンクする方法

```
$ cobol -c P2.cob P3.cob                                        [1]
最大重大度コードはI
最大重大度コードはI
最大重大度コードはIで,翻訳したプログラム数は2本です.
$ ar r libP0.a P2.o P3.o                                        [2]
$ cobol -dn -M -o P1 -lP0 P1.cob                                [3]
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

[1] 副プログラムを翻訳し、再配置可能プログラムを作成します。

```
入力        : P2.cob
              P3.cob(ソースファイル)
出力        : P2.o  P3.o(オブジェクトファイル)
オプション : -c(翻訳だけ行う指定)
```

[2] 副プログラムのアーカイブライブラリサブルーチンを作成します。

```
入力        : P2.o  P3.o(オブジェクトファイル)
出力        : libP0.a (アーカイブライブラリサブルーチン)
オプション : r (アーカイブライブラリサブルーチンの指定)
```

[3] 主プログラムを翻訳・リンクし、実行可能プログラムを作成します。

```
入力        : P1.cob(ソースファイル)
              libP0.a(アーカイブライブラリサブルーチン)
出力        : P1(実行可能ファイル)
オプション : -dn (静的結合の指定)
              -M(主プログラムの指定)
```

-o(実行可能プログラムの出力先)
-l(リンクするライブラリ)

リンクをldコマンドで行う方法

リンクをldコマンドで行う方法については、“L.2.2 プログラム構造ごとのldコマンドの使い方”を参照してください。

3.2.4 動的リンク構造の実行可能プログラムの作成方法

動的リンク構造の実行可能プログラムの作成方法の例を以下に示します。

P1,P2,P3の3つのソースプログラムを翻訳・リンクします。なお、プログラム間の呼出し関係は、次のとおりです。

- P1はP2とP3を呼び出します。
- P2,P3は、ほかのプログラムを呼び出しません。

P2,P3を個別の共用オブジェクトプログラムとして作成する場合

```
$ cobol -dy -G -o libP2.so P2.cob                    [1]
最大重大度コードはIで,翻訳したプログラム数は1本です.
$ cobol -dy -G -o libP3.so P3.cob                    [2]
最大重大度コードはIで,翻訳したプログラム数は1本です.
$ cobol -dy -M -o P1 -lP2 -lP3 P1.cob                [3]
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

[1] 副プログラムを翻訳・リンクし、共用オブジェクトプログラムを作成します。

入力 : P2.cob(ソースファイル)
出力 : libP2.so(共用オブジェクトファイル)

[2] 副プログラムを翻訳・リンクし、共用オブジェクトプログラムを作成します。

入力 : P3.cob(ソースファイル)
出力 : libP3.so(共用オブジェクトファイル)

[3] 主プログラムを翻訳・リンクし、実行可能プログラムを作成します。

入力 : P1.cob(ソースファイル)
　　　libP2.so　libP3.so(共用オブジェクトファイル)
出力 : P1(実行可能ファイル)

P2,P3を1つの共用オブジェクトプログラムとして作成する場合

```
$ cobol -dy -G -o libP0.so P2.cob P3.cob             [1]
最大重大度コードはI
最大重大度コードはI
最大重大度コードはIで,翻訳したプログラム数は2本です.
$ cobol -dy -M -o P1 -lP0 P1.cob                     [2]
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

[1] 副プログラムを翻訳・リンクし、共用オブジェクトプログラムを作成します。

入力 : P2.cob　P3.cob(ソースファイル)
出力 : libP0.so(共用オブジェクトファイル)

[2] 主プログラムを翻訳・リンクし、実行可能プログラムを作成します。

入力 : P1.cob(ソースファイル)
　　　libP0.so(共用オブジェクトファイル)
出力 : P1(実行可能ファイル)

リンクをldコマンドで行う方法

リンクをldコマンドで行う方法については、“L.2.2 プログラム構造ごとのldコマンドの使い方”を参照してください。

3.2.5 動的プログラム構造の実行可能プログラムの作成方法

動的プログラム構造の実行可能プログラムの作成方法の例を以下に示します。

P1,P2,P3の3つのソースプログラムを翻訳・リンクします。なお、プログラム間の呼出し関係は、次のとおりです。

- P1はP2とP3を呼び出します。
- P2,P3は、ほかのプログラムを呼び出しません。

動的プログラム構造では、副プログラムのローディングは、COBOLランタイムシステムが行います。このとき、COBOLランタイムシステムは、ローディングする共用オブジェクトプログラムを特定するために、エントリ情報の指定が必要になります。ただし、以下の名前の共用オブジェクトプログラムの場合、エントリ情報を指定する必要がありません。エントリ情報の詳細については、"4.1.4 副プログラムのエントリ情報"を参照してください。

```
libCALL文で指定されたプログラム名.so
```

P2,P3を個別の共用オブジェクトプログラムとして作成する場合

```
$ cobol -dy -G -o libP2.so P2.cob                                    [1]
最大重大度コードはIで,翻訳したプログラム数は1本です.
$ cobol -dy -G -o libP3.so P3.cob                                    [2]
最大重大度コードはIで,翻訳したプログラム数は1本です.
$ cobol -dy -M -o P1 -WC,"DLOAD" P1.cob                              [3]
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

[1] 副プログラムを翻訳・リンクし、共用オブジェクトプログラムを作成します。

```
入力：P2.cob(ソースファイル)
出力：libP2.so(共用オブジェクトファイル)
```

[2] 副プログラムを翻訳・リンクし、共用オブジェクトプログラムを作成します。

```
入力：P3.cob(ソースファイル)
出力：libP3.so(共用オブジェクトファイル)
```

[3] 主プログラムを翻訳・リンクし、実行可能プログラムを作成します。

```
入力：P1.cob(ソースファイル)
出力：P1(実行可能ファイル)
```

リンクをldコマンドで行う方法

リンクをldコマンドで行う方法については、"L.2.2 プログラム構造ごとのldコマンドの使い方"を参照してください。

動的プログラム構造の注意事項

動的プログラム構造では、以下のどちらかを指定しなければ、1つの共用オブジェクトプログラムとして作成されている複数の副プログラムを呼び出すことはできません。したがって、この例の場合、P2とP3は別々の共用オブジェクトプログラムとして作成する必要があります。

1つの共用オブジェクトプログラムとして作成されている複数のプログラムを呼び出したい場合、次の2つの方法があります。

- 環境変数情報CBR_ENTRYFILEにエントリ情報ファイル名を指定する。
- 実行可能プログラムに呼び出す共用オブジェクトを-lオプションでリンクする。ただし、この方法で呼び出されたプログラムに対するCANCEL文は無効になるため注意が必要です。

動的プログラム構造で副プログラムを呼び出す場合の注意事項については、"第9章 サブプログラムを呼び出す～プログラム間連絡機能～"を参照してください。

3.3 cobolコマンド

cobolコマンドは、ソースプログラムを翻訳・リンクし、再配置可能プログラム、共用オブジェクトプログラムおよび実行可能プログラムを作成するときに使用します。以下に、cobolコマンドの入力形式を説明します。

```
$ cobol [翻訳に関するオプションおよびリンクに関するオプションの並び]  ファイル名...
```

オプションおよびオペランドの説明

コマンド、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。空白の代わりにTABを用いることもできます。

オプションは、環境変数COBOLOPTSに指定しておくこともできます。毎回指定するオプションをCOBOLOPTSに指定しておくことにより、cobolコマンドに対しての指定を省略することができます。

```
$ COBOLOPTS="-Dt -WC,LINESIZE(80),MESSAGE" ; export COBOLOPTS
$ cobol -M p1.cob
```

上の例は、以下のcobolコマンドと等価です。

```
$ cobol -Dt -WC,"LINESIZE(80),MESSAGE" -M p1.cob
```

以降の説明のディレクトリ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。

翻訳に関するオプション

COBOLコンパイラに通知する各種情報を指定します。指定する内容については、"3.3.1 翻訳に関するオプション"を参照してください。

リンクに関するオプション

リンカに通知する各種情報を指定します。指定する内容については、"3.3.2 リンクに関するオプション"を参照してください。

ファイル名

ソースプログラムが格納されているファイル(ソースファイル)のパス名、または再配置可能プログラムの格納されているファイル(オブジェクトファイル)のパス名を指定します。ファイルは複数個指定することができます。

再配置可能プログラムおよび共用オブジェクトプログラムに対しては、翻訳処理は行われず、リンク処理だけ行われます。

注意事項

cobolコマンドに指定するパラメタ(翻訳に関するオプション、リンクに関するオプションおよびファイル名)の総数が4000個を超える場合、cobolコマンドが異常終了することがあります。この場合、cobolコマンドに指定するパラメタの総数を4000個以内に収めてください。

3.3.1 翻訳に関するオプション

以下にcobolコマンドの翻訳に関するオプションを示します。

翻訳の資源に関するもの

- "3.3.1.5 -dr (リポジトリファイルの入出力先ディレクトリの指定)"
- "3.3.1.11 -f (ファイル定義体ファイルのディレクトリの指定)"
- "3.3.1.12 -I (登録集ファイルのディレクトリの指定)"
- "3.3.1.15 -m (画面帳票定義体ファイルのディレクトリの指定)"
- "3.3.1.17 -R (リポジトリファイルの入力先ディレクトリの指定)"
- "3.3.1.6 -ds (ソース解析情報ファイルの出力ディレクトリの指定)"

翻訳リストに関するもの

- "3.3.1.10 -dp (翻訳リストファイルのディレクトリの指定)"
- "3.3.1.16 -P (翻訳リストのファイル名の指定)"

目的プログラムの作成に関するもの

- "3.3.1.9 -do (オブジェクトファイルのディレクトリの指定)"
- "3.3.1.14 -M (主プログラムを翻訳するときの指定)"
- "3.3.1.18 -Tm (マルチスレッドモデルのプログラムを翻訳するときの指定)"

実行時のデバッグ機能に関するもの

- “3.3.1.2 -Dc（COUNT機能 を使用する指定）”
- “3.3.1.8 -dd（デバッグ情報ファイルのディレクトリの指定）”
- “3.3.1.3 -Dk（CHECK機能を使用する指定）”
- “3.3.1.4 -Dr（TRACE機能を使用する指定）”
- “3.3.1.7 -Dt（対話型デバッガを使用する指定）”

その他

- “3.3.1.1 -c（翻訳だけを行う指定）”
- “3.3.1.13 -i（オプションファイルの指定）”
- “3.3.1.19 -WC（翻訳オプションの指定）”

ポイント

翻訳に関するオプションで、同じオプションを複数個指定した場合、複数個指定についての説明が特にないときには、最後に指定したオプションが有効になります。

3.3.1.1 -c（翻訳だけを行う指定）

リンクは行わず、翻訳だけを行う場合に指定します。

```
-c
```

3.3.1.2 -Dc（COUNT機能 を使用する指定）

```
-Dc
```

COUNT機能を使用する場合、-Dcオプションを指定します。COUNT機能については、“5.4 COUNT機能の使い方”を参照してください。

注意

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

参考

-Dcオプションは、翻訳オプションCOUNTと同じ意味です。

その他の情報については、“A.2.7 COUNT（COUNT機能の使用の可否）”を参照してください。

3.3.1.3 -Dk（CHECK機能を使用する指定）

```
-Dk
```

CHECK機能を使用する場合に指定します。CHECK機能については、“5.3 CHECK機能の使い方”を参照してください。なお、-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。したがって、本番運用時には-Dkオプションを指定しないことをおすすめします。

参考

-Dkオプションは、翻訳オプションCHECKと同じ意味です。

その他の情報については、“A.2.3 CHECK（CHECK機能の使用の可否）”を参照してください。

3.3.1.4 -Dr（TRACE機能を使用する指定）

```
-Dr
```

TRACE機能を使用する場合に指定します。TRACE機能については、“5.2 TRACE機能の使い方”を参照してください。なお、-Drオプションを指定すると、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。したがって、本番運用時には、-Drオプションを指定しないことをおすすめします。

参考

-Drオプションは、翻訳オプションTRACEと同じ意味です。

その他の情報については、“A.2.46 TRACE（TRACE機能の使用の可否）”を参照してください。

3.3.1.5 -dr（リポジトリファイルの入出力先ディレクトリの指定）

```
-dr ディレクトリ名
```

リポジトリファイルを格納するディレクトリを変更したい場合、-drオプションにディレクトリを指定します。

-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ディレクトリに作成されます。

また、-drオプションで指定されたディレクトリは、外部リポジトリを取り込む場合の入力先ディレクトリとしても使用されます。

3.3.1.6 -ds（ソース解析情報ファイルの出力ディレクトリの指定）

```
-ds
```

ソース解析情報ファイルを格納するディレクトリを変更したい場合、-dsオプションにディレクトリを指定します。

-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.2.34 SAI（ソース解析情報ファイルの出力の可否）”

3.3.1.7 -Dt（対話型デバッガを使用する指定）

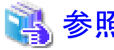
```
-Dt
```

対話型デバッガを使用する場合に指定します。対話型デバッガについては、“第23章 対話型デバッガの使い方”を参照してください。

参考

-Dtオプションは、翻訳オプションTESTと同じ意味です。

その他の情報は、“第23章 対話型デバッガの使い方”を参照してください。

 参照

"3.3.1.8 -dd（デバッグ情報ファイルのディレクトリの指定）"

"A.2.44 TEST（対話型デバッガの使用の可否）"

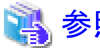
3.3.1.8 -dd（デバッグ情報ファイルのディレクトリの指定）

```
-dd ディレクトリ名
```

デバッグ情報ファイルを格納するディレクトリを指定します。

-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

"3.3.1.7 -Dt（対話型デバッガを使用する指定）"

"A.2.44 TEST（対話型デバッガの使用の可否）"

3.3.1.9 -do（オブジェクトファイルのディレクトリの指定）

```
-do ディレクトリ名
```

オブジェクトファイルの格納先を変更する場合、-doオプションにディレクトリを指定します。

-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.2.30 OBJECT（目的プログラムの出力の可否）"

3.3.1.10 -dp（翻訳リストファイルのディレクトリの指定）

```
-dp ディレクトリ名
```

翻訳リストファイルを格納するディレクトリを変更したい場合、-dpオプションにディレクトリを指定します。

-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を指定した場合、翻訳リストファイル名は、"-dpオプションで指定したディレクトリ名"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

  -Pオプションにファイル名を指定した場合

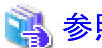
  ```
  cobol -P out.lst -dp /tmp/test cobtest.cob  → /tmp/test/out.lst
  ```

- -Pオプションにハイフン(-)を指定した場合、翻訳リストファイル名は、"-dpオプションで指定したディレクトリ名"に"ソースファイル名.lst"を結合した名前になります。

-Pオプションにハイフン(-)を指定した場合

```
cobol -P- -dp /tmp/test cobtest.cob  → /tmp/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います。



“3.3.1.16 -P (翻訳リストのファイル名の指定)”

3.3.1.11 -f (ファイル定義体ファイルのディレクトリの指定)

```
-f ディレクトリ名
```

IN/OF XFDLIB指定のCOPY文によりファイル定義体からレコード定義文を取り込む場合、ファイル定義体ファイルが存在するディレクトリを指定します。-fオプションを複数個指定すると、ファイル定義体ファイルは指定された順番に検索されます。指定されたディレクトリに検索対象のファイル定義体ファイルが存在しない場合は、カレントディレクトリが検索されます。

3.3.1.12 -I (登録集ファイルのディレクトリの指定)

```
-I ディレクトリ名
```

ソースプログラムでCOPY文を使用している場合、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ディレクトリを指定します。-Iオプションを複数個指定すると、登録集は指定された順番に検索されます。指定されたディレクトリに検索対象の登録集ファイルが存在しない場合は、カレントディレクトリが検索されます。

3.3.1.13 -i (オプションファイルの指定)

```
-i ファイル名
```

翻訳オプションをオプションファイル(翻訳オプションの文字列が格納されているファイル)で指定する場合、オプションファイル名を指定します。オプションファイルはテキストエディタを使用して作成します。オプションファイルの内容は、-WCオプションで指定する翻訳オプション列と同じです。以下にオプションファイルの内容の例を示します。

例

```
MESSAGE,NUMBER,OPTIMIZE
```

3.3.1.14 -M (主プログラムを翻訳するときの指定)

```
-M
```

実行時に主プログラムとなるソースプログラムの翻訳を行う場合に指定します。

参考

-Mオプションは、翻訳オプションMAINと同じ意味です。

その他の情報については、“A.2.22 MAIN(主プログラム/副プログラムの指定)”を参照してください。

3.3.1.15 -m (画面帳票定義体ファイルのディレクトリの指定)

```
-m ディレクトリ名
```

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文を取り込む場合、画面帳票定義体ファイルが存在するディレクトリを指定します。-mオプションを複数個指定すると、画面帳票定義体ファイルは指定された順番に検索されます。指定されたディレクトリに検索対象の画面帳票定義体ファイルが存在しない場合は、カレントディレクトリが検索されます。

3.3.1.16 -P（翻訳リストのファイル名の指定）

```
-P ファイル名
```

翻訳リストをファイルに格納する場合、翻訳リストを格納するファイル名を指定します。翻訳リストについては、"3.1.7 COBOLコンパイラが出力する情報"を参照してください。-Pオプションを複数個指定した場合、最後に指定したファイル名が有効となります。

翻訳リストファイルをソースファイル名.lstの形式で出力したい場合は、ファイル名の代わりにハイフン(-)を指定します。

注意

翻訳リストは以下のディレクトリを基点にして出力されます。

-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ディレクトリ

-dpオプションを指定しない、かつ、-Pオプションでファイル名を指定した場合

　カレントディレクトリ

-dpオプションを指定しない、かつ、-Pオプションでハイフン(-)を指定した場合

　ソースファイルと同じディレクトリ

3.3.1.17 -R（リポジトリファイルの入力先ディレクトリの指定）

```
-R ディレクトリ名
```

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ディレクトリを指定します。使用するリポジトリファイルが複数のディレクトリに存在する場合、-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、指定された順序でディレクトリが検索されます。

3.3.1.18 -Tm（マルチスレッドモデルのプログラムを翻訳するときの指定）

```
-Tm
```

マルチスレッドモデルのプログラムを翻訳する場合に指定します。マルチスレッドモデルのプログラムについては、"第17章 マルチスレッド"を参照してください。

参考

-Tmオプションは、翻訳オプションTHREAD(MULTI)と同じ意味です。

3.3.1.19 -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します。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。翻訳オプションの内容および指定形式については、"付録A 翻訳オプション"を参照してください。

翻訳に関する指示の優先順位を以下に示します。

1. ソースプログラム中の翻訳指示文で指定した翻訳オプション
2. cobolコマンドの-WCオプションで指定した翻訳オプション

3. 環境変数COBOLOPTSの-WCオプションで指定した翻訳オプション

4. cobolコマンドに指定したオプション

5. 環境変数COBOLOPTSに指定したオプション

6. cobolコマンドの-iオプションで指定したオプションファイル中の翻訳オプション

3.3.2 リンクに関するオプション

以下にcobolコマンドのリンクに関するオプションの一覧を示します。

- "3.3.2.1 -dy/-dn（結合モードの指定）"
- "3.3.2.2 -G（共用オブジェクトプログラムを生成する指定）"
- "3.3.2.3 -L（ライブラリサーチパス名を追加する指定）"
- "3.3.2.4 -l（リンクする副プログラムまたはライブラリの指定）"
- "3.3.2.5 -Ns（C言語で記述したプログラムから呼び出されるプログラムの指定）"
- "3.3.2.6 -o（オブジェクトファイルの指定）"
- "3.3.2.7 -pc（スクリーン操作機能を使うプログラムをリンクする指定）"
- "3.3.2.8 -pi（C-ISAMを使うプログラムをリンクする指定）"
- "3.3.2.9 -pm（画面定義体を使うプログラムをリンクする指定）"
- "3.3.2.10 -Tm（マルチスレッドモデルのプログラムをリンクする指定）"
- "3.3.2.11 -Wl（リンクオプションの指定）"

3.3.2.1 -dy/-dn（結合モードの指定）

```
-dy
```

または、

```
-dn
```

目的プログラムから呼ばれる副プログラムを格納したライブラリと静的結合を行う(-dn)か、動的結合を行う(-dy)かを指定します。省略された場合は、-dyが指定されたとみなし、動的結合を行います。

動的結合と静的結合が混在する場合は、-dyを指定します。この時、静的結合する副プログラムは、-Wlオプションで指示する必要があります。静的結合する副プログラムは、-Wlオプションの最後に指定してください。

3.3.2.2 -G（共用オブジェクトプログラムを生成する指定）

```
-G
```

共用オブジェクトプログラムを生成するときに指定します。-dyを指定した場合だけ有効となります。

3.3.2.3 -L（ライブラリサーチパス名を追加する指定）

```
-L ディレクトリ名
```

ライブラリを検索するディレクトリを追加したいときに指定します。

指定されたディレクトリ名は、環境変数LD_LIBRARY_PATHに設定されているライブラリサーチパス名の前に追加されます。-Lオプションは、同一コマンド行の-lオプションより前に指定した場合だけ有効となります。環境変数LD_LIBRARY_PATHは、アーカイブライブラリまたは共用オブジェクトライブラリの検索パスを設定する環境変数です。

3.3.2.4 -l（リンクする副プログラムまたはライブラリの指定）

```
-l 名前
```

動的リンク構造のプログラムを生成する場合、目的プログラムから呼ばれる副プログラムの共用オブジェクトライブラリ名を名前に指定します。また、COBOLの各種機能を使用する場合、必要なライブラリ名を名前に指定します。このオプションが指定されると、lib名前.soという共用オブジェクトプログラム、またはlib名前.aというアーカイブライブラリサブルーチンが以下のディレクトリから順に検索されます。なお、このオプションは、複数個指定することができ、指定した順に検索されます。

- cobolコマンドの-Lオプションに指定されたディレクトリ
- 環境変数LD_LIBRARY_PATHに設定されたディレクトリ

3.3.2.5 -Ns（C言語で記述したプログラムから呼び出されるプログラムの指定）

```
-Ns
```

主プログラムがC言語で記述したプログラムの場合に指定します。-Nsオプションが指定されると、この製品が提供しているスタートアップルーチンは結合されません。

3.3.2.6 -o（オブジェクトファイルの指定）

```
-o ファイル名
```

生成される実行可能プログラムまたは共用オブジェクトプログラムの格納先ファイルを指定します。-oオプションを省略した場合、a.outに格納されます。副プログラムを共用オブジェクトプログラムとして生成する場合には、-oオプションでlib副プログラム名.soというファイル名を指定します。

3.3.2.7 -pc（スクリーン操作機能を使うプログラムをリンクする指定）

```
-pc
```

スクリーン操作機能に必要なライブラリを、自動的にリンクする場合に指定します。

3.3.2.8 -pi（C-ISAMを使うプログラムをリンクする指定）

```
-pi
```

C-ISAMファイルを操作するプログラムに必要なライブラリを、自動的にリンクする場合に指定します。当オプションを指定した場合、C-ISAMから提供されるlibisam.aをリンクします。したがって、libisam.aが正しく検索されるよう環境変数LD_LIBRARY_PATHを設定してください。

3.3.2.9 -pm（画面定義体を使うプログラムをリンクする指定）

```
-pm
```

画面定義体を使用するプログラムに必要なライブラリを、自動的にリンクする場合に指定します。

3.3.2.10 -Tm（マルチスレッドモデルのプログラムをリンクする指定）

```
-Tm
```

マルチスレッドモデルのプログラムに必要なライブラリを、自動的にリンクする場合に指定します。マルチスレッドモデルのプログラムについては、“第17章 マルチスレッド”を参照してください。

3.3.2.11 -Wl（リンクオプションの指定）

```
-Wl,"リンクオプション"
```

ldコマンドに指示するオプションを指定します。-Wlオプションが複数指定された場合は、最後に指定した-Wlオプションが有効となります。

ldコマンドに指定するオプションの内容および指定形式については、"付録L ldコマンド"およびldコマンドのマニュアルを参照してください。

3.3.3 cobolコマンドの復帰値

cobolコマンドの復帰値は、プログラム翻訳時の最大重大度コードにより設定しています。最大重大度コードと復帰値の関係を以下に示します。

最大重大度コード	復帰値
I	0
W	
E	1
S	2
U	3

なお、cobolコマンドにより実行可能プログラムを作成する場合、cobolコマンドは内部でldコマンドを実行します。このldコマンドの実行でエラーが発生した場合、上記の復帰値とldコマンドの復帰値の大きい方がcobolコマンドの復帰値となります。

第4章 プログラムの実行

本章では、COBOLプログラムの実行方法について説明します。

4.1 実行環境の設定

ここでは、実行環境の設定方法について説明します。

4.1.1 実行環境

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といいます。

環境変数とは、ファイル識別名などを指定するための情報です。環境変数の詳細については、"付録E 環境変数一覧"を参照してください。

プログラムを実行する前に機能ごとに環境変数を指定しておく必要があります。各機能および環境変数の設定方法については、各機能についての説明および"付録E 環境変数一覧"を参照してください。

注意

環境変数はシステム、ほかのユーティリティおよびCOBOLプログラムが使用する環境変数と一致しないように注意する必要があります。COBOLランタイムシステムが使用しているSYS,CBRおよびCOBで始まる環境変数についてもCOBOLプログラムで使用しないでください。

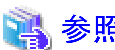
実行時に有効な環境変数は以下のとおりです。

実行環境に関するもの

- CBR_CBRFILE
- CBR_CBRINFO
- GOPT
- MGPRM

副プログラム呼出しに関するもの

- CBR_ENTRYFILE
- LD_LIBRARY_PATH


参照

"第9章 サブプログラムを呼び出す～プログラム間連絡機能～"

ファイル処理に関するもの

- ファイル識別名
- CBR_INPUT_BUFFERING
- CBR_CLOSE_SYNC
- CBR_TRAILING_BLANK_RECORD
- CBR_FILE_USE_MESSAGE
- CBR_EXFH_API
- CBR_EXFH_LOAD

- CBR_FILE_BOM_READ
- CBR_FILE_SEQUENTIAL_ACCESS
- CBR_FILE_LFS_ACCESS

参照

"第6章 ファイル処理"

表示ファイルに関するもの

- ファイル識別名
- MEFTDIR
- CBR_PSFILE_ACM
- CBR_PSFILE_APL
- CBR_PSFILE_DSP
- CBR_PSFILE_PRT

"第8章 画面を使った入出力"

印刷ファイルに関するもの

- ファイル識別名
- FCBDIR
- CBR_PRT_INF
- CBR_FCB_NAME
- FOVLDIR
- CBR_LP_OPTION
- CBR_PRINTFONTTABLE
- CBR_PRT_UTF8_CONVERT

"第7章 印刷処理"

スクリーン操作に関するもの

- CBR_SCR_KEYDEFFILE
- COLUMNS
- LINES

"第8章 画面を使った入出力"

整列・併合に関するもの

- BSORT_TMPDIR

参照

“第11章 SORT文およびMERGE文の使い方～整列併合機能～”

簡易アプリ間通信に関するもの

- CBR_CI_INF
- CBR_CI_CLG

参照

“第19章 通信機能”

小入出力に関するもの

- 翻訳オプションSSINまたはSSOUTに指定した名前
- CBR_CONSOLE
- CBR_JOBDATE
- CBR_MESSOUTFILE
- CBR_COMPOSER_SYSOUT
- CBR_COMPOSER_SYSERR
- CBR_COMPOSER_CONSOLE
- CBR_DISPLAY_CONSOLE_OUTPUT
- CBR_DISPLAY_SYSOUT_OUTPUT
- CBR_DISPLAY_SYSERR_OUTPUT
- CBR_DISPLAY_CONSOLE_SYSLOG_LEVEL
- CBR_DISPLAY_SYSOUT_SYSLOG_LEVEL
- CBR_DISPLAY_SYSERR_SYSLOG_LEVEL
- CBR_DISPLAY_CONSOLE_SYSLOG_IDENT
- CBR_DISPLAY_SYSOUT_SYSLOG_IDENT
- CBR_DISPLAY_SYSERR_SYSLOG_IDENT

参照

“第10章 ACCEPT文およびDISPLAY文の使い方”

“D.3 CURRENT-DATE関数を利用した西暦の取得”

オブジェクト指向に関するもの

- CBR_CLASSINFFILE
- CBR_INSTANCEBLOCK

参照

“第16章 オブジェクト指向プログラムの開発と実行”

マルチスレッドに関するもの

- CBR_THREAD_TIMEOUT
- CBR_SYMFOWARE_THREAD
- CBR_SYSERR_EXTEND
- CBR_SSIN_FILE

参照

“第17章 マルチスレッド”

デバッグ機能に関するもの

- SYSCOUNT
- CBR_TRACE_FILE
- CBR_TRACE_PROCESS_MODE
- CBR_MEMORY_CHECK

参照

“第5章 プログラムのデバッグ”

デバッグプログラムからのデバッガ起動に関するもの

- CBR_ATTACH_TOOL

参照

“第23章 対話型デバッガの使い方”

コード系に関するもの

- CBR_CODE_CHECK
- LANG
- LC_ALL

参照

“付録J 日本語コード系”

組込み関数に関するもの

- CBR_FUNCTION_NATIONAL

参照

"付録D 組込み関数の使用"

実行時メッセージの出力に関するもの

- CBR_CONSOLE
- CBR_MESS_LEVEL_CONSOLE
- CBR_MESS_LEVEL_SYSLOG
- CBR_MESSOUTFILE
- CBR_COMPOSER_MESS

参照

"4.3 実行時メッセージの出力方法の指定"

CSV形式データの操作に関するもの

- CBR_CSV_OVERFLOW_MESSAGE
- CBR_CSV_TYPE

参照

"第27章 CSV形式データの操作"

その他

- TMPDIR

4.1.2 実行環境の設定方法

実行環境の設定方法を以下に示します。

a. シェルの初期化ファイルに設定する

b. 環境変数設定コマンドで設定する

c. 実行用の初期化ファイルに設定する

d. コマンド行で設定する(実行時オプション)

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

実行環境は、a.またはb.で設定することをおすすめします。

注意

- a.とb.は、使用するシェルにより設定方法が異なります。設定方法は、使用するシェルのマニュアルを参照してください。
- 実行用の初期化ファイルに指定された実行環境は、COBOLアプリケーション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。
    - 環境変数は、プログラムの起動前にシェルプログラムなどでユーザー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。

4.1.2.1 シェルの初期化ファイルに設定する方法

この方法は、システム共通のシェルの初期化ファイルまたは、ユーザー固有のシェルの初期化ファイルを使用して環境変数を設定する方法です。

複数のアプリケーションに共通な環境変数の値を定義する場合に設定しておくと便利です。

4.1.2.2 環境変数設定コマンドで設定する方法

この方法は、シェルまたはシェルプログラムの環境変数設定コマンドを使って環境変数を設定する方法です。

シェルで環境変数を設定すると、そのシェルから起動されたプログラムでは、その環境変数が有効になります。また、シェルプログラムを使うことによって、環境設定から実行までを1回の作業で行うこともできます。

シェルプログラムで設定した環境変数は、そのシェルファイルで起動したプログラムにだけ有効になります。

4.1.2.3 実行用の初期化ファイルに設定する方法

実行用の初期化ファイルを作成し、実行環境を設定する方法です。

実行用の初期化ファイルとは、COBOL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の起動されたディレクトリの"COBOL.CBR"を実行用の初期化ファイルとして扱います。

実行可能プログラムがパス指定で起動された場合には、実行可能プログラムが格納されているディレクトリの"COBOL.CBR"を実行用の初期化ファイルとして扱います。

"COBOL.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。

- 環境変数CBR_CBRFILEで指定する場合は、"付録E 環境変数一覧"を参照してください。
- コマンド行オプション-CBRで指定する場合は、"4.2 実行操作"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。

4.1.2.3.1 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、それぞれのプログラムに共通する環境変数を記述します。ここに記述した環境変数は、アプリケーションが終了するまで有効になります。

実行用の初期化ファイルの内容を以下に示します。

```
; コメント                               … [1]
環境変数名=設定内容                       … [2]

     :
```

[1] 実行用の初期化ファイルのコメント

[2] プログラムに共通する環境変数

- 1つの行に2個以上の環境変数を記述することはできません。
- 同一の環境変数名を複数個指定しないでください。同一の環境変数名を複数個指定した場合の動作は保証されません。

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。

環境変数名に空白文字を指定することはできません。また、書式に誤りがあった場合、次の行を解析します。

例

実行用の初期化ファイルの記述例

```
; Environment
CBR_CBRINFO=YES
```

4.1.2.3.2 実行用の初期化ファイルの検索順序について

実行用の初期化ファイルの検索順序を以下に示します。

1. 実行可能プログラムのディレクトリ配下のCOBOL.CBR
2. 最初に動作したライブラリのあるディレクトリ配下のCOBOL.CBR
3. 環境変数CBR_CBRFILEに指定された実行用の初期化ファイル

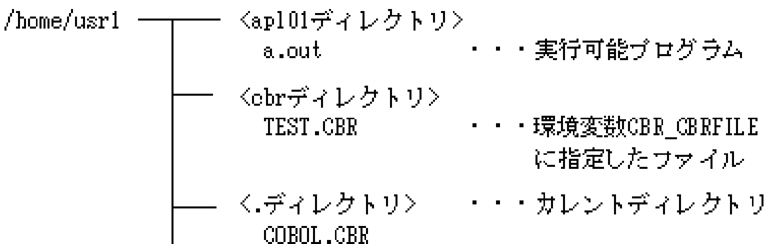
注意

最初に動作したライブラリのCOBOLアプリプログラム名が、実行可能プログラムに存在する場合、実行用の初期化ファイルの検索位置は、実行可能プログラムの格納ディレクトリとなり、最初に動作したライブラリの格納ディレクトリは検索しません。

例

検索順序の例

次の環境を例に実行用の初期化ファイルの検索順序を説明します。



上記の例では、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. /home/usr1/apl01/COBOL.CBR
2. /home/usr1/lib01/COBOL.CBR
3. /home/usr1/cbr/TEST.CBR

```
$ PATH=/home/usr1/apl01:$PATH
$ export PATH
$ LD_LIBRARY_PATH=/home/usr1/lib01:$LD_LIBRARY_PATH
$ export LD_LIBRARY_PATH
$ CBR_CBRFILE=/home/usr1/cbr/TEST.CBR
$ export CBR_CBRFILE
$ a.out
```

上記の例では、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. /home/usr1/apl01/COBOL.CBR

2. 1.がない場合、/home/usr1/lib01/COBOL.CBR
3. 2.がない場合、/home/usr1/cbr/TEST.CBR

実行用の初期化ファイルの読込みに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実行可能プログラムとは、COBOLプログラムおよび他言語プログラムを指します。また、ライブラリは、COBOLの実行環境開設時にローディングされているものとします。

		ライブラリの格納位置	
		COBOL.CBRファイルあり	COBOL.CBRファイルなし
実行可能プログラムの格納位置	**COBOL.CBRファイルあり**	実行可能プログラムの格納位置が有効	実行可能プログラムの格納位置が有効
	COBOL.CBRファイルなし	ライブラリの格納位置が有効	-

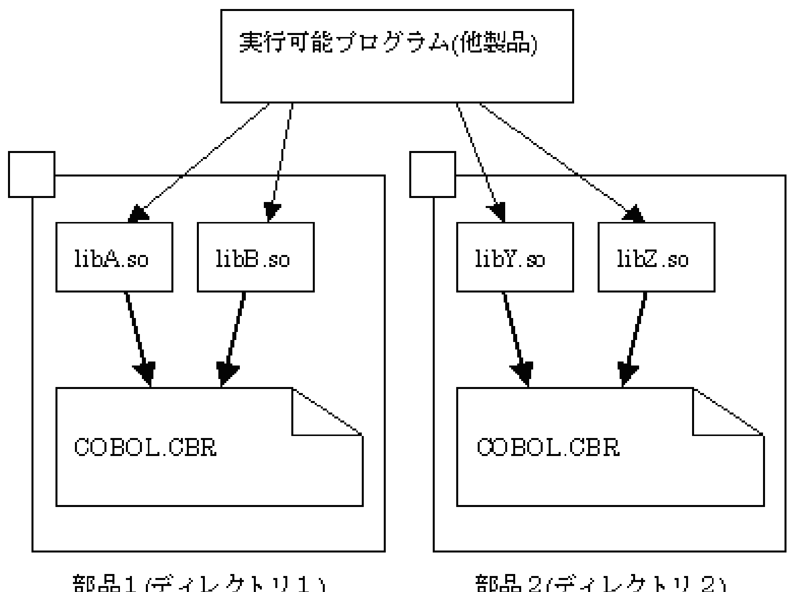
4.1.2.3.3 ライブラリ格納ディレクトリの実行用の初期化ファイルの使用について

通常、実行用の初期化ファイルは、実行可能プログラムのあるディレクトリ配下のCOBOL.CBRが使用されます。このため、COBOL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(ライブラリ)を起動する実行可能プログラム(他製品)が格納されているディレクトリに、COBOL.CBRを置く必要がある。
   → 実行可能プログラム(他製品)の格納ディレクトリを意識しなければならない。
2. 実行可能プログラムのあるディレクトリのCOBOL.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。
   → COBOL.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(ライブラリ)があるディレクトリにCOBOL.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(ライブラリ)のディレクトリ配下にCOBOL.CBRを置くため、実行可能プログラム(他製品)の格納場所を意識しなくてよい。
- それぞれのアプリケーション用の情報だけを記述したCOBOL.CBRを使用することができる。



例えば上記の場合、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(libA.so,libB.so)を、このプロセス単位で有効なCOBOL.CBRとともにディレクトリ1に置きます。実行可能プログラムから部品1を起動すると、ディレクトリ1のCOBOL.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、ディレクトリ2のCOBOL.CBRの情報が、部品2のプロセス単位で有効となります。

- 実行可能プログラムがCOBOLの場合、COBOLアプリケーション(ライブラリ)のあるディレクトリ配下のCOBOL.CBRは使用できません。
- COBOLアプリケーション(ライブラリ)は、実行可能プログラムから動的プログラム構造 で呼び出すようにしてください。動的リンク構造で呼び出した場合、COBOLアプリケー ション(ライブラリ)のあるディレクトリ配下のCOBOL.CBRは使用できません。
- 実行用の初期化ファイルは、実行可能プログラムのあるディレクトリから検索される ため、実行可能プログラムのあるディレクトリ配下にCOBOL.CBRを置かないでください。
- COBOLアプリケーション(ライブラリ)は、プロセス(実行環境)単位の情報を持つCOBOL.CBR とともに、プロセス(実行環境)単位で1つのディレクトリに格納してください。COBOL アプリケーション(ライブラリ)がプロセス(実行環境)単位で1つのディレクトリにない 場合、実行時に意図しない動作を取る原因になります。
- COBOL.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。

4.1.2.3.4 実行用の初期化ファイルの情報を表示する方法

環境変数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

"付録E 環境変数一覧"

4.1.2.4 コマンド行で設定する方法

この方法は、実行環境の内容をコマンドの引数として指定する方法です。

この方法では、OSIV系形式の実行時パラメタ（環境変数情報MGPRM）および実行用の初期化ファイル（環境変数情報GOPT）を指定することができます。

参照

"4.2.3 実行用の初期化ファイルを指定する"

"4.2.4 OSIV系形式の実行時パラメタを指定する"

4.1.3 環境変数による接続製品の指定

表示ファイルから実行時に環境変数を使用することにより、接続製品を指定することができます。

これにより、製品ごとに異なる環境変数を指定することなく、表示ファイル固有の環境変数を使用して接続製品を指定することができます。また、同一プログラムからファイルごと、あて先ごとに異なる接続製品を選択することができます。

ファイルごとに指定する場合

```
ファイル識別名 = [情報ファイル名] [,接続製品識別名]
```

情報ファイル名

以下に情報ファイルを示します。接続製品によって、ファイル名が異なります。

― ウィンドウ情報ファイル

― プリンタ情報ファイル

接続製品識別名

接続製品識別名には、次の文字列の中から指定します。

― MeFtの場合　：MEFT

― MeFt/NETの場合　：MEFTNET

あて先ごとに指定する場合

- あて先DSPの場合

```
CBR_PSFILE_DSP=接続製品識別名
```

- あて先PRTの場合

```
CBR_PSFILE_PRT=接続製品識別名
```

- あて先ACMの場合

```
CBR_PSFILE_ACM=接続製品識別名
```

- あて先APLの場合

```
CBR_PSFILE_APL=接続製品識別名
```

使い方

あて先DSPを指定した表示ファイルから、接続製品のMEFTNETを使用して画面表示を行う場合

```
$ ファイル識別名=ディスプレイ情報ファイル名,MEFTNET ; export ファイル識別名
```

または

```
$ ファイル識別名=ディスプレイ情報ファイル名 ; export ファイル識別名
$ CBR_PSFILE_DSP=MEFTNET ; export CBR_PSFILE_DSP
```

注意

- 環境変数に指定する接続製品識別名は、対象となる表示ファイルのあて先種別をサポートしている製品を指定する必要があります。
- 以下の順に環境変数の指定が検索されます。
  1. “ファイルごとに指定する場合”の指定
  2. “あて先ごとに指定する場合”の指定
- 対象となる表示ファイルに対して、“ファイルごとに指定する場合”と“あて先ごとに指定する場合”に異なる接続製品名を指定した場合、“ファイルごとに指定する場合”で指定した接続製品識別名が有効になります。
- 対象となる表示ファイルに対して、“ファイルごとに指定する場合”および“あて先ごとに指定する場合”が省略された場合、以下のあて先ごとに決められた接続製品識別名が指定されたものとみなされます。

  あて先DSP：MEFT
  あて先PRT：MEFT
  あて先ACM：ACM

- 接続製品名の文字列に誤りがある場合、接続製品名は省略されたものとみなされます。

4.1.4 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。ただし、呼び出すプログラムの共用オブジェクトファイル名が"libプログラム名.so"の場合は省略することができます。

エントリ情報の指定方法は、環境変数CBR_ENTRYFILEにエントリ情報ファイル名を指定します。

エントリ情報ファイルとは、副プログラムのエントリ情報の定義の開始を示す"ENTRY"セクションを持ち、そのセクションにエントリ情報が指定されているファイルのことをいいます。

エントリ情報ファイルの形式

```
[ENTRY]                    …[1]
エントリ情報
```

[1] 副プログラム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、固定文字列"ENTRY"です。このセクションは、エントリ情報ファイルに1つしか記述できません。

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、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は保証されません。

4.1.4.1 副プログラム名の指定形式

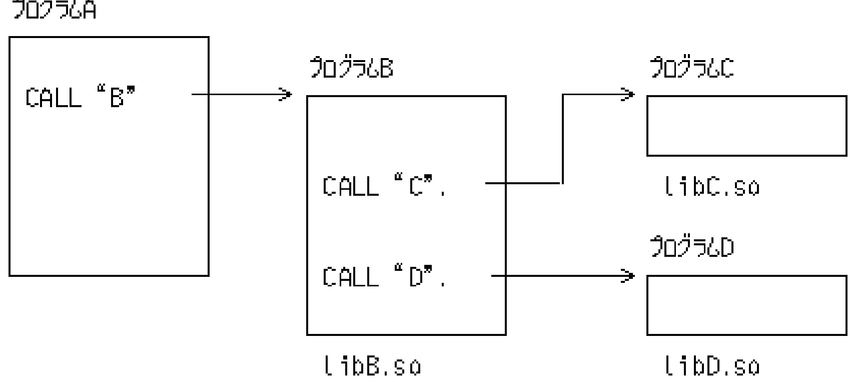
```
副プログラム名=共用オブジェクトファイル名
```

副プログラム名とその副プログラムを含む共用オブジェクトファイル名を関連付けるために、副プログラム名には、呼び出すプログラムのプログラム名を指定し、共用オブジェクトファイル名には、呼び出されるプログラムが格納されている共用オブジェクト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、環境変数"LD_LIBRARY_PATH"に設定されているディレクトリから検索されます。

共用オブジェクトファイル名の拡張子は、"so"でなければなりません。

共用オブジェクトが1つの副プログラムで構成されている場合の例

プログラムの呼出し関係



エントリ情報ファイルの指定例

```
CBR_ENTRYFILE=FILE
```

FILEの内容

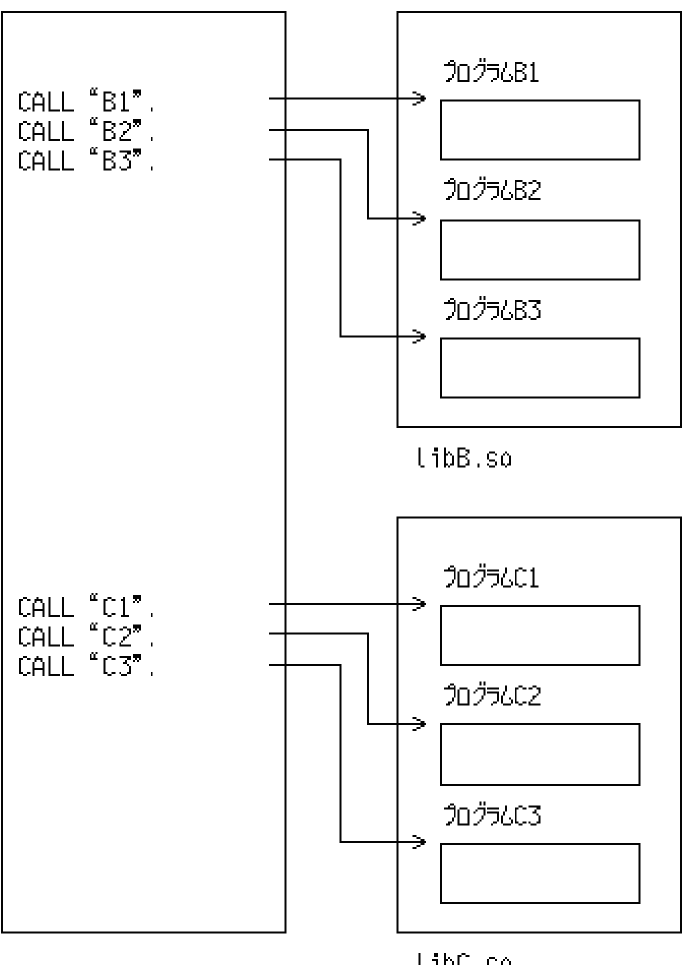
```
[ENTRY]
B=libB.so
C=libC.so
D=libD.so
```

ポイント

この例では、共用オブジェクトファイル名が“libプログラム名.so”となっているため、エントリ情報は省略することができます。

共用オブジェクトが複数の副プログラムで構成されている場合の例

プログラムの呼出し関係



エントリ情報ファイルの指定例

```
CBR_ENTRYFILE=FILE
```

FILEの内容

```
[ENTRY]
B1=libB.so
B2=libB.so
B3=libB.so
C1=libC.so
```

```
C2=libC.so
C3=libC.so
```

ポイント

共用オブジェクト内の呼び出された副プログラムに対してCANCEL文が実行されたときに、共用オブジェクトはメモリ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、libB.soがメモリ上から削除され、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、libC.soがメモリ上から削除されます。

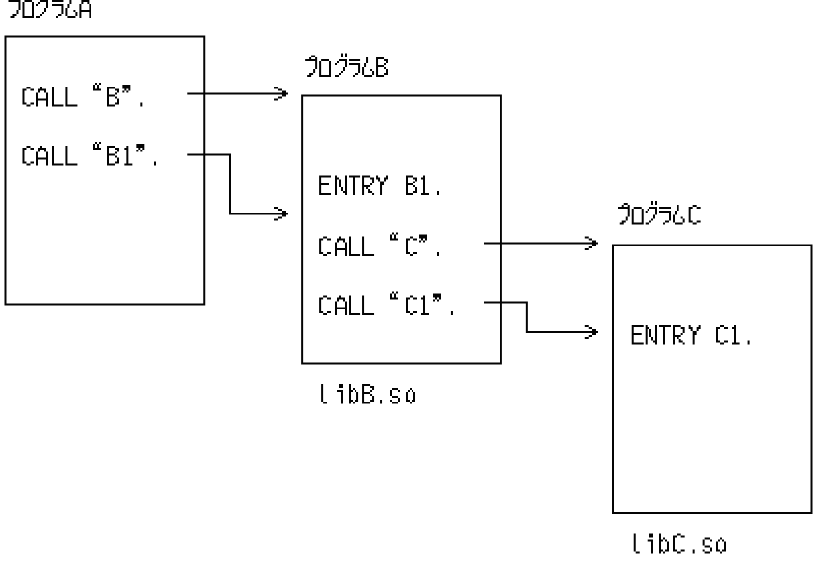
二次入口点を指定する場合や同一の共用オブジェクト内の複数の副プログラムを呼び出す必要がある場合は、"二次入口点名の指定形式"の指定を副プログラム名の指定形式に追加します。

4.1.4.2 二次入口点名の指定形式

```
二次入口点=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。

プログラムの呼出し関係



エントリ情報ファイルの指定例

```
CBR_ENTRYFILE=FILE
```

FILEの内容

```
[ENTRY]
B=libB.so
C=libC.so
B1=B
C1=C
```

4.2 実行操作

ここでは、COBOLプログラムの実行方法を説明します。

4.2.1 プログラムの実行形式

翻訳･リンクを行い作成した実行可能プログラムは、そのプログラムが格納されているファイル(実行可能ファイル)の名前をコマンド名として実行します。コマンドには、引数を指定することができます。COBOLプログラムでは、引数に指定した値を、コマンド行引数の操作機能を使って受け取ることができます。コマンド行引数の操作機能については、"10.2 コマンド行引数の取出し"を参照してください。

COBOLプログラムの実行方法を以下に示します。なお、実行時に実行可能ファイルがカレントディレクトリにない場合、絶対パス名で実行可能ファイル名を指定する必要があります。

```
$ ファイル名 [引数][-CBL 実行時オプション][-CBR 実行用の初期化ファイル名]
```

-CBRおよび-CBLは順不同です。

引数の指定方法

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

```
$ PROG1 A B C,D
```

引用として"A"、"B"、"C,D"の3つの引数が指定されました。

```
$ PROG1 "A B C,D"
```

引数として、"A B C,D"という1つの引数が指定されました。

OSIV系形式の実行時パラメタ

OSIV系形式で実行時パラメタを指定する場合、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。詳細については"4.2.4 OSIV系形式の実行時パラメタを指定する"を参照してください。

実行時オプションの指定方法

実行時オプションは、識別子-CBLの後ろに指定します。実行時オプションの指定形式については、"4.2.2 実行時オプションを指定する"を参照してください。

```
$ PROG1 -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。

実行用の初期化ファイル名の指定方法

実行用の初期化ファイル名は、識別子-CBRの後ろに指定します。実行用の初期化ファイル名の指定形式については、"4.2.3 実行用の初期化ファイルを指定する"を参照してください。

```
$ PROG1 -CBR abc.ini
```

実行用の初期化ファイル名として、カレントディレクトリのabc.iniが指定されました。

4.2.2 実行時オプションを指定する

実行時オプションは、実行時にCOBOL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、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、以下に示す形式で、コマンドの引数として指定します。

```
-CBL  実行時オプションの並び
```

または環境変数GOPTに指定することもできます。

```
$ GOPT=実行時オプションの並び;export GOPT
```

実行時オプションの並びに指定できるオプションを"表4.1 実行時オプション"に示します。実行時オプションの並びには、複数のオプションをコンマ(,)で区切って指定することができます。

表4.1 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を指定	[smsize値k]

4.2.2.1 [r個数 | nor]（トレース情報の個数の指定、およびTRACE機能の抑制指定）

TRACE機能が出力するトレース情報の数を変更したい場合に指定します。個数には、出力するトレース情報の数を1～999999で指定します。
TRACE機能を抑制する場合は、norを指定します。
これらのオプションは、翻訳時に-Drオプションまたは翻訳オプションTRACEを指定したプログラムにだけ有効です。

参照

"3.3.1.4 -Dr（TRACE機能を使用する指定）"

"A.2.46 TRACE（TRACE機能の使用の可否）"

4.2.2.2 [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

"3.3.1.3 -Dk (CHECK機能を使用する指定)"

"A.2.3 CHECK(CHECK機能の使用の可否)"

4.2.2.3 [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。

4.2.2.4 [smsize値k] (PowerSORTが使用するメモリ容量を指定)

SORT文およびMERGE文から呼出される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 ユーザーズガイド"をお読みください。
このオプションは、翻訳オプションSMSIZEおよび特殊レジスタSORT-CORE-SIZEに指定する値の意味と等価ですが、同時に指定された場合の優先順位は、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。

4.2.3 実行用の初期化ファイルを指定する

実行時にCOBOLプログラムに対して、実行用の初期化ファイルを指定することができます。

実行用の初期化ファイルは、以下に示す形式で、コマンドの引数として指定します。

```
-CBR 実行用の初期化ファイルパス名
```

実行用の初期化ファイル名の指定方法

実行用の初期化ファイル名は、識別子-CBR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

```
a.out -CBR abc.init
```

実行用の初期化ファイル名として、abc.initが指定されました。

4.2.4 OSIV系形式の実行時パラメタを指定する

本システムでは、OSIV系形式の実行時パラメタを指定することができます。

OSIV系システムとは、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です。

OSIV系システムでパラメタを渡す場合

[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.
```

[入力コマンド]

```
CALL 'X9999.A.LOAD(A)''ABCDE'
```

[パラメタの内容]

5	A	B	C	D	E

本システム上で上記と同じパラメタを渡す場合

コマンド名の直後に実行時パラメタを指定

```
$ PROG1 "ABCDE"
```

環境変数MGPRMに実行時パラメタを指定

```
$ MGPRM="ABCDE"; export MGPRM
```

注意

- OSIV系形式の実行時パラメタは、1つしか渡せません。
- パラメタ文字列の最大長は100バイトです。
- パラメタ文字列の有効長は、パラメタ長に設定された長さまでです。
- パラメタ長を超えた領域は参照できません。
- パラメタ文字列の領域へ値を設定することはできません。

4.3 実行時メッセージの出力方法の指定

COBOL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先や、メッセージを出力する重大度を指定するための環境変数を下表に示します。

出力先	出力先の指定	重大度コードの指定
システムの標準エラー出力(stderr)	CBR_MESSOUTFILE (*1) CBR_CONSOLE (*2)	CBR_MESS_LEVEL_CONSOLE (*4)
Interstage Business Application Serverの汎用ログ	CBR_COMPOSER_MESS (*3)	－
Syslog	－	CBR_MESS_LEVEL_SYSLOG (*5)

*1: 環境変数CBR_MESSOUTFILEについては、"4.3.2 実行時メッセージのファイル出力"を参照

*2: 環境変数CBR_CONSOLEについては、"第10章 ACCEPT文およびDISPLAY文の使い方"を参照

*3: 環境変数CBR_COMPOSER_MESSについては、"4.3.4 実行時メッセージのInterstage Business Application Serverの汎用ログへの出力"を参照

*4: 環境変数CBR_MESS_LEVEL_CONSOLEについては、"4.3.1 実行時メッセージの重大度指定"を参照

*5: 環境変数CBR_MESS_LEVEL_SYSLOGについては、"4.3.3 実行時メッセージのSyslog出力"を参照

4.3.1 実行時メッセージの重大度指定

環境変数CBR_MESS_LEVEL_CONSOLEの指定により、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の指定をすることができます。

```
CBR_MESS_LEVEL_CONSOLE=[重大度コード]
```

例

```
$ CBR_MESS_LEVEL_CONSOLE=I; export CBR_MESS_LEVEL_CONSOLE
```

重大度コードがI以上の実行時メッセージを出力します。

CBR_MESS_LEVEL_CONSOLE に指定できるパラメタは以下の通りです。

- 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。

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CBR_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、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。

4.3.2 実行時メッセージのファイル出力

環境変数CBR_MESSOUTFILEの指定により、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ことができます。

この機能は、Interstageなどの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でのエラーメッセージをロギングするために利用できます。

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、サーバの標準エラー出力に出力されます。サーバによって標準エラー出力が異なるので、実行時メッセージを確実に把握するためには、環境変数CBR_MESSOUTFILEを必ず指定してください。

```
CBR_MESSOUTFILE=メッセージ出力ファイル名
```

例

```
$ CBR_MESSOUTFILE=errmsg.log; export CBR_MESSOUTFILE
```

実行時メッセージ出力ファイルとしてerrmsg.logが指定されました。

注意

- 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、カレントディレクトリからの相対パスとなります。
- 同一名のファイルが存在した場合、そのファイルに情報が追加されます。

- 入出力機能の出力ファイルとして指定したファイルをCBR_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。
- ファイルの最大サイズは、2Gバイトです。
- ファイルに出力できない場合、実行時メッセージは、標準エラー出力に出力されます。

4.3.3 実行時メッセージのSyslog出力

環境変数CBR_MESS_LEVEL_SYSLOGの指定により、ランタイムシステムが出力する実行時メッセージのSyslogへの出力を抑止したり、実行時メッセージを出力する重大度コードを変更したりすることができます。

```
CBR_MESS_LEVEL_SYSLOG=[重大度コード]
```

例

```
$ CBR_MESS_LEVEL_SYSLOG=I; export CBR_MESS_LEVEL_SYSLOG
```

重大度コードがI以上の実行時メッセージをSyslogに出力します。

CBR_MESS_LEVEL_SYSLOG に指定できるパラメタは以下の通りです。

- NO ： 実行時メッセージをSyslogへ出力しません。
- I ： 重大度コードがI以上のメッセージをSyslogに出力します。
- W ： 重大度コードがW以上のメッセージをSyslogに出力します。
- E ： 重大度コードがE以上のメッセージをSyslogに出力します。
- U ： 重大度コードがUのメッセージをSyslogに出力します。

CBR_MESS_LEVEL_SYSLOG 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UのメッセージがSyslogに出力されます。(U指定と同意)

SyslogにメッセージをІ出力するときのSyslogのパラメタfacilityはLOG_USERです。また、Syslogのパラメタlevelは重大度コードにより以下のとおりとなります。

- I ： LOG_INFO
- W ： LOG_WARNING
- E ： LOG_ERR
- U ： LOG_ALERT

Syslogの運用によっては、facilityおよびlevelパラメタによって出力を抑止したり、あて先を変更したりすることができます。CBR_MESS_LEVEL_SYSLOGの指定どおりにメッセージが出力されない場合、Syslogの設定(syslog.conf)を確認してください。

4.3.4 実行時メッセージのInterstage Business Application Serverの汎用ログへの出力

環境変数CBR_COMPOSER_MESSの指定により、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の汎用ログへ出力することができます。

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

例

```
$ CBR_COMPOSER_MESS=mylog; export CBR_COMPOSER_MESS
```

ログ定義ファイルで定義されている管理名mylogの定義に従い実行時メッセージを出力します。

ログ定義ファイルで定義されている管理名については、Interstage Business Application Serverのマニュアルを参照してください。

注意

実行時メッセージを汎用ログに出力する場合、以下の環境変数の指定は無効になり、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_SYSLOG

参考

実行時メッセージの重大度と汎用ログへの出力レベルの対応は以下の通りです。

- I ： レベル6で出力します。
- W ： レベル4で出力します。
- E ： レベル3で出力します。
- U ： レベル1で出力します。

4.4 終了ステータス

STOP RUN文または主プログラムのEXIT PROGRAM文を実行し、COBOLプログラムを終了した場合、特殊レジスタPROGRAM-STATUSの値がCOBOLプログラムの復帰値となります。

特殊レジスタPROGRAM-STATUSは、暗にPIC S9(9) COMP-5と宣言された数字項目で、COBOLプログラム中で値を設定することができます。

COBOLプログラムを起動したシェルスクリプト中でこの値を参照することにより、実行制御を含む定型業務アプリを構築することが容易になります。以下に、使用例を示します。

COBOLソースプログラムの内容

```
IDENTIFICATION DIVISION.
  PROGRAM-ID. PROG1.
 ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SPECIAL-NAMES.
     ARGUMENT-VALUE IS 呼名引数の値.
 DATA DIVISION.
  WORKING-STORAGE  SECTION.
  01  引数の値      PIC  X.
 PROCEDURE  DIVISION.
     ACCEPT  引数の値  FROM  呼名引数の値.
     IF  引数の値  >=  "0"  AND  引数の値  <=  "9"
      THEN MOVE  0 TO PROGRAM-STATUS
      ELSE MOVE  1 TO PROGRAM-STATUS
     END-IF.
```

```
    EXIT  PROGRAM.
 END  PROGRAM  PROG1.
```

PROG1を実行するときに指定した引数の値が数字であるかを判定した値を復帰値として返却します。

実行のためのシェルスクリプトの内容(抜粋)

```
if PROG1 $1
   then
     echo $1 ″部印刷します″
   else
     echo ″error 数字を指定してください″
fi
```

PROG1 からの復帰値を判定し、メッセージを表示します。

注意

シェルスクリプトの“echo $？”を使ってプログラムの復帰値を正しく参照するためには、PROGRAM-STATUSに1バイトで表せる値(0～255)を設定する必要があります。

4.5 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。

4.5.1 COBOLプログラムの実行時にスタックオーバーフローが発生する場合

COBOLプログラムの実行時にバスエラーなどが発生する原因の1つとして、プロセスのスタック域がオーバーフローした場合があります。スタックオーバーフローが原因の場合には、スタックサイズを拡張することにより問題を回避することができます。

注意

スタックオーバーフローが発生したことを直接確認することはできません。

プロセスのスタックサイズを参照する方法

プロセスのスタックサイズは以下のコマンドで参照できます。

Bourne shellの場合

```
$ ulimit -s
```

C shellの場合

```
% limit stacksize
```

COBOLプログラムで必要とするスタックサイズの求め方

計算式

```
トータルスタックサイズ  ≒ （α1 + α2 …) + (β1 + β2 …)
                          |________|     |________|
                          ↑実行される    ↑実行される
                          プログラムの数分 メソッドの数分
```

- αn … プログラム定義での使用スタックサイズ値(n=1、2、…)
- βm … 呼び出されるメソッドの使用スタックサイズ値(m=1、2、…)

個々のスタックサイズはセクションサイズリストから求めることができます。セクションサイズリストについては、"3.1.7.6 データエリアに関するリスト"を参照してください。

スタックオーバーフローを回避する方法

スタックオーバーフローを回避するためには、プロセスのスタックサイズをCOBOLプログラムで必要とするスタックサイズより大きな値に設定します。

例

プロセスのスタックサイズを16384(Kbyte)にする場合

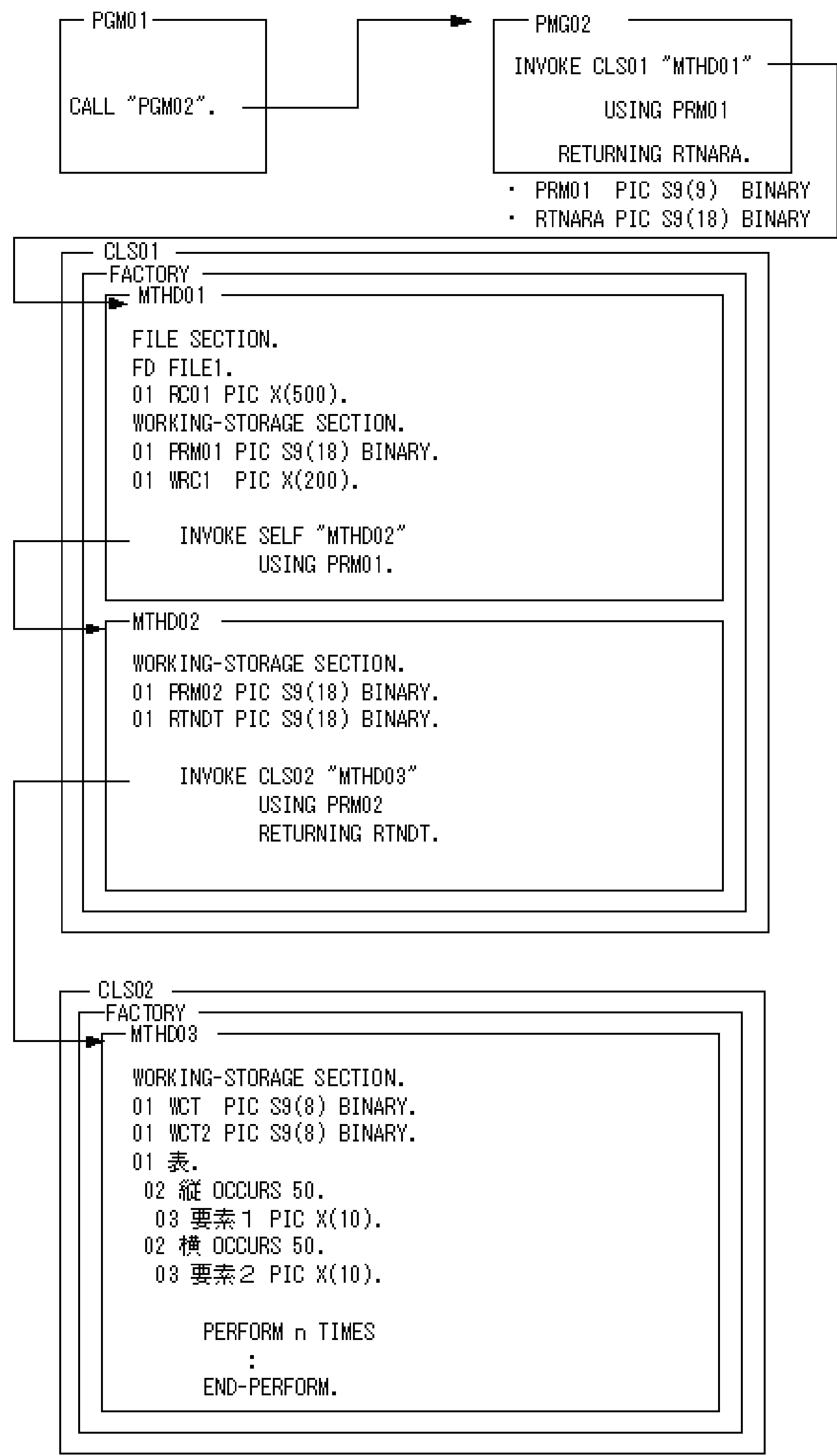
- Bourne shellの場合

```
$ ulimit -s 16384
```

- C shellの場合

```
% limit stacksize 16384
```

例題1 オーバーフローしないケース（スタックサイズが1Mバイトの場合）



例題1でのスタック使用状況

PGM（プログラム定義）	
PGM01	300バイト
PGM02	300バイト

CLS(クラス定義)

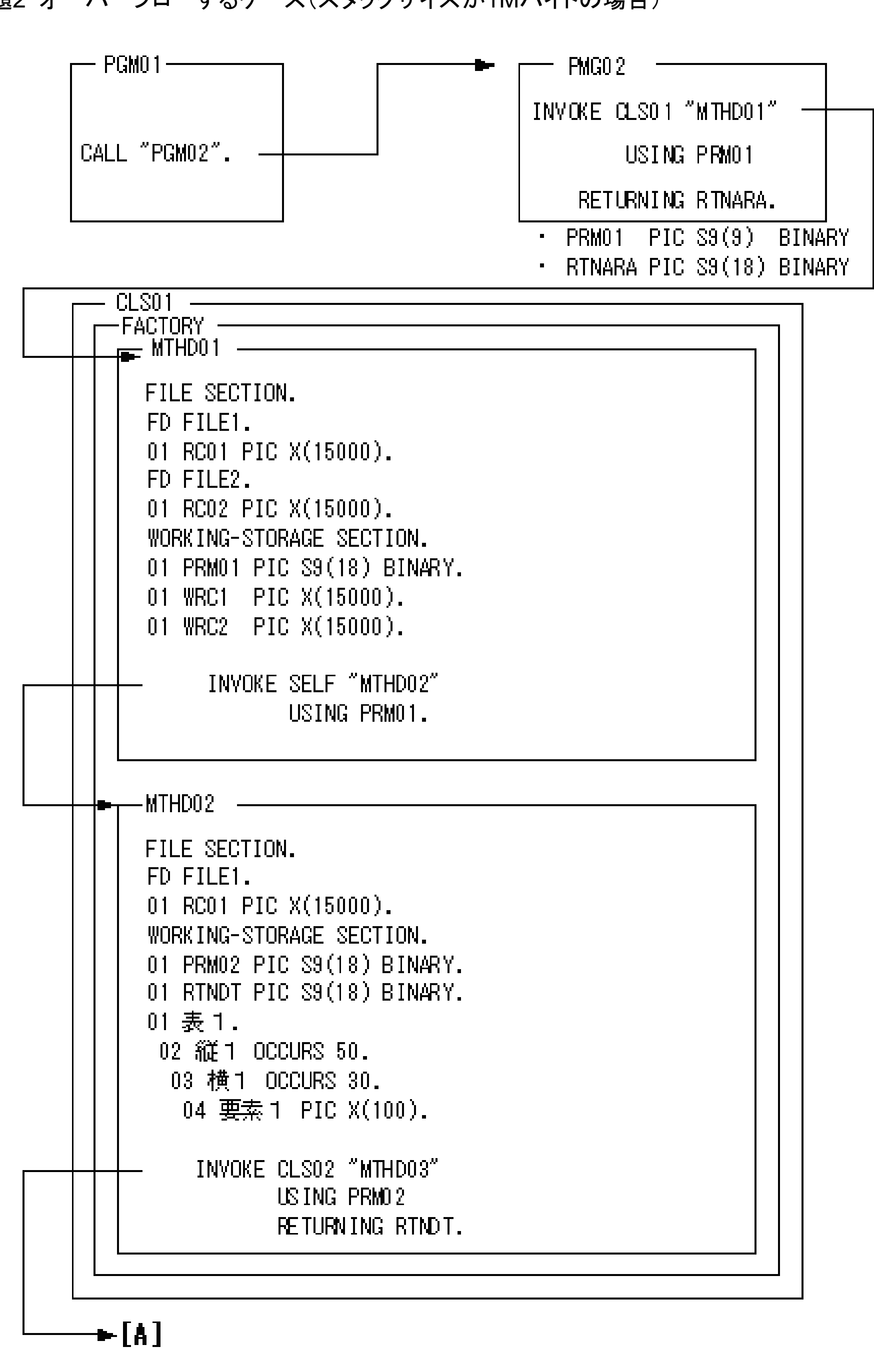
CLS01
METHOD(1)　1,300バイト

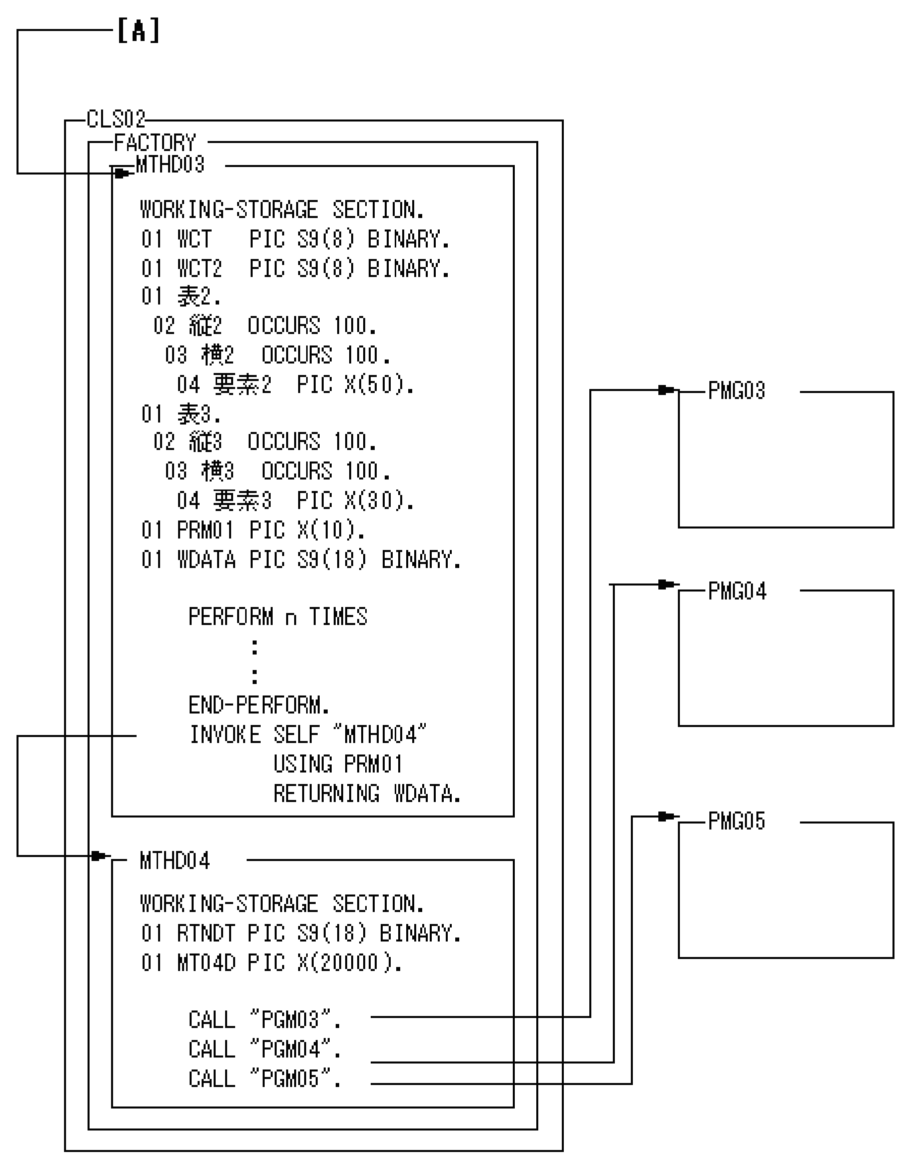
METHOD(2)　500バイト

CLS02
METHOD(3)　1,400バイト

計	3,800バイト

例題2 オーバーフローするケース(スタックサイズが1Mバイトの場合)





例題2でのスタック使用状況

PGM(プログラム定義)	
PGM01	300バイト
PGM02	300バイト
PGM03, PGM04, PGM05の最大スタックサイズ	300バイト
CLS(クラス定義)	
CLS01	
METHOD(1)	60,800バイト
METHOD(2)	166,000バイト
CLS02	
METHOD(3)	800,500バイト
METHOD(4)	20,400バイト
計	1,048,600バイト

4.5.2 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あります。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ほか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ほか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。

4.5.3 英語環境下でのフォントについて

英語環境では、プロポーショナルフォントを使用すると、リテラル文字の横幅がずれることがあるため注意してください。

第5章 プログラムのデバッグ

この製品で作成したプログラムをデバッグする手段には、以下の方法があります。

a. 直接ソースを修正し、実行しながらデバッグする方法

デバッグ行を利用してデバッグ用のロジックをソースに組み込んだり、DISPLAY文を挿入したりすることで、実行中のデータや文字列を出力しデータの値や制御の流れを直接確認します。
なお、デバッグ行の詳細は"COBOL文法書"を参照してください。

b. 翻訳オプション(TRACE、CHECK、COUNT)を指定してデバッグ専用のオブジェクトを出力し、実行時にプログラムの論理的な誤りを検出する方法

c. 実運用のオブジェクト(注)を使用し、COBOL専用のデバッガと対話しながらデバッグする方法

実運用のオブジェクトをそのまま使用し、COBOL専用のデバッガと対話しながら、デバッグプログラムを中断したり、ある時点でのデータを参照・更新したりします。

注：デバッグ用のオブジェクトは、その翻訳時に翻訳オプション(TEST)を指定します。このため、広域最適化の翻訳オプション(OPTIMIZE)は無効(NOOPTIMIZE)となります。

a.はCOBOL言語仕様の範囲で対応できます。

b.およびc.は専用の機構を用意しています。この章ではb.のデバッグ機能およびその使用方法について示します。

なお、c.で使用する対話型デバッガを使用したデバッグ方法については、"第23章 対話型デバッガの使い方"を参照してください。

5.1 デバッグ機能の種類

この製品には、プログラムの誤りを発見する手段として、次の3つのデバッグ機能があります。

- 実行したCOBOLの文のトレース(TRACE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の文ごと、文種別ごとの実行回数とその比率を出力(COUNT機能)

デバッグ機能を使うには、COBOLプログラムを翻訳するときに各デバッグ機能の翻訳オプションを指定し、そのプログラムを実行するときにデバッグ機能を動作させるための環境を指定します。

各デバッグ機能の概要と指定する翻訳オプションを"表5.1 デバッグ機能の概要と指定する翻訳オプション"に示します。

表5.1 デバッグ機能の概要と指定する翻訳オプション

機能名	概要		翻訳オプション
	用途	処理	
TRACE機能	・どの文で異常終了したのかを知りたい場合 ・異常終了までに実行した文の経路を知りたい場合 ・実行の途中で出力されたメッセージを確認したい場合	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および動詞番号 ・実行した文を含むプログラム名とプログラム属性情報 ・実行中に出力されたメッセージ	TRACE
CHECK機能	・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	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	CHECK

機能名	概要		翻訳オプション
	用途	処理	
		・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	
COUNT機能	・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	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	COUNT

TRACE機能とCOUNT機能は、同時に使用できません。

なお、対話型デバッガのカバレージ機能およびテスト網羅度測定機能を使用することでより詳細な性能検証の情報収集およびルート走行の確認ができます。

5.1.1 ステートメント番号

以降の説明でステートメント番号と記述した場合には、次の表現を意味します。

```
行番号[.行番号内動詞追番]
```

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1から1きざみに昇順に与えた値となります。

行番号内動詞追番

同じ行番号内に複数の文が書かれた場合を考慮し、各文に一意性を持たせるための追番です。最初の文に対して1、その次に2、3…というように昇順に割り当てられます。

参照

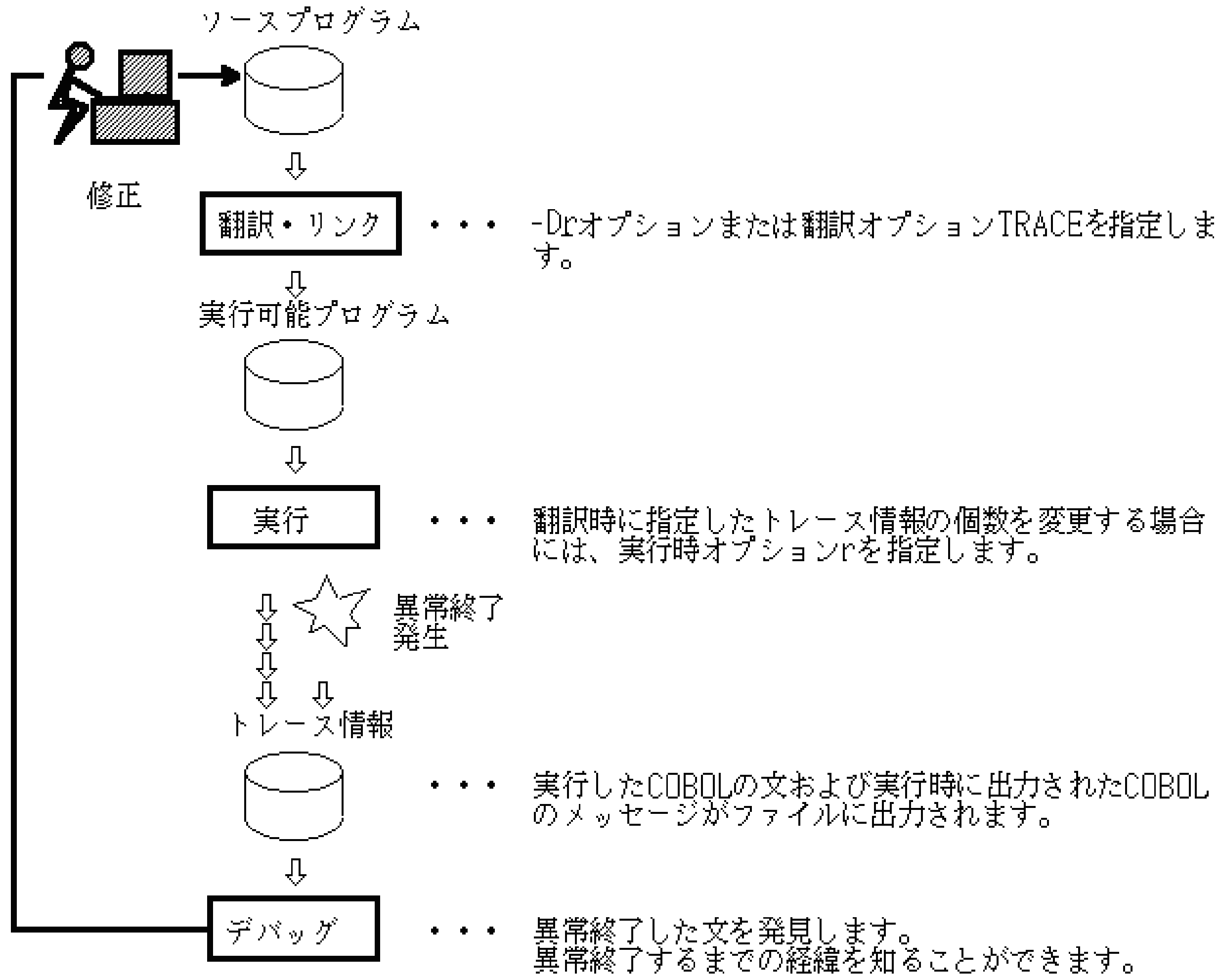
"3.1.7.4 ソースプログラムリスト"

5.2 TRACE機能の使い方

TRACE機能では、プログラムの異常終了時に、それまでに実行したCOBOLの文のトレース情報をファイルに出力します。出力されたトレース情報により、異常終了した文やそこまでの経緯を知ることができるので、デバッグ作業に役立ちます。ここでは、TRACE機能の使い方について説明します。

5.2.1 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。



5.2.2 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-Drオプションまたは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。

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。

トレース情報の格納先

トレース情報は、実行可能プログラムの名前に、拡張子trcを追加したファイル名のファイルに格納されます。以下に例を示します。

実行可能プログラムのファイル名

```
/home/xx/PROG1
```

トレース情報のファイル名(最新情報)

```
/home/xx/PROG1.trc
```

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。

なお、トレース情報のファイル名または格納先を変更する場合には、環境変数CBR_TRACE_FILEを実行時に指定します。

```
CBR_TRACE_FILE = ファイル名
```

トレース情報は環境変数CBR_TRACE_FILEに指定したファイル名に拡張子trcを追加したファイル名のファイルに出力されます。

トレース情報の出力形式

以下にトレース情報の出力形式を示します。

```
NetCOBOL DEBUG INFORMATION                    DATE 2000-04-06  TIME 10:10:32
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
         7          2100.1 TID=00000099
         8          2200.1 TID=00000099
         9    JMPnnnnI-x xxxxxxxxx xx xxxxxxxxx.  [10]
```

[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。詳細は、"メッセージ説明書"を参照してください。

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CBR_TRACE_PROCESS_MODEを実行時に指定します。

```
CBR_TRACE_PROCESS_MODE=MULTI
```

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CBR_TRACE_PROCESS_MODEを指定した場合の例を示します。

例

環境変数CBR_TRACE_PROCESS_MODEを指定した場合

```
実行可能ファイル名:sample.out
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
```

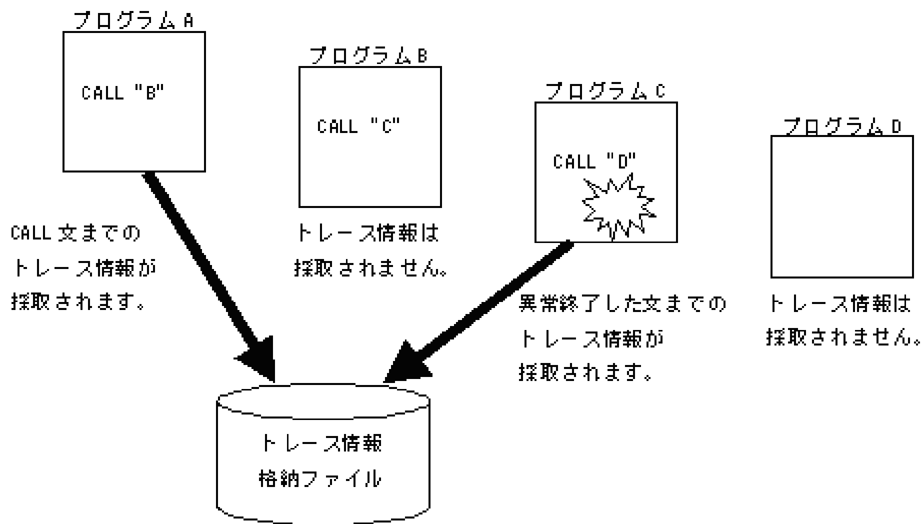
トレース情報ファイル名(一世代前の情報)

```
sample-00000EC4_20100112_104850.tro
```

5.2.3 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、-Drオプションまたは翻訳オプション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、デバッグ終了後には、翻訳オプションNOTRACEを指定して再翻訳してください。
- 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。
- 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。
- 環境変数CBR_TRACE_FILEと環境変数CBR_TRACE_PROCESS_MODEを同時に指定した場合、CBR_TRACE_FILEの指定が優先され、CBR_TRACE_PROCESS_MODEの指定は無効になります。

5.3 CHECK機能の使い方

CHECK機能では、以下の検査を行い、異常を検出するとメッセージを出力し、異常終了します。そのため、プログラムの誤動作を防ぐことができます。

- 添字・指標、部分参照
- 数字のデータ例外および除数ゼロ
- メソッド呼出しのパラメタ

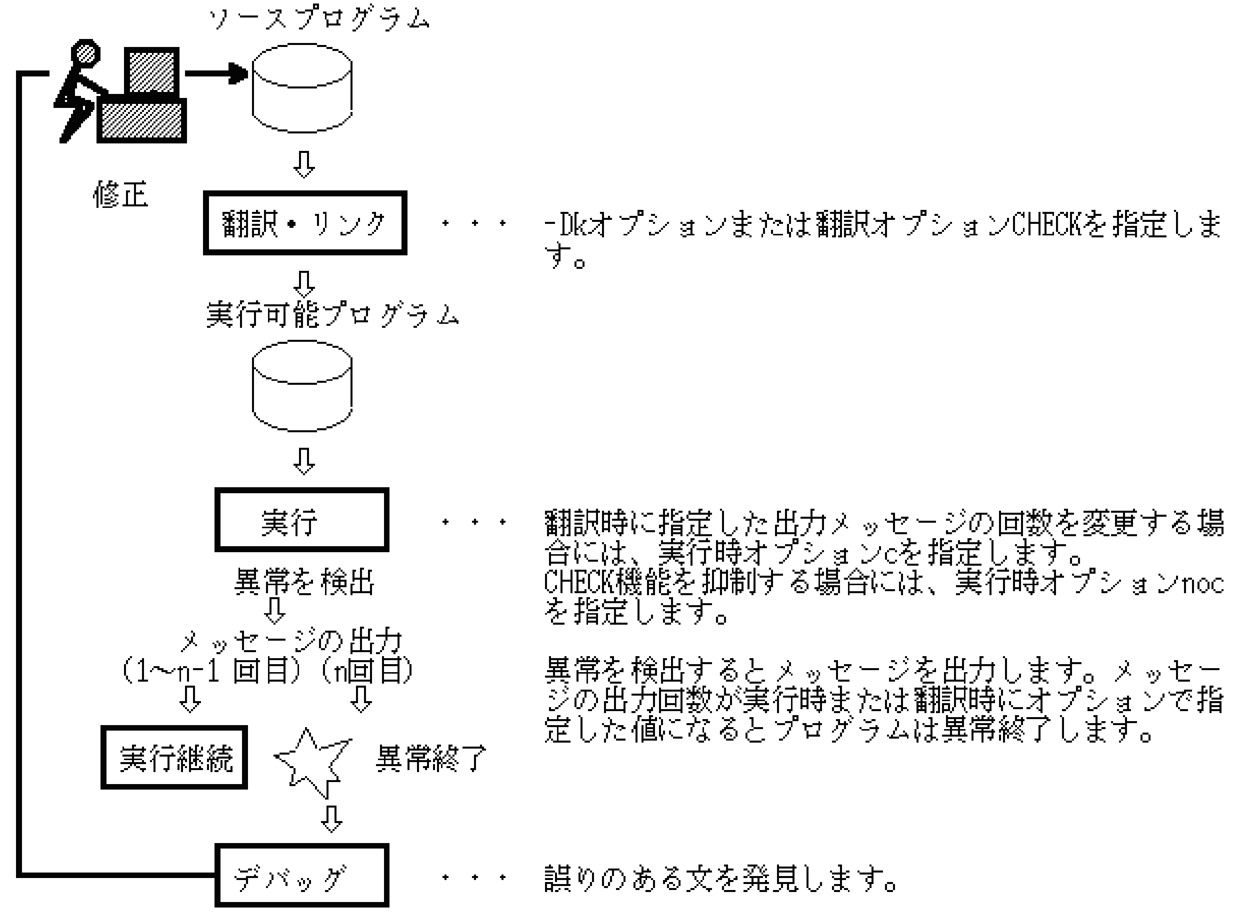
・ 内部プログラム呼出しのパラメタ

・ 外部プログラム呼出しのパラメタ

ここでは、CHECK機能の使い方について説明します。

5.3.1 デバッグ作業の流れ

以下にCHECK機能を使ったデバッグ作業の流れを示します。



5.3.2 出力メッセージ

CHECK機能では、検査によって異常を検出すると、メッセージを出力します。

このメッセージは、通常の実行時メッセージが出力されるところ(標準エラー出力先)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。しかし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。重大度コードおよびメッセージの内容については、“メッセージ説明書”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。

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。

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。

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。

翻訳時に-Dkオプションを指定した場合、メッセージの出力回数は1回です。メッセージの出力回数は、実行時に実行時オプション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。

5.3.3 CHECK機能の使用例

ここでは、CHECK機能の使用例を示します。

添字および指標検査

翻訳オプション CHECK(BOUND)またはCHECK(ALL)

```
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=1200.1. OPD=要素１ (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=1400.1. OPD=要素１ (1)
```

部分参照検査

翻訳オプション CHECK(BOUND)またはCHECK(ALL)

```
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=1200.1. OPD=データ２.
```

OCCURS DEPENDING ON句の目的語検査

翻訳オプション CHECK(BOUND)またはCHECK(ALL)

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

1300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=1300.1.  OPD=要素. ODO=個数.
```

数字のデータ例外検査

翻訳オプション CHECK(NUMERIC)またはCHECK(ALL)

```
000500  01  文字   PIC X(4)   VALUE "ABCD".
000600  01  外部１０進   REDEFINES   文字   PIC S9(4).
000700  01  数字   PIC S9(4).
            :
001500      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=1500.1. OPD=外部１０進.
```

除数のゼロ検査

翻訳オプション CHECK(NUMERIC)またはCHECK(ALL)

```
000600  01  被除数  PIC S9(8)  BINARY  VALUE  1234.
000700  01  除数    PIC S9(4)  BINARY  VALUE  0.
000800  01  結果    PIC S9(4)  BINARY  VALUE  0.
            :
001500      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=1500.1. OPD=除数
```

メソッド呼出しのパラメタの検査

- プログラムA

  翻訳オプション CHECK(ICONF)またはCHECK(ALL)

```
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=70.1
```

内部プログラム呼出しのパラメタの検査

翻訳オプション CHECK(PRM)またはCHECK(ALL)

- 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=10.1.
```

5.3.4 注意事項

ここでは、CHECK機能使用時の注意事項について説明します。

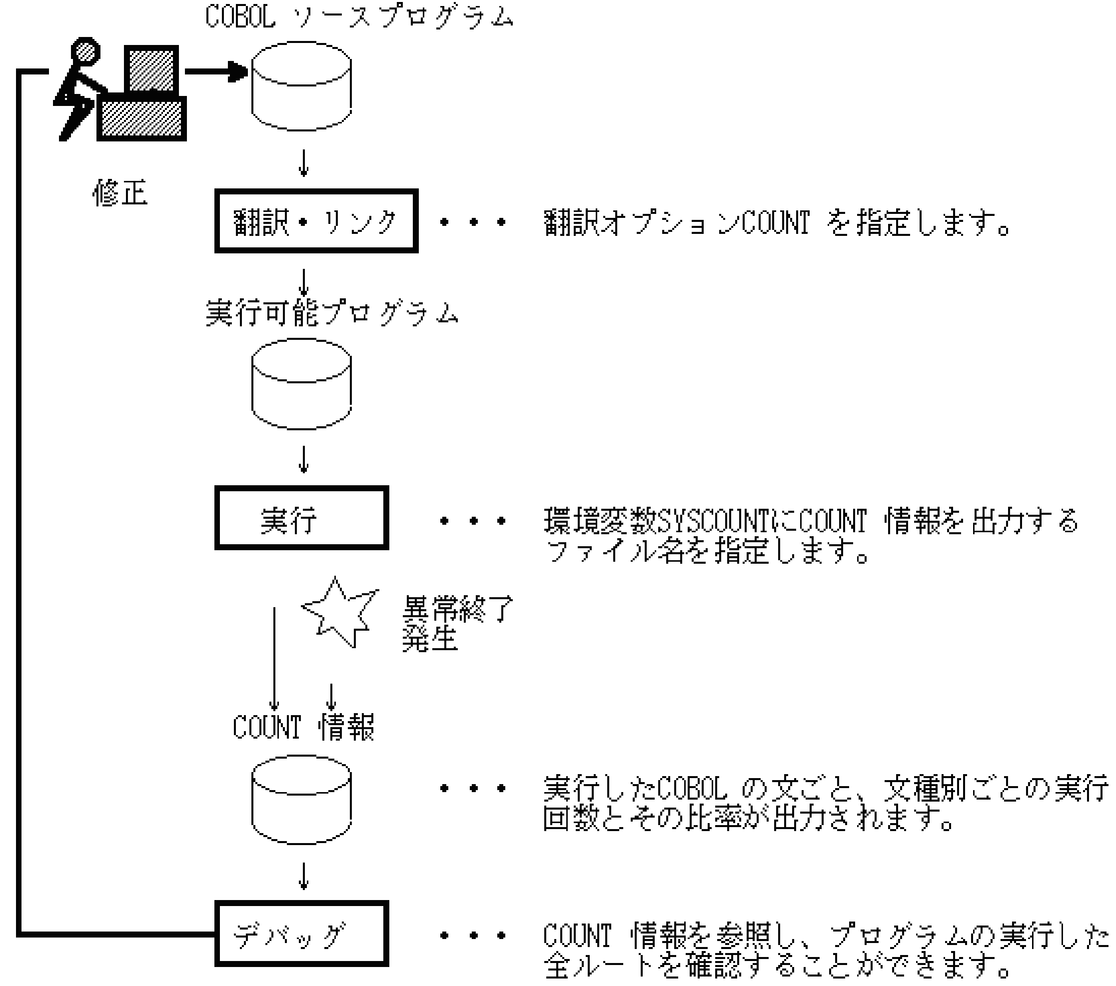
- CHECK機能による検査は必ず実施し、検出された情報を元に誤りを修正してください。検出された誤りが修正されない場合、メモリ破壊などの表面化しにくい重大なトラブルが発生することにつながります。検出された誤りが未修正のままアプリケーションを実行した場合、動作は保証されません。
- メッセージ出力回数を指定することにより、異常検出後も実行を継続することができます。しかし、異常検出後の動作は保証されません。
- CHECK機能では、データ内容の検査など、COBOLプログラムで記述した以外の処理を行います。そのため、CHECK機能使用時には、プログラムのサイズが大きくなり、実行速度も遅くなります。CHECK機能は、デバッグ時にだけ使用し、デバッグ終了後には、翻訳オプション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。
- 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NetCOBOL V7.2以降のコンパイラで翻訳オプションCHECK(PRM)を指定し、翻訳する必要があります。他言語で作成されたプログラムを呼び出すCALL文、または、他言語で作成されたプログラムから呼び出された場合、パラメタの検査は行われません。
- CHECK(PRM)の外部プログラム呼出しの検査で、呼出し元のパラメタ個数と呼出し先のパラメタ個数の差が4個以上の場合、誤りが検出されないことがあります。
- CHECK(PRM)のパラメタの検査では、可変長項目のパラメタの長さは実行時の長さではなく最大長が使われます。そのため、可変長項目の場合は実際にはパラメタの長さが一致していても、メッセージが出力されることがあります。
- CHECK(PRM)の外部プログラム呼出しの検査で、呼出し元または呼出し先のプログラムにRETURNING指定の記述がない場合、暗黙にPROGRAM-STATUSが受け渡されるため、長さ4バイトのRETURNINGパラメタが指定されたものとして検査します。
- CHECK機能は、CHECKオプションを指定して翻訳したプログラムにだけ有効になります。
  複数のプログラムをリンクしている場合に特定のプログラムだけをCHECK機能の対象にするには、対象にするプログラムはCHECKオプションを指定して翻訳し、対象にしないプログラムはCHECKオプションを指定しないで翻訳してください。

5.4 COUNT機能の使い方

COUNT機能は、ソースプログラム上に書かれた各文の実行回数と全文の実行回数に対する各文の実行回数比率を表示する機能です。また、ソースプログラム中に書かれた文種別ごとの実行回数とその比率などを表示します。利用者は、COUNT機能により、各文の実行頻度を的確に把握し、プログラムの最適化に役立てることができます。

5.4.1 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。



5.4.2 COUNT情報

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

参照

"A.2.7 COUNT(COUNT機能の使用の可否)"

COUNT情報の格納先

COUNT情報は、環境変数SYSCOUNTに指定したファイルに格納されます。以下に例を示します。

```
SYSCOUNT=ファイル名[,MOD[,NOLIMIT]]
```

- ファイル名には、COUNT機能を使用する場合にCOUNT情報の出力先となるファイルの名前を指定します。
- 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、カレントディレクトリからの相対パスになります。
- ディレクトリ名またはファイル名にコンマ(,)を含む場合、ディレクトリ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。
- MODはファイルの追加書きを指示します。
  - MODを指定した場合、ファイル名に指定したファイルが存在すれば追加書きします。ファイル名に指定し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を指定しない場合、ファイル名に指定したファイルが存在すれば上書きします。ファイル名に指定し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。

・ 第二引数にMOD以外の文字列、第三引数にNOLIMIT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。

・ NOLIMITが指定されない場合、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。

・ NOLIMITが指定された場合、追加書きするCOUNT情報ファイルのサイズの検査を行いません。ファイルのサイズが1GBを超えた場合、JMP0727I-Wのメッセージは出力しないでCOUNT情報を出力します。ファイルの最大サイズについては、"表6.9 各ファイルシステムの機能差"の大容量ファイルを参照してください。

COUNT情報の出力形式

以下に、COUNT情報の出力形式を示します。

```
[1]
NetCOBOL  COUNT INFORMATION (END OF RUN UNIT)     DATE 2000-04-06  TIME 11:02:19
  PID=000014B1  TID=00000001

   [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 2000-04-06  TIME 11:02:19
  PID=000014B1  TID=00000001

   [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。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。

```
[COPY修飾値-] 行番号
```

[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"の文字列のあと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    [9]÷[10]×100
```

[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。

```
計算式    各文の実行回数÷実行回数×100
```

[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　　[16]÷[17]×100
```

[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
  または
計算式　　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表示します。

```
計算式　　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
```

5.4.3 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。

5.4.4 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、デバッグ終了後には、翻訳オプションNOCOUNTを指定して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SYSCOUNTを指定する必要があります。
- 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。

5.5 メモリチェック機能の使い方

メモリチェック機能は、COBOLアプリケーションの実行時、領域破壊が発生した場合に使用します。メモリチェック機能は、COBOLアプリケーションの手続き部の開始、終了でランタイムシステム領域をチェックします。以下の実行時メッセージが出力された場合またはアプリケーションエラー(セグメンテーション違反)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。

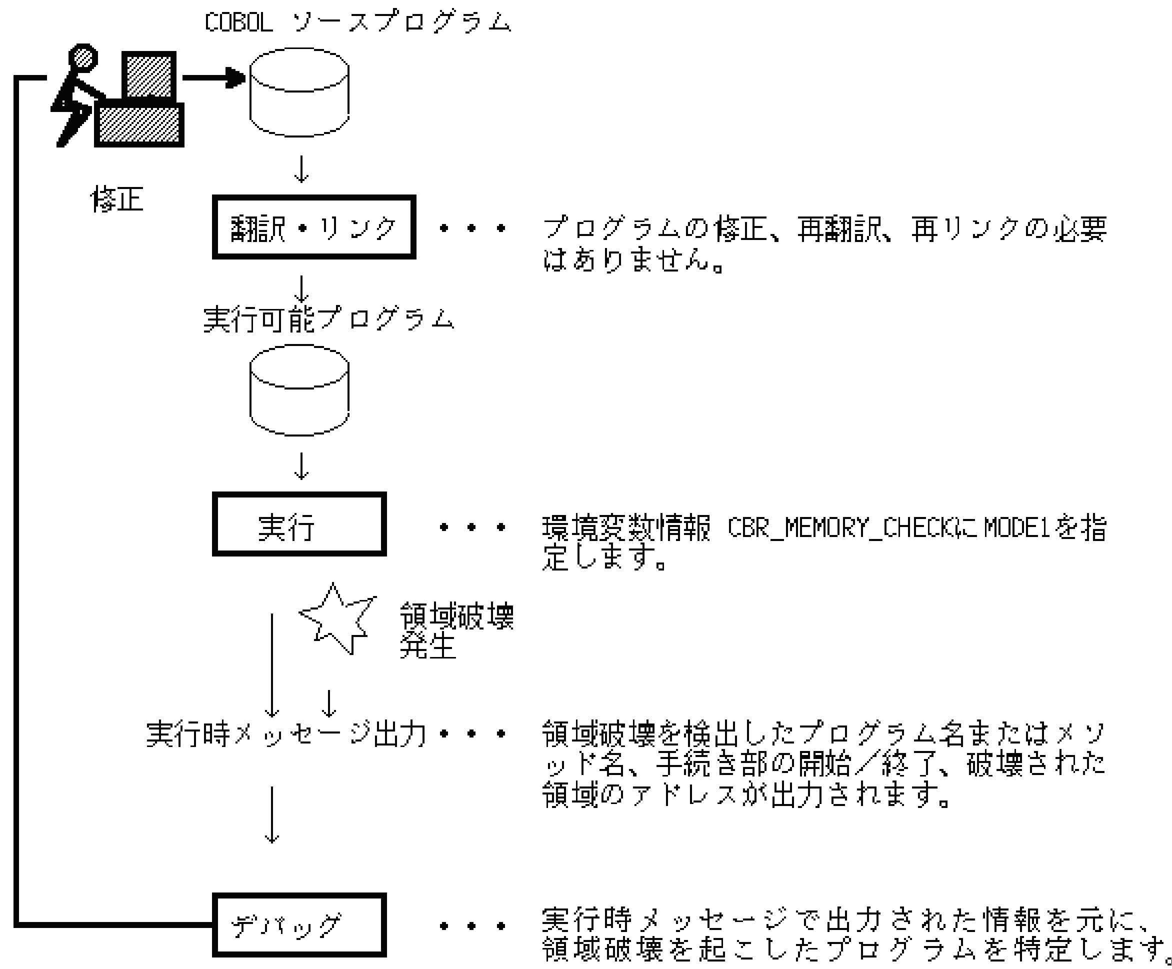
- JMP0009I-U ライブラリ作業域が確保できません.
- JMP0010I-U ライブラリ作業域が破壊されています.

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を指定してください。

```
$ CBR_MEMORY_CHECK=MODE1 ; export CBR_MEMORY_CHECK
```

5.5.1 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、5.5.3 プログラムの特定を参照してください。

5.5.2 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。

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

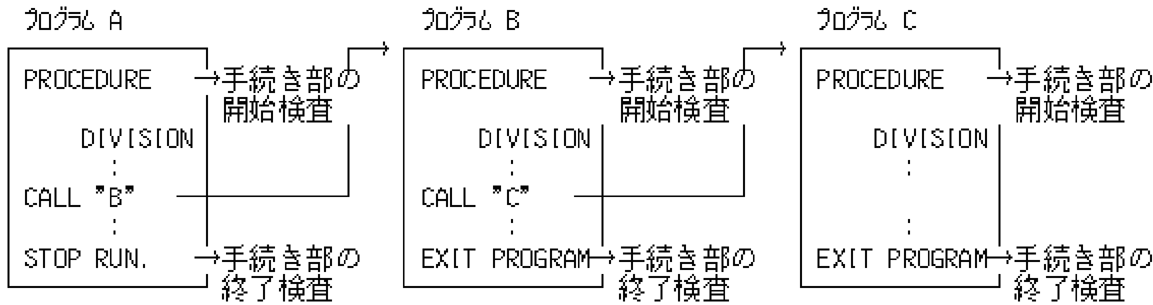
JMP0071I-U
[PID:xxxxxxxx TID:xxxxxxxx] 領域破壊を検出しました. END PGM=プログラム名 BRKADR=破壊された領域の先頭アドレス

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。

5.5.3 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。



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。

5.5.4 注意事項

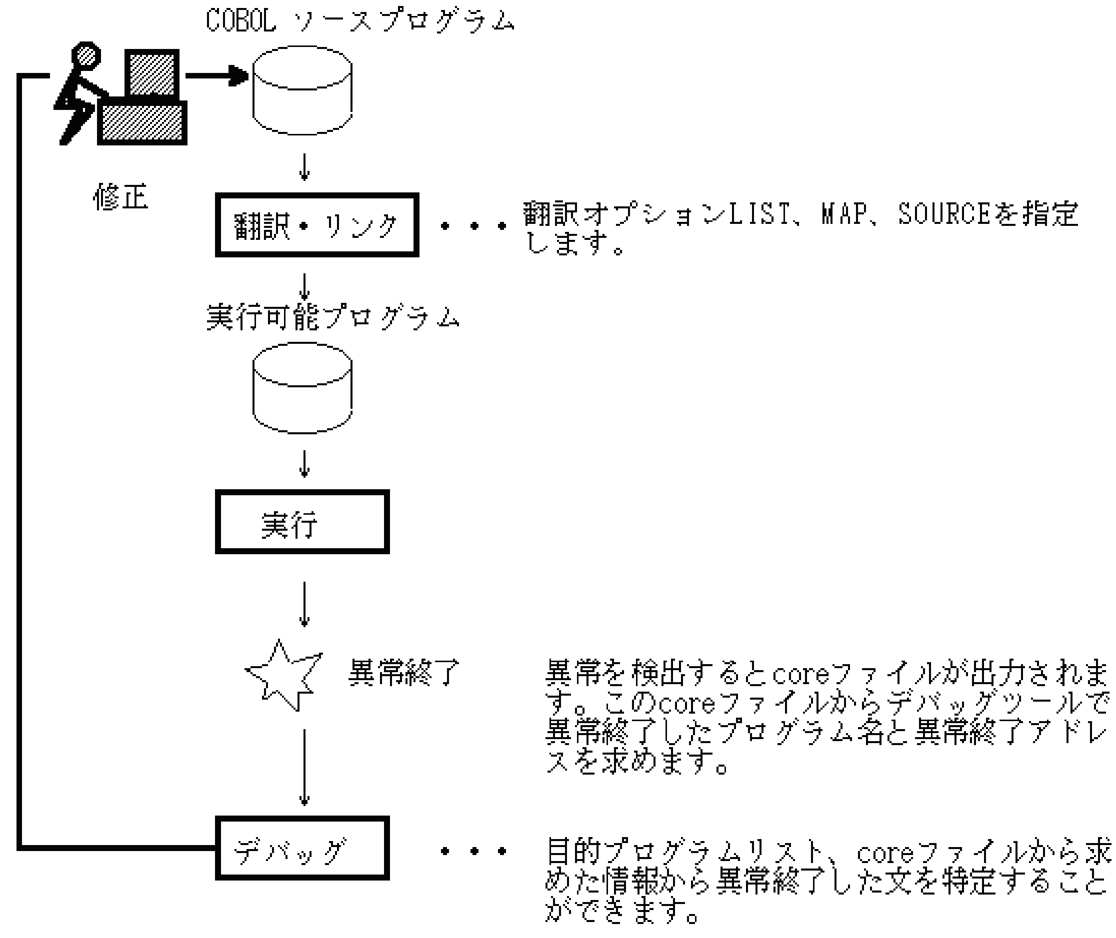
メモリチェック機能では、プログラムの手続き部の開始/終了でランタイムシステム領域の検査を行うため、実行速度が遅くなります。デバッグ終了後には、メモリチェック機能の指定(環境変数情報CBR_MEMORY_CHECK)を無効にしてください。

5.6 異常終了時の障害発生箇所の特定方法

COBOLのデバッグ機能以外に目的プログラムリストとデータマップリストを使用して障害発生箇所を特定することができます。ここでは、簡単な例題から障害発生箇所の特定方法について説明します。

5.6.1 デバッグ作業の流れ

以下に目的プログラムリストとデータマップリストを使った障害発生箇所の特定方法の流れを示します。



5.6.2 障害発生箇所の特定方法

実行時に異常終了したときの障害発生箇所の特定方法について、簡単な例を用いて説明します。目的プログラムリストの形式については“3.1.7.5 目的プログラムリスト”を参照してください。

1. 異常終了情報の取得

   実行時に異常終了した場合、たとえば、以下のようなエラーメッセージが表示されます。

   ```
   浮動小数点例外 （コアダンプしました。）
   ```

   デバッガを使用し、異常終了したアドレスの情報を求めます。求める時の手順の例を以下に示します。(下線部分が入力)

   ```
   % mdb listdbg core
   Loading modules: [ libc.so.1 ld.so.1 ]
   > :r
   mdb: stop on SIGFPE
   mdb: target stopped at:
   LISTDBG1.text+0x2cc:     sdiv       %i4, %o0, %i5
   ```

   デバッガの出力結果から以下の情報を調べます。

   - 例外発生理由 : 上図では、“integer divide by zero”のメッセージが表示されているため“LISTDBG1の0による除算エラーです。”
   - 異常終了したオブジェクト: 上図では、プログラムLISTDBG1を含むオブジェクト(ここでは“listdbg1.o”とします)
   - 異常終了したアドレス: 上図では、“LISTDBG1+0x288”または“LISTDBG1.text+0x2cc”

2. 目的プログラムリスト内の基点オフセット

1.で調べた異常終了したアドレスの基点である入口名“LISTDBG1”の目的プログラムリストの先頭からの相対アドレスを求めます。

ただし、デバッガのバージョンによっては、“LISTDBG1.text”のように“.text”が付加されることがあります。この場合は翻訳リストのオフセットをそのまま使用することができます。

```
  番地     機械語  手続き名  アセンブラ形式命令 BEGINNING OF LISTDBG1
                            .section ".text"
 00000000 00 42              .half   0x0042
 00000002 08                .byte   0x08
 00000003 4C495354 44424731 .ascii  "LISTDBG1"
 0000000B 08                .byte   0x08
 0000000C 4E657443 4F424F4C .ascii  "NetCOBOL"
 00000014 07                .byte   0x07
 00000015 56372E304C3130    .ascii  "V7.0L10"
 0000001C 32303034 30393237 .ascii  "20040927"
 00000024 31363234 35323030 .ascii  "16245200"
 0000002C 2B303930 30       .ascii  "+0900"
 00000031 2031              .ascii  " 1"
 00000033 40282329 434F424F .ascii  "@(#)COBOL_SINGLE>"
 00000044 03XXXXXX          sethi   %hi(0xXXXXXXXX),%g1  ←基点オフセット
 00000048 8210XXXX          or      %g1,0xXXXX,%g1
 0000004C 30XXXXXX          ba    PLB.1
 00000050 01000000          nop
 0000004C 00000002
                                    :
```

プログラムLISTDBG1の入口点、つまり上図のsethi命令が存在するアドレスが、求める基点オフセットとなります。したがって、ここでは0x44です。

3. 異常終了アドレスの目的プログラムリスト内相対オフセットの求め方

異常終了アドレスの目的プログラムリスト内相対オフセットは、以下の式で求めること ができます。

― 異常終了アドレスが“入口名+相対アドレス”の形式の場合

```
異常終了アドレスの目的プログラムリスト内相対オフセット  =
                  基点オフセット  +  異常終了の相対アドレス
```

― 異常終了アドレスが“入口名.text+相対アドレス”の形式の場合

```
異常終了アドレスの目的プログラムリスト内相対オフセット  =
                  異常終了の相対アドレス
```

この例の場合、異常終了アドレスは、1.で異常終了した場合に表示される“LISTDBG1+0x288”です。

目的プログラムリスト内の基点オフセットは、2.で調査した異常終了しているオブジェクト“listdbg1.o”の開始アドレスである“0x44”です。

したがって、計算式は以下のとおりです。

```
      基点オフセット                 00000044
  +   異常終了の相対アドレス         00000288
 ------------------------------------------------
      オブジェクト内相対オフセット   000002CC
```

4. 異常終了したCOBOLの文の特定

異常終了アドレスのオブジェクト内相対オフセットを目的プログラムリスト内のオブジェクト内相対オフセットから探し、異常終了したアセンブラ命令を探します。

異常終了したアセンブラ命令の位置から目的プログラムリスト内に出力されているCOBOLソースの記述行とCOBOLの文を探します。

```
 --- 12 ---                MOVE
0000022C 3B0C0C4C            sethi    %hi(0x30313000),%i5
```

```
00000230 BA176030          or       %i5,0x0030,%i5
00000234 FA27E058          st       %i5,[%i7+0x058] :被除数
 --- 14 ---              COMPUTE
00000238 820F60F0          and      %i5,0x00f0,%g1
0000023C 090003C0          sethi    %hi(0x000f0000),%g4
00000240 8811200F          or       %g4,0x000f,%g4
00000244 91376008          srl      %i5,0x0008,%o0
00000248 B80F4004          and      %i5,%g4,%i4
0000024C 900A0004          and      %o0,%g4,%o0
00000250 B8070008          add      %i4,%o0,%i4
00000254 B8070008          add      %i4,%o0,%i4
00000258 912A2003          sll      %o0,0x0003,%o0
0000025C B8070008          add      %i4,%o0,%i4
00000260 9137200E          srl      %i4,0x000e,%o0
00000264 B80F20FF          and      %i4,0x00ff,%i4
00000268 B8070008          add      %i4,%o0,%i4
0000026C 912A2003          sll      %o0,0x0003,%o0
00000270 B8070008          add      %i4,%o0,%i4
00000274 B8070008          add     %i4,%o0,%i4
00000278 B8070008          add      %i4,%o0,%i4
0000027C 80A06050          subcc    %g1,+0x050,%g0
00000280 22XXXXXX          be,a     GLB.3
00000284 B820001C          sub      %g0,%i4,%i4
00000288            GLB.3:
00000280 00000002
00000288 D205E07C          ld       [%l7+0x07c],%o1 :BVA.1
0000028C FA0A6000          ldub     [%o1+0x000],%i5 :除数
00000290 C20A6001          ldub     [%o1+0x001],%g1 :除数+1
00000294 BB2F6008          sll      %i5,0x0008,%i5
00000298 BA174001          or       %i5,%g1,%i5
0000029C 820F60F0          and      %i5,0x00f0,%g1
000002A0 89376007          srl      %i5,0x0007,%g4
000002A4 900F600F          and      %i5,0x000f,%o0
000002A8 8809201E          and      %g4,0x001e,%g4
000002AC 90020004          add      %o0,%g4,%o0
000002B0 89292002          sll      %g4,0x0002,%g4
000002B4 90020004          add      %o0,%g4,%o0
000002B8 80A06050          subcc    %g1,+0x050,%g0
000002BC 22XXXXXX          be,a     GLB.4
000002C0 90200008          sub      %g0,%o0,%o0
000002C4  GLB.4:
000002BC 00000002
000002C4 BB3F201F          sra      %i4,0x001f,%i5
000002C8 81874000          wry      %i5,%g0,%g0
000002CC BA7F0008          sdiv     %i4,%o0,%i5       ←ここで異常終了
000002D0 96A76000          subcc    %i5,+0x000,%o3
000002D4 26XXXXXX    bl,a    GLB.5
```

目的プログラム内に記述されている記述行とCOBOLの文をソースプログラムリスト上から探します。記述行に対応するCOBOLの文が異常終了していることがわかります。

```
               行番号一連番号A   B
 1    000010 IDENTIFICATION DIVISION.
 2    000020  PROGRAM-ID. LISTDBG1.
 3    000030*
 4    000040 DATA DIVISION.
 5    000050  WORKING-STORAGE SECTION.
 6    000060   77  答え PIC 9(8).
 7    000070   77  被除数 PIC 9(4).
 8    000080  LINKAGE SECTION.
 9    000090   01  除数 PIC 9(2).
10    000100
11    000110 PROCEDURE DIVISION USING 除数.
```

```
12   000120      MOVE 100 TO 被除数.
13   000130*
14   000140      COMPUTE 答え = 被除数 / 除数.  ←この文で異常終了
15   000150*
16   000160      EXIT PROGRAM.
17   000170 END PROGRAM LISTDBG1.
```

例の場合、3.で求めたモジュール内相対オフセット"0x2CC"を目的プログラムリストから参照すると、異常終了したアセンブラ命令が"sdiv"であることがわかります。その異常終了したアセンブラ命令の位置から、COBOLソースの記述行が"---14---"行目で、COBOLの文が"COMPUTE"であることがわかります。

異常終了した場所がプログラムまたはメソッドの場合、デバッガを使用して、そのプログラムまたはメソッドのデータを参照することができます。以降では、データの参照方法について説明します。

1. データのアドレスの取得

   データマップリストから、データが存在するアドレスの情報を求めます。データマップリストの形式は"3.1.7.6 データエリアに関するリスト"を参照してください。

```
行番号  番地       オフセット  変位 レベル 名標     長さ(10)  属性      基点 次元数
**LISTDBG1**
  6  i7+00000050                0 01 答え            8 EXT-DEC    BGW.000000
  7  i7+00000058                0 01 被除数          4 EXT-DEC    BGW.000000
  9 [l7+0000007C]+00000000      0 01 除数            2 EXT-DEC    BVA.000001
```

   それぞれのデータは以下のアドレスに存在します。

```
答え    %i7+00000050から8バイト
被除数  %i7+00000058から4バイト
除数    %l7+0000007Cに格納されているアドレス(4バイト)にオフセットを加えたアドレスから2バイト
```

2. データの参照

   デバッガを使用して、5.で求めたアドレスの内容を参照します。データマップリストは16進表記で記述されているため、mdbでは16進数を表す文字(0x)を付加します。手順の例を以下に示します。(下線部分が入力)

```
% mdb listdbg core
Loading modules: [ libc.so.1 ld.so.1 ]
> <i7+0x58/X                           ←被除数の領域を4バイト表示          (*1)
listdbg`LISTDBG1.data+0x58:      30313030 ←被除数には'0100'が格納されている

> <l7+0x7c/X                           ←除数のアドレスを表示
listdbg`LISTDBG1.bss+0x7c:       210e8
> 0x210e8+0x00000000/X                 ←除数の領域を4バイト表示
listdbg`LISTDBG.data+0x58:       30300000 ←除数には'00'が格納されている

> <i7+0x50/2X                          ←答えの領域を8バイト表示          (*2)
listdbg`LISTDBG1.data+0x50:      0        0 ←答えにはX'0000000000000000'が格納されている
```

   *1: <i7+0x58/X のXは、4バイトを表示することを指定しています。

   *2: <i7+0x50/2X の2Xは、X(4バイト)を2個表示することを指定しています。

注意

- 参照することができるデータは、異常終了したプログラムまたはメソッドで使用するデータのみです。他のプログラムおよびメソッドのデータは参照することができません。
- 翻訳オプションNOOPTIMIZEを指定していない場合、最適化処理により、実行結果が領域に格納されていないことがあります。そのため、上記の方法でデータが正しく参照できないことがあります。

第6章 ファイル処理

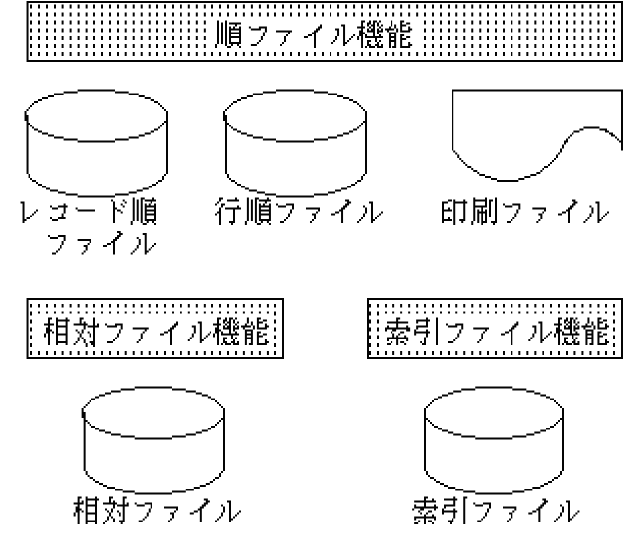
本章では、ファイルからデータを読み込んだり、ファイルにデータを書き出したりする処理およびほかのファイルシステムの使用法について説明します。

6.1 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。

6.1.1 ファイルの種類と特徴

この製品では、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下表に示します。

表6.1 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(注1)	ハードディスク(注1)	印刷装置 ハードディスク(注1)	ハードディスク(注1)	ハードディスク(注1)
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
ファイルの最大サイズ(バイト数)	1G(注2)	1G(注2)	1G(注3)	1G	1.7G

注1: 仮想デバイス(/dev/nullなど)は使用できません。
注2: ファイルの高速処理を指定した場合、ファイルの最大サイズは2Gになります。
注3: 使用できる媒体をハードディスクにした場合です。

注意

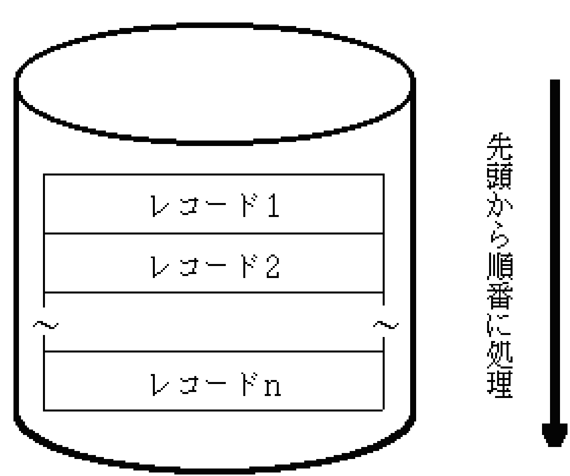
ファイルの最大サイズは、COBOLファイルシステムを利用して処理した場合です。大容量ファイルやその他のファイルシステムを利用した場合のファイルの最大サイズについては、"表6.9 各ファイルシステムの機能差"を参照してください。また、ファイルの高速処理については"6.8.4.2 ファイルの高速処理"を参照してください。

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。

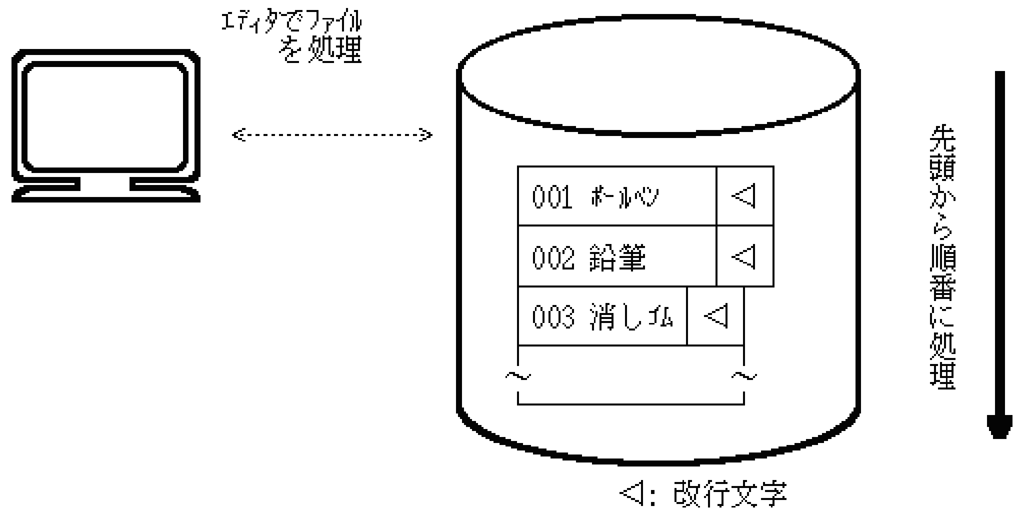
レコード順ファイルは、最も簡単に扱うことのできるファイルで、データを順次蓄積する場合や、大量データを保存する場合などに効果的です。



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ファイルを操作するときなどに利用します。



改行文字は、1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。

```
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。

印刷ファイル

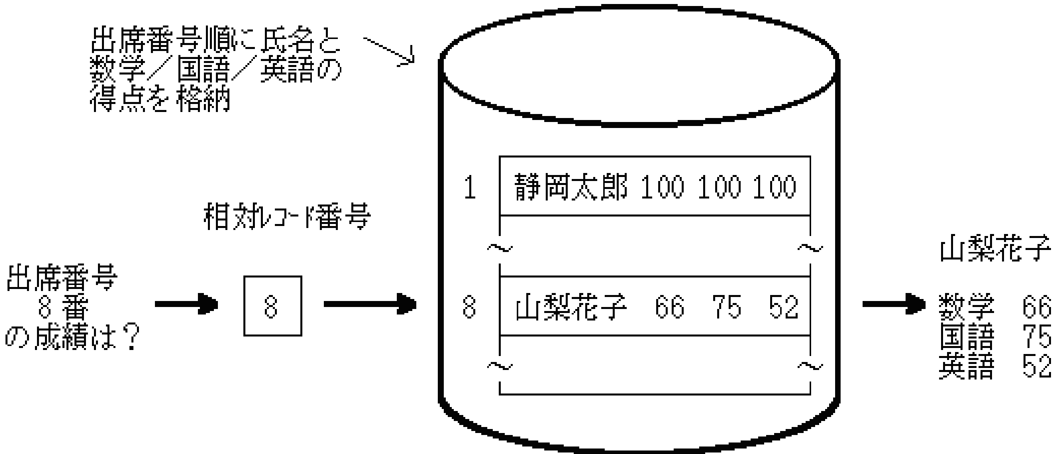
印刷ファイルとは、順ファイル機能を使用した印刷するためのファイルのことで、印刷ファイルという特別なファイルがあるわけではありません。

印刷ファイルについては、“第7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。

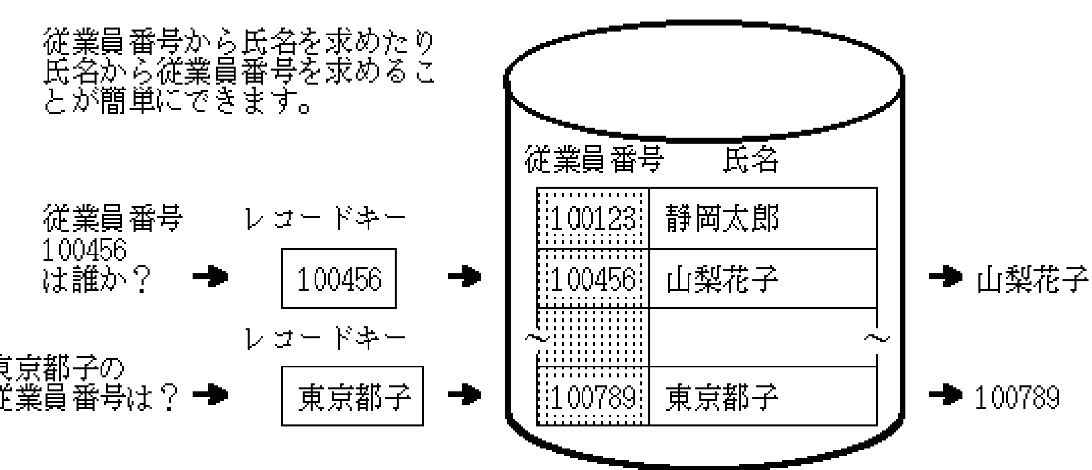
相対ファイルは、相対レコード番号をキーとしてアクセスする作業ファイルなどに利用します。



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ほかの情報を引き出すマスタファイルなどに利用します。



6.1.2 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。

6.1.2.1 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、ファイルサイズを小さくする場合に有効です。

6.1.2.2 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、主レコードキーと副レコードキーがあり、ファイル中のレコードは主レコード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、主レコードキーおよび副レコード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、主レコードキーは、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(レコードの先頭からの位置が常に一定のところ)に設定する必要があります。

6.1.3 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。

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ろ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。

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6.2 ファイルの種類と処理"に示します。

表6.2 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	○	○	×	×	×	×
相対ファイル/索引ファイル	順呼出し	○	○	×	○	○	○
	乱呼出し	○	×	○	○	○	○
	動的呼出し	○	×	○	○	○	○

○:処理可能
×:処理不可能

また、ファイル処理では、ファイル自体を排他モードにしたり、使用中のレコードを排他状態にすることにより、ほか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、"6.7.3 ファイルの排他制御"で説明します。

6.2 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.
```

6.2.1 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い場合や、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、“6.8.4.2 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6.3 SELECT句およびASSIGN句の記述例”に示します。

表6.3 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 "/home/data"	－
DISK	SELECT data1 ASSIGN TO DISK	ファイル名を英小文字で記述した場合、翻訳時に翻訳オプションNOALPHALを指定してください。翻訳オプションALPHALが有効な場合は、カレントディレクトリにあるDATA1を処理します。ファイル名に絶対パス名を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。

6.2.2 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。

以下に、レコードの定義例を示します。

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.
        :
```

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。

6.2.3 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 CLOSE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。
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、ファイルにレコードを書き出していきます。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。

ファイルの創成は、次の手順で行います。

```
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.
```

1. OUTPUT指定のOPEN文でファイルをオープンします。
2. 書き出すレコードの内容を設定します。
3. WRITE文でレコードを書き出します。
4. 2.～3.を繰り返し、すべてのレコードを書き出したら、CLOSE文でファイルをクローズします。

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、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。

ファイルの拡張は、次の手順で行います。

```
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.
```

1. EXTEND指定のOPEN文でファイルをオープンします。
2. 書き出すレコードの内容を設定します。
3. WRITE文でレコードを書き出します。
4. 2.～3.を繰り返し、すべてのレコードを書き出したら、CLOSE文でファイルをクローズします。

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、ファイルのレコードを先頭から順番に読み込みます。

レコードの参照は、次の手順で行います。

```
OPEN INPUT  ファイル名.
READ   ファイル名  ～.
CLOSE  ファイル名.
```

1. INPUT指定のOPEN文でファイルをオープンします。
2. READ文で、先頭から順番にレコードを読み込みます。
3. 2.を繰り返し、必要なレコードをすべて読み終わったら、CLOSE文でファイルをクローズします。

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、“6.6 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、ファイルのレコードを書き換えます。

レコードの更新は、次の手順で行います。

```
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.
```

1. I-O指定のOPEN文でファイルをオープンします。
2. READ文で、先頭から順番にレコードを読み込みます。
3. 読み込んだレコードの内容を更新します。
4. REWRITE文で、更新したレコードを書き出します。
5. 2.～4.を繰り返し、必要なレコードをすべて更新したら、CLOSE文でファイルをクローズします。

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。

6.3 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.
```

6.3.1 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。ファイル名とファイル参照子については、“6.2.1 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。

6.3.2 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レコード形式とレコード長の説明と同じです。“6.2.2 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

レコード中のあるデータの属性、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)	く

く : 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.
```

6.3.3 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 CLOSE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。
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、ファイルにレコードを書き出していきます。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。

ファイルの創成は、次の手順で行います。

```
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.
```

1. OUTPUT指定のOPEN文でファイルをオープンします。
2. 書き出すレコードの内容を設定します。
3. WRITE文でレコードを書き出します。
4. 2.～3.を繰り返し、すべてのレコードを書き出したら、CLOSE文でファイルをクローズします。

注意

- レコードの書出しでは、レコードの領域の内容と改行文字が書き出されます。
- レコード内の後置空白を取り除いてレコードを書き出す場合は、環境変数CBR_TRAILING_BLANK_RECORDに文字列“REMOVE”を指定します。[参照]“6.8.5.2 行順ファイルの後置空白に関する指定”

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、ファイル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。

ファイルの拡張は、次の手順で行います。

```
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.
```

1. EXTEND指定のOPEN文でファイルをオープンします。
2. 書き出すレコードの内容を設定します。
3. WRITE文でレコードを書き出します。
4. 2.～3.を繰り返し、すべてのレコードを書き出したら、CLOSE文でファイルをクローズします。

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CBR_TRAILING_BLANK_RECORDに文字列“REMOVE”を指定します。[参照]“6.8.5.2 行順ファイルの後置空白に関する指定”

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、ファイルのレコードを先頭から順番に読み込みます。

レコードの参照は、次の手順で行います。

```
OPEN INPUT  ファイル名.
READ   ファイル名  ～.
CLOSE  ファイル名.
```

1. INPUT指定のOPEN文でファイルをオープンします。
2. READ文で、先頭から順番にレコードを読み込みます。
3. 2.を繰り返し、必要なレコードをすべて読み終わったら、CLOSE文でファイルをクローズします。

注意

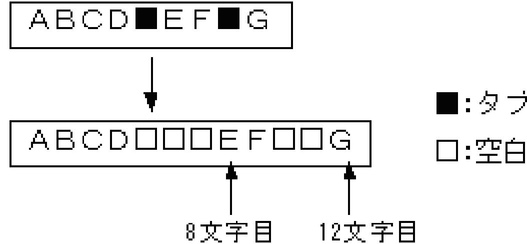
- ファイル管理記述項のSELECT句にOPTIONALを指定した場合、OPEN文実行時にファイルが存在していなくても、OPEN文は成功となり、最初のREAD文の実行でファイル終了条件が成立します。ファイル終了条件については、"6.6 入出力エラー処理"を参照してください。
- 読み込んだレコードの大きさがレコード長より大きいときには、1回のREAD文の実行でレコード長と同じ長さのデータがレコード領域に設定されます。次のREAD文の実行では、同じレコードのデータの続きから、データがレコード領域に設定されます。設定するデータがレコード長より小さい場合には、レコード領域の残りの領域に、空白が設定されます。

レコード内のタブの扱い

読み込んだレコードにタブが存在した場合、以下のように空白が設定されます。

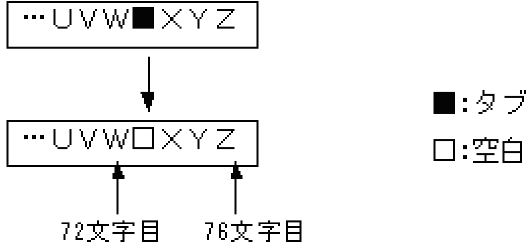
72文字以内にタブが存在する場合

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



72文字を超えた位置にタブが存在する場合

1文字の空白が設定されます。



レコード内の制御文字の扱い

読み込むレコードに制御文字が含まれている場合の動作について、以下に説明します。

制御文字	制御文字の意味	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。

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}0x0A
BEFORE	n LINES (*1)	{レコードデータ}0x0A … 0x0A (0x0A … 0x0A: n個)
	PAGE	{レコードデータ}0x0C
AFTER	n LINES (*1)	0x0A … 0x0A{レコードデータ}0x0D (0x0A … 0x0A: 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

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

Unicodeの行順ファイルを参照する場合

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

使い方

実行時に環境変数CBR_FILE_BOM_READを指定します。

$ CBR_FILE_BOM_READ = { CHECK / DATA / AUTO } ; export CBR_FILE_BOM_READ

(DATA is underlined as the default.)

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。

注意

Windows系システムで作成されたUnicodeのテキストファイルには、ファイルの先頭にBOMと呼ばれる識別コードが付加されています。このようなテキストファイルを本システムで利用する場合は、環境変数CBR_FILE_BOM_READに“CHECK”または“AUTO”を指定してください。

6.3.4 注意事項

行順ファイルの日本語項目に対して、1文字が3バイトのコードである拡張漢字、拡張非漢字および利用者定義文字は使用できません。

行順ファイルで、拡張漢字、拡張非漢字および利用者定義文字を使用したい場合は、COBOLが提供する関数mbston16sおよびn16stombsを使用してコード変換し、英数字項目として処理してください。なお、変換関数の使用方法は、“付録G COBOLが提供するサブルーチン”を参照してください。

6.4 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].
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    PIC 9(5) BINARY.]
 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.]
    [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.
```

6.4.1 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、“6.2.1 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

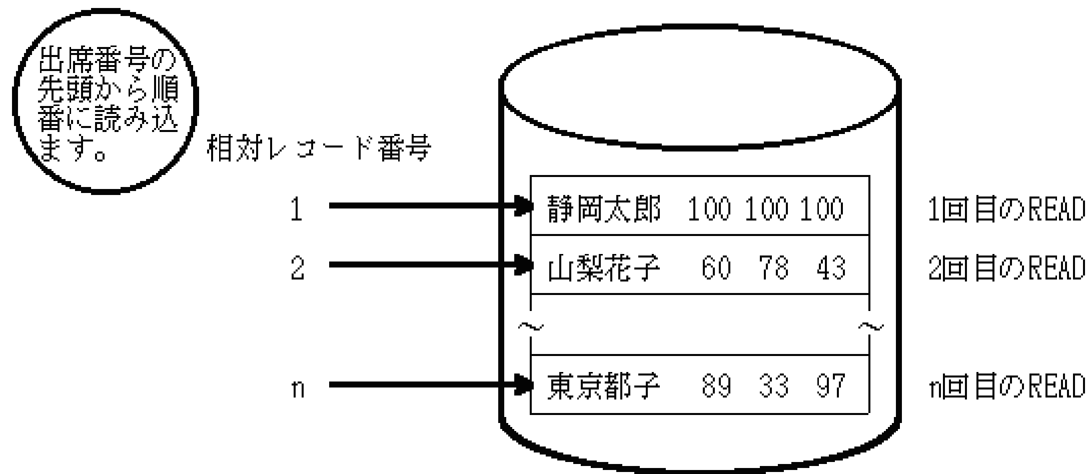
ORGANIZATION句にRELATIVEを指定します。

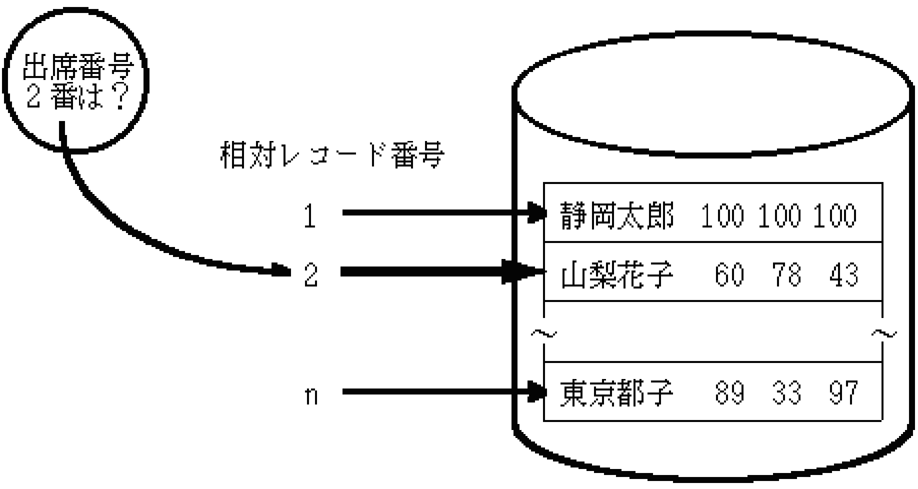
アクセス形態

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。





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。

6.4.2 相対ファイルのレコードの定義

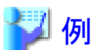
ここでは、レコード形式とレコード長およびレコードの構成について説明します。

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。“6.2.2 レコード順ファイルのレコードの定義”を参照してください。

レコードの構成

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

 例

固定長レコード形式のレコードの定義例

氏名 (10 文字の日本語文字)	国語 (3桁の数字)	数学 (3桁の数字)	英語 (3桁の数字)

```
FD   学級ファイル.
 01   成績レコード.
       02  氏名        PIC N(10).
       02  国語        PIC 9(3).
       02  数学        PIC 9(3).
       02  英語        PIC 9(3).
```

6.4.3 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 CLOSE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。
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。

ファイルの創成は、次の手順で行います。

```
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.
```

1. OUTPUT指定のOPEN文でファイルをオープンします。
2. 書き出すレコードの内容を設定します。
3. 乱呼出しまたは動的呼出しの場合、RELATIVE KEY句に指定したデータ名に相対レコード番号を設定します。
4. WRITE文でレコードを書き出します。
5. 2.～4.を繰り返し、すべてのレコードを書き出したら、CLOSE文でファイルをクローズします。

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、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。

ファイルの拡張は、次の手順で行います。

```
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.
```

1. EXTEND指定のOPEN文でファイルをオープンします。
2. 書き出すレコードの内容を設定します。
3. WRITE文でレコードを書き出します。
4. 2.～3.を繰り返し、すべてのレコードを書き出したら、CLOSE文でファイルをクローズします。

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、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.
```

1. INPUT指定のOPEN文でファイルをオープンします。
2. 先頭のレコード以外から処理を開始したい場合、RELATIVE KEY句に指定したデータ名に処理を開始したいレコードの相対レコード番号を設定し、START文を実行します。
3. READ文で、相対レコード番号の昇順にレコードを読み込みます。動的呼出しの場合は、READ文にNEXT指定を記述します。
4. 3.を繰り返し、必要なレコードをすべて読み終わったら、CLOSE文でファイルをクローズします。

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.
```

1. INPUT指定のOPEN文でファイルをオープンします。
2. RELATIVE KEY句に指定したデータ名に、参照するレコードの相対レコード番号を設定します。
3. READ文で、2.で設定した相対レコード番号を持つレコードを読み込みます。
4. 2.～3.を繰り返し、必要なレコードをすべて読み終わったら、CLOSE文でファイルをクローズします。

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、“6.6 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、“6.6 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.
```

1. I-O指定のOPEN文でファイルをオープンします。
2. 先頭のレコード以外から処理を開始したい場合、RELATIVE KEY句に指定したデータ名に処理を開始したいレコードの相対レコード番号を設定し、START文を実行します。
3. READ文で、相対レコード番号の昇順にレコードを読み込みます。動的呼出しの場合は、READ文にNEXT指定を記述します。
4. 読み込んだレコードを更新します。
5. REWRITE文で、更新したレコードを書き出します。
6. 3.～5.を繰り返し、必要なレコードをすべて更新したら、CLOSE文でファイルをクローズします。

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.
```

1. I-O指定のOPEN文でファイルをオープンします。
2. RELATIVE KEY句に指定したデータ名に、更新するレコードの相対レコード番号を設定します。
3. 必要であればREAD文で、2.で設定した相対レコード番号を持つレコードを読み込みます。
4. レコードを更新または編集します。
5. REWRITE文で、更新または編集したレコードを書き出します。
6. 2.～5.を繰り返し、必要なレコードをすべて更新したら、CLOSE文でファイルをクローズします。

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、“6.6 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.
```

1. I-O指定のOPEN文でファイルをオープンします。
2. 先頭のレコード以外から処理を開始したい場合、RELATIVE KEY句に指定したデータ名に処理を開始したいレコードの相対レコード番号を設定し、START文を実行します。
3. READ文で、相対レコード番号の昇順にレコードを読み込みます。動的呼出しの場合は、READ文にNEXT指定を記述します。
4. DELETE文で、読み込んだレコードを削除します。
5. 3.～4.を繰り返し、不要なレコードをすべて削除したら、CLOSE文でファイルをクローズします。

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.
```

1. I-O指定のOPEN文でファイルをオープンします。
2. RELATIVE KEY句に指定したデータ名に、削除するレコードの相対レコード番号を設定します。
3. DELETE文で、レコードを削除します。
4. 2.～3.を繰り返し、不要なレコードをすべて削除したら、CLOSE文でファイルをクローズします。

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、“6.6 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.
```

1. I-O指定のOPEN文でファイルをオープンします。
2. 挿入するレコードの内容を設定します。
3. RELATIVE KEY句に指定したデータ名に挿入するレコードの相対レコード番号を設定します。
4. WRITE文でレコードを挿入します。
5. 2.～4.を繰り返し、すべてのレコードを挿入したら、CLOSE文でファイルをクローズします。

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、"6.6 入出力エラー処理"を参照してください。

例

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。

```
MOVE  1  TO  データ名.
```

6.5 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.
```

6.5.1 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、“6.2.1 レコード順ファイルの定義”を参照してください。

ファイル編成

ORGANIZATION句にINDEXEDを指定します。

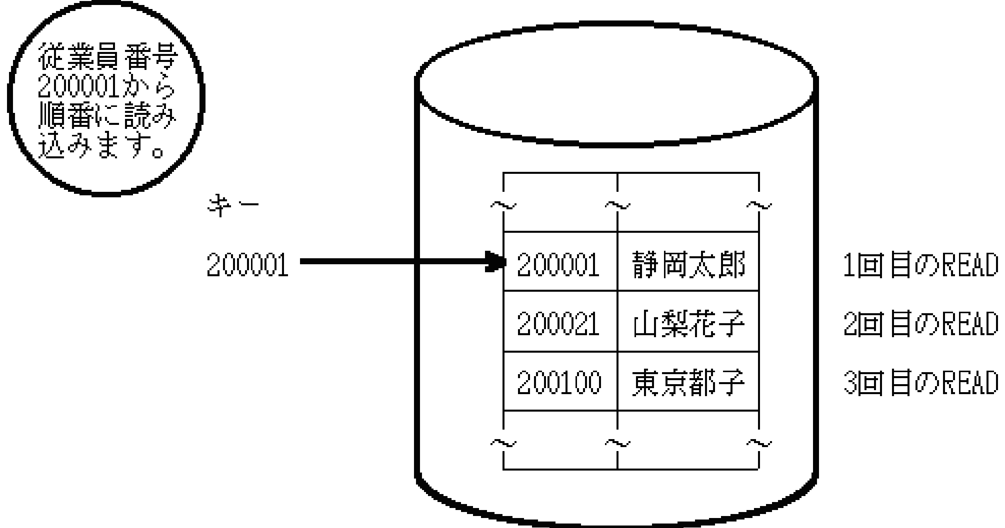
アクセス形態

ACCESS MODE句に以下のアクセス形態のどれかを指定します。

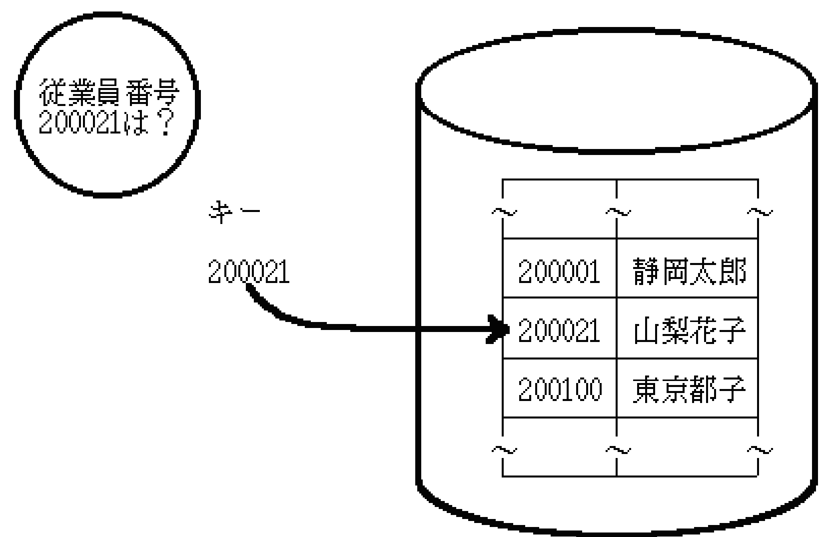
順呼出し法(SEQUENTIAL)	ファイルの先頭またはある特定の値のキーを持つレコードから、キーの昇順にレコードを処理することができます。
乱呼出し法(RANDOM)	ある特定の値のキーを持つレコードだけを単独に処理することができます。
動的呼出し法(DYNAMIC)	順呼出しと乱呼出しの両方の処理を行うことができます。

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に該当する者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者のデータを取り出します。



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もつこと(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。

6.5.2 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。“6.2.2 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、以下のこと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、ファイルが創成されたときに指定した主キーまたは副キーの項目と数および位置と大きさを同じにします。
- 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常に一定のところ)に設定します。

例

可変長レコード形式のレコードの定義例

主キー	副キー	
従業員番号 (6桁の数字)	氏名 (10文字の日本語)	所属 (1～16文字の日本語)

```
     :
   RECORD KEY IS  従業員番号
   ALTERNATE RECORD KEY IS  氏名.
     :
FD  従業員ファイル
   RECORD IS VARYING IN SIZE FROM 28 TO 58 CHARACTERS
                           DEPENDING ON レコード長.
     :
01  従業員レコード.
   02  従業員番号  PIC 9(6).
   02  氏名        PIC N(10).
   02  所属.
       03  PIC N OCCURS 1 TO 16 TIMES DEPENDING ON 所属の長さ.
     :
WORKING-STORAGE SECTION.
01  レコード長     PIC 9(3) BINARY.
01  所属の長さ     PIC 9(3) BINARY.
     :
```

6.5.3 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。

入出力文の種類	使い方
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。

主キー　　　　　　副キー

従業員番号 (6桁の数字)	氏名 (10文字の日本語)	所属 (1～16文字の日本語)

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ます。

```
OPEN OUTPUT  ファイル名.
 レコードの編集処理
 主キーの値の設定
WRITE  レコード名  ～.
CLOSE  ファイル名.
```

1. OUTPUT指定のOPEN文でファイルをオープンします。
2. 書き出すレコードの内容を設定します。
3. RECORD KEY句に指定したデータ名に、主キーの値を設定します。順呼出しの場合、主キーの値が昇順になるように、レコードを書き出します。
4. WRITE文でレコードを書き出します。
5. 2.～4.を繰り返し、すべてのレコードを書き出したら、CLOSE文でファイルをクローズします。

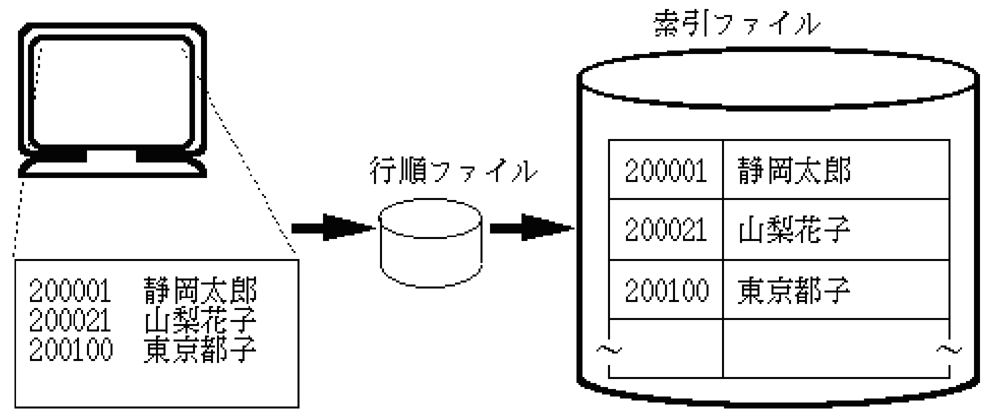
注意

- 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、主キーの値を必ず設定する必要があります。また、順呼出しの場合、主キーの内容が昇順になるように書き出す必要があります。

参考

索引ファイルを作成するときのデータの収集方法として次の機能を使うと便利です。

－ エディタでデータを作成し、行順ファイル機能を使ってそのデータを読み込み、索引ファイルに書き出します。



－ 画面処理機能(表示ファイル機能・スクリーン操作機能)を使って、画面からデータを入力し、索引ファイルに書き出します。

拡張〔順〕

索引ファイルの拡張は、ファイルを拡張モードで開いて、順番にレコードを書き出します。このとき、ファイルの最後のレコードの後ろにレコードが追加されます。ファイルの拡張が可能なアクセス形態は、順呼出し法だけです。

```
OPEN EXTEND  ファイル名.
 レコードの編集処理
WRITE  レコード名  ～.
CLOSE  ファイル名.
```

1. EXTEND指定のOPEN文でファイルをオープンします。
2. 書き出すレコードの内容を設定します。
3. WRITE文でレコードを書き出します。
4. 2.～3.を繰り返し、すべてのレコードを書き出したら、CLOSE文でファイルをクローズします。

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、最初に処理する主キーの値は以下の条件を満たす必要があります。

－ ファイル管理記述項のRECORD KEY句にDUPLICATES指定がある場合

（最初に処理する主キーの値）≧（そのファイルに存在する最大の主キーの値）

－ 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）＞（そのファイルに存在する最大の主キーの値）

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.
 キーの値の設定
```

```
START   ファイル名.
READ    ファイル名 [NEXT] ～.
CLOSE   ファイル名.
```

1. INPUT指定のOPEN文でファイルをオープンします。
2. 参照を開始するレコードの主キーおよび副キーの値を設定します。
3. START文で参照を開始するレコードに位置付けます。
4. READ文で、指定したキーの値の昇順にレコードを読み込みます。動的呼出しの場合は、READ文にNEXT指定を記述します。
5. 4.を繰り返し、必要なレコードをすべて読み終わったら、CLOSE文でファイルをクローズします。

乱呼出し

```
OPEN INPUT  ファイル名.
 キーの値の設定
READ    ファイル名  ～.
CLOSE   ファイル名.
```

1. INPUT指定のOPEN文でファイルをオープンします。
2. 参照するレコードの主キーおよび副キーの値を設定します。
3. READ文で、レコードを読み込みます。
4. 2.～3.を繰り返し、必要なレコードをすべて読み終わったら、CLOSE文でファイルをクローズします。

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、“6.6 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、指定したキー値をもつ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、“6.6 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.
 キーの値の設定
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.
```

1. I-O指定のOPEN文でファイルをオープンします。
2. 更新を開始するレコードの主キーおよび副キーの値を設定します。
3. START文で更新を開始するレコードに位置付けます。
4. READ文で、指定したキーの値の昇順にレコードを読み込みます。動的呼出しの場合は、READ文にNEXT指定を記述します。
5. 読み込んだレコードを更新します。

6. REWRITE文で、更新したレコードを書き出します。

7. 4.～6.を繰り返し、必要なレコードをすべて更新したら、CLOSE文でファイルをクローズします。

乱呼出し

```
OPEN I-O  ファイル名.
 キーの値の設定
 [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.
```

1. I-O指定のOPEN文でファイルをオープンします。

2. 更新を開始するレコードの主キーおよび副キーの値を設定します。

3. 必要であれば、READ文でレコードを読み込みます。

4. レコードを編集または更新します。

5. REWRITE文で、編集または更新したレコードを書き出します。

6. 2.～5.を繰り返し、必要なレコードをすべて更新したら、CLOSE文でファイルをクローズします。

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も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、"6.6 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.
 キーの値の設定
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.
```

1. I-O指定のOPEN文でファイルをオープンします。

2. 削除を開始するレコードの主キーおよび副キーの値を設定します。

3. START文で削除を開始するレコードに位置付けます。

4. READ文で、指定したキーの値の昇順にレコードを読み込みます。動的呼出しの場合は、READ文にNEXT指定を記述します。

5. DELETE文で、読み込んだレコードを削除します。

6. 3.～5.を繰り返し、必要なレコードをすべて削除したら、CLOSE文でファイルをクローズします。

乱呼出し/動的呼出し

```
OPEN I-O   ファイル名.
 キーの値の設定
DELETE ファイル名  ～.
CLOSE  ファイル名.
```

1. I-O指定のOPEN文でファイルをオープンします。

2. 削除するレコードの主キーおよび副キーの値を設定します。

3. DELETE文で、レコードを削除します。

4. 2.～3.を繰り返し、不要なレコードをすべて削除したら、CLOSE文でファイルをクローズします。

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も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、“6.6 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
 キーの値の設定
WRITE  レコード名  ～.
CLOSE  ファイル名.
```

1. I-O指定のOPEN文でファイルをオープンします。

2. 挿入するレコードの内容を設定します。

3. 挿入するレコードの主キーおよび副キーの値を設定します。

4. WRITE文でレコードを書き出します。

5. 2.～4.を繰り返し、すべてのレコードを挿入したら、CLOSE文でファイルをクローズします。

注意

以下の指定したキーの値をもつレコードがすでに存在する場合に、無効キー条件が成立します。無効キー条件については、“6.6 入出力エラー処理”を参照してください。

― ファイル管理記述項のRECORD KEY句にDUPLICATES指定がない場合

― ファイル管理記述項のALTERNATE RECORD KEY句にDUPLICATES指定がない場合

6.6 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)

6.6.1 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。

6.6.2 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も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"をも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。

6.6.3 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。入出力文のあとに、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実行が続けられる場合があるため、その後の動作は保証されません。

通知される入出力状態値については、“付録B 入出力状態一覧”を参照してください。入出力状態の成功の分類には、入出力文の実行は成功していても、その入出力の結果について何らかの情報が通知されるものが含まれます。

例

FILE STATUS句の記述例

```
     SELECT   ファイル
        FILE STATUS IS   入出力状態値
      :
WORKING-STORAGE SECTION.
01   入出力状態値  PIC X(2).
      :
      OPEN  INPUT  ファイル.
      IF   入出力状態値 NOT = "00"
        THEN GO TO   オープン失敗の処理.
```

ファイルのオープンに失敗すると、入出力状態値に"00"以外の値が設定されます。IF文でこの値をチェックしているので、オープンに失敗した場合にはのGO TO文が実行されます。

6.6.4 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

例

誤り処理手続きの記述例

```
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。

6.6.5 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6.4 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6.4 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あと、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 ; export CBR_FILE_USE_MESSAGE
```

- 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、環境変数情報CBR_CBRINFO=ENVを指定して、アプリケーション実行時に設定されていた環境変数の情報を確認する方法もあります。
  環境変数の詳細は、"付録E 環境変数一覧"を参照してください。

6.7 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。

6.7.1 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名とし、プログラム実行時に、入出力処理の対象となるファイルの名前を設定します。

入力コマンド

```
$ OUTDATA=/home/xx/data ; export OUTDATA
$ A
```

［プログラムの内容］

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. A.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT ファイル1
      ASSIGN TO OUTDATA.
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 A.
```

プログラムAを実行すると、ファイル/home/xx/dataが処理されます。

- ファイル識別名を英小文字で記述した場合、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。

入力コマンド

```
$ B
```

［プログラムの内容］

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. B.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT ファイル1
      ASSIGN TO "/home/xx/data".
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 B.
```

/home/xx/data

プログラムBを実行すると、ファイル/home/xx/dataが処理されます。

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、カレントディレクトリ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。

入力コマンド

```
$ C
```

［プログラムの内容］

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. C.
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 "/home/xx/data"
              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 C.
```

/home/xx/data

プログラムCを実行すると、ファイル/home/xx/data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、カレントディレクトリ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句に文字列DISKを記述した場合

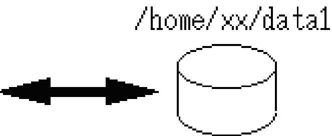
SELECT句に記述したファイル名の先頭にカレントディレクトリを付加したファイル名のファイルに対して入出力処理が行われます。

入力コマンド

```
$ cd /home/xx
$ D
```

［プログラムの内容］

```
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/home/xx/data1が処理されます。

注意

SELECT句には、絶対パス名のファイル名は記述できません。(COBOLの利用者語の規則に従わないため)

6.7.2 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。

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ていきます。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

なお、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になります。

・ ファイル識別名でファイルを割り当て、ファイルがオープン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文でクローズし、OPEN文を実行すると、そのあと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、ファイルに正しく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モードでオープンしているとき、ファイルをクローズす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。［参照］“第25章 ファイルユーティリティ”

6.7.3 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、ほか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。

6.7.3.1 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オープンすると、ほか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オープンされます。

・ 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を指定した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“表6.5 LOCK MODE句が指定されていない場合”、“表6.6 LOCK MODE句にEXCLUSIVEが指定されている場合”および“表6.7 LOCK MODE句にAUTOMATICまたはMANUALが指定されている場合”に示します。

表6.5 LOCK MODE句が指定されていない場合

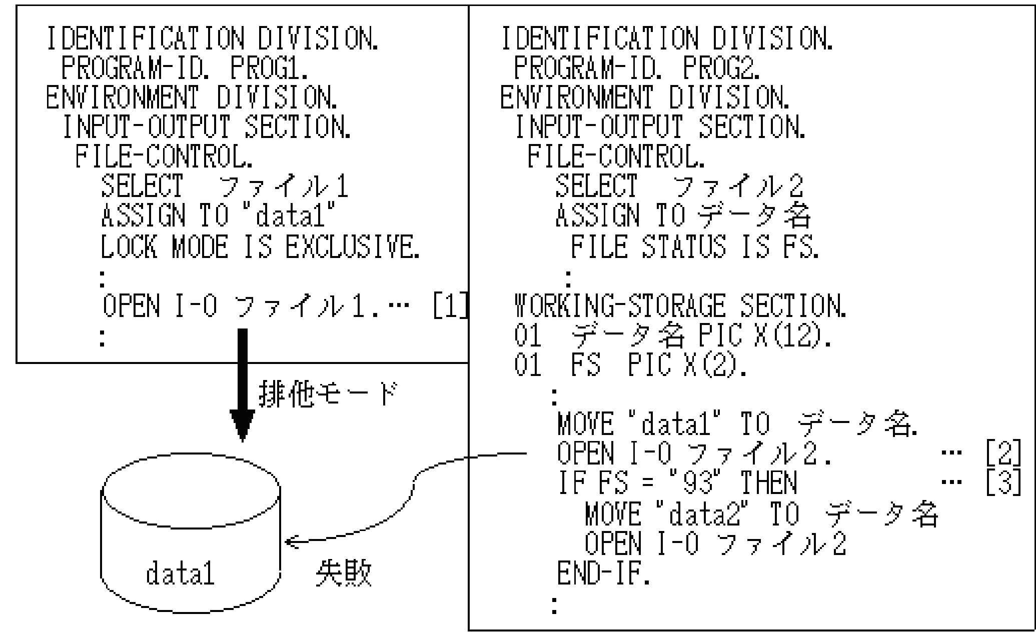
OPEN文のモード	入力INPUT		入出力I-O		拡張EXTEND		出力OUTPUT	
OPEN文のWITH LOCK 指定	なし	あり	なし	あり	なし	あり	なし	あり
ファイルのモード	共用	排他	排他	排他	排他	排他	排他	排他

表6.6 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6.7 LOCK MODE句にAUTOMATICまたはMANUALが指定されている場合

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



[1] 排他モードでファイル(DATA1)を開きます。

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

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

6.7.3.2 レコードを排他状態にする方法

使用中のレコードを排他状態にすると、ほかの利用者はそのレコードを処理することができません(ただし、レコードを排他状態にしない参照処理は行うことができます)。使用中のレコードだけ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オープンします。

次の場合、ファイルは共用モードでオープンされます。

- 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を指定しない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。

共用モードでオープンされたファイルは、ほかの利用者と共用して使用することができます。ただし、すでにほか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。共用モードでオープンさ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オープンし、WITH NO LOCKを指定しない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オープンし、WITH LOCKを指定したREAD文を実行します。

上記組合せによる、レコードの状態を"表6.8 レコードの状態(排他/共用)"に示します。

表6.8 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

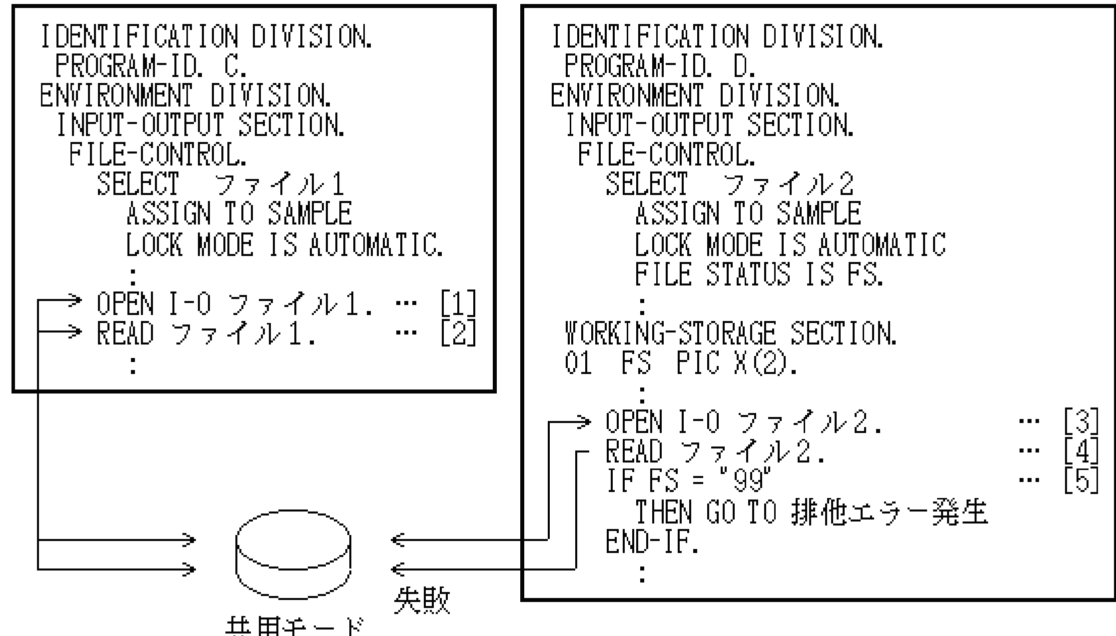
- READ文/REWRITE文/WRITE文/DELETE文/START文を実行します。
- UNLOCK文を実行します。
- CLOSE文を実行します。

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。

6.8 その他のファイル機能

この製品のファイル処理では、性能向上や各種ファイル操作のための以下の機能をサポートしています。ここではこれらの機能について説明します。

- C-ISAM連携
- 大容量ファイル
- RDMファイル
- 性能向上のための機構
- ファイル書込みに関する機構
- COBOLファイルアクセスルーチン
- ファイル追加書き
- ファイルの連結
- ダミーファイル
- 名前付きパイプ
- 外部ファイルハンドラ

COBOLファイルシステム、C-ISAM、大容量ファイルおよびRDMのそれぞれのファイルシステムで利用できる入出力機能の範囲を“表6.9 各ファイルシステムの機能差”に示します。

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。

表6.9 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	大容量ファイル	RDM(注1)	C-ISAM
ファイル	ファイルの最大サイズ(バイト数)	レコード順ファイル	1G	1T	2G(注2)	2G
		レコード順ファイル(ファイルの高速処理)	2G	1T	−	−
		行順ファイル	1G	1T	−	−
		行順ファイル(ファイルの高速処理)	2G	1T	−	−
		相対ファイル	1G	1T	2G	2G
		索引ファイル	1.7G	1T	2G	2G
レコード	レコード形式	固定長レコード形式	○	○	○	○
		可変長レコード形式	○	○	−	○
	レコードの最大長(バイト数)	固定長レコード形式	32,760	32,760	32,760	32,511
		可変長レコード形式	32,760	32,760	−	32,511
	レコードの最小長(バイト数)	レコード順ファイル	1	1	1	−
		行順ファイル	0	0	−	−
		相対ファイル	1	1	1	−
		索引ファイル	キーを構成する項目までの大きさ	キーを構成する項目までの大きさ	キーを構成する項目までの大きさ	1
ファイル管理記述項	SELECT句	OPTIONAL指定	○	○	○(注3)	○
	ASSIGN句	ファイル識別名指定	○	○	○	○
		ファイル識別名定数指定	○	○	○	○

分類	項目		COBOLファイルシステム	大容量ファイル	RDM(注1)	C-ISAM
		データ名指定	○	○	○	○
		DISK指定	○	−	−	−
	FILE STATUS句	ファイル状態	○	○	○(注4)	○(注4)
	LOCK MODE句	AUTOMATIC	○	○	○(注5)	○
		EXCLUSIVE	○	○	○(注5)	○
		MANUAL	○	○	−(注5)	−
	RECORD KEY句	指定できるデータの最大個数	254	254	128	8
		指定できるデータ総長の最大(バイト数)	254	254	255	120
	ALTERNATE RECORD KEY句	指定できるデータの最大個数	254(注6)	254(注6)	128	124
		指定できるデータ総長の最大(バイト数)	254	254	255	120
	レコードキーの項目(注7)	英数字	○	○	○	○
		日本語	○	○	○	○
		符号なし外部10進数	○	○	○	−
		符号付き外部10進数	−	−	○	−
		符号なし内部10進数	○	○	○	○
		符号付き内部10進数	−	−	○	−
		符号なし2進数(BINARY/COMP-5)	○	○	○	−
		符号付き2進数(BINARY/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	○	○	−	○
	DELETE文	キー値が重複しているレコードを削除	○	○	○	○
	UNLOCK文		○	○	−	○
機能	スレッド	プロセス(シングルスレッド)	○	○	○	○
		マルチスレッド	○	○	○(注8)	−
	トランザクション管理	トランザクション開始表示	−	−	○(注9)	○
		COMMIT指示	−	−	○(注9)	○
		ROLLBACK指示	−	−	○(注9)	○

○:サポート　−:非サポート

注1: RDMについては、“PowerRW+”の情報を基に挙げています。

注2: ファイル管理情報で使用するため、実際の制限値は小さくなります。

注3: RDMでは、OPEN INPUTで不定ファイルをオープンできます。ただし、OPEN I-OまたはOPEN EXTENDでは不定ファイルをオープンできません。

注4: RDMまたはC-ISAMでは、FILE STATUS=02を返却しません。また、排他エラー(ファイル排他/レコード排他とも)を示すFILE STATUS値は"92"です。

注5: RDMでは、プログラムの記述によりファイル排他およびファイル共用(レコード排他)を制御することはできません。RDMの拡張機能を用いて、ファイル排他およびファイル共用(レコード排他)を行います。

注6: COBOLファイルシステムでは、RECORD KEY句とALTERNATE RECORD KEY句に指定できるデータ個数の総和が最大255です。したがって、RECORD KEY句に指定するデータの個数が増加するとこの値は減少します。

注7: 非サポートのデータ項目をレコードキーに定義した場合、その実行結果は保証されません。

注8: マルチスレッドモデルでRDMを使用する場合、スレッド間で同じファイル結合子を操作してファイルを共用することはできません。

注9: C-ISAMまたはRDMに用意されている機能(関数)を、CALL文で呼び出す必要があります。RDMでトランザクション操作を行う方法については、“RDB/7000説明書”または“PowerRW+解説書”を参照してください。C-ISAMでトランザクション操作を行う方法については、“C-ISAM手引書”を参照してください。

6.8.1 C-ISAMの使い方

この製品では、順ファイル、相対ファイルおよび索引ファイルとしてC-ISAMを使用することができます。

C-ISAMを使用する場合でも、順ファイル、相対ファイルおよび索引ファイルの基本的な使い方は同じです。ここでは、C-ISAMを使用する場合の注意点だけを説明します。

6.8.1.1 C-ISAMの指定

プログラム中のファイル参照子と実際に入出力対象となるファイルを関連付ける場合に、C-ISAMを指定します。

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

環境変数の設定時に、割り当てるファイルのファイル名に続き、“,CIM”を指定します。

```
$ ファイル識別名=ファイル名,CIM ; export ファイル識別名
```

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、“,CIM”を指定します。

```
 ASSIGN TO "ファイル名,CIM".
```

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、“,CIM”を指定します。

```
 MOVE  "ファイル名,CIM" TO データ名.
```

6.8.1.2 プログラムの翻訳・リンク

C-ISAMを使用するプログラムの翻訳・リンクを行うための注意点を説明します。翻訳・リンクの詳細な説明は、“第3章 プログラムの翻訳とリンク”を参照してください。

C-ISAMを使用するプログラムを翻訳・リンクする場合、libcobcim.soおよびlibisam.aをプログラムにリンクする必要があります。

cobolコマンドで実行可能プログラムの翻訳・リンクを行う場合

実行可能プログラムの翻訳・リンクを行う際、翻訳オプション“-pi”を指定します。この翻訳オプションにより、実行可能プログラムにlibcobcim.soおよびlibisam.aがリンクされます。

```
$ cobol -M -pi -o PRG01 PRG01.cob
```

ldコマンドで実行可能プログラムのリンクを行う場合

実行可能プログラムのリンクを行う際、libcobcim.soおよびlibisam.aをリンクするよう指定します。なお、libcobcim.soとlibisam.aのリンク順序は、必ずlibcobcim.soが先にリンクされるように指定してください。

```
$ ld -o PRG01 /opt/FJSVcbl/lib/crti.o /opt/FJSVcbl/lib/crt1.o ¥
    /opt/FJSVcbl/lib/crtn.o /opt/FJSVcbl/lib/cblstr.o PRG01.o ¥
    -lcobcim -lisam -lcobol -lc -ldl
```

COBOLコマンドで共用オブジェクトの翻訳・リンクを行う場合

共用オブジェクトの翻訳・リンクを行う際、翻訳オプション"-pi"を指定します。この翻訳オプションにより、実行可能プログラムにlibcobcim.soおよびlibisam.aがリンクされます。

```
$ cobol -dy -G -pi -olibPRG01.so PRG01.cob
```

ldコマンドで共用オブジェクトファイルのリンクを行う場合

共用オブジェクトファイルのリンクを行う際、libcobcim.soおよびlibisam.aをリンクするよう指定します。なお、libcobcim.soとlibisam.aのリンク順序は、必ずlibcobcim.soが先にリンクされるように指定してください。

```
$ ld -dy -G -o libPRG01.so PRG01.o -lcobcim -lisam -lcobol -lc -ldl
```

6.8.1.3 C-ISAMでの注意点

- ファイル参照子にDISKを指定することはできません。
- 相対ファイルに対するWRITE文は、順呼出し以外では実行できません。
- 索引ファイルに対するREVERSED指定付きのSTART文は実行できません。
- NFS上のファイルを入出力対象とすることはできません。
- 実ファイル名の長さは、拡張子を除いて10文字以内にしてください。
- C-ISAMで扱えるファイルの定量制限については、"C-ISAM手引書"を参照してください。ただし、ALTERNATE RECORD KEY句の個数は、C-ISAMの定量制限にかかわらず、COBOLプログラムの翻訳時に124個に制限されます。
- libisam.aはC-ISAMから提供されているアーカイブライブラリです。プログラムをリンクする際には、libisam.aが正しく検索されるように環境変数LD_LIBRARY_PATHを設定してください。
- ファイルの排他規則は、同一プロセス上では有効になりません。
- 固定長レコード形式のとき、ファイル創成時の最大レコード長と一致している必要があります。
- 可変長レコード形式のとき、ファイル創成時の最大レコード以内である必要があります。
- 相対ファイルに対する順呼出しのREWRITE文では、書き換えるレコードの長さは一致している必要があります。
- 強制終了などによりC-ISAMファイルのクローズ処理が行われなかった場合、その後の動作は保証されません。C-ISAMファイルに対してファイルの復元、またはリカバリ処理を実施してください。

6.8.2 大容量ファイル処理

COBOLファイルシステムの最大ファイルサイズを超えるファイルを扱う場合、大容量ファイルを指定します。最大ファイルサイズについては、"表6.9 各ファイルシステムの機能差"を参照してください。

対象となるのはレコード順ファイル、行順ファイル、相対ファイルおよび索引ファイルです。またレコード順ファイル、行順ファイルについては、大容量ファイルアクセスでの高速処理機能も用意しています。

なおCOBOLで作成した大容量ファイルを整列・併合機能で使用するときにはPowerSORTが必要です。

識別名	意味
LFS	大容量ファイルのアクセス指定
LBSAM	大容量ファイルアクセス時の高速化機能

参考

LFS(またはLBSAM)指定により作成、更新などのファイル処理したファイルのファイルフォーマットに変更はありません。そのためLFS(またはLBSAM)指定により作成した大容量ファイルとLFS(またはLBSAM)指定なしで作成したファイルは交互に移行ができます。ただしファイルサイズの大きい大容量ファイルをLFS(またはLBSAM)指定なしではアクセスできないので注意が必要です。

6.8.2.1 大容量ファイルアクセスの指定

以下に大容量ファイルのアクセス方法を示します。

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

環境変数の設定時に、割り当てるファイルのファイル名に続き“,LFS”または“,LBSAM”を指定します。環境変数の設定方法については、“6.7.1 ファイルの割当て”を参照してください。

```
$ ファイル識別名=ファイル名,LFS ; export ファイル識別名
```

```
$ ファイル識別名=ファイル名,LBSAM ; export ファイル識別名
```

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“,LFS”または“,LBSAM”を指定します。

```
ASSIGN TO "ファイル名,LFS".
```

```
ASSIGN TO "ファイル名,LBSAM".
```

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“,LFS”または“,LBSAM”を指定します。

```
MOVE  "ファイル名,LFS" TO データ名.
```

```
MOVE  "ファイル名,LBSAM" TO データ名.
```

注意

- ファイルまたはレコードの共用･排他は以下の組合せで有効となります。

		“,LFS”指定または“,LBSAM”指定	
		あり	なし
“,LFS”指定 または“,LBSAM”指定	あり	○	×
	なし	×	○

○: 共用･排他を動作保証している　×: 共用･排他を動作保証していない

- LBSAM指定は、レコード順ファイルまたは行順ファイルで有効です。またBSAMを指定した場合と同じ注意事項があるため、“6.8.4.2 ファイルの高速処理”の注意事項を参照してください。

6.8.2.2 一括指定

大容量ファイル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

環境変数情報CBR_FILE_LFS_ACCESSに文字列”YES”を指定します。

```
$ CBR_FILE_LFS_ACCESS=YES ; export CBR_FILE_LFS_ACCESS
```

“YES”が指定された場合、すべてのCOBOLファイルを大容量ファイルとしてアクセスします。

また、データの出力先をファイルにしたDISPLAY文に対して、出力ファイルの最大サイズである1Gバイトの制限を解除します。[参照]“10.1.7.4 DISPLAY文のファイル出力拡張機能”

注意

- 本環境変数を指定した場合、ファイルまたはレコードの共用・排他制御は",LFS"を指定した場合と同様です。
  - 他のCOBOLプログラムとファイルを共有している場合、すべての処理で本環境変数、または、大容量ファイルの指定(",LFS"または",LBSAM")を有効にしてください。
  - 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場合、大容量ファイル指定を有効にしてください。大容量ファイルの指定方法は、製品の使用手引書を参照してください。
- ファイル機能を個別に指定したファイルに対し、本環境変数を指定した場合、以下の通り動作します。

ファイル機能			説明
COBOLファイル	大容量ファイル	LFS	大容量ファイルとして動作します。
	ファイルの高速処理	BSAM LBSAM	大容量ファイルアクセス時の高速処理として動作します。
	ファイルの追加書き	MOD	大容量ファイル、かつ、ファイルの追加書きを有効として動作します。
	ファイルの連結	CONCAT	大容量ファイル、かつ、ファイルの連結を有効として動作します。
	ダミーファイル	DUMMY	ダミーファイルとして動作します。
DISPLAY文のファイル出力	ファイルの追加書き	MOD	ファイルの最大サイズ制限解除、かつ、追加書きを有効として動作します。
	ファイルの最大サイズ制限解除	NOLIMIT	ファイルの最大サイズ制限解除を有効として動作します。
	ダミーファイル	DUMMY	ダミーファイルとして動作します。

- ファイル高速処理の一括指定(環境変数情報CBR_FILE_SEQUENTIAL_ACCESS)が有効なファイルに対しては、大容量ファイルアクセス時の高速処理(LBSAM)が指定されたものとみなします。[参照]“一括指定”

6.8.3 RDMファイル

RDMファイルは、順ファイル、相対ファイルおよび索引ファイルとして利用できます。RDMで利用できる機能については、“表6.9 各ファイルシステムの機能差”を参照してください。ここでは、RDMファイルの使用方法について説明します。

参考

RDMファイルのより詳細な使用方法については、“RDB/7000説明書”または“PowerRW+解説書”を参照してください。

6.8.3.1 ファイル環境の指定

COBOLファイルシステムを使用するか、RDMファイルシステムを使用するかは、ファイル管理記述項のASSIGN句の記述によって決まります。RDMファイルシステムを使用する場合は、以下のどれかの方法で指定してください。

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 "ファイル名 ,RDM"
```

ASSIGN句にデータ名を記述した場合

データ名の設定するファイル名は、ファイル識別名定数の場合と同じ形式で指定します。

```
MOVE "ファイル名 ,RDM" TO データ名.
```

ASSIGN句にDISKを記述した場合

RDMファイルシステムを使用することはできません。COBOLファイルシステムが使用されます。

ASSIGN句にファイル識別名を記述した場合

ファイル識別名を環境変数名で指定する場合には、ファイル識別名定数の場合と同じ形式でファイル名を指定します。

```
$ ファイル識別名=ファイル名 ,RDM ; export ファイル識別名
```

RDMファイルの拡張機能を使用する場合

上記の指定のほかに以下の環境変数(CRDB_XLIB_OVR)を指定します。

```
$ CRDB_XLIB_OVR=ファイル名,RDM,拡張機能を表す文字列 [:...] ; export CRDB_XLIB_OVR
```

参照

"拡張機能を表す文字列"については"RDB/7000説明書"または"PowerRW+ COBOL開発マニュアル"を参照してください。

6.8.3.2 注意事項

- COBOLプログラムで初めて使用する相対ファイルは、まず"相対ファイルの創成"を行ってください。相対ファイルの創成は、OUTPUT指定のOPEN文を実行すると行われます。創成していないファイルをI-OまたはEXTEND指定のOPEN文で開いて、WRITE文を実行した場合、"レコードが存在しない"旨の以下のどちらかのエラーになります。
  - 入出力状態FS=23
  - 実行時エラーJMP0324I-I/U

  また、ディスク容量不足が発生した場合、CLOSE文の実行時に以下のエラーメッセージが出力されます。

  ```
  JMP0310I-I/U 'アクセス名またはファイル名'ファイルで'CLOSE'エラーが発生しました.'ERFLD=1C'
  ```

- RDMファイルの排他制御は、COBOLファイルシステムと比較して動作が異なる場合があります。詳細については、"RDB/7000説明書"または"PowerRW+解説書"を参照してください。

6.8.4 性能向上のための機構

ファイルアクセスの性能向上のためにバッファリングおよび高速アクセス機構を用意しています。

なお、これらの機能は、性能向上と引換えに各種制限事項があります。ご注意ください。

6.8.4.1 ファイル処理の性能改善

入力モードでオープンしたファイル(順/行順/相対)の入力処理の性能を向上させることができます。

使い方

実行時に、環境変数CBR_INPUT_BUFFERINGに"yes"を設定します。

```
$ CBR_INPUT_BUFFERING=yes ; export CBR_INPUT_BUFFERING
```

注意

本機能は、COBOLランタイムシステムがファイルをアクセスする回数を減らすために、ファイルからレコードを読み込むときの処理を、レコード単位ではなく、バッファ単位で行います。1つのバッファは複数レコードを含みます。したがって、ほかのファイル識別子によりファイルの内容が更新された場合など、最新レコードの内容を保証できないことがあります。

また、本指定は"6.8.4.2 ファイルの高速処理"では無効となります。

6.8.4.2 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

指定方法

レコード順ファイルも行順ファイルも、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

環境変数の設定時に、割り当てるファイルのファイル名に続き、",BSAM"または",LBSAM"を指定します。環境変数の設定方法については、"6.7.1 ファイルの割当て"を参照してください。

```
$ ファイル識別名=ファイル名,BSAM ; export ファイル識別名
```

```
$ ファイル識別名=ファイル名,LBSAM ; export ファイル識別名
```

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"または",LBSAM"を指定します。

```
ASSIGN TO "ファイル名,BSAM"
```

```
ASSIGN TO "ファイル名,LBSAM"
```

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、",BSAM"または",LBSAM"を指定します。

```
MOVE  "ファイル名,BSAM" TO データ名
```

```
MOVE  "ファイル名,LBSAM" TO データ名
```

- レコードの更新(REWRITE文)はできません。レコード順ファイルで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。
- 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指定でオープンされている必要があります。INPUT指定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。

  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、ファイルの高速処理は使用できません。
- ファイルの高速処理を指定した場合、行順ファイルの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。
  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、"6.3.3 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"または"LBSAM"を指定します。

```
$ CBR_FILE_SEQUENTIAL_ACCESS={BSAM | LBSAM} ; export CBR_FILE_SEQUENTIAL_ACCESS
```

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
大容量ファイル	LFS
ファイルの高速処理	BSAM LBSAM
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	RDM
	EXFH
	CIM
	(名前付きパイプ)

注:ファイルの高速処理を同時に有効にしたい場合は、ファイル単位に指定してください。ファイル単位に指定する方法は、"6.8.12 注意事項"を参照してください。

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。

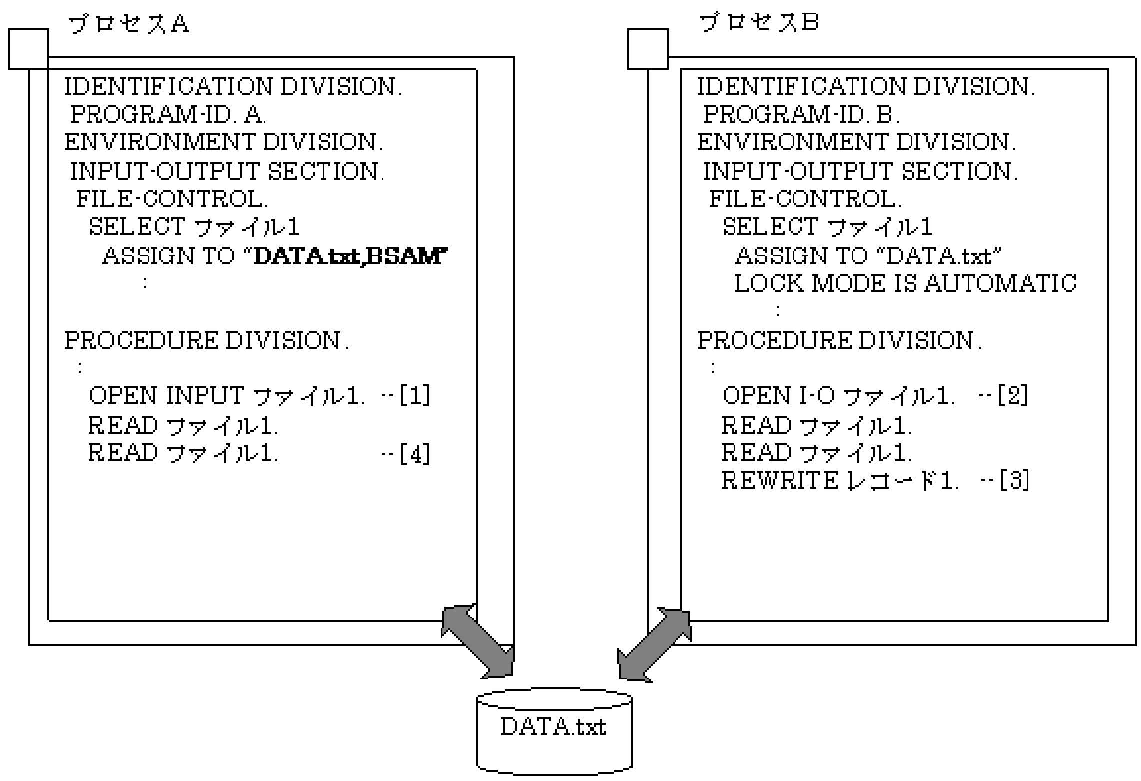
[参照]"ファイル共用時の注意事項"

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。



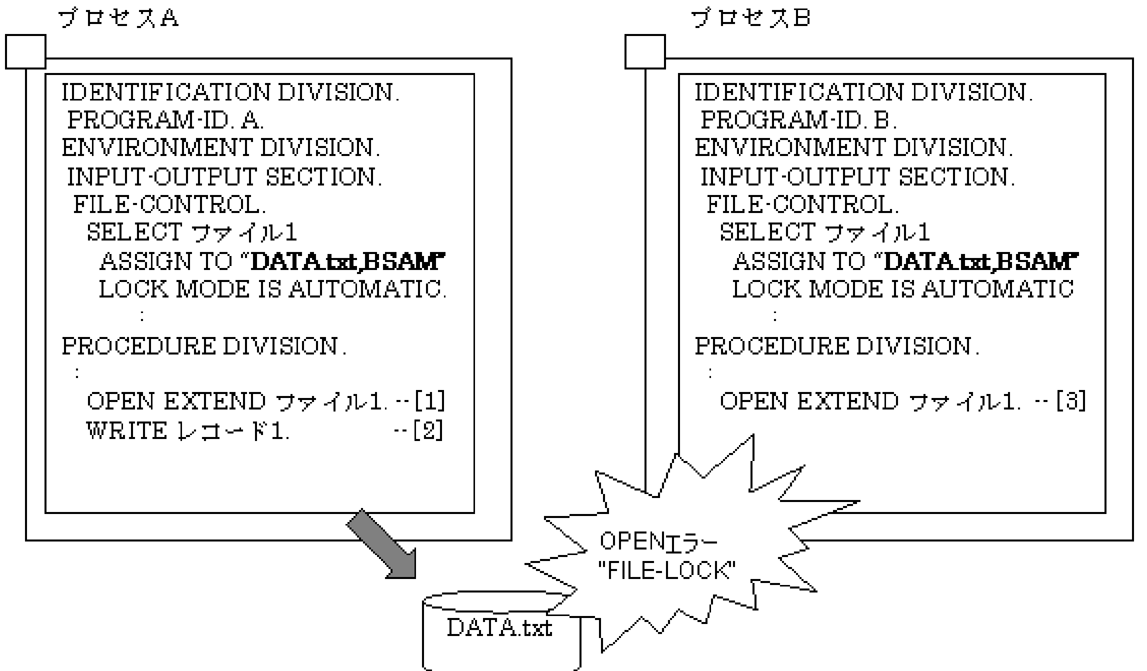
[1] プロセスA：共用モード、INPUT指定でファイルを開きます。（ファイルの高速処理）

