HITACHI
Inspire the Next

COBOL2002　使用の手引　手引編

解説・手引書

3021-3-602-50

COBOL2002

前書き

■ 対象製品

P-1M36-1141 COBOL2002 Net Server Suite 04-70（適用 OS：AIX V7.1，AIX V7.2，AIX 7.3）

P-1M36-2141 COBOL2002 Net Server Runtime 04-70（適用 OS：AIX V7.1，AIX V7.2，AIX 7.3）

P-1M36-1241 COBOL2002 Net Server Suite(64) 04-70（適用 OS：AIX V7.1，AIX V7.2，AIX 7.3）

P-1M36-2241 COBOL2002 Net Server Runtime(64) 04-70（適用 OS：AIX V7.1，AIX V7.2，AIX 7.3）

P-9W36-1241 COBOL2002 Net Server Suite(64) 04-70（適用 OS：Linux Server 7 (64-bit x86_64)，Linux Server 8 (64-bit x86_64)，Linux Server 9 (64-bit x86_64)）

P-9W36-2241 COBOL2002 Net Server Runtime(64) 04-70（適用 OS：Linux Server 7 (64-bit x86_64)，Linux Server 8 (64-bit x86_64)，Linux Server 9 (64-bit x86_64)）

これらのプログラムプロダクトのほかにもこのマニュアルをご利用になれる場合があります。詳細は「リリースノート」でご確認ください。

COBOL2002 は，COBOL2002 規格（ISO/IEC 1989:2002）の主な機能に対応しています。
COBOL85 互換機能（コンパイラオプション）指定時，COBOL2002 は，COBOL85 規格（ISO85，ANSI85，JIS88，JIS92）の上位水準に準拠します。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

HITACHI，Cosminexus，DCCM，HiRDB，OpenTP1，XDM，XMAP は，株式会社 日立製作所の商標または登録商標です。

AIX は，世界の多くの国で登録された International Business Machines Corporation の商標です。

AMD は，Advanced Micro Devices, Inc.の商標です。

Docker および Docker ロゴは、Docker Inc. の米国およびその他の国における商標もしくは登録商標です。

Intel は，Intel Corporation またはその子会社の商標です。

Linux は，Linus Torvalds 氏の米国およびその他の国における登録商標です。

Microsoft，Windows は，マイクロソフト 企業グループの商標です。

Oracle(R)，Java，MySQL 及び NetSuite は，Oracle，その子会社及び関連会社の米国及びその他の国における登録商標です。

■ 発行

2024 年 4 月　3021-3-602-50

■ 著作権

変更内容

変更内容（3021-3-602-50） COBOL2002 Net Server Suite 04-70，COBOL2002 Net Server Runtime 04-70，COBOL2002 Net Server Suite(64) 04-70，COBOL2002 Net Server Runtime(64) 04-70

追加・変更内容	変更個所
次の製品の適用 OS に「AIX 7.3」を追加した。 • P-1M36-1141 COBOL2002 Net Server Suite • P-1M36-2141 COBOL2002 Net Server Runtime • P-1M36-1241 COBOL2002 Net Server Suite(64) • P-1M36-2241 COBOL2002 Net Server Runtime(64)	－
Linux で，環境変数 CBLLPATH を使用した動的リンクによるプログラム呼び出しをサポートした。	18.6.2，18.6.4，35.3.2，35.3.3
直前に実行された組み込み関数の CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数で，コード変換に失敗した場合のエラー情報を取得するサービスルーチン（CBLCNVERRORINFO サービスルーチン）をサポートした。 また，組み込み関数の CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数で，文字コード変換エラーによる EC-ARGUMENT-FUNCTION/EC-ARGUMENT-IMP 例外の検出を抑止するオプション（-FunctionECSup,CodeConvErr オプション）をサポートした。	21.3.3，21.8.1，27.5.4，29.1.4，29.7.3，32.5.4，32.5.14，付録 H
STRING 文で数字項目のけた拡張機能への対応をサポートした。	28.4.3
ASSIGN 句の利用者定義語がデータ名と一致していても外部装置名とみなすオプション（-VOSCBL,AssignDataToDevice）をサポートした。	32.5.3，32.5.4，32.5.12，付録 H
EVALUATE 文の WHEN 指定と WHEN OTHER 指定の間の無条件文が省略された場合に，警告エラーメッセージを出力して CONTINUE 文を仮定するオプション（-VOSCBL,EvaluateWhenOther）をサポートした。	32.5.3，32.5.4，32.5.12，付録 H
AIX で実行可能ファイルや共用ライブラリの肥大化を回避するために，オブジェクトの形式を COBOL2002 V3 以前と同様の形式に戻すオプション（-OldStyleObject）をサポートした。	32.5.4，32.5.10，34.1.1，34.1.2，34.2.1，付録 B.1，付録 H
初期化漏れチェック機能で，次に示す情報を一覧で出力する機能（環境変数 CBLUNINITDATA_OUTRESULTLIST および CBLUNINITDATA_OUTRESULTLISTDIR）をサポートした。 • 初期化漏れの可能性があるデータ項目の情報（初期化漏れ確認結果一覧） • 初期化漏れの可能性がある集団項目に対する従属項目の情報（初期化漏れ従属項目一覧）	32.6.2，32.6.3，32.7.4
Linux(x64)で，リポジトリ管理ツールからのコマンド呼び出しに失敗したメッセージ（0027）を追加した。	33.4.2

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルは，次に示すプログラムプロダクトの機能と使用方法について説明したものです。

- P-1M36-1141 COBOL2002 Net Server Suite
- P-1M36-2141 COBOL2002 Net Server Runtime
- P-1M36-1241 COBOL2002 Net Server Suite(64)
- P-1M36-2241 COBOL2002 Net Server Runtime(64)
- P-9W36-1241 COBOL2002 Net Server Suite(64)
- P-9W36-2241 COBOL2002 Net Server Runtime(64)

なお，Linux(x86) COBOL2002，Linux(x64) COBOL2002 をご使用になる場合は，初めに「38. Linux(x86) COBOL2002，Linux(x64) COBOL2002 での UTF-8 ロケールの対応」をお読みください。

■ 対象読者

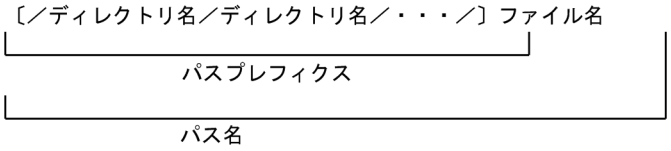
このマニュアルは，COBOL2002 の機能を知りたい方，または COBOL2002 の文法規則を機能から調べたい方を対象としています。また，COBOL の基本的な言語仕様と，UNIX の操作方法について理解していることを前提としています。

■ 用語の定義

このマニュアルでの用語の定義を次に示します。

パス名とパスプレフィクス

パス名とパスプレフィクスは，次のとおりです。



絶対パス名

ルートディレクトリのシンボル「/」で始まるパス名。

/ディレクトリ名/ディレクトリ名/ … /ファイル名

相対パス名

カレントディレクトリからの相対のパス名。

・カレントディレクトリの 1 階層上位のディレクトリを経由する場合

../ディレクトリ名/ディレクトリ名/ … /ファイル名

・カレントディレクトリ下のディレクトリを経由する場合
ディレクトリ名/ディレクトリ名/ … /ファイル名

絶対パスプレフィクス

ルートディレクトリのシンボル「/」で始まるパスプレフィクス。

相対パスプレフィクス

カレントディレクトリからの相対のパスプレフィクス。

■ このマニュアルで使用する記号

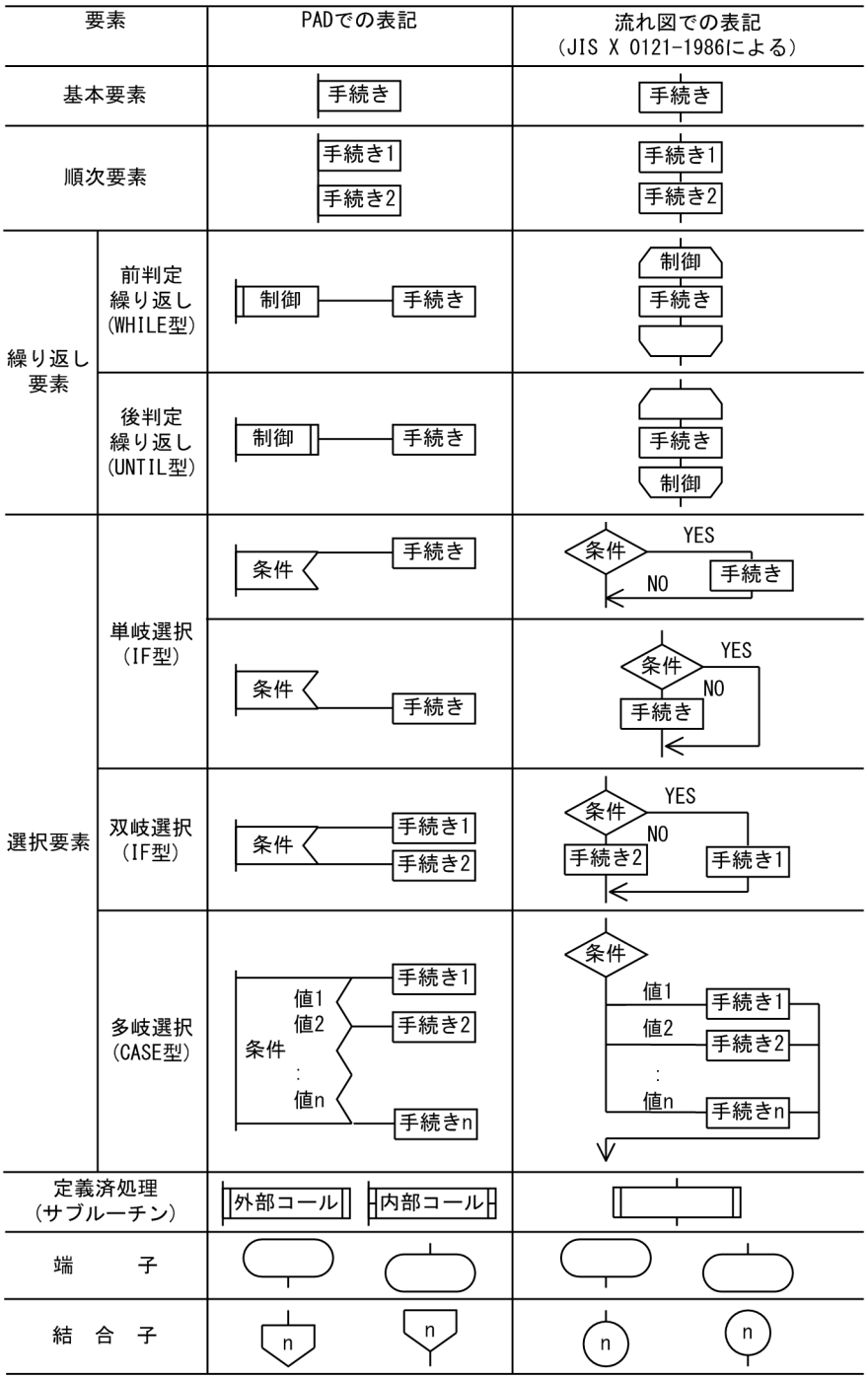
このマニュアルで使用する記号を次に示します。

記号	意味
［ ］キー	文字キーやＦ（ファンクション）キーを意味する。
［ ］＋ ［ ］キー	＋の前のキーを押したまま，あとのキーを押すことを意味する。
{ }	この記号で囲まれている複数の項目のうちから一つを選択することを意味する。項目が縦に複数行にわたって記述されている場合は，そのうちの 1 行分を選択する。
{ } ＋	この記号で囲まれている複数の項目のうちから一つを選択することを意味する。同じ要素の繰り返し指定はできないが，異なる要素であれば，複数指定できる。例えば，{A | B | C} ＋と表記されている場合は，A・B・C をすべて選択してもよい。 項目が縦に複数行にわたって記述されている場合は，そのうちの 1 行分を選択する。
〔 〕	この記号で囲まれている項目は省略してもよいことを意味する。 複数の項目が縦または横に並べて記述されている場合には，すべてを省略するか，記号 { } と同じく，どれか一つを選択する。
…	記述が省略されていることを意味する。 この記号の直前に示された項目を繰り返して複数個指定できる。
下線	括弧で囲まれた複数の項目のうち 1 項目に対して使用され，括弧内のすべてを省略したときにシステムがとる標準値を意味する。
|	横に並べられた複数の項目に対して項目間の区切りを示し，「または」を意味する。
［ ］	ウィンドウのメニューバーから選択するメニュー，コマンド，またはボタンを意味する。

■ プログラム構造表記法（PAD）と流れ図

このマニュアルでは，手続き的アルゴリズムの制御構造（主にプログラムの処理手続き）を PAD（Problem Analysis Diagram）または流れ図で示しています。

このマニュアルで使用する PAD と流れ図の要素，およびそれぞれの要素の対応を，次のように定義します。



■ ウィンドウ図

マニュアル中のウィンドウ図は，説明に直接関係のない部分を省略して記述しています。したがって，実際に操作していて表示されるウィンドウとは一部異なります。

■ サポート機能一覧

規格

機能	製品種別			
	AIX(32) COBOL 2002	AIX(64) COBOL 2002	Linux(x86) COBOL 2002	Linux(x64) COBOL 2002
基本機能	○	○	○	○
順編成ファイル	○	○	○	○
相対編成ファイル	○	○	○	○
索引編成ファイル	○	○	○	○
整列併合	○	○	○	○
プログラム間連絡	○	○	○	○
組み込み関数	○	○	○	○
オブジェクト指向	○	○	○	○
共通例外処理	○	○	○	○
再帰呼び出し	○	○	○	○
利用者定義関数	○	○	○	○
局所場所節(LOCAL-STORAGE SECTION)	○	○	○	○
原始文操作	○	○	○	○
自由形式正書法	○	○	○	○
TYPEDEF 句と SAME AS 句	○	○	○	○
翻訳指令	○	○	○	○
区分化※	○	○	○	○

(凡例)

○：サポートしている

注※

覚え書きとしてサポートしている。

X/Open

機能		製品種別			
		AIX(32) COBOL 2002	AIX(64) COBOL 2002	Linux(x86) COBOL 2002	Linux(x64) COBOL 2002
テキスト編成ファイル		○	○	○	○
ファイル共用（ファイルシェア）	順編成ファイル	○	○	○	○
	相対編成ファイル	○	○	○	○
	ISAM による索引編成ファイル	○	○	○	○
	テキスト編成ファイル	○	○	○	○
	CSV 編成ファイル	×	×	×	×
	HiRDB による索引編成ファイル	×	×	×	×
	Btrieve による索引編成ファイル	−	−	−	−
コマンド行および環境変数へのアクセス		○	○	○	○
画面節（SCREEN SECTION）による画面操作		○	○	×	×
C 言語インタフェース		○	○	○	○
インターナショナリゼーション		○	○	○	○

（凡例）

○：サポートしている

×：サポートしていない

−：該当しない

拡張機能

機能	製品種別			
	AIX(32) COBOL 2002	AIX(64) COBOL 2002	Linux(x86) COBOL 2002	Linux(x64) COBOL 2002
日本語	○	○	○	○
ブール演算	○	○	○	○
アドレス操作	○	○	○	○
1 バイト 2 進および COMP-X 項目	○	○	○	○
浮動小数点項目	○	○	○	○

機能		製品種別			
		AIX(32) COBOL 2002	AIX(64) COBOL 2002	Linux(x86) COBOL 2002	Linux(x64) COBOL 2002
ISAM による索引編成ファイル機能の拡張（合成キー，逆順読み）		○	○	○	○
CSV 編成ファイル		○	○	○	○
HiRDB による索引編成ファイル		○	○	×	×
Btrieve による索引編成ファイル		−	−	−	−
リモートファイルアクセス		×	×	×	×
ラージファイル入出力	順編成ファイル	○	○	○	○
	相対編成ファイル	×	×	×	×
	ISAM による索引編成ファイル	×	×	×	×
	テキスト編成ファイル	○	○	○	○
	CSV 編成ファイル	○	○	○	○
	HiRDB による索引編成ファイル	×	×	×	×
	Btrieve による索引編成ファイル	−	−	−	−
COBOL 入出力サービスルーチン		○	○	○	○
バイトストリーム入出力サービスルーチン		○	○	○	○
ファイル入出力拡張機能	ファイルサイズがレコード長の整数倍でない固定長形式の順ファイル入出力	○	○	○	○
	ファイルバッファサイズ指定機能	○	○	○	○
画面節（WINDOW SECTION）による画面操作	画面節（WINDOW SECTION）	○	○	×	×
	JCPOPUP サービスルーチン	○	○	×	×
通信節による画面操作（XMAP3）		○	×	×	×
COPY 文の接頭辞／接尾辞		○	○	○	○
プリンタへのアクセス	XMAP3 による印刷	○	×	×	×
	GDI モード印刷	−	−	−	−
	ESC/P モード印刷	−	−	−	−
ファイルのディスク書き込み保証（書き込み時）	順編成ファイル	○	○	○	○
	相対編成ファイル	○	○	○	○
	ISAM による索引編成ファイル	○	○	○	○

機能		製品種別			
		AIX(32) COBOL 2002	AIX(64) COBOL 2002	Linux(x86) COBOL 2002	Linux(x64) COBOL 2002
	テキスト編成ファイル	×	×	×	×
	CSV 編成ファイル	×	×	×	×
	HiRDB による索引編成ファイル	×	×	×	×
	Btrieve による索引編成ファイル	−	−	−	−
ファイルのディスク書き込み保証（クローズ時）	順編成ファイル	○	○	○	○
	相対編成ファイル	○	○	○	○
	ISAM による索引編成ファイル	○	○	○	○
	テキスト編成ファイル	○	○	○	○
	CSV 編成ファイル	○	○	○	○
	HiRDB による索引編成ファイル	×	×	×	×
	Btrieve による索引編成ファイル	−	−	−	−
報告書作成機能		○	○	○	○
MIOS7 COBOL85 との互換機能		×	×	×	×
イベントログファイル／syslog ファイル出力機能		×	×	×	×
データコミュニケーション機能		○	○	○	○
データベース操作機能（ODBC インタフェース）		−	−	○	○
XDM によるデータベースシミュレーション機能	構造型データベース（XDM/SD）	○	○	○	○
	リレーショナルデータベース（XDM/RD）	○	○	○	○
OLE2 オートメーション機能	クライアント機能	−	−	−	−
	サーバ機能	−	−	−	−
CGI プログラム作成支援機能		○	×	×	×
マルチスレッド環境	作成と実行	○	○	○	○
	デバッグ	△	△	△	△
MSMQ アクセス機能		−	−	−	−
エンディアン切り替え		×	×	○	○
Unicode 機能		○	○	○	○

機能		製品種別			
		AIX(32) COBOL 2002	AIX(64) COBOL 2002	Linux(x86) COBOL 2002	Linux(x64) COBOL 2002
数字項目のけた拡張機能		×	○	×	○
動的長基本項目機能		○	○	○	○
定数長拡張機能	英数字定数長の拡張	○	○	○	○

(凡例)

○：サポートしている

×：サポートしていない

△：サポートしている機能であるが，使える機能に一部制限がある

－：該当しない

デバッグ機能

機能		製品種別			
		AIX(32) COBOL2002	AIX(64) COBOL2002	Linux(x86) COBOL2002	Linux(x64) COBOL2002
実行時デバッグ機能		○	○	○	○
テストデバッグ機能	GUI モード	×	×	×	×
	バッチモード	○	○	○	○
	ラインモード	○	○	○	○
カバレージ機能	GUI モード	×	×	×	×
	バッチモード	○	○	○	○

(凡例)

○：サポートしている

×：サポートしていない

連携機能

機能	製品種別			
	AIX(32) COBOL 2002	AIX(64) COBOL 2002	Linux(x86) COBOL 2002	Linux(x64) COBOL 2002
XML 連携機能	○	○	○	○
Cosminexus 連携機能	×	○	×	○

(凡例)

○：サポートしている

×：サポートしていない

■ ウィンドウに表示されるタイトル名の表記

AIX(32) COBOL2002 と，AIX(64) COBOL2002 では，ウィンドウに表示されるタイトル名が異なります。このマニュアルでは，AIX(32) COBOL2002 のタイトル名を使用します。AIX(64) COBOL2002 をご使用になる場合は，次のように読み替えてください。

ウィンドウに表示されるタイトル名	AIX(64) COBOL2002 の場合の読み替え
cbl2002term	cbl2002term 64bit

■ プログラム例について

このマニュアルのプログラム例は，断り書きがない場合は AIX(32) COBOL2002 および Linux(x86) COBOL2002 用です。プログラム例を AIX(64) COBOL2002 および Linux(x64) COBOL2002 で使用するには，プログラムの記述に変更が必要な場合がありますのでご注意ください。

目次

1 概要

電子計算機を利用する場合，電子計算機にさせたい作業内容をプログラミング言語で記述します。COBOL（COmmon Business Oriented Language（事務用共通言語））はプログラミング言語の一つです。

この章では，COBOL2002 の概要について説明します。

1.1 COBOL2002 の概要

1.1.1 COBOL の概要

COBOL は，事務処理用に最も使用されているプログラミング言語です。当初はビジネス向けプログラミング言語として開発されましたが，第 1 次規格（ANSI68，ISO72，JIS72），第 2 次規格（ANSI74，ISO78，JIS80），第 3 次規格（ISO85，ANSI85，JIS88，JIS92），第 4 次規格（ISO/IEC 1989:2002，JIS X3002:2011）と規格の改訂を重ね，現在ではビジネス向けだけでなく，汎用プログラミングにも広がっています。

ここでは，COBOL の特長，および COBOL2002 の機能の概要について説明します。

なお，COBOL に関する公式の規格は，ISO（国際規格），ANSI（アメリカ規格），および JIS（日本工業規格）で決められています。

1.1.2 COBOL の特長

COBOL の特長を次に示します。

- 大量データの高速処理ができるため，事務作業の効率化ができます。
- 事務作業で頻繁に使う帳票の作成やデータの表現方法が容易です。
- 簡単な英語でプログラムが書けるようになっていて，書き方も決まっているので，プログラムの見直しが容易です。
- データとプログラムの手続きを分割して記述するので，まずデータが存在し，続いて処理方法を考えるという事務処理用プログラムの作成方法に対応しています。

1.1.3 COBOL2002 の機能

(1) COBOL2002 で追加された機能

COBOL2002 は，従来製品の COBOL85 との高い互換を保証しています。また，次に示す機能が追加されています。

- オブジェクト指向機能
- 共通例外処理
- 翻訳指令
- TYPEDEF 句と SAME AS 句
- 利用者定義関数

- 再帰呼び出し
- 局所場所節
- 自由形式のソース原文や登録集原文

(2) COBOL2002 の Web システム連携機能

COBOL2002 では，Web システム構築に必要な次の連携機能を使用できます。これらの機能を使用することで，Web アプリケーションを COBOL で開発できます。

XML 連携機能

COBOL プログラムから，XML データを COBOL のレコードとして入出力できます。COBOL のノウハウや既存 COBOL 資産を生かして，e ビジネス向けのデータ交換用 XML データを扱うアプリケーションを作成できます。

詳細については，マニュアル「COBOL2002　XML 連携機能ガイド」を参照してください。

Cosminexus 連携機能（AIX(64)，Linux(x64)で有効）

日立アプリケーションサーバ Cosminexus の Java アプリケーションから，COBOL プログラムを JavaBeans あるいは EJB として呼び出せます。これによって，Web アプリケーションの基本機能となるビジネスロジック部分を COBOL で作成できます。

詳細については，マニュアル「COBOL2002　Cosminexus 連携機能ガイド」を参照してください。

1.1.4 COBOL2002 の製品体系

COBOL2002 では，次に示すプログラムプロダクトを用意しています。利用形態に合わせて適宜お選びください。

プログラムプロダクト名	開発環境	実行環境	システム運用※
COBOL2002 Net Server Runtime	－	○	○
COBOL2002 Net Server Suite	○	○	○
COBOL2002 Net Server Runtime(64)	－	○	○
COBOL2002 Net Server Suite(64)	○	○	○

（凡例）

○：該当するプログラムプロダクトで使用できる

－：該当するプログラムプロダクトでは使用できない

注※

実際の運用環境で，開発したシステムを稼働できるライセンスを含むかどうかを表します。

(1) COBOL2002 Net Server Runtime／COBOL2002 Net Server Runtime(64)

開発した COBOL プログラムを UNIX 上で運用するために，実行環境とシステム運用環境を提供するプログラムプロダクトです。COBOL プログラムの実行に必要な実行時ライブラリを提供しています。

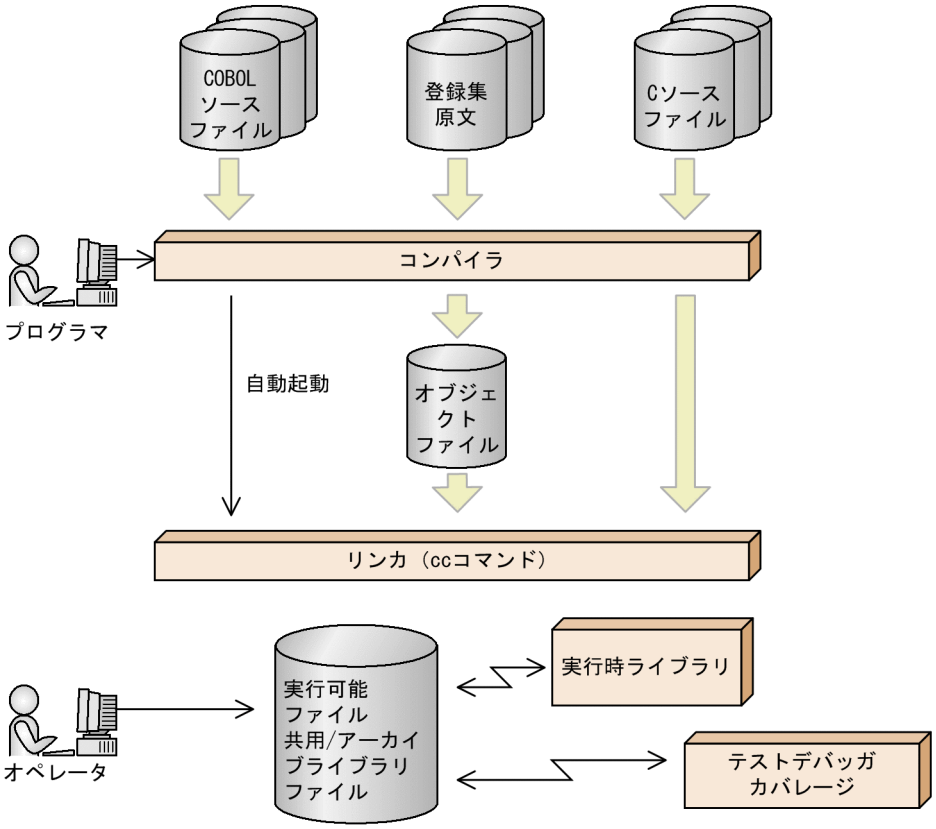
(2) COBOL2002 Net Server Suite／COBOL2002 Net Server Suite(64)

COBOL2002 の機能を活用したアプリケーションプログラムの開発から，テスト／デバッグまでを支援するためのプログラムプロダクトです。COBOL2002 コンパイラ，デバッガおよび実行時ライブラリを提供し，アプリケーションプログラムの開発から運用までを支援します。

1.2　COBOL2002 の構成

COBOL2002 の構成を，次に示します。

図 1–1　COBOL2002 の構成



1.3　COBOL2002 が提供するコンポーネントの種類と関連性

COBOL2002 では，コンパイラ・実行時ライブラリのほかにも，さまざまな周辺ツールを使用できます。ここでは，使用環境ごとに，使用できるコンポーネントを説明します。

1.3.1　開発環境について

COBOL2002 を利用した開発環境で使用できるコンポーネントと，その機能の概要を次に示します。

表 1-1　開発環境のコンポーネント

コンポーネント	機能概要
コンパイラ	COBOL プログラムのコンパイル

1.3.2　実行環境について

COBOL2002 を利用した実行環境で使用できるコンポーネントと，その機能の概要を次に示します。

表 1-2　実行環境のコンポーネント

コンポーネント	機能概要
実行時ライブラリ	COBOL プログラムの実行
ISAM	索引ファイルの作成や保守
SORT	COBOL の整列併合機能の実行

1.3.3　デバッグ環境について

COBOL2002 を利用したデバッグ環境で使用できるコンポーネントと，その機能の概要を次に示します。

表 1-3　デバッグ環境のコンポーネント

コンポーネント	機能概要
テストデバッガ	COBOL プログラムのデバッグ
カバレージ	COBOL プログラムのテスト進捗状況管理の支援

1.3.4　マニュアルについて

COBOL2002 で CD-ROM マニュアルとして提供されるマニュアルの概要を次に示します。

表 1-4　マニュアルの種類

マニュアル名	概要
COBOL2002　使用の手引　手引編	COBOL2002 の機能と使用方法について説明しています。
COBOL2002　使用の手引　操作編	COBOL2002 で使用できるテストデバッガ，カバレージ，TD コマンド生成機能の操作方法について説明しています。
COBOL2002　言語 標準仕様編	COBOL2002 で使用する言語の文法のうち，規格仕様の部分について説明します。
COBOL2002　言語 拡張仕様編	COBOL2002 で使用する言語の文法のうち，日立拡張仕様の部分について説明します。
COBOL2002　メッセージ	COBOL2002 がコンパイル時，実行時などに出力するメッセージの一覧を記載しています。
索引順編成ファイル管理 ISAM	索引順編成ファイルの機能概要，ユティリティの操作，およびユティリティコマンドについて説明しています。
ソートマージ	ソートマージの機能概要，コマンド，およびエラーメッセージについて説明しています。※

注※

このプログラムプロダクトでは，ソートマージのコマンドは使用できません。ソートマージの機能は，COBOL の整列併合機能を利用して使用します。

2 COBOL2002 の主な新機能

COBOL2002 には，幾つかの機能が追加されています。

この章では，COBOL2002 の主な新機能について説明します。

2.1　オブジェクト指向機能

COBOL2002 では，オブジェクト指向機能をサポートしています。オブジェクト指向プログラミングを適用すると，次のことができます。

- データと手続きのカプセル化
- 親クラスの性質を子クラスに引き継ぐ継承
- 一つの手続きで複数のクラスを扱うポリモルフィズム

詳細は，「20.　オブジェクト指向機能」を参照してください。

2.2　共通例外処理

COBOL2002 では，従来の入出力機能に関する例外宣言手続きに加えて，データ例外やけたあふれなど，さまざまな例外に対する例外宣言手続きを記述できます。また，例外オブジェクトを使用した例外処理もできます。

詳細は，「21.　共通例外処理」を参照してください。

2.3　翻訳指令

翻訳指令とは，COBOL プログラムのコンパイル（翻訳）時に，特定の動作や解釈をするようにコンパイラへの指示を与える指令です。

2.3.1　規格の互換性をチェックする翻訳指令

規格の互換性をチェックする翻訳指令を次に示します。

(1)　FLAG-85 指令

FLAG-85 指令は，第 3 次規格 COBOL と第 4 次規格 COBOL との間で互換性に欠けるおそれがある構文に対して，フラグを立てる選択種目を指定します。

FLAG-85 指令の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「3.3.9 FLAG-85 指令」を参照してください。

2.3.2　ソース原文の正書法を決定する翻訳指令

ソース原文の正書法を決定する翻訳指令を次に示します。

(1)　SOURCE FORMAT 指令

SOURCE FORMAT 指令は，後続するソースの正書法が，固定形式または自由形式のどちらであるかを指定します。

詳細は，「32.2.3　正書法」を参照してください。また，SOURCE FORMAT 指令の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「3.3.14　SOURCE FORMAT 指令」を参照してください。

2.3.3　条件翻訳に関連する翻訳指令

条件翻訳とは，ソースコード中に翻訳指令を記述すれば，コンパイル時に特定の行を有効にしたり，無効にしたりできる機能です。

条件翻訳に関する翻訳指令を次に示します。

(1)　DEFINE 指令

DEFINE 指令は，翻訳変数と呼ばれる記号名に対して特定の定数値を指定します。

詳細は，「32.3.3　条件翻訳の利用」を参照してください。また，DEFINE 指令の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「3.3.7　DEFINE 指令」を参照してください。

(2)　EVALUATE 指令

EVALUATE 指令は，多方向分岐を条件翻訳します。

詳細は，「32.3.3　条件翻訳の利用」を参照してください。また，EVALUATE 指令の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「3.3.8　EVALUATE 指令」を参照してください。

(3)　IF 指令

IF 指令は，単方向または双方向分岐の条件翻訳をします。

詳細は，「32.3.3　条件翻訳の利用」を参照してください。また，IF 指令の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「3.3.10　IF 指令」を参照してください。

2.3.4　コンパイルリストに関連する翻訳指令

コンパイルリストに関連する翻訳指令を次に示します。

コンパイルリストは，-SrcList オプションを指定した場合に出力されます。コンパイルリストの出力形式，および-SrcList オプションを指定したときの出力形式の違いについては，「付録 D　コンパイルリスト」を参照してください。

(1)　LISTING 指令

LISTING 指令は，コンパイルリストにソースを出力するかどうかを指定します。詳細は，「付録 D.1　リストの出力」の「(3)　コンパイルリストの出力に関連する翻訳指令」を参照してください。

また，LISTING 指令の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「3.3.11 LISTING 指令」を参照してください。

(2)　PAGE 指令

PAGE 指令は，コンパイルリストの改ページを指定します。PAGE 指令のコンパイルリストでの効果は，固定形式正書法の改ページ標識「/」と同じです。

詳細は，「付録 D.1　リストの出力」の「(3)　コンパイルリストの出力に関連する翻訳指令」を参照してください。また，PAGE 指令の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「3.3.12　PAGE 指令」を参照してください。

2.3.5　例外処理に関連する翻訳指令

例外処理に関連する翻訳指令を次に示します。

(1)　PROPAGATE 指令

PROPAGATE 指令は，呼び出し元のプログラムへ例外を伝播させるために使用します。

詳細は，「21.　共通例外処理」を参照してください。また，PROPAGATE 指令の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「3.3.13　PROPAGATE 指令」を参照してください。

(2)　TURN 指令

TURN 指令は，特定の例外に対してチェックするかどうかを指定します。

詳細は，「21.　共通例外処理」を参照してください。また，TURN 指令の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「3.3.15　TURN 指令」を参照してください。

2.4 TYPEDEF 句と SAME AS 句

COBOL では，レベル番号，データ項目の名前，PICTURE 句や USAGE 句などのデータ属性を指定して，データ項目の構造や形式を表現します。

TYPEDEF 句は，データ構造のひな型となるデータ型を定義します。TYPEDEF 句を使用して定義したデータ型を TYPE 句によって参照することで，同じ構造のデータ項目を定義できます。

SAME AS 句は，あるデータ名の記述項が，別のデータ記述項の指定と同じことを表し，同じ構造のデータ項目を定義します。

TYPEDEF 句，SAME AS 句の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「9.16.84　TYPEDEF 句」および「COBOL2002　言語　標準仕様編」「9.16.70　SAME AS 句」を参照してください。

2.4.1 TYPEDEF 句

TYPEDEF 句を使用したデータ型の指定について説明します。

(1) データ型の定義と参照

データ型は，TYPEDEF 句を使用して定義します。型名を指定した TYPE 句をデータ名に指定すれば，TYPEDEF 句で定義した型を参照できます。

なお，TYPEDEF 句で定義された型名を 1 か所で参照する場合，TYPE 句指定によって暗黙的に展開されるデータ名は一意となりますが，TYPEDEF 句で定義された型名を複数個所で参照する場合，TYPE 句指定によって暗黙的に展開されるデータ名が一意とならない（同じデータ名が複数個所に存在する）ため，一意に参照するためにデータ名の修飾が必要です。

データ型の定義と参照の例を次に示します。

```
        :
    DATA  DIVISION.
        :
   *> 型名の定義
   01 DATA-TYPE  TYPEDEF.
      02 DATA-NAME PIC X(15) OCCURS 10.   *> 型名"DATA-TYPE"を定義

   *> 型名の参照
    01 DATA-REF1.
      02 DATA1 OCCURS 100.
        03 DATA2 TYPE DATA-TYPE.          *> 型名"DATA-TYPE"を参照

              DATA-REF1は次のように定義されたのと同じ

    01 DATA-REF1.
      02 DATA1 OCCURS 100.
        03 DATA2.
          04 DATA-NAME PIC X(15) OCCURS 10.

    PROCEDURE  DIVISION.
   *> TYPE句を用いて定義したデータ名を手続き部で参照
        MOVE 'AAA' TO  DATA-NAME  OF DATA2(1 ,1).
```

(2)　弱く型付けされた項目と強く型付けされた項目

データ型は，TYPEDEF 句に STRONG 指定があるかどうかによって，次の二つに分類されます。

- STRONG 指定なし：弱く型付けされたデータ型
- STRONG 指定あり：強く型付けされたデータ型（集団項目のデータ型にだけ指定できます）

(a)　弱く型付けされた項目

弱く型付けされた項目とは，弱く型付けされたデータ型を TYPE 句に指定することで定義されたデータ項目のことです。この項目は，指定された型名からそのデータ構造が決まること以外は，型付けされていない項目と同じように使用できます。

(b)　強く型付けされた項目

強く型付けされた項目とは，強く型付けされたデータ型を TYPE 句に指定することで定義されたデータ項目のことです。この項目は，集団項目のデータ内容の妥当性を確保するための仕組みです。集団項目中の基本データ項目に格納する内容の整合性を損なうおそれがある操作は，すべて禁止されています。整合性を損なうおそれがある操作の手続き文を書いた場合，コンパイル時にエラーとなります。

(整合性を損なうおそれがある操作の例)

- 型の異なる集団項目からの転記
- VALUE 句による初期化
- 再命名，再定義された一意名を使ったデータ項目の更新
- 部分参照による値の更新

また，USAGE 句に OBJECT REFERENCE 指定のある項目を集団項目の従属項目として定義する場合は，STRONG 指定のあるデータ型の中で定義しなければなりません。

このシステムでは，強く型付けされた項目同士の比較を許していません。強く型付けされた項目を比較する場合，強く型付けされた項目に従属するすべての基本項目同士を比較してください。

強く型付けされた項目の詳細については，マニュアル「COBOL2002　言語　標準仕様編」「4.4.3(2) 強く型付けされた集団項目」，「COBOL2002　言語　標準仕様編」「9.16.83　TYPE 句」，および「COBOL2002　言語　標準仕様編」「9.16.84　TYPEDEF 句」を参照してください。

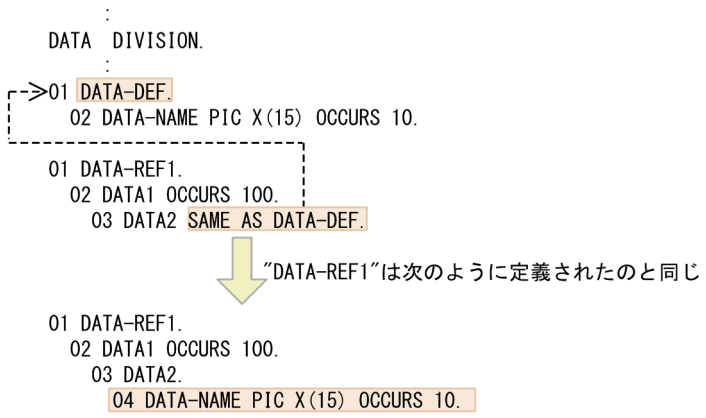
2.4.2　SAME AS 句

SAME AS 句は，あるデータ名のデータ構造を，ファイル節，作業場所節，局所場所節，または連絡節に定義された別のデータ記述項とまったく同じように定義することを示します。

SAME AS 句の使用例を，次に示します。

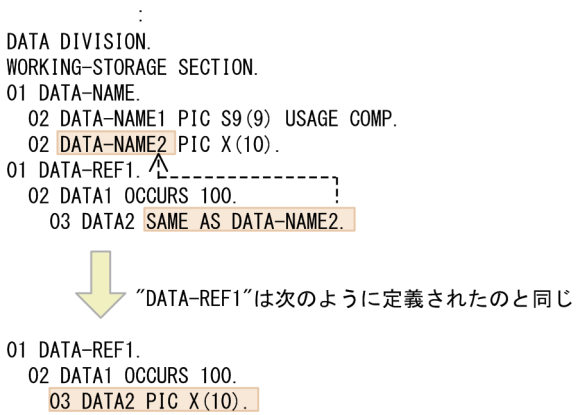
(例 1)

01 レベルの記述項を参照する場合



(例 2)

集団項目中の記述項を参照する場合



2.5　利用者定義関数

利用者定義関数とは，関数名段落（FUNCTION-ID）を指定すれば，ユーザが任意に作成できる関数です。

利用者定義関数は，関数を利用するプログラムの中から関数一意名によって参照（活性化）でき，関数定義の手続き部見出しの RETURNING 指定で規定した一つの値を返します。また，常に再帰属性となるので，自分自身を呼び出せます。

利用者定義関数については，マニュアル「COBOL2002　言語　標準仕様編」「5.3　利用者定義関数」および「COBOL2002　言語　標準仕様編」「7.4　関数名段落（FUNCTION-ID)」を参照してください。

2.5.1　利用者定義関数の参照

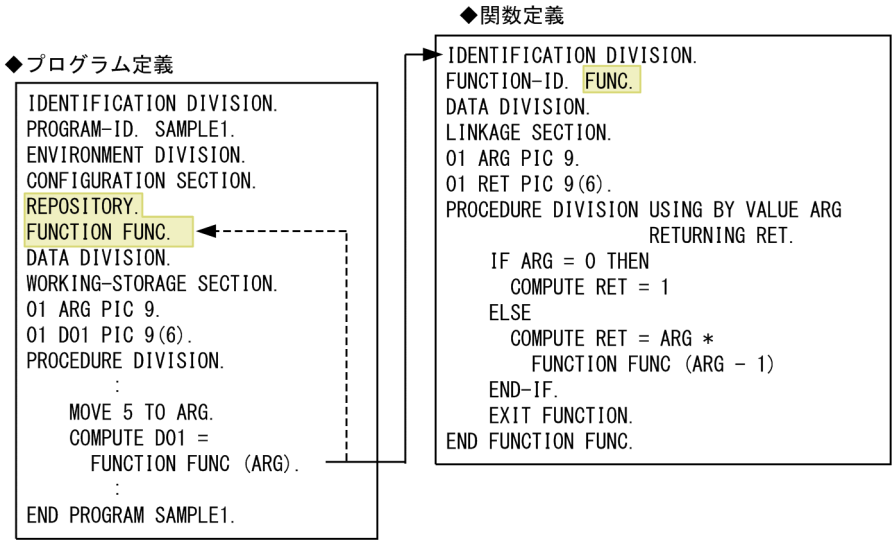
利用者定義関数は，送り出し側の作用対象として，関数一意名によって参照（呼び出し）されます。関数一意名で参照される利用者定義関数は，リポジトリ段落に指定された関数定義となります。

利用者定義関数を使用する場合，該当する利用者定義関数に対応する関数指定子をリポジトリ段落で指定しておく必要があります。

利用者定義関数の参照については，マニュアル「COBOL2002　言語　標準仕様編」「4.3.2(2)　関数一意名」を参照してください。また，リポジトリ段落に関する言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「8.2.7(1)　一般形式」を参照してください。

関数定義の概念を次に示します。

図 2-1　関数定義



上記の図では，関数一意名を使用して利用者定義関数を参照しています。

2.5.2　利用者定義関数の引数と返却項目

利用者定義関数の引数および返却項目について説明します。

(1)　引数

関数一意名で引数を指定した場合，関数定義に指定された仮引数の，BY REFERENCE／BY VALUE 指定に従って引数を受け渡します。ただし，実引数によっては，BY CONTENT が仮定されることがあります。

仮引数に関する言語仕様についてはマニュアル「COBOL2002　言語　標準仕様編」「10.1　手続き部の構成」を参照してください。実引数に関する言語仕様についてはマニュアル「COBOL2002　言語　標準仕様編」「4.3.2(2)　関数一意名」を参照してください。また，BY REFERENCE／BY VALUE 指定については，「17.1.1　引数の受け渡しの種類」を参照してください。

利用者定義関数の引数の扱いを次に示します。

表 2-1　利用者定義関数の引数の扱い

関数定義の仮引数の指定	実引数	実引数の扱い
BY REFERENCE	受け取り側作用対象として許可された，オブジェクトプロパティ／オブジェクト参照以外の項目	BY REFERENCE
	定数，算術式，ブール式，オブジェクトプロパティ，オブジェクト参照，および受け取り側作用対象として許可されない項目	BY CONTENT
BY VALUE	－	BY VALUE

（凡例）

－：該当しない

引数は，コンパイル時に整合性チェックされます。

(2)　返却項目

返却項目の属性は，関数一意名に対応する関数定義の RETURNING に指定された項目の属性で決まります。

(3)　利用者定義関数の注意事項

利用者定義関数の注意事項を，次に示します。

- 利用者定義関数に-Main,System または-Main,V3 オプションを指定した場合，エラーにはなりませんが，-Main オプションが指定されたプログラムとしてコンパイルはされません。
- 利用者定義関数には，ENTRY 文を指定できません。
- CBL，CLS，CLT，CLU で始まる関数名を指定した場合，動作は保証しません。

2.6　再帰呼び出し

再帰呼び出しとは，活性状態にあるプログラムを，直接的または間接的に呼び出すことです。COBOL2002では，RECURSIVE 句が指定されたプログラム定義，利用者定義関数，およびメソッド定義を再帰呼び出しできます。

詳細は，「18.4　プログラム属性」を参照してください。

注意事項

再帰呼び出しでは，呼び出す処理単位ごとにスタック領域を使用するため，大量に再帰呼び出しをした場合，スタック領域が不足する場合があります。スタック領域が不足した場合，システム（OS）の設定によりスタック領域を拡張する必要があります。詳細は，システムのマニュアルを参照してください。

2.7　局所場所節

局所場所節とは，プログラムの実行中だけ有効となるデータ領域項目を定義する個所です。局所場所節に定義したデータ項目は，プログラムが呼び出されたときに領域が確保され，プログラムの実行中だけ有効となります。また，以前呼び出されたときの状態は，保持されません。

詳細は，「4.1.3　局所場所節のデータ領域」を参照してください。

2.8　自由形式のソース原文や登録集原文

COBOL2002 では，自由形式正書法に従って，自由形式のソース原文や登録集原文を作成できます。自由形式正書法では，ソース原文や登録集原文を，行中の任意の位置に記述できます。

詳細は，マニュアル「COBOL2002　言語　標準仕様編」「2.5　自由形式正書法」を参照してください。

3 翻訳グループを構成する定義の種類

COBOL2002 には，コンパイラの翻訳単位として翻訳グループという概念があります。翻訳グループは，一つ以上のソース単位の集合のことです。翻訳グループには，さまざまなソース単位を含めることができます。COBOL2002 では，翻訳グループが 1 回のコンパイル単位となります。

この章では，翻訳グループについて説明します。

3.1　翻訳グループの概要と考え方

翻訳グループは，1 個以上の翻訳単位から構成されます。翻訳単位は，次に示す定義のどれかに該当します。

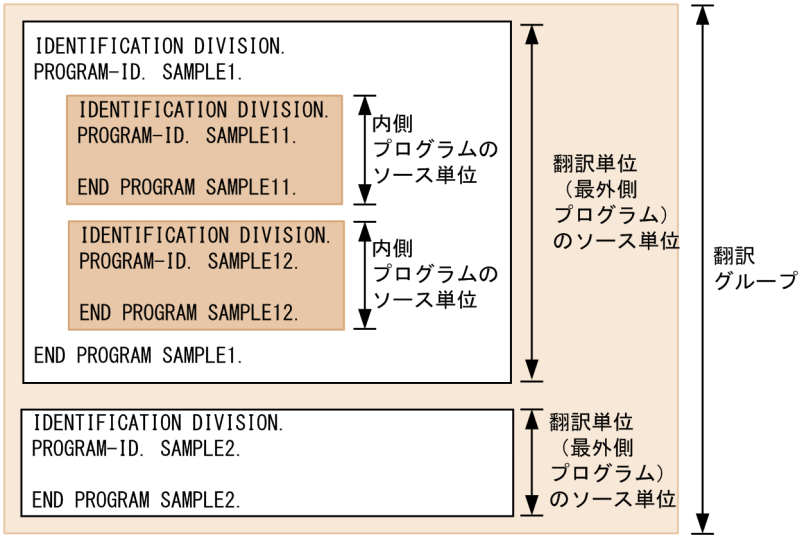
- 最外側のプログラム定義
- 関数定義
- クラス定義
- インタフェース定義

各翻訳単位のコンパイルが成功すると，プログラム定義，関数定義，クラス定義などに対してオブジェクトファイル（.o）が生成されます。実行単位は，オブジェクトファイルから生成された実行可能ファイルや共用ライブラリによって構成されます。

翻訳グループについては，マニュアル「COBOL2002　言語　標準仕様編」「6.　翻訳グループの構造 (Structured compilation group)」を参照してください。

COBOL2002 では，翻訳グループが 1 回のコンパイルの入力単位となります。翻訳グループの構成概念を次に示します。

図 3-1　翻訳グループの構成



3.2　定義の種類

COBOL2002 では，既存のプログラム定義のほかに，オブジェクト指向機能のためのクラス定義，および利用者定義関数のための関数定義があります。また，インタフェース定義のように，クラス定義とのインタフェースだけを表す定義があります。

各定義のコンパイル方法や生成されるオブジェクトファイルとの関係については，「32.　COBOL ソースの作成とコンパイル」を参照してください。

3.2.1　プログラム定義

プログラム定義は，見出し部でプログラム名段落を指定して定義します。COBOL85 で記述されたプログラムは，COBOL2002 のプログラム定義に該当します。

プログラム名段落の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「7.8　プログラム名段落（PROGRAM-ID）」を参照してください。

プログラム定義には，最外側のプログラムと内側のプログラムがあります。

- 最外側のプログラム

  ほかのプログラムに含まれないプログラムです。

  最外側のプログラムでは，プログラム名段落に指定したプログラム名称が，オブジェクトファイル中に外部参照として出力されます。このため，ほかの翻訳単位のプログラム（COBOL 言語以外を含む）から呼び出せます。

  最外側のプログラムは，複数の内側のプログラムを含むことができます。

- 内側のプログラム

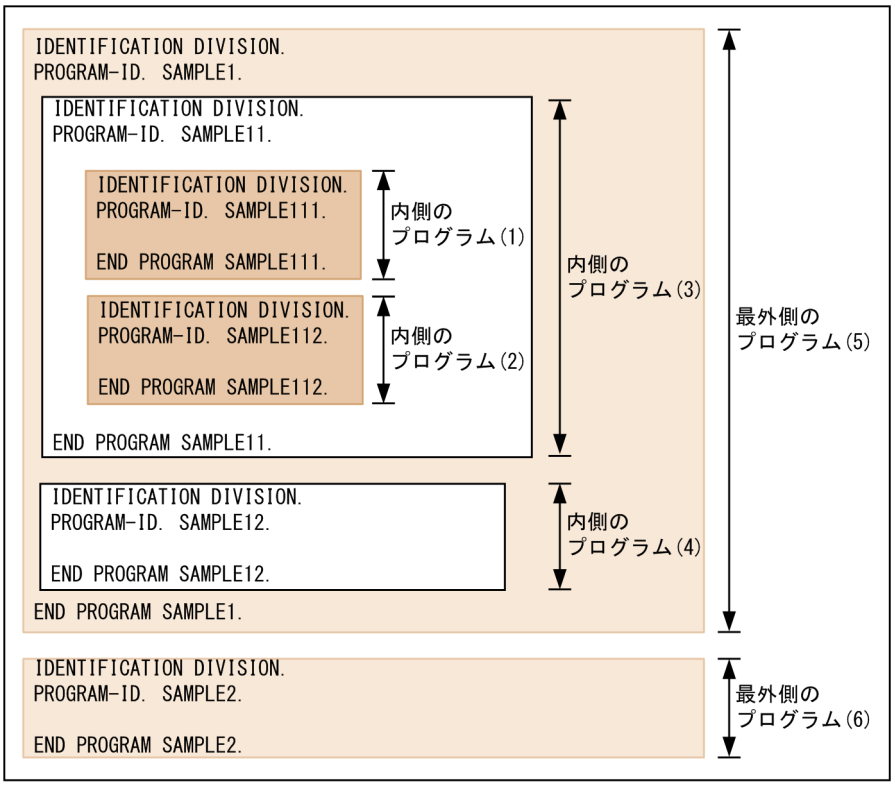
  ほかのプログラムに含まれるプログラムです。

  内側のプログラムは，そのプログラムを呼び出せる位置にある最外側のプログラム，または内側のプログラムから呼び出せます。また，プログラム名段落で指定したプログラム名称が，オブジェクトファイル中に外部参照として出力されないため，別の翻訳単位のプログラムからは呼び出せません。

  内側のプログラムは，複数の内側のプログラム（入れ子のプログラム）を含むことができます。

一つの翻訳グループ中には，複数の最外側のプログラムを記述できます。この翻訳グループを翻訳する場合，スタックコンパイル（連続コンパイル）が実行されます。翻訳グループに複数の最外側のプログラムおよび内側のプログラムを記述した場合の翻訳単位の考え方を次に示します。

図 3-2　翻訳単位の考え方



翻訳グループ（SAMPLE1.CBL）

内側のプログラム(1)，(2)

　内側のプログラム(3)に直接含まれる内側のプログラムです。

内側のプログラム(3)，(4)

　最外側のプログラム(5)に直接含まれる内側のプログラムです。

最外側のプログラム(5)，(6)

　翻訳グループ（SAMPLE1.CBL）に含まれる最外側のプログラムです。

このプログラム定義では，一つの翻訳グループ（SAMPLE1.CBL）に二つの最外側のプログラム（最外側のプログラム(5)，(6)）が含まれているので，コンパイラを 1 回起動すると，1 回のスタックコンパイルによって二つの翻訳単位がコンパイルされます。

3.2.2　関数定義

関数定義は，見出し部で関数名段落を指定して定義します。

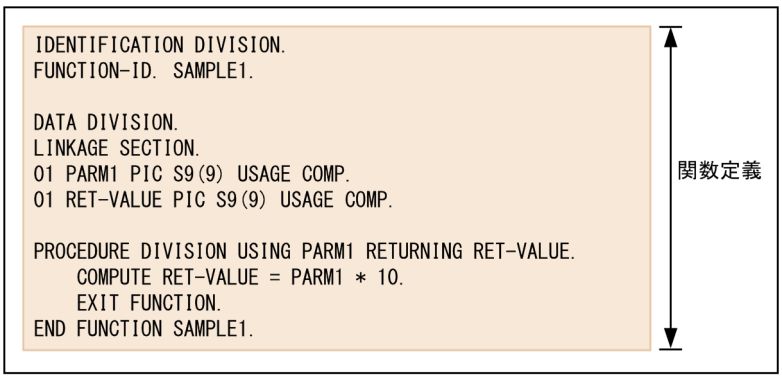
関数名段落の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「7.4　関数名段落(FUNCTION-ID)」を参照してください。

利用者定義関数の基本的な規則や動作は，内側のプログラムを持たない最外側のプログラム定義と同じです。ただし，利用者定義関数には，RETURNING 指定が必要です。また，常に再帰属性となるので，自分自身を呼び出すことができます。

利用者定義関数は，関数一意名を指定することで呼び出されます。

関数定義の例を次に示します。

図 3-3　関数定義



翻訳グループ（SAMPLE1.CBL）

3.2.3　クラス定義

クラス定義は，見出し部でクラス名段落を指定して定義します。

クラス名段落の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「7.2　クラス名段落（CLASS-ID)」を参照してください。また，クラス定義の詳細については，「20.　オブジェクト指向機能」を参照してください。

クラス定義は，ファクトリ定義，インスタンス定義，およびそれらの各メソッド定義を含むことができます。

- ファクトリ定義

  クラス定義固有のファクトリオブジェクトの型を定義します。ファクトリ定義は，ファクトリデータ定義およびファクトリメソッド定義から構成されます。すべてのクラスにはファクトリオブジェクトが一つあり，ファクトリに固有なデータとメソッドを持ちます。ファクトリオブジェクトの主な用途は，オブジェクトインスタンスの生成，およびクラスに属するすべてのインスタンスに共通するデータの管理です。

  ファクトリ名段落については，マニュアル「COBOL2002　言語　標準仕様編」「7.3　ファクトリ段落（FACTORY)」を参照してください。

- インスタンス定義

  クラス定義固有のインスタンスオブジェクトの型を定義します。インスタンス定義は，インスタンスデータ定義およびインスタンスメソッド定義から構成されます。インスタンスオブジェクトとは，それ自身に固有なデータ項目やファイル結合子から構成されるプログラムで，そのクラス定義の中で定義さ

れたメソッド群を共有します。インスタンス定義には，データの特性と，そのクラスに属する各インスタンスオブジェクトに関して呼び起こされるメソッド群を記述します。

オブジェクト段落については，マニュアル「COBOL2002　言語　標準仕様編」「7.7　オブジェクト段落（OBJECT）」を参照してください。

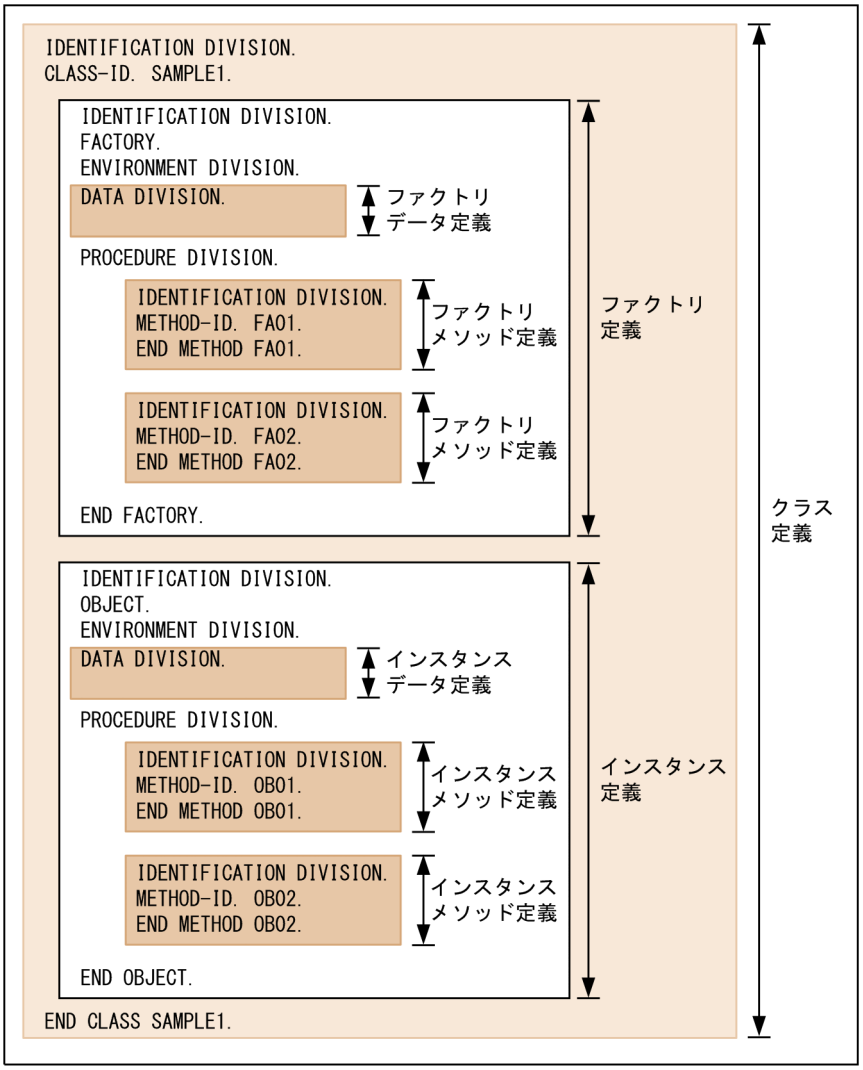
- メソッド定義

  メソッド定義は，メソッドが定義されたクラスのインスタンスオブジェクトおよびファクトリオブジェクトに含まれるデータを操作する手続き文の集まりです。メソッドは，それぞれ固有のメソッド名を持ち，固有のデータ部や手続き部を持ちます。メソッドは，インスタンスオブジェクトまたはファクトリオブジェクトを参照する一意名とそのメソッド名を指定することで呼び起こされます。メソッドには引数や返却項目を指定できます。

  メソッド名段落については，マニュアル「COBOL2002　言語　標準仕様編」「7.6　メソッド名段落（METHOD-ID）」を参照してください。

クラス定義の例を次に示します。

図 3-4　クラス定義



翻訳グループ（SAMPLE1.CBL）

3.2.4　インタフェース定義

インタフェース定義は，見出し部でインタフェース名段落を指定して定義します。インタフェース定義を使用することによって，特定のクラスに依存しないインタフェースを定義できます。

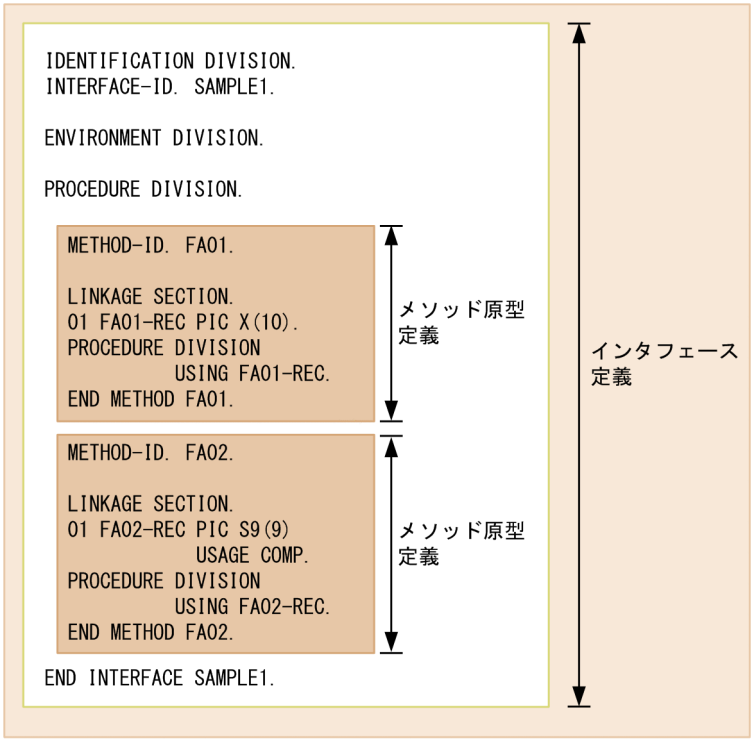
インタフェース名段落の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「7.5　インタフェース名段落（INTERFACE-ID)」を参照してください。インタフェースについては，マニュアル「COBOL2002　言語　標準仕様編」「付録Ｉ　用語の定義（Terms and Definitions)」のインタフェースの説明を参照してください。

また，インタフェース定義の詳細については，「20.　オブジェクト指向機能」を参照してください。

インタフェース定義には，メソッド原型定義を含めることができます。メソッド原型については，マニュアル「COBOL2002　言語　標準仕様編」「付録Ｉ　用語の定義（Terms and Definitions)」のメソッド原型の説明，およびマニュアル「COBOL2002　言語　標準仕様編」「7.6　メソッド名段落（METHOD-ID)」を参照してください。

インタフェース定義の例を次に示します。

図 3-5　インタフェース定義



4 COBOL プログラムのデータ領域

この章では，COBOL プログラムで使用するデータ領域について説明します。

4.1　データ領域の種類

COBOL プログラムのデータ領域とは，データ部で記述された領域を指します。

COBOL プログラムで使用できるデータ領域を次に示します。

- 連絡節のデータ領域
- 作業場所節のデータ領域
- 局所場所節のデータ領域
- ファイル節
- 報告書節
- 画面節（SCREEN SECTION）
- 画面節（WINDOW SECTION）
- 通信節
- サブスキーマ節

ソース単位の定義によって記述できるデータ定義の種別を，次に示します。

表 4-1　ソース単位の定義によって記述できるデータ定義の種別

定義の種類	連絡節	作業場所節	局所場所節	ファイル節 画面節（SCREEN SECTION／WINDOW SECTION） 通信節	報告書節	サブスキーマ節
プログラム定義	○	○	○	○	○	○
メソッド定義	○	×	○	×	×	×
関数定義	○	○	○	○	○	×
ファクトリ定義 インスタンス定義	×	○	×	○	×	×

（凡例）

○：記述できる

×：記述できない

4.1.1　連絡節のデータ領域

連絡節のデータ領域には，仮引数や返却項目を記述します。

ソース要素の連絡節のデータ領域に記述された仮引数や返却項目は，要素が呼ばれるとき，呼ばれる側の要素，呼ぶ側の要素の両方から参照されます。指標名の場合，呼ばれる側の指標名と呼ぶ側の指標名は，別の指標名（領域）を参照します。

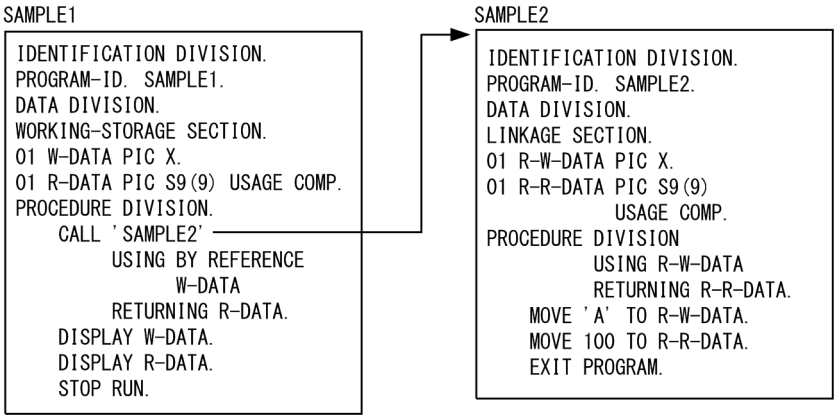
連絡節は，ソース要素が呼び出され，呼び出し先の要素の手続き部見出しに USING／RETURNING が指定されている場合だけ有効となります。

連絡節は，節の見出し，およびそれに続く 77 レベル記述項やレコード記述項から構成されます。

連絡節の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「9.6　連絡節（LINKAGE SECTION)」を参照してください。

連絡節の記述例を，次に示します。

図 4-1　連絡節の記述例



4.1.2　作業場所節のデータ領域

作業場所節のデータ領域には，ファイルの一部ではないレコード項目，およびその従属データ項目を記述します。

作業場所節中で記述されるデータは，静的データまたは初期化データです。

作業場所節は，節の見出し，およびそれに続く 77 レベル記述項やレコード記述項から構成されます。

作業場所節の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「9.4　作業場所節（WORKING-STORAGE SECTION)」を参照してください。また，静的データ，および初期化データの詳細については，マニュアル「COBOL2002　言語　標準仕様編」「10.5.2　関数，メソッド，オブジェクトまたはプログラムの状態」および「COBOL2002　言語　標準仕様編」「付録 I　用語の定義（Terms and Definitions)」を参照してください。

4.1.3　局所場所節のデータ領域

局所場所節のデータ領域には，自動データを記述します。

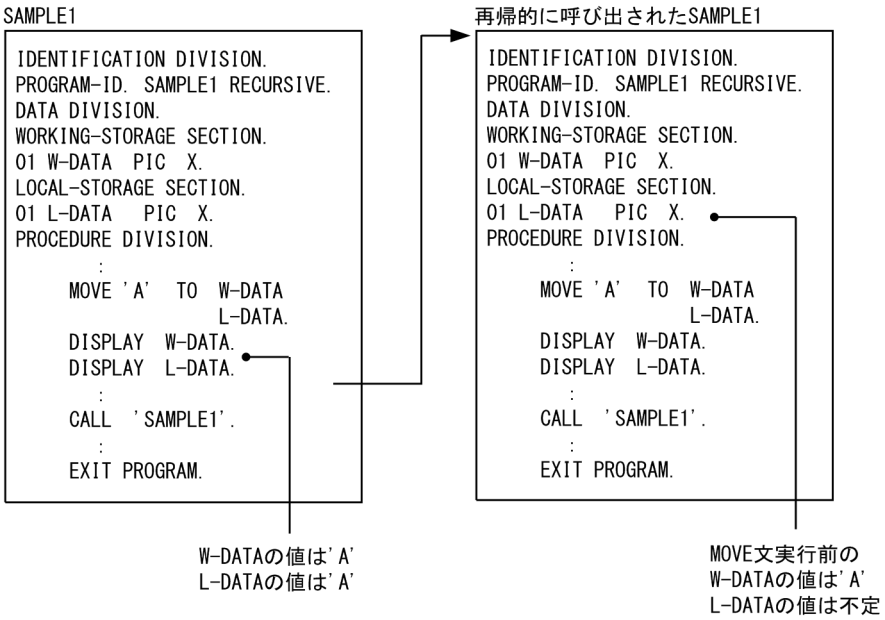
局所場所節は，プログラム定義，関数定義，メソッド定義などの再帰的に呼び出せるプログラムが呼び出されたときに，呼び出される単位ごとに確保されるデータを定義します。

局所場所節は，節の見出し，およびそれに続く 77 レベル記述項やレコード記述項から構成されます。

局所場所節の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「9.5　局所場所節(LOCAL-STORAGE SECTION)」を参照してください。

局所場所節の例を次に示します。

図 4-2　局所場所節の記述例



なお，自動データの詳細については，マニュアル「COBOL2002　言語　標準仕様編」「付録 I　用語の定義（Terms and Definitions)」を参照してください。

4.1.4　その他の節のデータ領域

その他の節に記述されるデータは，静的データまたは初期化データです。

それぞれの節の言語仕様については，次を参照してください。

節の種類	参照個所
ファイル節	「COBOL2002　言語　標準仕様編」「9.3　ファイル節（FILE SECTION)」
報告書節	「COBOL2002　言語　標準仕様編」「13.4　データ部（DATA DIVISION）(報告書作成機能)」

節の種類	参照個所
画面節（SCREEN SECTION）	「COBOL2002　言語　拡張仕様編」「12.　画面節（SCREEN SECTION）による画面機能」
画面節（WINDOW SECTION）	「COBOL2002　言語　拡張仕様編」「13.　画面節（WINDOW SECTION）による画面機能」
通信節	「COBOL2002　言語　拡張仕様編」「11.　通信節による画面機能」 「COBOL2002　言語　拡張仕様編」「8.　データコミュニケーション機能」
サブスキーマ節	「COBOL2002　言語　拡張仕様編」「19.1.1　データ部（構造型データベース（XDM/SD）操作シミュレーション機能）」

また，静的データ，および初期化データの詳細については，マニュアル「COBOL2002　言語　標準仕様編」「付録Ｉ　用語の定義（Terms and Definitions）」を参照してください。

4.2　データ属性の種類

COBOL2002 では，データやファイルを共用できます。データやファイルを共用すると，複数プログラム間でのデータやファイルの共用が容易になります。

データやファイルを共用するには，その属性が大域属性（GLOBAL 句）または外部属性（EXTERNAL 句）である必要があります。

4.2.1　大域属性（GLOBAL 句）

データ名およびファイル名は，大域名か局所名のどちらかに分類されます。

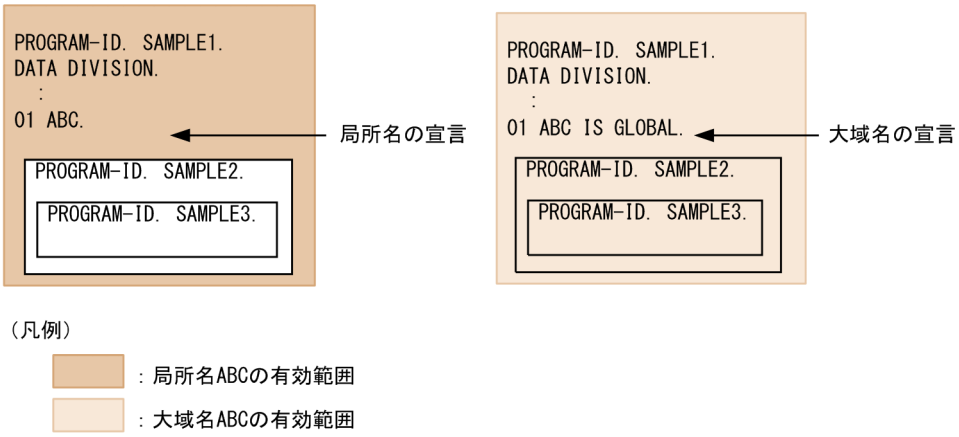
データ名やファイル名に GLOBAL 句を指定すると大域名となります。指定しない場合は，局所名となります。

局所名を使用すると，名前に関連づけられている対象を，局所名が定義されているプログラム中だけから参照できます。

大域名を使用すると，名前に関連づけられている対象を，大域名が定義されているプログラム中，およびそのプログラムに含まれるプログラム中から参照できます。大域名は複数の含まれるプログラム間で参照できますが，同じ名前が定義されているプログラムを含むプログラム中からは参照できません。

大域名および局所名の有効範囲を次に示します。

図 4-3　大域名と局所名の有効範囲



データ名やファイル名は，大域名か局所名のどちらかに分類されます。また，定義されたプログラム中の指定によって，大域属性か局所属性かが決まる名前もあります。ファイル名，レコード名，データ名，および条件名が，それぞれ局所／大域の属性になる場合を次に示します。

- ファイル名

  ファイル名が定義されているファイル記述項※に GLOBAL 句がある場合は大域名となります。大域名でない場合は局所名となります。

- レコード名
  レコード名が次のどちらかに該当する場合は大域名となります。大域名でない場合は局所名となります。
  - その名前が定義されているレコード記述項※に GLOBAL 句がある
  - GLOBAL 句があるファイル記述項※に関連している
- データ名
  データ名が次のどちらかに該当する場合は大域名となります。大域名でない場合は局所名となります。
  - その名前が定義されているデータ記述項※に GLOBAL 句がある
  - GLOBAL 句を持つほかのデータ記述項に従属している
- 条件名（データ記述項※で定義されているもの）
  条件名が定義されているデータ記述項が，GLOBAL 句があるほかの記述項に従属している場合は大域名となります。大域名でない場合は局所名となります。

注※

ファイル記述項，レコード記述項，およびデータ記述項では，GLOBAL 句の指定が規則によって禁止される場合があります。

GLOBAL 句については，マニュアル「COBOL2002　言語　標準仕様編」「9.16.30　GLOBAL 句」を参照してください。

4.2.2　外部属性（EXTERNAL 句）

EXTERNAL 句を指定したデータ名やファイル名は，外部属性を持ち，COBOL 実行単位内で一つの領域を共用できます。

COBOL 実行環境中では，次の範囲で EXTERNAL 領域を指定できます。

- ファイル数：255 個まで
- データ領域の数：32,767 まで

ファイル数，またはデータ領域の数が上限を超えた場合は，実行時エラーとなります。

EXTERNAL 句については，マニュアル「COBOL2002　言語　標準仕様編」「9.16.25　EXTERNAL 句」を参照してください。

(1)　共有可能属性と共有不可能属性

外部属性を持つデータ項目には，他言語と共有できる EXTERNAL 領域（共有可能属性），および他言語と共有できない EXTERNAL 領域（共有不可能属性）があります。

データ項目が次に示す条件のどれかに該当する場合，共有不可能属性の EXTERNAL 領域となります。

- EXTERNAL 句が指定されたデータ項目のデータ名称が日本語である
- EXTERNAL 句が指定されたデータ項目に指標名が指定されている
- EXTERNAL 句が指定されたデータ項目に DYNAMIC が指定されている
- -MultiThread オプションが指定されている
- EXTERNAL 句が指定されたファイル記述項中に定義されたデータ項目である

共有可能属性の EXTERNAL 領域に対する COBOL と C 言語との共有については，「19.1.5　外部属性を持つデータ項目の共用」を参照してください。

共有可能属性と共有不可能属性では，データ項目の初期値が異なります。

(a)　共有可能属性の初期値

- 実行可能ファイルが COBOL だけで構成されている場合
  その EXTERNAL 領域は X'00'が保証されます。
- COBOL と COBOL 以外の言語の実行可能ファイルが混在している場合
  COBOL 以外の言語で，共有する領域の初期値を設定しているかどうかによって異なります。
  - すべての外部変数が初期値なしで定義されている場合，その EXTERNAL 領域の初期値は X'00'が保証されます。
  - 外部変数が初期値ありで定義されている場合，その EXTERNAL 領域は定義された初期値となります（同じ外部変数に初期値ありが複数定義されている場合はリンクエラーとなります）。

(b)　共有不可能属性の初期値

共有不可能属性の EXTERNAL 領域の初期値は不定です。

ただし，環境変数 CBLEXVALUE に「NULL」を指定することで，EXTERNAL 領域の初期値を NULL（X'00'）に設定できます。環境変数 CBLEXVALUE は，COBOL プログラムの実行によって COBOL の実行環境中で初めて出現する共有不可能属性の各 EXTERNAL 領域に対して，一度だけ作用します。

環境変数 CBLEXVALUE の指定方法を次に示します。

形式

```
CBLEXVALUE=NULL
```

規則

- 環境変数 CBLEXVALUE は，作業場所節に定義されている共有不可能属性のデータ項目だけを初期化します。
- 初期値の指定に誤りがある場合，実行時エラーとなります。

(2) EXTERNAL 属性チェック

EXTERNAL 句を指定したデータ名やファイル名がある場合，プログラムの実行時に属性チェックをします。このとき，プログラム間で属性が一致していないと実行時エラーとなります。

EXTERNAL 属性チェックのチェック項目を次に示します。

(a) データ項目に対するチェック項目

- レコード長
- 指標数
- 指標名称
- 指標の指定順序（データ項目を記述したときの指標定義の出現順序）
- 指標の最大繰り返し数
- DYNAMIC 指定の有無

(b) ファイルに対するチェック項目

- ファイル定義名
- ASSIGN 句の内容（外部装置名または定数の内容です。データ名の場合，データ名の内容はチェックの対象外となります。）
- OPTIONAL 指定の有無
- ACCESS MODE 句の指定（アクセス法）
- ORGANIZATION 句の指定（ファイル編成）
- RESERVE 句の指定の有無，整数値
- BLOCK CONTAINS 句の指定の有無，整数 2
- CODE SET 句の有無
- LABEL RECORDS 句の指定の有無，指定内容
- LINAGE 句の指定（FOOTING，TOP，BOTTOM）の有無，整数指定時の整数値
- レコード句の VARYING IN SIZE 指定の有無
- レコード長（最小，最大）
- 主レコードキー長，主レコードキー属性，相対位置
- 副レコードキーの数，副レコードキー長，副レコードキー属性，相対位置，DUPLICATES 指定の有無
- 相対キー指定の有無，最大けた数
- コンパイラオプション
  次に示すコンパイラオプションの指定の有無

-StdVersion,1，-StdVersion,2，-Switch,EBCDIC，-Switch,EBCDIK，-XMAP,LinePrint，-IgnoreLCC，-Compati85,IoStatus，-Compati85,Linage，-NumCsv

- WRITE 文の制御文字の種別（ADVANCING／POSITIONING 指定の有無，AFTER／BEFORE 指定の混在）
- レコード名称
- ファイルに記述されたレコード数
- 各レコードの指定順序，各レコードのデータ項目属性（「4.2.2　外部属性（EXTERNAL 句）」の「(2) EXTERNAL 属性チェック」－「(a)　データ項目に対するチェック項目」を参照）
- レコード形式
- LOCK MODE 句の指定
- DECIMAL-POINT IS COMMA 句の有無（-NumCsv オプション指定時の CSV 編成ファイルだけが対象）
- CHARACTER TYPE 句の有無（WRITE 文の FROM 指定の一意名，またはその下位項目に CHARACTER TYPE 句が指定されている場合も含む）

5 手続き文

この章では，COBOL で使用する手続き文の仕様と使用例について説明します。

5.1　概要

ここでは，手続き文の概要について説明します。

5.1.1　基本的な内部操作手続き文

基本的な内部操作手続き文の分類を，次に示します。

表 5-1　基本的な内部操作手続き文の分類

分類	文	参照先
算術演算	• ADD • COMPUTE • DIVIDE • MULTIPLY • SUBTRACT	5.2　算術演算機能
文字列操作	• EXAMINE※1 • INSPECT • STRING • TRANSFORM※1 • UNSTRING	5.3　文字列操作文
条件分岐	• EVALUATE • IF	5.4　条件分岐文
表操作	• SEARCH	5.5　表操作
手続き分岐	• ALTER※1 • CONTINUE • EXIT※2 • GO TO • PERFORM	5.6　手続き分岐
データ転記	• INITIALIZE • MOVE • SET※2	5.7　データ転記文
オブジェクト指向	• INVOKE※2 • SET※2	20.　オブジェクト指向機能
例外	• EXIT※2 • GOBACK※2 • RAISE • RESUME • SET	21.　共通例外処理

分類	文	参照先
	• USE※2	
入出力	• CLOSE • COMMIT※2 • DELETE • OPEN • READ • REWRITE • ROLLBACK※2 • START • UNLOCK • USE※2 • WRITE	6. ファイル入出力機能
整列併合	• MERGE • RELEASE • RETURN • SORT	11. 整列併合機能
少量入出力	• ACCEPT※2 • DISPLAY※2 • STOP※2	10. ACCEPT／DISPLAY／STOP 文による入出力
報告書機能	• GENERATE • INITIATE • SUPPRESS • TERMINATE	9. 報告書作成機能
プログラム間連絡	• CALL • CANCEL • ENTRY • EXIT※2 • GOBACK※2 • STOP※2	16. COBOL の実行単位
通信	• COMMIT※2 • DISABLE • ENABLE • RECEIVE • ROLLBACK※2 • SEND • TRANSCEIVE	22. データコミュニケーション機能
画面	• ACCEPT※2 • DISPLAY※2 • ERASE	12. 画面入出力機能（AIX で有効）

分類	文	参照先
	• REPLY • USE※2 • WAIT	
その他	• ENTER※1	−

(凡例)

−：廃要素のため，このマニュアルでは解説なし

注※1

廃要素です。

注※2

複数の機能で使用します。

5.1.2 条件式

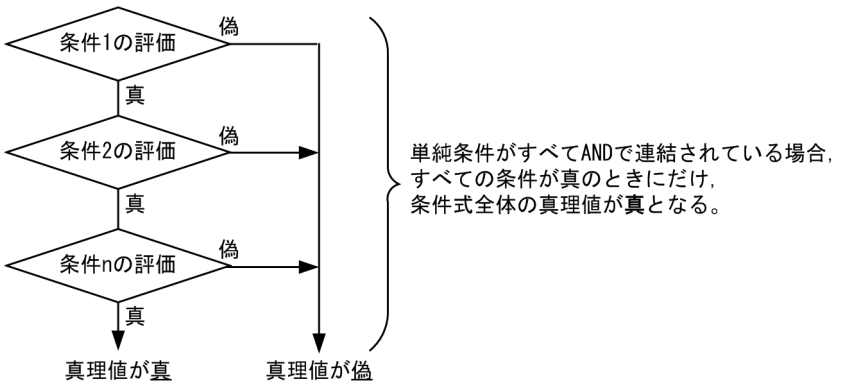
条件式の評価の流れを，幾つかの場合に分けて説明します。

条件式の文法規則については，マニュアル「COBOL2002 言語 標準仕様編」「4.7.4 条件式(Conditional expressions)」を参照してください。

(1) 条件がすべて AND で連結されている場合

条件がすべて AND で連結されている条件式の評価の流れを次に示します。

図 5-1 条件 1 AND 条件 2 AND … 条件 n の評価



この場合の条件式の例を次に示します。

(例)

```
IF A = 10 AND B = 20 AND C = 30
  THEN
    DISPLAY '--- THEN ---'
  ELSE
```

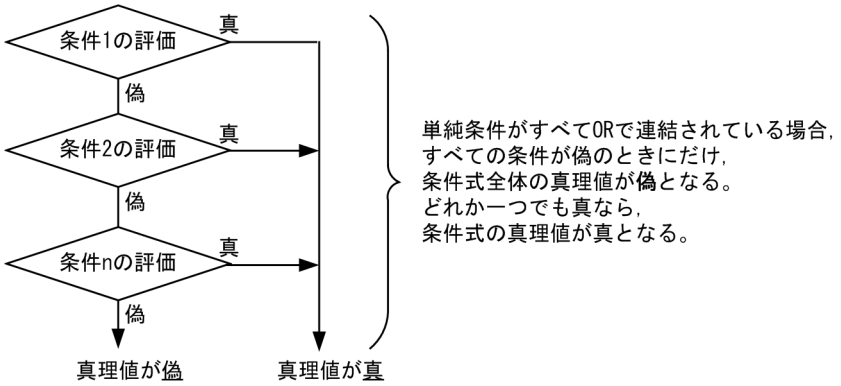
```
    DISPLAY '--- ELSE ---'
END-IF.
```

A=10 かつ B=20 かつ C=30 のときだけ THEN の DISPLAY 文を実行します。一つでも条件が真とならない場合は，ELSE の DISPLAY 文を実行します。

(2) 条件がすべて OR で連結されている場合

条件がすべて OR で連結されている条件式の評価の流れを次に示します。

図 5-2 条件 1 OR 条件 2 OR … 条件 n の評価



この場合の条件式の例を次に示します。

(例)

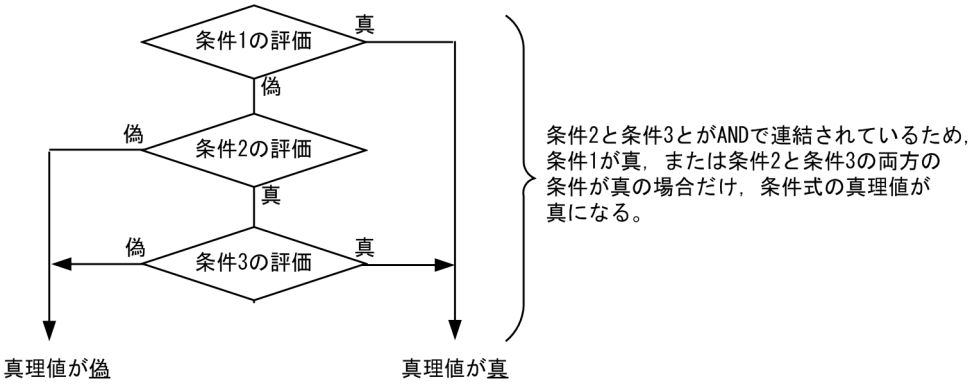
```
IF A = 10 OR B = 20 OR C = 30
  THEN
    DISPLAY '--- THEN ---'
  ELSE
    DISPLAY '--- ELSE ---'
END-IF.
```

A=10，B=20，C=30 のどれか一つでも条件が真ならば THEN の DISPLAY 文を実行します。すべての条件が偽の場合にだけ，ELSE の DISPLAY 文を実行します。

(3) 条件が OR と AND で連結されている場合

条件が OR と AND で連結されている条件式の評価の流れを次に示します。

図 5-3　条件 1 OR 条件 2 AND 条件 3 の評価



この場合の条件式の例を次に示します。

(例)

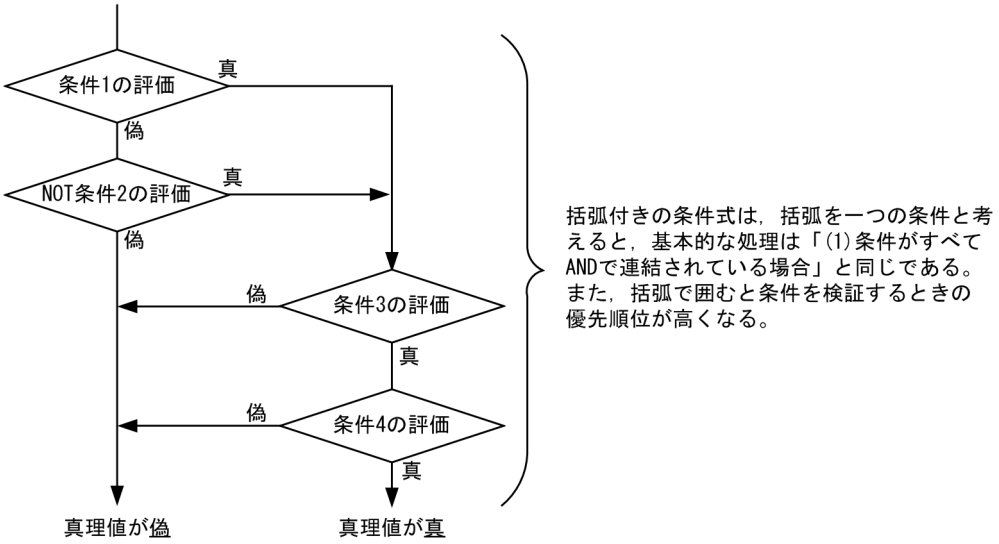
```
IF A = 10 OR B = 20 AND C = 30
  THEN
    DISPLAY '--- THEN ---'
  ELSE
    DISPLAY '--- ELSE ---'
END-IF.
```

A=10，または B=20 かつ C=30 のどちらか一つでも条件が真ならば THEN の DISPLAY 文を実行します。すべての条件が偽の場合にだけ，ELSE の DISPLAY 文を実行します。

(4)　条件が括弧で囲まれている場合

条件が括弧で囲まれている条件式の評価の流れを次に示します。

図 5-4　(条件 1 OR NOT 条件 2)　AND 条件 3 AND 条件 4 の評価



この場合の条件式の例を次に示します。

(例)

```
IF ( A = 10 OR NOT B = 20 ) AND C = 30 AND D = 40
  THEN
    DISPLAY '--- THEN ---'
  ELSE
    DISPLAY '--- ELSE ---'
END-IF.
```

括弧で囲まれている条件，C=30 および D=40 が AND で連結されているので，これらの条件がすべて真ならば，条件式の真理値は真となります。どれか一つでも偽となれば，条件式の真理値は偽となります。

次に括弧内の条件について検証します。A=10 と NOT B=20 は OR で連結されているので，どちらか一方の条件が真ならば，括弧内の条件は真となります。ここで，NOT は条件 B=20 の真理値を否定しているため，B=20 が真ならば NOT B=20 は偽，B=20 が偽ならば NOT B=20 は真となります。

5.1.3 中間結果の作成条件

コンパイラは，算術式の作用対象の演算順序などによって，一つ以上の中間結果を作成します。この中間結果は記憶装置の一部またはレジスタ上に作成され，算術演算の結果が出るまで保留されます（演算結果を連続して使用する場合，記憶装置内の同じ場所を使用することがあります）。

中間結果は次の場合に作成されます。

- ADD 文，SUBTRACT 文，MULTIPLY 文，DIVIDE 文，および COMPUTE 文
- EVALUATE 文，IF 文，PERFORM 文，および SEARCH 文中に算術式が含まれるとき
- 組み込み関数の引数，添字，部分参照中に算術式が含まれるとき

中間結果の作成例を次に示します。

(例)

```
COMPUTE A = B + (C / D) + ((E ** F) * G) - H
```

上記の COMPUTE 文は，次に示す連続した単一の演算操作に変換されます。

```
C / D      → IR1
E ** F     → IR2
IR2 * G    → IR3
B + IR1    → IR4
IR4 + IR3  → IR5
IR5 - H    → A
```

IRn（n は数字）は，コンパイラによって作成された中間結果の記憶場所を表します。

結果項目 A の精度は，作成される中間結果の整数部および小数部のけた数の影響を受けます。

なお，詳細は「5.2　算術演算機能」を参照してください。

5.1.4 CORRESPONDING 指定

CORRESPONDING 指定は，ADD 文，SUBTRACT 文，および MOVE 文で使用します。

CORRESPONDING 指定については，マニュアル「COBOL2002　言語　標準仕様編」「10.6.5 CORRESPONDING 指定」を参照してください。

CORRESPONDING 指定をすると，一つの文で集団項目中の幾つもの「対応する」項目間の加減算や転記ができます。

CORRESPONDING を指定した MOVE 文，ADD 文，または SUBTRACT 文で，対応する項目が一つもない場合は，転記が行われません。

対応とは，D1 と D2 をそれぞれ集団項目の一意名とするときに，D1 と D2 から一つずつとったデータ項目の組が，次の条件を満たす場合を指します。

- D1 に従属するデータ項目と D2 に従属するデータ項目に FILLER 以外の同じデータ名があり，D1 と D2 の直前までの修飾語の名前の系列が同じである。
- CORRESPONDING 指定付きの MOVE 文では，そのデータ項目の少なくとも一つは基本項目であり，結果的に行われる転記は転記の規則に従って正しいものである。

  CORRESPONDING 指定付きの ADD 文または SUBTRACT 文では，そのデータ項目の両方が基本数字データ項目である。
- D1 および D2 の記述は，レベル番号 66，77，88，または用途が指標，オブジェクトを含んでいてはならない。
- D1 または D2 に従属するデータ項目が，REDEFINES 句，RENAMES 句，OCCURS 句，または用途が指標，オブジェクトを含むとき，そのデータ項目は対応の対象とはならない。また，REDEFINES 句，OCCURS 句，または用途が指標，オブジェクトを含むデータ項目に従属するときも，対応の対象とはならない。D1 および D2 は，どちらも部分参照されてはならない。
- 上記の条件を満足する各データ項目の名前は，暗黙の修飾語の結果，一意にならなければならない。

  対応する項目の例を，次に示します。

(例 1)

```
01 X                  01 Y
  02 B1                 05 Z
    03 C1                 10 H1
      04 D1                 15 C1
      04 E1                   20 E1
      04 F1               10 B1
    03 G1                   15 C1
  02 H1                       20 D1
                              20 F1
```

- D1 同士と F1 同士は対応します。
- H1 は一方が基本項目で他方が集団項目ですが，MOVE 文では，基本項目から集団項目への転記ができるので対応しています。ただし，ADD 文および SUBTRACT 文では基本項目でなければならないので，対応しません。

- B1，C1 は集団項目なので対応しません。
- E1 は修飾語の系列が異なる（E1 OF C1 OF B1 と E1 OF C1 OF H1）ので対応しません。

つまり，(A)は(B)と同じことになります。

(A)

```
MOVE CORRESPONDING X TO Z
```

(B)

```
MOVE D1 OF C1 OF B1 OF X TO D1 OF C1 OF B1 OF Z
MOVE F1 OF C1 OF B1 OF X TO F1 OF C1 OF B1 OF Z
MOVE H1 OF X             TO H1 OF Z
```

(例 2)

```
                    01 W.
01 X                  02 Z
  02 Y                  03 A1
    03 A1               03 B1
    03 B1 OCCURS…         04 C1
      04 C1
      04 D1               04 D1
    03 E1               03 E1
    03 F1               03 F1 REDEFINES…
```

- A1 と E1 は対応します。
- B1 は OCCURS 句を含む項目，C1，D1 は OCCURS 句を含む項目に従属する項目なので，対応しません。
- F1 は REDEFINES 句を含む項目なので対応しません。

つまり，(A)は(B)と同じことになります。

(A)

```
MOVE CORRESPONDING Y TO Z
```

(B)

```
MOVE A1 OF Y TO A1 OF Z
MOVE E1 OF Y TO E1 OF Z
```

5.2　算術演算機能

ここでは，算術演算機能について説明します。

5.2.1　算術演算機能の特徴

算術文（詳細は「5.2.2　算術文」を参照）は，複数個の答えを持つことがあります。これらの文は，次のように書かれたものとして実行されます。

- 文の初期評価（添字などの評価）の一部であるすべてのデータ項目の参照を行い，これらのデータ項目に対して必要な計算や結合を行い，括弧の演算の結果を一時的な記憶場所に蓄える文（初期評価の一部である項目を表す規則は，個々の文によって異なる）
- 各単一の結果のデータ項目に，この一時的な記憶場所の値を転送したり，結び付けたりする一連の文（これらの文は，複数個の答が並べてあるのと同じ順序で，左から右に書かれているものとみなされる）

算術文の例を次に示します。なお，例中の temp は，任意の一時的な記憶場所を示します。

(例 1)

(A)と(B)は同じことを表します。

(A)

```
ADD a b c TO c d(c) e
```

(B)

```
ADD a b c GIVING temp
ADD temp TO c
ADD temp TO d(c)
ADD temp TO e
```

(例 2)

(A)と(B)は同じことを表します。

(A)

```
MULTIPLY a(i) BY i, a(i)
```

(B)

```
MOVE a(i) TO temp
MULTIPLY temp BY i
MULTIPLY temp BY a(i)
```

算術文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.2　ADD 文」，「COBOL2002　言語　標準仕様編」「10.8.8　COMPUTE 文」，「COBOL2002　言語　標準仕様編」「10.8.13　DIVIDE 文」，「COBOL2002　言語　標準仕様編」「10.8.29　MULTIPLY 文」，および「COBOL2002　言語　標準仕様編」「10.8.48　SUBTRACT 文」を参照してください。

5.2.2　算術文

算術文には，COMPUTE 文，ADD 文，SUBTRACT 文，MULTIPLY 文，および DIVIDE 文があります。各文の説明を，次に示します。

なお，例中の temp，quot，rem-tmp，および rem は，任意の一時的な記憶場所を示します。

(1)　COMPUTE 文

COMPUTE 文は，算術式の値を一つ以上のデータ項目に収めます。

COMPUTE 文による演算の例を次に示します。なお，A～D は整数項目を表します。

（例）

```
COMPUTE A = B +  C.     ←BにCを加算し，その結果をAに収める
COMPUTE A = B -  C.     ←BからCを減算し，その結果をAに収める
COMPUTE A = B *  C.     ←BにCを乗算し，その結果をAに収める
COMPUTE A = B /  C.※    ←BをCで除算し，その結果をAに収める
COMPUTE A = B ** C.     ←BをC乗し，その結果をAに収める
```

注※

剰余を求めたい場合には，次のような COMPUTE 文を組み合わせます。

```
COMPUTE A =  B / C          ←BをCで除算し，その商をAに収める
COMPUTE D =  B - C * A      ←商Aを使って剰余を求め，Dに収める
```

(2)　ADD 文

ADD 文は，幾つかの数字作用対象の和を一つ以上のデータ項目に収めます。

ADD 文による演算の例を次に示します。

（例 1）

ADD 文は，続く注記行の COMPUTE 文と同じです。

```
       ADD A TO B
      *COMPUTE B = A + B
       ADD A B C TO D
      *COMPUTE D = A + B + C + D
       ADD A B TO C D
      *COMPUTE temp = A + B
      *COMPUTE C = temp + C
      *COMPUTE D = temp + D
```

（例 2）

ADD 文は，続く注記行の COMPUTE 文と同じです。

```
       ADD A B TO C GIVING D
      *COMPUTE D = A + B + C
       ADD A B GIVING C D ROUNDED
```

```
      *COMPUTE temp = A + B
      *COMPUTE C = temp
      *COMPUTE D ROUNDED = temp
```

(例 3)

ADD 文は，続く注記行の二つの COMPUTE 文を合わせたものと同じです。

```
       05 A.
         10 D1 PIC S9(6).
         10 D2 PIC S9(3)V9(2) USAGE COMP.
       05 B.
         10 D1 PIC S9(5)V9 USAGE COMP.
         10 D2 PIC 9(5)V9(5).
          :
           ADD CORRESPONDING A TO B ROUNDED
      *    COMPUTE D1 OF B = D1 OF A + D1 OF B
      *    COMPUTE D2 OF B = D2 OF A + D2 OF B
```

(3) SUBTRACT 文

SUBTRACT 文は，幾つかの数字作用対象の和を，一つ以上のデータ項目から減じ，その結果を一つ以上のデータ項目に収めます。

SUBTRACT 文による演算の例を次に示します。

(例 1)

SUBTRACT 文は，続く注記行の COMPUTE 文と同じです。

```
       SUBTRACT A FROM B
      *COMPUTE B = B - A
       SUBTRACT A B C FROM D
      *COMPUTE D = D - (A + B + C)
       SUBTRACT A B FROM C D
      *COMPUTE temp = A + B
      *COMPUTE C = C - temp
      *COMPUTE D = D - temp
```

(例 2)

SUBTRACT 文は，続く注記行の COMPUTE 文と同じです。

```
       SUBTRACT A FROM B GIVING C
      *COMPUTE C = B - A
       SUBTRACT A B C FROM D GIVING E
      *COMPUTE E = D - (A + B + C)
       SUBTRACT A B FROM C GIVING D E ROUNDED
      *COMPUTE temp = C - (A + B)
      *COMPUTE D = temp
      *COMPUTE E ROUNDED = temp
```

(例 3)

SUBTRACT 文は，続く注記行の二つの COMPUTE 文を合わせたものと同じです。

```
       05 A.
        10 D1 PIC S9(6).
        10 D2 PIC S9(3)V9(2) USAGE COMP.
       05 B.
        10 D1 PIC S9(5)V9 USAGE COMP.
        10 D2 PIC 9(5)V9(5).
             :
           SUBTRACT CORRESPONDING A FROM B
      *COMPUTE D1 OF B = D1 OF B - D1 OF A
      *COMPUTE D2 OF B = D2 OF B - D2 OF A
```

(4) MULTIPLY 文

MULTIPLY 文は，数字作用対象同士の積を計算し，その結果を一つ以上のデータ項目に収めます。

MULTIPLY 文による演算の例を次に示します。

(例 1)

MULTIPLY 文は，続く四つの COMPUTE 文を合わせたものと同じです。

```
       MULTIPLY A BY B ROUNDED C D
      *COMPUTE temp = A
      *COMPUTE B ROUNDED = temp * B
      *COMPUTE C = temp * C
      *COMPUTE D = temp * D
```

(例 2)

MULTIPLY 文は，続く四つの COMPUTE 文を合わせたものと同じです。

```
       MULTIPLY A BY B GIVING C D ROUNDED E
      *COMPUTE temp = A * B
      *COMPUTE C = temp
      *COMPUTE D ROUNDED = temp
      *COMPUTE E = temp
```

(5) DIVIDE 文

DIVIDE 文は，一つの作用対象をほかの数字作用対象で割り，その商と剰余を一つ以上のデータ項目に収めます。

DIVIDE 文による演算の例を次に示します。

(例 1)

DIVIDE 文は，続く注記行の COMPUTE 文の組と同じです。

```
       DIVIDE A INTO B ROUNDED A C
      *COMPUTE temp = A
      *COMPUTE B ROUNDED = B / temp
      *COMPUTE A = A / temp
      *COMPUTE C = C / temp
```

(例 2)

DIVIDE 文は，続く注記行の COMPUTE 文と同じです。

```
       DIVIDE A INTO B GIVING C D ROUNDED
      *COMPUTE temp = B / A
      *COMPUTE C = temp
      *COMPUTE D ROUNDED = temp
```

(例 3)

DIVIDE 文は，続く注記行の COMPUTE 文と同じです。

```
       DIVIDE A BY B GIVING C D ROUNDED
      *COMPUTE temp = A / B
      *COMPUTE C = temp
      *COMPUTE D ROUNDED = temp
```

(例 4)

DIVIDE 文は，続く注記行の COMPUTE 文の組と同じです。

```
       DIVIDE A INTO B GIVING C ROUNDED REMAINDER D
      *COMPUTE quot = B / A
      *COMPUTE rem-tmp = quot.
      *COMPUTE rem = B - A * rem-tmp.
      *COMPUTE C ROUNDED = quot
      *COMPUTE D = rem
```

(例 5)

DIVIDE 文は，続く注記行の COMPUTE 文と同じです。

```
       DIVIDE A BY B GIVING C ROUNDED REMAINDER D
      *COMPUTE quot = A / B
      *COMPUTE rem = A - B * quot
      *COMPUTE C ROUNDED = quot
      *COMPUTE D = rem
```

5.2.3　有効けた数

算術文の有効けた数に関する共通事項を次に示します。

なお，Linux(x64)の場合，数字項目のけた拡張機能での有効けた数については，「28.2.3　数字項目のけた拡張機能での有効けた数」を参照してください。

- 作用対象のデータ記述が同じである必要はありません。
  計算の過程で，すべての必要な変数と小数点の位置が合わせられます。
- 作用対象の数字の最大けた数は 18 けたです。
  また，作用対象の合成※の最大けた数も 18 けたです。

注※

作用対象の合成とは，指定された各作用対象を小数点で位置合わせし，重ね合わせてできる仮想のデータ項目です。

- 各作用対象のけた数の合計は，想定した位取りも含めて 30 けた以内です。

5.2.4　演算の中間結果

中間結果のけた数は，次の規則に従って確保されます。

なお，Linux(x64)の場合，数字項目のけた拡張機能での演算の中間結果については，「28.3　数字項目のけた拡張機能での演算の中間結果」を参照してください。

- 演算の結果，上位けたや下位けたに切り捨てが発生しないけた数を確保します。
- 除算の場合，割り切れるとは限らないので，小数部のけた数は算術文中に現れる項目の最大のものとします。

  除数の PICTURE 句で，999PPV のように小数点の左側に P がある場合には，演算速度の効率を考慮して，P のけた数を商の中間結果に加えて大きくします。

  ここで確保されないけたに入る値は，切り捨てられます。
- べき乗の場合（右辺（べき数）が一意名のとき），演算結果の上位けたや下位けたがどこまで大きくなるかわからないため，中間結果を 30 けたとして，小数部のけた数は算術式中に現れる項目の小数部の最大値とします。

  ここで確保されないけたに入る値は，切り捨てられます。
- 中間結果の取れるけた数は，最大の 30 けたとし，これを超えた場合は補正をします。

  このシステムで適用する中間結果の計算式を「表 5–2　中間結果のけた数（加減算）」および「表 5–3　中間結果のけた数（乗算，除算，べき乗）」に示します。ただし，作用対象に数字定数がある場合，中間結果はその計算結果の取れる最大値を格納できるけた数を確保します。

表 5–2　中間結果のけた数（加減算）

演算の種類	整数部のけた数	小数部のけた数
加減算が連続($a_1 \pm a_2 \pm \ldots \pm a_n$)していて，かつ整数けたが最大 9 けた以下の場合	$Max(I_1,I_2,\cdots,I_n) + i$※	$MAX(D_{n-1},D_n)$
上記以外の場合	$Max(I_{n-1},I_n) + 1$	

（凡例）

I_1：連続する加減算の最初の数の整数部のけた数

I_{n-1}：演算の左辺の整数部のけた数

D_{n-1}：演算の左辺の小数部のけた数

I_n：演算の右辺の整数部のけた数

D_n：演算の右辺の小数部のけた数

注※

i は，次に示す式を満足する整数を表します。

$\log_{10}(n) \leqq i \leqq \log_{10}(n-1) + 1$

i を加算することで，中間結果の整数部のけた数が 9 けたを超えた場合，整数部のけた数の算出方法が「上記以外の場合」に変わります。

表 5-3　中間結果のけた数（乗算，除算，べき乗）

演算の種類	整数部のけた数（I）	小数部のけた数（D）
乗算	$I_1 + I_2$	$D_1 + D_2$
除算	$I_1 + D_2$	MAX (D_1 - D_2,Dmax)
べき乗	右辺が整数の数字直定数（L≠0）のとき $I_1 \times L$ 右辺が整数の数字直定数 (L ＝ 0)のとき 1 右辺が上記以外のとき 30 - Dmax	右辺が整数の数字直定数（L≠0）のとき $D_1 \times L$ 右辺が整数の数字直定数（L ＝ 0）のとき 0 右辺が上記以外のとき Dmax

（凡例）

I：中間結果の整数部のけた数

D：中間結果の小数部のけた数

I_1：演算の左辺の整数部のけた数

D_1：演算の左辺の小数部のけた数

I_2：演算の右辺の整数部のけた数

D_2：演算の右辺の小数部のけた数

Df：受け取り側作用対象の小数部のけた数

Dmax：算術式中のすべての作用対象と結果項目（ROUNDED 指定ありの場合は Df ＋ 1）の中で最大の小数部のけた数（算術式が条件式中に書かれたときは比較の作用対象を含む。また，べき数と除数の作用対象と浮動小数点数のものは除く。）

MAX(x,y)：x と y の最大数

L：整数の数字直定数の値

表 5-4　中間結果が 30 けたを超える場合の補正値

補正条件		最終的な整数部	最終的な小数部
D の値	I の値		
D ≦ Dmax	(任意)	30 - D	D
D ＞ Dmax	I ＋ Dmax ≦ 30	I	30 - I
	I ＋ Dmax ＞ 30	30 - Dmax	Dmax

注

表中の I と D は，「表 5-3　中間結果のけた数（乗算，除算，べき乗）」で計算された値です。

- PICTURE 句で整数部に P の指定がある場合，小数部のけた数は 0 として計算します。小数部に P の指定がある場合は，整数部は負として計算します。例えば，999PPV の整数部のけた数は 5 で，小数部のけた数は 0 です。また，中間結果のけた数（I ＋ D）が 30 けたを超えた場合，「表 5-2　中間結果

のけた数（加減算）」と「表 5-3　中間結果のけた数（乗算，除算，べき乗）」の値は，「表 5-4　中間結果が 30 けたを超える場合の補正値」のように補正されます。

演算の中間結果のけた数を求める例を次に示します。

（例）

```
77 A PIC S9(3)V9(3).
77 B PIC S9(4)V9(3).
77 C PIC S9(3)V9(2).
77 D PIC S9(7)V9(4).
    :
    COMPUTE D = C + (A / B).
```

上記の COMPUTE 文を実行すると，中間結果のけた数は次のようになります。

除算 A/B の中間結果のけた数

整数部：$I=I_1+D_2=3+3=6$（けた）

小数部：$D=MAX(D_1-D_2,Dmax)=MAX(0,4)=4$（けた）

加算 C+(A/B)の中間結果のけた数

除算 A/B の中間結果のけた数は，上記の計算式から，$I_2=6$，$D_2=4$ として計算します。

整数部：$I=MAX(I_1,I_2)+1=MAX(3,6)+1=7$（けた）

小数部：$D=MAX(D_1,D_2)=MAX(2,4)=4$（けた）

演算の中間結果についての注意事項

- 次の場合，浮動小数点演算となるため，中間結果のけた数の計算式は適用されません。

1. 浮動小数点項目や浮動小数点数字定数が算術演算の対象に指定されている場合
2. 被べき数（底）が浮動小数点項目であるか，またはべき数（指数）が整数でない場合
3. 関数値のデータ属性が内部浮動小数点形式の組み込み関数の場合
4. 浮動小数点項目，べき乗が含まれる算術式，または除算が含まれる算術式を引数に指定した組み込み関数の場合

- 中間結果が 30 けたを超えると，切り捨てが行われるため，正しい値が求まらないことがあります。そのため，中間結果が 30 けたを超える計算をしてはいけません。
- 中間結果の計算式は，COBOL の規格で規定されていないため，このシステムが独自に規定したものです。

5.3　文字列操作文

文字列操作文は，文字をつなぎ合わせたり，分解したり，置き換えたりします。

文字列操作文には，STRING 文，UNSTRING 文，および INSPECT 文があります。

5.3.1　STRING 文

STRING 文は，幾つかのデータ項目の内容の一部または全部をつなぎ合わせて，一つのデータ項目に移します。

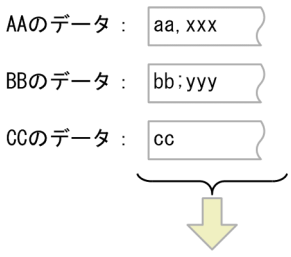
STRING 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.47　STRING 文」を参照してください。

STRING 文の例を次に示します。

（例 1）

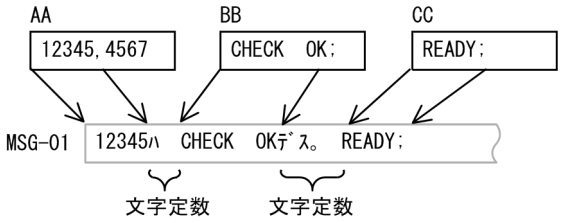
受け取り開始位置を指定しない（POINTER 指定のない）STRING 文

- 三つのデータ項目 AA，BB，CC のデータの一部または全部を取り出して，文字列を作成します。
- データ項目 AA にはコンマ（,），BB にはセミコロン（;），CC には空白（ ）がデータの区切り文字として入っていて，区切り文字までのデータを取り出して文字列を作成します。




```
77  AA        PIC  X  (10)  .
77  BB        PIC  X  (10)  .
77  CC        PIC  X  (15)  .
                :
77  MSG-01    PIC  X  (80)  .
                :
        STRING  AA      DELIMITED  BY  ','
                'ﾊ'     DELIMITED  BY  SIZE
                BB      DELIMITED  BY  ';'
                'ﾃﾞｽ。'  DELIMITED  BY  SIZE
                CC      DELIMITED  BY  SPACE
                INTO MSG-01.
```

この STRING 文を実行すると，次のようにデータを転記します。なお，文中のかたかなは，半角かたかな文字を表します。



1. AA のデータを，コンマ（,）の直前まで MSG-01 に転記します。

   ’12345’

2. 英数字定数’ハ’を，1.に続けて転記します。

   ’12345 ハ’

3. BB のデータを，セミコロン（;）の直前まで，2.に続けて転記します。

   ’12345 ハ CHECK OK’

4. 英数字定数’デス。’を，3.に続けて転記する。

   ’12345 ハ CHECK OK デス。’

5. CC のデータを，空白（ ）の直前まで，4.に続けて転記します（CC のデータの区切り文字は空白で，セミコロンはデータの一部です）。

   ’12345 ハ CHECK OK デス。 READY;’

   MSG-01 の残りの部分の内容は変更されません。

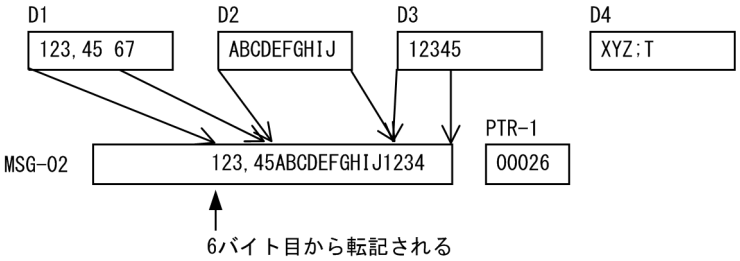
（例 2）

受け取り開始位置を指定する（POINTER 指定のある）STRING 文

- データ項目 D1，D2，D3，D4 のデータの一部または全部を取り出し，MSG-02 の領域がいっぱいになるまで転記して文字列を作成します。
- MSG-02 がいっぱいかどうかの判定には，OVERFLOW 指定を使用します。
- 送り出し側作用対象のデータ項目からは，先頭から最初の空白の直前までのデータを取り出します。
- 受け取り開始位置は，POINTER 項目の初期値で指定します。

```
       77 D1     PIC X(10).
       77 D2     PIC X(10).
       77 D3     PIC X(10).
       77 D4     PIC X(10).
       77 PTR-1  PIC 9(5) USAGE COMP.
       77 MSG-02 PIC X(25).
                      :
           MOVE 6     TO PTR-1.
           MOVE SPACE TO MSG-02.
           STRING D1 D2 D3 D4
             DELIMITED BY SPACE
             INTO MSG-02
             WITH POINTER PTR-1
             ON OVERFLOW
               DISPLAY  ’OVERFLOW CONDITION;’
           END-STRING.
```

PTR-1 の初期値が 6 の場合，この STRING 文を実行すると，次のようにデータを転記します。



1. D1 のデータを，最初の空白の直前まで MSG-02 に転記します。

   PTR-1 の初期値は 6 なので，MSG-02 の 6 文字目からデータを転記し，先頭の 5 文字の内容は変更しません。

   ’_____123,45’

2. D2 のデータを続けて転記します。

   D2 のデータには空白がないので，右端の文字 J まですべて転記します。

   ’_____123,45ABCDEFGHIJ’

3. D3 のデータを続けて転記します。

   D3 のデータを 4 文字転記すると受け取り側作用対象がいっぱいになるので，転記を終了して，OVERFLOW 指定に書かれた DISPLAY 文を実行します。D3 の残りのデータと D4 のデータは使用しません。

   PTR-1 には，転記した文字数が加えられています（この場合，MSG-02 のサイズ+1=26 が設定されています)。

5.3.2 UNSTRING 文

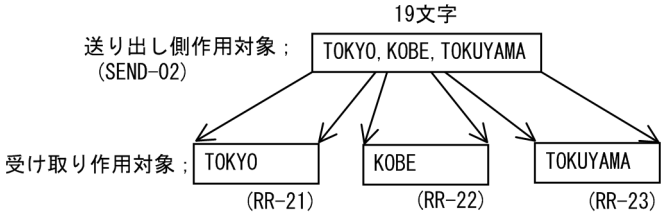
UNSTRING 文は，送り出し側作用対象の連続したデータを分解し，幾つかの受け取り側作用対象データ項目に移します。

UNSTRING 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.52　UNSTRING 文」を参照してください。

UNSTRING 文の使用例を次に示します。

(例 1)

コンマで区切られている 19 文字のデータを分解して転記します。

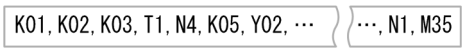


```
       77 SEND-02 PIC X(19).
       77 RR-21   PIC X(8).
       77 RR-22   PIC X(8).
       77 RR-23   PIC X(8).
             :
           UNSTRING SEND-02 DELIMITED BY ','
                    INTO RR-21 RR-22 RR-23.
```

(例 2)

- 100 バイトのデータ項目中にコンマで区切られたデータが入っているときに，データ項目のデータを 3 個ずつ取り出して転記します。
- すべてのデータを処理したかどうかの判定には，OVERFLOW 指定を使用します。

送り出し側作用対象：



受け取り側作用対象

実行	結果				
	AA の値	BB の値	CC の値	CTR の値	OVERFLOW 条件
1 回目の実行	K01	K02	K03	3	成立する
2 回目の実行	T1	N4	K05	3	成立する
:	:	:	:	:	:
n 回目の実行	N1	M35		2	成立しない

```
       77 SEND-03 PIC X(100).
       77 AA      PIC X(5).
       77 BB      PIC X(5).
       77 CC      PIC X(5).
       77 SW-1    PIC 9(1) VALUE 0.
       77 PTR-1   PIC 9(3) VALUE 1.
       77 CTR     PIC 9(3) VALUE 0.
                    :
           PERFORM TEST AFTER UNTIL SW-1 NOT = 0
             MOVE SPACE TO AA BB CC
             MOVE 0     TO SW-1, CTR
             UNSTRING SEND-03
                DELIMITED BY ',' OR ALL SPACE
                INTO AA, BB, CC,
                WITH POINTER PTR-1
                TALLYING IN CTR
                NOT ON OVERFLOW MOVE 1 TO SW-1
             END-UNSTRING
              :
             AA, BB, CCを使った処理をする
              :
           END-PERFORM.
                :
```

注

データの個数は CTR に入っています。

5.3.3 INSPECT 文

INSPECT 文は，データ項目中の文字や文字列の出現回数を数えたり，それらをほかの文字列で置き換えたりします。

INSPECT 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.25　INSPECT 文」を参照してください。

INSPECT 文の例を次に示します。COUNT-n は，その文の実行の直前にゼロになっているものと仮定します。

(例 1)

実行例

```
INSPECT ITEM TALLYING
    COUNT-0 FOR ALL 'AB', ALL 'D'
    COUNT-1 FOR ALL 'BC'
    COUNT-2 FOR LEADING 'EF'
    COUNT-3 FOR LEADING 'B'
    COUNT-4 FOR CHARACTERS.

INSPECT ITEM REPLACING
    ALL 'AB' BY 'XY', 'D' BY 'X'
    ALL 'BC' BY 'VW'
    LEADING 'EF' BY 'TU'
    LEADING 'B' BY 'S'
    FIRST 'G' BY 'R'
    FIRST 'G' BY 'P'
    CHARACTERS BY 'Z'.
```

実行結果

ITEM の初期値	COUNT -0	COUNT -1	COUNT -2	COUNT -3	COUNT -4	ITEM の終了値
EFABDBCGABCFGG	3	1	1	0	5	TUXYXVWRXYZZPZ
BABABC	2	0	0	1	1	SXYXYZ
BBBC	0	1	0	2	0	SSVW

(例 2)

実行例

```
INSPECT ITEM TALLYING
  COUNT-0 FOR CHARACTERS
  COUNT-1 FOR ALL 'A'.
INSPECT ITEM REPLACING
```

```
    CHARACTERS BY 'Z'
    ALL 'A' BY 'X'.
```

実行結果

ITEM の初期値	COUNT-0	COUNT-1	ITEM の終了値
BBB	3	0	ZZZ
ABA	3	0	ZZZ

(例 3)

実行例

```
   INSPECT ITEM TALLYING
     COUNT-0 FOR ALL 'AB' BEFORE 'BC'
     COUNT-1 FOR LEADING 'B' AFTER 'D'
     COUNT-2 FOR CHARACTERS AFTER 'A' BEFORE 'C'.
   INSPECT ITEM REPLACING
     ALL 'AB' BY 'XY' BEFORE 'BC'
     LEADING 'B' BY 'W' AFTER 'D'
     FIRST 'E' BY 'V' AFTER 'D'
     CHARACTERS BY 'Z' AFTER 'A' BEFORE 'C'.
```

実行結果

ITEM の初期値	COUNT-0	COUNT-1	COUNT-2	ITEM の終了値
BBEABDABABBCABEE	3	0	2	BBEXYZXYXYZCABVE
ADDDDC	0	0	4	AZZZZC
ADDDDA	0	0	5	AZZZZZ
CDDDDC	0	0	0	CDDDDC
BDBBBDB	0	3	0	BDWWWDB

(例 4)

実行例

```
   INSPECT ITEM TALLYING
     COUNT-0 FOR ALL 'AB' AFTER 'BA' BEFORE 'BC'.
   INSPECT ITEM REPLACING
     ALL 'AB' BY 'XY' AFTER 'BA' BEFORE 'BC'.
```

実行結果

ITEM の初期値	COUNT-0	ITEM の最終値
ABABABABC	1	ABABXYABC

(例 5)

実行例

```
  INSPECT ITEM CONVERTING
    'ABCD' TO 'XYZX' AFTER QUOTE BEFORE '#'.
```

上記の INSPECT 文は，次の INSPECT 文と同じです。

```
    INSPECT ITEM REPLACING
      ALL 'A' BY 'X' AFTER QUOTE BEFORE '#'
      ALL 'B' BY 'Y' AFTER QUOTE BEFORE '#'
      ALL 'C' BY 'Z' AFTER QUOTE BEFORE '#'
      ALL 'D' BY 'X' AFTER QUOTE BEFORE '#'.
```

実行結果

ITEM の初期値	ITEM の最終値
AC'AEBDFBCD#AB'D	AC'XEYXFYZX#AB'D

5.4 条件分岐文

条件分岐文には，EVALUATE 文および IF 文があります。

5.4.1 EVALUATE 文

EVALUATE 文は，多枝分岐，多枝結合の構造を記述し，複数の条件を評価できます。プログラムが次に取る動作は，これらの評価の結果によって決まります。

EVALUATE 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.18　EVALUATE 文」を参照してください。

EVALUATE 文の例を次に示します。

(例 1)

各選択対象の組中の選択対象の数は，選択主体の数と等しくする必要があります。

```
           :
EVALUATE X ALSO Y ALSO Z
    WHEN 1 ALSO 2 ALSO ANY
      ADD A TO B
    WHEN 3 ALSO 4 ALSO 5
      ADD C TO B
    WHEN OTHER
      ADD B TO A
END-EVALUATE.
           :
```

(例 2)

```
EVALUATE A
  WHEN  1
  WHEN  3
  WHEN  5
        :
  WHEN 19
    MOVE 'ODD' TO B             *>1.
  WHEN  2
  WHEN  4
        :
  WHEN 20
    MOVE 'EVEN' TO B            *>2.
  WHEN OTHER
    MOVE 'OUT  OF  RANGE' TO B  *>3.
END-EVALUATE.
```

- 整数項目 A の値が 20 以下の奇数のとき，1.の MOVE 文が実行されます。
- 整数項目 A の値が 20 以下の偶数のとき，2.の MOVE 文が実行されます。
- 整数項目 A の値が 20 を超えるとき，または 0 以下のとき，3.の MOVE 文が実行されます。

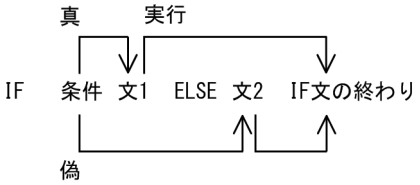
5.4.2　IF 文

IF 文は，条件を評価し，真ならば THEN の処理を，偽ならば ELSE の処理を実行します。

IF 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.23　IF 文」を参照してください。

IF 文の処理の流れを次に示します。

図 5-5　IF 文の処理の流れ



IF 文の例を次に示します。

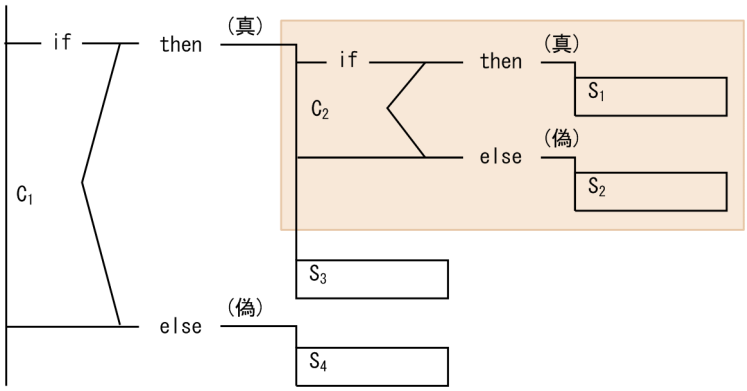
（例 1）

THEN 節と ELSE 節の両方に手続きがある場合

実行例

```
IF  C1
   THEN
      IF  C2
         THEN  S1
         ELSE  S2
      END-IF
      S3
   ELSE
      S4
END-IF
```

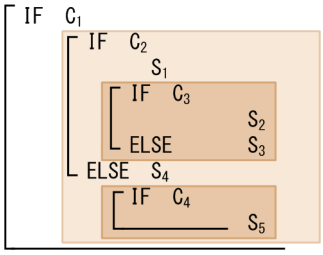
処理の流れ



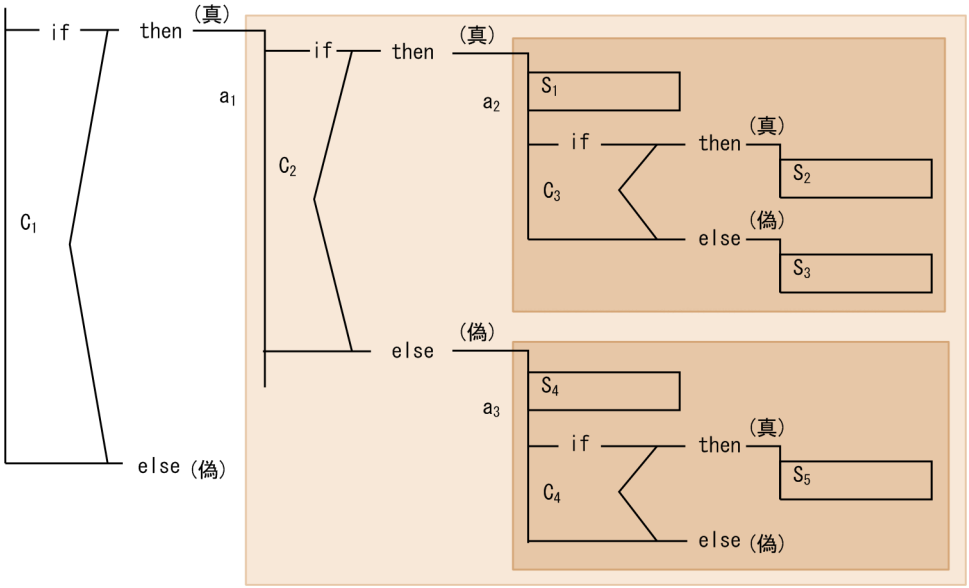
（例 2）

THEN 節だけに手続きがある場合（ELSE NEXT SENTENCE を省略したとき）

実行例



処理の流れ



- C1 に対する文 1 は a1 で，文 2 は省略されています。
- C2 に対する文 1 は a2 で，文 2 は a3 です。
- C3 に対する文 1 は S2 で，文 2 は S3 です。
- C4 に対する文 1 は S5 で，文 2 は省略されています。

省略されている ELSE NEXT SENTENCE を補うと，例 2 の実行例は次のようになります。

実行例

```
IF C1
    IF C2
                S1
      IF C3
                S2
      ELSE S3
    ELSE S4
      IF C4
                S5
      ELSE NEXT SENTENCE
ELSE NEXT SENTENCE
```

5.5　表操作

表操作には SEARCH 文があります。

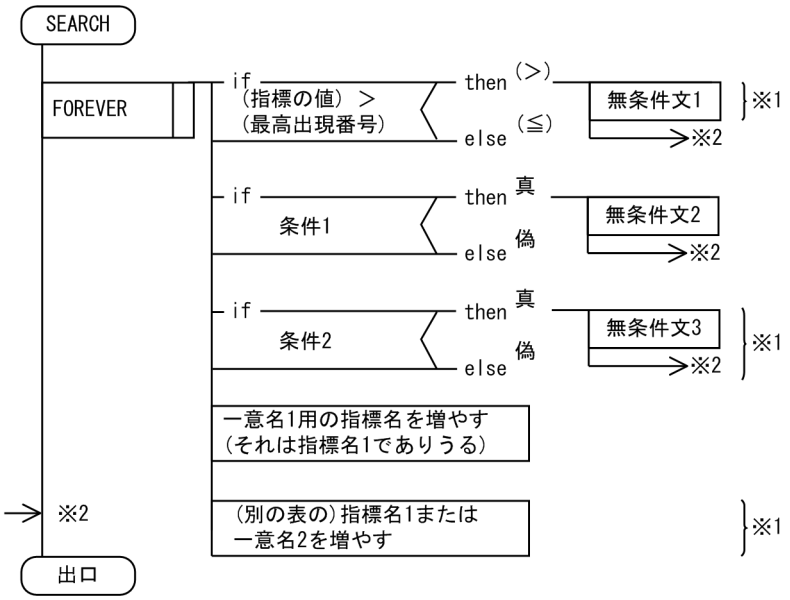
5.5.1　SEARCH 文

SEARCH 文は，指定した条件を満足する表要素を探し，対応する指標の値がその表要素を指すようにします。

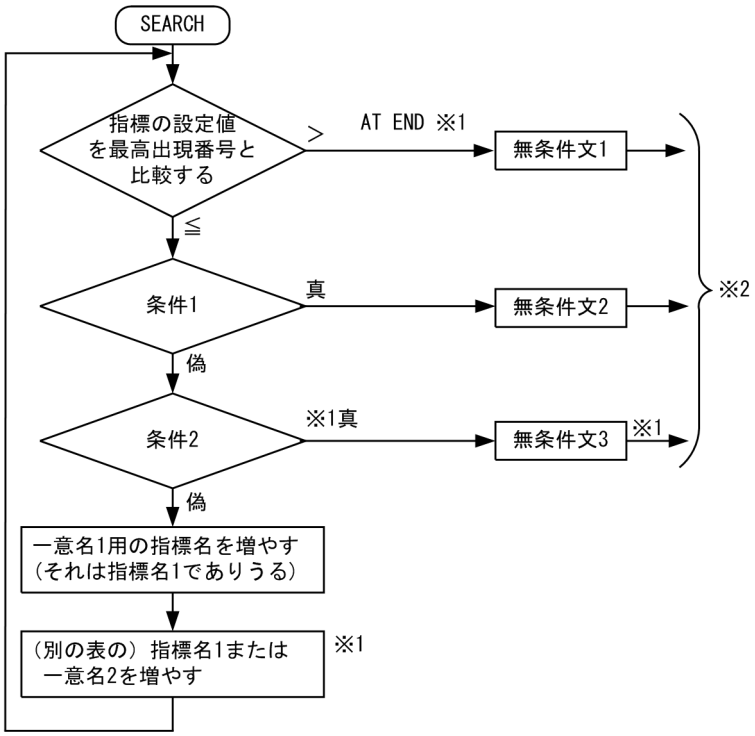
SEARCH 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.41　SEARCH 文」を参照してください。

WHEN 指定がある場合の，SEARCH 文の流れを次に示します。

図 5-6　SEARCH 文の処理の流れ（WHEN 指定が二つある場合）



流れ図



注※1

SEARCH 文中に書かれたときだけ実行されます。

注※2

GO TO 文のある無条件文の終了を必要としません。SEARCH 文の終わりに制御が移ります。

SEARCH 文の例を次に示します。

(例)

多次元の表引き操作と SEARCH 文

- SEARCH 文で参照する表が 2 次元以上の場合，各 OCCURS 句に INDEXED BY で，指標名を付けておく必要があります。
- SEARCH 文が変更するのは，一意名 1 用の指標名の値，および（VARYING 指定があれば）指標名 1 または一意名 2 だけです。
- 2 次元以上の表全体を表引きするには，各次元に SEARCH 文を何度か実行しなければなりません。

```
01 TBL.
  05 A OCCURS 10
       ASCENDING K1 INDEXED BY IX.
    10 K1 PIC X(2).
    10 B OCCURS 5
         DESCENDING K2 INDEXED BY JX.
      20 K2 PIC 9(4).
      20 C  PIC X(20).
```

K1 の値が 'AB 'で，K2 の値が 1950 のときの表要素 C を求めるには，次のようにします。

```
           SEARCH ALL A AT END GO TO OWARI
                  WHEN K1(IX) = 'AB'
                       NEXT SENTENCE.

           SEARCH ALL B AT END GO TO OWARI
                  WHEN K2(IX, JX) = 1950
                       MOVE C(IX, JX) TO X
                       GO TO OWARI.
```

5.6　手続き分岐

手続き分岐文には，GO TO 文，PERFORM 文，および CONTINUE 文があります。

5.6.1　GO TO 文

GO TO 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.21　GO TO 文」を参照してください。

GO TO 文の例を次に示します。

（例 1）

```
           MOVE A TO B
           GO TO P           *>1.
           ADD C TO D        *>2.
                :
       P.
                :
```

- 1.の GO TO 文を実行すると，制御は P に移ります。
- 2.の ADD 文は実行されません。

（例 2）

```
       01 DATA-1 PIC 9(9).
                :
            GO TO P1 P2 P3
               DEPENDING DATA-1.
                :
        P1.
                :
        P2.
                :
```

DATA-1 の値が 2 のとき，GO TO 文を実行すると，2 番目に指定した手続き名（P2）に制御が移ります。

5.6.2　PERFORM 文

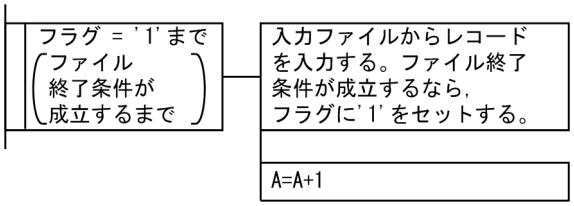
PERFORM（実行）文は，幾つかの手続きに明示的に制御を移し，指定した手続きの実行が終わると，暗黙的に制御を戻します。また，PERFORM 文は，その PERFORM 文の範囲に含まれる一つ以上の文の実行を制御するためにも使用されます。

PERFORM 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.31　PERFORM 文」を参照してください。

(1) 基本的な PERFORM 文の実行例

(a) ある条件を満たすまで処理を繰り返す場合（PERFORM UNTIL 条件）

処理の流れ



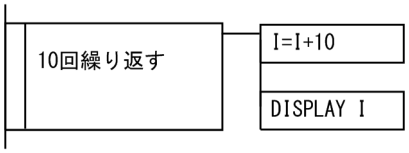
実行例

```
       PERFORM UNTIL フラグ = '1'     *>1.
         READ 入力ファイル
           AT END MOVE '1' TO フラグ  *>2.
         END-READ
         IF  フラグ = '0'
           COMPUTE A = A + 1          *>3.
         END-IF
       END-PERFORM.
```

1. ファイル終了フラグが'1'になるまで（ファイル入力が終了するまで），1.～3.の処理を繰り返します。
2. 入力ファイルからレコードを入力します。ファイル終了条件が成立すると，ファイル終了フラグに'1'をセットします。
3. ファイル終了条件が成立しない場合は，A に 1 を加算します。

(b) 指定した回数だけ処理を繰り返す場合（PERFORM 回数 TIMES）

処理の流れ



実行例

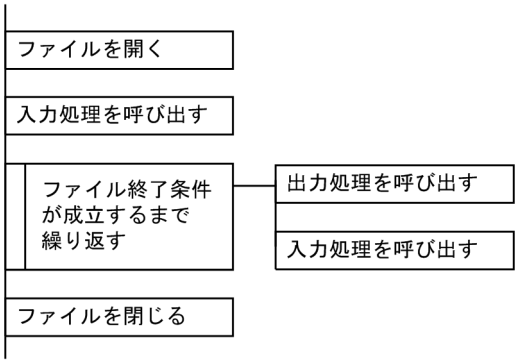
```
       PERFORM 10 TIMES      *>1.
         COMPUTE I = I + 10  *>2.
         DISPLAY I           *>3.
       END-PERFORM.
```

1. 2.と 3.を 10 回繰り返します。
2. I に 10 を加算した結果を I に入れます。
3. I を出力します。

   I が初期設定で 0 になっている場合，出力結果は，10 から始まって 10 飛びの数です。

(c) 手続きを呼び出す場合（そと PERFORM）

処理の流れ



実行例

```
       主処理.
           OPEN INPUT 入力ファイル
                OUTPUT 出力ファイル.
           PERFORM 入力処理.                    *>1.
           PERFORM UNTIL 終了フラグ = '1'       *>2.
             PERFORM 出力処理                   *>3.
             PERFORM 入力処理                   *>4.
           END-PERFORM.
           CLOSE 入力ファイル
                 出力ファイル.
           STOP RUN.
       入力処理.
           READ 入力ファイル
                AT END MOVE '1' TO 終了フラグ  *>5.
           END-READ.
       出力処理.
           MOVE 入力レコード TO 出力レコード.  *>6.
           WRITE 出力レコード.
```

1. 入力処理を呼び出します。
2. うち PERFORM で 3., 4.をファイル終了条件が成立するまで繰り返します。
3. 出力処理を呼び出します。
4. 入力処理を呼び出します。
5. 入力処理を実行します。
6. 出力処理を実行します。

(d) 条件を満たす前に PERFORM 文を終了する場合

実行例

```
           PERFORM VARYING I FROM 1 BY 1 UNTIL I > 10
               :
             PERFORM VARYING J FROM 1 BY 1 UNTIL J > 10
               :
```

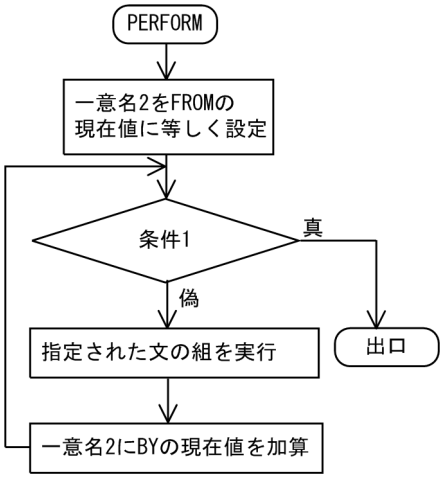
```
            IF A(J) = 0
              EXIT PERFORM  …1.
            END-IF
          END-PERFORM  …2.
            :
        END-PERFORM.
```

1.の EXIT PERFORM 文を実行すると，2.の END-PERFORM の次の文に制御が移ります。

(2) VARYING 指定のある PERFORM 文の実行例

(a) PERFORM 文に TEST BEFORE 指定があり，VARYING 指定中の条件が一つの場合（一つの一意名を変化させる場合）

処理の流れ



形式

```
        PERFORM WITH TEST BEFORE VARYING
                一意名2 FROM  {一意名3 | 定数1}
                        BY    {一意名4 | 定数2}
                        UNTIL  条件1
          無条件文1
            :
        END-PERFORM.
```

1. 一意名 2 のデータ項目の内容を定数 1，または PERFORM 文の開始時の一意名 3 のデータ項目の現在値に等しくします。
2. 条件 1 を検査します。

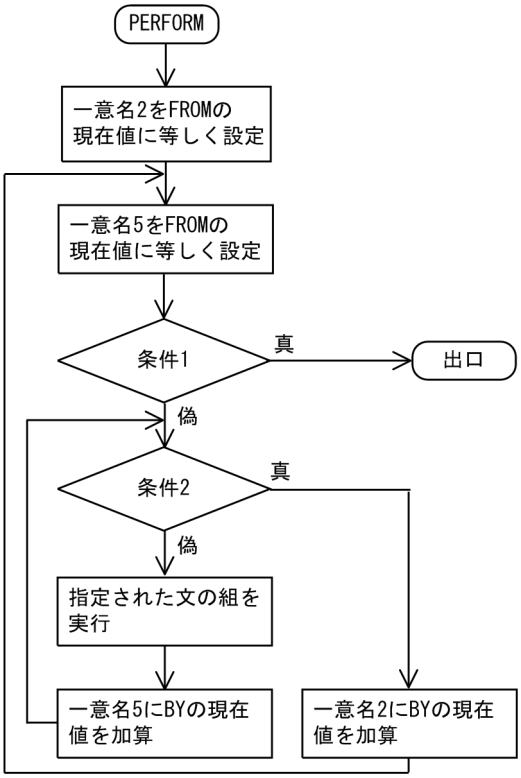
   条件 1 を満たしていれば，制御は PERFORM 文の終わりに移ります。

   条件 1 を満たしていなければ，3.に進みます。
3. 指定された文の組（無条件文 1）を 1 回実行します。
4. 一意名 2 のデータ項目の値に，定数 2 または一意名 4 のデータ項目の増加分または減少分を加え，2.に戻ります。

- PERFORM 文の実行を開始するとき，すでに条件 1 が満たされていれば，制御は PERFORM 文の終わりに移ります。
- PERFORM 文に TEST BEFORE 指定も TEST AFTER 指定も書かない場合は，TEST BEFORE 指定を書いたものとみなされます。

(b) PERFORM 文に TEST BEFORE 指定があり，VARYING 指定中の条件が二つの場合（二つの一意名のデータ項目を変化させる場合）

処理の流れ



形式

```
PERFORM WITH TEST BEFORE VARYING
        一意名2 FROM  {一意名3 | 定数1}
                BY    {一意名4 | 定数2}
                UNTIL 条件1
  AFTER 一意名5 FROM  {一意名6 | 定数3}
                BY    {一意名7 | 定数4}
                UNTIL  条件2
  無条件文1
     :
END-PERFORM
```

1. 一意名 2 のデータ項目の内容を，定数 1 または一意名 3 のデータ項目の現在値に等しくします。
2. 一意名 5 のデータ項目の内容を，定数 3 または一意名 6 のデータ項目の現在値に等しくします。
3. 条件 1 を検査します。
   条件 1 を満たしていれば，制御は PERFORM 文の終わりに移ります。

条件 1 を満たしていなければ，4.に進みます。

4. 条件 2 を検査します。

   条件 2 を満たしていれば，一意名 2 のデータ項目の内容に定数 2 または一意名 4 のデータ項目の内容を加え，2.に戻ります。

   条件 2 を満たしていなければ，5.に進みます。

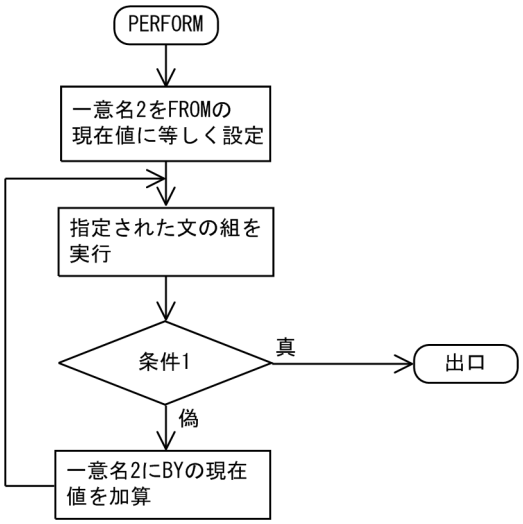
5. 指定された文の組（無条件文 1）を 1 回実行します。

6. 一意名 5 のデータ項目の内容に定数 4 または一意名 7 のデータ項目の内容を加えて，4.に戻ります。

- PERFORM 文の実行が終わったとき，一意名 5 のデータ項目には，定数 3 または一意名 6 のデータ項目の現在値が入っています。一意名 2 のデータ項目には，増加分や減少分によって最後に変更された値が入っています。
- PERFORM 文の実行を始めたときに条件 1 が満足されていると，一意名 2 のデータ項目には，定数 1 または一意名 3 のデータ項目の現在値が入っています。
- PERFORM 文に TEST BEFORE 指定も TEST AFTER 指定も書かない場合は，TEST BEFORE 指定を書いたものとみなされます。

(c) PERFORM 文に TEST AFTER 指定があり，VARYING 指定中の条件が一つの場合（一つの一意名を変化させる場合）

処理の流れ



形式

```
PERFORM WITH TEST AFTER
        VARYING 一意名2
        FROM  {一意名3 | 定数1}
        BY  {一意名4 | 定数2} UNTIL 条件1
  無条件文1
        :
END-PERFORM.
```

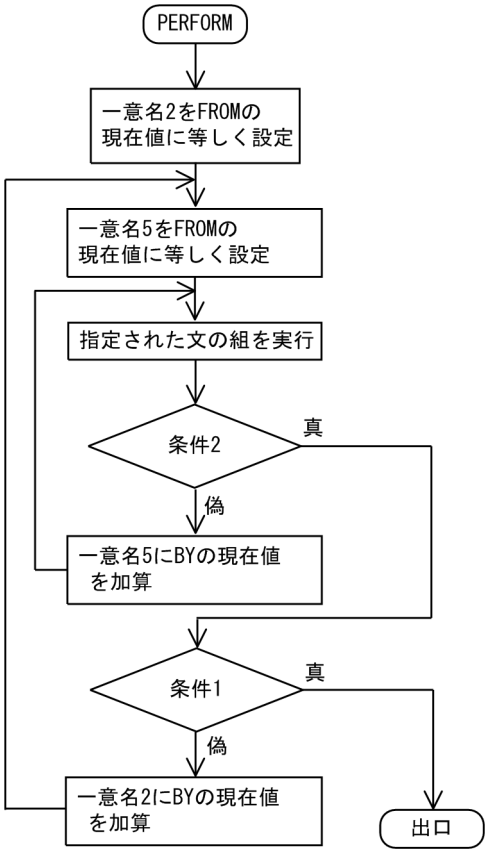
1. 一意名 2 のデータ項目の内容を，定数 1 または PERFORM 文開始時の一意名 3 のデータ項目の現在値に等しくします。
2. 指定された文の組（無条件文 1）を 1 回実行します。
3. 条件 1 を検査します。

   条件 1 を満たしていれば，制御は PERFORM 文の終わりに移ります。

   条件 1 を満たしていなければ，4.に進みます。
4. 一意名 2 のデータ項目の値に定数 2 または一意名 4 のデータ項目の値の増加分または減少分を加え，2.に戻ります。

(d) PERFORM 文に TEST AFTER 指定があり，VARYING 指定中の条件が二つの場合（二つの一意名のデータ項目を変化させる場合）

処理の流れ



形式

```
PERFORM WITH TEST AFTER
        VARYING 一意名2 FROM 一意名3
        BY 一意名4 UNTIL 条件1
        AFTER 一意名5 FROM 一意名6
        BY 一意名7 UNTIL 条件2
  無条件文1
      :
END-PERFORM.
```

1. 一意名 2 のデータ項目の内容を，定数 1 または一意名 3 のデータ項目の現在値に等しくします。
2. 一意名 5 のデータ項目の内容を，定数 3 または一意名 6 のデータ項目の現在値に等しくします。
3. 指定された文の組（無条件文 1）を 1 回実行します。
4. 条件 2 を検査します。

   条件 2 を満たしていれば，5.に進みます。

   条件 2 を満たしていなければ，一意名 5 のデータ項目の内容に定数 4 または一意名 7 のデータ項目の内容を加え，3.に戻ります。
5. 条件 1 を検査します。

   条件 1 を満たしていれば，制御は PERFORM 文の終わりに移ります。

   条件 1 を満たしていなければ，6.に進みます。
6. 一意名 2 のデータ項目の内容に定数 2 または一意名 4 のデータ項目の内容を加えて 2.に戻ります。

- PERFORM 文の実行が完全に終わったあと，AFTER 指定または VARYING 指定によって変更される各データ項目には，指定された文の組が最後に実行されたときと同じ値が入っています。
- PERFORM 文の指定された文の組の実行中に，次の項目を変更した場合，その変更が評価されて PERFORM 文の実行に影響を与えます。
  - VARYING 指定の一意名 2 のデータ項目
  - BY 指定の一意名 4 のデータ項目
  - AFTER 指定の一意名 5 のデータ項目
  - FROM 指定の一意名 3 のデータ項目
- 二つの一意名のデータ項目を変化させるとき，一意名 5 の繰り返し（FROM，BY，UNTIL）が完全に 1 回終わるたびに，一意名 2 のデータ項目の内容が変更されます。三つ以上の一意名のデータ項目を変化させるときの機構も，一意名が二つのときと同じで，AFTER 指定のデータ項目の繰り返しが完全に終わるたびに，先行する AFTER 指定のデータ項目が変更されます。

(3) PERFORM 文の実行範囲

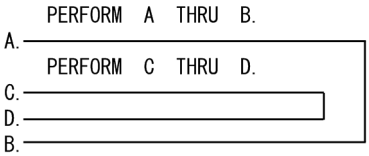
PERFORM 文の実行範囲にほかの PERFORM 文がある場合，含まれている PERFORM 文の実行範囲は，初めの PERFORM 文の論理的な実行範囲の内側に完全に含まれているか，または完全に外側である必要があります。PERFORM 文の実行範囲内で始められたほかの PERFORM 文の制御は，前者の PERFORM 文の出口を通ることはできません。また，実行中の 2 個以上の PERFORM 文は，共通の出口を持つこともできません。

PERFORM 文の詳細については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.31 PERFORM 文」を参照してください。

PERFORM 文の実行範囲の例を次に示します。

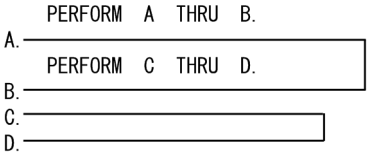
(例 1)

正しい例（完全に内側に含まれている）



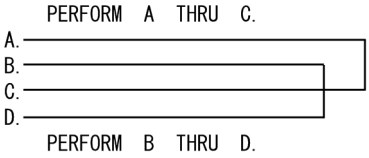
(例 2)

正しい例（完全に外側にある）



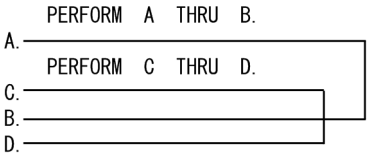
(例 3)

正しい例（同時に実行されていない）



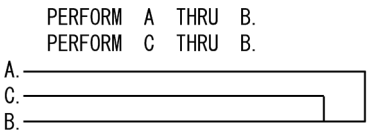
(例 4)

誤った例（後者の PERFORM 文の制御が，前者の PERFORM 文の出口を通っている）



(例 5)

正しい例（同時に実行されていない）



5.6.3 CONTINUE 文

CONTINUE 文は，条件文や無条件文などに使用し，実行文が存在しないことを示します。CONTINUE 文は，実行に影響を与えません。

CONTINUE 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.9　CONTINUE 文」を参照してください。

CONTINUE 文の例を次に示します。

(例)

```
           IF 条件1 THEN
             無条件文1
           ELSE
             CONTINUE  *>ELSE側の処理は何もないことを示す
           END-IF.
```

5.7　データ転記文

データ転記文には，INITIALIZE 文，MOVE 文，および SET 文があります。

5.7.1　INITIALIZE 文

INITIALIZE 文は，特定の型のデータ領域に，あらかじめ決められた値を設定します。例えば，数字データにはゼロを，英数字データには空白を設定できます。

INITIALIZE 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.24　INITIALIZE 文」を参照してください。

INITIALIZE 文の例を次に示します。

（例 1）

```
       01 DATAX .
         05 A      PIC A(5).
         05 B      PIC X(5).
         05 C      PIC 9999.
         05 D      PIC S9(5).
         05 E      REDEFINES D PIC X(5).
         05 F      PIC S9(3)V9(2).
         05 G      PIC 9(4)PP.
         05 H      PIC X(4)0(3)9.
         05 I      PIC A(5)BA(5).
         05 J      PIC **,**9.
         05 K      PIC 99/99.
         05 L      PIC 9(4).9(2).
         05 M      PIC +¥9(5).
         05 N      PIC ¥9(5)-.
         05 O      PIC ¥**,***CR.
         05 P      PIC ¥**,***DB.
         05 Q      PIC ZZZ.99.
         05 R      PIC ***.99.
         05 S      PIC ¥¥¥,¥¥9+.
         05 T      PIC S9(3) USAGE PACKED-DECIMAL.
         05 U      PIC S9(3) USAGE COMP OCCURS 10 TIMES.
         05 V      PIC 1(3).
         05 W      PIC 1(3) USAGE BIT.
         05 X      PIC XAX.
         05 Y      PIC N(3).
         05 Z      PIC N(2)BN(1).
         05 FILLER PIC X.
```

項目	INITIALIZE DATAX	INITIALIZE DATAX REPLACING ALPHABETIC DATA BY 'A'	INITIALIZE DATAX REPLACING ALPHANUMERIC DATA BY 'A1'	INITIALIZE DATAX REPLACING NUMERIC DATA BY 1	INITIALIZE DATAX REPLACING ALPHANUMERIC-EDITED DATA BY 'A'
A	△△△△△	A△△△△	設定しない	設定しない	設定しない
B	△△△△△	設定しない	A1△△△	〃	〃
C	符号なし外部 10 進の「0000」	〃	設定しない	符号なし外部 10 進の「0001」	〃
D	符号付き外部 10 進の「00000」	〃	〃	符号付き外部 10 進の「00001」	〃
E	設定しない	〃	〃	設定しない	〃
F	符号なし外部 10 進の「00000」	〃	〃	符号なし外部 10 進の「00100」	〃
G	符号なし外部 10 進の「0000」	〃	〃	符号なし外部 10 進の「0000」	〃
H	△△△△000	〃	〃	〃	A△△△000△
I	△△△△△△△△△△	〃	〃	〃	A△△△△△△△△△△
J	*****0	〃	〃	〃	設定しない
K	00/00	〃	〃	〃	〃
L	0000.00	〃	〃	〃	〃
M	+¥00000	〃	〃	〃	〃
N	¥00000△	〃	〃	〃	〃
O	*********	〃	〃	〃	〃
P	*********	〃	〃	〃	〃
Q	△△△.00	〃	〃	〃	〃
R	***.00	〃	〃	〃	〃
S	△△△△△¥0+	〃	〃	〃	〃
T	符号付き内部 10 進の「000」	〃	〃	符号付き内部 10 進の「001」	〃
U	各要素に 2 進形式の値 000	〃	〃	各要素に 2 進形式の値 001	〃
V	000	〃	〃	設定しない	〃
W	内部ブール形式の 0	〃	〃	〃	〃

項目	INITIALIZE DATAX	INITIALIZE DATAX REPLACING ALPHABETIC DATA BY 'A'	INITIALIZE DATAX REPLACING ALPHANUMERIC DATA BY 'A1'	INITIALIZE DATAX REPLACING NUMERIC DATA BY 1	INITIALIZE DATAX REPLACING ALPHANUMERIC-EDITED DATA BY 'A'
X	△△△	〃	A1△	〃	〃
Y	▲▲▲	〃	設定しない	〃	〃
Z	▲▲▲▲	〃	〃	〃	〃
FILLER※	設定しない	〃	〃	〃	〃

項目	INITIALIZE DATAX REPLACING NUMERIC -EDITED DATA BY 4	INITIALIZE DATAX REPLACING NATIONAL DATA BY N'花'	INITIALIZE DATAX REPLACING NATIONAL-EDITED DATA BY N'花'
A	設定しない	設定しない	設定しない
B	〃	〃	〃
C	〃	〃	〃
D	〃	〃	〃
E	〃	〃	〃
F	〃	〃	〃
G	〃	〃	〃
H	〃	〃	〃
I	〃	〃	〃
J	*****4	〃	〃
K	00/04	〃	〃
L	0004.00	〃	〃
M	+¥00004	〃	〃
N	¥00004△	〃	〃
O	¥*****4△△	〃	〃
P	¥*****4△△	〃	〃
Q	△△4.00	〃	〃
R	**4.00	〃	〃
S	△△△△△¥4+	〃	〃
T	設定しない	〃	〃

項目	INITIALIZE DATAX REPLACING NUMERIC -EDITED DATA BY 4	INITIALIZE DATAX REPLACING NATIONAL DATA BY N'花'	INITIALIZE DATAX REPLACING NATIONAL-EDITED DATA BY N'花'
U	〃	〃	〃
V	〃	〃	〃
W	〃	〃	〃
X	〃	〃	〃
Y	〃	花▲▲	〃
Z	〃	設定しない	花▲▲▲
FILLER※	〃	〃	設定しない

（凡例）

△：半角の空白を示す

▲：全角の空白を示す

注※

FILLER 句については，マニュアル「COBOL2002　言語　標準仕様編」「9.16.22　記述項名句」を参照してください。

（例 2）

オブジェクト参照を使用した INITIALIZE 文の例

```
       01 DATAY USAGE OBJECT REFERENCE.
       01 DATAZ USAGE OBJECT REFERENCE.
                :
           INITIALIZE DATAY REPLACING OBJECT-REFERENCE
                           DATA BY DATAZ.
                :
```

この場合，次の文を実行した結果と同じになります。

```
           SET DATAY TO DATAZ.
```

5.7.2　MOVE 文

MOVE 文は，編集規則に従って，データを一つ以上のデータ領域に移します。

MOVE 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.28　MOVE 文」を参照してください。

MOVE 文の例を次に示します。

（例 1）

一つまたは複数のデータを転記する場合

```
                :
       01 A PIC X(3).
```

```
       01 B PIC X(3).
       01 C PIC X(3).
       01 D PIC X(3).
       01 E PIC X(3).
                :
           MOVE A TO B.      *>1.
           MOVE A TO C D E.  *>2.
                :
```

1. データを項目 A から項目 B に転記します。
2. データを項目 A から項目 C，D，E に転記します。

(例 2)

MOVE 文の送り出し側作用対象に添字が付いている場合

添字はデータを先頭の受け取り側作用対象に移す直前に 1 回だけ評価されます。

下記の例では，(A)は(B)と同じです。

(A)

```
MOVE a(b) TO b c(b).
```

(B)

```
MOVE a(b) TO temp.
MOVE temp TO b.
MOVE temp TO c(b).
```

temp は，このシステムが用意した中間結果の項目を示します。

5.7.3 SET 文

SET 文は，次の手段を提供します。

- 表要素に関連する指標を設定して，表操作の参照点を確立する
- 外部スイッチの状態を変更する
- 条件変数の値を変更する
- オブジェクト参照を設定する
- 画面項目に関連する属性を変更する
- 最新例外状態をクリアする

SET 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.43　SET 文」，およびマニュアル「COBOL2002　言語　拡張仕様編」「13.5.8　SET 文（WINDOW SECTION)」を参照してください。

(1) 表要素に関連する指標を設定する SET 文

表要素に関連する指標を設定して，表操作の参照点を確立する SET 文の例を次に示します。

(例 1)

指標名から指標名に転記する場合

```
       05 A PIC X(5) OCCURS 10 INDEXED BY K.
       05 B PIC 9(8) OCCURS 10 INDEXED BY J.
```

指標名 K には，表の 6 番目の要素を参照するときの値，5×6=30 が入っています。ここで次の SET 文を実行すると，K の値 30 を出現番号 6 に換算して(30/5=6)，B の出現番号 6 に対応する値 8×6=48 を J に設定します。

```
       SET J TO K.
```

(例 2)

指標データ項目から指標名に転記する場合

```
       05 A PIC PP999 OCCURS 15 INDEXED BY K.
       05 B USAGE INDEX.
```

B の値が 30 のとき，次の SET 文を実行すると，B の値を出現番号に対応して換算しないで，そのまま K に設定します。

```
       SET K TO B.
```

(例 3)

整数項目または整数から指標名に転記する場合

```
       05 A PIC X(5) OCCURS 30 INDEXED BY K.
       05 B PIC 9(2) VALUE 11.
```

ここで次の SET 文のどちらかを実行すると，B の値 11 または整数 11 を出現番号とみなし，それに対応する値 5×11=55 を K に設定します。

```
       SET K TO B
```

または

```
       SET K TO 11
```

(例 4)

```
       05 A PIC X(5) OCCURS 12 INDEXED BY K.
       05 B USAGE INDEX.
       05 C USAGE INDEX.
```

K には，A の出現番号 7 に対応する値 5×7=35 が入っています。ここで次の SET 文を実行すると，K の値を変換しないで，35 をそのまま B に設定します。

```
       SET B TO K
```

そのあとで次の SET 文を実行すると，B の値を変換しないで，35 をそのまま C に設定します。

```
       SET C TO B
```

（例 5）

```
       05 A PIC 9(3) OCCURS 20 INDEXED BY K.
       05 B PIC 9(8).
```

K には，A の出現番号 12 に対応する値 3×12=36 が入っています。ここで次の SET 文を実行すると，K の値を出現番号 12 に変換(36/3)して B に 12 を設定します。

```
       SET B TO K
```

（例 6）

```
       05 A PIC X(7) OCCURS 30 INDEXED BY K.
       05 B PIC A(3) OCCURS 30 INDEXED BY J.
       05 C PIC 9(2) VALUE 3.
```

K には出現番号 6 に対応する値 7×6=42 が入っています。J には出現番号 25 に対応する値 3×25=75 が入っています。

ここで，次の SET 文を実行すると，K には出現番号 6+3=9 に対応する値 7×9=63 が入ります。

```
       SET K UP BY C
```

そのあとで次の SET 文を実行すると，K には出現番号 9-1=8 に対応する値 7×8=56 が入ります。

```
       SET K DOWN BY 1
```

また，次の SET 文を実行すると，J には出現番号 25-3=22 に対応する値 3×22=66 が入ります。

```
       SET J DOWN BY C
```

そのあとで次の SET 文を実行すると，J には出現番号 22+8=30 に対応する値 3×30=90 が入ります。

```
       SET J UP BY 8
```

(2) 外部スイッチの状態を変更する SET 文

外部スイッチの状態を変更する SET 文の例を次に示します。

（例）

```
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
       UPSI-0 IS SW0
              ON  STATUS IS ONIND0
              OFF STATUS IS OFFIND0.
             :
       PROCEDURE DIVISION.
            :
           IF OFFIND0 THEN
```

```
            SET SW0 TO ON
          END-IF.
```

外部スイッチの状態を変更する SET 文の詳細は，「16.2.4　外部スイッチ」を参照してください。

(3)　条件変数の値を変更する SET 文

条件変数の値を変更する SET 文の例を次に示します。

（例）

```
       03 OF-WEEK PIC X(3).
       88 HOLIDAY VALUE 'SUN'.  *>1.
       88 WEEKDAY VALUE 'MON' 'TUE' 'WED'
                        'THU' 'FRI' 'SAT'.  *>1.
```

1. OF-WEEK に対する条件名

ここで次の SET 文を実行すると，OF-WEEK には'SUN'が設定されます。

```
       SET HOLIDAY TO TRUE
```

また，次の SET 文を実行すると，OF-WEEK には WEEKDAY に指定された最初の定数である'MON'が設定されます。

```
       SET WEEKDAY TO TRUE
```

複数の条件名を書いたときは，その SET 文中に指定したのと同じ順序で，各条件名に対して別々の SET 文を書いたのと同じ結果となります。条件名に添字が付いている場合は，繰り返すたびに一つずつ順番に評価します。

(4)　オブジェクト参照を設定する SET 文

「20.2.7　オブジェクト指向による適合」を参照してください。

(5)　最新例外状態をクリアする SET 文

「21.7.3　最新例外状態のクリア」を参照してください。

6 ファイル入出力機能

COBOL2002 の入出力処理では，順編成ファイル，相対編成ファイル，索引編成ファイル，テキスト編成ファイル，CSV 編成ファイル，および HiRDB による索引編成ファイルが使用できます。この章では，これらのファイルに入出力するためのアクセス方法について説明します。

6.1　ファイル入出力機能の種類と概要

6.1.1　使用できるファイル編成

このシステムで使用できるファイル編成を次に示します。

表 6-1　COBOL2002 で使用できるファイル編成

ORGANIZATION 句の指定	ファイル編成	ファイル形式	ファイル管理システム	使用できるファイル
SEQUENTIAL	順編成ファイル（固定長）	バイナリデータの集合	－	• COBOL85 または COBOL2002 で作成した順編成固定長ファイル • ほかのアプリケーションで作成したレコード長の整数倍の長さを持つバイナリファイル
	順編成ファイル（可変長）	COBOL2002 が独自に定める形式のファイル	COBOL2002 独自のファイル管理システム	• COBOL85 または COBOL2002 で作成した順編成可変長ファイル※1
RELATIVE	相対編成ファイル	COBOL2002 が独自に定める形式のファイル	COBOL2002 独自のファイル管理システム	• COBOL85 または COBOL2002 で作成した相対編成ファイル
INDEXED	ISAM による索引編成ファイル	ファイル管理システムに依存する	索引順編成ファイル管理	• COBOL85 または COBOL2002 で作成した索引編成ファイル※2 • ISAM ユティリティで作成した索引編成ファイル
LINE SEQUENTIAL	テキスト編成ファイル	テキスト形式	－	• COBOL85 または COBOL2002 で作成したテキスト編成ファイル • エディタなどで作成したテキストファイル
CSV	CSV 編成ファイル	表計算プログラム用 CSV 形式	－	• COBOL85 または COBOL2002 で作成した CSV 編成ファイル • 表計算プログラムで作成した CSV ファイル
RDB	HiRDB による索引編成ファイル	ファイル管理システムに依存する	HiRDB	• HiRDB ユティリティで作成したファイル

（凡例）

－：該当しない

注※1

ただし，行制御ありで作成したファイルは使用できません。

注※2

既存の索引編成ファイルに対して OPEN OUTPUT を実行した場合，標準は追加書きとなります。詳細は，「6.6.1　ファイルの作成と割り当て方法」の「(5)　索引編成ファイルに対する OPEN モード」を参照してください。

6.1.2　使用できるファイル形式

- このシステムのファイル入出力処理で扱えるファイル形式は，それぞれのファイル編成で使用するファイル管理システムに依存します。ファイル編成とファイル管理システムの対応については，「表 6-1 COBOL2002 で使用できるファイル編成」を参照してください。
- SELECT 句の ASSIGN 句で，装置名に MT や LP を指定しても意味を持ちません。
- ファイルサイズが 2GB 以上のファイル（ラージファイル）への入出力の詳細は，「6.10　ラージファイル入出力機能」を参照してください。

  ラージファイル入出力機能を使用できるファイル編成は，次のとおりです。

  - 順編成ファイル
  - テキスト編成ファイル
  - CSV 編成ファイル

6.2　ファイル割り当ての共通規則

COBOL プログラムからファイルにアクセスするには，環境部のファイル管理記述項で，SELECT 句で指定した COBOL のファイル名に対して，ASSIGN 句を使って物理ファイル名（OS のファイルシステム上での実体ファイル名）を割り当てる必要があります。SELECT 句，ASSIGN 句の文法規則については，マニュアル「COBOL2002　言語　標準仕様編」「8.3.4　ファイル管理記述項」を参照してください。

SELECT 句で指定したファイル名に対して，物理ファイル名を割り当てる方法には，次の 3 種類があります。

1. 定数指定

   SELECT 句のファイル名に対して，ASSIGN 句で「'/DIR/FILE1.FIL'」のように物理ファイル名を直接指定する方法です。

2. 環境変数指定

   SELECT 句のファイル名に対して ASSIGN 句で「SYS001」のような外部装置名を指定しておき，実行時に環境変数を使って外部装置名に対応する物理ファイル名を割り当てる方法です。

3. データ名指定

   SELECT 句のファイル名に対して ASSIGN 句で COBOL のデータ名を指定しておき，データ名に物理ファイル名を転記して，指定する方法です。

ここでは，物理ファイル割り当て時の共通規則，それぞれの割り当て方法，およびプログラムとファイルとの関係について順に説明します。

6.2.1　定数指定

定数指定は，SELECT 句で指定したファイル名に対し，ASSIGN 句で物理ファイルを直接割り当てる方法です。

形式

```
SELECT 〔OPTIONAL〕 ファイル名 ASSIGN TO ’物理ファイル名’
```

構文規則

- 物理ファイル名は，引用符（'）で囲んで指定します。
- 物理ファイル名は，ルートからの絶対パス名を指定します。このファイル名は，ファイルシステムの規則に従って指定する必要があります。
- ディレクトリ名，ファイル名に NULL 文字（X'00'）および半角空白文字（X'20'）を含んではなりません。NULL 文字または半角空白文字を含んだディレクトリやファイル名を指定した場合，NULL 文字または半角空白文字の直前までの文字列が有効となり，以降の文字列は無視されます。

一般規則

- 次の場合，定数で指定した物理ファイルがなければ，指定した名称の物理ファイルが作成されます。

（物理ファイルが作成される場合）

1. ファイルを OUTPUT モードで開いたとき
2. SELECT 句の OPTIONAL 指定のあるファイルを I-O または EXTEND モードで開いたとき

   このとき，SELECT 句に OPTIONAL 指定がある場合は，入出力状態に 05 が設定されます。

- ファイルが作成される場所は，定数に指定した物理ファイル名の絶対パスに従います。ただし，指定した物理ファイル名中のディレクトリ名に相当するディレクトリがないと，物理ファイルは作成されません。
- 定数によって指定されるファイル名が有効となるかどうかは，ファイルシステムに依存します。
- ファイル名が絶対パス名でない場合は，OS の環境設定に従います。
- ファイルを標準入力（stdin）から読み込んだり，標準出力（stdout）または標準エラー出力（stderr）へ書き出したりしたい場合は，定数に stdin，stdout，または stderr を指定します。ただし，索引編成ファイル，相対編成ファイル，順編成の行制御のない可変長ファイルには，stdin，stdout，および stderr を指定できません。また，順編成の可変長ファイルには，stdin を指定できません。指定した場合，実行時にエラーとなります。
- 標準入出力ファイルを扱う場合，COBOL はすでに開かれているものとして処理します。
- 標準入力（stdin），標準出力（stdout），および標準エラー出力（stderr）を指定する場合は，英小文字で指定してください。「STDIN」のように英大文字で指定した場合，物理ファイル名として扱われます。

入出力状態の値とファイル自動生成規則（定数指定の場合）

定数指定でファイルを割り当てた場合，ファイル入出力時の入出力状態の値，および物理ファイルが自動作成されるかどうかは，次の規則に従います。

OPEN モード	OPTIONAL 指定の有無	物理ファイルの状態	
		すでに存在する	存在しない
INPUT	なし	FS=00	FS=35
	あり	FS=00	FS=05 自動作成されない
I-O	なし	FS=00	FS=35
	あり	FS=00	FS=05 自動作成される※1
OUTPUT	なし	FS=00 再作成する※2	FS=00 自動作成される※1
EXTEND	なし	FS=00	FS=35
	あり	FS=00	FS=05 自動作成される※1

（凡例）

FS=nn：FILE STATUS 句を指定したときに，入出力状態に nn が設定されることを示す

注※1

自動作成される物理ファイル名は，SELECT 句で指定した定数の名称となります。

注※2

再作成とは，ファイル中にデータレコードがない状態にすることです。

COBOL プログラムの記述例

プログラム内で使用するファイル名"FILE-1"を，"/DIR"ディレクトリに格納されている"FILE1.FIL"ファイルに割り当てる例を，次に示します。

```
       SELECT FILE-1 ASSIGN TO '/DIR/FILE1.FIL'
```

6.2.2　環境変数指定

環境変数指定は，指定したファイル名に対し，ASSIGN 句で外部装置名（処理系作成者語）を割り当てる方法です。外部装置名に対応する物理ファイル名は，外部装置名に対応する環境変数を使用して指定します。

COBOL プログラムでの外部装置名の記述と，環境変数の指定形式を次に示します。

形式（COBOL プログラム）

```
SELECT 〔OPTIONAL〕 ファイル名 ASSIGN TO 外部装置名
```

形式（環境変数）

```
CBL_外部装置名=物理ファイル
```

構文規則

- ASSIGN 句で指定した外部装置名に対応する環境変数は，外部装置名の先頭に CBL_を付けたものとなります。この環境変数に，外部装置名と対応づけたい物理ファイル名を指定します。
- 外部装置名に「-」が含まれる場合，環境変数名では「_」に置き換えます。
- 物理ファイル名は，ルートからの絶対パス名を指定します。このファイル名は，ファイルシステムの規則に従って指定する必要があります。
- ディレクトリ名，ファイル名に NULL 文字（X'00'）を含んではなりません。NULL 文字を含んだディレクトリやファイル名を指定した場合，NULL 文字の直前までの文字列が有効となり，以降の文字列は無視されます。

一般規則

- 次の場合，環境変数で指定した物理ファイルがなければ，指定した名称の物理ファイルが作成されます。

（物理ファイルが作成される場合）

1. ファイルを OUTPUT モードで開いたとき
2. SELECT 句の OPTIONAL 指定のあるファイルを I-O または EXTEND モードで開いたとき

このとき，SELECT 句に OPTIONAL 指定がある場合は，入出力状態に 05 が設定されます。

- ファイルが作成される場所は，環境変数に指定した物理ファイル名の絶対パスに従います。ただし，指定した物理ファイル名中のディレクトリ名に相当するディレクトリがないと，物理ファイルは作成されません。
- 外部装置名に SYSIN，SYSOUT，SYSPUNCH を割り当てた場合，次に示す文があると同一ファイルへの割り当てになります。このとき，結果は保証しませんので注意してください。

  ・SYSIN 指定の ACCEPT 文

  ・SYSOUT／SYSPUNCH 指定の DISPLAY 文
- 環境変数によって指定されるファイル名が有効となるかどうかは，ファイルシステムに依存します。
- ファイル名が絶対パス名でない場合は，OS の環境設定に従います。
- ファイルを標準入力（stdin）から読み込んだり，標準出力（stdout）または標準エラー出力（stderr）へ書き出したりしたい場合は，環境変数に stdin，stdout，または stderr を指定します。ただし，索引編成ファイル，相対編成ファイル，順編成の行制御のない可変長ファイルには，stdin，stdout，および stderr を指定できません。また，順編成の可変長ファイルには，stdin を指定できません。指定した場合，実行時にエラーとなります。
- 標準入出力ファイルを扱う場合，COBOL はすでに開かれているものとして処理します。
- 環境変数 CBL_外部装置名は，OPEN 文を実行するごとに環境変数の値が参照されます。
- 標準入力（stdin），標準出力（stdout），および標準エラー出力（stderr）を指定する場合は，英小文字で指定してください。「STDIN」のように英大文字で指定した場合，物理ファイル名として扱われます。
- 環境変数に物理ファイル名を割り当てていない場合は，作成するファイルの名称が不明のためエラーとなります。

入出力状態の値とファイル自動生成規則（環境変数指定の場合）

環境変数指定でファイルを割り当てた場合，ファイル入出力時の入出力状態の値，および物理ファイルが自動作成されるかどうかは，次の規則に従います。

OPEN モード	OPTIONAL 指定の有無	環境変数指定あり		環境変数指定なし
		物理ファイルの状態		
		すでに存在する	存在しない	
INPUT	なし	FS=00	FS=35	FS=90
	あり	FS=00	FS=05 自動作成されない	FS=05 自動作成されない
I-O	なし	FS=00	FS=35	FS=90

OPEN モード	OPTIONAL 指定の有無	環境変数指定あり		環境変数指定なし
		物理ファイルの状態		
		すでに存在する	存在しない	
	あり	FS=00	FS=05 自動作成される※1	FS=90
OUTPUT	なし	FS=00 再作成される※2	FS=00 自動作成される※1	FS=90
EXTEND	なし	FS=00	FS=35	FS=90
	あり	FS=00	FS=05 自動作成される※1	FS=90

（凡例）

FS=nn：FILE STATUS 句を指定したときに，入出力状態に nn が設定されることを示す

注

環境変数の値にファイル名を指定していない場合は，環境変数名なしとみなされます。

注※1

自動作成される物理ファイル名は，環境変数で指定した名称となります。

注※2

再作成とは，ファイル中にデータレコードがない状態にすることです。

COBOL プログラムの記述例

プログラム内で使用するファイル名"FILE-1"を外部装置名 SYS-01 に割り当てる例を，次に示します。

```
       SELECT FILE-1 ASSIGN TO SYS-01
```

外部装置名 SYS-01 に対応する環境変数「CBL_SYS_01」に，物理ファイル名"/DIR/FILE1.FIL"を指定する例を，次に示します。sh（B シェル）を使用する場合の指定例です。

```
CBL_SYS_01=/DIR/FILE1.FIL
export CBL_SYS_01
```

6.2.3 データ名指定

データ名指定は，SELECT 句で指定したファイル名に ASSIGN 句でデータ名を割り当て，このデータ名に物理ファイル名を転記して，物理ファイルを割り当てる方法です。

データ名指定で物理ファイルを割り当てる場合の COBOL プログラムの記述形式を次に示します。

形式（COBOL プログラム）

```
SELECT 〔OPTIONAL〕 ファイル名 ASSIGN TO データ名
```

構文規則

- データ名に指定する物理ファイル名は，ルートからの絶対パス名を指定します。このファイル名は，ファイルシステムの規則に従って指定する必要があります。
- ディレクトリ名，ファイル名に NULL 文字（X'00'）および半角空白文字（X'20'）を含んではなりません。NULL 文字または半角空白文字を含んだディレクトリやファイル名を指定した場合，NULL 文字または半角空白文字の直前までの文字列が有効となり，以降の文字列は無視されます。

一般規則

- 次の場合，データ名で指定した物理ファイルがなければ，指定した名称の物理ファイルが作成されます。

  （物理ファイルが作成される場合）

  1. ファイルを OUTPUT モードで開いたとき
  2. SELECT 句の OPTIONAL 指定のあるファイルを I-O または EXTEND モードで開いたとき

     このとき，SELECT 句に OPTIONAL 指定がある場合は，入出力状態に 05 が設定されます。
- ファイルが作成される場所は，データ名に指定した物理ファイル名の絶対パスに従います。ただし，指定した物理ファイル名中のディレクトリ名に相当するディレクトリがないと，物理ファイルは作成されません。
- データ名の値によって指定されるファイル名が有効となるかどうかは，ファイルシステムに依存します。
- ファイル名が絶対パス名でない場合は，OS の環境設定に従います。
- 物理ファイル名は，ファイルを開く前に設定する必要があります。また，ファイルを閉じる前に物理ファイル名を変更した場合，動作は保証しません。
- ファイルを標準入力（stdin）から読み込んだり，標準出力（stdout）または標準エラー出力（stderr）へ書き出したりしたい場合は，データ名に stdin，stdout，または stderr を指定します。ただし，索引編成ファイル，相対編成ファイル，順編成の行制御のない可変長ファイルには，stdin，stdout，および stderr を指定できません。また，順編成の可変長ファイルには，stdin を指定できません。指定した場合，実行時にエラーとなります。
- 標準入出力ファイルを扱う場合，COBOL はすでに開かれているものとして処理します。
- 標準入力（stdin），標準出力（stdout），および標準エラー出力（stderr）を指定する場合は，英小文字で指定してください。「STDIN」のように英大文字で指定した場合，物理ファイル名として扱われます。

入出力状態の値とファイル自動生成規則

定数指定の場合と同じです。詳細は，「6.2.1　定数指定」の「入出力状態の値とファイル自動生成規則（定数指定の場合）」を参照してください。

COBOL プログラムの記述例

プログラム内で使用するファイル名"FILE-1"を，"/dir"ディレクトリに格納されている"file.dat"ファイルに，データ名指定で割り当てる例を，次に示します。

```
        :
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT FILE-1 ASSIGN TO FILE-NAME.
        :
DATA DIVISION.
WORKING-STORAGE SECTION.
01 FILE-NAME PIC X(60).
        :
PROCEDURE DIVISION.
    MOVE '/dir/file.dat' TO FILE-NAME.
        :
```

6.3　入出力エラー処理

COBOL2002 では，ファイルの入出力処理中にエラーが発生した場合，プログラムを終了させる方法とプログラムを続行してエラーの回復を処理する方法があります。

一般には，エラーメッセージを参照してエラーの原因を追及し，プログラムを修正して再実行します。これ以外の方法として，FILE STATUS 句と USE 手続きを使用し，エラーの回復処理をプログラム中に設定しておくという方法があります。

FILE STATUS 句については，マニュアル「COBOL2002　言語　標準仕様編」「8.3.4(7)　FILE STATUS 句」を，USE 手続きについては，マニュアル「COBOL2002　言語　標準仕様編」「10.8.53 USE 文」を，それぞれ参照してください。

ここでは，FILE STATUS 句と USE 手続きを使用した入出力エラー処理について説明します。

なお，COBOL2002 では，共通例外処理を使って入出力エラー処理もできます。詳細は，「21.　共通例外処理」を参照してください。

6.3.1　入出力エラー処理の概要

入出力エラー発生時，USE 手続きと入出力手続き文の無条件文との関係を，次に示します。

表 6-2　入出力エラー発生時の制御の流れ

条件		AT END 指定※1		INVALID KEY 指定※1		入出力エラー	その他のエラー
		あり	なし	あり	なし		
USE 手続き	あり	AT END 指定の手続き	USE 処理※2	INVALID KEY 指定の手続き	USE 処理※2	USE 処理※2	USE 処理※2
	なし	AT END 指定の手続き	※3	INVALID KEY 指定の手続き	※3	※3	※3

注※1

NOT AT END，NOT INVALID KEY 指定の場合は，入出力文が正常終了したときだけ，NOT AT END，NOT INVALID KEY で指定した手続きに制御が渡ります。

注※2

USE 手続き処理後，エラーの発生した入出力文の次の文に制御が渡ります。FILE STATUS 句があれば，入出力状態の値が設定されます。

注※3

FILE STATUS 句があれば次の文に制御が渡ります。

FILE STATUS 句がなければ，次のように動作します。

- 共通例外処理の致命的例外の場合

プログラムの実行が，中止されます。

- 共通例外処理の非致命的例外の場合

  プログラムの実行が，継続されます。

詳細は，「21.9.2　従来形式の例外処理と共通例外処理の関係」の「(3)　FILE STATUS 句の指定と共通例外処理での異常終了」を参照してください。

共通例外処理の致命的例外，非致命的例外については，マニュアル「COBOL2002　言語　標準仕様編」「10.5.11　条件操作」を参照してください。

6.3.2　入出力状態の値

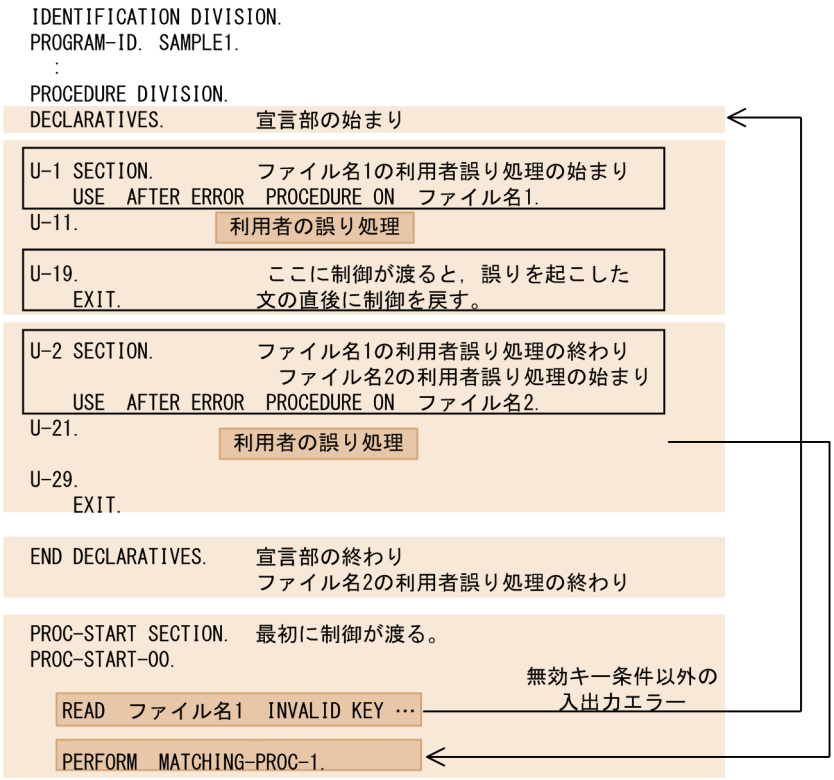
FILE STATUS 句を指定しておくと，入出力文でエラーが発生した場合，FILE STATUS 句で指定したデータ項目に入出力状態の値が設定されます。入出力状態の値と意味については，「付録 G　入出力状態の値」およびマニュアル「COBOL2002　言語　標準仕様編」「5.1.12　入出力状態」を参照してください。

6.3.3　USE 手続き

(1)　USE ERROR 手続きでの処理

COBOL プログラムで USE ERROR の手続き処理を使用したコーディング例を，次に示します。

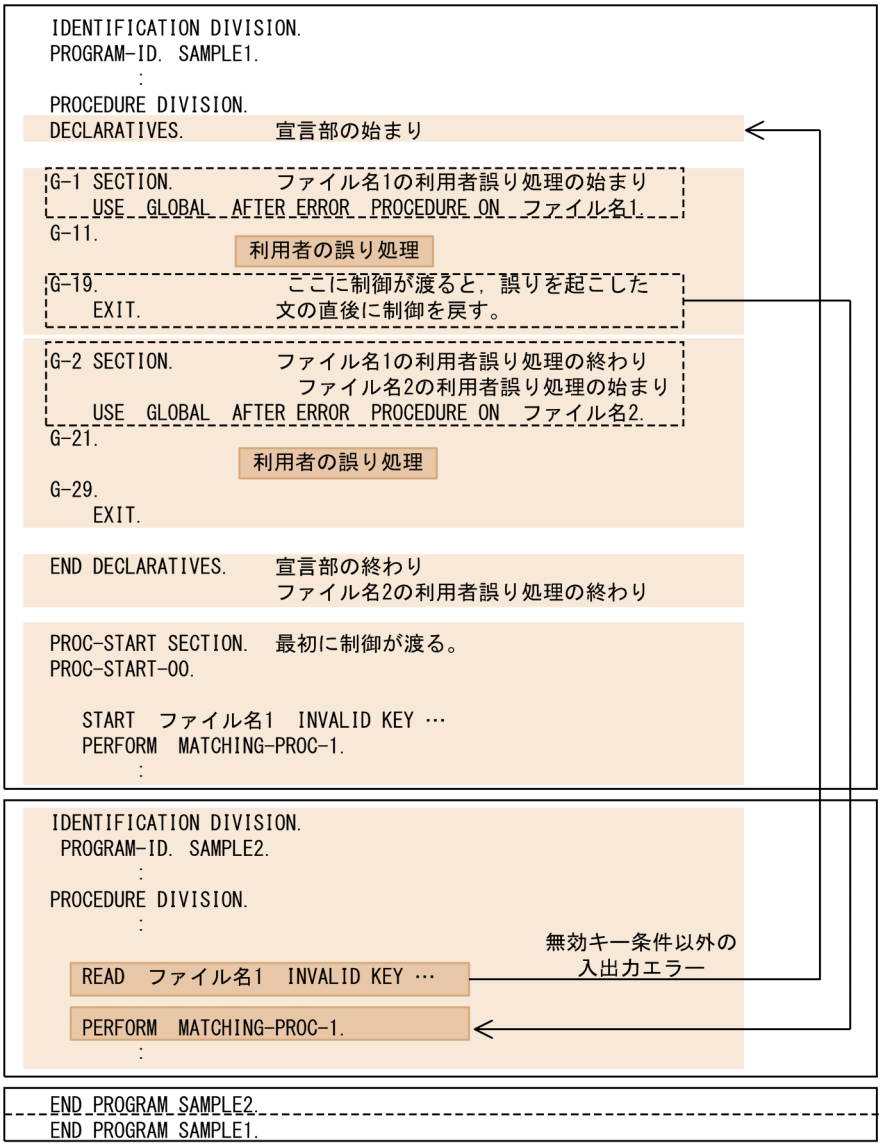
図 6-1　USE ERROR 手続き処理のコーディング例



(2)　USE GLOBAL ERROR 手続きでの処理

COBOL プログラムで USE GLOBAL ERROR の手続き処理を使用したコーディング例を，次に示します。

図 6-2　USE GLOBAL ERROR 手続き処理のコーディング例



6.3.4　ファイル入出力文でのエラー情報出力機能

FILE STATUS 句を指定した場合に，入出力文でエラーが発生したとき，FILE STATUS 句で指定したデータ項目に入出力状態の値が設定され，実行時メッセージは出力されません。通常は入出力状態の値と実行中プログラムの処理内容によってエラー要因を特定できますが，入出力状態値 90 はエラー要因の数が多いため，その特定が困難です。この場合，エラー要因特定のため，FILE STATUS 句の指定を取り消してプログラムを再実行し，実行時メッセージを出力する必要があります。

ファイル入出力文でエラー情報を出力する設定をしておくと，入出力文で入出力状態値 90 のエラーが発生した場合に，FILE STATUS 句の指定があっても実行時メッセージを出力できます。これによって，実行時メッセージでエラー要因を特定できます。

ファイル入出力文でエラー情報を出力する場合は，実行時環境変数 CBLIOMESSAGE に STATUS90 を指定します。

形式

```
CBLIOMESSAGE=STATUS90
```

規則

- この環境変数に STATUS90 を指定した場合に，FILE STATUS 句を指定したファイルの入出力文で入出力状態の値が 90 となるエラーが発生したとき，入出力状態の値を設定したあと，実行時メッセージを出力して処理を続行します。
- 出力する実行時メッセージは，FILE STATUS 句を指定しなかった場合に出力される実行時メッセージと同じメッセージです。ただし，メッセージレベルは I（お知らせ）となります※。
- この環境変数の指定がない，または STATUS90 以外の値を指定した場合は，この機能は無効になります。
- この環境変数の指定によって実行時メッセージの出力以外，COBOL プログラムの動作が変わることはありません。
- この環境変数の指定は，FILE STATUS 句を指定したすべてのファイルの入出力文に有効となります。ただし，例外として，テストデバッガ実行時にファイルシミュレーション機能の対象となるファイルに対しては，この機能は無効になります。
- FILE STATUS 句の指定がないファイルの動作は，この環境変数を指定しても変わりません。