[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。

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. A.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT ファイル1
      ASSIGN TO "DATA.txt,BSAM"
          :
PROCEDURE DIVISION.
 :
    OPEN INPUT ファイル1.  -- [1]
    READ ファイル1.        -- [3]
```

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. B.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT ファイル1
      ASSIGN TO "DATA.txt,BSAM"
          :
PROCEDURE DIVISION.
 :
    OPEN INPUT ファイル1.  -- [2]
    READ ファイル1.        -- [4]
```

DATA.txt

[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。

6.8.5 ファイル書込みに関する機構

ファイル書込みに関する機能を用意しています。

6.8.5.1 クローズ時の書込み内容の即時反映

CLOSE文実行時に書込み内容を確実に反映させることができます。

使い方

実行時に環境変数CBR_CLOSE_SYNCに"yes"を設定します。

```
$ CBR_CLOSE_SYNC=yes ; export CBR_CLOSE_SYNC
```

注意

本機能は、CLOSE文実行時にOSが管理しているバッファの内容をディスクに書き込む命令を発行します。そのため、この機能を使用した場合には、OSのバッファの状況に応じて性能が劣化します。

6.8.5.2 行順ファイルの後置空白に関する指定

行順ファイルでWRITE文実行時の後置空白の扱いを指定できます。

使い方

実行時に環境変数CBR_TRAILING_BLANK_RECORDに“REMOVE”を設定した場合、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きます。“VALID”を設定した場合、レコード内の後置空白を取り除きません。本環境変数の設定を省略した場合、“VALID”が指定されたものとみなします。

```
$ CBR_TRAILING_BLANK_RECORD={ REMOVE | VALID } ; export CBR_TRAILING_BLANK_RECORD
```

〔後置空白を削除する機能を無効にした場合（省略時）〕

ＡＢ□ＣＤＥ□ → ＡＢ□ＣＤＥ□

□：空白　　レコード書き込み時

〔後置空白を削除する機能を有効にした場合〕

ＡＢ□ＣＤＥ□ → ＡＢ□ＣＤＥ

□：空白　　レコード書き込み時

レコードの後ろに設定されている空白を削除して、ファイルに書き込みます。

注意

削除される空白は、コード系に依存します。

- 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
- コード系がUnicodeの場合は、字類が英数字の場合は半角空白が削除され、字類が日本語の場合は全角空白が削除されます

6.8.6 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、製品に格納されているREADMEまたは“COBOL ファイルアクセスルーチン使用手引書”を参照してください。

6.8.7 ファイル追加書き

OPEN OUTPUT文の実行で、既存ファイルにレコードを追加することができます。

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 ; export ファイル識別名
```

注意

COBOLファイルのレコード順ファイルだけに有効となります。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。

6.8.8 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。

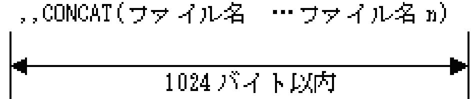
使い方

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n)"; export ファイル識別名
```

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。

6.8.9 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、“,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 ; export ファイル識別名
```

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。エラーにはなりません。ご注意ください。
- 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。

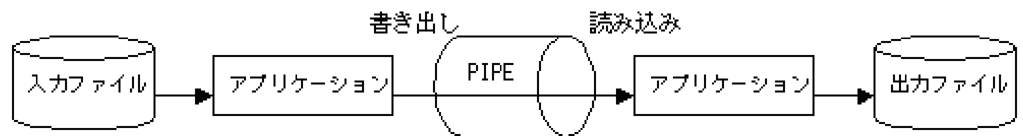
表6.10 ダミーファイルの機能範囲

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。

6.8.10 名前付きパイプ

順ファイルとしてシステムの名前付きパイプを利用することができます。

COBOLアプリケーション間でデータを受け渡す必要がある場合、中間ファイルの代わりに名前付きパイプを利用することができます。名前付きパイプを利用すると、COBOLアプリケーションを並列に動作させてデータを受け渡すことができます。



ここでは、名前付きパイプの使い方と機能範囲について説明します。

使い方

COBOLアプリケーションを実行する前に、名前付きパイプを作成しておきます。作成した名前付きパイプを通常のファイルを割り当てる場合と同じように指定します。

注意

名前付きパイプを作成するには、システムのmkfifoコマンドを使用します。

機能範囲

名前付きパイプを使用する場合の機能範囲を以下に示します。

表6.11 名前付きパイプの機能範囲

<table>
<tr><td>ファイル編成</td><td colspan="2">レコード順ファイル
行順ファイル</td></tr>
<tr><td>レコード形式</td><td colspan="2">固定長形式
可変長形式</td></tr>
<tr><td rowspan="2">ファイル管理記述項</td><td>SELECT句</td><td>ファイル識別名(注1)</td></tr>
<tr><td>LOCK MODE句</td><td>指定しても意味を持ちません。</td></tr>
<tr><td rowspan="5">入出力文</td><td>OPEN文</td><td>・ OUTPUT/INPUTモード(注2)
・ WITH LOCK指定は指定しても意味を持ちません。</td></tr>
<tr><td>READ文</td><td>WITH LOCK/WITH NO LOCK指定は意味を持ちません。</td></tr>
<tr><td colspan="2">WRITE文</td></tr>
<tr><td>REWRITE文</td><td>指定不可</td></tr>
<tr><td>UNLOCK文</td><td>指定しても意味を持ちません。</td></tr>
</table>

注1: ファイル識別名定数、データ名、DISK指定は指定できません。

注2: オープンモードI-Oは指定できません。

6.8.11 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 ; export ファイル識別名
```

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、“,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、共用オブジェクトファイル名と入口名を指定する必要があります。

2つとも同時に指定されている場合は、ファイル単位の指定が優先されます。

実行環境に有効とする方法

実行時に以下の環境変数を設定します。

```
$ CBR_EXFH_API=入口名 ; export CBR_EXFH_API
```

- 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=共用オブジェクトファイル名 ; export CBR_EXFH_LOAD
```

- 共用オブジェクトファイル名:結合するファイルシステムの共用オブジェクトファイル名を指定します。

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。

注意

制御文CBR_EXFH_LOADが指定されていない場合、共用オブジェクトファイル名には、制御文CBR_EXFH_APIに指定された入口名を“lib入口名.so”とみなして処理します。

例

結合するファイルシステムの入口名を“flsys”、共用オブジェクトファイル名を“libfilesys.so”とする場合

```
$ CBR_EXFH_API=flsys ; export CBR_EXFH_API
$ CBR_EXFH_LOAD=libfilesys.so ; export CBR_EXFH_LOAD
```

結合するファイルシステムの入口名を“file”、共用オブジェクトファイル名を“libfile.so”とする場合

```
$ CBR_EXFH_API=file ; export CBR_EXFH_API
```

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)" ; export ファイル識別名
```

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=共用オブジェクトファイル名
```

- 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)
- 共用オブジェクトファイル名:結合するファイルシステムの共用オブジェクトファイル名を指定します。

注意

制御文CBR_EXFH_LOADが指定されていない場合、共用オブジェクトファイル名には、制御文CBR_EXFH_APIに指定された入口名を“lib入口名.so”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”、共用オブジェクトファイル名を“libafilesys.so”とする場合

```
$ ファイル識別名="Afile,EXFH,INF(aflsys.inf)"; export ファイル識別名
```

ファイル"aflsys.inf"の内容

```
[EXFH]
CBR_EXFH_API=aflsys
CBR_EXFH_LOAD=libafilesys.so
```

注意事項

- 使用できる外部ファイルハンドラは共用オブジェクトファイル(lib*.so)のみです。Micro Focus COBOLと異なり、オブジェクトファイルを使用することはできません。
- 外部ファイルハンドラは、Unicode環境でコンパイルされたCOBOLアプリケーションでは使用できません。
- マルチスレッド用オプションを指定してコンパイルされたCOBOLアプリケーションから外部ファイルハンドラを使う場合、結合するファイルシステムもマルチスレッドに対応していなければなりません。
- FILE STATUS句を使う場合、外部ファイルハンドラから返される入出力状態値が返却されます。このため、"付録B 入出力状態一覧"の値とは異なる場合があります。
- 索引ファイルではFIRSTの指定されたSTART文はサポートされていません。

6.8.12 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。

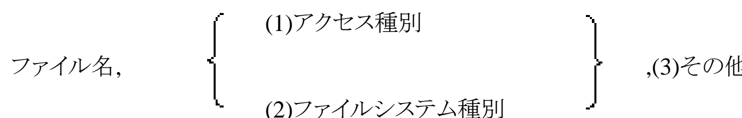
ただし、ファイル連結機能の指定の途中で文字列が1024バイトを超えた場合は、OPEN文実行時にエラーになります。

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
	LBSAM	大容量ファイルの高速処理
	LFS	大容量ファイル
	DUMMY	ダミーファイル
(2)ファイルシステム種別	RDM	RDMファイル
	CIM	C-ISAM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。

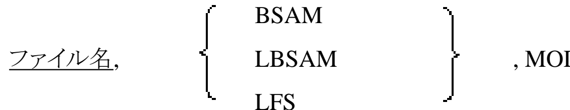


同時に指定可能な組合せとその動作は、以下の通りです。

- (1)アクセス種別 BSAM/LBSAM/LFS 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

BSAM/LBSAM/LFS とその他を同時に指定する場合(1)

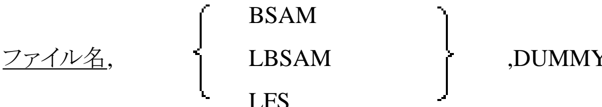


BSAM/LBSAM/LFS とその他を同時に指定する場合(2)

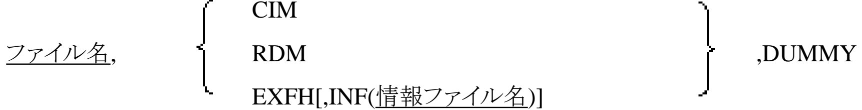
- (1)アクセス種別のダミーファイル(DUMMY)は、すべての機能と同時に指定することができます。この場合、ダミーファイル(DUMMY)だけが有効になり、他の機能は無効になります。

例

BSAM/LBSAM/LFS とダミーファイル(DUMMY)を同時に指定する場合



ファイルシステム種別とダミーファイル(**DUMMY**)を同時に指定する場合



その他とダミーファイル(**DUMMY**)を同時に指定する場合(**1**)

ファイル名,,MOD,DUMMY

その他とダミーファイル(**DUMMY**)を同時に指定する場合(**2**)

,,CONCAT(ファイル名1 ファイル名2 …),DUMMY

その他とダミーファイル(**DUMMY**)を同時に指定する場合(**3**)

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 …)

BSAM/LBSAM/LFS およびその他と、ダミーファイル**(DUMMY)**を同時に指定する場合**(1)**

ファイル名, { BSAM | LBSAM | LFS } ,MOD,DUMMY

BSAM/LBSAM/LFS およびその他と、ダミーファイル**(DUMMY)**を同時に指定する場合**(1)**

- (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

第7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。

7.1 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、これらの印刷方法の概要、印字文字、フォームオーバレイパターン、FCB、印刷情報ファイルおよび帳票定義体について説明します。なお、使用できる印刷機能は印刷装置によって異なります。

- 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する必要があります。データ中にバイナリの不正なコードが含まれる場合、正しく印字されないことがあります。
- 印刷ファイルで作成したファイル(スプール)は、印刷以外の目的で使用しないでください。
  例えば、印刷ファイルで作成したファイル(スプール)を入力し、加工後に出力するというように、印刷以外の目的で使用された場合、ファイル(スプール)は保証しません。

7.1.1 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する場合に使用します。さらに、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したりする場合にも使用します。

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、FORMで定義した帳票定義体を使った帳票形式のデータを印刷します。

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7.1 印刷方法の特徴・利点・用途”に、必要な関連製品を“表7.2 関連製品”に示します。

表7.1 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4)	○	○(注3)
	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	×	◎	◎	○
	出力するデータストリームを指定できる	○(注1)	○(注1)	◎(注2)	◎(注2)
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
×:使用不可能である

注1: 指定できるデータストリームについては“7.1.5 印刷情報ファイル”を参照してください。

注2: 指定できるデータストリームおよびサポートプリンタについてはMeFtのオンラインマニュアルを参照してください。

注3: 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4: PostScriptレベル1のデータストリームを出力する場合は、KOL6形式のオーバレイパターンは出力できません。

表7.2 関連製品

使用するファイルの種類		[1]	[2]	[3]	[4]
関連製品	**FORM**	−	○(注1)	○(注2)	○(注2)
	FORMオーバレイオプション	−	○	○	○
	PowerFORM	−	○(注1)	○(注2)	○(注2)
	MeFt	○(注3)	○(注3)	○	○
	MeFt/NET	−	−	−	○
	MeFt/Web	−	−	−	○
	PrintWalker/BPC (注4)	○	○	○	○
	Systemwalker/ListWORKS または **Interstage List Works (注5)**	○	○	○	○
	Systemwalker/e-DocGenerator または **Systemwalker/ListCREATOR EE** または **Interstage List Creator Enterprise Edition (注6)**	−	−	○	○
	Interstage List Manager または **Interstage List Works (帳票印刷配信機能) (注7)**	−	−	○	○

○:使用可能
−:使用不可能

- 注1: オーバレイを作成するために使用します。
- 注2: オーバレイを作成する場合または帳票定義体を作成する場合に使用します。PowerFORMで作成した帳票定義体を使用する場合の注意事項についてはMeFtのオンラインマニュアルを参照してください。
- 注3: 以下の場合に必要です。
  - PostScriptでフォームオーバレイパターンとの合成印刷を行う場合
  - PostScriptレベル2のデータストリームを出力する場合
  - ESC/Pのデータストリームを出力する場合
  - ESC/Pageのデータストリームを出力する場合
  - LIPS IIIのデータストリームを出力する場合
- 注4: VSPシリーズのプリンタ装置に出力する場合に必要です。
- 注5: 帳票を電子化したり、電子化された帳票に対するさまざまな操作・管理を行ったりする場合に必要です。この章では、特に断りがない限り、Systemwalker/ListWORKS および Interstage List Works を ListWorks と記述します。

- 注6: 帳票をPDFファイルとして出力する場合に必要となります。この章では、特に断りがない限り、Systemwalker/e-DocGenerator、Systemwalker/ListCREATOR EE および Interstage List Creator Enterprise Edition を e-DocGenerator と記述します。

- 注7: 帳票配信を行う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 Manager およびInterstage List Works の帳票印刷配信機能 を List Manager と記述します。

参照

電子化された帳票の詳細については以下を参照してください。

- 7.6 電子帳票出力機能を使う方法

- ListWorksのマニュアル

以下に各印刷方法の概要を説明します。

[1] FORMAT句なし印刷ファイル(行単位のデータを印刷する)

印刷ファイルとは、印刷ファイルという特別なファイルがあるのではなく、印字する目的で定義したファイルのことです。FORMAT句なし印刷ファイルは、レコード順ファイルと同様に定義し、WRITE文を使って行単位のデータを印刷装置やファイルに出力します。このとき、論理ページの大きさを指定したり、行送りや改ページ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は、"7.2 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)

印刷ファイルでは、WRITE文を使って制御レコードを出力し、使用するフォームオーバレイパターンやFCBなどの印刷情報を指示することができます。フォームオーバレイパターンを指定した制御レコードを出力すると、1ページ分の出力データが、フォームオーバレイパターンと合成されます。フォームオーバレイパターンについては、"7.1.7 フォームオーバレイパターン"を、FCBについては、"7.1.6 FCB"を参照してください。

制御レコードを使った印刷ファイルの使い方は、"7.3 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、"7.1.8 帳票定義体"を参照してください。

FORMAT句付き印刷ファイルの使い方は、"7.4 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。また、表示ファイルによる帳票印刷ではMeFtが必要となります。

表示ファイルの出力先には、印刷装置やディスプレイ装置を指定することができます。本章では印刷装置に出力する方法について説明します。なお、ディスプレイ装置への出力方法は、"8.2 表示ファイル機能(画面入出力)"を参照してください。帳票印刷を行う表示ファイルの使い方については、"7.5 表示ファイル(帳票印刷)の使い方"を参照してください。

7.1.2 印刷装置

FORMAT句なし印刷ファイルでは、UVPI(VSPプリンタ向けデータストリーム)、PostScript(レベル1およびレベル2)のデータストリームを出力できます。

標準データストリームはUVPIです。

上記のデータストリーム以外にも、ESC/P、ESC/Page、およびLIPSIIIなどの他社プリンタ向けのデータストリームを出力することが可能です。

どのプリンタ向けのデータストリームを出力するかは、印刷情報ファイルのprinter制御文で指定します。標準データストリームで印刷する場合、印刷情報ファイルへの指定は必要ありません。指定方法の詳細やサポート対象となるプリンタの種類については、"7.1.5 印刷情報ファイル"を参照してください。

以下の場合はMeFtが必要となります。

- PostScriptでフォームオーバレイパターンとの合成印刷を行う場合
- PostScriptレベル2のデータストリームを出力する場合
- ESC/Pのデータストリームを出力する場合
- ESC/Pageのデータストリームを出力する場合
- LIPS IIIのデータストリームを出力する場合

また、UVPIデータストリームのデータをPrintPartner VSPシリーズのプリンタ装置に印刷する場合には、PrintWalker/BPCをインストールしておく必要があります。

FORMAT句付き印刷ファイルおよび表示ファイルでは、出力するデータストリームの種別をプリンタ情報ファイルに指定します。出力できるデータストリームおよびサポート対象となるプリンタの種類については、MeFtのオンラインマニュアルを参照してください。

注意

プリンタ装置がサポートするデータストリームについては、プリンタ付属のマニュアルを参照してください。

7.1.3 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、日本語印刷を行う場合には、日本語項目を使用し、データ記述項でCHARACTER TYPE句を指定する必要があります。日本語項目に対してCHARACTER TYPE句が省略された場合、または英数字項目を使って日本語印刷を行った場合、その印字結果は保証されません。

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	○			○ (注2)		○ (注3)
CHARACTER TYPE 呼び名	○	○		○	○	○ (注3)
CHARACTER TYPE 印字モード名	○	○	○ (注1)	○	○	○

注1: PRINTING MODE句のFONT指定に FONT-nnnを指定しなければなりません。

注2: MODE-nの後にBY 呼び名 を指定しなければなりません。

注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。

印字可能な文字の種類は、プリンタの持つ機能によって異なります。

UVPIデータストリームをVSPプリンタに出力する場合に使用できる機能については
PrintWalker/BPCのマニュアルを参照してください。

他社プリンタ装置へ印字した場合の注意事項については、“7.1.10 他社プリンタ装置での注意事項”を参照してください。

以下に指定できる印字属性について説明します。

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。

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します。 文字サイズと文字間隔を両方省略した場合、以下の文字サイズで印字します。 ・ 日本語項目 : 12ポ ・ 英数字項目 : 7ポ

指定方法の詳細については、"COBOL文法書"を参照してください。

注意

指定された文字サイズが印刷装置で使用できない場合、印字される文字サイズは印刷装置の仕様に従います。

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	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

指定方法の詳細については、"COBOL文法書"を参照してください。

注意

- 書体番号とは、PRINTING MODE句に指定した"FONT-nnn"のことを指します。
- データストリームがUVPIの場合には、書体番号が指定された印字文字の書体はPrintWalker/BPCが提供するcobolフィルタの仕様に従います。
- データストリームがPostScriptレベル1の場合およびListWorks連携による電子帳票出力を行う場合には、書体番号によって印字文字の書体を指定することができます。この場合、指定し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。ただし、以下の場合は、デフォルトの書体で印字されます。
  - フォントテーブル名を指定しない場合

— 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

フォントテーブルの詳細については、“7.1.9 フォントテーブル”を参照してください。

- データストリームがPRxxの場合には、書体番号を指定したプログラムの翻訳はできますが、有効にはなりません。デフォルトの書体で印字されます。
- 指定した文字書体が印刷装置で使用できない場合、印字される文字書体は印刷装置の仕様に従います。

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。

指定方法

文字スタイルの指定方法については、“7.1.9 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

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だけ指定できます。

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	文字形態
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。

指定方法の詳細については、“COBOL文法書”を参照してください。

注意

指定した文字形態が印刷装置で使用できない場合、印字される文字形態は印刷装置の仕様に従います。

印字文字の方向

横書き/縦書きを指定できます。

指定方法

指定方法	文字方向
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

指定方法	文字方向
	文字方向の指定を省略した場合、“横書き”で印字します。

指定方法の詳細については、“COBOL文法書”を参照してください。

指定された文字方向が印刷装置で使用できない場合、印字される文字方向は印刷装置の仕様に従います。

印字文字の間隔

0.01～24.00(単位:cpi)を指定できます。

指定方法

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します。 文字間隔と文字サイズを両方省略した場合、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下表に示します。

表7.3 印字文字の大きさ/形態と文字間隔

文字の大きさ	文字の形態							
	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

(単位:cpi)

表中で使用している記号の意味と指定方法の詳細は、“COBOL文法書”を参照してください。

注意

指定された文字間隔が印刷装置で使用できない場合、印字される文字間隔は印刷装置の仕様に従います。

7.1.4 環境変数の設定

印刷処理を行うプログラムの実行時に設定する環境変数を“表7.4 プログラムの実行に必要な環境変数”に示します。

表7.4 プログラムの実行に必要な環境変数

環境変数	指定する内容
CBR_LP_OPTION	使用するlpコマンドのオプション
CBR_PRINTFONTTABLE	フォントテーブルのパス名
CBR_PRT_INF	印刷情報ファイルのパス名
CBR_PRT_UTF8_CONVERT	Unicode(UTF-8)印刷未サポート機能／環境下での丸め印刷指示
CBR_FCB_NAME	デフォルトFCB名
FCBDIR	FCBの格納ディレクトリ
FOVLDIR	フォームオーバレイパターンの格納ディレクトリ
ASSIGN句で指定したファイル識別名	出力先ファイルまたはプリンタ情報ファイルのパス名
PRINTER-n (n=1～9)	出力先ファイルのパス名
LD_LIBRARY_PATH	副プログラムおよびシステム提供ライブラリの格納ディレクトリ

CBR_LP_OPTION

FORMAT句なし印刷ファイルで、ファイル管理記述項のASSIGN句の指定がPRINTERの場合に、lpコマンドに渡すオプションを環境変数CBR_LP_OPTIONに指定します。

Solaris 10の場合

例

印刷装置(lp0)、部数(2)、印刷モード(罫線接続あり)を指定する場合

```
$ CBR_LP_OPTION="-dlp0 -n2 -ykeisen" ; export CBR_LP_OPTION
```

省略した場合、起動されるlpコマンドには以下のオプションが指定されます。

- データストリームがUVPIの場合

```
-Tcobol, -y fp=ディレクトリ１, op=ディレクトリ２
```

```
-Tcobol       ：UVPIのデータストリームを使用する場合
ディレクトリ１：環境変数FCBDIRが指定されている場合
ディレクトリ２：環境変数FOVLDIR が指定されている場合
```

- データストリームがUVPI以外の場合

  オプションなし

Solaris 11の場合

例

印刷装置(lp0)、部数(2)、印刷モード(罫線接続あり)を指定する場合

```
$ CBR_LP_OPTION="-dlp0 -n2 -o -y_keisen" ; export CBR_LP_OPTION
```

省略した場合、起動されるlpコマンドには以下のオプションが指定されます。

- データストリームがUVPIの場合

```
-o -T_cobol, -o "-y_fp=ディレクトリ１ -y_op=ディレクトリ２"
```

```
-o -T_cobol     : UVPIのデータストリームを使用する場合
ディレクトリ１ : 環境変数FCBDIRが指定されている場合
ディレクトリ２ : 環境変数FOVLDIR が指定されている場合
```

- データストリームがUVPI以外の場合

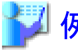
  オプションなし

注意

- 本機能を使って指定したオプションは、環境変数省略時のオプションの後ろに追加されるため、同じオプションを指定すると、lpコマンドのエラーメッセージが出力される場合があります。
- 本機能は以下のファイルに対しては有効になりません。
  - FORMAT句付き印刷ファイル
  - 表示ファイル
- 当環境変数を使用して印刷する場合、lpコマンドの印刷環境を整えておく必要があります。
- データストリームがUVPIでVSPプリンタへFNPエミュレーションでの印刷を行う場合、以下のオプションを指定してください。
  - -y truetypeオプション(Solaris 10の場合)
  - -o -y_truetypeオプション(Solaris 11の場合)
- データストリームがUVPIでシフトJISの印刷データを印刷する場合、以下のオプションを指定してください。
  - -y PCKオプション(Solaris 10の場合)
  - -o -y_PCKオプション(Solaris 11の場合)

CBR_PRINTFONTTABLE

実行単位全体で共通のフォントテーブルを使用する場合に指定します。フォントテーブルについては"7.1.9 フォントテーブル"を参照してください。


例

/home/usr1の下のフォントテーブルファイルfonttableを指定する場合

```
$ CBR_PRINTFONTTABLE=/home/usr1/fonttable ; export CBR_PRINTFONTTABLE
```

CBR_PRT_INF

FORMAT句なし印刷ファイルで、実行単位全体で共通の印刷情報ファイルを使用する場合に指定します。印刷情報ファイルについては"7.1.5 印刷情報ファイル"を参照してください。

例

/home/usr1の下の印刷情報ファイルinsatsu.infを指定する場合

```
$ CBR_PRT_INF=/home/usr1/insatsu.inf ; export CBR_PRT_INF
```

CBR_PRT_UTF8_CONVERT

FORMAT句なし印刷ファイルで、Unicode(UTF-8)印刷がサポートされていない機能やプリンタ装置を利用している環境において、Unicode(UTF-8)データを印刷可能な範囲に丸めて処理したい場合に指定します。指定可能な値は、以下の通りです。

- FJ_U90 : 標準コード変換を使用してU90コードに丸めて出力する。
- FJ_S90 : 標準コード変換を使用してS90コードに丸めて出力する。
- S90 : システムのコード変換関数を使用してS90コードに丸めて出力する。
- UTF8 : 丸めは行わず実行時エラーとする(省略値)。

なお、本環境変数の指定は、同一実行単位内のすべてのFORMAT句なし印刷ファイルに対して有効となります。ファイル単位に異なる指定を行う場合は、印刷情報ファイルにて同様の指定を行います。印刷情報ファイルの指定については、“7.1.5 印刷情報ファイル”を参照してください。

例

「Unicode(UTF-8)ロケール下でCOBOLアプリを実行し印刷処理を行いたいが、使用しているプリンタ装置がUnicode(UTF-8)印刷をサポートしていない。しかし、Unicode(UTF-8)固有文字を除いた範囲については可能な限り印刷したい。」という場合

```
$ CBR_PRT_UTF8_CONVERT=FJ_U90 ; export CBR_PRT_UTF8_CONVERT
```

CBR_FCB_NAME

FORMAT句付き印刷ファイルでFCBの省略値を変更する場合に指定します。FCBについては“7.1.6 FCB”を参照してください。

例

FCB1をデフォルトFCB名として使用する場合

```
$ CBR_FCB_NAME=FCB1 ; export CBR_FCB_NAME
```

注意

- FCB名は4文字以内の英数字の組合せで指定してください。
- デフォルトFCBを指定した場合、FCBの格納ディレクトリを環境変数FCBDIRで指定する必要があります。
- FORMAT句なし印刷ファイルの場合、デフォルトFCB名は印刷情報ファイルで指定します。

FCBDIR

プログラム中でFCB名を指定した制御レコードを出力し、FCBを使用する場合に指定します。FCBについては“7.1.6 FCB”を参照してください。

例

/home/usr1/fcbeの下のFCB1を使用する場合

```
$ FCBDIR=/home/usr1 ; export FCBDIR
```

FOVLDIR

FORMAT句なし印刷ファイルでフォームオーバレイ名を指定した制御レコードを出力し、フォームオーバレイパターンとの合成印刷を行う場合に指定します。フォームオーバレイパターンについては"7.1.7 フォームオーバレイパターン"を参照してください。

例

/home/usr1/kol5の下のOVL1を使用する場合

```
$ FOVLDIR=/home/usr1 ; export FOVLDIR
```

ASSIGN句で指定したファイル識別名

ファイル管理記述項のASSIGN句にファイル識別名を指定した場合、ファイル識別名を環境変数として、以下のファイルのパス名を設定します。

- FORMAT句なし印刷ファイルの場合:出力先ファイル

```
$ ファイル識別名=出力先ファイル名[,[,INF(印刷情報ファイル名),FONT(フォントテーブル名)]] ; export ファイル識別名
```

- FORMAT句付き印刷ファイルの場合:プリンタ情報ファイル

```
$ ファイル識別名=プリンタ情報ファイル[,,FONT(フォントテーブル名)] ; export ファイル識別名
```

- 表示ファイルの場合:プリンタ情報ファイル

```
$ ファイル識別名=プリンタ情報ファイル ; export ファイル識別名
```

印刷情報ファイル名には、印刷情報ファイルのパス名を指定します。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、"7.1.5 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パス名を指定します。フォントテーブルには、PRINTING MODE句のFONTnnn(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。フォントテーブルの詳細については、"7.1.9 フォントテーブル"を参照してください。

例

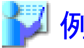
プログラム中の記述が"ASSIGN TO PRTFILE"であるFORMAT句なし印刷ファイルで、
/home/usr1の下のprtfileを出力先に割り当てる場合

```
$ PRTFILE=/home/usr1/prtfile ; export PRTFILE
```

PRINTER-n (n=1～9)

FORMAT句なし印刷ファイルでファイル管理記述項のASSIGN句にPRINTER-nを指定した場合、PRINTER-nを環境変数として、出力先ファイルのパス名を設定します。ただし、PRINTER-nはBourne shellでは使用できません。

以下にC shellでの実行例を示します。

 例

プログラム中の記述が"ASSIGN TO PRINTER-1"であるFORMAT句なし印刷ファイルで、
/home/usr1の下のprtfileを出力先に割り当てる場合

```
% setenv PRINTER-1 /home/usr1/prtfile
```

LD_LIBRARY_PATH

COBOLランタイムシステムおよび、印刷処理に必要なシステム提供ライブラリの格納ディレクトリを指定します。

LD_LIBRARY_PATHにMeFtの格納ディレクトリ/opt/FJSVmeft/libを追加する場合

```
$ LD_LIBRARY_PATH=/opt/FJSVmeft/lib:$LD_LIBRARY_PATH ; export LD_LIBRARY_PATH
```

7.1.5 印刷情報ファイル

印刷情報ファイルは、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使用するテキスト形式のファイルです。印刷情報ファイルでは、出力される帳票に関するいくつかの状態制御情報を設定します。

FORMAT句なし印刷ファイルでは、以下の場合には印刷情報ファイルの指定が必要です。

- 他社プリンタに印刷する
- ページのフォーマットを変更する
- ListWorks連携による電子帳票出力を行う

印刷情報ファイルの指定方法

印刷情報ファイルを指定するには、環境変数CBR_PRT_INFに印刷情報ファイルのパス名を指定します。この場合、ASSIGN句の記述に関係なく実行単位中のすべての印刷ファイルに同じ印刷情報ファイルが対応付けられます。

ASSIGN句の記述がファイル識別名、ファイル識別名定数、データ名またはPRINTER-nの場合には、印刷情報ファイルを印刷ファイルごとに指定することができます。印刷ファイルごとに印刷情報ファイルを指定するには、出力対象となる印刷ファイルのパス名のあとに、",,INF(印刷情報ファイルのパス名)"を指定します。ただし、印刷情報ファイルとフォントテーブルを同時に指定する場合には、以下のように、印刷情報ファイルのパス名とフォントテーブルのパス名をコンマで区切って指定してください。

```
"印刷ファイルのパス名,,INF(印刷情報ファイルのパス名),FONT(フォントテーブルのパス名)"
```

印刷情報ファイルのパス名に相対パスを指定した場合にはカレントディレクトリからの相対パスを検索します。

印刷ファイルごとの指定と実行単位全体の指定が共存する場合には、印刷ファイルごとの指定を優先します。

プログラム中の記述が"ASSIGN TO PRTFILE"である印刷ファイルで、/home/usr1の下のprtfileを出力先に、insatsu.infを印刷情報ファイルとして割り当てる場合

```
$ PRTFILE="/home/usr1/prtfile,,INF(/home/usr1/insatsu.inf)" ; export PRTFILE
```

プログラム中の記述が"ASSIGN TO PRTFILE"である印刷ファイルで、出力先の指定を省略し、/home/usr1の下のinsatsu.infを印刷情報ファイルとして、fonttabをフォントテーブルとして割り当てる場合

```
$ PRTFILE=",,INF(/home/usr1/insatsu.inf),FONT(/home/usr1/fonttab)" ; export PRTFILE
```

- "INF","FONT"は必ず大文字で記述してください。
- 出力先の指定を省略した場合、印刷情報ファイルのprtout制御文の指定が必要です。

印刷情報ファイルの記述形式

印刷情報ファイルの解析は印刷ファイルに対するOPEN文の実行時に行い、その内容は印刷ファイルに対するCLOSE文の実行まで有効となります。

印刷情報ファイルは、次の制御文から構成されるテキスト形式のファイルです。

- printer制御文
- papersize制御文(データストリームがUVPIの場合は無効)
- prtform制御文(データストリームがUVPIの場合は無効)
- fcbname制御文(データストリームがUVPIの場合は無効)
- ankfont制御文(データストリームがPostScriptレベル1の場合だけ有効)
- stream制御文
- prtout制御文
- streamenv制御文(電子帳票出力の場合だけ有効)
- documentname制御文(電子帳票出力の場合だけ有効)
- utf8_convert制御文

注意

データストリームがUVPIの場合、デフォルトの用紙サイズ(papersize)、印刷形式(prtform)、FCB名(fcbname)はlpコマンドのオプションで変更することができます。

以下に印刷情報ファイルの記述例と各制御文の記述形式を示します。

```
printer      PR70
papersize    a4
prtform      l
fcbname      fcb1
ankfont      2-byte
prtout       /home/usr1/prtfile
utf8_convert FJ_U90
```

注意

- 各制御文のキーワードとパラメタの区切り文字は、空白およびタブです。
- 行の先頭に“#”またはセミコロン(;)を記述した場合、その行はコメント行とみなします。
- 同一の制御文を複数回指定した場合、最後に指定したものを有効とします。
- 印刷情報ファイルに指定する絶対パスのファイル名および絶対パス名は、二重引用符(")で囲まないでください。

printer制御文

printer制御文は、出力想定プリンタに対応したデータストリーム種別を指定します。

キーワード	パラメタ
printer	出力データストリーム種別

パラメタの説明

出力想定プリンタに対応したデータストリーム種別を以下の文字列で指定します。

値	意味
PR70	ESC/P J84 プリンタ
PR75	ESC/Pageプリンタ (LP-8000ほか)
PR80	LIPS III (LBP-B406Gほか)
UVPI	UVPI (PrintPartner VSPシリーズほか)
PS1	PostScriptレベル1
PS2	PostScriptレベル2

省略時の解釈

printer制御文が指定されなかった場合、システムの標準データストリーム(UVPI)が指定されたものとみなします。

papersize制御文

papersize制御文は、デフォルトの用紙サイズを指定します。

キーワード	パラメタ
papersize	デフォルト用紙サイズ

パラメタの説明

デフォルトの用紙サイズを以下の文字列で指定します。

値	意味
a3	A3サイズの用紙
a4	A4サイズの用紙
a5	A5サイズの用紙
b4	B4サイズの用紙
b5	B5サイズの用紙
ltr	LETTERサイズの用紙

省略時の解釈

papersize制御文が指定されなかった場合、a4が指定されたものとみなします。

注意

連帳用紙の指定はできません。

prtform制御文

prtform制御文は、デフォルトの印刷形式を指定します。

キーワード	パラメタ
prtform	デフォルト印刷形式

パラメタの説明

デフォルトの印刷形式を以下の文字列で指定します。

値	意味
p	ポートレート

値	意味
l	ランドスケープ
pz	ポートレート縮小
lz	ランドスケープ縮小
lp	LP縮刷

省略時の解釈

PostScriptレベル1以外のデータストリームで印刷情報ファイルおよびI制御レコードで印刷形式を指定しない場合には、ポートレート("p")が指定されたものとみなします。なお、PostScriptレベル1のデータストリームで印刷情報ファイルおよびI制御レコードで印刷形式を指定しない場合には、LP縮刷("lp")が指定されたものとみなします。

注意

実際の印刷結果はプリンタ装置の機能に依存します。

fcbname制御文

fcbname制御文は、使用するFCB名を指定します。

キーワード	パラメタ
fcbname	デフォルトFCB名

パラメタの説明

使用するFCB名を4文字までの英数字で指定します。

省略時の解釈

fcbname制御文が指定されなかった場合、FCBが指定されなかったものとみなし、FCBの省略値を採用します。FCBの省略値については、“7.1.6 FCB”を参照してください。

注意

fcbname制御文を指定した場合、環境変数FCBDIRの指定が必要です。

ankfont制御文

ankfont制御文は、英数字項目の印刷文字(日本語/ASCII)を指定します。

キーワード	パラメタ
ankfont	英数字項目の印刷文字

パラメタの説明

英数字項目の印刷文字を以下の文字列で指定します。

値	意味
1-byte	1 byteのASCII英数字
2-byte	2 byteの日本語英数字

省略時の解釈

ankfont文が指定されなかった場合、2-byteが指定されたものとみなします。

stream制御文

stream制御文は帳票の出力方式を指定します。

キーワード	パラメタ
stream	出力方式

パラメタの説明

出力方式には通常印刷を行うか電子帳票出力を行うかを以下の文字列で指定します。

値	意味
PR	通常印刷
LW	電子帳票出力(ListWorks連携)

省略時の解釈

stream制御文が指定されなかった場合、通常印刷(PR)が指定されたものとみなします。

注意

- 通常印刷を指定した場合、printer制御文に指定されたデータストリーム種別に従って印刷ファイルを出力します。電子帳票出力を指定した場合、printer制御文は意味を持ちません。
- 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については、"7.6 電子帳票出力機能を使う方法"およびListWorksのマニュアルを参照してください。

prtout制御文

印刷データの出力先を指定します。prtout制御文の指定はCOBOLプログラム中のASSIGN句の記述よりも優先されます。ASSIGN句で指定した出力先を変更したいとき、またはASSIGN句の記述で出力先の指定を省略したときに指定します。

キーワード	パラメタ
prtout	印刷データの出力先

パラメタの説明

通常印刷時には出力する印刷ファイルのパス名を、電子帳票出力時(stream制御文にLWを指定している場合)には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名と呼びます)を指定します。

省略時の解釈

prtout制御文が指定されなかった場合、ASSIGN句の記述に対する指定が有効になります。prtout制御文が省略されて、かつASSIGN句に有効な指定がない場合には実行時にエラーとなります。

注意

prtout制御文ではプリンタへの直接印刷(ASSIGN TO PRINTER)の指定はできません。また、本指定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。

streamenv制御文

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パス名を指定します。streamenv制御文は電子帳票出力時(stream制御文にLWを指定した場合)だけ有効です。

キーワード	パラメタ
streamenv	電子帳票情報ファイルのパス名

パラメタの説明

電子帳票出力時に使用する電子帳票情報ファイルのパス名を絶対パスで指定します。

省略時の解釈

streamenv制御文が指定されなかった場合、ListWorksが保持する省略時の解釈に従って動作します。

documentname制御文

印刷ファイルに対する文書名を指定します。documentname制御文は電子帳票出力時(stream制御文にLWが指定された場合)だけ有効です。ただし、本制御文の指定は、電子帳票情報ファイルで日本語帳票名の指定が省略された場合に有効となります。

キーワード	パラメタ
documentname	文書名

パラメタの説明

印刷ファイルの文書名(帳票名)を指定します。文書名が64バイトを超える場合には先頭から64バイトが文書名として有効になります。

省略時の解釈

電子帳票情報ファイルに日本語帳票名が指定されている場合には、そちらが有効となります。どちらも省略されている場合には、文書名(帳票名)は設定されません。

utf8_convert制御文

utf8_convert制御文は、FORMAT句なし印刷ファイルで、Unicode(UTF-8)印刷がサポートされていない機能やプリンタ装置を利用している環境において、Unicode(UTF-8)データを印刷可能な範囲に丸めて処理したい場合に指定します。

なお、utf8_convert制御文の指定は、ファイル単位に有効な指定となります。同一実行単位内に含まれるすべてのFORMAT句なし印刷ファイルに対して共通の指定を行う場合は、実行環境変数にて同様の指定を行います。実行環境変数の指定については、“7.1.4 環境変数の設定”を参照してください。

キーワード	パラメタ
utf8_convert	本機能を活性化するか否かを指定

パラメタの説明

Unicode(UTF-8)データの丸め印刷を行う／行わないを以下の文字列で指定します。

値	意味
FJ_U90	標準コード変換を使用してU90コードに丸めて出力する
FJ_S90	標準コード変換を使用してS90コードに丸めて出力する
S90	システムのコード変換関数を使用してS90コードに丸めて出力する
UTF8	Unicode(UTF-8)データの丸めを行わない

省略時の解釈

utf8_convert制御文が指定されなかった場合、Unicode(UTF-8)データの丸めを行わない(UTF8)が指定されたものとみなします。この場合、実行環境変数“CBR_PRT_UTF8_CONVERT”の指定が有効となります。実行環境変数“CBR_PRT_UTF8_CONVERT”の指定も省略されている場合は、実行時エラーとなります。

7.1.6 FCB

FCBは、1ページ分の行数、行間隔および印字開始行を変更したい場合に使用します。

FCBの指定方法

FCBを使用するには、まずfcbeという名前のディレクトリの下にFCBを作成します(fcbeというディレクトリ名は固定です)。そして、I制御レコードにFCB名を指定し、プログラム実行時にfcbeディレクトリの直上のディレクトリのパス名を、環境変数FCBDIRに設定します。ただし、UVPIデータストリームでデータをファイルに出力する場合には、lpコマンド起動時に以下のオプションでFCB格納ディレクトリを指定します。

- -y fp=ディレクトリ（Solaris 10の場合）
- -o -y_fp=ディレクトリ(Solaris 11の場合)

また、以下の方法を使用して、デフォルトで使用するFCBの名前を指定することができます。デフォルトFCBは、I制御レコードでFCBが指定されていない場合に有効です。

- FORMAT句なし印刷ファイル
  - データストリームがUVPI以外の場合
    - 印刷情報ファイルのfcbname制御文
  - データストリームがUVPIの場合
    - lpコマンドの-y fp=ディレクトリ オプション(Solaris 10の場合)
    - lpコマンドの-o -y_fp=ディレクトリ オプション(Solaris 11の場合)
- FORMAT句付き印刷ファイル
  - 環境変数CBR_FCB_NAME

FCBが省略された場合、印刷情報は“表7.5 FCBの省略値”に示す値となります。

表7.5 FCBの省略値

印刷情報	FORMAT句なし印刷ファイル		FORMAT句付き印刷ファイル	
	UVPIデータストリーム以外	UVPIデータストリーム	帳票定義体を使用しない	帳票定義体を使用する
用紙の大きさ	11インチ	lpコマンドのcobolフィルタの省略値	11インチ	帳票定義体に指定した値
行間隔	6LPI		6LPI	
行数	66		66	
印字開始行	4		4	（注）

注:浮動パーティション出力時は4、固定パーティション出力時は帳票定義体に指定した値

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、“7.1.15 I制御レコード/S制御レコード”を参照してください。

FCBの形式

FCBは、次の2種類の制御文から構成されるテキスト形式のファイルです。

- lpi制御文
- print制御文

以下に、FCBの各制御文の記述形式を説明します。

- 各制御文のキーワードとパラメタの区切り文字は、空白およびタブです。
- 行の先頭に“#”を記述した場合、その行はコメント行とみなします。

lpi制御文

キーワード	パラメタ
lpi	行間隔 行数

行間隔

行間隔の長さを、行間隔の単位(1/7200インチ)によって指定します。

lpi制御文では、以下に示す値だけ指定可能です。

行間隔	指定する値
6LPI	1200
8LPI	900
12LPI	600

行数

行間隔が適用される行の数を指定します。行間隔×行数の値が次のprint制御文で指定する用紙の大きさを超える場合には、実行時にFCBのエラーとなります。

print制御文

キーワード	パラメタ
Print	印刷開始行 ［用紙の大きさ］

印刷開始行

ページ内の印刷開始行を指定します。

用紙の大きさ

使用する用紙の縦方向の長さを指定します(単位:1/7200インチ)。省略された場合は、79200(11インチ)が指定されたものとみなされます。なお、用紙の長さは3600/7200(0.5インチ)単位で切り上げられます。

以下にカット紙の物理的な用紙長を1/7200インチ単位で表した値を示します。ただし、この値にはプリンタの印字不可能域が含まれているので、実際に指定する場合は、印字不可能域の考慮が必要です。FCBで指示した用紙長よりも実際の用紙長が短い場合、その間のデータが失われる可能性があるので正しく指定してください。

用紙サイズ	用紙方向	
	ポートレート	ランドスケープ
A3	119000	84200
A4	84200	59500
A5	59500	42100
B4	103200	72900
B5	72900	51600
LETTER	79200	61200

このシステムでは、FCBによるCHANNEL位置の設定はできません。CHANNEL-02～CHANNEL-12に対応づけられた呼び名を指定したWRITE文は、ADVANCING 1 LINE指定のWRITE文と同じ動作をします。なお、CHANNEL-01に対応づけられた呼び名を指定したWRITE文では、改ページ処理を行います。

FCBの記述例

以下のような指定を行うFCBの記述例を示します。

- 用紙サイズ:A4
- 用紙方向:ランドスケープ
- 印字開始行:1行目
- 行間隔:6lpi

```
print    1  55900
lpi   1200     46
```

解説

用紙の物理長は59500/7200インチですが、印字不可能域が上下合わせて3600/7200インチ(=1/2インチ)ある場合、印字可能域は以下のように計算できます。

```
59500/7200 - 3600/7200 = 55900/7200
```

印字可能域に対して、行間隔6lpi(1200/7200インチ)で印字できる1ページの行数を計算すると、以下のようになります。

```
(55900/7200) / (1200/7200) = 46.58
```

注意

印字不可能域の大きさはプリンタによって異なります。

7.1.7 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ができます。

フォームオーバレイパターン(KOL5/KOL6形式)は、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を使用して印刷することができます。

フォームオーバレイパターンを使用するには、kol5という名前のディレクトリの下にフォームオーバレイパターンを格納します(kol5というディレクトリ名は固定です)。FORMAT句なし印刷ファイルの場合、プログラム実行時にkol5ディレクトリが格納されているパス名を環境変数FOVLDIRに設定します。ただし、UVPIデータストリームでデータをファイルに出力する場合には、lpコマンド起動時に以下のオプションでフォームオーバレイパターン格納ディレクトリを指定します。

- -y op=ディレクトリ名(Solaris 10の場合)
- -o -y_op=ディレクトリ名(Solaris 11の場合)

FORMAT句付き印刷ファイルおよび表示ファイルの場合、フォームオーバレイパターン格納ディレクトリをプリンタ情報ファイルに指定します。

注意

I制御レコードによるフォームオーバレイパターンを使った帳票印刷を行う場合、オーバレイパターンファイル名は4文字以内の英数字の組み合わせ(拡張子なし)である必要があります。I制御レコードにはフォームオーバレイパターンファイル名と同じ名前を指定してください。

フォームオーバレイパターンの作成方法については、FORMのマニュアルまたはヘルプ、またはPowerFORMヘルプを参照してください。I制御レコードによる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、“7.3 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。

7.1.8 帳票定義体

FORMまたはPowerFORMを使って帳票を設計すると、帳票定義体が作成されます。COBOL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。

帳票定義体の作成方法については、FORMのマニュアルまたは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、"7.4 帳票定義体を使う印刷ファイルの使い方"または"7.5 表示ファイル(帳票印刷)の使い方"を参照してください。

PowerFORMで作成した帳票定義体をSolaris上で印刷する場合の注意事項については、MeFtのオンラインマニュアルを参照してください。

注意

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。

- COBOL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。

7.1.9 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。フォントテーブルを使用することにより、特殊名段落のPRINTING MODE句に指定した書体番号(FONT-nnn)に対して、任意のフォントを対応付けることができます。

フォントテーブルの指定方法

フォントテーブルを指定するには、環境変数CBR_PRINTFONTTABLEにフォントテーブルのパス名を指定します。この場合、ASSIGN句の記述に関係なく実行環境単位中のすべての印刷ファイルに同じフォントテーブルが対応付けられます。

ASSIGN句の記述がファイル識別名、ファイル識別名定数、データ名またはPRINTER-nの場合には、フォントテーブルを印刷ファイルごとに指定することができます。印刷ファイルごとにフォントテーブルを指定するには、出力対象となる印刷ファイルのパス名のあとに、",,FONT(フォントテーブルのパス名)"を指定します。ただし、印刷情報ファイルとフォントテーブルを同時に指定する場合には、以下のように印刷情報ファイルのパス名とフォントテーブルのパス名をコンマで区切って指定してください。

```
"印刷ファイルのパス名,,INF(印刷情報ファイルのパス名),FONT(フォントテーブルのパス名)"
```

フォントテーブルのパス名に相対パスを指定した場合にはカレントディレクトリからの相対パスを検索します。

印刷ファイルごとの指定と実行単位全体の指定が共存する場合には、印刷ファイルごとの指定を優先します。

例

プログラム中の記述が"ASSIGN TO PRTFILE"である印刷ファイルで、/home/usr1の下のprtfileを出力先に、fonttabをフォントテーブルとして割り当てる場合

```
$ PRTFILE="/home/usr1/prtfile,,FONT(/home/usr1/fonttab)" ; export PRTFILE
```

プログラム中の記述が"ASSIGN TO PRTFILE"である印刷ファイルで、出力先の指定を省略し、/home/usr1の下のinsatsu.infを印刷情報ファイルとして、fonttabをフォントテーブルとして割り当てる場合

```
$ PRTFILE=",,INF(/home/usr1/insatsu.inf),FONT(/home/usr1/fonttab)" ; export PRTFILE
```

注意

- “INF”,“FONT”は必ず大文字で記述してください。
- FORMAT句なし印刷ファイルで出力先の指定を省略した場合、印刷情報ファイルのprtout制御文の指定が必要です。
- FORMAT句付き印刷ファイルおよび表示ファイルの場合、プリンタ情報ファイル名の指定を省略することはできません。

フォントテーブルの形式

フォントテーブルの形式を以下に示します。

```
[書体番号]                              セクション名（書体番号ごとに指定）
FontName=フォントフェース名             文字書体指定（省略可）
Style={  R  |  B  |  I  |  BI  }        文字スタイル指定(省略可)
```

- 行の先頭に“;”を記述した場合、その行はコメント行とみなします。
- 同一の書体番号を複数回指定した場合、最初に指定したものを有効とします。

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なの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。

```
[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、システムの定める書体で印字されます。

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には、プリンタ装置が直接認識できる書体名を指定します。プリンタ付属のマニュアルなどで印字できる書体を確認し、正しく指定してください。

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、標準で印字されます。

- R:標準
- B:太字
- I:斜体
- BI:太字・斜体

ただし、Styleによるスタイル指定は、FORMAT句なし印刷ファイルではListWorks連携による電子帳票出力を行う場合だけ有効です。FORMAT句付き印刷ファイルではPostScriptレベル2またはListWorks連携の場合に有効です。

フォントテーブル使用時の注意事項

書体番号によるフォント指定はデータストリームがPostScriptレベル1の場合またはListWorks連携による電子帳票出力を行う場合だけ有効です。

データストリームがPostScriptレベル1の場合には以下のことに留意してください。

- 日本語フォントを使用する場合には、文字セット、コード系、文字方向(縦書き/横書き)を意識してフォントフェース名を指定する必要があります。

例

Ryumin-Light書体をEUCコード系かつ横書きで使用する場合

```
FontName=Ryumin-Light-EUC-H
```

例

GothicBBB-Medium書体をシフトJISコード系かつ縦書きで使用する場合

```
FontName=GothicBBB-Medium-RKSJ-V
```

- Style=にスタイルを指定しても有効になりません。PostScriptプリンタには通常スタイルごとにフォントが用意されているので、使用したいスタイルの属性を持ったフォントのフォントフェース名をFontName=に指定してください。

例

太字体のCourier(Courier-Bold)を使用する場合

```
FontName=Courier-Bold
```

斜体のCourier(Courier-Oblique)を使用する場合

```
FontName=Courier-Oblique
```

- 以下に示すディレクトリにフォントテーブルファイルのひな型fonttableが格納されています。フォントテーブルを使用する場合には、このファイルをカストマイズして使用してください。システムで使用している番号にフォントを割り当てると正しく動作しない場合があるので、新しくフォントを割り当てる場合は未使用の番号(301から800を推奨)を使用してください。

  [ひな型格納場所]

  COBOLインストールディレクトリ/config/template/C/fonttable

- FORMAT句なし印刷ファイルでは、ANK文字を2バイト文字に変換して日本語フォントを使用して印字しています。このため、ANK文字を1バイトフォントで印字するには、印刷情報ファイルのankfont制御文に“1-byte”を指定する必要があります。

- 以下に示す書体番号と書体名の指定は等価に扱われます。これらの書体番号は絶対にカストマイズしないでください。印刷結果が異常となります。

  - FONT-001とMINCHOU
  - FONT-002とMINCHOU-HANKAKU
  - FONT-003とGOTHIC
  - FONT-004とGOTHIC-HANKAKU

ListWorks連携による電子帳票出力を行う場合の注意事項は“7.6 電子帳票出力機能を使う方法”を参照してください。

例

フォントテーブルの作成例(PostScriptレベル1の場合)

```
[FONT-301]
FontName=Ryumin-Light-EUC-H
```

```
[FONT-302]
FontName=Ryumin-Light-EUC-V
```

7.1.10 他社プリンタ装置での注意事項

全般的な注意事項

他社プリンタ装置の定義

UVPIおよびPostScript以外の他社プリンタ装置への印刷を行う場合、以下に示すプリンタ定義が必要です。

admintoolまたはlpadminコマンドを使用してプリンタを定義します。定義時のPrinter TypeとContent Typeには、次の値を指定してください。

－ Printer Type : unknown

－ Content Type : any

注意

Printer Typeに適当な値を指定したい場合は、terminfoデータベースに適当な情報を定義する必要があります。

他社プリンタ装置の接続

プリンタをシリアルポード(RS232Cなど)に接続する場合、印刷データ中にバイナリデータ(イメージデータなど)が混在すると、正しく印刷されないことがあります。これは、バイナリデータ中のNL、CRに相当するコードが変換されるためです。以下のコマンドを指定することにより回避してください。

```
% lpadmin  -p  プリンタ名  -o  stty="'…'"
```

sttyのパラメタには、以下の値を指定してください。

値	意味
-olcuc	小文字のアルファベットを大文字に変換しない
-onlcr	NLをCR_NLに変換しない
-ocrnl	CRをNLに変換しない

I/S制御レコードの機能に関する注意事項

他社プリンタ装置により、使用できるI/S制御レコードの機能が異なります。

このため、以下の注意事項に従った指定を行う必要があります。I/S制御レコードについては、“7.3 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。また、“表7.6 I/S制御レコード機能の指定可否”に他社プリンタ装置に対するI/S制御レコードの指定可否を示します。

他社プリンタ装置共通の注意事項

装置に未搭載の書体を指定した場合は、文字が出力されません。

LIPS IIIプリンタ装置(PR80)およびESC/Pageデータストリーム(PR75)固有の注意事項

－ オーバレイパターン名は、以下の順に検索されます。

1. 環境変数FOVLDIRに指定したディレクトリ
2. システムディレクトリ(/usr/spool/lp/kol5)

－ オーバレイパターン名の指定は1つだけです(単一オーバレイ)。

－ 焼き付け回数の指定は、複写数の指定と同じ値を指定します。違う値が指定された場合、実行時にエラーとなります。

表7.6 I/S制御レコード機能の指定可否

		プリンタ装置識別文字列				
		PR70	PR75	PR80	PS1	PS2
I/S制御レコードの機能	フォームオーバレイモジュール名	×	○	○	○	○
	焼き付け回数	×	○ (注1)	○ (注1)	○ (注8)	○ (注1)
	複写数	×	○	○	○	○
	FCB名（**FCB**）	○	○	○	○	○
	フォーマット定義体名	×	×	×	×	×
	複写修正モジュール	×	×	×	×	×
	複写修正開始番号	×	×	×	×	×
	複写修正用文字配列テーブル番号	×	×	×	×	×
	用紙識別名	×	×	×	×	×
	文字配列テーブルまたは追加文字セット	×	×	×	×	×
	ダイナミックロード	×	×	×	×	×
	オフセットスタック	×	×	×	×	×
	印刷形式	○ (注2)(注3)	○ (注2)	○ (注2)	○	○ (注2)
	用紙サイズ	○ (注3)(注11)	○ (注11)	○ (注4)(注11)	○ (注10)(注11)	○ (注11)
	用紙供給口	○ (注5)	○ (注7)	×	○ (注9)(注10)	○ (注7)
	用紙排出口	×	×	×	×	×
	印刷面指定	×	×	×	×	×
	印刷面位置付け	×	×	×	×	×
	印字禁止領域	○	○	○	×	○
	とじしろ方向	×	×	×	○	×
	とじしろ幅および印刷開始原点位置	×	○ (注6)	○ (注6)	○	×

○：指定可能
×：指定不可能

注1: 焼き付け回数の指定は、複写数の指定と同じ値を指定します。違う値を指定した場合、実行時エラーとなります。

注2: ポートレートモード、ランドスケープモードだけ指定できます。

注3: 印刷形式を指定する場合、用紙サイズも指定する必要があります。

注4: レター用紙は指定できません。

注5: ホッパのない装置に対しては無効です。

注6: とじしろ幅は無視されます。

注7: 主供給口1、主供給口2および副供給口からの給紙だけ可能です。

注8: 焼き付け回数の指定は、複写数の指定と同じ値とみなします。

注9: 主供給口1および主供給口2だけ指定可能です。

注10: プリンタ装置によっては有効にならない場合があります。この場合、lpコマンドのオプションで指定してください。ただし、指定可能なオプションはフィルタおよび装置の仕様によって異なるので注意してください。

注11: 単票用紙のみ指定できます。

文字属性に関する注意事項

ESC/Pプリンタ装置(PR70)への印刷を行う場合、印字可能な文字属性に以下の注意が必要です。
なお、以下の文中に現れる2バイト文字とは、日本語項目および日本語編集項目を示し、1バイト文字とは、日本語項目および日本語編集項目以外の項目を示します。

ESC/Pプリンタ装置(PR70)の注意事項

- ANK文字は、縦書き(反時計回りに90度回転)で印字されません。
- 文字サイズは、装置がサポートしている文字サイズの範囲内で、指定された大きさ以下の最大の文字を出力します。指定された大きさ以下の文字がない場合、装置がサポートしている最小の文字を出力します。
装置がサポートしている文字サイズは、デフォルト書体の場合、以下のとおりです。
    - 1バイト文字
9.6ポイント(標準、平体、半角)
    - 2バイト文字
9.6ポイント(標準、平体、半角、下付き、上付き、半角平体、下付き平体、上付き平体)
- 指定可能な文字ピッチは以下の値です。
    - 1バイト文字
20.0CPI～1.4CPI
    - 2バイト文字
15.0CPI～1.3CPI

  文字の開始点から次の文字の開始点までのドット数(180dpi換算)が整数値とならない文字ピッチを指定した場合、ドット数は小数点以下を四捨五入した値となります。そのため、正確な文字ピッチとはなりません。
- 文字同士が重なる文字ピッチを指定しても、文字は重なりません。文字ピッチを優先して文字を配置し、文字が重ならない文字幅で文字を出力します。
- 文字が用紙の右端からはみ出した場合、改行が行われます。この場合、以降の文字列の出力位置は保証されません。
- 行方向が用紙長をはみ出した場合、改ページが行われます。

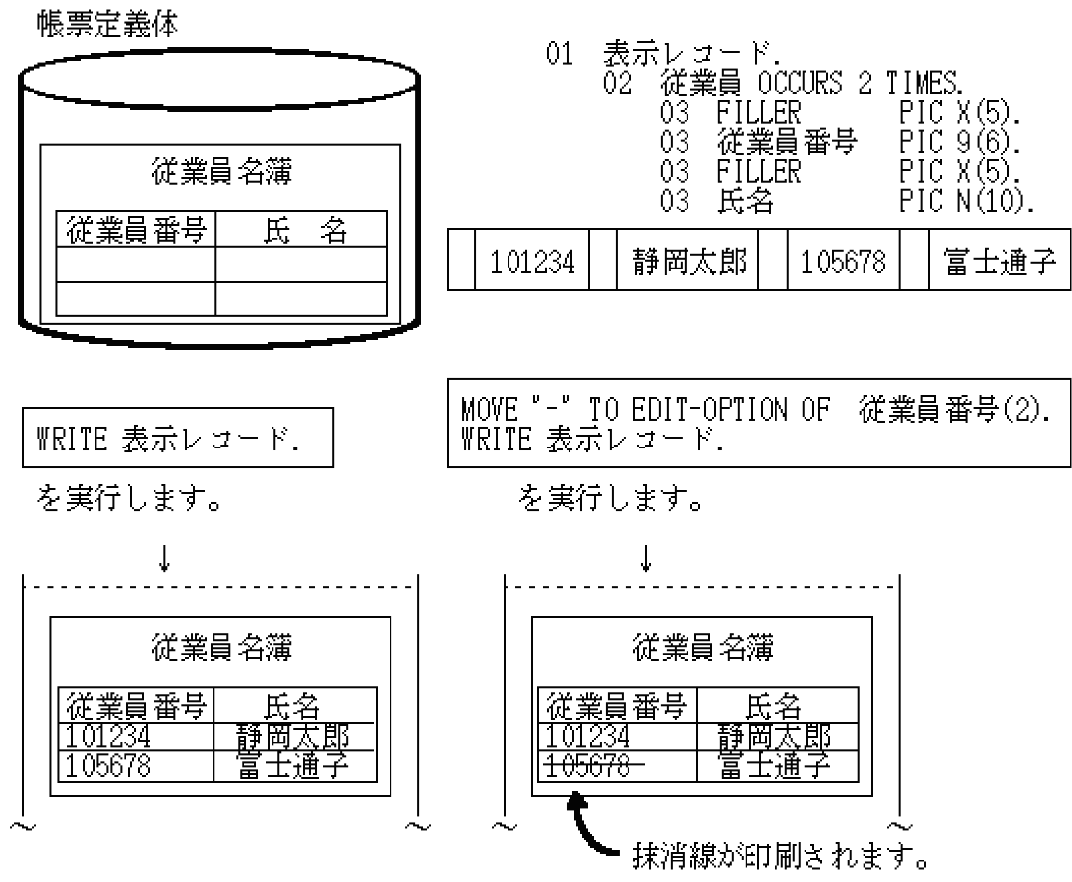
7.1.11 特殊レジスタ

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。
- 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 7.2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。

7.1.12 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)	○ (注)	○ (注)	○ (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。

7.1.13 帳票設計について

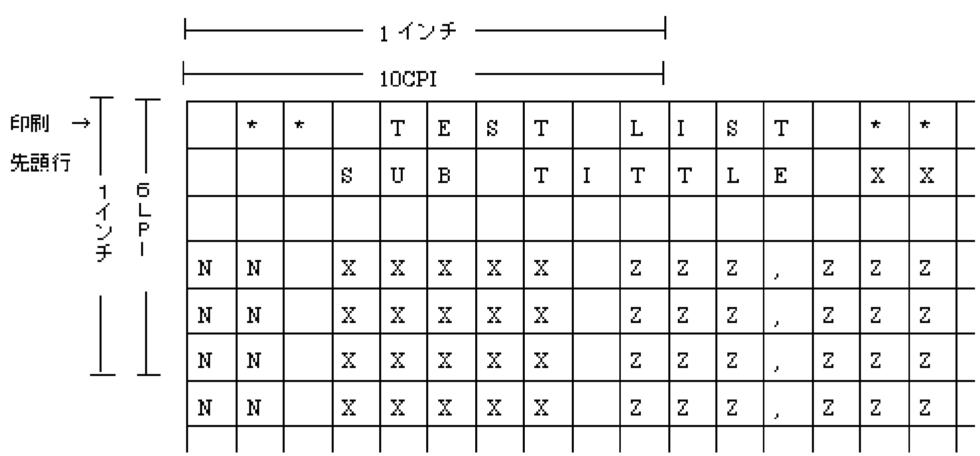
COBOLの帳票印刷には、行と桁の概念が不可欠です。特にFORMAT句なし印刷ファイルのように行レコード主体で帳票印刷を行う場合、実際にプログラムを作成する前にきめ細かい帳票設計が必要です。

このため、実際にプログラミングに入る前に、まず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てください。それから、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ファイルのlpi制御文やprint制御文に対応(反映)させてください。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ファイルの記述

```
lpi        1200    66
print      1
```

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.
```

7.1.14 印刷不可能な領域について

1. **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います。このため、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。

2. **用紙内で印字可能な行数**

   用紙内に印字可能な行数は、FCBファイルの定義により決定します。

   FCBファイルでは、印刷開始行位置、用紙長、および行間隔(LPI)とその行間隔が有効な行数を指定します。用紙内の印字可能行数は、用紙長と行間隔から決まります。

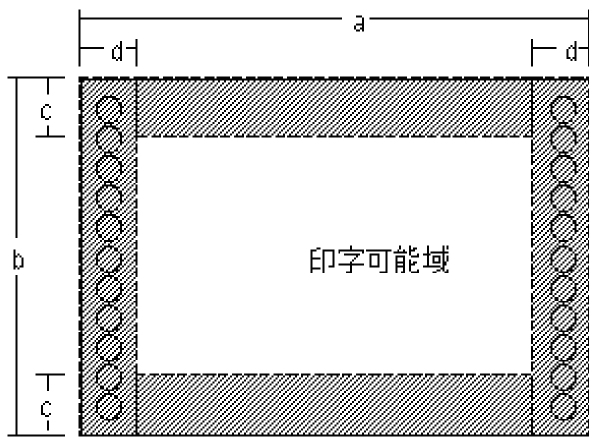
   以下に、15×11インチの連続用紙を使用した場合のFCBファイルの定義例を示します。

   - 行間隔:6LPI
   - 用紙内の最大印字可能行数:66行
   - 用紙サイズ(縦方向):11インチ(1/7200インチ単位で指定)

例

FCBファイルの定義例

```
lpi        1200    66
print      1       79200
```



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lpシステムのフィルタはこの部分への印字を抑止していることがあります(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様々であり、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。

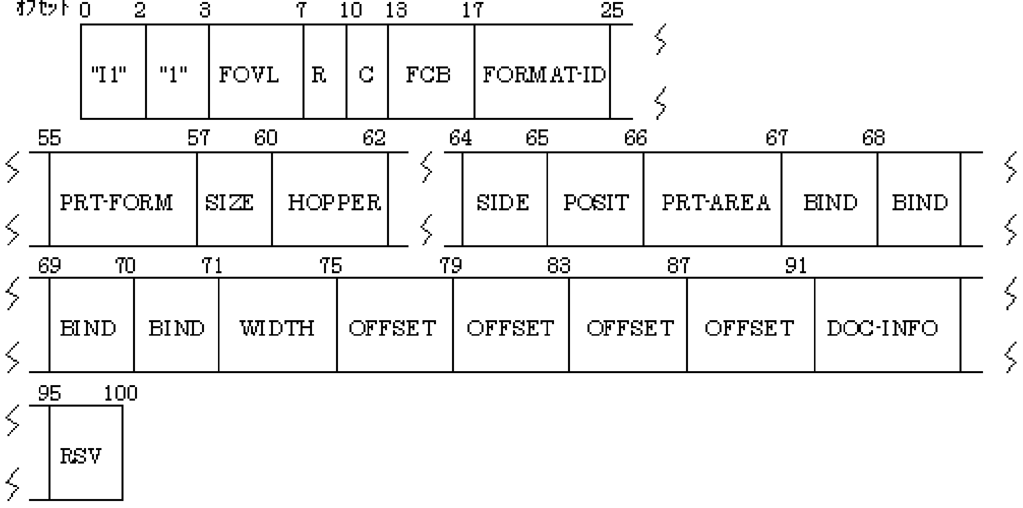
7.1.15 I制御レコード/S制御レコード

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。制御レコードの形式を以下に示します。

なお、プリンタ装置により有効となる機能が異なります。プリンタ装置の取扱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.
    02  識別子              PIC  X(2)  VALUE  "I1".
    02  形式                PIC  X(1)  VALUE  "1".
    02  オーバレイ名        PIC  X(4).              →FOVL
    02  焼き付け回数        PIC  9(3).              →R
    02  複写数              PIC  9(3).              →C
    02  ＦＣＢ名            PIC  X(4).              →FCB
    02  画面帳票定義体名    PIC  X(8).              →FORMAT-ID
    02                      PIC  X(30).
```

```
    02  印字形式              PIC  X(2).                  →PRT-FORM
    02  用紙サイズ            PIC  X(3).                  →SIZE
    02  用紙供給口            PIC  X(2).                  →HOPPER
    02                        PIC  X(2).
    02  印刷面指定            PIC  X(1).                  →SIDE
    02                        PIC  X(1).
    02  印字禁止域            PIC  X(1).                  →PRT-AREA
    02  とじしろ方向.
            03                PIC  X     OCCURS 4 TIMES.  →BIND
    02  印字位置情報.
      03  とじしろ幅          PIC  9(4).                  →WIDTH
      03  印刷原点位置.
            04                PIC  9(4)  OCCURS 4 TIMES.  →OFFSET
    02                        PIC  X(9)  VALUE  SPACE.    →RSV
```

FOVL

使用するフォームオーバレイパターン名を指定します。ただし、オーバレイパターンだけ指定できます。オーバレイグループを指定した場合、データストリームの種別がUVPIである場合だけ、先頭のオーバレイに対して単一オーバレイ処理を行います。UVPI以外のデータストリームでは実行時エラーとなります。

R

フォームオーバレイの焼き付け回数(0～255)を指定します。

C

ページ単位の複写数(0～255)を指定します。

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。複写数で2以上を指定した場合の動作は保証されません。両面印刷時は、複写数には0または1を指定してください。

FCB

適用するFCB名を指定します。

FORMAT-ID

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。

PRT-FORM

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）

ただし、プリンタによっては縮小印刷ができない場合があります。

注意

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、以下の解釈となります。

FORMAT句なし印刷ファイル

印刷情報ファイルの設定値→COBOLまたはlpシステムが定める省略値

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→プリンタの設定値

なお、実際に使用される印刷形式に合わせて、FCBファイルを作成・指定する必要があります。詳細は、“7.1.6 FCB”を参照してください。

SIZE

用紙サイズを指定します。指定可能な値を以下に示します。

－ "A3"

－ "A4"

－ "A5"

－ "B4"

－ "B5"

－ "LTR"（レター）

ただし、プリンタによっては使用できない用紙サイズがあります。

注意

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、以下の解釈となります。

FORMAT句なし印刷ファイル

印刷情報ファイルの設定値→COBOLまたはlpシステムが定める省略値

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ファイルを作成・指定する必要があります。詳細は、“7.1.6 FCB”を参照してください。

HOPPER

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）

－ "P3"（主供給口3）

－ "P4"（主供給口4）

－ "S"（副供給口）

－ "P"（任意の供給口）

ただし、プリンタによっては使用できない供給口があります。

SIDE

片面印刷を行う("F")か、両面印刷を行う("B")かを指示します。

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。

PRT-AREA

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。

BIND

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。指定可能な値を以下に示します。

- "L"(左とじ)
- "R"(右とじ)
- "U"(上とじ)
- "D"(下とじ)

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

とじしろ幅を0～9999(単位:1/1440インチ)で指定します。

OFFSET

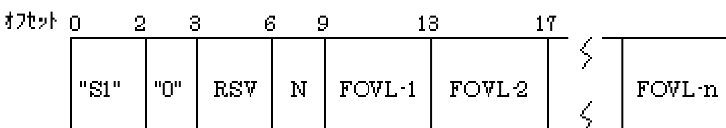
印刷原点位置を0～9999(単位:1/1440インチ)で指定します。

RSV

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.
    02  識別子                  PIC  X(2)  VALUE  "S1".
    02  形式                    PIC  X     VALUE  "0".
    02                          PIC  X(3)  VALUE  SPACE.     →RSV
    02  オーバレイシーケンスの個数
                                PIC  9(3).                   →N
    02  オーバレイ名－１        PIC  X(4).                   →FOVL-1
              :
    02  オーバレイ名－ｎ        PIC  X(4).                   →FOVL-n
```

RSV

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

FOVL-nのフォームオーバレイパターン名の個数を指定します。

FOVL-n

I制御レコードのCで指定した数の複写に対して、この順序でフォームオーバレイの焼き付けが行われます。

ただし、このシステムでは先頭に指定したオーバレイパターン名だけが有効となります。

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。

7.1.16 Unicode(UTF-8)印刷について

ここでは、FORMAT句なし印刷ファイルにおけるUnicode印刷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)におけるUnicode印刷については、MeFtのオンラインマニュアルを参照してください。

Unicode印刷のサポート範囲

FORMAT句なし印刷ファイルでは、COBOLアプリの翻訳時および実行時のロケールにしたがって出力データ(ファイル)のコード系が決定されます。したがって、ロケールがUnicode(UTF-8)である場合には、出力データ(ファイル)のコード系もUnicode、つまりUTF-8となります。

ただし、すべての出力データストリームにおいてUnicode(UTF-8)印刷をサポートしているわけではありません。以下に、各データストリームにおけるUnicode(UTF-8)印刷のサポート状況を記

します。

データストリーム種別	対象装置／連携ソフト	Unicode(UTF-8)印刷サポート状況	備考
UVPI	PrintPartner VSP	サポート済み	注1
PostScriptレベル1	PostScriptプリンタ	未サポート	注2
PostScriptレベル2	PostScriptプリンタ	未サポート	注2
他社プリンタ	ESC/P,ESC/Page,LIPS3	未サポート	注2
電子帳票	ListWorks	サポート済み	

注1: Unicode(UTF-8)印刷を行う場合、Unicode(UTF-8)印刷に対応しているFNPエミュレーションを搭載するVSPプリンタおよびFNPシーケンスに対応したVSPプリンタを制御するソフトウェアが必要です。

注2: OPEN文実行時にエラーとなります。

lpコマンドのオプション

- ディスクファイルに書き出した印刷ファイルを出力する場合、以下のオプションをlpコマンドに指定してください。
  - -y truetype -y UTF-8(Solaris 10の場合)
  - -o -y_truetype -o -y_UTF-8(Solaris 11の場合)
- ファイル管理記述項のASSIGN句の指定がPRINTERの場合、以下のコマンドオプションを環境変数CBR_LP_OPTIONに指定してください。
  - -y truetype -y UTF-8(Solaris 10の場合)
  - -o -y_truetype -o -y_UTF-8(Solaris 11の場合)

Unicode印刷に対応していない機能／環境下での印刷

Unicode(UTF-8)ロケール下でCOBOLアプリを実行する必要があるが、使用している機能(出力データストリーム)がUnicode(UTF-8)印刷をサポートしていない、あるいは使用しているプリンタ装置がUnicode(UTF-8)印刷をサポートしていないため印刷処理が行えないなどの場合、以下の指定を行うことで文字コードをEUCに変換して印刷することができます。ただし、印刷データにUnicode固有文字が含まれる場合、その文字は半角の“_”または半角の“?”に置き換えられますので注意してください。

同一実行単位内のすべてのFORMAT句なし印刷ファイルに共通の設定をする場合

実行環境情報"CBR_PRT_UTF8_CONVERT"を指定します。

ファイル単位に異なる設定をする場合

印刷情報ファイルを作成し"utf8_convert"制御文を指定します。

各指定に関する詳細は、"7.1.4 環境変数の設定"および"7.1.5 印刷情報ファイル"を参照してください。

なお、本指定は、下表の×となる組合せにおいて有効な機能となります。

データストリーム	COBOLのUnicode対応	PrintWalker/LXEのUnicode対応	プリンタのUnicode対応	Unicode印刷の可否
UVPI	○	○	○	◎
	○	○	−	×:上記指定にて回避
	○	−	○/−	×:上記指定にて回避
UVPI以外(注)	−	-------	○/−	×:上記指定にて回避

○:Unicodeサポート済
−:Unicode未サポート
◎:Unicode印刷可能
×:Unicode印刷不可

注: 電子帳票は除きます。電子帳票は、Unicode(UTF-8)ロケール下で使用できます。

7.2 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。

7.2.1 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)

7.2.2 プログラムの記述

ここでは、FORMAT句なし印刷ファイルを使ったプログラムの記述内容について、COBOLの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、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。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7.7 ファイル管理記述項に指定する情報"に示します。

表7.7 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またはレコード順ファイルと同様の媒体情報を指定します。PRINTERを指定すると、システムの標準ライタに出力されます。(注1)
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2)

注1: その他の指定方法については"6.2.1 レコード順ファイルの定義"を参照してください。
注2: 設定される値については、"付録B 入出力状態一覧"を参照してください。

データ部(DATA DIVISION)

データ部には、レコードの定義およびファイル管理記述項に指定したデータ名の定義を記述します。レコードの定義は、ファイル記述項とレコード記述項で記述します。ファイル記述項を記述するために必要な情報を"表7.8 ファイル記述項に指定する情報"に示します。

表7.8 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

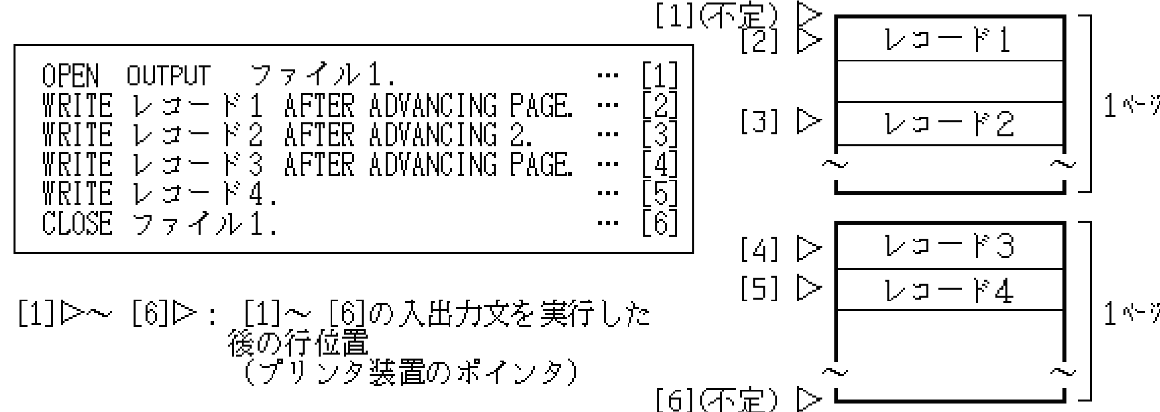
OPEN文およびCLOSE文

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。このとき、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あと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指定による印字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に定義しないで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、"6.6 入出力エラー処理"を参照してください。

7.2.3 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。

7.2.4 プログラムの実行

印刷ファイルを使ったプログラムを実行するときの操作は、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句の記述内容とプログラムの実行前および実行後に必要な操作を"表7.9 プログラム実行時に必要な操作"に示します。

表7.9 プログラム実行時に必要な操作

プログラム実行時の操作		ASSIGN句の記述		
		PRINTER	ファイル識別名 PRINTER-n	ファイル識別名定数またはデータ名
実行前	ファイル識別名を環境変数とした出力先ファイル名の設定	―	必須	―
	環境変数CBR_LP_OPTIONの設定	任意	―	―

プログラム実行時の操作		ASSIGN句の記述		
		PRINTER	ファイル識別名PRINTER-n	ファイル識別名定数またはデータ名
	環境変数CBR_PRT_INFの設定	任意	任意	任意
実行後	lpコマンドの起動	ー(注)	必須	必須
出力先		システムの標準ライタ	ファイル	ファイル

注: ASSIGN句の指定がPRINTERの場合、CLOSE文の実行でlpコマンドが自動的に起動されます。このときlpコマンドに指定されるオプションについては、"7.1.4 環境変数の設定"の"CBR_LP_OPTION"を参照してください。各環境変数の設定については"7.1.4 環境変数の設定"を参照してください。

注意

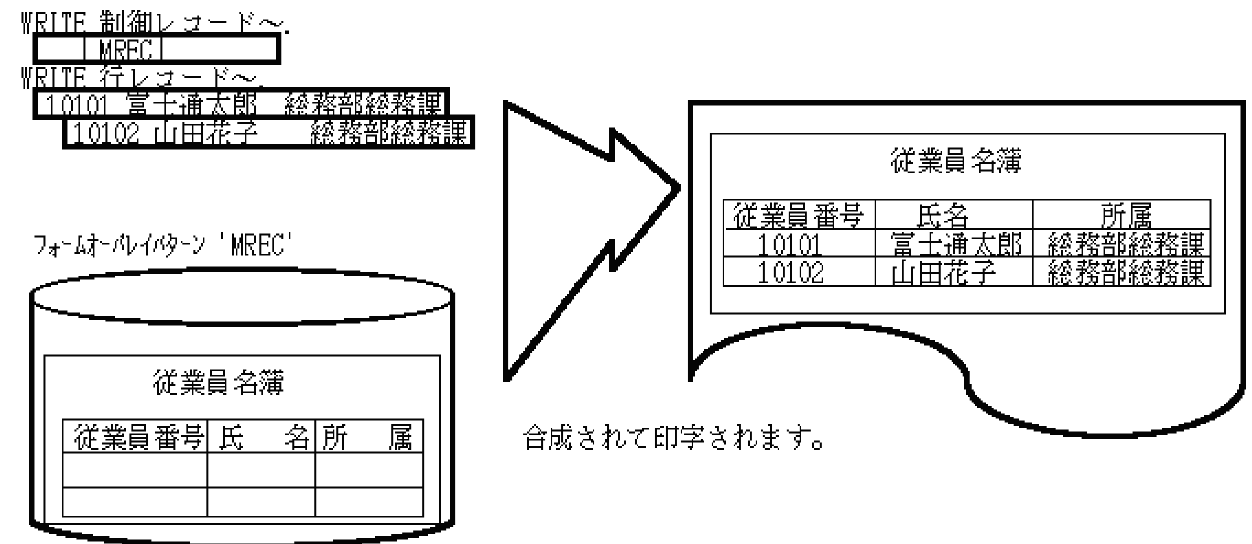
- データストリームがUVPIの場合、lpコマンド起動時に以下のオプションを指定してください。
  - -Tcobol(Solaris 10の場合)
  - -o -T_cobol(Solaris 11の場合)
- ASSIGN句の指定がPRINTERの場合、UVPIのデータストリームをVSPプリンタへFNPエミュレーションで出力するには、環境変数CBR_LP_OPTIONに以下のオプションを指定してください。
  - -y truetype(Solaris 10の場合)
  - -o -y_truetype(Solaris 11場合)
- lpコマンドを使ってディスクファイルに書き出したUVPIの印刷ファイルをVSPプリンタへFNPエミュレーションで出力する場合、lpコマンドに以下のオプションを指定してください。
  - -y truetype(Solaris 10の場合)
  - -o -y_truetype(Solaris 11の場合)
- データストリームがUVPI以外の場合、特に必要なオプションはありません。
- データストリームについては、"7.1.2 印刷装置"を参照してください。

7.3 フォームオーバレイおよびFCBを使う方法

ここでは、制御レコードを使用して、フォームオーバレイパターンとの合成印刷を行う方法について説明します。

7.3.1 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ことにより、帳票の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を使うときには、制御レコードを使います。制御レコードは、通常のデータを出力するときと同様に、WRITE文を使って出力します。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、形態、間隔、書体および方向は、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、"7.1.3 印字文字"およびFORMのマニュアルまたはヘルプを参照してください。



7.3.2 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7.7 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に記述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7.8 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、“7.1.3 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”7.1.15 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

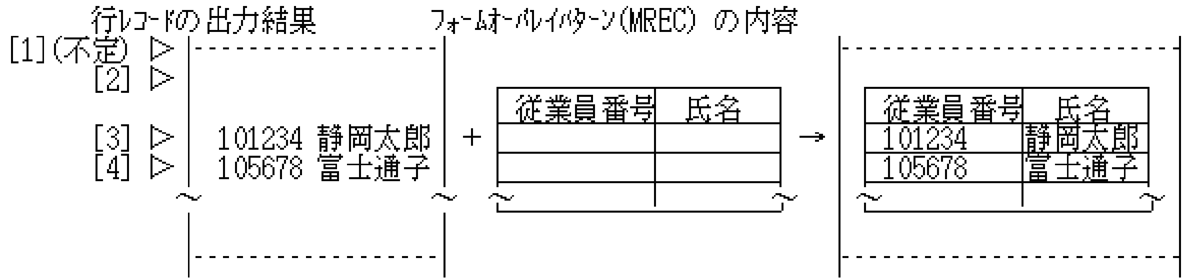
WRITE文は、制御レコードおよび行レコードを出力するときに実行します。行レコードの出力は、行単位のデータを出力するときのWRITE文の使い方と同じです。“7.2.2 プログラムの記述”を参照してください。

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、そのあとに出力するページの形式が制御レコードの内容のとおり設定されます。ただし、制御レコードを出力すると、現在のページには行レコードを出力できなくなるので、制御レコード出力直後の行レコードの出力には、AFTER ADVANCING PAGEを指定する必要があります。

```
  :
FILE SECTION.
FD  ファイル１.
  :
01  制御レコード.
  :
   02  FOVL        PIC X(4).
  :
    MOVE  "MREC"    TO FOVL.
```

```
WRITE  制御レコード AFTER ADVANCING  呼び名.            … [1]
MOVE  SPACE TO   行レコード.
WRITE  行レコード   AFTER ADVANCING  PAGE.             … [2]
MOVE  101234    TO   従業員番号.
MOVE  NC"静岡太郎" TO  氏名.
WRITE  行レコード   AFTER ADVANCING  2.                … [3]
MOVE  105678    TO   従業員番号.
MOVE  NC"富士通子" TO  氏名.
WRITE  行レコード   AFTER ADVANCING  1.                … [4]
   :
```



▷：[2] ～[4] の入出力文を実行した後の行位置

[1] で出力した制御レコードの指示により、ﾌｫｰﾑｵｰﾊﾞﾚｲﾊﾟﾀｰﾝ(MREC)と合成して印字されます。

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に定義しないで、Bに定義します。CHARACTER TYPE句が両方に定義された場合、Bの指定が有効となります。

入出力エラー処理

入出力エラーの検出方法および入出力エラーが発生したときの実行結果については、"6.6 入出力エラー処理"を参照してください。

7.3.3 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。

7.3.4 プログラムの実行

印刷ファイルを使ったプログラムを実行するときの操作は、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句の記述内容とプログラムの実行前および実行後に必要な操作を"表7.10 プログラムの実行時に必要な操作"に示します。

表7.10 プログラムの実行時に必要な操作

プログラム実行時の操作		ASSIGN句の記述		
		PRINTER	ファイル識別名 PRINTER-n	ファイル識別名定数 またはデータ名
実行前	ファイル識別名を環境変数とした出力先ファイル名の設定	－	必須	－
	環境変数CBR_LP_OPTIONの設定	任意	－	－
	環境変数CBR_PRT_INFの設定	任意	任意	任意
	環境変数CBR_FCB_NAMEの設定	任意	任意	任意
	環境変数FCBDIRの設定	任意	任意	任意

プログラム実行時の操作		ASSIGN句の記述		
		PRINTER	ファイル識別名 PRINTER-n	ファイル識別名定数 またはデータ名
	環境変数FOVLDIRの設定	任意	任意	任意
実行後	lpコマンドの起動	−	必須	必須
出力先		システムの標準ライタ	ファイル	ファイル

参照

各環境変数の設定については“7.1.4 環境変数の設定”を参照してください。

注意

データストリームがUVPIの場合

- 印字結果を直接印刷装置に出力する場合、プログラム実行時にフォームオーバレイパターンが格納されているディレクトリのパス名を環境変数FOVLDIRに設定してください。また、FCBを使用している場合、そのFCBが格納されているディレクトリのパス名を環境変数FCBDIRに設定してください。

- lpコマンドを使って、ディスクファイルに書き出した印刷ファイルを出力する場合、以下のオプションを指定してください。

  Solaris 10の場合

  -Tオプション： 文字列“cobol”を指定します。
  -yオプション： 使用するフォームオーバレイパターンおよびFCBが格納されているディレクトリを指定します。

  例

  ```
  $ lp -dprinter -Tcobol -y op=/home/usr1,fp=/home/usr1 report
  ```

  ```
  -dprinter      : 印刷先としてprinter を指定します。
  -Tcobol        : 文書形式としてcobol を指定します。
  op=/home/usr1  : フォームオーバレイパターンが/home/usr1/kol5 に格納されていることを指定します。
  fp=/home/usr1  : FCB が/home/usr1/fcbe に格納されていることを指定します。
  report         : 印刷するファイル
  ```

  Solaris 11の場合

  -o -T_オプション: 文字列“cobol”を指定します。
  -o -y_オプション: 使用するフォームオーバレイパターンおよびFCBが格納されているディレクトリを指定します。

  例

  ```
  $ lp -dprinter -o -T_cobol -o "-y_op=/home/usr1,-y_fp=/home/usr1" report
  ```

  ```
  -dprinter        : 印刷先としてprinter を指定します。
  -o -T_cobol      : 文書形式としてcobol を指定します。
  -y_op=/home/usr1 : フォームオーバレイパターンが/home/usr1/kol5 に格納されていることを指定します。
  -y_fp=/home/usr1 : FCB が/home/usr1/fcbe に格納されていることを指定します。
  report           : 印刷するファイル
  ```

注意

データストリームがUVPI以外の場合

- プログラム実行時にフォームオーバレイパターンが格納されているディレクトリのパス名を環境変数FOVLDIRに設定しておく必要があります。また、FCBを使用している場合、そのFCBが格納されているディレクトリのパス名を環境変数FCBDIRに設定しておく必要があります。
- ディスクファイルに書き出した印刷ファイルをlpコマンドを使って出力する場合、特に必要なオプションはありません。
- フォームオーバレイパターン出力時のサポート範囲は、MeFtのオンラインマニュアルを参照してください。
- データストリームがPostScriptレベル2の場合、KOL6形式のフォームオーバレイパターンに定義したオーバレイ文字は、以下の書体で印刷されます。

  日本語文字: 明朝
  英数字　: ゴシック

データストリームについては、“7.1.2 印刷装置”を参照してください。

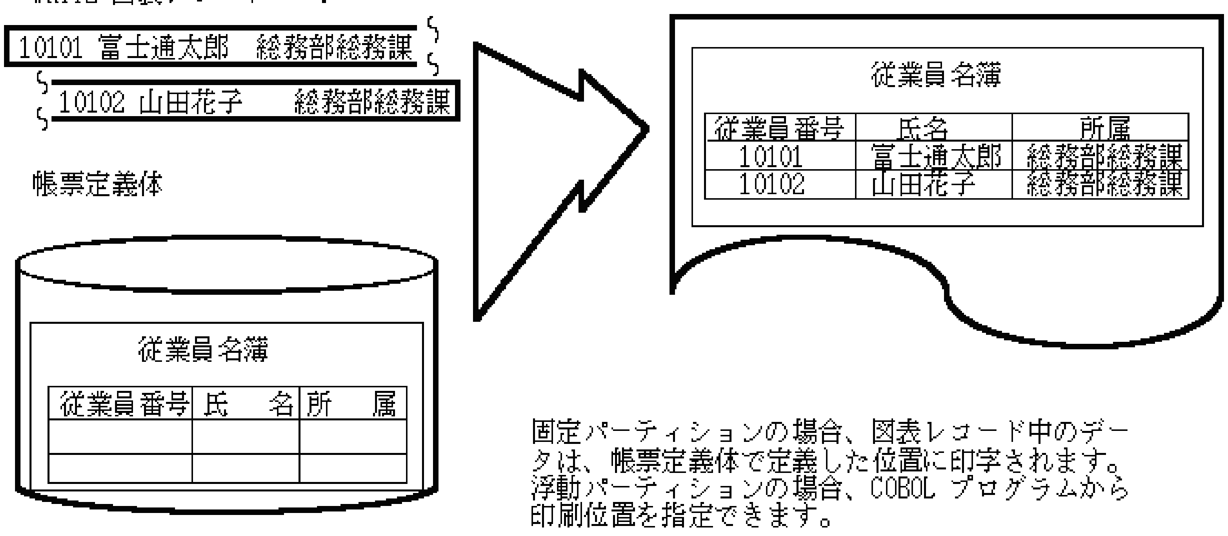
7.4 帳票定義体を使う印刷ファイルの使い方

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。

なお、FORMAT句付き印刷ファイルを使った例題プログラムをサンプルとして提供していますので、参考にしてください。

7.4.1 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(項目群)を定義します。図表レコードの定義文は、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。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。印字する文字の大きさ、形態、間隔、書体および方向は、COBOLプログラムおよび帳票定義体で指示することができます。指定できる内容については、“7.1.3 印字文字”およびFORMのマニュアルまたはヘルプを参照してください。



注意

- 自由形式の帳票定義体は、FORMAT句付き印刷ファイルでは使用できません。表示ファイルの帳票印刷機能を使用してください。

- 帳票定義体を作成する場合、COBOLプログラムで設定/参照される名前は、以下の注意が必要です。
  - 帳票定義体名は8文字以内の半角英数字で指定します。
  - 項目群名およびパーティション名は6文字以内の半角英数字で指定します。
  - 項目名はCOBOLの利用者語の記述規則に従って指定します。

7.4.1.1 帳票のパーティション

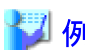
パーティションには、固定パーティション、浮動パーティションと呼ばれる、2種の属性があります。

固定パーティション

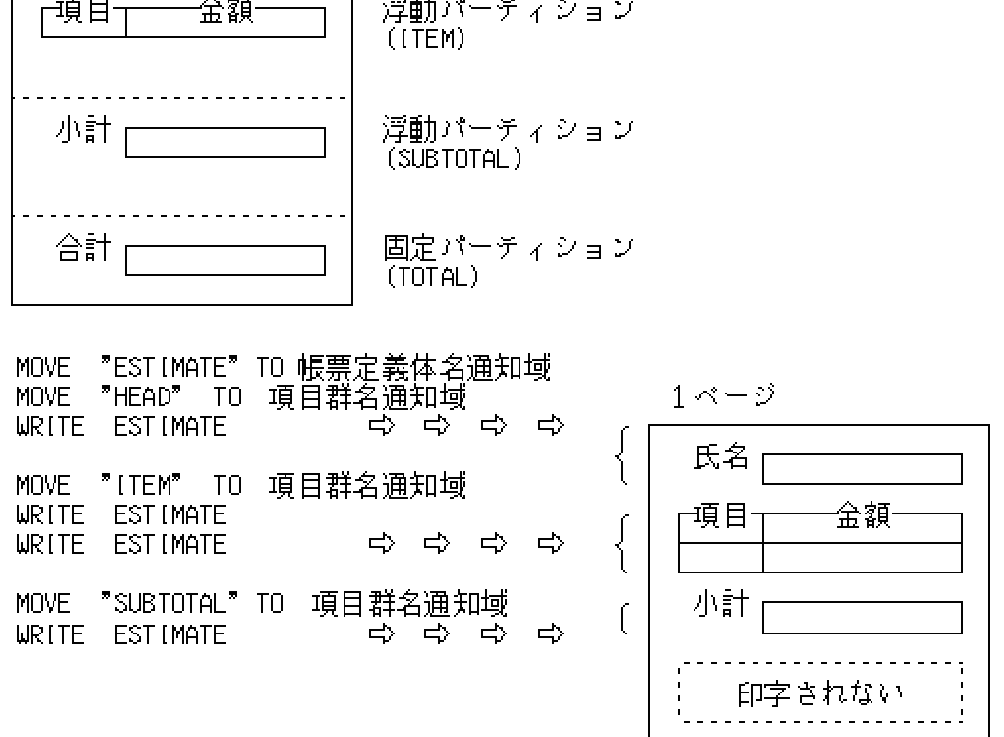
固定パーティションとはページ内での印刷位置が固定のパーティションで、帳票定義体で指定された位置に印刷されます。

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

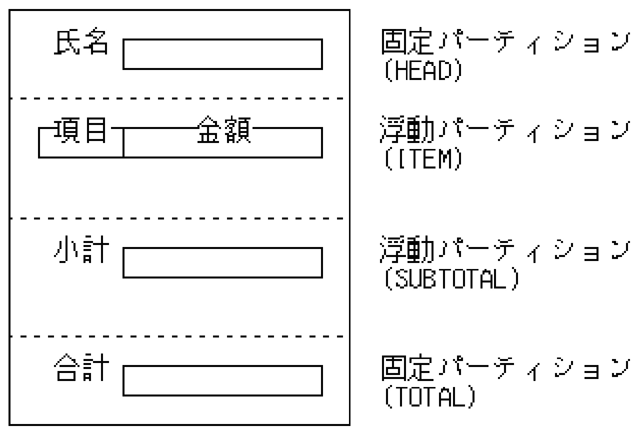
図表レコードを使うことで、帳票定義体に定義したパーティションを印刷することができます。また、そのページの任意の位置に、通常の行レコードを出力することもできます。

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。

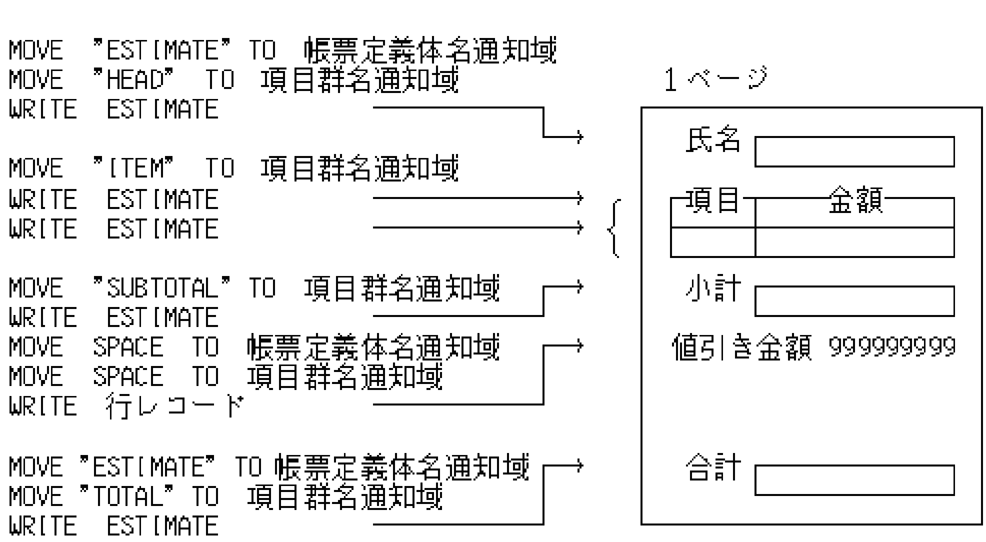
例

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。

帳票定義体：見積書〔ESTIMATE.pmd〕




```
01 行レコード.
  02 値引き金額 PIC 9(9).
```



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。

7.4.1.2 帳票の電子化

プリンタ情報ファイルに指定を追加することにより、MeFt経由で出力する帳票を電子化することができます。帳票を電子化する方法には、ListWorks連携、e-DocGenerator連携およびList Manager連携があります。それぞれの詳細については、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。

図7.1 プリンタ出力の場合

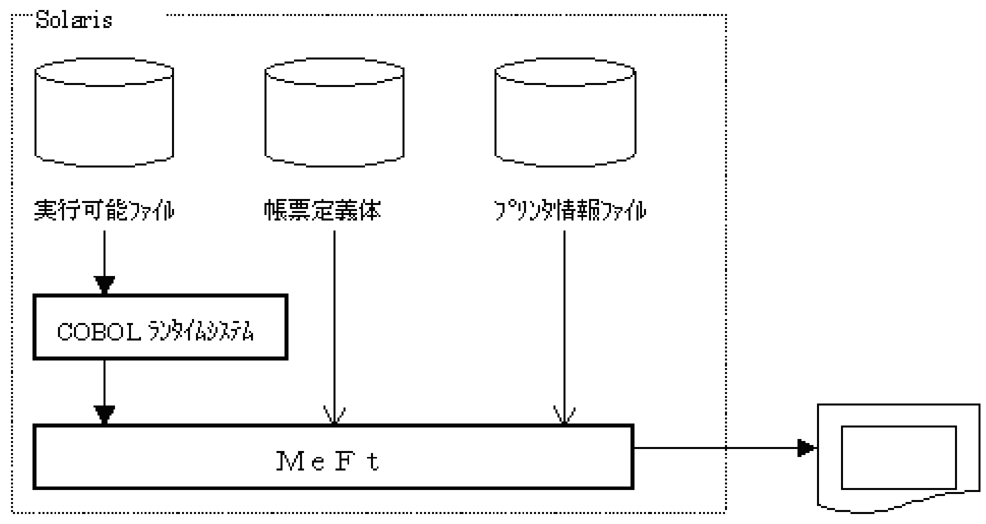
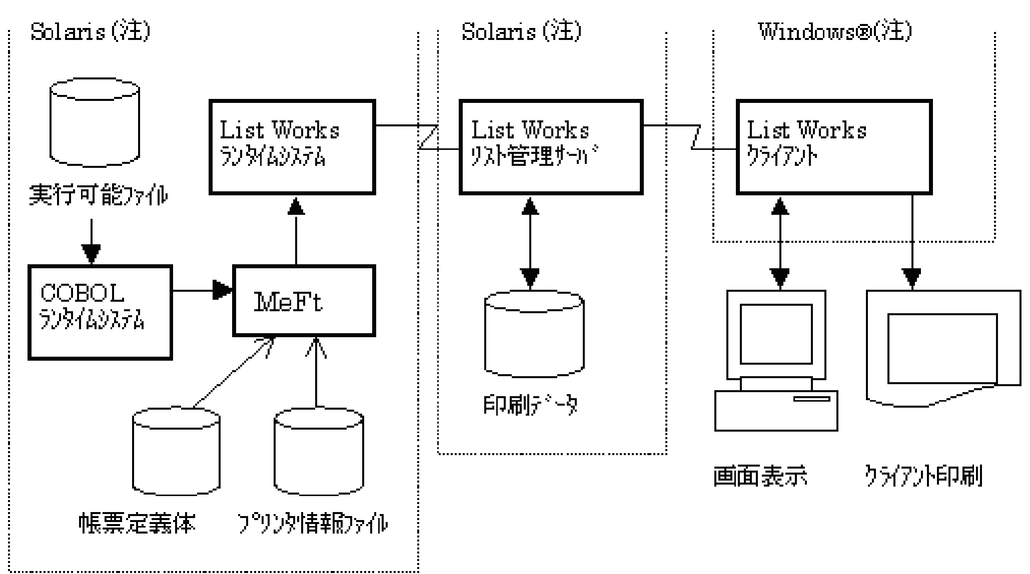


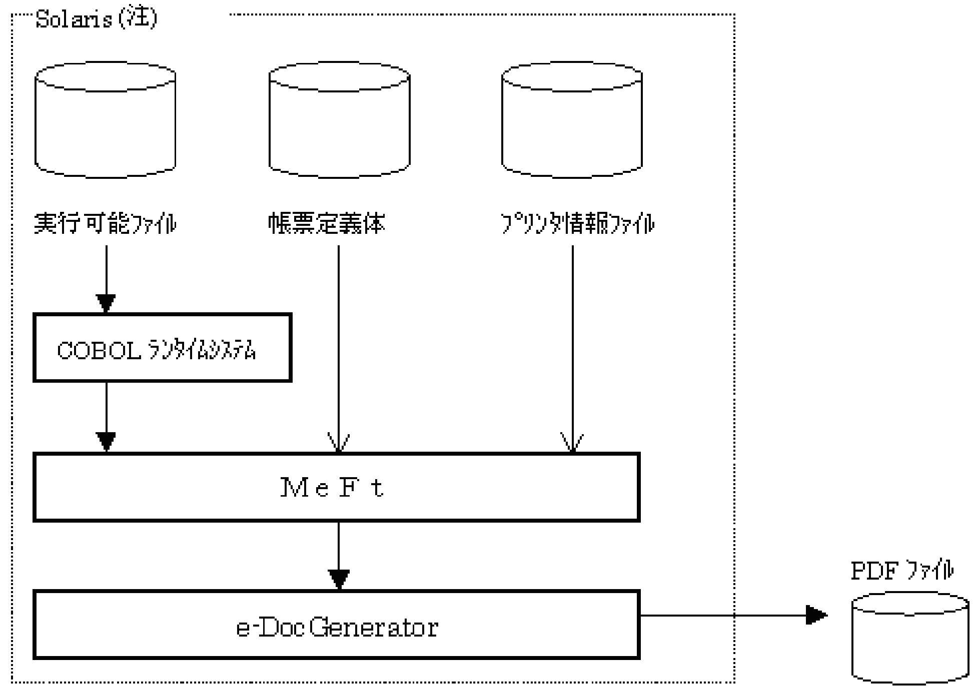
図7.2 ListWorks連携の場合



参考

ListWorks連携では、MeFtの出力をListWorksで扱うことのできる電子帳票の形式にします。

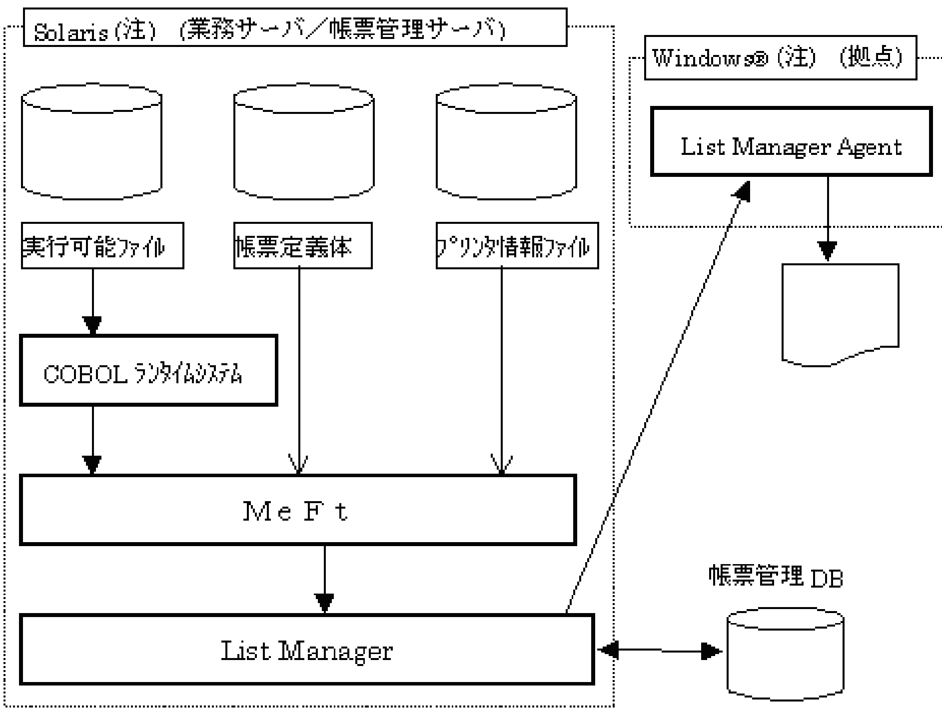
図7.3 e-DocGenerator連携の場合



参考

e-DocGenerator連携では、MeFtの出力をPDF(Portable Document Format)ファイルにします。

図7.4 List Manager連携の場合



参考

List Manager連携では、MeFtの出力を帳票管理サーバで一元管理し、接続拠点への帳票配信を行ないます。

注:ListWorks、e-DocGeneratorおよびList Manager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インストールガイドを確認してください。

7.4.2 プログラムの記述

ここでは、帳票定義体を使った印刷ファイルの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  [FORM 制御レコードデータ名].
     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  ～].
     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。

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7.11 ファイル管理記述項に指定する情報"に示します。

表7.11 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)

注: 設定される値については、"付録B 入出力状態一覧"を参照してください。詳細情報を取得する場合は、4桁の英数字項目を指定します。

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り、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

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7.12 ファイル記述項に指定する情報"に示します。

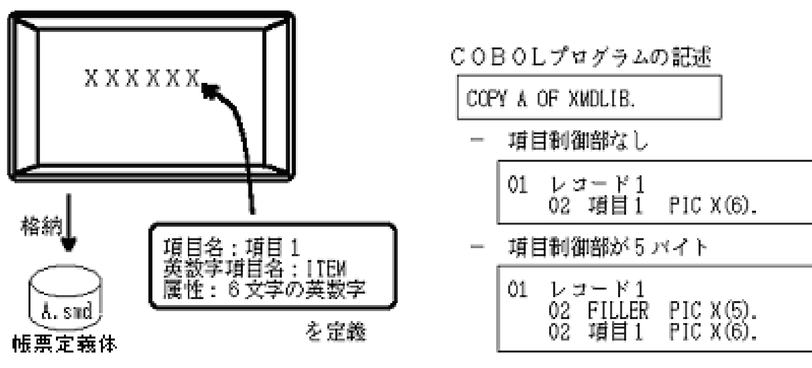
表7.12 ファイル記述項に指定する情報

	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、"7.3.2 プログラムの記述"を参照してください。

図表レコードには、帳票定義体で定義したパーティション(項目群)を定義します。この定義文は翻訳時に、登録集名に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。展開されるレコードの内容を以下に示します。



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

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おく必要があります。

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。

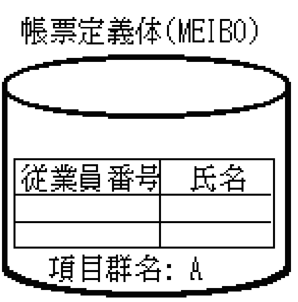
図表レコードの出力時に、そのページの形式を決定した帳票定義体名から別の帳票定義体名に変更して出力する場合は、改ページされます。

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。

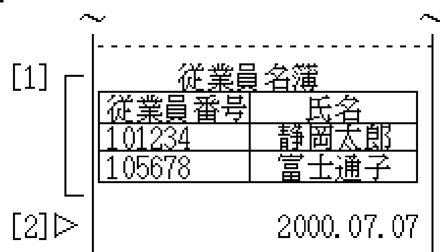
WRITE文のADVANCING指定については、“7.2.2 プログラムの記述”を参照してください。

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。

```
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 "2000.07.07" 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】



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、"6.6 入出力エラー処理"を参照してください。

7.4.3 プログラムの翻訳・リンク

印刷ファイルで帳票定義体を使うプログラムの翻訳・リンク方法を以下に示します。

cobolコマンドを使って翻訳とリンクを行う場合

```
$ cobol -M -m パス名 [その他の翻訳・リンクオプション] ファイル名
```

-Mオプションは、主プログラムの場合に指定します。

-mオプションに帳票定義体を格納したディレクトリのパス名を指定します。

注意

画面定義体を使用しないプログラムの翻訳・リンクでは、-pmオプションを指定しないでください。

ldコマンドを使ってリンクを行う場合

ldコマンドを使ってリンクを行う方法については、"付録L ldコマンド"を参照してください。

7.4.4 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、"7.5.6 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必要な環境設定を以下に示します。

- ASSIGN句にファイル識別名を指定した場合、ファイル識別名を環境変数名として、プリンタ情報ファイルのパス名を設定します。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に設定したディレクトリ
  2. カレントディレクトリ

- 印刷ファイルで帳票定義体と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格納ディレクトリのパス名をプリンタ情報ファイルに設定します。(環境変数FOVLDIRの設定は無効です。)
- 帳票定義体を使う印刷ファイルで、FCBのデフォルト値を変更する場合、環境変数CBR_FCB_NAMEに省略時に使用するFCB名を設定してください。
- 環境変数LD_LIBRARY_PATHに、MeFtの格納ディレクトリを設定します。
- その他の実行に必要な環境設定については、MeFtのオンラインマニュアルを参照してください。

例

印刷ファイルで帳票定義体を使う場合のプログラムの実行

- COBOLプログラムのASSIGN句の記述

```
SELECT  ファイル名  ASSIGN TO PRTFILE ……
```

- 入力コマンド

```
$ LD_LIBRARY_PATH=/home/usr1:$LD_LIBRARY_PATH                    …  [1]
$ export  LD_LIBRARY_PATH                                        …  [2]
$ cobol -M -o PROG1 -m/home/usr1/meddir PROG1.cobol              …  [3]
$ PRTFILE=/home/xx/MEFPRC ; export PRTFILE                       …  [4]
$ PROG1                                                          …  [5]
```

```
/home/usr1         :  MeFtの格納されているディレクトリ
/home/usr1/meddir  :  帳票定義体ファイルの格納ディレクトリ
/home/xx/MEFPRC    :  プリンタ情報ファイル
PROG1              :  実行可能プログラム
```

[1],[2] MeFtの格納されているディレクトリを環境変数LD_LIBRARY_PATHに設定します。

[3] cobolコマンドを使って翻訳・リンクを行います。

[4] 環境変数PRTFILEにMeFtの使用するプリンタ情報ファイルを設定します。

[5] 実行可能プログラムを実行します。実行可能プログラムの実行が終了すると、印刷装置に帳票が出力されます。

7.5 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。

7.5.1 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。帳票定義体で定義されている出力データの属性は、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。

図7.5 ローカル環境で使用する場合

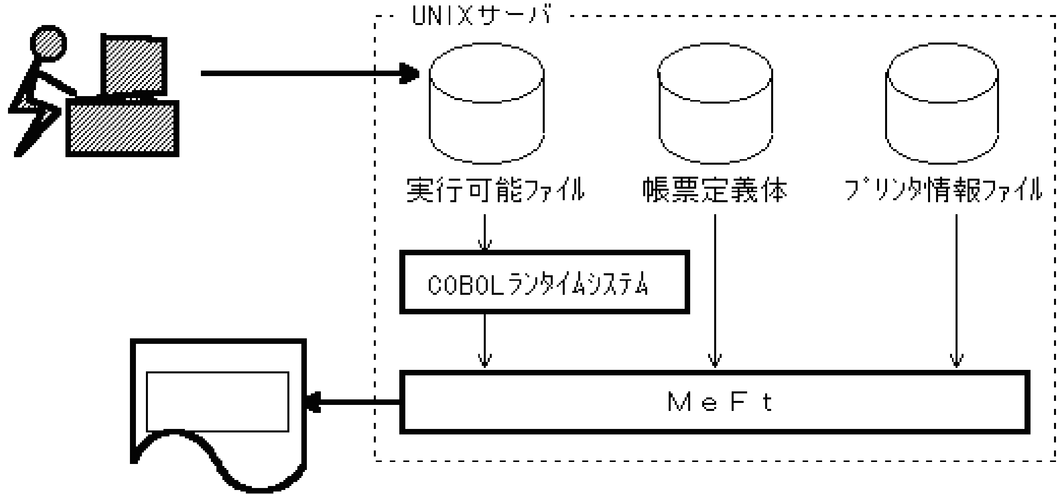
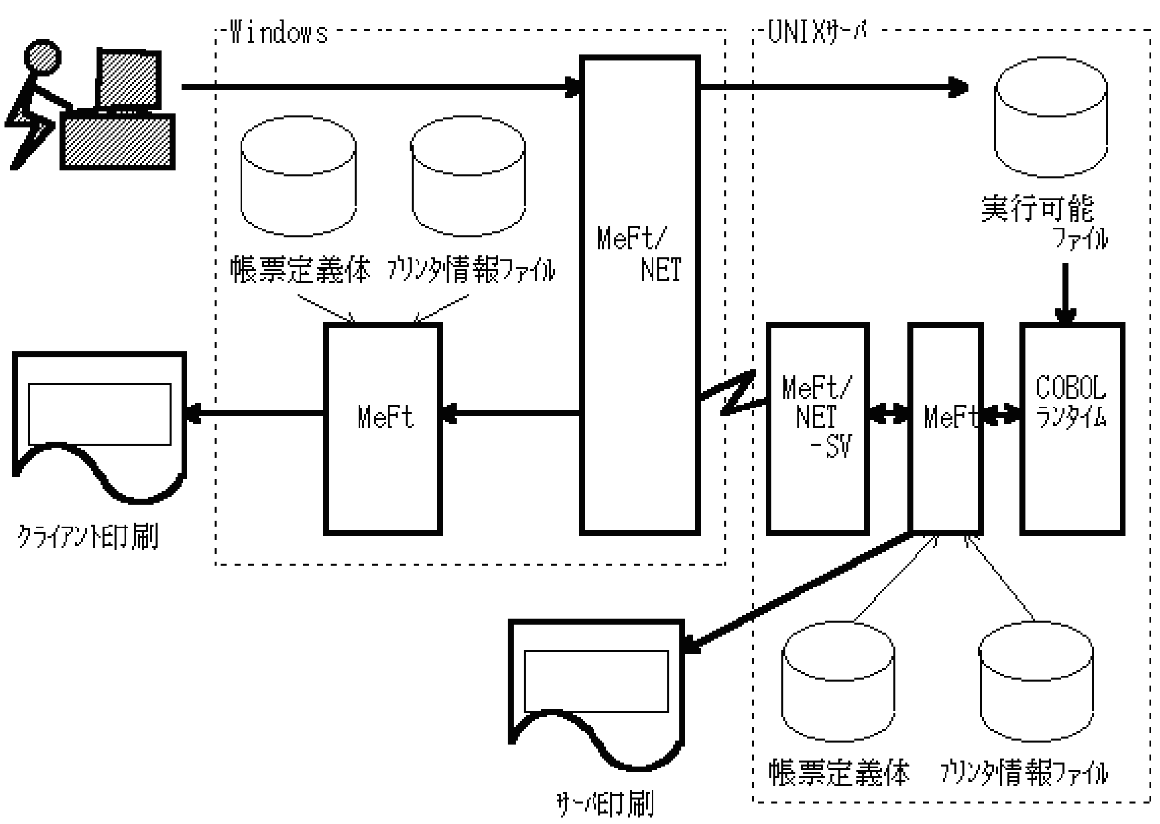
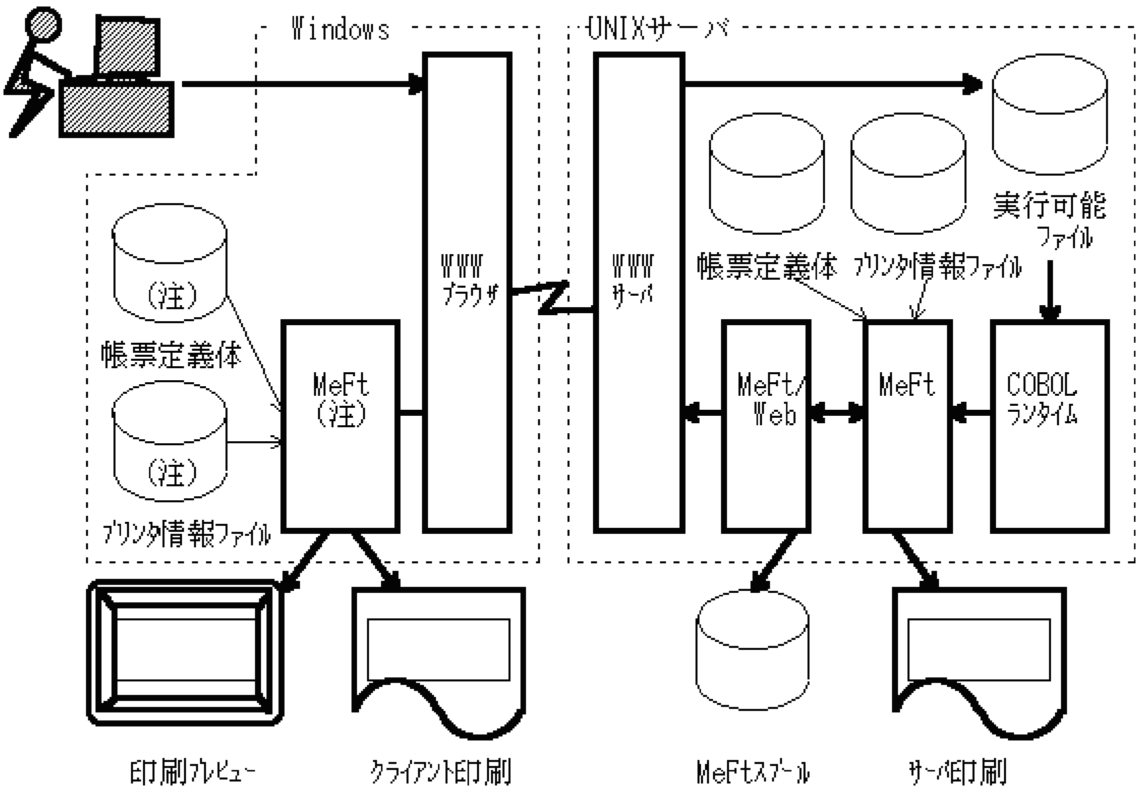


図7.6 リモート環境で使用する場合（MeFt/NET連携）



アプリケーション起動時に、クライアント印刷またはサーバ印刷を選択することができます。

図7.7 リモート環境で使用する場合（MeFt/Web連携）



注: サーバから自動的にダウンロードされます。

アプリケーション起動時に、クライアント印刷、サーバ印刷、印刷プレビューまたはスプールのどれかを選択することができます。

MeFt、MeFt/NET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インストールガイド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。詳細については、"7.4.1.2 帳票の電子化"を参照してください。

7.5.2 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. 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。

7.5.3 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のマニュアルまたは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7.13 帳票定義体を作成するときに設定する情報"に示します。

表7.13 帳票定義体を作成するときに設定する情報

	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)”、“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”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。

7.5.4 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通知域  入出力状態２通知域].
DATA DIVISION.
 FILE SECTION.
 FD  ファイル名.
     COPY  帳票定義体名 OF XMDLIB.
(01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通知域    PIC X(2).]
[01  入出力状態２通知域    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  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7.14 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7.14 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表示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)
	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、帳票出力を行うとき、入出力処理の処理種別を設定します。設定できる値についてはMeFtのオンラインマニュアルを参照してください。
	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。設定できる値についてはMeFtのオンラインマニュアルを参照してください。

注: 設定される値については、"付録B 入出力状態一覧"を参照してください。詳細情報を取得する場合は、4桁の英数字項目を指定します。

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、登録集名に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記述文の内容については、"7.4.2 プログラムの記述"を参照してください。

注意

表示ファイルに対してEXTERNAL句を指定する場合には、“9.2.5.2 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。入出力は次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. READ文： IDマークまたはバーコードの入力
4. 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、“7.1.11 特殊レジスタ”を参照してください。

READ文

READ文では、IDマークおよびバーコードの入力を行うことができます。ただし、入力に使用する帳票定義体の名前は、READ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。また、処理が可能なプリンタ装置を接続する必要があります。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。

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、“6.6 入出力エラー処理”を参照してください。

7.5.5 プログラムの翻訳・リンク

表示ファイルで帳票印刷を行うプログラムの翻訳・リンク方法を以下に示します。

cobolコマンドを使って翻訳とリンクを行う場合

```
$ cobol -M -m パス名 ［その他の翻訳・リンクオプション］ ファイル名
```

-Mオプションは、主プログラムの場合に指定します。

-mオプションに帳票定義体を格納したディレクトリのパス名を指定します。

プログラム中で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。

注意

画面定義体を使用しないプログラムの翻訳・リンクでは、-pmオプションを指定しないでください。

ldコマンドを使ってリンクを行う場合

ldコマンドを使ってリンクを行う方法については、“付録L ldコマンド”を参照してください。

7.5.6 プリンタ情報ファイルの作成

ここではMeFtを使って帳票を印刷するときに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、接続製品に応じて以下のマニュアルを参照してください。

- 接続製品がMeFt(ローカル環境での印刷)の場合：MeFtのオンラインマニュアル
- 接続製品がMeFt/NET(リモート環境での印刷)の場合：Windows版 MeFtのオンラインマニュアル
- 接続製品がMeFt/Webの場合：MeFtのオンラインマニュアル、MeFt/Web説明書

プリンタ情報ファイルに設定する代表的な情報を"表7.15 プリンタ情報ファイルを作成するときに設定する情報"に示します。

表7.15 プリンタ情報ファイルを作成するときに設定する情報

情報の種類	指定する内容および用途
PRTNAME	出力するプリンタ装置に対するプリンタ名またはクラス名を指定します。
PRTDEV	出力するプリンタ装置のプリンタ機種名を指定します。
MEDDIR	帳票定義体を格納したディレクトリの名前を設定します。
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の定めた省略値とみなされます。

注意

接続製品がMeFt/Webの場合、プリンタ情報ファイルは以下のコード系を使用して作成します。詳細は、"MeFt/Web説明書"を参照してください。

- サーバ印刷の場合はサーバのコード系を使用します。
- クライアント側で印刷する場合はクライアント環境のコード系を使用します。

7.5.7 プログラムの実行

表示ファイル機能を使った帳票印刷を行うプログラムを実行するときには、以下の環境設定が必要です。

MeFtを使用する場合

- ファイル識別名を環境変数として、プリンタ情報ファイルのパス名を設定します。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に設定したディレクトリ
  2. カレントディレクトリ
- 環境変数LD_LIBRARY_PATHに、MeFtの格納ディレクトリを設定します。
- その他の実行に必要な環境設定については、MeFtのオンラインマニュアルを参照してください。

MeFt/NET-SVを使用する場合

- ファイル識別名を環境変数として、ディレクトリ名を除いたプリンタ情報ファイル名を設定します。このとき、プリンタ情報ファイルは、Windows(クライアント)側に用意しておく必要があります。
- 環境変数LD_LIBRARY_PATHに、MeFt/NET-SVの格納ディレクトリを設定します。
- 接続製品としてMeFt/NET-SVを使用することを示す環境変数を指定する必要があります。指定方法には、以下の二つの方法があります。
  1. ファイル識別名を環境変数名として、接続製品名にMEFTNETを設定する。
  2. 環境変数CBR_PSFILE_PRTにMEFTNETを設定する。
     設定方法の詳細については、"4.1.3 環境変数による接続製品の指定"を参照してください。
- その他の実行に必要な環境設定については、"MeFt/NET-SV説明書"を参照してください。

MeFt/Webを使用する場合

- ファイル識別名を環境変数として、ディレクトリ名を除いたプリンタ情報ファイル名を設定します。(注)
- 環境変数MEFTDIRに、サーバ印刷時に使用するプリンタ情報ファイルの格納ディレクトリを設定します。
- 環境変数MEFTWEBDIRに、クライアント印刷時に使用するプリンタ情報ファイルの格納ディレクトリをURLまたはサーバのローカルパスで設定します。
- 環境変数LD_LIBRARY_PATHに、MeFt/WebおよびMeFtの格納ディレクトリを設定します。
- その他の実行に必要な環境変数については、MeFt/Web説明書を参照してください。

注: サーバ印刷用のプリンタ情報ファイルとクライアント印刷用のプリンタ情報ファイルは同一名で用意する必要があります。

7.6 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、"7.4.1.2 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。

7.6.1 電子帳票出力機能の概要

COBOLでは、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(あて先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す。これに加え、ListWorksと連携することにより、これらの帳票を電子化することが可能となります。帳票を電子化するためには、印刷情報ファイルまたはプリンタ情報ファイルに簡単な環境定義を追加します。従来のアプリケーションを変更する必要はありません。

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して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。

電子帳票およびListWorksに関する詳細は、以下を参照してください。

- ListWorksのオンラインマニュアル
- ListWorksのヘルプ

7.6.2 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。

7.6.2.1 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ます。そして、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

実行環境単位内で共通の印刷情報ファイルを使用する場合には、環境変数CBR_PRT_INFに印刷情報ファイルのパス名を指定します。

ASSIGN句の記述がファイル識別名、PRINTER-n、データ名およびファイル識別名定数の場合には、ファイルごとに印刷情報ファイルを指定することができます。以下にASSIGN句の記述別に指定方法を説明します。

また、電子帳票出力を行う場合には、出力先としてListWorksの電子保存装置名を指定します。

ASSIGN句にファイル識別名を指定した場合

ListWorksの電子保存装置名の後ろに、“,,INF(印刷情報ファイルのパス名)”を指定します。詳細は、“7.1.5 印刷情報ファイル”を参照してください。

例

[例1]

ファイル識別名“PRTFILE”にプリンタ名(lwprint)および印刷情報ファイルのパス名(/home/usr1/printinf.txt)を割り当てる場合

```
$ PRTFILE=” lwprint,,INF(/home/usr1/printinf.txt)” ; export PRTFILE
```

[例2]

ファイル識別名“PRTFILE”に対するプリンタの割当てを省略し、印刷情報ファイルのパス名(/home/usr1/printinf.txt)だけを割り当てる場合

```
$ PRTFILE=” ,,INF(/home/usr1/printinf.txt)” ; export PRTFILE
```

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制御文でプリンタの割当てを行う必要があります。

[例3]

ファイル識別名“PRTFILE”に対するプリンタの割当てを省略し、印刷情報ファイルのパス名(/home/usr1/printinf.txt)を環境変数CBR_PRT_INFに割り当てる場合

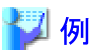
```
$ PRTFILE=,, ; export PRTFILE
$ CBR_PRT_INF=/home/usr1/printinf.txt ; export CBR_PRT_INF
```

この場合、“PRTFILE=,,”の指定は省略することができます。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制御文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。ただし、PRINTER-nはBourne shellでは使用できません。

以下にC shellでの実行例を示します。



[例1]

“PRINTER-1”にプリンタ名(lwprint)および印刷情報ファイルのパス名
(/home/usr1/printinf.txt)を割り当てる場合

```
% setenv PRINTER-1 ” lwprint,,INF(/home/usr1/printinf.txt)”
```

[例2]

“PRINTER-2”に対するプリンタの割当てを省略し、印刷情報ファイルのパス名
(/home/usr1/printinf.txt)だけを割り当てる場合

```
% setenv PRINTER-2 ” ,,INF(/home/usr1/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制御文でプリンタの割当てを行う必要があります。

[例3]

“PRINTER-3”に対するプリンタの割当てを省略し、印刷情報ファイルのパス名
(/home/usr1/printinf.txt)を環境変数CBR_PRT_INFに割り当てる場合

```
% setenv PRINTER-3 ,,
% CBR_PRT_INF /home/usr1/printinf.txt
```

この場合、“setenv PRINTER-3 ,,”の指定は省略することができます。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制御文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、ListWorksの電子保存装置名の後ろに、“,,INF(印刷情報ファイルのパス名)”を指定したデータを設定します。

[例1]

データ名“PRTDATA”にプリンタ名(lwprint)および印刷情報ファイルのパス名(/home/usr1/printinf.txt)を割り当てる場合

```
     :
    SELECT 印刷ファイル ASSIGN TO PRTDATA.
     :
 01  PRTDATA  PIC X(128).
     :
    MOVE  “lwprint,,INF(/home/usr1/printinf.txt)”  TO PRTDATA.
     :
```

[例2]

データ名“PRTDATA”に対するプリンタの割当てを省略し、印刷情報ファイルのパス名(/home/usr1/printinf.txt)だけを割り当てる場合

```
     :
    SELECT 印刷ファイル ASSIGN TO PRTDATA.
     :
 01  PRTDATA  PIC X(128).
     :
    MOVE  “,,INF(/home/usr1/printinf.txt)”  TO PRTDATA.
     :
```

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制御文でプリンタの割当てを行う必要があります。

[例3]

データ名“PRTDATA”に対するプリンタの割当てを省略し、印刷情報ファイルのパス名(/home/usr1/printinf.txt)を環境変数CBR_PRT_INFに割り当てる場合

```
    :
   SELECT 印刷ファイル ASSIGN TO PRTDATA.
    :
01  PRTDATA  PIC X(128).
    :
   MOVE “,,” TO PRTDATA.
    :
```

- 環境変数の設定

```
$ CBR_PRT_INF=/home/usr1/printinf.txt ; export CBR_PRT_INF
```

この場合、PRTDATAの内容はSPACEでも構いません。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制御文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、ListWorksの電子保存装置名の後ろに、“,,INF(印刷情報ファイルのパス名)”を記述します。

例

[例1]

プリンタ名(lwprint)および印刷情報ファイルのパス名(/home/usr1/printinf.txt)を割り当てる場合

```
    :
   SELECT 印刷ファイル
        ASSIGN TO “lwprint,,INF(/home/usr1/printinf.txt)”.
    :
```

[例2]

プリンタの割当てを省略し、印刷情報ファイルのパス名(/home/usr1/printinf.txt)だけを割り当てる場合

```
    :
   SELECT 印刷ファイル ASSIGN TO “,,INF(/home/usr1/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制御文でプリンタの割当てを行う必要があります。

[例3]

プリンタの割当てを省略し、印刷情報ファイルのパス名(/home/usr1/printinf.txt)を環境変数CBR_PRT_INFに割り当てる場合

```
    :
   SELECT 印刷ファイル ASSIGN TO “,,”.
    :
```

- 環境変数の設定

```
$ CBR_PRT_INF=/home/usr1/printinf.txt ; export CBR_PRT_INF
```

この場合、ファイル識別名定数は空白文字列でも構いません。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制御文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"CBR_PRT_INF"に印刷情報ファイルのパス名を指定します。

例

[例]

印刷情報ファイルのパス名(/home/usr1/printinf.txt)を環境変数CBR_PRT_INFに割り当てる場合

```
$ CBR_PRT_INF=/home/usr1/printinf.txt ; export CBR_PRT_INF
```

電子帳票出力機能を利用する場合、ASSIGN TO PRINTER(プリンタへの直接印刷)は意味を持ちません。印刷情報ファイルのprtout制御文に指定されたプリンタが有効となります。

7.6.2.2 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制御文の指定を追加します。

stream制御文

パラメタに"LW"を指定します。帳票を電子化することを意味します。stream制御文の詳細については、"7.1.5 印刷情報ファイル"を参照してください。

streamenv制御文

電子帳票情報ファイルのパス名を絶対パスで指定します。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制御文の詳細については、"7.1.5 印刷情報ファイル"を参照してください。電子帳票情報ファイルの詳細については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制御文

ListWorksの電子保存装置名を指定します。prtout制御文は、省略することもできます。prtout制御文を省略した場合、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。両方指定した場合は、prtout制御文の指定が優先されます。prtout制御文の詳細については、"7.1.5 印刷情報ファイル"を参照してください。

7.6.2.3 フォントの指定

ListWorks連携では、帳票データの参照や印刷はListWorksクライアントから行います。ListWorksクライアントがWindowsシステムであるため、Windowsシステムで有効な任意のフォントをCOBOLから指定することができます。この場合、書体番号によるフォント指定を利用し、フォントテーブルファイルにはWindowsシステムで有効なフォントフェース名を指定します。フォントテーブルの詳細については"7.1.9 フォントテーブル"を参照してください。

例

PostScript用のシステム提供のフォントテーブルをWindows(R)システム用にカストマイズする場合

```
[FONT-001]
FontName=ＭＳ 明朝
[FONT-002]
FontName=ＭＳ 明朝
[FONT-003]
FontName=ＭＳ ゴシック
[FONT-004]
FontName=ＭＳ ゴシック
[FONT-031]
```

```
FontName=Courier New
Style=R
[FONT-803]
FontName=Courier New
Style=I
[FONT-804]
FontName=Courier New
Style=B
[FONT-805]
FontName=Courier New
Style=BI
```

注意

- Windows(R)システムで指定できるフォント名の一覧は、コントロールパネルのフォントを選択し、表示されるフォントの一覧を参照してください。
- COBOLのフォント指定を有効にするには、フォントの置換えを行わないようにListWorksのフォント名対応ファイルを編集する必要があります。フォント名対応ファイルの詳細については、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。

7.6.2.4 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。これらの詳細については、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。

7.6.2.5 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

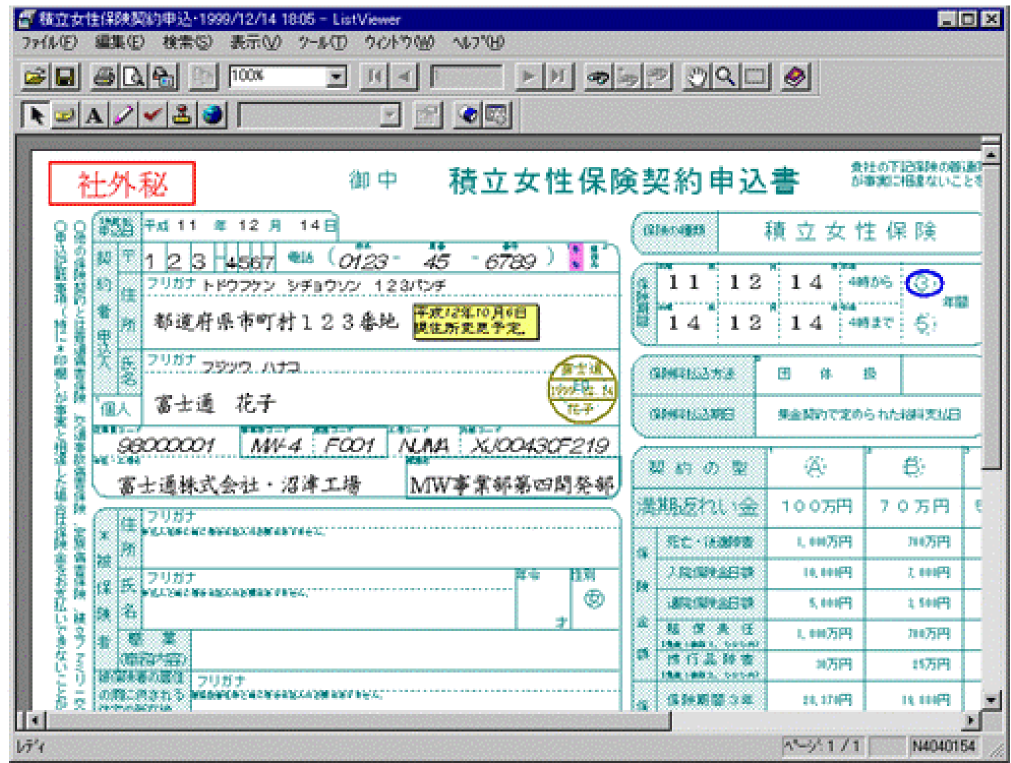
印刷ファイルの指定	既存アプリケーションの例	電子化する例
ASSIGN句の記述	SELECT 印刷ファイル 　ASSIGN TO S-PRTFILE.	SELECT 印刷ファイル 　ASSIGN TO S-PRTFILE.
ファイル識別名の割当て	PRTFILE=/usr/home1/prtfile	PRTFILE=/usr/home1/prtfile ,,INF(/usr/home1/printinf.txt)
印刷情報ファイルの記述	なし	stream　　LW streamenv　/usr/home1/lw.txt prtout　　lwprint

印刷情報ファイルのstream制御文に“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制御文に指定し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制御文にListWorksの電子保存装置名を指定することで、ファイル識別名に指定された出力ファイル名(/usr/home1/prtfile)が無視されます。そして、帳票の出力先が“lwprint”に向けられます。

また、ファイル識別名PRTFILEの割当てを変更しないで、印刷情報ファイルの割当てを環境変数CBR_PRT_INFで行うこともできます。

帳票の電子化に必要な定義はこれだけです。

7.6.3 電子帳票の出力例



7.6.4 プリンタ(紙)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。

7.6.4.1 環境変数を利用する機能

環境変数名	機能	プリンタ出力時	電子帳票出力時
CBR_LP_OPTION	lpコマンドに渡すオプションの指定	有効(注1)	無効
CBR_PRINTFONTTABLE	実行単位に一意のフォントテーブルの指定	有効(注2)	有効
CBR_PRT_INF	実行単位に一意の印刷情報ファイルの指定	有効	有効
FCBDIR	FCB格納ディレクトリの指定	有効	有効
FOVLDIR	フォームオーバレイパターン格納ディレクトリの指定	有効	有効
ファイル識別名	出力先の指定、印刷情報ファイル、およびフォントテーブルの指定	有効	有効

注1: ASSIGN句の記述がPRINTERの場合だけ有効です。
注2: データストリームがPostScriptレベル1の場合だけ有効です。

7.6.4.2 印刷情報ファイルを利用する機能

制御文	機能	プリンタ出力時	電子帳票出力時
ankfont	ANK文字の印字フォントの切替え	有効(注1)	無効
documentname	文書名の指定	無効	有効
fcbname	デフォルトFCBの指定	有効	有効
papersize	デフォルトの用紙サイズの指定	有効	有効(注2)

制御文	機能	プリンタ出力時	電子帳票出力時
printer	データストリームの指定	有効(注3)	無効
prtform	デフォルトの印刷形式の指定	有効	有効(注2)
prtout	出力先の指定	有効	有効
stream	電子帳票出力の指定	有効	有効(注4)
streamenv	電子帳票情報ファイルの指定	無効	有効

注1: データストリームがPostScriptレベル1の場合だけ有効です。
注2: 指定を省略した場合、ListWorksのデフォルトに従います。
注3: stream制御文に"PR"を指定したか、stream制御文の指定を省略した場合に有効です。
注4: 電子帳票出力時は必須です。常に"LW"を指定してください。

7.6.4.3 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	プリンタ出力時	電子帳票出力時
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]	○
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]

(□: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

プリンタによっては、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

電子帳票出力時に利用可能な用紙サイズは、A3/A4/A5/B4/B5/レターです。これ以外の用紙サイズを利用する場合、電子帳票情報ファイルで用紙の縦長／横長を指定します。また、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。

[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)に印刷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、紙に印刷す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あとで紙に印刷する場合には有効となります。

文字属性

COBOLの機能		プリンタ出力時	電子帳票出力時
文字サイズ 3.0～300.0ポ:0.1ポ単位		○	○
文字ピッチ 0.01～24.00CPI:0.01CPI単位		○	○
書体	書体名	○	○
	書体番号 FONT-001～FONT-999	○（注1）	○（注2）
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	○（注1）	○（注2）
水平スキップ 0.01～24.00CPI:0.01CPI単位		○	○

注1: 書体番号指定はデータストリームがPostScriptレベル1のときだけ有効です。ボールド・イタリックの指定は相当する形態を持つフォントを書体番号によって指定することで使用することができます。
注2: Windows(R)システム用にカストマイズしたフォントテーブルを指定する必要があります。

7.6.4.4 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、以下を参照してください。

- ListWorksのオンラインマニュアル
- ListWorksのヘルプ

7.6.5 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.
```

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。まず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てください。エラーの原因を取り除いた後、再度実行してください。

第8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。

8.1 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ことができます(以降、画面入出力と表します)。
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"表8.1 表示ファイル機能とスクリーン操作機能の特徴・用途"に示します。

表8.1 表示ファイル機能とスクリーン操作機能の特徴・用途

特徴・用途		表示ファイル機能	スクリーン操作機能
特徴	画面の設計	画面イメージで設計 (FORMを使用)	画面を行の集まりとして設計
	画面の数	プログラムに定義した表示ファイルの数	1つ
	プログラム実行中の画面の属性の変更	可能	可能
	プログラムの記述内容	・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
	必要な関連製品	・FORM (画面の定義) ・MeFt/NET(リモート入出力処理) ・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

8.2 表示ファイル機能(画面入出力)

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プログラムの作成および注意事項について説明します。

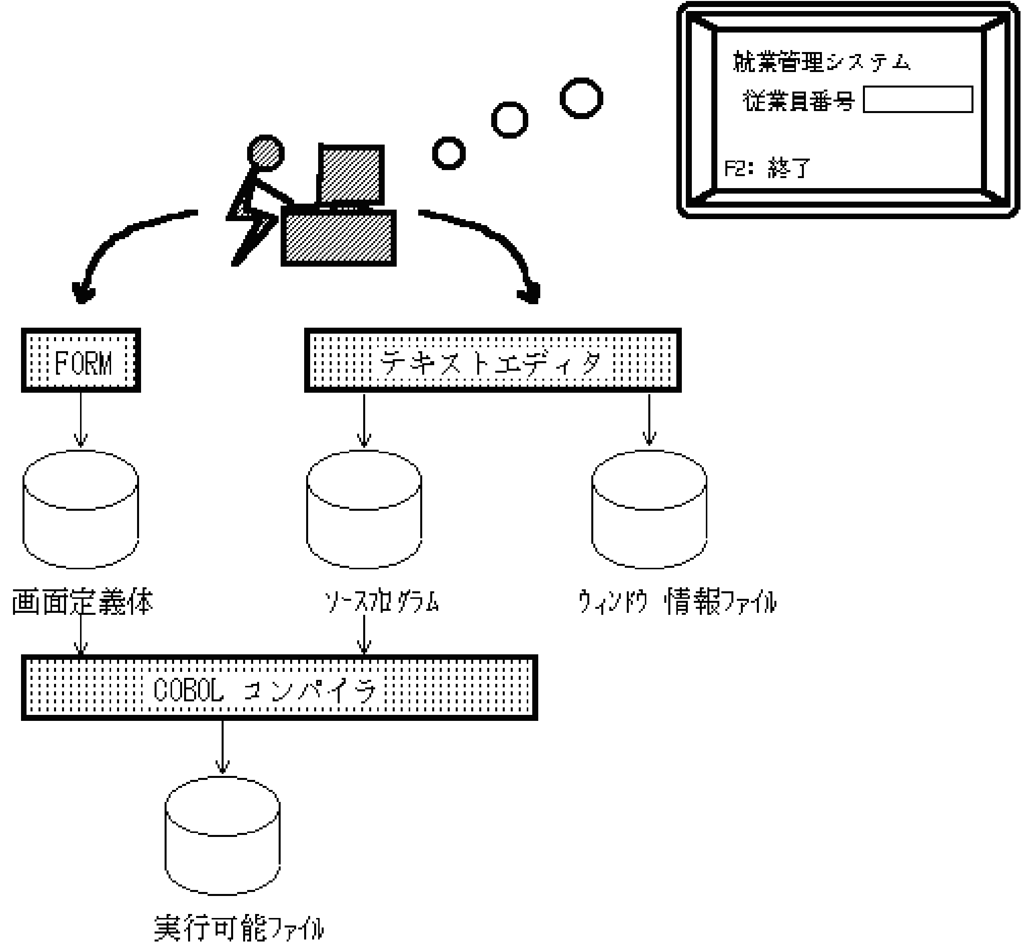
8.2.1 概要

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、開発環境(Windows)上の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、画面定義体で定義し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。特殊レジスタについては"8.2.4 プログラムの記述"の"特殊レジスタの使い方"を参照してください。

表示ファイル機能を使用するときには、FORMで定義した画面定義体と入出力を行うための接続製品が必要となります。

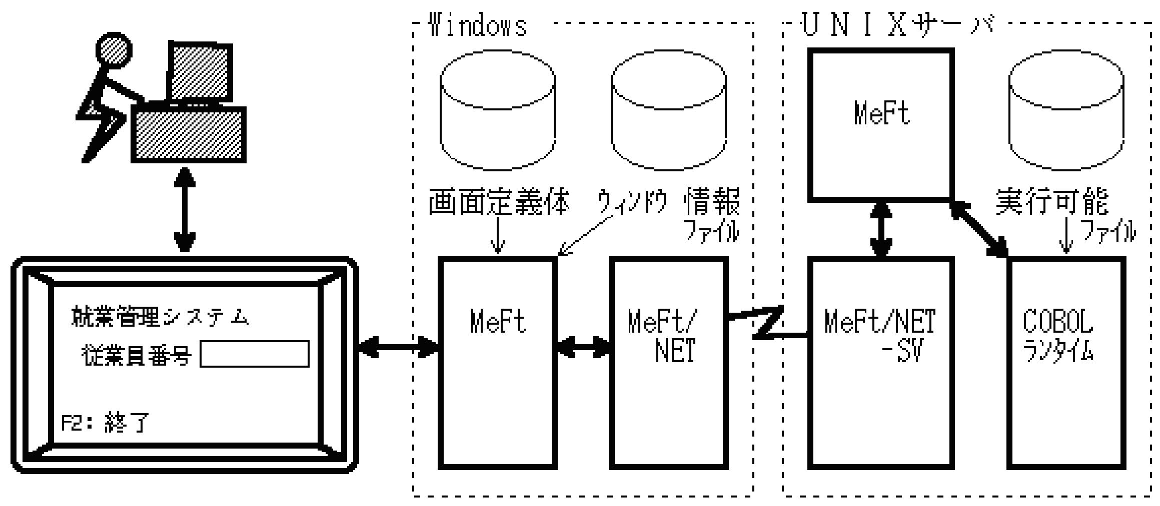
表示ファイル機能を使用するプログラムの、作成時の関連図と動作環境を以下に示します。

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

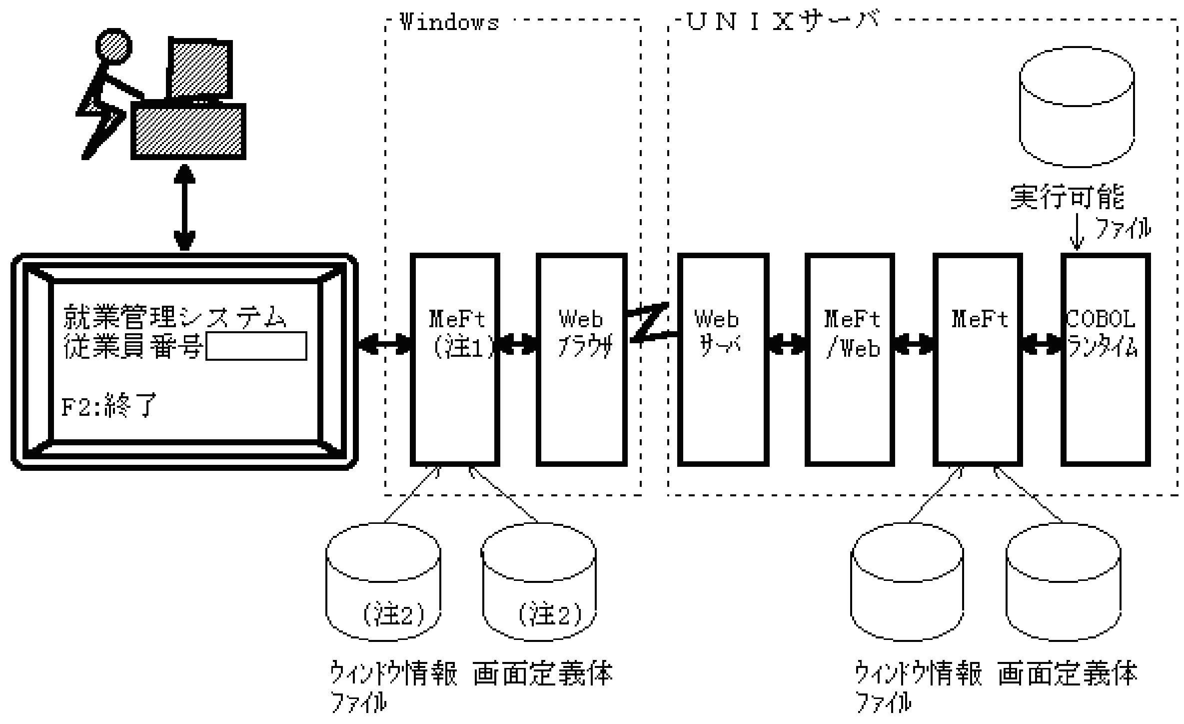


表示ファイル機能を使用するときの動作環境(リモート環境で使用する場合)

MeFt/NET連携



MeFt/Web連携



注1: WebブラウザがNetscape Navigator(TM)の場合はサーバからプラグインをダウンロードする必要があります。WebブラウザがMicrosoft(R) Internet Explorerの場合はサーバから自動的にダウンロードされます。

注2: サーバから自動的にダウンロードされます。

MeFt/NET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インストールガイドを確認してください。

8.2.2 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、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. 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。

8.2.3 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のマニュアルまたは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8.2 画面定義体に設定する情報”に示します。

表8.2 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。

情報の種類		指定する内容および用途
	定義体形式	自由形式を指定します。
	データ項目	画面入出力を行うための領域を指定します。COBOLプログラムを記述するときに、ここで指定した項目名をデータ名として使用します。
	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。COBOLプログラムを記述するときに、ここで指定した項目群名を使用します。
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、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"共用しない(5バイト)"と"共用する(3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"を指定してください。

8.2.4 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通知域  入出力状態２通知域].
 DATA DIVISION.
  FILE SECTION.
  FD  ファイル名.
      COPY  画面定義体名 OF XMDLIB.
 (01  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通知域            PIC X(2).]
 [01  入出力状態２通知域            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  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8.3 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8.3 ファイル管理記述項に指定する情報

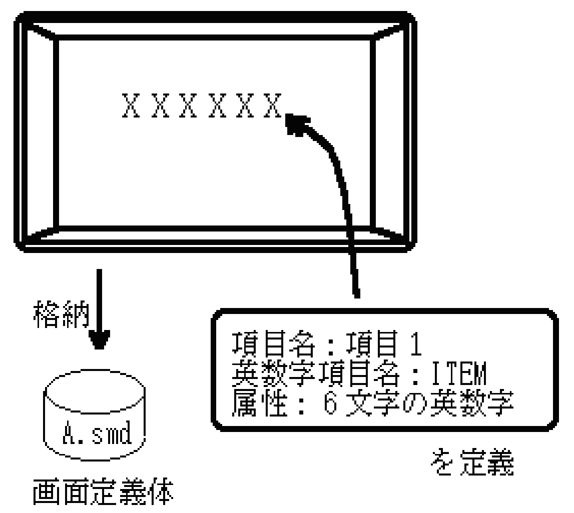
	指定する場所	情報の種類	指定する内容および用途
必須	SELECT句	ファイル名	COBOLプログラム中で使用する表示ファイル名を指定します。このファイル名は、COBOLの利用者語の規則(注)に従った名前にします。
	ASSIGN句	ファイル参照子	"GS-ファイル識別名"の形式で指定します。このファイル識別名は、実行時に接続製品が使用する情報ファイルのパス名を設定する環境変数となります。
	FORMAT句	データ名	作業場所節または連絡節で、8桁の英数字項目として定義したデータ名を指定します。このデータ名には、画面入出力処理を行うとき、画面定義体名を設定します。
	GROUP 句	データ名	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、画面入出力処理を行うとき、処理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。設定される値については、“付録B 入出力状態一覧”を参照してください。詳細情報を取得する場合は、さらに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、入出力処理の処理種別を設定します。設定できる値については、“MeFt/Web説明書”を参照してください。
	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。設定される値については“MeFt/Web説明書”を参照してください。
	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。設定できる値については“MeFt/Web説明書”を参照してください。

注: 利用者語の規則については、“COBOL文法書”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、登録集名に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。



COBOLプログラムの記述

```
COPY A OF XMDLIB.
```

- 項目制御部なし

```
01  レコード1
    02  項目1   PIC X(6).
```

- 項目制御部が5バイト

```
01  レコード1
    02  FILLER  PIC X(5).
    02  項目1   PIC X(6).
```

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文または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/Web説明書”を参照してください。

画面定義体で定義した項目属性のうち、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。

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/Web説明書”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。
- 項目制御部を入力処理と出力処理で“共用する(3バイト)”の画面帳票定義体では、画面帳票定義体に定義された以下の項目について同じ記憶領域が使用されます。
    - EDIT-MODEとEDIT-STATUS
    - EDIT-CURSORとEDIT-OPTION

  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、“6.6 入出力エラー処理”を参照してください。

8.2.5 プログラムの翻訳・リンク

表示ファイルで画面入出力を行うプログラムの翻訳・リンク方法を以下に示します。

cobolコマンドを使って翻訳とリンクを行う場合

```
$ cobol -M -m パス名 [その他の翻訳・リンクオプション] ファイル名
```

-Mオプションは、主プログラムの場合に指定します。

-mオプションに画面定義体を格納したディレクトリのパス名を指定します。

プログラム中で複数の画面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。

ldコマンドを使ってリンクを行う場合

ldコマンドを使ってリンクを行う方法については、“付録L ldコマンド”を参照してください。

注意

リモート環境で画面定義体を使用するプログラムを、cobolコマンドを使ってリンクする場合、-pmオプションを指定する必要はありません。

8.2.6 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。

情報ファイルの詳しい内容や作成方法については、接続製品に応じて以下のマニュアルを参照してください。

接続製品がMeFt/NETの場合

Windows版 MeFtのオンラインマニュアル

接続製品がMeFt/Webの場合

MeFtのオンラインマニュアル、"MeFt/Web説明書"

ウィンドウ情報ファイルに設定する情報を"表8.4 情報ファイルを作成するときに設定する情報"に示します。

表8.4 情報ファイルを作成するときに設定する情報

情報の種類	指定する内容および用途
MEDDIR	画面定義体を格納したディレクトリ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。
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。

注意

MeFt/Web連携で使用するウィンドウ情報ファイルは、クライアント環境のコード系を使用して作成します。詳細は、"MeFt/Web説明書"を参照してください。

8.2.7 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/NET-SVを使用する場合

- ファイル識別名を環境変数として、ディレクトリ名を除いたウィンドウ情報ファイル名を設定します。このとき、ウィンドウ情報ファイルは、Windows(クライアント)側に用意しておく必要があります。
- 環境変数LD_LIBRARY_PATHに、MeFt/NET-SVの格納ディレクトリを設定します。
- 接続製品としてMeFt/NET-SVを使用することを示す環境変数を指定する必要があります。指定方法には、以下の2つの方法があります。
  - ファイル識別名を環境変数名として、接続製品名にMEFTNETを設定する。
  - 環境変数CBR_PSFILE_DSPにMEFTNETを設定する。

  設定方法の詳細については、"4.1.3 環境変数による接続製品の指定"を参照してください。
- その他の実行に必要な環境設定については、"MeFt/NET-SV説明書"を参照してください。

MeFt/Webを使用する場合

- ファイル識別名を環境変数として、ディレクトリ名を除いたウィンドウ情報ファイル名を設定します。
- 環境変数MEFTWEBDIRに、クライアントのMeFt/Webコントロールが使用するウィンドウ情報ファイルの格納ディレクトリをURLまたはサーバのローカルパスで設定します。
- 環境変数LD_LIBRARY_PATHに、MeFt/WebおよびMeFtの格納ディレクトリを設定します。
- その他の実行に必要な環境変数については、"MeFt/Web説明書"を参照してください。

注意

MeFt/Webアプリケーションは、WebサーバあるいはMeFt/Webのデーモン配下で動作するアプリケーションとなります。

8.3 スクリーン操作機能

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として提供されているので、参考にしてください。

スクリーン操作機能は、シフトJISコード(LANG=ja_JP.PCK)およびUnicode(LANG=ja_JP.UTF-8)ではサポートしていません。

8.3.1 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。画面節に定義した画面項目は、行番号と列番号によって、画面に配置されます。

なお、この製品は、画面のレイアウトを画面イメージで簡単に設計することができるユーティリティSCREEN-DESIGNERを提供しています。SCREEN-DESIGNERについては、"第24章 SCREEN-DESIGNERの使い方"を参照してください。

8.3.2 画面項目

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。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を"表8.5 画面項目に指定するCOBOLの句"に示します。

表8.5 画面項目に指定するCOBOLの句

目的		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	**LOWLIGHT 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**	○	○	○	○	×	○

目的		句	画面項目の属性				(注)	
			定数	入力	出力	更新	集団	基本
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すること、基本は基本項目として定義することを示します。

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。

8.3.3 キー定義ファイルの利用

スクリーン機能操作機能では、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たりすることができます。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利用者が任意にカスタマイズすることができます。

キー定義ファイルは、環境変数で指定します。以下に、環境変数CBR_SCR_KEYDEFFILEの指定形式を示します。

```
$ CBR_SCR_KEYDEFFILE=キー定義ファイル名 ; export CBR_SCR_KEYDEFFILE
```

キー定義ファイルの記述形式

キー定義ファイルでは、以下の形式で情報を格納します。

```
[COBOL.KBD]
 キー名=XYZZZ
     :
```

[COBOL.KBD] :見出し
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: CRT STATUS句の状態キー1('1'または'2')
Z: CRT STATUS句の状態キー2('000'から'999'まで定義可能)

ファンクションキーの利用者定義は、定義したキーが画面から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に返却される値および状態キー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は終了します。

キー定義ファイルの例を示します。

例

キー定義ファイル(cobolkbd1)

```
[COBOL.KBD]
 ESC=11000
 F01=01001
 TAB=02006
```

キー定義ファイルの指定

```
$ CBR_SCR_KEYDEFFILE=cobolkbd1; export CBR_SCR_KEYDEFFILE
```

スクリーン操作機能からデータの読込み中にESCキーを押下した場合、CRT STATUS句の状態キー1および状態キー2にそれぞれ'1'と数字0が設定されます。同様にデータの読込み中にF01キーを押下した場合には、キー入力は無視されそのままACCEPT文は続行されます。

キー定義ファイルの雛型および指定可能なキー定義は、COBOLインストールディレクトリ配下のconfig/template/ja/keydeffile(EUCコード系の場合)のファイルに格納されているので参考にしてください。

画面入出力状態の設定値

画面入出力を実行したときに、CRT STATUS句に指定したデータ名に設定される値(状態値)の意味を"表8.6 画面入出力状態の設定値"に示します。

表8.6 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"	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指定されたキーです。
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指定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、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。

- CRT STATUS句の状態キー2の定義としてキー定義ファイルでは、'000'から'999'まで定義可能です。しかし、プログラムの状態キー2は、X"00"～X"FF"の1バイトで表現されます。そのため、キー定義ファイルで有効な指定は、'000'から'255'となります。'255'以上を指定するとプログラムの状態キー2の値は保証されないので注意してください。
- "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)以外の値は割り当てられません。
- キー定義ファイルで同一キー名を複数指定したときには、最初に現れたキー名の情報が有効になります。
- キー定義ファイルの文字列は空白文字を含んではいけません。
- キー定義ファイルのキー名指定は、大文字・小文字を区別して記述してください。
- キー定義ファイルは、注釈記号として"#"または";"を使用できます。行の先頭または行末に注釈記号を記述することができます。
- 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。
- キー名として定義可能であるキーのうち、キー定義ファイルに定義し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。

- キー定義ファイルに記述できる1行あたりの文字数は、最大500文字までです。

8.3.4 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は、"表8.6 画面入出力状態の設定値"を参照してください。

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、更新項目および入出力項目があります。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8.5 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。

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。

Solaris上ではスクリーン操作機能での以下の句は有効になりません。

- GRID
- LEFTLINE

- OVERLINE

また、有効にならない色もあるので注意してください。

8.3.5 プログラムの翻訳・リンク

特に必要な翻訳オプションはありません。リンク方法を、以下に示します。

cobolコマンドを使ってリンクする場合

-pcオプションを指定します。

ldコマンドを使ってリンクする場合

ldコマンドを使ってリンクを行う方法については“付録L ldコマンド”を参照してください。

ファンクションキーの利用者定義を行う場合、環境変数CBR_SCR_KEYDEFFILEを設定します。

注意

- スクリーン操作機能は、画面制御を行うcursesライブラリを使用して実現しています。このため、利用している端末装置のハード的な制限のほかに、スクリーン操作機能(キー操作、色など)が正しく動作しない場合は、以下の事項を確認してください。後者については、利用しているマシンのシステム管理者に確認をとってください。
  - 環境変数TERMに利用している端末の名前を正しく設定しているか。
  - terminfoデータベースに使用している端末の情報を正しく設定しているか。
- スクリーン操作機能により画面が表示されている状態で、小入出力機能のACCEPT文またはDISPLAY文を実行した場合、画面はスクロールされます。
- スクリーン機能で、FOREGROUND-COLORとBACKGROUND-COLORで色が指定されている場合、組み合わせにより、指定された色の組み合わせで表示されない場合があります。
- 画面項目に指定した各句(色、輝度、点滅の指定など)が機能するかどうかは、プログラムを実行する端末装置の機能に依存します。
- FOREGROUND-COLOR句およびBACKGROUND-COLOR句での色指定6は、黄色を意味します。
- スクリーン操作機能でファンクションキーF11およびF12を使用する場合、以下のコマンドをあらかじめ実行しておく必要があります。

```
# xmodmap -e "keycode 16 = F11" -e "keycode 18 = F12"
```

- 挿入キー、削除キーおよびファンクションキーなどが使用できるかどうかは、プログラムを実行する端末装置の定義に依存します。
- xtermのように、画面の大きさを変更できる端末装置を使用する場合、プログラム実行前にresizeコマンドで環境変数LINESおよびCOLUMNSを再設定しておく必要があります。
- 更新項目および入出力項目の操作を行う前に、更新項目および入出力項目に関連付けられているデータ項目に値を設定してください。未設定の場合は、動作は保証されません。

第9章 サブプログラムを呼び出す～プログラム間連絡機能～

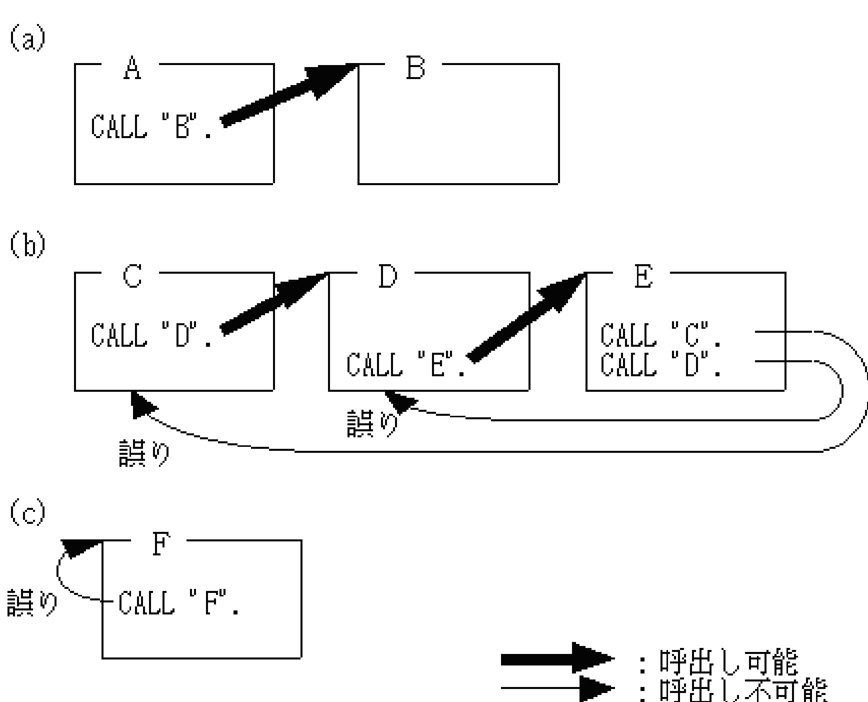
本章では、プログラムからプログラムを呼び出す機能について説明します。この機能をプログラム間連絡機能といいます。

9.1 呼出し関係の形態

ここでは、プログラムの呼出し関係の形態について説明します。

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、COBOLプログラムは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、自分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)。

呼出し関係の形態



9.1.1 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、一般に必要になったときにはじめて開設され、不要になった場合または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、“17.3.1 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を呼び出してください。また、最後のCOBOLプログラムの呼出しのあとでJMPCINT3を呼び出すようにしてください。

JMPCINT2は、COBOLプログラムの初期化手続きを行うサブルーチンです。また、JMPCINT3は、COBOLプログラムの実行環境を閉鎖するサブルーチンです。

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すと、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。そのため、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ません(下図(d))。

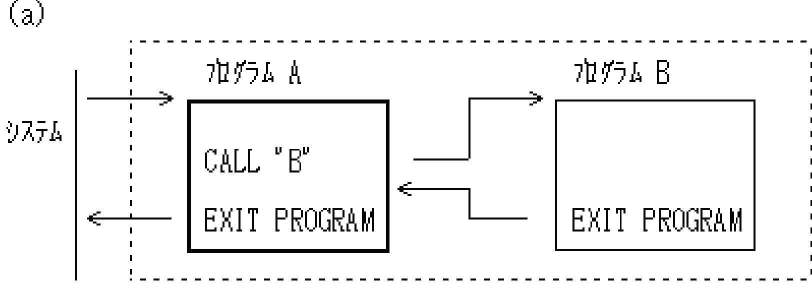
JMPCINT2とJMPCINT3の呼び出し方については、"G.4 他言語連携で使用するサブルーチン"を参照してください。

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9.1 COBOLプログラムだけ

(a)



(b)

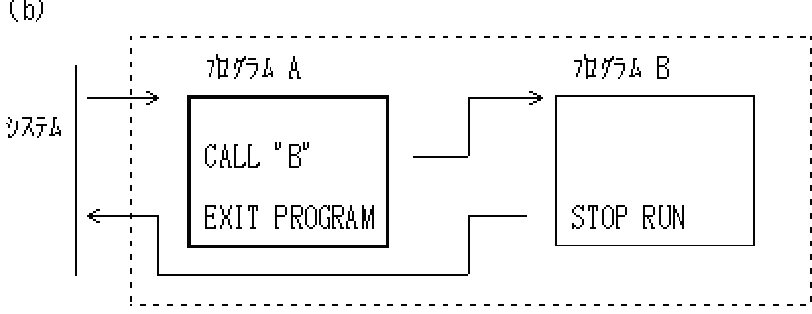
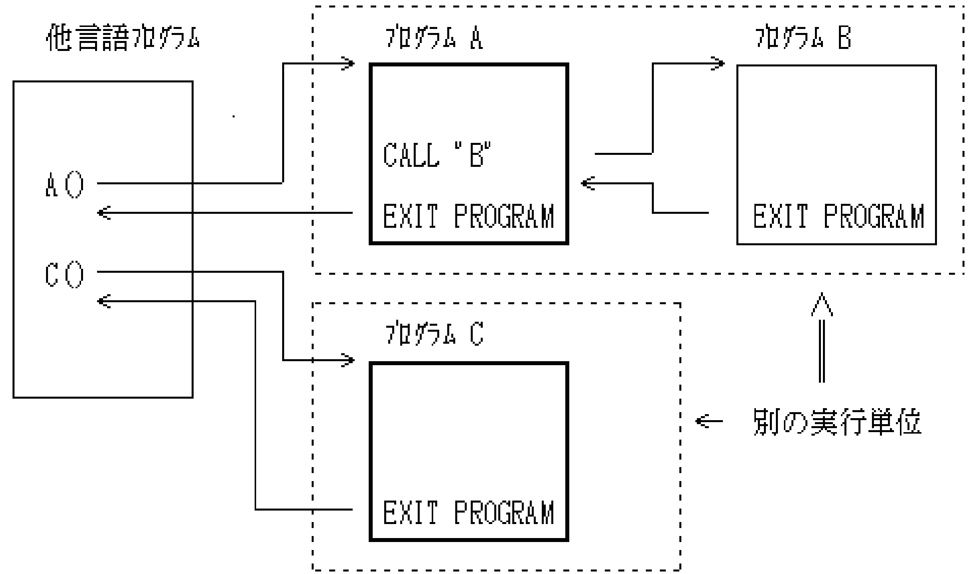
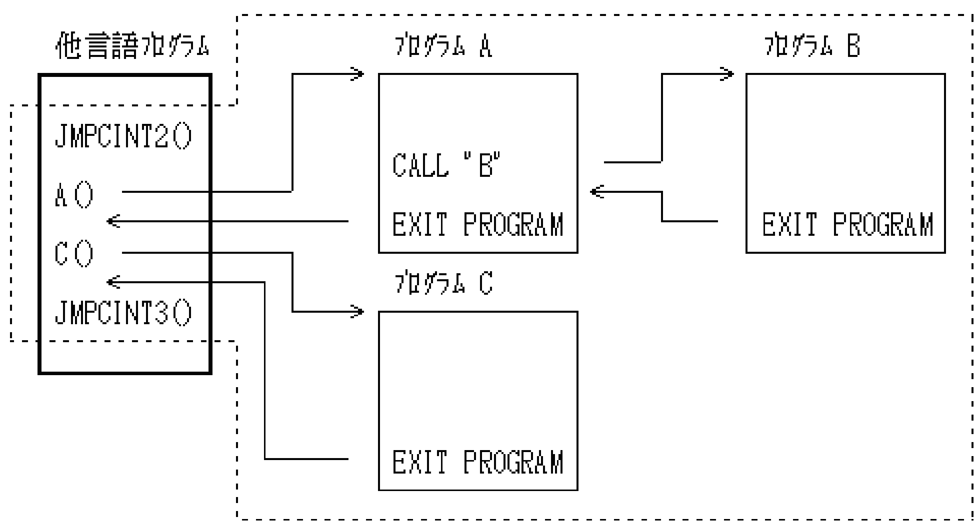


図9.2 他言語プログラムからCOBOLプログラムの呼出し

(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。

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

9.1.2 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。

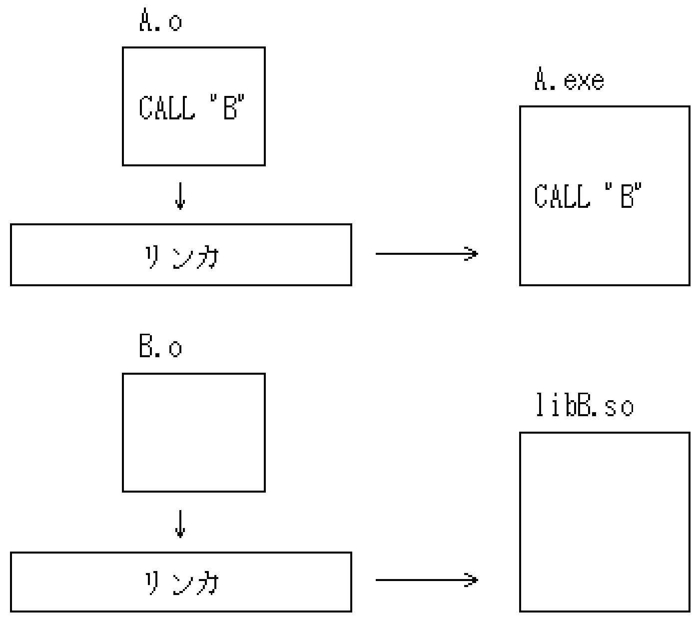
9.1.2.1 動的プログラム構造の特徴

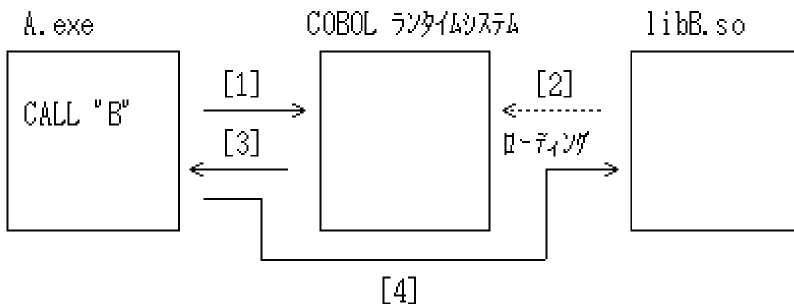
動的プログラム構造を利用すると、副プログラムのメモリ上へのローディングは、実際に副プログラムがCALL文によって呼び出されたときに、COBOLランタイムシステムによって行われます。また、一度ローディングされた副プログラムは、CANCEL文により、メモリ上から削除することができます。このため、実行可能ファイルの起動時に、副プログラムがすべてロードされる単純構造よりも実行可能ファイルの起動は速く、仮想メモリおよび実メモリの節約も期待できます。ただし、副プログラムの呼出しは、COBOLのランタイムシステムを介するために遅くなります。

参考

動的リンク構造との呼出し性能比較について

動的リンク構造により副プログラムを呼び出す場合、ダイナミックリンカは副プログラムのシンボル解決を呼出し元プログラムの呼出し時に行います。この時点では副プログラムのシンボル解決だけ行い、呼出し先の副プログラムはメモリにロードされません。ロードされるのは副プログラムが実質的に呼び出されたときです。そのため呼出し元プログラムの起動時の実行性能については、副プログラムを動的リンク構造で呼び出す場合と動的プログラム構造で呼び出す場合で違いはほとんどありません。





[1]～[4]は、処理する順番を示します。

[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。

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となります。ただし、副プログラムの共用オブジェクトファイル名を“libプログラム名.so”にすることにより、エントリ情報は不要となります。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共用オブジェクトは、1つの副プログラムを1つの共用オブジェクトとし、ファイル名は“libプログラム名.so”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（“9.1.2.3 注意事項”は必ずお読みください）。

9.1.2.2 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、“4.1.4 副プログラムのエントリ情報”を参照してください。

9.1.2.3 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。

副プログラムの作成とリンクについて

動的プログラム構造で副プログラムを呼び出す場合、その副プログラムは共用オブジェクトプログラムとして作成します。また、その共用オブジェクトプログラムファイル名は、“lib副プログラム名.so”という規則に沿って、副プログラムと共用オブジェクトプログラムが一対一に対応している必要があります。

この規則に沿っていれば、エントリ情報ファイルや、副プログラムを-lオプションを使用してそのプログラムを呼ぶプログラムまたは実行可能プログラムにリンクする必要はありません。

1つの共用オブジェクトプログラムにまとめられた複数の副プログラムを動的プログラム構造で呼び出したい場合には、エントリ情報ファイルを使用する方法と、-lオプションを使用してその共用オブジェクトプログラムを実行可能プログラムにリンクする方法があります。これにより、動的プログラム構造では、1つの共用オブジェクトプログラムにまとめられた複数の副プログラムを呼び出すことが可能となります。ただし、-lオプションを使用してリンクした共用オブジェクトファイル中のプログラムに対するCANCEL文は意味をもたなくなるため注意が必要です。

プログラム名について

システム制限により、動的プログラム構造で、日本語文字からなるプログラム名を呼び出すことができません。したがって、CALL文に以下の指定を行うことはできません。

－ 一意名に日本語項目を指定する。

－ 定数に日本語文字定数を指定する。

プログラムの初期状態について

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(“9.2.7 内部プログラム”の初期化プログラムを参照)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。

一意名を指定したCALL文について

a. 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。そのため、C言語連携によりCプログラムを呼び出す場合は、255バイトを超える関数名を呼び出すことはできないので注意してください。なお、COBOLの場合は、プログラム名の最大長が160バイトであるため、問題ありません。

b. 一意名を指定したCALL文で指定された文字列の間に空白が含まれる場合、最初の空白以降の文字列は無視されます。

COBOLアプリケーションで動的プログラム構造と動的リンク構造を混在して使用する場合について

動的プログラム構造と動的リンク構造を混在して使用すると利用者ミスが発生しやすいため、混在してはいけません。使用する場合には、それぞれの構造を十分理解し、以下の点に注意してください。

a. COBOLアプリケーション全体で共通に呼び出されるプログラムの共用オブジェクトを動的プログラム構造で呼び出されるプログラムの共用オブジェクトに動的リンク構造でリンクしてはいけません。この場合、動的プログラム構造で呼び出されるプロ

グラムがCANCEL文でメモリ上から削除されると、動的リンク構造でリンクされているプログラムもメモリ上から削除されます。よって、動的リンク構造でリンクされているプログラムは初期状態にもどされ、他のCOBOLアプリケーションから呼び出された場合、意図した結果にならない場合があります。
COBOLアプリケーション全体で共通に呼び出されるプログラムの共用オブジェクトは、実行可能プログラムに-lオプションでリンクするようにしてください。

b. 翻訳オプション“DLOAD”を指定した場合、そのプログラムから呼び出される副プログラムはすべて動的プログラム構造で呼び出されます。このため、動的リンク構造で呼び出されることを前提としたプログラムを"DLOAD"を指定したプログラムから呼び出すことはできません。なお、以下の機能を使用するプログラムを“DLOAD”を指定して作成する場合は、各機能の共用オブジェクトは実行可能プログラムに-lオプションでリンクしてください。

- コード変換サブルーチン(mbston16sまたはn16stombs)
- COBOL Webサブルーチン
- 簡易アプリ間通信機能

c. ある1つのプログラムを動的プログラム構造および動的リンク構造でそれぞれ呼び出した場合、そのプログラムはCANCEL文により、仮想記憶上から削除することができなくなります。

CANCEL文の実行後の動作について

CANCEL文を実行した場合、副プログラムはメモリから削除されます。しかし、CANCEL文に指定された副プログラムから単純構造または動的リンク構造でリンクされている副プログラムを呼び出している場合、その副プログラムでオープンされたファイルはクローズされません。この場合の動作については保証されないので、CANCEL文に指定されたプログラムから呼びだされている副プログラムでオープンしたファイルはCANCEL文の実行前に必ずクローズしてください。

図9.3 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

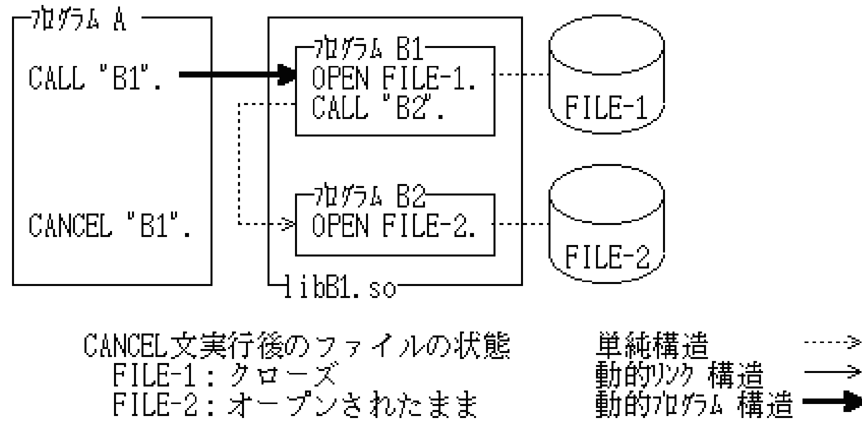
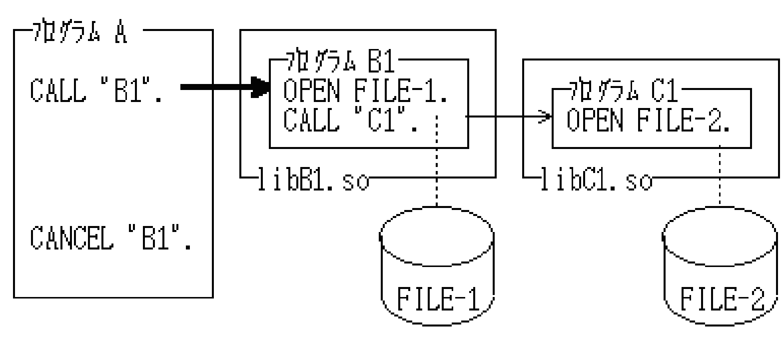


図9.4 キャンセルされる副プログラムが動的リンク構造の副プログラムを呼び出している場合



[備考]
この例では「CALL "B1"」の際にlibB1.so、libC1.soがロードされ、「CANCEL "B1"」の際にlibB1.so、libC1.soが仮想メモリ上から削除されます。

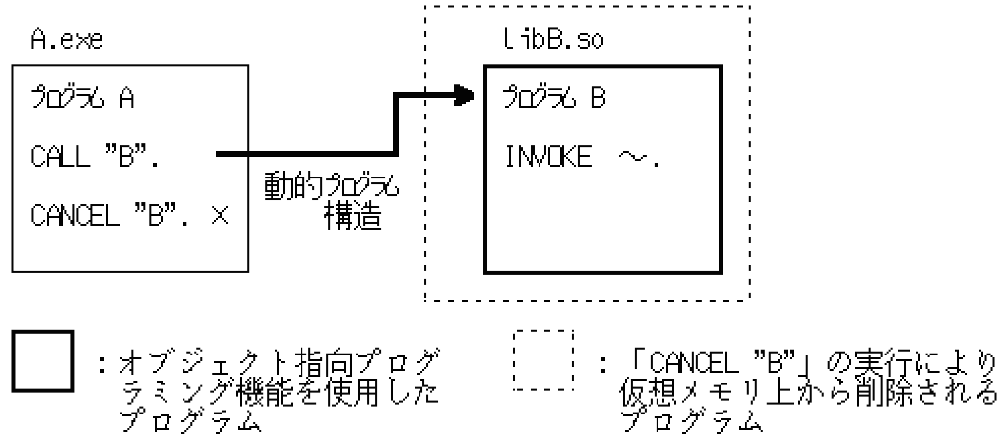
オブジェクト指向プログラミング機能を使用したプログラムについて

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(libB.so)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。

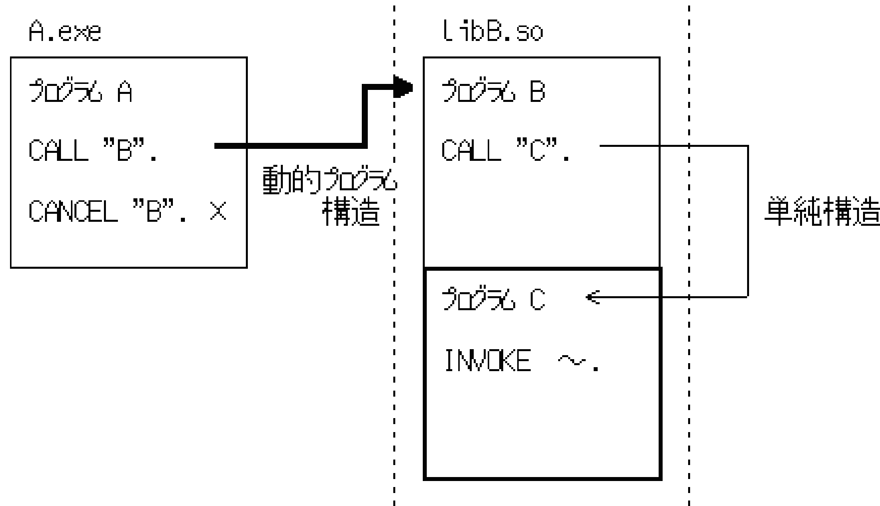


[例2]

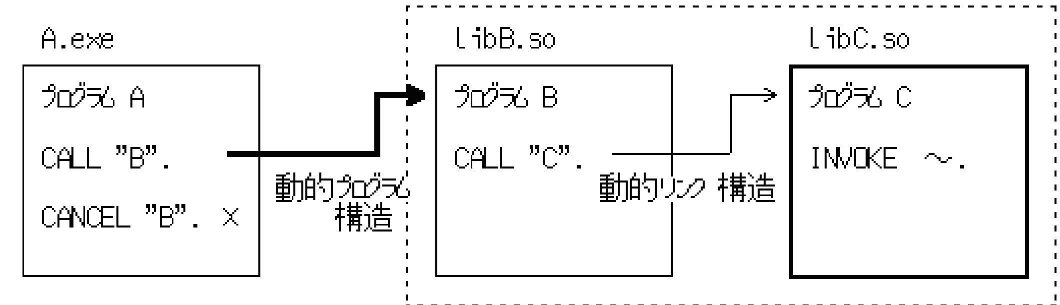
プログラムBとオブジェクト指向プログラミング機能を使用したプログラムCが単純構造または動的リンク構造の場合、「CANCEL "B"」の実行により、プログラムB(libB.so)が仮想メモリ上から削除されます。そのため、プログラムCも仮想メモリ上から削除されるので、このCANCEL文は使用できません。(下図(a)または(b)を参照してください。)

この場合、プログラムBとプログラムC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。

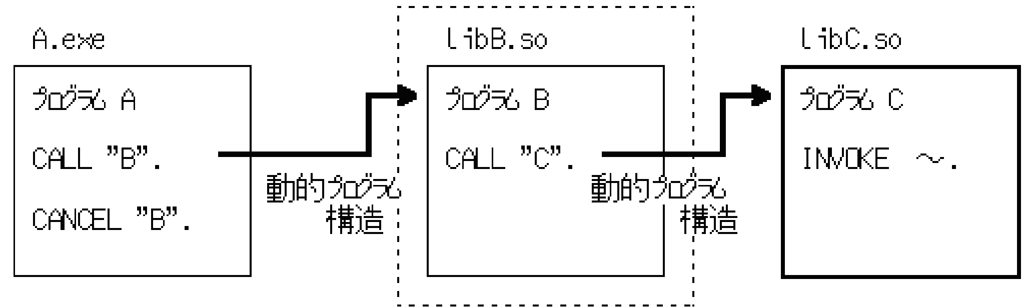
(a) プログラムＢとプログラムＣが単純構造



(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

9.2 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)からほかの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。

9.2.1 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、翻訳オプションDLOADの指定に関係なく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、"3.2.2 結合の種類とプログラム構造"を参照してください。

9.2.2 二次入口

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といい、手続きの途中に設定した入口点を二次入口といいます。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。

9.2.3 制御の復帰とプログラムの終了

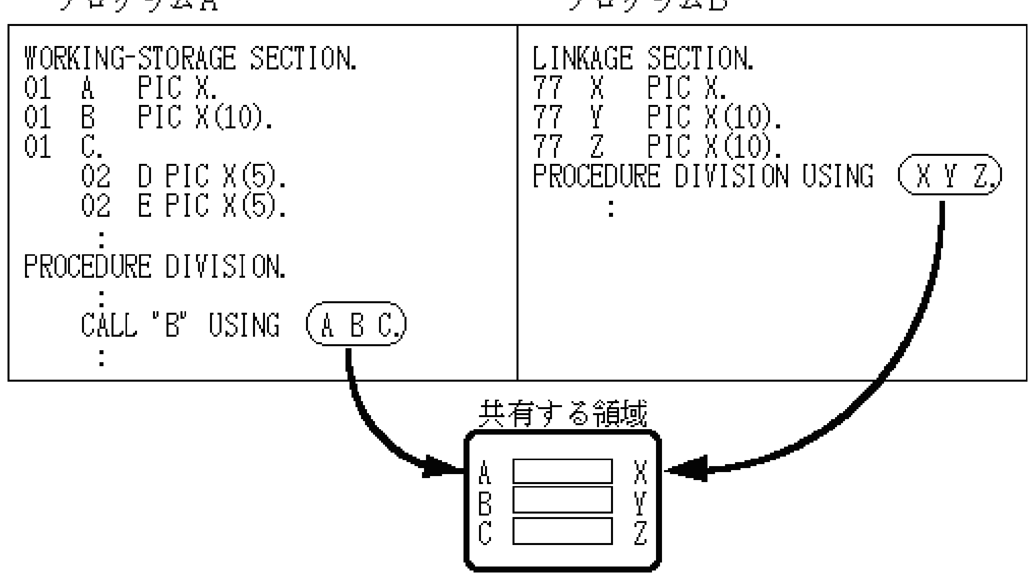
副プログラムから呼ぶプログラムに復帰するには、EXIT PROGRAM文を実行します。EXIT PROGRAM文を実行すると、呼ぶプログラムの実行したCALL文の直後に制御が戻ります。また、すべてのCOBOLプログラムの実行を終了させるには、STOP RUN文を実行します。STOP RUN文を実行すると、COBOLの主プログラムの呼出し元に制御が戻ります。

9.2.4 パラメタの受渡し

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名を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、呼ぶプログラムで、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、次の4つのことが重要です。

- パラメタを受け取るデータ項目を副プログラムの連絡節に定義する。
- 副プログラム側の手続き部の見出しまたはENTRY文のUSING指定にパラメタを受け取るデータ項目を記述する。
- 呼ぶプログラムのCALL文に指定したパラメタの個数と副プログラム側の手続き部の見出しまたはENTRY文のUSING指定に記述したパラメタの個数が一致している、かつ対応するパラメタの長さが一致している。
- 呼ぶプログラムのCALL文に指定した呼出し規約と副プログラム側の手続き部の見出しまたはENTRY文に指定した呼出し規約が一致する。

これらの記述に誤りがある場合、プログラムを正しく動作させることはできません。プログラムの翻訳時や実行時には、次の範囲でこれらの誤りのチェックを行うことができます。

チェック項目	翻訳時	実行時
パラメタ受取り用のデータ項目が連絡節に定義されていない	○	−
パラメタ受取り用のデータ項目のUSING指定への記述漏れ	△ (注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	○ (注2)	○ (注2)

注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。

注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、“5.3 CHECK機能の使い方”を参照してください。

注意

- COBOLプログラムからCOBOLプログラムを呼び出す場合、呼出し側では、“USING BY VALUE”は使用できません。
- パラメタとしてオブジェクト参照項目を受け渡す場合、オブジェクト参照項目のUSAGE句は一致している必要があります。

9.2.5 データの共用

データ記述項またはファイル記述項にEXTERNAL句を指定することにより、複数の外部プログラムの間で共通のデータ領域を使用することができます。EXTERNAL句が指定されると、そのデータまたはファイルは外部属性をもちます。外部属性をもつデータを外部データといい、外部属性をもつファイルを外部ファイルといいます。

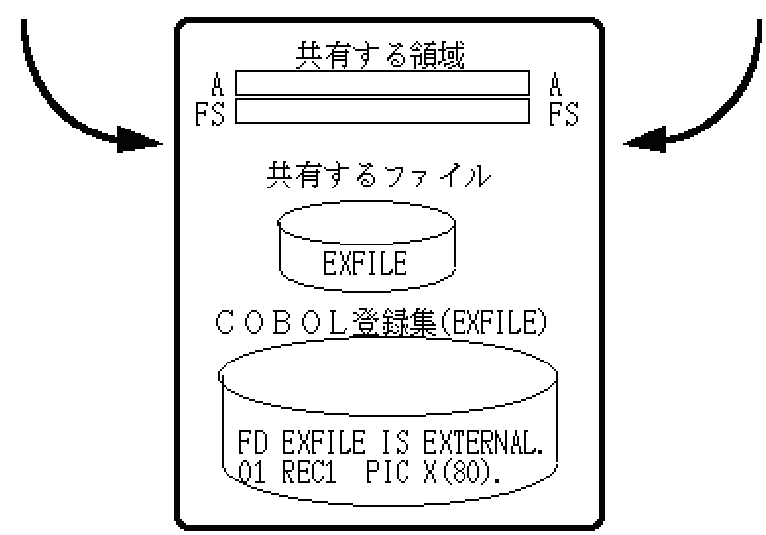
外部データまたは外部ファイルの定義をCOBOL登録集として作成しておき、そのデータをCOPY文でそれぞれのプログラムに取り込むと、保守性のよいプログラムを作成することができます。

プログラムA

```
    :
    SELECT EXFILE
       ASSIGN TO "/usr/data"
       FILE STATUS IS EXFS.
    :
FILE SECTION.
    COPY EXFILE.
    :
WORKING-STORAGE SECTION.
01  EXDATA01 IS EXTERNAL.
    02  A     PIC X(5).
01  EXFS IS EXTERNAL.
    02  FS    PIC X(2).
    :
PROCEDURE DIVISION.
    MOVE SPACE TO A.
    OPEN INPUT EXFILE.
    CALL "B".
    :
```

プログラムB

```
    :
    SELECT EXFILE
       ASSIGN TO "/usr/data"
       FILE STATUS IS EXFS.
    :
FILE SECTION.
    COPY EXFILE.
    :
WORKING-STORAGE SECTION.
01  EXDATA01 IS EXTERNAL.
    02  A     PIC X(5).
01  EXFS IS EXTERNAL.
    02  FS    PIC X(2).
    :
PROCEDURE DIVISION.
    MOVE "ABCDE" TO A.
    :
    READ  EXFILE.
    :
```



9.2.5.1 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する必要があります。

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。

9.2.5.2 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。ただし、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ため、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、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。

9.2.6 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、一致する必要があります。

- 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.
```

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、値を設定する必要があります。値が設定されていない場合、呼び出したプログラムのCALL文のRETURNING指定に記述された項目の値は、不定となります。

特殊レジスタPROGRAM-STATUSは、暗に"PIC S9(9) COMP-5"として宣言され、利用者自身がプログラム中で定義する必要はありません。

副プログラムが特殊レジスタPROGRAM-STATUSに値を設定すると、その値は呼ぶプログラムの特殊レジスタPROGRAM-STATUSに設定されます。

- プログラムA

```
     :
  MOVE 0 TO PROGRAM-STATUS.
  CALL "B".
  IF PROGRAM-STATUS NOT = 0
     THEN
      :
```

- プログラムB

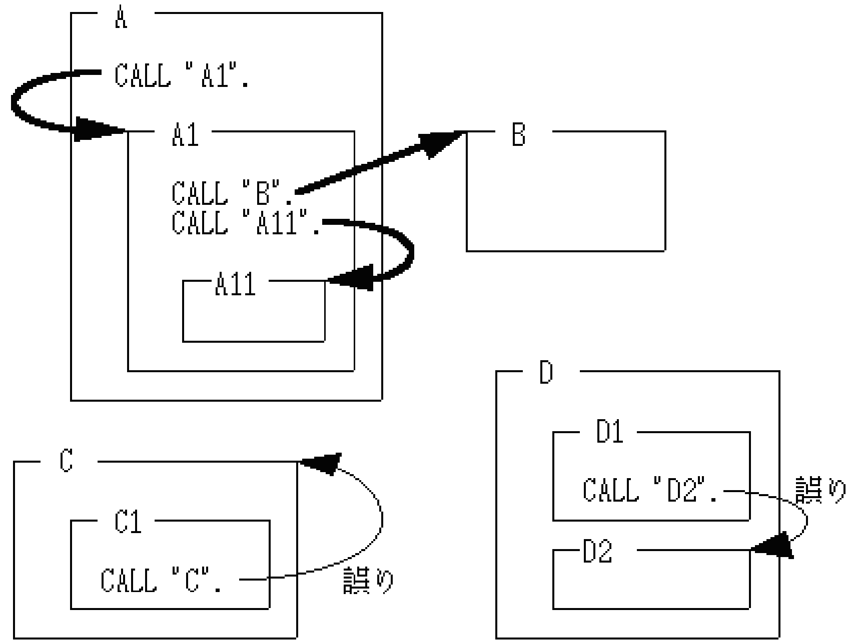
```
      :
   IF  エラー発生
     THEN
       MOVE 99 TO PROGRAM-STATUS
       EXIT PROGRAM
     END-IF.
```

特殊レジスタPROGRAM-STATUSを下層のプログラムから暗に上層のプログラムへ引き継ぐプログラム構造の場合、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します。この場合、上層のプログラムの特殊レジスタPROGRAM-STATUSには、値が引き継がれません。

9.2.7 内部プログラム

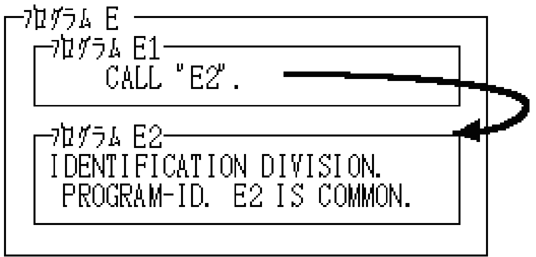
COBOLプログラムは、プログラムの構造の観点から、外部プログラムと内部プログラムに分類されます。ほかのプログラムに含まれない一番外側のプログラムを外部プログラムといいます。外部プログラムに直接的または間接的に含まれているプログラムを内部プログラムといいます。

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ほか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。



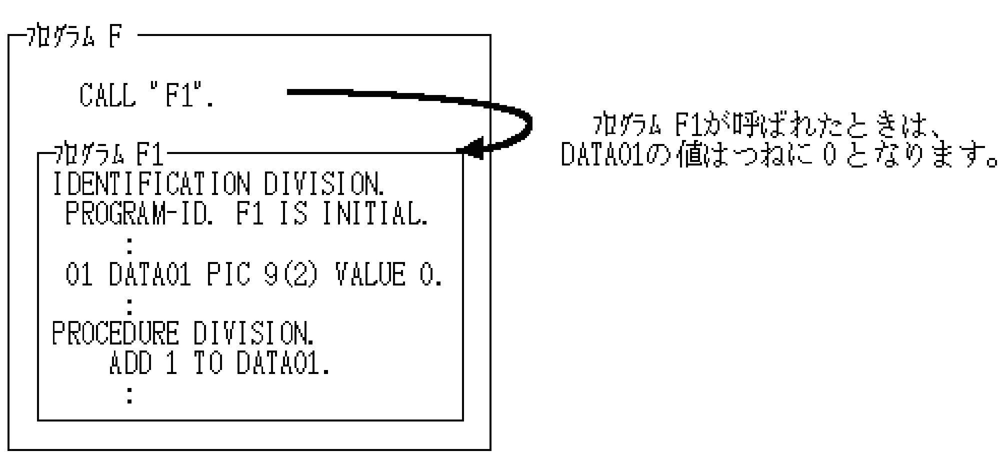
共通プログラム

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、常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常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、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。しかし、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。

9.2.8 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。

  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、常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、常に英大文字として扱われます。

参照

"A.2.1 ALPHAL(英小文字の扱い)"

- 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。特に、プログラム名に英小文字を使用するときには、翻訳オプションNOALPHALを指定することをおすすめします。
- 主プログラムを翻訳するときには、翻訳オプションMAINを指定する必要があります。また、副プログラムを翻訳するときには、翻訳オプションNOMAINを指定する必要があります。

参照

"A.2.22 MAIN(主プログラム/副プログラムの指定)"

- システムの制限により、動的リンク構造および動的プログラム構造で、日本語文字からなるプログラム名やメソッド名を呼び出すことができません。したがって、CALL文またはINVOKE文に以下の指定を行うことはできません。
  - 一意名に日本語項目を指定する。
  - 定数に日本語文字定数を指定する。

9.3 C言語プログラムとのリンク

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を呼び出す方法、およびC言語で記述されたプログラムからCOBOLプログラムを呼び出す方法について説明します。なおここでは、C言語で記述されたプログラムをCプログラムといいます。

9.3.1 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 int 関数名 (構造体名  *仮引数１,
                 short int *仮引数２,
                 char       仮引数３)
{
    ～
    return(関数値);
}
```

9.3.1.1 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。

9.3.1.2 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。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(COBOLとCのデータ型の対応については、"表9.1 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)をもつポインタを仮引数として宣言します。

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も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

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

－ 実行結果

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

上記のようなCOBOLプログラムからCプログラムを呼んだ結果は、BY REFERENCE指定のPRM1の内容が20になり、BY CONTENT指定のPRM2の内容は10のままとなります。これは、BY REFERENCE指定で受け渡すパラメタは、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。一方、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

－ 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.
```

－ Cプログラム

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

－ 実行結果

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

BY REFERENCE指定がその項目のアドレスを渡す方法であるのに対して、BY VALUE指定は項目の値そのものを渡すことになります。したがって、BY VALUE指定もBY CONTENT指定と同様に、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

タの内容は変更されません。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。

参考

BY指定を省略した場合の扱いは、BY REFERENCE指定となります。

9.3.1.3 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、"9.3.3 データ型の対応"を参照してください

- プログラムCOB

```
        :
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
:
```

- 関数C

```
        :
typedef struct
 {
    char aitem1 [10];
    char aitem2 [20];
  } agrp;

short int C (agrp *agrpp,
             short int *b)
{
  return(0);
}
```

特殊レジスタPROGRAM-STATUSで受け取る場合、Cの関数型は、long int型の関数として記述する必要があります。

- プログラムCOB

```
        :
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
```

- 関数C

```
        :
typedef struct
 {
    char aitem1 [10];
    char aitem2 [20];
  } agrp;

long int C (agrp *agrpp,
            short int *b)
{
  return(0);
}
```

注意

- long int型以外のCプログラムの関数値を受け取る場合

  long int型以外の関数値は、CALL文のRETURNING指定で受け取ります。short int型のCプログラムを呼出す場合の例を以下に示します。

  [例] short int型のCプログラム呼出し

```
         :
  01  関数値    PIC S9(4) COMP-5.
         :
     CALL "Cprog" RETURNING 関数値.
     IF 関数値 = 0 THEN   ～
         :
```

  [備考]
  特殊レジスタPROGRAM-STATUSの属性は、Cのlong int型に対応します。したがって、short int型のCプログラムを呼び出した場合、特殊レジスタPROGRAM-STATUSでは、正しい関数値を受け取ることができません。

- void型のCプログラム呼出しの場合

  void型のCプログラムを呼出した場合、特殊レジスタPROGRAM-STATUSが不定な値で更新されてしまいます。特殊レジスタPROGRAM-STATUSが更新されないようにするためには、以下の例に示すようにダミーのデータ項目(PIC S9(9) COMP-5)をRETURNING指定に記述してください。

  [例] void型のCプログラム呼出し

```
         :
  01  DUMMY-RET    PIC S9(9) COMP-5.
         :
     CALL "Cprog" RETURNING DUMMY-RET.
         :
```

9.3.2 CプログラムからCOBOLプログラムを呼び出す方法

ここでは、CプログラムからCOBOLプログラムを呼び出す方法について説明します。

- Cプログラム

```
関数名()
{
  typedef struct
  { char 要素1[4];
    char 要素2[10];
```

```
  }構造体名;
  extern void JMPCINT2(), JMPCINT3();
  extern long int プログラム名(構造体名 *, short int *);
  構造体名        実引数1;
  short int       実引数2;
      :
  JMPCINT2();
  if (プログラム名(&実引数1, &実引数2))
      :
  JMPCINT3();
  return(0);
}
```

- COBOLプログラム

```
IDENTIFICATION DIVISION.
    PROGRAM-ID.   プログラム名.
 DATA DIVISION.
 LINKAGE SECTION.
 01 データ名1.
   02 要素1 PIC 9(4).
   02 要素2 PIC X(10).
 77 データ名2  PIC S9(4) COMP-5.
 PROCEDURE DIVISION
                    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.
```

9.3.2.1 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。

主プログラムがCプログラムで、COBOLプログラムを呼び出す場合、最初のCOBOLプログラム呼出しの前にJMPCINT2を呼び出します。そして、最後のCOBOLプログラム呼出しのあとでJMPCINT3を呼び出します。JMPCINT2は、COBOLプログラムの初期化手続きを行うサブルーチンです。また、JMPCINT3は、COBOLプログラムの終了手続きを行うサブルーチンです。

注意

JMPCINT2およびJMPCINT3の呼出しを行わずにCOBOLプログラムの呼び出すと、COBOLプログラムを呼び出すたびにCOBOLプログラムの実行環境の開設/閉鎖処理が行われます。そのため、実行性能が低下します。

9.3.2.2 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す。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。

9.3.2.3 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])と対応している必要があります。データ型の対応については、“9.3.3 データ型の対応”を参照してください。

- Cプログラム

```
C()
{
  typedef struct
  { char aitem1[10];
    char aitem2[20];
  }agrp;
  extern void JMPCINT2(), JMPCINT3();
  extern short int COB(agrp *agrpp, short int *b);  // [1]
  agrp      prm1;
  short int prm2;
       :
  JMPCINT2();
  if (COB(&prm1, &prm2)==0)
       :
  JMPCINT3();
}
```

- COBOLプログラム

```
PROGRAM-ID.  COB.
 DATA DIVISION.
 LINKAGE SECTION.
 01 AGRP.
    03 AITEM1 PIC X(10).
    03 AITEM2 PIC X(20).
 77 B         PIC S9(4) COMP-5.
 01 RTN-ITM   PIC S9(4) COMP-5.  *>[2]
 PROCEDURE DIVISION
                   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。

PROGRAM-STATUS

特殊レジスタPROGRAM-STATUSで関数値を渡す場合、Cプログラムでは関数値をlong int型として受け取る必要があります。

- Cプログラム

```
C()
{
  typedef struct
  { char aitem1[10];
    char aitem2[20];
  }agrp;
  extern void JMPCINT2(), JMPCINT3();
  extern long int COB(agrp *agrpp, short int *b);
  agrp      prm1;
  short int prm2;
     :
  JMPCINT2();
  if (COB(&prm1, &prm2)==0)
```

```
      :
   JMPCINT3();
}
```

- COBOLプログラム

```
PROGRAM-ID.  COB.
 DATA DIVISION.
 LINKAGE SECTION.
 01 AGRP.
    03 AITEM1 PIC X(10).
    03 AITEM2 PIC X(20).
 77 B         PIC S9(4) COMP-5.
 PROCEDURE DIVISION
                   USING AGRP B.
     MOVE 0 TO PROGRAM-STATUS.
     EXIT PROGRAM.
```

9.3.3 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み合わせは任意です。しかし、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方法を下表に示します。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。

表9.1 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
	int	77 G PIC S9(9) COMP-5. あるいは	int G;	4バイト

COBOLのデータ項目		Cのデータ型	COBOLでの記述例	Cでの宣言例	大きさ
			77 G USAGE IS BINARY-LONG SIGNED.		
		long long int	77 H PIC S9(18) COMP-5. あるいは 77 H USAGE IS BINARY-DOUBLE SIGNED.	long long int H;	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 IGRP. 02 I1 PIC S9(4) COMP-5. 02 I2 PIC X(4).	struct{ short int I1; char I2[4]; } IGRP;	6バイト
内部浮動小数点	単精度	float	01 J COMP-1.	float J;	4バイト
	倍精度	double	01 K COMP-2.	double K;	8バイト

注1: COBOLでの外部10進項目の内部表現は、符号を表す文字と数字からなる文字列です。したがって、Cプログラムではこれを、数値データとしてではなく文字データとして取り扱います。Cプログラムで、外部10進項目を数値データとして扱いたい場合には、Cプログラムで型変換する必要があります。

注2: USAGE IS COMP-5の2進項目は、その桁数によってCプログラムのshort intまたはlong intに以下のように対応します。

- 1～4桁(BINARY(BYTE)オプション指定時は3～4桁): short int
- 5～9桁(BINARY(BYTE)オプション指定時は7～9桁): 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下さい。

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

注3: Cプログラムのshort int、long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。

9.3.4 C言語プログラムとのデータの共用

COBOLプログラムの外部データ項目とC言語プログラムの外部変数との間で、同じ名前のデータを共用させることができます。COBOLプログラムの外部データ項目に対し、C言語プログラムの外部変数との共用を可能とする属性を与えるには、EXTERNAL句を指定します。

・ DEFINITION(またはDEF)を指定すると、COBOLの外部データ項目をC言語プログラムの外部変数として参照できます。

EXTERNAL { DEFINITION / DEF }

C言語プログラムから参照可能な外部データ項目を定義する(データの実体をCOBOLプログラムが持つ)場合に指定します。

－ 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
       PIC X(10) EXTERNAL DEFINITION.
 01   データ名２
       PIC S9(4) COMP-5 EXTERNAL DEF.
PROCEDURE DIVISION.
    CALL  関数名.
    ～
END PROGRAM プログラム名.
```

－ Cプログラム

```
extern char       データ名１[10];
extern short int  データ名２;
long int    関数名 ( )
   {
      ～
    return (0);
   }
```

・ REFERENCE(またはREF) を指定すると、C言語プログラムの外部変数をCOBOLの外部データ項目として参照できます。

EXTERNAL { REFERENCE / REF }

C言語プログラムで定義した外部変数を参照する(データの実体をC言語プログラムが持つ)場合に指定します。

－ 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
       PIC X(10) EXTERNAL REFERENCE.
 01   データ名２
       PIC S9(4) COMP-5 EXTERNAL REF.
PROCEDURE DIVISION.
```

```
    CALL  関数名.
     ～
END PROGRAM プログラム名.
```

— Cプログラム

```
char        データ名１[10];
short int   データ名２;
long int    関数名 ( )
   {
      ～
    return (0);
   }
```

注意

1つの実行単位の中では、C言語プログラムの外部変数との共用属性を持つ外部データ項目と共用属性を持たない外部データ項目との間では、異なるデータ名を指定してください。

9.3.5 翻訳・リンク方法

C言語プログラムからCOBOLプログラムを呼び出す形態のアプリケーションの翻訳・リンク方法を各プログラム構造の例を用いて説明します。なお、プログラム構造については、"3.2.2 結合の種類とプログラム構造"を参照してください。

注意

C/C++などの他言語プログラムとのリンクでは、他言語プログラムの動作に必要なライブラリをリンクする必要があります。他言語プログラムの動作に必要なライブラリについては、その言語のマニュアルを参照してください。

例

主プログラム
x.c
(C言語プログラム)

副プログラム
y.cobol
(COBOLプログラム)

単純構造の場合

```
$ cc -c -o x.o x.c                                [1]
$ cobol -dn -Ns -o x x.o y.cobol (注)             [2]
```

[1] 再配置可能プログラム(x.o)を生成します。

[2] 実行可能プログラム(x)を生成します。

注:他言語が明または暗に使用しているライブラリを、適切にリンクしてください。

動的リンク構造の場合

```
$ cobol -dy -G -o liby.so y.cobol                 [1]
$ cc -c -o x.o x.c                                [2]
$ cobol -dy -Ns -ly -o x x.o (注)                 [3]
```

[1] 共用オブジェクト(liby.so)を生成します。

[2] 再配置可能プログラム(x.o)を生成します。

[3] 実行可能プログラム(x)を生成します。

注:他言語が明または暗に使用しているライブラリを、適切にリンクしてください。

動的プログラム構造の場合

C言語プログラムからCOBOLプログラムを呼び出す形態のアプリケーションで、動的プログラム構造を実現することができます。それは、C言語プログラムからdlopen/dlsym/dlclose関数を使用してCOBOLプログラムのローディング、呼出し、削除を行う必要があります。

なお、JMPCINT3を呼び出す場合、dlclose関数によるCOBOLプログラムの削除は、JMPCINT3を呼び出したあとに行う必要があります。

例

- x.c (C言語プログラム)

```
#include <stdio.h>
#include <dlfcn.h>

 main()
   {
   void * handle;
   long int (*fptr)(char *);
   char *arg,*errp;
   long int rtncd;

   arg="SAMPLE PROGRAM";

/* COBOL の副プログラムを仮想メモリ空間にローディング */
   handle=dlopen("liby.so", RTLD_NOW);

    /* ローディングの成功/失敗を判定 */
     if (handle==NULL) {

       /*  診断情報の取出し */
       errp=dlerror();
       printf("%s¥n",errp);
       return 1;
     }
    /* COBOL の副プログラムの入口点'y' のアドレスの取り出し */
     fptr=(long int(*)(char *))dlsym(handle,"y");

    /* COBOL の副プログラムの呼出し */
     rtncd=(*fptr)(arg);

    /* COBOL の副プログラムを仮想メモリ空間から削除 */
     dlclose(handle);
   }
```

- y.cobol (COBOL プログラム)

```
@OPTIONS NOALPHAL
 IDENTIFICATION              DIVISION.
 PROGRAM-ID.      y.
 ENVIRONMENT      DIVISION.
 DATA             DIVISION.
 WORKING-STORAGE SECTION.
 LINKAGE SECTION.
 01 PARM  PIC X(30).
 PROCEDURE        DIVISION
                 USING PARM.
```

```
    DISPLAY PARM.
    EXIT PROGRAM.
```

```
$ cobol -dy -G -o liby.so y.cobol                              [1]
$ cc -c -o x.o x.c                                             [2]
$ cobol -dy -Ns -o x x.o  (注1) (注2)                           [3]
```

[1] 共用オブジェクト(liby.so)を生成します。

[2] 再配置可能プログラム(x.o)を生成します。

[3] 実行可能プログラム(x)を生成します。

注1: 動的プログラム構造では、主プログラムでdlopen関数を使用して副プログラムをローディングするため、実行可能プログラムを生成するときに、副プログラムをリンクする必要はありません。

注2: 他言語が明または暗に使用しているライブラリを、適切にリンクしてください。

注意

以下の機能を使用した副プログラム(COBOLプログラム)の呼出しを行う場合、主プログラム(C言語プログラム)のリンク時にcobolコマンドのオプションを指定してください。

C-ISAM	-pi
スクリーン操作機能	-pc
MeFtを使用した表示ファイル	-pm

C-ISAMのライブラリ(libisam.a)は、アーカイブライブラリだけの提供となっています。そのため、COBOLでC-ISAMを使用する場合に、libcobcim.soを主プログラムにリンクする必要があります。

9.3.6 注意事項

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。

- COBOLの文字列の終わりには、C言語の文字列と違って自動的にナル文字が挿入されることはありません。
- 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。(指定してもほか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。
- 定数指定のCALL文によって小文字を含むプログラムを呼び出す場合には、翻訳オプションNOALPHALを指定して翻訳してください。翻訳オプションにALPHALを指定して翻訳した場合、プログラム名が常に大文字として扱われます。この場合、定数指定のCALL文によって小文字を含むプログラムを呼び出すことができません。

例

```
CALL "abc".
```

翻訳オプションNOALPHALを指定した場合、CALL文を実行するとプログラム"abc"が呼び出されます。
翻訳オプションALPHALが有効な場合、CALL文を実行するとプログラム"ABC"が呼び出されます。

```
MOVE "abc" TO A.
CALL A.
```

プログラム"abc"が呼び出されます。

- プログラム名段落に記述したプログラム名に小文字が含まれる場合には、翻訳オプションNOALPHALを指定してください。翻訳オプションにALPHALを指定して翻訳した場合、プログラム名が常に大文字として扱われるため、小文字を含むプログラム名を定義することができません。

例

```
PROGRAM-ID. abc.
```

翻訳オプションNOALPHALを指定した場合、プログラム名はabcとなります。
翻訳オプションALPHALが有効な場合、プログラム名はABCとなります。

- COBOLプログラムからCプログラムを呼び出す場合、パラメタに使用するデータ項目は、対応するCの変数の記憶領域の境界に合うように定義する必要があります。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。
- 主プログラムが他言語であっても、COBOLプログラムが呼び出されている場合は、主プログラムにCOBOLプログラムが動作するために必要なライブラリをリンクする必要があります。(リンクするライブラリについては、"付録L ldコマンド"を参照してください。)
- 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、またはCOBOLプログラムからC++プログラムを呼び出す場合は、以下のように「extern "C"」を指定してください。
  - C++プログラムからCOBOLプログラムを呼び出す場合
    - C++プログラム

```
#include <stdio.h>

extern "C" void JMPCINT2();
extern "C" void JMPCINT3();
extern "C" void COBSUB(int *P);

int main()
{
  int prm1;

  JMPCINT2();
  COBSUB(&prm1);
  JMPCINT3();
}
```

    - COBOLプログラム

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

DATA            DIVISION.
LINKAGE SECTION.
 01 C-PRM  PIC S9(8) COMP-5.
PROCEDURE DIVISION USING C-PRM.
     EXIT PROGRAM.
```

  - COBOLプログラムから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 COBPRM-1 PIC X(10).
   02 COBPRM-2 PIC X(20).
 77 COBPRM2 PIC S9(8) COMP-5.
```

```
PROCEDURE DIVISION.
    CALL "CSUB" USING COBPRM1 COBPRM2.
```

- C++プログラム

```
#include <stdio.h>

typedef struct
{
  char prm1[10];
  char prm2[20];
}argp;

extern "C" void CSUB(argp *argpp, int *B)
{
  return;
}
```

第10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。

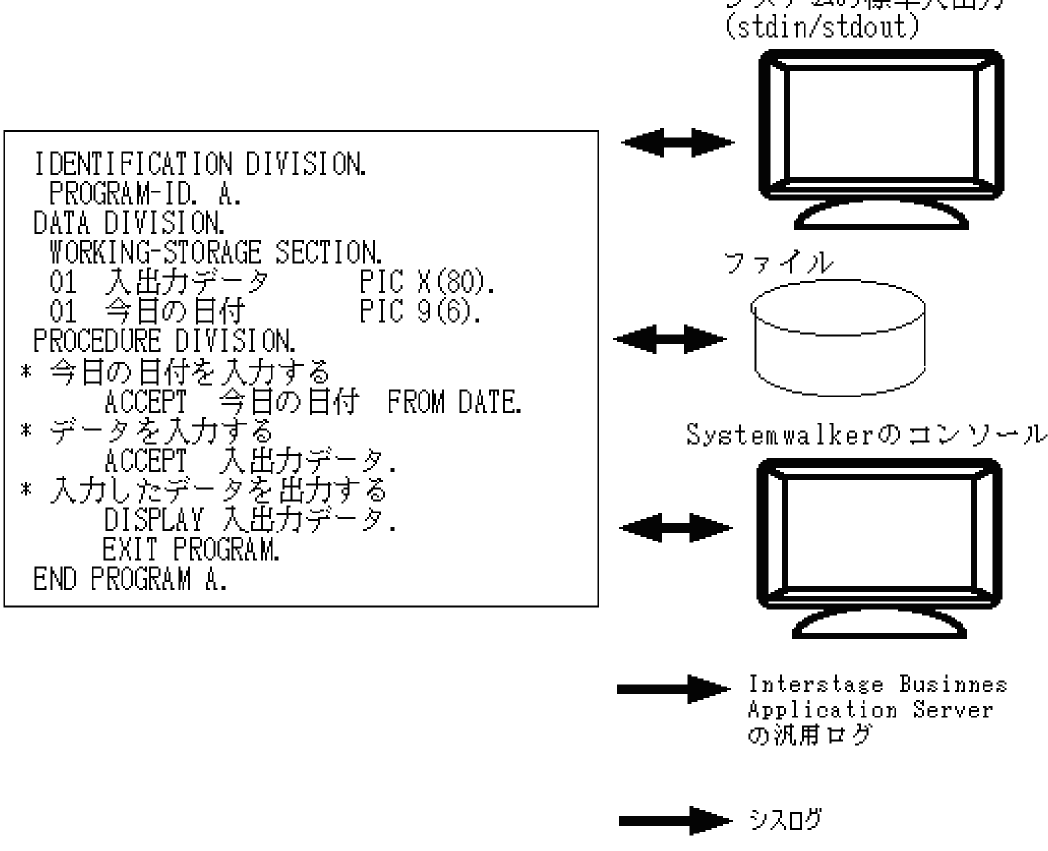
10.1 小入出力

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、小入出力について説明します。

なお、小入出力を使用した例題プログラムがサンプルとして提供されているので、参考にしてください。

10.1.1 概要

小入出力では、システムの標準入出力(stdin/stdout)、Systemwalker Centric Managerのコンソール、Interstage Business Application Serverの汎用ログ、シス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。



10.1.2 入出力先の種類と指定方法

小入出力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によって異なります。これらの指定と入出力先の関係を"表10.1 小入出力の入出力先"に示します。

表10.1 小入出力の入出力先

	FROM指定またはUPON指定の記述	指定する翻訳オプション	入出力先
1	なし または 機能名SYSIN/SYSOUTに対応付けた呼び名	なし	・ システムの標準入出力(stdin/stdout) ・ 環境変数CBR_DISPLAY_SYSOUT_OUTPUTが指定された場合、シスログ(注6,7)

	FROM指定またはUPON指定の記述	指定する翻訳オプション	入出力先
			・ 環境変数CBR_COMPOSER_SYSOUTが指定された場合、右辺に指定したログ定義ファイルで定義されている管理名の設定に従います(注3,4)
		SSIN(環境変数名) SSOUT(環境変数名)	・ プログラム実行時に環境変数名に設定したファイル(注1) ・ 環境変数CBR_DISPLAY_SYSOUT_OUTPUTが指定された場合、シスログ(注6,7) ・ 環境変数CBR_COMPOSER_SYSOUTが指定された場合、右辺に指定したログ定義ファイルで定義されている管理名の設定に従います(注3,4)
2	機能名SYSERRに対応付けた呼び名	なし	・ システムの標準エラー出力先(stderr) ・ 環境変数CBR_MESSOUTFILEが指定された場合、指定されたファイル ・ 環境変数CBR_DISPLAY_SYSERR_OUTPUTが指定された場合、シスログ(注6,7) ・ 環境変数CBR_COMPOSER_SYSERRが指定された場合、右辺に指定したログ定義ファイルで定義されている管理名の設定に従います(注4)
3	機能名CONSOLEに対応付けた呼び名	なし	・ システムの標準入出力先(stdin/stdout) (注2) ・ 環境変数CBR_DISPLAY_CONSOLE_OUTPUTが指定された場合、シスログ(注6,7) ・ 環境変数CBR_CONSOLEが有効な場合、Systemwalker Centric Managerのコンソール ・ 環境変数CBR_COMPOSER_CONSOLEが指定された場合、右辺に指定したログ定義ファイルで定義されている管理名の設定に従います(注3,5)

注1: 翻訳オプションSSIN/SSOUTに環境変数名としてSYSIN/SYSOUTを指定した場合、入出力先はシステムの標準入出力になります。

注2: システム標準入出力を入出力先にする場合、通常1を使用します。

注3: Interstage Business Application Serverの汎用ログはデータの入力を行うことはできません。

注4: 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5: Interstage Business Application Serverの汎用ログに出力する場合、システムの標準出力(stdout)には出力されません。CBR_CONSOLEを同時に指定している場合は汎用ログとSystemwalker Centric Managerのコンソールの両方に出力されます。

注6: シスログはデータの入力を行うことはできません。

注7: シス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の汎用ログまたはSystemwalker Centric Managerへの出力を同時に指定している場合は、シスログには出力されません。

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で、1と3の両方を使うことはできません。プログラム中で最初に実行したACCEPT文またはDISPLAY文に指定された指示が有効となります。

10.1.3 システムの標準入出力(stdin/stdout)を使うプログラム

ここでは、システムの標準入出力(stdin/stdout)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説明します。また、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。

10.1.3.1 プログラムの記述

ここでは、システムの標準入出力(stdin/stdout)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.
 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)

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。

手続き部(PROCEDURE DIVISION)

標準入力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、長さ分のデータを入力するまで入力要求が行われます。標準出力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。

10.1.3.2 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。

10.1.3.3 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てください。

プログラム中のACCEPT文が実行されると、標準入力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。また、プログラム中のDISPLAY文が実行されると、標準出力にデータが出力されます。

注意

標準出力(stdout)に出力するプログラムをバックグラウンドで実行する場合は、リダイレクション機能を使用してファイルに出力するか、ファイルに出力するプログラムに変更してください。プログラムを変更する場合は、"10.1.7 ファイルを使うプログラム"を参照してください。

10.1.3.4 数字データの入力

ACCEPT文のデータ受け取り項目(外部10進項目、内部10進項目および2進項目)に数字項目を記述することにより、数字データを入力することができます。

入力対象となる数字データは、ハードウェア装置の入力行の先頭から21桁以内で、復帰・改行キーまでを対象とします。入力する数字データの記述形式を以下に示します。

```
$ ［符号文字］数字桁
```

または、

```
$   数字桁［符号文字］
```

数字桁

数字(“0”～“9”)および小数点(“.”)が指定できます。入力データは、ACCEPT文に指定したデータ項目の形式に合わせて位取りを行い、格納されます。

符号文字

符号(“+”または“-”)が指定できます。

10.1.4 Systemwalkerのコンソールを使うプログラム

ここでは、Systemwalker Centric Managerのコンソール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説明します。また、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。

Systemwalker Centric Managerの概要、導入、使用方法については、Systemwalker Centric Mangerのマニュアルを参照してください。

10.1.4.1 プログラムの記述

ここでは、Systemwalker Centric Managerのコンソール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.
     ACCEPT   データ名 FROM 呼び名.
     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CONSOLEに呼び名を対応付けます。

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。

手続き部(PROCEDURE DIVISION)

Systemwalker Centric Managerのコンソールからデータを入力するには、FROM指定に機能名CONSOLEに対応付けた呼び名を指定した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、空白詰めが行われます。Systemwalker Centric Managerにデータを出力するには、UPON指定に機能名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。

10.1.4.2 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。

10.1.4.3 プログラムの実行

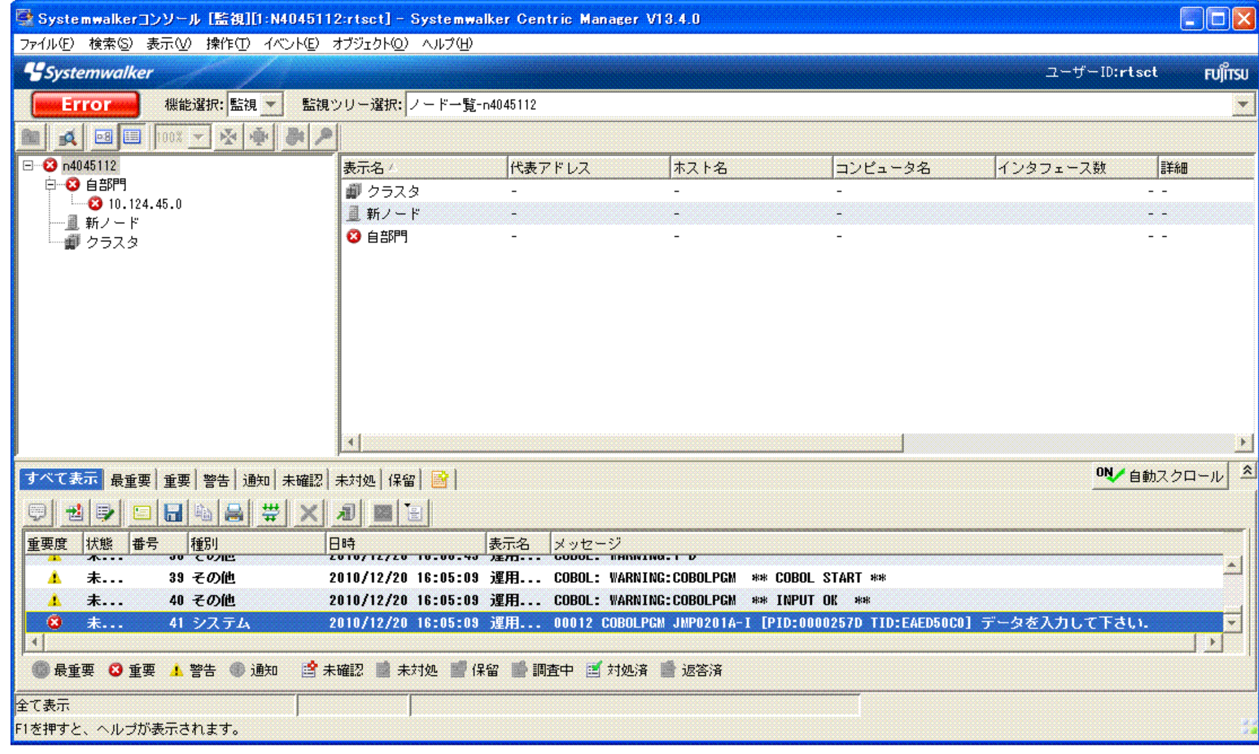
入出力先にSystemwalker Centric Managerのコンソールを使用するプログラムを実行するときには、環境変数CBR_CONSOLEの設定が必要です。

```
$ CBR_CONSOLE=CENTRICMANAGER ; export CBR_CONSOLE
```

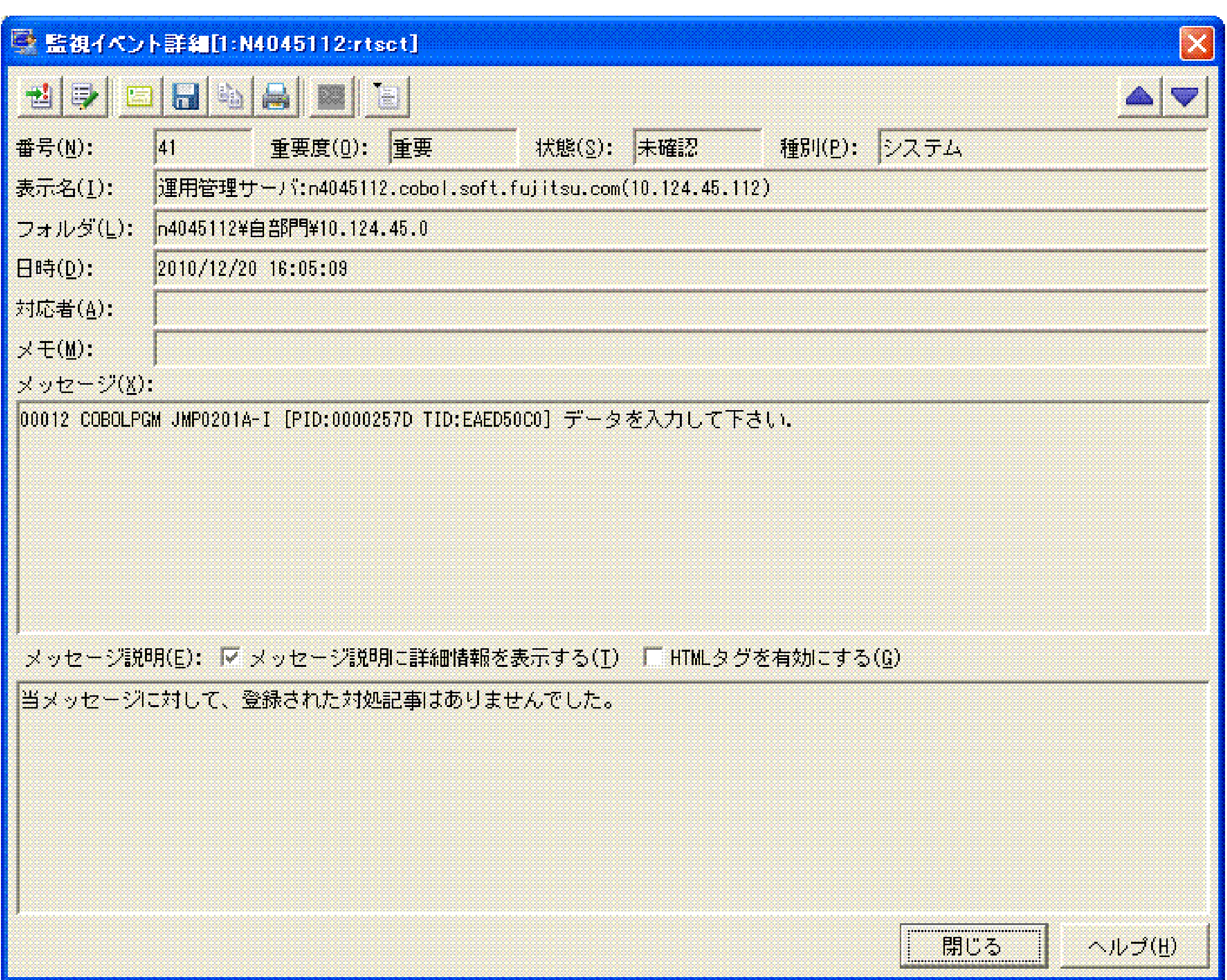
ACCEPT文の入力方法

- Systemwalker Centric Manager V13.4.0以降をお使いの場合

  1. ACCEPT文を実行すると、Systemwalker Centric Managerのコンソールに以下の画面が表示されて、返答待ちになります。

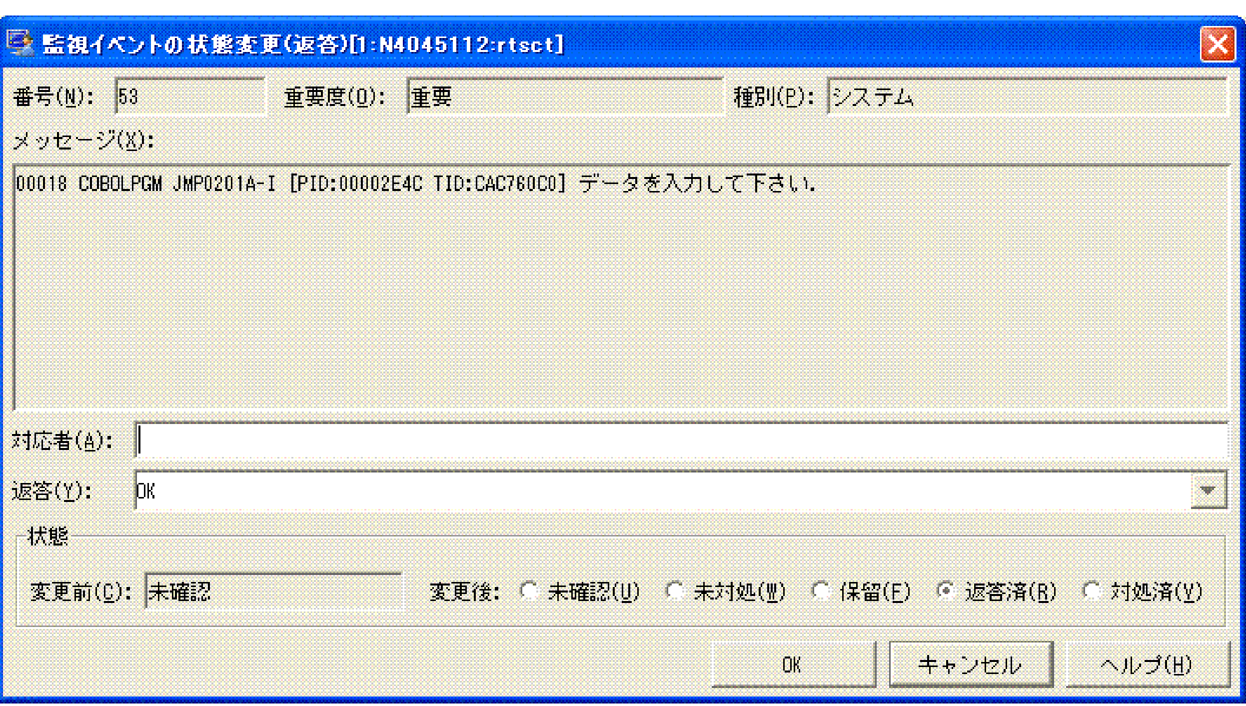


2. 青く表示されているメッセージをダブルクリックします。ダブルクリックすると、“監視イベント詳細”の画面が表示されます。

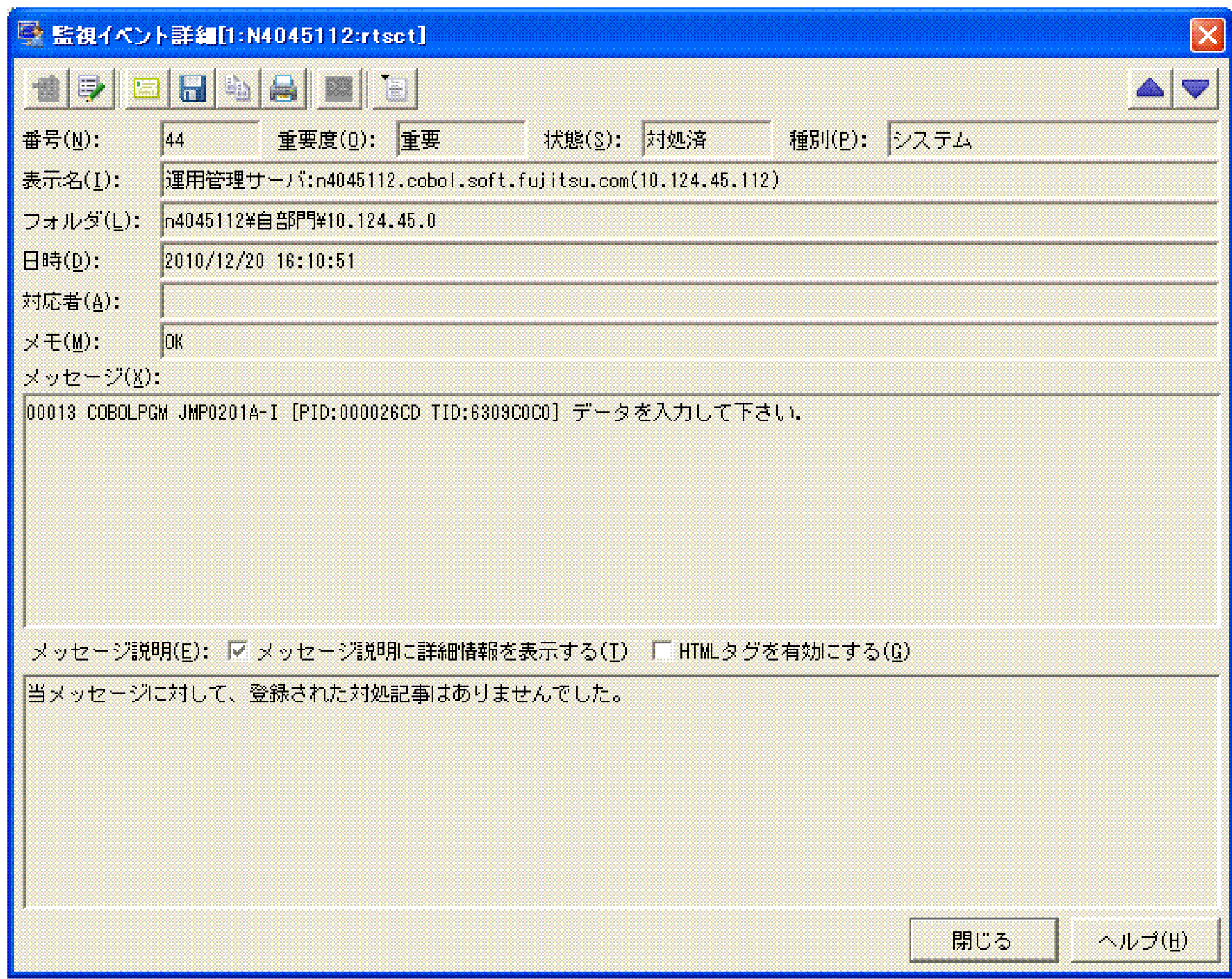


“監視イベント詳細”画面のツールバーで“イベントの状態変更”をクリックすると、“監視イベントの状態変更（返答）”の画面が表示されます。

3. “監視イベントの状態変更(返答)”の画面の“返答”フィールドにデータを入力します。続いて“状態”の“返答済”をチェックし、OKボタンをクリックします。

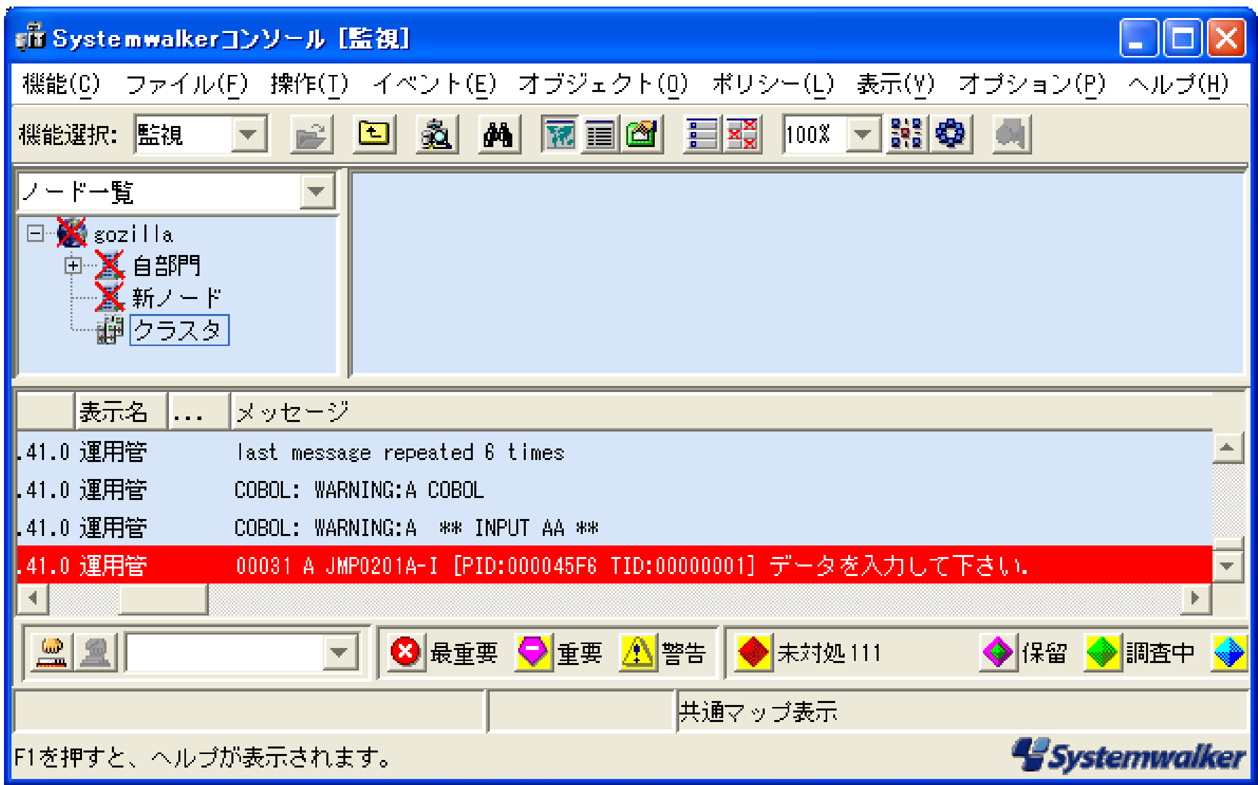


4. “監視イベント詳細”の画面に戻ったら、“閉じる”ボタンをクリックします。

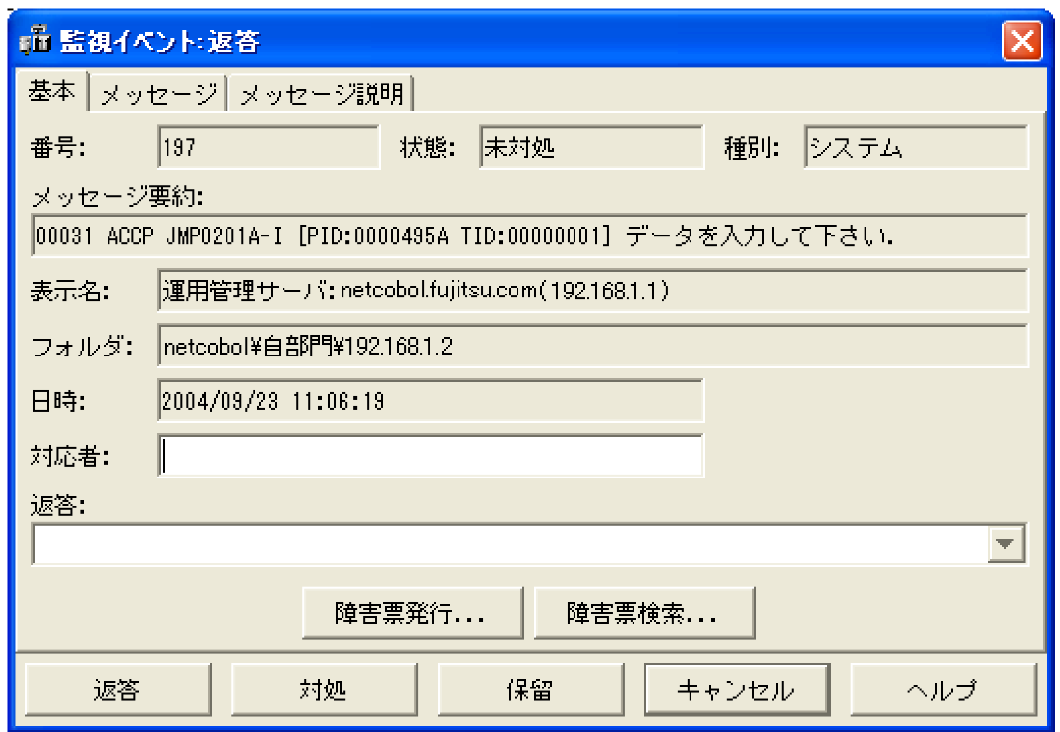


- Systemwalker Centric Manager V13.4.0より前のバージョンをお使いの場合

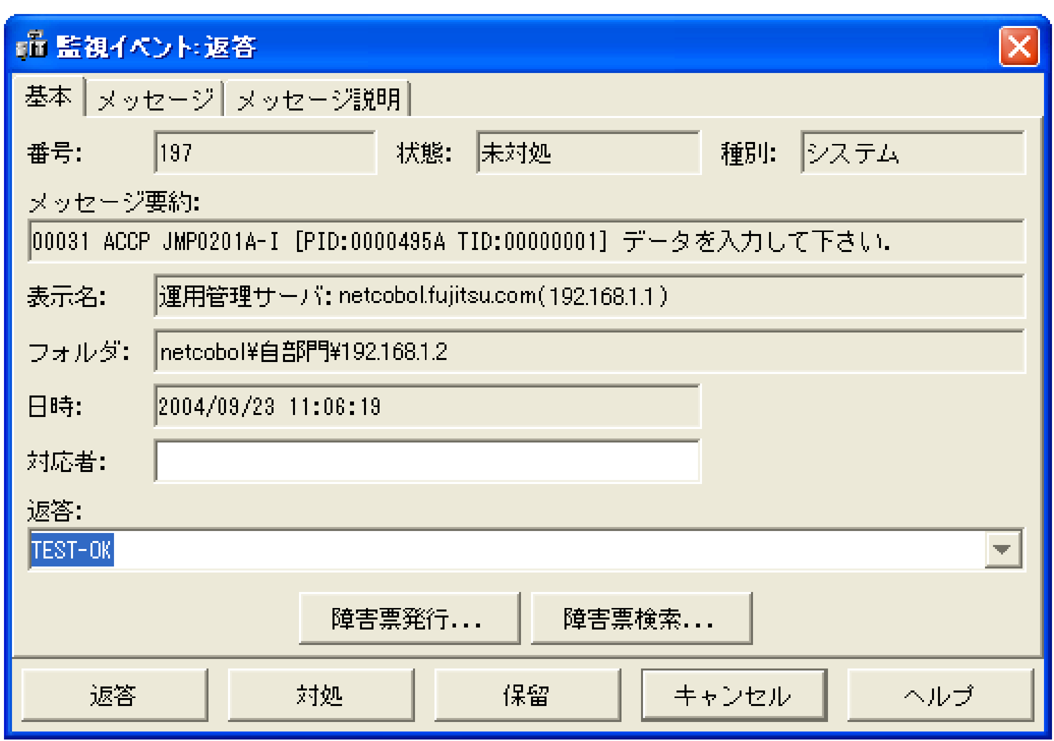
1. ACCEPT文を実行すると、Systemwalker Centric Managerのコンソールに以下のように表示されて返答待ちになります。



2. 赤く表示されているメッセージをダブルクリックします。ダブルクリックすると、"監視イベント:返答"画面が表示されます。



3. “監視イベント:返答”画面の“返答”フィールドにデータを入力します。



4. データを入力したら、返答ボタンをクリックします。

注意

- 動作モードがUnicodeの場合、Systemwalker Centric Managerのコンソールは使用できません。
- DISPLAY文で一度に出力できるデータは1023バイトです。
- DISPLAY文の出力形式について

  以下の形式で出力されます。

```
COBOL: WARNING:プログラム名 DISPLAY文に指定されたデータ
```

  WARNINGと出力されますが、動作には影響ありません。また、プログラム名は、実行単位を開設したプログラム名です。

- ACCEPT文で一度に入力できるデータは124バイトです。
- ACCEPT文で出力されるメッセージの表示形式について

  実行時メッセージ(JMP0201A-I)が以下の形式で表示されます。

```
nnnnn プログラム名 実行時メッセージ
```

  nnnnn(5桁の数字)はSystemwalker Centric Managerが設定します。プログラム名は、実行単位を開設したプログラム名です。

- Systemwalker Centric Managerのコンソールに以下のメッセージが表示された場合、コード変換エラーが発生したことを示します。

```
Invalid character code. Text is deleted.
```

  上記エラーが発生した場合、データの入出力は行われず、DISPLAY/ACCEPT文は正常終了します。入出力データや長さを確認してください。

10.1.4.4 数字データの入力

ACCEPT文のデータ受け取り項目に数字項目(外部10進項目、内部10進項目および2進項目)を記述することにより、数字データを入力することができます。

入力対象となる数字データは、ハードウェア装置の入力行の先頭から21桁以内で、復帰・改行キーまでを対象とします。入力する数字データの記述形式を以下に示します。

```
[符号文字] 数字桁
```

または、

```
数字桁 [符号文字]
```

数字桁

数字(“0” ～“9”)および小数点(“.”)が指定できます。入力データは、ACCEPT文に指定したデータ項目の形式に合わせて位取りを行い、格納されます。

符号文字

符号(“+”または“-”)が指定できます。

10.1.5 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。

10.1.5.1 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。

10.1.5.2 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。

10.1.5.3 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 ; export CBR_COMPOSER_SYSOUT
```

- 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 ; export CBR_COMPOSER_SYSERR
```

- 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 ; export CBR_COMPOSER_CONSOLE
```

注意

DISPLAY文の汎用ログへの出力レベルは5です。

10.1.6 システムの標準エラー出力(stderr)を使うプログラム

ここでは、システムの標準エラー出力(stderr)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。

10.1.6.1 プログラムの記述

ここでは、システムの標準エラー出力(stderr)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.
     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。

手続き部(PROCEDURE DIVISION)

標準エラー出力にデータを出力するには、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、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。

10.1.6.2 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。

10.1.6.3 プログラムの実行

通常のプログラムを実行するときと同様に実行します。

プログラム中のDISPLAY文が実行されると、標準エラー出力にデータが出力されます。

データをファイルに出力する場合、環境変数CBR_MESSOUTFILEを指定します。[参照]"4.3 実行時メッセージの出力方法の指定"

10.1.7 ファイルを使うプログラム

ここでは、小入出力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。

10.1.7.1 プログラムの記述

ここでは、小入出力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)

特に必要な記述はありません。

データ部(DATA DIVISION)

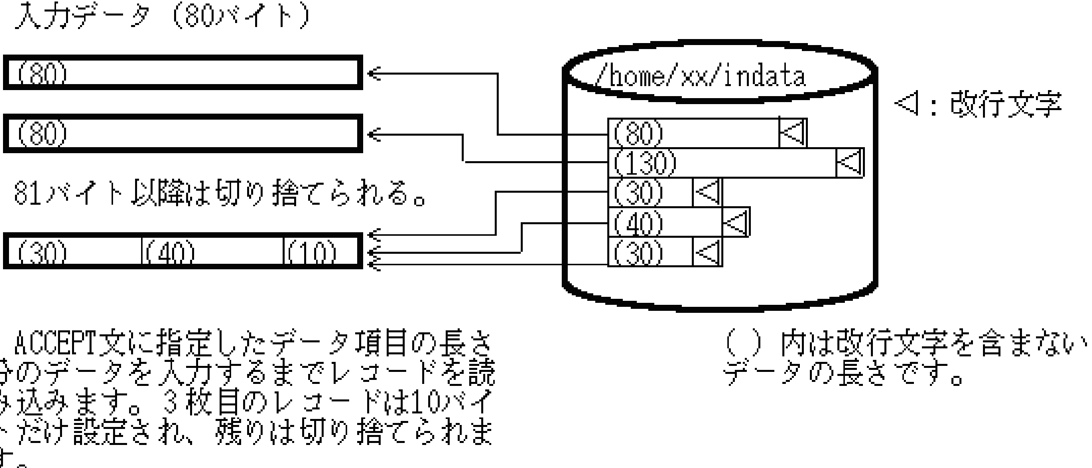
入力したデータを格納するためのデータ項目および出力するデータを設定するためのデータ項目を定義します。

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。

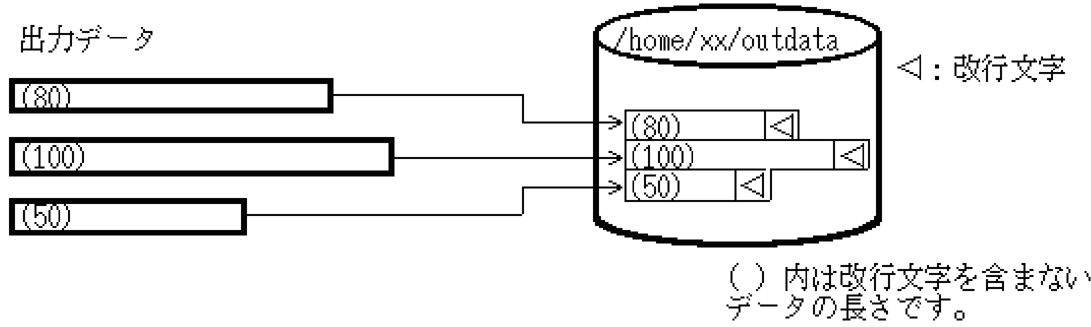
データの入力

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。長さ分のデータを入力す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。



10.1.7.2 プログラムの翻訳・リンク

小入出力のデータの入力先をファイルにする場合には、翻訳オプションSSINを指定します。[参照]"A.2.40 SSIN（ACCEPT文のデータの入力先）"

また、データの出力先をファイルにする場合には、翻訳オプションSSOUTを指定します。[参照]"A.2.41 SSOUT（DISPLAY文のデータの出力先）"

例

```
$ cobol -o PROG1 -WC,"SSIN(IN),SSOUT(OUT)" -M PROG1.cob
```

注意

- 翻訳オプションSSINに環境変数名としてSYSINを指定した場合、ACCEPT文のデータ入力先は、環境変数SYSINの設定にかかわらずシステムの標準入力(stdin)となります。
- 翻訳オプションSSOUTに環境変数名としてSYSOUTを指定した場合、DISPLAY文のデータ出力先は、環境変数SYSOUTの指定にかかわらずシステムの標準出力(stdout)になります。
- 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、システムの標準入出力(stdin/stdout)になります。

10.1.7.3 プログラムの実行

プログラムを実行する前に、翻訳オプションSSINおよびSSOUTに指定した環境変数名に、入出力処理を行うファイルの名前を設定します。

例

```
$ IN=/home/xx/indata ; export IN
$ OUT=/home/xx/outdata ; export OUT
```

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6.9 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(/dev/nullなど)を使用することはできません。

10.1.7.4 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
- ファイルの最大サイズ制限解除
- ダミーファイル

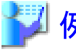
以下に拡張機能とその使い方について説明します。

ファイル追加書き

DISPLAY文のファイル出力で、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

```
$ OUT=/home/xx/outdata,MOD ; export OUT
```

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。

ファイルの最大サイズ制限解除

DISPLAY文のファイル出力において、ファイルの最大サイズである1Gバイトの制限を解除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、“,NOLIMIT”を指定します。

例

```
$ OUT=/home/xx/outdata,NOLIMIT ; export OUT
```

“NOLIMIT”を指定した場合、1Gバイトを超えるファイルを扱うことができます。ファイルの最大サイズについては、“表6.9 各ファイルシステムの機能差”の大容量ファイルを参照してください。

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、“6.8.9 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

```
$ OUT=/home/xx/outdata,DUMMY ; export OUT
```

または

```
$ OUT=,DUMMY ; export OUT
```

- “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”、“,NOLIMIT”および“,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”および“,NOLIMIT”を指定しても、“,MOD”および“,NOLIMIT”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”、“NOLIMIT”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。

10.1.7.5 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、"6.8.9 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

```
$ IN=/home/xx/indata,DUMMY ; export IN
```

または

```
$ IN=,DUMMY ; export IN
```

注意

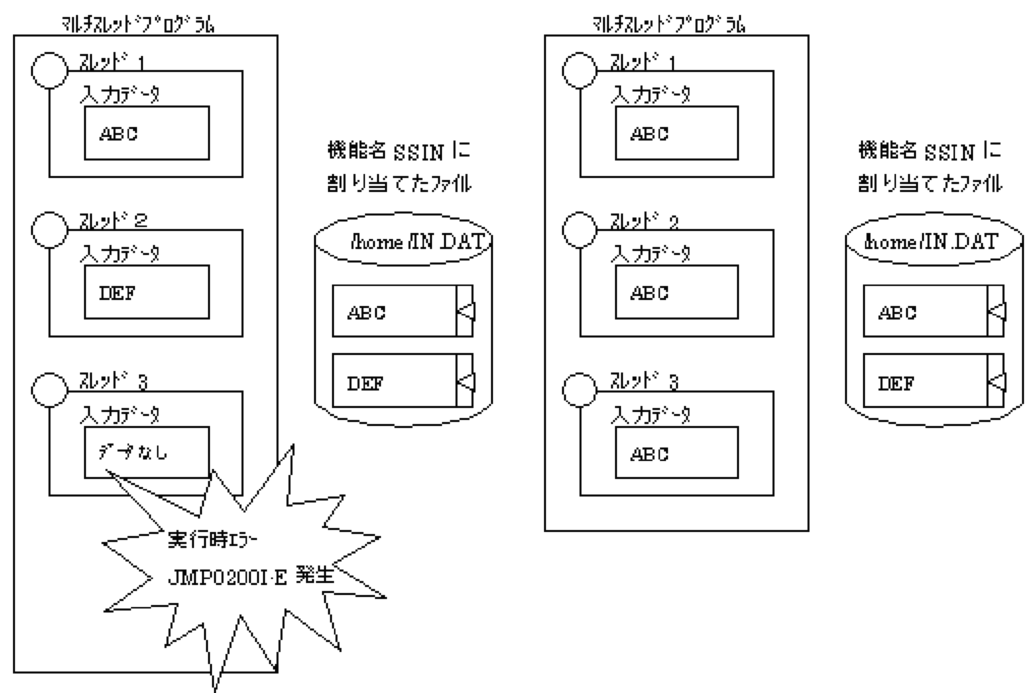
- ファイル名にコンマ(,)を含む場合、ファイル名を二重引用符(")で囲む必要があります。
- 文字列"DUMMY"の前にコンマ(,)が必要です。コンマ(,)がない場合、文字列"DUMMY"をファイル名とみなして、通常のファイルと同じ動作をします。
- ファイル名の指定の有無による動作の違いはありません。指定したファイル名は意味を持ちません。
- コンマ(,)に続き、"DUMMY"以外の文字列を指定した場合、最初のACCEPT文の実行でエラーになります。

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。

例

```
$ CBR_SSIN_FILE=THREAD ; export CBR_SSIN_FILE
```

10.1.8 現在の日付および時刻の入力

ここでは、小入出力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。

10.1.8.1 プログラムの記述

ここでは、小入出力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)

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)

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。

10.1.8.2 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。

10.1.8.3 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

10.1.9 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。

10.1.9.1 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。

10.1.9.2 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。

10.1.9.3 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、環境変数CBR_JOBDATEの設定が必要です。

```
$ CBR_JOBDATE=年.月.日; export CBR_JOBDATE
```

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

10.1.10 シスログを使うプログラム

ここではシス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。

10.1.10.1 プログラムの記述

ここでは、シス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)

シス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。

10.1.10.2 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。

10.1.10.3 プログラムの実行

出力先にシス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の出力先をシスログにする場合

```
CBR_DISPLAY_SYSOUT_OUTPUT=SYSLOG
```

機能名SYSERRの出力先をシスログにする場合

```
CBR_DISPLAY_SYSERR_OUTPUT=SYSLOG
```

機能名CONSOLEの出力先をシスログにする場合

```
CBR_DISPLAY_CONSOLE_OUTPUT=SYSLOG
```

注意

- Syslogの仕様により、一度に出力できるデータは、Syslogのヘッダー部とCOBOLで指定したデータを合わせて、1024バイトまでです。
- 出力できるデータのコード系は1バイトコード系です。多バイトコード系の出力は保証しません。
- Interstage Business Application Serverの汎用ログ、Systemwalker Centric Managerと同時に使用することはできません。
- シスログに出力する場合、他の出力先(システムの標準出力先、ファイル等)の指定は無効となり出力されません。
- シスログ出力時のアイデンティティ名は“NetCOBOL Application”となります。アイデンティティ名を変更する場合は、DISPLAY文のUPON指定ごとに以下の環境変数にアイデンティティ名を設定してください。

  UPON指定なしまたは機能名SYSOUTのアイデンティティ名を変更する場合

  ```
  CBR_DISPLAY_SYSOUT_SYSLOG_IDENT=アイデンティティ名
  ```

  機能名SYSERRのアイデンティティ名を変更する場合

  ```
  CBR_DISPLAY_SYSERR_SYSLOG_IDENT=アイデンティティ名
  ```

  機能名CONSOLEのアイデンティティ名を変更する場合

  ```
  CBR_DISPLAY_CONSOLE_SYSLOG_IDENT=アイデンティティ名
  ```

- シスログ出力時のレベルは、“インフォメーションメッセージ”となります。レベル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のレベルを変更する場合

```
CBR_DISPLAY_SYSOUT_SYSLOG_LEVEL = { I | W | E }
```

  機能名SYSERRのレベルを変更する場合

```
CBR_DISPLAY_SYSERR_SYSLOG_LEVEL = { I | W | E }
```

  機能名CONSOLEのレベルを変更する場合

```
CBR_DISPLAY_CONSOLE_SYSLOG_LEVEL = { I | W | E }
```

  レベルは以下を意味します。

  - I : インフォメーションメッセージ
  - W : ワーニングの状態
  - E : エラーの状態

  シスログの運用によっては、facilityおよびlevelパラメタによって出力を抑止したり、あて先を変更したりすることができます。環境変数の指定どおりにメッセージが出力されない場合、シスログの設定(syslog.conf)を確認してください。

例

アイデンティティ名に"APPLTEST"、レベルに"E"、DISPLAY文のデータに"APPL01:START"を指定した場合の出力例

```
DEC 11 14:43:37 sol APPLTEST[22038] : [ID 659399 user.error] APPL01:START
```

10.2 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。

10.2.1 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、引数の値を参照したり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。

10.2.2 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

例

データ項目の定義例

```
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。引数の位置は、コマンド名を0、コマンドの次に指定された引数を1とし、その次から2、3、…と順次数えます。DISPLAY文にデータ名を指定した場合にはそのデータ名に設定された値が、数字定数を指定した場合にはその数字が、次に値を参照する引数の位置となります。参照する引数の位置付けを行ったら、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を指定した(引数の数が3つなのに引数の位置に4を指定するなど)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の指定がある場合にはそこに記述された文([3])が実行されます。ACCEPT文にON EXCEPTIONの指定がない場合には、エラーメッセージを出力し、ACCEPT文の次の文を実行します。

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

注意

― 位置付けのためのDISPLAY文を実行しないで引数の値を参照した場合、プログラム実行開始時に引数の位置は1に位置付けられ、その後ACCEPT文を実行するごとに、次の引数に位置付けられます。

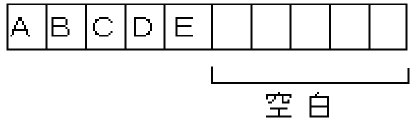
― 引数の値の長さを得ることはできません。

― 引数の数および値のデータ項目への設定は、COBOLのMOVE文の規則が適用されます。

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

コマンドに指定した引数の数が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。

10.2.3 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。

10.2.4 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。

10.3 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。

10.3.1 概要

プログラムの実行中に、DISPLAY文とACCEPT文を使用することにより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場合には、以下の文を使用します。
  - 機能名ENVIRONMENT-NAMEに対応付けた呼び名を使用したDISPLAY文
  - 機能名ENVIRONMENT-VALUEに対応付けた呼び名を使用したACCEPT文
- 環境変数の値を更新する場合には、以下の文を使用します。
  - 機能名ENVIRONMENT-NAMEに対応付けた呼び名を使用したDISPLAY文
  - 機能名ENVIRONMENT-VALUEに対応付けた呼び名を使用したDISPLAY文

10.3.2 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

```
ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SPECIAL-NAMES.
     ENVIRONMENT-NAME  IS   呼び名－環境変数名
     ENVIRONMENT-VALUE IS   呼び名－環境変数の値.
```

データ部(DATA DIVISION)

値の受け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。参照する環境変数名は、文字定数によって直接DISPLAY文に指定する方法と、環境変数名をデータ項目に設定し、そのデータ名をDISPLAY文に指定する方法があります。参照する環境変数名を指定したら、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未指定の場合や、存在しない環境変数名を指定した場合、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の指定がある場合には、そこに指

定した文([3])が実行されます。ACCEPT文にON EXCEPTIONの指定がない場合には、エラーメッセージを出力し、ACCEPT文の次の文を実行します。

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。更新する環境変数名の指定方法は、前述した環境変数の値を参照するときと同様です。更新する環境変数名を指定したら、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未指定の場合や、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の指定がある場合にはそこに指定された文([6])が実行されます。DISPLAY文にON EXCEPTIONの指定がない場合には、エラーメッセージを出力し、DISPLAY文の次の文を実行します。

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

注意

環境変数の値の長さを得ることはできません。

10.3.3 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。

10.3.4 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中でだけ有効となり、プロセス終了後のプログラムに対しては、有効となりません。

第11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。

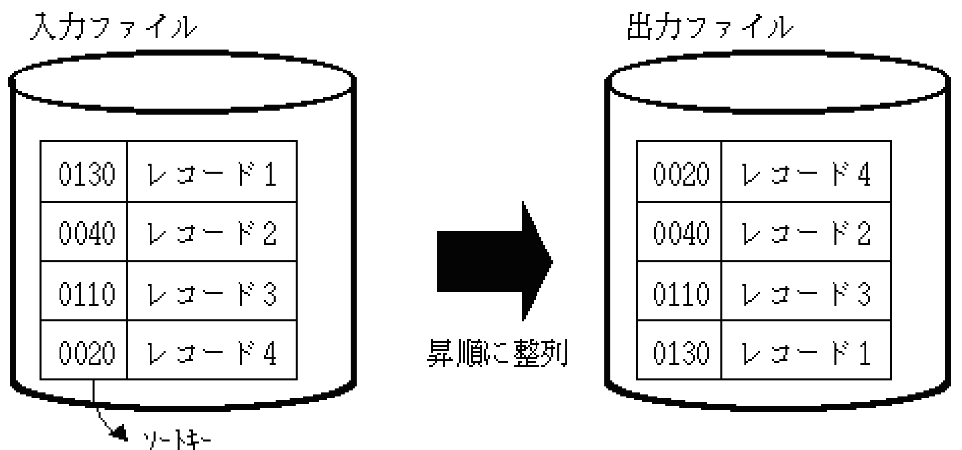
本章では、ソートマージ(整列併合)機能について説明します。

11.1 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。

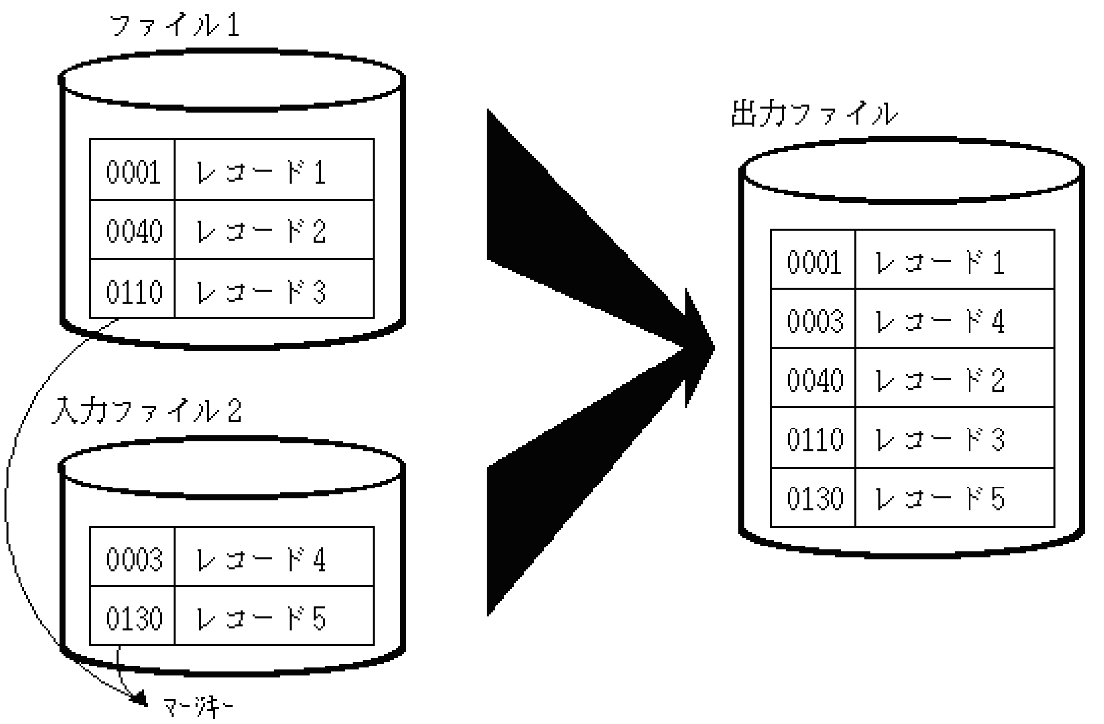
ソート

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。



11.2 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。

11.2.1 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。

11.2.2 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.
入力手続き１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.
出力手続き１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。

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。

注意

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。

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。"表11.1 SORT-STATUSに設定される値と意味"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11.1 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

入力ファイルおよび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オープンさ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。

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。

11.2.3 プログラムの翻訳・リンク

必要に応じて翻訳オプションEQUALSを指定します。[参照]"A.2.12 EQUALS（SORT文での同一キーデータの処理方法）"

EQUALSは、ソート処理で同じ値のソートキーをもつ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入力したレコードの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。

11.2.4 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、作業用ファイルが必要です。作業用ファイルは、環境変数BSORT_TMPDIRに指定したディレクトリ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、/tmpのディレクトリに一時的に作成されます。

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

ソート処理は、通常、COBOLランタイムシステムを使用して処理を行います。しかし、PowerSORTをインストールした場合、PowerSORTを使用します。

COBOLで作成した大容量ファイルを整列併合機能で使用するときにはPowerSORTが必要です。

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で指定されたディレクトリ
2. スタートアップファイル(注)のBSORT_TMPDIRで指定されたディレクトリ
3. 環境変数TMPDIRで指定されたディレクトリ
4. システム標準のディレクトリ(/var/tmp)

   注 ： スタートアップファイルは、PowerSORTの省略値を定義するファイルです。詳細は"PowerSORT ユーザーズガイド"を参照してください。

11.3 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。

11.3.1 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
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。

11.3.2 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.
 出力手続き１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。
入力ファイル	マージ処理の対象となるファイルをすべて定義します。
出力ファイル	マージ処理の結果をファイルに出力する場合に定義します。

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。

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、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。“表11.2 SORT-STATUSに設定される値と意味”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11.2 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出力ファイルを使用する場合、MERGE文実行時にそれらのファイルがオープンさ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。

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

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

この特殊レジスタは、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。

11.3.3 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。

11.3.4 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。

参考

ソート処理は、通常、COBOLランタイムシステムを使用して処理を行います。しかし、PowerSORTをインストールした場合、PowerSORTを使用します。

COBOLで作成した大容量ファイルを整列併合機能で使用するときにはPowerSORTが必要です。

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で指定されたディレクトリ
2. スタートアップファイル(注)のBSORT_TMPDIRで指定されたディレクトリ
3. 環境変数TMPDIRで指定されたディレクトリ
4. システム標準のディレクトリ(/var/tmp)

   注 ： スタートアップファイルは、PowerSORTの省略値を定義するファイルです。詳細は"PowerSORT ユーザーズガイド"を参照してください。

第12章　システムプログラムを記述するための機能

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。

12.1 SD機能の種類

この製品には、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。これらの機能をこの製品では、システムプログラム記述向け機能(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・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を領域アドレスとします。その場合、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

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

12.2 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。

12.2.1 概要

ポインタ付けは、ある特定のアドレスをも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もつ領域の内容となります。

12.2.2 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)

特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・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。

12.2.3 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。

12.2.4 プログラムの実行

特に必要な環境設定はありません。

12.3 ADDR関数とLENG関数の使い方

ここでは、ADDR関数とLENG関数の使い方について説明します。

12.3.1 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。

12.3.2 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)

特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。

12.3.3 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。

12.3.4 プログラムの実行

特に必要な環境設定はありません。

12.4 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。

12.4.1 概要

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

12.4.2 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)

特に必要な記述はありません。

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)

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

12.4.3 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。

12.4.4 プログラムの実行

特に必要な環境設定はありません。

第13章　オブジェクト指向プログラミングとは

本章では、オブジェクト指向プログラミングについて、簡単に紹介します。

はじめに、オブジェクト指向プログラミングが誕生した背景について、従来のプログラミング技法の問題点を挙げながら説明します。

次に、オブジェクト指向の基本的な概念について、従来のプログラミング技法と比較しながら説明します。

13.1 オブジェクト指向の歴史的背景

1950年代の終わりに最初のプログラミング言語FORTRANが誕生してから、すでに30年以上経っています。その間、多くのプログラミング言語が誕生し、消えていきました。そして、よりよいプログラムを書くために、さまざまなプログラミングスタイルが考案されてきました。

“プログラミングスタイル”とは、よりよいプログラムを書くための約束ごとです。構造化プログラミングをはじめ、これまで多くのプログラミングスタイルが提唱されてきました。オブジェクト指向も、こうしたプログラミングスタイルの1つです。

ここでは、オブジェクト指向が生まれた背景として、プログラミングスタイルの変遷について説明します。

13.1.1 ソフトウェア工学以前

1950年ごろ作られた、世界最初のプログラム内蔵式コンピュータEDSACでは、機械語でプログラムが書かれていました。初期のプログラマたちは、限られたメモリと少ない命令で、芸術的ともいえるコードを書いていました。

しかし、技術の発展とコンピュータの普及にともない、より多くの、より大規模のソフトウェアが要求されるようになりました。1950年代の終わりに、最初の高級言語であるFORTRANが登場しました。これにより、プログラムの生産性は飛躍的に向上しました。しかし、それ以上に、ソフトウェアに対する要求は複雑になり、また、コンピュータによるシステム化の要求も増大しました。これらの要求に、高級言語の出現だけでは対応しきれませんでした。そして、その対策として“ソフトウェア工学”が誕生しました。

13.1.2 構造化プログラミングの出現

ソフトウェア工学の最初の成果が、1970年代始めに現れた構造化プログラミングです。これは、プログラムの制御の流れとデータ構造を、わかりやすく記述するための手法です。

プログラム制御の構造化は、プログラムは以下の3つの構造を使って表現できるという“構造化定理”に基づいています。

- 連接処理(文の並び)
- 選択処理(IF文、EVALUATE文に相当)
- 繰返し処理(うちPERFORM文に相当)

GO TO文を多用すると、全体の流れがわかりにくいプログラムになってしまいます。しかし、これら3つの構造の組合せで記述すれば、わかりやすいプログラムを書くことができます。

また、データ構造についても同じです。従来は、個々のデータを別々に扱うしかありませんでした。しかし、データに構造を持たせることにより、複数のデータをまとめて1つのデータとして扱えるようになりました。データ構造も、基本的には以下の3つの構造の組合せです。

- 連接(集団項目)
- 選択(REDEFINES句)
- 繰返し(OCCURS句)

これらの考えは、COBOLにも取り入れられています。

13.1.3 モジュール化

1970年代半ばには、プログラム開発の規模はますます大きくなりました。そのため、モジュール化という方法が考え出されました。

モジュール化とは、プログラムを独立性の高いモジュールに分割し、さらにそのモジュールを、より小さなモジュールに分割する手法です。この手法を使うと、プログラムの規模が大きくても、全体の見通しは非常によくなります。これにより、大規模プログラムの共同開発が可能になりました。

モジュール化は、見方を変えれば、プログラムの持つ機能を抽象化することです。プログラミングする対象を分析し、それを抽象的な機能単位に分けます。もちろん、内部のコードがどうなっているのか意識する必要はありません。これを段階的に繰り返すと、全体的に見通しのよいプログラムを作ることができます。

13.1.4 抽象データ型

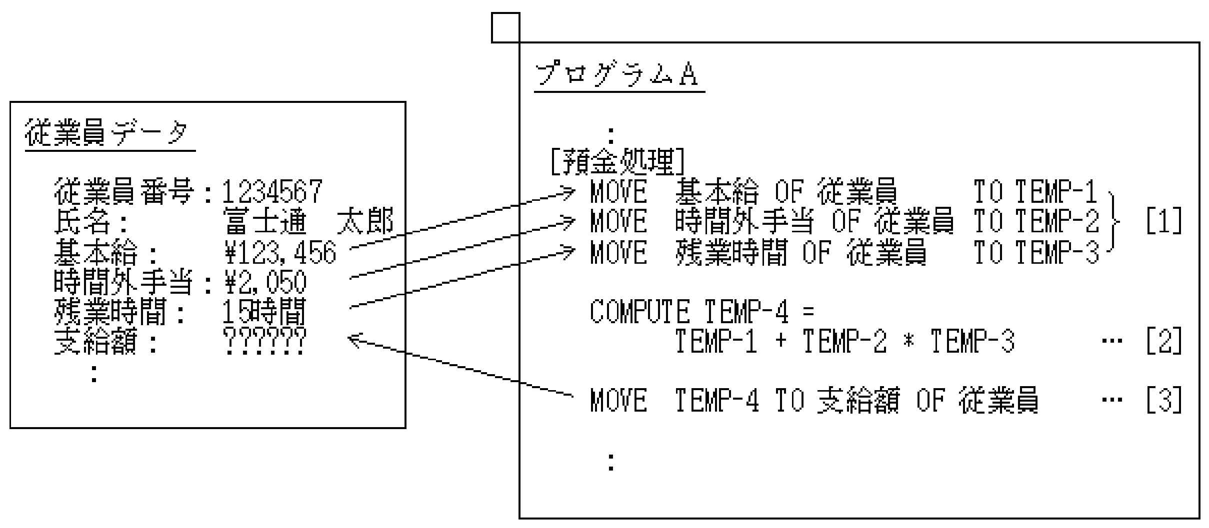
1970年代後半になると、開発済みのプログラムの保守作業の負荷が増大しました。それを軽減するために、抽象データ型という考え方が現れました。

モジュール化は手続きの抽象化でした。しかし、抽象データ型はデータの抽象化にあたります。

従来、プログラムで扱うデータは、数値、文字列などのコンピュータで表現しやすいものが中心でした。しかし、プログラムの対象となる現実の世界は、単純ではありません。たとえば、従業員管理システムで従業員データをモデル化しようとすると、従業員番号、氏名、住所、基本給など、さまざまな情報が集まった複合的なデータになります。データの構造化により、そのようなデータを定義できるようになりました。しかし、データ操作は、あいかわらず個々のデータごとに行っていました。たとえば、ある従業員の給与を計算するコードを書く場合、以下のようになります。

1. 従業員データから基本給、時間外手当および残業時間を取り出します。
2. 取り出したデータを基に、総支給額を計算します。
3. 計算した金額を、従業員データの支給額フィールドに書き込みます。

以下の図に示すように、プログラムAから従業員データの中の“基本給”、“時間外手当”、“残業時間”および“支給額”に直接アクセスします。



しかし、この方法には以下の問題があります。

- そのデータを扱うすべてのプログラムは、データの構造を知っている必要があります。
- データ内の個々の情報へのアクセスは制限されていないので、誤った操作により破壊される危険があります。
- データの構造を変更した場合、そのデータを利用していたモジュールをすべて変更する必要があります。

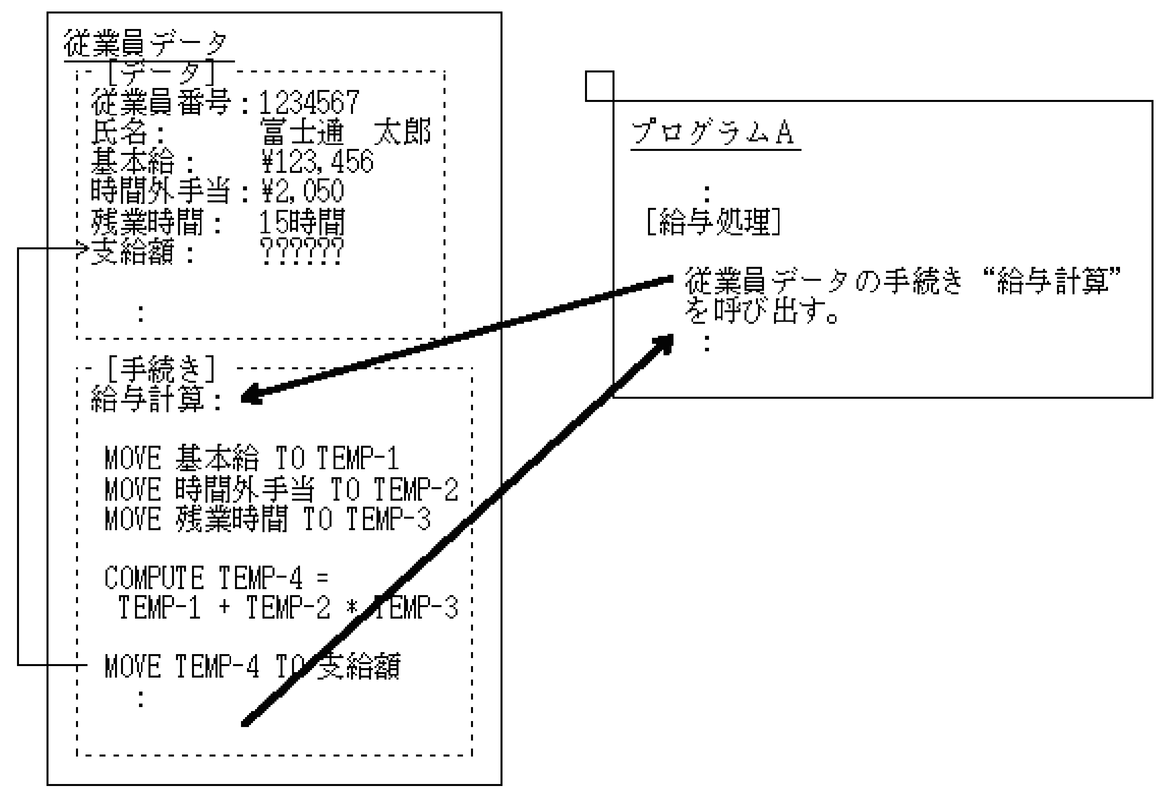
そこで登場したのが、抽象データ型という考えです。

抽象データ型では、現実世界の“物”の持つ性質を、抽象的なデータ型としてモデル化します。抽象データ型には、以下の特徴があります。

- 内部にデータと、それにアクセスする手続きのコードを持ちます。
- 内部のデータの手続きコードは、外部からは全く見えません。
- データをアクセスするためには、あらかじめ用意された手続きを呼び出す必要があります。
- 外部(そのデータを使用するプログラム)に公開されているのは、手続きのインタフェースだけです。

つまり、内部のデータおよびプログラム構造は外部から完全に隠されており、外部仕様である手続きのインタフェースを通じてだけアクセスできます。このような考え方を「情報隠蔽」といいます。また、データと手続きを一体化することを「カプセル化」といいます。

以下の図は、従業員データの中に“給与計算”という手続きを持たせています。プログラムから見えるのは“給与計算”手続きのため、これを呼び出して、給与関連の情報にアクセスします。また、“給与計算”という手続きは従業員データの一部として管理されます。



このような構成にしておくと、それぞれのデータ型との接点は公開されたインタフェースだけになります。そのため、内部の実現方法(たとえば、内部のデータ構造や手続きのコード)を変更しても、インタフェースが同じであれば、それを利用するプログラムには影響ありません。つまり、変更に対する影響を局所化でき、保守作業の負荷を減らせます。また、公開されているのはインタフェースだけなので、あらかじめ決められた正しい方法以外ではアクセスできません。そのため、自然にプログラムの信頼性が向上します。

13.1.5 オブジェクト指向の誕生

1980年ごろに、抽象データ型の延長として現れたのがオブジェクト指向です。

このころになると、ソフトウェア危機がより深刻になり、プログラムの飛躍的生産性向上が重要な課題になっていました。それに対する最も効果的な方法は、“プログラムを作らない”ことです。つまり、あらかじめプログラムを部品として用意しておき、それらを組み合わせるだけで新しいプログラムを作るということです。ICやLSIの出現で、家電製品の製造コストが下がったのと同じ論理です。

これまでも、モジュール化、抽象データ型により部品化が試みられてきました。しかし、これらの手法による部品化には、以下の欠点がありました。

- 再利用しにくい
- 柔軟性に欠ける

たとえば、既存の従業員管理システムに新しい勤務形態(たとえば年俸制)を加えるとします。しかし、年俸制の従業員であっても、ほとんどの部分は既存の一般従業員と同じ処理が使えます。それにもかかわらず、年俸制従業員に対しても、一般従業員と同じ処理を定義する必要があります。たとえば、両者の住所変更処理が同じであっても、それぞれの手続きを書く必要がありました。

また、一般従業員と管理職という2つの従業員データがあったとします。従来の方法では、これらを同時に扱う処理を作るのは困難でした。これらのデータは似ています。しかし、一般従業員用の処理と管理職用の処理は別々に作る必要がありました。さらに、これに年俸制従業員が加わると、もう1つ処理を作る必要があります。

オブジェクト指向では、前者は継承により、後者は多態と動的束縛により解決しました。これらの概念については、次節以降で詳しく説明します。

代表的なオブジェクト指向言語には、Smalltalk、C++などがあります。そして、最近では、COBOLでも取り入れられています。

13.2 オブジェクト指向の基本的な考え方

ここでは、オブジェクト指向の基本的な概念である、オブジェクト、クラス、メソッド、継承、多態および動的束縛について説明します。なお、ここでは概念についてだけ説明するため、プログラムの具体的な書き方については、“第14章 オブジェクト指向プログラミング機能～基本的な使い方～”を参照してください。

13.2.1 オブジェクトとクラス

以下に、オブジェクトおよびクラスについて説明します。

オブジェクト

オブジェクト指向では、プログラムで扱う“もの”はすべてオブジェクトと呼びます。それは、実在する“もの”であったり、概念上の“もの”であったりします。オブジェクト指向プログラミングとは、すべてオブジェクトを中心にしてプログラムを作ろうという試みです。

プログラム上では、オブジェクトは“データ”と“手続き”をカプセル化したものとして表現されます。データは、オブジェクトの中に隠蔽されており、公開されたインタフェースを通してしかアクセスできません。これは、抽象データ型の考え方そのものであり、その長所(保守の容易さ、高い信頼性)をすべて引き継いでいます。

また、手続きは“メソッド”と呼ばれます。本章でも、これ以降の説明では“メソッド”という用語を使います。

クラス

一般に、同じ形をしたオブジェクトは複数存在します。たとえば、従業員データの場合、富士通太郎さんの従業員データもあれば、ほかの人の従業員データもあります。これらはそれぞれ別々の値を持っています。しかし、すべて同じ形であり、かつ、同じメソッドを持っています。つまり、従業員データの定義情報があって、そこから作られた従業員オブジェクトが複数存在するわけです。オブジェクト指向では、この定義情報のことをクラスと呼びます。

クラスは、共通の属性を持ったオブジェクトのグループとみなすこともできます。あるクラスに属するオブジェクトは、すべて同じ構造のデータを持ち、同じメソッドを持っています。つまり、クラスは、そのクラスに属するオブジェクトの振舞いを定義しているわけです。また、すべてのオブジェクトは必ずどれかのクラスに属しています。

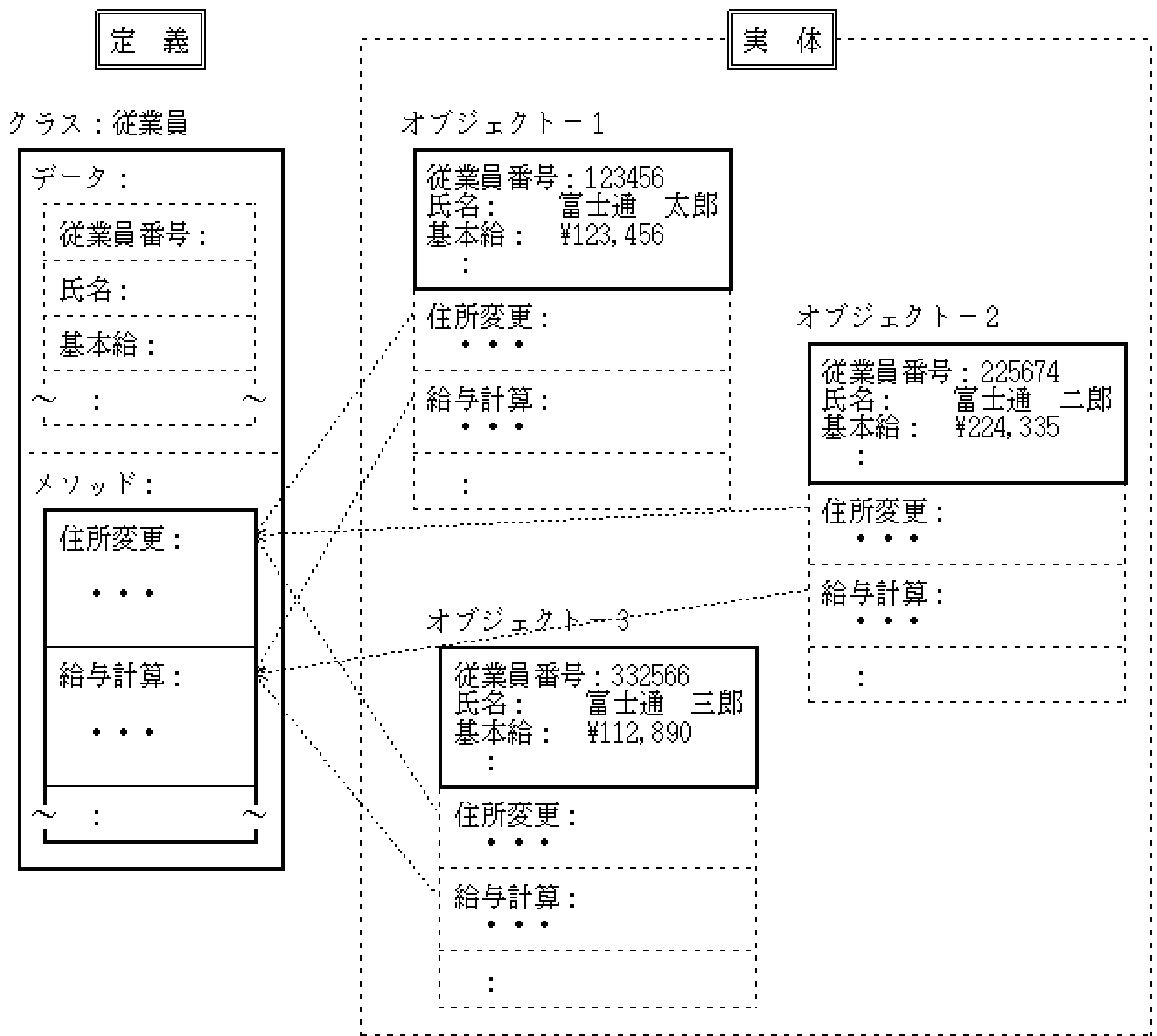
例

従業員クラスとオブジェクト

従業員管理プログラムでは、複数の従業員データを扱います(ここでは、単純化するために、従業員の種類は1種類だけと仮定します)。それぞれの従業員データは、いろいろな情報(たとえば、従業員番号、氏名や住所)を持っています。そして、それぞれの従業員データは、外部からアクセスするために、いくつかの手続き(たとえば、住所変更や給与計算)を持っています。そして、すべての従業員データは、同じデータ構造と同じメソッドを持っています。

これら個々の従業員データがオブジェクトで、それらの性質(データ構造と手続き)を定義しているのが、従業員クラスです。

すべての従業員オブジェクトは同じように振る舞います。従業員データのクラスとオブジェクトの関係は、以下の図のようになります。



この例では、従業員クラスのオブジェクトが3つあります。データ構造はクラスで定義されています。しかし、実際に値を持っているのは、3つのオブジェクトです。これらは、同じ構造です。しかし、持っている値は異なります。

また、メソッドもクラスで定義されています。メソッドは、一見それぞれのオブジェクトに含まれているように振る舞います。しかし、実は1つしか存在しません。メソッドの手続きは、データと違って、異なる値を持つ必要がないからです。それぞれのオブジェクト上でメソッドを実行すると、実際には、クラスで定義されたメソッドが実行されます。

13.2.2 オブジェクトインスタンスの作成・参照

個々のオブジェクトは、実行時に動的に作成されます。作成方法はプログラミング言語により異なります。COBOLの場合はファクトリオブジェクトという特殊なオブジェクトで作成します。詳細については、“第14章 オブジェクト指向プログラミング機能～基本的な使い方～”を参照してください。

オブジェクトは、それぞれオブジェクト参照という値を持っています。これは、オブジェクトを一意に指すためのポインタです。クラスが同じであっても、異なるオブジェクトであれば、オブジェクト参照の値は異なります。

オブジェクトを生成すると、必ずオブジェクト参照が割り当てられます。この値は、オブジェクト参照データ項目に格納します。そして、それ以降は、そのデータ項目を使うことにより、オブジェクトを一意に識別できます。

13.2.3 メソッドの呼出し

オブジェクト指向のプログラムは、オブジェクトにメッセージを送ることから始まります。オブジェクトは、メッセージを受け取ると、そのメッセージに対応したメソッドを実行します。メッセージの送信は、プログラム上ではオブジェクトに対するメソッド呼出しとして表します。

オブジェクトに対してメソッドを呼び出すためには、以下の情報が必要です。

- メソッドを実行するオブジェクトはどれか？
- メソッド名
- パラメタ(必要な場合だけ)

メソッドを実行するオブジェクトは、オブジェクト参照データ項目で指定します。

例

従業員クラスの給与計算処理

オブジェクト

従業員番号：123456
氏名：　富士通　太郎
基本給：　¥123,456
給与計算
・・・

データxは、富士通太郎さんの
従業員オブジェクトを
指しています。

プログラム

x

・・・

x::給与計算

・・・

[1]

[2]

クラス

従業員番号
氏名
基本給
・・・

メソッド：給与計算
・・・
[3] 給与を計算する
・・・

[4]

・・・

・・・

::　左辺のデータ項目が参照するオブジェクトに対して、右辺の
メソッドを呼び出すことを示します。

[1] 富士通太郎さんの従業員オブジェクトに対して、“給与計算”メソッドを呼び出します。

[2] 富士通太郎さんの従業員オブジェクト上で、“給与計算”メソッドを実行します。

[3] 給与計算”メソッドでは、富士通太郎さんの従業員オブジェクト内のデータを参照・設定しながら、給与計算処理を行います。

[4] 給与計算処理が終わったら、呼出し元に戻ります。

13.2.4 継承

オブジェクト指向の長所の1つは、既存のプログラムを簡単に再利用できることです。これは、継承と呼ばれる仕組みで実現されています。

継承とは、文字どおり、ほかのクラスの性質をそのまま受け継ぐことです。新しいクラスを作る場合に継承を使用すると、既存のクラスの性質をそのまま受け継ぐことができます。さらに、新しいデータを追加したり、新しいメソッドを追加したり、メソッドを置き換えたり、さまざまな改造ができます。つまり、既存のクラスを継承すれば、そこからの差分をコーディングするだけで、新しいクラスを作ることができます。もちろん、既存のクラスで定義されていたデータやメソッドは、新しいクラスでもそのまま使えます。

継承される(基となる)クラスを親クラス、継承する(性質を受け継ぐ)クラスを子クラスと呼びます。親クラスで定義されたデータは、子クラスでも暗黙に定義されます。また、親クラスで定義されたメソッドは、子クラスでも定義されているかのように使用できます。

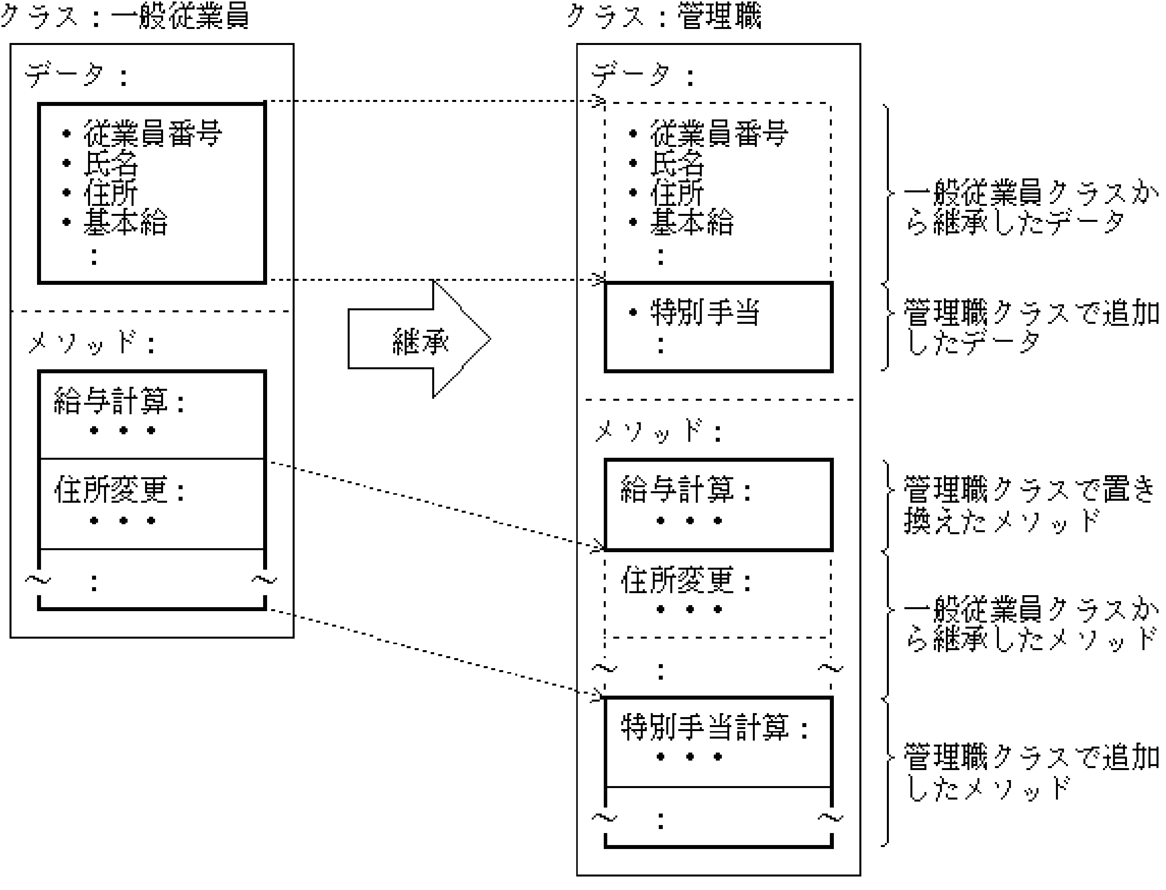
例

新しい従業員クラスの作成

継承による再利用の機構は、既存のシステムに新しい機能を追加するときに有効です。

たとえば、一般従業員クラスに対して、特別手当の支給だけが異なる管理職クラスを作る場合、一般従業員クラスを継承して、新しい“管理職”クラスを作ります。管理職クラスでは、一般従業員クラスに対して、特別手当の処理を追加します。そのためには、それに関するデータ(特別手当)とメソッド(特別手当の計算)の追加が必要です。また、特別手当を加算するために給与計算処理も変更する必要があります。

一般従業員クラスと管理職クラスの関係は、以下の図のようになります。



クラスの汎化

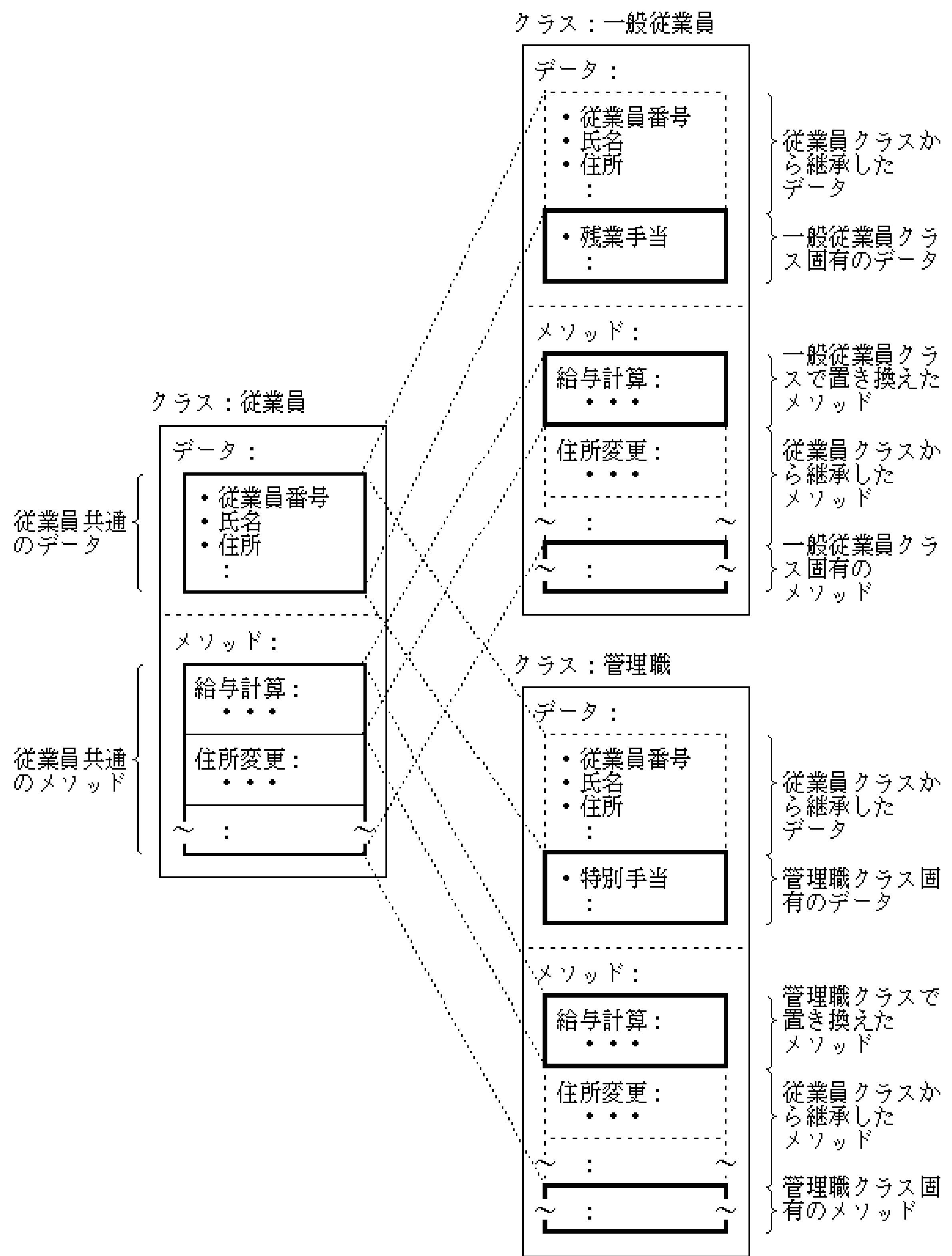
継承には、もう1つの使い方があります。

前の例では、一般従業員クラスに特別な処理を追加し、管理職クラスを作成しました。これは、一般従業員の特殊化されたものが管理職だと仮定したからです。しかし、実際には、一般従業員と管理職は、多くの共通点もあれば、多くの相違点もあります。見方を変えれば、一般従業員と管理職は、従業員という共通の性質のうえに、一般従業員・管理職それぞれ固有の性質を追加したものです。

たとえば、一般従業員と管理職は、給与の計算方法だけが異なっていると仮定します。一般従業員は残業時間に比例する残業手当が、管理職には一定額の特別手当が支給されると仮定します。それ以外の性質については、すべて同じとします。このような場合、まず、従業員共通の性質を持った従業員クラスを作ります。そして、そのクラスを継承して一般従業員クラスと管理職クラスを作ります。つまり、従業員クラスには、すべての従業員に共通なデータとメソッドを定義しておき、一般従業員クラスには一般従業員固有の性質を、管理職クラスには管理職固有の性質を追加します。

このような構成にしておくと、新しく勤務形態を追加しても、既存のコードの修正はほとんど不要です。たとえば、パートタイム制を追加する場合でも、従業員クラスを継承したパートタイム従業員クラスを追加するだけです。

従業員クラス、一般従業員クラスおよび管理職クラスの関係は、以下の図のようになります。


クラス：一般従業員
データ：
・従業員番号
・氏名
・住所
：
・残業手当
：
従業員クラスから継承したデータ
一般従業員クラス固有のデータ
メソッド：
給与計算：
・・・
住所変更：
・・・
：
：
一般従業員クラスで置き換えたメソッド
従業員クラスから継承したメソッド
一般従業員クラス固有のメソッド
クラス：従業員
データ：
・従業員番号
・氏名
・住所
：
メソッド：
給与計算：
・・・
住所変更：
・・・
：
従業員共通のデータ
従業員共通のメソッド
クラス：管理職
データ：
・従業員番号
・氏名
・住所
：
・特別手当
：
従業員クラスから継承したデータ
管理職クラス固有のデータ
メソッド：
給与計算：
・・・
住所変更：
・・・
：
：
管理職クラスで置き換えたメソッド
従業員クラスから継承したメソッド
管理職クラス固有のメソッド

参考

- 一般従業員クラス、管理職クラスのオブジェクトは存在します。しかし、従業員クラスのオブジェクトは存在しません。従業員クラスは、一般従業員クラス、管理職クラスに共通の性質を定義するために作成されたクラスです。このようなクラスを抽象クラスと呼びます。
- 従業員クラスの給与計算メソッドは、インタフェースを定義しているだけです。実際の処理は、子クラスである一般従業員クラスおよび管理職クラスで定義されています。このように、抽象クラスは、子クラスで定義する必要のあるメソッドの宣言のためによく使用されます。

13.2.5 多態と動的束縛

オブジェクト指向のもう1つの特徴は、柔軟性です。そのための仕組みが、多態および動的束縛です。

多態

多態とは、"1つのものが複数の形態をとることができる"ことです。

あるクラスAと、それを継承するクラスB、Cがある場合、BもCもAの性質を受け継いでいます。見方を変えれば、B、CはAの一種です(B is a A,C is a A)。そのため、継承関係はis_a関係とも呼びます。

前節で説明した一般従業員クラスと管理職クラスは、従業員クラスの子クラスです。つまり、一般従業員オブジェクトも管理職オブジェクトも、従業員オブジェクトの一種です。そして、従業員オブジェクトの性質をすべて引き継いでおり、従業員オブジェクトとして振る舞うことができます。たとえば、プログラム上で"従業員クラスのオブジェクトに対し、給与計算を行う"と書かれていても、一般従業員オブジェクトや管理職オブジェクトが従業員オブジェクトの代わりになることができます。それは、どちらのオブジェクトも"給与計算"メソッドを持つという性質を従業員クラスから引き継いでいるからです。

1つのもの(従業員オブジェクト)が、複数の形態(一般従業員オブジェクトと管理職オブジェクト)をとれたわけです。

動的束縛

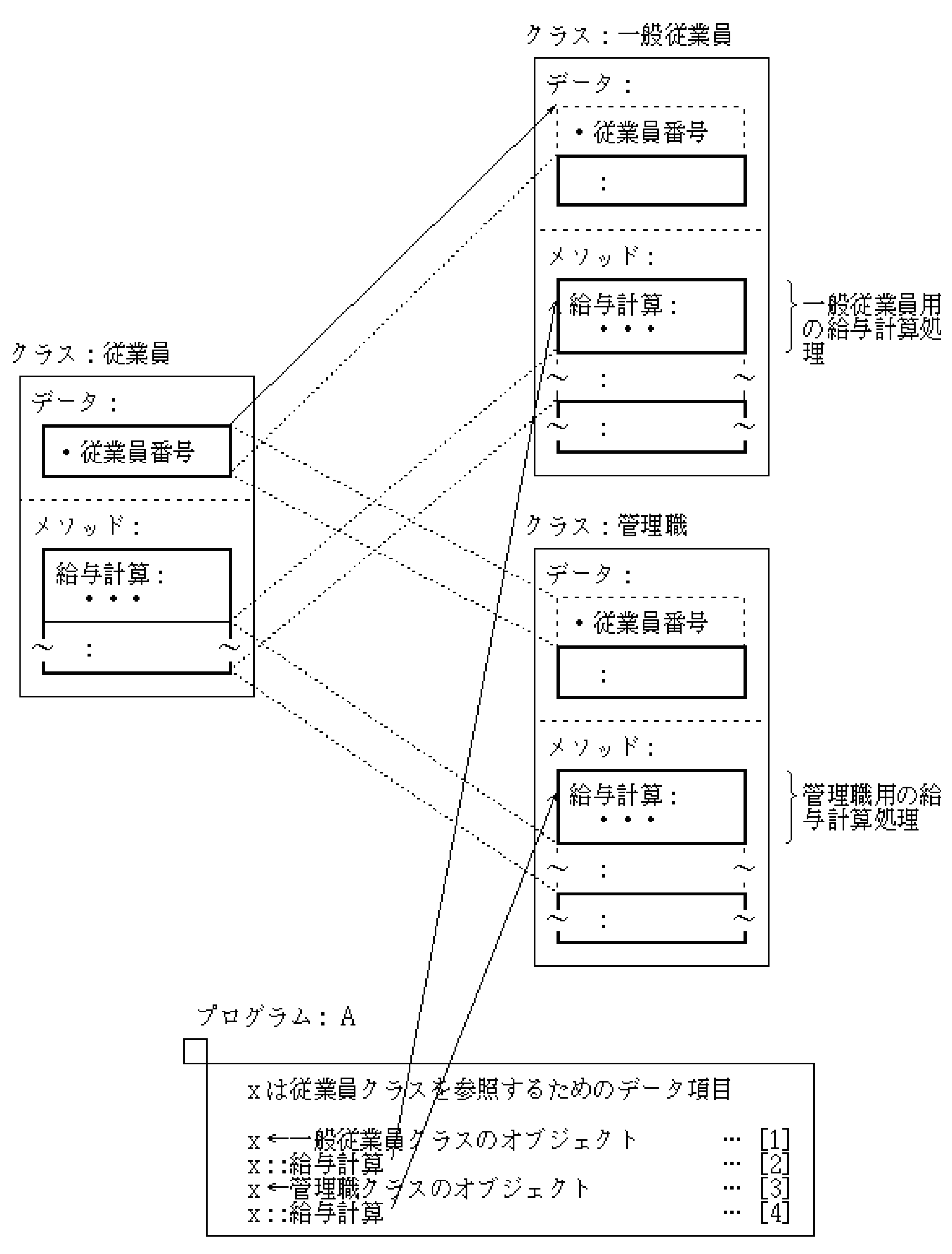
一般従業員クラスと管理職クラスは、同じインタフェースの給与計算処理を持つと仮定しました。しかし、その中の処理は異なります。一般従業員の場合は一般従業員用の、管理職の場合は管理職用の給与計算を実行する必要があります。従業員オブジェクト上で給与計算メソッドを呼び出します。しかし、オブジェクトが一般従業員クラスのものであれば一般従業員用の給与計算メソッドが呼び出されます。そして、管理職クラスのオブジェクトであれば管理職用の給与計算メソッドが呼び出される必要があります。

このように、オブジェクト指向では、同じメソッド呼出しであっても、そのメソッドを実行するオブジェクトによって呼び出されるメソッドが変わります。つまり、呼び出されるメソッドは実行時に動的に決まります。このような機構を、呼び出すメソッドが実行時に動的に決まるということで、動的束縛と呼びます。

例

一般従業員と管理職

一般従業員クラスも管理職クラスも、従業員クラスを継承しています。そして、それぞれのクラスは、固有の"給与計算"というメソッドを持っています。従業員クラスのオブジェクトを参照するデータ項目を用意すると、それは一般従業員オブジェクトも管理職オブジェクトも参照できます。一般従業員オブジェクトと管理職オブジェクトを使った多態と動的束縛の例を、以下の図に示します。



xは、従業員クラスのオブジェクトを参照するデータ項目です。そのため、xには一般従業員クラスのオブジェクト([1])も管理職クラスのオブジェクト([3])も代入できます。また、[2]と[4]でメソッドを呼び出しています。これらはすべて同じ構文です。しかし、実際に呼び出されるメソッドは異なります。[2]では一般従業員クラスの給与計算メソッドが、[4]では管理職クラスの給与計算メソッドが呼び出されます。

このような作りにしておくと、すべての従業員の処理を1箇所でできます。さらに、新しい勤務形態(たとえばパートタイム)を追加しても、コードを修正する必要がありません。

13.3 オブジェクト指向のメリット

オブジェクト指向プログラミングには、以下の特徴があります。

- データと手続きをカプセル化し、外部から隠蔽することにより、オブジェクトとの接点はインタフェースだけになります。
  - 内部の実現方法(データ構造や手続きのコード)を変更しても、インタフェースが同じであれば、それを利用するプログラムには影響がありません。つまり、変更に対する影響を局所化できます。
  - 公開されているのはインタフェースだけなので、あらかじめ決められた正しい方法以外では使用できません。その結果、必然的にプログラムの信頼性が向上します。
- 継承により、親クラスの性質を子クラスに引き継ぐことができます。
  - 既存のクラスをもとに新しいクラスを作る場合、差分だけのコーディングですみます。親クラスの変更は子クラスにも反映されるので、変更時の影響を局所化できます。
  - クラス間の共通の性質を抜き出して、1つのクラスにまとめることができます。共通処理が1箇所にまとめられているので、変更時の影響を局所化できます。
- 多態により、柔軟なプログラミングができます。
  - 1つの手続きで複数のクラスを扱うことができます。そのため、扱うクラス(従来ならデータ)の違いにより処理を分ける必要はありません。
  - 新しいクラスを追加しても、処理を変更する必要はありません。

これらの特徴から、オブジェクト指向を導入することにより、以下の効果が期待できます。

作成時

- 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。→ 開発作業の効率向上

保守時

- システム変更時の影響を局所化できます。
- システムの変更に対し、柔軟に対応できます。→ 保守作業の効率向上

このように、オブジェクト指向言語を導入することにより、開発効率および保守効率を向上させることができます。

第14章　オブジェクト指向プログラミング機能～基本的な使い方～

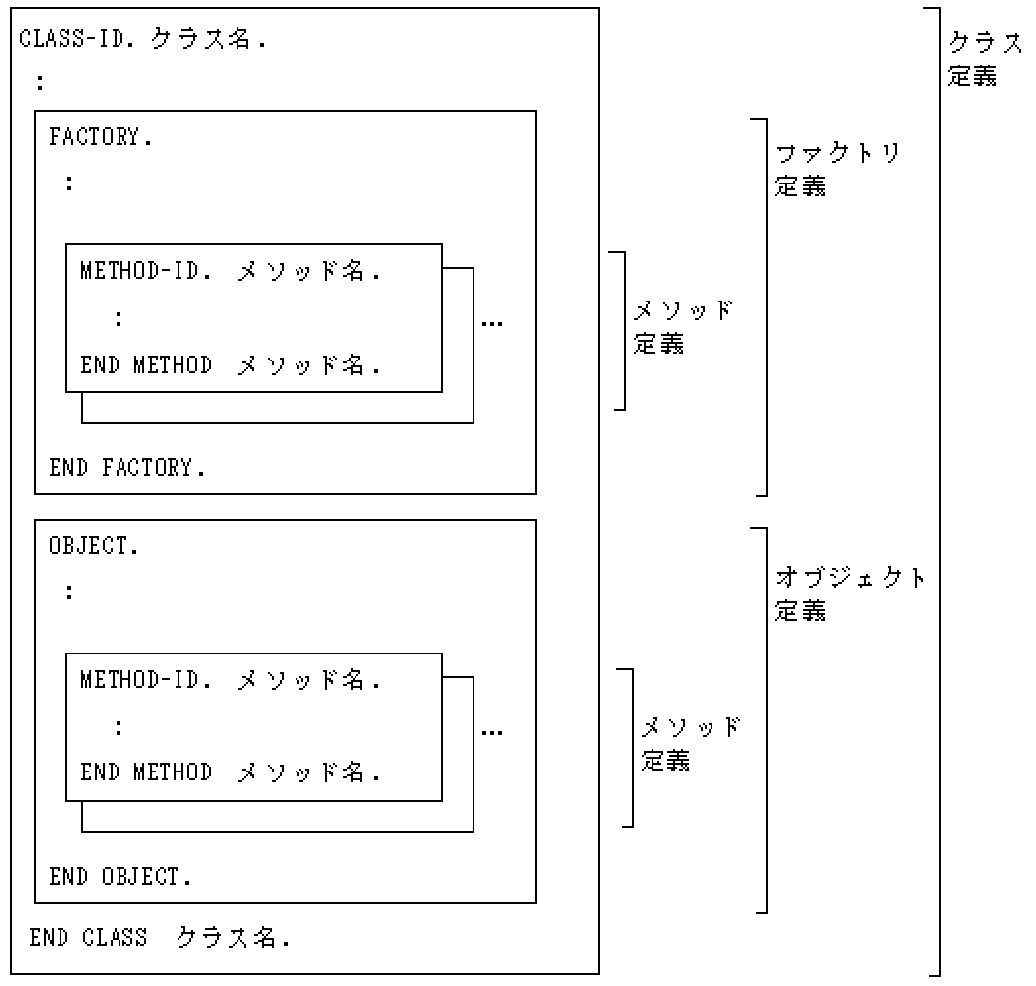
本章では、"第13章 オブジェクト指向プログラミングとは"で紹介したオブジェクト指向プログラミングについて、より具体的にコーディングレベルで説明します。

14.1 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します。そのために、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。



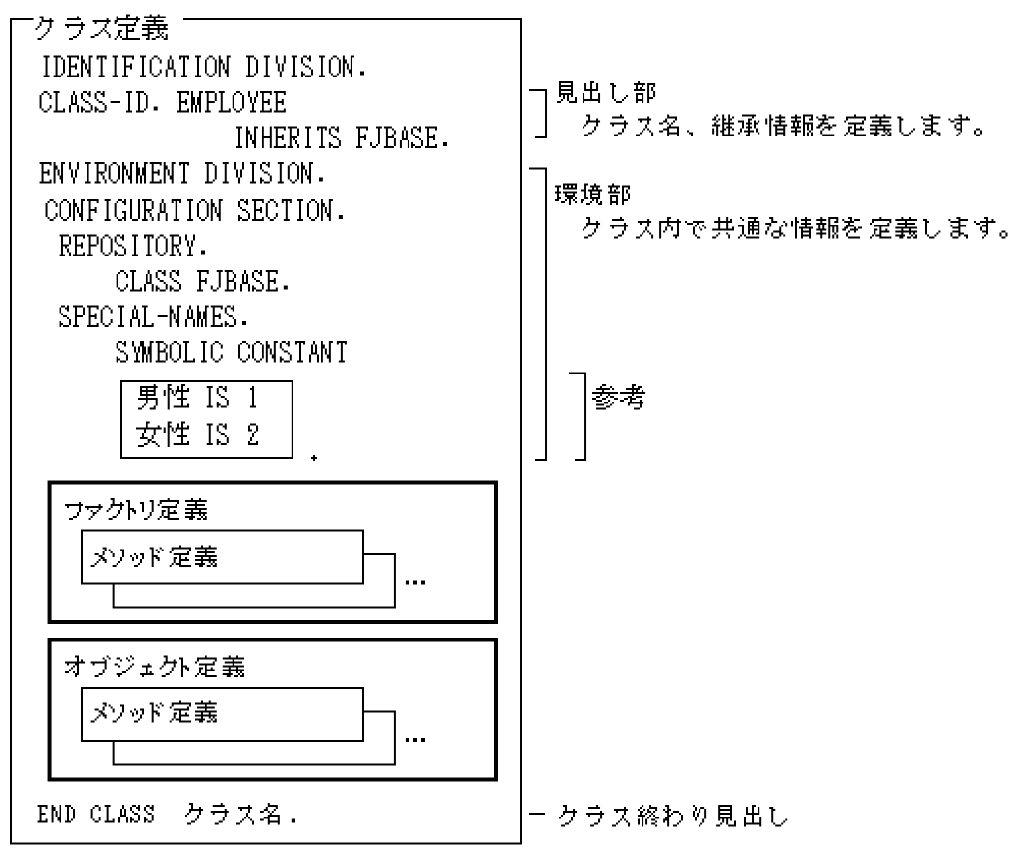
それぞれの定義について、具体的な定義方法や役割について説明します。

なお、以降の説明は、基本的に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。しかし、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。あらかじめ、ご了承ください。

14.1.1 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見出し部と、クラス定義内で共通の情報を定義する環境部を定義することができます。しかし、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は、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、記号定数などのデータを宣言します。クラス定義の環境部で宣言されたデータ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)。

注意

プログラム名／クラス名／メソッド名には、以下の場合に、使用できない文字があります。

- プログラム名段落(PROGRAM-ID)、CALL文およびCANCEL文でプログラム名を指定する場合
- クラス名段落(CLASS-ID)、メソッド名段落(METHOD-ID)、リポジトリ段落(REPOSITORY)およびINVOKE文でクラス名やメソッド名を指定する場合

上記の場合に指定できない文字は以下のとおりです。

- 最初の文字がアンダースコア
- コード系がEUCの場合の日本語定数で、拡張漢字、拡張非漢字、利用者定義文字
- コード系がEUCの場合の半角カナ文字(翻訳オプションKANA(JIS8)が指定されている場合だけ)

なお、上に示すような文字以外は使用できます。ただし、指定され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。

14.1.2 ファクトリ定義

ファクトリ定義は、オブジェクトに共通なデータ(ファクトリデータと呼びます)を定義したり、オブジェクトの管理(生成など)を行うメソッド(ファクトリメソッドと呼びます)を定義します。

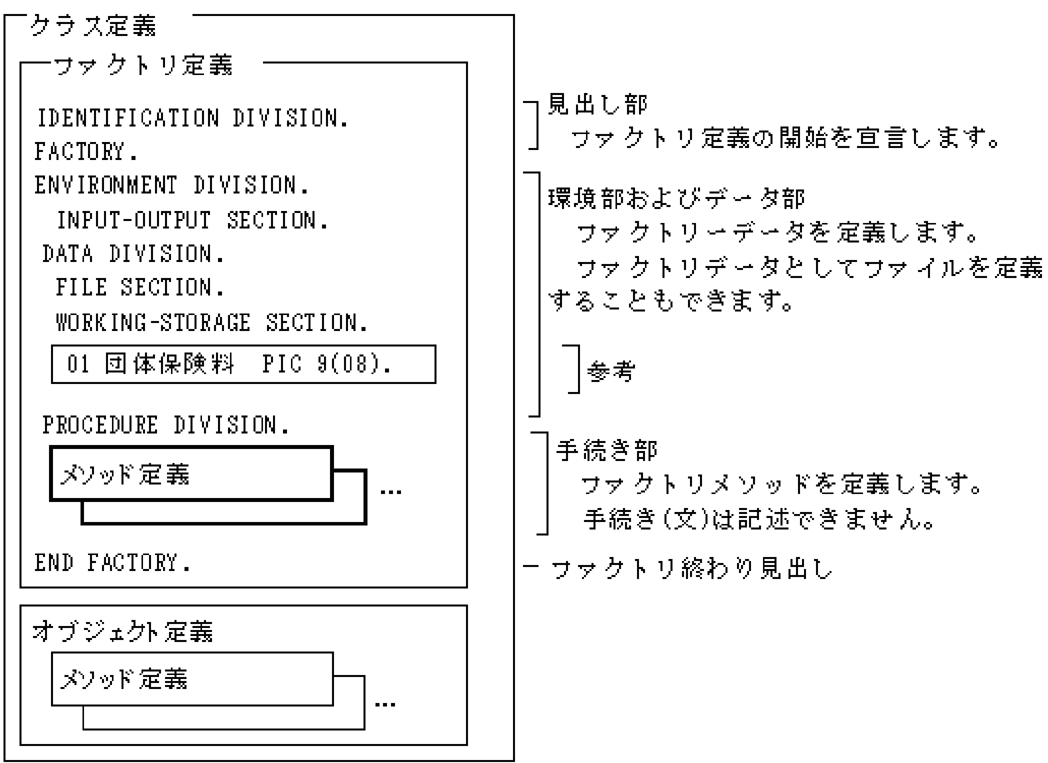
ファクトリ定義は、ファクトリ定義を表すための見出し部、ファクトリデータを定義するための環境部およびデータ部、ファクトリメソッドを定義するための手続き部から構成されます。

注意

手続き部は、ファクトリメソッドを定義するだけであり、手続き(COBOLの文)を記述することはできません。

参考

これらの定義を必要としないクラスの場合、ファクトリ定義(ファクトリ始め見出し～ファクトリ終わり見出しまで)を省略することもできます。



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

では、具体的にファクトリ定義の役割について説明します。

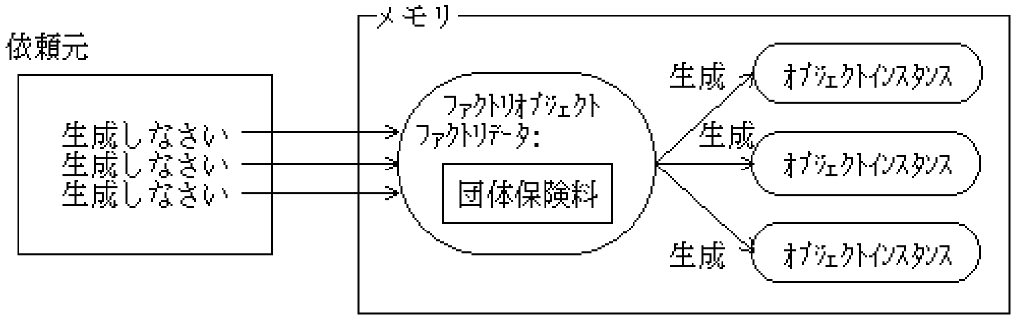
オブジェクトインスタンスの概念については"13.2.2 オブジェクトインスタンスの作成・参照"で説明しました。しかし、COBOLにはファクトリオブジェクトという概念があります。

ファクトリオブジェクト

オブジェクトインスタンスが複数個生成されるのに対して、このファクトリオブジェクトは、1クラスについて、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。しかし、その指示を受け取るのは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、ほか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、“14.3.2 FJBASEクラス”を参照してください。

14.1.3 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。

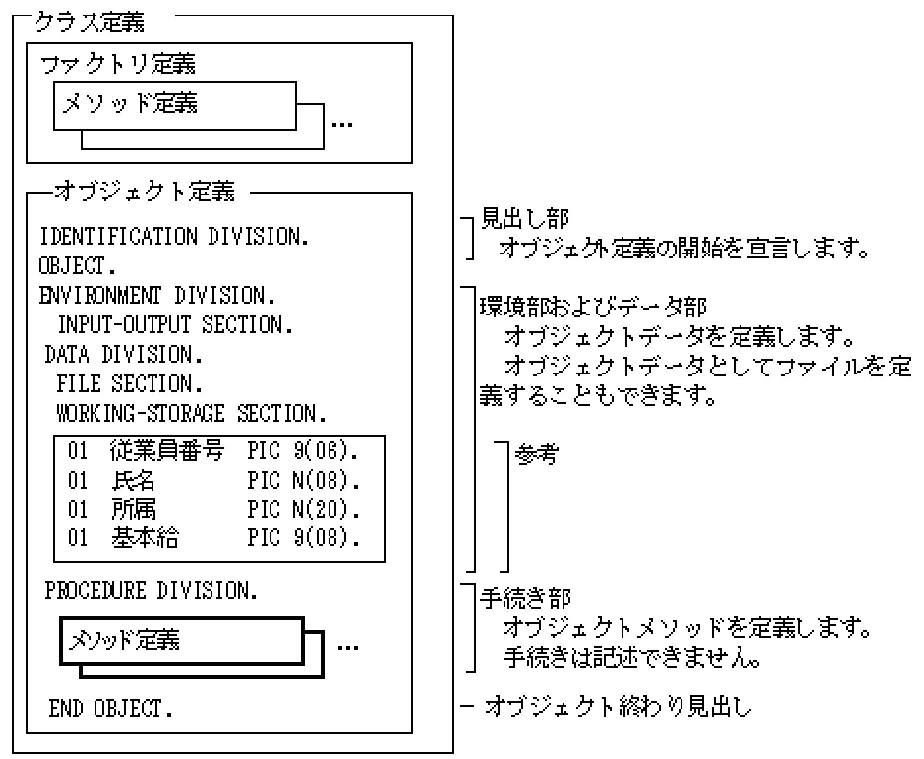
オブジェクト定義の構成はファクトリ定義と同じで、オブジェクト定義を表すための見出し部、オブジェクトデータを定義するための環境部およびデータ部、オブジェクトを定義するための手続き部からなります。

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。

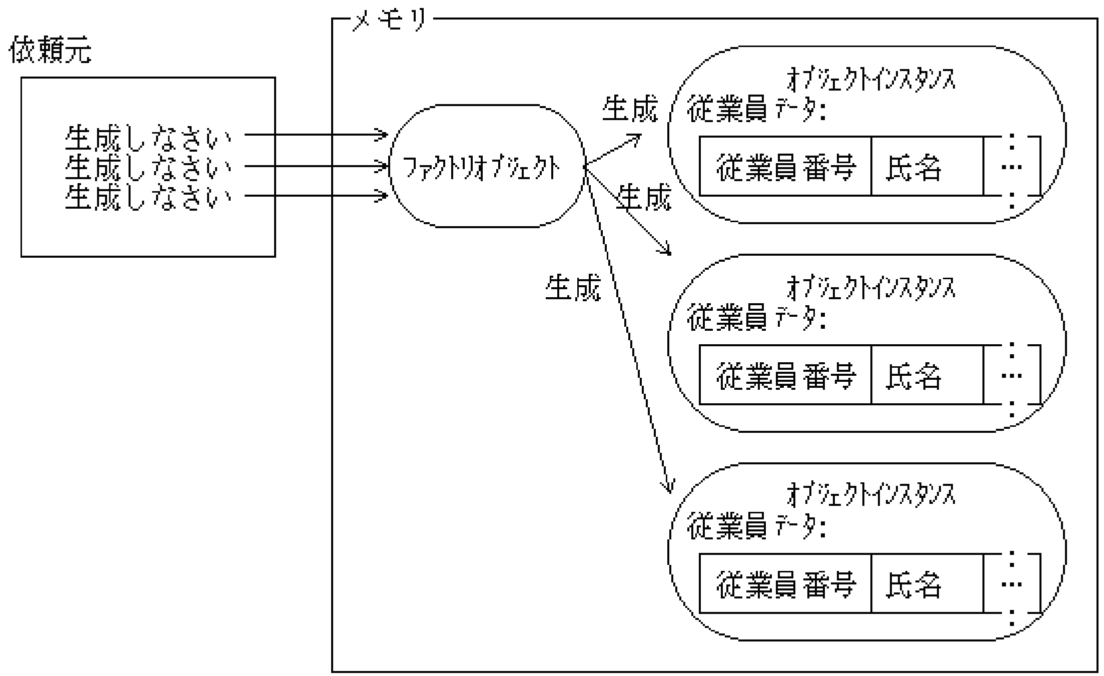


オブジェクト定義の環境部およびデータ部で定義されたデータは、オブジェクトメソッドでだけアクセスすることができます(上図の太線で囲まれている部分)。

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。

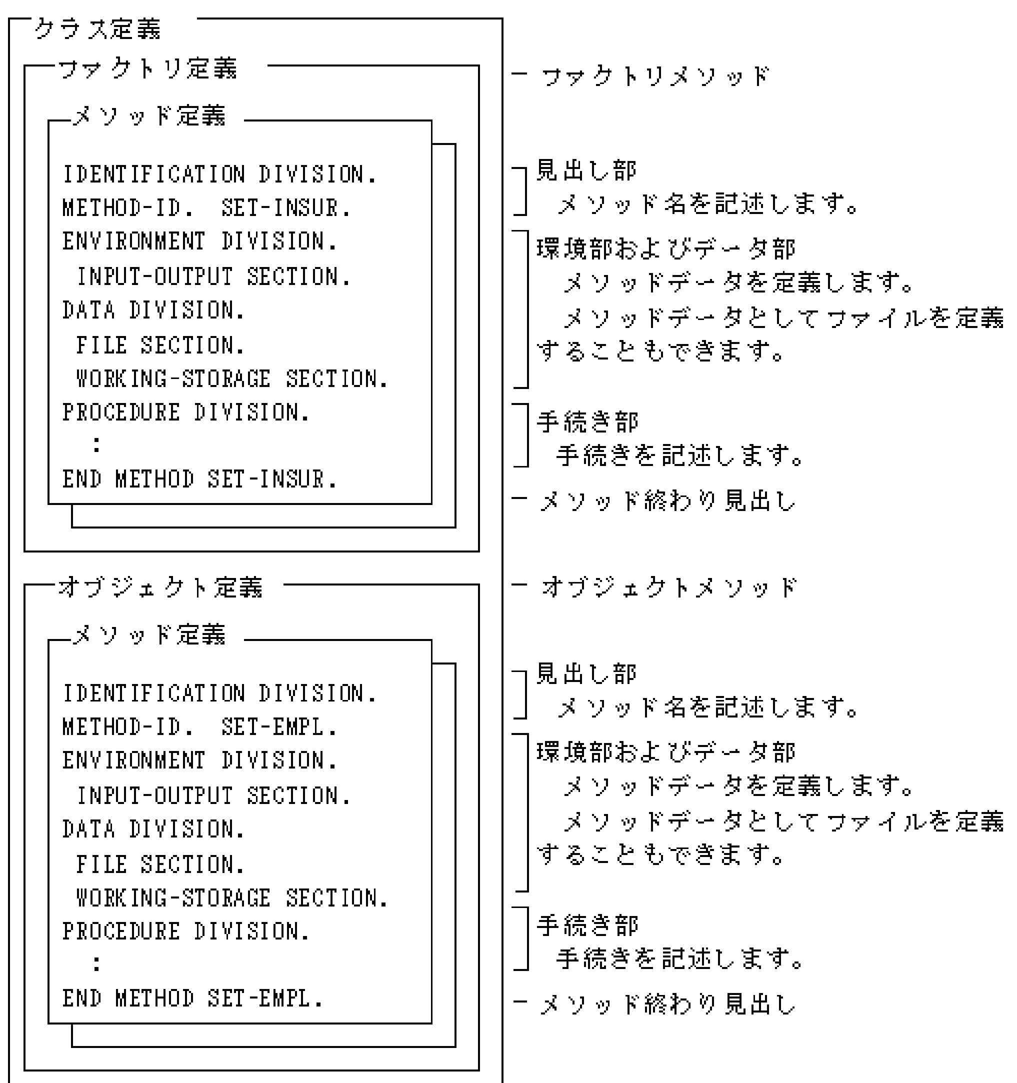


14.1.4 メソッド定義

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。

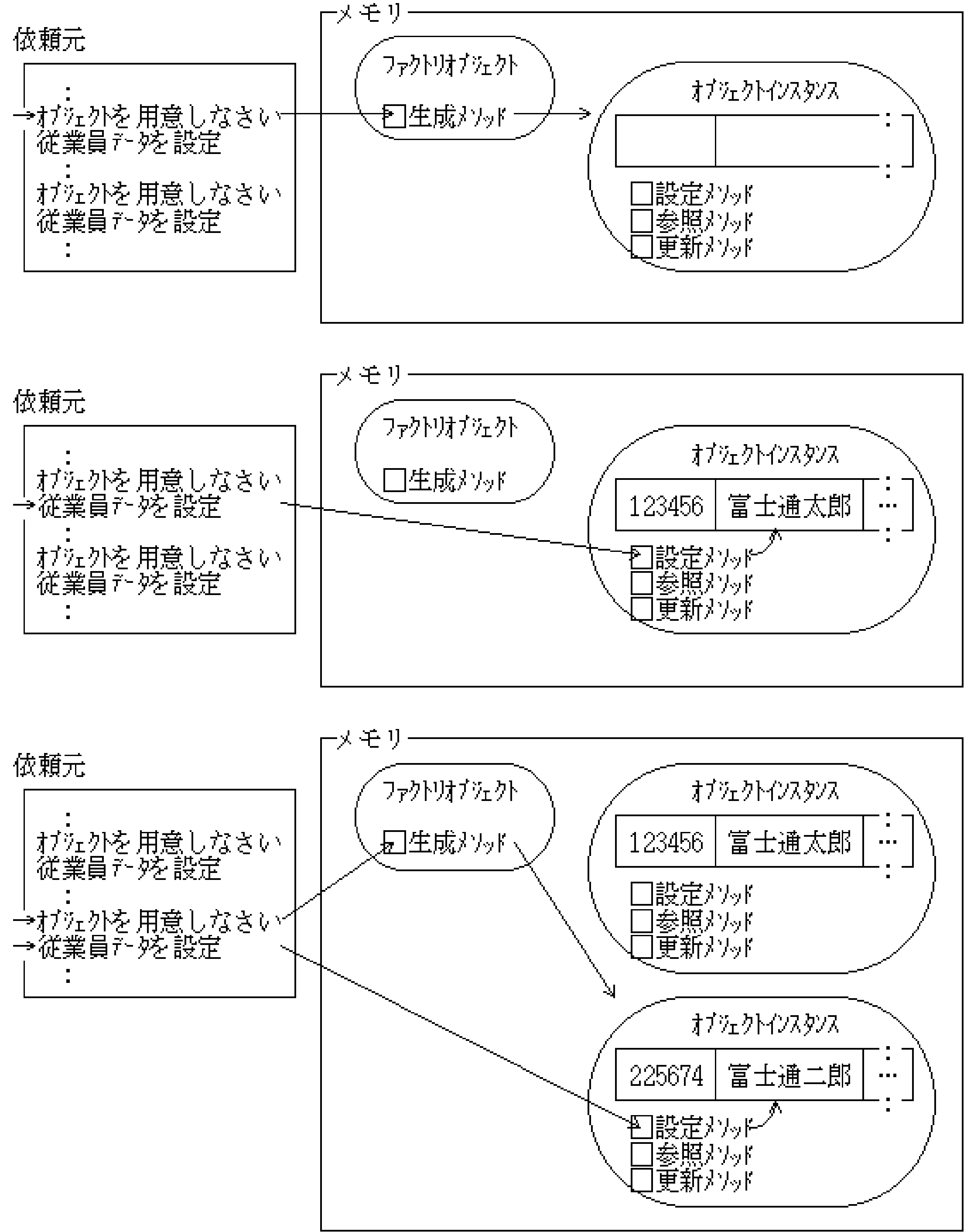


メソッド定義には、ファクトリメソッドとオブジェクトメソッドがあります。しかし、どちらも、メソッド内で定義されたデータは、そのメソッド内でだけアクセスすることができます。

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)が存在するとみなすと分かりやすくなります。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。



14.2 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。

14.2.1 メソッドの呼出し

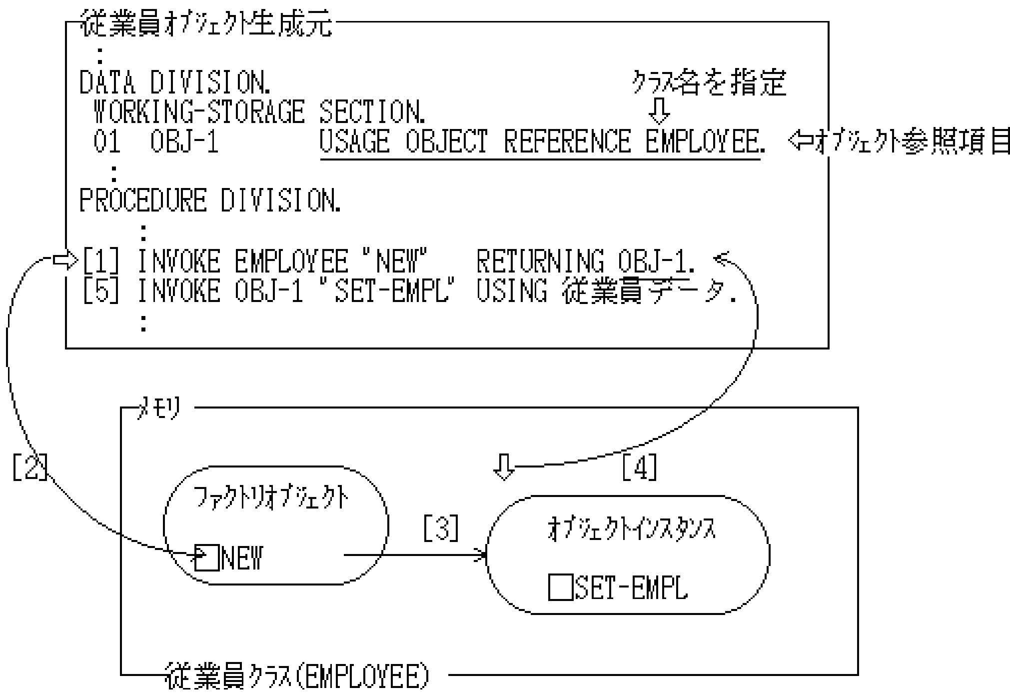
オブジェクトインスタンスを操作するためには、オブジェクトメソッドを呼び出す必要があります。このとき、「どのオブジェクトインスタンス」の「どのメソッド」を呼び出すかを指定します。しかし、「どのオブジェクトインスタンス」を表現するためにオブジェクト参照項目と呼ばれるデータ項目を利用します。

14.2.1.1 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14.1 従業員オブジェクトインスタンスの生成



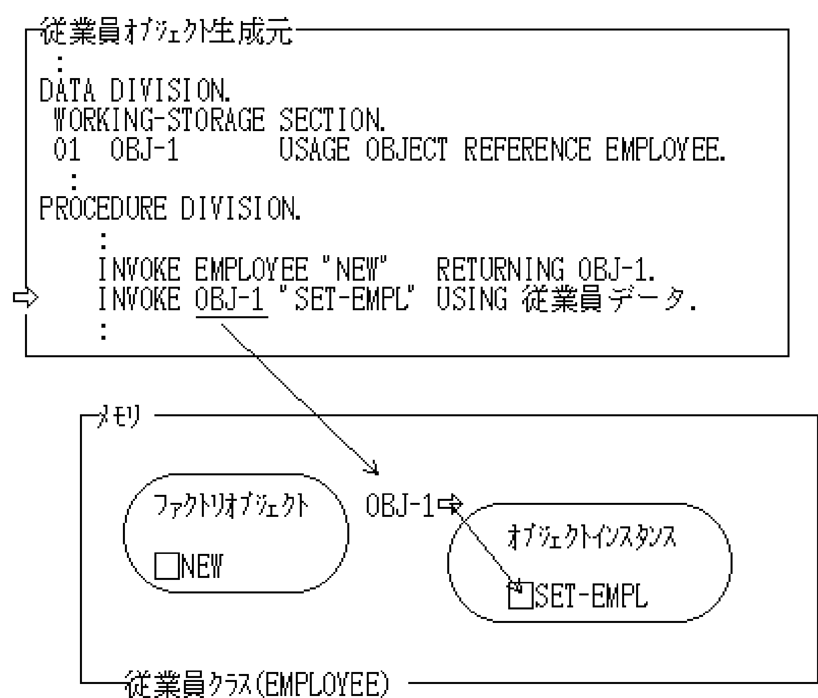
[1][2] NEW メソッドを呼び出すと、

[3] NEW メソッドはオブジェクトインスタンスを生成後、

[4] オブジェクト参照を呼出し元へ返却する。

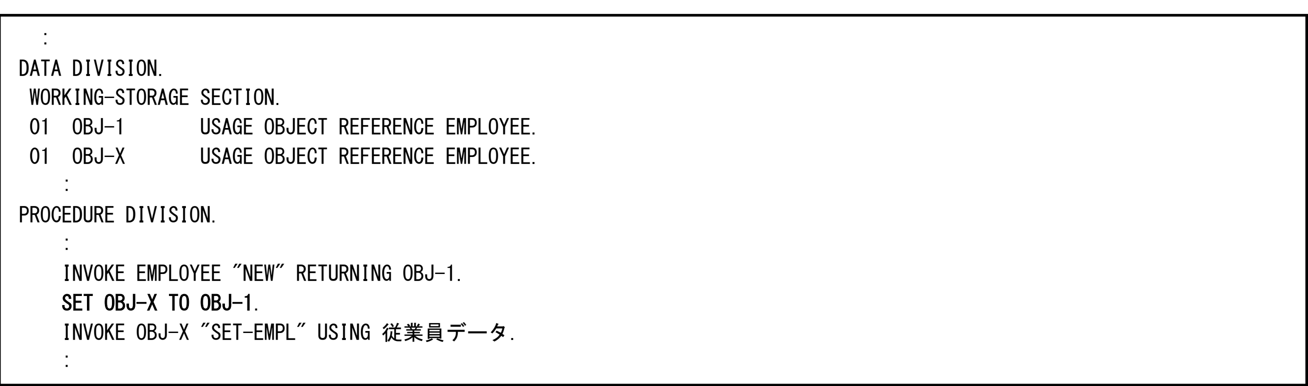
[5] オブジェクト参照を使用してメソッドを呼び出す。

図14.2 従業員オブジェクトインスタンスへのデータ登録



"図14.1 従業員オブジェクトインスタンスの生成"、"図14.2 従業員オブジェクトインスタンスへのデータ登録"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する必要があります。

オブジェクト参照項目は、SET文を用いてほか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を、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句に不一致があると正しく受け渡すことができません。

14.2.1.2 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。しかし、メソッドを呼び出す場合には、INVOKE文を利用する必要があります。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14.1 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。

14.2.1.3 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。

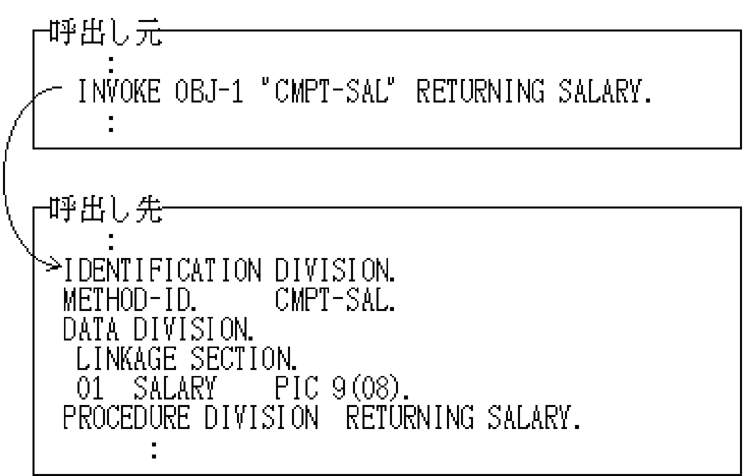
USING指定

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指定とは少し用途が異なります。また、USING指定が複数個のパラメタを指定できるのに対し、RETURNING指定は、1つだけ指定可能です。



RETURNING指定は、復帰値であるため、呼出し元で設定された値を呼出し先で参照することはできません。つまり、一方通行の関係となります。

呼出し元の処理イメージは、下図のとおりです。

```
呼出し元
    :
INVOKE OBJ-1 "CMPT-SAL" RETURNING SALARY.
    :
```

↓

```
処理イメージ
    :
INVOKE OBJ-1 "CMPT-SAL" USING TEMP.
MOVE TEMP TO SALARY.
    :
```

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。

14.2.2 オブジェクトの寿命

オブジェクトの生成、更新についてはこれまでに説明してきました。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

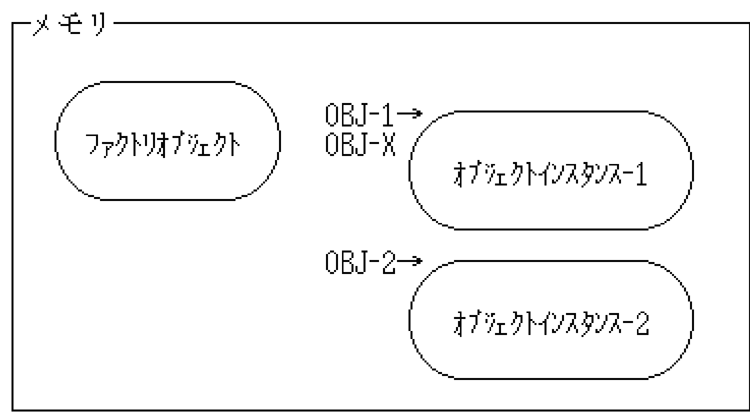
ファクトリオブジェクトは、クラスがローディングされてからCOBOLの実行環境が閉鎖されるまでメモリ上に存在し続けます。なお、アプリケーションの動作中に削除する手段はありません。

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に削除されます。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に、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オブジェクトを転記し、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オブジェクトを転記してから実行単位を終了してください。このような、終わり方をしない場合、終了処理メソッド_FINALIZEは呼び出されません。ただし、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。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。

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] 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] OBJ-1にNULL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] さらにOBJ-XにNULL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、メモリ上には残っています。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。

14.3 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。

14.3.1 継承の概念と実現

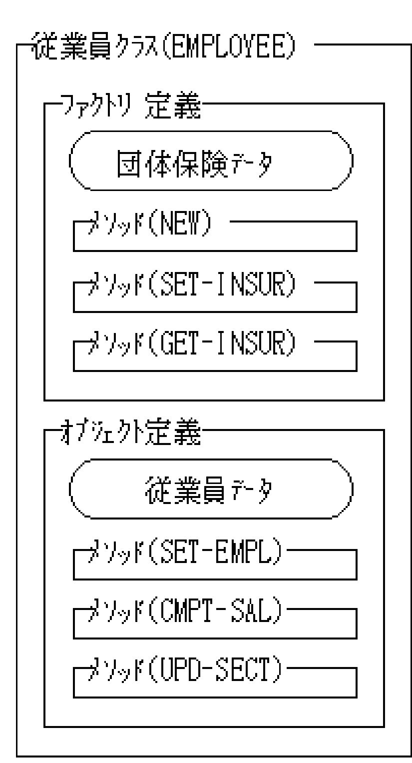
継承の概念については、すでに“第13章 オブジェクト指向プログラミングとは”で説明しているので、詳しい説明は省略します。しかし、少し、復習してみましょう。

継承の概念

従来のプログラミング手法では、既存の部品(ルーチン)とよく似た機能を持った部品を作成する場合がありました。このとき、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。

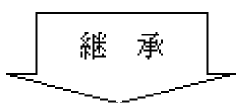


【団体保険ﾃﾞｰﾀ】
給与天引する団体保険料のﾃﾞｰﾀ

【NEWﾒｿｯﾄﾞ】
ｵﾌﾞｼﾞｪｸﾄｲﾝｽﾀﾝｽを生成します。
【SET-INSURﾒｿｯﾄﾞ】
団体保険ﾃﾞｰﾀを設定します。
【GET-INSURﾒｿｯﾄﾞ】
団体保険ﾃﾞｰﾀを参照（取出し）します。

【従業員ﾃﾞｰﾀ】
従業員番号、氏名、住所、給与などのﾃﾞｰﾀ

【SET-EMPLﾒｿｯﾄﾞ】
生成したｵﾌﾞｼﾞｪｸﾄｲﾝｽﾀﾝｽに従業員ﾃﾞｰﾀ を設定します。
【CMPT-SALﾒｿｯﾄﾞ】
従業員の給与計算を行います。
【UPD-SECTﾒｿｯﾄﾞ】
従業員の所属変更を行います。



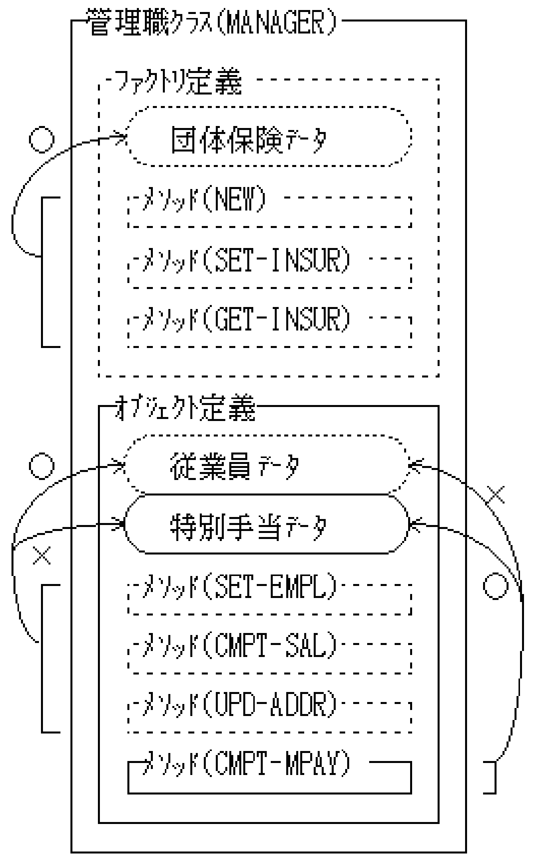
管理職ｸﾗｽ(MANAGER)　←従業員ｸﾗｽ(親ｸﾗｽ)からの差分だけを記述します。

ｵﾌﾞｼﾞｪｸﾄ定義
特別手当ﾃﾞｰﾀ
ﾒｿｯﾄﾞ(CMPT-MPAY)

【特別手当ﾃﾞｰﾀ 】
管理職の特別手当ﾃﾞｰﾀ

【CMPT-MPAY ﾒｿｯﾄﾞ】
特別手当の計算を行います。

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。



○：アクセス可能
×：アクセス不可

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負担になるので、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。この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

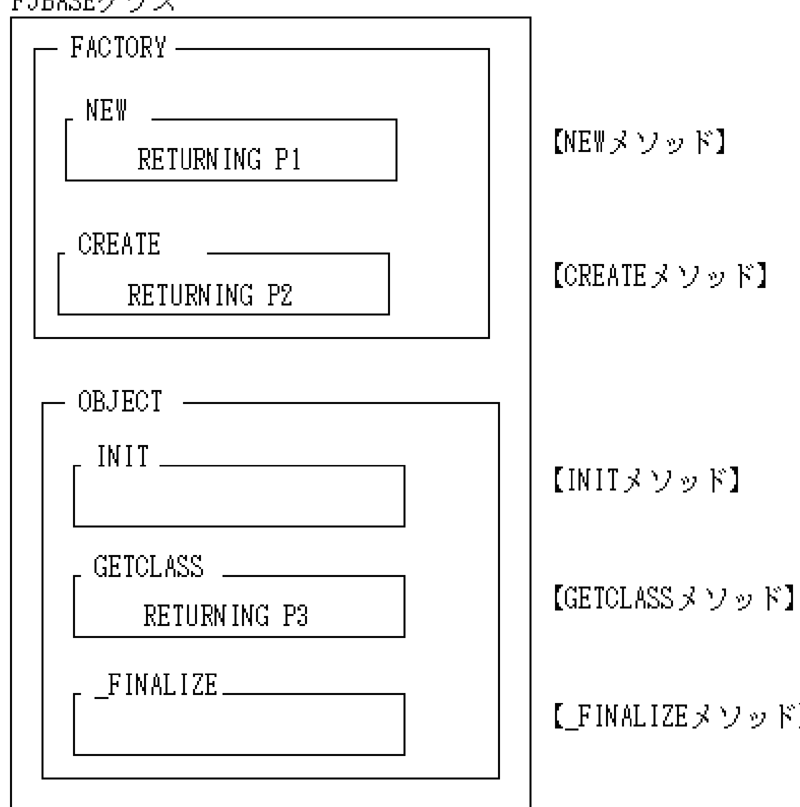
```
IDENTIFICATION DIVISION.        ┐オブジェクトメソッドの定義
 METHOD-ID. CMPT-MPAY.          │ 追加するメソッドだけを指定します。
    :                           │ (差分だけをコーディング)
 END METHOD CMPT-MPAY.          ┘
 END OBJECT.
 END CLASS MANAGER.
```

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。

14.3.2 FJBASEクラス

COBOLシステムでは、汎用的に利用されるような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、オブジェクトインスタンスの生成などを行うメソッド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。

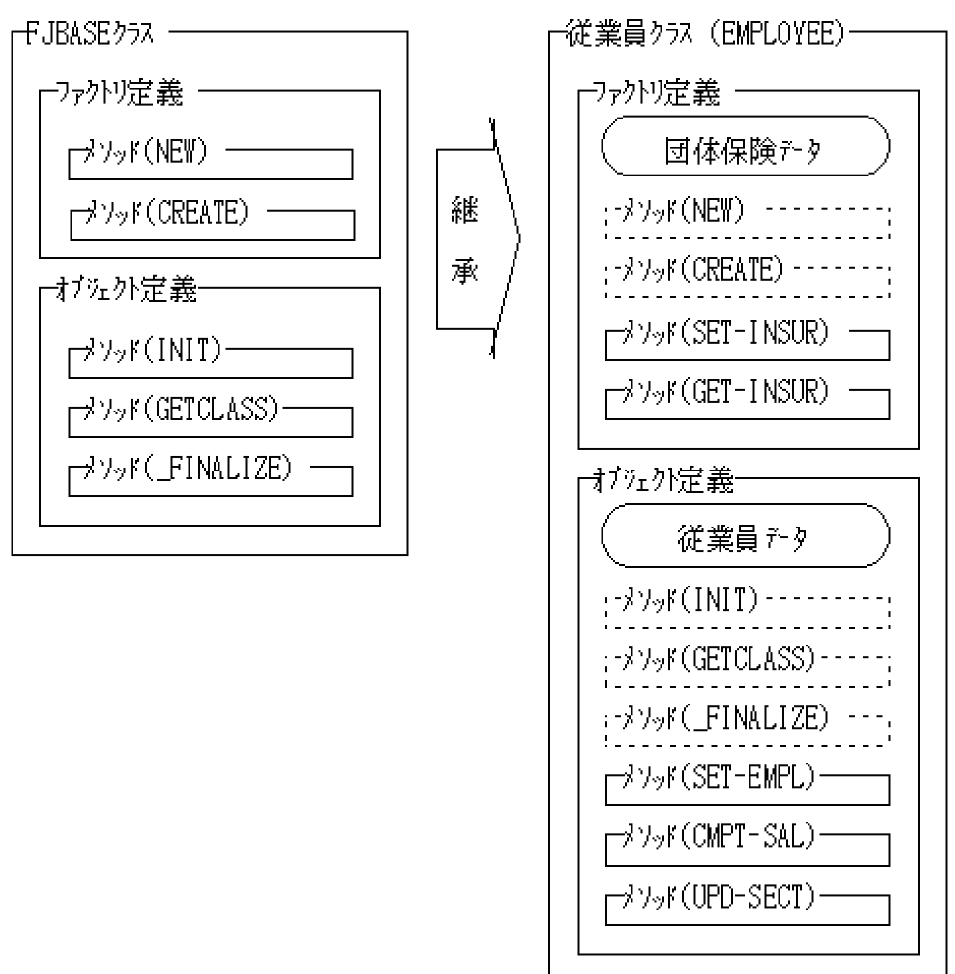


基本的な使い方で利用するのはNEWメソッドとGETCLASSメソッドです。その他のメソッドは、より高度なプログラミングを行う場合にだけ利用します(INITメソッドおよび_FINALIZEメソッドについては、"14.7.6 初期化処理メソッドと終了処理メソッド"を参照)。

継承はクラス定義に対して行われます。そのため、NEWメソッドだけが必要な場合も、ほかのメソッド(CREATE、INIT、GETCLASSおよび_FINALIZEの各メソッド)が組み込まれることになります。したがって、メソッドを呼び出す場合は注意してください。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ことはありません。新しくクラスを定義する場合は、必ずFJBASEクラスを継承してください。

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。

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。しかし、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でした。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。



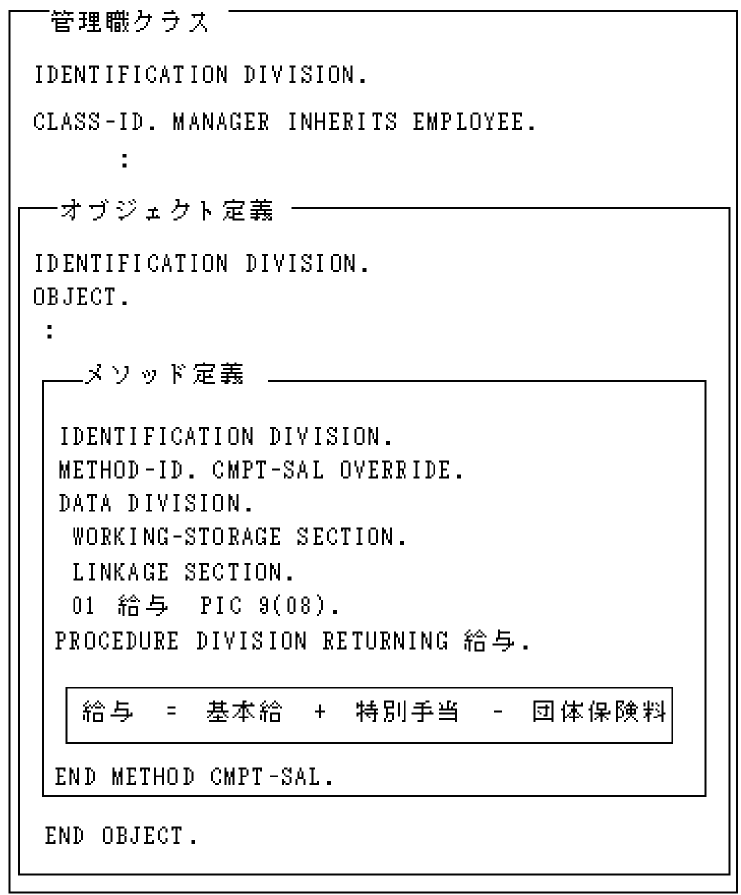
14.3.3 メソッドの上書き

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、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の例を以下に示します。

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ある必要があります。

14.4 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。

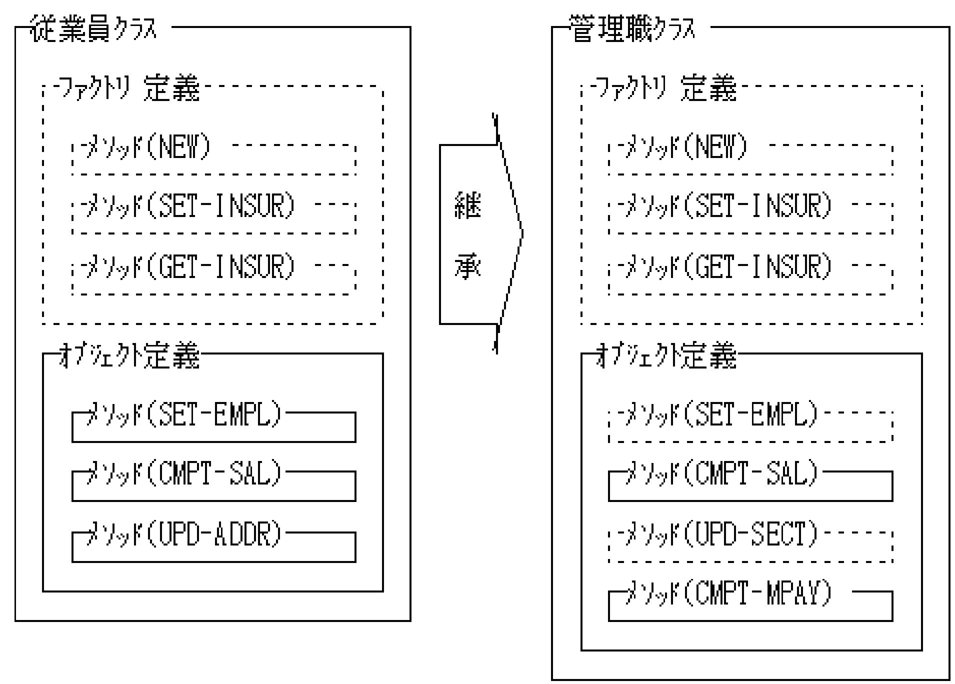
ここでは、適合の概念や規則について、具体例を用いて説明します。

14.4.1 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。しかし、適合関係が不成立となる場合、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いません。したがって、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。

14.4.2 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、"14.6.2 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。しかし、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。しかし、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.
      :
```

14.4.3 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。

14.4.3.1 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。しかし、[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。

14.4.3.2 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.    … [1]
    INVOKE OBJ-2 "CMPT-SAL" RETURNING SALARY.    … [2]
    INVOKE OBJ-3 "CMPT-SAL" RETURNING SALARY.    … [3]
     :
```

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。

14.5 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、"16.6.1 リポジトリファイルと翻訳の手順"を参照してください。

14.5.1 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。ただし、テキスト形式ではありません。したがって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。

14.5.1.1 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する必要があります。

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する必要があります。

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。しかし、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。

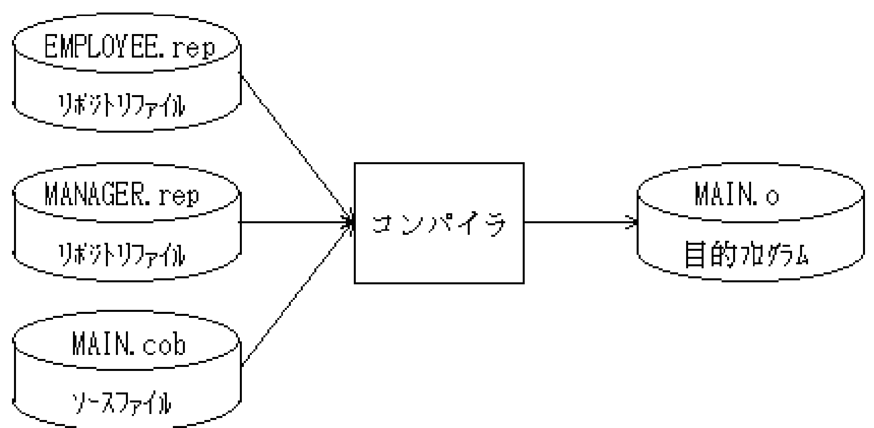
14.5.1.2 適合チェックの実現

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

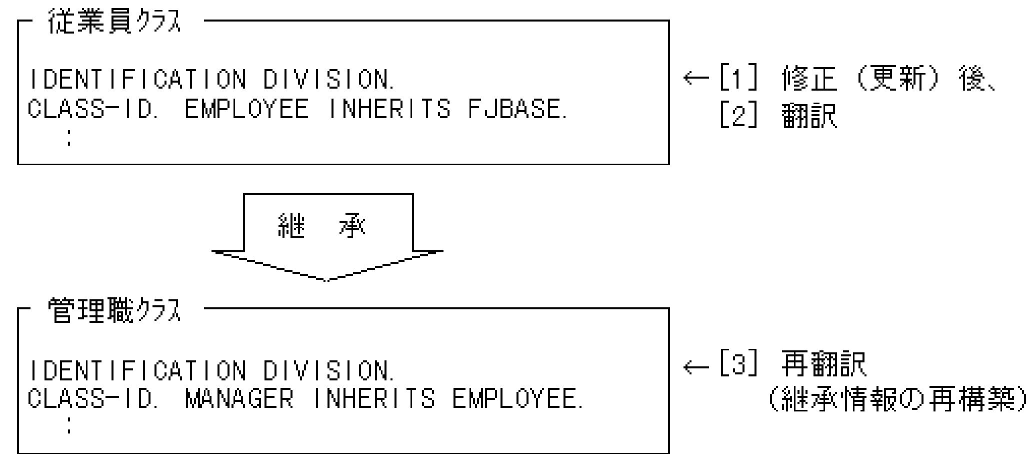
```
IDENTIFICATION DIVISION.
 PROGRAM-ID. MAINPGM.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     CLASS EMPLOYEE
     CLASS MANAGER.
       :
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  OBJ-1        USAGE OBJECT REFERENCE EMPLOYEE.
 01  OBJ-2        USAGE OBJECT REFERENCE MANAGER.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



14.5.2 リポジトリファイル更新の影響

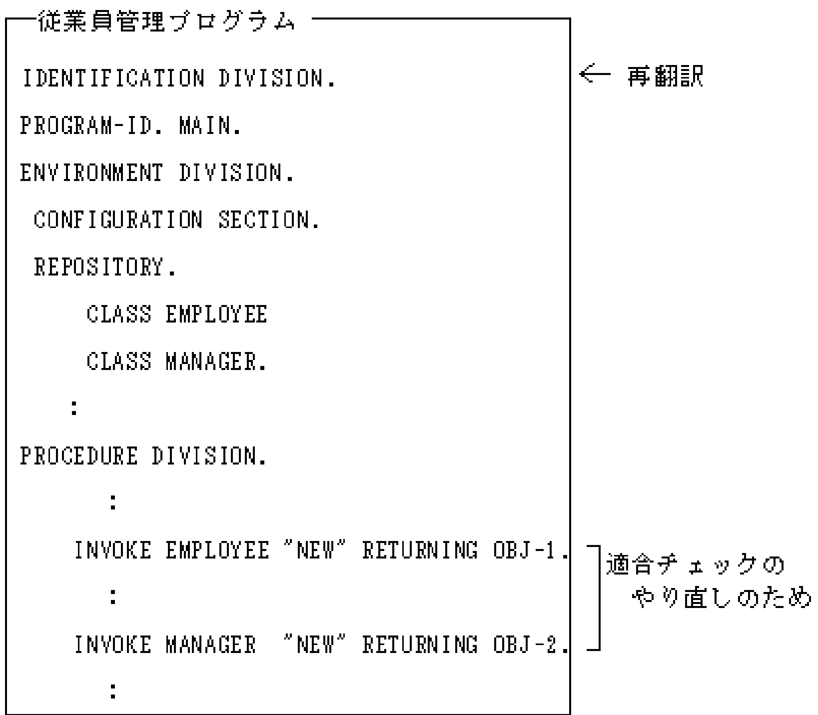
目的プログラムの生成後、親クラスや呼び出すクラスが修正された場合の対応について説明します。

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で、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が修正された場合、子クラスは、何も修正がなくても再翻訳する必要があります。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、再翻訳は不要です。



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、利用者が行う必要があります。そのため、一度構築したクラス定義を修正する場合は、十分注意してください。なお、プロジェクト管理機能を利用すれば、依存関係を最初に登録しておくだけで必要な再翻訳を実行するので修正が容易に行えます。プロジェクト管理機能については、“第16章 オブジェクト指向プログラムの開発と実行”を参照してください。

14.6 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。

14.6.1 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、“14.6.3 定義済みオブジェクト一意名SUPER”を参照してください。

14.6.2 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する必要がある場合にとられます。

```
    :
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。

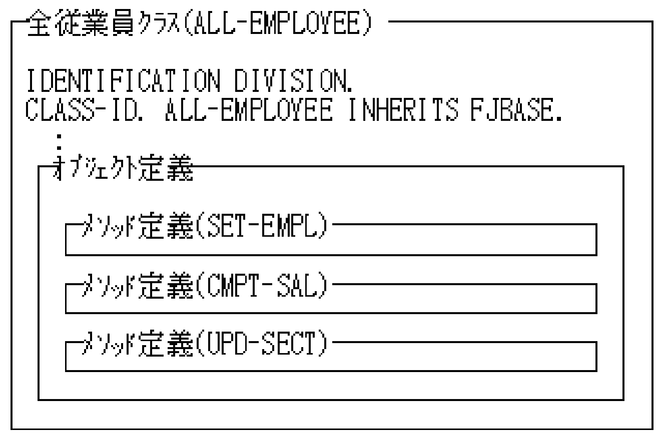
[2] OBJ-2には、従業員クラスに適合するクラスのオブジェクト参照が格納できるため、[1]場合と同様に動的束縛になります。

[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、“14.6.4 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。しかし、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。



継　承　　　継　承

一般従業員ｸﾗｽ(EMPLOYEE)

```
IDENTIFICATION DIVISION.
CLASS-ID. EMPLOYEE
            INHERITS ALL-EMPLOYEE.
```

オブジェクト定義

- SET-EMPL
- CMPT-SAL OVERRIDE
- UPD-SECT
- CMPT-GPAY(残業手当)

管理職ｸﾗｽ(MANAGER)

```
IDENTIFICATION DIVISION.
CLASS-ID. MANAGER
            INHERITS ALL-EMPLOYEE.
```

オブジェクト定義

- SET-EMPL
- CMPT-SAL OVERRIDE
- UPD-SECT
- CMPT-MPAY(特別手当)

従業員管理ﾌﾟﾛｸﾞﾗﾑ

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINPGM.
  :
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  OBJ-1        USAGE OBJECT REFERENCE EMPLOYEE.
 01  OBJ-2        USAGE OBJECT REFERENCE MANAGER.
 01  OBJ-X        USAGE OBJECT REFERENCE ALL-EMPLOYEE.
    :
PROCEDURE DIVISION.
      :
    SET OBJ-X TO OBJ-1.
      :
    SET OBJ-X TO OBJ-2.
      :
給与計算 SECTION.
    INVOKE OBJ-X "CMPT-SAL" RETURNING SALARY.
      :
```

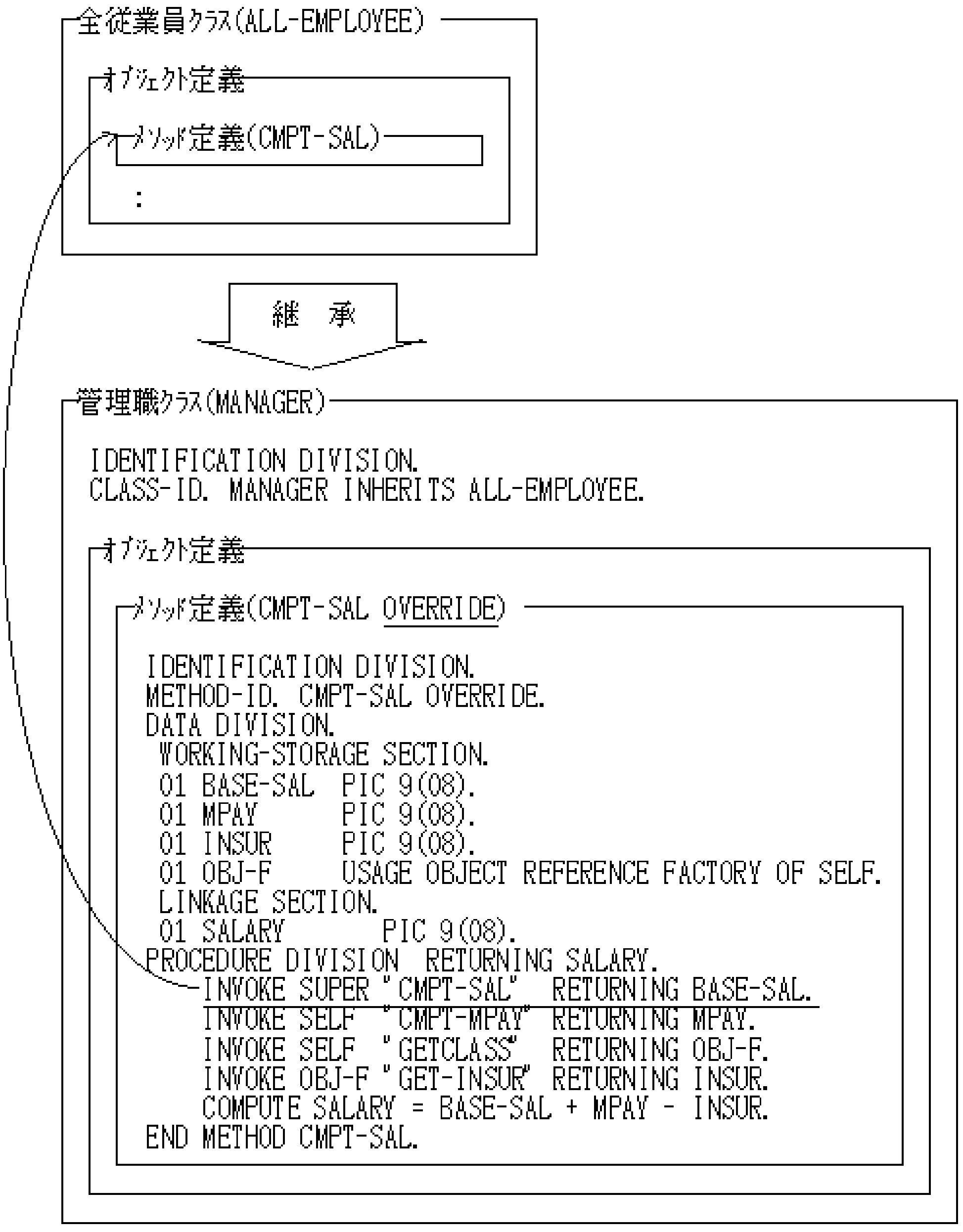
オブジェクトを意識 (一般従業員なのか、管理職なのか)しないで上図のように給与計算処理を行うことができます。つまり、1つのオブジェクト参照項目によって、複数の別定義オブジェクトを操作できたことになります。これを多態と呼びます。