注※

プログラム実行後に実行時メッセージが出力されているときは，メッセージレベルが I であれば処理が続行されていると判定できます。

注意事項

COBOL 入出力サービスルーチンおよびバイトストリーム入出力サービスルーチンは，この機能の対象となりません。

6.3.5 ファイル属性の整合性チェック

既存の物理ファイルを開く際に，ファイル属性情報とプログラムの指定に矛盾※があるかをチェックします。矛盾がある場合は物理ファイルを開けないため，ファイル属性情報とプログラムの指定を一致させておく必要があります。

注※

ファイル編成によっては，物理ファイルを作成したプログラムに記述されたファイル管理記述項とファイル記述項の内容の一部の情報が，ファイル属性情報として物理ファイル内に格納されています。このファイル属性情報と，物理ファイルを開くプログラムに記述されたファイル管理記述項およびファイル記述項の指定の矛盾をチェックします。

(1) 整合性チェックの内容

- 順編成ファイル（可変長レコード形式）および相対編成ファイルの場合は，ファイル属性情報のファイル形式とレコード長をチェックします。
- ISAM による索引編成ファイルの場合は，ファイル属性情報のファイル形式，最大レコード長，最小レコード長，キー個数，主レコードキー，副レコードキーをチェックします。

6.4　順編成ファイル

6.4.1　ファイルの作成と割り当て方法

順編成ファイルの作成方法と割り当て方法について説明します。

順編成ファイルについては，マニュアル「COBOL2002　言語　標準仕様編」「5.1.7(1)　順編成」を参照してください。

(1)　固定長ファイルの作成方法

次の方法で作成できます。

- COBOL2002 の入出力機能
- などのエディタ
- C プログラムなど，他言語プログラムでのファイル作成

(2)　可変長ファイルの作成方法

COBOL2002 の入出力機能によって作成できます。

他言語のプログラムで可変長の順編成ファイルを作成したい場合は，COBOL 入出力サービスルーチンを使用します。COBOL 入出力サービスルーチンの詳細は，「13.　COBOL 入出力サービスルーチン」を参照してください。

(3)　ファイルの割り当て方法

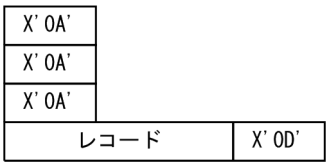
- 「6.2　ファイル割り当ての共通規則」に従って，物理ファイル名を割り当ててください。
  物理ファイル名には，OS のファイルシステム上で有効となるファイル名を指定してください。
- 順編成ファイルを使用する場合は，ORGANIZATION 句に SEQUENTIAL を指定します。

6.4.2　順編成ファイルの行制御

順編成ファイルでは，ADVANCING 指定の WRITE 文を実行した場合，指定に従って行制御をします。LINAGE 句の指定があるとき，PAGE 指定は論理ページの最初の行に位置づくまで行送りします。なお，行制御して出力された順編成ファイルは，順編成ファイルとしては読み込むことができません。

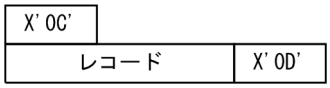
出力例を次に示します。出力例の X'0A'は改行，X'0D'は復帰，X'0C'は改ページを示します。

WRITE REC AFTER ADVANCING 3 を実行した場合

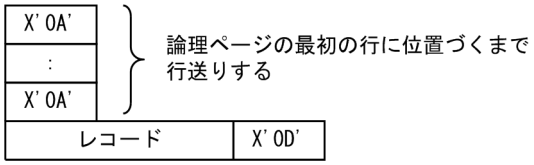


WRITE REC AFTER ADVANCING PAGE を実行した場合

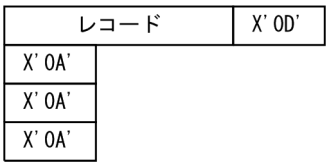
- LINAGE 句の指定がない場合



- LINAGE 句の指定がある場合



WRITE REC BEFORE ADVANCING 3 を実行した場合



6.5　相対編成ファイル

6.5.1　ファイルの作成と割り当て方法

相対編成ファイルの作成方法と割り当て方法について説明します。

相対編成ファイルについては，マニュアル「COBOL2002　言語　標準仕様編」「5.1.7(2)　相対編成」を参照してください。

(1)　ファイルの作成方法

固定長ファイル／可変長ファイル共に，COBOL2002 の入出力機能によって作成できます。

他言語のプログラムで相対編成ファイルを作成したい場合は，COBOL 入出力サービスルーチンを使用します。COBOL 入出力サービスルーチンの詳細は，「13.　COBOL 入出力サービスルーチン」を参照してください。

(2)　ファイルの割り当て方法

- 「6.2　ファイル割り当ての共通規則」に従って，物理ファイル名を割り当ててください。
  物理ファイル名には，OS のファイルシステム上で有効となるファイル名を指定してください。
- 相対編成ファイルを使用する場合は，ORGANIZATION 句に RELATIVE を指定します。

6.6　ISAM による索引編成ファイル

ISAM による索引編成ファイルの作成方法と割り当て方法について説明します。

索引編成ファイルについては，マニュアル「COBOL2002　言語　標準仕様編」「5.1.7(3)　索引編成」，およびマニュアル「索引順編成ファイル管理 ISAM」を参照してください。

なお，このシステムでは，索引編成ファイルの操作や保守をするために，ISAM ユティリティを使用できます。ISAM ユティリティの詳細については，マニュアル「索引順編成ファイル管理 ISAM」を参照してください。

6.6.1　ファイルの作成と割り当て方法

索引編成ファイルの作成方法と割り当て方法について説明します。

(1)　固定長ファイルの作成方法

COBOL2002 の入出力機能，または ISAM ユティリティによって作成できます。

(2)　可変長ファイルの作成方法

COBOL2002 の入出力機能，または ISAM ユティリティによって作成できます。

(3)　ファイルの割り当て方法

「6.2　ファイル割り当ての共通規則」に従って，物理ファイル名を割り当ててください。

ただし，次の点に注意してください。

- ISAM による索引編成ファイルでは，拡張子の異なる複数の物理ファイルが一つの索引編成ファイルを構成しています。そのため，物理ファイル名は，拡張子を付けないで指定してください。物理ファイル名に拡張子を付けて指定した場合，動作は保証しません。
- 索引編成ファイルを使用する場合は，ORGANIZATION 句に INDEXED を指定します。

(4)　生成される物理ファイル

索引編成ファイルによって生成される物理ファイルを，次に示します。

ISAM のファイル種別		ファイル拡張子	説明
キー定義ファイル		.DEF	データファイルとキーファイルとの対応を表すデータを格納しています。
キーファイル	主キーファイル	.K01	主レコードキーでレコードを検索するために必要な情報を保持しています。主レコードキー以外の

ISAM のファイル種別		ファイル拡張子	説明
			キーでレコードを検索するには，副キーファイルを使用します。
	副キーファイル	.K02～.K99	主レコードキーとは別のキーでレコードを検索するために必要な情報を保持しています。副レコードキーは，1 個の索引編成ファイルに対して複数指定できます。
データファイル		.DAT	実際にレコードを格納しているファイルです。格納するレコード形式は，固定長レコードでも可変長レコードでもかまいません。

(5) 索引編成ファイルに対する OPEN モード

索引編成ファイルに対して OUTPUT モードの OPEN 文を実行する場合，環境変数 CBLISAMDL，および CBLD_ファイル名の指定によって，動作が異なります。環境変数の指定による OPEN 文の動作の違いを，次に示します。

CBLISAMDL	CBLD_ファイル名		
	ISAMDL	NOISAMDL	指定なし
YES	作成	追加書き	作成
NO または指定なし	作成	追加書き	追加書き

（凡例）

作成：一度ファイルを削除した後，新規作成して出力する

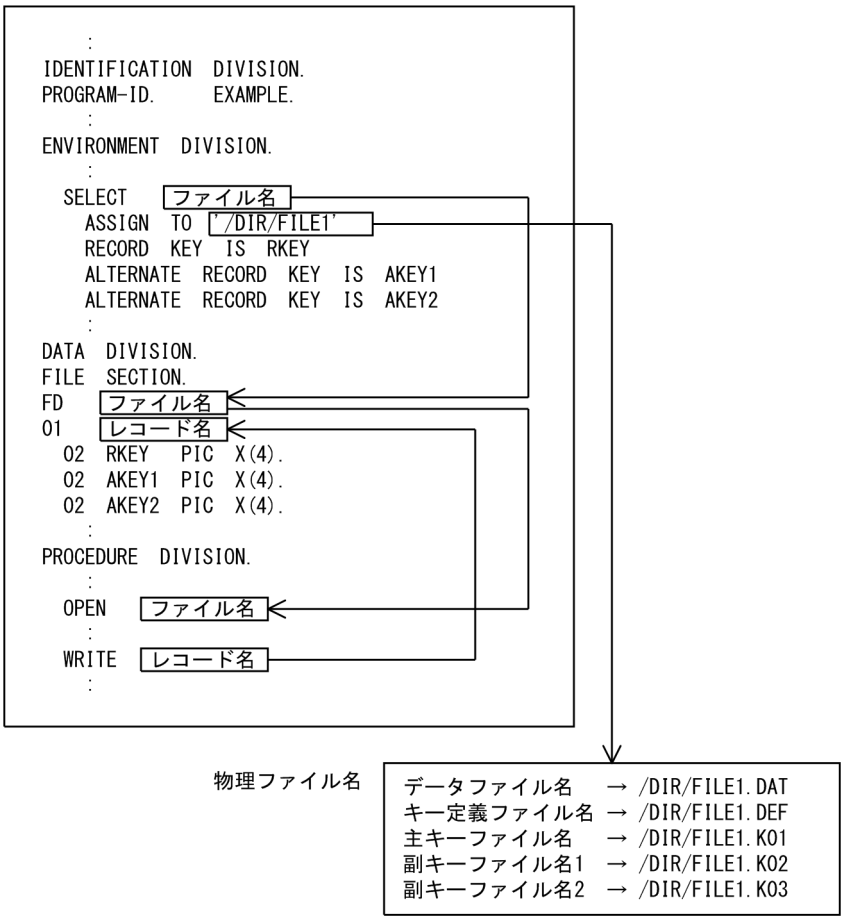
追加書き：存在するファイルに対して，追加で出力する

(6) プログラムとファイル割り当ての関係

定数指定，環境変数指定，データ名指定のそれぞれについて，プログラムと物理ファイルとの関係を索引編成ファイルの例で示します。

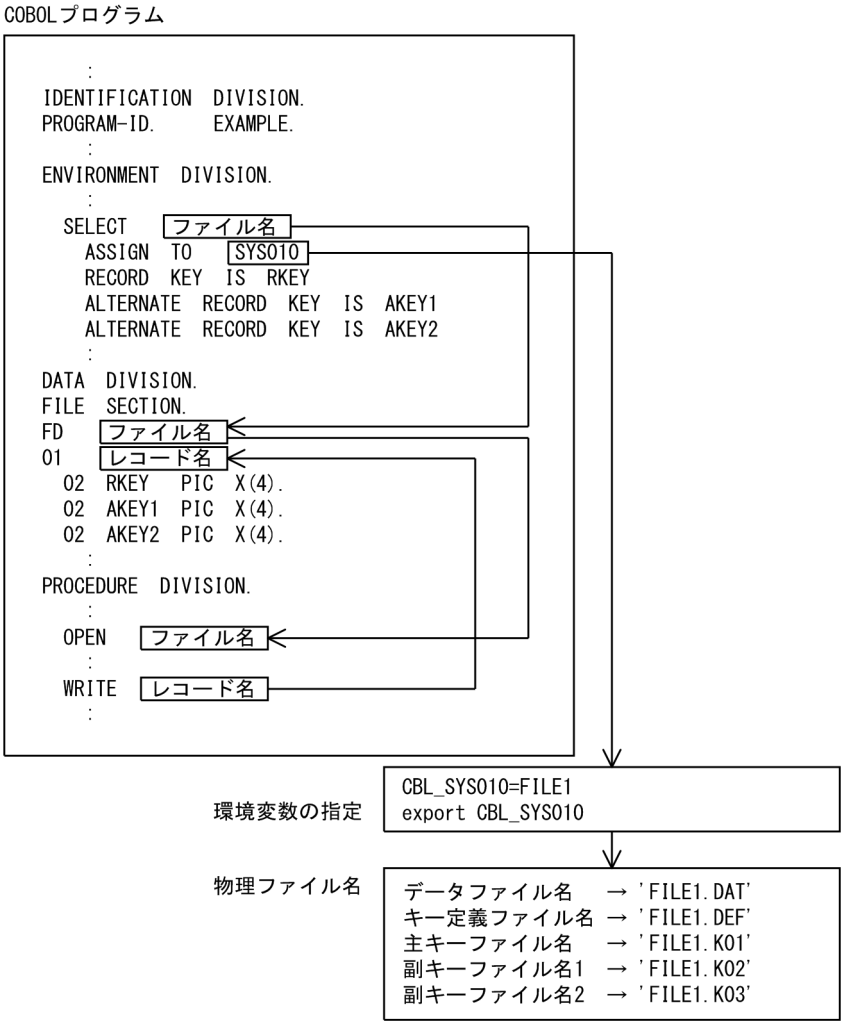
(a)　定数指定の ASSIGN 句の場合

COBOLプログラム

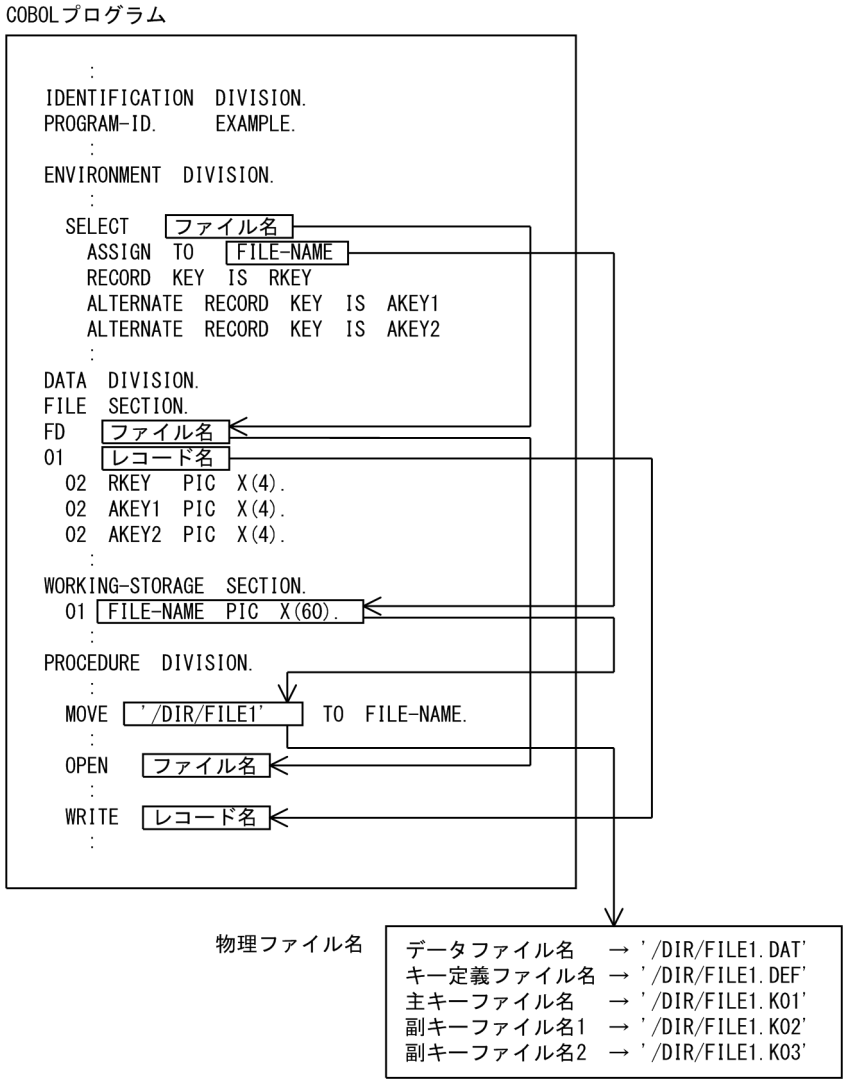


物理ファイル名

(b)　環境変数指定の ASSIGN 句の場合



(c)　データ名指定の ASSIGN 句の場合



6.6.2　ファイル編成とレコード形式

ISAM による索引編成ファイルでは，固定長と可変長のレコード形式が使用できます。

6.6.3　コンパイル，リンクの指定

索引編成ファイル入出力機能を使用したプログラムのコンパイル，およびリンクは，「34.1.3　ccbl2002 コマンドの-l オプション」を参照してください。

6.7　テキスト編成ファイル

ここでは，テキスト編成ファイルについて説明します。

テキスト編成ファイルについては，マニュアル「COBOL2002　言語　拡張仕様編」「2.1　テキスト編成」を参照してください。

6.7.1　ファイルの作成と割り当て方法

テキスト編成ファイルの作成方法と割り当て方法について説明します。

(1)　ファイルの作成方法

テキスト編成ファイルは，次の方法で作成できます。

- COBOL の入出力機能
- vi などのエディタ
- C プログラムなど，他言語プログラムでのファイル作成

(2)　ファイルの割り当て方法

- 「6.2　ファイル割り当ての共通規則」に従って，物理ファイル名を割り当ててください。
  物理ファイル名には，OS のファイルシステム上で有効となるファイル名を指定してください。
- テキスト編成ファイルを使用する場合は，ORGANIZATION 句に LINE SEQUENTIAL を指定します。

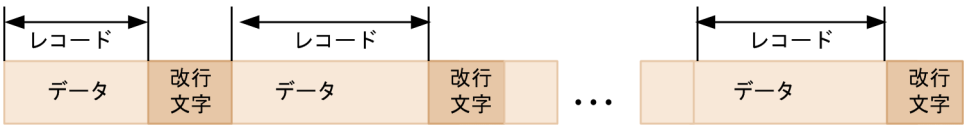
6.7.2　テキスト編成ファイルのファイル編成とレコード形式

テキスト編成ファイルは，固定長と可変長のレコード形式が使用できます。ただし，実際にアクセスする物理ファイル形式は，固定長／可変長による違いはありません。そのため，テキスト編成ファイルであれば，固定長／可変長のどちらでもアクセスできます。

(1)　ファイル形式

テキスト編成ファイルは，テキストファイルの各行（表示できる文字で構成されたテキスト行）を 1 レコードとして，改行文字をレコードの区切り文字とした可変長形式のファイルです。

テキスト編成ファイルの形式を次に示します。



(2) レコード形式

テキスト編成ファイルの各レコードは，レコード本体に改行文字が付けられた形式で構成されています。

テキスト編成ファイルのレコード形式を次に示します。

改行文字は，一般的にエディタなどで入力するとき［Enter］キーを押して入力する文字です。

次に Windows と UNIX の改行文字の相違を示します。Windows と UNIX の間でテキスト編成ファイルを移行するときは注意してください。

Windows の改行文字

Windows での改行文字は，復帰文字（X'0D'）+改行文字（X'0A'）の 2 バイトで構成されています。COBOL2002 のテキスト編成ファイルでは，この 2 バイトで構成された文字を改行文字と認識します。

ただし，テキスト編成ファイルの READ 文については，X'0A'だけでも改行文字と認識します。

UNIX の改行文字

UNIX での改行文字は，改行文字（X'0A'）だけの 1 バイトで構成されています。COBOL2002 のテキスト編成ファイルでは，この 1 バイトで構成された文字を改行文字と認識します。

6.7.3 入出力手続き文と動作

テキスト編成ファイルに対する入出力手続き文について説明します。なお，次に説明する規則以外については，順編成ファイルの場合と同じです。

(1) READ 文

(a) READ 文でのテキスト編成ファイル固有の規則

- 物理ファイル上にあるレコードの区切り文字（改行文字）は，COBOL プログラムのレコード領域に格納されません。
- テキスト行がレコード長より短いときは，改行文字までの文字列が入力され，残りの部分には空白（X'20'）が埋められます。改行文字はすべて切り捨てられます。
- テキスト行がレコード長より長いときは，ファイルの開くモードによって次のように動作が異なります。

（I-O モード以外で開いている場合）

レコード長で区切られた複数レコードとして入力されます。このとき，FILE STATUS 句を指定していると，入出力状態には 04 が返されます。

（I-O モードで開いている場合）

READ 文はエラーとなります。このとき，FILE STATUS 句を指定していると，入出力状態には 30 が返されます。

- 復帰文字（X'0 D'）は 1 バイトのデータとして扱われます。
- タブ文字（X'09'）はそのままデータとして入力され，空白には置き換えられません。
- ファイルの終わりが改行文字でないときは，ファイルの終わりに改行文字があるとみなされます。
- レコード中に NULL（X'00'）が含まれる場合，NULL（X'00'）以降はデータとして入力されません。
- ファイル定義が可変長の場合，レコード記述項の最大長分の固定長として読み込まれます。このとき，レコード長には，ファイルから実際に読み込んだ長さが設定されます。

  また，I-O モード以外でファイルを開き，かつレコード記述項の長さが最大レコード長より短い場合，最大レコード長で確保されたレコード領域に最大レコード長で読み込まれます。このとき，DEPENDING ON 指定のデータ名には，レコード記述項で定義したレコード長より大きい値が設定される場合があります。

  このため，入出力状態が 00 の場合は改行文字の直前までの長さが，入出力状態が 04 の場合は最大レコード長が，それぞれ DEPENDING ON 指定のデータ名に設定されます。どちらの場合でも，COBOL プログラムから参照できるのは，レコード記述項で定義した長さ分の領域だけです。

(2) WRITE 文

(a) WRITE 文でのテキスト編成ファイル固有の規則

- レコード領域の最後が半角空白文字（X'20'）以外のときは，改行文字を付けて出力されます。
- レコード領域の最後が半角空白文字（X'20'）のときは，末尾の半角空白文字（終端から半角空白文字以外の文字が出現するまでの部分）が切り捨てられ，改行文字を付けて出力されます。出力レコードの末尾の空白文字列を出力したい場合は，「6.7.5 レコード末尾の空白文字を出力する機能」を参照してください。

(3) REWRITE 文

(a) REWRITE 文でのテキスト編成ファイル固有の規則

- REWRITE 文を実行する直前の入出力文は READ 文で，この READ 文が成功していなければなりません。
- REWRITE 文は，読み込んだテキスト行の長さと更新用レコードの長さが等しい場合だけ，テキスト行が書き換えられます。それ以外の場合，REWRITE 文はエラーとなり，FILE STATUS 句を指定していると，入出力状態に 44 が返されます。なお，更新用レコードの長さとは，後続の半角空白文字を削除した長さを指します。更新するレコードの長さにレコード末尾の半角空白文字も含めたい場合は，「6.7.5 レコード末尾の空白文字を出力する機能」を参照してください。
- REWRITE 文では，テキスト行のうち改行文字，またはファイルの終わり（EOF）の直前までを書き換えます。

6.7.4　規則

- テキスト編成ファイルは，印字できる文字や制御コード（改行文字，復帰文字，タブ文字など）で構成されている必要があります。
- 出力先の物理ファイルに標準出力（stdout）または標準エラー出力（stderr）を指定し，そのファイルをリダイレクションした場合，動作は保証しません。
- WRITE 文を実行して文字を書き出す場合，X'20'未満の制御コードについてもそのまま書き出されます。
- 可変長ファイルに対して REWRITE 文を実行する場合，次のどの方法で FD 項を定義しても結果は同じになります。
  - 01 レベルを複数回記述する場合

```
FD T-FILE.
01 T-REC1 PIC X(10).
01 T-REC2 PIC X(30).
```

  - VARYING 句で可変長を定義する場合

```
FD T-FILE RECORD VARYING IN SIZE FROM 1 TO 30
                 DEPENDING ON L-REC.
01 T-REC PIC X(30).
```

  - OCCURS 句で可変長を定義する場合

```
FD T-FILE.
01 T-REC.
  02 T-RECD PIC X OCCURS 30 TIMES DEPENDING ON L-REC.
```

- Unicode 機能を使用している場合，テキスト編成ファイルは UTF-8 で記述されているものとして扱ってください。Unicode 機能については，「27.　Unicode 機能」を参照してください。

6.7.5　レコード末尾の空白文字を出力する機能

レコード末尾の空白文字を出力する設定をしておくと，テキスト編成ファイルに対する WRITE 文，および REWRITE 文でレコード末尾に半角空白文字（X'20'）があった場合に，半角空白文字を削除しないで，そのままファイルに書き出せます。

レコード末尾の空白文字を出力する機能を使用するには，実行時環境変数 CBLD_ファイル名に TEXTWRITESPACE を指定するか，実行時環境変数 CBLTEXTWRITESPACE に YES を指定してください。

(1)　ファイル単位に指定する方法

形式

```
CBLD_ファイル名={ TEXTWRITESPACE | NOTEXTWRITESPACE }
```

機能

- 環境変数CBLD_ファイル名に TEXTWRITESPACE を指定すると，実行単位中の任意のテキスト編成ファイルに対して，WRITE 文および REWRITE 文でレコード末尾の連続する半角空白文字をファイルに書き出します。
- 環境変数CBLD_ファイル名に NOTEXTWRITESPACE を指定すると，この機能は無効になります。

(2) 実行単位中のすべてのファイルに指定する方法

形式

```
CBLTEXTWRITESPACE=YES
```

機能

- 環境変数CBLTEXTWRITESPACE に YES を指定した場合は，実行単位中のすべてのテキスト編成ファイルに対する WRITE 文および REWRITE 文でレコード末尾からの連続する半角空白文字をファイルに書き出します。
- 環境変数CBLTEXTWRITESPACE に YES の指定がないか，または YES 以外を指定した場合は，この機能は無効になります。

(3) 規則

- この機能を使用した場合，WRITE 文や REWRITE 文で書き出すレコードの長さは次のようになります。
  固定長形式の場合：ファイル節に定義したレコード長
  可変長形式の場合：WRITE 文や REWRITE 文実行時に指定されたレコード長
- ファイル単位に指定する方法と実行単位中のすべてのファイルに指定する方法を併用した場合，ファイル単位に指定する方法での指定が優先されます。次に指定の組み合わせによる機能の有効・無効状態を示します。

表 6-3 環境変数 CBLTEXTWRITESPACE と環境変数 CBLD_ファイル名の指定の組み合わせ

CBLTEXTWRITESPACE	CBLD_ファイル名		
	TEXTWRITESPACE	NOTEXTWRITESPACE	指定なし
YES	有効	無効	有効
環境変数指定なし または YES 以外	有効	無効	無効

(4) 注意事項

- WRITE 文の ADVANCING 指定，または POSITIONING 指定で書き出された改行文字に対しては，この機能を使用しても改行文字の前に半角空白文字は出力されません。半角空白文字を含んで改行文字

を書き出したいときは，ADVANCING 指定や POSITIONING 指定がない WRITE 文で半角空白文字だけのレコードを書き出してください。

- この機能が有効な場合，READ 文で読み込んだ固定長形式のレコードを，REWRITE 文を実行してレコード領域長より短いレコードに更新しようとしたときは，読み込んだレコード長と更新するレコード長が不一致となり，REWRITE 文が失敗します。レコード領域長より短いレコードに対して REWRITE 文を実行するときは，可変長形式のファイルで定義して，読み込んだレコード長と変わらないようにしてください。

（例）

入力レコード

```
'AAAAA'
```

固定長形式のファイル／レコード定義

```
FD FILE-1.
01 FILE-1-REC   PIC X(10).
```

可変長形式のファイル／レコード定義

```
FD FILE-2 RECORD IS VARYING IN SIZE FROM 1 TO 10 CHARACTERS
                    DEPENDING ON REC-LEN.
01 FILE-2-REC   PIC X(10).
```

表 6-4　レコード定義より短いレコードに対する REWRITE 文の動作

内容と結果	機能の有効・無効状態			
	有効		無効	
	固定長形式	可変長形式	固定長形式	可変長形式
READ 文実行後のレコード領域の内容	'AAAAA△△△△△'	'AAAAA'※	'AAAAA△△△△△'	'AAAAA'※
REWRITE 文でレコードを'aaaaa'に更新するときの結果	半角空白を含む 10 バイトを更新しようとして KCCC3521R-S エラーとなる。	'aaaaa'に更新される。	レコード末尾の半角空白は削除され，'aaaaa'で更新される（入力レコード長と等しくなる）。	

（凡例）

△：半角空白

注※

REC-LEN 項目には入力レコード長が格納される。

(5)　使用例

環境変数 CBLTEXTWRITESPACE と環境変数 CBLD_ファイル名の指定を組み合わせて使用する場合の例を次に示します。

環境変数の指定

```
CBLTEXTWRITESPACE=YES
CBLD_FILE_2=NOTEXTWRITESPACE
```

プログラム例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FILE-1 ASSIGN TO SYS100
                         ORGANIZATION IS LINE SEQUENTIAL.
           SELECT FILE-2 ASSIGN TO SYS200
                         ORGANIZATION IS LINE SEQUENTIAL.
       DATA DIVISION.
       FILE SECTION.
       FD FILE-1.
       01 FILE-1-REC      PIC X(10).
       FD FILE-2.
       01 FILE-2-REC      PIC X(10).
       PROCEDURE DIVISION.
           OPEN OUTPUT FILE-1 FILE-2.
      * レコード領域の後ろ5バイトは半角空白となる
           MOVE '12345' TO FILE-1-REC FILE-2-REC.
      * FILE-1は，CBLTEXTWRITESPACE=YESによって機能が有効なため，レ
      * コード末尾の半角空白は削除されない。
           WRITE FILE-1-REC.
      * FILE-2は，CBLD_FILE_2=NOTEXTWRITESPACEによって機能が無効なた
      * め，レコード末尾の半角空白は削除される。
           WRITE FILE-2-REC.
            :
```

実行結果

FILE1 の内容：12345△△△△△改行

FILE2 の内容：12345 改行

（凡例）

△：半角空白文字

改行：改行文字

6.8　CSV 編成ファイル（表計算プログラムファイル）

CSV（Comma Separated Value）編成ファイルは，表計算プログラムファイルともいい，表のデータを 1 行ごとに，列をコンマで区切って出力したファイルです。ここでは，CSV 編成ファイルの入出力処理について説明します。

CSV 編成ファイルについては，マニュアル「COBOL2002　言語　拡張仕様編」「17.　CSV ファイル入出力機能」を参照してください。

6.8.1　ファイルの作成と割り当て方法

CSV 編成ファイルの作成方法と割り当て方法について説明します。

(1)　ファイルの作成方法

CSV 編成ファイルは，次の方法で作成できます。

- COBOL の入出力機能
- vi などのエディタ
- C プログラムなど，他言語プログラムでのファイル作成
- 表計算プログラムでのファイル作成

(2)　ファイルの割り当て方法

- 「6.2　ファイル割り当ての共通規則」に従って，物理ファイル名を割り当ててください。
  物理ファイル名には，OS のファイルシステム上で有効となるファイル名を指定してください。
- CSV 編成ファイルを使用する場合は，ORGANIZATION 句に CSV を指定します。

6.8.2　CSV 編成ファイルのファイル編成とレコード形式

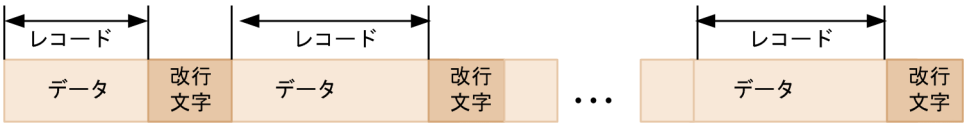
CSV 編成ファイルでは，テキスト編成ファイルと同じように，1 行（改行文字までの文字列）を 1 レコードとみなします。さらに，おのおののレコードは，コンマ（,）によって複数のデータ項目に区切られた形式になっています。コンマによって区切られたデータ項目をセルと呼びます。CSV 編成ファイルでは，このセル単位でデータを入出力できます。

なお，COBOL で CSV 編成ファイルを扱う場合は，次の形式に従った，固定長のレコード形式だけが使用できます。

(1)　ファイル形式

CSV 編成ファイルは，CSV ファイルの各行を 1 レコードとして，改行文字をレコードの区切り文字とした可変長形式のファイルです。

CSV 編成ファイルの形式を次に示します。



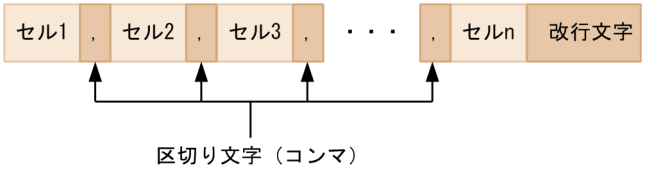
(2)　レコード形式

CSV 編成ファイルの各レコードは，レコード本体に改行文字が付けられた形式で構成されています。改行文字は，一般的にエディタなどで入力するとき［Enter］キーを押して入力する文字です。改行文字の詳細については，「6.7.2　テキスト編成ファイルのファイル編成とレコード形式」を参照してください。

さらに，おのおののレコードは，コンマを区切り文字とした「セル」と呼ばれるデータ項目単位に分割されています。

CSV 編成ファイルのデータ入出力は，レコード単位で実行しますが，レコード中の各セルの内容は，入力レコード領域に定義した基本項目ごとに処理できます。

CSV 編成ファイルのレコード形式を次に示します。



なお，コンマ（,）の替わりにタブ文字（X'09'）を区切り文字として使えます。詳細は，「6.8.9　セルデータをタブ文字区切りで入出力する機能」を参照してください。

6.8.3　入出力手続き文と動作

CSV 編成ファイルに対する入出力手続き文について説明します。なお，CSV 編成ファイルでは，OPEN 文，READ 文，WRITE 文，CLOSE 文だけが手続き文として使用できます。これ以外の手続き文は，使用できません。また，次に記載されている以外の規則については，順編成ファイルの場合と同じです。

(1)　OPEN 文

- CSV 編成ファイルは，INPUT，OUTPUT，EXTEND モードで開けます。

(2) READ 文

- CSV 編成ファイルから読み込まれた 1 行中に含まれる各セルの情報は，レコード定義の基本項目に 1 対 1 で対応して格納されます。このとき，レコード中に含まれるセルの個数がレコード定義の基本項目数より多い場合，対応する基本項目がないセルの情報は無視されます。逆に，レコード中に含まれるセルの個数がレコード定義の基本項目数より少ない場合，対応するセルがない基本項目の値は変更されません。
- セルの情報が基本項目に格納されるときは，英数字項目の転記の規則に従います。
- 入力したレコードにダブルコーテーション（"）が含まれない場合，コンマ（,）または改行文字を区切り文字として，データが入力されます。
- 入力したレコードにダブルコーテーション（"）が含まれる場合，次の規則に従ってデータが入力されます。

  （コンマの直後がダブルコーテーションの場合）

  次に現れるダブルコーテーションの直前までが，セルの情報として扱われます。ダブルコーテーションが出現する前にコンマがあっても，セルの区切り文字とはみなしません。

  この場合，セルの情報にダブルコーテーションが含まれる場合でも，区切り文字とみなされるので注意が必要です。

  （コンマの直後がダブルコーテーションでない場合）

  データ中に含まれるダブルコーテーションは，文字データとして扱われます。
- 入力したレコード中の，一つのセルの文字列長が入力領域より長い場合，入力領域の長さ分の文字列が読み込まれ，残りの部分は切り捨てられます。このとき，FILE STATUS 句を指定していると，入出力状態に 04 が返されます。
- INTO 指定がある場合は，INTO 指定のレコード構造に従ってデータが入力されます。INTO 指定がない場合は，ファイル記述項に定義したレコードの構造に従ってデータが入力されます
- READ 文の実行が失敗した場合，レコード領域の内容が不定となります。

(3) WRITE 文

- WRITE 文を実行すると，レコード領域中の基本項目単位をセルの情報として，おのおののセルがダブルコーテーション（"）で囲まれ，レコードの終端に改行文字が付いた形式で出力されます。各セルの情報をダブルコーテーションで囲まないで出力したい場合は，「6.8.6　セルデータをダブルコーテーションで囲まないで出力する機能」の説明を参照してください。
- 出力レコードの基本項目に含まれる右端の半角空白文字は出力されません。また，出力する基本項目のデータがすべて半角空白文字の場合，対応するセルの情報には，連続したダブルコーテーション（""）だけが出力されます。

  出力レコードの基本項目のデータの最後にある空白も出力したい場合は，「6.8.8　データの後の空白文字を出力する機能」を参照してください。
- FROM 指定がある場合は，FROM 指定のレコード構造に従ってデータが出力されます。FROM 指定がない場合は，ファイル記述項に定義されたレコードの構造に従ってデータが出力されます。

6.8.4　注意事項

- CSV 編成ファイルに対して，ファイル共用は使用できません。
- READ 文でセル情報を基本項目単位に格納する場合，レコード定義中に JUSTIFIED RIGHT 句を指定しても，無効となります。
- 一つの CSV 編成ファイルに対して，AFTER ADVANCING 指定した WRITE 文と BEFORE ADVANCING 指定した WRITE 文を同時に使用すると，レコードは正しく出力されません。
- Unicode 機能を使用している場合，CSV 編成ファイルは UTF-8 で記述されているものとして扱ってください。Unicode 機能については，「27.　Unicode 機能」を参照してください。

6.8.5　セルデータを数値として入出力する機能

(1)　セルデータを数値として入出力する機能の概要

-NumCsv オプションを指定すると，外部 10 進項目，外部浮動小数点数字項目で定義されているセルに対して，データを数値として入出力できます。

この機能は，表計算プログラムのような COBOL とは数値データの表現形式が異なるプログラムと，COBOL プログラムとの間で数値データをやり取りする場合に使用します。例えば，次のような場合は，-NumCsv オプションを指定することで，表計算プログラムと COBOL プログラムとで数値データをやり取りできます。

（例 1）

次のようなデータを，WRITE 文で CSV 編成ファイルに出力する場合

データ定義	データ項目に格納されている値	-NumCsv オプション	ファイルへの出力結果
99v99	1.23	なし	0123
		あり	01.23

-NumCsv オプションを指定すると小数点が出力されるため，表計算プログラムで正しい数値として入力できます。

（例 2）

次のようなデータを，READ 文で CSV 編成ファイルから入力する場合

データ定義	ファイルから入力した値	-NumCsv オプション	データ項目への格納結果
9(9)	123	なし	123△△△△△△
		あり	000000123

（凡例）

△：半角空白文字

-NumCsv オプションを指定すると数字項目に変換されるため，COBOL プログラムで数値として扱えます。

(2) データ変換の規則

-NumCsv オプションを指定した場合の，数字項目へのデータ変換規則を次に示します。

データ変換の一般規則

- 特殊名段落の DECIMAL-POINT IS COMMA 句の指定は有効です。ただし，小数点としてコンマ(,)を使用する場合は，その数値データをダブルコーテーション（"）で囲む必要があります。ダブルコーテーションで囲まれていないコンマは，すべてセルの区切り文字とみなされます。
- セルにデータを入出力するとき，次に示す文字が数値データとして有効です。

表 6-5　CSV 編成ファイルに数値データとして有効な文字の種類

セルの属性	入力時に有効な文字	出力時に有効な文字
外部 10 進項目	• 0～9，+，- • 小数点 • 右端，または左端にある空白文字（X'20'）	• 0～9，+，-
外部浮動小数点数字項目	• 0～9，+，-，E，e • 小数点 • 右端，または左端にある空白文字（X'20'） • 符号位置にある空白文字（X'20'）※	• 0～9，+，-，E，e • 小数点 • 空符号位置にある空白文字（X'20'）※

注※

外部浮動小数点数字項目の指数部，仮数部の符号をマイナス（-）で定義した場合，符号位置にある空白文字（X'20'）は，有効な文字となります。

- 外部 10 進項目，外部浮動小数点数字項目で定義したセルに，次のようなデータを入力した場合，実行時エラーになります。

  1.「表 6-5　CSV 編成ファイルに数値データとして有効な文字の種類」で示した「有効な文字」以外の文字が含まれるデータ

  2. 小数点が複数あるデータ

  3. 外部 10 進形式で，符号が複数ある，または符号の位置が不正なデータ

  4. 外部浮動小数点数字項目で，浮動小数点数字定数として正しくないデータ

  （ただし，指数部，仮数部の符号をマイナス（-）で定義した場合，データ中で符号を示す位置が空白文字（X'20'）であってもよい）

入力時のデータ変換規則

- 外部 10 進項目，外部浮動小数点数字項目で定義したセルでは，入力できるデータの長さは 512 バイトまでです。データの長さが 512 バイトを超える場合，それ以降のデータは切り捨てられます。切り捨てが起きた場合，FILE STATUS 句を指定していると，入出力状態に 04 が返されます。また，2 バイト文字の 1 バイト目でデータが切り捨てられる場合，その 2 バイト文字全体が入力されません。

- 外部 10 進項目，外部浮動小数点数字項目で定義したセルに入力したデータの有効文字が 22 バイトを超える場合，それ以降のデータは切り捨てられます。切り捨てが起きた場合，FILE STATUS 句を指定していると，入出力状態に 04 が返されます。
- 外部 10 進項目で定義したセルでは，小数点位置を合わせて入力されます。その際，必要に応じて数字の左側や右側にゼロを補ったり，あふれたけたを切り捨てたりします。詳細は，マニュアル「COBOL2002　言語　標準仕様編」「10.5.7　データ項目内でのデータのけた寄せ」を参照してください。

  また，このようなけた寄せが発生した場合，FILE STATUS 句を指定していると，入出力状態に 04 が返されます。
- 外部 10 進項目に符号の指定がある場合，SIGN 句の指定に関係なく左端の符号だけが有効となります。
- 外部浮動小数点数字項目で定義したセルには，外部 10 進形式，外部浮動小数点数字形式のどちらの形式のデータも，入力できます。ただし，入力データの中に"E"または"e"が含まれている場合，外部浮動小数点数字形式のデータとみなされます。
- 外部 10 進項目，外部浮動小数点数字項目以外の属性を持つ項目には，通常の-NumCsv オプション指定なしの場合と同様にデータが入力されます。
- 外部 10 進項目，外部浮動小数点数字項目で定義したセルに，次のようなデータを入力した場合，ゼロを入力したものとして扱われます。

  1.セル中に入力対象となるデータがない場合

  2.すべて空白文字のデータの場合
- 入力データの右端，または左端にある空白は，読み込まれません。

出力時のデータ変換規則

- 外部 10 進項目，外部浮動小数点数字項目に想定小数点が指定されている場合，想定小数点位置に出力される文字は DECIMAL-POINT IS COMMA 句の指定に従います。
- 外部浮動小数点数字項目に小数点が指定されている場合，小数点位置に出力される文字は DECIMAL-POINT IS COMMA 句の指定に従います。
- 外部 10 進項目に符号の指定がある場合，SIGN 句の指定に関係なく左端に符号が出力されます。ただし，符号がプラス（+）の場合は出力されません。

注意事項

- READ 文が失敗した場合，レコード領域は不定になります。
- 数字編集項目で定義したセルは，-NumCsv オプションを指定しても英数字項目属性でデータが入出力されます。

(3)　数値として入力するとき，不要な文字列を無視する機能

-NumCsv オプション指定時，セルデータ中に数値として無効な文字が含まれている場合，環境変数 CBLCSVCHAR にその文字を指定することで，文字を無視できます。この機能を使用すると，通貨記号などが付けられたデータを数値データとして読み込めます。

（例）

"¥123"，"¥456"，"¥789"という CSV 編成ファイルのデータを，-NumCsv オプションを指定して読み込もうとした場合

- 環境変数 CBLCSVCHAR に¥を指定したとき
  「123」「456」「789」という数値データとして読み込まれる。
- 環境変数 CBLCSVCHAR を指定しないとき
  ¥は数値データとして無効なので，実行時エラーになる。

形式

```
CBLCSVCHAR=’文字列〔;文字列…〕’
```

文字列

セルデータを数値として入力するとき，無視したい文字列を指定します。

規則

- 環境変数 CBLCSVCHAR に，複数の文字列を指定する場合は，各文字列をセミコロン（;）で区切り，全体をアポストロフィ（'）またはダブルコーテーション（"）で囲んで指定する必要があります。
- 環境変数 CBLCSVCHAR に指定した文字列は，数値の左端，右端，途中のどこにあっても無視されます。
- 入力データが外部浮動小数点数字項目の場合，環境変数 CBLCSVCHAR の指定は無効になります。
- 環境変数 CBLCSVCHAR には，1 バイト以上，512 バイト以下の文字列を指定してください。この範囲を超える長さの文字列を指定した場合，実行時エラーとなります。

注意事項

- -NumCsv オプションが指定されていない場合，環境変数 CBLCSVCHAR の指定は無効になります。
- 環境変数 CBLCSVCHAR に指定した文字列に含まれるセミコロン（;）は，すべて区切り文字とみなされます。無視する文字列としてセミコロンは指定できません。
- 環境変数 CBLCSVCHAR に，空白文字（X'20'）や数値データとして有効な文字を指定した場合，これらの文字列も無視されます。ただし，入力結果は保証しません。
- 環境変数 CBLCSVCHAR に設定する文字列の全体の長さは，1,024 バイト以内でなければなりません。1,024 バイトを超える文字列を設定した場合，メッセージを出力して環境変数 CBLCSVCHAR の指定は無効となります。

6.8.6　セルデータをダブルコーテーションで囲まないで出力する機能

環境変数 CBLD_ファイル名に NOCSVQUOTE を指定すると，CSV 編成ファイルを出力するとき，セルデータをダブルコーテーションで囲まないで出力できます。セルデータがダブルコーテーションで囲まれている形式に対応していない表計算プログラムとデータをやり取りするような場合，指定します。

形式

```
CBLD_ファイル名=NOCSVQUOTE
```

注意事項

セルデータをダブルコーテーションで囲まない場合，セルデータ中に含まれているコンマも区切り文字とみなされます。

（例）

「12,345,678」と「9,999」という二つのセルデータを出力する場合

- ダブルコーテーションで囲んだとき（CBLD_ファイル名=NOCSVQUOTE 指定なし）

  ”12,345,678”,”9,999”

  →「12,345,678」「9,999」という二つのセルデータとして扱われます。
- ダブルコーテーションで囲まないとき（CBLD_ファイル名=NOCSVQUOTE 指定あり）

  12,345,678,9,999

  →「12」「345」「678」「9」「999」という五つのセルデータとして扱われます。

6.8.7　入力時の未使用項目の初期化機能

(1)　入力時の未使用項目の初期化機能

CSV 編成ファイルから READ 文で入力したセルの個数が，対応するプログラム中の基本項目の個数より少ない場合，環境変数 CBLCSVINIT に YES を指定すると，セルと対応しない基本項目を初期化できます。

形式

```
CBLCSVINIT=YES
```

規則

- 環境変数 CBLCSVINIT に YES を指定すると，CSV 編成ファイルから入力したセルの個数が，対応するプログラム中の基本項目の個数より少ない場合，セルと対応しない基本項目を初期化します。
- 環境変数 CBLCSVINIT を指定しなかった場合や，YES 以外を指定した場合は，セルと対応しない基本項目は，初期化されません。
- CSV 編成ファイルで使用できる項目は，ブール項目以外で，用途が表示用（DISPLAY）の項目です。
- この機能は，-NumCsv オプションを使用した場合にも有効です。
- この機能を使用した場合，セルと対応しない基本項目には次の初期値データが設定されます。

未使用基本項目の属性	初期値データ
英字項目	半角空白文字（X'20'）
英数字項目	

未使用基本項目の属性		初期値データ
英数字編集項目		
数字編集項目		
数字項目	外部 10 進形式	ゼロ（X'30'）※1
	外部浮動小数点形式	
日本語項目		全角空白文字（X'8140'）※2※3
日本語編集項目		

注※1

初期値データは，数字項目にゼロを転記した結果となります。

注※2

環境変数 LANG に EUC コードが指定されている場合，EUC コードの全角空白文字（X'A1A1'）を初期値データに設定します。

注※3

Unicode 機能を使用している場合，バイトオーダによって全角空白文字（X'0030'），または全角空白文字（X'3000'）を初期値データに設定します。Unicode 機能については，「27. Unicode 機能」を参照してください。

注意事項

- この機能を使用した場合，空白文字やゼロだけのセルデータを入力した項目と，初期化で空白文字やゼロが設定された項目の違いが判断できません。

  この違いを判断する必要がある場合は，この機能を使用しないで，次に示すようなプログラムを作成する必要があります。

  環境変数 CBLCSVINIT によって初期化されたかどうかを見分けるための処理

  ・セルと対応しない基本項目は，READ 文の実行前にレコード領域をレコードとして存在しない値で初期化します。

  ・READ 文実行後，レコードの内容を初期化した値と比較します。
- この機能を使用して改行コードだけのレコードデータを入力した場合，ファイルに従属するすべての基本項目が未使用とみなされ，ファイルに従属するすべての基本項目が初期化されます。
- この機能は，ACCEPT 文を使用して CSV ファイルを入力する場合には使用できません。

6.8.8 データの後の空白文字を出力する機能

環境変数 CBLD_ファイル名に CSVWRITESPACE を指定すると，CSV 編成ファイルに出力するレコードの基本項目のデータの最後に半角空白文字がある場合，その半角空白文字も出力できます。

形式

```
CBLD_ファイル名=CSVWRITESPACE
```

規則

- 出力レコードの基本項目のデータの最後に半角空白文字がある場合，その半角空白文字も出力します。
- 出力する基本項目のデータがすべて半角空白文字の場合，対応するセルの情報には，基本項目のサイズ分の半角空白文字を出力します。

プログラム例

```
01 REC-1.
 02 CELL-1  PIC X(5).
 02 CELL-2  PIC X(5).
 02 CELL-3  PIC X(5).
       :
  MOVE ’ABC’     TO CELL-1.
  MOVE SPACE     TO CELL-2.
  MOVE ’△△CDE’ TO CELL-3.
  WRITE REC-1.
```

（凡例）

△：半角空白文字

データの後の半角空白文字を出力する機能を使用しない場合に出力されるレコードの結果（CBLD_ファイル名=CSVWRITESPACE 指定なし）

```
”ABC”,””,”△△CDE”
```

データの後の半角空白文字を出力する機能を使用した場合に出力されるレコードの結果（CBLD_ファイル名=CSVWRITESPACE 指定あり）

```
”ABC△△”,”△△△△△”,”△△CDE”
```

6.8.9　セルデータをタブ文字区切りで入出力する機能

環境変数 CBLD_ファイル名に CSVTABSEPARATED を指定すると，セルデータがタブ文字（X'09'）で区切られた形式のファイルを CSV 編成ファイルとして入出力することができます。CSVTABSEPARATED は，セルデータがタブ文字で区切られた形式に対応した表計算プログラムとデータをやり取りするような場合に指定します。

形式

```
CBLD_ファイル名=CSVTABSEPARATED
```

規則

- READ 文，および WRITE 文の動作は，区切り文字がコンマ（,）ではなくタブ文字となること以外は同じです。

6.9 HiRDB による索引編成ファイル（AIX で有効）

COBOL2002 では，HiRDB で作成されたデータベースの内容を，索引編成ファイルの言語仕様でアクセスできます。ここでは，HiRDB による索引編成ファイルの使用方法について説明します。

HiRDB による索引編成ファイルについては，マニュアル「COBOL2002　言語　拡張仕様編」「20. HiRDB による索引ファイル入出力機能」を参照してください。

6.9.1 プログラムの作成方法

HiRDB による索引編成ファイルを使用するには，次の指定が必要です。

コンパイラ環境変数の指定

HiRDB による索引編成ファイルを使用するプログラムのコンパイル時には，コンパイラ環境変数 CBL_RDBSYS に HiRDB を指定する必要があります。指定形式を次に示します。

形式

```
CBL_RDBSYS=HiRDB※
```

注※

データベースシステム名には，HiRDB だけが指定できます。この環境変数の指定がない場合は，初期値として HiRDB が仮定されます。

ORGANIZATION 句の指定

HiRDB による索引編成ファイルを使用する場合は，ORGANIZATION 句に RDB を指定します。

6.9.2 ファイルの作成と割り当て方法

HiRDB による索引編成ファイルを使用する場合，COBOL 上で指定したファイルは，HiRDB の表に相当します。

HiRDB による索引編成ファイルの作成方法と割り当て方法について説明します。

(1) ファイルの作成方法

HiRDB による索引編成ファイルは，次の方法で作成できます。

- SQL 文を利用して HiRDB にアクセスするユーザプログラム
- HiRDB ユティリティ

(2) ファイルの割り当て方法

操作対象となる表の名称を，ASSIGN 句で指定します。

形式

```
SELECT ファイル名 ASSIGN TO {定数｜外部装置名｜データ名}
```

- 定数指定の場合
  定数に，操作の対象となる表名を指定します。
- 環境変数指定の場合
  外部装置名に対応する環境変数に，操作の対象となる表名を指定します。
  次に指定方法を示します。

形式

```
CBL_外部装置名=表名
```

- データ名指定の場合
  データ名に，操作の対象となる表名を指定します。

表名の規則

操作対象に指定した表名は，HiRDB システムで規定された表名の規則に従います。ただし，表名に引用符（"）が含まれていない場合，実行時に引用符が付けられた名称で処理されます。

表名の規則については，「HiRDB の SQL リファレンスマニュアル」を参照してください。

6.9.3 ファイル編成とレコード形式

HiRDB による索引編成ファイルは，HiRDB が定めるファイル形式で構成されています。

6.9.4 HiRDB の定義と COBOL の定義の関連性

HiRDB による索引編成ファイルを使用する場合，HiRDB でのスキーマ定義と COBOL での定義との関連性や，SQL データ型と COBOL データ定義との関連性について，注意が必要です。HiRDB の定義と COBOL の定義の関連性について説明します。

(1) RDB の接続と切り離し

接続

COBOL の実行単位で，HiRDB による索引編成ファイルに対する最初の OPEN 文，または切り離し後の OPEN 文でファイルをオープンするときに，HiRDB の環境変数 PDUSER の設定に基づいて HiRDB と接続します。環境変数 PDUSER の詳細については，「HiRDB の UAP 開発ガイドマニュアル」を参照してください。

切り離し

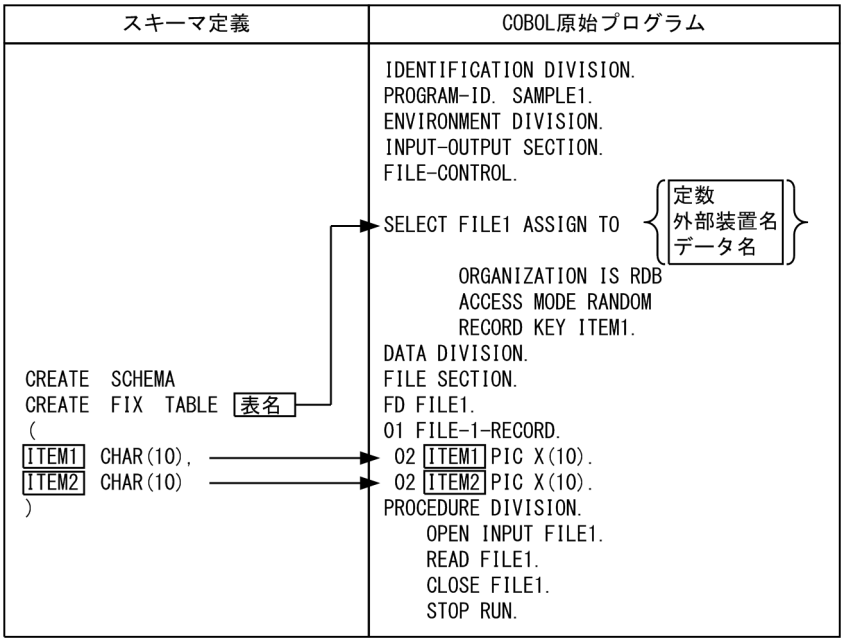
COBOL の実行単位で，HiRDB による索引編成ファイルに対するすべてのファイルを閉じたときに，HiRDB から切り離されます。

(2)　スキーマ定義と COBOL プログラムとの関連

HiRDB による索引編成ファイルを使用する場合は，はじめにデータベースを設計してからスキーマ定義を作成します。このスキーマ定義を基に，COBOL プログラムを作成します。

スキーマ定義と COBOL 原始プログラムとの関連を次に示します。

図 6-3　スキーマ定義と COBOL 原始プログラムとの関連



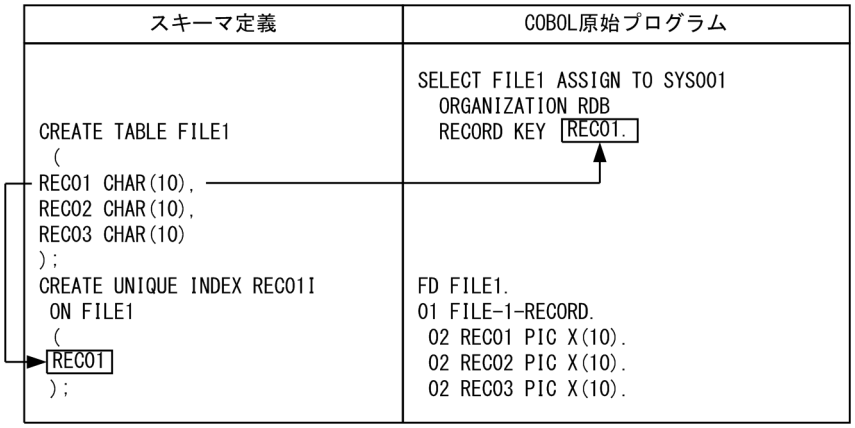
規則

- 表は，実表またはビュー表で定義します。
- スキーマ定義の表名を，ASSIGN 句の定数，外部装置名に関連づけられた環境変数，またはデータ名に設定します。
- スキーマ定義の列名と COBOL のレコードの項目名を一致させます。なお，-EquivRule オプションを指定する場合，等価規則が適用されなくてもスキーマ定義の列名と COBOL のレコードの項目名が一致するように，項目名を記述する必要があります。
- スキーマ定義の列と COBOL のレコード項目の属性は一致させなければなりません。一致していない場合，動作は保証しません。詳細は，「(3)　RDB の列のデータ型と COBOL のデータ記述」を参照してください。
- スキーマ定義の列と COBOL のレコードのデータ配置を一致させます。データの配置がずれるような境界調整はしないでください。
- スキーマ定義の長さと COBOL のレコードの長さを一致させます。

- 主／副レコードキーに指定した COBOL のデータ項目に対応する列は，インデクス定義を省略できます。ただし，重複キーを検知するには，インデクス定義時の CREATE INDEX で，UNIQUE を指定する必要があります。
- CREATE SCHEMA，CREATE TABLE，表名，列名の規則の詳細については，「HiRDB の SQL リファレンスマニュアル」を参照してください。
- WRITE 文で重複キーを検知する場合は，対応する列に対してユニークなインデクスを付けなくてはなりません。次にその例を示します。

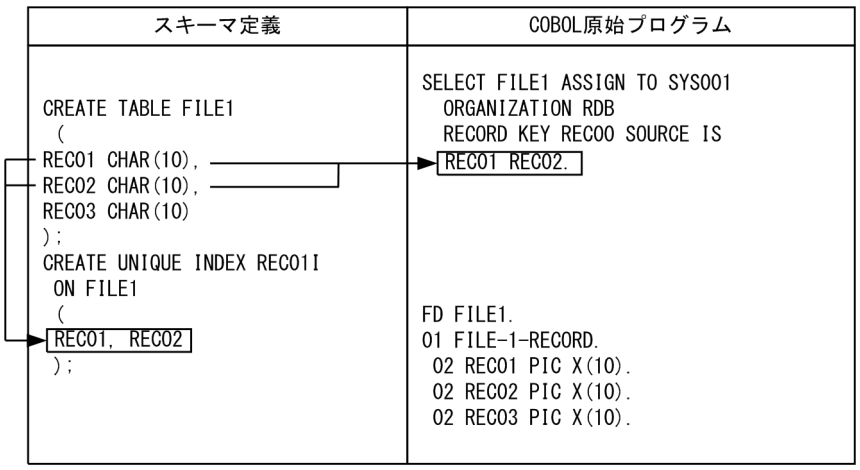
  1.単一列インデクスの場合（合成キー以外の場合）

  REC01 列の重複を検知する例



  2.複数列インデクスの場合（合成キーの場合）

  REC01，REC02 を合わせた値で重複を検知する例



（インデクスを付けるときの注意事項）

- WRITE 文で重複キーを検知しない場合は，対応する列に対してユニークなインデクスを付けてはなりません。
- WRITE 文で書き出すレコード中のキーデータ項目に対応する列以外に，ユニークなインデクスを付けてはなりません。キー以外の項目にユニークなインデクスを付けた場合，WRITE 文で正しくキー重複を検知できなくなります。例えば，レコード中のキーデータ項目に対応する列以外でキー重複を検知することがあります。

- 合成キーを使用する場合，CREATE INDEX で並べる列の順序と合成キーに指定する項目の順序は一致させなければなりません。この順序が一致していないときの動作は保証しません。

  CREATE INDEX の詳細については，「HiRDB の SQL リファレンスマニュアル」を参照してください。

(3) RDB の列のデータ型と COBOL のデータ記述

COBOL のレコードを定義するときは，RDB の列の属性に合わせて定義しなければなりません。

HiRDB の列のデータ型と COBOL のデータ記述の対応を次に示します。なお，この表にないデータ型は，使用できません。

表 6-6 HiRDB の列のデータ型と COBOL のデータ記述の対応

HiRDB の列のデータ型	COBOL のデータ記述	項目の記述	備考
SMALLINT	L1 基本項目名 PIC S9(4) USAGE COMP.	基本項目	
INTEGER	L1 基本項目名 PIC S9(9) USAGE COMP.	基本項目	
DECIMAL 〔(m〔,n〕)〕	L1 基本項目名 PIC S9(m-n)〔V9(n)〕 USAGE PACKED-DECIMAL.	基本項目	1≦m≦18, AIX(64)の場合で-MaxDigits38 オプション指定時 1≦m≦38, 0≦n≦m m ＝ n の場合は SV9(n)とする n ＝ 0 の場合は〔V9(n)〕を省略する
SMALLFLT	L1 基本項目名 USAGE COMP-1.	基本項目	
FLOAT	L1 基本項目名 USAGE COMP-2.	基本項目	
CHAR〔(n)〕	L1 基本項目名 PIC X(n).	基本項目	1≦n≦30000
VARCHAR(n)	L2 集団項目名. L3 基本項目名1 PIC S9(4) USAGE COMP. L3 基本項目名2 PIC X(n).	二つの基本項目から構成される集団項目 基本項目 1：文字長 基本項目 2：文字列	1≦n≦32000
NCHAR〔(n)〕	L1 基本項目名 PIC N(n).	基本項目	1≦n≦15000

HiRDB の列のデータ型	COBOL のデータ記述	項目の記述	備考
NVARCHAR(n)	L2 集団項目名. L3 基本項目名1 PIC S9(4) USAGE COMP. L3 基本項目名2 PIC N(n).	二つの基本項目から構成される集団項目 基本項目 1：文字長 基本項目 2：文字列	1≦n≦16000
DATE	L1 基本項目名 PIC X(4).	基本項目	4 バイト'yyyymmdd'の形式 （例えば 2002/7/11 の場合，X'20020711'が格納される）
TIME	L1 基本項目名 PIC X(3).	基本項目	3 バイト’hhmmss’の形式 （例えば 11:25:43 の場合，X'112543'が格納される）
INTERVAL YEAR TO DAY	L1 基本項目名 PIC S9(8) USAGE PACKED-DECIMAL.	基本項目	
INTERVAL HOUR TO SECOND	L1 基本項目名 PIC S9(6) USAGE PACKED-DECIMAL.	基本項目	
MCHAR［(n)］	L1 基本項目名 PIC X(n).	基本項目	1≦n≦30000
MVARCHAR(n)	L2 集団項目名. L3 基本項目名1 PIC S9(4) USAGE COMP. L3 基本項目名2 PIC X(n).	二つの基本項目から構成される集団項目 基本項目 1：文字長 基本項目 2：文字列	1≦n≦32000
BLOB	L2 集団項目名. L3 FILLER PIC S9(9) USAGE COMP. L3 基本項目名1 PIC S9(9) USAGE COMP. L3 基本項目名2 PIC X(n).	三つの基本項目名から構成される集団項目 基本項目 1：データ長 基本項目 2：バイナリデータ	1≦n≦65527

注 1

L1，L2，および L3 はレベル番号を表しています。

L1：レベル番号 01～49

L2：レベル番号 01～48

L3：レベル番号 02～49（ただし，L3 ＞ L2）

注 2

SQL のデータ型については，「HiRDB の SQL リファレンスマニュアル」を参照してください。

6.9.5　HiRDB による索引編成ファイル固有の機能と相違点

HiRDB による索引編成ファイルでは，HiRDB が持つ機能の一部を使用できます。また，ISAM による索引編成ファイルと相違があります。

(1)　トランザクション管理機能

HiRDB による索引編成ファイルのトランザクションは，最初の OPEN 文でファイルをオープンしたときに開始され，すべてのファイルが閉じられたときに暗黙的に COMMIT 文を実行し，終了します。なお，COMMIT 文および ROLLBACK 文を明示的には実行できません。COMMIT 文および ROLLBACK 文を明示的に実行した場合，実行時エラーが表示され，処理は中断されます。

HiRDB による索引編成ファイルでは，実行時環境変数 CBL_RDBCOMMIT を指定することで，HiRDB が持つトランザクション管理機能を制御できます。

(a)　環境変数 CBL_RDBCOMMIT

COMMIT 文および ROLLBACK 文で，RDB アクセスのトランザクションを管理する場合，環境変数 CBL_RDBCOMMIT に MANUAL，AUTO，YES のどれかを指定します。環境変数 CBL_RDBCOMMIT の指定方法を次に示します。

形式

```
CBL_RDBCOMMIT= {MANUAL | AUTO | YES}
```

MANUAL を指定した場合

明示的に COMMIT 文および ROLLBACK 文を実行できます。

AUTO または YES を指定した場合

明示的に CLOSE 文を実行した場合，および未クローズのファイルがクローズされた場合に，暗黙的に COMMIT 文が実行されます。

また，明示的に COMMIT 文および ROLLBACK 文を実行できます。

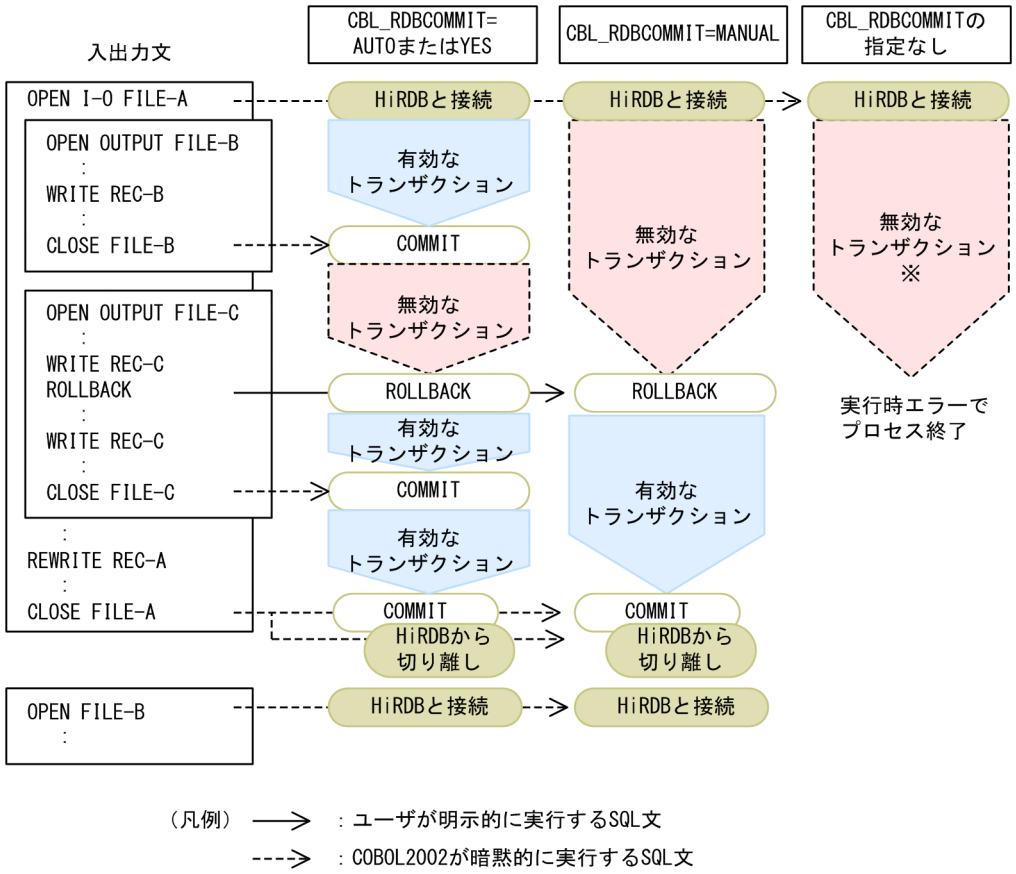
それ以外の値を指定した場合，または環境変数 CBL_RDBCOMMIT を指定しなかった場合

明示的に COMMIT 文および ROLLBACK 文を実行できません。

(b)　トランザクションの範囲

環境変数 CBL_RDBCOMMIT の値とトランザクションの範囲の関係を次に示します。なお，COMMIT 文，ROLLBACK 文の取り扱いの詳細については，「6.9.6　プログラムのコンパイルと実行」を参照してください。

図 6-4　環境変数 CBL_RDBCOMMIT の値とトランザクションの範囲



注※

HiRDB のトランザクション管理機能によって，HiRDB にアクセスを実行中プロセスが異常終了したことを検出して ROLLBACK が発生します。

(2)　ISAM による索引編成ファイルとの相違

HiRDB による索引編成ファイルと ISAM による索引編成ファイルとの相違点について説明します。

(a)　レコードの検索基準

レコードの検索基準の相違を，次に示します。

表 6-7　索引編成ファイルでのレコードの検索基準の相違

レコードキーの項目	キー判定属性	
	ISAM の場合	RDB の場合
固定長集団項目	文字	文字
外部 10 進項目	文字	文字
内部 10 進項目	文字	数値
外部浮動小数点数字項目	文字	文字

レコードキーの項目	キー判定属性	
	ISAM の場合	RDB の場合
数字編集項目	文字	文字
英字項目	文字	文字
日本語項目	文字	文字
2 進項目（2 バイト）	2 バイト固定小数点	2 バイト固定小数点
2 進項目（4 バイト）	4 バイト固定小数点	4 バイト固定小数点
内部浮動小数点数字項目（4 バイト）	単精度浮動小数点	単精度浮動小数点
内部浮動小数点数字項目（8 バイト）	倍精度浮動小数点	倍精度浮動小数点

(b)　入出力状態

HiRDB による索引編成ファイルでは，ISAM による索引編成ファイルと入出力状態に返される値が異なる場合があります。詳細は，「付録 G　入出力状態の値」を参照してください。

(c)　機能の相違点

1. スキーマ定義と一致したレコード記述でのアクセスが前提のため，ファイル記述項に記述できるレコード記述項は一つだけです。
2. スキーマ定義に可変長（行長が可変長）がないため，レコード記述から計算した長さを最大長とする固定長で入出力されます。このため，次のような制限があります。
   - ファイル記述項の DEPENDING ON データ名指定の RECORD 句は，データ名の内容に関係なく，常にレコード記述から計算した長さで入出力されます。
   - レコード記述中に DEPENDING ON 指定の OCCURS 句は，記述できません。
3. HiRDB による索引編成ファイルでは，DATA FORMAT 句を指定しても覚え書きとなります。
4. 環境変数 CBLD_ファイル名の ISAMDL／NOISAMDL オプションは常に ISAMDL として扱われます。また，環境変数 CBLISAMDL の値は，常に YES として扱われます。このため，OPEN 文の出力モードでファイルを開いた場合，ファイルは更新されないで新規作成となり，すべてのデータが削除されます。

   ただし，OPEN 文の出力モード，または拡張モードで開いたときに，ファイル実体（表）がない場合は，ファイル実体が作成されません。

   なお，出力モードでファイルを開く場合，すでにファイル内にあるレコードを高速で削除する機能があります。詳細は，「(5)　その他の拡張機能」の「(a)　出力ファイルの高速オープン機能」を参照してください。
5. 排他制御は RDB のスキーマ定義に従うため，COBOL のファイル共用は利用できません。また，RDB システムへのアクセスに用いる，内部的に入出力文から変換された SQL 文には，排他オプションが付けられません。このため，排他モードは，内部的に発行する SQL 文および実行環境によって RDB システムの規定に従います。

ただし，HiRDB による索引編成ファイルでは，固有のファイル共用を使用できます。詳細は，「(4) HiRDB による索引編成ファイル固有のファイル共用」を参照してください。

6. HiRDB による索引編成ファイルを使用するプログラムと，SQL を使用するプログラムを，同じ実行環境中に混在できません。
7. HiRDB による索引編成ファイルを使用するプログラムは，OpenTP1 環境下で使用できません。
8. トランザクション管理機能を使用しないで複数のファイルを同時に開いた場合，すべてのファイルを閉じるまでファイルに対しての追加・変更は反映されません。このため，追加書きしたレコードは，すべてのファイルを閉じるまで入力できません。
9. 複数の表を結合するビュー表に対して行の追加，更新，および削除はできません。
10. データ型が文字型（CHAR，VARCHAR，NCHAR，NVARCHAR，MCHAR，MVARCHAR）の列に対応させたデータ項目に，LOW-VALUE を使用できます。
11. レコードキーのデータが重複している場合，START 文や READ 文で重複したレコードキー値に位置づけた状態で READ 文を実行すると，そのあとに読み込まれるレコードの順序が不定となります。
12. START 文で次の指定を用いて位置づけた場合，そのあとに実行される READ 文は，常にキーの降順呼び出しとなります。
    - LESS THAN
    - LESS THAN OR EQUAL TO
    - NOT GREATER THAN
    - <
    - =<
    - NOT >
    - LAST 指定
13. レコードキーに指定した内部 10 進項目の照合順序は，文字属性の照合順序ではなく，数値の照合順序となります。
14. レコード中に次のデータが含まれる場合，入出力結果は保証しません。
    - レコード長が短い場合，または長い場合
    - 読み込むレコード中にナル値が含まれる場合
    - 書き出すレコード中に次のデータが含まれる場合

      a.データ項目の文字列長，またはデータ長の値が 1 より小さい

      b.DATE データ型の列に対応させたデータ項目の値が LOW-VALUE である。

      c.内部 10 進項目で定義したデータ項目の値が 0 で，かつ符号ビットが 0 である。

      レコード中に保証されないデータが含まれるかどうかをチェックする機能があります。詳細は，「(5) その他の拡張機能」の「(b)　データチェック機能」を参照してください。
15. データ型が BLOB の列に対応するレコードの項目は，キーにできません。
16. READ 文，WRITE 文，および REWRITE 文で重複キーを検知する場合，次の点に注意してください。

- HiRDB による索引編成ファイルの入出力では，ALTERNATE RECORD KEY 句の DUPLICATES 指定は覚え書きとなります。そのため，重複キーを検知するには，インデクス定義時の CREATE INDEX で UNIQUE を指定する必要があります。

  CREATE INDEX の詳細は，「HiRDB の SQL リファレンスマニュアル」を参照してください。

- 重複キーを検知するとき，インデクス定義での UNIQUE 指定の有無によって，入出力状態値が次のように異なります。

手続き文	HiRDB による索引編成ファイルの場合			ISAM による索引編成ファイルの場合	
	インデクス定義あり		インデクス定義なし	DUPLICATES 指定あり	DUPLICATES 指定なし
	UNIQUE 指定あり	UNIQUE 指定なし			
READ 文	00	00	00	02	00
REWRITE 文	22	00	00	02	22
WRITE 文	22	00	00	02	22

- 合成キーを使用するときは，CREATE INDEX で並べる列の順序と，合成キーに指定する項目の順序が一致している必要があります。順序が一致していないと，動作は保証しません。

(d) 物理的制限事項の相違点

HiRDB による索引編成ファイルの制限値を，次に示します。

キーに関する制限値

- 主レコードキーの最大長：255 バイト
- 副レコードキーの最大長：255 バイト
- 副レコードキーの最大定義個数：98 個
- 合成キーを構成する最大項目数：8 個
- 合成キーを構成する項目の合計最大長：255 バイト

同一実行環境中で RDB アクセスする索引編成ファイルの数

同一実行環境中で RDB アクセスする索引編成ファイルの数は，最大 63 個です。

また，HiRDB のシステム定義の pd_max_access_tables（同時に使用する表の数）に，使用するファイル数以上の値を指定しておく必要があります。

レコード長

定義できるレコード長は，65,535 バイト以内です。

ただし，CREATE TABLE 定義に FIX の指定がある場合は，CREATE TABLE 定義の<FIX 指定のある表>の列の長さの合計式に従います。CREATE TABLE 定義の詳細は，「HiRDB の SQL リファレンスマニュアル」の CREATE TABLE 定義の説明を参照してください。

作成できる列の数

HiRDB の一つの表に作成できる列の数は，最大 4,000 です。

(3) HiRDB サーバの設定での注意事項

(a) 環境変数 PDSWAITTIME の設定によるサーバの最大待ち時間の設定時の注意事項

HiRDB システムでは，HiRDB による索引編成ファイルを使用するクライアントからの要求に対する応答を返してから，次にクライアントから要求があるまでの HiRDB システムの最大待ち時間を，環境変数 PDSWAITTIME に秒単位で指定できます。

次に，環境変数 PDSWAITTIME を設定する場合の注意事項を示します。

- 環境変数 PDSWAITTIME に指定した時間での監視は，トランザクション開始から終了までの間，行われます。トランザクションについては，「(1) トランザクション管理機能」を参照してください。
- 実行中の HiRDB システムでは，指定した時間内にクライアントから次の要求が来ない場合，COBOL プログラムに異常が発生したものとみなし，トランザクションをロールバックします。
- 環境変数 PDSWAITTIME に 0 を指定した場合，HiRDB システムは，クライアント（COBOL プログラム）からの応答があるまで待ち続けます。
- ブロック転送機能を使用する場合，HiRDB システムからブロック転送されてきたレコードがなくなるまで，クライアント（COBOL プログラム）内で入力処理が行われます。このため，入力処理が終了するまで，クライアントから HiRDB システムに要求しません。したがって，ブロック転送機能を使用する場合，環境変数 PDSWAITTIME にはブロック転送文の入力処理時間を含めた値を設定してください。

環境変数 PDSWAITTIME の詳細は，「HiRDB の UAP 開発ガイドマニュアル」を参照してください。

(b) 環境変数 PDDDLDEAPRPEXE の設定時の注意事項

COBOL プログラム実行中に，環境変数 PDDDLDEAPRPEXE を設定した環境で定義系トランザクションを実行しないでください。環境変数 PDDDLDEAPRPEXE を設定した環境で定義系トランザクションを実行した場合，先行トランザクションの前処理が無効となるため，COBOL の管理情報との不整合が発生し，予期しない実行時エラーが発生するおそれがあります。

環境変数 PDDDLDEAPRPEXE の詳細は，「HiRDB の UAP 開発ガイドマニュアル」を参照してください。

(c) COBOL プログラムの異常終了時の対処

COBOL のプログラムが異常終了した場合，プロセスが待ち状態となります。待ち状態となったプロセスは，次の方法で終了させてください。

- 環境変数 PDSWAITTIME の設定によるトランザクションのロールバック

  HiRDB サーバ側で，環境変数 PDSWAITTIME にサーバの最大待ち時間を設定しておき，トランザクションをロールバックさせる。
- PDCANCEL コマンドによる強制終了

  PDCANCEL コマンドを入力してプロセスを強制終了させる。

  PDCANCEL コマンドの詳細については，「HiRDB のコマンドリファレンスマニュアル」を参照してください。

(4) HiRDB による索引編成ファイル固有のファイル共用

HiRDB による索引編成ファイルでは，ほかのファイル編成とは異なり，固有のファイル共用を使用できます。ここでは，HiRDB による索引編成ファイル固有のファイル共用について説明します。

指定形式

```
CBLRDBILWAIT=YES
```

機能

環境変数 CBLRDBILWAIT に YES を指定すると，ファイルを入力モードで開いたとき，内部的に発行される SELECT 文に対して「WITHOUT LOCK WAIT」の排他オプションを付けられます。排他オプションの詳細については，「HiRDB の SQL リファレンスマニュアル」を参照してください。

(5) その他の拡張機能

HiRDB による索引編成ファイルでは，次の拡張機能を使用できます。

(a) 出力ファイルの高速オープン機能

出力モードでファイルを開く場合，すでにファイル内にあるレコードを高速で削除したいときは，環境変数 CBLD_ファイル名に RDBOPURGE を指定するか，または環境変数 CBLRDBOPURGE に YES を指定してください。

指定形式（ファイル単位に指定する方法）

```
CBLD_ファイル名={ RDBOPURGE | NORDBOPURGE }
```

機能

RDBOPURGE を指定した場合，実行単位中の任意のファイルに対して OUTPUT モードでファイルを開いたときに，すべてのデータを削除するために内部的に発行する SQL 文を PURGE TABLE で行います。このとき，暗黙的に COMMIT 文を実行して現行のトランザクションを終了させるので，ほかのファイルを OPEN しているときなどは注意してください。

PURGE TABLE の詳細については，マニュアル「HiRDB の SQL リファレンスマニュアル」を参照してください。

- NORDBOPURGE を指定した場合，この機能は無効になります。

指定形式（実行単位中のすべてのファイルに指定する方法）

```
CBLRDBOPURGE=YES
```

機能

CBLRDBOPURGE=YES を指定した場合，実行単位中のすべてのファイルに対して OUTPUT モードでファイルを開いたときに，すべてのデータを削除するために内部的に発行する SQL 文を PURGE TABLE で行います。このとき，暗黙的に COMMIT 文を実行して現行のトランザクションを終了させるので，ほかのファイルを OPEN しているときなどは注意してください。

PURGE TABLE の詳細については，マニュアル「HiRDB の SQL リファレンスマニュアル」を参照してください。

- 環境変数 CBLRDBOPURGE に YES の指定がないか，または YES 以外を指定した場合は，この機能は無効になります。

規則

ファイル単位に指定する方法と実行単位中のすべてのファイルに指定する方法を併用した場合，ファイル単位に指定する方法が優先されます。次に指定の組み合わせによる動作を示します。

CBLRDBOPURGE	CBLD_ファイル名		
	RDBOPURGE	NORDBOPURGE	指定なし
YES	○	×	○
環境変数指定なし，または YES 以外	○	×	×

（凡例）

○：出力ファイルの高速オープン機能は有効

×：出力ファイルの高速オープン機能は無効

(b) データチェック機能

入出力時，レコード中に保証されないデータが含まれているかどうかをチェックする場合は，環境変数 CBLRDBDATAERR を指定します。

指定形式

```
CBLRDBDATAERR=YES
```

機能

環境変数 CBLRDBDATAERR に YES を指定すると，レコード中に次のような保証されないデータが含まれていないかがチェックされ，含まれていた場合には KCCC8353R-S の入出力エラーが出力されます。

（入出力が保証されないデータ）

- レコード長が短い場合，または長い場合
- 読み込むレコード中にナル値※が含まれる場合
- 書き出すレコード中に次のデータが含まれる場合
  - ・データ項目の文字列長またはデータ長が 1 より小さい
  - ・DATE データ型の列に対応させたデータ項目の値が LOW-VALUE である
  - ・内部 10 進データ項目で，値が 0 かつ符号ビットが 0 である

注※

ナル値については，「HiRDB の SQL リファレンスマニュアル」を参照してください。

(c) 内部発行される SQL 文で行値構成子を使用する機能

HiRDB による索引編成ファイルを使用した入出力で，実行時環境変数 CBLRDBROWVALCONSTRUCTOR が指定されているとき，COBOL2002 が内部で発行する SQL の SELECT 文で行値構成子を使用します。行値構成子を使用すると，次の条件すべてに該当する場合に，性能の向上が見込めます。

- 主レコードキー，または副レコードキーに合成キーが指定されていて，合成キーを構成する項目の個数が多い。
- 合成キーを構成する項目が HiRDB でインデクス定義されている。
- HiRDB のインデクス定義がメモリ上だけで処理されない（ディスクの実 I/O が発生する）。
- START 文，NEXT 指定の READ 文を繰り返す，または KEY 指定の READ 文を繰り返す。

この機能を使用するには，COBOL プログラム実行時にアクセスする HiRDB のバージョンが行値構成子をサポートしていることが前提です。サポートしていない場合は，動作は保証しません。

指定形式

```
CBLRDBROWVALCONSTRUCTOR=YES
```

機能

環境変数 CBLRDBROWVALCONSTRUCTOR に YES を指定すると，複数のキーを使用して入出力する場合，実行時ライブラリが内部で発行する SQL の SELECT 文に行値構成子が使用されます。

行値構成子が使用される条件

1. 主レコードキーまたは副レコードキーに合成キーが指定されていて，合成キーを構成する項目の数が 2 個以上である。
2. 1.のキーを指定したファイルに対して，次のどれかの文を実行した。
   ・START 文。ただし，FIRST 指定，LAST 指定の START 文は除く。
   ・KEY 指定の READ 文。
   ・乱アクセスまたは動的アクセスの REWRITE 文，DELETE 文で直前の入出力文が READ 文でない場合。

行値構成子を使用することで実行結果が変わることはありません。

環境変数 CBLRDBROWVALCONSTRUCTOR に YES 以外を指定した場合は，この機能は有効になりません。その場合，内部で発行する SQL 文に行値構成子は使用されません。

6.9.6 プログラムのコンパイルと実行

HiRDB による索引編成ファイルを使用するプログラムのコンパイル，および実行方法について説明します。

(1)　プログラムのコンパイル方法

HiRDB による索引編成ファイルを使用するプログラムのコンパイル方法を，次に示します。

形式（COMMIT／ROLLBACK 文を HiRDB による索引編成ファイルに適用しない場合）

```
ccbl2002 ファイル名 …
```

形式（COMMIT／ROLLBACK 文を HiRDB による索引編成ファイルに適用する場合）

```
ccbl2002 -RDBTran ファイル名 …
```

注意事項

- HiRDB による索引編成ファイルを使用するプログラムのコンパイル時には，コンパイラ環境変数 CBL_RDBSYS の指定も必要です。詳細は「6.9.1　プログラムの作成方法」を参照してください。
- COMMIT／ROLLBACK 文は，-RDBTran オプションの指定がある場合や，翻訳単位に HiRDB による索引編成ファイルの入出力の指定がある場合は HiRDB による索引編成ファイルに適用されます。ただし，DC シミュレーション機能と混在する場合はエラーとなります。それ以外は DC シミュレーション機能に適用されます。

  -RDBTran オプションの指定と，プログラムの記述による COMMIT／ROLLBACK 文の扱いを次に示します。

表 6-8　-RDBTran オプションの指定と COMMIT／ROLLBACK 文の扱い

プログラムの記述		通信節	ファイル管理記述項	
			HiRDB による索引編成ファイル（RDB）あり	HiRDB による索引編成ファイル（RDB）なし
オプションの指定	-RDBTran 指定あり	データコミュニケーション機能なし	RDB に適用	RDB に適用
		データコミュニケーション機能あり	S レベルエラー	S レベルエラー
	-RDBTran 指定なし	データコミュニケーション機能なし	RDB に適用	DC に適用
		データコミュニケーション機能あり	S レベルエラー	DC に適用

（凡例）

RDB に適用：HiRDB による索引編成ファイルの COMMIT／ROLLBACK 文となる

DC に適用：DC シミュレーション機能の COMMIT／ROLLBACK 文となる

S レベルエラー：コンパイル時に重大エラーとなる

(2)　プログラムのリンク

HiRDB による索引編成ファイルを使用するプログラムのリンクは，ccbl2002 コマンド，cc コマンドまたは ld コマンドで HiRDB のライブラリを指定する必要があります。詳細は「34.1.3　ccbl2002 コマンドの-l オプション」および「34.1.5　cc コマンドおよび ld コマンドの-l オプション」を参照してください。

(3) プログラムの実行方法

HiRDB による索引編成ファイルを使用するプログラムは，次の手順で実行してください。

1. HiRDB のサーバを起動する
2. HiRDB のクライアントの環境変数で，PDHOST，PDUSER，PDNAMEPORT の値をサーバ環境に合わせて設定する
3. HiRDB による索引編成ファイルを使用した COBOL プログラムを実行する

6.9.7 プログラム作成時の留意点

HiRDB による索引編成ファイルを使用するプログラムで，処理速度の速いプログラムを作成する場合の留意点を示します。

動的アクセス（DYNAMIC）を使用しない

理由

動的アクセスで KEY 指定の READ 文を実行すると，該当レコードの読み込みと同時に以降のレコードの読み込み準備をするため，処理に時間が掛かります。

対策

KEY 指定の READ 文で読み込んだレコードの次のレコードを読み込む必要がなければ，乱アクセス（RANDOM）での KEY 指定の READ 文を使用してください。乱アクセスでの KEY 指定の READ 文を使用すると，以降のレコードの読み込み準備が不要なため，処理時間を短縮できます。

I-O モードでファイルを開かない

理由

I-O モードでファイルを開くと，更新を前提とする SQL 文で RDB アクセスが行われるため，処理に時間が掛かります。

対策

DELETE 文，REWRITE 文，または WRITE 文でレコードを更新する必要がない場合は，INPUT モードでファイルを開いてください。更新を前提としない SQL 文で RDB にアクセスできるので，処理時間を短縮できます。

START 文を使用しない

理由

START 文を使用すると，該当するキー条件に一致するデータを検索したあと，NEXT 指定の READ 文でレコードを読み込めるように，以降のレコードの読み込み準備をするため，処理に時間が掛かります。

対策

START 文および NEXT 指定の READ 文でのレコードの読み込みを KEY 指定の READ 文で代替できる場合は，KEY 指定の READ 文を使用してください。KEY 指定の READ 文を使用すると，以降のレコードの読み込み準備が不要なため，処理時間を短縮できます。

キーに使用する項目（列）を一つにする

理由

複数のキーを使用して入出力をすると，それぞれのキーで関連づけをするため，OPEN 文，START 文実行後の最初の READ 文，および KEY 指定の READ 文の処理に時間が掛かります。

対策

使用するキーの数を減らしてください。データの関連づけに掛かる時間を短縮できます。

環境変数 CBLD_ファイル名=RDBOPURGE または環境変数 CBLRDBOPURGE=YES を指定する

理由

ファイルを OUTPUT モードでオープンする場合，ファイル中のすべてのデータが削除されます。このとき，多量のデータがあると，データの削除に時間が掛かります。

対策

ファイルを OUTPUT モードでオープンする場合は，環境変数 CBLD_ファイル名=RDBOPURGE または環境変数 CBLRDBOPURGE に YES を指定してください。データを削除するために内部的に発行する SQL 文に PURGE TABLE を使用することで削除の時間を短縮することができます。

6.10　ラージファイル入出力機能

ラージファイル入出力機能を使用すると，ラージファイル（ファイルサイズが 2GB 以上のファイル）に対する入出力ができます。

6.10.1　ラージファイル入出力機能の概要

実行時環境変数 CBLD_ファイル名に LARGEFILE を指定することによって，ラージファイル（ファイルサイズが 2GB 以上のファイル）に対する入出力ができます。

(1)　ラージファイル入出力機能の指定方法

- ファイル単位に指定する方法

形式

```
CBLD_ファイル名= { LARGEFILE | NOLARGEFILE }
```

LARGEFILE

ラージファイル入出力機能が有効になります。

NOLARGEFILE

ラージファイル入出力機能を使用しないで，通常の入出力をします。

実行時環境変数 CBLD_ファイル名の詳細については，「35.3　プログラムの実行環境の設定」を参照してください。

- 実行単位中のすべてのファイルに指定する方法

形式

```
CBLLARGEFILE=YES
```

注意事項

実行時環境変数 CBLD_ファイル名=LARGEFILE/NOLARGEFILE と，CBLLARGEFILE=YES を同時に指定した場合，実行時環境変数 CBLD_ファイル名=LARGEFILE/NOLARGEFILE の指定が優先されます。

実行時環境変数「CBLD_ファイル名」と「CBLLARGEFILE」の関係を次に示します。

CBLLARGEFILE	CBLD_ファイル名		
	LARGEFILE	NOLARGEFILE	指定なし
YES	○	×	○
YES 以外，または実行時環境変数指定なし	○	×	×

（凡例）

○：ラージファイル入出力機能を適用する

×：ラージファイル入出力機能を適用しない

(2) 使用できるファイルの種類

ファイル編成

ラージファイル入出力機能は，次のファイル編成で使用できます。

- 順編成ファイル
- テキスト編成ファイル
- CSV 編成ファイル

ファイルシステム

ラージファイルを使用するためには，ファイルシステムの属性がラージファイルに対応している必要があります。ファイルシステム属性については，システムのマニュアルを参照してください。

また，作成できるファイルサイズについては，システム資源の制限が設定されている場合があります。詳細は，システムの ulimit コマンドなどについて記載されたマニュアルを参照してください。

6.10.2 ラージファイル入出力機能でのファイルの共用

ラージファイル入出力機能を使用した場合，LOCK MODE 指定は無視され，ファイルシェア機能は無効となります。また，INPUT モードで開いた場合でも，他プロセスからのレコード施錠はエラーとなります。

ラージファイル入出力機能を使用した場合の入出力動作は次のとおりです。

(1) 先行プロセスでラージファイル入出力機能を使用する場合

後行プロセス			先行プロセス			
ラージファイル入出力機能	OPEN モード		OPEN モード			
			INPUT	OUTPUT	I-O	EXTEND
使用する	OPEN	INPUT	○	×	×	×
		OUTPUT	×	×	×	×
		I-O	×	×	×	×
		EXTEND	×	×	×	×
使用しない※1	OPEN※2	INPUT	○	×	×	×
		OUTPUT	×	×	×	×
		I-O	○	×	×	×
		EXTEND	×	×	×	×

後行プロセス			先行プロセス			
ラージファイル入出力機能	OPEN モード		OPEN モード			
			INPUT	OUTPUT	I-O	EXTEND
	READ※3	WITH LOCK	RE	−	−	−
		WITH NOLOCK	○	−	−	−
	REWRITE※3※4		○	−	−	−

（凡例）

○：成功

×：OPEN エラー

RE：READ エラー

−：対象外

注※1

LOCK MODE IS AUTOMATIC 指定

注※2

WITH LOCK 指定なし

注※3

OPEN I-O モードで開いたファイル

注※4

READ WITH NO LOCK で入力したレコードに対する REWRITE 文

(2)　先行プロセスでラージファイル入出力機能を使用しない場合

「7.2.4　各実行単位の施錠形式」に従います。

ただし，後行プロセスでラージファイル入出力機能を使用する場合は，LOCK MODE 指定なしの動作となります。

6.10.3　ネットワークファイルシステムでのラージファイルの動作

ネットワークファイルシステムでラージファイル入出力機能を使用できます。

ただし，AIX の場合は，双方のシステムが AIX で，かつラージファイルをサポートしているときに，ラージファイルを扱えます。ほかのシステムのファイルの場合，ラージファイルを扱えません。

6.10.4　ラージファイル入出力機能の制限事項

COBOL 入出力サービスルーチンでは，ラージファイル入出力機能を使用できません。ただし，順編成ファイルでは，COBOL 入出力サービスルーチンを使用してラージファイルに対する入出力ができます。相対

編成ファイルの場合，COBOL 入出力サービスルーチンを使用してラージファイルに対する入出力はできません。

6.11　ファイル入出力拡張機能

ファイル入出力の拡張機能について説明します。

6.11.1　ファイルサイズがレコード長の整数倍でない固定長形式の順ファイルの入出力

固定長形式の順ファイルで，ファイルサイズがレコード長の整数倍でない場合に，最終レコードの長さが定義レコード長より短くても入力できるようにする機能です。

(1)　環境変数の指定

環境変数 CBLD_ファイル名に SAMENDIO を指定すると，最終レコードが定義レコード長より短い場合でもエラーにならなくなり，残りのレコードの入出力を行えます。

形式

```
CBLD_ファイル名=SAMENDIO
```

環境変数 CBLD_ファイル名に SAMENDIO を指定した場合としない場合とで，入出力時の結果が異なります。なお，SAMENDIO を指定しない場合には，NOSAMENDIO が仮定されます。詳細は，「35.3.2 実行時環境変数の一覧」を参照してください。

(2)　環境変数の指定有無による結果の違い

環境変数 CBLD_ファイル名に SAMENDIO を指定した場合としない場合とで，それぞれの入出力文の動作の違いを次に示します。

(a)　READ 文

表 6-9　入力するレコードがレコード長より短い場合の READ 文の動作

CBLD_ファイル名の指定	READ 文の動作
NOSAMENDIO または指定なし	入力するレコードがレコード長より短い場合，エラーになります。 このとき，入出力状態の値は「30」になります。
SAMENDIO	入力するレコードがレコード長より短い場合，エラーにならないで残りのレコードを読み込みます。レコード長の残りには，半角の空白「X'20'」を埋めます。このとき，入出力状態の値は「00」になり，プログラムの実行は継続します。

(b)　REWRITE 文

REWRITE 文を実行するには，直前の READ 文が実行済みでなければなりません。

環境変数 CBLD_ファイル名に SAMENDIO を指定した場合，直前の READ 文で読み込んだ分のレコードを書き換えます。

表 6-10　更新するレコードがレコード長より短い場合の REWRITE 文の動作

CBLD_ファイル名の指定	REWRITE 文の動作
NOSAMENDIO または指定なし	直前の READ 文が実行時エラーになるので，REWRITE 文も実行されません。
SAMENDIO	直前の READ 文で読み込んだレコード長が，定義レコード長よりも短い場合は，READ 文で読み込んだ分のレコードを書き換えます。このとき，入出力状態の値は「00」になります。

(c)　その他の入出力文

READ 文および REWRITE 文以外の入出力文での入出力機能は，ファイルサイズがレコード長の整数倍である固定長ファイルの場合と同じです。

(3)　注意事項

- この機能を使用してファイル中の最終レコードを読み込むとき，最終レコードがプログラム中で定義したレコード長より短いために，定義したレコード長より短いレコード長しか読み込めなくても COBOL はエラーや警告メッセージを出力しません。そのため，実際に読み込んだレコード長は，ユーザプログラム側で管理しておく必要があります。
- この機能は，ファイル内の全データをレコードとして入力できるだけで，レコードフォーマットの形態，レコードのブロッキングおよびデブロッキングはユーザプログラム側で管理しておく必要があります。

6.11.2　ファイルバッファサイズ指定機能

ファイルバッファサイズ指定機能を使用すると，COBOL のファイル入出力機能で使用するファイルのバッファサイズを，使用する環境に合わせて任意に指定できます。バッファサイズを大きくすることで，ファイルに対する物理的な入出力回数を減らすことができ，性能向上を目的としたチューニングが可能となります。

この機能を使用しない場合，COBOL ではファイルのバッファサイズを 4,096 バイトに設定します。

この機能は，次に示すファイルを対象としています。

- 順編成ファイル※1
- 相対編成ファイル※1 ※2
- テキスト編成ファイル
- CSV 編成ファイル

注※1

環境変数 CBLINBUFSIZE/CBLOUTBUFSIZE については，COBOL 入出力サービスルーチンを使用した入出力機能でも有効になりますが，環境変数 CBLD_ファイル名の NOINBUFSIZE/NOOUTBUFSIZE については，COBOL 入出力サービスルーチンを使用した入出力機能では有効になりません。

注※2

ACCESS MODE 句が SEQUENTIAL 指定だけ有効です。DYNAMIC/RANDOM 指定では有効になりません。

この機能の作用対象を次に示します。

OPEN モード		INPUT		OUTPUT		EXTEND		I-O	
WITH LOCK 指定		あり	なし	あり	なし	あり	なし	あり	なし
LOCK MODE 句の指定	なし	○	○	○	○	○	○	×	×
	EXCLUSIVE	○	○	○	○	○	○	×	×
	AUTOMATIC	○	×※2	○	○	○	×※2	×	×
	MANUAL※1	○	×※2	○	○	○	×※2	×	×

（凡例）

○：この機能の対象

×：この機能の対象外

注※1

順編成ファイル（COBOL 入出力サービスルーチン機能も含む）およびテキスト編成ファイルの場合は，LOCK MODE 句に MANUAL 指定はできません。

注※2

CSV 編成ファイルの場合は，この機能の対象（○）となります。

(1) 環境変数の指定

この機能を使用するには，次に示す環境変数を指定する必要があります。

環境変数名	環境変数の有効範囲	環境変数の機能内容
CBLINBUFSIZE=nnnnnnnnnn※1	対象ファイルに有効※2	OPEN 文のモードが INPUT 指定のバッファサイズ制御によるファイル入力時に，使用するバッファサイズを 2,000,000,000（約 2GB）までの数値で指定する。
CBLOUTBUFSIZE=nnnnnnnnnn※1		OPEN 文のモードが OUTPUT/EXTEND 指定のバッファサイズ制御によるファイル出力時に，使用するバッファサイズを 2,000,000,000（約 2GB）までの数値で指定する。

環境変数名	環境変数の有効範囲	環境変数の機能内容
CBLD_ファイル名=NOINBUFSIZE	ファイル別に有効	指定されたファイルのバッファサイズ制御による入力時に，環境変数 CBLINBUFSIZE で設定された値が無効になる。無効になった場合のバッファサイズは，4,096 バイトになる。
CBLD_ファイル名=NOOUTBUFSIZE		指定されたファイルのバッファサイズ制御による出力時に，環境変数 CBLOUTBUFSIZE で設定された値が無効になる。無効になった場合のバッファサイズは，4,096 バイトになる。

注※1

「nnnnnnnnnn」には，COBOL のファイル入出力機能で使用するバッファ長として，1 ～2000000000 の数字を指定します。

注※2

プロセス内の対象ファイルに有効になります。

各環境変数間の関係を次に示します。

環境変数 CBLINBUFSIZE/CBLOUTBUFSIZE の指定	環境変数 CBLD_ファイル名の指定		
	指定なし	NOINBUFSIZE	NOOUTBUFSIZE
CBLINBUFSIZE	○	×	−
CBLOUTBUFSIZE	○	−	×

（凡例）

○：環境変数 CBLINBUFSIZE/CBLOUTBUFSIZE のバッファサイズの任意の値が有効

×：バッファサイズの任意の値が無効（バッファサイズは 4,096 バイト）

−：各環境変数の間に関連性はない（バッファサイズの任意の値が有効）

(2) 指定例（B シェル）

環境変数の指定例を次に示します。

```
CBLINBUFSIZE=32768  プロセス内の入力時にファイルバッファを32,768バイトとする
export  CBLINBUFSIZE
CBLOUTBUFSIZE=8192  プロセス内の出力時のファイルバッファを8,192バイトとする
export  CBLOUTBUFSIZE

CBLD_FILE01=NOINBUFSIZE  ファイル名FILE01について，入力の指定を取り消す※1
export  CBLD_FILE01      出力を取り消す場合はNOOUTBUFSIZEを指定する
```

注※1

複数のオプションを指定する場合は，次に示すように各オプションをコロン（：）で区切ります。

CBLD_FILE01=NOINBUFSIZE:NOOUTBUFSIZE

export CBLD_FILE01

(3) 規則

- 環境変数に指定するバッファサイズは，レコード長より大きい値を指定してください。
- 環境変数の指定に関して，次に示す場合は環境変数の指定は無効となり，ファイルのバッファサイズは，4,096 バイトになります。このとき，エラーメッセージなどは出力しません。
  - バッファサイズ 1～2,000,000,000 以外
  - 半角数字以外の文字
- 環境変数には，約 2GB までの値を指定できますが，実際に有効になる値は，マシン環境で使用できるメモリサイズに依存します。指定されたメモリサイズが確保できない場合は，システムがバッファを自動的に割り当てて動作します。
- OPEN 文のモードが I-O 指定の場合は，この機能の対象外となります。
- 環境変数名で指定する「ファイル名」は SELECT 句で指定したファイル名に対応します。ただし，ファイル名内のハイフン（-）はアンダーバー（_）に置き換えて指定してください。指定例を次に示します。

（COBOL での記述例）

```
SELECT A-FILE ASSIGN TO SYS000.
```

（環境変数の指定例（B シェル））

```
CBLD_A_FILE=NOINBUFSIZE
export CBLD_A_FILE
```

(4) 注意事項

ファイル入出力でのバッファサイズは，使用されるファイルサイズ，ディスク装置・マシン環境，およびシステムの仕様に依存することから実際に性能検証によって適切な値を使用してください。

7 ファイル共用（ファイルシェア）

この章では，COBOL プログラムでのファイル共用の方法について説明します。

7.1　ファイル共用（ファイルシェア）の概要

ファイル共用（ファイルシェア）とは，マルチユーザ環境でユーザ間のファイルを共用する機能です。この機能を使用すると，データの更新時にレコードまたはファイル全体を，ほかの COBOL プログラムと共用したり保護したりできます。

表 7-1　ファイル共用を使用できるファイル編成

ファイル編成	ファイル共用		備考
	ファイルレベル	レコードレベル	
順ファイル	○	○	
相対ファイル	○	○	
ISAM による索引編成ファイル	○	○	
テキストファイル	○	×	
CSV ファイル	×	×	
HiRDB による索引編成ファイル	HiRDB による索引編成ファイル固有のファイル共用を使用できる。		詳細は「6.　ファイル入出力機能」を参照。

（凡例）

○：ファイル共用が使用できる

×：ファイル共用が使用できない

ISAM による索引ファイル以外

マルチスレッド対応 COBOL プログラムに対応しません。

ISAM による索引ファイル

マルチスレッド対応 COBOL プログラムに対応します。

ただし，同一スレッド内で一つのファイル単位または一つのレコード単位を使用する場合，ファイル共用は使用できません。

7.2　ファイル共用の詳細

ファイル共用の詳細について説明します。

ファイル共用については，マニュアル「COBOL2002　言語　拡張仕様編」「3.　ファイル共用機能」を参照してください。

7.2.1　ファイルレベルのファイル共用

ファイルは，常に次の活性状態か不活性状態のどちらかの状態にあります。

- 活性状態
  ファイルが一つ以上の実行単位に対して開かれている状態
- 不活性状態
  どの実行単位からも開かれていない状態

活性状態のファイルは，ほかの実行単位から OUTPUT モードで開けません。存在しないファイルは，OUTPUT モードで自動的に生成されますが，このファイルはほかの COBOL プログラムとは共用できません。

活性状態のファイルのモードには，次に示す排他モードと共用モードの二つがあります。

(1)　排他モード

排他モードのファイルは，一つの実行単位に対してだけ開かれています。ほかの実行単位がこのファイルを使用しようとしても，ファイル施錠のエラーが返り，使用が拒否されます。

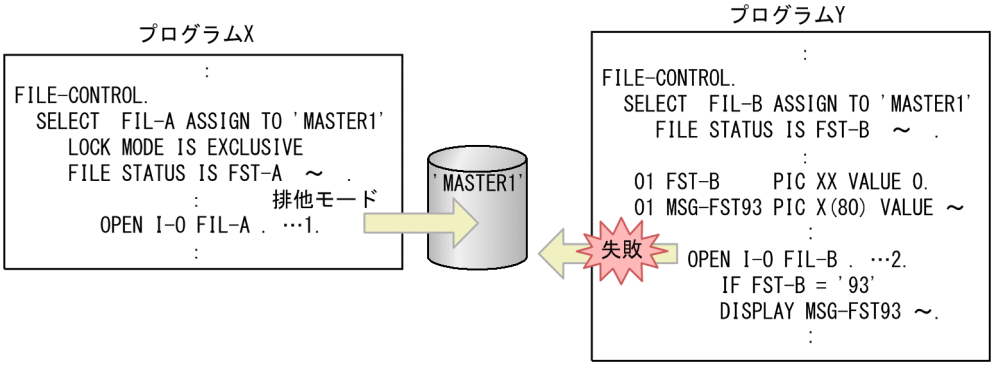
排他モードでは，ファイルが一つの実行単位に対して施錠され，その実行単位だけがファイルを使用できます。この実行単位がファイルを閉じるとファイルの施錠は解除されます。

順編成ファイル，相対編成ファイルおよびテキスト編成ファイルの場合，排他モードで開くときは，INPUT モードであってもファイルに対して書き込み権限が必要です。

(2)　共用モード

共用モードのファイルは，複数のプログラムから使用できます。各実行単位がファイルを使うとき，一つまたは複数のレコード単位に施錠すれば，データを保護できます。

次に，ファイルレベルのファイル共用の例を示します。



1. プログラム X が，排他モードでファイル'MASTER1'を開きます。
2. プログラム Y が'MASTER1'を開こうとしますが，すでにプログラム X が排他モードで開いているため，OPEN 文は失敗します。このとき，入出力状態には 93 が返されます。

7.2.2　レコードレベルのファイル共用

(1)　ファイルの開き方による相違

INPUT モードで開かれたファイルは共用できますが，ファイルのレコードは施錠できません。I-O または EXTEND モードで開かれたファイルは共用できます。

ファイルを EXTEND モードで開いている間に，ほかのファイル単位が INPUT または I-O モードで同じファイルを開いた場合，完全な論理ファイル（EXTEND モードで追加したレコードを含めた論理ファイル）のレコードを読み込めます。しかし，EXTEND モードで開かれたほかの実行単位については，レコード施錠によってレコードの出力を防げません。

INPUT モードで開かれた共用ファイルは，ほかのファイル単位からのレコードの参照，更新を防げません。ファイルを EXTEND モードで開いている間に，ほかのファイル単位が INPUT または I-O モードで開いてレコードを読み込もうとした場合，EXTEND モードで出力されたレコードについては，読み込みが保証されます。ただし，LOCK MODE 句の指定のない INPUT モードの場合，読み込みは保証しません。

ほかのファイル単位が EXTEND モードで開かれた場合に，双方に追加されるレコードの内容については，マニュアル「COBOL2002　言語　拡張仕様編」「3.　ファイル共用機能」を参照してください。

(2)　レコード施錠

単一のレコードを施錠する場合と複数のレコードを施錠する場合についてそれぞれ説明します。

(a)　単一レコード施錠

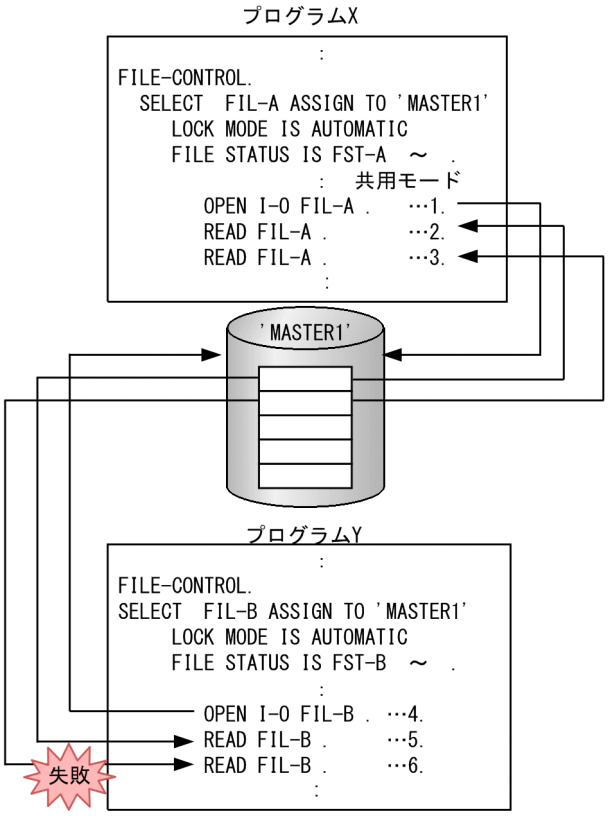
単一レコード施錠を指定した実行単位は，一つのファイル内に一つだけのレコード施錠を取得できます。

単一レコード施錠では，レコード施錠の解放を UNLOCK 文で指定できます。また，新しいレコードを施錠すると，それまでのレコード施錠は自動的に解放されます。同じ実行単位でレコード施錠が解放される条件を次に示します。

- START 文以外の入出力文が成功したとき
- ファイルを閉じたとき

START 文ではレコード施錠は解放されません。また，レコード施錠の取得，解放に失敗したとき，レコード施錠の状態は保証しません。

次に，単一レコード施錠の例を示します。



1. プログラム X が，共用モードでファイル'MASTER1'を開きます。
2. READ 文が実行されると，1 番目のレコードが施錠されます。
3. 次の READ 文が実行されると，1 番目のレコードの施錠が解除され，2 番目のレコードが施錠されます。
4. プログラム Y が，共用モードでファイル'MASTER1'を開きます。
5. READ 文が実行されると，1 番目のレコードが施錠されます。
6. READ 文が実行されますが，2 番目のレコードはプログラム X によって施錠されているため，READ 文は失敗します。

(b) 複数レコード施錠

複数レコード施錠を指定した実行単位は，一つのファイルに複数のレコード施錠を同時に取得できます。施錠されたレコードはほかの実行単位から使用できませんが，施錠されていないレコードは使用できます。

複数レコード施錠は，相対または索引ファイルにだけ使用できます。

複数レコード施錠は，LOCK MODE 句に MANUAL を指定し，WITH LOCK 指定の READ 文を実行するごとにレコード単位で取得できます。WITH LOCK 指定のない READ 文を実行した場合，レコード施錠は取得されません。このときでも，取得済みのレコード施錠は解放されません。

同じ実行単位でレコード施錠が解放される条件は次のとおりです。

- ファイルに対して UNLOCK 文を実行したとき
- ファイルを閉じたとき

レコード施錠の取得，解放に失敗したとき，レコード施錠の状態は保証しません。

また，COBOL 入出力サービスルーチンで作成した相対ファイルでも，複数レコード施錠を使用できます。

7.2.3 ファイルの排他・共用の区別

ファイルの排他・共用の区別を次に示します。

表 7-2 ファイルの排他・共用の区別

OPEN モード		INPUT		I-O		OUTPUT		EXTEND	
WITH LOCK 指定		あり	なし	あり	なし	あり	なし	あり	なし
LOCK MODE 句の指定	なし	×	○	×	×	×	×	×	×
	EXCLUSIVE	×	×	×	×	×	×	×	×
	AUTOMATIC	×	○	×	○	×	×	×	○※
	MANUAL	×	○	×	○	×	×	×	○※

（凡例）

○：共用

×：排他

注※

EXTEND モードで開かれ共用されているファイルに対して追加書きする場合は，複数のプログラムから実行しないようにしてください。詳細については，マニュアル「COBOL2002 言語 拡張仕様編」「3.2.8 WRITE 文（ファイル共用機能）」の説明を参照してください。

7.2.4　各実行単位の施錠形式

(1)　順編成ファイル，相対編成ファイル，ISAM による索引編成ファイルおよびテキスト編成ファイル（ファイルレベルのファイル共用だけ該当）の場合

順編成ファイル，相対編成ファイル，ISAM による索引編成ファイル，およびテキスト編成ファイル（ファイルレベルのファイル共用だけ該当）の場合の各実行単位の施錠形式を次に示します。

後行プロセス

先行プロセス		ASSIGN LOCK MODE の指定	MANUAL※			AUTOMATIC				EXCLUSIVE	指定なし			MANUAL入出力処理チェック			AUTOMATIC入出力処理チェック		
		OPENモードの指定	INPUT		I-O	INPUT		OUTPUT	I-O		INPUT		I-O		INPUT	I-O		INPUT	I-O	
		READ文のLOCK MODE指定	WITH LOCK / WITH NO LOCK	指定なし	WITH LOCK / WITH NO LOCK	WITH LOCK / WITH NO LOCK	指定なし		WITH LOCK / WITH NO LOCK	指定なし	WITH LOCK / WITH NO LOCK	指定なし	WITH LOCK / WITH NO LOCK	指定なし	START	REWRITE	START	START	REWRITE	START
MANUAL※	INPUT	WITH LOCK	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$		○	○	○	○	○	○
		WITH NO LOCK	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$		○	○	○	○	○	○
		指定なし	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$		○	○	○	○	○	○
		OUTPUT	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	0_E 0_E 0_E 0_E			0_E 0_E 0_E 0_E		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$		0_E	0_E	0_E	0_E	0_E	0_E
	I-O	WITH LOCK	$R_ER_ER_E0_E$		$R_ER_ER_E-$	R_E R_E R_E 0_E			R_E R_E R_E −		$0_E0_E0_E0_E$		△△△0_E		○	E_R	○	○	E_R	○
		WITH NO LOCK	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		○ ○ ○ 0_E		○	○	○	○	○	○
		指定なし	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		○ ○ ○ 0_E		○	○	○	○	○	○
		EXTEND	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		○ ○ ○ 0_E		○	○	○	○	○	○
AUTOMATIC	INPUT	WITH LOCK	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$		○	○	○	○	○	○
		WITH NO LOCK	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$		○	○	○	○	○	○
		指定なし	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$		○	○	○	○	○	○
		OUTPUT	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	0_E 0_E 0_E 0_E			0_E 0_E 0_E 0_E		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$		0_E	0_E	0_E	0_E	0_E	0_E
	I-O	WITH LOCK	$R_ER_ER_E0_E$		$R_ER_ER_E-$	R_E R_E R_E 0_E			R_E R_E R_E −		$0_E0_E0_E0_E$		△△△0_E		○	E_R	○	○	E_R	○
		WITH NO LOCK	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		○ ○ ○ 0_E		○	○	○	○	○	○
		指定なし	$R_ER_ER_E0_E$		$R_ER_ER_E-$	R_E R_E R_E 0_E			R_E R_E R_E −		$0_E0_E0_E0_E$		△△△0_E		○	E_R	○	○	E_R	○
		EXTEND	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		○ ○ ○ 0_E		○	○	○	○	○	○
EXCLUSIVE		INPUT	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	0_E 0_E 0_E 0_E			0_E 0_E 0_E 0_E		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$							
		OUTPUT	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	0_E 0_E 0_E 0_E			0_E 0_E 0_E 0_E		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$							
		I-O	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	0_E 0_E 0_E 0_E			0_E 0_E 0_E 0_E		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$							
		EXTEND	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	0_E 0_E 0_E 0_E			0_E 0_E 0_E 0_E		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$							
指定なし	INPUT	WITH LOCK	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$		○	○	○	○	○	○
		WITH NO LOCK	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$		○	○	○	○	○	○
		指定なし	○ ○ ○ 0_E		○ ○ ○ −	○ ○ ○ 0_E			○ ○ ○ −		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$		○	○	○	○	○	○
		OUTPUT	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	0_E 0_E 0_E 0_E			0_E 0_E 0_E 0_E		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$							
	I-O	WITH LOCK	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	0_E 0_E 0_E 0_E			0_E 0_E 0_E 0_E		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$							
		WITH NO LOCK	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	0_E 0_E 0_E 0_E			0_E 0_E 0_E 0_E		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$							
		指定なし	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	0_E 0_E 0_E 0_E			0_E 0_E 0_E 0_E		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$							
		EXTEND	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	0_E 0_E 0_E 0_E			0_E 0_E 0_E 0_E		$0_E0_E0_E0_E$		$0_E0_E0_E0_E$							

（凡例）

○ ：指定できる　　E_R ：入出力エラー　　0_E：OPENエラー　R_E ：READエラー（レコード排他）
△ ：指定できる。ただし，順ファイル，相対ファイルではREADエラー
空白：該当しない
－： 同じレコードを操作することはあり得ない

注※

順ファイルおよびテキストファイルでは，ASSIGN 句に LOCK MODE MANUAL の指定はできません。順ファイルおよびテキストファイルを対象外とします。

7.2.5　ファイル共用を省略した場合の処理

LOCK MODE 句を省略した場合，そのファイルは，LOCK MODE IS EXCLUSIVE が指定されたとみなされ，ファイルレベルのファイル共用が有効となります。ファイルレベルのファイル共用については，「7.2.1　ファイルレベルのファイル共用」を参照してください。

7.2.6　ラージファイル入出力機能でのファイルの共用

ラージファイル入出力でのファイル共用に関しては，「6.10.2　ラージファイル入出力機能でのファイルの共用」を参照してください。

7.2.7　ネットワーク上のファイルを使用する場合の注意事項

同じネットワーク環境内にある異なる UNIX 間で実行する場合，レコードレベルのファイル共用は保証しません。

ファイル共用を使用する場合，ネットワークを構築するときに OS のロック機能が動作するように環境をセットアップしなければならない場合があります。ネットワークのセットアップについては，システムのマニュアルを参照してください。

（例）

NFS を利用してリモートファイルにアクセスする場合，OS のロック機能はカーネルのロック機構と NFS のロック監視デーモン（rpc.statd）および NFS のロック管理デーモン（rpc.lockd）の補助機構によって実現されています。そのため，NFS ロックデーモン（rpc.statd，rpc.lockd）を起動しなければなりません。NFS ロックデーモンを起動しないと，リモートファイルに対するファイル共用は使用できません。

8 プリンタへのアクセス

COBOL プログラムでは，XMAP3 を使用してプリンタへ出力できます。この章では，プリンタへのアクセスについて説明します。

8.1　プリンタアクセスの種類と概要

8.1.1　印刷モード

COBOL プログラムからプリンタにアクセスするには，次の方法があります。

XMAP3 による印刷（AIX(32)で有効）

XMAP3 と連携してプリンタ出力する印刷モードです。書式オーバレイや CHARACTER TYPE 句での行データ印刷制御など，XMAP3 の機能を使用して印刷できます。

8.1.2　プリンタアクセスで使用できるファイル編成

プリンタアクセスで使用できるファイル編成は，順編成ファイルまたはテキスト編成ファイルです。ただし，印刷モードによっては，使用できるファイル編成に制限があります。

各印刷モードで使用できるファイル編成について，次に示します。

印刷モード	順編成ファイル	テキスト編成ファイル
XMAP3 による印刷	○	×

（凡例）

○：使用できる

×：使用できない

順編成については，マニュアル「COBOL2002　言語　標準仕様編」「5.1.7(1)　順編成」を，テキスト編成については，マニュアル「COBOL2002　言語　拡張仕様編」「2.1　テキスト編成」を，それぞれ参照してください。

8.2　プリンタアクセスの共通規則

印刷モードで共通な規則について，次に示します。

8.2.1　ファイル割り当て

プリンタ出力をするファイルは，環境部入出力節で ORGANIZATION 句に順編成またはテキスト編成のどちらかを指定します。

環境入出力節でファイル編成を指定したあと，定数指定，環境変数指定，データ名指定のどれかの方法で，ASSIGN 句に出力先の物理ファイル名を割り当てます。

割り当てる物理ファイル名は，印刷モードごとに異なります。詳細は，「8.4　XMAP3 による印刷（AIX(32)で有効)」を参照してください。また，ファイル割り当ての詳細については，「6.2　ファイル割り当ての共通規則」を参照してください。

定数指定，環境変数指定，データ名指定の記述例を，それぞれ次に示します。

(1)　定数指定

```
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FILE-1 ASSIGN TO '物理ファイル名'.
           :
```

(2)　環境変数指定

```
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FILE-1 ASSIGN TO PRT.
           :
```

注

実行時に，次の環境変数が指定されているものとします。

```
CBL_PRT=物理ファイル名
```

(3)　データ名指定

```
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FILE-2 ASSIGN TO FILE-NAME.
           :
```

```
       WORKING-STORAGE SECTION.
       01 FILE-NAME PIC X(40).
       PROCEDURE DIVISION.
           MOVE ’物理ファイル名’ TO FILE-NAME.
           :
```

8.2.2　入出力手続き文と動作

プリンタへ行データを出力するために使用する文を次に示します。

表 8-1　入出力手続き文と動作

使用する文	XMAP3 による印刷
OPEN 文	プリンタ出力処理の準備をする。OUTPUT モードで実行する。
WRITE 文	1 行のデータを行制御をして印刷用ファイルに書き出す。
CLOSE 文	プリンタ出力処理を終了する。

8.2.3　行制御出力

行制御出力とは，データ部のファイル記述項に LINAGE 句を指定したり，WRITE 文に ADVANCING／POSITIONING 指定または CHARACTER TYPE 句を指定したりして，行データを制御する出力処理のことです。行制御出力は，報告書作成機能を含みます。

CHARACTER TYPE 句については，マニュアル「COBOL2002　言語　拡張仕様編」「14.2.1(1) CHARACTER TYPE 句」を，WRITE 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.55　WRITE 文」を，それぞれ参照してください。

(1)　順編成ファイル，テキスト編成ファイルの行制御

順編成ファイル，テキスト編成ファイルに対して行制御が実行されるのは，次の場合です。

- データ部のファイル記述項に LINAGE 句を指定した場合。
- WRITE 文に ADVANCING 指定，POSITIONING 指定をした場合。

8.2.4　実行時環境変数で行制御を操作する方法

環境変数 CBLD_ファイル名に，次の環境変数を指定すると，WRITE 文の行制御指定を操作できます。

- SAMAADV／NOSAMAADV

  SAMAADV は，ADVANCING 指定を記述しない WRITE 文で自動的に AFTER ADVANCING 1 LINE が仮定されます。NOSAMAADV は，仮定されません。標準値は，NOSAMAADV です。

8.3　入出力エラー処理

入出力エラーの詳細は，「6.3　入出力エラー処理」を参照してください。

8.4 XMAP3 による印刷（AIX(32)で有効）

XMAP3 を使用すると，書式と行データを重ね合わせる印刷（書式オーバレイ印刷）や，印刷制御付きの行データの印刷ができます。

XMAP3 による印刷機能について説明します。

8.4.1 前提条件とプログラムの作成方法

- XMAP3 による印刷をするには，あらかじめ帳票の定義が必要です。帳票を定義するには，XMAP3 を使用します。
- XMAP3 による印刷を使用するプログラムは，コンパイル時に-XMAP,LinePrint オプションの指定が必要です。

8.4.2 プリンタへの出力と割り当て方法

(1) 印刷サービス名称の指定

印刷サービス名称とは，XMAP3 で使用する端末やプリンタの識別名称です。

COBOL プログラム実行時，印刷サービス名称は，SELECT 句で指定したファイルの ASSIGN 句で指定した外部装置名に割り当てられます。印刷サービス名称は，環境変数 CBLX_外部装置名で指定します。印刷サービス名称の指定方法を次に示します。

形式

```
SELECT ファイル名 ASSIGN TO 外部装置名
```

環境変数

```
CBLX_外部装置名=印刷サービス名称
```

注意事項

- ASSIGN 句で定数指定またはデータ名指定をした場合，または外部装置名を環境変数 CBL_外部装置名で指定した場合，指定した文字列は，印刷サービス名称ではなく物理ファイル名として扱われるので注意してください。
- ASSIGN 句で指定した外部装置名に対して，印刷サービス名称の指定（CBLX_外部装置名）と物理ファイル名の指定（CBL_外部装置名）を同時に指定した場合，印刷サービス名称の指定（CBLX_外部装置名）が有効となります。これらの環境変数を同時に指定するときには，注意が必要です。
- 環境変数 CBLX_外部装置名は，OPEN 文を実行するごとに環境変数の値が参照されます。

(2)　プリンタ出力の識別

COBOL プログラムの記述，-XMAP,LinePrint オプションの指定の有無と，プリンタ，通常ファイルへの出力の識別を次に示します。

表 8-2　プリンタ，通常ファイルへの出力の識別

-XMAP,LinePrint オプションの指定	COBOL プログラムの記述		外部装置名（環境変数）		ASSIGN 定数 または ASSIGN データ名
	APPLY FORMS-OVERLAY 句の指定	CHARACTERTYPE 句の指定	CBLX_xxx (印刷サービス名)	CBL_xxx (物理ファイル名)	
			プリンタ	ファイル	ファイル
あり	あり	あり	○※1	×※2	×※3
		なし	○※1	×※2	×※3
あり	なし	あり	○※1	×※2	×※3
		なし	○※1	○	○
なし	－	－	×	○	○

（凡例）

○：出力できる

×：出力できない

－：指定しても意味を持たない（覚え書きとなる）

注※1

OUTPUT 指定以外で OPEN 文を実行すると，実行時エラーとなります。

注※2

実行時にエラーメッセージが出力されます。

注※3

コンパイル時にエラーメッセージが出力されます。

出力先がプリンタの場合，コンパイルリスト（情報リスト）のファイル情報の個所には，プリンタ出力であることが表示されます。情報リストについては，「付録 D　コンパイルリスト」を参照してください。

8.4.3　出力形態とレコード形式

XMAP3 による印刷で出力するレコード形式は，XMAP3 に依存します。

8.4.4　入出力手続き文と動作

XMAP3 による印刷をする場合，印刷用ファイルへデータを出力するために，手続き部で次の文を使用します。

- OPEN 文
  印刷用ファイルを使用するための準備をします。OUTPUT モードで実行します。
- WRITE 文
  レコードを印刷用ファイルに書き出します。明示的または間接的に ADVANCING 指定をします。間接的にとは，同一ファイルに対するほかの WRITE 文に ADVANCING 指定がされている場合をいいます。
- CLOSE 文
  ファイル処理を終了します。
  印刷用ファイル中に残っている行データは，すべてプリンタに出力されます。

(1) 行データの印刷制御付きの指定

印刷制御付きの行データを出力するには，WRITE 文で出力するデータ項目に CHARACTER TYPE 句を指定します。CHARACTER TYPE 句の書き方や規則については，マニュアル「COBOL2002 言語 拡張仕様編」「14.2.1 データ記述項（書式印刷機能）」を参照してください。

また，-XMAP,LinePrint オプションを指定したプログラムに POINT-*l*，FORMAT-*n*，INTERVAL-*i* を指定した場合，各項目（*l*，*n*，*i*）に指定できる値の範囲と意味については，XMAP3 での定義に従います。詳細は，マニュアル「画面・帳票サポートシステム XMAP3 Server」を参照してください。

8.4.5 書式オーバレイの出力方法

書式付きで印刷をするときには，書式オーバレイイメージ名称を指定します。これには，次の二つの方法があります。

(1) プログラムによる書式オーバレイイメージの指定

プログラム中の入出力管理記述項（I-O-CONTROL.）に，APPLY FORMS-OVERLAY 句を指定すると，書式オーバレイイメージを重ねて印刷できます。

APPLY FORMS-OVERLAY 句については，マニュアル「COBOL2002 言語 拡張仕様編」「14.1.1(1) APPLY FORMS-OVERLAY 句」を参照してください。

（例）

```
        :
I-O-CONTROL.
    APPLY FORMS-OVERLAY TO FOV-NAME ON
    DAILY-FILE.
    :
WORKING-STORAGE SECTION.
77 FOV-NAME PIC X(8) VALUE 'FORMOVL1'.
    :
```

(2)　環境変数による書式オーバレイイメージの指定

プログラム中に APPLY FORMS-OVERLAY 句を指定しないときは，XMAP3 の環境変数の指定によって，COBOL プログラムに関係なく書式オーバレイイメージを重ねて印刷できます。

プログラム中に APPLY FORMS-OVERLAY 句があるとき，XMAP3 の環境変数によって書式名を有効にする場合は，書式名格納エリアに必ず NULL を格納してください。XMAP3 の環境変数については，マニュアル「画面・帳票サポートシステム　XMAP3 Server」を参照してください。

8.4.6　XMAP3 による印刷モードの注意事項

- 重ね打ちの制限

  行データの重ね打ちはできません。

- 制御文字の取り扱い

  制御文字は，次のように取り扱われます。

制御文字	プリンタ上での扱い
X'0A'（改行）	改行
X'0C'（改ページ）	改ページ
X'0D'（復帰）	印字されない
X'00'～X'1F'，X'7F' （ただし X'0A'，X'0C'，X'0D'は除く）	印字されない

- 可変長レコード形式を使用する場合の注意事項
  - WRITE 文で書き出すレコードは，可変長レコード形式として定義していても，ファイルの先頭のヘッダレコード（128 バイト）と各レコード先頭のレコード長領域（4 バイト）が付加されません。
  - WRITE 文で書き出すレコードは，レコード定義に CHARACTER TYPE 句を指定している場合，基本項目または集団項目の下位項目（基本項目）単位で出力します。そのため，RECORD 句の VARYING 指定の可変長レコードの場合，DEPENDING ON で指定するレコード長は基本項目の境界でなければなりません。レコード長が基本項目の境界でない場合，基本項目の相対位置がレコード長を超える基本項目は出力されません。
- リンクの指定

  XMAP3 による印刷モードを使用する COBOL プログラムをリンクするとき，次の形式に従って cc コマンドまたは ccbl2002 コマンドを実行する必要があります。

  XMAP3 を使用する場合の例を次に示します。

  形式（cc コマンド）

```
cc オブジェクトファイル名 … -lxmovl※ -lxpw※
```

  形式（ccbl2002 コマンド）

```
ccbl2002 -XMAP,LinePrint ファイル名 … -lxmovl※ -lxpw※
```

注※

-lxmovl，および-lxpw は，XMAP3 が組み込まれているライブラリです。

- 書式オーバレイなしの行データだけの印刷はできません。行データだけを印刷するときでも，けい線や固定文字列などを何も指定していない空の書式を指定する必要があります。

9 報告書作成機能

この章では，報告書作成機能を使った報告書の作成方法について説明します。

9.1　報告書作成機能の概要

一般に，事務計算で作成する報告書には，次のような幾つかの特徴があります。

報告書の一般的な特徴

- 報告書は，計算機のラインプリンタのミシン目ごとに区切られたページの集まりから成る。
- 報告書には，全体の表紙と裏表紙が付く。
- 各ページの上部や下部にはページの見出しが付く。
- 報告書の本文は，文章や図形で何か出来事を記述したものではなく，同じ形式のデータ（レコード）が縦に並んだものである。
- データは，データ中の幾つかの項目（キーデータ項目，制御用データ項目）に従って，昇順または降順に順序正しく並んでいる。
- 制御用データ項目の値が変化すると制御の切れ目（コントロールブレーク）となり，小計，中計，大計などの報告実行を作成する。

このような形式と内容を持った報告書を，COBOL の通常の句や文を組み合わせて作成する場合，手間が掛かります。このため COBOL には，通常の入出力機能とは別に，報告書作成機能が用意されています。

報告書作成機能は，定形的な報告書作成の手続きを，手続き部の文を組み合わせて指定するのではなく，データ部の記述項で指定します。

なお，このシステムでは，報告書作成機能（INITIATE／GENERATE／TERMINATE 文）で作成した報告書は，AFTER ADVANCING 指定の WRITE 文で順ファイル（報告書ファイル）にレコードとして出力されます。

なお，報告書機能の文法規則については，マニュアル「COBOL2002　言語　標準仕様編」「13.　報告書作成機能」を参照してください。

9.2　ファイル割り当ての共通規則

報告書を出力するファイルは，環境部の入出力節で，ORGANIZATION 句に順編成を指定します。

また，定数指定，環境変数指定，データ名指定のどれかの方法で，ASSIGN 句に出力先の物理ファイル名を割り当てます。物理ファイル名の割り当て方法の詳細は，「6.2　ファイル割り当ての共通規則」を参照してください。

9.3　入出力エラー処理

9.3.1　USE 手続き

手続きが通常の順序で実行されるのではなく，利用者が直接検出できない条件が発生したときに実行しなければならない手続きを，USE 文で指定します。

USE 文は，手続きの実行の条件を指定するための翻訳指示文です。そのため，USE 文自身が実行されることはありません。

USE 文で指定する手続きは，手続き部の宣言部分で指定します。手続き部の構成の規則については，マニュアル「COBOL2002　言語　標準仕様編」「10.　手続き部（PROCEDURE DIVISION)」を参照してください。

報告書作成機能で使用する USE 文では，報告書作成手続きを指定する BEFORE REPORTING 指定，および AFTER STANDARD EXCEPTION PROCEDURE 指定を使用できます。報告書作成機能を使用したプログラム中での USE 文の指定例を，次に示します。

```
   :
01 FOOT1 TYPE IS CONTROL FOOTING SHITEN.
  02 LINE NUMBER PLUS 2.
   :
PROCEDURE DIVISION.
DECLARATIVES.
CNT SECTION.
    USE BEFORE REPORTING FOOT1.
C.
  ADD 1 TO CNT-1.
END DECLARATIVES.
   :
```

このようにコーディングすると，FOOT1 を印刷する前に ADD 1 TO CNT-1 が実行されます。

9.4　ファイルの作成と割り当て方法

報告書ファイルの作成と割り当て方法は，順編成ファイルの場合と同じです。詳細は，「6.4.1　ファイルの作成と割り当て方法」を参照してください。

9.5　ファイル編成とレコード形式

報告書ファイルのファイル編成とレコード形式は，順編成ファイルの場合と同じです。

9.6　報告書ファイルの出力

報告書作成機能で作成した報告書は，AFTER ADVANCING 指定の WRITE 文で順ファイル（報告書ファイル）にレコードとして出力されます。

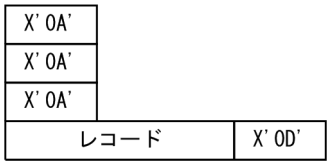
報告書ファイルの出力形式は，レコード形式と報告書記述項での CODE 句の有無によって異なります。報告書ファイルの出力形式と出力例を次に示します。出力例の X'0A'は改行，X'0D'は復帰，X'0C'は改ページを示します。

固定長または可変長で CODE 句の指定がない場合

順編成ファイルに対する，AFTER ADVANCING 指定の WRITE 文の形式で出力されます。

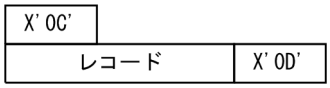
（例 1）

WRITE REC AFTER ADVANCING 3 で出力した場合



（例 2）

WRITE REC AFTER ADVANCING PAGE で出力した場合



固定長で CODE 句の指定がある場合

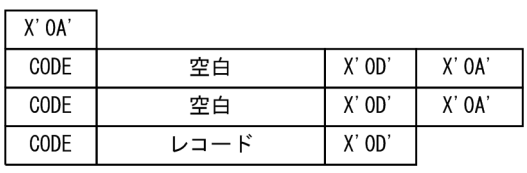
順編成ファイルに対する，AFTER ADVANCING 指定の WRITE 文の形式で出力されます。

利用者レコードの前後に付けられる制御コードについては，先頭の制御コード以外は，レコード長分の空白（CODE 句で指定したコードが先頭に付けられる）と復帰コード，改行コードを付けた形式で出力されます。

ただし，-IgnoreLCC オプションの指定があるときは，レコード長から 1 を引いた分の空白と復帰コード，改行コードを付けた形式で出力されます。

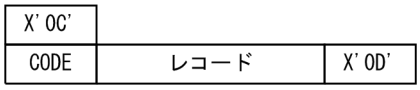
（例 1）

WRITE REC AFTER ADVANCING 3 で出力した場合



（例 2）

WRITE REC AFTER ADVANCING PAGE で出力した場合



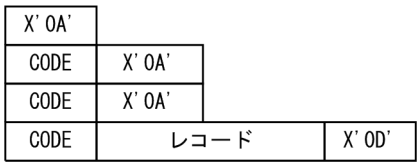
可変長で CODE 句の指定がある場合

順編成ファイルに対する，AFTER ADVANCING 指定の WRITE 文の形式で出力されます。

利用者レコードの前後に付けられる制御コードについては，先頭の制御コード以外は，CODE 句で指定したコードを付けた形式で出力されます。

（例 1）

WRITE REC AFTER ADVANCING 3 で生成した場合



（例 2）

WRITE REC AFTER ADVANCING PAGE で出力した場合

固定長で CODE 句の指定がある場合と同じです。

9.7　報告書ファイルの入力

報告書ファイルを読んでレコードを入力するときの形式は，レコード形式と報告書記述項で CODE 句を指定しているかどうかによって次のように異なります。

固定長で CODE 句の指定がない場合

順編成固定長の形式では読めません。

固定長で CODE 句の指定がある場合

順編成固定長で読むためには，利用者レコード長に制御コード分の 2 を加えた形式で読み込む必要があります。

ただし，-IgnoreLCC オプションの指定があるときは，先頭 1 バイトが出力されないため，レコード長に 1 を加えた形式で読み込みます。

（例）-IgnoreLCC オプション指定なしの場合

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       SELECT FILE1 ASSIGN SYS001.
       DATA DIVISION.
       FILE SECTION.
       FD FILE1.
       01 REC-1.
        02 R-DATA1 PIC X(1).              …1.
        02 R-DATA2 PIC X(1) or PIC X(2). …2.
        02 R-DATA3 PIC X(n).              …3.
        02 R-DATA4 PIC X(1).              …4.
       WORKING-STORAGE SECTION.
             :
       PROCEDURE DIVISION.
             :
           OPEN INPUT FILE1.
             :
           READ FILE1.
             :
           CLOSE FILE1.
           STOP RUN.
```

1. 行制御コード X'0C'（改ページ）または X'0A'（改行）
2. ファイル作成時に CODE 句で指定した文字の長さ分（1 バイトまたは 2 バイト）を確保する。
3. ファイル作成時に RD 句で指定したレコード長（2.の長さは含まない）
4. 行制御コード X'0D'（復帰）

可変長の場合

可変長の場合，CODE 句の指定があるかどうかに関係なく可変長ファイルとして作成されないため，順編成可変長では読めません。また，順編成固定長でも読めません。

10 ACCEPT／DISPLAY／STOP 文による入出力

この章では，ACCEPT 文，DISPLAY 文，および STOP 文を使用した入出力機能について説明します。

10.1　ACCEPT／DISPLAY／STOP 文による入出力の種類と概要

COBOL2002 では，ACCEPT 文，DISPLAY 文，STOP 文を使用して，次の入出力操作を実行できます。

少量入出力

標準入出力ファイル（stdin，stdout，stderr）に対して，データを入出力できます。

詳細は，「10.2　少量入出力」を参照してください。

日付や時刻の取得

日付や時刻の情報を取得できます。

詳細は，「10.2.3　日付や時刻を取得する ACCEPT 文」を参照してください。

コマンド行へのアクセス

コマンド名称，コマンド行の引数の個数，および引数の値を取得できます。

詳細は，「10.3　コマンド行へのアクセス」を参照してください。

環境変数へのアクセス

環境変数の値を取得したり，環境変数を設定したりできます。

詳細は，「10.4　環境変数へのアクセス」を参照してください。

COBOL ログファイルへの出力

システム入出力関数レベルを指定して，DISPLAY 文での書き込み失敗時に出力されたデータを COBOL 独自のログファイルとして出力できます。

詳細は，「10.5　COBOL ログファイル出力機能」を参照してください。

なお，ACCEPT 文，DISPLAY 文，および STOP 文の文法規則については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.1　ACCEPT 文」，「COBOL2002　言語　標準仕様編」「10.8.12　DISPLAY 文」，「COBOL2002　言語　標準仕様編」「10.8.46　STOP 文」をそれぞれ参照してください。

10.2　少量入出力

ここでは，ACCEPT 文，DISPLAY 文，および STOP 文を使用して，標準入出力ファイルにデータを入出力する方法について説明します。

10.2.1　入出力の対象とするファイルの割り当て方法

ACCEPT 文および DISPLAY 文で入出力の対象とするファイルは，環境変数によって割り当てます。

(1)　ACCEPT 文

ACCEPT 文の FROM 指定と，環境変数の値によって，データの入力元となるファイルが決まります。FROM 指定，環境変数の値と，データの入力元との関係を，次に示します。

FROM 句の指定	環境変数の指定	データの入力元
SYSIN 　または SYSIPT	CBL_SYSIN=stdin	標準入力
	CBL_SYSIN=stdout	（エラーとなる）
	CBL_SYSIN=stderr	（エラーとなる）
	CBL_SYSIN=syslog	ファイル名「syslog」
	CBL_SYSIN が上記以外の場合	環境変数 CBL_SYSIN で指定した名称のファイル
	CBL_SYSIN の指定がない場合	標準入力
CONSOLE		標準入力
SYSSTD	CBL_SYSSTD=stdin	標準入力
	CBL_SYSSTD=stdout	（エラーとなる）
	CBL_SYSSTD=stderr	（エラーとなる）
	CBL_SYSSTD=syslog	ファイル名「syslog」
	CBL_SYSSTD が上記以外の場合	環境変数 CBL_SYSSTD で指定した名称のファイル
	CBL_SYSSTD の指定がない場合	標準入力

注

標準入力（stdin）を指定する場合は，英小文字で指定してください。「STDIN」のように英大文字で指定した場合，物理ファイル名として扱われます。

(2)　DISPLAY 文

DISPLAY 文の UPON 指定と，環境変数の値によって，データの出力先となるファイルが決まります。UPON 指定，環境変数の値と，データの出力先との関係を，次に示します。

UPON 句の指定	環境変数の指定	データの出力先
SYSPUNCH 　または SYSPCH	CBL_SYSPUNCH=stdin〔+〕	（エラーとなる）
	CBL_SYSPUNCH=stdout〔+〕	標準出力※
	CBL_SYSPUNCH=stderr〔+〕	標準エラー出力※
	CBL_SYSPUNCH=syslog〔+〕	ファイル名「syslog」
	CBL_SYSPUNCH= 　上記以外のファイル名〔+〕	環境変数 CBL_SYSPUNCH で指定した名称のファイル
	CBL_SYSPUNCH 指定なし	標準出力
SYSOUT 　または SYSLST	CBL_SYSOUT=stdin〔+〕	（エラーとなる）
	CBL_SYSOUT=stdout〔+〕	標準出力※
	CBL_SYSOUT=stderr〔+〕	標準エラー出力※
	CBL_SYSOUT=syslog〔+〕	ファイル名「syslog」
	CBL_SYSOUT= 　上記以外のファイル名〔+〕	環境変数 CBL_SYSOUT で指定した名称のファイル
	CBL_SYSOUT 指定なし	標準出力
CONSOLE		標準出力

注

標準出力（stdout），標準エラー出力（stderr）を指定する場合は，英小文字で指定してください。「STDIN」のように英大文字で指定した場合，物理ファイル名として扱われます。

また，環境変数 CBL_SYSPUNCH，CBL_SYSOUT の指定で，ファイル名の末尾に「+」を付けると，追加モードでデータを出力できます。

```
CBL_SYSPUNCH=ファイル名+
CBL_SYSOUT=ファイル名+
```

注※

追加モードは無視されます。

10.2.2　外部からのデータを入力する ACCEPT 文

ACCEPT 文で外部からのデータを入力する場合，次の 2 種類の方法で受け取り側作用対象にデータを設定できます。

- 標準転記

  入力したデータをそのまま受け取り側作用対象に設定します。

- 数値へのデータ変換を伴う転記

  受け取り側作用対象が数字項目の場合，入力された英数字データを数字データへ変換して，受け取り側作用対象に設定します。

(1)　標準転記による ACCEPT 文

少量入出力に使用する ACCEPT 文について説明します。なお，ACCEPT 文の言語仕様の詳細については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.1　ACCEPT 文」を参照してください。

(a)　形式

```
ACCEPT 一意名
   〔FROM  {SYSIN | SYSIPT | CONSOLE | SYSSTD | 呼び名} 〕
```

(b)　一般規則

- FROM 指定を省略すると SYSIN が仮定されます。
- SYSIPT は，SYSIN と同様とみなされます。
- 入出力の対象とするファイルの規則については，「10.2.1　入出力の対象とするファイルの割り当て方法」の「(1)　ACCEPT 文」を参照してください。
- 環境変数 CBL_SYSIN または環境変数 CBL_SYSSTD が指定されていない場合の入力先は，標準入力となります。
- 一意名の領域には，入力データの最後の改行文字は格納されません。
- 入出力エラーが発生した場合は，エラーメッセージが出力され，処理が終了します。

(c)　SYSIN，SYSIPT，CONSOLE を指定した場合の規則

データ入力の単位

改行文字までが 1 回だけ入力されます。

転記規則

- 入力データが一意名の領域より長い場合，一意名の長さで区切られ，残りは切り捨てられます。
- 入力データが一意名の領域より短い場合，一意名の残りの領域には空白が埋められます。
- 入力データは一意名の領域の左端から順に転送されます。一意名が日本語項目なら日本語空白文字（シフト JIS のときは X'8140'，日本語 EUC のときは X'A1A1'）が，日本語項目以外なら標準コードの空白文字（X'20'）が埋められます。日本語項目で入力が語境界（2 バイトで 1 語）でなければエラーメッセージを出力して終了します。

(d)　SYSSTD を指定した場合の規則

データ入力の単位

1 レコードは常に 80 バイトであり，改行文字がレコードの区切りとなります。1 レコードが 80 バイトを超える場合は，先頭から 80 バイトが転送され，残りのデータは改行文字まで切り捨てられます。1 レコードが 80 バイト未満の場合は，残った領域に空白文字が埋められたレコードとして扱われます。

転記規則

- 一意名の長さが 80 バイトの場合，データは一度だけ転送されます。
- 一意名の長さが 80 バイトより短い場合は，データは一度だけ転送され，残りは切り捨てられます。
- 一意名の長さが 80 バイトより長い場合は，一意名の長さを満たすまで 80 バイト単位で転送が繰り返されます。一意名の長さが 80 の倍数でないときは，80 バイト単位での転送を繰り返したあと，最後に残った 80 バイト未満のデータ領域に入力データが格納されます。ただし，最後に残った 80 バイト未満のデータ領域に入りきらないデータがある場合，残ったバイト数から右側は，切り捨てられます。
- 入力データは一意名の領域の左端から順に転送されます。一意名が日本語項目なら日本語空白文字（シフト JIS のときは X'8140'，日本語 EUC のときは X'A1A1'）が，日本語項目以外なら標準コードの空白文字（X'20'）が埋められます。日本語項目で入力が語境界（2 バイトで 1 語）でなければエラーメッセージを出力して終了します。

(2) 数値へのデータ変換を伴う ACCEPT 文

-NumAccept オプションが指定されている場合，ACCEPT 文は数字項目，数字編集項目，外部浮動小数点数字項目，内部浮動小数点数字項目で定義されている受け取り側作用対象に対して，入力されたデータを受け取り側作用対象の属性に合わせたデータに変換して，設定します。

(a) データ変換の規則

データ変換の規則を，次に示します。

表 10–1 -NumAccept オプション指定時の ACCEPT 文の入力規則

一意名の項目		受け取り領域長	入力できる文字	備考
符号なし数字項目（S なし）	V なし	けた数	0～9	
	V あり	けた数+1	0～9 .	数字はけた数以内 小数点位置を合わせる
符号あり数字項目（S あり）	V なし	けた数+1	0～9 + -	数字はけた数以内
	V あり	けた数+2	0～9 + - .	数字はけた数以内 小数点位置を合わせる
数字編集項目		一意名長※	0～9 + - CR DB	数字は有効数字長以内 小数点位置を合わせる
外部浮動小数点数字項目	V .なし	22	0～9 E + -	数字は有効けた数以内
	V .あり	22	0～9 E + - .	
内部浮動小数点数字項目	単精度	22	0～9 E + - .	
	倍精度	22	0～9 E + - .	

注※

ただし，左端に P を書いた数字編集項目の場合，左端に V を想定するため一意名長+1 となります。

(b)　転記規則

転記規則

- 入力データが一意名の領域より長い場合，「表 10–1　-NumAccept オプション指定時の ACCEPT 文の入力規則」に示す受け取り領域長で区切られ，残りは切り捨てられます。
- 入力データが一意名の領域より短い場合，入力した長さだけが有効となり，一意名の残りの領域にはゼロが埋められます。
- 入力データは，一意名が数字項目の場合は小数点の位置に合わせて，数字編集項目の場合は編集文字に合わせて，外部浮動小数点数字項目，内部浮動小数点数字項目の場合は小数点位置，E の位置に合わせて，それぞれ右詰めで転記されます。

(c)　入力データのチェック

次の規則に従って，入力データがチェックされます。

不当文字を入力した場合

各項目に「表 10–1　-NumAccept オプション指定時の ACCEPT 文の入力規則」の入力できる文字以外の文字が入力された場合，不当文字とみなされます。入力データ中に使用できる文字と不当文字とが混在している場合，不当文字は，一意名へ転記されません。

符号，小数点，E を複数入力した場合

左端から検索し，最初に出現したものを有効とします。それ以降に出現したものは不当文字として扱います。

すべて不当文字の文字列を入力するか，または何も入力しなかった場合

一意名にすべてゼロを設定します。

一意名が外部浮動小数点数字項目，内部浮動小数点数字項目の場合

入力形式は小数点を含む外部 10 進形式，外部浮動小数点数字形式のどちらでもかまいません。ただし，外部浮動小数点数字形式で入力した場合，指数部のけた数は 2 けたまで有効となります。

10.2.3　日付や時刻を取得する ACCEPT 文

ACCEPT 文で，FROM 指定に DATE，DAY，DAY-OF-WEEK，および TIME を指定すると，それぞれの指定に応じた日付／時刻の情報を取得できます。

(1)　ACCEPT 文で取得できる日付／時刻のデータ

ACCEPT 文で取得できる日付／時刻のデータを，次に示します。

FROM 指定	内容	形式
DATE	yymmdd yy：西暦年号の下 2 けた mm：01～12（月） dd：01～31（日）	符号のない 6 けたの数字項目
DAY	yyddd yy：西暦年号の下 2 けた ddd：001～366（通年日）	符号のない 5 けたの数字項目
DAY-OF-WEEK	w w：1～7（曜日） 1：月曜日 2：火曜日 ： 7：日曜日	符号のない 1 けたの数字項目
TIME	hhmmsstt hh：00～23（時） mm：00～59（分） ss：00～59（秒） tt：00（常に 00）	符号のない 8 けたの数字項目

(2) ACCEPT 文で取得する日付の変更

通常，ACCEPT 文では，現在の日付を取得しますが，環境変数を指定すると任意の日付を取得できます。

ACCEPT 文で取得する日付を変更する方法を，次に示します。

(a) 西暦日付の変更（CBLDATE）

COBOL プログラムが取得する西暦表記の年月日を変更するには，環境変数 CBLDATE を指定します。

形式

```
CBLDATE=yyyymmdd
```

yyyy
　西暦の年を 4 けたで指定します。

mm
　月を 2 けたで指定します。

dd
　日を 2 けたで指定します。

規則

- 環境変数 CBLDATE を指定して取得する日付を変更できるのは，次の文および関数です。

環境変数 CBLDATE での日付指定が有効となる COBOL の文／関数

・ACCEPT 文 DATE 指定

・CURRENT-DATE 関数

・MOVE 文（日付と時刻用） CURRENT-DATE 指定

この環境変数は，上記の文のどれかを初めて実行したときに指定されている値を有効とします。

- 西暦年 yyyy のうち，DATE 指定の ACCEPT 文と MOVE 文の CURRENT-DATE 指定では下 2 けたが使用され，CURRENT-DATE 関数では 4 けたが使用されます。
- 上記以外の形式で値を指定した場合，実行時に警告のメッセージが出力されます。この場合，環境変数 CBLDATE を指定していない場合と同じように，DATE 指定の ACCEPT 文にはシステムの日付が返されます。

指定例

COBOL プログラムが取得する西暦日付を，2010 年 10 月 12 日に変更する場合の指定例を，次に示します。

```
CBLDATE=20101012
```

(b) 通算日付の変更（CBLDAY）

COBOL プログラムの ACCEPT 文 DAY 指定が取得する西暦年，および通年日を変更するには，環境変数 CBLDAY を指定します。

形式

```
CBLDAY=yyyyddd
```

yyyy

西暦の年を 4 けたで指定します。

ddd

通年日（その年が始まってからの通算の日付）を 3 けたで指定します。

規則

- この環境変数を指定すると，初めて DAY 指定の ACCEPT 文が実行されたとき，yyddd（yy は西暦の下 2 けた）の部分が取得されます。
- 上記以外の形式で値を指定した場合，実行時に警告のメッセージが出力されます。この場合，環境変数 CBLDAY を指定していない場合と同じように，DAY 指定の ACCEPT 文にはシステムの日付が返されます。

指定例

ACCEPT 文 DAY 指定が取得する通算日付を，2010 年の 56 日に変更する場合の指定例を，次に示します。

```
CBLDAY=2010056
```

10.2.4　DISPLAY 文によるデータの出力

少量入出力に使用する DISPLAY 文について説明します。なお，DISPLAY 文の言語仕様の詳細については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.12　DISPLAY 文」を参照してください。

(1)　形式

```
DISPLAY　{一意名 | 定数}
   〔UPON　{SYSPUNCH | SYSOUT | SYSLST | SYSPCH | CONSOLE | 呼び名} 〕
   〔WITH NO ADVANCING〕
   〔IN DATA DUMP〕
```

(2)　一般規則

- UPON 指定を省略すると SYSOUT が仮定されます。
- SYSLST は，SYSOUT と同様とみなされます。
- 入出力の対象とするファイルの規則については，「10.2.1　入出力の対象とするファイルの割り当て方法」の「(2)　DISPLAY 文」を参照してください。
- 指定した一意名または定数全部が出力されたあと，改行文字が最後に出力されます。ただし，WITH NO ADVANCING 指定がある場合は，最後の改行文字が出力されません。
- IN DATA DUMP 指定がある場合は，一意名の格納値はバイト単位に 16 進数に変換されて出力されます。一意名の格納値の表示形式については，「36.10　DISPLAY 文による一意名の 16 進ダンプ表示」を参照してください。
- 入出力エラーが発生した場合は，エラーメッセージが出力されてから処理が終了します。

10.2.5　システム入出力関数レベルの指定

COBOL 実行時ライブラリは，高水準システム入出力関数によって DISPLAY 文のデータを出力します。

そのため，標準出力または標準エラー出力を，高水準システム入出力関数（fwrite，fflush）の使用が制限される出力先に切り替えた環境（システムの PIPE,FIFO 機能など）では，DISPLAY 文でエラーが発生し，プログラムが異常終了することがあります。

この場合，環境変数 CBLSTDIOLVL を指定することによって，低水準システム入出力関数（write）を使用したデータ出力に変更できます。低水準システム入出力関数を使用したデータ出力に変更すると，DISPLAY 文でエラーが発生してもプログラムの実行は継続され，DISPLAY 文の出力データは COBOL ログファイルに出力されます。なお，標準出力または標準エラー出力を変更しない場合は，環境変数 CBLSTDIOLVL を指定する必要はありません。

- 環境変数 CBLSTDIOLVL の指定方法

DISPLAY 文によって，データを標準出力（stdout）または標準エラー出力（stderr）に出力する場合，COBOL 実行時ライブラリが使用するシステム入出力関数のレベルを指定します。環境変数 CBLSTDIOLVL の指定方法については「35.3　プログラムの実行環境の設定」を参照してください。

- エラー発生時の動作

  環境変数 CBLSTDIOLVL を指定して低水準システム入出力関数を指定したデータ出力に変更しても，高負荷で書き込みが連続して発生した場合など，バッファの状態によって，DISPLAY 文の実行中にエラーが発生することがあります。この場合，エラーが発生しても実行を継続します。

  エラーが発生した場合の動作を次に示します。

  - エラーメッセージの出力

    W レベルのエラーメッセージを出力して，実行を継続します。ただし，メッセージの出力中にエラーが発生した場合は，メッセージが出力されないことがあります。この場合，環境変数 CBL_SYSERR にファイル名を指定することによって，メッセージをファイルに出力できます。

  - DISPLAY 文のエラー出力

    エラー発生時の DISPLAY 文の出力データを COBOL ログファイルに出力します。COBOL ログファイルの詳細は，「10.5　COBOL ログファイル出力機能」を参照してください。

    また，COBOL ログファイル出力中にエラーが発生した場合は，標準出力に DISPLAY 文のデータを出力します。

10.2.6　STOP 文

少量入出力に使用する STOP 文について説明します。なお，STOP 文の言語仕様の詳細については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.46　STOP 文」を参照してください。

(1)　形式

```
STOP 定数1
```

(2)　一般規則

STOP 定数文が実行されると，次の入力要求メッセージが標準エラー出力（stderr）に出力されます（「_」はカーソルを示します）。

（STOP 定数文の出力結果）

```
KCCC2003R-I
定数1
_
```

ただし，環境変数 CBL_STOPNOADV に YES を指定した場合は，定数 1 だけが出力され，メッセージ ID と定数 1 の直後の改行文字は出力されません。

形式

```
CBL_STOPNOADV=YES
```

(環境変数 CBL_STOPNOADV に YES を指定した場合の STOP 定数文の出力結果)

```
定数1_
```

10.3　コマンド行へのアクセス

ACCEPT 文，DISPLAY 文を使用して，コマンド行の情報へアクセスする方法について説明します。

コマンド行へのアクセスについては，マニュアル「COBOL2002　言語　拡張仕様編」「10.　コマンド行のアクセス」を参照してください。

10.3.1　コマンド行へのアクセスの種類と概要

ACCEPT 文，DISPLAY 文を使用して，次のコマンド行の情報を取得できます。

- コマンド行の引数の個数
- コマンド名称，およびコマンド行の引数の値

コマンド行の情報を取得するには，次の 2 種類の方法があります。

コマンド行の情報を個別に取得する方法

機能名 ARGUMENT-NUMBER，ARGUMENT-VALUE を使用して，コマンド行の引数の個数，コマンド名称，またはコマンド行の個々の引数の値を，一つずつ取得します。

詳細は，「10.3.2　引数を個別に取得する方法」を参照してください。

コマンド行の情報を一括して取得する方法

機能名 COMMAND-LINE を使用して，コマンド行のすべての引数の値を，一括して取得します。

詳細は，「10.3.3　引数を一括して取得する方法」を参照してください。

10.3.2　引数を個別に取得する方法

機能名 ARGUMENT-NUMBER および ARGUMENT-VALUE を使うことで，コマンド行に指定された引数の個数を取得できます。また，コマンド名称や，複数の引数の個々の値を取得できます。

(1)　コマンド行の引数の個数を取得する

コマンド行の引数の個数を取得するには，ACCEPT 文を使用します。

形式（引数の個数の取得）

```
ACCEPT 一意名1 FROM 呼び名1※
        〔END-ACCEPT〕
```

注※

呼び名 1 は，環境部の特殊名段落で，ARGUMENT-NUMBER に関連づけておく必要があります。

(2) コマンド行のファイル名称，引数の値を取得する（順読み込み）

コマンド行のファイル名称および引数の値を，コマンド行の先頭から順番に読み出して取得するには，ACCEPT 文を使用します。

形式（ファイル名称または引数の値の順読み出し）

```
ACCEPT 一意名2 FROM 呼び名2※
         〔ON EXCEPTION 無条件文1〕
         〔NOT ON EXCEPTION 無条件文2〕
         〔END-ACCEPT〕
```

注※

呼び名 2 は，環境部の特殊名段落で ARGUMENT-VALUE に関連づけておく必要があります。

(3) コマンド行のファイル名称，引数の値を取得する（乱読み込み）

コマンド行のファイル名称および引数の値を，コマンド行の任意の位置から読み出して取得するには，最初に DISPLAY 文を使用して読み出す位置を指定し，次に ACCEPT 文を使用してファイル名称または引数の値を読み出します。

形式（読み出し位置の指定）

```
DISPLAY   {一意名3｜整数1}   UPON 呼び名1※1
          〔END-DISPLAY〕
```

形式（指定位置からのファイル名称または引数の値の読み出し）

```
ACCEPT 一意名2 FROM 呼び名2※2
         〔ON EXCEPTION 無条件文1〕
         〔NOT ON EXCEPTION 無条件文2〕
         〔END-ACCEPT〕
```

注※1

呼び名 1 は，環境部の特殊名段落で ARGUMENT-NUMBER に関連づけておく必要があります。

注※2

呼び名 2 は，環境部の特殊名段落で ARGUMENT-VALUE に関連づけておく必要があります。

(4) 規則

(1)(2)(3)に共通する規則

- コマンド行へのアクセスができるのは，-Main,System オプションを指定した COBOL 主プログラムと，主プログラムに-Main,System オプションを指定した COBOL プログラムを持つ COBOL の副プログラムでだけです。

  コマンド行へのアクセスを使用している COBOL 主プログラムに-Main,V3 オプションを指定した場合，コンパイルエラーとなります。また，COBOL 副プログラムに，-Main,System オプション

を指定した COBOL 主プログラム以外のプログラムを持つ場合，実行時にエラーメッセージが出力され異常終了します。

(1)の形式での規則

- 引数および引数の個数を取得するときの転記規則を次に示します。

転記規則

- 引数は，一意名の左端から順に転送されて転記されます。
- 引数が一意名の領域より長い場合，一意名の長さで区切られます。
- 引数が一意名の領域より短い場合，標準コードの空白文字（X'20'）が埋められます。

- コマンド行に引数を指定していない場合，ACCEPT 文で取得する引数の個数は 0 になります。

(2)の形式での規則

- 順読み込みで引数を取得するときに，実行単位で最初に取得する引数は，実行可能ファイル名の次に指定された第 1 引数になります。
- 順読み込みで引数を取得する場合，上位プログラムで引数を取得し，さらに下位プログラムで引数を取得するときは，上位プログラムで最後に取得した引数の次の引数になります。

(3)の形式での規則

- 乱読み込みで引数を取得する場合，引数の位置を指定する DISPLAY 文と指定した位置の引数を取得する ACCEPT 文は，プログラム間にわたって指定できます。
- 一意名 3 の内容または整数 1 に 0 を指定し，ACCEPT 文で引数を取得した場合，実行可能ファイル名を取得します。
- コマンド行に指定した実行可能ファイル名を取得するときは，入力した文字列のまま取得されます。

(5) プログラム例

コマンド行の引数および引数の個数を取得する方法を，プログラム例を使って説明します。

実行時のコマンド行指定

```
a.out AAA BBB CCC DDD EEE FFF
```

プログラム例 1

コマンド行の引数および引数の個数を取得する例を示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
       ARGUMENT-NUMBER IS ARGNUM
       ARGUMENT-VALUE  IS ARGVAL
           :
       WORKING-STORAGE SECTION.
       01 ARGCNT  PIC 99.
       01 ARGDATA PIC X(10).
```

```
PROCEDURE DIVISION.
    :
    ACCEPT ARGCNT FROM ARGNUM.    …1.
    :
    ACCEPT ARGDATA FROM ARGVAL    …2.
           ON EXCEPTION ～
           NOT ON EXCEPTION ～
    END-ACCEPT.
    :
    DISPLAY 3 UPON ARGNUM.        …3.
    ACCEPT ARGDATA FROM ARGVAL    …4.
           ON EXCEPTION ～
           NOT ON EXCEPTION ～
    END-ACCEPT.
    :
```

1. コマンド行に指定した引数の個数６を取得します。
2. コマンド行に指定した引数'AAA'を取得します。
3. コマンド行に指定した３番目の引数位置を設定します。
4. 3.の DISPLAY 文で指定した３番目の引数'CCC'を取得します。

プログラム例２

順読み込みで，引数の取得がプログラム間にわたるときに，コマンド行の引数および引数の個数を取得する例を示します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE1.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
ARGUMENT-VALUE IS ARGVAL
    :
WORKING-STORAGE SECTION.
01 DATA1 PIC X(10).
PROCEDURE DIVISION.
    :
    ACCEPT DATA1 FROM ARGVAL …1.
           ON EXCEPTION ～
           :
           NOT ON EXCEPTION ～
           :
    END-ACCEPT.
    CALL ’SAMPLE2’.
    :

IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE2.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
ARGUMENT-VALUE IS ARGVAL
    :
WORKING-STORAGE SECTION.
01 DATA1 PIC X(10).
PROCEDURE DIVISION.
```

```
        :
        ACCEPT DATA1 FROM ARGVAL …2.
               ON EXCEPTION ～
               :
               NOT ON EXCEPTION ～
               :
        END-ACCEPT.
        :
```

1. コマンド行に指定した第 1 引数'AAA'を取得します。

2. コマンド行に指定した第 2 引数'BBB'を取得します。

プログラム例 3

順読み込みで，引数の取得がプログラム間にわたるときに，コマンド行の引数および引数の個数を取得する例を示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
       ARGUMENT-NUMBER IS ARGNUM
           :
       WORKING-STORAGE SECTION.
           :
       PROCEDURE DIVISION.
           :
           DISPLAY 3 UPON ARGNUM. …1.
           :
           CALL 'SAMPLE2'.
           :

       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
       ARGUMENT-VALUE IS ARGVAL
           :
       WORKING-STORAGE SECTION.
       01 DATA1 PIC X(10).
       PROCEDURE DIVISION.
           :
           ACCEPT DATA1 FROM ARGVAL …2.
                  ON EXCEPTION ～
                  :
                  NOT ON EXCEPTION ～
                  :
           END-ACCEPT.
           :
```

1. コマンド行に指定した第 3 引数の引数位置 3 を指定します。

2. 1.の DISPLAY 文で指定した第 3 引数'CCC'を取得します。

10.3.3　引数を一括して取得する方法

機能名 COMMAND-LINE を使うことで，コマンド行に指定された複数の引数を一括して取得できます。

(1)　コマンド行の引数を取得

ACCEPT 文実行時に，コマンド行の引数の内容を取得します。このとき，空白行で区切られた複数の引数を 1 回の ACCEPT 文でまとめて取得できます。

ACCEPT 文の形式

```
ACCEPT 一意名
       FROM  {COMMAND-LINE | COMMAND-LINEに対する呼び名}
        〔END-ACCEPT〕
```

(2)　コマンド行の引数の内容を更新

DISPLAY 文実行時に，一意名または定数で指定された内容をコマンド行の引数の内容を格納しているコマンド行バッファに上書きします。COMMAND-LINE 指定のある DISPLAY 文の実行後に，COMMAND-LINE 指定のある ACCEPT 文を実行すると，先に実行した DISPLAY 文で設定した内容が取り出されます。

DISPLAY 文の形式

```
DISPLAY  {一意名 | 定数}
        UPON  {COMMAND-LINE | COMMAND-LINEに対する呼び名}
         〔END-DISPLAY〕
```

(3)　規則

- コマンド行へのアクセスができるのは，-Main,System オプションを指定した COBOL 主プログラムと，主プログラムに-Main,System オプションを指定した COBOL プログラムを持つ COBOL の副プログラムでだけです。

  コマンド行へのアクセスを使用している COBOL 主プログラムに-Main,V3 オプションを指定した場合，コンパイルエラーとなります。また，COBOL 副プログラムに，-Main,System オプションを指定した COBOL 主プログラム以外のプログラムを持つ場合，実行時にエラーメッセージが出力され異常終了します。

- ACCEPT 文で数字項目を正しく受け取るにはコンパイル時に-NumAccept オプションが必要です。また，2 進項目，内部 10 進項目，内部浮動小数点数字項目使用時には-NumAccept オプションを指定しないとコンパイルエラーとなります。

10.4　環境変数へのアクセス

ACCEPT 文，DISPLAY 文を使用して，環境変数の値を取得・設定する方法について説明します。

環境変数のアクセスについては，マニュアル「COBOL2002　言語　拡張仕様編」「10.　コマンド行のアクセス」を参照してください。

10.4.1　環境変数の値の読み込み

DISPLAY 文（環境変数名の設定）の形式

```
DISPLAY  {一意名4 | 定数1}  UPON 呼び名3※1
         〔END-DISPLAY〕
```

ACCEPT 文（環境変数の値の取得）の形式

```
ACCEPT 一意名2 FROM 呼び名4※2
        〔ON EXCEPTION 無条件文3〕
        〔NOT ON EXCEPTION 無条件文2〕
        〔END-ACCEPT〕
```

注※1

呼び名 3 は，環境部の特殊名段落で，ENVIRONMENT-NAME に関連づけておく必要があります。

注※2

呼び名 4 は，環境部の特殊名段落で，ENVIRONMENT-VALUE に関連づけておく必要があります。

10.4.2　環境変数への値の書き出し

DISPLAY 文（環境変数名の設定）の形式

```
DISPLAY  {一意名4 | 定数1}  UPON 呼び名3※1
         〔END-DISPLAY〕
```

DISPLAY 文（環境変数の値の書き出し）の形式

```
DISPLAY  {一意名2 | 定数2}  UPON 呼び名4※2
         〔ON EXCEPTION 無条件文1〕
         〔NOT ON EXCEPTION 無条件文2〕
         〔END-DISPLAY〕
```

注※1

呼び名 3 は，環境部の特殊名段落で，ENVIRONMENT-NAME に関連づけておく必要があります。

注※2

呼び名 4 は，環境部の特殊名段落で，ENVIRONMENT-VALUE に関連づけておく必要があります。

10.4.3　環境変数へのアクセスに関する規則

- 環境変数名および環境変数の値を設定する場合，DISPLAY 文に指定した一意名または定数中に含まれる空白も有効となります。
- 環境変数名を設定する場合，環境変数名には"="を指定できません。指定した場合，正常に動作しなくなることがあります。
- 環境変数名を設定する場合，環境変数名に NULL（X'00'）が含まれているとき，NULL の直前までの値が有効となります。環境変数の値を設定する場合，環境変数の値に NULL（X'00'）が含まれているとき，NULL の直前までの値が有効となります。
- 環境変数の値を取得する場合，環境変数が存在しないとき，ACCEPT 文で実行時エラーとなります。
- 環境変数名の先頭に NULL（X'00'）を指定したときの規則を次に示します。
  - 環境変数の値の取得処理（ACCEPT 文）は，エラーとなります。
  - 環境変数の値の設定処理（DISPLAY 文）は，何も処理しません。
- 次に示す文の組み合わせは，プログラム間にわたって指定できます。
  - 環境変数名を設定する DISPLAY 文と環境変数の値を取得する ACCEPT 文
  - 環境変数名を設定する DISPLAY 文と環境変数の値を設定する DISPLAY 文
- 環境変数の値を取得するときの転記規則を次に示します。
  - 環境変数の値は，一意名の左端から順に転送されて転記されます。
  - 環境変数の値が一意名の領域より長い場合，一意名の長さで区切られます。
  - 環境変数の値が一意名の領域より短い場合，標準コードの空白文字（X'20'）が埋められます。
- 次の環境変数には，環境変数へのアクセス機能で値を書き出せません。
  - CBLTDEXEC
- 環境変数の値の書き出しを行う DISPLAY 文の実行回数は，環境変数 CBLENVMAX で設定します。

  形式

  CBLENVMAX=回数

  規則

  - 回数は，8 けた以内の符号なしの正の整数で指定します。
  - 環境変数 CBLENVMAX を指定しなかった場合，または値が 0 の場合は，50 が仮定されます。
  - 環境変数 CBLENVMAX で負の値を設定した場合，または 9 けた以上の値を設定した場合はエラーメッセージが出力され異常終了します。
  - DISPLAY 文による環境変数への値の書き出しによって，環境変数 CBLENVMAX を設定することはできません。
- 環境変数の値をクリアせずに COBOL プログラムの実行を終了し，あとに同じプロセスで COBOL プログラムが動作した場合，環境変数を設定しなくても以前実行時に設定した環境変数の値を取得できます。

COBOL プログラムを終了する前に NULL（X'00'）で始まる値を設定することで，値のない環境変数を設定できます。

10.4.4 使用例

実行時の環境変数指定

```
CBLABNLST=/tmp/abnlst
CBLDDUMP=/tmp/dumplst
```

プログラム例 1

環境変数へのアクセス例を次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
       ENVIRONMENT-NAME IS ENVNAM
       ENVIRONMENT-VALUE IS ENVVAL.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ENVNAME1 PIC X(9).
       01 ENVNAME2 PIC X(8).
       01 ENVDATA PIC X(20).
       PROCEDURE DIVISION.
           :
           MOVE 'CBLABNLST' TO ENVNAME1.
           DISPLAY ENVNAME1 UPON ENVNAM.  …1.
           ACCEPT ENVDATA FROM ENVVAL     …2.
               ON EXCEPTION ～
               :
               NOT ON EXCEPTION ～
               :
           END-ACCEPT.
           :
           MOVE 'CBLDDUMP' TO ENVNAME2.
           MOVE '/tmp/dumplst2' TO ENVDATA.
           DISPLAY ENVNAME2 UPON ENVNAM.  …3.
           DISPLAY ENVDATA UPON ENVVAL    …4.
               ON EXCEPTION ～
               :
               NOT ON EXCEPTION ～
               :
           END-DISPLAY.
           :
```

1. 値を取得したい環境変数名（CBLABNLST）を指定します。
2. 1.で指定した環境変数（CBLABNLST）の値（/tmp/abnlst）を取得します。
3. 値を設定したい環境変数名（CBLDDUMP）を指定します。
4. 3.で指定した環境変数（CBLDDUMP）に値（/tmp/dumplst2）を設定します。

プログラム例 2

環境変数へのアクセスがプログラム間にわたる場合の，環境変数の値の取得例を次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
       ENVIRONMENT-NAME IS ENVNM.
                 :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ENVNAME PIC X(9).
       PROCEDURE DIVISION.
                 :
           MOVE 'CBLABNLST' TO ENVNAME.
           DISPLAY ENVNAME UPON ENVNM.    …1.
                 :
           CALL 'SAMPLE2'.
                 :


       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
       ENVIRONMENT-VALUE IS ENVVAL.
           :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ENVDATA PIC X(20).
       PROCEDURE DIVISION.
           :
           ACCEPT ENVDATA FROM ENVVAL  …2.
                  ON EXCEPTION ～
                  :
                  NOT ON EXCEPTION ～
                  :
           END-ACCEPT.
           :
```

1. 値を取得したい環境変数名（CBLABNLST）を指定します。

2. 1.で指定した環境変数（CBLABNLST）の値（/tmp/abnlst）を取得できます。

プログラム例 3

環境変数へのアクセスがプログラム間にわたる場合の，環境変数の値の設定例を次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
       ENVIRONMENT-NAME IS ENVNM.
                 :
       DATA DIVISION.
```

```
       WORKING-STORAGE SECTION.
       01 ENVNAME PIC X(9).
       PROCEDURE DIVISION.
                :
           MOVE 'CBLABNLST' TO ENVNAME.
           DISPLAY ENVNAME UPON ENVNM.  …1.
                :
           CALL 'SAMPLE2'.
                :


       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
       ENVIRONMENT-VALUE IS ENVVAL.
           :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ENVDATA PIC X(20).
       PROCEDURE DIVISION.
           :
           MOVE '/tmp/abnlst' TO ENVDATA.
           DISPLAY ENVDATA UPON ENVVAL  …2.
                   ON EXCEPTION ～
                   :
                   NOT ON EXCEPTION ～
                   :
           END-DISPLAY.
           :
```

1.値を設定したい環境変数名（CBLABNLST）を指定します。

2.1.で指定した環境変数（CBLABNLST）に（/tmp/abnlst）を設定できます。

10.5　COBOL ログファイル出力機能

10.5.1　COBOL ログファイル出力機能の概要

COBOL プログラムで DISPLAY 文によってデータやメッセージを出力する場合，その出力でエラーが発生すると実行中止となったり，出力内容が破棄されてユーザにとって必要な情報が取得できなくなったりすることがあります。このような場合に，出力内容を破棄しないで，COBOL 独自のログファイルに出力できます。

COBOL ログファイル出力機能は，システム入出力関数レベルを指定しているとき，DISPLAY 文でデータ出力中にエラーが発生した場合に使用できます。詳細は「10.2.5　システム入出力関数レベルの指定」を参照してください。

10.5.2　COBOL ログファイルの出力形式

COBOL ログファイルの出力先を次に示します。DISPLAY 文の出力データを COBOL ログファイルへ出力中にエラーが発生した場合は，標準出力に出力します。

(1)　出力先ディレクトリ

COBOL ログファイルの出力先ディレクトリは，COBOL2002 インストールディレクトリ/cbllog となります。

このディレクトリは，COBOL2002 のインストール時に作成されます。

(2)　ファイル名称

次の名称規則で COBOL ログファイルが作成されます。

形式

cblMMDD_HHMMSS_nnnn.log

MMDD

月日を 4 けたで表します。

HHMMSS

時刻を時分秒の 6 けたで表します。

nnnn

プロセス ID を表します。

けた数は可変です。

（例）

プロセス ID7847 の 9 月 27 日 19 時 24 分 33 秒に作成されたファイル

COBOL2002 インストールディレクトリ/cbllog/cbl0927_192433_7847.log

注意事項

上記のファイルを COBOL が自動的に消去することはありません。不要になったファイルは手動で消去してください。

(3) 出力内容

COBOL ログファイル出力機能で出力される内容を，次に示します。

日付※1	時刻	ホスト名	ログ識別※2	メッセージ／データ

注※1

表示は英語出力固定です。環境変数 LANG での指定はできません。

注※2

ログ種別は「DISPLAY:」と表示します。

（例）

Sep 27 19:24:33 TERM1 DISPLAY:DISPLAY 文のデータ…

11 整列併合機能

整列併合機能は，SORT 文，MERGE 文を記述したプログラムで実行します。この章では，整列併合機能で使用するファイルや，ファイルの割り当て方法，使用するメモリサイズなどについて説明します。

11.1　使用できるファイル

整列併合機能を使ったプログラムで使用できるファイルを次に示します。

表 11–1　整列併合機能で使用できるファイル

種別	ファイル編成	用途	備考
入力用ファイル	順編成ファイル 相対編成ファイル 索引編成ファイル テキスト編成ファイル	• 整列用レコードの入力ファイル • 併合用レコードの入力ファイル	USING 指定のとき，最大 12 ファイル指定できる。
出力用ファイル	順編成ファイル 相対編成ファイル 索引編成ファイル テキスト編成ファイル	• 整列済みレコードの出力ファイル • 併合済みレコードの出力ファイル	
作業用ファイル	－	• 整列機能が使用する整列作業用ファイル	整列機能が内部的に使用する。

なお，整列ファイル，併合ファイルについては，マニュアル「COBOL2002　言語　標準仕様編」「5.1.15　整列ファイル」および「COBOL2002　言語　標準仕様編」「5.1.16　併合ファイル」を参照してください。

11.2　ファイルの割り当て

整列処理，併合処理で使用するファイルの割り当て方法について説明します。

11.2.1　入出力用ファイル

SELECT 句で指定した整列併合用のファイルに対して実行時に物理ファイルを割り当てる方法については，「6.2　ファイル割り当ての共通規則」を参照してください。

11.2.2　整列作業用ファイル

SORT 文を使用する場合，整列処理用の作業ファイルのディレクトリの名称は，環境変数 CBLSORTWORK で指定します。

形式

```
CBLSORTWORK=ディレクトリ名
```

ディレクトリ名は，パスプレフィックスで指定します。

(例)

```
CBLSORTWORK=/TMP/WORK
export CBLSORTWORK
```

環境変数 TMPDIR の指定がある場合，整列処理用の作業ファイルは，環境変数 CBLSORTWORK で指定したディレクトリではなく，環境変数 TMPDIR で指定したディレクトリ下に割り当てます。

整列処理用の作業ファイルを割り当てるディレクトリの優先順位を次に示します。

1. 環境変数 TMPDIR で指定したディレクトリ
2. 環境変数 CBLSORTWORK で指定したディレクトリ
3. /tmp

作業ファイルのディレクトリには，読み込み権限と書き込み権限を付与してください。

11.2.3　注意事項

- 整列処理を実行中に異常終了が発生すると，整列作業用ファイルが削除されないで残る場合があります。
- 併合処理の入力ファイルは，MERGE 文の ASCENDING KEY または DESCENDING KEY で指定したとおりに整列されている必要があります。整列されていないとプログラムが異常終了します。

11.3　使用するメモリサイズ

整列処理，併合処理で使用するメモリサイズについて説明します。

11.3.1　整列処理のメモリサイズ

整列処理のメモリサイズは，環境変数 CBLSORTSIZE で指定します。

形式

```
CBLSORTSIZE=メモリサイズ
```

メモリサイズ

整列処理で使用するメモリサイズを，キロバイト単位で指定します。有効となる値は，8 けた以内の符号なし整数（0～99,999,999）です。この範囲を超える値を指定した場合，実行時エラーとなります。

（例）

```
CBLSORTSIZE=1000
export CBLSORTSIZE
```

規則

- 環境変数 CBLSORTSIZE に値が設定されていないときは，SORT-CORE-SIZE 特殊レジスタの値が適用されます。また，環境変数と特殊レジスタの両方に値が設定されているときは，環境変数に指定した値が有効となります。

  整列処理のメモリサイズの注意点については，「11.5　注意事項」を参照してください。
- 整列処理するレコード長が 32 キロバイトを超える場合，整列機能が確保するメモリサイズは 256 キロバイト以上を指定する必要があります。

メモリサイズの計算式

メモリ所要量（キロバイト）＝ 16 ＋ S

- S は，環境変数 CBLSORTSIZE または SORT-CORE-SIZE 特殊レジスタで指定した値です。

11.3.2　併合処理のメモリサイズ

併合処理で使用するメモリサイズを，次に示します。

メモリサイズの計算式

AIX(32)および Linux(x86)の場合

メモリ所要量（バイト）＝ 32 ＋（最大レコード長※1 ＋ 28 ＋ キーの合計長※2）×（併合ファイル数 ＋ 1）

注※1

4バイト境界に切り上げた値

注※2

MERGE文に指定されたすべてのキー長の合計値を，4バイト境界に切り上げた値

AIX(64)およびLinux(x64)の場合

メモリ所要量（バイト）＝48＋（最大レコード長※1＋32＋キーの合計長※2）×（併合ファイル数＋1）

注※1

8バイト境界に切り上げた値

注※2

MERGE文に指定されたすべてのキー長の合計値を，8バイト境界に切り上げた値

なお，算出したメモリサイズの記憶領域が確保できなかった場合，プログラムは異常終了します。

11.4　使用できる特殊レジスタ

整列併合機能で使用できる特殊レジスタについて説明します。

11.4.1　特殊レジスタの種類

特殊レジスタの種類を次に示します。

表 11–2　特殊レジスタの種類

特殊レジスタ	属性	初期値	有効となる値
SORT-RETURN	PIC S9(4) USAGE COMP	0	0，16
SORT-CORE-SIZE	PIC S9(8) USAGE COMP	0	-99,999,999～99,999,999※1
SORT-FILE-SIZE	PIC S9(8) USAGE COMP	0	―※2
SORT-MODE-SIZE	PIC S9(5) USAGE COMP	0	―※2
SORT-MESSAGE	PIC X(8) USAGE DISPLAY	空白	―※2

注※1

128 未満の値を指定した場合，1,024 が仮定されます。

注※2

これらの特殊レジスタにも値は設定できますが，どの値も整列処理に対しては無効となります。

11.4.2　SORT-RETURN 特殊レジスタ

この特殊レジスタに値 16 を設定すると，整列併合の操作を強制的に終了できます。

入力または出力手続きで値 16 を設定すると，次の RELEASE 文または RETURN 文を実行後，整列併合の操作が終了し，次の手続き文に制御が渡ります。

16 以外の値を設定した場合は，整列併合の操作が継続して実行されます。

11.4.3　SORT-CORE-SIZE 特殊レジスタ

整列処理実行前にこの特殊レジスタに値を設定することで，整列処理の作業用メモリのサイズをキロバイト単位で指定できます。128 未満の値を指定した場合，1,024 が仮定されます。また，環境変数 CBLSORTSIZE が指定されているときは，環境変数で指定された値が適用されます。

なお，この特殊レジスタは，併合処理に関しては意味を持ちません。

11.5　注意事項

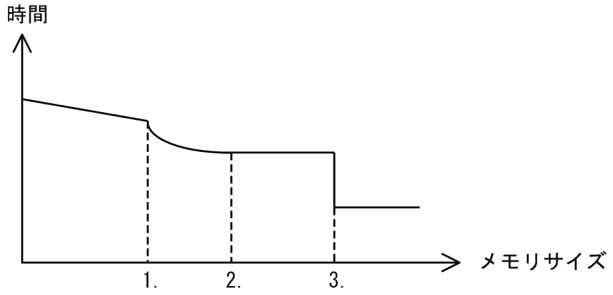
11.5.1　処理時間の短縮

(1)　メモリサイズと処理時間

メモリサイズと処理時間との間には，メモリサイズが大きいほど処理時間が短くなるという関係があります。しかし，実際には，設定したメモリサイズがソート実行時に使用されるメモリよりも著しく大きいと，逆に処理時間が掛かることがあります。したがって，メモリサイズを指定するときは，レコード件数，レコード長，キー長，および実メモリサイズを考慮し，適切な値にする必要があります。

メモリサイズと処理時間との関係を次に示します。

図 11–1　メモリサイズと処理時間との関係



1. : $\sqrt{\text{データ件数}} \times (\text{レコード長}+\text{キー長}+40)$ バイト

2. : $\sqrt{\text{データ件数} \times 66000 \times (\text{レコード長}+\text{キー長}+20)}$ バイト

3. : データ件数×(レコード長＋キー長＋16)バイト

メモリサイズが 1.で示す値より小さいとき，何度かソートを繰り返すため，処理時間は長くなります。

メモリサイズが 3.で示す値以上のとき，一時ファイルを使用しなくてもソートできるため，処理時間は短くなります。ただし，データ件数が多過ぎると，逆に処理時間が長くなったり，処理できなかったりする場合があります。

通常メモリサイズは，1.で示す値と 3.で示す値の間の値を設定します。その場合の目安は，2.で示す値です。1.で示す値以上で，3.で示す値未満の値では，2.で示す値より大きな値を設定しても処理時間は短くなりません。

(2)　キー属性の指定

キー属性が文字の場合，数字の場合と比べてソート処理の効率が向上します。したがって，ファイル設計時にはキーとなる部分の属性を文字にしておくとソート処理の時間を短くできます。

(3) WITH DUPLICATES 指定

SORT 文に WITH DUPLICATES 指定をしない方がソート処理の時間を短くでき，また，作業ファイルの容量も小さくて済みます。

11.5.2 その他の注意事項

(1) RELEASE 文と RETURN 文

RELEASE 文は INPUT PROCEDURE 中で実行します。RETURN 文は OUTPUT PROCEDURE 中で実行します。誤って実行するとプログラムが異常終了します。

(2) 入出力エラー

整列処理，併合処理で使用するファイルに対して入出力エラーが発生したときの処理について説明します。

なお，共通例外処理を使用する場合の詳細については，「21. 共通例外処理」を参照してください。

(a) 入出力ファイル

入出力ファイルに入出力エラーが発生すると，FILE STATUS 句，USE ERROR 手続きが有効となります。これらの指定がない場合，プログラムは異常終了します。