14.6.3 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。



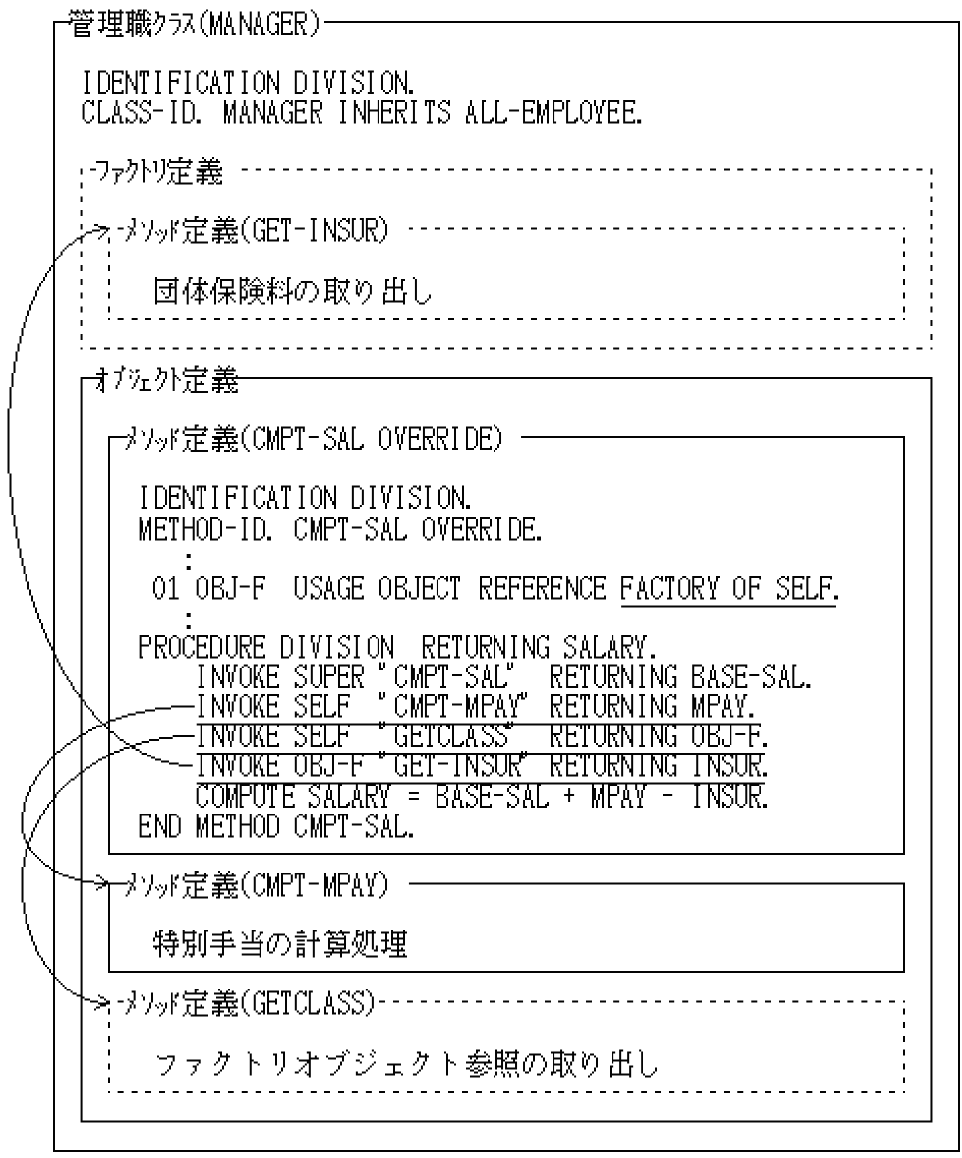
このように、親クラスを表現する場合に、定義済みオブジェクト一意名SUPERが利用されます。

14.6.4 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。

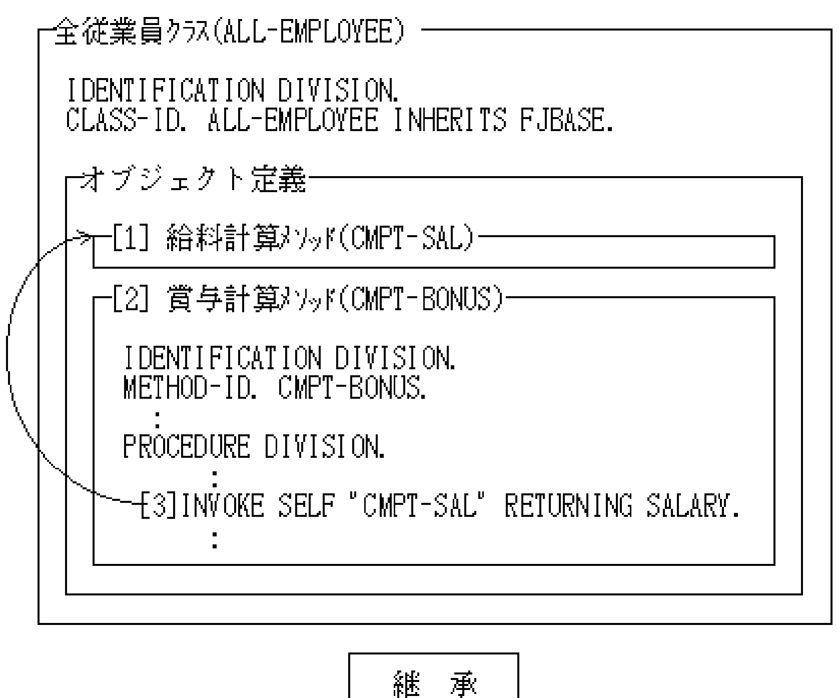
この定義済みオブジェクト一意名SELFの使用方法について説明します。

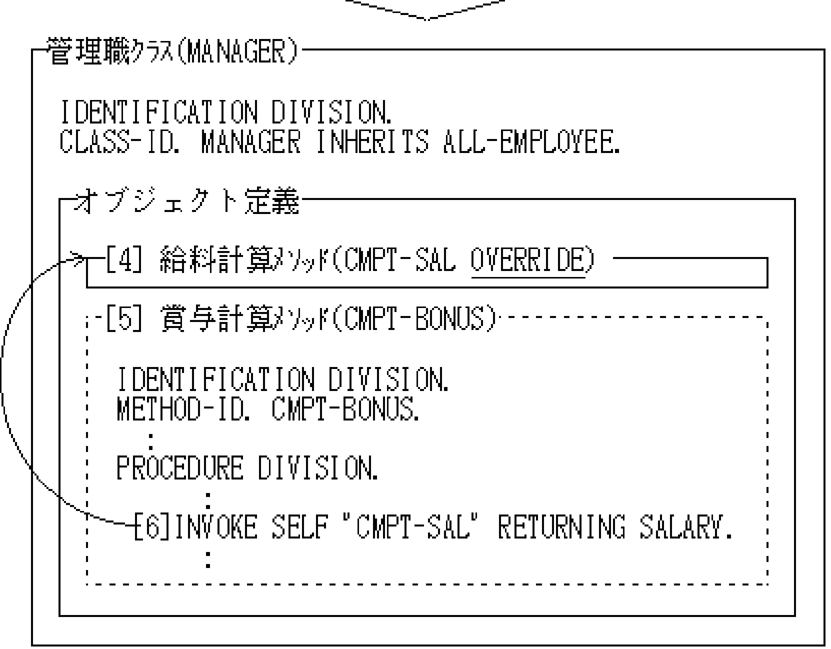
管理職クラスで、給料計算(CMPT-SAL)時、特別手当を求めるために、特別手当の計算メソッド(CMPT-MPAY)を呼び出す場合などに利用することができます。



定義済みオブジェクト一意名SELFの使用時に注意する必要があるのは、呼び出すメソッドは、常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な場合があります。そのとき、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。しかし、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。

14.7 少し進んだ使い方

これまでの説明で、オブジェクト指向プログラミングでの基本的な概念や機能の説明は、すべて完了しました。つまり、ここまでに出てきた機能だけを利用してオブジェクト指向を実現することが可能です。しかし、これら基本機能をより利用(記述)しやすくする機能や、一歩進めた機能などをほかにも多く提供しています。

ここでは、それらの少し進んだ使い方について説明します。

14.7.1 メソッドのPROTOTYPE宣言

通常、クラス定義内に記述するメソッド定義を、物理的に別ファイルに定義することができます。

このとき、クラス定義内には、メソッド名とそのインタフェースだけを定義し、メソッドデータや手続きは別翻訳単位内で定義します。クラス定義内に記述されたメソッドを「PROTOTYPEメソッド」と呼び、別翻訳単位で定義したメソッドを「分離されたメソッド」と呼びます。

```
従業員クラス
IDENTIFICATION DIVISION.
 CLASS-ID. ALL-EMPLOYEE INHERITS FJBASE.
    :
 IDENTIFICATION DIVISION.                      PROTOTYPEメソッド
 METHOD-ID. UPD-SECT PROTOTYPE.                  インターフェースだけを記述
 DATA DIVISION.                                します。
 LINKAGE SECTION.
 01  NEW-SECT     PIC N(20).
PROCEDURE DIVISION  USING NEW-SECT.
  END METHOD UPD-SECT.
    :
```

```
UPD-SECT(分離されたメソッド)
IDENTIFICATION DIVISION.
METHOD-ID. UPD-SECT OF ALL-EMPLOYEE.          → このメソッドを論理的に含む
ENVIRONMENT DIVISION.                            クラス名を指定します。
 CONFIGURATION SECTION.
 REPOSITORY.
     CLASS ALL-EMPLOYEE.                      → 同上
DATA DIVISION.
 WORKING-STORAGE SECTION.                     このメソッド内で使用するデータ
   :                                          を定義します。
  LINKAGE SECTION.
  01  NEW-SECT     PIC N(20).
PROCEDURE DIVISION  USING NEW-SECT.           メソッドの手続きを記述します。
   :
END METHOD UPD-SECT.
```

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。

14.7.2 多重継承

継承については前述しました。しかし、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。

```
IDENTIFICATION DIVISION.
CLASS-ID. CL-1
          INHERITS FJBASE.
 ファクトリ定義
   NEW
 オブジェクト定義
   M1
```

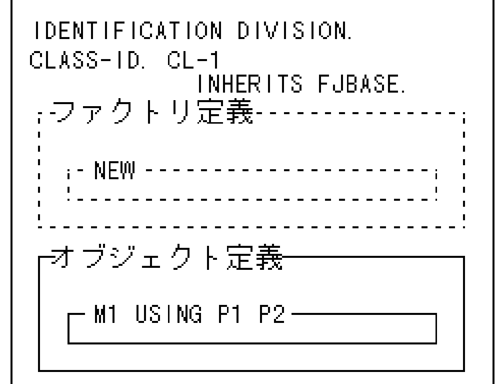
```
IDENTIFICATION DIVISION.
CLASS-ID. CL-2
          INHERITS FJBASE.
 ファクトリ定義
   NEW
 オブジェクト定義
   M2
```

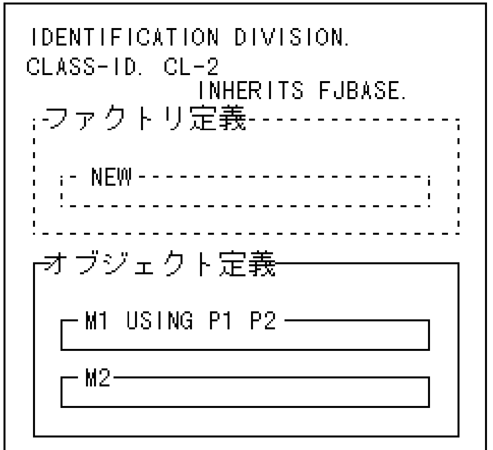
多重継承

```
IDENTIFICATION DIVISION.
CLASS-ID. CL-3 INHERITS CL-1 CL-2.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     CLASS CL-1
     CLASS CL-2.
 ファクトリ定義
   NEW
 オブジェクト定義
   M1
   M2
   M3
```

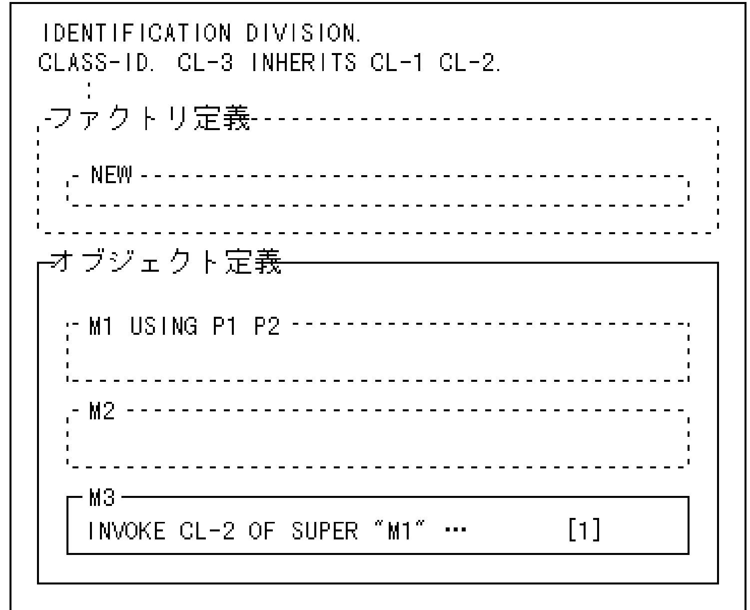
継承の論理については、1つのクラスを継承する場合と同じです。

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は、それらのメソッドのインタフェース(パラメタ)は同じでなければならない」という規約があります(下図を参照してください)。





多重継承



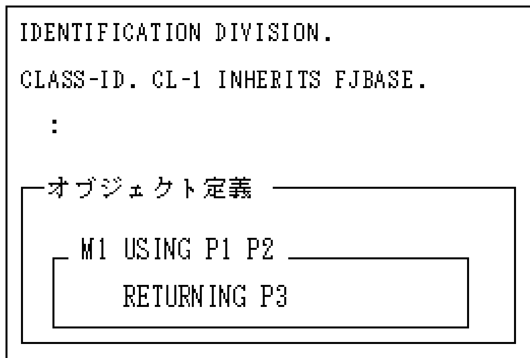
複数の親クラスで同名のメソッドが定義されていた場合、利用者が明にこのメソッドを上書きしないかぎり、INHERITS句に指定されたクラス名の並びを左から検索して、最初に見つかるメソッドが引き継がれます。

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。

14.7.3 行内呼出し

通常、メソッドの呼出しにはINVOKE文を使用します。しかし、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ます。したがって復帰項目を持つメソッドに対してだけ利用できます。



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。

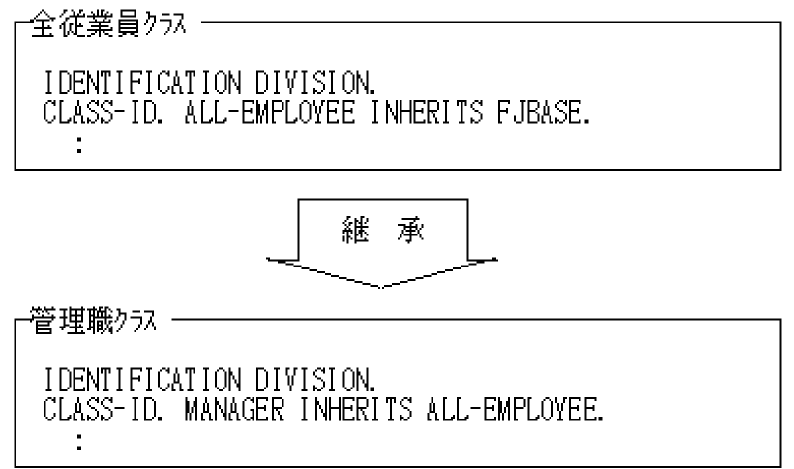
```
    :
WORKING-STORAGE SECTION.
01  OBJ-1 USAGE OBJECT REFERENCE CL-1.
PROCEDURE DIVISION.
       :
    INVOKE OBJ-1 "M1" USING P1 P2
                     RETURNING P3.
    IF P3 = 0 THEN …
       :
```

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 …
       :
```

14.7.4 オブジェクト指定子

適合の規則に違反している場合、翻訳時に実施する適合チェックでエラーとなることがあります。しかし、オブジェクト指定子を利用することで、翻訳時の適合チェックをゆるめ、適合の規則に違反している場合でも問題なく翻訳できるようになります。



たとえば、上のような継承関係があった場合、

従業員管理プログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINPGM.
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、全従業員クラスは管理職クラスの子クラスではありません。したがって、その後、[2]で元の項目に代入しよう(戻そう)とした場合、エラーになります。つまり、格納されているデータでは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. MAINPGM.
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。

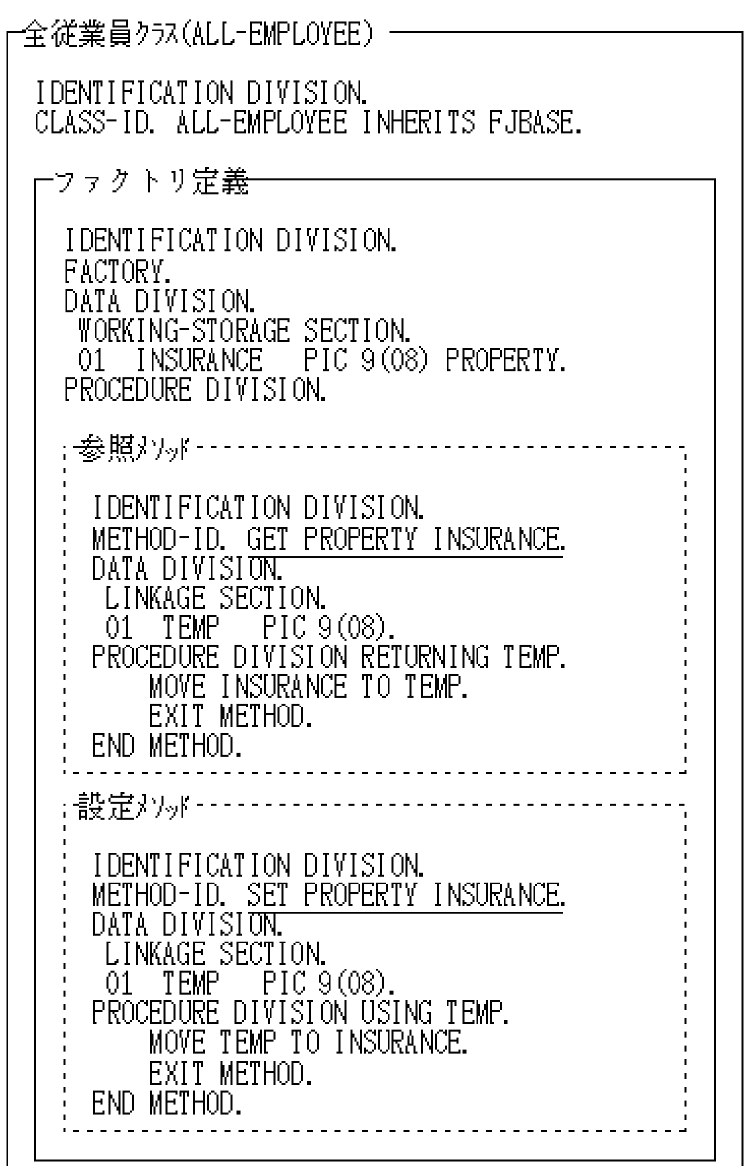
このように、オブジェクト指定子を用いた場合、翻訳時は指定されたクラス名で適合チェックが行われます。しかし、実行時は実際に格納されているオブジェクト参照によって適合チェックが行われます。つまり、適合に違反している場合は、実行時にチェックアウトされます。

14.7.5 PROPERTY句

PROPERTY句を使用することによって、ファクトリデータおよびオブジェクトデータの参照および設定が容易に実現できます。

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

たとえば、全従業員クラスの例で、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。このデータ宣言にPROPERTY句を指定することによって、以下のように暗黙メソッド(ソース記述はなく、論理的に存在するメソッド)が自動生成されます。



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

ただし、プロパティメソッドはINVOKE文を利用して呼び出すことはできません。以下のようにオブジェクトプロパティと呼ばれる一意参照を利用します。

従業員管理プログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINPGM.
      :
PROCEDURE DIVISION.
      :
団体保険料の設定処理
    MOVE 保険額 TO INSURANCE OF ALL-EMPLOYEE ←設定メソッドの呼出し
    MOVE 保険額 TO INSURANCE OF MANAGER.     ←設定メソッドの呼出し
      :
団体保険料の取り出し処理
    EVALUATE EMP-ID
    WHEN ID-EMPL
      MOVE INSURANCE OF ALL-EMPLOYEE TO 保険←参照メソッドの呼出し
    WHEN ID-MAN
      MOVE INSURANCE OF MANAGER  TO 保険額 ←参照メソッドの呼出し
    END-EVALUATE.
      :
```

参照メソッドを呼び出すか、設定メソッドを呼び出すかは、オブジェクトプロパティが送出し側に指定されたか、受取り側に指定されたかによって決定されます。

また、プロパティメソッドにプラスアルファの処理を持たせたい場合には、利用者がプロパティメソッドを明示定義することもできます。この場合、データにPROPERTY句を指定する必要はありません。

```
全従業員クラス
IDENTIFICATION DIVISION.
CLASS-ID. ALL-EMPLOYEE INHERITS FJBASE.
  ファクトリ定義
  IDENTIFICATION DIVISION.
  FACTORY.
  DATA DIVISION.
   WORKING-STORAGE SECTION.
    01  INSURANCE   PIC 9(08).
   PROCEDURE DIVISION.
    参照メソッド
    IDENTIFICATION DIVISION.
    METHOD-ID. GET PROPERTY INSURANCE.
    DATA DIVISION.
     LINKAGE SECTION.
      01  OUT-DATA    PIC 9(08).
    PROCEDURE DIVISION RETURNING OUT-DATA.
        MOVE INSURANCE TO OUT-DATA.
        EXIT METHOD.
    END METHOD.
    設定メソッド
    IDENTIFICATION DIVISION.
    METHOD-ID. SET PROPERTY INSURANCE.
    DATA DIVISION.
     LINKAGE SECTION.
      01  IN-DATA     PIC 9(08).
    PROCEDURE DIVISION USING IN-DATA.
        MOVE IN-DATA TO INSURANCE.
        DISPLAY "団体保険料が更新されました".    ← 追加した処理
        EXIT METHOD.
    END METHOD.
```

利用者定義の
プロパティメソッド

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、両方を明に定義する必要があります。

14.7.6 初期化処理メソッドと終了処理メソッド

オブジェクトインスタンスの生成は“NEW”メソッドを呼ぶことで行われます。一方、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(“14.2.2 オブジェクトの寿命”を参照)ことを自動的に判断して行います。

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文で呼ぶ必要はなく、当該メソッドを上書き(上書きについては、“14.3.3 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。しかし、実際の処理は行われません。

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.
END CLASS I-F-SAMPLE.
```

注意

終了処理メソッド_FINALIZE使用する場合、_FINALIZEメソッド中にSTOP RUN文を記述してはいけません。また、_FINALIZEから呼び出されるプログラムまたはメソッド中にもSTOP RUN文を記述してはいけません。

14.7.7 間接参照クラス

呼び出したメソッドの復帰値がオブジェクト参照項目だった場合、ソース上には現れないクラス、つまり暗黙的な参照クラスが必要となる場合があります。この暗黙的に参照するクラスのことを間接参照クラスと呼びます。

ここでは、間接参照クラスが顕著に現れるパターンとして、行内呼出しの入れ子を例に使用方法を説明します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. P.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     CLASS A.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 OBJ-A USAGE OBJECT REFERENCE A.
 01 RET   PIC X(4).
PROCEDURE DIVISION.
       :
 ⇨ MOVE OBJ-A::"M-A"::"M-B" TO RET.
       :
```

```
IDENTIFICATION DIVISION.
CLASS-ID. A.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     CLASS B.
  METHOD-ID. M-A.
  DATA DIVISION.
   LINKAGE SECTION.
   01 OBJ-B USAGE OBJECT REFERENCE B.
  PROCEDURE DIVISION RETURNING OBJ-B.
        :
```

```
IDENTIFICATION DIVISION.
CLASS-ID. B.
  METHOD-ID. M-B.
  DATA DIVISION.
   LINKAGE SECTION.
   01 RET PIC X(4).
  PROCEDURE DIVISION
            RETURNING RET.
        :
```

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する必要があります。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。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。

14.7.8 相互参照クラス

実行時、複数のオブジェクトインスタンスを結びつけたい場合、つまり、オブジェクトデータ中にオブジェクト参照項目を定義したい場合があります。このような場合に、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。

14.7.8.1 相互参照パターン

相互参照のパターンには、以下の3つがあります。

- 自クラスの相互参照
- 他クラスとの直接相互参照
- 他クラスとの間接相互参照

それぞれのパターンについて、以下に具体的に説明します。

自クラスの相互参照

自クラスのオブジェクトインスタンスをリスト構造で管理するような場合、オブジェクトデータ中に自クラスを保持するオブジェクト参照項目を宣言します。

```
IDENTIFICATION DIVISION.
CLASS-ID. A INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    CLASS FJBASE.
OBJECT.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 FWCH     OBJECT REFERENCE A.
 01 BWCH     OBJECT REFERENCE A.
 01 OBJ-DATA PIC X(20).
PROCEDURE DIVISION.
   :
```

図14.3 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する必要があります。ただし、自クラスについては宣言してはいけません。宣言した場合、翻訳時エラーとなるので、注意してください。

他クラスとの直接相互参照

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。

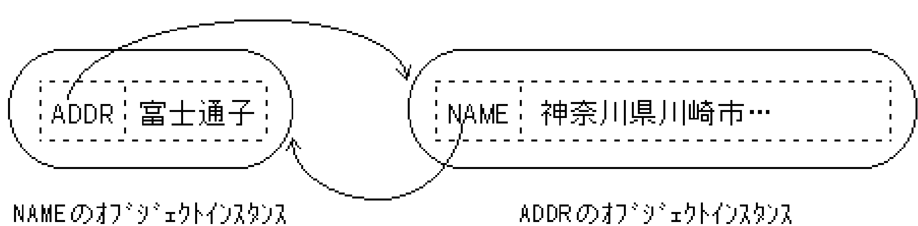
名前クラス

```
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14.4 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.
  :
```

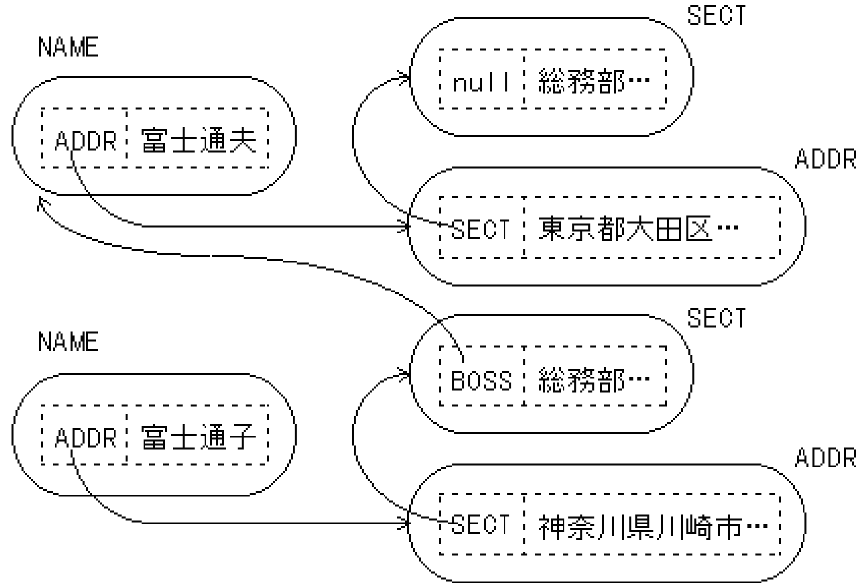
住所クラス

```
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14.5 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。

14.7.8.2 相互参照クラスの翻訳

前述のとおり、相互参照クラスには大きく3つのパターンがあります。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する必要があります。

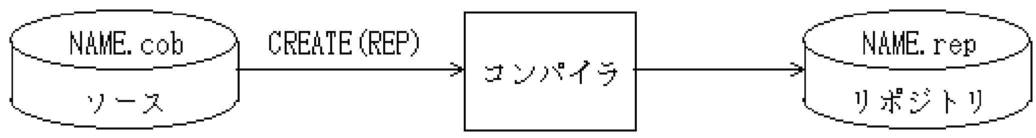
以下、直接相互参照(名前クラスと住所クラス)の場合を例に説明します。

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する必要があり、その際に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。

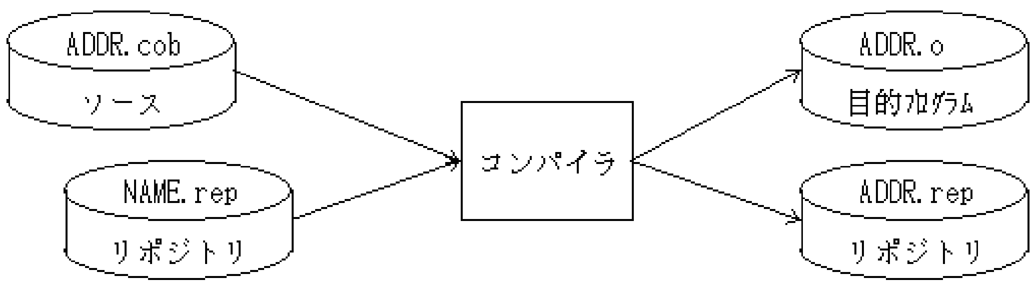
ステップ1 ： 名前クラスのリポジトリを生成

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。



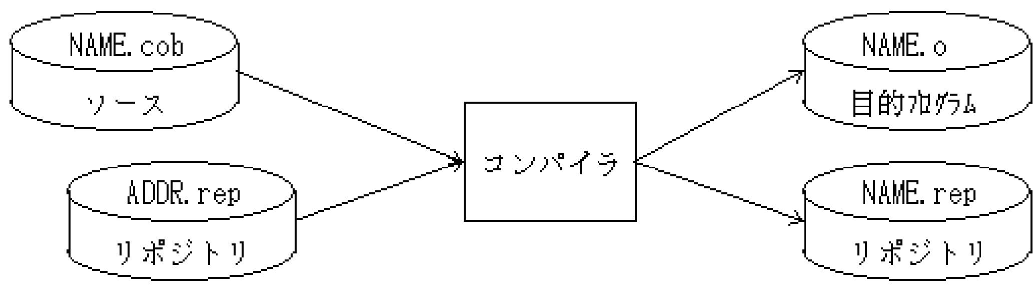
ステップ2 ： 生成したリポジトリを利用して、住所クラスを翻訳

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、どちら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、どちらかのクラスのリポジトリだけを生成し、あと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る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。

14.7.8.3 相互参照クラスのリンク

実行形式を静的リンク構造や動的プログラム構造で構築する場合、通常のクラス定義と同じようにリンクします。ただし、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、特別な考慮が必要となります。なお、(相互参照関係にあるクラスを1つの共用オブジェクトファイル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共用オブジェクトファイル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。

名前クラスの共用オブジェクトをリンクするためには、住所クラスの共用オブジェクトファイルが必要です。住所クラスの共用オブジェクトファイルを生成するには住所クラスをリンクする必要があり、その際、名前クラスの共用オブジェクトファイルが必要です。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、それぞれのクラスの共用オブジェクトファイルを作成するときに相互参照関係にあるクラスをリンクしないようにします(ステップ1)。実行可能ファイルをリンクする段階で相互参照関係を持つクラスをリンクするようにします(ステップ2-1)。また実行可能ファイルを作成しない場合は、ステップ1で作成した共用オブジェクトファイルを使用して再リンクする必要があります。再リンクするときに相互参照関係にあるクラスをリンクします(ステップ2-2)。

ステップ1 ： 名前クラスと住所クラスの共用オブジェクトファイルを作成

名前クラスと住所クラスの共用オブジェクトファイルを作成します。このときには相互参照関係にあるクラスはリンクしません。

```
$ cobol -dy -G -o libNAME.so NAME.o
$ cobol -dy -G -o libADDR.so ADDR.o
```

ステップ2-1 ： 実行可能ファイルを作成する場合

実行可能ファイルを作成します。このときに名前クラスと住所クラスの共用オブジェクトファイルをリンクします。

```
$ cobol -o sample -lNAME -lADDR sample.o
```

ステップ2-2 ： 実行可能ファイルを作成しない場合

実行可能ファイルを作成しない場合はステップ1で作成した名前クラスと住所クラスの共用オブジェクトファイルを使用して再リンクします。

```
$ cobol -dy -G -o libNAME.so -lADDR NAME.o
$ cobol -dy -G -o libADDR.so -lNAME ADDR.o
```

14.7.8.4 相互参照クラスの実行

実行の際には、特に考慮すべきことはありません。

通常のクラス定義と同じように実行することができます。

第15章　オブジェクト指向プログラミング機能～さらに進んだ使い方～

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

15.1 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。

15.1.1 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。

15.1.2 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)を定義します。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。
  - 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.
```

- 次の場合、例外を発生させた文によって以下のように動作します。

  a. 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。

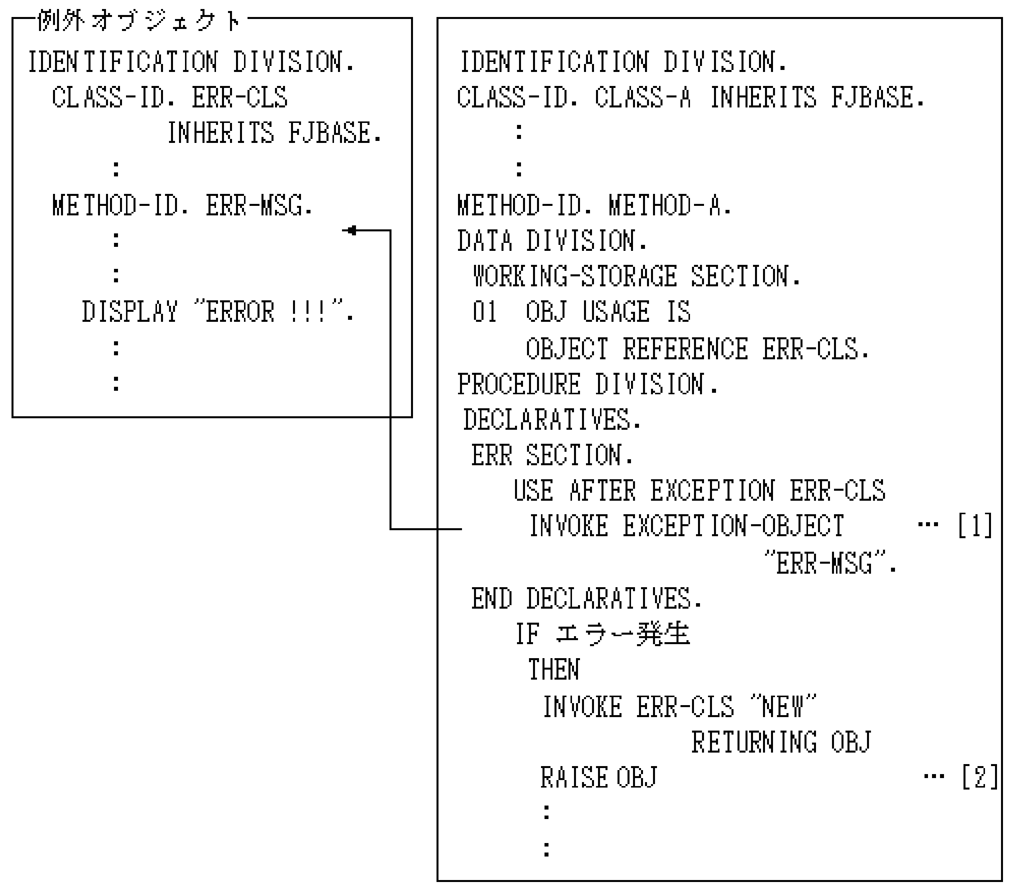
b. 例外オブジェクトが、NULLオブジェクトなど例外手続きを実行できない。

- 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。

15.1.3 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。

15.1.4 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。

```
例外オブジェクト
IDENTIFICATION DIVISION.
  CLASS-ID. ERR-CLS
          INHERITS FJBASE.
      :
  METHOD-ID. ERR-MSG.
      :
      :
    DISPLAY "ERROR !!!".
      :
      :
```

```
 IDENTIFICATION DIVISION.
  PROGRAM-ID. PROGRAM-A.
     :
     :
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  OBJ USAGE IS
     OBJECT REFERENCE CLASS-A.
PROCEDURE DIVISION.
 DECLARATIVES.
  ERR SECTION.
    USE AFTER EXCEPTION ERR-CLS
     INVOKE EXCEPTION-OBJECT     … [2]
                    "ERR-MSG".
 END DECLARATIVES.
    INVOKE CLASS-A "NEW"
                    RETURNING OBJ.
    INVOKE OBJ "METHOD-A".
END PROGRAM PROGRAM-A.
```

```
 IDENTIFICATION DIVISION.
  CLASS-ID. CLASS-A INHERITS FJBASE.
     :
     :
METHOD-ID. METHOD-A.
DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  OBJ USAGE IS
     OBJECT REFERENCE ERR-CLS.
PROCEDURE DIVISION RAISING ERR-CLS.
      :
    IF エラー発生
     THEN
      INVOKE ERR-CLS "NEW"
                    RETURNING OBJ
      EXIT METHOD RAISING OBJ       … [1]
      :
      :
      :
```

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。

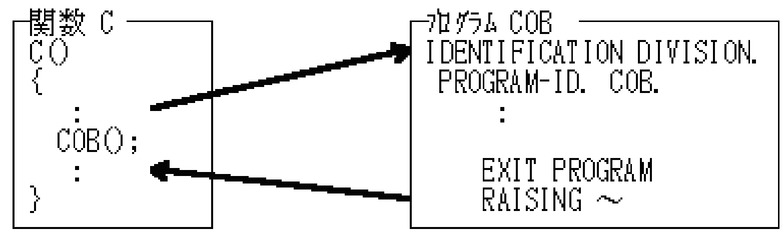
注意

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。

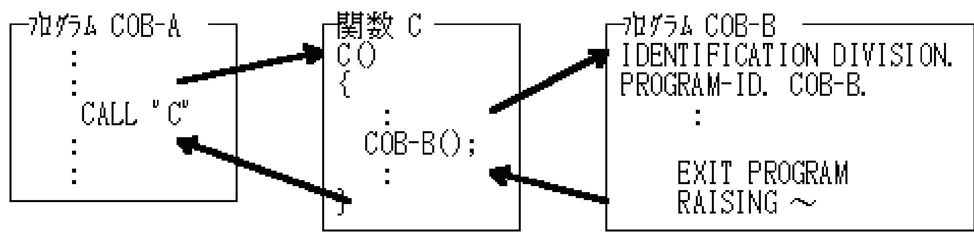
1. 主プログラム
2. 他言語プログラムから呼び出されたCOBOLプログラム(下図a)

   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b)の場合、EXIT文で例外条件を発生させることができます。

a. 例外条件が発生しません。



b. 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。



15.2 C++プログラムとの連携

ここでは、COBOLからC++のオブジェクトを操作する方法について説明します。

15.2.1 概要

ここでは、COBOLと他言語との連携のうち、C++のオブジェクトを操作する方法について説明します。C++プログラムとの連携では、内部的にCまたはC++の関数呼出しを利用するので、“9.3 C言語プログラムとのリンク”の知識を前提とします。

15.2.2 C++連携の方法

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。しかし、次の方法を用いて、C++のオブジェクトを操作することができます。

1. 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. 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.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、C++のクラスをCOBOLのクラスと同じように操作できるようになります。つまり、COBOLのINVOKE文により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。

15.2.3 C++連携の概要

ここでは、C++連携の概要について説明します。

15.2.3.1 COBOLおよびC++でのクラスの対応

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C++のオブジェクトは"表15.1 COBOLおよびC++のクラスの対応"のように対応付けられます。

表15.1 COBOLおよびC++のクラスの対応

概念	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

15.2.3.2 処理の概要

C++のオブジェクトを操作するために、COBOLおよびC++でインタフェースプログラムを作成します。

COBOL側

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。

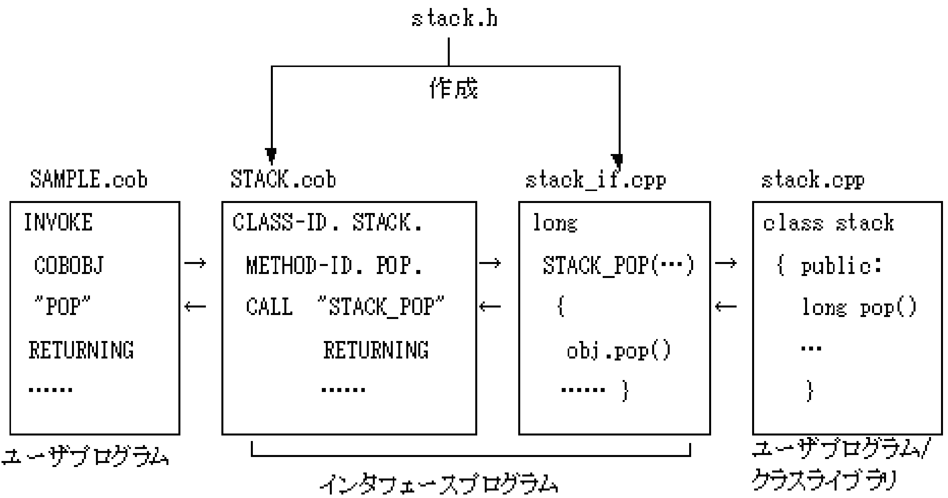
C++側

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、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、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15.1 インタフェースプログラムのイメージ"にインタフェースプログラムのイメージを示します。"STACK.cob"と"stack_if.cpp"はクラス定義(stack.h)から作成します。COBOLプログラムのINVOKE文は、COBOL、C++のインタフェースプログラムを中継して、最終的に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15.1 インタフェースプログラムのイメージ



15.2.3.3 インタフェースプログラムの仕組み

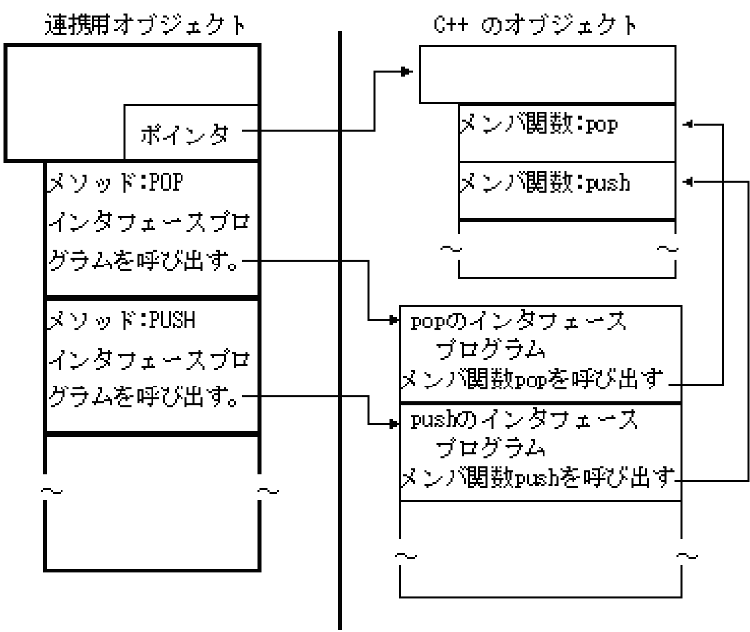
C++のオブジェクトを操作する仕組みは以下のようになります。

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、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。
- 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。

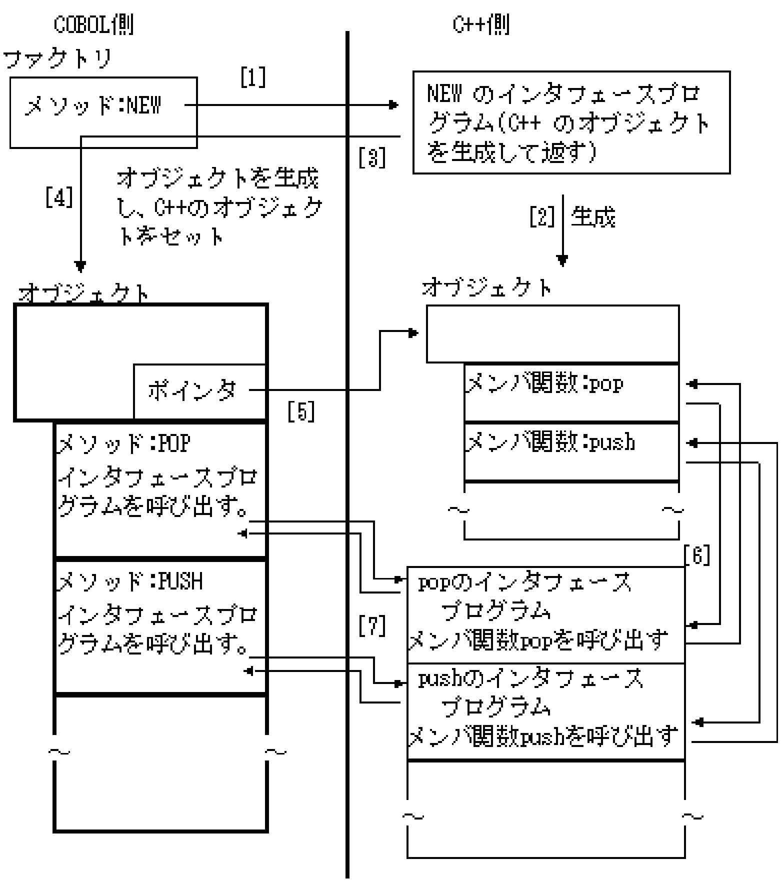
図15.2 インタフェースプログラムの構造



実行時の動き

実行時の制御の流れを"図15.3 実行時の制御の流れ"に示します。

図15.3 実行時の制御の流れ



COBOL側で連携用に定義したクラスのNEWメソッドが呼び出されると、NEWメソッドはC++のインタフェースプログラムを呼び出します([1])。C++のインタフェースプログラムは、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C++のインタフェースプログラムを呼び出します([5])。C++のインタフェースプログラムは、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])。

15.2.4 C++連携のプログラム手順

C++連携のプログラム手順を以下に示します。

1. 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. C++側のインタフェースプログラムを定義します。

以下のクラス定義を基に説明します。

クラス定義の例

```
class stack {
public:
unsigned long pntr;
long pop();
void push(long val);
private:
long data[100];
} ;
```

15.2.4.1 C++で定義されているクラスを調べる

まず、COBOLで操作するC++のクラスを調べます。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

15.2.4.2 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。

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で利用不可となっている文字が使われているなどの制約がないかぎり、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。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、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、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、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。
- C++のオブジェクト生成のインタフェースプログラムを呼び出し、C++側で生成され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、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、PUSHを定義します。
また、C++側のオブジェクトの削除用のメソッドDELETE-OBJを定義します。
DELETE-OBJは、オブジェクト削除用のインタフェースプログラムを呼び出します。

15.2.4.3 C++側での定義

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、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、C++側で生成したオブジェクトを削除します。

“表15.2 例題で定義されるインタフェースプログラム”に、例題で定義されるインタフェースプログラムとクラス定義の関係を示します。

表15.2 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	C++のインタフェースプログラム	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	

15.2.5 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C++のオブジェクトを操作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.
        :
```

15.2.6 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“C++連携のCOBOLのクラス定義”および“C++連携のC++側のインタフェースプログラム”に示します。

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 USE-VALUE TO CPP-OBJ-POINTER.
     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.
```

C++連携の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;
}
```

15.3 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。

15.3.1 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あとで参照したり、ほかのプログラムで参照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。

15.3.2 概要

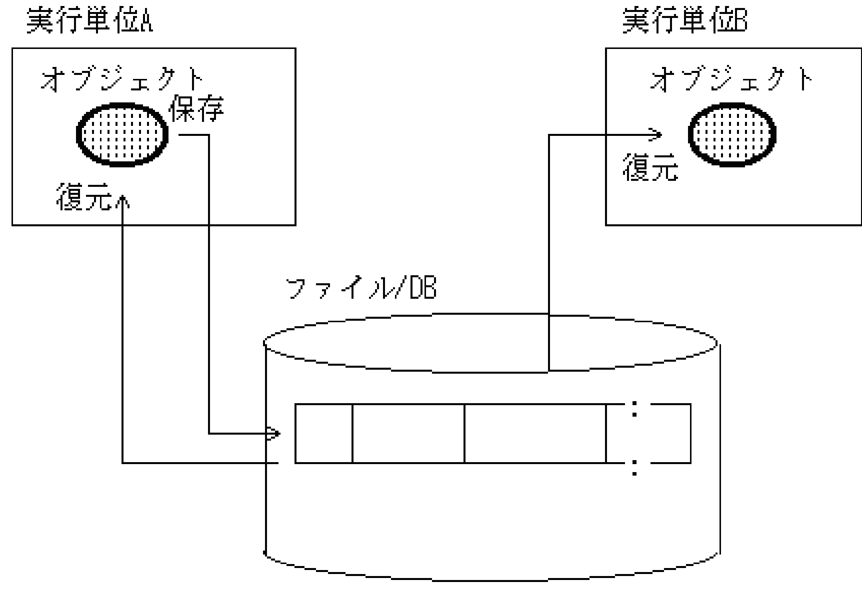
オブジェクトの永続化は、COBOLの実行中に生成したオブジェクトを外部記憶装置に保存し、別の実行単位で読み戻すことで実現します。保存のための記憶装置には、データベース、ファイルなどが用いられます([参照]"図15.4 永続オブジェクトの流れ")。

オブジェクトには適当な識別子を付け、その識別子を元に保存/復元を行います。

参考

オブジェクトの保存/復元は、本質的にはオブジェクトデータの保存/復元によって実現されます。しかし、実際には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ています。そのため本節の説明では、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15.4 永続オブジェクトの流れ

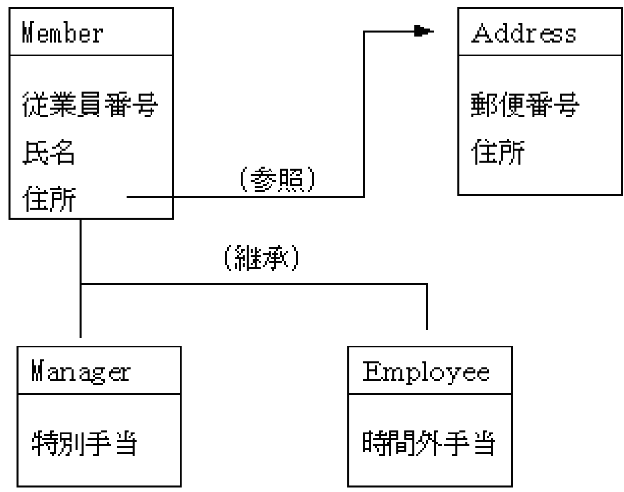


ここでは、索引ファイルを利用してオブジェクトの永続化を行う方法について説明します。

15.3.3 クラス構造の例

本節の説明で使用するクラス構造は、"図15.5 クラス構造の例"のようになっています。以下の説明では、この図を基に説明するため、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15.5 クラス構造の例



クラスには、従業員全体を表すMember、管理職を表すManager、一般従業員を表すEmployee、住所を表すAddressがあります。

- Memberは、オブジェクトデータとして、従業員番号、氏名、住所を持ちます。住所はAddressオブジェクトに関連付けられます。
- Managerは、Memberを継承し、オブジェクトデータとして特別手当を持ちます。
- Employeeは、Memberを継承し、オブジェクトデータとして時間外手当を持ちます。
- Addressは、オブジェクトデータとして、郵便番号、住所を持ちます。Addressクラスは、ほかのクラスとの継承関係はありません。

15.3.4 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。

15.3.4.1 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

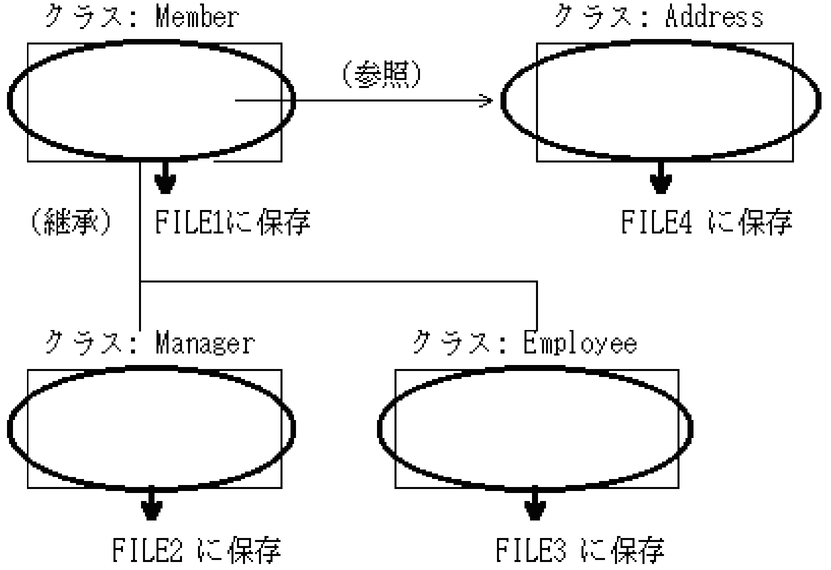
オブジェクトと保存ファイルの対応の基本は、クラスとファイルを一対一に対応付けることです。この例では、MemberをFILE1に、ManagerをFILE2に、EmployeeをFILE3に、AddressをFILE4に保存することになります([参照]“図15.6 クラスごとに異なるファイルに保存する場合”)。

Managerクラスのオブジェクトを保存する場合、Manager固有のオブジェクトデータをFILE2に、Member固有のオブジェクトデータはFILE1に保存します。

Memberオブジェクトを復元する場合、最初にFILE1からMember固有のオブジェクトデータを読み込みます。そして、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15.6 クラスごとに異なるファイルに保存する場合



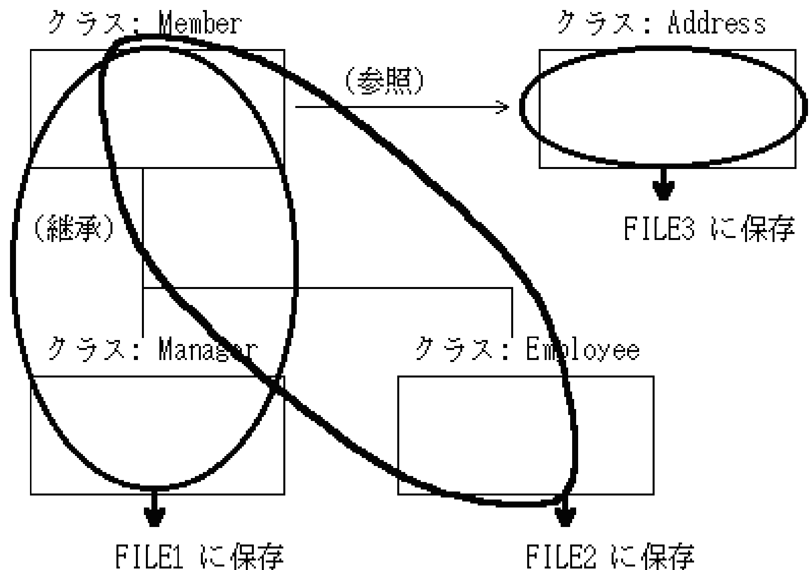
親クラスのオブジェクトデータも含めて1つのファイルに保存する方法

Memberクラスが純粋に抽象クラスとして定義されている場合、Memberクラスとして独立したオブジェクトは存在しません。この場合、Managerクラスの保存で、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することをおすすめします。このようにすると、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15.7 親クラスのオブジェクトデータも含めて1つのファイルに保存する方法")。

Managerクラスのオブジェクトを復元する場合、FILE1からオブジェクトデータを読み込み、Managerオブジェクトを生成して返します。しかし、Managerであるか、Employeeであるかは、クラスが持っている情報であり、オブジェクトを復元してはじめてわかる情報です。たとえば、従業員番号の情報だけでは、FILE1とFILE2のどちらのファイルからオブジェクトデータを読み戻したらよいのか判断できません。

この方法は、保存するクラスが1つしかない場合には有効な方法です。しかし、"図15.7 親クラスのオブジェクトデータも含めて1つのファイルに保存する方法"の説明のとおり、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する必要があります。

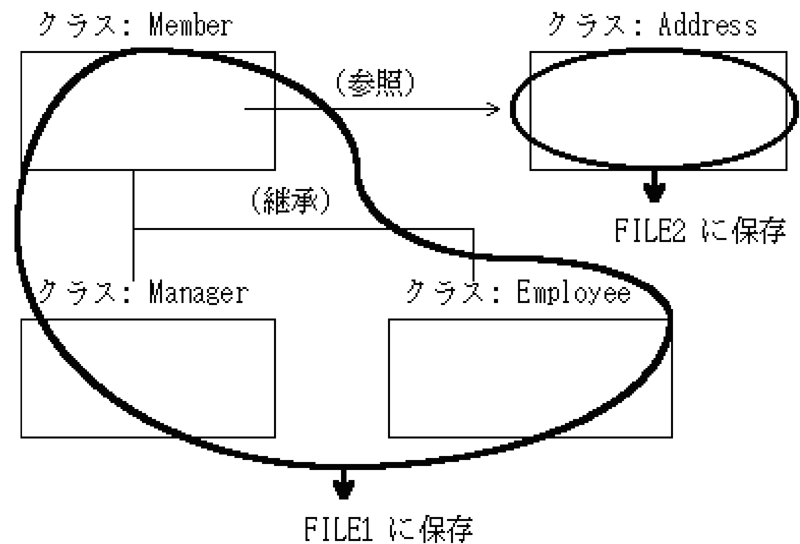
図15.7 親クラスのオブジェクトデータも含めて1つのファイルに保存する方法



同じ親クラスを持つクラスを同じファイルに保存する方法

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([参照] "図15.7 親クラスのオブジェクトデータも含めて1つのファイルに保存する方法")。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できるようになります。

図15.8 同じ親クラスを持つクラスを同じファイルに保存する方法



以上の3つの方法のどれが適しているかは、アプリケーションの性質に依存します。

ここでは、"図15.8 同じ親クラスを持つクラスを同じファイルに保存する方法"に従って説明します。

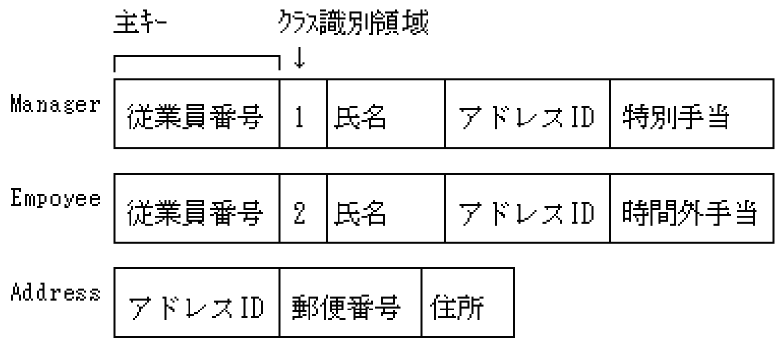
15.3.4.2 索引ファイルの定義

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ある必要があります。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。“図15.9 索引ファイル中のレコード”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。“図15.9 索引ファイル中のレコード”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15.9 索引ファイル中のレコード



アドレスIDは、アドレスオブジェクトを一意に識別するための識別子です。

15.3.5 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。

15.3.5.1 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。

15.3.5.2 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

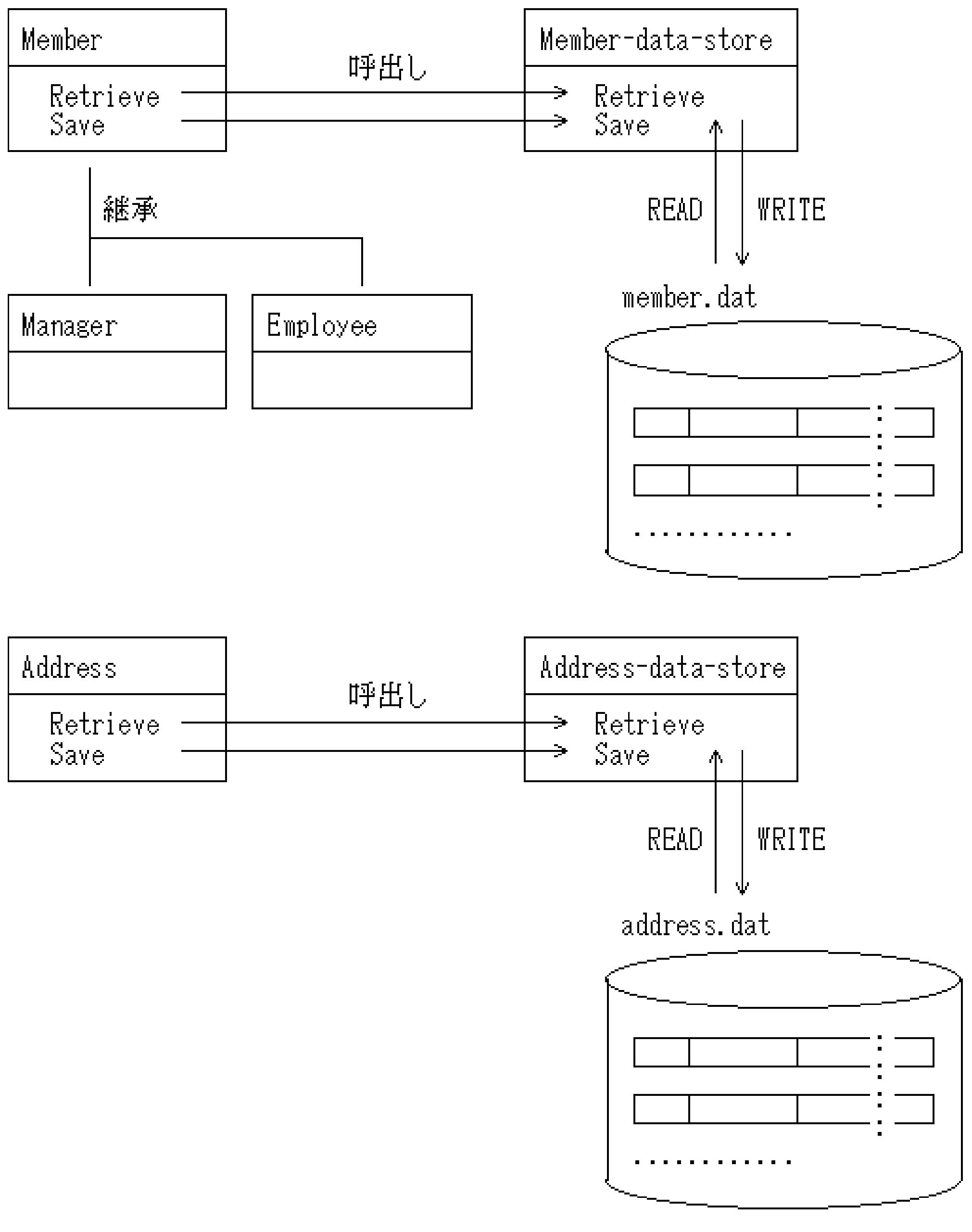
Saveメソッドはオブジェクトを保存するメソッドです。実際には、自分自身を引数として、対応する索引ファイル操作クラスのSaveメソッドを呼び出します。Saveメソッドは、親クラスで定義すれば十分で、個々のクラスに定義する必要はありません。

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15.3 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15.10 索引ファイル操作クラスとの関係"に示します。

表15.3 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15.10 索引ファイル操作クラスとの関係



15.3.6 処理の流れ

保存/復元の処理の流れを"図15.10 索引ファイル操作クラスとの関係"に従って説明します。

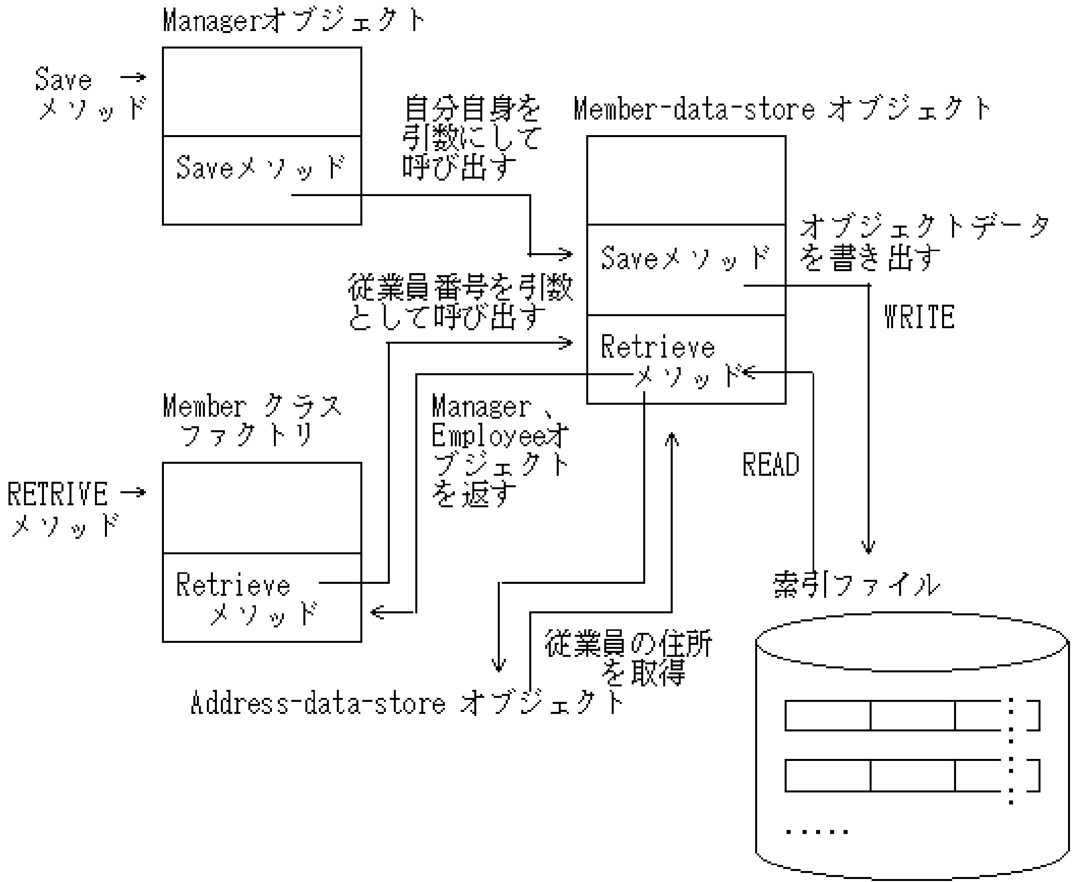
保存

Managerオブジェクトに対してSaveメソッドが呼ばれると、Saveメソッドは自分自身を引数にします。そして、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ます(この場合Managerクラスになる)。そして、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、副キーを調べます。そして、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ます。Address-data-storeオブジェクトのRetrieveメソッドを呼び出します。そして、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15.11 保存/復元の処理の流れ"では省略)。

図15.11 保存/復元の処理の流れ



15.4 特殊クラス

ここでは、特殊クラスを使用したアプリケーションの概要および使用方法について説明します。

15.4.1 クラスライブラリ連携(*COB-BINDTABLE)

ここでは、業務用の各種クラスライブラリとの連携で、特殊クラス「*COB-BINDTABLE」を使用して、COBOLの変数をクラスライブラリに登録および通知する方法について説明します。

15.4.1.1 概要

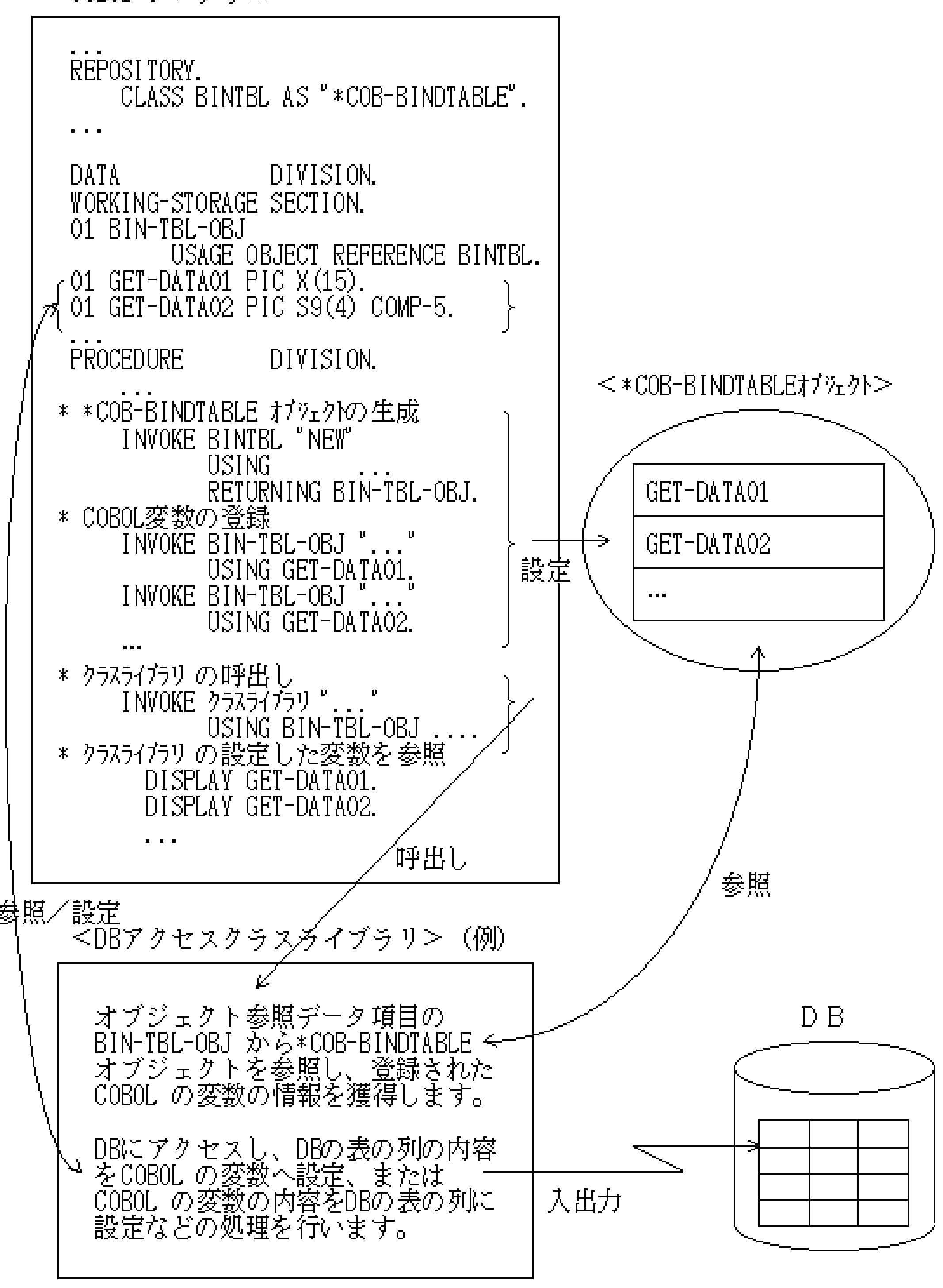
COBOLアプリケーションの作成のために、業務用の各種クラスライブラリが提供されます。各種クラスライブラリを利用することで、データベースのアクセスなどの業務用アプリケーションを簡単に作成することができます。

クラスライブラリのメソッドの中には、使用するCOBOLの変数の情報をメソッド呼出し時にアプリケーションから通知する必要があるものがあります。

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

COBOLの変数をクラスライブラリに通知するには、特殊クラス*COB-BINDTABLEを使用します。

以下にデータベースアクセス用クラスライブラリを例とした概要図を示します。



特殊クラス*COB-BINDTABLEを使用するアプリケーションの処理の流れおよび補足事項を以下に示します。

1. 使用するCOBOLの変数の宣言
   どのような変数を宣言するかは、各種クラスライブラリ添付のマニュアルまたはヘルプを参照してください。
2. *COB-BINDTABLEオブジェクトの生成
   *COB-BINDTABLEクラスのファクトリメソッドのNEW メソッドを使用して*COB-BINDTABLEオブジェクトの生成を行います。
3. *COB-BINDTABLEオブジェクトへCOBOL の変数の登録
   各種クラスライブラリでの変数の規則や使用方法により、その目的に一致するメソッドが用意されています。各種クラスライブラリ添付のマニュアルまたはヘルプも参照して使用するメソッドを決定してください。
4. クラスライブラリのメソッドの呼出し
   各種クラスライブラリのメソッドの詳細については、各種クラスライブラリ添付のマニュアルまたはヘルプを参照してください。
   各種クラスライブラリのメソッドの仕様に従い、USING の引数に*COB-BINDTABLEのオブジェクト参照データ項目を指定してください。
5. クラスライブラリの出力した変数(情報)の処理

15.4.1.2 *COB-BINDTABLEクラスの説明

ここでは、*COB-BINDTABLEクラスのメソッドについて説明します。

15.4.1.2.1 *COB-BINDTABLEクラスのファクトリメソッド

ファクトリメソッドには、以下のメソッドがあります。

NEWメソッド

NEWメソッドは*COB-BINDTABLEのオブジェクトの生成を行います。

NEWメソッドの機能および引数を以下に示します。

メソッド名	引数	引数の型	意味
NEW	【USING 第1】 最大登録データ項目数	PIC S9(9) BINARY | COMP-5	USING の第1引数で指定されたデータ項目の値を登録データ項目数の最大個数としてオブジェクトを作成します。 また、そのオブジェクトのオブジェクト参照データを返却します。
	【RETURNING】 作成オブジェクト	USAGE OBJECT REFERENCE 特殊クラス名	

使用規則

登録するデータ項目の個数の最大値を第1引数で指定する必要があります。

使用例

```
  :
 REPOSITORY.
     CLASS  BINTBL AS "*COB-BINDTABLE".
  :
 DATA            DIVISION.
 WORKING-STORAGE SECTION.
 01 OBJ-SPCL USAGE OBJECT     REFERENCE  BINTBL.
 01 ELM-NO   PIC S9(9) COMP-5.
  :
として
     MOVE 20 TO ELM-NO.
     INVOKE  BINTBL "NEW"    USING ELM-NO
                             RETURNING OBJ-SPCL.
```

15.4.1.2.2 *COB-BINDTABLEクラスのオブジェクトメソッド

オブジェクトメソッドには、以下のメソッドがあります。

- REGISTERメソッド

・ REGISTER-INDICATORメソッド

・ REGISTER-VARYINGメソッド

・ REGISTER-VARYING-INDICATORメソッド

・ REGISTER-LARGE-DATAメソッド

・ REGISTER-LARGE-DATA-INDICATORメソッド

以下、それぞれのメソッドについて説明します。

REGISTERメソッド

REGISTERメソッドは、データ項目の登録を行います。

REGISTERメソッドの機能および引数を以下に示します。

メソッド名	引数	引数の型	意味
REGISTER	【USING 第1】 登録データ項目	任意 (基本項目)	USING の第1引数で指定されたデータ項目を登録します。
	【RETURNING】 なし	―	

使用規則

1. このメソッドを呼び出す順序で第1引数に指定されたデータ項目が登録されるため、呼び出す順序を考慮する必要があります。
2. 第1引数は基本項目である必要があります。
3. 第1引数に指定するデータ項目の型については、各種クラスライブラリの取り扱い可能なデータ型である必要があります。

使用例

```
01 OBJ-SPCL USAGE OBJECT REFERENCE  BINTBL.
01 GET-DATA1 PIC X(15).
01 GET-DATA2 PIC X(20).
01 ELM-NO   PIC S9(4) COMP-5 VALUE 2.

として

    INVOKE  BINTBL "NEW"            USING ELM-NO
                                    RETURNING OBJ-SPCL.
    INVOKE  OBJ-SPCL     "REGISTER" USING GET-DATA1.
    INVOKE  OBJ-SPCL     "REGISTER" USING GET-DATA2.
```

REGISTER-INDICATORメソッド

REGISTER-INDICATORメソッドは、データ項目の登録を行います。また、登録するデータ項目に対応する標識変数の指定を行います。

REGISTER-INDICATORメソッドの機能および引数を以下に示します。

メソッド名	引数	引数の型	意味
REGISTER-INDICATOR	【USING 第1】 登録データ項目	任意 (基本項目)	USING の第1引数で指定されたデータ項目を登録します。 登録するデータ項目に対応する標識変数の指定を行います。
	【USING 第2】 標識変数	PIC S9(4) BINARY | COMP-5	
	【RETURNING】 なし	―	

使用規則

1. 第1引数は、REGISTERメソッドの使用規則に従う必要があります。
2. 第2引数は、符号付き2進2バイトのデータ項目である必要があります。
3. 標識変数の使用方法および設定する値、設定される値については、各クラスライブラリの使用規則に従います。

使用例

```
01 OBJ-SPCL USAGE OBJECT REFERENCE  BINTBL.
01 GET-DATA1 PIC X(15).
01 GET-DATA2 PIC X(20).
01 IND1      PIC S9(4) COMP-5.
01 ELM-NO    PIC S9(4) COMP-5 VALUE 2.

として

    INVOKE  BINTBL "NEW"             USING ELM-NO
                                     RETURNING OBJ-SPCL.
    INVOKE  OBJ-SPCL     "REGISTER"  USING GET-DATA1.
    INVOKE  OBJ-SPCL     "REGISTER-INDICATOR"
                                     USING GET-DATA2
                                     IND1.
```

REGISTER-VARYINGメソッド

REGISTER-VARYINGメソッドは、可変長データに対応するCOBOLのデータ項目の登録を行います。

REGISTER-VARYINGメソッドの機能および引数を以下に示します。

メソッド名	引数	引数の型	意味
REGISTER-VARYING	【USING 第1】 可変長データ長さ部	PIC S9(4 | 9) BINARY | COMP-5	USINGの第1引数で指定されたデータ項目を長さ部、第2引数に指定されたデータ項目をデータ部とするデータ項目を登録します。
	【USING 第2】 可変長データデータ部	PIC X(p) | PIC N(q)	
	【RETURNING】 なし	−	

使用規則

1. 基本的な規則は、REGISTERメソッドの規則に従う必要があります。
2. 第1引数と第2引数で指定するデータ項目は、以下の形式の同じ集団項目に従属する基本項目1、基本項目2である必要があります。

```
nn  集団項目1.
  mm  基本項目1  PIC S9(4 | 9) BINARY | COMP-5.
  mm  基本項目2  PIC X(p) | N (q).
```

   - 基本項目1は、符号付き2進である必要があります。
   - 基本項目1および基本項目2には、上記に記述されている句を指定する必要があります。

3. 登録するデータ項目は1つであるとみなします。

使用例

```
01 OBJ-SPCL USAGE OBJECT REFERENCE  BINTBL.
01 GET-DATA1.
  02 GET-DATA1-LNG  PIC S9(4) BINARY.
  02 GET-DATA1-DAT  PIC X(255).
01 GET-DATA2 PIC X(20).
01 ELM-NO    PIC S9(4) COMP-5 VALUE 2.
```

```
として

    INVOKE  BINTBL "NEW"              USING ELM-NO
                                      RETURNING OBJ-SPCL.
    INVOKE  OBJ-SPCL "REGISTER-VARYING" USING GET-DATA1-LNG
                                            GET-DATA1-DAT.
    INVOKE  OBJ-SPCL "REGISTER"       USING GET-DATA2.
```

REGISTER-VARYING-INDICATORメソッド

REGISTER-VARYING-INDICATORメソッドは、可変長データに対応するCOBOLのデータ項目の登録を行います。また、登録するデータ項目に対応する標識変数の指定を行います。

REGISTER-VARYING-INDICATORメソッドの機能および引数を以下に示します。

メソッド名	引数	引数の型	意味
REGISTER-VARYING-INDICATOR	【USING 第1】 可変長データ長さ部	PIC S9(4 | 9) BINARY | COMP-5	USINGの第1引数で指定されたデータ項目を長さ部、第2引数に指定されたデータ項目をデータ部とするデータ項目を登録します。 登録するデータ項目に対応する標識変数の指定を行います。
	【USING 第2】 可変長データデータ部	PIC X(n) | PIC N(m)	
	【USING 第3】 標識変数	PIC S9(4) BINARY | COMP-5	
	【RETURNING】 なし	－	

使用規則

1. 第1引数および第2引数は、REGISTER-VARYINGメソッドの使用規則に従う必要があります。
2. 第3引数は、符号付き2進2バイトのデータ項目である必要があります。
3. 標識変数の使用方法および設定する値、設定される値については、各クラスライブラリの使用規則に従います。

使用例

```
01 OBJ-SPCL USAGE OBJECT REFERENCE  BINTBL.
01 GET-DATA1.
  02 GET-DATA1-LNG  PIC S9(4) BINARY.
  02 GET-DATA1-DAT  PIC X(255).
01 GET-DATA2 PIC X(20).
01 IND1      PIC S9(4) COMP-5.
01 ELM-NO    PIC S9(4) COMP-5 VALUE 2.

として

    INVOKE  BINTBL "NEW"              USING ELM-NO
                                      RETURNING OBJ-SPCL.
    INVOKE  OBJ-SPCL "REGISTER-VARYING-INDICATOR"
                                      USING GET-DATA1-LNG
                                            GET-DATA1-DAT
                                            IND1.
    INVOKE  OBJ-SPCL "REGISTER"       USING GET-DATA2.
```

REGISTER-LARGE-DATAメソッド

REGISTER-LARGE-DATAメソッドは、バイナリデータに対応するCOBOLのデータ項目の登録を行います。

REGISTER-LARGE-DATAメソッドの機能および引数を以下に示します。

メソッド名	引数	引数の型	意味
REGISTER-LARGE-DATA	【USING 第1】 バイナリデータリザーブ部	PIC S9(9)　BINARY ｜COMP-5	USINGの第1引数で指定されたデータ項目をリザーブ部、第2引数で指定されたデータ項目を長さ部、第3引数で指定されたデータ項目をデータ部とするデータ項目を登録します。
	【USING 第2】 バイナリデータ長さ部	PIC　9(9)　BINARY ｜COMP-5	
	【USING 第3】 バイナリデータデータ部	PIC X(p) ｜PIC N(q)	
	【RETURNING】 なし	－	

使用規則

1. 基本的な規則は、REGISTERメソッドの規則に従う必要があります。
2. 第1引数、第2引数および第3引数で指定するデータ項目は、以下の形式の同じ集団項目に従属する基本項目1、基本項目2および基本項目3である必要があります。

```
nn  集団項目1.
  mm  基本項目1  PIC S9(9) BINARY | COMP-5.
  mm  基本項目2  PIC  9(9) BINARY | COMP-5.
  mm  基本項目3  PIC  X(p) | N(q).
```

   - 基本項目1は、符号付き2進4バイトのデータ項目である必要があります。
   - 基本項目2は、符号なし2進4バイトのデータ項目である必要があります。
   - 基本項目3は、英数字項目または日本語項目である必要があります。
   - 基本項目1、基本項目2および基本項目3には、上記に記述されている句を指定する必要があります。

3. 登録するデータ項目は1つであるとみなします。

使用例

```
01 OBJ-SPCL USAGE OBJECT REFERENCE  BINTBL.
01 GET-DATA1.
  02 GET-DATA1-RSV  PIC S9(9) BINARY.
  02 GET-DATA1-LNG  PIC  9(9) BINARY.
  02 GET-DATA1-DAT  PIC X(32767).
01 ELM-NO   PIC S9(4) COMP-5 VALUE 1.

として

    INVOKE  BINTBL "NEW"                   USING ELM-NO
                                           RETURNING OBJ-SPCL.
    INVOKE  OBJ-SPCL "REGISTER-LARGE-DATA" USING GET-DATA1-RSV
                                                 GET-DATA1-LNG
                                                 GET-DATA1-DAT.
```

REGISTER-LARGE-DATA-INDICATORメソッド

REGISTER-LARGE-DATA-INDICATORメソッドは、バイナリデータに対応するCOBOLのデータ項目の登録を行います。また、登録するデータ項目に対応する標識変数の指定を行います。

REGISTER-LARGE-DATA-INDICATORメソッドの機能および引数を以下に示します。

メソッド名	引数	引数の型	意味
REGISTER-LARGE-DATA-INDICATOR	【USING 第1】 バイナリデータリザーブ部	PIC S9(9)　BINARY ｜COMP-5	USINGの第1引数で指定されたデータ項目をリザーブ部、第2引数で指定されたデータ項目を長さ部、第3引数で指定されたデータ項目をデータ部とするデータ項目を登録します。
	【USING 第2】 バイナリデータ長さ部	PIC　9(9)　BINARY ｜COMP-5	

メソッド名	引数	引数の型	意味
	【USING 第3】 バイナリデータデータ部	PIC X(p) | PIC N(q)	登録するデータ項目に対応する標識変数の指定を行います。
	【USING 第4】 標識変数	PIC S9(4) BINARY | COMP-5	
	【RETURNING】 なし	―	

使用規則

1. 第1引数、第2引数および第3引数はREGISTER-LARGE-DATAメソッドの使用規則に従う必要があります。
2. 第4引数は、符号付き2進2バイトのデータ項目である必要があります。
3. 標識変数の使用方法および設定する値、設定される値については、各クラスライブラリの使用規則に従います。

使用例

```
01 OBJ-SPCL USAGE OBJECT REFERENCE  BINTBL.
01 GET-DATA1.
  02 GET-DATA1-RSV  PIC S9(9) BINARY.
  02 GET-DATA1-LNG  PIC  9(9) BINARY.
  02 GET-DATA1-DAT  PIC X(32767).
01 IND1      PIC S9(4) COMP-5.
01 ELM-NO    PIC S9(4) COMP-5 VALUE 1.

として

    INVOKE  BINTBL "NEW"                  USING ELM-NO
                                          RETURNING OBJ-SPCL.
    INVOKE  OBJ-SPCL "REGISTER-LARGE-DATA" USING GET-DATA1-RSV
                                                GET-DATA1-LNG
                                                GET-DATA1-DAT
                                                IND1.
```

15.5 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。

15.5.1 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、長さの異なる文字列を扱いたいことがあります。そのような場合、COBOLの文字列として、最大長不定のまま宣言する方法がないため、扱う文字列の最大長を決定する必要があります。また、呼出し側は実際は違う長さの文字列を渡したい場合でも、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、呼び出す必要があります。これは、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従うためです。

たとえば文字列の長さを最初に決めた最大長から変更したい場合、以下の作業が必要です。

1. 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長さに修正します。
2. 再翻訳します。

仮に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しても、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となります。

たとえば、下記の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。ここでは、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.   INFORMATION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
 REPOSITORY.
     CLASS  CONFIRM-CLASS.
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  CONFIRM-CLASS.
PROCEDURE DIVISION.
    INVOKE  CONFIRM-CLASS "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下さい".
    ACCEPT  名前文字列.
    ACCEPT  パスワード.
    INVOKE  認証オブジェクト "CONFIRM-METHOD" 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 INFORMATION.
```

・ 渡された文字列を扱う汎用クラス

```
CLASS-ID.   CONFIRM-CLASS INHERITS FJBASE.
 :
OBJECT.
PROCEDURE DIVISION.
METHOD-ID.  CONFIRM-METHOD.
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  CONFIRM-METHOD.
END OBJECT.
END CLASS   CONFIRM-CLASS.
```

この状態で、CONFIRM-CLASSを利用する処理がほかにも発生し、パスワードの最大長を変更したい場合、まず、CONFIRM-CLASS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、CONFIRM-CLASSを継承するクラスの再翻訳が必要となります。

15.5.2 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。したがって下記の記述により、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。

図15.12 ANY LENGTH句を用いた例

```
CLASS-ID.   CONFIRM-CLASS INHERITS FJBASE.
  :
  OBJECT.
  PROCEDURE DIVISION.
  METHOD-ID.  CONFIRM-METHOD.
  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.┐  … CONFIRM-METHODに制御が渡された
  01  パスワード   PIC X  ANY LENGTH.│     ときに、呼出し側の対応する項目
  01  認証結果     PIC X  ANY LENGTH.┘     の長さに決まる。
  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  CONFIRM-METHOD.
  END OBJECT.
  END CLASS   CONFIRM-CLASS.
```

汎用的なクラスを作成する場合、そのクラスが抽象的であればあるほどインタフェースの変更の影響は大きくなります。そのため、最大長を意識しなくても汎用クラスの作成ができることは重要なテクニックといえます。

第16章　オブジェクト指向プログラムの開発と実行

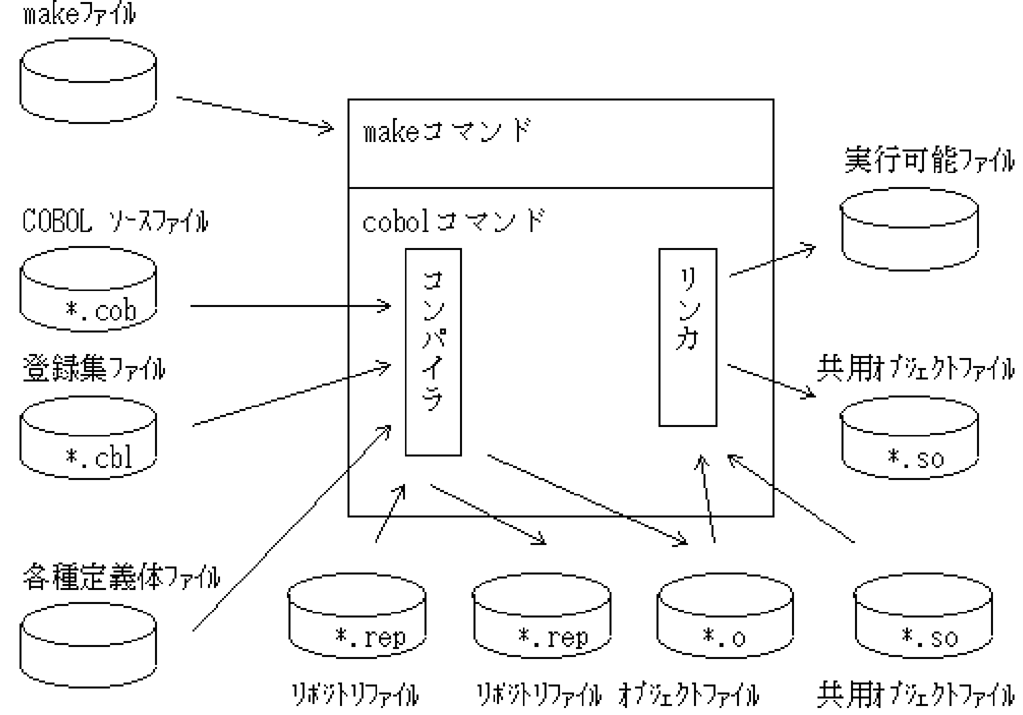
本章では、オブジェクト指向プログラムの開発方法および実行について説明します。

16.1 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とその関係について説明します。

オブジェクト指向プログラミングでは、従来の開発資源にリポジトリファイルが加わります。

オブジェクト指向プログラミングの資源の関係を以下に示します。



“表16.1 オブジェクト指向プログラミングで使用するファイル”に、オブジェクト指向プログラミングで使用するファイルを示します。

表16.1 オブジェクト指向プログラミングで使用するファイル

ファイル識別	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
リポジトリファイル	継承および適合チェックのためのクラス関連情報	クラス名(外部名).rep	入力	リポジトリ段落を指定したソースファイルを翻訳する場合に使用します。
			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 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、makeコマンドを使用することをおすすめします。makeコマンドについては、“付録M makeコマンドの活用”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. 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。

16.3 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。

16.4 使用するクラスの選定

"13.3 オブジェクト指向のメリット"で説明したように、オブジェクト指向プログラミングには、以下のメリットがあります。

- 部品化が容易にできます。
- 既存の部品の流用が容易にできます。

しかし、どんなに部品化が容易であっても、ほかの人が使用するためには、その部品を使用するための情報を伝える必要があります。また、既存の部品の流用が容易であっても、使用するためには、その部品を使用するための情報を入手する必要があります。

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く必要のある情報として、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、ある目的を実現するために行われます。

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ある必要があります。

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おく必要があります。

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場合およびそのオブジェクトを動作させる場合も、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には、目的に合った処理を行うメソッドがある場合、利用者はクラスを利用することができます。

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ます。そして利用者はそのメソッドの名前がわかれば、使用することができます。

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、思ったとおりの結果を得られないことがあります。なぜなら、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るのかがわからないからです。

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以下のインタフェース情報がわかれば、そのメソッドを使用することができます。

- 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)
- どのような情報が返ってくるのか

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。

16.5 プログラム構造

ここでは、プログラム構造について説明します。

16.5.1 翻訳単位とリンク単位

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共用オブジェクトファイル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共用オブジェクトファイルは複数のオブジェクトファイルで構成することも可能です。そのため翻訳単位とリンク単位は一致しない場合もあります。

翻訳単位とリンク単位の関係を下図に示します。

図16.1 翻訳単位とリンク単位が一致する場合

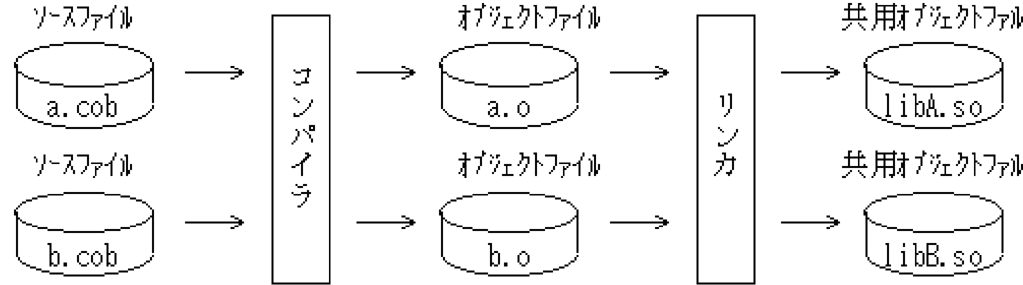
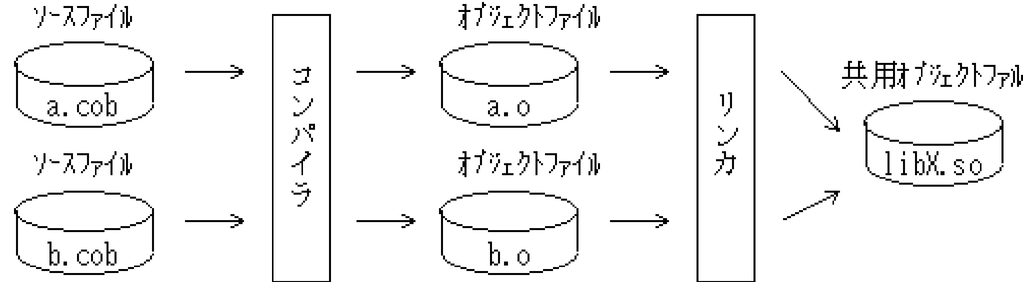


図16.2 翻訳単位とリンク単位が一致しない場合



16.5.2 プログラム構造の概要

“3.2.2 結合の種類とプログラム構造”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。

- 16.5.2.1 静的構造
- 16.5.2.2 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。

16.5.2.1 静的構造

静的構造は、複数のオブジェクトファイルで1つの実行可能ファイルまたは共用オブジェクトファイルを構成します。

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共用オブジェクトファイル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、以下の場合に使用します。

- PROTOTYPE宣言したメソッドを含むクラス定義と、それによって分離されたメソッド定義

16.5.2.2 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。

動的リンク構造

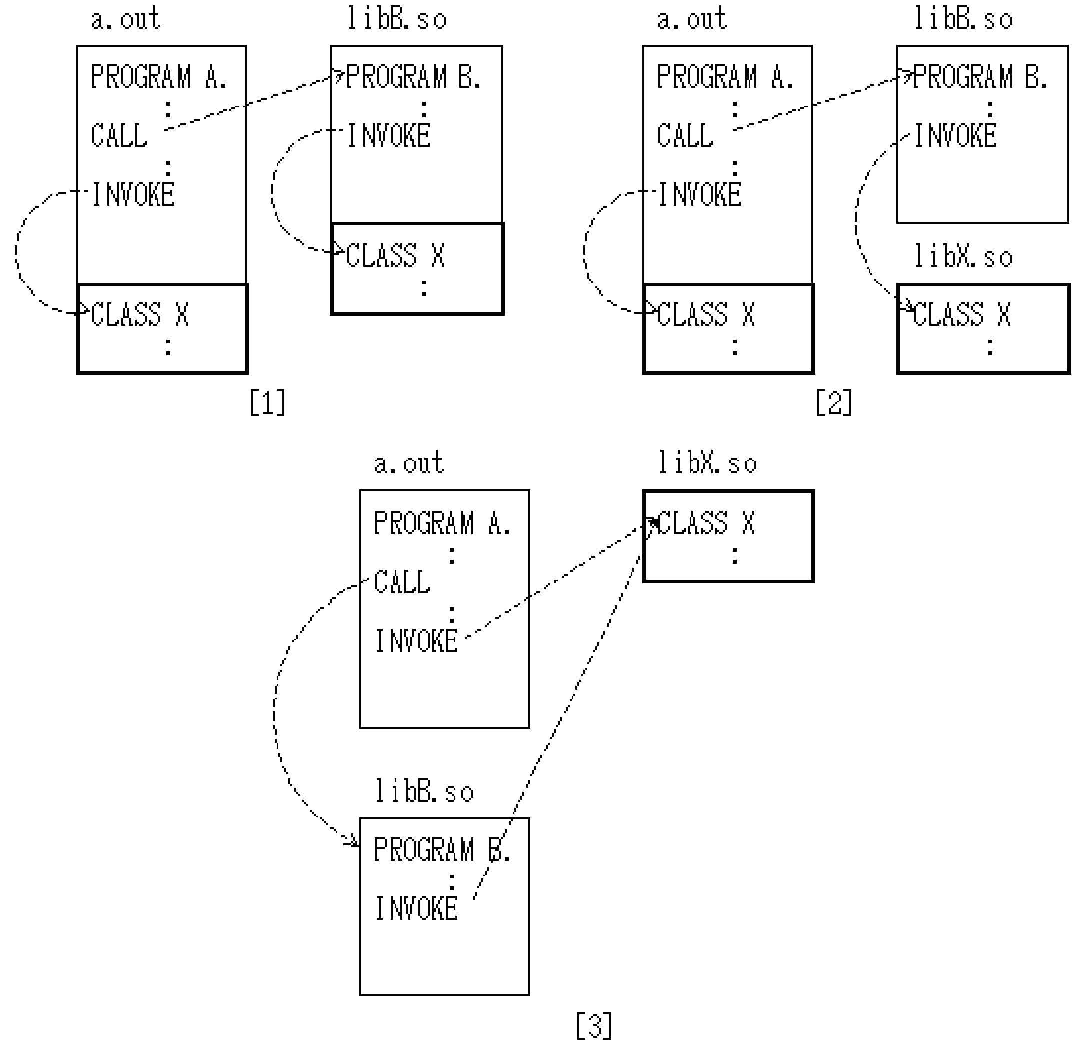
動的リンク構造では、ある機能単位ごとに作成された複数の共用オブジェクトファイルを実行可能ファイルの起動時にリンクします。

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、対象の共用オブジェクトファイルだけをリンクするだけです(ほかの実行可能ファイルおよび共用オブジェクトファイル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

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。



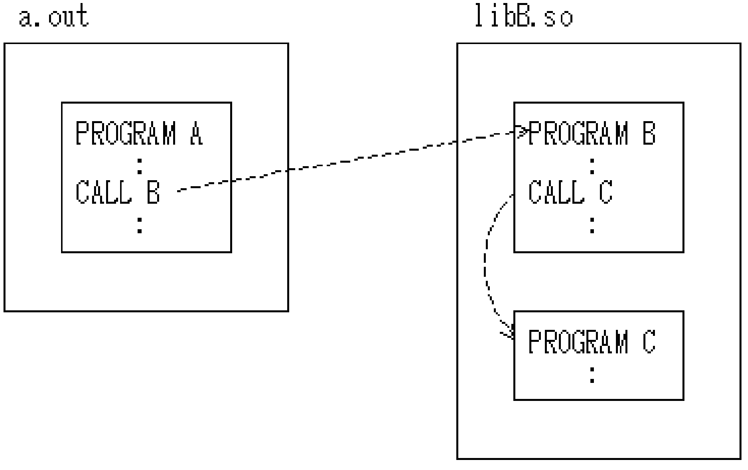
[1] 静的構造により、同一クラスのオブジェクトが2つ存在する場合
[2] 静的構造と動的構造により、同一クラスのオブジェクトが2つ存在する場合
[3] クラスのオブジェクトファイルを動的構造に修正

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、同一の共用オブジェクトファイルに含まれていれば、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。しかし、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。



動的プログラム構造

動的プログラム構造では、クラスおよびメソッドの仮想メモリ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の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や動的リンク構造と比べると、実行可能ファイルの起動は速くなります。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び出しは、COBOLの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。

注意

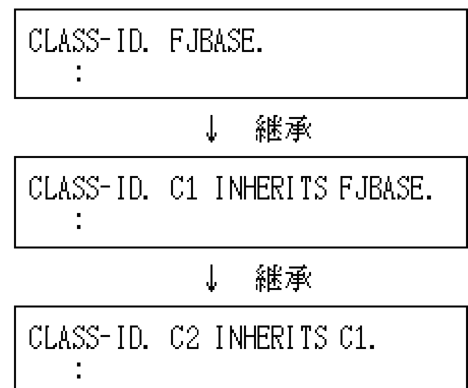
- 動的プログラム構造のクラスを実行する場合は、クラスのエントリ情報が必要となります。ただし、クラスの共用オブジェクトファイル名を"libクラス名.so"にすることにより、エントリ情報ファイルは不要となります。このため、動的プログラム構造で呼び出すクラスの共用オブジェクトファイルは、1つのクラスを1つの共用オブジェクトファイルとし、ファイル名は、"libクラス名.so"にすることをおすすめします。
- オブジェクト指向プログラムではない従来のプログラムでは、CANCEL文によって動的プログラム構造でローディングしたプログラムを仮想メモリ上から削除することが可能でした。しかし、オブジェクト指向プログラムでは、CANCEL文に相当する機能が存在しません。そのため、このメリットはありません。
- プログラムの動的プログラム構造と併用する場合は、"9.1.2.3 注意事項"を必ずお読みください。

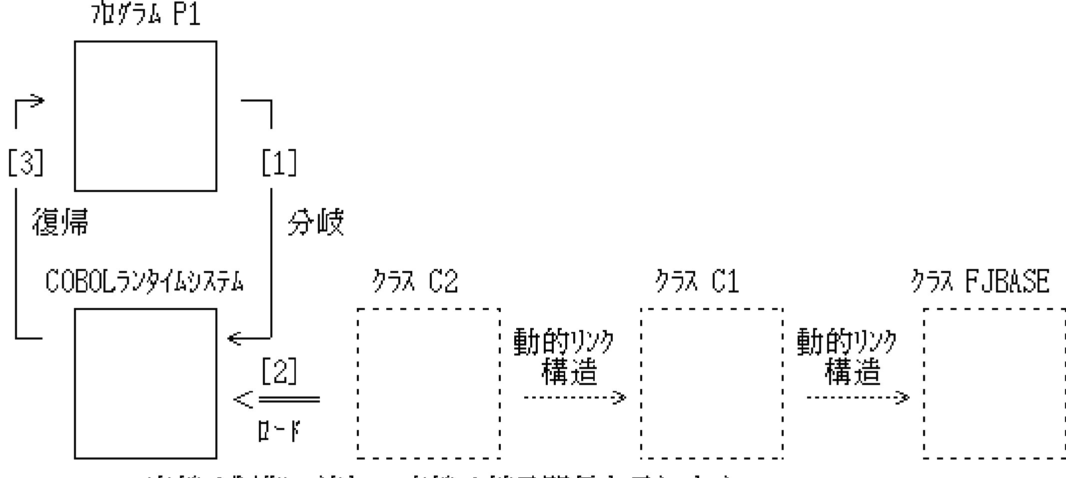
クラスの動的プログラム構造

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メモリ上へのローディングは、クラスが参照されたときにCOBOLのランタイムシステムによって行われます。このとき、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クラスC2が動的プログラム構造の場合、クラスC2は、[a]のINVOKE文が実行されたときに仮想メモリ上にロードされます。クラスC2が直接継承しているクラスC1および間接的に継承しているクラスFJBASEも、このときロードされます。

```
PROGRAM-ID. P1.
    :
PROCEDURE DIVISION.
     :
    INVOKE C2 "M1".   … [a]
     :
    EXIT PROGRAM.
END PROGRAM P1.
```





実線は制御の流れ、点線は継承関係を示します。

[1] INVOKE文の実行により、クラスC2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、クラスC2をロードします。このとき、クラスC2が継承しているクラスC1とFJBASE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

- システムの制限により、動的プログラム構造で、日本語文字からなるクラス名やメソッド名を呼び出すことはできません。
- クラスを動的プログラム構造とした場合、PROTOTYPE宣言によってメソッドの呼出しも動的プログラム構造となります。

メソッドの動的プログラム構造

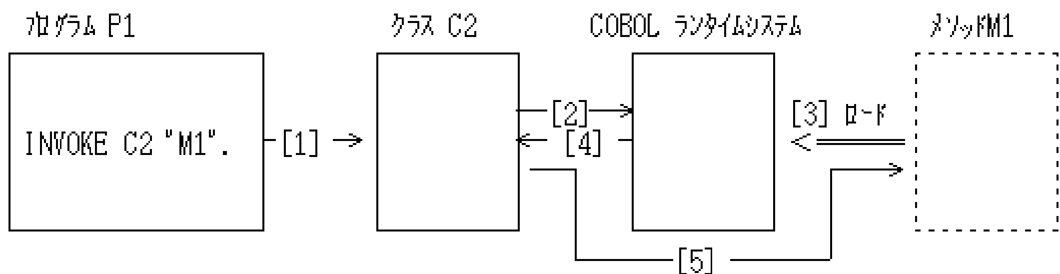
PROTOTYPE宣言により分離されたメソッドを動的プログラム構造にすることができます。そうすることにより、アプリケーションで使用しているメソッドの仮想メモリ上へのローディングは、メソッドが呼び出されたときに、COBOLのランタイムシステムによって行われるようになります。

以下のようなプログラムでメソッドM1が動的プログラム構造の場合、メソッドM1は、[a]のINVOKE文が実行されたときに仮想メモリ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] メソッドM1の呼出しにより、クラスC2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メソッドM1をロードします。

[4] クラスC2に復帰します。

[5] メソッドM1が呼び出されます。

16.6 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の2つについて説明します。

- 16.6.1 リポジトリファイルと翻訳の手順
- 16.6.2 動的プログラム構造での翻訳処理

なお、一般的な翻訳処理については、“第3章 プログラムの翻訳とリンク”を参照してください。

16.6.1 リポジトリファイルと翻訳の手順

ここでは、まず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、“クラス名(外部名).rep”になります。

図16.3 リポジトリファイルの出力

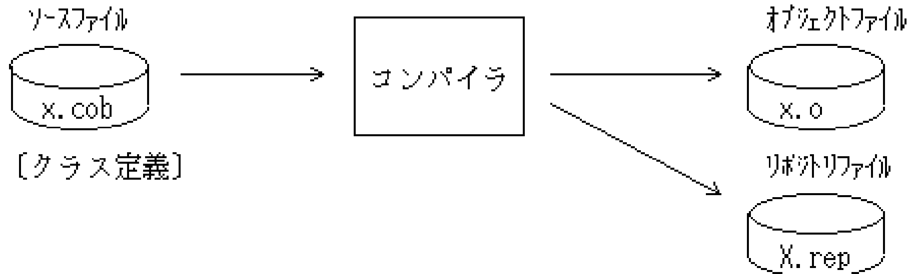
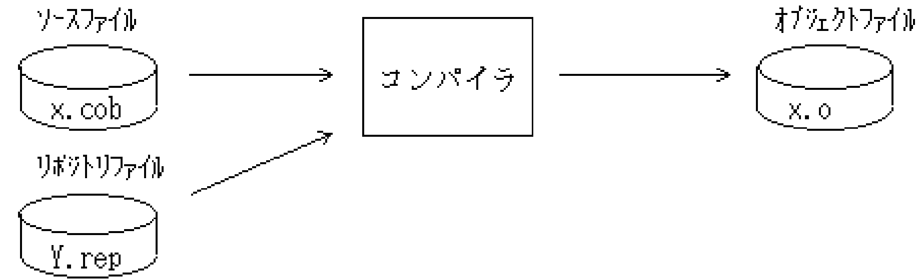


図16.4 リポジトリファイルの入力



翻訳時にコンパイラが入力するリポジトリファイルは、リポジトリ段落にクラス名が記述されたものだけです。

リポジトリ段落に記述される必要のある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

“例題プログラム”の例題13の“member.cob”を例にとります。

```
    :
CLASS-ID. Member-class INHERITS AllMember-class.              [a]
ENVIRONMENT DIVISION.
 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る必要があります。(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”を例にとります。

allmem.cob

```
    :
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する必要があります (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"を例にとります。

sala_mem.cob

```
METHOD-ID. Salary-method  OF  Member-class.                    [a]
 ENVIRONMENT DIVISION.
  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、再度翻訳する必要があります。

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。

"例題プログラム"の例題13の"member.cob"は、以下のように記述されています。

member.cob

```
     :
 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。

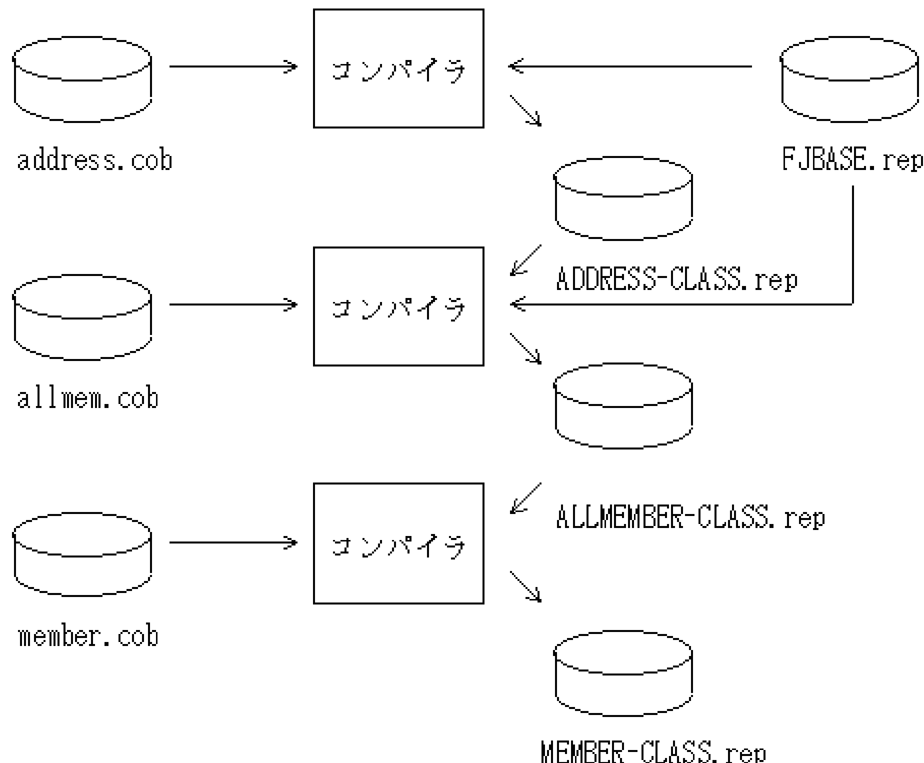
“address.cob”のソースプログラムは、以下のように記述されています。

```
    :
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16.5 翻訳順序



“図16.5 翻訳順序”からもわかるように、以下の順序で翻訳処理が行われる必要があります。

1. address.cob
2. allmem.cob
3. member.cob

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16.5 翻訳順序”の“member.cob”のターゲットリポジトリファイルは、“MEMBER-CLASS.rep”になります。

ターゲットリポジトリファイル生成ディレクトリ

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ディレクトリは、翻訳コマンドのオプション-drの指定によって、下表のように決まります。

[参照]“3.3.1.5 -dr (リポジトリファイルの入出力先ディレクトリの指定)”

-dr オプション	ディレクトリ名
あり	-dr オプションで指定したディレクトリ
なし	COBOL ソースファイルが存在するディレクトリ

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16.5 翻訳順序”の“member.cob”の依存リポジトリファイルは、“ALLMEMBER-CLASS.rep”になります。

注意

FJBASEクラスおよび特殊クラスに対しては、依存リポジトリファイルは必要ありません。FJBASEクラスについては、“14.3.2 FJBASEクラス”を参照してください。特殊クラスについては、“15.4 特殊クラス”を参照してください。

依存リポジトリファイル検索ディレクトリ

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ディレクトリは、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]“3.3.1.17 -R (リポジトリファイルの入力先ディレクトリの指定)”、“3.3.1.5 -dr (リポジトリファイルの入出力先ディレクトリの指定)”

-Rオプション	-drオプション	検索パスの順序
あり	あり	(1) -Rオプションで指定したディレクトリ (2) -drオプションで指定したディレクトリ (3) カレントディレクトリ
	なし	(1) -Rオプションで指定したディレクトリ (2) カレントディレクトリ
なし	あり	(1) -Rオプションで指定したディレクトリ (2) カレントディレクトリ
	なし	(1) カレントディレクトリ

cobolコマンドで翻訳を行う例

以下に、“例題プログラム”の例題13のallmem.cobをcobolコマンドを使って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、“3.3 cobolコマンド”を参照してください。

```
$ cobol -c -R /home/sample13 -dr /home/sample13 allmem.cob
最大重大度コードはIで,翻訳したプログラム数は1本です.
```

```
入力       : allmem.cob(ソースファイル)
             ADDRESS-CLASS.rep(リポジトリファイル:/home/sample13に格納)
             MEMBERMASTER-CLASS.rep(リポジトリファイル:/home/sample13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:/home/sample13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ディレクトリ)
             -dr(リポジトリファイルの入出力先ディレクトリ)
```

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"14.7.8.2 相互参照クラスの翻訳"を参照してください。

16.6.2 動的プログラム構造での翻訳処理

ここでは、翻訳時に必要となる翻訳オプションおよび注意事項について説明します。

クラスを動的プログラム構造にする場合

クラスを動的プログラム構造にするには、クラスを参照しているソースプログラムの翻訳時に、翻訳オプションDLOADを指定します。この指定により、このソースプログラムから参照されているクラスは、動的プログラム構造になります。

[参照]"A.2.10 DLOAD（プログラム構造の指定）"

ただし、ほかのソースプログラムでそのクラスが動的リンク構造になっていると、実行可能ファイルの起動時にそのクラスはシステムによってロードされてしまいます。このため、COBOLの実行単位でプログラム構造を統一してください。

メソッドを動的プログラム構造にする場合

メソッドを動的プログラム構造にするには、動的プログラム構造にしたいメソッドを原型定義し、そのメソッドの原型定義を含むクラスの翻訳時に、翻訳オプションDLOADを指定します。このように、クラスの翻訳の方法によって、分離されたメソッドのプログラム構造が決まります。

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

[参照]"A.2.10 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クラスC1とクラスC2があり、クラスC1が翻訳オプションNODLOADで翻訳され、クラスC2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、クラスC2で定義されているメソッドのうち、原型定義されているメソッドM5とメソッドM7が動的プログラム構造となります。メソッドM6とメソッドM8は原型定義されていないため、クラスC2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。

クラスC1で定義されているメソッドは、クラスC1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-ID. C1 INHERITS FJBASE.
    :
 FACTORY.
    :
 PROCEDURE DIVISION.
  METHOD-ID. M1 PROTOTYPE.
    :
  METHOD-ID. M2.
    :
END FACTORY.
OBJECT.
   :
PROCEDURE DIVISION.
 METHOD-ID. M3 PROTOTYPE.
   :
 METHOD-ID. M4.
   :
 END OBJECT.
END CLASS C1.
```

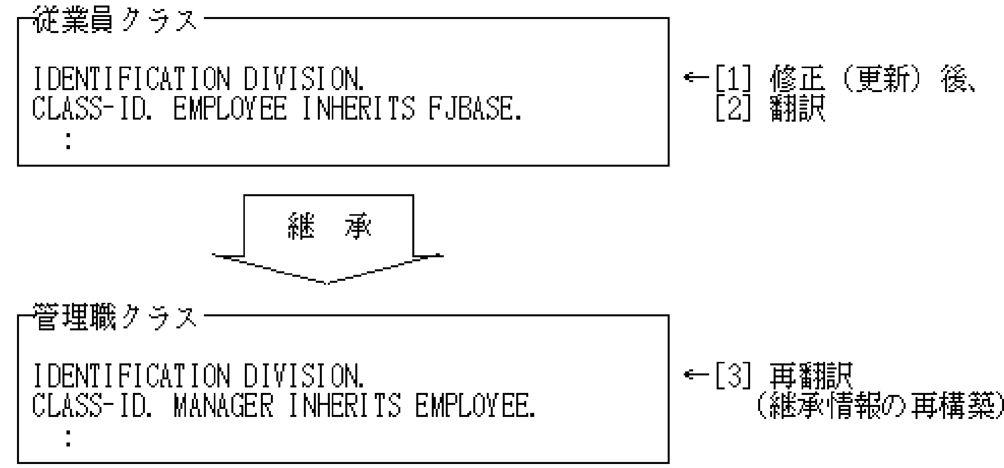
翻訳オプションDLOAD 指定

```
CLASS-ID. C2 INHERITS C1.
    :
 FACTORY.
    :
 PROCEDURE DIVISION.
  METHOD-ID. M5 PROTOTYPE.
    :
  METHOD-ID. M6.
    :
END FACTORY.
OBJECT.
   :
PROCEDURE DIVISION.
 METHOD-ID. M7 PROTOTYPE.
   :
 METHOD-ID. M8.
   :
 END OBJECT.
END CLASS C2.
```

リポジトリファイル更新の影響

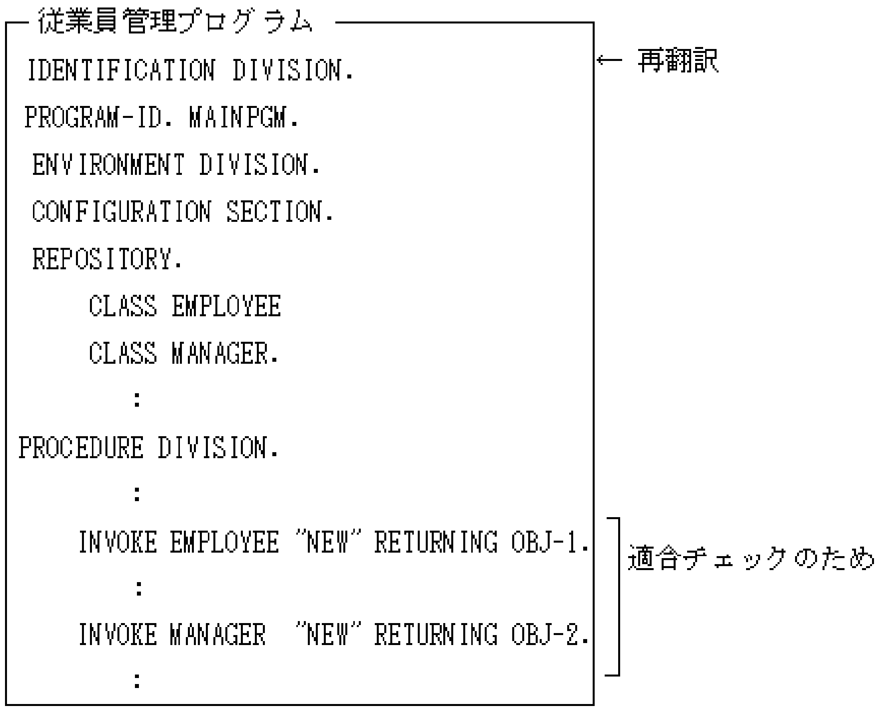
目的プログラムの生成後、親クラスや呼び出すクラスに修正が入った場合、以下の点に注意してください。

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う必要があります。また、インタフェースに変更がある修正の場合も再翻訳する必要があります。



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継承情報の再構築のために再翻訳が必要になります。

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適合チェックのために再翻訳が必要です。



これらの再翻訳は、利用者が行う必要があります。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。

16.7 リンク処理

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リンク処理で、特に意識する必要がある以下の2つについて説明します。

- 16.7.1 リンク関係とリンクの手順
- 16.7.2 動的プログラム構造でのリンク処理

16.7.1 リンク関係とリンクの手順

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のリンク関係が発生するパターンは増えました。

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"表16.2 オブジェクト指向プログラミングでのリンク"に示します。

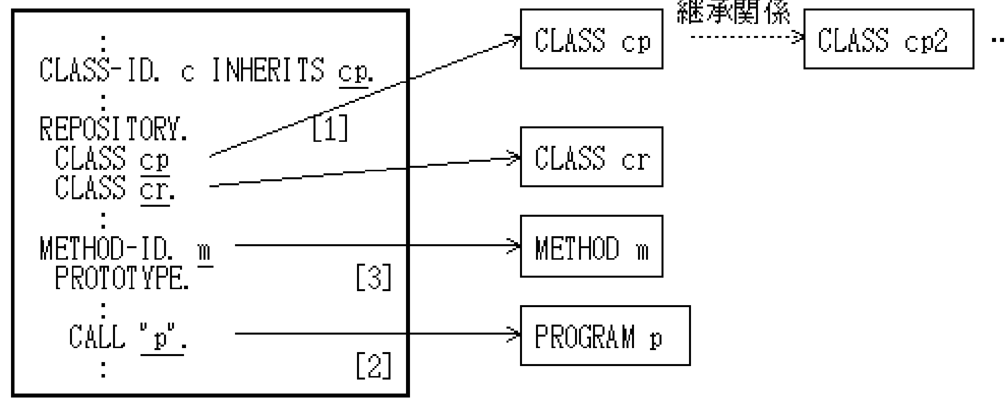
表16.2 オブジェクト指向プログラミングでのリンク

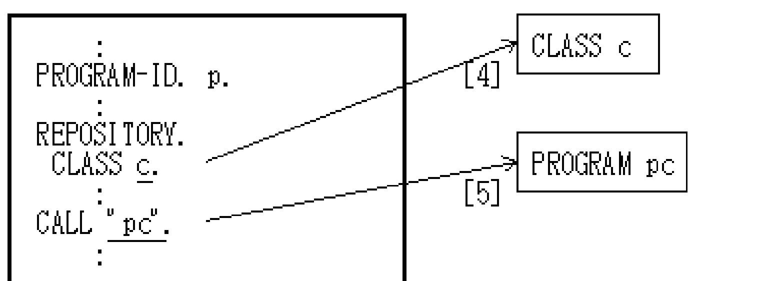
定義名		リンク先		
		クラス定義	プログラム定義	メソッド定義
リンク元	クラス定義	[1] リポジトリ段落に記述したクラス	[2] CALL文で呼び出すプログラム	[3] PROTOTYPE宣言で分離されたメソッド
	プログラム定義	[4] リポジトリ段落に記述したクラス	[5] CALL文で呼び出すプログラム	−
	メソッド定義	[6] リポジトリ段落に記述したクラス(ただし、自メソッドのPROTOTYPE宣言を含むクラスは除く) [7] 自メソッドのPROTOTYPE宣言を含むクラスの親クラス (注)	[8] CALL文で呼び出すプログラム	−

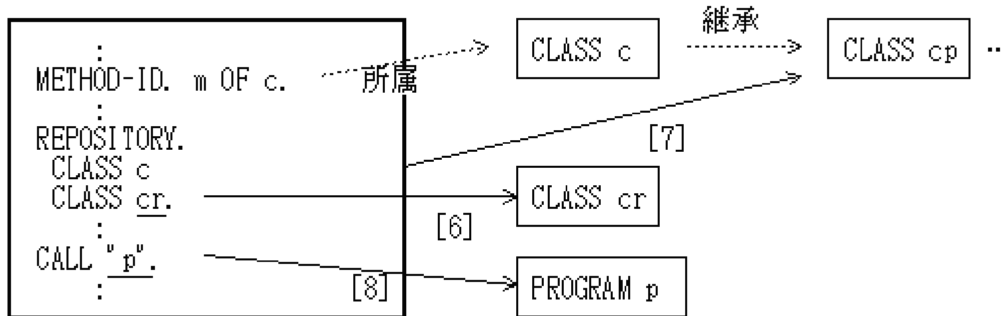
注: 手続き部中にSUPERが指定されている場合だけです。

各定義でのリンクを"図16.6 各定義でのリンク"に示します。

図16.6 各定義でのリンク







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共用オブジェクトファイルが必要になります。

注意

継承関係の親クラスである共用オブジェクトファイルについては動的リンク構造で解決する必要があります。

リンクの手順

リンカを使用して実行可能ファイルまたは共用オブジェクトファイルを作成するときに、リンクする共用オブジェクトファイルが必要になります。そのため、複数の実行可能ファイルまたは共用オブジェクトファイルを作成する場合、リンク処理の順序に制約が発生します。

“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

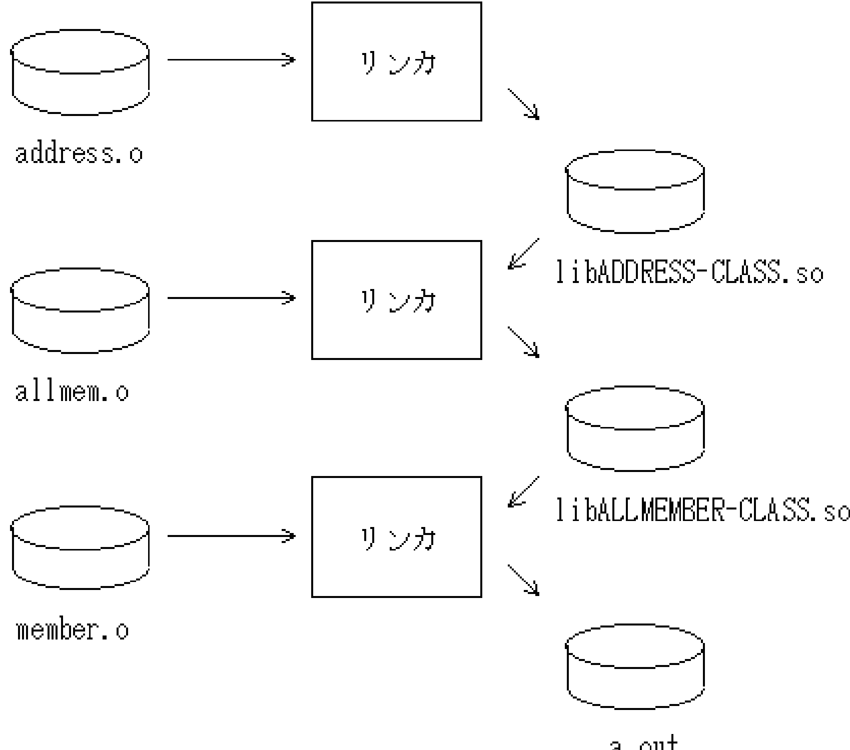
```
      :
CLASS-ID. Address-class …
      :
```

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共用オブジェクトファイルおよび生成される実行可能ファイルまたは共用オブジェクトファイル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	生成する実行可能ファイルまたは共用オブジェクトファイル	使用する共用オブジェクトファイル
member.cob / member.o	a.out	libALLMEMBER-CLASS.so
allmem.cob / allmem.o	libALLMEMBER-CLASS.so	libADDRESS-CLASS.so
address.cob / address.o	libADDRESS-CLASS.so	—

この関係を“図16.7 共用オブジェクトファイルとリンク処理の順序”に示します。

図16.7 共用オブジェクトファイルとリンク処理の順序



FJBASEクラスの共用オブジェクトファイルは、利用者が指定しなくても、自動的にリンクされます。

“図16.7 共用オブジェクトファイル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.o
2. allmem.o
3. member.o

cobolコマンドでリンクを行う例

以下に、cobolコマンドを使ってリンクした場合の例を示します。

```
$ cobol -G -o libADDRESS-CLASS.so address.o
$ cobol -G -o libALLMEMBER-CLASS.so -l ADDRESS-CLASS allmem.o
$ cobol -o a.out -l ALLMEMBER-CLASS member.o
```

```
入力      :  address.o(オブジェクトファイル)
             allmem.o(オブジェクトファイル)
             member.o(オブジェクトファイル)
出力      :  libADDRESS-CLASS.so(共用オブジェクトファイル)
             libALLMEMBER-CLASS.so(共用オブジェクトファイル)
             a.out(実行可能ファイル)
オプション:  -o(実行可能プログラムまたは共用オブジェクトファイルの出力先)
             -l(リンクするライブラリの指定)
```

参考

ldコマンドを使って実行可能プログラムを生成することもできます。

ldコマンドの使い方については、“付録L ldコマンド”を参照してください。

16.7.2 動的プログラム構造でのリンク処理

ここでは、リンク時に必要となる共用オブジェクトファイルについて説明します。

プログラムの実行可能ファイルまたは共用オブジェクトファイルを作成する場合

共用オブジェクトファイルは不要です。

クラスの共用オブジェクトファイルを作成する場合

共用オブジェクトファイルを作成するクラスが直接継承しているすべてのクラスの共用オブジェクトファイルだけが必要となります。

分離されたメソッドの共用オブジェクトファイルを作成する場合

共用オブジェクトファイルを作成する分離されたメソッドの手続き部で、定義済みオブジェクト一意名SUPERが利用されていることがあります。この場合だけ、そのメソッドが定義されているクラスが直接継承しているクラスの共用オブジェクトファイルが必要となります。

16.7.3 共用オブジェクトファイルの構成とファイル名

ここでは、クラスおよびメソッドの共用オブジェクトファイルについて、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ファイルを省略できます。

クラスの共用オブジェクトファイルの構成とファイル名

クラス単位で共用オブジェクトを作成し、ファイル名を“libクラス名.so”とします。

メソッドの共用オブジェクトファイルの構成とファイル名

メソッド単位で共用オブジェクトを作成し、ファイル名を“libクラス名_メソッド名.so”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。

16.7.4 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ファイル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。

なお、副プログラムのエントリ情報については、“4.1.4 副プログラムのエントリ情報”を参照してください。

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共用オブジェクトファイル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=共用オブジェクトファイル名         …[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、共用オブジェクトファイル名には利用するクラスが格納されている共用オブジェクトファイル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。共用オブジェクトファイル名の拡張子は“so”でなければなりません。

共用オブジェクトファイル名が“libクラス名.so”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共用オブジェクトファイル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=共用オブジェクトファイル名       …[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、共用オブジェクトファイル名には呼び出すメソッドが格納されている共用オブジェクトファイル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。共用オブジェクトファイル名の拡張子は“so”でなければなりません。

共用オブジェクトファイル名が“libクラス名_メソッド名.so”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メソッドの共用オブジェクトファイル名が“lib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メソッドの共用オブジェクトファイル名が“lib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GETまたはSETの間には、アンダースコアを2つ指定します。

注意

クラスおよびメソッドのソースプログラムが翻訳オプションALPHALで翻訳された場合、クラス名、メソッド名は大文字と小文字は等価に扱われます。[参照]“A.2.1 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。

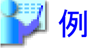
実行に必要となるエントリ情報

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ファイルが必要となります。

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中の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、“14.4 適合”、“A.2.3 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について説明します。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

```
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=libC1.so
C2=libC2.so
C3=libC3.so

[C1.METHOD]
M1=libC1_M1.so

[C2.METHOD]
M2=libC2_M2.so
```

libC1.so

```
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.
```

libC2.so

```
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.
```

libC1_M1.so

```
METHOD-ID. M1 OF C1.
END METHOD M1.
```

libC2_M2.so

```
METHOD-ID. M2 OF C2.
END METHOD M2.
```

libC3.so

```
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、クラスの共用オブジェクトファイル名は“libクラス名.so”、メソッドの共用オブジェクトファイル名は“libクラス名_メソッド名.so”となっているため、省略することができます。

16.8 クラスの公開

作成およびテストが完了したクラスは、新しい部品として、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ある必要があります。

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。

クラス公開で公開対象となる資源およびその用途は、“表16.4 公開資源”のようになります。

表16.4 公開資源

	資源名	用途
[1]	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となります。
[2]	共用オブジェクトファイル(またはオブジェクトファイル)	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となります(クラスを単純構造で作成する場合にはオブジェクトファイルが必要となります)。
[3]	リポジトリファイル	再利用するクラス/プログラムを翻訳する場合に必要となります。

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

16.9 実行時の注意事項

ここでは、オブジェクト指向プログラムの実行時の注意事項について説明します。

16.9.1 スタックオーバーフロー

オブジェクト指向プログラムでは、これまで以上にスタックを使用するようになります。

このため、スタックオーバーフローに対する十分な注意が必要です。詳細については“4.5.1 COBOLプログラムの実行時にスタックオーバーフローが発生する場合”を参照してください。

16.9.2 オブジェクトインスタンスのブロック化

ここでは、COBOLのオブジェクト指向プログラムでのオブジェクトインスタンスのブロック化について説明します。

16.9.2.1 概要

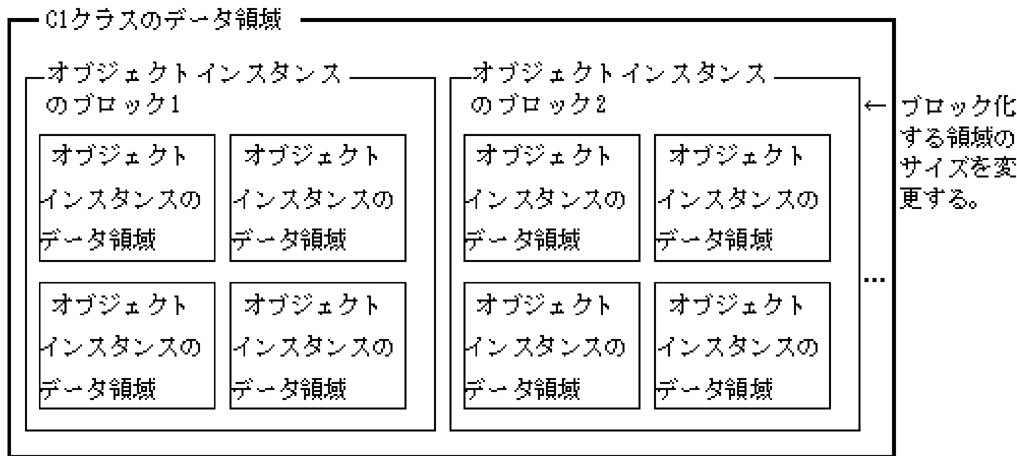
オブジェクト指向プログラムでは、NEWメソッドを実行することにより、オブジェクトインスタンスが生成されます。このとき、オブジェクトインスタンスのデータ部に記述したデータ項目およびオブジェクトインスタンスを動作させるために必要な作業域などが、オブジェクトインスタンスのデータ領域として獲得されます。

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。

オブジェクトインスタンスのブロック化とは、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。



16.9.2.2 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、特に指定がなければ、COBOLランタイムシステムが実行性能も考慮し、最適と判断した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することで必要な領域だけを獲得するため、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の指定の詳細については、“16.9.2.4 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
```

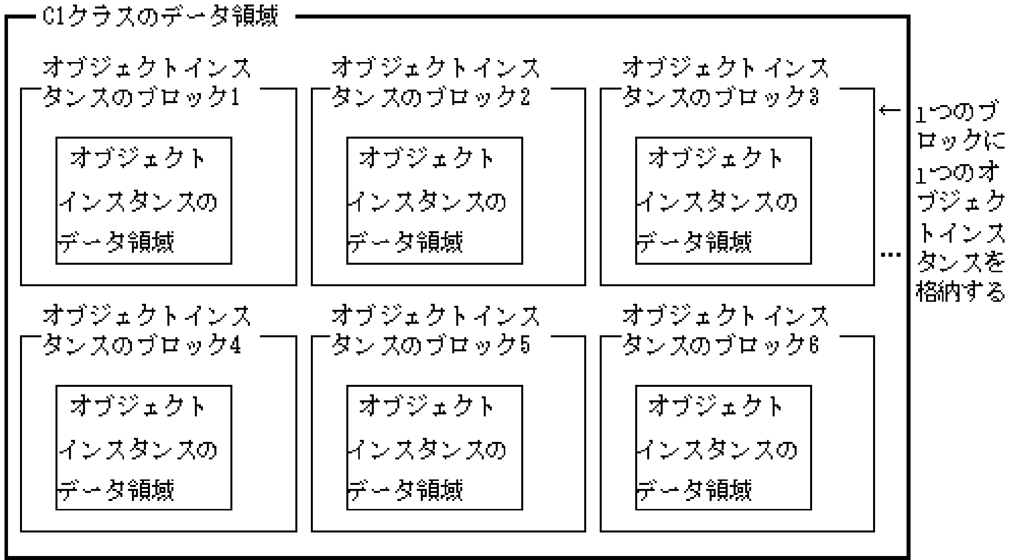
アプリケーションの実行単位に対する指定(環境変数)

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。



16.9.2.3 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、"16.9.2.4 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。

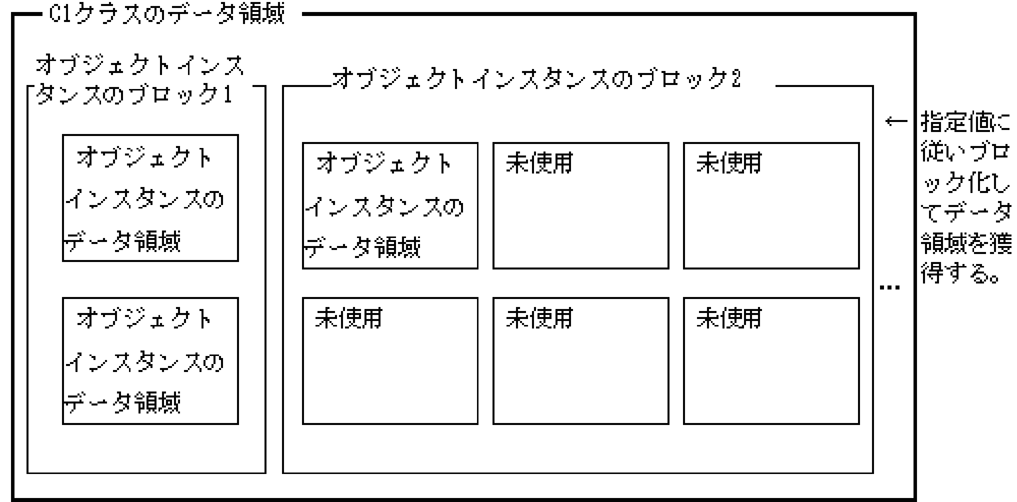
クラスごとの指定(クラス情報)

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。

16.9.2.4 メモリのチューニングに関する実行環境情報

メモリのチューニングに関する実行環境情報について説明します。

16.9.2.4.1 環境変数

ファイルに関するもの

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、“16.9.2.4.2 クラス情報”を参照してください。

クラス情報ファイルには、絶対パスと相対パスを指定できます。相対パスが指定された場合は、実行している実行可能ファイルが存在するディレクトリからの相対パスとなります。

オブジェクトインスタンスに関するもの

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。

16.9.2.4.2 クラス情報

クラス情報は、オブジェクト指向プログラムで使用するクラスに対する情報をセクションごとに指定します。指定する情報を以下に示します。

クラス情報

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、“16.9.2.4.1 環境変数”を参照してください。

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CBR_INSTANCEBLOCKの指定に従います。

参照

"16.9.2.4.1 環境変数"

第17章　マルチスレッド

本章では、マルチスレッド環境下で動作可能なCOBOLプログラムについて説明します。

17.1 概要

Webや分散オブジェクトなどの機能を利用して、COBOLアプリケーションをサーバアプリケーションとして使用する場合、多くのクライアントからの実行要求により、サーバの負荷は増加します。このような運用形態に対して、マルチスレッドを適用することにより、サーバの負荷を軽減し、多くのクライアントからの要求にも耐えられるようになります。また、システムの状態によっては実行性能も向上させることができます。

17.1.1 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。

マルチスレッド環境下でプログラムを呼び出すようなサーバ製品にCOBOL資産を移行する場合や流用する場合などでも問題ありません。マルチスレッド環境下でプログラムを呼び出すようなサーバ製品には以下のようなものがあります。

分散オブジェクト技術(CORBAなど)を利用したアプリケーションサーバ製品

クライアントのCOBOLアプリケーションから、クラス定義を呼び出すことにより、アプリケーションサーバを介在して、サーバ上にある複数の環境に配置されたクラス定義が実行できます。マルチスレッド環境下では、クラス定義の配置をスレッドごとにすることが可能となります。

Webサーバ製品

Web連携では、クライアント上のブラウザからWebサーバに要求を送信し、サーバに登録されているCOBOLアプリケーションが実行されます。マルチスレッド環境下では、複数の要求を別々のスレッドに割り当てることが可能となります。

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ます。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

COBOLの提供するTRACE機能、CHECK機能、COUNT機能および対話型デバッガにより、マルチスレッド環境下で動作するプログラムをデバッグすることができます。また、デバッグ補助のために、プロセスID/スレッドIDの取得サブルーチンを提供しています。

注意

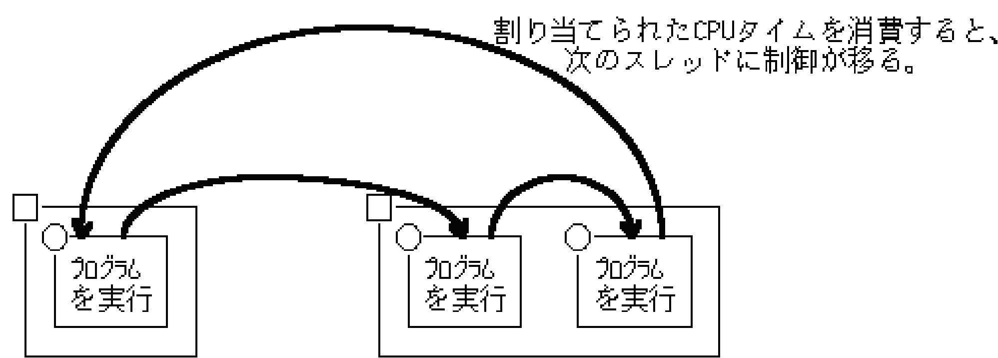
COBOLでは、スレッドを起動する機能を提供していません。

17.2 マルチスレッドのメリット

17.2.1 スレッドとは

スレッドとは、オペレーティングシステムによってスケジュール管理される最小の実行単位です。オペレーティングシステムは、実行するスレッドおよびその実行時期をスケジュールし、CPUタイムを割り当てます。割り当てられたCPUタイムを消費すると、次のスレッドにCPUタイムを割り当てます。

スレッドはプロセスの中に存在し、スレッドによってプログラムが実行されます。プロセスは、メモリ上にあるプログラムのコード、データ、オープンされているファイル、動的に割り当てられたメモリなどの資源から構成され、少なくとも1つのスレッドが存在します。



：実行制御の流れを示す。

：プロセスを示す。

：スレッドを示す。

以降、プロセスとスレッドをこの図で表記します。

17.2.2 マルチスレッドモデルとプロセスモデル

マルチスレッド環境下で動作しているアプリケーションには、マルチスレッド機能を有効にするためのシステムライブラリがリンクされています。なお、従来のプロセス環境下のアプリケーションには、このライブラリはリンクされていません。マルチスレッド機能を有効にするためのシステムライブラリは、システムの動作環境を変えてしまうため、それぞれの環境下で実行可能なモデルのプログラムを用意する必要があります。それが、マルチスレッドモデルのプログラムとプロセスモデルのプログラムです。

Webサーバなどの連携製品からマルチスレッド環境下で呼び出される場合、その連携製品から呼ばれるCOBOLプログラムはマルチスレッドモデルにする必要があります。なお、連携製品からプロセス環境下で呼び出される場合はプロセスモデルのプログラムを実行する必要があります。

プロセスモデルのプログラム

プロセスモデルのプログラムは、プロセス内の1つのスレッドでしか実行できません。このため、プロセス内の複数のスレッドで、同じプログラムを同時に実行することも、異なるプログラムを同時に実行することもできません。

プロセスモデルのCOBOLプログラムを作成するには、プロセスモデル用に翻訳されたオブジェクトファイルとプロセスモデルに対応したCOBOLランタイムシステムのリンクが必要です。

マルチスレッドモデルのプログラム

マルチスレッドモデルのプログラムは、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。

マルチスレッドモデルのCOBOLプログラムを作成するには、マルチスレッドモデル用に翻訳されたオブジェクトファイルとマルチスレッドモデルに対応したCOBOLランタイムシステムのリンクが必要です。

注意

マルチスレッドモデルのプログラムとプロセスモデルのプログラムを混在して実行することはできません。

17.2.3 マルチスレッドの効果

サーバから起動されるWebや分散オブジェクトなどの機能を利用したアプリケーションをマルチスレッドモデルのプログラムにすることによって、以下の効果が得られます。

高速なスタートアップ

プロセスモデルのプログラムを複数回実行した場合、実行した回数分のプロセスの初期処理を実行します。これに対して、複数のスレッドでマルチスレッドモデルの同じプログラムを実行する場合、2回目以降はプロセスの初期処理が不要となり、実行した回数分のスレッドの初期処理だけとなります。プロセスの初期処理の時間よりもスレッドの初期処理の時間の方が短いので、結果的に起動時間が短縮されます。

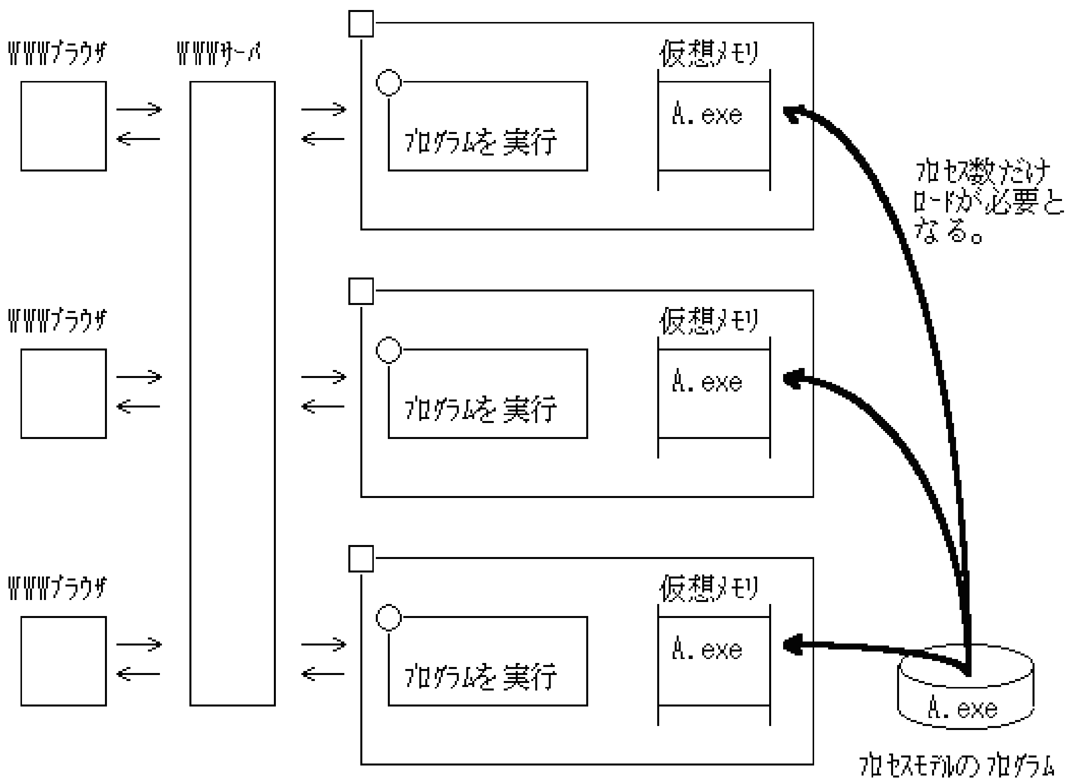
多重動作

マルチスレッドモデルのプログラムでは、プロセス内のすべてのスレッドで、プロセス空間を共有するため、メモリを節約することができます。このため、プログラムが多重に動作してもシステムへの負荷がプロセスモデルに比べて軽減されます。つまり、同じメモリ環境上では、マルチスレッドモデルの方が多くプログラムが同時に動作できるようになります。

以下に、Webサーバを利用した場合を例にとってマルチスレッドモデルのプログラムの効果を説明します。

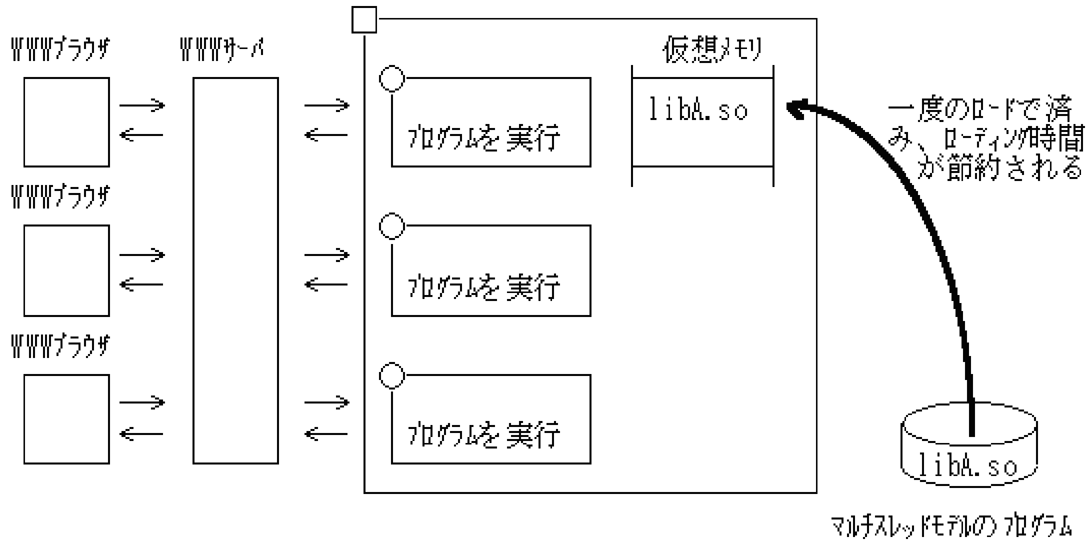
プロセスモデルのプログラム

プロセスモデルをベースとしているサーバアプリケーションには、CGI(Common Gateway Interface)のようなWebアプリケーションなどがあります。WebサーバはクライアントからのCGIアプリケーションの起動要求を受信すると、実行形式ファイルであるプロセスモデルのプログラムを新しいプロセスとして起動します。プロセスを起動するためには、プロセス空間の獲得、ロードモジュールのロード、データの初期化などの処理が必要です。また、必要なメモリ量もプロセスごとに獲得することになります。



マルチスレッドモデルのプログラム

マルチスレッドモデルをベースとしているサーバを利用した場合、サーバは新しいプロセスを起動するのではなく、プロセス中の1つのスレッドによってマルチスレッドモデルのプログラムを実行します。マルチスレッドモデルのプログラムは、共用オブジェクトファイルであり、最初に呼び出されたときに、プロセス空間にロードされ、通常はその後も常駐します。このため、スタートアップ時の処理が短くなるとともに、必要なメモリの量も削減されます。



17.3 COBOLプログラムの動作

プロセスモデルとマルチスレッドモデルで、COBOLプログラムの動作の違いについて説明します。

17.3.1 実行環境と実行単位

"9.1.1 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。マルチスレッドモデルでは、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。以下にマルチスレッドモデルでの実行環境と実行単位について説明します。

実行環境

マルチスレッドモデルの実行環境は、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、プロセスモデル同様、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、以下のものがあります。

- ファクトリオブジェクト
- オブジェクトインスタンス
- システムの標準入出力
- 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

参考

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するので注意してください。JMPCINT4の呼び出し形式については、"G.4 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。

実行単位

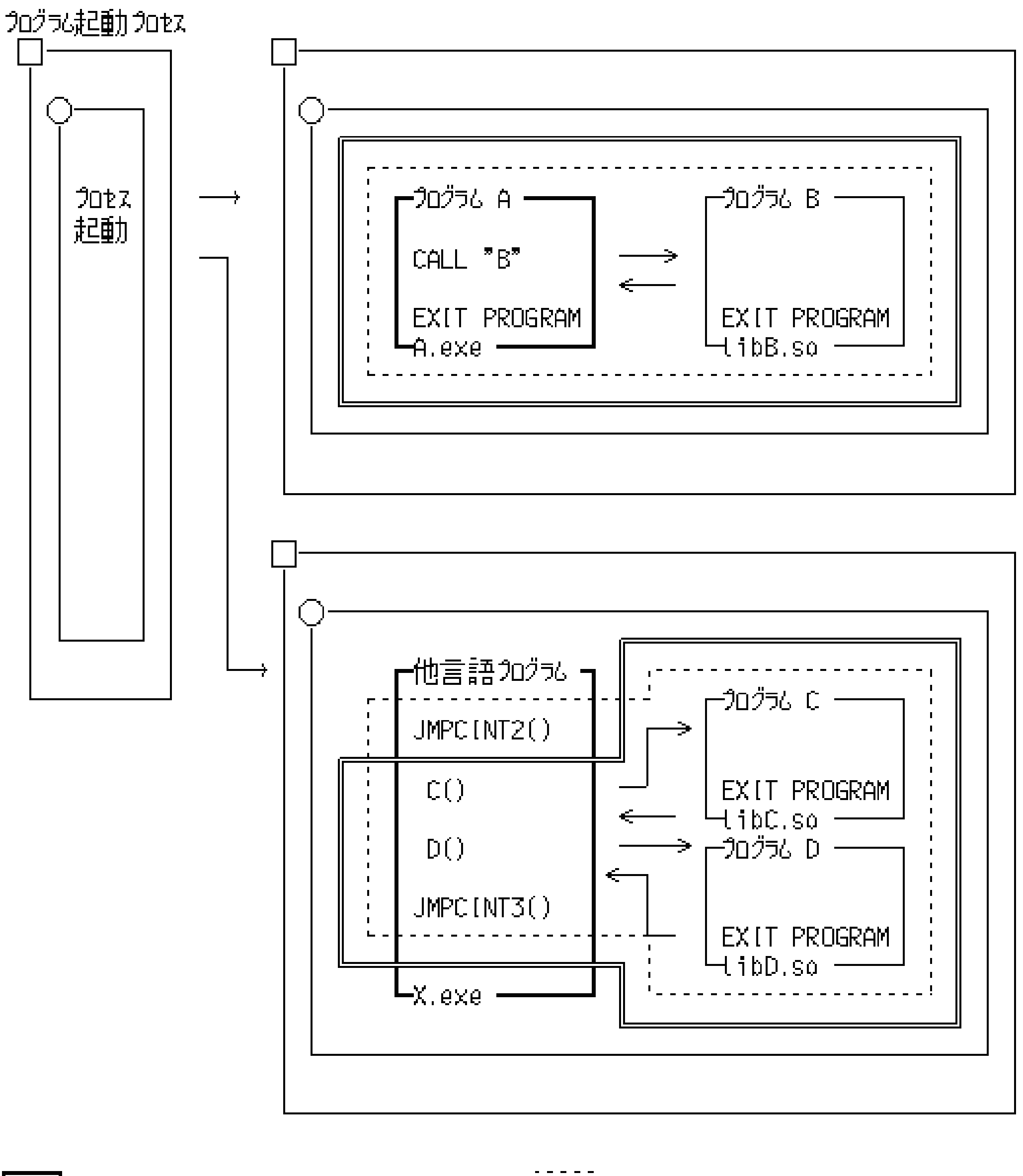
マルチスレッドモデルの実行単位の開始と終了のタイミングはプロセスモデルと同じです。しかし、COBOLプログラムが複数のスレッドで実行されるため、1つのプロセスに同時に複数の実行単位が存在することになります。実行単位の終了時には、スレッド単位で管理される資源の解放が行われます。スレッド単位で管理される資源には、プログラム定義に宣言されたデータ(スレッド間共有外部データ/外部ファイルは除きます)などがあります。

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ません。この場合、スレッドで利用された資源が解放されず、誤動作します。[参照]"G.4 他言語連携で使用するサブルーチン"

以下に、プロセスモデルのプログラムとマルチスレッドモデルのプログラムの実行環境と実行単位の関係を図示します。プロセスモデルのプログラムは、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、マルチスレッドモデルのプログラムは、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。

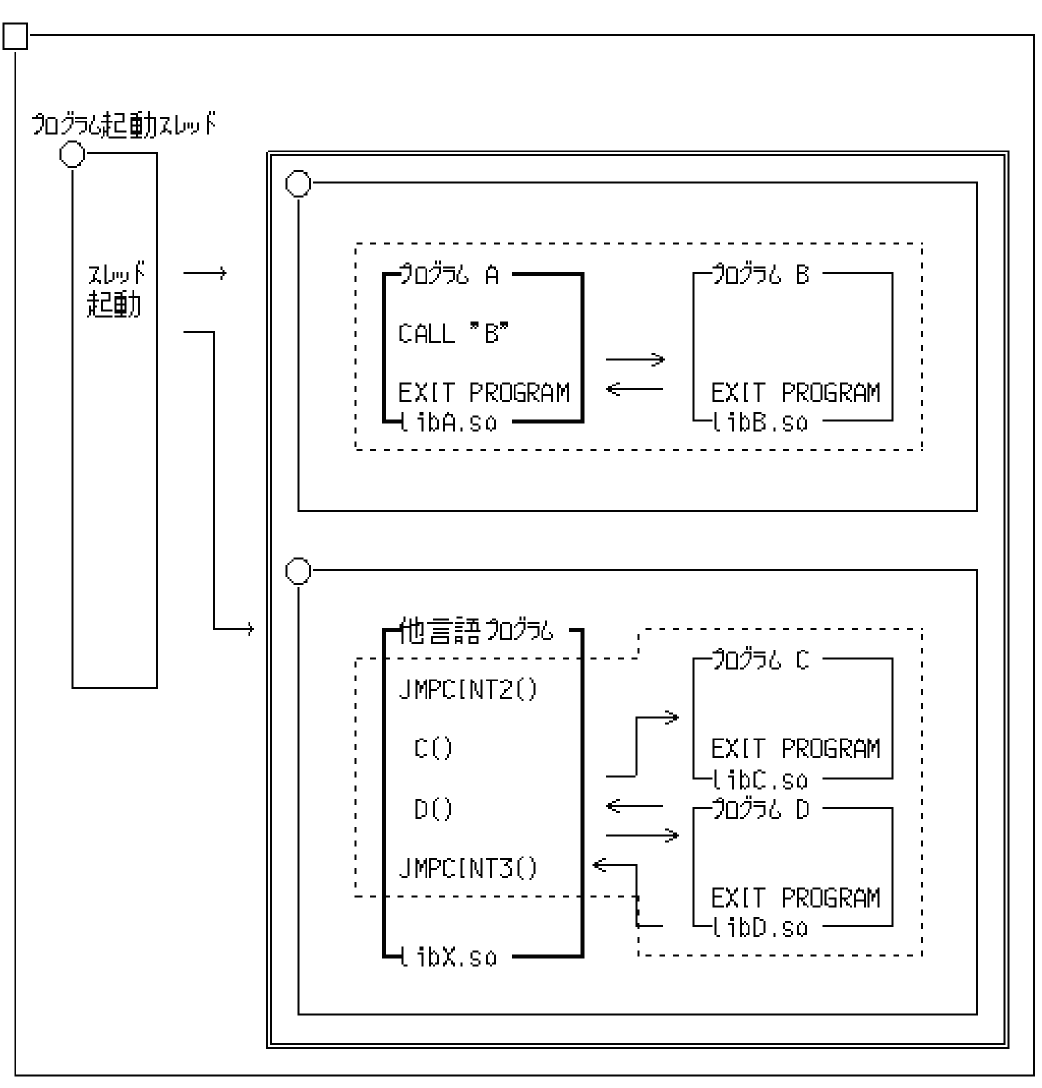
プロセスモデルのプログラム

プロセスモデルのプログラムでは、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ません。したがって、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。



マルチスレッドモデルのプログラム

マルチスレッドモデルのプログラムでは、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。



17.3.2 マルチスレッドモデルのプログラムのデータの扱い

ここでは、マルチスレッドモデルの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。

マルチスレッドモデルのプログラムには、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:マルチスレッドモデルのプログラム作成時に、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。

それぞれのデータについて、以下に説明します。

17.3.2.1 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などは強制的にクローズされます。

注意

プレ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。

プログラム定義に宣言されたデータとファイル

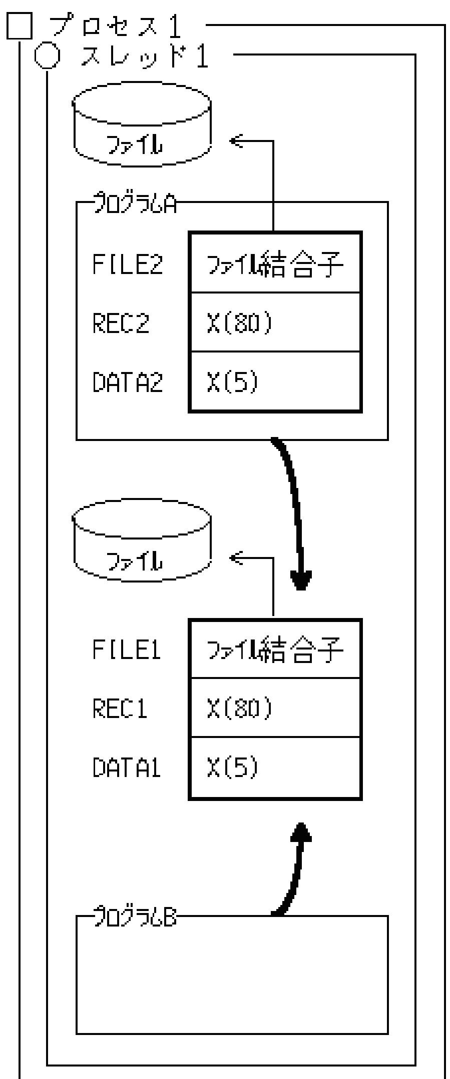
```
IDENTIFICATION DIVISION.
PROGRAM-ID. A.
    :
DATA DIVISION.
 FILE SECTION.
 FD  FILE1   EXTERNAL.
 01  REC1    PIC X(80).
 FD  FILE2.
 01  REC2    PIC X(80).
      :
WORKING-STORAGE SECTION.
 01  DATA1  PIC X(5) EXTERNAL.
 01  DATA2  PIC X(5).
      :
PROCEDURE DIVISION.
      :
    CALL "B".   --------->
      :
```

```
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.
      :
```

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。プロセスモデルのプログラムでは2つのプロセスが起動され、マルチスレッドモデルのプログラムでは2つのスレッドが起動されています。

プロセスモデルのプログラムでプロセスごとに確保されていたデータが、マルチスレッドモデルのプログラムではスレッドごとに確保されます。

図17.1 プロセスモデルのプログラム



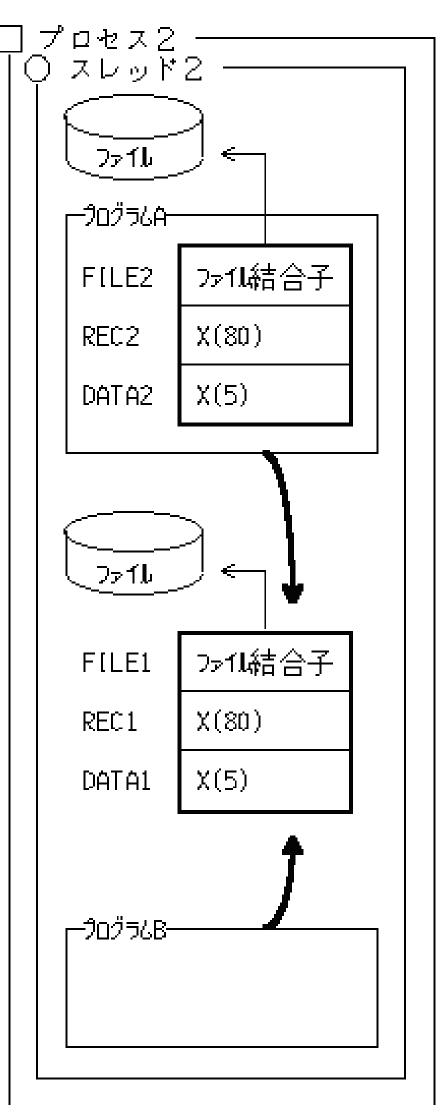
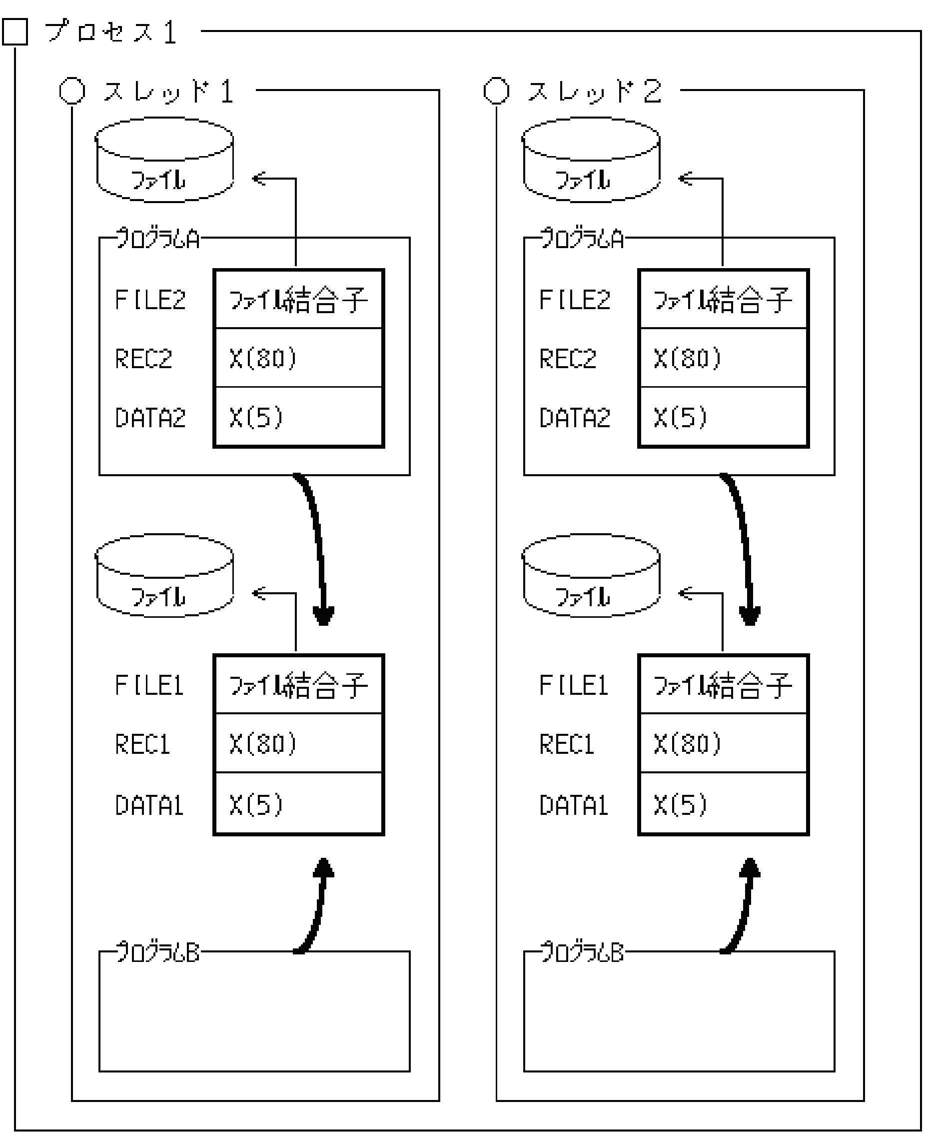


図17.2 マルチスレッドモデルのプログラム



17.3.2.2 ファクトリオブジェクトとオブジェクトインスタンス

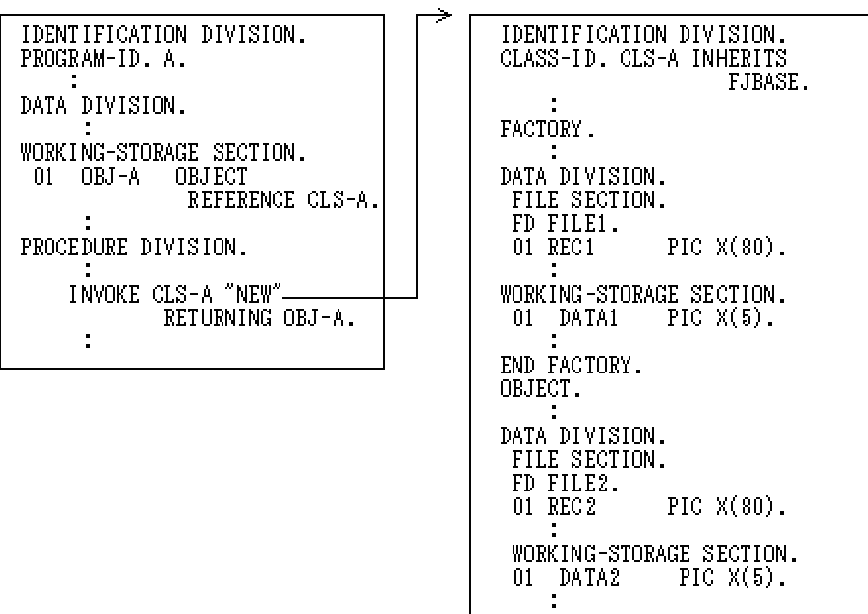
ファクトリオブジェクトとオブジェクトインスタンスはプロセスで管理されます。

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、マルチスレッドモデルのプログラムでは、常にスレッド間で共有されます。[参照]"17.4.3.2 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]"17.4.3.3 オブジェクトインスタンス"

実行環境の終了時に、ファクトリオブジェクトと未解放のオブジェクトインスタンスは解放されます。

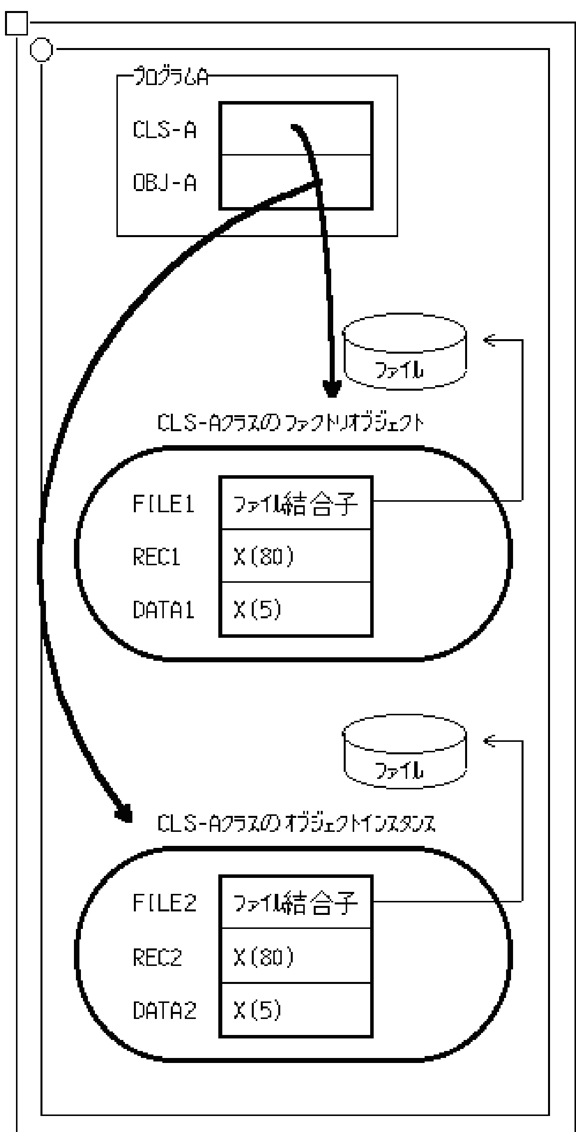
ファクトリ定義とオブジェクト定義に宣言されたデータとファイル



以下の図は、上記のプログラムが2つ起動された場合を表しています。プロセスモデルのプログラムでは2つのプロセスが起動され、マルチスレッドモデルのプログラムでは2つのスレッドが起動されています。

マルチスレッドモデルのプログラムでは、プロセスモデルの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、マルチスレッドモデルのプログラムでは、ファクトリオブジェクトが常にスレッド間で共有されることになります。

図17.3 プロセスモデルのプログラム



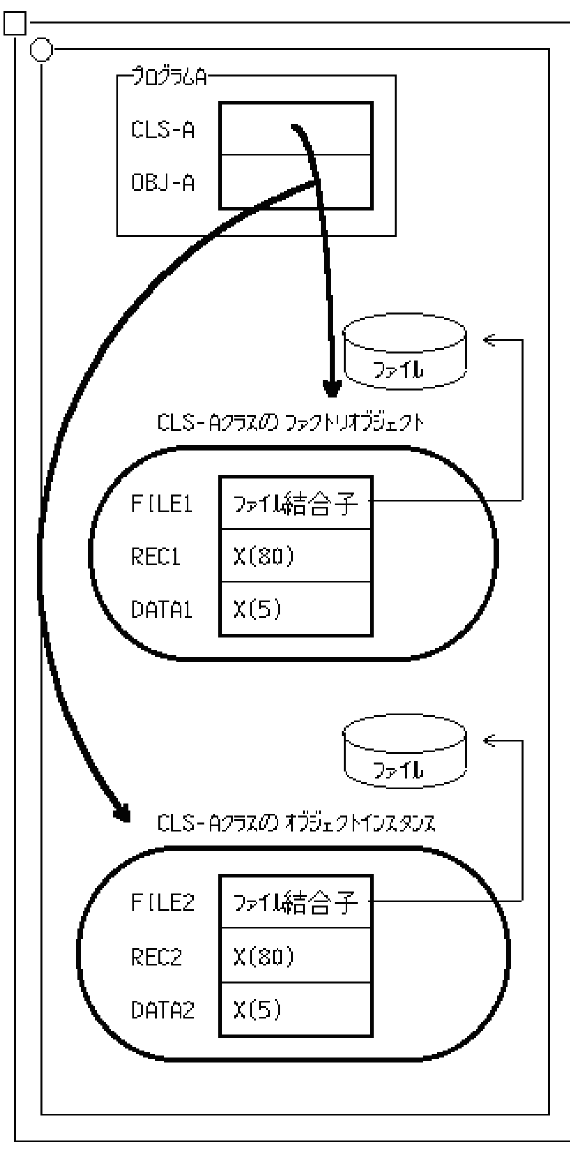
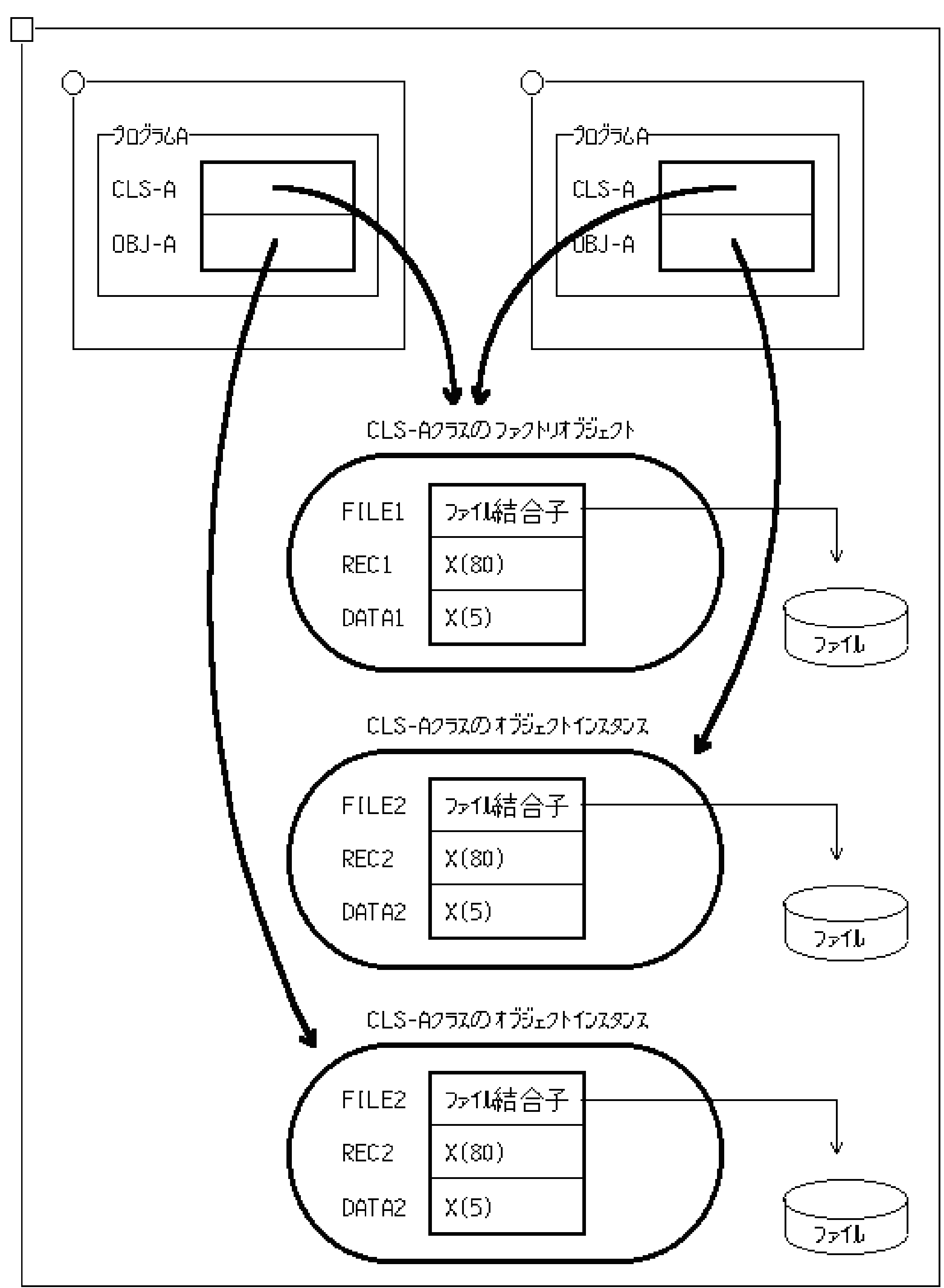


図17.4 マルチスレッドモデルのプログラム



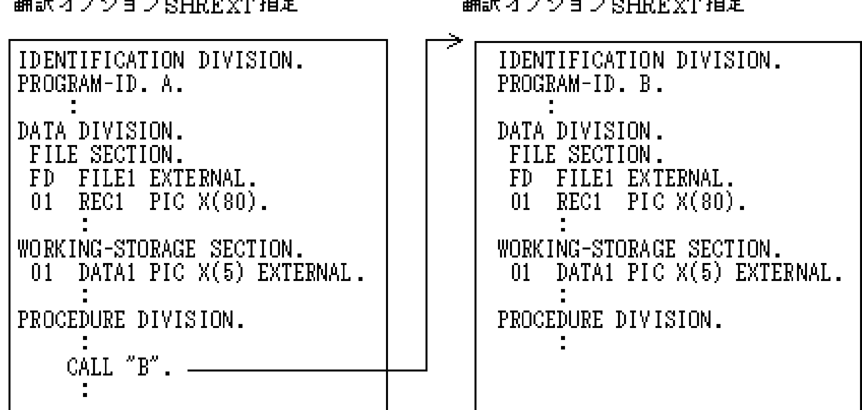
17.3.2.3 メソッド定義に宣言されたデータ

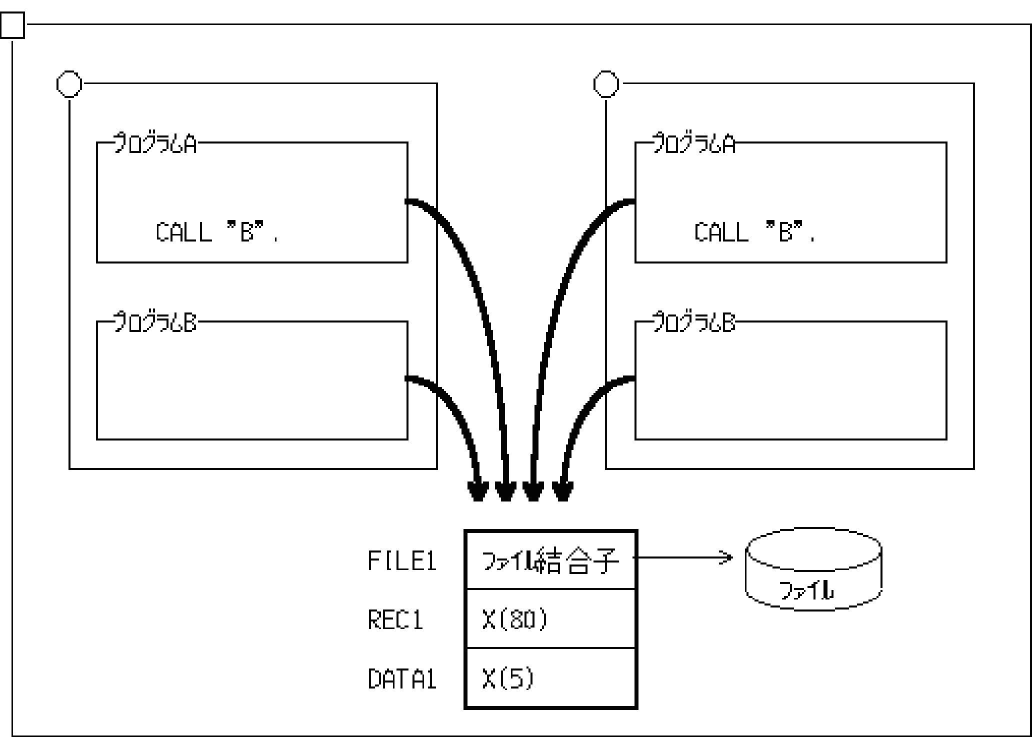
メソッド定義に宣言されたデータの割り付けは、プロセスモデルのプログラムと変わりありません。つまり、メソッドの呼出し時に確保され、そのメソッドの呼出し元に戻るときに解放されます。メソッドの呼出し元に戻るとき、クローズされていないファイルは強制的にクローズされます。

17.3.2.4 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、マルチスレッドモデルのプログラム翻訳時に翻訳オプションSHREXTを指定することで、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、マルチスレッドモデルのプログラムが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。





17.4 スレッド間の資源の共有

COBOLのマルチスレッドモデルのプログラムでは、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作成できます。

17.4.1 資源の共有

プロセスモデルのプログラムでは、資源をプロセス間で共有することはできませんでした。これに対して、マルチスレッドモデルのプログラムでは、資源をスレッド間で共有することができます。スレッド間で資源を共有することで、別々に起動したプログラムでオープン済みの同じファイルを同時にアクセスすることが可能になったり、プログラム間の連携処理を簡単に実現することができます。

17.4.2 競合状態

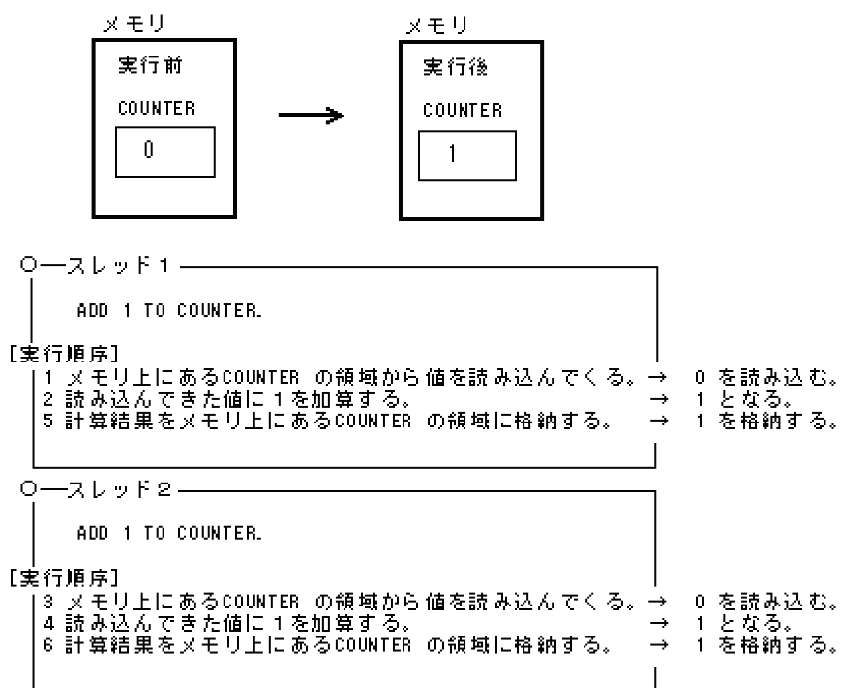
ここでは、スレッド間で資源を共有する場合に、一般的に発生する競合状態について説明します。

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。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む。
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1となってしまいます（期待している値は2です）。



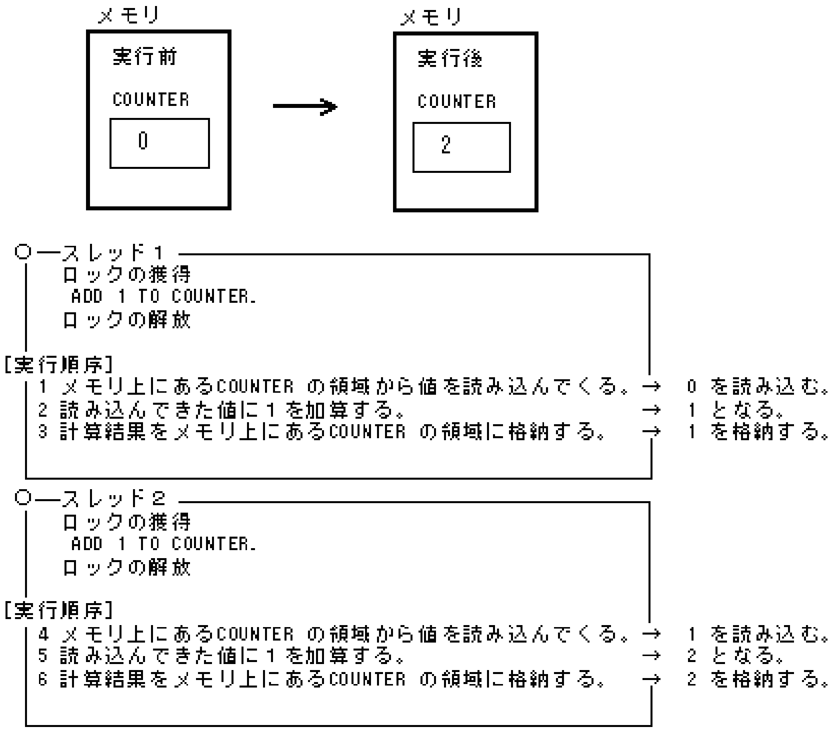
このような競合状態が発生するのは、次の条件のときです。もちろん、同一データをすべてのスレッドが参照するだけなら、同時にデータをアクセスしても問題ありません。

- 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

同期制御

同期制御とは、競合状態を発生させないようにするために、共有する資源をアクセスする一連の手続きを連続に動作することを保証するための機構です。手続きの連続動作を保証するために、実行権（「ロック」と呼びます）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。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)。



17.4.3 COBOLでの資源の共有

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、“17.9 スレッド同期制御サブルーチン”を参照してください。

17.4.3.1 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、マルチスレッドモデルのプログラム翻訳時に翻訳オプションSHREXTを指定することで、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、“17.6.1.1 スレッド間共有外部ファイル”を参照してください。

```
○─初期化プログラム
      :
  WORKING-STORAGE SECTION.
  01  総販売数        PIC 9(9) COMP-5 EXTERNAL.
  01  総売上          PIC 9(9) COMP-5 EXTERNAL.
      :
 PROCEDURE DIVISION.
      :
     MOVE 0 TO 総販売数.
     MOVE 0 TO 総売上.                              ] [1]
      :
```

```
○─データ更新プログラム3
○┴データ更新プログラム2
○┴データ更新プログラム1
      :
  WORKING-STORAGE SECTION.
  01  総販売数        PIC 9(9) COMP-5 EXTERNAL.
  01  総売上          PIC 9(9) COMP-5 EXTERNAL.
  01  販売数          PIC 9(9) COMP-5.
  01  売上            PIC 9(9) COMP-5.
  01  LOCK-KEY        PIC X(30) VALUE "TOTAL".
  01  WAIT-TIME       PIC S9(9) COMP-5 VALUE -1.
  01  ERR-DETAIL      PIC 9(9) COMP-5.
  01  RET-VALUE       PIC S9(9) COMP-5.
      :
 PROCEDURE DIVISION.
      :

     CALL "COB_LOCK_DATA" USING BY REFERENCE LOCK-KEY
                                BY VALUE WAIT-TIME
                                BY REFERENCE ERR-DETAIL
                          RETURNING RET-VALUE.      … [2]

     ADD 販売数 TO 総販売数.
     ADD 売上   TO 総売上.                           ] [3]

     CALL "COB_UNLOCK_DATA" USING BY REFERENCE LOCK-KEY
                                  BY REFERENCE ERR-DETAIL
                            RETURNING  RET-VALUE.   … [4]
```

[1] 初期化プログラムで、共有データを初期化しています。なお、初期化プログラムは、ほかのデータ更新プログラムよりも先に実行される必要があります。

[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、"17.9.1 データロックサブルーチン"を参照してください。

[3] 共有データに値を加算しています。この処理は、[2]でロックを獲得したスレッドにより実行されます。

[4] "TOTAL"というデータ名に対応するロックキーに対して獲得したロックを解放しています。ロックの解放により、別のスレッドで、[2]と同様の処理でロックを獲得できます。

17.4.3.2 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"を参照してください。この自動的に行うスレッドの同期制御により、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。

```
  ○─ファクトリオブジェクトの操作プログラム3
 ○┴ファクトリオブジェクトの操作プログラム2
○┴ファクトリオブジェクトの操作プログラム1
       :
WORKING-STORAGE SECTION.
01  保険額       PIC 9(9) COMP-5.
01  契約年       PIC 9(9) COMP-5.
01  金額         PIC 9(9) COMP-5.
01  OBJ          USAGE OBJECT REFERENCE FACTORY OF EMPLOYEE.
01  WAIT-TIME    PIC S9(9) COMP-5 VALUE -1.
01  ERR-DETAIL   PIC 9(9) COMP-5.
01  RET-VALUE    PIC S9(9) COMP-5.
       :
PROCEDURE DIVISION.
       :
    SET OBJ TO EMPLOYEE.
    CALL "COB_LOCK_OBJECT" USING BY REFERENCE OBJ
                                 BY VALUE WAIT-TIME
                                 BY REFERENCE ERR-DETAIL
                           RETURNING RET-VALUE.      … [1]

    MOVE  保険額 TO 金額 OF  EMPLOYEE.               … [2]
    MOVE  契約年 TO 年数 OF  EMPLOYEE.               … [3]
    INVOKE EMPLOYEE  "合計" RETURNING 金額.           … [4]
    CALL "COB_UNLOCK_OBJECT" USING BY REFERENCE OBJ
                                   BY REFERENCE ERR-DETAIL
                             RETURNING  RET-VALUE.   … [5]
       :
```

[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、"17.9.2 オブジェクトロックサブルーチン"を参照してください。

[2] EMPLOYEEクラスのファクトリオブジェクトのプロパティメソッドを呼び出し、保険額をファクトリデータに設定しています。

[3] EMPLOYEEクラスのファクトリオブジェクトのプロパティメソッドを呼び出し、契約年をファクトリデータに設定しています。

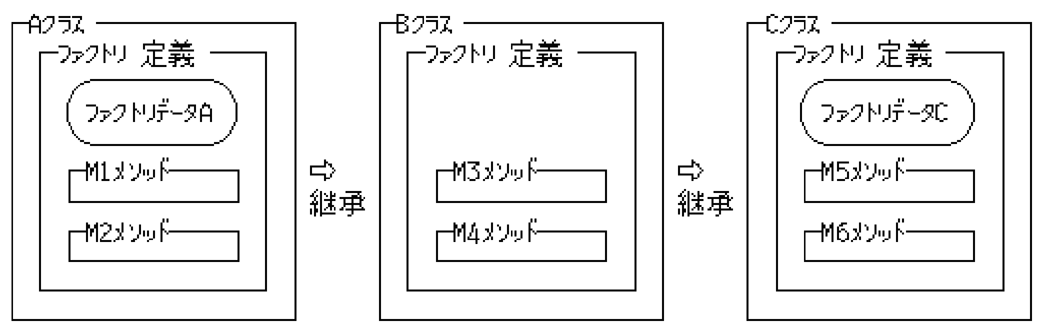
[4] EMPLOYEEクラスのファクトリオブジェクトの合計メソッドを呼び出し、金額を求めています。[2]～[4]までの処理は、[1]でロックを獲得したスレッドにより実行されます。

[5] ファクトリオブジェクトのロックを解放しています。ロックの解放により、別のスレッドで、[1]と同様の処理でロックを獲得できます。

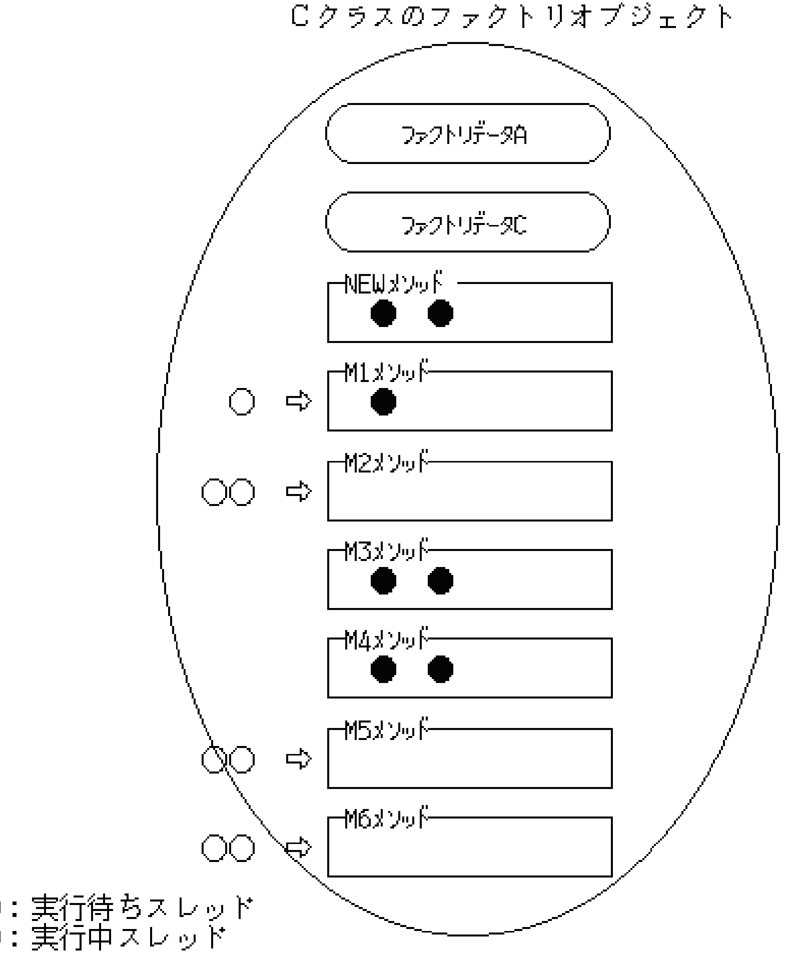
COBOLランタイムシステムによる同期制御

ここでは、マルチスレッドモデルのプログラムで、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ます。したがって、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。

17.4.3.3 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。

```
○─初期化プログラム
     :
WORKING-STORAGE SECTION.
01 OBJ1 USAGE
     OBJECT REFERENCE C1.
     :
PROCEDURE DIVISION.
     INVOKE C1 "NEW"
          RETURNING OBJ1.
     :
     SET SOBJ OF C1 TO OBJ1. … [1]
     :
```

```
IDENTIFICATION DIVISION.
CLASS-ID. C1 INHERITS FJBASE.
     :
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 SOBJ USAGE
     OBJECT REFERENCE C1 PROPERTY.
     :
END FACTORY.
OBJECT.
     :
METHOD-ID. M1.
     :
```

```
○─オブジェクトインスタンスの操作プログラム3
○─オブジェクトインスタンスの操作プログラム2
○─オブジェクトインスタンスの操作プログラム1
     :
WORKING-STORAGE SECTION.
01 OBJ2           USAGE OBJECT REFERENCE C1.
01  WAIT-TIME     PIC S9(9) COMP-5 VALUE -1.
01  ERR-DETAIL    PIC 9(9) COMP-5.
01  RET-VALUE     PIC S9(9) COMP-5.
     :
PROCEDURE DIVISION.
     SET OBJ2 TO SOBJ OF C1.                              … [2]
     CALL "COB_LOCK_OBJECT" USING BY REFERENCE OBJ2
                                  BY VALUE WAIT-TIME
                                  BY REFERENCE ERR-DETAIL
                            RETURNING RET-VALUE.         … [3]

     INVOKE OBJ2 "M1".                                    … [4]

     CALL "COB_UNLOCK_OBJECT" USING BY REFERENCE OBJ2
                                    BY REFERENCE ERR-DETAIL
                              RETURNING  RET-VALUE.      … [5]
```

```
○─終了スレッド
     :
PROCEDURE DIVISION.
     :
     SET SOBJ OF C1 TO NULL. … [6]
     :
```

[1] 初期化プログラムで、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。なお、初期化プログラムは、ほかのオブジェクトインスタンス操作プログラムよりも先に実行される必要があります。

[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が未定義であるか、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、"17.9.2 オブジェクトロックサブルーチン"を参照してください。

[4] C1クラスのオブジェクトインスタンスのM1メソッドを呼び出し、処理を行っています。この処理は、[3]でロックを獲得したスレッドにより実行されます。

[5] オブジェクトインスタンスのロックを解放しています。ロックの解放により、別のスレッドが[3]でロックを獲得できます。

[6] 終了スレッドは、実行環境で最後に1回だけ起動されます。終了スレッドで、C1クラスのファクトリオブジェクトのプロパティメソッドを呼び出し、ファクトリデータに設定されているオブジェクトインスタンスを削除しています。

17.5 基本的な使い方

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プログラムであれば、マルチスレッドモデルのオプションを指定して再翻訳・再リンクするだけでマルチスレッド環境へ簡単に移行できます。[参照]“17.3.2 マルチスレッドモデルのプログラムのデータの扱い” “17.7.1 翻訳とリンク”

ファクトリオブジェクトにファクトリデータまたはファイルを持つ場合は、“17.4 スレッド間の資源の共有”を必ず読んでください。

プロセスモデルのプログラムからマルチスレッドモデルの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。

17.5.1 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。

マルチスレッドモデルのプログラムでも、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共用オブジェクトプログラム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。

17.5.2 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モデルのプログラムを作成する方法について説明します。

17.5.2.1 同一ファイルの共有

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。

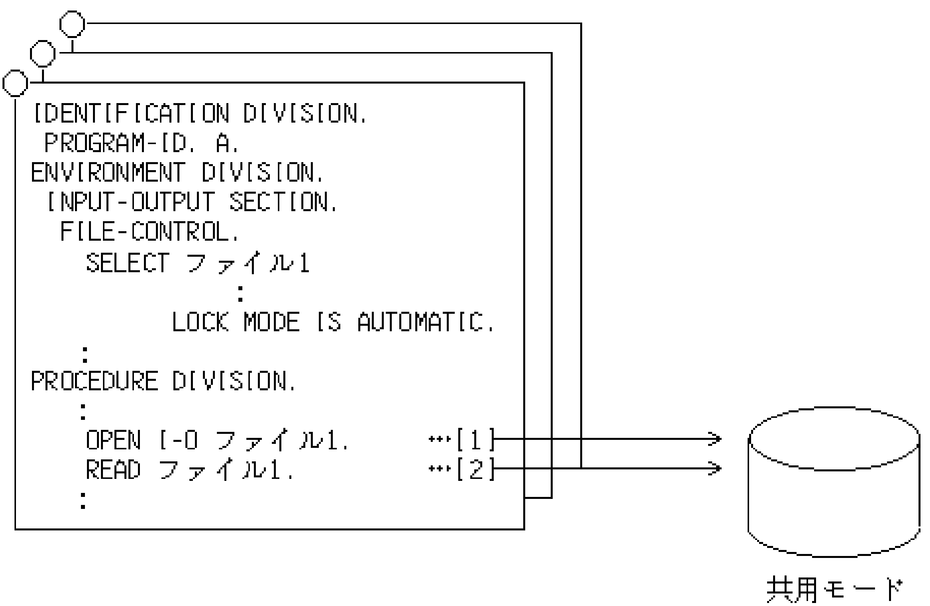
ここでは、内部属性を持つファイル結合子を操作して同一ファイルを共有する方法について説明します。

プログラム内/メソッド内/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。

プロセスモデルの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、"6.7.3 ファイルの排他制御"を参照してください。

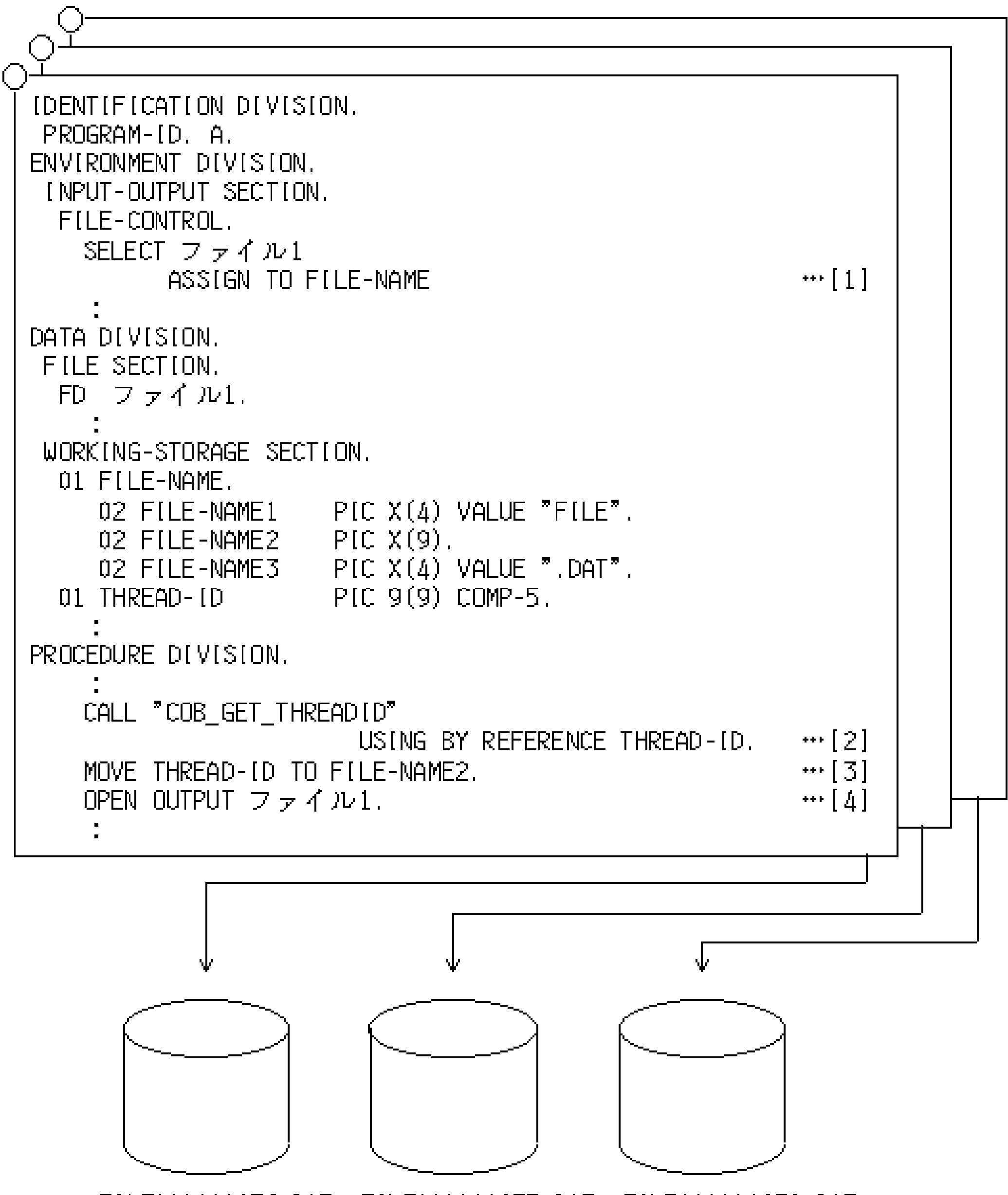
注意

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、"17.6 少し進んだ使い方"を参照してください。

17.5.2.2 同一プログラムでの複数ファイルの操作

同一プログラムをマルチスレッドモデルの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、“G.3.2 スレッドID取得サブルーチン”を参照してください。

注意

スレッドIDは、プログラムを実行するたびに変化します。このため、スレッドIDをファイル名とする場合は、一時的な作業ファイルとして利用してください。

17.5.2.3 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モデルの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。マルチスレッドモデルの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。

17.5.3 印刷機能の利用

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ります。したがって、印刷データの出力順序が一定となりません。このため、意図しない印刷結果を得ることがあります。

意図した印刷結果を得るためには、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、"17.5.2 入出力機能の利用"を参照してください。

17.5.4 スクリーン操作機能の利用

マルチスレッドモデルのプログラムで、スクリーン操作機能を使用する場合、以下の点に注意してください。

- スクリーン操作機能を同時に複数のスレッドから呼び出さない

  スクリーン操作機能は、システムのもつCURSES機能により実装されています。ただし、CURSES機能はマルチスレッドで同時に動作した場合に、その動作は保証されません。このため、マルチスレッドモデルのプログラムでは、1つのスレッドだけでスクリーン操作機能を使用するようにしてください。

  また、他言語プログラムがCURSES機能を使用している場合も同様の注意が必要となります。

- スクリーン操作機能を呼び出すスレッドと、画面を使用した小入出力機能を呼び出すスレッドを混在させた場合の注意

  スクリーン操作機能と、画面を使用した小入出力機能では、同じ画面を使用します。このため、スクリーン操作機能の動作するスレッドと、画面を使用する小入出力機能の動作するスレッドが混在した場合、その動作については保証されません。

  このような場合は、小入出力機能の入出力先をファイルに割り当てるようにしてください。

  また、スクリーン操作機能の動作中に他のスレッドで実行時メッセージが発生した場合についても、実行時メッセージの出力先をファイルに割り当てることをおすすめします。

17.5.5 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。

17.5.5.1 小入出力機能について

マルチスレッドモデルのプログラムでも、システムの標準入出力およびファイルを使用した小入出力では、プロセスで1つの標準入出力およびファイルを共有します。

入出力の例

図17.5 システムの標準入出力を使用する場合

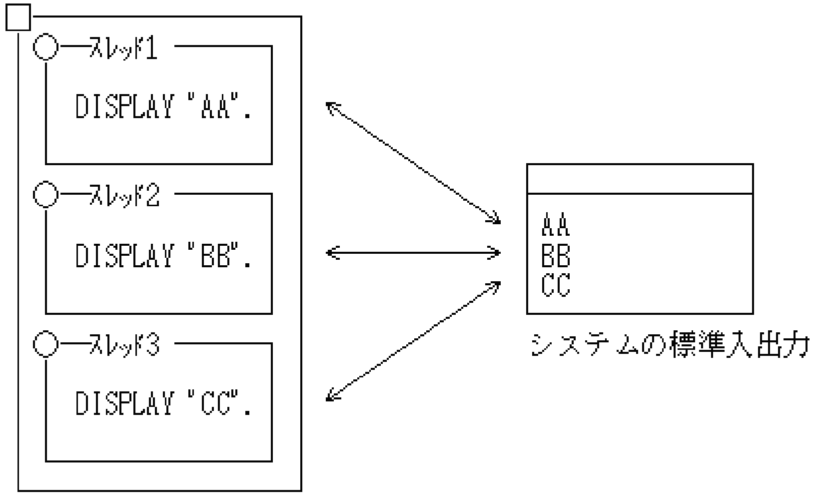
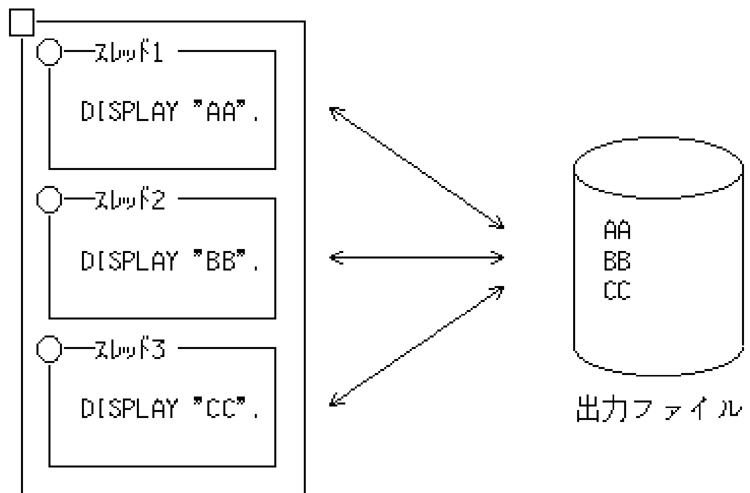


図17.6 ファイルを使用する場合



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、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場合があります。

実行順序の同期制御を行いたい場合(たとえば、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してください。[参照]"17.9 スレッド同期制御サブルーチン"

複数の入出力文の同期制御の例([1]～[3]:実行順序)

図17.7 スレッド同期制御サブルーチンを使用しない場合

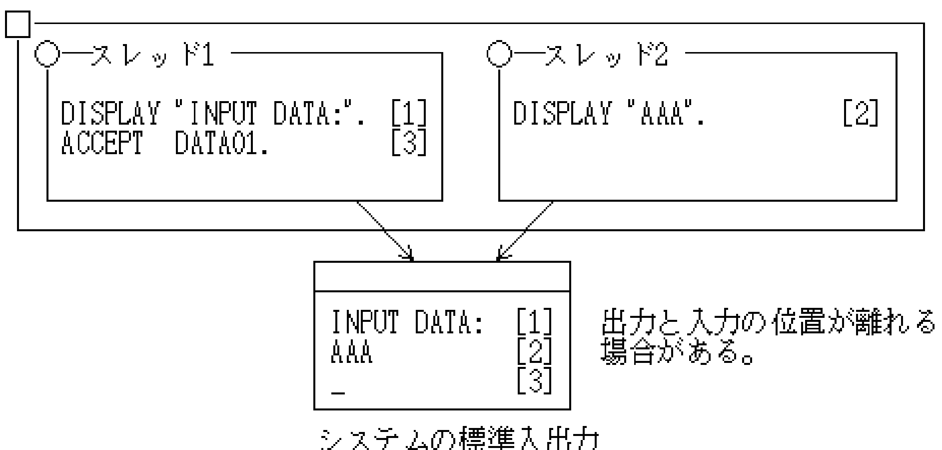
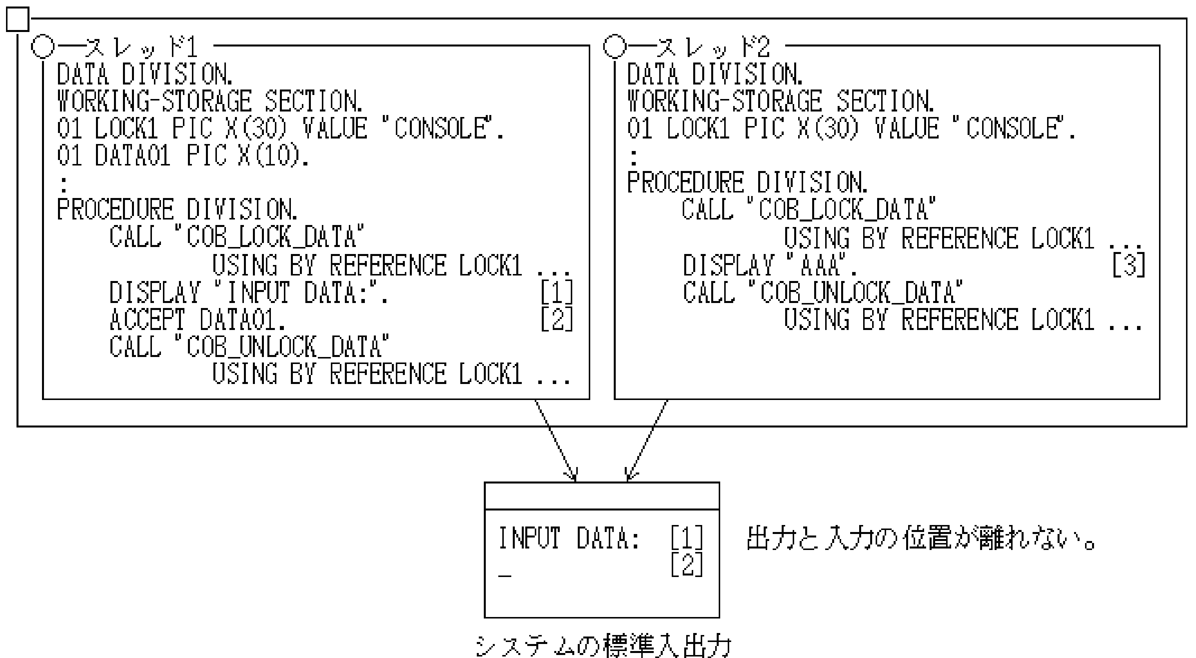


図17.8 スレッド同期制御サブルーチンを使用した場合



DISPLAY文およびACCEPT文で、以下の内容については、同一実行環境では最初に動作したDISPLAY文または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。

- 標準入出力とファイルのどちらを使用するか。
- ファイルを使用する場合、どのファイル識別名が有効となるか。

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CBR_MESSOUTFILEに出力するファイルを指定してください。

17.5.5.2 コマンド行引数および環境変数の操作機能について

17.5.5.2.1 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。

17.5.5.2.2 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、マルチスレッドモデルのプログラム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。

17.5.6 オブジェクト指向プログラミング機能の利用

プロセスモデルのプログラムでは、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ました。したがって、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、特に問題は発生しませんでした。

これに対して、マルチスレッドモデルのプログラムでは、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、オブジェクトを破棄しないで実行単位を終了すると、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ます。これがメモリ不足につながります。したがって、必ず、オブジェクトを破棄するために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。マルチスレッドモデルのプログラムの実行単位と実行環境については、"17.3.1 実行環境と実行単位"を参照してください。

17.5.7 簡易アプリ間通信機能の利用

簡易アプリ間通信機能を使用して、マルチスレッドモデルのメッセージ通信を行うプログラムを作成することができます。この場合、実行可能プログラムまたは副プログラムの共用ライブラリのリンク時に、librcobcicl.soをリンクしてください。librcobcicl.soは、COBOLランタイムシステムの一部として提供される共用ライブラリです。

なお、マルチスレッドモードでは、簡易アプリ間通信機能の各サブルーチンで使用するサーバ識別子をスレッド間で共用できません。各スレッドごとに一意のサーバ識別子を使用してください。

17.5.8 連携機能の利用

ここでは、マルチスレッドモデルで関連製品を利用する場合の注意事項について説明します。

ここで説明しない関連製品については、その関連製品の対応状況を確認のうえ、使用してください。

17.5.8.1 Symfoware連携

Symfoware RDBにアクセスするマルチスレッドモデルのプログラムは、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。

17.5.8.1.1 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。

17.5.8.1.2 プログラムの翻訳・リンク

sqlcobolシェルプロシジャにより翻訳・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。

- sqlcobolのCOBOLオプションに-cを指定してください。
- 翻訳・リンクについては“17.7 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、オプション -l sqldrvm および -L /opt/FSUNrdb2b/libを指定してください。

17.5.8.1.3 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、環境変数情報CBR_SYMFOWARE_THREAD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]“17.7.2.2.1 実行環境変数の指定形式”の“CBR_SYMFOWARE_THREAD(Symfoware連携でマルチスレッド動作可能にする指定)”

17.5.9 多重動作ができないプログラムの呼出し

以下のようなプログラムを呼び出している場合、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。データロックサブルーチンについては、“17.9.1 データロックサブルーチン”を参照してください。

- マルチスレッドモデルであるが、複数のスレッドが同時に動作しない環境でだけ、その動作を保証しているプログラム(関連製品から提供されたプログラムを含む)

17.6 少し進んだ使い方

17.6.1 入出力機能の利用

基本的な使い方では、スレッド間で異なるファイル結合子を操作したファイルの共有について説明しました。

ここでは、スレッド間で同じファイル結合子を操作してファイルを共有する方法について説明します。

スレッド間で同じファイル結合子を操作するには、以下の方法があります。

- スレッド間共有外部ファイル
- ファクトリオブジェクト内に定義したファイル
- オブジェクト内に定義したファイル

注意

RDMファイルは、スレッド間で同じファイル結合子を操作することはできません。

各ファイルの利用方法について、以降で説明します。

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。

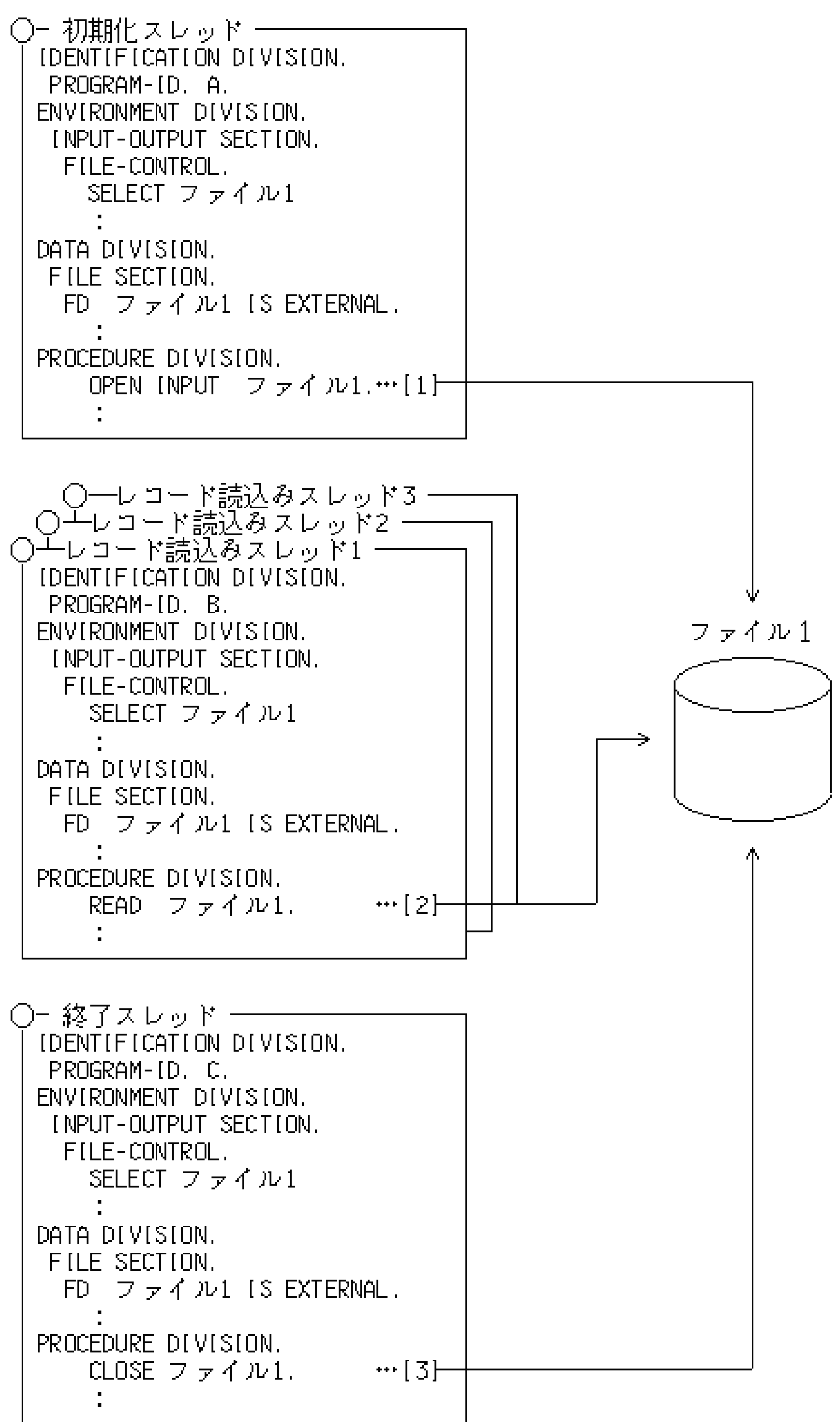
17.6.1.1 スレッド間共有外部ファイル

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、マルチスレッドモデルのプログラム翻訳時に翻訳オプション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。

以下に、スレッド間共有外部ファイルを使用したプログラム例を示します。



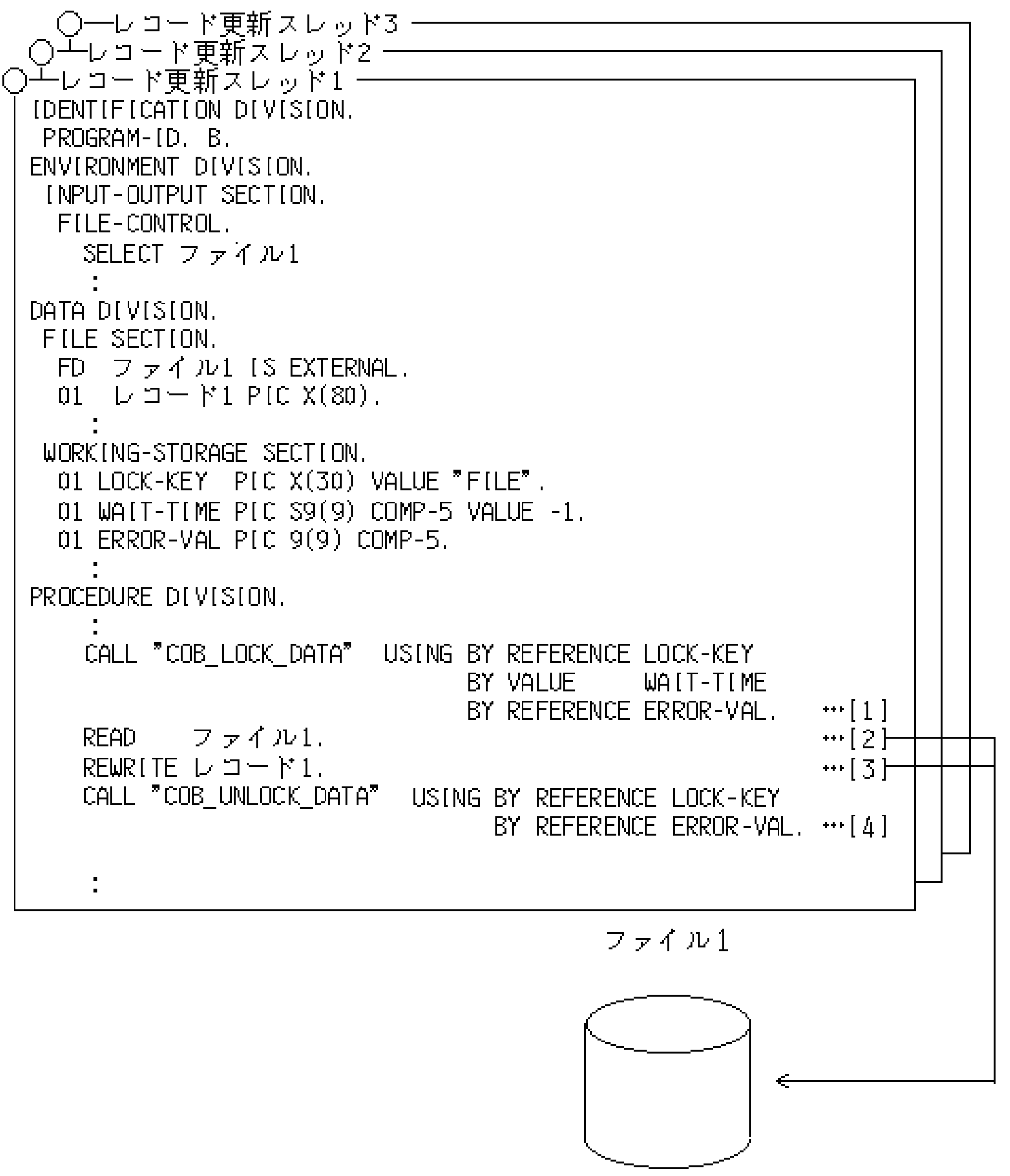
[1] 初期化スレッドは、実行環境で最初に1回だけ起動されます。初期化スレッドで、外部属性のファイル結合子を持つファイルを開きます。

[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、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、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を使用します。このサブルーチンを使用することにより、たとえば、READ文と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。

以下に、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を示します。



[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。データロックサブルーチンについては、“17.9.1 データロックサブルーチン”を参照してください。

17.6.1.2 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。

以下に、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を示します。

初期化スレッド

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. A.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
  REPOSITORY.
    CLASS X.
     :
PROCEDURE DIVISION.
     INVOKE X "M-OPEN".           …[1]
     :
```

レコード読込みスレッド3
レコード読込みスレッド2
レコード読込みスレッド1

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. B.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
  REPOSITORY.
    CLASS X.
     :
PROCEDURE DIVISION.
     INVOKE X "M-READ".           …[2]
     :
```

終了スレッド

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. C.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
  REPOSITORY.
    CLASS X.
     :
PROCEDURE DIVISION.
     INVOKE X "M-CLOSE".          …[3]
     :
```

```
CLASS-ID. X  INHERITS FJBASE.
FACTORY.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT ファイル1
     :
DATA DIVISION.
 FILE SECTION.
  FD  ファイル1.
     :
METHOD-ID. M-OPEN.
PROCEDURE DIVISION.
    OPEN INPUT ファイル1.
     :
END METHOD M-OPEN.

METHOD-ID. M-READ.
PROCEDURE DIVISION.
    READ ファイル1.
     :
END METHOD M-READ.

METHOD-ID. M-CLOSE.
PROCEDURE DIVISION.
    CLOSE  ファイル1.
     :
END METHOD M-CLOSE.
END FACTORY.
```

ファイル1

[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。

以下に、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を示します。

```
○─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.
  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 OBJ1 USAGE OBJECT REFERENCE FACTORY X.
     :
 PROCEDURE DIVISION.
     :
     SET OBJ1 TO X.
     CALL "COB_LOCK_OBJECT" USING BY REFERENCE X
                                  BY VALUE     WAIT-TIME
                                  BY REFERENCE ERROR-VAL.   …[1]
     INVOKE X "M-READ".                                     …[2]
     INVOKE X "M-REWRITE".                                  …[3]
     CALL "COB_UNLOCK_OBJECT" USING BY REFERENCE X
                                    BY REFERENCE ERROR-VAL. …[4]

     :
```

[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。

オブジェクトロックサブルーチンについては、"17.9.2 オブジェクトロックサブルーチン"を参照してください。

17.6.1.3 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。

以下に、オブジェクト内のファイルを使用したプログラム例を示します。

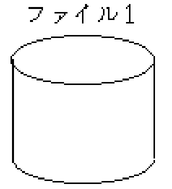
○─初期化スレッド
```
IDENTIFICATION DIVISION.
 PROGRAM-ID. A.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
  REPOSITORY.
    CLASS X.
    :
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ1 USAGE OBJECT REFERENCE X.
    :
PROCEDURE DIVISION.
    INVOKE X "NEW" RETURNING OBJ1.…[1]
    INVOKE OBJ1 "M-OPEN".         …[2]
    SET FOBJ OF X TO OBJ1.        …[3]
    :
```

○─レコード読込みスレッド3
○─レコード読込みスレッド2
○─レコード読込みスレッド1
```
IDENTIFICATION DIVISION.
 PROGRAM-ID. B.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
  REPOSITORY.
    CLASS X.
    :
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ2 USAGE OBJECT REFERENCE X.
01 WAIT-TIME PIC S9(9) COMP-5 VALUE -1.
01 ERR-VAL   PIC  9(9) COMP-5.
01 RTN-VAL   PIC S9(9) COMP-5.
    :
PROCEDURE DIVISION.

    SET OBJ2 TO FOBJ OF X.          …[4]
    CALL "COB_LOCK_OBJECT"
              USING BY REFERENCE OBJ2
                    BY VALUE WAIT-TIME
                    BY REFERENCE ERR-VAL
              RETURNING RET-VAL.    …[5]
    INVOKE OBJ2 'M-READ'.           …[6]
    CALL "COB_UNLOCK_OBJECT"
              USING BY REFERENCE OBJ2
                    BY REFERENCE ERR-VAL
              RETURNING RET-VAL.    …[7]
    :
```

○─終了スレッド
```
IDENTIFICATION DIVISION.
 PROGRAM-ID. C.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
  REPOSITORY.
    CLASS X.
    :
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OBJ3 USAGE OBJECT REFERENCE X.
    :
PROCEDURE DIVISION.
    SET OBJ3 TO FOBJ OF X.        …[8]
    INVOKE OBJ3 "M-CLOSE".        …[9]
    :
```

```
CLASS-ID. X  INHERITS FJBASE.
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 FOBJ USAGE OBJECT
         REFERENCE X PROPERTY.
    :
END FACTORY.
OBJECT.
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT ファイル1
    :
DATA DIVISION.
 FILE SECTION.
  FD  ファイル1.
    :
METHOD-ID. M-OPEN.
PROCEDURE DIVISION.
    OPEN INPUT ファイル1.
    :
END METHOD M-OPEN.

METHOD-ID. M-READ.
PROCEDURE DIVISION.
    READ ファイル1.
    :
END METHOD M-READ.

METHOD-ID. M-CLOSE.
PROCEDURE DIVISION.
    CLOSE  ファイル1.
    :
END METHOD M-CLOSE.
```

ファイル1



初期化スレッド([1]～[3])は、実行環境で最初に1回だけ起動されます。

[1] クラス'X'のオブジェクトインスタンスを獲得します。

[2] メソッド'M-OPEN'を呼び出し、ファイルを開きます。

[3] PROPERTY句を指定したファクトリデータ'FOBJ'に、オブジェクトインスタンスを代入します。

レコード読込みスレッド([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。

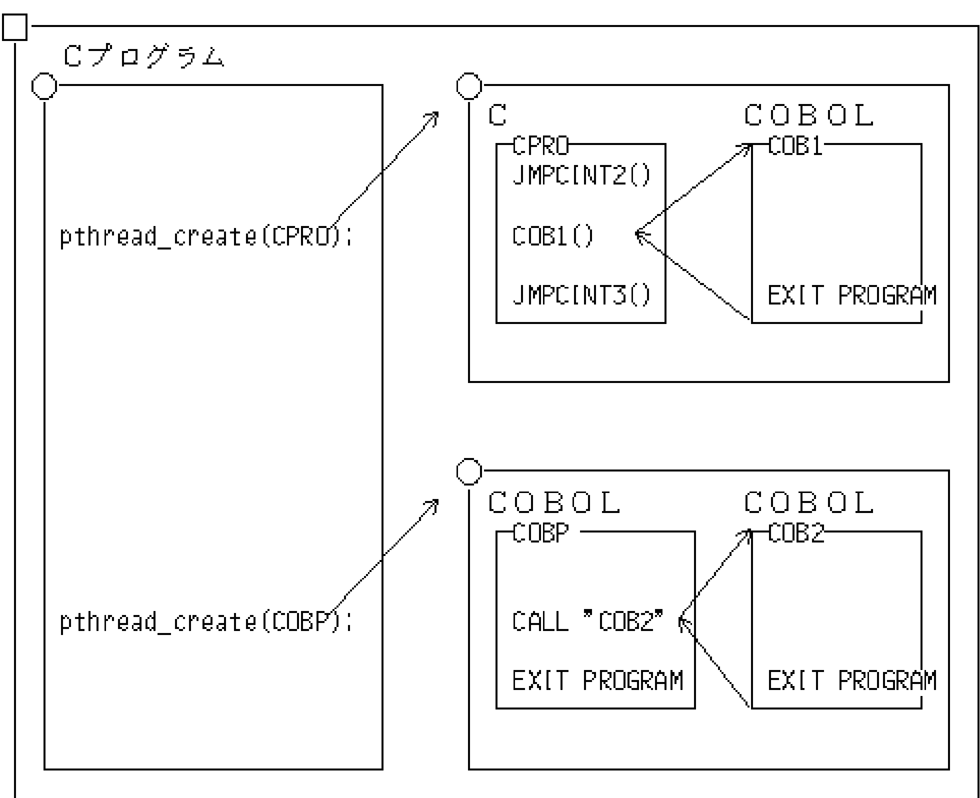
[9] メソッド'M-CLOSE'を呼び出し、初期化スレッドで開いたファイルを閉じます。オブジェクトロックサブルーチンについては、“17.9.2 オブジェクトロックサブルーチン”を参照してください。

17.6.2 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。なお、スレッドの起動方法の例として、POSIXスレッドを使用しています。POSIXスレッドやその他のスレッドの詳細については、C言語のマニュアルを参照してください。

17.6.2.1 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、システム関数を使用します。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。



17.6.2.2 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、システム関数のpthread_create()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、システム関数のpthread_join()を使用します。

17.6.2.3 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COBOLプログラムへ引数を渡す場合には、システム関数のpthread_create()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ある必要があります。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。

17.6.2.4 復帰コード（関数値）

手続き部の見出しのRETURNING指定の項目または特殊レジスタPROGRAM-STATUSに設定した値は、システム関数のpthread_join()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])と対応させる必要があります。データ型の対応については、"9.3.3 データ型の対応"を参照してください。

・ 関数 C

```
extern long int *COB(void *arg); …[1]

                        :
main(  ～  )
 {
```

```
/*  スレッドID                                  */
  pthread_t  cobtid;
/*  COBOL プログラムに渡すパラメタ                 */
  int    cobprm;
/*  復帰コード                                  */
  int    cobrcd;                 …[2]
  int    ret;
                 :
/*  COBOL プログラム(COB)をスレッドとして起動      */
  ret = pthread_create(&cobtid,
                       0,
                       COB,
                       &cobprm);
                 :
 pthread_join(cobtid, (void **)&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
```

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。

17.6.2.5 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 <errno.h>
#include <pthread.h>
              :

/*  スレッドとして起動するCOBOLプログラムを宣言する。      */
extern void *COBTHD1(void *arg);
extern void *COBTHD2(void *arg);

main()
{
/*  データ宣言                                        */
int        cobrcd1;
int        cobrcd2;
              :

/*  パラメタに1を設定してCOBTHD1を起動する                 */
cobprm1 = 1;
pthread_create (&cobtid1, NULL, COBTHD1, &cobprm1);
```

```
/*  COBTHD1が終了するのを待つ                             */
pthread_join(cobtid1, (void **)&cobrcd1);

/*  パラメタに2を設定してCOBTHD2を起動する                 */
cobprm2 = 2;
pthread_create (&cobtid2, NULL, COBTHD2, &cobprm2);
               :
               :

/*  COBTHD2が終了するのを待つ                             */
pthread_join(cobtid2, (void **)&cobrcd2);
               :
               :
}
```

・ 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
              :
              :
     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
              :
              :
     EXIT PROGRAM.
```

17.6.2.5.1 翻訳

・ Cプログラムを翻訳する

```
$ cc -c -D_POSIX_C_SOURCE=199506L CPROG.c
```

・ COBOLプログラムCOBTHD1を翻訳する

```
$ cobol -c -Tm COBTHD1.cob
```

・ COBOLプログラムCOBTHD2を翻訳と同時にリンクも実施する

```
$ cobol -dy -G -Tm -o libCOBTHD2.so COBTHD2.cob
```

17.6.2.5.2 リンク

- COBOLプログラムCOBTHD1をリンクする

```
$ cobol -dy -G -Tm -o libCOBTHD1.so COBTHD1.o
```

COBTHD1.o : COBOLプログラムCOBTHD1のオブジェクト

- Cプログラムをリンクする

```
$ cobol -Ns -Tm -o CPROG.exe -lCOBTHD1 -lCOBTHD2 CPROG.o
```

CPROG.o : CプログラムCPROGのオブジェクト

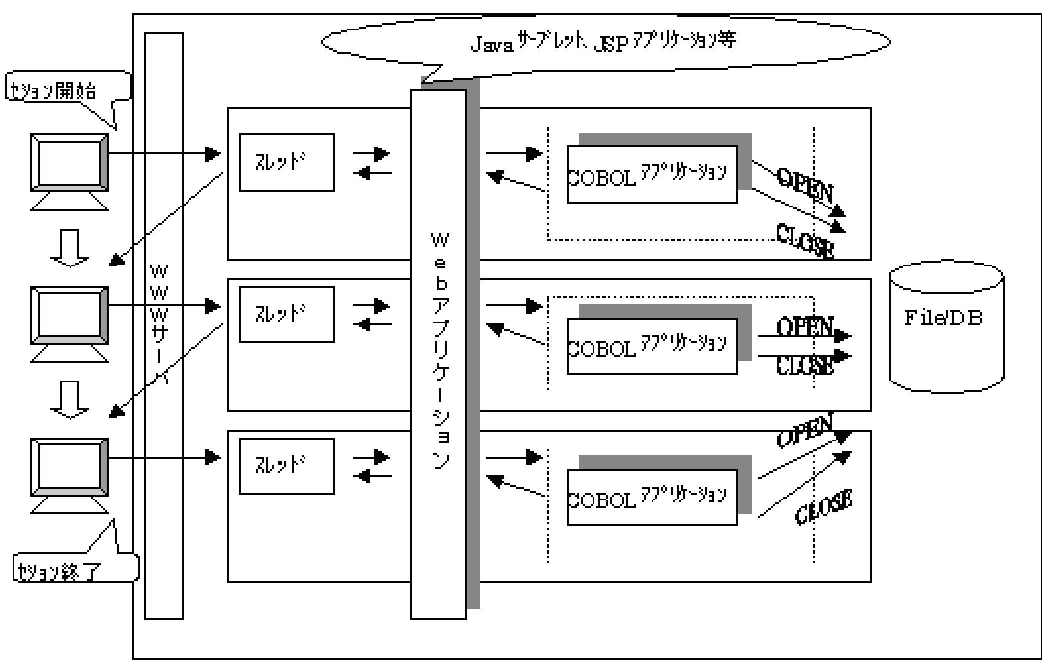
17.6.3 スレッド間で実行単位の資源を引き継ぐ方法

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。

17.6.3.1 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出しごとにCOBOLの実行単位に管理される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。

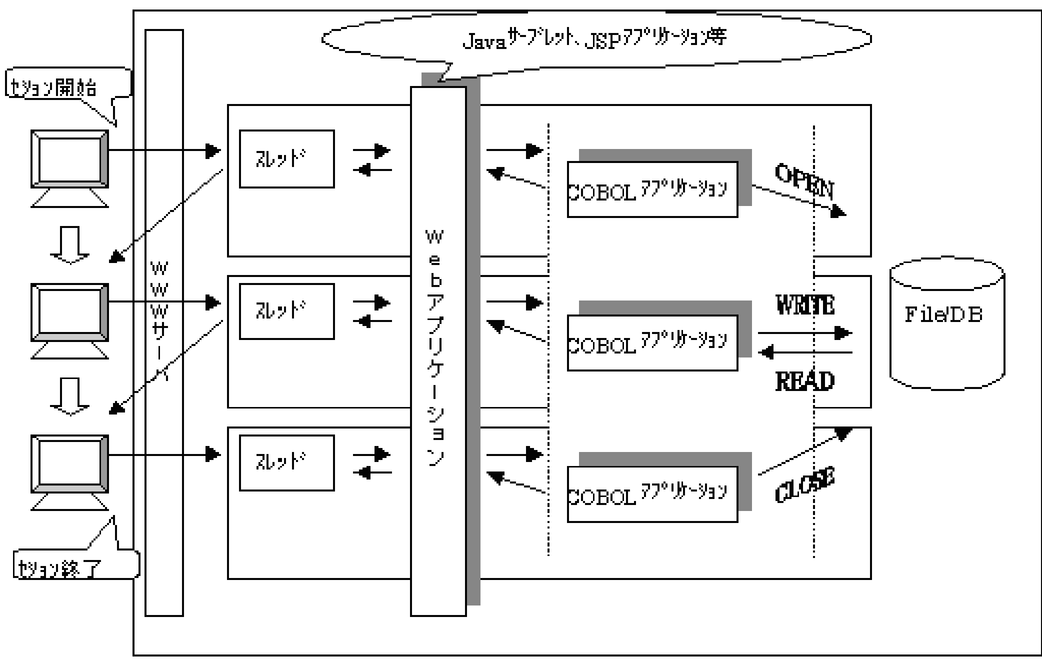
［実行単位を引き継がない場合］



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。

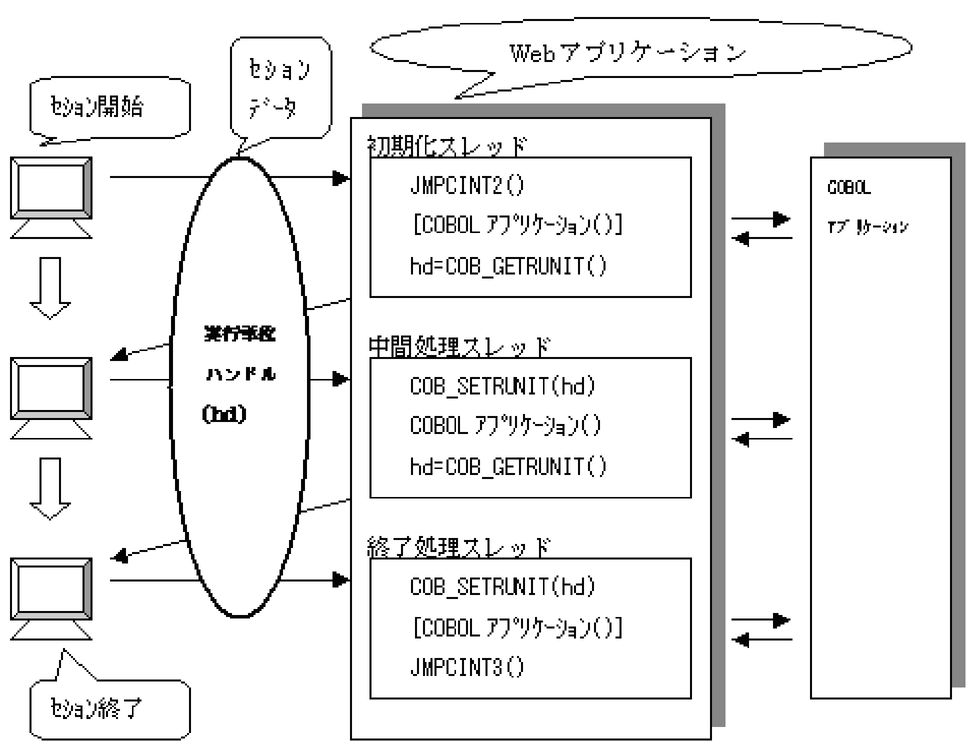
このような場合に、COBOLの実行単位のハンドルを返却するサブルーチンと、その実行単位のハンドルをCOBOLランタイムシステムに通知するサブルーチンを使用して、スレッド間で実行単位を引き継ぐことができます。引き継ぐ実行単位のハンドルとセションの関連付けは、Cookieなどを用いた管理機構によって、COBOLアプリケーションの呼出し側で行う必要があります。

[実行単位を引き継ぐ場合]



17.6.3.2 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。

参考

JavaプログラムからCOBOLプログラムの呼出しは、J Bussiness Kit(以降、JBKといいます)を使用して簡単に実現できます。JBKのラッピング機能では、JavaのオブジェクトがCOBOLプログラムの実行単位を引き継ぎます。このため、同じJavaのオブジェクトから呼び出されるCOBOLプログラムの間では、実行単位の資源を引き継いで利用することができます。

JBKは、Interstageに同梱されているコンポーネントです。

17.6.3.3 サブルーチンの使い方

17.6.3.3.1 COBOL実行単位ハンドル取得サブルーチン

COBOLの実行単位を識別するハンドルを取得する場合には、COB_GETRUNITサブルーチンを使用します。

初期化スレッドの処理の最初に、JMPCINT2を呼び出して実行単位を開始する必要があります。

指定方法

呼出しの記述(C言語での呼出し方)

```
型宣言部:
 extern unsigned long COB_GETRUNIT(void);

データ宣言部:
 unsigned long hd;  /* COBOLの実行単位ハンドル格納域 */

手続き部:
 hd = COB_GETRUNIT();
```

インタフェース

パラメタ

なし

復帰コード

正常時: COBOL実行単位のハンドル

異常時: 0

17.6.3.3.2 COBOL実行単位ハンドル設定サブルーチン

COBOLの実行単位を識別するハンドルを呼出し元のスレッドに設定する場合には、COB_SETRUNITサブルーチンを使用します。

終了スレッドの処理の最後に、JMPCINT3を呼び出して実行単位を終了する必要があります。

指定方法

呼出しの記述(C言語での呼出し方)

```
型宣言部:
 extern int COB_SETRUNIT(unsigned long hd);

データ宣言部:
 extern unsigned long hd;
             /* COBOLの実行単位ハンドル格納域(COB_GETRUNITで取得) */

手続き部:
 COB_SETRUNIT(hd);
```

インタフェース

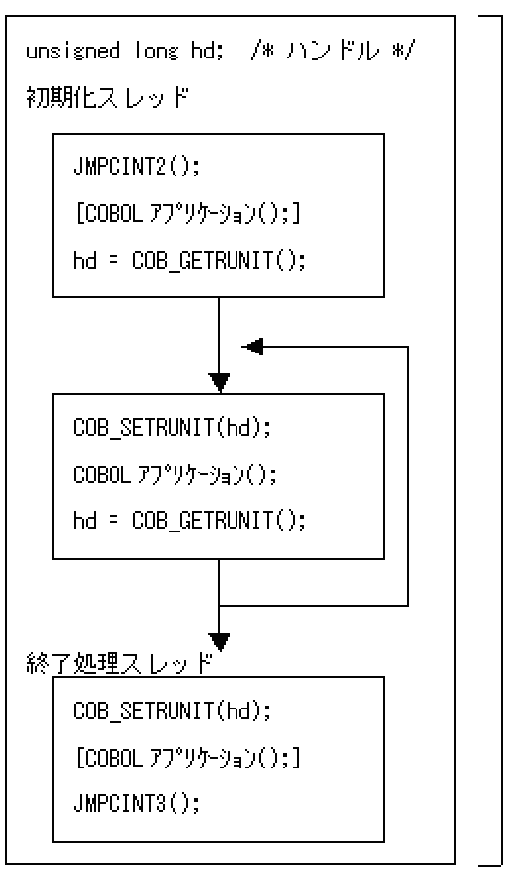
パラメタ

hd: COBOL実行単位のハンドル

復帰コード

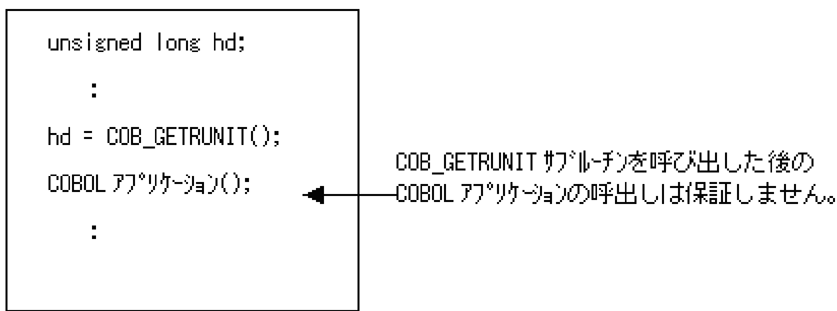
正常時: 0

異常時:
-1 入力パラメタが正しくない
-2 既にCOBOLの実行単位が存在する
-99 その他のエラー(SEに連絡してください)



17.6.3.4 注意事項

- タイムアウトなどのCOBOLアプリケーションが動作しているプロセス自身が終了しないような異常終了が発生した場合は、COB_SETRUNITサブルーチンで実行単位のハンドル設定後、JMPCINT3を呼び出して実行単位の終了を行う必要があります。JMPCINT3の呼出しを行わない場合、実行単位の資源が解放されずに残るため、メモリリークの原因となります。
- 複数のスレッドが同時に1つのCOBOLの実行単位を共有することはできません。必ず個々のCOBOLの実行単位が単一のスレッドだけで使用されるようにしてください。
- COB_GETRUNITサブルーチンを呼び出すと、呼び出したスレッドのCOBOLの実行単位をクリアします。このため、COB_GETRUNITサブルーチンを呼び出した後に、同一スレッドからCOBOLアプリケーションを呼び出すことはできません。



- COB_SETRUNITサブルーチンを呼び出すと、呼び出したスレッドのCOBOLの実行単位を上書きします。このため、COB_SETRUNITサブルーチンを呼び出す前に、そのスレッドからCOBOLアプリケーションを呼び出すことはできません。

```
int   rtncode;
extern unsigned long hd;
    :
COBOLｱﾌﾟﾘｹｰｼｮﾝ();
rtncode = COB_SETRUNIT(hd);
if (rtncode == -2) {
  ｴﾗｰ処理
}
    :
```

COB_SETRUNITｻﾌﾞﾙｰﾁﾝを呼び出す前のCOBOLｱﾌﾟﾘｹｰｼｮﾝの呼出しはできません。この状態でCOB_SETRUNITｻﾌﾞﾙｰﾁﾝを呼び出すと復帰ｺｰﾄﾞ-2でｴﾗｰになります。

- COUNT情報中に出力されるプロセスID(PID=)およびスレッドID(TID=)は、COUNT機能が情報を出力するときのプロセスIDおよびスレッドIDです。COUNT情報の出力時期の詳細については、“5.4.2 COUNT情報”を参照してください。
- 当サブルーチンを利用する際には、以下の制限があります。
  - スクリーン操作機能
    スクリーン操作機能は使用できません。
  - 表示ファイル
    表示ファイルを使用した機能では、スレッド間でファイル結合子を引き継ぐことができません。OPEN文からCLOSE文までを同一スレッドで行うようにしてください。
  - FORMAT句付き印刷ファイル
    FORMAT句付き印刷ファイルは、スレッド間でファイル結合子を引き継ぐことができません。OPEN文からCLOSE文までを同一スレッドで行うようにしてください。
  - 簡易アプリ間通信機能
    簡易アプリ間通信機能は、スレッド間でサーバ識別子を引き継ぐことができません。COBCI_OPENサブルーチンの呼出しからCOBCI_CLOSEサブルーチンの呼出しまでを同一スレッドで行うようにしてください。
  - RDMファイル
    RDMファイルは、スレッド間でファイル結合子を引き継ぐことができません。OPEN文からCLOSE文までを同一スレッドで行うようにしてください。

17.7 翻訳から実行までの方法

ここでは、マルチスレッドモデルのプログラムの翻訳から実行までの手順を説明します。

17.7.1 翻訳とリンク

ここでは、マルチスレッドモデルのプログラムの翻訳とリンクの方法について説明します。

マルチスレッドモデルのプログラムの翻訳とリンクの手順で、プロセスモデルのプログラムと異なるのは、-Tmオプション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、このほかに翻訳オプションSHREXTが必要となります。“表17.1 マルチスレッドモデルとプロセスモデルの翻訳とリンクの手順の違い”に、マルチスレッドモデルとプロセスモデルの翻訳とリンクの手順の違いを示します。

表17.1 マルチスレッドモデルとプロセスモデルの翻訳とリンクの手順の違い

	スレッド間共有の外部データまたは外部ファイルの使用	cobolコマンドによる翻訳時の指定	cobolコマンドによるリンク時の指定
マルチスレッドモデル	使用しない	-Tmオプション または	-Tmオプション

	スレッド間共有の外部データまたは外部ファイルの使用	cobolコマンドによる翻訳時の指定	cobolコマンドによるリンク時の指定
		翻訳オプション THREAD(MULTI)	
	使用する	-Tmおよび翻訳オプション SHREXT または 翻訳オプション THREAD(MULTI)および SHREXT	-Tmオプション
プロセスモデル	―	翻訳オプション THREAD(SINGLE) または 指定なし	指定なし

cobolコマンドに-Tmオプションを指定すると、プロセスモデルのCOBOLランタイムシステムの代わりにマルチスレッドモデルのCOBOLランタイムシステムが自動的にリンクされます。

翻訳およびリンク方法でプロセスモデルと共通の内容については、“第3章 プログラムの翻訳とリンク”を参照してください。

注意

マルチスレッドモデルを作成する際の翻訳オプションは、-TmのほかにTHREAD(MULTI)も有効です。しかし、リンク処理でマルチスレッドモデル用のCOBOLランタイムシステムが自動的にリンクされる-Tmの利用をおすすめします。
以下のようなプログラムを実行した場合、意図したとおりに動作しない場合があります。

- 翻訳オプションTHREAD(MULTI)を指定して翻訳したオブジェクトファイルを、cobolコマンドの-Tmオプションを指定しないでリンクしたプログラム
- 翻訳オプションTHREAD(SINGLE)を指定して翻訳したオブジェクトファイルを、cobolコマンドの-Tmオプションを指定してリンクしたプログラム

このようなリンクのミスを事前に防ぐためにリンクチェックコマンド(pmgr_chklnk)を用意しています。プログラムのリンク前のチェックやリンク後のチェックに使用してください。[参照]“17.7.3.2 プログラムのリンクチェック”

17.7.1.1 COBOLプログラムだけで共用オブジェクトプログラムを作成する場合

COBOLプログラムAでlibCOB.soを作成します。

```
COBOL プログラムA
01 DATE01 ～ EXTERNAL.
```

```
$ cobol -dy -G -Tm -o libCOB.so A.cob
```

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、翻訳オプションSHREXTを指定してください。

17.7.1.2 COBOLプログラムとCプログラムで共用オブジェクトプログラムを作成する場合

CプログラムCPROとCOBOLプログラムCOBでlibCCOB.soを作成します。

```
C プログラムCPRO
```

```
COBOL プログラムCOB
01 DATE01 ～ EXTERNAL.
```

- Cプログラム CPROを翻訳する

```
$ cc -c -D_POSIX_C_SOURCE=199506L CPRO.c
```

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。

- COBOLプログラムCOBを翻訳し、CプログラムCPROとリンクして、libCCOB.soを作成する

```
$ cobol -dy -G -Tm -o libCCOB.so -WC,SHREXT CPRO.o COB.cob
```

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、翻訳オプションSHREXTを指定してください。

17.7.2 実行

ここでは、マルチスレッドモデルのプログラムの実行の手順について説明します。なお、プロセスモデルのプログラムと重複する説明については、"第4章 プログラムの実行"を参照してください。

17.7.2.1 実行用の初期化ファイル

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。つまり、別々のスレッドで動作するマルチスレッドモデルのプログラムで、実行用の初期化ファイルは共有します。したがって、実行用の初期化ファイルの内容は、同じ実行環境で動作するプログラムの実行前に設定してください。

17.7.2.2 実行環境変数の設定

ここでは、実行環境変数の設定の手順について説明します。

17.7.2.2.1 実行環境変数の指定形式

ここでは、マルチスレッドモデルのプログラムにだけ有効な環境変数を説明します。

CBR_SYMFOWARE_THREAD(Symfoware連携でマルチスレッド動作可能にする指定)

```
CBR_SYMFOWARE_THREAD=MULTI
```

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]"17.5.8.1 Symfoware連携"

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]"17.9 スレッド同期制御サブルーチン"

17.7.3 マルチスレッドモデルとプロセスモデルの混在チェック

COBOLプログラムの作成時や実行時にマルチスレッドモデルとプロセスモデルとを混在させると、誤動作の原因となります。

ここでは、マルチスレッドモデルのプログラムとプロセスモデルのプログラムを混在して実行した場合の実行時チェックやCOBOLプログラムの作成時に必要なリンクチェックについて説明します。

17.7.3.1 実行時チェック

1つのプロセス上で、マルチスレッドモデルのプログラムとプロセスモデルのプログラムが混在して実行すると2つの実行環境が存在してしまい誤動作します。

COBOLランタイムシステムでは、マルチスレッドモデルのプログラムとプロセスモデルのプログラムを混在して実行した場合に、これを検出し実行時にエラーとします。

17.7.3.2 プログラムのリンクチェック

以下に示すような場合に、COBOLプログラムを実行すると、実行時エラーとなる場合、または意図したとおりに動作しない場合があります。

- マルチスレッドモデルのプログラムとプロセスモデルのプログラムを混在して実行した場合
- マルチスレッドモデル用のオブジェクトファイルとプロセスモデル用のオブジェクトファイルを混在してリンクした場合
- マルチスレッドモデル用に翻訳したオブジェクトファイルに対してプロセスモデルのプログラムをリンクする方法でプログラムを作成した場合
- プロセスモデル用に翻訳したオブジェクトファイルに対してマルチスレッドモデルのプログラムをリンクする方法でプログラムを作成した場合

このようなミスを事前に防ぐため、COBOLではリンクチェックコマンド(pmgr_chklnk)を用意しています。作成したオブジェクトファイルやプログラムがプロセスモデルか、マルチスレッドモデルか、またそれらが正しく作成されているかをこのコマンドで確認することができます。このコマンドは、Makefile中でcobolコマンドなどと組み合わせて利用すると効果的です。

pmgr_chklnkコマンドの操作方法は、manマニュアルを参照してください。

17.8 マルチスレッドモデルのプログラムのデバッグ方法

ここではマルチスレッドモデルのプログラムをデバッグする方法について説明します。

17.8.1 マルチスレッドモデルのデバッグ

マルチスレッドモデルのプログラムを実行して発生する問題には、次の2つがあります。

- プロセスモデルで実行した場合に発生する問題
- 複数のプログラムを同時に実行した場合に発生する問題

プロセスモデルで実行した場合に発生する問題は、マルチスレッドモデルでも再現します。プロセスモデルで発生する問題については従来のデバッグ方法でデバッグします。

複数のプログラムを同時に実行した場合に発生する問題は、再現性がないことが多くあります。これはスレッドの実行順序が定まっていないためです。複数のプログラムを同時に実行した場合に発生する問題は、一般的に統計的な現象傾向を示します。このため、実行レベルの問題を検出する場合には、ブレークポイントを設定するデバッグよりもトレース情報を用いたデバッグが有効です。

このように、プロセスモデルのデバッグと複数のプログラムを同時に実行した場合のデバッグでは、デバッグ方法に違いがあります。このため、マルチスレッドモデルのプログラムをデバッグする場合、プロセスモデルで実行した場合と複数のプログラムを同時に実行した場合の問題を分類することが重要です。マルチスレッドモデルのプログラムを実行して発生した問題は、次の方法で問題が再現することを確認することにより、どちらの問題であるかを明らかにすることができます。

- マルチスレッドモデルのプログラムをプロセス内で1つだけ実行する
- プロセスモデルに変更し、プログラムを実行する

また、上記の方法によって発生条件を絞り込むことができるため、問題の早期解決につながります。

17.8.2 マルチスレッドモデルのデバッグ機能

COBOLが提供する以下の機能は、マルチスレッドモデルのプログラムで用いても、基本的にデバッグ方法に変更はありません。

- TRACE機能
- CHECK機能
- COUNT機能
- 対話型デバッガ

これらの機能は、プロセスモデルのプログラムをデバッグする機能と同様に使用することができます。

ここでは、マルチスレッドモデルのプログラムをデバッグする場合に留意する事項について説明します。

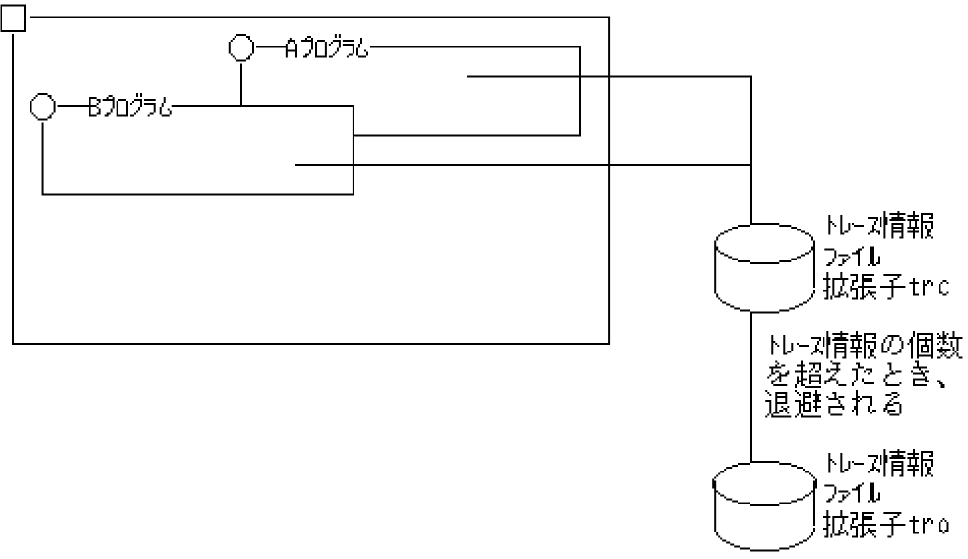
なお、デバッグ機能の基本的な使い方については"第5章 プログラムのデバッグ"を参照してください。

17.8.2.1 TRACE機能

トレース情報の内容に変更はありません。[参照]“5.2 TRACE機能の使い方”

ただし、各スレッドから採取されたトレース情報は、1つのファイル(拡張子trc)に格納されます。

以下に図で示します。



トレース情報の見方を、以下の図で説明します。

```
NetCOBOL DEBUG INFORMATION            DATE 2000-03-31  TIME 14:50:54  PID=00003488

TRACE INFORMATION

[1]       1  A           DATE 2000-03-31  TIME 14:50:47  TID=00000004
          2               7.1 TID=00000004
          3               8.1 TID=00000004
          4              11.1 TID=00000004
[2]       5     EXIT-THREAD TID=00000004
[1]       6  B           DATE 2000-03-31  TIME 14:50:48  TID=00000005
          7               9.1 TID=00000005
          8              10.1 TID=00000005
          9              13.1 TID=00000005
[3]'     10              14.1 TID=00000005
[1]      11  C           DATE 2000-03-31  TIME 14:50:49  TID=00000006
         12               7.1 TID=00000006
[3]      13    JMP0015I-U [PID:00003488 TID:00000005] プログラム'D'を呼び
出すのに失敗しました. ld.so.1: PROG: 重大なエラー: D: シンボルを見つける
ことができません。  PGM=B. LINE=14
```

[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00000004です。
プログラムBのスレッドIDは00000005です。
プログラムCのスレッドIDは00000006です。

[2] スレッド終了通知メッセージ

スレッドが終了したのは、スレッドIDが00000004のプログラムAです。

上記の結果から、以下のことがわかります。

- スレッドIDが00000004のプログラムAは正常に動作した。
- スレッドIDが00000005のプログラムBは14行目の実行文で実行時エラーが発生した。
- スレッドIDが00000006のプログラムCは強制終了した。

[3]実行時メッセージ

実行時エラーが発生したのは、スレッドIDが00000005のプログラムBです。なお[3]'より、プログラムBが最後に実行したのは14行目ということがわかります。

参考

DISPLAY文のUPON指定にSYSERRを指定すると、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。

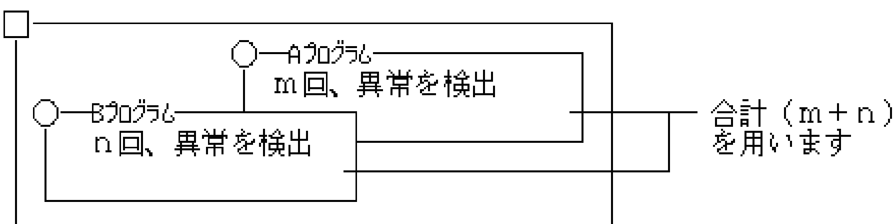
注意

一度に多くのスレッドを実行する場合、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GOPTのr指定)を調整してください。

17.8.2.2 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]“5.3 CHECK機能の使い方”

ただし、出力メッセージの回数は、プロセス内で検出された回数の合計を用います。以下の図に示します。



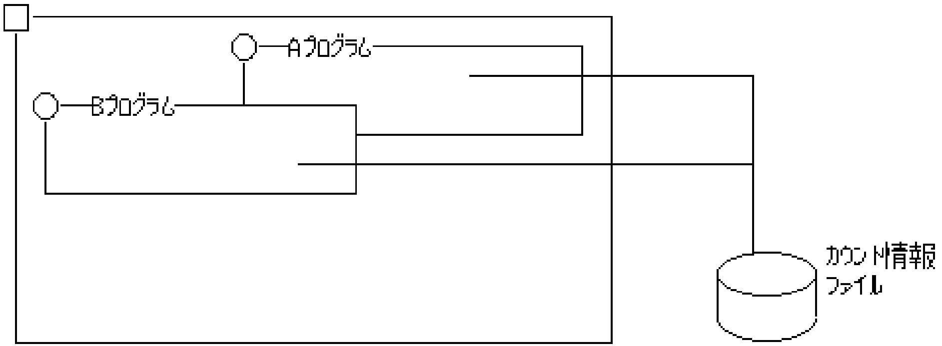
17.8.2.3 COUNT機能

カウント情報の内容に変更はありません。[参照]“5.4 COUNT機能の使い方”

ただし、以下のような動作となります。

- 各スレッドから採取されたカウント情報は、1つのファイルに格納されます。
- カウント情報が書き込まれる集計結果は、スレッド単位に出力します。プロセス全体で集計されません。

以下に図で示します。



出力されるカウント情報の見方を、以下の図で説明します。

```
NetCOBOL  COUNT INFORMATION (END OF RUN UNIT)        DATE 2000-03-31  TIME 13:19:43
PID=0000063C  TID=00000004 [1]
  STATEMENT EXECUTION COUNT    PROGRAM-NAME : A

   :

NetCOBOL  COUNT INFORMATION (END OF RUN UNIT)        DATE 2000-03-31  TIME 13:19:43
PID=0000063C  TID=00000005 [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00000004です。
プログラムBのスレッドIDは00000005です。

17.8.2.4 対話型デバッガ

プロセスモデルのプログラムを実行した場合、デバッガの操作方法に変更はありません。詳細については、"第23章 対話型デバッガの使い方"を参照してください。

マルチスレッドモデルのデバッグ時には、複数のCOBOLプログラムが同時に実行しないように、自動的に同期制御を行います。このため、プログラムが多重で動作している場合でも、プロセスモデルに変更することなく、プロセスモデルと同様のデバッグを行うことができます。

複数のプログラムを同時に実行した場合に発生する問題は、対話型デバッガを使用すると再現しない場合があります。これは、スレッドの実行順序が変更されるために発生します。このような問題についてはトレース情報(COBOLのトレース情報やシステムコマンドのtrussコマンドが出力する情報など)を使用し、統計的な現象傾向を調査する方法と併用してデバッグしてください。

17.8.2.5 障害発生箇所の特定方法

実行時に、COBOLランタイムシステムで致命的なエラーが検出され、異常終了したときの障害発生箇所と、実行時に異常終了したときの障害原因の特定方法は、"5.6 異常終了時の障害発生箇所の特定方法"を参照してください。

17.9 スレッド同期制御サブルーチン

ここでは、スレッドの同期制御を行うためのサブルーチンについて説明します。スレッド同期制御サブルーチンには、データロックサブルーチンとオブジェクトロックサブルーチンがあります。

注意

動的プログラム構造で、スレッド同期制御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ファイルが必要になります。エントリ情報の指定方法については、"4.1.4 副プログラムのエントリ情報"を参照してください。

```
[ENTRY]
サブルーチン名=librcobol.so
```

17.9.1 データロックサブルーチン

サブルーチン名	機能
COB_LOCK_DATA	ロックキーに対するロックの獲得
COB_UNLOCK_DATA	ロックキーに対するロックの解放

同一プロセス内のスレッド間で同期制御が必要となる場合に、当サブルーチンを使用して、互いに同じデータ名のロックキーに対してロックの獲得と解放を行うことで相互排他ロックが可能となります。このときに指定するデータ名は、プロセス内で一意にする必要があります。

COB_LOCK_DATAの呼び出し時に、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すでに存在する場合は、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。

COB_LOCK_DATAを呼び出してからCOB_UNLOCK_DATAを呼び出すまでの手続きが同期制御の対象となります。

17.9.1.1 COB_LOCK_DATA

機能

指定されたデータ名に対応するロックキーに対してロックを獲得します。

呼出し形式

```
DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  LOCK-KEY        PIC X(30).
  01  WAIT-TIME       PIC S9(9) COMP-5.
  01  ERR-DETAIL      PIC 9(9)  COMP-5.
  01  RET-VALUE       PIC S9(9) COMP-5.

 PROCEDURE DIVISION.

     CALL  "COB_LOCK_DATA"  USING  BY REFERENCE LOCK-KEY
                                   BY VALUE WAIT-TIME
                                   BY REFERENCE ERR-DETAIL
                            RETURNING  RET-VALUE.
```

パラメタ

LOCK-KEY

ロックを獲得するロックキーの名前を30バイト以内で指定します。ロックキーの名前が30バイトに満たない場合は、末尾に空白が必要となります。

WAIT-TIME

ロックを獲得するまでの待ち時間(秒)を指定します。-1を指定すると無限待ちとなります。

無限待ちを指定した場合、環境変数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する場合などに利用します。

ERR-DETAIL

戻り値が-255のとき、システムエラーコードが返ります。

戻り値

RET-VALUE

成功時は、0が返ります。また、プロセスモデルのプログラムの場合は、ロックが不要であるため、ロックを獲得しないで、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、“17.9.3 エラーコード”を参照してください。

17.9.1.2 COB_UNLOCK_DATA

機能

指定されたデータ名に対応するロックキーに対してロックを解放します。

呼出し形式

```
DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  LOCK-KEY        PIC X(30).
  01  ERR-DETAIL      PIC 9(9)   COMP-5.
  01  RET-VALUE       PIC S9(9)  COMP-5.

 PROCEDURE DIVISION.

     CALL   "COB_UNLOCK_DATA"  USING  BY REFERENCE LOCK-KEY
                                      BY REFERENCE ERR-DETAIL
                               RETURNING  RET-VALUE.
```

パラメタ

LOCK-KEY

ロックを獲得するロックキーの名前を30バイト以内で指定します。ロックキーの名前が30バイトに満たない場合は、末尾に空白が必要となります。

ERR-DETAIL

戻り値が-255のとき、システムエラーコードが返ります。

戻り値

RET-VALUE

成功時は、0が返ります。また、プロセスモデルのプログラムの場合は、ロックが不要であるため、ロックを解放しないで、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、“17.9.3 エラーコード”を参照してください。

17.9.2 オブジェクトロックサブルーチン

サブルーチン名	機能
COB_LOCK_OBJECT	オブジェクトに対するロックの獲得
COB_UNLOCK_OBJECT	オブジェクトに対するロックの解放

同一プロセス内のスレッド間でオブジェクトを共有している場合に、当サブルーチンを使用して、同一のオブジェクトに対してロックの獲得と解放を行うことで、相互排他ロックが可能となります。

COB_LOCK_OBJECTの呼出し時に、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。

17.9.2.1 COB_LOCK_OBJECT

機能

指定されたオブジェクトに対してロックを獲得します。

呼出し形式

```
DATA DIVISION.
 WORKING-STORAGE SECTION.
  01  OBJ             OBJECT REFERENCE クラス名.
  01  WAIT-TIME       PIC S9(9) COMP-5.
  01  ERR-DETAIL      PIC 9(9)  COMP-5.
  01  RET-VALUE       PIC S9(9) COMP-5.

 PROCEDURE DIVISION.

     CALL   "COB_LOCK_OBJECT"  USING  BY REFERENCE OBJ
                                      BY VALUE WAIT-TIME
                                      BY REFERENCE ERR-DETAIL
                               RETURNING  RET-VALUE.
```

パラメタ

OBJ

ロックを獲得するオブジェクトのオブジェクト参照を指定します。

WAIT-TIME

ロックを獲得するまでの待ち時間(秒)を指定します。-1を指定すると無限待ちとなります。

無限待ちを指定した場合、環境変数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する場合などに利用します。

ERR-DETAIL

戻り値が-255のとき、システムエラーコードが返ります。

戻り値

RET-VALUE

成功時は、0が返ります。また、プロセスモデルのプログラムの場合は、ロックが不要であるため、ロックを獲得しないで、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、"17.9.3 エラーコード"を参照してください。

17.9.2.2 COB_UNLOCK_OBJECT

機能

指定されたオブジェクトに対してロックを解放します。

呼出し形式

```
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

ロックを解放するオブジェクトのオブジェクト参照を指定します。

ERR-DETAIL

戻り値が-255のとき、システムエラーコードが返ります。

戻り値

RET-VALUE

成功時は、0が返ります。また、プロセスモデルのプログラムの場合は、ロックが不要であるため、ロックを獲得しないで、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、“17.9.3 エラーコード”を参照してください。

17.9.3 エラーコード

ここでは、スレッド同期制御サブルーチンのエラー発生時の戻り値について説明します。
“表17.2 エラーコード一覧”の“対象サブルーチン”の記号の意味は以下のとおりです。

LD:COB_LOCK_DATA
UD:COB_UNLOCK_DATA
LO:COB_LOCK_OBJECT
UO:COB_UNLOCK_OBJECT

表17.2 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			
		LD	UD	LO	UO
-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。ロックキーの名前の指定誤り(LD、UD)、待ち時間の指定誤り(LD、LO)またはオブジェクト参照にNULLオブジェクトが指定されている(LO、UO)可能性があります。	○	○	○	○
-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にシステムエラーコードが設定されます。	○	○	○	○

○:通知される

第18章　Unicode

本章では、Unicodeで動作するCOBOLアプリケーションの作成方法について説明します。

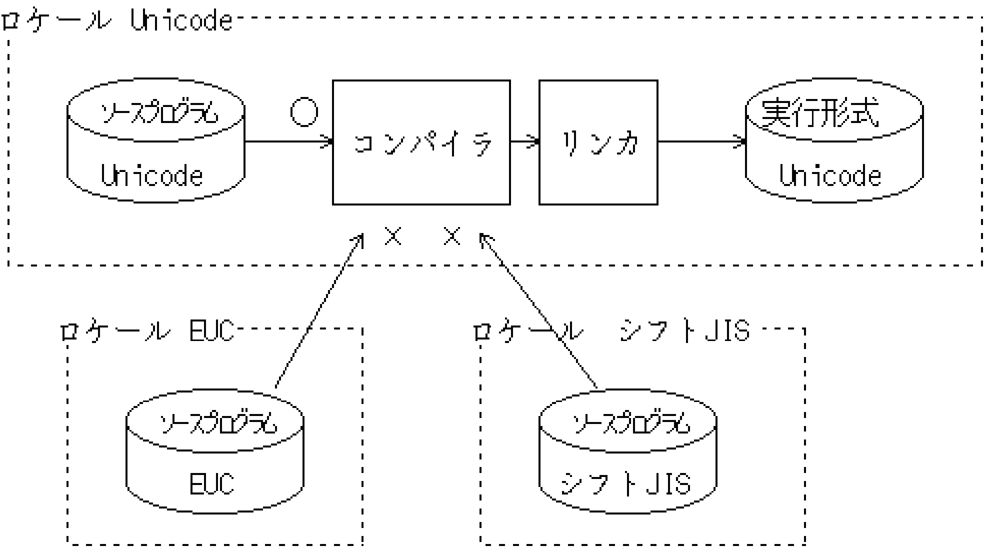
18.1 Unicode概要

COBOLでのUnicodeの実装について、まずは概要から説明します。

18.1.1 資源

COBOLはシステムのロケールに従ってコード系を決定します。

このため、Unicodeで動作するアプリケーションを作成する場合、ソースなどの翻訳資産は全てUnicode(UTF-8)で統一させる必要があります(ロケールについては“付録J 日本語コード系”を参照してください)。



18.1.2 表現形式

Windows Vistaを発端に、最近ではJIS X 0213:2004に対応した文字セット(以降、JIS2004と記述します)を利用する環境が整いつつあります。

このJIS2004で定義された文字を全て活用するにはUnicodeしか選択肢がありません。そして、Unicodeでは、一部の文字(300文字程度)が2バイトで表現できる範囲外(2面)にマップされたため、従来のUCS-2と呼ばれる表現形式ではこれらの文字が表現できなくなりました。そこで、UCS-2に代わってUTF-16と呼ばれる表現形式が利用されるようになっています。このUTF-16は、0面(いわゆるBMPと呼ばれる範囲)に配置された文字はUCS-2と互換性のある2バイトのコードで、2面に配置された文字はサロゲートペアと呼ばれる4バイトのコードで表現されます。

NetCOBOLでは、UTF-16表現で文字データを扱う場合、日本語項目を使用します。このとき、サロゲートペアで表現される文字は1文字につき2けたの宣言が必要となります。従来の、格納できる文字数＝けた数が成立しないため、注意が必要です。なお、エンディアンには、Solarisシステムで一般的に用いられるビッグエンディアンを採用しました。

英数字項目の表現形式は従来どおりUTF-8です。BMPの範囲であれば1文字を格納するために最大で3けた用意すればよかったのですが、前述したサロゲートペアの文字を英数字項目に格納する場合、1文字につき4けたの宣言が必要となるため、注意してください。

以下に字類と表現形式をまとめます。

項目のレベル	字類	表現形式
基本項目	英字	ASCII
	英数字	ASCII(UTF-8)

項目のレベル	字類	表現形式
	日本語	UTF-16
集団項目	英数字	ASCII(UTF-8)

18.1.3 言語要素

ここでは、Unicodeに関連した言語要素について解説します。

日本語文字定数

UTF-16では、ASCII文字(いわゆる半角の英数字)も1文字2バイトで表現されるため、日本語項目でのASCII文字のハンドリングが可能です。これに伴い、日本語文字定数中にASCII文字が記述できるようになりました。

```
WORKING-STORAGE SECTION.
01 ADDR  PIC N(40).
    :
  MOVE NC"沼津市宮本140番地 コーポ富士通 B-5" TO ADDR.
```

組込み関数

UTF-16とUTF-8を相互に変換する組込み関数を提供します。それぞれの場合に応じて、以下の組込み関数を使用してください。

- UTF-16のデータをUTF-8へ変換する場合はDISPLAY-OF関数を使用
- UTF-8のデータをUTF-16へ変換する場合はNATIONAL-OF関数を使用

```
WORKING-STORAGE SECTION.
01 PIC-X  PIC X(10).
01 PIC-N  PIC N(06) VALUE NC"12ABあ亜".
    :
  MOVE FUNCTION DISPLAY-OF(PIC-N)  TO PIC-X.
  MOVE FUNCTION NATIONAL-OF(PIC-X) TO PIC-N.
```

- 関数の結果の長さは、引数に指定された文字列の変換結果の長さになります。なお、引数の値が規定されたコードとして正しくない場合、結果は保証しません。
- NetCOBOL V9.1以前は、UTF8-OF関数およびUCS2-OF関数を提供していました。
  これらの関数は引き続き使用できますが、NetCOBOL V10以降は、UTF8-OF関数の代わりにDISPLAY-OF関数、UCS2-OF関数の代わりにNATIONAL-OF関数の使用を推奨します。

字類条件

Windows系システムが先行して、JIS2004を利用できる環境、つまり、Unicode3.2に対応した環境を整えつつあります。しかし、UNIX系システムをはじめとして、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。よって、それらと連携してシステムを構築する場合、互換をとれる範囲でデータを流通させる必要があります。

NetCOBOLでは、データ項目に格納されている文字がUnicode1.x範囲内であるか、容易に検査するための字類条件(UNICODE1)を用意しました。前述したシステムやミドルウェアと連携する場合、要所に検査を入れることでシステムの安定性が保持できます。

```
WORKING-STORAGE SECTION.
01 PIC-N  PIC N(10).
    :
  IF PIC-N IS NOT UNICODE1 THEN
    DISPLAY "JIS2004固有文字が含まれています。"
  END-IF.
```

また、日本語項目の表現形式に採用したUTF-16では、サロゲートペアの文字に2けたの領域を必要とします。このため、1文字につき1けたで設計された従来系システムは、厳密には正しく動作しないことになります。ただし、サロゲートペアに該当する文字は、4000文

字を超えるJIS2004追加文字のうち、わずか300文字程度であり、当該文字を利用できなくとも、システム再設計するまで踏み込まずに動作させたいという局面は多いはずです。

NetCOBOLでは、データ項目に格納されている文字がBMP範囲内かを容易に検査するための字類条件(BMP)を用意しました。要所に検査を入れることで、従来資産をJIS2004が実装されたシステム上で安定して運用することができます。

```
WORKING-STORAGE SECTION.
01 PIC-N  PIC N(10).
    :
   IF PIC-N IS NOT BMP THEN
     DISPLAY "サロゲートペアが含まれています。"
   END-IF.
```

この字類条件BMPは、格納された文字数をカウントする場合や、部分参照や転記の際に"文字の泣き別れ"が発生してないかを確認する場合にも利用できます。

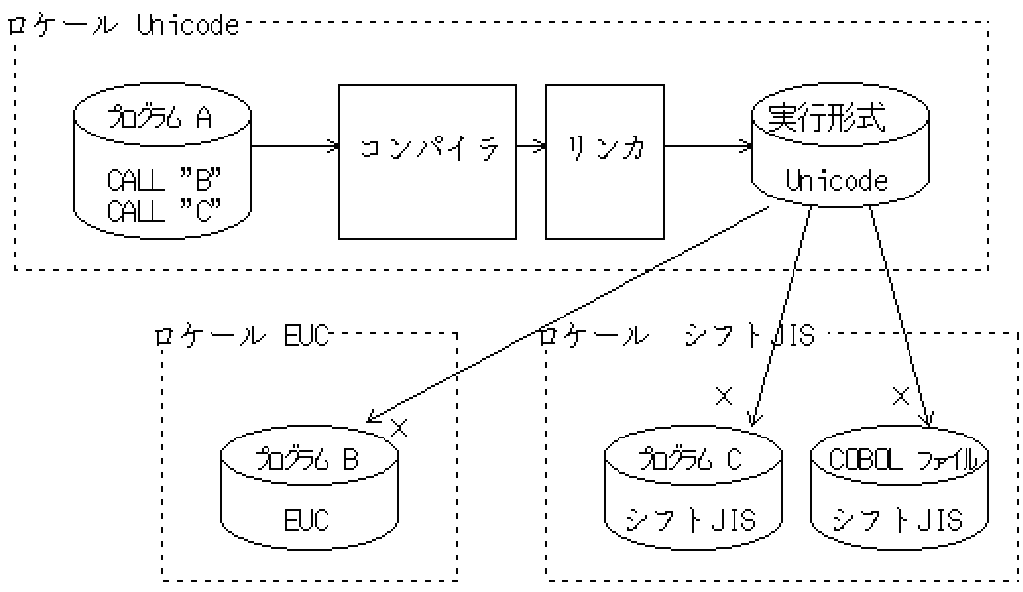
例

文字数をカウントする例

```
WORKING-STORAGE SECTION.
01 PIC-N   PIC N(10).
01 CNT     PIC 9(2).
01 CHAR-NO PIC S9(4) BINARY VALUE ZERO.
    :
PERFORM VARYING CNT FROM 1 BY 1 UNTIL CNT > 10
  IF PIC-N(CNT:1) IS NOT BMP THEN
     ADD 1 TO CNT
  END-IF
  ADD 1 TO CHAR-NO
END-PERFORM.
```

18.1.4 コード系の混在

COBOLでは実行単位内でコード系が混在することを許していません。翻訳時のロケール(コード系)と同じロケールで実行してください。以下の図のように、コード系の異なる共用オブジェクトプログラムを呼び出した場合、実行時エラーとなります。



ただし、日本語のデータを使用しないことを前提として、すべてのロケールで共通に利用できる実行形式を作成したい局面もあります。このような場合は、翻訳オプションNOCODECHKを指定することによって、実行時のコード系混在チェックを抑止することができます。

この場合、もし日本語のデータが流れたとしてもCOBOLランタイムシステムは一切チェックしません。すべては利用者責任となりますので、注意してください。

18.1.5 資産移行支援

EUCまたはシフトJISで運用していたアプリケーションをUnicodeへ移行する場合、同時にCOBOLファイルの移行が必要になることがあります。COBOLでは、関連製品であるSIMPLIA/TF-MDPORTを利用することによって、COBOLファイルのEUC／シフトJIS→Unicode変換を実現しました。詳細については、"SIMPLIA/TF-MDPORT オンラインマニュアル"を参照してください。

18.2 Unicodeアプリケーションの作成

ここでは、具体的にUnicodeアプリケーションを作成する手順を説明します。なお、実際にコーディングする際にはいくつか注意点があるので、"18.3 コーディング上の注意点"を一読の上、作業を開始してください。

18.2.1 プログラムの作成、編集

ソースプログラムや登録集はUnicode(UTF-8)で作成します。

Unicode(UTF-8)での編集が可能なエディタを使用してください。

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あり、最大長を超えた場合、コンパイラは超えた部分を無視しますので、日本語が含まれる行は注意して下さい。

翻訳時に下記のエラーが出力された場合、上記正書法の規則により行が途中で切れている可能性があります。

```
cobol:ERROR:システムエラー'errno=0x016'が'iconv_error'で発生しました。
```

この場合は、継続行を利用するなどして、行を分割してください。

18.2.2 翻訳、結合、実行

ロケールをUnicodeにしておくこと以外、とくに注意すべきことはありません。

EUC／シフトJISと同様に翻訳、結合、実行してください。

18.2.3 デバッグ

ロケールをUnicodeにすることで、対話型デバッガを利用してUnicodeアプリケーションのデバッグを行うことができます。基本的にはEUCやシフトJISの場合と同じ要領でデバッグできますが、スクリーンモードは使用できず、ラインモード、または、リモートデバッガを使用します。デバッガの使い方は"第23章 対話型デバッガの使い方"を参照してください。

18.3 コーディング上の注意点

ここでは、言語要素ごとにUnicode利用時の注意点やポイントを説明します。

なお、ここで挙げた修正が必要なコーディングの多くは、特定のコード系のみで通用するコーディングです。つまり、もともと可搬性(移植性)の高いCOBOLアプリケーションを作成するためにはおすすめできないコーディングであることを補足しておきます。

18.3.1 半角カナについて

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。

18.3.2 文字定数

表意定数

表意定数は作用対象によって決まります。

たとえば、表意定数SPACEは、作用対象が英数字項目の場合はASCII空白(半角空白)となり、日本語項目の場合は全角空白になります。

18.3.3 項目の再定義

日本語項目を英数字項目で再定義(REDEFINES句)している、またはその逆の場合、注意が必要です。

```
WORKING-STORAGE SECTION.
01 PERSON.
  02 AGE          PIC 9(3).
  02 NAME         PIC N(8).
  02 NAME-X       REDEFINES NAME PIC X(16).
01 TMP-NAME     PIC X(16).
     :
    MOVE NAME-X TO TMP-NAME.
    DISPLAY TMP-NAME.          ←文字化け
```

Unicodeでは、日本語項目(UTF-16)と英数字項目(UTF-8)の表現形式は全く異なります。このため、再定義によって同一のデータを別の字類で参照する場合、作用対象に合わせたデータ変換が必要になることがあります。データ変換には、DISPAY-OF関数およびNATIONAL-OF関数を利用してください。

```
01 TMP-NAME     PIC X(24).
     :
   MOVE FUNCTION DISPLAY-OF(NAME) TO TMP-NAME.
   DISPLAY TMP-NAME.
```

18.3.4 転記

集団項目転記

日本語項目を含む集団項目を転記に使用する場合、注意が必要です。

```
WORKING-STORAGE SECTION.
01 PERSON.
  02 AGE          PIC 9(3).
  02 NAME         PIC N(8).
01 TMP-AREA     PIC X(80).
     :
    MOVE PERSON TO TMP-AREA.
      :
    MOVE TMP-AREA TO PERSON.      …[1]
    DISPLAY "DATA = " TMP-AREA.   …[2]
```

上記の例で、TMP-AREAを一時的な作業域として使う[1]のであれば特に問題ありませんが、直接データを参照する[2]場合、表現形式の異なるデータが混在していることから、正しく表示されません。このような場合、一時域を元(PERSON)のデータ構造に合わせてください。

空白づめ

COBOLでは、文字転記の際、受取り側項目が送出し側項目よりも大きい場合、受取り側項目の字類に合わせて空白づめを行います。Unicodeでは、日本語転記の際の空白づめには全角空白が使用されます。

```
WORKING-STORAGE SECTION.
01 PIC-N        PIC N(10) VALUE NC"Fujitsu".    …[1]
01 PIC-X        PIC X(10).
     :
    MOVE FUNCTION DISPLAY-OF(PIC-N) TO PIC-X.   …[2]
    IF PIC-X = "Fujitsu" THEN DISPLAY "OK!!".   …[3]
```

上記の例で、[3]の比較条件は真が成立するように見えますが、実際には偽が成立します。PIC-Nには3文字の全角空白がつめられる[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。

文字の泣き別れ

日本語項目にサロゲートペアの文字を格納する際は、2けたの領域が必要となります。これは、表示などの際は1文字に見えますが、言語仕様上は2文字の扱いとなるため、転記や比較、部分参照などの局面で1文字が分割され、不正な文字データとなる可能性があることを意味します。このように、1文字が分割された状態を文字の泣き別れと呼びます。

このような不正データを作らないために、以下の注意が必要です。

- 十分な領域を用意する
- “18.1.3 言語要素”で説明した字類条件を利用してサロゲートペアを除外する、または、意識して文字列操作する

なお、英数字項目については、Unicodeに限らず、同様の注意が必要です。

18.3.5 比較

集団項目比較

集団項目比較の場合、事実上、字類の異なる項目どうしの比較が可能になります。Unicodeでは表現形式が異なるため、注意が必要です。

```
WORKING-STORAGE SECTION.
01 DATA-X.
  02 NAME  PIC X(6) VALUE "日本".    *> X"E697A5E69CAC"
01 DATA-N.
  02 NAME  PIC N(2) VALUE NC"日本". *> X"65E5672C"
      :
    IF DATA-X = DATA-N THEN DISPLAY "OK??".
```

集団項目比較を行う場合、作用対象のデータ構造(宣言)は同じにしてください。

18.3.6 ACCEPT/DISPLAY文

小入出力

UnicodeのデータをACCEPT文、DISPLAY文を使用して入出力できますが、作用対象が日本語を含む集団項目の場合に注意が必要です。

```
WORKING-STORAGE SECTION.
01 PERSONAL-DATA.
  02 NAME   PIC N(8).
  02 TEL    PIC 9(10).
      :
    DISPLAY PERSONAL-DATA.  …[1]
    DISPLAY NAME TEL.       …[2]
```

Unicodeは字類によって表現形式が異なるため、日本語が含まれる集団項目をDISPLAY文で表示する場合、文字化けを起こします[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、ロケール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。

18.3.7 COBOLファイル

レコード順ファイル、相対ファイル、索引ファイル、行順ファイルおよび表示ファイル(ACM/APL)はUnicodeデータを何も加工せずに入出力しますが、印刷ファイルおよび表示ファイル(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、注意点をまとめます。

ファイル識別名(全ファイル)

ファイル識別名にデータ名を指定している場合で、かつ、そのデータ名が日本語項目を含む集団項目だった場合、翻訳時エラーになります。

```
FILE-CONTROL.
     SELECT OUTFILE ASSIGN TO FILE-NAME.
       :
01 FILE-NAME.                 →翻訳エラー
  02 F-PATH  PIC X(10).
  02 F-NAME  PIC N(4) VALUE NC"ファイル".
     :
    MOVE "/home/foo/" TO F-PATH.
    OPEN OUTPUT OUTFILE.
```

これは、異なる表現形式の混在によるファイル名の文字化けを防ぐためです。ファイル識別名に集団項目を指定する場合は字類を英数字で統一してください。

行順ファイル

行順ファイルは、各種テキストエディタで表示・編集可能な形式であることが前提となるため、ひとつのファイルはひとつの表現形式で統一する必要があります。つまり、レコードを構成する各項目の字類を統一しなければなりません。COBOLでは、レコード定義の字類が英数字で統一されている場合はUTF-8で、字類が日本語で統一されている場合はUTF-16のビッグエンディアンで入出力します。以下の例のように字類が混在する場合、翻訳時エラーが出力されるため、用途に合わせて字類を統一してください。

```
FILE-CONTROL.
     SELECT OUTFILE ASSIGN TO "data.txt"
                   ORGANIZATION IS LINE SEQUENTIAL.
       :
FD OUTFILE.                   →翻訳時エラー
01 OUT-REC.
  02 REC-ID    PIC X(4).
  02 REC-DATA  PIC N(20).
```

作成された行順ファイルは、Unicodeを扱えるエディタで参照および更新することができます。

印刷ファイル

印刷ファイルの場合、レコード内の字類の統一は不要です。各項目の字類に合わせてCOBOLランタイムシステムがコード変換するため、表現形式が混在しても問題なく動作します。

```
FILE-CONTROL.
     SELECT OUT-FILE ASSIGN TO PRTFILE.
       :
FILE SECTION.
FD OUT-FILE.
01 OUT-REC   PIC X(80).
WORKING-STORAGE SECTION.
01 PRT-DATA  CHARACTER TYPE IS MODE-1.
  02 PRT-NO    PIC 9(4).
  02 PRT-ID    PIC X(4).
  02 PRT-NAME  PIC N(20).
       :
    WRITE OUT-REC FROM PRT-DATA AFTER ADVANCING 1 LINE.
```

ただし、日本語項目を含む集団項目を転記の受取り側項目に使用した場合、集団項目に字類と合わないコードの空白づめが行われます。これにより、EUCやシフトJISでは印刷可能であったものが、Unicodeでは文字化けなどの意図しない印刷結果になることがあります。

```
    FILE-CONTROL.
         SELECT OUT-FILE ASSIGN TO PRTFILE.
            :
    FILE SECTION.
    FD OUT-FILE.
    01 OUT-REC   CHARACTER TYPE IS MODE-1.
      02 OUT-DATA  PIC N(40).
    WORKING-STORAGE SECTION.
    01 PRT-DATA  CHARACTER TYPE IS MODE-1.
      02 PRT-NO    PIC N(4).
      02 PRT-ID    PIC N(4).
      02 PRT-NAME  PIC N(20).
            :
        MOVE  PRT-DATA TO OUT-REC.                  … [1]
        WRITE OUT-REC AFTER ADVANCING 1 LINE.
   *
        MOVE 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。

帳票定義体(FORMAT句付き印刷ファイル、表示ファイル(PRT))

帳票定義体に定義した英数字項目には、1バイトコードで表現される文字しか格納できません。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、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。

18.3.8 スクリーン機能

動作モードがUnicodeの場合、スクリーン機能は使用できません。

18.4 実行時の注意点

18.4.1 メッセージを出力するファイル

実行時メッセージを出力するファイル

動作モードがUnicodeの場合、実行時メッセージを出力するファイル(環境変数情報CBR_MESSOUTFILEにより指定)のコード系は以下のようになります。

－ 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。

参照

“4.3.2 実行時メッセージのファイル出力”

TRACE機能、COUNT機能が出力するファイル

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。

18.4.2 フォントについて

以下の機能を使用する場合、Unicodeに対応したフォントを使用してください。

FORMAT句付き印刷ファイルおよび表示ファイル

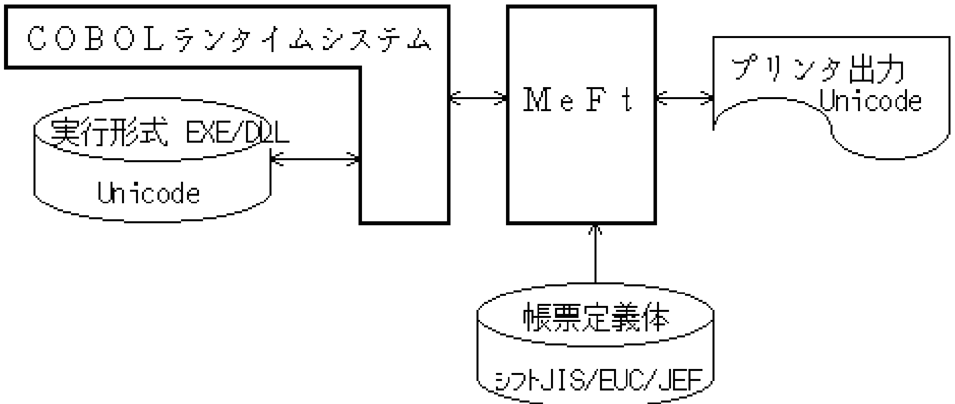
プリンタ情報ファイルおよびフォントテーブルに、Unicodeに対応したフォントを指定してください。指定方法については、“MeFtのオンラインマニュアル”および、“7.1.9 フォントテーブル”を参照してください。

18.5 関連製品連携

18.5.1 FORM/MeFt

FORMを使用して作成した帳票定義体をUnicodeアプリケーションで使用することができます。定義体は既存のものを使用することも、新規に作成することも可能です。また、実行時にMeFtが自動でUnicodeへコード変換するため、定義体のコード系は、シフトJIS、EUC、JEFのどれでも利用できます。

入出力するデータのコード系がUTF-16の場合、使用する製品のバージョンレベルによって、サロゲートペアの利用可否が異なります。MeFtのマニュアルでご確認ください。



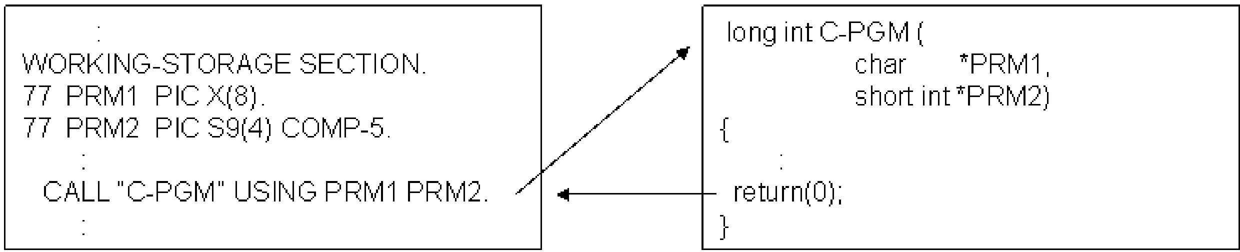
FORMAT句付き印刷ファイル、表示ファイル(帳票印刷)の利用方法および機能範囲については、EUCやシフトJISの場合と同じです。詳細は、“第7章 印刷処理”を参照してください。

表示ファイル(画面入出力)は、Unicodeアプリケーションで使用することができません。ご注意ください。

18.5.2 他言語間結合

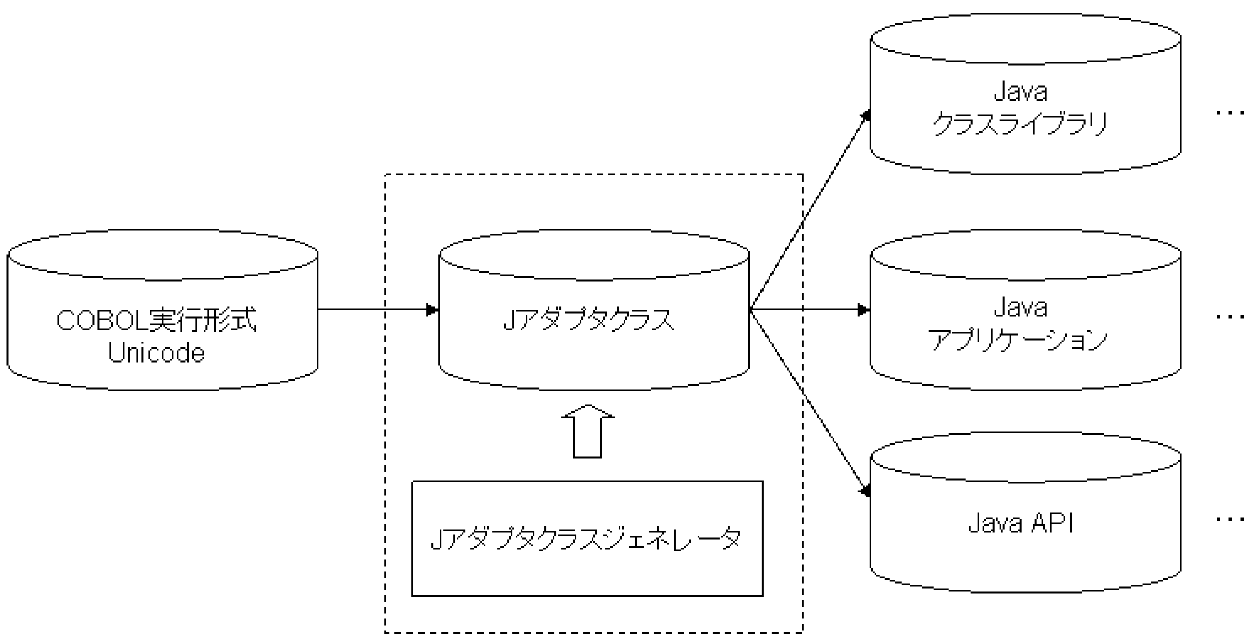
C言語

C言語では、char型がUTF-8表現となるため、COBOLの英数字項目との間でデータの送受が可能です。もちろん、数字系のデータはEUCやシフトJISと同じ要領で送受できます。



Java

Javaと連携する場合、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ができます。この際、Unicodeデータを送受することができます。



18.5.3 他のファイルシステム

RDMファイル

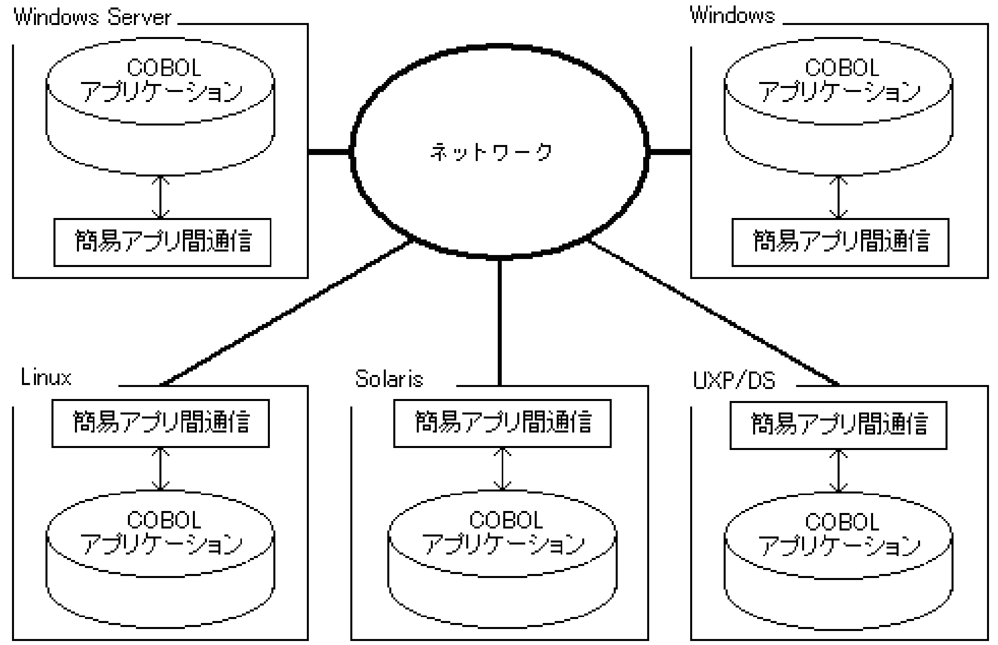
RDMファイルはUnicodeアプリケーションで使用することができません。ご注意ください。

C-ISAMファイル

C-ISAMファイルはUnicodeアプリケーションで使用することができません。ご注意ください。

18.5.4 通信機能

表示ファイル機能を利用した通信機能(ACM)、簡易アプリ間通信機能ともにUnicodeアプリケーションで使用することができます。



なお、メッセージおよび転送データのコード系は利用者責任で相手先に合わせてください。

18.5.5 Web連携

NetCOBOLが提供するCOBOL CGIサブルーチンはUnicodeをサポートしています。これらを利用してUnicodeで動作するWebアプリケーションが作成できます。

詳細は、"COBOL Webサブルーチン使用手引書"の"Webアプリケーションにおける Unicodeの利用"を参照してください。

COBOL SAFサブルーチンはUnicodeをサポートしておりません。ご注意ください。

第19章　通信機能

本章では、COBOLプログラムからメッセージ通信を行うための表示ファイル機能および簡易アプリ間通信機能について説明します。

19.1 通信の種類

ここでは、COBOLプログラムから通信を行う場合の特徴と用途について、説明します。

次の方法で通信を行います。

- 表示ファイル機能(非同期型メッセージ通信)
- 簡易アプリ間通信機能

それぞれの機能の特徴および用途を"表19.1 表示ファイル機能と簡易アプリ間通信機能の特徴・用途"に示します。

表19.1 表示ファイル機能と簡易アプリ間通信機能の特徴・用途

<table>
<tr><th colspan="2">特徴・用途</th><th>表示ファイル機能</th><th colspan="2">簡易アプリ間通信機能</th></tr>
<tr><td rowspan="4">特徴</td><td>通信機能</td><td>非同期型メッセージ通信</td><td colspan="2">非同期型メッセージ通信</td></tr>
<tr><td>通信形態</td><td>[クライアント-サーバ]
システム内通信だけ[1]</td><td>[クライアント]
Solaris
Linux
Windows XP
Windows Server 2003
Windows Vista
Windows Server 2008
Windows 7
Windows Server 2012
Windows 8</td><td>[サーバ]
Solaris
Linux
Windows XP
Windows Server 2003
Windows Vista
Windows Server 2008
Windows 7
Windows Server 2012
Windows 8</td></tr>
<tr><td>プログラムの記述内容</td><td>表示ファイルの定義
通信レコードの定義
OPEN文
READ文
WRITE 文
CLOSE 文</td><td colspan="2">通信レコードの定義
CALL文による簡易アプリ間通信機能の関数の呼出し
COBCI_OPEN
COBCI_READ
COBCI_WRITE
COBCI_CLOSE</td></tr>
<tr><td>必要な関連製品</td><td>PowerRW+(注1)([1])</td><td colspan="2">COBOL 単独</td></tr>
<tr><td colspan="2">用途</td><td>表示ファイル機能を使用して、間接的なメッセージ通信を行いたい場合</td><td colspan="2">COBOL 単体による簡単なデータのやりとりを行いたい場合</td></tr>
</table>

注1: ACM制御による通信を行う場合

備考: 表中の[1]は、通信形態および必要な関連製品が対応付いていることを表しています。

19.2 表示ファイル機能(非同期型メッセージ通信)

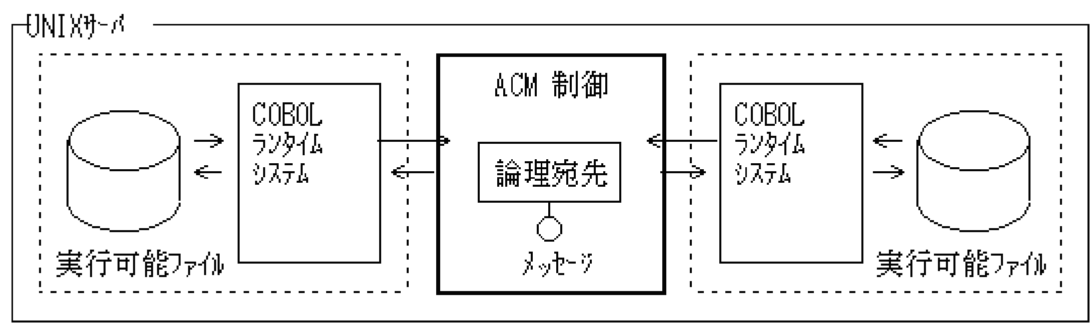
ここでは、メッセージ通信を行う表示ファイル機能の概要、動作環境、COBOLプログラムの作成および注意事項について説明します。

19.2.1 動作環境

表示ファイル機能を使用するときには、通信機能を実現する接続製品が必要です。

下表にACM制御を使用した場合の関連図を示します。

図19.1 ACM制御による通信(システム内通信)の関連図



ACM制御の動作可能なオペレーティングシステムについては、ACM制御のマニュアルおよびインストールガイドを確認してください。

19.2.2 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 "ACM"
        DESTINATION-1 IS   論理宛先名通知域
       [PROCESSING MODE IS 処理種別通知域]
       [PROCESSING TIME IS 監視時間通知域]
       [MESSAGE CLASS IS   優先順位通知域]
       [FILE STATUS IS   入出力状態１  入出力状態２].
 DATA DIVISION.
   FILE SECTION.
   FD  ファイル名
      [RECORD IS VARYING IN SIZE DEPENDING ON  レコードの大きさ].
   01  通信レコード名  ～.
   WORKING-STORAGE SECTION.
   01  論理宛先名通知域         PIC X(8).
  [01  処理種別通知域           PIC X(2).]
  [01  監視時間通知域           PIC 9(4).]
  [01  優先順位通知域           PIC 9.]
  [01  入出力状態１              PIC X(2).]
  [01  入出力状態２              PIC X(4).]
  [01  レコードの大きさ         PIC 9(5).]
 PROCEDURE DIVISION.
      OPEN I-O ファイル名.
      MOVE   論理宛先名      TO  論理宛先名通知域.
     [MOVE   処理種別        TO  処理種別通知域.]
     [MOVE   監視時間        TO  監視時間通知域.]
     [MOVE   優先順位        TO  優先順位通知域.]
      WRITE  通信レコード名.
      READ   ファイル名.
```

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

注意

コード系の異なるシステムとの通信を行う場合、iconv関数などを使用して、送受信メッセージのコード系を合わせる必要があります。

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19.2 ファイル管理記述項に指定する情報"に示します。

表19.2 ファイル管理記述項に指定する情報

	指定する場所	情報の種類	指定する内容および用途
必須	SELECT 句	ファイル名	COBOLプログラム中で使用するファイル名を指定します。このファイル名は、COBOLの利用者語の規則に従った名前にします。
	ASSIGN 句	ファイル参照子	"GS-ファイル識別名"の形式で指定します。このファイル識別名は、実行時に接続製品名を設定する環境変数となります。
	SYMBOLIC DESTINATION句	宛先種別の指定	"ACM"を指定します。
	DESTINATION-1 句	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、通信を行うとき、入出力処理の論理宛先名を設定します。
任意	FILE STATUS 句	データ名	作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、入出力処理の実行結果が設定されます。設定される値については、"付録B 入出力状態一覧"を参照してください。詳細情報は、4桁の英数字項目を指定します。
	MESSAGE CLASS 句	データ名	作業場所節または連絡節で、1桁の数字項目として定義したデータ名を指定します。このデータ名には、通信を行うとき、入出力処理の優先順位を1～9の範囲で設定します。1が最も高い優先順位です。省略または0を指定すると、指定した論理宛先の最低レベルとみなします。
	PROCESSING MODE 句	データ名	作業場所節または連絡節で、2桁の英数字項目として定義したデータ名を指定します。このデータ名には、通信を行うとき、入出力処理の処理種別を設定します。([参照]"表19.3 処理種別の種類と指定する値")
	PROCESSING TIME 句	データ名	作業場所節または連絡節で、4桁の数字項目として定義したデータ名を指定します。このデータ名には、通信を行うとき、入出力処理の監視時間を秒単位で設定します。省略または0を指定すると、無限待ち時間となります。

注意

ACM制御で接続する場合に有効になる句および設定する値については、"ACM制御説明書"を参照してください。

表19.3 処理種別の種類と指定する値

処理種別			処理内容
入力	待合せなし指示	空白"NW"	論理宛先にデータがない場合、入出力状態に"0A"を通知します。
	待合せ指示	"WT"	論理宛先にデータがない場合、監視時間の指定に従い、データが書き込まれるのを待ち合わせます。
出力	待合せなし指示	空白"NW"	論理宛先にデータが最大数まで書き込まれている場合、書込みを中止し、入出力状態に"9G"を通知します。

処理種別			処理内容
	待合せ指示	"WT"	論理宛先にデータが最大数まで書き込まれている場合、監視時間の指定に従い、データが読み込まれるのを待ち合わせます。
	強制実行指示	"NE"	論理宛先にデータが最大数まで書き込まれている場合、最大数を超えて書込みを行い、入出力状態に"0B"を通知します。

データ部(DATA DIVISION)

データ部には、通信に使用するレコードの定義およびファイル管理記述項に指定したデータの定義を記述します。

注意

表示ファイルに対してEXTERNAL句を指定する場合には“9.2.5.2 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)

通信処理には、通常のファイル処理を行うときと同様に、入出力文を使います。入出力文は、次に示す順序で実行します。

1. OPEN文 ： メッセージの送受信の開始
2. READ文およびWRITE文 ： メッセージの送受信
3. CLOSE文 ： メッセージの送受信の終了

OPEN文およびCLOSE文

OPEN文はメッセージの送受信処理の開始時に、CLOSE文はメッセージの送受信処理の終了時に、それぞれ1回だけ実行します。

READ文またはWRITE文

メッセージを送信するときには通信レコードを指定したWRITE文を、メッセージを受信するときには表示ファイルを指定したREAD文を使います。READ文およびWRITE文を実行する前には、DESTINATION-1句に指定したデータ名に論理宛先名を設定しておく必要があります。設定された論理宛先名に対応する宛先がメッセージの送受信の対象となります。

また、ファイル管理記述項に指定したデータに必要な情報を設定しておく必要があります。

入出力エラー処理

入出力エラーの検出方法および入出力エラーが発生したときの実行結果については、“6.6 入出力エラー処理”を参照してください。

19.2.3 プログラムの翻訳・リンク

ACM制御を使用する場合

特に必要な翻訳・リンクオプションはありません。

19.2.4 プログラムの実行

表示ファイル機能を使ったメッセージの送受信を行うプログラムを実行するときには、以下の環境設定が必要です。

ACM制御を使用する場合

COBOLプログラムの実行より前に、論理宛先の定義やACMの起動などの環境を整えておく必要があります。詳細は、“ACM制御説明書”を参照してください。

19.3 簡易アプリ間通信機能

ここでは、簡易アプリ間通信機能の使い方について説明します。なお、簡易アプリ間通信機能を使ってメッセージ通信を行う例題プログラムがサンプルプログラムとして提供されているので、参考にしてください。

19.3.1 概要

簡易アプリ間通信機能は、利用者プログラム間でメッセージの受渡しを行うための機能です。

利用者プログラム(以降、クライアントと呼びます)間のメッセージの受渡しは、メッセージを制御するシステムプロセス(以降、サーバと呼びます)と、その制御配下にある論理宛先を介して行われます。つまり、クライアントは、論理宛先に対してメッセージを送信することにより、サーバ内にそのメッセージを蓄積します。また、論理宛先を介してメッセージを受信することにより、サーバ内に蓄積されたメッセージを受け取ります。したがって、簡易アプリ間通信機能によってメッセージの受渡しを行うためには、サーバの起動や論理宛先の定義などの操作が必要です。これらの操作は、“19.3.10 コマンド”で示すコマンドを使用して行います。

メッセージの受渡し形態には、ローカルアクセスとリモートアクセスがあります。ローカルアクセスとは、サーバ、クライアントとも同一システムに存在する通信形態です。リモートアクセスとは、サーバとクライアントが別々のシステムに存在し、TCP/IPネットワークを介してメッセージの受渡しを行う通信形態です。それぞれの概要を“図19.2 ローカルアクセスの概要”、“図19.3 リモートアクセスの概要”に示します。

図19.2 ローカルアクセスの概要

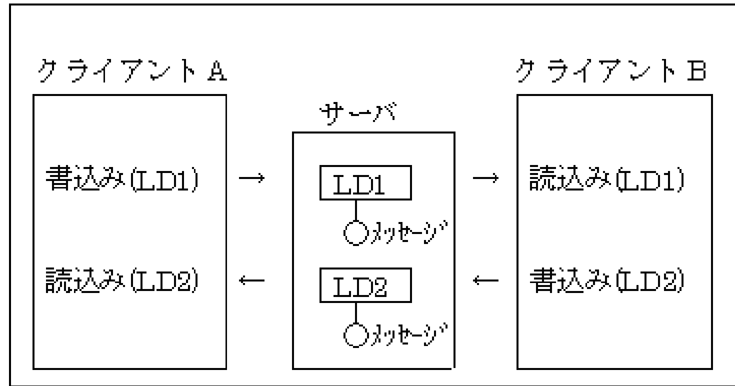
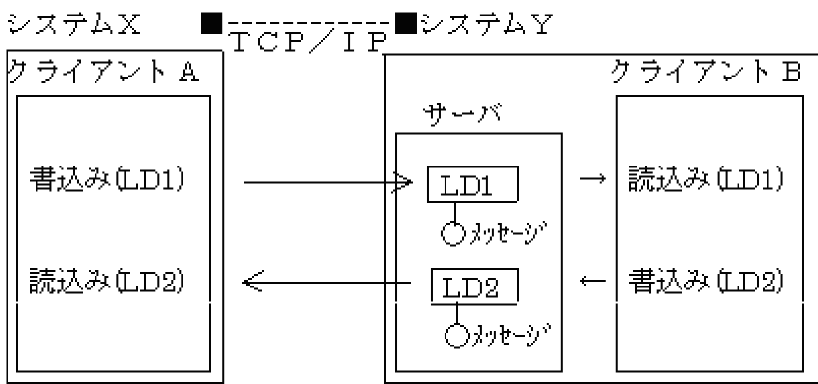
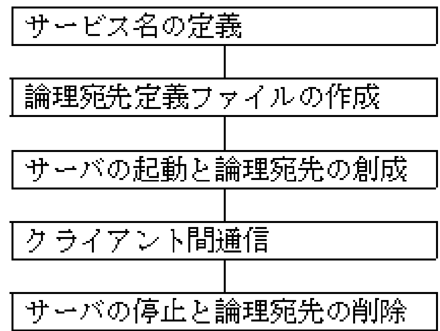


図19.3 リモートアクセスの概要



19.3.2 基本的な使い方

簡易アプリ間通信機能を使用してメッセージの受渡しを行う場合の基本的な使い方は以下のとおりです。



19.3.3 サービス名の定義

簡易アプリ間通信機能では、システムの通信機能を使用します。このため、/etc/servicesファイルに、システムの通信機能で使用するポート番号とサービス名を定義する必要があります。ポート番号には、システムで一意な値を設定し、リモートアクセスを行う場合には、ネットワーク内で統一した値を設定してください。なお、ポート番号の設定については、システム管理者と相談してください。

簡易アプリ間通信機能では、サービス名として"cobci"を固定的に使用します。以下に/etc/servicesファイルの指定形式を示します。

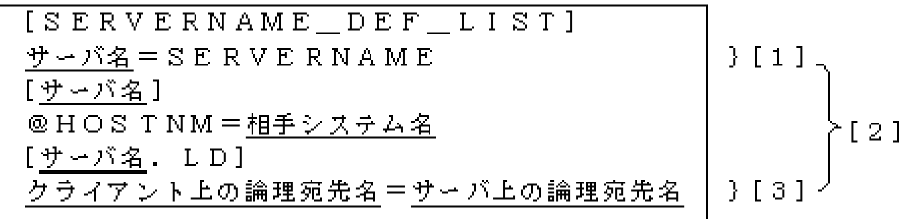
```
cobci ポート番号/tcp
```

下線部分に任意のポート番号を記述してください。

19.3.4 論理宛先定義ファイル

簡易アプリ間通信機能を使用する場合、クライアントごとに論理宛先定義ファイルを定義する必要があります。論理宛先定義ファイルは、サーバが動作しているシステム名、クライアント上の論理宛先名とサーバ上の論理宛先名の対応関係を定義します。

論理宛先定義ファイルは、テキスト形式のファイルで、以下の形式で記述します。



- 下線部に任意のサーバ名、相手システム名などを指定してください。
- サーバ名、相手システム名は、15文字以内の英数字で指定してください。
- クライアント上の論理宛先名、サーバ上の論理宛先名は、8文字以内の英数字で指定してください。
- [1]、[2]および[3]のそれぞれは、この範囲で繰り返し指定が可能であることを示します。
- ただし、[1]と[2]の繰り返し数は同じである必要があります。
- "#"または";"を記述すると、これらの記号からその行の右端までの記述を注釈とみなします。

以下に論理宛先定義ファイルの指定例を示します。

```
[SERVERNAME_DEF_LIST]
 SVR1=SERVERNAME
 SVR2=SERVERNAME
 [SVR1]
 @HOSTNM=HOST1
 [SVR1.LD]
 PLD1=SLD1
```

```
PLD2=SLD2
[SVR2]
@HOSTNM=HOST2
[SVR2.LD]
PLD3=SLD1
```

論理宛先定義ファイルとクライアントとの関連付けは、環境変数CBR_CI_INFに、使用する論理宛先定義ファイルのパス名を指定して行います。たとえば、/users/cobolに存在するldinfという論理宛先定義ファイルを使用する場合は、次のように指定します。

```
$ CBR_CI_INF=/users/cobol/ldinf
$ export CBR_CI_INF
```

注意

相手マシン名にIPアドレスを指定することはできません。IPアドレスへの名前解決可能なホスト名を指定してください。

本機能において、以下の範囲でIPv6アドレスに対応しています。

表19.4 簡易アプリ間通信機能のIPv6対応

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

一定時間ごとに変わる一時アドレス(プライバシ拡張)は、サポートしていません。IP アドレスが変更された場合は、簡易アプリ間通信機能のサーバとの接続を切断したあと、サーバとの接続を再度確立する必要があります。

19.3.5 サーバの起動と論理宛先の創成

サーバの起動と論理宛先の創成は、以下のように行います。各コマンドの詳細は、"19.3.10 コマンド"を参照してください。

サーバの起動

cobstrciコマンドにより、サーバを起動します。

論理宛先の創成

cobcrtldコマンドにより、論理宛先を創成します。

論理宛先のモード変更

cobactldコマンドにより、論理宛先を使用可能状態にします。

19.3.6 クライアントの処理

簡易アプリ間通信機能を使用してメッセージの受渡しを行うプログラムでは、以下の処理を行います。各関数の詳細は、"19.3.11 関数"を参照してください。

サーバとの接続

COBCI_OPEN関数を呼び出し、サーバとの接続を行います。

メッセージの書込み

COBCI_WRITE関数を呼び出し、論理宛先にメッセージを書き込みます。

メッセージの読込み

COBCI_READ関数を呼び出し、論理宛先からメッセージを読み込みます。

サーバとの切断

COBCI_CLOSE関数を呼び出し、サーバとの接続を解除します。

19.3.7 サーバの停止と論理宛先の削除

サーバの停止と論理宛先の削除は、以下のように行います。各コマンドの詳細は、"19.3.10 コマンド"を参照してください。

論理宛先のモード変更

cobdacldコマンドにより、論理宛先を使用不可能状態にします。

論理宛先のメッセージ削除

cobpurldコマンドにより、論理宛先に蓄積されているメッセージを削除します。

論理宛先の削除

cobdltldコマンドにより、論理宛先を削除します。

サーバの停止

cobstpciコマンドにより、サーバを停止します。

19.3.8 論理宛先の状態表示

cobdspldコマンドにより、論理宛先の状態を表示することができます。

図19.4 表示形式

```
最大格納ﾒｯｾｰｼﾞ数 999999999[1]          最大待ち命令数 999999999[2]

論理宛先名  ﾓｰﾄﾞ          現ﾒｯｾｰｼﾞ数     最大ﾒｯｾｰｼﾞ数  待ち命令数     最終処理時刻
            読み込み側                                 読み込み側       読み込み側
            書き込み側                                 書き込み側       書き込み側

XXXXXXXX[3] X[4]          999999999[6]    999999999[7]999999999[8]  MM/DD HH:MM[10]
            X[5]                                      999999999[9]  MM/DD HH:MM[11]
```

[1] サーバの最大格納メッセージ数を表示します。

[2] サーバの最大待ち命令数を表示します。

[3] 創成されている論理宛先名を表示します。

[4] 論理宛先からの読込みの可否を“○”または“×”で表示します。

[5] 論理宛先への書込みの可否を“○”または“×”で表示します。

[6] 論理宛先に格納されている現メッセージ数を表示します。

[7] 論理宛先に格納できる最大メッセージ数を表示します。

[8] 論理宛先に対する読込みの待ち命令数を表示します。

[9] 論理宛先に対する書込みの待ち命令数を表示します。

[10] 論理宛先に対する最終読込み時刻を表示します。

[11] 論理宛先に対する最終書込み時刻を表示します。

図19.5 表示例

```
最大格納ﾒｯｾｰｼﾞ数        256          最大待ち命令数         32

論理宛先名  ﾓｰﾄﾞ          現ﾒｯｾｰｼﾞ数  最大ﾒｯｾｰｼﾞ数  待ち命令数     最終処理時刻
            読み込み側                             読み込み側       読み込み側
            書き込み側                             書き込み側       書き込み側

LD1         ○                     3           20           0    03/22 18:50
            ○                                              0    03/22 17:50

LD2         ○                     0           10           2    03/22 18:55
            ○                                              0    03/22 17:00
```

19.3.9 ログ

簡易アプリ間通信機能では、メッセージ受渡しの履歴をログファイルに保存することができます。ログの採取は、サーバとクライアントで別々に指定することができます。

サーバでは、エラー情報とトレース情報の選択、および、メッセージをログとして採取する・しないの選択が可能です。エラー情報では、処理中に発生したエラーだけがログとして採取されます。トレース情報では、エラー情報に加えて処理要求の内容とその処理結果についての情報がログとして採取されます。

クライアントでは、トレース情報とメッセージがログとして採取されます。

ログの採取を行う場合、ログファイル名、ログファイルサイズの指定について以下の点に注意してください。

- ログファイルが存在しない場合、新規にログファイルが作成されます。
- ログファイルがすでに存在する場合、既存のログファイルにログ情報が追加されます。
- ログファイルサイズを指定した場合、ログの採取は、指定したサイズにログファイルが達するまで行われます。指定したサイズを超えた場合は、それ以降のログ採取は行われません。

- ログファイルサイズを指定しなかった場合、ログの採取は、使用中のシステム環境上で許されるファイルサイズに達するまで行われます。その限界を超えた場合、以降のログ採取は行われません。

ログ機能を使用するには、サーバ、クライアントごとに以下のように指定します。

サーバでの指定

サーバでは、ログ採取の開始と終了をコマンドによって指定することができます。

ログ採取の開始は、cobstrlgコマンドにより指定します。この際、ログファイル名と同時に、ログとして採取する情報の種類とログファイルのサイズを指定することができます。

ログ採取の終了は、cobstplgコマンドにより行います。

各コマンドの詳細は、"19.3.10 コマンド"を参照してください。

クライアントでの指定

クライアントでは、環境変数CBR_CI_CLGによってログの採取を指定します。この際、ログファイル名およびログファイルのサイズを","で区切って指定します。ログファイルのサイズは、KB(キロバイト)単位で指定します。なお、環境変数CBR_CI_CLGを指定した場合だけ、クライアントでのログ採取は行われます。

以下に/tmp/logclというログファイルを、100KBのサイズで使用する場合の指定例を示します。

```
$ CBR_CI_CLG=/tmp/logcl,100
$ export CBR_CI_CLG
```

ログ内容

ログは、以下の形式で採取されます。

サーバログ

```
< [1] > [ [2] ] ID：[3] Client：[4]   [5]
```

クライアントログ

```
< [1] > [ [2] ] ID：[3] Server：[4]   [5]
```

- [1] 採取年月日時刻
- [2] 事象

事象	意味
OPEN要求	接続を要求しました。または、接続の要求がありました。
CLOSE 要求	切断を要求しました。または、切断の要求がありました。
READ要求	読込みを要求しました。または、読込みの要求がありました。
WRITE 要求	書込みを要求しました。または、書込みの要求がありました。
OPEN終了	接続処理が終了しました。
CLOSE 終了	切断処理が終了しました。
READ終了	読込み処理が終了しました。
WRITE 終了	書込み処理が終了しました。
通信エラー	通信処理でエラーが発生しました。
内部処理関数エラー	システムの内部処理関数でエラーが発生しました。
内部矛盾	システムで内部矛盾が発生しました。

- [3] 識別番号(サーバまたはクライアントを識別する番号)
- [4] IPアドレス

  サーバログ ： クライアントのIPアドレス

クライアントログ ： サーバのホスト名またはIPアドレス

- [5] 付加情報

事象	付加情報
OPEN要求	－
CLOSE 要求	－
READ要求	COBCI_READ関数の以下のパラメタ ・ サーバの論理宛先名 ・ バッファ長 ・ 処理種別 ・ 監視時間
WRITE 要求	COBCI_WRITE 関数の以下のパラメタ ・ サーバの論理宛先名 ・ メッセージ長 ・ 優先順位 ・ 処理種別 ・ 監視時間 ・ メッセージ内容の出力指定がある場合は、書込みメッセージのダンプ
OPEN終了	・ エラーコード(rtn :10進表示)(注1) ・ 詳細コード(detail:10進表示)(注1)
CLOSE 終了	・ エラーコード(rtn :10進表示)(注1) ・ 詳細コード(detail:10進表示)(注1)
READ終了	・ エラーコード(rtn :10進表示)(注1) ・ 詳細コード(detail:10進表示)(注1) ・ メッセージ内容の出力指定がある場合は、読込みメッセージのダンプ
WRITE 終了	・ エラーコード(rtn :10進表示)(注1) ・ 詳細コード(detail:10進表示)(注1)
通信エラー	・ 通信関数名 ・ エラーコード(err :10進表示)(注2) ・ エラー内容
内部処理関数エラー	・ 内部処理関数名 ・ エラーコード(err :10進表示)(注2) ・ エラー内容
内部矛盾	エラー内容

注1: “19.3.12 関数のエラーコード”を参照してください。
注2: システムのerrno を参照してください。

注意

[3]以降の情報は、出力される場合とされない場合があります。

以下にエラー発生時の処置を示します。

事象	処置
通信エラー	システムのerrnoのエラーコードを参照して対処してください。
内部処理関数エラー	システムのerrno を参照して対処してください。
内部矛盾	メモリまたはディスク資源の不足が発生していないか、動作環境を見直してください。SEに連絡してください。

19.3.10 コマンド

簡易アプリ間通信機能では、以下のコマンドを使用してサーバの環境を設定することができます。

コマンド名	機能
cobstrci	サーバの起動
cobstpci	サーバの停止
cobcrtld	論理宛先の創成
cobdltld	論理宛先の削除
cobactld	論理宛先のモードの使用可能状態への変更
cobdacld	論理宛先のモードの使用不可状態への変更
cobpurld	論理宛先のメッセージ削除
cobdspld	論理宛先の状態の表示
cobstrlg	サーバのログ取得の開始
cobstplg	サーバのログ取得の停止

19.3.10.1 cobstrciコマンド

機能

サーバの起動を行います。サーバは、システムに1つだけ起動することができます。

コマンド形式

```
cobstrci [-w 最大待ち命令数] [-s 最大格納メッセージ数]
```

最大待ち命令数

サーバで待ち合わせできるクライアントからの読込み命令および書込み命令の最大数を、0～999999999の範囲で指定します。

0を指定した場合、または指定を省略した場合は、最大値を指定したとみなされます。

最大格納メッセージ数

サーバに格納できるメッセージの最大数を、0～999999999の範囲で指定します。

0を指定した場合、または指定を省略した場合は、最大値を指定したとみなされます。

19.3.10.2 cobstpciコマンド

機能

サーバの停止を行います。

コマンド形式

```
cobstpci
```

19.3.10.3 cobcrtldコマンド

機能

論理宛先の創成を行います。

コマンド形式

```
cobcrtld [-c 最大メッセージ数] 論理宛先名
```

最大メッセージ数

論理宛先に格納できるメッセージの最大数を、0～999999999の範囲で指定します。

0を指定した場合、または指定を省略した場合は、制限なしとみなされます。ただし、この場合でも、サーバとして格納可能なメッセージの最大数を超えてメッセージを格納することはできません。

論理宛先名

創成する論理宛先名を8文字以内の英数字で指定します。

19.3.10.4 cobdltldコマンド

機能

論理宛先の削除を行います。

コマンド形式

```
cobdltld  論理宛先名
```

論理宛先名

削除する論理宛先名を指定します。

19.3.10.5 cobactldコマンド

機能

論理宛先のモードを変更し、論理宛先を使用可能にします。

コマンド形式

```
cobactld [-t 種別] 論理宛先名
```

種別

使用可能とする種別を以下の文字列で指定します。省略した場合は、“RW”を指定したものとみなされます。

RW

読込み、書込みとも使用可能にします。

R

読込みだけ使用可能にします。

W

書込みだけ使用可能にします。

論理宛先名

使用可能にする論理宛先名を指定します。

19.3.10.6 cobdacldコマンド

機能

論理宛先のモードを変更し、論理宛先を使用不可能にします。

コマンド形式

```
cobdacld [-t 種別] 論理宛先名
```

種別

使用不可能とする種別を以下の文字列で指定します。省略した場合は、“RW”を指定したものとみなされます。

RW

読込み、書込みとも使用不可能にします。

R

読込みだけ使用不可能にします。

W

書込みだけ使用不可能にします。

論理宛先名

使用不可能にする論理宛先名を指定します。

19.3.10.7 cobpurldコマンド

機能

論理宛先に格納されているメッセージを削除します。

コマンド形式

```
cobpurld 論理宛先名
```

論理宛先名

メッセージを削除する論理宛先名を指定します。

19.3.10.8 cobdspldコマンド

機能

論理宛先の状態を表示します。表示内容、表示形式については、“19.3.8 論理宛先の状態表示”を参照してください。

コマンド形式

```
cobdspld
```

19.3.10.9 cobstrlgコマンド

機能

サーバでのログ採取を開始します。

コマンド形式

```
cobstrlg [-E] [-D] [-s ログファイルサイズ] -l ログファイル名
```

パラメタの説明

-E

エラー情報だけをログとして採取する場合に指定します。省略した場合は、すべての処理履歴をログとして採取します。

-D

メッセージの内容をログとして採取する場合に指定します。省略した場合、メッセージの内容はログとして採取されません。

ログファイルサイズ

ログファイルの上限サイズをKB(キロバイト)単位で指定します。省略した場合、ログファイルのサイズに上限を設定しないで、可能なかぎりログファイルを拡張してログの採取を続行します。

ログファイル名

ログファイルのパス名を指定します。相対パスで指定した場合は、カレントディレクトリからの相対パスとみなします。

19.3.10.10 cobstplgコマンド

機能

サーバでのログ採取を終了します。

コマンド形式

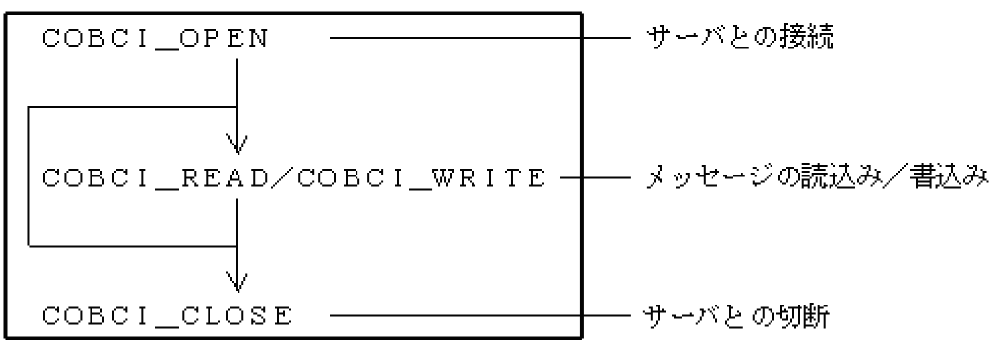
```
cobstplg
```

19.3.11 関数

簡易アプリ間通信機能のクライアントは、以下の関数をCALL文を使用して呼び出すことにより、サーバを介してクライアント間でメッセージの受渡しを行うことができます。

関数名	機能
COBCI_OPEN	サーバとの接続の確立
COBCI_CLOSE	サーバとの接続の切断
COBCI_READ	サーバからのメッセージの読込み
COBCI_WRITE	サーバへのメッセージの書込み

図19.6 呼出し順序



例

アプリケーション内での使用方法の例

```
 IDENTIFICATION DIVISION.
  PROGRAM-ID.  プログラム名.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
  WORKING-STORAGE SECTION.
```

```
  01  STATUS-REC.
      05  STATUS-ERROR     PIC S9(9)  COMP-5.
      05  STATUS-DETAIL    PIC S9(9)  COMP-5.
  01  SERVERNAME-REC       PIC X(15).
  01  SERVERHD-REC         PIC S9(9)  COMP-5.
  01  UNUSED-REC           PIC S9(9)  COMP-5  VALUE 0.
  01  LDNAME-REC           PIC X(8).
  01  BUFFER-REC           PIC X(n).
  01  RECEIVETYPE-REC.
      05  BUFFERLEN        PIC S9(9)  COMP-5.
      05  MSGLEN           PIC S9(9)  COMP-5.
      05  MSGTYPE          PIC S9(9)  COMP-5.
      05  WAITTIME         PIC S9(9)  COMP-5.
      05  UNUSED           PIC X(16)  VALUE LOW-VALUE.
  01  SENDTYPE-REC.
      05  MSGLEN           PIC S9(9)  COMP-5.
      05  PRIORITY         PIC S9(9)  COMP-5.
      05  MSGTYPE          PIC S9(9)  COMP-5.
      05  WAITTIME         PIC S9(9)  COMP-5.
      05  UNUSED           PIC X(16)  VALUE LOW-VALUE.
 PROCEDURE DIVISION.
     :
*  [サーバとの接続]
     CALL  "COBCI_OPEN"    USING  BY REFERENCE STATUS-REC
                                  BY REFERENCE SERVERNAME-REC
                                  BY REFERENCE SERVERHD-REC
                                  BY VALUE     UNUSED-REC.
     :
*  [メッセージの読込み]
     CALL  "COBCI_READ"    USING  BY REFERENCE STATUS-REC
                                  BY VALUE     SERVERHD-REC
                                  BY REFERENCE LDNAME-REC
                                  BY REFERENCE RECEIVETYPE-REC
                                  BY REFERENCE BUFFER-REC
                                  BY VALUE     UNUSED-REC.
     :
*  [メッセージの書込み]
     CALL  "COBCI_WRITE"   USING  BY REFERENCE STATUS-REC
                                  BY VALUE     SERVERHD-REC
                                  BY REFERENCE LDNAME-REC
                                  BY REFERENCE SENDTYPE-REC
                                  BY REFERENCE BUFFER-REC
                                  BY VALUE     UNUSED-REC.
     :
*  [サーバとの切断]
     CALL  "COBCI_CLOSE"   USING  BY REFERENCE STATUS-REC
                                  BY VALUE     SERVERHD-REC
                                  BY VALUE     UNUSED-REC.
     :
 END PROGRAM プログラム名.
```

19.3.11.1 COBCI_OPEN関数

機能

クライアントとサーバの接続を確立します。

サーバはサーバ名によって識別されます。接続が成功するとサーバ識別子が返されます。

呼出し形式

```
       :
DATA  DIVISION.
WORKING-STORAGE SECTION.
01  STATUS-REC.
    05  STATUS-ERROR   PIC S9(9)  COMP-5.
    05  STATUS-DETAIL  PIC S9(9)  COMP-5.
01  SERVERNAME-REC     PIC X(15).
01  SERVERHD-REC       PIC S9(9)  COMP-5.
01  UNUSED-REC         PIC S9(9)  COMP-5  VALUE  0.

PROCEDURE  DIVISION.

    CALL  "COBCI_OPEN"  USING  BY REFERENCE STATUS-REC
                               BY REFERENCE SERVERNAME-REC
                               BY REFERENCE SERVERHD-REC
                               BY VALUE   UNUSED-REC.
       :
```

パラメタ

STATUS-REC

STATUS-ERRORにはエラーコード、STATUS-DETAILには詳細コードをそれぞれ返します。値の詳細は、"19.3.12 関数のエラーコード"を参照してください。

SERVERNAME-REC

クライアントが接続するサーバのサーバ名を指定します。サーバ名は、論理宛先定義ファイルに定義している必要があります。サーバ名が15バイトに満たない場合、末尾には空白が必要となります。

SERVERHD-REC

サーバ識別子を返します。

UNUSED-REC

将来拡張のための予約域であり、0を設定している必要があります。

戻り値

サーバとの接続に成功した場合は、特殊レジスタPROGRAM-STATUSに0を返します。
サーバとの接続に失敗した場合は、特殊レジスタPROGRAM-STATUSに-1を返します。

19.3.11.2 COBCI_CLOSE関数

機能

クライアントとサーバを切断します。

サーバはCOBCI_OPEN関数で返されたサーバ識別子によって識別されます。

呼出し形式

```
       :
DATA  DIVISION.
WORKING-STORAGE SECTION.
01  STATUS-REC.
    05  STATUS-ERROR   PIC S9(9)  COMP-5.
    05  STATUS-DETAIL  PIC S9(9)  COMP-5.
01  SERVERHD-REC       PIC S9(9)  COMP-5.
01  UNUSED-REC         PIC S9(9)  COMP-5  VALUE  0.

PROCEDURE  DIVISION.
```

```
    CALL  "COBCI_CLOSE"  USING BY REFERENCE STATUS-REC
                               BY VALUE     SERVERHD-REC
                               BY VALUE     UNUSED-REC.
      :
```

パラメタ

STATUS-REC

STATUS-ERRORにはエラーコード、STATUS-DETAILには詳細コードをそれぞれ返します。値の詳細は、"19.3.12 関数のエラーコード"を参照してください。

SERVERHD-REC

COBCI_OPEN関数の呼出しで返されたサーバ識別子を指定します。

UNUSED-REC

将来拡張のための予約域であり、0を設定している必要があります。

戻り値

サーバとの切断に成功した場合は、特殊レジスタPROGRAM-STATUSに0を返します。
サーバとの切断に失敗した場合は、特殊レジスタPROGRAM-STATUSに-1を返します。

19.3.11.3 COBCI_READ関数

機能

サーバの論理宛先からメッセージを読み込みます。

なお、読み込まれたメッセージは、サーバ上から削除されます。

メッセージは、優先順位の高いものから読み込まれ、優先順位が同一のメッセージは、書き込まれた順に読み込まれます。論理宛先にメッセージが存在しない場合、処理種別の指定により処理の待ち合わせ方法を指定できます。

呼出し形式

```
      :
DATA DIVISION.
WORKING-STORAGE SECTION.
01  STATUS-REC.
    05  STATUS-ERROR   PIC S9(9)   COMP-5.
    05  STATUS-DETAIL  PIC S9(9)   COMP-5.
01  SERVERHD-REC       PIC S9(9)   COMP-5.
01  LDNAME-REC         PIC X(8).
01  RECEIVETYPE-REC.
    05  BUFFERLEN      PIC S9(9)   COMP-5.
    05  MSGLEN         PIC S9(9)   COMP-5.
    05  MSGTYPE        PIC S9(9)   COMP-5.
    05  WAITTIME       PIC S9(9)   COMP-5.
    05  UNUSED         PIC X(16)   VALUE  LOW-VALUE.
01  BUFFER-REC         PIC X(n).
01  UNUSED-REC         PIC S9(9)   COMP-5  VALUE  0.

PROCEDURE DIVISION.

    CALL  "COBCI_READ"   USING  BY REFERENCE STATUS-REC
                                BY VALUE     SERVERHD-REC
                                BY REFERENCE LDNAME-REC
                                BY REFERENCE RECEIVETYPE-REC
                                BY REFERENCE BUFFER-REC
                                BY VALUE     UNUSED-REC.
      :
```

パラメタ

STATUS-REC

STATUS-ERRORにはエラーコード、STATUS-DETAILには詳細コードをそれぞれ返します。値の詳細は、"19.3.12 関数のエラーコード"を参照してください。

SERVERHD-REC

COBCI_OPEN関数の呼出しで返されたサーバ識別子を指定します。

LDNAME-REC

メッセージを読み込むクライアント側の論理宛先名を指定します。論理宛先名が8バイトに満たない場合は、末尾に空白が必要となります。

RECEIVETYPE-REC

BUFFERLENには、メッセージを受け取るレコード領域の長さをバイト数で指定します。BUFFERLENの最大長は、32000バイトです。

MSGLENには、論理宛先から読み込んだメッセージのデータ長を返します。

MSGTYPEには、処理種別を指定します。処理種別には、以下のものがあります。

待ち合わせなし指示

MSGTYPEに0を指定することにより、待ち合わせなし指示となります。この場合、論理宛先にメッセージが存在しなければ、COBCI_READ関数は即時にエラー復帰します。このとき、その旨のエラーコードを返します。

待ち合わせ指示

MSGTYPEに1を指定することにより、待ち合わせ指示となります。この場合、サーバの最大待ち命令数以内で、論理宛先にメッセージが書き込まれるまで待ち合わせることができます。このとき、同時にWAITTIMEに監視時間を指定できます。監視時間は0～999999999(秒)の範囲で指定することができ、0を指定した場合は、無限待ち合わせとなります。

指定した時間内に論理宛先にメッセージが書き込まれなければ、COBCI_READ関数はエラー復帰します。このとき、その旨のエラーコードを返します。

WAITTIMEには、MSGTYPEに待ち合わせ指示を指定した場合の監視時間を指定します。

UNUSEDは、将来拡張のための予約域であり、LOW-VALUEを設定している必要があります。

BUFFER-REC

論理宛先のメッセージを受け取るレコード領域を指定します。ここに、メッセージ本文を返します。MSGLENに返された長さ以降の領域の内容は保証されません。

読み込んだメッセージがBUFFERLENに指定した長さより大きい場合は、BUFFERLENを超えた後ろのデータは切り捨てられます。

UNUSED-REC

将来拡張のための予約域であり、0を設定している必要があります。

戻り値

メッセージの読込みに成功した場合は、特殊レジスタPROGRAM-STATUSに0を返します。
メッセージの読込みに失敗した場合は、特殊レジスタPROGRAM-STATUSに-1を返します。

19.3.11.4 COBCI_WRITE関数

機能

サーバの論理宛先にメッセージを書き込みます。書き込むメッセージの優先順位を指定できます。論理宛先に空きがない場合、処理種別の指定により処理の待ち合わせ方法を指定できます。

呼出し形式

```
       :
 DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
01  STATUS-REC.
    05  STATUS-ERROR   PIC S9(9)  COMP-5.
    05  STATUS-DETAIL  PIC S9(9)  COMP-5.
01  SERVERHD-REC       PIC S9(9)  COMP-5.
01  LDNAME-REC         PIC X(8).
01  SENDTYPE-REC.
    05  MSGLEN         PIC S9(9)  COMP-5.
    05  PRIORITY       PIC S9(9)  COMP-5.
    05  MSGTYPE        PIC S9(9)  COMP-5.
    05  WAITTIME       PIC S9(9)  COMP-5.
    05  UNUSED         PIC X(16)  VALUE  LOW-VALUE.
01  BUFFER-REC         PIC X(n).
01  UNUSED-REC         PIC S9(9)  COMP-5  VALUE  0.

PROCEDURE DIVISION.

    CALL  "COBCI_WRITE"   USING   BY REFERENCE  STATUS-REC
                                  BY VALUE      SERVERHD-REC
                                  BY REFERENCE  LDNAME-REC
                                  BY REFERENCE  SENDTYPE-REC
                                  BY REFERENCE  BUFFER-REC
                                  BY VALUE      UNUSED-REC.
         :
```

パラメタ

STATUS-REC

STATUS-ERRORにはエラーコード、STATUS-DETAILには詳細コードをそれぞれ返します。値の詳細は、“19.3.12 関数のエラーコード”を参照してください。

SERVERHD-REC

COBCI_OPEN関数の呼出しで返されたサーバ識別子を指定します。

LDNAME-REC

メッセージを書き込むクライアント側の論理宛先名を指定します。論理宛先名が8バイトに満たない場合は、末尾に空白が必要となります。

SENDTYPE-REC

MSGLENには、論理宛先に書き込むメッセージのデータ長を指定します。書き込むことができる最大データ長は、32000バイトです。

PRIORITYには、論理宛先に書き込むメッセージの優先順位を指定します。

優先順位は1～9の範囲で、1が最高の優先順位です。優先順位が同一のメッセージは、書き込まれた順に読み込まれます。

MSGTYPEには、処理種別を指定します。処理種別には、以下のものがあります。

待ち合わせなし指示

MSGTYPEに0を指定することにより、待ち合わせなし指示となります。この場合、論理宛先に空きがなければ、COBCI_WRITE関数は即時にエラー復帰します。このとき、その旨のエラーコードを返します。

待ち合わせ指示

MSGTYPEに1を指定することにより、待ち合わせ指示となります。

この場合、サーバの最大待ち命令数以内で、論理宛先に空きができるまで待ち合わせることができます。

このとき、同時にWAITTIMEに監視時間を指定できます。監視時間は0～999999999(秒)の範囲で指定することができ、0を指定した場合は、無限待ち合わせとなります。

指定した時間内に論理宛先に空きができなければ、COBCI_WRITE関数はエラー復帰します。このとき、その旨のエラーコードを返します。

強制実行指示

MSGTYPEに2を指定することにより、強制実行指示となります。この場合、論理宛先の最大格納メッセージ数を超えてメッセージの書込みを行うことができます。論理宛先の最大格納メッセージ数を超えてメッセージの書込みを行った場合、COBCI_WRITE関数の復帰時にその旨のエラーコードを返します。ただし、サーバの最大格納メッセージ数を超えてメッセージの書込みを行うことはできません。

WAITTIMEには、MSGTYPEに待ち合わせ指示を指定した場合の監視時間を指定します。
UNUSEDは、将来拡張のための予約域であり、LOW-VALUEを設定している必要があります。

BUFFER-REC

論理宛先に書き込むメッセージを格納したレコード領域を指定します。

UNUSED-REC

将来拡張のための予約域であり、0を設定している必要があります。

戻り値

メッセージの書込みに成功した場合は、特殊レジスタPROGRAM-STATUSに0を返します。
メッセージの書込みに失敗した場合は、特殊レジスタPROGRAM-STATUSに-1を返します。

19.3.12 関数のエラーコード

簡易アプリ間通信機能で使用する関数のエラーコードについて説明します。

関数が正常に終了した場合、特殊レジスタPROGRAM-STATUSに0を返し、エラーコード(**STATUS-REC**の**STATUS-ERROR**)および詳細コード(**STATUS-REC**の**STATUS-DETAIL**)にも0を返します。エラーが発生した場合は、特殊レジスタPROGRAM-STATUSに-1を返し、“表19.5 簡易アプリ間通信の関数エラーコード一覧”に示すエラーコードおよび詳細コードを返します。なお、表中の“対象関数”の記号の意味は以下のとおりです。

O：COBCI_OPEN関数
C：COBCI_CLOSE関数
R：COBCI_READ関数
W：COBCI_WRITE関数

表19.5 簡易アプリ間通信の関数エラーコード一覧

エラーコード	詳細コード	意味	処置	対象関数			
				O	C	R	W
0	0	正常終了しました。	－	○	○	○	○
1	257	メッセージの読込みに成功しました。しかし、メッセージのデータ長よりバッファ長が小さいため、バッファ長以降のデータは切り捨てられました。	－			○	
2	513	読み込むメッセージがありません。(待ち合わせなし)	－			○	
3	769	論理宛先の最大格納メッセージ数を超えてメッセージを格納しました。(強制実行)	－				○
4	1025	監視時間が経過しました。そして、論理宛先にメッセージがなかったため、メッセージの読込みに失敗しました。(待ち合わせあり)	－			○	
	1026	監視時間が経過しました。しかし、論理宛先の最大格納メッセージ数、またはサーバ内最大格納メッセージ数を超えていたため、メッセージの書込みに失敗しました。(待ち合わせあり)	不要な未処理メッセージを削除してから実行してください。				○
6	1537	論理宛先が読込み可能状態でないため、メッセージの読込みに失敗しました。	サーバの論理宛先を読込み可能状態に変更してください。			○	

エラーコード	詳細コード	意味	処置	対象関数			
				O	C	R	W
	1538	論理宛先が書込み可能状態でないため、メッセージの書込みに失敗しました。	サーバの論理宛先を書込み可能状態に変更してください。				○
	1539	創成されていない論理宛先に対してメッセージの読込み/書込みを行いました。	サーバに論理宛先を創成してください。			○	○
7	1793	論理宛先の最大格納メッセージ数を超えたため、メッセージの書込みに失敗しました。(待ち合わせなし)	不要な未処理メッセージを削除してから実行してください。				○
8	2049	サーバの最大格納メッセージ数を超えたため、メッセージの書込みに失敗しました。または、サーバの最大待ち命令数を超えたため、メッセージの読込み/書込みに失敗しました。	不要な未処理メッセージを削除してから実行してください。			○	○
10	2561	すでに接続されているサーバに対して再度接続要求がされました。	同一サーバに対して再度接続要求をしていないか確認してください。	○			
11	2833	論理宛先名に誤りがあります	論理宛先名の指定に誤りがないか確認してください。			○	○
	2834	処理種別に誤りがあります。	処理種別の指定に誤りがないか確認してください。			○	○
	2835	メッセージ長/バッファ長に	メッセージ長/バッファ長の誤りがあります。指定に誤りがないか確認してください。			○	○
	2836	優先順位に誤りがあります。	優先順位の指定に誤りがないか確認してください。				○
	2837	監視時間に誤りがあります。	監視時間の指定に誤りがないか確認してください。			○	○
	2838	サーバ名に誤りがあります。	サーバ名の指定に誤りがないか確認してください。	○			
	2839	サーバ識別子に誤りがあります。	サーバに接続されているか、サーバ識別子の指定に誤りがないか確認してください。	○		○	○
12	3105	論理宛先定義ファイルが正しく指定されていません。	環境変数CBR_CI_INFに論理宛先定義ファイルのパス名が正しく設定されているか確認してください。	○		○	○
	3106	論理宛先定義ファイル内の指定に誤りがあります。	論理宛先定義ファイルの記述が正しいか、または、相手システム名(@HOSTNMに設定している相手システム名)を、/etc/hostsファイルに正しく定義しているか確認してください。	○		○	○
13	XXXX	システム間通信異常が発生しました。XXXXは、システムのerrno を示しています。	システムのerrnoを参照して原因を調査してください。なお、このエラーが発生した場合、サーバとの接続が切断されます。通信を続けるには、サーバとの接続から再度行ってください。	○	○	○	○
14	3633	メモリ資源不足が発生しました。	動作環境の見直しを行ってください。	○	○	○	○
15	3842	サーバとの接続が切断されました。	通信を続ける場合には、サーバとの接続から再度行ってください。	○	○	○	○
	3844	クライアントが強制終了されました。	－	○	○	○	○

エラーコード	詳細コード	意味	処置	対象関数			
				O	C	R	W
	4080以上	内部矛盾を検出しました。	SEに連絡してください。	○	○	○	○

○：通知される

19.3.13 プログラムのリンクに関する注意事項

簡易アプリ間通信機能を使用してメッセージ通信を行うプログラムを作成する場合、実行可能プログラムまたは副プログラムの共用ライブラリのリンク時に、libcobcicl.soまたはlibrcobcicl.so(マルチスレッドの場合)をリンクしてください。なお、libcobcicl.soおよびlibrcobcicl.soは、COBOLランタイムシステムの一部として提供される共用ライブラリです。

プログラムのリンクについては、“第3章 プログラムの翻訳とリンク”を参照してください。

第20章　Web連携

COBOLが提供するWeb連携機能を利用して、インターネット/イントラネットを活用した基幹業務システムをCOBOLだけで構築することができます。

インターネット/イントラネット環境では、一般に、Perl、java、Cなどの言語を利用して構築する例が広く知られています。COBOL Web連携機能では、これらの言語に加え、COBOLを利用したインターネット/イントラネットアプリケーションを構築することが可能となります。

これにより、膨大なCOBOL既存資産やCOBOL技術者のノウハウを有効活用し、先端技術・急速に普及するネットワーク環境に対応することができます。

COBOLが提供するWeb連携機能は、以下のカテゴリに分けることができます。

COBOL Webサブルーチンを利用したアプリケーションの開発

Webブラウザとのデータ入出力を使用したアプリケーション開発に必要なパラメタ解析、処理結果出力などの複雑な処理を意識しないで、COBOLの知識だけでアプリケーションを作成できます。

COBOL Webサブルーチンでは、CGI、SAF(NSAPI)をサポートしています。

MeFt/Webの利用

以下の既存のCOBOLアプリケーションを利用して、短期間でイントラネット環境の基幹業務として運用できます。

- MeFtを使用したCOBOLアプリケーション
- クライアント/サーバ環境のMeFt/NET-SVを使用したCOBOLアプリケーション

詳細は、"NetCOBOL Web連携ガイド"、"COBOL Webサブルーチン使用手引書"およびMeFt/Web説明書を参照してください。

注意

Webアプリケーションは、Webサーバあるいは MeFt/Webなどのデーモン配下で動作するアプリケーションとなります。

第21章　CORBAアプリケーション

COBOLインタフェースのCORBAサーバアプリケーションおよびCORBAクライアントアプリケーションを作成し、CORBAの提供するサービスを利用することができます。

21.1 CORBAの概要

分散オブジェクトシステムを構築

CORBAは、UNIXやPCなどのプラットフォームやアプリケーション開発言語に依存しないアプリケーションの実行環境を提供します。また、ネットワーク上に分散するアプリケーションなどの分散資源を一元管理し、業務の拡張や規模の拡大に応じたシステム形態の拡張に即応可能です。

既存・新規、あるいは部門別などで開発する様々なアプリケーションをコンポーネントとし、それらを組み合わせることにより、様々なビジネスシーンに即応できる情報システムを実現します。

クライアント／サーバ通信

CORBAクライアント／サーバアプリケーションを使用することにより、システム間の通信が可能です。CORBAクライアント／サーバアプリケーションは、COBOLのほかに、Java、C、C++やVisual Basicなどで記述できます。

トランザクションの利用による高信頼化

データベース管理システムが提供するデータベースと連携して、トランザクション処理を行うCORBAアプリケーションをトランザクションアプリケーションと呼びます。トランザクション機能を利用することにより、信頼性を要求される業務システムを容易に構築することが可能になります。

利用方法

CORBAアプリケーションの作成および実行を行う方法については、以下のマニュアルを参照してください。

－ Interstage Application Server 分散アプリケーション作成ガイド（CORBAサービス編）

－ Interstage Application Server 分散アプリケーション作成ガイド（コンポーネントトランザクションサービス編）

21.2 注意事項

COBOLインタフェースによるCORBAアプリケーションを作成する場合は、以下のマニュアルの注意事項に従って作成してください。

・“Interstage Application Server 分散アプリケーション作成ガイド（CORBAサービス編）”のCOBOLアプリケーション使用時の注意

第22章　プロジェクトマネージャの使い方

本章では、プロジェクトマネージャについて説明します。

プロジェクトマネージャは、将来のNetCOBOLリリースで提供を停止する予定です。Windows版NetCOBOL Studioのリモート開発による開発形態への移行をご検討ください。

22.1 プロジェクトマネージャの概念

プロジェクトマネージャは、プログラム開発作業全般を支援するためのユーティリティであり、Xウィンドウ上で動作するツール群によって構成されています。

プロジェクトマネージャではプロジェクト単位での資産管理や各開発工程の状況把握などの機能を提供しているため、小規模から大規模なプログラムの開発に有効な開発環境として利用できます。

プロジェクトマネージャを構成するツールはそれぞれ独立して動作できるので、直接コマンドによって起動し、単体でも利用することができます。プロジェクトマネージャを構成するツールは以下のとおりです。

ツール	コマンド
プロジェクトマネージャ	pmgr
エディタ	pmgredit
メッセージ管理ツール	pmgrmsg
ソース解析ツール	pmgrgrep
ビルダ	pmgrbuild
翻訳オプション設定ツール	pmgrcbi
リンクオプション設定ツール	pmgrlni
対話型デバッガ	svd
SCREEN DESIGNER	sdesign
ファイルユーティリティ	cobfuty
クラス情報出力コマンド	pmgr_vrep

各ツールの詳細についてはマニュアル、各ツールのヘルプまたはmanマニュアルを参照してください。

なお、プロジェクトマネージャはUNIXコマンドとの連携も可能です。UNIXで提供されているツールや他社製のツールもプロジェクトマネージャから利用できるように設計されています。

プロジェクトマネージャにより提供されている特徴的な機能について説明します。

プロジェクト管理機能

プロジェクト管理機能とは、アプリケーションを開発するうえで必要な資源を任意のひとまとまりとしてのプロジェクトという単位で管理できるようにするための機能です。

プロジェクトをプログラム構成にあわせて定義することにより、Makefileが生成され、効率的かつ確実に必要な翻訳およびリンク作業が実施されます。

さらに、プロジェクト単位で、構成ファイルへのメモ設定、状態確認・オプション設定・環境変数(実行時/デバッグ時)の保存などが行えるため、構成ファイルが少ない場合にも有効です。

ツール管理機能

ツール管理機能とは、ユーザが任意のツールを組み込んで利用できるようにするための機能です。

プロジェクトマネージャおよび構成ツールでは、ユーザが自由にカスタマイズ可能なツールメニューが用意されており、それにより任意のツールを自由に起動できるようにしてあります。

さらに、プロジェクトマネージャでは、プロジェクトを構成するファイルの拡張子に応じたポップアップメニューについてもカスタマイズが可能です。

これらを活用することで、自分の好みに合わせた開発支援環境を構築することが可能です。

なお、インストール時には、COBOLの開発におすすめの環境設定がデフォルトとして設定されているので参考にしてください。

履歴およびテンプレート機能

履歴およびテンプレート機能とは、最近利用したファイルやユーザ指定などを記憶し履歴として利用できるようにする機能です。よく使う指定や間違いやすい指定などをテンプレートとして定義することにより選択式で操作可能とし、操作ミスおよび記憶を軽減することができます。

メッセージ管理機能

メッセージ管理機能とは、翻訳エラーおよびgrepコマンドによる解析結果のように、“ファイル名:行番号:メッセージ内容”という3つのフィールドからなるメッセージを1つの単位として管理する機能です。

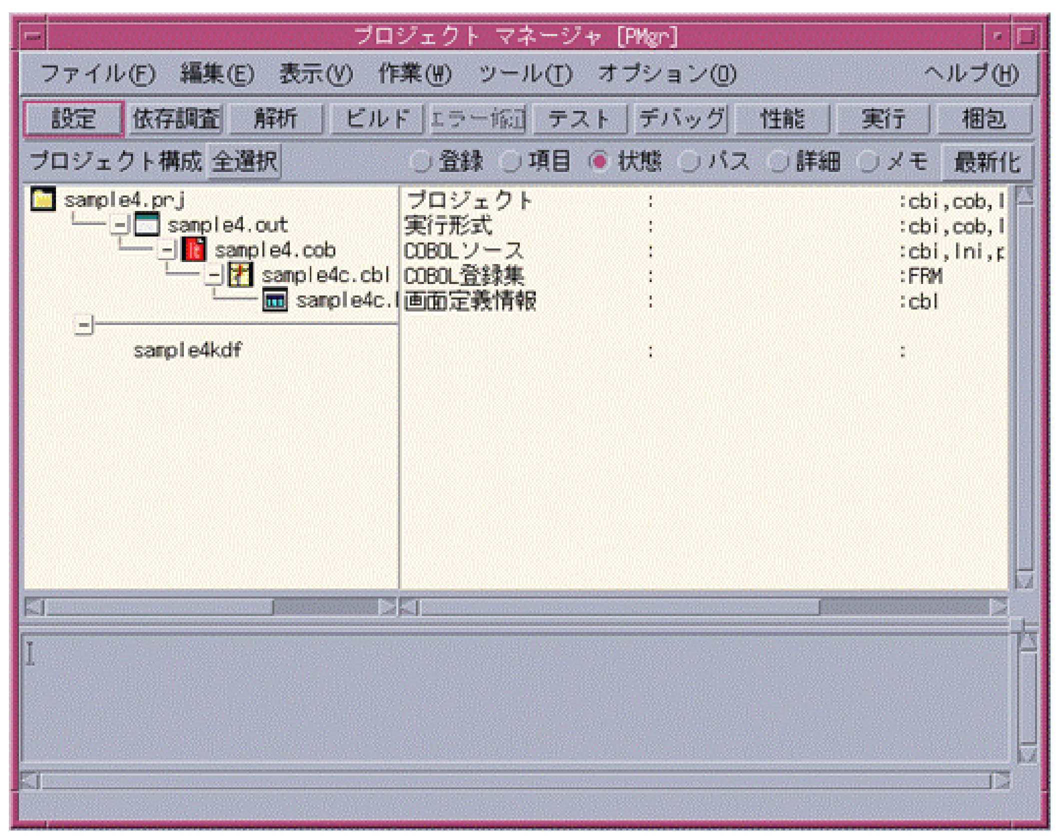
メッセージ管理では、各フィールドに桁合わせして分かりやすく表示し、タグジャンプ、ソート、および検索機能が利用できます。

また、複数のメッセージ管理ウィンドウ間でのデータ転送が可能であり、任意のメッセージだけをまとめて保存することも可能です。

22.2 プロジェクトマネージャ

プロジェクトマネージャは、プログラム開発作業全般を支援するためのユーティリティであり、Xウィンドウ上で動作するツール群によって構成されています。

プロジェクトマネージャではプロジェクト単位での資産管理や各開発工程の状況把握などの機能を提供しているため、小規模から大規模なプログラムの開発に有効な開発環境として利用できます。



プロジェクトマネージャを開くには

以下のコマンドにより、プロジェクトマネージャを起動することができます。