(b) 整列作業用ファイル

整列作業用ファイルに入出力エラーが発生すると，プログラムは異常終了します。

(3) キー数

SORT 文，MERGE 文で指定できるキーの数は最大 64 個です。

12 画面入出力機能（AIX で有効）

通信節，画面節（SCREEN SECTION および WINDOW SECTION）による画面機能を使うと，ディスプレイやプリンタとの間でデータの送受信ができます。この章では，それぞれの画面機能の使い方について説明します。

12.1　通信節による画面機能（AIX(32)で有効）

12.1.1　機能の概要

通信節（COMMUNICATION SECTION）による画面機能を使用すると，ディスプレイとの間で画面データを送受信したり，プリンタに帳票データを送信したりできます。画面データを送受信する際には，画面全体を制御したり，けい線，高輝度表示などの機能を利用したりできます。

通信記述項による画面機能については，マニュアル「COBOL2002　言語　拡張仕様編」「11.　通信節による画面機能」を参照してください。

12.1.2　画面に対する入出力

(1)　画面の定義

通信節による画面機能を使用する場合，画面は XMAP3 のドロー（パネル定義）機能を利用して定義します。ドロー（パネル定義）機能については，次に示すマニュアルを参照してください。

XMAP3 Version 4 の場合

- マニュアル「画面・帳票サポートシステム XMAP3 プログラミングガイド 画面編」
- マニュアル「画面・帳票サポートシステム XMAP3 開発・実行ガイド」

XMAP3 Version 5 の場合

- マニュアル「XMAP3 Version 5 画面・帳票サポートシステム XMAP3 開発ガイド」
- マニュアル「XMAP3 Version 5 画面・帳票サポートシステム XMAP3 プログラミングガイド」
- マニュアル「XMAP3 Version 5 画面・帳票サポートシステム XMAP3 実行ガイド」

(2)　使用する文

画面データを送受信したり画面を閉じたりするには，原始プログラム中のデータ部通信記述項の FOR 句で"I-O WS"を指定し，次に示す文を使用します。

表 12-1　画面データの送受信で使用する文

使用する文	処理
SEND 文	ディスプレイに画面データを送信し，応答の完了を待たないで非同期にプログラムを続行する。 • NO REPLY 指定がないとき 受信した画面データは，RECEIVE 文で受け取る。 • NO REPLY 指定があるとき ディスプレイに画面データを送信する。RECEIVE 文で画面データを受け取ることはできない。

使用する文	処理
RECEIVE 文	NO REPLY 指定のない SEND 文で受信したデータを受け取る。
TRANSCEIVE 文	ディスプレイに画面データを送信し，受信した画面データを受け取る。
DISABLE 文	ディスプレイに出力された画面データを消去する。

- NO REPLY 指定のない SEND 文と RECEIVE 文との間で SEND 文または TRANSCEIVE 文を実行した場合，最後の SEND 文または TRANSCEIVE 文が有効となります。

(3) 画面の表示モード

画面データの表示モードには次の 2 種類のモードがあります。

- 書き換え（ERASE）モード：画面を消去して画面データを表示する。
- 上書き（WRITE）モード：画面を消去しないで画面データを表示する。

SEND 文または TRANSCEIVE 文では，ドロー（パネル定義）機能で画面（パネル）定義時に指定した表示モードで画面データを表示します。ただし，開かれていない画面データの送受信先に対して SEND 文または TRANSCEIVE 文を初めて実行する場合，上書き（WRITE）モードを指定してはなりません。

ドロー（パネル定義）機能で，画面（パネル）定義時にこのシステムに任せる指定をした場合，次の規則に従って画面データが表示されます。

- 初めての送受信要求で指定した物理マップ名と異なるときは，書き換え（ERASE）モードで表示される。
- 現在の送受信要求で指定した物理マップ名が直前の送受信要求で指定した物理マップ名と同じときは，上書き（WRITE）モードで表示される。

(4) 実行時のカーソルの位置づけ

プログラム実行時にカーソルの位置を変更するときは，原始プログラム中で論理マップとして展開したカーソルフィールドに値を設定します。

(5) データ有無コード

データ有無コードとは，論理項目に値を設定しているかどうかを区別するためのコードで，1 バイトのコード X'00'～X'FF'で指定します。

データ有無コードには，原始プログラム中の通信記述項の DATA ABSENCE CODE 句で指定したデータ名に設定した値が使われます。

DATA ABSENCE CODE 句を指定したときは，プログラム中の通信記述項ごとの最初の通信文を実行する前に値を必ず設定しておく必要があります。データ名に値を設定しなかった場合の動作は保証しません。また，通信文の実行後に値を変更してはなりません。

(6) 実行状態を示すコード

続行できない異常な状態が通信文で発生すると，エラーメッセージが出力されます。通信記述項に STATUS KEY 句の指定があれば，通信文の実行状態を示す 5 けたのコードがこの句で指定したデータ名に設定され，処理が続行されます。

STATUS KEY 句がないときに続行できないエラーが発生すると，プログラムは異常終了します。

STATUS KEY 句のデータ名に設定されるコードについては，マニュアル「COBOL2002　言語　拡張仕様編」「11.1.1　通信記述項（CD）（通信節による画面機能）」を参照してください。

(7) 送受信先の設定方法

画面機能では，送受信先を指定することで，ネットワーク上に接続されたディスプレイに画面データを送受信できます。送受信先の指定方法について，次に示します。

(a) 送受信先の決定のしかた

画面データの送受信先は，通信記述項での SYMBOLIC TERMINAL 句の指定の有無と，環境変数の指定の有無によって次のように決定されます。

表 12-2　画面データの送受信先

条件		SYMBOLIC TERMINAL 句でのデータ名の指定	
		あり	なし
環境変数の指定	あり	環境変数 CBLTERM_xxx で指定したディスプレイ ただし，データ名に設定した内容の先頭 1 バイトが空白文字（X'20'）の場合は，環境変数 CBLTERMID で指定したディスプレイ	環境変数 CBLTERMID で指定したディスプレイ
	なし	データ名で指定した仮想端末 ただし，データ名に設定した内容の先頭 1 バイトが空白文字（X'20'）の場合は，XMAP3 のセットアップの値（仮定値）※	XMAP3 のセットアップの値（仮定値）※

注※

マニュアル「画面・帳票サポートシステム XMAP3 Server」を参照してください。

(b) 環境変数による送受信先の設定

送受信先を設定するための環境変数には次の 2 種類があります。

仮想端末名を指定する環境変数

次の環境変数で仮想端末名を指定します。ここで指定した仮想端末名は，SYMBOLIC TERMINAL 句のデータ名に"xxx"で示す名称を指定したときに有効となります。

形式

```
CBLTERM_xxx=ディスプレイの仮想端末名
```

規則

xxx および仮想端末名は 8 文字以内の文字列で指定します。

仮想端末名の仮定値を指定する環境変数

SYMBOLIC TERMINAL 句の指定を省略したときに仮定される仮想端末の名称を次の環境変数で指定します。

形式

```
CBLTERMID=ディスプレイの仮想端末名
```

規則

仮想端末名は 8 文字以内の文字列で指定します。

(c) SYMBOLIC TERMINAL 句の指定

通信記述項の SYMBOLIC TERMINAL 句のデータ名に環境変数名または仮想端末名を指定することで，画面データの送受信先を指定できます。

環境変数名による指定方法

データ名に環境変数 CBLTERM_xxx の"xxx"の名称を指定します。環境変数で指定した仮想端末名が送受信先として設定されます。

仮想端末名による指定方法

環境変数が指定されていないときは，データ名に仮想端末名を指定することで送受信先を指定できます。

(8) 送受信間の物理マップ

ディスプレイに対して画面データを送受信する場合，SEND 文（NO REPLY 指定なし）の前に通信記述項の MAP NAME 句のデータ名に物理マップ名を指定し，画面データを送受信します。RECEIVE 文はその物理マップ名に対して画面データを受信するため，RECEIVE 文で物理マップ名は指定しません。

複数の仮想端末に送信した画面データを受信するとき，それぞれの最後に送信した SEND 文で指定した物理マップが，RECEIVE 文の物理マップとなります。

12.1.3 仮想端末の共用

送受信先が同じ仮想端末に対して，複数プログラムで定義した通信記述項を使って通信文を実行すると，複数の画面が生成されます。このとき実行時環境変数を指定することで，一つの仮想端末を複数プログラム間で共用し，送受信できます。

また，-CompatiV3 オプションを指定してコンパイルしたプログラムでも同様に，一つの仮想端末を共用できます。

(1) 仮想端末の共用の指定

複数プログラム間で一つの仮想端末を共用して送受信したい場合は，次の実行時環境変数を指定します。

```
CBLTERMSHAR=YES
```

この環境変数に YES を指定すると，実行単位中のすべてのプログラムで同一名称の仮想端末を共用できます。指定がない，または YES 以外の文字が指定されている場合は，NO が仮定されます。NO の場合は，送受信先に同一名称の仮想端末を指定しても異なる仮想端末として送受信します（SEND 文実行時に複数の画面が生成される）。

(2) 注意事項

- 一つの実行単位ファイル中の通信節を使用するプログラムに対して，-CompatiV3 オプションを指定したオブジェクトファイルと未指定のオブジェクトファイルを混在して動作させてはいけません。混在させた場合の動作は保証しません。
- 複数プログラム間で共用した仮想端末に対して送信した画面データを受信するとき，最後に送信した SEND 文と同じ通信記述項を使って受信しなければいけません。
- 仮想端末は，一つの COBOL 実行単位中に，複数のプログラム間で共用できます。CALL 文で呼び出す実行可能ファイルなど，別の COBOL 実行単位との共用はできません。

12.1.4 プリンタに対する帳票出力

(1) 帳票の定義

通信節による帳票出力機能を使用する場合，帳票は XMAP3 のドロー（パネル定義）機能を利用して定義します。ドロー（パネル定義）機能については，次に示すマニュアルを参照してください。

XMAP3 Version 4 の場合

- マニュアル「画面・帳票サポートシステム XMAP3 プログラミングガイド 帳票編」
- マニュアル「画面・帳票サポートシステム XMAP3 開発・実行ガイド」

XMAP3 Version 5 の場合

- マニュアル「XMAP3 Version 5 画面・帳票サポートシステム XMAP3 開発ガイド」
- マニュアル「XMAP3 Version 5 画面・帳票サポートシステム XMAP3 プログラミングガイド」
- マニュアル「XMAP3 Version 5 画面・帳票サポートシステム XMAP3 実行ガイド」

(2) 使用する文

帳票データをプリンタへ送信したり，プリンタを閉じたりするには，原始プログラム中のデータ部通信記述項の FOR 句で"OUTPUT WS"を指定し，次に示す文を使用します。

表 12–3　帳票データの送信で使用する文

使用する文	処理
SEND 文	帳票データをプリンタへ送信する。
DISABLE 文	プリンタを閉じる（閉じるプリンタは，DISABLE 文を実行した時点で指定されているプリンタである）。

(3)　実行状態を示すコード

続行できない異常な状態が通信文で発生すると，エラーメッセージが出力されます。通信記述項に STATUS KEY 句の指定があれば，通信文の実行状態を示す 5 けたのコードがこの句で指定したデータ名に設定され，処理が続行されます。

STATUS KEY 句がないときに続行できないエラーが発生すると，プログラムは異常終了します。

STATUS KEY 句のデータ名に設定されるコードについては，マニュアル「COBOL2002　言語　拡張仕様編」「11.1.1　通信記述項（CD）（通信節による画面機能）」を参照してください。

(4)　送信先の設定方法

帳票出力機能では，送受信先を指定することで，ネットワーク上に接続されたプリンタに帳票データを送信できます。送信先の指定方法について，次に示します。

(a)　送信先の決定のしかた

帳票データの送信先は，通信記述項での SYMBOLIC TERMINAL 句の指定の有無と，環境変数の指定の有無によって次に示すように決定されます。

表 12–4　帳票データの送信先

条件		SYMBOLIC TERMINAL 句でのデータ名の指定	
		あり	なし
環境変数の指定	あり	環境変数 CBLPRNT_xxx で指定したプリンタ ただし，データ名に設定した内容の先頭 1 バイトが空白文字（X'20'）の場合は，環境変数 CBLPRNTID で指定したプリンタ	環境変数 CBLPRNTID で指定したプリンタ
	なし	データ名で指定した仮想端末 ただし，データ名に設定した内容の先頭 1 バイトが空白文字（X'20'）の場合は，XMAP3 のセットアップの値（仮定値）※	XMAP3 のセットアップの値（仮定値）※

注※

マニュアル「画面・帳票サポートシステム XMAP3 Server」を参照してください。

(b)　環境変数による送信先の設定

送信先を設定するための環境変数には次の 2 種類があります。

仮想端末名を指定する環境変数

次の環境変数で仮想端末名を指定します。ここで指定した仮想端末名は，SYMBOLIC TERMINAL 句のデータ名に"xxx"で示す名称を指定したときに有効となります。

形式

```
CBLPRNT_xxx=プリンタの仮想端末名
```

規則

xxx および仮想端末名は 8 文字以内の文字列で指定します。

仮想端末名の仮定値を指定する環境変数

SYMBOLIC TERMINAL 句の指定を省略したときに仮定される仮想端末の名称を次の環境変数で指定します。

形式

```
CBLPRNTID=プリンタの仮想端末名
```

規則

仮想端末名は 8 文字以内の文字列で指定します。

(c) SYMBOLIC TERMINAL 句の指定

通信記述項の SYMBOLIC TERMINAL 句のデータ名に環境変数名または仮想端末名を指定することで，帳票データの送信先を指定できます。

環境変数名による指定方法

データ名に環境変数 CBLPRNT_xxx の"xxx"の名称を指定します。環境変数で指定した仮想端末名が送受信先として設定されます。

仮想端末名による指定方法

環境変数が指定されていないときは，データ名に仮想端末名を指定することで送信先を指定できます。

(5) 送受信間の物理マップ

プリンタに対して帳票データを送受信する場合，SEND 文（NO REPLY 指定なし）の前に通信記述項の MAP NAME 句のデータ名に物理マップ名を指定し，帳票データを送信します。

12.2　画面節（SCREEN SECTION）による画面機能

画面節（SCREEN SECTION）を使用した画面の入出力について説明します。

画面節（SCREEN SECTION）を使用した画面の入出力については，マニュアル「COBOL2002　言語拡張仕様編」「12.　画面節（SCREEN SECTION）による画面機能」を参照してください。

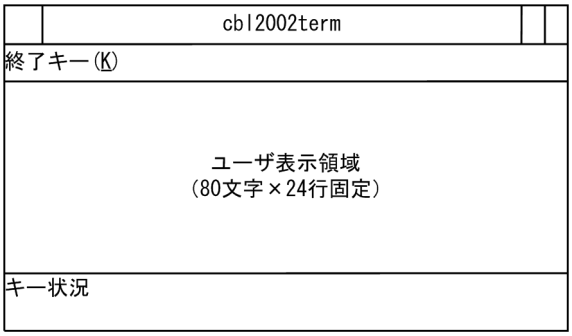
なお，SCREEN SECTION と WINDOW SECTION は，同時に使用できません。同時に使用した場合，エラーメッセージが出力されます。

12.2.1　画面の種類と構成

画面節（SCREEN SECTION）では，主画面とエラー表示画面の 2 種類の画面を使用します。

(1)　主画面

主画面は，画面の入出力で使用する主画面であり，タイトルバー，メニューバー，ユーザ表示領域から構成されます。主画面の構成を次に示します。



メニューバーの［終了キー］をクリックすると，次のドロップダウンメニューが表示されます。これらをクリックすると，［Enter］キーや［F1］～［F24］キーを押したのと同じ結果が得られます。

実行(R)
PF1
PF2
:
PF21(l)
PF22(m)
PF23(n)
PF24(o)

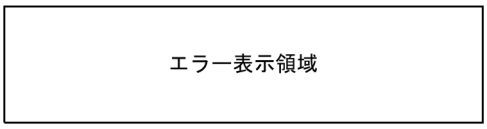
注

メニューは PF24 まで表示されます。

［PF13］～［PF24］キーは，［Ctrl］キーを押したまま［PF1］～［PF12］キーを押すことを意味します。

(2) エラー表示画面

エラー表示画面は，誤ったデータを入力したときなどに自動的に表示される画面であり，エラー表示領域だけから構成されます。エラー表示画面の構成を次に示します。



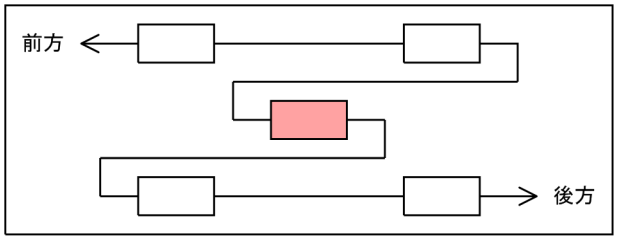
12.2.2 キーの機能

カーソルの移動などで使用するキーの意味を次に示します。

使用するキー	意味
[→]	カーソルが右に 1 カラム移動します。ただし，フィールドの右端のときは，後方のフィールドの先頭に移動します。
[←]	カーソルが左に 1 カラム移動します。ただし，フィールドの左端のときは，前方のフィールドの末尾に移動します。
[↑]	前方のフィールドの先頭にカーソルが移動します。
[↓]	後方のフィールドの先頭にカーソルが移動します。ただし，後方にフィールドがない場合は，フィールドの右端にカーソルが移動します。
[Tab]	後方のフィールドの先頭にカーソルが移動します。
[BackSpace]	現在カーソルが位置づけられているカラムの入力を取り消し，カーソルが左に 1 カラム移動します。
[Home]	フィールドの先頭にカーソルを移動します。フィールドが複数指定されている場合は，最初のフィールドの先頭にカーソルを移動します。
[End]	カーソルの位置に関係なく，そのフィールドのデータをクリアします。 数字項目のフィールドは 0 でクリアされ，カーソルは表示されているデータの左端に位置づけられます。 数字編集項目のフィールドは 0 でクリアされ，カーソルは表示されているデータの左端または小数点上に位置づけられます。 その他の項目は空白文字でクリアされ，カーソルは表示されているデータの左端に位置づけられます。
[Enter]， [F1] ～ [F24]	入力が終了します。 この場合，[F13] ～ [F24] キーは，[Ctrl] キーを押したまま [F1] ～ [F12] キーを押すことを意味します。

注意事項

- フィールドの前方，後方とは画面上の次のような位置関係をいいます。



（凡例）　：現在カーソルが位置づけられているフィールド

12.2.3　LINE／COLUMN 句を使用した画面の座標指定

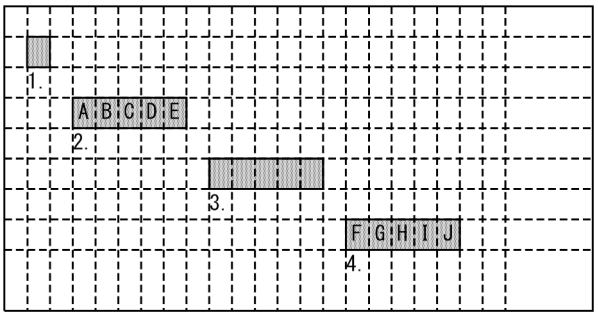
画面節（SCREEN SECTION）では，LINE／COLUMN 句を指定すると，画面項目に画面上での座標を与えられます。例を次に示します。

（例）

```
      :
SCREEN SECTION.
01 SC-11.
  02 SC-12 VALUE 'ABCDE'
      LINE PLUS 2 COLUMN PLUS 2.     …2.
  02 SC-13 PIC X(5) TO WC-11
      LINE PLUS 2 COLUMN PLUS 2.     …3.
  02 SC-14 VALUE 'FGHIJ'
      LINE PLUS 2 COLUMN PLUS 2.     …4.
    :
PROCEDURE DIVISION.
    DISPLAY SC-11 AT LINE 2 COLUMN 2. …1.
    :
```

1. 起点アドレス（2，2）が与えられます。
2. 起点からの相対値 2 のアドレスに画面項目が表示されます。
3. 入力画面項目なので表示されませんが，LINE／COLUMN 句の指定は有効です。
4. 前画面項目からの相対値 2 のアドレスに画面項目が表示されます。

実行結果は次のようになります。



12.2.4　CRT STATUS 句を使用したファンクションキー入力結果の取得

(1)　ファンクションキーとキー番号

CRT STATUS 句を指定した場合，ACCEPT 文を実行したときにファンクションキーを入力すると，各ファンクションキーに対応するキー番号が CRT STATUS 句で指定したデータの 2 バイト目に設定されます。

次にファンクションキーに対応するキー番号を示します。

ファンクションキー	割り当てられている キーの標準値	キー番号
[PF1]	[F1]	1
[PF2]	[F2]	2
[PF3]	[F3]	3
[PF4]	[F4]	4
[PF5]	[F5]	5
[PF6]	[F6]	6
[PF7]	[F7]	7
[PF8]	[F8]	8
[PF9]	[F9]	9
[PF10]	[F10]	10
[PF11]	[F11]	11
[PF12]	[F12]	12
[PF13]	[F13]	13
[PF14]	[F14]	14
[PF15]	[F15]	15
[PF16]	[F16]	16
[PF17]	[F17]	17
[PF18]	[F18]	18
[PF19]	[F19]	19
[PF20]	[F20]	20
[PF21]	[F21]	21
[PF22]	[F22]	22
[PF23]	[F23]	23

ファンクションキー	割り当てられている キーの標準値	キー番号
[PF24]	[F24]	24

注 1

デフォルトでは，[F24] まで使用できます。[F13] ～ [F24] キーは，[Ctrl] キーを押したまま [F1] ～ [F12] キーを押すことを意味します。

注 2

[F1] ～ [F24] キーが Motif のキーボード操作，およびその標準値としても割り当てられている場合，Motif の標準値が優先されます。Motif のキーボード操作，およびその標準値については，システムの Motif について記載されたマニュアルを参照してください。

CRT STATUS 句の詳細については，マニュアル「COBOL2002　言語　拡張仕様編」「12.1.4　CRT STATUS 句（SCREEN SECTION）」を参照してください。

12.2.5　注意事項

- COBOL では，画面機能を使用する場合，次のシグナルを登録します。
  - SIGCLD（子プロセス）

  COBOL 以外のプログラムで上記プロセスを登録した場合や，子プロセスを生成した場合，画面機能の動作は保証しません。
- 画面節（SCREEN SECTION）で画面を表示する場合，COBOL 専用の画面（cbl2002term）が表示されます。
- 画面節（SCREEN SECTION）による画面機能を使用する場合，setlocale または setlocale_r※システムコールを発行しないでください。発行した場合の結果は保証しません。

  注※

  setlocale_r は，setlocale のリエントラント対応関数です。
- 画面節（SCREEN SECTION）のプログラムを実行した場合に，何も応答がなくプロセスが終了することがあります。この場合，COBOL 専用の画面（cbl2002term）の初期化に失敗している可能性があります。次の要因が考えられますので確認してください。
  - X Window 上で実行していない。
  - 環境変数 PATH の設定内容に誤りがある。
- 指定したフィールドが画面（主画面のユーザ表示領域）に入りきらない場合，フィールドは入出力の対象になりません。次に示す対応を行い，画面に入るように調整してください。
  - フィールドの大きさを小さくする。

12.3　画面節（WINDOW SECTION）による画面機能

画面節（WINDOW SECTION）を使用した画面の入出力について説明します。

画面節（WINDOW SECTION）を使用した画面の入出力については，マニュアル「COBOL2002　言語　拡張仕様編」「13.　画面節（WINDOW SECTION）による画面機能」を参照してください。

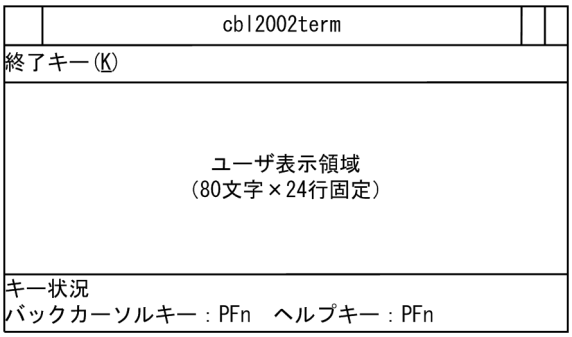
なお，SCREEN SECTION と WINDOW SECTION は，同時に使用できません。同時に使用した場合，エラーメッセージが出力されます。

12.3.1　画面の種類と構成

画面節（WINDOW SECTION）では，主画面，ポップアップ画面，ユーザポップアップ HELP 画面，およびエラー表示画面の 4 種類の画面を使用します。これらの画面について説明します。

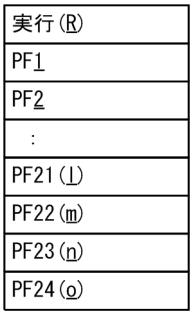
(1)　主画面

主画面は，画面の入出力で使用する主画面であり，タイトルバー，メニューバー，ステータスバー，ユーザ表示領域から構成されます。主画面の構成を次に示します。



ステータスバーには，SET 文の BACK CURSOR KEY 指定で割り当てたバックカーソルキー，および HELP KEY 句で割り当てたヘルプキーが，それぞれ表示されます。

メニューバーの［終了キー］をクリックすると，次のドロップダウンメニューが表示されます。これらをクリックすると，［Enter］キーや［F1］～［F24］キーを押したのと同じ結果が得られます。



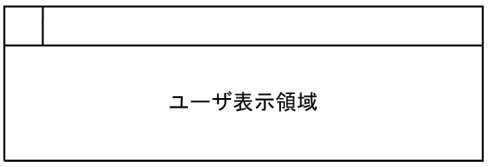
注

メニューは PF24 まで表示されます。

［PF13］～［PF24］キーは，［Ctrl］キーを押したまま［PF1］～［PF12］キーを押すことを意味します。

(2)　ポップアップ画面

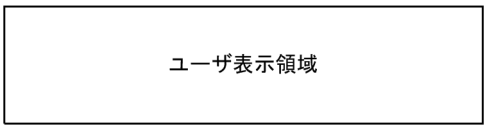
ポップアップ画面は，ポップアップ入出力機能で使用する画面であり，タイトルバーとユーザ表示領域から構成されます。ユーザ表示領域のサイズはプログラム中で指定します。ポップアップ画面の構成を次に示します。



ポップアップ画面の詳細については，「12.3.5　ポップアップ画面入出力機能」を参照してください。

(3)　ユーザポップアップ HELP 画面

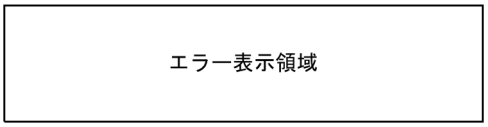
ユーザポップアップ HELP 画面は，ユーザポップアップ HELP 機能で使用する画面であり，ユーザ表示領域だけから構成されます。ユーザ表示領域のサイズはプログラム中で指定します。ユーザポップアップ HELP 画面の構成を次に示します。



ユーザポップアップ HELP 画面の詳細については，「12.3.6　ユーザポップアップ HELP 機能」を参照してください。

(4)　エラー表示画面

エラー表示画面は，誤ったデータを入力したときなどに自動的に表示される画面であり，エラー表示領域だけから構成されます。エラー表示画面の構成を次に示します。



12.3.2　データの表示形式

(1)　数字項目の表示形式

数字項目はフィールドに右詰めで表示されます。このとき，必要があれば左側に空白を補って表示されます。仮想小数点（V）の指定が右端以外にある場合は，その位置にピリオド（.）が表示されます。また，符号付き数字項目の場合は，先頭または末尾に符号（+，-）を付けて表示されます。

（例）

PICTURE句の指定	データの値	表示
9(5)V9(2)	123.4	□□123.40
9(6)V	123456	123456
S9(3)V9(2)	-12.34	□-12.34（左符号の場合）
S9(3)V9(2)	+12.34	□12.34+（右符号の場合）

(2)　その他の項目の表示形式

数字項目以外の場合は，データ項目の値がそのまま（左詰め）フィールドに表示されます。

12.3.3　データの入力方式

(1)　数字項目の入力方式

数字項目を入力する場合，整数部は数字を 1 けた入力するごとに全体の表示が左にシフトし，小数部は数字を 1 けた入力するごとにカーソルが右にシフトします。また，符号部はいつでも変更できます。

（例 1）

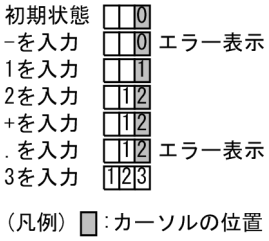
PIC S99V99 の場合

初期状態	□+0.00	
-を入力	□-0.00	
1を入力	□-1.00	
2を入力	-12.00	
3を入力	-12.00	エラー表示
.を入力	-12.00	
+を入力	+12.00	
4を入力	+12.40	
.を入力	+12.40	エラー表示

（凡例）□：カーソルの位置

（例 2）

PIC 999 の場合

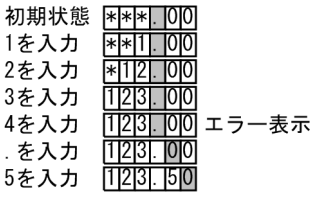


(2) 数字編集項目の入力方式

数字編集項目を入力する場合，キーを押すと同時に値が編集され，画面に反映されます。浮動挿入編集状態，ゼロ抑制編集状態では数字を 1 けた入力するごとに全体の表示が左にシフトし，ほかの編集状態では数字を 1 けた入力するごとにカーソルが右にシフトします。また，符号部はいつでも変更できます。

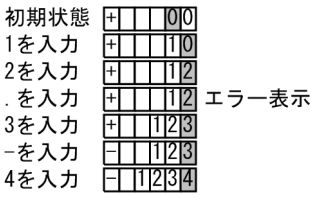
(例 1)

PIC ***.**の場合



(例 2)

PIC +ZZZ99 の場合

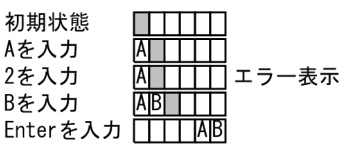


(3) 英字項目，英数字項目，日本語項目の入力方式

英字項目，英数字項目，日本語項目は，フィールドの左端から順次入力します。不当な文字を入力するとエラーが表示されます。RESET 句，JUST 句が指定されているときは，フィールドへの入力を終了すると，入力した文字が右詰めで再表示されます。

(例)

PIC AAAAAA JUST の場合

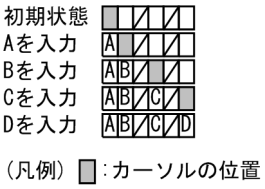


(4) 英数字編集項目，日本語編集項目の入力方式

英数字編集項目，日本語編集項目は，フィールドの左端から順次入力します。不当な文字を入力するとエラーが表示されます。キーを押すと同時に値が編集され，画面に反映されます。

(例)

PIC XX/X/X の場合



(凡例) ：カーソルの位置

(5) 注意事項

- 入力フィールドにカーソルを移動した場合，入力を助けるために，フィールドに初期表示がされます。この状態で何も入力しなければ，データ項目の内容は変更されません。また，何も入力しないでカーソルを別のフィールドに移動すると，フィールドの表示は元に戻ります。
- 入出力フィールドにカーソルを移動した場合，すでにデータが出力されているときはそのままですが，データが出力されていないときは入力フィールドと同様に初期表示がされます。この状態で何も入力しなければ，データ項目の内容は変更されません。また，何も入力しないでカーソルを別のフィールドに移動すると，フィールドの表示は元に戻ります。

12.3.4 キーの機能

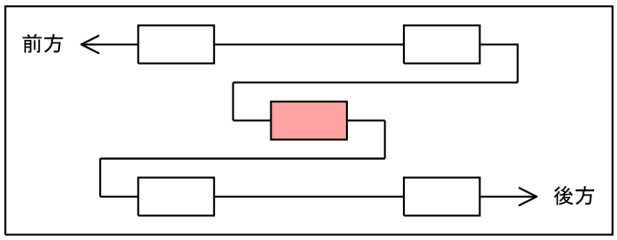
カーソルの移動などで使用するキーの意味を次に示します。

使用するキー	意味
[→]	カーソルが右に 1 カラム移動します。ただし，フィールドの右端のときは，後方のフィールドの先頭に移動します。
[←]	カーソルが左に 1 カラム移動します。ただし，フィールドの左端のときは，前方のフィールドの末尾に移動します。
[↑]	前方のフィールドの先頭にカーソルが移動します。
[↓]	後方のフィールドの先頭にカーソルが移動します。
[Tab]	後方のフィールドの先頭にカーソルが移動します。
[BackSpace]	現在カーソルが位置づけられているカラムの入力を取り消し，カーソルが左に 1 カラム移動します。
[End]	カーソルの位置に関係なく，そのフィールドのデータをクリアします。 数字項目のフィールドは 0 でクリアされ，カーソルは整数一の位に位置づけられます。ただし整数項目がない場合，小数第一位に位置づけられます。

使用するキー	意味
	数字編集項目のフィールドは 0 でクリアされ，カーソルは表示されているデータの左端または小数点上に位置づけられます。 その他の項目は空白文字でクリアされ，カーソルは表示されているデータの左端に位置づけられます。
[Enter]， [F1] ～ [F24]	入力が終了します。 この場合，[F13] ～ [F24] キーは，[Ctrl] キーを押したまま [F1] ～ [F12] キーを押すことを意味します。

注意事項

- フィールドの前方，後方とは画面上の次のような位置関係をいいます。

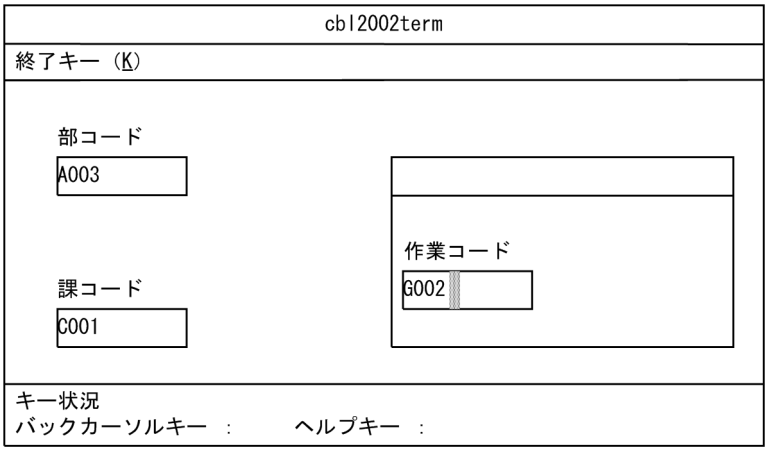


（凡例） ：現在カーソルが位置づけられているフィールド

- 前方または後方のフィールドがない場合は，末尾または先頭のフィールドにカーソルが移動します。

12.3.5 ポップアップ画面入出力機能

ポップアップ画面入出力機能は，主画面とは別にもう一つの画面（ポップアップ画面）を表示し，この画面からデータを入出力するときに使用します。ポップアップ画面の例を，次に示します。



ポップアップ画面は，プログラム中の REPLY 文が実行されると表示されます。ポップアップ画面のフィールドへの入力などが終わり，終了キー（[Enter] キー，[F1] ～ [F24] キー）を押すと，ポップアップ画面が閉じます。

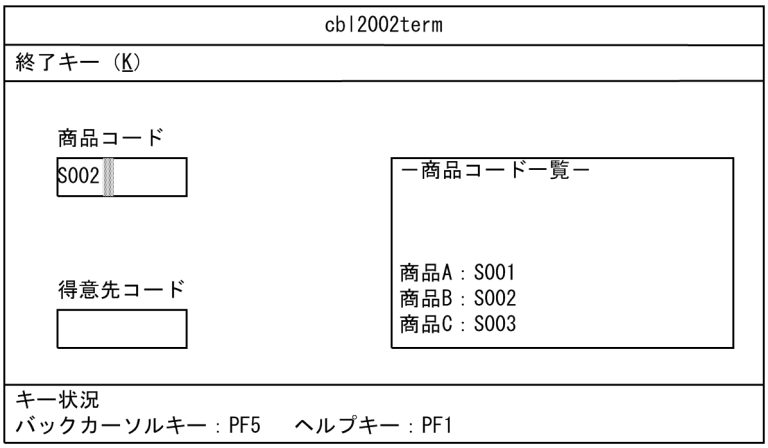
注意事項

- HELP LINE 指定，HELP COLUMN 指定で指定する表示位置は，ポップアップ画面（タイトルバーを含む）の左隅上です。
- ポップアップ画面が表示されているときは，主画面のタイトルバーのメニューは使用できません。

12.3.6　ユーザポップアップ HELP 機能

ユーザポップアップ HELP 機能は，商品コードなどを確認しながら画面に入力するために，一時的にこれらのコードの一覧をガイダンス画面に表示するような場合に使用します。このとき表示するガイダンス画面をユーザポップアップ HELP 画面といいます。ユーザポップアップ HELP 画面やこの画面を表示するためのキーは画面節（WINDOW SECTION）の HELP WINDOW 句で指定します。

ユーザポップアップ HELP 画面の操作手順の例を次に示します。



1. カーソルをフィールド（商品コード）に位置づけてヘルプキー（[PF1］キー）を押すと，対応するユーザポップアップ HELP 画面が主画面に重ねて表示されます。
2. ユーザポップアップ HELP 画面を見ながら商品コードを入力します。
3. カーソルをほかのフィールド（商品コード以外）に移すか，終了キーを押すと，ユーザポップアップ HELP 画面が閉じます。

注意事項

- ここでいう終了キーとは，メニューバーの［終了キー］で表示される［Enter］キー，［F1］～［F24］キーコマンド，およびこれらに対応する［Enter］キー，［F1］～［F24］キーを指します。

12.3.7　注意事項

(1)　ACCEPT／REPLY 文を使用する場合

ACCEPT／REPLY 文で FIRST FIELD 指定を使用する場合，次のような制限があります。

- -DebugCompati オプションを指定しても，FIRST FIELD 指定の一意名に対して添字はチェックされません。
- FIRST FIELD 指定の直後には，IN NUMERIC MODE 指定を記述できません。ただし，IN を省略した場合は，IN NUMERIC MODE 指定を記述できます。

(2)　olwn マネージャを使用する場合

olwn マネージャの画面でダブルクリックイベントを取得する場合は，「$HOME/.Xdefaults」ファイルを次に示すように変更してください。

(修正内容)

変更前

```
OpenWindows.SetInput:select
```

変更後

```
OpenWindows.SetInput:followmouse
```

(3)　Motif のキーボード操作，およびその標準値に関する注意事項

［F1］～［F24］キーが Motif のキーボード操作，およびその標準値としても割り当てられている場合，Motif の標準値が優先されます。Motif のキーボード操作，およびその標準値については，システムの Motif について記載されたマニュアルを参照してください。

(4)　指定したフィールドが画面（主画面のユーザ表示領域）に入りきらない場合

指定したフィールドが画面（主画面のユーザ表示領域）に入りきらない場合，実行時エラーとなります。次に示す対応を行い，画面に入るように調整してください。

- フィールドの大きさを小さくする。

(5)　その他の注意事項

- COBOL では，画面機能を使用する場合，次のシグナルを登録します。
  - SIGCLD（子プロセス）

COBOL 以外のプログラムで上記プロセスを登録した場合や，子プロセスを生成した場合，画面機能の動作は保証しません。

- 画面節（WINDOW SECTION）による画面機能を使用する場合，setlocale または setlocale_r※システムコールを発行しないでください。発行した場合の結果は保証しません。

  注※

  setlocale_r は，setlocale のリエントラント対応関数です。

- 画面節（WINDOW SECTION）のプログラムを実行した場合に，何も応答がなくプロセスが終了することがあります。この場合，COBOL 専用の画面（cbl2002term）の初期化に失敗している可能性があります。次の要因が考えられますので確認してください。
  - X Window 上で実行していない。
  - 環境変数 PATH の設定内容に誤りがある。

12.4　リソース一覧

画面節（SCREEN SECTION および WINDOW SECTION）による画面機能では，主画面の動作環境を定義するためのリソースの標準値が，アプリケーションリソースファイルに格納されています。このアプリケーションリソースファイル中のリソース値を設定すると，主画面のメニューバーの［終了キー］に表示されるファンクションキーなどを変更できます。

アプリケーションリソースファイル名

AIX(32)の場合：Cbl2002term

AIX(64)の場合：Cbl2002term64

アプリケーションリソースファイルのクラス名

AIX(32)の場合：Cbl2002term

AIX(64)の場合：Cbl2002term64

アプリケーションリソースファイルのキャプション文字列

AIX(32)の場合：cbl2002term

AIX(64)の場合：cbl2002term 64bit

リソースファイルの格納場所

リソースファイルは/usr 下に格納します。

形式

AIX(32)の場合

```
Cbl2002term*リソース名:リソース値
```

AIX(64)の場合

```
Cbl2002term64*リソース名:リソース値
```

画面節（SCREEN SECTION および WINDOW SECTION）による画面機能でのリソース一覧を次に示します。

リソース名	内容	リソース値の 標準値
background	背景色	white
foreground	前景色	black
highlightColor	高輝度前景色	red
showPF1	[終了キー]メニューに[PF1]キーを表示	True
showPF2	[終了キー]メニューに[PF2]キーを表示	True
showPF3	[終了キー]メニューに[PF3]キーを表示	True
showPF4	[終了キー]メニューに[PF4]キーを表示	True

リソース名	内容	リソース値の 標準値
showPF5	[終了キー]メニューに[PF5]キーを表示	True
showPF6	[終了キー]メニューに[PF6]キーを表示	True
showPF7	[終了キー]メニューに[PF7]キーを表示	True
showPF8	[終了キー]メニューに[PF8]キーを表示	True
showPF9	[終了キー]メニューに[PF9]キーを表示	True
showPF10	[終了キー]メニューに[PF10]キーを表示	True
showPF11	[終了キー]メニューに[PF11]キーを表示	True
showPF12	[終了キー]メニューに[PF12]キーを表示	True
showPF13	[終了キー]メニューに[PF13]キーを表示	True
showPF14	[終了キー]メニューに[PF14]キーを表示	True
showPF15	[終了キー]メニューに[PF15]キーを表示	True
showPF16	[終了キー]メニューに[PF16]キーを表示	True
showPF17	[終了キー]メニューに[PF17]キーを表示	True
showPF18	[終了キー]メニューに[PF18]キーを表示	True
showPF19	[終了キー]メニューに[PF19]キーを表示	True
showPF20	[終了キー]メニューに[PF20]キーを表示	True
showPF21	[終了キー]メニューに[PF21]キーを表示	True
showPF22	[終了キー]メニューに[PF22]キーを表示	True
showPF23	[終了キー]メニューに[PF23]キーを表示	True
showPF24	[終了キー]メニューに[PF24]キーを表示	True
PF1.labelString	[PF1]キーのラベル文字列	PF1
PF2.labelString	[PF2]キーのラベル文字列	PF2
PF3.labelString	[PF3]キーのラベル文字列	PF3
PF4.labelString	[PF4]キーのラベル文字列	PF4
PF5.labelString	[PF5]キーのラベル文字列	PF5
PF6.labelString	[PF6]キーのラベル文字列	PF6
PF7.labelString	[PF7]キーのラベル文字列	PF7
PF8.labelString	[PF8]キーのラベル文字列	PF8
PF9.labelString	[PF9]キーのラベル文字列	PF9
PF10.labelString	[PF10]キーのラベル文字列	PF10(a)

リソース名	内容	リソース値の 標準値
PF11.labelString	[PF11]キーのラベル文字列	PF11(b)
PF12.labelString	[PF12]キーのラベル文字列	PF12(c)
PF13.labelString	[PF13]キーのラベル文字列	PF13(d)
PF14.labelString	[PF14]キーのラベル文字列	PF14(e)
PF15.labelString	[PF15]キーのラベル文字列	PF15(f)
PF16.labelString	[PF16]キーのラベル文字列	PF16(g)
PF17.labelString	[PF17]キーのラベル文字列	PF17(h)
PF18.labelString	[PF18]キーのラベル文字列	PF18(i)
PF19.labelString	[PF19]キーのラベル文字列	PF19(j)
PF20.labelString	[PF20]キーのラベル文字列	PF20(k)
PF21.labelString	[PF21]キーのラベル文字列	PF21(l)
PF22.labelString	[PF22]キーのラベル文字列	PF22(m)
PF23.labelString	[PF23]キーのラベル文字列	PF23(n)
PF24.labelString	[PF24]キーのラベル文字列	PF24(o)

13 COBOL 入出力サービスルーチン

COBOL 入出力サービスルーチンを使うと，ほかの言語で書かれたプログラムから，COBOL2002 で作成した順編成ファイル，および相対編成ファイルにアクセスできます。

この章では，COBOL 入出力サービスルーチンの使用方法について説明します。

13.1 COBOL 入出力サービスルーチンの概要

13.1.1 概要

COBOL2002 の順編成ファイル，および相対編成ファイルは，COBOL2002 独自のファイル形式で作成されます。そのため，他言語で作成したプログラムから，直接これらの形式のファイルにはアクセスできません。

COBOL 入出力サービスルーチンを使用すると，他言語のプログラムからでも COBOL2002 の順編成ファイル，および相対編成ファイルにアクセスできます。

(1) COBOL 入出力サービスルーチンの使用方法

COBOL 入出力サービスルーチンは，COBOL プログラム，または他言語のプログラムから呼び出して使用します。ファイルの入出力に必要な情報は，インタフェース領域と呼ばれる構造体を通じて受け渡されます。各サービスルーチンを呼び出すときには，このインタフェース領域のアドレスを引数として渡します。

(2) COBOL 入出力サービスルーチンで発生したエラーのデバッグ方法

COBOL 入出力サービスルーチンの実行中にエラーが発生した場合，エラーメッセージ番号がインタフェース領域に格納されます。エラーメッセージは出力されません。

また，環境変数 CBL_FLSRVDUMP を設定すると，デバッグ情報をファイルに出力することもできます。

13.1.2 COBOL 入出力サービスルーチンが対応している機能

(1) サービスルーチンが対応しているファイル形式

COBOL 入出力サービスルーチンでアクセスできるファイル形式を，次に示します。

- 固定長の順編成ファイル
- 可変長の順編成ファイル
- 固定長の相対編成ファイル
- 可変長の相対編成ファイル

COBOL 入出力サービスルーチンを使うと，これらの形式のファイルに対して COBOL2002 の入出力機能，およびファイル共用が使用できます。また，COBOL プログラムからアクセスすることを考慮したエラーチェックも実行されます。

(2) COBOL 入出力サービスルーチンの制限事項

- COBOL 入出力サービスルーチンを使ってファイルの入出力をする場合，次の指定，および機能は使用できません。
  - CLOSE 文の WITH LOCK 指定
  - 不定ファイル（SELECT 句の OPTIONAL 指定）
  - 入出力で誤り状態が発生したときの制御移行（USE 文，AT END 指定など）
  - 行制御機能（WRITE 文の ADVANCING 指定，ファイル記述項の LINAGE 句）
  - 特殊ファイルへの入出力（テープ装置，プリンタなど）
  - マルチスレッド環境下での実行
  - ラージファイル入出力機能（ただし，順編成ファイルでは，COBOL 入出力サービスルーチンを使用してラージファイルに対する入出力ができます）
- COBOL 入出力サービスルーチンでは，エラーが発生しても終了処理がされません。エラーが発生した場合は，COBOL 入出力サービスルーチンを呼び出したプログラム側で終了処理するようにしてください。

13.2 COBOL 入出力サービスルーチンの説明

COBOL 入出力サービスルーチンの一覧を，次に示します。

表 13–1 COBOL 入出力サービスルーチンの一覧

サービスルーチンの名称	順ファイルの入出力	相対ファイルの入出力	機能
CBLOPEN	○		ファイルを開く
CBLCLOSE	○		ファイルを閉じる
CBLREAD	○		レコードの入力
CBLWRITE	○		レコードの出力
CBLREWRITE	○		レコードの書き換え
CBLUNLOCK	○		レコード施錠の解除
CBLDELETE	×	○	レコードの削除
CBLSTART	×	○	レコードの位置づけ
CBLWDISK	○		ファイルのディスクへの書き込み保証

（凡例）

○：使用できる

×：使用できない

各サービスルーチンの処理概要を，次に示します。

CBLOPEN サービスルーチン

- OPEN 文のモード，ファイル属性に従い，ファイルを開く。
- 書き込みモードの場合，ファイルを作成する。ファイルがすでにある場合は，既存のファイルに上書きする。
- 追加書き込みモードの場合，ファイルの終端にファイルポインタを移動する。
- ファイルの排他・共用のための施錠要求をする。
- 内部的に必要な領域を確保する。

CBLCLOSE サービスルーチン

- ファイルを閉じる。
- レコードの施錠を解除する。
- オープン時に確保した領域を解放する。

CBLREAD サービスルーチン

- レコードをバッファに読み込む。

- 読み込んだレコードの長さを設定する。
- レコード施錠の指定に従い，レコードを施錠する，または施錠を解除する。
- 読み込んだレコードの相対レコード番号を設定する（相対ファイルだけ）。

CBLWRITE サービスルーチン

- バッファのレコードを書き出す。
- レコードの施錠を解除する。
- 書き込んだレコードの相対番号を返す（相対ファイルだけ）。

CBLREWRITE サービスルーチン

- 書き込みバッファのレコードとの書き換えをする。
- 直前の READ 文をチェックする（順アクセスの場合だけ）。
- 書き換えレコード長をチェックする。
- レコードの施錠を解除する。

CBLUNLOCK サービスルーチン

- すべてのレコード施錠を解除する。

CBLDELETE サービスルーチン

- レコードを削除する。
- 直前の READ 文をチェックする（順アクセスの場合だけ）。
- レコードの施錠を解除する。

CBLSTART サービスルーチン

- キー番号と条件からレコードを位置づける。
- ファイルのアクセスモードをチェックする（乱アクセスの場合はできない）。

CBLWDISK サービスルーチン

- ファイルに出力したデータの，ディスクへの書き込み保証を適用する。

COBOL 入出力サービスルーチンを呼び出すときの注意事項

COBOL2002 のファイル入出力機能と同様に，オープン時の OPEN 文のモードによって実行できないサービスルーチンがあります。

13.3 COBOL 入出力サービスルーチンのインタフェース

13.3.1 サービスルーチンを呼び出す関数の形式

COBOL 入出力サービスルーチンは，すべて次の形式で呼び出します。

形式

```
#include ”CBL85fl.h”
int  サービスルーチン名(CBLCOMFL * com, CBLPARMFL * parm)
```

引数

CBLCOMFL * com：管理情報インタフェース領域のアドレス

CBLPARMFL * parm：パラメタインタフェース領域のアドレス

戻り値

0：正常に終了した場合

-1：エラーが発生した場合

注意事項

- 管理情報インタフェース領域，およびパラメタインタフェース領域は，ユーザプログラム側で確保する必要があります。
- 各インタフェース領域中のシステムが使用する領域，および予備領域には，最初の CBLOPEN サービスルーチンを実行する前に NULL（X'00'）を指定して，領域をクリアしておく必要があります。

  ただし，一度 CBLCLOSE サービスルーチンで入出力を終了したあと，再度同じインタフェース領域を使って CBLOPEN サービスルーチンで入出力を開始する場合は，管理情報インタフェース領域中のシステムが使用する領域，および予備領域をクリアする必要はありません。
- 同じファイルにアクセスするときには，CBLOPEN サービスルーチンでファイルを開いてから CBLCLOSE サービスルーチンでファイルを閉じるまで，同じ管理情報インタフェース領域を使用する必要があります。
- 同時に複数のファイルにアクセスする場合は，各ファイルに対して別々の管理情報インタフェース領域を用意する必要があります。
- 管理情報インタフェース領域中のファイル情報は，CBLOPEN サービスルーチンを実行したときの設定値が有効となります。ファイルを開いたあとの入出力サービスルーチンでファイル情報を変更しても無効です。ただし，デバッグ情報出力指示の設定は，ファイルを開いたあとでも有効となります。

13.3.2 インタフェース領域の形式

COBOL 入出力サービスルーチンで使用するインタフェース領域を，次に示します。

表 13–2　COBOL 入出力サービスルーチンで使用するインタフェース領域

インタフェース領域の名称 (構造体の typedef 名称)	サイズ（バイト）		用途
	AIX(32)，Linux(x86)の場合	AIX(64)，Linux(x64)の場合	
管理情報インタフェース領域 (CBLCOMFL)	256	312	• ファイル情報を管理する • OPEN 時のファイル情報を指定する • エラー発生時，エラー情報を指定する • COBOL2002 の内部情報として使用される
パラメタインタフェース領域 (CBLPARMFL)	32	48	• 入出力パラメタを指定する

(1)　管理情報インタフェース領域

管理情報インタフェース領域の形式を，「表 13–3　管理情報インタフェース領域の形式（1）」，および「表 13–4　管理情報インタフェース領域の形式（2）」に示します。

表 13–3　管理情報インタフェース領域の形式（1）

データ項目名		AIX(32)，Linux(x86)の場合			AIX(64)，Linux(x64)の場合			設定／参照する値	CBLOPEN サービスルーチンでの指定		
区分	名称	位置	長さ	データ形式	位置	長さ	データ形式		OUTPUT 指定	OUTPUT 以外	
										属性チェック無	属性チェック有
ファイル情報	ファイル編成とレコード形式	0	4	char[4]	0	4	char[4]	SAMF：順ファイル固定長レコード形式 SAMV：順ファイル可変長レコード形式 RELF：相対ファイル固定長レコード形式 RELV：相対ファイル可変長レコード形式	○	△※1	○
	最大レコード長	4	4	int	4	4	int	レコード長 (固定長形式の場合レコード長)	○	△※1	○
	最小レ	8	4	int	8	4	int	レコード長 (固定長形式の場合は無視される)	○	△	○

データ項目名		AIX(32), Linux(x86)の場合			AIX(64), Linux(x64)の場合			設定／参照する値	CBLOPEN サービスルーチンでの指定		
区分	名称	位置	長さ	データ形式	位置	長さ	データ形式		OUTPUT 指定	OUTPUT 以外	
										属性チェック無	属性チェック有
	コード長										
	未使用	−	−	−	12	4	char[4]	サービスルーチン予備領域	NULL（X'00'）を指定してクリアしておくこと	NULL（X'00'）を指定してクリアしておくこと	NULL（X'00'）を指定してクリアしておくこと
	ファイル名称アドレス	12	4	char*	16	8	char*	オープンするファイル名称のアドレスファイル名称は NULL で終わる文字列とする。 CBLOPEN サービスルーチンを呼び出す時だけ指定が必要。 CBLOPEN サービスルーチンの呼び出しが成功したあと，CBLCLOSE サービスルーチン実行までの入出力文で，COBOL2002 は，このアドレス，およびアドレスが指す領域を参照しない。	○	○	○
	属性チェックオプション※2	16	1	char	24	1	char	CBLCOM_NOCHK： 管理情報インタフェース領域とファイル実体の属性をチェックしない。 CBLCOM_CHK： 管理情報インタフェース領域とファイル実体の属性をチェックする。	−	○ CBLCOM_NOCHK	○ CBLCOM_CHK
	ファイル OPEN	17	1	char	25	1	char	CBLCOM_OPEN_INPUT： 読み取り専用（OPEN INPUT）	○ CBLCOM_OPEN_OUTPUT	○	○

データ項目名		AIX(32), Linux(x86)の場合			AIX(64), Linux(x64)の場合			設定／参照する値	CBLOPEN サービスルーチンでの指定		
区分	名称	位置	長さ	データ形式	位置	長さ	データ形式		OUTPUT 指定	OUTPUT 以外	
										属性チェック無	属性チェック有
	文のモード							CBLCOM_OPEN_IO： 読み取りと書き込み（OPEN I-O） CBLCOM_OPEN_OUTPUT： 書き込み専用・新規作成（OPEN OUTPUT） CBLCOM_OPEN_EXTEND： 追加書き（OPEN EXTEND）			
	ファイル施錠モード	18	1	char	26	1	char	CBLCOM_LOCK_NO： LOCK MODE 句指定なし CBLCOM_LOCK_EXCL： 排他施錠（LOCK MODE IS EXCLUSIVE） CBLCOM_LOCK_AUTO： 自動施錠（LOCK MODE IS AUTOMATIC） CBLCOM_LOCK_MANU： 手動施錠（LOCK MODE IS MANUAL）※3	○	○	○
	ディスク書き込みオプション	19	1	char	27	1	char	CBLCOM_ENVFSYNC： 環境変数 CBLFSYNC に従う。	○	○ （読み取り専用では無視）	○ （読み取り専用では無視）

データ項目名		AIX(32), Linux(x86)の場合			AIX(64), Linux(x64)の場合			設定／参照する値	CBLOPEN サービスルーチンでの指定		
区分	名称	位置	長さ	データ形式	位置	長さ	データ形式		OUTPUT 指定	OUTPUT 以外	
										属性チェック無	属性チェック有
								CBLCOM_NOFSYNC： ディスク書き込み保証を適用しない。 CBLCOM_FSYNC： クローズ時のディスク書き込み保証を適用する。 CBLCOM_WDISK： 書き込み時のディスク書き込み保証を適用する。 詳細は「13.6 COBOL 入出力サービスルーチンでのディスク書き込み保証」を参照。			
	ファイルアクセスモード	20	1	char	28	1	char	CBLCOM_ACCESS_SEQ： 順アクセス法（ACCESS MODE IS SEQUENTIAL） CBLCOM_ACCESS_RAND： 乱アクセス法（ACCESS MODE IS RANDOM） CBLCOM_ACCESS_DYNA： 動的アクセス法（ACCESS MODE IS DYNAMIC） 順ファイルでは順アクセス法だけが指定できる。	○	○	○

データ項目名		AIX(32), Linux(x86)の場合			AIX(64), Linux(x64)の場合			設定／参照する値	CBLOPEN サービスルーチンでの指定		
区分	名称	位置	長さ	データ形式	位置	長さ	データ形式		OUTPUT 指定	OUTPUT 以外	
										属性チェック無	属性チェック有
	ラージファイル対応オプション	21	1	char	29	1	char	CBLCOM_ENVLARGEFILE： 環境変数 CBLLARGEFILE の指定に従う（順編成ファイルだけ）。 CBLCOM_NOLARGEFILE： 環境変数 CBLLARGEFILE の指定を有効としない（順編成ファイルだけ）。 CBLCOM_LARGEFILE： ラージファイル入出力機能を使用する（順編成ファイルだけ）。	○	○	○
	未使用	22	6	char[6]	30	10	char[10]	サービスルーチン予備領域	○ NULL（X'00'）を指定してクリアしておくこと	○ NULL（X'00'）を指定してクリアしておくこと	○ NULL（X'00'）を指定してクリアしておくこと

（凡例）

○：必ず指定する項目

△：COBOL2002 が値を返す項目

－：無視される項目

注

表中の「CBLCOM_xxx」は，COBOL2002 で使用できるインクルードファイル（CBL85fl.h）中で定義されているマクロです。CBL85fl.h ファイルは，次のディレクトリの下に提供されています。

AIX(32)，Linux(x86)の場合

```
/opt/HILNGcbl2k/include/cbl
```

AIX(64)，Linux(x64)の場合

```
/opt/HILNGcbl2k64/include/cbl
```

注※1

順固定長ファイルに対して入出力する場合は，必ず指定してください。

注※2

順固定長ファイルではこの指定は無視され，レコード長に従います。

注※3

順ファイルでは手動施錠を指定できません。

表 13-4　管理情報インタフェース領域の形式（2）

データ項目名		AIX(32)，Linux(x86)の場合			AIX(64)，Linux(x64)の場合			設定／参照する値	備考
区分	名称	位置	長さ	データ形式	位置	長さ	データ形式		
エラー情報	入出力状態(FILE STATUS)	28	2	short	40	2	short	0～99	正常／エラーに関係なく，必ず値が返される。 返される値の詳細は「付録 G 入出力状態の値」を参照
	システムエラー情報有無	30	1	char	42	1	char	CBLCOM_ERR_NOSYSTEM： システムエラー情報なし CBLCOM_ERR_SYSTEM： システムエラー情報有り	
	未使用	31	1	char	43	1	char	サービスルーチン予備領域	
	COBOLメッセージ番号	32	2	short	44	2	short	3001～4099 COBOL エラーメッセージ番号	「13.5.1 COBOL 入出力サービスルーチンで出力されるエラーメッセージ番号」を参照
	システムコール番号	34	2	short	46	2	short	1～999 エラーとなったシステムコールを表す COBOL2002 が定める関数番号	システムエラー情報有りの場合に有効。 番号と内容については，マニュアル「COBOL2002　メッセージ」を参照

データ項目名		AIX(32)，Linux(x86)の場合			AIX(64)，Linux(x64)の場合			設定／参照する値	備考
区分	名称	位置	長さ	データ形式	位置	長さ	データ形式		
	システムエラーコード	36	4	int	48	4	int	システムが返すエラーコード	システムエラー情報有りの場合に有効。 システムのマニュアルを参照
	COBOL エラー詳細情報	40	1	char	52	1	char	インタフェース領域指定誤り，およびファイル属性エラー発生時の詳細情報	COBOL メッセージ番号が 3701～3703，および 3801～3803 の場合に有効。 設定される値は，「表 13-5 COBOL エラー詳細情報の一覧」を参照
	未使用	41	15	char[15]	53	15	char[15]	サービスルーチン予備領域	
デバッグ情報	デバッグ情報出力指示	56	1	char	68	1	char	CBLCOM_DBG_NO： デバッグ情報を出力しない CBLCOM_DBG_BEF： 各入出力サービスルーチン実行の前にデバッグ情報を出力する CBLCOM_DBG_AFT： 各入出力サービスルーチン実行後にデバッグ情報を出力する CBLCOM_DBG_BEFAFT： 各入出力サービスルーチン実行の前とあとでデバッグ情報を出力する	「13.5.2 インタフェース領域のダンプ出力」を参照
	未使用	57	11	char[11]	69	11	char[11]	サービスルーチン予備領域	
	ユーザ領域	68	16	char[16]	80	16	char[16]	プログラムで使用可能な領域。 COBOL2002 では管理しない	
	システムで使用する領域	84	172	char[132] char *[10]	96	216	char[136] char *[10]	システムで使用する領域。 最初のオープンを実行する前に NULL（X'00'）を指定してクリアしておく	NULL（X'00'）クリア以降，この領域を更新してはならない

注

表中の「CBLCOM_xxx」は，COBOL2002 で使用できるインクルードファイル（CBL85fl.h）中で定義されているマクロです。CBL85fl.h ファイルは，次のディレクトリの下に提供されています。

AIX(32)，Linux(x86)の場合

```
/opt/HILNGcbl2k/include/cbl
```

Linux(x64)の場合

```
/opt/HILNGcbl2k64/include/cbl
```

管理情報インタフェース領域のデータ項目に関する注意事項

(a)属性チェックオプション

属性チェックオプションには，ファイルのオープン時に管理情報インタフェース領域の値とファイル実体の属性をチェックするかどうかを指定します。

(i)属性をチェックしない場合

属性チェックオプションに CBLCOM_NOCHK を指定すると，ファイルの属性がチェックされません。この場合，CBLOPEN サービスルーチンの呼び出し成功時に，ファイル実体に基づいて次のデータ項目が自動的に設定されます。

- ファイル形式

  順ファイル可変長レコード形式の場合　：SAMV

  相対ファイル固定長レコード形式の場合：RELF

  相対ファイル可変長レコード形式の場合：RELV
- 最大レコード長
- 最小レコード長
- 入出力状態

  00 が返されます。

ユーザは，設定されたデータ項目の値から，入出力に必要なバッファ領域を用意するようなプログラムを作成する必要があります。また，ファイル形式が妥当かどうかをユーザプログラム側で必ず判定するようにしてください。

(ii)属性をチェックする場合

属性チェックオプションに CBLCOM_CHK を指定すると，次のデータ項目についてファイルの属性がチェックされます。

- ファイル編成とレコード形式
- 最大レコード長
- 最小レコード長

また，CBLOPEN サービスルーチンの呼び出し成功時に，入出力状態に 00 が返されます。

属性チェックの結果，管理情報インタフェース領域の設定とファイル実体の属性が一致しなかった場合，サービスルーチンはエラーの戻り値を返し，ファイルは開かれません。

注意事項

- OPEN 文のモードに「書き込み専用・新規作成（OPEN_OUTPUT）」を指定した場合，属性チェックオプションの指定は無視され，管理情報インタフェース領域に指定されたファイル編成，およびレコード長に従って新規にファイルが作成されます。
- ファイル編成とレコード形式に SAMF（順ファイル固定長レコード形式）を指定した場合，属性チェックオプションの指定は無視され，指定したレコード長に従ってファイルの入出力処理が実行されます。
- ファイル編成とレコード形式に順ファイル固定長レコード形式（SAMF）を指定しないで，かつ，属性チェックオプションにチェックしない（CBLCOM_NOCHK）を指定した場合，順固定長ファイルに対して CBLOPEN サービスルーチンを実行すると，対象外ファイルとみなされます。

(b)エラー情報

エラー情報に設定される値は，サービスルーチンの終了状態によって次のように異なります。

サービスルーチンの終了状態	入出力状態	COBOL メッセージ番号	システムエラー情報有無	システムコール番号／システムエラーコード
正常終了	00	0	なし （CBLCOM_ERR_NOSYSTEM）	0
エラー発生	入出力状態を表す値	メッセージ番号 （0 以外）	なし （CBLCOM_ERR_NOSYSTEM）	0
			あり （CBLCOM_ERR_SYSTEM）	該当するコードが設定される

COBOL メッセージ番号，システムコール番号，およびシステムエラーコードの値は，COBOL2002 が独自に定めた値，またはシステム固有のコードです。そのため，エラー発生時のプログラム実行制御には，入出力状態の値を使用してください。

(c)COBOL エラー詳細情報

COBOL エラー詳細情報に返される情報，およびその意味を次に示します。

表 13-5　COBOL エラー詳細情報の一覧

COBOL メッセージ番号	エラー種別	マクロ名称	値	エラー項目／エラーの意味
3701 3801	管理情報インタフェース領域指定誤り	CBLCOM_ERRC_FORM	1	ファイル編成とレコード形式
		CBLCOM_ERRC_MAX	2	最大レコード長 （最大レコード長の値が 1～1,073,741,799 以外，または最小レコード長より小さい）

COBOL メッセージ番号	エラー種別	マクロ名称	値	エラー項目／エラーの意味
		CBLCOM_ERRC_MIN	3	最小レコード長 （最小レコード長の値が 1～1,073,741,799 以外）
		CBLCOM_ERRC_PATH	4	ファイル名称アドレス
		CBLCOM_ERRC_FLOPT	5	属性チェックオプション
		CBLCOM_ERRC_OPEN	6	ファイルオープンモード
		CBLCOM_ERRC_LOCK	7	ファイル施錠モード
		CBLCOM_ERRC_FSYNC	8	ディスク書き込みオプション
		CBLCOM_ERRC_DBGINF	9	デバッグ情報出力指示
		CBLCOM_ERRC_ACCESS	10	ファイルアクセスモード
		CBLCOM_ERRC_MSGBUFSIZ	11	メッセージ出力用バッファ長
		CBLCOM_ERRC_LARGEFILE	12	ラージファイル対応オプション
3702 3802	パラメタインタフェース領域指定誤り	CBLCOM_ERRP_RDBUF	21	読み込みバッファアドレス
		CBLCOM_ERRP_WRBUF	22	書き出しバッファアドレス
		CBLCOM_ERRP_LOCK	23	WITH LOCK 指定
		CBLCOM_ERRP_KEY	24	レコード番号キー
		CBLCOM_ERRP_ACCESS	25	レコード単位アクセスモード
		CBLCOM_ERRP_KEYMODE	26	キー比較条件
3703 3803	ファイル属性エラー	CBLCOM_ERRF_CBL	41	ファイルフォーマット （COBOL 以外で作成したファイル，またはファイル破壊）
		CBLCOM_ERRF_FORM	42	ファイル形式（ファイル編成，またはレコード形式）
		CBLCOM_ERRF_MAX	43	最大レコード長
		CBLCOM_ERRF_MIN	44	最小レコード長

インタフェース領域中の複数のデータ項目に誤りがある場合や，ファイルの属性チェックで複数の項目の属性が一致しなかった場合，COBOL エラー詳細情報には，最初に発見されたエラー項目に関する値が設定されます。

(2) パラメタインタフェース領域

パラメタインタフェース領域の形式を，次に示します。

表 13-6　パラメタインタフェース領域の形式

データ項目名		AIX(32), Linux(x86)の場合			AIX(64), Linux(x64)の場合			設定／参照する値		各サービスルーチンでの指定								
区分	名称および意味	位置	長さ	データ形式	位置	長さ	データ形式			OP	CL	RE	WR	RW	DE	ST	UN	WD
入出力パラメタ	読み込みバッファアドレス	0	4	char *	0	8	char *	読み込んだレコードを格納する領域のアドレス		−	−	○	−	−	−	−	−	−
	読み込みサイズ	4	4	int	8	4	int	読み込んだレコードの長さを返す		−	−	△	−	−	−	−	−	−
	未使用	−	−	−	12	4	char[4]	サービスルーチン予備領域		−	−	−	−	−	−	−	−	−
	書き出しバッファアドレス	8	4	char *	16		char *	書き出すレコードのバッファアドレス		−	−	−	○	○	−	−	−	−
	書き出しサイズ	12	4	int	24	4	int	書き出すレコードの長さ		−	−	−	○※1	○※1	−	−	−	−
	相対レコード番号キー(RELATIVE KEY	16	4	int	28	4	int	入出力対象レコードキー番号※2	ファイルアクセスモード：順アクセス	−	−	△	△	N	N	○	−	−
									ファイルアクセスモード：乱アクセス	−	−	○	○	○	○	E	−	−
									ファイルアクセスモード：動的アクセス	−	−	※3	○	○	○	○	−	−
	WITH LOCK指定	20	1	char	32	1	char	CBLPARM_NOLOCK：レコード施錠要求なし（WITH NO LOCK） CBLPARM_LOCK：レコード施錠要求あり（WITH LOCK） レコード施錠の詳細は，マニュアル「COBOL2002　言語標準仕様編」を参照		○	−	○	−	−	−	−	−	−
	レコード単位アクセスモード	21	1	char	33	1	char	CBLPARM_ACCESS_NEXT：順アクセス（READ NEXT）		−	−	○※4	−	−	−	−	−	−

データ項目名		AIX(32), Linux(x86)の場合			AIX(64), Linux(x64)の場合			設定／参照する値	各サービスルーチンでの指定								
区分	名称および意味	位置	長さ	データ形式	位置	長さ	データ形式		OP	CL	RE	WR	RW	DE	ST	UN	WD
								CBLPARM_ACCESS_KEY：乱アクセス（READ KEY）									
	キー比較条件	22	1	char	34	1	char	位置づける相対レコード番号とキー（key）の比較条件 CBLPARM_KEY_EQUAL：キーの値と等しい（EQUAL） CBLPARM_KEY_GREAT：キーの値より大きい（GREATER THAN） CBLPARM_KEY_NOTLESS：キーの値より小さくない（NOT LESS THAN） CBLPARM_KEY_GREATEQ：キーの値より大きいか，または等しい（GREATER THAN OR EQUAL）	－	－	－	－	－	－	○	－	－
	未使用	23	9	char[9]	35	13	char[13]	サービスルーチン予備領域	※5	※5	※5	※5	※5	※5	※5	※5	※5

（凡例）

OP：CBLOPEN サービスルーチン
CL：CBLCLOSE サービスルーチン
RE：CBLREAD サービスルーチン
WR：CBLWRITE サービスルーチン
RW：CBLREWRITE サービスルーチン
DE：CBLDELETE サービスルーチン
ST：CBLSTART サービスルーチン
UN：CBLUNLOCK サービスルーチン
WD：CBLWDISK サービスルーチン
○：必ず指定する項目
△：COBOL2002 が値を返す項目
E：エラー（乱アクセス法の場合，CBLSTART サービスルーチンは実行できない）
N：無視される項目（直前に CBLREAD サービスルーチンの呼び出しが成功したレコードが対象となる）
－：無視される項目

注※1

固定長レコード形式の場合，このデータ項目の指定は無視されます。

注※2

入出力対象レコードキー番号について次に示します（○△だけ該当）。

- 最初のレコード番号は 1，以降 2，3，…となり 1 ずつ増えます。
- このデータ項目の指定は，相対ファイルだけで有効となります。順ファイルで指定した場合は，無視されます。
- 順アクセスの場合，CBLREAD サービスルーチンや CBLWRITE サービスルーチンによって読み書きされたレコードの，相対レコード番号が返されます。
- 乱アクセス，または動的アクセスの場合，入出力文を実行する前に入出力対象レコードの相対レコード番号を設定する必要があります。

注※3

ファイルアクセスモードが動的アクセスの CBLREAD サービスルーチンを実行したときは，レコード単位アクセスモードの指定によって，順アクセスか乱アクセスかが決まります。

注※4

順ファイルの場合，このデータ項目の指定は無視されます。

注※5

最初に CBLOPEN サービスルーチンを実行する前に NULL（X'00'）を指定して，領域をクリアしておく必要があります。

パラメタインタフェース領域のデータ項目に関する注意事項

(a)レコード単位アクセスモード

レコード単位アクセスモードには，相対ファイルに対する CBLREAD サービスルーチンの実行が，順アクセスか，乱アクセスかを指定します。順ファイルに対してこの項目を指定した場合，指定は無視され，常に順アクセスとなります。

レコード単位アクセスモードに指定する値による動作の違いを，次に示します。

指定する値	レコード単位 アクセスモード	指定可能な ファイルアクセスモード （CBLOPEN サービスルーチン実行時の指定）	動作
CBLPARM_ACCESS_NEXT	順アクセス	• 順アクセス法 • 動的アクセス法	次のどちらかのレコードを読み込む。 • 直前の CBLREAD サービスルーチンによって位置づけられたレコードの，次のレコード • 直前の CBLSTART サービスルーチンによって位置づけられたレコード

指定する値	レコード単位 アクセスモード	指定可能な ファイルアクセスモード (CBLOPEN サービスルーチン実行時の指定)	動作
			ただし，CBLOPEN サービスルーチンを実行した直後の場合は，有効な先頭レコードを読み込む。
CBLPARM_ACCESS_KEY	乱アクセス	• 乱アクセス法 • 動的アクセス法	相対レコード番号キーに指定した番号のレコードを読み込む。

レコードのアクセスモードが順アクセスの場合，直前の CBLREAD サービスルーチン，または CBLSTART サービスルーチンがエラー（パラメタエラーを含む）になると，それに続く CBLREAD サービスルーチンもエラーになります。

(b)キー比較条件

キー比較条件には，相対ファイルに対する CBLSTART サービスルーチンで，位置づける相対レコード番号とキーの比較条件を指定します。キー比較条件の指定値と意味を，次に示します。

指定する値	比較条件	対応する COBOL START 文の KEY 指定	位置づけるレコード（以下の条件を満たさない時はエラー）
CBLPARM_KEY_EQUAL	キーの値と等しい	KEY IS EQUAL TO (KEY IS = TO)	相対レコード番号がキーと等しいレコード
CBLPARM_KEY_GREAT	キーの値より大きい	KEY IS GREATER THAN (KEY IS >)	相対レコード番号がキーの値より大きい最初のレコード
CBLPARM_KEY_NOTLESS	キーの値より小さくない	KEY IS NOT LESS THAN (KEY IS NOT <)	相対レコード番号がキーの値より小さくない最初のレコード
CBLPARM_KEY_GREATEQ	キーの値より大きいか，または等しい	KEY IS GREATER THAN OR EQUAL TO (KEY IS >=)	相対レコード番号がキーの値より大きいか，または等しい最初のレコード

13.4　リンクの指定

COBOL 入出力サービスルーチンを使用したプログラムをリンクする場合，-lcbl2kfl または-lcbl2kfl64 の指定が必要です。指定形式を次に示します。

形式

AIX(32)，Linux(x86)の場合

```
cc  -Iyyy  xxx.c  -L/opt/HILNGcbl2k/lib -lcbl2k -lcbl2kml -lcbl2kfl… -lm
```

Linux(x64)の場合

```
cc  -Iyyy  xxx.c  -L/opt/HILNGcbl2k64/lib -lcbl2k -lcbl2kml -lcbl2kfl… -lm
```

AIX(64)の場合

```
cc  -Iyyy  xxx.c  -q64 -L/opt/HILNGcbl2k64/lib -lcbl2k64 -lcbl2kml64 -lcbl2kfl64… -lm
```

xxx.c

COBOL 入出力サービスルーチンを使用する C プログラムのファイル名

yyy

COBOL インクルードファイルがあるディレクトリ

また，COBOL 入出力サービスルーチンを使用した C プログラムを，COBOL プログラムとともに ccbl2002 コマンドでリンクすることもできます。指定形式を次に示します。

形式

AIX(32)，Linux の場合

```
ccbl2002 -Main,System zzz.cbl xxx.o -lcbl2kfl
```

AIX(64)の場合

```
ccbl2002 -Main,System zzz.cbl xxx.o -lcbl2kfl64
```

xxx.o

COBOL 入出力サービスルーチンを使用する C プログラムのオブジェクトファイル名

zzz.cbl

COBOL ソースファイル

COBOL 入出力サービスルーチンライブラリは，共用ライブラリで提供されています。次に，共用ライブラリを示します。

システム	共用ライブラリ
Linux	libcbl2kfl.so
AIX(32)	libcbl2kfl.a

システム	共用ライブラリ
AIX(64)	libcbl2kfl64.a

13.5　デバッグ情報の取得

COBOL 入出力サービスルーチンでエラーが発生した場合，エラーメッセージは出力されません。その代わり，管理情報インタフェース領域にメッセージ番号や入出力状態などのエラー情報が出力されます。ユーザは，これらの情報や，メッセージ番号が指すエラーメッセージの内容から，エラーの内容を知ることができます。

また，デバッグ情報として，インタフェース領域の内容をダンプ形式で出力できます。

ここでは，これらのデバッグ情報の取得方法，および利用方法について説明します。

13.5.1　COBOL 入出力サービスルーチンで出力されるエラーメッセージ番号

COBOL 入出力サービスルーチンでエラーが発生したときだけに出力されるメッセージ番号と，その対処方法を次の表に示します。次の表にないメッセージ番号が出力された場合は，マニュアル「COBOL2002 メッセージ」の実行時のメッセージの説明を参照してください。

表 13–7　COBOL 入出力サービスルーチンで出力されるメッセージ番号

メッセージ番号		エラーの内容	対処
順ファイル	相対ファイル		
3701	3801	管理情報インタフェース領域の指定に誤りがあります。	管理情報インタフェース領域中の COBOL エラー詳細情報を基に，指定値を見直す。※1
3702	3802	パラメタインタフェース領域の指定に誤りがあります。	管理情報インタフェース領域中の COBOL エラー詳細情報を基に，指定値を見直す。※1
3703	3803	ファイルの属性情報とプログラムの指定との間に矛盾があります。	管理情報インタフェース領域中の COBOL エラー詳細情報を基に，ファイル属性情報とプログラムの指定を一致するように変更する。※2
3704	3804	入出力エラーが発生しました。	管理情報インタフェース領域中のシステムコール番号とシステムエラーコードを基に，原因を調査する。
3705	3805	デバッグ情報の出力中に入出力エラーが発生しました。	管理情報インタフェース領域中のシステムコール番号とシステムエラーコードを基に，原因を調査する。サービスルーチンの処理自体は正常終了している。
3706	3806	デバッグ情報の出力中に入出力エラーが発生しました。サービスルーチンでもエラーが発生しています。	管理情報インタフェース領域中のシステムコール番号とシステムエラーコードを基に，原因を調査する。 管理情報インタフェース領域のエラー情報は，デバッグ情報出力中に発生したエラーに関する情報である。

メッセージ番号		エラーの内容	対処
順ファイル	相対ファイル		
			サービスルーチンの処理自体でもエラーが発生しているため，デバッグ情報出力中のエラーを対策のあと，再実行して，サービスルーチンのエラー原因を調査する。
3707	3807	メッセージ番号が不正です。	関数情報のエラーリターン後に管理情報インタフェース領域が更新された可能性がある。入出力サービスルーチンの発行順序を見直す。
3751	3851	メモリが不足しました。	不要な資源を削除して再実行する。
3752		指定されたファイルは取り扱えません。	COBOL 入出力サービスルーチンで取り扱えるのは，COBOL2002 で作成した順可変長レコード形式，または相対ファイルである。 また，ファイルが破壊されたおそれもある。指定したファイルを見直す。
−	3861	CBLSTART には乱アクセス以外で相対ファイルだけが指定できます。	エラーの内容に従いプログラムを見直す。
−	3862	NEXT 指定の CBLREAD は，相対ファイルの動的アクセス，および順アクセスファイルに指定できます。	エラーの内容に従いプログラムを見直す。
3763	3863	KEY 指定の CBLREAD は，順アクセス以外で相対ファイルだけが指定できます。	エラーの内容に従いプログラムを見直す。
3764	−	CBLDELETE は相対ファイルだけが指定できます。	エラーの内容に従いプログラムを見直す。
3765	−	順ファイルには LOCK MODE MANUAL の指定はできません。	エラーの内容に従いプログラムを見直す。
−	3866	EXTEND 指定の CBLOPEN は，順アクセスのファイルだけが指定できます。	エラーの内容に従いプログラムを見直す。
3767	3867	ファイルが開かれていないため CBLWDISK サービスルーチンが実行できません。	開かれているファイルに対して CBLWDISK サービスルーチンを実行するようにプログラムを修正して再実行する。

（凡例）

−：該当するメッセージ番号はない

注

順ファイルのエラーに対しては 3700 番台のメッセージ番号が，相対ファイルのエラーに対しては 3800 番台のメッセージ番号が，それぞれ設定されます。ただし，エラー発生時にファイル編成が特定できない場合は，3700 番台のメッセージ番号が設定されます。

注※1

各インタフェース領域で複数のデータ項目の指定値に誤りがある場合，COBOL エラー詳細情報には，最初に誤りが発見されたデータ項目名に該当する値が設定されます。

注※2

ファイル属性エラーが発生したときの COBOL エラー詳細情報には，次に示す順で最初に発見されたエラー項目に該当する値が設定されます。

1. ファイルフォーマット

   ・オープンするファイルに，COBOL2002 が定めるヘッダ情報がない。

   ・COBOL2002 で作成した順可変長ファイル，または相対ファイルでない。または，ファイルが破壊されている。

2. ファイル編成／レコード形式
3. 最大レコード長
4. 最小レコード長

13.5.2 インタフェース領域のダンプ出力

COBOL 入出力サービスルーチンが使用する次の三つのインタフェース領域の内容を，デバッグ情報としてダンプ形式で出力できます。

- 管理情報インタフェース領域
- パラメタインタフェース領域
- 入出力レコード領域

ここでは，これらのデバッグ情報の出力方法について説明します。

(1) 出力先ファイル名の指定

デバッグ情報を出力するときは，環境変数 CBL_FLSRVDUMP に出力先となるファイル名を指定します。次に，環境変数 CBL_FLSRVDUMP の指定方法を示します。

形式

```
CBL_FLSRVDUMP=出力先ファイル名
```

出力先ファイル名

デバッグ情報を出力するファイル名を絶対パスで指定します。

規則

- 環境変数 CBL_FLSRVDUMP に指定したファイルがない場合，新規にファイルが作成されます。すでにファイルがある場合は，ファイルの最後にデバッグ情報が追加されます。
- 環境変数 CBL_FLSRVDUMP に指定したパスがない場合，実行時エラーとなります。

- 環境変数 CBL_FLSRVDUMP を指定しない場合，または値を指定しない場合は，デバッグ情報が出力されません。

(2) 出力タイミングの指定

デバッグ情報は，各入出力サービスルーチンの実行前，実行後，またはその両方のときに出力できます。管理情報インタフェース領域のデバッグ情報出力指示に値を指定すると，どのタイミングでデバッグ情報を出力するかを指定できます。次に，デバッグ情報出力指示に指定する値と，デバッグ情報が出力されるタイミングの関係を示します。

デバッグ情報出力指示の値	デバッグ情報が出力されるタイミング
CBLCOM_DBG_NO	デバッグ情報は出力されない
CBLCOM_DBG_BEF	各入出力サービスルーチンの実行前
CBLCOM_DBG_AFT	各入出力サービスルーチンの実行後
CBLCOM_DBG_BEFAFT	各入出力サービスルーチン実行の前後両方

規則

- デバッグ情報出力指示の値は，ファイルを開いている間でも任意に変更できます。これによって，入出力サービスルーチンごとに異なるタイミングでデバッグ情報を出力できます。
- COBOL 入出力サービスルーチンでパラメタエラーが発生した場合，サービスルーチン実行後にはデバッグ情報が出力されません。
- デバッグ情報出力指示の値は，COBOL2002 によって変更されません。

(3) 出力の対象となる領域

デバッグ情報へ出力されるインタフェース領域の種類は，実行する COBOL 入出力サービスルーチンの種類によって異なります。サービスルーチンごとに出力されるデバッグ情報の種類を，次に示します。

COBOL 入出力サービスルーチン	出力される領域			
	管理情報インタフェース領域	パラメタインタフェース領域	読み込みバッファ※1	書き出しバッファ※1
CBLOPEN	○	○	×	×
CBLCLOSE	○	×	×	×
CBLREAD	○	○	○※2	×
CBLWRITE	○	○	×	○
CBLREWRITE	○	○	×	○
CBLUNLOCK	○	×	×	×
CBLDELETE	○	○	×	×

COBOL 入出力サービスルーチン	出力される領域			
	管理情報インタフェース領域	パラメタインタフェース領域	読み込みバッファ※1	書き出しバッファ※1
CBLSTART	○	○	×	×
CBLWDISK	○	×	×	×

（凡例）

○：デバッグ情報に出力される

×：デバッグ情報に出力されない

注※1

バッファ領域は，実際に入出力した長さでダンプ出力されます。

注※2

CBLREAD サービスルーチンの実行前には出力されません。

(4) 出力形式

デバッグ情報には，それぞれの先頭の領域を 0 とした相対的な位置と内容が，ダンプ形式で出力されます。

デバッグ情報の出力例を，次に示します。

```
COBOL2002 (c) VV-RR *** 入出力サービスルーチンデバッグリスト *** YYYY-MM-DD HH:MM:SS

<CBLOPEN 呼び出し前 ファイル名 = /tmp/outfile>
CBLCOMFL※1
位置----   内容---- -------- -------- --------   --------------------
00000000   53414d56 30303032 00000000 00000000   SAMV ..........
00000010   00000000 00000000 00000000 00000000   ..........
           LINES 00000020-00000200 SAME AS ABOVE

CBLPARMFL※2
位置----   内容---- -------- -------- --------   --------------------
00000000   00000000 00000000 00000000 00000000   ..........
00000010   00000000 00000000 00000000 00000000   ..........

<CBLOPEN 呼び出し後 ファイル名=/tmp/outfile>
CBLCOMFL
位置----   内容---- -------- -------- --------   --------------------
00000000   53414d56 00000000 00000000 00000000   SAMV .......
00000010   00000000 00000000 00000000 00000000   ..........
           LINES 00000020-00000200 SAME AS ABOVE

CBLPARMFL
位置----   内容---- -------- -------- --------   --------------------
00000000   00000000 00000000 00000000 00000000   ..........
00000010   00000000 00000000 00000000 00000000   ..........

<CBLWRITE 呼び出し前 ファイル名=/tmp/outfile>
CBLCOMFL
位置----   内容---- -------- -------- --------   --------------------
00000000   53414d56 30303032 00000000 00000000   SAMV ........
00000010   00000000 00000000 00000000 00000000   ..........
           LINES 00000020-00000200 SAME AS ABOVE

CBLPARMFL
位置----   内容---- -------- -------- --------   --------------------
00000000   00000000 00000000 40000001 00000008   ..........
00000010   00000000 00000000 00000000 00000000   ..........

書き出しバッファ
位置----   内容---- -------- -------- --------   --------------------
00000000   41414141 41414141 20202020 20202020   AAAAAAAA
```

注※1

CBLCOMFL に続くダンプリストは，管理情報インタフェース領域を表しています。

注※2

CBLPARMFL に続くダンプリストは，パラメタインタフェース領域を表しています。

13.6　COBOL 入出力サービスルーチンでのディスク書き込み保証

COBOL2002 のファイル入出力機能と同様に，COBOL 入出力サービスルーチンでも，ファイルのクローズ時，または書き込み時に，ディスクへの書き込み保証が適用された入出力処理ができます。

ファイルのディスクへの書き込み保証は，次の COBOL 入出力サービスルーチンの呼び出し完了時に適用されます。

ファイルクローズ時の保証

CBLCLOSE サービスルーチン

ファイル書き込み時の保証

- CBLWRITE サービスルーチン
- CBLREWRITE サービスルーチン
- CBLDELETE サービスルーチン

また，CBLWDISK サービスルーチンを呼び出すと，その時点でのディスクへの書き込み保証が適用されます。

COBOL2002 のファイル入出力機能でのディスク書き込み保証については，「14.1　ファイルのディスク書き込み保証」を参照してください。

COBOL 入出力サービスルーチンでは，次の 3 種類の方法でファイルのディスクへの書き込みを保証します。

- ファイルごとに指定する
- プロセス内のすべてのファイルに対して指定する
- CBLWDISK サービスルーチンを呼び出す

13.6.1　ファイルごとに指定する方法

ファイルごとにディスクへの書き込み保証を適用する場合は，書き込み保証を適用したいファイルに対応する管理情報インタフェース領域中のディスク書き込みオプションを指定します。ディスク書き込みオプションに指定できる値と，その意味を次に示します。

マクロ名	値	意味
CBLCOM_ENVFSYNC	0	環境変数 CBLFSYNC の指定に従う。環境変数 CBLFSYNC の詳細は，「14.1　ファイルのディスク書き込み保証」を参照のこと。
CBLCOM_NOFSYNC	1	クローズ時のディスクへの書き込み保証が適用されない。
CBLCOM_FSYNC	2	クローズ時のディスクへの書き込み保証が適用される。
CBLCOM_WDISK	3	書き込み時のディスクへの書き込み保証が適用される。

13.6.2　プロセス内のすべてのファイルに対して指定する方法

クローズ時のディスクへの書き込み保証は，プロセス内のすべてのファイルに対して指定できます。指定方法を次に示します。

1. 環境変数 CBLFSYNC に YES を指定する。
   環境変数 CBLFSYNC の詳細は，「14.1　ファイルのディスク書き込み保証」を参照してください。
2. すべてのファイルに対応する管理情報インタフェース領域のディスク書き込みオプションに，「CBLCOM_ENVFSYNC」または「CBLCOM_FSYNC」を指定する。
   管理情報インタフェース領域については，「13.3.2　インタフェース領域の形式」を参照してください。

注意事項

ディスク書き込みオプションに CBLCOM_NOFSYNC を指定したファイルは，ディスクへの書き込み保証が適用されません。

13.6.3　CBLWDISK サービスルーチンを呼び出して保証する方法

CBLWDISK サービスルーチンを呼び出すと，指定されたファイルに対して，サービスルーチン呼び出し時までに出力したデータのディスクへの書き込み保証が適用されます。

CBLWDISK サービスルーチンの呼び出し方法については，「13.3　COBOL 入出力サービスルーチンのインタフェース」を参照してください。

また，次のサービスルーチンによってファイルの書き込みが発生したときに，ディスクへの書き込み保証を適用する場合は，ファイルに対応する管理情報インタフェース領域のディスク書き込みオプションに，「CBLCOM_WDISK」を指定してください。

- CBLWRITE サービスルーチン
- CBLREWRITE サービスルーチン
- CBLDELETE サービスルーチン

管理情報インタフェース領域については，「13.3.2　インタフェース領域の形式」を参照してください。

13.6.4　注意事項

COBOL 入出力サービスルーチンでディスク書き込み保証を使用する場合，次の点に注意してください。

- ディスクへの書き込み保証が適用されるサービスルーチンの呼び出しが完了する前にシステムダウンが発生したときは，データのディスクへの書き込み保証は適用されません。
- 管理情報インタフェース領域のディスク書き込みオプションに指定した値は，CBLOPEN サービスルーチンの呼び出し時だけチェックされます。

13.7　COBOL 入出力サービスルーチンでのラージファイル入出力機能

ファイル入出力機能の入出力文と同様に，COBOL 入出力サービスルーチンでもラージファイル入出力機能を使用できます。ラージファイル入出力機能は，ファイルサイズが 2GB 以上のファイル（ラージファイル）に対して入出力できる機能です。ファイル入出力機能のラージファイル入出力機能の詳細は，「6.10 ラージファイル入出力機能」を参照してください。

13.7.1　ラージファイル入出力機能の指定方法

COBOL 入出力サービスルーチンでのラージファイル入出力機能は，ファイル編成が順編成ファイルのときに有効になります。

COBOL 入出力サービスルーチンにラージファイル入出力機能を適用するには，次の方法があります。

(1)　ファイル単位に指定する方法

ファイルごとにラージファイル入出力機能を適用するには，管理情報インタフェース領域のラージファイル対応オプションに CBLCOM_LARGEFILE を指定します。

環境変数 CBLLARGEFILE に YES を指定したとき，ラージファイル入出力機能を適用しないファイルに対しては，CBLCOM_NOLARGEFILE を指定してください。

(2)　実行単位中のすべてのファイルに指定する方法

実行単位中のすべてのファイルにラージファイル入出力機能を適用するには，次の方法があります。

- 環境変数 CBLLARGEFILE に YES を指定する。この場合，管理情報インタフェース領域のラージファイル対応オプションには，CBLCOM_ENVLARGEFILE を指定する。
- すべてのファイルに対して，管理情報インタフェース領域のラージファイル対応オプションに CBLCOM_LARGEFILE を指定する。

(3)　環境変数 CBLLARGEFILE とラージファイル対応オプションの組み合わせ

環境変数 CBLLARGEFILE と管理情報インタフェース領域のラージファイル対応オプションの組み合わせを次の表に示します。

表 13–8　環境変数 CBLLARGEFILE とラージファイル対応オプションの組み合わせ

ラージファイル対応オプション	環境変数 CBLLARGEFILE	
	YES	指定なし
CBLCOM_ENVLARGEFILE	○	×
CBLCOM_NOLARGEFILE	×	×

ラージファイル対応オプション	環境変数 CBLLARGEFILE	
	YES	指定なし
CBLCOM_LARGEFILE	○	○

（凡例）

○：ラージファイル入出力機能を使用する。

×：ラージファイル入出力機能を使用しない。

13.7.2　管理情報インタフェース領域の指定

ラージファイル入出力機能を適用するかどうかをファイル単位に指定したい場合，管理情報インタフェース領域のラージファイル対応オプションを指定します。ラージファイル対応オプションについては，「13.3.2　インタフェース領域の形式」を参照してください。

ラージファイル対応オプションに不正な値を指定して CBLOPEN サービスルーチンを呼び出した場合，サービスルーチンはエラーを返します。このとき，管理情報インタフェース領域の COBOL メッセージ番号には「3701」，COBOL エラー詳細情報には「12」が設定されます。

13.7.3　注意事項

- 相対編成ファイルは，ラージファイル入出力機能には対応していません。
- ラージファイル入出力機能を使用した場合，管理情報インタフェース領域のファイルロックモードの指定や，パラメタインタフェース領域の WITH LOCK 指定は無効となり，ファイル共用はできません。

13.8 COBOL 入出力サービスルーチンの使用例

COBOL 入出力サービスルーチンの使用例を，次に示します。

(例 1) サービスルーチンを使ってファイルを読み込む C プログラム

```
   :
#include "CBL85fl.h"   /* COBOL提供ヘッダファイルの取り込み */
   :
CBLCOMFL   com ;          /* 管理情報インタフェース領域 */
CBLPARMFL  parm ;         /* パラメタインタフェース領域 */
char  filename[100] ;     /* ファイル名称 */
int   end_flg ;

memset(&com, 0x00, sizeof(CBLCOMFL)) ; /* テーブル0クリア */
memset(&parm, 0x00, sizeof(CBLPARMFL)) ;
/* OPENパラメタを設定する */
strcpy(filename, "/tmp/outfile") ;
com.cblcom_pathname = filename ;
com.cblcom_flopt    = CBLCOM_NOCHK ;
                                  /*属性チェックをしない*/
com.cblcom_openmode = CBLCOM_OPEN_INPUT ;
                                  /*OPENモードはINPUT*/
parm.cblparm_lock   = CBLPARM_NOLOCK ;     /*施錠なし*/
if (CBLOPEN(&com, &parm) != 0) {
                               /*CBLOPENサービスルーチン実行*/
ERROUT(&com) ;          /*エラー情報出力用サブルーチン呼び出し*/
return(-1) ;
}
if (memcmp(com.cblcom_form, "SAMV", 4) != 0) {
   printf("ファイルはCOBOL順可変長形式ではない。¥n") ;
   return(-1) ;
}
/* READパラメタを設定する */
parm.cblparm_lock = CBLPARM_NOLOCK ;        /*施錠なし*/
/* OPENのリターン情報に従い入出力バッファ取得 */
if ((parm.cblparm_read_buf = malloc(com.cblcom_maxrec))
      == NULL) {
   :        /* 必要なメモリが取得できない場合のエラー処理 */
}
end_flg = 0 ;
while(end_flg == 0) {              /*レコード終了までループ*/
if (CBLREAD(&com , &parm) == 0) {
                              /*CBLREADサービスルーチン実行*/
   :                               /* READ成功時の処理 */
} else {
   end_flg = 1 ;                /*読み込み終了フラグ　オン */
     if (com.cblcom_fs == 10) {  /*入出力状態値10:ファイル終了*/
       :                         /* ファイル終了時の処理 */
   } else {
      ERROUT(&com) ;             /*エラー処理*/
     }
  }
}
/* 終了処理 */
if (CBLCLOSE(&com, &parm) != 0) {
```

```
                                        /*CBLCLOSEサービスルーチン実行*/
 ERROUT(&com) ;
}
```

(例 2) エラー情報を出力する C 副プログラム

```
extern void ERROUT(p1)
CBLCOMFL *p1 ;
{
   if ((p1->cblcom_msgno >= 3701)
        && (p1->cblcom_msgno <= 3703)) {
     printf("パラメタエラー(詳細情報番号=%d)¥n",
                  p1->cblcom_cbldetail) ;
   } else {
     printf("メッセージ番号=%d¥n",p1->cblcom_msgno) ;
     if (p1->cblcom_sysinf == CBLCOM_ERR_SYSTEM) {
        /* システムエラー情報がある時 */
        printf("システムコール番号=%dエラーコード=%d¥n",
                  p1->cblcom_funcno, p1->cblcom_errno) ;
    }
  }
}
```

13.9 注意事項

ここでは，COBOL 入出力サービスルーチンを使用するときの注意事項について説明します。

- 各インタフェース領域中のシステムが使用する領域，および予備領域は，インタフェース領域を使用する最初の CBLOPEN サービスルーチンを実行する前に，すべて NULL（X'00'）を指定して内容をクリアしてください。また，クリアしたあとは，これらの領域を更新しないでください。領域をクリアしなかった場合，またはクリア後に更新した場合，動作は保証しません。
- 管理情報インタフェース領域に出力されるエラー情報は，サービスルーチンがエラーを返した直後だけ，内容を保証します。
- CBLOPEN サービスルーチンの呼び出しが成功したあとに処理を中断する場合，必ず CBLCLOSE サービスルーチンを呼び出すようにしてください。CBLCLOSE サービスルーチンを呼び出さないで処理を中断した場合，領域の解放やレコード施錠の解除がされません。

14 ファイルのディスク書き込み保証

COBOL2002 には，システムダウンなどが発生したときにデータのディスクへの書き込み漏れが発生しないように，ファイルのディスク書き込み保証機能があります。この章では，ファイルのディスク書き込み保証機能について説明します。

14.1　ファイルのディスク書き込み保証

COBOL2002 の入出力処理では，入出力文の実行とデータのディスクへの書き込みが非同期に処理されます。そのため，入出力文の実行直後にシステムダウンが発生した場合，OS が管理するバッファ内のデータが，ディスクに書き込まれないことがあります。

このようなディスクへの書き込み漏れを防ぐため，COBOL2002 には次のようなファイルのディスク書き込みを保証する機能があります。

ファイルクローズ時の保証

プログラムの終了時，または CLOSE 文の実行終了時に，ファイルのディスク書き込みを保証する機能です。

ファイル書き込み時の保証

WRITE 文，REWRITE 文，または DELETE 文の実行終了時に，ファイルのディスク書き込みを保証する機能です。

ここでは，ファイルのディスク書き込み保証機能について説明します。

14.1.1　対象となるファイル編成

ディスクへの書き込み保証の対象となるファイル編成を次に示します。

表 14–1　ディスクへの書き込み保証の対象となるファイル編成

ファイル種別	ファイルクローズ時の保証	ファイル書き込み時の保証
順編成ファイル	○	○
相対編成ファイル	○	○
ISAM による索引編成ファイル	○	○
HiRDB による索引編成ファイル	×	×
テキスト編成ファイル	○	×
CSV 編成ファイル	○	×

（凡例）

○：ディスク書き込み保証を適用できる

×：ディスク書き込み保証を適用できない※

注※

ディスク書き込み保証を適用できないファイル編成に対して，ディスク書き込み保証を有効にするための環境変数を指定しても実行時エラーにはなりません。この場合，ディスク書き込み保証が無効なままプログラムを続行します。

HiRDB による索引編成ファイルのディスク書き込み保証

HiRDB による索引編成ファイルのディスク書き込み保証は，HiRDB の仕様に従います。詳細は，「HiRDB の解説マニュアル」を参照してください。

なお，HiRDB による索引編成ファイルでは，COMMIT 文の実行後は，ファイルのディスクへの書き込みを保証します。

COBOL 入出力サービスルーチン使用時のディスク書き込み保証

COBOL 入出力サービスルーチン使用時のディスク書き込み保証については，「13.6　COBOL 入出力サービスルーチンでのディスク書き込み保証」を参照してください。

バイトストリーム入出力サービスルーチン使用時のディスク書き込み保証

バイトストリーム入出力サービスルーチン使用時のディスク書き込み保証については，「15.5　バイトストリーム入出力サービスルーチンでのディスク書き込み保証機能」を参照してください。

14.1.2　ファイルクローズ時のディスク書き込み保証の指定方法

ファイルクローズ時のディスク書き込み保証は，「14.1.1　対象となるファイル編成」で示すファイルに対して，ファイル別に指定する方法と，プロセス内のすべてのファイルに対して指定する方法があります。

(1)　ファイル別に指定する方法

形式

```
CBLD_ファイル名= { FSYNC  |  NOFSYNC }
```

機能

環境変数 CBLD_ファイル名に FSYNC を指定すると，COBOL プログラムの環境部のファイル管理記述項で指定したファイル名のファイルに対してクローズ時のディスクへの書き込み保証が適用されます。

NOFSYNC を指定した場合は，ディスクへの書き込み保証が適用されません。

(2)　プロセス内のすべてのファイルを指定する方法

(a)　順編成ファイル，相対編成ファイル，テキスト編成ファイル，CSV 編成ファイル

形式

```
CBLFSYNC=YES
```

機能

環境変数 CBLFSYNC に YES を指定すると，プロセス内のすべてのファイルに対してクローズ時のディスクへの書き込み保証が適用されます。この環境変数の指定をしなかった場合，および YES 以外の値を指定した場合は，クローズ時のディスクへの書き込み保証は適用されません。

また，環境変数 CBLD_ファイル名に NOFSYNC が指定されたファイルに対しては，環境変数 CBLFSYNC の指定は有効とならないため，クローズ時のディスクへの書き込み保証は適用されません。

(b)　ISAM による索引編成ファイル

形式

```
CBLISAMFSYNC=YES
```

機能

環境変数 CBLISAMFSYNC に YES を指定すると，プロセス内のすべての ISAM による索引編成ファイルに対してクローズ時のディスクへの書き込み保証が適用されます。この環境変数の指定をしなかった場合，および YES 以外の値を指定した場合は，クローズ時のディスクへの書き込み保証は適用されません。

また，環境変数 CBLD_ファイル名に NOFSYNC が指定されたファイルに対しては，環境変数 CBLISAMFSYNC の指定は有効とならないため，クローズ時のディスクへの書き込み保証は適用されません。

(3)　注意事項

次の場合，ファイルのディスクへの書き込み保証は適用されません。

- CLOSE 文の動作が完了する前にシステムダウンが発生した場合
- ファイルクローズ時のディスク書き込み保証が行なわれる前にシステムダウンが発生して，ディスクへ書き込まれなかった場合

14.1.3　ファイル書き込み時のディスク書き込み保証の指定方法

形式

```
CBLD_ファイル名=WDISK
```

機能

環境変数 CBLD_ファイル名に WDISK を指定すると，COBOL プログラムの環境部のファイル管理記述項で指定したファイル名のうち，「14.1.1　対象となるファイル編成」で示すファイルに対して，書き込み時のディスクへの書き込み保証が適用されます。

注意事項

次の場合，ファイルのディスクへの書き込み保証は適用されません。

- WRITE 文，REWRITE 文，DELETE 文の動作が完了する前にシステムダウンが発生した場合

14.1.4　整列併合機能の出力ファイルに対するディスク書き込み保証

SORT 文および MERGE 文で使用する出力用ファイルにディスク書き込み保証を適用する場合は，出力用ファイル名に対して環境変数 CBLD_ファイル名に FSYNC，または CBLD_ファイル名に WDISK を指定

します。また，環境変数 CBLFSYNC に YES を指定した場合は，プロセス内のすべてのファイルに対してディスク書き込み保証が有効となり，整列併合機能で使用する出力用ファイルにもクローズ時のディスクへの書き込み保証が適用されます。

ただし，出力用ファイルとして索引ファイルを指定した場合は，環境変数 CBLD_ファイル名に WDISK を指定した場合だけにディスク書き込み保証が適用されます。

ディスク書き込み保証の対象ファイルについては，「14.1.1　対象となるファイル編成」を参照してください。

次に，COBOL プログラムとファイル別の環境変数の指定例を示します。

COBOL プログラムの指定例

```
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
            SELECT INFILE   ASSIGN SYS010.
            SELECT OUTFILE  ASSIGN SYS020.
            SELECT SORTFILE ASSIGN SYS001S.
           :
       PROCEDURE DIVISION.
            SORT SORTFILE DESCENDING KEY SORT-KEY
                 USING  INFILE
                 GIVING OUTFILE.
```

環境変数の指定例

```
CBLD_OUTFILE=FSYNC
```

14.1.5　注意事項

- 環境変数 CBLD_ファイル名の FSYNC／NOFSYNC／WDISK と，環境変数 CBLFSYNC および環境変数 CBLISAMFSYNC を同時に指定した場合，環境変数 CBLD_ファイル名の指定が有効となります。このとき，環境変数 CBLD_ファイル名を指定したファイルに対しては，環境変数 CBLFSYNC および環境変数 CBLISAMFSYNC の指定は無効となります。
- 環境変数 CBLD_ファイル名に FSYNC／NOFSYNC／WDISK の指定がないファイルでは，環境変数 CBLFSYNC および環境変数 CBLISAMFSYNC の指定が有効となります。

環境変数の組み合わせと，保証するディスク書き込みについて次に示します。

環境変数 CBLFSYNC および環境変数 CBLISAMFSYNC	環境変数 CBLD_ファイル名			
	FSYNC	NOFSYNC	WDISK	指定なし
YES	C	×	W	C
YES 以外，または 環境変数指定なし	C	×	W	×

（凡例）

C：該当ファイルに対して，ファイルクローズ時のディスクへの書き込み保証を適用する

W：該当ファイルに対して，ファイル書き込み時のディスクへの書き込み保証を適用する

×：該当ファイルに対して，ファイルのディスクへの書き込み保証を適用しない

- 標準出力（stdout）または標準エラー出力（stderr）は，ディスク書き込み保証機能の対象になりません。

15 バイトストリーム入出力サービスルーチン

バイトストリーム入出力サービスルーチンを使用すると，COBOL のレコード定義に依存しないで，C プログラムなどで作成したバイナリファイルの読み書きができます。

この章では，バイトストリーム入出力サービスルーチンの使い方について説明します。

15.1　バイトストリーム入出力サービスルーチンの概要

バイトストリーム入出力サービスルーチンを使用すると，COBOL のレコード定義に依存しないで，C プログラムなどで作成したバイナリファイルの読み書きができます。

15.1.1　バイトストリーム入出力サービスルーチンの一覧

バイトストリーム入出力サービスルーチンの一覧を，次に示します。

表 15–1　バイトストリーム入出力サービスルーチンの一覧

サービスルーチンの名称	機能
CBLSTMCLOSE	バイトストリーム処理用に開かれたファイルを閉じる。
CBLSTMCREATE	バイトストリーム処理用の新しいファイルを作成する。
CBLSTMOPEN	バイトストリーム処理用に既存のファイルを開く。
CBLSTMREAD	ファイルからバイト列を読み込む。
CBLSTMWRITE	ファイルへバイト列を書き出す。

15.1.2　バイトストリーム入出力サービスルーチンを使用するときの注意事項

- バイトストリーム入出力サービスルーチンが正常終了したかどうかは，RETURN-CODE 特殊レジスタを参照して確認してください。
- バイトストリーム入出力サービスルーチンは，COBOL プログラムだけから呼び出せます。COBOL 以外のプログラムから呼び出されたときの動作は保証しません。
- 引数を誤って指定したり，省略したりしたときの動作は保証しません。
- 初期化属性プログラムが呼び出された場合，バイトストリームファイルの状態は初期化されません。また，CANCEL 文を実行しても，バイトストリームファイルの状態は取り消されません。
- ラージファイル入出力機能を使用すると，バイトストリーム入出力サービスルーチンで，ラージファイル（ファイルサイズが 2GB 以上のファイル）を取り扱えます。ラージファイル入出力機能を使用しない場合は，ラージファイルを取り扱えません。この場合に取り扱うことができる最大ファイルサイズは，2,147,483,135 バイト以内です。これを超えたサイズのファイルを読み書きした場合，結果は保証しません。

  バイトストリーム入出力サービスルーチンでのラージファイル入出力機能については，「15.4　バイトストリーム入出力サービスルーチンでのラージファイル入出力機能」を参照してください。
- バイトストリーム入出力サービスルーチンは，COBOL ファイル入出力機能および COBOL 入出力サービスルーチン※で作成した順固定長ファイルを読み書きできます。順可変長ファイルまたは順編成ファイル以外のファイル編成で作成されたファイルに対する読み書きはできません。

注※

COBOL 入出力サービスルーチンとは，COBOL で作成した順ファイル，および相対ファイルにアクセスできる CBLOPEN などのサービスルーチンであり，バイトストリーム入出力サービスルーチンではありません。COBOL 入出力サービスルーチンの詳細は，「13.　COBOL 入出力サービスルーチン」を参照してください。

- バイトストリーム入出力サービスルーチンを使用して，固定長レコードの繰り返しとして作成したファイルは，COBOL ファイル入出力機能および COBOL 入出力サービスルーチンで，順固定長ファイルとして読み書きできます。ただし，ファイルの内容は，ユーザ側で保証してください。また，順可変長ファイルまたは順編成ファイル以外のファイル編成としては読み書きできません。
- バイトストリーム入出力サービスルーチンは，マルチスレッド環境下では実行できません。
- バイトストリーム入出力サービスルーチンの排他モード指定は，複数の実行単位（プロセス）でファイルを共用する場合の動作を指定します。なお，排他モードで読み込み／書き出しを禁止する場合は，OPEN モードが読み込み専用であっても物理ファイルに対して書き込み権限が必要です。
- バイトストリーム入出力サービスルーチン呼び出し時にエラーが発生したときは，COBOL 実行時メッセージを出力し，RETURN-CODE 特殊レジスタにエラーの要因別の値を設定します。出力される実行時メッセージについては，マニュアル「COBOL2002　メッセージ」を参照してください。

RETURN-CODE 特殊レジスタに返す値

0：処理が正常終了した。

負の値：引数の指定に誤りがある。

正の値：上記以外で処理中にエラーが発生した。

表 15–2　RETURN-CODE 特殊レジスタに返す値とその内容

値	内容
0	処理が正常に終了した。
-1	OPEN モードが 1／2／3 以外。
-2	排他モードが 0／1／2／3 以外。
-3	OPEN モードが 2 のとき，排他モードが 0 または 2 でない。
-4	フラグが 0／128 以外（CBLSTMREAD サービスルーチン），または 0 以外（CBLSTMWRITE サービスルーチン）。
-5	相対位置に指定された値が最大値を超えている。
-6	読み書きするバイト数に指定された値が最大値を超えている。
-7	ファイル名の長さが最大値を超えている。
-8	ファイル名の指定がない。
-9	ファイルハンドルに指定された値が正しくない。 • CBLSTMCREATE サービスルーチン，または CBLSTMOPEN サービスルーチンの場合，0 でない。

値	内容
	• CBLSTMREAD サービスルーチン，CBLSTMWRITE サービスルーチン，または CBLSTMCLOSE サービスルーチンの場合，CBLSTMCREATE サービスルーチン，または CBLSTMOPEN サービスルーチンで返される値ではない。
10	ファイルの終わりに達した。
30	ファイルの範囲を超えて読み込もうとした。
34	指定したファイルサイズが上限に達した。
35	ファイルが存在しないとき CBLSTMOPEN サービスルーチンを実行しようとした。
37	指定されたファイルでは CBLSTMCREATE サービスルーチンまたは CBLSTMOPEN サービスルーチンに書かれたモードは使用できない。
47	読み込み専用または読み込み／書き出しモードで開かれていないファイルに対して CBLSTMREAD サービスルーチンを実行しようとした。
48	書き出しまたは読み込み／書き出しモードで開かれていないファイルに対して CBLSTMWRITE サービスルーチンを実行しようとした。
90	入出力エラーが発生した。
93	ファイルはすでに使用されている。

15.2 バイトストリーム入出力サービスルーチンの説明

15.2.1 CBLSTMCLOSE

バイトストリーム処理用に開いたファイルを閉じます。

形式

```
CALL ’CBLSTMCLOSE’ USING  引数1
```

規則

- AIX(32)，Linux(x86)の場合，引数 1 には，ファイルを開いたときに返されたファイルハンドルを 4 バイトの符号なし 2 進項目（COMP-X）で指定します。
- AIX(64)，Linux(x64)の場合，引数 1 には，ファイルを開いたときに返されたファイルハンドルを 8 バイトの符号なし 2 進項目（COMP-X）で指定します。
- STOP RUN 文，または COBOL 主プログラム中の GOBACK 文が実行されるときに開いているバイトストリームファイルがあれば自動的に閉じられます。
- ファイルが正常に閉じられた場合，引数 1 には 0 を返します。

15.2.2 CBLSTMCREATE

バイトストリーム処理用の新しいファイルを作成します。

形式

```
CALL ’CBLSTMCREATE’ USING  引数1 引数2 引数3 引数4 引数5
```

規則

- 引数 1 には，作成するファイル名を英数字項目で指定します。領域の長さはファイル名の長さ＋ 1 バイト以上が必要で，ファイル名は空白（X'20'），または NULL 文字（X'00'）で止める必要があります。
- 引数 2 には，OPEN モードを 1 バイトの符号なし 2 進項目（COMP-X）で指定します。指定する値は次のとおりです。

  1：読み込み専用

  2：書き出し専用（排他モードに 0 または 2 を指定する必要がある）

  3：読み込み／書き出し
- 引数 3 には，排他モードを 1 バイトの符号なし 2 進項目（COMP-X）で指定します。指定する値は次のとおりです。

  0，2：読み込み／書き出しを禁止

  1：書き出しを禁止

3：読み込み／書き出しを禁止しない

- 引数 4 には，1 バイトの符号なし 2 進項目（COMP-X）で，0 を指定します。

  ファイルごとにラージファイル入出力機能を適用するには，ラージファイル対応オプションを指定します。指定する値は，「15.4.1　ラージファイル入出力機能の指定方法」を参照してください。

- AIX(32)，Linux(x86)の場合，引数 5 には，開いたファイルハンドルを受け取る領域を 4 バイトの符号なし 2 進項目（COMP-X）で指定します。CBLSTMCREATE サービスルーチン呼び出し時，引数 5 の値は 0 でなければなりません。
- AIX(64)，Linux(x64)の場合，引数 5 には，開いたファイルハンドルを受け取る領域を 8 バイトの符号なし 2 進項目（COMP-X）で指定します。CBLSTMCREATE サービスルーチン呼び出し時，引数 5 の値は 0 でなければなりません。

注意事項

- ファイルハンドルを受け取る領域の値は，そのファイルを閉じるまで必要です。この値を更新した場合の動作は保証しません。
- 空白を含むファイル名は指定できません。指定できるファイル名の最大長は 255 バイトです。

15.2.3　CBLSTMOPEN

バイトストリーム処理用に既存のファイルを開きます。

形式

```
CALL ’CBLSTMOPEN’ USING  引数1 引数2 引数3 引数4 引数5
```

規則

- 引数 1 には，開くファイル名を英数字項目で指定します。領域の長さはファイル名の長さ＋ 1 バイト以上必要で，ファイル名は空白（X'20'）または NULL 文字（X'00'）で止める必要があります。
- 引数 2 には，OPEN モードを 1 バイトの符号なし 2 進項目（COMP-X）で指定します。指定する値は次のとおりです。

  1：読み込み専用

  2：書き出し専用（排他モードに 0 または 2 を指定する）

  3：読み込み／書き出し

- 引数 3 には，排他モードを 1 バイトの符号なし 2 進項目（COMP-X）で指定します。指定する値は次のとおりです。

  0，2：読み込み／書き出しを禁止

  1：書き出しを禁止

  3：読み込み／書き出しを禁止しない

- 引数 4 には，1 バイトの符号なし 2 進項目（COMP-X）で，0 を指定します。

ファイルごとにラージファイル入出力機能を適用するには，ラージファイル対応オプションを指定します。指定する値は，「15.4.1　ラージファイル入出力機能の指定方法」を参照してください。

- AIX(32)，Linux(x86)の場合，引数 5 には，開いたファイルハンドルを受け取る領域を 4 バイトの符号なし 2 進項目（COMP-X）で指定します。CBLSTMOPEN サービスルーチン呼び出し時，引数 5 の値は 0 でなければなりません。
- AIX(64)，Linux(x64)の場合，引数 5 には，開いたファイルハンドルを受け取る領域を 8 バイトの符号なし 2 進項目（COMP-X）で指定します。CBLSTMOPEN サービスルーチン呼び出し時，引数 5 の値は 0 でなければなりません。

注意事項

- ファイルハンドルを受け取る領域の値は，そのファイルを閉じるまで必要です。この値を更新した場合の動作は保証しません。
- 空白を含むファイル名は指定できません。指定できるファイル名の最大長は 255 バイトです。

15.2.4　CBLSTMREAD

ファイルからバイト列を読み込みます。

形式

```
CALL ’CBLSTMREAD’ USING 引数1 引数2 引数3 引数4 引数5
```

規則

- AIX(32)，Linux(x86)の場合，引数 1 には，ファイルを開いたときに返されたファイルハンドルを 4 バイトの符号なし 2 進項目（COMP-X）で指定します。
- AIX(64)，Linux(x64)の場合，引数 1 には，ファイルを開いたときに返されたファイルハンドルを 8 バイトの符号なし 2 進項目（COMP-X）で指定します。
- 引数 2 には，読み込む位置を 8 バイトの符号なし 2 進項目（COMP-X）で指定します。この値はファイルの先頭を 0 とした相対位置です。指定できる最大値は 2,147,483,135 です。ただし，引数 4 の値が 128 のときは，この領域に現在のファイルサイズが返されます。
- 引数 3 には，読み込むバイト数を 4 バイトの符号なし 2 進項目（COMP-X）で指定します。指定できる最大値は 65,535 です。0 を指定した場合は読み込まれません。
- 引数 4 には，パラメタを 1 バイトの符号なし 2 進項目（COMP-X）で指定します。指定する値は次のとおりです。

  0：標準の読み込み用。

  128：現在のファイルサイズを引数 2 に指定した領域に返す。このとき，バイト列は読み込まれない。
- 引数 5 には，バイト列が読み込まれるバッファを英数字項目で指定します。バッファのサイズは読み込まれるバイト列の格納に十分な大きさを確保しておく必要があります。

注意事項

- 引数５に指定したバッファのうち，引数３で指定した読み込むバイト数を超える領域は，バイトストリーム入出力サービスルーチンでは更新されません。

15.2.5 CBLSTMWRITE

ファイルへバイト列を書き出します。

形式

```
CALL ’CBLSTMWRITE’ USING 引数1 引数2 引数3 引数4 引数5
```

規則

- AIX(32)，Linux(x86)の場合，引数１には，ファイルを開いたときに返されたファイルハンドルを４バイトの符号なし２進項目（COMP-X）で指定します。
- AIX(64)，Linux(x64)の場合，引数１には，ファイルを開いたときに返されたファイルハンドルを８バイトの符号なし２進項目（COMP-X）で指定します。
- 引数２には，書き出すファイル内の位置を８バイトの符号なし２進項目（COMP-X）で指定します。この値はファイルの先頭を０とした相対位置です。指定できる最大値は 2,147,483,135 です。
- 引数３には，書き出すバイト数を４バイトの符号なし２進項目（COMP-X）で指定します。指定できる最大値は 65,535 です。０を指定した場合は書き出されません。
- 引数４には，パラメタを１バイトの符号なし２進項目（COMP-X）で指定します。指定する値は次のとおりです。

  ０：標準の書き出し用
- 引数５には，書き出すバイト列を格納したバッファを英数字項目で指定します。

15.3 使用例

バイトストリーム入出力サービスルーチンの使用例を次に示します。

AIX(32)，Linux(x86)の場合

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SAMPLE1.
   :
 DATA  DIVISION.
 WORKING-STORAGE SECTION.
   01 FILE-NAME      PIC X(255)      VALUE '/sample/file.dat'.
   01 RSV            PIC X     COMP-X VALUE 0.
   01 PARM           PIC X     COMP-X VALUE 0.
   01 OPEN-MODE      PIC X     COMP-X VALUE 2.
   01 EXCLUSION-MODE PIC X     COMP-X VALUE 0.
   01 HANDLE         PIC X(4)  COMP-X VALUE 0.
   01 DATA-START     PIC X(8)  COMP-X.
   01 DATA-LENGTH    PIC X(4)  COMP-X.
   01 BUF            PIC X(128)      VALUE ALL '*'.
      :
 PROCEDURE DIVISION.
* バイトストリーム処理用のファイルを書き出しモードで開く
     CALL 'CBLSTMOPEN'
           USING FILE-NAME OPEN-MODE EXCLUSION-MODE RSV HANDLE.
     IF (RETURN-CODE NOT = 0) THEN
* エラー処理
       :
     END-IF.
       :
* ファイルの先頭を0とする相対位置63(64バイト目)から
* 128バイトの長さのデータを書き出す
     MOVE 63  TO DATA-START.
     MOVE 128 TO DATA-LENGTH.
     CALL 'CBLSTMWRITE'
           USING HANDLE DATA-START DATA-LENGTH PARM BUF.
     IF (RETURN-CODE NOT = 0) THEN
* エラー処理
       :
     END-IF.
       :
* ファイルを閉じる
     CALL 'CBLSTMCLOSE' USING HANDLE.
     IF (RETURN-CODE NOT = 0) THEN
* エラー処理
       :
     END-IF.

     STOP RUN.
```

AIX(64)，Linux(x64)の場合

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SAMPLE1.
   :
 DATA  DIVISION.
```

```
 WORKING-STORAGE SECTION.
   01 FILE-NAME      PIC X(255)      VALUE '/sample/file.dat'.
   01 RSV            PIC X     COMP-X VALUE 0.
   01 PARM           PIC X     COMP-X VALUE 0.
   01 OPEN-MODE      PIC X     COMP-X VALUE 2.
   01 EXCLUSION-MODE PIC X     COMP-X VALUE 0.
   01 HANDLE         PIC X(8)  COMP-X VALUE 0.
   01 DATA-START     PIC X(8)  COMP-X.
   01 DATA-LENGTH    PIC X(4)  COMP-X.
   01 BUF            PIC X(128)      VALUE ALL '*'.
      :
 PROCEDURE DIVISION.
* バイトストリーム処理用のファイルを書き出しモードで開く
     CALL 'CBLSTMOPEN'
          USING FILE-NAME OPEN-MODE EXCLUSION-MODE RSV HANDLE.
     IF (RETURN-CODE NOT = 0) THEN
* エラー処理
       :
     END-IF.
       :
* ファイルの先頭を0とする相対位置63(64バイト目)から
* 128バイトの長さのデータを書き出す
     MOVE 63  TO DATA-START.
     MOVE 128 TO DATA-LENGTH.
     CALL 'CBLSTMWRITE'
          USING HANDLE DATA-START DATA-LENGTH PARM BUF.
     IF (RETURN-CODE NOT = 0) THEN
* エラー処理
       :
     END-IF.
       :
* ファイルを閉じる
     CALL 'CBLSTMCLOSE' USING HANDLE.
     IF (RETURN-CODE NOT = 0) THEN
* エラー処理
       :
     END-IF.

     STOP RUN.
```

15.4　バイトストリーム入出力サービスルーチンでのラージファイル入出力機能

ファイル入出力機能の入出力文と同様に，バイトストリーム入出力サービスルーチンでもラージファイル入出力機能を使用できます。ラージファイル入出力機能は，ファイルサイズが 2GB 以上のファイル（ラージファイル）に対して入出力できる機能です。ファイル入出力機能のラージファイル入出力機能の詳細は，「6.10　ラージファイル入出力機能」を参照してください。

15.4.1　ラージファイル入出力機能の指定方法

バイトストリーム入出力サービスルーチンにラージファイル入出力機能を適用するには，次の方法があります。

(1)　実行単位中のすべてのファイルに指定する方法

実行単位中のすべてのファイルにラージファイル入出力機能を適用するには，次の方法があります。

- すべてのファイルに対して，ラージファイル対応オプションに 1 を指定します。ラージファイル対応オプションについては，「(2)　ファイルごとに指定する方法」を参照してください。
- 環境変数 CBLSTMLARGEFILE に YES を指定します。この場合，ラージファイル対応オプションに 0 または 1 を指定します。

  環境変数 CBLSTMLARGEFILE について，次に示します。

形式

```
CBLSTMLARGEFILE=YES
```

規則

- ラージファイル対応オプションに 0 または 1 を指定したバイトストリーム入出力サービスルーチンによるファイルに対してはラージファイル入出力機能が適用されます。2 を指定したバイトストリーム入出力サービスルーチンによるファイルに対してはラージファイル入出力機能が適用されません。
- YES 以外の値を指定した場合，バイトストリーム入出力サービスルーチンでのラージファイル入出力機能はラージファイル対応オプションの指定に従います。

(2)　ファイルごとに指定する方法

ファイルごとにラージファイル入出力機能を適用するには，次のバイトストリーム入出力サービスルーチンの引数 4 に，ラージファイル対応オプションを 1 バイトの符号なし 2 進項目(COMP-X)で指定します。

- CBLSTMOPEN
- CBLSTMCREATE

ラージファイル対応オプションに指定する値は次のとおりです。

- 0：環境変数 CBLSTMLARGEFILE の指定に従う。
- 1：ラージファイル入出力機能を使用する。
- 2：ラージファイル入出力機能を使用しない。

適用したいファイルに対して，ラージファイル対応オプションに 1 を指定します。なお，環境変数 CBLSTMLARGEFILE に YES を指定した環境でバイトストリーム入出力サービスルーチンでのラージファイル入出力機能を適用したくないファイルがある場合は，ラージファイル対応オプションに 2 を指定します。

(3) 環境変数 CBLSTMLARGEFILE とラージファイル対応オプションの組み合わせ

環境変数 CBLSTMLARGEFILE とラージファイル対応オプションの組み合わせを次の表に示します。

表 15–3 環境変数 CBLSTMLARGEFILE とラージファイル対応オプションの組み合わせ

ラージファイル対応オプション	環境変数 CBLSTMLARGEFILE の指定値	
	YES	指定なし
0	○	×
1	○	○
2	×	×

（凡例）

○：バイトストリーム入出力サービスルーチンでラージファイル入出力機能を使用する。

×：バイトストリーム入出力サービスルーチンでラージファイル入出力機能を使用しない。

15.4.2 ラージファイル入出力機能を使用したときのバイトストリーム入出力サービスルーチン引数の上限値

ラージファイル入出力機能の使用によって拡張される，バイトストリーム入出力サービスルーチン引数の上限値を次の表に示します。

表 15–4 ラージファイル入出力機能の使用によって拡張される上限値

バイトストリーム入出力サービスルーチン	引数	ラージファイル入出力機能を使用しない場合	ラージファイル入出力機能を使用する場合
CBLSTMREAD	引数 2	指定できる最大値は 2,147,483,135 です。	2,147,483,136 以上の値も指定できます。指定できる相対位置の最大値は，システムで作成できるファイルサイズに依存します。
	引数 3	指定できる最大値は 65,535 です。	指定できる最大値は 1,073,741,823 です。

バイトストリーム入出力サービスルーチン	引数	ラージファイル入出力機能を使用しない場合	ラージファイル入出力機能を使用する場合
CBLSTMWRITE	引数 2	指定できる最大値は 2,147,483,135 です。	2,147,483,136 以上の値も指定できます。指定できる相対位置の最大値は，システムで作成できるファイルサイズに依存します。
	引数 3	指定できる最大値は 65,535 です。	指定できる最大値は 1,073,741,823 です。

15.4.3 注意事項

- CBLSTMOPEN サービスルーチンまたは CBLSTMCREATE サービスルーチンのラージファイル対応オプションに不正な値を指定して呼び出した場合，サービスルーチンはエラーになります。このとき，RETURN-CODE 特殊レジスタに「-10」を返します。
- ラージファイルを使用するには，ファイルシステムの属性がラージファイルに対応している必要があります。ファイルシステムの属性については，システムのマニュアルを参照してください。また，作成できるファイルサイズについては，システム資源の制限が設定されている場合があります。詳細は，システムの ulimit コマンドなどについて記載されたマニュアルを参照してください。
- 次のどちらかに該当する場合，ラージファイル使用時の動作保証はしません。
  - ファイルシステムの属性がラージファイルに対応していない。
  - バイトストリーム入出力サービスルーチンでのラージファイル入出力機能を使用していない。

15.5 バイトストリーム入出力サービスルーチンでのディスク書き込み保証機能

バイトストリーム入出力サービスルーチンで，ファイルのクローズ時にディスクへの書き込み保証を適用するには，環境変数 CBLSTMFSYNC に YES を指定します。

環境変数 CBLSTMFSYNC について，次に示します。

形式

```
CBLSTMFSYNC=YES
```

機能

環境変数 CBLSTMFSYNC に YES を指定すると，プロセス内のすべてのバイトストリーム入出力サービスルーチンのファイルに対して，CBLSTMCLOSE サービスルーチンの呼び出し完了時にクローズ時のディスクへの書き込み保証が適用されます。この環境変数の指定をしなかった場合，および YES 以外の値を指定した場合は，クローズ時のディスクへの書き込み保証は適用されません。

注意事項

次の場合，ファイルのクローズ時にディスクへの書き込み保証は適用されません。

- CBLSTMCLOSE サービスルーチンの動作が完了する前にシステムダウンが発生した。
- ファイルクローズ時のディスク書き込み保証が行なわれる前にシステムダウンが発生して，ディスクへ書き込まれなかった。

16 COBOL の実行単位

この章では，COBOL の実行単位について説明します。

16.1　実行単位の構成

16.1.1　実行単位とは

プログラムの実行時に，プログラムを構成する最も包括的な単位を実行単位といいます。実行単位は，一つの実行可能ファイル，または複数のお互いに連絡し合う実行可能ファイルや共用ライブラリから構成されている，一つの閉じた処理の単位です。

また，実行単位には，COBOL 以外の言語で作成された実行可能ファイルや共用ライブラリ（C プログラムなど）を含むことができます。

16.1.2　実行可能ファイルや共用ライブラリとは

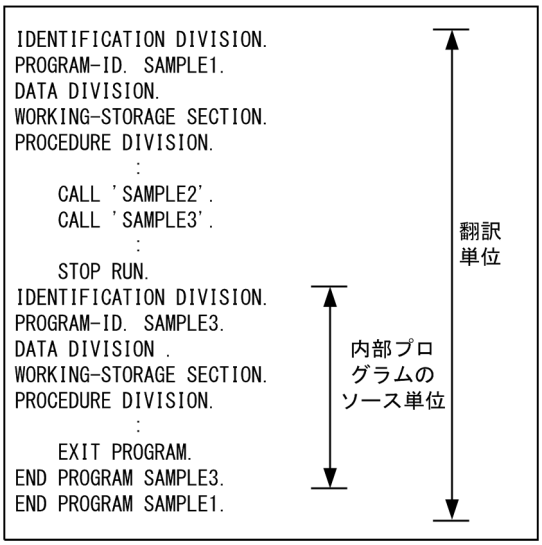
実行可能ファイルや共用ライブラリは，翻訳単位（原始プログラム）をコンパイルして生成した結果です。実行可能ファイルや共用ライブラリは，それに含まれる関数，メソッド，およびプログラムの翻訳結果である一つ以上のオブジェクトファイルから構成されています。

16.1.3　実行単位と実行モジュールの例

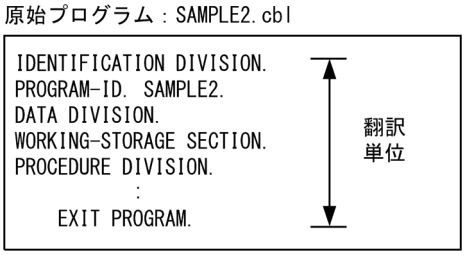
COBOL 原始プログラム「SAMPLE1.cbl」「SAMPLE2.cbl」をコンパイルした場合に，生成される COBOL プログラムの実行単位，実行モジュール，およびプログラムを次に示します。

（原始プログラム：SAMPLE1.cbl）

原始プログラム：SAMPLE1.cbl

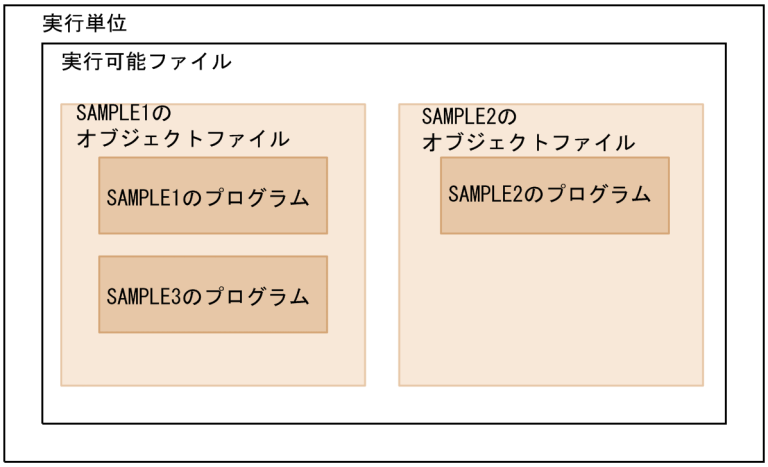


(原始プログラム：SAMPLE2.cbl)

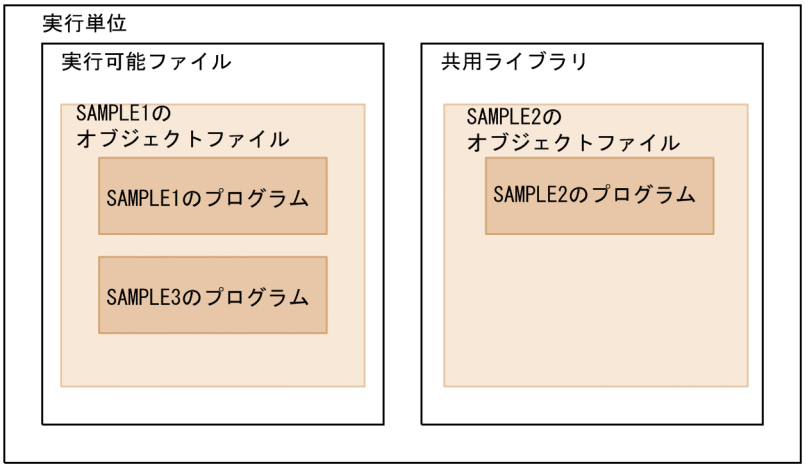


(生成されるプログラムの構成)

一つの実行可能ファイルで構成される場合



複数の実行可能ファイルや共用ライブラリで構成される場合



16.2　引数の受け取りと外部スイッチ

実行単位は，ほかの実行単位と共通のデータファイルや通信文を処理したり，外部スイッチを設定・参照したりできます。また，実行単位は，プログラムの実行時にコマンド行に指定した引数を受け取れます。

ここでは，実行単位で受け取るコマンド行の引数の形式と受け取り方法，および外部スイッチの設定・参照方法について説明します。

16.2.1　コマンド行に指定する引数の形式

コマンド行から受け取る引数の形式には，C 言語インタフェースに従った形式と，VOS3 インタフェースに従った形式の 2 種類があります。

(1)　C 言語インタフェースに従った形式

C 言語インタフェースに従った形式でコマンド行に指定した引数を受け取る場合，COBOL プログラムには，次の形式で引数が渡されます。

コマンド行

```
実行可能ファイル名△引数1△引数2△…
```

（凡例）

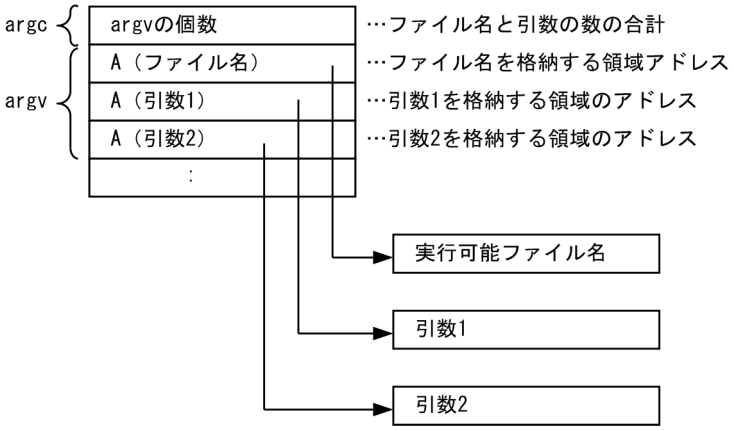
△：半角空白文字

COBOL プログラムに渡される引数の形式

コマンドライン

実行可能ファイル名△引数1△引数2△…

プログラム制御から渡される引数構造体



（凡例）

△：半角空白文字

規則

- C 言語インタフェースに従った形式で引数を受け取る COBOL プログラムは，コンパイル時に-Main,System オプションを指定する必要があります。
- コマンド行に指定した実行可能ファイル名は，入力した文字列のまま取得されます。

なお，COBOL プログラムで C 言語インタフェースに従った形式の引数を受け取るコーディング方法については，「16.2.2　引数の受け取り方法（C 言語インタフェースに従った形式の場合）」を参照してください。

(2)　VOS3 インタフェースに従った形式

VOS3 インタフェースに従った形式でコマンド行に指定した引数を受け取る場合，COBOL プログラムには，次の形式で引数が渡されます。

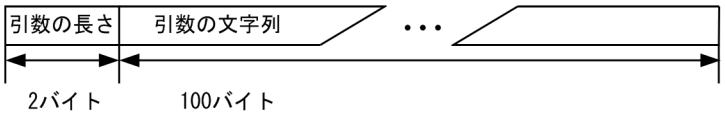
コマンド行

実行可能ファイル名△引数

（凡例）

△：半角空白文字

COBOL プログラムに渡される引数の形式



規則

- VOS3 インタフェースに従った形式で引数を受け取る COBOL プログラムは，コンパイル時に-Main,V3 オプションを指定する必要があります。

なお，COBOL プログラムで VOS3 インタフェースに従った形式の引数を受け取るコーディング方法については，「16.2.3　引数の受け取り方法（VOS3 インタフェースに従った形式の場合）」を参照してください。

16.2.2　引数の受け取り方法（C 言語インタフェースに従った形式の場合）

C 言語インタフェースに従った形式でコマンド行に指定した引数を受け取る方法には，次の 3 種類があります。

- アドレスデータ項目を使った引数の受け取り
- サービスルーチンを使った引数の受け取り
- ACCEPT／DISPLAY 文を使った引数の受け取り

それぞれの受け取り方法について，次に示します。

(1) アドレスデータ項目を使った引数の受け取り

コマンド行から渡された引数の形式を連絡節（LINKAGE SECTION）で定義して，直接 COBOL プログラムから参照する方法です。なお，引数の値は，アドレスとして渡されるため，アドレスデータ項目で参照する必要があります。

アドレスデータ項目を使った引数の受け取りの例

```
              :
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE1.
              :
DATA DIVISION.
WORKING-STORAGE SECTION.                ···1.
01 CMDN-G ADDRESSED BY A-CMDN.
  02 CMDN PIC X(8).
01 OPT1-G ADDRESSED BY A-OPT1.
  02 OPT1 PIC X(8).
01 OPT2-G ADDRESSED BY A-OPT2.
  02 OPT2 PIC X(8).
              :
LINKAGE SECTION.                        ···2.
01 ARGC PIC 9(9) USAGE COMP.
01 ARGV.
  02 ARGV1 USAGE ADDRESS.
  02 ARGV2 USAGE ADDRESS.
  02 ARGV3 USAGE ADDRESS.
              :
PROCEDURE DIVISION USING                ···3.
                  BY VALUE     ARGC
                  BY REFERENCE ARGV.
    COMPUTE A-CMDN = ARGV1.
    COMPUTE A-OPT1 = ARGV2.
    COMPUTE A-OPT2 = ARGV3.
              :
```

1. 作業場所節（WORKING-STORAGE SECTION）で，アドレスデータ項目で参照するデータを定義します。
2. 連絡節（LINKAGE SECTION）で，受け取る引数の形式を定義します。
3. アドレスデータ項目のアドレス参照を解決し，引数の値を COBOL のデータ項目に格納します。

規則

- COBOL プログラムのコンパイル時に，-Main,System オプションを指定する必要があります。
- 連絡節で定義した受け取る引数の個数と，実際にコマンド行に入力した実行可能ファイル名，および引数の個数が一致していなければなりません。
- 受け取った引数の終端には，NULL 文字（X'00'）が付加されています。COBOL プログラム中で引数の長さなどを求める場合は，この NULL 文字の分を考慮する必要があります。
- 最初の引数を受け取るデータ項目は，「PIC 9(9) USAGE COMP」で定義する必要があります。

(2)　サービスルーチンを使った引数の受け取り

CBLARGC サービスルーチンおよび CBLARGV サービスルーチンを呼び出して，引数を受け取る方法です。

CBLARGC サービスルーチンでコマンド行の引数の個数を取得し，引数の個数だけ CBLARGV サービスルーチンを繰り返し呼び出すことで，すべてのコマンド行の引数を取得できます。

サービスルーチンを使った引数の受け取りの例

```
              :
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE1.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ARGC          PIC 9(9) USAGE COMP.  …1.
01 ARGV.
  02 ARGV-LENGTH PIC 9(4) USAGE COMP.  …2.
  02 ARGV-AREA.                        …3.
    03 ARGV-AREA-C PIC X(1)
       OCCURS 1 TO 100 DEPENDING ON ARGV-LENGTH.
77 CMD-NO        PIC 9(9) USAGE COMP.  …4.
              :
PROCEDURE DIVISION.
    CALL 'CBLARGC' USING BY REFERENCE ARGC.
    IF RETURN-CODE NOT = 0 THEN
       (CBLARGC異常時の処理)
    END-IF.
              :
    MOVE 1 TO CMD-NO.
    PERFORM UNTIL ARGC = 0
      CALL 'CBLARGV' USING BY REFERENCE CMD-NO ARGV
      IF RETURN-CODE NOT = 0 THEN
        (CBLARGV異常時の処理)
      END-IF
      (CBLARGVで受け取った引数の処理)
      ADD 1 TO CMD-NO
      SUBTRACT 1 FROM ARGC
    END-PERFORM.
              :
```

1. CBLARGC サービスルーチンで受け取る引数の個数の領域
2. CBLARGV サービスルーチンで受け取る引数の長さ
3. CBLARGV サービスルーチンで受け取る引数の文字列
4. CBLARGV サービスルーチンに引き渡す引数の順序

規則

- COBOL プログラムのコンパイル時に，-Main,System オプションを指定する必要があります。
- CBLARGC サービスルーチンで取得した値は，実行可能ファイル名と引数の個数の合計値となります。詳細については，「29.4.3　CBLARGC」を参照してください。
- CBLARGV サービスルーチンで引数の文字列を受け取る領域は，100 バイトで定義する必要があります。受け取ったコマンド行の引数が 100 バイト未満の場合，残りの領域は，空白文字で埋められ

ます。また，受け取ったコマンド行の引数が 100 バイトを超える場合は，先頭から 100 バイトまでが有効となり，以降の文字列は切り捨てられます。詳細は，「29.4.4　CBLARGV」を参照してください。

(3)　ACCEPT／DISPLAY 文を使った引数の受け取り

ACCEPT 文や DISPLAY 文を使って，コマンド行に指定された引数の個数や値を取得できます。詳細は，「10.3　コマンド行へのアクセス」を参照してください。

16.2.3　引数の受け取り方法（VOS3 インタフェースに従った形式の場合）

VOS3 インタフェースに従った形式でコマンド行に指定した引数を受け取る方法を，次に示します。

VOS3 インタフェースに従った形式の場合の引数の受け取りの例

```
                :
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE1.
DATA DIVISION.
                :
LINKAGE SECTION.
01 PARM-1.
  02 PLEN     PIC 9(4) USAGE COMP.   …1.
  02 PTAB OCCURS 10 TIMES.           …2.
    03 PTYPE  PIC X(3).
    03 PCOUNT PIC 9(3).
    03 PCOMMA PIC X(1).
PROCEDURE DIVISION USING PARM-1.
    IF PLEN = 0 THEN
       (引数の指定がないときの処理)
    ELSE
       (引数の指定があるときの処理)
    END-IF.
                :
```

1. 受け取った引数の長さが格納されます。
2. 受け取った引数の値が格納されます。

   受け取った引数の長さが PTAB の長さ未満の場合は，PLEN に指定した長さ分のデータだけを保証します。また，受け取った値の長さが PLEN に指定した長さを超える場合は，PTAB の長さ分のデータだけを保証します。

規則

- COBOL プログラムのコンパイル時に，-Main,V3 オプションを指定する必要があります。
- コマンド行中の実行可能ファイル名の後に指定した文字列だけが，引数として渡されます。引数が空白文字を含む場合は，引用符（"）で囲む必要があります。
- 受け取る引数の長さは 100 バイト以内でなければなりません。100 バイトを超える文字列を指定した場合，先頭から 100 バイトまでが有効となります。

16.2.4　外部スイッチ

外部スイッチを使うと，プログラムの実行中に外部（別の実行単位）からプログラムを制御できます。ここでは，外部スイッチの使い方や値の設定方法について説明します。

(1)　外部スイッチとは

COBOL2002 では，外部からの制御情報を受け取るための 0 と 1 から構成される 8 けたの領域があります。この領域の各けたは，プログラムの特殊名段落で記述する外部スイッチ名（UPSI-0，UPSI-1，……，UPSI-7）に対応します。外部スイッチ名と領域上の位置との関係を次に示します。"1"になっている個所がそれぞれの外部スイッチに対応します。

表 16-1　外部スイッチと位置との関係

外部スイッチ名	対応する位置
UPSI-0	10000000
UPSI-1	01000000
UPSI-2	00100000
UPSI-3	00010000
UPSI-4	00001000
UPSI-5	00000100
UPSI-6	00000010
UPSI-7	00000001

(2)　外部スイッチの設定方法

外部スイッチの状態の値は，環境変数 CBLUPSI で設定します。ここでは，値の設定方法について説明します。

(a)　環境変数による外部スイッチの初期化

外部スイッチの状態の値は，プログラムの実行中に初めてスイッチ状態条件を参照したとき，環境変数 CBLUPSI に指定された値の内容で初期化されます。

環境変数 CBLUPSI の設定形式と規則を次に示します。

形式

```
CBLUPSI=文字列
```

規則

- 文字列は，8 けたの 0 または 1 で設定します。8 けたの文字列は，左から順に UPSI-0，UPSI-1，……，UPSI-7 に対応します。

- 指定した文字列が 8 けたを超える場合は，8 けたを超える部分が無視され，8 けたに満たない場合は，文字列の後ろに 0 が仮定されます。また，0 と 1 以外の文字を指定した場合，プログラムが異常終了します。
- この環境変数を設定していない場合，すべての外部スイッチに 0 が仮定されます。
- 環境変数 CBLUPSI は，COBOL 実行単位中で初めて外部スイッチの参照／更新をしたときに，1 回だけ参照されます。

(b) SET 文による外部スイッチの変更

SET 文を使うと，プログラムの実行中に外部スイッチの状態を変更できます。ただし，SET 文で変更した外部スイッチの状態は，現在実行中の実行環境だけで有効であり，環境変数 CBLUPSI の値には反映されません。

(3) 外部スイッチの参照方法

プログラムで外部スイッチを参照するには，特殊名段落で外部スイッチの機能名が参照する外部スイッチのオン状態（on status）やオフ状態（off status）を，条件名に関連づけます。外部スイッチのスイッチ状態は，この条件名を使って調べられます。

外部スイッチの値を変更してプログラムの処理の流れを変更する例を示します。

（例）

次の COBOL プログラムに対して外部スイッチの値を変更します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
                    :
       ENVIRONMENT DIVISION.
                    :
        CONFIGURATION SECTION.
        SPECIAL-NAMES.
            UPSI-0 IS SW0
                   ON STATUS IS ONIND0
                   OFF STATUS IS OFFIND0
            UPSI-3 IS SW3
                   ON STATUS IS ONIND3
                   OFF STATUS IS OFFIND3
                    :
            UPSI-5 OFF STATUS IS OFFS5.
                    :
        INPUT-OUTPUT SECTION.
                    :
       PROCEDURE DIVISION.
                    :
           IF ONIND0 GO TO A   …1.
           ELSE      GO TO B.
                    :
           IF ONIND3 GO TO C   …2.
           ELSE      GO TO D.
                    :
           IF OFFS5  GO TO A   …3.
```

```
           ELSE      GO TO C.
                     :
       A.
                     :
       B.
                     :
       C.
                     :
       D.
                     :
```

環境変数を"CBLUPSI=10010100"と設定すると，おのおのの外部スイッチの状態は次のようになります。

外部スイッチ名	外部スイッチの状態
UPSI-0	1（オン）
UPSI-1	0（オフ）
UPSI-2	0（オフ）
UPSI-3	1（オン）
UPSI-4	0（オフ）
UPSI-5	1（オン）
UPSI-6	0（オフ）
UPSI-7	0（オフ）

外部スイッチの状態が変更されたため，プログラムの 1.，2.，3.の制御は次のようになります。

1. UPSI-0 がオン状態なので制御は手続き A に移ります。
2. UPSI-3 がオン状態なので制御は手続き C に移ります。
3. UPSI-5 がオン状態なので制御は手続き C に移ります。

16.3　COBOL 実行単位の終了

明示的または暗黙的に STOP RUN 文が実行されると，COBOL 実行単位は終了します。このとき，実行単位で使用した COBOL の資源の解放処理※が実行されます。

注※

閉じられていないファイルのクローズ，仮想メモリの解放，カバレージ機能のカウント情報の蓄積などの処理です。

なお，暗黙的に STOP RUN 文が実行されるのは，次の場合です。

- 実行中のプロセス内で，最初に呼ばれた COBOL プログラムで GOBACK 文が実行された場合

16.3.1　実行単位の終了方法

実行単位の終了方法には，次の三つがあります。

- STOP RUN 文によって終了する方法
- 実行単位中で最初に呼ばれた COBOL プログラムの GOBACK 文で終了する方法
- CBLEND サービスルーチンを呼び出して終了する方法

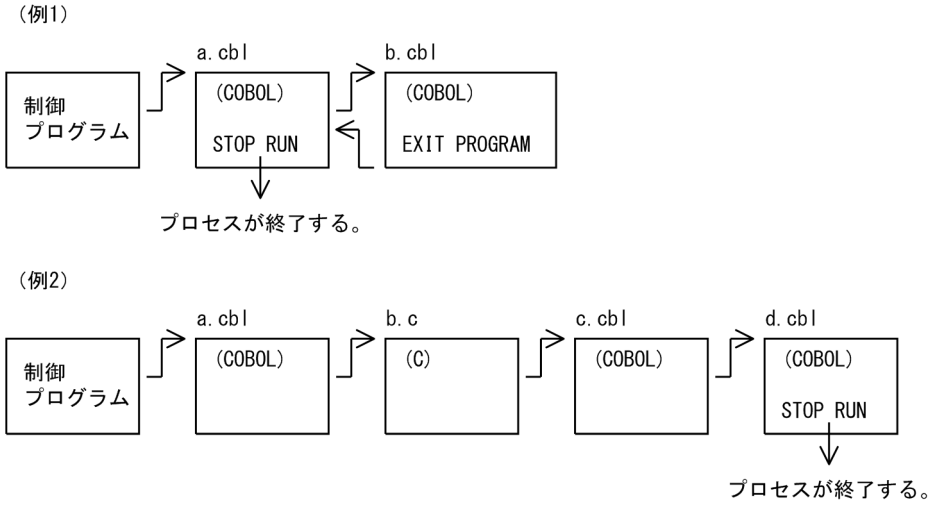
(1)　STOP RUN 文による終了

COBOL プログラムで STOP RUN 文を実行すると，COBOL の実行環境が終了します。

プログラムが終了すると，閉じられていないファイルが閉じられるとともに，実行時に確保した仮想メモリが解放され，プロセスが終了します。

STOP RUN 文による終了の例を次に示します。

図 16–1　STOP RUN 文による終了の例



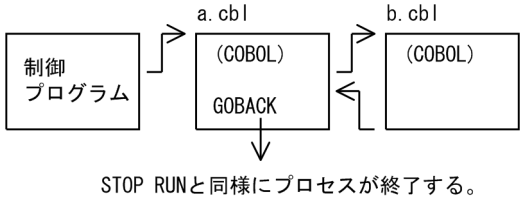
(2) GOBACK 文による終了

実行単位内で最初に呼び出された COBOL プログラムで GOBACK 文を実行すると，COBOL の実行環境が終了します。

プログラムが終了すると，閉じられていないファイルが閉じられるとともに，実行時に確保した仮想メモリが解放され，プロセスが終了します。

GOBACK 文による終了の例を次に示します。

図 16-2 GOBACK 文による終了の例



(3) CBLEND サービスルーチンによる終了

COBOL 副プログラムから他言語の主プログラムへ戻る場合，閉じられていないファイルのクローズ処理や，実行時に確保した仮想メモリの解放が実行されません。これらの COBOL 資源を解放するには，CBLEND サービスルーチンを呼び出す必要があります。

CBLEND サービスルーチンは，COBOL プログラムから呼び出すのではなく，他言語のプログラムから呼び出します。C プログラムから CBLEND サービスルーチンを呼び出す形式を，次に示します。

形式

```
extern int CBLEND();        /* CBLENDの外部参照宣言 */
CBLEND();                   /* CBLENDの呼び出し     */
```

CBLEND サービスルーチンの詳細は，「29.4.1 CBLEND」を参照してください。

(4) 注意事項

- COBOL 以外のプログラムで exit 関数などを発行してプロセスを終了した場合，COBOL2002 は実行環境の終了処理をしないで，OS の終了処理に任せます。
- COBOL プログラムの呼び出し元が制御プログラムでない場合，呼び出し元のプログラムでは，COBOL 実行環境を終了させるために CBLEND サービスルーチンを呼び出さなければなりません。CBLEND サービスルーチンを呼び出さないでほかの COBOL プログラムを呼び出した場合，結果は保証しません。

16.3.2　実行単位の終了コード

COBOL の実行単位が終了するときに返す終了コードについて説明します。

COBOL プログラムから終了コードを返すには，RETURN-CODE 特殊レジスタを使用します。また，プログラムが異常終了した場合は，COBOL2002 によって終了コードが設定される場合があります。

(1)　RETURN-CODE 特殊レジスタを使用する方法

呼び出し元の制御プログラムに終了コードを返すには，プログラムが終了する前に RETURN-CODE 特殊レジスタに値を設定します。

指定例

```
        MOVE 0 TO RETURN-CODE.
```

規則

- RETURN-CODE 特殊レジスタに設定できる値の範囲は，S9(9)の項目に設定できる値の範囲です。
- ただし，このシステムで受け取れる RETURN-CODE 特殊レジスタの値は，下位 1 バイト（8 ビット）の内容です。
- VOS3 COBOL85 との動作と互換性を保つ必要がある場合は，RETURN-CODE 特殊レジスタには，0～4,095 の範囲で設定してください。
- 終了コードの値は，プログラム終了時に RETURN-CODE 特殊レジスタに設定されている値になります。ただし，COBOL プログラム中で RETURN-CODE 特殊レジスタに値を設定した後，RETURNING 指定のない CALL 文を実行した場合は，呼び出し先プログラムの戻り値が終了コードとなります。
- RETURN-CODE 特殊レジスタに値を設定しないで，COBOL プログラムが正常終了した場合は，終了コードとして 0 が返されます。ただし，COBOL プログラムから他言語のプログラムを呼び出している場合，呼び出し先プログラムの戻り値※が終了コードに設定されます。

注※

void 型の C 言語プログラムを最後に呼び出すと，終了コードの値が不定となることがあります。

(2)　プログラムが異常終了した場合の終了コード

プログラムが異常終了した場合，COBOL2002 が終了コードを設定する場合があります。

COBOL 実行時エラーが発生した場合

- -DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange，または-MultiThread オプション指定ありの COBOL プログラムの場合

  終了コードには，1 が設定されます。ただし，実行時環境変数 CBLCORE に 1 を指定した場合，COBOL プログラムは，abort 命令を発行して終了します。このため，プログラムの終了コードには，シグナル SIGIOT 発生時のシステムの終了コードが設定されます。

詳細は，システムのヘッダファイル（signal.h）およびシステムのマニュアルを参照してください。

- 上記のコンパイラオプション指定なしの COBOL プログラムの場合

  COBOL プログラムは，abort 命令を発行して終了します。このため，プログラムの終了コードには，シグナル SIGIOT 発生時のシステムの終了コードが設定されます。

  詳細は，システムのヘッダファイル（signal.h）およびシステムのマニュアルを参照してください。

CBLABN サービスルーチンが実行された場合

- -DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange，または-MultiThread オプション指定ありの COBOL プログラムの場合

  終了コードには，1 が設定されます。ただし，実行時環境変数 CBLCORE に 1 を指定した場合，COBOL プログラムは，abort 命令を発行して終了します。このため，プログラムの終了コードには，シグナル SIGIOT 発生時のシステムの終了コードが設定されます。

  詳細は，システムのヘッダファイル（signal.h）およびシステムのマニュアルを参照してください。

- 上記のコンパイラオプション指定なしの COBOL プログラムの場合

  COBOL プログラムは，abort 命令を発行して終了します。このため，プログラムの終了コードには，シグナル SIGIOT 発生時のシステムの終了コードが設定されます。

  詳細は，システムのヘッダファイル（signal.h）およびシステムのマニュアルを参照してください。

上記以外のエラーが発生した場合

- -DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange，または-MultiThread オプション指定ありの COBOL プログラムの場合

  終了コードには，1 が設定されます。ただし，実行時環境変数 CBLCORE に 1 を指定した場合，COBOL プログラムは，abort 命令を発行して終了します。このため，プログラムの終了コードには，シグナル SIGIOT 発生時のシステムの終了コードが設定されます。

  詳細は，システムのヘッダファイル（signal.h）およびシステムのマニュアルを参照してください。

- 上記のコンパイラオプション指定なしの COBOL プログラムの場合

  終了コードには，発生したシグナル別に OS が返す値が設定されます。

(3) 注意事項

- COBOL プログラムが上位プロセスから起動される場合や，プログラム内でソフトウェアやハードウェアによって検出されて発生したシグナルを処理するプログラムがある場合，プログラムの終了コードは，それぞれ上位プロセス，および例外処理プログラムの戻り値となるため，(1)（2）の規則は適用されません。次に，(1)（2）の規則が適用されない例を示します。

  - テストデバッガのコマンド起動で，COBOL プログラムを実行した場合

    （例）cblcv2k -Execute SAMPLE1

- 終了コードをシェルプログラミングで参照する場合，COBOL プログラムで設定した RETURN-CODE 特殊レジスタの内容を終了コードとして参照できます。ただし，参照できる範囲は，RETURN-CODE 特殊レジスタの下位 8 ビットの内容で，sh（B シェル）は 0～255 です。

17 プログラム間の引数と返却項目

プログラム間，およびメソッド呼び起こしの情報の受け渡しの方法には，CALL 文および INVOKE 文の USING 指定（引数）による方法と RETURNING 指定（返却項目）による方法があります。また，RETURN-CODE 特殊レジスタを使用すると，呼び出し元プログラムに対して復帰コードを返却することもできます。

この章では，引数，返却項目，および復帰コードを使用したプログラム間の情報の受け渡し方法について説明します。

17.1　引数の受け渡し

17.1.1　引数の受け渡しの種類

プログラム間で引数を受け渡す場合は，CALL 文の USING 指定を，メソッドを呼び起こす場合は，INVOKE 文の USING 指定を，それぞれ使用します。

引数の受け渡しの種類

引数の受け渡しには，次の 3 種類があります。受け渡しの種類の指定を省略した場合は，BY REFERENCE が仮定されます。

引数の受け渡しの種類	データの渡し方	仮引数の更新
BY REFERENCE	参照渡し	実引数に反映される。
BY CONTENT	内容渡し	実引数に反映されない。
BY VALUE	値渡し	実引数に反映されない。

それぞれの受け渡し方法の違いについて，次に説明します。

参照渡し（BY REFERENCE 指定）

呼び出し元プログラムのデータ項目を直接参照するアドレスを，呼び出し先プログラムに渡します。呼び出し先プログラムでデータ項目の値を変更すると，呼び出し元プログラムのデータ項目にも反映されます。

内容渡し（BY CONTENT 指定）

呼び出し元プログラムのデータ項目の内容を一時領域に転記し，転記後のデータ項目を参照するアドレスを呼び出し先プログラムに渡します。呼び出し先プログラムでデータ項目の値を変更しても，呼び出し元プログラムのデータ項目には影響しません。

値渡し（BY VALUE 指定）

呼び出し元プログラムのデータ項目の値を，呼び出し先プログラムに渡します。呼び出し先プログラムでデータ項目の値を変更しても，呼び出し元プログラムのデータ項目には影響しません。

引数を値渡しする場合は，呼び出し元プログラムのデータ項目と呼び出し先プログラムのデータ項目での型の対応に注意する必要があります。

17.1.2　使用例

CALL 文の USING 指定による引数の受け渡しの例を次に示します。

呼び出し元プログラム（MAIN1）

```
CALL 'SAMPLE1'
     USING  {BY REFERENCE | BY CONTENT | BY VALUE}  A1 B1 C1.
```

呼び出し先プログラム（SAMPLE1）

```
PROCEDURE DIVISION
     USING  {BY REFERENCE | BY VALUE}  A2 B2 C2.
```

プログラム MAIN1 の A1，B1，C1 がプログラム SAMPLE1 に渡すデータ項目です。また，プログラム SAMPLE1 の A2，B2，C2 がプログラム MAIN1 から渡された情報を受け取るデータ項目です。データ項目 A1，B1，C1 はそれぞれデータ項目 A2，B2，C2 に対応します。

17.1.3　引数の受け渡しの規則

引数の受け渡しの規則について，説明します。

(1)　CALL 文での引数の整合性チェック

プログラムのコンパイル時に引数の適合チェックのオプション（-DebugCompati，または-TDInf）を指定すると，CALL 文でのプログラム間の引数の整合性をチェックできます。ただし，INVOKE 文では有効となりません。

引数の整合性チェックについては，「36.5　プログラム間整合性チェック」を参照してください。

(2)　CALL 文での定数指定の引数

値渡し（BY VALUE 指定）の CALL 文では，引数に定数を指定できます。

- 数字定数，および ZERO は 4 バイトの 2 進形式として設定されます。
- 浮動小数点数字定数は，倍精度浮動小数点形式として設定されます。
- AIX(32)，Linux(x86)の場合，NULL はポインタ（4 バイト 2 進形式）として設定されます。
- AIX(64)，Linux(x64)の場合，NULL はポインタ（8 バイト 2 進形式）として設定されます。

COBOL プログラム間で定数の引数を渡す場合の型の対応について，次に示します。

表 17–1　COBOL プログラム間でのデータ項目の型の対応

受け取り側作用対象	送り出し側作用対象				
	呼び出し元に指定された定数				
	数字定数	英数字定数	浮動小数点数字定数	ZERO	NULL
固定長集団項目	×		×	×	
英字項目		○	×		
英数字項目		○	×		
英数字編集項目			×		

受け取り側作用対象	送り出し側作用対象				
	呼び出し元に指定された定数				
	数字定数	英数字定数	浮動小数点数字定数	ZERO	NULL
外部 10 進項目		○	×		
内部 10 進項目			×		
2 進項目	○		×	○	
数字編集項目			×		
指標データ項目	×※		×	×※	○
オブジェクト参照	×※		×	×※	
外部浮動小数点数字項目			×		
単精度内部浮動小数点数字項目	×	×	×	×	×
倍精度内部浮動小数点数字項目	×	×	○	×	×
アドレスデータ項目	×※		×	×※	○
日本語項目			×		
日本語編集項目			×		
外部ブール項目			×		
内部ブール項目			×		
ポインタ項目	×※		×	×※	○

（凡例）

○：指定できる

空白：受け渡しできるかどうかはプラットフォームに依存するため，動作保証についてはユーザ責任とする

×：指定してはならない（指定したときの結果は保証しない）

注※

AIX(32)，Linux(x86)の場合は「空白」になります。

(3) INVOKE 文での定数指定の引数

値渡し（BY VALUE 指定）の INVOKE 文では，引数に定数を指定できます。

INVOKE 文に指定された実引数と呼び起こされるメソッドの仮引数は，適合していなければなりません。双方が適合している場合，実引数のデータ属性は，仮引数に指定されたデータ属性と同様になります。

(4) INVOKE 文で BY 指定を省略した場合

INVOKE 文で BY 指定を省略した場合，指定された引数の属性，および呼び出し先メソッドに定義された引数の情報によって，仮定される BY 指定が異なります。INVOKE 文で BY 指定を省略した場合に仮定される BY 指定の内容について，次に示します。

- INVOKE 文にクラス名を指定した場合，または指定した一意名のデータ項目が普遍的オブジェクト参照でない場合は，次の表に示す BY 指定が仮定されます。

表 17-2　INVOKE 文で BY 指定を省略した場合に仮定される BY 指定の属性

INVOKE 文で BY 指定を省略した引数の種別		呼ばれるメソッド定義の引数の BY 指定	
		REFERENCE	VALUE
ADDRESS OF 一意名		REFERENCE	REFERENCE
ファクトリ定義，オブジェクト定義		CONTENT	VALUE
プログラム定義 関数定義 メソッド定義	ファイル節※ 作業場所節※ 局所場所節 連絡節 通信節※	REFERENCE	VALUE
	画面節（WINDOW SECTION）※ サブスキーマ節※	CONTENT	VALUE

注※

メソッド定義には指定できません。

- INVOKE 文に指定した一意名のデータ項目が普遍的オブジェクト参照の場合，BY REFERENCE が仮定されます。

17.2　復帰コードと返却項目

呼び出し先のプログラムやメソッドが呼び出し元プログラムに値を返すには，RETURN-CODE 特殊レジスタ（復帰コード）を使用する方法と，RETURNING 指定（返却項目）を使用する方法があります。復帰コードや返却項目は，COBOL プログラムから COBOL プログラムや C 言語のプログラムを呼び出し，呼び出し先プログラムで設定した返却項目を受け取るときに指定します。

規則

COBOL プログラム間で値を返す場合の規則を，次に示します。

呼び出し元プログラム		呼び出し先プログラム／メソッド	
		PROCEDURE DIVISION の指定	
		RETURNING 指定あり	RETURNING 指定なし※
CALL 文または INVOKE 文の指定	RETURNING 指定あり	○	×
	RETURNING 指定なし※	×	○

（凡例）

○：返却項目を正しく受け渡せる

×：返却項目を正しく受け渡せない

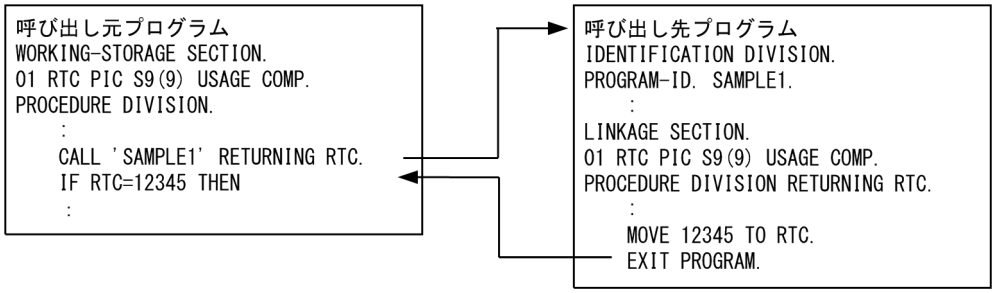
注※

RETURNING 指定がない場合は，RETURN-CODE 特殊レジスタを使用して返却項目を参照または設定します。

- CALL 文または INVOKE 文の RETURNING 指定のデータ項目と，呼び出し先プログラムまたはメソッドの手続き部見出しの RETURNING 指定のデータ項目の長さ，および用途は同じでなければなりません。
- 同じ最外側のプログラムを複数の CALL 文に指定した場合，RETURNING の指定ありと，指定なしが混在してはいけません。また，RETURNING の指定がある場合は，それぞれの RETURNING に指定されたデータ項目の長さ，および用途が同じでなければなりません。
- 最外側のプログラムを呼び出す場合，RETURNING に指定したデータ項目の属性で返却項目を参照します。
- RETURNING 指定でデータ項目に可変長項目を設定した場合，戻り値には常に最大長が受け渡されます。

例

CALL 文の RETURNING 指定による情報の受け渡しの例を，次に示します。



呼び出し元プログラムは，呼び出し先プログラムの返却項目（COBOL プログラムの手続き部見出しの RETURNING に指定したデータ項目）の値を，CALL 文の RETURNING に指定したデータ項目で参照できます。

17.2.1　復帰コードと返却項目の使用方法

復帰コードと返却項目の使用方法について説明します。

(1)　返却項目の規則

規則

- 呼び出し元プログラムと呼び出し先プログラムが両方とも COBOL プログラムの場合，呼び出し先プログラムで手続き部見出しの RETURNING に指定したデータ項目の値が，呼び出し元プログラムの CALL 文 RETURNING に指定したデータ項目に格納されます。このとき，CALL 文の RETURNING 指定のデータ項目と，呼び出し先プログラムの手続き部見出しの RETURNING 指定のデータ項目の長さ，および用途は同じでなければなりません。同じでない場合，動作は保証しません。
- 呼び出し先プログラムで手続き部見出しの RETURNING のデータ項目に設定された戻り値は，呼び出し元プログラムの RETURN-CODE 特殊レジスタでは参照できません。同様に，呼び出し先プログラムの RETURN-CODE 特殊レジスタに設定された復帰コードは，呼び出し元プログラムの RETURNING 指定のデータ項目では参照できません。
- 呼び出し元プログラムと呼び出し先プログラムの RETURNING に指定されたデータ項目の型が異なってはなりません。
- RETURNING でデータ項目に可変長項目を設定した場合，常に最大長のデータが返却項目として受け渡されます。

返却項目の受け渡しの例

CALL 文の RETURNING 指定による返却項目の受け渡しの例を次に示します。

呼び出し元プログラム

```
WORKING-STORAGE SECTION.
01 RTC PIC S9(9) USAGE COMP.
PROCEDURE DIVISION.
    :
    CALL 'SAMPLE1' RETURNING RTC.
```

```
            IF RTC = 12345 THEN
             :
```

呼び出し先プログラム

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
           :
       LINKAGE SECTION.
       01 RTC PIC S9(9) USAGE COMP.
       PROCEDURE DIVISION RETURNING RTC.
           :
           MOVE 12345 TO RTC.
           EXIT PROGRAM.
```

(2) 復帰コードの規則

COBOL プログラムでは，RETURN-CODE 特殊レジスタに値を設定することで，C 言語のプログラムのように復帰コードを設定できます。復帰コードの規則について，次に示します。

(a) 復帰コードの設定方法

呼び出し先プログラムで RETURN-CODE 特殊レジスタに値を設定できます。

呼び出し先プログラムで復帰コードに 0 を設定する例を，次に示します。

（呼び出し先プログラムが COBOL プログラムの場合）

```
MOVE 0 TO RETURN-CODE.
```

（呼び出し先プログラムが C プログラムの場合）

```
return(0);
```

(b) 復帰コードの値

- RETURN-CODE 特殊レジスタに設定できる値の範囲は，S9(9)の項目に設定できる値の範囲です。
- ただし，このシステムで受け取れる RETURN-CODE 特殊レジスタの値は，下位 1 バイト（8 ビット）の内容です。
- VOS3 COBOL85 との動作と互換性を保つ必要がある場合は，RETURN-CODE 特殊レジスタには，0～4,095 の範囲で設定してください。
- RETURN-CODE 特殊レジスタには，初回の呼び出しのときに初期値 0 が設定されます。このため，RETURN-CODE 特殊レジスタを使用しないで正常終了したときは，復帰コードには 0 が設定されています。

(c) 復帰コードの参照方法

呼び出し先プログラムで RETURN-CODE 特殊レジスタに設定された復帰コードを，呼び出し元プログラムで参照する方法を，次に示します。

COBOL プログラムの場合

ほかのプログラムから制御が戻ってきたとき，呼び出し先プログラムで設定した RETURN-CODE 特殊レジスタの値を参照できます。

（例）

```
CALL 'SAMPLE1'.
IF RETURN-CODE = 20 THEN
    :
```

C プログラムの場合

復帰コードは，関数の戻り値として参照できます。

（例）

```
int  rtn_value;
    :
rtn_value  =  SAMPLE1();
if  (rtn_value  ==  20)  {
```

(d) 注意事項

- RETURN-CODE 特殊レジスタの値は，C プログラムとの混在がなく，呼び出し元以降に CALL 文，INVOKE 文，および RETURN-CODE 特殊レジスタの値の設定処理がない場合，COBOL 主プログラムまで引き継がれます。
- 呼び出し元プログラムが COBOL プログラム，呼び出し先プログラムが C プログラムの場合，復帰コードに設定できる値と参照できる値の範囲が次のように異なるので注意が必要です。

  #### C プログラムで設定できる戻り値

  int 型の変数に設定できる値の範囲が，戻り値として指定できます。

  #### COBOL プログラムが RETURN-CODE 特殊レジスタで参照できる値の範囲

  S9(9) COMP で表現できる範囲が，RETURN-CODE 特殊レジスタで参照できます。

18 プログラムの呼び出し

この章では，プログラム間の呼び出し方法の種類，プログラム属性，およびプログラム呼び出しでのリンク方法の違いなどについて説明します。

18.1　プログラム呼び出しの種類と概要

COBOL2002 では，プログラムから別のプログラムを呼び出せます。プログラム中で同じ副プログラムを 2 回以上呼び出し，2 回目以降に制御が渡るとき，呼び出し先プログラムは，直前に制御を戻したときの状態を保持しています。※

注※

ただし，呼び出し先プログラムが CANCEL 文を実行した場合や，初期化属性のプログラムの場合は，2 回目以降の呼び出し時に状態が初期化されて副プログラムが実行されます。

COBOL2002 でプログラムを呼び出すには，CALL 文に呼び出すプログラム名を指定します。このとき，呼び出し先プログラムの PROGRAM-ID 段落で指定したプログラム名は，等価規則が適用されるため，CALL 文に指定するプログラム名も等価規則適用後のプログラム名で指定する必要があります。英小文字を含むプログラム名を定義したい場合は，呼び出し先プログラムの PROGRAM-ID 段落で，プログラム名を英数字定数として指定する必要があります。また，CBL，CLS，CLT，CLU で始まる最外側のプログラム名を指定した場合，動作は保証しません。

CALL 文で呼び出すプログラム名を指定する方法は，定数指定と一意名指定の 2 種類があります。

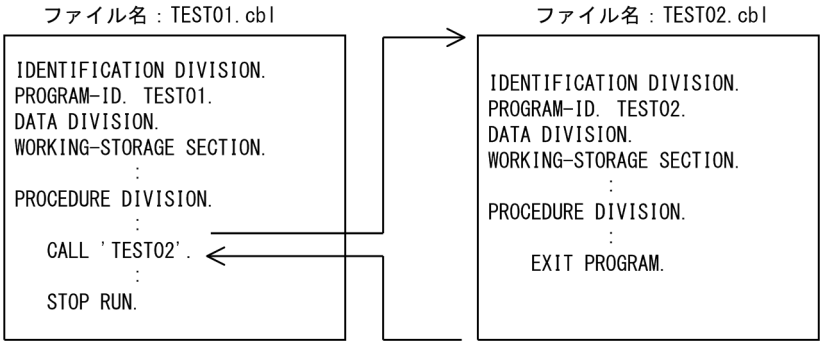
また，プログラム属性が再帰属性プログラム（RECURSIVE 句指定あり）の場合，自分自身のプログラムを再帰的に呼び出せます。再帰属性プログラムの詳細は，「18.4.1　プログラム属性」の「(3)　再帰属性プログラム」を参照してください。

18.1.1　定数指定の CALL 文

定数指定の CALL 文は，CALL 文に呼び出すプログラム名を定数で指定する方法です。

定数指定の CALL 文の場合，呼び出すプログラムがリンク時に静的に解決されるため，実行性能は良くなります。

(例)

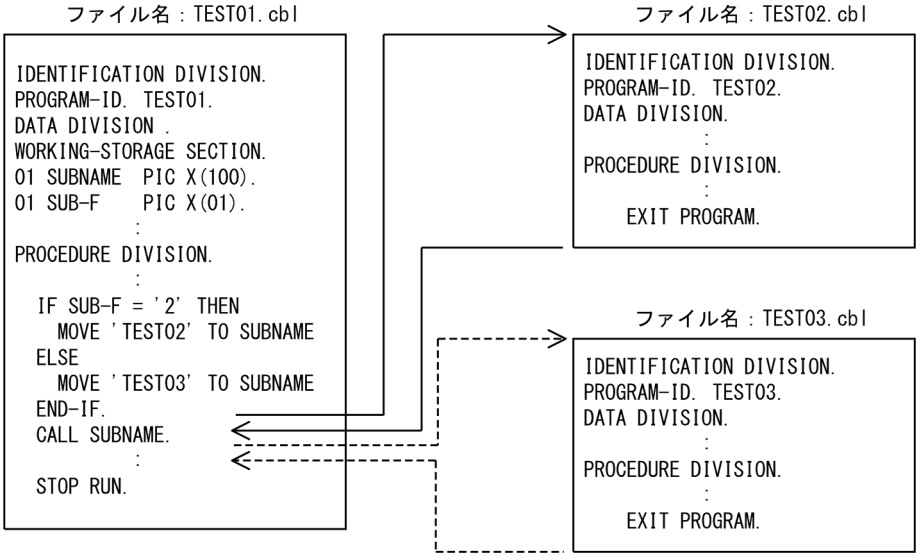


18.1.2　一意名指定の CALL 文

一意名指定の CALL 文は，CALL 文に呼び出すプログラム名を一意名で指定する方法です。一意名指定の CALL 文の場合，呼び出し先のプログラム名は常に実行時に解決されます。

一意名指定の CALL 文は，呼び出し先のプログラム名が実行時に解決されるため，定数指定の CALL 文と比較すると実行性能は劣化します。特に，実行環境中で最初に呼び出すプログラムの場合，そのプログラムの検索に必要な処理を実行するため，最も処理時間が掛かります。ただし，すでに一意名指定で呼び出したプログラムを再び呼び出す場合，COBOL 実行環境が保持しているプログラム情報を検索するので，最初の呼び出しと比べると，処理時間は短縮されます。

(例)



一意名指定の CALL 文で最外側のプログラムを呼び出す場合は，次に示すコンパイラオプションの指定が必要です。

- -DynamicLink
- -IdentCall

18.2　プログラムの取り消し

通常，呼び出し先プログラムが処理を終えて呼び出し元プログラムに戻ったとき，呼び出し先プログラムは，制御を戻したときの状態を保持しています。COBOL2002 では，この状態の保持を取り消せます。

ここでは，プログラムの取り消しについて説明します。

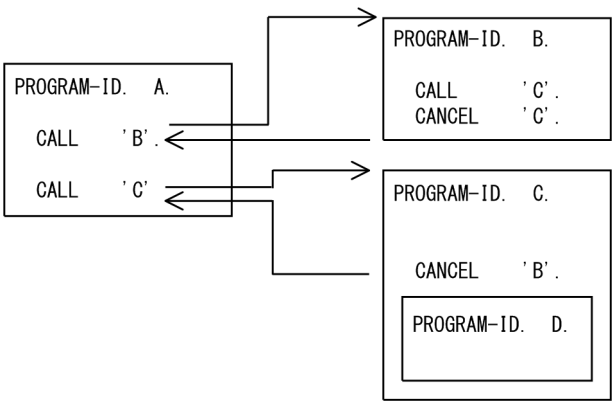
18.2.1　取り消し対象のプログラム

実行中のプログラムを取り消すには，CANCEL 文を使用します。

(1)　取り消しの対象に指定できるプログラム

CANCEL 文で取り消しの対象に指定できるプログラムは，CANCEL 文を実行したプログラムから呼び出せるプログラムだけです。

（例）

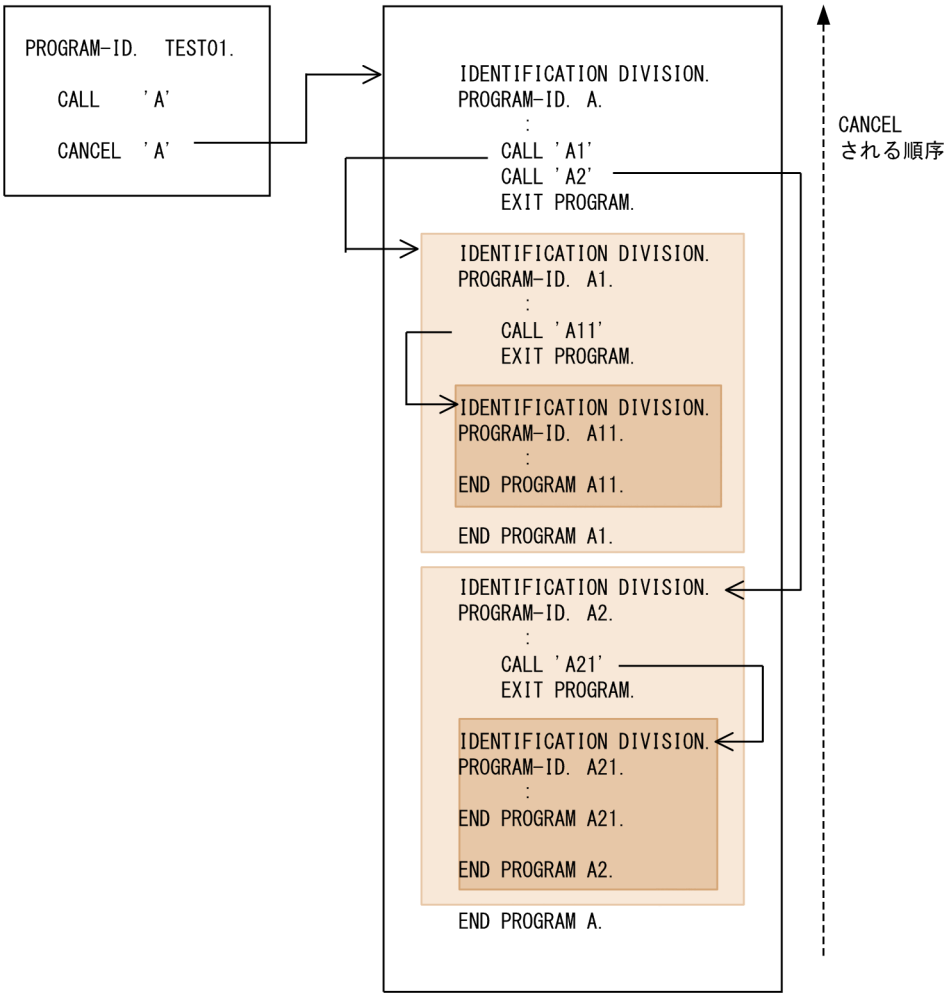


- プログラム A は，自身が呼び出せるプログラム B，およびプログラム C に対して CANCEL 文を実行できます。
- プログラム B，プログラム C がお互いを呼び出せる場合，直接呼び出したプログラムでなくても，プログラム B からプログラム C に対して，プログラム C からプログラム B に対して，それぞれ CANCEL 文を実行できます。
- プログラム D は，プログラム C の内側のプログラムなので，プログラム A からプログラム D に対して CANCEL 文を実行できません。

(2)　取り消しの実行順序

CANCEL 文を実行したとき，CANCEL 文に指定されたプログラム中に含まれるすべてのプログラムも取り消されます。このとき，プログラムが取り消される順番は，翻訳単位のプログラム中に現れた順序とは逆順に，含まれる各プログラムに対して正しい CANCEL 文を実行した場合と同じです。

（例）

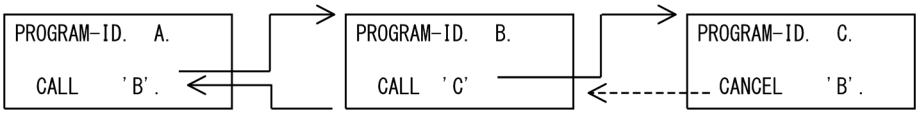


プログラム TEST01 から「CANCEL 'A'.」を実行すると，プログラム A に含まれるすべての内側のプログラムに対しても CANCEL 文が実行されたものとして扱われます。

(3) CANCEL 文に指定できないプログラム

CANCEL 文を実行する場合，すでにほかのプログラムから呼び出されていて，まだ EXIT PROGRAM 文が実行されていないプログラムを，直接または間接に参照してはいけません。このような実行中のプログラムを参照した場合，実行時エラーとなります。

（例）



プログラム C から，まだ実行中のプログラム B を CANCEL 文で取り消そうとした場合，実行時エラーとなります。

18.2.2　取り消しで解放される資源

CANCEL 文の実行によって解放される資源を，次に示します。

1. 閉じていないファイルのクローズ処理（報告書作成機能を使用している報告書に関連づけられたファイルを含む）
2. 通信節による画面機能で開かれた画面やプリンタのクローズ処理（AIX(32)で有効）
3. 最外側プログラムの場合，プログラム内で確保した仮想メモリの解放

注意事項

- 一度も呼び出されていないプログラムに対して CANCEL 文を実行した場合，CANCEL 文は無効となります。
- CANCEL 文で取り消すプログラムに内側のプログラムがある場合，その内側のプログラムも CANCEL 文の対象となります。
- 他言語のプログラムに対して CANCEL 文を実行しても，無効となります。

18.2.3　取り消し後の呼び出し

CANCEL 文実行後のプログラムは，初期状態になっています。初期状態とは，プログラムが実行単位中で最初に呼び出された状態のことで，初期化属性を指定したプログラムの場合と同じ処理が実行されます。

CANCEL 文実行後のプログラムの状態を次の表に示します。

表 18–1　CANCEL 文実行後のプログラムの状態

データ領域の種別	VALUE 句の指定あり	VALUE 句の指定なし	
		CBLVALUE の指定あり	CBLVALUE の指定なし
連絡節	−	−	−
作業場所節	(1)	(2)	不定
局所場所節	(1)	不定	不定
ファイル節	−	(2)	不定
報告書節	(1)	(2)	不定
画面節（SCREEN SECTION）※1	(1)	(2)	不定
画面節（WINDOW SECTION）※1	(1)	(2)	不定
通信節（画面機能）※2	−	不定	不定

データ領域の種別	VALUE 句の指定あり	VALUE 句の指定なし	
		CBLVALUE の指定あり	CBLVALUE の指定なし
通信節（データコミュニケーション機能）	－	不定	不定
サブスキーマ節	－	(2)	不定

(凡例)

－：該当しない

(1)：VALUE 句に指定した値で初期化される

(2)：-MultiThread オプションを指定した最外側のプログラムの再呼び出しをした場合だけ，コンパイラ環境変数 CBLVALUE に指定した値で初期化される。それ以外の場合は，不定となる

注※1

AIX で有効です。

注※2

AIX(32)で有効です。

(説明)

- 作業場所節，報告書節，画面節（SCREEN SECTION）※1，画面節（WINDOW SECTION）※1 に含まれていて，VALUE 句が書かれているデータ項目の場合，VALUE 句に定義された値で初期化されます。VALUE 句が書かれていないデータ項目の場合，初期値は規定されません。

  ただし，-MultiThread オプションおよび-CBLVALUE オプションを指定した最外側のプログラムを取り消したあと，そのプログラムを再び呼び出した場合，作業場所節，報告書節，画面節（SCREEN SECTION）※1，画面節（WINDOW SECTION）※1，ファイル節，サブスキーマ節に含まれているデータ項目は，コンパイラ環境変数 CBLVALUE に指定した値で初期化されます。
- 局所場所節に含まれていて，VALUE 句が書かれているデータ項目の場合，VALUE 句に定義された値で初期化されます。VALUE 句が書かれていないデータ項目の場合，初期値は常に規定されません。
- プログラムに関連する内部ファイル結合子を持つファイルや報告書は，開かれた状態ではありません。
- 通信節による画面機能の送信先画面やプリンタは，開かれた状態ではありません。※2
- プログラム中に含まれるすべての PERFORM 文に対する制御機構は，その初期状態に設定されます。
- 同じプログラム中に含まれる ALTER 文によって参照される GO TO 文は，その初期状態に設定されます。

注※1

AIX で有効です。

注※2

AIX(32)で有効です。

18.3 COBOL主プログラムと副プログラム

COBOL プログラムには，COBOL 主プログラムと COBOL 副プログラムの 2 種類があります。

COBOL 主プログラムは，現在実行中のプロセス内で初めて呼ばれた COBOL プログラムです。COBOL 副プログラムは，COBOL 主プログラムまたは COBOL 副プログラムから呼ばれた COBOL プログラムです。

ここでは，COBOL 主プログラムと COBOL 副プログラムについて説明します。

18.3.1 COBOL プログラムを主プログラムとして動作させる場合

COBOL 主プログラムは，制御プログラムから直接呼ばれる場合と，制御プログラムから間接的に呼ばれる場合（他言語のプログラムから呼ばれる場合）があります。それぞれの場合の動作の違いを，次に示します。

(1) 制御プログラムから直接呼ばれる場合（アプリケーションの主プログラムの場合）

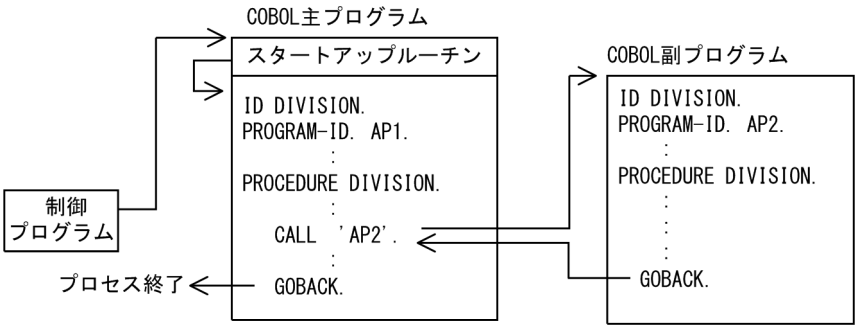
制御プログラムから直接呼ばれる COBOL 主プログラムを作成するには，-Main,System または-Main,V3 オプションを指定してプログラムをコンパイルします。

-Main,System または-Main,V3 オプションを指定してコンパイルした COBOL 主プログラム（-Main オプションが指定されたプログラム）には，スタートアップルーチン※が組み込まれます。制御プログラムから COBOL 主プログラムを呼び出す場合は，まず制御プログラムからスタートアップルーチンが呼ばれ，スタートアップルーチンから COBOL 主プログラムの手続きへと制御が移ります。

注※

スタートアップルーチンとは，main 関数を含む COBOL2002 の実行時ライブラリのことです。

COBOL プログラムが制御プログラムから直接呼ばれる場合の例を次に示します。

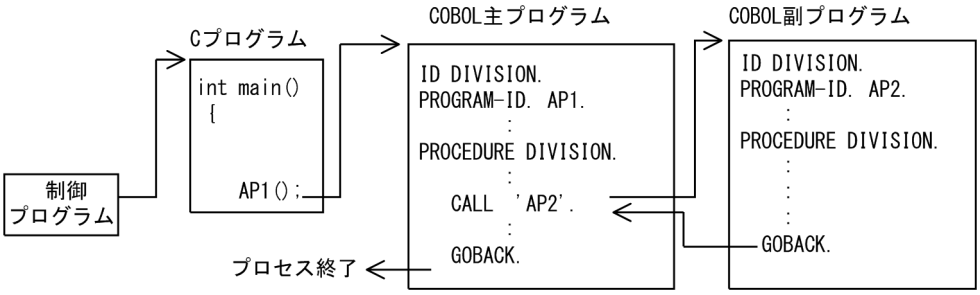


(2)　制御プログラムから間接的に呼ばれる場合

制御プログラムから間接的に呼ばれる COBOL 主プログラムを作成するには，-Main,System および-Main,V3 オプションを指定しないで，プログラムをコンパイルします。

-Main,System および-Main,V3 オプションを指定しないでコンパイルした COBOL 主プログラムには，スタートアップルーチンが組み込まれません。この場合，制御プログラムからはスタートアップルーチンに代わる主プログラムが呼ばれ，主プログラムから COBOL 主プログラムが呼ばれます。

COBOL プログラムが制御プログラムから間接的に呼ばれる場合の例を次に示します。



(3)　注意事項

COBOL プログラムで GOBACK 文を実行すると，STOP RUN 文と同様にプロセスを終了します。したがって，COBOL 主プログラムを呼んだプログラムが制御プログラムでないとき，呼び出し元のプログラムには制御が戻りません。

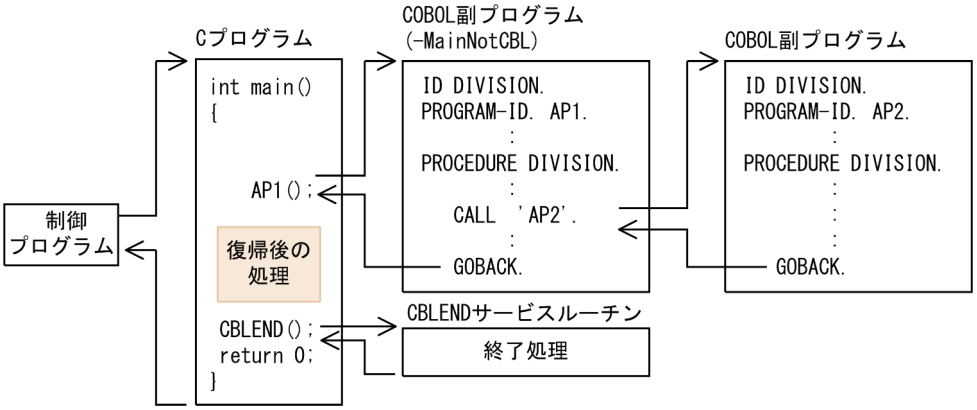
18.3.2　COBOL プログラムを副プログラムとして動作させる場合

「18.3.1　COBOL プログラムを主プログラムとして動作させる場合」に示したように，実行中のプロセス内で初めて呼ばれた COBOL プログラム（COBOL 主プログラム）で GOBACK 文を実行した場合，STOP RUN 文と同じ動作となり，プロセスが終了するため，呼び出し元のプログラムに制御が戻りません。このため，COBOL 主プログラムを呼び出したプログラム（C 言語の主プログラムなど）が COBOL プログラムから復帰したあとの処理を継続できなくなります。

このような場合，-MainNotCBL オプションを指定して COBOL プログラムをコンパイルすると，実行中のプロセス内で初めて呼ばれた COBOL プログラムを COBOL 副プログラムとみなし，GOBACK 文または EXIT PROGRAM 文を実行したときでも呼び出し元のプログラムに復帰できます。

また，-MainNotCBL オプションを指定すると，他言語のプログラムから COBOL プログラムを繰り返し呼び出す場合の COBOL 初期処理／終了処理の負荷を軽減できます。

-MainNotCBL オプションを指定した場合の COBOL プログラムの動作を，次に示します。



上記の場合，最初に呼ばれる COBOL プログラム AP1 に-MainNotCBL オプションを指定しているので，以降に呼ばれる COBOL プログラムは，すべて COBOL 副プログラムとして動作します。そのため，この実行環境内では GOBACK 文や EXIT PROGRAM 文が有効となり，COBOL プログラムの呼び出し元へ制御が戻るため，復帰後の処理ができます。

(1)　COBOL プログラムの資源

COBOL プログラムが呼び出し元のプログラムに戻っても，COBOL の実行環境は解除されません。このため，INITIAL 句を指定していない COBOL プログラムか，または CANCEL 文で取り消されていない COBOL プログラムは，以前に呼ばれた状態を保持しています。保持している情報は，通常属性プログラムがほかのプログラムから 2 回以上呼び出された場合の規則に従います。詳細は，「18.4.1　プログラム属性」の「(1)　通常属性プログラム」を参照してください。

(2)　COBOL 実行環境の終了方法

利用者プログラムは，COBOL プログラムの呼び出しが完了し，以降は呼び出しをしない状態になったとき，次の方法で実行環境を終了します。

- COBOL プログラム内から STOP RUN 文を実行する。
- COBOL プログラムの呼び出し元プログラムから CBLEND サービスルーチンを呼び出す。
  CBLEND サービスルーチンについては，「29.4.1　CBLEND」を参照してください。

(3)　注意事項

-MainNotCBL オプションを指定する場合，該当する実行可能ファイル中に含まれるすべての COBOL プログラムを-MainNotCBL オプション指定でコンパイルする必要があります。-MainNotCBL オプションを指定したプログラムと指定していないプログラムが混在するときは，実行環境中で最初に呼ばれたプログラムの-MainNotCBL オプションの有無に従います。

18.4　プログラム属性

ここでは，COBOL プログラムのプログラム属性について説明します。

18.4.1　プログラム属性

COBOL プログラムでは，プログラムの使用形態に応じて次の属性を指定できます。

- 通常属性プログラム
- 初期化属性プログラム
- 再帰属性プログラム
- 共通属性プログラム

(1)　通常属性プログラム

プログラムの見出し部に特に何も指定しない場合，通常属性プログラムとなります。

通常属性プログラムは，ほかのプログラムから 2 回以上呼び出された場合，前回呼ばれたときの，次の状態を保持しています。

- プログラムに関連するファイルの状態（報告書作成機能を使用している報告書に関連づけられたファイルを含む）
- 画面節（SCREEN SECTION）※1，画面節（WINDOW SECTION）※1，通信節（画面機能）※2，通信節（データコミュニケーション機能），サブスキーマ節で使用された資源の状態
- 作業場所節，報告書節，画面節（SCREEN SECTION）※1，画面節（WINDOW SECTION）※1，ファイル節，サブスキーマ節，通信節（画面機能）※2，通信節（データコミュニケーション機能）の内容
- ALTER 文で設定した GO TO 文

注※1

AIX で有効です。

注※2

AIX(32)で有効です。

ただし，プログラム中に含まれるすべての PERFORM 文に対する制御機構は，プログラムが呼び出されるたびに，初期状態に設定されます。

保持している状態を取り消すには，CANCEL 文を実行します。詳細は，「18.2　プログラムの取り消し」を参照してください。

また，通常属性プログラムは，自分自身を再帰的に呼ぶことはできません。

(2)　初期化属性プログラム

プログラムの見出し部に INITIAL 句を指定した場合，初期化属性プログラムとなります。

初期化属性プログラムは，ほかのプログラムから呼ばれたときに，毎回プログラムの状態が初期化されます。そのため，プログラムの状態は，実行単位でそのプログラムが最初に呼ばれたときと同じになります。

(a)　初期化プログラムが呼ばれたときの初期化の内容

初期化プログラムが呼ばれたときのプログラムの状態を次の表に示します。

初期化プログラムが呼ばれたときの作業場所節の VALUE 句の指定がないデータ項目の初期値については，「(b)　初期化プログラムが呼ばれたときの作業場所節の VALUE 句の指定がないデータ項目の状態」を参照してください。

表 18–2　初期化プログラムが呼ばれたときのプログラムの状態

データ領域の種別	VALUE 句の指定あり	VALUE 句の指定なし	
		CBLVALUE の指定あり	CBLVALUE の指定なし
連絡節	－	－	－
作業場所節	(1)	(2)	不定
局所場所節	(1)	不定	不定
ファイル節	－	(2)	不定
報告書節	(1)	(2)	不定
画面節（SCREEN SECTION）※1	(1)	不定	不定
画面節（WINDOW SECTION）※1	(1)	(2)	不定
通信節（画面機能）※2	－	不定	不定
通信節（データコミュニケーション機能）	－	不定	不定
サブスキーマ節	－	(2)	不定

(凡例)

－：該当しない

(1)：VALUE 句に指定した値で初期化される

(2)：最初に呼ばれたときだけ，コンパイラ環境変数 CBLVALUE に指定した値での初期化は保証されるが，2 回目以降に呼ばれたときは不定となる

注※1

AIX で有効です。

注※2

AIX(32)で有効です。

（説明）

- 作業場所節，報告書節，画面節（SCREEN SECTION）※1，画面節（WINDOW SECTION）※1 に含まれていて，VALUE 句が書かれているデータ項目の場合，VALUE 句に定義された値で初期化されます。VALUE 句が書かれていないデータ項目の場合，初期値は規定されません。
- 局所場所節に含まれていて，VALUE 句が書かれているデータ項目の場合，VALUE 句に定義された値で初期化されます。VALUE 句が書かれていないデータ項目の場合，初期値は常に規定されません。
- プログラムに関連する内部ファイル結合子を持つファイルや報告書は，開かれた状態ではありません。
- 通信節による画面機能の送信先画面やプリンタは，開かれた状態ではありません。※2
- プログラム中に含まれるすべての PERFORM 文に対する制御機構は，その初期状態に設定されます。
- 同じプログラム中に含まれる ALTER 文によって参照される GO TO 文は，その初期状態に設定されます。

注※1

AIX で有効です。

注※2

AIX(32)で有効です。

(b) 初期化プログラムが呼ばれたときの作業場所節の VALUE 句の指定がないデータ項目の状態

コンパイル時にコンパイラ環境変数 CBLINITVALUE があるかどうかで，初期化プログラムの作業場所節の VALUE 句の指定がないデータ項目の状態が異なります。初期化プログラムが呼ばれたときの，作業場所節の VALUE 句の指定がないデータ項目の状態を次の表に示します。

表 18–3 初期化プログラムが呼ばれたときの，作業場所節の VALUE 句の指定がないデータ項目の状態

VALUE 句の指定なし			
CBLINITVALUE の指定あり		CBLINITVALUE の指定なし	
CBLVALUE の指定あり	CBLVALUE の指定なし	CBLVALUE の指定あり	CBLVALUE の指定なし
(1)		(2)	不定

（凡例）

(1)：呼ばれるたびに，作業場所節のデータ項目が NULL（X'00'）で初期化される

(2)：最初に呼ばれたときだけ，コンパイラ環境変数 CBLVALUE に指定した値での初期化が保証されるが，2 回目以降に呼ばれたときは不定となる

(3) 再帰属性プログラム

プログラムの見出し部に RECURSIVE 句を指定した場合，再帰属性プログラムとなります。

再帰属性プログラムは，自分自身を直接的，または間接的に再帰呼び出しできます。

再帰属性プログラムが呼ばれたときのプログラムの状態を次の表に示します。

表 18-4　再帰属性プログラムが呼ばれたときのプログラムの状態

データ領域の種別	VALUE 句の指定あり	VALUE 句の指定なし	
		CBLVALUE の指定あり	CBLVALUE の指定なし
連絡節	－	－	－
作業場所節	(1)	(3)	(4)
局所場所節	(2)	不定	不定
ファイル節	－	(3)	(4)
報告書節	(1)	(3)	(4)
画面節（SCREEN SECTION）※1	(1)	(4)	(4)
画面節（WINDOW SECTION）※1	(1)	(3)	(4)
通信節（画面機能）※2	－	(4)	(4)
通信節（データコミュニケーション機能）	－	(4)	(4)
サブスキーマ節	－	(3)	(4)

(凡例)

－：該当しない

(1)：最初に呼び出された場合だけ，VALUE 句に指定した値で初期化される

(2)：呼び出されるたびに初期化される

(3)：最初に呼び出された場合だけ，コンパイラ環境変数 CBLVALUE に指定した値で初期化される。再帰呼び出しされた場合は，直前に呼び出された状態を保持している

(4)：最初に呼び出された場合は，不定となる。再帰呼び出しされた場合は，直前に呼び出された状態を保持している

注※1

AIX で有効です。

注※2

AIX(32)で有効です。

(説明)

次の個所は，プログラムが呼び出されるたびに初期化されます。

- 局所場所節に含まれていて，VALUE 句が書かれているデータ項目の場合，VALUE 句に定義された値で初期化されます。VALUE 句が書かれていないデータ項目の場合，初期値は常に規定されません。
- プログラム中に含まれるすべての PERFORM 文に対する制御機構は，その初期状態に設定されます。

- 同じプログラム中に含まれる ALTER 文によって参照される GO TO 文は，その初期状態に設定されます。

次の個所は，プログラムが再帰呼び出しされた場合，前回呼ばれた状態を保持しています。

- 作業場所節，報告書節，画面節（SCREEN SECTION）※1，画面節（WINDOW SECTION）※1 に含まれていて，VALUE 句が書かれているデータ項目の場合，最初に呼び出されたときは，VALUE 句に定義された値で初期化されます。VALUE 句が書かれていないデータ項目の場合，最初に呼び出されたときは，初期値は規定されません。
- 作業場所節，ファイル節，報告書節，画面節（SCREEN SECTION）※1，画面節（WINDOW SECTION）※1，通信節（画面機能）※2，通信節（データコミュニケーション機能），サブスキーマ節に含まれているデータ項目で，再帰呼び出しされたときは，直前の状態を保持しています。
- 局所場所節に含まれていて，VALUE 句が書かれているデータ項目の場合，定義された値で初期化されます。VALUE 句が書かれていないデータ項目の場合，初期値は常に規定されません。
- プログラムに関連する内部ファイル結合子を持つファイルや報告書は，開かれた状態で再帰呼び出しをすると，開かれた状態で引き継がれます。
- 画面節（SCREEN SECTION）※1，画面節（WINDOW SECTION）※1，通信節（画面機能）※2，通信節（データコミュニケーション機能），サブスキーマ節で使用された資源の状態は，そのままの状態で引き継がれます。

注※1

AIX で有効です。

注※2

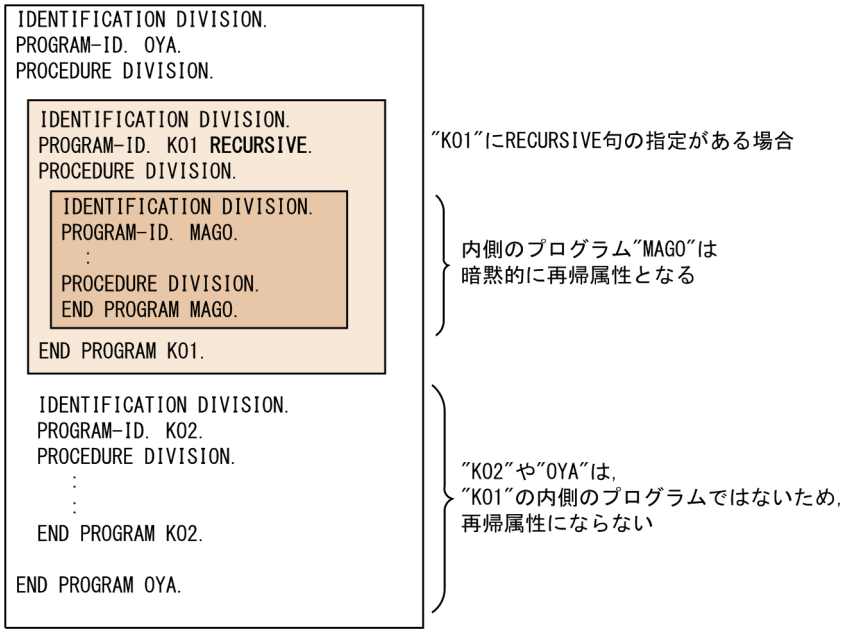
AIX(32)で有効です。

(a) RECURSIVE 句の効果

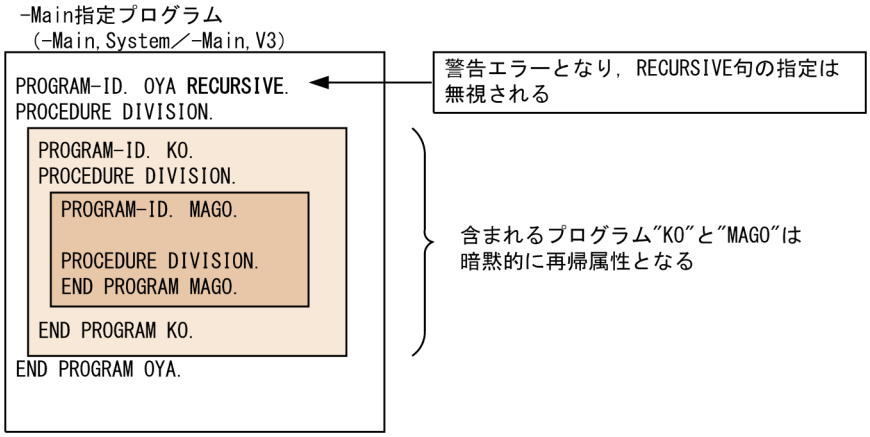
RECURSIVE 句を指定した場合の効果は，次のとおりです。

- RECURSIVE 句は，最外側のプログラム（この場合は，-Main,System／-Main,V3 オプションを指定しないプログラム）でも指定できます。

  また，プログラムが入れ子状態であるとき，外側のプログラムに RECURSIVE 句の指定があれば，その直接または間接的な内側のプログラムに RECURSIVE 句の指定がなくても，暗黙的に再帰属性プログラムとなります。



- -Main,System／-Main,V3 オプションを指定した最外側のプログラムに RECURSIVE 句を指定した場合，警告エラーとなり，最外側のプログラムの RECURSIVE 句の指定は無視されます。ただし，内側のプログラムについては，再帰属性となります。



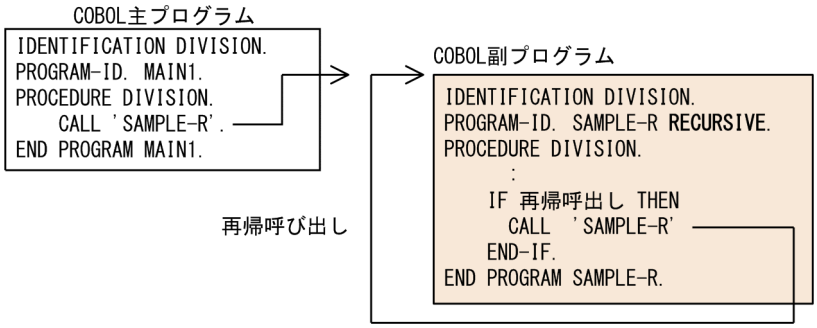
上記の最外側のプログラム（"OYA"）を呼び出そうとした場合，次の結果となります。

- 最外側のプログラムの手続き，または含まれるプログラムの手続きから，CALL 定数（CALL 'OYA'）で呼び出そうとした場合は，コンパイル時にエラーとなります。
- 別翻訳単位のプログラムの手続きから，CALL 定数(CALL 'OYA')で呼び出そうとした場合，-DynamicLink,Call オプションの指定があれば実行時にエラー，-DynamicLink,Call オプションの指定がなければリンク時にリンクエラーとなります。
- CALL 一意名で呼び出そうとした場合は，実行時にエラーとなります。

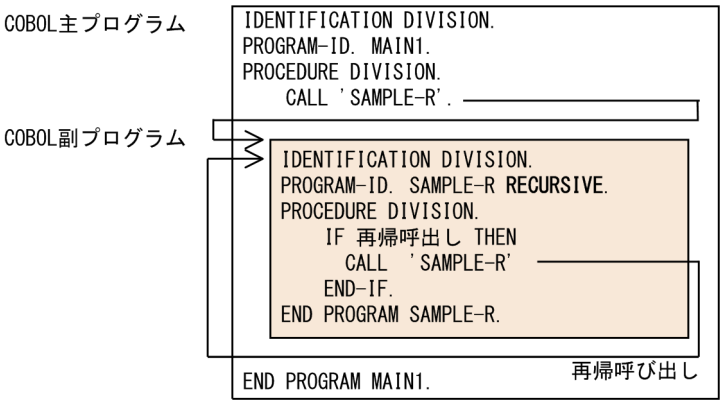
(b)　再帰呼び出しの例

COBOL 副プログラムの再帰呼び出し

（最外側のプログラムの再帰呼び出し）

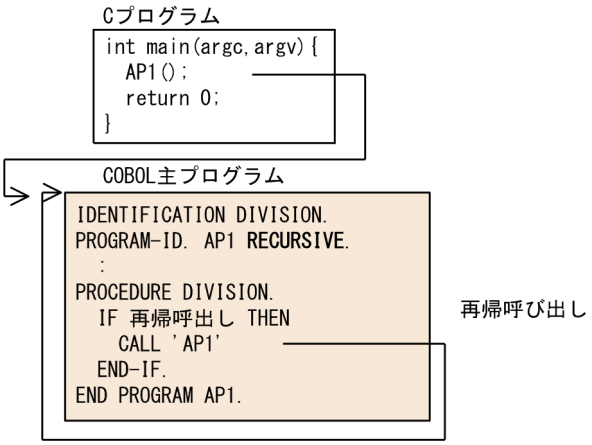


（内側のプログラムの再帰呼び出し）



COBOL 主プログラムの再帰呼び出し

C プログラムなどを介して，制御プログラムから間接的に呼ばれる COBOL 主プログラムについても，再帰呼び出しができます。



(4)　共通属性プログラム

プログラムの見出し部に COMMON 句を指定した場合，共通属性プログラムとなります。

共通属性プログラムは，その共通属性プログラムを直接含まない，内側のプログラムから呼び出せます。

18.5　静的なリンクと動的なリンク

ここでは，プログラム間連絡での静的なリンクと動的なリンクの違いについて説明します。

リンク方法や，実行可能ファイルの作成方法については，「34.　実行可能ファイルと共用ライブラリの作成」を参照してください。また，共用ライブラリの静的なリンクと動的なリンクについては，「18.6.2　共用ライブラリに含まれるプログラムの呼び出し方法」を参照してください。

18.5.1　静的なリンク

静的なリンクとは

静的なリンクとは，オブジェクトファイルのリンク時にプログラムの呼び出しを解決して，実行可能ファイルを生成するリンク方法です。

静的なリンクとは，次の両方のことを指します。

- オブジェクトファイルを静的に結合して，一つの実行可能ファイルを生成すること
- 実行可能ファイルに共用ライブラリをリンクし，呼び出し先プログラム名の名前解決を静的にすること

静的なリンクの長所と短所

静的にリンクすると，主プログラムがメモリにロードされるのと同時に，呼び出し先プログラムもロードされます。そのため，プロセス起動時にはロード時間が掛かりますが，CALL 文でほかのプログラムを呼び出すとき，すでにメモリ上にプログラムがロードされているので，高速に呼び出せます。

静的にリンクした方がよいケース

同じプログラムを何度も呼び出すようなプログラム構造の場合は，静的にリンクした方がプログラムの実行性能が良くなります。

18.5.2　動的なリンク

動的なリンクとは

動的なリンクとは，呼び出し先プログラムの情報を保持しない実行可能ファイルを生成するリンク方法です。

動的なリンクでは，CALL 文での呼び出し先プログラム（副プログラム）と呼び出し元プログラム（主プログラム）とを別ファイルで生成しておきます。呼び出し先プログラムは，呼び出し元プログラムが CALL 文を実行したとき，メモリにロードされます。

動的なリンクの長所と短所

動的にリンクすると，主プログラムがメモリにロードされても，副プログラムはロードされないため，プロセス起動時のロード時間が静的リンクより高速になります。また，プログラムが消費するメモリ空間が少なくて済みます。しかし，CALL 文の実行時に，呼び出し先プログラムのロードと検索処理が実行されるため，CALL 文の処理時間は遅くなります。

動的にリンクした方がよいケース

処理の流れによって呼ばれないことがある副プログラムがある場合は，動的にリンクした方がプログラムの実行性能が良くなります。

18.6 共用ライブラリに含まれるプログラムの呼び出しと共用ライブラリのアンロード

18.6.1 共用ライブラリの概要

主プログラム（呼び出し元プログラム）と副プログラム（呼び出し先プログラム）を別ファイルにして，呼び出し先プログラムを一つのファイルにまとめたものを共用ライブラリといいます。

呼び出し先プログラムを共用ライブラリにすると，次の利点があります。

- 複数のユーザアプリケーションで一つの共用ライブラリを共用できます。このため，メモリ空間が節約でき，スワップの回数も減らせます。
- 関数の引数と戻り値を変更しなければ，共用ライブラリ中の関数を変更しても実行可能ファイルやほかの共用ライブラリの再コンパイルや再リンクは不要です。
- 不要となった共用ライブラリをメモリからアンロードすることによって，メモリを効率良く使用できます。

18.6.2 共用ライブラリに含まれるプログラムの呼び出し方法

共用ライブラリに含まれるプログラムの呼び出しには，静的なリンクによる呼び出しと，動的なリンクによる呼び出しの二つの方法があります。

(1) 静的なリンク

呼び出し元プログラムのリンク時に共用ライブラリを指定して，呼び出し先プログラム名を解決する方法です。この場合，実行可能ファイルがロードされるときに，共用ライブラリもロードされます。

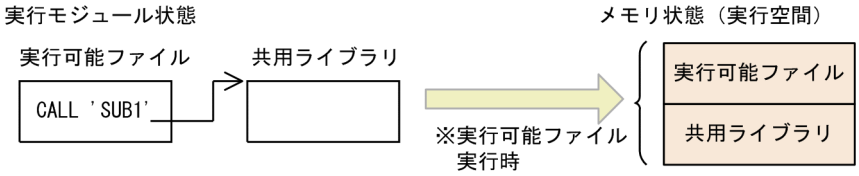
共用ライブラリを静的にリンクするには，呼び出し元プログラムから定数指定の CALL 文で，呼び出し先プログラムの共用ライブラリを呼び出します。

プログラム実行時，リンク時に指定した検索パスに共用ライブラリがないと，プロセス起動エラーとなります。

UNIX の場合，-DynamicLink,Call オプションを指定していない定数指定の CALL 文を実行することで，静的なリンクによって呼び出しを行います。

（例）

副プログラムを含む共用ライブラリが CALL '定数'で呼ばれ，実行可能ファイルの作成時に共用ライブラリを指定してリンクした場合，静的なリンクでの呼び出しとなります。



※実行可能ファイルの実行時に，共用ライブラリも同時にロードされる。

(2)　動的なリンク

実行時に呼び出し先プログラムを検索する方式です。

共用ライブラリを動的にリンクするには，呼び出し元プログラムから-DynamicLink,Call オプションを指定した場合の CALL 文，または-DynamicLink,IdentCall オプションを指定した場合の一意名指定の CALL 文で，呼び出し先プログラムの共用ライブラリを呼び出します。

動的なリンクの場合，CALL 文実行時に動的なリンクによって共用ライブラリがロードされます。

共用ライブラリのロード条件

CALL 文で副プログラムを呼び出すか，または ADDR 関数でプログラムのアドレスを求める場合，次に示すプログラムの優先順位に従ってプログラムが検索され，見つかったプログラムが呼び出されるか，またはアドレスが求められます。プログラムが見つからなかった場合，動的なリンクによって，共用ライブラリがロードされます。

プログラムの優先順位は次のとおりです。

1. 呼び出しできる内側のプログラム
2. 静的にリンクされた最外側のプログラム
3. すでにロードされている共用ライブラリ中のプログラム

実行性能を重視するシステムでは，次に示す環境変数で検索対象となる共用ライブラリを限定することを推奨します。なお，動的なリンクのプレロード機能を使用してプログラム起動時に検索対象となる共用ライブラリをロードしておくことで，CALL 文実行時点でのプログラム検索を高速化することもできます。動的なリンクのプレロード機能については，「18.6.3　動的なリンクのプレロード機能」を参照してください。

環境変数名	概要	対象 OS
CBLLPATH	検索対象となる共用ライブラリのディレクトリを指定します。指定がない場合は，カレントディレクトリだけが検索対象となります。	AIX
	検索対象となる共用ライブラリのディレクトリを指定します。指定がない場合は，共用ライブラリをディレクトリから検索しません。	Linux
CBLLSLIB	検索対象となる共用ライブラリを限定する場合に指定します。	UNIX

環境変数名	概要	対象 OS
	指定がない場合，検索する共用ライブラリは環境変数 CBLLPATH の指定に従います。	
LD_LIBRARY_PATH	環境変数 CBLLSLIB に共用ライブラリ名だけを指定し，ディレクトリを指定していない場合に，検索対象となる共用ライブラリのディレクトリを指定します。 環境変数 CBLLPATH と同時に指定すると，環境変数 CBLLPATH に指定したディレクトリだけが検索対象となります。	Linux
CBLLTAG	検索対象とする共用ライブラリの拡張子，ライブラリ種別，および検索方法を指定します。 指定がない場合は，拡張子（.so）の共用ライブラリおよびアーカイブ形式の共用ライブラリのデータセクション内のシンボル情報を検索対象にします。	AIX

注意事項

- 環境変数 CBLLPATH，CBLLSLIB，CBLLTAG は，実行単位で最初の動的なリンクによる呼び出し時に取得した値が，その実行単位中で有効になります。
- システム環境変数 LD_LIBRARY_PATH は，動的なリンクによる呼び出しをするごとに最新の値が参照されます。

環境変数 CBLLPATH

検索対象となる共用ライブラリのディレクトリを指定します。

形式

```
CBLLPATH=検索ディレクトリ[:検索ディレクトリ]…
```

検索ディレクトリ

動的にリンクする共用ライブラリのディレクトリを指定します。

規則

- 複数ディレクトリを検索対象とする場合，コロン（:）で区切って指定します。
- 動的なリンクの対象となる共用ライブラリは，次のディレクトリにある共用ライブラリから順に，共用ライブラリの登録順で検索されます。

  AIX の場合

  1. 環境変数 CBLLPATH 設定のディレクトリ下の共用ライブラリ
  2. カレントディレクトリ下の共用ライブラリ

  Linux の場合

  1. 環境変数 CBLLPATH 設定のディレクトリ下の共用ライブラリ
- 検索ディレクトリの指定がないとき，共用ライブラリの検索対象は次のとおりです。

  AIX の場合

カレントディレクトリ下の共用ライブラリ

Linux の場合

環境変数 CBLLSLIB の指定があるとき，システム環境変数 LD_LIBRARY_PATH に従います。

- 環境変数 CBLLPATH に指定した各ディレクトリ名称に，ロード対象の共用ライブラリ名称を付加した文字列長は，システムのパス名の長さの制限値を超えてはなりません。

注意事項

- Linux の場合，環境変数 CBLLPATH を指定したときにカレントディレクトリは検索対象として仮定されません。カレントディレクトリを検索対象とするときは，環境変数 CBLLPATH にカレントディレクトリの指定が必要です。
- Linux の場合，環境変数 CBLLPATH を指定しないで動的なリンクによる呼び出しをするときは，環境変数 CBLLSLIB およびシステム環境変数 LD_LIBRARY_PATH を指定してください。

環境変数 CBLLSLIB

検索対象となる共用ライブラリを限定する場合に指定します。指定された共用ライブラリだけを検索対象とします。

形式

```
CBLLSLIB=共用ライブラリ名称[:共用ライブラリ名称]…
```

共用ライブラリ名称

動的にリンクする共用ライブラリの名称を指定します。

規則

- 共用ライブラリの指定方法には，次の方法があります。

  ・共用ライブラリのファイル名称だけを指定する方法

  ・ディレクトリを含めたファイル名称を指定する方法

  ファイル名称だけを指定した場合は，次の環境変数と組み合わせて検索対象のディレクトリを指定できます。

  ・環境変数 CBLLPATH

  ・システム環境変数 LD_LIBRARY_PATH（Linux で有効）

  なお，ディレクトリを含めたファイル名称を指定した場合，これらの環境変数に指定したディレクトリは検索対象になりません。
- ファイル名称だけを指定した場合，カレントディレクトリの共用ライブラリだけが動的なリンクの対象となります。
- 複数の共用ライブラリを検索対象とする場合，コロン（:）で区切って指定します。コロン（:）で区切られた最後の’/’は無視されます。
- 複数の共用ライブラリを検索対象とする場合，指定した順に検索を行うため，指定順序によって検索時間を短縮できます。

注意事項

- Linux の場合，環境変数 CBLLSLIB にファイル名称だけを指定したとき，カレントディレクトリは検索対象になりませんので，必ず，システム環境変数 LD_LIBRARY_PATH または環境変数 CBLLPATH で検索パスを設定してください。
- AIX の場合，アーカイブ形式の共用ライブラリも指定できます。
- 環境変数 CBLLSLIB の指定がない場合，ロード対象の共用ライブラリの検索方法を次に示します。

  AIX のときは，環境変数 CBLLPATH に指定したディレクトリおよびカレントディレクトリから検索します。

  Linux で環境変数 CBLLPATH の指定があるときは，環境変数 CBLLPATH に指定したディレクトリから検索し，環境変数 CBLLPATH の指定がないときは検索する共用ライブラリがないことになります。

システム環境変数 LD_LIBRARY_PATH（Linux で有効）

環境変数 CBLLSLIB に共用ライブラリのファイル名称だけを指定する場合に，システム環境変数 LD_LIBRARY_PATH に共用ライブラリの検索パスを指定します。

システム環境変数 LD_LIBRARY_PATH については，システムの ld コマンドのリファレンスを参照してください。

注意事項

- カレントディレクトリを検索ディレクトリとする場合でも，システム環境変数 LD_LIBRARY_PATH で検索パスを設定してください。
- 環境変数 CBLLPATH と同時に指定すると，環境変数 CBLLPATH に指定したディレクトリだけが検索対象となります。

環境変数 CBLLTAG（AIX で有効）

環境変数 CBLLTAG は，動的なリンクの動作を指示するオプションを指定します。

形式

```
CBLLTAG=オプション[:オプション]
```

オプション：SO／NOSO，ARMBR／NOARMBR／ARMBRLSYM

規則

- 複数のオプションを指定する場合はコロン(:)で区切ります。
- この環境変数に指定できるオプションを次に示します。

  SO／NOSO

  動的なリンクで，検索対象とする共用ライブラリの拡張子を選択できます。

  SO を指定した場合は，拡張子「.so」の共用ライブラリを検索対象とします。このオプションの標準値は SO です。

  NOSO を指定した場合は，拡張子「.so」の共用ライブラリを検索対象としません。ただし，すでにロードされている共用ライブラリには，このオプションは作用しません。すでにロードされている共用ライブラリとは，次に示す共用ライブラリを指します。

・実行可能ファイル作成時にリンクした共用ライブラリ

・前に呼び出したプログラムのライブラリ作成時にリンクした共用ライブラリ

・ロード関数により，すでにロードされている共用ライブラリ

ARMBR／NOARMBR／ARMBRLSYM

動的なリンクで，検索対象とするライブラリ種別や検索方法を選択できます。

ARMBR と ARMBRLSYM を指定した場合は，アーカイブ化された共用ライブラリを検索対象とします。このオプションの標準値は ARMBR です。

NOARMBR を指定した場合は，アーカイブ化された共用ライブラリを検索対象としません。

ARMBR と ARMBRLSYM の違いを次に示します。

・ARMBR：共用ライブラリ中のデータセクションのシンボル情報から呼び出し先プログラムを検索

・ARMBRLSYM：共用ライブラリ中のローダセクションのシンボル情報から呼び出し先プログラムを検索

注　データセクションのシンボル情報は削除される場合があります。

ARMBR／NOARMBR／ARMBRLSYM は，すでにロードされているアーカイブ化された共用ライブラリについても有効となります。そのため，NOARMBR を指定した場合は，実行可能ファイル作成時にリンクしたアーカイブ化された共用ライブラリは検索対象になりません。

ARMBR は，データセクションのシンボル情報から呼び出し先プログラムを検索します。データセクションのシンボル情報には，エクスポートされている（外部参照できる）シンボル情報（関数名，変数名，位置，タイプ，属性情報など）およびエクスポートされていない（内部参照だけの）シンボル情報などが多数格納されています。

ARMBRLSYM は，共用ライブラリ中のローダセクションのシンボル情報から呼び出し先プログラムを検索します。ローダセクションのシンボル情報には，エクスポートされている（外部参照できる）シンボル情報だけが格納されています。

共用ライブラリのファイル構造については，システムのリファレンスなどを参照してください。

検索方法の違いから，ARMBR より ARMBRLSYM を指定する方がプログラム検索が高速になります。ただし，ARMBRLSYM を指定すると ARMBR とは呼び出し先プログラムが異なる場合があります。

アーカイブ化された共用ライブラリが検索ディレクトリに存在するかどうかによって，プログラム検索がより高速となる指定を次に示します。

アーカイブ化された共用ライブラリ	呼び出しプログラム	プログラム検索が高速な指定
検索ディレクトリに存在する	アーカイブ化された共用ライブラリに含まれる	ARMBR より ARMBRLSYM の方が高速となる
	アーカイブ化された共用ライブラリに含まれない	ARMBR より NOARMBR の方が高速となる※1
検索ディレクトリに存在しない	なし	ARMBR より NOARMBR の方が高速となる※2

注※1

検索ディレクトリに存在するアーカイブ化された共用ライブラリ，および静的にリンクされているアーカイブ化された共用ライブラリの数や大きさによっては，NOARMBR よりも ARMBRLSYM を指定した方が，プログラム検索が高速な場合があります。

注※2

静的にリンクされているアーカイブ化された共用ライブラリの数や大きさによっては，NOARMBR よりも ARMBRLSYM を指定した方が，プログラム検索が高速な場合があります。

注意事項

- この環境変数を使用する場合は，プログラムの開発環境と実行環境で，必ず，同じオプションを指定してください。
- 環境変数 CBLLSLIB に拡張子「.so」の共用ライブラリ指定した場合，環境変数 CBLLTAG に NOSO を指定したときでも，環境変数 CBLLSLIB に指定された共用ライブラリは検索対象となります。
- NOSO，NOARMBR オプションを指定する場合は，動的なリンクで呼び出すサブプログラムは，次に示す点に注意して作成してください。

指定するオプション	共用ライブラリファイル作成時の注意点
NOSO オプション	共用ライブラリファイルは ld コマンドで作成し，拡張子は「.a」にしてください。拡張子を「.so」にした場合は，動的なリンクで呼び出すことができません。
NOARMBR オプション	共用ライブラリファイルは，ld コマンドで作成してください。 ld コマンドで作成した共用ライブラリを ar コマンドでアーカイブファイル化した場合は，動的なリンクで呼び出すことができません。

- ARMBRLSYM オプションの指定の有無で，ld コマンドの-s オプションや strip コマンド※でシンボル情報を除去した実行可能ファイルや，共用ライブラリ内のプログラムは動的なリンクでの呼び出し可否が次のように異なります。

ARMBRLSYM オプションの指定	あり	なし
ld コマンドの-s オプションや strip コマンドでシンボル情報を除去した実行可能ファイルや共用ライブラリ内のプログラムの呼び出し	呼び出せる	呼び出せない

ld コマンドの-s オプションや strip コマンドでシンボル情報を除去した実行可能ファイルや共用ライブラリ内のプログラムを呼び出す可能性がある場合は，ARMBRLSYM オプションの指定によって呼び出し先プログラムが変わるおそれがあります。動的なリンクのプログラム検索トレース機能を使用して呼び出し先プログラムが変わらないことを確認することを推奨します。

注※

ld コマンドの-s オプションや strip コマンドは，オブジェクト中のデータセクションのシンボル情報を削除することでオブジェクトサイズを小さくするための機能です。詳細は，ld コマンドおよび strip コマンドのリファレンスを参照してください。

アーカイブ形式の共用ライブラリ（AIX で有効）

共用ライブラリを ar コマンドによってアーカイブファイルにして（アーカイブ形式の共用ライブラリ），共有オブジェクトのメンバとして扱うことができます。ar コマンドの使用例は，「34.　実行可能ファイルと共用ライブラリの作成」を参照してください。

共用ライブラリを動的にリンクする場合，アーカイブ形式の共用ライブラリ中のプログラムを動的なリンクで呼び出せます。

ロード対象

環境変数 CBLLSLIB にアーカイブ形式の共用ライブラリを指定した場合，アーカイブファイルにある共有オブジェクトのメンバがロード対象になります。

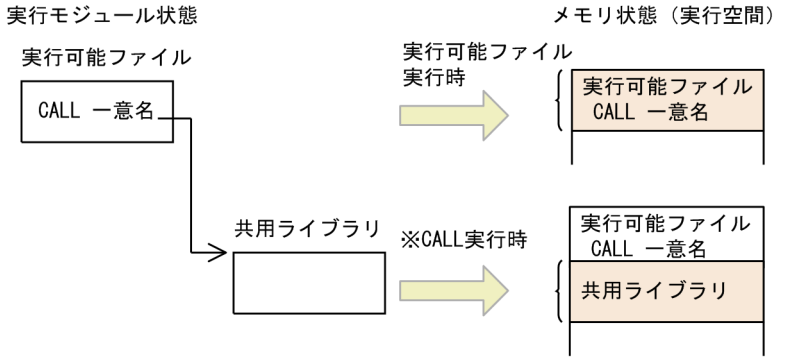
環境変数 CBLLSLIB の指定がない場合，カレントディレクトリおよび環境変数 CBLLPATH で指定したディレクトリ下のすべての共用ライブラリがロード対象となります。また，その共用ライブラリがアーカイブ形式の場合，ファイル中の共有オブジェクトのメンバがロード対象となります。

注意事項

- ネストされたアーカイブは使用できません（ネストされたアーカイブとは，ar コマンドでいったん作成したアーカイブを，さらに ar コマンドでアーカイブ化することです）。
- アーカイブ形式の共用ライブラリに同じ名称のメンバが複数存在する場合は，最初のメンバを対象とします。
- アーカイブ形式の共用ライブラリを静的にリンクした場合は，アーカイブ形式でない共用ライブラリを静的にリンクした場合と同じく，リンク時に指定した共用ライブラリが実行開始時にロードされます。

（例）

一意名指定の CALL 文での呼び出しの場合は，動的なリンクでの呼び出しとなります。



※共用ライブラリは，CALL一意名の実行時にロードされる。

注意事項

- 環境変数の値は，プログラムで最初に呼ばれた CALL 文の実行時に取得した値が，そのプログラム中で有効になります。

(3) 注意事項

- マルチスレッド対応でない COBOL プログラムの場合，動的にロードされた共用ライブラリは，COBOL 実行環境の終了時にアンロードされます。したがって，atexit システム関数などで，COBOL 実行環境の終了後のタイミングで呼び出される関数を登録する場合は，次の点に注意してください。
  - COBOL 実行環境終了後のタイミングで呼び出される関数は，動的にロードされる共用ライブラリに含まれる関数であってはなりません。動的にロードされた共用ライブラリがアンロードされると，その共用ライブラリに静的にリンクされた共用ライブラリもアンロードされます。この場合，静的にリンクされた共用ライブラリに登録する関数についても注意が必要です。
  - 動的にロードされる共用ライブラリのアンロードを抑止するには，環境変数 CBLNO_LIBFREE を使用してください。環境変数 CBLNO_LIBFREE については，「18.6.5　共用ライブラリのアンロード」を参照してください。

(4) プログラムのデバッグ

実行可能ファイルに静的にリンクした COBOL プログラムと同様に，動的にリンクした共用ライブラリについてもテストデバッガを使用したデバッグができます。

(5) 動的なリンクを使用するプログラム作成時の注意事項

プログラムの作成時には次のような注意が必要になります。

- C プログラム，または-DynamicLink オプションを指定しないでコンパイルした呼び出し元プログラムから CALL 文で呼び出し先プログラムを呼び出す場合，動的なリンクは行いません。したがって，呼び出し先プログラムは次のような状態でなければなりません。
  - 主プログラムに静的にリンクされている。
  - すでにロードされた共用ライブラリに含まれる。
  - 呼び出し元プログラムと同じ共用ライブラリに含まれる。

  また，呼び出し先プログラムが動的にロードされた共用ライブラリにある場合，その共用ライブラリはアンロードされる場合があります。この場合，呼び出し元プログラムを再実行したときの動作は保証しません。
- 動的にロードされる共用ライブラリに，静的にリンクされている共用ライブラリに含まれる C プログラムを呼び出す場合，次の注意が必要になります。
  - C プログラムには，外部属性を持つ変数の実体を定義してはならない。

  動的なリンクによってロードされた共用ライブラリは，COBOL の実行環境終了時にアンロードされます。このとき，システムによってアンロード対象の共用ライブラリに静的にリンクされている共用ライブラリもアンロードされます。この場合，外部属性を持つ変数の実体がなくなるため，プログラムが正常に動作しなくなる場合があります。
- C プログラムで外部属性を持つ変数を参照する場合，その変数の実体はすでに実行されたプログラムで定義しなければなりません。

未実行のプログラムで外部属性を持つ変数の実体が定義されている場合，そのプログラムを含む共用ライブラリがアンロードされる場合があります。このため，外部属性を持つ変数の実体がなくなり，プログラムが正常に動作しない場合があります。

- -DynamicLink オプションを指定した COBOL オブジェクトファイルをリンクする場合，-lelf オプションの指定が必要です。ccbl2002 コマンドを使用してリンクまで実行する場合，-DynamicLink オプションを指定していると，ccbl2002 コマンドがリンク時に-lelf オプションを仮定します。
- Linux の場合，実行可能ファイル中に存在するプログラムを動的なリンクで呼び出すときは，実行可能ファイル作成（リンク）時に，-E または-export-dynamic リンカオプションの指定が必要です。

  ccbl2002 コマンドを使用してリンクまで実行する場合，-DynamicLink オプションを指定していると，ccbl2002 コマンドがリンク時に-E リンカオプションを仮定します。

  -E リンカオプションについては，システムのマニュアルを参照してください。
- AIX の場合，動的なリンクを使用時に作業場所節のデータ記述項に EXTERNAL 句を指定し，データを共用するプログラムを使用するときは，実行可能ファイル作成時に，次のリンカオプションを指定してください。
  - -bexpall オプションまたは-bE オプション

    EXTERNAL 指定のデータ項目の名称を外部シンボルとしてエクスポートする場合に指定します。

    すべてのシンボルをエクスポートする場合は，-bexpall オプションを指定します。エクスポートするシンボルを限定する場合は，-bE オプションでエクスポートする EXTERNAL 指定のデータ項目の名称を指定したファイルを指定します。
  - -brtl オプション

    実行時リンクを使用する場合に指定します。

  すべてのシンボルをエクスポートして，EXTERNAL 句を用いたデータの共用を行う指定例を次に示します。

  （例）

  ```
  ccbl2002 -DynamicLink,Call -OutputFile prog1 prog1.cbl -Wl,-bexpall,-brtl
  ```

  リンカオプションについては，システムのマニュアルの ld コマンドに関する説明を参照してください。
- 利用者定義関数を含めた共用ライブラリは，動的なリンク（プログラム呼び出した時にメモリにロードされるリンク方法）では呼び出せません。

プログラムの作成方法については，「34.　実行可能ファイルと共用ライブラリの作成」を参照してください。

(6)　動的なリンクを使用する場合の注意事項

動的なリンクでは，独自に共用ライブラリのロードやアンロードを管理しています。そのため，プログラムの実行中に，共用ライブラリのロードやアンロードに関連するシステム関数（dlopen，dlclose）を，ユーザプログラムから呼び出してはなりません。

18.6.3　動的なリンクのプレロード機能

実行単位内で最初に実行される COBOL プログラムの起動時に，動的なリンクで検索対象となる共用ライブラリをロードすることで，動的なリンクによる CALL 文実行時の呼び出し先プログラムの検索を高速化できます。

(1)　環境変数

動的なリンクのプレロード機能を有効にするには，実行時環境変数 CBLPRELOAD に，プレロードする共用ライブラリ名称を記述したファイル（プレロードリストファイル）名を絶対パスで指定します。

形式

```
CBLPRELOAD=プレロードリストファイル名
```

プレロードリストファイルの形式

- プレロードする共用ライブラリ名称を記述します。アーカイブ形式の共用ライブラリ（アーカイブファイルのメンバ）をプレロードする場合は，「アーカイブファイル名称(メンバ名称)」の形式で指定します。
- 1 行に一つの共用ライブラリ名称を記述します。複数行，記述できます。
- 空行は無視されます。
- 行の先頭がシャープ"#"(X'23')の行はコメント行とみなされます。
- 行の長さが 4,096 バイトを超えた場合，その行は無視されます。
- プレロードリストファイルに記述した文字の扱いを次に示します。

文字	文字の扱い
半角空白および水平タブ文字	次の半角空白文字（X'20'），水平タブ文字（X'09'）は無視されます。 • 行の先頭から共用ライブラリ名称の先頭まで • 共用ライブラリ名称の終端から改行まで
シャープ"#"	行の先頭がシャープ"#"（X'23'）の行はコメント行とみなされます。
上記以外の文字	共用ライブラリ名称の一部とみなされます。

プレロードリストファイルの記述例

```
# プレロードする共用ライブラリ
/tmp/test01.a
# プレロードするアーカイブ形式の共用ライブラリ
/tmp/archive.a(test02.a)
```

規則

- プレロードする共用ライブラリは，実行単位内で最初に実行される COBOL プログラムの起動時にロードします。プレロード機能でロードした共用ライブラリはプロセス終了までアンロードされません。

- プレロードする共用ライブラリ名称は，絶対パスで指定してください。
- プレロード機能でロードした共用ライブラリ内のプログラムは，環境変数 CBLLPATH や CBLLSLIB を指定しなくても動的なリンクによる CALL 文で呼び出せます。
- プレロードリストファイル名に誤りがある，アクセス権がないなどの理由でプレロードリストファイルが開けない場合，警告メッセージが出力されます。このとき，プレロード機能は有効となりません。
- プレロードリストファイルに記述した共用ライブラリのロードに失敗した場合，警告メッセージが出力されて処理が続行されます。

18.6.4 動的なリンクのプログラム検索トレース機能

プログラム検索トレース機能では，次のトレース情報を出力します。

- 動的なリンクによる，プログラム呼び出し時の共用ライブラリの検索情報
- プレロード時の共用ライブラリのロード情報

(1) 環境変数 CBLPGMSEARCHTRC

実行時環境変数 CBLPGMSEARCHTRC には，トレース情報を出力するファイル名を指定します。

形式

```
CBLPGMSEARCHTRC=プログラム検索トレースファイル名
```

規則

- プログラム検索トレースファイル名は，絶対パスで指定してください。
- プログラム検索トレースファイル名のパスに誤りがある，アクセス権がないなどの理由で，プログラム検索トレースファイルが開けない場合，警告メッセージが出力されます。このとき，プログラム検索トレース情報は出力されません。
- 次のどれかに該当する CALL 文では，プログラム検索トレース情報は出力されません。

  ・-DynamicLink オプションの指定がない COBOL プログラムの CALL 文

  ・-DynamicLink,IdentCall オプションを指定した COBOL プログラムの定数指定の CALL 文

  ・-DynamicLink,Call オプションを指定した COBOL プログラムの定数指定の CALL 文による内部プログラム呼び出し

  ・-DynamicLink,Call オプションを指定した COBOL プログラムの定数指定の CALL 文による 2 回目以降のプログラム呼び出し
- 実行時環境変数 CBLPGMSEARCHTRC に指定したプログラム検索トレースファイル名は，次の形式のファイル名に変更されて出力されます。

  [形式]

  ファイル名[_i]_j.拡張子

i：スレッド識別子の値です。マルチスレッド対応 COBOL プログラムのときだけ付加されます。

j：管理番号です。1 または 2 が付加されます。

- プログラム検索トレースファイルのファイルサイズが実行時環境変数 CBLPGMSEARCHTRC_SIZE の指定値を超えた場合，管理番号を切り替えて，ファイルの先頭からトレース情報が出力されます。切り替え後のファイル名と同一名称のファイルが存在する場合，同一名称のファイルは削除されます。必要な場合は，削除される前にバックアップをとってください。
- 複数プロセスが並列で実行される場合，プログラム検索トレースファイル名には一意となる名称を指定する必要があります。同じプログラム検索トレースファイル名を指定し，プログラムを並列で実行した場合のプログラム検索トレースの出力結果は保証しません。
- -DynamicLink,Call オプションを指定した COBOL プログラムで ADDR 関数にプログラム名を指定した場合，ADDR 関数でのプログラム検索情報が CALL 文のプログラム検索トレースとして出力されます。
- ディスクの空き容量不足などで，プログラム検索トレースファイルへのトレース出力中にエラーが発生した場合は，それ以降のトレース情報は出力されません。なお，このとき，実行時メッセージは出力されません。

(2) 環境変数 CBLPGMSEARCHTRC_SIZE

実行時環境変数 CBLPGMSEARCHTRC_SIZE には，プログラム検索トレースファイル名を切り替えるサイズを指定します。

形式

```
CBLPGMSEARCHTRC_SIZE=プログラム検索トレースファイル名を切り替えるサイズ(1～2,000,000)
```

規則

- ファイルサイズは KB 単位で指定してください。
- 指定に誤りがある場合，または指定がない場合は 10,240 が仮定されます。
- 実行時環境変数 CBLPGMSEARCHTRC_SIZE の指定に従い，プログラム検索トレースファイルのサイズが指定値の上限を超えた場合，ファイル名の管理番号が切り替えられます。

  切り替え先のファイルが開けないときは，警告メッセージが出力されます。なお，それ以降のプログラム検索トレースは出力されません。
- プログラム検索トレースファイルの管理番号 1 のファイルが存在しない場合は，管理番号 1 のファイルにトレース情報が出力されます。管理番号 1 のファイルがすでに存在する場合は，次に示す管理番号のファイルにトレース情報が出力されます。

管理番号 1 ファイル	管理番号 2 のファイル	トレース情報出力
ファイルサイズが上限を超えていない	−	管理番号 1 のファイルに追加書きで出力する
ファイルサイズが上限を超えている	ファイルが存在しない	管理番号 2 のファイルを作成して出力する

管理番号 1 ファイル	管理番号 2 のファイル		トレース情報出力
	ファイルが存在する	ファイルサイズが上限を超えていない	管理番号 2 のファイルに追加書きで出力する
		ファイルサイズが上限を超えている	最終更新日時が古い管理番号のファイルを削除，作成して出力する

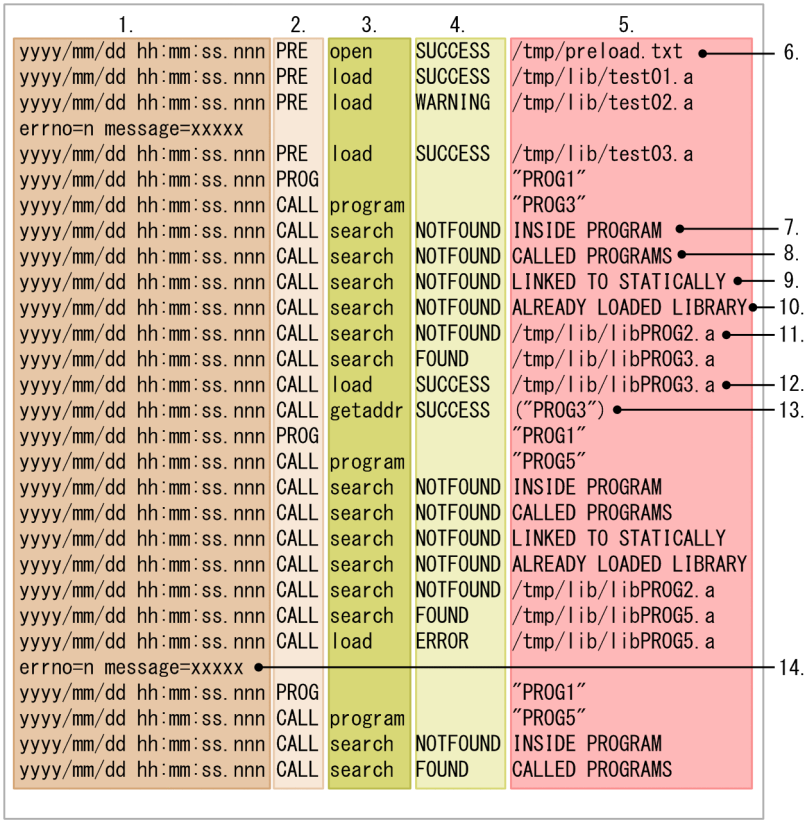
（凡例）

－：条件なし

(3) 注意事項

- プログラム検索トレース機能機能はデバッグ機能です。プログラム検索トレース機能を使用すると，動的なリンクの実行性能が低下するため，デバッグ時にだけ使用してください。
- Linux の場合，プログラム検索トレースにディレクトリを含まない共用ライブラリ名称が出力されているとき，検索対象の共用ライブラリが存在するディレクトリは，システム環境変数 LD_LIBRARY_PATH に指定されたパスを先頭から順に確認することで特定できます。

(4) プログラム検索トレースの内容



1. 日付/時刻を出力する。
2. トレース出力の契機を出力する。

PRE	環境変数 CBLPRELOAD で指定されたプレロードリストファイルに記述された共用ライブラリの情報を出力する。
PROG	CALL 文が実行されたプログラム名を 5.に示す。
CALL	CALL 文の処理であることを示す。3.が program の場合，呼び出すプログラム名を 5.に示す。

3. 処理内容の詳細を出力する。

open	ファイルのオープン処理であることを示す。
load	共用ライブラリのロード処理であることを示す。
program	呼び出し先プログラムの検索処理を開始したことを示す。
search	プログラムの検索処理であることを示す。
getaddr	プログラムのアドレス取得処理であることを示す。

4. 処理の実行結果を出力する。

SUCCESS	処理が成功したことを示す。
WARNING	処理中にエラーが発生したことを示す。処理は継続する。
ERROR	処理中にエラーが発生したことを示す。処理は終了する。
NOTFOUND	処理を実行した結果，呼び出し先プログラムが見つからなかったことを示す。
FOUND	処理を実行した結果，呼び出し先プログラムが見つかり，プログラム検索処理が終了したことを示す。

5. 処理の対象ファイルまたは対象プログラムを出力する。

INSIDE PROGRAM	呼び出しできる内側のプログラムに対する処理であることを示す。
CALLED PROGRAMS	呼び出し済みプログラムに対する処理であることを示す。
LINKED TO STATICALLY	静的にリンクされた最外側のプログラムに対する処理であることを示す。
ALREADY LOADED LIBRARY	ロード済みの共用ライブラリに対する処理であることを示す。
上記以外(ファイル名またはプログラム名)	ファイルまたはプログラムに対する処理であることを示す。

6. 環境変数 CBLPRELOAD の指定がある場合は，プレロードリストファイルに記述された共用ライブラリに対するロード結果を出力する。ロード失敗時は，エラー番号(n)と詳細メッセージ(xxxxx)を出力する。

   なお，エラー原因によっては，詳細メッセージが出力されない場合がある。また，Linux の場合のエラー番号は不定である。

   環境変数 CBLPRELOAD の指定がない場合，環境変数 CBLPRELOAD の情報は出力しない。

7. 呼び出しできる内側のプログラムの中から呼び出し先プログラムを検索した結果を出力する。この情報は，一意名指定の CALL 文のときだけ出力する。

8. 動的なリンクによってすでに呼び出しされたプログラムから呼び出し先プログラムを検索した結果を出力する。

9. 静的にリンクされた最外側のプログラムから呼び出し先プログラムを検索した結果を出力する。ただし，Linux の場合，9.と 10.を同時に処理するため，10.の出力だけとなる。
10. すでにロードされている共用ライブラリ中から呼び出し先プログラムを検索した結果を出力する。なお，Linux の場合，9.の静的にリンクされた最外側のプログラムからの検索処理を含む。
11. 未ロードの共用ライブラリから呼び出し先プログラムを検索した結果を出力する。

    AIX の場合，環境変数 CBLLTAG に NOARMBR が指定されていない環境では，すでにロードされているシステムの共用ライブラリなどを検索した結果も出力する。
12. 呼び出し先プログラムを含む共用ライブラリをロードした結果を出力する。
13. 呼び出し先プログラムのアドレスを取得した結果を出力する。
14. 共用ライブラリのロードに失敗した場合，エラー番号と詳細メッセージを出力する。なお，エラー原因によっては，詳細メッセージが出力されない場合がある。また，Linux の場合のエラー番号は不定である。

18.6.5 共用ライブラリのアンロード

動的なリンクによって呼び出された共用ライブラリだけがアンロードされます。

(1) アンロード条件

動的にロードされた共用ライブラリは，COBOL の実行環境の終了時にアンロードされます。

注意事項

次の場合，共用ライブラリはアンロードされません。

- マルチスレッド対応 COBOL プログラムの場合

(2) アンロードの抑止

環境変数 CBLNO_LIBFREE を使用すると，動的にロードされた共用ライブラリのアンロードを抑止できます。

環境変数 CBLNO_LIBFREE

COBOL 実行環境の終了時に，動的にロードされた共用ライブラリのアンロードを抑止します。

形式

CBLNO_LIBFREE=EXIT

規則

動的にロードされた共用ライブラリは，COBOL 実行環境の終了時にアンロードされます。atexit システム関数などのように，COBOL 実行環境の終了後のタイミングで呼び出される関数を，共用ライブラリに登録する場合に，環境変数 CBLNO_LIBFREE を使用し，関数の呼び出しに失敗し異常終了することを回避します。

注意事項

- 環境変数 CBLNO_LIBFREE に EXIT を指定した場合，動的にロードされたすべての共用ライブラリが対象となります。

18.7　実行可能ファイルの呼び出し

CALL 文で実行可能ファイルの呼び出しができます。

実行可能ファイルとは，ccbl2002 コマンドなどで作成した実行可能ファイルやシェルのように，単独で実行できるファイルのことです。

18.7.1　実行可能ファイル呼び出しの概要

CALL 文で呼び出すプログラムの名称中にファイル拡張子の区切り文字のピリオド（.）が含まれる場合（例：a.out），実行可能ファイルが指定されたものとみなされます。

注意事項

実行可能ファイルのファイル名を引用符（ " ）で囲むと，引用符（ " ）もファイル名の一部として認識されます。

（例）

「PROG.1 ZIKKOU.out」というファイルがあった場合

```
・CALL ’ PROG.1 ZIKKOU.out ’
    →   「PROG.1 ZIKKOU.out」が呼び出される。

・CALL ’ ”PROG.1 ZIKKOU.out” ’
    →   「”PROG.1 ZIKKOU.out”」と認識され，実行時エラーになる。
```

18.7.2　実行方式

CALL 文で実行可能ファイルを指定すると，新しいプロセスが起動し，実行可能ファイルは，そのプロセスで実行されます。呼び出し元のプログラムは，実行可能ファイルの終了後，次の処理に移ります。

新しいプロセスが起動できない場合，CALL 文に ON EXCEPTION または ON OVERFLOW の指定があれば，この指定が有効となります。指定がなければ，共通例外処理で定義した処理を実行します。共通例外処理を使用しないときは，エラーメッセージが出力され，実行単位が異常終了します。

新しいプロセスのカレントディレクトリや環境変数などの実行環境は，呼び出し元のプログラムの実行環境が引き継がれます。

外部属性を持つデータ項目のようなプログラム内で定義した情報は，呼び出し元プログラムと呼び出し先プログラムとでプロセスが異なるため共用はできません。

18.7.3　実行可能ファイルの指定

(1)　実行可能ファイル名の指定方法

実行可能ファイル名は，CALL 定数，CALL 一意名のどちらでも指定できます。実行可能ファイルの指定方法には次の二つの方法があります。

- ディレクトリを含めたパス名で指定する。
- ディレクトリを含めないファイル名だけで指定する。

どちらの場合も，ファイル名は拡張子を含めて指定します。拡張子のないファイルを指定するときは，ファイル名のあとに区切り文字のピリオド（.）を付けて指定します。

ファイル名がピリオド（.）で終わる場合，ファイル名の最後のピリオド（.）のあとに，さらに実行可能ファイルの呼び出しであることを示す区切り文字のピリオド（.）を指定します。

(2)　ファイルの検索順序

パス名で指定した場合，実行されるファイルは，指定したディレクトリのファイルとなります。

ファイル名だけを指定した場合は，環境変数 PATH 内に登録されているディレクトリから検索され，最初に見つかったファイルが実行されます。

1. 環境変数 PATH 内に登録されているディレクトリ

18.7.4　引数の受け渡し

実行可能ファイルが利用者引数（例えば，a.out AAA として実行した場合の AAA）を必要とする場合，実行可能ファイルに引数を渡すには次の方法があります。

(1)　USING 一意名による方法

USING 一意名を指定すると，一意名に指定した値は，そのままの利用者引数となります。

（例）

```
01 PARM PIC X(3) VALUE 'AAA'.
    :
CALL 'a.out' USING PARM.
```

この場合，「a.out AAA」という形で a.out が起動します。

複数の引数を渡す場合は，複数の一意名に分けて指定します。

なお，BY REFERENCE／BY VALUE／BY CONTENT などの BY～指定に関係なく，一意名に格納された値を引数として渡します。

18.7.5　実行可能ファイルの終了コードの取得

CALL 文で実行可能ファイルを呼び出す場合，実行可能ファイルの終了コードは，呼び出し元の RETURN-CODE 特殊レジスタ，または CALL 文の RETURNING 指定で参照できます。CALL 文に RETURNING 指定をした場合は，RETURN-CODE 特殊レジスタでは参照できません。

上記のどちらの場合でも，UNIX では利用しているシステムの仕様に従って，1 バイト（8 ビット）で表現できる値が取得できます。

詳細は，「17.2　復帰コードと返却項目」を参照してください。

18.7.6　実行可能ファイルを呼び出す場合の注意事項

実行可能ファイルを呼び出す場合，次の点に注意する必要があります。

- 実行可能ファイルの呼び出しでエラーが発生した場合，SIGUSR1 シグナルが発生します。このため，次のような処理をする場合で，SIGUSR1 シグナルの変更があるときは，デフォルト状態へ変更するために SIG_DFL の割り当てを行う必要があります。
  - 実行可能ファイルを呼び出す COBOL プログラムを，ほかのプログラムから呼び出す場合
  - ほかのプログラムから，実行可能ファイルを呼び出す COBOL プログラムに戻る場合

19 他言語とのプログラム間連絡

この章では，COBOL プログラムとほかの言語で作成したプログラムとを連携させる方法について説明します。

19.1 C 言語との連携

19.1.1 概要

(1) COBOL プログラム，C プログラム間でのプログラムの呼び出し

COBOL プログラムから C プログラムを呼び出したり，逆に C プログラムから COBOL プログラムを呼び出したりできます。

ただし，COBOL プログラムを呼び出す C プログラムや，COBOL プログラムから呼び出される C プログラムは，システムの規則に従って作成する必要があります。

(2) COBOL プログラム，C プログラム間での引数の引き渡し方法

COBOL プログラムと C プログラムとの間で引数を引き渡すには次の方法があります。

- ポインタ型で引き渡す方法
- 値渡しで引き渡す方法
- CALL 文の引数に ADDRESS OF／LENGTH OF 一意名を指定する方法（ただし，COBOL から C を呼ぶ場合だけ）

(3) 64bit アプリケーション作成時の注意事項（AIX(64)，Linux(x64)で有効）

C プログラムは，64bit アプリケーションとして作成しなければならず，システムによっては C コンパイラのオプションが必要な場合があります。C コンパイラのオプションについては，システムのマニュアルを参照してください。64bit アプリケーションでは，ポインタ型および long 型のサイズは 8 バイトになります。

(4) 他言語で標準出力や標準エラー出力をする場合の注意事項

標準出力や標準エラー出力をしている他言語プログラムのあとに呼び出される COBOL プログラムで，DISPLAY 文や実行時メッセージを標準出力や標準エラー出力に出力している場合，データの出力順が前後することがありますので，COBOL プログラムを呼び出す前に標準出力や標準エラー出力のバッファをフラッシュしてください。

(5) Linux(x64) COBOL2002 コンパイル時の注意

Linux(x64) COBOL2002 は，スモールコードモデルのオブジェクトコードを生成します。したがって，COBOL プログラムと連携させる C 言語プログラムはスモールコードモデルでコンパイルされている必要があります。スモールコードモデルの詳細および C 言語プログラムをスモールコードモデルでコンパイルする方法については，システムのマニュアルを参照してください。

19.1.2　C プログラムから COBOL プログラムを呼び出す方法

C プログラムから COBOL プログラムを呼ぶときの規則を示します。

(1)　C から COBOL を呼ぶときの規則

- C プログラム内で宣言された外部変数，および COBOL プログラム内の外部属性を持つデータ項目は，それぞれのプログラムで参照できます。外部属性を持つデータ項目の参照方法については，「19.1.5 外部属性を持つデータ項目の共用」を参照してください。
- C プログラムは，COBOL プログラムの RETURN-CODE 特殊レジスタに設定された戻り値を関数の戻り値として参照できます。COBOL プログラムが返す戻り値の有効な範囲については，「17.2　復帰コードと返却項目」を参照してください。
- C プログラムから COBOL プログラムを呼び出す場合，COBOL 主プログラムと COBOL 副プログラムによって GOBACK 文，EXIT PROGRAM 文の動作が異なります。詳細については，「16.3 COBOL 実行単位の終了」を参照してください。
- C プログラムから COBOL プログラムを呼び出して，浮動小数点型データの引数渡しをする場合，C プログラムでプロトタイプ宣言を行い，データ型を明確にする必要があります。また，COBOL プログラムの受け取り側作用対象は，次の規則に従う必要があります。
  1. 単精度浮動小数点型データの引数渡しをする場合，受け取り側作用対象の属性は COMP-1 とする。
  2. 倍精度浮動小数点型データの引数渡しをする場合，受け取り側作用対象の属性は COMP-2 とする。
- 手続き部見出しに RETURNING 指定がある場合，COBOL プログラムの返却項目は，そのデータ項目に対応した C 言語のデータ型にしなければなりません。RETURNING 指定がない場合は int 型にしなければなりません。
- C プログラムでは，COBOL プログラムの手続き部見出しの RETURNING のデータ項目（返却項目）に設定した戻り値を参照できます。RETURNING 指定がない場合は，RETURN-CODE 特殊レジスタを参照します。
- COBOL プログラムに-MultiThread オプションを指定した場合，システムのスレッドライブラリを指定する必要があります。詳細は，「34.　実行可能ファイルと共用ライブラリの作成」を参照してください。マルチスレッド対応 COBOL プログラムについては，「26.　マルチスレッド環境での実行」を参照してください。
- 利用者定義関数やメソッド定義は，C プログラムから呼び出せません。

(2)　COBOL のデータ項目と C プログラムの型の対応

COBOL と C のデータ型の対応を，次に示します。引数や戻り値などでデータを受け渡す場合は，この規則に従う必要があります。

表 19-1　COBOL のデータ項目と C プログラムの型の対応

COBOL	C											
	char	unsigned char	short	unsigned short	int, long	unsigned int, unsigned long	long long	unsigned long long	float	double	ポインタ	構造体
固定長集団項目									×	×		○※1
可変長集団項目									×	×		×
英字項目	○※2	○※2							×	×		
英数字項目	○※2	○※2							×	×		
英数字編集項目									×	×		
外部 10 進項目	○※3	○※3							×	×		
内部 10 進項目									×	×		
2 進項目	○※4 ※15	○※4	○※5	○※5	○※6 ※11	○※6 ※11	○※13	○※13	×	×		
COMP-X	○※7	○※7	○※8	○※8	○※9 ※12	○※9 ※12	○※14	○※14	×	×		
数字編集項目									×	×		
外部浮動小数点数字項目									×	×		
単精度内部浮動小数点数字項目	×	×	×	×	×	×	×	×	○※10	×	×	×
倍精度内部浮動小数点数字項目	×	×	×	×	×	×	×	×	×	○※10	×	×
アドレスデータ項目									×	×	○	
指標データ項目									×	×		
日本語項目									×	×		
日本語編集項目									×	×		
外部ブール項目									×	×		
内部ブール項目									×	×		
英数字定数	○※2	○※2							×	×		
数字定数					○				×	×		

COBOL	C											
	char	unsigned char	short	unsigned short	int, long	unsigned int, unsigned long	long long	unsigned long long	float	double	ポインタ	構造体
浮動小数点数字定数	×	×	×	×	×	×	×	×	×	○	×	×
ZERO					○				×	×		
NULL									×	×	○	
オブジェクト参照項目									×	×		
強く型付けされた集団項目									×	×		

(凡例)

○：指定できる

空白：受け渡しできるかどうかはプラットフォームに依存するため，動作保証についてはユーザ責任とする

×：指定してはならない。指定したときの結果は保証しない

注※1

COBOL の集団項目を受け渡しする場合，C プログラムでの境界調整を考慮して引数のデータ項目を定義する必要があります。

COBOL の集団項目は，標準では各基本項目がすき間なく配置されるのに対し，C プログラムでは計算機固有の境界に従って配置されます。このため，あらかじめ境界調整される分の未使用領域を定義しておくなど，計算機固有の境界調整を意識したデータを定義しておくことを推奨します。

なお，COBOL では，SYNCHRONIZED 句を指定することで計算機固有の境界調整に従って各基本項目を配置できます。

Linux(x64)では，プラットフォームの規約と COBOL 言語仕様の差異によって，従属項目として内部浮動小数点項目をもつ 16 バイト以下の集団項目を値渡し（BY VALUE）の引数および返却項目に指定した場合，C 言語プログラムとの間での引数および返却項目の受け渡しは保証しません。

このような場合，次に示すどれかの方法でプログラムを修正する必要があります。

・集団項目ではなく同じサイズの文字列項目として値の受け渡しをする。

・値渡しではなく参照渡し（BY REFERENCE）または内容渡し（BY CONTENT）に変更する。

・集団項目のサイズが 16 バイトを超えるようにする。

なお，-Lx64ConventionCheck オプションを指定すると，次の条件を満たす場合に警告レベルのエラーメッセージが出力されます。

1. サイズが 16 バイト以下の集団項目を値渡しの引数および返却項目に指定している。
2. 1.の集団項目の従属項目に内部浮動小数点項目がある。

Linux(x86)では，プラットフォームの規約と COBOL 言語仕様の差異によって，サイズが 1 バイトの COBOL 集団項目とサイズが 1 バイトの C 言語構造体との間で返却項目の受け渡しはできません。

AIX では，プラットフォームの規約と COBOL 言語仕様の差異によって，サイズが 1 バイトの COBOL 集団項目とサイズが 1 バイトの C 言語構造体との間で値渡し（BY VALUE）の引数および返却項目の受け渡しはできません。

注※2

標準文字集合で 1 文字の場合は char 型，2 文字以上の場合は次の構造体と対応づけます（n は文字数，name は任意のフィールド名とします）。

```
struct {
    char name[n];
};
```

注※3　SEPARATE 指定なしの 1 けたです。

注※4　-Bin1Byte オプションの指定がある場合，1～2 けた（割り当てられる記憶域のサイズが 1 バイト）の整数です。

注※5

- -Bin1Byte オプションの指定がない場合，1～4 けたの整数です。
- -Bin1Byte オプションの指定がある場合，3～4 けた（割り当てられる記憶域のサイズが 2 バイト）の整数です。

注※6

AIX(32)および Linux(x86)の場合，5～9 けたの整数です。

AIX(64)および Linux(x64)の場合，int または unsigned int のとき，5～9 けたの整数です。

注※7　割り当てられる記憶域のサイズが 1 バイトの場合，指定できます。

注※8　割り当てられる記憶域のサイズが 2 バイトの場合，指定できます。

注※9

AIX(32)および Linux(x86)の場合，割り当てられる記憶域のサイズが 4 バイトの場合，指定できます。

AIX(64)および Linux(x64)の場合，int または unsigned int のとき，割り当てられる記憶域のサイズが 4 バイトの場合，指定できます。

注※10　C プログラムから COBOL プログラムに引数を渡す場合，プロトタイプ宣言が必要です。

注※11　AIX(64)および Linux(x64)の場合，long および unsigned long のとき，10～18 けたの整数です。

注※12　AIX(64)および Linux(x64)の場合，データ型が long および unsigned long で，割り当てられる記憶域のサイズが 8 バイトのときに指定できます。

注※13　10～18 けたの整数です。

注※14　割り当てられる記憶域のサイズが 8 バイトの場合，指定できます。

注※15　符号修飾子を省略したとき，unsigned char と解釈する C コンパイラを使用する場合，明示的に signed char と書くか，または適切な C コンパイラのオプションを指定して signed char と解釈されるようにする必要があります。

(3)　引数の受け渡し

C プログラムから COBOL プログラムへ引数を渡す方法を，次に示します。

(a)　引数をポインタ型で受け取る例

C プログラム

```
int SAMPLE2(int *,char *);
    :
int SAMPLE1(....)
{
int  cnt;
char  str[80];
    :
  SAMPLE2(&cnt,str);
    :
  return(0);
}
```

COBOL プログラム

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
           :
       LINKAGE SECTION.
       01 CNT PIC S9(9) USAGE COMP.
       01 STR PIC X(80).

       PROCEDURE DIVISION USING BY REFERENCE CNT STR.
           :
```

- C プログラムから COBOL プログラムへ引数がアドレスによって渡された場合，COBOL プログラム内で受け取る引数に BY VALUE を指定してはいけません。
- COBOL プログラムは，C プログラムから数字項目を受け取る場合，内部・外部 10 進項目，内部・外部浮動小数点数字項目などの数字属性を意識する必要があります。

(b) 引数を値渡しで受け取る例

C プログラム

```
int  SAMPLE2(int,char);
int  SAMPLE1(....)
{
int  cnt;
char  str;
    :
  SAMPLE2(cnt,str);
    :
  return(0);
}
```

COBOL プログラム

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
           :
       LINKAGE SECTION.
       01 CNT PIC S9(9) USAGE COMP.
       01 STR PIC X(1).

       PROCEDURE DIVISION USING BY VALUE CNT STR.
           :
```

- C プログラムから COBOL プログラムへ引数が値渡しされた場合，PROCEDURE DIVISION USING で BY VALUE 指定する必要があります。
- C プログラム内で設定する引数は，すべて値渡しである必要があります。
- COBOL のデータ項目と C プログラムの型の対応については，「表 19-1　COBOL のデータ項目と C プログラムの型の対応」を参照してください。

(4) 戻り値の受け渡し

COBOL プログラムから C プログラムへ戻り値を返す方法を，次に示します。

(a) 整数型のデータ項目を返す例

C プログラム

```
extern int SAMPLE2();
    :
int main()
    {
    int rtc;
    :
    rtc = SAMPLE2();
    :
    }
```

COBOL プログラム

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
           :
       LINKAGE SECTION.
       01 RTC PIC S9(9) USAGE COMP.
       PROCEDURE DIVISION RETURNING RTC.
           :
           MOVE 12345 TO RTC.
           EXIT PROGRAM.
```

(b) 構造体型のデータ項目を返す例

C プログラム

```
struct tbl{ int a; char b[10];};
extern struct tbl SAMPLE2();
int main()
    {
  struct tbl rtc;
    :
    rtc = SAMPLE2();
    :
    }
```

COBOL プログラム

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
           :
       LINKAGE SECTION.
       01 TBL.
         02 A PIC S9(9) USAGE COMP.
         02 B PIC X(10).

       PROCEDURE DIVISION RETURNING TBL.
```

```
        :
        EXIT PROGRAM.
```

注意事項

C 言語の構造体の場合，要素の境界調整でパディング（領域間のギャップを埋めるために挿入される暗黙の FILLER 項目）が挿入されます。そのほかに，構造体のサイズを最大の基本要素サイズの倍数に切り上げるため，パディングが挿入されることがあります。この場合，パディングは構造体の末尾に挿入されます。例えば上記の例の構造体 tbl の場合，要素の合計のサイズは COBOL の集団項目 TBL と同じ 14 バイトですが，構造体のサイズを 4 バイト（最大の基本要素のサイズ）の倍数の 16 バイトにするために，2 バイトのパディングが構造体の末尾に挿入されます。このため，COBOL の集団項目 TBL とはサイズが不一致になります。プラットフォームによっては，このサイズの不一致が原因で，引数や戻り値の受け渡しで異常終了や結果不正となることがあるため，次のどちらかの方法で C 言語の構造体のサイズと COBOL の集団項目のサイズを一致させる必要があります。

- C 言語の構造体のサイズを COBOL の集団項目のサイズに合わせる方法

  構造体を詰めて配置する C 言語の機能（C 言語の言語仕様の#pragma pack や C 言語のコンパイラのコンパイラオプションで提供されている）を使って構造体 tbl のサイズを 14 バイトにします。構造体を詰めて配置する C 言語の機能については，各システムの C コンパイラのリファレンスを参照してください。

- COBOL の集団項目のサイズを C 言語の構造体のサイズに合わせる方法

  COBOL の集団項目 TBL のサイズを 16 バイトにするために，COBOL の集団項目 TBL の末尾に次の 2 バイト分の FILLER 項目を追加してください。

```
    02 FILLER PIC X(2).
```

19.1.3　COBOL プログラムから C プログラムを呼び出す方法

COBOL プログラムから C プログラムを呼ぶときの規則を示します。

(1)　COBOL から C を呼ぶときの規則

- COBOL プログラムを-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf または-DebugRange オプション指定でコンパイルし，C プログラムで例外処理をした場合，異常終了時の結果は保証しません。
- C プログラム内で宣言された外部変数，および COBOL プログラム内の外部属性を持つデータ項目はそれぞれのプログラムで参照できます。外部属性を持つデータ項目の参照方法については，「19.1.5 外部属性を持つデータ項目の共用」を参照してください。
- COBOL プログラムは，C プログラムの return 文で設定された関数値を RETURN-CODE 特殊レジスタとして参照できます。C プログラムが返す関数値の有効な範囲については，「17.2　復帰コードと返却項目」を参照してください。
- COBOL プログラムから C プログラムを呼び出して，内部浮動小数点項目の引数渡しをする場合，受け取りのデータ型は次の規則に従う必要があります。

1. 単精度内部浮動小数点項目の引数渡しをする場合，受け取りのデータ型は float とする。
2. 倍精度内部浮動小数点項目，または浮動小数点数字定数の引数渡しをする場合，受け取りのデータ型は double とする。

- CALL 文に RETURNING 指定がある場合，C プログラムの戻り値は，そのデータ項目に対応した C 言語のデータ型にする必要があります。RETURNING 指定がない場合は int 型にする必要があります。
- COBOL プログラムでは，C プログラムの return 文で設定した戻り値を，呼び出された CALL 文に指定した RETURNING のデータ項目として参照できます。RETURNING 指定がない場合は RETURN-CODE 特殊レジスタで参照します。
- CALL 一意名で C プログラムを呼ぶ場合，次の点に注意する必要があります。
  - 共用ライブラリ中の C プログラムを呼ぶ場合，共用ライブラリのエクスポート情報中にプログラム名がなければなりません。このため，リンク時にオプションの指定によってプログラムをエクスポートしなかった場合，そのプログラムは CALL 一意名では呼び出せません。

(2) C プログラムの型と COBOL での宣言

C プログラムのデータ型と，対応する COBOL のデータ項目の型を，次に示します。

C プログラムのデータ型	対応する COBOL のデータ項目の型		
unsigned char	-Bin1Byte 指定あり	符号なし 2 進項目（1～2 けた）	`9(1) COMP ～ 9(2) COMP`
	-Bin1Byte 指定なし		`9(1) COMP-X ～ 9(2) COMP-X`
	−	英数字項目（1 けた）	`X(1)`
char	-Bin1Byte 指定あり	符号付き 2 進項目（1～2 けた）	`S9(1) COMP ～ S9(2) COMP`※4
	−	英数字項目（1 けた）	`X(1)`
unsigned short	−	符号なし 2 進項目（3～4 けた）	`9(3) COMP ～ 9(4) COMP` `9(3) COMP-X ～ 9(4) COMP-X`
short	−	符号付き 2 進項目（3～4 けた）	`S9(3) COMP ～ S9(4) COMP`
unsigned int	−	符号なし 2 進項目（5～9 けた）	`9(5) COMP ～ 9(9) COMP` `9(5) COMP-X ～ 9(9) COMP-X`
int	−	符号付き 2 進項目（5～9 けた）	`S9(5) COMP ～ S9(9) COMP`
unsigned long	−	符号なし 2 進項目（5～9 けた）※1	`9(5) COMP ～ 9(9) COMP` `9(5) COMP-X ～ 9(9) COMP-X`
		符号なし 2 進項目（10～18 けた）※2	`9(10) COMP ～ 9(18) COMP` `9(10) COMP-X ～ 9(18) COMP-X`

C プログラムのデータ型	対応する COBOL のデータ項目の型		
long	−	符号付き 2 進項目（5～9 けた）※1	S9(5) COMP ～ S9(9) COMP
		符号付き 2 進項目（10～18 けた）※2	S9(10) COMP ～ S9(18) COMP
unsigned long long	−	符号なし 2 進項目（10～18 けた）	9(10) COMP ～ 9(18) COMP 9(10) COMP-X ～ 9(18) COMP-X
long long	−	符号付き 2 進項目（10～18 けた）	S9(10) COMP ～ S9(18) COMP
float	−	単精度内部浮動小数点数字項目	COMP-1
double	−	倍精度内部浮動小数点数字項目	COMP-2
ポインタ型	−	アドレス名，アドレスデータ項目	ADDRESS
配列型	−	アドレスデータ項目	ADDRESS
構造体※3	−	集団項目	−

（凡例）

−：該当しない

注※1　AIX(32)および Linux(x86)の場合です。

注※2　AIX(64)および Linux(x64)の場合です。

注※3

COBOL の集団項目を受け渡しする場合，C プログラムでの境界調整を考慮して引数のデータ項目を定義する必要があります。

COBOL の集団項目は，標準では各基本項目がすき間なく配置されるのに対し，C プログラムでは計算機固有の境界に従って配置されます。このため，あらかじめ境界調整される分の未使用領域を定義しておくなど，計算機固有の境界調整を意識したデータを定義しておくことを推奨します。

なお，COBOL では，SYNCHRONIZED 句を指定することで計算機固有の境界調整に従って各基本項目を配置できます。

Linux(x64)では，プラットフォームの規約と COBOL 言語仕様の差異によって，従属項目として内部浮動小数点項目をもつ 16 バイト以下の集団項目を値渡し（BY VALUE）の引数および返却項目に指定した場合，C 言語プログラムとの間での引数および返却項目の受け渡しは保証しません。

このような場合，次に示すどれかの方法でプログラムを修正する必要があります。

・集団項目ではなく同じサイズの文字列項目として値の受け渡しをする。

・値渡しではなく参照渡し（BY REFERENCE）または内容渡し（BY CONTENT）に変更する。

・集団項目のサイズが 16 バイトを超えるようにする。

なお，-Lx64ConventionCheck オプションを指定すると，次の条件を満たす場合に警告レベルのエラーメッセージが出力されます。

1. サイズが 16 バイト以下の集団項目を値渡しの引数および返却項目に指定している。
2. 1.の集団項目の従属項目に内部浮動小数点項目がある。

Linux(x86)では，プラットフォームの規約と COBOL 言語仕様の差異によって，サイズが 1 バイトの COBOL 集団項目とサイズが 1 バイトの C 言語構造体との間で返却項目の受け渡しはできません。

AIX では，プラットフォームの規約と COBOL 言語仕様の差異によって，サイズが 1 バイトの COBOL 集団項目とサイズが 1 バイトの C 言語構造体との間で値渡し（BY VALUE）の引数および返却項目の受け渡しはできません。

注※4　符号修飾子を省略したとき，unsigned char と解釈する C コンパイラを使用する場合，明示的に signed char と書くか，または適切な C コンパイラのオプションを指定して signed char と解釈されるようにする必要があります。

(3)　引数の受け渡し

COBOL プログラムから C プログラムへ引数を渡す方法を，次に示します。

(a)　引数をポインタ型で引き渡す例

COBOL プログラム

```
        :
      WORKING-STORAGE SECTION.
      01 CNT PIC S9(9) USAGE COMP.
      01 STR.
        02 STRCHAR PIC X(79).
        02 FILLER  PIC X VALUE LOW-VALUE.
        :
      PROCEDURE DIVISION.
        :
        CALL 'sample2' USING BY REFERENCE CNT STR.
        :
```

C プログラム

```
int  sample2(int *cnt, char *str)
{
    :
  return(0);
}
```

- COBOL プログラムから C プログラムに BY VALUE 指定なしで引き渡す引数は，すべてポインタ型として引き渡されます。このため，C プログラム内で受け取る引数は，すべてポインタ型で宣言する必要があります。
- C プログラムは，COBOL プログラムから数字項目を受け取る場合，内部・外部 10 進項目，内部・外部浮動小数点数字項目などの数字属性を意識する必要があります。

(b)　引数を値渡しで引き渡す例

COBOL プログラム

```
        :
      WORKING-STORAGE SECTION.
      01 CNT PIC S9(9) USAGE COMP.
      01 STR PIC X(1).
        :
      PROCEDURE DIVISION.
        :
```

```
          CALL 'sample2' USING BY VALUE CNT STR.
            :
```

C プログラム

```
int  sample2(int cnt, char str)
{
    :
  return(0);
}
```

- C プログラム内で受け取る引数は，BY VALUE 指定の場合は値渡しで受け取ります。CALL 文の BY VALUE で指定した COBOL のデータ項目に対する C 言語のデータ型の対応については，「表 19-1 COBOL のデータ項目と C プログラムの型の対応」を参照してください。
- BY VALUE 指定時，引数は次のように設定されます。
  - 単精度内部浮動小数点数字項目は 4 バイトで，倍精度内部浮動小数点数字項目は 8 バイトで設定されます。
  - 数字定数，および ZERO は 4 バイトの 2 進形式として設定されます。
  - 浮動小数点数字定数は倍精度浮動小数点形式として設定されます。

(c) CALL 文の引数に ADDRESS OF 一意名，LENGTH OF 一意名を指定した例

COBOL 主プログラム

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 WK-1.
         02 W01    PIC X(30)
            VALUE 'ABCDEFGHIJKLMNOPQRSTUVWXYZ1234'.
         02 FILLER PIC X VALUE LOW-VALUE. …5.
       PROCEDURE  DIVISION.
           CALL 'SAMPLE2'
                USING BY REFERENCE WK-1. …1.
           CALL 'SAMPLE3'
                USING BY REFERENCE ADDRESS OF WK-1. …2.
           CALL 'SAMPLE4'
                USING BY CONTENT LENGTH  OF WK-1. …3.
           CALL 'SAMPLE5'
                USING BY VALUE LENGTH  OF WK-1. …4.
           STOP RUN.
       END PROGRAM SAMPLE1.
```

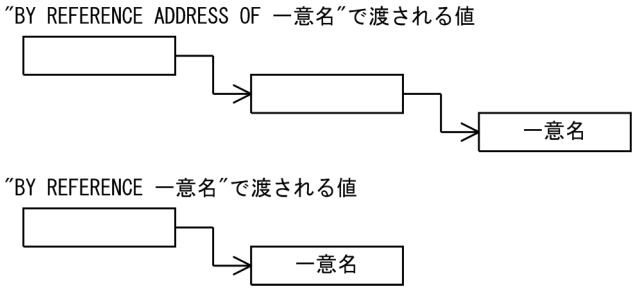
C 副プログラム

```
#include <stdio.h>
 int SAMPLE2(char *p_ptr_r)
 {
   printf("first char(r) =(%s)¥n",p_ptr_r);   … 1.
    :
 }
```

```
int SAMPLE3(char **p_ptr_ra)
{
  printf("first char(ra)=(%s)¥n",*p_ptr_ra);   … 2.
   :
}
int SAMPLE4(
    int *p_ptr_cl  /* この引数の型は32bitアプリケーションの場合  */
                   /* 64bitアプリケーションの場合は long long *  */
    )
{
  printf("length(p)     =(%d)¥n",*p_ptr_cl);   … 3.
                  /* この書式指定は32bitアプリケーションの場合  */
                  /* 64bitアプリケーションの場合  %lld          */
   :
}
int SAMPLE5(
    int p_ptr_vl   /* この引数の型は32bitアプリケーションの場合  */
                   /* 64bitアプリケーションの場合は long long    */
    )
{
  printf("length(p)     =(%d)¥n",p_ptr_vl);   … 4.
                  /* この書式指定は32bitアプリケーションの場合  */
                  /* 64bitアプリケーションの場合  %lld          */
   :
}
```

1. BY REFERENCE 指定の CALL 文で渡される値は，指定された一意名のアドレスです。
2. ADDRESS OF で修飾した場合に渡される値は，一意名のアドレスではなく，一意名のアドレスが入っている領域のアドレスとなります。
3. LENGTH OF で修飾した CONTENT 指定では，一意名の長さが入っている領域のアドレスが渡されます。
4. LENGTH OF で修飾した VALUE 指定では，一意名の長さが値として渡されます。
5. C プログラム中で文字列として参照するため，COBOL プログラム中で英数字項目の終端に NULL 文字（X'00'）を設定しています。

BY REFERENCE 指定で渡される引数の値



実行結果

```
first char(r) =(ABCDEFGHIJKLMNOPQRSTUVWXYZ1234)
    :
first char(ra)=(ABCDEFGHIJKLMNOPQRSTUVWXYZ1234)
    :
```

```
length(p)     =(31)
    :
length(p)     =(31)
    :
```

(4) 戻り値の受け渡し

C プログラムから COBOL プログラムへ戻り値を返す方法を，次に示します。

(a) 整数型のデータ項目をリターンする例

COBOL プログラム

```
       WORKING-STORAGE SECTION.
       01 RTC PIC S9(9) USAGE COMP.
           :
       PROCEDURE DIVISION.
           :
           CALL 'sample2' RETURNING RTC.
```

C プログラム

```
int sample2()
  {
  int rtc;
    :
  return (rtc);
  }
```

(b) 構造体型のデータ項目をリターンする例

構造体型のデータ項目をリターンする場合，COBOL プログラムと C プログラムでは境界調整によってデータのマッピング方法が異なることがあるので注意が必要です。

COBOL プログラム

```
       WORKING-STORAGE SECTION.
       01 TBL.
         02 A PIC S9(9) USAGE COMP.
         02 B PIC X(10).
           :
       PROCEDURE DIVISION.
           :
           CALL 'sample2' RETURNING TBL.
```

C プログラム

```
struct tbl {int a; char b[10];};
struct tbl sample2()
{
    struct tbl rtn;
    :
    return (rtn);
}
```

19.1.4　注意事項

C 言語の構造体の場合，要素の境界調整でパディング（領域間のギャップを埋めるために挿入される暗黙の FILLER 項目）が挿入されます。そのほかに，構造体のサイズを最大の基本要素サイズの倍数に切り上げるため，パディングが挿入されることがあります。この場合，パディングは構造体の末尾に挿入されます。例えば上記の例の構造体 tbl の場合，要素の合計のサイズは COBOL の集団項目 TBL と同じ 14 バイトですが，構造体のサイズを 4 バイト（最大の基本要素のサイズ）の倍数の 16 バイトにするために，2 バイトのパディングが構造体の末尾に挿入されます。このため，COBOL の集団項目 TBL とはサイズが不一致になります。プラットフォームによっては，このサイズの不一致が原因で，引数や戻り値の受け渡しで異常終了や結果不正となることがあるため，次のどちらかの方法で C 言語の構造体のサイズと COBOL の集団項目のサイズを一致させる必要があります。

- C 言語の構造体のサイズを COBOL の集団項目のサイズに合わせる方法

  構造体を詰めて配置する C 言語の機能（C 言語の言語仕様の#pragma pack や C 言語のコンパイラのコンパイラオプションで提供されている）を使って構造体 tbl のサイズを 14 バイトにします。構造体を詰めて配置する C 言語の機能については，各システムの C コンパイラのリファレンスを参照してください。

- COBOL の集団項目のサイズを C 言語の構造体のサイズに合わせる方法

  COBOL の集団項目 TBL のサイズを 16 バイトにするために，COBOL の集団項目 TBL の末尾に次の 2 バイト分の FILLER 項目を追加してください。

```
    02 FILLER PIC X(2).
```

C 言語のシステム関数は，#define によって実際の関数名とは別の関数名で宣言され，マクロ展開後に置き換えられることがあります。COBOL では C 言語のマクロは展開しないため，COBOL プログラムから C 言語のシステム関数を CALL 文で直接呼び出すと，COBOL からマクロ展開前の関数名を呼び出し，予期せぬ不具合が発生するおそれがあります。

COBOL プログラムから C 言語のシステム関数を呼び出す場合は，直接呼び出さないで，システム関数を呼び出す C プログラムを作成し，この C プログラムを呼び出すようにしてください。

19.1.5　外部属性を持つデータ項目の共用

COBOL プログラムで定義した外部属性を持つデータ項目を C プログラムと共有する方法を，次に示します。

なお，外部属性を持つデータ項目の共有属性については，「4.2.2　外部属性（EXTERNAL 句）」を参照してください。

COBOL プログラム

```
                :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
```

```
       01 EXTREC IS EXTERNAL.          ····· EXTERNAL領域定義
         02 EXT-REC1 PIC S9(9) USAGE COMP.
         02 EXT-REC2 PIC S9(4) USAGE COMP.
         02 EXT-REC3 PIC X(14).
                    :
```

C プログラム

```
struct extarea{
    int   ext_rec1;
    short ext_rec2;
    char  ext_rec3[14];
};
int sample2()
{
  extern struct extarea EXTREC;  ····· 外部参照宣言
  if (EXTREC.ext_rec1 == 1){
      EXTREC.ext_rec2 = 2;
  }
  return(0);
}
```

COBOL プログラムで外部属性を持つデータ項目を指定した場合，C プログラムからこれを参照，更新できます。このときの注意事項を次に示します。

- C プログラムで参照する外部変数名には，COBOL プログラムで定義したデータ項目の名称を使用してください。ただし，-EquivRule,NotAny オプションが指定されていない場合，COBOL プログラム中のデータ項目名に英小文字を用いると英大文字に変換されるので，外部変数名に英小文字は使用できません。

  また，外部変数名には，2 バイトコードや，ハイフン（-），下線（_），#なども使用できません。
- COBOL プログラム内で定義するデータ項目の名称は，最外側のプログラム名と異なる名称にしてください。
- C プログラムとの間では，EXTERNAL 句による属性チェックはされません。このため，C プログラムで外部属性を持つデータ項目の参照，および更新する場合，各データ項目の属性，位置，サイズを同じにする必要があります。
- EXTERNAL 句に INDEXED BY 指定をした場合，C プログラムからの参照や更新はできません。
- EXTERNAL 句に DYNAMIC を指定した場合，C プログラムからの参照や更新はできません。
- 外部属性を持つデータ項目が日本語の場合，C プログラムからの参照や更新はできません。

19.1.6　COBOL プログラムと C プログラムのリンク方法

COBOL プログラムと C プログラムをリンクする方法については，「34.　実行可能ファイルと共用ライブラリの作成」を参照してください。

20 オブジェクト指向機能

オブジェクト指向機能は，メッセージのやり取りによる処理，カプセル化，継承など，オブジェクト指向技術を導入した COBOL 言語仕様です。

この章では，オブジェクト指向の考え方やオブジェクト指向機能の特徴について説明します。また，オブジェクト指向機能の基本概念や用語についても説明します。

20.1　オブジェクト指向の紹介

ここでは，オブジェクト指向について説明します。

20.1.1　ソフトウェア開発の現状

ソフトウェア開発の技術は，今日まで目覚ましい発展を遂げてきました。次々に新しい技術が生まれ，ソフトウェアの規模は拡張されてきました。しかし，ソフトウェアの拡張に伴い，さまざまな問題が出てきました。

(1)　生産面での問題点

従来のプログラムは，ある特定の業務を処理するために開発される，機能中心のものでした。このようにして設計されるプログラムは，適用される業務が限定され，それ以外の業務には対応できない柔軟性に欠けたものでした。そのため，プログラマは業務ごとに新しいプログラムを設計しなければならず，作業に多くの時間を必要としていました。

(2)　信頼面での問題点

従来のプログラミング言語によるプログラムでは，処理形式などの標準化が徹底されていないため，既存のプログラムを再利用するのが難しく，新しいプログラムのほとんどは最初から設計されていました。また，従来のプログラム設計では最後まで設計が終了し，実際に実行させないと期待どおりの動きをするかどうかがわかりませんでした。そのため，設計に必要な時間の割に信頼性の低いプログラムしか作成できませんでした。

(3)　保守面での問題点

従来のプログラミング言語によるプログラムを変更または修正するのには，大変な手間と時間が掛かりました。変更，修正がうまくできずに性能が劣化したり，使えなくなる場合もありました。これは，データ間またはプログラム間の依存度が高いため，一つの変更が，予想外の結果まで引き起こすことが頻発したからです。

これらの問題点に対応するための手段として，オブジェクト指向によるシステム開発技法があります。オブジェクト指向とは，どのようなものかを次に示します。

20.1.2　オブジェクト指向

オブジェクトとは，英語で「もの」という意味です。この言葉からもわかるように，オブジェクト指向とは，現実にある「もの」や，「もの」同士が作用し合うことによって作り出される「現象」を，そのままコンピュータの世界に反映しようという考えです。

(1) データ中心の考え方（生産面での問題点の解決）

従来の COBOL プログラミングでは，実現したいシステムの処理（機能）を分析し，機能ごとのモジュールを段階的に，細かく分割する手法がとられます。こうした分割手法では，あるモジュールはその上位のモジュールの機能を実現するために必要な機能の一つとして存在します。このため，あるモジュールをまったく別のモジュール機能として，部分的に再利用するのは困難です。また，プログラムを変更したり拡張したりする際には，機能を実現するために使用されるデータが，どこで，何の目的で参照・設定されているのかを把握しなければなりませんでした。

オブジェクト指向では，実現したいシステムで使用されるデータを中心に考えます。システムの中で何らかの役割を担うデータをオブジェクトとして抽出し，データとそのデータに対して与えられる機能を一まとまりにしてモジュールを構成します。つまり，データを基にモジュールを分割する手法です。

このようにモジュールを分割すれば，同じデータを使用するシステムには，モジュールを再利用できます。また，データを直接参照・設定する部分がそのモジュールに限られるため，プログラムの修正や拡張も容易になります。

(2) オブジェクトの構造

オブジェクトとは，処理対象のデータとそのデータを操作する手続き（属性とサービス）をカプセル化したものです。次に例を挙げて説明します。

(例) A さんの銀行口座のオブジェクト

A さんは，ある銀行に普通預金の口座を開設しています。ここでは，普通預金の口座を例にして，オブジェクトを説明します。

A さんの普通預金口座での処理をオブジェクト指向で実現させるために，必要な情報を抽出します。まず，この口座が A さんのものであるという情報が必要です。これらは，次に示す情報で表されます。

- 名前
- 口座番号

次に，A さんの口座の状態を知るための情報を示します。

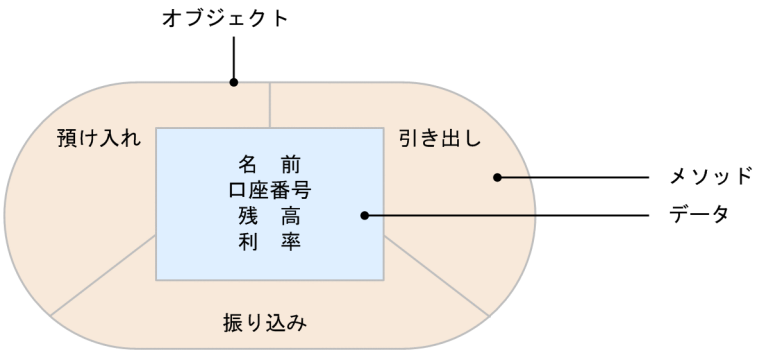
- 残高
- 利率

これらが，A さんの口座であるということと，その口座の状態を認識するのに必要な情報です。これらの情報に対し，銀行の普通預金口座として機能を果たすために必要な動作として，次の三つを挙げてみます。

- 預け入れ
- 引き出し
- 振り込み

A さんの口座は，「名前」や「口座番号」で認識され，「残高」に対し「預け入れ」や「引き出し」の処理を行うことができます。オブジェクト指向では，これらの情報をデータ，処理をメソッド（英語で手段を表す言葉）と表現します。

このような，A さんの普通預金口座のオブジェクトは，次のような概念図で表せます。以後，このマニュアルではオブジェクトの概念図をこのように表します。



(3)　カプセル化（保守面での問題点の解決）

オブジェクトを表す概念図は，オブジェクトのデータおよびメソッドの実装が隠ぺいされている状態を示しています。オブジェクト内のデータは，原則としてそのオブジェクトのメソッドによってだけ操作されます。また，オブジェクト内で変化したデータの値が，そのオブジェクトの外部のデータ値に，暗黙的に影響を与えることもありません。

このように，データもメソッドの実装もオブジェクト内に閉じ込め，外部から隠ぺいすることを，カプセル化といいます。

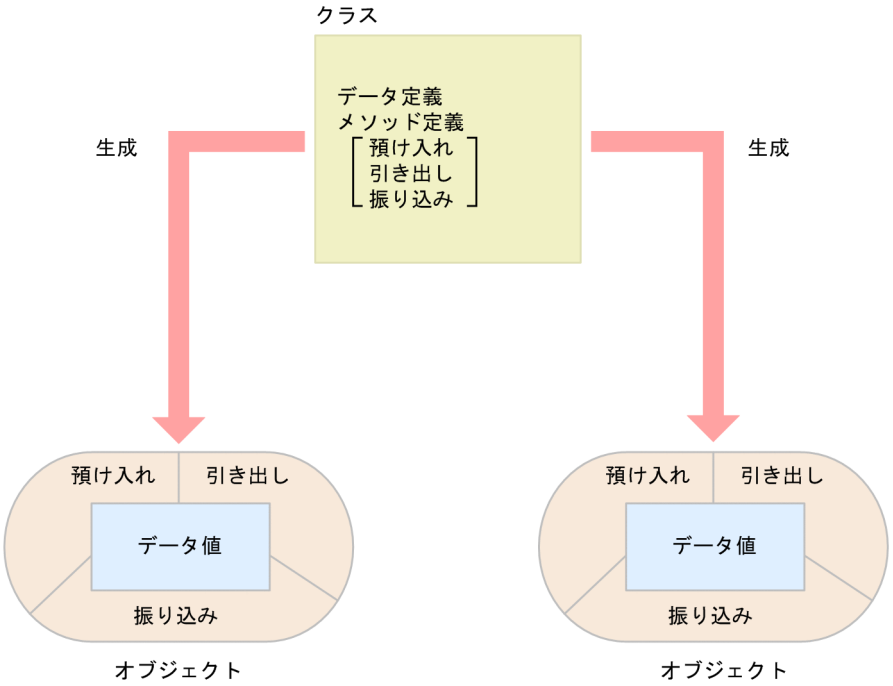
このカプセル化は，従来のプログラミング言語によるシステム開発で課題となっていた，データ間またはデータとプログラムの依存度の高さによって発生する変更時の障害を解消します。

(4)　クラス

クラスとは，オブジェクトを生成するときの型のことです。オブジェクト指向では，クラスを使用してオブジェクトのデータ属性やメソッドを定義し，オブジェクトをカプセル化します。

クラスとオブジェクトの関係を銀行口座を例にして説明します。

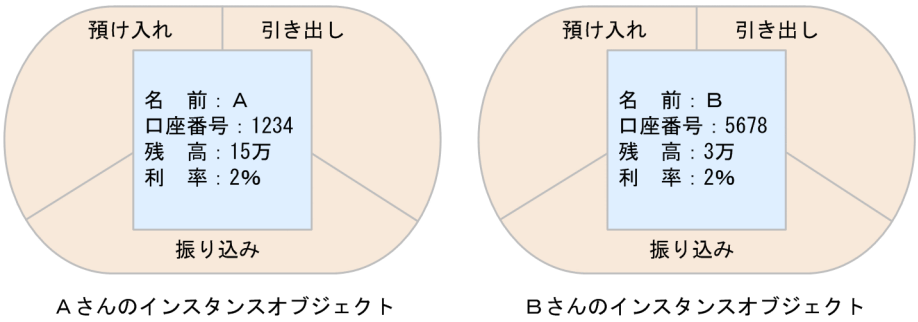
銀行口座に必要なデータ（名前，口座番号など）と，それらのデータを操作するメソッド（預け入れ，引き出しなど）は，銀行口座クラスとして定義します。こうして定義されたクラスは，オブジェクトを生成するための型であり，だれの口座でもありません。実際の口座は，銀行口座クラスを基にオブジェクトとして生成します。各人の口座のデータの定義は同じですが，名前，口座番号などのデータには，それぞれ異なる値が設定されます。



(a)　インスタンスオブジェクト

クラスから生成されるオブジェクトの実体のことを，インスタンスオブジェクトといいます。

一つのクラス定義から，複数のインスタンスオブジェクトを生成できます。生成されたインスタンスオブジェクトは，定義されたデータを個別に持っています。そのため，同じクラスから生成されても，それぞれのインスタンスオブジェクトは，固有のデータ値を持っています。



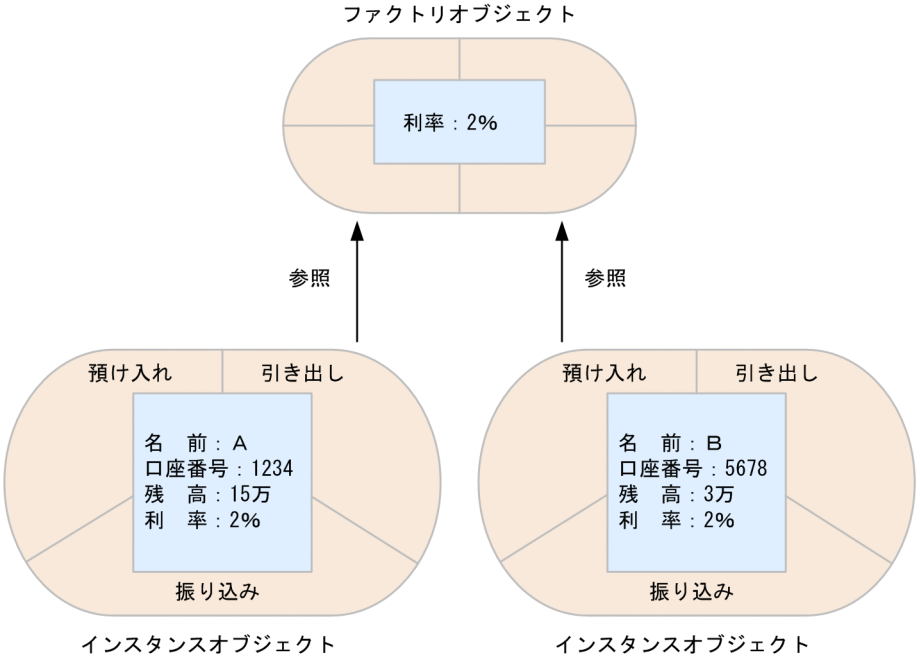
(b)　ファクトリオブジェクト

同じクラスから生成されたインスタンスオブジェクトは，それぞれ固有のデータ値を持っています。しかし，中には口座オブジェクトの「利率」のように，各オブジェクトに共通するデータ値もあります。このようなデータをすべてのオブジェクトが持っていてもかまいませんが，まとめて 1 か所に保持しておく方が，保守しやすいオブジェクトになります。

ファクトリオブジェクトは，このように全オブジェクトに共通するデータを，1 か所で管理するためのオブジェクトです。

例えば，「利率」を変更する場合，各インスタンスオブジェクトがそれぞれ「利率」を持っていると，それらをすべて変更しなければなりません。しかし，「利率」がファクトリオブジェクトに保持されていれば，ファクトリオブジェクトの「利率」だけを変更すればよいのです。

ファクトリオブジェクトは，一つのクラスに対して一つだけ存在します。



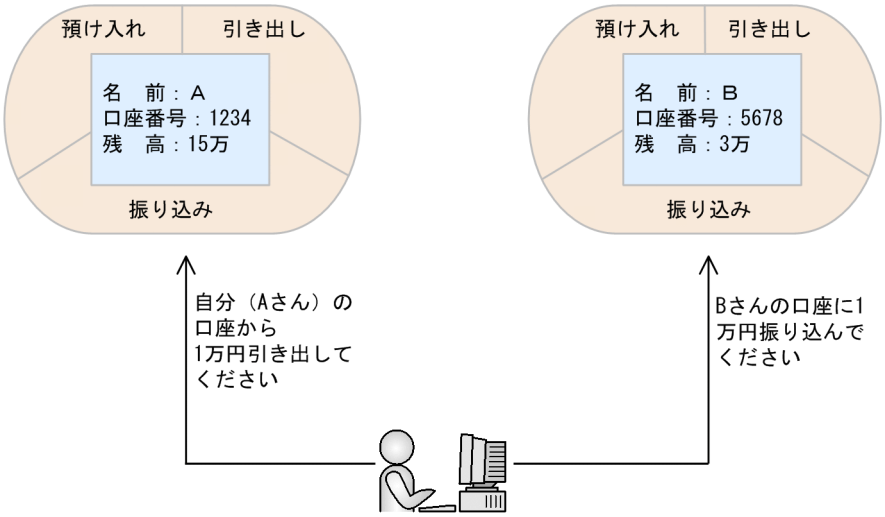
(5)　メソッドの呼び起こし（メッセージパッシング）

オブジェクト指向では，データと処理は，それぞれにカプセル化されます。カプセル化されたデータは，他オブジェクトからの影響を受けてはなりません。このカプセル化という概念を守りながら，処理プログラムを実行させるために，オブジェクト指向ではメッセージを使用します。

メッセージとは，クラスからオブジェクトを生成させたり，またはあるオブジェクトが自分では処理できない問題の解決を，その処理ができるほかのオブジェクトに依頼するときに使用されるものです。次に例を示します。

（例）A さんが B さんの口座に現金を振り込む場合

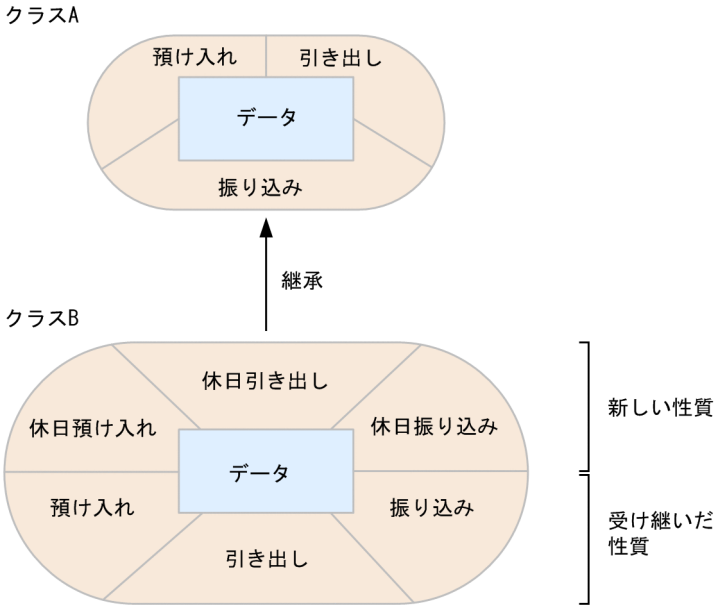
A さんが B さんの口座に 1 万円を振り込もうとしています。しかし，A さんの口座オブジェクトのメソッドは，B さんの口座オブジェクトのデータを操作できません。そこで，B さんの口座オブジェクトに次のようなメッセージを送ります。



このようにして，メッセージを介してオブジェクトのメソッドを操作することをメソッドの呼び起こし（メッセージパッシング）といいます。

(6)　継承（信頼面での問題点の解決）

継承とは，あるクラスが持つ性質をほかのクラスが受け継ぐことです。例えば，銀行が従来のサービスに新たなサービスを加えて，新しいサービスを提供する制度を設けようとした場合，継承を利用します。次に例を示します。



注

継承の矢印は，あるクラスがどのクラスを継承しているのかを示します。例えば，クラスＢがクラスＡを継承しているとき，矢印の向きもクラスＢがクラスＡを指します。

従来のサービスを持つクラスを継承したクラスでは，新しく追加するサービスだけを定義します。継承によって，従来のサービスが受け継がれ，使用できるようになっているので，これらのサービスについては定義する必要はありません。また，受け継いだ性質の再定義もできます。

このように，継承によって既存のクラスを利用して新しいクラスを作成すると，原始プログラム中のコーディングの重複が避けられるので保守資産の増加を抑えられ，システム開発にかかる労力を短縮できます。また，既存のクラスを利用することで，最初から品質が保証されている，信頼性の高いクラスを作成できます。

なお，このマニュアルでは継承されるクラスを「スーパクラス」，継承するクラスを「サブクラス」といいます。

(7) インタフェース

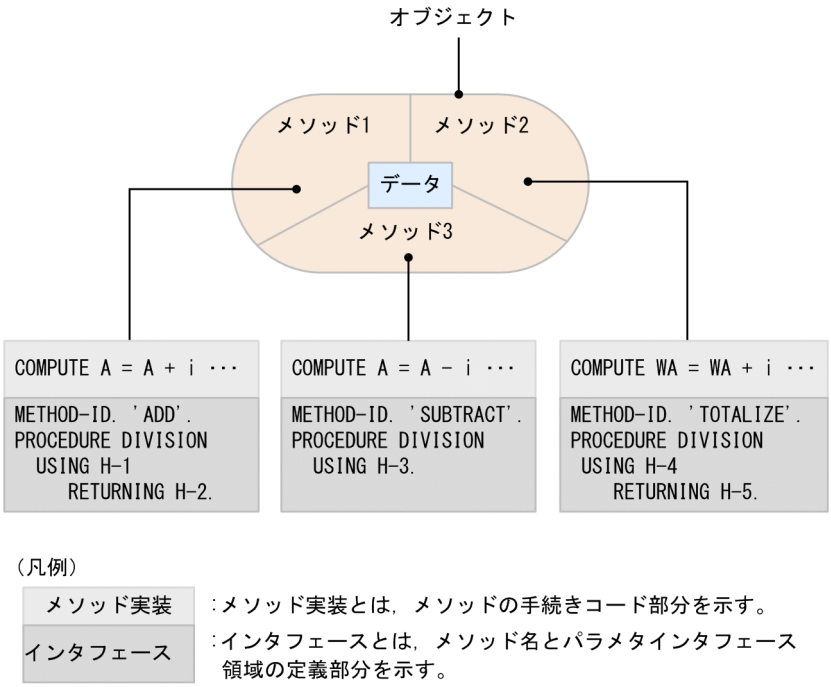
インタフェースは，このあとに説明する適合やポリモルフィズムに深くかかわっている重要な概念です。インタフェースは，オブジェクトがどのようなサービスを提供するのかを表します。具体的には，オブジェクトが持つメソッドの名前とパラメタの並び（これらをまとめて，メソッド原型といいます）の集まりから成ります。インタフェースは，オブジェクトが持つすべてのメソッド原型を表現することも，一部を表現することもできます。

(a) メソッド名

メソッドには，それぞれ名前が付いています。メソッド名を，メソッドが提供するサービス，つまり実装を簡潔に表すものにすると，プログラムが理解しやすくなります。メソッド名は，一つのクラス中で一意でなければなりません。ただし，クラスが違えば，異なるサービスを提供するメソッドに対し，同じメソッド名を付けてかまいません。

(b) パラメタ

パラメタには，処理するデータと，その処理の結果を返すデータを指定します。



（凡例）

メソッド実装	：メソッド実装とは，メソッドの手続きコード部分を示す。
インタフェース	：インタフェースとは，メソッド名とパラメタインタフェース領域の定義部分を示す。

注

メソッド実装とは，メソッド内の実処理を示します。

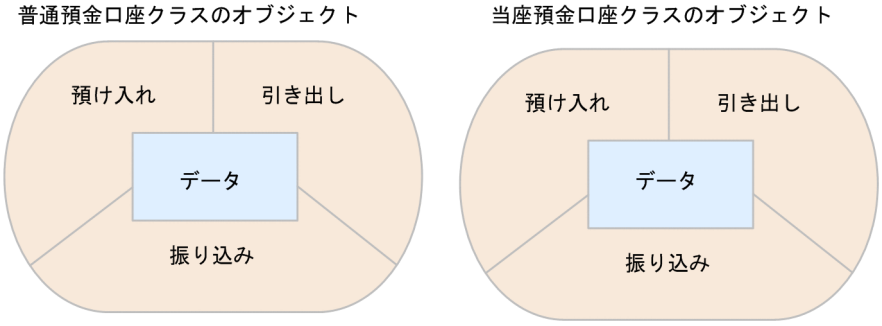
(8) ポリモルフィズム

ポリモルフィズムとは，同じメッセージを送っても，メッセージを受け取るオブジェクトの生成元のクラスが異なれば，そのオブジェクトごとに異なる処理が行われる仕組みです。この仕組みを使用すると，複数の処理を一つのメッセージで実現できます。

次に例を示します。

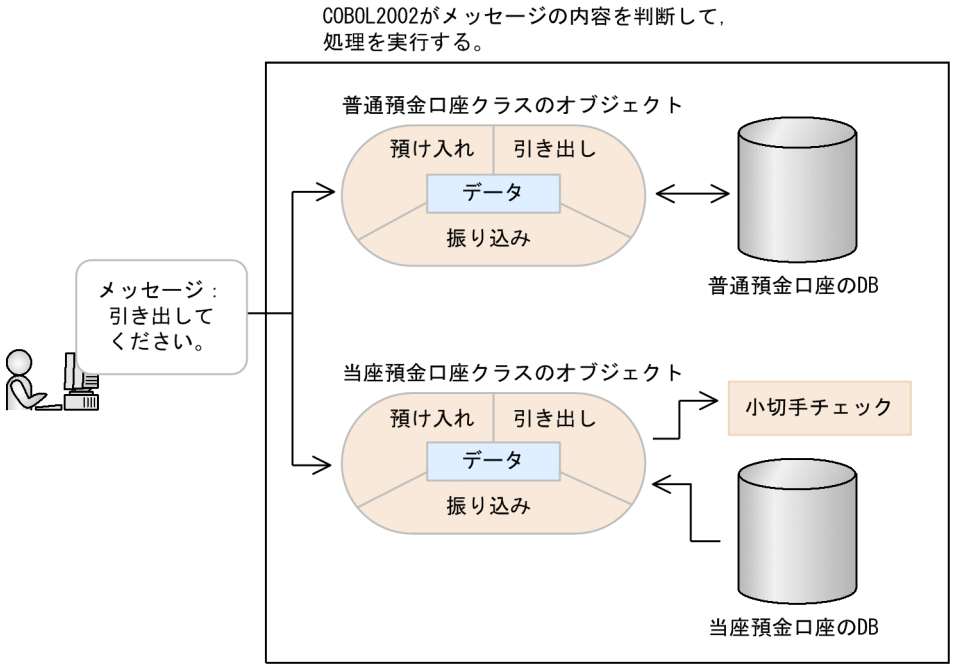
（例）普通預金口座および当座預金口座からの引き出し

普通預金口座および当座預金口座は，それぞれ別のクラスから生成されたオブジェクトです。クラスが異なるので，それぞれのオブジェクトのメソッドに同じ名前を付けることができます。



これらのオブジェクトは，「引き出してください。」というメッセージを受け取ることができます。普通預金口座クラスのオブジェクトのメソッドは普通預金から現金を引き出します。当座預金口座クラスのオブジェクトのメソッドは，当座預金からの引き出しを要求する小切手をチェックして，現金を引き出します。

つまり，同じメッセージであってもそれを受け取るオブジェクトのクラスが異なっていれば，異なる動作をするわけです。



ポリモルフィズムの仕組みは，メッセージのやり取りの単純化に役立ちます。また，新規に受け手のクラスを定義しても，メッセージの送り手側を修正する必要がないため，再利用性や保守性を向上させることができます。

注

COBOL2002 では，メソッドが適合していなければメソッドに同じ名前を付けること（オーバライド）は，できません。

メソッドが適合しているかどうかは，マニュアル「COBOL2002　言語　標準仕様編」「5.2.7(1)　オブジェクト指向での「適合」」の規則に従ってチェックされます。

20.2　COBOL2002 でのオブジェクト指向機能

「20.1　オブジェクト指向の紹介」で紹介したオブジェクト指向を実現する言語仕様として，COBOL2002 ではオブジェクト指向機能があります。ここでは，オブジェクト指向機能について簡単に説明します。

20.2.1　オブジェクト指向機能による定義

(1)　クラスの定義

オブジェクト指向を取り入れたプログラム開発を，COBOL2002 で実現するためには，プログラム定義とクラス定義が必要です。

＜プログラム定義＞

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  MAIN-A.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
 REPOSITORY.
  CLASS 普通預金口座クラス.
       :
DATA DIVISION.
WORKING-STORAGE SECTION.

01 A USAGE OBJECT REFERENCE.
       :
PROCEDURE DIVISION.
       :
    INVOKE 普通預金口座クラス 'NEW'
           RETURNING A.
       :
```

＜クラス定義＞

```
IDENTIFICATION DIVISION.
CLASS-ID.  普通預金口座クラス
    INHERITS BASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
  CLASS BASE.
IDENTIFICATION DIVISION.
OBJECT.
       :
IDENTIFICATION DIVISION.
METHOD-ID.  '預け入れ'.
       :
END METHOD '預け入れ'.
END OBJECT.
END CLASS 普通預金口座クラス.
```

(a)　プログラム定義

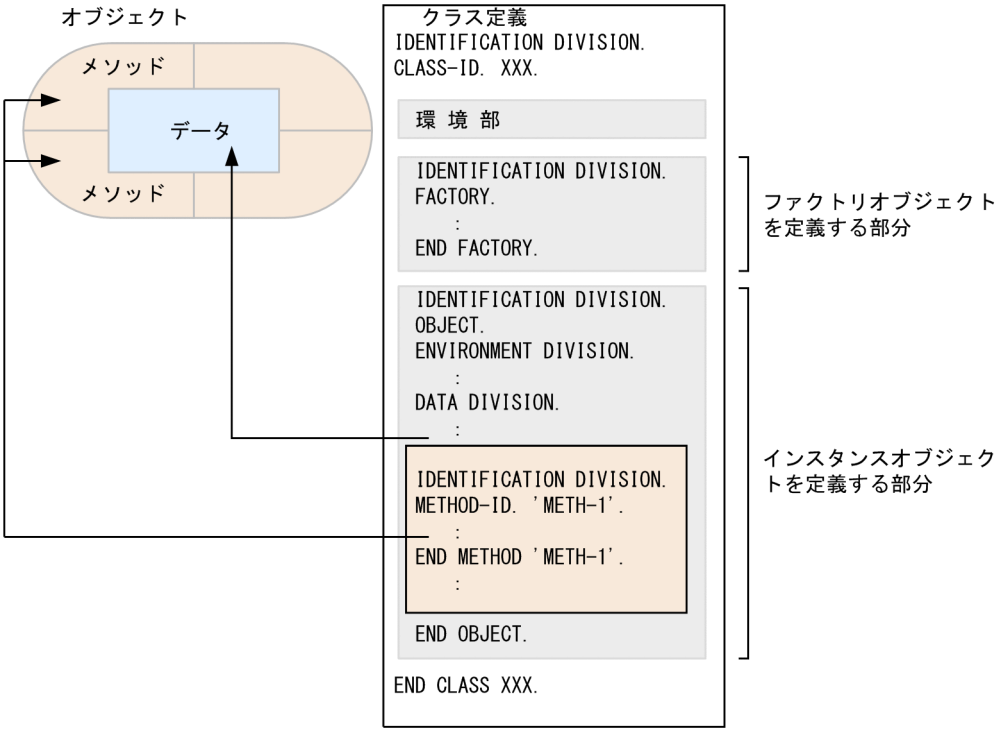
プログラム定義では，リポジトリ段落に使用するクラス名を指定します。また，インスタンスオブジェクトを生成する場合，オブジェクトを識別するためのオブジェクト参照を定義します。手続き部では，メソッドを呼び起こすための手続き文である INVOKE 文を記述します。

リポジトリ段落の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「8.2.7　リポジトリ段落（REPOSITORY)」を参照してください。

(b)　クラス定義

クラス定義は，インスタンス定義とファクトリ定義という二つの定義部分で構成されます。また，インスタンス定義，ファクトリ定義にはそれぞれメソッド定義を定義できます。ファクトリ定義から生成されるオブジェクトがファクトリオブジェクト，インスタンス定義から生成されるオブジェクトがインスタンスオブジェクトとなります。

それぞれのオブジェクトをどのように定義しているのかを，オブジェクトの概念図との対応で示します。



このクラス定義から，オブジェクトを生成します。

ファクトリオブジェクトは，クラス定義に対して一つだけ存在します。インスタンスオブジェクトは，クラス定義に対して複数生成できます。

ファクトリ定義，インスタンス定義，およびクラス定義の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「7.　見出し部（IDENTIFICATION DIVISION)」を参照してください。

20.2.2　インスタンスオブジェクトの生成と消滅

(1)　インスタンスオブジェクトの生成

定義したインスタンスオブジェクトは型であり，生成することで使用できるようになります。インスタンスオブジェクトの生成とは，処理対象であるデータ値を格納するための記憶領域を確保することです。

インスタンスオブジェクトを生成するためには，NEW メソッドという特別なメソッドを使用します。NEW メソッドは，COBOL2002 が提供する「BASE クラス」というクラスのファクトリオブジェクトのメソッドです。NEW メソッドおよび BASE クラスについては，マニュアル「COBOL2002　言語　標準仕様編」「12.　標準クラス」を参照してください。なお，NEW メソッドを指定する場合は，'NEW'と大文字で指定する必要があります。

次に，インスタンスオブジェクトの生成例を示します。

＜プログラム定義＞

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN-A.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
 REPOSITORY.
  CLASS 普通預金口座クラス.
      :
DATA DIVISION.
WORKING-STORAGE SECTION.

01 A USAGE OBJECT REFERENCE.
01 B PIC 9(4).
      :
PROCEDURE DIVISION.
      :
    INVOKE 普通預金口座クラス 'NEW'…1.
           RETURNING A.
      :
    INVOKE A '預け入れ' USING B.    …2.
      :
```

＜クラス定義＞

```
IDENTIFICATION DIVISION.
CLASS-ID. 普通預金口座クラス
    INHERITS BASE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
  CLASS BASE.
IDENTIFICATION DIVISION.
OBJECT.
      :
IDENTIFICATION DIVISION.
METHOD-ID. '預け入れ'.
      :
END METHOD '預け入れ'.
END OBJECT.
END CLASS 普通預金口座クラス.
```

1. ここでは，NEW メソッドを呼び起こして普通預金口座クラスのインスタンスオブジェクトを生成しています。インスタンスオブジェクトが生成されると，そのオブジェクト参照が返却項目に設定されます。

   オブジェクト参照とは，生成されたオブジェクトを参照するデータ項目です。オブジェクト参照に関する言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「5.2.2　オブジェクト参照」を参照してください。
2. ここでは，生成したインスタンスオブジェクトのオブジェクト参照を使用して，「預け入れ」メソッドを呼び起こしています。

(2)　インスタンスオブジェクトの消滅

生成されたインスタンスオブジェクトが消滅させられるのは，実行単位の終了時か，COBOL2002 が提供するガーベジコレクション（メモリ管理機能）によって不要と判断されたときです。

ガーベジコレクションとは，COBOL2002 実行時ライブラリのメモリ管理機能です。ガーベジコレクションが実行されると，どこからも参照されていないインスタンスオブジェクト※が整理され，そのメモリ領域が再利用されます。この処理は，COBOL2002 が自動的に実行し，ユーザが意識する必要はありません。

注※

オブジェクト参照データ項目は，SET 文で NULL が設定されるか，別のオブジェクト参照データ項目を転記された場合，インスタンスオブジェクトを参照しなくなります。

ガーベジコレクションは，NEW メソッドが実行されたとき，次の条件を満たすと発生します。

- インスタンスオブジェクトのメモリ使用の累積が，512 キロバイトを超えたとき。
  なお，環境変数 CBLGCSTART によって，上記の値を変更できます。
- 直前のガーベジコレクション完了時点からのメモリ追加量が，64 キロバイトを超えたとき。
  なお，環境変数 CBLGCINTERVAL によって，上記の値を変更できます。

環境変数 CBLGCSTART および環境変数 CBLGCINTERVAL については，「35.3　プログラムの実行環境の設定」を参照してください。

20.2.3　メソッドの呼び起こし（メッセージパッシング）

COBOL2002 でのメソッドの呼び起こし（メッセージパッシング）には，「INVOKE 文」を使用します。次に例を示します。

```
INVOKE CLASS-A 'HIKIDASHI' USING H-1 RETURNING H-2
         1          2             3           4
```

1. 呼び起こすメソッドを参照するためのクラス名またはオブジェクト参照
2. 呼び起こしたいメソッド名
3. メソッドに渡すデータ項目
4. メソッドから返される結果を受け取る項目の名称

(1)　メソッドの呼び起こし

COBOL2002 は，クラス名またはオブジェクト参照で指定したオブジェクトが実装しているメソッドを探します。INVOKE 文によって，オブジェクトに対しメソッドの実行を要求することを「メソッドを呼び起こす」といいます。メソッドを呼び起こす場合には，INVOKE 文にクラス名，オブジェクト参照データ項目名，または既定義オブジェクト参照を指定します。

INVOKE 文の文法規則については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.26 INVOKE 文」を参照してください。

(a)　ファクトリメソッドの呼び起こし

クラス名を指定すると，そのクラスに定義されているファクトリメソッドが呼び起こされます。記述例を，次に示します。

```
       INVOKE CLASS-A 'FACTORY-METHOD' USING ARG1
                 1                           RETURNING ARG2.
```

1. クラス名

(b)　インスタンスメソッドの呼び起こし

インスタンスメソッドを呼び起こす場合，そのメソッドを内包するインスタンスオブジェクトを指すオブジェクト参照を指定します。

オブジェクト参照とは，生成されたオブジェクトを参照するデータ項目です。オブジェクト参照は，USAGE IS OBJECT REFERENCE 句を使用して宣言します。オブジェクト参照の詳細はマニュアル「COBOL2002　言語　標準仕様編」「9.16.86　USAGE 句」を参照してください。

記述例を，次に示します。

```
        :
01 H-1 USAGE OBJECT-REFERENCE.
        :
     INVOKE CLASS1 'NEW' RETURNING H-1.
        :
     INVOKE H-1 'OBJECT-METHOD' USING ARG1
            1                   RETURNING ARG2.
```

1. オブジェクト参照

(c) 既定義オブジェクトを使用したメソッドの呼び起こし

現在実行中のメソッドの呼び起こし対象であるオブジェクトのメソッドを呼び起こす場合，既定義オブジェクト参照の SELF または SUPER を使用します。

記述例を，次に示します。

```
     INVOKE SELF 'OBJECT-METHOD' USING ARG1
            1                    RETURNING ARG2.
```

1. 既定義オブジェクト参照

既定義オブジェクト参照には，ほかに NULL および EXCEPTION-OBJECT があります。既定義オブジェクト参照については，マニュアル「COBOL2002　言語　標準仕様編」「4.3.2　一意名のいろいろ」を参照してください。

(2) メソッドの検索

INVOKE 文を実行すると，呼び起こすメソッドが検索されます。メソッドは，INVOKE 文の記述によって次のように検索されます。

- INVOKE クラス名 メソッド名

  指定したクラスのファクトリメソッドが検索されます。指定したクラスに該当するメソッドがなければ，指定したクラスのスーパクラスのファクトリメソッドが，継承した順に検索されます。

- INVOKE SELF メソッド名

  実行中のメソッドを呼び起こした際に指定したオブジェクトが属するクラスのファクトリメソッドまたはインスタンスメソッドが検索されます。指定したクラスに該当するメソッドがなければ，そのクラスのスーパクラスのメソッドが，継承した順に検索されます。

- INVOKE SUPER メソッド名

  実行中のメソッドを呼び起こした際に指定されたオブジェクトが属するクラスのスーパクラスのファクトリメソッドまたはインスタンスメソッドが検索されます。指定したクラスに該当するメソッドがなければ，そのクラスのスーパクラスのメソッドが，継承した順に検索されます。

- INVOKE 一意名 メソッド名

オブジェクト参照が指すオブジェクトがファクトリオブジェクトの場合は，そのファクトリ定義を含むクラスのファクトリメソッドが検索されます。オブジェクト参照が指すオブジェクトがインスタンスオブジェクトの場合は，そのインスタンス定義を含むクラスのインスタンスメソッドが検索されます。指定したクラスに該当するメソッドがなければ，そのクラスのスーパクラスのメソッドが，継承した順に検索されます。

(3) メソッドに渡す引数

メソッドに渡す引数を指定します。ここで指定した引数は，メソッド側に渡されます。詳細は，「17. プログラム間の引数と返却項目」を参照してください。

(4) メソッドから返される結果を受け取る項目の名称

オブジェクトのメソッドを実行させた結果の値を受け取るための，データ項目を指定します。

20.2.4 オブジェクトプロパティ

オブジェクトプロパティとは，オブジェクトが持つ属性の中で，外部からでもメソッドを呼び起こさず，直接参照または更新できると宣言されたデータ項目のことです。また，オブジェクトプロパティの参照および更新には，GET プロパティメソッド，SET プロパティメソッドを使用します。また，プロパティメソッドには，暗黙的なプロパティメソッドと明示的なプロパティメソッドとがあります。

オブジェクトプロパティの詳細についてはマニュアル「COBOL2002　言語　標準仕様編」「4.3.2(8) オブジェクトプロパティ」を参照してください。

(1) 明示的なプロパティメソッド

GET プロパティメソッド

メソッド名段落に GET PROPERTY を指定すると，そのメソッドは，GET プロパティメソッドとなります。

GET プロパティメソッドは，オブジェクトプロパティが送り出し側作用対象として参照されたとき，自動的に呼び起こされます。このとき，GET プロパティメソッドの RETURNING に指定された項目が返却されます。

SET プロパティメソッド

メソッド名段落に SET PROPERTY を指定すると，そのメソッドは，SET プロパティメソッドとなります。

SET プロパティメソッドは，オブジェクトプロパティが受け取り側作用対象として参照されたとき，自動的に呼び起こされます。このとき，引数として SET プロパティメソッドに値が渡されます。

記述例を，次に示します。

＜プログラム定義＞

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  MAIN-A.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
  CLASS CL-A.
  PROPERTY P-1.
    :
DATA DIVISION.
WORKING-STORAGE SECTION.
01 W-1 PIC X(10).
01 X-1 PIC X(10) VALUE 'ABC'.
PROCEDURE DIVISION.
    MOVE P-1 OF CL-A TO W-1.  …1.
         ｵﾌﾞｼﾞｪｸﾄﾌﾟﾛﾊﾟﾃｨ
    MOVE X-1 TO P-1 OF CL-A.  …2.
                ｵﾌﾞｼﾞｪｸﾄﾌﾟﾛﾊﾟﾃｨ
        :
END PROGRAM MAIN-A.
```

＜クラス定義＞

```
IDENTIFICATION DIVISION.
CLASS-ID.  CL-A
     INHERITS BASE.
    :
IDENTIFICATION DIVISION.
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 GET-DATA PIC X(10).
01 SET-DATA PIC X(10).
 ＜GETプロパティメソッド＞
 IDENTIFICATION DIVISION.
 METHOD-ID.  GET PROPERTY  P-1 .
                           ﾌﾟﾛﾊﾟﾃｨ名
 DATA DIVISION.
 LINKAGE SECTION.
 01 RET-DATA PIC X(10).
 PROCEDURE DIVISION
           RETURNING RET-DATA.
     MOVE GET-DATA TO RET-DATA.
     EXIT METHOD.
 END METHOD.

    :
 ＜SETプロパティメソッド＞
 IDENTIFICATION DIVISION.
 METHOD-ID.  SET PROPERTY  P-1 .
                           ﾌﾟﾛﾊﾟﾃｨ名
 DATA DIVISION.
 LINKAGE SECTION.
 01 PARA-DATA PIC X(10).
 PROCEDURE DIVISION
           USING PARA-DATA.
     MOVE PARA-DATA TO SET-DATA.
     EXIT METHOD.
 END METHOD.

    :
END FACTORY.
END CLASS CL-A.
```

1. プログラム MAIN-A からオブジェクトプロパティを送り出し側作用対象として参照します。これによって，GET プロパティメソッド P-1 が呼び起こされます。

   P-1 は，ファクトリデータ GET-DATA を返却項目 RET-DATA に設定し，呼び出し元へ制御を戻します。その結果，1.の W-1 には，GET-DATA と同じ値が転記されます。

2. プログラム MAIN-A からオブジェクトプロパティを受け取り側作用対象として参照します。これによって，SET プロパティメソッド P-1 が呼び起こされます。

   このとき，2.の MOVE 文では，X-1 の値を SET プロパティメソッド P-1 の引数に指定して呼び起こします。P-1 は，引数として渡された PARA-DATA をファクトリデータ SET-DATA に転記します。この結果，2.の MOVE 文の X-1 の値が SET-DATA に設定されます。

明示的なプロパティメソッドの詳細については，マニュアル「COBOL2002　言語　標準仕様編」「7.6 メソッド名段落（METHOD-ID)」を参照してください。

(2)　暗黙的なプロパティメソッド

ファクトリ定義およびインスタンス定義のデータ記述項に PROPERTY 句を指定すると，そのデータ項目は，オブジェクトプロパティとして参照できます。PROPERTY 句が指定されたデータ項目には，暗黙的に GET プロパティメソッドと SET プロパティメソッドが生成されます。

記述例を，次に示します。

＜プログラム定義＞

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  MAIN-A.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
  CLASS CL-A.
  PROPERTY P-1.
    :
DATA DIVISION.
WORKING-STORAGE SECTION.
01 W-1 PIC X(10).
01 X-1 PIC X(10) VALUE 'ABC'.
PROCEDURE DIVISION.
    MOVE P-1 OF CL-A TO W-1.  …1.
         ｵﾌﾞｼﾞｪｸﾄﾌﾟﾛﾊﾟﾃｨ
    MOVE X-1 TO P-1 OF CL-A.  …2.
                ｵﾌﾞｼﾞｪｸﾄﾌﾟﾛﾊﾟﾃｨ
    :
END PROGRAM MAIN-A.
```

＜クラス定義＞

```
IDENTIFICATION DIVISION.
CLASS-ID.  CL-A
     INHERITS BASE.
    :
IDENTIFICATION DIVISION.
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 P-1  PIC X(10) PROPERTY.
 ﾌﾟﾛﾊﾟﾃｨ名
    :
    :
END FACTORY.
END CLASS CL-A.
```

1. プログラム MAIN-A からオブジェクトプロパティを送り出し側作用対象として参照します。これによって，暗黙的な GET プロパティメソッドが呼び起こされます。

   暗黙的な GET プロパティメソッドは，ファクトリデータ P-1 の値を取得して返却します。その結果，1.の W-1 には，P-1 と同じ値が転記されます。

2. プログラム MAIN-A からオブジェクトプロパティを受け取り側作用対象として参照します。これによって，暗黙的な SET プロパティメソッドが呼び起こされ，2.の MOVE 文の X-1 の値がファクトリデータ P-1 に設定されます。

暗黙的なプロパティメソッドの場合，オブジェクトプロパティの値がそのままファクトリデータおよびインスタンスデータに設定されます。これに対して，明示的なプロパティメソッドの場合，オブジェクトプロパティの値を加工してファクトリデータおよびインスタンスデータに設定できます。

暗黙的なプロパティメソッドの詳細については，マニュアル「COBOL2002　言語　標準仕様編」「9.16.56　PROPERTY 句」を参照してください。

20.2.5　オブジェクト指向による継承

継承を行うには，COBOL2002 の「INHERITS 句」を使用します。

(1)　INHERITS 句

INHERITS 句は，クラスの"CLASS-ID. クラス名"に続けて指定します。

(例)

```
CLASS-ID. クラス名 INHERITS スーパクラス名.
```

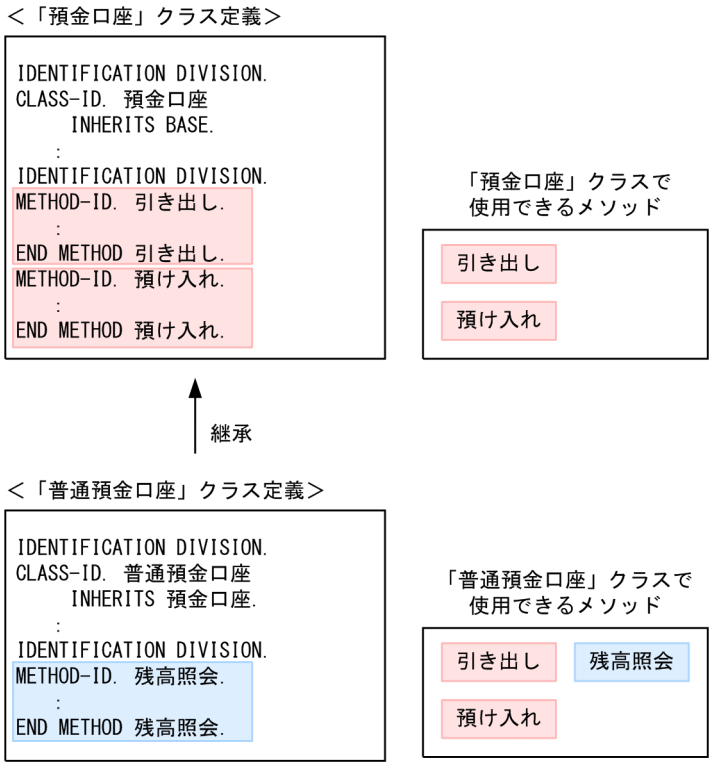
継承すると，スーパクラスの性質が，継承するクラスおよび継承するクラスのサブクラスにも定義されているかのように扱われます。継承の詳細については，マニュアル「COBOL2002　言語　標準仕様編」「5.2.9　クラス継承」を参照してください。

(2)　継承の使い方

「預け入れ」メソッドと「引き出し」メソッドを持つ「預金口座」クラスがある場合，この「預金口座」クラスを継承する例を，次に示します。

(例 1)

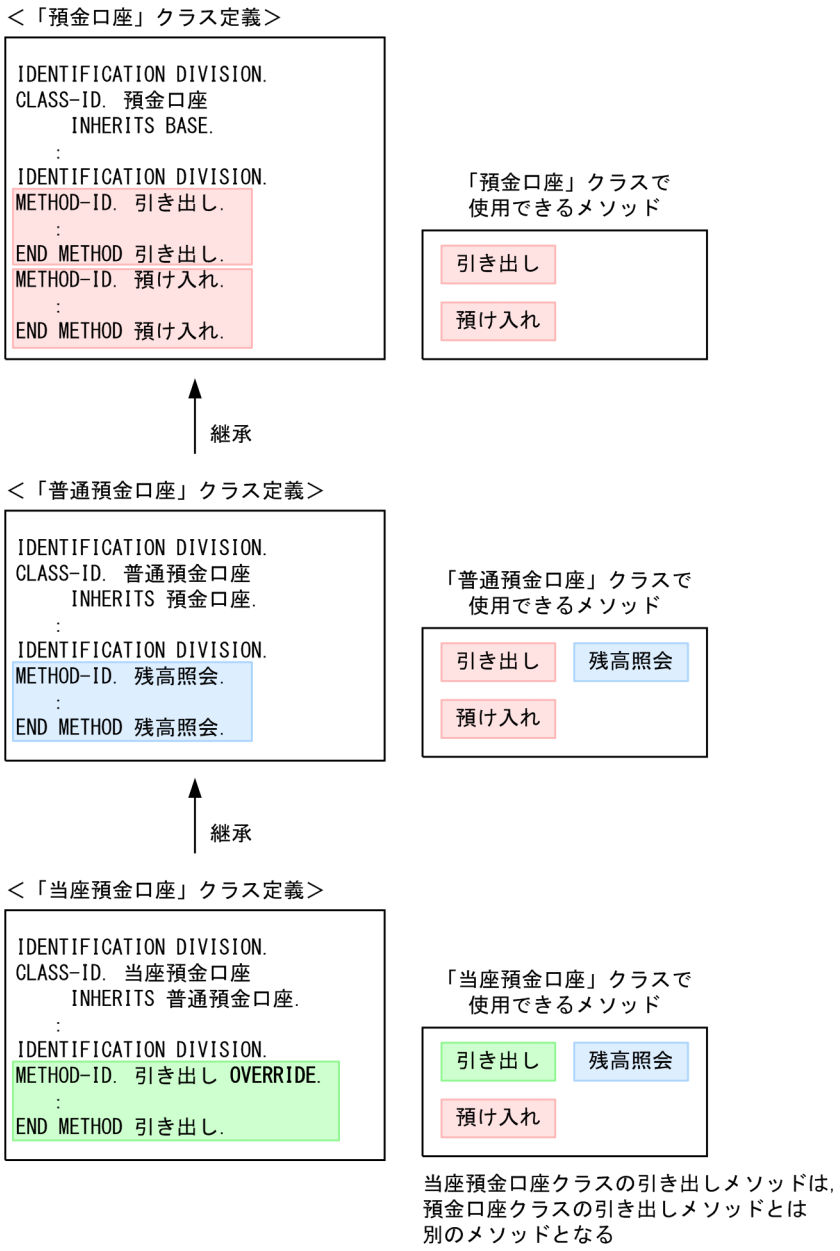
「預金口座」クラスの「預け入れ」メソッド，「引き出し」メソッドに加えて「残高照会」メソッドを持つ「普通預金口座」クラスを作成します。この場合，「普通預金口座」クラスは，「預金口座」クラスを継承し，「残高照会」メソッドだけを定義すれば，必要なメソッドを備えられます。



(例 2)

「預金口座」クラスの「預け入れ」メソッド，「引き出し」メソッドに加えて「残高照会」メソッドを持ち，さらに「引き出し」メソッドが独自の処理をする「当座預金口座」クラスを作成します。この場合，すでに「普通預金口座」クラスが存在するときは，「当座預金口座」クラスは，「普通預金口座」クラスを継承し，「引き出し」メソッドだけを定義すれば，必要なメソッドを備えられます。

「引き出し」メソッドは，継承する「預金口座」クラスですでに定義されているため，「当座預金口座」クラスで独自の「引き出し」メソッドを定義する場合は，METHOD-ID 段落に OVERRIDE 句を指定します。これによって，当座預金口座クラスの引き出しメソッドが預金口座クラスの引き出しメソッドを上書きし，当座預金口座クラスの引き出しメソッドが有効となります。



20.2.6　オブジェクト指向でのインタフェース

インタフェースとは，メソッドの名称，引数，および返却項目の定義の集まりです。インタフェースは，メソッドの実装を持ちません。

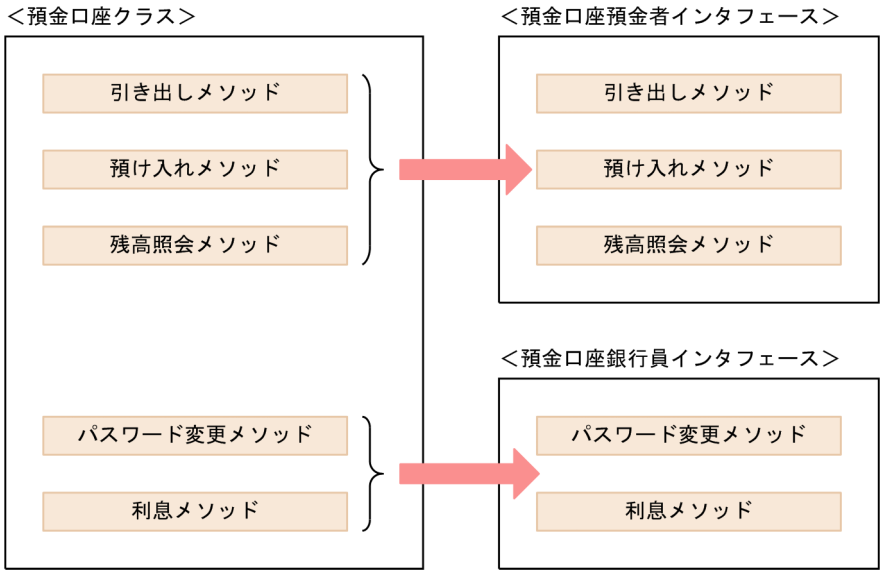
ファクトリオブジェクトとインスタンスオブジェクトは，それぞれインタフェースを持ちます。このインタフェースは，それぞれのオブジェクトが持つ，すべてのメソッド定義の集まりです。

インタフェースについては「COBOL2002　言語　標準仕様編」「5.2.7　適合とインタフェース」を参照してください。

(1)　インタフェースの利用

インタフェースを使用すると，クラス中の一部のメソッドだけを公開できます。例えば，プログラム A にはメソッド M1 と M2 だけを公開し，プログラム B にはメソッド M3 と M4 だけを公開することができ，誤ったメソッドの使用などを防止できます。

（インタフェースの利用例）



上記の「預金口座」クラスは，五つのメソッドを持っています。

このうち，「パスワード変更メソッド」と「利息メソッド」を，預金者用のプログラムから使用させたくない場合，「引き出しメソッド」「預け入れメソッド」「残高照会メソッド」だけを含む「預金口座預金者」インタフェースを定義し，預金者用のプログラムではこのインタフェースを使用します。

このようにすると，預金者用のプログラムからは，「パスワード変更メソッド」と「利息メソッド」を隠ぺいできるため，誤ってメソッドを使用されることがありません。

このように，インタフェースを利用すると，プログラムごとに公開するメソッドの範囲を変更できます。

インタフェースは，ファクトリメソッド，インスタンスメソッドのそれぞれに定義できます。クラスがインタフェースを持っていることを，「インタフェースを実装する」といいます。例えば，クラス A のファクトリ定義がインタフェース 1 を持つ場合は，「クラス A のファクトリオブジェクトは，インタフェース 1 を実装する」といいます。

また，クラス定義が未完成であっても，そのインタフェース部分だけが決まっていれば，-Repository,Gen オプションによってリポジトリファイルを作成できます。必要なリポジトリファイルを作成しておけば，リポジトリ段落に未完成のクラスの名前を指定した原始プログラムでも，コンパイルできるようになります。詳細は「33.3.2　リポジトリファイルの単独生成」を参照してください。

(2) インタフェースの定義

インタフェースは，INTERFACE-ID を使って定義します。INTERFACE-ID の詳細については，マニュアル「COBOL2002　言語　標準仕様編」「7.5　インタフェース名段落（INTERFACE-ID)」を参照してください。

インタフェースの定義例を，次に示します。

```
       IDENTIFICATION DIVISION.
       INTERFACE-ID. I-INT1. *> インタフェース名定義
       PROCEDURE DIVISION.
       IDENTIFICATION DIVISION.
       METHOD-ID. METH1. *> メソッド名定義
       DATA DIVISION.
       LINKAGE SECTION.
       01 PRM1 PIC X. *> 引数定義
       01 RET1 PIC X. *> 返却項目定義
       PROCEDURE DIVISION USING PRM1 RETURNING RET1.
           :
       END METHOD METH1. *> メソッドの処理は定義しない
       IDENTIFICATION DIVISION.
       METHOD-ID. METH2. *> 必要なメソッドをすべて定義する
           :
       END METHOD METH2.
       END INTERFACE I-INT1.
```

(3) インタフェースの実装

クラス定義にインタフェースを実装する場合，ファクトリ定義またはインスタンス定義の IMPLEMENTS 句にインタフェース名を指定します。IMPLEMENTS 句に指定されたインタフェース名は，そのクラスから生成されるファクトリオブジェクトまたはインスタンスオブジェクトに実装されたインタフェースとなります。IMPLEMENTS 句の詳細については，マニュアル「COBOL2002　言語　標準仕様編」「7.7　オブジェクト段落（OBJECT)」を参照してください。

```
       IDENTIFICATION DIVISION.
       CLASS-ID. CLS1.
            :
       IDENTIFICATION DIVISION.
       FACTORY. IMPLEMENTS F-INT1 F-INT2.
                    *> ファクトリオブジェクトが実装する
                    *> インタフェース名の指定
           :
       END FACTORY.

       IDENTIFICATION DIVISION.
       OBJECT. IMPLEMENTS I-INT1 I-INT2.
                    *> インスタンスオブジェクトが実装する
                    *> インタフェース名の指定
           :
       END OBJECT.
       END CLASS CLS1.
```

(4)　インタフェースを使ったオブジェクトの使用

インタフェースによってインスタンスオブジェクトを使用する場合，インタフェース名を指定したオブジェクト参照を使用します。

```
       01 OBJ-REF1 USAGE OBJECT REFERENCE I-INT1.
                                         インタフェース名

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 I-INT-OR USAGE OBJECT REFERENCE I-INT1.
                 *> インタフェース名を指定した
                 *> オブジェクト参照
       PROCEDURE DIVISION.
           INVOKE CLS1 'NEW' RETURNING I-INT-OR.
                 *> 「CLS1」クラスのインスタンス
                 *> オブジェクトの作成
           INVOKE I-INT1-OR 'METH1'.
                 *> インタフェースによるメソッド実行（1.）
```

インタフェース指定のオブジェクト参照によってメソッドを実行する場合，そのメソッドは，インタフェース定義中に定義されている必要があります。インタフェース中に定義されていないメソッドを実行しようとした場合，コンパイル時にエラーとなります。

上記の例で，メソッド「METH1」がインタフェース「I-INT1」の定義中にない場合は，インスタンスオブジェクト中に「METH1」があっても，1.の INVOKE 文がコンパイル時にエラーとなります。

また，クラスに実装されていないインタフェース名を指定したオブジェクト参照を使用してメソッドを呼び出した場合，コンパイル時にエラーとなります。例えば，上記の例で CLS1 のインスタンス定義にインタフェース名 I-INT1 の実装がない（IMPLEMENTS 句に指定されていない）場合，コンパイル時に適合エラーが発生します。適合については，「20.2.7　オブジェクト指向による適合」を参照してください。

20.2.7　オブジェクト指向による適合

オブジェクト参照データ項目にクラス名またはインタフェース名を指定すると，メソッドの呼び起こしやオブジェクト参照の転記の際に，適合がチェックされます。これによって，不当なメソッド呼び起こしや転記を防げます。適合するかどうかは，コンパイル時または実行時にチェックされます。

(1)　オブジェクト参照の適合チェック

通常，オブジェクトを使用する場合は，オブジェクト参照を使用します。

オブジェクト参照の定義にクラス名やインタフェース名を指定した場合，そのオブジェクト参照を使用したプログラムが正しく動作するかが判定されます。

オブジェクト参照への値設定は，SET 文の実行，メソッド呼び起こし時の引数への指定，または返却項目へのオブジェクト参照指定などによって発生します。オブジェクト参照へ値を設定する際，設定するデー

タの情報とオブジェクト参照に指定された情報との適合がチェックされ，適合するデータだけが設定できます。これによってオブジェクト参照が不適合なオブジェクトを指すことを防げます。

適合チェックの詳細については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.43　SET 文」，および「COBOL2002　言語　標準仕様編」「10.7　引数と返却項目の適合」を参照してください。

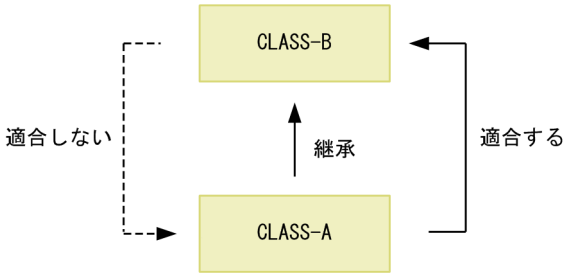
ここでは，適合チェックの概要について，説明します。

(a)　クラス同士またはインタフェース同士の適合

クラス同士，またはインタフェース同士の適合は，クラスおよびインタフェースの継承関係によって，次の規則で判定されます。

- あるクラス A に適合するクラスは，そのサブクラス（クラス A を継承するクラス），またはクラス A である。
- あるインタフェース B に適合するインタフェースは，インタフェース B を継承するインタフェース，またはインタフェース B である。

例えば，クラス「CLASS-A」がクラス「CLASS-B」を継承している場合，「CLASS-A は，CLASS-B に適合している」といえますが，「CLASS-B は，CLASS-A に適合していない」ことになります。これは，CLASS-B が CLASS-A を継承していないためです。




```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 OR1 USAGE OBJECT REFERENCE CLASS-A.
       01 OR2 USAGE OBJECT REFERENCE CLASS-B.
           :
       PROCEDURE DIVISION.
           :
          SET OR2 TO OR1. *> 1.
          SET OR1 TO OR2. *> 2.
```

1. 送り出し側作用対象である OR1 に指定された CLASS-A は，受け取り側作用対象である OR2 に指定された CLASS-B に適合しているので，この SET 文は正しいと判定されます。
2. 送り出し側作用対象である OR2 に指定された CLASS-B は，受け取り側作用対象である OR1 に指定された CLASS-A に適合していないので，この SET 文はエラーとなります。

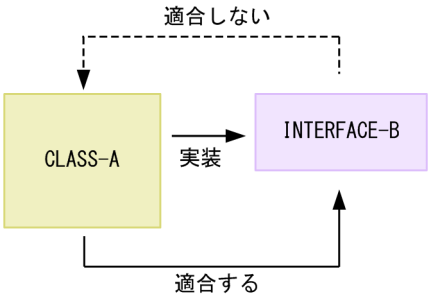
(b) インタフェースとクラスの適合

クラスとインタフェースの適合は，次の規則で判定されます。

- あるインタフェース A に適合するのは，それを実装するクラスである。

クラスとインタフェースが適合するのは，上記のとおり「あるクラスがあるインタフェースに適合する」場合だけです。「あるインタフェースがあるクラスに適合する」ということはありません。

例えば，クラス「CLASS-A」がインタフェース「INTERFACE-B」を実装している場合，「CLASS-A は INTERFACE-B に適合している」といえますが，「INTERFACE-B は CLASS-A に適合していない」ことになります。




```
       DATA  DIVISION.
       WORKING-STORAGE  SECTION.
       01 OR1 USAGE OBJECT REFERENCE CLASS-A.
       01 OR2 USAGE OBJECT REFERENCE INTERFACE-B.
           :
       PROCEDURE  DIVISION.
           :
           SET OR2 TO OR1. *> 1.
           SET OR1 TO OR2. *> 2.
```

1. 送り出し側作用対象である OR1 に指定された CLASS-A は，受け取り側作用対象である OR2 に指定された INTERFACE-B に適合しているので，この SET 文は正しいと判定されます。
2. 送り出し側作用対象である OR2 に指定された INTERFACE-B は，受け取り側作用対象である OR1 に指定された CLASS-A に適合していないので，この SET 文はエラーとなります。

(c) メソッド呼び起こしの適合

INVOKE 文によってメソッドを呼び起こす場合，そのメソッドが実行できるかどうかは，次の規則で判定されます。

- INVOKE 文で使用するオブジェクト参照にクラス名が指定されていれば，実行するメソッドは，そのクラスまたはそのクラスが継承するクラスのどちらかで定義されていなければならない。
- INVOKE 文で使用するオブジェクト参照にインタフェース名が指定されていれば，実行するメソッドは，そのインタフェースまたはそのインタフェースが継承するインタフェースのどちらかで定義されていなければならない。

クラス名またはインタフェース名指定のオブジェクト参照で，上記の規則に従っていないメソッドの呼び起こしがあると，コンパイル時にエラーとなります。

(2)　実装インタフェースの適合チェック

インタフェースを実装する場合，ファクトリ段落またはオブジェクト段落の IMPLEMENTS 句にインタフェース名を指定します。このとき，インタフェースは，ファクトリオブジェクトまたはインスタンスオブジェクトが実装するインタフェースに適合しているかどうかがチェックされます。このチェックでは，次の条件が成立する場合だけ，適合しているとみなされます（適合条件の詳細については，マニュアル「COBOL2002　言語　標準仕様編」「5.2.7　適合とインタフェース」を参照してください）。

- インタフェースが定義する各メソッドに対して，同名のメソッドがオブジェクトにも存在する。
- 同名メソッドの引数と返却項目の定義が同じである。

このため，実装されるインタフェースの持つメソッド定義の集合は，それを実装するオブジェクトが持つメソッド定義と同じ集合か，またはその部分集合となります。

例として，次のようなメソッド定義集合のインスタンスオブジェクトとインタフェースがあるとします。

クラス「CL-1」のインスタンスオブジェクトのメソッド定義集合

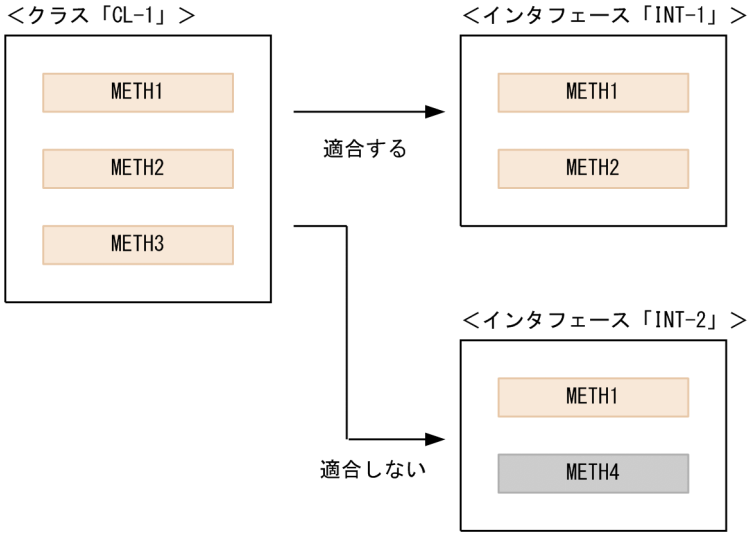
　METH1，METH2，METH3

インタフェース「INT-1」のメソッド定義集合

　METH1，METH2

インタフェース「INT-2」のメソッド定義集合

　METH1，METH4



インタフェース INT-1 のメソッドは，すべてクラス CL-1 に含まれています。このため，インタフェース INT-1 は，クラス CL-1 に適合しているといえます。

一方，インタフェース INT-2 は，クラス CL-1 が持たないメソッド METH4 を含んでいます。このため，インタフェース INT-2 は，クラス CL-1 に適合していないといえます。

インタフェース INT-1，INT-2 に IMPLEMENTS 句を指定すると，コンパイル時に次のように判定されます。

（インタフェース INT1 の場合）

```
       IDENTIFICATION DIVISION.
       CLASS-ID. CL-1.
           :
       OBJECT. IMPLEMENTS INT-1. …適合していると判定されます。
           :
       END OBJECT.
       END CLASS CL-1.
```

（インタフェース INT2 の場合）

```
       IDENTIFICATION DIVISION.
       CLASS-ID. CL-2.
           :
       OBJECT. IMPLEMENTS INT-2. …INT-2が持つMETH4がオブジェクト
           :                        にないため，適合エラーとなります。
       END OBJECT.
       END CLASS CL-2.
```

(3)　オブジェクトビューによる実行時の適合チェック

オブジェクトビューとは，一時的に AS の指定どおりの記述を持つように，オブジェクト参照を扱うものです。オブジェクトビューの言語規則については，マニュアル「COBOL2002　言語　標準仕様編」「4.3.2(4)　オブジェクトビュー」を参照してください。

オブジェクトビューをオブジェクト参照に指定すると，コンパイル時にオブジェクト参照の適合がチェックされないで，プログラム実行時にオブジェクト参照が指すオブジェクトとオブジェクトビューの指定内容の適合がチェックされます。チェックは，オブジェクト参照にオブジェクトビューの情報（クラス名，インタフェース名など）が設定された場合の，オブジェクト参照の適合チェックと同等となります。

オブジェクトビューを指定して，プログラム実行時に適合をチェックする例を，次に示します。

- クラス「CLASS-A」は，クラス「CLASS-B」を継承している。
- CLASS-A は，メソッド「METH-A」を，CLASS-B は，メソッド「METH-B」を，それぞれ持っている。

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 OR-B USAGE OBJECT REFERENCE CLASS-B.
           :
       PROCEDURE DIVISION.
           :
           INVOKE CLASS-A 'NEW' RETURNING OR-B.
```

```
        :
    INVOKE OR-B 'METH-A'. *> 1.
```

上記の場合，1.の INVOKE 文は，OR-B に指定された CLASS-B が METH-A を持たないため，コンパイル時にエラーとなります。しかし，OR-B が指すのは CLASS-A から作成されたインスタンスオブジェクトなので，METH-A を実装しています。このため，1.の INVOKE 文では，METH-A を実行できます。

このような場合，次のようにオブジェクトビューを指定すれば，コンパイルエラーとならないでプログラムを実行できます。

```
    INVOKE OR-B AS CLASS-A 'METH-A'.
                オブジェクトビュー
```

20.2.8　オブジェクト指向によるポリモルフィズム

ポリモルフィズムとは，一つのメッセージに対して異なる処理をすることを許す機能です。オブジェクト指向機能では，メソッドの呼び起こし（INVOKE 文）でポリモルフィズムができます。これは，「あるオブジェクト参照は，それに適合するオブジェクトのメソッドを呼び起こせる」というオブジェクト参照の特徴を利用したものです。

ポリモルフィズムの使用例を，次に示します。

（例）

「20.2.5　オブジェクト指向による継承」の「(2)　継承の使い方」の例の場合，クラスの継承関係は，次のようになっています。

「預金口座」クラス←「普通預金口座」クラス←「当座預金口座」クラス

（A←B は，B が A を継承することを表します）

また，「引き出し」メソッドは，オブジェクトごとに次のようになっていて，それぞれの「引き出し」メソッドの処理内容は異なります。

- 普通預金口座が実装する「引き出し」メソッド
  預金口座から継承したメソッド
- 当座預金口座が実装する「引き出し」メソッド
  当座預金口座で実装したメソッド

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 HA USAGE OBJECT REFERENCE 預金口座. *> 1.
    :
PROCEDURE DIVISION.
        :
    IF 要求 = 普通預金処理 *> 2.
      INVOKE 普通預金口座 'NEW' RETURNING HA
    ELSE
      INVOKE 当座預金口座 'NEW' RETURNING HA
```

```
END-IF.
    :
INVOKE HA '引き出し'. *> 3.
```

1. 継承関係による適合によって，普通預金口座と当座預金口座の両方が適合している預金口座クラスを指定したオブジェクト参照を使います。
2. 要求によって，普通預金口座を使うか当座預金口座を使うか切り分けます。
3. HA が指すオブジェクトにより，異なる「引き出し」メソッド（普通預金口座のメソッドまたは当座預金口座のメソッド）を実行できます。

20.2.9　オブジェクト指向機能でのマルチスレッド対応

マルチスレッド環境下で，複数のスレッドで同時に実行できる COBOL プログラムのことを，マルチスレッド対応 COBOL プログラムといいます。

ここでは，オブジェクト指向機能を使用したマルチスレッド対応 COBOL プログラムの特徴について説明します。マルチスレッド対応 COBOL プログラムについては，「26.　マルチスレッド環境での実行」を参照してください。

(1)　マルチスレッド環境下でのメソッドの実行

オブジェクト指向機能を使用したマルチスレッド対応 COBOL プログラムのスレッド呼び出しの対象は，プログラム定義および他言語のプログラムです。メソッドはスレッド呼び出しの対象とはなりません。マルチスレッド環境下でメソッドが実行されるのは，スレッド起動後に，プログラム定義から呼び出される場合だけです。

この場合，END METHOD によってメソッド処理が終了すると，COBOL 実行環境を終了しないで，呼び出し元に制御を返します。

また，スレッド実行中に STOP RUN 文が実行されると，実行中のスレッドを終了します。

(2)　マルチスレッド環境下でのファクトリデータ

マルチスレッド環境下のファクトリデータは，スレッド単位に存在するデータです。データの存在するスレッドだけで参照したり更新したりできます。ほかのスレッドからこのデータを参照したり更新したりすることはできません。

21 共通例外処理

例外処理とは，手続き文の実行中に発生したエラーに対して処理する機能です。COBOL2002 では，従来の COBOL で使用できた入出力機能での例外処理に加えて，さまざまなエラーに対応する処理を記述できます。ここでは，COBOL2002 で新しく追加された共通例外処理について説明します。

21.1　共通例外処理の概要

COBOL2002 では，プログラムでエラーが発生した場合に，そのエラーに対する回復処理を定義できます。これを，例外処理といいます。

COBOL2002 規格では，従来の COBOL から仕様が拡張され，より細かい例外を処理できるようになりました。

COBOL2002 で例外処理を定義する方法を，次に示します。

1. 手続き文固有の例外処理をする場合

   次の手続き文には，その文固有の例外処理を定義できます。

   - AT END 指定のある READ 文／SEARCH 文
   - AT EOP 指定のある WRITE 文
   - INVALID KEY 指定がある WRITE 文／REWRITE 文／DELETE 文／START 文
   - ON OVERFLOW／ON EXCEPTION 指定がある CALL 文
   - ON OVERFLOW 指定がある STRING 文／UNSTRING 文
   - ON SIZE ERROR 指定がある ADD 文／SUBTRACT 文／MULTIPLY 文／DIVIDE 文／COMPUTE 文

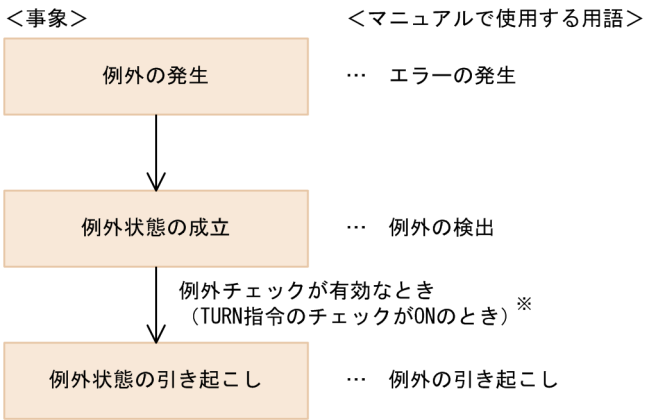
2. エラーの種類または入出力ファイル別に例外処理を定義する場合

   USE 文の宣言手続きを使用すると，発生するエラーの種類や入出力ファイルごとに例外処理を定義できます。

21.1.1　共通例外の仕組みと使用する用語

共通例外の発生から例外処理が実行される状態になるまでには，次のような共通例外の仕組みが規定されています。

図 21–1　共通例外の仕組み



注※

TURN 指令については，「21.3　TURN 指令」を参照してください。

この章で使用する例外に関する用語について，次に定義します。

この章で使用する用語	意味
例外	例外状態を示します。
TURN 指令のチェックが ON	TURN 指令で，該当する例外名が CHECKING ON 指定であり，指定された例外が引き起こせる状態であることを示します。
従来形式の例外処理	従来の COBOL の言語仕様にある，USE 手続きや INVALID KEY などの文ごとに指定できる無条件文を示します。これに対して，COBOL2002 で新たに追加された例外処理を「共通例外処理」といいます。

21.1.2　共通例外処理の機能

COBOL2002 で新たに追加された例外処理を「共通例外処理」といいます。共通例外処理は，従来形式の例外処理から次の機能が拡張されています。

1. 例外宣言手続きの実行

   従来の例外処理では，入出力機能の例外が検出された場合に対してだけ，宣言手続きが定義できましたが，共通例外処理では，さまざまな機能に対する例外宣言手続きを定義できます。また，宣言手続きから指定した手続き名へ復帰できます。

2. 呼び出し元プログラムへの例外の伝播

   検出された例外の情報を，呼び出し元プログラムに伝えられます。このことを，例外の伝播といいます。呼び出し元プログラムでは，伝播によって例外を検出し，例外処理を実行できます。

3. 例外を引き起こす文（RAISE 文）を使用して，明示的に例外を引き起こせます。

4. 引き起こした例外の情報を，プログラム中で組み込み関数などを使用して参照できます。

21.1.3　共通例外処理の使用例

プログラムに共通例外処理を追加する例を，次に示します。

この例では，ACCEPT 文に内部浮動小数点項目を指定しているため，-NumAccept コンパイラオプションの指定が必要です。

＜主プログラム：SAMPLE1.CBL（変更前）＞

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
         :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
```

```
       01 PARAM USAGE COMP-1.
         :
       PROCEDURE DIVISION.

         DISPLAY ’値を入力してください ’.
         ACCEPT PARAM.
         CALL ’SAMPLE2’ USING PARAM.

       END PROGRAM SAMPLE1.
```

＜副プログラム：SAMPLE2.CBL（変更前）＞

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
           :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ANS USAGE COMP-1.
       LINKAGE SECTION.
       01 PARAM USAGE COMP-1.
           :
       PROCEDURE DIVISION USING PARAM.

           COMPUTE ANS = FUNCTION ACOS ( PARAM ).
           DISPLAY ’ANSWER  =’ ANS.

       END PROGRAM SAMPLE2.
```

(変更前のプログラムの説明)

主プログラム SAMPLE1 で入力された値について，副プログラム SAMPLE2 内で ACOS 関数を使用して逆余弦の近似値を求めるプログラムです。入力した値が不正な場合，実行時エラーメッセージが出力され，プログラムが異常終了します。

不正な値が入力された場合に共通例外処理を使用して例外を処理するために，上記のプログラムを次のように修正します。

1. 組み込み関数の引数エラーについて，例外チェックを有効にする。

組み込み関数の引数エラーについて例外を引き起こすには，翻訳指令の TURN 指令を指定して，対応する例外名 EC-ARGUMENT-FUNCTION のチェックを ON にします。

```
>>TURN EC-ARGUMENT-FUNCTION CHECKING ON
```

TURN 指令については，「21.3　TURN 指令」を参照してください。

2. 宣言手続きを記述する。

共通例外処理として，EC-ARGUMENT-FUNCTION 例外発生時の宣言手続きを記述します。宣言手続きは，USE 文を使用して次のように記述します。

```
DECLARATIVES.
USE-EC-ARGUMENT SECTION.
  USE AFTER EXCEPTION CONDITION EC-ARGUMENT-FUNCTION.
    DISPLAY ’入力値不正!!’.
```

```
    RESUME AT ERR-RETRY.
END DECLARATIVES.
```

この例題では，主プログラム SAMPLE1 で共通例外処理するため，SAMPLE1 に宣言手続きを記述します。宣言手続きについては，「21.4　共通例外の宣言手続き」を参照してください。

3. 自動伝播を有効にする。

呼び出し先プログラム（SAMPLE2）で引き起こした例外を，呼び出し元プログラム（SAMPLE1）に自動伝播するためには，PROPAGATE 指令を ON にします。

```
>>PROPAGATE ON
```

伝播については，「21.5　例外の伝播」を参照してください。

上記の例外処理を追加したサンプルプログラムを，次に示します。**太字（色付き）**の個所が，追加部分です。

＜主プログラム：SAMPLE1.CBL（変更後）＞

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
            :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 PARAM USAGE COMP-1.
            :
       PROCEDURE DIVISION.
      DECLARATIVES. *> 3.
        USE-EC-ARGUMENT SECTION.
          USE AFTER EXCEPTION CONDITION
                                EC-ARGUMENT-FUNCTION.
          DISPLAY ’入力値不正!!’.
          RESUME AT ERR-RETRY. *> 4.
      END DECLARATIVES.

      ERR-RETRY.
        DISPLAY ’値を入力してください ’.
        ACCEPT PARAM.
      >>TURN EC-ARGUMENT-FUNCTION CHECKING ON
        CALL ’SAMPLE2’ USING PARAM. *> 2.

       END PROGRAM SAMPLE1.
```

＜副プログラム：SAMPLE2.CBL（変更後）＞

```
      >>PROPAGATE ON
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
         :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ANS USAGE COMP-1.
       LINKAGE SECTION.
       01 PARAM USAGE COMP-1.
         :
       PROCEDURE DIVISION USING PARAM.
```

```
       >>TURN EC-ARGUMENT-FUNCTION CHECKING ON
           COMPUTE ANS = FUNCTION ACOS ( PARAM ). *> 1.
           DISPLAY 'ANSWER  =' ANS.

       END PROGRAM SAMPLE2.
```

（変更後のプログラムの説明）

このプログラムを実行すると，SAMPLE2 の COMPUTE 文（1.）で組み込み関数 ACOS の引数エラーが発生した場合，次のように例外宣言処理が実行されます。

1. SAMPLE2 の COMPUTE 文で組み込み関数 ACOS の引数エラーが発生します。
2. 組み込み関数の引数エラーに対応する例外名 EC-ARGUMENT-FUNCTION の TURN 指令のチェックが ON のため，例外が引き起こされます。このとき，PROPAGATE 指令が ON になっているので，SAMPLE1 の CALL 文に例外が伝播します。
3. 例外宣言手続きが実行されます。
4. RESUME 文によって，例外宣言手続きから ERR-RETRY 節に制御が戻ります。

21.1.4　共通例外処理に対応している機能

COBOL2002 で共通例外処理に対応している機能を，次に示します。

表 21-1　共通例外が使用できる機能一覧

種類	機能名	共通例外への対応	備考
規格	基本機能	○	
	順編成ファイル	○	
	相対編成ファイル	○	
	索引編成ファイル	○	
	整列併合	△※1	
	プログラム間連絡	△	詳細については，「21.8.2　例外検出での注意事項」の「(2)　プログラム間連絡での注意事項」を参照のこと。
	組み込み関数	○	
	オブジェクト指向	○	
	再帰呼び出し	○	
	利用者定義関数	○	
	局所場所節（LOCAL-STORAGE SECTION）	○	

種類	機能名	共通例外への対応	備考
X/Open	テキスト編成ファイル	○	
	ファイル共用（ファイルシェア）	×	
	コマンド行および環境変数へのアクセス	×	
	画面節（SCREEN SECTION）による画面操作※3	×	
	C 言語インタフェース	×	
	インタナショナリゼーション	×	
拡張機能	日本語	×	
	ブール（ビット操作）	○	
	アドレス操作	○	
	1 バイト 2 進機能・COMP-X 項目	○	
	浮動小数点項目	○	
	報告書作成機能	×	
	HiRDB による索引編成ファイル※3	○	
	CSV 編成ファイル	○	
	ラージファイル入出力	○	
	ファイル入出力拡張機能	○	
	プリンタへのアクセス（入出力による書式印刷機能）※2	○	
	ファイルのディスク書き込み保証	○	
	通信節による画面操作※2	×	
	画面節（WINDOW SECTION）による画面操作※3	×	
	データコミュニケーション機能	×	
	データベース操作機能	×	
	XDM によるデータベースシミュレーション機能	×	
	マルチスレッド環境での実行	○※4	
	基本機能サービスルーチン	×	
	CGI プログラム作成支援機能※2	×	
	COBOL 入出力サービスルーチン	×	
	バイトストリーム入出力サービスルーチン	×	
	Unicode 機能	○	
	数字項目のけた拡張機能※5	△※6	

種類	機能名	共通例外への対応	備考
	動的長基本項目機能	○	
	定数長拡張機能	○	
デバッグ	実行時デバッグ機能	○	
	テストデバッグ機能	○	
	カバレージ機能	○	

（凡例）

○：共通例外処理を使用できる

△：共通例外処理を制限付きで使用できる

×：エラーは発生するが，共通例外処理を使用できない

注※1

整列併合処理の共通例外処理への対応を，次に示します。

処理	共通例外への対応
SORT 文	使用できない
MERGE 文	使用できない
SORT／MERGE 文の USING／GIVING に指定したファイルの入出力処理	使用できない
整列併合処理の入出力手続きに指定したファイルの入出力処理	使用できる
整列処理の RELEASE 文	使用できる
整列処理の RETURN 文	使用できる （ただし，併合処理に使用している整列併合ファイルに対する，整列処理の RETURN 文では使用できない）
併合処理の RETURN 文	使用できない

注※2

AIX(32)で有効です。

注※3

AIX で有効です。

注※4

-MultiThread オプションを指定したプログラムと-MultiThread オプションを指定していないプログラムが混在して動作した場合，UNIX では例外を検出しません。

注※5

AIX(64)，Linux(x64)で有効です。

注※6

数字項目のけた拡張機能を使用する場合，EC-SIZE 例外を使用できません。詳細については，「28.　数字項目のけた拡張機能（AIX(64)，Linux(x64)で有効）」を参照してください。

21.2　例外

例外は，手続き文の実行中にエラーが発生した場合，または例外を伝播した場合に検出されます。検出された例外に対して共通例外処理を実行するには，例外に対応する例外名を，TURN 指令で ON に指定しておきます。また，RAISE 文を実行した場合は，無条件に指定した例外が引き起こされます。

共通例外処理は，引き起こされた例外に基づいて処理を実行します。例外は，例外名，または例外オブジェクトとして表されます。

21.2.1　例外名

例外名とは，おのおのの例外に関連づけられた名前のことです。例えば，「0 による除算」の例外は，例外名「EC-SIZE-ZERO-DIVIDE」として識別されます。

(1)　例外名のレベル

例外名には，次の三つのレベルがあります。

レベル 3

最下位のレベルに位置する例外名です。

レベル 3 に分類される例外名は，おのおのの例外を表します。

レベル 2

レベル 3 の上位に位置する例外名です。

レベル 2 に分類される例外名は，レベル 3 例外名を機能や分類によってまとめたものです。機能・分類単位の例外を包括して指定したい場合に使用します。

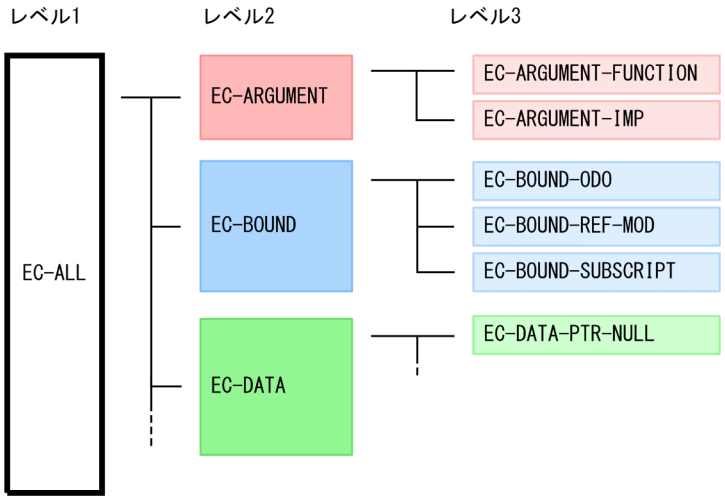
レベル 1

最上位に位置する例外名です。

レベル 1 に分類される例外名は，「EC-ALL」だけです。「EC-ALL」を指定すると，すべての例外名を指定したのと同じように処理されます。

すべての例外を包括して指定したい場合に使用します。

図 21-2　例外名のレベル構成



(2)　例外名の一覧

例外名の一覧を，次に示します。

表 21-2　例外名の一覧

例外名	致命度	説明	検出される文
EC-ALL	−	すべての例外	−
EC-ARGUMENT	−	引数エラー	−
EC-ARGUMENT-FUNCTION	致命的	関数で引数エラーが発生した	組み込み関数を指定できる手続き文※1
EC-ARGUMENT-IMP	致命的	組み込み関数の処理中にメモリ不足などのエラーが発生した	組み込み関数を指定できる手続き文※1
EC-BOUND	−	区域外	−
EC-BOUND-ODO	致命的	OCCURS～DEPENDING ON データ項目が区域外である※2	OCCURS DEPENDING ON 一意名に値を設定する手続き文※1
EC-BOUND-REF-MOD	致命的	区域外の部分参照子である※3	部分参照を指定できる手続き文※1
EC-BOUND-SUBSCRIPT	致命的	区域外の添字である※4	添字を指定できる手続き文※1
EC-DATA	−	データ例外	−
EC-DATA-PTR-NULL	致命的	アドレス名によって参照されるデータ項目を参照するとき，アドレス名の設定値が NULL である	アドレス名によって参照されるデータ項目を指定した手続き文※5
EC-FLOW	−	実行制御フロー違反	−

例外名	致命度	説明	検出される文
EC-FLOW-GLOBAL-EXIT	致命的	大域的な宣言手続き中で EXIT PROGRAM 文が実行された	EXIT 文
EC-FLOW-GLOBAL-GOBACK	致命的	大域的な宣言手続き中で GOBACK 文が実行された	• GOBACK 文 • 大域的な宣言手続き中での自動伝播の対象となる例外を引き起こした手続き文
EC-FLOW-IMP	致命的	ALTER 文で行き先を指定する前に，行き先を省略した GO TO 文が実行された	GO TO 文
EC-FLOW-RELEASE	致命的	RELEASE 文が SORT 文の範囲内にない	RELEASE 文
EC-FLOW-RETURN	致命的	RETURN 文が MERGE 文や SORT 文の範囲内にない	RETURN 文
EC-FLOW-USE	致命的	USE 文が別の USE 手続きを実行させた	宣言手続き中で実行中の宣言手続きを実行する例外を引き起こした手続き文
EC-I-O	−	入出力例外	−
EC-I-O-AT-END	非致命的	入出力状態「1x」が発生した	READ 文
EC-I-O-EOP	非致命的	ページ終了条件が発生した	WRITE 文
EC-I-O-EOP-OVERFLOW	非致命的	ページあふれ条件が発生した	WRITE 文
EC-I-O-IMP	致命的	入出力状態「9x」が発生した（入出力状態の詳細は，「付録 G　入出力状態の値」を参照）	• OPEN 文 • CLOSE 文 • READ 文 • WRITE 文 • REWRITE 文 • DELETE 文 • START 文
EC-I-O-INVALID-KEY	非致命的	入出力状態「2x」が発生した	• READ 文 • REWRITE 文 • START 文 • WRITE 文 • DELETE 文
EC-I-O-LINAGE	致命的	LINAGE 句に指定したデータ名の値が要求範囲外である	• WRITE 文 • OPEN 文
EC-I-O-LOGIC-ERROR	致命的	入出力状態「4x」が発生した	• READ 文 • CLOSE 文 • REWRITE 文

例外名	致命度	説明	検出される文
			• DELETE 文 • START 文 • OPEN 文 • WRITE 文
EC-I-O-PERMANENT-ERROR	致命的	入出力状態「3x」が発生した	• READ 文 • OPEN 文 • WRITE 文 • REWRITE 文 • DELETE 文
EC-OO	−	オブジェクト指向に関連するすべての既定義例外	−
EC-OO-CONFORMANCE	致命的	オブジェクトビューに対する不成功が発生した	オブジェクトビューを指定できる手続き文
EC-OO-EXCEPTION	致命的	例外オブジェクトが処理されなかった	• EXIT 文 • GOBACK 文
EC-OO-IMP	致命的	INVOKE 文の一意名 1 に指定されたハンドルの値が正しくない	INVOKE 文
EC-OO-METHOD	致命的	要求されるメソッドが使用できない	INVOKE 文
EC-OO-NULL	致命的	NULL オブジェクト参照を使用してメソッドを呼び起こそうとした	INVOKE 文
EC-OO-RESOURCE	致命的	オブジェクトの作成や拡張に必要なシステム資源が不十分である	INVOKE 文
EC-OO-UNIVERSAL	致命的	実行時の型チェックが失敗した	INVOKE 文
EC-OVERFLOW	−	けたあふれ条件	−
EC-OVERFLOW-STRING	非致命的	STRING 文のけたあふれ条件が発生した	STRING 文
EC-OVERFLOW-UNSTRING	非致命的	UNSTRING 文のけたあふれ条件が発生した	UNSTRING 文
EC-PROGRAM	−	プログラム間連絡例外	−
EC-PROGRAM-ARG-MISMATCH	致命的	引数が不一致である	CALL 文
EC-PROGRAM-CANCEL-ACTIVE	致命的	取り消されるプログラムが活性状態である	CANCEL 文
EC-PROGRAM-IMP	致命的	• 共用ライブラリをロード中にエラーが発生した • CALL 文で実行可能ファイルの処理中にエラーが発生した	• CALL 文 • 利用者定義関数を指定できる手続き文

例外名	致命度	説明	検出される文
		• EXTERNAL 句を指定したデータ項目やファイル定義を処理中にエラーが発生した	
EC-PROGRAM-NOT-FOUND	致命的	呼び出し先プログラムが見つからない	CALL 文
EC-PROGRAM-RECURSIVE-CALL	致命的	呼び出し先プログラムが活性状態である	CALL 文
EC-PROGRAM-RESOURCES	致命的	呼び出し先プログラムで資源が使用できない	• CALL 文 • 利用者定義関数を指定できる手続き文
EC-RAISING	−	EXIT 文 RAISING 指定か GOBACK 文 RAISING 指定で発生する例外	−
EC-RAISING-NOT-SPECIFIED	致命的	EXIT 文 RAISING 指定か GOBACK 文 RAISING 指定の EC-USER 例外条件が手続き部見出しの RAISING 指定にない	• EXIT 文 • GOBACK 文
EC-RANGE	−	範囲例外	−
EC-RANGE-INSPECT-SIZE	致命的	INSPECT 文での置き換え項目のサイズが異なる	INSPECT 文
EC-RANGE-INVALID	非致命的	THROUGH 範囲の開始値が終了値よりも大きい	• EVALUATE 文 • 条件名指定の IF 文
EC-RANGE-PERFORM-VARYING	致命的	PERFORM 文で変更項目の設定が負値である	PERFORM 文
EC-RANGE-SEARCH-INDEX	非致命的	指標の初期値が範囲外のため，SEARCH 文で表要素が見つからない	SEARCH 文
EC-RANGE-SEARCH-NO-MATCH	非致命的	探索基準に合致する要素がないため，SEARCH 文で表要素が見つからない	SEARCH 文
EC-SIZE	−	けたあふれ例外	−
EC-SIZE-EXPONENTIATION	致命的	べき乗演算規則の違反が発生した	COMPUTE 文
EC-SIZE-OVERFLOW	致命的	計算でのけたあふれが発生した	• ADD 文 • COMPUTE 文 • MULTIPLY 文 • DIVIDE 文 • SUBTRACT 文
EC-SIZE-TRUNCATION	致命的	格納での有効けた切り捨てが発生した	• MOVE 文 • COMPUTE 文

例外名	致命度	説明	検出される文
			• ADD 文 • SUBTRACT 文 • DIVIDE 文 • MULTIPLY 文
EC-SIZE-UNDERFLOW	致命的	浮動小数点での下位けたあふれが発生した	• ADD 文 • COMPUTE 文 • MULTIPLY 文 • DIVIDE 文 • SUBTRACT 文
EC-SIZE-ZERO-DIVIDE	致命的	ゼロによる除算が発生した	• COMPUTE 文 • DIVIDE 文
EC-SORT-MERGE	−	整列併合機能の例外	−
EC-SORT-MERGE-IMP	致命的	整列併合用ファイル記述項の RECORD 句で指定した DEPENDING ON データ名の値が数字ではない	• RELEASE 文 • RETURN 文
EC-SORT-MERGE-RELEASE	致命的	RELEASE 文のレコードが長過ぎるまたは短過ぎる	RELEASE 文
EC-SORT-MERGE-RETURN	致命的	RETURN 文がファイル終了条件発生中に実行された	RETURN 文
EC-USER	−	利用者定義の例外	−
EC-USER-（ユーザ定義例外名）	非致命的	レベル 3 の利用者定義の例外条件	• RAISE 文 • EXIT 文 • GOBACK 文

（凡例）

−：発生したレベル 3 の例外名の動作に従う

注※1

詳細は「21.8.1　例外が検出される文の詳細」を参照してください。

注※2

表操作で，DEPENDING ON 指定のデータ名に格納されている繰り返し回数が，表の最小から最大の範囲にないことを示します。

注※3

部分参照の指定が，一意名の範囲内にないことを示します。

注※4

表操作で使用する添字または指標名が指す表要素が，表の範囲内にないことを示します。

注※5

ただし，連絡節中のデータ項目に対するポインタ変数が NULL である場合，例外が検出されません。

（例）

連絡節中のデータ項目に対して SET 文の ADDRESS OF によって NULL を設定したあと，そのデータ項目を参照した場合は，例外が検出されません。

(3)　利用者定義例外名

利用者定義例外名とは，ユーザが独自の例外を定義して使用できる機能です。

利用者定義例外名は，「EC-USER-」で始まる任意の例外名を記述して定義します。利用者定義例外名の命名規則については，マニュアル「COBOL2002　言語　標準仕様編」「10.5.11(1)　例外」を参照してください。

利用者定義例外名として，「EC-USER-EXCEPTION」という例外名を使用する場合のコーディング例を，次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE.