```
pmgr [プロジェクトファイル名]
```

22.2.1 プロジェクトマネージャの使い方

プロジェクトマネージャはツールバーによって開発作業を順次進めることができます。また、ステータスバーやポップアップメニューを利用することにより、可能な作業を把握することができます。さらに、ソースプログラムなどの各資源に対応するツールや作業が関連付けられているため、資源をダブルクリックするだけで適切なツールが起動されます。これにより複雑なメニュー操作を行うことなく開発作業を進めることができます。

ここではプロジェクトマネージャのツールバーに沿って基本的な使い方について説明します。

なお、各資源に関連付けられているツールや、ツールが使用する機能(動作)などプロジェクトマネージャを構成する各機能の定義は、あらかじめ基本的な使い方ができるように設定されています。そのため特に環境設定は必要ありません。なお、利用者にとってより最適な環境に変更することができるように設計されているので、必要に応じてカスタマイズしてください。

22.2.1.1 設定（プロジェクト管理機能）

プロジェクトファイルを新規作成するには

1. [ファイル] メニューの[新規作成]を選択します。
2. 設定ダイアログのプロジェクトにプロジェクト名とプロジェクトのベースディレクトリを入力します。
3. 設定ダイアログのユーザ資源にソースプログラムなど翻訳時に使用する各資源を格納したディレクトリを指定します。
4. 設定ダイアログの開発資源に翻訳時に出力されるオブジェクトや翻訳リストなどの各資源の格納ディレクトリを指定します。
5. 設定ダイアログの環境変数にプログラム実行時に使用する環境変数を指定します。
6. 設定ダイアログのデバッグ用環境変数にデバッグ時に使用する環境変数を指定します。

プロジェクト名以外の各情報については、いつでも変更可能です。

既存のプロジェクトファイルを開くには

1. [ファイル] メニューの[オープン]を選択します。
2. 保存しているプロジェクトファイルを選択します。

22.2.1.2 プロジェクト構成の定義

プロジェクトを構成する資源を登録するには

1. [編集] メニューの[登録] または[表示] メニューの[登録]を選択します。
2. 登録するファイル名を指定します。なお、ファイルは登録時に存在する必要はありません。
3. 以下の種別を選んで登録します。

ターゲット	作成する実行形式や共用オブジェクト(.so)、ターゲットリポジトリなどのファイル名
ソース	ターゲットを作成するためのソース、オブジェクト(ソースプログラムが存在する場合にはオブジェクトを登録する必要はありません。)
登録集	ソースで利用される登録集、定義体、依存リポジトリのファイル名
リソース	画面定義情報ファイルなど、登録集を作成するための資源のファイル名
その他	ドキュメントやデータファイルなど、プログラム構成に関係のない資源のファイル名

なお、[依存調査による自動登録]がチェックされている場合には、すでに存在するソースを登録すると、[作業] メニューの[依存調査]を選択されたのと同様に、登録集などの情報は自動的に登録されます。詳細については、“22.2.1.5 依存調査”を参照してください。

22.2.1.3 ソースプログラムの作成・変更

登録したソースプログラムをエディタで作成・変更するには

1. 作成・変更したい資源をダブルクリックしエディタを起動します。(ソースが存在しない場合、登録した名前で新規作成されます。)

2. ソースプログラムを保存します。

エディタの操作に関する詳細についてはエディタのヘルプを参照してください。

22.2.1.4 ソースプログラムの解析機能

ソースプログラム中の任意の文字列を検索することによりプログラム解析作業を補助します。プログラム名やデータ名などの検索も可能なので、コーディング作業時やプログラムの分析や修正による影響調査に威力を発揮します。

1. [ツール] メニューの[ソース解析] を選択します。
2. ソース解析ツールのディレクトリにソース解析時のカレントディレクトリを入力します。
3. ソース解析ツールの対象ソースに解析するソースプログラム名を入力します。
4. ソース解析ツールのパターンに検索するデータを入力します。または、検索パターンが登録されているテンプレートまたは履歴のリストを選択します。
パターンに検索するデータを入力する場合には次のように設定します。

検索文字列	入力データ
A	A
A または B	-e A -e B
ファイルでパターンを指定	-f ファイル名

5. [解析開始]ボタンを選択します。

なお、ソース解析ツールの詳細についてはソース解析ツールのヘルプを参照してください。

翻訳オプションを設定するには

1. [作業] メニューの[設定]-[翻訳オプション] を選択します。
2. 翻訳オプション画面のオプション一覧で設定したいオプションを選択します。
3. 翻訳オプション画面のオプション設定で詳細な設定値を選択します。

プロジェクトを選択または何も選択しないで翻訳オプション画面で翻訳オプションを設定すると、プロジェクトを構成するすべてのソースプログラムに設定した翻訳オプションが有効になります。

また、各ソースプログラムをそれぞれ選択して翻訳オプションを設定すると選択したソースプログラムにだけ設定した翻訳オプションが有効になります。

プロジェクト全体に指定した翻訳オプションと各ソースプログラムに指定した翻訳オプションがそれぞれ指定された場合、各ソースプログラムに指定した翻訳オプションが有効となります。

なお、設定した翻訳オプションは.cbiファイルとして格納されます。.cbiファイルはソースファイル名+拡張子(.cbi)として作成されます。

リンクオプションを設定するには

1. [作業] メニューの[設定]-[リンクオプション] を選択します。
2. リンクオプション画面の結合モードの指定でリンクモジュールの有無またはリンク方法を選択します。
3. リンクオプション画面のWlオプションの指定で追加するリンクオプション(ldコマンドのオプション)を指定します。
4. リンクオプション画面のライブラリサーチパスの指定でリンクモジュールの格納ディレクトリを指定します。
5. リンクオプション画面のリンクするライブラリの指定でリンクモジュール名(libA.soをリンクする場合はA)を指定します。
6. 次の各機能を使用している場合、必要なリンクオプションを選択します。
   - 画面帳票定義体を使用している場合
   - スクリーン操作機能を使用している場合
   - C言語から呼び出されるプログラムをリンクしている場合
   - C-ISAMを使用している場合

7. リンクオプション画面のその他のオプションの指定でその他追加するオプションを指定します。

プロジェクトを選択または何も選択しないでリンクオプション画面でリンクオプションを設定すると、プロジェクトを構成するすべてのターゲットに設定したリンクオプションが有効になります。

また、各ターゲットをそれぞれ選択してリンクオプションを設定すると選択したターゲットにだけ設定したリンクオプションが有効になります。

プロジェクト全体に指定したリンクオプションと各ターゲットに指定したリンクオプションがそれぞれ指定された場合、各ターゲットに指定したリンクオプションが有効となります。

なお、設定したリンクオプションは.lniファイルとして格納されます。.lniファイルはターゲットファイル名+拡張子(.lni)として作成されます。

22.2.1.5 依存調査

1. 依存関係を調査する対象となるソースを選択します。
2. [作業]メニューの[依存調査]またはツールバーの[依存調査]を選択します。

依存調査はすでに定義されているプロジェクト構成を元に、ソースプログラムを検索し依存関係を調査して自動的に依存関係を設定します。

ただし、ソースプログラム中に“CALL 一意名”が含まれている場合、“CALL 一意名”によって参照されるソースプログラムの依存関係は設定されません。

"CALL 一意名"によって参照されるソースプログラムについてはそれぞれ手動で登録してください。登録方法については“22.2.1.2 プロジェクト構成の定義”を参照してください。

22.2.1.6 解析

ソースプログラム中の任意の文字列を検索することによりプログラム解析作業を補助します。プログラム名やデータ名などの検索も可能なので、コーディング作業時やプログラムの分析や修正による影響調査に威力を発揮します。

1. [ツール] メニューの[ソース解析] を選択します。
2. ソース解析ツールのディレクトリにソース解析時のカレントディレクトリを入力します。
3. ソース解析ツールの対象ソースに解析するソースプログラム名を入力します。
4. ソース解析ツールのパターンに検索するデータを入力します。または、検索パターンが登録されているテンプレートまたは履歴のリストを選択します。
パターンに検索するデータを入力する場合には次のように設定します。

検索文字列	入力データ
A	A
A または B	-e A -e B
ファイルでパターンを指定	-f ファイル名

5. [解析開始]ボタンを選択します。

なお、ソース解析ツールの詳細についてはソース解析ツールのヘルプを参照してください。

22.2.1.7 ビルド

翻訳オプションを設定するには

1. 翻訳オプションを設定するソースプログラムを選択します。
2. ポップアップメニューの[翻訳オプション設定]を選択します。
3. 翻訳オプション画面のオプション一覧で設定したいオプションを選択します。
4. 翻訳オプション画面のオプション設定で詳細な設定値を選択します。

なお、翻訳オプションの設定方法については“22.7 翻訳オプション設定ツール”を参照してください。

リンクオプションを設定するには

ビルドではすでに作成されたMakefileを利用して作業を行うため、リンクオプションを変更することはできません。

このため、リンクオプションを変更する場合には、[作業]メニューの[設定]-[リンクオプション]を選択し、リンクオプションを設定した後、ツールバーの[ビルド]を再度指定してください。

なお、リンクオプションの設定方法については“22.8 リンクオプション設定ツール”を参照してください。

プログラムを翻訳/リンクするには

1. [作業]メニューの[ビルド]またはツールバーの[ビルド]を選択します。
2. ビルダ画面のMakefileに既存のMakefile名を指定します。デフォルト値としてビルダ起動時に自動生成されたMakefile (プロジェクト名.mk)が設定されます。
3. [ビルド]メニューの[ビルド]またはビルダ画面の[ビルド]ボタンを選択します。

22.2.1.8 エラー修正

翻訳エラーを修正するには

1. ビルダ画面に表示されている翻訳エラーメッセージを選択(ダブルクリック)すると、自動的にエディタが起動され、エラーが発生しているソースプログラムのエラー行にカーソルが位置づけられます。
2. 翻訳エラーメッセージに従ってソースプログラムを修正します。

また、ソースプログラムの翻訳時に発生した翻訳エラーはメッセージ管理機能により管理できるので、エラーを残したまま作業を中断/再開することができます。作業を再開する場合には[作業]メニューの[エラー修正]を選択します。メッセージ管理機能の詳細についてはメッセージ管理機能のヘルプを参照してください。

22.2.1.9 テスト機能

テスト進捗を管理するには

1. [作業]メニューの[テスト]を選択します。
2. テスト進捗管理モードで対話型デバッガが起動されます。

テスト進捗管理モードでは，対話型デバッガのテスト網羅度測定機能が使用されます。テスト網羅度測定機能の詳細については“23.6 テスト網羅度測定機能”を参照してください。

テスト環境を設定するには

テスト・デバッグ用の環境変数定義が指定されている場合、動作の前にその環境変数が設定されます。環境変数定義については“22.2.1.1 設定（プロジェクト管理機能）”を参照してください。

22.2.1.10 デバッグ機能

プログラムをデバッグするには

1. [作業]メニューの[デバッグ]を選択します。
2. 対話型デバッガを操作してデバッグします。

なお、対話型デバッガを使ったデバッグ方法の詳細については“第23章 対話型デバッガの使い方”を参照してください。

デバッグ環境を設定するには

テスト・デバッグ用の環境変数定義が指定されている場合、動作の前にその環境変数が設定されます。環境変数定義については“22.2.1.1 設定（プロジェクト管理機能）”を参照してください。

22.2.1.11 性能

性能チューニングするには

1. [作業]メニューの[性能]を選択します。
2. 性能チューニングモードで対話型デバッガが起動されます。

性能チューニングモードでは、対話型デバッガのカバレージ機能が使用されます。カバレージ機能の詳細については"23.7 カバレージ機能"を参照してください。

性能チューニングの環境を設定するには

テスト･デバッグ用の環境変数定義が指定されている場合、動作の前にその環境変数が設定されます。

22.2.1.12 実行

プログラムを実行するには

1. 実行する実行形式プログラムを選択します。
2. [作業]メニューの[実行]を選択します。

プログラム実行時の環境を設定するには

環境変数の設定など、プログラムの実行に必要な実行環境をあらかじめ定義しておくことにより、プログラム実行前にその環境変数を自動的に設定します。

22.2.1.13 梱包

プロジェクト資産を梱包するには

[作業]メニューの[梱包]を選択します。

梱包を実施するとプロジェクトの全資源をtar形式で梱包した.fixとターゲットをtar形式で梱包した.dptが生成されます。

22.2.1.14 カスタマイズ

プロジェクトマネージャをカスタマイズするには

プロジェクトマネージャの以下の動作はシェルプログラムによって定義されているので、必要に応じて動作を変更することが可能です。

作業	シェルプログラム名
解析	pmgr_analy.csh
ビルド	pmgr_build.csh
エラー修正	pmgr_err.csh
テスト	pmgr_test.csh
デバッグ	pmgr_debug.csh
性能	pmgr_cover.csh
実行	pmgr_exec.csh
梱包	pmgr_pack.csh

なお、各ツールのカスタマイズについては、各ツールの[カスタマイズ]メニューで変更できます。詳細については各ツールのヘルプを参照してください。

22.2.2 プロジェクトマネージャのリファレンス

22.2.2.1 プロジェクトマネージャのメニュー

メニュー・アクセラレータ

カット	[Ctrl] + [X]キー
コピー	[Ctrl] + [C]キー
ペースト	[Ctrl] + [V]キー
終了	[Alt] + [F4] キー

[ファイル]メニュー

新規作成	プロジェクトファイルを新規に作成します。
オープン	既存のプロジェクトファイルをオープンします。
保存	プロジェクトファイルを上書きして保存します。
別名保存	プロジェクトファイルを変名して保存します。
終了	プロジェクトマネージャを終了します。

[編集]メニュー

登録	プロジェクトファイルに資源を登録します。
カット	資源をカットします。
コピー	資源をコピーします。
ペースト	カットまたはコピーした資源をペーストします。
解除	プロジェクトファイルに登録されている資源を解除(削除)します。
全選択	登録されている全資源を選択します。
選択解除	すべての選択を解除します。

[表示]メニュー

全構成表示	登録されているすべての資源を表示します。
ターゲット表示	登録されているターゲットだけを表示します。
ソース表示	登録されているソースおよびそのファイルに依存するターゲットだけを表示します。
登録集表示	登録されている登録集およびそのファイルが依存する資源を表示します。
リソース表示	登録されているリソースおよびそのファイルが依存する資源を表示します。
登録	資源を登録する登録画面を表示します。
項目	資源の項目情報を設定または表示する項目画面を表示します。
状態	資源の状態を表示する状態画面を表示します。
パス	資源の格納されているパスを表示するパス画面を表示します。
詳細	資源の参照権や作成日付などの情報を表示する詳細画面を表示します。
メモ	資源のメモ情報を表示するメモ画面を表示します。

[作業]メニュー

設定	プロジェクトファイルや翻訳/リンクオプションの設定ツールを起動します。
依存調査	資源の依存関係を調査し、プロジェクト構成を追加/変更します。
解析	ソースプログラムの呼出し関係などが調査できるプログラム解析ツールを起動します。
ビルド	Makefileを生成しビルダを起動します。

エラー修正	翻訳エラーを管理するメッセージ管理ツールを起動します。
テスト	対話型デバッガをテスト進捗管理モードで起動します。
デバッグ	対話型デバッガを起動します。
性能	対話型デバッガを性能チューニングモードで起動します。
実行	実行環境を設定し、プログラムを実行します。
梱包	プロジェクトの資源を梱包します。

[ツール]メニュー

テキストエディタ	テキストエディタを起動します。
スクリーンデザイナ	スクリーン操作機能で使用する画面を設計/構築するスクリーンデザイナを起動します。
ソース解析	ソース解析ツールを起動します。
翻訳オプション設定	COBOLの翻訳オプションを設定します。
リンクオプション設定	リンクオプションを設定します。
ビルダ	ビルダツールを起動します。
メッセージ管理	メッセージ管理ツールを起動します。
COBOLデバッガ	COBOL対話型デバッガを起動します。
COBOLファイルユーティリティ	COBOLファイルユーティリティを起動します。

[オプション]メニュー

カスタマイズ	起動するツールのコマンドや起動パラメタの変更、拡張子に対応するツールの指定や実行時の環境変数などのカスタマイズ画面を表示します。
Makeルール編集	Makeルールを変更する変更画面を表示します。
ツールバー	ツールバーを表示または非表示します。
ステータスバー	ステータスバーを表示または非表示します。

22.2.2.2 プロジェクトマネージャのウィンドウおよびダイアログ・ボックス

プロジェクトマネージャ・ウィンドウ

メニュー・バー	プロジェクトマネージャには、次の七つのメニューがあります。[ファイル]、[編集]、[表示]、[作業]、[ツール]、[オプション]、および[ヘルプ]です。
ツール・バー	プロジェクトマネージャには、次の10のツールがあります。[設定]、[依存調査]、[解析]、[ビルド]、[エラー修正]、[テスト]、[デバッグ]、[性能]、[実行]、および[梱包]です。
全選択／選択解除・ボタン	プロジェクトを構成する資源の全選択または選択されている資源の選択を解除します。
プロジェクト情報・ウィンドウ	プロジェクトマネージャには、次の6つのプロジェクト情報ウィンドウがあります。[登録]、[項目]、[状態]、[パス]、[詳細]、または[メモ]です。プロジェクト情報ウィンドウはプロジェクト情報ウィンドウ切り替えラジオボタンによって切り替えることができます。
最新化・ボタン	プロジェクトマネージャの表示データを最新データに更新します。
メッセージ・ウィンドウ	プロジェクトマネージャ実行時に出力される作業状況のログやエラー状況を表示します。ポップアップメニューにより、表示されているログやエラー状況のデータを[クリア]、[コピー]、または[ログ出力]することができます。
ステータス・バー	カーソル位置に応じて、実行可能な作業や表示されているデータの説明などをメッセージで表示します。

[設定ダイアログ]ダイアログ・ボックス

プロジェクト

プロジェクト名	プロジェクトの名前を指定します。この項目は省略できません。
プロジェクトメモ	プロジェクトに関するコメントを記述します。
ベースディレクトリ	プロジェクトのベースディレクトリを指定します。この項目は省略できません。

ユーザ資源

ソース	ソースプログラムが格納されているディレクトリを指定します。
登録集	登録集が格納されているディレクトリを指定します。
画面帳票定義体	画面帳票定義体が格納されているディレクトリを指定します。
ファイル定義体	ファイル定義体が格納されているディレクトリを指定します。
依存リポジトリ	依存対象となるリポジトリファイルが格納されているディレクトリを指定します。

開発資源

オブジェクト	オブジェクトを格納するディレクトリを指定します。
デバッグ情報	デバッグ情報を格納するディレクトリを指定します。
デバッグ関連	デバッグ時にデバッガやランタイムが出力する各資源を格納するディレクトリを指定します。
ターゲットリポジトリ	リポジトリを格納するディレクトリを指定します。
翻訳リスト	翻訳リストを格納するディレクトリを指定します。

環境変数

環境変数設定状況ウィンドウ	設定されている環境変数を表示します。[追加]ボタンを選択することにより、環境変数名を追加することができます。また、オプションメニューには[現在値の読込み]、[カーソル位置に読込み]、または[ファイルに出力]があります。
環境変数ウィンドウ	設定する環境変数を指定します。一覧表示されている環境変数を選択（ダブルクリック）するとその入力データとして引用できます。
ディレクトリ名追加	設定する環境変数にディレクトリ名を指定します。
ファイル名追加	設定する環境変数にファイル名を指定します。

デバッグ用環境変数

環境変数設定状況ウィンドウ	デバッグおよびテスト時に使用する環境変数に設定されている環境変数を表示します。[追加]ボタンを選択することにより、環境変数名を追加することができます。また、オプションメニューには[現在値の読込み]、[カーソル位置に読込み]、または[ファイルに出力]があります。
環境変数ウィンドウ	デバッグおよびテスト時に使用する環境変数を指定します。一覧表示されている環境変数を選択（ダブルクリック）するとその入力データとして引用できます。
ディレクトリ名追加	設定する環境変数にディレクトリ名を指定します。
ファイル名追加	設定する環境変数にファイル名を指定します。

[プロジェクトファイルの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。

選択	選択するファイル名を指定します。

[ベースディレクトリの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ディレクトリ	フィルタの指定に従って、現在のディレクトリに存在するディレクトリを表示します。
選択	選択するディレクトリ名を指定します。

[ソースディレクトリの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ディレクトリ	フィルタの指定に従って、現在のディレクトリに存在するディレクトリを表示します。
選択	選択するディレクトリ名を指定します。

[登録集ディレクトリの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ディレクトリ	フィルタの指定に従って、現在のディレクトリに存在するディレクトリを表示します。
選択	選択するディレクトリ名を指定します。

[画面定義体ディレクトリの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ディレクトリ	フィルタの指定に従って、現在のディレクトリに存在するディレクトリを表示します。
選択	選択するディレクトリ名を指定します。

[ファイル定義体ディレクトリの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ディレクトリ	フィルタの指定に従って、現在のディレクトリに存在するディレクトリを表示します。
選択	選択するディレクトリ名を指定します。

[依存リポジトリディレクトリの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ディレクトリ	フィルタの指定に従って、現在のディレクトリに存在するディレクトリを表示します。

選択	選択するディレクトリ名を指定します。

[オブジェクトディレクトリの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ディレクトリ	フィルタの指定に従って、現在のディレクトリに存在するディレクトリを表示します。
選択	選択するディレクトリ名を指定します。

[デバッグ情報ディレクトリの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ディレクトリ	フィルタの指定に従って、現在のディレクトリに存在するディレクトリを表示します。
選択	選択するディレクトリ名を指定します。

[デバッグ関連ディレクトリの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ディレクトリ	フィルタの指定に従って、現在のディレクトリに存在するディレクトリを表示します。
選択	選択するディレクトリ名を指定します。

[ターゲットリポジトリディレクトリの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ディレクトリ	フィルタの指定に従って、現在のディレクトリに存在するディレクトリを表示します。
選択	選択するディレクトリ名を指定します。

[翻訳リストディレクトリの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ディレクトリ	フィルタの指定に従って、現在のディレクトリに存在するディレクトリを表示します。
選択	選択するディレクトリ名を指定します。

[環境変数設定ファイルの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。

選択	カーソル位置に読み込む環境変数設定ファイル名を指定します。

[環境変数出力ファイルの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	保存する環境変数設定ファイル名を指定します。

[追加ディレクトリの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ディレクトリ	フィルタの指定に従って、現在のディレクトリに存在するディレクトリを表示します。
選択	追加するディレクトリ名を指定します。

[追加ファイルの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	追加するファイル名を指定します。

[プロジェクトを開く]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	オープンするファイル名を指定します。

[プロジェクトを別名で保存]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	保存するファイル名を指定します。

[登録ダイアログ]ダイアログ・ボックス

依存調査による自動登録	登録するソースの依存関係を自動的に調査する場合に指定します。
ファイル・フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。

ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
追加登録するファイル名	登録するファイル名を指定します。
ターゲット	実行形式プログラム、共用オブジェクト(.so)、ターゲットリポジトリを登録します。
ソース	ソースプログラム、オブジェクトを登録します。
登録集	登録集原文、画面帳票定義体、ファイル定義体、依存リポジトリを登録します。
リソース	画面定義情報ファイルを登録します。
その他	上記以外の仕様書やCOBOLファイルなどの資源を登録します。

[登録]画面

依存調査による自動登録	登録するソースの依存関係を自動的に調査する場合に指定します。
フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
追加登録するファイル名	登録するファイル名を指定します。
ターゲット	実行形式プログラム、共用オブジェクト(.so)、ターゲットリポジトリを登録します。
ソース	ソースプログラム、オブジェクトを登録します。
登録集	登録集原文、画面帳票定義体、ファイル定義体、依存リポジトリを登録します。
リソース	画面定義情報ファイルを登録します。
その他	上記以外の仕様書やCOBOLファイルなどの資源を登録します。

[項目] 画面

資源ファイル名	ファイル名を指定します。
状態	資源の状態をメッセージで表示します。
詳細	資源の更新日付、プロテクト情報や所有者などの情報を表示します。
メモ	資源に関するコメントなどを指定します。
プレビュー	各資源の内容をプレビューします。
更新	資源ファイル名、メモのデータを更新します。
解除	選択した資源を資源一覧から削除します。

[状態] 画面

状態	資源の種別、状態、関連する資源などの情報を表示します。

[ファイル名] 画面

ファイル名	資源のファイル名を表示します。

[詳細] 画面

詳細	資源の更新日付、プロテクト情報、所有者などの情報を表示します。

[メモ] 画面

メモ	資源のコメントを指定します。

[項目資源パスの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	変更する項目資源のファイル名を指定します。

[カスタマイズ]ダイアログ・ボックス

全般

Makeルールファイル指定	Makeファイル生成時に使用する、Makeルールのファイル名を指定します。
Makeルール編集コマンド指定	Makeルールファイルの編集に使用するコマンド(ツール)を指定します。
翻訳オプション編集コマンド指定	翻訳オプションを指定する翻訳オプションファイルを編集するコマンド(ツール)を指定します。
リンクオプション編集コマンド指定	リンクオプションを指定するリンクオプションファイルを編集するコマンド(ツール)を指定します。
ターミナル指定1	ターミナルを起動する際の起動オプションを指定します。デフォルトの設定では、ターミナル1はコマンド実行後にターミナルのウィンドウはそのまま残ります。
ターミナル指定2	ターミナルを起動する際の起動オプションを指定します。デフォルトの設定では、ターミナル2はコマンド実行後にターミナルのウィンドウは残りません。
自動的に最新化する間隔の指定	ウィンドウを再描画し、最新情報を表示する時間を指定します。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

ツール

登録済一覧	登録されているツールの一覧です。登録されているツールの削除または登録順序を入れ替える場合に使用します。
メニュー情報	登録済一覧に登録されているツールの詳細情報を表示します。また、ツールを登録または登録した情報の変更に使用します。
ラベル	[ツール]メニューのラベル名を指定します。
ニーモニック	ニーモニックキーを指定します。
コマンド行	メニュー選択時に実行するコマンドを指定します。
コマンド種別	コマンドの起動種別を指定します。
ステータス表示	ステータスバーに表示するメッセージを指定します。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

拡張子

登録済一覧	登録されている拡張子の一覧です。登録されている拡張子の削除または登録順序を入れ替える場合に使用します。
拡張子情報	登録済一覧に登録されている拡張子の詳細情報を表示します。また、拡張子を登録または登録した情報の変更に使用します。

拡張子	拡張子を指定します。
資源名称	拡張子に対応する資源の名称を指定します。
ポップアップメニュー	登録されているポップアップメニューの一覧です。登録されているポップアップメニューの削除または登録順序を入れ替える場合に使用します。
ラベル	ポップアップメニューに表示するラベルを指定します。
コマンド行	ポップアップメニュー選択時に実行するコマンドを指定します。
コマンド種別	コマンドの起動種別を指定します。
ステータス表示	ステータスバーに表示するメッセージを指定します。
デフォルト動作として設定	該当する拡張子を持つファイル名をダブルクリックした場合に関連付ける動作かどうかを指定します。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

環境変数

登録済一覧	登録されている環境変数の一覧です。登録されている環境変数の削除または登録順序を入れ替える場合に使用します。
環境変数テンプレート設定	登録済一覧に登録されている環境変数の詳細情報を表示します。また、環境変数を登録または登録した情報の変更に使用します。
環境変数テンプレート	環境変数のテンプレートを指定します。“環境変数名＝値”の形式で指定します。
ステータス表示	ステータスバーに表示するメッセージを指定します。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

[全般・Makeルールファイル指定]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[全般・Makeルール編集コマンド指定]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[全般・翻訳オプション編集コマンド指定]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[全般・リンクオプション編集コマンド指定]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[全般・ターミナル指定1]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[全般・ターミナル指定2]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[ツールメニュー・コマンドファイルの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[ポップアップメニュー・コマンドファイルの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[パラメタ入力]ダイアログ・ボックス

パラメタ文字列	コマンドに渡すパラメタを指定します。

[ログ出力ファイルの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。

ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

22.2.3 カスタマイズ変数一覧

カスタマイズのコマンド指定で使える変数

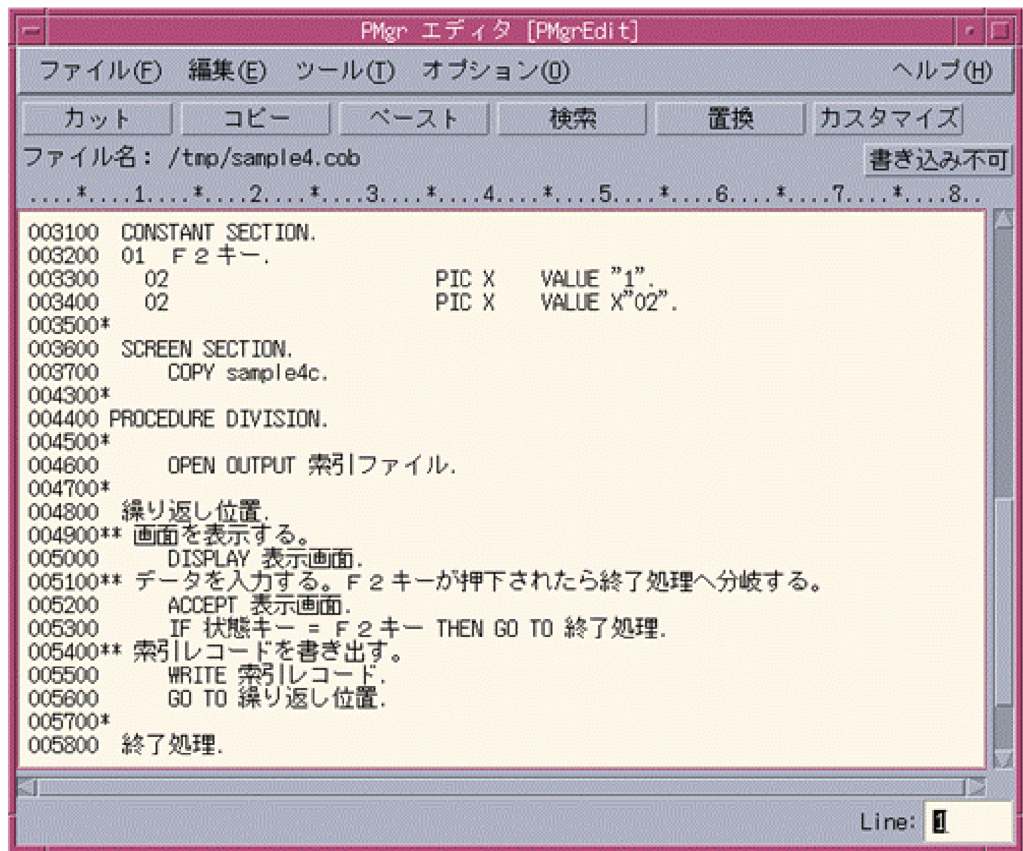
%COBPM_CWD%	ベースディレクトリ
%COBPM_PRJ%	プロジェクトパス
%COBPM_PNAME%	プロジェクト名(拡張子なし)
%COBPM_FILE%	選択ファイルパス
%COBPM_NAME%	選択ファイル名(拡張子なし)
%COBPM_MEMO%	選択ファイルのメモ

プロジェクトマネージャでのシェルで使える変数

COBPM_CWD	ベースディレクトリ
COBPM_PRJ	プロジェクトパス
COBPM_EXE	先頭ターゲット
COBPM_SELECT	選択状態(ALL/SINGLE/MULTI/NON)
COBPM_FILE	選択ファイルのパス
COBPM_MEMO	選択ファイルのメモ
COBPM_SRCDIR	ソースプログラム検索ディレクトリ
COBPM_COPYDIR	登録集検索ディレクトリ
COBPM_SMEDDIR	画面帳票定義体検索ディレクトリ
COBPM_FFDDIR	ファイル定義体検索ディレクトリ
COBPM_REPINDIR	リポジトリ検索ディレクトリ
COBPM_OBJDIR	オブジェクト格納ディレクトリ
COBPM_SVDDIR	デバッグ情報ファイル格納ディレクトリ
COBPM_DBGDIR	デバッグ関連ファイル格納ディレクトリ
COBPM_REPDIR	リポジトリ格納ディレクトリ
COBPM_LISTDIR	リスト格納ディレクトリ

22.3 エディタ

エディタは、ソースプログラムなどのテキストデータを作成/編集するためのユーティリティであり、Xウィンドウ上で動作します。



エディタを開くには

以下のコマンドにより、エディタを起動することができます。

```
pmgredit [ [-num 行番号] ファイル名]
```

-num 行番号

カーソルを位置付ける行番号を指定します。

ファイル名

編集するテキストデータのファイル名を指定します。

22.3.1 エディタの使い方

エディタはプロジェクトマネージャの各ツールとの連携機能を強化した簡易テキストエディタです。

ここではエディタの使い方について説明します。

22.3.1.1 テキストファイルの新規作成および編集

新規作成するには

[ファイル] メニューの[新規作成]を選択します。

既存のテキストファイルを開くには

1. [ファイル] メニューの[オープン]を選択します。
2. 保存しているテキストファイルを選択します。

22.3.1.2 ツール連携

ソースプログラムの翻訳オプション設定

ソースプログラムに対する翻訳オプションの設定が可能です。

ソースプログラムの解析機能

ソースプログラム中の任意の文字列を検索することによりプログラム解析作業を補助します。プログラム名やデータ名などの検索も可能なので、コーディング作業時やプログラムの分析や修正による影響調査に威力を発揮します。

なお、詳細についてはソース解析ツールのマニュアルまたはヘルプを参照してください。

22.3.2 エディタのリファレンス

22.3.2.1 エディタのメニュー

メニュー･アクセラレータ

カット	[Ctrl] + [X]キー
コピー	[Ctrl] + [C]キー
ペースト	[Ctrl] + [V]キー
終了	[Alt] + [F4] キー

[ファイル]メニュー

新規作成	ファイルを新規に作成します。
オープン	既存のファイルをオープンします。
カーソル位置に読込み	カーソル位置にファイルの内容を挿入します。
保存	ファイルを上書きして保存します。
別名保存	ファイルを変名して保存します。
印刷	ファイルを印刷します。
終了	エディタを終了します。

[編集]メニュー

カット	選択されたデータをカットします。
コピー	選択されたデータをコピーします。
ペースト	カットまたはコピーしたデータをペーストします。
削除	選択されたデータを削除します。
検索	文字列を検索します。
置換	文字列を置換します。

[ツール]メニュー

エディタ	新規にエディタを起動します。
COBOL翻訳オプション設定	翻訳オプションを設定します。
ソース解析	ソースプログラム解析ツールを起動します。

[オプション]メニュー

カスタマイズ	ツールの起動を変更するカスタマイズ画面を表示します。

ツールバー	ツールバーを表示または非表示します。
ルーラ	ルーラを表示または非表示します。
ステータスバー	ステータスバーを表示または非表示します。

22.3.2.2 エディタのウィンドウおよびダイアログ・ボックス

エディタ・ウィンドウ

メニュー・バー	エディタには、次の5つのメニューがあります。[ファイル]、[編集]、[ツール]、[オプション]、および[ヘルプ]です。
ツール・バー	ツールバーには、次の6つの機能が登録されています。[カット]、[コピー]、[ペースト]、[検索]、[置換]、および[カスタマイズ]です。
変更なし／書込不可／保存／別名保存	現在作業中のファイルの保存や書込み権に関するパーミッションの変更を行います。
テキスト・ウィンドウ	テキストデータを入力するウィンドウです。
ステータス・バー	カーソル位置に応じて、実行可能な作業や表示されているデータの説明などをメッセージで表示します。また、テキスト・ウィンドウ内の行番号も表示します。

[オープン]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[カーソル位置に読込み]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
プレビュー表示	読み込むファイルの内容をプレビューします。
選択	選択するファイル名を指定します。

[別名保存]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[検索]ダイアログ・ボックス

検索文字列	検索する文字列を指定します。
大文字小文字を区別する	検索する文字列を大文字と小文字で区別して検索する場合に指定します。
前検索	カーソル位置よりも前に存在する文字列を検索します。

次検索	カーソル位置よりも後ろに存在する文字列を検索します。

[置換]ダイアログ・ボックス

検索文字列	検索する文字列を指定します。
大文字小文字を区別する	検索する文字列を大文字と小文字で区別して検索する場合に指定します。
置換文字列	置換する文字列を指定します。
前検索	カーソル位置よりも前に存在する文字列を検索します。
置換前	文字列を置換し、カーソル位置よりも前に存在する次の文字列を検索します。
置換次	文字列を置換し、カーソル位置よりも後ろに存在する次の文字列を検索します。
次検索	カーソル位置よりも後ろに存在する文字列を検索します。
全置換	検索文字列に該当するすべての文字列を置換文字列に置換します。

[カスタマイズ]ダイアログ・ボックス

全般

バックアップファイルの作成	バックアップファイルを作成するかどうかを指定します。
印刷用コマンド	印刷コマンドを指定します。
ルーラ文字列	テキスト・ウィンドウのルーラの表示データを指定します。
カーソル位置への読込みでのデフォルト設定	カーソル位置にデータを読み込む場合の検索ディレクトリを指定します。プロジェクトのベースディレクトリまたはテンプレートの格納先ディレクトリを選択することができます。
テンプレートの格納先指定	テンプレートの格納ディレクトリを指定します。
ターミナル指定1	ターミナルを起動する際の起動オプションを指定します。デフォルトの設定では、ターミナル1はコマンド実行後にターミナルのウィンドウはそのまま残ります。
ターミナル指定2	ターミナルを起動する際の起動オプションを指定します。デフォルトの設定では、ターミナル2はコマンド実行後にターミナルのウィンドウは残りません。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

ツール

登録済一覧	[ツール]メニューに登録されているツールを表示します。
追加	[ツール]メニューにツールを登録します。
上へ	[ツール]メニューの表示順序を上に変更します。
下へ	[ツール]メニューの表示順序を下に変更します。
削除	[ツール]メニューからツールを削除します。
ラベル	[ツール]メニューの表示するラベルを指定します。
ニーモニック	ツールのニーモニックを指定します。
コマンド行	ツールの起動コマンドを指定します。
コマンド種別	ツールのコマンドを実行する種別を指定します。
ステータス表示	ステータスバーに表示するメッセージを指定します。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

[全般・印刷用コマンドの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するコマンド名を指定します。

[全般・読込みデフォルトディレクトリの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するディレクトリ名を指定します。

[全般・ターミナル指定1]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[全般・ターミナル指定2]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[ツールメニュー・コマンドファイルの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するコマンド名を指定します。

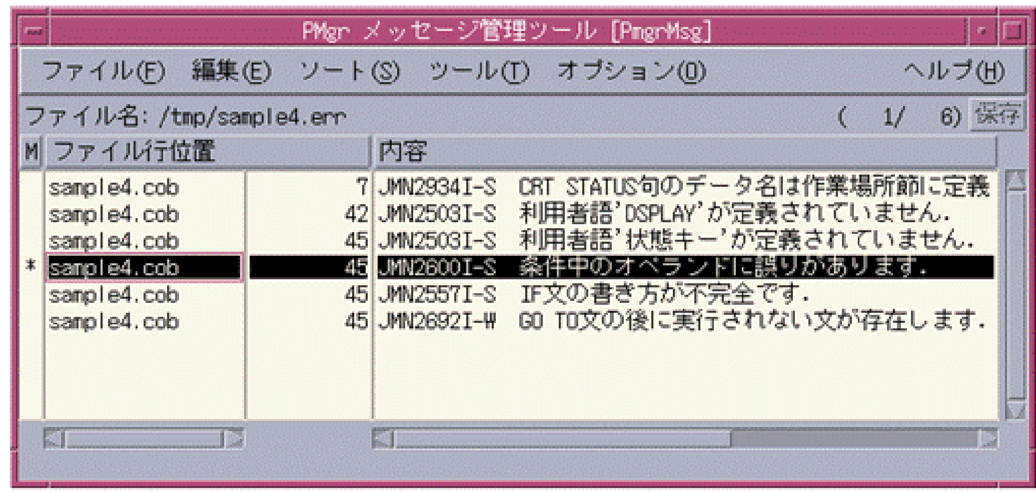
22.3.3 カスタマイズ変数一覧

カスタマイズのコマンド指定で使える変数

%COBPM_FILE%	ファイルパス
%COBPM_NAME%	ファイル名(拡張子なし)
%COBPM_NUM%	カーソル行位置

22.4 メッセージ管理ツール

メッセージ管理ツールは、翻訳時のエラーメッセージやソースプログラム解析ツールのメッセージなどのメッセージを管理するためのユーティリティであり、Xウィンドウ上で動作します。



メッセージ管理ツールを開くには

以下のコマンドにより、メッセージ管理ツールを起動することができます。

```
pmgrmsg [メッセージ構文指定] [ファイル名]
```

メッセージ構文指定

管理するメッセージの種別を指定します。

- -mc "パス 行番号 内容"
- -mr "パス:行番号:内容"
- -me "パス 行番号:JMNxxx内容"

ファイル名

メッセージファイル名を指定します。

22.4.1 メッセージ管理ツールの使い方

メッセージ管理ツールは翻訳エラーメッセージや各ツールの出力するメッセージを管理するツールです。このツールを使用することにより、メッセージに対応するタグジャンプ機能や必要なメッセージだけを収集することができます。

ここではメッセージ管理ツールの使い方について説明します。

22.4.1.1 メッセージファイルの新規作成および編集

新規作成するには

[ファイル] メニューの[新規作成]を選択します。

既存のメッセージファイルを開くには

1. [ファイル] メニューの[オープン]を選択します。
2. 保存しているメッセージファイルを選択します。

メッセージファイルにメッセージを追加するには

1. [ファイル] メニューの[カーソル位置に読込み]を選択します。

2. 追加するメッセージファイルを選択します。

22.4.1.2 メッセージの管理

メッセージにマークを付けるには

[編集] メニューの[マークをつける]を選択します。

なお、各メッセージに付けたマークの情報は保存されません。

メッセージに付けたマークを外すには

[編集] メニューの[マークをはずす]を選択します。

メッセージを検索するには

1. [編集] メニューの[検索]を選択します。
2. 文字列を指定して検索します。

メッセージをソートするには

マークでソートするには

[ソート] メニューの[マークでソート]を選択します。

ファイルの行位置でソートするには

[ソート] メニューの[ファイル行位置でソート]を選択します。

メッセージの内容でソートするには

[ソート] メニューの[内容でソート]を選択します。

22.4.1.3 ツール連携

エディタ

エディタと連携することにより、メッセージの内容を修正することができます。

22.4.2 メッセージ管理ツールのリファレンス

22.4.2.1 メッセージ管理ツールのメニュー

メニュー・アクセラレータ

カット	[Ctrl] + [X]キー
コピー	[Ctrl] + [C]キー
ペースト	[Ctrl] + [V]キー
終了	[Alt] + [F4] キー

[ファイル]メニュー

新規作成	メッセージファイルを新規に作成します。
オープン	既存のメッセージファイルをオープンします。
カーソル位置に読込み	カーソル位置にメッセージを追加します。
保存	メッセージファイルを上書きして保存します。
別名保存	メッセージファイルを変名して保存します。
終了	メッセージ管理ツールを終了します。

[編集]メニュー

マークをつける	選択されたメッセージにマークを付けます。
マークをはずす	選択されたメッセージのマークを外します。
カット	選択されたメッセージをカットします。
コピー	選択されたメッセージをコピーします。
ペースト	カットまたはコピーしたメッセージをペーストします。
削除	選択されたメッセージを削除します。
検索	メッセージを検索します。
全選択	すべてのメッセージを選択します。
マークのある項目の選択	マークの付いているメッセージを選択します。
マークのない項目の選択	マークの付いていないメッセージを選択します。
全解除	すべてのメッセージの選択を解除します。

[ソート]メニュー

マークでソート	マークの有無でメッセージをソートします。
ファイル行位置でソート	ファイルの行位置でソートします。
内容でソート	メッセージでソートします。

[ツール]メニュー

エディタ	エディタを起動します。
メッセージ管理	メッセージ管理ツールを起動します。
メッセージ編集	メッセージを編集するエディタを起動します。

[オプション]メニュー

カスタマイズ	ツールの動作を変更するカスタマイズ画面を表示します。
ステータスバー	ステータスバーを表示または非表示します。

22.4.2.2 メッセージ管理ツールのウィンドウおよびダイアログ・ボックス

メッセージ管理・ウィンドウ

メニュー・バー	メッセージ管理ツールには、次の6つのメニューがあります。[ファイル]、[編集]、[ソート]、[ツール]、[オプション]、および[ヘルプ]です。
メッセージ表示ウィンドウ	管理しているメッセージを表示します。
ステータス・バー	カーソル位置に応じて、実行可能な作業や表示されているデータの説明などをメッセージで表示します。

[オープン]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。

選択	オープンするファイル名を指定します。

[カーソル位置に読込み]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	オープンするファイル名を指定します。

[別名保存]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	保存するファイル名を指定します。

[検索]ダイアログ・ボックス

検索文字列	検索する文字列を指定します。
大文字小文字を区別する	検索する文字列を大文字と小文字で区別して検索する場合に指定します。
前検索	カーソル位置よりも前に存在する文字列を検索します。
次検索	カーソル位置よりも後ろに存在する文字列を検索します。
該当項目の全選択	内容に検索文字列を含むメッセージを全選択します。

[カスタマイズ]ダイアログ・ボックス

全般

メッセージ構文	管理対象のメッセージ構文の形式を指定します。形式には [パス 行番号 内容] [パス:行番号:内容] [パス 行番号:JMNxxx内容]があります。
ダブルクリックアクション	メッセージをダブルクリックした場合に実行するコマンドを指定します。
ダブルクリック時のマーキング	ダブルクリックアクションを実行後、メッセージにマークを付けるかどうかを指定します。
ターミナル指定1	ターミナルを起動する際の起動オプションを指定します。デフォルトの設定では、ターミナル1はコマンド実行後にターミナルのウィンドウはそのまま残ります。
ターミナル指定2	ターミナルを起動する際の起動オプションを指定します。デフォルトの設定では、ターミナル2はコマンド実行後にターミナルのウィンドウは残りません。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

ツール

登録済一覧	[ツール]メニューに登録されているツールを表示します。
追加	[ツール]メニューにツールを登録します。
上へ	[ツール]メニューの表示順序を上に変更します。
下へ	[ツール]メニューの表示順序を下に変更します。
削除	[ツール]メニューからツールを削除します。

ラベル	[ツール]メニューの表示するラベルを指定します。
ニーモニック	ツールのニーモニックを指定します。
コマンド行	ツールの起動コマンドを指定します。
コマンド種別	ツールのコマンドを実行する種別を指定します。
ステータス表示	ステータスバーに表示するメッセージを指定します。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

[全般･ダブルクリックアクション]ダイアログ･ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するコマンド名を指定します。

[全般･ターミナル指定1]ダイアログ･ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[全般･ターミナル指定2]ダイアログ･ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[ツールメニュー･コマンドファイルの選択]ダイアログ･ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するコマンド名を指定します。

22.4.3 カスタマイズ変数一覧

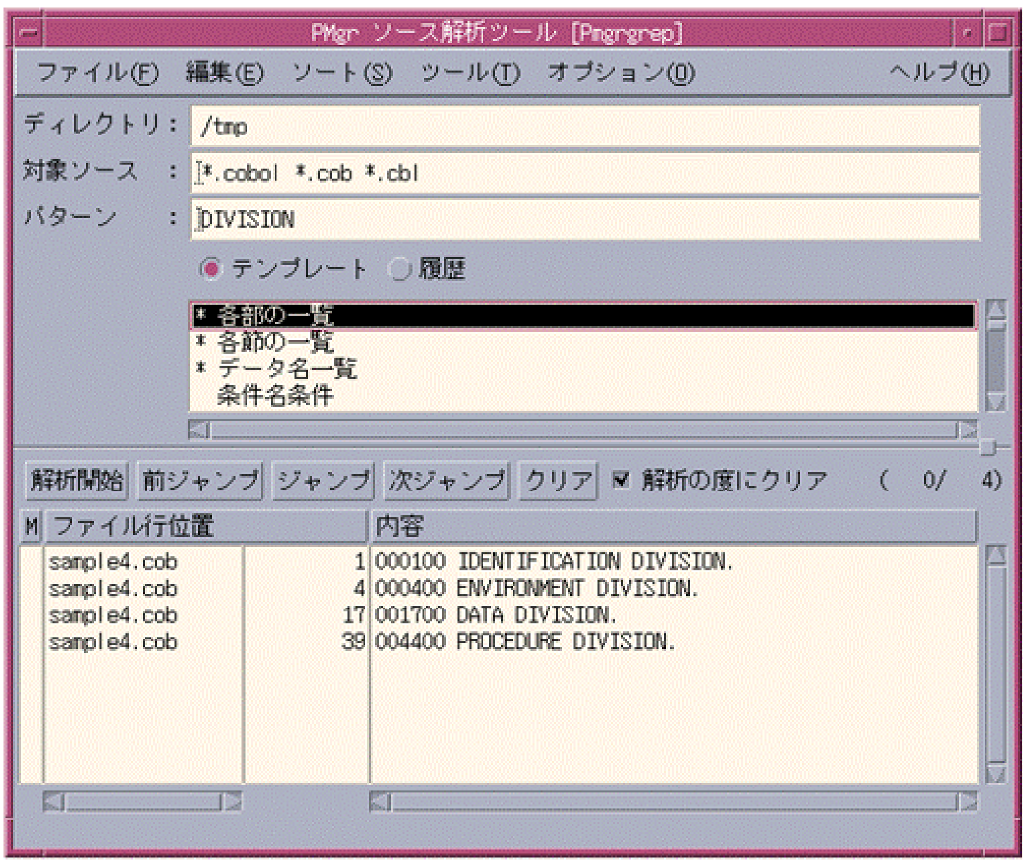
カスタマイズのコマンド指定で使える変数

%COBPM_MSGFILE%	メッセージファイルパス
%COBPM_MSGNAME%	メッセージファイル名(拡張子なし)
%COBPM_MSGPOS%	メッセージのカーソル行位置

%COBPM_FILE%	選択メッセージファイルパス
%COBPM_NAME%	選択メッセージファイル名(拡張子なし)
%COBPM_NUM%	選択メッセージ行
%COBPM_MSG%	選択メッセージ内容

22.5 ソース解析ツール

ソース解析ツールは、ソースプログラム中のデータ名や呼出し関係などの情報をgrepコマンドの機能を利用して検索することにより、ソースプログラムを解析するためのユーティリティです。ソース解析ツールはXウィンドウ上で動作します。



ソース解析ツールを開くには

以下のコマンドにより、ソース解析ツールを起動することができます。

```
pmgrgrep [メッセージ構文指定] [-grp] [-dr ディレクトリ] [-e パターンリスト]… [-f パターンファイル名]… [ファイル名…]
```

```
pmgrgrep [メッセージ構文指定] [-grp] [-dr ディレクトリ] パターン [ファイル名…]
```

メッセージ構文指定

管理するメッセージの種別を指定します。

- -mc "パス 行番号 内容"
- -mr "パス:行番号:内容"
- -me "パス 行番号:JMNxxx内容"

-grp

起動と同時に検索を行います。

-dr ディレクトリ

検索対象のディレクトリ名を指定します。

-e パターンリスト

検索パターンリストを指定します。

-f パターンファイル名

検索パターンファイル名を指定します。

パターン

検索対象のパターンを指定します。

ファイル名

検索対象のファイル名を指定します。正規表現を使用する場合はファイル名をシングルクォート(‘)で囲んで指定します。

22.5.1 ソース解析ツールの使い方

ソース解析ツールは、ソースプログラム中のデータ名やプログラム名をgrepコマンドの機能を利用して検索することにより、ソースプログラムの解析を行うツールです。このツールでは解析メッセージに対応するソースプログラムをダイレクトに参照できるタグジャンプ機能や解析メッセージの管理機能を利用することができます。

ここではソース解析ツールの使い方について説明します。

22.5.1.1 ソースプログラムの解析

ソースプログラムを解析するには

1. ソース解析ツール画面のディレクトリに検索対象ソースプログラムの格納ディレクトリを指定します。
2. ソース解析ツール画面の対象ソースにソースプログラムのファイル名(総称名指定可能)を指定します。複数指定する場合は、空白を1文字挿入し、ファイル名を指定します。
3. ソース解析ツール画面のパターンに検索パターンを指定します。検索パターンはテンプレートから選択することもできます。
4. [ファイル] メニューの[解析開始]を選択します。

解析内容を参照(タグジャンプ)するには

ソース解析ツール画面の検索メッセージを選択(ダブルクリック)します。

22.5.1.2 解析結果の管理

メッセージにマークを付けるには

[編集] メニューの[マークをつける]を選択します。

メッセージに付けたマークを外すには

[編集] メニューの[マークをはずす]を選択します。

メッセージを検索するには

1. [編集] メニューの[検索]を選択します。
2. 文字列を指定して検索します。

メッセージをソートするには

マークでソートするには

[ソート] メニューの[マークでソート]を選択します。

ファイルの行位置でソートするには

[ソート] メニューの[ファイル行位置でソート]を選択します。

メッセージの内容でソートするには

[ソート] メニューの[内容でソート]を選択します。

22.5.1.3 ツール連携

エディタ

エディタと連携することにより、メッセージの示す個所を直接参照することができます。

メッセージ管理ツール

メッセージ管理ツールと連携することにより解析結果メッセージを管理することができます。

22.5.2 ソース解析ツールのリファレンス

22.5.2.1 ソース解析ツールのメニュー

メニュー・アクセラレータ

カット	[Ctrl] + [X]キー
コピー	[Ctrl] + [C]キー
ペースト	[Ctrl] + [V]キー
終了	[Alt] + [F4] キー

[ファイル]メニュー

解析ディレクトリ指定	解析するソースプログラムが格納されているディレクトリを指定します。
解析開始	解析を開始します。
結果保存	解析結果を保存します。
終了	ソース解析ツールを終了します。

[編集]メニュー

マークをつける	選択されたメッセージにマークを付けます。
マークをはずす	選択されたメッセージのマークを外します。
カット	選択されたメッセージをカットします。
コピー	選択されたメッセージをコピーします。
ペースト	カットまたはコピーしたメッセージをペーストします。
削除	選択されたメッセージを削除します。
検索	メッセージを検索します。
全選択	すべてのメッセージを選択します。
マークのある項目の選択	マークの付いているメッセージを選択します。
マークのない項目の選択	マークの付いていないメッセージを選択します。
全解除	すべてのメッセージの選択を解除します。

[ソート]メニュー

マークでソート	マークの有無でメッセージをソートします。
ファイル行位置でソート	ファイルの行位置でソートします。
内容でソート	メッセージでソートします。

[ツール]メニュー

エディタ	エディタを起動します。
メッセージ管理	メッセージ管理ツールを起動します。
ソース解析	ソース解析ツールを起動します。

[オプション]メニュー

カスタマイズ	ツールの動作を変更するカスタマイズ画面を表示します。
ステータスバー	ステータスバーを表示または非表示します。

22.5.2.2 ソース解析ツールのウィンドウおよびダイアログ・ボックス

ソース解析・ウィンドウ

メニュー・バー	ソース解析ツールには、次の6つのメニューがあります。[解析]、[編集]、[ソート]、[ツール]、[オプション]、および[ヘルプ]です。
ディレクトリ	検索ディレクトリを指定します。
対象ソース	検索対象のソースファイル名を指定します。アスタリスク(*)やワイルドカード文字を使用して指定することもできます。
パターン	検索パターンを指定します。パターンを複数指定する場合には空白で区切って指定します。また、[テンプレート]や[履歴]でパターンを指定することもできます。
ツール・バー	ソース解析ツールには、次の5つのツールがあります。[解析開始]、[前ジャンプ]、[ジャンプ]、[次ジャンプ]、および[クリア]です。
解析の度にクリア	解析の度にメッセージ表示ウィンドウのメッセージをクリアするかどうかを指定します。
メッセージ表示ウィンドウ	検索結果メッセージを表示します。
ステータス・バー	カーソル位置に応じて、実行可能な作業や表示されているデータの説明などをメッセージで表示します。

[解析ディレクトリ指定]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
解析ディレクトリ指定	解析するディレクトリを指定します。

[結果保存]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	保存するファイル名を指定します。

[検索]ダイアログ・ボックス

検索文字列	検索する文字列を指定します。
大文字小文字を区別する	検索する文字列を大文字と小文字で区別して検索する場合に指定します。
前検索	カーソル位置よりも前に存在する文字列を検索します。
次検索	カーソル位置よりも後ろに存在する文字列を検索します。
該当項目の全選択	内容に検索文字列を含むメッセージを全選択します。

[カスタマイズ]ダイアログ・ボックス

全般

解析コマンド設定	解析コマンドを指定します。
対象ソースのデフォルト指定	対象ソースのデフォルト値を指定します。
メッセージ構文	管理対象のメッセージ構文の形式を指定します。形式には [パス 行番号 内容] [パス:行番号:内容] [パス 行番号:JMNxxx内容]があります。
ダブルクリックアクション	メッセージをダブルクリックした場合に実行するコマンドを指定します。
ダブルクリック時のマーキング	ダブルクリックアクションを実行後、メッセージにマークを付けるかどうかを指定します。
ターミナル指定1	ターミナルを起動する際の起動オプションを指定します。デフォルトの設定では、ターミナル1はコマンド実行後にターミナルのウィンドウはそのまま残ります。
ターミナル指定2	ターミナルを起動する際の起動オプションを指定します。デフォルトの設定では、ターミナル2はコマンド実行後にターミナルのウィンドウは残りません。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

ツール

登録済一覧	[ツール]メニューに登録されているツールを表示します。
追加	[ツール]メニューにツールを登録します。
上へ	[ツール]メニューの表示順序を上に変更します。
下へ	[ツール]メニューの表示順序を下に変更します。
削除	[ツール]メニューからツールを削除します。
ラベル	[ツール]メニューの表示するラベルを指定します。
ニーモニック	ツールのニーモニックを指定します。
コマンド行	ツールの起動コマンドを指定します。
コマンド種別	ツールのコマンドを実行する種別を指定します。
ステータス表示	ステータスバーに表示するメッセージを指定します。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

解析テンプレート

登録済一覧	[テンプレート]に登録されているテンプレートを表示します。
追加	[テンプレート]にテンプレートを登録します。
上へ	[テンプレート]の表示順序を上に変更します。
下へ	[テンプレート]の表示順序を下に変更します。

削除	[テンプレート]からテンプレートを削除します。
解析テンプレート名	解析テンプレート名を指定します。
解析パターン	解析パターンを指定します。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

[全般・ダブルクリックアクション]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するコマンド名を指定します。

[全般・ターミナル指定1]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[全般・ターミナル指定2]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[ツールメニュー・コマンドファイルの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するコマンド名を指定します。

22.5.3 カスタマイズ変数一覧

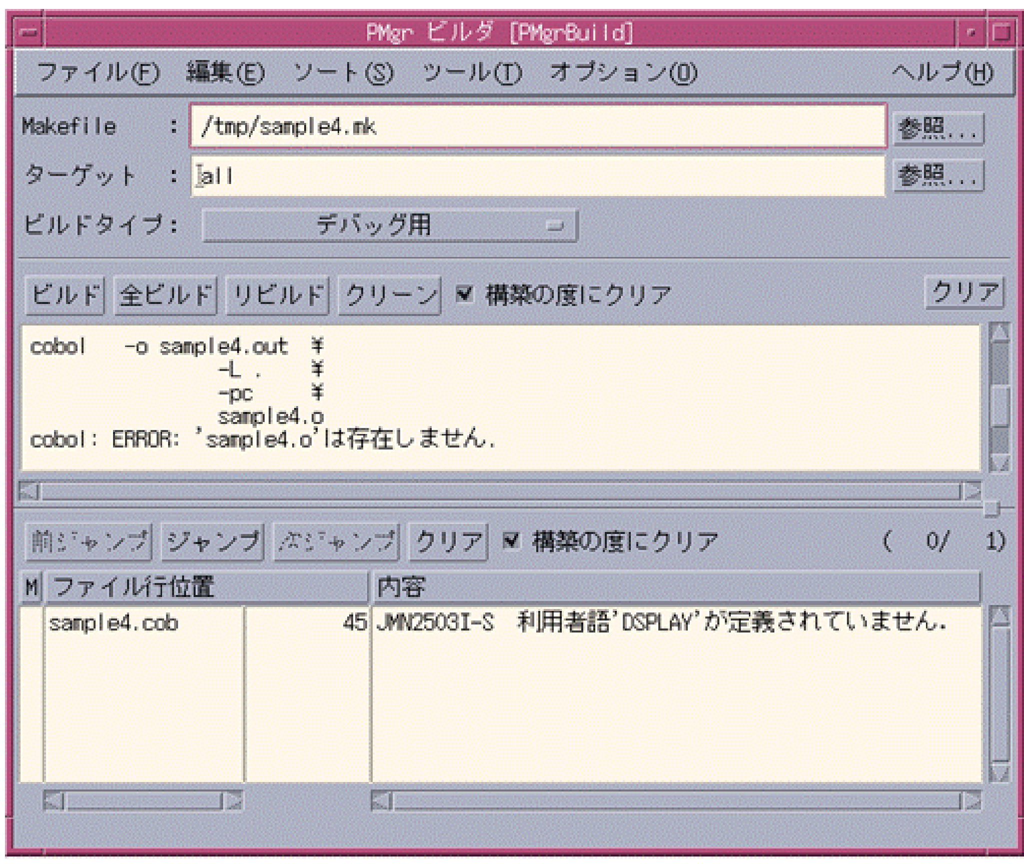
カスタマイズのコマンド指定で使える変数

%COBPM_CWD%	解析ディレクトリ(カレントディレクトリ)
%COBPM_SRC%	対象ソース
%COBPM_PATTERN%	解析パターン
%COBPM_FILE%	選択メッセージファイルパス
%COBPM_NAME%	選択メッセージファイル名(拡張子なし)

%COBPM_NUM%	選択メッセージ行
%COBPM_MSG%	選択メッセージ内容

22.6 ビルダ

ビルダは、Makefileによる翻訳およびリンクを行うためのユーティリティであり、Xウィンドウ上で動作します。



ビルダを開くには:

以下のコマンドにより、ビルダを起動することができます。

```
pmgrbuild [メッセージ構文指定] [-f Makefile] [-bld] [ターゲット名]
```

メッセージ構文指定

管理するメッセージの種別を指定します。

- -mc　"パス 行番号 内容"
- -mr　"パス:行番号:内容"
- -me　"パス 行番号:JMNxxx内容"

-f Makefile

Makefile名を指定します。

-bld

起動と同時に構築を行います。

ターゲット名

ターゲット名を指定します。

省略時にはALLが指定されたものとみなします。

22.6.1 ビルダの使い方

ビルダは、指定されたMakefileを元に翻訳およびリンクを行うツールです。また、翻訳エラーが発生した場合には翻訳エラーメッセージから該当するエラー箇所を直接修正することができるタグジャンプ機能も利用できるため、効率よくビルドすることができます。

ビルダではビルドタイプを指定するだけで、デバッグ環境の構築や運用環境の構築など目的に応じた環境の構築が簡単に行えます。

ここではビルダの使い方について説明します。

22.6.1.1 翻訳およびリンク

ソースプログラムを翻訳およびリンクするには

1. ビルダ画面のMakefileにMakefile名を指定します。
2. ビルダ画面のターゲットにターゲット名を指定します。
3. ビルダ画面のビルドタイプを指定します。
4. [ファイル] メニューの[ビルド]を選択します。

翻訳結果を元にエラー個所を参照するには

ビルダ画面のメッセージを選択(ダブルクリック)します。

22.6.1.2 翻訳結果の管理

メッセージにマークを付けるには

[編集] メニューの[マークをつける]を選択します。

メッセージに付けたマークを外すには

[編集] メニューの[マークをはずす]を選択します。

メッセージを検索するには

1. [編集] メニューの[検索]を選択します。
2. 文字列を指定して検索します。

メッセージをソートするには

マークでソートするには

[ソート] メニューの[マークをつける]を選択します。

ファイルの行位置でソートするには

[ソート] メニューの[ファイル行位置でソート]を選択します。

メッセージの内容でソートするには

[ソート] メニューの[内容でソート]を選択します。

22.6.1.3 ツール連携

エディタ

エディタと連携することにより、メッセージの示す個所を直接参照することができます。

Make編集

エディタと連携することにより、Makefileを編集することができます。

メッセージ管理ツール

メッセージ管理ツールと連携することにより解析結果メッセージを管理することができます。

22.6.2 ビルダのリファレンス

22.6.2.1 ビルダのメニュー

メニュー・アクセラレータ

カット	[Ctrl] + [X]キー
コピー	[Ctrl] + [C]キー
ペースト	[Ctrl] + [V]キー
終了	[Alt] + [F4] キー

[ファイル]メニュー

Makefile指定	Makefile名を指定します。
ビルド	構築(翻訳およびリンク)を指定したターゲットで開始します。
全ビルド	ターゲット(all)で構築を開始します。
リビルド	クリーン後、再構築を開始します。
クリーン	構築時に生成されるオブジェクトを削除します。
結果保存	構築結果を保存します。
終了	ビルダを終了します。

[編集]メニュー

マークをつける	選択されたメッセージにマークを付けます。
マークをはずす	選択されたメッセージのマークを外します。
カット	選択されたメッセージをカットします。
コピー	選択されたメッセージをコピーします。
ペースト	カットまたはコピーしたメッセージをペーストします。
削除	選択されたメッセージを削除します。
検索	メッセージを検索します。
全選択	すべてのメッセージを選択します。
マークのある項目の選択	マークの付いているメッセージを選択します。
マークのない項目の選択	マークの付いていないメッセージを選択します。
全解除	すべてのメッセージの選択を解除します。
ログ表示域クリア	表示されているログをクリアします。
メッセージ表示域クリア	表示されているメッセージをクリアします。

[ソート]メニュー

マークでソート	マークの有無でメッセージをソートします。
ファイル行位置でソート	ファイルの行位置でソートします。
内容でソート	メッセージでソートします。

[ツール]メニュー

エディタ	エディタを起動します。
Make編集	Make編集ツールを起動します。
メッセージ管理	メッセージ管理ツールを起動します。

[オプション]メニュー

カスタマイズ	ツールの動作を変更するカスタマイズ画面を表示します。
ステータスバー	ステータスバーを表示または非表示します。

22.6.2.2 ビルダのウィンドウおよびダイアログ・ボックス

ビルダ・ウィンドウ

メニュー・バー	ビルダには、次の6つのメニューがあります。[ビルド]、[編集]、[ソート]、[ツール]、[オプション]、および[ヘルプ]です。
Makefile	Makefile名を指定します。デフォルト値はビルダが自動生成したMakefile名が設定されています。
ターゲット	構築する対象を指定します。allを指定するとすべての構築を行います。
ビルドタイプ	構築する種別を指定します。[デバッグ用]、[実行用]、[実行用(性能重視)]、[構文チェック用]、[凍結用(リスト出力)]、[凍結用(性能重視、リスト出力)]、[実行経路確認用]、[誤動作防止確認用]、[性能確認用]などの種別があります。
ツール・バー	ビルダにはビルドに関するツールバーと翻訳エラーなどのビルド時のメッセージ管理に関するツールバーがあります。ビルドに関するツールバーには、次の4つのツールがあります。[ビルド]、[全ビルド]、[リビルド]、および[クリーン]です。また、ビルド時のメッセージ管理に関するツールバーには次の4つがあります。[前ジャンプ]、[ジャンプ]、[次ジャンプ]、および[クリア]です。
構築の度にクリア	構築の度にメッセージ表示ウィンドウのメッセージをクリアするかどうかを指定します。
ビルドメッセージ表示ウィンドウ	Makeコマンドのメッセージを表示します。
ビルドメッセージ管理ウィンドウ	翻訳エラーなどのメッセージを表示します。
ステータス・バー	カーソル位置に応じて、実行可能な作業や表示されているデータの説明などをメッセージで表示します。

[Makefile指定]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
Makefile指定	Makefileを指定します。

[結果保存]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。

選択	保存するファイル名を指定します。

[検索]ダイアログ・ボックス

検索文字列	検索する文字列を指定します。
大文字小文字を区別する	検索する文字列を大文字と小文字で区別して検索する場合に指定します。
前検索	カーソル位置よりも前に存在する文字列を検索します。
次検索	カーソル位置よりも後ろに存在する文字列を検索します。
該当項目の全選択	内容に検索文字列を含むメッセージを全選択します。

[カスタマイズ]ダイアログ・ボックス

全般

Makeコマンド指定	Makeコマンドを指定します。
メッセージ構文	管理対象のメッセージ構文の形式を指定します。形式には [パス 行番号 内容] [パス:行番号:内容] [パス 行番号:JMNxxx内容]があります。
ダブルクリックアクション	メッセージをダブルクリックした場合に実行するコマンドを指定します。
ダブルクリック時のマーキング	ダブルクリックアクションを実行後、メッセージにマークを付けるかどうかを指定します。
ターミナル指定1	ターミナルを起動する際の起動オプションを指定します。デフォルトの設定では、ターミナル1はコマンド実行後にターミナルのウィンドウはそのまま残ります。
ターミナル指定2	ターミナルを起動する際の起動オプションを指定します。デフォルトの設定では、ターミナル2はコマンド実行後にターミナルのウィンドウは残りません。
エラーファイル操作	プロジェクトマネージャとの連携のためにエラーファイルを保存するかどうかを指定します。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

ツール

登録済一覧	[ツール]メニューに登録されているツールを表示します。
追加	[ツール]メニューにツールを登録します。
上へ	[ツール]メニューの表示順序を上に変更します。
下へ	[ツール]メニューの表示順序を下に変更します。
削除	[ツール]メニューからツールを削除します。
ラベル	[ツール]メニューの表示するラベルを指定します。
ニーモニック	ツールのニーモニックを指定します。
コマンド行	ツールの起動コマンドを指定します。
コマンド種別	ツールのコマンドを実行する種別を指定します。
ステータス表示	ステータスバーに表示するメッセージを指定します。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

構築テンプレート

登録済一覧	[ビルドタイプ]に登録されているビルド種別を表示します。
追加	[ビルドタイプ]にビルド種別を登録します。

上へ	[ビルドタイプ]の表示順序を上に変更します。
下へ	[ビルドタイプ]の表示順序を下に変更します。
削除	[ビルドタイプ]からビルド種別を削除します。
構築テンプレート名	構築テンプレート名を指定します。
構築オプション	構築オプションとして、“COBOLOPTS=オプション”の形式でオプションを指定します。
システム値保存	システムに現状の値を保存します。
システム値読込み	システムに登録されている情報を入力します。

[全般・ダブルクリックアクション]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するコマンド名を指定します。

[全般・ターミナル指定1]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[全般・ターミナル指定2]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[ツールメニュー・コマンドファイルの選択]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

22.6.3 カスタマイズ変数一覧

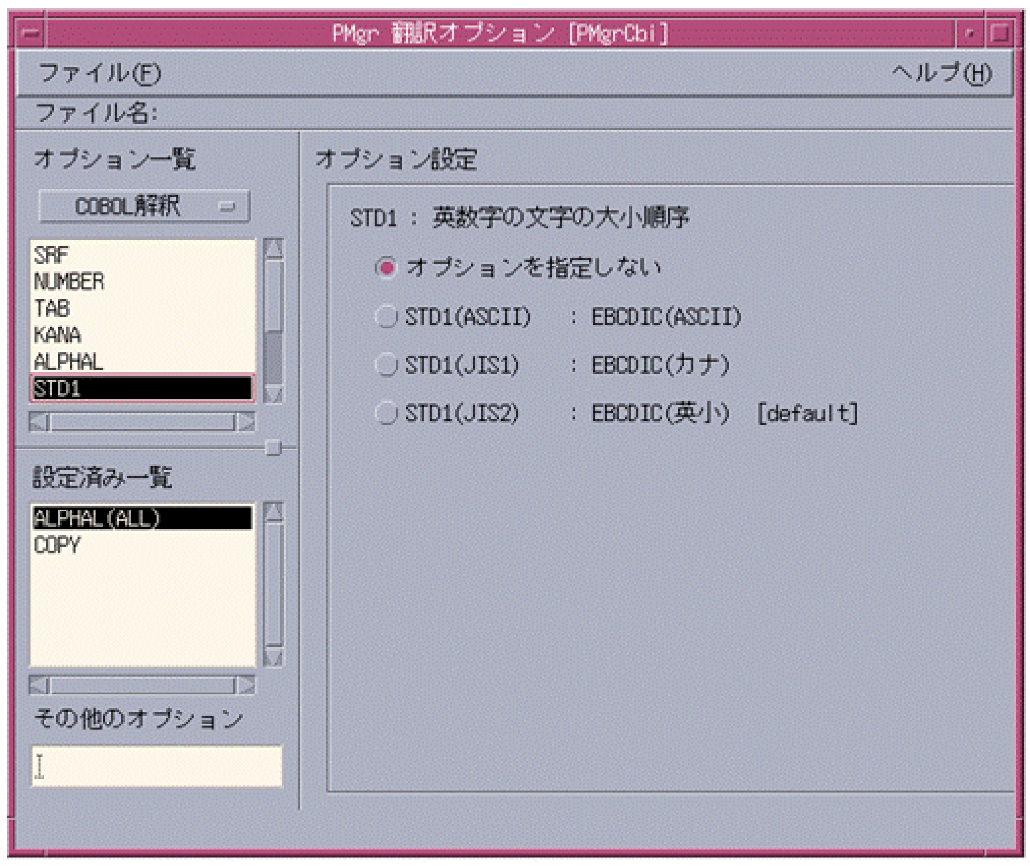
カスタマイズのコマンド指定で使える変数

%COBPM_CWD%	カレントディレクトリ
%COBPM_MAKEFILE%	Makefileパス
%COBPM_MAKENAME%	Makefile名

%COBPM_TARGET%	ターゲット
%COBPM_FILE%	選択メッセージファイルパス
%COBPM_NAME%	選択メッセージファイル名(拡張子なし)
%COBPM_NUM%	選択メッセージ行
%COBPM_MSG%	選択メッセージ内容

22.7 翻訳オプション設定ツール

翻訳オプション設定ツールは、翻訳オプションの設定を行うためのユーティリティであり、Xウィンドウ上で動作します。



翻訳オプション設定ツールを開くには

以下のコマンドにより、翻訳オプション設定ツールを起動することができます。

```
pmgrcbi [ファイル名]
```

ファイル名

翻訳オプションファイル(.cbi)のファイル名を指定します。

22.7.1 翻訳オプション設定ツールの使い方

翻訳オプション設定ツールは、cobolコマンドの翻訳オプションを設定するツールです。

ここでは翻訳オプション設定ツールの使い方について説明します。

22.7.1.1 翻訳オプション設定

翻訳オプションを設定するには

1. 翻訳オプション画面のオプション一覧でオプションの分類を指定します。
2. 翻訳オプション画面のオプション一覧で設定するオプションを選択します。
3. 翻訳オプション画面のオプション設定で指定するオプションの詳細情報を選択します。
4. [ファイル] メニューの[保存]を選択します。

既存の翻訳オプションファイルの内容を変更するには

1. [ファイル] メニューの[オープン]を選択し、既存の翻訳オプションファイルを選択します。
2. 翻訳オプション画面の設定済み一覧で変更するオプションを選択します。
3. 翻訳オプション画面のオプション指定でオプションの詳細情報を選択します。
4. [ファイル]メニューの[保存]を選択します。

22.7.2 翻訳オプション設定ツールのリファレンス

22.7.2.1 翻訳オプション設定ツールのメニュー

メニュー・アクセラレータ

終了	[Alt] + [F4] キー

[ファイル]メニュー

オープン	既存の翻訳オプションファイルのファイル名を指定します。
保存	翻訳オプションをオプションファイルに保存します。
別名保存	翻訳オプションを名前を付けて保存します。
終了	翻訳オプション設定ツールを終了します。

22.7.2.2 翻訳オプション設定ツールのウィンドウおよびダイアログ・ボックス

翻訳オプション・ウィンドウ

メニュー・バー	翻訳オプション設定ツールには、[ファイル] メニューがあります。
オプション一覧	オプション一覧には、翻訳オプションが表示されています。表示方法には、次の七つの種類があります。[全オプション]、[COBOL解釈]、[翻訳レベル]、[翻訳リスト]、[目的プログラム]、[実行時の処理]、および[デバッグ支援]です。
指定済み一覧	指定されている翻訳オプションが表示されます。
その他のオプション	翻訳オプションを文字列で指定します。
オプション設定	オプション一覧または指定済み一覧に表示されている翻訳オプションを選択すると、該当する翻訳オプションの詳細な設定情報が表示されます。

[オープン]ダイアログ・ボックス

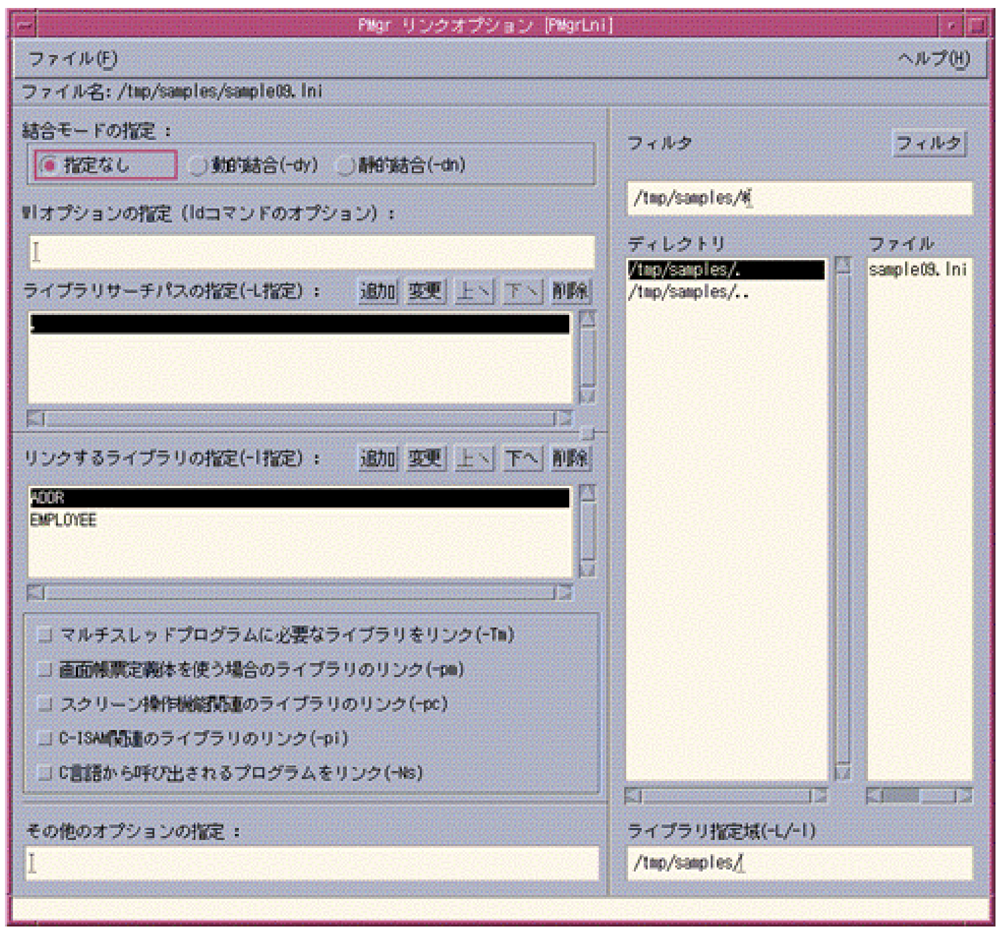
フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[別名保存]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

22.8 リンクオプション設定ツール

リンクオプション設定ツールは、リンクオプションの設定を行うためのユーティリティであり、Xウィンドウ上で動作します。



リンクオプション設定ツールを開くには:

以下のコマンドにより、リンクオプション設定ツールを起動することができます。

```
pmgrlni [ファイル名]
```

ファイル名

リンクオプションファイル(.lni)のファイル名を指定します。

22.8.1 リンクオプション設定ツールの使い方

リンクオプション設定ツールは、リンクオプションを設定するツールです。

ここではリンクオプション設定ツールの使い方について説明します。

22.8.1.1 リンクオプション設定

リンクオプションを設定するには

1. リンクオプション画面の結合モードの指定でリンク方法を選択します。
2. リンクオプション画面のWlオプション指定でldコマンドのオプションを指定します。
3. リンクオプション画面のライブラリサーチパスの指定でリンクする共用オブジェクト(.so)またはアーカイブが格納されているディレクトリ名を指定します。
4. リンクオプション画面のリンクするライブラリの指定でリンクする共用オブジェクト名(libA.soの場合はA)またはアーカイブ名(libA.aの場合はA)を指定します。
5. 画面帳票定義体を使用している場合には、リンクオプション画面の画面帳票定義体のライブラリのリンクを選択します。
6. スクリーン操作機能を使用している場合には、リンクオプション画面のスクリーン操作機能関連のライブラリのリンクを選択します。
7. C-ISAMを使用している場合、リンクオプション画面のC-ISAM関連のライブラリのリンクを選択します。
8. C言語で作成したプログラムからCOBOLプログラムを呼び出す場合、リンクオプション画面のC言語から呼び出されるプログラムをリンクを選択します。
9. その他追加したいオプションが存在する場合、リンクオプション画面のその他のオプションの指定に指定します。
10. [ファイル] メニューの[保存]を選択します。

既存のリンクオプションファイルの内容を変更するには

1. [ファイル] メニューの[オープン]を選択し、既存のリンクオプションファイルを選択します。
2. 追加/変更するオプションを指定します。
3. [ファイル] メニューの[保存]を選択します。

22.8.2 リンクオプション設定ツールのリファレンス

22.8.2.1 リンクオプション設定ツールのメニュー

メニュー・アクセラレータ

終了	[Alt] + [F4] キー

[ファイル]メニュー

オープン	既存のリンクオプションファイルのファイル名を指定します。
保存	リンクオプションをオプションファイルに保存します。
別名保存	リンクオプションを名前を付けて保存します。
終了	リンクオプション設定ツールを終了します。

22.8.2.2 リンクオプション設定ツールのウィンドウおよびダイアログ・ボックス

リンクオプション・ウィンドウ

メニュー・バー	リンクオプション設定ツールには、[ファイル] メニューがあります。
結合モードの指定	リンク時の結合モードを指定します。結合モードには、次の3つがあります。[指定なし]、[動的結合(-dy)]、および[静的結合(-dn)]です。
Wlオプションの指定	ldコマンドのリンクオプションを指定します。
ライブラリサーチパスの指定	ライブラリサーチパス(-L)を指定します。[追加]、[変更]、[上へ]、[下へ]、[削除]ボタンによって指定します。

リンクするライブラリの指定	リンクする共用ライブラリ(-l)を指定します。[追加]、[変更]、[上へ]、[下へ]、[削除]ボタンによって指定します。
マルチスレッドプログラムに必要なライブラリをリンク	マルチスレッドプログラムをリンクする場合に指定します。(-Tmオプション)
画面帳票定義体を使う場合のライブラリのリンク	画面帳票定義体を使用しているプログラムをリンクする場合に指定します。(-pmオプション)
スクリーン操作機能関連のライブラリのリンク	スクリーン操作機能を使用しているプログラムをリンクする場合に指定します。(-pcオプション)
C-ISAM関連のライブラリのリンク	C-ISAMを使用しているプログラムをリンクする場合に指定します。(-piオプション)
C言語から呼び出されるプログラムのリンク	C言語から呼び出されるプログラムをリンクする場合に指定します。(-Nsオプション)
その他のオプション指定	リンクオプションを文字列で指定します。
フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
ライブラリ指定域	ライブラリサーチパス(-L)またはリンクする共用ライブラリ(-l)を指定する場合に、対象となる共用ライブラリまたはアーカイブを指定します。

[オープン]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

[別名保存]ダイアログ・ボックス

フィルタ	表示するファイルを指定します。アスタリスク (*) は、ディレクトリ中のすべてのファイルを表示します。また、ワイルドカード文字を使用して指定することもできます。
ディレクトリ	参照するディレクトリを指定します。
ファイル	フィルタの指定に従って、現在のディレクトリに存在するファイルを表示します。
選択	選択するファイル名を指定します。

第23章　対話型デバッガの使い方

本章では、作成したプログラムをCOBOLの対話型デバッガを使ってデバッグする方法について説明します。

注意

対話型デバッガのスクリーンモードは、将来のNetCOBOLリリースで提供を停止する予定です。Windows版NetCOBOL Studioのリモートデバッグ機能の利用をご検討ください。

NetCOBOL Studioのリモートデバッグ機能については、"第26章 分散開発支援機能"を参照してください。

23.1 デバッガの概要

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する場合に、デバッガを使用します。当デバッガは、実行可能プログラムをそのままデバッグの対象にしています。したがって、デバッグ用の実行可能プログラムで実際の業務を運用することができ、運用中のトラブルに対して、ただちに原因の追求を始めることができます。

23.1.1 デバッガの処理モード

デバッガの処理モードには、スクリーンモードとラインモードがあります。

以下に、それぞれの処理モードについて説明します。

スクリーンモード

Xウィンドウの画面を表示してデバッグする、一般的な処理形態です。デバッガは、ソースプログラムをメインウィンドウに表示し、現在実行している文位置や設定されている中断点の位置を網かけ表示します。表示しているソースプログラムに対する直接的な操作でデバッグ作業を簡単に行うことができます。

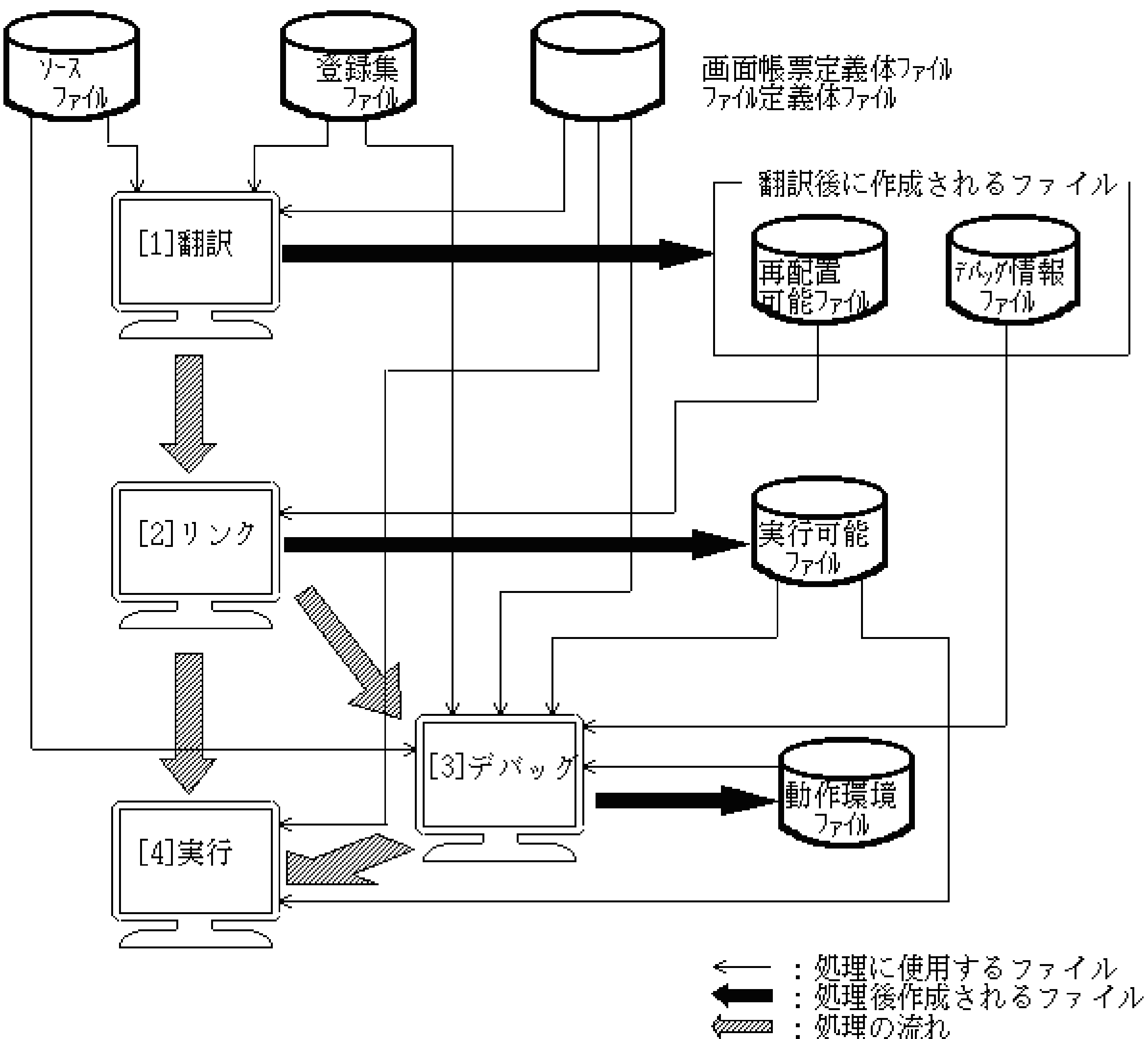
実行時のコード系がUnicodeの場合は、スクリーンモードはサポートしていないため、ラインモード、または、リモートデバッガを使用してください。リモートデバッガについては、"26.2 分散開発支援機能"を参照してください。

ラインモード

TTY端末を使用し、デバッグ画面を表示しないでコマンドの入力および出力だけでデバッグする方法です。シンボリック・デバッガと同等の機能をもち、スクリーンモードと同様にCOBOLソースレベルのデバッグが可能です。ただし、機械語レベルのデバッグはできません。

23.2 デバッグの手順

ここでは、デバッガを使用して、COBOLプログラムをデバッグする手順について説明します。



[1] 翻訳

翻訳時にTESTオプションを指定します。詳細は“23.2.2 翻訳”を参照してください。

[2] リンク

“23.2.3 リンク”を参照してください。

[3] デバッグ

翻訳時に出力されたデバッグ情報ファイル、ソースプログラムおよび実行可能プログラムなどの資産を入力することにより、デバッグを開始します。

[4] 実行

デバッグが終了したプログラムを実際に実行します。ソースプログラムを修正した場合にかぎり、再翻訳する必要があります。

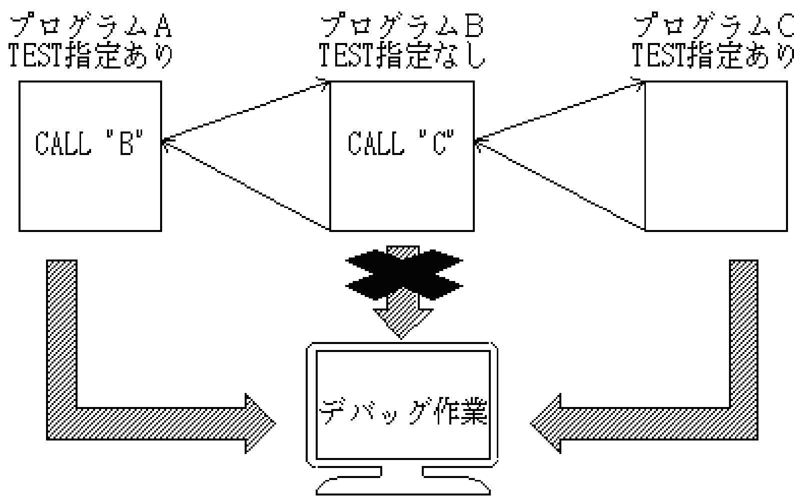
23.2.1 デバッグ対象プログラム

デバッガでは、デバッグの対象となるCOBOLプログラムを、デバッグプログラムと呼びます。デバッグプログラムは、実行可能プログラムまたは共用ライブラリプログラムです。デバッグプログラムは、COBOLコンパイラにより作成されます。

23.2.2 翻訳

デバッグプログラムを作成するためには、cobolコマンドで翻訳オプションTESTを指定して翻訳します。翻訳オプションTESTを指定すると、コンパイラはデバッガのための情報(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に置換えたファイル名で作成されます。

デバッガの機能を利用しないCOBOLプログラムでは、翻訳オプションTESTを指定して翻訳する必要はありません。TESTオプションが指定されたCOBOLプログラムだけが、デバッグ対象となります。TESTオプション指定のないCOBOLプログラムやほかの言語で記述されたプログラムは、デバッガ配下で実行することができますが、デバッグはできません。



注意

cobolコマンドで翻訳オプションTESTを指定する代わりに、"-Dt"オプションを指定することもできます。

23.2.3 リンク

デバッグプログラムは、"23.2.2 翻訳"で作成した再配置可能プログラムを、リンカでリンクします。デバッグプログラムとして、リンク時に特に意識する必要はありません。

23.3 デバッガの起動

デバッグを開始する方法には、大別すると次の2種類の方法があります。

a. デバッガがデバッグするプログラムを起動する一般的な起動方法

b. デバッグしたいプログラムからデバッガを起動し、デバッグを開始する方法

ここではa.での一般的な起動方法について示します。b.については、"23.8 Interstageなどのサーバ環境で動作するプログラムのデバッグ"を参照してください。

デバッガの処理モードには次の2つのモードがあります。

- スクリーンモード
- ラインモード

23.3.1 スクリーンモードでの起動

デバッガをスクリーンモードで起動するには、次の手順に従ってください。

```
$ svd
```

スクリーンモードでデバッガが起動された場合、“図23.1 読み込みウィンドウ”のウィンドウが表示されます。

図23.1 読み込みウィンドウ

ここでは、“図23.1 読み込みウィンドウ”を参照しながら、スクリーンモードでのデバッグに必要な資源の指定について説明します。なお、ディレクトリ名指定への入力を省略した場合は、カレントディレクトリが指定されたものとみなされます。

コマンドライン

起動プログラムとする実行可能ファイル名を指定します。実行可能ファイル名に続けて、実行時オプションを指定することができます。実行時オプションと実行時オプションとの間には、少なくとも1つ以上の空白をあける必要があります。
このパラメタは、省略できません。

開始プログラム名

デバッグを開始するプログラム名を指定します。
このパラメタは、省略可能です。省略した場合は、プログラムの実行で最初に見つかったプログラムがデバッグ開始プログラムとなります。

コマンドファイル

デバッグの処理手順が記録されているコマンドファイル名を指定します。バッチデバッグの入力ファイルとなります。
拡張子を省略した場合は拡張子logのファイルが指定されたものとみなされます。バッチデバッグおよびコマンドファイルについては、“23.5.6 デバッグ作業を自動化する”を参照してください。
このパラメタは、省略可能です。省略した場合は、バッチデバッグは行われません。

履歴ファイル

デバッグの処理手順、または実行結果を記録するファイル名を指定します。
拡張子を省略した場合は、拡張子logのファイルが指定されたものとみなされます。
指定されたファイルがすでに存在している場合、そのファイルの元の内容は失われます。
このパラメタは、省略可能です。省略した場合は、実行結果は記録されません。コマンドファイルの指定時に省略した場合、バッチデバッグの実行結果の確認は行えません。

テスト進捗ファイル

テスト網羅度測定機能を使用する場合に、テスト進捗ファイル名を指定します。
拡張子を省略した場合は、拡張子smpのファイルが指定されたものとみなされます。
テスト進捗ファイルには、テスト進捗情報が格納されます。テスト進捗情報については、"テスト進捗情報"を参照してください。
このパラメタは、省略可能です。省略した場合は、テスト網羅度測定機能を使用することはできません。

画面帳票定義体拡張子

画面帳票定義体の拡張子を指定します。拡張子を複数指定する場合は","で区切って指定します。この場合、指定された順番で画面帳票定義体が検索されます。初期値は"pmd,pxd,smd"が設定されています。
このパラメタを省略した場合は、"pmd,pxd,smd"を指定したものとみなされます。拡張子なしを指定するには、文字列Noneを指定してください。

ファイル定義体拡張子

ファイル定義体の拡張子を指定します。拡張子を複数指定する場合は","で区切って指定します。この場合、指定された順番でファイル定義体が検索されます。初期値は"ffd"が設定されています。
このパラメタを省略した場合は、"ffd"を指定したものとみなされます。拡張子なしを指定するには、文字列Noneを指定してください。

原始プログラム

ソースプログラムを格納しているディレクトリを指定します。複数個のディレクトリを指定したい場合は、コロン(:)で区切って指定します。
デバッガは、指定されたディレクトリ順にソースプログラムを検索します。
デバッグ対象のCOBOLソースプログラムをメインウィンドウに表示するために必要となります。
このパラメタは、省略可能です。省略した場合は、カレントディレクトリが指定されたものとみなされます。

登録集

登録集原文を格納しているディレクトリを指定します。複数個のディレクトリを指定したい場合は、コロン(:)で区切って指定します。
デバッガは、指定されたディレクトリ順に登録集原文を検索します。
デバッグ対象のCOBOLソースプログラムから取り込まれる登録集ファイルをメインウィンドウに表示するために必要となります。
このパラメタは、省略可能です。省略した場合は、カレントディレクトリが指定されたものとみなされます。

デバッグ情報

デバッグ情報ファイルを格納しているディレクトリを指定します。複数個のディレクトリを指定したい場合は、コロン(:)で区切って指定します。
デバッガは、指定されたディレクトリ順にデバッグ情報ファイルを検索します。
このパラメタは、省略可能です。省略した場合は、カレントディレクトリが指定されたものとみなされます。

画面帳票定義体

画面帳票定義体を格納しているディレクトリを指定します。複数個のディレクトリを指定したい場合は、コロン(:)で区切って指定します。
デバッガは、指定されたディレクトリ順に画面帳票定義体を検索します。
デバッグ対象のCOBOLソースプログラムから取り込まれる画面帳票定義体ファイルをメインウィンドウに表示するために必要となります。
このパラメタは、省略可能です。省略した場合は、カレントディレクトリが指定されたものとみなされます。

ファイル定義体

ファイル定義体を格納しているディレクトリを指定します。複数個のディレクトリを指定したい場合は、コロン(:)で区切って指定します。
デバッガは、指定されたディレクトリ順にファイル定義体を検索します。
デバッグ対象のCOBOLソースプログラムから取り込まれるファイル定義体ファイルをメインウィンドウに表示するために必要となります。
このパラメタは、省略可能です。省略した場合は、カレントディレクトリが指定されたものとみなされます。

動作環境

デバッグ環境を保持しておくためのファイル(動作環境ファイル)を格納するディレクトリを指定します。デバッガ終了時に"デバッグ環境を保持する"を指定した場合、ここで指定したディレクトリに動作環境ファイル(cobdebug.ini)が作成されます。動作環境ファイ

ルがすでに存在する場合は、元の内容は失われます。
このパラメタは、省略可能です。省略した場合は、カレントディレクトリが指定されたものとみなされます。

環境変数設定

デバッグプログラムに対する環境変数を設定します。通常、デバッグプログラムに対する環境変数は、デバッガを起動したときの環境変数がそのまま引き継がれます。この機能は、デバッガを起動した後、デバッガから引き継いでいない別の環境変数をデバッグプログラムに渡したり、また、デバッガから引き継いだ環境変数を明に変更したりするために使用します。

注意

デバッガから引き継いだ環境変数は、表示されません。実行用の初期化ファイルが指定されている場合は、実行用の初期化ファイルの設定が優先されます。

実行時のコード系がUnicodeの場合は、スクリーンモードで起動しようとした場合もラインモードで起動されます。

23.3.2 ラインモードでの起動

デバッガをラインモードで起動するには、次の手順で行ってください。

```
$ svd -r prog01
```

ラインモードでデバッガが起動された場合、以下のようなプロンプト表示になります。

```
$ svd -r prog01
 +
```

注意

- ラインモードでは、追尾実行を行うことはできません。
- ラインモードでは、デバッガを起動した画面にコマンド結果が出力されます。形式は操作履歴機能を使用したときに出力される操作履歴ファイルと同じです。
- ラインモードでデバッガを終了する場合は、QUITサブコマンドを指定します。

23.3.3 svdコマンド

svdコマンドの入力形式について説明します。

入力形式

コマンド	オプションおよびオペランド
svd	[デバッグに関するオプションの並び] [実行形式ファイル名 [実行形式プログラムに渡す引数]]

オプションおよびオペランドの説明

コマンドとオプションおよびオペランドの間には、1つ以上の空白が必要です。空白の代わりにTABを用いることもできます。

TTY端末を使用する場合は、-rオプションを指定してデバッガをラインモードで起動してください。

以降の説明のディレクトリ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。

- デバッグに関するオプションの並び

  デバッガに通知する各種情報を指定します。

  デバッグに関するオプションを“表23.1 デバッグに関するオプション”に示します。

- 実行形式ファイル名

  デバッグ情報があるCOBOLの実行可能ファイルのパス名を指定します。

- 実行形式プログラムに渡す引数

  指定した実行形式ファイル名の実行形式プログラムに渡す引数を指定します。引数の指定方法については“4.2.1 プログラムの実行形式”を参照してください。

表23.1 デバッグに関するオプション

内容	オプション
デバッグ開始プログラム名の指定	[-p プログラム名]
コマンドファイルの指定	[-b ファイル名]
操作履歴ファイルの指定	[-l ファイル名]
テスト進捗ファイルの指定	[-q[w] ファイル名]
ソースプログラム格納ディレクトリの指定	[-s ディレクトリ名]
登録集格納ディレクトリの指定	[-c ディレクトリ名]
デバッグ情報格納ディレクトリの指定	[-k ディレクトリ名]
画面帳票定義体格納ディレクトリの指定	[-m ディレクトリ名]
画面帳票定義体拡張子の指定	[-e 拡張子]
ファイル定義体格納ディレクトリの指定	[-h ディレクトリ名]
ファイル定義体拡張子の指定	[-o 拡張子]
動作環境ファイル格納ディレクトリの指定	[-a ディレクトリ名]
動作環境ファイル読み込み指定	[-j]
ラインモード指定	[-r]
利用者プログラム標準入出力ウィンドウの抑止指定	[-w]
カバレージファイルの指定	[-v [カバレージレベル] ファイル名]

以下にデバッグに関するオプションについて説明します。

[-p プログラム名]

デバッグ処理を開始する外部プログラム名を指定します。

[-b ファイル名]

デバッガの操作履歴を格納しているファイルを指定します。バッチデバッグの場合に使用します。

[-l ファイル名]

デバッグの処理手順、または実行結果を記録するファイルを指定します。

注意

操作履歴ファイルをコマンドファイルと同一の名前で同時に指定することはできません。同時に指定した場合、ファイルが破壊されることがあります。

[-q[w] ファイル名]

テスト進捗情報を格納するテスト進捗ファイルを指定します。wを指定すると、デバッグプログラムの起動後にテスト進捗管理ウィンドウも表示します。
当オプションは、-vオプションと同時に指定することはできません。

[-s ディレクトリ名]

ソースプログラムを格納しているディレクトリを指定します。複数個のディレクトリを指定したい場合は、コロン(:)で区切って指定します。
デバッガは、指定されたディレクトリ順にソースプログラムを検索します。
このオプションは、省略可能です。省略した場合は、カレントディレクトリが指定されたものとみなされます。
ラインモードで起動する場合には、指定する必要はありません。

[-c ディレクトリ名]

登録集原文を格納しているディレクトリを指定します。複数個のディレクトリを指定したい場合は、コロン(:)で区切って指定します。
デバッガは、指定されたディレクトリ順に登録集原文を検索します。
このオプションは、省略可能です。省略した場合は、カレントディレクトリが指定されたものとみなされます。
ラインモードで起動する場合には、指定する必要はありません。

[-k ディレクトリ名]

デバッグ情報ファイルを格納しているディレクトリを指定します。複数個のディレクトリを指定したい場合は、コロン(:)で区切って指定します。
デバッガは、指定されたディレクトリ順にデバッグ情報ファイルを検索します。
このオプションは、省略可能です。省略した場合および指定されたディレクトリから検索できなかった場合は、(1)カレントディレクトリ、(2)実行可能プログラムまたは共用オブジェクトプログラムが格納されているディレクトリの順で検索します。

[-m ディレクトリ名]

画面帳票定義体を格納しているディレクトリを指定します。複数個のディレクトリを指定したい場合は、コロン(:)で区切って指定します。
デバッガは、指定されたディレクトリ順に画面帳票定義体を検索します。
このオプションは、省略可能です。省略した場合は、カレントディレクトリが指定されたものとみなされます。
ラインモードで起動する場合には、指定する必要はありません。

[-e 拡張子]

画面帳票定義体の拡張子を指定します。拡張子なしにする場合は、文字列Noneを指定します。なお、環境変数SMED_SUFFIXを使って拡張子を指定することもできます。SMED_SUFFIXでは拡張子を“,”で区切ることで複数の拡張子を指定することができます。その場合は、指定された順に画面帳票定義体の検索が行われます。
ラインモードで起動する場合には、指定する必要はありません。

[-h ディレクトリ名]

ファイル定義体を格納しているディレクトリを指定します。複数個のディレクトリを指定したい場合は、コロン(:)で区切って指定します。
デバッガは、指定されたディレクトリ順にファイル定義体を検索します。
このオプションは、省略可能です。省略した場合は、カレントディレクトリが指定されたものとみなされます。
ラインモードで起動する場合には、指定する必要はありません。

[-o 拡張子]

ファイル定義体の拡張子を指定します。拡張子なしにする場合は、文字列Noneを指定します。なお、環境変数FFD_SUFFIXを使って拡張子を指定することもできます。FFD_SUFFIXでは拡張子を“,”で区切ることで複数の拡張子を指定することができます。その場合は、指定された順にファイル定義体の検索が行われます。
ラインモードで起動する場合には、指定する必要はありません。

[-a ディレクトリ名]

動作環境ファイル(cobdebug.ini)を格納するディレクトリを指定します。
このオプションは、省略可能です。省略した場合は、カレントディレクトリが指定されたものとみなされます。

[-j]

動作環境ファイル(cobdebug.ini)に保存されたデバッグ資源情報を読み込んでデバッガを起動します。

[-r]

デバッガをラインモードで起動します。
当オプションを省略した場合の処理モードは、実行時のコード系によって異なります。当オプションを省略した場合の処理モードを以下に示します。

実行時のコード系	当オプション省略時の処理モード
EUC	スクリーンモード
シフトJIS	スクリーンモード
Unicode	ラインモード

注意

ラインモードで起動する場合、実行形式ファイル名を省略することはできません。

[-w]

利用者プログラム標準入出力ウィンドウの表示を抑止します。
当指定は、スクリーンモードで起動した場合にだけ有効です。

注意

スクリーン操作機能を使用したプログラムの入出力の画面として、利用者プログラム標準入出力ウィンドウは使用できません。スクリーン操作機能を使用したプログラムをデバッグする場合には、当オプションを指定する必要があります。このとき、スクリーン操作機能での入出力画面は、デバッガを起動した画面となります。

[-v [カバレージレベル] ファイル名]

カバレージレベルには、カバレージ機能でのカバレージレベルを指定します。
指定する値を以下に示します。

カバレージレベル	意味
1	カバレージレベル1
2	カバレージレベル2

カバレージレベルを省略した場合には、カバレージレベル1が指定されたものとみなされます。
ファイル名には、カバレージファイル名を指定します。拡張子を省略した場合には、拡張子cvrが指定されたものとみなされます。指定されたファイルがすでに存在している場合には、そのファイルを上書きします。カバレージレベルおよびカバレージ機能については、"23.7 カバレージ機能"を参照してください。
当指定は、ラインモードで起動した場合にだけ指定可能です。
当オプションは、-qオプションと同時に指定することはできません。

23.3.4 注意事項

パラメタの優先順位

スクリーンモードの場合は、デバッガコマンドとして指定したパラメタは読み込みウィンドウに表示されます。環境変数から指定可能なパラメタも含めると、1.読み込みウィンドウ、2.コマンド指定、3.環境変数、4.動作環境ファイルの順番に優先順位が低くなります。

排他処理

デバッガで必要とする各資源については、排他していません。デバッガの使用中にそれらの資源をアクセスすると、誤動作することがあるので注意してください。

登録集原文

登録集原文を格納したディレクトリは、読み込みウィンドウおよび起動コマンド以外に環境変数COBCOPYで指定することができます。また、登録集名が指定された登録集ファイルの場合は、登録集名と同名の環境変数から指定することもできます。

画面帳票定義体

画面帳票定義体を格納したディレクトリは、読み込みウィンドウおよび起動コマンド以外に環境変数FORMLIBで指定することができます。

ファイル定義体

ファイル定義体を格納したディレクトリは、読み込みウィンドウおよび起動コマンド以外に環境変数FILELIBで指定することができます。

svdコマンドのオプションについて

-c,-m,-e,-h,-sおよび-oオプションは、ラインモードで起動する場合には指定する必要はありません。

動作環境ファイル(cobdebug.ini)について

デバッガが使用する動作環境ファイル(cobdebug.ini)は、以下の権限で作成されます。

所有者

参照権および更新権

グループ

参照権

その他のユーザ

参照権

そのため、動作環境ファイルを作成したユーザ以外が動作環境ファイルに書き込みをすることはできません(更新しようとした場合、エラーメッセージが表示されます)。動作環境ファイルに対する更新権のないユーザが動作環境を更新する場合は、動作環境ファイルに更新権を付与しておく必要があります。

23.4 デバッガの操作

23.4.1 画面の構成

以下にスクリーンモードのデバッガの主なウィンドウを示します。これらのウィンドウはそれぞれアイコン化することができます。

- メインウィンドウ
- データ監視ウィンドウ
- ラインコマンド入力ウィンドウ
- 利用者プログラム標準入出力ウィンドウ

その他のウィンドウについては、ヘルプを参照してください。

メインウィンドウ

デバッガの基本ウィンドウであり、デバッガが起動されると必ず表示されます。利用者はこのウィンドウに表示されているソースプログラムを参照しながら、デバッグ作業を行います。ウィンドウ上部のプログラム名(クラス名)リストを選択することで、表示するソースプログラムを切替えることができます。デバッグ作業は、メニューやツールバーのボタンを操作し、デバッガに対して各種命令を発行することにより行います。各メニューやボタンについての詳細は、ヘルプを参照してください。

以下の文については、網かけ表示されほかの文とは区別されます。括弧内は設定を変更しない場合の網かけ表示の色です。

- 現在中断している文(青)
- 現在逆トレースされている文(青)
- 中断点が設定されている文(赤)
- 未実行文の検索機能で検索された文(黄)

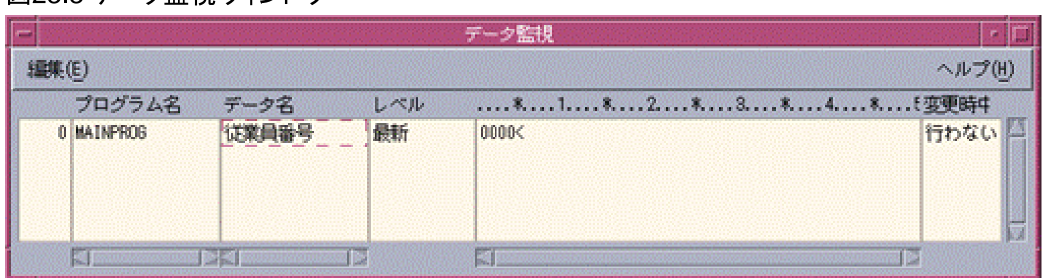
“図23.2 メインウィンドウの形式”にメインウィンドウの形式を示します。

図23.2 メインウィンドウの形式

データ監視ウィンドウ

デバッグプログラムで定義されている各種データ項目の内容を表示するウィンドウです。このウィンドウはメインウィンドウで、データ機能の監視設定が指定された場合にだけ表示されます。“図23.3 データ監視ウィンドウ”にデータ監視ウィンドウを示します。

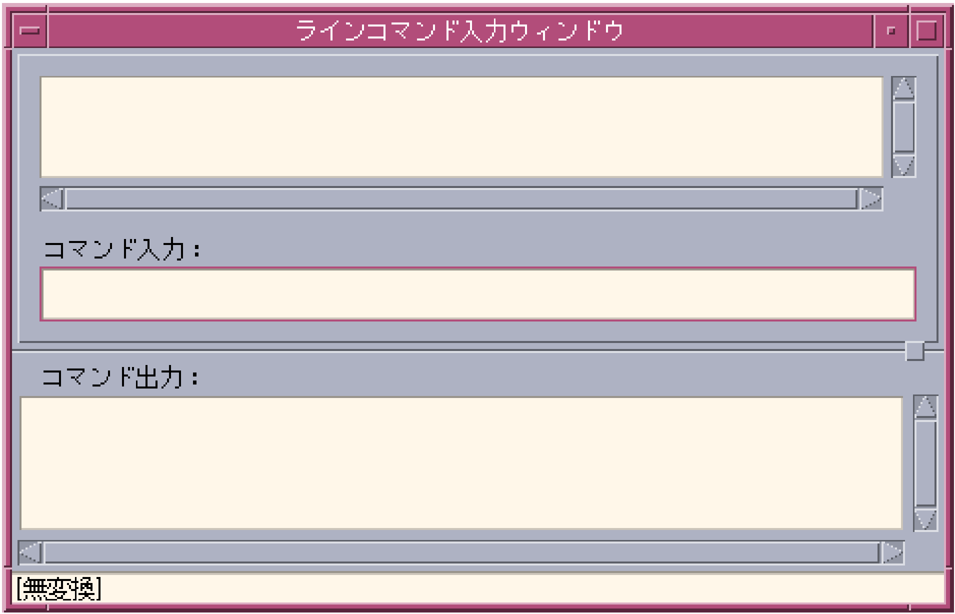
図23.3 データ監視ウィンドウ



ラインコマンド入力ウィンドウ

デバッガに対する各種命令をサブコマンドで指定するときに使用します。このウィンドウは、メインウィンドウの[オプション]メニューの“ラインコマンド入力ウィンドウ”を選択することにより表示されます。サブコマンドの使用方法については、“23.4.2 サブコマンドの入力”を参照してください。“図23.4 ラインコマンド入力ウィンドウ”にラインコマンド入力ウィンドウを示します。

図23.4 ラインコマンド入力ウィンドウ



ラインコマンド入力ウィンドウからのコマンド指定と関連ウィンドウ

以下のウィンドウについては、ラインコマンド入力ウィンドウからのコマンド結果が表示中に反映されます。ほかのウィンドウについては、ウィンドウを再表示することで結果が反映されます。

- ソースウィンドウ
- データ監視ウィンドウ
- 中断点管理ウィンドウ
- 呼出し経路一覧ウィンドウ
- 通過回数表示管理ウィンドウ
- テスト進捗管理ウィンドウ

利用者プログラム標準入出力ウィンドウ

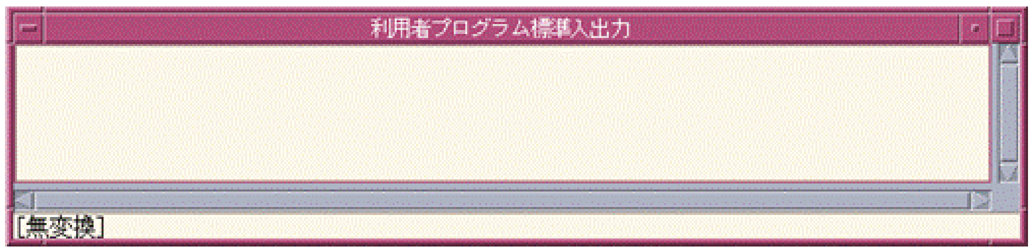
利用者プログラムから小入出力を使用して、ACCEPT文またはDISPLAY文によりデータの入出力を行うときに、システムの標準入出力となるウィンドウです。

利用者プログラムへのデータ入力については、ACCEPT文で行います。プログラムがACCEPT文で入力待ち状態になったときに、当ウィンドウにデータを入力します。リターンキーを押すことにより、プログラムへの入力が完了します。利用者プログラムからのデータ出力については、DISPLAY文で行います。DISPLAY文で出力されたデータが当ウィンドウに出力されます。

当ウィンドウは、デバッガの起動時に“-w”オプションを指定することにより、ウィンドウ表示を抑止することができます。“-w”オプションについては、“23.3.3 svdコマンド”を参照してください。

“図23.5 利用者プログラム標準入出力ウィンドウ”に利用者プログラム標準入出力ウィンドウを示します。

図23.5 利用者プログラム標準入出力ウィンドウ



23.4.2 サブコマンドの入力

デバッガの機能ごとの操作および命令をサブコマンドといいます。サブコマンドの指定方法について、処理モードごとに以下に示します。サブコマンドの詳細については、“23.9 サブコマンド機能”を参照してください。

- ラインモード

  TTY端末から各サブコマンドを入力します。

- スクリーンモード

  サブコマンドを入力するために、次の3種類の手段を用意しています。

  ー　メインウィンドウのメニューを選択する。

  ー　メインウィンドウのツールバーのボタンを選択する。

  ー　ラインコマンド入力ウィンドウから各サブコマンドを入力する。

23.4.2.1 画面入力

デバッガをスクリーンモードとして起動した場合には、以下の方法でサブコマンドを指定します。

メニューまたはツールバーのボタンの選択

メニューおよびツールバーのボタンは、マスク状態(低輝度)ではないものが選択可能です。

以下に対応するサブコマンド名を示します。

メニュー	ボタン名	サブコマンド名
デバッグ制御	環境変更	ENV
	再デバッグ	RERUN
	終了	QUIT
実行	実行	CONTINUE
	1ステップ	RUNTO
	nステップ	RUNTO
	指定位置実行	RUNTO
	特定実行	RUNTO
	特定実行条件	RUNTO
	実行開始点変更	SKIP
中断	中断点設定	BREAK
	中断点管理	STATUS,BREAK,DELETE
	実行監視条件設定	DATACHK,DELDCHK
データ	監視設定	DTRACE,MONITOR
	監視	MONITOR,DELDTR,DELMON
	修正	LIST,SET
	印刷	PRINT
	連絡節獲得	LINKAGE
画面操作	現在位置表示	WHERE
	呼出し経路一覧	CALLS
	行検索	LSEARCH
	検索	SEARCH

メニュー	ボタン名	サブコマンド名
	検索条件	SEARCH
	未実行文検索	OFFSEARCH
	未実行文検索条件	OFFSEARCH
オプション	自動デバッグ	AUTORUN
	通過回数表示設定	COUNT
	通過回数表示管理	STATUS,COUNT,DELCOUNT
	テスト進捗管理	STAMP
ヘルプ	−	HELP

ラインコマンド入力ウィンドウからの入力

サブコマンドは、ラインコマンド入力ウィンドウから入力できます。入力したサブコマンドに誤りがある場合、エラーメッセージが表示されます。この場合は、誤りを修正して、サブコマンドを再入力してください。

23.4.2.2 キーボード入力

キーボードから直接サブコマンドを指定する場合には、ラインモードかスクリーンモードのラインコマンド入力ウィンドウを使用します。ただし、ラインモードでは使用できないサブコマンドがあります。詳細については、“23.9 サブコマンド機能”を参照してください。

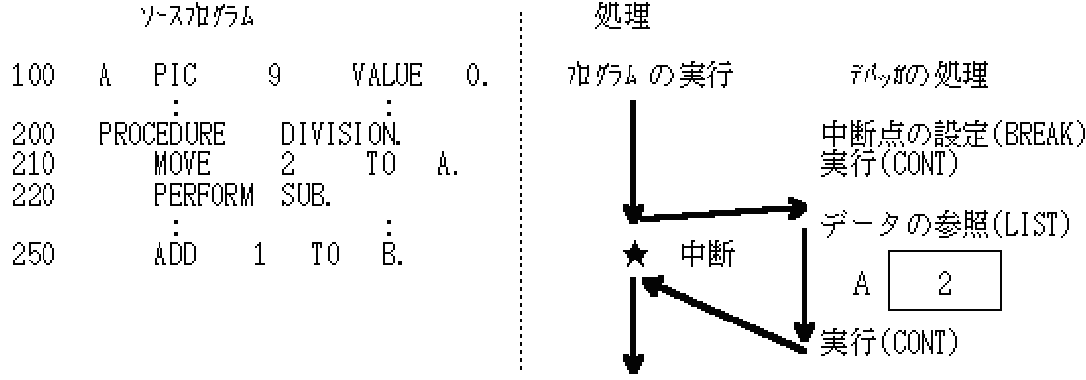
23.5 デバッガの機能

デバッガの各機能については、ヘルプを参照してください。ヘルプは、各ウィンドウのヘルプボタンをクリックすることにより表示されます。

ここでは、デバッガの使い方について説明します。

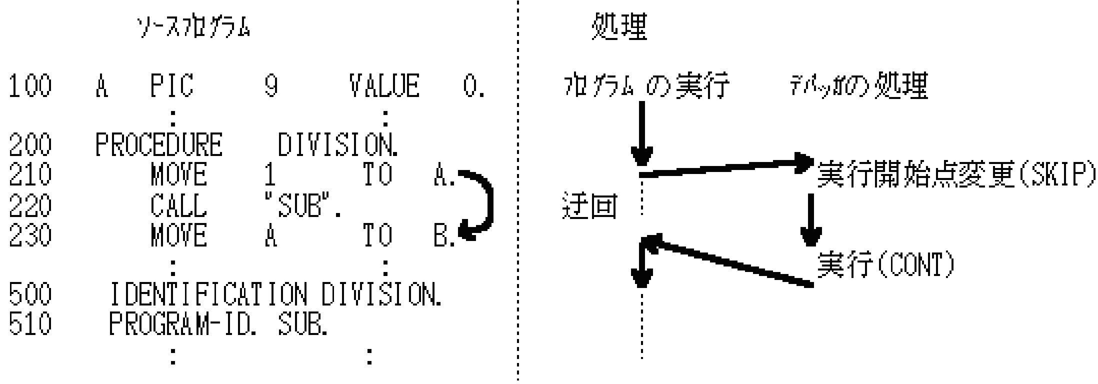
23.5.1 データの内容を確認しながらプログラムの誤りを検出する

プログラムの論理の正当性を確認するために、プログラムの実行を中断してプログラムの制御の流れやデータの内容を調べることができます。これらは、中断機能の中断点設定、データ機能の管理(表示・修正・印刷)を使用することにより実現できます。



23.5.2 プログラムの実行順序を変更する

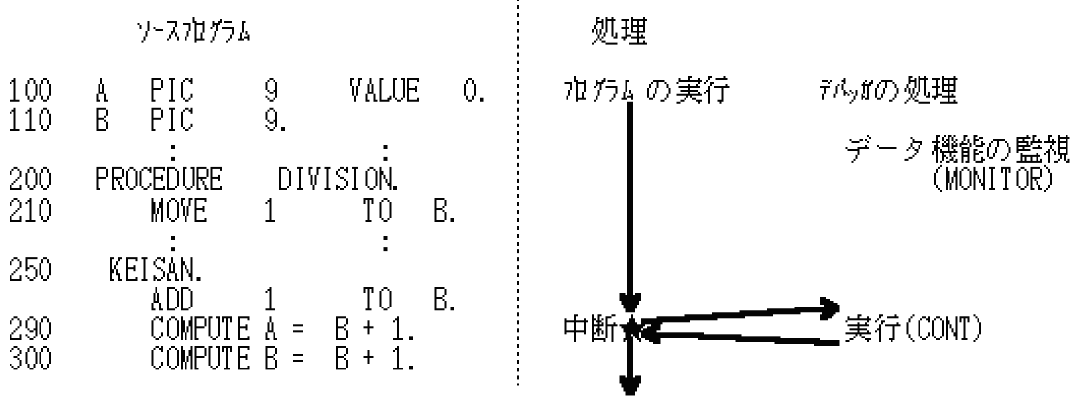
呼び出している副プログラムに誤りがあるため、その副プログラムを迂回したり、デバッグしたい命令だけに制御を移して、無駄なプログラムの実行を省略したりすることができます。このように本来のプログラムの実行を変えるには、実行機能の実行開始点変更を利用します。



23.5.3 データの変更箇所を調べる

データの値が正しくないためにプログラムが誤動作した場合、誤ったデータがプログラムのどこで変更されたのか、確認する必要があります。また、データが変更された時点やデータがある特定の条件を満たしたときにプログラムを中断して原因を調査します。このように、データに着目してプログラムをデバッグするときには、データ機能の監視や中断機能の実行監視条件設定を使用します。

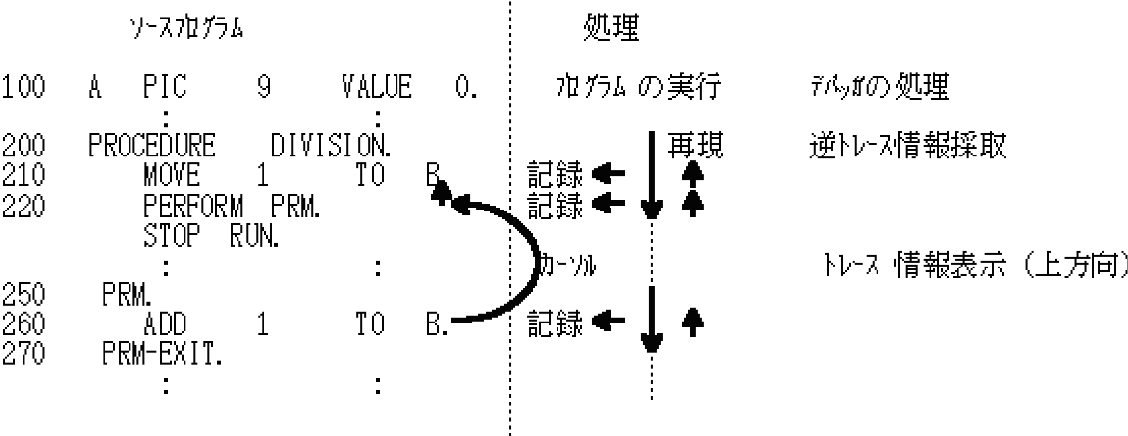
データ機能の監視では、データ監視ウィンドウによりデータの内容を常に画面に表示することができます。



23.5.4 実行経路をたどる

プログラムの実行経路をデバッガで再現したいときに、逆トレース機能を使用します。逆トレース機能では、プログラムの制御の流れをカーソルと網かけ表示(青)の移動で視覚的に確認することができます。なお、トレース方向は上方向と下方向の指定が可能です。逆トレース情報の採取指定は環境変更(ENV)で行います。逆トレース機能はメインウィンドウのツールバーの逆トレースボタン(前文、次文、前手続き、次手続き)をクリックすることにより、実行されます。

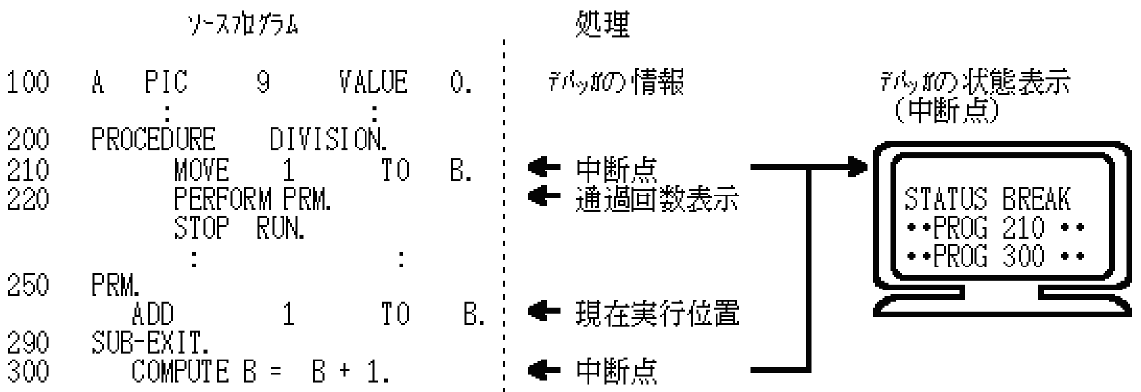
手続きＰＲＭがどこから呼ばれたか、呼び出し元を確認します。



23.5.5 デバッガの状態を知る

デバッガに指定した各種状態(中断点、データ監視、通過回数表示)がわからなくなった場合、設定した情報を確認することができます。これらは各機能に用意されています(中断点管理、監視一覧、通過回数表示管理)。また、現在の実行位置を確かめたいときには、現在位置表示機能を使用すると、メインウィンドウに現在の実行位置のソースプログラムが表示されます。

プログラム（ＰＲＯＧ）の中断点を調べます。



23.5.6 デバッグ作業を自動化する

デバッグ操作を省略したり、同一のデバッグ処理手順を頻繁に行ったりするときには、デバッグ作業を自動化すると便利です。デバッグ作業の自動化は、デバッグの処理手順を記述したコマンドファイルを使用することで実現します。デバッグ形態によりバッチデバッグと自動デバッグに分類されます。

バッチデバッグ

デバッガの起動時に利用者がコマンドファイルを指定する形態です。ラインモードおよびスクリーンモードでの指定ができます。デバッガ起動後は、利用者はデバッガに対して指示を与える必要はありません。スクリーンモードでのバッチデバッグ処理中は、メインウィンドウがアイコン化されます。

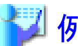
自動デバッグ

デバッガを起動したあとに、デバッグサブコマンドの1つとして実行します。

コマンドファイルの作成方法

コマンドファイルは、以下の方法で作成します。

- デバッガが出力した操作履歴ファイルをコマンドファイルとして利用します。操作履歴ファイルは、デバッガの起動時や環境変更ウィンドウで指定することにより作成されます。
- 利用者がテキストエディタにより、テキストファイルとして作成します。コマンドファイルに記述するデバッグ処理手順は、サブコマンドの構文に従います(“23.9 サブコマンド機能”参照)。

 例

コマンドファイルの記述例

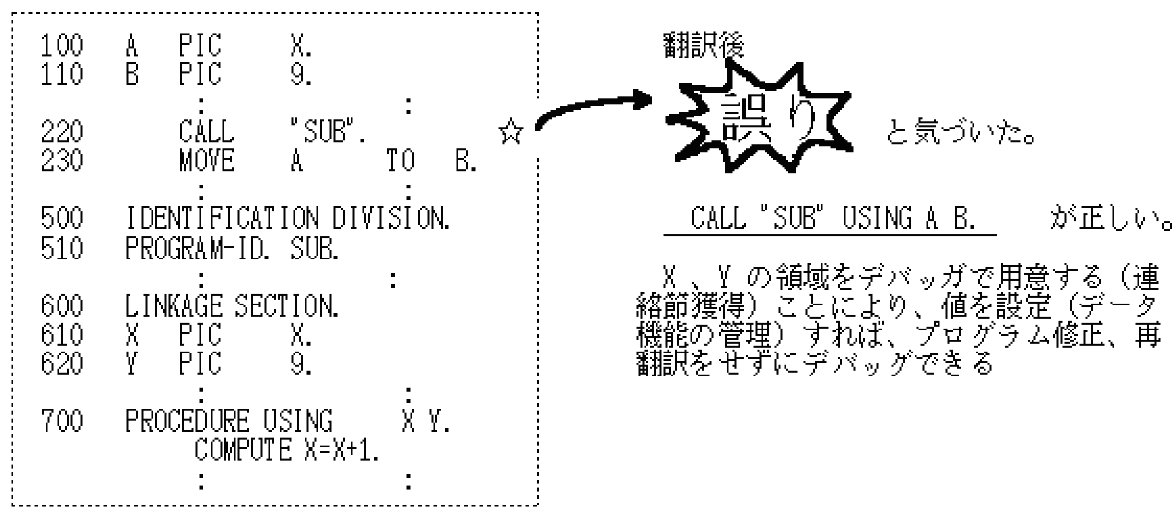
```
BREAK  49                    ;    中断点設定
BREAK  120                   ;    中断点設定
LIST  (i)  FORMAT  (A)       ;    データ表示
LIST  (j)  FORMAT  (H)       ;    データ表示
CONTINUE                     ;    無条件実行
SET   i=10                   ;    データ設定
SET   j=20                   ;    データ設定
CONTINUE                     ;    無条件実行
```

注意

“;”以降は注釈とみなされます。

23.5.7 パラメタ領域を獲得する

副プログラムのデバッグ時に呼出し元プログラムから渡されたパラメタの数が足りない場合、パラメタ領域(連絡節のデータ領域)を確保できます。これにより、呼出し元プログラムの修正および再翻訳なしに副プログラムのデバッグを行うことができます。



23.5.8 デバッグ環境を変更する

デバッガでは、以下の環境を操作することができます。詳細は、ヘルプを参照してください。

- 登録集の展開表示
- ラインコマンドの結果表示
- 履歴採取
- 実行追尾
- 逆トレース情報の採取
- 子プログラム内表示
- PERFORM文内表示レベル
- 属性エラー時の置換文字
- 大文字/小文字の比較基準
- 例外事象発生時の動作

23.5.9 テスト検証およびチューニングを行う

デバッガではテスト検証およびチューニング検証のため以下の機能を用意しています。

- テスト網羅度測定機能
- カバレージ機能

テスト網羅度測定機能は、テストの網羅性を検証しデバッグ作業の効率化およびルートテストの確実性を向上させる機能です。詳細は、“23.6 テスト網羅度測定機能”を参照してください。

カバレージ機能は、デバッグ完了後にプログラムの性能チューニングを手助けする機能です。詳細は、“23.7 カバレージ機能”を参照してください。

23.5.10 動的結合のプログラムをデバッグする

動的結合のプログラムをデバッグする場合、静的結合のプログラムと同じ操作方法でデバッグできます。ただし、以下の注意が必要です。結合の種類については“3.2.2 結合の種類とプログラム構造”を参照してください。

動的リンク構造

動的リンク構造の場合、呼出し元のプログラムがロードされたときに、共用オブジェクトプログラムもロードされます。このため、共用オブジェクトプログラムもデバッグの対象となり、静的結合のプログラムと同じ操作方法でデバッグできます。

動的プログラム構造

動的プログラム構造の場合、共用オブジェクトプログラムは呼出し元から呼び出されたときにロードされます。共用オブジェクトプログラムはロードされた時点から、デバッグの対象になります。

共用オブジェクトプログラムに含まれるプログラムに中断点を設定したい場合、共用オブジェクトプログラムが呼び出される前に中断点の設定を指示しておく必要があります。デバッガではロードされていないプログラムのソースファイルを表示できません。中断点を設定する場合は、BREAKサブコマンドや中断点設定ウィンドウで、ロードされるプログラムの名前を指定して指示します。なお、設定した中断点は解除しないかぎり、プログラムがアンロードされた後に再度ロードされたときも有効になります。

たとえば、PROG05という名前のプログラムの行番号5000に中断点を設定する場合、サブコマンドによる指示は以下のようになります。

```
BREAK 5000 IN(PROG05)
```

23.5.11 オブジェクト指向に対応したデバッグ機能

デバッガでは、ファクトリデータおよびオブジェクトデータを柔軟に参照および変更できるように、一意名にデバッガ独自に拡張した指定形式を用意しています。

拡張した指定形式であるオブジェクト参照修飾子を用いた一意名を使用することにより、通常はプログラムから参照できないファクトリデータやオブジェクトデータを参照することができます。詳細は“23.9.2.5 識別名”の“一意名”を参照してください。

例

オブジェクト参照修飾子を用いた一意名の指定例

```
<ｸﾗｽ定義 A>
IDENTIFICATION DIVISION.
CLASS-ID. A    INHERITS FJBASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    CLASS FJBASE.
  FACTORY.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01 A-F-DT PIC 9 VALUE 1.
  END FACTORY.
  OBJECT.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01 A-O-DT PIC X.
  END OBJECT.
END CLASS A.
```

```
<ｸﾗｽ定義 B>
IDENTIFICATION DIVISION.
CLASS-ID. B    INHERITS A.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    CLASS A.
  FACTORY.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01 B-F-DT PIC 9 VALUE 1.
  END FACTORY.
  OBJECT.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01 B-O-DT PIC X.
  END OBJECT.
END CLASS B.
```

```
<ﾌﾟﾛｸﾞﾗﾑ定義>
IDENTIFICATION DIVISION.
PROGRAM-ID. EX.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    CLASS A
    CLASS B.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 P1.
   02 P10 OBJECT REFERENCE FACTORY OF A.
   02 P11 OBJECT REFERENCE A.
01 P2.
   02 P20 OBJECT REFERENCE FACTORY OF B.
   02 P21 OBJECT REFERENCE B.
PROCEDURE DIVISION.
    INVOKE A "NEW" RETURNING P11.
    INVOKE B "NEW" RETURNING P21.
    :
```

クラスAのファクトリデータ(A-F-DT)を参照する場合は以下のどちらかで記述します。

- P10 OF P1 :: A-F-DT
- A :: A-F-DT

・ P2O OF P2 AS A :: A-F-DT

クラスAのオブジェクトデータ(A-O-DT)を参照する場合は以下のどちらかで記述します。

・ P11 OF P1 :: A-O-DT

・ P21 OF P2 AS A :: A-O-DT

23.6 テスト網羅度測定機能

テスト網羅度測定機能とは、テストの網羅性を測定しテスト進捗を管理することでデバッグ作業の向上およびプログラムの品質向上に役立てる機能です。以下にテスト網羅度測定機能の概要を示します。

テスト進捗の測定

デバッグの作業時に文単位で実行の有無を調べ、ルートテストの消化率を示すことでテストの進捗を定量的に測定することができます。また、テスト進捗に関する情報がデバッグ単位ごとに累積されるため、前回のデバッグ時のテスト進捗情報を使用することで時系列的に作業状況を把握することができます。このテスト進捗を累積するファイルを"テスト進捗ファイル"といいます。

テスト網羅性の測定およびプログラムの品質向上

プログラムは、記述されたすべての文が実行されるという前提でプログラミングされています。そのため、未実行の部分が残っていない状態ならば、プログラムのデバッグは完了したことになり、プログラムの品質も保証することができます。テスト網羅度測定機能には、デバッグ作業中に未実行文を検索する機能が用意されています。この機能を使用することで実行が確認されていない部分を検索し、デバッグ対話中に特定位置に分岐させたり、プログラムを再起動させたりしながらテストの網羅性を高めてプログラムの品質を向上させることができます。

測定結果のマージによるチームデバッグ

テスト進捗および網羅性に関する情報はデバッグごとに累積することができます。特に大規模なプログラム開発の場合、デバッグは担当者ごとにプログラム単位で行うのが一般的です。しかし各人のデバッグ作業の進捗状況により、同一プログラムに対してデバッグ項目を複数で分担しながらデバッグすることもあります。このような場合には個々のプログラム単位やプログラム内でのテスト進捗および網羅性に関する情報を測定できます。しかし、全体のデバッグに対するテスト進捗および網羅性に関する情報を得ることはできません。この機能ではこれら個々の情報をマージするsvdsmpmgコマンドを用意しています。このコマンドを使用することにより、総合的なテストの進捗情報および網羅性の情報を測定することができチームデバッグを円滑に行うことができます。

以下にテスト網羅度測定機能で使用するコマンド一覧を示します。
テスト網羅度測定機能での各コマンドの位置づけおよび使用方法は、次項以降を参照してください。

コマンド名	用途
svd	テスト網羅度測定機能
svdsmpmg	テスト進捗情報のマージ
svdsmpls	テスト進捗ファイルの一覧表示

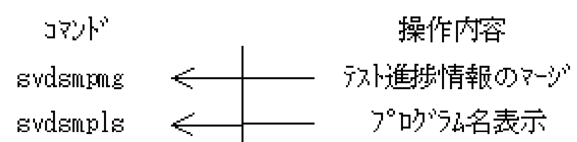
23.6.1 テスト網羅度測定機能の流れ

23.6.1.1 処理の流れ

テスト網羅度測定機能を使用したデバッグ手順、デバッグ操作の全体の流れ図およびテスト進捗ファイルの管理方法(概略図)を以下に示します。

ﾃﾞﾊﾞｯｸﾞ手順	ﾃﾞﾊﾞｯｸﾞ操作
ﾃﾞﾊﾞｯｸﾞ開始 (svd)	ﾃﾞﾊﾞｯｶﾞ起動 ・ﾃｽﾄ進捗ﾌｧｲﾙ(新規/既存)のｵｰﾌﾟﾝ
ﾌﾟﾛｸﾞﾗﾑ実行	実行系ｺﾏﾝﾄﾞ(CONTINUE など) ・ﾃｽﾄ進捗情報の採取
ﾌﾟﾛｸﾞﾗﾑ中断	中断ｺﾏﾝﾄﾞ(BREAK など) ・実行済み行の表示
ﾃﾞﾊﾞｯｸﾞ進捗の 確認・管理	ﾃｽﾄ進捗管理(STAMP) ・ﾃｽﾄ進捗情報の表示 ・ﾃｽﾄ進捗情報のｸﾘｱ・削除 ・ﾃｽﾄ進捗情報のﾌｧｲﾙ出力 ・状態変更(凍結/測定) ・対応ｿｰｽ切替え
未ﾃｽﾄ部分の 洗い出し	未実行文検索(OFFSERACH) ・未実行ﾌﾞﾛｯｸの検索
再ﾃﾞﾊﾞｯｸﾞ	ﾃｽﾄ進捗情報のﾃｽﾄ進捗ﾌｧｲﾙへの 保存・破棄(RERUN)
終了	ﾃｽﾄ進捗情報のﾃｽﾄ進捗ﾌｧｲﾙへの 保存・破棄(QUIT)

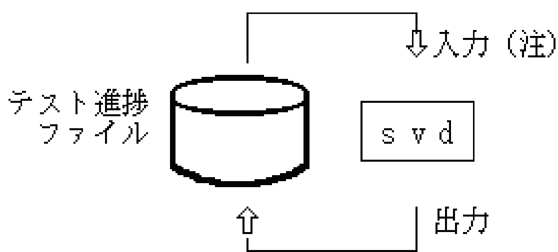
ﾃｽﾄ進捗ﾌｧｲﾙ管理



23.6.1.2 資源の流れ

テスト網羅度測定機能を使用したときに必要となる資源概略図を以下に示します。

図23.6 テスト進捗ファイルとデバッガ



（注）テスト進捗ファイルは、新規および既存が可能

図23.7 テスト進捗情報のマージ

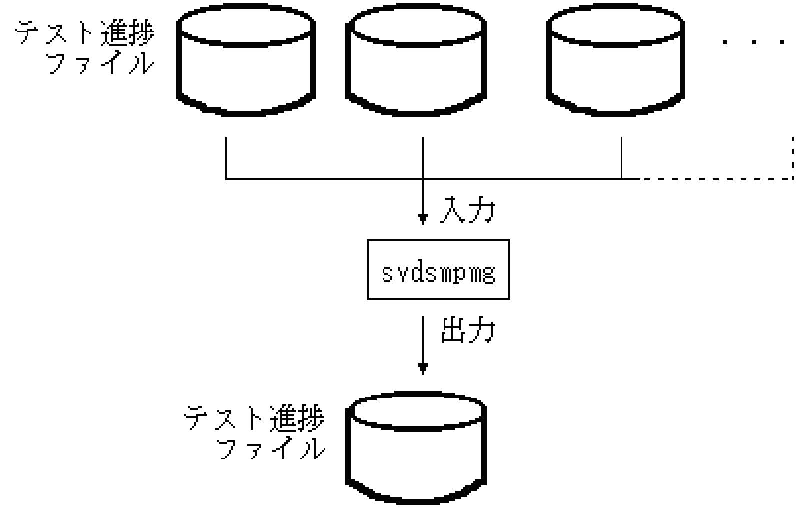
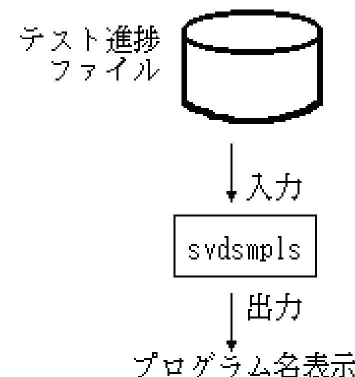


図23.8 プログラム名の表示



23.6.2 テスト網羅度測定機能の指定方法

テスト網羅度測定機能を使用するには以下の2つの指定方法があります。

- デバッガの起動時に、読み込みウィンドウでテスト進捗ファイルを指定する。
- svdコマンドの“-q”オプションでテスト進捗ファイルを指定する。

テスト網羅度測定機能が指定されると、デバッガはテスト進捗情報の採取を開始します。そして、デバッガの終了時または再デバッグ時にテスト進捗情報がテスト進捗ファイルに保存されます。そのため、テスト進捗ファイルはデバッガの終了ではなく、対象プログラムのデバッグ作業の終了まで継続して使用できます。

“-q”オプションの指定については、“23.3.3 svdコマンド”を参照してください。また、テスト進捗情報の詳細については“テスト進捗情報”を参照してください。

以下に起動時のパラメタで指定する例を示します。

例

```
$ svd -q a.smp a.out
```

23.6.3 テスト網羅度測定の機能分類

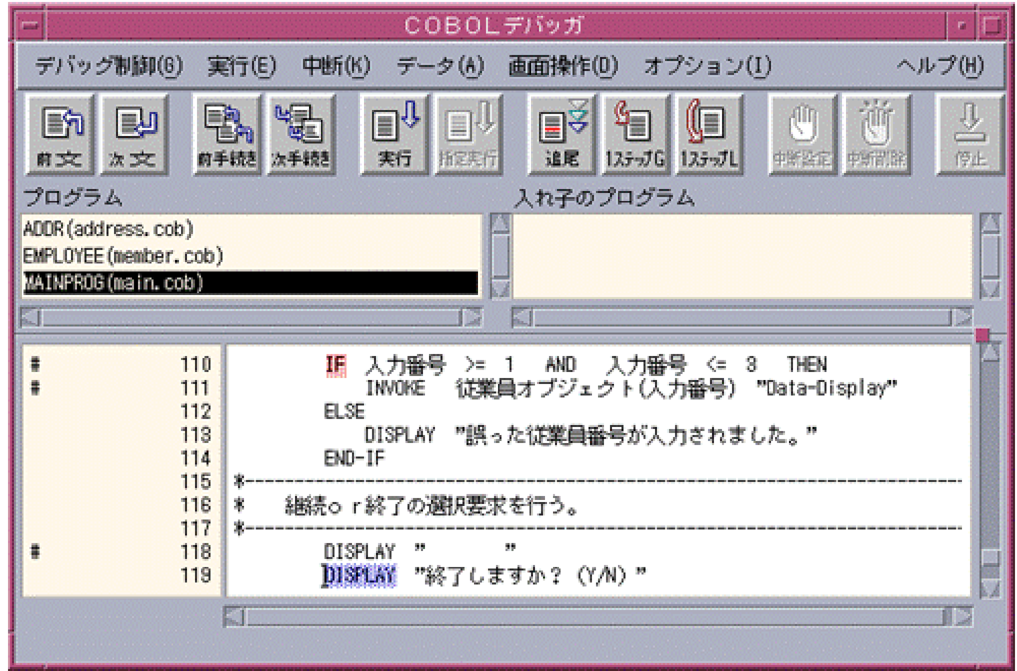
テスト網羅度測定機能は以下の機能からなります。

- 実行済み行の表示
- テスト進捗の管理(STAMP)
- 未実行文の検索(OFFSEARCH)
- テスト網羅度測定の状態変更

23.6.3.1 実行済み行の表示

デバッグプログラムの実行済みの行に対して、行番号の最左端位置に“#”が表示される機能です。追尾実行中は追尾にあわせてメインウィンドウに表示されます。追尾以外の実行ではプログラムの実行が中断したタイミングでメインウィンドウに表示されます。1行に複数のCOBOL文が記述されている場合、その行内のすべての文が実行された時点で対応する行に“#”が表示されます。

当指定は、スクリーンモードで起動した場合にだけ有効です。



23.6.3.2 テスト進捗の管理

テスト進捗情報を管理するための機能です。この機能はテスト進捗管理(STAMP)で実現します。

STAMPサブコマンドの詳細は、“STAMPサブコマンド”を参照してください。

テスト進捗情報

テスト網羅度測定機能は、以下に示す測定対象についてテスト進捗情報を採取します。

測定対象

デバッグ対象の外部プログラム、クラス、インタフェース。

ただし、測定対象となるタイミングはプログラム構造により異なります。以下に測定対象となるタイミングを示します。

起動時

主プログラム、静的結合または動的リンク構造の外部プログラム、クラス。

実行時

動的プログラム構造の外部プログラム、クラス。

またテスト進捗ファイルにあらかじめ保存されているプログラムも測定対象となります。

テスト進捗情報〔例〕

プログラム名

測定対象の外部プログラム、クラス名を表示します。

進捗率

実行可能文数に対する実行済み文数の割合を示します。スクリーンモードではグラフでも表示されます。実行可能文数は登録集で記述された実行可能文を含みます。なお、実行可能文は、厳密には中断点設定可能な文を意味します。中断点設定可能な文の詳細は、BREAKコマンドで指定できる文キーワードを参照してください。

テスト回数

プログラムを再起動した回数を累積します。同一プログラムに対する再デバッグ(RERUN)、デバッガの終了(QUIT)のあと、次にプログラムが実行されたタイミングでテスト回数が1つずつ増えます。対象プログラムの状態が“測定”かつ、1文でも実行されるとテスト回数として加算されます。

前回テスト実施日

テスト進捗ファイルにテスト進捗情報を格納した前回の日付を表示します。

翻訳時間

測定対象のプログラムの翻訳時間を示します。

更新回数

テスト進捗情報を上書きした回数を示します。テスト進捗情報をテスト進捗ファイルに格納したあとに、プログラムを再翻訳したときに、テスト進捗情報を上書きすることができます。

状態

テスト網羅度測定の採取状態を示します。初期値は“測定”です。詳細は、“23.6.3.4 テスト網羅度測定の状態変更”を参照してください。

デバッガは、プログラム終わり見出しEND PROGRAMおよびメソッド終わり見出しEND METHODを実行可能文(中断点設定可能な文)として扱います。これは、プログラムの出口に関する文(STOP RUN, EXIT PROGRAMおよびEXIT METHOD)の記述がない場合にもプログラムの出口を指定して中断できるようにするためです。これらのプログラムの出口に関する文の記述がある場合には、END PROGRAMやEND METHODが実行されないことがあります。したがって、進捗率が100%にならない場合があります。

テスト進捗情報の管理

テスト進捗情報を管理するための以下の機能があります。

文クリア

ソースウィンドウから対象となる文をオブジェクト選択し、当メニューを選択することで実行状態を未実行状態にします。また、メインウィンドウの実行済み行表示(“#”)もクリアします。

クリア、全クリア

選択したプログラムに対する進捗情報(進捗率、テスト回数、前回テスト実施日、状態)をクリアまたは全クリアします。状態は“測定”となります。ただし翻訳時間、更新回数はクリアしません。また、進捗率グラフ、メインウィンドウの実行済み行表示(“#”)もクリアします。

削除、全削除

選択したプログラムに対する進捗情報を削除または全削除し、測定対象から除外します。また進捗率グラフ、メインウィンドウの実行済み行表示(“#”)も削除します。なお、削除指定は、再デバッグまたはデバッガの終了まで有効です。

状態変更

テスト網羅度測定の採取状態を変更します。詳細は、“23.6.3.4 テスト網羅度測定の状態変更”参照してください。

ファイル出力

指定したファイルに現在のテスト進捗情報を出力します。これにより、デバッグ担当者は電子メールなどで管理者に進捗報告をすることができます。なお、指定したファイルが存在しない場合、ファイルが新規に作成されます。ファイルがすでに存在する場合は、テスト進捗情報は追加出力されます。

ソース切替え

選択したプログラムに対するソースプログラムをソースウィンドウに表示します。

23.6.3.3 未実行文の検索

未実行文をメインウィンドウから検索し、カーソルを位置付ける機能を用意しています。未実行文の検索単位として、次の2種類が指定できます。

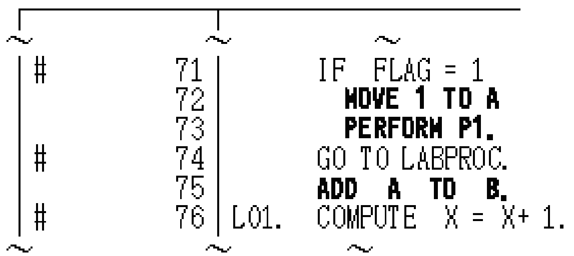
- ブロック
- 文数指定

ブロックを指定すると、未実行文のまとまりを1つの検索ブロックとして各ブロックごとに検索します。文数指定では、指定した文数(連続する未実行文の文数を示し、省略値は1)単位に検索ブロックとして未実行部分を検索します。

検索ブロックに対応するカーソルの位置づけは、検索方向が下(上)方向の場合には、検索ブロックの先頭(最後)の文となります。また、検索された文は、網かけ表示(黄)されます。なお、1行に複数文があり、かつ未実行文がある場合にはその未実行の文も含めたブロックとして扱います。この機能は、未実行文検索、未実行文検索条件(OFFSEARCH)で実現します。

例

図23.9 例1 ブロック単位の検索



下方向に検索するとカーソルは72行のMOVE、75行目のADD の各文先頭に位置付けられます。（ゴシック部分が一つの検索ブロック）

図23.10 例2 文数指定に3を指定した場合

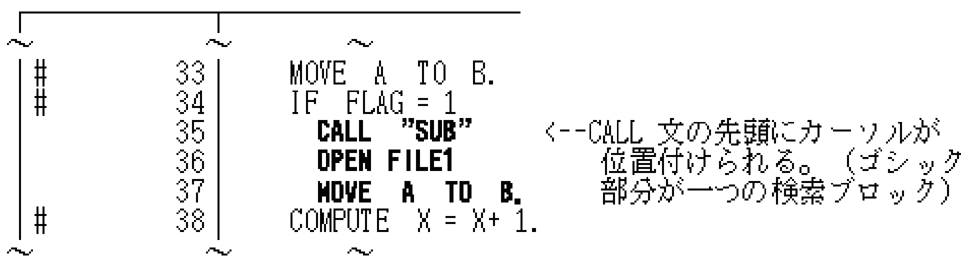


図23.11 例3 文数指定に2を指定した場合

注意

文数指定により利用者が指定した文数が、実際の連続する未実行文の文数よりも小さい場合も検索されます。たとえば例2では、実際の連続する未実行文の文数は3です。しかし、利用者が文数指定により文数として1を指定した場合、デバッガは1つの文からなる検索ブロックを3回にわけて検索します。

23.6.3.4 テスト網羅度測定の状態変更

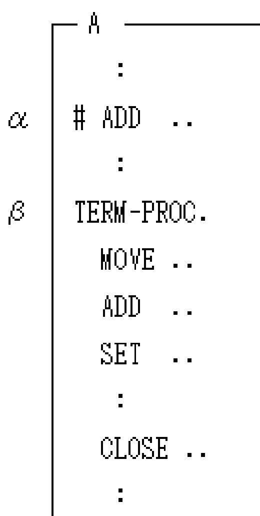
テスト進捗情報に表示されている“状態”はテスト網羅度測定の状態を示します。初期値は、“測定”です。“凍結”状態にすることもできます。凍結状態とはテスト網羅度測定機能を一時的に凍結し、凍結前の状態からテスト進捗情報を一切更新しない状態を示します。すなわち

- 実行済み行はカウントしないで、進捗率は凍結前のままとなります。
- メインウィンドウの実行済み行の表示“#”は凍結前の状態となります。たとえ凍結状態後にプログラムがはじめて実行されたとしても、その実行済み行の印“#”は表示されません。
- “凍結”状態でもテスト進捗情報に対するクリア(文クリア、クリア、全クリア)および削除(削除、全削除)による更新は可能です。

例

“凍結”機能を使用したデバッグ例

- 対象プログラムの網羅度を測定中に、デバッグのために故意に実行制御を変更させるので、一時的に測定を中断したい場合

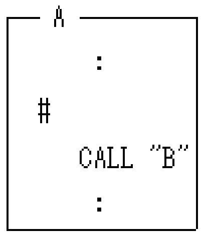


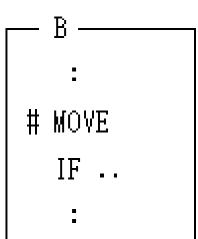
αまではﾃｽﾄ網羅度を測定した。
ﾌﾟﾛｸﾞﾗﾑを再ﾃﾞﾊﾞｯｸﾞではなく､開始点変更により
途中から再開させてﾃﾞﾊﾞｯｸﾞを継続したい。しかし、
ﾌﾟﾛｸﾞﾗﾑを途中から正常に再開するためには､ﾌﾟﾛ
ｸﾞﾗﾑの終了処理(β)を意図的に実行させる必要が
ある。また､終了処理(β)は別に正規なﾙｰﾄで網
羅度を測定したい。

====> 凍結

終了処理(β)を仮実行させたとき､一時的に網羅度
の測定を凍結する。

- プログラムA,プログラムBのデバッグ作業を担当して、正常ルートについてはプログラムAとプログラムBを同時にデバッグして実行ルートを確認した。しかし異常系のテストについては、プログラムBから先に完了させたい。そのため呼出し元のプログラムAで擬似的に動作させプログラムBを呼んでテストする。そこで一時的にプログラムAの網羅度の測定を凍結したい。プログラムAの網羅度の測定は、プログラムBのデバッグが完了してから再開する。





23.6.4 テスト進捗ファイルの管理

テスト進捗ファイルを管理するために各種専用のコマンドが用意されています。

23.6.4.1 テスト進捗情報のマージ

同一プログラムに対するデバッグ項目を複数名で分担してデバッグした場合、個々のプログラム単位や同一プログラム内でのテスト進捗情報を別々のテスト進捗情報ファイルに保存することができます。しかし、全体のデバッグに対するテスト進捗情報を得ることはできません。svdsmpmgコマンドは、これらのテスト進捗情報ファイルの情報をマージするコマンドです。これにより、総合的なテストの進捗情報を求めることができます。

```
svdsmpmg -o ファイル名 file1 file2 .. fileN
```

-o ファイル名

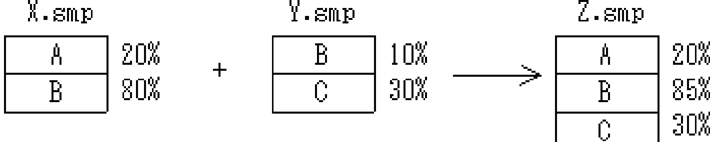
マージ後のテスト進捗ファイル名(出力ファイル)

fileN

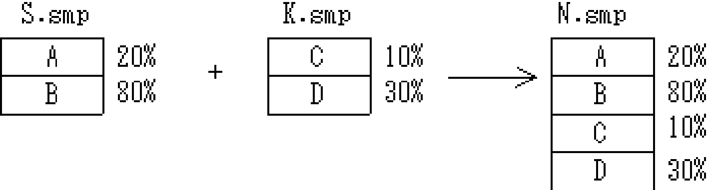
マージ対象のテスト進捗ファイル名N(入力ファイル)

“-o”オプション指定は必ずコマンドの直後に指定する必要があります。

指定例 1: svdsmpmg -o Z.smp X.smp Y.smp



指定例 2: svdsmpmg -o N.smp S.smp K.smp



なお、プログラム名、および、翻訳時間が同じ場合には、各テスト進捗情報(進捗率の実行済み回数、テスト回数、更新回数)が加算マージされます。前回テスト実施日はマージ対象中の最新の日付となります。状態は“測定”になります。なお、同じプログラムでも翻訳時間が異なる場合には、エラーとなり、最新の情報に置換えるかどうか指定できます。

23.6.4.2 テスト進捗情報のプログラム表示

テスト進捗ファイルはバイナリファイルであるため、ファイルの内容を直接参照することはできません。svdsmplsコマンドは、テスト進捗ファイルに保存されている測定対象のプログラム名を一覧表示するコマンドです。これにより、デバッガを起動しなくても、テスト進捗ファイルに保存されている測定対象のプログラム名を知ることができます。なお、プログラム名は、アルファベット順で表示されます。

```
$ svdsmpls    テスト進捗ファイル
```

指定例 1: $ svdsmpls Z.smp

A
B
C

表示結果

23.6.5 テスト網羅度測定機能の注意事項

実行中のプログラム名やクラス名が切り替わったとき、プログラムの翻訳時間とテスト進捗ファイルに格納されているプログラムの翻訳時間が一致しないことがあります。このとき、デバッガは、テスト進捗情報をクリア(翻訳時間、更新回数は除く)するかどうかをたずねるメッセージを表示します。この場合、メッセージが表示された時点でプログラムを実行させたサブコマンド(実行、追尾など実行系サブコマンド)は完了状態となります。

23.7 カバレージ機能

当デバッガでのカバレージ機能とは、デバッガにより論理的な誤りを取り除いたプログラムに対して、プログラムを性能向上させるためのチューニング情報を提供する機能です。チューニング情報として、以下の情報を出力します。

- ソース情報
- プログラム定義情報
- クラス定義情報
- メソッド定義情報
- 翻訳情報
- 行情報
- 走行情報
- 実行情報
- 内部プログラム情報
- メソッド情報
- 時間情報

各情報の詳細および出力情報については、svdcoverコマンドのオンライン・マニュアルを参照してください。

カバレージ機能は、svdコマンドとsvdcoverコマンドを使用することで実現します。svdコマンドはカバレージの情報をカバレージファイルとして出力します。svdcoverコマンドはカバレージファイルのカバレージ統計情報を標準出力に出力します。svdコマンドでカバレージ情報を採取するには"-v"オプションを指定します。このとき、必ず、"-r"オプションを指定してください。"-v"オプションはラインモードの場合にだけ有効になります。

なお、カバレージ統計情報の出力内容および出力形式は、カバレージレベルにより異なります。カバレージレベルの指定は、svdコマンドによりカバレージファイル作成時に行います。カバレージレベルを省略した場合には、カバレージレベルに1が指定されたものとみなされます。

カバレージレベル1

行情報で出力される文ごとの情報として実行回数を出力します。カバレージレベルの省略値です。

カバレージレベル2

行情報で出力される文ごとの情報として実行回数のほかに、実行時間の情報も出力します。

注意

- カバレージレベル2で求める実行時間は、実行された機械語命令の数で表現します。したがって、求められた実行時間は厳密な時間ではなく相対的なCPU使用率を示します。
- カバレージレベル2を指定すると、内部的には機械語命令を1ステップごとに中断・実行を繰り返します。そのため、svdコマンドによるカバレージファイルの作成に時間がかかるので注意が必要です。あらかじめ測定対象を特定して採取することをおすすめします。
- カバレージレベル2では、"開始処理"というフィールドが行情報に出力されます。これは開始プログラムの先頭に到達するまでに要した時間です。これにはプロセスの起動時間や主プログラムがC言語などの他言語(デバッグ対象ではないCOBOLを含む)の開始プログラムに達するまでに実行された時間も含まれます。また、CALL文の実行時間は、次のデバッグプログラムの入口文までに費やした時間となります。

以下にカバレージファイルの出力方法およびカバレージ統計情報の出力例を示します。

カバレージファイルの出力例

```
$ svd -r -v prog.cvr prog01
```

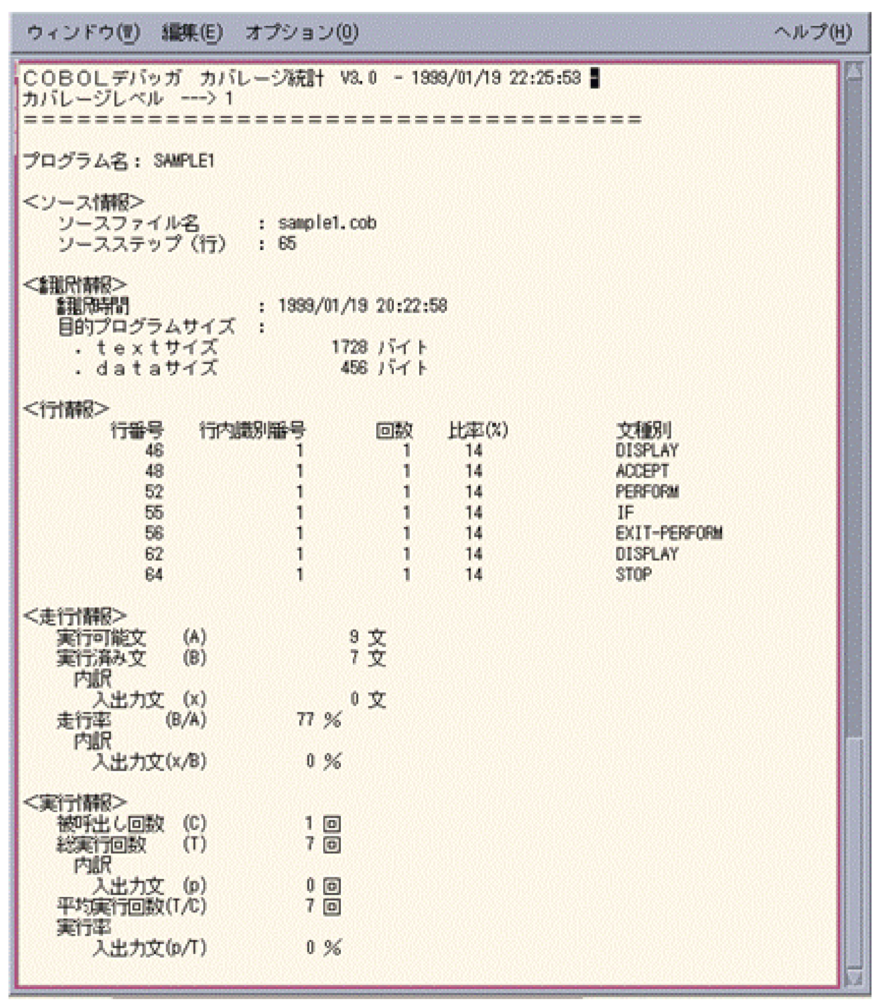
カバレージ統計情報の出力方法

```
$ svdcover [オプション指定] カバレージファイル
```

カバレージ統計情報の出力例

```
$ svdcover prog.cvr
```

カバレージ統計情報〔例〕



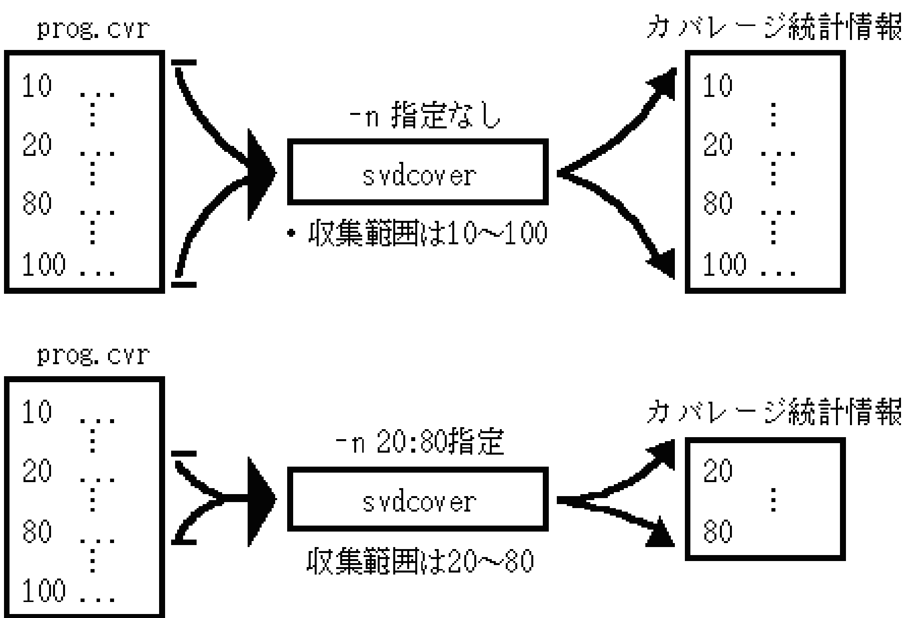
23.7.1 カバレージコマンドのオプション

集計範囲指定(-n)

カバレージ統計情報は、カバレージコマンド(svdcover)により外部プログラムごとに行単位に出力されます。また、カバレージ統計情報の収集範囲は、対象となる外部プログラムの先頭行から最終行までです。“-n”オプションの指定によりカバレージ統計情報の収集範囲を行番号で指定することができます。なお、カバレージ統計情報は実行された外部プログラムと実行された行についてだけ出力されます。指定方法を以下に示します。

```
-n  start:end
```

開始行番号(start)から終了行番号(end)までの範囲で情報を収集します。

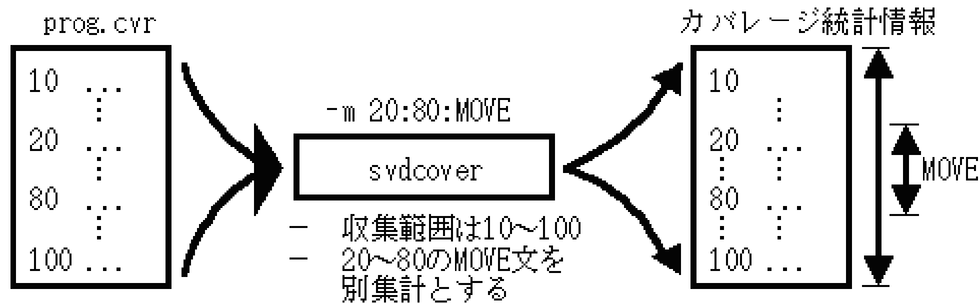


刻印文指定(-m)

“-m”オプションは、指定した実行文に対して刻印をし、その指定した文(刻印文)について、カバレージ統計情報を集計します。指定方法を以下に示します。

```
-m  {verb1[ : verb2]… | start : end[ : verb3]…}
```

刻印文(verb1, verb2..)は複数指定することができます。また、開始行番号(start)、終了行番号(end)により、行番号で範囲指定することもできます。さらに刻印文(verb3)を記述すると、startからendまでの刻印文(verb3)を集計することができます。



刻印文には、カバレージ統計情報の“文種別”に出力される文キーワードを指定することができます。文キーワードについては、“23.9.2.8 文キーワード”を参照してください。

また、刻印文として総称名を指定することができます。刻印文に指定できる総称名を以下に示します。

総称名	指定文
"IO"	OPEN, READ, WRITE, REWRITE, START, DELETE, CLOSE
"ARITH"	ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT
"CHAR"	STRING, UNSTRING, INSPECT

なお、“EXIT PROGRAM”など空白を含む文キーワードを指定する場合には、両端を”で囲む必要があります。

例

STRING, UNSTRING, INSPECT, ADD, EXIT PROGRAMを刻印文に指定する。

```
-m "CHAR":ADD:"EXIT PROGRAM"
```

その他の指定

その他のカバレージコマンドのオプションを以下に示します。

[-a]

複数の外部プログラムをアルファベット順に出力します。省略した場合、実行順に出力します。

[-c]

行情報の回数を降順に出力します。省略した場合、行番号順に出力されます。

[-h]

複数の外部プログラムを時間情報の平均実行時間の降順に出力します。カバレージレベル2のカバレージ情報が収集されている場合にだけ有効となります。カバレージレベル1のカバレージ情報が収集されている場合に指定すると無効となります。当オプションを省略すると実行順に出力されます。

[-p プログラム名]

指定した外部プログラムのカバレージ統計情報を出力します。

[-s]

行情報の出力を抑制します。

[-t]

行情報の実行時間を降順に出力します。カバレージレベル2のカバレージ情報が収集されている場合に有効となります。カバレージレベル1のカバレージ情報が収集されている場合に指定すると無効となります。当オプションを省略すると行番号順に出力されます。

[-u]

複数の外部プログラムを平均実行回数の降順に出力します。

[-V]

カバレージコマンドのバージョン情報を出力します。

ほかのオプション指定はすべて無視されます。

[-x]

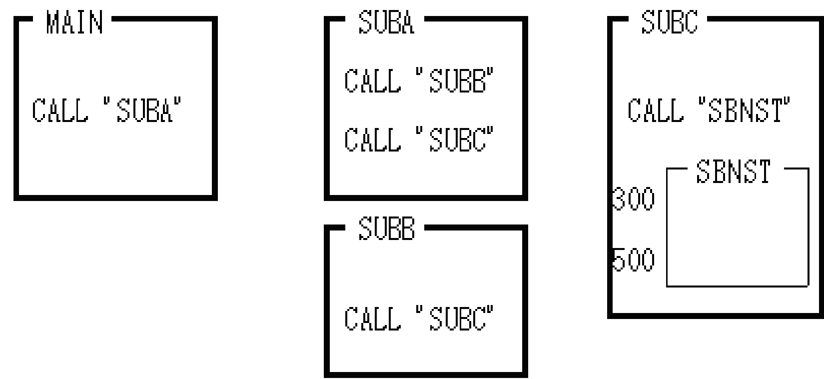
外部プログラムごとに総実行回数および総実行時間(カバレージレベル2の場合)の大きい順にグラフ表示します。当オプションを指定すると、ほかのオプションの指定はすべて無効となります。

23.7.2 カバレージコマンド使用例

以下にカバレージコマンドの使用例を示します。

ここではカバレージレベル1でカバレージ情報を収集した場合を想定しています。

チューニング対象プログラム



翻訳単位は、MAIN, SUBA, SUBB, SUBCとする。
また、カバレージファイルはprog.cvrとする。

チューニング例

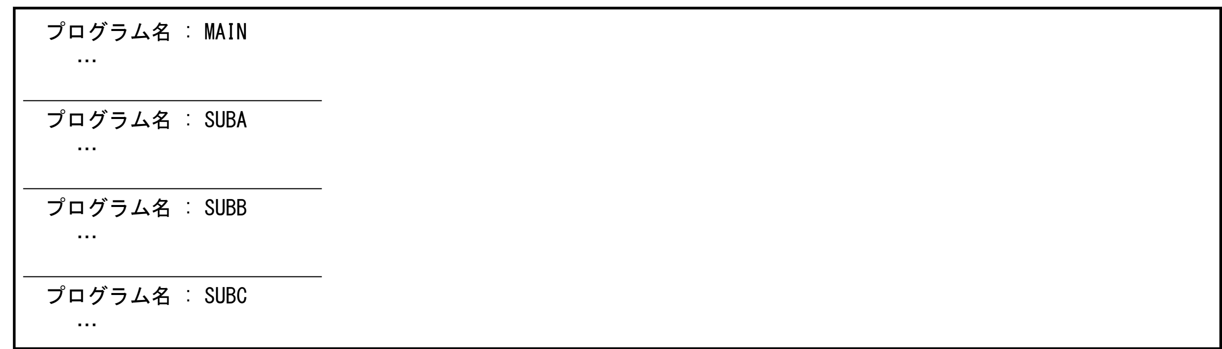
上記例についてチューニングまでの流れを以下に示します。

1. プログラムごとのカバレージ統計情報を出力します。

   - コマンド指定

```
$ svdcover prog.cvr
```

   - 出力概要



```
プログラム名 : MAIN
   …
_______________________
プログラム名 : SUBA
   …
_______________________
プログラム名 : SUBB
   …
_______________________
プログラム名 : SUBC
   …
```

   - 次のステップへ

     いちばん多く実行したプログラムに的を絞ります。まず、行情報を除いたカバレージ統計情報を出力します。

2. “-s”オプションにより、サマリー情報を出力します。これにより、行情報の出力は抑止されカバレージ統計情報が見やすくなります。

   - コマンド指定

```
$ svdcover -s prog.cvr
```

   - 出力概要

```
プログラム名 : MAIN
   …
_______________________
プログラム名 : SUBA
   …
```

```
プログラム名 : SUBB
   …

プログラム名 : SUBC
   …
<実行情報>
  被呼出し回数 (C)     50回
  総実行回数   (T)   1234回
  …
```

－ 次のステップへ

プログラムSUBCの実行回数および呼出し頻度が大きいことがわかります。そこでプログラムSUBCについてカバレージ統計情報を出力します。

3. “-p”オプションによりプログラムSUBCの情報を出力します。

－ コマンド指定

```
$ svdcover -p SUBC prog.cvr
```

－ 出力概要

```
プログラム名:SUBC
   …
<行情報>
    行番号 …  回数  比率(%)   文種別
    …
      300      254    12        MOVE
      301      122     6        IF
      302       24     1        PERFORM
      305      254    12        MOVE
    …
<内部プログラム情報>（呼出し順）
   プログラム名: SBNST
     入口点              300 行
     被呼出し回数        254 回
```

－ 次のステップへ

プログラムSUBCで、内部プログラムSBNSTの呼び出しが多いことがわかります。プログラムSUBCでの内部プログラムSBNSTの実行頻度を調べます。

4. “-m”オプションによりプログラムSUBC中に占めるプログラムSBNSTの頻度を調べます。内部プログラムの範囲は300行から500行です。

－ コマンド指定

```
$ svdcover -p SUBC -m 300:500 prog.cvr
```

－ 出力概要

```
刻印範囲: 300    ←行→   500 ----> *
    …
  プログラム名:SUBC
<行情報>
    行番号 …  回数   比率(%)   刻印
    …
      298        1       0
      300      254      12        *
      301      122       6        *
      302       24       1        *
      500      254      12        *
      501        1       0
    …
```

```
＜実行情報＞
  …
 実行率
    入出力文(p/T)       1 %
    刻印文 (m/T)       54 %
  …
```

ー 次のステップへ

プログラムSBNSTがプログラムSUBCで占める割合が高く、ここがチューニングポイントであるとわかります。今度はプログラムSUBCに占める割合でなく、プログラムSBNSTに占める動詞ごとの実行率を求めます。

5. "-n"オプションによりプログラムSBNSTに占める動詞ごとの頻度を調べます。
プログラムSBNSTのMOVE文およびPERFORM文の実行回数が多いため、プログラムSBNST(300行～500行)の範囲を100%にした場合の走行率および実行率情報を求めます。

ー コマンド指定

```
$ svdcover -p SUBC -n 300:500 -m PERFORM : MOVE prog.cvr
```

ー〔出力概要〕

```
刻印文  :  PERFORM,MOVE     ---->   *
     …
プログラム名 :SUBC
 <行情報>
   行番号 …   回数   比率(%)   文種別
     300       25      *      MOVE
     301       12             IF
     302       25      *      PERFORM
     305        0      *      MOVE
     …
     500        0             IF
   …
＜実行情報＞  集計範囲  300  ←行→  500
   …
  実行率
     入出力文(p/T)       0 %
     刻印文 (m/T)       68 %
   …
```

ー 次のステップへ

プログラムSBNST中に占めるPERFORM, MOVEの実行回数が多いため、300行から500行までのロジックを見直します。

6. PERFORM文のループ内に必要のないMOVE文が多いため、MOVE文を外に出すことでチューニングを行います。

23.8 Interstageなどのサーバ環境で動作するプログラムのデバッグ

23.8.1 概要

デバッグしたいプログラムからデバッガを起動してデバッグを行うことができます。COBOLプログラムを実行するとデバッガが起動され、スクリーンモードの場合にはメインウィンドウ、ラインモードの場合にはプロンプトが表示されてデバッグを開始することができます。この機能は、InterstageやWebサーバなどの環境下で動作するCOBOLアプリケーションのデバッグに使用します。

23.8.2 指定方法

デバッグプログラムからデバッガを起動する手順を以下に示します。

1. デバッグプログラム実行時の環境変数の設定
2. デバッガのウィンドウを表示するためのXサーバへの接続許可(この操作はリモートデバッグでは不要です)
3. デバッグプログラムの起動

これによりデバッガが起動されます。

23.8.2.1 環境変数の指定

デバッグプログラムからデバッガを起動するためには、以下の環境変数を指定する必要があります。

CBR_ATTACH_TOOL=[接続先/TEST] [起動オプション]

接続先

接続先を指定した場合は、リモートデバッガを使用します。接続先には、クライアント側のリモートデバッガコネクタが動作しているコンピュータとポート番号を以下の形式で指定します。

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、Windows版NetCOBOLに含まれる“NetCOBOL Studio使用手引書”または“NetCOBOL 使用手引書”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。また、“TEST”はリモートデバッグを表す文字列として指定します。接続先を指定した場合は、省略できません。NetCOBOL Studioからのリモートデバッグである場合は、“TEST”の代わりに“STUDIO”を指定します。

例

〔IPv4アドレス(192.168.0.1)およびポート番号(2000)での指定〕
```
CBR_ATTACH_TOOL="192.168.0.1:2000/TEST"
```

〔IPv6アドレス(fe80::1:23:456:789a)およびポート番号(2000)での指定〕
```
CBR_ATTACH_TOOL="[fe80::1:23:456:789a]:2000/TEST"
```

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

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

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

起動オプション

実行形式ファイル名以降を除くsvdコマンドのオプションを指定します。起動オプションについては、“表23.1 デバッグに関するオプション”を参照してください。

NetCOBOL Studioからのリモートデバッグを指定する場合は、“起動オプション”の代わりに“追加パスリスト”を指定します。“追加パスリスト”については、Windows版NetCOBOLに含まれる“NetCOBOL Studio使用手引書”を参照してください。

例

〔デバッグ情報格納ディレクトリ(debuginfodir)の指定〕

```
CBR_ATTACH_TOOL="-k debuginfodir"
```

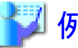
- スクリーンモードでデバッガを起動する場合は、環境変数CBR_ATTACH_TOOLの起動オプションに-wオプションを必ず指定してください。
- ラインモードでデバッガを起動する場合は、環境変数CBR_ATTACH_TOOLの起動オプションに-rオプションを必ず指定してください。
- デバッグ情報ファイル格納ディレクトリなどの必要な情報は、あらかじめ環境変数CBR_ATTACH_TOOLの起動オプションに指定しておく必要があります。
- 環境変数CBR_ATTACH_TOOLの指定に誤りなどがある場合は、デバッガのエラーメッセージが標準エラー出力に出力されます。

更に、InterstageやWebサーバなどの環境下で動作するCOBOLアプリケーションを、svdコマンドを使ってデバッグするには、以下の環境変数を指定する必要があります。

DISPLAY=[host]:server[.screen]

デバッガのウィンドウを表示するディスプレイを指定します。環境変数DISPLAYについては、X Window Systemのマニュアルまたは参考書を参照してください。

サーバアプリケーションが動作するホストにデバッガのウィンドウを表示する場合は、以下のように指定します。



```
DISPLAY=:0
```

SVD_TERM=Xウィンドウ端末エミュレータ

実行時のコード系がUnicodeの場合や、ラインモードでデバッガを起動する場合にXウィンドウ端末エミュレータを指定します。標準的なXウィンドウ端末エミュレータには、xtermやdttermがあります。

dttermを使う場合は、以下のように指定します。

```
SVD_TERM=dtterm
```

この環境変数を指定した場合、-eオプションにsvdコマンドが指定されて、指定したXウィンドウ端末エミュレータが起動されます。

23.8.2.2 Xサーバへの接続許可

InterstageやWebサーバなどの環境下で動作するCOBOLアプリケーションから起動されたデバッガのウィンドウを表示するために、Xサーバへの接続を許可します。サーバアプリケーションが動作するホストとデバッガのウィンドウを表示するホストが同じ場合にもこの操作を行う必要があります。Xサーバへの接続を許可するには、xauthコマンドやxhostコマンドを使用します。xauthコマンドとxhostコマンドについては、X Window Systemのマニュアルまたは参考書を参照してください。

xhostコマンドによってホスト名がmyhostのホストからのXサーバへの接続を許可する場合の例

```
$ xhost  +myhost
```

デバッグが終了したら、Xサーバへの接続許可を解除します。

xhostコマンドによってホスト名がmyhostのホストからのXサーバへの接続許可を解除する場合の例

```
$ xhost  -myhost
```

23.8.3 注意事項

デバッグプログラムからデバッガを起動した場合の注意事項を以下に示します。

利用者プログラム標準入出力ウィンドウの制限

利用者プログラム標準入出力ウィンドウでの入力、出力ができません。デバッグプログラムでのデータ入出力は、デバッガの起動時の"-w"オプションを指定して、デバッグプログラムを起動したウィンドウで行ってください。

デバッガ終了時の動作

デバッガの終了(QUIT)時には、デバッガはすべてのブレークポイントを解除し、デバッグプログラムをデバッガから解放します。したがって、デバッガの終了後は、デバッグプログラムはそのまま処理を続行します。

繰り返しデバッグを行う場合の注意事項を以下に示します。

再デバッグ、読み込みの禁止

デバッグプログラムからデバッガを起動した場合は、再デバッグ機能は使用できません。また、読み込みウィンドウを使用したデバッグプログラムの再起動も行えません。繰り返しデバッグを行う場合は、再度デバッグプログラムを起動してください。

Webサーバなどから起動されるサーバアプリケーションの場合

Webサーバなどから起動されるサーバアプリケーションのデバッグの場合は、一度デバッガを終了させると、その後のサーバアプリケーションの呼び出しでデバッガが起動されないことがあります。このような場合に、繰り返しデバッグを行うときは、サーバ自体を再起動させてください。

23.9 サブコマンド機能

ここでは、サブコマンドについて説明します。サブコマンドはラインコマンド入力ウィンドウからの入力や自動デバッグで指定するコマンドファイルで使用します。

"表23.2 デバッガのサブコマンド一覧"にデバッガのサブコマンド一覧とラインモードからの入力の可否を示します。

ラインモードからの入力ができないサブコマンドは、スクリーンモードのラインコマンド入力ウィンドウからだけ指定することができます。

表23.2 デバッガのサブコマンド一覧

サブコマンド名	機能	ラインモード入力可否
AUTORUNサブコマンド	自動デバッグ	指定可
BACKTRサブコマンド	逆トレース	指定可
BREAKサブコマンド	中断点の設定	指定可
CALLSサブコマンド	呼出経路一覧表示	指定可
CONTINUEサブコマンド	実行の再開	指定可
COUNTサブコマンド	カウント点の設定	指定可
DATACHKサブコマンド	条件式の成立の監視	指定可
DELCOUNTサブコマンド	カウント点の解除	指定可
DELDCHKサブコマンド	条件式監視の解除	指定可
DELDTRサブコマンド	データ項目の内容変更監視の解除	指定不可
DELETEサブコマンド	中断点の解除	指定可
DELMONサブコマンド	データ域の内容変更の監視	指定可
DTRACEサブコマンド	データ項目の内容変更の監視(変更時中断しない)	指定不可
ENVサブコマンド	デバッグ環境の設定/変更	指定可
HELPサブコマンド	ヘルプ	指定可

サブコマンド名	機能	ラインモード入力可否
LINKAGEサブコマンド	連絡節のデータ域の獲得	指定可
LISTサブコマンド	データ域の内容参照	指定可
LSEARCHサブコマンド	原始プログラムの表示位置変更	指定不可
MONITORサブコマンド	データ項目の内容変更の監視(変更時中断する)	指定可
OFFSEARCHサブコマンド	未実行文の検索	指定可
PRINTサブコマンド	データ域の内容の印刷	指定可
PROGサブコマンド	表示原始プログラムの変更	指定不可
QUITサブコマンド	デバッグの終了	指定可
RERUNサブコマンド	再デバッグ	指定可
RUNTOサブコマンド	中断点指定実行	指定可
SCOPEサブコマンド	プログラム修飾の変更	指定可
SEARCHサブコマンド	文字列検索	指定不可
SETサブコマンド	データ域の内容変更	指定可
SKIPサブコマンド	実行開始点の変更	指定可
STATUSサブコマンド	デバッグ状態の表示	指定可
STAMPサブコマンド	テスト網羅度測定機能の各種操作	指定可
WHEREサブコマンド	実行開始位置の表示	指定可

23.9.1 記述形式

サブコマンドの構文表記記号など、各サブコマンドに共通の記述形式について説明します。

サブコマンドの構成

サブコマンドは、サブコマンド名とオペランドからなります。サブコマンド名は、サブコマンドの機能を示し、オペランドは、それに必要な情報を与えます。サブコマンド名とオペランド、オペランドとオペランドの区切りには、1つ以上の分離符を用います。分離符には、ブランク“ ”を使用します。

オペランドには、位置オペランドとキーワードオペランドがあります。

位置オペランドは、決められた位置に、決められた順序で入力するオペランドです。位置オペランドは、非終端記号で示されます。位置オペランドにブランクを含む文字列を指定する場合は位置オペランド全体を括弧()で囲んで指定します。

キーワードオペランドは、特定の文字列によって識別されるオペランドです。キーワードオペランドは、終端記号で示されます。サブパラメタを指定する場合には、キーワードの直後に括弧()で囲んで指定します。

サブコマンド名およびキーワードオペランドは、英大文字または英小文字のどちらで入力しても同じ意味をもちます。ほかのオペランドは、翻訳時に翻訳オプションALPHALが有効な場合は、英小文字は対応する英大文字と同様に扱われ、翻訳オプションNOALPHALが有効な場合は、英小文字は英小文字として扱われます。ただし、文字定数を引用する場合には、翻訳オプションの指定とは無関係に、指定されたとおりの文字列として扱われます。

構文表記法

サブコマンドの構文を表記する表記法を以下に説明します。

構文表記記号

構文表記記号とは、構文を記述するうえで、特別の意味が定められた記号です。本書で使用する記号は、本書の“まえがき”を参照してください。

終端記号

終端記号は、決められた文字列をそのままの形で指定します。終端記号は、英文字で表すものと次の記号があります。

((左括弧)　　) (右括弧)　　= (等号)　　: (コロン)　　& (アンパサンド)

＊ （アスタリスク）　＃ （シャープ記号）　. （ピリオド）　－ （ハイフン）

なお、英文字で示される終端記号で、英大文字で記述されている部分は、終端記号の省略形を示します。たとえば、以下のような記述の場合は、“ENTRY”と“ENT”が指定できます。

```
ENTry
```

非終端記号

非終端記号は、実際の値に置換えて指定します。非終端記号は和単語で表します。

23.9.2 共通オペランドの説明

ここでは、デバッガ全体で共通なオペランドについて説明します。

23.9.2.1 プログラム名

プログラム名はサブコマンドの動作の対象となるプログラムを指示するものです。プログラム名は、以下の形式のどれかで指定します。

- 外部プログラム名
- 二次入口名
- 外部プログラム名.内部プログラム名
- 外部プログラム名.*
- 外部プログラム名.内部プログラム名.*
- クラス名:メソッド名
- クラス名:*
- *
- &

外部プログラムは1つの翻訳単位のいちばん外側に記述されたプログラムです。

内部プログラムは外部プログラムに含まれるプログラムです。

二次入口名はENTRY文で定められる入口点の名前です。

ピリオド(.)、コロン(:)などの特殊な文字を含んだプログラム名を指定する場合は、外部プログラム名や内部プログラム名を文字定数で指定してください。

“表23.3 プログラム名の指定形式と有効範囲”に各指定形式での有効範囲について説明します。

表23.3 プログラム名の指定形式と有効範囲

指定形式	有効範囲
外部プログラム名	指定された外部プログラム
ENTRY名	指定された二次入口点二次入口名は各サブコマンドでENTRYオペランドを指定する場合に指定できます。
外部プログラム名.内部プログラム名	指定された内部プログラム
外部プログラム名.*	指定された外部プログラムとそれに含まれるすべての内部プログラム
外部プログラム名.内部プログラム名.*	指定された内部プログラムとそれに含まれるすべての内部プログラム
クラス名:メソッド名	指定されたメソッド定義(ファクトリまたはオブジェクト) (注1)
クラス名:*	指定されたクラス定義に含まれるすべてのメソッド定義
*	すべての外部プログラムとすべての内部プログラム
&	中断位置のプログラム (注2)

注1: 利用者定義のプロパティメソッドを指定する場合は、以下の形式で指定します。

- GETメソッド: _GET_プロパティ名
- SETメソッド: _SET_プロパティ名

注2: 以下の場合、“&”は指定できません。

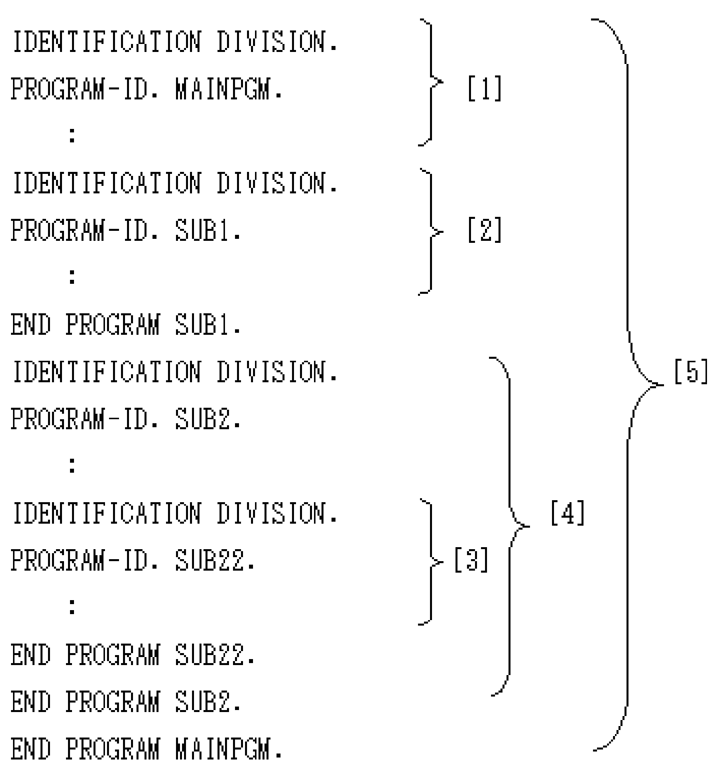
- 動作中のデバッグプログラムが1つもない場合
- 実行が終了した場合

例

プログラム名の指定形式と有効範囲の関係

“図23.12 プログラム名の指定形式の例”のプログラムに対するプログラム名の指定形式と有効となるプログラムの関係は次のようになります。

図23.12 プログラム名の指定形式の例



MAINPGM: MAINPGMだけが有効範囲となります。([1]部分)

MAINPGM.SUB1: SUB1だけが有効範囲となります。([2]部分)

MAINPGM.SUB22: SUB22だけが有効範囲となります。([3]部分)

MAINPGM.SUB2.*: SUB2とSUB22 が有効範囲となります。([4]部分)

MAINPGM.*: MAINPGM, SUB1, SUB2, SUB22 が有効範囲となります。([5]部分)

23.9.2.2 プログラム修飾

プログラム修飾は、サブコマンドに指定する文識別番号および識別名が存在するプログラムを指示するものです。文識別番号と識別名の説明については、それぞれ、“23.9.2.4 文識別番号”および“23.9.2.5 識別名”を参照してください。

プログラム修飾は、サブコマンドのINオペランドで明示的に指定する方法と、INオペランドを指定しないで、現在、有効なプログラム修飾(暗黙プログラム修飾)を使用する方法とがあります。

暗黙プログラム修飾は、プログラムの実行が中断したときに自動的に設定されます。デバッグプログラムで中断した場合には、中断位置のプログラムを暗黙プログラム修飾とします。デバッグプログラム以外で中断した場合には、呼出し経路の中で最も近いデバッグプログラムを暗黙プログラム修飾とします。動作中のデバッグプログラムが確定すれば、暗黙プログラム修飾が決まるため、INオペランドは省略することができます。

暗黙プログラム修飾は、SCOPEサブコマンドによって変更することができます。また、暗黙プログラム修飾は、SCOPEオペランド指定のSTATUSサブコマンドによって表示することができます。INオペランドおよびSCOPEサブコマンドに指定するプログラム修飾は、以下のように指定します。

```
{   外部プログラム名                          }
{   [外部プログラム名].内部プログラム名        }
{   [クラス名]:メソッド名                      }
```

利用者定義のプロパティメソッドを指定する場合は、以下の形式で指定します。

- GETメソッド： _GET_プロパティ名
- SETメソッド： _SET_プロパティ名

ピリオド(.)、コロン(:)などの特殊な文字を含んだプログラム名を指定する場合は、外部プログラム名や内部プログラム名を文字定数で指定してください。

外部プログラム名が省略された場合は、暗黙プログラム修飾の外部プログラム名が有効となります。また、クラス名が省略された場合も暗黙プログラム修飾のクラス名が有効となります。

	プログラム修飾の指定形式(プログラム指定)	
	外部プログラムだけ指定	内部プログラムも指定
行番号	内部プログラムを含むプログラム全体	内部プログラムを含むプログラム全体(内部プログラム名の指定は意味を持たない)
段落名 節名	外部プログラム(内部プログラムは含まない)	内部プログラム
識別名	外部プログラム	内部プログラム(ただし、大域名の場合は親プログラムも含む)

23.9.2.3 行番号および行内識別番号

デバッガでは、COBOLプログラムの行を指定するために行番号を用います。行番号には、エディタ行番号とファイル内相対行番号があります。行番号は、デバッグプログラムの翻訳オプション(NUMBER/NONUMBER)で規定されます。また、1行に複数の文が記述されたときには、行内識別番号を用います。

翻訳オプション	デバッガで指定する行番号
NUMBER	エディタ行番号
NONUMBER	ファイル内相対行番号

エディタ行番号

ソースプログラム中の一連番号領域に記述された番号を指定します。

ファイル内相対行番号

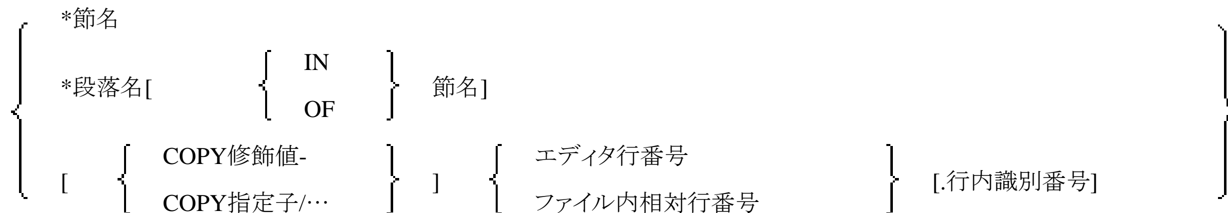
ファイルの先頭からの行数を指定します。

行内識別番号

ソースプログラムの1行に複数の文が記述され、行の先頭以外の文を引用する場合に、行の先頭の文を1とし、順次加算した値を指定します。行に1つの文しか記述されていない場合、または行に複数の文が記述されていても先頭の文を引用する場合には省略することができます。

23.9.2.4 文識別番号

ソースプログラム中の文を引用するための指定であり、以下の形式のどれかで指定します。指定された文識別番号がプログラム修飾の検索範囲内で一意ではない場合は、プログラム修飾の検索範囲内で最初に定義された文識別番号が指定されたものとみなされます。



***節名**

プログラム中に記述した節名の前にアスタリスク(*)を付加したものを指定します。

***段落名[{IN | OF} 節名]**

プログラム中に記述した段落名の前にアスタリスク(*)を付加したものを指定します。

段落名が節名で修飾されない場合、対象となるプログラムが動作中であれば、中断している位置を含む節内の段落名を最初に検索します。そして、次にプログラムの先頭から検索します。節内に同名の段落名がある場合には、節の先頭からみて最初に見つかったものが有効となります。

指定したプログラムが動作中でなければ、プログラムの先頭から検索します。

段落名が節名で修飾された場合は、指定された節内を検索します。節内に同名の段落名がある場合には、節の先頭からみて最初に見つかったものを指定したものとみなされます。節名、段落名による文の指定は、それらで識別される節、段落の先頭の文が指定された場合と同じです。

COPY修飾値-

COPY文によって組み込んだCOBOL登録集内の文を引用する場合に指定します。COPY修飾値は、メインウィンドウや翻訳リストを参照して指定します。COPY修飾値は翻訳時にコンパイラが自動的に付加した番号です。

COPY指定子/…

COPY文によって取り込んだCOBOL登録集内の文を引用する場合に指定します。COPY指定子は、COPY文が記述されている行番号をエディタ行番号か、ファイル内相対行番号で指定します。

例

COPY指定子の指定例

- ソースプログラム

```
                 :
000100           COPY COPYFILE.
000101           MOVE ALL SPACE TO レコード域
                 :
```

- ソースプログラム COPYFILE

```
001000        OPEN INPUT  入力ファイル
```

- COPY指定子の指定方法

  COPYFILEの1000行を指定したい場合、次のように指定します。

```
BREAK 100/1000
```

エディタ行番号

ソースプログラム中の一連番号領域に記述された番号を指定します。

ファイル内相対行番号

ファイルの先頭からの行数を指定します。

.行内識別番号

ソースプログラムの1行に複数の文が記述され、行の先頭以外の文を引用する場合に、行の先頭の文を1とし、順次加算した値を指定します。行に1つの文しか記述されていない場合、または行に複数の文が記述されていても先頭の文を引用する場合には省略することができます。

23.9.2.5 識別名

識別名には、次の5種類を指定することができます。

一意名

デバッガでは、一意名をCOBOLの言語仕様と同様の範囲で指定できます。制約があるのは、添字と部分参照子の部分です。

添字および部分参照子は以下の形式で指定できます。

－ 添字