       PROCEDURE DIVISION.
       DECLARATIVES.
         EXCEPTIONHANDLER SECTION.
           USE AFTER EXCEPTION CONDITION EC-USER-EXCEPTION. *> 1.
             DISPLAY  'EC-USER-EXCEPTION発生'.
       END DECLARATIVES.

           RAISE EXCEPTION EC-USER-EXCEPTION. *> 2.

       END PROGRAM SAMPLE.
```

1. 例外宣言手続き部に利用者定義例外名「EC-USER-EXCEPTION」を定義します。
2. RAISE 文に EC-USER-EXCEPTION を指定して，利用者定義の例外を引き起こします。

   RAISE 文で例外を引き起こす場合，TURN 指令のチェックを ON にして例外チェックを有効にする必要はありません。

   RAISE 文については，「21.6　明示的な例外の引き起こし」を参照してください。

21.2.2　例外オブジェクト

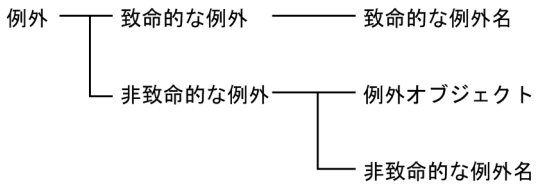
例外オブジェクトは，オブジェクト指向機能を使用した COBOL プログラムで例外を取得するときに，例外名に関連づけられた例外と同じように使用します。例外オブジェクトは，例外名に関連づけられた例外と比べて，多様な例外状況を取得できます。例外オブジェクトの詳細については，「21.7.2　EXCEPTION-OBJECT」を参照してください。また，オブジェクト指向機能については，「20.　オブジェクト指向機能」を参照してください。

21.2.3　例外の致命度

例外には致命的例外と非致命的例外の 2 種類があります。

例外の致命度と例外名／例外オブジェクトとの対応を，次に示します。

図 21–3　例外の致命度と例外名／例外オブジェクトとの対応



例外が発生した場合の動作，および共通例外処理での例外処理は，例外の致命度によってそれぞれ次のように異なります。

致命的例外の場合

実行単位が異常終了します。

非致命的例外の場合

実行が継続されます。

ただし，例外宣言手続きを記述することで，致命度に関係なく，例外発生時に対応する宣言手続きが実行できます。宣言手続き実行後の動作については，「21.4　共通例外の宣言手続き」を参照してください。

例外の種類と致命度の対応については，「21.2.1　例外名」の「(2)　例外名の一覧」を参照してください。

なお，利用者定義例外名は，すべて非致命的な例外となります。

21.2.4　最新例外状態

実行単位で最後に引き起こされた例外は，最新例外状態として保持されます。最新例外状態は，例外が検出されていない状態か，検出されていれば例外に対応する例外名，例外オブジェクトのどちらかを示します。

最新例外状態は，例外が引き起こされた場合に更新され，例外が引き起こされないで実行が完了したか，SET LAST EXCEPTION TO OFF によって最新状態がクリアされるまで保持されます。最新例外状態は，次の場合に更新されます。

- 手続き文の実行中にエラーが発生し，例外が引き起こされた場合
- RAISE 文を実行した場合
- EXIT，GOBACK 文の RAISING 指定に例外名または一意名を指定して，例外を伝播し，伝播先のプログラムで例外が引き起こされた場合。

最新例外状態は，組み込み関数を使用して参照できます。詳細は，「21.7.1　組み込み関数を使用した例外情報の参照」を参照してください。

また，最新例外状態のクリアについては，「21.7.3　最新例外状態のクリア」を参照してください。

21.3　TURN 指令

特定の例外に対する共通例外処理は，TURN 指令によって対応する例外名のチェックを ON にすることで，指定します。ただし，RAISE 文や例外オブジェクトで引き起こされる例外だけを扱う場合は，TURN 指令のチェックを ON にする必要はありません。

21.3.1　TURN 指令によるチェック

TURN 指令に例外名を指定してチェックを ON にすると，例外名に対する例外を引き起こせます。

TURN 指令には，例外を表すレベル 3 例外名以外にも，レベル 2 またはレベル 1 の例外名を指定できます。また，EC-I-O 例外名については，ファイル名を指定して，そのファイル名に対してだけチェックを ON，または OFF にできます。

TURN 指令については，マニュアル「COBOL2002　言語　標準仕様編」「3.3.15　TURN 指令」を参照してください。

TURN 指令に指定する例外名のレベルと，その TURN 指令の対象となる例外名の対応を，次に示します。

表 21–3　TURN 指令と対象の例外名

TURN 指令に指定する例外名のレベル	対象となる例外名
レベル 1	すべてのレベル 3 例外名
レベル 2	指定した機能，分類に対応するレベル 3 例外名
レベル 3	指定したレベル 3 例外名

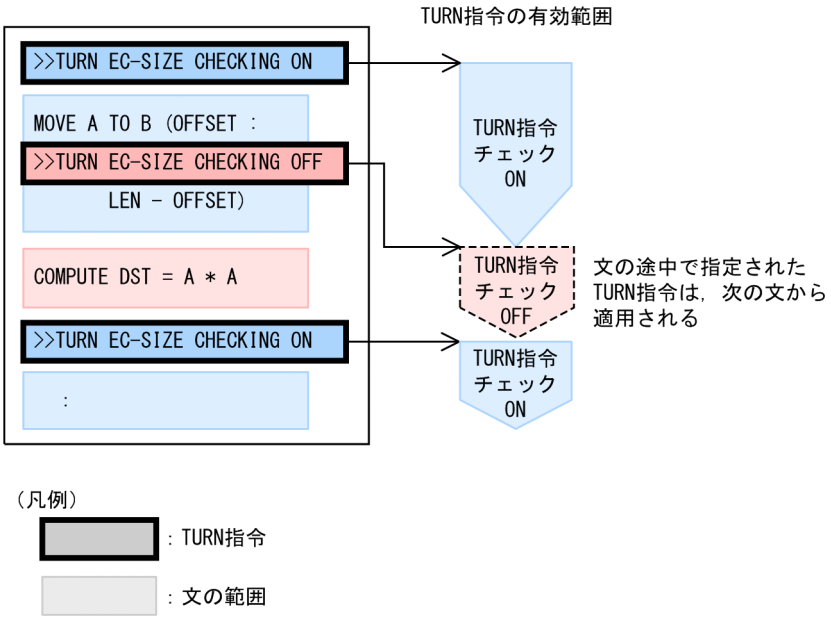
表 21–4　TURN 指令と対象の例外名（EC-I-O 例外名）

TURN 指令に指定する例外名のレベル	ファイル名の指定	対象となる EC-I-O 例外名
レベル 2	あり	指定したファイル名に対応するすべてのレベル 3 例外名
	なし	すべてのファイルに対応するすべてのレベル 3 例外名
レベル 3	あり	指定したファイル名に対応するレベル 3 例外名
	なし	すべてのファイルに対応する指定したレベル 3 例外名

21.3.2　TURN 指令の有効範囲

TURN 指令は，その TURN 指令を記述した行以降で開始される文に対して有効となります。TURN 指令を文の途中に記述した場合，TURN 指令はその文には適用されないで，次の文から適用されます。このため，一つの文の中で TURN 指令のチェックの ON／OFF を切り替えられません。

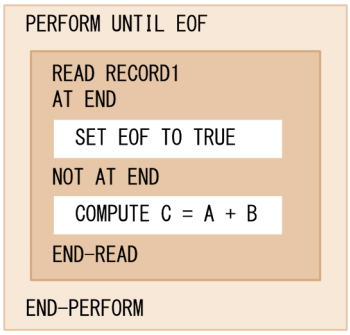
TURN 指令を文の途中に記述した場合の例を，次に示します。



文の範囲は，先頭の語である文の名前から，明示的または暗黙的に文が終了するまでとなります。文の開始から終了までの間の構文に現れたすべての手続き文も，文の範囲に含まれます。このとき，入れ子構造になっている文の一部を共通例外処理の対象としたい場合は，対象の文の直前で TURN 指令のチェックを ON にします。

次の例では，PERFORM 文，READ 文，SET 文，COMPUTE 文は，それぞれ文です。このとき，READ 文は，SET 文および COMPUTE 文を含み，PERFORM 文はその READ 文全体を含みます。

図 21-4　文の範囲



上記の例で，COMPUTE 文の EC-SIZE チェックを有効にしたい場合，次のように記述します。

（COMPUTE 文の EC-SIZE チェックを有効にする例）

```
READ RECORD1
AT END
  SET EOF TO TRUE
NOT AT END
  >>TURN EC-SIZE CHECKING ON
  COMPUTE C = A + B
END-READ
```

- READ 文から見た場合，TURN 指令が文の途中に記述されています。そのため，READ 文に対しては，TURN 指令が適用されません。
- COMPUTE 文から見た場合，TURN 指令が文の外に記述されています。そのため，COMPUTE 文に対しては，TURN 指令が適用されます。

また，SET 文および COMPUTE 文の EC-SIZE チェックを有効にしたい場合，次のように記述します。

（SET 文および COMPUTE 文の EC-SIZE チェックを有効にする例）

```
READ RECORD1
>>TURN EC-SIZE CHECKING ON
AT END
  SET EOF TO TRUE
NOT AT END
  COMPUTE C = A + B
END-READ
```

- READ 文から見た場合，TURN 指令が文の途中に記述されています。そのため，READ 文に対しては，TURN 指令が適用されません。
- SET 文および COMPUTE 文から見た場合，TURN 指令が文の外に記述されています。そのため，SET 文および COMPUTE 文に対しては，TURN 指令が適用されます。

21.3.3　例外チェックが無効な場合の動作

(1)　手続き文の実行中にエラーが発生し，例外を検出した場合

手続き文でエラーが発生して例外を検出した際に，該当する例外チェックが無効な場合は，検出された例外，コンパイラオプションの指定，および例外が検出された文に対する無条件文の指定によって，動作が異なります。

共通例外処理で取り扱える例外名について，例外チェックが無効な場合の動作を，次に示します。

表 21-5　エラー発生後に検出した例外に対して例外チェックが無効な場合の動作

例外名	例外チェックが無効な場合の動作
EC-ALL	以下のレベル 3 の例外が発生した場合，おのおのに対応する動作となる。

例外名	例外チェックが無効な場合の動作
EC-ARGUMENT	EC-ARGUMENT に関連するレベル 3 の例外が発生した場合，おのおのに対応する動作となる。
EC-ARGUMENT-FUNCTION	• 組み込み関数が CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数で文字コード変換の例外が発生した場合 ・-FunctionECSup,CodeConvErr オプションなしのとき 実行時エラーで異常終了。 ・-FunctionECSup,CodeConvErr オプションありのとき エラーとならない。 • 組み込み関数が CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数以外の場合 実行時エラーで異常終了。
EC-ARGUMENT-IMP	• 組み込み関数が CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数で文字コード変換の例外が発生した場合 ・-FunctionECSup,CodeConvErr オプションなしのとき 実行時エラーで異常終了。 ・-FunctionECSup,CodeConvErr オプションありのとき エラーとならない。 • 組み込み関数が CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数以外の場合 実行時エラーで異常終了。
EC-BOUND	EC-BOUND に関連するレベル 3 の例外が発生した場合，おのおのに対応する動作となる。
EC-BOUND-ODO	エラーとならないが，実行結果は保証しない。※1
EC-BOUND-REF-MOD	• -DebugCompati オプションなしの場合 エラーとならないが，実行結果は保証しない。※1 • -DebugCompati オプションありの場合 実行時エラーで異常終了。
EC-BOUND-SUBSCRIPT	• -DebugCompati または-DebugRange オプションなしの場合 エラーとならないが，実行結果は保証しない。※1 • -DebugCompati または-DebugRange オプションありの場合 実行時エラーで異常終了。
EC-DATA	EC-DATA に関連するレベル 3 の例外が発生した場合，おのおのに対応する動作となる。
EC-DATA-PTR-NULL	シグナル発生による異常終了。
EC-FLOW	EC-FLOW に関連するレベル 3 の例外が発生した場合，おのおのに対応する動作となる。
EC-FLOW-GLOBAL-EXIT	エラーとならない。

例外名	例外チェックが無効な場合の動作
EC-FLOW-GLOBAL-GOBACK	エラーとならない。
EC-FLOW-IMP	実行時エラーで異常終了。
EC-FLOW-RELEASE	実行時エラーで異常終了。
EC-FLOW-RETURN	実行時エラーで異常終了。
EC-FLOW-USE	実行時エラーで異常終了。
EC-I-O	EC-I-O に関連するレベル 3 の例外が発生した場合，おのおのに対応する動作となる。
EC-I-O-AT-END	• -Compati85,IoStatus オプションなしの場合 エラーとならない。 • -Compati85,IoStatus オプションありの場合 AT END 指定がない場合は実行時エラーで異常終了。
EC-I-O-EOP	エラーとならない。
EC-I-O-EOP-OVERFLOW	エラーとならない。
EC-I-O-IMP	実行時エラーで異常終了。
EC-I-O-INVALID-KEY	• -Compati85,IoStatus オプションなしの場合 エラーとならない。 • -Compati85,IoStatus オプションありの場合 INVALID KEY 指定がない場合は実行時エラーで異常終了。
EC-I-O-LINAGE	• -Compati85,Linage オプションなしの場合 実行時エラーで異常終了。 • -Compati85,Linage オプションありの場合 エラーとならない。
EC-I-O-LOGIC-ERROR	実行時エラーで異常終了。
EC-I-O-PERMANENT-ERROR	実行時エラーで異常終了。
EC-OO	EC-OO に関連するレベル 3 の例外が発生した場合，おのおのに対応する動作となる。
EC-OO-CONFORMANCE	実行時エラーで異常終了。
EC-OO-EXCEPTION	実行時エラーで異常終了。
EC-OO-IMP	実行時エラーで異常終了。
EC-OO-METHOD	実行時エラーで異常終了。
EC-OO-NULL	実行時エラーで異常終了。
EC-OO-RESOURCE	実行時エラーで異常終了。
EC-OO-UNIVERSAL	実行時エラーで異常終了。

例外名	例外チェックが無効な場合の動作
EC-OVERFLOW	EC-OVERFLOW に関連するレベル 3 の例外が発生した場合，おのおのに対応する動作となる。
EC-OVERFLOW-STRING	ON OVERFLOW 指定がない場合でもエラーとはならない。例外が検出される時点までの処理は実行されており，その時点までの実行結果は保証する。
EC-OVERFLOW-UNSTRING	ON OVERFLOW 指定がない場合でもエラーとはならない。例外が検出される時点までの処理は実行されており，その時点までの実行結果は保証する。
EC-PROGRAM	EC-PROGRAM に関連するレベル 3 の例外が発生した場合，おのおのに対応する動作となる。
EC-PROGRAM-ARG-MISMATCH	コンパイラオプションの指定状況や CALL 文の書き方によって動作が異なる。※2
EC-PROGRAM-CANCEL-ACTIVE	実行時エラーで異常終了。
EC-PROGRAM-IMP	コンパイラオプションの指定状況や CALL 文の書き方によって動作が異なる。※2
EC-PROGRAM-NOT-FOUND	コンパイラオプションの指定状況や CALL 文の書き方によって動作が異なる。※2
EC-PROGRAM-RECURSIVE-CALL	コンパイラオプションの指定状況や CALL 文の書き方によって動作が異なる。※2
EC-PROGRAM-RESOURCES	コンパイラオプションの指定状況や CALL 文の書き方によって動作が異なる。※2
EC-RAISING	EC-RAISING に関連するレベル 3 の例外が発生した場合，おのおのに対応する動作となる。
EC-RAISING-NOT-SPECIFIED	実行時エラーで異常終了。
EC-RANGE	EC-RANGE に関連するレベル 3 の例外が発生した場合，おのおのに対応する動作となる。
EC-RANGE-INSPECT-SIZE	実行時エラーで異常終了。
EC-RANGE-INVALID	エラーとならないが，実行結果は保証しない。
EC-RANGE-PERFORM-VARYING	エラーとならないが，実行結果は保証しない。
EC-RANGE-SEARCH-INDEX	エラーとならないが，実行結果は保証しない。
EC-RANGE-SEARCH-NO-MATCH	エラーとならないが，実行結果は保証しない。
EC-SIZE	EC-SIZE に関連するレベル 3 の例外が発生した場合，おのおのに対応する動作となる。
EC-SIZE-EXPONENTIATION	コンパイラオプションの指定状況や COMPUTE 文の書き方によって動作が異なる。※3
EC-SIZE-OVERFLOW	エラーとならないが，ON SIZE ERROR, NOT ON SIZE ERROR 指定がない場合の実行結果はけたあふれが発生している。なお，べき乗演算につ

例外名	例外チェックが無効な場合の動作
	いてはコンパイラオプションの指定状況や COMPUTE 文の書き方によって動作が異なる。※3
EC-SIZE-TRUNCATION	エラーとならないが，ON SIZE ERROR, NOT ON SIZE ERROR 指定がない場合の実行結果はけたあふれが発生している。
EC-SIZE-UNDERFLOW	エラーとならないが，ON SIZE ERROR, NOT ON SIZE ERROR 指定がない場合の実行結果はけたあふれが発生している。なお，べき乗演算についてはコンパイラオプションの指定状況や COMPUTE 文の書き方によって動作が異なる。※3
EC-SIZE-ZERO-DIVIDE	ON SIZE ERROR，または NOT ON SIZE ERROR 指定がない場合は実行時エラーで異常終了。
EC-SORT-MERGE	EC-SORT-MERGE に関連するレベル 3 の例外が発生した場合，おのおのに対応する動作となる。
EC-SORT-MERGE-IMP	実行時エラーで異常終了。
EC-SORT-MERGE-RELEASE	実行時エラーで異常終了。
EC-SORT-MERGE-RETURN	実行時エラーで異常終了。
EC-USER	EC-USER に関連するレベル 3 の例外が発生した場合，おのおのに対応する動作となる。
EC-USER-（ユーザ定義例外名）	エラーとならない。

注※1

メモリの状態によっては，シグナル発生による異常終了となる場合があります。

注※2

動作の詳細については，「表 21-6　EC-PROGRAM とコンパイラオプションの組み合わせによる動作の相違」を参照してください。

注※3

動作の詳細については，「表 21-7　べき乗演算とコンパイラオプションの組み合わせによる動作の相違」を参照してください。

表 21-6　EC-PROGRAM とコンパイラオプションの組み合わせによる動作の相違

例外名	-Compati85, Call 指定の有無	CALL 文の ON EXCEPTION または ON OVERFLOW の指定	
		あり	なし
EC-PROGRAM-ARG-MISMATCH	あり	• -DebugCompati ありの場合 実行時エラーで異常終了。 • -DebugCompati なしの場合 エラーとはならないが，実行結果を保証しない場合がある（引数や返却項目の参照／設定の状態によっては，シグナル発生による異常終了や実行結果不正となる場合がある）。	
	なし	• -DebugCompati ありの場合 指定された無条件文を実行。 • -DebugCompati なしの場合	• -DebugCompati ありの場合 実行時エラーで異常終了。 • -DebugCompati なしの場合

例外名	-Compati85,Call 指定の有無	CALL 文の ON EXCEPTION または ON OVERFLOW の指定	
		あり	なし
		エラーとはならないが，実行結果を保証しない場合がある（引数や返却項目の参照／設定の状態によっては，シグナル発生による異常終了や実行結果不正となる場合がある）。	エラーとはならないが，実行結果を保証しない場合がある（引数や返却項目の参照／設定の状態によっては，シグナル発生による異常終了や実行結果不正となる場合がある）。
EC-PROGRAM-IMP	あり	• 実行可能ファイル呼び出し，または共用ライブラリロード時に例外を検出した場合，指定された無条件文を実行。 • 上記以外の場合，実行時エラーで異常終了。	実行時エラーで異常終了。
	なし	指定された無条件文を実行。	
EC-PROGRAM-NOT-FOUND	あり	指定された無条件文を実行。	実行時エラーで異常終了。
	なし		
EC-PROGRAM-RECURSIVE-CALL	あり	実行時エラーで異常終了。	
	なし	指定された無条件文を実行。	実行時エラーで異常終了。
EC-PROGRAM-RESOURCES	あり	実行時エラーで異常終了。	
	なし	指定された無条件文を実行。	実行時エラーで異常終了。

表 21-7　べき乗演算とコンパイラオプションの組み合わせによる動作の相違

種別※1	例外名		-Compati85,Power 指定の有無	COMPUTE 文の ON SIZE ERROR の指定			
				あり		なし	
				NOT ON SIZE ERROR		NOT ON SIZE ERROR	
				あり	なし	あり	なし
整数べき乗	EC-SIZE-EXPONENTIATION	底 ：0 指数：0	あり	NOT ON SIZE を実行。	エラーとならない。	NOT ON SIZE を実行。	エラーとならない。
			なし	ON SIZE を実行。		エラーとならない。	
		底 ：0 指数：負	あり	ON SIZE を実行。		エラーとならない。	実行時エラーで異常終了。
			なし				
浮動小数点べ	EC-SIZE-EXPONENTIATION	底 ：0 指数：0	あり	NOT ON SIZE を実行。	エラーとならない。	NOT ON SIZE を実行。	エラーとならない。
			なし	ON SIZE を実行。		エラーとならない。	
		底 ：0 指数：負	あり※2	NOT ON SIZE を実行。	エラーとならない。	NOT ON SIZE を実行。	エラーとならない。

種別※1	例外名		-Compati 85,Power 指定の有無	COMPUTE 文の ON SIZE ERROR の指定			
				あり		なし	
				NOT ON SIZE ERROR		NOT ON SIZE ERROR	
				あり	なし	あり	なし
き乗			あり※3	ON SIZE を実行。		エラーとならない。	実行時エラーで異常終了。
			なし	ON SIZE を実行。		エラーとならない	実行時エラーで異常終了。
		底 ：負 指数：小数	あり	ON SIZE を実行。		エラーとならない。	実行時エラーで異常終了。
			なし				
	EC-SIZE-OVERFLOW		あり	NOT ON SIZE を実行。	エラーとならない。	NOT ON SIZE を実行。	エラーとならない。
			なし	ON SIZE を実行。		エラーとならない。	
	EC-SIZE-UNDERFLOW		あり	NOT ON SIZE を実行。	エラーとならない。	NOT ON SIZE を実行。	エラーとならない。
			なし	ON SIZE を実行。		エラーとならない。	

注※1

被べき数（底）が浮動小数点項目であるか，またはべき数（指数）が整数でない場合に演算の中間結果が浮動小数点形式となり，上表の浮動小数点べき乗演算となります。

これ以外は，演算の中間結果が整数形式となり，整数べき乗演算となります。

演算の中間結果の詳細については，「5.2.4　演算の中間結果」を参照してください。

注※2　AIX の場合。

注※3　Linux の場合。

(2)　伝播によって例外を検出した場合

伝播によって例外を検出した場合，CALL 文，INVOKE 文，および利用者定義関数の呼び出しで検出された例外名の例外チェックが無効なときは，例外の致命度によって，次に示す処理が実行されます。

表 21-8　伝播によって検出した例外に対して例外チェックが無効な場合の動作

致命度	動作
致命的な例外	「図 21-5　プログラム「CHILDPROGRAM」から致命的な例外の伝播を実行するが，プログラム「PARENTPROGRAM」の例外チェックが無効な場合の動作例」のように，制御遷移した文（CALL 文，INVOKE 文，および利用者定義関数の呼び出し）で，実行時エラーメッセージ（KCCC0402R-S）を出力し，実行単位が異常終了となる。 なお，例外を受け取れないプログラム※の場合は，「図 21-6　プログラム「CHILDPROGRAM」から致命的な例外の伝播を実行しようとするが，プログラム「PARENTPROGRAM」が例外を受け取

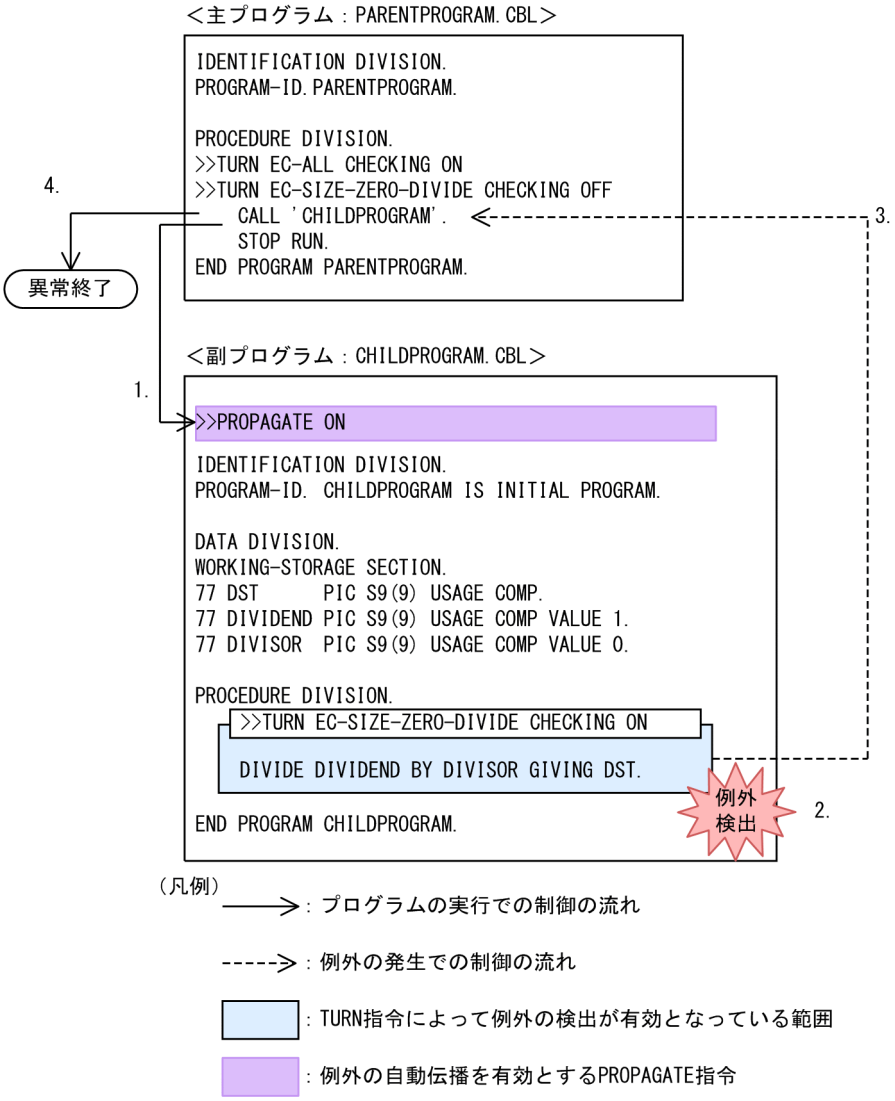
致命度	動作
	れない場合の動作例」のように，呼び出し先プログラム中の例外を検出した文で，実行時エラーメッセージ（KCCC0403R-S）を出力し，実行単位が異常終了となる。
非致命的な例外	「図 21-7　非致命的な例外のときは，例外の伝播は無視され，実行が継続される場合の動作例」のように，例外の伝播は無視され，実行が継続される。

注※

例外を受け取れないプログラムの詳細については，「21.5.3　例外を受け取れないプログラムに例外を伝播させた場合の動作」を参照してください。

例外の伝播によって検出された例外の詳細については，「表 21-15　例外の伝播によって検出された例外」を参照してください。

図 21-5　プログラム「CHILDPROGRAM」から致命的な例外の伝播を実行するが，プログラム「PARENTPROGRAM」の例外チェックが無効な場合の動作例



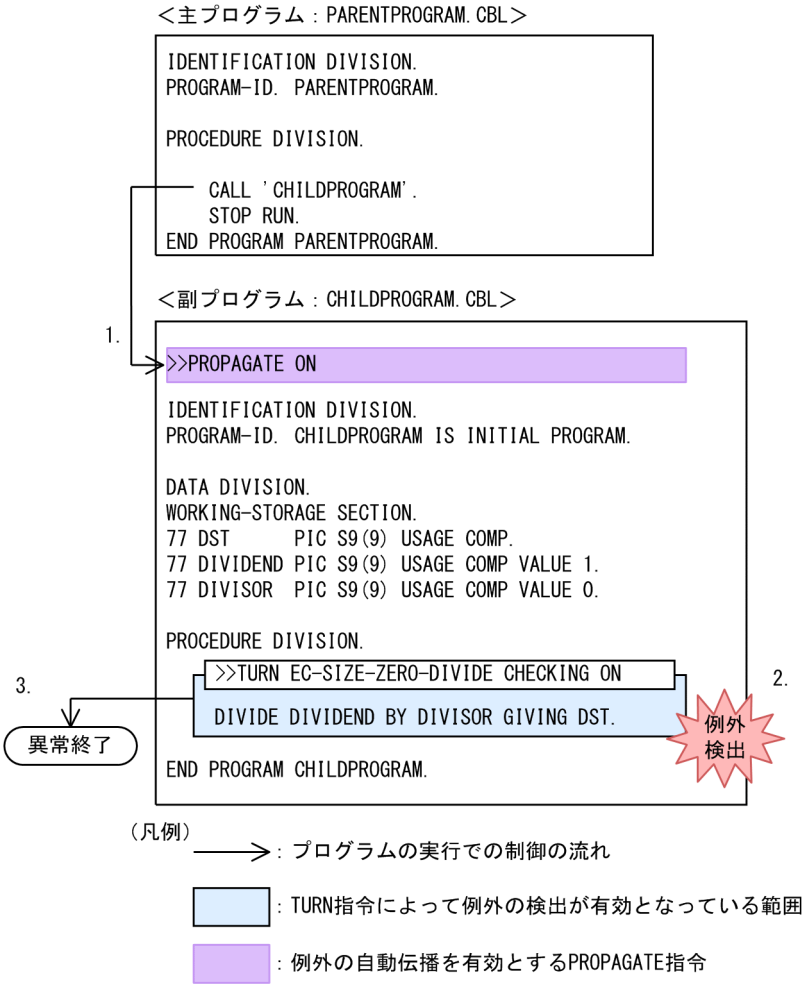
1. プログラム「PARENTPROGRAM」中の CALL 文で，プログラム「CHILDPROGRAM」を呼び出します。

2. プログラム「CHILDPROGRAM」の DIVIDE 文で，ゼロによる除算の例外が検出されます（例外名：EC-SIZE-ZERO-DIVIDE）。

3. PROPAGATE 指令が ON なので，プログラム「PARENTPROGRAM」の CALL 文に致命的な例外が伝播します。

4. プログラム「PARENTPROGRAM」中で，伝播された致命的な例外（例外名：EC-SIZE-ZERO-DIVIDE）に対する例外チェックが無効なので，CALL 文で実行時エラーメッセージ（KCCC0402R-S：「伝播により例外が検出されました。」）を出力し，プログラムが異常終了します。

図 21-6　プログラム「CHILDPROGRAM」から致命的な例外の伝播を実行しようとするが，プログラム「PARENTPROGRAM」が例外を受け取れない場合の動作例



1. プログラム「PARENTPROGRAM」中の CALL 文で，プログラム「CHILDPROGRAM」を呼び出します。

2. プログラム「CHILDPROGRAM」中の DIVIDE 文で，ゼロによる除算の例外が検出されます（例外名：EC-SIZE-ZERO-DIVIDE）。

3. PROPAGATE 指令が ON になっていますが，プログラム「PARENTPROGRAM」が例外を受け取れないプログラム※なので，致命的な例外が伝播できません。そのため，プログラム「CHILDPROGRAM」の例外を検出した文（DIVIDE 文）で実行時エラーメッセージ（KCCC0403R-S：「例外を伝播できません。」）を出力し，プログラムが異常終了します。

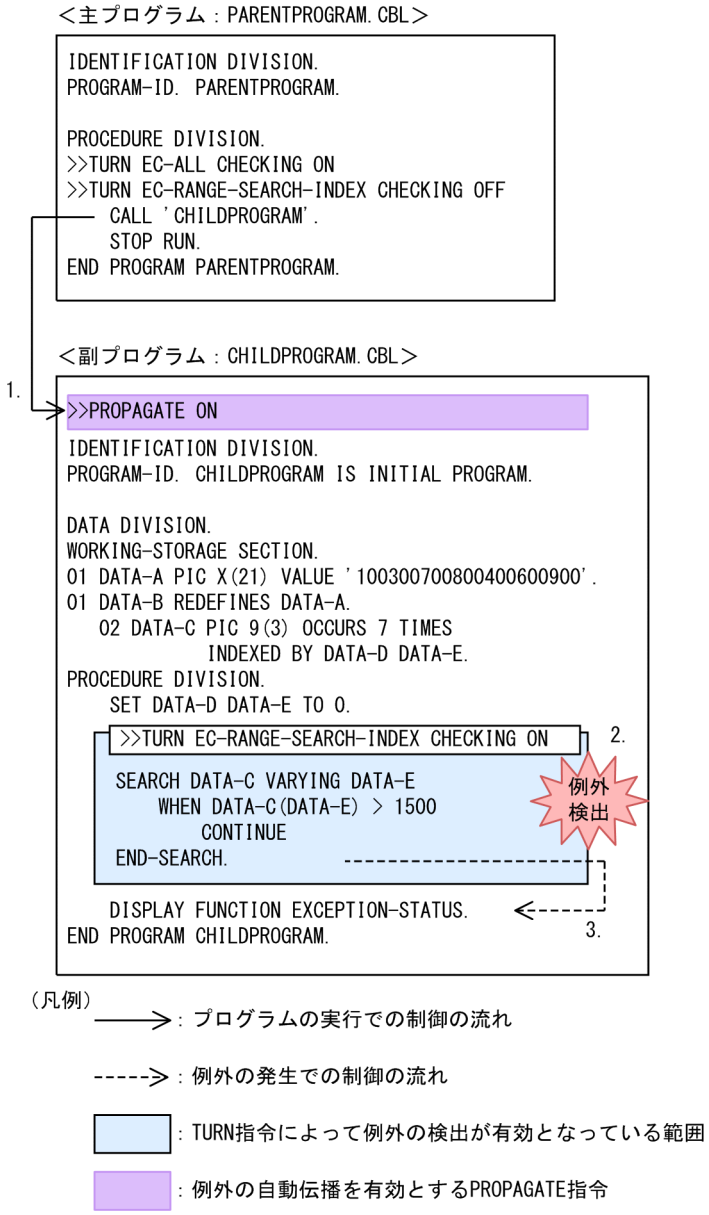
注※

この場合の例外を受け取れないプログラムとは，次の条件がすべて重なった場合を指します。

1. プログラム中に ON 指定のある PROPAGATE 指令が書かれていない。
2. プログラム中に TURN 指令が一つも書かれていない。
3. 宣言節中の USE 文に EXCEPTION OBJECT 指定が一つもない。
4. CALL 文に ON OVERFLOW 指定，ON EXCEPTION 指定，NOT ON EXCEPTION 指定がない。

例外を受け取れないプログラムの詳細については，「21.5.3　例外を受け取れないプログラムに例外を伝播させた場合の動作」を参照してください。

図 21-7　非致命的な例外のときは，例外の伝播は無視され，実行が継続される場合の動作例



1. プログラム「PARENTPROGRAM」中の CALL 文で，プログラム「CHILDPROGRAM」を呼び出します。
2. プログラム「CHILDPROGRAM」中の SEARCH 文で，指標名の範囲外による例外が検出されます（例外名：EC-RANGE-SEARCH-INDEX）。
3. PROPAGATE 指令が ON になっていますが，非致命的な例外のときは，例外の伝播は無視され，実行が継続されます。

21.4　共通例外の宣言手続き

USE 文に，引き起こされた例外に対する例外名，または例外オブジェクトを指定している場合，共通例外の宣言手続きが実行されます。

共通例外の宣言手続きは，USE 文に例外名，例外オブジェクトのクラス名，インタフェース名を指定して記述します。さらに，例外名 EC-I-O については，ファイル名を指定すると，特定のファイルに対する USE 文を記述できます。USE 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.53 USE 文」を参照してください。

21.4.1　実行される宣言手続き

実行される宣言手続きは，引き起こされた例外と，USE 文に指定した例外によって決定します。実行される宣言手続きの詳細については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.53　USE 文」を参照してください。

(1)　例外によって実行される宣言手続き

例外名に関連する場合

引き起こされた例外が例外名に関連する場合は，例外に対応する例外名，または上位レベルの例外名が指定された USE 文の宣言手続きが実行されます。

例外オブジェクトの場合

引き起こされた例外オブジェクトに適合するクラス名またはインタフェース名が指定された USE 文の宣言手続きが実行されます。

(2)　実行される宣言手続きの優先順位

引き起こされた例外に対して宣言手続きを複数記述した場合，レベル 3，レベル 2，レベル 1 の順に検索され，宣言手続きが一つだけ実行されます。

次の例では，引き起こされた例外の例外名 EC-SIZE-ZERO-DIVIDE に対して，実行される宣言手続きが EXCEPTIONHANDLER1（例外名 EC-SIZE に該当）と EXCEPTIONHANDLER2（例外名 EC-SIZE-ZERO-DIVIDE に該当）の 2 種類があります。この場合，レベル 2 の例外よりもレベル 3 の例外が優先されるため，EXCEPTIONHANDLER1 は実行されないで，EXCEPTIONHANDLER2 が実行されます。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE.
       >>TURN EC-SIZE CHECKING ON
       DATA DIVISION.
       WORKING-STORAGE  SECTION.
       01 A PIC 9.
       01 B PIC 9 VALUE 0.
       01 C PIC 9.
```

```
       PROCEDURE DIVISION.
       DECLARATIVES.
       EXCEPTION-HANDLER1 SECTION.
         USE AFTER EXCEPTION CONDITION EC-SIZE. *> 2.
           DISPLAY 'EC-SIZE'.
       EXCEPTION-HANDLER2 SECTION.
         USE AFTER EXCEPTION CONDITION EC-SIZE-ZERO-DIVIDE. *> 3.
           DISPLAY FUNCTION EXCEPTION-STATUS.
       END DECLARATIVES.

           DIVIDE A BY B GIVING C. *> 1.

       END PROGRAM SAMPLE.
```

1. DIVIDE 文で例外 EC-SIZE-ZERO-DIVIDE が引き起こされます。
2. EXCEPTION-HANDLER1 は，例外名 EC-SIZE に対応するため，1.の例外に該当します。しかし，EXCEPTION-HANDLER1 は実行されないで，より下位のレベル EC-SIZE-ZERO-DIVIDE の例外宣言手続きが優先されます。
3. EXCEPTION-HANDLER2 は，例外名 EC-SIZE-DIVIDE に対応するため，1.の例外に該当します。さらに，EXCEPTION-HANDLER1 の例外名 EC-SIZE より下位のレベルのため，EXCEPTION-HANDLER2 が優先となります。EXCEPTIONHANDLER2 が実行され，DISPLAY 文で「EC-SIZE-ZERO-DIVIDE」と表示されます。

なお，例外名が「EC-I-O-」ではじまる入出力例外は，ファイル名を指定する形式とファイル名を指定しない形式の 2 種類が指定できますが，両方の形式の例外宣言手続きを指定した場合，ファイル名を指定する形式の方が優先されます。

次の例では，FILE1 の READ 文で例外が引き起こされると，IO-EXCEPTION-HANDLER2 の方が実行されます。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE.

       >>TURN EC-I-O CHECKING ON
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       SELECT FILE1 ASSIGN TO 'FILENAME.TXT'.

       DATA DIVISION.
       FILE SECTION.
       FD FILE1.
       01 REC PIC X.
       WORKING-STORAGE SECTION.
       77 FILESTATUS PIC X(2).
       77 LOOP-EXIT-FLAG PIC 9 VALUE 0.

       PROCEDURE DIVISION.
       DECLARATIVES.
       IO-EXCEPTION-HANDLER1 SECTION.
           USE AFTER EXCEPTION CONDITION EC-I-O-AT-END. *> 2.
           DISPLAY 'GENERAL HANDLER'
```

```
        RESUME AT TERMINATION.
    IO-EXCEPTION-HANDLER2 SECTION.
        USE AFTER EXCEPTION CONDITION
                            EC-I-O-AT-END FILE FILE1. *> 3.
        DISPLAY 'SPECIAL HANDLER FOR FILE1'
        CLOSE FILE1
        RESUME AT TERMINATION.
    END DECLARATIVES.
        OPEN INPUT FILE1
        PERFORM UNTIL LOOP-EXIT-FLAG = 1
          READ FILE1 *> 1.
          IF  FILESTATUS NOT = '00' AND '10' THEN
            MOVE 1 TO LOOP-EXIT-FLAG
            DISPLAY 'READ FILESTATUS = ' FILESTATUS
          END-IF
        END-PERFORM.
    TERMINATION.
        STOP RUN.

    END PROGRAM SAMPLE.
```

1. READ 文で例外 EC-I-O-AT-END が引き起こされます。
2. IO-EXCEPTION-HANDLER1 は，例外名 EC-I-O-AT-END に対応するため，1.の例外に該当します。しかし，IO-EXCEPTION-HANDLER1 は実行されないで，ファイル名を指定した例外宣言手続きが優先されます。
3. EXCEPTION-HANDLER2 は，例外名 EC-I-O-AT-END に対応するため，1.の例外に該当します。さらに，ファイル名が指定されているため，IO-EXCEPTION-HANDLER1 より IO-EXCEPTION-HANDLER2 の方が優先となります。

   IO-EXCEPTION-HANDLER2 が実行され，DISPLAY 文で「SPECIAL HANDLER FOR FILE1」と表示したあと，FILE1 を閉じて手続き名 TERMINATION に復帰し，プログラムが終了します。

(3)　宣言手続き実行後の処理

宣言手続きからの復帰が行われない場合，例外の致命度によって，次の処理が実行されます。

致命的な例外の場合

宣言手続きの実行後，実行単位が終了します。ただし，例外名 EC-I-O に関連する例外の場合は，各文の一般規則に従って，実行が継続されます。

非致命的な例外の場合

宣言手続きを実行後，各文の一般規則に従って実行が継続されます。

宣言手続きから復帰した場合は，致命度に関係なく実行が継続されます。宣言手続きからの復帰については，「21.4.2　宣言手続きからの復帰」を参照してください。

(4) 注意事項

例外名 EC-FLOW-USE に対して実行される宣言手続きを，再帰的に実行するような処理にしないでください。このような処理にすると，無限ループが発生します。

21.4.2 宣言手続きからの復帰

宣言手続きの実行中に次の文を使用すると，宣言手続きの実行を中断して，制御を復帰できます。

(1) RESUME 文

RESUME 文は，宣言手続きから復帰するときに使用します。RESUME 文では，例外が引き起こされた次の文に復帰させる方法と，指定した手続き名へ復帰させる方法があります。

RESUME 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.37　RESUME 文」を参照してください。

RESUME 文を使用した宣言手続きからの復帰の例を，次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE.
       >>TURN EC-SIZE CHECKING ON

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 A PIC 9 VALUE 9.
       01 B PIC 9 VALUE 0.
       01 C PIC 9.

       PROCEDURE DIVISION.
       DECLARATIVES.
         USE-1 SECTION.
           USE AFTER EXCEPTION CONDITION EC-SIZE-ZERO-DIVIDE. *> 2.
             DISPLAY 'RESUME NEXT実行'.
             RESUME NEXT STATEMENT. *> 3.
         USE-2 SECTION.
           USE AFTER EXCEPTION CONDITION EC-SIZE-TRUNCATION. *> 5.
             DISPLAY 'RESUME 手続き名実行'.
             RESUME 復帰手続き. *> 6.
       END DECLARATIVES.

           DIVIDE A BY B GIVING C. *> 1.

           COMPUTE C = A * 2. *> 4.
           STOP RUN.

       復帰手続き. *> 7.
           DISPLAY '宣言手続きからの復帰'.
                 :
       END PROGRAM SAMPLE.
```

1. DIVIDE 文で，ゼロによる除算の例外 EC-SIZE-ZERO-DIVIDE が検出されます。
2. 例外名 EC-SIZE-ZERO-DIVIDE を指定した例外宣言手続きが実行されます。
3. RESUME 文によって宣言手続きから復帰します。RESUME 文に NEXT STATEMENT が指定されているため，例外が引き起こされた次の行（COMPUTE 文）に制御が移ります。
4. COMPUTE 文で，算術けたあふれの例外 EC-SIZE-TRUNCATION が検出されます。
5. 例外名 EC-SIZE-TRUNCATION を指定した例外宣言手続きが実行されます。
6. RESUME 文によって宣言手続きから復帰します。RESUME 文に手続き名が指定されているため，「復帰手続き」に制御が移ります。
7. 手続きが実行され，DISPLAY 文で「宣言手続きからの復帰」と表示されます。

(2) GOBACK 文，EXIT 文

宣言手続きの実行中に，GOBACK 文，または EXIT 文を実行すると，暗黙的に宣言手続きからの復帰が行われます。

GOBACK 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.22　GOBACK 文」を参照してください。

EXIT 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.20　EXIT 文」を参照してください。

GOBACK 文を使用した宣言手続きからの復帰の例を，次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.

       PROCEDURE DIVISION.
           CALL 'SAMPLE2'. *> 1.
       END PROGRAM SAMPLE1.


       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.

       >>TURN EC-SIZE CHECKING ON
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 A PIC 9 VALUE 9.
       01 B PIC 9 VALUE 0.
       01 C PIC 9.
       PROCEDURE DIVISION.
       DECLARATIVES.
         USE-1 SECTION.
           USE AFTER EXCEPTION CONDITION EC-ALL. *> 3.
             DISPLAY 'GOBACK実行'.
             GOBACK. *> 4.
       END DECLARATIVES.
                :
           DIVIDE A BY B GIVING C. *> 2.
```

```
        EXIT PROGRAM.
    END PROGRAM SAMPLE2.
```

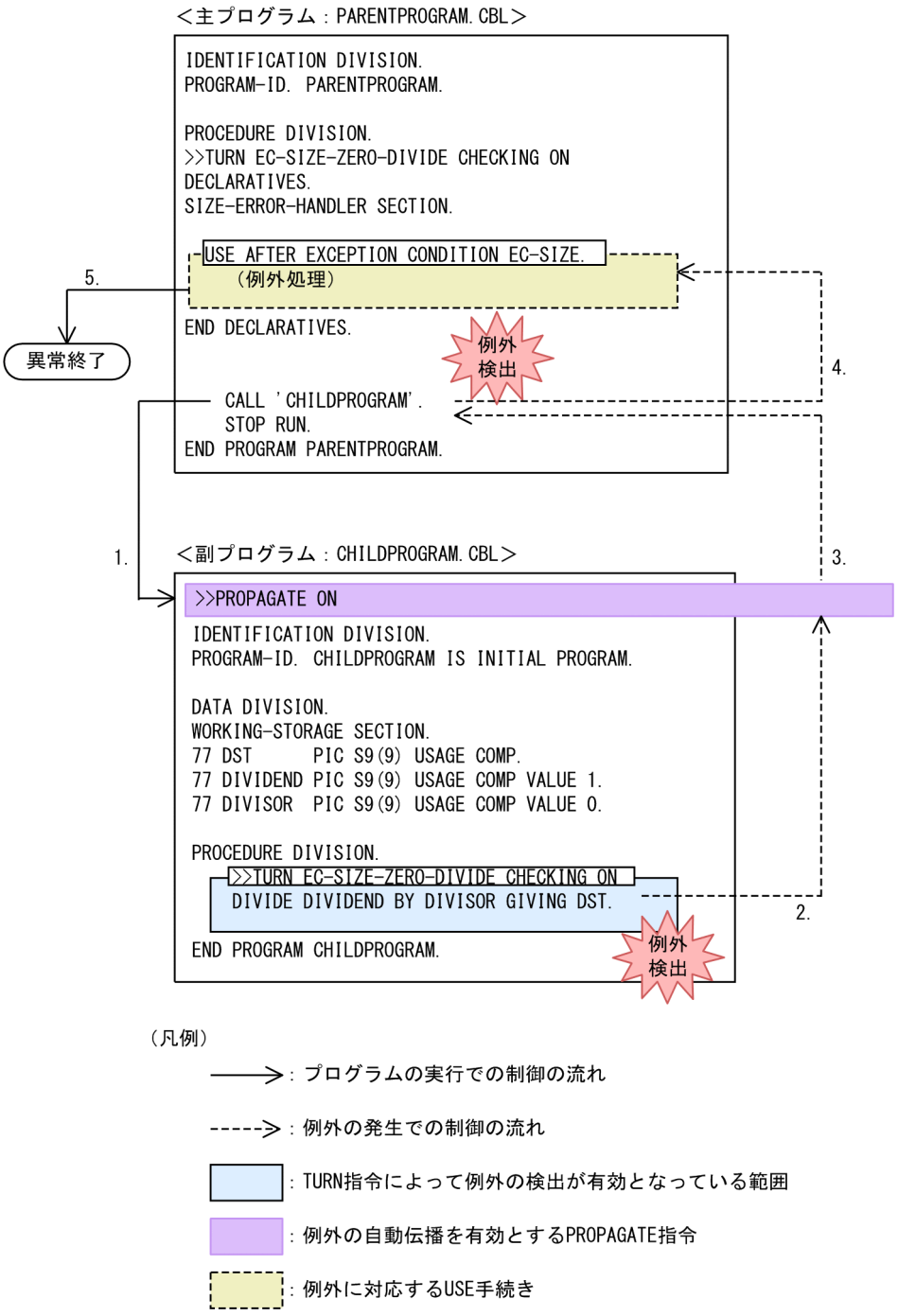
1. プログラム「SAMPLE1」中の CALL 文で，プログラム「SAMPLE2」を呼び出します。
2. プログラム「SAMPLE2」中の DIVIDE 文で，例外が検出されます。
3. 例外宣言手続きが実行されます。
4. GOBACK 文によって，宣言手続きから呼び出し元のプログラム「SAMPLE1」の CALL 文の次の行へ復帰します。

21.5　例外の伝播

呼び出し先プログラムで検出された例外情報を，呼び出し元プログラムに伝える（以下，伝播という）ことができます。

あるプログラムで例外が引き起こされ，該当する宣言手続きがない場合，呼び出し元プログラムに例外を伝播できます。

例外の伝播の例を，次に示します。



1. プログラム「PARENTPROGRAM」中の CALL 文で，プログラム「CHILDPROGRAM」を呼び出します。
2. プログラム「CHILDPROGRAM」中の DIVIDE 文で，ゼロによる除算の例外が検出されます（例外名：EC-SIZE-ZERO-DIVIDE)。
3. PROPAGATE 指令が ON なので，呼び出し元の CALL 文に例外が伝播します。
4. 例外の伝播によって CALL 文で例外が検出され（例外名：EC-SIZE-ZERO-DIVIDE)，例外が引き起こされます。これによって，例外名 EC-SIZE を指定した USE 手続きへ制御が移ります。
5. EC-SIZE-ZERO-DIVIDE は致命的な例外のため，例外宣言節の実行後，プログラムが異常終了します。

例外を伝播させる方法には，次の 2 種類があります。

- PROPAGATE 指令によって，例外を自動伝播させる
- GOBACK 文または EXIT 文の RAISING 指定を使用して，明示的に呼び出し元へ例外を伝播させる

21.5.1 PROPAGATE 指令による例外の自動伝播

PROPAGATE 指令を ON にすると，引き起こされた例外を自動的に呼び出し元プログラムへ伝播させられます。

PROPAGATE 指令については，マニュアル「COBOL2002　言語　標準仕様編」「3.3.13 PROPAGATE 指令」を参照してください。

PROPAGATE 指令を使用した例外の伝播の例を，次に示します。

＜主プログラム：SAMPLE1.CBL ＞

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
          :
       PROCEDURE DIVISION.
       DECLARATIVES.
       ERROR-HANDLER SECTION.
         USE  AFTER EXCEPTION CONDITION EC-BOUND-SUBSCRIPT. *> 4.
            :
       END DECLARATIVES.
       >>TURN EC-BOUND-SUBSCRIPT CHECKING ON

           CALL 'SAMPLE2'. *> 1.

       END PROGRAM SAMPLE1.
```

＜副プログラム：SAMPLE2.CBL ＞

```
       >>PROPAGATE ON *> 3.
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
          :
       DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
01 A.
  02 AA PIC 9 OCCURS 10 INDEXED BY B.
01 C PIC 9.
   :
PROCEDURE DIVISION.
    SET B DOWN BY 1.
>>TURN EC-BOUND-SUBSCRIPT CHECKING ON
    MOVE AA ( B ) TO C. *> 2.
END PROGRAM SAMPLE2.
```

1. プログラム「SAMPLE1」中の CALL 文で，プログラム「SAMPLE2」を呼び出します。
2. プログラム「SAMPLE2」中の MOVE 文で，例外が検出されます（例外名：EC-BOUND-SUBSCRIPT）。
3. PROPAGATE 指令が ON なので，呼び出し元の CALL 文に例外が伝播します。
4. 例外の伝播によって CALL 文で例外が検出され，例外が引き起こされます。これによって，例外名 EC-BOUND-SUBSCRIPT を指定した USE 手続きへ制御が移ります。

注意事項

- PROPAGATE ON が指定されたプログラムを COBOL 主プログラムとした場合，例外が引き起こされても例外は伝播されないで，COBOL プログラムが正常終了します。
- 自動伝播される例外は，例外の致命度が致命的なものだけです。致命度が非致命的な例外の場合，自動伝播されないで例外を引き起こした文の次の文が実行されます。

21.5.2 EXIT 文，GOBACK 文の RAISING 指定による例外の伝播

宣言手続き内で，GOBACK 文または EXIT 文の RAISING 指定に伝播する例外を指定すると，指定した例外を呼び出し元のプログラムに伝播できます。RAISING 指定には，例外名，オブジェクト参照一意名，または LAST 指定します。

RAISING 指定については，「COBOL2002　言語　標準仕様編」「10.8.20　EXIT 文」を参照してください。

EXIT PROGRAM 文の RAISING 指定を使用した例外の伝播の例を，次に示します。

＜主プログラム：SAMPLE1.CBL ＞

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE1.
    :
PROCEDURE DIVISION.
DECLARATIVES.
ERROR-HANDLER SECTION.
  USE AFTER EXCEPTION CONDITION EC-BOUND-SUBSCRIPT. *> 3.
     :
END DECLARATIVES.
```

```
>>TURN EC-BOUND-SUBSCRIPT CHECKING ON
  CALL 'SAMPLE2'. *> 1.

END PROGRAM SAMPLE1.
```

＜副プログラム：SAMPLE2.CBL ＞

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE2.
   :
PROCEDURE DIVISION.

    EXIT PROGRAM RAISING EXCEPTION EC-BOUND-SUBSCRIPT. *> 2.

END PROGRAM SAMPLE2.
```

1. プログラム「SAMPLE1」中の CALL 文で，プログラム「SAMPLE2」を呼び出します。
2. プログラム「SAMPLE2」中の EXIT PROGRAM 文の RAISING 指定に，伝播させる例外名 EC-BOUND-SUBSCRIPT が指定されているため，呼び出し元に例外が伝播します。
3. 例外の伝播によって CALL 文で例外が検出され，例外が引き起こされます。これによって，例外名 EC-BOUND-SUBSCRIPT を指定した USE 手続きへ制御が移ります。

(1) RAISING LAST 指定

EXIT 文または GOBACK 文の RAISING 指定に LAST を指定すると，実行単位で最後に引き起こされた例外を伝播できます。RAISING LAST 指定は，宣言手続きの実行中だけで使用できます。

なお，何も例外が引き起こされていない状態では，RAISING LAST 指定は無視されます。

RAISING LAST 指定の例を，次に示します。

＜主プログラム：SAMPLE1.CBL ＞

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE1.

PROCEDURE DIVISION.
DECLARATIVES.
EXCEPTION-HANDLER SECTION.
    USE AFTER EXCEPTION CONDITION EC-SIZE-ZERO-DIVIDE. *> 4.
    DISPLAY FUNCTION EXCEPTION-STATUS.
END DECLARATIVES.
>>TURN EC-SIZE-ZERO-DIVIDE CHECKING ON
    CALL 'SAMPLE2'. *> 1.
    STOP RUN.
END PROGRAM SAMPLE1.
```

＜副プログラム：SAMPLE2.CBL ＞

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE2.
       :
DATA DIVISION.
```

```
       WORKING-STORAGE SECTION.
       01 A-REC PIC S9(9) USAGE COMP VALUE 0.
       01 B-REC PIC S9(9) USAGE COMP VALUE 0.
       01 C-REC PIC S9(9) USAGE COMP VALUE 0.
              :
       PROCEDURE DIVISION.
       DECLARATIVES.
       EXCEPTION-HANDLER SECTION.
           USE AFTER EXCEPTION CONDITION EC-SIZE-ZERO-DIVIDE.
           EXIT PROGRAM RAISING LAST EXCEPTION. *> 3.
       END DECLARATIVES.
       >>TURN EC-SIZE-ZERO-DIVIDE CHECKING ON
           COMPUTE A-REC = B-REC / C-REC. *> 2.
       END PROGRAM SAMPLE2.
```

1. プログラム「SAMPLE1」中の CALL 文で，プログラム「SAMPLE2」を呼び出します。
2. プログラム「SAMPLE2」中の COMPUTE 文で，例外 EC-SIZE-ZERO-DIVIDE が引き起こされ，例外宣言手続きが実行されます。
3. EXIT PROGRAM 文の RAISING LAST 指定によって，最後に引き起こされた例外が呼び出し元のプログラムに伝播します。
4. 例外の伝播によって CALL 文で例外が検出され，例外が引き起こされます。これによって，例外名 EC-SIZE-ZERO-DIVIDE を指定した USE 手続きへ制御が移ります。

(2) 手続き部見出しの RAISING 指定

EXIT 文または GOBACK 文の RAISING 指定を使って，利用者定義例外，または例外オブジェクトを伝播する場合，手続き部見出しの RAISING 指定に，利用者定義例外名，またはオブジェクトのクラス名もしくはインタフェース名を指定する必要があります。指定していない場合，RAISING に例外名またはオブジェクト参照データを指定したときはコンパイルエラーとなり，RAISING LAST 指定のときは例外名 EC-RAISING-NOT-SPECIFIED または EC-OO-EXCEPTION が伝播されます。

手続き部見出しの RAISING 指定の例を，次に示します。

＜主プログラム：SAMPLE1.CBL ＞

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.

       PROCEDURE DIVISION.
       DECLARATIVES.
       EXCEPTION-HANDLER SECTION.
           USE AFTER EXCEPTION CONDITION EC-USER-XXX.
           DISPLAY FUNCTION EXCEPTION-STATUS.
       END DECLARATIVES.
       >>TURN EC-USER-XXX CHECKING ON
           CALL 'SAMPLE2'.
           STOP RUN.
       END PROGRAM SAMPLE1.
```

＜副プログラム：SAMPLE2.CBL ＞

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
             :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 A-REC PIC S9(9) USAGE COMP VALUE 0.
       01 B-REC PIC S9(9) USAGE COMP VALUE 0.
       01 C-REC PIC S9(9) USAGE COMP VALUE 0.
             :
       PROCEDURE DIVISION RAISING EC-USER-XXX.
       DECLARATIVES.
       ERROR-HANDLER SECTION.
         USE AFTER EXCEPTION CONDITION EC-SIZE-ZERO-DIVIDE.
           EXIT PROGRAM RAISING EXCEPTION EC-USER-XXX
       END DECLARATIVES.
       >>TURN EC-SIZE-ZERO-DIVIDE CHECKING ON
            :
           COMPUTE A-REC = B-REC / C-REC.
            :
       END PROGRAM SAMPLE2.
```

(3)　RAISING 指定による例外オブジェクトの伝播

RAISING 指定を使用して例外オブジェクトを伝播する場合，次の規則が適用されます。詳細は，マニュアル「COBOL2002　言語　標準仕様編」「10.5.11(1)　例外」の例外オブジェクトの説明を参照してください。

EXIT 文および GOBACK 文を使って例外オブジェクトを伝播する場合，呼び出し元プログラムおよび呼び出し先プログラムの状態によって，次に示す処理が実行されます。

呼び先し先のプログラムの状態	呼び出し元のプログラムの状態			
	該当する USE 宣言手続きあり	該当する USE 宣言手続きなし		
		PROPAGATE 指令 ON		PROPAGATE 指令 OFF
		手続き部見出しの RAISING 指定あり	手続き部見出しの RAISING 指定なし	
手続き部見出しの RAISING 指定あり	例外オブジェクトを伝播する。※1	例外オブジェクトを伝播する。※2	例外オブジェクトを伝播する。※3	EC-OO-EXCEPTION を伝播する。
手続き部見出しの RAISING 指定なし	EC-OO-EXCEPTION を伝播する。			

注※1

伝播後，該当する宣言手続きを実行します。

注※2

伝播後，GOBACK RAISING LAST を実行します。

注※3

伝播後，GOBACK RAISING EXCEPTION EC-OO-EXCEPTION を実行します。

RAISING 指定による例外オブジェクトの伝播の例を，次に示します。

＜主プログラム：SAMPLE1.CBL ＞

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.

       PROCEDURE DIVISION.
       DECLARATIVES.
       EXCEPTION-HANDLER SECTION.
           USE AFTER EXCEPTION CONDITION EC-OO-EXCEPTION. *> 3.
           DISPLAY FUNCTION EXCEPTION-STATUS.
       END DECLARATIVES.
       >>TURN EC-OO-EXCEPTION CHECKING ON
           CALL 'SAMPLE2'. *> 1.
           STOP RUN.
```

＜副プログラム：SAMPLE2.CBL ＞

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       REPOSITORY.
       CLASS EXCEPT-CLASS.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 OBJ USAGE OBJECT REFERENCE EXCEPT-CLASS.

       PROCEDURE DIVISION RAISING EXCEPT-CLASS.
           INVOKE EXCEPT-CLASS 'NEW' RETURNING OBJ.
           GOBACK RAISING OBJ. *> 2.
       END PROGRAM SAMPLE2.


       IDENTIFICATION DIVISION.
       CLASS-ID. EXCEPT-CLASS INHERITS BASE.

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       REPOSITORY.
       CLASS BASE.
              :
       END CLASS EXCEPT-CLASS.
```

1. プログラム「SAMPLE1」中の CALL 文で，プログラム「SAMPLE2」を呼び出します。
2. プログラム「SAMPLE2」中の GOBACK 文の RAISING 指定によって，例外オブジェクトの伝播が発生します。

   このとき，SAMPLE2 の手続き部見出しの RAISING 指定には，クラス名が指定されていますが，主プログラム「SAMPLE1」に発生した例外に対応する USE 宣言手続きや PROPAGATE 指令がないため，GOBACK 文の処理は，GOBACK RAISING EXCEPTION EC-OO-EXCEPTION となります。

3. 例外の伝播によって CALL 文で例外 EC-OO-EXCEPTION が検出され，該当する USE 手続きへ制御が移ります。

21.5.3　例外を受け取れないプログラムに例外を伝播させた場合の動作

(1)　例外を受け取れないプログラム

CALL 文，INVOKE 文，および利用者定義関数によって呼び出されたプログラム（例外を送る側）で例外が発生し，呼び出し元プログラム（例外を受け取る側）へ例外の伝播を実行しようとしても，呼び出し元プログラムが例外を受け取れない場合，例外は伝播しません。

例外を受け取れる呼び出し元プログラム，例外を受け取れない呼び出し元プログラムの条件を次に示します。

条件			プログラム種別／指定					
			COBOL 85	COBOL2002				COBOL 以外の言語
				-Compati85,NoPropagate 指定なし※1		-Compati85,NoPropagate 指定あり※1		
				伝播抑止条件※2	左記以外	伝播抑止条件※2	左記以外	
例外を受け取る文	オブジェクト指向の INVOKE 文		－	×	○	×	×	－
	利用者定義関数を参照する文		－	×	○	×	×	－
	CALL 文	無条件指定なし※3	×	×	○	×	×	－
		無条件指定あり※3	×	○※4	○	×	×	－

（凡例）

○：例外を受け取れる

×：例外を受け取れない

－：該当しない

注※1

例外の伝播を抑止するコンパイラオプションについては，「32.5.12　他システムとの移行の設定」の「(2)　-Compati85 オプション」を参照してください。

注※2

次の条件が重なった場合，例外の伝播を抑止します。

1. プログラム中に ON 指定のある PROPAGATE 指令が書かれていない。
2. プログラム中に TURN 指令が一つも書かれていない。

ただし上記の 1.～2.では，入れ子のプログラムや複数の最外側のプログラムが書かれたソースで，該当するプログラムの前に ON 指定のある PROPAGATE 指令や有効な TURN 指令がある場合，それ以降に定義されたすべてのプログラムは，例外の伝播を抑止する対象にはなりません。

3. 宣言節中の USE 文に EXCEPTION OBJECT 指定が一つもない。

注※3

ON OVERFLOW 指定，ON EXCEPTION 指定，NOT ON EXCEPTION 指定

注※4

呼び出されたプログラム（例外を送る側）で例外が発生し，呼び出し元プログラム（例外を受け取る側）へ例外を伝播する場合，CALL 文の無条件文は実行されません。詳細については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.4(3)　一般規則」を参照してください。

(2)　呼び出し元プログラムが例外を受け取れない場合の動作

呼び出し元プログラムが例外を受け取れない場合，呼び出し先プログラムで例外が検出されたときの動作が異なります。

例外が検出されたときの動作の詳細については，「21.8.3　例外処理の動作」を参照してください。

例外を受け取れないプログラムへ例外の伝播を実行しようとした場合の例を次に示します。

＜主プログラム：SAMPLE1.CBL（COBOL85 で作成したプログラム）＞

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
            :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 PARAM USAGE COMP-1 VALUE +9.9E+01.
            :
       PROCEDURE DIVISION.
           CALL 'SAMPLE2' USING PARAM. *> 1.
       END PROGRAM SAMPLE1.
```

＜副プログラム：SAMPLE2.CBL（COBOL2002 で作成したプログラム）＞

```
       >>PROPAGATE ON
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
            :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ANS USAGE COMP-1.
       01 DATA-A PIC X(21) VALUE '100300700800400600900'.
       01 DATA-B REDEFINES DATA-A.
          02 DATA-C PIC 9(3) OCCURS 7 TIMES
                     INDEXED BY DATA-D DATA-E.
       LINKAGE SECTION.
       01 PARAM USAGE COMP-1.
            :
       PROCEDURE DIVISION USING PARAM.
```

```
        SET DATA-D DATA-E TO 0.
    >>TURN EC-RANGE-SEARCH-INDEX CHECKING ON
        SEARCH DATA-C VARYING DATA-E      *>2.
            WHEN  DATA-C(DATA-E) > 1500
                CONTINUE
        END-SEARCH.
    >>TURN EC-ARGUMENT-FUNCTION CHECKING ON
        COMPUTE ANS = FUNCTION ACOS ( PARAM ). *> 3.
        DISPLAY 'ANSWER  =' ANS.
    END PROGRAM SAMPLE2.
```

1. プログラム「SAMPLE1」中の CALL 文で，プログラム「SAMPLE2」を呼び出します。
2. PROPAGATE 指令が ON になっていますが，プログラム「SAMPLE2」中の SEARCH 文で検出した例外が非致命的な例外のため，例外は伝播されず次の文（COMPUTE 文）へ実行が継続されます。
3. COMPUTE 文で致命的な例外が検出され，かつ，PROPAGATE 指令が ON になっていますが，プログラム「SAMPLE1」は例外を受け取れないため，例外は伝播しません。この場合，実行時エラーメッセージ（KCCC0403R-S:「例外を伝播できません。」）を出力し，プログラム「SAMPLE2」が異常終了します。

21.6　明示的な例外の引き起こし

COBOL2002 では，明示的に例外を引き起こせます。明示的に例外を引き起こすと，ユーザプログラムが固有に定めたエラー状態を利用者定義例外として発生させたり，ある例外種別で実行される宣言手続き処理に対してユーザプログラムから明示的に例外を発生させて実行できます。これによって，宣言手続き処理を共通化できます。

共通例外処理で明示的に例外を引き起こすには，RAISE 文を使用します。RAISE 文にレベル 3 の例外名またはオブジェクト参照の一意名を指定して実行すると，指定した例外が引き起こされます。

RAISE 文に指定する例外が例外名の場合，該当する TURN 指令のチェックが ON でなくても，例外が引き起こされます。

また，RAISE 文に例外名 EC-I-O に関連する例外を指定して実行する場合，入出力状態は変更されません。

このシステムでは，ファイル名指定の例外宣言手続きは，ファイル名指定のない例外宣言手続きとみなします。

そのため，同じ名称の例外名を指定した例外宣言手続きが複数個定義されている場合は，最後に定義した例外宣言手続きが実行されます。

RAISE 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.32　RAISE 文」を参照してください。

RAISE 文に例外名を指定して実行した場合の例を，次に示します。

```
       IDENTIFICATION   DIVISION.
        PROGRAM-ID.      SAMPLE.

       >>TURN  EC-I-O  CHECKING  ON
       ENVIRONMENT      DIVISION.
        INPUT-OUTPUT     SECTION.
         FILE-CONTROL.
           SELECT  FILE1  ASSIGN  TO  'FILENAME.EXT'.

       DATA  DIVISION.
        FILE  SECTION.
           FD  FILE1.
            01  REC  PIC  X.
        WORKING-STORAGE  SECTION.
           77  FILESTATUS  PIC X(2)  VALUE  '00'.

       PROCEDURE        DIVISION.
        DECLARATIVES.
         IO-EXCEPTION-HANDLER1  SECTION.
           USE  AFTER  EXCEPTION  CONDITION  EC-I-O-AT-END FILE FILE1.  *> 2.
             DISPLAY  'GENERAL HANDLER '.
             CLOSE  FILE1.
             RESUME  AT  TERMINATION.
         IO-EXCEPTION-HANDLER2  SECTION.
           USE  AFTER  EXCEPTION  CONDITION  EC-I-O-AT-END.            *> 3.
```

```
        DISPLAY  'SPECIAL HANDLER FOR FILE1'.
        CLOSE  FILE1.
        RESUME  AT  TERMINATION.
    END DECLARATIVES.
       OPEN  INPUT  FILE1.
       RAISE EXCEPTION EC-I-O-AT-END.  *> 1.
       READ  FILE1.
    TERMINATION.
       IF FILESTATUS NOT = '00'
         DISPLAY 'RAISE FILESTATUS = ' FILESTATUS
       END-IF.
       STOP  RUN.
    END  PROGRAM     SAMPLE.
```

1. RAISE 文によって，例外 EC-I-O-AT-END が引き起こされます。
2. IO-EXCEPTION-HANDLER1 では，指定されたファイル名は無視されて，例外名 EC-I-O-AT-END に対応します。ただし，3.に例外名 EC-I-O-AT-END に対応する例外宣言手続きがあるため，2.の例外宣言手続きに処理は移りません。
3. IO-EXCEPTION-HANDLER2 では，例外名 EC-I-O-AT-END に対応します。2.の例外名 EC-I-O-AT-END と同じ名称であるため，あとに指定された 3.の例外名に対応する例外宣言手続きに処理が移り，DISPLAY 文および CLOSE 文が実行されます。

21.7　例外情報の参照

最新例外状態を参照すると，最後に発生した例外情報を参照できます。最新例外状態の参照には，組み込み関数や，EXCEPTION-OBJECT を使用します。

21.7.1　組み込み関数を使用した例外情報の参照

次の組み込み関数を使用すると，実行単位で最後に引き起こされた例外（最新例外状態）の詳細な情報を取得できます。

(1)　EXCEPTION-FILE 関数

最新例外状態に関連するファイル結合子の入出力状態の値，およびファイル名称を返します。関数の戻り値の長さは，64 バイトです。

EXCEPTION-FILE 関数の戻り値を，次に示します。

例外の種類	関数の戻り値の内容
入出力に関連する例外の場合※1	入出力状態とファイル名
入出力以外の例外の場合※2	00

注※1

入出力に関連する例外（例外名 EC-I-O に関連する例外）であっても，例外名 EC-I-O-LINAGE，EC-I-O-EOP，または EC-I-O-EOP-OVERFLOW の場合は，入出力状態が 00 となります。

注※2

入出力以外の例外の場合とは，次の状態を指します。

- 例外が検出されていない状態
- RAISE 文で，例外名 EC-I-O に関連する例外を指定して実行した場合
- 例外名 EC-I-O 以外の例外が引き起こされている状態

EXCEPTION-FILE 関数については，マニュアル「COBOL2002　言語　標準仕様編」「11.23 EXCEPTION-FILE 関数」を参照してください。

(2)　EXCEPTION-LOCATION 関数

最新例外状態に関連する文の位置を返します。関数の戻り値の長さは，303 バイトです。

EXCEPTION-LOCATION 関数の戻り値は，プログラム／利用者定義関数／メソッドの名称，手続き名，およびソース行識別子から構成され，それぞれがセミコロン（;）と空白 1 文字で区切られた形式となっています。

EXCEPTION-LOCATION 関数の戻り値を，次に示します。

例外の有無	関数の戻り値の内容
例外あり	• プログラム／利用者定義関数／メソッドの名称 • 手続き名 • ソース行識別子
例外なし	英数字空白

例外ありの場合の形式

```
AAA…AA;△BB…B;△CCCCCCC/CCC
```

AAA…AA

　プログラム／利用者定義関数／メソッドの名称

BB…B

　手続き名

CCCCCCC/CCC

　ソース行識別子

△

　半角空白文字を示します。

それぞれの項目に返される値を，次に示します。

プログラム／利用者定義関数／メソッド

プログラム／利用者定義関数／メソッド名称として返される値について，次に示します。

例外の引き起こされた位置を含む定義の種別	関数の戻り値の内容
プログラム	プログラム名
利用者定義関数	利用者定義関数名
メソッド	メソッド名

手続き名

手続き名として返される値について，次に示します。

手続き名は，例外が発生した文の段落名，節名の有無によって出力内容が異なります。

手続きの種別	関数の戻り値の内容
段落名，節名あり※	段落名 OF 節名
段落名だけあり	段落名
節名だけあり※	節名
段落名，節名なし	何も戻さない

注※

-Optimize,2 オプションの指定によって，そと PERFORM 文のインライン展開が処理された場合，インライン展開によって段落名および手続き文が移動されます。このため，インライン展開された個所で例外が発生したとき，インライン展開されていない場合に返される手続き名の値と相違があります。

そと PERFORM 文のインライン展開がされなかった個所（COMPUTE 文）で例外が発生した場合に返される手続き名の相違の例を，次に示します。

インライン展開前の形式	インライン展開後の形式
AAA SECTION. 　　PERFORM XXX. 　　STOP RUN. BBB SECTION. XXX. 　　**COMPUTE A = B + 1.** CCC SECTION. 　　COMPUTE A = B + 2.	AAA SECTION. XXX. 　　**COMPUTE A = B + 1.** 　　STOP RUN. BBB SECTION. CCC SECTION. 　　COMPUTE A = B + 2.
返される手続き名 XXX△OF△BBB	返される手続き名 XXX△OF△AAA

（凡例）　△：空白（X'20'）

ソース行識別子

ソース行識別子として返される値について，次に示します。

このシステムでのソース行識別子は，例外が発生した文の行番号 7 けたと欄番号 3 けたの間に/を追加したものとなります。なお，行番号が 7 けたを超える場合，または欄番号が 3 けたを超える場合，各最大けたを超える上位けたについては，けた落ちが発生します。

EXCEPTION-LOCATION 関数については，マニュアル「COBOL2002　言語　標準仕様編」「11.24 EXCEPTION-LOCATION 関数」を参照してください。

(3)　EXCEPTION-STATEMENT 関数

例外が引き起こされた文の名前を返します。関数の戻り値の長さは，31 バイトです。

EXCEPTION-STATEMENT 関数の戻り値を，次に示します。

例外の有無	関数の戻り値の内容
例外あり	例外の発生した文の名前
例外なし	英数字空白 31 文字

EXCEPTION-STATEMENT 関数については，マニュアル「COBOL2002　言語　標準仕様編」「11.25　EXCEPTION-STATEMENT 関数」を参照してください。

(4) EXCEPTION-STATUS 関数

最新例外状態に関連する例外名称を返します。関数の戻り値の長さは，31 バイトです。

EXCEPTION-STATUS 関数の戻り値を，次に示します。

引き起こされた例外の種別	関数の戻り値の内容
例外名	例外名
例外オブジェクト	EXCEPTION-OBJECT
例外なし	英数字空白 31 文字

EXCEPTION-STATUS 関数については，マニュアル「COBOL2002　言語　標準仕様編」「11.26 EXCEPTION-STATUS 関数」を参照してください。

(5) 使用例

組み込み関数を使用して最新例外状態を参照する COBOL プログラムの例を，次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE.
            :
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       SELECT INFILE  ASSIGN 'INPUT-FILE'.
       SELECT OUTFILE ASSIGN 'OUTPUT-FILE'.
            :
       DATA DIVISION.
       FILE SECTION.
       FD INFILE.
       01 INREC   PIC X(10).
       FD OUTFILE.
       01 OUTREC  PIC X(10).
       WORKING-STORAGE SECTION.
       01 EOF-FLG PIC 9 VALUE 0.
            :
       PROCEDURE DIVISION.
       DECLARATIVES.
       EXCEPTION-HANDLING SECTION.
           USE AFTER EXCEPTION CONDITION EC-I-O.  *> 2.
             DISPLAY FUNCTION EXCEPTION-FILE.
             DISPLAY FUNCTION EXCEPTION-LOCATION.
             DISPLAY FUNCTION EXCEPTION-STATEMENT.
             DISPLAY FUNCTION EXCEPTION-STATUS.
             IF FUNCTION EXCEPTION-STATEMENT = 'OPEN' THEN
               IF FUNCTION EXCEPTION-FILE (3  :  ) = 'OUTFILE' THEN
                 RESUME AT ERROR-HANDLE2
               END-IF
               RESUME AT ERROR-HANDLE3
             END-IF.
             RESUME AT ERROR-HANDLE1.
       END DECLARATIVES.
```

```
>>TURN EC-I-O CHECKING ON
    OPEN INPUT  INFILE. *> 1.
    OPEN OUTPUT OUTFILE.
    PERFORM UNTIL EOF-FLG = 1
      READ INFILE
      WRITE OUTREC FROM INREC
    END-PERFORM.
ERROR-HANDLE1.
    CLOSE OUTFILE.
ERROR-HANDLE2.
    CLOSE INFILE.
ERROR-HANDLE3. *> 3.
    STOP RUN.
END PROGRAM 'SAMPLE'.
```

1. 物理ファイル「INPUT-FILE」がない場合，OPEN 文で永続誤りが発生し，最新例外状態が更新されます。
2. 例外に該当する例外宣言手続きが実行されます。この手続き中で組み込み関数を使用すれば，例外の情報を参照できます。この例では，次の例外情報が組み込み関数の戻り値として取得されます。

組み込み関数名	関数の戻り値
EXCEPTION-FILE	35INFILE
EXCEPTION-LOCATION	SAMPLE;△;△0000035/011 ※
EXCEPTION-STATEMENT	OPEN
EXCEPTION-STATUS	EC-I-O-PERMANENT-ERROR

（凡例）

△：半角空白文字を示す

注※

例外が発生した OPEN 文の行番号が 35，欄番号が 11 の場合です。

3. 例外宣言手続き中の IF 文の判定によって，RESUME ERROR-HANDLE3 が実行され，手続き ERROR-HANDLE3 に復帰します。

(6) 注意事項

- EXCEPTION-STATEMENT 関数，EXCEPTION-STATUS 関数では，関数の戻り値の英数字は，すべて大文字となります。
- 関数の戻り値に補われるスラント（/），セミコロン（;），空白文字は，英数字文字として扱われます。
- 関数の戻り値の長さに満たない場合は，英数字空白が補われます。
- EXCEPTION-FILE 関数，および EXCEPTION-LOCATION 関数に含まれる利用者定義語は，-EquivRule,NotExtend，-EquivRule,NotAny，または-EquivRule,StdCode オプションが指定されていない場合，等価規則に基づいて変換されます。

- EXCEPTION-LOCATION 関数でのプログラム／利用者定義関数／メソッドの名称は，プログラム名，メソッド名，および利用者定義関数名の変換規則に基づいて変換されます。

21.7.2 EXCEPTION-OBJECT

例外オブジェクトが引き起こされると，そのオブジェクト参照が EXCEPTION-OBJECT にセットされます。宣言手続き中で EXCEPTION-OBJECT を参照すると，引き起こされたオブジェクトへの参照を取得できます。

EXCEPTION-OBJECT の詳細については，マニュアル「COBOL2002　言語　標準仕様編」「4.3.2(5) EXCEPTION-OBJECT」を参照してください。

EXCEPTION-OBJECT の規則

- 例外オブジェクトが引き起こされていない場合，または例外名による例外が引き起こされた場合，EXCEPTION-OBJECT には NULL がセットされます。
- EXCEPTION-OBJECT は，値の参照およびクリアだけができます。値の代入はできません。
- EXCEPTION-OBJECT は，外部属性を持つオブジェクト参照データ項目で，実行単位に一つだけ存在します。

使用例

EXCEPTION-OBJECT の使用例を，次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE.

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       REPOSITORY.
       CLASS EXCEPT-CLASS.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 OBJ USAGE OBJECT REFERENCE EXCEPT-CLASS.

       PROCEDURE DIVISION.
       DECLARATIVES.
       EXCEPTION-HANDLER SECTION.
           USE AFTER EXCEPTION OBJECT EXCEPT-CLASS. *> 2.
           INVOKE EXCEPTION-OBJECT 'SAMPLEMETHOD'.
       END DECLARATIVES.

           INVOKE EXCEPT-CLASS 'NEW' RETURNING  OBJ.
           RAISE  OBJ. *> 1.

       END PROGRAM SAMPLE.


       IDENTIFICATION DIVISION.
       CLASS-ID. EXCEPT-CLASS INHERITS BASE.
```

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
CLASS BASE.
IDENTIFICATION DIVISION.
OBJECT.

IDENTIFICATION DIVISION.

METHOD-ID. SAMPLEMETHOD.
PROCEDURE DIVISION.

    DISPLAY 'EXCEPTION-OBJECTの使用例'.

END METHOD SAMPLEMETHOD.
END OBJECT.
END CLASS EXCEPT-CLASS.
```

1. プログラム「SAMPLE」中の RAISE 文に，オブジェクト参照一意名が指定されているため，例外オブジェクトが引き起こされます。
2. 例外宣言手続き内で，INVOKE 文に EXCEPTION-OBJECT が指定されているため，例外オブジェクト（OBJ）を参照できます。SAMPLEMETHOD が実行されます。

21.7.3 最新例外状態のクリア

最新例外状態は，一度例外が発生したあとに処理を継続すると，次に例外が発生するまで，前の最新例外状態を示しています。前の最新例外状態を削除（クリア）するには，SET 文を使用します。このとき，EXCEPTION-OBJECT も，同時にクリアされます。

SET 文の詳細については，「COBOL2002　言語　標準仕様編」「10.8.43　SET 文」を参照してください。

SET 文を使用して最新例外状態をクリアする例を，次に示します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE.
PROCEDURE DIVISION.

    RAISE EXCEPTION EC-OVERFLOW-STRING. *> 1.
    DISPLAY '-----RAISE文実行後-----'. *> 2.
    DISPLAY 'EXCEPTION-STATEMENT = '
            FUNCTION EXCEPTION-STATEMENT. *> 2.
    DISPLAY 'EXCEPTION-STATUS = '
            FUNCTION EXCEPTION-STATUS. *> 2.

    SET LAST EXCEPTION TO OFF. *> 3.
    DISPLAY '-----最新例外状態のクリア後-----'. *> 4.
    DISPLAY 'EXCEPTION-STATEMENT = '
            FUNCTION EXCEPTION-STATEMENT. *> 4.
    DISPLAY 'EXCEPTION-STATUS = '
            FUNCTION EXCEPTION-STATUS. *> 4.
```

```
        STOP RUN.
    END PROGRAM SAMPLE.
```

1. RAISE 文によって例外が引き起こされ，最新例外状態が更新されます。
2. 1.で発生した最新例外状態の内容が DISPLAY 文によって出力されます。例外が発生した文の名前として「RAISE」が，最新例外状態の例外名として「EC-OVERFLOW-STRING」が，それぞれ出力されます。
3. SET 文によって，1.で設定された最新例外状態がクリアされます。
4. 2.と同様，最新例外状態の内容が DISPLAY 文によって出力されます。3.の SET 文によって最新例外状態がクリアされているため，例外が発生した文の名前，および最新例外状態の例外名には，それぞれ空白が出力されます。

21.8　例外の検出条件

共通例外処理では，文の種類や TURN 指令の有無，PROPAGATE 指令の有無などによって，検出される例外が異なります。

21.8.1　例外が検出される文の詳細

例外が検出される手続き文および組み込み関数の詳細を，次に示します。なお，例外名の一覧については，「21.2.1　例外名」の「(2)　例外名の一覧」を参照してください。

(1)　例外を検出する組み込み関数

例外を検出する組み込み関数の一覧を，次に示します。

組み込み関数	EC-ARGUMENT	
	FUNCTION	IMP
ACOS	○	○
ANNUITY	○	−
ASIN	○	○
ATAN	−	○
CHAR	○	−
CONVERT-CODE※	−	○
COS	−	○
COUNT-CHAR	○	−
DATE-OF-INTEGER	○	−
DAY-OF-INTEGER	○	−
DISPLAY-OF※	○	○
FACTORIAL	○	−
INTEGER-OF-DATE	○	−
INTEGER-OF-DAY	○	−
LENGTH-OF-SUBSTRING	○	−
LOG	○	○
LOG10	○	○
LOWER-CASE	○	−

組み込み関数	EC-ARGUMENT	
	FUNCTION	IMP
MEDIAN	○	○
MOD	○	−
NATIONAL-OF※	○	○
NUMVAL	○	−
NUMVAL-C	○	−
ORD	○	−
PRESENT-VALUE	○	−
RANDOM	○	−
REM	○	−
REVERSE	○	−
SIN	−	○
SQRT	○	○
SUBSTRING	○	−
TAN	−	○
UPPER-CASE	○	−

（凡例）

○：例外が検出される

−：例外が検出されない

注※

-FunctionECSup,CodeConvErr オプションが有効な場合，文字コード変換エラーによる EC-ARGUMENT-FUNCTION／EC-ARGUMENT-IMP 例外は検出されません。

上記の表に記述のない組み込み関数については，例外が検出されません。

(2) おのおのの例外が検出される文

おのおのの例外が検出される文について，次に示します。

文	EC-BOUND			EC-PROGRAM（利用者定義関数を指定可能な文）
	ODO	REF-MOD	SUBSCRIPT	
ACCEPT※	○	○	○	−
ADD	○	−	○	−
CALL	○	−	○	−

文	EC-BOUND			EC-PROGRAM（利用者定義関数を指定可能な文）
	ODO	REF-MOD	SUBSCRIPT	
CANCEL	−	−	○	−
COMPUTE	○	−	○	○
DISPLAY	−	○	○	○
DIVIDE	○	−	○	○
EVALUATE	−	○	○	○
EXIT	−	−	−	−
GOBACK	−	−	−	−
IF	−	○	○	○
INITIALIZE	○	○	○	−
INSPECT	○	○	○	−
INVOKE	○	−	○	−
MOVE	○	○	○	○
MULTIPLY	○	−	○	−
PERFORM	○	○	○	○
RAISE	−	−	−	−
READ	○	○	○	−
RELEASE	−	○	○	○
RETURN	○	○	○	−
SEARCH	−	○	○	○
SET	−	−	−	○
STRING	○	○	○	○
SUBTRACT	○	−	○	−
UNSTRING	○	○	○	○
WRITE	−	○	○	○

（凡例）

○：例外が検出される

−：例外が検出されない

注※

AIX の場合，画面節（WINDOW SECTION）の画面操作で使用する，FIRST FIELD 指定の一意名については，例外が検出されません。

21.8.2　例外検出での注意事項

(1)　共通の注意事項

- U レベルエラー（KCCCnnnnR-U）が発生した場合，発生した U レベルエラーに対しては，共通例外処理を実行できません。U レベルエラーは，常に実行時エラーとなります。

(2)　プログラム間連絡での注意事項

- CALL 文で実行可能ファイルを呼び出す場合，次に示す例外に対してだけ共通例外処理を実行できます。なお，実行可能ファイル内で引き起こされた例外は，呼び出し元プログラムに伝播できません。

表 21–9　実行可能ファイルの呼び出しで検出される例外と例外名

検出される例外	例外名
CALL 文で指定した実行可能ファイルが見つからない。	EC-PROGRAM-NOT-FOUND
実行可能ファイルを実行中にメモリが不足した。	EC-PROGRAM-RESOURCES
実行可能ファイルの処理中にエラーが発生した。	EC-PROGRAM-IMP

- COBOL85※で作成したプログラムを CALL 文に指定して呼び出す場合は，次に示す例外に対してだけ共通例外処理を実行できます。

注※

COBOL85 でコンパイルしたプログラムが動作可能なシステム

表 21–10　COBOL85 で作成したプログラムの呼び出しで検出される共通例外と例外名

共通例外処理で検出される例外	例外名
CALL 文で指定したプログラム名が，呼び出しできないプログラムである。	EC-PROGRAM-NOT-FOUND
動的なリンクの処理中にメモリが不足した。	EC-PROGRAM-RESOURCES
共用ライブラリのロード中にエラーが発生した。	EC-PROGRAM-IMP

- CALL 文に，ENTRY 文で定義した呼び出し先プログラムの入口点を指定し，かつその入口点に指定したプログラムがマルチスレッド対応 COBOL プログラムの場合は，次に示す共通例外処理を実行できません。

表 21–11　CALL 文に ENTRY 文で定義した入口点を指定した場合に共通例外処理ができない例外

共通例外処理ができない例外	例外名
マルチスレッド機能の処理中にメモリが不足した。	EC-PROGRAM-RESOURCES

- 再帰属性でないプログラムを再帰的に呼び出した場合，例外名 EC-PROGRAM-RECURSIVE-CALL が引き起こされるかどうかは，呼び出し先プロブラムの入口点時点で該当する例外の有効状態に従います。

(3)　入出力での注意事項

- 共通例外処理では，プログラムが順次処理していく過程で例外を検出するため，発生する例外は，常に 1 種類となります。ただし，次に示す場合は，処理の順序に関係なく，致命的な例外が優先して検出されます。
  - ページあふれ条件での例外が発生したあと，LINAGE 句に指定したデータ名の値不正によって例外が検出された場合。
- 入出力文を実行する際に，SELECT 句のファイル名に割り当てられた物理ファイル，または物理ファイルに対するレコードが，ほかの実行単位によって排他モードで使用されている場合，入出力文の実行は不成功となり，入出力状態が 9x を示しますが，例外名 EC-I-O-IMP は検出されません。

(4)　コンパイラオプションとの関連性

共通例外処理とコンパイラオプションとの関連性について，次に示します。

- デバッグオプションを指定した場合に実行されるエラーチェックと，共通例外処理での例外の検出は，同時に実行できません。デバッグオプションを指定した場合に無効となる例外名を，次に示します。

表 21–12　デバッグオプション指定時に無効となる例外名

デバッグオプション	デバッグオプション指定時に無効となる例外名
-DebugCompati	• EC-PROGRAM-ARG-MISMATCH • EC-BOUND-REF-MOD • EC-BOUND-SUBSCRIPT
-DebugRange	• EC-BOUND-REF-MOD • EC-BOUND-SUBSCRIPT

- デバッグオプションによるエラーチェックは，共通例外処理の例外の検出よりも優先されます。そのため，デバッグオプションの指定の有無によって，実行結果が異なることがあります。
- EC-ALL や複数の例外名を指定した場合，-DebugData オプションの指定の有無によって，検出される例外が変わることがあります。

21.8.3　例外処理の動作

共通例外処理では，例外の検出された要因，および例外処理の指定有無によって，実行される処理が異なります。それぞれの場合の処理について，「表 21–13　手続き文で検出された例外」，「表 21–14　RAISE 文によって引き起こされた例外」，および「表 21–15　例外の伝播によって検出された例外」に示します。

なお，TURN 指令のチェックが ON の場合の動作は，表の左から順にチェックされ，該当するものが実行されます。

表 21–13　手続き文で検出された例外

例外の致命度	TURN 指令のチェックが OFF	TURN 指令のチェックが ON			
		該当する宣言手続きがある※1	PROPAGATE 指令が ON		例外処理なし
			呼び出し元が例外を受け取れないプログラム※2	左記以外	
致命的	実行時エラーまたは実行を継続※3	該当する宣言手続きを実行したあと，エラーメッセージ（KCCC0015R-S）を出力し，実行単位が異常終了※4	エラーメッセージ（KCCC0403R-S）を出力し，実行単位が異常終了	例外を伝播	エラーメッセージ（KCCC0401R-S）を出力し，実行単位が異常終了
非致命的	実行を継続※5	該当する宣言手続きを実行し，次の文から実行を継続	実行を継続※5	実行を継続※5	実行を継続※5

注※1

宣言手続きからの復帰が実行されない場合です。

注※2

例外を受け取れないプログラムの詳細については，「21.5.3　例外を受け取れないプログラムに例外を伝播させた場合の動作」を参照してください。

注※3

致命的な例外に対する TURN 指令のチェックが OFF の場合の動作については，「21.3.3　例外チェックが無効な場合の動作」の「(1)　手続き文の実行中にエラーが発生し，例外を検出した場合」を参照してください。

注※4

致命的な例外のうち，入出力に分類される例外（例外名 EC-I-O）については，実行が継続されます。

注※5

実行継続の詳細については，各文の一般規則を参照してください。

表 21–14　RAISE 文によって引き起こされた例外

例外の致命度	該当する宣言手続きがある※1	PROPAGATE 指令が ON	例外処理なし
致命的	該当する宣言手続きを実行したあと，エラーメッセージ（KCCC0015R-S）を出力し，実行単位が異常終了※2	例外を伝播	エラーメッセージ（KCCC0401R-S）を出力し，実行単位が異常終了
非致命的	該当する宣言手続きを実行し，RAISE 文の次の文から実行を継続	RAISE 文の次の文から実行を継続	RAISE 文の次の文から実行を継続

注※1

宣言手続きからの復帰が実行されない場合です。

注※2

致命的な例外のうち，入出力に分類される例外（例外名 EC-I-O）については，実行が継続されます。

表 21–15　例外の伝播によって検出された例外

例外の致命度	TURN 指令のチェックが OFF	TURN 指令のチェックが ON		
		該当する宣言手続きがある※1	PROPAGATE 指令が ON	例外処理なし
致命的	エラーメッセージ（KCCC0402R-S）を出力し，実行単位が異常終了	該当する宣言手続きを実行したあと，エラーメッセージ（KCCC0015R-S）を出力し，実行単位が異常終了※2	例外を伝播	エラーメッセージ（KCCC0401R-S）を出力し，実行単位が異常終了
非致命的※3	次の文から実行を継続※4	該当する宣言手続きを実行し，次の文から実行を継続※4	次の文から実行を継続※4	次の文から実行を継続※4

注※1

宣言手続きからの復帰が実行されない場合です。

注※2

致命的な例外のうち，入出力に分類される例外（例外名 EC-I-O）については，実行が継続されます。

注※3

EXIT 文，GOBACK 文の RAISING 指定によって例外が伝播した場合だけが対象です。

注※4

次の文とは，CALL 文，INVOKE 文，および利用者定義関数を呼び出した文の次の文を指します。

21.9　共通例外処理の注意事項

21.9.1　共通例外処理を使用した場合の性能について

共通例外処理を使用する場合（TURN 指令のチェックを ON にした場合），例外処理のためのオブジェクトが生成されるため，TURN 指令のチェックを OFF にした場合と比較して，オブジェクトサイズや実行性能が劣化します。

21.9.2　従来形式の例外処理と共通例外処理の関係

(1)　従来形式の例外処理を実行する場合の最新例外状態の更新

従来形式の例外処理を実行する場合，検出した例外に対する例外名の TURN 指令のチェックが ON であれば，最新例外状態が更新されます。ただし，従来形式の例外処理のうち ON SIZE ERROR 指定については，ON SIZE ERROR を実行する際に例外が検出されないため，TURN 指令のチェックが ON であっても最新例外状態は更新されません。従来形式の例外処理を実行する場合に，最新例外状態が更新される例外処理と，更新されない例外処理を，次に示します。

最新例外状態が更新される従来形式の例外処理

- 従来形式の宣言手続き
- READ 文，SEARCH 文の AT END 指定
- WRITE 文の AT EOP 指定
- WRITE 文，REWRITE 文，DELETE 文，START 文の INVALID KEY 指定
- CALL 文の ON OVERFLOW，ON EXCEPTION 指定
- STRING 文，UNSTRING 文の ON OVERFLOW 指定

最新例外状態が更新されない従来形式の例外処理

- ADD 文，SUBTRACT 文，MULTIPLY 文，DIVIDE 文，COMPUTE 文の ON SIZE ERROR 指定

(2)　従来形式の宣言手続きと共通例外処理の宣言手続きの優先順序

従来形式の宣言手続きと共通例外処理の宣言手続きを同時に指定した場合，従来形式の宣言手続きが優先して実行されます。ただし，例外を検出したプログラムが入れ子のプログラムで，上位プログラムに GLOBAL 句を指定した従来形式の宣言手続きが定義されているときは，入れ子のプログラム中の共通例外処理の宣言手続きが優先して実行されます。

従来形式の宣言手続きが優先される例を，次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE.

       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       SELECT INFILE ASSIGN 'INPUT-FILE'.

       DATA DIVISION.
       FILE SECTION.
       FD INFILE.
       01 INREC    PIC X(10).
       WORKING-STORAGE SECTION.
       01 READDATA PIC X(10).

       PROCEDURE DIVISION.
       DECLARATIVES.
       EXCEPTION-HANDLE1 SECTION.
           USE AFTER STANDARD EXCEPTION PROCEDURE INFILE. *> 2.
           DISPLAY '従来形式USE実行'.
       EXCEPTION-HANDLE2 SECTION.
           USE AFTER EXCEPTION CONDITION EC-I-O. *> 3.
           DISPLAY '共通例外処理USE実行'.
       END DECLARATIVES.

       >>TURN EC-I-O CHECKING ON
         OPEN INPUT INFILE.

         READ INFILE INTO READDATA. *> 1.
         DISPLAY READDATA.

         CLOSE INFILE.

       END PROGRAM SAMPLE.
```

上記の例の場合，プログラム中の READ 文（1.）によって例外が引き起こされた場合，従来形式の宣言手続き（2.）が共通例外処理（3.）より優先して実行されます。

(3) FILE STATUS 句の指定と共通例外処理での異常終了

COBOL85 では，入出力エラーが発生した場合に FILE STATUS 句の指定があると実行を継続しますが，COBOL2002 では，FILE STATUS 句より共通例外処理が優先されます。そのため，入出力エラーが発生して共通例外処理が実行された場合，FILE STATUS 句の指定があっても実行が継続されません。

FILE STATUS 句の指定の有無と共通例外処理の関係を，次に示します。

発生した入出力エラーに対応する TURN 指令の状態	入出力エラーに対応する USE 文	FILE STATUS 句の有無	入出力エラー発生時の動作
ON	なし	○	実行時エラー※
		×	実行時エラー※

発生した入出力エラーに対応するTURN指令の状態	入出力エラーに対応するUSE文	FILE STATUS句の有無	入出力エラー発生時の動作
	新形式	○	新形式のUSE文を実行
		×	新形式のUSE文を実行
	従来形式	○	従来形式のUSE文を実行
		×	従来形式のUSE文を実行
	新形式 従来形式	○	従来形式のUSE文を実行
		×	従来形式のUSE文を実行
OFF	なし	○	実行を継続
		×	実行時エラー※
	新形式	○	実行を継続
		×	実行時エラー※
	従来形式	○	従来形式のUSE文を実行
		×	従来形式のUSE文を実行
	新形式 従来形式	○	従来形式のUSE文を実行
		×	従来形式のUSE文を実行

(凡例)

ON：入出力エラーに対応するTURN指令のチェックがON

OFF：入出力エラーに対応するTURN指令のチェックがOFF

なし：入出力エラーに対応するUSE文がない

新形式：入出力エラーに対応するCOBOL2002形式のUSE文がある

従来形式：入出力エラーに対応する従来のCOBOL形式のUSE文がある

○：FILE STATUS句が指定されている

×：FILE STATUS句が指定されていない

注※

発生した入出力エラーが非致命的な場合，実行時エラーとはならないで，実行が継続されます。

22 データコミュニケーション機能

データコミュニケーション機能は，さまざまな端末またはコンピュータシステムと，メッセージを受け渡しする機能です。この章では，データコミュニケーション機能の使用方法について説明します。

22.1　データコミュニケーション機能の概要

データコミュニケーション機能は，オンラインコントロールプログラムを経由して，端末，ファイル，またはほかのプログラムとメッセージを受け渡しする機能です。このシステムでは，オンラインコントロールプログラムとして，OpenTP1（分散トランザクション処理機能）を使用できます。

データコミュニケーション機能の文法規則については，マニュアル「COBOL2002　言語　拡張仕様編」「8.　データコミュニケーション機能」を参照してください。

データコミュニケーション機能で使用する文

データコミュニケーション機能は，メインフレーム（VOS3）COBOL85 のデータコミュニケーション機能と同じインタフェースの SEND 文や RECEIVE 文でメッセージの送受信などができます。

データコミュニケーション機能で用いる文とその機能を，次に示します。

表 22-1　データコミュニケーション機能で用いる文

文	機能
RECEIVE	メッセージ受信
SEND	メッセージ送信
ENABLE	ファイル送信の開始
DISABLE	ファイル送信の終了
COMMIT	同期点取得処理
ROLLBACK	部分回復処理

なお，データコミュニケーション機能で使用する文の厳密な意味については，オンラインコントロールプログラム側で規定しています。

22.2 DC シミュレーション

テストデバッガを使用すると，OpenTP1 が組み込まれていない環境であっても，COBOL2002 で使用できる擬似 OpenTP1 用ライブラリをリンク時に指定すれば，DC シミュレーションができます。

DC シミュレーションの操作方法については，マニュアル「COBOL2002　使用の手引　操作編」のテストデバッガの説明を参照してください。

リンク方法

擬似 OpenTP1 用ライブラリを使用した場合の実行可能ファイル生成例を次に示します。

AIX(32)，Linux の場合

```
ccbl2002 -Main,System sample.cbl -lcbl2kdc
```

AIX(64)の場合

```
ccbl2002 -Main,System sample.cbl -lcbl2kdc64
```

注意事項

- DC シミュレーション機能を使用するプログラムから実行可能ファイルまたは共用ライブラリを作成する場合は，リンク時に-lcbl2kdc または-lcbl2kdc64 オプションの指定が必要です。
- 擬似 OpenTP1 用ライブラリをリンクした実行可能ファイルで，DC シミュレーションをしない場合，擬似 OpenTP1 関数を実行中であることを通知するメッセージが表示されます。
- OpenTP1 では，標準入出力（stdin，stdout，stderr）のリダイレクションが行われる場合があるため，COBOL 実行時メッセージ，および ACCEPT／DISPLAY 文については，出力先に注意する必要があります。

22.3　データコミュニケーション機能を使用したCOBOLプログラムの例

データコミュニケーション機能を使用したCOBOLプログラムの例を，次に示します。

```
          :
DATA DIVISION.
          :
WORKING-STORAGE SECTION.
01 データ名1 PIC X(100).
01 データ名2 PIC X(100).
01 データ名3 PIC X(100).
          :
COMMUNICATION SECTION.
CD 通信記述名1 FOR I-O
   STATUS KEY IS データ名4.
CD 通信記述名2 FOR OUTPUT
   STATUS KEY IS データ名5
   SYMBOLIC TERMINAL IS データ名6
   MAP NAME IS データ名7.
          :
PROCEDURE DIVISION.
          :
    RECEIVE 通信記述名1 MESSAGE INTO データ名1. *> メッセージ受信
          :
*   業務処理
          :
    SEND    通信記述名2 FROM データ名2 *> 分岐メッセージ送信
                        WITH EMI.
          :
    SEND    通信記述名1 FROM データ名3 *> 応答メッセージ送信
                        WITH EMI.
          :
    IF （エラー発生か？） THEN
       ROLLBACK
    END-IF.
          :
```

23 データベース操作機能（Linux で有効）

COBOL2002 では，データベースインタフェース機能として ODBC インタフェース機能をサポートしています。ODBC インタフェース機能を使うことによって，SQL 埋め込み COBOL プログラムで，データベースにアクセスできます。この章では，ODBC インタフェース機能について説明します。

23.1　データベースアクセス機能

この節では，データベースアクセス機能について説明します。

なお，データベースアクセス機能の文法規則については，マニュアル「COBOL2002　言語　拡張仕様編」「9.　データベースアクセス機能」を参照してください。

23.1.1　埋め込み SQL 文を使った COBOL プログラムの作成

埋め込み SQL 文によるデータベースアクセスの方法は，SQL を COBOL プログラム中に埋め込んで実行する方法（静的に行う方法）と，SQL を実行時に生成し，SQL 文の埋め込み変数に設定して実行する方法（動的に行う方法）があります。

埋め込み SQL 文の SELECT 文は，表から 1 行分の値を取り出す文（SELECT 文：単一行）です。複数行の値を取り出す場合は，カーソルを使用します。カーソルの使用には，静的に行う方法と動的に行う方法があります。また，ストアドプロシージャを呼び出して，一連の手続きを行う方法もあります。

(1)　COBOL プログラムで使用できる SQL 文

COBOL プログラムで使用できる埋め込み SQL 文を次に示します。

表 23–1　COBOL プログラムで使用できる埋め込み SQL 文

種類	分類	SQL 文	内容
宣言系	埋め込み SQL 宣言節	BEGIN DECLARE SECTION	埋め込み SQL 開始宣言
		END DECLARE SECTION	埋め込み SQL 終了宣言
	埋め込み例外宣言	WHENEVER	埋め込み例外宣言
	カーソル	DECLARE CURSOR	カーソル宣言
制御系	コネクション	CONNECT	コネクションを確立する
		DISCONNECT	コネクションを解除する
		SET CONNECTION	現行コネクションを変更する
	トランザクション	COMMIT	コミットでトランザクションを終了させる
		ROLLBACK	ロールバックでトランザクションを終了させる
操作系	静的	SELECT	表の指定された行から値を取り出す
		INSERT	表に新しい行を作成する
		DELETE	表から行を削除する
		UPDATE	表の行を更新する

種類	分類	SQL 文	内容
	カーソル	OPEN	カーソルを開く
		FETCH	カーソルを表の次の行に位置づけ，その行を取り出す
		CLOSE	カーソルを閉じる
	ストアドプロシージャ	CALL	ストアドプロシージャを呼び出す
	動的	EXECUTE IMMEDIATE	動的 SQL を準備し，実行する
		PREPARE	動的 SQL を準備する
		DEALLOCATE PREPARE	準備した動的 SQL を解放する
		EXECUTE	動的 SQL を実行する

(2) 埋め込み SQL 宣言節

埋め込み SQL 開始宣言から埋め込み SQL 終了宣言までを，埋め込み SQL 宣言節といいます。埋め込み SQL 宣言節は，データ部に定義します。

埋め込み SQL 宣言節には，SQLCODE 変数，埋め込み変数，および標識変数を定義します。

(3) 埋め込み例外宣言

埋め込み例外宣言は，宣言系以外の SQL 文の実行後の例外処理を記述します。

(4) カーソル宣言

カーソル宣言は，OPEN 文などでカーソルを使用する場合に，手続き部に定義します。カーソル宣言に誤りがあると，そのカーソル名に対する OPEN 文の実行がエラーになります。

(5) 識別子

カーソル名，表名，列名，プロシージャ名などを表す識別子は，データベースによっても規定されます。したがって，正常にコンパイルされたプログラムでも，SQL 文の実行が失敗することがあります。このような場合は，DBMS のマニュアルなどを参照してください。また，-EquivRule コンパイラオプションの指定がない場合，等価規則が適用されます。等価規則については，マニュアル「COBOL2002　言語　標準仕様編」「4.1.2　COBOL 文字集合」を参照してください。

使用するデータベースを意識しないで運用したい場合は，カーソル名を英字で始まる 18 文字以内の文字列で定義することを推奨します。

(6) トランザクション

トランザクションは，コネクションを確立したとき，または明示的にコミットやロールバックしたときから開始され，明示的，または暗黙的にコミット，ロールバックしたときに終了されます。トランザクションは，コネクションごとに管理されます。

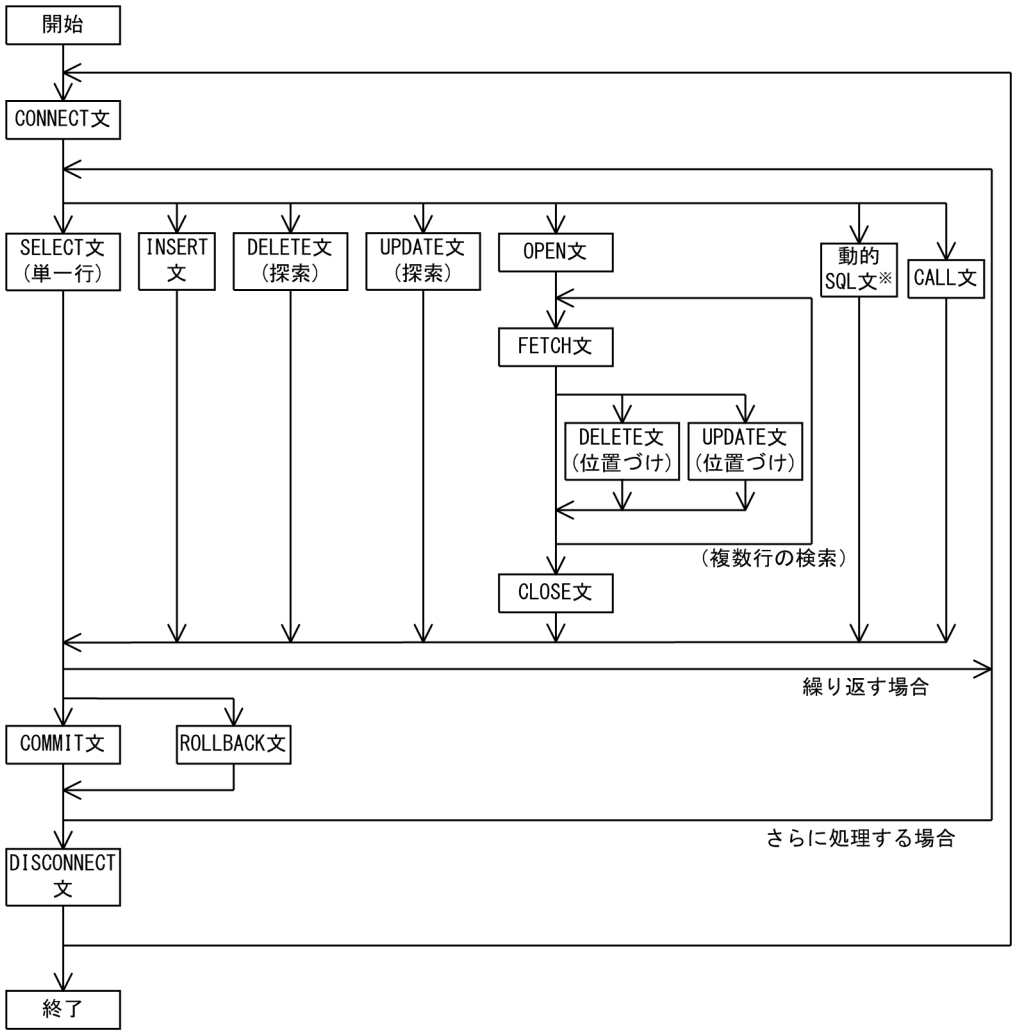
一つのデータベースに対して複数のプログラム（コネクション）からアクセスする場合，デッドロックが発生する可能性があります。デッドロックが発生する可能性を低くするために，プログラム作成時には次のことに注意してください。

- 同時にアクセスするすべてのプログラムのデータベースへのアクセス順序を同じにすると，デッドロックの発生する可能性は低くなります。例えば，データベースへの更新アクセスをストアドプロシージャで登録し，これを使用することでアクセス順序を規定できます。
- 応答処理を含むプログラムの場合，応答処理がトランザクション内に存在すると，応答時間分，ほかのトランザクションによるアクセスがブロックされることがあります。このようなトランザクション内での応答処理をしないようにプログラムを作成することで，ブロックする時間を最小にできます。
- デッドロックは，主に，同じデータベースに対するアクセスで長時間動作する複数のトランザクションが同時に実行されている場合に起こります。トランザクションが長くなれば，排他ロックまたは更新ロックが長時間になり，ほかの処理をブロックしてしまうので，デッドロックが発生する可能性が高くなります。このことから，トランザクションを最小単位で行うことで，デッドロックが発生する可能性を低くできます。

(7) SQL 文の実行順序

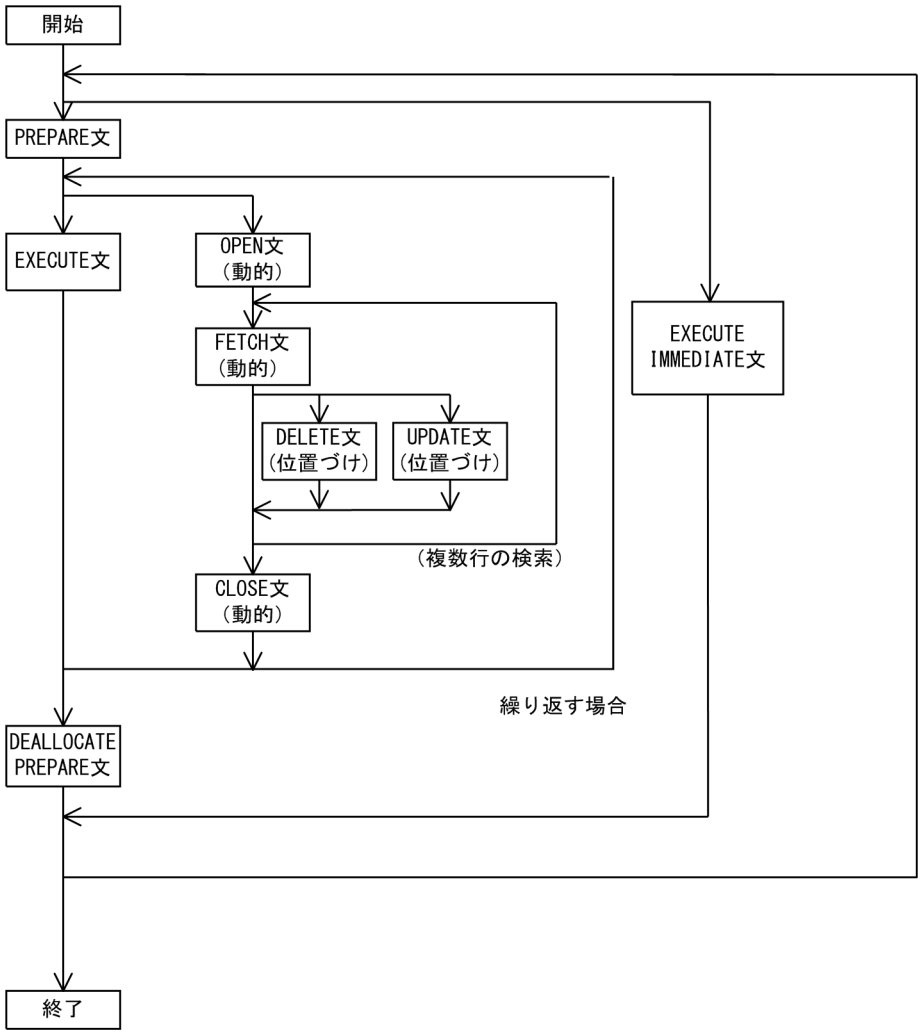
SQL 文は，次に示す順序で実行する必要があります。

図 23-1　SQL 文の実行順序



注※　動的 SQL 文を使用する場合については，次に示す順序を参照してください。

図 23-2　動的 SQL 文の実行順序



23.1.2　プログラムの例

埋め込み SQL 文を使って，データベースの表にアクセスする例を示します。表は，DBMS が提供するツールなどを使用して定義します。詳細は，使用する DBMS の SQL リファレンスなどを参照してください。

なお，ここに示すのは，HiRDB を使用した場合のアクセス方法の例です。

(1)　静的に行う方法によるプログラムの例

(a)　表の定義

氏名と住所にそれぞれ対応する，PERSONNAME 列と PERSONADDRESS 列を持つ「ADDRESSBOOK」という住所録の表を定義します。

```
CREATE TABLE ADDRESSBOOK ( PERSONNAME char(20),
                           PERSONADDRESS varchar(255))
```

(b) 表へのアクセス

「ADDRESSBOOK」の表にアクセスします。

すでに表へ登録されている人の場合，住所を更新します。また，登録されていない人の場合，住所と氏名を新規に登録します。

```
      *<埋め込みSQL宣言節>
           EXEC SQL BEGIN DECLARE SECTION END-EXEC.
           01 ODBC-DSN PIC X(10) VALUE 'SAMPLE'.
           01 ODBC-UID PIC X(10) VALUE 'USER1'.
           01 ODBC-PWD PIC X(10) VALUE 'USER1'.
           01 行数     PIC S9(9) USAGE COMP VALUE ZERO.
           01 ADDRESSBOOK.
            02 PERSONNAME    PIC  X(20).
            02 PERSONADDRESS.
             03 ODBC-length PIC S9(9) USAGE COMP.
             03 ODBC-char   PIC  X(255).
           EXEC SQL END DECLARE SECTION END-EXEC.
          :
       PROCEDURE DIVISION.
          :
      *<埋め込み例外宣言>
           EXEC SQL WHENEVER SQLERROR STOP END-EXEC.
      *<コネクションの確立>
           EXEC SQL
             CONNECT  :ODBC-UID  IDENTIFIED BY :ODBC-PWD
             USING    :ODBC-DSN
           END-EXEC.
      *<埋め込み例外宣言>
           EXEC SQL WHENEVER SQLERROR
                    GO TO :ROLLBACK-PROC END-EXEC.
           EXEC SQL WHENEVER NOT FOUND
                    PERFORM :NOTFOUND-PROC END-EXEC.
      *<登録されているかを参照する>
           MOVE '日立 太郎' TO PERSONNAME.
           MOVE '福岡市博多区１丁目' TO ODBC-char OF PERSONADDRESS.
           MOVE 27 TO ODBC-length OF PERSONADDRESS.
           EXEC SQL
             SELECT COUNT(*) INTO :行数 FROM ADDRESSBOOK
             WHERE PERSONNAME = :PERSONNAME
           END-EXEC.
      *<住所録の表に登録する>
           IF 行数 = 0 THEN
             EXEC SQL
               INSERT INTO ADDRESSBOOK ( PERSONNAME, PERSONADDRESS)
               VALUES ( :PERSONNAME, :PERSONADDRESS)
             END-EXEC
      *<住所録の表の住所を更新する>
           ELSE
             EXEC SQL
               UPDATE ADDRESSBOOK SET PERSONADDRESS = :PERSONADDRESS
                           WHERE PERSONNAME = :PERSONNAME
             END-EXEC
           END-IF.
           EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
```

```
           EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
           EXEC SQL
             COMMIT WORK
           END-EXEC.
           GO TO DISCONNECT-PROC.
       ROLLBACK-PROC.
           EXEC SQL
             ROLLBACK WORK
           END-EXEC.
       DISCONNECT-PROC.
           EXEC SQL
             DISCONNECT
           END-EXEC.
           :
           STOP RUN.
       NOTFOUND-PROC SECTION.
           :
       NOTFOUND-PROC-END.
           EXIT.
           :
```

(2)　ストアドプロシージャの呼び出しの例

(a)　ストアドプロシージャの定義

「(1)　静的に行う方法によるプログラムの例」に示した「ADDRESSBOOK」の表にアクセスします。

すでに表へ登録されている人の場合，住所を更新し，表に未登録の人の場合，住所と氏名を登録するようなストアドプロシージャを定義します。

```
           CREATE PROCEDURE
             ADDRESSUPDATE (IN @PERSONNAME char(20)
                          , IN @PERSONADDRESS varchar(255) )
           BEGIN
             DECLARE REC_CNT INTEGER;
             SELECT COUNT(*) INTO REC_CNT FROM ADDRESSBOOK
               WHERE PERSONNAME = @PERSONNAME;
             IF 0 = REC_CNT THEN
               INSERT INTO ADDRESSBOOK ( PERSONNAME,PERSONADDRESS )
                 VALUES ( @PERSONNAME,@PERSONADDRESS );
             ELSE
               UPDATE ADDRESSBOOK SET PERSONADDRESS = @PERSONADDRESS
                 WHERE PERSONNAME = @PERSONNAME;
             END IF;
           END
```

(b)　ストアドプロシージャの呼び出し

定義したストアドプロシージャを呼び出します。

```
       *〈埋め込みSQL宣言節〉
           EXEC SQL BEGIN DECLARE SECTION END-EXEC.
           :
```

```
       01 ADDRESSUPDATE.
         02 PERSONNAME PIC X(20).
         02 PERSONADDRESS.
           03 ODBC-length PIC S9(9) USAGE COMP.
           03 ODBC-char   PIC X(255).
       EXEC SQL END DECLARE SECTION END-EXEC.
       :

 PROCEDURE DIVISION.
       :
*<住所録の表の住所を更新する>
       MOVE '日立 太郎' TO PERSONNAME.
       MOVE '福岡市博多区２丁目' TO ODBC-char OF PERSONADDRESS.
       MOVE 27 TO ODBC-length OF PERSONADDRESS.
       EXEC SQL
         CALL ADDRESSUPDATE ( :PERSONNAME, :PERSONADDRESS )
       END-EXEC.
       :
```

(3) 動的に行う方法によるプログラムの例

(a) 表の定義

氏名と住所にそれぞれ対応する，PERSONNAME 列と PERSONADDRESS 列を持つ「ADDRESSBOOK」という住所録の表を定義します。

```
       CREATE TABLE ADDRESSBOOK ( PERSONNAME char(20),
                                  PERSONADDRESS varchar(255))
```

(b) 表へのアクセス

「ADDRESSBOOK」の表にアクセスします。

住所録の表に住所，氏名を登録し，住所録一覧を取得します。

```
      *<埋め込みSQL宣言節>
       EXEC SQL BEGIN DECLARE SECTION END-EXEC.
      :
       01 動的SQL PIC X(80).
       01 ADDRESSBOOK.
         02 PERSONNAME PIC X(20).
         02 PERSONADDRESS.
           03 ODBC-length PIC S9(9) USAGE COMP.
           03 ODBC-char   PIC X(255).
       EXEC SQL END DECLARE SECTION END-EXEC.
      :
 PROCEDURE DIVISION.
       :
      *<埋め込み例外宣言>
       EXEC SQL WHENEVER SQLERROR
         GO TO :ROLLBACK-PROC END-EXEC.
       EXEC SQL WHENEVER NOT FOUND
         PERFORM :NOTFOUND-PROC END-EXEC.
```

```
*<住所録の表に登録する>
     MOVE 'INSERT INTO ADDRESSBOOK (PERSONNAME, PERSONADDRESS)'
-         'VALUES (?,?)' TO 動的SQL.
     EXEC SQL
       PREPARE DYNSQL1 FROM :動的SQL
     END-EXEC.
     MOVE '日立 太郎' TO PERSONNAME.
     MOVE '福岡市博多区３丁目' TO ODBC-char OF PERSONADDRESS.
     MOVE 27 TO ODBC-length OF PERSONADDRESS.
     EXEC SQL
       EXECUTE DYNSQL1 USING :PERSONNAME, :PERSONADDRESS
     END-EXEC.
     EXEC SQL
       DEALLOCATE PREPARE DYNSQL1
     END-EXEC.
     EXEC SQL
       COMMIT WORK
     END-EXEC.
*<住所録の表を一覧表示する>
     MOVE 'SELECT PERSONNAME, PERSONADDRESS FROM ADDRESSBOOK'
       TO 動的SQL.
     EXEC SQL
       PREPARE DYNSQL1 FROM :動的SQL
     END-EXEC.
     EXEC SQL
       DECLARE CRS00 CURSOR
         FOR DYNSQL1
     END-EXEC.
     EXEC SQL
       OPEN CRS00
     END-EXEC.
     EXEC SQL
         FETCH CRS00 INTO :PERSONNAME, :PERSONADDRESS
     END-EXEC.
      :
     EXEC SQL
       CLOSE CRS00
     END-EXEC.
     EXEC SQL
       DEALLOCATE PREPARE DYNSQL1
     END-EXEC.
     :
```

(4) 現行コネクションの変更の例

testdb1 の接続先データベースの「ADDRESSBOOK」の表にアクセスし，取得した住所データで testdb2 の接続先データベースの「RECIPIENTS」の表の住所を更新します。

```
      :
*<CONNDB1コネクションの確立>
     MOVE 'testdb1' TO ODBC-DSN.
     EXEC SQL
      CONNECT :ODBC-UID IDENTIFIED BY :ODBC-PWD …1.
        USING :ODBC-DSN AS CONNDB1
     END-EXEC.
*<住所を取得する>
```

```
      MOVE '日立 太郎' TO PERSONNAME.
      EXEC SQL
       SELECT PERSONADDRESS INTO :PERSONADDRESS FROM ADDRESSBOOK
                   WHERE PERSONNAME = :PERSONNAME
      END-EXEC.
       :
    *<CONNDB2コネクションの確立>
      MOVE 'testdb2' TO ODBC-DSN.
      EXEC SQL
       CONNECT :ODBC-UID IDENTIFIED BY :ODBC-PWD …2.
         USING :ODBC-DSN AS CONNDB2
      END-EXEC.
    *<取得した住所で更新する>
      MOVE '日立 太郎' TO PERSONNAME.
      EXEC SQL
        UPDATE RECIPIENTS SET PERSONADDRESS = :PERSONADDRESS
          WHERE PERSONNAME = :PERSONNAME
      END-EXEC.
       :
    *<現行コネクション(CONNDB2)を解除>
      EXEC SQL
       DISCONNECT CURRENT …3.
      END-EXEC.
    *<現行コネクションを変更する>
      EXEC SQL
       SET CONNECTION CONNDB1  …4.
      END-EXEC.
       :
```

例の説明

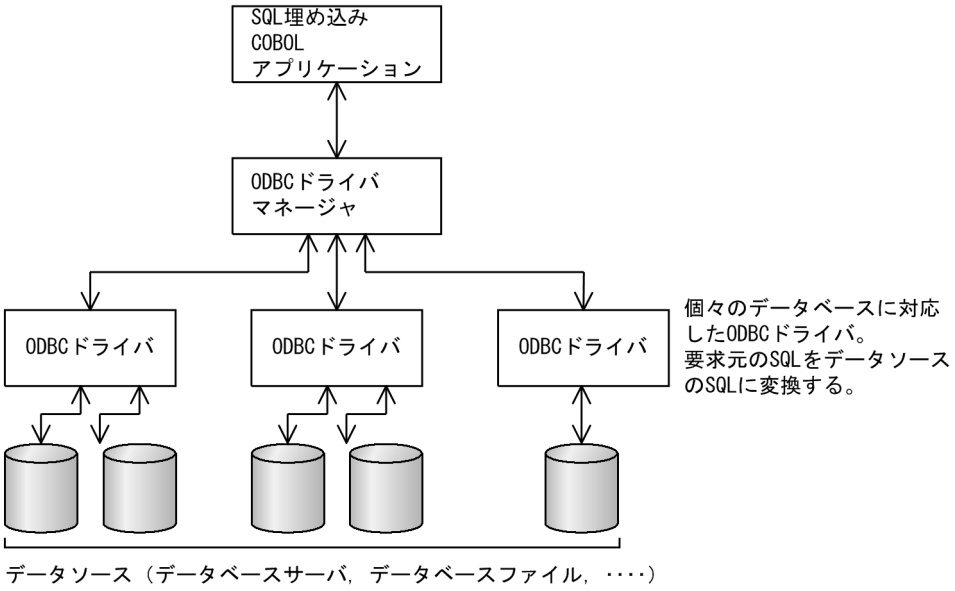
1. testdb1 への接続を確立。
   CONNDB1 が現行コネクションとなる。
2. testdb2 への接続を確立。
   CONNDB2 が現行コネクションとなる。
3. 現行コネクション CONNDB2 を解除。
   現行コネクションは不定となる。
4. 現行コネクションを CONNDB1 に変更。
   CONNDB1 が現行コネクションとなる。

23.2　ODBC インタフェース機能の概要

ODBC インタフェース機能は，COBOL プログラムからデータベースアクセス用標準言語の構造化照会言語（SQL）を使って，データベース管理システム（DBMS）上のデータにアクセスする機能です。

COBOL プログラムに記述された SQL 文は，ODBC API を使って ODBC ドライバマネージャに渡され，さらに適切な ODBC ドライバへ渡されます。ODBC ドライバは，渡された SQL 文を使ってデータベースにアクセスします。SQL 埋め込み COBOL アプリケーションがデータベースにアクセスする流れを，次に示します。

図 23–3　ODBC インタフェース機能の概要



注

Linux 上で動作する ODBC ドライバマネージャ（unixODBC）を別途インストールしてください。インストール方法およびデータソースの登録方法の詳細は，使用するデータベースのマニュアルなどを参照してください。

23.2.1　ODBC インタフェース機能が動作する環境

(1)　動作に必要な環境

ODBC インタフェース機能を使用するには，次のような環境が必要です。

- アクセスするデータが存在する DBMS
- 使用する DBMS がサーバ／クライアント形態の場合は，サーバへアクセスできるクライアント環境
- 使用する DBMS へアクセスできる ODBC ドライバ
- ODBC2.0 以降の ODBC ドライバマネージャ

(2)　ODBC の合致レベル

使用する ODBC ドライバが次のような ODBC の合致レベルを満たしていないと，SQL 埋め込み COBOL プログラムの実行時に，機能が制限されます。

(a)　API の合致レベル

ODBC ドライバは，コアおよびレベル 1 のすべての関数と，次のレベル 2 の関数をサポートしている必要があります。

- SQLDescribeParam（PREPARE 文を使用する場合）
- SQLNumParams（PREPARE 文を使用する場合）
- SQLExtendedFetch（FETCH 文を使用する場合）
- SQLMoreResult（FETCH（動的）文を使用する場合）
- SQLProcedureColumns（CALL 文を使用する場合）

(b)　SQL の合致レベル

ODBC ドライバは，そのドライバがサポートする拡張 SQL 文法が COBOL2002 が対応している埋め込み SQL の文法をサポートしている必要があります。COBOL2002 が対応している埋め込み SQL については，マニュアル「COBOL2002　言語　拡張仕様編」「9.　データベースアクセス機能」を参照してください。

23.2.2　コンパイル

SQL 文を埋め込んだ COBOL プログラムをコンパイルするときは，コンパイル時に-SQL,ODBC オプションを指定します。

実行可能ファイルまたは共用ライブラリを作成する場合，リンク時に ODBC ドライバマネージャのライブラリが必要です。ODBC ドライバマネージャのライブラリ指定については，unixODBC に関するリファレンスなどを参照してください。

23.2.3　SQL 文のエラー処理

(1)　SQL 文の実行の確認

SQL 文が正常に実行されたかどうかを確認するには，次の二つの方法があります。

- 埋め込み例外宣言を記述する

  埋め込み例外宣言については，「23.1.1　埋め込み SQL 文を使った COBOL プログラムの作成」の「(3)　埋め込み例外宣言」を参照してください。

- SQLCODE 変数の値を参照する
  SQLCODE 変数については，「(3)　SQLCODE 変数」を参照してください。

(2)　SQL 文でエラーが発生した場合の対処方法

エラーが発生した場合，SQL 文の実行によって出力された実行時メッセージを参照して，対処してください。出力される実行時メッセージのうち，次の項目は ODBC ドライバ，またはデータソースから返される値です。

- SQLSTATE
  ODBC ドライバから返される，ODBC インタフェースで規定された値です。詳細は，ODBC のプログラミングに関するマニュアルを参照してください。
- SQL エラーコード
  データソースが固有に持つエラーコードです。詳細は，ODBC ドライバ，または DBMS のマニュアルなどを参照してください。
- ODBC メッセージ
  ドライバマネージャ，またはデータソースから返されるメッセージです。※

  注※

  実行時メッセージは，環境変数 LANG の設定値に従った文字コードで出力します。このため，ODBC メッセージは，データソースまたはデータベースの設定で，環境変数 LANG の設定値に従った文字コードまたは英文で返されるように設定してください。英文メッセージにする方法などは，ODBC ドライバ提供元のマニュアルなどを参照してください。

(3)　SQLCODE 変数

SQLCODE 変数を定義しておき，その値を参照することで，制御系，および操作系の SQL 文が正しく実行されたか確認できます。

SQLCODE 変数の値とその意味を次に示します。

表 23–2　SQLCODE 変数の値と意味

SQLCODE 変数の値	意味
0	正常に終了した。
100	該当行が存在しない。
100 以外の正の数 （＞ 0 and ≠100）	1. SQLCODE 変数の値が 1 で，警告メッセージが出力されていない場合 外部 10 進項目にデータを設定中に整数部，または小数部のけたあふれが発生した。 2. 1.以外の場合 警告メッセージを出力した。
負の数（＜ 0）	実行は失敗し，エラーメッセージを出力した。

次の環境変数を指定すると SQLCODE 変数に設定される値が変更されます。

(a) 実行時環境変数 CBLSQLROWCOUNT

形式

```
CBLSQLROWCOUNT=YES
```

規則

- この環境変数に YES を指定した場合，埋め込み SQL 文の DELETE 文，INSERT 文，または UPDATE 文を実行して，影響を受けた行数が 0 行の場合に SQLCODE 変数に 100 が設定されます。
- WHENEVER（埋め込み例外宣言）文の NOT FOUND 指定の有効範囲内で実行される埋め込み SQL 文（DELETE 文，INSERT 文，または UPDATE 文）に対して，SQLCODE 変数に 100 が設定されることで，NOT FOUND 条件が実行されます。
- この環境変数を指定しなかったとき，または YES 以外の値を指定したときは，埋め込み SQL 文の DELETE 文，または UPDATE 文の実行で，削除／更新行が 0 行の場合，SQLCODE 変数に 0 が設定されます。ただし，INSERT 文に関しては，注意事項を参照してください。
- この環境変数を指定した場合，影響を受けた行数を取得するために，ODBC ドライバがサポートする SQLRowCount 関数が内部的に発行されるようになります。

注意事項

問い合わせ指定を記述した INSERT 文の実行時，問い合わせ指定によって導出される表が空の場合（挿入行数が 0 行）は，この環境変数の指定がない場合でも SQLCODE 変数に 100 が設定されます。

(4) SQL 文実行時のエラーメッセージ出力抑止

埋め込み SQL 文実行時に出力される実行時メッセージの出力を抑止したい場合は，次の環境変数を指定します。

次の環境変数を指定すると埋め込み SQL 文実行時に実行時メッセージが出力されなくなるので，CBLSQLERROR サービスルーチンを使用して，エラー情報はすべて取得することを推奨します。

(a) 実行時環境変数 CBLSQLWMSG

形式

```
CBLSQLWMSG=YES
```

規則

- 設定値が形式に従ってない場合，この環境変数の指定は無効となります。
- 埋め込み SQL 文実行時に出力される警告メッセージの出力を抑止したいときに YES を指定します。
- この環境変数を指定しても，SQLCODE 値，および WHENEVER（埋め込み例外宣言）文の動作は変わりません。

(b) 実行時環境変数 CBLSQLSUPPRESSMSG

形式

```
CBLSQLSUPPRESSMSG=8002
```

規則

- 設定値が形式に従ってない場合，この環境変数の指定は無効となります。設定値には，8002 だけを指定できます。
- ODBC インタフェースを使ってデータベースにアクセスする場合，KCCC8002R-S メッセージの出力を抑止したいときに 8002 を指定します。
- この環境変数を指定しても，SQLCODE 値，および WHENEVER（埋め込み例外宣言）文の動作は変わりません。

23.2.4 埋め込み変数

SQL 文中で指定する埋め込み変数は，対応する列，または式で有効な ODBC SQL データ型と対応づけられた COBOL データ定義で，定義する必要があります。

ODBC SQL データ型と COBOL のデータ記述の対応を次に示します。

表 23-3 ODBC SQL データ型に対応した COBOL のデータ記述

ODBC SQL データ型	埋め込み変数の型	データ記述
CHAR(n)	文字列型 固定長形式	`Ln データ名　　PIC X(n).`

VARCHAR(n)	文字列型
	可変長形式
		`L1 データ名.`
		`  L2 ODBC-length PIC S9(9) USAGE COMP.`
		`  L2 ODBC-char   PIC  X(n).`

LONG VARCHAR	文字列型
	可変長形式
		`L1 データ名.`
		`  L2 ODBC-length PIC S9(9) USAGE COMP.`
		`  L2 ODBC-char   PIC  X(max).`

DECIMAL(p,s)	数値型	`Ln データ名　　PIC S9(p-s)V9(s)`
		`SIGN IS LEADING`
		`SEPARATE CHARACTER.`

NUMERIC(p,s)	数値型	`Ln データ名　　PIC S9(p-s)V9(s)`
		`SIGN IS LEADING`
		`SEPARATE CHARACTER.`

SMALLINT	整数型符号付き 2 バイト 2 進項目	`Ln データ名　　PIC S9(4) USAGE COMP.`
	または，整数型符号なし 2 バイト 2 進項目	`Ln データ名　　PIC  9(4) USAGE COMP.`

ODBC SQL データ型	埋め込み変数の型	データ記述
INTEGER	整数型符号付き４バイト２進項目 または，整数型符号なし４バイト２進項目	Ln データ名　PIC S9(9) USAGE COMP. Ln データ名　PIC 9(9) USAGE COMP.
REAL	内部浮動小数点型単精度形式	Ln データ名　USAGE COMPUTATIONAL-1.
FLOAT	内部浮動小数点型倍精度形式	Ln データ名　USAGE COMPUTATIONAL-2.
DOUBLE PRECISION	内部浮動小数点型倍精度形式	Ln データ名　USAGE COMPUTATIONAL-2.
BIT	ビット列型	L1 データ名. L2 FILLER　PIC 1(7) USAGE BIT. L2 ODBC-bit　PIC 1(1) USAGE BIT.
TINYINT	整数型符号付き１バイト２進形式 または整数型符号なし１バイト２進形式	Ln データ名　PIC X(1). Ln データ名　PIC S9(2) USAGE COMP.※ Ln データ名　PIC 9(2) USAGE COMP.※
BIGINT	整数型符号付き８バイト２進形式または整数型符号なし８バイト２進形式	Ln データ名　PIC X(20).
BINARY(n)	バイナリデータ型固定長形式	Ln データ名　PIC X(n).
VARBINARY(n)	バイナリデータ型可変長形式	L1 データ名. L2 ODBC-length PIC S9(9) USAGE COMP. L2 ODBC-binary PIC X(n).
LONG VARBINARY	バイナリデータ型可変長形式	L1 データ名. L2 ODBC-length PIC S9(9) USAGE COMP. L2 ODBC-binary PIC X(max).
DATE	日時型日付形式	L1 データ名. L2 ODBC-year　PIC S9(4) USAGE COMP. L2 ODBC-month　PIC 9(4) USAGE COMP. L2 ODBC-day　PIC 9(4) USAGE COMP.
TIME	日時型時刻形式	L1 データ名. L2 ODBC-hour　PIC 9(4) USAGE COMP. L2 ODBC-minute PIC 9(4) USAGE COMP. L2 ODBC-second PIC 9(4) USAGE COMP.
TIMESTAMP	日時型日付／時刻形式	L1 データ名. L2 ODBC-year　PIC S9(4) USAGE COMP. L2 ODBC-month　PIC 9(4) USAGE COMP. L2 ODBC-day　PIC 9(4) USAGE COMP. L2 ODBC-hour　PIC 9(4) USAGE COMP. L2 ODBC-minute PIC 9(4) USAGE COMP. L2 ODBC-second PIC 9(4) USAGE COMP. L2 ODBC-fraction PIC 9(9) USAGE COMP.

（凡例）

Ln：レベル番号 01～49，または 77

L1：レベル番号 01～48

L2：レベル番号 02～49（ただし，L1 ＜ L2）

max：最大データ長

ODBC-xxx：任意のデータ名

データ名：SQL 文中で指定する埋め込み変数名

注

上表以外の SQL データ型には対応していません。

注※

1 バイト 2 進項目は，COMP-X 項目で定義するか，コンパイル時に-Bin1Byte オプションを指定する必要があります。1 バイト 2 進項目の詳細については，マニュアル「COBOL2002　言語　拡張仕様編」「25.6.1　1 バイト 2 進機能」を参照してください。

23.2.5　トランザクション

(1)　トランザクション分離レベル

あるデータベース上のデータに対して複数のユーザから同時にアクセスする場合，トランザクションはそれぞれの接続ごとにドライバによって管理され，ドライバのトランザクション分離レベルに従って，トランザクションが処理されます。

トランザクション分離レベルは，ドライバ，またはデータソースのデフォルトに従います。詳細は，ODBC ドライバ提供元のマニュアルなどを参照してください。

23.2.6　コネクション

コネクションは，CONNECT 文によって接続を確立します。CONNECT 文で指定する接続先データベース名は，データソース名を指定します。

データソースは ODBC ドライバマネージャ経由でアクセスできるように登録されていなければなりません。データソース名を省略したときは，デフォルトデータソース（default）を仮定します。

CONNECT 文で指定する接続文字列については，ODBC ドライバ提供元のマニュアルなどを参照してください。

（例）

次の CONNECT 文は，同じデータソースに対しての接続を要求します。

- データソース名を使用する方法

```
MOVE 'USER1' TO ODBC-UID.
MOVE 'USER1' TO ODBC-PWD.
MOVE 'SAMPLE' TO ODBC-DSN.
```

```
EXEC SQL
  CONNECT :ODBC-UID
    IDENTIFIED BY :ODBC-PWD
    USING :ODBC-DSN
END-EXEC.
```

- 接続文字列を使用する方法

```
MOVE 'DSN=SAMPLE;UID=USER1;PWD=USER1;'
  TO ODBC-CONN.
EXEC SQL
  CONNECT TO :ODBC-CONN
END-EXEC.
```

23.2.7　タイムアウト秒数の設定

ODBC インタフェースを使って埋め込み SQL 文実行時に，実行時環境変数に指定したタイムアウト秒数で ODBC オプションを設定できます。この機能を使用しない場合，タイムアウト秒数は設定しません。

(1)　CONNECT 文でのタイムアウト秒数の設定

形式

```
CBLSQLLOGINTIMEOUT=タイムアウト秒数
```

タイムアウト秒数には，0～2,147,483,647 の符号なし整数を指定します。ただし，タイムアウトの最大値は，使用する ODBC ドライバによって規定されます。

なお，タイムアウト秒数に 0 を指定した場合，タイムアウトは発生しないので，永久に待ち状態となります。

規則

- 設定値が形式に従ってない場合，この環境変数の指定は無効となります。
- ODBC インタフェースを使って埋め込み SQL 文（CONNECT）実行時に，環境変数に設定されたタイムアウト秒数を，SQLSetConnectOption 関数で SQL_LOGIN_TIMEOUT オプションに設定します。

(2)　埋め込み SQL 文（操作系）でのタイムアウト秒数の設定

形式

```
CBLSQLQUERYTIMEOUT=タイムアウト秒数
```

タイムアウト秒数には，0～2,147,483,647 の符号なし整数を指定します。ただし，タイムアウトの最大値は，使用する ODBC ドライバによって規定されます。

なお，タイムアウト秒数に 0 を指定した場合は，タイムアウトは発生しないので，永久に待ち状態となります。

規則

- 設定値が形式に従ってない場合，この環境変数の指定は無効となります。
- ODBC インタフェースを使って埋め込み SQL 文（操作系）実行時に，環境変数で設定されたタイムアウト秒数を，SQLSetStmtOption 関数で SQL_QUERY_TIMEOUT オプションに設定します。

23.2.8 カーソルオプションの設定

ODBC インタフェース機能では，ODBC カーソルライブラリの静的カーソルを使用するように次の ODBC のオプションを設定して SQL のカーソル処理を実現しています。

表 23–4 ODBC のオプションに対応する SQL のカーソル処理

オプション名	設定値	機能
接続オプション	SQLSetConnectOption（hdbc,SQL_ODBC_CURSORS,SQL_CUR_USE_ODBC）；	ドライバマネージャは，ドライバのスクロール機能を使用します。
ステートメントオプション	SQLSetStmtOption（hstmt,SQL_CONCURRENCY,SQL_CONCUR_VALUES）；	カーソルは最適化同時実行制御を使用して値を比較します。
	SQLSetStmtOption（hstmt,SQL_CURSOR_TYPE,SQL_SCROLL_STATIC）；	結果セットのデータは静的なデータとなります。

ODBC ドライバによって，ODBC カーソルライブラリの使用をサポートしていないことがあります。使用する ODBC ドライバが，これらの接続オプションおよびステートメントオプションをサポートしているかどうかを確認してください。

ODBC カーソルライブラリを使用しない場合は，実行時環境変数 CBLSQLCURUSE を指定してください。

(1) 実行時環境変数 CBLSQLCURUSE

形式

```
CBLSQLCURUSE=DYNAMIC
```

規則

DYNAMIC

ODBC ドライバのスクロール機能の動的カーソルを使用するように設定します。この実行時環境変数の指定がない場合は，ODBC カーソルライブラリの静的カーソルを使用するように ODBC オプションを設定します。DYNAMIC 以外の値を指定した場合の結果は，保証しません。

DYNAMIC を指定した場合，次の太字のオプションがデフォルトから変更されます。

表 23-5　DYNAMIC を指定した場合の設定値と機能

オプション名	設定値	機能
接続オプション	SQLSetConnectOption（hdbc,SQL_ODBC_CURSORS,SQL_CUR_USE_DRIVER）；	ドライバマネージャは，ドライバのスクロール機能を使用します。
ステートメントオプション	SQLSetStmtOption（hstmt,SQL_CONCURRENCY,SQL_CONCUR_VALUES）；	カーソルは最適化同時実行制御を使用して値を比較します。
	SQLSetStmtOption（hstmt,SQL_CURSOR_TYPE,SQL_SCROLL_DYNAMIC）；	ドライバは行セットの行に対するキーだけを保存して使用します。

(2)　注意事項

この環境変数の指定で設定する接続オプションまたはステートメントオプションを ODBC ドライバがサポートしていない場合，動作は保証しません。なお，使用する ODBC ドライバによって，暗黙的にオプションが変更されることがあります。使用する ODBC ドライバが，この環境変数の指定で設定する接続オプションおよびステートメントオプションをサポートしているかどうかは，ODBC ドライバ提供元のマニュアルなどを参照してください。

23.2.9　データベース固有の注意事項

HiRDB のデータベースに接続する場合，COBOL プログラム実行中に環境変数 PDDDLDEAPRPEXE を設定した環境で定義系トランザクションを実行しないでください。環境変数 PDDDLDEAPRPEXE を設定した環境で定義系トランザクションを実行すると，先行トランザクションの前処理が無効となるため，COBOL の管理情報との不整合が発生し，予期しない実行時エラーが発生するおそれがあります。

環境変数 PDDDLDEAPRPEXE の詳細は，「HiRDB の UAP 開発ガイドマニュアル」を参照してください。

動的 SQL で定義系のクエリを指定して実行しているトランザクションと，動的 SQL で操作系のクエリ，または静的に行う方法やカーソル，ストアドプロシージャ呼び出しの埋め込み SQL 文を実行しているトランザクションは，別々のトランザクションとしてください。一つのトランザクション内で混在させた場合の動作は保証しません。

表を参照，または更新している場合※に，その表の定義を定義系のクエリやほかのプログラムやツールなどを使用して変更したときの動作は保証しません。参照，または更新している表の定義を変更した場合，記述された埋め込み SQL 文との不整合が発生し，予期しない実行時エラーが発生することがあります。

注※

表を参照，更新しているときとは，次の期間です。

- 埋め込み SQL 文を実行し，COMMIT 文または ROLLBACK 文でトランザクションを終了するまでの間
- PREPARE 文で準備した SQL 文を明示的に DEALLOCATE PREPARE 文で解放するまでの間

埋め込み SQL 文内では，変数名以外の表名や列名などは必ず ASCII 文字の範囲で定義してください。検索条件などで多バイト文字を含む定数と比較したいときは，定数を埋め込み変数に転記したあと，その埋め込み変数を使って比較してください。

23.2.10　動的 SQL の ODBC API 関数発行の変更

ODBC インタフェース機能の動的 SQL で内部的に発行する ODBC API を変更したい場合に，実行時環境変数 CBLSQLDYNAMIC を指定します。

実行時環境変数 CBLSQLDYNAMIC を指定すると，実行性能が改善することがあります。

形式

```
CBLSQLDYNAMIC=REUSEBINDINFO
```

規則

- 設定値が形式に従ってない場合，この環境変数の指定は無効です。
- 環境変数 CBLSQLDYNAMIC に REUSEBINDINFO を指定すると，埋め込み SQL 文で内部的に発行する ODBC API を次のとおりに変更し，1 回目の埋め込み SQL 文実行時に取得した情報で 2 回目以降も実行します。

・次の埋め込み SQL 文を連続で実行した場合

2 回目以降の実行では，1 回目に取得した情報をバインド処理で再利用します。※1

- EXECUTE 文
- 動的 SQL をカーソルで実行する場合の OPEN 文
- 動的 SQL をカーソルで実行する場合の OPEN 文に対する最初の FETCH 文

・PREPARE 文で準備した動的 SQL が，INSERT 文，DELETE 文，または UPDATE 文の場合

埋め込み SQL 文の EXECUTE 文を連続で実行したときに，2 回目以降の実行では 1 回目に取得した ODBC ハンドル情報を再利用します。※2

注※1

2 回目以降の埋め込み SQL 文の実行で，内部的に SQLDescribeParam，SQLDescribeCol，SQLNumResultCols を呼び出さない（パラメタ，列情報を取得しない）ようにすることで，1 回目の埋め込み SQL 文の実行で取得したパラメタ，列情報を再利用します。

注※2

埋め込み SQL 文の EXECUTE 文の実行で，内部的に SQLFreeStmt を SQL_CLOSE パラメタ指定で呼び出さない（ODBC ハンドル情報をクローズしない）ようにすることで，1 回目の埋め込み SQL 文の EXECUTE 文の実行で取得した ODBC ハンドル情報を再利用します。

24 XDM によるデータベース操作シミュレーション機能

この章では，XDM によるデータベース操作シミュレーション機能について説明します。

24.1　データベース操作シミュレーションの概要

データベース操作シミュレーション機能とは，メインフレーム（VOS3）COBOL85 の環境で作成した XDM によるデータベース操作を行うプログラムを，UNIX COBOL2002 上でシミュレーションする機能です。

データベース操作シミュレーション機能には次の二つの機能があります。

- 構造型データベース操作シミュレーション機能

  メインフレームで構造型データベース XDM/SD（Extensible Data Manager／Structured Database）を操作するプログラムを，UNIX 上でコンパイルし，実行する機能です。

  詳細は，「24.2　構造型データベース（XDM/SD）操作シミュレーション」を参照してください。

- リレーショナルデータベース操作シミュレーション機能

  メインフレームでリレーショナルデータベース XDM/RD（Extensible Data Manager／Relational Database）を操作するプログラムを，SQL 文を覚え書きとしてコンパイルし，テストデバッガの TD コマンド（ASSIGN DATA コマンドなど）を使用してテストする機能です。

  詳細は，「24.3　リレーショナルデータベース（XDM/RD）操作シミュレーション」を参照してください。

これらのシミュレーション機能の使い方について説明します。

なお，データベース操作シミュレーション機能の文法規則については，マニュアル「COBOL2002　言語　拡張仕様編」「19.　データベース操作シミュレーション機能」を参照してください。

24.2　構造型データベース（XDM/SD）操作シミュレーション

メインフレームで構造型データベース XDM/SD（Extensible Data Manager／Structured Database）を操作するプログラムを，UNIX 上でコンパイル，実行できます。このプログラムのコンパイル，実行，テストの方法について説明します。

24.2.1　コンパイル方法

プログラムのコンパイルまでの手順を次に示します。

1. データベース情報を COBOL 宣言文に展開する。

   メインフレーム上で XDM のユティリティを使用し，データベース情報であるスキーマ情報，サブスキーマ情報を COBOL 宣言文に展開します。このとき，COBOL 宣言文（COPY 文で展開される原文）は，XDM E2 系ユティリティ JXBSAID を使用して作成します。

   (COBOL 宣言文の展開例)

```
000100*****************************************************************
000200*                                                               *
000300*       SUBSCHEMA NAME = HSB1                                   *
000400*       SCHEMA NAME    = HSC1                                   *
000500*       CREATE TIME    = YY-MM-DD HH:MM:SS                      *
000600*       VERSION        = CURRENT                                *
000700*       RUN TIME       = YY-MM-DD HH:MM:SS                      *
000800*                                                               *
000900*****************************************************************
001000 01  REC01.
001100   02  CT01.
001200     03  CT01-01        PIC S9(10)       DISPLAY.
001300     03  CT01-02        PIC S9(4)        DISPLAY.
001400     03  CT01-03        PIC X(88)        DISPLAY.
001500 01  REC02.
001600   02  CT02.
001700     03  CT02-01        PIC S9(4)        DISPLAY.
001800     03  CT02-02        PIC X(236)       DISPLAY.
```

2. COBOL 宣言文を UNIX に転送する。

   作成した COBOL 宣言文を，ファイル転送プログラムを使用して UNIX に転送します。このとき，UNIX 上で受け取るファイル名は"サブスキーマ名スキーマ名.cbl"（上記の例では，HSB1HSC1.cbl）とする必要があります。

3. プログラムをコンパイルする。

   UNIX に転送された COBOL 宣言文（COPY 展開される原文）は，コンパイル時の SUBSCHEMA SECTION 解析中に COPY 文と同様に展開されます。このため，COBOL 宣言文の入ったファイルは，登録集原文が展開できるディレクトリ下に置く必要があります。

24.2.2　実行方法

手続き部に記述した FIND 文，FETCH 文などのデータベース操作文は，"CALL 'CBLXDMSD' USING 引数 ……"と内部的に展開されます。展開された CALL 文で呼び出されるプログラムでシミュレーションをする場合は，'CBLXDMSD'という名称の関数を作成し，コンパイル，リンクして実行可能ファイルを生成しておきます。この結果，手続き部のデータベース操作文は，'CBLXDMSD'で作成した関数を呼び出せます。

なお，'CBLXDMSD'という名称の関数を，コンパイル，リンクして実行可能ファイルを生成しないときは，コンパイラが提供する，構造型データベース（XDM/SD）操作シミュレーション機能の実行時ライブラリ用ダミールーチン CBLXDMSD が実行されます。

24.2.3　内部的に展開される CALL 文の引数

CALL 文に展開されたデータベース操作文の引数によって，利用者は XDM/SD プログラムをテストできます。内部的に展開される CALL 文の引数の形式を次に示します。

形式

```
CALL ’CBLXDMSD’ USING インタフェースエリア
   DML情報エリア 固有情報エリア（その他の引数）…
```

データベース操作文で内部的に展開される CALL 文を次に示します。

表 24–1　データベース操作文で内部的に展開される CALL 文

データベース操作文	内部的に展開される CALL 文と設定される引数
データベース条件文	CALL ’CBLXDMSD’ USING インタフェースエリア ,DML情報エリア ,固有情報エリア
CONNECT 文 DISCONNECT 文 ERASE 文 NULLIFY 文 RECONNECT 文	CALL ’CBLXDMSD’ USING インタフェースエリア ,DML情報エリア ,固有情報エリア
FETCH 文※1 FIND 文 GET 文※1 MODIFY 文 STORE 文	CALL ’CBLXDMSD’ USING インタフェースエリア ,DML情報エリア ,固有情報エリア ,データ受け渡しエリア※2 〔,付加機能用情報エリア〕※3
FETCH 文※4	CALL ’CBLXDMSD’ USING インタフェースエリア ,DML情報エリア ,固有情報エリア ,データ受け渡しエリア※5

データベース操作文	内部的に展開される CALL 文と設定される引数
GET 文※4	CALL 'CBLXDMSD' USING インタフェースエリア ,DML情報エリア ,固有情報エリア ,データ受け渡しエリア※2 ,データ受け渡しエリア※5

注※1　DATA AREA 指定がありません。

注※2

FROM，INTO で指定したデータ名です。指定しない場合にはサブスキーマ節の先頭のレコードビュー名が使用されます。

注※3

FIND 文，DATA AREA 指定なしの FETCH 文で USING を指定したときのデータ名です。〔　〕で囲まれた引数を指定しない場合，引数は BY REFERENCE 指定で ZERO（4 バイトの 0）を渡します。

注※4　DATA AREA 指定があります。

注※5　DATA AREA で指定したデータ名です。

データベース操作文で内部的に展開される CALL 文の引数を「表 24-2　データベース操作文で内部的に展開される CALL 文の引数」に示します。また，各項目の詳細を次の表に示します。

- 表 24-3　インタフェースエリアの詳細
- 表 24-4　DML 情報エリアの詳細
- 表 24-5　固有情報エリアの詳細

表 24-2　データベース操作文で内部的に展開される CALL 文の引数

引数の項目	設定内容
インタフェースエリア	01 SD-IFA. 02 FILLER　　PIC X(4). 02 SD-IFA-01 PIC X(5).　…… 実行結果を表す状態コード 02 SD-IFA-02 PIC X(1).　…… データベース条件の実行結果 02 SD-IFA-03 PIC X(30). …… 検索したレコードビュー名 02 FILLER　　PIC X(30). 02 SD-IFA-04 PIC X(2).　…… 使用しない 02 SD-IFA-05 PIC X(2).　…… 使用しない 02 FILLER　　PIC X(26). 02 SD-IFA-06 PIC X(20). …… 使用しない 02 FILLER　　PIC X(12). 02 SD-IFA-07 PIC S9(4) COMP.　… カーソルグループ番号 02 SD-IFA-08 PIC X(4).　…… サブ状態コード 02 SD-IFA-09 PIC S9(4) COMP.　… レコードビュー長 02 FILLER　　PIC X(20). 02 SD-IFA-10 PIC X(30). …… データベース操作文種別 02 FILLER　　PIC X(66). 02 SD-IFA-11 PIC X(30). …… サブスキーマ名 02 SD-IFA-12 PIC X(30). …… スキーマ名 02 FILLER　　PIC X(196).
DML 情報エリア	01 SD-DML. 02 SD-DML-01　PIC S9(4) COMP.　… DML情報エリア長 02 FILLER　　PIC X(2). 02 SD-DML-02 PIC X(2).　…… DML区分 02 SD-DML-03 PIC X(2).　…… 要求種別 02 SD-DML-04 PIC X(2).　…… 一括検索範囲指定 02 SD-DML-05 PIC X(2).　…… 位置指示子指定 02 SD-DML-06 PIC X(30). …… ビュー名

引数の項目	設定内容
	02 FILLER PIC X(2). 02 SD-DML-07 PIC X(30). …… インデクス名／ 親子集合ビュー名 02 FILLER PIC X(2). 02 SD-DML-08 PIC X(2). …… 探索方向 02 SD-DML-09 PIC X(2). …… 位置決め目的 02 SD-DML-10 PIC S9(9) COMP. … 探索方向の整数値 02 SD-DML-11 PIC X(30). …… パス名 02 FILLER PIC X(14).
固有情報エリア (手続き文の各指定情報)	01 SD-TYP. 02 SD-TYP-01 PIC X(2). …… データ名指定情報 02 FILLER PIC X(6). 02 SD-TYP-02 PIC X(2). …… 条件種別 02 FILLER PIC X(2). 02 SD-TYP-03 PIC X(2). …… 左辺指定子 02 SD-TYP-04 PIC X(30). …… 左辺指定子ビュー名 02 SD-TYP-05 PIC X(2). …… 右辺指定子 02 SD-TYP-06 PIC X(30). …… 右辺指定子ビュー名 02 SD-TYP-07 PIC X(2). …… 検索条件の比較演算子 02 SD-TYP-08 PIC X(30). …… 検索条件の構成要素 ビュー名 02 FILLER PIC X(32). 02 SD-TYP-09 PIC X(1). …… RETAINING種別 02 FILLER PIC X(1). 02 SD-TYP-10 PIC S9(4) COMP. … 親子集合ビュー数 02 FILLER OCCURS 2000 DEPENDING ON SD-TYP-10. 03 SD-TYP-11 PIC X(30). …… 親子集合ビュー名 03 FILLER PIC X(2).
データ受け渡しエリアおよび付加機能用情報エリア	データ名

表 24-3　インタフェースエリアの詳細

領域名	CALL 文 実行前	シミュレータ	CALL 文 実行後	内容
SD-IFA-01	－	設定	参照	実行結果を表す。 STATUS 句で指定したデータ名の領域に転記される。
SD-IFA-02	－	設定	参照	データベース条件の結果を表し，データベース条件の判定に使用する。 '1'：判定結果は真 '0'：判定結果は偽
SD-IFA-03	－	設定	参照	検索したレコードビュー名を表す。 RECORD NAME 句に指定したデータ名の領域に転記される。RECORD NAME 句の指定がない場合は転記されない。
SD-IFA-04	－	－	－	使用しない。
SD-IFA-05	－	－	－	使用しない。
SD-IFA-06	－	－	－	使用しない。
SD-IFA-07	設定	参照	－	カーソルグループ番号を表す。

領域名	CALL 文 実行前	シミュ レータ	CALL 文 実行後	内容
				CURSOR NUMBER 句に指定したデータ名の領域から転記される。 CURSOR NUMBER 句の指定がない場合は 0 が転記される。
SD-IFA-08	－	設定	参照	サブ状態コードを表す。 DETAIL CODE 句で指定したデータ名の領域に転記される。 DETAIL CODE 句の指定がない場合は転記されない。
SD-IFA-09	－	設定	参照	処理対象のレコードビュー長を表す。 RECORD LENGTH 句で指定したデータ名の領域に転記される。 RECORD LENGTH 句の指定がない場合は転記されない。
SD-IFA-10	設定	参照	－	データベース操作文の種別を表す。 'CONNECT' = CONNECT 文 'DISCONNECT' = DISCONNECT 文 'ERASE' = ERASE 文 'FETCH' = FETCH 文 'FIND' = FIND 文 'GET' = GET 文 'MODIFY' = MODIFY 文 'NULLIFY' = NULLIFY 文 'RECONNECT' = RECONNECT 文 'STORE' = STORE 文 'TEST' = データベース条件文
SD-IFA-11	設定	参照	－	サブスキーマを表す。 サブスキーマが転記される。
SD-IFA-12	設定	参照	－	スキーマを表す。 スキーマが転記される。

（凡例）

－：該当しない

データ受け渡しエリアおよび付加情報エリアでは，各文中に指定されたデータ名が引数になります。指定できるデータ名を次に示します。

- FETCH 文の DATA AREA または INTO に指定されたデータ名
- GET 文の DATA AREA または INTO に指定されたデータ名
- MODIFY 文の FROM に指定されたデータ名
- STORE 文の FROM に指定されたデータ名
- FETCH 文または FIND 文の USING に指定されたデータ名

表 24–4　DML 情報エリアの詳細

領域名	CALL 文 実行前	シミュ レータ	CALL 文 実行後	内容
SD-DML-01	設定	−	−	DML 情報エリア長を表す。 128 の固定長が転記される。
SD-DML-02	−	−	−	使用しない。
SD-DML-03	設定	参照	−	要求種別を表す。 'P' ＝ ERASE 文で ALL 指定なし。 'R' ＝ FETCH(1)文で WITHIN，INDEXED の指定なし。 または，FIND 文で CURRENT，WITHIN，INDEXED の指定なし。 'RI'＝ FETCH(1)，FIND 文で INDEXED 指定あり。 'S' ＝ FETCH(1)，FIND 文で WITHIN 指定あり。 'C' ＝ FIND 文で CURRENT 指定あり。 'H' ＝ FETCH(2)文で WITHIN，INDEXED の指定なし。 'HI'＝ FETCH(2)文で INDEXED 指定あり。 'M' ＝ FETCH(2)文で WITHIN 指定あり。 'N' ＝ GET(2)文。
SD-DML-04	設定	参照	−	一括検索範囲指定を表す。 'A' ＝ FETCH 文で ADVANCING 指定あり。
SD-DML-05	設定	参照	−	位置指示子指定を表す。 'S' ＝ NULLIFY 文でレコードビュー名，OWNER，MEMBER の指定なし。 または，CURRENT 指定ありの FIND 文で，レコードビュー名，OWNER，MEMBER の指定なし。 'R' ＝ NULLIFY 文でレコードビュー名の指定あり。 または，FIND 文で CURRENT　レコードビュー名の指定あり。 'O' ＝ NULLIFY 文で OWNER OF　親子集合ビュー名の指定あり。 または，FIND 文で CURRENT OWNER OF　親子集合ビュー名の指定あり。 'M' ＝ NULLIFY 文で MEMBER OF　親子集合ビュー名の指定あり。 または，FIND 文で CURRENT MEMBER OF　親子集合ビュー名の指定あり。
SD-DML-06	設定	参照	−	次のレコードビュー名または親子集合ビュー名を表す。 • CONNECT，DISCONNECT，ERASE，FETCH，GET，MODIFY，RECONNECT，STORE 文で指定されたレコードビュー名 • FIND 文で CURRENT 指定がない場合のレコードビュー名

領域名	CALL 文 実行前	シミュ レータ	CALL 文 実行後	内容
				• FIND，NULLIFY 文で CURRENT に指定されたレコードビュー名または親子集合ビュー名
SD-DML-07	設定	参照	−	次のインデクス名または親子集合ビュー名を表す。 • FETCH，FIND 文で INDEXED 指定がある場合のインデクス名 • FETCH，FIND，GET 文で WITHIN 指定がある場合の親子集合ビュー名 • CONNECT，DISCONNECT，RECONNECT 文で指定された先頭の親子集合ビュー名
SD-DML-08	設定	参照	−	探索方向を表す。 'F' = FETCH，FIND 文で FIRST 指定あり。 'L' = 〃 LAST 指定あり。 'N' = 〃 NEXT 指定あり。 'P' = 〃 PRIOR 指定あり。 'A' = 〃 RELATIVE 指定なし。 'R' = 〃 RELATIVE 指定あり。 'D' = 〃 DYNAMIC 指定あり。
SD-DML-09	設定	参照	−	位置決め目的を表す。 'U' = FETCH，FIND 文で UPDATE 指定あり。
SD-DML-10	設定	参照	−	探索方向の整数値を表す。 RELATIVE 指定時，未指定時の一意名または整数の値。
SD-DML-11	設定	参照	−	パス名を表す。 FETCH 文に指定されたパス名。

（凡例）

−：該当しない

FETCH(1)文，FETCH(2)文，GET(2)文は，それぞれ次の文を指します。

- FETCH(1)文：DATA AREA 指定なしの FETCH 文
- FETCH(2)文：DATA AREA 指定ありの FETCH 文
- GET(2)文：DATA AREA 指定ありの GET 文

表 24-5　固有情報エリアの詳細

領域名	CALL 文 実行前	シミュ レータ	CALL 文 実行後	内容
SD-TYP-01	設定	参照	−	データ名指定を表す。 'U' = USING 指定あり。
SD-TYP-02	設定	参照	−	条件種別を表す。 'A' = データベース条件文の ALSO 指定

領域名	CALL 文 実行前	シミュ レータ	CALL 文 実行後	内容
				'NA'= 〃 NOT ALSO 指定 'N' = 〃 NULL 指定 'NN'= 〃 NOT NULL 指定 'E' = 〃 EMPTY 指定 'NE'= 〃 NOT EMPTY 指定 'C' = 〃 CONTAINS 指定 空白 = 上記以外
SD-TYP-03	設定	参照	−	左辺指定子を表す。 'R' = データベース条件左辺にレコードビュー名を指定 'O' = 〃 OWNER OF　親子集合ビュー名を指定 'M' = 〃 MEMBER OF　親子集合ビュー名を指定 'S' = 〃 レコードビュー名，OWNER，MEMBER のどの指定もなし 'X' = 〃 親子集合ビュー名を指定 空白 =上記以外
SD-TYP-04	設定	参照	−	データベース条件左辺のレコードビュー名または親子集合ビュー名を表す（SD-TYP-03 が 'S'または空白のときは，この領域も空白となる）。
SD-TYP-05	設定	参照	−	右辺指定子を表す。 'R' = データベース条件右辺にレコードビュー名を指定 'O' = 〃 OWNER OF　親子集合ビュー名を指定 'M' = 〃 MEMBER OF　親子集合ビュー名を指定 'S' = 〃 レコードビュー名，OWNER，MEMBER のどの指定もなし 'X' = 〃 親子集合ビュー名を指定 空白 =上記以外
SD-TYP-06	設定	参照	−	データベース条件右辺のレコードビュー名または親子集合ビュー名を表す（SD-TYP-05 が 'S'または空白のときは，この領域も空白となる）。
SD-TYP-07	設定	参照	−	探索条件の最初に現れた比較演算子を表す。 '>' = GREATER THAN　または > 指定 '<' = LESS THAN　または < 指定 '=' = EQUAL　または = 指定 '>='= GREATER THAN OR EQUAL または >= 指定 '<='= LESS THAN OR EQUAL　または <= 指定 空白= 上記以外
SD-TYP-08	設定	参照	−	探索条件の最初に現れた構成要素ビュー名または KEY を表す（SD-TYP-07 が空白のときは，この領域も空白となる）。
SD-TYP-09	設定	参照	−	RETAINING 種別を表す。

領域名	CALL 文 実行前	シミュ レータ	CALL 文 実行後	内容
				'A' = RETAINING 指定ありの FETCH，FIND，STORE 文で，RECORD，親子集合ビュー名の指定がない。 'R' = ERASE 文で RETAINING 指定あり。 または，RETAINING RECORD 指定ありの FETCH，FIND，STORE 文で，親子集合ビュー名の指定がない。 'S' = RETAINING　親子集合ビュー名指定ありの FETCH，FIND，STORE 文で，RECORD の指定がない。 'B' = FETCH，FIND，STORE 文に，RETAINING RECORD　親子集合ビュー名の指定がある。
SD-TYP-10	設定	参照	－	RETAINING に指定された親子集合ビューの個数を表す。親子集合ビュー名の指定があれば，その個数が入る。
SD-TYP-11	設定	参照	－	RETAINING に指定された親子集合ビュー名を表す（SD-TYP-10 が 0 の場合，この領域はない）。

（凡例）

－：該当しない

注

SD-TYP-02～06 は，データベース条件文情報です。

SD-TYP-07，08 は，探索条件情報です。

SD-TYP-09～11 は，RETAINING 情報です。

24.2.4　XDM/SD プログラムのテスト方法の制限事項

COBOL 原始プログラムのコンパイル時，次の部分はエラーチェックの対象となりません。

(1)　XDM システム定義のスキーマ定義にだけ指定され，サブスキーマ節に展開されない部分

- 親子集合で親レコード，子レコードの対応関係があるもの

  例えば，親子集合型内を検索する FETCH 文で，レコードビュー名で示すレコードは，親子集合ビュー名で示す親子集合の子レコードにしなければなりません。

- パスで指定したレコードの関係にあるもの

  例えば，パス順に複数のレコードを検索する FETCH 文で，エントリレコード名で示すレコードは，パス名で示すパスのエントリレコードにしなければなりません。

- 副構成要素に関するもの

  例えば，条件を指定した FETCH 文で，比較条件に指定した構成要素ビュー名で示す構成要素は，副構成要素以外にしなければなりません。

(2) XDM システム定義のサブスキーマ定義にだけ指定され，サブスキーマ節に展開されない部分

- アクセス目的に関するもの

  例えば，FOR UPDATE（更新）を指定した FETCH 文で，レコードビュー名で示すレコードは，サブスキーマ定義で UPDATE 指定をしなければなりません。

24.3.1 コンパイル方法

リレーショナルデータベース（XDM/RD）操作シミュレーション機能を使用する場合，-SQL,XDM オプションを指定してコンパイルします。また，SQL 連絡領域 SQLCA は，メインフレームの XDM/RD が提供している SQL 連絡領域を SQLCA.cbl という名称で UNIX 上に転送して使用します。このシステムでは，リレーショナルデータベース（XDM/RD）操作シミュレーション機能の実行時ライブラリ用ダミールーチンとして，CBLXDMRD を提供します。

24.3.2 テスト方法

実行可能ファイルでは SQL 文は実行に関係しないため，SQL 文のテストはできません。SQL 以外の文のテストをする場合は，テストデバッガの TD コマンド（ASSIGN DATA コマンドなど）を用いて次のようにテストします。

（例）

行番号 3000～3200 の SQL 文の擬似実行で，NAME-AREA にデータを代入し，行番号 3300 の IF 文の THEN の処理と ELSE の処理をテストする。

（原始プログラム）

```
001000 WORKING-STORAGE SECTION.
001100 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
001200 01 NAME-AREA PIC N(20).
001300 EXEC SQL END DECLARE SECTION END-EXEC.
           :
002000 PROCEDURE DIVISION.
           :
003000     EXEC SQL
003100       SELECT NAME INTO :NAME-AREA FROM SHAIN
                              WHERE AGE = 30
003200     END-EXEC.
003300     IF NAME-AREA = N'山田　太郎'
003400       THEN
               :
004000       ELSE
               :
005000     END-IF.
```

（入力する TD コマンド）

```
SET BREAK STATEMENT(3000) COUNTER(CNT) DO
  IF CONDITION(CNT=1)
    ASSIGN DATA(NAME-AREA) VALUE(N'山田　太郎')
  ELSE
    ASSIGN DATA(NAME-AREA) VALUE(N'日立　花子')
  ENDIF
```

```
  GO
ENDDO
```

1. 行番号 3000 の行に制御が渡るごとに実行を中断します。
2. 最初の中断のときは"山田　太郎"を，2 回目以降の中断のときは"日立　花子"を NAME-AREA に設定します。
3. NAME-AREA に値を設定後，実行を再開始します。

25 CGI プログラム作成支援機能（AIX(32)で有効）

CGI（Common Gateway Interface）プログラム作成支援機能を使うと，CGI を使ったプログラムを COBOL2002 で作成できます。この章では，CGI プログラムの作成を支援する機能と，その利用方法について説明します。

25.1　CGI プログラム作成支援機能の概要

25.1.1　概要

CGI（Common Gateway Interface）とは，Web ページで，画面対話のような複雑なサービスを使用できるようにするために用意された拡張 API のことです。

CGI プログラム作成支援機能は，COBOL2002 で作成したプログラムを CGI プログラムとして利用するための機能です。

25.1.2　CGI プログラムが動作するのに必要な環境

CGI プログラムを利用するには，サーバ，クライアントがそれぞれ次の条件を満たしている必要があります。

サーバ

CGI プログラムが動作する Web サーバが必要です。

クライアント

Web ページを見るための Web ブラウザが必要です。

25.1.3　CGI プログラム作成支援機能が提供する機能

CGI プログラム作成支援機能には，次のような機能があります。

(1)　CGI プログラムの作成を支援するサービスルーチン

次のような機能を，サービスルーチンを使って利用できます。

- 標準 HTML ヘッダの出力
- システム環境変数へのアクセス
- フォーム情報の解析
- HTML コードの自動生成
- デバッグ

詳細は「25.7　CGI プログラムの作成を支援するサービスルーチン」を参照してください。

(2) HTML トランスレータ

HTML トランスレータは，HTML ファイルを COBOL 副プログラムのソースファイルに変換するツールです。HTML トランスレータを使うと，COBOL をプログラミングすることなく，HTML ファイルから COBOL 副プログラムが作成できます。

詳細は「25.5 HTML ファイルを COBOL ソースファイルに変換する方法」を参照してください。

(3) HTML テンプレート機能

HTML テンプレートとは，HTML ファイル中に HTML 拡張言語と呼ばれる拡張タグを埋め込んだものです。HTML テンプレートを利用すると，動的な Web ページを出力する COBOL プログラムを効率良く開発できます。

詳細は「25.6 HTML テンプレート機能」を参照してください。

(4) 注意事項

- COBOL2002 で作成した CGI プログラムは，サーバの高速化や ISAPI，NSAPI などの拡張 CGI 機能を利用できません。
- CGI プログラムの環境変数の設定，および CGI プログラムを起動する起動ファイルは，セキュリティなどの理由からサーバ管理者に起動を制限されている場合があります。
- CGI プログラムのコンパイル時と実行時の環境変数 LANG は，同じでなければなりません。

25.2　CGI プログラムの種類と作成方法

CGI プログラムは，プログラム定義として作成します。

CGI プログラムの作成には，3 種類の方法があります。ここでは，それぞれの作成方法と，作成されたプログラムの特徴について説明します。

25.2.1　スタティック型 CGI プログラムの作成方法

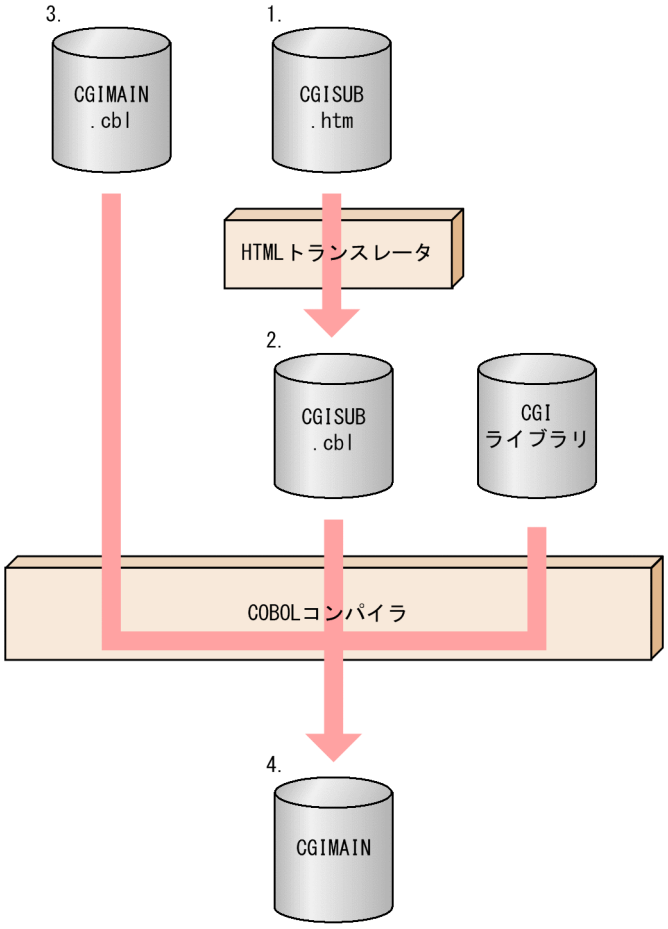
スタティック型 CGI プログラムは，あらかじめ HTML トランスレータを使って，HTML テンプレートから COBOL 副プログラムを生成しておく形式のプログラムです。

スタティック型 CGI プログラムは，実行速度が高速です。しかし，HTML テンプレートを変更するたびに再コンパイルする必要があります。

スタティック型 CGI プログラムの作成手順

1. Web ページに出力する HTML テンプレート（CGISUB.htm）を作成します。
2. HTML トランスレータを使用して，HTML テンプレートから COBOL 副プログラム（CGISUB.cbl）を生成します。
3. 2.で作成した COBOL 副プログラムを呼び出す COBOL 主プログラム（CGIMAIN.cbl）を作成します。
4. COBOL 主プログラム，COBOL 副プログラムをコンパイルし，CGI プログラム作成支援ライブラリ（CGI ライブラリ）とリンクして，CGI プログラム（CGIMAIN）を生成します。

図 25-1　スタティック型 CGI プログラムの作成手順



25.2.2　インタプリット型 CGI プログラムの作成方法

インタプリット型 CGI プログラムは，COBOL プログラムが実行中に CBLFILLTEMPLATE サービスルーチンを利用して，HTML テンプレートをインタプリット（動的変換）して出力する形式のプログラムです。

インタプリット型 CGI プログラムは，HTML テンプレートを変更しても再コンパイルする必要がありません。したがって，HTML テンプレートを変更することで，容易に Web ページの出力内容を変更できます。しかし，実行時に HTML テンプレートをインタプリットするため，実行速度は低速です。

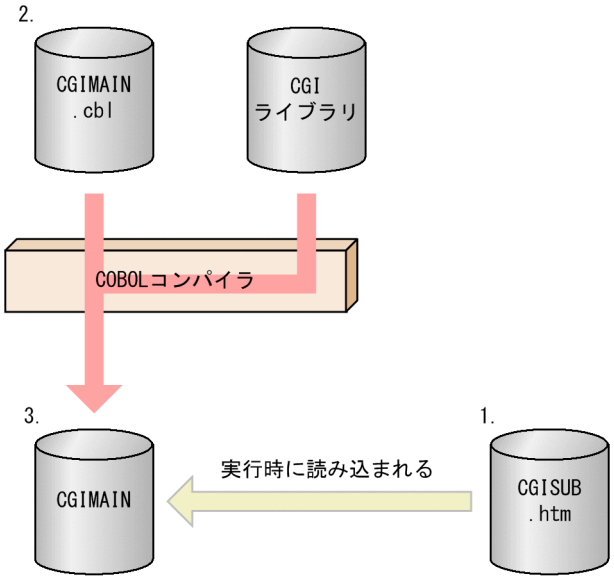
インタプリット型 CGI プログラムの作成手順

1. Web ページに出力する HTML テンプレート（CGISUB.htm）を作成します。
2. 1.の HTML テンプレートをインタプリットする COBOL 主プログラム（CGIMAIN.cbl）を作成します。

   インタプリットには CBLFILLTEMPLATE サービスルーチンを使用します。
3. COBOL 主プログラムだけをコンパイルし，CGI ライブラリとリンクして，CGI プログラム（CGIMAIN）を生成します。

CGI プログラムの実行時に，HTML テンプレートがインタプリットされます。

図 25–2　インタプリット型 CGI プログラムの作成手順



25.2.3　ダイナミック型 CGI プログラムの作成方法

ダイナミック型 CGI プログラムは，COBOL プログラムが実行中に CBLFILLTEMPLATE サービスルーチンを利用して，共用ライブラリを呼び出す形式のプログラムです。共用ライブラリは，あらかじめ HTML トランスレータを利用して HTML テンプレートから生成しておきます。

ダイナミック型 CGI プログラムは，テストしながら HTML テンプレートを修正できます。また，HTML テンプレートをテスト後に共用ライブラリ化しても，CGI プログラムを再コンパイルする必要がありません。ただし，共用ライブラリの名称を変更した場合は，CGI プログラムを再コンパイルする必要があります。また，HTML テンプレートが複数ある場合でも，一つの共用ライブラリファイルにまとめられません。この場合，HTML テンプレートごとに共用ライブラリファイルを作成する必要があります。

ダイナミック型 CGI プログラムの作成手順

1. Web ページに出力する HTML テンプレート（CGISUB.htm）を作成します。
2. 1.で作成したファイルを呼び出す COBOL 主プログラム（CGIMAIN.cbl）を作成します。

   COBOL 主プログラムからは，CBLFILLTEMPLATE サービスルーチンを使用して HTML テンプレートを呼び出すようにします。このとき，CBLFILLTEMPLATE サービスルーチンに指定する HTML テンプレートの名称には，1.のファイル名から拡張子を除いた名称（CGISUB）を指定してください。
3. COBOL 主プログラムだけをコンパイルし，CGI ライブラリとリンクして CGI プログラム（CGIMAIN）を生成します。
4. CGI プログラムと HTML テンプレートを使って，HTML テンプレートが正しく Web ページに出力されるかテストします。

正しく出力されるまで，HTML テンプレートを修正しながらテストを繰り返します。

5. HTML トランスレータを使用して，HTML テンプレートから COBOL 副プログラムのソースファイル（CGISUB.cbl）を生成します。次に指定例を示します。

   cblhtmtr CGISUB.htm

6. COBOL 副プログラムのソースファイルをコンパイルし，共用ライブラリファイル（CGISUB）を生成します。

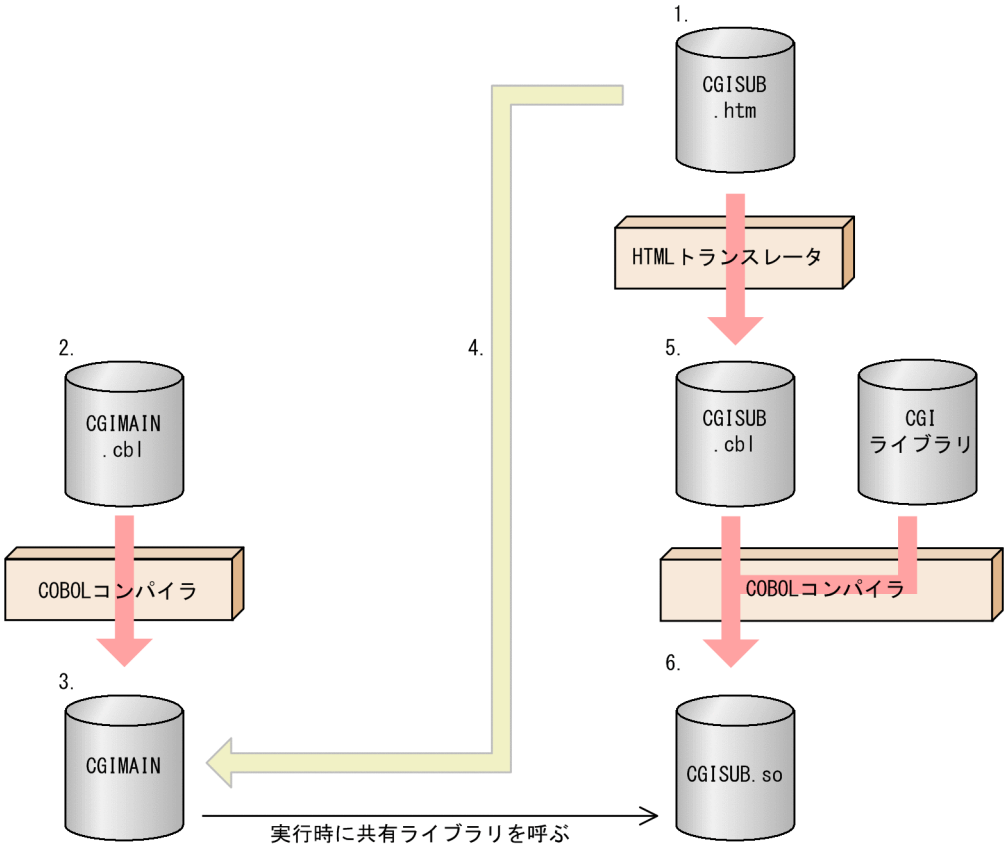
   CGI プログラムは，実行時に共用ライブラリファイル※を呼び出します。

注※

共用ライブラリファイルを作成する ld コマンドでのリンク例

```
ld -o CGISUB.a CGISUB.o -bpT:0x10000000
   -bpD:0x20000000 -bnoentry -bM:SRE -bexpall
   -L/opt/HILNGcbl2k/lib -lcbl2kcgi -lcbl2k
    -lcbl2kml -lm -lc
```

図 25-3　ダイナミック型 CGI プログラムの作成手順



25.2.4　CGI プログラムのコンパイル，およびリンク方法

ここでは，CGI プログラム作成支援機能を使用した COBOL プログラムのコンパイル，およびリンク方法について説明します。

(1)　ccbl2002 コマンドを使用する場合

ccbl2002 コマンドを使う場合は，CGI ライブラリとリンクするように指定してください。次に，形式を示します。

(a)　スタティック型 CGI プログラムを作成する場合

```
ccbl2002 -OutputFile CGIMAIN -Main,System CGIMAIN.cbl
         CGISUB.htm -lcbl2kcgi
```

(b)　インタプリット型 CGI プログラムを作成する場合

```
ccbl2002 -OutputFile CGIMAIN -Main,System CGIMAIN.cbl
         -lcbl2kcgi
```

(c)　ダイナミック型 CGI プログラムを作成する場合

```
ccbl2002 -OutputFile CGIMAIN -Main,System CGIMAIN.cbl
         -lcbl2kcgi
```

(2)　cc／ld コマンドを使用する場合

cc コマンド，または ld コマンドを使う場合は，CGI ライブラリとリンクするように指定してください。次に，形式を示します。

(a)　スタティック型 CGI プログラムを作成する場合

```
cc CGIMAIN.o CGISUB.o -L/opt/HILNGcbl2k/lib -lcbl2kcgi -lcbl2k
   -lcbl2kml … -ldl …
```

(b)　インタプリット型 CGI プログラムを作成する場合

```
cc CGIMAIN.o -L/opt/HILNGcbl2k/lib -lcbl2kcgi -lcbl2k
   -lcbl2kml … -ldl …
```

(c)　ダイナミック型 CGI プログラムを作成する場合

```
cc CGIMAIN.o -L/opt/HILNGcbl2k/lib -lcbl2kcgi -lcbl2k
   -lcbl2kml …
```

25.3　フォーム情報の取得と CGI リスト

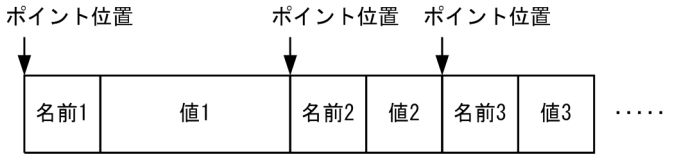
CGI プログラム作成支援機能では，Web ページでフォームタグに入力された情報を，CGI リストという形式で扱えます。ここでは，フォームタグに入力された情報から CGI リストを作成する方法，および COBOL プログラムで CGI リストを扱う方法について説明します。

25.3.1　CGI リストの概要

クライアントの Web ブラウザでフォームタグに対して入力された情報は，Web サーバを経由して CGI プログラムに渡されます。これをフォーム情報といいます。フォーム情報は，データの「名前」とデータの「値」が対になっているデータ構造をしています。

CGI プログラム作成支援機能では，フォーム情報を解析し，COBOL で扱えるリスト構造体に格納できます。このリスト構造体のことを CGI リストと呼びます。次に CGI リストの構造を示します。

図 25–4　CGI リストの構造



図中のポイント位置とは，COBOL プログラムが CGI リストにアクセスする時に，任意のデータを指定するために必要なポインタのことです。詳細は，「25.7　CGI プログラムの作成を支援するサービスルーチン」を参照してください。

25.3.2　CGI リストのデータと COBOL のデータ記述

CGI リストの「名前」と「値」の対応は，COBOL では次に示す集団項目となります。

表 25–1　CGI リストのデータと COBOL のデータ記述の対応

CGI リストのデータ	COBOL のデータ記述
名前	`01 データ名1 PIC X(28).`

値	`01 データ名2.`
	`  02 データ名3 PIC S9(9) USAGE COMP.`
	`  02 データ名4.`
	`  03 PIC X OCCURS 1024 DEPENDING ON データ名3.`

注

「名前」は，COBOL のデータ名 1 に対応し，28 バイトの英数字項目で指定します。

「値」の長さは，COBOL 集団項目のデータ名 3 に対応し，4 バイトの 2 進項目で指定します。

「値」は，COBOL 集団項目のデータ名 4 に対応し，最大 1,024 バイトの可変長英数字項目で指定します。

25.3.3　CGI リストの作成と編集

フォーム情報から CGI リストを作成したり，CGI リストの内容を編集したりするには，サービスルーチンを使用します。詳細は，「25.7　CGI プログラムの作成を支援するサービスルーチン」を参照してください。

25.4　CGI 環境変数へのアクセス

CGI プログラム作成支援機能では，環境変数を使って CGI プログラムに必要な情報を取得できます。CGI プログラムに必要な情報を提供する環境変数を，CGI 環境変数と呼びます。

CGI 環境変数から値を取得するには，CBLGETENV サービスルーチンを使用します。また，CGI 環境変数の一覧と値を HTML 形式で出力するには，CBLPRINTENV サービスルーチンを使用します。各サービスルーチンの詳細については，「25.7　CGI プログラムの作成を支援するサービスルーチン」を参照してください。

次に，アクセスできる CGI 環境変数の一覧を示します。

表 25–2　CGI プログラム作成支援機能でアクセスできる CGI 環境変数

CGI 環境変数名	説明
AUTH_TYPE	ユーザを認証するときに使う認証のタイプ
CONTENT_LENGTH	POST 要求の場合の入力バイト数
CONTENT_TYPE	text/HTML のような MIME タイプ HTTP ファイルのアップロード用にはmultipart/form-data がセットされる。
GATEWAY_INTERFACE	サーバが使用する CGI プロトコルのバージョン
HTTP_ACCEPT	Web ブラウザが受け取る MIME タイプ
HTTP_COOKIE	クッキー情報
HTTP_USER_AGENT	クライアントが使用する Web ブラウザ名，バージョン
PATH_INFO	CGI プログラムに渡される追加パス情報 （URL 中のスクリプト名のあとのクエリー文字列の前にある部分）
PATH_TRANSLATED	環境変数 PATH_INFO に指定された絶対パス情報 （仮想パス名がディレクトリに展開される）
QUERY_STRING	GET 要求の場合，CGI プログラムに渡されるクエリー情報
REMOTE_ADDR	クライアントの IP アドレス
REMOTE_HOST	クライアントのホスト名
REMOTE_IDENT	要求を出しているクライアント
REMOTE_USER	クライアントによって与えられ，サーバに認証されたユーザ名称（AUTH_TYPE が設定されている時だけ有効）
REQUEST_METHOD	HTTP 要求メソッド名（"GET"または"POST"）
SCRIPT_NAME	実行中の CGI プログラム名
SERVER_NAME	サーバのホスト名
SERVER_PORT	サーバが動いているホストの TCP/IP ポート番号

CGI 環境変数名	説明
SERVER_PROTOCOL	要求された情報を取り出すプロトコル名とバージョン
SERVER_SOFTWARE	クライアントの要求に答えている Web サーバ名とバージョン

25.5 HTML ファイルを COBOL ソースファイルに変換する方法

HTML トランスレータを使うと，HTML ファイルを COBOL 副プログラムのソースファイルに変換できます。ここでは，HTML トランスレータの使用方法について説明します。

25.5.1 HTML トランスレータを使った HTML ファイルの変換

HTML トランスレータで HTML ファイルを COBOL ソースファイルに変換するには，次のように指定します。

形式

```
cblhtmtr 〔-H2cbl〕 HTMLファイル名 〔…〕
```

-H2cbl

固定形式ソースに変換するオプションです。オプションを指定しなかった場合は，-H2cbl が仮定されます。

HTML ファイル名

COBOL ソースファイルに変換したい HTML ファイルの名前を指定します。複数の HTML ファイル名を指定した場合，別々の COBOL ソースファイルに変換されます。

戻り値

0：正常終了した場合

1：変換中にエラーが発生した場合

規則

- HTML ファイル名に指定できるのは，拡張子が.htm，および.html のファイルだけです。
- HTML ファイル名に指定するファイル名またはディレクトリ名に空白が含まれる場合，ファイルを引用符（ " ）で囲って指定します。
- ファイル名の長さが 30 バイトを超える場合，HTML ファイル名に指定できません。
- HTML ファイル名に指定するファイル名は，COBOL のプログラム名の構文規則に従った名称でなければなりません。プログラム名に指定した英小文字，ハイフン（－），#，¥，@はコンパイル時に別の文字に変換されるため，CBLFILLTEMPLATE サービスルーチンに指定する HTML ファイル名に使用しないでください。
- 絶対パス名の長さが 255 バイトを超えるファイル名は，HTML ファイル名に指定できません。
- 出力される COBOL ソースファイルの名称は，「HTML ファイル名.cbl」となります。
- 出力された COBOL ソースファイルの PROGRAM-ID は，HTML ファイル名から拡張子を除いた名称になります。
- HTML ファイルの変換中にエラーが発生した場合，エラーメッセージが出力されます。出力されるエラーメッセージの形式は，コンパイラが出力するメッセージと同じです。

注意事項

- 次の文字列は，HTML トランスレータの予約語になっています。
  - $%
  - %$
  - REPEAT
  - END-REPEAT
  - IF
  - ELSE
  - END-IF
  - NULL

  「$」および「%」を Web ページに出力したい場合，実体参照で記述する必要があります。実体参照での"$"，"%"の表記を，次に示します。

表示する文字	実体参照での表記
$	&#36;
%	&#37;

- HTML トランスレータで固定形式ソースを作成すると，テキスト領域にある日本語などの多バイト文字は，環境変数 LANG の値によって次のような文字コードを使用しているとみなして処理されます。

環境変数 LANG の設定値によって有効となる文字コード※	文字コード
シフト JIS コード	シフト JIS コード
日本語 EUC コード	日本語 EUC コード
上記以外	シフト JIS コード

注※
環境変数 LANG の設定値と文字コードについては，「付録 A　COBOL で使用する文字集合」を参照してください。

漢字コードによるコード一覧を次に示します。

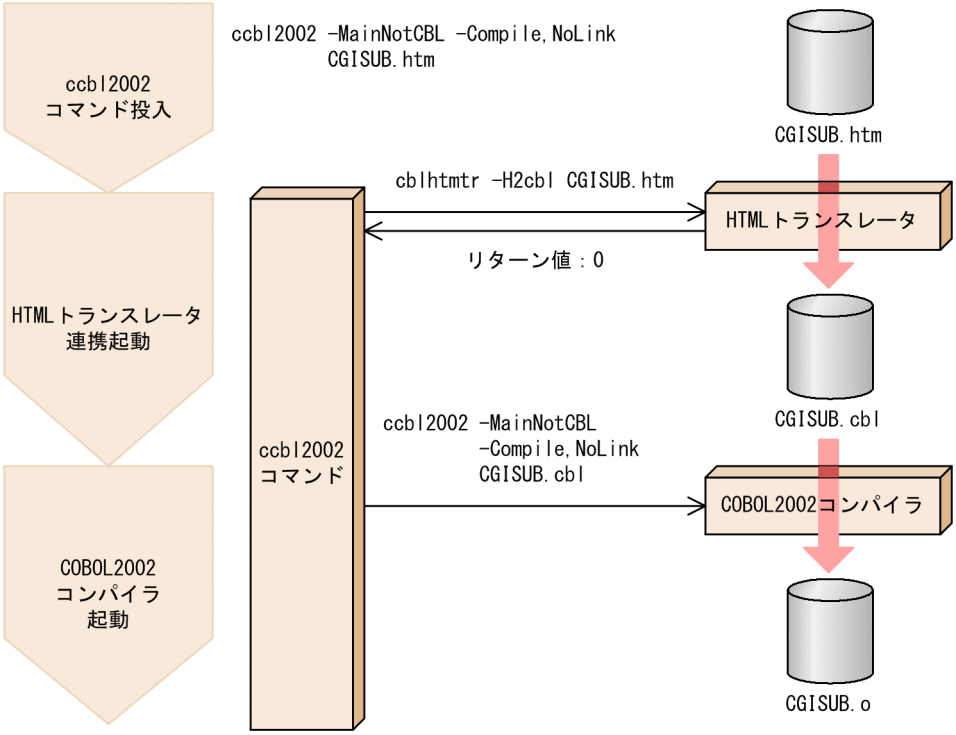
漢字コード	先行コード	後続コード
シフト JIS コード	0x81～0x9F，0xE0～0xFC	0x40～0x7E,0x80～0xFC
日本語 EUC コード	0xA1～0xFE	0xA1～0xFE

- HTML テンプレートから COBOL ファイルを生成中に HTML 拡張言語でエラーが発生した場合，エラーメッセージの行情報がずれて出力されることがあります。

25.5.2　ccbl2002 コマンドからの HTML ファイルの変換

ccbl2002 コマンドに HTML ファイルを指定すると，HTML ファイルの COBOL ソースファイルへの変換，COBOL プログラムへのコンパイルが連携して実行されます。次に ccbl2002 コマンドで HTML ファイルをコンパイルする流れを示します。

図 25-5　ccbl2002 コマンドで HTML ファイルをコンパイルするときの流れ



規則

- ccbl2002 コマンドに指定されたファイルのうち，拡張子が.htm，および.html のファイルだけ，HTML トランスレータの処理対象となります。
- ccbl2002 コマンドに指定されたコンパイラオプションは，HTML トランスレータが生成した COBOL ソースファイルに対しても有効になります。
- ccbl2002 コマンドから連携して HTML トランスレータが起動された場合，変換された COBOL ソースファイルは，カレントディレクトリに出力されます。

注意事項

ccbl2002 コマンドに HTML ファイルと同名の COBOL ソースファイルを指定してはいけません。

例えば，「ccbl2002 CGIMAIN.cbl CGIMAIN.htm -lcbl2kcgi」と指定すると，CGIMAIN.htm が COBOL ソースファイルに変換され，CGIMAIN.cbl に上書きされます。

25.6　HTML テンプレート機能

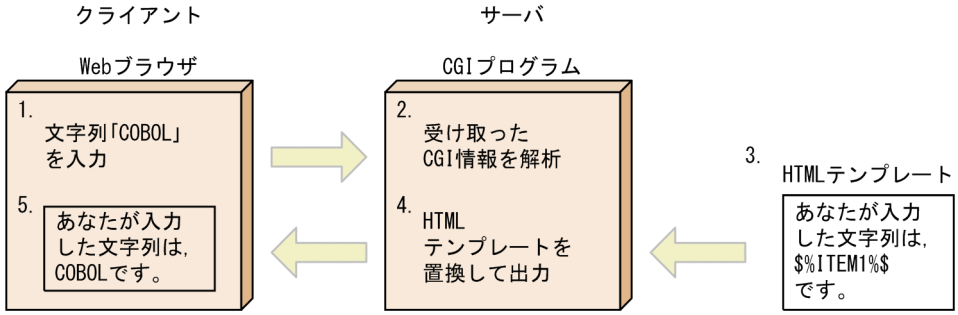
ここでは，HTML テンプレートと，その使用方法について説明します。

25.6.1　HTML テンプレート機能の概要

HTML テンプレートとは，Web ページの原形となる HTML ファイルに HTML 拡張言語を埋め込んだものです。HTML 拡張言語を利用すると，動的な Web ページが効率良く作成できます。

HTML テンプレートで，動的な Web ページが出力される例を次に示します。

図 25-6　HTML テンプレート機能で動的な Web ページが出力される例



1. クライアント側の Web ブラウザで「COBOL」という文字列を入力すると，サーバ側の CGI プログラムに送られる。
2. CGI プログラムは，受け取った CGI 情報を解析する。
3. CGI プログラムは，Web ブラウザに出力する Web ページのひな形となる，HTML テンプレートを読み込む。
4. CGI プログラムは，読み込んだ HTML テンプレート中の HTML 拡張言語「$%ITEM1%$」と入力された文字列「COBOL」を置換して Web ページに出力する。
5. CGI プログラムから出力された Web ページが，Web ブラウザに表示される。

25.6.2　HTML 拡張言語の文法

HTML テンプレートに埋め込む HTML 拡張言語の文法について説明します。

(1)　$%変数名%$

形式

```
$%変数名%$
```

機能

COBOL 主プログラムから渡された CGI リストの先頭から変数名と一致する「名前」を検索し，「名前」に対応する「値」と置換します。

構文規則

変数名は，以下の条件を満たす必要があります。

- 先頭の文字が英字である
- 2 文字目以降が，英数字（A～Z，a～z，0～9），ハイフン（－），下線（__）のどれかで構成されている
- 長さが 28 バイト以下である

使用例

CGI リストの「名前」が"ITEM"の項目の「値」を表示する例です。なお，この使用例は，HTML トランスレータを使用する CGI プログラムの作成例です。

HTML テンプレート（CGISUB.htm）

```
<HTML>
<HEAD>
<TITLE>変数名</TITLE>
</HEAD>
<BODY>
Name : $%ITEM1%$
</BODY>
</HTML>
```

COBOL 主プログラム（CGIMAIN.cbl）

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CGIMAIN.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 NAMELIST USAGE ADDRESS VALUE NULL.
       01 NAMEX PIC X(28).
       01 VALUEX.
         02 VALUE-LEN PIC S9(9) USAGE COMP.
         02 VALUE-STRING.
           03 PIC X OCCURS 1024 DEPENDING ON VALUE-LEN.
       PROCEDURE DIVISION.
      *> CGIリスト作成
            CALL 'CBLCREATELIST' USING NAMELIST.
            MOVE 'ITEM1' TO NAMEX.
            COMPUTE VALUE-LEN = FUNCTION LENGTH('COBOL2002').
            MOVE 'COBOL2002' TO VALUE-STRING.
            CALL 'CBLADDPAIR' USING NAMELIST NAMEX VALUEX.
      *> COBOL副プログラムの呼び出し
            CALL 'CGISUB' USING NAMELIST.
      *> CGIリスト削除
            CALL 'CBLDESTROYLIST' USING NAMELIST.
            STOP RUN.
```

Web ブラウザの出力例

```
Name:COBOL2002
```

(2) $%REPEAT%$文

形式

書き方 1（$%REPEAT%$文の間に$%変数名%$が一つだけある場合）

```
$%REPEAT%$
  $%変数名%$
$%END-REPEAT%$
```

書き方 2（$%REPEAT%$文の間に$%変数名%$が複数ある場合）

```
$%REPEAT%$
  $%変数名1%$
  $%変数名2%$
  :
$%END-REPEAT%$
```

機能

書き方 1 の機能

COBOL 主プログラムから渡された CGI リストの先頭から変数名と一致する「名前」を検索し，次の条件が満たされるまで「名前」に対応する「値」を繰り返し出力します。

- CGI リストの終端まで検索する
- 検索中に，CBLENDREPEAT サービスルーチンで CGI リストに追加した終端インジケータ（REPEAT.TERMINATOR）が見つかる

書き方 2 の機能

まず，COBOL 主プログラムから渡された CGI リストの先頭から変数名 1 と一致する「名前」を検索し，その「値」を出力します。次に，変数名 1 を発見した位置以降から変数名 2 と一致する「名前」を検索し，その「値」を出力します。同様の作業を繰り返し，最後の変数名の検索が終わったら，また，最初の変数名 1 から検索を続けます。

このような作業を，次の条件が満たされるまで繰り返します。

- CGI リストの終端まで検索する
- 検索中に，CBLENDREPEAT サービスルーチンで CGI リストに追加した終端インジケータ（REPEAT.TERMINATOR）が見つかる

（例）

CGI リストの内容

名前	値
ITEM1	DATA01

名前	値
ITEM2	DATA02
ITEM1	DATA11
ITEM2	DATA12

HTML テンプレート

```
$%REPEAT%$
$%ITEM1%$
$%ITEM2%$
$%END-REPEAT%$
```

表示される内容

```
DATA01
DATA02
DATA11
DATA12
```

構文規則

- 変数名は，以下の条件を満たす必要があります。

  1. 先頭の文字が英字である

  2. 2 文字目以降が，英数字（A～Z，a～z，0～9），ハイフン（－），下線（__）のどれかで構成されている

  3. 長さが 28 バイト以下である

- $%REPEAT%$～$%END-REPEAT%$の間には，$%IF%$文を記述できません。
- $%REPEAT%$文を入れ子にした場合，2 階層までしか記述できません。
- $%REPEAT%$～$%END-REPEAT%$の間に$%変数名%$がない場合，$%REPEAT%$文は COBOL 副プログラムに変換されません。

使用例 1

CGI リストの「名前」が"CITY"または"COLOR"である項目の「値」をすべて表示する例です。なお，この使用例は，HTML トランスレータを使用する CGI プログラムの作成例です。

HTML テンプレート（CGISUB.htm）

```
<HTML>
<HEAD>
<TITLE>REPEAT</TITLE>
</HEAD>
<BODY>
$%REPEAT%$
$%CITY%$ : $%COLOR%$ <BR>
$%END-REPEAT%$
</BODY>
</HTML>
```

COBOL 主プログラム（CGIMAIN.cbl）

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CGIMAIN.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 NAMELIST USAGE ADDRESS VALUE NULL.
       01 NAMEX PIC X(28).
       01 VALUEX.
         02 VALUE-LEN PIC S9(9) USAGE COMP.
         02 VALUE-STRING.
           03 PIC X OCCURS 1024 DEPENDING ON VALUE-LEN.
       PROCEDURE DIVISION.
      *> CGIリスト作成
           CALL 'CBLCREATELIST' USING NAMELIST.
           MOVE 'CITY' TO NAMEX.
           MOVE 5 TO VALUE-LEN.
           MOVE 'TOKYO' TO VALUE-STRING.
           CALL 'CBLADDPAIR' USING NAMELIST NAMEX VALUEX.
           MOVE 'COLOR' TO NAMEX.
           MOVE 4 TO VALUE-LEN.
           MOVE 'GRAY' TO VALUE-STRING.
           CALL 'CBLADDPAIR' USING NAMELIST NAMEX VALUEX.
           MOVE 'CITY' TO NAMEX.
           MOVE 7 TO VALUE-LEN.
           MOVE 'FUKUOKA' TO VALUE-STRING.
           CALL 'CBLADDPAIR' USING NAMELIST NAMEX VALUEX.
           MOVE 'COLOR' TO NAMEX.
           MOVE 3 TO VALUE-LEN.
           MOVE 'RED' TO VALUE-STRING.
           CALL 'CBLADDPAIR' USING NAMELIST NAMEX VALUEX.
      *> COBOL副プログラムの呼び出し
           CALL 'CGISUB' USING NAMELIST.
      *> CGIリスト削除
           CALL 'CBLDESTROYLIST' USING NAMELIST.
           STOP RUN.
```

Web ブラウザの出力例

```
TOKYO:GRAY
FUKUOKA:RED
```

使用例 2

<TEXTAREA>のように 1,024 バイト以上入力できるデータを$%REPEAT%$文で繰り返して表示する場合は，入力用 HTML テンプレートに REPEAT 文の終端インジケータとなる「名前」と「値」（.REPEAT.TERMINATOR,1）を，非表示の INPUT メソッドとして追加します。

また，CGI リストに分割された「名前」と「値」の終端として（.REPEAT.TERMINATOR,1）が追加できます。次に，入力用 HTML テンプレートと出力用 HTML テンプレートの例を示します。

入力用 HTML テンプレート

```
<FORM NAME="MAIN" METHOD="POST" ACTION=
"/scripts/cgisample">
<INPUT TYPE="text" NAME="INPUTTEXT1" VALUE="DATA1"
MAXLENGTH="20"><BR>
<TEXTAREA COLS="60" ROWS="8" NAME="TEXTAREA1">
```

```
</TEXTAREA><BR>
<INPUT TYPE="hidden" NAME=".REPEAT.TERMINATOR"
VALUE="1">
<INPUT TYPE="text" NAME="INPUTTEXT1" VALUE="DATA2"
MAXLENGTH="20"><BR>
<TEXTAREA COLS="60" ROWS="8" NAME="TEXTAREA1">
</TEXTAREA><BR>
<INPUT TYPE="hidden" NAME=".REPEAT.TERMINATOR"
VALUE="1"><INPUT TYPE="submit" NAME="submit"
VALUE="送信" ><BR></FORM>  :
```

出力用 HTML テンプレート

```
  :
$%REPEAT%$
データ名：$%INPUTTEXT1%$ <BR>
データ内容：
$%REPEAT%$
$%TEXTAREA1%$
$%END-REPEAT%$
<BR>
$%END-REPEAT%$
  :
```

使用例 3

分割された「値」を$%REPEAT%$文で連続した「値」として出力する（$%変数名%$の間に空白文字を出力しない）HTML テンプレートの例を示します。ただし，連続した「値」を出力する場合を除いて，HTML 拡張言語を並べて記述しないでください。

```
 :$%REPEAT%$$%NAME1%$$%END-REPEAT%$ :
```

(3) $%IF%$文

形式

書き方 1（変数名とリテラル文字を比較する場合）

```
$%IF ( 変数名 演算子 'リテラル文字' )%$
  :
  {$%ELSE%$}
  :
$%END-IF%$
```

書き方 2（変数名と変数名を比較する場合）

```
$%IF ( 変数名1 演算子 変数名2 )%$
  :
  {$%ELSE%$}
  :
$%END-IF%$
```

書き方 3（変数名の有無を確認する場合）

```
$%IF ( 変数名1 演算子 NULL )%$
  :
```

```
  {$%ELSE%$}
   :
$%END-IF%$
```

機能

書き方 1 の機能

COBOL 主プログラムから渡された CGI リストの先頭から変数名と一致する「名前」を検索し，「名前」に対応する「値」と'リテラル文字'を演算子の条件で評価します。

書き方 2 の機能

COBOL 主プログラムから渡された CGI リストの先頭から，変数名 1 と一致する「名前」，および変数名 2 と一致する「名前」を検索し，それぞれの「名前」に対応する「値」同士を演算子の条件で評価します。

書き方 3 の機能

COBOL 主プログラムから渡された CGI リスト中に，変数名 1 と一致する「名前」があるかどうかを演算子の条件で評価します。

例えば，「$%IF(変数名 1 = NULL)%$」は，変数名 1 に該当する名前が CGI リスト中にないことを意味します。CGI リスト中に変数名があるかどうかわからない場合は，この書き方 3 を使って変数名があるかどうかを確認してから，CGI リストを参照するようにしてください。NULL を比較する演算子は"="または"!="でなければなりません。

構文規則

- 変数名は，次の条件を満たす必要があります。

  1. 先頭の文字が英字である

  2. 2 文字目以降が，英数字（A～Z，a～z，0～9），ハイフン（－），下線（__）のどれかで構成されている

  3. 長さが 28 バイト以下である

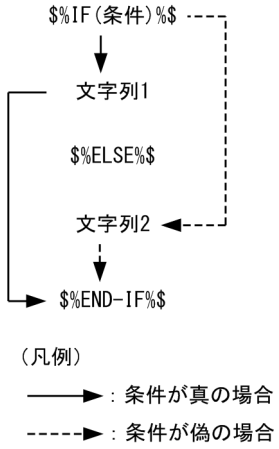
- 演算子は，次に示すものでなければなりません。

表 25-3　$%IF%$文で使用できる演算子

演算子	意味
=	等しい
!=	等しくない
>	より大きい
>=	より大きいか，または等しい
<	より小さい
<=	より小さいか，または等しい

- リテラル文字は，160 バイト以下の文字列で指定します。
- $%IF%$文を入れ子にした場合，3 階層までしか記述できません。

- $%IF%$文は，条件によって次のように制御されます。



- $%ELSE%$は省略できます。
- 変数名と一致する「名前」に対応する「値」，およびリテラル文字は，英数字項目として扱われます。

（例）'100'と'99'を比較する場合

英数字項目の比較なので'99'の末尾に空白が補われる。したがって，比較結果は，

'100' ＜ '99 '

となる。

- $%IF%$文による条件比較は，コンパイラオプションの影響を受けません。

注意事項

- CGI リストが不定で，変数名があるかどうかわからない場合は，書き方 3 で変数があることを確認してから処理を実行することを推奨します。
- $%IF%$文に指定した変数名に一致する「名前」が CGI リストにない場合は，変数名の値には NULL が仮定されます。

使用例

CGI リスト中にある「名前」が"ITEM"の項目の「値」と，リテラル文字'VALUE-X'を比較し，結果を表示する例です。なお，この使用例は，HTML トランスレータを使用する CGI プログラムの作成例です。

HTML テンプレート（CGISUB.htm）

```
Content-type: text/HTML

<HTML>
<HEAD>
<TITLE>IF ELSE END-IF</TITLE>
</HEAD>
<BODY>
$%IF ( ITEM = 'VALUE-X' )%$
  ITEM IS 'VALUE-X'. <BR>
$%ELSE%$
  NOT FOUND 'VALUE-X'. <BR>
$%END-IF%$
```

```
</BODY>
</HTML>
```

COBOL 主プログラム（CGIMAIN.cbl）

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CGIMAIN.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 NAMELIST USAGE ADDRESS VALUE NULL.
       01 NAMEX PIC X(28).
       01 VALUEX.
         02 VALUE-LEN PIC S9(9) USAGE COMP.
         02 VALUE-STRING.
           03 PIC X OCCURS 1024 DEPENDING ON VALUE-LEN.
       PROCEDURE DIVISION.
      *> CGIリスト作成
            CALL 'CBLCREATELIST' USING NAMELIST.
            MOVE 'ITEM' TO NAMEX.
            MOVE 7 TO VALUE-LEN.
            MOVE 'VALUE-X' TO VALUE-STRING.
            CALL 'CBLADDPAIR' USING NAMELIST NAMEX VALUEX.
      *> COBOL副プログラムの呼び出し
            CALL 'CGISUB' USING NAMELIST.
      *> CGIリスト削除
            CALL 'CBLDESTROYLIST' USING NAMELIST.
            STOP RUN.
```

Web ブラウザの出力例

```
ITEM IS 'VALUE-X'
```

(4) コメント

形式

```
$%-- コメント --%$
```

機能

HTML テンプレートにコメントを記述します。"$%--", "--%$"で囲まれた部分は，HTML トランスレータで生成される COBOL 副プログラムに展開されません。

構文規則

$%--：コメントの始まり

--%$：コメントの終わり

使用例

コメントの使用例です。

HTML テンプレート（CGISUB.htm）

```
Content-type: text/HTML
<HTML>
<HEAD>
```

```
<TITLE>COMMENT</TITLE>
</HEAD>
<BODY>
Webページに出力される
<!-- Webページに出力されるが表示されない -->
$%-- Webページに出力されないコメント --%$
</BODY>
</HTML>
```

COBOL 主プログラム（CGIMAIN.cbl）

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CGIMAIN.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       PROCEDURE DIVISION.
      *> COBOL副プログラムの呼び出し
           CALL 'CGISUB'.
           STOP RUN.
```

Web ブラウザの出力例

```
Webページに出力される
```

Web ブラウザに表示された Web ページの内容

```
<HTML>
<HEAD>
<TITLE>COMMENT</TITLE>
</HEAD>
<BODY>
Webページに出力される
<!-- Webページに出力されるが表示されない -->
</BODY>
</HTML>
```

(5) HTML 拡張言語を使用するときの注意事項

- HTML 拡張言語の予約語は，英大文字で記述してください。
- HTML 拡張言語の変数名と文は，1 行に記述する必要があります。

25.7　CGI プログラムの作成を支援するサービスルーチン

ここでは，CGI プログラム作成支援機能が提供するサービスルーチンについて説明します。

25.7.1　サービスルーチンの一覧

CGI プログラムの作成を支援するために COBOL2002 が用意しているサービスルーチンの種類を，次に示します。

表 25–4　サービスルーチンの種類

項番	サービスルーチン	機能
1	CBLHTMLBEGIN	HTML の先頭部分を出力する。
2	CBLHTMLEND	HTML の終端部分を出力する。
3	CBLDISPLAYTEXT	テキスト文字列を出力する。
4	CBLCONVERTTEXT	テキスト文字列を実体参照形式に変換し，出力する。
5	CBLPRINTENV	CGI 環境変数の値を HTML 形式で出力する。
6	CBLGETENV	環境変数の値を取得する。
7	CBLCGIINIT	受け取ったフォーム情報から CGI リストを作成する。
8	CBLCREATELIST	CGI リストを新たに作成する。
9	CBLDESTROYLIST	CGI リストを削除し，領域を解放する。
10	CBLADDPAIR	CGI リストの最後に「名前」と「値」の対を追加する。
11	CBLDELETEPAIR	CGI リストの現在のポイント位置の「名前」と「値」の対を削除する。
12	CBLFINDPAIR	CGI リストの先頭ポイント位置から「名前」で検索し，「値」を取得する。
13	CBLFINDNEXTPAIR	CGI リストの，次のポイント位置から「名前」をキーにして検索し，「値」を取得する。
14	CBLGETPAIR	CGI リストの現在のポイント位置から「名前」と「値」の対を取得する。
15	CBLGETPAIRNEXT	CGI リストの現在のポイント位置から「名前」と「値」の対を取得し，ポイント位置を進める。
16	CBLLISTCOUNT	CGI リストから「名前」と「値」の対の数を取得する。
17	CBLENDREPEAT	CGI リストに HTML 拡張言語で使用する終端インジケータを追加する。
18	CBLFILLTEMPLATE	HTML テンプレートをインタプリットし，動的な Web ページを出力する。
19	CBLPRINTLIST	CGI リストの内容を HTML 形式で出力する。
20	CBLSENDERROR	エラーメッセージを HTML 形式で出力する。

項番	サービスルーチン	機能
21	CBLCGITRACE	CGI プログラムの作成を支援するサービスルーチンのトレース情報をファイルに出力する。

なお，各サービスルーチンの戻り値は，ほかのサービスルーチンと同様に RETURN-CODE 特殊レジスタで参照できます。詳細は，「29.2　戻り値の使い方」を参照してください。

25.7.2　サービスルーチンの説明

(1)　CBLHTMLBEGIN

HTML の先頭部分を出力するサービスルーチンです。CBLHTMLBEGIN サービスルーチンが出力する内容を，次に示します。

```
<HTML>
<HEAD>
<TITLE> 引数1 </TITLE>
</HEAD>
<BODY>
```

形式

```
CALL 'CBLHTMLBEGIN' USING 引数1
```

引数

引数 1 には，Web ブラウザのタイトルバーに表示する文字列を，128 バイトの英数字項目で指定します。

戻り値

0：正常終了した場合

-1：エラーが発生した場合（標準出力に出力できない場合）

規則

- HTML テンプレートを使用する場合，CBLFILLTEMPLATE サービスルーチンや HTML トランスレータによって HTML の先頭部分が自動生成されるので，CBLHTMLBEGIN サービスルーチンは使用しないでください。
- Web ブラウザのタイトルバーに表示される文字数は，使用する Web ブラウザの種類によって異なります。また，Web ブラウザによっては，タイトルバーに表示する文字列の中に連続した空白がある場合，一つの空白に置き換えて出力するものもあります。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 TITLE-NAME PIC X(128).
       PROCEDURE DIVISION.
```

```
           MOVE 'This my test title' TO TITLE-NAME.
           CALL 'CBLHTMLBEGIN' USING TITLE-NAME.
           CALL 'CBLHTMLEND'.
```

(2) CBLHTMLEND

HTML の終端部分を出力するサービスルーチンです。CBLHTMLEND サービスルーチンが出力する内容を，次に示します。

```
</BODY>
</HTML>
```

形式

```
CALL 'CBLHTMLEND'
```

引数

なし。

戻り値

0：正常終了した場合

-1：エラーが発生した場合（標準出力に出力できない場合）

規則

HTML テンプレートを使用する場合，CBLFILLTEMPLATE サービスルーチンや HTML トランスレータによって HTML の先頭部分が自動生成されるので，CBLHTMLEND サービスルーチンは使用しないでください。

使用例

「(1) CBLHTMLBEGIN」の使用例を参照してください。

(3) CBLDISPLAYTEXT

引数に指定された文字列を出力するサービスルーチンです。

形式

```
CALL 'CBLDISPLAYTEXT' USING 引数1
```

引数

引数 1 には，次に示すインタフェース領域の名前を指定します。

表 25-5 CBLDISPLAYTEXT サービスルーチンのインタフェース領域

記述形式	内容
01 データ名1.	CALL 文の USING で指定するインタフェース領域の名前。

記述形式	内容
02 データ名2 PIC S9(9) USAGE COMP.	出力する文字列の長さを４バイトの２進項目で指定する。
02 データ名3. 03 PIC X OCCURS 1024 DEPENDING ON データ名2.	出力する文字列を可変長英数字項目で指定する。最大 1,024 バイトまで指定できる。

戻り値

0：正常終了した場合

-1：エラーが発生した場合（標準出力に出力できない場合）

規則

CBLDISPLAYTEXT サービスルーチンが出力する文字列の終端には，改行文字("¥n")が追加されません。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 NAMEX.
         02 NAME-LEN PIC S9(9) USAGE COMP.
         02 NAME-STRING.
           03 PIC X OCCURS 1024 DEPENDING ON NAME-LEN.
       PROCEDURE DIVISION.
           COMPUTE NAME-LEN = FUNCTION LENGTH('COBOL').
           MOVE 'COBOL' TO NAME-STRING.
           CALL 'CBLDISPLAYTEXT' USING NAMEX.
```

(4) CBLCONVERTTEXT

引数に指定した文字列を，実体参照の形式に変換して出力するサービスルーチンです。

実体参照とは，HTML 言語で予約されているため表示できない文字を，別の文字列を使って表す記法です。例えば，HTML 言語では「<」「>」の文字が予約されているため，実体参照ではこれを「<」「>」と表します。

形式

```
CALL 'CBLCONVERTTEXT' USING 引数1
```

引数

引数 1 には，次に示すインタフェース領域の名前を指定します。

表 25-6　CBLCONVERTTEXT サービスルーチンのインタフェース領域

記述形式	内容
01 データ名1.	CALL 文の USING で指定するインタフェース領域の名前。
02 データ名2 PIC S9(9) USAGE COMP.	変換して出力する文字列の長さを４バイトの２進項目で指定する。

記述形式	内容
02 データ名3. 03 PIC X OCCURS 1024 DEPENDING ON データ名2.	変換して出力する文字列を可変長英数字項目で指定する。 最大 1,024 バイトまで指定できる。

戻り値

0：正常終了した場合

-1：エラーが発生した場合（標準出力に出力できない場合）

規則

日本語などの多バイト文字は，CBLCONVERTTEXT サービスルーチンで実体参照文字に変換されません。詳細については，以降の注意事項を参照してください。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 NAMEX.
         02 NAME-LEN PIC S9(9) USAGE COMP.
         02 NAME-STRING.
           03 PIC X OCCURS 1024 DEPENDING ON NAME-LEN.
       01 TEXT-STR PIC X(17) VALUE 'SHOW AN <OPTION>.'.
       PROCEDURE DIVISION.
           :
           COMPUTE NAME-LEN = FUNCTION  LENGTH (TEXT-STR).

           MOVE TEXT-STR TO NAME-STRING.
           CALL 'CBLCONVERTTEXT' USING NAMEX.
```

HTML の出力例

```
&#83;&#72;&#79;&#87;&#32;&#65;&#78;&#32;&#60;&#79;
&#80;&#84;&#73;&#79;&#78;&#62;&#46;
```

実体参照での表した文字の一覧

実体参照での表記の一覧を，次に示します。

文字	実体参照での表記
半角空白	&#32;
!	&#33;
″	&#34;
#	&#35;
$	&#36;
%	&#37;
&	&#38;
,	&#39;
(	&#40;
)	&#41;
*	&#42;
+	&#43;
’	&#44;
−	&#45;
.	&#46;
/	&#47;
0	&#48;
1	&#49;
2	&#50;
3	&#51;
4	&#52;
5	&#53;
6	&#54;
7	&#55;
8	&#56;
9	&#57;
:	&#58;
;	&#59;
<	&#60;
=	&#61;
>	&#62;
?	&#63;

文字	実体参照での表記
@	&#64;
A	&#65;
B	&#66;
C	&#67;
D	&#68;
E	&#69;
F	&#70;
G	&#71;
H	&#72;
I	&#73;
J	&#74;
K	&#75;
L	&#76;
M	&#77;
N	&#78;
O	&#79;
P	&#80;
Q	&#81;
R	&#82;
S	&#83;
T	&#84;
U	&#85;
V	&#86;
W	&#87;
X	&#88;
Y	&#89;
Z	&#90;
[	&#91;
¥	&#92;
]	&#93;
^	&#94;
_	&#95;

文字	実体参照での表記
`	&#96;
a	&#97;
b	&#98;
c	&#99;
d	&#100;
e	&#101;
f	&#102;
g	&#103;
h	&#104;
i	&#105;
j	&#106;
k	&#107;
l	&#108;
m	&#109;
n	&#110;
o	&#111;
p	&#112;
q	&#113;
r	&#114;
s	&#115;
t	&#116;
u	&#117;
v	&#118;
w	&#119;
x	&#120;
y	&#121;
z	&#122;
{	&#123;
		&#124;
}	&#125;
~	&#126;

注意事項

- 日本語などの多バイト文字は，CBLCONVERTTEXT で変換されません。また，環境変数 LANG に指定した値によって次の表のようになります。

環境変数 LANG の設定値によって有効となる文字コード※	変換しない文字
シフト JIS コード	シフト JIS コードは変換しない。
日本語 EUC コード	日本語 EUC コードは変換しない。
上記以外	「実体参照での表記の一覧」に記載されている文字以外は無視する。

注※

環境変数 LANG の設定値と文字コードについては，「付録 A　COBOL で使用する文字集合」を参照してください。

「実体参照での表記の一覧」に記載されている文字，および環境変数 LANG の指定によって変換されない文字以外の文字を使用できません。

(5)　CBLPRINTENV

CGI 環境変数の名前と現在の値を，HTML 形式で出力するサービスルーチンです。

出力される環境変数の種類については，「25.4　CGI 環境変数へのアクセス」を参照してください。

形式

```
CALL 'CBLPRINTENV'
```

引数

なし。

戻り値

0：正常終了した場合

-1：エラーが発生した場合（標準出力に出力できない場合）

規則

環境変数に値が設定されていない場合，「(NULL)」が出力されます。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 INPUTLIST USAGE ADDRESS VALUE NULL.
       PROCEDURE DIVISION.
           CALL 'CBLCGIINIT' USING INPUTLIST.
           CALL 'CBLPRINTENV'.
           :
           CALL 'CBLDESTROYLIST' USING INPUTLIST.
```

HTML の出力例

```
<HTML>
   <BODY>
<H3> Environment variable list: </H3>
AUTH_TYPE= (NULL) <BR>
CONTENT_LENGTH= 0 <BR>
CONTENT_TYPE= (NULL) <BR>
GATEWAY_INTERFACE= CGI/1.1 <BR>
 :
 :
   </BODY>
</HTML>
```

(6) CBLGETENV

環境変数の値を取得するサービスルーチンです。

形式

```
CALL ’CBLGETENV’ USING 引数1 引数2
```

引数

- 引数 1 には，環境変数名を 28 バイトの英数字項目で指定します。
- 引数 2 には，次に示すインタフェース領域の名前を指定します。

表 25-7　CBLGETENV サービスルーチンのインタフェース領域

記述形式	内容
01 データ名1.	CALL 文の USING で指定するインタフェース領域の名前。
02 データ名2 PIC S9(9) USAGE COMP.	取得した環境変数の値（文字列）の長さを格納する領域で，4 バイトの 2 進項目で指定する。
02 データ名3. 03 PIC X OCCURS 1024 DEPENDING ON データ名2.	取得した環境変数の値を格納する領域で，最大 1,024 バイトの可変長集団項目を指定する。

戻り値

0：正常終了した場合

-1：エラーが発生した場合（該当する環境変数が見つからない場合）

1：環境変数の値が 1,024 バイトを超えている場合

規則

- 環境変数に値が設定されていない場合，引数 2 のデータ名 2 には 0 が，データ名 3 には NULL 文字が返されます。また，RETURN-CODE 特殊レジスタの値は 0 になります。
- 指定した環境変数名の末尾が空白文字の場合，環境変数の内容は取得されません。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 NAMEX PIC X(28) VALUE ’REMOTE_HOST’.
       01 VALUEX.
         02 VALUE-LEN PIC S9(9) USAGE COMP.
         02 VALUE-STRING.
           03 PIC X OCCURS 1024 DEPENDING ON VALUE-LEN.
       PROCEDURE DIVISION.
           CALL ’CBLGETENV’ USING NAMEX VALUEX.
           IF RETURN-CODE = 0 THEN
             DISPLAY ’THE CLIENT MACHINE IS ’ VALUE-STRING
           ELSE
             DISPLAY ’ERROR’
           END-IF.
```

(7) CBLCGIINIT

フォーム情報から CGI リストを作成し，リストのアドレスを取得するサービスルーチンです。フォーム情報が POST メソッド（標準出力経由），GET メソッド（環境変数経由）のどちらから転送されても，CGI リストを作成できます。

また，-TDInf オプションを指定してコンパイルした CGI プログラムで CBLCGIINIT サービスルーチンが呼び出されると，CGI 用環境変数の値を CGI 環境ファイル（実行可能ファイル名.env）として，CGI プログラムと同じディレクトリに出力します。

形式

```
CALL ’CBLCGIINIT’ USING 引数1
```

引数

引数 1 には，作成した CGI リストのアドレスを格納する領域で，アドレスデータ項目を指定します。

戻り値

1：環境変数 CBLCGIINITSIZE に不当な値を設定した場合

0：正常終了した場合

-1：エラーが発生した場合

-2：POST，GET 以外をメソッドに指定した場合

-3：フォーム情報の文字列長が制限値（デフォルトは 64 キロバイト），または環境変数 CBLCGIINITSIZE で指定した値を超えた場合

変換規則

- フォーム情報から引き渡される「値」の長さが 1,024 バイトを超える場合，1,024 バイト（または 1,023 バイト）で分割した「値」を CGI リストに追加します。

環境変数 LANG の設定値によって有効となる文字コード※	「値」が分割される位置
シフト JIS コード	「値」の文字列は，シフト JIS コードとして 1,024 バイトをチェックします。1,024 バイトが 2 バイトコードの先頭バイトの場合は，1,023 バイトで文字列が分割されます。それ以外の場合は，1,024 バイトに文字列が分割されます。
日本語 EUC コード	「値」の文字列は，日本語 EUC コードとして 1,024 バイトをチェックします。1,024 バイトが 2 バイトコードの先頭バイトの場合は，1,023 バイトで文字列が分割されます。それ以外の場合は 1,024 バイトに文字列が分割されます。
上記以外	無条件に「値」を 1,024 バイトで分割して CGI リストに追加します。

注※

環境変数 LANG の設定値と文字コードについては，「付録 A　COBOL で使用する文字集合」を参照してください。

名前 = "NAME1"と値="(3,000 バイト)"を受け取って，CGI リストに追加する例を示します。

名前	値
NAME1	1,024 バイト
NAME1	1,024 バイト
NAME1	952 バイト

- 受け取れるフォーム情報の「名前」と「値」のデコードされていない最大の文字列長は，デフォルトで 64 キロバイトです。64 キロバイトを超える文字列長の場合は，CBLCGIINIT サービスルーチンで-3 が返され，処理が中断されます。

  最大文字列長を変更する場合は，環境変数 CBLCGIINITSIZE を使用します。

  CBLCGIINITSIZE=最大文字列長

  1～2,000,000（単位：キロバイト）以外の値を指定した場合は 64 が仮定されます。この場合，CBLCGIINIT サービスルーチンの戻り値は 1 となります。

- フォーム情報の「名前」や「値」に日本語などの多バイト文字が含まれている場合，CGI リストの「名前」「値」に格納されるデータは，CGI クライアントの Web ブラウザで入力された文字コードに依存します。

一般規則

- CBLCGIINIT サービスルーチンは，2 回以上呼び出さないでください。
- CGI プログラムでは，ほかのサービスルーチンを呼び出す前に CBLCGIINIT サービスルーチンを呼び出してください。ただし，CBLCGITRACE サービスルーチンは除きます。
- CBLCGIINIT サービスルーチンで CGI リストを作成した場合，CGI サブルーチンの処理を終了する時に，CBLDESTROYLIST サービスルーチンを呼び出して CGI リストを削除する必要があります。
- CBLCGIINIT サービスルーチンで CGI リストを作成した場合，作成した直後の CGI リストのポイント位置は，不定となります。CGI リストのポイント位置は，CBLFINDPAIR サービスルーチンで位置づけてください。
- マルチスレッド対応 COBOL プログラムで CBLCGIINIT サービスルーチンを呼び出した場合，CGI 環境ファイルの出力内容は保証しません。
- CBLCGIINIT サービスルーチンが呼び出されたとき，CGI プログラムの実行を支援するサービスルーチンですでに CGI ヘッダが出力されていなければ，"Content-type: text/HTML¥n¥n"が出力されます。
- CGI 環境ファイルは，REMOTE_ADDR，REMOTE_HOST 環境変数の値がある場合，または CGI 環境ファイルがない場合に出力されます。
- CGI リストのアドレスが NULL 以外の場合，CBLCGIINIT サービスルーチンの戻り値は-1 となります。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 INPUTLIST USAGE ADDRESS VALUE NULL.
```

```
PROCEDURE DIVISION.
    CALL 'CBLCGIINIT' USING INPUTLIST.
     :
    CALL 'CBLDESTROYLIST' USING INPUTLIST.
```

(8) CBLCREATELIST

新たに CGI リストを作成し，リストのアドレスを取得するサービスルーチンです。

形式

```
CALL 'CBLCREATELIST' USING 引数1
```

引数

引数 1 は，作成した CGI リストのアドレスを格納する領域で，アドレスデータ項目を指定します。

戻り値

0：正常終了した場合

-1：エラーが発生した場合（領域確保に失敗した場合など）

規則

- CBLCREATELIST サービスルーチンで CGI リストを作成した場合，CGI サブルーチンの処理を終了する時に，CBLDESTROYLIST サービスルーチンを呼び出して CGI リストを削除する必要があります。
- CBLCREATELIST サービスルーチンで CGI リストを作成した場合，作成した直後の CGI リストのポイント位置は，不定となります。
- CGI リストのアドレスが NULL 以外の場合，CBLCREATELIST サービスルーチンの戻り値は-1 となります。

使用例

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 NAMELIST USAGE ADDRESS VALUE NULL.
PROCEDURE DIVISION.
    CALL 'CBLCREATELIST' USING NAMELIST.
     :
    CALL 'CBLDESTROYLIST' USING NAMELIST.
```

(9) CBLDESTROYLIST

CGI リストを削除して，領域を解放するサービスルーチンです。

形式

```
CALL 'CBLDESTROYLIST' USING 引数1
```

引数

引数 1 には，削除する CGI リストのアドレスを，アドレスデータ項目で指定します。

戻り値

0：正常終了した場合

-1：エラーが発生した場合（領域の解放に失敗した場合など）

規則

CGI リストを削除したあとの，CGI リストのポインタの内容は保証しません。

使用例

「(7)　CBLCGIINIT」の使用例を参照してください。

(10)　CBLADDPAIR

指定した CGI リストの最後に，「名前」と「値」の対を追加するサービスルーチンです。

形式

```
CALL ’CBLADDPAIR’ USING 引数1 引数2 引数3
```

引数

- 引数 1 には，CBLCGIINIT や CBLCREATELIST で作成した CGI リストのアドレスを，アドレスデータ項目で指定します。
- 引数 2 には，CGI リストに追加する「名前」を，28 バイトの英数字項目で指定します。
- 引数 3 には，次に示すインタフェース領域の名前を指定します。

表 25-8　CBLADDPAIR サービスルーチンのインタフェース領域

記述形式	内容
01 データ名1.	CALL 文の USING で指定するインタフェース領域の名前。
02 データ名2 PIC S9(9) USAGE COMP.	CGI リストに追加する「値」の長さを格納する領域で，4 バイトの 2 進項目で指定する。
02 データ名3. 03 PIC X OCCURS 1024 DEPENDING ON データ名2.	CGI リストに追加する「値」を格納する領域で，最大 1,024 バイトの可変長集団項目を指定する。

戻り値

0：正常終了した場合

-1：エラーが発生した場合（追加時，領域確保に失敗した場合など）

規則

「名前」と「値」を追加した場合，ポイント位置は変更されません。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 NAMELIST USAGE ADDRESS VALUE NULL.
       01 VALUEX.
```

```
  02 VALUE-LEN PIC S9(9) USAGE COMP.
  02 VALUE-STRING.
    03 PIC X OCCURS 1024 DEPENDING ON VALUE-LEN.
01 NAMEX       PIC X(28) VALUE 'Hitachi'.
01 NAMEX-VALUE PIC X(9) VALUE 'COBOL2002'.
PROCEDURE DIVISION.
    CALL 'CBLCREATELIST' USING NAMELIST.
    COMPUTE VALUE-LEN = FUNCTION LENGTH(NAMEX-VALUE).
    MOVE NAMEX-VALUE TO VALUE-STRING.
    CALL 'CBLADDPAIR' USING NAMELIST NAMEX VALUEX.
    CALL 'CBLDESTROYLIST' USING NAMELIST.
```

(11) CBLDELETEPAIR

指定したCGIリストの現在のポイント位置から,「名前」と「値」の対を削除するサービスルーチンです。

形式

```
CALL 'CBLDELETEPAIR' USING 引数1
```

引数

引数1には,CBLCGIINITやCBLCREATELISTで作成したCGIリストのアドレスを,アドレスデータ項目で指定します。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

1：CGIリストが空の場合

使用例

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 INPUTLIST USAGE ADDRESS VALUE NULL.
01 NAMEX PIC X(28) VALUE 'COBOL'.
01 VALUEX.
  02 VALUE-LEN PIC S9(9) USAGE COMP.
  02 VALUE-STRING.
    03 PIC X OCCURS 1024 DEPENDING ON VALUE-LEN.
PROCEDURE DIVISION.
    CALL 'CBLCGIINIT' USING INPUTLIST.
    CALL 'CBLFINDPAIR' USING INPUTLIST
         NAMEX  VALUEX.
    CALL 'CBLDELETEPAIR' USING INPUTLIST.
    CALL 'CBLDESTROYLIST' USING INPUTLIST.
```

規則

「名前」と「値」を削除した場合,ポイント位置を次に移動します。それ以外の場合,ポイント位置はNULLになります。

(12) CBLFINDPAIR

指定したCGIリストの先頭のポイント位置から「名前」を検索し，「名前」に対応する「値」を取得するサービスルーチンです。また，一致した「名前」にポイント位置を移動します。

形式

```
CALL 'CBLFINDPAIR' USING 引数1 引数2 引数3
```

引数

- 引数1には，CBLCGIINITやCBLCREATELISTで作成したCGIリストのアドレスを，アドレスデータ項目で指定します。
- 引数2には，検索する「名前」を，28バイトの英数字項目で指定します。
- 引数3には，次に示すインタフェース領域の名前を指定します。

表 25-9 CBLFINDPAIRサービスルーチンのインタフェース領域

記述形式	内容
01 データ名1.	CALL文のUSINGで指定するインタフェース領域の名前。
02 データ名2 PIC S9(9) USAGE COMP.	検索した「名前」に対応する「値」の長さが格納される領域。4バイトの2進項目を指定する。
02 データ名3. 03 PIC X OCCURS 1024 DEPENDING ON データ名2.	検索した「名前」に対応する「値」が格納される領域。 最大1,024バイトの可変長集団項目を指定する。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

1：検索した「名前」が見つからない場合，またはポイント位置が終端の場合

規則

- 検索した「名前」に対応する「値」がNULLの場合，引数3のデータ名2には0が出力されます。
- 検索した「名前」が見つからない場合，ポイント位置はNULLになります。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 INPUTLIST USAGE ADDRESS VALUE NULL.
       01 NAMEX PIC X(28) VALUE 'Hitachi'.
       01 VALUEX.
         02 VALUE-LEN PIC S9(9) USAGE COMP.
         02 VALUE-STRING.
           03 PIC X OCCURS 1024 DEPENDING ON VALUE-LEN.
       PROCEDURE DIVISION.
           CALL 'CBLCGIINIT' USING INPUTLIST.
           CALL 'CBLFINDPAIR' USING INPUTLIST NAMEX VALUEX.
```

```
           CALL 'CBLDISPLAYTEXT' USING VALUEX.
           CALL 'CBLDESTROYLIST' USING INPUTLIST.
```

(13) CBLFINDNEXTPAIR

指定したCGIリストの，次のポイント位置から「名前」を検索し，「名前」に対応する「値」を取得するサービスルーチンです。また，一致した「名前」にポイント位置を移動します。

形式

```
CALL 'CBLFINDNEXTPAIR' USING 引数1 引数2 引数3
```

引数

- 引数1には，CBLCGIINITやCBLCREATELISTで作成したCGIリストのアドレスを，アドレスデータ項目で指定します。
- 引数2には，検索する「名前」を，28バイトの英数字項目で指定します。
- 引数3には，次に示すインタフェース領域の名前を指定します。

表 25-10 CBLFINDNEXTPAIRサービスルーチンのインタフェース領域

記述形式	内容
`01 データ名1.`	CALL文のUSINGで指定するインタフェース領域の名前。
`02 データ名2 PIC S9(9) USAGE COMP.`	検索した「名前」に対応する「値」の長さが格納される領域。4バイトの2進項目を指定する。
`02 データ名3.` `03 PIC X OCCURS 1024 DEPENDING ON データ名2.`	検索した「名前」に対応する「値」が格納される領域。 最大1,024バイトの可変長集団項目を指定する。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

1：検索した「名前」が見つからない場合，または終端インジケータ（.REPEAT.TERMINATOR,1）が見つかった場合

規則

- 検索した「名前」に対応する「値」がNULLの場合，引数3のデータ名2には0が出力されます。
- 検索した「名前」が見つからない場合，ポイント位置はNULLになります。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 INPUTLIST USAGE ADDRESS VALUE NULL.
       01 NAMEX PIC X(28) VALUE 'Hitachi'.
       01 VALUEX.
         02 VALUE-LEN PIC S9(9) USAGE COMP.
```

```
        02 VALUE-STRING.
          03 PIC X OCCURS 1024 DEPENDING ON VALUE-LEN.
      PROCEDURE DIVISION.
          CALL 'CBLCGIINIT' USING INPUTLIST.
          CALL 'CBLFINDPAIR'
                USING  INPUTLIST NAMEX VALUEX.
          CALL 'CBLDISPLAYTEXT' USING VALUEX.
          CALL 'CBLFINDNEXTPAIR'
                USING  INPUTLIST NAMEX VALUEX.
          CALL 'CBLDISPLAYTEXT' USING VALUEX.
          CALL 'CBLDESTROYLIST' USING INPUTLIST.
```

(14) CBLGETPAIR

指定した CGI リストの現在のポイント位置から「名前」と「値」の対を取得するサービスルーチンです。ポイント位置は移動しません。

形式

```
CALL 'CBLGETPAIR' USING 引数1 引数2 引数3
```

引数

- 引数 1 には，CBLCGIINIT や CBLCREATELIST で作成した CGI リストのアドレスを，アドレスデータ項目で指定します。
- 引数 2 には，「名前」を格納する領域を 28 バイトの英数字項目で指定します。
- 引数 3 には，次に示すインタフェース領域の名前を指定します。

表 25-11 CBLGETPAIR サービスルーチンのインタフェース領域

記述形式	内容
01 データ名1.	CALL 文の USING で指定するインタフェース領域の名前。
02 データ名2 PIC S9(9) USAGE COMP.	「値」の長さが格納される領域。4 バイトの 2 進項目を指定する。
02 データ名3. 03 PIC X OCCURS 1024 DEPENDING ON データ名2.	「値」が格納される領域。最大 1,024 バイトの可変長集団項目を指定する。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

1：現在のポイント位置が CGI リストの終端にあるため，「名前」および「値」が取得できない場合

規則

「名前」と「値」の取得時にエラーが発生した場合，ポイント位置は NULL になります。

使用例

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 INPUTLIST USAGE ADDRESS VALUE NULL.
01 NAMEX PIC X(28).
01 VALUEX.
  02 VALUE-LEN PIC S9(9) USAGE COMP.
  02 VALUE-STRING.
    03 PIC X OCCURS 1024 DEPENDING ON VALUE-LEN.
PROCEDURE DIVISION.
    CALL 'CBLCGIINIT' USING INPUTLIST.
    CALL 'CBLGETPAIR' USING INPUTLIST NAMEX VALUEX.
    DISPLAY 'NAME = ' NAMEX.
    CALL 'CBLDESTROYLIST' USING INPUTLIST.
```

(15) CBLGETPAIRNEXT

指定した CGI リストの現在のポイント位置から「名前」と「値」の対を取得するサービスルーチンです。また，ポイント位置を次に移動します。

形式

```
CALL 'CBLGETPAIRNEXT' USING 引数1 引数2 引数3
```

引数

- 引数 1 には，CBLCGIINIT や CBLCREATELIST で作成した CGI リストのアドレスを，アドレスデータ項目で指定します。
- 引数 2 には，「名前」を格納する領域を 28 バイトの英数字項目で指定します。
- 引数 3 には，次に示すインタフェース領域の名前を指定します。

表 25-12 CBLGETPAIRNEXT サービスルーチンのインタフェース領域

記述形式	内容
01 データ名1.	CALL 文の USING で指定するインタフェース領域の名前。
02 データ名2 PIC S9(9) USAGE COMP.	「値」の長さが格納される領域。4 バイトの 2 進項目を指定する。
02 データ名3. 03 PIC X OCCURS 1024 DEPENDING ON データ名2.	「値」が格納される領域。最大 1,024 バイトの可変長集団項目を指定する。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

（現在のポイント位置が CGI リストの終端にあった場合）

1：現在のポイント位置が CGI リストの終端にあるため，「名前」および「値」が取得できない場合

規則

「名前」と「値」の取得時にエラーが発生した場合，ポイント位置は NULL になります。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 INPUTLIST USAGE ADDRESS VALUE NULL.
       01 NAMEX PIC X(28).
       01 VALUEX.
         02 VALUE-LEN PIC S9(9) USAGE COMP.
         02 VALUE-STRING.
           03 PIC X OCCURS 1024 DEPENDING ON VALUE-LEN.
       PROCEDURE DIVISION.
           CALL 'CBLCGIINIT' USING INPUTLIST.
           CALL 'CBLGETPAIRNEXT'
                 USING INPUTLIST NAMEX VALUEX.
           DISPLAY ' NAME = ' NAMEX.
           CALL 'CBLGETPAIRNEXT'
                 USING INPUTLIST NAMEX VALUEX.
           DISPLAY ' NAME = ' NAMEX.
           CALL 'CBLDESTROYLIST' USING INPUTLIST.
```

(16) CBLLISTCOUNT

指定した CGI リスト中の「名前」と「値」の対の数を取得するサービスルーチンです。

形式

```
CALL 'CBLLISTCOUNT' USING 引数1 引数2
```

引数

- 引数 1 には，CBLCGIINIT や CBLCREATELIST で作成した CGI リストのアドレスを，アドレスデータ項目で指定します。
- 引数 2 には，CGI リスト中の「名前」と「値」の対の数を格納する領域を，4 バイトの 2 進項目で指定します。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

（不当な CGI アドレスを指定した場合など）

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 INPUTLIST USAGE ADDRESS VALUE NULL.
       01 LIST-COUNT PIC S9(9) USAGE COMP.
       PROCEDURE DIVISION.
           CALL 'CBLCGIINIT' USING INPUTLIST.
           CALL 'CBLLISTCOUNT'
                 USING INPUTLIST LIST-COUNT.
```

```
           IF RETURN-CODE = 0 THEN
             DISPLAY 'LIST COUNT : ' LIST-COUNT
           END-IF.
            :
           CALL 'CBLDESTROYLIST' USING INPUTLIST.
```

(17) CBLENDREPEAT

CGI リストに HTML 拡張言語の REPEAT で終端を認識する終端インジケータを追加するサービスルーチンです。終端インジケータは，特殊な名前と値（.REPEAT.TERMINATOR，1）で構成されています。

このサービスルーチンは，CGI リストの途中で%REPEAT%文を打ち切りたい場合などに使用します。

形式

```
CALL 'CBLENDREPEAT' USING 引数1
```

引数

引数 1 には，CBLCGIINIT や CBLCREATELIST で作成した CGI リストのアドレスを，アドレスデータ項目で指定します。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

（追加時，メモリの領域確保に失敗した場合など）

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 NAMELIST USAGE ADDRESS VALUE NULL.
        :
       PROCEDURE DIVISION.
           CALL 'CBLCREATELIST' USING NAMELIST.
           MOVE 'CITY' TO NAMEX.
           COMPUTE VALUE-LEN = FUNCTION LENGTH ('TOKYO').
           MOVE 'TOKYO' TO VALUE-STRING.
           CALL 'CBLADDPAIR' USING NAMELIST NAMEX VALUEX.
           CALL 'CBLENDREPEAT' USING NAMELIST.
           COMPUTE VALUE-LEN = FUNCTION LENGTH ('FUKUOKA').
           MOVE 'FUKUOKA' TO VALUE-STRING.
           CALL 'CBLADDPAIR' USING NAMELIST NAMEX VALUEX.
           CALL 'CBLENDREPEAT' USING NAMELIST.
            :
           CALL 'CBLDESTROYLIST' USING NAMELIST.
```

(18) CBLFILLTEMPLATE

CBLFILLTEMPLATE サービスルーチンは，HTML テンプレートをインタプリットするサービスルーチンです。また，HTML テンプレートからあらかじめ生成した共用ライブラリファイルを，実行時に呼び出せます。HTML テンプレートについては，「25.6 HTML テンプレート機能」を参照してください。

形式

```
CALL ’CBLFILLTEMPLATE’ USING 引数1 引数2
```

引数

- 引数 1 には，HTML テンプレート名を，256 バイトの英数字項目で指定します。
- 引数 2 には，HTML テンプレートで使用する CGI リストのアドレスを，アドレスデータ項目で指定します。

戻り値

0：正常終了した場合

-1：エラーが発生した場合（標準出力に出力できない場合）

規則

- 引数 1 に指定するファイル名のパス指定には，次の 3 種類があります。

1. 絶対パス名

   （例：/usr/local/httpd/cgi-bin/filename）

2. 環境変数 PATH_TRANSLATED の相対パス名

   （例：/cgi-bin/filename）

3. CGI プログラムのあるディレクトリからの相対パス名

   ファイル名だけ，またはファイルインジケータ（..）を使ったファイル名が，相対パス名として指定できます。（例：../filename）

   ただし，エイリアス指定の仮想ディレクトリにあるファイルは，絶対パス名で指定してください。

- 引数 1 には，共用ライブラリファイル（.a），HTML ファイル（.htm，または.html）※，または拡張子なしのファイル名が指定できます。拡張子なしのファイル名を指定した場合，次の順序でファイルが検索されます。
  1. 共用ライブラリファイル
  2. HTML ファイル（拡張子.htm）
  3. HTML ファイル（拡張子.html）

注※

HTML ファイルの拡張子は，大文字と小文字は等価とみなされます。

- 引数 1 に指定した HTML テンプレート名や共用ライブラリ名の末尾に空白文字があった場合，その空白はファイル名に含まれません。
- 呼び出す共用ライブラリプログラムの名称は，共用ライブラリファイル名と同じになります。引数 1 に指定する共用ライブラリファイル名に次のような文字列が含まれる場合，COBOL のプログラム名の変換規則に注意する必要があります。

  ハイフン(−)，#，¥，@，英小文字

  変換規則の詳細は，マニュアル「COBOL2002　言語　標準仕様編」「7.8.2　構文規則」を参照してください。

- HTML テンプレートをインタプリット中に HTML 拡張言語でエラーが発生した場合，エラーメッセージの行情報がずれて出力されることがあります。
- CBLFILLTEMPLATE サービスルーチンが呼び出されたとき，CGI プログラムの実行を支援するサービスルーチンですでに CGI ヘッダが出力されていなければ，"Content-type: text/HTML"が出力されます。
- HTML テンプレートのインタプリット時には，コンパイラオプションの指定は無効になります。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 INPUTLIST USAGE ADDRESS VALUE NULL.
       01 TMPLT-NAME PIC X(256) VALUE 'HITACHI.htm'.
       PROCEDURE DIVISION.
           CALL 'CBLCGIINIT' USING INPUTLIST.
           CALL 'CBLFILLTEMPLATE' USING TMPLT-NAME
                 INPUTLIST.
            :
           CALL 'CBLDESTROYLIST' USING INPUTLIST.
```

(19) CBLPRINTLIST

指定した CGI リストの内容を，HTML 形式で出力するサービスルーチンです。

形式

```
CALL 'CBLPRINTLIST' USING 引数1
```

引数

引数 1 には，CBLCGIINIT や CBLCREATELIST で作成した CGI リストのアドレスを，アドレスデータ項目で指定します。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

（標準出力に出力できない場合）

規則

CGI リストに「名前」と「値」が格納されていない場合，「名前」と「値」は出力されません。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 INPUTLIST USAGE ADDRESS VALUE NULL.
       PROCEDURE DIVISION.
           CALL 'CBLCGIINIT' USING INPUTLIST.
           CALL 'CBLPRINTLIST' USING INPUTLIST.
            :
           CALL 'CBLDESTROYLIST' USING INPUTLIST.
```

HTML の出力例

```
<HTML>
   <BODY>
<H3> CGI list: </H3>
<DL>
<DT> name1
<DD> option1
<DT> name2
<DD> option2
</DL>
    </BODY>
</HTML>
```

(20) CBLSENDERROR

HTML 形式でエラーメッセージを出力するサービスルーチンです。

形式

```
CALL 'CBLSENDERROR' USING 引数1
```

引数

引数 1 には，出力するエラーメッセージが格納されている領域を，256 バイトの英数字項目で指定します。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

(標準出力に出力できない場合)

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ERROR-MESSAGE PIC X(256) VALUE 'ERROR STRING.'.
       PROCEDURE DIVISION.
            :
           CALL 'CBLSENDERROR' USING ERROR-MESSAGE.
```

HTML の出力例

```
<HTML>
   <HEAD>
     <TITLE>Error Message </TITLE>
   </HEAD>
   <BODY>
<BR>
<H2>!! Error Message from COBOL CGI program !!</H2>
<H3> ERROR STRING. </H3>
   </BODY>
 </HTML>
```

(21) CBLCGITRACE

CGI プログラムの作成を支援するサービスルーチンの呼び出しをトレースするサービスルーチンです。また，CGI 環境変数の値も取得します。トレース情報や環境変数の情報は，トレースファイル（実行可能ファイル名.trc）に出力されます。

トレース情報には，呼び出したサービスルーチンの名称と引数情報が出力されます。例えば，CBLCGIINIT では，CGI リストのアドレスが引数情報になります。

形式

```
CALL 'CBLCGITRACE'
```

引数

なし。

戻り値

なし。

規則

- CBLCGITRACE サービスルーチンは，CBLCGIINIT サービスルーチンよりも先に呼び出してください。
- CBLCGITRACE サービスルーチンが呼び出されると，トレースファイルは初期化されます。
- CBLCGIINIT サービスルーチンが呼び出されると，CGI 環境変数の値をトレースファイルに出力します。
- マルチスレッド対応 COBOL プログラムで CBLCGITRACE サービスルーチンを呼び出した場合，トレースファイルの出力内容は保証しません。
- 次に示すサービスルーチンは，CBLCGITRACE サービスルーチンでトレースされません。

  CBLDISPLAYTEXT
  CBLPRINTENV
  CBLPRINTLIST
  CBLSENDERROR
- CBLCGITRACE サービスルーチンが呼び出されたあとに CBLCGIINIT サービスルーチンが呼び出されると，CGI 環境変数の値がトレースファイルに出力されます。

出力されるトレース情報

各サービスルーチンを呼び出したときに出力されるトレース情報を，次に示します。

サービスルーチン	出力されるトレース情報
CBLHTMLBEGIN	<! CBLHTMLBEGIN(); Title = "文字列" >
CBLHTMLEND	<! CBLHTMLEND(); >
CBLDISPLAYTEXT	出力されない。

サービスルーチン	出力されるトレース情報
CBLCONVERTTEXT	<! CBLCONVERTTEXT(); input string = "文字列" >
CBLPRINTENV	出力されない。
CBLGETENV	<! CBLGETENV(); name = "環境変数名", value = "環境変数の値" >
CBLCGIINIT	CBLCGIINIT を初めて呼び出した場合（CGI リストのアドレスが NULL の場合） <! CBLCGIINIT(); list = "CGI リストのアドレス" >
	取得済み CGI リストのアドレスを指定した場合（CGI リストのアドレスが NULL 以外の場合） <! CBLCGIINIT(); list = second call (error) >
CBLCREATELIST	正常終了した場合 <! CBLCREATELIST(); list = " CGI リストのアドレス" >
	取得済み CGI リストのアドレスを指定した場合（CGI リストのアドレスが NULL 以外の場合） <! CBLCREATELIST(); list = second call (error) >
CBLDESTROYLIST	<! CBLDESTROYLIST(); list = " CGI リストのアドレス" >
CBLADDPAIR	<! CBLADDPAIR(); list = " CGI リストのアドレス", name = "名前", value = "値" >
CBLDELETEPAIR	<! CBLDELETEPAIR(); list = " CGI リストのアドレス" >
CBLFINDPAIR	正常終了した場合 <! CBLFINDPAIR(); list = " CGI リストのアドレス", name = "名前", value = "値" >
	検索した名前が見つからない場合 <! CBLFINDPAIR(); list = " CGI リストのアドレス" At end >
CBLFINDNEXTPAIR	正常終了した場合 <! CBLFINDNEXTPAIR(); list = " CGI リストのアドレス", name = "名前", value = "値" >
	検索した名前が見つからない場合 <! CBLFINDNEXTPAIR(); list = " CGI リストのアドレス" At end >
CBLGETPAIR	正常終了した場合 <! CBLGETPAIR(); list = " CGI リストのアドレス", name = "名前", value = "値" >
	名前／値を取得できない場合（空の CGI リスト，CGI リストのポインタが終端） <! CBLGETPAIR(); list = " CGI リストのアドレス" At end >
CBLGETPAIRNEXT	正常終了した場合 <! CBLGETPAIRNEXT(); list = " CGI リストのアドレス", name = "名前", value = "値" >
	名前／値を取得できない場合（空の CGI リスト，CGI リストのポインタが終端） <! CBLGETPAIRNEXT(); list = " CGI リストのアドレス" At end >

サービスルーチン	出力されるトレース情報
CBLLISTCOUNT	<! CBLLISTCOUNT(); list = " CGI リストのアドレス", count = "名前／値の数" >
CBLENDREPEAT	<! CBLENDREPEAT(); list = " CGI リストのアドレス" >
CBLFILLTEMPLATE	<! CBLFILLTEMPLATE(); list = " CGI リストのアドレス", filename = 'テンプレート名' >
CBLPRINTLIST	出力されない。
CBLSENDERROR	出力されない。
CBLCGITRACE	<! CBLCGITRACE(); YYYY-MM-DD (HH:MM:SS) > YYYY-MM-DD は年-月-日，HH:MM:SS は時:分:秒を示す。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 INPUTLIST USAGE ADDRESS VALUE NULL.
       01 NAMEX PIC X(28) VALUE 'COBOL'.
       01 VALUEX.
         02 VALUE-LEN PIC S9(9) USAGE COMP.
         02 VALUE-STRING.
           03 PIC X OCCURS 1024 DEPENDING ON VALUE-LEN.
       PROCEDURE DIVISION.
           CALL 'CBLCGITRACE'.
           CALL 'CBLCGIINIT' USING INPUTLIST.
           CALL 'CBLPRINTENV'.
           CALL 'CBLFINDPAIR' USING INPUTLIST NAMEX VALUEX.
            :
           CALL 'CBLDESTROYLIST' USING INPUTLIST.
```

トレースファイルの例

```
<! CBLCGITRACE();YYYY-MM-DD (HH:MM:SS)>

<! **** CGI ENVIRONMENT VARIABLES ***>
AUTH_TYPE= (null)
CONTENT_LENGTH= 0
 :
<! **** END OF CGI ENVIRONMENT VARIABLES ***>

<! CBLCGIINIT();list = 0x890980 >
 :
```

25.7.3　サービスルーチンに関する注意事項

(1)　エラーメッセージの出力に関する注意事項

- CBLFILLTEMPLATE サービスルーチンで，HTML テンプレートをインタプリットするとき，HTML 拡張言語のエラーメッセージの行情報がずれて表示されることがあります。

- CBLFILLTEMPLATE サービスルーチンで，インタプリット中にエラーが発生した場合，Web ブラウザによっては標準出力の HTTP ヘッダなどが文字列として表示されることがあります。

  CGI プログラムで実行時エラーが発生したときの Web ブラウザの表示例を，次に示します。

```
KCCC9713R-S
変数名の先頭文字が英字ではありません。
HTMLテンプレート名＝"/usr/local/httpd/cgi-bin/CGITEST.htm"
行番号=9                                                          ] 1.

KCCC1000R-I
プログラムが異常終了しました。
プログラム名＝CGITEST
行番号／欄＝000028/12
シグナル種別＝SIGIOT（abortによる異常終了）                       ] 2.

Content-type: text/html                                           ] 3.

COBOL2002(c)VV-RR  ***異常終了時要約情報リスト*** YYYY-MM-DD HH:MM:SS
最終実行文情報     プログラム名＝CGITEST
                   コンパイル日付（時間）＝YYYY-MM-DD(HH:MM:SS)
                   行番号＝000028.欄＝012                         ] 4.

シグナル種別＝SIGIOT（abortによる異常終了）
```

1. CBLFILLTEMPLATE サービスルーチンの COBOL 実行時エラーメッセージ
2. COBOL 実行時エラーメッセージ
3. CBLFILLTEMPLATE が標準出力した HTTP ヘッダ
4. 異常終了時要約情報リスト

エラーメッセージについては，マニュアル「COBOL2002　メッセージ」を参照してください。

(2)　CGI リストのアドレスに関する注意事項

CBLCGIINIT サービスルーチンおよび CBLCREATELIST サービスルーチンで取得した CGI リストのアドレス以外の不当なアドレスを，CGI プログラムの作成を支援するサービスルーチンに指定しないでください。不当なアドレスを指定した場合，動作は保証しません。

25.8　起動ファイルの作成方法

起動ファイルは，CGI プログラムの環境変数を設定し，CGI プログラムを起動するファイルです。起動ファイルを使用しないで，直接 CGI プログラムを起動することもできますが，Web サーバの環境によっては COBOL で指定した環境変数が無効になる場合があります。このような場合は，起動ファイルを使って環境変数を定義し，CGI プログラムを実行する必要があります。

起動ファイルの例を，次に示します。

sh（B シェル）を使用した起動ファイルの例

```
#!/bin/sh                          …1.
LIBPATH=/opt/HILNGcbl2k/lib
export LIBPATH                     …2.
CBL_SYSERR=./errlog.txt
export CBL_SYSERR                  …3.
./CGIMAIN                          …4.
unset CBL_SYSERR                   …5.
unset LIBPATH                      …6.
```

1. sh（B シェル）を起動する
2. COBOL2002 の共用ライブラリパスを環境変数 LIBPATH に設定する
3. 実行時エラーメッセージを出力するファイルを環境変数 CBL_SYSERR に設定する
4. CGI プログラムを起動する
5. 環境変数 CBL_SYSERR を削除する
6. 環境変数 LIBPATH を削除する

25.9 実行時エラーメッセージの取得方法

COBOL で作成した CGI プログラムで実行時エラーが発生した場合，Web サーバや Web ブラウザによって実行時エラーメッセージを表示する動作が異なります。CGI プログラムの実行時エラーメッセージや異常終了時要約情報リストを取得するには，起動ファイルに環境変数 CBL_SYSERR，および環境変数 CBLABNLST を設定し，エラーメッセージを取得してください。

システムに従った設定方法で，環境変数 CBL_SYSERR，および環境変数 CBLABNLST を指定すると，COBOL で作成した CGI プログラムのすべてに有効となり，複数の CGI プログラムでエラーが発生した場合，実行時エラーメッセージや異常終了時要約情報リストを出力するファイルを上書きする可能性があるため，注意して使用してください。

CGI プログラムのデバッグで Web ブラウザに実行時エラーメッセージを表示したい場合は，環境変数 CBLCGIERR に YES を指定します。環境変数 CBLCGIERR について次に示します。

形式

```
CBLCGIERR=YES
```

規則

- 環境変数 CBLCGIERR に YES を指定すると，実行時エラーメッセージの先頭に HTTP ヘッダの MIME フォーマットを表す"Content-type: text/plain¥n¥n"（以降，CGI ヘッダと呼びます）が付けられ，出力されます。ただし，COBOL 実行時ライブラリが環境変数を取得する前にエラーになった場合，この指定は無効です。
- 環境変数 CBLCGIERR に YES を指定しても，異常終了時要約情報リストには CGI ヘッダが付けられません。

注意事項

Web サーバや Web ブラウザによって実行時エラーメッセージが出力されていても，画面に表示されない場合があります。

25.10　CGI プログラムのデバッグ

ここでは，CGI プログラムをデバッグする方法について説明します。CGI プログラムをデバッグするには，Web サーバを使用しない方法と，Web サーバを使用する方法の 2 種類があります。

25.10.1　Web サーバを使用しない方法

デバッグの実行時に Web サーバを使用しない方法について説明します。

1. Web サーバ上にある CGI プログラムを，-TDInf オプションを指定してコンパイルする。
2. Web ブラウザから CGI プログラムを実行する。
   CGI プログラム中で CBLCGIINIT サービスルーチンが呼び出されると，サーバで処理した情報，およびクライアント（Web ブラウザ）から受け取った情報を含んだ CGI 環境ファイル（.env）が，CGI プログラムと同じディレクトリに作成されます。
   以降は，Web サーバを使用しないでデバッグできます。
3. CGI 環境ファイルの環境変数を実行環境ファイルに追加する。
4. 実行環境ファイルを CGI プログラムと同じディレクトリに格納する。
5. ラインモード，またはバッチモードのテストデバッガを起動し，CGI プログラムをデバッグする。

注意事項

デバッグ対象となる CGI プログラムがネットワーク上で実行される場合には，CGI 環境ファイル中に記述されている環境変数 PATH_TRANSLATED のドライブ文字を，対象となるドライブ文字に変更する必要があります。

25.10.2　Web サーバを使用する方法

Web サーバを使用するデバッグ方法について説明します。

1. CGI プログラムを-TDInf オプションを指定してコンパイルする。
2. 環境変数 CBLTDEXEC の指定を追加した実行環境ファイルを作成する。
   環境変数 CBLTDEXEC の詳細については，「35.3　プログラムの実行環境の設定」を参照してください。
3. サーバ上の Web ブラウザから，CGI プログラムを実行する。
   CGI プログラムが起動されると，ラインモードのテストデバッガが連動実行されます。
4. CGI プログラムをデバッグする。

注意事項

- 3.で，クライアント上の Web ブラウザからは CGI プログラムを起動できません。
  必ずサーバ上の Web ブラウザから起動してください。
- デバッグ時に，ユーザアカウントの相違によってファイル参照が拒否されることがあります。このようなときは，テストデバッガを起動したログオン ID でファイルが参照できるように設定する必要があります。
- 連動実行でのデバッグ時に，テストデバッガを表示するための端末が表示されないことがあります。この場合は，明示的に環境変数 CBLTDDISPLAY にウィンドウの表示先を設定する必要があります。環境変数 CBLTDDISPLAY の詳細については，マニュアル「COBOL2002　使用の手引　操作編」を参照してください。

25.11　注意事項

CGI プログラムを作成，実行する上で注意する点について説明します。

25.11.1　CGI プログラムを作成する場合の注意点

CGI プログラムの応答には，CGI 経由で Web ブラウザを使用することが前提となっています。したがって，画面出力や応答メッセージなどの，入力待ち状態が発生するプログラムを作成してはいけません。

次のような機能は，入力待ち状態が発生するので，CGI プログラムでは使用しないでください。

- 入力待ち状態が発生する ACCEPT 文
- STOP　定数文
- 画面機能（SCREEN SECTION，WINDOW SECTION）
- サービスルーチン（COBOL の実行を制御する CBLABN,CBLEND サービスルーチンは除く）
- 入力待ち状態が発生するプログラムの呼び出し

25.11.2　出力文字に関する注意点

半角かたかな文字を出力する CGI プログラムを作成しないでください。Web ブラウザによっては，文字が正しく表示されないことがあります。

26 マルチスレッド環境での実行

マルチスレッド環境に対応する COBOL プログラムは，複数スレッドで同時に実行できます。この章では，マルチスレッド環境に対応した COBOL プログラムの作成方法や，固有の機能について説明します。

26.1　マルチスレッド対応 COBOL プログラムの概要

マルチスレッド対応 COBOL プログラムとは，マルチスレッド環境下で，複数スレッドで同時に実行できる COBOL プログラムのことです。

ただし，マルチスレッド対応 COBOL プログラム機能は，マルチスレッド環境自体を作成するものではありません。

マルチスレッド対応 COBOL プログラムには，制限や注意点があります。

26.1.1　スレッドの制御

スレッドの起動やスレッド間の同期制御など，スレッド制御に関する操作は，COBOL プログラムで実行できません。

26.1.2　実行単位

マルチスレッド対応 COBOL プログラムの実行単位は，スレッドになります。各スレッドはそれぞれに COBOL 実行環境を持ち，独立して動作しています。

26.1.3　データの共用

COBOL データ項目の領域は，スレッドごとに割り当てられます。

COBOL データは，同じスレッド内に限り共用できます。したがって，ほかのスレッドの COBOL データは共用できません。また，同じスレッド内であっても，C プログラムのデータは共用できません。

26.1.4　プログラムのコーディング

マルチスレッド対応 COBOL プログラムは，通常の COBOL プログラムを作成するときと同じようにコーディングできます。特殊な命令文は使用しません。

26.1.5　スレッド制御方式の指定

マルチスレッド対応 COBOL プログラムを作成するときには，システムごとのスレッド制御方式に対応したスレッド関数ライブラリをリンクする必要があります。各システムで使用できるスレッド制御方式，およびスレッド関数のリンク方法については，「26.2.2　マルチスレッド対応 COBOL プログラムのリンク」を参照してください。

26.2　マルチスレッド対応 COBOL プログラムの生成

26.2.1　マルチスレッド対応 COBOL プログラムのコンパイル

COBOL プログラムをマルチスレッド対応 COBOL プログラムにするためには，-MultiThread オプションを指定してプログラムをコンパイルします。

今まで通常のシングルスレッド対応 COBOL プログラムとして利用してきたプログラムも，-MultiThread オプションを指定して再コンパイルすれば，そのままマルチスレッド環境で実行できるようになります。ただし，マルチスレッド対応 COBOL プログラムで対応していない機能を使用しているプログラムは除きます。詳細は，「26.5　マルチスレッド対応 COBOL プログラムが対応している機能」を参照してください。

注意事項

- マルチスレッド対応 COBOL プログラムで対応していない機能を使用したプログラムを，-MultiThread オプションを指定してコンパイルしても，実行時の動作は保証しません。
- -MultiThread オプションを使ってコンパイルする場合，プロセス内で動作するすべての COBOL プログラムを，-MultiThread オプションを使ってコンパイルする必要があります。プロセス内に，-MultiThread オプション指定有りでコンパイルしたプログラムと，オプション指定無しでコンパイルしたプログラムが混在していると，実行時の動作は保証しません。
- マルチスレッド対応 COBOL プログラムを ccbl2002 コマンドでリンクする場合，必ず-MultiThread オプションを指定しなければなりません。

26.2.2　マルチスレッド対応 COBOL プログラムのリンク

マルチスレッド対応 COBOL プログラムは，ccbl2002 コマンド，または cc／ld コマンドでリンクします。

リンク時には，システムごとのスレッド制御方式に対応したスレッド関数ライブラリを指定する必要があります。また，cc／ld コマンドを使用してリンクするときは，さらに，COBOL2002 スレッド関数インタフェースライブラリを指定する必要があります。

なお，cc／ld コマンドを使用して実行可能ファイルを生成する方法については，「34.1.5　cc コマンドおよび ld コマンドの-l オプション」を参照してください。

(1)　マルチスレッド対応 COBOL プログラムのリンク時に指定するライブラリ

マルチスレッド対応 COBOL プログラムで使用するスレッド関数ライブラリと COBOL2002 スレッド関数インタフェースライブラリとの対応を次に示します。

使用できる OS	スレッド関数ライブラリ	COBOL2002 スレッド関数 インタフェースライブラリ
AIX(32)	libpthreads.a	libcbl2kmp.a
AIX(64)	libpthreads.a	libcbl2kmp64.a
Linux	libpthread.so	libcbl2kmp.so

(2)　リンク方法

(a)　AIX(32)の場合

AIX(32)の場合，スレッド関数ライブラリには libpthreads.a を，COBOL2002 スレッド関数インタフェースライブラリには libcbl2kmp.a を使用します。指定するオプション，および指定例を次に示します。

ccbl2002 コマンドを使用する場合

-MultiThread，-lpthreads オプションを指定します。

（例）

```
ccbl2002 -MultiThread  cmain.o SUBCBL.o -lpthreads
```

cc／ld コマンドを使用する場合

-lcbl2kmp，-lpthreads オプションを指定します。

（例）

```
cc cmain.o SUBCBL.o -L/opt/HILNGcbl2k/lib -lcbl2k
   -lcbl2kml -lcbl2kmp -lpthreads -lm
```

(b)　AIX(64)の場合

AIX(64)の場合，スレッド関数ライブラリには libpthreads.a を，COBOL2002 スレッド関数インタフェースライブラリには libcbl2kmp64.a を使用します。指定するオプション，および指定例を次に示します。

ccbl2002 コマンドを使用する場合

-MultiThread，-lpthreads オプションを指定します。

（例）

```
ccbl2002 -MultiThread  cmain.o SUBCBL.o -lpthreads
```

cc／ld コマンドを使用する場合

-lcbl2kmp64，-lpthreads オプションを指定します。

（例）

```
cc cmain.o SUBCBL.o -q64 -L/opt/HILNGcbl2k64/lib -lcbl2k64
   -lcbl2kml64 -lcbl2kmp64 -lpthreads -lm
```

(c) Linux の場合

Linux の場合，スレッド関数ライブラリには libpthread.so を，COBOL2002 スレッド関数インタフェースライブラリには libcbl2kmp.so を使用します。指定するオプション，および指定例を次に示します。

ccbl2002 コマンドを使用する場合

-MultiThread，-lpthread オプションを指定します。

（例）

```
ccbl2002 -MultiThread -UniObjGen cmain.o SUBCBL.o -lpthread
```

cc／ld コマンドを使用する場合

-lcbl2kmp，-lpthread オプションを指定します。

（例）Linux(x86)の場合

```
cc cmain.o SUBCBL.o -L/opt/HILNGcbl2k/lib -lcbl2k
   -lcbl2kml -lcbl2kmp -lpthread -lm
```

（例）Linux(x64)の場合

```
cc cmain.o SUBCBL.o -L/opt/HILNGcbl2k64/lib -lcbl2k
   -lcbl2kml -lcbl2kmp -lpthread -lm
```

(3) 注意事項

- 次のようなリンクをした場合，生成された実行可能ファイルの動作は保証しません。
  - 指定したライブラリの組み合わせが不正である
  - 必要なライブラリをリンクしなかった
  - 必要なオプションを指定しなかった
- cc／ld コマンドを使用したリンクで，システムのスレッドライブラリを指定する場合は，ほかのシステムライブラリとの指定順序に制約がないか注意が必要です。詳細は，システムのリファレンスで確認してください。
- Linux では，スレッド関数ライブラリ libpthread.so をリンクしないと，プログラムの実行時にメッセージが出力され，実行が中止されます。

26.3 整列併合機能を使用したマルチスレッド対応 COBOL プログラムのリンク

整列併合機能を使用したマルチスレッド対応 COBOL プログラムは，次の形式でリンクします。

AIX(32)

```
ccbl2002 -MultiThread tp.cbl -L/opt/HISORTlib/lib -lmsort -lpthreads
```

-lmsort

マルチスレッド対応整列併合ライブラリ（libmsort.a）を使用するために指定します。

-lpthreads

AIX(32)が提供するスレッドライブラリを使用するために指定します。

AIX(64)

```
ccbl2002 -MultiThread tp.cbl -L/opt/HISORTlib64/lib -lmsort64 -lpthreads
```

-lmsort64

マルチスレッド対応整列併合ライブラリ（libmsort64.a）を使用するために指定します。

-lpthreads

AIX(64)が提供するスレッドライブラリを使用するために指定します。

Linux(x86)

```
ccbl2002 -MultiThread -UniObjGen tp.cbl -L/opt/HISORTlib/lib -lmsort -lpthread
```

-lmsort

マルチスレッド対応整列併合ライブラリ（libmsort.so）を使用するために指定します。

-lpthread

Linux(x86)が提供するスレッドライブラリを使用するために指定します。

Linux(x64)

```
ccbl2002 -MultiThread -UniObjGen tp.cbl -L/opt/HISORTlib64/lib -lmsort64 -lpthread
```

-lmsort64

マルチスレッド対応整列併合ライブラリ（libmsort64.so）を使用するために指定します。

-lpthread

Linux(x64)が提供するスレッドライブラリを使用するために指定します。

26.4 索引ファイル機能を使用したマルチスレッド対応 COBOL プログラムのリンク

索引ファイル機能を使用したマルチスレッド対応 COBOL プログラムは，次の形式でリンクします。

AIX(32)

```
ccbl2002 -MultiThread tp.cbl -L/opt/HIISlib/lib -lmisam -L/opt/HISORTlib/lib -lmsort -lpt
hreads
```

-lmisam

マルチスレッド対応索引ファイルライブラリ（libmisam.a）を使用するために指定します。

-lmsort

マルチスレッド対応整列併合ライブラリ（libmsort.a）を使用するために指定します。

-lpthreads

AIX(32)が提供するスレッドライブラリを使用するために指定します。

AIX(64)

```
ccbl2002 -MultiThread tp.cbl -L/opt/HIISlib64/lib -lmisam64 -L/opt/HISORTlib64/lib -lmsor
t64 -lpthreads
```

-lmisam64

マルチスレッド対応索引ファイルライブラリ（libmisam64.a）を使用するために指定します。

-lmsort64

マルチスレッド対応整列併合ライブラリ（libmsort64.a）を使用するために指定します。

-lpthreads

AIX(64)が提供するスレッドライブラリを使用するために指定します。

Linux(x86)

```
ccbl2002 -MultiThread -UniObjGen tp.cbl -L/opt/HIISlib/lib -lmisam -L/opt/HISORTlib/lib -
lmsort -lpthread
```

-lmisam

マルチスレッド対応索引ファイルライブラリ（libmisam.so）を使用するために指定します。

-lmsort

マルチスレッド対応整列併合ライブラリ（libmsort.so）を使用するために指定します。

-lpthread

Linux(x86)が提供するスレッドライブラリを使用するために指定します。

Linux(x64)

```
ccbl2002 -MultiThread -UniObjGen tp.cbl -L/opt/HIISlib64/lib -lmisam64 -L/opt/HISORTlib64
/lib -lmsort64 -lpthread
```

-lmisam64

マルチスレッド対応索引ファイルライブラリ（libmisam64.so）を使用するために指定します。

-lmsort64

マルチスレッド対応整列併合ライブラリ（libmsort64.so）を使用するために指定します。

-lpthread

Linux(x64)が提供するスレッドライブラリを使用するために指定します。

26.5　マルチスレッド対応 COBOL プログラムが対応している機能

マルチスレッド対応 COBOL プログラムが対応している機能を，次に示します。対応していない機能をマルチスレッド対応 COBOL プログラムに適用しないよう注意してください。

表 26–1　マルチスレッド対応 COBOL プログラムが対応する機能一覧

種類	機能名		マルチスレッドへの対応	備考
規格	基本機能		○	
	順編成ファイル		×※1	
	相対編成ファイル		×※1	
	索引編成ファイル		○	
	整列併合		△※1	USING／GIVING 指定に対して，マルチスレッドで使用できないファイル編成は指定できない。
	プログラム間連絡		○	
	組み込み関数		○	
	オブジェクト指向		○	
	共通例外処理		○	
	再帰呼び出し		○	
	利用者定義関数		○	
	局所場所節（LOCAL-STORAGE SECTION）		○	
X/Open	テキスト編成ファイル		×※1	
	ファイル共用（ファイルシェア）	索引編成ファイル以外	×※1	
		索引編成ファイル	○※1	同一スレッド内でのファイル共用は使用できない。
	コマンド行および環境変数へのアクセス		△	環境変数アクセスだけで使用できる。
	画面節（SCREEN SECTION）による画面操作※4		×※2	
	C 言語インタフェース		△	データの共用はできない。
	インタナショナリゼーション		○	
拡張機能	日本語		○	
	ブール（ビット操作）		○	
	アドレス操作		○	

種類	機能名	マルチスレッドへの対応	備考
	1 バイト 2 進機能・COMP-X 項目	○	
	浮動小数点項目	○	
	報告書作成機能	×※1	
	HiRDB による索引編成ファイル※4	×※1	
	CSV 編成ファイル	×※1	
	ラージファイル入出力	△※1	使用するファイル編成がマルチスレッドに対応していれば使用できる。
	ファイル入出力拡張機能	×※1	
	プリンタへのアクセス（入出力による書式印刷機能）※5	×※1	
	ファイルのディスク書き込み保証	○	
	通信節による画面操作※5	×※1	
	画面節（WINDOW SECTION）による画面操作※4	×※2	
	データコミュニケーション機能	×※1	
	データベース操作機能※8	△※1	連携しているドライバおよびデータベースが，マルチスレッドに対応している必要がある。
	XDM によるデータベースシミュレーション機能	○	
	基本機能サービスルーチン	△	実行できないサービスルーチンがある。詳細は「26.10.2　呼び出してはいけないサービスルーチン」を参照のこと。
	CGI プログラム作成支援機能※5	△	実行できないサービスルーチンがある。詳細は「25.7　CGI プログラムの作成を支援するサービスルーチン」を参照のこと。
	COBOL 入出力サービスルーチン	×	
	バイトストリーム入出力サービスルーチン	×	
	Unicode 機能	○	
	数字項目のけた拡張機能※6	○	
	動的長基本項目機能	○	
	定数長拡張機能	○	

種類	機能名	マルチスレッドへの対応	備考
デバッグ	実行時デバッグ機能	○	
	テストデバッグ機能※3	△	
	カバレージ機能※7	△	
開発／実行環境	プロファイル機能	×	プロファイル機能オプション（-Profile）は，-MultiThread オプションと同時に指定した場合，無視される。

（凡例）

○：使用できる

△：制限付きで使用できる

×：使用できない

注※1

コンパイル時，この機能を使用する文に対して W レベル（警告エラー）のメッセージが出力されます。

注※2

コンパイル時，この機能を使用する文に対して S レベル（重大エラー）のメッセージが出力され，コンパイルは中止します。

注※3

AIX で，ユーザスレッド，カーネルスレッドが 1：1 のときだけ有効です。

注※4

AIX で有効です。

注※5

AIX(32)で有効です。

注※6

AIX(64)，Linux(x64)で有効です。

注※7

AIX では，ユーザスレッド，カーネルスレッドが 1：1 のときだけ有効です。

Linux では，カバレージ情報の蓄積だけできます。

注※8

Linux で有効です。

26.6　マルチスレッド対応 COBOL プログラムの開始と終了

マルチスレッドプログラムの実行を開始する方法には，次の三つがあります。

- COBOL 以外のプログラムからの呼び出しによる方法
- マルチスレッド対応 COBOL プログラムをスレッド開始関数として指定する方法
- マルチスレッド対応 COBOL プログラムをアプリケーションの主プログラムにする方法

それぞれの開始方法と終了方法について説明します。

26.6.1　COBOL 以外のプログラムからの呼び出しによる方法

COBOL プログラム以外のプログラムを入り口としてスレッドを開始して，そのプログラムから間接的にマルチスレッド対応 COBOL プログラムを呼び出せます。

この方法では，スレッド内で最初に呼ばれる COBOL プログラムが，COBOL 主プログラムの場合と，COBOL 副プログラムの場合とで，プログラム終了時の動作が異なります。COBOL 主プログラムと副プログラムについては，「18.3　COBOL 主プログラムと副プログラム」を参照してください。

(1)　COBOL 主プログラムの場合

-MainNotCBL オプションを指定しないでコンパイルしたプログラムがスレッド内で最初に実行された場合，COBOL 主プログラムになります。

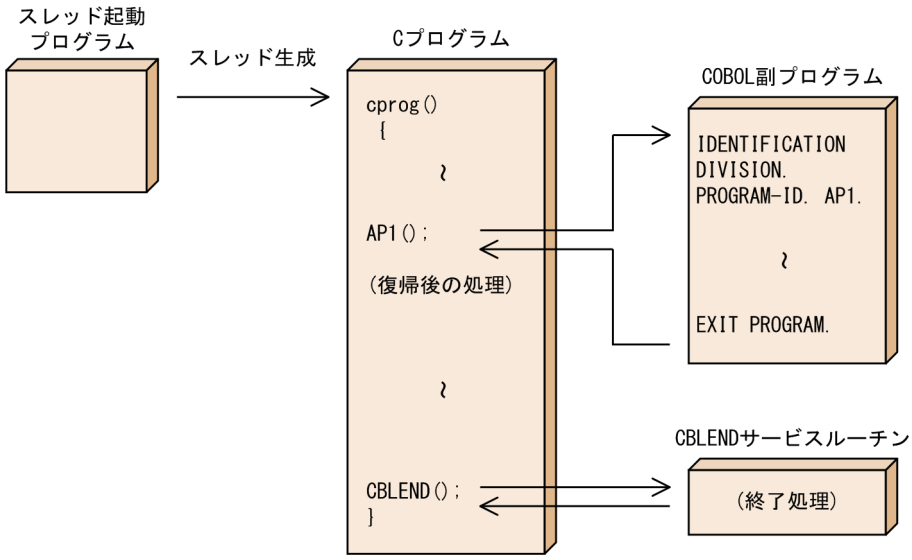
COBOL 主プログラムを終了させるには，STOP RUN 文を実行します。これによって，COBOL 実行環境が終了し，スレッドが終了されます。このとき，RETURN-CODE 特殊レジスタの値がスレッドの終了コードになります。

(2)　COBOL 副プログラムの場合

-MainNotCBL オプションを指定してコンパイルしたプログラムは，スレッド内で最初に実行した場合でも，常に COBOL 副プログラムとなります。

COBOL 副プログラムは，呼び出し元のプログラムに復帰できます。このとき，COBOL 実行環境は保持されます。そのため，呼び出し元のプログラムは，スレッドが終了する前に COBOL 実行環境を終了させる必要があります。COBOL 実行環境を終了させるには，CBLEND サービスルーチンを呼び出します。

図 26-1　COBOL 以外のプログラムからの呼び出しの例



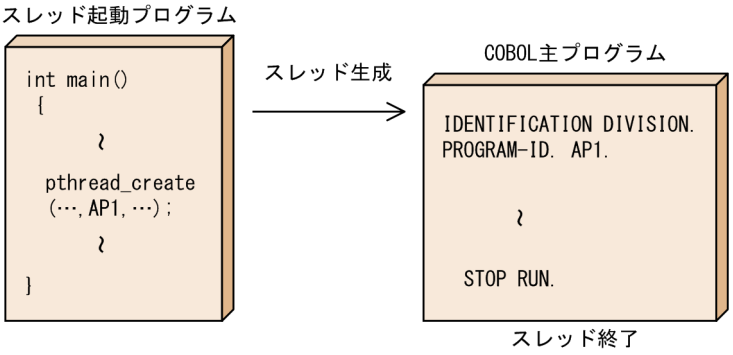
26.6.2　マルチスレッド対応 COBOL プログラムをスレッド開始関数として指定する方法

マルチスレッド対応 COBOL プログラムを入り口としてスレッドを開始した場合，このプログラムが終了したときに COBOL 実行環境を終了し，スレッドが終了します。このとき，RETURN-CODE 特殊レジスタに指定した値が，スレッドの終了コードになります。

注意事項

この方法を用いる場合は，プログラムをコンパイルする時に-MainNotCBL オプションを指定しないでください。

図 26-2　COBOL プログラムがスレッド開始関数となる例



26.6.3　マルチスレッド対応 COBOL プログラムをアプリケーションの主プログラムにする方法

マルチスレッド対応 COBOL プログラムに対して，-Main,System または-Main,V3 オプションを指定してコンパイルした COBOL プログラムは，アプリケーションの主プログラムになります。この場合，COBOL プログラムがプライマリスレッドになります。

マルチスレッド対応 COBOL プログラムがアプリケーションの主プログラムである場合，マルチスレッド対応 COBOL プログラムが終了したときに，プライマリスレッドが終了し，プロセスも終了します。

なお，コンパイル時に例外を検知するオプション（-DebugInf オプションなど）を指定した場合，プライマリスレッド内で発生した例外を取得できます。

26.6.4　戻り文に対する動作

それぞれのマルチスレッド対応 COBOL プログラムの実行方法について，戻り文に対する動作の違いを，次に示します。

COBOL プログラムの種類	STOP RUN 文	GOBACK 文	EXIT PROGRAM 文
COBOL 主プログラム	STOP RUN 文を実行したスレッドが終了する。	GOBACK 文を実行したスレッドが終了する。	何も動作しない。
COBOL 副プログラム	STOP RUN 文を実行したスレッドが終了する。	呼び出し元に制御が戻る。	呼び出し元に制御が戻る。

26.7　実行時エラーが発生したときの動作

実行時エラーが検知された場合は，実行時エラーメッセージが出力されたあと，プログラムの動作していたスレッドが終了されます。実行時メッセージの形式は次のようになります。

形式

```
KCCCnnnnR-x(i) メッセージテキスト
```

nnnn
　メッセージ番号

x
　メッセージレベル

i
　スレッド識別子の値

メッセージ番号，メッセージレベルなど，実行時メッセージの形式については，マニュアル「COBOL2002　メッセージ」を参照してください。

注意事項

- COBOL 実行時メッセージなどの COBOL2002 が出力する情報は，スレッド識別子が付けられます。これによって対応するスレッドを識別できます。スレッド識別子には，pthread_self 関数が返すスレッド ID を用います。
- マルチスレッド対応 COBOL プログラム実行時の初期処理，または終了処理中にエラーが発生した場合，KCCC03nnR-S などのメッセージ番号が 03nn で示される 300 番台の実行時メッセージが，英語で出力されます。なお，この場合，異常終了時要約情報リストは出力されません。

26.8　環境変数の取り扱い

マルチスレッド環境で環境変数を扱う場合，設定値がすべてのスレッドで共用されるので注意してください。

マルチスレッド対応 COBOL プログラムでは，環境変数を扱う上での混乱を避けるため，次の二つの機能を持っています。

- スレッドごとに固有の出力ファイル名称を付ける機能
- スレッドごとに環境変数を設定する機能

26.8.1　スレッドごとに固有の出力ファイル名称を付ける機能

出力ファイル名を指定する実行時環境変数の設定がある場合，この機能が実行されます。この機能は，実行時環境変数によって指定したファイル名へ，自動的にスレッドの識別子を付けます。これによって，スレッドごとに固有のファイルが出力されることになります。

形式

```
ファイル名_i.拡張子
```

i

スレッド識別子の値

注意事項

この機能の対象となる実行時環境変数を次に示します。

- CBL_SYSOUT
- CBL_SYSPUNCH
- CBL_SYSERR
- CBLABNLST
- CBLDDUMP
- CBLDATADUMPFILE
- CBLPGMSEARCHTRC

ただし，CUI モードのときに，出力ファイル名に標準入力（stdin）,標準出力（stdout），または標準エラー出力（stderr）が指定された場合，この機能の対象になりません。

これらの実行時環境変数の詳細は，「35.3　プログラムの実行環境の設定」を参照してください。

26.8.2　スレッドごとに環境変数を設定する機能

マルチスレッド対応 COBOL プログラムでは，環境変数へのアクセス機能を使えば，スレッドごとに環境変数の値を設定・取得できます。環境変数へのアクセス機能の詳細については，「10.4　環境変数へのアクセス」を参照してください。

規則

- 環境変数の値を設定・変更した場合，その値は実行スレッド内だけで有効となります。
- 同一の環境変数に対して，スレッドごとに別々の値を設定できます。
- 設定した環境変数は，COBOL 以外のプログラムで使用できません。また，PATH などのシステム環境変数や，COBOL2002 以外が管理する環境変数の値を設定・変更しても有効になりません。
- この機能を使って，「26.8.1　スレッドごとに固有の出力ファイル名称を付ける機能」で示した環境変数を設定した場合，この機能が優先され，出力ファイル名にスレッド識別子は付けられません。

  マルチスレッド対応 COBOL プログラムで環境変数へのアクセスを使用した場合，内部的に環境変数名にスレッド識別子の値を付けた環境変数名で値を登録および取得します。

  スレッド識別子が 1000 で環境変数 CBLABNLST を設定する例を次に示します。

（例）

マルチスレッド対応 COBOL プログラムで，CBLABNLST=a.txt を設定したとき，次のように環境変数を登録および取得します。

1. スレッド識別子を付けた環境変数名を使用して，「CBLABNLST_1000=a.txt」を値に登録します。
2. マルチスレッド対応 COBOL プログラムで，環境変数を取得する場合，次の順序で環境変数名をアクセスし，取得した値を返します。
   1. スレッド識別子を付けた環境変数名（CBLABNLST_1000）
   2. スレッド識別子を付けない環境変数名（CBLABNLST）

このため，環境変数の値をクリアしないで COBOL プログラムの実行を終了し，あとで同じスレッド識別子のスレッドで COBOL プログラムが動作した場合，そのスレッドで環境変数を設定していなくても，以前のスレッドで設定した環境変数の値が取得されます。

COBOL プログラムを終了する前に NULL（X'00'）で始まる値を設定することで，スレッド識別子を付けた環境変数に対して値のない環境変数を設定できます※。

注※

　値のない環境変数を設定したあとに環境変数へのアクセス機能で環境変数の値の読み込みを行った場合，COBOL はスレッド識別子を付けない環境変数名（CBLABNLST）にアクセスし，取得した値を返します。

注意事項

マルチスレッド環境下で動作するプログラムでは，環境変数の設定中に，ほかのスレッドで環境変数を設定または取得することは，OS で保証されていません。環境変数の設定中に，同時に別のスレッドで

環境変数アクセス関数※の処理が実行された場合，環境変数アクセス関数がエラーリターンしたり，シグナルが発生したりすることでプログラムが異常終了することがあります。

DISPLAY 文または C 言語プログラムで環境変数を設定する場合，環境変数アクセス関数がスレッド間で同時に実行されないようにしてください。

注※

putenv 関数，または getenv 関数

26.9　マルチスレッド対応 COBOL プログラムのデバッグ

26.9.1　マルチスレッド対応 COBOL プログラムのデバッグ

(1)　テストデバッガを使用したデバッグ

COBOL プログラムが実行されたすべてのスレッドに対して，テスト，およびデバッグができます。詳細は，マニュアル「COBOL2002　使用の手引　操作編」を参照してください。

(2)　カバレージ機能

COBOL プログラムが実行されたすべてのスレッドに対して，カバレージ情報の採取，およびカウント情報の表示ができます。詳細は，マニュアル「COBOL2002　使用の手引　操作編」を参照してください。

26.9.2　実行時デバッグ機能

デバッグ用のオプション（-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange）を指定しコンパイルしたマルチスレッド対応 COBOL プログラムが，実行時に異常終了，またはエラーが発生した場合，異常終了時要約情報が出力された後，スレッドが終了されます。

(1)　シグナルの登録

マルチスレッド対応 COBOL プログラムでは，シグナルの登録に sigaction システムコールを使用します。登録するシグナルに関する詳細は，「36.9　シグナル」を参照してください。

(2)　シグナルの種別

マルチスレッド対応 COBOL プログラムでは，異常終了時要約情報リスト中のシグナル種別として以下を追加する。

- RUNTIME ERROR（実行時エラーの発生）
- CBLABN（CBLABN サービスルーチンの呼び出し）

なお，-MultiThread オプションの指定がない COBOL プログラムで，実行時エラーが発生したり，CBLABN サービスルーチンが呼び出されたりした場合には，シグナル種別は SIGIOT（abort による異常終了）となります。

(3)　シグナルに関する注意事項

- アプリケーションプログラムを実行中に，C プログラムでシグナルの動作を指示するシステムコールが発行された場合，異常終了時の結果は保証しません。

- マルチスレッド対応 COBOL プログラムの実行時に次の理由で終了した場合，コアダンプは出力されません。
  - COBOL がメッセージを出力して実行を中止したとき
  - デバッグ用のオプション（-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange）を指定して異常終了したとき（環境変数 CBLCORE は無効となります）
  - CBLABN サービスルーチンを実行したとき
- マルチスレッド対応 COBOL プログラムでのシステムのシグナル処理は，プロセス単位の管理となっており，シグナルの登録，回復は，プロセス内すべてのスレッドに影響があります。シグナルの詳細は，「36.9　シグナル」を参照してください。

26.10　マルチスレッド対応 COBOL プログラムを使用する上での注意事項

26.10.1　EXTERNAL 句を用いたデータの共用

EXTERNAL 句を指定した COBOL データ項目は，同じスレッド内のプログラムでだけ共用できます。異なるスレッドの間では，COBOL データ項目を共用できません。

また，COBOL プログラムと C プログラムとの間では，同じスレッド内であってもデータを共用できません。

26.10.2　呼び出してはいけないサービスルーチン

- マルチスレッド対応 COBOL プログラムから，次のサービスルーチンを呼び出さないでください。呼び出した場合，動作は保証しません。
  - JCPOPUP サービスルーチン
- マルチスレッド対応 COBOL プログラム，およびマルチスレッド環境下で動作する他言語のプログラムから COBOL 入出力サービスルーチンを呼び出さないでください。呼び出した場合，動作は保証しません。

26.10.3　共用ライブラリがメモリ上から削除される条件

マルチスレッド対応 COBOL プログラムで CANCEL 文を実行しても，共用ライブラリはメモリ上から削除されません。

また，STOP RUN 文などによるスレッド実行環境の終了処理の場合，共用ライブラリはメモリ上から削除されません。この場合，共用ライブラリは，プロセスの終了までメモリ上に読み込まれたままの状態になります。

27 Unicode 機能

Unicode 機能は，COBOL が扱うデータを Unicode として扱う機能です。Unicode 機能を使用すると，プログラム間およびファイル入出力を Unicode でやり取りできます。この章では，Unicode 機能について説明します。

27.1　Unicode 機能の概要

ここでは，コンパイル，実行，およびデバッグでの Unicode 機能の概要について説明します。

Unicode 機能を使用できる環境を次に示します。

表 27–1　Unicode 機能を使用できる環境

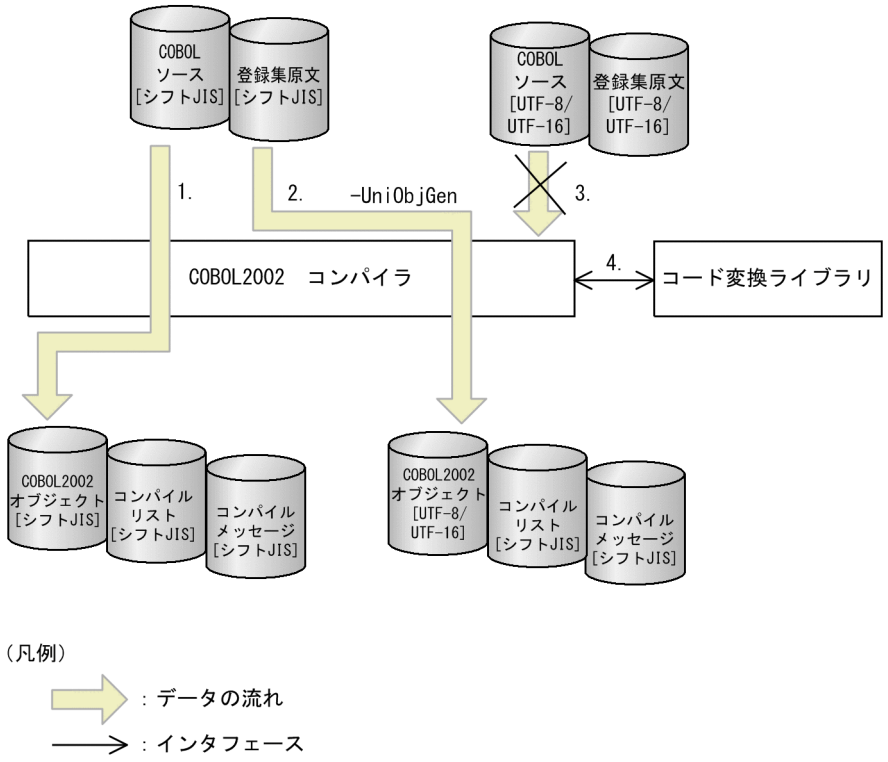
OS	使用できる動作環境	環境変数 LANG の設定値
AIX	シフト JIS 環境	「付録 A.2　シフト JIS の場合」を参照してください。
Linux	UTF-8 環境	「付録 A.4　Unicode の場合」を参照してください。

27.1.1　コンパイルでの Unicode 機能

シフト JIS で記述された COBOL ソースプログラムを，-UniObjGen オプションを指定してコンパイルすることで，コード系が Unicode のオブジェクトを生成します。これによって，Unicode データ同士の転記または比較ができます。コンパイルでの Unicode 機能を次に示します。

(1)　シフト JIS 環境下の場合

図 27–1　コンパイルでの Unicode 機能（シフト JIS 環境下）



1. シフト JIS で記述された COBOL ソースプログラム，登録集原文を入力し，-UniObjGen オプションを指定しないでコンパイルすると，シフト JIS 環境で動作するオブジェクトが生成されます。

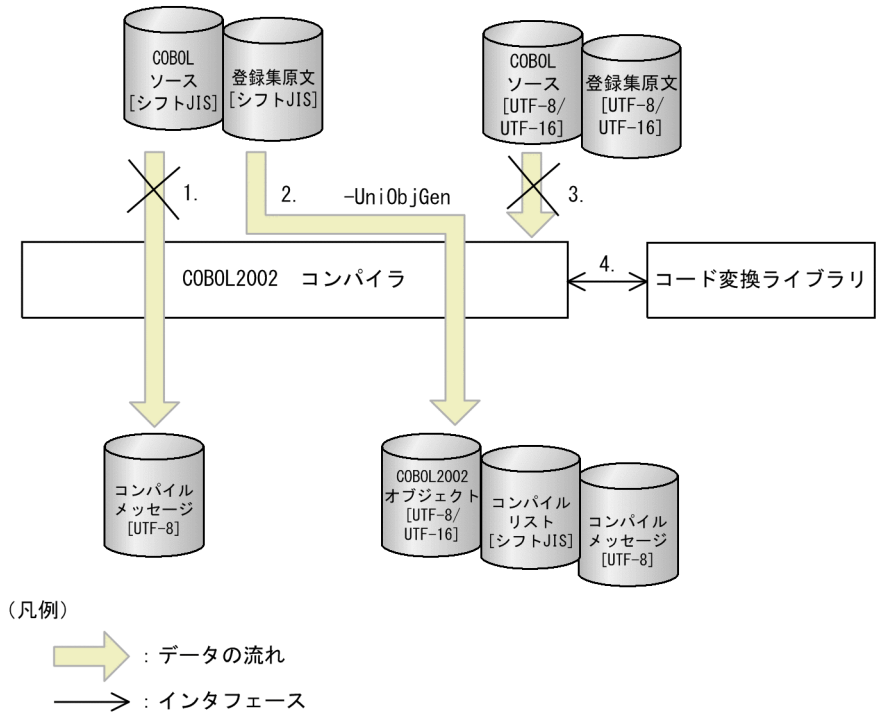
2. シフト JIS で記述された COBOL ソースプログラム，登録集原文を入力し，-UniObjGen オプションを指定してコンパイルすると，COBOL の Unicode 機能を使用する環境で動作するオブジェクトが生成されます。コンパイル時に出力されるコンパイルメッセージ，コンパイルリストはシフト JIS で出力されます。

3. Unicode で記述された COBOL ソースプログラムは，コンパイルできません。コンパイルした場合，動作は保証しません。

4. COBOL2002 コンパイラは，COBOL ソースプログラム上に記述された英数字文字定数を UTF-8 に，日本語文字定数を UTF-16 に変換します。

(2) UTF-8 環境下の場合

図 27–2 コンパイルでの Unicode 機能（UTF-8 環境下）



1. シフト JIS で記述された COBOL ソースプログラム，登録集原文を入力し，-UniObjGen オプションを指定しないでコンパイルすると，コンパイルエラーとなります。

2. シフト JIS で記述された COBOL ソースプログラム，登録集原文を入力し，-UniObjGen オプションを指定してコンパイルすると，COBOL の Unicode 機能を使用する環境で動作するオブジェクトが生成されます。コンパイル時に出力されるコンパイルメッセージは UTF-8，コンパイルリストはシフト JIS で出力されます。

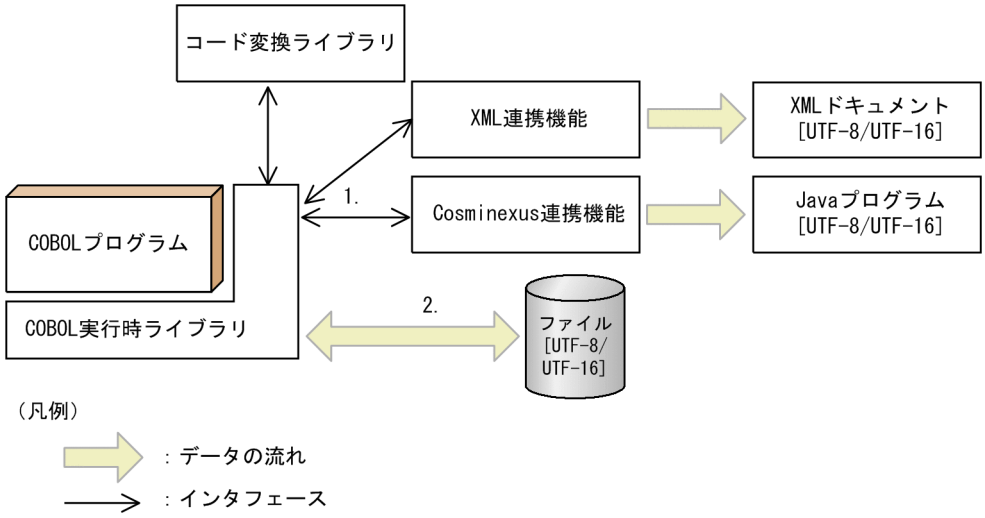
3. Unicode で記述された COBOL ソースプログラムは，コンパイルできません。コンパイルした場合，動作は保証しません。

4. COBOL2002 コンパイラは，COBOL ソースプログラム上に記述された英数字文字定数を UTF-8 に，日本語文字定数を UTF-16 に変換します。

27.1.2　実行での Unicode 機能

プログラム実行時に実行時環境変数 CBLLANG に UNICODE が指定されている場合，コード系が Unicode とみなして実行します。プログラム実行時の Unicode 機能について次に示します。

図 27-3　プログラム実行時の Unicode 機能



1. XML 連携機能と Cosminexus 連携機能※で Unicode データを使用できます。

   注※

   AIX(64)，Linux(x64)で有効です。

2. ファイル入出力機能で Unicode データの読み書きができます。

27.1.3　デバッグでの Unicode 機能

テストデバッガでプログラムをデバッグすると，Unicode データに対して次の操作ができます。詳細は，マニュアル「COBOL2002　使用の手引　操作編」を参照してください。

- データ項目の値の表示
- データ項目への値の代入
- 比較条件式の設定

27.2 Unicode 機能のサポート範囲

Unicode 機能で使用できる Unicode の文字は，UCS-4 の範囲（UCS-2 の範囲を含む）とします。

このシステムで異体字セレクタとみなすコード範囲は，U+E0100～U+E01EF とします。

27.3　Unicode 機能の前提条件

Unicode 機能を使用する場合の前提条件を次に示します。

27.3.1　コード変換ライブラリ

Unicode 機能を使用する場合，次のプログラムプロダクト（以降これらのプログラムプロダクトをコード変換ライブラリと表記します）のどれかをインストールしてください。

- AIX(32)，Linux(x86)の場合
  - 日立コード変換 - Runtime
  - Hitachi Code Converter - Runtime for C/COBOL
- AIX(64)，Linux(x64)の場合
  - 日立コード変換 - Runtime(64)
  - Hitachi Code Converter - Runtime for C/COBOL(64)

また，COBOL プログラムから直接コード変換ライブラリを呼び出す場合は，上記に加えて次のプログラムプロダクトのどれかが必要です。

- AIX(32)，Linux(x86)の場合
  - 日立コード変換 - Development Kit
  - Hitachi Code Converter - Development Kit for C/COBOL
  - Hitachi Code Converter - Development Kit for C/COBOL(64)
- AIX(64)，Linux(x64)の場合
  - 日立コード変換 - Development Kit(64)
  - Hitachi Code Converter - Development Kit for C/COBOL(64)

27.3.2　Unicode 機能を使用したプログラムのコンパイルおよび実行

(1)　AIX の場合

Unicode 機能を使用したプログラムのコンパイルおよび実行は，シフト JIS 環境下で行ってください。シフト JIS 環境にする方法は「付録 A.2　シフト JIS の場合」を参照してください。

(2)　Linux の場合

Unicode 機能を使用したプログラムのコンパイルおよび実行は，UTF-8 環境下で行ってください。UTF-8 環境にする方法は「付録 A.4　Unicode の場合」を参照してください。

27.4　Unicode 機能の詳細

27.4.1　コンパイル

-UniObjGen オプションおよび-UniEndian オプションを指定すると英数字定数，および日本語文字定数の文字コードを Unicode に変換します。-UniObjGen オプションについては，「32.5.14　その他の設定」の「(17)　-UniObjGen オプション」を参照してください。-UniEndian オプションについては，「32.5.14　その他の設定」の「(18)　-UniEndian オプション」を参照してください。

(1)　-UniObjGen オプションと-UniEndian オプションの関係

-UniObjGen オプションと-UniEndian オプションの関係を，次に示します。

表 27–2　-UniObjGen オプションと-UniEndian オプションの関係

-UniObjGen オプション指定		-UniEndian オプション指定			
		-UniEndian,Little	-UniEndian,Big	指定なし（デフォルト）	
				AIX	Linux
指定あり	日本語文字定数	UTF-16LE に変換	UTF-16BE に変換	UTF-16BE に変換	UTF-16LE に変換
	英数字定数	UTF-8 に変換（-UniEndian オプションの影響はない）			
指定なし※		無変換			該当しない

注※

Linux の場合，-UniObjGen オプションを指定しなければなりません。指定がない場合はコンパイルエラーとなります。

(2)　-UniObjGen オプション指定時の注意事項

-UniObjGen オプション指定時の注意事項を次に示します。

- -UniObjGen オプション指定の有無が異なる COBOL プログラムを混在して実行することはできません。混在して実行した場合，動作は保証しません。
- 前提条件を満たしていない環境で-UniObjGen オプションを指定してコンパイルした場合，コンパイルエラーとなります。前提条件を満たしていない環境を次に示します。
  - コード変換ライブラリがインストールされていない場合
  - 使用するコード変換ライブラリが前提バージョンに満たない場合
- NATIVE 指定以外の符号系名を，実行用計算機段落の PROGRAM COLLATING SEQUENCE 句，または SORT，MERGE 文の COLLATING SEQUENCE 句に指定した場合，コンパイルエラーとなります。

- -Switch,EBCDIK オプションまたは-Switch,EBCDIC オプションと-UniObjGen オプションを同時に指定した場合，コンパイルエラーとなります。
- コンパイル中にコード変換に失敗した場合，コンパイルエラーとなります。
- AIX の場合，EUC 環境下で-UniObjGen オプションを指定すると，コンパイルエラーとなります。
- Linux の場合，-UniObjGen オプション指定時に環境変数 CBLSRCENCODING を指定していないと，コンパイルエラーとなります。
- Linux の場合，UTF-8 環境下で-UniObjGen オプションを指定していないと，コンパイルエラーとなります。
- -JPN オプション，-CompatiV3 オプション，-V3Rec オプションは指定できません。ただし，コンパイラ環境変数 CBLV3UNICODE に YES を指定すると，-UniObjGen オプションと-CompatiV3 または-V3Rec オプションを同時に指定できます。コンパイラ環境変数 CBLV3UNICODE については，「32.6.3　コンパイラ環境変数の詳細」の「(20)　CBLV3UNICODE」を参照してください。

(3)　-UniEndian オプション指定時の注意事項

-UniEndian オプション指定時の注意事項を次に示します。

- -UniEndian オプション指定（Little／Big）の異なる COBOL プログラムを混在して使用することはできません。混在して使用した場合，動作は保証しません。

27.4.2　実行

-UniObjGen オプションを指定してコンパイルしたプログラムを正しく実行するには，実行時環境変数 CBLLANG に UNICODE を指定してください。実行時環境変数 CBLLANG については，「35.3.3　一般」を参照してください。

ここでは，実行時での規則について説明します。

(1)　実行時環境変数 CBLLANG と CBLUNIENDIAN の関係

実行時環境変数 CBLLANG の設定形式を次に示します。

形式

```
CBLLANG=UNICODE
```

実行時環境変数 CBLUNIENDIAN の設定形式を次に示します。

形式

```
CBLUNIENDIAN= {LITTLE | BIG}
```

実行時環境変数 CBLLANG と CBLUNIENDIAN の関係を，次に示します。

表 27–3　実行時環境変数 CBLLANG と CBLUNIENDIAN の関係

環境変数 CBLLANG 指定			環境変数 CBLUNIENDIAN 指定			
			LITTLE	BIG	指定なし（デフォルト）	
					AIX	Linux
指定あり	UNICODE 指定あり	用途が NATIONAL の項目	UTF-16LE を仮定	UTF-16BE を仮定	UTF-16BE を仮定	UTF-16LE を仮定
		用途が DISPLAY の項目	UTF-8 を仮定（環境変数 CBLUNIENDIAN の影響はない）			
指定なし※			無変換			該当しない

注※

Linux の場合，環境変数 CBLLANG に UNICODE を指定しなければなりません。指定がないときは実行時エラーとなります。

(2)　-UniObjGen オプションと実行時環境変数 CBLLANG の関係（AIX の場合）

-UniObjGen オプションと実行時環境変数 CBLLANG の関係を，次に示します。

表 27–4　-UniObjGen オプションと実行時環境変数 CBLLANG の関係

-UniObjGen オプション指定	環境変数 CBLLANG 指定	
	UNICODE	指定なし（デフォルト）
指定あり	○※	×
指定なし	×	○

（凡例）

○：動作を保証する

×：動作を保証しない

注※

Unicode 機能で動作します。

(3)　-UniEndian オプションと実行時環境変数 CBLUNIENDIAN の関係

-UniEndian オプションの指定（Little／Big）と，実行時環境変数 CBLUNIENDIAN の指定（LITTLE／BIG）の組み合わせが異なる場合，動作は保証しません。-UniEndian オプションと実行時環境変数 CBLUNIENDIAN の関係を次に示します。

表 27–5　-UniEndian オプションと実行時環境変数 CBLUNIENDIAN の関係

-UniEndian オプションの指定		環境変数 CBLUNIENDIAN の指定			
		LITTLE	BIG	指定なし（デフォルト）	
				AIX	Linux
-UniEndian,Little		○	×	×	○
-UniEndian,Big		×	○	○	×
指定なし	AIX	×	○	○	−
	Linux	○	×	−	○

（凡例）

○：動作を保証する

×：動作を保証しない

−：該当しない

(4)　コード変換失敗時の動作

コード変換ライブラリの前提条件を満たしていない環境で，実行時環境変数 CBLLANG に UNICODE を指定して実行した場合，コード変換失敗となります。コード変換失敗時の動作を次に示します。

表 27–6　コード変換失敗時の動作

機能	コード変換失敗時の動作
CSV 編成ファイルの入出力	実行時エラーとなる。
CBLNCNV サービスルーチン	戻り値として-2 または-3 を返す。詳細については，「29.7.1 CBLNCNV」を参照のこと。
DISPLAY-OF 関数／NATIONAL-OF 関数	実行時エラーとなる。

(5)　EUC 環境で COBOL プログラムを実行した場合（AIX で有効）

EUC 環境で，実行時環境変数 CBLLANG に UNICODE を指定して COBOL プログラムを実行した場合，実行時エラーとなり，プログラムの実行を中止します。

(6)　UTF-8 環境で COBOL プログラムを実行した場合（Linux で有効）

UTF-8 環境で，実行時環境変数 CBLLANG に UNICODE を指定しないで COBOL プログラムを実行した場合，実行時エラーとなり，プログラムの実行を中止します。

27.5 Unicode に対応する機能

Unicode に対応する機能を，次に示します。

表 27–7 Unicode に対応する機能（規格）

機能名	Unicode 対応状況						備考
	AIX			Linux			
	用途 DISPLAY		用途 NATIONAL	用途 DISPLAY		用途 NATIONAL	
	ASCII 範囲内※1	ASCII 範囲外※1		ASCII 範囲内※1	ASCII 範囲外※1		
基本機能	○	△	△	○	△	△	定数に多バイト文字を指定できない句または文がある。詳細については，「27.5.1 基本機能」，または「27.7 Unicode 機能での制限事項」を参照のこと。
順編成ファイル	○	○※2	○※2	○	○※2	○※2	
相対編成ファイル	○	○※2	○※2	○	○※2	○※2	
ISAM による索引編成ファイル	○	○※2	○※2	○	○※2	○※2	キーとして用途が NATIONAL の項目を指定した場合，意図しない行に位置づけられることがある。
整列併合	○	○※2	○※2	○	○※2	○※2	キーとして用途が NATIONAL の項目を指定した場合，整列併合の結果が意図しない結果となることがある。
プログラム間連絡	○	○※3※4	○※3※4	○	○※3※4	○※3※4	
組み込み関数	○	○	○	○	○	○	
オブジェクト指向	○	○※3※4	○※3※4	○	○※3※4	○※3※4	
共通例外処理	○	○	○	○	○	○	次に示す組み込み関数の戻り値はシフト JIS とする。 • EXCEPTION-FILE 関数 • EXCEPTION-LOCATION 関数 • EXCEPTION-STATUS 関数

機能名	Unicode 対応状況						備考
	AIX			Linux			
	用途 DISPLAY		用途 NATIONAL	用途 DISPLAY		用途 NATIONAL	
	ASCII 範囲内※1	ASCII 範囲外※1		ASCII 範囲内※1	ASCII 範囲外※1		
							• EXCEPTION-STATEMENT 関数
再帰呼び出し	○	○	○	○	○	○	
利用者定義関数	○	○	○	○	○	○	
局所場所節（LOCAL-STORAGE SECTION）	○	○	○	○	○	○	
原始文操作	○	○	○	○	○	○	
自由形式正書法	○	○	○	○	○	○	
TYPEDEF 句と SAME AS 句	○	○	○	○	○	○	
条件翻訳	○	○	○	○	×	×	Unicode 機能の影響は受けない。 翻訳指令行の定数および-Define オプションで受け取る値は環境変数 LANG に依存する。
区分化	○	○	○	○	○	○	

（凡例）

○：制限なく使用できる

△：制限付きで使用できる

×：未対応

注※1

USAGE DISPLAY 項目の ASCII 範囲内は 1 バイト文字，ASCII 範囲外は 2 バイト以上の文字を指します。

注※2

ファイル名の定数指定で多バイト文字を指定した場合，コンパイルエラーとなります。

注※3

プログラム名などの外部シンボルとなる名称に多バイト文字を含む場合，コンパイルエラーとなります。詳細は，「27.7.2　コンパイル時の制限事項」を参照してください。

注※4

CALL 文の定数（プログラム名）または INVOKE 文の定数（メソッド名）に多バイト文字を指定した場合，コンパイルエラーとなります。

表 27-8　Unicode に対応する機能（X/Open）

機能名		Unicode 対応状況						備考
		AIX			Linux			
		用途 DISPLAY		用途 NATIONAL	用途 DISPLAY		用途 NATIONAL	
		ASCII 範囲内※1	ASCII 範囲外※1		ASCII 範囲内※1	ASCII 範囲外※1		
テキスト編成ファイル		○	△※2	△※2	○	△※2	△※2	
ファイル共用（ファイルシェア）		○	○	○	○	○	○	
コマンド行および環境変数へのアクセス	コマンド行	○	○	○	○	○	×	環境変数 LANG に依存して取得する。
	環境変数の取得	○	○	○	○	○	×	
	環境変数の設定	○	×	×	○	×	×	多バイト文字を指定した場合，動作は保証しない。
画面節（SCREEN SECTION）による画面操作		○	×	×	−	−	−	定数に多バイト文字を指定できない句または文がある。詳細については，「27.5.2　入出力機能」の「(6)　画面入出力機能（AIX で有効）」，または「27.7　Unicode 機能での制限事項」を参照のこと。
C 言語インタフェース		○	○※3	○※3	○	○※3	○※3	
インタナショナリゼーション		○	○	○	○	○	○	

（凡例）

○：制限なく使用できる

△：制限付きで使用できる

×：未対応

−：該当しない

注※1

USAGE DISPLAY 項目の ASCII 範囲内は 1 バイト文字，ASCII 範囲外は 2 バイト以上の文字を指します。

注※2

ファイル名の定数指定で多バイト文字を指定した場合，コンパイルエラーとなります。

注※3

プログラム名などの外部シンボルとなる名称に多バイト文字を含む場合，コンパイルエラーとなります。詳細は，「27.7.2　コンパイル時の制限事項」を参照してください。

表 27–9　Unicode に対応する機能（拡張機能）

機能名		Unicode 対応状況						備考
		AIX			Linux			
		用途 DISPLAY		用途 NATIONAL	用途 DISPLAY		用途 NATIONAL	
		ASCII 範囲内※1	ASCII 範囲外※1		ASCII 範囲内※1	ASCII 範囲外※1		
日本語		○	○	○	○	○	○	
ブール演算		○	○	○	○	○	○	
アドレス操作		○	○	○	○	○	○	
1 バイト 2 進および COMP-X 項目		○	○	○	○	○	○	
浮動小数点項目		○	○	○	○	○	○	
報告書作成機能		○	×	×	○	×	×	
ISAM による索引編成ファイル機能の拡張（合成キー，逆順読み）		○	○	○	○	○	○	キーとして用途が NATIONAL の項目を指定した場合，意図しない行に位置づけられることがある。
HiRDB による索引編成ファイル		○	○※2	○※2	−	−	−	キーとして用途が NATIONAL の項目を指定した場合，意図しない行に位置づけられることがある。
CSV 編成ファイル		○	△※2	△※2	○	△※2	△※2	
ラージファイル入出力		○	○	○	○	○	○	
ファイル入出力拡張機能		○	○	○	○	○	○	
画面節（WINDOW SECTION）による画面操作		○	×	×	−	−	−	定数に多バイト文字を指定できない句または文がある。詳細については，「27.5.2　入出力機能」の「(6)　画面入出力機能（AIX で有効）」，または「27.7　Unicode 機能での制限事項」を参照のこと。
通信節による画面操作		○※3	×	×	−	−	−	
COPY 文の接頭辞／接尾辞		○	○	○	○	○	○	
プリンタへのアクセス	XMAP3 による印刷	○※3	×	×	−	−	−	

機能名		Unicode 対応状況						備考
		AIX			Linux			
		用途 DISPLAY		用途 NATIONAL	用途 DISPLAY		用途 NATIONAL	
		ASCII 範囲内※1	ASCII 範囲外※1		ASCII 範囲内※1	ASCII 範囲外※1		
ファイルのディスク書き込み保証		○	○	○	○	○	○	
データコミュニケーション機能		○	○	○	○	○	○	
データベース操作機能（ODBC インタフェース）※6		−	−	−	△	△※7	×	埋め込み SQL 文中の埋め込み変数名以外（表名，列名，カーソル名，定数，プロシージャ名など）に多バイト文字は書けない。
XDM によるデータベースシミュレーション機能	構造型データベース（XDM/SD）	○	○	○	○	○	○	
	リレーショナルデータベース（XDM/RD）	○	○	○	○	○	○	
マルチスレッド環境での実行		○	○	○	○	○	○	
EUC コード		×	×	×	×	×	×	
サービスルーチン	基本機能サービスルーチン	○	△	○	○	△	○	引数を画面に出力するサービスルーチンには対応しない。
	CGI プログラム作成支援機能	×	×	×	−	−	−	
	COBOL 入出力ルーチン	○	○	○	○	○	○	
	バイトストリーム入出力	○	○	×	○	○	×	ファイル名に多バイト文字を指定した場合，動作は保証しない。
数字項目のけた拡張機能		○※5	○※5	○※5	○※4	○※4	○※4	

機能名	Unicode 対応状況						備考
	AIX			Linux			
	用途 DISPLAY		用途 NATIONAL	用途 DISPLAY		用途 NATIONAL	
	ASCII 範囲内※1	ASCII 範囲外※1		ASCII 範囲内※1	ASCII 範囲外※1		
動的長基本項目機能	○	○	○	○	○	○	
定数長拡張機能	○	○	○	○	○	○	

（凡例）

○：制限なく使用できる

△：制限付きで使用できる

×：未対応

－：該当しない

注※1

USAGE DISPLAY 項目の ASCII 範囲内は 1 バイト文字，ASCII 範囲外は 2 バイト以上の文字を指します。

注※2

ファイル名の定数指定で多バイト文字を指定した場合，コンパイルエラーとなります。

注※3

AIX(64)は未対応です。

注※4

Linux(x86)は未対応です。

注※5

AIX(32)は未対応です。

注※6

Unicode に対応する ODBC の SQL データ型は使用できません。

注※7

埋め込み変数で受け渡しするデータとして多バイト文字が使用できるかどうかは，使用する DBMS 環境に依存します。

表 27-10　Unicode に対応する機能（連携機能）

機能名	Unicode 対応状況						備考
	AIX			Linux			
	用途 DISPLAY		用途 NATIONAL	用途 DISPLAY		用途 NATIONAL	
	ASCII 範囲内※1	ASCII 範囲外※1		ASCII 範囲内※1	ASCII 範囲外※1		
XML 連携機能	○	○	○	○	○	○	
Cosminexus 連携機能	○※2	○※2	○※2	○※3	○※3	○※3	

（凡例）

○：制限なく使用できる

注※1

USAGE DISPLAY 項目の ASCII 範囲内は 1 バイト文字，ASCII 範囲外は 2 バイト以上の文字を指します。

注※2

AIX(32)は未対応です。

注※3

Linux(x86)は未対応です。

表 27-11　Unicode に対応する機能（テストデバッガ）

機能名		Unicode 対応状況						備考
		AIX			Linux			
		用途 DISPLAY		用途 NATIONAL	用途 DISPLAY		用途 NATIONAL	
		ASCII 範囲内※	ASCII 範囲外※		ASCII 範囲内※	ASCII 範囲外※		
実行時デバッグ機能		○	△	△	○	△	△	異常終了時要約リスト，データ領域ダンプリストはシフト JIS で出力されるが，リスト中に含まれる Unicode 文字は変換されない。
テストデバッグ機能	バッチモード	○	○	○	○	○	○	
	ラインモード	○	○	○	○	○	○	
カバレージ機能	バッチモード	○	○	○	○	○	○	

（凡例）

○：制限なく使用できる

△：制限付きで使用できる

注※

USAGE DISPLAY 項目の ASCII 範囲内は 1 バイト文字，ASCII 範囲外は 2 バイト以上の文字を指します。

27.5.1　基本機能

ここでは，転記，文字列操作文などの基本機能での規則について説明します。

(1)　Unicode と COBOL の文字数の関係

(a)　Unicode と COBOL の文字数

シフト JIS では，1 文字のバイト数は半角 1 バイト，全角 2 バイトと決まっていますが，Unicode では，1 文字のバイト数は文字によって異なります。

例えば，UTF-8 では半角英数字は 1 文字 1 バイトですが，半角かたかなは 1 文字 3 バイト，全角日本語は 3～8 バイトの可変長になります。また，UTF-16 でも，全角日本語は 2 バイト，4 バイト（サロゲートペア文字），6～8 バイト（IVS 文字）の可変長となります。

一方，COBOL の文字項目は，1 文字のバイト数は文字に関係なく固定となります。

例えば，英数字項目（PIC X）では，1 文字は 1 バイト固定，日本語項目（PIC N）では，1 文字は 2 バイト固定となります。このため，COBOL の文字項目に Unicode データを格納する場合，見た目の 1 文字と COBOL が扱う 1 文字は異なります。

見た目の 1 文字（Character）と区別するため，PICTURE 句に指定する長さをけた（COBOL2002 規格では「文字位置」（Character Position））と呼びます。

サロゲートペアや IVS を含めて，字類が英字，英数字，および日本語の項目に格納した各種文字のバイト数，けた数，文字数，および見た目幅の値を次の表に示します。なお，文字の長さの具体例については，「(2)　文字の長さ」を参照してください。

表 27–12　字類が英字，英数字の項目の場合（UTF-8 エンコーディング）

説明		バイト数	けた数※1	文字数	見た目幅
半角※2	ASCII 文字	1	1	1	1
	半角かたかな	3	3	1	1
	その他の半角文字※3	3	3	1	1
全角（日本語）	Unicode の基本多言語面の文字	2～3	2～3	1	2
	Unicode の追加漢字面の文字（UTF-16 のサロゲートペアで表される文字）	4	4	1	2
	IVS で表される文字	7～8	7～8	1	2

注※1

PICTURE 句の文字列に指定した値です。

注※2

次の範囲を Unicode の半角文字とみなします。

U+0000～U+007F，U+FF61～U+FFDC，U+FFE8～U+FFEE

注※3

その他の半角文字には，半角の矢印（ ← ， → ）などがあります。コード範囲は U+FFE8～U+FFEE です。

表 27–13　字類が日本語の項目の場合（UTF-16 エンコーディング）

説明		バイト数	けた数※1	文字数	見た目幅
半角※2	ASCII 文字	2	1	1	1
	半角かたかな	2	1	1	1
	その他の半角文字※3	2	1	1	1
全角（日本語）	Unicode の基本多言語面の文字	2	1	1	2
	Unicode の追加漢字面の文字（UTF-16 のサロゲートペアで表される文字）	4	2	1	2
	IVS で表される文字	6 または 8	3 または 4	1	2

注※1

PICTURE 句の文字列に指定した値です。

注※2

次の範囲を Unicode の半角文字とみなします。

U+0000～U+007F，U+FF61～U+FFDC，U+FFE8～U+FFEE

注※3

その他の半角文字には，半角の矢印（ ← ， → ）などがあります。コード範囲は U+FFE8～U+FFEE です。

(b)　結合文字

結合文字（例：「か」＋「゛」⇒「が」）が格納されている場合は，2 文字として扱います。

(c)　サロゲートペア文字

Unicode 機能でのサロゲートペア文字は，次の点を考慮してプログラミングすることで，ユーザデータとして使用できます。なお，IVS の異体字セレクタはサロゲートペア文字で表すため，IVS も同様に注意が必要です。

COBOL プログラムでサロゲートペア文字を含むデータ使用時の注意事項

- データ定義での注意事項

  サロゲートペア文字は，4 バイトで 1 文字を表します。Unicode（UTF-16）で 1 文字が 2 バイトで表される範囲の文字だけ使う場合，COBOL2002 の Unicode 機能では，PIC N の日本語項目 1 けたで 1 文字を扱うことができます。したがって，サロゲートペア文字を含むデータが格納されるデータ項目は，日本語項目 2 けたでサロゲートペア文字の 1 文字を扱うことになる点に注意してください。

- 手続き部での注意事項

  サロゲートペア文字が格納されるデータ領域が，サロゲートペア文字 1 文字に対して PIC N(2)の日本語項目 2 けたで定義されていれば，転記や比較などで問題はありません。

  ただし，文字列中の文字位置を固定で参照・更新されるような処理や，部分参照での文字位置や長さで文字数を意識しているような処理では，サロゲートペア文字 1 文字に対して日本語項目 2 けたであることを考慮した処理にする必要があります。

- テストデバッガでの注意事項

  サロゲートペア文字が格納されたデータ領域を画面に表示する場合，16 進で表示させてください。データ属性で表示させた場合，「〓」で表示されます。

- 上記以外の注意事項

  次に示す機能でサロゲートペア文字を含むデータ値を使用した場合，動作は保証しません。

  ・データコミュニケーション機能

  ・XML 連携機能

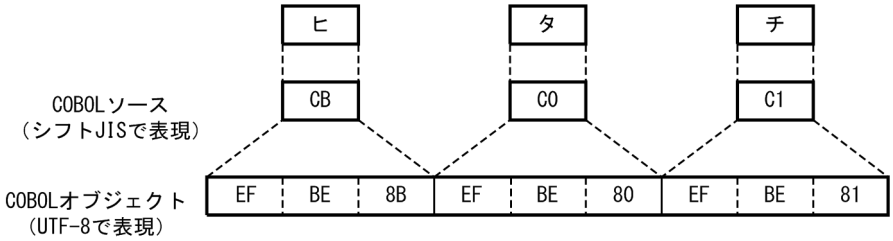
(2) 文字の長さ

文字の長さの例を次に示します。

(例 1)

UTF-8 の場合，半角かたかなは 3 バイトで表現するため，TEST-DATA1 は英数字で 9 けた必要となります。

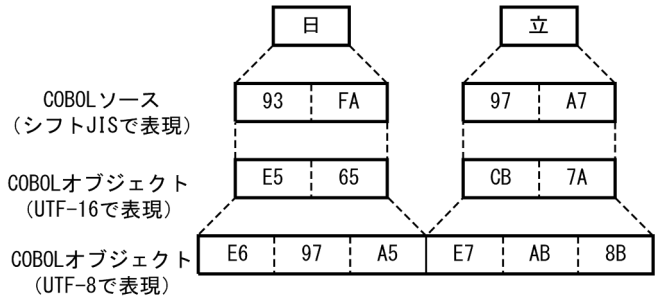
```
01 TEST-DATA1 PIC X(9) USAGE DISPLAY VALUE 'ﾋﾀﾁ'.
```



(例 2)

UTF-16 の場合，Unicode の基本多言語面の全角文字は 2 バイトで表現するため，TEST-DATA2 は 2 けた必要となります。UTF-8 の場合，Unicode の基本多言語面の全角文字は 3 バイトで表現するため，TEST-DATA3 は 6 けた必要となります。

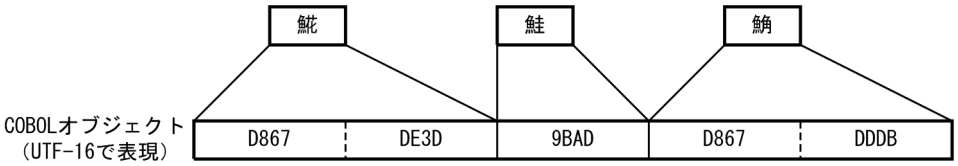
```
01 TEST-DATA2 PIC N(2) USAGE NATIONAL VALUE NC'日立'.
01 TEST-DATA3 PIC X(6) USAGE DISPLAY  VALUE   '日立'.
```



(例 3)

UTF-16 でサロゲートペア文字を扱う場合，次の例では 3 文字を表すのに日本語項目で 5 けた（10 バイト）必要となります。

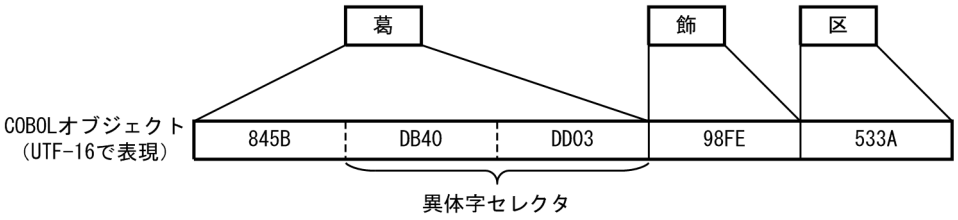
```
01 TEST-DATA2 PIC N(5) USAGE NATIONAL
   VALUE NX'D867DE3D9BADD867DDDB'.  *> '鰯鮭鯖'
```



(例 4)

UTF-16 で IVS 文字を扱う場合，基となる文字（基底文字）に異体字セレクタと呼ばれる 4 バイトが付加されるため，次の例では 3 文字を表すのに日本語項目で 5 けた（10 バイト）必要となります。

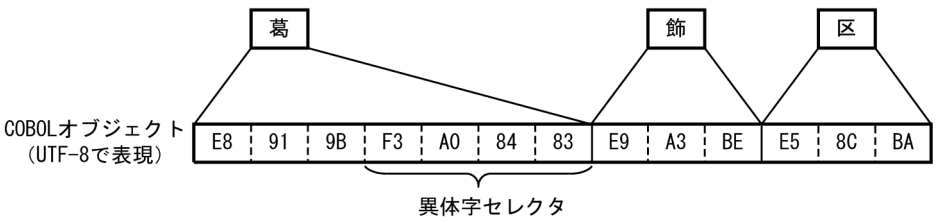
```
01 TEST-DATA3 PIC N(5) USAGE NATIONAL
   VALUE NX'845BDB40DD0398FE533A'.  *> '葛飾区'
```



(例 5)

UTF-8 で IVS 文字を扱う場合，基となる文字（基底文字）に異体字セレクタと呼ばれる 4 バイトが付加されるため，次の例では 3 文字を表すのに英数字項目で 13 けた（13 バイト）必要となります。

```
01 TEST-DATA4 PIC X(13) USAGE DISPLAY
   VALUE X'E8919BF3A08483E9A3BEE58CBA'.  *> '葛飾区'
```



(3) 空白文字，表意定数 SPACE，および転記の空白詰めの文字コード

空白文字，表意定数 SPACE，および転記の空白詰めの文字コードについて，次に示します。

- 用途が DISPLAY の場合，UTF-8 の半角空白（X'20'）を設定します。
- 用途が NATIONAL の場合，バイトオーダによって次の文字コードを設定します。
  - バイトオーダがリトルエンディアンの場合，全角空白（X'0030'）とします。
  - バイトオーダがビッグエンディアンの場合，全角空白（X'3000'）とします。

(4) 表意定数 ZERO の文字コード

表意定数 ZERO の文字コードについて，次に示します。

- 用途が DISPLAY の場合，UTF-8 の半角ゼロ（X'30'）を設定します。
- 用途が NATIONAL の場合，バイトオーダによって次の文字コードを設定します。
  - バイトオーダが UTF-16LE の場合，全角ゼロ（X'10FF'）とします。
  - バイトオーダが UTF-16BE の場合，全角ゼロ（X'FF10'）とします。

(5) 文字コードを変換するプログラム例

Unicode 機能を使用するプログラム実行時には，用途が DISPLAY／NATIONAL の項目に格納される文字データはすべて UTF-8／UTF-16 として処理します。たとえば，ファイルから読み込んだレコード項目にシフト JIS の文字データが格納されることで，用途が DISPLAY／NATIONAL の項目に UTF-8／UTF-16 以外の文字データが格納される場合は，コード変換ライブラリを使用して入力データを UTF-8／UTF-16 へ変換する必要があります。

コード変換ライブラリを使用してシフト JIS の文字データを UTF-8 へ変換するコーディング例を次に示します。

コンパイル例

AIX(32)の場合

```
ccbl2002 -UniObjGen sample1.cbl  -L /opt/hcodecnv/lib -lhcodecnv -brtl
```

AIX(64)の場合

```
ccbl2002 -UniObjGen sample1.cbl  -L /opt/hcodecnv64/lib -lhcodecnv64 -brtl
```

Linux(x86)の場合

```
ccbl2002 -UniObjGen sample1.cbl  -L /opt/hcodecnv/lib -lhcodecnv
```

Linux(x64)の場合

```
ccbl2002 -UniObjGen sample1.cbl  -L /opt/hcodecnv64/lib -lhcodecnv64
```

プログラム例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
      *>コード変換ライブラリ提供の登録集原文の取り込み
       COPY 'codeconv.cbl'.
       01 CONV-TABLE-P   ADDRESS.
       01 DEFAULT-CODE1  PIC X(1).
       01 DEFAULT-CODE2  PIC X(1).
       01 TABLE-FULLPATH ADDRESS VALUE NULL.
       01 MAPPING-FULLPATH ADDRESS VALUE NULL.
       01 RETCODE        PIC 9(9) USAGE COMP.
       01 SYSTEM-ERRCODE PIC 9(9) USAGE COMP.
       01 INLEN          PIC 9(9) USAGE COMP.
       01 INBUF          PIC X(100).
       01 OUTLEN         PIC 9(9) USAGE COMP.
       01 OUTBUF         PIC X(300).
       PROCEDURE DIVISION.
           CALL 'CodeConvOpen' USING
               BY VALUE     CCF-SJIS-UTF
                            0
               BY REFERENCE DEFAULT-CODE1
                            DEFAULT-CODE2
               BY VALUE     TABLE-FULLPATH
                            MAPPING-FULLPATH
               BY REFERENCE RETCODE
                            SYSTEM-ERRCODE
               RETURNING CONV-TABLE-P.
           IF CONV-TABLE-P NOT = NULL THEN
               COMPUTE CCT-CODECONVTABLEA = CONV-TABLE-P
                       :
                (INBUFにシフトJISデータを格納)
                       :
               MOVE 100 TO INLEN
               MOVE 300 TO OUTLEN
               CALL 'CodeConvString' USING
                   BY REFERENCE CCT-CODECONVTABLE
                                INLEN
                                INBUF
                                OUTLEN
                                OUTBUF
           ELSE      *>エラー時の処理
                     :
           END-IF.
           CALL 'CodeConvClose' USING
               BY REFERENCE CCT-CODECONVTABLE
                            SYSTEM-ERRCODE.
           DISPLAY 'INBUF=' INBUF.
           DISPLAY 'OUTBUF=' OUTBUF.
           EXIT PROGRAM.
```

(6) 用途 DISPLAY と用途 NATIONAL との間の変換

用途が DISPLAY の項目と，用途が NATIONAL の項目との間で変換（転記）する場合，コード変換が必要です。コード変換する方法を次に示します。

- DISPLAY-OF 関数，NATIONAL-OF 関数を使用する
- コード変換ライブラリを使用して UTF-8 と UTF-16 を相互に変換する

(7) 入出力データの変換

CSV 編成ファイルを除き，入出力データは無変換とします。入出力データとは，次の機能使用時に入出力されるデータのことです。

- ファイル入出力
- 少量入出力（DISPLAY 文，ACCEPT 文）
- 画面機能（AIX で有効）
- データベース操作機能（Linux で有効）
- サービスルーチン

(8) シフト JIS 範囲外の文字

シフト JIS 範囲外の文字については，16 進英数字定数または 16 進日本語文字定数を使用し，UTF-8 または UTF-16 のコード値を直接指定してください。このとき，16 進日本語文字定数は，常に UTF-16BE で指定してください。COBOL2002 コンパイラは，-UniEndian オプションの指定に従い，16 進日本語文字定数の文字コードを変換します。

(9) データの比較

データの比較は，文字コードによるバイナリ比較となります。そのため，用途が NATIONAL の項目または日本語文字定数の大小比較はバイトオーダによって結果が異なります。

27.5.2 入出力機能

Unicode 機能では，順編成ファイル，相対編成ファイル，および索引編成ファイルについては，文字コードを意識することなく，入出力できます。しかし，テキスト編成ファイルおよび CSV 編成ファイルは，ほかのアプリケーションで参照可能とするため，文字コードを UTF-8 に統一します。

(1) テキスト編成ファイル

テキスト編成ファイルでは，UTF-8 で作成されたファイルの入出力ができます。この機能の規則を次に示します。

- 使用するテキスト編成ファイルは UTF-8 で作成してください。また，テキスト編成ファイルに関連づけられたレコード記述項は，用途を DISPLAY に統一してください。用途を DISPLAY に統一していない場合は，コンパイルエラーとなります。
- 新規ファイルを作成時，該当するファイルに Unicode シグニチャとして（X'EFBBBF'）の 3 バイトを付加します。
- Unicode シグニチャを含むテキスト編成ファイルを入力する場合，Unicode シグニチャは自動的に読み飛ばします。そのため，ユーザが Unicode シグニチャを意識する必要はありません。
- COBOL プログラム以外で，COBOL で作成するテキスト編成ファイルを処理するとき，Unicode シグニチャを意識しなければなりません。

  ただし，次に示すテキスト編成ファイルの Unicode シグニチャ出力の切り替え機能を使用すると，Unicode シグニチャを出力しないファイルを作成できるため，Unicode シグニチャを意識する必要がなくなります。

(a) テキスト編成ファイルの Unicode シグニチャ出力の切り替え機能

新規ファイル作成時に Unicode シグニチャを出力しないようにする場合，実行時環境変数 CBLD_ファイル名に TEXTSUPPRESSBOM を設定するか，または実行時環境変数 CBLTEXTSUPPRESSBOM を設定します。

なお，この機能はテキスト編成ファイル作成時だけ有効です。

ファイル単位に指定する方法

形式

```
CBLD_ファイル名= {TEXTSUPPRESSBOM | NOTEXTSUPPRESSBOM}
```

機能

- この環境変数に TEXTSUPPRESSBOM が指定された場合，実行単位中の任意の新規作成のテキスト編成ファイルに対して Unicode シグニチャを出力しません。
- この環境変数に NOTEXTSUPPRESSBOM が指定された場合，テキスト編成ファイルの Unicode シグニチャ出力の切り替え機能は無効となります。

実行単位中のすべてのファイルに指定する方法

形式

```
CBLTEXTSUPPRESSBOM=YES
```

機能

- この環境変数に YES が指定された場合，実行単位中のすべての新規作成のテキスト編成ファイルに対して Unicode シグニチャを出力しません。
- この環境変数に YES の指定がないか，または YES 以外が指定された場合，テキスト編成ファイルの Unicode シグニチャ出力の切り替え機能は無効となります。

規則

ファイル単位に指定する方法と実行単位中のすべてのファイルに指定する方法を併用した場合，ファイル単位に指定する方法が優先されます。

環境変数 CBLTEXTSUPPRESSBOM と環境変数 CBLD_ファイル名の指定の組み合わせによる Unicode シグニチャの出力状態を次の表に示します。

表 27–14　環境変数 CBLTEXTSUPPRESSBOM と環境変数 CBLD_ファイル名の指定の組み合わせによる Unicode シグニチャの出力状態

CBLTEXTSUPPRESSBOM	CBLD_ファイル名		
	TEXTSUPPRESSBOM	NOTEXTSUPPRESSBOM	指定なし
YES	出力しない	出力する	出力しない
環境変数指定なし， または YES 以外	出力しない	出力する	出力する

(2)　CSV 編成ファイル

CSV 編成ファイルでは UTF-8 で作成されたファイルの入出力ができます。この機能の規則を次に示します。

(a)　CSV 編成ファイルの入出力

CSV 編成ファイルを入力する場合，対象ファイルは UTF-8 で作成してください。対象ファイルのレコード記述項に用途が NATIONAL の項目が定義されている場合，入出力時に UTF-8／UTF-16 の相互変換をします。そのため，CSV 編成ファイルに UTF-8 以外の文字コードが含まれる場合，動作は保証しません。CSV 編成ファイルの入出力について次に示します。

- 入力したデータに対応する項目の用途が NATIONAL の場合，実行時環境変数 CBLUNIENDIAN の指定に従い，UTF-8 から UTF-16 に自動変換し，データ項目に格納します。
- 出力するレコード領域に用途が NATIONAL の項目を含む場合，該当する項目の文字コードを UTF-16 から UTF-8 に自動変換することで，CSV 編成ファイルの文字コードを UTF-8 に統一します。

CSV 編成ファイルでは，用途が NATIONAL の項目の文字コードを UTF-16 から UTF-8 に変換することで，プログラム上のサイズと実際のレコードサイズが異なります。

(例 1)

```
       IDENTIFICATION DIVISION.
              :
       DATA DIVISION.
       FILE SECTION.
       FD FILE01.
       01 REC01 PIC X(5) USAGE DISPLAY.
       PROCEDURE DIVISION.
              :
```

```
        MOVE ’ABCDE’ TO REC01.
        WRITE REC01.
```

この場合，データ項目「REC01」は用途が DISPLAY なので，文字コードは UTF-8 であり，実際に REC01 に格納されたデータとファイルに書き出されたデータサイズは等しくなります。

（例 2）

```
       IDENTIFICATION DIVISION.
             :
       DATA DIVISION.
       FILE SECTION.
       FD FILE02.
       01 REC02 PIC N(5) USAGE NATIONAL.
       PROCEDURE DIVISION.
             :
           MOVE NC’あいうえお’ TO REC02.
           WRITE REC02.
```

この場合，データ項目「REC02」は用途が NATIONAL であり，文字コードは UTF-16 なので，ファイルへ書き出すときに UTF-8 に自動変換します。UTF-16（2 バイトの固定）から UTF-8（可変。日本語は 3 バイト）に変換することで，データサイズは大きくなります。

（例 3）

```
       IDENTIFICATION DIVISION.
             :
       DATA DIVISION.
       FILE SECTION.
       FD FILE02.
       01 REC02 PIC N(5) USAGE NATIONAL.
       PROCEDURE DIVISION.
             :
           READ  FILE02.
```

例 2 で出力したデータを入力するとき，入力データは UTF-8 で 15 バイトですが，入力時に UTF-16 に変換することで 10 バイトとなり，データサイズは小さくなります。

(b) Unicode シグニチャ

CSV 編成ファイルは，Unicode シグニチャを含まないでください。CSV 編成ファイルに Unicode シグニチャが含まれる場合，データとして扱います。

(3) 順編成ファイル，相対編成ファイル，および索引編成ファイルを使用した入出力

順編成ファイル，相対編成ファイル，および索引編成ファイルを使用した入出力では，用途が NATIONAL の項目について UTF-16 から UTF-8 への自動変換はしません。この機能の規則を次に示します。

- 順編成ファイル，索引編成ファイル，および相対編成ファイルの入力時，ファイル中の用途が NATIONAL の項目に対応する UTF-16 データと，実行時環境変数 CBLUNIENDIAN のバイトオーダを一致させてください。

(4) HiRDB による索引編成ファイル使用時の，HiRDB のクライアント環境変数 PDCLTCNVMODE の指定

HiRDB による索引編成ファイルを使用している場合，HiRDB のクライアント環境変数 PDCLTCNVMODE には，NOUSE（デフォルト値）を指定してください。ほかの値を指定した場合，不当なコード変換によって入出力エラーになることがあります。

(5) ACCEPT／DISPLAY 文による入出力

(a) ACCEPT 文による入力

ACCEPT 文で外部から入力したデータは，文字コードを変換しません。AIX の場合，外部から入力することで，用途が DISPLAY／NATIONAL の項目に UTF-8／UTF-16 以外の文字データが格納されるときは，コード変換ライブラリを使用して入力データを UTF-8／UTF-16 へ変換してください。なお，Linux の場合，ACCEPT 文で外部から入力したデータを用途が NATIONAL の項目に格納できません。格納する場合の動作は保証しません。

(b) DISPLAY 文による出力

DISPLAY 文で用途が DISPLAY／NATIONAL の項目の内容を画面上に出力するとき，文字コードは変換しません。出力する内容が Unicode の場合，文字情報を正しく出力するためにはコード変換ライブラリを使用し文字コードを変換してください。

ただし，Linux の場合，UTF-8 ロケールでの動作となるため，出力する内容が UTF-8 のときは文字コードの変換は必要ありません。

(c) 注意事項

DISPLAY 文でファイルに出力する場合，指定した項目の用途にかかわらず，改行コードは X'0A'となります。

(6) 画面入出力機能（AIX で有効）

日本語，半角かたかななど，Unicode の多バイト文字を COBOL プログラムから直接画面表示はできません。

(7) プリンタへのアクセス（AIX(32)で有効）

日本語，半角かたかななど，Unicode の多バイト文字を COBOL プログラムから直接印刷はできません。

27.5.3　CBLNCNV サービスルーチン

CBLNCNV サービスルーチンを呼び出すと，UTF-8 で記述された半角文字から UTF-16 で記述された全角文字への変換ができます。CBLNCNV サービスルーチンについては，「29.7.1　CBLNCNV」を参照してください。

(1)　CBLNCNV サービスルーチンでの文字変換

CBLNCNV サービスルーチンでの文字変換は，次の順序でします。

1. UTF-8 のコード範囲の半角文字を UTF-8 の全角文字コードに変換します。
2. 変換した UTF-8 の全角文字コードを，実行時環境変数 CBLUNIENDIAN の指定に従い UTF-16LE または UTF-16BE に変換します。

(2)　CBLNCNV サービスルーチンの戻り値

Unicode 機能での CBLNCNV サービスルーチンの戻り値について次に示します。

- コード変換ライブラリが正しくインストールされていない場合，戻り値として-2 を返します。
- コード変換ライブラリがエラーを返し，コード変換に失敗した場合，戻り値として-3 を返します。主なエラーの要因を次に示します。
  - 送り出し側作用対象の項目に，UTF-16 に変換できない文字が含まれている。
  - 送り出し側作用対象の項目が，日本語文字や半角かたかななどの多バイト文字の途中で切れている。

  その他のエラーの要因については，「コード変換ライブラリのマニュアル」の CodeConvString 関数の説明を参照してください。

27.5.4　組み込み関数

(1)　Unicode に対応した組み込み関数

Unicode の文字操作や文字数／けた数の取得，コード変換をする組み込み関数を次の表に示します。これらの組み込み関数の形式や使用例などの詳細は，マニュアル「COBOL2002　言語　拡張仕様編」「23.2 組み込み関数」を参照してください。

表 27–15　Unicode に対応した組み込み関数

項番	用途	組み込み関数名	概要
1	文字列の取り出し	SUBSTRING	字類が英字，英数字，または日本語のデータ項目から部分文字列を返します。

項番	用途	組み込み関数名	概要
2	文字数の取得	COUNT-CHAR	字類が英字，英数字，または日本語のデータ項目の文字数を返します。
3	けた数の取得	LENGTH-OF-SUBSTRING	字類が英字，英数字，または日本語のデータ項目の部分文字列のけた数を返します。
4	コード変換	DISPLAY-OF	字類が日本語のデータ項目中の UTF-16 の文字列を UTF-8 の文字列に変換して返します。
5		NATIONAL-OF	字類が英数字のデータ項目中の UTF-8 の文字列を UTF-16 の文字列に変換して返します。

なお，Unicode に対応した組み込み関数を使用した場合，Unicode 文字の特性上，文字の判定や取得する処理の実行時間が増加する場合や，動作を保証できない場合があります。詳細は，「(2)　Unicode に対応した組み込み関数の注意事項」を参照してください。

(2)　Unicode に対応した組み込み関数の注意事項

- Unicode 文字は特性上，シフト JIS や日本語 EUC と比べて 1 文字のサイズが多様なため，文字の判定や取得する処理の実行時間が増加します。シフト JIS や日本語 EUC の既存のプログラムを Unicode 機能に移行して Unicode に対応した組み込み関数を使用する場合は，実行時間が大きく増える場合があります。そのため，次の点に注意して組み込み関数を使用してください。
  - 1 文字のサイズが可変となる可能性がないデータ項目※の場合は，Unicode に対応した組み込み関数を使用しないで，部分参照を使用して処理する。

    注※

    例えば，半角英数字だけ格納される英数字項目や，サロゲートペアおよび IVS を含む可能性がない日本語項目などが該当します。
  - Unicode に対応した組み込み関数は，文字数の考慮が必要な処理の場合だけ使用する。

    例えば，文字列の取り出しで，けた数と文字数の両方がわかっている場合は，けた数による部分参照を使用して処理します。
- -UniObjGen コンパイラオプションを指定してコンパイルした組み込み関数を使用するプログラムを，実行時環境変数 CBLLANG で UNICODE を指定していない環境で実行すると，実行時エラーが発生してプログラムの実行が中止します。
- Unicode の文字を格納するデータ項目のけた数が，格納しようとする文字の長さより短い場合，データ項目のけた数までのデータが格納されます。

  Unicode に対応した組み込み関数には，格納される Unicode 文字の長さを考慮したけた数を定義したデータ項目を指定する必要があります。
- 英数字集団項目の従属項目に，字類が英字および英数字の項目と，字類が日本語の項目が混在している場合，各項目の文字列を Unicode 文字として扱えないことがあります。そのため，英数字集団項目の従属項目は，字類が英字および英数字の項目だけを使用してください。

- 組み込み関数の引数の指定に誤りがある場合は，次の表に示すように，実行時に EC-ARGUMENT-FUNCTION 例外が発生します。

要因	組み込み関数名				
	COUNT-CHAR	DISPLAY-OF	LENGTH-OF-SUBSTRING	NATIONAL-OF	SUBSTRING
引数 1 の文字コードが不当な値である。※1	○	○※4	○	○※4	○
引数 1 の文字列中にある異体字セレクタが不当な位置にある。※2	○	－	○	－	○
引数 2 と引数 3 が一意名または算術式で，値が正の整数でない。	－	－	○	－	○
引数 2 で指定する開始位置，または引数 2 で指定する開始位置から引数 3 で指定する長さだけ進めた文字位置が引数 1 の終端を超えている。	－	－	○	－	○
引数 2 で指定する開始位置，または引数 2 で指定する開始位置から引数 3 で指定する長さだけ進めた文字位置が全角文字の途中である。※3	－	－	○	－	○

（凡例）

○：EC-ARGUMENT-FUNCTION 例外が成立する。

－：EC-ARGUMENT-FUNCTION 例外が成立しない。

注※1

文字コードが不当な値の場合で，実行時エラーとなる例を次に示します。

（例）

```
01 DATA1 PIC X(30)
VALUE X'EFBDB1EFBDB2EFBDB3EFBDB4EFFDB5EFBDB6EFBDB7EFBDB8EFBDB9EFBDBA'.
*> ｱｲｳｴｵｶｷｸｹｺにしたいが，14バイト目が不正（BDのはずがFDになっている。
*> この位置にFDがあるのはUTF-8として不正な文字と扱う）

MOVE FUNCTION SUBSTRING(DATA1, 2, 4) TO DATA2.
```

<実行結果>

```
KCCC2310R-S
組み込み関数処理中に文字コード不正を検出しました。
プログラム名＝MAIN
行番号／欄＝000022/15
関数名＝SUBSTRING
詳細情報＝引数1の14バイト目で文字コード不正を検出しました。
4バイト目以降:efbdb2efbdb3efbdb4effdb5efbdb6efbdb7efbdb8
```

「n バイト目以降:」の行には，不正を検出した位置とその前後 10 バイトが表示されます。上記の例では，14 バイト目で不正を検出したため，前の 10 バイト（4～13 バイト目），不正バイト（14 バイト目），および後の 10 バイト（15～24 バイト目）の計 21 バイトが表示されています。

注※2

次の場合が該当します。

- 異体字セレクタが文字列の先頭にある
- 異体字セレクタが連続している

(例)

```
FD FILE1.
01 GRP01 GROUP-USAGE NATIONAL.
  02  DATA1 PIC N(1).
  02  DATA2 PIC N(2).

01 DATA3    PIC N(6).

READ FILE1.
*>UTF-16の'葛'(X'845BDB40DD03')をファイルから読み取る。
*>このときDATA1にX'845B', DATA2にX'DB40DD03'が転記される。
MOVE FUNCTION SUBSTRING(DATA2, 1) TO DATA3.
*>DATA2の文字列は先頭から異体字セレクタとなっているため，不正な文字として扱う。
```

注※3

引数に指定した開始位置が，データ項目中の全角文字の途中となった場合の例を次に示します。

(例 1)

```
01 DATA1 PIC X(4) VALUE 'あBC'.

MOVE FUNCTION SUBSTRING(DATA1, 2, 2 WIDTH) TO DATA2.
*>引数2で指定する開始位置は'あ'の途中を指しているため文字を正確に取り出せない。
```

(例 2)

```
01 DATA1 PIC X(4) VALUE 'ABう'.

MOVE FUNCTION SUBSTRING(DATA1, 2, 2 WIDTH) TO DATA2.
*>引数2で指定する開始位置から引数3の長さだけ進めた文字位置は'う'の途中を
*> 指しているため文字を正確に取り出せない。
```

なお，IVS は基底文字＋異体字で構成されます。ただし，基底文字だけでも Unicode 文字として扱われるため，部分文字列として取得してもエラーになりません。

注※4

コンパイル時にコンパイラオプション-FunctionECSup,CodeConvErr の指定がない場合は，例外が成立します。コンパイル時にコンパイラオプション-FunctionECSup,CodeConvErr の指定がある場合は，例外が成立しません。

27.6　入出力情報と取り扱う文字コード

Unicode 機能を使用している場合，入出力時にデータを Unicode（UTF-8 と UTF-16 の混在）として扱わないで，固定の文字コードとして取り扱う情報があります。

入出力情報と取り扱う文字コードの関係を次の表に示します。なお，次の表に記載していない入出力情報は，Unicode として取り扱います。

表 27–16　入出力情報と取り扱う文字コードの関係

分類	入力／出力	入出力情報	取り扱う文字コード	
			AIX	Linux
翻訳時	入力	COBOL ソースプログラム（登録集原文を含む）	シフト JIS	シフト JIS
		外部で設定された環境変数の設定値	シフト JIS	UTF-8※1
		コマンドラインの引数	シフト JIS	UTF-8※1
	出力	コンパイラメッセージ	シフト JIS	UTF-8
		コンパイルリストファイル	シフト JIS	シフト JIS
		TD コマンド格納ファイル	シフト JIS	シフト JIS
実行時	入力	外部で設定された環境変数の設定値	シフト JIS	UTF-8※1
		コマンドラインの引数	シフト JIS	UTF-8※1
		ACCEPT 文で入力したデータ	シフト JIS	UTF-8
		プレロードリストファイル	シフト JIS	UTF-8※4
	出力	実行時メッセージ	シフト JIS	UTF-8※3
		異常終了時要約情報リスト	シフト JIS	UTF-8
		データ領域ダンプリスト	シフト JIS	UTF-8※3
		入出力サービスルーチンデバッグリスト	シフト JIS	UTF-8
		プログラム検索トレース	シフト JIS	UTF-8
リポジトリ管理ツール	出力	エラーメッセージ	シフト JIS	シフト JIS※2
		リポジトリファイル情報リスト	シフト JIS	シフト JIS※2
テストデバッガ	入力	TD コマンド格納ファイル	シフト JIS	シフト JIS
		外部で設定された環境変数の設定値	シフト JIS	UTF-8※1
		コマンドラインの引数	シフト JIS	UTF-8※1

分類	入力／出力	入出力情報	取り扱う文字コード	
			AIX	Linux
	出力	モニタ表示出力ファイル（ログ出力ファイル）	シフト JIS	UTF-8
		結果蓄積ファイル	シフト JIS	UTF-8
		結果出力ファイル	シフト JIS	UTF-8
		エラーメッセージ	シフト JIS	UTF-8
カバレージ	入力	外部で設定された環境変数の設定値	シフト JIS	UTF-8※1
		コマンドラインの引数	シフト JIS	UTF-8※1
	出力	カウント情報リストファイル	シフト JIS	UTF-8
		カバレージ情報リストファイル	シフト JIS	UTF-8
		実行結果出力ファイル	シフト JIS	UTF-8
		エラーメッセージ	シフト JIS	UTF-8

注※1

Unicode で多バイトとなる文字は使用できません。使用した場合，動作は保証しません。

注※2

エラーメッセージおよびリポジトリファイル情報リストは英文（ASCII 範囲）で出力されます。

注※3

Linux の場合，EXTERNAL 句を指定したデータ名およびファイル名に Unicode で多バイトとなる文字は使用できません。使用した場合の動作は保証しません。

注※4

Unicode で多バイトとなる文字は使用できません。使用した場合の動作は保証しません。ただし，コメントに対しては使用できます。

27.7　Unicode 機能での制限事項

ここでは，Unicode 機能での制限事項を説明します。

27.7.1　Unicode に対応していない機能

Unicode に対応していない機能を次に示します。

(1)　Unicode を使用した場合，動作を保証しない機能

- CGI プログラム作成支援機能（AIX(32)で有効）

(2)　コンパイルエラーとなる機能（Linux の場合）

- HiRDB による索引編成ファイル

27.7.2　コンパイル時の制限事項

- 日本語，半角かたかななど，Unicode で多バイト文字となる文字を含む語，英数字定数または日本語文字定数を，次の個所に指定した場合，コンパイルエラーとなります。

表 27–17　UTF-8 の多バイト文字および UTF-16 の文字が指定できない個所

分類	翻訳時にエラーとなる個所
翻訳グループの構造	• END PROGRAM の定数 1 および END METHOD の定数 1
見出し部	• プログラム名段落のプログラム名 1／定数 2 • メソッド名段落のメソッド名 1／定数 2 • 関数名段落の利用者定義関数名 1※3 • クラス名段落のクラス名 1，およびインタフェース名段落のインタフェース名 1※3
環境部	• ALPHABET 句の定数指定の定数 1／2／3 • ASSIGN 句の定数 1 • CLASS 句の定数 5／6 • CURRENCY SIGN 句の定数 7 • リポジトリ段落のクラス指定子，インタフェース指定子，関数指定子，およびプロパティ指定子※3 • 特殊名段落の定数 10※5
データ部	• CODE 句の定数 1
手続き部	• CALL 文の定数 1 • CANCEL 文の定数 1

分類	翻訳時にエラーとなる個所
	• ENTRY 文の定数 • DISPLAY 文の定数 1／2／3※1※2 • INVOKE 文の定数 1 • STOP 文の定数 2※2
組み込み関数	• ADDR 関数の引数 1
画面機能（SCREEN／WINDOW SECTION）※4	• DISPLAY 文の定数 1（SCREEN SECTION） • PICTURE 句 FROM 指定の定数 1（SCREEN SECTION） • PROMPT 句の定数 1（SCREEN SECTION） • VALUE 句の定数 1（SCREEN SECTION） • RESET 句の定数 1（WINDOW SECTION） • VALUE 句の定数 1（WINDOW SECTION）

注※1

画面への出力，環境変数へのアクセス，コマンドラインへのアクセス

注※2

Linux の場合，DISPLAY 文の定数 1 および STOP 文の定数 2 に，Unicode で多バイトとなる文字を含む英数字定数を指定できます。

注※3

Linux で有効です。

注※4

AIX で有効です。

注※5

コンパイラ環境変数 CBLV3UNICODE に YES を指定すると，-UniObjGen オプションと-CompatiV3 オプションを同時に指定できます。そのため，特殊名段落の定数 10 が使用できるようになりますが，Unicode で多バイトとなる文字を指定した場合，コンパイルエラーとなります。

- 16 進英数字定数または 16 進日本語文字定数と，文字定数が混在した連結式を指定した場合，コンパイルエラーとなります。
- Linux の場合，次のファイル名を指定する個所に，Unicode で多バイトとなる文字を指定した場合，コンパイルエラーとなります。
  - COPY 文の定数 1
  - INCLUDE 文の原文名（定数で指定した場合）
- Linux の場合，コマンドライン引数および環境変数の設定値に，Unicode で多バイトとなる文字を使用した場合，動作は保証しません。
- Linux の場合，次に示す項目に Unicode で多バイトとなる文字は使用できません。使用した場合の動作は保証しません。
  - EXTERNAL 句を指定したデータ名およびファイル名
  - SELECT 句に指定したファイル名 1

27.7.3　実行時の制限事項

実行時の制限事項について，次に示します。

- 使用できる文字コードは UCS-4 の範囲（UCS-2 の範囲を含む）です。
- AIX の場合，日本語，半角かたかななど，Unicode で多バイト文字となる文字を，次のサービスルーチンに指定した場合，動作は保証しません。
  - JCPOPUP サービスルーチン
- 次に示す例外に関する組み込み関数の関数値の文字コードはシフト JIS です。関数値を Unicode のデータで使用する場合，コード変換ライブラリを使用して，文字コードを統一してください。
  - EXCEPTION-FILE
  - EXCEPTION-LOCATION
  - EXCEPTION-STATEMENT
  - EXCEPTION-STATUS
- 用途が NATIONAL の項目を従属として含む集団項目と，用途が DISPLAY の項目間の転記処理した場合，転記結果は保証しません。
- 転記や文字列操作機能（部分参照や STRING 文などの文字列操作文）では，用途が DISPLAY の項目に格納された文字は，1 文字 1 バイトとして処理し，用途が NATIONAL の項目に格納された文字は，1 文字 2 バイトとして処理します。文字列操作文が，用途が DISPLAY の項目に対して作用する場合，定数または一意名の値に多バイト文字を含まないでください。
- 日本語，半角かたかななど，Unicode の多バイト文字を含むソースファイル名，およびパス名は使用できません。
- シフト JIS など Unicode 以外のデータを入力し，Unicode に変換しないで処理を続行した場合，動作は保証しません。
- Linux の場合，実行時環境変数 CBLD_ファイル名のファイル名（SELECT 句に指定したファイル名 1）に，Unicode で多バイトとなる文字は指定できません。指定した場合，この環境変数は無視されます。
- Linux の場合，次に示す項目に Unicode で多バイトとなる文字は使用できません。使用した場合の動作は保証しません。
  - EXTERNAL 句を指定したデータ名およびファイル名
  - 外部で設定された環境変数の設定値
  - コマンドラインの引数
- Linux の場合，ACCEPT 文で外部から入力したデータを用途が NATIONAL の項目に格納できません。格納する場合の動作は保証しません。

28 数字項目のけた拡張機能（AIX(64)，Linux(x64)で有効）

数字項目のけた拡張機能は，数字項目，数字編集項目，および数字定数で扱えるけた数の上限を18 けたから 38 けたに拡張し，算術演算，転記，および比較条件で使用できるようにします。この章では，数字項目のけた拡張機能について説明します。