```
{整数|一意名[{+|-}整数]|指標名[{+|-}整数]|記号定数[{+|-}整数]}
```

－ 部分参照子

```
      (最左端文字位置:[長さ])
最左端文字位置/長さ
            {整数|一意名[{+|-}整数]|記号定数[{+|-}整数]}
```

また、定義済みオブジェクト一意名は、COBOLの言語仕様からSUPERを除いた範囲で指定できます。さらにデバッガでは、ファクトリ定義およびオブジェクト定義に記述されたデータを外部から扱えるように、一意名の指定形式をデバッガ独自に拡張しています。

一意名は以下の形式で指定できます。オブジェクト参照修飾子が、デバッガ独自に拡張された部分です。

－ 一意名

```
[{ポインタ修飾子|オブジェクト参照修飾子}]データ名[{ IN | OF } データ名]…
[{ IN | OF }{ファイル名|報告書名}][({添字} …)][部分参照子]
```

－ ポインタ修飾子

```
[(]…{[オブジェクト参照修飾子]{データ名[{ IN | OF }データ名]…|
(データ名[{ IN | OF }データ名]…({添字}…))}
|ADDR関数}->[{データ名[{ IN | OF }データ名 ]…|
(データ名[{ IN | OF }データ名 ]…({添字}…))})->]…
```

－ オブジェクト参照修飾子

```
[クラス名[スーパークラス指定子]::][オブジェクト参照項目
[{ IN | OF }データ名]…[({添字}…)][スーパークラス指定子]::]…
```

－ スーパークラス指定子

```
ASクラス名
```

オブジェクト参照修飾子は、ファクトリ定義またはオブジェクト定義に記述されたデータを外部から扱う場合に指定します。スーパークラス指定子は、スーパークラスのデータを識別する場合にそのスーパークラス名を指定します。

条件名

条件名の記述形式は添字を除いてCOBOLの言語仕様に従います。

添字は、整数、一意名または指標名で指定します。

指標名

指標名の記述形式はCOBOLの言語仕様に従います。

特殊レジスタ

特殊レジスタの指定形式および設定可能な値については、COBOLの言語仕様に従います。

記号定数

記号定数の記述形式はCOBOLの言語仕様に従います。

23.9.2.6 定数

COBOLの言語仕様と同様の形式で定数を記述することができます。ただし、次の定数を記述することはできません。

[ALL]記号文字

定数を記述するときに使用するクォーテーションマーク(”)とアポストロフィ(’)は、どちらも使用することができます。ただし、1つの定数に対して混在して使用することはできません。

23.9.2.7 式

式には、次の2種類を指定することができます。

代入式の規則

代入式は以下の形式で記述できます。

```
一意名　=　一意名
一意名　=　定数
指標名　=　指標名
指標名　=　一意名
指標名　=　整数
```

代入規則はCOBOLの言語仕様に従って行います。ただし、数字項目に16進文字定数を指定することができます。その場合には、指定された16進文字定数と数字項目のメモリ上で占めるバイト数は同じである必要があります。指定された16進文字定数はメモリ上に左から右に順に指定された内容そのままで格納されます。

代入式の指定は“=”の前後に、空白を記述する必要はありません。

条件式の規則

条件式として、以下の形式を記述することができます。ただし、定数と定数の比較を行う条件式は記述できません。

```
             { [IS] [NOT] > }
{ 一意名 }   { [IS] [NOT] < }   { 一意名 }
{ 定数   }   { [IS] [NOT] = }   { 定数   }
{ 指標名 }   { [IS] >=      }   { 指標名 }
             { [IS] <=      }
```

23.9.2.8 文キーワード

文キーワードは、デバッガで認識できる以下のCOBOLの文を意味します。

- 条件文(条件文を記述していないEVALUATE文を除く)
- 以下の文を除く無条件文
  - ENTRY文
  - 書き方1の、うちPERFORM文
  - NO LIMIT指定のPERFORM文
- END-PERFORM(書き方1の、うちPERFORM文を除く)
- EVALUATE文のWHEN指定(WHEN OTHER指定及びWHEN ANY指定は除く)
- NEXT SENTENCE指定

- プログラム終わり見出し(END PROGRAM)
- メソッド終わり見出し(END METHOD)

23.9.2.9 再帰呼出しレベル

再帰呼出しレベルは、再帰的に呼び出しているプログラムのデータを表示、比較、代入、印刷する際に指定します。このためメソッドに対してだけ指定することができます。再帰呼出しレベルは、いちばん始めに呼び出されたプログラムを1とした追番で、1～4294967295の範囲で指定します。

23.9.3 サブコマンドの説明

ここでは、サブコマンドの機能、記述形式およびオペランドについて説明します。

AUTORUNサブコマンド

AUTORUNサブコマンドは、あらかじめ作成しておいたコマンドファイルを元に、自動的にデバッガを動作させる機能です。デバッガの操作履歴機能を使用して作成したファイルを入力とすることもできます。コマンドファイルについては、"23.5.6 デバッグ作業を自動化する"を参照してください。

サブコマンド	オペランド
AUTORUN AUTO	ファイル名

ファイル名

自動デバッグ用のデータが格納されているコマンドファイルを指定します。

BACKTRサブコマンド

BACKTRサブコマンドは、実行したCOBOL文に関するトレース情報を、逆順に表示します。トレースの情報をデバッガで採取するには、ENVサブコマンドでBTRオペランドを指定する必要があります。

なお、トレース情報は最大1000まで採取することができます。

サブコマンド	オペランド
BACKTR BTR	[{ STMt (下線) / Proc } [({ * (下線) / 整数 })]]

STMT

文単位の情報を表示します。

PROC

手続き名単位の情報を表示します。

*

採取したすべての情報を表示します。

整数

表示するトレース情報の数を指定します。

注意

このサブコマンドはラインモードでデバッガを起動した場合に使用できます。スクリーンモードのラインコマンド入力ウィンドウからは指定できません。

BREAKサブコマンド

BREAKサブコマンドは、デバッグプログラムの実行を中断する位置を示す中断点を指定します。ただし、オブジェクトが生成されない以下の文に中断点を設定するとエラーとなります。

- CONTINUE文
- 選択主体に条件式以外を記述したEVALUATE文など

中断点の情報は、BREAKオペランド指定のSTATUSサブコマンドで表示できます。

以下の条件の場合、設定されている中断点は無効になります。

- DELETEサブコマンドの実行
- 同じ位置に対して、BREAKサブコマンドでの中断点の設定
- デバッガの終了
- BREAK(CANCEL)オペランド指定のRERUNサブコマンドの実行

注意

中断点の設定指示を記憶する際に誤りが検出されれば、エラーメッセージが出力されます。仮想記憶領域上に存在しないプログラムに対し、中断点を設定した場合、中断点の設定指示は記憶されるだけです。STATUSサブコマンドの結果では、“(保留)”が表示され、ほかの設定済みの中断点とは区別されます。実際に設定されるのは、該当するプログラムが仮想記憶領域上へローディングされた時点です。このため、プログラムのローディング時に検出される誤りもあります。

サブコマンド	オペランド
BREAK B	{ 文識別番号 [ＩＮ (プログラム修飾)] [ＣＯｎｄ (条件式)] [ＣＹｃｌｅ (整定数)] { ＥＮＴｒｙ / ＥＸｉｔ / ＳＴＭｔ (文キーワード並び) } [Ｐｒｏｃ ({ プログラム名並び / ＆ / ＊ })] [ＣＯｎｄ (条件式)] [ＣＹｃｌｅ (整定数)] }

文識別番号

実行を中断する位置を指定します。文識別番号については、“23.9.2.4 文識別番号”を参照してください。

IN(プログラム修飾)

文識別番号を指定する場合に、文識別番号が存在するプログラムを指定します。

ENTRY

プログラムの入口に中断点を設定する場合に指定します。

EXIT

プログラムの出口に中断点を設定する場合に指定します。

STMT(文キーワード並び)

指定した文キーワードを持つ文に中断点を設定します。中断点は、指定したプログラム内の指定した文キーワードを持つ文すべてに設定されます。

文キーワード並びを指定する場合は、文キーワードと文キーワードの間に“,”(コンマ)を用いて指定します。文キーワードについては、“23.9.2.8 文キーワード”を参照してください。

ENTRY, EXITおよびSTMTは、構文表記上ではキーワードオペランドの形式です。しかし、ここでは、位置オペランドとして扱うため、オペランドの最初に指定する必要があります。

PROC({プログラム名並び|&|*})

ENTRY, EXITまたはSTMTオペランドを指定する場合に、中断点を設定するプログラムを指定します。ENTRY, EXITおよびSTMTオペランドの指定時の、PROCオペランドの省略値は、PROC(*)となります。

CONDオペランドを同時に指定する場合、PROCオペランドで指定できるプログラムは1つだけです。

COND(条件式)

指定された条件が満たされるときに中断する中断点を設定します。指定された条件式が評価できない場合、評価結果は"偽"となります。

CYCLE(整定数)

通過した回数により中断する中断点を設定します。CYCLEオペランドが省略された場合には、中断点に制御が到達するたびに中断します。

CALLSサブコマンド

CALLSサブコマンドは、最初に呼び出されたプログラムから、現在の中断位置までのプログラムの呼出し経路を表示します。先頭行に表示されているプログラム名が現在中断しているプログラムであり、以降、現在位置表示に近いプログラムの呼出し順番に従って、プログラム名が表示されます。なお、デバッグ対象外のプログラムが呼び出された場合には、"(デバッグ対象外)"が表示されます。

サブコマンド	オペランド
CALLS CALL	なし

CONTINUEサブコマンド

CONTINUEサブコマンドは、中断している位置から実行を再開します。ただし、実行終了による中断および、再帰呼出しエラーにより中断した場合には、CONTINUEサブコマンドで実行を再開することはできません。

サブコマンド	オペランド
CONTINUE CONT C	なし

COUNTサブコマンド

COUNTサブコマンドは通過回数を記録する位置(カウント点)を設定します。

通過回数は、COUNTオペランドを指定したSTATUSサブコマンドを使用することで表示することができます。ただし、オブジェクトコードが生成されない文にカウント点を設定するとエラーとなります。

次の場合、設定したカウント点は無効になります。

- DELCOUNTサブコマンドの実行
- 同じ位置に対するCOUNTサブコマンドの実行
- デバッガの終了

- COUNT(CANCEL)指定のRERUNサブコマンドの実行

同一の中断対象文に複数回カウント点設定をした場合は、最後に設定したものが有効となり、通過回数は0からカウントします。

注意

カウント点の設定指示を記憶する際に誤りが検出されれば、エラーメッセージが出力されます。仮想記憶領域上に存在しないプログラムに対し、カウント点を設定した場合、カウント点の設定指示は記憶されるだけです。STATUSサブコマンドの結果では、“(保留)”が表示され、ほかの設定済みのカウント点とは区別されます。実際に設定されるのは該当するプログラムが仮想記憶領域上へローディングされた時点です。このため、プログラムのローディング時に検出される誤りもあります。

サブコマンド	オペランド
COUNT CO	{ 文識別番号 [IN (プログラム修飾)] { ENTry | EXit | STMt (文キーワード並び) } [Proc ({ プログラム名並び | & | <u>*</u> })] }

文識別番号

カウント点を設定する位置を指定します。文識別番号については、“23.9.2.4 文識別番号”を参照してください。

IN(プログラム修飾)

文識別番号を指定する場合に、文識別番号が存在するプログラムを指定します。

ENTRY

プログラムの入口にカウント点を設定する場合に指定します。

EXIT

プログラムの出口にカウント点を設定する場合に指定します。

STMT(文キーワード並び)

指定した文キーワードを持つ文にカウント点を設定します。カウント点は、指定したプログラム内の指定した文キーワードを持つ文すべてに設定されます。

文キーワード並びを指定する場合は、文キーワードと文キーワードの間に“,”(コンマ)を用いて指定します。文キーワードについては、“23.9.2.8 文キーワード”を参照してください。

注意

ENTRY, EXITおよびSTMTは、構文表記上ではキーワードオペランドの形式です。しかし、ここでは、位置オペランドとして扱うため、オペランドの最初に指定する必要があります。

PROC({プログラム名並び|&|*})

ENTRY, EXITまたはSTMTオペランドを指定する場合に、カウント点を設定するプログラムを指定します。

注意

ENTRY, EXITおよびSTMTオペランドを指定し、PROCオペランドを省略した場合は、PROC(*)を指定したものとみなされます。

DATACHKサブコマンド

DATACHKサブコマンドは指定された条件式を監視する機能であり、条件が成立したときにプログラムの実行を中断します。

サブコマンド	オペランド
DATACHK DCHK	条件式　[IN(プログラム修飾)] [Level({再帰呼出しレベル | LATEST})]

条件式の評価は、条件式で指定された識別名の値が変化した直後の中断対象文の実行前に行います。条件式の評価範囲は、全プログラムが対象となります。

評価時に以下のエラーが発生した場合、評価結果は“偽”となります。

- 添字または指標の範囲例外
- 部分参照の範囲例外
- アドレス例外

注意

評価結果が“偽”から“真”に変化したとき、中断条件は成立します。それ以外(評価結果が一定、評価結果が“真”から“偽”に変化)は不成立となります。

条件式

中断条件となる条件式を指定します。条件式を構成する識別名がプログラムの動作中だけ参照可能なデータの場合は、そのプログラムの動作中以外は監視を行いません。条件式の詳細については、“23.9.2.7 式”を参照してください。

指定した条件式が評価時に真の場合、真から偽になり、その後、偽から真になった時点で、はじめて条件が成立します。

注意

条件式の評価は、あとに指定されたものから順に行います。評価順序は、同一箇所で複数の条件式が成立するような場合の、情報の出力順序にだけ影響があります。

IN(プログラム修飾)

条件式を構成する識別名の存在するプログラムを指定します。

対象となるプログラムがメモリ上に存在しない場合、プログラムがメモリ上に存在するまで監視が延期されます。

LEVEL({再帰呼出しレベル|LATEST})

プログラムの再帰呼出しレベルを指定します。LATESTが指定された場合は常に最新レベルのデータを対象にします。このオペランドを省略した場合は、LATESTを指定したものとみなされます。

DELCOUNTサブコマンド

DELCOUNTサブコマンドは、設定されたカウント点を解除します。

注意

- カウント点の解除指示を記憶する際に誤りが検出されれば、エラーメッセージが出力されます。仮想記憶領域上に存在しないプログラムに対し、カウント点を解除した場合、カウント点の解除指示は記憶されるだけです。実際に解除されるのは、該当するプログラムが仮想記憶領域上へローディングされた時点です。このため、プログラムのローディング時に検出される誤りもあります。

- ENTRY, STMT, EXITおよび*は、構文表記上は、キーワードオペランドの形式です。しかしここでは、位置オペランドとして扱うため、オペランドの最初に指定する必要があります。

サブコマンド	オペランド
DELCOUNT DCO	{ 文識別番号 [IN(プログラム修飾)] [Clear] { ENTry / EXit / STMt(文キーワード並び) } / * [Proc({プログラム名並び / & / <u>*</u>})] [Clear] }

文識別番号

カウント点を解除する位置を指定します。文識別番号については、"23.9.2.4 文識別番号"を参照してください。

IN(プログラム修飾)

文識別番号を指定する場合に、文識別番号が存在するプログラムを指定します。

ENTRY

プログラムの入口のカウント点を解除する場合に指定します。

EXIT

プログラムの出口のカウント点を解除する場合に指定します。

STMT(文キーワード並び)

指定した文キーワードを持つ文のカウント点を解除します。カウント点は、指定したプログラム内の指定した文キーワードを持つ文すべてが解除されます。

文キーワード並びを指定する場合は、文キーワードと文キーワードの間に","(コンマ)を用いて指定します。文キーワードについては、"23.9.2.8 文キーワード"を参照してください。

*

プログラムに設定されているすべてのカウント点を解除します。

PROC({プログラム名並び|&|*})

ENTRY, EXITまたはSTMTオペランドを指定する場合に、カウント点を解除するプログラムを指定します。

ENTRY, EXITおよびSTMTオペランドを指定し、PROCオペランドを省略した場合は、PROC(*)を指定したものとみなされます。

CLEAR

有効となっているカウント点の通過回数をゼロクリアします。

DELDCHKサブコマンド

DELDCHKサブコマンドはDATACHKサブコマンドで設定した中断条件を解除します。

サブコマンド	オペランド
DELDCHK DDCHK	なし

DELDTRサブコマンド

DELDTRサブコマンドはデータトレース機能を解除します。なお、DELDTRサブコマンドは、ラインモードでは使用できません。

サブコマンド	オペランド
DELDTR DDTR	{ シーケンス番号 * }

シーケンス番号

データ監視を解除したい番号を、データ監視ウィンドウのシーケンス番号欄を参照して指定します。

*

すべてのデータトレースを解除したい場合に指定します。

注意

*を指定するとMONITORサブコマンドで設定したデータ項目の監視も解除されます。

DELETEサブコマンド

DELETEサブコマンドは、設定された実行中断点を解除します。

中断点の解除指示を記憶する際に誤りが検出されれば、エラーメッセージが出力されます。仮想記憶領域上に存在しないプログラムに対し、中断点を解除した場合、中断点の解除指示は記憶されるだけです。実際に解除されるのは、該当するプログラムが仮想記憶領域上へローディングされた時点です。プログラムがローディングされ、中断点を解除する際に検出される誤りもあります。

サブコマンド	オペランド
DELETE D	{ 文識別番号 [IN(プログラム修飾)] [{ Effect NOEffect }] { ENTry EXit STMt(文キーワード並び) * } [Proc({ プログラム名並び & * })] [{ Effect NOEffect }] }

文識別番号

中断点を解除する位置を指定します。文識別番号については、“23.9.2.4 文識別番号”を参照してください。

IN(プログラム修飾)

文識別番号を指定する場合に、文識別番号が存在するプログラムを指定します。

ENTRY

プログラムの入口の中断点を解除する場合に指定します。

EXIT

プログラムの出口の中断点を解除する場合に指定します。

STMT(文キーワード並び)

指定した文キーワードを持つ文の中断点を解除します。中断点は、指定したプログラム内の指定した文キーワードを持つ文すべてが解除されます。

文キーワード並びを指定する場合は、文キーワードと文キーワードの間に“,”(コンマ)を用いて指定します。文キーワードについては、“23.9.2.8 文キーワード”を参照してください。

*

プログラムに設定されているすべての中断点を解除します。

注意

ENTRY, EXIT, STMT,および*は、構文表記上は、キーワードオペランドの形式です。ここでは、位置オペランドとして扱うため、オペランドの最初に指定する必要があります。

PROC({プログラム名並び|&|*})

ENTRY, EXITまたはSTMTオペランドを指定する場合に、中断点を解除するプログラムを指定します。

注意

ENTRY, EXITおよびSTMTオペランドを指定し、PROCオペランドを省略した場合は、PROC(*)を指定したものとみなされます。

EFFECT|NOEFFECT

BREAKサブコマンドで設定した中断点を一時的に無効にする場合には、NOEFFECTを指定します。また、NOEFFECTによって無効になっている中断点を有効にする場合には、EFFECTを指定します。

オペランドの指定(EFFECT|NOEFFECT)があった場合にだけ、この機能は動作します。

中断点を一時無効にした場合は、中断点通過カウンタをゼロクリアし、一時無効になっている間は、中断点通過カウンタは変化しません。

注意

実行中のデバッグプログラムが1つもない場合、または、プログラムの実行状態が、終了後の中断では、PROCオペランドにアンパサンド(&)を指定できません。

DELMONサブコマンド

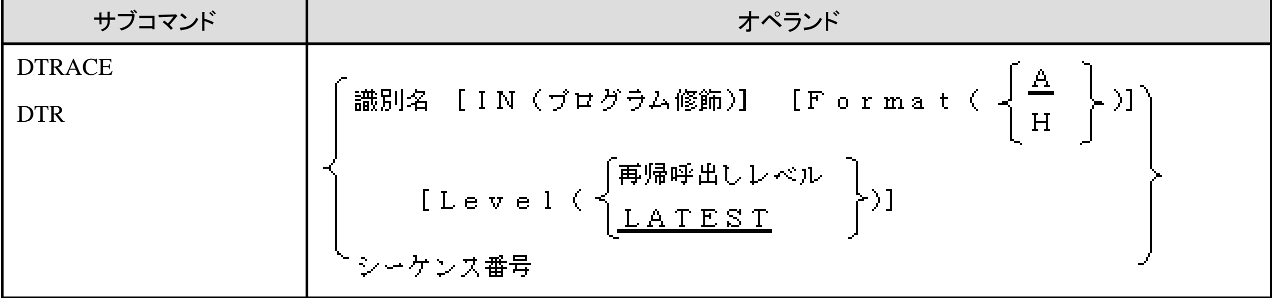
DELMONサブコマンドはMONITORサブコマンドで設定した監視機能を解除する機能です。

サブコマンド	オペランド
DELMON DM	なし

DTRACEサブコマンド

DTRACEサブコマンドは識別名の内容の更新を通知します。このあと、識別名で指定されたデータの値および、アクセスの可否が変わるたびにその内容が通知されます。データトレースの対象範囲は全プログラムです。

なお、DTRACEサブコマンドは、ラインモードでは使用できません。



サブコマンド	オペランド
DTRACE DTR	{ 識別名 [IN(プログラム修飾)] [Format({A|H})] [Level({再帰呼出しレベル|LATEST})] シーケンス番号 }

識別名

データトレースの対象となる識別名を指定します。

IN(プログラム修飾)

指定した識別名が存在するプログラムを指定します。INオペランドの省略時は、暗黙プログラム修飾で示されるプログラムとなります。

FORMAT({A|H})

表示形式を自動形式としたい場合には“A”を、16進形式として表示したい場合には“H”を指定します。

注意

プログラムの動作中だけ参照可能なデータ項目が指定された場合には、そのプログラムの動作中だけ監視されます。

LEVEL({再帰呼出しレベル|LATEST})

プログラムの再帰呼出しレベルを指定します。LATESTが指定された場合は常に最新レベルのデータを対象にします。このオペランドを省略した場合は、LATESTを指定したものとみなされます。

シーケンス番号

データ監視の変更時中断を陽に解除したい(“行わない”)とき、データ監視の番号を指定します。シーケンス番号はデータ監視ウィンドウのシーケンス番号欄に表示されています。

ENVサブコマンド

ENVサブコマンドは、デバッグ環境の操作を行う機能です。

指定が省略されたオペランドのデバッグ環境は、変更されません。また、すべてのオペランドが省略された場合は、現在のデバッグ環境を表示します。

デバッグ環境の表示形式は、デバッガの起動モードにより異なります。

サブコマンド	オペランド
ENV	[{LIB | NOLIB}] [{LINE | NOLINE}] [TRace（整定数）] [{TP（整定数） | NOTP}] [{BTR | NOBTR}] [{Global | Local}] [ABend（置換文字）] [{DBTR（ファイル名） | NODBTR}] [{SAME [({ALL | WORD})] | DIFF}] [EXCEPTion({Break | Continue})]

LIB|NOLIB

登録集の展開を行う(LIB)か、行わない(NOLIB)かを指定します。なお、ラインモードでは無効となります。

LIBオペランドを指定した場合の表示形式を示します。

```
  000070 DISPLAY A.
  000080 COPY PROG.
1-000010 DISPLAY "PROG START".
1-000020 DISPLAY "PROG END".
  000090 MOVE A TO B.
  000100 DISPLAY B.
```

NOLIBオペランドを選択した場合の表示形式を示します。

```
000070 DISPLAY A.
000080 COPY PROG.
000090 MOVE A TO B.
000100 DISPLAY B.
```

LINE|NOLINE

自動デバッグ実行中の動作結果を、ラインコマンド入力ウィンドウに表示する(LINE)か、表示しない(NOLINE)かを指定します。ラインモードでは無効となります。

TRACE(整定数)

追尾速度を指定します。整定数には、0～9の整数が指定可能です。

TP(整定数)|NOTP

追尾または、逆トレース時にPERFORM文内を指定したネストレベルまで表示する(TP(整定数))か、すべてを表示する(NOTP)かを指定します。

TPオペランドに指定する整定数で、ネストレベルを指定できます。ネストレベルの範囲は、1～99の整数です。

BTR|NOBTR

逆トレース情報の採取を行う(BTR)か、行わない(NOBTR)かを指定します。

GLOBAL|LOCAL

追尾実行、および逆トレースを行うときに、呼び出される側のプログラムを操作の対象とする(GLOBAL)か、対象としない(LOCAL)かを指定します。

RUNTOサブコマンドでのGLOBAL|LOCALオペランドの暗黙値を設定します。

ABEND(置換文字)

サブコマンドの出力結果で、表示されない文字を置き換える文字を指定します。置換文字には1文字の英数字が指定できます。

DBTR(ファイル名)|NODBTR

操作履歴の採取を開始(DBTR(ファイル名))するか、終了(NODBTR)するかを指定します。

DBTRオペランドのファイル名に、操作履歴を記録するファイルを指定します。ファイル名には、英数字・日本語混在のファイル名の指定が可能です。

SAME({ALL|WORD})|DIFF

サブコマンドのオペランドに指定するプログラム名およびデータ名の英字の大文字と小文字をデバッガで等価として扱う(SAMEまたはSAME(ALL))か、不等価として扱う(DIFF)かを指定します。なお、当指定はデバッガに対するサブコマンドのオペランドの区別であり、プログラムでの英小文字と英大文字の扱いについては、翻訳オプション(ALPHAL|NOALPHAL)に従います。SAME(WORD)の場合には、データ名だけを等価として扱います。

SAME|DIFFの指定は検索文字列にも有効となります。ただし、ALL|WORDオペランドでの検索区別はできません。

例

LIST A = LIST a (ENV SAME)

→ “LIST a”は“LIST A”を指定したものとみなされます。

LIST A ≠ LIST a (ENV DIFF)

→ “LIST a”と“LIST A”は指定が別となります。

EXCEPTION({BREAK|CONTINUE})

例外事象が発生した場合にプログラムの実行を中断する(BREAK)か、そのまま続行する(CONTINUE)かを指定します。

HELPサブコマンド

HELPサブコマンドはラインモードで使用できるサブコマンドの形式を表示します。オペランドを省略するとラインモードで使用できるサブコマンド一覧を表示します。

サブコマンド	オペランド
HELP H	[{ENV │ RERUN │ Quit │ Continue │ Runto │ SKIP │ Break │ Delete │ STatus │ DataCHK │ DelDCHK │ Monitor │ DelMon │ List │ Set │ PRinT │ LINKage │ Where │ CALLs │ OFFSEArch │ AUTOrun │ COunt │ DelCOunt │ StaMP │ SCope │ BackTR │ Help │ !}]

形式を表示したいサブコマンド名を指定します。

HELPサブコマンドはラインモードでデバッガを起動した場合にだけ使用できます。スクリーンモードのラインコマンド入力ウィンドウからは指定できません。

LINKAGEサブコマンド

LINKAGEサブコマンドは現在中断しているプログラムの連絡節のデータ領域を獲得します。

獲得されたデータ領域は、手続きの終了または、再獲得(同じ連絡節を指定した連絡節の獲得)によって解放されます。

呼出し元プログラムの修正および再翻訳なしに副プログラムのデバッグが可能です。なお、その際に領域獲得に成功したかを示すメッセージがメインウィンドウのインジケータに表示されます。また獲得した領域は、副プログラムから呼出し元プログラムに制御が戻るとき、デバッガにより解放されます。

サブコマンド	オペランド
LINKAGE	識別名[LENgth(項目長)]

サブコマンド	オペランド
LINK	

識別名

データ領域を獲得する連絡節の識別名を指定します。

指定可能な識別名を以下に示します。

- 副プログラムの手続き部の見出しのUSING句の作用対象であるデータ項目
- ENTRY文のUSING句の作用対象であるデータ項目

注意

指定された識別名が可変長集団項目の場合には、指定された識別名の最大長のデータ領域を獲得します。

LENGTH(項目長)

ANY LENGTH句付き項目以外には指定できません。また、ANY LENGTH付き項目では省略できません。項目長には10桁以上指定するとエラーとなります。

LISTサブコマンド

LISTサブコマンドは識別名の内容を表示します。ただし、動作中でないプログラムの連絡節のデータ項目を参照した場合、エラーとなります。

サブコマンド	オペランド
LIST L	識別名　[ＩＮ（プログラム修飾)] [Ｆｏｒｍａｔ（{Ａ (下線) / Ｈ})] [Ｌｅｖｅｌ（再帰呼出しレベル)]

識別名

内容を出力したい識別名を指定します。

識別名の詳細については、“23.9.2.5 識別名”を参照してください。

IN(プログラム修飾)

識別名が存在するプログラムを指定します。

FORMAT({A|H})

表示形式を自動形式としたい場合には“A”を、16進形式としたい場合には“H”を指定します。

注意

- 部分参照について長さが省略された場合は、そのデータ項目の最右端の文字まで表示します。
- 最左端文字位置の値が項目の範囲を超えている場合および、長さの値が項目の範囲を超えている、または最左端文字位置以降の項目の範囲を超えている場合は、エラーとなります。

LEVEL(再帰呼出しレベル)

プログラムの再帰呼出しレベルを指定します。このオペランドを省略した場合は、INで指定されたプログラムの最新の再帰呼出しレベルを指定したものとみなされます。

LSEARCHサブコマンド

LSEARCHサブコマンドはメインウィンドウに表示されているソースプログラムの表示箇所を指定された位置に変更します。なお、LSEARCHサブコマンドはラインモードでは使用できません。

サブコマンド	オペランド
LSEARCH LSEA	{ 行番号 | Top | Last }

行番号

表示したいプログラムの文を引用するための指定です。プログラムに指定された翻訳オプション(NUMBER|NONUMBER)に対応した行番号の形式(エディタ行番号|ファイル内相対行番号)で指定します。

TOP

ソースプログラムの先頭の行をメインウィンドウに表示させたい場合に指定します。

LAST

ソースプログラムの最後の行をメインウィンドウに表示させたい場合に指定します。

MONITORサブコマンド

MONITORサブコマンドは識別名に対する更新を監視する機能です。識別名の値が更新されるとプログラムの実行は中断されます。

MONITORサブコマンドは同時に複数有効とすることはできません。

サブコマンド	オペランド
MONITOR M	識別名 [IN(プログラム修飾)] [Format({A|H})] [Level({再帰呼出しレベル|<u>LATEST</u>})]

識別名

内容を監視したい識別名を指定します。識別名については、"23.9.2.5 識別名"を参照してください。

IN(プログラム修飾)

内容を表示したい識別名の存在するプログラム名を指定します。

FORMAT({A|H})

表示形式を自動形式としたい場合には"A"を、16進形式としたい場合には"H"を指定します。

LEVEL({再帰呼出しレベル|LATEST})

プログラムの再帰呼出しレベルを指定します。LATESTが指定された場合は常に最新レベルのデータを対象にします。このオペランドを省略した場合は、LATESTを指定したものとみなされます。

OFFSEARCHサブコマンド

未実行文のまとまりを1つの検索ブロックとして検索します。また指定した文数ごとに、連続する未実行文を検索ブロックとして検索することもできます。対象となるブロックを検索すると、該当するブロックの先頭の文(検索方向が上方向の場合には最後の文)にカーソルを位置付けます。なお当コマンドは、テスト網羅度測定機能が使用されている場合にだけ有効となります。

サブコマンド	オペランド
OFFSEARCH OFFSEA	{ Block / SEQuence (文数) } { Up / Down / Top / Last }

BLOCK

未実行文のまとまりを1つの検索ブロックとしてブロックごとに検索する場合に指定します。当オペランドは1回目の指定の場合は必須です。しかし、2回目以降は省略可能です。

SEQUENCE(文数)

指定した文数ごとに、連続する未実行文を検索ブロックとして検索する場合に指定します。文数として1以上の値を指定します。当オペランドは1回目の指定の場合は必須です。しかし、2回目以降は省略可能です。省略した場合には前回指定された文数を指定したものとみなされます。

UP

現在のカーソル位置から上方向に検索を行う。

DOWN

現在のカーソル位置から下方向に検索を行う。

TOP

プログラムの先頭の行から下方向に検索する場合に指定します。

LAST

プログラムの最後の行から上方向に検索する場合に指定します。

ラインモードではソースが表示されないため、検索対象のプログラムおよび検索起点がスクリーンモードと異なるので注意してください。

	スクリーンモード	ラインモード
検索対象プログラム	表示中プログラム	中断中プログラム
検索起点	カーソル	現在位置

PRINTサブコマンド

指定した識別名の内容を印刷する機能です。LISTサブコマンドと同じ形式で印刷されます。

サブコマンド	オペランド
PRINT PRT	識別名 [IN (プログラム修飾)] [Format ({ A / H })] [Level (再帰呼出しレベル)] [LpOPT (lpへのパラメタ)]

識別名

内容を印刷したい識別名を指定します。識別名については、“23.9.2.5 識別名”を参照してください。

IN(プログラム修飾)

内容を表示したい識別名の存在するプログラム名を指定します。

FORMAT({A|H})

表示形式を自動形式としたい場合には“A”を、16進形式としたい場合には“H”を指定します。

LEVEL(再帰呼出しレベル)

プログラムの再帰呼出しレベルを指定します。このオペランドを省略した場合は、INで指定されたプログラムの最新の再帰呼出しレベルを指定したものとみなされます。

LPOPT(lpへのパラメタ)

lpコマンドに指定するオプションを空白で区切って指定します。当オペランドの指定が省略された場合は、lpコマンドへのパラメタの指定はないものとみなされます。

注意

印刷されるデータおよび印刷リストのタイトル文字は、デバッガが動作している言語環境に従います。そのため、lpコマンド指定した言語と動作している言語環境が異なる場合、文字化けが発生することがあります。

PROGサブコマンド

PROGサブコマンドは指定したプログラムをメインウィンドウに表示します。なお、PROGサブコマンドはラインモードでは使用できません。

サブコマンド	オペランド
PROC	プログラム名

プログラム名

メインウィンドウに表示したいプログラム名を指定します。

QUITサブコマンド

QUITサブコマンドは、デバッガを終了します。

サブコマンド	オペランド
QUIT Q	[{ ENV / <u>NOENV</u> }] [{ <u>S t aMP</u> / NOS t aMP }]

ENV

デバッガ使用時の動作環境を動作環境ファイル(cobdebug.ini)に保存してデバッガを終了する場合に指定します。

NOENV

デバッガ使用時の動作環境を保存しないでデバッガを終了する場合に指定します。オペランド省略時はNOENVオペランドを指定したものとみなされます。

STAMP

テスト進捗情報をテスト進捗ファイルに保存して、デバッガを終了するときに指定します。なお当オペランドは、テスト網羅度測定機能が使用されている場合にだけ有効となります。

NOSTAMP

テスト進捗情報を破棄するときに指定します。すなわち、テスト進捗情報をテスト進捗ファイルに保存しないでデバッガを終了します。なお当オペランドは、テスト網羅度測定機能が使用されている場合にだけ有効となります。

RERUNサブコマンド

RERUNサブコマンドはデバッグプログラムを終了させ、再度、利用者プログラムのデバッグを開始します。デバッグプログラムに対して設定されている以下の状態の扱いを指示することができます。

- 中断点(BREAK)を保持したままとするかどうか
- 実行監視条件(DATACHK)を保持したままとするかどうか

- カウント点(COUNT)を保持したままとするかどうか
- データ監視(MONITOR, DTRACE)を保持したままとするかどうか
- テスト進捗情報(STAMP)を保存するかどうか

サブコマンド	オペランド
RERUN	[Break ({Keep | Cancel})] [DTRace ({Keep | Cancel})] [DataCHK ({Keep | Cancel})] [COunt ({Keep | Cancel})] [StaMP ({Keep | Cancel})]

BREAK({KEEP|CANCEL})

中断点の状態を保存(KEEP)するか、解除(CANCEL)するかを指定します。KEEPが指示された場合には、次の情報が保存されます。

－ すでにローディングされているプログラムに設定された中断点は、中断点通過カウンタが0に戻ることを除いて、ほかの情報はすべて保存されます。

－ ローディングされていないプログラムに対する中断点設定指示は、すべて保存されます。

DTRACE({KEEP|CANCEL})

データ監視の状態を保存(KEEP)するか、解除(CANCEL)するかを指定します。

DATACHK({KEEP|CANCEL})

実行監視状態を保存(KEEP)するか、解除(CANCEL)するかを指定します。KEEPが指示された場合でも、評価する条件式のプログラムが不在となれば、実行監視条件は解除されます。

COUNT({KEEP|CANCEL})

利用者プログラムに対するカウント点の状態を保存(KEEP)するか、解除(CANCEL)するかを指定します。KEEPが指示された場合には、次の情報が保存されます。

－ すでにローディングされているプログラムに設定されたカウント点は、通過カウンタが0に戻ることを除いて、ほかの情報はすべて保存されます。

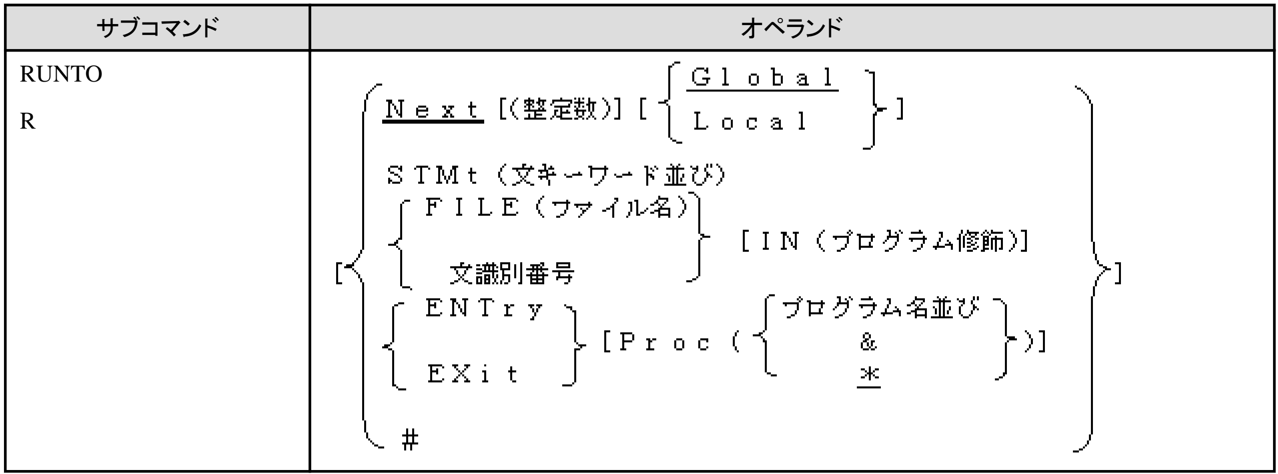
－ ローディングされていないプログラムに対するカウント点設定指示は、すべて保存されます。

STAMP({KEEP|CANCEL})

テスト進捗情報をテスト進捗ファイルに保存(KEEP)するか、破棄(CANCEL)するかを指定します。

RUNTOサブコマンド

RUNTOサブコマンドは指定された任意の文まで実行し、そこで中断する機能です。

サブコマンド	オペランド
RUNTO R	

NEXT(整定数)

整定数で指定された数だけ文を実行します。整定数の省略値は1です。6桁以上指定するとエラーになります。

GLOBAL

ほかのプログラムの文を含めて実行する文の数を数える場合に指定します。

LOCAL

ほかのプログラムの文は含めず、現在中断中のプログラム内の文だけを数える場合に指定します。

STMT(文キーワード並び)

中断中の位置から最初に現れる指定した文キーワードを持つ文の直前まで実行し、該当の文で中断させる場合に指定します。

文キーワード並びを指定する場合は、文キーワードと文キーワードの間に“,”(コンマ)を用いて指定します。文キーワードについては、“23.9.2.8 文キーワード”を参照してください。

FILE(ファイル名)

指定したファイル名を使用している入出力文の直前まで実行し、該当の入出力文で中断することを指定します。

FILEオペランドには、データ部のファイル記述項(FD)で定義されたファイル名、整列併合用ファイル記述項(SD)、および報告書記述項(RD)で定義された報告書名を指定することができます。

入出力文の検索範囲は、外部属性や大域属性を持つファイルであっても、プログラム修飾で示されるプログラム内です。

文識別番号

指定した文まで実行し、中断します。文識別番号については、“23.9.2.4 文識別番号”を参照してください。

IN(プログラム修飾)

FILEオペランドまたは文識別番号オペランドを指定したとき、どのプログラムのファイル名を対象とするかを指定します。

INオペランドの省略時は、暗黙プログラム修飾で示されるプログラムを指定したものとみなされます。

ENTRY

指定したプログラムの入口まで実行し、中断します。入口には、主入口と二次入口を指定することができます。

EXIT

指定したプログラムの出口まで実行し、中断します。

PROC({プログラム名並び|&|*})

ENTRYまたは、EXITオペランドで、どのプログラムまで実行させるかを指定します。

PROCオペランド省略時はすべてのプログラムを指定したものとみなされます。

#

先に、RUNTOサブコマンド(NEXT指定を除く)で指定した位置まで実行することを指定します。

#オペランドは、RUNTOサブコマンドにより実行再開直後の以下の状態のとき、指定可能です。

- BREAKサブコマンドの実行中断点による中断
- DATACHKサブコマンドによる中断
- MONITORサブコマンドによる中断
- 実行時エラーによる中断
- アテンションによる中断

SCOPEサブコマンド

SCOPEサブコマンドは、各サブコマンドで、INオペランドが省略された場合に使用する、暗黙プログラム修飾を定義します。

SCOPEサブコマンドで指定された暗黙プログラム修飾は、SCOPEオペランド指定のSTATUSサブコマンドで参照できます。

注意

SCOPEサブコマンドの入力および、プログラムの実行が再開された場合、以前定義した暗黙プログラム修飾は、無効となります。

サブコマンド	オペランド
SCOPE SC	[プログラム修飾]

プログラム修飾

INオペランドを省略した場合に使用する暗黙プログラム修飾を指定します。

プログラム修飾オペランドを省略した場合は、現在実行が中断しているプログラムを指定したものとみなされます。ただし、動作中のプログラムが1つもない場合は、プログラム修飾オペランドを省略するとエラーとなります。

SEARCHサブコマンド

SEARCHサブコマンドは任意の文字列を検索し、ソースプログラムの表示位置を変更します。なお、SEARCHサブコマンドはラインモードでは使用できません。

サブコマンド	オペランド
SEARCH SEA	"文字列" [{ Up | Down | Top | Last }]

"文字列"

検索したい文字列を指定します。文字列中に二重引用符(")が含まれる場合は2つ連続して指定します。2回目以降は文字列の省略が可能です。

UP

現在中断中の位置から上方向に検索を開始する場合に指定します。

DOWN

現在中断中の位置から下方向に検索を開始する場合に指定します。

TOP

プログラムの先頭の行から下方向に検索を開始する場合に指定します。

LAST

プログラムの最後の行から上方向に検索を開始する場合に指定します。

注意

オペランドで指定された値は、次のオペランドが指定されるまで有効となります。前回指定されたオペランドと条件が同一である場合は、オペランドを省略することができます。

SETサブコマンド

SETサブコマンドは変数の値を変更します。

16進数定数での代入は16進文字定数を使用して行うことができます。

16進数定数を使用するとすべての属性を持つデータ項目の値を変更することができます。16進数定数での代入については“23.9.2.7 式”を参照してください。

注意

動作中でないプログラムの連絡節のデータ項目識別名を指定するとエラーとなります。

サブコマンド	オペランド
SET S	代入式[IN(プログラム修飾)][Level(再帰呼出しレベル)]

代入式

実行したい代入式を指定します。代入式の詳細については“23.9.2.7 式”を参照してください。

注意

代入不可能な、特殊レジスタを識別名の代入式の左辺に指定することはできません。

IN(プログラム修飾)

代入式に指定する識別名が存在するプログラムを指定します。

LEVEL(再帰呼出しレベル)

プログラムの再帰呼出しレベルを指定します。このオペランドを省略した場合は、INで指定されたプログラムの最新の再帰呼出しレベルが指定されたものとみなされます。

SKIPサブコマンド

SKIPサブコマンドは中断中のプログラムの実行の再開位置を変更します。SKIPサブコマンドは再開する位置を変更するだけで、プログラムの実行は行いません。ただし、オブジェクトが生成されない文に対して、実行再開位置を変更するとエラーとなります。

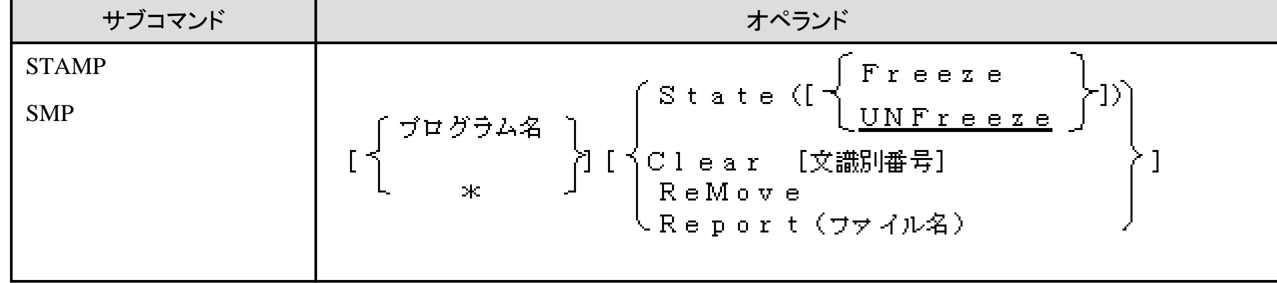
サブコマンド	オペランド
SKIP	文識別番号

文識別番号

次に実行を再開する場合の再開位置を指定します。文識別番号については、“23.9.2.4 文識別番号”を参照してください。

STAMPサブコマンド

テスト網羅度測定機能により測定対象となっているプログラムについて、測定状態の変更および測定対象からの削除を行います。また対象プログラムのテスト進捗情報について、クリアおよびファイルへの出力も行います。なお当コマンドは、テスト網羅度測定機能が使用されている場合にだけ有効となります。

サブコマンド	オペランド
STAMP SMP	

プログラム名|*

テスト網羅度測定機能で測定対象となっている外部プログラム名を指定します。測定対象ではないプログラムを指定するとエラーとなります。*を指定すると測定対象となっているすべての外部プログラムを指定したものとみなされます。

STATE({FREEZE|UNFREEZE})

対象プログラムのテスト進捗情報の状態を測定(UNFREEZE)するか、凍結(FREEZE)するかを指定します。初期値は測定(UNFREEZE)です。

CLEAR

対象プログラムのテスト進捗情報(進捗率、テスト回数、前回テスト実施日、状態)をクリアします。状態は“測定”となります。ただし翻訳時間、更新回数はクリアしません。なお同時に文識別番号も指定した場合は、指定した文を実行済み状態から未実行状態にクリアします。

文識別番号

実行済み状態を未実行状態にする位置を指定します。文識別番号については、“23.9.2.4 文識別番号”を参照してください。なお対象プログラムとして“*”や未ロードプログラムを同時に指定した場合はエラーとなります。

REMOVE

対象プログラムのテスト進捗情報を削除し、測定対象から除外します。当指定は、再デバッグまたはデバッガの終了まで有効です。

REPORT(ファイル名)

指定したファイルに対象プログラムのテスト進捗情報を出力します。

指定したファイルが存在しない場合、ファイルを新規に作成します。ファイルがすでに存在する場合は、テスト進捗情報は追加出力されます。

STATUSサブコマンド

STATUSサブコマンドは、以下の情報を通知します。

- 実行中断点の設定状況
- 通過カウント点の設定状況
- データトレース機能の動作状況
- 暗黙プログラム修飾として使用されるプログラム修飾
- テスト進捗情報の採取状況

サブコマンド	オペランド
STATUS ST	{ Break COunt DataCHK DTRace SCope StaMP }

BREAK

実行中断点の設定状況を表示します。中断点リストの一番下に表示される行番号0の中断点は、すべてのプログラムの入口または出口に中断点が設定されていることを表します。

COUNT

通過カウント点の設定状況を表示します。通過カウント点リストの一番下に表示される行番号0の通過カウント点は、すべてのプログラムの入口または出口に通過カウント点が設定されていることを表します。

DATACHK

実行監視条件設定機能の動作状況を表示します。

DTRACE

データトレース機能の動作状況を表示します。なお、DTRACEオペランドの指定は、ラインモードでは無視されます。

SCOPE

INオペランドを省略したときに使用される暗黙プログラム修飾を表示します。

STAMP

テスト進捗情報の状況を表示します。当オペランドはテスト網羅度測定機能が使用されているときに有効となります。

WHEREサブコマンド

WHEREサブコマンドは実行開始位置をメインウィンドウに表示します。当コマンドは、LSEARCH、SEARCHおよびPROGの各サブコマンドの実行によりソースプログラムの表示位置を変更した場合に使用します。当コマンドを実行することにより、表示位置を実行開始位置に戻すことができます。

サブコマンド	オペランド
WHERE	なし

23.9.4 UNIXコマンド連携

ラインモードの場合に、デバッガのプロンプトからほかの一般UNIXコマンドを実行することができます。以下に指定方法を示します。なお、スクリーンモードのラインコマンド入力ウィンドウからは指定できません。

指定方法

```
! [UNIXコマンド]
```

UNIXコマンドを省略すると、シェルが起動され、UNIXコマンドの入力待ち状態となります。

例

[例1]

```
$ svd -r prog01
 + ! ls
  prog01     prog01.cob      prog01.svd     prog02.cob
 +
```

[例2]

```
$ svd -r prog01
 + !
 $ ls
   prog01    prog01.cob      prog01.svd     prog02.cob
 $ exit
 +
```

注意

操作履歴機能を使用している場合、当コマンドは操作履歴ファイルに反映されません。
また自動デバッグ時には、当コマンドは不当文字として扱われます。

23.10 注意事項

デバッガを使う場合の注意事項を以下に示します。

COPY文プログラムの制限

- COPY文を以下のように記述すると、原文内の現在表示位置や、中断点を示す網かけ位置がずれたり、オブジェクト選択による各種コマンド操作で誤動作したりすることがあります。
  - 複数行に渡り記述した場合
  - 同一行内にCOPY文の終了に続けて有効な動詞などを記述した場合
  - 手続き部の文の開始から文の完結までの間にCOPY文を記述した場合
  - REPLACING指定、JOINING指定またはDISJOINING指定を行い、置き換える文字が1行に収められず、新しい行が追加された場合
- 以下の場合、プログラムを正しくデバッグできません。それぞれについて、回避方法を示します。
  - 以下を含む登録集原文をCOPY文で取り込んでいる場合
    - プログラム定義全体(IDENTIFICATION DIVISIONからEND PROGRAMまで)
    - ファクトリ定義全体(IDENTIFICATION DIVISIONからEND FACTORYまで)
    - オブジェクト定義全体(IDENTIFICATION DIVISIONからEND OBJECTまで)
    - メソッド定義全体(IDENTIFICATION DIVISIONからEND METHODまで)

    [回避方法]

    上記の登録集原文に記述された文を、登録集原文を取り込んでいるプログラムに直接記述します。
  - 手続き部にCOPY文を記述し、かつ、取り込んでいる登録集原文中の最後の行がそとPERFORM文である場合

    [回避方法]

    登録集原文の最後にCONTINUE文などを記述します。
  - 手続き部にCOPY文を記述し、かつ、手続き部の最後の行がCOPY文であり、かつ、END PROGARM文の記述がない場合

    [回避方法]

    COPY文を記述しているプログラムにEND PROGARM文を記述します。
  - 手続き部にCOPY文を記述し、かつ、取り込んでいる登録集原文中の最後の行がそとPERFORM文であり、かつ、COPY文の直後がそとPERFORM文である場合

    [回避方法]

    登録集原文の最後とこの登録集原文を指定しているCOPY文の直後にCONTINUE文などを記述します。
  - 手続き部にCOPY文を記述し、かつ、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]

    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。

REPLACE文プログラムの制限

REPLACE文があると置換結果が表示されなかったり、置換後の網かけ位置がずれたりする場合があります。

シグナル発生時の制限

デバッグ処理中にゼロ除算などのシグナル(割込みを除く)が発生した場合、それ以降のデバッグ作業(デバッグプログラムの実行)を行うことができません。

ACCEPT文実行中のデバッガの終了

ACCEPT文実行中の入力待ち状態で、デバッガを直ちに終了(再デバッグ)させることはできません。

[回避方法]実行中のデバッグプログラムを強制中断(CTRL+C)させてからQUITまたはRERUNサブコマンドを入力します。その後、利用者プログラム標準入出力ウィンドウ("-w"オプション指定の場合には、デバッガ起動ウィンドウ)でReturnキーを押してください。

同一のエディタ行番号が複数ある場合の中断点の設定方法

翻訳オプション(NUMBER)で翻訳したソースプログラム中に同一のエディタ行番号が複数ある場合、同一行番号の文に対する中断点の設定は、最初に出現する同一行番号の文に対して行われます。

KANA(JIS8)で翻訳したプログラムの制限

翻訳オプションKANA(JIS8)で翻訳されたプログラムで、半角カナを含むデータ名、手続き名を識別名として認識することはできません。

また、プログラム名に半角カナを含む場合、デバッガが出力するコマンド結果および操作履歴ファイルのプログラム名は正しく出力されません。

TAB文字の設定方法

コマンドファイルおよびラインモードでTAB文字が指定された場合、デバッガ内部では空白文字(ASCII X"20")として扱われます。そのため、TAB文字の設定を行う場合には、16進(ASCII X"09")で設定してください。

日本語空白の扱い

デバッガのコマンドで指定する条件文では、翻訳オプションNSPCOMP(ASP)は無効となります。

デバッグプログラムの標準入出力処理の制限

デバッグプログラムの標準入出力を、リダイレクション機能により別のファイルに切り換えることはできません。

プログラムのソースがない場合について

プログラムが、指定された格納ディレクトリに存在しない場合、プログラムの代わりに"No Source"が表示されます。なお、追尾、逆トレース、実行中断により"No Source"表示後、環境変更の"登録集の展開表示"は無条件に"行わない"に変更されているので注意してください。

サブコマンドについて

1つのサブコマンドは英数字で最大2048文字まで有効になります。2048文字を超えると、超えた文字は無視されます。

プログラム名長について

プログラム名は英数字で最大1024文字まで有効になります。

逆トレース中のコマンド操作について

逆トレース中にほかのデバッガコマンドを実行すると、トレース位置はトレース開始位置に再設定されます。

英小文字を記述したプログラムをNOALPHALで翻訳した場合

ENVのデフォルトは"SAME"または"SAME(ALL)"です。したがって、サブコマンドのオペランドとしてプログラム名およびデータ名に英小文字を指定する場合、プログラム起動後に環境変更(ENV)で"DIFF"に変更する必要があります。なお、起動時に"DIFF"を指定したい場合には、あらかじめ動作環境ファイル(cobdebug.ini)に"ENV DIFF"を保存します。そして"-j"オプション指定により動作環境ファイルから読み込むようにしてください。

サブコマンドのオペランドに英小文字が記述されたコマンドファイルに対し、“-b”オプションを指定してバッチデバッグをする場合には注意が必要です。またプログラム名に英小文字を使用し、以下の起動パラメタでデバッガを起動する場合には、必ず上記の方法で動作環境ファイルを読み込むようにしてください。

- “-v”オプションを指定してカバレージ機能を使用する。
- “-p”オプションを指定してデバッグ開始プログラムを指定する。

“-p”オプション指定時(svd)のカバレージ情報採取について

カバレージ情報を採取する際に“-p”オプションで開始プログラムを指定すると、開始プログラムが初めて呼ばれた時点からカバレージ情報が出力されます。開始プログラムが呼ばれる前に実行されたプログラムのカバレージ情報は出力されません。

環境変数DISPLAYの設定

Webサーバなどのサーバアプリケーションでは、デバッガのウィンドウの表示先(ディスプレイ名)が正しく設定されていないと、ウィンドウが表示されないことがあります。この場合は、環境変数DISPLAYにディスプレイ名を設定してからサーバを起動してください。

「開始プログラムに到達する前に実行が終了しました。」が表示された場合の対処方法

「開始プログラムに到達する前に実行が終了しました。」が表示された場合の対処方法を以下に示します。

- 開始プログラムに指定したプログラムが実行されずにプログラムが終了した。または、指定したプログラム名が間違っている。

  開始プログラム名に正しいプログラム名を指定してください。

- プログラムがデバッグできるプログラムになっていない。

  “23.2 デバッグの手順”を参照してプログラムを作成しなおしてください。

- デバッグ情報格納ディレクトリで指定したディレクトリにデバッグ情報ファイルが存在しない。

  デバッグ情報ファイルは、カレントディレクトリを検索しますが、それ以外の場所にデバッグ情報ファイルが存在する場合は、デバッグ情報格納ディレクトリを指定してください。

開始プログラムに到達しなかった場合の動作

開始プログラムに到達して初めてデバッグ操作が可能になります。スクリーンモードの場合、開始プログラムを検出するまでは、メインウィンドウには何も表示されません。また、デバッグプログラムが無限ループしている場合には、プログラムが終了しないため、killコマンドを使用してデバッガのプロセスを強制終了させてください。

デバッグ情報ファイルがない場合の動作

デバッグ開始プログラムに到達して初めてデバッグ操作が可能になります。デバッグ情報ファイルがない場合は、開始プログラムに到達しないまま、デバッグプログラムが終了し、メッセージ「開始プログラムに到達する前に実行が終了しました」が表示されます。また、デバッグプログラムが無限ループしている場合には、プログラムが終了しないため、上記のメッセージは表示されません。この場合は、killコマンドを使用してデバッガのプロセスを強制終了させてください。

埋込みSQL文のデバッグ

PROCEDURE DIVISIONに記述された“EXEC SQL”に中断点や通過カウント点を設定できます。ただし、非実行文(WHENEVER文など)を含む“EXEC SQL”には中断点や通過カウント点を設定できません。

また、SQL文に中断点や通過カウント点を直接設定することはできません。

第24章　SCREEN-DESIGNERの使い方

本章では、SCREEN-DESIGNER(略称S-DESIGN)の使い方について説明します。

SCREEN-DESIGNERは、将来のNetCOBOLリリースで提供を停止する予定です。

24.1 概要

SCREEN-DESIGNER(略称S-DESIGN)は、スクリーン操作機能で使用する画面の設計を行うためのユーティリティです。S-DESIGNでは、画面設計エディタを使って会話的に、画面の設計を行うことができます。また、設計した画面から、それに対応するCOBOLの画面節用の登録集の自動生成を行うことができ、画面入出力を行うCOBOLプログラムを簡単に作成することができます。

24.2 S-DESIGNの機能

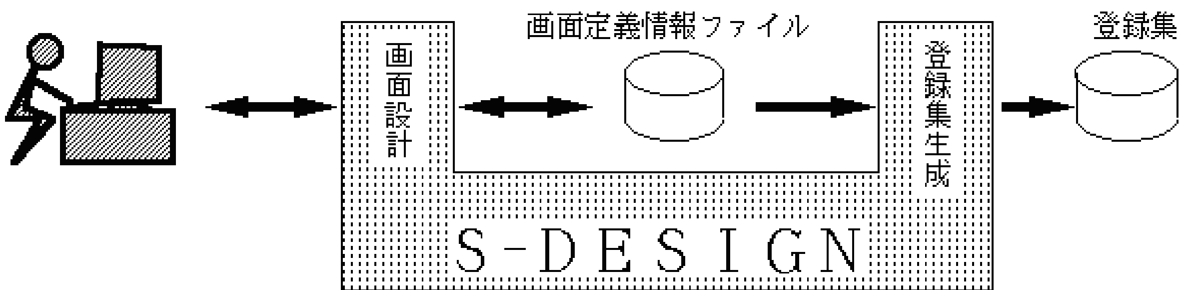
S-DESIGNは、次の2つの機能があります。

画面設計

画面設計エディタを使って、画面のスクリーン表示イメージで画面設計を行い、定義情報を画面定義情報ファイルに出力します。

登録集作成

画面設計で作成した画面定義情報ファイルから画面節用の登録集をファイルに出力します。



24.3 S-DESIGNの操作概要

以下にS-DESIGNの操作概要について説明します。

詳細については、manマニュアルおよびHELPを参照してください。

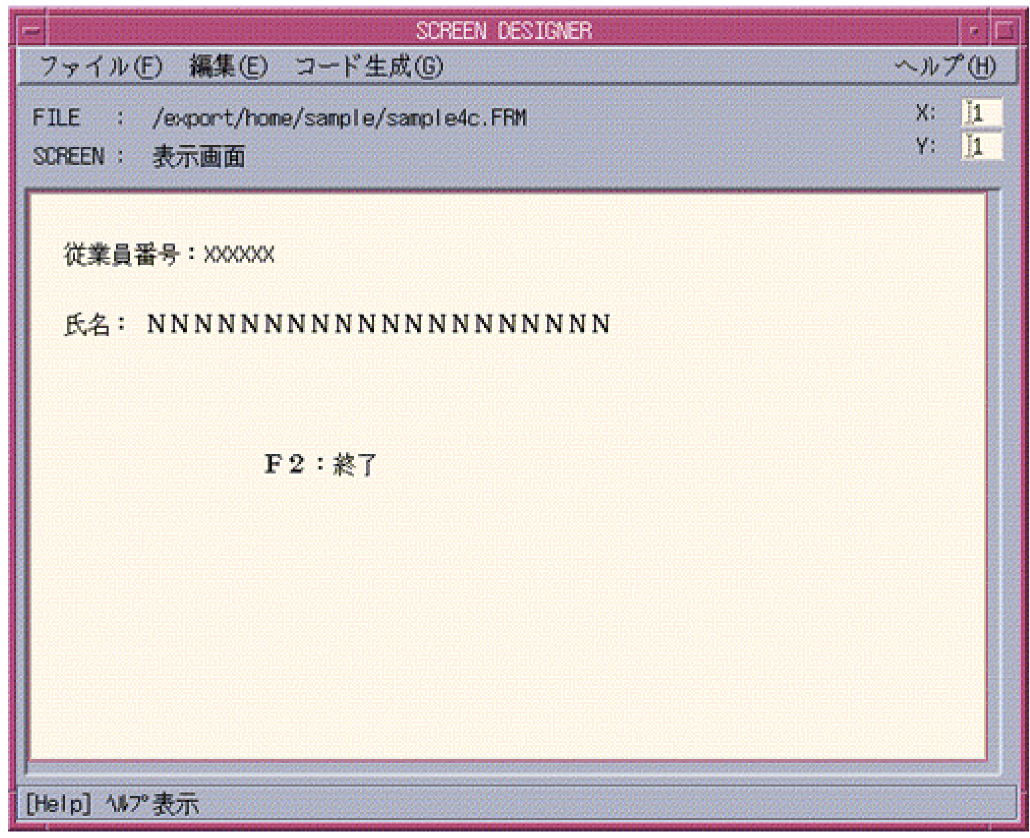
24.3.1 起動

S-DESIGNを起動するには、以下のコマンドを入力します。

```
$ sdesign ［ファイル名］
```

S-DESIGNを起動すると、"図24.1 メインウィンドウ"に示すウィンドウが表示されます。ファイル名を指定した場合は、画面設計域に指定されたファイルの画面イメージが表示されます。

図24.1 メインウィンドウ



24.3.2 画面設計

メインウィンドウの画面設計域では、実際に利用するときの画面イメージを確かめながら、画面を設計することができます。

24.3.3 画面設計域の操作方法

編集の規則

1. 編集画面のサイズは縦24×横80です。
2. 入出力操作を行うための項目の定義は項目情報設定ダイアログで行います。
3. 表示だけの項目は、半角文字で色指定や反転などの指定を行わないときは、編集画面上に直接表示文字を入力するだけで定義することができます。色指定や反転指定などを行う場合や、表示文字が日本語文字または全角文字の場合には、項目定義ダイアログで定義します。
4. 表示画面上に表示されている文字列を変更するときは、以下のように操作します。
   - 編集画面に直接入力した文字は、編集画面上で直接変更します。
   - 項目情報設定ダイアログで定義されている文字列は、項目情報設定ダイアログを開いて変更します。項目情報設定ダイアログで定義した項目は編集画面上では直接変更できません。
5. 定義または移動する項目は、ほかの定義項目と重ならないようにします。
6. 行挿入は、最終行が空白行のときにだけ行うことができます。
7. 一文字挿入は、カーソル位置より右側に定義項目がある場合には、その定義項目の直前のカラム位置の文字が一文字削除されます。カーソル位置より右側に定義項目がない場合には、行の最右端の文字が一文字削除されます。
8. 選択は、以下のように操作します。
   - 項目情報設定ダイアログで定義した項目にカーソルが移動すると、自動的に項目が選択されます。

－ 行をまたがる選択や項目が定義さている範囲の選択はできません。直接入力した文字の範囲選択または項目ごとの選択を利用してください。

画面設計域では、以下に示す操作により画面設計を行います。

- 画面項目の定義
- カット
- コピー
- ペースト
- 削除
- 行カット
- 行コピー
- 行ペースト
- 行の挿入
- 行の削除

画面項目の定義

1. 新規に項目を定義するときには、画面設計域で、項目を定義したい位置にカーソルを移動します。すでに定義してある項目を更新するときには、画面設計域で、更新する項目の位置にカーソルを移動します。
2. 編集メニューから"項目情報設定"を選択すると、項目情報設定ダイアログが表示されるので、画面項目に対する属性を設定します。"表24.1 項目情報設定ダイアログで設定する情報"に、項目情報設定ダイアログで設定する情報を示します。

表24.1 項目情報設定ダイアログで設定する情報

情報名	指定内容	選択肢
項目種別	設定したい項目の種別と項類	(注1)
項目名	項目名	－
位置	項目の開始位置	－
VALUE	表示する定数値	－
PICTURE 文字列	項目の型や大きさを示すPICTURE 文字列	－
SIZE	SIZE句とSIZEの大きさ	指定なし/指定あり
FROM	出力項目に対応付けるデータ名、定数	－
TO	入力項目に対応付けるデータ名	－
USING	更新項目に対応付けるデータ名	－
前景色	表示する文字の色	(注2)
背景色	表示する文字の背景の色	(注2)
輝度	項目の輝度	通常/高輝度/低輝度
BLINK	BLINK 句	指定なし/指定あり
REVERSE VIDEO	REVERSE-VIDEO 句	指定なし/指定あり
UNDERLINE	UNDERLINE 句	指定なし/指定あり
OVERLINE	OVERLINE句	指定なし/指定あり
GRID	GRID句	指定なし/指定あり
LEFTLINE	LEFTLINE句	指定なし/指定あり
BELL	BELL句	指定なし/指定あり
BLANK	BLANK 句	指定なし/LINE/SCREEN

情報名	指定内容	選択肢
ERASE	ERASE 句	指定なし/EOL /EOS
AUTO	AUTO句	指定なし/指定あり
REQUIRED	REQUIRED句	指定なし/指定あり
SECURE	SECURE句	指定なし/指定あり
FULL	FULL句	指定なし/指定あり
JUSTIFIED	JUSTIFIED 句	指定なし/RIGHT
ZERO-FILL	ZERO-FILL 句	指定なし/指定あり
PROMPT	PROMPT句と定数値	指定なし/指定あり
SIGN	SIGN句	指定なし/LEADING /TRAILING
SEPARATE	SIGN句のSEPARATE指定	指定なし/指定あり
BLANK WHEN ZERO	BLANK WHEN ZERO 句	指定なし/指定あり

注1: 種別は定数/入力/出力/入出力/更新、項類は英字/数字/英数字/英数字編集/数字/編集/日本語/日本語編集からそれぞれ選択します。

注2: 指定なし/黒/青/緑/シアン/赤/マゼンダ/黄/白から選択します。ただし、このシステムでは指定が有効になりません。

参考

指定した項目種別により、ダイアログに表示されない情報名があります。

24.3.4 スクリーン名の指定

スクリーン名を指定または変更したいときは、ファイルメニューで“スクリーン名の指定”を選択してください。

24.3.5 登録集生成

COBOLの登録集を生成したいときは、コード生成メニューで“画面節ソース”を選択してください。画面節ソース表示ダイアログが表示され、画面節ソース表示域に生成された登録集が表示されます。これを出力したファイルがCOBOL応用プログラムの画面節の登録集になります。

画面節ソースの修正

画面節ソース表示域では、登録集を修正することができます。画面節ソース表示域では、ボタンを操作することで選択範囲のカット、コピー、ペーストおよび削除を行うことができます。

注意

画面節ソース表示域で登録集を修正した場合、画面設計域に修正した結果は反映されません。なお、修正した登録集をファイルに出力できます。

登録集のファイル出力

登録集をファイルに出力するときは、“ファイル出力”ボタンを選択することにより、ファイル出力ダイアログが表示されます。ファイルを出力する場合は、ファイル名を指定して、“了解”ボタンを選択してください。ファイルを出力しない場合は“取消”ボタンを選択してください。

参考

ファイル名には、拡張子として“cbl”を付けることをおすすめします。

24.4 関連コマンド

S-DESIGNには以下の関連コマンドが用意されています。

コマンド詳細については、manマニュアルを参照してください。

ソース生成

S-DESIGNで作成した画面定義情報ファイルから画面節の登録集を作成します。

```
$ sdegenerate 画面定義情報ファイル名 画面節の登録集ファイル名
```

Makeルールとして、このコマンドを指定しておくことで最新の登録集を利用するように設定できます。

例

```
.FRM.cbl:        sdegenerate $*.FRM $*.cbl
```

画面定義情報ファイルのコード変換

S-DESIGNで作成した画面定義情報ファイルをシフトJISとEUCの間でコード変換することが可能です。

シフトJISからEUCに変換

```
$ sdestou 変換元画面定義情報ファイル名 変換先画面定義情報ファイル名
```

EUCからシフトJISに変換

```
$ sdeutos 変換元画面定義情報ファイル名 変換先画面定義情報ファイル名
```

第25章　ファイルユーティリティ

本章では、ファイルユーティリティの使い方について説明します。

25.1 概要

ファイルユーティリティは、COBOLファイルシステムが扱うファイル(以下、COBOLファイルと表します)を、COBOLプログラムを介することなく操作するためのユーティリティです。

- 各種テキストエディタを使って作成したテキストデータからCOBOLファイルを作成する
- COBOLファイルに対する操作(ファイル構造の変更、索引ファイルの再編成/復旧/属性の表示など)
- COBOLファイルのレコードの操作(表示/編集/整列など)を行う

ファイルユーティリティにはスクリーンモードとコマンドモードがあります。スクリーンモードでは、Xウィンドウを使用した対話形式によってCOBOLファイルの操作を行います。コマンドモードでは、機能ごとのコマンドでCOBOLファイルの操作を行います。

- 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6.2 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。

25.2 ファイルユーティリティの機能

ここでは、ファイルユーティリティの機能について説明します。

25.2.1 機能概要

ファイルユーティリティには、以下の機能があります。

機能名	機能概要	スクリーンモード	コマンドモード
変換	テキストファイルから可変長の順ファイルへの変換、および、可変長の順ファイルからテキストファイルの変換を行う。また、印刷ファイルからテキストファイルへの変換を行う。	○	○
ロード	可変長の順ファイルから、固定長または可変長の順/相対/索引ファイルの創成または拡張を行う。	○	○
アンロード	固定長または可変長の順/相対/索引ファイルから可変長の順ファイルの創成を行う。	○	○
表示	レコードの内容を表示する。	○	○
印刷	レコードの内容を印刷する。	○	－
編集	レコードを編集する。	○	－
拡張	ファイルの拡張を行う。	○	－
整列	任意のキーでレコードを整列し、可変長の順ファイルに出力する。	○	○

機能名	機能概要	スクリーンモード	コマンドモード
属性	索引ファイルの属性を表示する。	○	○
復旧	索引ファイルを復旧する。	○	○
再編成	索引ファイルの未使用域を削除する。	○	○

○ : サポートあり
－ : サポートなし

25.2.2 ファイルの操作方法

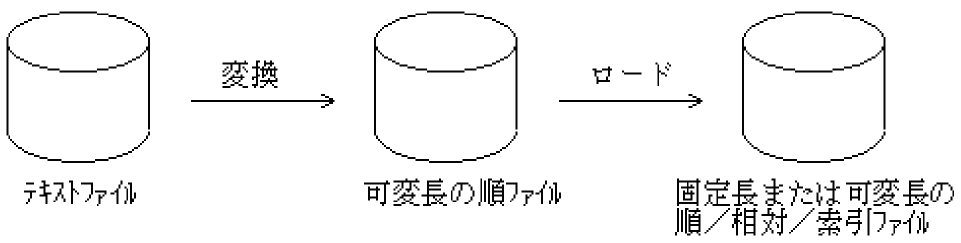
ここでは、ファイルユーティリティの機能を利用して、基本的なファイル操作を行う方法について説明します。

ファイルの創成

ファイルの創成を行うには、以下の方法があります。

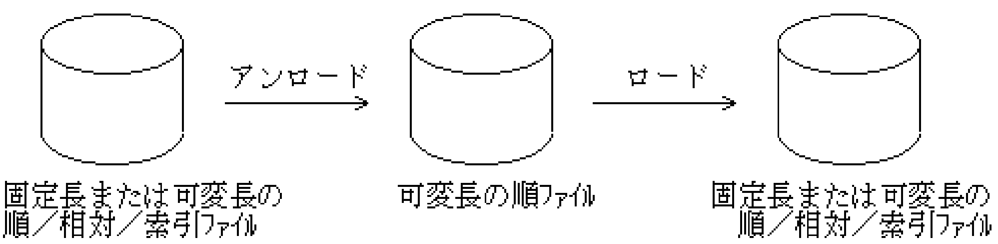
テキストファイルを使用したファイルの創成

変換機能とロード機能を組み合わせることにより、各種テキストエディタで作成したテキストファイルから、固定長または可変長の順/相対/索引ファイルを創成することができます。



COBOLファイルを使用したファイルの創成

アンロード機能とロード機能を組み合わせることにより、既存の固定長または可変長の順/相対/索引ファイルから、固定長または可変長の順/相対/索引ファイルを創成することができます。

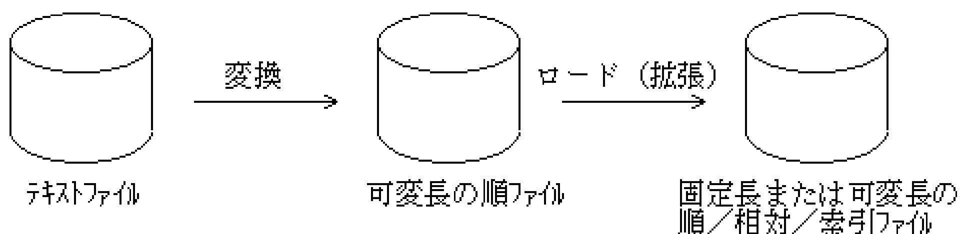


ファイルの拡張

ファイルの拡張（追加）を行うには、以下の方法があります。

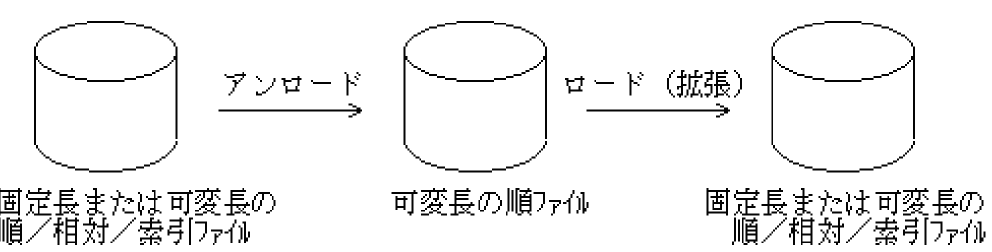
テキストファイルを使用したファイルの拡張

変換機能とロード機能を組み合わせることにより、各種テキストエディタで作成したテキストファイルの内容で、既存の固定長または可変長の順/相対/索引ファイルを拡張することができます。



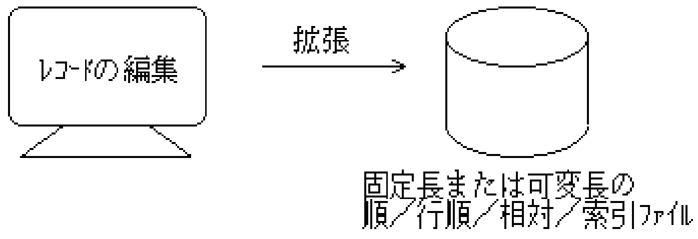
COBOLファイルを使用したファイルの拡張

アンロード機能とロード機能を組み合わせることにより、既存の固定長または可変長の順/相対/索引ファイルの内容で、固定長または可変長の順/相対/索引ファイルを拡張することができます。



対話形式でのファイルの拡張

スクリーンモードのファイルユーティリティでは、拡張機能を使用することにより、出力すべきレコードを対話形式で編集しながら、ファイルを拡張することができます。



注意

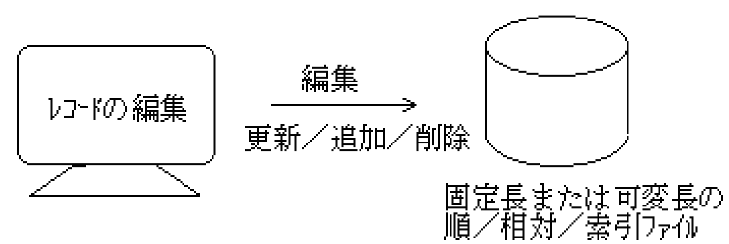
コマンドモードでは、対話形式でのファイルの拡張は行えません。

レコードの更新/追加/削除

スクリーンモードのファイルユーティリティでは、編集機能を使用することにより、以下の操作を行うことができます。

- 固定長または可変長の順ファイルに対してのレコードの更新

- 相対/索引ファイルに対してのレコードの更新/追加/削除



注意

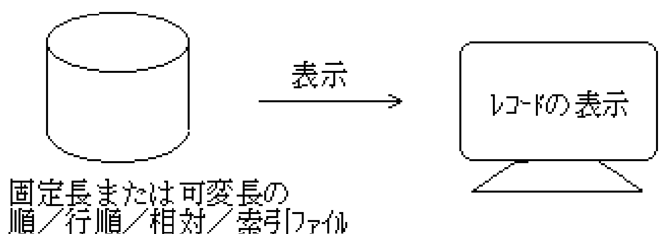
- 順ファイルおよび索引ファイルでは、連続して同じレコードを更新することはできません。同じレコードを再度更新したい場合は、更新するレコードを再度表示し直してください。
- コマンドモードでは、レコードの更新/追加/削除は行えません。

レコードの表示

表示機能を使用することにより、レコードの表示を行うことができます。

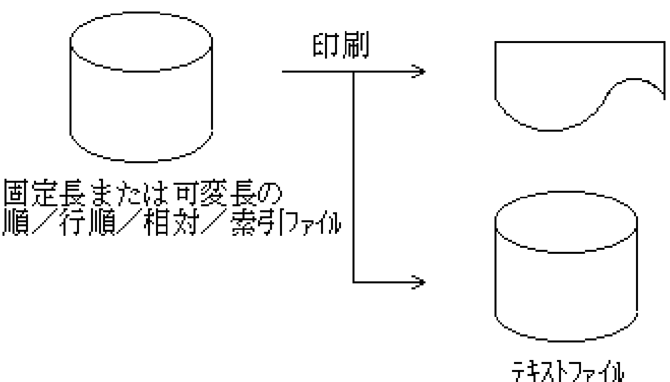
スクリーンモードのファイルユーティリティでは、対話形式でレコードを選択しながら表示を行うことができます。

コマンドモードのファイルユーティリティでは、表示開始位置、表示終了位置または表示レコード件数を指定することにより、任意の範囲のレコードを表示することができます。



レコードの印刷

スクリーンモードのファイルユーティリティでは、印刷機能を使用することにより、レコードの印刷を行うことができます。この際、印刷範囲を指定することができます。また、印刷結果の出力先として、lpと印刷ファイルのどちらかを指定することができます。lpを指定した場合、印刷結果は直接lpコマンドに渡され、印刷が行われます。印刷ファイルを指定した場合、印刷結果は指定した印刷ファイルにテキスト形式で格納されます。

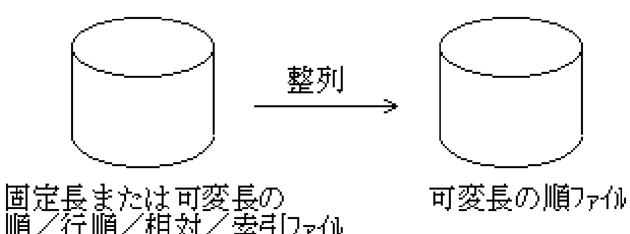


注意

- 出力先にlpを指定した場合、lpコマンドへのパラメタを指定することができます。しかし、パラメタに誤りがあってもファイルユーティリティとしてのエラーメッセージは表示されません。
- コマンドモードでは、レコードの印刷は行えません。

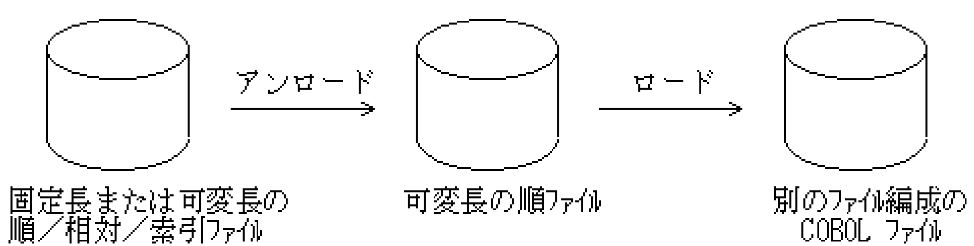
レコードの整列

整列機能を使用することにより、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整列し、その結果を可変長の順ファイルに出力することができます。



ファイル編成の変更

ロード機能とアンロード機能を組み合わせることにより、固定長または可変長の順/相対/索引ファイルを、別のファイル編成に変更することができます。



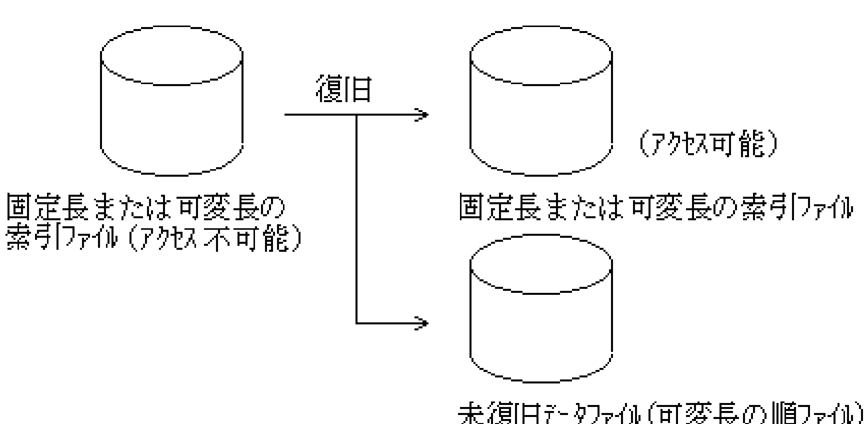
UVPIデータのテキスト変換

変換機能を使用することにより、UVPIデータが付加された印刷ファイルをテキストファイルに変換することができます。ただし、この変換は次の印刷ファイルに対しては、動作が保証できません。

- FORMAT句付き印刷ファイルとして出力した印刷ファイル
- COBOLランタイムシステム以外が作成した印刷ファイル

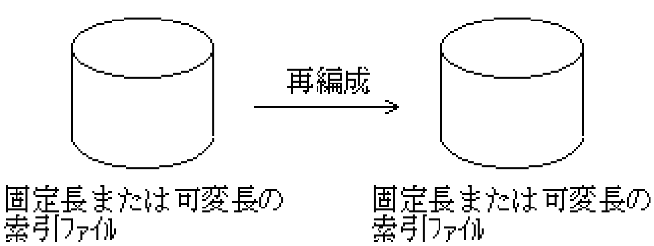
索引ファイルの復旧

索引ファイルをオープン中にCOBOLプログラムが異常終了し、索引ファイルのクローズ処理が正常に行われないことがあります。このとき索引ファイル中のレコードとキーの対応関係が壊れてしまう場合があります。このような場合、その索引ファイルは再度アクセスすることができなくなります。しかし、復旧機能を使用することにより、レコードとキーの対応関係を復旧することができます。ただし、データに異常があり、復旧できないレコードがあった場合には、それらのレコードは可変長の順ファイル形式で未復旧データファイルとして出力されます。



索引ファイルの再編成

再編成機能を使用することにより、索引ファイル中の空きブロックを可能なかぎり削除し、ファイルサイズを縮小することができます。

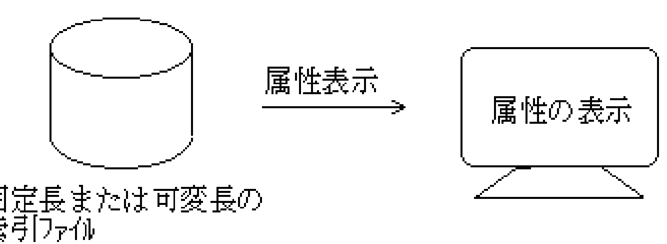


注意

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。

索引ファイルの属性表示

属性表示機能を使用することにより、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)を表示することができます。



注意

索引ファイル以外は、属性情報を表示できません。

ファイル操作上の注意事項

- 編集および拡張では、バックアップファイルを作成することができます。バックアップファイルを作成することにより、ファイルユーティリティの実行中にエラーが発生し、処理対象ファイルが破壊された場合でも、もとのファイルの内容はバックアップファイルとして保存されます。したがって、ファイルの編集および拡張を行う場合には、バックアップファイルを作成するようにしてください。バックアップファイルは、処理対象のファイルと同じディレクトリに拡張子bakを付加したファイル名で生成されます。
- 編集および表示では、COBOLの文法規則に従ったファイルアクセスを行います。このため、順ファイルに対するレコードの追加/削除はできません。また、相対/索引ファイルでは順または乱にレコードをアクセスできます。ただし、順ファイルでは順アクセスしかできません。
- 順ファイルと索引ファイルでは、連続して同じレコードを更新することはできません。同じレコードを再度更新したい場合は、更新するレコードを再度表示し直してください。
- 大容量ファイルをサポートしているのは、スクリーン機能の復旧機能、およびコマンドモードのみです。復旧機能以外のスクリーン機能では大容量ファイルを使用できません。

25.3 スクリーンモードの使い方

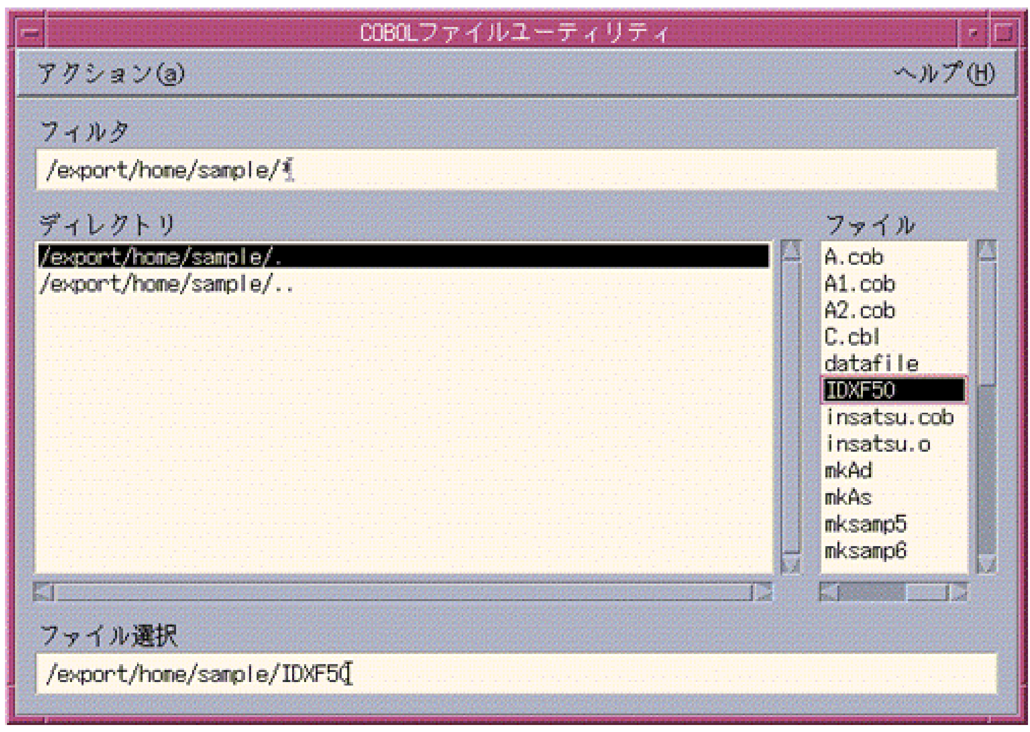
スクリーンモードでファイルユーティリティを使用する方法について説明します。

25.3.1 起動と機能選択

以下のコマンドにより、スクリーンモードのファイルユーティリティが起動されます。

```
$ cobfuty
```

起動後、ファイルユーティリティのメインウィンドウが表示されます。処理対象とするファイルを選択し、アクションメニューから実行したい機能を選択してください。

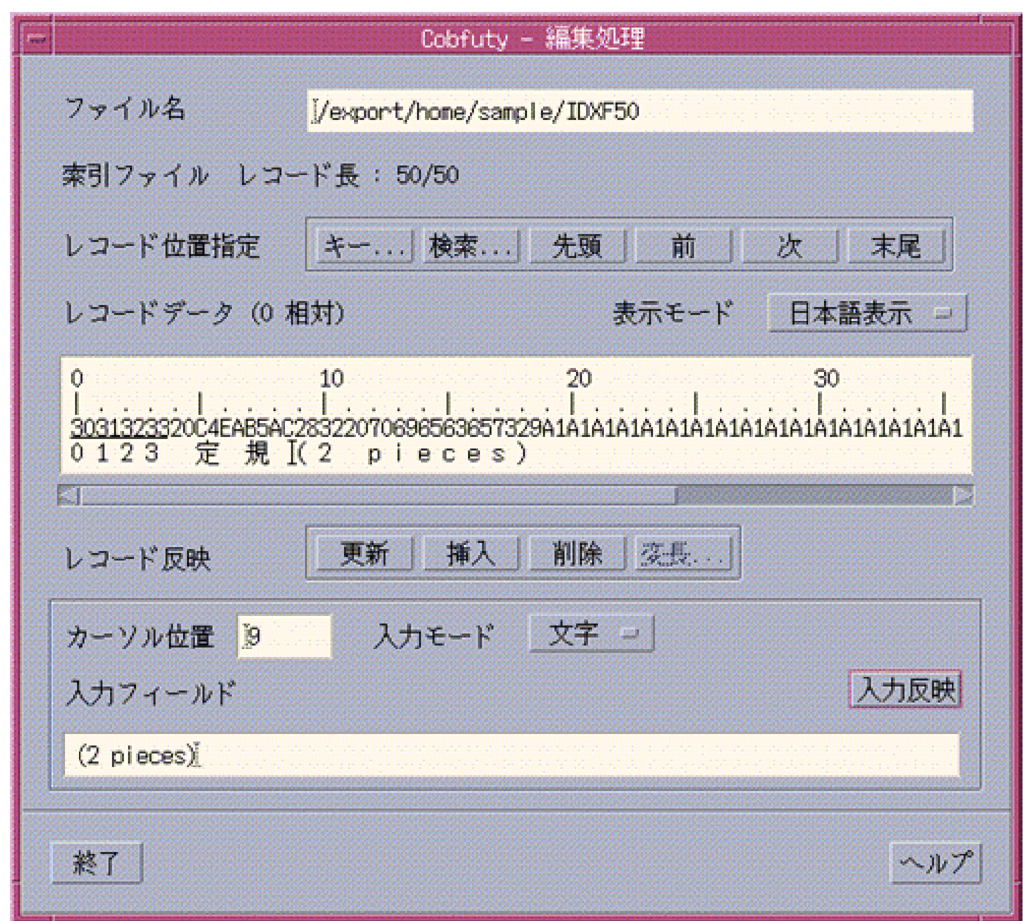


アクションメニューで機能を選択すると、それぞれの機能ごとに必要なパラメタを指定するウィンドウが表示されます。それぞれのウィンドウの指示に従い、パラメタを指定してください。

メインウィンドウと機能ごとのパラメタを指定する各ウィンドウにはヘルプが用意されています。ヘルプには、各機能とパラメタの指定方法についての詳細な説明が記述されているので、ヘルプを参照しながら使用してください。ここでは、編集を例にした操作方法を簡単に説明します。

アクションメニューで編集を選択すると、ファイル属性を指定するウィンドウが表示されます。編集するファイルのファイル属性を指定してください。なお、ファイル編成として索引ファイルを選択した場合は、ファイル中に設定されているファイル属性が使用されるため、レコード形式および最大レコード長の指定は不要です。

ファイル属性を指定すると編集ウィンドウが表示されます。編集ウィンドウ上の各種ボタン、“入力フィールド”、“レコードデータ”などでレコードの編集を行ってください。



レコードの編集

レコードの更新または追加を行うためには、次の操作が必要です。

1. “入力フィールド”上へのデータ設定

   “入力フィールド”上に直接編集データを入力することにより行います。

2. “入力フィールド”上のデータを“レコードデータ”上に反映

   “入力フィールド”上へのデータの入力後、リターンキーまたは“入力反映”ボタンを押すことにより行います。データの反映位置は、“レコードデータ”上でカーソルがある位置です。このとき、“入力モード”ボタンの選択により、入力フィールドのデータを文字形式とみなすか、16進形式とみなすかを選択できます。

3. “レコードデータ”上に編集したレコードを実際のファイルに出力

   編集レコードを表示フィールド上に生成した後、“更新”ボタンまたは“挿入”ボタンを押すことにより行います。

レコード内相対位置

表示フィールドでは、レコードの先頭を相対位置0として表示します。したがって、レコード長が30バイトの場合、レコードの表示は、相対位置が29の位置までです。また、カーソル位置の指定によって、任意のレコード位置を表示したい場合もレコード先頭を0とした位置で指定してください。

25.4 コマンドモードの使い方

ここでは、コマンドモードでファイルユーティリティを使用する方法について説明します。

25.4.1 変換

機能

テキストファイルから可変長の順ファイルを創成、または可変長の順ファイルからテキストファイルを創成します。

また、COBOLプログラムの実行により作成されたUVPIデータを含む印刷ファイルからテキストファイルを創成します。

参考

- “テキストファイル”とは、COBOLの行順ファイルであり、文字表現のデータと16進表現のデータで表されたファイルです。改行文字で区切られた1行のデータを1レコードとします。
- “文字表現のデータ”とは、文字そのものを表現するデータ(文字データ)です。
- 3バイトの文字列“abc”は、“abc”と表記します。
- “16進表現のデータ”とは、バイナリデータを16進数字で表現したデータです。
- 2バイトのバイナリ値“5”は、“0005”と表記します。
- 4バイトのバイナリ値“3456”は、“00000d80”と表記します。
- 3バイトの文字“abc”、2バイトのバイナリ値“5”、3バイトの文字“999”の連続したデータは、“abc0005999”と表記します。
- 環境変数LANGのコード系がUnicodeの場合、テキストファイルの文字コードはUTF-8として扱います。

コマンド形式

```
cobfconv -o出力ファイル名 -c方法,形式  入力ファイル名[,LFS]
```

出力ファイル名

変換結果を出力するファイルのパス名を指定します。

方法,形式

変換方法および変換時のデータ記述形式を、以下の形式で指定します。

－ 環境変数LANGのコード系がEUCまたはシフトJISの場合

```
        { b }     { allchar                          }
-c      { t }  ,  { alltxtbin                        }
        { p }     { " { c 長さ }  [; …]"             }
                  {   { t 長さ }                     }
```

－ 環境変数LANGのコード系がUnicodeの場合

```
        { b }     { allucs2                          }
-c      {   }  ,  { allutf8                          }
        { t }     { " { u 長さ }  [; …]"             }
                  {   { f 長さ }                     }
                  {   { t 長さ }                     }
```

変換方法を以下の文字で指定します。

b

テキストファイルを順ファイルに変換します。レコード中のデータ記述形式を指定します。

t

順ファイルをテキストファイルに変換します。レコード中のデータ記述形式を指定します。

p

UVPIデータをテキストファイルに変換します。データ記述形式を指定できません。

allchar

レコード中の全データを文字形式とみなして変換を行います。

alltxtbin

レコード中の全データを16進形式とみなして変換を行います。

"{c長さ|t長さ}[;…]"

レコード中に文字形式と16進形式が混在する場合に、混在形態を指定します。データ形式をキーワード文字"c"(文字形式)または"t"(16進形式)で指定し、キーワード文字に続けてそれぞれのデータ形式での長さを指定します。たとえば、c8;t4は、文字形式の8バイトと16進形式の4バイトです。

allucs2

レコード中の全データをUCS-2形式とみなして変換を行います。

allutf8

レコード中の全データをUTF-8形式とみなして変換を行います。

"{u長さ|f長さ|t長さ}[;…]"

レコード中にデータ形式が混在する場合に、混在形態を指定します。データ形式をキーワード文字"u"(UCS-2形式)、"f"(UTF-8形式)または"t"(16進形式)で指定し、キーワード文字に続けてそれぞれのデータ形式での長さ(UCS-2形式の場合は文字数)を指定します。

混在形式での長さの指定には、以下の注意が必要です。

- 順ファイルからテキストファイルへの変換では、16進形式に変換する前の長さを指定します。たとえば、t4の場合、変換後の16進形式では、8バイトのデータです。
- テキストファイルから順ファイルへの変換では、文字形式に変換した後の長さを指定します。たとえば、t4の場合、変換前の16進形式では、8バイトのデータです。
- レコード中に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

変換対象となるファイルのパス名を指定します。COBOLで作成した大容量ファイルを扱う場合、入力ファイル名に"ファイル名,LFS"を指定します。

テキストファイルを順ファイルに変換(文字、16進混在の場合)

```
$ cobfconv -ooutfile -cb,"c3;t2;c3" infile
```

順ファイルをテキストファイルに変換(コード系がUnicodeで、すべてUTF-8形式の場合)

```
$ cobfconv -ooutfile -ct,allutf8 infile
```

25.4.2 ロード

機能

可変長の順ファイルから可変長または固定長の順/相対/索引ファイルを創成します。また、可変長の順ファイルのレコードを、すでに存在する順/相対/索引ファイルに拡張することもできます。ファイルの拡張を行う場合、バックアップファイルが作成され、エラーが発生した場合には、出力ファイルはコマンド実行前の状態に戻されます。

```
cobfload -o出力ファイル名 [-e] -dファイル属性  入力ファイル名[,LFS]
```

出力ファイル名

創成または拡張するファイルのパス名を指定します。

-e

ファイルの拡張を行う場合、"-e"を指定します。なお、索引ファイルの拡張を行う場合には、"-d"パラメタのレコード形式、レコード長およびキー情報を指定することはできません。

ファイル属性

創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。

```
      { S }     { f }
-d    { R }  ,  {   }  , レコード長, "(オフセット, 長さ[/オフセット, 長さ]  …)[,D][;  …]"
      { I }     { v }
```

ファイル編成

以下の文字で指定します。

S： 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

以下の文字で指定します。

f： 固定長
v： 可変長

レコード長

可変長の場合のレコード長は、最大レコード長で指定します。

レコードキー情報

ファイル編成に索引ファイルを指定した場合、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。

/

1つのレコードキーとして、非連続な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

創成または拡張するレコードが格納されているファイルのパス名を指定します。指定するファイルは、可変長の順ファイルである必要があります。
COBOLで作成した大容量ファイルを扱う場合、入力ファイル名に“ファイル名,LFS”を指定します。

例

索引ファイルの創成

```
$ cobfload -oixdfile -dI,v,80,"(0,5/10,5),D;(5,5)" infile
```

相対ファイルの拡張

```
$ cobfload -orelfile -e -dR,f,80 infile
```

注意

- 索引ファイルの拡張指定では、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。
- 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。
- ファイルの拡張指定でエラーが発生した場合、出力ファイルはファイル拡張を行う前の状態です。

25.4.3 アンロード

機能

固定長または可変長の順/相対/索引ファイルから可変長の順ファイルを創成します。

コマンド形式

```
cobfulod -o出力ファイル名 -iファイル属性　入力ファイル名[,LFS]
```

出力ファイル名

創成する可変長の順ファイルのパス名を指定します。

ファイル属性

アンロードする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

以下の文字で指定します。

S：　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

以下の文字で指定します。ただし、ファイル編成に索引ファイルを指定した場合は、ファイル中の属性情報が使用されるため、レコード形式の指定はできません。

f：　固定長
v：　可変長

レコード長

可変長の場合のレコード長は、最大レコード長で指定します。ただし、ファイル編成に索引ファイルを指定した場合は、ファイル中の属性情報が使用されるため、レコード長の指定はできません。

入力ファイル名

アンロード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、固定長または可変長の順/相対/索引ファイルである必要があります。
COBOLで作成した大容量ファイルを扱う場合、入力ファイル名に"ファイル名,LFS"を指定します。

例

相対ファイルから順ファイルを創成

```
$ cobfulod -ooutfile -iR,f,80 relfile
```

注意

- 入力ファイルのファイル編成が実際のファイルと異なる場合、正しく処理できないことがあります。
- 入力ファイルのレコード長が実際のファイルと異なる場合、正しく処理できません。

25.4.4 表示

機能

ファイルの内容をレコード単位に文字形式と16進数形式で表示します。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。

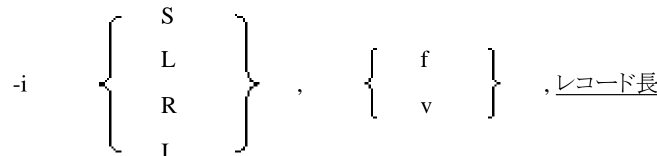
表示範囲を指定しなかった場合は、ファイル中の全レコードを表示します。

コマンド形式

```
cobfbrws -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[,LFS]
```

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

以下の文字で指定します。

S： 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

以下の文字で指定します。ただし、ファイル編成に索引ファイルを指定した場合は、ファイル中の属性情報が使用されるため、レコード形式の指定はできません。

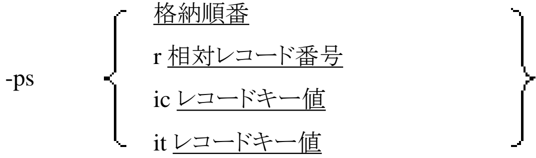
f： 固定長
v： 可変長

レコード長

可変長の場合のレコード長は、最大レコード長で指定します。ただし、ファイル編成に索引ファイルを指定した場合は、ファイル中の属性情報が使用されるため、レコード長の指定はできません。

開始位置

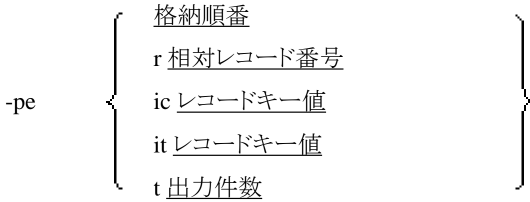
表示を開始するレコード位置を以下の形式で指定します。



順/行順ファイルでは、レコードの格納順番を指定します。相対ファイルでは、キーワード文字“r”に続けて相対レコード番号を指定します。索引ファイルでは、レコードキー値を文字形式で指定するか16進形式で指定するかによって指定方法が異なります。文字形式の場合は、キーワード文字列“ic”に続けてレコードキー値を指定します。16進形式の場合は、キーワード文字列“it”に続けてレコードキー値を指定します。
開始位置を省略した場合は、ファイルの先頭レコードから表示されます。

終了位置

表示を終了するレコード位置を以下の形式で指定します。



出力件数の指定以外は、開始位置と同様です。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は、ファイルの最後のレコードまで表示されます。

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

レコードが昇順に表示されます。順/行順ファイルでは、レコードの格納順番の小さいレコードから表示されます。相対ファイルでは、相対レコード番号の小さいレコードから表示されます。索引ファイルでは、レコードキーの値が小さいレコードから表示されます。

D

レコードが降順に表示されます。相対ファイルでは、相対レコード番号の大きいレコードから表示されます。索引ファイルでは、レコードキーの値が大きいレコードから表示されます。なお、順/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は、昇順を指定したものとみなされます。

検索キー番号

索引ファイルを表示する場合、レコードを検索するレコードキーの番号を指定します。レコードキー番号とは、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は、主レコードキーが指定されたものとみなされます。

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。
COBOLで作成した大容量ファイルを扱う場合、入力ファイル名に"ファイル名,LFS"を指定します。

例

順ファイルの内容を表示

```
$ cobfbrws -iS,v,80 -ps5 -pet10 seqfile
```

相対ファイルの内容を表示

```
$ cobfbrws -iR,f,50 -psr20 -per10 -poD relfile
```

索引ファイルの内容を表示

```
$ cobfbrws -iI -psit0001 -peit0010 -poA -pk1 ixdfile
```

- 入力ファイルのファイル編成が実際のファイルと異なる場合、正しく処理できないことがあります。
- 入力ファイルのレコード長が実際のファイルと異なる場合、正しく処理できません。

25.4.5 整列

機能

レコード中の任意のデータ項目をキーとして、ファイル中のレコードが昇順または降順に整列し、整列された結果を可変長の順ファイルに出力します。

環境変数BSORT_TMPDIRにより、整列用作業ファイル(~SRTnnnn(nnnnは英数字))を作成するディレクトリを指定することができます。整列用作業ファイルを作成するディレクトリを指定する場合は、そのディレクトリのパス名を指定してください。環境変数BSORT_TMPDIRの指定がない場合は、/tmpに整列用作業ファイルが作成されます。

COBOLで作成した大容量ファイルを入力する場合は、PowerSORTが必要です。

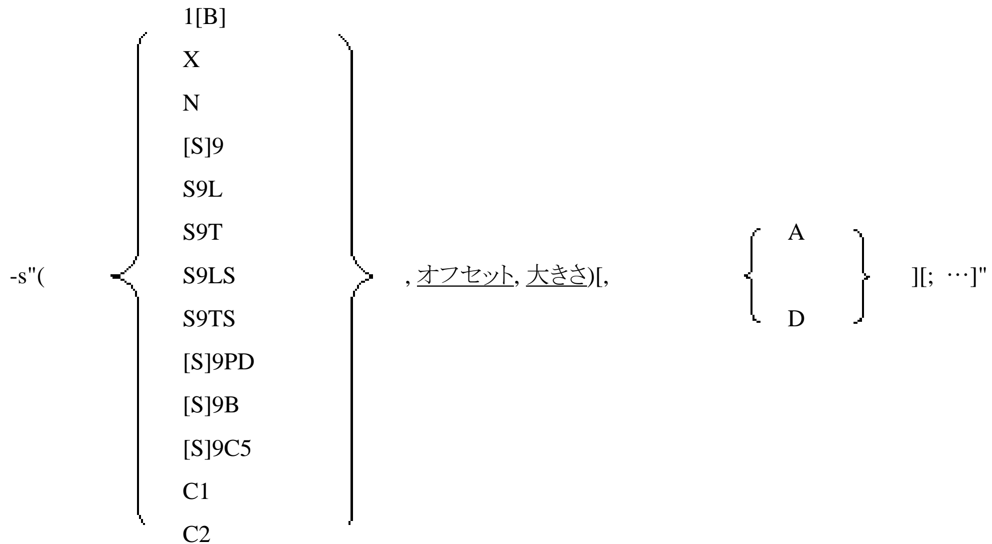
```
cobfsort -o出力ファイル名 -s整列条件 -iファイル属性  入力ファイル名[,LFS]
```

出力ファイル名

整列結果を出力するファイルのパス名を指定します。

整列条件

レコードを整列するときに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。



キーの項目属性

以下の値で指定します。指定する項目属性のキーワード文字はCOBOLのデータ属性として表現します。

```
1[B]   :  PIC  1()[BIT]
X      :  PIC  X()
N      :  PIC  N()
[S]9   :  PIC  [S]9()
S9L    :  PIC  S9()  LEADING
S9T    :  PIC  S9()  TRAILING
S9LS   :  PIC  S9()  LEADING  SEPARATE
S9TS   :  PIC  S9()  TRAILING  SEPARATE
[S]9PD :  PIC  [S]9()  PACKED-DECIMAL
[S]9B  :  PIC  [S]9()  BINARY
[S]9C5 :  PIC  [S]9()  COMP-5
C1     :  COMP-1
C2     :  COMP-2
```

オフセット

キー項目のレコード内オフセットをレコードの先頭を0とした相対バイト位置で指定します。

大きさ

キー項目の大きさをPICTURE句での数字項目の桁数、英数字項目または日本語項目の文字数で指定します。キー項目に“1B”を指定した場合、キーの大きさは無条件に1バイトになり、大きさは指定できません。
大きさではなく、1バイトのマスク値を10進数で指定してください。キー項目に“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、キー項目“C1”に対しては4、キー項目“C2”に対しては8を指定します。

整列順序

レコードをキー項目の属性に従って昇順に整列する(キーワード文字“A”)か、降順(キーワード文字“D”)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。

参考

データ属性に“1B”を指定した場合、整列キーの値は、指定されたレコード中の位置から1バイトのデータとマスク値に指定された値との論理積になります。たとえば、オフセットに“1”を、マスク値に“227”(10進表示)を指定した場合、レコードの内容が“05ad”(16

進表現)であれば、整列キーの値は“a1”(16進表現)となります。これは、レコード中の相対1バイト目の値“ad”(16進表現)とマスク値“e3”(“227”の16進表現)との論理積です。

注意

指定できる整列条件の最大個数は64です。

ファイル属性

整列する入力ファイルの属性を以下の形式で指定します。

```
        ┌ S ┐     ┌ f ┐
-i      │ L │  ,  │   │  ,レコード長
        │ R │     └ v ┘
        └ I ┘
```

ファイル編成

以下の文字で指定します。

S： 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

以下の文字で指定します。ただし、ファイル編成に索引ファイルを指定した場合は、ファイル中の属性情報が使用されるため、レコード形式の指定はできません。

f： 固定長
v： 可変長

レコード長

可変長の場合のレコード長は、最大レコード長で指定します。ただし、ファイル編成に索引ファイルを指定した場合は、ファイル中の属性情報が使用されるため、レコード長の指定はできません。

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。
COBOLで作成した大容量ファイルを扱う場合、入力ファイル名に“ファイル名,LFS”を指定します。

例

相対ファイルを整列して、順ファイルに出力

```
$ cobfsort -ooutfile -s"(N,0,2),A;(X,10,5),D" -iR,v,80 relfile
```

注意

- 入力ファイルのレコード長が実際のファイルと異なる場合、正しく処理できません。
- 入力ファイルのファイル編成が実際のファイルと異なる場合、正しく処理できないことがあります。

25.4.6 属性

機能

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。

コマンド形式

```
cobfattr　入力ファイル名[,LFS]
```

入力ファイル名

属性情報を表示する索引ファイルのパス名を指定します。
COBOLで作成した大容量ファイルを扱う場合、入力ファイル名に“ファイル名,LFS”を指定します。

```
$ cobfattr ixdfile
```

索引ファイル以外は、属性情報を表示できません。

25.4.7 復旧

機能

プロセスの異常終了などのために、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、データに異常が認められ、復旧できないレコードが存在した場合には、それらのデータが未復旧データファイルとして可変長の順ファイルに出力されます。

コマンド形式

```
cobfrcov　復旧ファイル名　未復旧データファイル名[,LFS]
```

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。

COBOLで作成した大容量ファイルを復旧する場合、復旧ファイル名として“ファイル名,LFS”を指定します。

未復旧データファイル名

復旧不可能であった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。なお、復旧できないデータがある場合だけ、未復旧データを作成します。

```
$ cobfrcov ixdfile seqfile
```

注意

- 復旧機能を使用した場合、“環境変数TMPDIR”に指定されたディレクトリまたは“/tmp”に復旧ファイルと同じ大きさの一時的な作業ファイル(先頭が“~UTY”で始まる名前のファイル)が生成されます。
- 復旧機能では、ファイル中に保持しているファイル情報をもとに処理を行います。したがって、この情報が破壊されている場合は復旧できません。

- 復旧機能では、復旧前のファイルを書き換えます。このため、復旧前のファイルを保存するためには、復旧機能の実行前にファイルをあらかじめ複写しておく必要があります。
- 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。

25.4.8 再編成

機能

索引ファイルの空きブロックを可能なかぎり削除し、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。

コマンド形式

```
cobfreog -o出力ファイル名　入力ファイル名[,LFS]
```

出力ファイル名

再編成後の(新しく作成される)索引ファイルのパス名を指定します。

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。
COBOLで作成した大容量ファイルを扱う場合、入力ファイル名に“ファイル名,LFS”を指定します。

例

```
$ 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。

第26章　分散開発支援機能

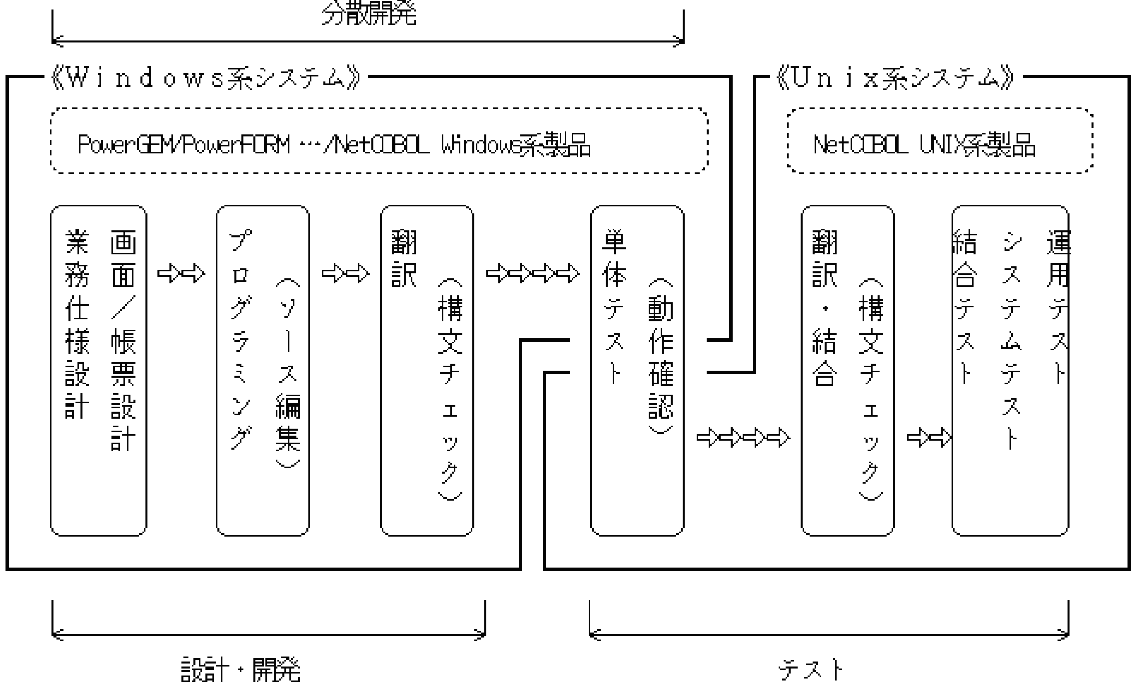
NetCOBOLは、COBOL85規格および国際規格COBOL2002の一部を採用しています。この仕様の範囲でプログラムを記述することにより、各種システムで動作するプログラムを開発できます。また、開発過程の一部でNetCOBOLのWindows版製品を使用することで、より効率のよいプログラム開発を行うことができます。

本章では、Windows版製品の開発環境と連携しての分散開発と、それを支援する機能の概要について説明します。

26.1 分散開発の概要

NetCOBOLのWindows版製品を使用して、UNIX系システムで動作するプログラム(以降、UNIX系システムで動作するプログラムのことをUNIX系プログラムといいます)を開発することができます。

図26.1 UNIX系プログラムの分散開発作業の流れ



26.1.1 分散開発のメリット

UNIX系システムは、優れたネットワーク機能と安定性、セキュリティ強度の高さなどの利点により、各種サーバ用途に多く利用されてきました。

しかし、通常の業務クライアントにはPCが採用される場合が多く、PCとその標準的なプラットフォームであるWindowsに対する操作性により親しみを感じるという人も少なくありません。このような場合、各種のコマンドを駆使して行われるUNIX系システムの標準の開発作業は効率的でないものになりがちです。

また、Windows系システムでは、大規模なチーム開発においてプログラム資産の構成管理に力を発揮するPowerGEM Plus Administratorなどの先進的な技術を取り入れた帳票設計ツールPowerFORM、Interstage Form Coordinatorといったツールを使用することができます。

このように分散開発では、使い慣れたWindowsの操作環境を利用することによって、より効率的なUNIX系プログラムの開発を行うことができます。

26.1.2 分散開発の機能範囲

UNIX系システムとWindows系システムの機能範囲を“図26.2 UNIX系システムとWindows系システムの機能範囲”に示します。

図26.2 UNIX系システムとWindows系システムの機能範囲

"図26.2 UNIX系システムとWindows系システムの機能範囲"に示すように、NetCOBOLのUNIX系製品とWindows版製品でサポートする機能にほとんど違いがありません。

このためほとんどの場合において、プログラムの作成から単体テストまでをWindows系の開発環境を利用して行うことができます。

ただし、"図26.2 UNIX系システムとWindows系システムの機能範囲"で共通機能範囲として示した以外の機能を使用する場合、コード系の違いや関連するソフトウェア製品の違いなどから、Windows系システム上とUNIX系システムで動作などが異なります。

日本語定数/日本語16進定数

シフトJISの範囲に含まれない値を指定した場合、Windows版製品では翻訳エラーになります。

文字比較

半角カナ文字、日本語文字の大小順序の結果が異なる場合があります。

索引ファイルのキー順序

半角カナ文字、日本語文字の大小順序の結果が異なる場合があります。

ソートキーの順序

半角カナ文字、日本語文字の大小順序の結果が異なる場合があります。

日本語字類条件

コード系の違いにより、結果が異なる場合があります。

拡張日本語印刷

使用できる文字、書体などが異なります。

データベース機能

使用するデータベース製品ごとの違いから、翻訳、実行の結果に違いがでる場合があります。

Web連携機能

Windows系とUNIX系で使用できる機能に違いがあります。また、Windows系とUNIX系ではファイルやパス名の規則が異なります。

26.2 分散開発支援機能

NetCOBOLのWindows版製品では、UNIX系プログラムの分散開発をより効率的に行うために、プロジェクトマネージャとNetCOBOL Studioで次のような機能を提供しています。

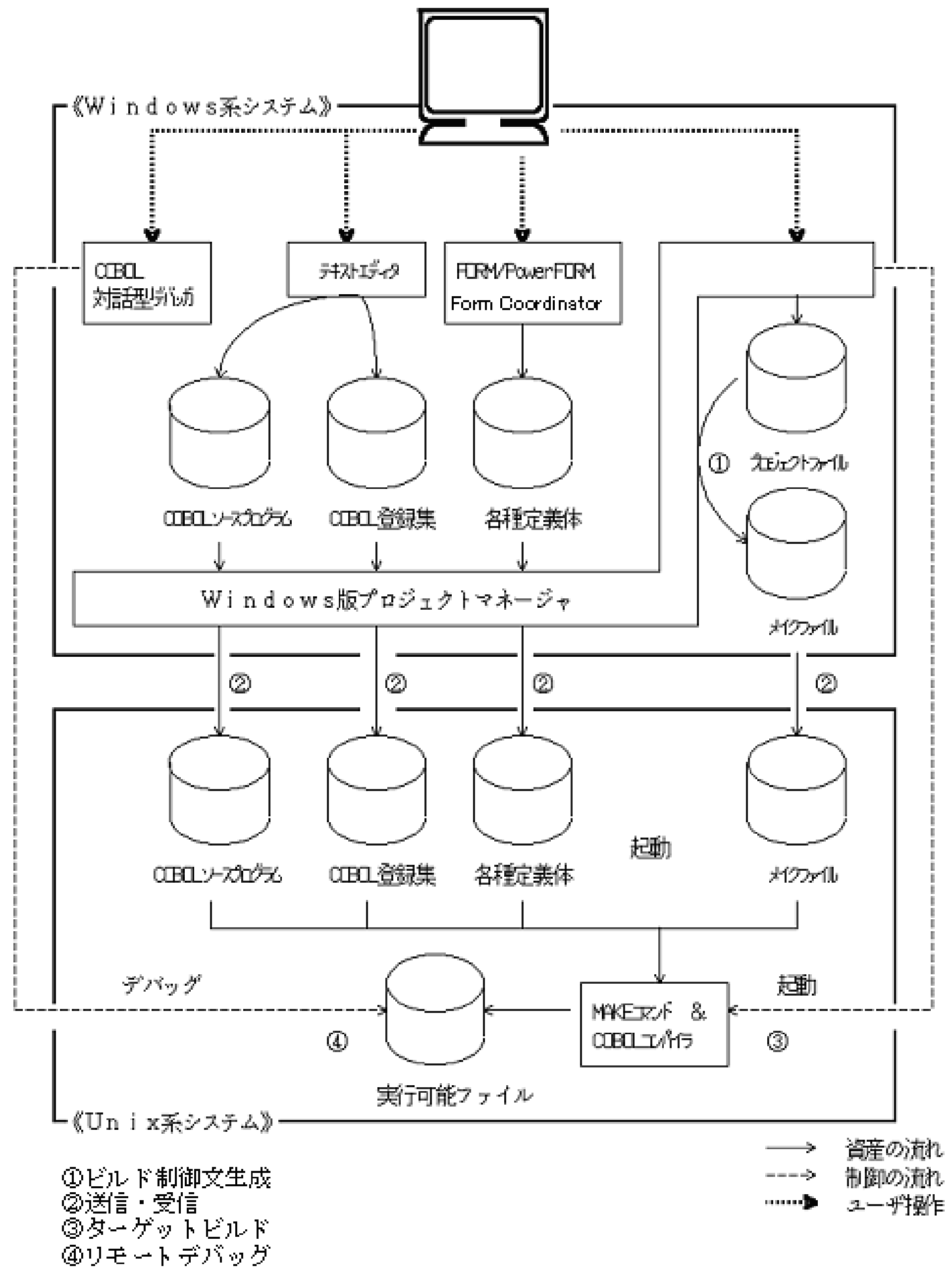
プロジェクトマネージャ

- プログラム資産（ソースプログラム、登録集、各種定義体等）をWindows系システムとUNIX系システムの間で送受信する（分散開発：送信・受信機能）
- Windows版製品のプロジェクトマネージャのプロジェクトファイルからメイクファイルを作成する（分散開発：ビルド制御文生成）
- Windows版製品のプロジェクトマネージャからUNIX系システム上のプログラムをビルドする（分散開発：ターゲットビルド）
- UNIX系システム上のプログラムをWindows系システムからデバッグ実行する（リモートデバッグ）

NetCOBOL Studio

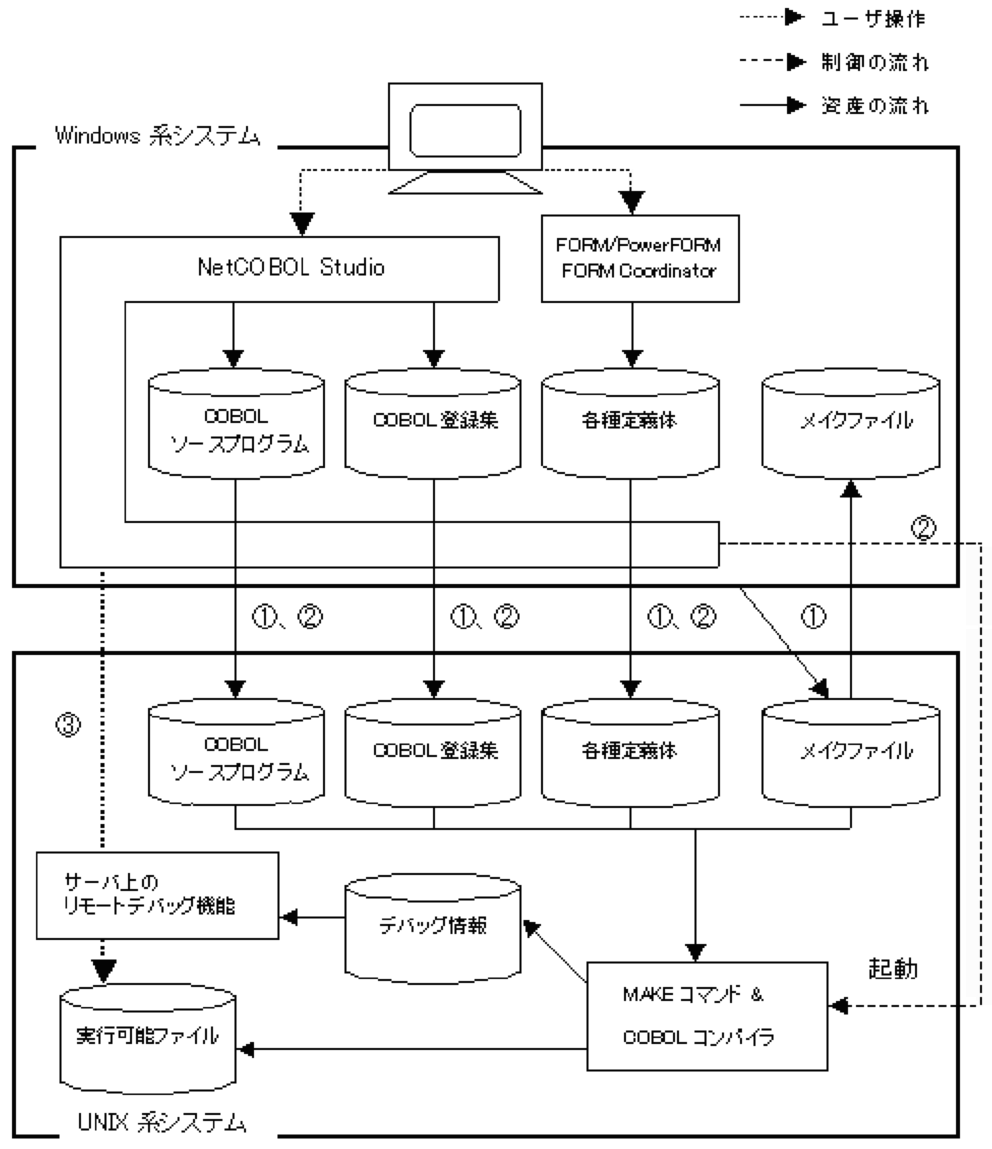
- Windows版製品のNetCOBOL StudioのプロジェクトからCOBOL資産をサーバへ転送して、メイクファイルを作成する(リモート開発：メイクファイル生成)
- Windows版製品のNetCOBOL StudioのプロジェクトからUNIX系システム上のプログラムをビルドする(リモート開発：ビルド、再ビルド)
- Windows版製品のNetCOBOL StudioのプロジェクトからUNIX系システム上のプログラムをデバッグする(リモート開発：デバッガ)

図26.3 分散開発支援機能の概要(プロジェクトマネージャの場合)



これらの機能を使用するための設定･操作の方法の詳細については、Windows版製品のヘルプおよび添付マニュアル“UNIX分散開発の手引き”を参照してください。

図26.4 分散開発支援機能の概要(NetCOBOL Studioの場合)



① リモート開発 ： メイクファイル生成 （ビルドに必要な資源が送信されます。）

② リモート開発 ： ビルド・リビルド （前回、ビルド以降に更新された資源が送信されます。）

③ リモート開発 ： デバッグ

これらの機能を使用するための設定･操作の方法の詳細については、Windows版製品のヘルプおよび添付マニュアル“NetCOBOL Studio 使用手引書”を参照してください。

第27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。

27.1 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。

従業員データ

```
870128,富士通太郎,38,営業部,1234-5678
020053,静岡花子,27,総務部,2234-5963
020395,東京二郎,29,開発企画部,3234-0223
   :
```

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.
    :
```

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構文を追加して、容易に操作できるようにしました。

27.2 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。

27.2.1 基本操作

集団項目に格納されている文字列データを、従属する項目単位でCSV形式へ編集する場合、STRING文(書き方2)を使用します。

```
WORKING-STORAGE SECTION.
77 従業員編集  PIC X(80).
01 従業員.
  02 従業員番号  PIC 9(6).  *> 870128      が格納されている状態
  02 氏名        PIC N(10). *> 富士通太郎       (以下同)
  02 年齢        PIC 9(2).  *> 38
  02 所属        PIC N(10). *> 営業部
  02 内線        PIC X(10). *> 1234-5678
         :
    MOVE SPACE TO 従業員編集.
    STRING 従業員 INTO 従業員編集 BY CSV-FORMAT.
```

上の例のSTRING文を実行すると、データ項目“従業員編集”には、以下のCSV形式データが格納されます。

```
870128,富士通太郎,38,営業部,1234-5678
```

なお、CSV形式を生成する際、格納されているデータを以下のとおり編集します。

- 送出し側項目と受取り側項目の字類が異なる場合、受取り側項目の字類に合わせ、以下のように変換します。
  - 送出し側項目が日本語の場合、Unicode動作時にはエンコードの変換を行います。
  - 送出し側項目が符号つき数字項目の場合、SIGN句の指定に関わらず、符号を左端に付加します。また、小数部を含む場合、小数点文字を付加します。
- 送出し側データ中に区切り文字が含まれていた場合、データ全体を、二重引用符で囲みます。
- 送出し側データ中に二重引用符が含まれていた場合、連続する2つの二重引用符に置き換え、データ全体を二重引用符で囲みます。
- 送出し側項目の字類が英数字または日本語の場合、後置空白は削除します。
- 送出し側項目が数字項目の場合、先行するゼロ列は削除します。ただし、値がゼロだった場合は、1けたのゼロを転記します。また、小数部を含む場合、後置ゼロは削除します。

- TYPE指定に従い、データ全体を二重引用符で囲みます。詳細は、"27.4 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。

- 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。

27.2.2 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27.1 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27.1 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。

27.3 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。

27.3.1 基本操作

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

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

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

なお、CSV形式データの分解では、データを以下のとおり編集します。

- 送出し側項目と受取り側項目の字類が異なる場合、受取り側項目の字類に合わせ、次のような変換を行います。
  - 受取り側項目が日本語の場合、Unicode動作時には文字コードの変換を行います。
  - 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(Tab Separated Values)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]"6.3.3 行順ファイルの処理"、"6.8.4.2 ファイルの高速処理"

27.3.2 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27.2 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27.2 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。

27.4 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“27.5.2 CBR_CSV_TYPE(生成するCSV形式のバリエーション)”で指定します。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。

27.5 環境変数の設定

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

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する場合、実行環境変数“CBR_CSV_OVERFLOW_MESSAGE=NO”を指定してください。

- JMP0262I-W
- JMP0263I-W

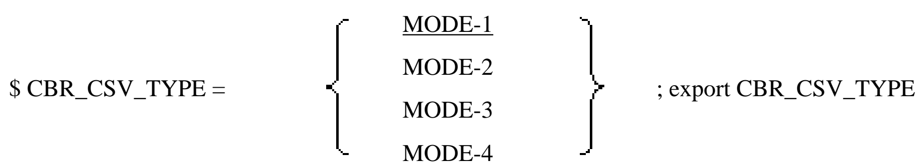
```
$ CBR_CSV_OVERFLOW_MESSAGE=NO ; export CBR_CSV_OVERFLOW_MESSAGE
```

パラメタに“NO”以外の文字を指定した場合、実行時メッセージの出力は抑止されません。

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

STRING文(書き方2)で生成するCSV形式のバリエーションを、実行環境変数“CBR_CSV_TYPE”で指定することができます。

この指定は、TYPE指定を省略したSTRING文だけに有効です。



パラメタを省略した時は、MODE-1が選択されたものとみなします。

生成されるCSV形式の詳細は、“27.4 CSV形式のバリエーション”を参照してください。

付録A　翻訳オプション

ここでは、翻訳オプションについて説明します。

指定する翻訳オプションがわからないときには、"A.1 翻訳オプション一覧"から翻訳オプションを確認し、"A.2 翻訳オプションの指定形式"を参照してください。

A.1　翻訳オプション一覧

以下に、翻訳オプション一覧を示します。

翻訳リストに関するもの

- "A.2.6 COPY(登録集原文の表示)"
- "A.2.19 LINECOUNT(翻訳リストの1ページあたりの行数)"
- "A.2.20 LINESIZE(翻訳リストの1行あたりの文字数)"
- "A.2.21 LIST(目的プログラムリストの出力の可否)"
- "A.2.23 MAP(データマップリスト、プログラム制御情報リストおよびセクションサイズリストの出力の可否)"
- "A.2.24 MESSAGE(オプション情報リスト、翻訳単位統計情報リストの出力の可否)"
- "A.2.29 NUMBER(ソースプログラムの一連番号領域の指定)"
- "A.2.38 SOURCE(ソースプログラムリストの出力の可否)"
- "A.2.48 XREF(相互参照リストの出力の可否)"

翻訳時メッセージに関するもの

- "A.2.5 CONF(規格の違いによるメッセージの出力の可否)"
- "A.2.13 FLAG(診断メッセージのレベル)"
- "A.2.14 FLAGSW(COBOL文法の言語要素に対しての指摘メッセージ表示の可否)"

COBOLプログラムの解釈に関するもの

- "A.2.1 ALPHAL(英小文字の扱い)"
- "A.2.2 BINARY(2進項目の扱い)"
- "A.2.9 CURRENCY(通貨編集用文字の扱い)"
- "A.2.11 DUPCHAR(重複文字の扱い)"
- "A.2.15 INITVALUE(作業場所節でのVALUE句なし項目の扱い)"
- "A.2.16 KANA(文字コードの扱い)"
- "A.2.18 LANGLVL(ANSI COBOL規格の指定)"
- "A.2.27 NCW(日本語利用者語の文字集合の指定)"
- "A.2.28 NSPCOMP(日本語空白の比較方法の指定)"
- "A.2.32 QUOTE/APOST(表意定数QUOTEの扱い)"
- "A.2.33 RSV(予約語の種類)"
- "A.2.35 SDS(符号付き10進項目の符号の整形の可否)"
- "A.2.36 SHREXT(マルチスレッドモデルのプログラムの外部属性に関する扱い)"
- "A.2.39 SRF(正書法の種類)"

- “A.2.42 STD1（英数字の文字の大小順序の指定）”
- “A.2.43 TAB（タブの扱い）”
- “A.2.49 ZWB（符号付き外部10進項目と英数字項目の比較）”

ソースプログラムの解析に関するもの

- “A.2.34 SAI（ソース解析情報ファイルの出力の可否）”

目的プログラムの作成に関するもの

- “A.2.8 CREATE（創成ファイルの指定）”
- “A.2.10 DLOAD（プログラム構造の指定）”
- “A.2.17 LALIGN（連絡節のデータ宣言の扱い）”
- “A.2.22 MAIN（主プログラム/副プログラムの指定）”
- “A.2.25 MODE（ACCEPT文の動作の指定）”
- “A.2.26 NAME（翻訳単位ごとのオブジェクトファイルの出力の可否）”
- “A.2.30 OBJECT（目的プログラムの出力の可否）”
- “A.2.31 OPTIMIZE（広域最適化の扱い）”
- “A.2.45 THREAD（マルチスレッドモデルのプログラム作成の指定）”

実行時の処理に関するもの

- “A.2.12 EQUALS（SORT文での同一キーデータの処理方法）”
- “A.2.47 TRUNC（桁落とし処理の可否）”

実行時の資源に関するもの

- “A.2.37 SMSIZE（PowerSORTが使用するメモリ容量を指定）”
- “A.2.40 SSIN（ACCEPT文のデータの入力先）”
- “A.2.41 SSOUT（DISPLAY文のデータの出力先）”

実行時のデバッグ機能に関するもの

- “A.2.3 CHECK（CHECK機能の使用の可否）”
- “A.2.4 CODECHK（実行時のコード系チェックの指定）”
- “A.2.7 COUNT（COUNT機能の使用の可否）”
- “A.2.44 TEST（対話型デバッガの使用の可否）”
- “A.2.46 TRACE（TRACE機能の使用の可否）”

A.2 翻訳オプションの指定形式

以下に翻訳オプションの指定形式を示します。

- 翻訳オプションは、アルファベット順に並んでいます。
- 翻訳オプションの指定方法は、以下の2種類があります。優先順位は、2. > 1.となります。
    1. コマンドオプション-WCによる指定
    2. ソースプログラム中の翻訳指示文 (@OPTIONS)による指定

- ソースプログラム中の翻訳指示文に指定された場合、各翻訳オプションの内容によって指定できる翻訳単位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。

-WC	コマンドオプション-WCで指定可能
@	翻訳指示文で指定可能

A.2.1 ALPHAL(英小文字の扱い)

-WC,@

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。

参照

"3.1.1.3 COB_COPYNAME(登録集原文の検索条件の指定)"

"3.1.3 登録集(COPY文)を使ったプログラムの翻訳方法"の"注意"

"9.3.6 注意事項"

A.2.2 BINARY(2進項目の扱い)

-WC,@

```
BINARY( { WORD[, { MLBON  } ] } )
        {        { MLBOFF }   }
        { BYTE                }
```

2進データの基本項目が割り付けられる領域長を指定します。桁数より求められるワード単位の領域長 (2,4,8) (BINARY(WORD))か、バイト単位の領域長 (1～8)(BINARY(BYTE))か、を指定します。なお、符号なし2進項目の最左端ビットの扱いも指定できます。

- BINARY(WORD,MLBON):最左端ビットは符号

- BINARY(WORD,MLBOFF):最左端ビットは数値

注意

BINARY(BYTE)を指定した場合、最左端ビットは数値として扱われます。

参考

宣言した桁数と、割り当てられる領域長の関係は、下表のとおりです。

PICの桁数		割り当てられる領域長	
符号付き	符号なし	BINARY(BYTE)	BINARY(WORD)
1～2	1～2	1	2
3～4	3～4	2	2
5～6	5～7	3	4
7～9	8～9	4	4
10～11	10～12	5	8
12～14	13～14	6	8
15～16	15～16	7	8
17～18	17～18	8	8

A.2.3 CHECK（CHECK機能の使用の可否）

```
-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およびPRMの検査を行います。
- CHECK(BOUND)：
  添字・指標および部分参照の範囲外検査を行います。
- CHECK(ICONF)：
  INVOKE文のパラメタと呼び出すメソッドの仮パラメタの適合検査を行います。
- CHECK(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。しかし、領域破壊などによりプログラムが期待どおり動作しない場合があります。なお、nに0を指定した場合には、メッセージの表示回数に関係なく、プログラムの処理が続行されます。
- CHECKを指定すると、上記の検査をするための処理が目的プログラム中に組み込まれるため、実行性能が低下します。デバッグ終了時には、NOCHECKを指定して再翻訳してください。
- ON SIZE ERROR指定またはNOT ON SIZE ERROR指定の算術文では、ON SIZE ERRORの除数のゼロ検査が行われ、CHECK(NUMERIC)の除数のゼロ検査は行われません。
- CHECK(NUMERIC)のデータ例外検査は、外部10進項目または内部10進項目が参照で使用される場合、および、英数字項目または集団項目から、外部10進項目または内部10進項目へ転記される場合に行われます。ただし次は、チェックの対象とはなりません。
  - 添字としてALLが指定されている表要素
  - SEARCH ALL文におけるキー項目(ただしキー項目に対する添字が1次元かつWHEN条件がひとつのみである場合は除く)
  - SORT/MERGE文におけるキー項目
  - SQL文中で使用されているホスト変数
  - CALL文、INVOKE文および行内呼び出しのBY REFERENCEパラメタ
  - 次の組み込み関数の引数
    - FUNCTION ADDR
    - FUNCTION LENG
    - FUNCTION LENGTH
  - 英数字項目または集団項目から、外部10進項目または内部10進項目のオブジェクトプロパティへの転記

参照

"5.3 CHECK機能の使い方"

A.2.4 CODECHK(実行時のコード系チェックの指定)

```
-WC,@
```

{ CODECHK | NOCODECHK }

実行時に翻訳時の日本語コード系のチェックを行う(CODECHK)か、行わない(NOCODECHK)か、を指定します。

日本語のコード系に依存しないプログラム(シフトJIS/EUC/Unicode共通プログラム)を作成する場合、NOCODECHKを指定する必要があります。

A.2.5 CONF(規格の違いによるメッセージの出力の可否)

-WC, @

```
{ CONF( { 68  } ) }
        { 74  }
        { OBS }
  NOCONF
```

(NOCONF is underlined.)

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 ANSI COBOLと '85 ANSI COBOLとで意味の解釈が異なる項目を指摘します。
- CONF(74):
  '74 ANSI COBOLと '85 ANSI 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。

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)を指定した場合にだけ意味を持ちます。

参照

"A.2.18 LANGLVL(ANSI COBOL規格の指定)"

参考

CONFは、従来の規格に従って作成したプログラムを、'85 ANSI COBOLの規格に従うように変更する場合に有効です。

A.2.6 COPY(登録集原文の表示)

-WC, @

```
{ COPY   }
{ NOCOPY }
```

(NOCOPY is underlined.)

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。

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。

参照

"A.2.38 SOURCE(ソースプログラムリストの出力の可否)"

A.2.7 COUNT(COUNT機能の使用の可否)

```
-WC,@
```

{ COUNT | <u>NOCOUNT</u> }

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と同時に指定できません。同時に指定された場合、あとに指定された方が有効となります。

参照

"5.4 COUNT機能の使い方"

A.2.8 CREATE(創成ファイルの指定)

```
-WC,@
```

CREATE ({ <u>OBJ</u> | 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ません。したがって目的プログラムは生成されないので、-cオプションを同時に指定してください。

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、常にCREATE(OBJ)とみなされます。

A.2.9 CURRENCY(通貨編集用文字の扱い)

```
-WC,@
```

CURRENCY ({ <u>¥</u> | $ })

──

通貨編集用文字として使用している文字に、¥を使用する(CURRENCY(¥))か、$を使用する(CURRENCY($))か、を指定します。

A.2.10 DLOAD(プログラム構造の指定)

-WC,@

```
{ DLOAD   }
{ NODLOAD }
```

(NODLOADに下線)

──

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。

参照

“3.2.2 結合の種類とプログラム構造”

“9.1.2 動的プログラム構造”

“16.6.2 動的プログラム構造での翻訳処理”

A.2.11 DUPCHAR(重複文字の扱い)

-WC,@

```
DUPCHAR ( { STD } )
          { EXT }
```

(STDに下線)

──

以下のソースプログラムをUnicode環境で翻訳した時、コンパイラが付加／置換する全角ハイフンをシステム標準(DUPCHAR(STD))とするか、拡張文字(DUPCHAR(EXT))とするか、を指定します。

- 3バイト項目制御部を指定した画面帳票定義体を取り込んでいる。
- COPY文の書き方2と書き方3で日本語利用者語を使用している。

項目制御部については、“7.5.3 帳票定義体の作成”を参照してください。また、COPY文の書き方については、“COBOL文法書”を参照してください。

注意

EUCまたはシフトJISの全角ハイフンをUnicodeにコード変換した時、システム標準のiconvを使用して変換した場合とInterstage Charset Managerの標準コード変換を使用して変換した場合とで結果が異なります。

システム標準のiconvを使用して変換した場合にはDUPCHAR(STD)を、Interstage Charset Managerの標準コード変換を使用して変換した場合にはDUPCHAR(EXT)を指定してください。

参照

“J.3.2 JIS非漢字の負号について”

A.2.12 EQUALS（SORT文での同一キーデータの処理方法）

```
-WC,@
```

```
{ EQUALS   }
{ NOEQUALS }
```

（NOEQUALSに下線）

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があります。それらに関して、SORT文の出力でのレコードの順序をSORT文の入力で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文の出力での同一キーを持つレコードの順序は規定されません。

注意

EQUALSを指定すると、整列操作で入力順序を保証するための特別な処理が行われるために実行性能が低下します。

A.2.13 FLAG（診断メッセージのレベル）

```
-WC,@
```

```
FLAG( { I } )
      { W }
      { E }
```

（Iに下線）

表示する診断メッセージを指定します。

- FLAG(I)：すべての診断メッセージを表示します。
- FLAG(W)：Wレベル以上の診断メッセージだけ表示します。
- FLAG(E)：Eレベル以上の診断メッセージだけ表示します。

注意

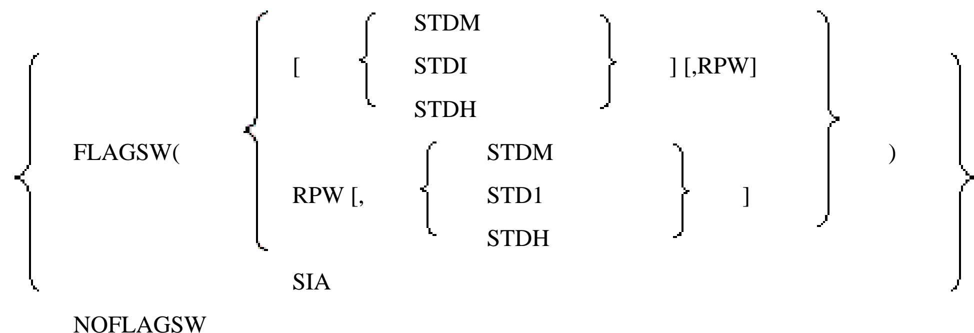
翻訳オプションCONFによる指摘メッセージは、FLAGの指定に関係なく表示されます。

参照

“A.2.5 CONF（規格の違いによるメッセージの出力の可否）”

A.2.14 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）

```
-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 ANSI COBOL規格の下位レベル外
- FLAGSW(STDI): '85 ANSI COBOL規格の中位レベル外
- FLAGSW(STDH): '85 ANSI COBOL規格の上位レベル外
- FLAGSW(RPW) : '85 ANSI 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

参考

FLAGSW(SIA)は、他システムで動かすプログラムを作成するときに有効です。

A.2.15 INITVALUE（作業場所節でのVALUE句なし項目の扱い）

```
-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。

xxは、2桁の16進数を指定してください。xxは省略できません。

A.2.16 KANA（文字コードの扱い）

```
-WC,@
```

KANA({ EUC | JIS8 })

文字定数および英字・英数字項目内のカナ文字のコード系を指定します。

- KANA(EUC) : カナ文字の文字コードは、2バイトコード(EUC)となります。
- KANA(JIS8): カナ文字の文字コードは、1バイトコード(JIS)となります。

KANA指定はロケールがEUCの場合にだけ意味を持ちます。

ロケールがEUC以外の場合、KANA指定は意味を持ちません。

A.2.17 LALIGN（連絡節のデータ宣言の扱い）

```
-WC,@
```

{ LALIGN | <u>NOALIGN</u> }

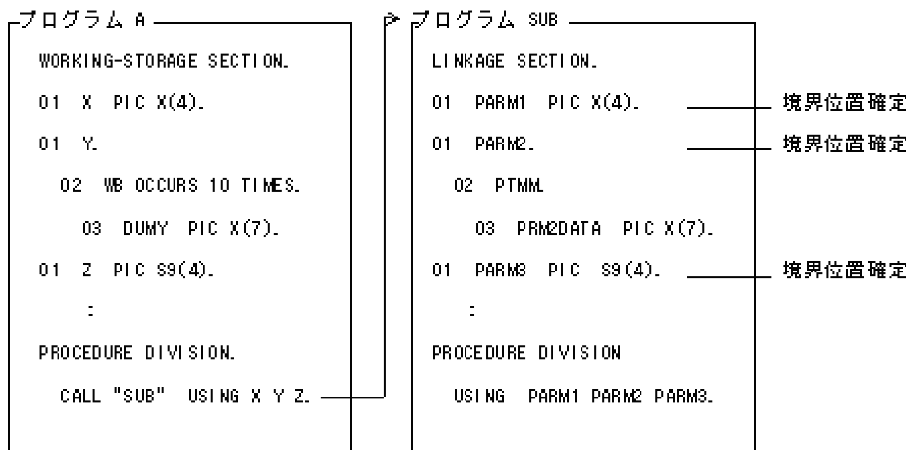
連絡節に宣言されたデータを参照する場合、8バイトの整列境界にあっていることを前提としたオブジェクトを生成する(LALIGN)か、前提としないオブジェクトを生成する(NOLALIGN)か、を指定します。

なお、整列境界が8バイト境界にあっていることを前提としたオブジェクトを生成する場合、データの処理速度が向上します。

参考

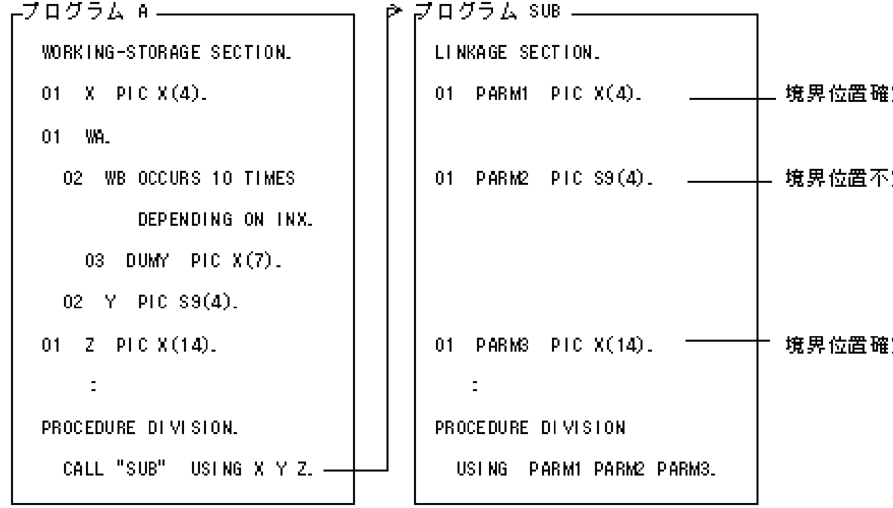
呼出し元のプログラムでLINKAGE SECTIONの各データに対応するすべてのデータが01項目で宣言されている場合、当オプションが使用できます。

例1) 整列境界があっている(オプションが指定されると性能が向上する)場合



プログラムAからSUBに渡すパラメタX,Y,ZともにWORKING-STORAGE SECTION の先頭に記述され、かつ、01項目であるため、境界位置が8バイト境界にあっています。

例2) 整列境界があっていない(オプションを指定できない)場合



プログラムAからSUBに渡すパラメタX,Zについては、WORKING-STORAGE SECTIONの先頭に記述されかつ、01項目であるため、境界位置が確定しています。Yは、可変長項目を含む集団項目内に指定されていて、実行時にアドレスが決まるため、不定となります。このように、データの整列境界があっているかを利用者が判断するのは困難となります。

注意

整列境界があっていないデータを含むソースプログラムの翻訳時に、当オプションが指定されていた場合、翻訳はできます。しかし、実行時の動作はシステムに依存(異常終了やシステムエラーなど)します。

A.2.18 LANGLVL（ANSI COBOL規格の指定）

```
-WC,@
```

LANGLVL({ 85 | 74 | 68 })

（85は下線付き：省略値）

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。

- LANGLVL(85)： '85 ANSI COBOL
- LANGLVL(74)： '74 ANSI COBOL
- LANGLVL(68)： '68 ANSI COBOL

A.2.19 LINECOUNT（翻訳リストの1ページあたりの行数）

```
-WC,@
```

LINECOUNT(n)

翻訳リストの1ページあたりの行数を指定します。
nは、3桁以内の整数を指定してください。
本オプションを指定しなかった場合、LINECOUNT(0)が指定されたものとみなします。

注意

0から12までの値を指定すると、ページ替えのない出力となります。

A.2.20 LINESIZE(翻訳リストの1行あたりの文字数)

```
-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には、0、80または120以上の3桁の整数を指定することができます。
0を指定した場合、行の途中で改行せずにソースプログラムリストを出力します。
本オプションを指定しなかった場合、LINESIZE(0)が指定されたものとみなします。

注意

- オプション情報リスト、診断メッセージリストおよび翻訳単位統計情報リストは、翻訳オプションLINESIZEに指定した最大文字数に関係なく固定の文字数(120)で出力されます。
- 文字数として有効な最大の値は136です。翻訳オプションLINESIZEに136より大きい値を指定した場合、136として扱われます。

A.2.21 LIST(目的プログラムリストの出力の可否)

```
-WC,@
```

{ LIST / <u>NOLIST</u> }

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オプションによって、翻訳リストの出力を有効にしている場合だけ出力されます。

翻訳リストの出力については、"3.3.1.10 -dp(翻訳リストファイルのディレクトリの指定)"、および"3.3.1.16 -P(翻訳リストのファイル名の指定)"を参照してください。

A.2.22 MAIN(主プログラム/副プログラムの指定)

```
-WC,@
```

{ MAIN / <u>NOMAIN</u> }

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。

注意

- 主プログラムとなるCOBOLソースプログラムにMAINを指定してください。ただし、複数翻訳およびプロジェクトから翻訳する場合には、〔主プログラム〕ボタンで指定してください。

- 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。

A.2.23 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）

```
-WC,@
```

{ MAP | <u>NOMAP</u> }

データマップリスト、プログラム制御情報リストおよびセクションサイズリストを出力する(MAP)か、しない(NOMAP)か、を指定します。

これらのリストは、-Pオプションによって、翻訳リストの出力を有効にしている場合だけ出力されます。

翻訳リストの出力については、"3.3.1.10 -dp（翻訳リストファイルのディレクトリの指定）"、および"3.3.1.16 -P（翻訳リストのファイル名の指定）"を参照してください。

A.2.24 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）

```
-WC,@
```

{ MESSAGE | <u>NOMESSAGE</u> }

オプション情報リストおよび翻訳単位統計情報リストを出力する(MESSAGE)か、しない(NOMESSAGE)か、を指定します。

A.2.25 MODE（ACCEPT文の動作の指定）

```
-WC,@
```

MODE ({ <u>STD</u> | 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場合の転記方法を指定します。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.2.26 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 | <u>NONAME</u> }

複数の翻訳単位(プログラム、クラスまたはメソッド定義)が記述された1つのソースファイルを翻訳する場合があります。そのとき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。

当オプションを指定すると、翻訳単位ごとに外部名.oというオブジェクトファイルが出力されます。NAMEを指定すると、1つのソースファイルに複数の翻訳単位が存在する場合、それぞれのオブジェクトファイルが出力されます。

当オプションは、OBJECT指定時だけ有効となります。また、-cオプションの指定にかかわらず、オブジェクトファイルの出力だけ行います。リンクは行いません。

A.2.27 NCW（日本語利用者語の文字集合の指定）

```
-WC,@
```

NCW ({ STD(下線) | 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1水準
- JIS第2水準
- JIS非漢字(以下の文字)

　０、１、…、１
　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.2.28 NSPCOMP（日本語空白の比較方法の指定）

```
-WC,@
```

NSPCOMP ({ NSP(下線) | ASP })

後述する比較において、日本語空白を、日本語空白として扱う(NSPCOMP(NSP))か、ANK空白とみなす(NSPCOMP(ASP))か、を指定します。日本語空白をANK空白とみなす場合には、日本語空白は2バイトのANK空白と等価なものとして扱われます。

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

注意

- 以下の場合、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。
  - INSPECT文
  - STRING文
  - UNSTRING文
  - 索引ファイルのキー操作
- NSPCOMP(ASP)が指定された場合、字類条件JAPANESEでのANK空白が日本語として扱われます。
- ロケールがEUCの場合で、かつKANA(JIS8)オプションが有効な場合、動作結果は保証されません。

参考

OSIV系システムのコード系(JEF)では、日本語空白がANK空白の2文字分と同じ値を持っており、この特性を利用した文字比較が多用されています。しかし、シフトJISやEUCの場合は同じ値を持たないため、システムで動作していたCOBOLプログラムを本システムに移植する場合、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処理についてはソースプログラムを修正しなくても、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

“J.3.1 日本語空白と英数字空白の文字コード”

A.2.29 NUMBER(ソースプログラムの一連番号領域の指定)

```
-WC,@
```

{ NUMBER | <u>NONUMBER</u> }

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使用する値を指定します。ソースプログラムで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。このとき、後続の行番号と同一の行番号が生成された場合は、一意の補正された番号がCOPY修飾値と同じ表現形式で付加されます。

- 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。

- NONUMBER:
  行番号は、1から1きざみに昇順に与えられます。

- 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とみなされません。
- NUMBERを指定した場合、ビルダのエラージャンプ機能は使用できません。
- NUMBERを指定した場合、翻訳オプションSRFにFREEは指定できません。翻訳オプションSRFにFREEを指定した場合、プログラムの動作は保証されません。

A.2.30 OBJECT(目的プログラムの出力の可否)

```
-WC
```

{ OBJECT | NOOBJECT }

(OBJECT is underlined as the default.)

目的プログラムを出力する(OBJECT)か、しない(NOOBJECT)かを指定します。

目的プログラムは、ソースプログラムと同じディレクトリに格納されます。

"3.1.6 COBOLコンパイラが使用するファイル"

A.2.31 OPTIMIZE(広域最適化の扱い)

```
-WC,@
```

{ OPTIMIZE | NOOPTIMIZE }

(OPTIMIZE is underlined as the default.)

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。

TESTと同時に指定した場合、NOOPTIMIZEとして翻訳が行われます(広域最適化は行われません)。

参照

“付録C 広域最適化”

A.2.32 QUOTE/APOST(表意定数QUOTEの扱い)

```
-WC,@
```

{ QUOTE | APOST }

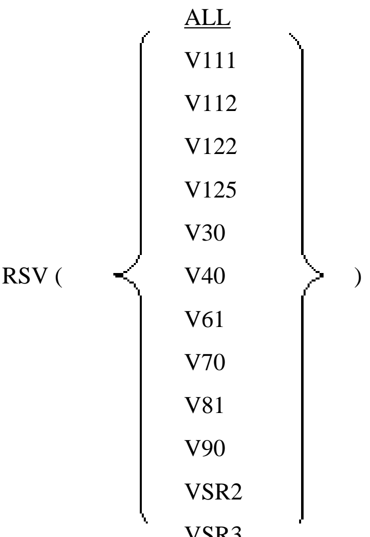
表意定数QUOTEおよびQUOTESとしてクォーテーションマーク(”) を使う(QUOTE)か、アポストロフィ(’)を使う(APOST)か、を指定します。

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、同じである必要があります。

A.2.33 RSV(予約語の種類)

```
-WC,@
```



予約語の種類を指定します。

以下に予約語集合名の意味を示します。

- 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 および Sun日本語COBOL用
- RSV(V30)：COBOL85 V30用
- RSV(V40)：COBOL97 V40用 および COBOL拡張オプション用
- RSV(V61)：COBOL97 V61用
- RSV(V70)：NetCOBOL 7.0用
- RSV(V81)：NetCOBOL V8.0用
- RSV(V90)：NetCOBOL V9.0用
- RSV(VSR2)：VS COBOLII REL2.0用
- RSV(VSR3)：VS COBOLII REL3.0用

A.2.34 SAI(ソース解析情報ファイルの出力の可否)

```
-WC,@
```

{ SAI | 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。

参照

"3.1.6 COBOLコンパイラが使用するファイル"

A.2.35 SDS(符号付き10進項目の符号の整形の可否)

```
-WC,@
```

{ SDS | NOSDS }

符号付き内部10進項目から符号付き内部10進項目への転記で、送出し側項目の符号をそのまま転記する(SDS)か、整形された符号を転記する(NOSDS)か、を指定します。

負符号にはX'B'およびX'D'の2種類があり、そのほかは正符号として扱われます。ここでいう整形された符号とは、送出し側項目の符号が正ならばX'C'に、負ならばX'D'に変換することです。

A.2.36 SHREXT(マルチスレッドモデルのプログラムの外部属性に関する扱い)

```
-WC,@
```

{ SHREXT | NOSHREXT }

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。このオプションは、オブジェクト形式をマルチスレッドモデルとして翻訳(-Tmオプション指定またはTHREAD(MULTI)指定)する場合に有効となります。

注意

オブジェクト形式をプロセスモデルとして翻訳(THREAD(SINGLE))する場合は、このオプションの指定に関係なくNOSHREXTとして翻訳されます。ただし、SHREXTが指定された場合、オプション情報リストの確定翻訳オプションには、SHREXTと表示されます。

A.2.37 SMSIZE（PowerSORTが使用するメモリ容量を指定）

```
-WC,@
```

SMSIZE(値K)

PowerSORTが使用するメモリ容量をキロバイト単位の数字で指定します。

注意

- SORT文およびMERGE文から呼び出されるPowerSORTが使用するメモリ空間の容量を限定したい場合に指定します。指定する値は、キロバイト単位の数字です。指定された値を、PowerSORTの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。
このオプションを指定しない場合、作業域の大きさはPowerSORTによって自動的に設定されます。NetCOBOLからPowerSORTを利用している場合は、“PowerSORTユーザーズガイド”の“環境設定”にある“PowerSORTが使用する作業域”で説明している「入力がファイル以外の場合」の値が設定されます。
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が、同時に指定された場合の優先順位は、特殊レジスタSORT-CORE-SIZEが一番強く、以降、実行時オプションsmsize、翻訳オプションSMSIZE()の順で弱くなります。

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

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

- このオプションは、別製品PowerSORTをインストールしている場合に有効であり、インストールしていない場合には無効です。

A.2.38 SOURCE（ソースプログラムリストの出力の可否）

```
-WC,@
```

{ SOURCE | <u>NOSOURCE</u> }

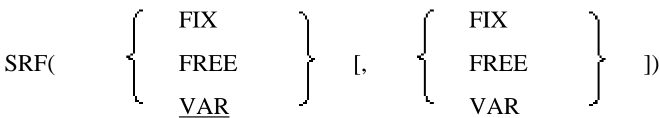
ソースプログラムリストを出力する(SOURCE)か、しない(NOSOURCE)かを指定します。

ソースプログラムリストは、-Pオプションによって、翻訳リストの出力を有効にしている場合だけ出力されます。

翻訳リストの出力については、“3.3.1.10 -dp（翻訳リストファイルのディレクトリの指定）”、および“3.3.1.16 -P（翻訳リストのファイル名の指定）”を参照してください。

A.2.39 SRF(正書法の種類)

```
-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、最初にCOBOLソースプログラムを、次に登録集を指定します。

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

SRFにFREEを指定した場合、翻訳オプションNUMBERは指定できません。翻訳オプションNUMBERを指定した場合、プログラムの動作は保証されません。

参考

ソースファイルと登録集ファイルの正書法が同じ場合、登録集ファイルの正書法の指定は省略することができます。

A.2.40 SSIN(ACCEPT文のデータの入力先)

```
-WC,@
```

SSIN ({ 環境変数名 | SYSIN })

小入出力のACCEPT文のデータの入力先を指定します。

- SSIN(環境変数名):
  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標準入力を使用します。

注意

環境変数名は英大文字(A～Z)で始まる8文字以内の英大文字および数字である必要があります。また、環境変数名は、ほかのファイルで使用する環境変数名(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。

参照

"10.1 小入出力"

A.2.41 SSOUT(DISPLAY文のデータの出力先)

```
-WC,@
```

SSOUT ({ 環境変数名 | SYSOUT })

(SYSOUT is underlined)

小入出力のDISPLAY文のデータの出力先を指定します。

- SSOUT(環境変数名):
  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標準出力を使用します。

注意

環境変数名は英大文字(A～Z)で始まる8文字以内の英大文字または数字で構成されている必要があります。また、環境変数名は、ほかのファイルで使用する環境変数名(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。

参照

“10.1 小入出力”

A.2.42 STD1(英数字の文字の大小順序の指定)

```
-WC,@
```

STD1({ ASCII | JIS1 | JIS2 })

(JIS2 is underlined)

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)の取扱いについて指定します。ASCII(ASCII)として取り扱うか、JIS8単位コード (JIS1)として取り扱うか、またはJIS7単位ローマ字コード (JIS2)として取り扱うかを指定します。

注意

ALPHABET句でEBCDIC指定を記述した場合、このオプションの指定に応じて、以下に示す文字符号系を採用します。

- STD1(ASCII): EBCDIC(ASCII)
- STD1(JIS1): EBCDIC(カナ)
- STD1(JIS2): EBCDIC(英小)

A.2.43 TAB(タブの扱い)

```
-WC
```

```
TAB ( { 8 (underlined) | 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.2.44 TEST（対話型デバッガの使用の可否）

-WC,@

```
{ TEST | NOTEST (underlined) }
```

実行時にデバッガを使用する(TEST)か、しない(NOTEST)か、を指定します。

デバッガが使用するデバッグ情報ファイルは、通常、ソースプログラムと同じディレクトリに作成されます。作成先を変更したい場合は、格納先を指定してください。

注意

OPTIMIZEと同時に指定した場合、NOOPTIMIZEとして翻訳が行われます(広域最適化は行われません)。ただし、確定翻訳オプションにはOPTIMIZEと表示されます。

参照

“3.1.6 COBOLコンパイラが使用するファイル”

“3.3.1.7 -Dt（対話型デバッガを使用する指定）”

“3.3.1.8 -dd（デバッグ情報ファイルのディレクトリの指定）”

“第23章 対話型デバッガの使い方”

“A.2.31 OPTIMIZE（広域最適化の扱い）”

A.2.45 THREAD（マルチスレッドモデルのプログラム作成の指定）

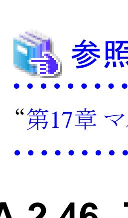
-WC,@

```
THREAD ( { MULTI | SINGLE (underlined) } )
```

オブジェクトの形式をマルチスレッドモデルとする(THREAD(MULTI))か、プロセスモデルとする(THREAD(SINGLE))か、を指定します。

注意

マルチスレッドモデル(THREAD(MULTI))を指定してできた目的プログラムは、マルチスレッドモデルのプログラムとしてリンクする必要があります。したがって、-cオプションまたは-Tmオプションを同時に指定してください。



参照

“第17章 マルチスレッド”

A.2.46 TRACE（TRACE機能の使用の可否）

```
-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、あとに指定された方が有効となります。

参照

“5.2 TRACE機能の使い方”

“A.2.7 COUNT（COUNT機能の使用の可否）”

A.2.47 TRUNC（桁落とし処理の可否）

```
-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。桁落としを行うと実行速度が遅くなる場合には、桁落としは行いません。

```
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
```

注意

- NOTRUNCで、送出し側項目の整数部の桁数が、受け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、プログラムを設計する必要があります。
- 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の機能(桁落としが行われない)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。

参照

"A.2.31 OPTIMIZE（広域最適化の扱い）"

A.2.48 XREF（相互参照リストの出力の可否）

```
-WC,@
```

{ XREF / <u>NOXREF</u> }

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オプションによって、翻訳リストの出力を有効にしている場合だけ出力されます。

翻訳リストの出力については、"3.3.1.10 -dp（翻訳リストファイルのディレクトリの指定）"、および"3.3.1.16 -P（翻訳リストのファイル名の指定）"を参照してください。

注意

翻訳オプションXREFが指定されている場合で、翻訳の結果、最大重大度コードがSレベル以上の場合、相互参照リストの出力は抑止されます。

A.2.49 ZWB（符号付き外部10進項目と英数字項目の比較）

```
-WC,@
```

{ <u>ZWB</u> / NOZWB }

符号付き外部10進項目を英数字フィールドと比較するときに、外部10進項目の符号部を無視して比較する(ZWB)か、符号部を含めて比較する(NOZWB)か、を指定します。ここで、英数字とは、英数字項目、英字項目、英数字編集項目、数字編集項目、文字定数およびZERO以外の表意定数のことです。

例

```
    77  ED    PIC  S9(3)  VALUE  +123.
    77  AN    PIC   X(3)  VALUE  "123".
```

この場合、条件式 ED = AN の真偽は、以下のようになります。

```
ZWB を指定した場合     :  真
NOZWB を指定した場合   :  偽
```

A.3 プログラム定義にだけ指定可能な翻訳オプション

以下に示す翻訳オプションは、プログラム定義の翻訳時にだけ指定できます。

- BINARY(BYTE)
- CONF(OBS)
- NOFLAGSW以外のFLAGSW
- LANGLVL(74)またはLANGLVL(68)
- MAIN

A.4 メソッド原型定義と分離されたメソッド定義間での翻訳オプション

メソッド原型定義および分離されたメソッド定義のそれぞれの翻訳時に指定された翻訳オプションは、一致している必要があります。

ただし、以下の翻訳オプションについては、一致している必要はありません。

- CHECK
- CONF
- COPY
- COUNT
- CURRENCY
- DLOAD
- FLAG
- LINECOUNT
- LINESIZE
- LIST
- MAP
- MESSAGE
- NUMBER
- OBJECT
- QUOTE/APOST
- SAI
- SOURCE
- SRF
- TEST
- TRACE
- XREF

付録B　入出力状態一覧

ここでは、入出力文を実行した時にファイル管理記述項のFILE STATUS句に指定されたデータ名に設定される値(入出力状態値)の意味を説明します。

表B.1 入出力状態値

大分類	入出力状態値	詳細情報	意味
成功	00	－	入出力文の実行が成功しました。
	02	－	入出力文の実行は成功しましたが、以下の状態のどちらかです。 ・ 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ます。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ありません。
	04	－	READ文の実行は成功しましたが、入力したレコードの長さが最大レコード長よりも大きい。
		MeFt	表示ファイルでREAD文の実行は成功しましたが、以下の状態のどれかになりました。 ・ 入力したレコードの長さが最大レコード長よりも大きい。 ・ 全桁必須入力項目で入力エラーが発生しました。 ・ 入力必須項目で入力エラーが発生しました。 ・ 日本語データエラーが発生しました。 ・ ANKデータエラーが発生しました。 ・ 数字項目構成データエラーが発生しました。 ・ 数字項目符号エラーが発生しました。 ・ 数字項目小数点エラーが発生しました。 ・ リダンダンシチェックエラーが発生しました。 ・ データ項目に入力されたデータに誤りがあります。
		ACM	READ文の実行は成功しましたが、入力したレコードの長さが最大レコード長よりも大きい。
	05	－	OPTIONAL句指定のファイルで、入出力文の実行は成功しましたが、以下の状態のどれかになりました。 ・ ファイルに対してINPUT/I-O/EXTENDモードのOPEN文を実行しました。しかし、ファイルが未創成状態でした。 ・ ファイルに対してINPUTモードのOPEN文を実行しました。しかし、ファイルが存在しませんでした。このときファイルは生成されず、最初のREAD文の実行時にファイル終了条件(入出力状態値='10')となります。 ・ ファイルに対してI-OまたはEXTENDモードのOPEN文を実行しました。しかし、ファイルが存在しませんでした。このときファイルは生成されます。
	07	－	入出力文の実行は成功しましたが、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りました。 ・ NO REWIND指定のOPEN文 ・ NO REWIND指定のCLOSE文 ・ REEL/UNIT指定のCLOSE文
	0A	ACM	READ文の実行は成功しましたが、指定した論理宛先に入力すべきメッセージが存在しませんでした。

大分類	入出力状態値	詳細情報	意味
	0B	ACM	WRITE文の実行は成功しましたが、指定した論理宛先の最大格納メッセージ数を超えてメッセージを出力しました。
ファイル終了条件 (不成功)	10	―	順呼出しのREAD文でファイル終了条件となりました。 ・ ファイルの終わりに達しました。 ・ 存在しない不定入力ファイルに対して初めてのREAD文を実行しました。ここで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モードで開いたとき、そのファイルが未創成状態である場合のことです。
	14	―	順呼出しのREAD文でファイル終了条件となりました。 ・ 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。
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す。 ・ 順呼出しで、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ました。 ・ 乱呼出しまたは動的呼出しで、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ました。 ・ 順呼出しで、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ません。
	22	―	WRITE文またはREWRITE文の実行時、書こうとしたレコードの相対レコード番号、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。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きます。
	23	―	レコードが見つかりません。 ・ 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ません。 ・ 相対ファイルで、相対レコード番号に0が指定されました。
	24	―	次の状態のどれかです。 ・ WRITE文の実行で、領域不足が発生しました。 ・ WRITE文の実行で、指定されたキーがキーレンジ外です。 ・ 区域外書出し発生後、さらにWRITE文を実行しようとしました。
永続誤り条件 (不成功)	30	―	物理的なエラーが発生しました。またはラージファイル対応システムでLFSまたはLBSAM指定により作成したサイズの大きい大容量ファイルをLFSまたはLBSAM指定なしでファイルをオープンしました。
永続誤り条件 (不成功)	30	―	物理的なエラーが発生しました。
		ACM	ACMがエラーを検出しました。
	34	―	WRITE文の実行で、領域不足が発生しました。
	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文を実行しました。しかし、ファイルが未創成状態でした。
	37	―	指定された機能は未サポートです。
	38	―	以前にCLOSELOCK文を実行したファイルに対してOPEN文を実行しました。
	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ました。
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文を実行しました。
	42	―	開いていないファイルに対して、CLOSE文を実行しました。

大分類	入出力状態値	詳細情報	意味
	43	ー	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文ではありませんでした。
	44	ー	次の状態のどちらかです。 ・ 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大きいか、またはレコード長として誤った数値が指定されました。 ・ REWRITE文実行時に、そのレコードは書き換えるレコードの長さと等しくありませんでした。
	46	ー	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す。 ・ 先行するSTART文が不成功です。 ・ 先行するREAD文が不成功(ファイル終了条件も含む)です。
	47	ー	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	ー	以下のどちらかの状態でないファイルに対してWRITE文を実行しました。 ・ OUTPUTまたはEXTENDモードで開かれた順ファイル ・ OUTPUT/EXTEND/I-Oモードで開かれた相対または索引ファイル
	49	ー	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文を実行しました。
その他の誤り (不成功)	90	ー	ほかのどれにも含まれないエラーです。次のような状態です。 ・ ファイル情報が不完全またはその情報に誤りがあります。 ・ OPEN/CLOSE文の実行時に、OPEN/CLOSE関数でエラーが発生しました。 ・ 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ました。 ・ 主記憶などの資源が利用できません。 ・ 正しく閉じられていないファイルに対してOPEN文を実行しました。 ・ 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ました。 ・ no-space状態発生後、レコードを書き出そうとしました。 ・ テキストファイルのレコードに不当な文字があります。 ・ ネイティブからコードセットに訳せない文字があります。 ・ 同一のファイルに対し、多数のアプリケーションからOPEN要求がありました。その結果、ロックテーブルに不足が発生しました。 ・ 必要な関連製品のローディングに失敗しました。 ・ システムエラーが発生しました。 ・ 上記以外の誤りが存在します。その入出力動作に関しては、それ以上の情報がありません。
		MeFt	MeFtがエラーを検出しました。
		ACM	ACMがエラーを検出しました。
	91	ー	・ ファイルが割り当てられていません。 ・ 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されていません。
	92	RDM	排他エラーが発生しました。(RDMファイル)
	93	ー	排他エラーが発生しました。(ファイルロック)

大分類	入出力状態値	詳細情報	意味
	99	ー	排他エラーが発生しました。(レコードロック)
		MeFt	システム異常が発生しました。
		ACM	システム障害が発生しました。
	9A	ACM	すでにメッセージを入力しているなどの誤りが発生しました。
	9B	ACM	宛先異常が発生しました。
	9C	ACM	メッセージオーバが発生しました。
	9F	ACM	タイムオーバが発生しました。
	9G	ACM	論理宛先の最大メッセージ数オーバが発生しました。
	9H	ACM	論理宛先のファイル領域不足または記憶域不足が発生しました。

注:FORMAT句付き印刷ファイルおよび表示ファイルの場合は、入出力状態値と詳細情報が通知されます。詳細情報の“MeFt”はMeFtの通知コードを、“ACM”はPowerRW+またはACM制御のエラーコードを参照してください。“ー”は詳細情報に0000が通知されます。詳細情報については、以下のマニュアルを参照してください。

- MeFt:MeFtのオンラインマニュアル
- RDM :“RDB/7000説明書”
- ACM :“ACM制御説明書”

付録C　広域最適化

ここでは、コンパイラが行う広域最適化の内容および使用上の注意事項について記述します。

C.1　最適化の項目

広域最適化では、入口が1箇所・出口が1箇所の手続き部を、記述順に実行されるような文の列(これを基本ブロックといいます)に分割します。そして、制御の移行およびデータの使用状態を解析し、主にループ(繰り返し実行される部分)に着目した最適化を行います。以下に、最適化の項目を示します。

- 共通式の除去
- 不変式の移動
- 誘導変数の最適化
- PERFORM文の最適化
- 隣接転記の統合
- 無駄な代入の除去

C.2　共通式の除去

演算や変換で、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1]

```
 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1で、[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1)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1:

```
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
```

[例2]

```
 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１ =  数字１ *  数字２.            … [2]
```

例2で、[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。

C.3 不変式の移動

演算や変換がループ内にあり、ループ内外両方で行っても結果が変わらない場合、これをループ外に移動します。

[例]

```
 77   添字             PIC S9(4)  BINARY.
 77   外部１０進項目   PIC S9(7)  DISPLAY.
 01   集団項目.
      02  ２進項目     PIC S9(7) BINARY  OCCURS 20 TIMES.
       :
    MOVE 1  TO  添字.
ループ開始.
    IF  ２進項目 (添字) = 外部１０進項目  GO TO ループ終了.
       :
    ADD 1 TO  添字.
    IF  添字 IS <= 20 GO TO ループ開始.
ループ終了.
```

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)と比較するときの外部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。

C.4 誘導変数の最適化

ループ内で、定数または値が不変な項目によってだけ再帰的に定義される項目を誘導変数といいます。誘導変数を使用している部分式がある場合、新しい誘導変数を導入することにより、添字計算のための乗算を加算に変更します。

[例]

```
 77   添字             PIC S9(4) COMP-5.
 01   集団項目.
      02  繰り返し項目 PIC X(10) OCCURS 20 TIMES.
       :
ループ開始.
    IF  繰り返し項目(添字) =  ・・・
       :
    ADD 1 TO  添字.                                    …  [1]
    IF  添字 IS <= 20 GO TO ループ開始.                …  [2]
```

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、繰り返し項目(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、[1]のあとに “ADD 10 TO t” が生成されます。さらに、ループ中で添字がほかに使用されず、かつ、ループを出たあと、ループ中で計算した添字の値を未使用の場合、[2]は“IF t IS <= 200 GO TO ループ開始.”で置き換えられ、[1]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
```

C.5 PERFORM文の最適化

PERFORM文は、その復帰機構として、戻り先のアドレスを退避、設定および復元するために、いくつかの機械命令に展開されます。そこで、PERFORM文の出口に、そのPERFORM文以外で制御が渡る場合、復帰機構の機械命令のうちのいくつかが冗長となることがあります。これらの冗長な機械命令は削除されます。

C.6　隣接転記の統合

複数の英数字転記文があり、領域の連続した項目が同じように領域の連続した項目に転記される場合、これらを1つの文にまとめます。

[例]

```
    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]
```

例で、[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。

C.7　無駄な代入の除去

代入で、それ以降一度も明または暗に参照されないデータ項目への代入は、削除されます。

C.8　広域最適化での注意事項

翻訳オプションNOOPTIMIZEが有効でない場合、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。なお、詳細については、“付録A 翻訳オプション”を参照してください。このときの注意事項を以下に示します。

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、複数のパラメタの、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

次のプログラムでは、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も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
- 翻訳オプションTESTを指定したプログラム
  [参照]“A.2.44 TEST(対話型デバッガの使用の可否)”

広域最適化の効果が得にくい場合

次のプログラムでは、広域最適化の効果は得にくくなります。

- 入出力操作を主とし、もともとCPUをあまり使わないプログラム
- 数字項目を使わず、英数字項目ばかりを使うプログラム
- 宣言部分から非宣言部分を参照しているプログラム
- 非宣言部分から宣言部分を参照しているプログラム

- 翻訳オプションTRUNCを指定しているプログラム
  [参照]“A.2.47 TRUNC（桁落とし処理の可否）”

デバッグを行う場合

次の注意が必要です。

- 広域最適化によって文の削除、移動および変更が行われるので、データ例外などのプログラム割込みの起こる回数や場所が変わることがあります。
- プログラム割込みなどで中断したとき、プログラム上の記述でデータ項目に値を設定していても、実際には設定されていないことがあります。
- 再帰的に定義される項目が内部10進項目または外部10進項目の場合、翻訳オプションNOTRUNCが指定されていると、意図したとおりにプログラムが動作しないことがあります。
  [参照]“A.2.47 TRUNC（桁落とし処理の可否）”

付録D　組込み関数の使用

ここでは、組込み関数の用例や、記述の際の注意点について説明します。

D.1　関数の型と記述の関係

関数はそれぞれに型を持っています。そして、その型の違いによってプログラム中に記述できる場所も異なります。関数と型の対応については、“表D.5 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。

NUMVAL関数使用時の注意事項

NUMVAL関数のパラメタに一意名を指定した場合、関数の精度は、整数部桁数および小数部桁数のどちらもパラメタに指定されたデータ項目の領域長となります。つまり、関数の結果は、PIC S9(領域長)V9(領域長)で表すことができます。ただし、各桁数の最大は18桁です。一意名の領域長が18を越える場合でも、関数の結果はPIC S9(18)V(18)となります。また、一意名が可変長の場合は、無条件にPIC S9(18)V(18)となります。

下記の場合、NUMVAL関数の結果はPIC S9(18)V(18)となります。

```
         :
 01 A    COMP-2.
 01 B    PIC X(20) VALUE "1234567890          ".
         :
    COMPUTE A = FUNCTION NUMVAL(B(1:20)).
         :
```

なお関数結果のデータ項目への格納は、転記の規則で行われます。精度については、格納域に指定した各データ項目に関する精度で行われます。

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。また、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
```

EUCコード系で動作するプログラムをKANA(JIS8)オプションを指定して翻訳した場合、NATIONAL関数の実行結果が意図したとおりにならない場合があります。これは、EUCコード系での日本語文字とJIS8単位コード系でのカナ文字との間に文字コード値の重なりがあるためです。

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、“12.3 ADDR関数とLENG関数の使い方”を参照してください。

D.2 引数の型によって決定される関数の型

関数の中には、引数の型によって関数の型が決まるものがあります。最大値を求める関数を例に挙げて説明します。

例

MAX関数

・ COBOL プログラムの記述

```
DATA            DIVISION.
 WORKING-STORAGE SECTION.
  01 C1        PIC X(10).
  01 C2        PIC X(5).
  01 C3        PIC X(5).
  01 V1        PIC S9(3).
  01 V2        PIC S9(3)V9(2).
  01 V3        PIC S9(3).
  01 MAXCHAR   PIC X(10).
  01 MAXVALUE PIC S9(3)V9(2).
 PROCEDURE DIVISION.
     MOVE FUNCTION MAX(C1 C2 C3) TO MAXCHAR.                …[1]
         :
     COMPUTE MAXVALUE =  FUNCTION MAX(V1 V2 V3).            …[2]
```

MAX関数は、最大値を求める関数で、関数の型は引数の型によって決まります。

[1]は、引数の型が英数字であるため、関数の型は英数字となります。また、[2]は引数の型が数字であるため、関数の型は数字となります。

D.3 CURRENT-DATE関数を利用した西暦の取得

小入出力機能を使った日付入力では、西暦の下2桁しか取得できません。CURRENT-DATE関数を利用すると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 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。

CURRENT-DATE機能の使用には、TZ環境変数が必要です。TZ環境変数に関する詳しい情報については、

```
$ man tzset
```

を実行してください。

- sh

```
$ TZ="JST-9" ; export TZ
```

- csh

```
$ setenv TZ "JST-9"
```

環境変数CBR_JOBDATEに任意な日付を指定すると、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。

- 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。

環境変数の指定形式については、"10.1.9 任意の日付の入力"を参照してください。

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。しかし、受取り側が数字項目であった場合は、数字転記が行われます。数字転記と集団項目転記では、桁よせの規則が異なるため、以下のように4桁の領域を準備しても、西暦は取得できません。

```
DATA            DIVISION.
 WORKING-STORAGE SECTION.
  77 LAG      PIC 9(4).
 PROCEDURE DIVISION.
     MOVE FUNCTION CURRENT-DATE TO LAG.
```

転記後の変数LAGの内容は、西暦ではなく、グリニッジ標準時からの進みまたは遅れ(CURRENT-DATE関数の関数値の、18～21桁)を示します。

D.4 任意の基準日からの通日計算

通日計算の結果得られた値の差を取り、任意の基準日からの通日を知ることができます。

例では、任意の基準日から現在の日付までの通日を計算し、その期間内での利息計算を行っています。

例

- COBOL プログラムの記述

```
DATA DIVISION.
 WORKING-STORAGE SECTION.
  01 TODAY      PIC X(8).
  01 TODAY-R    REDEFINES TODAY  PIC 9(8).
  01 FROM-DAY   PIC 9(8).
  01 INT        PIC 9(5).
  01 PAY        PIC 9(6).
  01 EDT-PAY    PIC ZZZ,ZZ9.
  01 EDT-INT    PIC ZZZZ9.
  01 EDT1       PIC XXXX/XX/XX.
  01 EDT2       PIC XXXX/XX/XX.
 PROCEDURE DIVISION.
*> 基準日を設定する
     ACCEPT FROM-DAY
     MOVE FROM-DAY TO EDT1
*> 今日の日付を取得する
     MOVE FUNCTION CURRENT-DATE TO TODAY EDT2
*>  2つの日付間の日数を計算する
     COMPUTE INT =  (FUNCTION INTEGER-OF-DATE(FROM-DAY))
                  - (FUNCTION INTEGER-OF-DATE(TODAY-R))
*> 利息計算 (例 : 20日あたり133.3円固定)
     COMPUTE PAY = FUNCTION INTEGER-PART((INT / 20) * 133.3)
     MOVE PAY TO EDT-PAY.
     MOVE INT TO EDT-INT.
*> 結果の表示
     DISPLAY "預入日(" EDT1 ")から " EDT-INT "日 経過しています。" .
     DISPLAY  EDT2 " 現在の利息合計は " EDT-PAY "円です。" .
```

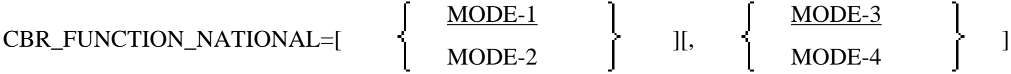
- 実行結果(基準日として"19920521"を入力した場合)

```
預入日(1992/05/21)から2272日 経過しています。
1998/08/10 現在の利息合計は  15,062円です。
```

D.5 NATIONAL関数の変換モード

NATIONAL関数の変換モードは、環境変数CBR_FUNCTION_NATIONALに指定します。

D.5.1 CBR_FUNCTION_NATIONAL(NATIONAL関数の変換モードの指定)



NATIONAL関数の変換モードには、第1オペランドに従来(V40以前)どおりの変換を行う(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。

コード系（ロケール）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、“MODE3”が指定されたものとみなします。

本指定に誤りがある場合、全体を省略したものとみなします。

コード系が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。

MODE1とMODE2の違いを以下に示します。

表D.1 Unicode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D.2 EUCの場合、かつ、翻訳オプションKANA(EUC)が有効な場合

指定	変換方法
MODE1	「`」(0x60)を「‘」(0xA1C6)に変換します。
MODE2	「`」(0x60)を「｀」(0xA1AE)に変換します。

表D.3 EUCの場合、かつ、翻訳オプションKANA(JIS8)が有効な場合

指定	変換方法
MODE1	「ｰ」(0xB0)を「-」(0xA1BD)に変換します。 「`」(0x60)を「‘」(0xA1C6)に変換します。
MODE2	「ｰ」(0xB0)を「ー」(0xA1BC)に変換します。 「`」(0x60)を「｀」(0xA1AE)に変換します。