この機能は，AIX(64) COBOL2002，Linux(x64) COBOL2002 で使用できます。なお，エンディアン切換えができるのは，Linux(x64) COBOL2002 だけです。

28.1　数字項目のけた拡張機能の概要

ここでは，数字項目のけた拡張機能の概要について説明します。

28.1.1　概要

数字項目のけた拡張機能とは，数字項目（外部 10 進項目，内部 10 進項目），数字編集項目，および数字定数で扱えるけた数の上限を 18 けたから 38 けたに拡張するものです。これによって，38 けたに拡張した数字項目，数字編集項目，および数字定数を，算術演算，転記，および比較条件で使用できます。なお，算術演算での中間結果を保持する領域は，30 けたの固定小数点形式から 40 けたの 10 進浮動小数点形式となります。

数字項目のけた拡張機能で 38 けたの数字項目を定義したコーディング例を次に示します。

図 28-1　38 けたの数字項目を定義したコーディング例

```
                                        :
DATA DIVISION.
  WORKING-STORAGE SECTION.
   01 DATA1 PIC 9(38) VALUE ZERO.
   01 DATA2 PIC 9(38) VALUE 11111111111111111111111111111111111111.
                    ↑                                   ↑
                    └─────────────┬─────────────────────┘
                                  │
                  数字項目および数字定数の上限として，
                  38けたが指定できるようになります。

   01 DATA3 PIC 9(38) VALUE ZERO.

PROCEDURE DIVISION.
   MOVE 22222222222222222222222222222222222222 TO DATA1.
   COMPUTE DATA3 = DATA1 + DATA2.
   IF DATA3 = 33333333333333333333333333333333333333
      THEN
        DISPLAY '###  OK  ###'
      ELSE
        DISPLAY '***  NG  ***'
   END-IF.
   STOP RUN.
```

38けたの数字項目および数字定数を算術演算，転記，比較条件で使用できます。

数字項目のけた拡張機能を使用する場合の注意事項を次に示します。

- 数字項目や数字編集項目のけた数は，従来の 18 けたより大きなけた数が扱えますが，けた数が増えると実行時間も大きく増えるおそれがあります。
- 数字項目のけた拡張機能を使用した場合，中間結果で保持するけた数が，数字項目のけた拡張機能を使用しない場合より大きい 40 けたとなるため，従来の 18 けた以下の演算であっても演算結果が異なり，算術式および条件式の結果が異なる場合があります。
- 次の場合は，数字項目のけた拡張機能と同時に使用できません。
  - MEDIAN 関数の引数に，算術式または組み込み関数を指定したとき。
  - TURN 指令に EC-SIZE を指定したとき。

28.1.2　数字項目のけた拡張機能で必要なコンパイラオプション

数字項目のけた拡張機能を使用する場合は，コンパイル時に-MaxDigits38 オプションと-IntResult,DecFloat40 オプションを同時に指定してください。-MaxDigits38 オプションと-IntResult,DecFloat40 オプションを同時に指定していない場合は，コンパイルエラーとなります。

-MaxDigits38 オプションの詳細については，「32.5.14　その他の設定」の「(20)　-MaxDigits38 オプション（AIX(64)，Linux(x64)で有効)」を参照してください。

-IntResult,DecFloat40 オプションの詳細については，「32.5.14　その他の設定」の「(21)　-IntResult,DecFloat40 オプション（AIX(64)，Linux(x64)で有効)」を参照してください。

28.2　数字項目のけた拡張機能の詳細

数字項目のけた拡張機能に対応している数字項目，数字編集項目，および数字定数は，最大けた数を 18 けたから 38 けたに拡張できます。数字項目のけた拡張機能の詳細について説明します。

28.2.1　数字項目のけた拡張機能で対象となるデータ項目

数字項目のけた拡張機能の対象となるデータ項目の指定方法および表現形式を次の表に示します。

表 28–1　数字項目のけた拡張機能に対応したデータ項目の指定方法および表現形式

項目	PICTURE 句の指定	USAGE 句の指定	表現形式	数字項目のけた拡張機能への対応
数字項目	数字	DISPLAY	外部 10 進形式	○
		PACKED-DECIMAL または COMPUTATIONAL-3	内部 10 進形式	○
		BINARY または COMPUTATIONAL COMPUTATIONAL-4 COMPUTATIONAL-5	2 進形式	×
	'9'，'X'，'A'だけ	COMPUTATIONAL-X	2 進形式	×
	外部浮動小数点の数字※	DISPLAY	外部浮動小数点形式	×
	PICTURE 句の指定なし	COMPUTATIONAL-1※	内部浮動小数点形式（4 バイト）	×
		COMPUTATIONAL-2※	内部浮動小数点形式（8 バイト）	×
数字編集項目	数字編集	DISPLAY	1 バイトで 1 文字を表現する	○

（凡例）

○：指定できる

×：指定できない

注※

内部浮動小数点形式または外部浮動小数点形式の詳細については，マニュアル「COBOL2002　言語　拡張仕様編」「6.　浮動小数点形式データを扱う機能」を参照してください。

28.2.2　数字項目のけた拡張機能で対象となる定数

数字項目のけた拡張機能の対象となる定数について次の表に示します。

表 28–2　数字項目のけた拡張機能に対応した定数

字類と項類	定数		数字項目のけた拡張機能への対応
数字	数字定数	固定小数点数字定数	○
		浮動小数点数字定数	×
		16 進数字定数	×

（凡例）

○：指定できる

×：指定できない

28.2.3　数字項目のけた拡張機能での有効けた数

算術文に対して，数字項目（外部 10 進項目，内部 10 進項目）および数字編集項目を算術文の作用対象に指定したときの有効けた数に関する共通事項を次に示します。

- 作用対象のデータ記述が同一である必要はありません。

  計算の過程で，すべての必要な変数と小数点の位置が合わせられます。

- 作用対象の数字の最大けた数は 38 けたです。

  また，作用対象の合成※の最大けた数も 38 けたです。

  注※

  作用対象の合成とは，指定された各作用対象を小数点で位置合わせし，重ね合わせてできる仮想のデータ項目です。

28.3　数字項目のけた拡張機能での演算の中間結果

ここでは，数字項目のけた拡張機能での演算の中間結果について説明します。

28.3.1　演算の中間結果

数字項目のけた拡張機能での中間結果には，次の規則があります。

- 中間結果の表現形式は，40 けた 10 進浮動小数点形式です。中間結果の値は作用対象のけた数にかかわらず，演算の結果の値によって，上位けたから有効な 40 けたを格納します。10 進浮動小数点形式の詳細については，「28.3.2　10 進浮動小数点形式について」を参照してください。
- 中間結果の 40 けたに入りきれない値は，下位のけたが切り捨てられます。

数字項目のけた拡張機能での演算の中間結果についての注意事項

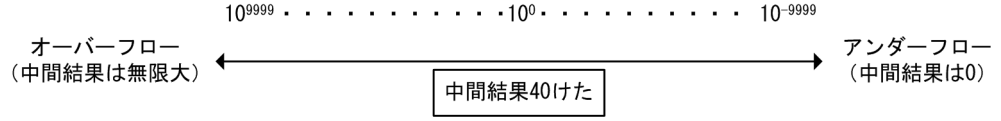
- 次のどれかに該当する演算の場合，内部浮動小数点演算となるため，中間結果に 40 けた 10 進浮動小数点形式は適用されません。
  - ・被べき数（底）またはべき数（指数）が浮動小数点項目または浮動小数点定数のべき乗演算
  - ・べき数（指数）が算術式のべき乗演算
  - ・べき数（指数）が整数でないべき乗演算
  - ・関数値のデータ属性が内部浮動小数点形式の組み込み関数
  - ・浮動小数点項目および浮動小数点定数に対するマイナス（-）単項演算子
  - ・浮動小数点項目および浮動小数点定数同士の演算

28.3.2　10 進浮動小数点形式について

数字項目のけた拡張機能での中間結果の表現形式は，40 けた 10 進浮動小数点形式です。

40 けた 10 進浮動小数点形式は，符号，仮数部（40 けた）および指数部（4 けた）で構成し，図 28-2 に示す範囲の数値を格納します。格納できる値の範囲を超えた場合は，オーバーフロー（指数部が+9999 より大きくなった状態）またはアンダーフロー（指数部が-9999 より小さくなった状態）となります。オーバーフローまたはアンダーフローとなった場合の結果は保証しません。

図 28–2　40 けた 10 進浮動小数点の領域の範囲



中間結果は，演算の結果の値によって有効な 40 けたのけた数（精度）を保持します。図 28-3 および図 28-4 に，数字項目の加算を行う場合の例を示します。

図 28-3　数字項目の加算を行う場合の中間結果（例 1）

```
A PIC 9(38)    VALUE 99999999999999999999999999999999999999.
B PIC v9(38)   VALUE .99999999999999999999999999999999999999.
C PIC 9(38)

COMPUTE C = A + B
```

上記の式の中間結果は，40けた10進浮動小数点の値として
「9.99999999999999999999999999999999999999E37」が保持されます。

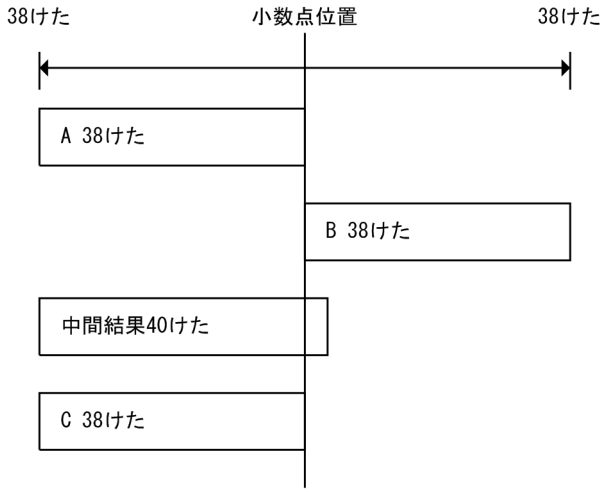
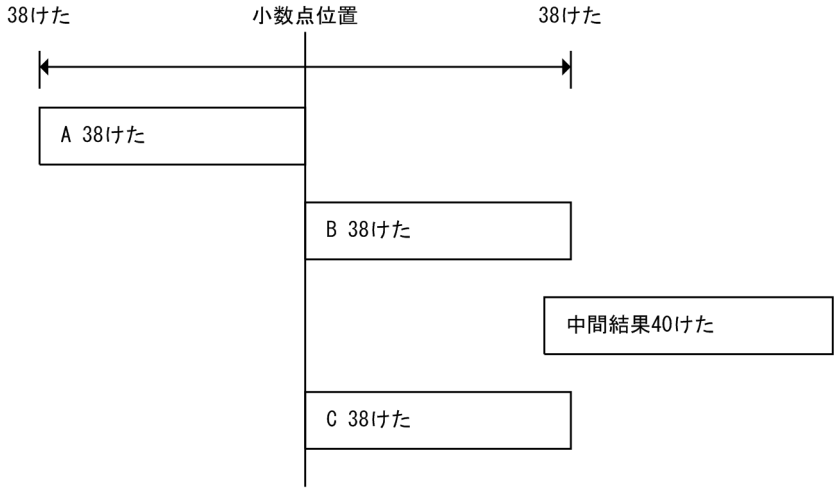


図 28-4　数字項目の加算を行う場合の中間結果（例 2）

```
A PIC 9(38)    VALUE 0.
B PIC v9(38)   VALUE 0.00000000000000000000000000000000000001.
C PIC v9(38)

COMPUTE C = A + B
```

上記の式の中間結果は，40けた10進浮動小数点として
「1.00000000000000000000000000000000000000E-37」が保持されます。



28.4　数字項目のけた拡張機能に対応する機能一覧

ここでは，数字項目のけた拡張機能に対応する機能について説明します。

28.4.1　数字項目のけた拡張機能で対象となる機能

数字項目のけた拡張機能に対応する機能を次に示します。

表 28–3　数字項目のけた拡張機能に対応する機能

種類	機能名	けた拡張機能への対応
規格	基本機能	○
	順編成ファイル	○
	相対編成ファイル	○
	索引編成ファイル	×
	整列併合	×
	プログラム間連絡	○
	組み込み関数	○※1
	オブジェクト指向	×
	共通例外処理	○※2
	再帰呼び出し	○
	利用者定義関数	×
	局所場所節（LOCAL-STORAGE SECTION）	○
	原始文操作	○
	自由形式正書法	○
	TYPEDEF 句と SAME AS 句	○
	翻訳指令	×
	区分化	○
X/Open	テキスト編成ファイル	○
	ファイル共用（ファイルシェア）	○
	コマンド行および環境変数へのアクセス	×
	画面節（SCREEN SECTION）による画面操作	×
	C 言語インタフェース	×
拡張機能	日本語	×

種類	機能名		けた拡張機能への対応
	ブール（ビット処理）		×
	アドレス操作		×
	1 バイト 2 進および COMP-X 項目		×
	浮動小数点データ		×
	ISAM による索引編成ファイル機能の拡張（合成キー，逆順読み）		×
	CSV 編成ファイル		×
	HiRDB による索引編成ファイル		○※3
	ラージファイル入出力		○
	COBOL 入出力サービスルーチン		×
	バイトストリーム入出力サービスルーチン		×
	画面節（WINDOW SECTION）による画面操作		×
	COPY 文の接頭辞／接尾辞		○
	プリンタへのアクセス		○
	ファイルのディスク書き込み保証		○
	報告書作成機能		×
	MIOS7 COBOL85 との互換機能		○
	イベントログファイル／syslog ファイル出力機能		○
	データベース操作機能（ODBC インタフェース）		×
	XDM によるデータベース操作シミュレーション機能	構造型データベース（XDM/SD）操作シミュレーション	×
		リレーショナルデータベース(XDM/RD)操作シミュレーション	○
	OLE2 オートメーション機能		×
	マルチスレッド環境での実行		○
	MSMQ アクセス機能		×
	エンディアン切換え		○
	Unicode 機能		○
	動的長基本項目機能		○
	定数長拡張機能		○
デバッグ機能	実行時デバッグ機能		○

種類	機能名	けた拡張機能への対応
	テストデバッグ機能	○
	カバレージ機能	○
連携機能	XML 連携機能	×
	Cosminexus 連携機能	×

(凡例)

○：使用できる

×：使用できない

注

19～38 けたの数字項目および数字定数を指定できない個所や文については，マニュアル「COBOL2002　言語　拡張仕様編」「21.　数字項目のけた拡張機能」を参照してください。

注※1

組み込み関数の ADDR 関数および LENGTH 関数の引数に，19～38 けたの数字項目，または数字編集項目を指定できます。

注※2

-MaxDigits38 オプションを指定した場合，TURN 指令に次の例外名は指定できません。

EC-SIZE，EC-SIZE-EXPONENTIATION，EC-SIZE-OVERFLOW，EC-SIZE-TRUNCATION，EC-SIZE-UNDERFLOW，EC-SIZE-ZERO-DIVIDE

注※3

AIX(64)で対応しています。

28.4.2　数字項目のけた拡張機能で対象となるソース単位

数字項目のけた拡張機能で対象となるソース単位について，次に示します。

表 28–4　数字項目のけた拡張機能で対象となるソース単位

定義名	数字項目のけた拡張機能への対応
プログラム定義	○
関数定義	×
クラス定義	×
ファクトリ定義	×
インスタンス定義	×
メソッド定義	×
インタフェース定義	×

(凡例)

○：使用できる

×：使用できない

28.4.3　数字項目のけた拡張機能で対象となる節や文

数字項目のけた拡張機能の対象となる節や文について，次に示します。

(1)　環境部

環境部では，データ名および定数の指定できる個所に 19～38 けたの数字項目，数字編集項目，および数字定数の定義や参照はできません。

(2)　データ部

数字項目のけた拡張機能の対象となる節を次に示します。

表 28–5　数字項目のけた拡張機能の対象となる節

節名	数字項目のけた拡張機能への対応
ファイル節（FILE SECTION）	○
作業場所節（WORKING-STORAGE SECTION）	○
局所場所節（LOCAL-STORAGE SECTION）	○
連絡節（LINKAGE SECTION）	○
報告書節（REPORT SECTION）	×
画面節（SCREEN SECTION）※	×
画面節（WINDOW SECTION）※	×
通信節（COMMUNICATION SECTION）	×
サブスキーマ節（SUBSCHEMA SECTION）	×

（凡例）

○：使用できる

×：使用できない

注※

AIX(64)で有効です。

数字項目のけた拡張機能の対象となるファイルを次に示します。

表 28–6　数字項目のけた拡張機能の対象となるファイル

ファイル編成	数字項目のけた拡張機能への対応
順編成ファイル	○
相対編成ファイル	○
索引編成ファイル	×
テキスト編成ファイル	○

ファイル編成	数字項目のけた拡張機能への対応
CSV 編成ファイル	×
整列併合ファイル	×
報告書ファイル	×
HiRDB による索引編成ファイル	○※

(凡例)

○：使用できる

×：使用できない

注※

AIX(64)で有効です。

数字項目のけた拡張機能の対象となるその他のデータ部の句について，次に示します。

表 28–7　数字項目のけた拡張機能の対象となるその他のデータ部の句

データ部の句		数字項目のけた拡張機能への対応
BLANK WHEN ZERO 句		○
BLOCK CONTAINS 句		×
DATA RECORDS 句		×
EXTERNAL 句		×
LINAGE 句		×
OCCURS 句		×
PICTURE 句	記号「P」を含む場合	×
USAGE 句	DISPLAY COMPUTATIONAL-3 または COMP-3 PACKED-DECIMAL	○
	上記以外	×
VALUE 句（作業場所節，局所場所節，ファイル節，連絡節）		○

(凡例)

○：使用できる

×：使用できない

(3) 手続き部

数字項目のけた拡張機能の対象となる手続き部見出しおよび手続き文について，次に示します。各手続き文の対応の詳細については，マニュアル「COBOL2002　言語　標準仕様編」およびマニュアル「COBOL2002　言語　拡張仕様編」を参照してください。

表 28-8　数字項目のけた拡張機能の対象となる手続き部見出しおよび手続き文

手続き部見出しおよび手続き文	数字項目のけた拡張機能への対応
手続き部見出しの引数と返却項目	○
ACCEPT 文	×
ADD 文	○
ALTER 文	−
CALL 文	△
CANCEL 文	−
CLOSE 文	−
COMMIT 文	−
COMPUTE 文	○
CONTINUE 文	−
DELETE 文	−
DISABLE 文	×
DISPLAY 文	△
DIVIDE 文	○
ENABLE 文	×
ENTER 文	−
ENTRY 文	○
ERASE 文	×
EVALUATE 文	○
EXAMINE 文	×
EXIT 文	−
GO TO 文	×
GOBACK 文	−
IF 文	○
INITIALIZE 文	○

手続き部見出しおよび手続き文	数字項目のけた拡張機能への対応
INSPECT 文	×
INVOKE 文	×
MERGE 文	×
MOVE 文	○
MULTIPLY 文	○
OPEN 文	−
PERFORM 文	△
RAISE 文	−
READ 文	△
RECEIVE 文	×
RELEASE 文	×
REPLY 文	×
RESUME 文	−
RETURN 文	×
REWRITE 文	○
ROLLBACK 文	−
SEARCH 文	△
SEND 文	×
SET 文	△
SORT 文	×
START 文	×
STOP 文	×
STRING 文	○
SUBTRACT 文	○
TRANSCEIVE 文	×
TRANSFORM 文	×
UNLOCK 文	−
UNSTRING 文	×
USE 文	−
WAIT 文	−
WRITE 文	△

（凡例）

○：使用できる

△：一部使用できない構文がある

×：使用できない

－：該当する指定がない

28.5　数字項目のけた拡張機能の注意事項

ここでは，数字項目のけた拡張機能の注意事項について説明します。

28.5.1　数字項目のけた拡張機能を使用する場合の演算結果

数字項目のけた拡張機能を使用する場合，中間結果で保持するけた数が 40 けたとなります。そのため，従来の 1～18 けたの演算であっても，数字項目のけた拡張機能を使用しない場合と演算を含む算術式および条件式の結果が異なるときがあります。数字項目のけた拡張機能を使用しない場合の結果と合わせるときは，別の数字項目を使用して演算を分けてください。

```
(例)
  01 A  PIC 9(4)  VALUE  0.
  01 B  PIC 9(2)  VALUE 10.
  01 C  PIC 9(1)  VALUE  4.
          :
  COMPUTE A = (B / C) * 100.
```

上記の COMPUTE 文は，次のような連続した操作に変換します。なお，temp はこのシステムが用意する中間結果項目です。

```
  B / C      → temp
  temp * 100 → A
```

数字項目のけた拡張機能を使用する場合，演算結果は次のようになります。

表 28-9　数字項目のけた拡張機能を使用する場合の演算結果

数字項目のけた拡張機能の使用有無	temp の属性	temp に設定される（B / C）の値	temp * 100 の値	A に設定される値
使用する	40 けた 10 進浮動小数点	2.500・・・・000 （40 けたの数値）	250.000・・・・000 （40 けたの数値）	250
使用しない	次のけた数の数字項目※ 整数部のけた数：2 小数部のけた数：なし	2	200	200

注※

数字項目のけた拡張機能を使用しないときの中間結果のけた数は，「5.2.4　演算の中間結果」を参照してください。

数字項目のけた拡張機能を使用する場合の計算結果を，数字項目のけた拡張機能を使用しないときと同じにするには，次のように除算（B/C）の結果を別の数字項目 WK1（数字項目のけた拡張機能を使用しない場合の temp の属性と同じけた数で定義）に転記したもので演算します。

```
  01 A   PIC 9(4)  VALUE  0.
  01 B   PIC 9(2)  VALUE 10.
```

```
  01 C   PIC 9(1)  VALUE  4.
  01 WK1 PIC 9(2)  VALUE  0.
         :
  COMPUTE WK1 = B / C.
  COMPUTE A = WK1 * 100.
```

28.5.2　内部浮動小数点項目から固定小数点形式の数字項目への転記

内部浮動小数点項目から固定小数点形式の数字項目への転記で，固定小数点形式の数字項目が内部浮動小数点項目の有効けたを超える下位のけたを持つ場合，下位のけたが大きくなるほど，誤差の影響も大きくなります。

内部浮動小数点数の小数点位置合わせに伴う誤差の影響が，けた数の拡張によって現れる例を次に示します。なお，内部浮動小数点数の有効けた数および標準けた寄せ規則については，マニュアル「COBOL2002　言語　拡張仕様編」「6.　浮動小数点形式データを扱う機能」を参照してください。

```
（例）
  01 A  USAGE COMP-2    VALUE 2.0E+0.
  01 B  PIC 9(1)V9(14).
  01 C  PIC 9(1)V9(23).
         :
  COMPUTE B = A.
  COMPUTE C = A.
         :

 （出力結果）
  B = +200000000000000
  C = +19999999999999999983222784
```

29 サービスルーチン

サービスルーチンは，COBOL の言語仕様にない機能を，CALL 文で呼び出すプログラムとして提供しているものです。この章では，このシステムが提供しているサービスルーチンの機能と使い方について説明します。

29.1　サービスルーチンの概要

サービスルーチンは，COBOL の言語仕様にない機能を，CALL 文で呼び出すプログラムとして提供しているものです。

ここでは，このシステムが提供しているサービスルーチンについて説明します。

29.1.1　プログラム実行制御

主に COBOL プログラムの開始／終了時に，プログラムを制御するサービスルーチンです。

プログラム実行制御のサービスルーチンを，次に示します。

サービスルーチン	機能	OS	
		AIX	Linux
CBLEND※	COBOL の実行環境を終了させる。	○	○
CBLABN	プログラムを異常終了させる。	○	○
CBLARGC	コマンド行に指定した引数の個数を取得する。	○	○
CBLARGV	コマンド行に指定した引数の内容を取得する。	○	○

（凡例）

○：サポートしている

注※

COBOL プログラムの CALL 文で呼び出して使用するのではなく，他言語のプログラムから呼び出して使用するサービスルーチンです。

サービスルーチンの詳細については，「29.4　プログラム実行制御」を参照してください。

29.1.2　ダイアログボックス／ウィンドウ

ダイアログボックスへの入出力，およびウィンドウへの入出力を制御するサービスルーチンです。

ダイアログボックスのサービスルーチンを，次に示します。

サービスルーチン	機能	OS	
		AIX	Linux
JCPOPUP	表形式のデータ項目を主画面とは別の画面に表示し，選ばれたブロック番号をインタフェース領域に格納する。	○	×

（凡例）

○：サポートしている

×：サポートしていない

サービスルーチンの詳細については，「29.5　ダイアログボックス／ウィンドウ」を参照してください。

29.1.3　デバッグ機能

COBOL が使用できるアプリケーションデバッグ機能によって，データ領域ダンプリストを出力するサービスルーチンです。

デバッグ機能のサービスルーチンを，次に示します。

サービスルーチン	機能	OS	
		AIX	Linux
CBLDATADUMP	COBOL プログラム実行時の任意の時点でのデータ領域ダンプリストを出力する。	○	○

（凡例）

○：サポートしている

サービスルーチンの詳細については，「29.6　デバッグ機能」を参照してください。

29.1.4　変換・転記・演算

特殊な変換・転記・演算をするためのサービスルーチンです。

変換・転記・演算のサービスルーチンを，次に示します。

サービスルーチン	機能	OS	
		AIX	Linux
CBLNCNV	英字・英数字・英数字編集・数字編集項目を日本語・日本語編集項目に変換する。	○	○
CBLUBIT	ビットデータを論理演算する。	○	○
CBLCNVERRORINFO	組み込み関数の CONVERT-CODE 関数，DISPLAY-OF 関数または NATIONAL-OF 関数のエラー情報を取得する。	○	○

（凡例）

○：サポートしている

サービスルーチンの詳細については，「29.7　変換・転記・演算」を参照してください。

29.1.5　データベース操作機能（Linux で有効）

SQL 文による ODBC インタフェース使用時に，出力された実行時メッセージのエラー情報を取得するサービスルーチンです。

サービスルーチンの詳細については，「29.8　データベース操作機能」を参照してください。

29.1.6　COBOL の入出力機能

COBOL の入出力機能についてのサービスルーチンは次のとおりです。

- COBOL で作成したファイルを他言語のプログラムから入出力するためのサービスルーチン
  サービスルーチンの詳細については，「13.　COBOL 入出力サービスルーチン」を参照してください。
- 他言語のプログラムで作成したバイナリファイルを COBOL で入出力するためのサービスルーチン
  サービスルーチンの詳細については，「15.　バイトストリーム入出力サービスルーチン」を参照してください。

29.1.7　XMAP3 を使用した画面・帳票関連（AIX(32)で有効）

XMAP3 を使用した書式印刷機能，XMAP3 を使用した通信節による画面操作・帳票出力機能に関するサービスルーチンです。

サービスルーチンの詳細については，「29.9　XMAP3 を使用した画面・帳票関連」を参照してください。

29.2　戻り値の使い方

各サービスルーチンの戻り値は，COBOL プログラムでは RETURN-CODE 特殊レジスタで参照できます。

サービスルーチンを呼び出した直後は，戻り値を参照し，異常終了時またはエラー発生時の処理を記述しておくことを推奨します。詳細は，「29.4.4　CBLARGV」の使用例を参照してください。

29.3　サービスルーチン使用時の注意事項

サービスルーチンを使用する場合の注意事項を，次に示します。

- Linux の場合，-BigEndian,Bin オプションを指定すると，サービスルーチンの引数で要求する 2 進項目がビッグエンディアン形式になるため正常に動作しません。この場合は，引数となる 2 進項目を COMP-5 で定義してください。
- 引数を省略したり，引数の属性や長さを誤って指定した場合，動作は保証しません。
- 各サービスルーチンの引数の説明では，2 進形式の項目を指定する場合，「4 バイトの 2 進項目」のように，割り当てられる項目のサイズであるバイト数を示すものがあります。

  2 進形式の項目では，PICTURE 句で指定した項目のけた数と，その項目に割り当てられる項目のサイズは異なります。

  次に関係を示します。

表 29–1　2 進形式の項目のけた数とサイズ

PICTURE 句で指定したけた数	項目が占めるバイト数
1～2 けた	2 バイト （1 バイト 2 進機能，または COMP-X 項目の場合は，1 バイト）
3～4 けた	2 バイト
5～9 けた	4 バイト
10～18 けた	8 バイト

29.4 プログラム実行制御

29.4.1 CBLEND

CBLEND サービスルーチンは，-MainNotCBL オプションを指定してコンパイルしたプログラムが動作したときに，設定した COBOL の実行環境を終了させるものです。

このサービスルーチンは，CALL 文で呼び出すのではなく，COBOL 以外のプログラム内で使用します。

記述例

```
extern int CBLEND();        /* CBLENDの外部参照宣言 */
CBLEND();                   /* CBLENDの呼び出し     */
```

引数

なし。

戻り値

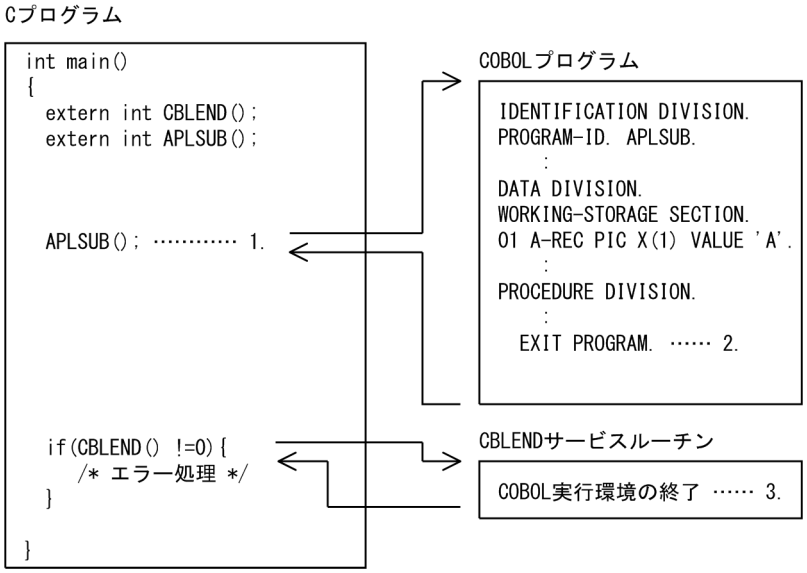
0：正常終了した場合

-1：COBOL プログラムが動作中であるため，CBLEND サービスルーチンが無視された場合

注意事項

- CBLEND サービスルーチンは，COBOL プログラムが終了したあと，1 回だけ呼び出さなければなりません。
- COBOL プログラム内で STOP RUN 文を実行した場合，CBLEND サービスルーチンを呼び出す必要はありません。
- COBOL プログラム内で STOP RUN 文を実行したあと，CBLEND サービスルーチンを呼び出した場合，何も処理しないでリターンします。
- CBLEND サービスルーチンは，COBOL 以外の言語から呼び出さなければなりません。
  CBLEND サービスルーチンを COBOL プログラムから呼び出した場合，結果は保証しません。
- CBLEND サービスルーチンを呼び出したあとでも，再度 COBOL プログラムを呼び出せます。この場合，CBLEND サービスルーチンを呼び出す前の COBOL 実行環境が終了しているため，新たな実行環境で COBOL プログラムが実行されます。
- 再度 COBOL プログラムを呼び出さないで，CBLEND サービスルーチンを複数回呼び出した場合，2 回目以降の呼び出しでは何も処理しないでリターンします。
- このサービスルーチンを使用するプログラムでは，リンク時に次の共用ライブラリを指定してください。
  AIX(32)の場合：libcbl2k.a
  AIX(64)の場合：libcbl2k64.a
  Linux の場合：libcbl2k.so

使用例



1. C 言語で記述された main 関数から，-MainNotCBL オプション指定でコンパイルした COBOL プログラム APLSUB を呼び出します。
2. APLSUB は，COBOL 副プログラムとして処理されるため，EXIT PROGRAM 文で制御が戻るとき，COBOL 実行環境が終了されません。
3. CBLEND サービスルーチンを呼び出して，COBOL 実行環境を終了します。

29.4.2 CBLABN

CBLABN サービスルーチンは，利用者がプログラムを異常終了させるときに呼び出すものです。このサービスルーチンを実行すると，引数 1 に指定した要因コードと実行時メッセージを表示し，実行プロセスが異常終了します。

形式

```
CALL 'CBLABN' USING 引数1
```

引数

引数 1 には，異常終了時の要因コードを 2 バイトの 2 進項目で指定します。

戻り値

なし。

規則

- このサービスルーチンは，COBOL プログラムだけから呼び出せます。
- このサービスルーチンは呼び出し元に制御を戻しません。
- VOS3 COBOL85 との互換性を保つ必要があるときは，引数 1 の要因コードは 0～4,095 の範囲で指定します。

使用例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
           :
       WORKING-STORAGE SECTION.
       01 AB-CODE PIC S9(4) USAGE COMP. …1.
           :
       PROCEDURE DIVISION.
           :
           MOVE 100 TO AB-CODE.  …2.
           CALL 'CBLABN' USING AB-CODE.   …3.
           :
```

1. 異常終了時の要因コードを 2 バイトの 2 進項目で指定します。
2. 異常終了時の要因コードを設定します。
3. CBLABN サービスルーチンを呼び出します。

29.4.3 CBLARGC

CBLARGC サービスルーチンは，コマンド行に指定した引数の個数を，引数 1 で指定した領域に格納するものです。コマンド引数の個数は，C 言語の main 関数で受け取る argc に該当します。

形式

```
CALL 'CBLARGC' USING 引数1
```

引数

引数 1 には，コマンド行に指定した引数の個数を受け取る領域を 4 バイトの 2 進項目で指定します。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

規則

- このサービスルーチンは，-Main,System オプションを指定した最外側の COBOL プログラムだけから呼び出せます。これ以外のプログラムから呼び出した場合は，戻り値-1 が返されます。
- このサービスルーチンが異常終了した場合，CBLARGC に渡された引数 1 の内容は保証しません。

使用例

「29.4.4 CBLARGV」の使用例を参照してください。

29.4.4 CBLARGV

CBLARGV サービスルーチンは，コマンド行に指定した引数の内容を，引数 2 の領域に転送するものです。コマンド引数の内容は，C 言語の main 関数で受け取る argv に該当します。

形式

```
CALL ’CBLARGV’ USING 引数1 引数2
```

引数

- 引数 1 には，CBLARGV サービスルーチンによって受け取るコマンド引数の順序を 4 バイトの 2 進項目で指定します。
- 引数 2 には，コマンド引数の情報を受け取る領域を指定します。この項目は次の形式で定義します。

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 PARM.
         02 PARM-LENGTH PIC 9(4) USAGE COMP.  …1.
         02 PARM-AREA.
           03 PARM-AREA-DETAIL PIC X(1)
                    OCCURS 1 TO 100 TIMES  …2.
                    DEPENDING ON PARM-LENGTH.
```

1. コマンド引数の長さを格納する領域で，2 バイトの 2 進項目で指定します。コマンド引数が 100 バイトを超えたときは常に 100 が設定されます。
2. コマンド引数の値を格納する領域で，最大長 100 バイトの可変長項目で定義します。コマンド引数が 100 バイトを超えたときは先頭から 100 バイトが格納されます。

戻り値

0：正常終了した場合

1：コマンド引数の長さが 100 バイトを超えた場合

2：CBLARGC で戻された引数の個数を超えた値で引数 1 を指定した場合，またはこのサービスルーチンに引き渡された引数 1（引数の順序）に示される引数を，実行する実行可能ファイルの引数に指定していない場合

-1：エラーが発生した場合（-Main,System オプションを指定した最外側の COBOL プログラム以外から呼び出した場合）

規則

- このサービスルーチンは，-Main,System オプションを指定した最外側の COBOL プログラムだけから呼び出せます。これ以外のプログラムから呼び出した場合は，戻り値-1 が返されます。
- このサービスルーチンの戻り値が 0 または 1 以外の場合，CBLARGV に渡された引数 2 の内容は保証しません。
- 引数 2 の領域は，コマンド引数として受け取れる最大長分の 100 バイトを準備する必要があります。引数 2 に 100 バイト未満の領域を指定した場合，領域を破壊することがあります。

- コマンド引数の長さが 100 バイト未満の場合，引数 2 の引数格納領域の余った領域には空白が設定されます。

使用例

```
       IDENTIFICATION DIVISION.
             :
       DATA DIVISION.
        WORKING-STORAGE SECTION.
         77 ARGC    PIC 9(9) USAGE COMP.
         77 COUNTER PIC 9(9) USAGE COMP.
             :
         01 PARM.                                  …1.
           02 PARM-LENGTH PIC 9(4) USAGE COMP. …2.
           02 PARM-AREA.                           …3.
               03 PARM-AREA-C PIC X(1)
                   OCCURS 1 TO 100
                   DEPENDING ON PARM-LENGTH.
             :
       PROCEDURE DIVISION.
             :
           CALL 'CBLARGC' USING ARGC.
           IF RETURN-CODE NOT = 0 THEN
               CBLARGC異常時の処理
           END-IF.
             :
           MOVE 1 TO COUNTER.
           PERFORM UNTIL ARGC = 0
               MOVE SPACES TO PARM-AREA
               CALL 'CBLARGV' USING COUNTER PARM
               IF RETURN-CODE NOT = 0 THEN
                  CBLARGV異常時の処理
               END-IF
               CBLARGVで受け取ったパラメタに対応する処理
               ADD 1 TO COUNTER
               SUBTRACT 1 FROM ARGC
           END-PERFORM.
             :
```

1. CBLARGV に引き渡す実引数領域です。
2. CBLARGV によって 3.に示す引数格納領域に設定された文字列の長さを示します。最大長を超えて引数を設定したときは 100 が設定されます。
3. CBLARGV で設定する引数領域で，最大長 100 バイトの領域として可変長で定義します。100 バイト以上の引数を指定した場合，実際のコマンドに指定した引数情報の先頭から 100 バイトまでが設定されます。

29.5　ダイアログボックス／ウィンドウ

JCPOPUP サービスルーチンは，Linux(x86) COBOL2002 および Linux(x64) COBOL2002 では提供していません。

29.5.1　JCPOPUP（AIX で有効）

JCPOPUP サービスルーチンは，表形式のデータ項目を主画面とは別の画面に表示し，画面中でブロックカーソルを移動して選んだブロック番号をインタフェース領域に格納するものです。これを利用すると，画面上での各種のコードの入力などが目的の項目をマウスやキーで選ぶだけでできるようになります。

なお，ここで表示される画面をポップアップブロックカーソル画面といいます。

形式

```
CALL ’JCPOPUP’ USING 引数1 引数2
```

引数

- 引数 1 には，次に示すインタフェース領域の名前を指定します。

表 29-2　JCPOPUP サービスルーチンのインタフェース領域

記述形式	内容
01 データ名01.	CALL 文の USING で指定するインタフェース領域の名前を指定する。
02 データ名02 PIC 9(02).	JCPOPUP サービスルーチンが次の戻り値を設定する。 00：正常終了した。 10：異常終了した。
02 データ名03 PIC 9(04).	JCPOPUP サービスルーチンがブロックカーソルで選択したブロック番号を設定する（先頭を 1 とし，固定部分は含まない）。
02 データ名04 PIC 9(02).	データを表示する画面上の行の番号を 1～99 で指定する。※1
02 データ名05 PIC 9(02).	データを表示する画面上のカラムの番号を 1～99 で指定する。※1
02 データ名06 PIC 9(02).	画面の行数を 1～24 で指定する。
02 データ名07 PIC 9(02).	画面の列数を 1～80 で指定する。
02 データ名08 PIC 9(02).	画面の固定領域行数を 1～24 で指定する。
02 データ名09 PIC 9(02).	画面の 1 ブロック（1 エントリ）のサイズを指定する。
02 データ名10 PIC X(02).	固定領域の色を指定する。 G△：緑色　W△：白色　R△：赤色　B△：青色 P△：紫色　Y△：黄色　S△：空色

記述形式	内容
02 データ名11 PIC X(02).	ブロックカーソルの色をデータ名 10 と同じ形式で指定する。
02 データ名12 PIC X(02).	可変領域の色をデータ名 10 と同じ形式で指定する。
02 データ名13 PIC X(02).	可変領域の初期表示位置を指定する。 T△：先頭表示 B△：最終表示
02 データ名14 PIC 9(04).	画面の固定領域のサイズを指定する。
02 データ名15 PIC 9(02).	可変領域のデータ項目（1 エントリ）のサイズを指定する。
02 データ名16 PIC 9(04).	出力データのサイズ（固定領域＋可変領域）を指定する。
02 データ名17 PIC 9(02).	JCPOPUP サービスルーチンが終了キーコードを設定する。※2
02 データ名18 PIC X(01).	画面に枠けい線を出力するかどうかを指定する。 Y：出力する。 N：出力しない。
02 FILLER PIC X(09).	予備。値は X'00'でなければならない。ただし，このシステムではこの領域は使用しない。

（凡例）

△：半角空白を示す

注※1

使用できません。ポップアップウィンドウの出力位置は UNIX のシステムに依存します。

注※2

終了キーと終了キーコードとの対応を次に示します。

Enter キー：89

PF1 キー：00　PF9 キー：08　PF17 キー：53

PF2 キー：01　PF10 キー：09　PF18 キー：54

PF3 キー：02　PF11 キー：18　PF19 キー：55

PF4 キー：03　PF12 キー：19　PF20 キー：56

PF5 キー：04　PF13 キー：49　PF21 キー：57

PF6 キー：05　PF14 キー：50　PF22 キー：58

PF7 キー：06　PF15 キー：51　PF23 キー：59

PF8 キー：07　PF16 キー：52　PF24 キー：60

- 引数 2 には，1 次元の繰り返し構造（表形式）を持つレベル番号 01 のデータ項目の名前を指定します。

（例）

```
01 データ名20.
  02 FILLER  PIC  X(16)  VALUE  LOW-VALUE.
  02 表示データ項目（固定領域+可変領域）を
     表形式で指定する。… 1.
  02 FILLER  PIC  X(nn)  VALUE  LOW-VALUE. … 2.
```

1. 表示データ項目のデータは文字形式でなければなりません。文字形式でない場合，出力結果は保証しません。

2. nn には次のサイズを指定します。

   ・枠けい線機能を使用しない場合：32 バイト

   ・枠けい線機能を使用する場合：（可変領域のサイズ／可変領域のデータ項目のサイズ）×9 ＋ 50 バイト

   ただし，この領域はこのシステムでは使用しません。

戻り値

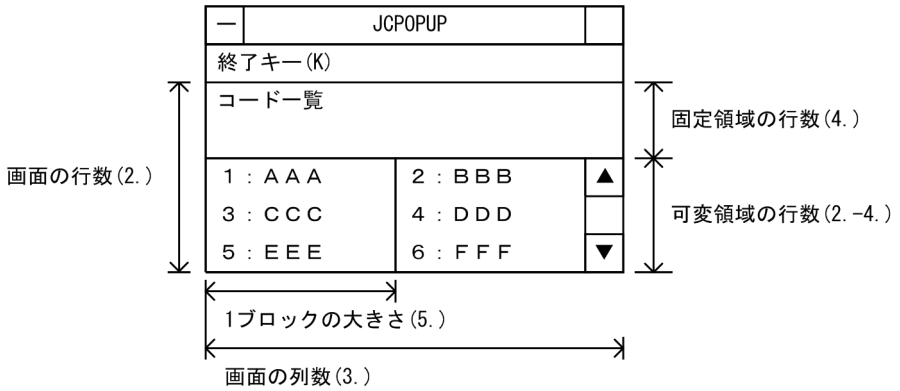
なし。

使用例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
           :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ICHIRAN-GAMEN.
         02 HUKKI-CODE      PIC 9(2).
         02 BLOCK-NUMBER    PIC 9(4).  …1.
         02 Y-LOCATION      PIC 9(2) VALUE 10.
         02 X-LOCATION      PIC 9(2) VALUE 10.
         02 WINDOW-LINE     PIC 9(2) VALUE  5. …2.
         02 WINDOW-COLUMN   PIC 9(2) VALUE 20. …3.
         02 KOTEI-LINE      PIC 9(2) VALUE  2. …4.
         02 BLOCK-SIZE      PIC 9(2) VALUE 10. …5.
         02 KOTEI-COLOR     PIC X(2) VALUE 'G '.
         02 CURSOR-COLOR    PIC X(2) VALUE 'R '.
         02 KAHEN-COLOR     PIC X(2) VALUE 'B '.
         02 HYOUJI-ICHI     PIC X(2) VALUE 'T '. …6.
         02 KOTEI-SIZE      PIC 9(4) VALUE 60. …7.
         02 ENTRY-SIZE      PIC 9(2) VALUE 15. …8.
         02 DATA-SIZE       PIC 9(4) VALUE 210. …9.
         02 END-KEY         PIC 9(2). …10.
         02 WAKU-KEISEN     PIC X(1) VALUE 'Y'. …11.
         02 FILLER          PIC X(9) VALUE LOW-VALUE.
       01 ICHIRAN-DATA.
         02 FILLER          PIC X(16) VALUE LOW-VALUE.
         02 HD1             PIC X(15) VALUE 'コード一覧'.
         02 HD2             PIC X(15) VALUE SPACE.
         02 HD2             PIC X(15) VALUE SPACE.
         02 HD2             PIC X(15) VALUE SPACE.
         02 ATEM            PIC X(15) OCCURS 10 TIMES.
         02 FILLER          PIC X(140) VALUE LOW-VALUE.
           :
```

```
       PROCEDURE DIVISION.
           :
           MOVE '1:AAA' TO ATEM(1).
           MOVE '2:BBB' TO ATEM(2).
           :
           MOVE '10:JJJ' TO ATEM(10).
           CALL 'JCPOPUP' USING ICHIRAN-GAMEN ICHIRAN-DATA.
           :
```

上記の指定によって表示されるポップアップブロックカーソル画面を次に示します。図中の数字は，プログラム中のコメントに記載している数字の個所と対応しています。



固定領域には，表示データ項目の先頭から固定領域のサイズ分（7.）の文字が，固定領域に表示できる分（3.×4.）だけ表示されます。

可変領域には，表示データ項目のうち，可変領域のサイズ（表示データ項目のサイズ（9.）－固定領域のサイズ（7.））分の文字が，可変領域のデータ項目のサイズ（8.）ごとに区切って各ブロックに表示されます。ただし，表示されるのは各項目とも先頭から 1 ブロック分のサイズ（5.）までで，はみ出す部分は表示されません。なお，上図で表示されていない"7：GGG"以降の項目はスクロールによって表示できます。

上記の画面で［↑］キー，［↓］キー，［←］キー，［→］キー，またはマウスでブロックカーソルを移動すると，ブロックを選べます。そのあと，終了キーを押すかマウスでダブルクリックすると，画面が閉じ，選んだブロックの番号が BLOCK-NUMBER の領域（1.）に格納されます。

また，このとき押された終了キーに対応する終了キーコードが 10.の領域に格納されます。ダブルクリックで終了した場合の終了キーコードは 89 です。

PF キーの割り当て

PF キーとして割り当てられているデフォルトのキーを示します。

［PF1］キー：［PF1］キー　［PF13］キー：［Ctrl］＋［PF1］キー
［PF2］キー：［PF2］キー　［PF14］キー：［Ctrl］＋［PF2］キー
［PF3］キー：［PF3］キー　［PF15］キー：［Ctrl］＋［PF3］キー
［PF4］キー：［PF4］キー　［PF16］キー：［Ctrl］＋［PF4］キー
［PF5］キー：［PF5］キー　［PF17］キー：［Ctrl］＋［PF5］キー
［PF6］キー：［PF6］キー　［PF18］キー：［Ctrl］＋［PF6］キー
［PF7］キー：［PF7］キー　［PF19］キー：［Ctrl］＋［PF7］キー
［PF8］キー：［PF8］キー　［PF20］キー：［Ctrl］＋［PF8］キー

[PF9]キー：[PF9]キー　[PF21]キー：[Ctrl]＋[PF9]キー

[PF10]キー：[PF10]キー　[PF22]キー：[Ctrl]＋[PF10]キー

[PF11]キー：[PF11]キー　[PF23]キー：[Ctrl]＋[PF11]キー

[PF12]キー：[PF12]キー　[PF24]キー：[Ctrl]＋[PF12]キー

ただし，この割り当てキーがMotifのキーボード操作，およびそのデフォルトとして割り当てられている場合は，Motifのデフォルト設定が優先されます。詳細は，Motifについて記載されているマニュアルを参照してください。

その他の説明

リソース

JCPOPUPサービスルーチンでは，ポップアップウィンドウの動作環境を定義するためのリソースを用意しています。ユーザがリソースファイル中のリソース値を設定することで，ポップアップウィンドウの終了キーメニュー中に表示するファンクションキーなどを変更できます。

JCPOPUPサービスルーチン固有のリソース値については，「付録Ⅰ　サービスルーチンのリソース一覧」を参照してください。

次に，設定方法を示します。

AIX(32)の場合

```
Cbl2002popup*リソース名:リソース値
```

AIX(64)の場合

```
Cbl2002popup64*リソース名:リソース値
```

29.6　デバッグ機能

29.6.1　CBLDATADUMP

CBLDATADUMP サービスルーチンは，COBOL プログラム実行時の任意の時点でのデータ領域ダンプリストを出力するサービスルーチンです。

通常は，COBOL プログラムが異常終了したときにデータ領域ダンプリストを出力しますが，CBLDATADUMP サービスルーチンによって，プログラム実行中に異常終了時と同様のデータ領域ダンプリストを出力できます。これにより，異常終了時だけでなく，プログラム実行中の任意の時点のデータ領域の状態を求めることができます。

データ領域ダンプリストには，CBLDATADUMP サービスルーチンを呼び出すまでに実行されたすべてのプログラムのうち，デバッグ対象プログラムのデータ領域の状態が出力されます※。

データ領域ダンプリストは，CBLDATADUMP サービスルーチンを実行するごとに追加書きされるので，各時点のデータをトレース情報として保持できます。

データ領域ダンプリストについては，「36.3　データ領域ダンプリスト」を参照してください。

CBLDATADUMP サービスルーチンは，次に示す場合に実行します。

- COBOL プログラムのデバッグ時に，ある状態のデータ領域の内容を参照する場合
- COBOL プログラムのデバッグ時に，データ領域の内容の遷移を確認する場合
- COBOL プログラムが異常終了したときに，データ領域の内容の遷移を採取し，異常終了の原因を調査する場合

注※

- デバッグ対象プログラムは，次のどれかのコンパイラオプションを指定したプログラムを示します。どのプログラムをデータ領域ダンプリストに出力するかは，これらのコンパイラオプションの指定有無によって切り分けることができます。

  -DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange
- デバッグ対象プログラムであっても，プログラムの実行状態などによって，情報が出力されない場合があります。例えば，実行後に CANCEL されたプログラムはダンプ出力の対象となりません。情報出力の対象となるプログラムの実行状態については，規則のデータ領域ダンプリスト出力の対象となるプログラムを参照してください。

記述例

```
CALL  'CBLDATADUMP'
```

引数

なし。

戻り値

戻り値は次のように設定されます。

0：正常終了した場合

1：環境変数 CBLDATADUMPFILE が指定されていない場合

2：どのプログラムのデータ状態も出力されなかった場合

どのプログラムのデータ状態も出力されなかった場合，データ領域ダンプリストにはヘッダだけが出力されます。次の場合に該当します。

- 1 回以上実行されたプログラムにデバッグ対象プログラムがない場合
- 実行されたすべてのデバッグ対象プログラムが，規則のデータ領域ダンプリスト出力の対象となるプログラムではなく，EXTERNAL 句を指定したファイル名およびデータ名の出力もない場合
- データ領域ダンプリスト出力情報の選択で，選択した範囲に出力情報がない場合

  データ領域ダンプリスト出力情報の選択方法については，「36.4　データ領域ダンプリスト出力情報の選択」を参照してください。

-1：データ領域ダンプリストの出力中にエラー※が発生した場合

注※

次の要因が考えられます。

- 環境変数 CBLDATADUMPFILE で指定したファイルが開けない
- データ領域ダンプリストの出力処理中に入出力エラーが発生した
- 環境変数 CBLDATADUMPFILE で指定したファイルを閉じられない

規則

- データ領域ダンプリストの出力先は，環境変数 CBLDATADUMPFILE で指定します。環境変数 CBLDATADUMPFILE の指定がない場合，または出力先が指定されていない場合，データ領域ダンプリストは出力されません。環境変数 CBLDATADUMPFILE の指定方法については，「36.3.2 データ領域ダンプリストの出力先」を参照してください。
- データ領域ダンプリストの出力対象とするプログラムは，プログラム翻訳時に次のどれかのコンパイラオプションの指定が必要です。

  -DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange
- データ領域ダンプリスト出力の対象となるのは次のプログラムです。

  ・1 回以上実行され，CANCEL 文で取り消されていないプログラム

  ・1 回以上実行された利用者定義関数

  ・CBLDATADUMP サービスルーチン実行時に動作中のメソッド

  ・CBLDATADUMP サービスルーチン実行時に動作中のメソッドを含むクラス

注意事項

- データ領域ダンプリストは追加書きのため，古い情報が残ります。そのため，不要なデータ領域ダンプリストは削除してください。

使用例

CBLDATADUMP サービスルーチン呼び出しの例を次に示します。

```
       IDENTIFICATION  DIVISION.
       PROGRAM-ID.  SAMPLE1.
              :
       DATA  DIVISION.
       WORKING-STORAGE  SECTION.
       01 DATA1 PIC S9(4).
       01 DATA2 PIC S9(4) VALUE 1.
       01 DATA3 PIC S9(4) VALUE 0.
              :
       PROCEDURE  DIVISION.
         MOVE 1 TO DATA1.·····························1.
         CALL  'CBLDATADUMP'.·························2.
         IF  RETURN-CODE NOT = 0 THEN
             DISPLAY 'CBLDATADUMPが失敗しました。'
         END-IF.
              :
         MOVE 100 TO DATA1.···························3.
         CALL  'CBLDATADUMP'.·························4.
         IF  RETURN-CODE NOT = 0 THEN
             DISPLAY 'CBLDATADUMPが失敗しました。'
         END-IF.
              :
```

1. データ領域の内容を設定します。
2. CBLDATADUMP サービスルーチンを呼び出し，データ領域ダンプリストを出力します。
3. データ領域の内容を更新します。
4. CBLDATADUMP サービスルーチンを呼び出し，データ領域ダンプリストを出力します。2.で出力したデータ領域ダンプリストと比較し，各データが正しく更新できていることを確認します。

29.7　変換・転記・演算

29.7.1　CBLNCNV

CBLNCNV サービスルーチンは，英字項目・英数字項目・英数字編集項目・数字編集項目を日本語項目・日本語編集項目に変換するものです。

形式

```
CALL ’CBLNCNV’ USING 引数1 引数2 引数3
```

引数

- 引数 1 には，次に示すインタフェース領域の名前を指定します。

表 29-3　CBLNCNV サービスルーチンのインタフェース領域

記述形式	内容
01 データ名1.	CALL 文の USING で指定するインタフェース領域の名前を指定する。
02 データ名2.	転記インジケータの名前を指定する。
03 データ名3 PIC X(01).	ALL 指定※の有無を指定する。 0：ALL 指定なし。 1：ALL 指定あり。
03 データ名4 PIC X(01).	送り出し側作用対象のデータ属性を指定する。 0：英字，英数字，英数字編集，数字編集項目 1：数字項目
03 データ名5 PIC X(01).	受け取り側作用対象のデータ属性を指定する。 0：日本語項目 1：日本語編集項目
03 FILLER　PIC X(01).	予備。値は 0 でなければならない。このシステムではこの領域は使用しない。
02 データ名6 PIC 9(8) USAGE COMP.	送り出し側作用対象の項目のけた数を指定する。
02 データ名7 PIC 9(8) USAGE COMP.	受け取り側作用対象の項目のけた数を指定する。

注※

ALL 指定は，COBOL の言語仕様の ALL 定数と同じように扱われます。ALL 指定がある場合，変換後のデータは，変換前のデータのデータ名 6 で指定したけた数の何回かの繰り返しによって，データ名 7 で指定したけた数まで生成されます。ALL 指定がない場合は，データ名 6 で指定したけた数しか変換しません。

- 引数 2 には，変換前のデータを指定します。
- 引数 3 には，変換後のデータが格納されます。

戻り値

0：正常終了した場合

-1：パラメタエラーが発生した場合

-2：コード変換のための環境が整っていない場合（Unicode 機能を使用する場合）

-3：コード変換ライブラリでエラーが発生した場合（Unicode 機能を使用する場合）

変換規則（シフト JIS を使用する場合）

このサービスルーチンでの変換規則を次に示します。

- 次の半角文字はそのまま同じ全角文字に変換します。

  ```
  0～9  A～Z  a～z  !  ″  #  $  %  &  ’  (  )  *  +  ,  -  .  /  :
  ;  <  =  >  ?  [  ¥  ]  ^  _  @  {  |  }  。 「  」 、 ・  ゛  ゜  ‐
  ｱ～ﾝ  ｧ～ｫ  ｬ～ｮ  ｯ
  ```

- ￣（半角オーバライン）は全角の「～」に変換します。
- '（アポストロフィ）は全角の「’」（X'8166'）に変換します。
- "（引用符）は全角の「"」（X'8168'）に変換します。
- 次の文字は半角の空白 2 個（X'2020'）に変換します。
  `（アクサングラーブ，X'60'） X'01'～X'0F'
  X'10'～X'1F' X'80' X'A0' X'FD' X'FE'
- X'00'は，X'0000'に変換します。
- X'7F'および X'FF は，X'FFFF'に変換します。
- 半角の空白が偶数個のときは変換しません。
  半角の空白が奇数個のときは，受け取り側作用対象が日本語項目か日本語編集項目かによって次のように異なります。
  ・受け取り側作用対象が日本語項目のときは，最後の 1 個を全角の空白（X'8140'）に変換します。
  ・受け取り側作用対象が日本語編集項目のときは，最後に半角の空白（X'20'）を追加します。
- 次の文字は変換しません。
  全角文字 X'81'～X'8F' X'90'～X'9F' X'E0'～X'EF'
  X'F0'～X'FC'
  このとき，受け取り側作用対象の項目の後ろの空き領域には半角の空白（X'20'）を埋めます。

一般規則（シフト JIS を使用する場合）

- 規則に反した引数を指定した場合の結果は保証しません。

使用例（シフト JIS を使用する場合）

半角の"AAAA"を全角の"ＡＡＡＡ"に変換する場合の使用方法を次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
             :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 EISUU PIC X(4) VALUE 'AAAA'.
       01 NIHON PIC N(4).
       01 CHECK-PARM.
         02 CHECK-IND.
           03 CHECK-ALL PIC X(01).
           03 CHECK-TYP PIC X(01).
           03 CHECK-UKE PIC X(01).
           03 FILLER    PIC X(01)  VALUE  '0'.
         02  CHECK-LNG1 PIC 9(08) USAGE COMP.
         02  CHECK-LNG2 PIC 9(08) USAGE COMP.
             :
       PROCEDURE DIVISION.
             :
           MOVE '0' TO CHECK-ALL CHECK-TYP CHECK-UKE.
           MOVE  4  TO CHECK-LNG1.
           MOVE  4  TO CHECK-LNG2.
           CALL 'CBLNCNV' USING CHECK-PARM EISUU NIHON.
           IF RETURN-CODE NOT = 0 THEN
               CBLNCNVエラー処理
           END-IF.
             :
```

変換規則（EUC を使用する場合）

このサービスルーチンでの変換規則を次に示します。

- 次の半角文字はそのまま同じ全角文字に変換します。

  0～9 A～Z ! ″ # $ % & ' ( ) * + , - . / : ;
  < = > ? [¥] ^ _ @ { | }

- 次の半角文字は，対応する全角文字に変換されます。

  。「 」、・ ゛ ゜ - ｱ～ﾝ ｧ～ｫ ｬ～ｮ ｯ（X'8EA1'～X'8EDF'）

  ただし，X'8E'で始まる 2 バイトのエリアで，2 バイト目が X'A1'～X'DF'以外の場合は，半角の空白 2 個（X'2020'）に変換します。

- ‾（半角オーバライン）は全角の「～」に変換します。
- '（アポストロフィ）は全角の「’」（X'A1C7'）に変換します。
- "（引用符）は全角の「"」（X'A1C9'）に変換します。
- ,（X'8EA4'）は全角の「､」（X'A1A2'）に変換します。
- 次の文字は X'2020'（半角の空白 2 個）に変換します。
  `（アクサングラーブ，X'60'） X'01'～X'0F'
  X'10'～X'1F' X'80'～X'8D' X'90'～X'9F' X'A0'
- X'00'は，X'0000'に変換します。
- X'7F'および X'FF は，X'FFFF'に変換します。

- 半角の空白が偶数個のときは変換しません。

  半角の空白が奇数個のときは，受け取り側作用対象が日本語項目か日本語編集項目かによって次のように異なります。

  ・受け取り側作用対象が日本語項目のときは，最後の 1 個を全角の空白（X'A1A1'）に変換します。

  ・受け取り側作用対象が日本語編集項目のときは，最後に半角の空白（X'20'）を追加します。
- 日本語 EUC の全角文字は変換しません。
- 制御文字 X'8F'は変換しません。変換後のデータの後ろの空き領域には半角の空白（X'20'）を埋めます。

一般規則（EUC を使用する場合）

- 規則に反した引数を指定した場合の結果は保証しません。
- 日本語以外の EUC 環境では結果が保証しません。

使用例（EUC を使用する場合）

半角の"AAAA"を全角の"ＡＡＡＡ"に変換する場合の使用方法を次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
             :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 EISUU PIC X(8) VALUE 'AAAA'.
       01 NIHON PIC N(4).
       01 CHECK-PARM.
         02 CHECK-IND.
           03 CHECK-ALL PIC X(01).
           03 CHECK-TYP PIC X(01).
           03 CHECK-UKE PIC X(01).
           03 FILLER    PIC X(01)  VALUE  '0'.
         02  CHECK-LNG1 PIC 9(08) USAGE COMP.
         02  CHECK-LNG2 PIC 9(08) USAGE COMP.
             :
       PROCEDURE DIVISION.
             :
           MOVE '0' TO CHECK-ALL CHECK-TYP CHECK-UKE.
           MOVE  8  TO CHECK-LNG1.
           MOVE  4  TO CHECK-LNG2.
           CALL 'CBLNCNV' USING CHECK-PARM EISUU NIHON.
           IF RETURN-CODE NOT = 0 THEN
               CBLNCNVエラー処理
           END-IF.
             :
```

変換規則（Unicode 機能を使用する場合）

Unicode 機能については，「27.　Unicode 機能」を参照してください。

このサービスルーチンでの変換規則を次に示します。

- 次の半角文字（UTF-8）は，該当する全角文字（UTF-16）に変換します。UTF-16 への変換は，実行時環境変数 CBLUNIENDIAN の設定に従い UTF-16LE，または UTF-16BE に変換します。

```
0～9  A～Z  a～z  !  ″  #  $  %  &  ’  (  )  *  +  ,  －  .  /  :
;  <  =  >  ?  [  ¥  ]  ^  _  @  {  |  }  。 「  」 、  ・  ゛  ゜  －
ｱ～ﾝ  ｧ～ｫ  ｬ～ｮ  ｯ
```

- ￣（半角オーバライン）は全角の「～」に変換します。
- '（アポストロフィ）は全角の「’」（X'2019'）に変換します。
- "（引用符）は全角の「”」（X'201D'）に変換します。
- 次の文字は半角空白 1 個※1 に変換します。
  `（アクサングラーブ，X'60'） X'01'～X'0F' X'10'～X'1F'
- X'00'は X'0000'に変換します。
- X'7F'は X'FFFF'に変換します。
- 半角の空白（X'20'）が偶数個のときは全角変換※1 しません。
  半角空白が奇数個のときは，受け取り側作用対象が日本語項目か日本語編集項目かによって次のように異なります。
  ・受け取り側作用対象が日本語項目のときは，最後の 1 個を全角の空白※2 に変換します。
  ・受け取り側作用対象が日本語編集項目のときは，最後に半角の空白※1 を追加します。
- 全角文字（UTF-8）は，対応する全角文字（UTF-16）へ変換します。

注※1

半角空白の文字コードは，UTF-16LE のときは X'2000'，UTF-16BE のときは X'0020'となります。

注※2

全角空白の文字コードは，UTF-16LE のときは X'0030'，UTF-16BE のときは X'3000'となります。

一般規則（Unicode 機能を使用する場合）

- 規則に反した引数を指定した場合の結果は保証しません。

使用例（Unicode 機能を使用する場合）

半角の" ｱｱｱｱ "を全角の"ＡＡアア"に変換する場合の使用方法を次に示します。

用途が DISPLAY の項目の領域は，文字コードを意識して設定してください。シフト JIS と UTF-8 では必要とする領域のサイズが異なります。半角かたかなの場合，シフト JIS では 1 文字が 1 バイトですが，UTF-8 では 1 文字が 3 バイトとなります。

```
       IDENTIFICATION  DIVISION.
       PROGRAM-ID.  SAMPLE1.
                :
       DATA  DIVISION.
       WORKING-STORAGE  SECTION.
       01  EISUU  PIC  X(8)  VALUE  ’ ｱｱｱｱ ’.
       01  NIHON  PIC  N(4).
       01  CHECK-PARM.
         02  CHECK-IND.
           03  CHECK-ALL  PIC  X(01).
           03  CHECK-TYP  PIC  X(01).
           03  CHECK-UKE  PIC  X(01).
```

```
          03  FILLER       PIC  X(01)  VALUE  '0'.
        02  CHECK-LNG1     PIC  9(8)  USAGE  COMP.
        02  CHECK-LNG2     PIC  9(8)  USAGE  COMP.
              :
      PROCEDURE  DIVISION.
              :
          MOVE  '0'  TO  CHECK-ALL  CHECK-TYP  CHECK-UKE.
          MOVE   8   TO  CHECK-LNG1.
          MOVE   4   TO  CHECK-LNG2.
          CALL  'CBLNCNV'  USING  CHECK-PARM  EISUU  NIHON.
          IF  RETURN-CODE  NOT = 0  THEN
              *> CBLNCNVエラー処理
          END-IF.
               :
```

29.7.2 CBLUBIT

CBLUBIT サービスルーチンは，ビットデータを論理演算するものです。

形式

```
CALL 'CBLUBIT' USING 引数1 引数2 引数3 引数4
```

引数

- 引数 1 には，第一演算項のデータを 1 バイトの英数字項目で指定します。
- 引数 2 には，第二演算項のデータを 1 バイトの英数字項目で指定します。
- 引数 3 には，次の演算命令コードを指定します。

  A：AND（論理積）

  O：OR（論理和）

  X：XOR（排他的論理和）
- 引数 4 には，演算結果が格納されます。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

規則

引数 1～4 が 1 バイト以上の長さの場合，先頭の 1 バイトだけが有効となります。

使用例

X'00'と X'FF'の論理積を求める場合の使用例を次に示します。

```
      IDENTIFICATION DIVISION.
      PROGRAM-ID. SAMPLE1.
           :
      DATA DIVISION.
      WORKING-STORAGE  SECTION.
      01 OP-1 PIC X(01) VALUE X'00'.
```

```
01 OP-2 PIC X(01) VALUE X'FF'.
01 OP-3 PIC X(01) VALUE 'A'.
01 OP-4 PIC X(01).
    :
PROCEDURE DIVISION.
    :
    CALL 'CBLUBIT' USING OP-1 OP-2 OP-3 OP-4.
    IF RETURN-CODE NOT = 0 THEN
        CBLUBITエラー処理
    END-IF.
    :
```

29.7.3 CBLCNVERRORINFO

CBLCNVERRORINFO サービスルーチンは，直前に実行された組み込み関数の CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数のエラー情報を取得するものです。

形式

```
CALL 'CBLCNVERRORINFO' USING 引数1 引数2 引数3
```

引数

- 引数 1 には，4 バイトの符号なし 2 進項目を指定します。
- 引数 2 には，4 バイトの符号なし 2 進項目を指定します。
- 引数 3 には，4 バイトの符号付き 2 進項目を指定します。

戻り値

0：正常終了した場合

100：エラー情報がなかった場合

規則

- このサービスルーチンでエラー情報を取得するためには，-FunctionECSup,CodeConvErr オプションを指定して，組み込み関数の CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数が記述された COBOL ソースをコンパイルする必要があります。
- 引数 1 には，実行時エラーのメッセージ番号が格納されます。このサービスルーチンで取得できるのは，次の表に示す実行時メッセージに対応するエラー情報です。直前の組み込み関数の CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数で，これらの実行時メッセージが出力されていない場合，サービスルーチンはエラーの戻り値 100 を返します。

表 29-4　引数 1 に格納されるメッセージ番号

実行時メッセージ	引数 1 に格納される値
KCCC2313R-W	2313
KCCC2314R-W	2314
KCCC2315R-W	2315

実行時メッセージ	引数 1 に格納される値
KCCC2316R-W	2316

- 引数 2 には，組み込み関数の引数に指定されたデータのうち，コード変換失敗の要因となったデータの位置情報が格納されます。DISPLAY-OF 関数／NATIONAL-OF 関数で引数 1 が 2315 以外の場合，引数 2 には 0 が格納されます。
- DISPLAY-OF 関数／NATIONAL-OF 関数の場合，引数 3 には，コード変換失敗の要因となった日立コード変換の関数が返した戻り値が格納されます。戻り値の内容については，コード変換ライブラリのマニュアルの CodeConvString 関数に関する説明を参照してください。

  CONVERT-CODE 関数（Linux で有効）の場合，引数 3 には，コード変換失敗の要因となった iconv 関数が設定した errno の値が格納されます。errno については，システムのマニュアルのシステム関数に関する説明を参照してください。
- 戻り値が 0 以外の場合，引数 1，引数 2，および引数 3 の内容は変更されません。

注意事項

取得できるエラー情報は，直前に実行された組み込み関数の CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数のエラー情報です。コード変換が失敗する組み込み関数の CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数を実行したあとから，このサービスルーチンを呼び出す前までの間に，正常終了する組み込み関数の CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数を実行した場合，エラー情報は取得できません。

使用例

```
       IDENTIFICATION   DIVISION.
       PROGRAM-ID.      SAMPLE1.
       DATA             DIVISION.
       WORKING-STORAGE  SECTION.
       01 ALNUM-DATA    PIC X(10).
       01 NATIONAL-DATA PIC N(10).
      *エラー情報取得用
       01 ERR-NUMBER  PIC 9(9)  USAGE COMP.
       01 ERR-OFFSET  PIC 9(9)  USAGE COMP.
       01 ERR-RETCODE PIC S9(9) USAGE COMP.
       01 戻り値      PIC S9(9) USAGE COMP.
       PROCEDURE DIVISION.
             :
           *>NATIONAL-OF関数で呼び出し
           MOVE FUNCTION NATIONAL-OF(ALNUM-DATA) TO NATIONAL-DATA.
           *>NATIONAL-OF関数呼び出し直後にサービスルーチン呼び出し
           CALL 'CBLCNVERRORINFO' USING ERR-NUMBER
                                        ERR-OFFSET
                                        ERR-RETCODE.
             :
           MOVE RETURN-CODE TO 戻り値.
             :
           IF 戻り値 = 0
             THEN
               *> NATIONAL-OF関数で発生したエラーに対する処理
                :
```

```
           ELSE
              *> NATIONAL-OF関数で正常実行後の処理
           END-IF.
```

29.8.1　CBLSQLERROR（Linux で有効）

CBLSQLERROR サービスルーチンは，埋め込み SQL 文による ODBC インタフェース機能の使用時に，出力された実行時メッセージのエラー情報（COBOL メッセージ番号，SQLSTATE，SQL エラーコード）を取得します。

形式

```
CALL ’CBLSQLERROR’ USING 引数1
```

引数

- 引数 1 は，取得する実行時エラーメッセージのエラー情報を指定します。この項目は次の形式の集団項目とします。

表 29-5　CBLSQLERROR サービスルーチンのインタフェース領域

記述形式	内容
01 データ名01.	CALL 文の USING で指定するエラー情報取得領域の名前を指定する。
02 データ名02 PIC S9(9) USAGE COMP.	COBOL メッセージ番号
02 FILLER　PIC X(7).	予備。このサービスルーチンを呼び出す前に LOW-VALUE(X'00')を設定しなければならない。
02 データ名03 PIC X(5).	SQLSTATE
02 データ名04 PIC S9(9) USAGE COMP.	SQL エラーコード
02 FILLER　PIC X(516).	予備。このサービスルーチンを呼び出す前に LOW-VALUE(X'00')を設定しなければならない。

戻り値

0：エラー情報が取得できた場合

100：エラー情報がない場合

-1：エラー情報の取得に失敗した場合

規則

- CBLSQLERROR サービスルーチンを呼び出すと，直前に実行された埋め込み SQL 文で出力されたエラーメッセージに対するエラー情報（COBOL メッセージ番号，SQLSTATE，SQL エラーコード）が引数 1 に設定されます。また，SQL 文実行時のエラーメッセージ出力抑止の環境変数を指定した場合でも，抑止されたメッセージのエラー情報が設定されます。引数 1 に設定されるエラー情報を次に示します。

表 29-6　引数 1 に設定されるエラー情報の説明

エラー情報	説明
COBOL メッセージ番号	直前に実行された埋め込み SQL 文で出力された KCCC8000R-S から KCCC8026R-S のメッセージ番号が設定されます。メッセージについては，マニュアル「COBOL2002　メッセージ」を参照してください。
SQLSTATE	ODBC ドライバマネージャが返す診断コードが設定されます。詳細については，使用している ODBC ドライバまたは DBMS のマニュアルを参照してください。
SQL エラーコード	データソース固有のネイティブエラーコードが設定されます。詳細については，使用している ODBC ドライバまたは DBMS のマニュアルを参照してください。

- CBLSQLERROR サービスルーチンの戻り値は，RETURN-CODE 特殊レジスタで参照できます。
- SQLSTATE および SQL エラーコードは，実行時メッセージ KCCC8002R-S または KCCC8007R-W が出力された場合に設定されます。それ以外の実行時メッセージの場合は，SQLSTATE には空白，SQL エラーコードには 0 が設定されます。
- 直前に実行された埋め込み SQL 文で複数のエラーメッセージが出力された場合，CBLSQLERROR サービスルーチンの戻り値が 100 になるまで繰り返し呼び出すことで，すべてのエラー情報を取得できます。CBLSQLERROR サービスルーチンの戻り値が 100 の場合，COBOL メッセージ番号，SQL エラーコードは 0，SQLSTATE には空白が設定されます。
- 埋め込み SQL 文を実行しないで CBLSQLERROR サービスルーチンを実行すると，戻り値は 100 を返します。
- CBLSQLERROR サービスルーチンでエラーが発生し，エラー情報の取得に失敗した場合，戻り値は-1 を返します。
- 暗黙的に実行される SQL 文（DISCONNECT 文での暗黙的な COMMIT 文など）でエラーが発生した場合でも，CBLSQLERROR サービスルーチンはエラー情報を取得します。暗黙的に実行される SQL 文は，マニュアル「COBOL2002　言語　拡張仕様編」「9.　データベースアクセス機能」の各 SQL 文の一般規則を参照してください。

注意事項

引数の指定，および形式が「表 29-5　CBLSQLERROR サービスルーチンのインタフェース領域」と異なる場合の動作は保証しません。

使用例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
                :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
                :
       01  OP-1.
         02 ERR-MSGNUMBER     PIC S9(9) USAGE COMP.
         02 FILLER            PIC X(7)   VALUE LOW-VALUE.
         02 ERR-SQLSTATE      PIC X(5).
         02 ERR-SQLERRORCODE  PIC S9(9) USAGE COMP.
         02 FILLER            PIC X(516) VALUE LOW-VALUE.
```

```
 01  STMT-PROC          PIC X(20).
 01  戻り値             PIC S9(9) USAGE COMP.
          :
 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01 ODBC-DSN            PIC X(10) VALUE 'SAMPLE'.
 01 ODBC-UID            PIC X(10) VALUE 'USER1'.
 01 ODBC-PWD            PIC X(10) VALUE 'USER1'.
 EXEC SQL END DECLARE SECTION END-EXEC.
      :
 PROCEDURE DIVISION.
          :
*>SQL文を実行できたかどうかを<埋め込み例外宣言>にて判断します。
     EXEC SQL
          WHENEVER SQLERROR PERFORM :エラー処理  … 1.
     END-EXEC.
          :
*>データソースに接続する。
     MOVE 'データソースへの接続' TO STMT-PROC.  … 2.
     EXEC SQL
          CONNECT :ODBC-UID IDENTIFIED BY :ODBC-PWD
              USING :ODBC-DSN
     END-EXEC.
          :
*>表を参照する。
     MOVE '表のデータ参照' TO STMT-PROC.  … 2.
     EXEC SQL
          SELECT  … 3.
     END-EXEC.
          :
*>データソースから切断する。
     MOVE 'データソースから切断' TO STMT-PROC.  … 2.
     EXEC SQL
          DISCONNECT
     END-EXEC.
     STOP RUN.
          :
 エラー処理.     … 4.
     PERFORM WITH TEST AFTER UNTIL 戻り値 NOT = ZERO
       CALL 'CBLSQLERROR' USING OP-1
       MOVE RETURN-CODE TO 戻り値
       EVALUATE 戻り値
         WHEN  0
           DISPLAY STMT-PROC … 5.
*>              << エラー情報判定，リカバリ処理 >>
                        :
         WHEN  -1
           DISPLAY 'CBLSQLERRORが失敗しました。'
       END-EVALUATE
     END-PERFORM.
          :
```

上記の例では，3.の埋め込み SQL 文実行でエラーが発生した場合に，1.の埋め込み例外宣言の指定によって，4.のエラー処理段落が実行されます。ここで CBLSQLERROR サービスルーチンを呼び出し，エラー情報を取得します。2.のように埋め込み SQL 文実行ごとに，実行状態を退避しておくと，5.で，どの実行状態でエラーが発生したかがわかります。

29.9　XMAP3 を使用した画面・帳票関連

XMAP3 を使用した書式印刷機能，XMAP3 を使用した通信節による画面操作・帳票出力機能に関するサービスルーチンです。

29.9.1　CBLXMAPERROR（AIX(32)で有効）

CBLXMAPERROR サービスルーチンは，XMAP3 を使用した書式印刷機能，XMAP3 を使用した通信節による画面操作・帳票出力機能で，出力される実行時メッセージのエラー情報（COBOL メッセージ番号，エラーコード）を取得します。

形式

```
CALL　'CBLXMAPERROR'　USING　引数1
```

引数

引数 1 は，次に示す形式の集団項目です。

表 29-7　CBLXMAPERROR サービスルーチンの集団項目

記述形式	内容
01 データ名01.	CALL 文の USING で指定するエラー情報取得領域の名前を指定します。
02 データ名02　PIC S9(9) USAGE COMP.	COBOL メッセージ番号です。
02 データ名03　PIC 9(9)　USAGE COMP.	エラーコードです。
02 FILLER　　PIC X(528).	予備。このサービスルーチンを呼び出す前に LOW-VALUE(X'00') を設定しておきます。

注

引数の指定，および形式が異なる場合の動作は保証しません。

戻り値

0：エラー情報が取得できた場合

100：エラー情報がない場合

-1：エラー情報の取得に失敗した場合

規則

- CBLXMAPERROR サービスルーチンを呼び出すと，直前に実行された XMAP3 を使用した書式印刷機能の入出力文，または XMAP3 を使用した通信節による画面操作・帳票出力機能の通信文で出力されるエラーメッセージに対するエラー情報（COBOL メッセージ番号，エラーコード）が引数 1 に設定されます。また，XMAP3 を使用した書式印刷機能で FILE STATUS 句を指定した場合でも，該当する実行時メッセージのエラー情報が設定されます。

引数 1 に設定されるエラー情報を次に示します。

表 29-8　引数 1 に設定されるエラー情報の説明

エラー情報	説明
COBOL メッセージ番号	直前に実行された XMAP3 を使用した書式印刷機能の入出力文，または XMAP3 を使用した通信節による画面操作・帳票出力機能の通信文で出力された KCCC3401R-S から KCCC3418R-S，または KCCC5000R-W から KCCC5051R-W までの S レベル以下のメッセージ番号が設定されます。メッセージについては，マニュアル「COBOL2002　メッセージ」を参照してください。
エラーコード	XMAP3 が返したエラー詳細コードです。XMAP3 のエラー詳細コードについては，XMAP3 のマニュアル「画面・帳票サポートシステム XMAP3 開発・実行ガイド」またはマニュアル「XMAP3 Version 5 画面・帳票サポートシステム XMAP3 実行ガイド」のリターンコードの説明を参照してください。

- CBLXMAPERROR サービスルーチンの戻り値は，RETURN-CODE 特殊レジスタで参照できます。
- エラーコードは，実行時メッセージ KCCC3416R-S，KCCC3417R-S，KCCC5007R-S，KCCC5008R-S，または KCCC5040R-S～KCCC5051R-W が出力された場合に設定されます。それ以外の実行時メッセージの場合は，エラーコードには 0 が設定されます。
- 次に示す場合，CBLXMAPERROR サービスルーチンの戻り値は 100 を返します。CBLXMAPERROR サービスルーチンの戻り値が 100 の場合，COBOL メッセージ番号，エラーコードは 0 が設定されます。

  ・直前に実行された入出力文，通信文でエラーが発生していない場合

  ・入出力文，通信文を実行しないで CBLXMAPERROR サービスルーチンを呼び出した場合

  ・CBLXMAPERROR サービスルーチンを呼び出したあとに入出力文，通信文を実行しないで，再度，CBLXMAPERROR サービスルーチンを呼び出した場合
- CBLXMAPERROR サービスルーチンでエラーが発生し，エラー情報の取得に失敗した場合，戻り値は-1 を返します。

使用例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
                :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
                :
       01  OP-1.
         02 ERR-MSGNUMBER     PIC S9(9) USAGE COMP.
         02 ERR-XMAPERRORCODE PIC 9(9)  USAGE COMP.
         02 FILLER            PIC X(528) VALUE LOW-VALUE.
       01  STMT-PROC          PIC X(20).
       01  戻り値             PIC S9(9) USAGE COMP.
                :
           COPY MDC1PC0.
                :
       COMMUNICATION  SECTION.
       CD  PRINT-0  FOR  OUTPUT  WS
           MAP  NAME  IS  PRINT-MAP
```

```
       MAPPING  MODE  IS  PRINT-MAPMODE
       STATUS  KEY  IS  PRINT-KEY
       DATA  ABSENCE  CODE  IS  PRINT-ABSENCE
       SYMBOLIC  TERMINAL  IS  PRINT-TERMINAL.
            :
PROCEDURE DIVISION.
            :
       MOVE ’PRT001へMDC1PC6G帳票を送信’ TO STMT-PROC.   … 2.
       MOVE  ’MDC1PC6G’ TO  PRINT-MAP.
       MOVE  ’PRT001’  TO  PRINT-TERMINAL.
       SEND  PRINT-O  FROM  MDC1PCO  WITH  EMI.          … 3.
       IF PRINT-KEY NOT = ’00000’
         PERFORM エラー処理                              … 1.
            :
       STOP RUN.
           :
  エラー処理.                                            … 4.
         CALL ’CBLXMAPERROR’ USING OP-1
         MOVE RETURN-CODE TO 戻り値
         EVALUATE 戻り値
           WHEN  0
             DISPLAY STMT-PROC                          … 5.
 *>              << エラー情報判定，リカバリ処理 >>
                          :
           WHEN  -1
             DISPLAY ’CBLXMAPERRORが失敗しました。’
         END-EVALUATE.
            :
```

この例では，3.の SEND 文実行でエラーが発生した場合に，1.の PERFORM 文から，4.のエラー処理段落が実行されます。ここで CBLXMAPERROR サービスルーチンを呼び出し，エラー情報を取得します。2.のように通信文実行ごとに実行状態を退避しておくと，5.で，どの実行状態でエラーが発生したかがわかります。

30 プログラミング上の留意点

この章では，処理速度が速く，移植性の良いプログラムを作成する上での留意点について説明します。

30.1　処理速度の速いプログラムの作成

処理速度の速いプログラムを作成するためのチェック項目と，その理由などについて説明します。

30.1.1　チェック項目一覧

項番	分類	チェック項目
1	演算，転記，比較	数字項目の定義や演算を工夫できないか。
2		添字は 4 バイトの符号付き 2 進項目で定義しているか。
3		添字は定数で指定できないか。
4		比較する項目の長さは同じか。
5		2 進項目，指標データ項目，浮動小数点項目の境界は適切な位置にあるか。
6		USAGE 句項目の指定や，小数点以下のけた数は一致しているか。
7		データ転送にむだはないか。
8		不要な小数点はないか。
9		可変長項目のあとに固定長項目を定義していないか。
10	ファイル処理	READ INTO，WRITE FROM を使っていないか。
11	領域	作業場所節および局所場所節で，一緒に使う項目は互いに近くで定義しているか。

30.1.2　チェック項目の説明

(1)　数字項目の定義や演算を工夫できないか

データ項目の属性変換が発生しないようにしたり，算術演算子を少なくしたりして中間結果のけた数を最小にすると処理速度が向上します。そのために留意する点を次に示します。

(a)　数字項目のけた拡張機能を使用しない場合

- 数字項目を定義する場合は，SYNC 句指定のある符号付き 2 進項目（COMP）とし，また，けた数を必要以上に大きくしないようにします。
- 10 進項目は演算に使わないようにします。例えば，入出力レコード中の 10 進のデータ項目を演算で頻繁に使うときは，データ項目を 2 進項目の作業領域に移してから演算するようにします。
- 属性の異なるもの同士の演算は避けます。例えば，2 進項目と内部 10 進項目，2 進項目と外部 10 進項目などの演算を避けます。

- 複雑な COMPUTE 文は分割するなどし，一つの COMPUTE 文中の算術演算子を少なくします。COMPUTE 文中に多くの算術演算子を使うと，けたあふれも発生しやすくなります。
- 1 バイト 2 進項目を使用する処理はできるだけ局所化し，演算に使用する項目には 2 バイト 2 進項目を使用するようにします。

(b)　数字項目のけた拡張機能を使用する場合

- けた数が 19～38 けたの数字項目を定義する場合は，内部 10 進項目（COMP-3）とし，けた数を必要以上に大きくしないようにします。
- けた数が 19～38 けたの数字項目や数字定数を使用する処理は，ソースファイルごとにできるだけ局所化します。
- 属性の異なるもの同士の演算は避けます。例えば，2 進項目と内部 10 進項目，2 進項目と外部 10 進項目などの演算を避けます。
- 複雑な COMPUTE 文は分割するなどし，一つの COMPUTE 文中の算術演算子を少なくします。COMPUTE 文中に多くの算術演算子を使うと，けたあふれも発生しやすくなります。
- 1 バイト 2 進項目を使用する処理はできるだけ局所化し，演算に使用する項目には 2 バイト 2 進項目を使用するようにします。

数字項目のけた拡張機能については，「28.　数字項目のけた拡張機能（AIX(64)，Linux(x64)で有効）」を参照してください。

(2)　添字は 4 バイトの符号付き 2 進項目で定義しているか

添字は 4 バイトの符号付き 2 進項目として処理されます。それ以外の形式のときは，データ変換命令が生成されるため処理速度が遅くなります。

(3)　添字は定数で指定できないか

添字を定数にすると，アドレス計算のための命令が生成されないので，処理速度が向上します。

(悪い例)

```
          :
MOVE 3 TO J.
MOVE A TO B(J).
```

(良い例)

```
          :
MOVE A TO B(3).
```

(4)　比較する項目の長さは同じか

英数字項目を比較するときに項目の長さが異なると，コンパイラは長さの差の部分が空白かどうかを調べるオブジェクトコードを生成します。このため，項目の長さを同じにすると処理速度が向上します。

(5) 2進項目，指標データ項目，浮動小数点項目の境界は適切な位置にあるか

2進項目，指標データ項目，浮動小数点項目がそれぞれの適切な境界にないと，処理速度が遅くなります。境界については，マニュアル「COBOL2002 言語 標準仕様編」「4.4.1(7) 実行用コードの効率を高めるための項目のけた詰め」を参照してください。

境界を合わせるには，SYNC句を指定する方法と，利用者がバイト数を計算して合わせる方法とがあります。ただし，SYNC句を使うと処理速度は向上しますが，多用すると実行可能プログラムが必要以上に大きくなることがあるので注意してください。

バイト数を計算して境界を合わせる例を次に示します。

(例)

```
WORKING-STORAGE SECTION.
01 DATA-SGROUPS.
   *> 境界合わせが必要なものをまとめ,境界合わせを
   *> 計算しながら並べる
  02 A PIC S9(12) USAGE COMP.
  02 B PIC S9(7) USAGE COMP.
  02 C PIC S9(7) USAGE COMP.
  02 D USAGE COMP-1.
  02 E USAGE COMP-1.
  02 F USAGE COMP-2.
  02 G PIC S9(14) USAGE PACKED-DECIMAL.
  02 H PIC S9(10) USAGE COMP.
  02 I USAGE INDEX.
   *> 境界合わせが不要なものをまとめる
  02 J PIC S9(10) DISPLAY.
```

(6) USAGE句項目の指定や，小数点以下のけた数は一致しているか

USAGE句項目が一致していないと，データ変換命令が生成されるため処理速度が遅くなります。また，小数点以下のけた数が一致していないと，演算のときにけた数を合わせる命令が生成されるため処理速度が遅くなります。

(悪い例)

```
WORKING-STORAGE SECTION.
77 TOL PIC S9(3) USAGE COMP.
77 NUM PIC S9(3)V99 USAGE PACKED-DECIMAL.
77 AVE PIC S9(3)V9.
     :
PROCEDURE DIVISION.
     :
    COMPUTE AVE = TOL / NUM.
```

(良い例)

```
WORKING-STORAGE SECTION.
77 TOL PIC S9(3)V9 USAGE PACKED-DECIMAL.
77 NUM PIC S9(3)V9 USAGE PACKED-DECIMAL.
77 AVE PIC S9(3)V9 USAGE PACKED-DECIMAL.
```

```
        :
PROCEDURE DIVISION.
        :
        COMPUTE AVE = TOL / NUM.
```

(7) データ転送にむだはないか

入力レコードを作業領域に転送するとき，むだなデータを転送することがよくあります。転記する文は，項目の長さに比例して時間が掛かります。複数の MOVE 文に分けて実際に転送する長さを短くすると処理速度が向上します。

(悪い例 1)

```
01 A1.      *> 入力領域
  02 FILLER PIC X(300).
  02 B1     PIC X(5).
  02 FILLER PIC X(30).
  02 C1     PIC X(5).
01 A2.      *> 作業領域
  02 FILLER PIC X(300).
  02 B2     PIC X(5).
  02 FILLER PIC X(30).
  02 C2     PIC X(5).
        :
PROCEDURE DIVISION.
        :
    MOVE A1 TO A2.
```

(良い例 1)

```
01 A1.      *> 入力領域
  02 FILLER PIC X(300).
  02 B1     PIC X(5).
  02 FILLER PIC X(30).
  02 C1     PIC X(5).
01 A2.      *> 作業領域
  02 FILLER PIC X(300).
  02 B2     PIC X(5).
  02 FILLER PIC X(30).
  02 C2     PIC X(5).
    :
PROCEDURE DIVISION.
    :
    MOVE B1 TO B2.
    MOVE C1 TO C2.
```

(悪い例 2)

```
FILE SECTION.
FD AFILE RECORDING MODE V.
01 A1.
  02 L      PIC S9(5) USAGE COMP. *> 入力レコード長
  02 FILLER PIC X(4000).
     *> 最大レコードだけを定義している
    :
WORKING-STORAGE SECTION.
```

```
01 WK.
  02 詳細なレコード定義
     :
PROCEDURE DIVISION.
     :
     MOVE A1 TO WK.
```

(良い例 2)

```
FILE SECTION.
FD AFILE
RECORD VARYING IN SIZE
        TO 4000
        DEPENDING ON CNT.
01 A1 PIC X(4000).
        :
WORKING-STORAGE SECTION.
77 CNT PIC 9(4).
01 WK.
  02 詳細なレコード定義
        :
PROCEDURE DIVISION.
     :
     MOVE A1(1:CNT) TO WK(1:CNT).
```

(8) 不要な小数点はないか

小数点があると処理が複雑になることがあり，処理速度が遅くなります。

(9) 可変長項目のあとに固定長項目を定義していないか

可変長項目の長さは実行時に変化します。可変長項目のあとにある固定長項目を参照する文があると，コンパイラはアドレスづけのための特別なオブジェクトコードを生成します。このため，通常の固定長項目を参照するよりも処理速度が遅くなります。

また，可変長項目のあとにその可変長項目に従属する項目以外を記述することは，COBOL 規格で禁止されています。そのため，このような記述をする場合は，COBOL 規格外の記述として注意する必要があります。

(悪い例)

```
WORKING-STORAGE SECTION.
01 V-GROUP.
  02 C PIC X
     OCCURS …… DEPENDING  ON I.
  02 A PIC S9(5) USAGE COMP.
  02 B PIC 9(6).
        :
```

(良い例)

```
WORKING-STORAGE SECTION.
01 J-GROUP.
```

```
02 A PIC S9(5) USAGE COMP.
02 B PIC 9(6).
02 C PIC X
  OCCURS …… DEPENDING ON I.
      :
```

(10) READ INTO，WRITE FROM を使っていないか

READ INTO，WRITE FROM を使うと，レコードの入出力処理でプログラム内のレコード領域との間でデータ転送が発生し，処理速度が遅くなります。

(11) 作業場所節および局所場所節で，一緒に使う項目は互いに近くで定義しているか

このシステムでは，作業場所節および局所場所節で定義した項目は，定義した場所に確保されます。このため，一緒に使用する項目は近くに定義して参照を局所化すると処理速度が向上します。

30.2　移植性の良いプログラムの作成

移植性の良いプログラムを作成するためのチェック項目と，その理由などについて説明します。

30.2.1　チェック項目一覧

項番	分類	チェック項目
1	定数	半角かたかな文字は使わないようにできないか。
2		16 進英数字定数は使わないようにできないか。
3		日本語文字データの転記，比較に英数字定数を使わないようにできないか。
4		日本語定数の中に空白を使わないようにしているか。
5	プログラム名	プログラム名には英小文字を使わないようにできないか。
6	文字比較	異なった文字集合間での順序の比較はしていないか。
7	内部表現	数値の内部表現を意識したコーディングをしていないか。
8	入出力	9x の入出力状態は利用しないようにできないか。
9	正書法	固定形式正書法の場合，日本語を含む原始プログラムの 1 行を，プログラム原文領域の 2/3 程度までで終わらせているか。
10	COPY 文 REPLACE 文	COPY 文の REPLACING 指定および REPLACE 文の作用対象に日本語を使わないようにできないか。
11	C 言語との連携	CALL 文で直接システム関数を呼び出していないか。
12	その他	システム固有の言語仕様を使わないようにできないか。

30.2.2　チェック項目の説明

(1)　半角かたかなは使わないようにできないか

コンパイラがサポートするコード系には，シフト JIS（SJIS）コードまたは日本語 EUC コード（UJIS）の２種類があり，どちらの種類のコード系を対象とするかは，システムによって異なります。

半角かたかな文字のバイト数が，シフト JIS コードと日本語 EUC コードでは次のように異なります。このため，半角かたかな文字は使わないで全角かたかなを使った方が移植性が良くなります。

(シフトJISコード)　A　PIC　X(4)　VALUE　'ｻﾖﾅﾗ'.
(日本語EUCコード)　A　PIC　X(8)　VALUE　'ｻﾖﾅﾗ'.

(悪い例)

```
A　PIC　X(4)　VALUE　'ｻﾖﾅﾗ'.
```

(良い例)

```
A　PIC　N(4)　VALUE　N'サヨナラ'.
```

(2)　16 進英数字定数は使わないようにできないか

内部コードはシステムによって異なります。16 進英数字定数を使うということは内部コードを意識しているということなので移植性が悪くなります。

(3)　日本語文字データの転記，比較に英数字定数を使わないようにできないか

システムによってはシフトコードを指定することで日本語文字データを区別しているものがあります。日本語定数を使わないで，英数字定数で日本語文字データを指定すると，英数字定数の長さが異なってきます。日本語文字データの転記，比較などは日本語定数を使った方が移植性が良くなります。

(4)　日本語定数の中に空白を含まないようにしているか

全角の空白を記述した場合，システムによってコード値が$(2020)_{16}$になったり，$(8140)_{16}$になったりします。このため，比較や転記などで使用する日本語定数の中に空白があると，生成されるオブジェクトコードが異なります。

(5)　プログラム名には英小文字を使わないようにできないか

システムによっては，プログラム名に英小文字を使えないものや英大文字と英小文字の等価規則が適用されないものがあります。このため，プログラム名段落や CALL 文のプログラム名に英小文字を使わない方が移植性が良くなります。

(6)　異なった文字集合間での順序の比較はしていないか

異なった文字集合（例えば，かたかな文字と英数字文字）の順序は，システムによって異なります。このため，異なった文字集合間の比較をしない方が移植性が良くなります。

(7)　数値の内部表現を意識したコーディングをしていないか

数値の表現方法は，符号表現を含めてシステムに依存します。このため，数値の内部表現を意識したコーディングを避けた方が移植性が良くなります。

(悪い例)

```
       01 X.
         02 A PIC 9(2) USAGE COMP-X.
         02 B PIC X.
```

```
01 FILLER REDEFINES X.
  02 AA PIC X.
  02 BB PIC 9(2) USAGE COMP-X.
     :
PROCEDURE DIVISION.
     :
     MOVE 65 TO A.
     IF AA = 'A' THEN
         :
```

(8) 9x の入出力状態は利用しないようにできないか

9x の入出力状態はシステムによって異なります。このため，比較などに使わない方が移植性が良くなります。

9x を使う必要があるときは，90 以上かどうかを問う比較にした方が，移植性が良くなります。

(悪い例)

```
     :
SELECT A-FILE...
        FILE STATUS IS RTN.
     :
PROCEDURE DIVISION.
     :
  IF RTN = '90'
```

(良い例)

```
     :
SELECT A-FILE...
        FILE STATUS IS RTN.
     :
PROCEDURE DIVISION.
     :
  IF RTN NOT < '90'
```

(9) 固定形式正書法の場合，日本語を含む原始プログラムの 1 行をプログラム原文領域の 2/3 程度で終わらせているか

システムによっては，シフトコードを指定することで日本語文字を区別しているものがあります。

日本語を含む 1 行でプログラム原文領域すべてを使った原始プログラムをほかのシステム（例えば，VOS3 など）に移植すると，シフトコードを付加してプログラム原文領域をはみ出てしまうおそれがあります。このため，日本語を含む原始プログラムは，シフトコードを意識して 1 行を終わらせた方が移植性が良くなります。

(10) COPY 文の REPLACING 指定および REPLACE 文の作用対象に日本語を使わないようにできないか

システムによっては，COPY 文の REPLACING 指定および REPLACE 文の作用対象に，日本語が指定できないものがあります。このため，COPY 文の REPLACING 指定および REPLACE 文の作用対象に日本語を使わない方が移植性が良くなります。

(11) CALL 文で直接システム関数を呼び出していないか

C 言語のシステム関数は，#define によって実際の関数名とは別の関数名で宣言され，マクロ展開後に置き換えられることがあります。COBOL では C 言語のマクロは展開しないため，COBOL プログラムから C 言語のシステム関数を CALL 文で直接呼び出すと，COBOL からマクロ展開前の関数名を呼び出し，予期しない不具合が発生するおそれがあります。

COBOL プログラムから C 言語のシステム関数を呼び出す場合は，直接呼び出さないで，システム関数を呼び出す C プログラムを作成し，この C プログラムを呼び出すようにしてください。

30.3　COBOL プログラムが使用するスタック領域

プログラムが実行時に異常終了する原因の一つに，スタックオーバフローがあります。スタックオーバフローを回避するには，プログラムが使用するスタック領域の量について事前に把握しておく必要があります。ここでは，COBOL プログラムがスタック領域に配置するデータの種類とサイズについて説明します。

30.3.1　スタック領域に配置されるデータ

COBOL プログラム実行時にスタック領域に配置されるデータを次に示します。

- 局所場所節に定義したデータ項目
- 連絡節に定義したデータ項目
- COBOL プログラムが内部的に使用するデータ

(1)　局所場所節に定義したデータ項目

局所場所節に定義したデータ項目はスタック領域に配置します。データの配置については，マニュアル「COBOL2002　言語　標準仕様編」「4.4.1　機種によらないデータ記述」を参照してください。局所場所節に非常にサイズの大きいデータを定義した場合，スタック領域が不足するおそれがありますので注意してください。

(2)　連絡節に定義したデータ項目

連絡節に定義したデータ項目はスタック領域に配置します。配置するサイズは，手続き部の USING／RETURNING 指定によって異なります。

(a)　BY REFERENCE 指定のデータ項目

- 参照渡しの場合

  配置するサイズはポインタサイズ（4 バイト）※1 となります。手続き部の USING に指定したデータの数だけポインタサイズを配置します。

- 内容渡しの場合

  配置するサイズは，定義したデータ項目のサイズ（4 バイト境界に切り上げ）※2 とポインタサイズ（4 バイト）※1 を合計した値となります。

  なお，サイズが非常に大きいデータを定義した場合は，スタック領域が不足するおそれがあるため注意してください。

注※1

AIX(64)，Linux(x64)の場合では 8 バイトとなります。

注※2

AIX(64)，Linux(x64)の場合は８バイト境界に切り上げます。

(b)　BY VALUE／RETURNING 指定のデータ項目

配置するサイズは，定義したデータ項目のサイズ（４バイト境界に切り上げ）※となります。スタック領域に定義したデータ項目の値をそのまま配置します。データの配置については，マニュアル「COBOL2002　言語　標準仕様編」「4.4.1　機種によらないデータ記述」を参照してください。

なお，サイズが非常に大きいデータを定義した場合，スタック領域が不足するおそれがあるため注意してください。

注※

AIX(64)，Linux(x64)の場合は８バイト境界に切り上げます。

(3)　COBOL プログラムが内部的に使用するデータ

COBOL プログラムで明示的に定義したデータ項目以外に，COBOL プログラムが内部的に使用するデータがあります。このデータはプログラムを実行するために必要なデータです。このデータは COBOL コンパイラが内部的に自動生成するものであり，またプログラムの内容に依存します。

30.3.2　プログラム実行時呼び出し関係に依存するスタック領域の消費量

プログラム実行時に必要なスタック領域のサイズは，実行時要素の呼び出し関係やその属性に依存します。実行時要素が必要とするスタック領域のサイズは，スタック領域に配置されるデータの合計サイズです。スタック領域に配置されるデータについては，「30.3.1　スタック領域に配置されるデータ」を参照してください。

(1)　実行時要素の呼び出し関係

プログラム実行時に必要なスタック領域のサイズは，実行時要素の呼び出し関係に依存します。例えば，ある実行時要素が別の実行時要素を呼び出し，呼び出された実行時要素がまた別の実行時要素を呼び出します。このように実行時要素の呼び出しが入れ子になっている場合，実行時に必要なスタック領域のサイズは，各実行時要素が必要とするスタック領域のサイズの合計となります。

(2)　再帰属性プログラム

再帰属性プログラムが実行時に必要とするスタック領域のサイズは，プログラムが再帰的に呼び出される回数に依存します。再帰属性プログラムが実行時に必要とするスタック領域のサイズを次に示します。

再帰属性プログラムが一つの実行時要素として必要とするスタック領域のサイズ×再帰的に呼び出される回数

30.3.3　スタック領域のサイズ変更方法

プログラムが使用できるスタック領域のサイズは変更できます。規定の状態ではスタックオーバフローが起こるプログラムであっても，スタック領域のサイズを変更すると正常に実行できます。ただし，この対処方法はプログラムが使用するスタック領域の最大値があらかじめ予想できる場合だけ使用できます。

システムの設定値を変更して使用できるスタック領域のサイズを変更します。変更方法についてはシステムのマニュアルを参照してください。

31 最適化機能

COBOL2002 には，プログラムの実行効率を上げるための最適化機能があります。最適化をするためには，最適化オプションを指定してコンパイルします。この章では，指定するオプションと最適化の内容について説明します。

31.1　最適化のレベル

31.1.1　最適化オプションの種類

最適化用のオプションには次に示す-Optimize,0，-Optimize,1，-Optimize,2，-Optimize,3 の 4 種類があり，省略時は-Optimize,1 が仮定されます。

(1)　-Optimize,0 オプション

最適化をしません。このため，コンパイル時間は最短になります。

(2)　-Optimize,1 オプション

文の中で閉じた次の最適化をします。

- 命令レベルでの最適化（ピープホールによる不要命令の削除など）

(3)　-Optimize,2 オプション

広域的な次の最適化をします。

- 命令レベルでの最適化（ピープホールによる不要命令の削除など）
- 不変式のループ外への移動
- コピー伝播
- 定数の畳み込み
- 共通式の削除
- 演算の強さの軽減
- そと PERFORM 文のインライン展開

(4)　-Optimize,3 オプション

-Optimize,2 オプションでの最適化に加えて，10 進項目を 2 進項目に変換します。

(5)　注意事項

- 最適化によって文または文の一部が移動または削除されると，実行時メッセージや異常終了時要約情報リストなどの行番号／欄の情報が正しく出力されないことがあります。

  正しい行番号／欄の情報を出力するには，コンパイル時に-Optimize,0 オプションを指定して最適化を抑止する必要があります。
- -Optimize,3 オプションを指定すると次のような副作用を伴うことがあります。

（例）

```
       01 A PIC S9(9) VALUE +123456789.
       01 B PIC S9(5).
       01 C PIC S9(9).
           :
       PROCEDURE DIVISION.
            :
           MOVE A TO B.
           MOVE B TO C.
           DISPLAY C.
```

-Optimize,3 指定以外のときはデータ項目 C に+56,789 が設定されますが，-Optimize,3 指定の場合，+123,456,789 が設定されます（-DigitsTrunc 指定ありの場合を除く）。

また，演算よりも 10 進項目への代入の方が多いプログラムなど，プログラムによっては-Optimize,3 オプションを指定すると実行速度がかえって低下する場合があります。

31.2　最適化の内容

31.2.1　そと PERFORM 文のインライン展開

そと PERFORM 文のインライン展開の規則について説明します。

なお，PERFORM 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.31 PERFORM 文」を参照してください。

(1)　対象となる節，段落

手続き部分で定義された節と段落であることが必要です。宣言節で定義された節，段落はインライン化の対象となりません。

(2)　THRU 指定のある PERFORM 文

THRU 指定のある PERFORM 文の場合，次のどちらかの条件を満たしていなければなりません。

- 手続き名 1 と手続き名 2 の両方が節である
- 手続き名 1 と手続き名 2 の両方が段落である

例 1 の PERFORM 文は，SECT-1 が節，PARG-2-1 が段落であるため，インライン化の対象となりません。

(例 1)

```
        :
PROCEDURE DIVISION.
        :
        PERFORM SECT-1 THRU PARG-2-1.
        SECT-1 SECTION.
        PARG-1-1.
        :
        SECT-2 SECTION.
        PARG-2-1.
        :
```

- 手続き名 1 と手続き名 2 が節の場合，手続き名 2 の節は手続き名 1 の節の直後に定義された節である

例 2 の PERFORM 文は，SECT-2 が SECT-1 の直後に定義されています。

(例 2)

```
            :
PROCEDURE DIVISION.
            :
        PERFORM SECT-1 THRU SECT-2.
        SECT-1 SECTION.
        PARG-1-1.
            :
```

```
SECT-2 SECTION.
PARG-2-1.
    :
```

- 手続き名 1 と手続き名 2 が段落の場合，手続き名 2 の段落は手続き名 1 の段落の直後に定義された段落である

例 3 の PERFORM 文は，PARG-1-3 が PARG-1-1 の直後に定義された段落ではないため，インライン化の対象となりません。

(例 3)

```
      :
PROCEDURE DIVISION.
      :
    PERFORM PARG-1-1 THRU PARG-1-3.
    SECT-1 SECTION.
    PARG-1-1.
      :
    PARG-1-2.
      :
    PARG-1-3.
      :
```

(3) 呼ばれる節と段落

呼ばれる節と段落が次の（a）（b）の条件を満たしていることが必要です。

(a) THRU 指定がある場合

- 手続き名 1 と手続き名 2 が節の場合，節または節に従属する段落を分岐文や複数の PERFORM 文で参照してはなりません。
- 手続き名 1 と手続き名 2 が段落の場合，段落を分岐文や複数の PERFORM 文で参照してはなりません。

例 1 の PERFORM 文は，PARG-2-1 が GO TO 文によって参照されているため，インライン化の対象となりません。

(例 1)

```
      :
PROCEDURE DIVISION.
      :
    PERFORM SECT-1 THRU SECT-2.
      :
    GO TO PARG-2-1.
      :
    SECT-1 SECTION.
    PARG-1-1.
      :
    SECT-2 SECTION.
    PARG-2-1.
      :
```

例 2 の（1）の PERFORM 文は，PARG-1-2 と PARG-1-3 が（2）の PERFORM 文と（3）の GO TO 文によって参照されているため，インライン化の対象となりません。

（例 2）

```
      :
PROCEDURE DIVISION.
      :
    PERFORM PARG-1-2 THRU PARG-1-3. …(1)
      :
    PERFORM PARG-1-1 THRU PARG-1-4. …(2)
      :
    GO TO SECT-1. …(3)
      :
    SECT-1 SECTION.
    PARG-1-1.
      :
    PARG-1-2.
      :
    PARG-1-3.
      :
    PARG-1-4.
      :
```

(b) THRU 指定がない場合

- 手続き名 1 が節の場合，節または節に従属する段落を分岐文や複数の PERFORM 文で参照してはなりません。
- 手続き名 1 が段落の場合，段落を分岐文や複数の PERFORM 文で参照してはなりません。

例 1 の PERFORM 文は，PARG-1-2 が GO TO 文によって参照されているため，インライン化の対象となりません。

（例 1）

```
      :
PROCEDURE DIVISION.
      :
    PERFORM SECT-1.
      :
    GO TO PARG-1-2.
      :
    SECT-1 SECTION.
    PARG-1-1.
      :
    PARG-1-2.
      :
    SECT-2 SECTION.
      :
```

例 2 の（1）の PERFORM 文は，PARG-1-2 が（2）の PERFORM 文と（3）の GO TO 文によって参照されているため，インライン化の対象となりません。

(例 2)

```
      :
PROCEDURE DIVISION.
      :
    PERFORM PARG-1-2. …(1)
      :
    PERFORM PARG-1-1 THRU PARG-1-3. …(2)
      :
    GO TO SECT-1. …(3)
      :
    SECT-1 SECTION.
    PARG-1-1.
      :
    PARG-1-2.
      :
    PARG-1-3.
      :
    PARG-1-4.
      :
```

(4) PERFORM 文の位置

PERFORM 文が記述されている位置が次の(a)(b)の条件を満たしていることが必要です。

(a) THRU 指定がある場合

- 手続き名 1 と手続き名 2 が節の場合，節または節に従属する段落の中に呼び出し元の PERFORM 文があってはなりません。
- 手続き名 1 と手続き名 2 が段落の場合，段落の中に呼び出し元の PERFORM 文があってはなりません。

例 1 の PERFORM 文は，SECT-2 の中に書かれているため，インライン化の対象となりません。

(例 1)

```
      :
PROCEDURE DIVISION.
      :
    SECT-1 SECTION.
      :
    SECT-2 SECTION.
    PARG-2-1.
      :
    PERFORM SECT-1 THRU SECT-2.
      :
    SECT-3 SECTION.
      :
```

例 2 の PERFORM 文は，PARG-1-2 中に書かれているため，インライン化の対象となりません。

(例 2)

```
      :
PROCEDURE DIVISION.
      :
    SECT-1 SECTION.
    PARG-1-1.
      :
    PARG-1-2.
      :
    PERFORM PARG-1-1 THRU PARG-1-2.
      :
    PARG-1-3.
      :
```

(b) THRU 指定がない場合

- 手続き名 1 が節の場合，節または節に従属する段落の中に呼び出し元の PERFORM 文があってはなりません。
- 手続き名 1 が段落の場合，段落の中に呼び出し元の PERFORM 文があってはなりません。

例 1 の PERFORM 文は SECT-1 中に書かれているため，インライン化の対象となりません。

(例 1)

```
      :
PROCEDURE DIVISION.
      :
    SECT-1 SECTION.
    PARG-1-1.
      :
    PERFORM SECT-1.
      :
    SECT-2 SECTION.
      :
```

例 2 の PERFORM 文は PARG-1-1 中に書かれているため，インライン化の対象となりません。

(例 2)

```
      :
PROCEDURE DIVISION.
      :
    SECT-1 SECTION.
    PARG-1-1.
      :
    PERFORM PARG-1-1.
      :
    SECT-2 SECTION.
      :
```

(5)　呼ばれる節と段落までの間にある文

呼び出し元の PERFORM 文または手続き部先頭から（「(7)　注意事項」を参照），呼ばれる節と段落までの間にある文が次の二つの条件を満たしていることが必要です。

1. 最上位レベルの STOP RUN 文（「(7)　注意事項」を参照）など，プログラムの実行を終了する文がなければなりません。

   例 1 では，STOP RUN 文が文のレベルの最上位でないため，インライン化の対象とはなりません。

   （例 1）

```
        :
PROCEDURE DIVISION.
        :
    PERFORM SECT-1.
        :
    IF FLAG = 'ON' THEN
    STOP  RUN.
        :
    SECT-1 SECTION.
        :
```

2. 文のレベルに関係なく，ENTRY 文があってはなりません。

   例 1 では ENTRY 文があるため，インライン化の対象となりません。

   （例 1）

```
        :
PROCEDURE DIVISION.
        :
    PERFORM SECT-1.
        :
    IF FLAG = 'ON' THEN
    ENTRY 'SUB01'.
        :
    SECT-1 SECTION.
        :
```

(6)　呼ばれる節と段落までの間にある節と段落

呼び出し元の PERFORM 文または手続き部先頭から（「(7)　注意事項」を参照），呼ばれる節と段落までの間にある節と段落が GO TO 文と ALTER 文の飛び先で参照されてはなりません。

例の PERFORM 文は，1.の GO TO 文によって SECT-1 が参照されているため，インライン化の対象となりません。

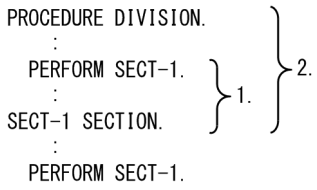
（例）

```
        :
PROCEDURE DIVISION.
        :
    GO TO SECT-1.  *> 1.
        :
```

```
PERFORM SECT-2.
 :
SECT-1 SECTION.
 :
SECT-2 SECTION.
 :
```

(7)　注意事項

- (5)，(6) の適用範囲を次の図に示します。
  - PERFORM 文の方が節と段落の定義より前に書かれたときは 1.の範囲。
  - 節と段落の定義の方が PERFORM 文より前に書かれたときは 2.の範囲。



- STOP RUN 文，-MainNotCBL オプションの指定，または内側のプログラムの EXIT PROGRAM 文，EXIT METHOD 文，EXIT FUNCTION 文，GOBACK 文を示します。宣言節中の節と段落は対象外となるため，EXIT USE 文，GO TO MORE-LABELS 文は対象となりません。
- インライン展開された個所で例外が発生した場合，EXCEPTION-LOCATION 関数で返される手続き名の値は，インライン展開されていない場合と相違があります。詳細については，「21.7.1　組み込み関数を使用した例外情報の参照」の「(2)　EXCEPTION-LOCATION 関数」の注を参照してください。

31.2.2　10 進項目の 2 進項目化

10 進項目の 2 進項目化とは，作業場所節および局所場所節で定義された内部 10 進項目，外部 10 進項目を 2 進項目として扱うことをいいます。すなわち，USAGE 句に BINARY が指定されているものとして扱います。-Bin1Byte オプションが指定されていても，けた数が 1～2 けたの 10 進項目なら 2 バイトの 2 進項目として扱います。ただし，次のどれかの条件に該当する場合は，2 進項目化の対象にはなりません。

(1)　2 進項目が指定できない個所に指定されているもの

- 画面機能（SCREEN SECTION）※の CURSOR データ名
- BLANK WHEN ZERO 句がある
- SIGN 句があるデータ名
- サブスキーマ節の STATUS データ名
- サブスキーマ節の DETAIL CODE データ名
- 部分参照がある
- 字類条件 一意名

- 英字，英数字，英数字型関数，英数字編集，数字編集，ZERO 以外の表意定数，英数字定数との比較条件（EVALUATE の選択主体と選択対象の比較も含む）
- EXAMINE 文 一意名 1
- INSPECT 文 一意名 1，一意名 3～一意名 n
- STRING 文 一意名 1，2，3
- TRANSFORM 文 一意名 1
- UNSTRING 文 一意名 4

注※　AIX で有効です。

(2)　2 進化することでサイズが大きくなるもの

次の場合，2 進化することでサイズが大きくなります。ただし，01／77 レベルは除きます。

- 1 けたの外部 10 進
- 1，5，10，11，12，13 けたの内部 10 進

(3)　データを外部と入出力するもの

- EXTERNAL 属性
- ADDR 関数の引数
- 次の文に指定された ADDRESS OF 一意名
  - CALL 文 REFERENCE 一意名
  - 比較条件 一意名
  - SET 文 送り出し側作用対象 一意名
  - SET 文 受け取り側作用対象 一意名
- ACCEPT 文
  ただし，次の一意名，データ名は除きます。
  - ACCEPT 一意名 FROM DATE／DAY／DAY-OF-WEEK／TIME
  - 画面節（SCREEN SECTION）※での ACCEPT 文の LINE／COLUMN 一意名
  - コマンド行の引数の個数を取得する ACCEPT 文の一意名 9
- DISPLAY 文
  ただし，次の一意名，データ名は除きます。
  - 画面節（SCREEN SECTION）※での DISPLAY 文の LINE／COLUMN 一意名
  - コマンド行の引数の読み込み位置を設定する DISPLAY 文の一意名 7
- CALL 文 USING

- CALL 文 RETURNING
- DISABLE 文 一意名 1
- ENABLE 文 一意名 1
- RECEIVE 文 一意名 1
- SEND 文 一意名 1，2，3，4
- TRANSCEIVE 文 一意名 1，2
- INVOKE 文 USING
- INVOKE 文 RETURNING
- 利用者定義関数

注※　AIX で有効です。

(4)　2 進にすると長さが異なるもの

- LENGTH 関数
- 次の文に指定された LENGTH OF 一意名
  - CALL 文 BY CONTENT 一意名
  - CALL 文 BY VALUE 一意名
  - INVOKE 文 BY CONTENT 一意名
  - INVOKE 文 BY VALUE 一意名

(5)　英数字転記／比較をするもの

MOVE 文の転記の規則に従って転記する WRITE FROM，REWRITE FROM，RELEASE FROM，報告書節（REPORT SECTION）の SOURCE 句，画面節（WINDOW SECTION）※の SOURCE 句などを含みます。

- 集団項目との比較（EVALUATE の選択主体と選択対象の比較も含む）
- MOVE 2 進項目 TO 集団項目
- MOVE 集団項目 TO 2 進項目
- READ 文 INTO 一意名
- RETURN 文 INTO 一意名
- -H8Switch オプションがあるときの MOVE 2 進項目 TO 英数字／英数字編集項目

注※　AIX で有効です。

(6) 外部 10 進，内部 10 進，2 進で設定される値が異なるもの

- MOVE WHEN-COMPILED TO 一意名
- MOVE CURRENT-DATE TO 一意名
- MOVE 外部 10 進項目／内部 10 進項目／2 進項目／浮動小数点項目／数字編集項目以外の項目，英数字型関数 TO 一意名
  （MOVE 文の転記の規則に従って転記する場合も含む）

(7) 10 進項目を間接的に参照／更新するもの

- 10 進項目を含む集団項目が参照／更新されている

  注

  CORRESPONDING 指定のある ADD 文，SUBTRACT 文，MOVE 文，および INITIALIZE 文で指定した集団項目中の 10 進項目は 2 進化されません。
- 10 進項目が再定義項目または被再定義項目となっている
- 10 進項目を含む集団項目が再定義項目または被再定義項目となっている
- 10 進項目にアドレス名指定がある
- 10 進項目を含む集団項目にアドレス名指定または VALUE 句がある
- 再命名項目（RENAMES 句がある）
- 被再命名項目（THRU なし）
- 被再命名項目（THRU あり）
- 被再命名項目（THRU 範囲内の項目）

(8) 2 進化すると実行速度が遅くなるもの

- MOVE 2 進 TO 数字編集項目（MOVE 文の転記の規則に従って転記する場合も含む）

(9) ファクトリ定義およびインスタンス定義中の 10 進項目

- ファクトリ定義およびインスタンス定義中の 10 進項目

(10) その他

- 作業場所節または局所場所節以外で定義されている
- SYNCHRONIZED 句がある
- FILLER 項目
- 特殊レジスタ
- 可変長集団項目中の 10 進項目

- 外部 10 進項目を含む集団項目に SIGN LEADING 指定または SIGN SEPARATE 指定がある
- PICTURE 句の左端に「P」がある
- CSV 編成ファイルに対する WRITE 文 FROM 一意名

31.2.3　不変式のループ外移動

繰り返し処理中で結果が常に同じになるような式や代入を繰り返し処理の前に移動することによって，繰り返し内の実行命令数を削減します。

(最適化前)

```
          :
          :
          :
       PERFORM VARYING I
         FROM 1 BY 1 UNTIL I = 100
           MULTIPLY A BY B GIVING C(I)
           MOVE 10 TO D
       END-PERFORM.
```

(最適化後)

```
          :
       MULTIPLY A BY B GIVING temp.
       MOVE 10 TO D.
       PERFORM VARYING I
         FROM 1 BY 1 UNTIL I = 100
           MOVE temp TO C(I)
       END-PERFORM.
          :
```

演算の最適化条件

- 四則演算である。

代入の最適化条件

- 代入元変数（この最適化によって前に出された式の結果の一時的記憶域を含む）がループ内で不変である。
- 代入先変数はループ内ではこの代入以外で変更されない。
- 代入元変数として代入先変数を参照している場合，必ずこの代入の結果を参照する。すなわち，ループ外で設定された値を参照することはない。

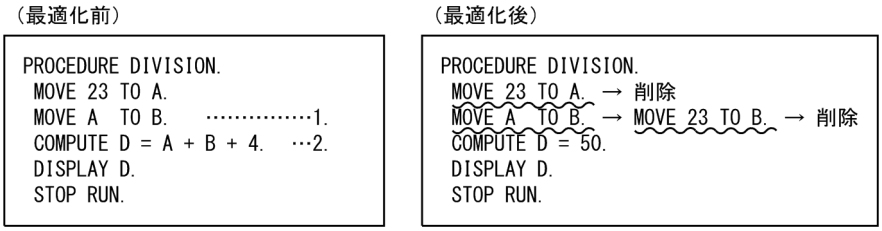
31.2.4　コピー伝播

変数 1=定数

変数 1=変数 2

このような単純な代入文があったとき，次の変数 1 への代入までに現れた変数 1 の参照を定数や変数 2 で置き換えることによって，定数の畳み込みや無用命令の削除（代入があり，次の代入または終わりまでに参照がない場合，その代入は無用である）などをします。ただし，-Optimize,2 オプション指定時には，ユーザが定義したデータ項目への代入に対する無用命令の削除はしません。

次の例では，A，B はユーザ定義変数ではないものとします。



定数"23"を代入している A は 1.，2.で参照されているだけなので，1.，2.の A はそれぞれ定数"23"に変更されます。さらに，B は 2.で参照されているだけなので，B も定数"23"に変更されます。なお，この例では 2.の COMPUTE 文が定数の演算だけになるため，定数の畳み込みによって最終的に定数"50"に変更されます。

最適化条件

次の条件を満たす代入 X=Y によってだけコピー伝播は行われます。

- X は添字指定を含まない。
- X に対する別名はない。
- Y が定数なら，Y は X のデータの型およびサイズに変換できる。
- Y が変数なら，Y は X のデータの型およびサイズに等しい。

31.2.5 共通式の削除

同一の式が複数ある場合，最初の式だけ計算をし，残りの式は最初の式の結果で置き換えることによって演算時間の短縮を図ります。

(最適化前)

```
COMPUTE C = A + B.
COMPUTE D(I) = A + B.
COMPUTE E = A + B + C.
MOVE D(I) TO F.
```

(最適化後)

```
ADD     A TO B GIVING C temp.
MOVE    temp TO D(I).
COMPUTE E = temp + C.
MOVE D(I) TO F.
```

最適化条件

二つの式 X と Y の間で式の共通化が行われる条件は次のとおりです。

- 式 X は Y の前にある。
- 式 X または Y がループ内にある場合，式 Y または X も同一ループ内にある。
- 式 Y は代入を含まない。
- 式 X，式 Y の両方で参照である同じ演算項に対して，式間でこの演算項への代入がない。

31.2.6　定数の畳み込み

式の中に複数の定数があった場合，計算可能な範囲でコンパイラが定数同士の演算を計算することによって式を単純化します。

(最適化前)

```
COMPUTE C = 1 + 3.
COMPUTE D = 0.1415926 + 3.0000000 + A.
```

(最適化後)

```
MOVE    4 TO C.
COMPUTE D = 3.1415926 + A.
```

最適化条件

COBOL の演算順序に従った演算の過程で定数同士の演算が発生する場合だけ，この最適化は行われます。

31.2.7　演算強さの軽減

(1)　ループの削除

次のようなループを削除します。

- ループ本体が空であり，かつループ制御変数が不要であるループ
- ほかの最適化によってループ本体が空になったループ

最適化条件

- ループ本体が空である。
- ループ制御変数が不要なループである。

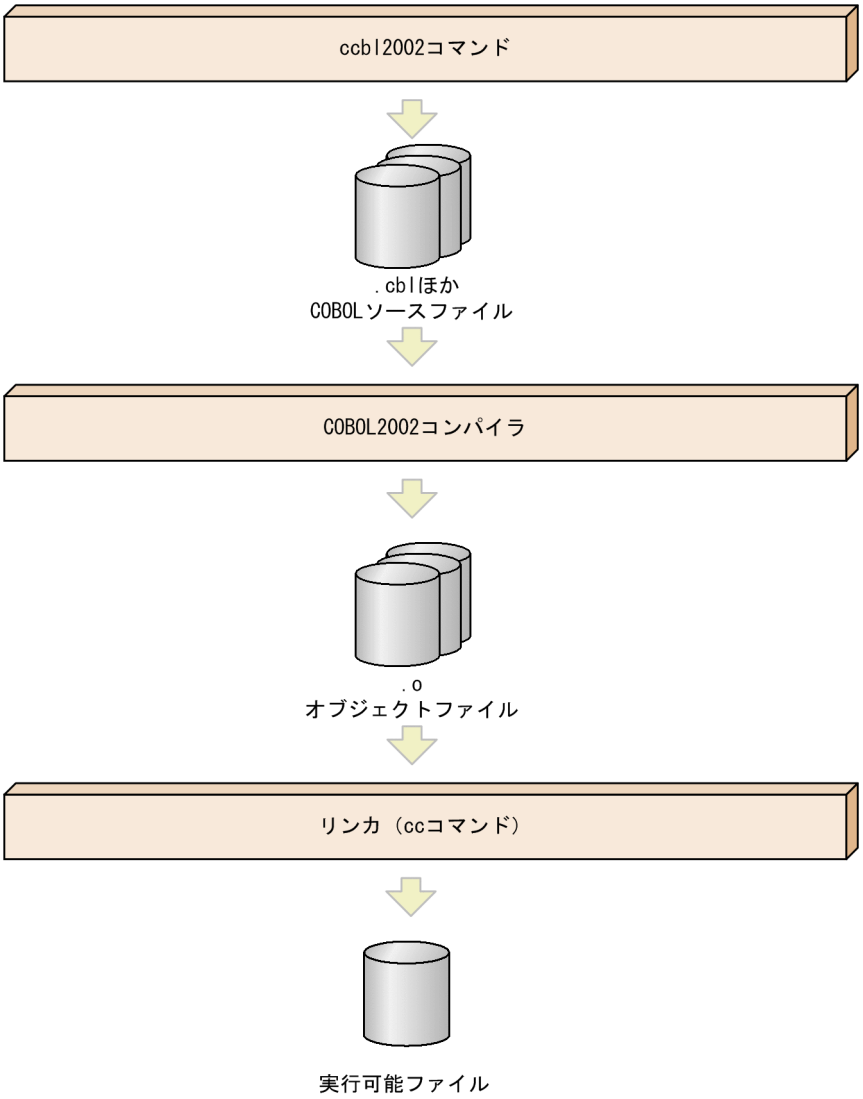
32 COBOL ソースの作成とコンパイル

この章では，COBOL ソースの作成方法とコンパイル方法について，コンパイラ環境の設定方法や，コンパイル時の入出力構成，使用するファイル，登録集原文の使用方法なども含めて説明します。

32.1　コンパイル時の主な入出力ファイル

プログラムをコンパイルするときの主な入出力の流れを次に示します。

図 32-1　コンパイル時の入出力の流れ



1. ccbl2002 コマンドを実行すると，COBOL2002 コンパイラに制御が渡り，COBOL ソースファイルからオブジェクトファイル（.o）が生成されます。
2. オブジェクトファイルがリンカに取り込まれ，実行可能ファイルが生成されます。

上記に示したファイルや，その他 COBOL2002 で使用するファイルの内容については，「付録 E　COBOL で使用するファイル」を参照してください。

32.2　COBOL ソースの作成方法

コンパイル対象の COBOL ソースファイルを作成するときの規則について説明します。

32.2.1　ソースファイル名と拡張子

ソースファイル名，拡張子の指定規則を次に示します。

(1)　ソースファイル名

ソースファイル名は UNIX のファイル名の規則に従って指定します。ただし，ccbl2002 コマンドに指定するソースファイル名に次の文字を使用した場合の動作は保証しません。

ハイフン（-）

(2)　拡張子

COBOL ソースファイルは，ファイル名のあとに次の拡張子を付けます。

- 固定形式正書法で書かれた原始プログラムをコンパイルする場合
  .cbl，.CBL※，.cob，.ocb，または環境変数 CBLFIX で指定した拡張子
  注※
  拡張子に等価規則は適用されません。
- 自由形式正書法で書かれた原始プログラムをコンパイルする場合
  .cbf，.ocf，または環境変数 CBLFREE で指定した拡張子

32.2.2　原始プログラムの作成規則

COBOL 原始プログラムは次の規則に従って記述します。

- 各行は復帰改行文字で終了させます。
- 固定形式正書法の場合は，1 行は 80 バイト以内（ただし，コンパイラ環境変数 CBLFIXEDFORMLINE=255 が有効な場合は，255 バイト以内）とします。また，自由形式正書法の場合は，1 行は 255 バイト以内とします。これらの制限を超えた分は無視されます。
- タブ文字以外の制御文字（X'00'～X'1F'，および X'7F'）は，使用できません。
- タブ文字は，次のタブ位置まで空白として扱われます。タブを空白に置き換えるときの空白の個数は，環境変数 CBLTAB で設定できます。

32.2.3　正書法

ここでは，COBOL ソースファイルの拡張子と正書法の関係，および SOURCE FORMAT 指令を使った正書法の切り替え方法について説明します。

正書法および SOURCE FORMAT 指令の文法規則については，マニュアル「COBOL2002　言語　標準仕様編」「2.　正書法（Reference format)」および「COBOL2002　言語　標準仕様編」「3.3.14　SOURCE FORMAT 指令」を参照してください。

(1)　正書法の種類と概要

正書法は，COBOL 原始プログラムや登録集原文を記述するための規約です。COBOL には，自由形式正書法と固定形式正書法の二つの正書法があります。それぞれの正書法の特徴を，次に示します。

- 固定形式正書法

  行の文字位置によって，一連番号領域，標識領域，プログラム記述領域などの用途がはっきり決められている書き方です。
- 自由形式正書法

  一連番号領域，標識領域などはなく，プログラムを行中の任意の位置に記述できる書き方です。

また，SOURCE FORMAT 指令を使用すると，2 種類の正書法を原始プログラムや登録集原文の途中で切り替えることができます。

なお，正書法の言語仕様についてはマニュアル「COBOL2002　言語　標準仕様編」「2.　正書法（Reference format)」を，SOURCE FORMAT 指令の言語仕様についてはマニュアル「COBOL2002　言語　標準仕様編」「3.3.14　SOURCE FORMAT 指令」を，それぞれ参照してください。

(2)　拡張子による正書法の指定

(a)　ソースファイルの拡張子と正書法の種類の対応

COBOL2002 のコンパイラは，拡張子によって COBOL ソースファイルの種類が固定形式正書法か，自由形式正書法かを区別します。

拡張子が.cbl，.CBL，.cob，.ocb の COBOL ソースファイル

固定形式正書法で記述されているものとして扱われます。

拡張子が.cbf，.ocf の COBOL ソースファイル

自由形式正書法で記述されているものとして扱われます。

また，上記以外の拡張子が付いたファイルであっても，環境変数 CBLFIX または CBLFREE に設定しておくことで，それぞれ固定形式正書法，自由形式正書法で書かれた COBOL 原始プログラムとしてコンパイルできます。

環境変数 CBLFIX または CBLFREE を使用して，COBOL 原始プログラムの拡張子を指定する方法については，「32.6　コンパイラ環境変数」を参照してください。

(b)　正書法の決定の優先順位

COBOL 原始プログラムの正書法

COBOL 原始プログラムが，固定形式／自由形式のどちらの正書法としてコンパイルされるかは，次の 1.～3.の判定順序に従って決定されます。

1. 環境変数 CBLFIX に設定した拡張子のファイルは，固定形式正書法で書かれた COBOL 原始プログラムとしてコンパイルされます。
2. 環境変数 CBLFREE に設定した拡張子のファイルは，自由形式正書法で書かれた COBOL 原始プログラムとしてコンパイルされます。
3. 拡張子が.cbl，.CBL，.cob，.ocb のファイルは，固定形式正書法で書かれた COBOL 原始プログラムとしてコンパイルされます。また，拡張子が.cbf，.ocf のファイルは，自由形式正書法で書かれた COBOL 原始プログラムとしてコンパイルされます。

したがって，同じ拡張子を環境変数 CBLFIX，環境変数 CBLFREE の両方に指定した場合，環境変数 CBLFIX の指定が有効となり，COBOL 原始プログラムは固定形式正書法としてコンパイルされます。

登録集原文の正書法

登録集原文の正書法も，COBOL 原始プログラムと同じ規則で決定されます。ただし，原文名定数に書かれた登録集原文ファイルの拡張子が，上記以外の拡張子の場合，登録集原文は，固定形式正書法で書かれたものとして扱われます。

登録集原文ファイルの検索

登録集原文ファイルは，次の 1.～4.の順に検索されます。ただし，原文名定数で書かれた場合は，除きます。

1. 環境変数 CBLFIX に設定した拡張子の左から順に，その拡張子が付くファイル。
2. 環境変数 CBLFREE に設定された拡張子の左から順に，その拡張子が付くファイル。
3. .cbl，.CBL，.cob，.ocb の順に，その拡張子が付くファイル。
4. .cbf，.ocf の順に，その拡張子が付くファイル。

上記の検索順位は，登録集原文の複写元の原始プログラムの正書法とは関係しません。

例えば自由形式の原始プログラム TP001.cbf の中に「COPY ABC.」という記述があって，ABC.cbl と ABC.cbf という二つのファイルがある場合，ABC.cbl の方が展開されます。

正書法の異なるプログラム間の複写

COPY 文を使用して，固定形式正書法の原始プログラムから自由形式正書法の原始プログラムを取り込んだり，自由形式正書法の原始プログラムから固定形式正書法の原始プログラムを取り込んだりできます。

この場合，ファイルの拡張子で正書法が判断されるため，SOURCE FORMAT 指令によって明示的に正書法を指定する必要はありません。

(3) SOURCE FORMAT 指令による正書法の切り替え

SOURCE FORMAT 指令を使用すると，原始プログラムの正書法を切り替えられます。原始プログラムの先頭に SOURCE FORMAT 指令を記述すると，原始プログラム全体の正書法を変更できます

また，原始プログラムの途中で SOURCE FORMAT 指令を記述すると，SOURCE FORMAT 指令の次の行から正書法を変更できます。

次に，SOURCE FORMAT 指令の使用例を示します。

(例 1)

固定形式正書法の COBOL ソースファイルに自由形式正書法を適用する場合

```
000100 >>SOURCE FORMAT IS FREE     …1.
IDENTIFICATION DIVISION.           …2.
PROGRAM-ID. FIX-FORM-EXAMPLE.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 A-LONG-ITEM PIC X(100) VALUE 'THE FIRST PART CONTINUED'-
   ' ON THE NEXT LINE - SIMPLE ENOUGH'.
01 NUM-ITEM PIC 9(10).
PROCEDURE DIVISION.
A-PARAGRAPH-NAME.
   :
```

1. SOURCE FORMAT 指令が有効となるのは，SOURCE FORMAT 指令の次の行からです。このため，「>>SOURCE FORMAT IS FREE」を指定している行自身は，固定形式正書法に従って記述する必要があります。
2. 別の SOURCE FORMAT 指令が現れるまで，1.の指定が有効となります。

(例 2)

自由形式正書法の COBOL ソースファイルに固定形式正書法を適用する場合

```
>>SOURCE FORMAT IS FIXED                    …1.
000010 IDENTIFICATION DIVISION.             …2.
000020 PROGRAM-ID. FIX-FORM-EXAMPLE.
000030 DATA DIVISION.
000040 WORKING-STORAGE SECTION.
000050 01 A-LONG-ITEM PIC X(100) VALUE 'THE FIRST PART CONTINUED
000060- '  ON THE NEXT LINE - SIMPLE ENOUGH'.
000070 01 NUM-ITEM PIC 9(10).
   :
```

1. SOURCE FORMAT 指令が有効となるのは，SOURCE FORMAT 指令の次の行からです。このため，「>>SOURCE FORMAT IS FIXED」を指定している行自身は，自由形式正書法に従って記述する必要があります。
2. 別の SOURCE FORMAT 指令が現れるまで，1.の指定が有効となります。

(4) 自由形式正書法で指定できないコンパイラオプション

次のオプションは，自由形式正書法で書かれたCOBOL原始プログラムとしてコンパイルされるプログラムには，指定できません。指定してコンパイルした場合，エラーとなってコンパイルが中止されます。

表 32–1 自由形式正書法で指定できないコンパイラオプション

オプション名	機能
-StdVersion	第1次規格／第2次規格の解釈でコンパイルする
-V3Rec	メインフレーム（VOS3）の固定長または可変長レコード形式のプログラムを，VOS3の日本語文字の扱いに合わせてコンパイルする
-CompatiV3	VOS3 COBOL85互換を指定する
-StdMIA	MIA仕様の範囲外チェックをする
-Std85	JIS仕様でチェックする
-EucPosition	EUCコード使用時，見かけ上の文字位置で固定形式正書法の境界を決定する

なお，固定形式正書法で書かれたソースの途中に「>>SOURCE FORMAT IS FREE」が指定されている場合も，これらのコンパイラオプションを指定するとエラーとなります。

32.3.1　原始文操作機能

ここでは，COBOL2002 の原始文操作機能について説明します。

(1)　COPY 文による登録集原文の取り込み

登録集原文（COPY 登録集）は，プログラムでよく利用する標準化した手続き，ファイル記述，完全なプログラムなどを登録したファイルです。プログラム中に COPY 文を記述すると，この登録集原文を複写してコンパイルできます。

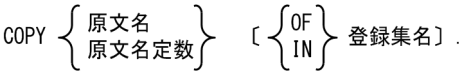
(2)　登録集原文のファイル形式

登録集原文は，ファイル単位に登録・複写します。

登録集原文を登録するファイルのファイル形式は，.cbl，.CBL，.cob，.ocb，.cbf，.ocf のどれかの拡張子の付いた形式とします。ただし，環境変数 CBLFREE または CBLFIX で自由形式拡張子，固定形式拡張子が設定してあれば，該当する拡張子の付いた形式も有効です。

(3)　COPY 文の指定形式

COPY 文の指定形式を次に示します。COPY 文の文法の詳細については，マニュアル「COBOL2002 言語　標準仕様編」「3.2.2　COPY 文」を参照してください。



原文名

- 原文名には，登録集原文が登録されているファイルの名称を，拡張子を付けないで指定します。
- 原文名には，かな文字と拡張コード文字は使用できません。また，原文名で英大文字と英小文字は区別されます。
- 原文名は，COBOL 語の定義に従い 31 文字まで指定できます。ただし，-Compati85,Syntax オプション指定時は，先頭の 30 文字までしか有効となりません。また，次のオプションを指定している場合は，先頭の 8 文字までしか有効となりません。

  -V3Rec,Fixed　-V3Rec,Variable　-CompatiV3

原文名定数

原文名定数は，登録集原文を登録してあるファイルの絶対パス名を引用符で囲んで指定します。このとき，ファイル名には拡張子も付けます。

-V3ConvtName オプションを指定したときの原文名定数の扱い

-V3ConvtName オプションを指定する場合，原文名定数には登録集原文を登録してあるファイルのファイル名を引用符で囲んで指定します。

-V3ConvtName オプションを指定すると，VOS3 COBOL85 のソースファイルとの互換のため，原文名定数中の文字が変換されて検索されます。変換規則については，「32.5.12　他システムとの移行の設定」の「(11)　-V3ConvtName オプション」を参照してください。

登録集名

登録集名は，登録集環境変数として次のように設定しておきます。

形式

```
登録集名=登録集検索ディレクトリ〔; …〕
export 登録集名
```

規則

- 登録集検索ディレクトリには，原文名に対応するファイルを検索する任意のディレクトリを絶対パスで指定します。このパス名に，原文名または原文名定数で指定したファイル名を連結して絶対パス名とします。
- 登録集環境変数名は，先頭が英大文字で始まる，英大文字と数字で構成される 8 文字以下の文字列でなければなりません。
- 登録集環境変数に設定するパス名は，複数指定できます。複数指定した場合，左側から指定した順番に検索されます。

(4)　登録集原文の検索順序

原文名で指定したファイルは，拡張子，ディレクトリの二つの条件で検索されます。それぞれの検索順序は次のようになっていて，両者のうちでは拡張子による検索順序が優先します。また，-V3ConvtName オプションを指定した場合，原文名定数に指定したファイルは，ディレクトリによる検索順序で検索されます。

拡張子による検索順序

1. 固定形式拡張子（環境変数 CBLFIX で設定）
2. 自由形式拡張子（環境変数 CBLFREE で設定）
3. .cbl
4. .CBL
5. .cob
6. .ocb
7. .cbf
8. .ocf

ディレクトリによる検索順序

1. 登録集環境変数で設定したディレクトリ
2. 環境変数 CBLLIB で指定したディレクトリ
3. カレントディレクトリ

例えば，固定形式拡張子，自由形式拡張子とも設定されていない場合，"ファイル名.cbl"で 1.～3.の順にディレクトリを検索し，目的のファイルがなければ，次に"ファイル名.CBL"で同様に検索します。

32.3.2　スタックコンパイル機能（連続コンパイル機能）の利用

スタックコンパイル機能とは，複数個の翻訳単位のソース単位を含む翻訳グループを一つのコンパイル単位としてコンパイルする機能です。この機能には次のような利点があります。

- 一つの COBOL ソースファイルに主プログラム，副プログラムを含めることができます。このソースファイルから作成される実行可能ファイルと 1 対 1 で対応するため管理がしやすくなります。
- 一つのライブラリを構成する副プログラム群を，同じソースファイルにまとめることができます。

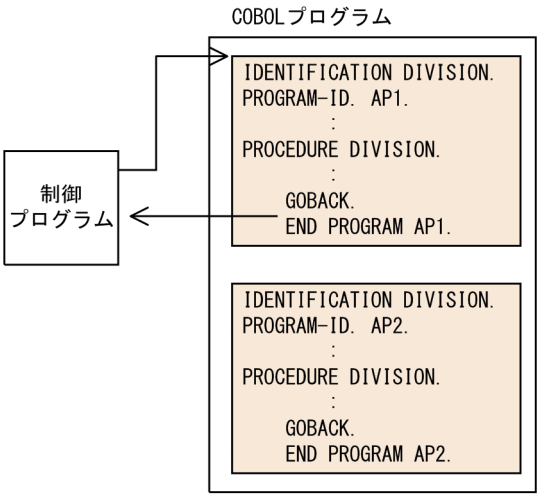
(1)　コンパイラオプションとの関係

主プログラム指定（-Main,System／-Main,V3）以外のオプションは，すべての翻訳単位のソース単位に対して有効となります。-Main,System，-Main,V3 オプションとほかのオプションとの関係については，「32.5.5　最終生成物の種類の設定」を参照してください。

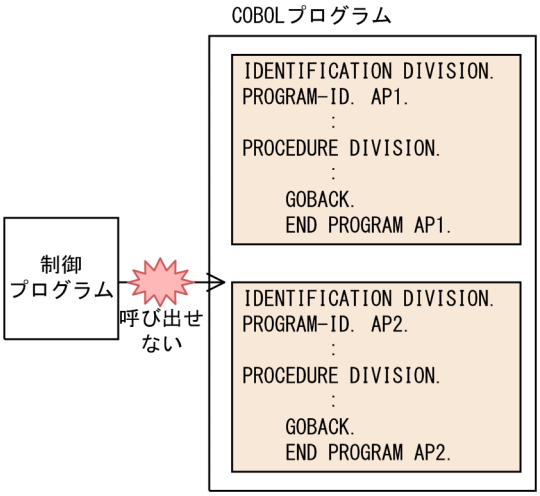
(2)　主プログラム指定

主プログラム指定（-Main,System／-Main,V3）を指定した場合は，ソースファイル中の先頭の最外側のプログラムが主プログラムとなります。したがって，制御プログラムからは先頭の最外側のプログラムだけが呼び出せます。2 番目以降の最外側のプログラムは，呼び出せません。

（呼び出せる例）



（呼び出せない例）



(3) スタックコンパイル（連続コンパイル）時の出現順序

クラス定義，インタフェース定義を含むソースファイルを，連続コンパイルできます。ただし，リポジトリ段落に指定された翻訳単位が別のソースファイルで定義されている場合，別のソースファイルを先にコンパイルしておく必要があります。詳細は，「33. 定義別のコンパイル方法とリポジトリファイル」を参照してください。

なお，インタフェース定義は，ほかのソース単位より先に記述されていてもかまいません。

(4) コンパイルリスト

スタックコンパイルを実行したときのコンパイルリストについては，「付録 D コンパイルリスト」を参照してください。

(5)　プログラム間連絡

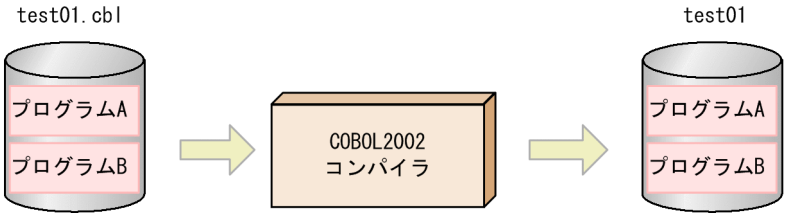
一つのコンパイル単位に最外側のプログラムが複数個含まれていても，これらの最外側のプログラムはすべて別のコンパイル単位の CALL 文で呼び出せます。

(6)　ソースファイルと各種ファイルの関係

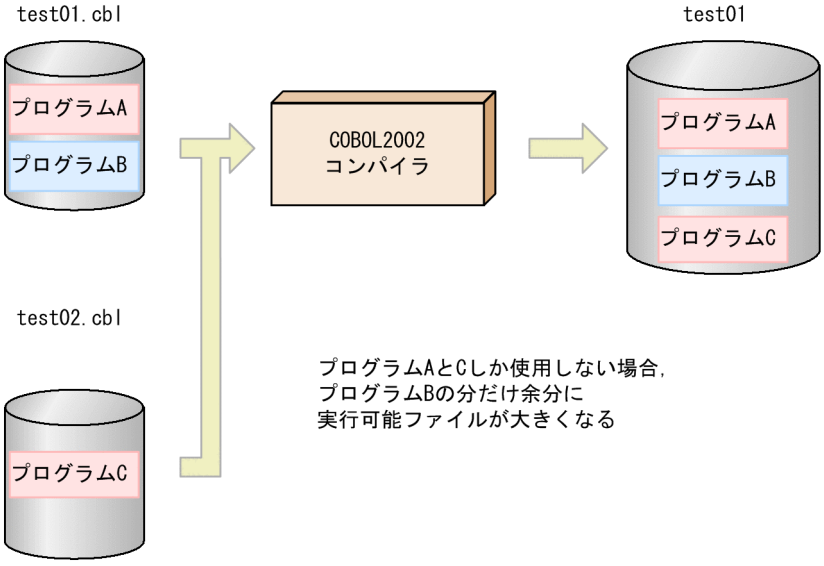
スタックコンパイル時にコンパイラが生成するファイルは，通常のコンパイル時に生成するファイルと同じです。

(7)　実行可能ファイルの作成単位

スタックコンパイル機能を使用した場合，オブジェクトファイルは，ソースファイル単位で作成されます。このため，ソースファイル中の複数のプログラムから，別々の実行可能ファイルを作成することはできません。



複数の最外側のプログラムを含むソースファイルをスタックコンパイルし，生成されたオブジェクトファイルの一部のプログラムだけを利用するような実行可能プログラムは，作成できます。ただし，使用しないプログラムを含んでいるため，必要以上に実行可能ファイルのサイズが大きくなります。



32.3.3　条件翻訳の利用

条件翻訳を使用すると，翻訳変数を用いた条件式の真偽に従って，ソースコードの有効／無効を制御できます。翻訳変数は，原始プログラム中に DEFINE を書いて定義する方法と，コンパイル時に-Define オプションを指定して定義する方法があります。

翻訳指令の詳細な構文規則および一般規則については，マニュアル「COBOL2002　言語　標準仕様編」「3.　翻訳指令機能」を参照してください。

条件翻訳の使用例を，次に示します。

（例 1）IF 指令の使用例

IF 指令は，分岐の条件翻訳をしたい場合に使用します。

>>IF に続く条件式の真偽によって，プログラムテキストの有効・無効を指示できます。

```
000100*>IFとDEFINED条件の例
000200 IDENTIFICATION DIVISION.
000300 PROGRAM-ID. IF-SYNTAX.
000400 DATA DIVISION.
000500 >>DEFINE CHR01 AS 'ABCD'
000600 WORKING-STORAGE SECTION.
000700 PROCEDURE DIVISION.
000800 >>IF CHR01 IS DEFINED
000810     *> 翻訳変数CHR01は定義されているのでこちらが有効になる
000900     DISPLAY '--- CHR01 IS DEFINED ---'.
001000 >>ELSE
001100     *> こちらの行はコンパイルの結果無効となる
001100     DISPLAY '--- CHR01 IS NOT DEFINED ---'.
001200 >>END-IF
```

（例 2）IF 指令および DEFINED 条件と-Define オプションの例

DEFINED 条件は，指定された翻訳変数名が定義済みかどうかを判定する条件文です。

次の原始プログラムでは，翻訳変数 DEBUG の定義の有無によって，IF 指令の真偽が決定します。コンパイル時に「-Define DEBUG」を指定すると，翻訳変数 DEBUG が定義され，IF 指令が真となります。-Define オプションを指定しない場合は，IF 指令が偽となります。

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. TEST001.
000300 PROCEDURE DIVISION.
000400
000500 >>IF DEBUG IS DEFINED
000600     DISPLAY 'this is debug message'.
000700 >>END-IF
```

デバッグ行を使用しても同様の処理ができますが，デバッグ行は第 4 次規格の廃要素であり，次回規格改訂時には削除される予定のため，条件翻訳を使用することを推奨します。

（例 3）EVALUATE 指令の例－書き方 1

EVALUATE 指令は，多方向分岐の条件翻訳をしたい場合に使用します。

WHEN 以下の条件によって，有効となるプログラムテキストが選択されます。

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. TEST001.
000300 DATA DIVISION.
000400 WORKING-STORAGE SECTION.
000500 PROCEDURE DIVISION.
000600 >>DEFINE EXD01 AS 10
000700
000800 >>EVALUATE EXD01 *> 翻訳変数 EXD01の値を評価する
000900   >>WHEN  3
001000     DISPLAY '--- 3          ---'.
001100   >>WHEN  15
001200     DISPLAY '--- 15         ---'.
001300   >>WHEN  9  THRU 11 *> 9から11の間にある場合
001400     DISPLAY '---  9 THRU 11 ---'. *> ここが有効となる
001500   >>WHEN OTHER
001600     DISPLAY '--- WHEN OTHER ---'.
001700 >>END-EVALUATE
```

(例 4) EVALUATE 指令の例－書き方 2

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. DEFINE01.
000300 DATA DIVISION.
000400 WORKING-STORAGE SECTION.
000500 >>DEFINE LEVEL AS 2
000600 PROCEDURE DIVISION.
000700 >>EVALUATE TRUE
000800   >>WHEN LEVEL < 3  *> ここが真
000900     DISPLAY 'lower'.
001000   >>WHEN LEVEL > 5
001100     DISPLAY 'higher'.
001200   >>WHEN OTHER
001300     DISPLAY 'other'.
001400 >>END-EVALUATE
```

(例 5) EVALUATE 指令の例－書き方 2

EVALUATE 指令では，一度真の条件が現れた場合，それ以降にさらに真の条件があっても実行されません。

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. DEFINE01.
000300 DATA DIVISION.
000400 WORKING-STORAGE SECTION.
000500 >>DEFINE LEVEL AS 2
000600 PROCEDURE DIVISION.
000700 >>EVALUATE TRUE
000800   >>WHEN LEVEL < 3 *> ここが真
000900     DISPLAY 'lower'.
001000   >>WHEN LEVEL > 5
001100     DISPLAY 'higher'.
001200   >>WHEN LEVEL < 5 *>ここも真だが，すでに真となるWHENが存在する。
001300     DISPLAY 'lower 2'. *> よってここは実行されない。
001400   >>WHEN OTHER
001500     DISPLAY 'other'.
001600 >>END-EVALUATE
```

（例 6）-Define オプションの使用例

次のプログラムを「ccbl2002 -Define CHR01=ABCD SAMPLE1.cbl」と指定してコンパイルした場合，「>>IF CHR01 = 'ABCD'」の条件が真となります。

```
ソースファイル名称：SAMPLE1.cbl
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. DEFINEOPTION.
000300 DATA DIVISION.
000400 WORKING-STORAGE SECTION.
000500 PROCEDURE DIVISION.
000600 >>IF CHR01 = 'ABCD'
000700     *> 翻訳変数CHR01は-DefineオプションでABCDと定義。
000750     *> よってこちらが真。
000800     DISPLAY '--- CHR01 IS "ABCD" ---'.
000900 >>ELSE
001000     *> こちらの行は偽
001100     DISPLAY '--- CHR01 IS NOT "ABCD" ---'.
001200 >>END-IF
```

注意事項

-Define オプションでは，翻訳変数値を文字列として扱います。このため，数値は渡せません。

32.3.4　条件翻訳結果のコンパイルリスト

条件翻訳の結果，コンパイル対象とならなかった行を，無効行と呼びます。

COBOL2002 では，コンパイルリスト上で，原始プログラム中の無効行となった部分を確認できます。また，コンパイルリストに無効行を出力しない機能もあります。

(1)　コンパイルリスト上の表示

条件翻訳の結果無効行となった行は，コンパイルリストに'X'が出力されます。

また，コンパイルリストに無効行を出力したくない場合は，-SrcList オプションに NoFalsePath サブオプションを付けて指定します。

コンパイルリストに関係するその他のオプションについては，「付録 D.1　リストの出力」を参照してください。

コンパイルリストの例 1

-SrcList,CopyAll オプションを指定した場合

```
D CP N ----+----1----+----2----+----3----+----4----+----5----+----6----+
       000100 IDENTIFICATION DIVISION.
       000200 >>DEFINE CMPVAR1 AS -123456
       000500 PROGRAM-ID. AAA.
       000900 PROCEDURE DIVISION.
       001000 >>IF CMPVAR1 IS DEFINED
       001100     DISPLAY '--- test (1) ---'.
       001200     DISPLAY '--- test (2) ---'.
```

```
       001300 >>ELSE
X      001400     DISPLAY '--- test (3) ---'.
X      001500     DISPLAY '--- test (4) ---'.
       001600 >>END-IF
```

（説明）

翻訳変数 CMPVAR1 が 000200 行で定義されているので，ELSE 側である 001400，001500 行は無効行となります。そのため，これらの行の先頭に X が出力されます。

コンパイルリストの例 2

-SrcList,CopyAll,NoFalsePath オプションを指定した場合

```
D CP N ----+----1----+----2----+----3----+----4----+----5----+----6----+
       000100 IDENTIFICATION DIVISION.
       000200 >>DEFINE CMPVAR1 AS -123456
       000500 PROGRAM-ID. AAA.
       000900 PROCEDURE DIVISION.
       001000 >>IF CMPVAR1 IS DEFINED
       001100    DISPLAY '--- test (1) ---'.
       001200    DISPLAY '--- test (2) ---'.
       001300 >>ELSE
       001600 >>END-IF
```

（説明）

-SrcList オプションに NoFalsePath サブオプションを指定した場合，無効行となった 001400，001500 行は，コンパイルリストに出力されません。

その結果，001600 行が詰められて，001300 行の次に出力されます。

(2) 無効行の定義

無効行の定義を，次に示します。

1. 条件翻訳として判定中の>>IF，>>ELSE，>>END-IF，>>EVALUATE，>>WHEN，>>END-EVALUATE が指定された行自身は，無効行とはなりません。
2. 条件翻訳のための翻訳指令行でも，無効な領域にある場合は無効行となります。
3. >>SOURCE FORMAT は，常に無効行となりません。

次に，各定義の例を示します。なお，例中の「X」は，その行が無効行となることを示します。

1.の例（その 1）

>>IF の>>ELSE 側が無効な場合，>>IF や>>ELSE などは無効行となりません。

```
      >>DEFINE ABC 123
      >>IF ABC = 123
          DISPLAY 'OK'.
      >>ELSE
X         DISPLAY 'NG'.
      >>END-IF
```

1.の例（その 2）

>>IF の>>ELSE 側が有効な場合の例です。

```
      >>DEFINE ABC 123
      >>IF ABC = 12345
X         DISPLAY 'NG'.
      >>ELSE
          DISPLAY 'OK'.
      >>END-IF
```

1.の例（その 3）

>>EVALUATE の例です。

```
      >>DEFINE CHAR01  AS  'ABCD'
      >>EVALUATE TRUE
        >>WHEN CHAR01 = 'ABCD'
          DISPLAY '0001'.
          DISPLAY '0002'.
        >>WHEN CHAR01 = 'XYZ'
X         DISPLAY '0003'.
X         DISPLAY '0004'.
        >>WHEN OTHER
X         DISPLAY 'OTHER'.
      >>END-EVALUATE
```

2.の例

無効な領域にある場合は，>>IF などの翻訳指令行自身も無効行となります。

```
      >>DEFINE ABC 123
      >>IF ABC = 123
          DISPLAY 'OK'.
      >>ELSE
X         DISPLAY 'NG'.
X         >>IF ABC = 456
X           DISPLAY 'OK'.
X         >>ELSE
X           DISPLAY 'NG'.
X         >>END-IF
      >>END-IF
```

3.の例

無効な領域にある場合でも，>>SOURCE FORMAT 指令は，無効行になりません（>>SOURCE FORMAT 指令は，どこに指定されても有効となります）。

```
      >>DEFINE ABC 123
      >>IF ABC = 123
          DISPLAY 'OK'.
      >>ELSE
X         DISPLAY 'NG1'.
X         DISPLAY 'NG2'.
          >>SOURCE FORMAT IS FREE
X         DISPLAY 'NG3'.
X         DISPLAY 'NG4'.
      >>END-IF
```

32.4　コンパイラの起動方法

COBOL2002 で作成したプログラムをコンパイルするには次の方法があります。

- ccbl2002 コマンドによる方法
  ccbl2002 コマンドを実行することで，COBOL ソースファイルをコンパイル，リンクし，実行可能ファイルを生成できます。
- ccbl コマンドによる方法
  ccbl コマンドを使用することで，COBOL85 形式のコンパイラオプションを指定して，プログラムのコンパイル，リンクができます。

32.4.1　ccbl2002 コマンド

ccbl2002 コマンドの入力によって，COBOL ソースファイル中のプログラムをコンパイルして，オブジェクトファイルや実行可能ファイルなどを出力できます。

ccbl2002 コマンドの形式を次に示します。

形式

```
ccbl2002 〔オプション〔…〕〕 ファイル名〔…〕
```

オプション

コンパイラオプションを指定します。

コンパイラオプションの指定方法には，ccbl2002 コマンドの引数に指定する方法と環境変数 CBLCOPT2002 で指定する方法の 2 種類があります。なお，ccbl2002 コマンドの引数と環境変数 CBLCOPT2002 の両方にコンパイラオプションを指定した場合は，引数で指定したオプションの方が優先されます。

個々のコンパイラオプションの機能と使用方法については，「32.5　コンパイラオプション」を参照してください。

ファイル名

コンパイルする COBOL ソースファイルや，その他必要なファイルの名称を指定します。指定できるファイルの種類は次のとおりです。

- COBOL ソースファイル（.cbl ほか）
- オブジェクトファイル（.o）
- HTML ファイル※1（.htm，.html）
- ライブラリファイル（共用ライブラリ）※2

注※1

AIX で有効です。

注※2

ライブラリファイルやライブラリ検索パスの指定方法は，システムの規則に従ってください。

指定規則

- 各引数は，一つ以上の空白，タブ，または改行文字で区切ります。

  ただし，ccbl2002 コマンドに空白を含むファイル名を指定する場合は，ファイル名をアポストロフィ（'）で囲む必要があります。
- コマンドライン上には，リンカに渡すオプションも指定できます。この場合，-Link オプションを使用します。-Link オプションの使い方については，「32.5　コンパイラオプション」を参照してください。

終了コード

ccbl2002 コマンドは，次の終了コードを返します。

終了コード	意味	出力される エラーメッセージのレベル
0	コンパイルは正常終了した。	I レベルエラー W レベルエラー
1	重大エラーが発生した。	S レベルエラー
2	回復不能エラーが発生した。	U レベルエラー

なお，エラーの詳細については，コンパイルリストやエラーメッセージを参照してください。

ccbl2002 コマンドのヘルプ

オプションもファイル名も指定しないで，ccbl2002 コマンドだけが入力された場合，または-Help オプションを指定された場合に ccbl2002 コマンドのヘルプを出力できます。

形式

```
ccbl2002 〔-Help〕
```

32.4.2　ccbl コマンド（AIX で有効）

ccbl コマンドは，COBOL85 から COBOL2002 への移行を円滑にするために使用するコマンドです。

ccbl コマンドを使用すると，COBOL85 形式のコンパイラオプションを指定して，COBOL プログラムをコンパイルおよびリンクできます。これによって，既存の COBOL85 で作成した資産（コンパイル用のメイクファイルなど）を COBOL2002 で流用できます。ただし，COBOL2002 で新たに追加されたコンパイラオプションは，ccbl コマンドでは使用できません。したがって，通常は，ccbl2002 コマンドを使用してください。

ccbl コマンドは，オプションの指定形式が ccbl2002 コマンドとは異なりますが，以下の「注意事項」にある場合を除き，コンパイル結果や生成するオブジェクトは，ccbl2002 と同じです。ただし，ccbl コマ

ンドによって生成したさまざまなファイル（オブジェクトファイル，実行可能ファイル，プログラム情報ファイルなど）は COBOL85 環境で使用することはできません。

形式

```
ccbl 〔オプション〔…〕〕 ファイル名〔…〕
```

- ccbl コマンドの形式，指定方法，および終了コードは，COBOL85 の ccbl コマンドと同じです。また，コンパイラオプションの指定形式も，COBOL85 と同じ形式です。
- COBOL85 形式のコンパイラオプションの詳細については，「付録 H　COBOL85 と COBOL2002 のコンパイラオプションの対応」を参照してください。

ccbl コマンドのヘルプ

オプションもファイル名も指定せず，ccbl コマンドだけが入力された場合，ccbl コマンドのヘルプを出力できます。なお，COBOL85 でサポートされているオプションだけが出力され，COBOL2002 で新規に追加されたオプションは出力されません。

形式

```
ccbl
```

注意事項

- ccbl コマンドには，COBOL2002 形式のコンパイラオプションを指定できません。また，COBOL2002 で新規に追加されたコンパイラオプションは，ccbl コマンドには指定できません。
- コンパイルリストやコンパイラオプション詳細情報表示では，ccbl コマンドに指定された旧形式のオプションは，新形式に変換されて表示されます。
- ccbl コマンドは，常に-Compati85,All オプションを暗黙的に仮定して動作します。

32.5　コンパイラオプション

ここでは，ccbl2002 コマンドに指定するコンパイラオプションについて説明します。なお，COBOL85 のコンパイラオプションと COBOL2002 のコンパイラオプションとの対応については，「付録 H COBOL85 と COBOL2002 のコンパイラオプションの対応」を参照してください。

32.5.1　構文規則

コンパイラオプションの構文規則について説明します。

(1)　オプション指定項

コマンドライン上で，コンパイラオプションを指定する部分をオプション指定項と呼びます。

次の例の下線部分は，それぞれオプション指定項です。

```
ccbl2002 -Option,SubOption -AnotherOption filename.cbl
```

(2)　オプション指定項の構成

一つのオプション指定項を構成する要素，および構成要素の名称を次に示します。

```
-Option,SubOption Argument
   1.       2.        3.
```

- 1.をオプション名と呼びます。
- 2.をサブオプション名と呼びます。
- 3.をオプションの引数と呼びます。上記の例では，サブオプション名に引数が従属していますが，オプション名に直接引数が従属することもあります。
- 2.と 3.を合わせて，サブオプションと呼びます。
- 1.～3.を合わせて，オプションと呼びます（オプションは，一つのオプション指定項全体を表します）。

規則

- あるオプション Option に従属するオプションのことを「Option のサブオプション」と呼びます。また，説明の対象となるサブオプションがどのオプションに従属するか明確な場合には，主従関係を特定しないで単に「サブオプション」と呼びます。
- SubOption が Option のサブオプションである場合，構文規則およびオプション指定項では，次のように記述します。

  -Option,SubOption

  また，複数のサブオプションを持つオプションは，次のように記述します。

-Option,SubOption1,SubOption2

- オプションとサブオプションの区切りには，コンマ（,）を使用します。また，サブオプション同士の区切りにもコンマを使用します。
- オプションには，サブオプションのほかに，ファイル名や数値などの値を引数として持つものもあります。

  オプション Option が引数 Argument を持つ場合は，次のように記述します。

  -Option Argument
- 引数は，オプション一つにつき一つだけしか指定できません。
- オプション名同士，およびオプション名と引数の区切りには，半角空白文字を使用します。

32.5.2　一般規則

コンパイラオプションの一般規則について説明します。

- 大文字小文字は，等価とみなされます。
- オプション指定項の先頭文字は，ハイフン（-）となります。
- オプションとサブオプションを区切るコンマの前後には，空白を入れてはいけません。
- コンパイラオプションの指定に構文誤りがある場合，コンパイル時に回復不能（U レベル）エラーとなり，コンパイルが中止されます。

32.5.3　コンパイラオプションの優先順位

複数のコンパイラオプションを指定した場合の，各オプションの優先順位を次に示します。

(1)　指定個所による優先順位

コンパイラオプションを指定した個所によって，優先順位が高くなります。

ccbl2002 コマンドの場合

次の順序で，コンパイラオプションの優先順位が高くなります。

優先度	指定個所	オプションの形式
1	ccbl2002 のコマンド行に指定したオプション	新形式
2	環境変数 CBLCOPT2002 に指定したオプション	新形式

ccbl コマンドの場合

次の順序で，コンパイラオプションの優先順位が高くなります。

優先度	指定個所	オプションの形式
1	ccbl のコマンド行に指定したオプション	旧形式
2	環境変数 CBLCOPT に指定したオプション	旧形式
3	ccbl コマンドで暗黙的に仮定されるオプション	新形式

環境変数 CBLCOPT2002，CBLCOPT については，「32.6　コンパイラ環境変数」を参照してください。

(2)　オプション間の優先順位

コンパイラオプションには，オプション同士が背反の関係となっていたり，あるオプションを指定するとほかのオプションが仮定されたりするものがあります。このようなオプション間の優先順位を，次に示します。

(a)　あとに指定した方のオプションが有効となる場合

次のオプションを同時に指定した場合，あとに指定した方のオプションが有効となります。

- -DigitsTrunc／-Comp5
- -V3Rec／-EquivRule
- -CompatiV3／-EquivRule

(b)　オプションを指定すると，ほかのオプションが無効となる場合

次のコンパイラオプションを指定した場合，無効となるオプションがあります。

指定したオプション	無効となるオプション
-MainNotCBL	• -Main
-Compile	• -OutputFile
-Compile,CheckOnly	• -DebugInf • -DebugCompati • -DebugData • -TDInf • -CVInf • -DebugRange • -TestCmd • -PIC • -Profile • -IdentCall
-StdMIA -Std85 -Std2002	• -V3Spec • -StdVersion • -CompatiV3 • -H8Switch

指定したオプション	無効となるオプション
	• -Cblctr • -IgnoreLCC • -JPN • -CmDol • -Comp5 • -NumAccept • -V3Rec • -V3RecFCSpace • -V3RecEased • -EquivRule,NotAny • -SQL • -SQLDisp • -BinExtend • -MaxDigits38 • -IntResult,DecFloat40 • -LiteralExtend
-StdMIA	• -EquivRule,StdCode • -Bin1Byte
-V3Spec	• -Comp5 • -CmAster • -BinExtend • -CBLVALUE • -Bin1Byte • -NumAccept • -CmDol • -MaxDigits38 • -IntResult,DecFloat40 • -LiteralExtend • -V3RecEased
-Repository,Gen	• -Compile
-PIC	• -Profile
-MultiThread	• -Profile
-UniObjGen	• -JPN • -CompatiV3※1 • -V3Rec※1 • -V3RecFCSpace※1 • -V3RecEased※1
-Compati85,Syntax	• -LiteralExtend
-IntResult,DecFloat40	• -Compati85,Power（-Compati85,All 指定時も含む）

指定したオプション	無効となるオプション
	• -CompatiV3※2
-CompatiV3	• -LiteralExtend
-SimMain -SimSub -SimIdent	• -LiteralExtend

注※1

コンパイラ環境変数 CBLV3UNICODE に YES を指定した場合，無効になりません。

注※2

Linux(x64)では無効になりません。

(c) 仕様チェックオプションを複数指定した場合

-StdMIA オプション，-Std85 オプション，-Std2002 オプションは，同時に指定できません。同時に指定した場合，警告のメッセージが出力され，次の優先順位で有効となります。

1. -Std2002 オプション
2. -Std85 オプション
3. -StdMIA オプション

(d) オプションを指定することによって，仮定されるオプション

次のコンパイラオプションを指定した場合，仮定されるオプションがあります。

指定したオプション	仮定されるオプション
-V3Spec	• -V3Rec,Variable※1
-CompatiV3	• -JPN,Alnum※2 • -V3Rec,Variable
-DebugCompati	• -DebugInf
-DebugData	• -DebugInf
-DebugRange	• -DebugInf • -DebugCompati
-TestCmd	• -DebugInf • -TDInf
-TDInf	• -DebugInf
-TDInf と-Optimize,3 を同時に指定	• -Optimize,2 （-Optimize,3 を指定しても-Optimize,2 が仮定される）
-CVInf	• -DebugInf

指定したオプション	仮定されるオプション
-StdMIA,14	• -StdMIA,13
-PIC	• -Compile,NoLink
-MaxDigits38 および-IntResult,DecFloat40 と-Optimize,3 を同時に指定	• -Optimize,2 (-Optimize,3 を指定しても-Optimize,2 が仮定される)
-SpaceAsZero と-Optimize,3 を同時に指定	• -Optimize,2 (-Optimize,3 を指定しても-Optimize,2 が仮定される)

注※1

-V3Spec オプションと-UniObjGen オプションを同時に指定した場合，-V3Rec,Variable オプションは仮定されません。

注※2

コンパイラ環境変数 CBLV3UNICODE に YES を指定し，-UniObjGen オプションを指定した場合は，-CompatiV3 オプションを指定しても-JPN,Alnum オプションは仮定されません。

(e) ほかのオプションの指定を必要とするオプション

次のコンパイラオプションは，同時に指定する必要があるオプションを指定しない場合，無視されます。

指定したオプション	同時に指定する必要があるオプション
-SQLDisp	-SQL,ODBC
-UniEndian	-UniObjGen
-V3RecFCSpace	-V3Rec
-V3RecEased	-V3Rec
-CheckUninitData	-Compile,CheckOnly

次のコンパイラオプションは，同時に指定する必要があるオプションを指定しない場合，コンパイルエラーとなります。

指定したオプション	同時に指定する必要があるオプション
-IntResult,DecFloat40	-MaxDigits38
-MaxDigits38	-IntResult,DecFloat40
-SpaceAsZero	-Compati85,All
-VOSCBL,OccursKey	-CompatiV3
-VOSCBL,ReportControl	-CompatiV3
-VOSCBL,DataComm	-CompatiV3
-VOSCBL,RedefinesData	-CompatiV3
-VOSCBL,AssignDataToDevice	-CompatiV3

指定したオプション	同時に指定する必要があるオプション
-VOSCBL,EvaluateWhenOther	-CompatiV3
-JPN,V3JPNSpace	-CompatiV3

(3) 同じオプションを重複して指定した場合の規則

同じオプションを重複して指定した場合，次の規則に従ってオプションが決定されます。なお，オプションごとのサブオプションの指定規則については，「32.5.4 コンパイラオプションの一覧」を参照してください。

(a) 背反関係にあるサブオプション同士を指定した場合

最後に指定されたオプションが有効となります。

（例）

-Compile，{CheckOnly ｜ NoLink}

に対して

-Compile,CheckOnly -Compile,NoLink

と指定した場合，-Compile,NoLink が有効となります。

ただし，次の場合は，特定のサブオプションが有効となります。

- -JPN,V3JPNSpace オプションと-JPN,Alnum オプションを重複して指定した場合は，-JPN,V3JPNSpace オプションが有効となります。
- -JPN,V3JPNSpace オプションと-JPN,V3JPN オプションを重複して指定した場合は，-JPN,V3JPNSpace オプションが有効となります。
- -JPN,Alnum オプションと-JPN,V3JPN オプションを重複して指定した場合は，-JPN,V3JPN オプションが有効となります。
- -Std85,High オプション，-Std85,Middle オプション，-Std85,Low オプションを重複して指定した場合は，次の優先順位で有効となります。
  1. -Std85,Low オプション
  2. -Std85,Middle オプション
  3. -Std85,High オプション

(b) 省略可能なサブオプション同士，または複数選択できるサブオプション同士を指定した場合

前に指定したサブオプションの指定を引き継ぎ，あとに指定したサブオプションの指定が追加で有効となります。

（例）

-ErrSup {,I | ,W} ＋

に対して

-ErrSup,I -ErrSup,W

と指定した場合，Iサブオプションと Wサブオプションの両方が有効となります。これは，「-ErrSup,I,W」を指定した場合と同じです。

(4) オプションの打ち消し指定

デフォルト設定や環境変数などで指定済みのオプションを，コンパイル時に打ち消したい場合は，プリフィクス'no'の付いたコンパイラオプションを使用します。

例えば，環境変数 CBLCOPT2002 でオプション「-Details」が指定されている場合，コマンドラインで「ccbl2002 -noDetails …」と指定すると，環境変数 CBLCOPT2002 で設定済みのオプション「-Details」を打ち消せます。

規則

- オプションにデフォルト値の設定がある場合，プリフィクス'no'は指定できません。この場合，オプションのデフォルト値を指定することで，オプションを打ち消します。

  例えば，環境変数 CBLCOPT2002 でオプション「-Optimize,3」が指定されている場合，デフォルト値に戻したいときは，コマンドラインで「-Optimize,1」を指定します。
- 通常のオプションと打ち消しのオプションを同時に指定した場合は，あとに指定したオプションが有効となります。

注意事項

コンパイルリストに出力されるオプション一覧や，-Details オプションを指定した場合にコマンドラインへ出力されるコンパイラオプションの詳細情報表示では，no 指定のコンパイラオプションは表示されません。コンパイラオプションとして有効となったオプションだけが表示されます。

32.5.4 コンパイラオプションの一覧

コンパイラオプションの一覧を次に示します。

(1) 最終生成物の種類

最終生成物※の種類を設定するコンパイラオプションを，次に示します。

注※

最終生成物とは，コンパイラが最終的に生成する実行可能ファイルのことを示します。

表中のコンパイラオプションの詳細については，「32.5.5 最終生成物の種類の設定」を参照してください。

オプション	機能	OS	
		AIX	Linux
-Main,｛System ｜ V3｝ ファイル名	先頭の最外側プログラムを主プログラムとして作成する。	○	○
-PIC,｛Std ｜ Expand｝ -noPIC	共用ライブラリに使う位置独立コードの作成	○	○

（凡例）

○：サポートしている

(2) 製品連携

他製品との連携を設定するコンパイラオプションを，次に示します。

表中のコンパイラオプションの詳細については，「32.5.6 他製品との連携の設定」を参照してください。

オプション	機能	OS		
		AIX(32)	AIX(64)	Linux
-SQL,｛XDM ｜ ODBC〔,NoCont〕｝※ -noSQL	埋め込み SQL 文を XDM/RD または ODBC インタフェース機能で使用できるようにする。	○	○	○
-SQLDisp -noSQLDisp	埋め込み SQL 文に用途（USAGE 句）が表示用（DISPLAY）のデータ項目を指定できるようにする。	×	×	○
-RDBTran -noRDBTran	HiRDB による索引ファイル入出力機能を使用する。	○	○	×
-XMAP,LinePrint -noXMAP	書式印刷機能を使用して，順編成ファイルをプリンタに出力する。	○	×	×

（凡例）

○：サポートしている

×：サポートしていない

注※

ODBC サブオプションは，Linux で有効です。

(3) 実行

実行時の動作を設定するコンパイラオプションを，次に示します。

表中のコンパイラオプションの詳細については，「32.5.7 実行の設定」を参照してください。

オプション	機能	OS	
		AIX	Linux
-NumAccept	ACCEPT 文に数字項目を指定できるようにする。	○	○

オプション	機能	OS	
		AIX	Linux
-noNumAccept			
-NumCsv -noNumCsv	CSV 編成ファイルで，セルデータを数値として入出力できるようにする。	○	○
-MultiThread -noMultiThread	マルチスレッド対応 COBOL プログラムを作成する。	○	○
-MainNotCBL -noMainNotCBL	すべて副プログラムとして作成する。	○	○

（凡例）

○：サポートしている

(4) 最適化

プログラムの最適化を設定するコンパイラオプションを，次に示します。

表中のコンパイラオプションの詳細については，「32.5.8 プログラムの最適化の設定」を参照してください。

オプション	機能	OS	
		AIX	Linux
-Optimize，{0 | 1 | 2 | 3}	コンパイル時の最適化のレベルを指定する。	○	○

（凡例）

○：サポートしている

(5) デバッグ

デバッグを設定するコンパイラオプションを，次に示します。

表中のコンパイラオプションの詳細については，「32.5.9 デバッグの設定」を参照してください。

オプション	機能	OS	
		AIX	Linux
-DebugLine -noDebugLine	デバッグ行を有効にする。	○	○
-DebugInf〔,Trace〕 -noDebugInf	異常終了時，エラー要約情報を出力する。	○	○
-DebugCompati -noDebugCompati	実行時に次のチェックをする。 • 添字，指標名の繰り返し回数の範囲外チェック • プログラム間整合性チェック	○	○

オプション	機能	OS	
		AIX	Linux
-DebugData〔,ValueHex〕 -noDebugData	データ例外を検出する。	○	○
-TDInf -noTDInf	テストデバッグ情報を出力する。	○	○
-CVInf -noCVInf	カバレージ情報を出力する。	○	○
-DebugRange -noDebugRange	添字，指標名の繰り返し回数について，次元ごとの範囲外チェックをする。	○	○
-TestCmd｛,Full｜,Break｜,Sim｝＋ -noTestCmd	TD コマンド格納ファイルに出力する情報の種類を指定する。	○	○
-SimMain プログラム名 -noSimMain	主プログラムをシミュレーションする。	○	○
-SimSub プログラム名 -noSimSub	副プログラムをシミュレーションする。	○	○
-SimIdent -noSimIdent	副プログラム（一意名 CALL 文）をシミュレーションする。	○	○

（凡例）

○：サポートしている

(6) リンク

リンクを設定するコンパイラオプションを，次に示します。

表中のコンパイラオプションの詳細については，「32.5.10 リンクの設定」を参照してください。

オプション	機能	OS	
		AIX	Linux
-Compile,｛CheckOnly｜NoLink｝ -noCompile	コンパイルの処理範囲を指定する。	○	○
-OutputFile ファイル名 -noOutputFile	生成する実行可能ファイル名を指定する。	○	○
-Link オプションの並び -noLink	リンカに渡すオプションを指定する。	○	○
-DynamicLink,｛Call｜IdentCall｝ -noDynamicLink	ダイナミックリンクを使用する。	○	○

オプション	機能	OS	
		AIX	Linux
-OldLinkOpt,GCBypass -noOldLinkOpt	ccbl2002 コマンドで内部的に-bgcbypass リンカオプションを仮定する。	○	×
-OldStyleObject -noOldStyleObject	オブジェクトの形式を COBOL2002 V3 以前と同様の形式に戻す。	○	×

(凡例)

○：サポートしている

×：サポートしていない

(7) 規格

規格仕様のチェックを設定するコンパイラオプションを，次に示します。

表中のコンパイラオプションの詳細については，「32.5.11 規格の設定」を参照してください。

オプション	機能	OS	
		AIX	Linux
-StdMIA {,13 | ,14} ＋ -noStdMIA※	MIA 仕様の範囲外チェックをする。	○	○
-Std85 {, {High | Middle | Low} | ,Obso | ,Report} ＋ -noStd85※	JIS 仕様をチェックする。	○	○
-Std2002 {,OutRange | ,Obso | ,Archaic} ＋ -noStd2002	COBOL2002 規格仕様をチェックする。	○	○
-StdVersion, {1 | 2} -noStdVersion※	第 1 次規格／第 2 次規格の解釈でコンパイルする。	○	○

(凡例)

○：サポートしている

注※

これらのオプションは，自由形式正書法で書かれた COBOL 原始プログラムとしてコンパイルするプログラムには指定できません。指定してコンパイルすると，エラーとなってコンパイルが中止されます。

(8) 移行

他システムとの移行を設定するコンパイラオプションを，次に示します。

表中のコンパイラオプションの詳細については，「32.5.12 他システムとの移行の設定」を参照してください。

オプション	機能	OS	
		AIX	Linux
-CompatiV3※1※2 -noCompatiV3	VOS3 COBOL85 との互換機能を有効にする。	○	○
-Compati85｛,IoStatus｜,Linage｜,Call｜,Power｜,Syntax｜,IDParag｜,RsvWord｜,NoPropagate｜,All｝＋ -noCompati85	COBOL85 互換機能を有効にする。	○	○
-H8Switch -noH8Switch	HITAC8000 シリーズの仕様でコンパイルする。	○	○
-Cblctr -noCblctr	CBL-CTR 特殊レジスタを使用できるようにする。	○	○
-DigitsTrunc -noDigitsTrunc	転記文で上位けたを切り捨てる。	○	○
-IgnoreLCC -noIgnoreLCC	行送り制御文字を無視する。	○	○
-CmAster -noCmAster	1 カラム目が'*'の行を注記行とする。	○	○
-CmDol -noCmDol	7 カラム目が'$'の行を注記行とする。	○	○
-Comp5 -noComp5	COMP-5 を指定できるようにする。	○	○
-V3Spec〔,CopyEased〕 -noV3Spec	VOS3 COBOL85 に対する UNIX COBOL2002 固有の構文をチェックする。	○	○
-V3ConvtName -noV3ConvtName	VOS3 COBOL85 からのソースファイル互換のため，COPY 文の原文名定数中の'¥'と'@'を変換する。	○	○
-Switch，{EBCDIC ｜ EBCDIK}〔,Unprintable〕〔,noApplyJpnItem〕 -noSwitch	字類条件を指定する。	○	×
-V3Rec，{Fixed ｜ Variable} ※1※2 -noV3Rec	メインフレーム（VOS3）の固定長または可変長レコード形式のプログラムを，VOS3 の日本語文字の扱いに合わせてコンパイルする。	○	○
-V3RecFCSpace※1 -noV3RecFCSpace	空白に関する機能キャラクタの扱いをメインフレーム（VOS3）と同等にする。	○	○
-V3RecEased{,QuoteCheck¦,WordCheck}+ -noV3RecEased	-V3Rec オプション指定時の仕様チェックを緩和する。	○	○
-DoubleQuote	引用符（ " ）を分離符とみなしてコンパイルする。	○	○

オプション	機能	OS	
		AIX	Linux
-noDoubleQuote			
-BigEndian｛,Bin ｜,Float｝＋ -noBigEndian	2 進データ項目または浮動小数点データ項目をビッグエンディアン形式で処理する。	×	○
-International -noInternational	インタナショナリゼーション機能を使用する。	○	○
-EucPosition※1 -noEucPosition	EUC コード使用時，見かけ上の文字位置でコンパイルする。	○	×
-VOSCBL｛,OccursKey ｜,ReportControl ｜,DataComm ｜,RedefinesData ｜,AssignDataToDevice ｜,EvaluateWhenOther｝＋ -noVOSCBL	メインフレーム互換機能を有効にする。	○	○
-PortabilityCheck｛,Literal ｜,Numeric｝＋ -noPortabilityCheck	移行向けチェック機能を有効にする。	○	○
-IgnoreAPPLY,FILESHARE -noIgnoreAPPLY	APPLY FILE-SHARE 句を覚え書きとみなす。	○	○

（凡例）

○：サポートしている

×：サポートしていない

注※1

これらのオプションは，自由形式正書法で書かれた COBOL 原始プログラムとしてコンパイルするプログラムには指定できません。指定してコンパイルすると，エラーとなってコンパイルが中止されます。

注※2

Linux の場合，コンパイラ環境変数 CBLV3UNICODE に YES を指定したときだけ有効です。

(9) リスト出力

リスト出力の設定をするコンパイラオプションを，次に示します。

表中のコンパイラオプションの詳細については，「32.5.13 リスト出力の設定」を参照してください。

オプション	機能	OS	
		AIX	Linux
-SrcList,｛OutputAll ｜ CopyAll ｜ CopySup ｜ NoCopy｝〔,NoFalsePath〕〔,DataLoc〕 -noSrcList	コンパイルリストを出力する。	○	○

オプション	機能	OS	
		AIX	Linux
-ErrSup｛,I｜,W｝＋ -noErrSup	I レベルまたは W レベルエラーの出力を抑止する。	○	○

（凡例）

○：サポートしている

(10) その他

その他の設定をするコンパイラオプションを，次に示します。

表中のコンパイラオプションの詳細については，「32.5.14　その他の設定」を参照してください。

オプション	機能	OS			
		AIX(32)	AIX(64)	Linux(x86)	Linux(x64)
-Bin1Byte -noBin1Byte	1 バイトの 2 進項目を有効にする（PICTURE 句の指定で 2 けたまでは 1 バイトとして扱う）。	○	○	○	○
-JPN,｛Alnum ｜ V3JPN ｜ V3JPNSpace｝※1 -noJPN	日本語項目の扱いを指定する。	○※1	○※1	×	×
-EquivRule,｛NotExtend ｜ NotAny ｜ StdCode｝ -noEquivRule	文字の等価規則をどう変更するか指定する。	○	○	○	○
-UscoreStart -noUscoreStart	先頭が下線の CALL 定数を指定できるようにする。	○	○	○	○
-BinExtend -noBinExtend	2 進データ項目に指定できる初期値を拡張する。	○	○	○	○
-MinusZero -noMinusZero	10 進項目で負の符号を持つゼロを正の符号を持つゼロに変換する。	○	○	○	○
-TruncCheck〔,Binary〕 -noTruncCheck	転記でのデータ切り捨てをチェックする。 送り出し側作用対象が 2 進項目で，受け取り側作用対象が外部 10 進項目／内部 10 進項目のとき，2 進項目は格納できる最大けた数で転記のけた数をチェックする。	○	○	○	○

オプション	機能	OS			
		AIX(32)	AIX(64)	Linux(x86)	Linux(x64)
-LowerAsUpper -noLowerAsUpper	定数指定の CALL に指定された英小文字を英大文字に変換してプログラムを呼び出す。	○	○	○	○
-CBLVALUE -noCBLVALUE	環境変数 CBLVALUE を有効にする。	○	○	○	○
-Repository, {Gen | Sup} -noRepository	リポジトリファイルの生成時，強制的に出力するか，更新しないかを指定する。	○	○	○	○
-RepositoryCheck -noRepositoryCheck	同じソースファイル中の翻訳単位の定義と外部リポジトリ中の情報に相違があるかどうかをチェックする。	○	○	○	○
-Define 翻訳変数名〔=値〕〔,翻訳変数名〔=値〕〕… -noDefine	コンパイル時に有効となる，翻訳変数名とその値を定義する。	○	○	○	○
-Details -noDetails	コンパイラオプションの詳細情報を出力する。	○	○	○	○
-OldForm '旧オプションの並び'	UNIX COBOL85 のオプションを指定する。	○	○	○	○
-Help	ccbl2002 コマンドのヘルプを出力する。	○	○	○	○
-Profile, {Prof | Gprof} ※2 -noProfile	prof または gprof でのプロファイル用オブジェクトファイルを作成する。	○	○	○※2	○※2
-UniObjGen -noUniObjGen	シフト JIS の COBOL ソースから Unicode のオブジェクトを生成する。	○	○	○	○
-UniEndian, {Little | Big} -noUniEndian	シフト JIS で記述された日本語文字定数を UTF-16LE，または UTF-16BE に変換する。	○	○	○	○
-Lx64ConventionCheck -noLx64ConventionCheck	Linux(x64)で C 言語との連携を行うときに問題となる可能性のあるコードがある場合，チェックする。	×	×	×	○
-MaxDigits38	数字項目および数字定数に指定できる最大けた数を 18 けたから 38 けたに拡張する。	×	○	×	○

オプション	機能	OS			
		AIX(32)	AIX(64)	Linux(x86)	Linux(x64)
-IntResult,DecFloat40	算術演算の中間結果の表現形式を 40 けたの 10 進浮動小数点形式にする。	×	○	×	○
-LiteralExtend,Alnum -noLiteralExtend	英数字定数と定数指定のプログラム名の長さを拡張する。	○	○	○	○
-SpaceAsZero -noSpaceAsZero	外部 10 進項目中の空白に対して特殊処理を行う。	○	○	○	○
-CheckUninitData -noCheckUninitData	データ項目の初期化漏れをチェックする。	○	○	○	○
-FunctionECSup,CodeConvErr -noFunctionECSup	組み込み関数でコード変換エラーが発生しても例外条件を成立させない。	○	○	○	○

(凡例)

○：サポートしている

×：サポートしていない

注※1

次のオプションは，AIX で使用できます。