MODE3とMODE4の違いを以下に示します。

表D.4 Unicode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。

D.6 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。

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.
```

- 引数の値(種子)が同じ場合は、同じ疑似乱数が返却されます。ただし、種子が異なる場合および種子を指定しない場合でも、同じ値が返却されることもあります。
- 関数値の範囲内であっても、返却されない疑似乱数はあります。

D.7 組込み関数一覧

組込み関数の一覧を"表D.5 組込み関数一覧"に記述します。

表D.5 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	項類および時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

分類	関数	用途	関数の型
	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で、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で、指定した文字の順序位置を求めます。	整数
数値操作	INTEGER	指定した値を超えない最大の整数を求めます。	整数
	INTEGER-PART	整数部を求めます。	整数
	RANDOM	乱数を求めます。	数字
金利計算	ANNUITY	元金を1とし、利率と期間から均等払いの比率の近似値を求めます。	数字
	PRESENT-VALUE	減価率による現在の価格を求めます。	数字
日付操作	CURRENT-DATE	現在の日付、時刻、グリニッジ標準時からの時差を求めます。	英数字
	DATE-OF-INTEGER	通日に対応する年月日を求めます。	整数
	DAY-OF-INTEGER	通日に対応する年日を求めます。	整数
	INTEGER-OF-DATE	年月日に対応する通日を求めます。	整数
	INTEGER-OF-DAY	年日に対応する通日を求めます。	整数
	WHEN-COMPILED	プログラムが翻訳された日時を求めます。	英数字
算術計算	SQRT	平方根の近似値を求めます。	数字
	FACTORIAL	階乗を求めます。	整数
	LOG	自然対数を求めます。	数字
	LOG10	常用対数を求めます。	数字
	MEAN	平均値を求めます。	数字
	MEDIAN	中央値を求めます。	数字
	MIDRANGE	最小値と最大値の平均値を求めます。	数字
	RANGE	最大値と最小値の差を求めます。	整数または数字
	STANDARD-DEVIATION	標準偏差を求めます。	数字
	MOD	指定した法での指定した値の値を求めます。	整数
	REM	余りを求めます。	数字
	SUM	和を求めます。	整数または数字
	VARIANCE	分散を求めます。	数字
三角関数	SIN	正弦の近似値を求めます。	数字
	COS	余弦の近似値を求めます。	数字
	TAN	正接の近似値を求めます。	数字
	ASIN	逆正弦の近似値を求めます。	数字
	ACOS	逆余弦の近似値を求めます。	数字
	ATAN	逆正接の近似値を求めます。	数字
ポインタ	ADDR	先頭アドレスを求めます。	ポインタデータ

付録E　環境変数一覧

COBOLを実行するために必要な環境変数を、下表に示します。

なお、表中の条件の記号の意味は以下のとおりです。

◎：指定された条件で製品を使用する場合に環境変数の設定が必要です。
　　また、ランタイムシステムでは、実行時の初期化ファイルに環境変数が設定された場合も有効となります。
○：指定された条件で製品を使用する場合に環境変数の設定が必要です。
　　また、ランタイムシステムでは、実行時の初期化ファイルに環境変数が設定されても無効となります。
－：製品を使用する場合に環境変数の設定は必要ありません。

表E.1 COBOLを実行するための環境変数一覧

環境変数名	設定する内容	条件						
		コンパイラ	ランタイムシステム	デバッガ	プロジェクトマネージャ	S-DESIGN	ファイルユーティリティ	
BSORT_TMPDIR	作業用ファイルを作成するディレクトリのパス名	－	◎	－	－	－	○	SORT/MERGE文で使用する作業ファイルのディレクトリを指定する場合(“第11章 SORT文およびMERGE文の使い方～整列併合機能～”参照)
CBR_ATTACH_TOOL	デバッガをデバッグ対象プログラムから起動する場合のデバッガ起動パラメタ	－	◎	◎	－	－	－	デバッガをデバッグ対象プログラムから起動する場合(“第23章 対話型デバッガの使い方”参照)
CBR_CBRFILE	実行用の初期化ファイル名	－	○	－	－	－	－	実行用の初期化ファイルを使用する場合(“第4章 プログラムの実行”参照)
CBR_CBRINFO	文字列YES(実行時の情報を出力する場合) または ENV(実行時の情報および環境変数の情報を出力する場合)	－	◎	－	－	－	－	プログラムの実行時の情報を実行時メッセージ(JMP0070I-I)として出力する場合(“メッセージ説明書”参照) 文字列YESを指定した場合、プログラムの実行時の情報が実行時メッセージ(JMP0070I-I)として出力されます。 文字列ENVを指定した場合、プログラムの実行時の情報が実行時メッセージ(JMP0070I-I)として出力され、実行時の環境変数の情報が環境変数CBR_MESSOUTFILEに指定されたファイルに出力されます。文字列ENVを指定する場合、環境変数CBR_MESSOUTFILEも設定してください。CBR_MESSOUTFILEが設定されていない場合、実行時の環境変数の情報は出力されません。
CBR_CI_CLG	簡易アプリ間通信でのクライアント側ログファイルのパス名 および	－	◎	－	－	－	－	簡易アプリ間通信を行う場合(“第19章 通信機能”参照)

環境変数名	設定する内容	条件						
		コンパイラ	ランタイムシステム	デバッガ	プロジェクトマネージャ	S-DESIGN	ファイルユーティリティ	
	ログファイルサイズ							
CBR_CI_INF	論理あて先定義ファイルのパス名	−	◎	−	−	−	−	簡易アプリ間通信を行う場合(“第19章 通信機能”参照)
CBR_CLASSINFFILE	クラス情報ファイルのパス名	−	◎	−	−	−	−	オブジェクト指向プログラムで使用するクラス情報を変更したい場合(“第16章 オブジェクト指向プログラムの開発と実行”参照)
CBR_CLOSE_SYNC	文字列yes (指定する場合)	−	◎	−	−	−	−	CLOSE文実行時に即時書き出し処理を行う場合
CBR_CODE_CHECK	文字列no (指定する場合)	−	◎	−	−	−	−	プログラムのコード系一致チェックを行わない場合(“付録J 日本語コード系”参照)
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	−	◎	−	−	−	−	機能名CONSOLEに対応付けた呼び名を指定した小入出力の出力先として、Interstage Business Application Serverの汎用ログを使用する場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
CBR_COMPOSER_MESS	Interstage Business Application Serverの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の汎用ログへ出力する場合("4.3.4 実行時メッセージのInterstage Business Application Serverの汎用ログへの出力"参照)
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	−	◎	−	−	−	−	機能名SYSERRに対応付けた呼び名を指定した小入出力の出力先として、Interstage Business Application Serverの汎用ログを使用する場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
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	−	◎	−	−	−	−	UPON指定なしまたは、機能名SYSOUTに対応付けた呼び名を指定した小入出力の出力先として、Interstage Business Application Serverの汎用ログを使用する場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
CBR_CONSOLE	文字列CENTRICMANAGER (指定する場合)	−	◎	−	−	−	−	小入出力のCONSOLE指定、および実行時メッセージの出力先として、Systemwalker Centric Managerの集中コンソールを使用する場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
CBR_CSV_OVERFLOW_MESSAGE	文字列NO (指定する場合)	−	◎	−	−	−	−	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する場合 ・ JMP0262I-W

環境変数名	設定する内容	条件						
		コンパイラ	ランタイムシステム	デバッガ	プロジェクトマネージャ	S-DESIGN	ファイルユーティリティ	
								・ JMP0263I-W (“第27章 CSV形式データの操作”参照)
CBR_CSV_TYPE	文字列 MODE-1 または 文字列 MODE-2 または 文字列 MODE-3 または 文字列 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する場合(“第27章 CSV形式データの操作”参照)
CBR_DISPLAY_CONSOLE_OUTPUT	文字列SYSLOG(指定する場合)	—	◎	—	—	—	—	機能名CONSOLEに対応付けた呼び名を指定した小入出力の出力先として、シスログを使用する場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
CBR_DISPLAY_SYSOUT_OUTPUT	文字列SYSLOG(指定する場合)	—	◎	—	—	—	—	UPON指定なしまたは、機能名SYSOUTに対応付けた呼び名を指定した小入出力の出力先として、シスログを使用する場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
CBR_DISPLAY_SYSERR_OUTPUT	文字列SYSLOG(指定する場合)	—	◎	—	—	—	—	機能名SYSERRに対応付けた呼び名を指定した小入出力の出力先として、シスログを使用する場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
CBR_DISPLAY_CONSOLE_SYSLOG_LEVEL	文字列 I または 文字列 W または 文字列 E	—	◎	—	—	—	—	シスログに出力する、機能名CONSOLEに対応付けたDISPLAY文のレベルを変更したい場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
CBR_DISPLAY_SYSOUT_SYSLOG_LEVEL	文字列 I または 文字列 W または 文字列 E	—	◎	—	—	—	—	シスログに出力する、UPON指定なしまたは、機能名SYSOUTに対応付けたDISPLAY文のレベルを変更したい場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
CBR_DISPLAY_SYSERR_SYSLOG_LEVEL	文字列 I または 文字列 W または 文字列 E	—	◎	—	—	—	—	シスログに出力する、機能名SYSERRに対応付けたDISPLAY文のレベルを変更したい場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
CBR_DISPLAY_CONSOLE_SYSLOG_IDENT	アイデンティティ名(指定する場合)	—	◎	—	—	—	—	シスログに出力する、機能名CONSOLEに対応付けたDISPLAY文のアイデンティティ名を

環境変数名	設定する内容	条件						
		コンパイラ	ランタイムシステム	デバッガ	プロジェクトマネージャ	S-DESIGN	ファイルユーティリティ	
								変更したい場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
CBR_DISPLAY_SYSOUT_SYSLOG_IDENT	アイデンティティ名(指定する場合)	−	◎	−	−	−	−	シスログに出力する、UPON指定なしまたは、機能名SYSOUTに対応付けたDISPLAY文のアイデンティティ名を変更したい場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
CBR_DISPLAY_SYSERR_SYSLOG_IDENT	アイデンティティ名(指定する場合)	−	◎	−	−	−	−	シスログに出力する、機能名SYSERRに対応付けたDISPLAY文のアイデンティティ名を変更したい場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
CBR_ENTRYFILE	エントリ情報ファイル名	−	◎	−	−	−	−	動的プログラム構造で、副プログラムを呼出したい場合
CBR_EXFH_API	外部ファイルハンドラの入口名	−	◎	−	−	−	−	結合するファイルシステムの入口名を指定する場合(“6.8.11 外部ファイルハンドラ”参照)
CBR_EXFH_LOAD	外部ファイルハンドラの共用オブジェクトファイル名	−	◎	−	−	−	−	結合するファイルシステムの共用オブジェクトファイル名を指定する場合(“6.8.11 外部ファイルハンドラ”参照)
CBR_FCB_NAME	FORMAT句付き印刷ファイルのFCB省略時に使用するFCB名	−	◎	−	−	−	−	FORMAT句付き印刷ファイルでFCBの省略値を変更する場合
CBR_FILE_BOM_READ	文字列 CHECK または 文字列 DATA または 文字列 AUTO	−	◎	−	−	−	−	Unicodeの行順ファイルに付加されている認識コードの扱いを選択する場合(“6.3.3 行順ファイルの処理”参照)
CBR_FILE_LFS_ACCESS	文字列 YES	−	◎	−	−	−	−	大容量ファイルを一括して有効にする場合(“6.8.2.2 一括指定”参照)
CBR_FILE_SEQUENTIAL_ACCESS	文字列 BSAM	−	◎	−	−	−	−	ファイルの高速処理を一括して有効にする場合(“一括指定”参照)
CBR_FILE_USE_MESSAGE	文字列YES (指定する場合)	−	◎	−	−	−	−	有効な誤り処理手続きがあり、入出力エラーの実行時メッセージを出力する場合(“6.6 入出力エラー処理”参照)
CBR_FUNCTION_NATIONAL	第1オペランド: 文字列MODE1(従来(V40L20以前)どおりの変換を行う(省略時)) または MODE2(視覚的により近い変換を行う)	−	◎	−	−	−	−	NATIONAL関数の変換モードを指定する場合(“D.5 NATIONAL関数の変換モード”参照)

環境変数名	設定する内容	条件						
		コンパイラ	ランタイムシステム	デバッガ	プロジェクトマネージャ	S-DESIGN	ファイルユーティリティ	
	第2オペランド: 文字列 MODE3(UNIX系システムに近い変換を行う) または MODE4(Windows系システムに近い変換を行う)							
CBR_INPUT_BUFFERING	文字列yes	―	◎	―	―	―	―	ファイル入力の先読み処理を指定する場合(“第6章 ファイル処理”参照)
CBR_INSTANCEBLOCK	文字列use(ブロック化する場合) または unuse(ブロック化しない場合)	―	◎	―	―	―	―	オブジェクト指向プログラムでのオブジェクトインスタンスの獲得方法を指定する場合
CBR_JOBDATE	任意の日付をYY.MM.DDまたはYYYY.MM.DD(指定する場合)	―	◎	―	―	―	―	ACCEPT～FROM DATEまたは組み込み関数CURRENT-DATEで任意の日付を取得する場合(“10.1.9 任意の日付の入力”および“D.3 CURRENT-DATE関数を利用した西暦の取得”参照)
CBR_LP_OPTION	lpコマンドのオプション文字	―	◎	―	―	―	―	ファイル管理記述項のASSIGN句指定がPRINTERの場合(“第7章 印刷処理”参照)
CBR_MEMORY_CHECK	文字列 MODE1	―	◎	―	―	―	―	アプリケーション実行時、メモリチェック機能を使ってランタイムシステム領域を検査する場合(“5.5 メモリチェック機能の使い方”参照)
CBR_MESS_LEVEL_CONSOLE	文字列 NO または 文字列 I または 文字列 W または 文字列 E または 文字列 U	―	◎	―	―	―	―	表示される実行時メッセージ(CBR_MESSOUTFILEを指定している場合、ファイルに出力される実行時メッセージ)の重大度を変更したい場合(“4.3.1 実行時メッセージの重大度指定”参照)
CBR_MESS_LEVEL_SYSLOG	文字列 NO または 文字列 I または 文字列 W または 文字列 E または 文字列 U	―	◎	―	―	―	―	Syslogに出力される実行時メッセージの重大度を変更したい場合(“4.3.3 実行時メッセージのSyslog出力”参照)

環境変数名	設定する内容	条件						
		コンパイラ	ランタイムシステム	デバッガ	プロジェクトマネージャ	S-DESIGN	ファイルユーティリティ	
CBR_MESSOUTFILE	実行時メッセージを出力するファイル名	―	◎	―	―	―	―	実行時メッセージおよびSYSERRに対応付けられたDISPLAY文の結果をファイルに出力したい場合("4.3.2 実行時メッセージのファイル出力"参照)
CBR_PRINTFONTTABLE	実行単位で共通のフォントテーブル	―	◎	―	―	―	―	印刷ファイルでフォントテーブルを使用する場合("7.1.9 フォントテーブル"参照)
CBR_PRT_INF	印刷情報ファイルのパス名	―	◎	―	―	―	―	印刷情報ファイルを使用する場合("第7章 印刷処理"参照)
CBR_PRT_UTF8_CONVERT	Unicode(UTF-8)印刷未サポート機能/環境下での丸め印刷指示	―	◎	―	―	―	―	FORMAT句なし印刷ファイルにおいて、Unicode(UTF-8)印刷が未サポートである機能/環境下で印刷可能な範囲に丸めて出力する場合("第4章 プログラムの実行"および"第7章 印刷処理"参照)
CBR_PSFILE_ACM	あて先ACM指定の表示ファイルへの接続製品識別名	―	◎	―	―	―	―	表示ファイルを使用する場合("第4章 プログラムの実行"および"第19章 通信機能"参照)
CBR_PSFILE_APL	あて先APL指定の表示ファイルへの接続製品識別名	―	◎	―	―	―	―	表示ファイルを使用する場合("第4章 プログラムの実行"および"第19章 通信機能"参照)
CBR_PSFILE_DSP	あて先DSP指定の表示ファイルへの接続製品識別名	―	◎	―	―	―	―	表示ファイルを使用する場合("第4章 プログラムの実行"および"第8章 画面を使った入出力"参照)
CBR_PSFILE_PRT	あて先PRT指定の表示ファイルへの接続製品識別名	―	◎	―	―	―	―	表示ファイルを使用する場合("第4章 プログラムの実行"および"第7章 印刷処理"参照)
CBR_SCR_KEYDEFFILE	キー定義ファイルのパス名	―	◎	―	―	―	―	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行う場合("第8章 画面を使った入出力"参照)
CBR_SSIN_FILE	文字列THREAD(指定する場合)	―	◎	―	―	―	―	スレッド単位に入力ファイルをオープンする場合
CBR_SYMFOWARE_THREAD	文字列MULTI(指定する場合)	―	◎	―	―	―	―	Symfoware連携のマルチスレッドプログラムを動作可能にする場合
CBR_SYSERR_EXTEND	文字列YES(指定する場合)	―	◎	―	―	―	―	SYSERRへのDISPLAY文での出力に、プロセスID,スレッドIDの情報を付加する場合
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に、待ち時間を変更する場合
CBR_TRACE_FILE	トレース情報パス名	―	◎	―	―	―	―	TRACE機能を使用する場合("第5章 プログラムのデバッグ"参照)
CBR_TRACE_PROCESS_MODE	文字列MULTI(指定する場合)	―	◎	―	―	―	―	TRACE機能を使用する場合("第5章 プログラムのデバッグ"参照)

環境変数名	設定する内容	条件						
		コンパイラ	ランタイムシステム	デバッガ	プロジェクトマネージャ	S-DESIGN	ファイルユーティリティ	
CBR_TRAILING_BLANK_RECORD	文字列REMOVE(レコード内の後置空白を取り除く) または VALID(有効にする)	―	◎	―	―	―	―	行順ファイルのWRITE文を実行する場合("6.3 行順ファイルの使い方"参照)
COBCOPY	登録集原文格納ディレクトリ	○	―	○	―	―	―	登録集原文格納ディレクトリのデフォルトを指定したい場合("第3章 プログラムの翻訳とリンク"参照、および"第23章 対話型デバッガの使い方"参照)
COB_COPYNAME	登録集原文名の検索条件	○	―	○	○	―	―	登録集ファイルの検索条件を指定したい場合("第3章 プログラムの翻訳とリンク"参照)
COB_LIBSUFFIX	登録集ファイルの拡張子	○	―	―	―	―	―	登録集ファイルの拡張子を変更する場合
COBOLOPTS	cobolコマンドで指定するオプションの並び	○	―	―	―	―	―	cobolコマンドによる翻訳時に翻訳オプションの省略値を指定する場合("第3章 プログラムの翻訳とリンク"参照)
COB_REPIN	リポジトリファイルの入力先ディレクトリ	○	―	―	○	―	―	リポジトリファイル入力先ディレクトリのデフォルトを指定する場合("第3章 プログラムの翻訳とリンク"参照)
COLUMNS	桁数	―	○	―	―	―	―	画面の大きさを変更できる端末装置でスクリーン操作機能を使用する場合("第8章 画面を使った入出力"参照)
DTHELPUSERSEARCHPATH	CDEヘルプ用ファイルのパス名	―	―	○	○	○	○	必須
FCBDIR	FCBモジュールが格納されているディレクトリのパス名	―	◎	―	―	―	―	FCBを使用する場合("第7章 印刷処理"参照)
FFD_SUFFIX	ファイル定義体ファイルの拡張子	○	―	○	―	―	―	ffd以外を拡張子とする場合("第3章 プログラムの翻訳とリンク"および"第23章 対話型デバッガの使い方"参照)
FILELIB	ファイル定義体ファイルの格納ディレクトリ名	○	―	○	―	―	―	ファイル定義体を使用する場合("第3章 プログラムの翻訳とリンク"参照)
FORMLIB	画面帳票定義体ファイルの格納ディレクトリ名	○	―	○	―	―	―	画面帳票定義体を使用する場合("第3章 プログラムの翻訳とリンク"参照)
FOVLDIR	フォームオーバレイパターン格納ディレクトリ	―	◎	―	―	―	―	フォームオーバレイパターンとの合成印刷を行う場合("第7章 印刷処理"参照)
GOPT	実行時オプションの並び	―	◎	―	―	―	―	実行時オプションを指定する場合("4.2.2 実行時オプションを指定する"参照)
LANG	言語名	○	◎	○	○	○	○	COBOLを使用するコード系を設定する

環境変数名	設定する内容	条件						
		コンパイラ	ランタイムシステム	デバッガ	プロジェクトマネージャ	S-DESIGN	ファイルユーティリティ	
LC_ALL	言語名	○	◎	○	○	○	○	設定する必要がある場合は、LANGと同じ値を設定する
LD_LIBRARY_PATH	・ リンクされるライブラリパス ・ 実行時に動的リンクまたは動的プログラムで呼び出されるモジュールのパス	○	○	○	○	○	○	必須
LINES	行数	—	○	—	—	—	—	画面の大きさを変更できる端末装置でスクリーン操作機能を使用する場合("第8章 画面を使った入出力"参照)
MANPATH	オンラインマニュアルのパス	○	○	○	○	○	○	必須
MEFTDIR	接続製品が使用する情報ファイルが格納されているディレクトリのパス名	—	◎	—	—	—	—	MeFtを使用する場合("第7章 印刷処理"および"第8章 画面を使った入出力"参照)
MGPRM	OSIV系形式の実行時パラメタ	—	◎	—	—	—	—	OSIV系形式の実行時パラメタを指定する場合("第4章 プログラムの実行"参照)
NLSPATH	オンラインヘルプ および メッセージカタログのパス	○	○	○	○	○	○	必須
PATH	コマンド検索パス	○	○	○	○	○	○	必須
SMED_SUFFIX	画面帳票定義体ファイルの拡張子	○	—	○	—	—	—	画面帳票定義体ファイルの拡張子を変更する場合("第3章 プログラムの翻訳とリンク"および"第23章 対話型デバッガの使い方"参照)
SYSCOUNT	COUNT情報ファイルのパス名	—	◎	—	—	—	—	COUNT機能を使用する場合("第5章 プログラムのデバッグ"参照)
TMPDIR	一時的な作業ファイルの格納場所	—	○	—	○	—	○	ファイルユーティリティの復旧コマンドを実行する場合("第25章 ファイルユーティリティ"参照) FORMAT句なし印刷ファイルで、他社プリンタ印刷を行う場合、またはオーバレイ印刷を行う場合("第7章 印刷処理"参照)
XUSERFILESEARCHPATH	リソースファイルのパス名	—	—	○	○	○	○	必須
登録集名	登録集原文格納ディレクトリ	○	—	○	—	—	—	COPY文に登録集名を記述した場合("第3章 プログラムの翻訳とリンク"参照)

環境変数名	設定する内容	条件						
		コンパイラ	ランタイムシステム	デバッガ	プロジェクトマネージャ	S-DESIGN	ファイルユーティリティ	
ファイル識別名	・ファイルのパス名 ・接続製品が使用する情報ファイルのパス名 ・接続製品識別名	―	◎	―	―	―	―	ファイル処理を行う場合 表示ファイル/印刷ファイルを使用する場合(“第6章 ファイル処理”、“第7章 印刷処理”、“第8章 画面を使った入出力”および“第11章 SORT文およびMERGE文の使い方～整列併合機能～”参照)
翻訳オプションSSINに指定した名前	ファイルのパス名	―	◎	―	―	―	―	ACCEPT文の入力ファイルを翻訳オプションSSINで指定した場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)
翻訳オプションSSOUTに指定した名前	ファイルのパス名	―	◎	―	―	―	―	DISPLAY文の出力ファイルを翻訳オプションSSOUTで指定する場合(“第10章 ACCEPT文およびDISPLAY文の使い方”参照)

以下の環境変数に設定する値については/opt/FJSVcbl/configのディレクトリの下にある環境変数を設定するテンプレートを参照してください。

- PATH
- NLSPATH
- MANPATH
- XUSERFILESEARCHPATH

注意

環境変数LC_ALLを指定する場合は、環境変数LANGと同じ値を設定してください。異なる値を設定した場合には、デバッガ、プロジェクトマネージャ、S-DESIGNおよびファイルユーティリティの動作は保証されません。

付録F　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。

F.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、以下に示す文字を使用することはできません。

- コード系がEUCの場合の日本語定数で、拡張漢字、拡張非漢字、利用者定義文字
- コード系がEUCの場合の半角カナ文字(翻訳オプションKANA(JIS8)が指定されている場合だけ)

これらの文字以外は使用できます。ただし、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。

F.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次の形式で記述します。

```
"ファイル名"
```

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。

F.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の名前を次の形式で指定します。

```
"[パス名][ファイル参照名]"
```

パス名には、相対パス名または絶対パス名を指定することができます。パス名が省略された場合、ファイル参照名で示されるファイルはカレントディレクトリの中のファイルとみなされます。

F.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には、以下に示す文字を使用することができません。これらの文字以外は使用できます。ただし、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。

- 最初の文字がアンダースコア
- コード系がEUCの場合の日本語定数で、拡張漢字、拡張非漢字、利用者定義文字
- コード系がEUCの場合の半角カナ文字(翻訳オプションKANA(JIS8)が指定されている場合だけ)

付録G　COBOLが提供するサブルーチン

ここでは、COBOLが提供するサブルーチンについて説明します。COBOLが提供しているサブルーチンには次のものがあります。

G.1　Web連携

Web連携を実現するために、COBOL Webサブルーチンを提供しています。このサブルーチンを使用することにより、インターネット/イントラネットを活用した基幹業務システムを構築することができます。

Web連携およびCOBOL Webサブルーチンの詳細については"Web連携ガイド"を参照してください。

G.2　簡易アプリ間通信機能

COBOLだけでCOBOLプログラムからメッセージ通信を行うことができる簡易アプリ間通信機能を実現するためにサブルーチンを提供しています。このサブルーチンを使用することにより、COBOLプログラム間で簡単なデータの受け渡しを行うことができます。

簡易アプリ間通信機能およびサブルーチンの詳細については"19.3 簡易アプリ間通信機能"を参照してください。

G.3　システム情報を取得するサブルーチン

COBOLは次に示すシステム情報を取得するためのサブルーチンを提供しています。

- プロセスID
- スレッドID

これらのサブルーチンは、COBOLプログラムで使用することができます。ここでは、COBOLプログラムからサブルーチンを呼び出してシステム情報を取り出す方法について説明します。

動的プログラム構造で、システム情報を取得する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ファイルが必要になります。エントリ情報の指定方法については、"4.1.4 副プログラムのエントリ情報"を参照してください。

プロセスモデルの場合

```
[ENTRY]
サブルーチン名=libcobol.so
```

マルチスレッドモデルの場合

```
[ENTRY]
サブルーチン名=librcobol.so
```

G.3.1　プロセスID取得サブルーチン

サブルーチンCOB_GET_PROCESSIDを利用することによって、この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。

呼出し形式

```
CALL  "COB_GET_PROCESSID" USING  BY  REFERENCE データ名－１.
```

パラメタの説明

データ名－１

```
01  データ名－１   PIC  9(9) COMP-5.
```

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COB_GET_PROCESSIDを呼び出した場合、特殊レジスタPROGRAM-STATUSが不定な値で更新されてしまいます。特殊レジスタPROGRAM-STATUSが更新されないようにするには、ダミーのデータ項目(PIC S9(9) COMP-5)をCALL文のRETURNING指定に記述してください。

G.3.2 スレッドID取得サブルーチン

サブルーチンCOB_GET_THREADIDを利用することによって、この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－１.
```

パラメタの説明

データ名－１

```
01  データ名－１   PIC  9(9) COMP-5.
```

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COB_GET_THREADIDを呼び出した場合、特殊レジスタPROGRAM-STATUSが不定な値で更新されてしまいます。特殊レジスタPROGRAM-STATUSが更新されないようにするには、ダミーのデータ項目(PIC S9(9) COMP-5)をCALL文のRETURNING指定に記述してください。

G.4 他言語連携で使用するサブルーチン

ここでは、他言語連携用にCOBOLが提供するサブルーチンについて説明します。

G.4.1 実行単位の開始サブルーチン

他言語のプログラムから複数のCOBOLプログラムを呼び出す場合で、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

呼出しの記述(C言語での呼び出し方)

型宣言部

```
extern void JMPCINT2(void);
```

手続き部

```
JMPCINT2();
```

インタフェース

呼出し時にパラメタは必要ありません。

復帰値

サブルーチンからの復帰値はありません。

実行単位の開始サブルーチン使用時の注意事項

- この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、"9.1.1 COBOLの言語間の環境"および"17.3.1 実行環境と実行単位"を参照してください。

例

```
extern void JMPCINT2(void);
extern void JMPCINT3(void);
extern int  COBSUB1(void);          ← COBOL プログラム
extern int  COBSUB2(void);          ← COBOL プログラム

int csub(void) {
    JMPCINT2();                     ← 実行単位の開始
    COBSUB1();                      ← COBOL プログラム
    COBSUB2();                      ← COBOL プログラム
    JMPCINT3();                     ← 実行単位の閉鎖
}
```

G.4.2 実行単位の終了サブルーチン

実行単位の開始サブルーチンを使用した際に、実行単位を閉鎖させる場合に他言語のプログラムから使用します。

呼出し形式

呼出しの記述(C言語での呼び出し方)

型宣言部

```
extern void JMPCINT3(void);
```

手続き部

```
JMPCINT3();
```

インタフェース

呼出し時にパラメタは必要ありません。

復帰値

サブルーチンからの復帰値はありません。

実行単位の終了サブルーチン使用時の注意事項

- この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、“9.1.1 COBOLの言語間の環境”および“17.3.1 実行環境と実行単位”を参照してください。

例

```
extern void JMPCINT2(void);
extern void JMPCINT3(void);
extern int  COBSUB1(void);                  ← COBOL プログラム
extern int  COBSUB2(void);                  ← COBOL プログラム

int csub(void) {
    JMPCINT2();                             ← 実行単位の開始
    COBSUB1();                              ← COBOL プログラム
    COBSUB2();                              ← COBOL プログラム
    JMPCINT3();                             ← 実行単位の閉鎖
}
```

G.4.3 実行環境の閉鎖サブルーチン

プロセス内のすべての実行単位が終了した状態で、他言語プログラムからJMPCINT4を呼び出すことにより、実行環境を閉鎖することができます。

呼出し形式

呼出しの記述(C言語での呼び出し方)

型宣言部

```
extern void JMPCINT4(void);
```

手続き部

```
JMPCINT4();
```

インタフェース

呼出し時にパラメタは必要ありません。

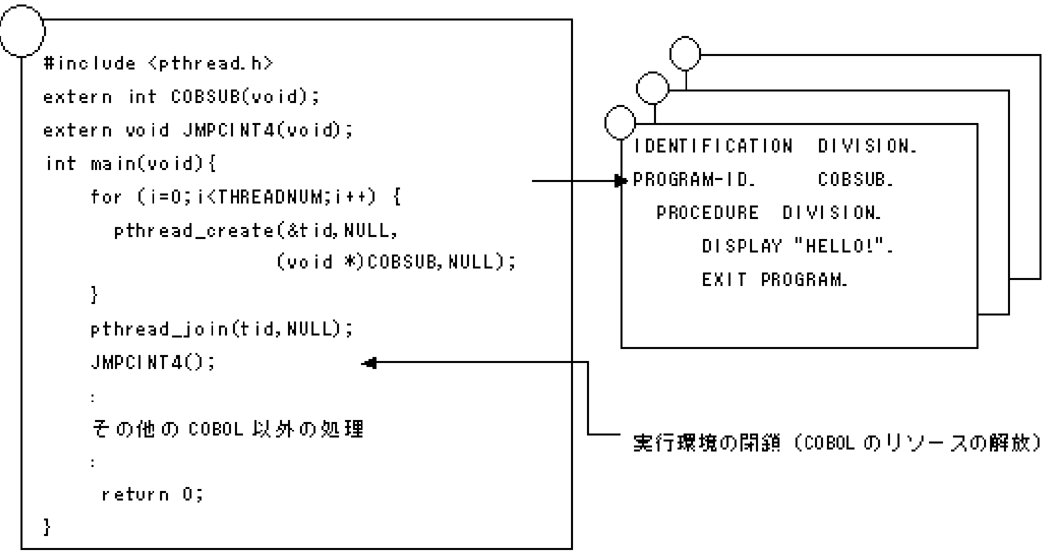
復帰値

サブルーチンからの復帰値はありません。

実行環境の閉鎖サブルーチン使用時の注意事項

- この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この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、このサブルーチンを呼び出す前に、JMPCINT2サブルーチンを呼び出して実行単位の開始を行っている場合は、必ずJMPCINT3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、"9.1.1 COBOLの言語間の環境"および"17.3.1 実行環境と実行単位"を参照してください。
- プロセスモデルのプログラムでは、実行単位の終了時に実行環境も閉鎖します。このため、プロセスモデルの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



G.5 日本語内部表現のコードセット(COBOL独自の16ビットワイドキャラクタ)操作

COBOLで使用する日本語内部表現のコードセット(COBOL独自の16ビットワイドキャラクタ)と多バイトEUCコードセット表現との相互参照を行う、次に示すサブルーチン(C関数)を提供しています。

関数名	ライブラリ名	機能概要
mbston16s	・プロセスモデルのプログラムから呼び出す場合 libQWCCMTON.so ・マルチスレッドプログラムから呼び出す場合 librQWCCMTON.so	EUC → COBOL16ビットワイドキャラクタ
n16stombs	・プロセスモデルのプログラムから呼び出す場合 libQWCCNTOM.so ・マルチスレッドプログラムから呼び出す場合 librQWCCNTOM.so	COBOL16 ビットワイドキャラクタ → EUC

これらのサブルーチンは、COBOLおよびCプログラムで使用することができます。ここでは、各サブルーチンの使い方について説明します。

注意

- このサブルーチンは、文字コード系が日本語EUCの場合にだけ使用できます。
- 日本語EUC以外の文字コード系では、使用する意味がなく、サブルーチンの結果は不定になります。

G.5.1 mbston16s

EUCからCOBOL16ビットワイドキャラクタへの変換を行います。

COBOLでの使用方法

呼出し形式

文字列を変換する場合

```
CALL "mbston16s" USING データ名－１ データ名－２ BY VALUE データ名－３.
```

変換した文字列を格納するための領域の長さを得る場合

```
CALL "mbston16s" USING BY VALUE データ名－４ BY REFERENCE データ名－２
                                             BY VALUE     データ名－３.
```

パラメタの説明

データ名－1(変換した文字列)

変換結果(COBOL16ビットワイドキャラクタ文字)の格納先のデータ名(日本語項目)を指定します。

データ名－2(変換する文字列)

変換する文字列(EUC)を格納したデータ名(英数字項目)を指定します。

データ名－3(領域長)

変換結果を格納する領域(データ名－1)の文字数を設定したデータ名(5～9桁の符号なし2進項目)を指定します。

データ名－4

値に0を設定したデータ名(5～9桁の符号なし2進項目)を指定します。

例

```
01 A PIC 9(5) COMP-5 VALUE 0.
```

機能概要

- データ名－2に設定されたEUC文字を取り出し、COBOL16ビットワイドキャラクタ文字に変換し、その結果をデータ名－1に格納します。この処理を繰り返し、データ名－3に設定されている文字数分格納したら処理を終了します。
  ただし、データ名－2で指定した領域に1バイトのLOW-VALUEが出現した場合、残った部分に空白文字(X"A1A1")を格納し、処理を終了します。
  また、変換した文字列がデータ名－3に設定された値を超える場合には、その値まで変換した文字を格納します。
- データ名－4に0を設定し、パラメタを値渡し(BY VALUE指定)した場合、文字列の変換は行われず、変換結果を格納するために必要な領域の長さを返します。

復帰値

正常

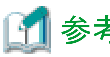
変換したバイト数、または変換結果を格納するために必要なバイト数を返します。

異常

以下のどちらかの場合、-1を返します。

- データ名－2に変換すべきデータがない場合
- 変換できないコードがある場合

ASCII文字および半角カナ文字は変換できません。

 参考

復帰値は、特殊レジスタPROGRAM-STATUSに設定されます。

注意

呼出し名が英小文字のため、プログラム翻訳時には、翻訳オプションNOALPHALを必ず指定してください。

C言語での使用方法

呼出し形式

```
#include <n16.h>
 size_t mbston16s(n16_t *n16s,const char *s,size_t n);
```

引数

n16s

COBOL16ビットワイドキャラクタ文字の格納先

s

EUC文字の格納先

n

COBOL16ビットワイドキャラクタ文字の格納先(n16s)の配列数

機能概要

引数sで指定された配列からEUC文字を取り出し、COBOL16ビットワイドキャラクタに変換し、その結果を引数n16sで指定された配列へ格納します。

繰り返し配列に引数をn個格納したら処理を終了します。
格納先の配列の最後にはナル文字は出力されません。
引数sで指定された配列の途中にナル文字が出現した場合は、残った部分に空白文字(JIS X 0208 1区1点)を格納し、処理を終了します。
引数n16sにNULLポインタが指定された場合、変換は行わず変換結果を格納するために必要なバッファサイズを返します。

復帰値

正常

変換したバイト数(パディングされる空白文字は含まない)を返します。

異常

errnoを設定し、(size_t)-1を返します。

エラー番号

EILSEQ

sで指定された入力データに有効でない値があります。
ASCII文字および半角カナ文字は変換できません。

EINVAL

nが0または入力データを格納する領域を指すポインタsがNULLポインタです。

関連事項

G.5.2 n16stombs

G.5.2 n16stombs

COBOL16ビットワイドキャラクタからEUCへの変換を行います。

COBOLでの使用方法

呼出し形式

文字列を変換する場合

```
CALL "n16stombs" USING  データ名－１ データ名－２
BY VALUE データ名－３ データ名－４.
```

変換した文字列を格納するための領域の長さを得る場合

```
CALL "n16stombs" USING BY VALUE データ名－５ BY REFERENCE データ名－２
                                BY VALUE データ名－３  データ名－４.
```

パラメタの説明

データ名－1(変換後の文字列)

変換後の文字列(EUC)を格納する領域のデータ名(英数字項目)を指定します。
なお、用意する領域の大きさは、以下の計算式で求めることができます。

```
用意する領域の大きさ ＝ （変換元の文字列の文字数 × 3） ＋ 1(注)
```

注:変換後の文字列の末尾に、1バイトのLOW-VALUEが付加されます。

データ名－2(変換元の文字列)

変換元の文字列(COBOL16ビットワイドキャラクタ文字)を格納したデータ名(日本語項目)を指定します。

データ名－3(領域長)

変換後の文字列を格納する領域(データ名－1)の長さ(バイト数)を設定したデータ名(5～9桁の符号なし2進項目)を指定します。領域長が足りない場合、超過した文字列は格納されませんので、十分な領域長が必要です。領域長の最大長は、以下の計算式で求めることができます。

```
最大長 ＝ （変換元の文字列の文字数 × 3） ＋ 1(注)
```

注:変換後の文字列の末尾に、1バイトのLOW-VALUEが付加されます。

データ名－4(変換文字数)

変換する文字数を設定したデータ名(5～9桁の符号なし2進項目)を指定します。

データ名－5

値に0を設定したデータ名(5～9桁の符号なし2進項目)を指定します。

例

```
01 A PIC 9(5) COMP-5 VALUE 0.
```

機能概要

- データ名－2に設定されたCOBOL16ビットワイドキャラクタ文字を取り出し、EUC文字に変換し、その結果をデータ名－1に格納します。この処理を繰り返し、以下の条件を満たしたら処理を終了します。
  - データ名－3に設定されている領域長分格納した
  - データ名－4に設定されている文字数分変換した

  ただし、データ名－2で指定した領域に2バイトのLOW-VALUEが出現した場合、処理を終了します。
- 変換した文字列の長さがデータ名－3に設定された領域長を超える場合には、正しく格納できる文字までを格納します。
- データ名－5に0を設定し、パラメタを値渡し(BY VALUE指定)した場合、文字列の変換は行われず、変換結果を格納するために必要な領域の長さを返します。

復帰値

正常

文字列を変換する場合

変換したバイト数を返します。

変換した文字列を格納するための領域の長さを得る場合

変換結果を格納するために必要なバイト数＋1(付加するLOW-VALUE 1バイト分)を返します。

異常

変換できないコードがある場合、-1を返します。
ASCII文字および半角カナ文字は変換できません。

注意

呼出し名が英小文字のため、プログラム翻訳時には、翻訳オプションNOALPHALを必ず指定してください。

C言語での使用方法

呼出し形式

```
#include <n16.h>
 size_t n16stombs(char *s,const n16_t *n16s,size_t n,size_t n16n);
```

引数

s

EUC文字の格納先

n16s

COBOL16ビットワイドキャラクタ文字の格納先

n

書込みバイト数

n16n

COBOL16ビットワイドキャラクタの文字数

機能概要

引数n16sで指定された配列からCOBOL16ビットワイドキャラクタの文字列をEUC文字に変換し、その結果を引数sで指定された配列へ格納します。この処理は、引数n16nで指定された個数の変換が終了するか、引数nバイトのEUC文字がsに格納されるまで繰り返し行われます。
変換した文字がnで指定した領域長を超える場合、正しく格納できる文字データまでを格納します。
sに格納された文字列がn個にみたない場合は、最後にナル文字“0”を付加します。
引数sにNULLポインタが指定されると、変換は行われず変換結果を格納するために必要なバッファサイズを返します。このとき、最後に付加されるナル文字“0”の文を含みます。

復帰値

正常

文字列を変換する場合

変換したバイト数(配列の最後のナル文字は含まない)を返します。

変換した文字列を格納するための領域の長さを得る場合

変換したバイト数＋1(配列の最後のナル文字)を返します。

異常

errnoを設定し、(size_t)-1を返します。

エラー番号

EILSEQ

n16sで指定された入力データに有効でない値があります。
ASCII文字および半角カナ文字は変換できません。

EINVAL

nまたはn16nが0です。または、入力データを格納する領域を指すポインタn16sがNULLポインタです。

関連事項

G.5.1 mbston16s

G.6 ファイルアクセスルーチン

C言語からCOBOLの各編成のファイルへアクセスするために、API(Application Program Interface)関数群を提供しています。COBOLファイルアクセスルーチンを使用することにより、C言語から以下のようなことが実現できます。

- 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形式の各編成のファイルの創成
- COBOLアプリケーションとのファイルの共用/排他
- 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

ファイルアクセスルーチンの詳細については、製品に格納されているREADMEまたは“COBOL ファイルアクセスルーチン 使用手引書”を参照してください。

G.7 メモリ割当てサブルーチン

COBOLでは、動的にメモリを割り当てる/解放する、以下のサブルーチンを提供しています。

サブルーチン名	機能
COB_ALLOC_MEMORY	動的にメモリを割り当てる。
COB_FREE_MEMORY	動的に割り当てられたメモリを解放する。

COB_ALLOC_MEMORYは、COBOLプログラムから動的にメモリの割当てを行うサブルーチンです。

このルーチンを使用して割り当てたメモリは、COB_FREE_MEMORYを呼び出すことで解放することができます。

COB_FREE_MEMORYを呼び出さない場合、メモリの解放のタイミングは以下のようになります。

プログラムモデル	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
プロセスモデル	ー	実行環境の閉鎖時
マルチスレッドモデル	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、プロセスモデルのプログラムでは意味を持ちません。マルチスレッドモデルとプロセスモデルについては、"17.2.2 マルチスレッドモデルとプロセスモデル"を参照してください。

プロセスモデルのプログラム

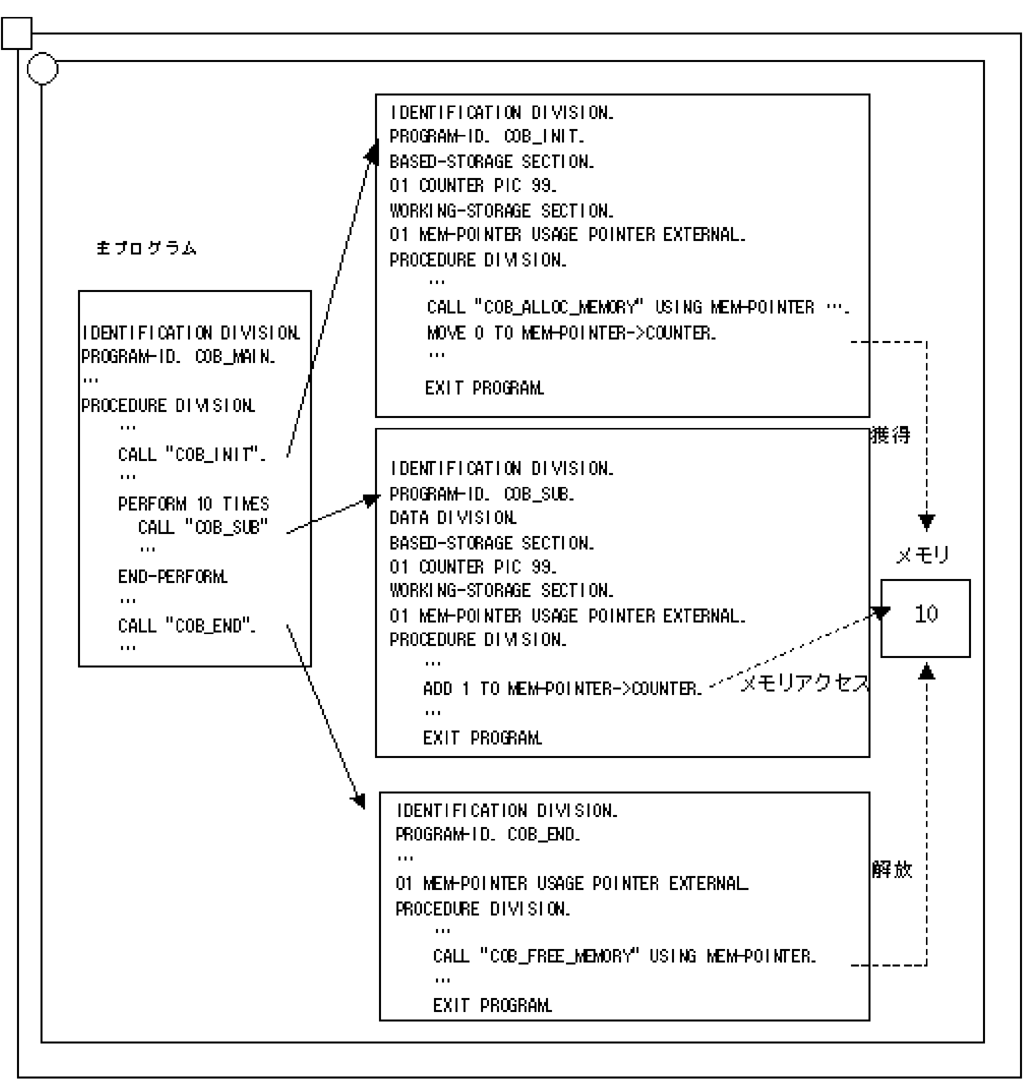
プロセスモデルのプログラムで、主プログラムがCOBOLの場合と他言語の場合について説明します。

主プログラムがCOBOLの場合

プログラムCOB_INITで、サブルーチンCOB_ALLOC_MEMORYを呼び出してメモリを割り当てます。このとき、メモリの種別として「プロセス指定」と「スレッド指定」のどちらを選択しても動作に違いはありません。

割り当てたメモリは、同一実行単位上に存在するプログラムCOB_SUBからアクセスできます。

図G.1 プロセスモデルにおけるメモリ割当て(主プログラムがCOBOL)



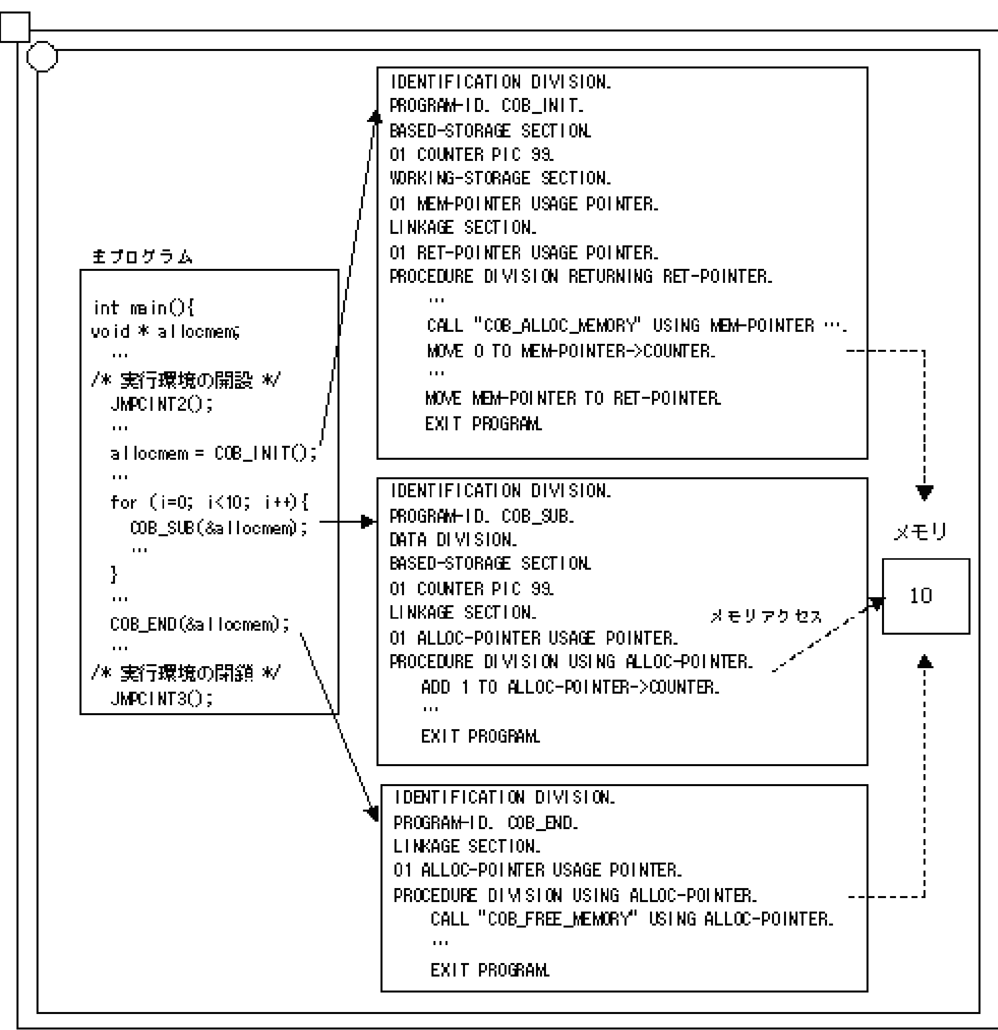
主プログラムが他言語の場合

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G.1 プロセスモデル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、"G.4 他言語連携で使用するサブルーチン"を参照してください。

図G.2 プロセスモデルにおけるメモリ割当て(主プログラムが他言語)



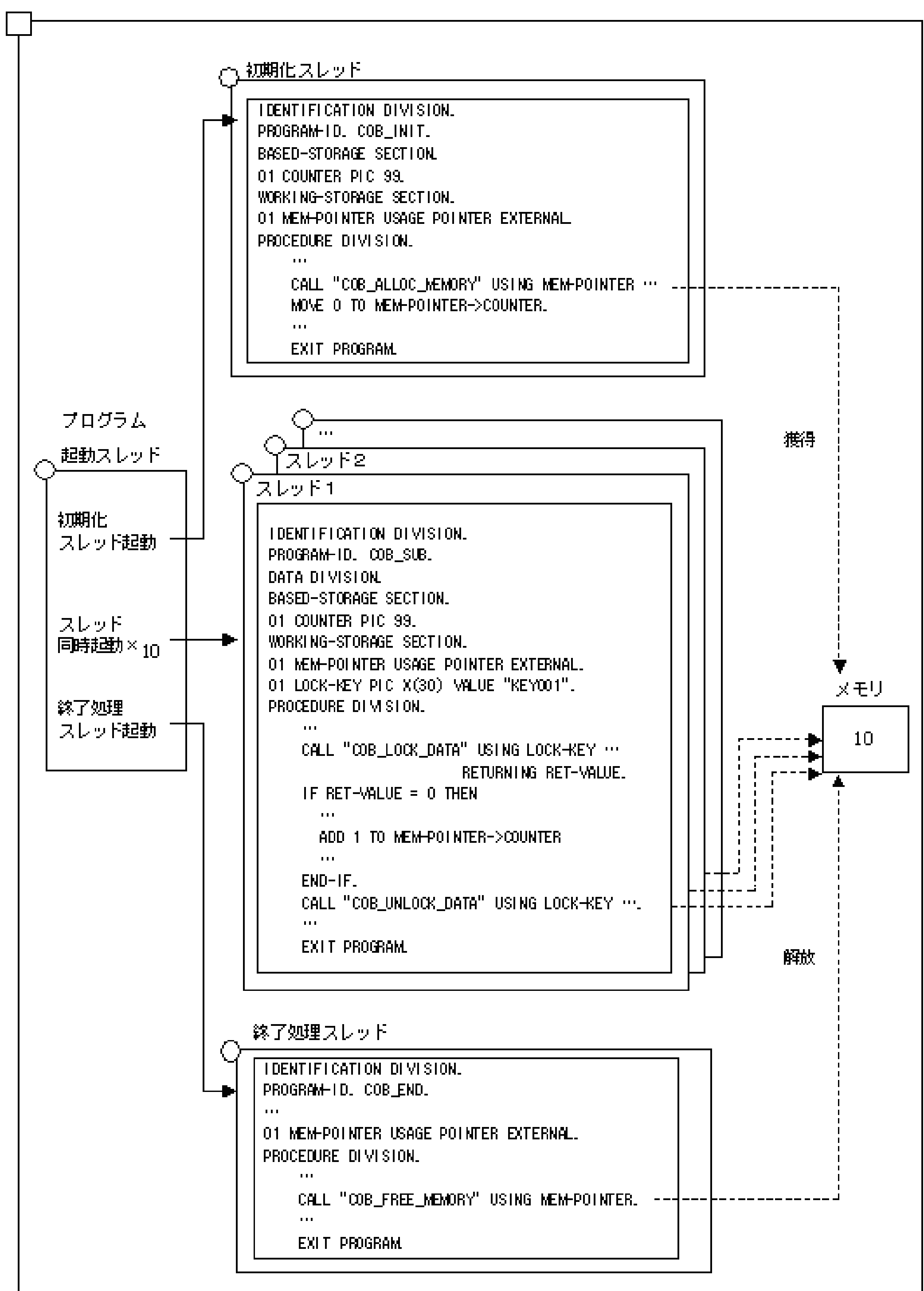
マルチスレッドモデルのプログラム

マルチスレッドモデルのプログラムでは、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、"17.4 スレッド間の資源の共有"を参考にしてください。

図G.3 プロセス指定(マルチスレッドモデル)



スレッド指定

スレッドAのプログラムCOB_INIT、スレッドBのプログラムCOB_Bにおいて、それぞれCOB_ALLOC_MEMORYを「スレッド指定」で呼び出してメモリを割り当てます。

スレッドの主プログラムがCOBOLの場合には主プログラムの終了時、他言語の場合にはJMPCINT3の呼出し時にそれぞれ解放処理が行われます。「スレッド指定」は、メモリの使用範囲がスレッド内に閉じている場合に使用してください。

図G.4 スレッド指定(マルチスレッドモデル)

プログラム
起動スレッド

スレッド
起動

スレッドA

他言語プログラム

```
void main(){
/*実行単位開始*/
  JMPCINT2();
  COB_INIT();
  ...
  COB_SUB();
  ...
  COB_END();
  ...
/*実行単位終了*/
  JMPCINT3();
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COB_INIT.
...
01 MEM-POINTER USAGE POINTER EXTERNAL.
PROCEDURE DIVISION.
    ...
    CALL "COB_ALLOC_MEMORY"
             USING MEM-POINTER ….
    ...
    EXIT PROGRAM.
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COB_SUB.
DATA DIVISION.
BASED-STORAGE SECTION.
01 DATA01 PIC X(999999).
WORKING-STORAGE SECTION.
01 MEM-POINTER USAGE POINTER EXTERNAL.
PROCEDURE DIVISION.
    ...
    MOVE ALL "A" TO MEM-POINTER->DATA01.
    ...
    EXIT PROGRAM.
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COB_END.
...
01 MEM-POINTER USAGE POINTER EXTERNAL.
PROCEDURE DIVISION.
    ...
    CALL "COB_FREE_MEMORY"
            USING MEM-POINTER.
    ...
    EXIT PROGRAM.
```

獲得

メモリアクセス

解放

```
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
```

スレッドB

COBOLプログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COB_B.
DATA DIVISION.
BASED-STORAGE SECTION.
01 DATA01 PIC X(999999).
WORKING-STORAGE SECTION.
01 MEM-POINTER USAGE POINTER.
PROCEDURE DIVISION.
    ...
    CALL "COB_ALLOC_MEMORY" USING MEM-POINTER ….
    ...
    MOVE ALL "B" TO MEM-POINTER->DATA01.
    ...
    CALL "COB_FREE_MEMORY" USING MEM-POINTER.
    ...
    EXIT PROGRAM.
```

メモリ

```
BBBBBBBBB
BBBBBBBBB
BBBBBBBBB
BBBBBBBBB
BBBBBBBBB
```

獲得

メモリアクセス

解放

注意

動的プログラム構造で、COB_ALLOC_MEMORYおよびCOB_FREE_MEMORYを呼び出す場合、以下のようなエントリ情報ファイルが必要になります。エントリ情報の指定方法については、"4.1.4 副プログラムのエントリ情報"を参照してください。

プロセスモデルの場合

```
[ENTRY]
サブルーチン名=libcobol.so
```

マルチスレッドモデルの場合

```
[ENTRY]
サブルーチン名=librcobol.so
```

G.7.1 COB_ALLOC_MEMORY

サブルーチンCOB_ALLOC_MEMORYを利用することによって、動的にメモリを割り当てることができます。

なお、割り当てられたメモリ域は初期化されません。

呼出し形式

```
CALL "COB_ALLOC_MEMORY" USING BY REFERENCE データ名－１
BY VALUE データ名－２
BY VALUE データ名－３
            RETURNING データ名－４ パラメタの説明
```

データ名－１

```
01 データ名－１ USAGE POINTER.
```

サブルーチンによって割り当てられるメモリのアドレス格納域を指定します。

データ名－２

```
01 データ名－２ PIC 9(9) COMP-5.
```

割り当てるメモリのバイト数を指定します。

データ名－３

```
01 データ名－３ PIC 9(9) COMP-5.
```

メモリの種別を以下の値から指定します。

値	意味	
0	プロセス指定	メモリをプロセス単位に割り当てます。解放ルーチンが呼び出されない場合、割り当てたメモリは実行環境の閉鎖時に解放されます。
1	スレッド指定	メモリをスレッド単位に割り当てます。解放ルーチンが呼び出されない場合、割り当てたメモリは実行単位の終了時に解放されます。

復帰値

データ名－４

```
01 データ名－４ PIC S9(9) COMP-5.
```

成功した時は0が返されます。失敗した場合の復帰値とエラーの原因は以下のとおりです。

値	意味
-1	パラメタの指定に誤りがあります。
-2	メモリが不足しています。
-3	実行単位が既に終了しています。

G.7.2 COB_FREE_MEMORY

サブルーチンCOB_FREE_MEMORYを利用することによって、動的に割り当てたメモリを解放することができます。

呼出し形式

```
CALL "COB_FREE_MEMORY" USING BY REFERENCE データ名－１
         RETURNING データ名－２
```

パラメタの説明

データ名－1

```
01 データ名－１ USAGE POINTER.
```

サブルーチンCOB_ALLOC_MEMORYによって割り当てられたメモリのアドレスを指定します。

復帰値

データ名－2

```
01 データ名－２ PIC S9(9) COMP-5.
```

成功した時は0、失敗した時は-1が返されます。
失敗した場合、エラーの原因として以下が考えられます。

- 指定したメモリが既に解放されている、または破壊されている。
- サブルーチンCOB_ALLOC_MEMORYを使用して割り当てたメモリでない。

注意

スレッド指定で割り当てたメモリを解放する場合、必ずメモリを割り当てた実行単位内で解放処理を行うようにしてください。COB_ALLOC_MEMORYを呼び出した実行単位とCOB_FREE_MEMORYを呼び出した実行単位が異なる場合、メモリの解放処理は失敗します。

G.8 プロセス終了サブルーチン

COBOLでは、プロセスを強制的に終了させるサブルーチンを提供しています。

プロセスの終了方法には、以下の2つの方法があります。

プロセスを正常に終了する

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。

SIGABRTシグナルを発行し、異常終了する

abort関数(SIGABRTシグナル)を発行しプロセスを異常終了します。
coreファイルの解析などを行いたい場合に使用することができます。

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、他のスレッドで操作中のファイルがクローズされない等の問題が発生する場合があります。

G.8.1 COB_EXIT_PROCESS

サブルーチンCOB_EXIT_PROCESSを利用することによって、プロセスを強制的に終了させることができます。

呼出し形式

```
CALL "COB_EXIT_PROCESS" USING BY VALUE データ名－１
                              BY VALUE データ名－２
          RETURNING データ名－３
```

パラメタの説明

データ名－１

```
01 データ名－１ PIC 9(9) COMP-5.
```

プロセスの終了方法を以下の値から指定します。

値	意味
0	プロセスを正常に終了させ、データ名－2に指定した値を親プロセスに返します。
1	Abort関数(SIGABRTシグナル)を発行し、プロセスを異常終了します。

データ名－２

```
01 データ名－２ PIC 9(9) COMP-5.
```

データ名－1に0を指定した場合、親プロセスに返却する値を指定します。データ名－1に1を指定した場合は、無効になります。
返却できる値の範囲は、0から255です。これを超える値を入力した場合は、下位1バイトのみを有効にします。

復帰値

データ名－３

```
01 データ名－３ PIC S9(9) COMP-5.
```

成功した時は値を返却しません。パラメタの指定に誤りがあった場合、-1を返却します。

付録H　オブジェクト指向と従来機能の組合せ

COBOLのオブジェクト指向では、従来のCOBOLで使用していた機能であっても、クラス定義、メソッド定義の中では、使用できないものがあります。

ここでは、クラス定義およびPROTOTYPE宣言により分離されたメソッド定義で使用できない機能について説明します。

H.1　クラス定義で使用できない機能

下表に、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示します。

表H.1 クラス定義で使用できない機能

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 ・ 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の構成節入出力管理段落には、APPLY MULTICONVERSATION-MODE句およびAPPLY SAVED-AREA句を指定できません。
報告書作成機能	ファクトリ定義、オブジェクト定義およびメソッド定義のデータ部には、報告書節を書くことはできません。
スクリーン操作機能	ファクトリ定義、オブジェクト定義およびメソッド定義では、スクリーン操作機能を使用できません。
CHARACTER TYPE句	ファクトリ定義およびオブジェクト定義のデータ部では、CHARACTER TYPE句を指定できません。ただし、メソッド定義のデータ部では指定できます。 また、メソッド原型定義の連絡節には、CHARACTER TYPE句を指定できません。
EXTERNAL句	ファクトリ定義およびオブジェクト定義のデータ部では、EXTERNAL句を指定できません。ただし、メソッド定義のデータ部では指定できます。
GLOBAL句	ファクトリ定義、オブジェクト定義およびメソッド定義のデータ部には、GLOBAL句を指定できません。 ファクトリ定義およびオブジェクト定義のデータ部で宣言された名前は、すべて大域名として扱われます。
LINAGE句	ファクトリ定義およびオブジェクト定義で定義したファイルには、LINAGE句を指定できません。 メソッド定義で定義したファイルには指定できます。ただし、EXTERNAL句を指定したファイルには指定できません。

機能	説明
PRINTING POSITION句	メソッド原型定義の連絡節には、PRINTING POSITION句を指定できません。
特殊レジスタ 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。

H.2 分離されたメソッド定義で使用できない機能

下表に、PROTOTYPE宣言により分離されたメソッド定義中で使用できない機能を示します。ここでは、分離された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。

下表に示した機能のうち、メソッドに関する記述については、分離されたメソッド定義の場合にも該当されます。

表H.2 分離されたメソッド定義で使用できない機能

機能	説明
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。
特殊名段落	分離された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	分離された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。

付録I　データベース連携

ここでは、COBOLとデータベース連携する際に注意すべき点について説明します。

I.1　機能概要

この製品では、埋込みSQL文を含むCOBOLプログラム(プリコンパイラの入力となったCOBOLプログラム)に対して翻訳エラーメッセージを出力したり、デバッグしたりする機能を用意しています。これにより、データベース連携を行うプログラムを効率よく開発できます。

このようなCOBOLと連携ができるデータベースを以下に示します。

- Oracle
- Symfoware

I.1.1　Oracle連携

Oracle連携時には、COBOLと連携するために専用のツール(insdbinf)が用意されています。insdbinfは、COBOLプログラムに行番号情報を埋め込みます。行番号情報の埋め込まれたソースをCOBOLで翻訳することで、SQL文を含むCOBOLプログラムに対して、エラーメッセージ出力やデバッグの連携が可能となります。

insdbinf（行番号情報埋込みツール）の使用方法

名称

insdbinf

機能

OracleのPro*COBOLが出力するファイルにCOBOL用行番号情報を挿入します

構文

```
insdbinf [-I include-path] ... [-S search-rule] -f source-file [RDB-generated-file] [-¥?]
```

オプション

-I include-path

includeファイルの検索パス名を指定します。
Pro*COBOLが処理するincludeファイルと同じ検索パス名を指定します。検索する必要があるパス名はすべて指定してください。
省略時は、カレントディレクトリが検索されます。

例

```
-I . -I ../inc -I /usr/include
```

-S search-rule

includeファイル検索順序を指定します。
Pro*COBOLが処理するincludeファイルの拡張子を検索する順に指定します。
省略時は、拡張子なしが検索されます。

例

拡張子なし、.pco,.cobの順に検索する場合

```
-S //pco/cob/
```

includeファイル名は、COBOLソースプログラムのinclude文に記述されている名称を使用します。

-f source-file

Pro*COBOLの入力ファイルを指定します。

RDB-generated-file

Pro*COBOLの出力ファイルを指定します。省略した場合は、標準入力から読み込みます。

-¥?

簡単に使用方法が表示されます。実行はされません。

oratoolの使用方法

名称

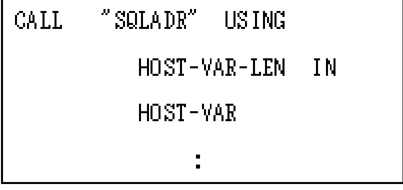
oratool

機能

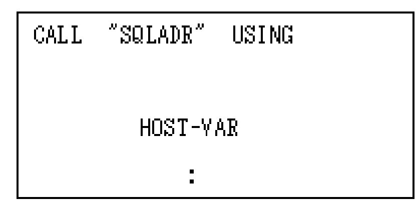
CALL文に記述されている可変長文字列のホスト変数を、そのホスト変数が含まれる集団項目に変更します。このときCALL文そのものの動作には変更ありません。

処理概要

COBOLプログラムの置換えを行い、別のCOBOLプログラムとして結果を出力します。置換えの対象は、CALL文に現れるホスト変数だけであるため、埋込みSQL文から展開されたCALL文中の“ホスト変数-LEN IN”だけを空白に置き換えます。



→



使用方法

```
oratool [-I インクルードファイル格納ディレクトリ]...
         COBOLプログラム1 COBOLプログラム2 > COBOLプログラム3
```

パラメタの意味

インクルードファイル格納ディレクトリ(入力)

インクルードファイルが格納されているディレクトリ

COBOLプログラム1(入力)

Pro*COBOLに入力したCOBOLプログラムのファイル名

COBOLプログラム2(入力)

Pro*COBOLが展開したCOBOLプログラムのファイル名

COBOLプログラム3(出力)

置き換えた後のCOBOLプログラムのファイル名

格納ディレクトリ

oratoolは、COBOLコンパイラと同じディレクトリに格納されています。シェルスクリプトで記述されているので、データベース連携時に不都合が発生した場合など、必要に応じて修正して使用してください。

制限事項

oratoolを使用する場合は、予約語は英大文字または英小文字のどちらかで統一されている必要があります。

```
正しい記述 ： CALL
正しい記述 ： call
誤った記述 ： Call
```

“Call”のように英大文字と英小文字が混在している予約語を使用する場合は、oratool内に定義されている予約語に“Call”の文字を追加する必要があります。

また、文字定数内に置き換えの対象となる文字列(CALL ホスト変数-LEN IN)が存在する場合、文字定数内も置き換えるので注意してください。

Oracle連携時の注意事項

OracleのPro*COBOLがリリース9.2.0より前の版の場合、Pro*COBOLが出力するCOBOLプログラムにおいて、集団項目名を集団の先頭にある基本項目名に置き換えてCALL文が生成されることがあります。また、COBOLでは翻訳オプションOPTIMIZE(デフォルト値は、OPTIMIZE)により広域最適化を行っており、この最適化には、参照されないデータ項目を削除する処理も含まれています。このため、呼び出す側のCOBOLプログラム中で集団項目に含まれる2番目以降の基本項目を、Pro*COBOLの提供サブルーチン内で参照していない場合には、その基本項目への転記などが削除されます。このとき、呼び出されたCOBOLプログラムでは、集団項目に含まれる2番目以降の基本項目への代入が存在するため、広域最適化を無効にしないと実行時に誤動作する可能性があります。

そのため、COBOLではCALL文に記述されている可変長文字列のホスト変数を、そのホスト変数が含まれている集団項目に置き換えるツール(oratool)を提供しています。このツールを使用すると、COBOLの翻訳オプションOPTIMIZEを指定しても、実行時に誤動作することはありません。使用方法については“oratoolの使用方法”を参照してください。

[Pro*COBOLが出力したプログラム]

```
01  HOST-VAR.
    02  HOST-VAR-LEN     PIC ～.
    02  HOST-VAR-DATA    PIC ～.
              :
 MOVE "ABCD" TO HOST-VAR-DATA.
              :
 CALL   "SQLADR"  USING
          HOST-VAR-LEN  IN
          HOST-VAR
```

→

[COBOLの翻訳結果]

```
01  HOST-VAR.
    02  HOST-VAR-LEN     PIC ～.
    02  HOST-VAR-DATA    PIC ～.
              :
(HOST-VAR-DATA への代入文を削除)
              :
 CALL  "SQLADR"  USING
          HOST-VAR-LEN  IN
          HOST-VAR
```

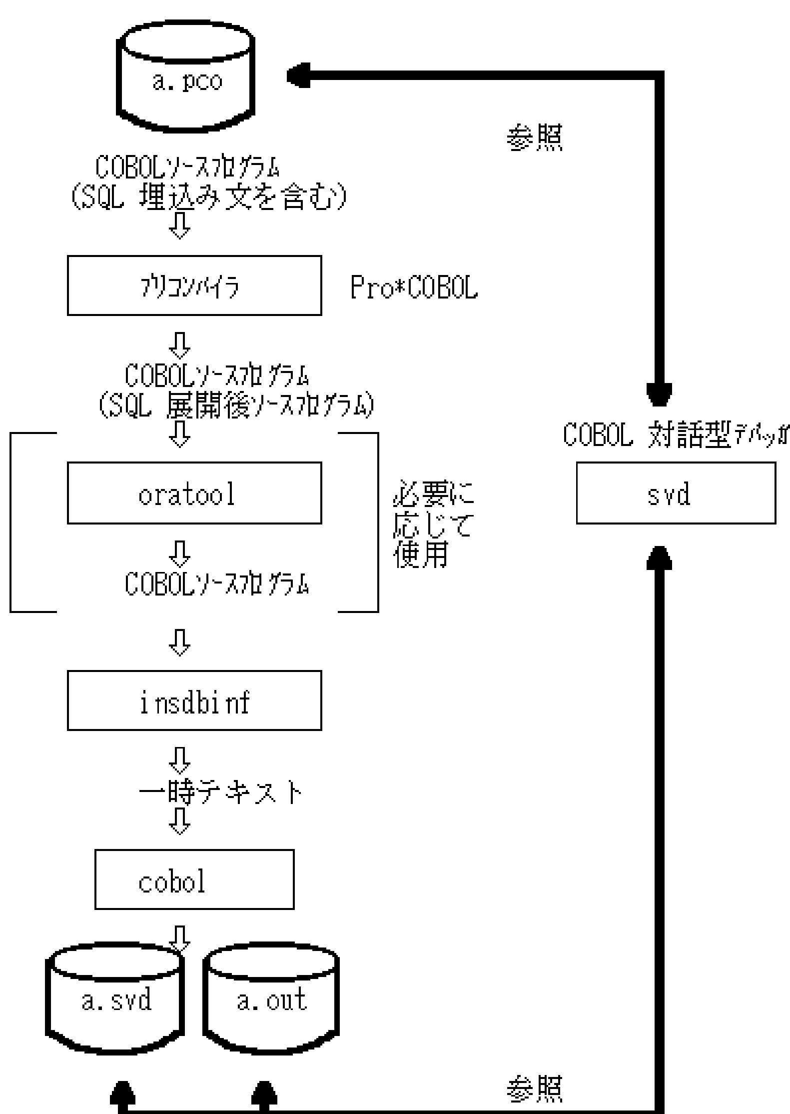
I.1.2 Symfoware連携

Symfoware/RDBとCOBOLが連携する場合、Symfoware/RDBのプリコンパイラ(Esql-COBOL)にCOBOLと連携するためのオプションが直接用意されています。そのためOracle連携時のようなCOBOLと連携するためツール(insdbinf)などを使用する必要はありません。COBOLと連携するためのSymfoware/RDBのオプション詳細については、“Symfoware Server RDB ユーザーズガイド”を参照してください。

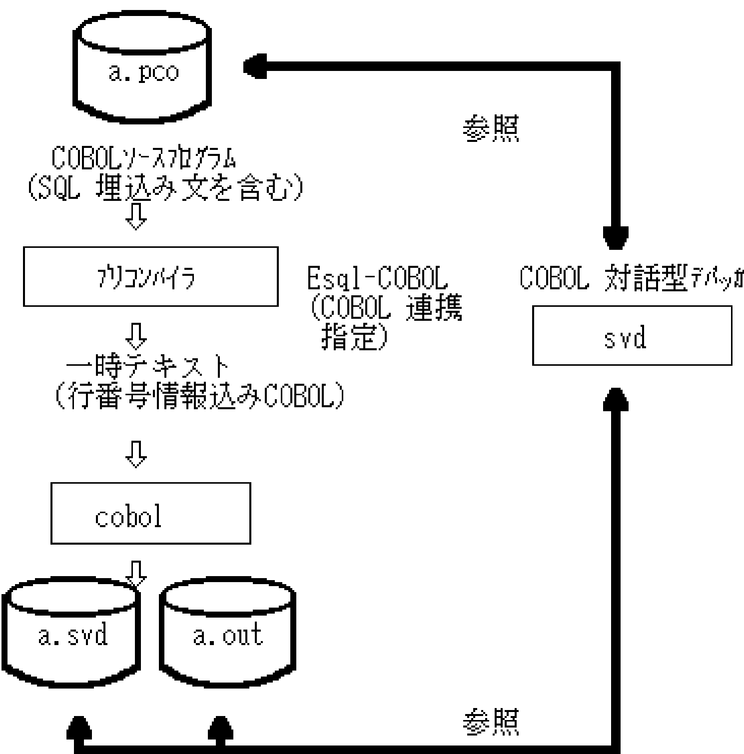
I.2 埋込みSQL文のデバッグまでの流れ

埋込みSQL文のデバッグ操作を可能にするためには、“I.1.1 Oracle連携”で示したように、DBごとに対処する必要があります。以下に各データベース連携時の流れ図を示します。

図I.1 Oracle連携時



図I.2 Symfoware連携時



I.2.1 埋込みSQL文のデバッグ方法

COBOL対話型デバッガでは、埋込みSQL文を含むCOBOLソースプログラムのデバッグ機能をサポートしています。以下に共通のデバッグ仕様を示します。

・ 埋込みSQL文のINCLUDE文は、COBOLのCOPY文と同じ扱いとなります。そのため、デバッガ起動時に必要な登録集の格納先(INCLUDE文の格納先)を指定します。そして"環境変更ウインドウ"で登録集を展開指定することにより、メインウインドウ中にINCLUDE文を展開表示することができます。

Oracle連携時の注意事項

デバッグ時に以下の注意事項があります。

・ Pro*COBOLが用意するSQL共通宣言集(SQLCA)などがある場合、デバッガでは登録集格納ディレクトリは1つしか指定できません。このためユーザが定義したINCLUDE文の格納ディレクトリに複写し、指定する必要があります。

・ 埋込みSQL文を含むCOBOLソースプログラムでは、"EXEC","SQL","END-EXEC","INCLUDE","SQLCA"のSQL文は、英大文字または英小文字で統一する必要があります。

・ COBOLソースプログラムで、以下をデータ名として使用すると、誤動作することがあります。

　－ "EXEC"

　－ "SQL"

　－ "INCLUDE"

　－ "SQLCA"

　－ "exec"

　－ "sql"

　－ "include"

－ “sqlca”

Symfoware連携時の注意事項

デバッグ時に以下の注意事項があります。

・ SQL文に対する中断点設定、追尾などを示す画面上のあみかけは、A領域(8～11カラム)に設定されます。

・ INCLUDE文が展開表示される場合、キーワードEXECの次行に展開されます。

I.3 デッドロック出口

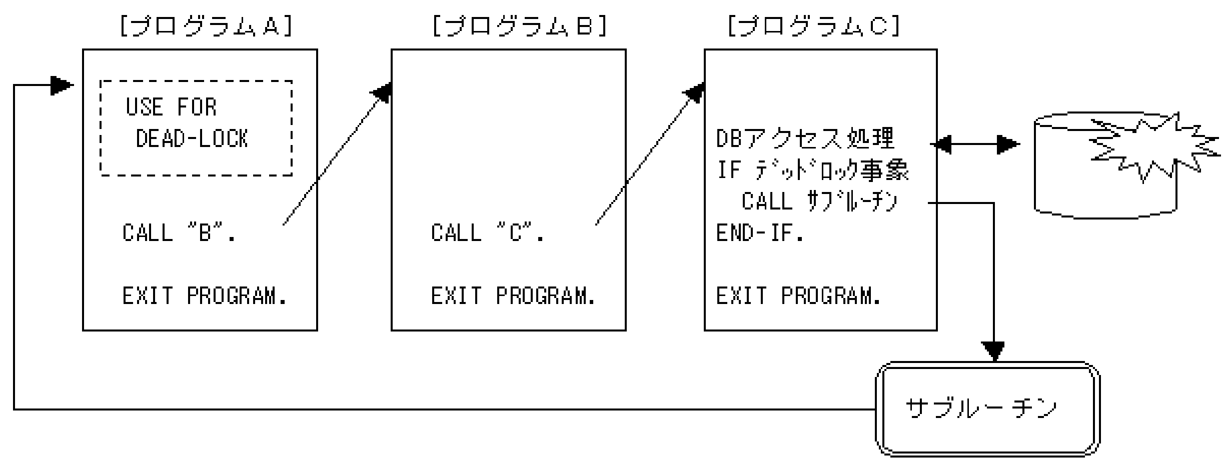
プログラムがデータベースをアクセスする時にアクセス競合が起こり、プログラム同士で占有解除を待つような状態が発生することをデッドロック状態といいます。デッドロック状態が発生すると、デッドロック事象がデータベースからプログラムに通知されます。

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。

I.3.1 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クラムへ制御を戻すことを言います。



データベースアクセスはプリコンパイラで展開されるので、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。

I.3.2 デッドロック出口スケジュールサブルーチン

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。

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。

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。

復帰値

サブルーチンからの復帰値はありません。

呼出し条件

データベースアクセスの実行によってSQLSTATEに復帰コードが通知されます。復帰コードにデッドロックを示す値が設定されているとき、デッドロック出口へプログラムの制御を戻すために呼び出します。

注意事項

- 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。

リンクに関する注意事項

デッドロック出口スケジュールサブルーチンを使用するプログラムを作成する場合、実行可能プログラムまたは副プログラムの共用ライブラリのリンク時に、libcobdlk.soをリンクしてください。libcobdlk.soは、NetCOBOLランタイムシステムのインストールディレクトリに格納される共用ライブラリです。マルチスレッドの場合は、librcobdlk.soになります。

例

共用ライブラリを作成する

- プロセスモデル

```
$ cobol  -dy  -G  -o libDEAD1.so  -lcobdlk  DEAD1.cob
```

- マルチスレッドモデル

```
$ cobol  -dy  -G  -Tm  -o libDEAD1.so  -lrcobdlk  DEAD1.cob
```

I.3.3 注意事項

- 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。

付録J　日本語コード系

ここでは、この製品での日本語の扱いについて説明します。

J.1　日本語処理のコード系

ここでは、この製品で日本語処理を行う場合のコード系について説明します。

J.1.1　概要

この製品では、日本語処理のコード系として以下を扱うことができます。

- Unicode
- EUC
- シフトJIS

コード系の指定は、システムの環境変数LANGに下表で示す値を設定して行います。

表J.1 日本語コード系の指定

コード系	LANGの指定値
EUC	ja ja_JP.U90 ja_JP.eucJP
Unicode	ja_JP.UTF-8
シフトJIS	ja_JP.PCK

コード系が1つのシステムでプログラムの作成、翻訳から実行までを行っている場合には、コード系を意識する必要はありません。しかし、コード系の異なるシステムを混在して使用している場合には、注意が必要です。

この製品のコード系には、プログラムごとのコード系と実行時のコード系という2つの意味があります。通常、プログラム中の日本語処理が正しく動作するためには、この2つのコード系は一致している必要があります。このため、この製品では、プログラムの実行時に2つのコード系の比較を行い、プログラムの実行に支障がないかどうかをチェックします。

プログラムごとのコード系、実行時のコード系、コード系の一致チェックについては、以降で説明します。

注意

日本語処理を行う場合だけ、環境変数LANGの指定値は意味を持ちます。しかし、この製品では、プログラムの実行時にコード系の一致チェックを行うため、プログラムごとのコード系と実行時のコード系は一致させるようにしてください。

J.1.2　プログラムごとのコード系と実行時のコード系

プログラムのコード系は、ソースプログラム中の日本語定数および日本語データ項目中の日本語データのコード系を決定するために使用されます。

プログラムのコード系は、cobolコマンドでソースプログラムを翻訳する際の環境変数LANGの指定値によって決まります。

実行時のコード系は、プログラムがシステムに対して、日本語データを表示したり印刷したりする際のコード系を決定するために使用します。

実行時のコード系は、実行可能プログラムを起動した際の環境変数LANGの指定値によって決まります。

下表に、プログラムごとのコード系と実行時のコード系の指定を示します。

表J.2 プログラムごとのコード系と実行時のコード系の指定

LANGの指定値 (注1)	プログラムごとのコード系および実行時のコード系	日本語処理の動作
ja ja_JP.U90 ja_JP.eucJP	EUC	EUCコード系で処理する。
ja_JP.PCK	シフトJIS	シフトJIS コード系で処理する。
ja_JP.UTF-8	Unicode	Unicode コード系で処理する。
上記以外	その他	EUC コード系で処理する。(注2)

注1: 環境変数LANGの指定値が有効になるのは、以下のタイミングです。

- プログラムごとのコード系:cobolコマンドの起動時
- 実行時のコード系:実行可能ファイルの起動時

注2: プログラムのコード系および実行時のコード系では、日本語は使用されていないものとして扱います。しかし、プログラム中では、EUCコード系の日本語処理機能が使用されているものとして処理します。

注意

- 日本語利用者語や日本語定数を含むソースプログラムを翻訳する場合は、ソースプログラムのコード系と、cobolコマンド起動時のコード系が、一致していなければなりません。
  一致している場合だけ、プログラムの翻訳結果を保証します。
- JMPCINT2/JMPCINT3を使用してCプログラムからCOBOLプログラムを呼び出す場合、実行時のコード系は、JMPCINT2を呼び出した時点の環境変数LANGによって決定されます。

J.1.3 コード系の一致チェック

この製品では、プログラムごとのコード系と実行時のコード系を比較し、プログラムごとに実行を認めるかどうかをチェックします。このチェックは、主プログラム、副プログラムの区別なく、各プログラムの呼出しに対して、そのプログラムがメモリ上にロードされた場合に行われます。

プログラムのコード系と実行時のコード系の組み合わせによる実行の可否を、下表に示します。

		プログラムのコード系			
		EUC	シフトJIS	Unicode	その他
実行時のコード系	**EUC**	○	×	×	○
	シフトJIS	×	○	×	○
	Unicode	×	×	○	○
	その他	×	×	×	○

○: 実行可能
×: 実行不可能(異常終了)

注意

日本語COBOL1.1以前のコンパイラで生成したプログラムは、EUCコード系のプログラムとして処理されます。

コード系一致チェックを迂回する方法

以下の方法によって、コード系一致チェックを迂回することができます。

翻訳オプションNOCODECHKを指定してプログラムを翻訳する方法

この方法によって翻訳されたプログラムは、コード系一致チェックの対象とはなりません。特定のプログラムだけをコード系一致チェックの対象から除外したい場合に便利です。

環境変数CBR_CODE_CHECKにnoを指定する方法

環境変数CBR_CODE_CHECKに対して文字列"no"を指定すると、主プログラム、副プログラムの区別なく、すべてのプログラムがコード系一致チェックの対象から除外されます。プログラムを限定しないで、コード系一致チェックを迂回したい場合に便利です。

注意

コード系一致チェックを迂回するプログラムの実行時に、コード系の不一致があると、正しい実行結果が得られない場合があります。コード系一致チェックを迂回するプログラムの作成には十分注意してください。

J.2 日本語文字の種類と表現

ここでは、COBOLで使用できる日本語文字の種類とその表現形式について説明します。

J.2.1 日本語文字の種類

日本語文字は、JIS漢字符号系(日本工業規格 JIS X0208-1990)に準拠した文字種を使用することができます。これらの文字種を以下に示します。

- JIS第1水準漢字
- JIS第2水準漢字
- JIS非漢字
- 拡張漢字、拡張非漢字

このほかに利用者が任意に定義した利用者定義文字を使用することができます。

J.2.2 日本語文字の表現形式

日本語文字の表現形式には、外部表現形式と内部表現形式の2種類があります。

外部表現形式とは、日本語文字をプログラムに記述したり、日本語文字を印刷装置や表示装置に印刷･表示したりするための形式です。内部表現形式とは、日本語文字をプログラム中でデータとして操作するための形式です。

COBOLの日本語項目、日本語編集項目および日本語定数の値は、内部表現形式で表されます。

外部表現形式には、環境変数LANGの指定により、以下のコードセットを使用できます。

- Unicodeコードセット
- EUCコードセット
- シフトJISコードセット

内部表現形式には、COBOL独自の16ビットワイドキャラクタ表現を使用しています。

外部表現形式がUnicodeコードセットの場合、内部表現形式との間に相違はありません。

外部表現形式がEUCコードセットの場合、内部表現形式との間に相違があります。以下に日本語文字のEUCコードセットでの外部表現形式と内部表現形式について説明します。

外部表現形式がシフトJISコードセット場合、内部表現形式との間に相違はありません。

日本語文字の各表現形式を“表J.3 外部表現形式”および“表J.4 内部表現形式”に示します。

表J.3 外部表現形式

文字種	外部表現形式(EUC コードセット表現)
JIS 第1水準漢字 JIS 第2水準漢字 JIS 非漢字	1xxx xxxx 1xxx xxxx
JIS カタカナ	SS2 1xxx xxxx
拡張漢字 拡張非漢字 利用者定義文字	SS3 1xxx xxxx 1xxx xxxx

参考

SS2、SS3は、それぞれ以下の16進の値です。

- SS2:0x8E
- SS3:0x8F

表J.4 内部表現形式

文字種	内部表現形式(COBOL16 ビットワイドキャラクタ表現)
JIS 第1水準漢字 JIS 第2水準漢字 JIS 非漢字	1xxx xxxx 1xxx xxxx
JIS カタカナ	SS2 1xxx xxxx
拡張漢字 拡張非漢字 利用者定義文字	1xxx xxxx 0xxx xxxx

参考

- 英小文字“x”で示されるビットの値は、外部表現形式と同じビットの値です。
- SS2は、以下の16進の値です。
  - SS2:0x8E

J.3　他システムからの移行上の注意

J.3.1　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

コード系	英数字の空白	日本語の空白	主なシステム
Unicode	X"20"	NX"3000"	Windows系、UNIX系

以下は、非互換が発生するプログラムの例です。

```
    WORKING-STORAGE SECTION.
    01 GR01.
      02 DATA1  PIC N(1).
    01 DATA2  PIC N(1).
         :
    MOVE SPACE TO GR01.  *> 英数字空白(X"2020")を転記
    MOVE SPACE TO DATA2. *> 日本語空白(X"8140")を転記
        :
    IF DATA1 = DATA2 THEN DISPLAY "EQUAL"
                     ELSE DISPLAY "NOT EQUAL".
```

上記のプログラムを実行すると、EBCDICコード系の場合は "EQUAL" が表示され、シフトJISコード系および動作モードがUnicodeの場合は "NOT EQUAL" が表示されます。

この場合、プログラムを修正して対応すべきですが、特定の条件下であれば、翻訳オプションNSPCOMP(ASP)を指定して日本語空白の比較方法を変更することによって、プログラムを修正せずに動作可能になります。

以下、注意事項と共に説明します。

使用する文や項目に関する条件

翻訳オプションNSPCOMPは、以下の文には作用しません。したがって、プログラム中の以下の文で日本語を扱わないことが条件です。

- INSPECT文
- STRING文
- UNSTRING文
- 索引ファイルのキー操作

2進項目などの非表示項目(USAGE DISPLAYではない項目)が含まれる集団項目に作用しない場合があります。注意してください。

NSPCOMPオプションが作用する比較について、下表にまとめます。

<table>
<tr><th colspan="3" rowspan="5"></th><th colspan="9">作用対象2</th></tr>
<tr><th colspan="6">データ項目</th><th colspan="3">定数</th></tr>
<tr><th colspan="4">集団項目</th><th colspan="2">基本項目</th><th rowspan="3">表意定数
SPACE</th><th rowspan="3">文字定数</th><th rowspan="3">日本語定数</th></tr>
<tr><th colspan="2">日本語項目あり</th><th colspan="2">日本語項目なし</th><th rowspan="2">英数字項目</th><th rowspan="2">日本語項目</th></tr>
<tr><th>非表示項目あり</th><th>非表示項目なし</th><th>非表示項目あり</th><th>非表示項目なし</th></tr>
<tr><td rowspan="4">作用対象1</td><td rowspan="2">日本語項目あり</td><td>非表示項目あり</td><td>－</td><td>●</td><td>－</td><td>－</td><td>－</td><td>●</td><td>－</td><td>－</td><td>●</td></tr>
<tr><td>非表示項目なし</td><td>●</td><td>●</td><td>●</td><td>●</td><td>●</td><td>●</td><td>●</td><td>●</td><td>●</td></tr>
<tr><td rowspan="2">日本語項目なし</td><td>非表示項目あり</td><td>－</td><td>●</td><td>－</td><td>－</td><td>－</td><td>●</td><td>－</td><td>－</td><td>●</td></tr>
<tr><td>非表示項目なし</td><td>－</td><td>●</td><td>－</td><td>－</td><td>－</td><td>●</td><td>－</td><td>－</td><td>●</td></tr>
</table>

		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					
	英数字項目		●	－	－	－	ERR	－	－	ERR
	日本語項目	●	●	●	●	ERR	●	●	ERR	●
	その他(2進項目など)		●	－	－	ERR	ERR	ERR	ERR	ERR

●： 作用対象1、作用対象2共に、全角空白を半角空白2文字に変換してから比較します。
－： 作用しません。
ERR： 翻訳エラーになります。

コード系共通の注意事項

- 日本語に泣き別れが発生するような部分参照を行っている場合は、NSPCOMP(ASP)を指定してもJEFと同じ結果にはなりません。

```
01 G1.
  02 G1-N  PIC N(4) VALUE SPACE.
01 G2.
  02 G2-N  PIC N(2) VALUE SPACE.
    :
IF G1(2:4) = G2 THEN DISPLAY "EQUAL"      *> JEFではEQUAL
                ELSE DISPLAY "NOT EQUAL". *> NSPCOMPを指定してもNOT EQUAL
```

- NSPCOMPは、等価比較だけでなく、大小比較にも作用します。

```
01 G1.
  02 G1-N  PIC N(1) VALUE SPACE. *> X"8140"
01 G2.
  02 G2-X  PIC X(2) VALUE SPACE. *> X"2020"
    :
IF G1 > G2 THEN DISPLAY "OK?"    *> NSPCOMP(NSP)ではTHEN
           ELSE DISPLAY "NG?".   *> NSPCOMP(ASP)ではELSE
```

- 集団項目中に2進項目などの非表示項目が含まれていた場合、誤動作する危険があります。

```
01 G1.
  02 G1-B  PIC S9(8) BINARY VALUE 33088.  *> X"00008140"
  02 G1-X  PIC  X(2)        VALUE SPACE.
01 G2.
  02 G2-B  PIC S9(8) BINARY VALUE  8224.  *> X"00002020"
  02 G2-N  PIC  N(1)        VALUE SPACE.
    :
IF G1 = G2 THEN DISPLAY "OK?"    *> NSPCOMP(ASP)ではTHEN
           ELSE DISPLAY "NG?".   *> NSPCOMP(NSP)ではELSE
```

動作モードがUnicode固有の注意事項

- Unicodeの場合、日本語字類でX"2020"に該当する文字(短剣符†)が存在します。このため、データ中に短剣符が含まれる場合、誤動作する可能性があります。

```
01 N     PIC N(1) VALUE NC"†".
    :
IF N = SPACE THEN DISPLAY "OK?"    *> NSPCOMP(ASP)ではTHEN
             ELSE DISPLAY "NG?".   *> NSPCOMP(NSP)ではELSE
```

・ 動作モードがUnicodeの場合、英数字項目の表現形式はUTF-8となります。全角空白のUTF-16表現とUTF-8表現は異なるため、シフトJISでの動作と異なります。

```
01 G1.
  02 G1-X  PIC X(8) VALUE "□□".  *> □は全角空白を表す。
01 N     PIC N(2) VALUE SPACE.
    :
IF G1 = N THEN DISPLAY "OK?"      *> シフトJISではTHEN
          ELSE DISPLAY "NG?".     *> UnicodeではELSE
```

J.3.2　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「MINUS SIGN」を採用しています。このため、日本語利用者語に「FULLWIDTH HYPHEN-MINUS」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(省略値)

・ DUPCHAR(EXT) ： FULLWIDTH HYPHEN-MINUS

上記の場合、DUPCHAR(EXT)を指定して翻訳してください。

J.3.3　英数字項目のカナ文字の扱い

EUCコード系では、カナ文字は2バイトで表現されます。

カナ文字を1バイトで表現するコード系からEUCコード系に移行した場合、英数字項目に格納できるカナ文字の文字数が少なくなります。移行元と同じ動作をさせるためには、カナ文字を格納する英数字項目の領域を拡張する必要があります。

```
01 DATA1 PIC X(2) VALUE "アイ".  *> EBCDICコード系、シフトJISコード系は翻訳正常終了
                                 *> EUCコード系では4バイトになるため翻訳エラー
```

対応方法としては、プログラムを修正すべきですが、プログラムの記述が以下の条件を満たす場合には、翻訳オプションKANA(JIS8)を指定することにより、プログラムを修正せずに動作可能になります。条件を満たさないプログラムに翻訳オプションKANA(JIS8)を指定した場合、動作は保証されません。

・ 英数字項目に格納する文字が1バイト文字またはカナ文字のみである。

・ DISPLAY文およびACCEPT文に指定する項目は基本項目のみである。

翻訳オプションKANA(JIS8)が指定された場合は、プログラムは以下のとおりに動作します。

・ 項目の宣言でVALUE句によって初期値を指定した場合
　カナ文字を1バイト(JISコード)で格納する。

・ MOVE文でカナ文字を含む定数を、基本項目または集団項目に転記した場合
　カナ文字を1バイト(JISコード)で転記する。

・ ACCEPT文に英数字項目または集団項目を指定した場合
　入力した文字列中のカナ文字を1バイト(JISコード)で格納する。

・ DISPLAY文に英数字項目または集団項目を指定した場合
　英数字項目に含まれるカナ文字を2バイト(EUCコード)で表示する。

・ 印刷ファイルのWRITE文に英数字項目、または集団項目を指定した場合
  英数字項目または集団項目に従属する英数字項目に含まれるカナ文字を2バイト(EUCコード)で印刷する。

付録K　SCCSの利用方法

ここでは、COBOLの開発でSCCS(Source Code Control System)を利用する方法について説明します。

SCCSは、ソースプログラムに対して行った変更履歴を保存しておきたい場合や、その履歴情報を実行可能オブジェクトに反映したい場合など、ファイルを管理するために使用します。

さらに詳しい説明については、manマニュアルおよびシステムのマニュアルを参照してください。

K.1 プログラムの記述方法

SCCSを利用する場合のCOBOLプログラムの記述方法について説明します。

ファイルの履歴管理のために、SCCSから以下のようなマクロ文字列が提供されています。

%W%

“@(#)”とプログラム名と修正版数を組み込むマクロ文字列

%Z%

“@(#)”を組み込むマクロ文字列

%G%

プログラムの最終更新日付を組み込むマクロ文字列

%M%

プログラム名を組み込むマクロ文字列

%I%

修正版数を組み込むマクロ文字列

ここで、“@(#)”は、whatコマンドにより認識できる文字列です。

COBOLプログラム上にマクロ文字列を記述し、SCCSの機能である登録(deltaコマンド)、取り出し(getコマンド)を実行することでCOBOLプログラムの履歴を管理することができます。もし、COBOLプログラムの履歴情報を実行可能オブジェクトまで反映させたいのであれば、マクロが展開する文字列が翻訳後もオブジェクトに静的結合される場所にSCCSのマクロ文字列を記述する必要があります。COBOLの場合は、定数節(CONSTANT SECTION)に記述します。
COBOLプログラムに記述した場合の例を以下に示します。

```
       :
  DATA DIVISION.
  CONSTANT SECTION.
  01  SCCSID    PIC X(30)    VALUE "%W%>".
       :
```

ここで、“>”は、whatコマンドが認識できる区切り文字です。このほかに区切り文字には、”、> 、改行、¥、NULLがあります。
C言語で記述した場合の例を以下に示します。

```
  static char SccsId[] = "%W%";
```

上のように記述した場合、実際に翻訳用にgetコマンドで取り出したCOBOLプログラムでは、SCCSの機能により、以下に示すようにマクロ文字列がwhatコマンドで参照できる文字列に置き換えられます。

```
01  SCCSID    PIC X(30)    VALUE "@(#)PROG.cob    1.2".
```

このようにSCCSの履歴情報を記述したCOBOLプログラムを翻訳・リンクすると、作成した実行可能オブジェクトにも、この履歴情報は反映されます。

K.2 履歴情報の参照方法

SCCSを使って履歴情報が組み込まれた実行可能オブジェクトの履歴情報を参照するには、whatコマンドを実行します。
履歴情報を参照した場合の例を以下に示します。

```
$ what PROG

 PROG:
           PROG.cob     1.2
```

付録L　ldコマンド

ここでは、COBOLコンパイラが生成した再配置可能プログラムをリンクするときのldコマンドの入力形式および使い方について説明します。

L.1　入力形式

```
$ ld ［オプションの並び］ スタートアップルーチン　ファイル名　…　ライブラリ名
```

オペランドの説明

オプションの並び

ldコマンドのオプションについては、ldコマンドのマニュアルを参照してください。

スタートアップルーチン

以下のファイルを格納してあるパス名を付加して番号の順に指定します。

－ crti.o,crt1.o,crtn.o(C言語用のスタートアップルーチン)

－ cblstr.o(COBOL用のスタートアップルーチン)

ファイル名

静的結合を行う場合には、結合したいオブジェクトファイル名をすべて指定し、動的結合を行う場合には、主プログラムのオブジェクトファイル名だけを指定します。

ライブラリ名

実行可能プログラムを作成する場合、以下のライブラリを、-lオプションを使ってコマンドラインの最後に指定します。

機能	ライブラリ名		必須
	プロセスモデルのプログラムリンク時	マルチスレッドモデルのプログラムリンク時	
動的リンク構造の実行可能プログラム	副プログラムの共用オブジェクトファイル(プロセスモデルのプログラム)	副プログラムの共用オブジェクトファイル（マルチスレッドモデルのプログラム）	
COBOLランタイム	libcobol.so libFJBASE.so(*1)	librcobol.so libFJBASE.so(*1)	○
CGIサブルーチン	libcobw3cgi.so	－	
SAFサブルーチン	－	libcobw3saf.so	
簡易アプリ間通信機能	libcobcicl.so	librcobcicl.so	
デッドロック出口サブルーチン	libcobdlk.so	librcobdlk.so	
C-ISAMファイル	libcobcim.so libisam.a	librcobcim.so libisam.a	
スクリーン操作機能	libcobscr.so libcurses.a	librcobscr.so libcurses.a	
MeFtを使って画面帳票定義体の使用	libXm.so libX11.so libXt.so	libXm.so libX11.so libXt.so	
C標準ライブラリ（システム）	libc.so	libthread.so libc.so	○
ダイナミックリンクライブラリ（システム）	libdl.so	libdl.so	○

*1 ： libFJBASE.soはCOBOLで作成したオブジェクト指向プログラムをリンクする場合に必須となります。

注意

- libcobcim.soとlibisam.aをリンクする場合には、libcobcim.soが先にリンクされるように指定してください。また、libcobol.soよりもlibcobcim.soおよびlibisam.aが先にリンクされるように指定してください。
- libisam.aはC-ISAMから提供されるアーカイブライブラリです。リンク時にlibisam.aが正しく検索されるように環境変数LD_LIBRARY_PATHを設定してください。
- マルチスレッドモデルのプログラムで、libthread.soとlibc.soをリンクする場合には、libthread.soが先にリンクされるように指定してください。
- マルチスレッドモデルのプログラムのリンク時には、マルチスレッドを未サポートとするライブラリを-lオプションで指定しないでください。

L.2 ldコマンドの使い方

ここでは、cobolコマンドで作成した再配置可能プログラムを、ldコマンドを使って実行可能プログラムを生成する方法について例を使って説明します。

L.2.1 リンクをcobolコマンドで行う場合との比較

"L.1 入力形式"で説明したように、ldコマンドを使ってリンクを行う場合、利用者は、COBOLが提供する各種ライブラリサブルーチンをldコマンドに指定する必要があります。

以下に、cobolコマンドでリンクを行う場合とldコマンドを使ってリンクを行うときの比較を示します。

cobolコマンドで翻訳からリンクまでを一度に行う場合

```
$ cobol -dy -M -o P1 P1.cob
最大重大度コードはＩで，翻訳したプログラム数は１本です．
```

cobolコマンドで翻訳を行い、cobolコマンドでリンクを行う場合

```
$ cobol -c -M -o P1 P1.cob
最大重大度コードはＩで，翻訳したプログラム数は１本です．
$ cobol -dy -o P1 P1.o
```

cobolコマンドで翻訳を行い、ldコマンドでリンクを行う場合

```
$ cobol -c -M P1.cob
最大重大度コードはＩで，翻訳したプログラム数は１本です．
$ ld -dy -o P1 $CCSLIB/crti.o $CCSLIB/crt1.o $CCSLIB/crtn.o $COBLIB/cblstr.o  ￥
P1.o -lcobol -lc -ldl
```

cobolコマンド

入力	:	P1.cob(ソースファイル)
出力	:	P1.o(オブジェクトファイル)
オプション	:	-M(主プログラムの指定)　-c(翻訳だけ行う指定)

ldコマンド

入力	:	crti.o crt1.o crtn.o cblstr.o(スタートアップルーチン) P1.o(オブジェクトファイル) libcobol.so libc.so libdl.so (ライブラリ)
出力	:	P1(実行可能ファイル)
オプション	:	-dy (動的結合の指定)　-o(実行可能プログラムの出力先) -l (リンクするライブラリ)
$CCSLIB	:	スタートアップルーチンの格納ディレクトリ
$COBLIB	:	COBOL用のスタートアップルーチンの格納ディレクトリ

L.2.2 プログラム構造ごとのldコマンドの使い方

ここでは、単純構造、動的リンク構造および動的プログラム構造の実行可能プログラムを作成するときのldコマンドの使い方について説明します。

単純構造の実行可能プログラムを作成する場合

主プログラム(P1)と副プログラム(SUB)で単純構造の実行可能プログラムを作成する場合のldコマンドの使用例を示します。

```
$ LD_LIBRARY_PATH=.:/opt/FJSVcbl/lib:$LD_LIBRARY_PATH                    [1]
$ export LD_LIBRARY_PATH
$ CCSLIB=/opt/FJSVcbl/lib ; export CCSLIB                                 [2]
$ COBLIB=/opt/FJSVcbl/lib ; export COBLIB                                 [3]
$ ld -dy -o P1 $CCSLIB/crti.o $CCSLIB/crt1.o $CCSLIB/crtn.o $COBLIB/cblstr.o ¥
    P1.o SUB.o -lFJBASE -lcobol -lc -ldl                                  [4]
```

[1] 副プログラムの共用オブジェクトファイルの格納ディレクトリ(この例ではカレントディレクトリとします)とCOBOLランタイムシステムの格納ディレクトリを環境変数LD_LIBRARY_PATHに設定します。LD_LIBRARY_PATHは、アーカイブファイルまたは共用オブジェクトファイルの検索パスを設定する環境変数です。

[2] C言語用のスタートアップルーチンの格納ディレクトリを環境変数CCSLIBに設定します。
この設定は、ldコマンドでのファイル名の指定を簡単にするために行います。

[3] [2]と同様にCOBOLのスタートアップルーチンの格納ディレクトリを環境変数COBLIBに設定します。

[4] ldコマンドを実行します。

動的リンク構造の実行可能プログラムを作成する場合

主プログラム(P1)と副プログラム(SUB)で動的リンク構造の実行可能プログラムを作成する場合のldコマンドの使用例を示します。

副プログラムの共用オブジェクトプログラムの作成

```
$ ld -dy -G -o libSUB.so SUB.o -lcobol -L/opt/FJSVcbl/lib
```

ldコマンドを実行して副プログラムの共用オブジェクトプログラムを作成します。

実行可能ファイルの作成

```
$ LD_LIBRARY_PATH=.: /opt/FJSVcbl/lib:$LD_LIBRARY_PATH                   [1]
$ export LD_LIBRARY_PATH
$ CCSLIB=/opt/FJSVcbl/lib ; export CCSLIB                                 [2]
$ COBLIB=/opt/FJSVcbl/lib ; export COBLIB                                 [3]
$ ld -dy -o P1 $CCSLIB/crti.o $CCSLIB/crt1.o $CCSLIB/crtn.o $COBLIB/cblstr.o ¥
    P1.o -lFJBASE -lSUB -lcobol -lc -ldl                                  [4]
```

[1] 副プログラムの共用オブジェクトファイルの格納ディレクトリ(この例ではカレントディレクトリとします。)とCOBOLランタイムシステムの格納ディレクトリを環境変数LD_LIBRARY_PATHに設定します。LD_LIBRARY_PATHは、アーカイブファイルまたは共用オブジェクトファイルの検索パスを設定する環境変数です。

[2] C言語用のスタートアップルーチンの格納ディレクトリを環境変数CCSLIBに設定します。
この設定は、ldコマンドでのファイル名の指定を簡単にするために行います。

[3] [2]と同様にCOBOLのスタートアップルーチンの格納ディレクトリを環境変数COBLIBに設定します。

[4] ldコマンドを実行します。

動的プログラム構造の実行可能プログラムを作成する場合

主プログラム(P1)と副プログラム(SUB)で動的プログラム構造の実行可能プログラムを作成する場合のldコマンドの使用例を示します。

副プログラムの共用オブジェクトプログラムの作成

```
$ ld -dy -G -o libSUB.so SUB.o -lcobol -L/opt/FJSVcbl/lib
```

ldコマンドを実行して副プログラムの共用オブジェクトプログラムを作成します。

実行可能ファイルの作成

```
$ LD_LIBRARY_PATH=.: /opt/FJSVcbl/lib:$LD_LIBRARY_PATH                         [1]
$ export LD_LIBRARY_PATH
$ CCSLIB=/opt/FJSVcbl/lib ; export CCSLIB                                      [2]
$ COBLIB=/opt/FJSVcbl/lib ; export COBLIB                                      [3]
$ ld -dy -o P1 $CCSLIB/crti.o $CCSLIB/crt1.o $CCSLIB/crtn.o $COBLIB/cblstr.o  ¥
     P1.o -lFJBASE -lcobol -lc -ldl                                            [4]
```

[1] 副プログラムの共用オブジェクトファイルの格納ディレクトリ(この例ではカレントディレクトリとします)とCOBOLランタイムシステムの格納ディレクトリを環境変数LD_LIBRARY_PATHに設定します。LD_LIBRARY_PATHは、アーカイブファイルまたは共用オブジェクトファイルの検索パスを設定する環境変数です。

[2] C言語用のスタートアップルーチンの格納ディレクトリを環境変数CCSLIBに設定します。
この設定は、ldコマンドでのファイル名の指定を簡単にするために行います。

[3] [2]と同様にCOBOLのスタートアップルーチンの格納ディレクトリを環境変数COBLIBに設定します。

[4] ldコマンドを実行します。動的プログラム構造の実行可能プログラムを作成する場合、副プログラムの共用オブジェクトプログラムをリンクする必要はありません。

付録M　makeコマンドの活用

ここでは、COBOL開発でのmakeコマンドの活用方法について説明します。

M.1　makeコマンドについて

COBOLの開発で依存関係のある複数の資源を扱う場合、makeコマンドを活用することで、より簡単かつ確実にプログラム開発を行うことが可能になります。

特に、オブジェクト指向プログラミングを行う場合には、多くの資源の関係を考慮する必要があるため、makeコマンドを利用することをおすすめします。

また、プロジェクトマネージャを利用すれば、簡単な資源構成の定義だけで自動的にMakefileが生成され処理されるため、Makeに関する知識がなくても、確実なプログラム開発が可能です。

なお、makeコマンドおよびMakefileの詳細については、manマニュアルおよびシステムのマニュアルを参考にしてください。

M.2　Makefileの記述方法

COBOLでのMakefileの記述方法について説明します。

M.2.1　基本的な記述方法

Makefileの基本は、以下の記述形式の繰返しで構成されています。

```
ターゲット : 依存ファイル ...
          コマンド ...
           :
```

ターゲット

作成対象とするファイル名を指定します。

依存ファイル

ターゲットを作成するために必要なファイルです。ターゲットが存在しない場合または、ここで記述されたファイルのどれかの最終更新日時がターゲットの最終更新日時よりも新しい場合、ターゲットの作成処理が実施されます。さらに、ここに記述されている依存ファイルがターゲットとして定義されている場合、そのターゲットに対する処理を優先実行します。

コマンド

ターゲットを作成するためのコマンドを指定します。

例

COBOLでのリンク規則の記述例

```
実行形式プログラム名 : オブジェクトファイル名 ...
     cobol -o  実行形式プログラム名  オブジェクトファイル名 ...
```

COBOLでの翻訳規則の記述例

```
オブジェクトファイル名 : COBOLソースファイル名  登録集ファイル名 ...
     cobol -c COBOLソースファイル名
```

M.2.2　COBOL資源の依存関係

COBOLでは、以下の資源に関する依存関係を定義する必要があります。

```
ターゲット : ¥
     依存ファイル
```

```
実行形式 : ¥
      オブジェクト、共用オブジェクト
共用オブジェクト : ¥
      オブジェクト、共用オブジェクト
オブジェクト : ¥
      ソース、登録集、各種定義対、依存リポジトリ、オプションファイル
ターゲットリポジトリ : ¥
      ソース、登録集、各種定義対、依存リポジトリ、オプションファイルまたはオブジェクト
分離されたメソッドのオブジェクト : ¥
      ソース、登録集、各種定義対、依存リポジトリ、オプションファイル、¥
      分離されたメソッドを定義しているクラスのオブジェクト
```

M.2.3 クラスが相互に参照している場合の依存関係

クラスが相互に参照している場合、依存関係が相互になります。このため単純な依存関係では、翻訳時に必要なリポジトリファイルまたはリンク時に必要な共用オブジェクトの作成ができないことがあります。この現象を回避するためには、従来の翻訳、リンクを行う前に、リポジトリファイルを作るための翻訳と、依存関係を持たないリンクを行う必要があります。

相互参照を持つクラスオブジェクト作成の共用には以下の手順を踏む必要があります。

1. リポジトリファイルの作成
   翻訳に必要なリポジトリファイルがそろっていない可能性があります。そのため、クラスを翻訳オプション "CREATE(REP)"を付けて翻訳し、リポジトリファイルを作成します。
   相互参照ではなく、親クラスから子クラスを参照している場合は、親クラス、子クラスの順に“CREATE(REP)”を付けて翻訳し、リポジトリファイルを作成します。

2. オブジェクトファイルの作成
   翻訳オプションに“CREATE(OBJ)”を付けて翻訳し、オブジェクトファイルを作成します。

3. 共用オブジェクトがリンクされていない共用オブジェクトの作成
   リンクに必要な共用オブジェクトがそろっていない可能性があります。そのため、-l オプションを付けずにオブジェクトをリンクし、共用オブジェクトを作成します。

4. 共用オブジェクトをリンクしている共用オブジェクトの作成
   -l オプションを付けてオブジェクトをリンクし、共用オブジェクトを作成します。

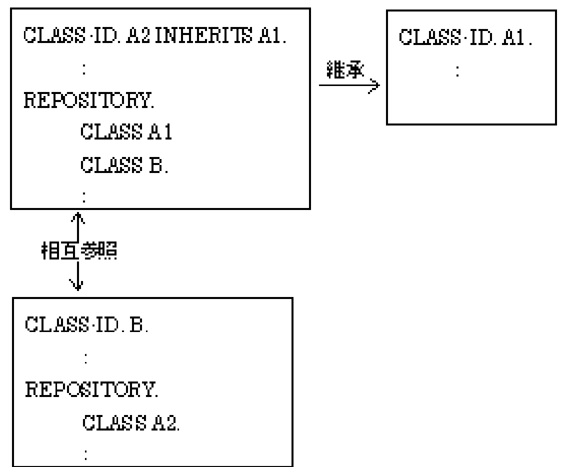
Makefileで 3.と4.を自動的に切り分けさせるためにはフラグとして仮リンク識別ファイルを使用する必要があります。

クラスが相互に参照している場合のMakefileは以下のように記述します。

```
リポジトリファイル : ソース 登録集 各種定義体 継承クラスのリポジトリ
        cobol -c -WC,"CREATE(REP)" ソース

オブジェクト : ソース 登録集 各種定義体 依存リポジトリ オプションファイル
        cobol -c -WC,"CREATE(OBJ)" ソース

仮リンク識別 : オブジェクト
        cobol -G -o 共用オブジェクト オブジェクト
        sleep 1
        touch 仮リンク識別

共用オブジェクト : オブジェクト 依存仮リンク識別
        cobol -G -o 共用オブジェクト -l 共用オブジェクト オブジェクト
        touch 仮リンク識別
```

たとえば、クラスA1を継承するクラスA2とクラスBが相互参照の関係にある場合、Makefileは以下のようになります。

関連図



Makefile

```
.SUFFIXES :
.SUFFIXES : .cob .rep .o .so

all: libA1.so libA2.so libB.so

### 1.リポジトリファイルの作成
.cob.rep:
	cobol -c -WC,"CREATE(REP)" $<

A2.rep: A1.rep
	cobol -c -WC,"CREATE(REP)" $<

### 2.オブジェクトファイルの作成
A1.o: A1.cob
	cobol -c -WC,"CREATE(OBJ)" A1.cob

A2.o: A2.cob A1.rep B.rep
	cobol -c -WC,"CREATE(OBJ)" A2.cob

B.o: B.cob A2.rep
	cobol -c -WC,"CREATE(OBJ)" B.cob

### 3.共用オブジェクトがリンクされていない共用オブジェクトの作成
A2.1: A2.o
	cobol -G -o libA2.so A2.o
	sleep 1
	touch $@

B.1: B.o
	cobol -G -o libB.so B.o
	sleep 1
	touch $@

### 4.共用オブジェクトをリンクしている共用オブジェクトの作成
libA1.so: A1.o
	cobol -G -o $@ A1.o

libA2.so: A2.o B.1 libA1.so
	cobol -G -o $@ -L. -lA1 -lB A2.o
	touch A2.1

libB.so: B.o A2.1
```

```
cobol -G -o $@ -L. -lA2 B.o
touch B.1
```

M.2.4 Makefile作成支援コマンド

Makefileの作成支援用に以下のコマンドを提供しています。

コマンド名	使用目的
cobmkmf	COBOL用Makefileの作成
cobdepend	COBOLソースプログラムの依存関係の調査
pmgr_rename	ファイル名の変換
pmgr_chsuffix	拡張子の変換
pmgr_getsym	実行形式プログラムまたは共用オブジェクトプログラムから外部シンボル名の一覧を表示する
pmgr_nonsuffix	拡張子を除いたファイル名を表示する

なお、各コマンドの詳細については、manマニュアルを参照してください。

M.2.5 Makefileのサンプル

Makefileの記述例として、cobmkmfコマンドで作成したMakefileまたは各サンプルプログラムに添付されているMakefileを参考にしてください。

付録N　Windowsを利用したUNIX版COBOLアプリケーションの作成手順

ここでは、Windowsを利用してUNIX版COBOLアプリケーションを作成するための手順について説明します。

UNIX版COBOLアプリケーションを開発する場合でも、より豊富な開発支援ツールがあるWindowsを活用することでより生産性の高い開発を行うことができます。

N.1　Windowsでの分散開発作業

豊富な開発支援ツールがあるWindowsを利用し、設計から試作までの作業を行えます。

Windows版 COBOL開発環境では、以下に示すプログラムおよびユーティリティが提供されており、Windows上でのCOBOLプログラムの開発および試作を行うことができます。

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

また、以下のような関連製品および各種ユーティリティも豊富に整っており、より生産性の高い開発を行うことができます。

関連製品名	使用目的
FORM	画面・帳票定義体を作成するツールです。 作成した画面・帳票定義体は、MeFtとの連携により、ユーザの画面からの入力操作の制御や、プリンタの表現力を最大限に活かしたきめこまかい表現力豊かな帳票印刷を実現します。
SIMPLIA	テストデータや各種仕様書の作成などソフトウェアの開発・保守作業の効率化に役立つツール群です。
PowerGEM Plus	Windowsをクライアントとしたクライアント/サーバ環境で、アプリケーション開発を効率よく行えるよう資産管理および各ターゲットシステムとの連携機能を提供します。ソースプログラムなど開発資産の修正履歴管理・版数管理およびシステム間での資産移行を容易に行え、品質の向上、開発効率の向上が図れます。
PowerGEM Plus Administrator	マルチサーバ環境での大規模なチーム開発の構成管理を行うための管理者向けツールで、大規模分散開発での資産の一元管理を実現します。

N.2　WindowsからUNIXへのユーザ資産の移行

Windowsで開発したユーザ資産（COBOLソース、登録集、各種定義体など）をPowerGEMやftpコマンドを利用してUNIX上に移行します。

なお、開発運用資産(プロジェクトファイル、Makefile、翻訳オプションファイル、オブジェクトファイルおよび実行形式ファイルなど)については UNIXに移行しても正しく動作しません。

注意

移行時には、文字コード系およびファイル名の大文字／小文字の扱いに注意する必要があります。特に拡張子(cobol,cob,cblなど)については、小文字にする必要があります。以下のコマンドを活用し対処してください。

コマンド名	使用目的
Iconv	コードセット変換ユーティリティ
pmgr_rename	ファイル名の変換

なお、各コマンドの詳細については、manマニュアルを参照してください。

N.3 UNIXでの開発作業

UNIX上では構築・テスト・デバッグを行います。

Windowsから移行してきたユーザ資産はUNIX上で構築しなおし、テスト・デバッグする必要があります。

UNIX上での開発作業を効率的に実施できるように以下のコマンドおよびユーティリティを提供しています。

名称	コマンド名	使用目的
COBOL コンパイラ	cobol	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
プロジェクトマネージャ	pmgr	COBOL アプリケーションの開発支援
エディタ	pmgredit	ソースの編集
メッセージ管理ツール	pmgrmsg	エラーなどのタグジャンプ可能なメッセージを管理
ソース解析ツール	pmgrgrep	ソースプログラム中のデータ名や呼出し関係などの解析
ビルダ	pmgrbuild	Makefileによる翻訳およびリンクの実施
翻訳オプション設定ツール	pmgrcbi	COBOL 翻訳オプションの設定
リンクオプション設定ツール	pmgrlni	COBOL リンクオプションの設定
COBOL 対話型デバッガ	svd	COBOL アプリケーションのデバッグ
SCREEN DESIGNER	sdesign	COBOL スクリーン節の設計
COBOL ファイルユーティリティ	cobfuty	COBOL ファイルの処理
クラス情報出力コマンド	pmgr_vrep	COBOL クラス情報の表示
Makefile作成支援コマンド	cobmkmf	COBOL用Makefileの作成

各コマンドおよびユーティリティの詳細については、各ツールのヘルプまたはmanマニュアルを参照してください。

注意

プロジェクトマネージャなどのXウィンドウ上で動作するツールは、将来のNetCOBOLリリースで提供を停止する予定です。Windows版NetCOBOL Studioのリモート開発による開発形態への移行をご検討ください。

Xウィンドウが利用できる場合

プロジェクトマネージャの依存関係調査機能により、Makefileを自動生成することが可能となり、簡単に再構築が実施できます。また、テストおよびデバッグもプロジェクトマネージャから容易に行えるため便利です。

“第22章 プロジェクトマネージャの使い方”を参照してください。

Xウィンドウが利用できない場合

cobmkmfコマンドでMakefileを生成し、makeコマンドで再構築することをおすすめします。これにより、確実かつ効率的な構築作業を実施することができます。

“付録M makeコマンドの活用”を参照してください。

付録O　OSIV系システムとの機能比較

OSIV系システム(グローバルサーバまたはPRIMEFORCEシリーズで動作するOS)とこのシステムのCOBOLの機能比較を“表O.1 OSIV系システムと本システムの機能比較”に示します。

“表O.1 OSIV系システムと本システムの機能比較”で、“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。
△：この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。しかし、実行時にその機能が有効となりません。
■：使用できます。しかし、OSIV系システムと動作が異なります。
×：このシステムでは使用できません。

表O.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		システム依存	○	
COBOLの語	利用者語	利用者語の種類すべて	○	_(アンダースコア)の使用はこの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 このシステム：S9(9) COMP-5
		上記以外	○	
	機能名	SYSPUNCH STACKER-01～12 CSP S01～02 SYSPCH BUSHU SOKAKU ON-YOMI KUN-YOMI	●	
		SWITCH-8 SYSERR	△	
		CHANNEL02～12 C02～12	●	
		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	×	

機能分類				比較	備考
分類		機能概要			
		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してください。
		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に従います。 このシステム:自動的に判定します。
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	EXTERNAL REFERENCE EXTERNAL DEFINITION		△	
		上記以外の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	COMP-5		△	
		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どうかは、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	○	
		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		○	

機能分類			比較	備考
分類		機能概要		
	ファイル定義	CODE-SET句	●	
		BLOCK CONTAINS句	□	この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	□	KOL5固有です。
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	□	この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システム固有です。 このシステムではデータ名の総数、総長に制限があります。
		上記以外	○	
	ファイル定義	CODE-SET句	●	
		BLOCK CONTAINS句	□	この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システム固有です。
		上記以外	○	

機能分類			比較	備考
分類		機能概要		
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	CALL文	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システム固有です。
		OF/INのXMDLIB、XFDLIB指定	△	
		登録集原文名定数	△	
		JOINING指定だけの指定	△	
報告書作成	ファイル定義	BLOCK CONTAINS句 CODE句	●	
		上記以外	○	
	その他	すべて	□	日本語項目の出力は正しく動作しません。
表示ファイル	環境定義	SYMBOLIC DESTINATION句の"CMD"、"TRM"、"WST"指定	●	
		APPLY MULTICONVERSATION-MODE句	●	
		PROCESSING TIME句	△	
		DESTINATION CONTROL句	●	
		MESSAGE SEQUENCE句	●	
		上記以外	○	
	ファイル定義	EXTERNAL	□	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。
		上記以外	○	
	入出力文	すべて	○	
	特殊レジスタ	すべて	○	
	その他	フォームオーバレイ	□	KOL5固有です。
		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。
		上記以外	○	
デバッグ機能		COUNT	○	
		CHECK 指定なし	■	

<table>
<tr><th colspan="3">機能分類</th><th rowspan="2">比較</th><th rowspan="2">備考</th></tr>
<tr><th>分類</th><th colspan="2">機能概要</th></tr>
<tr><td rowspan="4"></td><td rowspan="2"></td><td>EXTEND</td><td>●</td><td></td></tr>
<tr><td>上記以外</td><td>△</td><td></td></tr>
<tr><td colspan="2">TRACE</td><td>△</td><td></td></tr>
<tr><td colspan="2">すべて</td><td>●</td><td></td></tr>
<tr><td>区分化機能</td><td colspan="2">すべて</td><td>●</td><td>OSIV系システムも翻訳だけです。</td></tr>
<tr><td>通信機能</td><td colspan="2">すべて</td><td>●</td><td></td></tr>
<tr><td>拡張機能 / システム制御</td><td colspan="2">すべて</td><td>●</td><td></td></tr>
<tr><td>拡張機能 / ネットワークデータベース</td><td colspan="2">すべて</td><td>●</td><td></td></tr>
<tr><td>拡張機能 / AIM/RDB</td><td colspan="2">すべて</td><td>×</td><td></td></tr>
<tr><td>拡張機能 / SD機能</td><td colspan="2">すべて</td><td>○</td><td></td></tr>
<tr><td>浮動小数点</td><td colspan="2">すべて</td><td>■</td><td>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。</td></tr>
<tr><td rowspan="2">組込み関数機能</td><td colspan="2">CURRENT-DATE関数</td><td>○</td><td></td></tr>
<tr><td colspan="2">上記以外</td><td>△</td><td></td></tr>
<tr><td>スクリーン操作機能</td><td colspan="2">すべて</td><td>△</td><td></td></tr>
<tr><td>コマンド行引数/環境変数操作機能</td><td colspan="2">すべて</td><td>△</td><td></td></tr>
<tr><td>オブジェクト指向機能</td><td colspan="2">すべて</td><td>△</td><td></td></tr>
<tr><td>翻訳オプションカスタマイズ機能</td><td colspan="2">すべて</td><td>×</td><td></td></tr>
<tr><td>ユーザテーラリング機能</td><td colspan="2">すべて</td><td>×</td><td></td></tr>
</table>

プログラムの動作確認

共通の機能範囲内で作成したプログラムは、このシステム上で動作確認を行うことができます。ただし、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システムにより異なる場合があります。

OSIV系システムを使ったプログラムをこのシステム上で動作させるには、特別な操作が必要となります。OSIV系システムを使ったプログラムをこの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

たとえば、通信機能を使ったプログラムを動作させる場合、入出力文を順ファイルに対して実行します。順ファイルの内容を確認することにより、プログラムが意図したとおりに動作しているかを確認することができます。

対話型デバッガを使って実行不可能な命令を迂回する方法

たとえば、ファイル処理の対象となるファイルまたは副プログラムが存在しない場合、対話型デバッガを使って、入出力文およびCALL文を迂回することができます。対話型デバッガの使用方法については、"第23章 対話型デバッガの使い方"を参照してください。

- 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(このシステム)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を使用する場合は、FCB制御文が必要となります。

- 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。詳細については、FORMに添付されている文書の“OS別留意事項”(system.txt)を参照してください。
- 画面帳票定義体の移行については、“OSIV PSAM使用手引書”の“ホスト・ワークステーション連携”を参照してください。

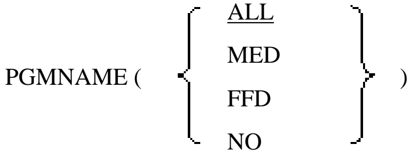
付録P　COBOL G移行支援オプション

SX/GシリーズおよびKシリーズのCOBOL G資産を活用するために、NetCOBOLシステムでは、移行支援オプションを用意しています。ただし、これらのオプションは、クラス定義およびメソッド定義には指定できません。

登録集の項目名の展開方法の指定

画面帳票定義体またはファイル定義体からCOPY文を使ってレコード定義を取り込むときに展開される項目名を、プログラミング名で展開するか、英数字項目名で展開するか、翻訳オプションPGMNAMEを使って指定します。

翻訳オプションPGMNAMEの指定形式



意味

画面帳票定義体またはファイル定義体の項目名の展開を指定します。

－ ALL :画面帳票定義体、ファイル定義体ともにプログラミング名で展開します。

－ MED :画面帳票定義体だけプログラミング名で展開します。

－ FFD :ファイル定義体だけプログラミング名で展開します。

－ NO :画面帳票定義体、ファイル定義体ともに英数字項目名で展開します。

プログラミング名での展開を指定した場合、項目定義にプログラミング名が存在しなければ、英数字項目名で展開します。

COBOL GからNetCOBOLへの移行支援

NetCOBOLでは次に示すCOBOL Gの機能を翻訳オプションで指示することにより、使用することができます。

- 名前を省略したクラス“LP”または“SP”のファイル識別名がファイル参照子に指定された場合、印刷ファイルの出力先を印刷装置とします。
  (NetCOBOLでは、ファイル参照子にPRINTERを指定した場合、印刷ファイルの出力先が印刷装置となります。)
- APPLY INDEXED-AREA句が記述できます。
  (NetCOBOLでは、翻訳エラーとなります。)

翻訳オプションMODEの指定形式

MODE(CBLG)

意味

COBOL G からNetCOBOL への移行支援を行うことを指定します。

MODE(CBLG)を指定した場合の扱いを、以下に示します。

－ 順ファイルのファイル参照子に、名前を省略したクラス“LP” または“SP”のファイル識別名が記述された場合、ファイル参照子にPRINTER が記述されたものとして扱います。

－ APPLY INDEXED-AREA句が記述されている場合、この句を注釈として扱います(翻訳エラーにはなりません)。

COBOL Gから移行したプログラムで、表示ファイルの画面帳票定義体で項目制御部の長さが3バイトの場合、翻訳オプションMODE(CBLG)を指定しないと、デバッガで以下の現象が発生します。

- 「登録集の展開表示」に“行う”を選択すると、デバッガのソースウインドウに表示される網かけ位置がずれて表示されます。

付録Q　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。

Q.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、ファイアウォールを配置したイントラネット環境内に保持してください。

イントラネット環境の外部に配置するWebサーバ上で、アプリケーションを使用する場合も同様です。データに関する重要な資源はイントラネット環境内に保持してください。なお、Webサーバ上に配置した、プログラムおよびプログラムの動作に必要な各種の情報ファイルについても、OSの機能によるアクセス制限を行い、不正なアクセスや改ざんから保護してください。

Q.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無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。

Q.3 インターネット接続対象外の通信機能

本製品は、通信および通信を行う他製品と連携する機能(表示ファイル、簡易アプリ間通信機能、Web連携機能)を提供しますが、Web連携機能を除く通信機能は、インターネットへのサービスを提供する用途のためには設計・製造されていません。インターネットに接続しない環境で使用するか、ファイアウォールを配置したイントラネット環境内でセキュリティ侵害対策を構築した上で使用してください。

Q.4 Web連携機能のセキュリティ

Webサーバの認証機構および暗号化(SSL)通信機能を使用して、不正なアクセスや情報の漏洩、改ざんなどの危険を回避してください。また、Webサーバのアクセスログを採取して、不正なアクセスに対する調査・追跡ができるようにしてください。

MeFt/Webを使用する場合、詳細はMeFt/Webのオンラインマニュアルを参照してください。Webサブルーチンを使用する場合、詳細はご使用になるWebサーバのドキュメントを参照してください。

Q.5 リモートデバッガ

本製品は、ネットワーク上の別のコンピュータで動作するプログラムをデバッグすることができるリモートデバッガのサーバ側機能を提供しますが、リモートデバッガは、インターネットで利用するためには設計・製造されていません。インターネットに接続しない環境で使用するか、ファイアウォールを配置したイントラネット環境内でセキュリティ侵害対策を構築した上で使用してください。

付録R　COBOLプログラムの作成技法

ここでは、効率の良いプログラムを作成するためのテクニックについて説明します。

R.1　効率のよいプログラム

ここでは、プログラムの実行時間を短縮するための細かい注意点を述べます。

プログラムは、効率が良いことの他に、読みやすく、拡張しやすいことも必要です。しかし、これから述べる項目の中には、読みやすさに逆行するものもあります。

プログラム作成時には、以下の知識を念頭において、プログラムを読みやすく作り、実行時命令統計機能を使ってボトルネックを発見し、ボトルネックになっている一連の文に対して再度効率向上のための修正を加える、という方法をおすすめします。

また、細かいコーディング技術を駆使するより、プログラムのアルゴリズムを検討する方が、効率を向上させる程度が大きいことがしばしばあります。細部の検討に入る前に、まず、アルゴリズムを改善できないかを考えることも重要です。

R.1.1　一般的な注意

作業場所節の項目

- レコードを設計する際は、よく使われる項目や関連のある項目を一か所に集めて定義するようにします。よく使われる項目が数キロバイト以上の大きな項目にはさまれるような設計は避けてください。
- 転記しても参照されないまま再転記されるような、不要な転記は避けてください。
例えば、集団項目全体に空白文字を転記した後で、改めて個々の項目に別の値を転記するのでなく、必要な項目だけに空白詰めを行ってください。
- 作業場所節にある項目で、プログラム実行中に値を変更する必要のないものは、VALUE句で初期値を設定してください。

ループの最適化

- ループの中では、特に、COBOLの文では見かけ上わからない添字計算や、データの属性の変換など、目的コードの生成を極力抑えるような配慮が必要です。
  - ループの中で実行しなくてもいい文はループの外に出してください。
  - ループの中では、データ名による添字付けを避け、指標名による添字付けを用いてください。
  - ループの中で数字転記や数字比較などに用いる項目は、ループの外であらかじめ最適な属性の項目に移してください。

複合条件の判定順序

- ANDだけ、またはORだけで結ばれた複合条件は、括弧がない限り左から右へ順に評価されます。以下の様に書くと、平均実行時間を短縮することができます。
  - ORで結ばれている場合は、真になりやすい条件を先に書く
  - ANDで結ばれている場合は、偽になりやすい条件を先に書く

R.1.2　データ項目の属性の選択

英数字項目と数字項目

- 英数字項目が使用できるところは、数字項目でなく、英数字項目を使ってください。
数字項目は、その中に入っている数値が意味を持っています。
例えば、PIC S9 DISPLAYの項目のビットパターンがX'39'でもX'49'でも、数値としては等しく、共に+9を示すものとみなされます。
このため、比較などの目的コードは、英数字項目に比べて遅くなります。

USAGE DISPLAYの数字項目(外部10進項目)

- 各文字位置には、文字"0"～"9"(16進表記でX”30”～X”39”)が入ります。最後の文字の先頭4ビットは符号を表し、数値が正の場合はX"4"、負の場合はX"5"が入ります。
- 印字表示用として使用してください。演算や比較に使用したときの処理速度は、数字項目の中で最も遅く、使用領域も最も大きくなります。

USAGE PACKED-DECIMALの数字項目(内部10進項目)

- 4ビットで1桁の数値を表し、最後の4ビットは符号を表します。数値が正の場合はX"C"、負の場合はX"D"、符号なしの場合はX”F”が入ります。
- 演算や比較に使用したときの処理速度は、外部10進項目よりは速く、2進項目よりは遅くなります。

USAGE BINARY/COMP/COMP-5の数字項目(2進項目)

- COMP-5の内部表現形式はシステムのエンディアンに従います。BINARYとCOMPは同義で、システムのエンディアンに従わず、常にビッグエンディアンの内部表現形式になります。
- 表示を目的としない演算、添字に適しています。演算や比較は外部10進項目および内部10進項目より速く、リトルエンディアン・システムではBINARYよりCOMP-5が速くなります。

数字項目の符号

- 数字項目には、その項目に値を転記する時に絶対値をとる必要がある場合を除き、PICTURE句でSを指定してください。符号をつける場合、SIGN LEADINGやSIGN SEPARATEは指定しないでください。
  - Sの指定がないと、転記する時に絶対値をとるための目的コードが生成されます。
  - SIGN LEADINGやSIGN SEPARATEの指定をすると、符号処理のための余分な命令が生成されます。

R.1.3 数字転記・数字比較・算術演算

属性

- 数字転記、数字比較、演算では、作用対象のUSAGE句を統一してください。
- 数字転記、数字比較、加減算では、作用対象の小数部桁数を一致させてください。乗除算では、中間結果の小数部桁数と受取り側項目の小数部桁数を一致させてください。
  - 一致していないと、演算や比較のたびに、一致させるための変換や桁合せのための目的コードが生成されます。
  - 乗算C=B*Aではdc=db+daを、除算C=B/Aではdc=db-daを満たすようにすると、桁合せは発生しません。(da,db,dcはそれぞれA,B,Cの小数部桁数を表します)
- 算術式では、属性が同じもの同士の演算が多くなるような順に、演算を行ってください。

桁数

- 桁数は、必要以上に大きくとらないでください。一般的に、演算や比較の時間は、桁数が多いほど長くなります。

べき乗の指数

- べき乗の指数は、整数の定数が最も適しています。次に適しているのは、整数項目です。
  整数でない指数が指定されると、COBOLランタイムシステムによる浮動小数点演算となるので、効率は極めて悪くなります。

ROUNDED指定

- ROUNDED指定の使用は、必要最小限にしてください。
  ROUNDED指定をすると、演算結果が1桁余分に求められ、正負の判定と四捨五入を行う目的コードが生成されます。

ON SIZE ERROR指定

- ON SIZE ERROR指定の使用は、必要最小限にしてください。
  ON SIZE ERROR指定すると、桁あふれを判定するために以下のような目的コードが生成されます。
  - 演算結果が2進で求まる場合:
    絶対値をとるなどして受取り側項目に収まる最大値との比較
  - 演算結果が内部10進で求まる場合:
    受取り側項目の文字位置を超える部分とゼロとの比較

TRUNCオプション

- TRUNCオプションの使用が必要最小限になるように、プログラムを設計してください。
- TRUNCオプションを指定した場合、2進項目間の転記における切り捨ては、10**n(nは受取り側項目の桁数)で除算し、剰余を求めて行っています。したがって、2進項目を多用するプログラムでは、TRUNCオプションを指定すると、大幅に効率が悪くなります。
- NOTRUNCオプションを指定する場合、受取り側項目に文字位置を超える値が入らないようにプログラムを設計しなければなりません。入力データによってそのような問題が起こる可能性がある場合は、不当な入力データを除外するプログラムに変更した上で、NOTRUNCオプションを指定してください。

R.1.4 英数字転記・英数字比較

境界合せ

- 機種によって異なりますが、英数字項目も左端を4バイトまたは8バイト境界に合わせると、一般に効率がよくなります。ただし、英数字項目に対して指定されたSYNCHRONIZED句は注釈とみなされるので、使用しない項目を定義して境界を合わせるようしてください。
  境界合せによって全体の使用領域は大きくなります。境界合せは、よく使われる項目を対象にしてください。

項目長

- 英数字転記では、送出し側項目長が受取り側項目長より大きいか等しい時、効率よく実行できます。英数字比較では、両方の項目長が等しい時、効率よく実行できます。
  一方が定数の時は、その長さを他方の項目長に合わせると、効率よく実行できます。
- 上記は、大きな項目(数百バイト以上)の場合、あてはまりません。

転記の統合

- 集団項目のすべての項目を転記する時は、項目ごとにMOVE文で転記せず、集団項目をMOVE文1つで転記してください。

R.1.5 入出力

SAME RECORD AREA句

- SAME RECORD AREA句は、2つ以上のファイルでレコード領域の内容を共有したい場合や、WRITE文の実行後もレコードを使用する必要がある場合に限って指定してください。
  SAME RECORD AREA句が指定された物理順ファイルのREAD文およびWRITE文は、レコード領域とバッファ領域の間でレコードの転送を行うため、効率が悪くなります。

ACCEPT文、DISPLAY文

- ACCEPT文(DATE、DAY、TIME指定を除く)およびDISPLAY文は、少量のデータの入出力のみに用いてください。
  これらの文は、READ文およびWRITE文よりも一般的に効率が悪くなります。

OPEN文、CLOSE文

- OPEN文およびCLOSE文は、非常に複雑な内部処理を伴う文であるため、1つのファイルに対するOPEN文およびCLOSE文の実行回数は、必要最小限に抑えてください。

R.1.6 プログラム間連絡

副プログラムの分割の基準

- 1つのシステムを多数のプログラムから構成する場合は、必要以上に小さい副プログラムに分割しないことをおすすめします。
  - 副プログラムの呼出しから復帰までに、静的構造の場合でも最低数10ステップの機械文が実行されます。したがって,小さい副プログラムでは、このステップ数が相対的に大きな比重を占めることになり、効率を悪化させてしまいます。副プログラムの手続き部が数百行以上あれば、効率は悪化しません。
  - 目的プログラムは初期化・終了ルーチンや作業領域などを必ず持っているので、小さい副プログラムに分割すると全体の領域が多く必要になります。

動的プログラム構造と動的リンク構造

- 動的プログラム構造(CALL一意名、またはDLOADオプションを指定して翻訳したCALL定数を用いるプログラム構造)は、非常に大きなシステムで、仮想記憶を節約するために副プログラムを削除する必要のある場合以外は使用しないでください。
  動的リンク構造で済むときは、動的リンク構造を使うことをおすすめします。
  - 動的プログラム構造では、副プログラムがローディングされた後も、副プログラムが既にローディングされているかどうかを調べるサーチ処理や、プログラム名のチェックが、呼出しのたびに行われます。そのため、オーバヘッドは、静的プログラム構造より大きくなってしまいます。
  - 動的リンク構造では、副プログラムがローディングされた後は、呼出しは直接行われます。そのため、オーバヘッドは、静的リンク構造の場合よりわずかに多い程度です。

CANCEL文

- 動的プログラム構造を使う場合、CANCEL文は必要最小限に抑えてください。

パラメタの個数

- CALL文のUSING指定、およびENTRY文またはPROCEDURE DIVISIONのUSING指定にパラメタを記述すると、個々のパラメタについてアドレスの設定が行われます。したがって、パラメタは可能な限り集団項目にまとめ、USING指定での個数を少なくする方が、効率がよくなります。

R.1.7 デバッグ

- CHECK,COUNT,TRACEオプションを使用したデバッグを完了した後は、これらのオプションを取り除いて翻訳してください。
- CHECKオプションの指定によって、実行時間が2倍以上遅くなることがあります。運用時にCHECK(NUMERIC)を有効にする場合は、可能な限り10進項目を2進項目に変更すると、CHECKオプションによる性能劣化を防ぐことができます。

R.2 数字項目の標準規則

ここでは、COBOLプログラムで数字データを扱う上での標準的な規則を述べます。

R.2.1 10進項目

10進項目の入力

- 入力レコード中に10進項目が含まれている場合、誤った内容表現のデータが入らないように注意してください。
  正しいビットパターンが入っているかどうかを確かめるには、字類条件(IF … IS NUMERIC)を使います。コンパイラは、READ文の実行時に、10進項目のビットパターンを検査しません。
- 10進項目を含む集団項目へCORRESPONDING指定のない転記を行う場合、誤ったビットパターンが入らないよう注意してください。
  この場合も、コンパイラは検査を行いません。

10進項目に誤ったビットパターンが入った場合

- 外部10進項目のゾーン部(符号を持つバイトを除く)が16進数の3でない場合、誤りです。

- 10進項目の数字部が許されるビットパターン(16進数の0～9)でない場合、この項目を転記、演算または比較などに使用すると、結果は規定されません。

誤ったプログラム例

英数字項目から数字項目の転記は数字転記になり、英数字項目を符号なし10進項目にみなして転記します。よって、(a)のSND-DATAはPIC 9(4)とみなし翻訳されます。空白が入っている場合などの不正な値は誤りとなり、結果は予測できません。(a)のMOVE文が予測できないため、(b)の比較結果も予測できません。

```
01 SND-DATA PIC X(4).
01 RSV-DATA PIC 9(4).
...
   MOVE  SPACE  TO  SND-DATA
   ...
   MOVE  SND-DATA  TO  RSV-DATA  …(a)
   IF  RSV-DATA  =  SPACE  THEN  …(b)
```

SND-DATA					RSV-DATA			
20	20	20	20	転記 →	xx	xx	xx	xx

正しいプログラム例

10進項目に不正な値が設定される可能性がある場合は、(c)のように字類条件(IS NUMERIC)を使用し、正しい値が格納されていることを確認してから使用します。

```
01 SND-DATA PIC X(4).
01 RSV-DATA PIC 9(4).
...
   MOVE  SPACE  TO  SND-DATA
   MOVE  0      TO  RSV-DATA
   ...
   IF SND-DATA IS NUMERIC THEN     …(c)
     MOVE  SND-DATA  TO  RSV-DATA
   ELSE
     DISPLAY NC"データ異常" SND-DATA
   END-IF
```

R.2.2 2進項目

2進項目の値の範囲

2進項目は、一般に、PICTURE句で示された値の範囲より大きい絶対値をもつ値を含むことができます。
NOTRUNC指定の場合、2進項目への転記の際にPICTURE句に合わせた切り捨てが行われなかった結果、正の値が負の値になることもあります。
[参照]"A.2.47 TRUNC(桁落とし処理の可否)"

2進項目のけた数の扱い

2進項目は、PICTURE句より大きい値を含むことができますが、これをDISPLAY文で参照した時は、PICTURE句で示された桁数だけ表示されます。
ON SIZE ERROR指定、またはNOT ON SIZE ERROR指定が記述された演算文では、PICTURE句の指定を超えた値を格納しようとしているかどうかの判定が行われます。
一般に、コンパイラは、2進項目の値はPICTURE句で示された範囲内にあることを前提としてコンパイルを行うため、PICTURE句より大きい値を持つ2進項目を演算などに使用すると、場合によっては異常終了を起こすことがあります。

R.2.3 浮動小数点項目

固定小数点への変換

演算の結果が浮動小数点で求まり、これを固定小数点項目に格納する場合、変換の誤差が最小になるように格納されます。同様に、浮動小数点項目から固定小数点項目へ転記する場合も、変換の誤差が最小になるような値が格納されます。

R.2.4 数字項目の注意事項

乗除算の混合時の小数部桁数

次のプログラムの[1]と[2]を比較します。

```
77  X  PIC  S99  VALUE 10.
77  Y  PIC  S9   VALUE 3.
77  Z  PIC  S999V99.
    COMPUTE  Z = X / Y * 100.    [1]
    COMPUTE  Z = X * 100 / Y.    [2]
```

[1]の答はZ=333.00、[2]の答はZ=333.33となります。
この違いは、除算を行うタイミングによって発生します。
どちらも、除算の結果はXとZの小数部桁数の大きい方、すなわち小数第2位まで求められます。[1]では、X/Yが3.33と求められ、これが100倍されます。[2]ではX*100の中間結果である1000がYで割られ、333.33が求まります。
よって精度のよい結果を求めるには、[2]のように、乗算を先に除算を後に行ってください。

絶対値がとられる転記

- 受取り側項目が符号なし数字項目である転記の場合、送出し側項目の絶対値が、受取り側項目に格納されます。
- 符号付き数字項目から英数字項目への転記の場合、送出し側項目の絶対値が、受取り側項目に格納されます。

R.3 注意事項

外部ブール項目のビットパターン

- 外部ブール項目の内容は、16進数で30または31です。それ以外は許されません。
- 0～6ビットに不適当な値を持つ外部ブール項目を比較や演算に使用したときの結果は、規定されません。
- 不適当な値を持つ可能性がある場合は、外部ブール項目を字類条件により検査してください。

レコード領域の参照

- OPEN文実行前、またはCLOSE文実行後のファイルのレコード領域を参照してはいけません。

索 引

[]

~SRTnnnn....641

[数字]

10進項目....769
16進表現のデータ....635
16ビットワイドキャラクタ....714
1行の形式....7
1バイト文字....178
2進項目....770
2進項目の扱い....660
2バイト文字....178

[記号]

#....574
*COB-BINDTABLE....379
*COB-BINDTABLEクラスのオブジェクトメソッド....381
*COB-BINDTABLEクラスのファクトリメソッド....381
*節名....594
*段落名....594
-a....559,583
-b....558
-c....36,559,583
-Dc....36
-dd....38
-Dk....36
-dn....41
-do....38
-dp....38
-Dr....37
-dr....37
-Dt....37
-Dtオプション....554
-dy....41
-e....559
-f....39
-G....41
-h....559,583
-I....39
-i....39
-j....559
-k....559
-L....41
-l....42,558
-M....39
-m....39,559,582
-n....581
-Ns....42
-o....42,559,578
-P....40
-p....558,583
-pc....42
-pi....42
-pm....42
-pオプション....620
-q....558
-R....40
-r....559
-s....558,583
-t....583
-Tm....40,42
-Tオプション....194
-u....583
-v....560,580
-V....583
-w....560
-WC....40
-Wl....42
-x....583
-yオプション....194
@OPTIONS....9
_FINALIZEメソッド....351

[A]

A3....184
A4....184
A5....184
ACCEPT文....263,439,619
ACCEPT文のデータの入力先....678
ACCEPT文の動作....671
ACCEPT文のファイル入力拡張機能....278
ACCESS MODE句....110,117
ACM....53
ADDR関数....296,297
AFTER指定....189
ALPHAL....590,660
ALTERNATE RECORD KEY句....118
ankfont制御文....167
ANSI COBOL規格....669
ANY LENGTH句....387
a.out....16
APL....53
APOST....675
ASSIGN句....127,128,129,163
AT END指定....124
AUTORUNサブコマンド....597
A領域....7

[B]

B....184
B4....184
B5....184
BACKTRサブコマンド....597
BEFORE指定....189
BINARY....660,683
BIND....185
BREAKサブコマンド....598
BSAM....141
BSORT_TMPDIR....641
BY REFERENCE指定とBY CONTENT指定の違い....249
BY REFERENCE指定とBY VALUE指定の違い....250
B領域....7

[C]

c.....58
C.....183
C++.....303
C++で定義されているクラスを調べる.....368
C++連携の概要.....364
C++連携のプログラム手順.....367
C++連携の方法.....364
CALLSサブコマンド.....599
CANCEL文.....239,436
cbl.....16
CBR_CI_CLG.....491
CBR_CI_INF.....488
CBR_CLASSINFFILE.....414
CBR_CLOSE_SYNC.....144
CBR_CODE_CHECK.....738
CBR_COMPOSER_CONSOLE.....273
CBR_COMPOSER_MESS.....62
CBR_COMPOSER_SYSERR.....273
CBR_COMPOSER_SYSOUT.....273
CBR_CSV_OVERFLOW_MESSAGE.....656
CBR_CSV_TYPE.....656
CBR_FCB_NAME.....162,204
CBR_INPUT_BUFFERING.....140
CBR_INSTANCEBLOCK.....414
CBR_LP_OPTION.....160
CBR_MEMORY_CHECK.....88
CBR_MESS_LEVEL_CONSOLE.....60
CBR_MESS_LEVEL_SYSLOG.....62
CBR_MESSOUTFILE.....61
CBR_PRINTFONTTABLE.....161,173
CBR_PRT_INF.....161,164
CBR_PRT_UTF8_CONVERT.....162
CBR_PSFILE_DSP.....228
CBR_PSFILE_PRT.....210
CBR_SYMFOWARE_THREAD.....461
CBR_THREAD_TIMEOUT.....461
CBR_TRAILING_BLANK_RECORD.....106,145
CBR_CBRFILE.....50
CBR_CBRINFO.....52
CBR_CONSOLE.....266
CBR_ENTRYFILE.....54
CBR_JOBDATE.....281
CBR_SSIN_FILE.....279
CBR_SYMFOWARE_THREAD.....443
CBR_TRACE_FILE.....73
CHARACTER TYPE句.....156,188
CHECK.....661
CHECK機能.....36,58,70,75,464,661
CHECK機能の使用例.....78
CIM.....136,148
C-ISAM.....136
C-ISAMでの注意点.....137
C-ISAMの指定.....136
C-ISAMを使うプログラムをリンクする指定.....42
CLASS-ID.....327
CLOSE文.....189
cob.....15
cobactldコマンド.....494
cobci.....487
COBCI_CLOSE関数.....498
COBCI_OPEN関数.....497
COBCI_READ関数.....499
COBCI_WRITE関数.....500
COBCOPY.....10,13
COB_COPYNAME.....10,13
cobcrtldコマンド.....494
cobdacldコマンド.....494
cobdebug.ini.....556
cobdepend.....753
cobdltldコマンド.....494
cobdspldコマンド.....495
cobfattr.....644
cobfbrws.....639
cobfconv.....635
cobfload.....637
cobfrcov.....644
cobfreog.....645
cobfsort.....641
cobfulod.....638
cobfuty.....755
COB_GET_PROCESSID.....710
COB_GET_THREADID.....711
COB_LIBSUFFIX.....11,17
COB_LOCK_DATA.....466
COB_LOCK_OBJECT.....467,468
cobmkmf.....753,755
cobol.....755
COBOL16ビットワイドキャラクタ.....714,716
COBOL.CBR.....49
COBOL G移行支援オプション.....763
COBOLOPTS.....10,35
COBOL Webサブルーチン.....710
COBOLアプリケーション.....2
COBOLおよびC++でのクラスの対応.....365
COBOLが提供するサブルーチン.....710
cobolコマンド.....34,554
cobolコマンドの復帰値.....43
COBOLコンパイラ.....2
COBOLソースプログラム.....7
COBOLでの資源の共有.....431
COBOLでの使用方法.....714,716
COBOLの機能.....1
COBOLの言語間の環境.....234
COBOLの実行単位.....234
COBOLの主プログラム.....234
COBOLのデータ項目とCのデータ型との対応例.....255
COBOLファイル.....626
COBOLファイルアクセスルーチン.....145
COBOLファイルユーティリティ.....2
COBOLプログラムからCOBOLプログラムを呼び出す.....241
COBOLプログラムからCプログラムを呼び出す.....248
COBOLプログラムの作成技法.....766
COBOLランタイムシステム.....2

cobpurldコマンド......495
COB_REPIN......11
cobstpciコマンド......493
cobstplgコマンド......496
cobstrciコマンド......493
cobstrlgコマンド......495
COB_UNLOCK_DATA......466,467
COB_UNLOCK_OBJECT......467,468
CODECHK......662
COLUMNS......233
COMMON......247
CONF......663,683
CONTINUEサブコマンド......599
COPY......663
COPY指定子......594
COPY修飾値......594
COPY文......618
CORBA......506
CORBAアプリケーション......506
COUNT......83,664
COUNT機能......36,70,82,83,464,664
COUNT機能の使い方......82
COUNT機能を使用したプログラムのデバッグ......87
COUNTサブコマンド......599
COUNT情報......83
COUNT情報の出力形式......84
CREATE......664
CSV形式データ操作時のメッセージ抑止......656
CSV形式データとは......651
CSV形式データの作成......652
CSV形式データの操作......651
CSV形式データの分解......653
CSV形式のバリエーション......655
CTL......192
CURRENCY......664
CURRENT-DATE関数......694
cvr......560
C言語で記述したプログラムから呼び出されるプログラムの指定42
C言語での使用方法......715,717
C言語プログラムとのデータの共用......256
C言語プログラムとのリンク......248
Cプログラム......248,451
CプログラムからCOBOLプログラムを呼び出す......252

[D]

D......185
DATACHKサブコマンド......601
DELCOUNTサブコマンド......601
DELDCHKサブコマンド......602
DELDTRサブコマンド......603
DELETEサブコマンド......603
DELMONサブコマンド......604
DISK......129
DISPLAY文......263,439
DISPLAY文のデータの出力先......679
DISPLAY文のファイル出力拡張機能......276
DLOAD......665
documentname制御文......169
DSP......53
DTRACEサブコマンド......604
DUPCHAR......665
DUPLICATES......118
DYNAMIC......110,117

[E]

EDIT-COLOR......178
EDIT-MODE......178
EDIT-OPTION......178
EDIT-OPTION2......178
EDIT-OPTION3......178
EDSAC......300
ENTRY文......242
ENVサブコマンド......605
EQUALS......666
EUC......714,716,736
EXIT PROGRAM文......242
EXTERNAL句......256

[F]

F......184
FCB......169,183
FCBDIR......162
fcbname制御文......167
FCB名......183
ffd......16,556
FFD_SUFFIX......11,17,559
FILELIB......11
FILE STATUS句......125,684
FJBASEクラス......328
FLAG......666
FLAGSW......666,683
FONT-nnn......173
FORM......173
FORMAT-ID......183
FORMAT句付き印刷ファイル......153,155
FORMAT句なし印刷ファイル......153
FORMAT句なし印刷ファイル(行単位のデータを印刷する)155
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)......155
FORMLIB......11
FORMオーバレイオプション......172
FORTRAN......300
FOVL......183
FOVLDIR......163
FOVL-n......185

[G]

GLOBAL句......248
GOPT......58

[H]

HELPサブコマンド......607
HOPPER......184

[I]
iconv.....755
IN.....594
INDEXED.....117
INHERITS句.....327
INITIAL.....247
INITVALUE.....667
INITメソッド.....351
IN/OFありのCOPY文.....13
IN/OFなしのCOPY文.....13
insdbinf.....729
insdbinfの使用方法.....729
INSTANCEBLOCKセクション.....415
Interstage Business Application Server.....62,272
Interstageなどのサーバ環境で動作するプログラムのデバッグ586
INVALID KEY指定.....125
INVOKE文.....320,322
INオペランド.....593
is_a関係.....309
I/S制御レコード.....176
I制御レコード.....182

[J]
JMPCINT2.....234,253,711
JMPCINT3.....234,253,712
JMPCINT4.....419,713

[K]
KANA.....667
KANA(JIS8).....619
KOL6.....154,195

[L]
L.....183,185
LALIGN.....668
LANG.....736
LANGLVL.....669,683
LBSAM.....138,141
LD_LIBRARY_PATH.....11,164,210
ldコマンド.....746
ldコマンドの使い方.....747
LENG関数.....296,297
LFS.....138
libcobcicl.so.....504
librcobcicl.so.....504
LINECOUNT.....669
LINES.....233
LINE SEQUENTIAL.....105
LINESIZE.....670
LINKAGEサブコマンド.....607
LIST.....670
ListWorks.....211
LISTサブコマンド.....608
long int型.....251,252,254
LP.....183
lpi制御文.....170
LSEARCHサブコマンド.....609
lst.....16
LTR.....184
LZ.....183

[M]
MAIN.....670,683
Makefile作成支援コマンド.....753,755
Makefileの記述方法.....750
makeコマンドについて.....750
makeコマンドの活用.....750
MAP.....22,671
mbston16s.....714
MeFt.....173
MEFTNET.....210
MeFt/NET-SV.....210,228
MeFtを使用する場合.....210
MERGE文.....288
MESSAGE.....671
MGPRM.....60
MODE.....671,763
MONITORサブコマンド.....609

[N]
N.....185
n16stombs.....716
NAME.....671
NCW.....672
NEW.....351
NEWメソッド.....381
NOALPHAL.....590,619,660
noc.....58
nocb.....58
NOCHECK.....661
noci.....58
nocn.....58
NOCODECHK.....663,738
NOCONF.....663
NOCOPY.....663
NOCOUNT.....664
nocp.....58
NODLOAD.....665
NOEQUALS.....666
NOFLAGSW.....667
NOLALIGN.....668
NOLIST.....670
NOMAIN.....670
NOMESSAGE.....671
NONAME.....672
NONUMBER.....593,674
NOOBJECT.....674
NOOPTIMIZE.....674
nor.....58
NOSDS.....676
NOSHREXT.....677
NOSOURCE.....677
NOTEST.....680
NOTRACE.....681
NOTRUNC.....681
NOXREF.....682

NOZWB.....682
NSPCOMP.....672
NUMBER.....593,673

[O]

o.....15
OBJECT.....674
OF.....594
OFFSEARCHサブコマンド.....609
OFFSET.....185
OPEN-DATA-FILE.....376
OPEN文.....189
OPTIMIZE.....674
OPTIONAL.....107,114
Oracle連携.....729
Oracle連携時の注意事項.....731,733
oratool.....731
oratoolの使用方法.....730
ORGANIZATION句.....101,105,110,117
OSIV系システムと本システムの機能比較.....757
OVERRIDE句.....330

[P]

P.....183,184
P1.....184
P2.....184
P3.....184
P4.....184
PAGE.....189
papersize制御文.....166
PATH.....11
PERFORM文の最適化.....689
PGMNAME.....763
pmd.....16,556
pmgr.....508,755
pmgrbuild.....541,755
pmgrcbi.....547,755
pmgr_chklnk.....462
pmgr_chsuffix.....753
pmgredit.....525,755
pmgr_getsym.....753
pmgrgrep.....535,755
pmgrlni.....549,755
pmgrmsg.....530,755
pmgr_nonsuffix.....753
pmgr_rename.....753,755
pmgr_vrep.....755
PowerSORTが使用するメモリ容量.....59
PowerSORTが使用するメモリ容量を指定.....677
PRINTER-n.....163
printer制御文.....165
PRINTサブコマンド.....610
print制御文.....171
PROGRAM-STATUS.....246,251,254
PROGサブコマンド.....611
PROPERTY句.....349
PROTOTYPEメソッド.....344
PRT.....53
PRT-AREA.....185
PRT-FORM.....183
prtform制御文.....166
prtout制御文.....168
pxd.....16,556
PZ.....183

[Q]

QUITサブコマンド.....611
QUOTE.....675

[R]

r.....58
R.....183,185
RAISE文の動作.....362
RAISING指定のEXIT文の動作.....362
RANDOM.....110,117
RDMファイル.....139
RECORD　KEY句.....118
RECORD句.....101
REGISTER-INDICATORメソッド.....382
REGISTER-LARGE-DATA-INDICATORメソッド.....385
REGISTER-LARGE-DATAメソッド.....384
REGISTER-VARYING-INDICATORメソッド.....384
REGISTER-VARYINGメソッド.....383
REGISTERメソッド.....382
RELATIVE.....110
RELATIVE　KEY句.....111
rep.....16
REPLACE文.....618
RERUNサブコマンド.....611
Retrieve.....376
RETURNING指定.....245,251,253,323
RSV.....185,675
RUNTOサブコマンド.....612

[S]

s.....59
S.....184
SAI.....676
Save.....377
SCCSの利用方法.....744
SCOPEサブコマンド.....593,614
SCREEN-DESIGNER.....2,621
sdegenerate.....625
S-DESIGN.....621
sdesign.....621,755
S-DESIGNの機能.....621
S-DESIGNの操作概要.....621
sdestou.....625
sdeutos.....625
SDS.....676
SD機能.....296
SD機能の種類.....296
SEARCHサブコマンド.....614
SELECT句およびASSIGN句の記述例.....101
SEQUENTIAL.....101,110,117
SETサブコマンド.....615

SHREXT......676
SIDE......184
SIZE......184
SKIPサブコマンド......615
Smalltalk......303
smd......16,556
SMED_SUFFIX......11,17,559
smsize......59
SMSIZE......677
so......16
SORT-STATUS......291,294
SORT文......288
SORT文での同一キーデータの処理方法......666
SOURCE......677
SRF......678
SSIN......264,678
SSOUT......264,679
STAMPサブコマンド......615
STATUSサブコマンド......616
STD1......679
STOP RUN文......242
streamenv制御文......168
stream制御文......168
STRING文......652
svd......16,580,755
svdcover......580
svdsmpls......579
svdsmpmg......578
svdコマンド......557
SWITCH-0......59
SWITCH-7......59
SWITCH-8......59
Symfoware連携......443,731
Symfoware連携時の注意事項......734
SYSCOUNT......83
Syslog......62
Systemwalker Centric Manager......266
Systemwalkerのコンソール......266
S制御レコード......182

[T]

TAB......679
TAB文字......8,619
TCP/IP......486
TEST......554,680
TESTオプション......553
THREAD......680
TRACE......681
TRACE機能......37,58,70,71,463,681
TRUNC......681

[U]

U......185
UCS-2......470
Unicode......470,736
UNIXコマンド連携......617
UNIX版COBOLアプリケーション......754
UNSTRING文......653
USAGE IS DISPLAY......153
USAGE OBJECT REFERENCE句......320
USING指定......322
USING指定の記述の違い......249
UTF-16......470
utf8_convert制御文......169
UVPIデータのテキスト変換......631

[V]

void型......252

[W]

Web連携......505,710
WHEREサブコマンド......617
WIDTH......185
Windows......754
WRITE文......189

[X]

XMDLIB......225
XREF......682

[Z]

ZWB......682

[あ]

アクセス形態......99,110,117
誤り処理手続き......126
暗黙プログラム修飾......593
アンロード......638
異常終了時の障害発生箇所の特定方法......90
依存リポジトリファイル......400
一意名......595
位置オペランド......590
一次入口......242
一連番号領域......7
一般的な注意......766
印刷機能......439
印刷形式......183
印刷原点位置......185
印刷情報ファイル......164
印刷処理......153
印刷装置......155
印刷ファイル......97,155,179,187,439
印刷ファイルの定義......188
印刷方法の種類......153
印刷方法の特徴・利点・用途......153
印字文字......156
印字文字の大きさ......157
印字文字の間隔......159
印字文字の形態......158
印字文字の書体......157
印字文字のスタイル......158
印字文字の方向......158
インタフェース......331
インタフェースプログラムの構造......366
インタフェースプログラムの仕組み......365
ウィンドウ情報ファイル......228

上とじ....185
埋込みSQL文のデバッグ....733
埋込みSQL文のデバッグまでの流れ....731
英小文字の扱い....660
英小文字を記述したプログラム....619
英数字転記・英数字比較....768
英数字の文字の大小順序....679
永続オブジェクト....372
永続オブジェクトの流れ....372
エディタ....524
エディタ行番号....593,594
エラー検出時の処理実行回数....58
エラーコード....469
エントリ情報....34,54,238,408
エントリ情報の記述形式....408
エントリ情報ファイル....238
同じ親クラスを持つクラスを同じファイルに保存する....375
オブジェクト....303,448
オブジェクトインスタンス....425,434
オブジェクトインスタンスの獲得方法....414
オブジェクトインスタンスの格納数....415
オブジェクトインスタンスの作成・参照....304
オブジェクトインスタンスの寿命....324
オブジェクトインスタンスの操作....319
オブジェクトインスタンスのブロック化....411
オブジェクト削除インタフェースプログラム....369
オブジェクト参照項目....320,333
オブジェクト参照修飾子....595
オブジェクト指向....302
オブジェクト指向と従来機能の組合せ....727
オブジェクト指向に対応したデバッグ機能....569
オブジェクト指向のメリット....311
オブジェクト指向の歴史的背景....300
オブジェクト指向プログラミング....300,303
オブジェクト指向プログラミング機能....312,361,442
オブジェクト指向プログラミングで使用する資源....389
オブジェクト指向プログラミングで使用するファイル....389
オブジェクト指向プログラミングの開発と実行....389
オブジェクト指定子....347
オブジェクト生成インタフェースプログラム....368
オブジェクト定義....315,595
オブジェクトの永続化....372
オブジェクトの寿命....324
オブジェクトの保存/復元....376
オブジェクトファイル....15
オブジェクトファイルの指定....42
オブジェクトファイルのディレクトリ....38
オブジェクトプロパティ....349
オブジェクトメソッド....315,317,368,377
オブジェクトロックサブルーチン....467
オプション....581
オプション情報リスト....18,671
オプションファイル....16
オプションファイルの指定....39
オペランド....590
親クラス....306,327
親クラスのオブジェクトデータも含めて一つのファイルに保存する....374
オーバレイパターン名....185

[か]

改行文字....7
開始プログラム名....555
開発環境....3
開発手順....389
外部スイッチの値....59
外部データ....244,429,431
外部表現形式....738
外部ファイル....244,429,431
外部ファイルハンドラ....148
外部プログラム....246
外部プログラム呼出しのパラメタの検査....81
外部名を指定する....709
概要....1
各印刷方法の概要....153
拡張....103,106,113,121,130
拡張子trc....73
拡張子tro....73
仮想メモリ不足....69
片面印刷....184
カバレージ機能....580
カバレージコマンド....581
カバレージコマンド使用例....583
カバレージ統計情報....580
カバレージファイル....580
カバレージレベル....560
カプセル化....302
可変形式....8
可変長レコード形式....99,101
画面項目....229
画面情報ファイルのコード変換....625
画面設計....621,622
画面設計域の操作方法....622
画面設計エディタ....621
画面節ソースの修正....624
画面節ソース表示ダイアログ....624
画面帳票定義体....8,556
画面帳票定義体拡張子....556
画面帳票定義体ファイル....16
画面帳票定義体ファイルの格納ディレクトリ....11
画面帳票定義体ファイルのディレクトリ....39
画面帳票定義体ファイル名の拡張子....11
画面定義情報ファイル....621
画面定義体の作成....223
画面定義体を使うプログラムをリンクする指定....42
画面入出力....221
画面入出力状態の設定値....231
画面入力....564
画面の構成....561
画面を使った入出力....221
仮リポジトリ....359
簡易アプリ間通信....2,485
簡易アプリ間通信機能....442,482,710

環境変数..10,44,52,414,442
環境変数一覧..700
環境変数設定..557
環境変数設定コマンドで設定..49
環境変数の指定..587,588
環境変数の設定..160
環境変数の操作機能..285
関数..496,692
関数一意名..692
関数値..251,253,452
関数のエラーコード..502
間接参照クラス..353
関連ウィンドウ..563
関連コマンド..625
機械語..300
規格の違いによるメッセージの出力..663
記号定数..596
記述形式..590
起動..621,632
起動方法..452
機能..1
機能選択..632
機能比較..757
機能分類..574
基本ブロック..688
逆トレース中のコマンド操作..619
競合状態..430
行順ファイル..96,104
行順ファイルの後置空白に関する指定..145
行順ファイルの処理..105
行順ファイルの定義..105
行順ファイルのレコードの定義..105
強制クローズ..130
行単位のデータを印刷する方法..187
共通オペランド..591
共通式の除去..688
共通プログラム..247
行内識別番号..593,595
行内呼出し..346
行番号..71,593
行番号情報埋込みツール..729
行番号内動詞追番..71
共用オブジェクトファイル..16
共用オブジェクトファイルの構成とファイル名..407
共用オブジェクトプログラムを生成する指定..41
行レコード..192
キー情報..644
キー定義ファイル..230
キー定義ファイルの記述形式..230
キーボード入力..565
キーワードオペランド..590
組込み関数..692,698
クライアント..486
クライアントでの指定..491
クライアントの処理..489
クラス..303,368
クラスごとに保存ファイルを分ける..373
クラス情報..414
クラス情報ファイル..414
クラス定義..313
クラス定義で使用できない機能..727
クラスとファイルの対応..373
クラスとメソッドのエントリ情報..408
クラスのエントリ情報..408
クラスの公開..411
クラスの設計..390
クラスの動的プログラム構造..394
クラス名.rep..335
クラス名段落..327
クラスライブラリ連携..379
クラスを動的プログラム構造にする..401
クリア..576
クロスサイトスクリプティング..764
クローズ時の書込み内容の即時反映..144
継承..306,325,326
継承の概念..326
継承の実現..335
継承の定義方法..327
桁落とし処理..681
結合の種類..29
結合モードの指定..41
言語要素に対しての指摘メッセージ..666
現在の日付および時刻の入力..279
原始プログラム..556
原文名定数..709
広域最適化..674,688
更新..104,114,122,130
更新回数..576
更新項目..229
構造化定理..300
構造化プログラミング..300
構文表記記号..590
構文表記法..590
項目情報設定ダイアログ..623
効率のよいプログラム..766
刻印文..582
刻印文指定..582
子クラス..306,327
固定形式..8
固定長レコード形式..98,101
コマンド..493
コマンド行で設定..52
コマンド行引数..442
コマンド行引数の取出し..283
コマンドファイル..555
コマンドモード..634
コマンドモードの使い方..634
コマンドライン..555
混在チェック..461
コンパイラが出力する情報..17
コンパイラが使用するファイル..15
コード系..736
コード系一致チェックを迂回する方法..737
コード系の一致チェック..737

[さ]

再帰呼出しレベル....597
最適化の項目....688
再編成....645
作業手順....206
索引ファイル....98,116
索引ファイル操作クラス....376
索引ファイルとオブジェクトの対応....373
索引ファイルの再編成....631
索引ファイルの処理....119
索引ファイルの属性情報....644
索引ファイルの属性表示....632
索引ファイルの定義....116
索引ファイルの復旧....631
索引ファイルのレコードの定義....118
削除....115,123,130,325,576
サブコマンド....564,597,619
サブコマンド機能....589
サブコマンドの入力....564
サブコマンド名....590
サブプログラムを呼び出す....234
参照....103,107,113,121,130
サーバ....486
サーバでの指定....491
サーバとの接続....489
サーバとの切断....489
サーバの起動....489
サーバの停止....489
サービス名....487
シェルの初期化ファイルに設定....49
式....596
識別名....595
資源一覧....5
資源の共有....430
システムの使用する領域....185
システムの標準エラー出力....273
システムの標準入出力....264
システムプログラム記述向け機能....296
システムプログラムを記述するための機能....296
シスログを使うプログラム....281
下とじ....185
実行可能ファイル....16
実行可能プログラム....28,32,33,34
実行可能プログラムの構造....28
実行環境....44,419
実行環境の設定....44
実行環境の設定方法....48
実行環境の閉鎖サブルーチン....713
実行環境変数....461
実行経路をたどる....566
実行時オプション....58
実行時チェック....461
実行時に有効な環境変数....44
実行時のコード系....736
実行時のコード系チェック....662
実行時の制御の流れ....367
実行時の注意事項....411
実行時の適合チェック....334
実行時メッセージのInterstage Business Application Serverの汎用ログへの出力....62
実行時メッセージのSyslog出力....62
実行時メッセージの重大度指定....60
実行時メッセージのファイル出力....61
実行順序を変更する....566
実行済み行の表示....574
実行性能の向上....413
実行操作....56
実行単位....419
実行単位の開始サブルーチン....711
実行単位の終了サブルーチン....712
実行に必要となるエントリ情報....409
実行用の初期化ファイル....49
実行用の初期化ファイルに設定....49
実行用の初期化ファイル名の指定方法....59
指定方法....574
自動化....567
自動デバッグ....568
指標名....595
シフトJIS....736
自由形式....9
集計範囲指定....581
終端記号....590
終了キー....230
終了条件なしのPERFORM文....296,298
終了処理メソッド....351
終了ステータス....63
主供給口1....184
主供給口2....184
主供給口3....184
主供給口4....184
主キー....118
縮小印刷のポートレートモード....183
縮小印刷のランドスケープモード....183
出力項目....229
出力ファイル....290
出力メッセージ....76
主プログラム....14,29,39,670
主レコードキー....99,118
順呼出し....99,110,117
障害発生箇所の特定方法....91
条件式....596
条件名....595
使用するクラスの選定....390
状態....576,577
状態変更....576,577
小入出力....263
小入出力機能....439
情報隠蔽....302
情報ファイル....52
使用メモリの節約....412
初期化処理メソッド....351
初期化ファイル....461
初期化プログラム....247
処理種別の種類....484

処理モード.....552
字類条件.....471
診断メッセージのレベル.....666
診断メッセージリスト.....18
進捗率.....575
数字項目の注意事項.....771
数字項目の標準規則.....769
数字転記・数字比較・算術演算.....767
数字データの入力.....265,272
スクリーン操作機能.....221,229,439,621
スクリーン操作機能を使うプログラムをリンクする指定.....42
スクリーン名.....624
スクリーン名の指定.....624
スクリーンモード.....552,554,632
スクリーンモードの使い方.....632
スタックオーバーフロー.....64,411
スタックサイズの求め方.....64
ステートメント番号.....71
図表レコード.....195
スレッド.....416,451
スレッドID取得サブルーチン.....711
スレッド間共有外部ファイル.....444
スレッド単位で確保/管理されるデータ.....423
スレッド同期制御サブルーチン.....465
スレッドの同期制御.....416
スーパークラス指定子.....595
制御の復帰とプログラムの終了.....242
制御レコード.....190,192
整形された符号.....676
正書法.....8
正書法の種類.....678
生成するCSV形式のバリエーションの指定.....656
静的結合.....14,29
静的構造.....392,393
静的束縛.....338
性能向上.....140
西暦の取得.....694
整列.....288,641
整列併合機能.....288
整列併合用ファイル.....290
整列用作業ファイル.....641
セキュリティ.....764
セクションサイズリスト.....28,671
接続製品.....52
接続製品識別名.....53
節名.....594
前回テスト実施日.....575
全クリア.....576
全削除.....576
相互参照クラス.....355
相互参照クラスの実行.....360
相互参照クラスの翻訳.....358
相互参照クラスのリンク.....359
相互参照パターン.....355
相互参照リスト.....19,682
創成.....103,106,112,120,130
創成ファイルの指定.....664
相対ファイル.....97,109
相対ファイルの処理.....112
相対ファイルの定義.....110
相対ファイルのレコードの定義.....111
相対レコード番号.....111
挿入.....115,124,130
添字.....595
添字および指標検査.....78
属性.....643
測定.....577
測定結果のマージ.....571
測定対象.....575
その他のファイル機能.....134
ソフトウェア工学.....300
ソース解析情報ファイルの出力.....676
ソース解析ツール.....535
ソース切替.....576
ソース生成.....625
ソース定義.....312
ソースファイル.....15
ソースプログラム.....7
ソースプログラムの一連番号領域.....673
ソースプログラムの作成.....7
ソースプログラムリスト.....20,677
ソート.....288,289
ソート処理の種類.....289

[た]

代入式.....596
代入時の適合チェック.....334
大容量ファイルアクセスの指定.....138
大容量ファイル処理.....137
対話型デバッガ.....2,15,37,465,552,680
他言語プログラム.....234
他社プリンタ装置.....176
多重継承.....345
多態.....309,338,341
多バイトEUCコードセット表現.....714
タブの扱い.....679
ダミーファイル.....146
単純構造.....29,32
単純構造の実行可能プログラムを作成.....748
ターゲットリポジトリファイル.....399
注意事項.....771
抽象クラス.....309
抽象データ型.....301
チューニング.....569
帳票定義体.....173,195,204
帳票定義体の作成.....206
帳票定義体を使う印刷ファイルの使い方.....195
重複文字の扱い.....665
チームデバッグ.....571
通貨編集用文字の扱い.....664
通信機能.....482
通信の種類.....482
ツール管理機能.....507
定義済みオブジェクト一意名.....595

定義済みオブジェクト一意名SELF....342
定義済みオブジェクト一意名SUPER....341
定数....596
定数項目....229
適合....331
適合チェック....332,333
適合チェックの実現....336
適合の概念....331
テキストファイル....635
テスト回数....575
テスト検証....569
テスト進捗情報....575
テスト進捗情報の管理....576
テスト進捗の管理....575
テスト進捗の測定....571
テスト進捗ファイル....556,571
テスト進捗ファイルの管理....578
テスト網羅性の測定....571
テスト網羅度測定機能....571
デッドロック....734
デバッガの起動....554
デバッガの機能....565
デバッガの状態を知る....567
デバッガの操作....561
デバッグ....462,769
デバッグ環境を変更する....569
デバッグ機能....462
デバッグ機能の種類....70
デバッグ作業....567
デバッグ作業の流れ....72,76,82,88,90
デバッグ情報....556
デバッグ情報ファイル....16
デバッグ情報ファイルのディレクトリ....38
デバッグ対象プログラム....553
デバッグの手順....552
デバッグプログラム....553
電子帳票出力機能....211
データエリアに関するリスト....22
データ型の対応....255
データ監視ウィンドウ....562
データ項目の属性の選択....766
データのアクセス....327
データの扱い....422
データの共用....244
データの内容を確認....565
データベース連携....729
データマップリスト....22,671
データ名....128
データロックサブルーチン....466
同期制御....430,433
凍結....577
凍結状態....577
動作環境....556
動作環境ファイル....556
動的結合....15,29
動的構造....392
動的束縛....309,338
動的プログラム構造....30,34,237,394,569
動的プログラム構造での翻訳処理....401
動的プログラム構造でのリンク処理....407
動的プログラム構造の実行可能プログラムを作成....748
動的プログラム構造の注意事項....34
動的プログラム構造の特徴....237
動的呼出し....99,110,117
動的リンク構造....30,33,392,569
動的リンク構造の実行可能プログラムを作成....748
登録集....12,556
登録集原文....7
登録集原文の検索条件....10
登録集原文の作成....8
登録集原文の表示....663
登録集作成....621
登録集生成....624
登録集のファイル出力....624
登録集ファイル....13,16
登録集ファイルの格納ディレクトリ....10
登録集ファイルのディレクトリ....39
登録集ファイル名の拡張子....11
特殊クラス....379
特殊レジスタ....178,596
とじしろ幅....185
とじしろ方向....185
トランザクションアプリケーション....506
トレース情報....72
トレース情報の個数....58

[な]

内部表現形式....738
内部プログラム....246
名前付きパイプ....147
名前の有効範囲....248
二次入口....242
二次入口点名の指定形式....56
日本語印刷....156
日本語空白....619
日本語空白の比較方法....672
日本語コード系....736
日本語コード系の指定....736
日本語処理のコード系....736
日本語内部表現のコードセット....714
日本語文字の種類....738
日本語文字の種類と表現....738
日本語文字の表現形式....738
日本語利用者語の文字集合....672
入出力....768
入出力エラーが発生したときの実行結果....126
入出力エラー処理....124
入出力エラーの検出....124
入出力機能....443
入出力先の種類と指定方法....263
入出力状態一覧....684
入出力状態のチェック....124
入力項目....229
入力ファイル....290

任意の供給口........184
任意の日付の入力........280

[は]

パス名........709
バッチデバッグ........567
バッファオーバフロー........764
バッファオーバラン........764
パラメタの受渡し........242,452
パラメタの受渡し方法........249,253
パラメタの指定........322
パラメタ領域を獲得する........568
汎用ログ........62,272
パーティション形式........195
引数........57
引数の型........694
非終端記号........590,591
左とじ........185
非同期型メッセージ通信........482
表意定数QUOTEの扱い........675
表示........639
標識領域........7
表示ファイル........155,179,439
表示ファイル機能........221,482
表示ファイル機能(画面入出力)........221
表示ファイル機能(非同期型メッセージ通信)........482
表示ファイル(帳票印刷)の使い方........204
標準入出力処理........619
ビルダ........541
ファイルアクセスルーチン........718
ファイル管理記述項に指定する情報........484
ファイル記述項に指定する情報........188
ファイル結合子........436
ファイル参照子........100
ファイル参照名........709
ファイル識別名........127,163,210
ファイル識別名定数........128,709
ファイル終了条件の発生........124
ファイル終了条件発生の検出........124
ファイル出力........576
ファイル出力ダイアログ........624
ファイル処理........95
ファイル処理の結果........130
ファイル処理の実行........127
ファイル処理の性能改善........140
ファイル追加書き........145
ファイル定義体........8,556
ファイル定義体拡張子........556
ファイル定義体ファイル........16
ファイル定義体ファイルの格納ディレクトリ........11
ファイル定義体ファイルのディレクトリ........39
ファイル定義体ファイル名の拡張子........11
ファイル内相対行番号........593,595
ファイルの拡張........99,627
ファイルの高速処理........141
ファイルの種類........95
ファイルの種類と処理........99
ファイルの種類と特徴........95
ファイルの処理方法........99
ファイルの操作方法........627
ファイルの創成........99,627
ファイルの排他制御........100,131
ファイルの連結........146
ファイルの割当て........127
ファイル編成........101,105,110,117
ファイル編成の変更........630
ファイル名........100,709
ファイルユーティリティ........626
ファイルユーティリティの機能........626
ファイルを使うプログラム........274
ファイルを排他モードにする方法........131
ファクトリオブジェクト........314,425,432,447
ファクトリオブジェクトの寿命........324
ファクトリ定義........314,595
ファクトリデータ........314
ファクトリメソッド........314,317,368,376
ファンクションキーの利用者定義........230
フォントテーブル........173
フォーマット定義体........761
フォームオーバレイおよびFCBを使う方法........190
フォームオーバレイの焼付け回数........183
フォームオーバレイパターン........172,190
フォームオーバレイパターン名........183
フォームオーバレイパターン名の個数........185
副供給口........184
副キー........118
復元........379
複写数........183
複数ファイルの操作........437
副プログラム........14,29,670
副プログラムのエントリ情報........54
副プログラム名の指定形式........54
副レコードキー........99,118
符号付き10進項目の符号の整形........676
符号付き外部10進項目と英数字項目の比較........682
復帰コード........245,251,253,452
復旧........644
不定ファイル........130
浮動小数点項目........770
部分参照子........595
不変式の移動........689
プリンタ情報ファイルの作成........209
プログラミングスタイル........300
プログラム........1
プログラム間結合........769
プログラム間連絡機能........234
プログラム構造........29,391
プログラム構造の概要........392
プログラム構造の指定........665
プログラムごとのコード系........736
プログラム修飾........592
プログラム制御情報リスト........24,671
プログラム定義........423,683
プログラムの誤りを検出する........565

プログラムの書き方......7
プログラムの記述....187,191,200,207,224,286,296,297,298,483
プログラムの形式......8
プログラムの作成と編集......7
プログラムの実行......44,189,193,203,210,228,485
プログラムの実行形式......57
プログラムのデバッグ......70
プログラムの品質向上......571
プログラムの翻訳とリンク......10
プログラムの翻訳・リンク......136,203,209,227,233,485
プログラム表示......579
プログラム名......575,591
プログラム名長......619
プログラム名定数......709
プロジェクト管理機能......507
プロジェクトマネージャ......2,507,508
プロセスID取得サブルーチン......710
プロセス終了サブルーチン......725
プロセス単位で確保/管理されるデータ......422
プロセスモデル......417
ブロック......576
ブロック化......412
プロパティ......368
プロパティメソッド......349
文キーワード......596
文クリア......576
分散開発作業......754
分散開発支援機能......646,648
文識別番号......594
文数指定......576
分離されたメソッド......344
分離されたメソッド定義で使用できない機能......728
併合......288
変換......634
変更箇所を調べる......566
編集の規則......622
ポインタ修飾......296
ポインタ修飾子......595
ポインタ付け......296
保存......378
保存するオブジェクトのメソッドの追加......376
翻訳......10,453,459,554
翻訳オプション......10,658,683
翻訳オプション一覧......658
翻訳オプション設定ツール......547
翻訳オプションの指定......40
翻訳オプションの指定形式......659
翻訳時間......575
翻訳指示文......9
翻訳時の適合チェック......334
翻訳だけを行う指定......36
翻訳単位......391
翻訳単位ごとのオブジェクトファイルの出力......671
翻訳単位統計情報リスト......19,671
翻訳に関するオプション......35
翻訳に関する指示の優先順位......40
翻訳の手順......398
翻訳リストの1行あたりの文字数......670
翻訳リストの1ページあたりの行数......669
翻訳リストのファイル名......40
翻訳リストファイル......16
翻訳リストファイルのディレクトリ......38
翻訳・リンク方法......258
ポートレートモード......183

[ま]

マルチスレッドの効果......417
マルチスレッドのプログラムをリンクする指定......42
マルチスレッドモデル......417,680
マルチスレッドモデルのプログラム......40
マルチスレッドモデルのプログラムの外部属性......676
マージ......288,292,578
マージ処理の種類......292
右とじ......185
未実行文の検索......576
無効キー条件の発生......125
無効キー条件発生の検出......124
無駄な代入の除去......690
メインウィンドウ......561,632
メソッド......303
メソッド定義......317,428
メソッドのPROTOTYPE宣言......344
メソッドの上書き......330
メソッドのエントリ情報......408
メソッドの行内呼出し......346
メソッドの静的束縛......338
メソッドの束縛......337
メソッドの動的束縛......338
メソッドの動的プログラム構造......395
メソッドの呼出し......305,320
メソッド呼出し時の適合チェック......334
メソッドを動的プログラム構造にする......401
メッセージ管理機能......508
メッセージ管理ツール......530
メッセージの受渡し形態......486
メッセージの書込み......489
メッセージの読込み......489
メモリチェック機能......88
メモリチェック機能の使い方......88
メモリ割当てサブルーチン......718
メンバ関数呼出しインタフェースプログラム......368
メンバ変数参照/設定インタフェースプログラム......369
目的プログラムの出力......674
目的プログラムリスト......21
目的プログラムリストの出力の可否......670
文字コードの扱い......667
文字属性......178
文字の泣き別れ......475
文字表現のデータ......635
モジュール化......300
文字列を扱うクラス......386

[や]

誘導変数......689
誘導変数の最適化......689

ユーティリティ........1
用紙供給口........184
用紙サイズ........184
呼出し関係の形態........234
呼出し単位で確保/管理されるデータ........423
呼出し方法........242,249,253
予約語の種類........675

[ら]

ライブラリサーチパス名を追加する指定........41
ラインコマンド入力ウィンドウ........562
ラインコマンド入力ウィンドウからのコマンド指定........563
ラインプリンタモード........183
ラインモード........552,557
ランドスケープモード........183
乱呼出し........99,110,117
リポジトリ........335
リポジトリファイル........16,335
リポジトリファイル更新の影響........336,402
リポジトリファイルと翻訳の手順........396
リポジトリファイルの概要........335
リポジトリファイルの出力........397
リポジトリファイルの入出力先ディレクトリ........37
リポジトリファイルの入力........397
リポジトリファイルの入力先ディレクトリ........11,40
リモートアクセス........486
領域破壊........88
利用者定義のファンクションキー........231
利用者プログラム標準入出力ウィンドウ........563
両面印刷........184
履歴およびテンプレート機能........508
履歴ファイル........556
リンク........10,453,459,554
リンクオプション設定ツール........549
リンクオプションの指定........42
リンク関係とリンクの手順........404
リンクする副プログラムまたはライブラリの指定........42
リンク単位........391
リンクチェック........462
リンクに関するオプション........41
隣接転記の統合........690
例外オブジェクト........361
例外処理........361
レコードキー........99
レコードキー番号........640
レコード形式........98,101,644
レコード順ファイル........96,100
レコード順ファイルの処理........102
レコード順ファイルの定義........100
レコード順ファイルのレコードの定義........101
レコード長........101,644
レコード内相対位置........634
レコードの印刷........629
レコードの更新........99
レコードの更新/追加/削除........628
レコードの構成........101,105,111,119
レコードの削除........99
レコードの参照........99
レコードの整列........630
レコードの設計........98
レコードの挿入........99
レコードの表示........629
レコードの編集........634
レコードを排他状態にする方法........132
連携機能........2,443
連携プログラムの構造........365
連絡節のデータ宣言の扱い........668
ログ........490
ログ内容........491
ロック........430
ロックの解放........466,467
ロックの獲得........466,467
論理宛先定義ファイル........487
論理宛先の削除........489
論理宛先の状態表示........489
論理宛先の創成........489
論理宛先のメッセージ削除........489
論理宛先のモード変更........489
ローカルアクセス........486
ロード........636