```
-JPN,V3JPNSpace
```

注※2

次のオプションは，Linux では使用できません。

```
-Profile,Prof
```

32.5.5　最終生成物の種類の設定

最終生成物※の種類を設定するコンパイラオプションについて，説明します。

注※

最終生成物とは，コンパイラが最終的に生成する実行可能ファイルのことを示します。

(1)　-Main オプション

(a)　形式

```
-Main, {System | V3}　ファイル名
```

(b)　機能

最外側のプログラムをアプリケーションの主プログラムとしてコンパイルします。

ファイル中に複数の最外側のプログラムがあるときは，先頭の最外側のプログラムを主プログラムとしてコンパイルします。

-Main,System ファイル名

最外側のプログラムをアプリケーションの主プログラムとしてコンパイルします。

このとき，主プログラムが制御プログラムから受け取る引数の形式を，システム固有の argc，argv 形式に合わせます。

-Main,V3 ファイル名

最外側のプログラムをアプリケーションの主プログラムとしてコンパイルします。

このとき，主プログラムが制御プログラムから受け取る引数の形式を，メインフレーム（VOS3）に合わせます。

(c) 注意事項

- -Main,System オプションまたは-Main,V3 オプションの指定がない場合で，次の条件をすべて満たすときには，先頭の COBOL ソースファイルに-Main,System オプションを指定したものと仮定されます。
  1. -Compile オプションの指定がなく，かつ，オブジェクトファイル（.o），C ソースファイル（.c），アセンブラソースファイル（.s）の指定がない。
  2. 環境変数 CBLCC の指定がない。
  3. -SimMain オプションの指定がない。
- -Main オプションと-MainNotCBL オプションを同時に指定した場合，-Main オプションが無効となります。

  （例）

  -MainNotCBL -Main,System

  と指定した場合，-MainNotCBL オプションが有効となり，-Main オプションは無効となります。
- -Main,System オプション，-Main,V3 オプションについては，「16.2　引数の受け取りと外部スイッチ」の記述も参照してください。
- このオプションは，直後に指定されたファイルをアプリケーションの主プログラムとして扱います。

```
ccbl2002 -Main,System main1.cbl sub1.cbl test1.o
                      主ﾌﾟﾛｸﾞﾗﾑ   副ﾌﾟﾛｸﾞﾗﾑ
```

- ENTRY 文の記述があるソースファイルに，このオプションを指定しても ENTRY 文に制御は渡りません。
- -Main,V3 オプションを指定して作成した実行可能ファイルを実行するとき，受け取れるコマンド引数は一つ（空白まで）です。コマンド引数に空白を含むときは，アポストロフィ（'）で囲みます。

(2) -PIC オプション

(a) 形式

```
-PIC, {Std | Expand}
-noPIC
```

(b) 機能

-PIC,Std

共用ライブラリに使う位置独立（PIC）コードを作成する場合に指定します。

-PIC,Expand

-PIC,Std を指定した場合と同じコードを生成します。

-noPIC

-PIC オプションの指定を打ち消します。

(c) 注意事項

- -PIC オプションを指定した場合，リンクできるオブジェクトファイル（.o）が作成されます（-Compile,NoLink が仮定される）。
- -PIC オプションは，次のオプションと相反しています。同時に指定した場合は，このオプションを優先します。

  -Profile オプション
- -PIC オプションは，共用ライブラリ化する COBOL オブジェクトファイル作成のために指定します。実行可能ファイルにリンクする COBOL オブジェクトファイルや，アーカイブファイル化する COBOL オブジェクトファイルを作成する場合は，これらのオプションを指定しないでください。指定した場合，関数の呼び出し元への戻り制御が正常に動作しない場合があります。
- C コンパイラにも，位置独立コード（PIC）を作成するオプションがあります。Cコンパイラのオプションについては，システムのマニュアルを参照してください。
- 共用ライブラリは，ld コマンドで作成します。次に，共用ライブラリの作成例を示します。

（例）

オブジェクトファイル名：xxx.o

AIX(32)の場合（共用ライブラリ名：libucbl.a）

```
ld -o libucbl.a xxx.o -bpT:0x10000000
   -bpD:0x20000000 -bnoentry -bM:SRE -bE:libucbl.exp
   -L/opt/HILNGcbl2k/lib -lcbl2k -lcbl2kml -lm -lc
```

AIX(64)の場合（共用ライブラリ名：libucbl.a）

```
ld -o libucbl.a a.o b.o -b64 -bpT:0x100000000
   -bpD:0x110000000 -bnoentry -bM:SRE -bE:libucbl.exp
   -L/opt/HILNGcbl2k64/lib -lcbl2k64 -lcbl2kml64 -lm -lc
```

Linux(x86)の場合（共用ライブラリ名：libucbl.so）

```
ld -E xxx.o -o libucbl.so -shared -L/opt/HILNGcbl2k/lib -lcbl2kml
```

Linux(x64)の場合（共用ライブラリ名：libucbl.so）

```
ld -E xxx.o -o libucbl.so -shared -L/opt/HILNGcbl2k64/lib -lcbl2kml
```

オブジェクトファイルは複数指定できます。

32.5.6　他製品との連携の設定

他製品との連携を設定するコンパイラオプションについて，説明します。

(1)　-SQL オプション

(a)　形式

```
-SQL, {XDM | ODBC 〔,NoCont〕 }
-noSQL
```

(b)　機能

-SQL,XDM

リレーショナルデータベース（XDM/RD）操作シミュレーション機能を使用する場合に指定します。このオプションを指定すると，VOS3 XDM/RD の SQL は覚え書きとみなされます。

-SQL,ODBC（Linux で有効）

埋め込み SQL 文を ODBC インタフェース機能で実行する場合に指定します。このオプションを指定すると，埋め込み SQL 文を使用して，ODBC インタフェース機能でデータベースにアクセスします。

-SQL,ODBC,NoCont（Linux で有効）

埋め込み SQL 文を ODBC インタフェース機能で実行する場合で，SQL 構文内の浮動継続指示子を有効としないときに指定します。

-noSQL

-SQL オプションの指定を打ち消します。

(c) 注意事項

- Linux では，-SQL,XDM オプションと-SQL,ODBC オプションを同時に指定した場合，あとに指定したオプションが有効となります。

(2) -SQLDisp オプション（Linux で有効）

(a) 形式

```
-SQLDisp
-noSQLDisp
```

(b) 機能

-SQLDisp

埋め込み SQL 文に，用途が表示用のデータ項目を指定できるようにします。

このオプションを指定すると，データベースアクセス機能で，埋め込み変数の定義に次に示すデータ記述項を指定できます。埋め込み SQL 文では，これらの項目が占めるバイト数分と同じバイト数を占める，英数字項目の埋め込み変数として使用できます。

- 英字項目
- 〔SIGN IS〕 LEADING SEPARATE CHARACTER の指定がない外部 10 進形式の数字項目
- 英数字編集項目
- 数字編集項目
- 日本語項目
- 日本語編集項目
- 外部ブール項目
- 外部浮動小数点項目

-noSQLDisp

-SQLDisp オプションの指定を打ち消します。

(3) -RDBTran オプション（AIX で有効）

(a) 形式

```
-RDBTran
-noRDBTran
```

(b)　機能

-RDBTran

翻訳単位中に記述されたCOMMIT文，ROLLBACK文を，HiRDBによる索引編成ファイルに対して適用します。また，HiRDBによる索引編成ファイル入出力機能に対するライブラリのリンクを行います。詳細は，「6.9.6　プログラムのコンパイルと実行」を参照してください。

なお，このオプションを指定するとデータコミュニケーション（DC）機能，およびDCシミュレーション機能は使用できません。

また，このオプションを指定しないと，COMMIT文，ROLLBACK文は，DC機能に対して適用されます。

-noRDBTran

-RDBTranオプションの指定を打ち消します。

(4)　-XMAPオプション（AIX(32)で有効）

(a)　形式

```
-XMAP,LinePrint
-noXMAP
```

(b)　機能

-XMAP,LinePrint

XMAP3を使用して，順編成ファイルをプリンタに出力するときに指定します。このとき，次の印刷機能も使用できます。

- 書式オーバレイ（APPLY FORMS-OVERLAY句）
- 印刷制御付き（CHARACTER TYPE句）

COBOLプログラム中にAPPLY FORMS-OVERLAY句またはCHARACTER TYPE句があっても，-XMAP,LinePrintオプションの指定がなければ，これらの句は覚え書きとなります。

これらの印刷機能の詳細については，「8.　プリンタへのアクセス」の該当する節を参照してください。

-noXMAP

-XMAPオプションの指定を打ち消します。

32.5.7　実行の設定

実行の設定をするコンパイラオプションについて，説明します。

(1)　-NumAccept オプション

(a)　形式

```
-NumAccept
-noNumAccept
```

(b)　機能

-NumAccept

ACCEPT 文の一意名 1 に次の項目を指定できるようにします。

- 符号なし数字項目
- 符号あり数字項目
- 数字編集項目
- 外部浮動小数点数字項目
- 内部浮動小数点数字項目

数字項目を指定した場合，入力データは右詰めで格納され，残りの部分には左側に 0 が埋められます。数字編集項目を指定した場合，編集文字に合わせて右詰めで格納されます。浮動小数点数字項目を指定した場合，受け取りの有効けた数分の値が格納されます。

-noNumAccept

-NumAccept オプションの指定を打ち消します。

(2)　-NumCsv オプション

(a)　形式

```
-NumCsv
-noNumCsv
```

(b)　機能

-NumCsv

CSV 編成ファイルで，セルのデータを数値として入出力したい場合に指定するオプションです。詳細は，「6.8.5　セルデータを数値として入出力する機能」を参照してください。

-noNumCsv

-NumCsv オプションの指定を打ち消します。

(3) -MultiThread オプション

(a) 形式

```
-MultiThread
-noMultiThread
```

(b) 機能

-MultiThread

COBOL プログラムをマルチスレッド環境下で動作させたい場合に指定します。

詳細は，「26.2 マルチスレッド対応 COBOL プログラムの生成」を参照してください。

-noMultiThread

-MultiThread オプションの指定を打ち消します。

(c) 注意事項

- -MultiThread オプションと-Profile オプションを同時に指定した場合，-Profile オプションは無効となります。

(4) -MainNotCBL オプション

(a) 形式

```
-MainNotCBL
-noMainNotCBL
```

(b) 機能

-MainNotCBL

実行中のプロセスで最初に呼び出される COBOL プログラムを，副プログラムとします。他言語で作成したアプリケーションのメインプログラムから COBOL プログラムを呼び出す場合，このオプションを指定します。

-noMainNotCBL

-MainNotCBL オプションの指定を打ち消します。

(c) 注意事項

- -Main オプションと-MainNotCBL オプションを同時に指定した場合，-MainNotCBL オプションが有効となります。

32.5.8　プログラムの最適化の設定

プログラムの最適化を設定するコンパイラオプションについて，説明します。

(1)　-Optimize オプション

(a)　形式

```
-Optimize, {0 | 1 | 2 | 3}
```

(b)　機能

最適化のレベルを指定します。-Optimize オプションの詳細については，「31.　最適化機能」を参照してください。

-Optimize,0

最適化しません。

-Optimize,1

文の中で閉じた次の最適化をします。

- 命令レベルでの最適化（ピープホールによる不要命令の削除など）

-Optimize オプションを指定しなかった場合，このオプションが仮定されます。

-Optimize,2

広域的な次の最適化をします。

- 命令レベルでの最適化（ピープホールによる不要命令の削除など）
- 不変式のループ外への移動
- コピー伝播
- 定数の畳み込み
- 共通式の削除
- 演算の強さの軽減
- そと PERFORM 文のインライン展開

-Optimize,3

-Optimize,2 オプションでの最適化に加えて，10 進項目を 2 進項目に変換します。

(c)　注意事項

- 最適化オプションを指定すると，データ項目の使用状況の解析や文の順番の入れ替えなどをするため，コンパイル時間が増加します。-Optimize,0 指定時が最短で，-Optimize,3 指定時が最長となります。
- -Optimize,3 オプションと-TDInf オプションを同時に指定すると，-Optimize,3 オプションが無効になり，-Optimize,2 オプションが仮定されます。

- 最適化の内容については，「31. 最適化機能」を参照してください。
- -Optimize,3 オプションを指定すると 10 進項目を 2 進項目化するため，アドレス操作機能を使用しているプログラムは正しく動作しないことがあります。
- AIX(64)，Linux(x64)の場合，-Optimize,3 オプション，-MaxDigits38 オプション，および-IntResult,DecFloat40 オプションを同時に指定したときは，-Optimize,3 オプションは無効となり，-Optimize,2 オプションが仮定されます。

32.5.9 デバッグの設定

デバッグを設定するコンパイラオプションについて，説明します。

(1) -DebugLine オプション

(a) 形式

```
-DebugLine
-noDebugLine
```

(b) 機能

-DebugLine

原始プログラム中のデバッグ行をコンパイルの対象とします。デバッグ行とは，原始プログラムの標識領域にデバッグ標識（「D」または「d」）が記述されているか，浮動デバッグ指示子（「>>D」）が記述されている行のことです。

このシステムでは，WITH DEBUGGING MODE 句は覚え書きとみなされるので，デバッグ行を有効にするためにはこのオプションの指定が必要です。

-noDebugLine

-DebugLine オプションの指定を打ち消します。

(2) -DebugInf オプション

(a) 形式

```
-DebugInf [,Trace]
-noDebugInf
```

(b) 機能

-DebugInf

実行時に異常終了した場合，または実行時エラーが発生した場合，異常終了時要約情報リストを出力します。異常終了時要約情報リストの詳細については，「36.2 異常終了時要約情報リスト」を参照してください。

-DebugInf,Trace

異常終了時要約情報リスト中にトレースバック情報を出力します。

-noDebugInf

-DebugInf オプションの指定を打ち消します。

(c) 注意事項

- -DebugInf オプション，-TDInf オプション，-CVInf オプション，-DebugRange オプション，-DebugCompati オプション，-DebugData オプション，および-TestCmd オプションを指定していない場合，エラー／デバッグ情報（異常終了時要約情報リストやデータ領域ダンプリスト）が出力されません。
- -DebugInf オプションと-Compile,CheckOnly オプションを同時に指定した場合，-DebugInf オプションが無効となります。
- -DebugInf,Trace オプションを指定すると，実行時の処理速度が遅くなる可能性があります。このため，-DebugInf,Trace オプションは，デバッグの目的でコンパイルするときだけ指定してください。

(3) -DebugCompati オプション

(a) 形式

```
-DebugCompati
-noDebugCompati
```

(b) 機能

-DebugCompati

次のチェックをします。なお，表要素または部分参照された一意名に OCCURS DEPENDINGON の指定がある場合は，実行時点の制御変数の値を用いてチェックします。

- 表操作で使用する添字または指標名が指す表要素が表の範囲内であるかどうか。
- 部分参照の指定が一意名の範囲内であるかどうか。
- プログラム間で整合性が取れているかどうか。

ただし，実行時環境変数 CBLPRMCHKW に NOCHK を指定したときは，プログラム間で整合性が取れていなくてもエラーとしないで，実行を継続します。

詳細は，「36.5 プログラム間整合性チェック」を参照してください。

-noDebugCompati

-DebugCompati オプションの指定を打ち消します。

(c) 注意事項

- -DebugCompati オプションと-Compile,CheckOnly オプションを同時に指定した場合，-DebugCompati オプションが無効となります。
- -DebugCompati オプションを指定すると，-DebugInf オプションが仮定されます。
- -DebugCompati オプションを指定すると，実行時の処理速度が遅くなる可能性があります。このため，-DebugCompati オプションは，デバッグの目的でコンパイルするときだけ指定してください。
- -DebugCompati オプションを指定すると，一部の例外名に対する TURN 指令が無効となります。詳細は「21.8.2　例外検出での注意事項」の「(4)　コンパイラオプションとの関連性」を参照してください。

(4) -DebugData オプション

(a) 形式

```
-DebugData [,ValueHex]
-noDebugData
```

(b) 機能

-DebugData

外部または内部 10 進項目に対して，格納値がデータ項目の属性と矛盾している場合，データ例外エラーを検出します。

詳細は，「36.7　データ例外検出機能」を参照してください。

-DebugData,ValueHex

データ例外検知機能でデータ例外が検出された場合，出力されるエラーメッセージにデータ項目の属性と矛盾している格納値を 16 進数で表示します。

詳細は，「36.7　データ例外検出機能」を参照してください。

-noDebugData

-DebugData オプションの指定を打ち消します。

(c) 注意事項

- -DebugData オプションと-Compile,CheckOnly オプションを同時に指定した場合，-DebugData オプションが無効となります。
- -DebugData オプションを指定すると，-DebugInf オプションが仮定されます。
- -DebugData オプションを指定すると，実行時の処理速度が遅くなる可能性があります。このため，-DebugData オプションは，デバッグの目的でコンパイルするときだけ指定してください。

- ValueHex サブオプションを指定した場合，-noDebugData オプションの指定がないかぎり，ValueHex サブオプションの指定は有効となります。ValueHex サブオプションの指定を打ち消すときは，-noDebugData オプションを指定してください。

  （例）

```
ccbl2002 -DebugData -DebugData,ValueHex -DebugData …
 （コンパイル結果）
  -DebugData,ValueHex

ccbl2002 -DebugData,ValueHex -noDebugData -DebugData …
 （コンパイル結果）
  -DebugData
```

(5) -TDInf オプション

(a) 形式

```
-TDInf
-noTDInf
```

(b) 機能

-TDInf

デバッグ情報を含むプログラム情報ファイル（.cbp）を出力します。

-TDInf オプションを指定したプログラムをテストデバッガでデバッグする場合，-DebugCompati オプションの指定の有無に関係なく，プログラム間の整合性がチェックされます。プログラム間の整合性チェックについては，「(3) -DebugCompati オプション」を参照してください。

-noTDInf

-TDInf オプションの指定を打ち消します。

(c) 注意事項

- -TDInf オプションと-Compile,CheckOnly オプションを同時に指定した場合，-TDInf オプションが無効となります。
- -TDInf オプションを指定すると，-DebugInf オプションが仮定されます。
- -TDInf オプションと-Optimize,3 オプションを同時に指定すると，-Optimize,3 オプションが無効になり，-Optimize,2 オプションが仮定されます。
- コンパイルとリンクを別々に実行する場合で，-TDInf オプションを指定してコンパイルしたときは，ccbl2002 コマンドでリンクを実行するときにも，同じオプションを指定する必要があります。
- -TDInf オプションを指定すると，実行時の処理速度が遅くなる可能性があります。このため，-TDInf オプションは，デバッグの目的でコンパイルするときだけ指定してください。
- -TDInf オプションはテストデバッガのためにゼロによる除算チェックなど最適化に影響を与えるオブジェクトコードを生成します。これによって，-TDInf オプションを指定することで，異常終了時要約

リストなどの行番号／欄の情報が変化することがあります。行番号／欄の情報を常に正しく保つためには，-Optimize,0 オプションを指定する必要があります。

(6) -CVInf オプション

(a) 形式

```
-CVInf
-noCVInf
```

(b) 機能

-CVInf

カバレージ情報を含むプログラム情報ファイル（.cbp）を出力します。

-noCVInf

-CVInf オプションの指定を打ち消します。

(c) 注意事項

- -CVInf オプションと-Compile,CheckOnly オプションを同時に指定した場合，-CVInf オプションが無効となります。
- -CVInf オプションを指定すると，-DebugInf オプションが仮定されます。
- コンパイルとリンクを別々に実行する場合で，-CVInf オプションを指定してコンパイルしたときは，ccbl2002 コマンドでリンクを実行するときにも，同じオプションを指定する必要があります。
- -CVInf オプションを指定すると，実行時の処理速度が遅くなる可能性があります。このため，-CVInf オプションは，デバッグの目的でコンパイルするときだけ指定してください。
- -CVInf オプションを指定してカバレージ情報を出力できるのは，ccbl2002 コマンドでコンパイルして出力したプログラム情報ファイルだけです。COBOL85 で作成したプログラム情報ファイルを使用した場合，ccbl2002 コマンドに-CVInf オプションを指定してコンパイルしても，プログラム情報ファイルにカバレージ情報を出力しません。

(7) -DebugRange オプション

(a) 形式

```
-DebugRange
-noDebugRange
```

(b) 機能

-DebugRange

表操作で使用する添字および指標名の値が，次元ごとに OCCURS 句で指定した繰り返し回数の範囲内であるかどうかを，実行時に調べます。

なお，表要素に OCCURS DEPENDING ON の指定がある場合は，実行時点の制御変数の値を用いて調べます。

-noDebugRange

-DebugRange オプションの指定を打ち消します。

(c) 注意事項

- -DebugRange オプションと-Compile,CheckOnly オプションを同時に指定した場合，-DebugRange オプションが無効となります。
- -DebugRange オプションを指定すると，-DebugInf オプションおよび-DebugCompati オプションが仮定されます。
- -DebugRange オプションを指定すると，実行時の処理速度が遅くなる可能性があります。このため，-DebugRange オプションは，デバッグの目的でコンパイルするときだけ指定してください。
- -DebugRange オプションを指定すると，一部の例外名に対する TURN 指令が無効となります。詳細は「21.8.2　例外検出での注意事項」の「(4)　コンパイラオプションとの関連性」を参照してください。

(8) -TestCmd オプション

(a) 形式

```
-TestCmd {,Full|,Break|,Sim} +
-noTestCmd
```

(b) 機能

TD コマンド格納ファイルに出力する情報の種類を指定します。

-TestCmd,Full

中断点情報，シミュレーション情報の TD コマンドを TD コマンド格納ファイル（.tdi）に出力します。

-TestCmd,Break

中断点情報の TD コマンドを TD コマンド格納ファイル（.tdi）に出力します。

-TestCmd,Sim

シミュレーション情報の TD コマンドを TD コマンド格納ファイル（.tds）に出力します。

-noTestCmd

-TestCmd オプションの指定を打ち消します。

(c) 注意事項

- -TestCmd,Full オプションと-TestCmd,Break オプション，または-TestCmd,Full オプションと-TestCmd,Sim オプションを重複して指定した場合，-TestCmd,Full オプションが有効となります。
- -TestCmd オプションを指定すると，-DebugInf および-TDInf オプションが仮定されます。
- コンパイルとリンクを別々にする場合で，コンパイル時に-TestCmd オプションを指定したときは，ccbl2002 コマンドでのリンク時に-TDInf オプションを指定する必要があります。

(9) -SimMain オプション

(a) 形式

```
-SimMain プログラム名
-noSimMain
```

(b) 機能

-SimMain プログラム名

テストデバッガの主プログラムシミュレーション機能を使用します。

プログラム名には，主プログラムシミュレーションの対象となる副プログラムの名称（最外側のプログラムのプログラム名段落で指定した名称）を指定します。

このオプションを指定すると，主プログラムシミュレーション用の一時的な COBOL ソースファイル（擬似主プログラム）が内部的に生成され，コンパイルされます。コンパイルによって生成されたオブジェクトファイルは，リンク時に取り込まれます。ここで生成されたソースファイル，およびオブジェクトファイルは，コンパイル終了時に消去されます。

また，テストデバッガで必要な擬似主プログラム用のプログラム情報ファイル（.cbs）が生成されます。テストデバッガの主プログラムシミュレーション機能を使用するとき，生成されたプログラム情報ファイル（.cbs）をテストデバッガ環境に指定してください。生成するプログラム情報ファイルの名称については，「(12) 主／副プログラムシミュレーションで生成する擬似プログラム用プログラム情報ファイル」を参照してください。

擬似主プログラムを生成する COBOL ソースファイルの情報は，プログラム情報ファイル（.cbp）から取得されます。このため，このオプションを指定する場合は，次のどちらかをする必要があります。

- -TDInf オプションを同時に指定してコンパイルする。
- 副プログラムを含む COBOL ソースファイルを-TDInf オプションを指定してコンパイルし，生成された.cbp ファイルを，環境変数 CBLPIDIR で指定したディレクトリまたはカレントディレクトリに入れておく。

-OutputFile オプションで実行可能ファイル名を指定しないと，生成される主プログラムの実行可能ファイル名は a.out となります。

-noSimMain

-SimMain オプションの指定を打ち消します。

(c) 注意事項

- 下記に示すデータ項目を主プログラムシミュレーションの対象となる副プログラムの手続き部見出しに指定することはできません。
  - プロパティ項目
  - 関数一意名
- -SimMain オプションを指定した場合，コンパイラのバージョンが 03-01 未満で作成したプログラム情報ファイル（.cbp）は使用できません。使用した場合は，コンパイルエラーとなり，コンパイルを中止します。
- 共用ライブラリやアーカイブライブラリを使用しての主プログラムシミュレーションはできません。
- -LiteralExtend オプションと同時に指定した場合は，-LiteralExtend オプションが無効となります。

(10) -SimSub オプション

(a) 形式

```
-SimSub プログラム名〔,…〕
-noSimSub
```

(b) 機能

-SimSub プログラム名〔,…〕

テストデバッガの副プログラムシミュレーション機能を使用します。

プログラム名には，副プログラムシミュレーションの対象となる副プログラムの名称（CALL 文で指定した名称）を指定します。ただし，副プログラムの呼び出し方法は，CALL 文の定数指定呼び出しでなければなりません。

このオプションを指定すると，副プログラムシミュレーション用の一時的な COBOL ソースファイル（擬似副プログラム）が内部的に生成され，コンパイルされます。コンパイルによって生成されたオブジェクトファイルは，リンク時に取り込まれます。ここで生成されたソースファイル，およびオブジェクトファイルは，コンパイル終了時に消去されます。

また，テストデバッガで必要な擬似副プログラム用のプログラム情報ファイル（.cbs）が生成されます。テストデバッガの副プログラムシミュレーション機能を使用するとき，生成されたプログラム情報ファイル（.cbs）をテストデバッガ環境に指定してください。生成するプログラム情報ファイルの名称については，「(12) 主／副プログラムシミュレーションで生成する擬似プログラム用プログラム情報ファイル」を参照してください。

擬似副プログラムを生成する COBOL ソースファイルの情報は，プログラム情報ファイル（.cbp）から取得されます。このため，このオプションを指定する場合は，次のどちらかをする必要があります。

- -TDInf オプションを同時に指定してコンパイルする。

- 副プログラムを呼び出している COBOL ソースファイルを-TDInf オプションを指定してコンパイルし，生成された.cbp ファイルを，環境変数 CBLPIDIR で指定したディレクトリまたはカレントディレクトリに入れておく。

-noSimSub

-SimSub オプションの指定を打ち消します。

(c) 注意事項

- 下記に示すデータ項目を副プログラムシミュレーションの対象となるプログラムに指定した CALL 文の引数に指定することはできません。
  - プロパティ項目
  - 関数一意名
- -SimSub オプションを指定した場合，コンパイラのバージョンが 03-01 未満で作成したプログラム情報ファイル（.cbp）は使用できません。使用した場合は，コンパイルエラーとなり，コンパイルを中止します。
- 共用ライブラリやアーカイブライブラリを使用しての副プログラムシミュレーションはできません。
- -LiteralExtend オプションと同時に指定した場合は，-LiteralExtend オプションが無効となります。

(11) -SimIdent オプション

(a) 形式

```
-SimIdent
-noSimIdent
```

(b) 機能

-SimIdent

テストデバッガの副プログラムシミュレーション機能を使用します。この場合，副プログラムシミュレーションの対象となるのは，一意名呼び出しの CALL 文を指定したプログラムです。

このオプションを指定すると，副プログラムシミュレーション用の一時的な COBOL ソースファイルが内部的に生成され，コンパイルされます。コンパイルによって生成されたオブジェクトファイルは，リンク時に取り込まれます。ここで生成されたソースファイル，およびオブジェクトファイルは，コンパイル終了時に消去されます。

また，テストデバッガで必要な擬似副プログラム用のプログラム情報ファイル（.cbs）が生成されます。テストデバッガの副プログラムシミュレーション機能を使用するとき，生成されたプログラム情報ファイル（.cbs）をテストデバッガ環境に指定してください。生成するプログラム情報ファイルの名称については，「(12) 主／副プログラムシミュレーションで生成する擬似プログラム用プログラム情報ファイル」を参照してください。

擬似副プログラムを生成する COBOL ソースファイルの情報は，プログラム情報ファイル（.cbp）から取得されます。このため，このオプションを指定する場合は，次のどちらかをする必要があります。

- -TDInf オプションを同時に指定してコンパイルする。
- 副プログラムを呼び出している COBOL ソースファイルを-TDInf オプションを指定してコンパイルし，生成された.cbp ファイルを，環境変数 CBLPIDIR で指定したディレクトリまたはカレントディレクトリに入れておく。

注意事項

- 下記に示すデータ項目を副プログラムシミュレーションの対象となるプログラムに指定した CALL 文の引数に指定することはできません。
  ・プロパティ項目
  ・関数一意名
- 共用ライブラリやアーカイブライブラリを使用しての副プログラムシミュレーションはできません。

-noSimIdent

-SimIdent オプションの指定を打ち消します。

(c) 注意事項

- 下記に示すデータ項目を副プログラムシミュレーションの対象となるプログラムに指定した CALL 文の引数に指定することはできません。
  - プロパティ項目
  - 関数一意名
- -SimIdent オプションを指定した場合，コンパイラのバージョンが 03-01 未満で作成したプログラム情報ファイル（.cbp）は使用できません。使用した場合は，コンパイルエラーとなり，コンパイルを中止します。
- 共用ライブラリやアーカイブライブラリを使用しての副プログラムシミュレーションはできません。
- -LiteralExtend オプションと同時に指定した場合は，-LiteralExtend オプションが無効となります。

(12) 主／副プログラムシミュレーションで生成する擬似プログラム用プログラム情報ファイル

主／副プログラムシミュレーションでは，テストデバッガで必要な情報である擬似プログラム用プログラム情報ファイル（.cbs）が生成されます。擬似プログラム用プログラム情報ファイル（.cbs）は，カレントディレクトリ，または環境変数 CBLPIDIR で指定したディレクトリに出力されます。

生成される擬似プログラム用プログラム情報ファイル（.cbs）の名称規則を，次に示します。

- -SimMain オプションによって，擬似主プログラムシミュレーションする場合
  SimMain@（-SimMain オプションで指定したプログラム名）.cbs
- -SimSub オプションによって，定数指定の CALL 文の擬似副プログラムシミュレーションする場合
  SimSub@（-SimSub オプションで指定したプログラム名※）.cbs

注※

プログラム名を複数指定した場合，先頭のプログラム名が使用されます。

- -SimIdent オプションによって，一意名指定の CALL 文の擬似副プログラムシミュレーションする場合
  SimIdent@001.cbs

32.5.10　リンクの設定

リンクを設定するコンパイラオプションについて，説明します。

(1)　-Compile オプション

(a)　形式

```
-Compile, {CheckOnly | NoLink}
-noCompile
```

(b)　機能

プログラムのコンパイル時，オブジェクトファイル（.o），および実行可能ファイルの生成を抑止します。

-Compile,CheckOnly

このオプションを指定すると，実行可能ファイルを生成しないため，コンパイル速度が向上します。単にエラーチェックだけをしたい場合に指定します。

-Compile,NoLink

オブジェクトファイルは出力しますが，実行可能ファイルは生成しません。出力したオブジェクトファイルは，あとでリンカを使用して実行可能ファイルにできます。

-noCompile

-Compile オプションの指定を打ち消します。

(c)　注意事項

- -Compile オプションを指定した場合，-OutputFile オプションの指定は無効となります。
  また，-Compile,CheckOnly オプションを指定した場合，アプリケーションデバッグ機能に関連するオプション（-DebugInf，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange，または-TestCmd オプション）が指定されても，無効となります。
- -Compile,CheckOnly オプションを指定した場合，-PIC オプション，-Profile オプション，-IdentCall オプションが指定されても無効となります。

(2) -OutputFile オプション

(a) 形式

```
-OutputFile ファイル名
-noOutputFile
```

(b) 機能

-OutputFile ファイル名

生成する実行可能ファイルの名称を指定します。

ファイル名には，生成する実行可能ファイルの名称を指定します。

このオプションの指定がない場合，実行可能ファイルの名称は，a.out となります。

実行可能ファイルをカレントディレクトリに出力する場合の指定例を，次に示します。

(例)

```
ccbl2002 -OutputFile TEST1 -Main,System source.cbl
```

実行可能ファイルは，TEST1 という名称で出力されます。

実行可能ファイルを任意のディレクトリに出力する場合の指定例を，次に示します。

(例)

```
ccbl2002 -Main,System TEST1.cbl -OutputFile /usr/user1/TEST1.out
```

実行可能ファイルは，/usr/user1 ディレクトリに TEST1.out という名称で出力されます。パスプレフィックスの指定が省略された場合，実行可能ファイルは，カレントディレクトリに出力されます。

-OutputFile オプションと-Compile オプションを同時に指定した場合，-OutputFile オプションは，無効となります。

-noOutputFile

-OutputFile オプションの指定を打ち消します。

(3) -Link オプション

(a) 形式

```
-Link オプションの並び
-noLink
```

(b) 機能

-Link オプションの並び

リンカに渡すオプションの並びを指定します。オプションとオプションの間は，空白で区切ります。

-noLink

-Link オプションの指定を打ち消します。

(c) 注意事項

- -Link オプション以降の文字は，ほかのオプション名，ファイル名なども含めすべてリンカに渡されます。したがって，このオプションは最後に指定してください。
- -Link オプションは，-Details オプションでのコンパイラオプションの詳細情報表示の対象にはなりません。
- -Link オプションに指定した文字列は，そのまま cc コマンドに渡されるため，指定するリンカオプションは，cc コマンドのオプションとなります。ただし，OS が AIX で C コンパイラが組み込まれていない場合は，ld コマンドが直接起動されます。この場合，-Link オプション以降の文字列には，ld コマンドのオプションを指定してください。

(4) -DynamicLink オプション

(a) 形式

```
-DynamicLink, {Call | IdentCall}
-noDynamicLink
```

(b) 機能

-DynamicLink,Call

プログラムを呼び出す場合，呼び出されるプログラムが静的にリンクされていない場合，動的なリンク（ダイナミックリンク）が設定されます。詳細は「18. プログラムの呼び出し」を参照してください。

-DynamicLink,IdentCall

一意名指定の CALL 文でプログラムを呼び出す場合，動的なリンク（ダイナミックリンク）を行います。定数指定の CALL 文は，静的なリンク（スタティックリンク）となります。詳細は「18. プログラムの呼び出し」を参照してください。

-noDynamicLink

-DynamicLink オプションの指定を打ち消します。

(5) -OldLinkOpt,GCBypass オプション（AIX で有効）

(a) 形式

```
-OldLinkOpt,GCBypass
-noOldLinkOpt
```

(b) 機能

-OldLinkOpt,GCBypass

翻訳とリンクを同時に行う場合，静的な参照関係のないオブジェクトをリンク対象とするため，ccbl2002コマンドで-bgcbypass リンカオプションを仮定します。

-noOldLinkOpt

-OldLinkOpt オプションの指定を打ち消します。

(c) 注意事項

- 動的なリンクによるプログラム呼び出しを使用している場合，COBOL2002 V4 より前に作成したCOBOL オブジェクトや C プログラムのオブジェクト中のプログラムを呼び出し先に指定すると，参照関係のないオブジェクトがリンク対象とならないでプログラムの呼び出しに失敗することがあります。

  COBOL2002 V4 より前に作成した COBOL オブジェクトや C プログラムのオブジェクトを混在させる場合は，このオプションを指定してください。

- このオプションの指定によって，-bgcbypass リンカオプションの対象を適切にするためには，ccbl2002コマンドで，COBOL ソース，オブジェクトファイル，アーカイブファイル，-l リンカオプションで指定するライブラリの順に指定する必要があります。

  -bgcbypass リンカオプションの対象が適切になる例

  ```
  ccbl2002 -OldLinkOpt,GCBypass -OutputFile TEST01 -Main,System TEST01.CBL
  TEST02.o libTEST03.a -L/TEST/LIBRRY -lTEST04
  ```

  -bgcbypass リンカオプションの対象が適切にならない例

  ```
  ccbl2002 -OldLinkOpt,GCBypass -OutputFile TEST01 -Main,System TEST01.CBL
  libTEST03.a -L/TEST/LIBRRY -lTEST04 TEST02.o
  ```

- 翻訳とリンクを別に行う場合は，リンク（cc/ld コマンド）時に-bgcbypass リンカオプションを指定する必要があります。

(6) -OldStyleObject オプション（AIX で有効）

(a) 形式

```
-OldStyleObject
-noOldStyleObject
```

(b) 機能

-OldStyleObject

AIX システムで COBOL2002 が生成するオブジェクトの形式を，AIX COBOL2002 V3 以前と同様の形式※に戻します。

注※

- AIX COBOL2002 V3 以前のコンパイラが生成するオブジェクトの形式の場合

  実行可能ファイルをリンクする際に，この形式のオブジェクトファイルを指定した場合，AIX システムのリンカは，特にリンカオプションの指定がないと，静的な参照関係がないオブジェクトファイルをリンク対象としません。

- AIX COBOL2002 V4 のコンパイラが生成するオブジェクトの形式の場合

  実行可能ファイルをリンクする際に，この形式のオブジェクトファイルを指定した場合，AIX システムのリンカは，静的な参照関係がないオブジェクトファイルでもリンク対象とします。

-noOldStyleObject

-OldStyleObject オプションの指定を打ち消します。

(c) 注意事項

- -OldStyleObject オプションを指定しないで作成したオブジェクトファイルを使用して実行可能ファイルまたは共用ライブラリを作成する場合に，次の条件をすべて満たすときは，実行可能ファイルまたは共用ライブラリのサイズが，AIX COBOL2002 V3 以前に作成したもののサイズと比べて，極端に肥大化することがあります。

  1. アーカイブファイルを指定して，実行可能ファイルまたは共用ライブラリを作成している
  2. 1.のアーカイブファイルに，-OldStyleObject オプションを指定しないで作成したオブジェクトファイルが含まれている
  3. 2.のオブジェクトファイルのプログラムに，メインプログラムまたはメインプログラムから直接的または間接的に呼び出されるプログラム中で宣言されている EXTERNAL データ項目と，同じ名称の EXTERNAL データ項目が宣言されている

  サイズの肥大化を回避する場合，2.のオブジェクトファイルは，-OldStyleObject オプションを指定して作成したオブジェクトファイルとしてください。

- 実行可能ファイルを作成するとき，このオプションを指定して作成した COBOL プログラムのオブジェクトファイルはリンク時に直接指定しても，静的な参照関係がないとリンク対象となりません。静的な参照関係がないオブジェクトファイルをリンク対象とする場合は，次のどれかのオプションを指定してください。

  - -bgcbypass リンカオプション（-bgcbypass:入力ファイル数）
  - -bkeepfile リンカオプション（-bkeepfile:入力ファイル名）
  - -u リンカオプション（-u プログラム名）
  - -OldLinkOpt,GCBypass オプション（ccbl2002 コマンドでリンクする場合だけ）

    -OldLinkOpt,GCBypass オプションの詳細については，「(5) -OldLinkOpt,GCBypass オプション（AIX で有効）」を参照してください。

32.5.11　規格の設定

規格との仕様チェックを設定するコンパイラオプションについて，説明します。

(1)　-StdMIA オプション

(a)　形式

```
-StdMIA {,13|,14} +
-noStdMIA
```

(b)　機能

仕様チェック機能として，MIA 仕様※を範囲外チェックします。

仕様範囲外の記述があると，警告レベルのエラーメッセージが出力されます。

注※

MULTIVENDOR INTEGRATION ARCHITECTURE VERSION 1.3 および 1.4（日本電信電話株式会社）の仕様（JIS X 3002-1988　電子計算機プログラム COBOL の仕様が前提）

-StdMIA,13

MIA バージョン 1.3 仕様を範囲外チェックします。

-StdMIA,14

MIA バージョン 1.4 仕様を範囲外チェックします。

なお，このオプションを指定すると，-StdMIA,13 サブオプションを仮定します。

-noStdMIA

-StdMIA オプションの指定を打ち消します。

(c)　注意事項

- -StdMIA オプション，-Std85 オプション，-Std2002 オプションは，同時に指定できません。同時に指定した場合は，警告のメッセージが出力され，次の優先順位に従ってオプションが有効となります。
  1. -Std2002 オプション
  2. -Std85 オプション
  3. -StdMIA オプション
- 次のオプションは，-StdMIA オプションと背反となるため，同時に指定できません。同時に指定した場合は，-StdMIA オプションが有効となり，次のオプションが無効となります。

  -V3Spec　-StdVersion　-CompatiV3　-H8Switch

  -Cblctr　-IgnoreLCC　-JPN　-CmDol　-Comp5　-NumAccept

  -V3Rec　-EquivRule,NotAny　-SQL　-BinExtend　-MaxDigits38

  -IntResult,DecFloat40　-LiteralExtend　-V3RecFCSpace　-SQLDisp

- -StdMIA,13 オプションは，次のオプションと背反となるため，同時に指定できません。同時に指定した場合は，-StdMIA,13 オプションが有効となり，次のオプションが無効となります。

  -EquivRule,StdCode　-Bin1Byte
- このオプションは，自由形式正書法で書かれた COBOL 原始プログラムをコンパイルする場合には指定できません。指定した場合，エラーとなってコンパイルが中止されます。

(2)　-Std85 オプション

(a)　形式

```
-Std85 {, {High | Middle | Low }  | ,Obso | ,Report} +
-noStd85
```

(b)　機能

仕様チェック機能として，JIS 仕様※の範囲外チェックや廃要素をチェックします。

仕様範囲外の記述や廃要素があると，警告レベルのエラーメッセージが出力されます。

注※

JIS X 3002-1992　電子計算機プログラム COBOL の仕様

-Std85,High

JIS 仕様の範囲外チェックをします。

-Std85,Middle

JIS 仕様の中位集合範囲外（上位集合，JIS 仕様の範囲外）チェックをします。

-Std85,Low

JIS 仕様の下位集合範囲外（中位集合，上位集合，JIS 仕様の範囲外）チェックをします。

-Std85,Obso

JIS 仕様の廃要素をチェックします。

-Std85,Report

JIS 仕様の報告書作成機能をチェックします。

-noStd85

-Std85 オプションの指定を打ち消します。

(c)　注意事項

- -Std85 オプションは，次のオプションと背反となるため，同時に指定できません。同時に指定した場合は，-Std85 オプションが有効となり，次のオプションが無効となります。

  -V3Spec　-StdVersion　-CompatiV3　-H8Switch　-Cblctr

  -IgnoreLCC　-JPN　-CmDol　-Comp5　-NumAccept　-V3Rec

-EquivRule,NotAny -SQL -BinExtend -MaxDigits38
-IntResult,DecFloat40 -LiteralExtend -V3RecFCSpace -SQLDisp

- このオプションは，自由形式正書法で書かれた COBOL 原始プログラムをコンパイルする場合には指定できません。指定した場合，エラーとなってコンパイルが中止されます。

(3) -Std2002 オプション

(a) 形式

```
-Std2002 {,OutRange|,Obso|,Archaic} +
-noStd2002
```

(b) 機能

仕様チェック機能として，COBOL2002 仕様の範囲外チェックや廃要素をチェックします。

仕様範囲外の記述や廃要素があると，警告レベルのエラーメッセージが出力されます。

-Std2002,OutRange
COBOL2002 仕様の範囲外チェックをします。

-Std2002,Obso
COBOL2002 仕様の廃要素をチェックします。

-Std2002,Archaic
COBOL2002 仕様の古典的要素をチェックします。

-noStd2002
-Std2002 オプションの指定を打ち消します。

(c) 注意事項

- -Std2002 オプションは，次のオプションと背反関係にあり同時に指定した場合は，-Std2002 オプションを有効とします。
  -V3Spec -StdVersion -CompatiV3 -H8Switch -Cblctr
  -IgnoreLCC -JPN -CmDol -Comp5 -NumAccept -V3Rec
  -EquivRule,NotAny -SQL -BinExtend -MaxDigits38
  -IntResult,DecFloat40 -LiteralExtend -V3RecFCSpace -SQLDisp

(4) -StdVersion オプション

(a) 形式

```
-StdVersion, {1|2}
-noStdVersion
```

(b) 機能

第 1 次または第 2 次規格の言語仕様の解釈でコンパイルするためのオプションです。このオプションを指定しないときは，第 3 次規格以降の解釈でコンパイルされます。

-StdVersion,1

第 1 次規格（JIS72，ISO72，ANSI68）の解釈でコンパイルします。

-StdVersion,2

第 2 次規格（JIS80，ISO78，ANSI74）の解釈でコンパイルします。

-noStdVersion

-StdVersion オプションの指定を打ち消します。

このオプションを指定した場合，第 3 次規格以降の解釈※でコンパイルします。

注※

第 3 次規格以降の解釈とは，次の二つの規格の解釈を指します。

- 第 3 次規格（ISO85，ANSI85，JIS88，JIS92）
- 第 4 次規格（ISO/IEC 1989:2002，JIS X3002:2011）

-StdVersion オプション指定時，および未指定時の言語仕様の相違を，次に示します。

差異の生じる言語仕様	第 1 次規格	第 2 次規格	第 3 次規格以降
比較条件の略記法の NOT	略記法の条件の中の NOT が比較演算子の一部とも取れるときには論理演算子とみなす。	NOT の直後に GREATER，>，LESS，<，EQUAL，=，>=，<=のどれかが続くとき，NOT は比較条件の一部となる。これ以外の NOT は論理演算子とみなす。	
MOVE 文の位取り	送り出し側作用対象が位取りした整数項目（PICTURE 句の右端文字が P）で，受け取り側作用対象が英数字編集項目の場合，あとに続くゼロを切り捨てて空白とする。	左記の場合，あとに続くゼロを切り捨てない。	
UNSTRING 文の DELIMITED BY ALL 指定	複数の連続する区切り文字が現れた場合，区切り文字の受け取り領域に入るだけ区切り文字を移す。	左記の場合，区切り文字の受け取り領域には 1 個（1 組）の区切り文字しか移さない。	
COPY 文の展開	第 1 次規格の仕様に従った書き方は旧仕様として展開する。	第 2 次規格以降の仕様に従って COPY 文を展開する。	
JUST 句の処理	JUST 句の指定がある英数字，英字，英数字編集に VALUE 句で初期値を与える場合，JUST 句の機能を働かせる。	左記の場合，JUST 句の機能を働かせない。	
WRITE 文の ADVANCING の仮定	1 ファイルに対する複数個の WRITE 文で ADVANCING 指定があるものとないものを混用した場合，ADVANCING 指定のないものに対しては BEFORE ADVANCING 1 か AFTER ADVANCING 1 のどちらかを仮定	左記の場合，常に AFTER ADVANCING 1 を仮定する。	

差異の生じる言語仕様	第 1 次規格	第 2 次規格	第 3 次規格以降
	する。どちらを仮定するかは WRITE 文の指定の内容による。		
PERFORM 文の VARYING AFTER 指定がある場合の反復制御変数を初期化する位置	• TEST AFTER 指定時 外側のループの反復制御変数を BY 指定の値で増加させてから，内側のループの反復制御変数を FROM 指定の変数で再び初期化する。 • TEST BEFORE 指定時 内側のループの反復制御変数 FROM 指定の変数で初期化してから，外側のループの反復制御変数を BY 指定の値で増加させる。		TEST AFTER 指定時も TEST BEFORE 指定時も，外側のループの反復制御変数を BY 指定の値で増加させてから，内側の反復制御変数を FROM 指定で再び初期化する。
SELECT OPTIONAL 句	覚え書きとする。	覚え書きとしない。	
CURRENCY SIGN 句に指定できる定数	"=", "/"が指定できる。	"=", "/"は指定できない。	

(c) 注意事項

- このオプションは，自由形式正書法で書かれた COBOL 原始プログラムをコンパイルする場合には指定できません。指定した場合，エラーとなってコンパイルが中止されます。

32.5.12 他システムとの移行の設定

他システムとの移行に関連するコンパイラオプションについて，説明します。

(1) -CompatiV3 オプション

(a) 形式

```
-CompatiV3
-noCompatiV3
```

(b) 機能

-CompatiV3

メインフレーム（VOS3）COBOL85 との互換を指定するオプションです。-CompatiV3 オプションは，VOS3 COBOL85 で開発した COBOL プログラムを UNIX 環境に移行することを目的として提供しているオプションで，日立拡張機能の中で使用頻度の高いものを対象としています。

このオプションを指定した場合，指定しない場合と比べて次の点が異なります。

- ASSIGN 句で指定した外部装置名に関連する環境変数名が，次のように変更されます。

（書き方）

```
SYSnnn〔-装置クラス〕〔-装置名〕〔-編成〕〔-ファイル定義名〕
または，〔装置クラス-〕〔装置名-〕〔編成-〕ファイル定義名
```

- -CompatiV3 オプション指定時の環境変数名

  次のどちらかとなる。

  CBL_SYSnnn

  CBL_ファイル定義名

- -CompatiV3 オプション未指定時の環境変数名

  次のどちらかとなる。

  CBL_SYSnnn〔_装置クラス〕〔_装置名〕〔_編成〕〔_ファイル定義名〕

  CBL_〔装置クラス_〕〔装置名_〕〔編成_〕ファイル定義名

- OPEN I-O で開かれた順ファイルに対して WRITE 文が指定できます。このとき，次の条件をすべて満たすと，WRITE 文は REWRITE 文として扱われます。

（条件）

- -CompatiV3 オプションの指定がある。
- EXTERNAL 句が指定されているか，または OPEN I-O で開かれている。
- WRITE 文に ADVANCING 指定がない。
- LINAGE 句の指定がない。

- -CompatiV3 オプションを指定した場合，次のオプションが仮定されます。

（仮定されるオプション）

- -V3Rec,Variable

  可変長形式の原始プログラムのコンパイルを，メインフレームの日本語項目の扱いに合わせるオプションです。

  なお，固定長形式でコンパイルする場合，-CompatiV3 オプションと-V3Rec,Fixed オプションを指定する必要があります。

- -JPN,Alnum

  日本語項目，日本語編集項目，および日本語文字定数を，それぞれ英数字項目，英数字編集項目，および英数字定数として扱うオプションです。

  VOS3 COBOL85 で XCOBOL=(N)を指定したときと同じ動作となります。

- VOS3 COBOL85 上で使用できる報告書作成機能を記述したプログラムをコンパイルできます。

  なお，-CompatiV3 オプションを指定しない場合，UNIX COBOL85 と同じ言語仕様の報告書作成機能を使用します。

- 通信節による画面機能で，複数プログラムで通信文が実行された場合，一つの仮想端末を複数のプログラム間で共用し，送受信します。

- DIVIDE 文書き方 4 および書き方 5 の，除算の剰余を計算するのに使う商は，商を計算したときの中間結果と同じけた数，同じ小数点位置，同じ符号の有無を持ちます。

（例）

```
DIVIDE A BY B GIVING C REMAINDER D.
```

上記の例の DIVIDE 文は，下記のような連続した操作に変換されます（ただし，TMP1，TMP2，TMP3 はコンパイラによって作成された中間結果の記憶場所を表します）。

```
A/B → TMP1
TMP1 → TMP2
A - (B * TMP2) → TMP3
TMP1 → C
TMP3 → D
```

- -CompatiV3 オプションを指定したときは，TMP2 は TMP1 と同じけた数，同じ小数点位置，同じ符号になります。
- -CompatiV3 オプション未指定のときは，TMP2 は商（C）と同じけた数，同じ小数点位置，同じ符号になります。

- -CompatiV3 オプションを指定した場合，英数字定数の分離符には，アポストロフィ（'）だけが使用できます。
- -CompatiV3 オプションと-EquivRule オプションは，同時に指定できません。指定した場合，あとに指定したオプションが有効となります。
- -CompatiV3 オプションと-IntResult,DecFloat40 オプションを同時に指定したとき，算術式と算術文に対する-CompatiV3 オプションの仕様（DIVIDE 文の書き方 4 および書き方 5）は無効となります。この場合，-IntResult,DecFloat40 オプション指定時での演算を行います。-IntResult,DecFloat40 オプションについては，「(21)　-IntResult,DecFloat40 オプション（AIX(64)，Linux(x64)で有効）」を参照してください。

-noCompatiV3

-CompatiV3 オプションの指定を打ち消します。

-CompatiV3 オプションを指定した場合，-V3Rec,Variable オプションおよび-JPN,Alnum オプションが仮定されますが，-noCompatiV3 オプションを指定しても，仮定された-V3Rec,Variable オプションおよび-JPN,Alnum オプションは打ち消されません。

(c)　注意事項

- このオプションは，自由形式正書法で書かれた COBOL 原始プログラムをコンパイルする場合には指定できません。指定した場合，エラーとなってコンパイルが中止されます。
- このオプションを指定した場合，連結式を記述できません。
- コンパイラ環境変数 CBLV3UNICODE に YES を指定し，-UniObjGen オプションを指定した場合，-CompatiV3 オプションを指定しても-JPN,Alnum オプションは仮定されません。
- このオプションを指定した場合，特殊名段落の定数 10 が有効になります。ただし，コンパイラ環境変数 CBLV3UNICODE に YES を指定すると，-UniObjGen オプションと-CompatiV3 オプションの

同時指定が可能となるため，Unicode で多バイトになる文字を指定した場合，コンパイルエラーとなります。

(2) -Compati85 オプション

(a) 形式

```
-Compati85 {,IoStatus|,Linage|,Call|,Power|,Syntax|,IDParag|,RsvWord|,NoPropagate|,All} +
-noCompati85
```

(b) 機能

-Compati85,IoStatus

入出力状態 1x，2x に対する処理を COBOL85 と同様にします。

-Compati85,Linage

LINAGE 値が不正なときの処理を COBOL85 と同様にします。

-Compati85,Call

CALL 文の ON EXCEPTION，または ON OVERFLOW 指定の無条件文を実行するエラーの条件を COBOL85 と同様にします。

-Compati85,Power

べき乗演算でのエラー時の処理を COBOL85 と同様にします。また，べき乗演算の精度を，同一システムの COBOL85（製品が存在する場合だけ）と同様にします。べき乗演算と-Compati85,Power オプションとの関係については，「21.3.3 例外チェックが無効な場合の動作」の「(1) 手続き文の実行中にエラーが発生し，例外を検出した場合」を参照してください。

-Compati85,Syntax

次の機能について，コンパイル時の解釈を COBOL85 と同様にします。

- COPY 文の REPLACING 指定での置換位置
- 見出し部注記項（-Compati85,IDParag オプション指定時と同等の解釈をする）
- 部の見出し，節名，段落名，END PROGRAM などの開始位置チェック
- 直前のピリオドが省略された A 領域に置かれた語の解釈
- COBOL 語の最大長（30 文字とする）
- 予約語（COBOL85 では予約語で，COBOL2002 では文脈依存語に変更されたものを予約語に戻す）

  なお，COBOL85 で予約語だったものが COBOL2002 で文脈依存語となっているのは，次の語です。

  ONLY　PREVIOUS　RECURSIVE
- プログラム名の長さ（30 バイトとする）
- 自由形式正書法の 1 行の長さ（80 文字とする）

-Compati85,IDParag

見出し部の構文チェックを緩和します。

このオプションを指定すると，見出し部に AUTHOR 段落，INSTALLATION 段落，DATE-WRITTEN 段落，DATE-COMPILED 段落，SECURITY 段落，および REMARKS 段落が記述できます。

-Compati85,RsvWord

予約語のうち COBOL2002 で新たに追加されたものを除き，COBOL85 の予約語と同じ範囲でコンパイルします。

-Compati85,NoPropagate

CALL 文，INVOKE 文，および利用者定義関数で伝播を抑止します。

次のプログラムを呼び出す場合，例外が伝播することはないため，このオプションを指定するとオブジェクトファイルサイズを縮小できます。

- PROPAGATE 指令が指定されていないプログラム
- COBOL85 で作成したプログラム
- C プログラムなど，COBOL 言語以外の言語を使用したプログラム
- COBOL2002 で使用できるサービスルーチン
- OpenTP1，XMAP3（AIX(32)で有効）などの関連プログラムが提供している関数など

ただし，条件によっては，自動的に呼び出し元プログラムへの伝播が抑止される場合があります。その場合は，このオプションを指定する必要はありません。詳細は，「21.5.3　例外を受け取れないプログラムに例外を伝播させた場合の動作」を参照してください。

-Compati85,All

-Compati85 オプションのすべてのサブオプションを仮定します。

-Compati85,All を指定した場合，-Compati85 オプションのすべてのサブオプションを指定したものとみなします。

-noCompati85

-Compati85 オプションの指定を打ち消します。

(c)　注意事項

- -Compati85 オプションは，単独で指定できません。必ずサブオプションを指定する必要があります。
- -Compati85,IDParag オプションを指定した場合，見出し部中に COPY 文，REPLACE 文，および INCLUDE 文を記述できません。注記項中に COPY 文，REPLACE 文，または INCLUDE 文を記述した場合，その COPY 文，REPLACE 文，または INCLUDE 文を注記とみなします。
- -Compati85,IDParag オプションを指定した場合，注記項中に継続行を表す標識（-）が現れたときは，コンパイルエラーとなります。

- -Compati85,IDParag オプションを指定した場合，見出し部には，COBOL85 で指定できる段落見出し（PROGRAM-ID，AUTHOR，INSTALLATION，DATE-WRITTEN，DATE-COMPILED，SECURITY，および REMARKS）以外を指定できません。
- -Compati85,Syntax オプションを指定した場合に適用される予約語は，COBOL2002 の予約語および ONLY，PREVIOUS，RECURSIVE の三つの語となります。

  -Compati85,RsvWord オプションを指定した場合に適用される予約語は，COBOL85 の予約語だけとなります。
- -Compati85,RsvWord オプションを指定した場合，COBOL2002 で新しく追加された構文が利用できなくなります。このため，既存プログラムに COBOL2002 の新機能を加えて拡張することが困難になります。既存プログラムに COBOL2002 の新しい予約語が使われている場合，新しい予約語とは異なる語に置き換えることを推奨します。COBOL2002 で追加された新しい予約語については，マニュアル「COBOL2002　言語　標準仕様編」「4.8　予約語」を参照してください。
- -Compati85 オプションのうち次のオプションを指定した場合，COBOL ソース中に TURN 指令を指定できません。指定した場合，コンパイルエラーとなります。
  - -Compati85,IoStatus
  - -Compati85,Linage
  - -Compati85,Call
  - -Compati85,Power
  - -Compati85,NoPropagate
- -Compati85,NoPropagate オプションは，CALL 文のほかに INVOKE 文，および利用者定義関数の呼び出しの場合にも，伝播が抑止されます。なお，CALL 文，INVOKE 文，および利用者定義関数を使用していない場合，このオプションは意味を持ちません。
- -Compati85,Syntax オプションは，-LiteralExtend オプションと同時に指定すると，-Compati85,Syntax オプションが有効となり，-LiteralExtend オプションは無効となります。

(3)　-H8Switch オプション

(a)　形式

```
-H8Switch
-noH8Switch
```

(b)　機能

-H8Switch

HITAC8000 シリーズの仕様でコンパイルするためのオプションです。

次の文で，送り出し側作用対象が数字項目（または固定小数点数字定数）で，受け取り側作用対象が英数字項目（または英数字編集項目）の場合，数字項目を英数字項目とみなして転記します。

- MOVE

- WRITE FROM
- REWRITE FROM
- RELEASE FROM
- ACCEPT FROM DAY
- ACCEPT FROM DATE
- ACCEPT FROM TIME
- ACCEPT FROM DAY-OF-WEEK

また，比較条件で，両辺の組み合わせが数字項目（または固定小数点数字定数）と英数字項目（または英数字編集項目，英字項目，英数字定数）の場合，数字項目を英数字項目とみなして比較します。

このオプションを指定しなかった場合，数字項目は数字項目として転記，比較されます。

-noH8Switch

-H8Switch オプションの指定を打ち消します。

(4) -Cblctr オプション

(a) 形式

```
-Cblctr
-noCblctr
```

(b) 機能

-Cblctr

報告書作成機能を使用するときに指定します。このオプションを指定すると，CBL-CTR 特殊レジスタが生成され，COBOL プログラムと報告書作成機能間で連絡ができるようになります。

詳細は，マニュアル「COBOL2002　言語　拡張仕様編」「7. 報告書作成機能の拡張」を参照してください。

-noCblctr

-Cblctr オプションの指定を打ち消します。

(5) -DigitsTrunc オプション

(a) 形式

```
-DigitsTrunc
-noDigitsTrunc
```

(b) 機能

-DigitsTrunc

原始プログラム中の転記文で受け取り側作用対象が 2 進項目の場合，送り出し側作用対象のけたが受け取り側作用対象よりも大きいときに，上位のけたを切り捨てます。

（例）次のプログラムの実行結果を示します。

```
       77 A PIC S9(4) USAGE COMP VALUE +123.
       77 B PIC S9(2) USAGE COMP.
           :
           MOVE A TO B.
```

- -DigitsTrunc オプションを指定した場合 …… +23
- -DigitsTrunc オプションを指定しない場合 … +123

このオプションを指定しなかった場合，原始プログラム中の転記文で受け取り側作用対象が 2 進項目のとき，送り出し側作用対象の有効けた数が受け取り側作用対象よりも大きいと，入り切らない部分が切り捨てられます。

-noDigitsTrunc

-DigitsTrunc オプションの指定を打ち消します。

(c) 注意事項

- -DigitsTrunc オプションと-Comp5 オプションを同時に指定した場合，あとに指定したオプションだけが有効となります。

(6) -IgnoreLCC オプション

(a) 形式

```
-IgnoreLCC
-noIgnoreLCC
```

(b) 機能

-IgnoreLCC

WRITE～ADVANCING／POSITIONING 文で書き出すレコードの先頭 1 バイトを出力しないことを指定します。

あらかじめ，レコード記述項で行送り制御文字用の 1 バイト分の領域を確保しておいたプログラムをコンパイルするときに指定します。ただし，次の場合には-IgnoreLCC オプションの指定があっても，レコードの先頭から出力されます。

- 順ファイル以外のファイルの場合
- 順ファイルの WRITE 文に行送りの指定（ADVANCING，POSITIONING）がない場合
- 順ファイルのファイル記述項に LINAGE 句の指定がある場合

-noIgnoreLCC

-IgnoreLCC オプションの指定を打ち消します。

(7) -CmAster オプション

(a) 形式

```
-CmAster
-noCmAster
```

(b) 機能

-CmAster

固定形式正書法のとき，行の先頭文字（1 カラム目の文字）が標準コードの星印（*）である行を注記行とします。

ただし，このオプションは固定形式正書法で記述された COBOL ソースファイルをコンパイルするときだけ有効となります。

-noCmAster

-CmAster オプションの指定を打ち消します。

(8) -CmDol オプション

(a) 形式

```
-CmDol
-noCmDol
```

(b) 機能

-CmDol

固定形式正書法のとき，行の先頭から 7 カラム目の文字が「$」記号である行を注記行とします。

ただし，このオプションは固定形式正書法で記述された COBOL ソースファイルをコンパイルするときだけ有効となります。

-noCmDol

-CmDol オプションの指定を打ち消します。

(9) -Comp5 オプション

(a) 形式

```
-Comp5
-noComp5
```

(b) 機能

-Comp5

COBOL プログラム中の USAGE 句の COMP-5 を使用できるようにするオプションです。

このオプションを指定すると，COMP-5 は，COMP を指定した場合と同様に処理されます。このオプションを指定しなかった場合，USAGE 句に COMP-5 が指定されているとエラーになります。

COMP-5 は，-BigEndian,Bin オプションの対象にはなりません。(Linux で有効)

COMP-5 を指定できる場所は，COMP を指定できる場所と同じです。また，COMP-5 は，COMP と同じ仕様となるため，PICTURE 句のけた数，データのバイト数，および実際に格納できる数値は次のとおりです。

表 32-2　COMP-5 で実際に格納できる数値

PICTURE 句のけた数	データのバイト数	実際に格納できる数値	
		符号あり	符号なし
1～4 けた	2 バイト	-2^{15}～$2^{15}-1$	0～$2^{15}-1$
5～9 けた	4 バイト	-2^{31}～$2^{31}-1$	0～$2^{31}-1$
10～18 けた	8 バイト	-2^{63}～$2^{63}-1$	0～$2^{63}-1$

-noComp5

-Comp5 オプションの指定を打ち消します。

(c) 注意事項

- -Comp5 オプションと-DigitsTrunc オプションを同時に指定した場合，あとに指定したオプションだけが有効となります。

(10) -V3Spec オプション

(a) 形式

```
-V3Spec 〔,CopyEased〕
-noV3Spec
```

(b) 機能

-V3Spec

VOS3 COBOL85 でサポートされていない，UNIX 固有の言語仕様に対して構文チェックするためのオプションです。

VOS3 COBOL85 09-00 を対象として，ほぼ同等の言語仕様で構文チェックをし，VOS3 COBOL85 ではサポートされていない UNIX 固有の言語仕様に対して警告メッセージを出力します。

出力するオブジェクトファイルは，このオプションの影響を受けません。

このオプションは，VOS3 COBOL85 互換機能です。

-V3Spec,CopyEased

-V3Spec オプションで指定された構文チェックの範囲を緩和し，COPY 文，REPLACE 文の一部のエラーチェックをしません。

このオプションは，VOS3 COBOL85 互換機能です。

-noV3Spec

-V3Spec オプションの指定を打ち消します。

-V3Spec オプションを指定した場合，-V3Rec,Variable オプションが仮定されますが，-noV3Spec オプションを指定しても，仮定された-V3Rec,Variable オプションは打ち消されません。

(c) 注意事項

- 日本語項目の部分参照については，VOS3 COBOL85 の XCOBOL=(-N)オプションを指定した状態を対象とします。
- -V3Spec オプションと-CompatiV3 オプションを同時に指定した場合，一部のチェック項目が重複するため，-CompatiV3 オプションのエラーメッセージが出力され，このオプションのエラーメッセージは出力されません。
- -V3Spec オプションを指定すると，-V3Rec,Variable オプションが仮定されます。このため，固定長形式でコンパイルする場合は-V3Rec,Fixed オプションを同時に指定する必要があります。
- -V3Spec オプションと次のオプションを同時に指定すると，-V3Spec オプションが有効になり，次のオプションは無効になります。

  -Comp5　-CmAster　-BinExtend　-CBLVALUE　-Bin1Byte

  -NumAccept　-CmDol　-MaxDigits38　-IntResult,DecFloat40

  -LiteralExtend
- -V3Spec オプションと-UniObjGen オプションを同時に指定した場合，-V3Rec,Variable オプションは仮定されません。-V3Rec オプションを有効とするときは，環境変数 CBLV3UNICODE を指定してください。

(11) -V3ConvtName オプション

(a) 形式

```
-V3ConvtName
-noV3ConvtName
```

(b) 機能

-V3ConvtName

原文名定数の「¥」を「_」に，「@」を「!」に変換した名称で登録集原文を検索する場合，このオプションを指定します。このオプションは，COPY 文および INCLUDE 文で有効になります。また，原

文名定数の末尾の拡張子を省略した場合は，「.cbl」が仮定されます。なお，原文名指定の場合は，このオプションの対象となりません。

（例 1）原文名定数に¥，@が含まれる場合

原文名定数中に¥，@が含まれる場合は，-V3ConvtName オプションを指定してコンパイルします。

- COPY '¥ABC'.
  _ABC.cbl を登録集原文として検索します。
- COPY '@ABC'.
  !ABC.cbl を登録集原文として検索します。

（例 2）原文名に¥，@が含まれる場合

¥，@，#は，COBOL2002 では原文名に使用できない文字のため，コンパイルエラーとなります。この場合は，¥，@を含む原文名を定数指定に変更してから-V3ConvtName オプションを指定してコンパイルします。※

- COPY ¥ABC.　→　COPY '¥ABC'.
  _ABC.cbl を登録集原文として検索します。
- COPY @ABC.　→　COPY '@ABC'.
  !ABC.cbl を登録集原文として検索します。

-V3ConvtName オプションを使用しない場合は，¥，@を¥，@，#以外の文字に変更してください。

注※

#を含む原文名は，原文名定数に変更するだけで使用できます。

COPY 文での登録集原文の利用については，「32.3.1　原始文操作機能」を参照してください。

このオプションは，VOS3 COBOL85 互換機能です。

-noV3ConvtName

-V3ConvtName オプションの指定を打ち消します。

(12)　-Switch オプション（AIX で有効）

(a)　形式

```
-Switch, {EBCDIC | EBCDIK}　〔,Unprintable〕　〔,noApplyJpnItem〕
-noSwitch
```

(b)　機能

照合順序および字類条件を EBCDIC コードまたは EBCDIK コードに切り替えるためのオプションです。

-Switch,EBCDIC

英数字の照合順序および字類条件を EBCDIC コードに切り替えます。ただし，1 バイトで印字できないコードは，照合順序として X'00'または X'FF'に割り付けられます。

-Switch,EBCDIK

英数字の照合順序および字類条件を EBCDIK コードに切り替えます。ただし，1 バイトで印字できないコードは，照合順序として X'00'または X'FF'に割り付けられます。

-Switch,xxxxx,Unprintable

英数字の照合順序を，xxxxx に指定したコードに切り替えます。

1 バイトで印字できないコードも，照合順序として固有の文字位置に割り付けます。詳細は，「(c)　注意事項」を参照してください。

xxxxx は，EBCDIC，EBCDIK のどちらかを指定します。

-Switch,xxxxx,noApplyJpnItem

英数字の照合順序を，xxxxx に指定したコードに切り替えます。

日本語項目を-Switch オプションの適用対象外とします。

xxxxx は，EBCDIC，EBCDIK のどちらかを指定します。

-noSwitch

-Switch オプションの指定を打ち消します。

(c)　注意事項

字類条件の判断基準と文字の大小関係

プログラムに記述した字類条件に対するコードの判断基準と文字の大小関係を，次に示します。

表 32-3　字類条件の判断基準と文字の大小関係

条件		ASCII	EBCDIC	EBCDIK
字類の判断基準	ALPHABETIC	a～z，A～Z	A～Z，a～z	A～Z
	ALPHABETIC-UPPER	A～Z	A～Z	A～Z
	ALPHABETIC-LOWER	a～z	a～z	－
文字の大小関係		a～z ＞ A～Z	a～z ＜ A～Z	a～z ＜ A～Z
組み込み関数の結果	UPPER-CASE	a～z→A～Z	a～z→A～Z	a～z→A～Z
	LOWER-CASE	A～Z→a～z	A～Z→a～z	A～Z→a～z

（凡例）

－：字類条件が真になる事はない

＞：大きい（a ＞ b　a は b より大きい）

＜：小さい（a ＜ b　a は b より小さい）

→：置換（a→b　a を b に置換する）

PROGRAM COLLATING SEQUENCE 句と-Switch オプションの指定値の対応を次に示します。

PROGRAM COLLATING SEQUENCE 句の符号系名	-Switch オプション	
	EBCDIC サブオプションを指定した場合	EBCDIK サブオプションを指定した場合
指定なし	EBCDIC コード	EBCDIK コード
EBCDIC	EBCDIC コード	S レベルエラー
EBCDIK	S レベルエラー	EBCDIK コード
EBCDIC，EBCDIK 以外	S レベルエラー	S レベルエラー

-Switch オプションの対象

-Switch オプションの対象を次に示します。

機能	対象の字類
比較条件	英字，英数字，日本語※1
整列併合※2	英字，英数字

注※1

noApplyJpnItem サブオプションを指定している場合，対象の字類が日本語のときは-Switch オプションの対象となりません。

注※2

-Switch オプション指定による照合順序は，索引ファイルの参照キーには適用されません。参照キーの比較は，固有文字集合の大小順序に従います。

照合順序

比較のときに使用する照合順序として，1 バイトの印字可能文字は対応する文字位置に割り付けられ，その他の印字できないコードは X'00'または X'FF'に割り付けられます。

ただし，Unprintable サブオプションが有効な場合は，印字できないコードに，印字可能文字が割り付けられていない固有の文字位置が割り付けられます。Unprintable サブオプションの有無での照合順序の違いを次に示します。

文字コード種別	Unprintable サブオプション	
	無効	有効
印字可能文字	対応する文字の文字位置に割り付けられます。	
印字できないコード	X'80'は X'00'に，X'80'以外はすべて X'FF'に，それぞれ割り付けられます。	印字可能文字が割り付けられていない固有の文字位置に割り付けられます。 ただし，印字できないコードの大小関係は保証しません。

Unprintable サブオプションが有効な場合，JIS8 単位コードが，EBCDIC コードまたは EBCDIK コードのどの位置に割り付けられるかをそれぞれ次に示します。なお，表中の網掛けは，印字できないコードを示します。

図 32-2　EBCDIC コードと JIS8 単位コードの対応表

下4ビット	上4ビット															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	X'00'	X'10'	X'80'	X'90'	X'20'	X'26'	X'2D'	X'BA'	X'C3'	X'CA'	X'D1'	X'D8'	X'7B'	X'7D'	X'24'	X'30'
	NUL	DLE			空白	&	-						{	}	$	0
1	X'01'	X'11'	X'81'	X'91'	X'A0'	X'A9'	X'2F'	X'BB'	X'61'	X'6A'	X'7E'	X'D9'	X'41'	X'4A'	X'9F'	X'31'
	SOH	DC1					/		a	j	¯		A	J		1
2	X'02'	X'12'	X'82'	X'16'	X'A1'	X'AA'	X'B2'	X'BC'	X'62'	X'6B'	X'73'	X'DA'	X'42'	X'4B'	X'53'	X'32'
	STX	DC2		SYN					b	k	s		B	K	S	2
3	X'03'	X'13'	X'83'	X'93'	X'A2'	X'AB'	X'B3'	X'BD'	X'63'	X'6C'	X'74'	X'DB'	X'43'	X'4C'	X'54'	X'33'
	ETX	DC3							c	l	t		C	L	T	3
4	X'9C'	X'9D'	X'84'	X'94'	X'A3'	X'AC'	X'B4'	X'BE'	X'64'	X'6D'	X'75'	X'DC'	X'44'	X'4D'	X'55'	X'34'
									d	m	u		D	M	U	4
5	X'09'	X'0a'	X'85'	X'95'	X'A4'	X'AD'	X'B5'	X'BF'	X'65'	X'6E'	X'76'	X'DD'	X'45'	X'4E'	X'56'	X'35'
	HT	NL	LF						e	n	v		E	N	V	5
6	X'86'	X'08'	X'17'	X'96'	X'A5'	X'AE'	X'B6'	X'C0'	X'66'	X'6F'	X'77'	X'DE'	X'46'	X'4F'	X'57'	X'36'
		BS	ETB						f	o	w		F	O	W	6
7	X'7F'	X'87'	X'1B'	X'04'	X'A6'	X'AF'	X'B7'	X'C1'	X'67'	X'70'	X'78'	X'DF'	X'47'	X'50'	X'58'	X'37'
	DEL		ESC	EOT					g	p	x		G	P	X	7
8	X'97'	X'18'	X'88'	X'98'	X'A7'	X'B0'	X'B8'	X'C2'	X'68'	X'71'	X'79'	X'E0'	X'48'	X'51'	X'59'	X'38'
		CAN							h	q	y		H	Q	Y	8
9	X'8D'	X'19'	X'89'	X'99'	X'A8'	X'B1'	X'B9'	X'60'	X'69'	X'72'	X'7A'	X'E1'	X'49'	X'52'	X'5A'	X'39'
		EM						`	i	r	z		I	R	Z	9
A	X'8E'	X'92'	X'8A'	X'9A'	X'5B'	X'5D'	X'7C'	X'3A'	X'C4'	X'CB'	X'D2'	X'E2'	X'E8'	X'EE'	X'F4'	X'FA'
					[	]	|	:								
B	X'0B'	X'8F'	X'8B'	X'9B'	X'2E'	X'5C'	X'2C'	X'23'	X'C5'	X'CC'	X'D3'	X'E3'	X'E9'	X'EF'	X'F5'	X'FB'
	VT				.	¥	,	#								
C	X'0C'	X'1C'	X'8C'	X'14'	X'3C'	X'2A'	X'25'	X'40'	X'C6'	X'CD'	X'D4'	X'E4'	X'EA'	X'F0'	X'F6'	X'FC'
	FF	FS		DC4	<	*	%	@								
D	X'0D'	X'1D'	X'05'	X'15'	X'28'	X'29'	X'5F'	X'27'	X'C7'	X'CE'	X'D5'	X'E5'	X'EB'	X'F1'	X'F7'	X'FD'
	CR	GS	ENQ	NAK	(	)	_	'								
E	X'0E'	X'1E'	X'06'	X'9E'	X'2B'	X'3B'	X'3E'	X'3D'	X'C8'	X'CF'	X'D6'	X'E6'	X'EC'	X'F2'	X'F8'	X'FE'
	SO	RS	ACK		+	;	>	=								
F	X'0F'	X'1F'	X'07'	X'1A'	X'21'	X'5E'	X'3F'	X'22'	X'C9'	X'D0'	X'D7'	X'E7'	X'ED'	X'F3'	X'F9'	X'FF'
	SI	US	BEL	SUB	!	^	?	″								

図 32-3　EBCDIK コードと JIS8 単位コードの対応表

下4ビット	上4ビット															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	X'00'	X'10'	X'80'	X'90'	X'20'	X'26'	X'2D'	X'6A'	X'73'	X'BF'	X'77'	X'79'	X'7B'	X'7D'	X'24'	X'30'
	NUL	DLE			空白	&	-	j	s	ソ	w	y	{	}	$	0
1	X'01'	X'11'	X'81'	X'91'	X'A1'	X'AA'	X'2F'	X'6B'	X'B1'	X'C0'	X'7E'	X'7A'	X'41'	X'4A'	X'9F'	X'31'
	SOH	DC1			。	ｪ	/	k	ｱ	ﾀ	￣	z	A	J		1
2	X'02'	X'12'	X'82'	X'16'	X'A2'	X'AB'	X'62'	X'6C'	X'B2'	X'C1'	X'CD'	X'E0'	X'42'	X'4B'	X'53'	X'32'
	STX	DC2		SYN	「	ｫ	b	l	ｲ	ﾁ	ﾍ		B	K	S	2
3	X'03'	X'13'	X'83'	X'93'	X'A3'	X'AC'	X'63'	X'6D'	X'B3'	X'C2'	X'CE'	X'E1'	X'43'	X'4C'	X'54'	X'33'
	ETX	DC3			」	ｬ	c	m	ｳ	ﾂ	ﾎ		C	L	T	3
4	X'9C'	X'9D'	X'84'	X'94'	X'A4'	X'AD'	X'64'	X'6E'	X'B4'	X'C3'	X'CF'	X'E2'	X'44'	X'4D'	X'55'	X'34'
					、	ｭ	d	n	ｴ	ﾃ	ﾏ		D	M	U	4
5	X'09'	X'0a'	X'85'	X'95'	X'A5'	X'AE'	X'65'	X'6F'	X'B5'	X'C4'	X'D0'	X'E3'	X'45'	X'4E'	X'56'	X'35'
	HT	NL	LF		・	ｮ	e	o	ｵ	ﾄ	ﾐ		E	N	V	5
6	X'86'	X'08'	X'17'	X'96'	X'A6'	X'AF'	X'66'	X'70'	X'B6'	X'C5'	X'D1'	X'E4'	X'46'	X'4F'	X'57'	X'36'
		BS	ETB		ヲ	ｯ	f	p	ｶ	ﾅ	ﾑ		F	O	W	6
7	X'7F'	X'87'	X'1B'	X'04'	X'A7'	X'A0'	X'67'	X'71'	X'B7'	X'C6'	X'D2'	X'E5'	X'47'	X'50'	X'58'	X'37'
	DEL		ESC	EOT	ｧ		g	q	ｷ	ﾆ	ﾒ		G	P	X	7
8	X'97'	X'18'	X'88'	X'98'	X'A8'	X'B0'	X'68'	X'72'	X'B8'	X'C7'	X'D3'	X'E6'	X'48'	X'51'	X'59'	X'38'
		CAN			ｨ	-	h	r	ｸ	ﾇ	ﾓ		H	Q	Y	8
9	X'8D'	X'19'	X'89'	X'99'	X'A9'	X'61'	X'69'	X'60'	X'B9'	X'C8'	X'D4'	X'E7'	X'49'	X'52'	X'5A'	X'39'
		EM			ｳ	a	i	`	ｹ	ﾈ	ﾔ		I	R	Z	9
A	X'8E'	X'92'	X'8A'	X'9A'	X'5B'	X'5D'	X'7C'	X'3A'	X'BA'	X'C9'	X'D5'	X'DA'	X'E8'	X'EE'	X'F4'	X'FA'
					[	]	|	:	ｺ	ﾉ	ｭ	ﾚ				
B	X'0B'	X'8F'	X'8B'	X'9B'	X'2E'	X'5C'	X'2C'	X'23'	X'74'	X'75'	X'78'	X'DB'	X'E9'	X'EF'	X'F5'	X'FB'
	VT				.	¥	,	#	t	u	x	ﾛ				
C	X'0C'	X'1C'	X'8C'	X'14'	X'3C'	X'2A'	X'25'	X'40'	X'BB'	X'76'	X'D6'	X'DC'	X'EA'	X'F0'	X'F6'	X'FC'
	FF	FS		DC4	<	*	%	@	ｻ	v	ﾖ	ﾜ				
D	X'0D'	X'1D'	X'05'	X'15'	X'28'	X'29'	X'5F'	X'27'	X'BC'	X'CA'	X'D7'	X'DD'	X'EB'	X'F1'	X'F7'	X'FD'
	CR	GS	ENQ	NAK	(	)	_	'	ｼ	ﾊ	ﾗ	ﾝ				
E	X'0E'	X'1E'	X'06'	X'9E'	X'2B'	X'3B'	X'3E'	X'3D'	X'BD'	X'CB'	X'D8'	X'DE'	X'EC'	X'F2'	X'F8'	X'FE'
	SO	RS	ACK		+	;	>	=	ｽ	ﾋ	ﾘ	ﾞ				
F	X'0F'	X'1F'	X'07'	X'1A'	X'21'	X'5E'	X'3F'	X'22'	X'BE'	X'CC'	X'D9'	X'DF'	X'ED'	X'F3'	X'F9'	X'FF'
	SI	US	BEL	SUB	!	^	?	″	ｾ	ﾌ	ﾙ	ﾟ				

Unprintable サブオプションが有効でない場合の照合順序の対応表については，マニュアル「COBOL2002　言語　標準仕様編」「付録 B　計算機文字集合」を参照してください。

-Switch オプションを指定したプログラムでの比較結果

-Switch オプションを指定したプログラムでは，特定のデータ種別の比較結果で注意が必要です。

次のように，1 バイトで印字できないコード（日本語データや数字データなど）を含む作用対象（データ項目や定数）を比較条件に指定している場合は，値が異なるのに等しいと解釈されたり，-Switch オプション未指定時とは大小関係が逆転したりすることがあります。

データ種別

- 日本語データ（英数字項目や日本語項目※に日本語文字が格納されている場合など）
- 数字データ（集団項目の従属項目に，用途（USAGE 句）が DISPLAY 以外の数字項目（2 進項目，内部 10 進項目，内部浮動小数点項目）がある場合など）

注※

noApplyJpnItem サブオプションが無効の場合です。

比較結果

- 日本語データや数字データで，1 バイトの印字できないコードでは，等しいと解釈されることがあります。

  ただし，Unprintable サブオプションが有効な場合は，コードの一致および不一致を判定できます。なお，Unprintable サブオプションが有効な場合でも，照合順序が切り替わることで，大小関係が-Switch オプション未指定時とは逆転することがあります。
- 日本語データや数字データ中の印字可能文字は，照合順序が切り替わることで，-Switch オプション未指定時とは大小関係が逆転することがあります。

(13) -V3Rec オプション

(a) 形式

```
-V3Rec, {Fixed | Variable}
-noV3Rec
```

(b) 機能

メインフレーム（VOS3）COBOL85 の固定長または可変長レコード形式のプログラムを，VOS3 COBOL85 の日本語文字の扱いに合わせてコンパイルするためのオプションです。

-V3Rec,Fixed

VOS3 COBOL85 の固定長レコード形式の COBOL 原始プログラムを，VOS3 COBOL85 の日本語文字の扱いに合わせてコンパイルします。

-V3Rec,Variable

VOS3 COBOL85 の可変長レコード形式の COBOL 原始プログラムを，VOS3 COBOL85 の日本語文字の扱いに合わせてコンパイルします。

-V3Rec,Variable オプションは，-CompatiV3 オプションと同時に指定できます。また，-CompatiV3 オプションを指定すると，-V3Rec,Variable オプションが仮定されます。

-noV3Rec

-V3Rec オプションの指定を打ち消します。

(c) 注意事項

- -V3Rec オプションと-EquivRule オプションを重複指定した場合，あとに指定したオプションを優先します。
- このオプションを指定した場合，このシステムの標準コードと拡張コードの等価規則は適用されません。また，日本語文字定数に標準コードの空白を書いてもエラーとはなりません。ただし，日本語文字定数に空白以外の標準コード文字を書けますが，警告レベルのエラーとなります。なお，日本語文字定数の標準コード文字数は偶数としてください。日本語文字定数の標準コード文字数が奇数の場合，警告レベルのエラーとし，日本語文字定数に標準コードの空白を追加します。
- 日本語空白は半角空白 2 個と等価とみなし，機能キャラクタが付加されたものとみなす対象としません。日本語空白を使用しているメインフレームの固定長形式の原始プログラムを-V3Rec,Fixed オプ

ションを指定してコンパイルする場合は，カラムずれなどの影響がないか確認する必要があります。機能キャラクタについては，マニュアル「COBOL2002　言語　拡張仕様編」「24.3.3(2)　文字集合」を参照してください。また，日本語空白を拡張コードとして扱いたいときは，-V3Rec オプションと合わせて-V3RecFCSpace オプションを指定してください。-V3RecFCSpace オプションの詳細については，「(14)　-V3RecFCSpace オプション」を参照してください。

- VOS3 COBOL85 の固定長レコード形式の原始プログラムと可変長レコード形式の原始プログラムは，このオプションを指定したときだけコンパイルできます。

  したがって，COPY 文で複写した原始プログラムを含め，固定長レコード形式の原始プログラムと可変長レコード形式の原始プログラムは同時にコンパイルできません。
- このオプションを指定した場合，72 カラムより前に改行文字がある COBOL 原始プログラムは，改行文字から 72 カラムまで空白に置き換えてコンパイルされます。
- このオプションは，自由形式正書法で書かれた COBOL 原始プログラムをコンパイルする場合には指定できません。指定した場合，エラーとなってコンパイルが中止されます。
- このオプションを指定した場合，英数字定数の分離符には，アポストロフィ（'）だけが使用できます。

(14)　-V3RecFCSpace オプション

(a)　形式

```
-V3RecFCSpace
-noV3RecFCSpace
```

(b)　機能

-V3RecFCSpace

-V3Rec オプションでの空白に関する機能キャラクタの扱いを，VOS3 COBOL85 と同等にします。機能キャラクタが付加されたものとみなす条件については，「(d)　機能キャラクタが付加されたものとみなす条件」を参照してください。

-noV3RecFCSpace

-V3RecFCSpace オプションの指定を打ち消します。

(c)　注意事項

- -V3RecFCSpace オプションの指定は，-V3Rec オプションの指定が有効な場合にだけ有効となります。

(d)　機能キャラクタが付加されたものとみなす条件

このコンパイラは，-V3Rec オプションの指定が有効な場合に，-V3RecFCSpace オプションの有無で，空白に関する機能キャラクタの扱いが異なります。次の条件文中の太字の部分は，-V3RecFCSpace オプションの有無での相違点です。

なお，これ以降「KEIS コード開始」と「EBCDIK コード開始」を表す 2 バイトの機能キャラクタをそれぞれ，[漢]と[E]で表します。また，改行を[改行]で表します。

- -V3RecFCSpace オプション指定がない場合
  - **日本語空白を除く**日本語文字列の先頭文字の直前に[漢]が付加されたものとみなします。
  - 直前に[漢]が付加されたものとみなした日本語文字列のあとに，最初に現れた半角空白以外の半角文字の直前に[E]が付加されたものとみなします。
  - [漢]が付加されたものとみなしたあとは，[E]が付加されたものとみなすまで，同じ行には[漢]が付加されたものとみなしません。
- -V3RecFCSpace オプション指定がある場合
  - **日本語空白を含む**日本語文字列の先頭文字の直前に[漢]が付加されたものとみなします。
  - 直前に[漢]が付加されたものとみなした日本語文字列のあとに，最初に現れた半角空白以外の半角文字の直前に[E]が付加されたものとみなします。**また，日本語文字列の直後が改行か日本語文字列と改行との間が半角空白だけのとき，日本語文字列の直後に[E]が付加されたものとみなします。**
  - [漢]が付加されたものとみなしたあとは，[E]が付加されたものとみなすまで，同じ行には[漢]が付加されたものとみなしません。

-V3Rec オプションの指定が有効な場合に，-V3RecFCSpace オプションの有無での機能キャラクタが付加されたものとみなした状態を次の例に示します。

（例 1）日本語空白に関する機能キャラクタの扱い

COBOL 原始プログラム

```
000100 01△DATA1△PIC△N(5)△VALUE△N'▲あいう▲'.
```

-V3Rec,Fixed オプション指定ありで， -V3RecFCSpace オプション指定がない場合

```
000100 01△DATA1△PIC△N(5)△VALUE△N'▲[漢]あいう▲[E]'.
```

-V3Rec,Fixed オプション指定ありで，-V3RecFCSpace オプション指定がある場合

```
000100 01△DATA1△PIC△N(5)△VALUE△N'[漢]▲あいう▲[E]'.
```

（凡例）

▲：日本語空白

△：半角空白

-V3RecFCSpace オプション指定がない場合，日本語空白を日本語文字列の先頭文字と扱いません。このため，日本語空白以外の先頭文字「あ」の直前に[漢]が付加されたものとみなします。また，直前に[漢]が付加されたものとみなした日本語文字列「あいう」のあとで最初に現れた半角空白以外の半角文字「'」の直前に[E]が付加されたものとみなします。

-V3RecFCSpace オプション指定がある場合，日本語空白を日本語文字列の先頭文字と扱い，直前に[漢]が付加されたものとみなします。また，直前に[漢]が付加されたものとみなした日本語文字列「▲あいう▲」のあとで最初に現れた半角空白以外の半角文字「'」の直前に[E]が付加されたものとみなします。

（例 2）空白と改行に関する機能キャラクタの扱い

COBOL 原始プログラム

```
000100 01△DATA1△PIC△N(5)△VALUE△N'あいう△△[改行]
000200-'△△'.
```

-V3Rec,Fixed オプション指定ありで， -V3RecFCSpace オプション指定がない場合

```
000100 01△DATA1△PIC△N(5)△VALUE△N'[漢]あいう△△[改行]
000200-'△△'.
```

-V3Rec,Fixed オプション指定ありで，-V3RecFCSpace オプション指定がある場合

```
000100 01△DATA1△PIC△N(5)△VALUE△N'[漢]あいう[E]△△[改行]
000200-'△△'.
```

（凡例）

▲：日本語空白

△：半角空白

-V3RecFCSpace オプション指定がない場合，日本語文字列「あいう」のあとには，半角空白以外の半角文字がないため[E]が付加されたものとみなしません。

-V3RecFCSpace オプション指定がある場合，日本語文字列「あいう」と[改行]の間が半角空白だけであるため，日本語文字列「あいう」の直後に[E]が付加されたものとみなします。

(15) -V3RecEased オプション

(a) 形式

```
-V3RecEased {,QuoteCheck | ,WordCheck} +
-noV3RecEased
```

(b) 機能

-V3RecEased,QuoteCheck

-CompatiV3 または-V3Rec オプションを指定したとき，定数の分離符のチェックを緩和します。

-V3RecEased,QuoteCheck オプションが有効な場合，-DoubleQuote オプションに関係なく，定数の分離符は，アポストロフィ（'）と引用符（"）のどちらでも指定できます。

-V3RecEased,WordCheck

-CompatiV3 または-V3Rec オプションを指定したとき，語または PICTURE 文字列の制限値のチェックを緩和します。

-V3RecEased,WordCheck オプションが有効な場合，語または PICTURE 文字列の制限値を超えたときのコンパイルエラーを，重大エラーから警告エラーに緩和できます。

-noV3RecEased

-V3RecEased オプションの指定を打ち消します。

(c) 注意事項

- -V3RecEased オプションと-V3Spec オプションを同時に指定したとき，-V3RecEased オプションの指定は無効になります。
- -V3RecEased オプションの指定は，-V3Rec オプションの指定が有効な場合にだけ有効です。
- -V3RecEased,QuoteCheck オプションの指定に関係なく，SQL 文中の定数の分離符はアポストロフィ（'）だけ使用できます。
- -V3RecEased,QuoteCheck オプションは，翻訳指令に対しても有効です。

(16) -DoubleQuote オプション

(a) 形式

```
-DoubleQuote
-noDoubleQuote
```

(b) 機能

-DoubleQuote

VOS3 COBOL85 互換に関連するオプションである-V3Rec，-CompatiV3 のどちらかを指定したときに，英数字定数の分離符として引用符（ " ）を用いることを指定するオプションです。このオプションを指定しないときは，アポストロフィ（ ' ）を分離符として用います。

-V3Rec オプション，-CompatiV3 オプションのどちらも指定していない場合は，-DoubleQuote オプションを指定しても無効となります。この場合は，引用符（ " ），アポストロフィ（ ' ）のどちらも分離符として使用できます。

ただし，SQL 文中の定数の分離符は，-DoubleQuote オプションの指定に関係なく，常にアポストロフィ（ ' ）を使用します。

また，QUOTE 表意定数は，-V3Rec オプション，-CompatiV3 オプションの指定に関係なく，-DoubleQuote オプション指定時は引用符（ " ）を意味し，-DoubleQuote オプション指定なしのときはアポストロフィ（ ' ）を意味します。

なお，このオプションは，翻訳指令に対しても有効となります。

-noDoubleQuote

-DoubleQuote オプションの指定を打ち消します。

(17) -BigEndian オプション（Linux で有効）

(a) 形式

```
-BigEndian {,Bin|,Float} +
-noBigEndian
```

(b) 機能

用途（USAGE 句）が 2 進または浮動小数点のデータ項目の形式を，ビッグエンディアン形式にするためのオプションです。

-BigEndian,Bin

用途が 2 進（BINARY，COMP，COMP-4）のデータ項目の形式を，ビッグエンディアン形式にします。

このオプションを指定した場合，2 進項目を演算などで参照すると，実行性能が劣化します。このため，必要のないかぎりこのオプションは指定しないでください。

-BigEndian,Float

用途が浮動小数点（COMP-1，COMP-2）のデータ項目の形式を，ビッグエンディアン形式にします。

このオプションを指定した場合，浮動小数点を演算などで参照すると，実行性能が劣化します。このため，必要のないかぎりこのオプションは指定しないでください。

-noBigEndian

-BigEndian オプションの指定を打ち消します。

(c) 注意事項

ビッグエンディアン形式とリトルエンディアン形式との違い

コンピュータで扱うバイナリ形式のデータ（2 進データ，内部浮動小数点数字データ）には，ビッグエンディアン形式とリトルエンディアン形式の 2 種類があります。

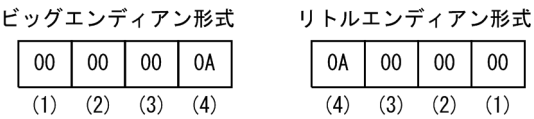
これは，マイクロプロセッサのアーキテクチャの違いによるもので，システムによって採用されるエンディアン形式が異なります。システムごとのエンディアン形式を次に示します。

システム	エンディアン形式
Windows Linux	リトルエンディアン
HP-UX AIX Solaris	ビッグエンディアン

ビッグエンディアン形式とリトルエンディアン形式では，バイナリデータがメモリ上に配置されるときの順序が次の例のように逆になります。

（例）

10（=H'0000000A'）が 4 バイトのメモリに配置された場合の違い



ビッグエンディアン形式では左側から右側へ 1 バイトずつ配置されるのに対して，リトルエンディアン形式では右側から左側へ 1 バイトずつ配置されます。このため，ビッグエンディアン形式のシステムと

リトルエンディアン形式のシステムで同じデータを扱う場合でも，参照のしかたによっては違った結果になることがあります。-BigEndian オプションは，このような場合にデータを正しく処理できるようにするために使用します。

-BigEndian オプションの使い方

リトルエンディアン形式のシステムで 2 進データをそのまま参照したり，リトルエンディアン形式のシステムで作成したファイルを参照したりする場合は，このオプションを指定する必要はありません。しかし，REDEFINES 句などで英数字データを 2 進データとして参照したり，ビッグエンディアン形式のシステムで作られたファイル（可変長ファイルは対象外）を参照したりする場合は，ビッグエンディアン形式のシステムとリトルエンディアン形式のシステムで 2 進データ，内部浮動小数点数字データの表現形式が異なるため，このオプションを指定する必要があります。このオプションは次のような場合に指定してください。

- ビッグエンディアン形式のシステム上で開発した 2 進データや内部浮動小数点数字データのメモリ上での配置を意識しているようなプログラムをリトルエンディアン形式のシステム上に移植する場合
- ビッグエンディアン形式のシステム上で作成した 2 進データや内部浮動小数点数字データを含むファイルをそのままリトルエンディアン形式のシステム上へ移行し，リトルエンディアン形式のシステム上のプログラムで参照する場合
- ビッグエンディアン形式のシステム上で開発した REDEFINES 句でほかの形式のデータ項目を 2 進データとして参照するようなプログラムをリトルエンディアン形式のシステム上に移植する場合

使用上の注意

-BigEndian オプションの使用上の注意を次に示します。

- C 言語と連携している場合，-BigEndian,Bin オプション指定時には，C プログラムとインタフェースを取る 2 進データは COMP-5 で定義してください。
- コマンド行に指定した引数を受け取る場合（16.2.2　引数の受け取り方法（C 言語インタフェースに従った形式の場合）または 16.2.3　引数の受け取り方法（VOS3 インタフェースに従った形式の場合）を参照）-BigEndian,Bin 指定時には，連絡節の 2 進データは，COMP-5 で定義してください。
- 2 進項目を引数とするサービスルーチンは，受け取る項目を COMP-5 で定義してください。
- CALL 文で LENGTH OF 一意名を指定した場合，呼び出し先プログラムは，受け取る項目を COMP-5 で定義してください。
- AIX(32)で XMAP3 を使用する場合は，XMAP3 のオプションと合わせる必要があります。
- 索引ファイル，および整列併合機能で使用するキーデータ項目の 2 進データ項目は，リトルエンディアン形式でキーを大小比較します。このため，このオプションを指定したビッグエンディアン形式では，正常に動作できません。

  キーデータ項目の 2 進データ項目は COMP-5 で定義するか，またはリトルエンディアン形式データとしてください。

(18) -VOSCBL オプション

(a) 形式

```
-VOSCBL {,OccursKey|,ReportControl|,DataComm|,RedefinesData|,AssignDataToDevice|,Evalua
teWhenOther} +
-noVOSCBL
```

(b) 機能

メインフレーム互換機能を有効にします。

-VOSCBL,OccursKey

OCCURS 句の KEY IS 指定のデータ名の名前の有効範囲を，その OCCURS 句のある記述項または，それに従属する記述項とします。

OCCURS 句の KEY IS 指定のデータ名は，修飾してもその修飾は無視されます。

（例 1）

```
       WORKING-STORAGE SECTION.
         01 DAT0.
           02 DAT1 PIC X.    ・・・・・・・・・・・・・・・(1)
         01 DAT2.
           02 DAT3 OCCURS 10 ASCENDING KEY DAT1.   ・・・(2)
             03 DAT1  PIC X.       ・・・・・・・・・・・・(3)
             03 DAT5  PIC X.
```

- -VOSCBL,OccursKey オプションの指定がある場合は，（2）の DAT1 は修飾しなくても，（2）の従属項目として定義された（3）の DAT1 が参照されます。
- -VOSCBL,OccursKey オプションの指定がない場合は，DAT1 が（1）と（3）にあるため，名前が一意になりません。一意に参照するためには，（2）の DAT1 を「OF DAT3」で修飾する必要があります。

（例 2）

```
       WORKING-STORAGE SECTION.
         01 DAT0.
           02 DAT1 PIC X.      ・・・・・・・・・・・・・・・・(1)
         01 DAT2.
           02 DAT3 OCCURS 10 ASCENDING KEY DAT1 OF DAT0.・・(2)
             03 DAT1  PIC X.       ・・・・・・・・・・・・・(3)
             03 DAT5  PIC X.
```

- -VOSCBL,OccursKey オプションの指定がある場合は，（2）の DAT1 を修飾（OF DAT0）してもその修飾は無視されるため，（2）の DAT1 は，（2）の従属項目として定義された（3）の DAT1 が参照されます。
- -VOSCBL,OccursKey オプションの指定がない場合は，（2）の DAT1 の修飾（OF DAT0）を無視されないため，（1）の DAT1 が参照されます。しかし，KEY IS 指定のデータ名は，OCCURS 句のある記述項または，それに従属する記述項でなければなりません。

したがって，(1) の DAT1 は (2) の従属項目でないため，(2) の修飾を「OF DAT3」に変更する必要があります。

-VOSCBL,ReportControl

制御脚書きの印刷中に改ページが起こった場合，ページの頭書きで SOURCE 句によって制御用データ項目を参照しているときは，その制御用データ項目の値を元の値で参照します。

-VOSCBL,DataComm

データコミュニケーション機能の通信記述項で，次のどちらかに該当するときの初期値を，メインフレーム(VOS3/VOS1)の XDM/DCCM データコミュニケーション機能を使用する場合の値とします。

- 通信記述項の句の指定を省略している。
- 通信文の実行前に，通信記述項の句に指定されたデータ名に初期値を設定していない。

なお，このオプションの指定がない場合は，TP1/Server Base を使用する場合の初期値とします。

-VOSCBL,RedefinesData

REDEFINES 句の右辺のデータ名が未定義のとき，直前の同一レベル番号のデータ名を REDEFINES 句の右辺として仮定します。

(例)

```
 4              WORKING-STORAGE SECTION.
 5                01 DAT1.
 6                  02 DAT2.
 7                    03 DAT3  PIC XX.
 8                  02 DAT4  REDEFINES UNDEF.
 9                    03 DAT5  PIC X.
10                    03 DAT6  PIC X.
```

この例では，8 行目の REDEFINES 句の右辺のデータ名 UNDEF が定義されていません。-VOSCBL,RedefinesData オプションが有効な場合，直前の同一レベル番号のデータ名 DAT2 を REDEFINES 句の右辺のデータ名として仮定します。-VOSCBL,RedefinesData オプションが無効な場合，8 行目の REDEFINES 句は無視されます。

-VOSCBL,AssignDataToDevice

ASSIGN 句の利用者定義語がデータ名と一致していても外部装置名とみなします。

(例)

```
 4       ENVIRONMENT         DIVISION.
 5        INPUT-OUTPUT       SECTION.
 6         FILE-CONTROL.
 7           SELECT  TEST-F  ASSIGN    SYS001.
 8       DATA                DIVISION.
 9        FILE               SECTION.
10         FD    TEST-F      LABEL  RECORD  STANDARD.
11          01   TEST-R      PIC  X(80).
12       WORKING-STORAGE    SECTION.
13          01   SYS001      PIC  X(16).
```

この例では，7 行目の ASSIGN 句でデータ名「SYS001」を定義しています。

-VOSCBL,AssignDataToDevice オプションが有効な場合は，「SYS001」を外部装置名とみなして，お知らせメッセージを出力します。-VOSCBL,AssignDataToDevice オプションが無効な場合は，「SYS001」をデータ名とみなします。

-VOSCBL,EvaluateWhenOther

EVALUATE 文の WHEN 指定と WHEN OTHER 指定の間の無条件文が省略された場合に，-VOSCBL,EvaluateWhenOther オプションが有効なときは，警告エラーメッセージを出力して CONTINUE 文を仮定します。-VOSCBL,EvaluateWhenOther オプションが無効なときは，重大エラーメッセージを出力します。

-noVOSCBL

-VOSCBL オプションの指定を打ち消します。

(c) 注意事項

- このオプションは，-CompatiV3 オプションと同時に指定してください。-CompatiV3 オプションの指定がない場合は，エラーメッセージが出力されます。
- 次のオプションは-CompatiV3 オプションを無効とするため，このオプションおよび-CompatiV3 オプションと次のオプションを同時に指定しないでください。-CompatiV3 オプションの指定が無効となり，U レベルのエラーになります。

  -StdMIA　-Std85　-Std2002
- このオプションと-UniObjGen オプション，および-CompatiV3 オプションを同時に指定する場合，-CompatiV3 オプションが有効となるよう，コンパイラ環境変数 CBLV3UNICODE を指定してください。

(19) -PortabilityCheck オプション

メインフレームから移行する場合，ユーザが注意すべき個所にお知らせメッセージ（I レベルメッセージ）を出力し，ユーザの移行作業を支援します。このオプションを指定すると，COBOL2002 とメインフレームで動作の異なる可能性がある個所を対象に，お知らせメッセージを出力します。

なお，このオプションは，S レベル以上のエラーがないプログラムに適用してください。S レベル以上のエラーがあるプログラムで適用した場合の動作は，保証しません。

(a) 形式

```
-PortabilityCheck {,Literal|,Numeric} +
-noPortabilityCheck
```

(b) 機能

-PortabilityCheck,Literal

- -UniObjGen オプション指定時，英数字定数および日本語文字定数中の日本語または半角かたかなにお知らせメッセージを出力します。

- 次に示す 16 進定数にお知らせメッセージを出力します。
  16 進英数字定数，16 進数字定数，16 進日本語文字定数

-PortabilityCheck,Numeric

- RECORD KEY 句，ALTERNATE RECORD KEY 句の 2 進項目，内部浮動小数点項目にお知らせメッセージを出力します。
- 浮動小数点の演算が発生する個所にお知らせメッセージを出力します。
  浮動小数点の演算が発生する個所は次を対象とします。

1. 算術式，算術文に指定された浮動小数点項目や浮動小数点定数
   算術式が記述できる場所に書かれた単項の浮動小数点項目や浮動小数点定数も含みます。
   ただし，受け取り項目など，演算に関係しない浮動小数点項目は除外します。
2. べき乗の場合，べき数に指定されたデータ項目および算術式が小数点を持つ場合，または，べき数が小数点を含む数字定数である場合
3. 次の組み込み関数
   ACOS，ASIN，ATAN，COS，LOG，LOG10，MOD，RANDOM，REM，SIN，SQRT，STANDARD-DEVIATION，TAN，VARIANCE
4. 次のどれかが引数に指定されている組み込み関数
   浮動小数点項目※，べき乗が含まれる算術式，または除算が含まれる算術式
   注※
   ADDR 関数，および LENGTH 関数を除きます。

-noPortabilityCheck

-PortabilityCheck オプションの指定を打ち消します。

(20) -IgnoreAPPLY,FILESHARE オプション

(a) 形式

```
-IgnoreAPPLY,FILESHARE
-noIgnoreAPPLY
```

(b) 機能

-IgnoreAPPLY,FILESHARE

入出力管理記述項の APPLY FILE-SHARE 句を覚え書きとみなします。覚え書きとみなしたときは，コンパイル時にお知らせメッセージを出力します。

-noIgnoreAPPLY

-IgnoreAPPLY オプションの指定を打ち消します。

(21) -International オプション

(a) 形式

```
-International
-noInternational
```

(b) 機能

-International

インターナショナリゼーション機能を使用するときに指定します。インターナショナリゼーション機能の適用範囲は，ロケールが日本語または英語に対応するときだけです。インターナショナリゼーション機能を使用すれば，通貨記号を自動的に変換できます。

-noInternational

-International オプションの指定を打ち消します。

(22) -EucPosition オプション（AIX で有効）

(a) 形式

```
-EucPosition
-noEucPosition
```

(b) 機能

-EucPosition

EUC コード使用時，見かけ上の文字位置で固定形式正書法の境界を決定するときに指定します。このオプションを指定した場合の半角かたかな文字の扱いは，正書法の境界を決定するときには見かけ上の 1 バイトとして扱われますが，文字定数中の半角かたかな文字は 2 バイトとして扱われます。

-noEucPosition

-EucPosition オプションの指定を打ち消します。

(c) 注意事項

- -EucPosition オプションは，EUC コード使用時，固定形式正書法の COBOL ソースファイルに対してだけ有効です。
- -EucPosition オプションを指定した場合，次の項目中に 3 バイトコード文字は使用できません。
  - COBOL ソースファイル，および登録集原文中
  - COBOL ソースファイルのファイル名
  - カレントディレクトリ，および COBOL ソースファイルがあるディレクトリのパス名
  - 登録集名の環境変数，および環境変数 CBLLIB に指定したディレクトリのパス名

- 環境変数 CBLFIX，および環境変数 CBLFREE に指定したサフィックス名

32.5.13　リスト出力の設定

リスト出力を設定するコンパイラオプションについて，説明します。

(1)　-SrcList オプション

(a)　形式

```
-SrcList, {OutputAll | CopyAll | CopySup | NoCopy}　〔,NoFalsePath〕　〔,DataLoc〕
-noSrcList
```

(b)　機能

コンパイルリストの出力形式を指定するオプションです。このオプションを指定しなかった場合，コンパイルリストは，出力されません。

出力されるリストの内容については，「付録 D　コンパイルリスト」を参照してください。また，コンパイルリストに関連する翻訳指令については，「2.3.4　コンパイルリストに関連する翻訳指令」を参照してください。

-SrcList,OutputAll

すべての情報を出力します。

原始プログラム中に，COPY 文の SUPPRESS 指定や LISTING OFF 指令がある場合でも，すべてのソースをコンパイルリストに展開します。

-SrcList,CopyAll

SUPPRESS 指定を無視して，すべての COPY 文を強制的に展開します。

-SrcList,CopySup

プログラムに SUPPRESS 指定があるときだけ，COPY 文の展開を抑止します。SUPPRESS 指定がない COPY 文は，すべて展開します。

-SrcList,NoCopy

すべての COPY 文の展開を抑止します。

-SrcList,xxxxx,NoFalsePath

条件翻訳の無効行を出力しません。

xxxxx には，CopyAll，CopySup，NoCopy のどれかを指定します。

-SrcList,OutputAll,NoFalsePath と指定した場合は，OutputAll サブオプションが有効となり，NoFalsePath サブオプションは無効になります。

-SrcList,xxxxx,DataLoc

コンパイルリスト（原始プログラムリスト）にデータ項目の相対位置と長さ（バイト）を 16 進数で表示します。相対位置は，データ部のファイル節／作業場所節／局所場所節の各節の先頭からの位置を表示します。

xxxxx には，OutputAll，CopyAll，CopySup，NoCopy のどれかを指定します。

S レベル／U レベルのコンパイルエラーが発生した場合は，指定があっても相対位置は表示しません。なお，相対位置の表示については，「付録 D.2　リストの見方」の「(4)　相対位置表示時の原始プログラムリスト」を参照してください。

-noSrcList

-SrcList オプションの指定を打ち消します。

(c)　注意事項

NoFalsePath サブオプションまたは，DataLoc サブオプションを指定した場合，-noSrcList オプションの指定がないかぎり，NoFalsePath サブオプション，および DataLoc サブオプションの指定は有効となります。

NoFalsePath サブオプション，および DataLoc サブオプションの指定を打ち消すときは，-noSrcList オプションを指定してください。

（例 1）　NoFalsePath サブオプション，および DataLoc サブオプション指定が打ち消されない場合

```
ccbl2002 -SrcList,OutputAll,NoFalsePath,DataLoc,-SrcList,CopyAll …
```

（コンパイル結果）

```
-SrcList,CopyAll,NoFalsePath,DataLoc
```

```
ccbl2002 -SrcList,OutputAll,NoFalsePath -SrcList,CopySup -SrcList,CopyAll,DataLoc
-SrcList,CopySup -SrcList,CopyAll …
```

（コンパイル結果）

```
-SrcList,CopyAll,NoFalsePath,DataLoc
```

（例 2）NoFalsePath サブオプション，および DataLoc サブオプション指定が打ち消される場合

```
ccbl2002 -SrcList,CopySup,NoFalsePath,DataLoc,-noSrcList -SrcList,CopyAll …
```

（コンパイル結果）

```
-SrcList,CopyAll
```

(2) -ErrSup オプション

(a) 形式

```
-ErrSup {,I|,W} +
-noErrSup
```

(b) 機能

コンパイル時にＩレベルまたは W レベルメッセージの出力を抑止します。

-ErrSup,I

コンパイル時にＩレベル（お知らせ）メッセージの出力を抑止します。

-ErrSup,W

コンパイル時に W レベル（警告）メッセージの出力を抑止します。

-noErrSup

-ErrSup オプションの指定を打ち消します。

(c) 注意事項

- -ErrSup,W オプションを指定した場合，利用者が注意する必要があるエラー（コンパイラによる解釈の変更や，中間結果けた数制限の適用など）の出力が抑止されます。このため，必要な場合以外は，-ErrSup,W オプションを指定しないようにしてください。
- -ErrSup オプションを指定してエラーメッセージの出力を抑止した場合，標準エラーおよびコンパイルリストにも，該当レベルのエラーが出力されません。
- 抑止されたエラーメッセージは，コンパイルリストに出力されるエラー件数にはカウントされません。
- コマンドライン上の記述誤りに対する W レベルメッセージは，抑止されません。

32.5.14 その他の設定

その他のコンパイラオプションについて，説明します。

(1) -Bin1Byte オプション

(a) 形式

```
-Bin1Byte
-noBin1Byte
```

(b) 機能

-Bin1Byte

1 バイトの 2 進項目を有効にします。

-noBin1Byte

-Bin1Byte オプションの指定を打ち消します。

(2) -JPN オプション（AIX で有効）

(a) 形式

```
-JPN, {Alnum | V3JPN | V3JPNSpace}
-noJPN
```

(b) 機能

-JPN,Alnum

日本語項目，日本語編集項目，および日本語文字定数をそれぞれ英数字項目，英数字編集項目，および英数字定数として扱います。

これは，VOS3 COBOL85 の LANGOPT=(-D)オプションと XCOBOL=(N)オプションを同時に指定したときと同じ動作になります。

なお，-CompatiV3 オプションを指定すると，このオプションが仮定されます。

-JPN,V3JPN

日本語項目，日本語編集項目，および日本語文字定数をそれぞれ英数字項目，英数字編集項目，および英数字定数として扱うようにするためのオプションです。ただし，LENGTH 関数の引数※，STRING 文，UNSTRING 文，または INSPECT 文では日本語項目，日本語編集項目，および日本語文字定数はそのままの属性として扱います。

また，日本語項目または日本語編集項目の部分参照は，英数字項目または英数字編集項目として扱いますが，最左端の文字位置と長さは日本語文字数を表します。これは，VOS3 COBOL85 の LANGOPT=(-D)オプションと XCOBOL=(-N)オプションを同時に指定したときと同じ動作になります。

注※

VOS3 COBOL85 では，LENGTH 関数の引数に日本語項目，日本語編集項目，および日本語文字定数は指定できません。

-JPN,V3JPNSpace

-CompatiV3 オプションと同時に指定した場合，日本語項目，日本語編集項目および日本語文字定数をそのままの属性で扱います。これは，VOS3 COBOL85 の LANGOPT=(D)オプションと XCOBOL=(-N)オプションを同時に指定したときと同じ動作になります。

VOS3 COBOL85 で LANGOPT=(D)オプションを指定した場合と COBOL2002 での動作の相違については，マニュアル「COBOL2002　言語　拡張仕様編」「付録 B　LANGOPT=(D)オプションと-JPN,V3JPNSpace オプションを指定した場合の仕様の相違」を参照してください。

このオプションを指定してコンパイルすると，日本語項目および日本語編集項目に対して，次に示すとおりに扱います。

- 表意定数 SPACE（SPACES）の扱い

  日本語項目および日本語編集項目に対する表意定数 SPACE（SPACES）は，日本語空白（X'8140'）となります。

- 日本語項目および日本語編集項目同士の転記の扱い

  日本語項目および日本語編集項目同士の転記で，受け取り側のけた数が送り出し側のけた数より長い場合，受け取り側の右側に日本語空白（X'8140'）を補います。

- 日本語項目および日本語編集項目同士の比較の扱い

  日本語項目および日本語編集項目同士の比較で，作用対象のけた数が等しくない場合，短い方の作用対象の右側に，長い方の作用対象のけた数に等しくなるまで日本語空白（X'8140'）があるものとみなして比較します。

-noJPN

-JPN オプションの指定を打ち消します。

(c)　注意事項

- -JPN,Alnum オプションと-JPN,V3JPN オプションを重複して同時に指定した場合，-JPN,V3JPN オプションが有効になり，-JPN,Alnum オプションが無効になります。
- コンパイラ環境変数 CBLV3UNICODE に YES を指定し，かつ-UniObjGen オプションを指定すると，-CompatiV3 オプションを指定しても-JPN,Alnum オプションは仮定されません。
- -JPN,V3JPNSpace オプションを指定するとき，-CompatiV3 オプションを同時に指定する必要があります。-CompatiV3 オプションの指定がない場合，コンパイルエラーとなります。
- -JPN,V3JPNSpace オプションと-JPN,Alnum オプション，-JPN,V3JPN オプションを同時に指定すると，-JPN,Alnum オプション，-JPN,V3JPN オプションは無効となります。

  また，-JPN,V3JPNSpace オプションと-CompatiV3 オプションを同時に指定すると，-CompatiV3 オプションが仮定する-JPN,Alnum オプションは無効となります。
- -JPN,V3JPNSpace オプションと-UniObjGen オプションを同時に指定すると，-JPN,V3JPNSpace オプションは無効となります。
- -JPN,Alnum オプションを指定するとき，動的長基本項目はすべて字類を英数字として扱います。また，PIC N が指定された動的長基本項目の LIMIT 指定の値は，その 2 倍の値が指定されたものと仮定します。

(3) -EquivRule オプション

(a) 形式

```
-EquivRule, {NotExtend | NotAny | StdCode}
-noEquivRule
```

(b) 機能

拡張コード文字と標準コード文字を等価とみなさないようにするためのオプションです。

標準コードおよび拡張コードについては，マニュアル「COBOL2002 言語 標準仕様編」「付録 I 用語の定義（Terms and Definitions)」を参照してください。

-EquivRule,NotExtend

拡張コード文字と標準コード文字を等価とみなしません。

-EquivRule,NotAny

拡張コード文字と標準コード文字を等価とみなしません。さらに，標準コードの英大文字と標準コードの英小文字も等価とみなしません。

この場合，予約語および文脈依存語はすべて標準コードの英大文字で記述してください。

-EquivRule,StdCode

拡張コード文字と標準コード文字を等価とみなしません。さらに，日本語文字定数中に標準コードの空白を書いてもエラーとしません。ただし，日本語文字定数中に空白以外の標準コード文字を書いた場合は，警告レベルのエラーとなります。なお，日本語文字定数の標準コード文字数は偶数としてください。日本語文字定数の標準コード文字数が奇数の場合，警告レベルのエラーとし，日本語文字定数に標準コードの空白を追加します。

また，72 カラムより前に改行文字がある COBOL 原始プログラムは，改行文字から 72 カラムまでを空白に置き換えてコンパイルします。

-noEquivRule

-EquivRule オプションの指定を打ち消します。

(4) -UscoreStart オプション

(a) 形式

```
-UscoreStart
-noUscoreStart
```

(b) 機能

-UscoreStart

先頭が下線の CALL 定数を指定できるようにします。

先頭が下線のプログラム名を CALL 定数で指定した場合，定数の先頭から'_'または'__'（下線 2 個）を除いた名称でプログラムを呼び出します。

-noUscoreStart

-UscoreStart オプションの指定を打ち消します。

(5) -BinExtend オプション

(a) 形式

```
-BinExtend
-noBinExtend
```

(b) 機能

-BinExtend

用途（USAGE 句）が 2 進（BINARY，COMP，COMP-4）のデータ項目に指定できる初期値（VALUE 句の値）を拡張します。

PICTURE 句で指定したけた数と VALUE 句に指定できる値との対応は次のとおりです。

PICTURE 句で指定したけた数	VALUE 句に指定できる値
1～4 けた	-2^{15}～$2^{15}-1$
5～9 けた	-2^{31}～$2^{31}-1$
10～18 けた	-999,999,999,999,999,999～999,999,999,999,999,999

-noBinExtend

-BinExtend オプションの指定を打ち消します。

(6) -MinusZero オプション

(a) 形式

```
-MinusZero
-noMinusZero
```

(b) 機能

符号付き内部 10 進項目または符号付き外部 10 進項目への転記で切り捨てが発生した結果，負の符号を持つゼロ（以下，-0 と表記します）が発生することがあります。この-0 は演算や比較で数値として参照する場合は，正の符号を持つゼロ（以下，+0 と表記します）と同じに扱われ，プログラムの動作には影響しません。しかし，DISPLAY 文で表示したり，上位の集団項目で参照したり，REDEFINES 句で別の属性として参照した場合に，実行結果が異なることがあります。-MinusZero オプションは，符号付き内部 10 進項目および符号付き外部 10 進項目に，ほかのデータ項目の値や演算結果を転記する場合に発生した-0 を

強制的に+0 に変換して，-0 の発生を防止するオプションです。他システムや他社の COBOL とのデータの互換性や移行性が向上します。

（例 1）

```
       01 A PIC S9(5) USAGE DISPLAY.
       01 B PIC S9(5) USAGE DISPLAY VALUE 0.
           COMPUTE A = B - 0.1.
```

右辺の式の結果は-0.1 になり，それを A に格納する際に，小数部が切り捨てられて-0 となる。※-MinusZero オプションを指定すると，-0 が+0 に変換されて A に格納される。

注※

ただし，演算結果が必ず-0 になるとは限りません。

（例 2）

```
       01 A PIC S9(4) USAGE DISPLAY.
       01 B PIC S9(5) USAGE DISPLAY VALUE -10000.
           MOVE B TO A.
```

A に格納する際に，10,000 の最上位けたの 1 が切り捨てられて-0 となる。

-MinusZero オプションを指定すると，-0 が+0 に変換されて A に格納される。

-MinusZero

負の符号を持つゼロ（-0）を強制的に正の符号を持つゼロ（+0）に変換します。

-noMinusZero

-MinusZero オプションの指定を打ち消します。

(c) 注意事項

- -MinusZero オプションは，符号付き外部 10 進項目への転記および符号付き内部 10 進項目への転記に対してだけ有効です。したがって，-MinusZero オプションを指定していても，REDEFINES 句や集団項目名で，10 進項目を 10 進項目以外の項目として転記した場合，-0（例えば，内部 10 進数 3 けたの場合には X'000D'）がそのまま転記されます。

(7) -TruncCheck オプション

(a) 形式

```
-TruncCheck [,Binary]
-noTruncCheck
```

(b) 機能

-TruncCheck

転記での送り出し側作用対象のサイズをチェックします。

送り出し側作用対象のけた数が受け取り側作用対象のけた数より大きい場合は，メッセージが出力されます。

-TruncCheck,Binary

-TruncCheck オプションのチェックに加え，送り出し側作用対象が 2 進項目で，受け取り側作用対象が外部 10 進項目／内部 10 進項目のとき，送り出し側作用対象の 2 進項目は格納可能な最大けた数でチェックします。

-noTruncCheck

-TruncCheck オプションの指定を打ち消します。

(c) -TruncCheck オプションのチェック対象

-TruncCheck オプションは，次の項目について，転記での送り出し側作用対象のサイズをチェックします。

チェック対象となる項目

- 外部 10 進項目
- 内部 10 進項目
- 2 進項目（1 バイト 2 進項目を含む）
- 数字定数（16 進数字定数を含む）
- 固定長集団項目
- 英数字項目
- 英数字定数（16 進英数字定数を含む）
- ALL 英数字定数
- 数字型，整数型，または英数字型の，組み込み関数※および利用者定義関数

注※

送り出し側が英数字型の組み込み関数で，かつ次に示す条件の場合はチェックしません。

- 引数が部分参照されていて，長さが可変である。
  長さが可変になるのは次のどちらかのときです。
  ・部分参照の長さがデータ名指定である
  ・部分参照の開始位置がデータ名指定で，長さが省略されている
- 組み込み関数が MAX 関数または MIN 関数で，データ名の引数が複数指定されている。
- 組み込み関数が TRIM 関数または SUBSTRING 関数である。

チェック対象となる転記の個所

- VALUE 句
- 画面節（WINDOW SECTION）の SOURCE 句※2
- ACCEPT 文の書き方 2（日付と時刻を取得する ACCEPT 文）

- INITIALIZE 文の REPLACING 指定
- MOVE 文
- READ 文の INTO 指定
- RELEASE 文の FROM 指定
- RETURN 文の INTO 指定
- SET 文の書き方 4（条件設定の SET 文）
- WRITE 文の FROM 指定
- COMPUTE 文（送り出し側作用対象が単一の数字項目や数字定数の場合）
- 上記以外の算術文（中間結果のけた数※1 が受け取り側作用対象のけた数より大きい場合※3）

注※1

中間結果のけた数については，「5.2.4　演算の中間結果」を参照してください。

注※2

AIX で有効です。

注※3

AIX(64)，Linux(x64)の場合，-MaxDigits38 オプションおよび-IntResult,DecFloat40 オプションを指定しているときは，常に中間結果のけた数の方が大きいと判定します。

受け取り側作用対象が英数字の場合

- 受け取り側作用対象が固定長集団項目のときはバイト数で比較されます。
- 送り出し側作用対象が数字で，受け取り側作用対象が英数字のときは，送り出し側作用対象はけた数（小数けたを含む）で比較します。
- -H8Switch オプションが指定してあり，送り出し側作用対象が数字で受け取り側作用対象が英数字のときは，数字項目はバイト数で比較されます。

受け取り側作用対象が数字の場合

- 整数けた数で比較します。
- 次のすべての条件を満たすときはチェックしません。

  ・-DigitsTrunc オプションの指定がある

  ・-MaxDigits38 オプションおよび-IntResult,DecFloat40 オプションを指定していない（AIX(64)，Linux(x64)の場合）

  ・受け取り側作用対象が 2 進項目である

  ・VALUE 句の転記ではない
- 送り出し側作用対象が 18 けた※を超える英数字項目のときはチェックしません。

注※

AIX(64)，Linux(x64)の場合，-MaxDigits38 オプションおよび-IntResult,DecFloat40 オプションを指定しているときは，38 けたとなります。

送り出し側が 2 進項目で，受け取り側が外部 10 進項目／内部 10 進項目の場合の規則（-TruncCheck,Binary オプション指定時のチェック項目）

- 整数けた数の比較で，送り出し側作用対象のけた数が大きいときは，メッセージを出力します（-TruncCheck と同じチェック）。
- 整数けた数の比較で，送り出し側作用対象のけた数が大きくなく，次に示す 2 進項目に格納可能な最大けた数が，受け取り側作用対象の整数けた数より大きいときは，メッセージを出力します（-TruncCheck,Binary 固有のチェック）。

PICTURE 句のけた数※1	2 進項目に格納できる最大けた数※2
1～2 けた	5 けた （-Bin1Byte 指定時は 3 けた）
3～4 けた	5 けた
5～9 けた	10 けた
10～18 けた	19 けた

注※1

整数部と小数部のけた数の合計値です。

注※2

小数部があるときは，小数部のけた数を除いたけた数です。

（例）

```
01 A PIC 9(4)V9(2) COMP.
```

上記の場合，整数部と小数部の PICTURE 句のけた数の合計値は 6 けたになります。2 進項目では PICTURE 句のけた数 6 けたで格納できる最大けた数は，10 けたになりますが，これから小数部のけた数 2 を除いた 8 けたを使用します。

-TruncCheck オプションと-TruncCheck,Binary オプションの相違点

次に-TruncCheck オプションと-TruncCheck,Binary オプションの相違点を示します。

項目	-TruncCheck	-TruncCheck,Binary
MOVE B5 TO E6※	ノーエラー	お知らせのメッセージを出力する
MOVE B6 TO E6※	ノーエラー	お知らせのメッセージを出力する
MOVE B7 TO E6※	お知らせのメッセージを出力する	お知らせのメッセージを出力する

注※

```
      01 B5  PIC 9(5) COMP.
      01 B6  PIC 9(6) COMP.
      01 B7  PIC 9(7) COMP.
      01 E6  PIC 9(6).
```

(8)　-LowerAsUpper オプション

(a)　形式

```
-LowerAsUpper
-noLowerAsUpper
```

(b)　機能

-LowerAsUpper

CALL 定数の英小文字を英大文字に変換した名称でプログラムを呼び出します。

-noLowerAsUpper

-LowerAsUpper オプションの指定を打ち消します。

(9)　-CBLVALUE オプション

(a)　形式

```
-CBLVALUE
-noCBLVALUE
```

(b)　機能

-CBLVALUE

環境変数 CBLVALUE を有効にするためのオプションです。

環境変数 CBLVALUE の詳細については，「32.6.3　コンパイラ環境変数の詳細」の「(19) CBLVALUE」を参照してください。

-noCBLVALUE

-CBLVALUE オプションの指定を打ち消します。

(10)　-Repository オプション

(a)　形式

```
-Repository, {Gen | Sup}
-noRepository
```

(b)　機能

-Repository,Gen

ソースファイルからリポジトリファイルを作成するときに指定します。この場合，オブジェクトファイルは作成されません。

なお，翻訳単位（プログラム定義を除く）が未完成でも，シグニチャと呼ばれるインタフェース部分が決まっていれば，リポジトリファイルを作成できます。詳細は，「33.3.2　リポジトリファイルの単独生成」を参照してください。

注意事項

- クラス定義，インタフェース定義，および関数定義が一つも格納されていないファイルに-Repository,Gen オプションを指定した場合，オプションの指定が無効となります。
- -Repository,Gen オプションと-Compile オプションが同時に指定された場合，-Repository,Gen オプションが有効になり，-Compile オプションが無効になります。

-Repository,Sup

リポジトリファイルを更新しません。

ただし，リポジトリファイルがない場合は，新規に作成します。

また，オブジェクトファイルは，生成されます。

-noRepository

-Repository オプションの指定を打ち消します。

(11)　-RepositoryCheck オプション

(a)　形式

```
-RepositoryCheck
-noRepositoryCheck
```

(b)　機能

-RepositoryCheck

同じソースファイル中の翻訳単位の定義と外部リポジトリ中の情報に相違があるかどうかをチェックし，相違がある場合には警告メッセージを出力します。-RepositoryCheck オプションを指定した場合，リポジトリファイルは更新されません。

詳細は，「33.2.3　リポジトリファイルの生成方法」を参照してください。

-noRepositoryCheck

-RepositoryCheck オプションの指定を打ち消します。

(12)　-Define オプション

(a)　形式

```
-Define 翻訳変数名〔=値〕〔,翻訳変数名〔=値〕〕…
-noDefine
```

(b) 機能

-Define 翻訳変数名〔=値〕〔,翻訳変数名〔=値〕〕…

翻訳変数名（条件翻訳で，ソース行の取り込みや読み飛ばしを制御する変数の名称）を定義します。

詳細は，「32.3.3 条件翻訳の利用」を参照してください。

-noDefine

-Define オプションの指定を打ち消します。

(c) 注意事項

- 翻訳変数名は，31 文字まで指定できます。32 文字以上の文字列を指定した場合，コンパイル時に警告メッセージが出力され，先頭の 31 文字だけが翻訳変数名として有効となります。
- 翻訳変数名は，-Define オプションに指定した文字列がそのまま使われます。この文字列には，COBOL の語の等価変換が適用されません。

  -Define オプションで定義した英小文字の翻訳変数名を COBOL プログラム中で参照するには，-EquivRule,NotAny オプションの指定が必要です。
- 翻訳変数の値は，160 バイトまで指定できます。161 バイト以上の文字列を指定した場合，コンパイル時に警告メッセージが出力され，先頭の 160 バイトだけが翻訳変数の値として有効となります。
- 翻訳変数名で使用できる文字は，COBOL の語で使用できる文字と同じです。ただし，コンパイラオプションの区切り文字となるイコール（=），コンマ（,）および半角空白文字は，指定できません。
- 翻訳変数の値を指定する場合の注意事項を，次に示します。

  1. 翻訳変数に指定した値は，すべて英数字定数として扱われます。

     （例 1）

     コンパイラオプションに，-Define VER=3 を指定したとき，ソース中に

     >>IF VER = '3'

     という記述があれば，上記条件が真となる。

     （例 2）

     コンパイラオプションに，-Define VER=5 を指定したとき，ソース中に

     >>IF VER = '3'

     という記述があれば，上記条件が偽となる。
  2. コンパイラオプションの区切り文字となるイコール（=），コンマ（,）は，指定できません。
  3. 文字列内に区切り文字（空白）を指定するときは，値全体をアポストロフィ（'）で囲む必要があります。このとき，アポストロフィは値として扱われません。

     （例）

     -Define DEF01='aaa bbb'

     翻訳変数名：DEF01

     値　　：aaa bbb

- -Define オプションで定義した英小文字の翻訳変数名を COBOL プログラム中で参照するには，-EquivRule,NotAny オプションを指定する必要があります。
- プログラムに記述した文字コードと同じ文字コードで，翻訳変数名および翻訳変数の値を指定する必要があります。

  なお，Linux の場合，プログラムに記述した文字コードとコマンドライン引数の文字コードが異なるため，翻訳変数名および翻訳変数の値に Unicode で多バイトとなる文字は含まないでください。多バイト文字を含む場合，指定した定義が有効になりません。

(13) -Details オプション

(a) 形式

```
-Details
-noDetails
```

(b) 機能

-Details

コンパイラオプションの詳細情報を標準エラー出力（stderr）へ出力します。

出力形式は，次のようになります。

形式

COBOL2002 インストールディレクトリ/bin/ccbl2002: ccbl2002 オプション群　COBOL ソースファイル名群

ccbl2002 オプション群

ccbl2002 コマンドが認識するオプションがすべて出力されます。ほかのオプションを指定したために仮定されたオプションも出力されます。反対に，ほかのオプションを指定したために無視されたオプションは出力されません。オプションの引数を指定した場合は，オプションの引数も出力されます。

COBOL ソースファイル名群

コンパイル対象の COBOL ソースファイル名がすべて出力されます。

（例）

入力

```
ccbl2002 -TDInf -SrcList,CopyAll -SimSub PROG1 -Details sample1.cbl
```

出力

```
COBOL2002 インストールディレクトリ/bin/ccbl2002: ccbl2002 -DebugInf -TDInf
-SrcList,CopyAll -Details -Optimize,1 -SimSub PROG1 -Main,System sample1.cbl
```

注

COBOL2002 インストールディレクトリは，OS によって異なります。

AIX(32)，Linux(x86)の場合

/opt/HILNGcbl2k

AIX(64)，Linux(x64)の場合

/opt/HILNGcbl2k64

-noDetails

-Details オプションの指定を打ち消します。

(14) -OldForm オプション

(a) 形式

```
-OldForm ’旧オプションの並び’
```

(b) 機能

-OldForm '旧オプションの並び'

旧形式のオプション（UNIX　COBOL85 用のコンパイラオプション）を指定できるようにします。

(c) 注意事項

- -OldForm オプションはアポストロフィ（'）で囲んで指定します。

  指定例を，次に示します。

  ```
  -OldForm ’-P3 -v -Mw sample.cbl’
  ```

- -OldForm オプションに指定する旧オプションの並びに，旧オプションの引数ではない単独ファイル名を指定した場合，そのファイル名は無視されます。ただし，旧オプションの引数として指定したファイル名については，有効となります。

  指定例を，次に示します。

  ```
  -OldForm ’-Mw sample.cbl subsample.cbl’
  ```

  sample.cbl は，-Mw オプションの引数として有効となりますが，subsample.cbl は旧オプションの引数でない単独のファイル名のため，無視されます。

(15) -Help オプション

(a) 形式

```
-Help
```

(b) 機能

ccbl2002 コマンドのヘルプを表示します。

(c) 注意事項

- -Help オプションを指定した場合，ほかのオプションやファイル名を指定しても無視されます。
- このオプションは，コマンドライン（ccbl2002 コマンド）で指定できます。

(16) -Profile オプション

(a) 形式

```
-Profile, {Prof | Gprof}
-noProfile
```

(b) 機能

-Profile,Prof（AIX で有効）

prof でのプロファイル用オブジェクトファイルを作成します。prof の詳細については，システムのマニュアルを参照してください。

-Profile,Gprof

gprof でのプロファイル用オブジェクトファイルを作成します。gprof の詳細については，システムのマニュアルを参照してください。

-noProfile

-Profile オプションの指定を打ち消します。

(c) 注意事項

- プロファイリング（実行回数および実行時間の分析）の対象になるのは，プログラム，利用者定義関数，メソッド，およびプロパティメソッドです。それぞれ，次の名前でプロファイル情報が出力されます。n は，オブジェクトモジュール内で名前を一意にするためにつけられた番号です。

  Linux の場合

  - 最外側のプログラム：プログラム名（ただし，-Main コンパイラオプションが適用された最外側のプログラムは main）
  - 入れ子のプログラム：_最外側プログラム名_n_入れ子のプログラム名
  - 利用者定義関数：利用者定義関数名
  - メソッド：_クラス名_n_メソッド名
  - プロパティメソッド：プロファイル対象外

  AIX の場合

  - 最外側のプログラム：.プログラム名（ただし，-Main コンパイラオプションが適用された最外側のプログラムは.main）
  - 入れ子のプログラム：.最外側プログラム名.n._入れ子のプログラム名

- 利用者定義関数：.利用者定義関数名
- メソッド：.クラス名.n._メソッド名
- GET プロパティメソッド：.クラス名.n._GET$プロパティ名
- SET プロパティメソッド：.クラス名.n._SET$プロパティ名

- COBOL プログラムが COBOL 実行時ライブラリを使って実装されている機能を使用している場合，プログラムの実行時間の情報には実行時ライブラリの実行時間は含まれません。純粋にプログラムのオブジェクトコードの実行時間だけが対象となります。
- ENTRY 文で指定した入口点に対しては，呼び出し回数に関する情報は，それぞれの入口点ごとに作成されます。しかし，実行時間の情報に関しては，ENTRY 文に指定した入口点が呼び出された場合も，ENTRY 文の入口点の実行時間の情報ではなく，ENTRY 文を含むプログラムの実行時間の情報に累積されます。
- AIX の場合，prof を使用して入れ子のプログラムやクラス定義を含む COBOL プログラムのプロファイル情報を出力するときは，prof に-g オプションを指定してください。-g オプションの詳細については，システムのマニュアルを参照してください。

(17) -UniObjGen オプション

(a) 形式

```
-UniObjGen
-noUniObjGen
```

(b) 機能

-UniObjGen

シフト JIS で記述された COBOL ソースから英数字定数を UTF-8 に，日本語文字定数を UTF-16LE または UTF-16BE に変換したオブジェクトファイルを生成します。

-noUniObjGen

-UniObjGen オプションの指定を打ち消します。

(c) 注意事項

- 日本語文字定数の文字コードのバイトオーダは，-UniEndian オプションの指定に従います。
- このオプションを指定する場合，次のオプションは同時に指定できません。同時に指定した場合，このオプションが有効となり，次のオプションは無効になります。

  -JPN　-CompatiV3　-V3Rec　-V3RecFCSpace

  ただし，コンパイラ環境変数 CBLV3UNICODE に YES を指定すると，このオプションを指定しても-CompatiV3，-V3Rec，および-V3RecFCSpace オプションは有効となります。
- AIX の場合，EUC 環境下で-UniObjGen オプションを指定した場合，コンパイルエラーとなります。

- Linux の場合，UTF-8 環境下で，-UniObjGen オプションを指定しないでコンパイルすると，コンパイルエラーとなります。また，-UniObjGen オプションを指定する場合は，環境変数 CBLSRCENCODING を同時に指定する必要があります。

(18) -UniEndian オプション

(a) 形式

```
-UniEndian, {Little | Big}
-noUniEndian
```

(b) 機能

-UniEndian,Little

日本語文字定数を UTF-16LE に変換したオブジェクトファイルを生成します。

-UniEndian,Big

日本語文字定数を UTF-16BE に変換したオブジェクトファイルを生成します。

-noUniEndian

-UniEndian オプションの指定を打ち消します。

(c) 注意事項

- このオプションは，-UniObjGen オプションが指定された場合だけ有効となります。
- -UniObjGen オプションが指定されていて，-UniEndian オプションが指定されていない場合，日本語文字定数は次のコードに変換されます。
  - AIX の場合：UTF-16BE
  - Linux の場合：UTF-16LE

(19) -Lx64ConventionCheck オプション（Linux(x64)で有効）

(a) 形式

```
-Lx64ConventionCheck
-noLx64ConventionCheck
```

(b) 機能

-Lx64ConventionCheck

Linux(x64) COBOL2002 で C 言語との連携を行うときに問題となる可能性がある場合（次に示す条件を満たす場合），警告レベルのエラーメッセージを出力します。詳細については，「19.1.2 C プログラムから COBOL プログラムを呼び出す方法」の「表 19-1 COBOL のデータ項目と C プログラムの型の対応」の注※1 を参照してください。

1. サイズが 16 バイト以下の集団項目を値渡し（BY VALUE）の引数および返却項目に指定している。
2. 1.の集団項目の従属項目に内部浮動小数点項目がある。

-noLx64ConventionCheck

-Lx64ConventionCheck オプションの指定を打ち消します。

(c) 注意事項

- CALL 文によって呼び出されるプログラムが COBOL プログラムであっても警告レベルのエラーメッセージが出力されます。

(20) -MaxDigits38 オプション（AIX(64)，Linux(x64)で有効）

(a) 形式

```
-MaxDigits38
-noMaxDigits38
```

(b) 機能

-MaxDigits38

数字項目および数字定数に指定できる数字の最大けた数を 18 けたから 38 けたに拡張します。外部 10 進形式および内部 10 進形式の数字項目と数字編集項目と固定小数点数字定数で，けた数を拡張できます。詳細については，「28. 数字項目のけた拡張機能（AIX(64)，Linux(x64)で有効）」およびマニュアル「COBOL2002 言語 拡張仕様編」「21. 数字項目のけた拡張機能」を参照してください。

-noMaxDigits38

-MaxDigits38 オプションの指定を打ち消します。

(c) 注意事項

- -IntResult,DecFloat40 オプションと同時に指定してください。-IntResult,DecFloat40 オプションの指定がない場合，エラーメッセージが出力されます。
- -MaxDigits38 オプション，-IntResult,DecFloat40 オプション，および-Optimize,3 オプションを同時に指定した場合，-Optimize,3 オプションは無効となり，-Optimize,2 オプションが仮定されます。
- -MaxDigits38 オプションを指定しても，19～38 けたの数字項目および数字定数が指定できない機能や文があります。詳細については，「28. 数字項目のけた拡張機能（AIX(64)，Linux(x64)で有効）」およびマニュアル「COBOL2002 言語 拡張仕様編」「21. 数字項目のけた拡張機能」を参照してください。

(21)　-IntResult,DecFloat40 オプション（AIX(64)，Linux(x64)で有効）

(a)　形式

```
-IntResult,DecFloat40
-noIntResult
```

(b)　機能

-IntResult,DecFloat40

算術演算の中間結果の表現形式を 40 けた 10 進浮動小数点形式にします。詳細については，「28.3　数字項目のけた拡張機能での演算の中間結果」を参照してください。

-noIntResult

-IntResult オプションの指定を打ち消します。

(c)　注意事項

- -MaxDigits38 オプションと同時に指定してください。-MaxDigits38 オプションの指定がない場合，エラーメッセージが出力されます。
- -IntResult,DecFloat40 オプション，-MaxDigits38 オプション，および-Optimize,3 オプションを同時に指定した場合，-Optimize,3 オプションは無効となり，-Optimize,2 オプションが仮定されます。
- -IntResult,DecFloat40 オプション，-MaxDigits38 オプション，および-Compati85,Power オプション（-Compati85,All 指定による仮定時を含む）を同時に指定した場合，-Compati85,Power オプションは無効となります。
- AIX(64)，Linux(x64)の場合，-IntResult,DecFloat40 オプションと-CompatiV3 オプションを同時に指定すると，-CompatiV3 オプションの仕様が一部無効となります。詳細については，「32.5.12 他システムとの移行の設定」の「(1)　-CompatiV3 オプション」を参照してください。
- AIX(64)，Linux(x64)の場合，コンパイラ環境変数 CBLV3UNICODE に YES の指定がないとき，-IntResult,DecFloat40 オプション，-MaxDigits38 オプション，-UniObjGen オプションおよび-CompatiV3 オプションを同時に指定すると，-CompatiV3 オプションを打ち消します。また，-CompatiV3 オプションによって仮定される-JPN オプションおよび-V3Rec オプションも打ち消します。

(22)　-LiteralExtend オプション

(a)　形式

```
-LiteralExtend,Alnum
-noLiteralExtend
```

(b) 機能

-LiteralExtend,Alnum

英数字定数の最大長を拡張します。拡張する項目と制限値および限界値を次に示します。

表 32-4 最大長を拡張する項目と制限値および限界値

最大長を拡張する項目	制限値および限界値	
	-LiteralExtend,Alnum なし -noLiteralExtend あり	-LiteralExtend,Alnum あり
英数字定数の長さ	1～160 文字（バイト）	1～8,191 文字（バイト）
STOP 文の定数に指定した英数字定数の長さ	1～160 文字（バイト）	1～8,191 文字（バイト）
定数指定する場合のプログラム名，メソッド名の長さ	1～160 文字（バイト）	1～1,024 文字（バイト）

ただし，次に示す項目では英数字定数の最大長は拡張されません。

表 32-5 最大長を拡張しない項目

最大長を拡張しない項目	制限値および限界値（-LiteralExtend の有無に関係なく同じ）
連結式で連結した英数字定数の長さ	2～1,024 文字（バイト）
指定できる文字位置のけた数が個別に規定されている構文	各構文の規定に従う
COPY 文，REPLACE 文の構文中に指定する原文語の長さ	1～322 文字（バイト）
翻訳指令行で使用する英数字定数の長さ	1～160 文字（バイト）
SQL 文中で使用する英数字定数の長さ	1～160 文字（バイト）

-noLiteralExtend

-LiteralExtend オプションの指定を打ち消します。

(c) 注意事項

- -LiteralExtend オプションは，次のオプションと背反関係にあり，同時に指定した場合は，-LiteralExtend オプションが無効となり，次のオプションが有効となります。

  -StdMIA -Std85 -Std2002 -V3Spec -Compati85,Syntax -CompatiV3

  -SimMain -SimSub -SimIdent
- COBOL プログラム中に COPY 文または REPLACE 文がある場合，322 バイトを超える英数字定数に対して，原文語の長さが 322 文字を超えている旨の警告エラーを出力することがありますが，プログラムの動作には影響しません。

(23) -SpaceAsZero オプション

(a) 形式

```
-SpaceAsZero
-noSpaceAsZero
```

(b) 機能

-SpaceAsZero

外部 10 進項目に空白文字（X'20'）データがあるとき，ゼロ（X'30'）とみなして比較，演算，転記を実行します。このオプションを指定すると，約 1.5 倍の実行性能劣化となります。このため，必要のないかぎり，このオプションは指定しないでください。

-noSpaceAsZero

-SpaceAsZero オプションの指定を打ち消します。

(c) 注意事項

- -SpaceAsZero オプションは，-Compati85,All オプションと同時に指定してください。-Compati85,All オプションの指定がない場合，エラーメッセージが出力されます。なお，-Compati85,All オプションと同じ意味となる-Compati85 のサブオプションが指定された場合，-SpaceAsZero オプションの指定が有効になります。
- -SpaceAsZero オプションと-Optimize,3 オプションを同時に指定した場合，-Optimize,3 オプションは無効となり，-Optimize,2 オプションが仮定されます。
- AIX(64)，Linux(x64)の場合，-SpaceAsZero オプションと，-Compati85,All，-MaxDigits38 および-IntResult,DecFloat40 オプションを同時に指定したとき，-IntResult,DecFloat40 オプションの仕様によって，-Compati85,Power サブオプションが無効となります。そのため，同時に指定した-Compati85,All オプションが無効となり，エラーメッセージが出力されます。

  -SpaceAsZero オプションを有効にする場合，-MaxDigits38 および-IntResult,DecFloat40 オプションを指定しないでください。
- -DebugData オプションが指定されている場合，比較，演算，転記の外部 10 進項目に含まれる空白文字（X'20'）は，-SpaceAsZero オプションを指定しても，不当なデータ（データ例外）として検出されます。
- 外部 10 進項目に空白文字（X'20'）データがあるとき，比較，演算，転記以外の実行結果（画面節（SCREEN SECTION および WINDOW SECTION），索引ファイルのキー項目，整列・併合処理のキー指定，DISPLAY 文，組み込み関数，埋め込み SQL 文などの実行結果）は，-SpaceAsZero オプションを指定しても保証しません。

(24)　-CheckUninitData オプション

(a)　形式

```
-CheckUninitData
-noCheckUninitData
```

(b)　機能

-CheckUninitData

ソースに閉じた初期化漏れチェック機能を有効にします。ソースに閉じた初期化漏れチェック機能については，「32.7.4　初期化漏れチェック機能」を参照してください。

-noCheckUninitData

-CheckUninitData オプションの指定を打ち消します。

(c)　注意事項

- このオプションは，-Compile,CheckOnly オプションと同時に指定する必要があります。-Compile,CheckOnly オプションを指定していない場合，または-Compile,CheckOnly オプションと，-Compile,NoLink オプションもしくは-Repository,Gen オプションを同時に指定した場合，-CheckUninitData オプションは無効となります。

(25)　-FunctionECSup,CodeConvErr オプション

(a)　形式

```
-FunctionECSup,CodeConvErr
-noFunctionECSup
```

(b)　機能

-FunctionECSup,CodeConvErr

組み込み関数の CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数で，変換前の文字に対応する変換後の文字を持たない文字が引数中にある場合，文字変換を行わないで，EC-ARGUMENT-FUNCTION／EC-ARGUMENT-IMP 例外が成立しないようにします。

組み込み関数の CONVERT-CODE 関数（Linux で有効），DISPLAY-OF 関数または NATIONAL-OF 関数のエラー情報の取得には，CBLCNVERRORINFO サービスルーチンを使用します。CBLCNVERRORINFO サービスルーチンの詳細は，「29.7.3　CBLCNVERRORINFO」を参照してください。

なお，このオプションの指定がない場合は，EC-ARGUMENT-FUNCTION／EC-ARGUMENT-IMP 例外が成立します。

-noFunctionECSup

-FunctionECSup オプションの指定を打ち消します。

32.6　コンパイラ環境変数

ここでは，ccbl2002 コマンドでコンパイルするときに使用する環境変数について，設定方法と環境変数の種類を説明します。

32.6.1　コンパイラ環境変数の設定方法

(1)　システムに従った環境変数の設定方法

ccbl2002 コマンドでのコンパイラ環境は環境変数で設定できます。

(a)　sh（B シェル）の場合

形式

```
環境変数=環境変数の値
export 環境変数
```

(2)　コンパイラ環境変数の規則

環境変数に YES を設定するとき，大文字と小文字は非等価とみなします。

32.6.2　コンパイラ環境変数の一覧

コンパイラ環境変数の一覧を，次に示します。

表 32-6　コンパイラ環境変数の一覧

環境変数	設定内容	OS	
		AIX	Linux
CBL_RDBSYS	HiRDB による索引編成ファイルで操作対象となるデータベースシステムの種別	○	×
CBL_UNINITDATA_BREAKOFF	初期化漏れチェック機能使用時，初期化漏れチェック処理を打ち切るかどうか	○	○
CBLCC	cc コマンドに渡すオプション列およびファイル名	○	○
CBLCOPT	ccbl コマンドに指定する旧形式のオプション列	○	×
CBLCOPT2002	ccbl2002 コマンドに指定する新形式のオプション列	○	○
CBLERRMAX	コンパイルを打ち切る S レベルのエラーの数	○	○
CBLFIX	固定形式正書法の COBOL ソースファイルの拡張子	○	○

環境変数	設定内容	OS	
		AIX	Linux
CBLFIXEDFORMLINE	固定形式正書法の 1 行の長さの制限値	○	○
CBLFREE	自由形式正書法の COBOL ソースファイルの拡張子	○	○
CBLINITVALUE	初期化属性プログラムの作業場所節にある，VALUE 句の指定がないデータ項目の初期値を NULL（X'00'）に設定し，初期化属性プログラムが呼ばれるたびに初期値を設定	×	○
CBLLIB	登録集原文の検索ディレクトリ	○	○
CBLPIDIR	プログラム情報ファイル（.cbp）の生成先ディレクトリ	○	○
CBLREP	リポジトリファイルの出力先のディレクトリ，およびリポジトリ段落で指定した翻訳単位名を含むリポジトリファイルを検索するディレクトリ	○	○
CBLSRCENCODING	Unicode 機能で入力する COBOL ソースの文字コード	×	○
CBLSYSREP	リポジトリファイルの参照時に検索するディレクトリ	○	○
CBLTAB	タブコードを空白に変換するときの値	○	○
CBLUNINITDATA_OUTRESULTLIST	初期化漏れチェック機能使用時，初期化漏れ確認結果一覧および初期化漏れ従属項目一覧を出力するかどうか	○	○
CBLUNINITDATA_OUTRESULTLISTDIR	初期化漏れチェック機能使用時，初期化漏れ確認結果一覧および初期化漏れ従属項目一覧の出力先ディレクトリ	○	○
CBLVALUE	VALUE 句の指定のないデータ項目の初期値	○	○
CBLV3UNICODE	-UniObjGen オプション指定時に，-CompatiV3 オプションおよび-V3Rec オプションの指定を有効とするかどうか	×	○
登録集環境変数	登録集原文の検索ディレクトリ	○	○

（凡例）

○：サポートしている

×：サポートしていない

32.6.3　コンパイラ環境変数の詳細

(1)　CBL_RDBSYS（AIX で有効）

HiRDB による索引編成ファイルの操作対象になるデータベースシステムを設定します。指定できる値は，HiRDB だけです。この環境変数の指定がない場合，および指定した値に誤りがある場合は，HiRDB が仮定されます。

（例）

```
CBL_RDBSYS=HiRDB
export CBL_RDBSYS
```

(2) CBL_UNINITDATA_BREAKOFF

初期化漏れチェック処理を打ち切るかどうかを指定します。

この環境変数の値に NO を指定すると，初期化漏れチェック処理は打ち切られません。プログラムを実行したときに通る可能性のある経路（制御ブロック）をすべて走査するまで，初期化漏れチェック処理を続行します。

次の場合は，この環境変数の指定は無効となり，初期化漏れチェック処理で走査した制御ブロックの数が内部的な上限を超えると，初期化漏れチェック処理は打ち切られます。

- この環境変数の指定がない，または環境変数の値に NO 以外を指定している
- -CheckUninitData オプションを指定していない

（例）

```
CBL_UNINITDATA_BREAKOFF=NO
export CBL_UNINITDATA_BREAKOFF
```

(3) CBLCC

cc コマンドに渡すオプション列およびファイル名を設定します。当環境変数に設定することによって，指定のオプションおよびオプション引数を c コンパイラに渡します。

コマンドの指定は，cc コマンドのコマンド列全体をアポストロフィ（'）で囲みます。

（例）

```
CBLCC='-c -o TEST  a.c b.s'
export CBLCC
```

(4) CBLCOPT（AIX で有効）

ccbl コマンドに指定するオプション列（コンパイラオプションの並び）を設定します。この環境変数に設定しておけば，ccbl のコマンドラインにオプションを指定する必要がなくなります。ただし，この環境変数は COBOL85 からの移行を目的とする場合にだけ使用してください。

環境変数 CBLCOPT にファイル名を指定した場合は，ファイル名が無視されます。

各オプションを空白で区切り，コマンド列全体をアポストロフィ（'）で囲みます。

（例）

```
CBLCOPT='-S1 -T4 -Ek'
export CBLCOPT
```

(5) CBLCOPT2002

ccbl2002 コマンドに指定するオプション列（コンパイラオプションの並び）を設定します。この環境変数に設定しておけば，ccbl2002 のコマンドラインにオプションを指定する必要がなくなります。

環境変数 CBLCOPT2002 にファイル名を指定した場合は，ファイル名が無視されます。

各オプションを空白で区切り，コマンド列全体をアポストロフィ（'）で囲みます。

（例）

```
CBLCOPT2002='-StdVersion,1 -DebugData -Switch,EBCDIK'
export CBLCOPT2002
```

(6) CBLERRMAX

コンパイルを打ち切る S レベルのエラーの数を設定します。設定した個数分のエラーが発生すると，メッセージが出力され，コンパイルが打ち切られます。

設定できる範囲は 0～999,999 で，省略時は 30 が仮定されます。コンパイルを続行し，すべてのエラーメッセージを出力したい場合は 0 を設定します。

なお，コンパイルリストを出力する場合（-SrcList オプションを指定した場合），この環境変数は無効となります。

（例）

```
CBLERRMAX=15
export CBLERRMAX
```

(7) CBLFIX

固定形式正書法で書かれた COBOL 原始プログラムとしてコンパイルする COBOL ソースファイルの拡張子を設定します。ただし，拡張子.cbl，.CBL，.cob，.ocb の付いたファイルは，ここで設定しなくても固定形式正書法で書かれた COBOL 原始プログラムとしてコンパイルされます。

拡張子は，先頭のピリオド（.）と 3 文字以内の英数字で指定します。複数の拡張子を設定する場合は，それぞれの拡張子を半角空白文字で区切り，全体をアポストロフィ（'）で囲みます。

（例）

```
CBLFIX=.fix
export CBLFIX
```

注意事項

- コンパイル後のファイル種別の変換

  環境変数 CBLFIX および CBLFREE に，コンパイラで使用するファイル（COBOL ソースファイルを除く）の拡張子は指定しないでください。COBOL ソースファイルが別のファイル種別に変換されることがあります。

(8) CBLFIXEDFORMLINE

固定形式正書法の 1 行の長さの制限値（バイト単位）を 80 または 255 で指定します。次の場合は，80 を仮定します。

- この環境変数の指定がない場合
- この環境変数に 80 または 255 以外を指定している場合
- 次のコンパイラオプションのどれかが有効である場合

  -V3Spec，-V3Rec,Fixed，-TDInf，-CVInf

（例）

```
CBLFIXEDFORMLINE=255
export CBLFIXEDFORMLINE
```

注意事項

- この環境変数に 255 を指定した場合でも，固定形式正書法のプログラム原文領域は，72 カラムまでです。
- この環境変数の指定値（制限値）を超える部分は無視されます。コンパイルリストの原始プログラムリストに，制限値を超える部分は表示されません。

(9) CBLFREE

自由形式正書法で書かれた COBOL 原始プログラムとしてコンパイルする COBOL ソースファイルの拡張子を設定します。ただし，拡張子.cbf，.ocf の付いたファイルは，ここで設定しなくても自由形式正書法で書かれた COBOL 原始プログラムとしてコンパイルされます。

拡張子は，先頭のピリオド（.）と 3 文字以内の英数字で指定します。複数の拡張子を設定する場合は，それぞれの拡張子を半角空白文字で区切り，全体をアポストロフィ（'）で囲みます。

（例）

```
CBLFREE='.aaa .bbb .ccc'
export CBLFREE
```

注意事項

- 環境変数 CBLFIX および環境変数 CBLFREE に，コンパイラで使用するファイル（COBOL ソースファイルを除く）の拡張子は指定しないでください。COBOL ソースファイルが別のファイル種別に変換されることがあります。

- 自由形式正書法で書かれた COBOL ソースに対するコンパイラオプションの制限
  自由形式正書法で書かれた COBOL ソースをコンパイルするとき，次のコンパイラオプションは指定できません。
  -StdVersion，-V3Rec，-CompatiV3，-StdMIA，-Std85

(10) CBLINITVALUE

初期化属性プログラムの作業場所節にある，VALUE 句の指定がないデータ項目の初期値を NULL（X'00'）に設定するときに，環境変数 CBLINITVALUE に NULL を設定します。初期化属性プログラムが呼ばれるたびに，初期値が設定されます。

環境変数 CBLINITVALUE に NULL 以外の値を設定した場合は，環境変数 CBLINITVALUE は無視されます。

（例）

```
CBLINITVALUE=NULL
export CBLINITVALUE
```

環境変数 CBLINITVALUE は，次のデータ項目には適用されません。

- VALUE 句の指定があるデータ項目
- ADDRESSED 句の指定があるデータ項目
- EXTERNAL 句の指定があるデータ項目

注意事項

初期化属性プログラムの作業場所節に対する初期化は，環境変数 CBLVALUE と環境変数 CBLINITVALUE が両方とも有効な場合と，どちらか片方だけが有効な場合とで初期値が異なります。詳細については，「18.4.1 プログラム属性」の「(2) 初期化属性プログラム」を参照してください。また，環境変数 CBLVALUE については，「(19) CBLVALUE」を参照してください。

(11) CBLLIB

登録集原文を検索するパスプレフィクスを設定します。ディレクトリを複数指定する場合は，それぞれパスプレフィックスをコロン（:）で区切って指定します。

（例）

```
CBLLIB=/usr/user/copylib
export CBLLIB
```

(12) CBLPIDIR

プログラム情報ファイル（.cbp）を任意のディレクトリに生成したい場合に設定します。プログラム情報ファイルは，テストデバッガを使用するときに必要となるファイルです。設定できるディレクトリは一つだけです。

（例）

```
CBLPIDIR=/temp
export CBLPIDIR
```

(13) CBLREP

リポジトリファイル（.rep）を任意のディレクトリに生成，または更新したい場合に，出力先のディレクトリを指定します。指定されたディレクトリは，リポジトリ段落で指定された名前の翻訳単位（関数定義，クラス定義，またはインタフェース定義）を含むリポジトリファイル参照時の検索対象となります。

詳細は，「33. 定義別のコンパイル方法とリポジトリファイル」を参照してください。

規則

- 指定するディレクトリが複数ある場合は，各ディレクトリをセミコロン（;）で区切って指定します。
- 指定したディレクトリ群に同じ名称のリポジトリファイルが複数ある場合，先に指定したディレクトリに含まれるリポジトリファイルを優先します。
- 生成または更新時に，同じ名称のリポジトリファイルがない場合，最初に指定されたディレクトリにリポジトリファイルを新規に生成します。
- リポジトリファイル検索時に同じ名称のリポジトリファイルが見つからなければ，カレントディレクトリが検索されます。また，環境変数 CBLREP の指定がない場合，リポジトリファイルの生成，更新，および検索は，カレントディレクトリが対象となります。

（例）

ディレクトリとして/usr/user/replib を指定します。

```
CBLREP=/usr/user/replib
export CBLREP
```

(14) CBLSRCENCODING（Linux で有効）

Unicode 機能で入力する COBOL ソースの文字コードがシフト JIS であるときに文字コードを設定します。指定できる値は，SJIS だけです。

（例）

```
CBLSRCENCODING=SJIS
export CBLSRCENCODING
```

なお，この環境変数は次の場合は無効となります。

- この環境変数に SJIS を指定していない場合
- -UniObjGen オプションを指定していない場合
- UTF-8 環境下ではない場合

(15) CBLSYSREP

リポジトリ段落で指定された名前の翻訳単位（関数定義，クラス定義，またはインタフェース定義）を含むリポジトリファイルを検索するディレクトリを指定します。

環境変数 CBLSYSREP に指定したディレクトリに格納されたリポジトリファイルは，リポジトリファイルに情報が格納されている翻訳単位と同じ名称の翻訳単位を作成してコンパイルした場合でも，上書きされることはありません。このため，環境変数 CBLSYSREP には，主に（共用ファイルとリポジトリファイルだけが提供されている場合など）生成元ソースファイルのないリポジトリファイルの検索ディレクトリを指定する場合に使用します。

詳細は，「33. 定義別のコンパイル方法とリポジトリファイル」を参照してください。

規則

- 指定するディレクトリが複数ある場合は，各ディレクトリをコロン（:）で区切って指定します。
- リポジトリファイルの検索順序は，カレントディレクトリが優先されます。

（例）

ディレクトリとして/usr/user/sysreplib と/users/lib/rep を指定します。

```
CBLSYSREP=/usr/user/sysreplib:/users/lib/rep
export CBLSYSREP
```

(16) CBLTAB

COBOL ソース中のタブ位置を設定します。指定できる範囲は 1～72 で，省略時は 8 を仮定します。

（例）

```
CBLTAB=4
export CBLTAB
```

(17) CBLUNINITDATA_OUTRESULTLIST

初期化漏れチェック機能使用時，初期化漏れ確認結果一覧および初期化漏れ従属項目一覧を出力するかどうかを指定します。

この環境変数に ON を指定すると，初期化漏れ確認結果一覧および初期化漏れ従属項目一覧が出力されます。初期化漏れ確認結果一覧および初期化漏れ従属項目一覧の詳細は，「32.7.4 初期化漏れチェック機能」を参照してください。

次の場合は，この環境変数の指定は無効となり，初期化漏れ確認結果一覧および初期化漏れ従属項目一覧が出力されません。

- この環境変数の指定がない，または環境変数の値に ON 以外を指定している
- -CheckUninitData オプションを指定していない

（例）

```
CBLUNINITDATA_OUTRESULTLIST=ON
export CBLUNINITDATA_OUTRESULTLIST
```

(18) CBLUNINITDATA_OUTRESULTLISTDIR

初期化漏れチェック機能使用時，初期化漏れ確認結果一覧および初期化漏れ従属項目一覧を任意のディレクトリに出力したい場合に，その出力先ディレクトリを 255 バイト以内の絶対パス名で指定します。この環境変数の指定がない場合，初期化漏れ確認結果一覧および初期化漏れ従属項目一覧の出力先ディレクトリには，カレントディレクトリが仮定されます。

注意事項

- 出力先ディレクトリは絶対パスで指定してください。絶対パスで指定していない場合の動作は保証しません。
- この環境変数に指定する値の長さは 255 バイト以内でなければなりません。255 バイトを超える値を指定した場合，255 バイトまでの値を有効とします。
- 次の場合は，この環境変数の指定は無効となります。
  - ・環境変数 CBLUNINITDATA_OUTRESULTLIST に ON を指定していない
  - ・-CheckUninitData オプションを指定していない

（例）

```
CBLUNINITDATA_OUTRESULTLISTDIR=/usr/user/Work
export CBLUNINITDATA_OUTRESULTLISTDIR
```

(19) CBLVALUE

サブスキーマ節，作業場所節，画面節（WINDOW SECTION／SCREEN SECTION）※，報告書節，ファイル節で定義しているデータ項目の VALUE 句の指定のない初期値を，このシステムで採用している計算機文字集合である JIS8 単位コードの順序位置（1～256）で指定します。JIS8 単位コードについては，マニュアル「COBOL2002　言語　標準仕様編」「付録 B　計算機文字集合」を参照してください。

なお，ASCII 範囲外の文字コードは，動作するロケールによって扱いが異なります。

注※　AIX で有効です。

実行単位でプログラムが最初に呼び出された場合だけ，この環境変数に指定した初期値をデータ項目に設定します。

この環境変数は，-CBLVALUE オプションを指定した場合だけ有効です。また，-CBLVALUE オプションを指定している場合で，この環境変数の指定がないとき，および指定した値に誤りがあるときは，1 が仮定されます。

（例）

```
CBLVALUE=33
export CBLVALUE
```

VALUE 句のないデータ項目の初期値に，空白文字（JIS8 単位コードで 33 番目の文字）を指定します。

なお，この環境変数は次のデータ項目には適用されません。

- VALUE 句の指定のあるデータ項目
- ADDRESSED 句の指定のあるデータ項目
- EXTERNAL 句の指定のあるデータ項目
- DYNAMIC LENGTH 句のあるデータ項目

注意事項

初期化属性プログラムの作業場所節に対する初期化は，環境変数 CBLVALUE と環境変数 CBLINITVALUE が両方とも有効な場合と，どちらか片方だけが有効な場合とで初期値が異なります。詳細については，「18.4.1　プログラム属性」の「(2)　初期化属性プログラム」を参照してください。また，環境変数 CBLINITVALUE については，「(10)　CBLINITVALUE」を参照してください。

(20)　CBLV3UNICODE

-UniObjGen オプション指定時，-CompatiV3 オプションおよび-V3Rec オプションの指定を有効としたい場合に設定します。

ただし，この環境変数を設定した場合，-CompatiV3 オプションは-JPN,Alnum オプションを仮定しません。

（例）

```
CBLV3UNICODE=YES
export CBLV3UNICODE
```

なお，この環境変数は次の場合は無効となります。

- この環境変数に YES を指定していない場合
- -UniObjGen オプションを指定していない場合

この環境変数とコンパイラオプションの組み合わせによるメインフレーム（VOS3）COBOL85 移行用オプションである，-CompatiV3 オプション，-JPN オプション，および-V3Rec オプションの有効／無効の関係を次に示します。

環境変数 CBLV3UNICODE	-UniObjGen オプションと同時に指定するオプション		
	-CompatiV3	-JPN	-V3Rec
YES	○※	×	○
YES 以外	×	×	×

(凡例)

○：有効となる

×：無効となる

注※

-V3Rec,Variable オプションは仮定されますが，-JPN,Alnum オプションは仮定されません。

注意事項

- -JPN,Alnum オプションが仮定されないことで，日本語項目，日本語編集項目，および日本語文字定数をそれぞれ英数字項目，英数字編集項目，および英数字定数として扱うことができないため，日本語項目と英数字項目間での比較および転記ができません。また，VOS3 COBOL85 と日本語機能に差異があるため，日本語項目または英数字項目のどちらか一方に合わせる必要があります。VOS3 COBOL85 と COBOL2002 との日本語機能差異を次に示します。

項目	VOS3 COBOL85	COBOL2002※
日本語項目と英数字項目間の転記／比較	LANGOPT=(-D)の場合，転記または比較できる。	転記または比較できない。
INITIALIZE 文（日本語項目の初期設定）	• REPLACING 指定がないと半角空白で初期化する。 • REPLACING ALPHANUMERIC, ALPHANUMERIC-EDITED 指定時，一意名 2，定数 1 の値で初期化する。	• REPLACING 指定がないと全角空白で初期化する。 • NATIONAL,NATIONAL-EDITED 以外の REPLACING 指定の場合，初期化されない。
INSPECT，STRING，UNSTRING 文の日本語項目	XCOBOL=(N)オプションのときは 1 バイト単位で処理する。	日本語文字単位で処理する。
日本語項目パディング	LANGOPT=(-D)の場合，パディング文字に半角空白(X'40')を使用する。	パディング文字に全角空白(X'3000')を使用する。

注※

環境変数 CBLV3UNICODE=YES，-UniObjGen オプションおよび-CompatiV3 オプション指定あり

- -V3Rec,Variable オプションが指定できることで，日本語文字定数に標準コード文字が指定できます。日本語文字定数が UTF-16 となり，標準コード文字も 2 バイトとなります。このことから，転記先のけた数を増やす必要があり，その日本語項目または日本語編集項目を参照している個所も修正が必要となる場合があります。例を次に示します。

  (転記先のけた数の見直しの例)

  01 ABC PIC N(2).

  01 DEF PIC N(1).

：

MOVE N'あ AB' TO ABC.

MOVE ABC(2:1) TO DEF.

日本語文字定数は 3 けた（6 バイト）になることから，転記先の ABC けた数を 3 けたに修正する必要があります。

また，標準コード文字が 2 バイトになることから，DEF に'AB'が格納されていることを期待する場合，標準コード文字'AB'を参照する部分参照の長さを 1 から 2 に修正する必要があり，その転記先となる DEF のけた数も 1 から 2 に修正する必要があります。

（日本語文字定数の半角空白補完による転記先のけた数の見直しの例）

01 ABC PIC N(3).

：

MOVE N'あ abc' TO ABC.

日本語文字定数の標準コード文字数が奇数になる場合は，標準コード文字数が偶数になるように日本語文字定数の終端に半角空白が補完されます。これによって，補完された半角空白を含み，日本語文字定数は 5 けた（10 バイト）になることから，転記先となる ABC のけた数を 5 けたに修正する必要があります。

- -CompatiV3 オプションが指定できることで，報告書作成機能が拡張され，特殊名段落の定数 10 が指定可能となりますが，1 バイト文字の指定が前提であるため，UTF-8 で多バイトとなる半角かたかなは指定できません。このことから，定数 10 に UTF-8 で多バイトとなる文字を指定している場合，UTF-8 で 1 バイトとなる文字に変更する必要があります。

(21)　登録集環境変数

登録集原文を検索するディレクトリを設定します。ディレクトリを複数指定する場合は，コロン（:）で区切って指定します。

（例）

```
登録集名=/usr/user/copylib
export 登録集名
```

登録集名は，英大文字と数字から成る 8 文字以内の任意の文字列で指定します。

ここで指定したディレクトリは，環境変数 CBLLIB で指定したディレクトリよりも優先します。詳細については，「32.3.1　原始文操作機能」を参照してください。

32.7.1　TD コマンド生成機能

TD コマンド生成機能とは，COBOL ソースファイルを解析して，テストデバッガで使用する TD コマンドをファイルに生成する機能です。

生成される TD コマンドには，中断点情報，プログラムのシミュレーション情報，およびファイルのシミュレーション情報の 3 種類があります。

これらの TD コマンド群は，-TestCmd,Break，-TestCmd,Sim，-TestCmd,Full オプションを指定してコンパイルしたときに生成されます。生成された TD コマンド群に，必要に応じて TD コマンドの修正や追加をすることで，TD コマンド格納ファイルを作成できます。

また，TD コマンド生成機能の対象となるプログラムは，コンパイルしたときすべてのプログラムで，S レベル，U レベルのコンパイルエラーのなかった原始プログラムファイルだけです。

TD コマンド生成機能の詳細については，マニュアル「COBOL2002　使用の手引　操作編」を参照してください。

32.7.2　makefile 生成機能

(1)　makefile 生成機能の概要

makefile 生成機能とは，COBOL2002 で作成するプログラムやライブラリの作成方法や保守方法を，make に通知する makefile を生成する機能です。この機能は，cbl2kmf コマンドで使用します。生成した makefile は，テキストエディタで変更できます。

(2)　makefile の生成方法

(a)　cbl2kmf コマンド

形式

```
cbl2kmf　〔-ls〕〔-f makefile名〕〔 マクロ名＝値〕 …
```

機能

- COBOL 用の makefile を生成します。
- makefile 生成前に，既存の makefile の有無を調べます。

規則

-l

makefile をライブラリ用の makefile にします。

-s

cbl2kmf 起動前に makefile が存在するとき，このオプションを指定して cbl2kmf コマンドを実行すると，存在する makefile にあるマクロ定義や COBOL 言語用サフィックスは変更しないで makefile を生成します。

ただし，次に示すものは，このオプションを指定しても，cbl2kmf コマンドを実行したときに対象となる COBOL ソースファイルを基に値を変更します。

・EXTHDRS マクロ

・HDRS マクロ

・OBJS マクロ

・SRCS マクロ

・COBOL 原始プログラムと COBOL 登録集原文の依存関係を示すターゲット行とシェル行

なお，このオプションと同時に，cbl2kmf コマンドで指定できるマクロ定義や環境変数を指定したときは，指定されたマクロ定義や環境変数の値に変更して makefile を生成します。

-f makefile 名

makefile のファイル名を指定します。省略した場合は，Makefile になります。

マクロ定義

次のマクロ定義を使用できます。

・CBLMAIN

メインプログラムのソースファイル名を指定します（COBOL 以外のプログラムがメインプログラムのときは指定しないでください)。

・CBLFLAGS

COBOL2002 のコンパイラオプションを指定します。

複数のフラグを指定するときは，引用符（"）で囲んでください。

・LDFLAGS

リンカに渡すオプションを指定します。複数のオプションを指定するときは，引用符(")で囲んでください。

・PROGRAM

実行可能ファイルの名称を指定します。このマクロの指定がない場合，実行可能ファイルの名称のデフォルトは，a.out です。

・LIBS

リンク時に必要なライブラリを指定します。

・DEST

実行可能ファイルおよびライブラリのインストール先を指定します。

・INSTALL
インストールコマンドを指定します。
・LD
リンクに使用するコマンドを指定します。
・MAKEFILE
makefile の名称を指定します。このマクロ定義と-f オプションを同時に指定しているときは，-f オプションの指定値が有効になります。
また，このマクロ定義と-f オプションの両方の指定がない場合，makefile の名称のデフォルトは，Makefile です。
・PRINT
プリントコマンドを指定します。
・SHELL
シェルを指定します。
・LIBRARY
ライブラリの名称を指定します。このマクロの指定がない場合，ライブラリの名称のデフォルトは，lib.a です。
また，このマクロ定義の指定がある場合は，-l オプションを仮定します。
ただし，このマクロ定義と PROGRAM マクロ定義の両方の指定がある場合は，-l オプションを仮定しません。

環境変数

cbl2kmf に指定できる環境変数を次に示します。

環境変数	意味	デフォルト値
LD	リンクに使用するコマンドを指定	"ccbl2002"
CBLMAIN	メインプログラムのソースファイル名を指定	""
CBLFLAGS	COBOL2002 のコンパイラオプションを指定	""

- LD，CBLMAIN および CBLFLAGS は，マクロ定義，環境変数，-s オプションの順に優先します。
- 登録集原文を検索するディレクトリを設定する環境変数 CBLLIB など，ccbl2002 コマンドでコンパイルするときに必要な環境変数は，cbl2kmf コマンド起動時も必要となります。環境変数の詳細については，「32.6.2　コンパイラ環境変数の一覧」を参照してください。
- cbl2kmf コマンドは，次の終了コードを返します。

終了コード	意味
0	cbl2kmf は，正常に終了した。
1	cbl2kmf で，エラーが発生した。
2	cbl2kmf で，回復不能エラーが発生した。

注意事項

- cbl2kmf コマンドでマクロ定義，環境変数およびオプションの指定がない場合，作成された makefile 中のマクロの値は，システム既定のデフォルト値を使用します。

  cbl2kmf コマンドで-s オプションの指定がある場合は，cbl2kmf 起動前に存在する makefile 中にあるマクロの値を使用します。

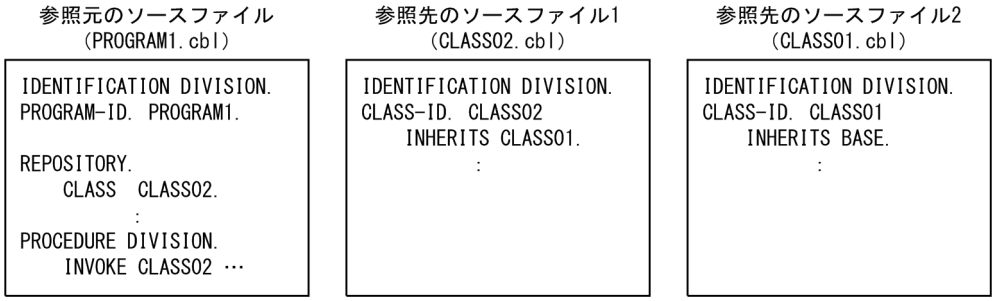
- cbl2kmf コマンドに LIBRARY と PROGRAM の両方のマクロ定義を指定して，ライブラリ用の makefile を生成する場合は，-l オプションの指定が必要です。
- 生成した makefile 中の COBOL マクロの名称は変更してはなりません。
- 生成されたマクロ定義によるマクロ情報および COBOL 言語用サフィックス情報は，###で始まる makefile の行のあとに現れます。なお，###の行は変更してはなりません。
- 生成された依存関係の情報は，####で始まる makefile の行の後に現れます。これらの行を変更してはなりません。
- ###の行以降に現れるマクロ定義によるマクロ情報の値および COBOL 言語用サフィックス情報のシェル行は変更できます。
- ソースファイル中にクラス定義，インタフェース定義，関数定義があった場合でも，次のディレクトリに対する環境変数は生成されません。

  ・リポジトリファイルの検索対象ディレクトリ

  ・リポジトリファイルの出力先のディレクトリ

- ソースファイル中にクラス定義，インタフェース定義，関数定義があった場合でも，リポジトリファイルに対する COBOL 原始プログラムや登録集原文との依存関係を示す情報は生成されません。
- cbl2kmf コマンドで生成された makefile を使用すると，ソースファイル名のアルファベット順にソースファイルがコンパイルされます。参照関係を持つクラス定義，インタフェース定義，関数定義がある場合は，参照先のソースファイルを先にコンパイルするために，参照先のソースファイル名は参照元のソースファイル名よりアルファベット順が先になるようにする必要があります。参照元および参照先のソースファイル（翻訳単位）の関係については，「33.1　リポジトリファイルを使用する COBOL プログラム開発の概要」を参照してください。

  参照元および参照先のソースファイルと，生成された makefile でのコンパイル順序の例を次に示します。



makefile の生成

```
cbl2kmf CBLMAIN=PROGRAM1.cbl
```

生成された makefile でのコンパイル順序

```
ccbl2002 -Compile,NoLink CLASS01.cbl
ccbl2002 -Compile,NoLink CLASS02.cbl
ccbl2002 -Compile,NoLink -Main,System PROGRAM1.cbl
```

- 生成した makefile でコンパイルを実行するときに指定しなければならないコンパイラオプションがある場合は，cbl2kmf コマンドを実行するときに必要なすべてのコンパイラオプションを次のどれかに指定してください。

  ・マクロ定義 CBLFLAGS

  ・環境変数 CBLFLAGS

  ・コンパイラ環境変数 CBLCOPT2002

  また，必要なコンパイラオプションをコンパイラ環境変数 CBLCOPT2002 に指定して makefile を生成した場合は，make を実行するときにも同じコンパイラオプションを指定する必要があります。

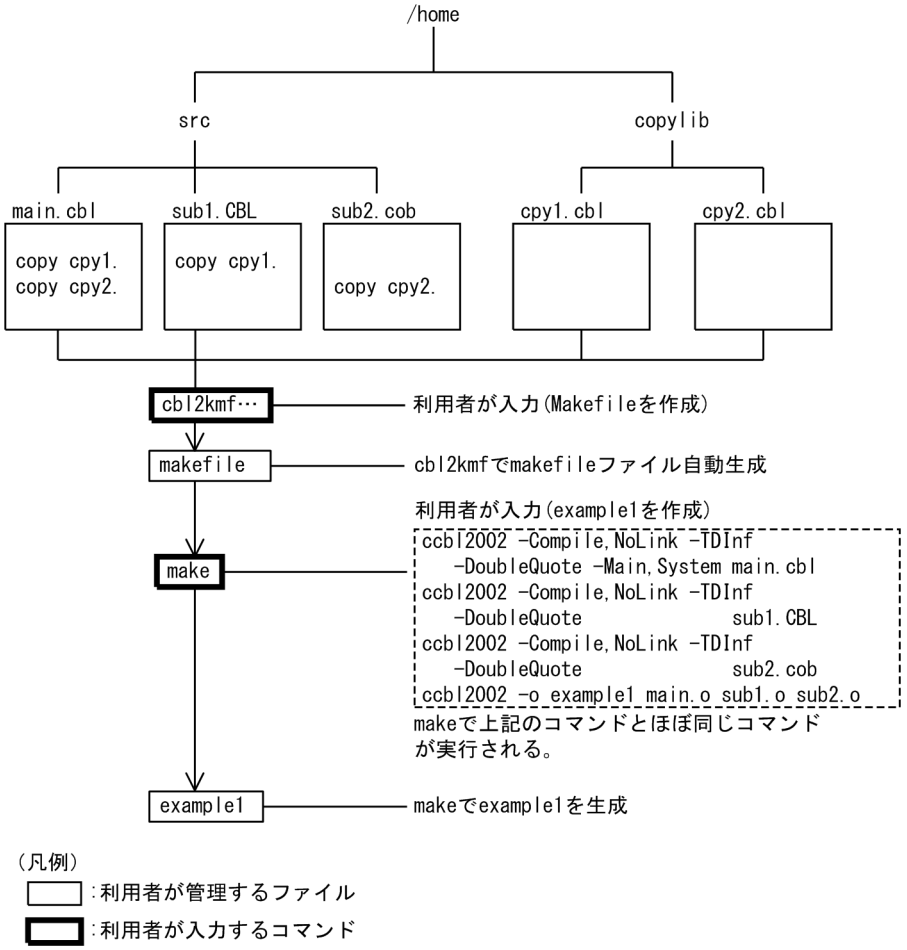
(b) 使用例

makefile 生成機能の使用例を示します。

前提条件

作成する実行可能ファイル名称	example1
原始プログラム名称	main.cbl （主プログラム，原始プログラムに登録集 cpy1.cbl，cpy2.cbl を複写）
	sub1.CBL （副プログラム，原始プログラムに登録集 cpy1.cbl を複写）
	sub2.cob （副プログラム，原始プログラムに登録集 cpy2.cbl を複写）
登録集原文名	cpy1.cbl （原始プログラムとは別ディレクトリ）
	cpy2.cbl （原始プログラムとは別ディレクトリ）
コンパイラオプション	-TDInf （テストデバッグ機能が使用できる実行可能ファイルを作成）
	-DoubleQuote （原文中に引用符を使用）

ファイル構成と操作概要



makefile の内容

```
CBLMAIN       = main.cbl
CFLAGS        =
DEST          = .
EXTHDRS       = ../copylib/cpy1.cbl ¥
                    ../copylib/cpy2.cbl
HDRS          =
INSTALL       = /usr/bin/install
LD            = ccbl2002
LDFLAGS       = -TDInf
LIBS          =
MAKEFILE      = Makefile
OBJS          = main.o ¥
                    sub1.o ¥
                    sub2.o
PRINT         = pr
PROGRAM       = example1
SHELL         = /bin/sh
SRCS          = main.cbl ¥
                    sub1.CBL ¥
                    sub2.cob
SYSHDRS       =
all:            $(PROGRAM)
$(PROGRAM):     $(OBJS) $(LIBS)
                    @echo "Linking $(PROGRAM) ..."
```

```
                    @$(LD) $(LDFLAGS) $(OBJS) $(LIBS) -o $(PROGRAM)
                    @echo "done"
clean:;         @rm -f $(OBJS) core
clobber:;       @rm -f $(OBJS) $(PROGRAM) core tags
depend:;        @cbl2kmf -s -f $(MAKEFILE) CBLMAIN=$(CBLMAIN)
echo:;          @echo $(HDRS) $(SRCS)
install:        $(PROGRAM)
                    @echo Installing $(PROGRAM) in $(DEST)
                    @-strip $(PROGRAM)  ※1
                    @-strip -X 64 $(PROGRAM)  ※2
                    @if [ $(DEST) != . ]; then ¥
                 (rm -f $(DEST)/$(PROGRAM); $(INSTALL) -f $(DEST) $(PROGRAM)); fi  ※3
                 (rm -f $(DEST)/$(PROGRAM); $(INSTALL) $(PROGRAM) $(DEST)); fi  ※4
print:;         @$(PRINT) $(HDRS) $(SRCS)
update:         $(SRCS) $(LIBS) $(HDRS) $(EXTHDRS)
                @$(MAKE) -f $(MAKEFILE) ROOT=$(ROOT) DEST=$(DEST) install
###
```

注※1

AIX(32)，Linux の場合に生成されます。

注※2

AIX(64)の場合に生成されます。

注※3

AIX の場合に生成されます。

注※4

Linux の場合に生成されます。

上記の makefile の内容の詳細は，システムのマニュアルを参照してください。

環境変数指定またはマクロ定義によるマクロ

```
CBL          = ccbl2002
CBLFLAGS     = -TDInf -DoubleQuote
CBLMAINOPT   = -Main,System
CBLCMPLOPT   = -Compile,NoLink
```

COBOL 言語用サフィックス

.SUFFIXES : .cbl .CBL .cob .ocb .cbf .ocf

.cbl:

$(CBL) $(CBLFLAGS) $(CBLMAINOPT) $< $(LDFLAGS) -o $@

.cbl.o:

$(CBL) $(CBLFLAGS) $(CBLCMPLOPT) $<

.CBL:

$(CBL) $(CBLFLAGS) $(CBLMAINOPT) $< $(LDFLAGS) -o $@

.CBL.o:

$(CBL) $(CBLFLAGS) $(CBLCMPLOPT) $<

```
.cob:
    $(CBL) $(CBLFLAGS) $(CBLMAINOPT) $< $(LDFLAGS) -o $@
.cob.o:
    $(CBL) $(CBLFLAGS) $(CBLCMPLOPT) $<
.ocb:
    $(CBL) $(CBLFLAGS) $(CBLMAINOPT) $< $(LDFLAGS) -o $@
.ocb.o:
    $(CBL) $(CBLFLAGS) $(CBLCMPLOPT) $<
.cbf:
    $(CBL) $(CBLFLAGS) $(CBLMAINOPT) $< $(LDFLAGS) -o $@
.cbf.o:
    $(CBL) $(CBLFLAGS) $(CBLCMPLOPT) $<
.ocf:
    $(CBL) $(CBLFLAGS) $(CBLMAINOPT) $< $(LDFLAGS) -o $@
.ocf.o:
    $(CBL) $(CBLFLAGS) $(CBLCMPLOPT) $<
```

COBOL 原始プログラムと COBOL 登録集原文の依存関係を示すターゲット行とシェル行

```
.main.o: main.cbl ../copylib/cpy1.cbl ../copylib/cpy2.cbl
$(CBL) $(CBLFLAGS) $(CBLCMPLOPT) $(CBLMAINOPT) $<
sub1.o: sub1.CBL ../copylib/cpy1.cbl
sub2.o: sub2.cob ../copylib/cpy2.cbl
```

具体的な操作

コマンド

```
cd     /home/src      *>1.
CBLLIB=/home/copylib  *>2.
export CBLLIB         *>3.
cbl2kmf  PROGRAM=example1 CBLMAIN=main.cbl
         CBLFLAGS="-TDInf -DoubleQuote"
         LDFLAGS=-TDInf  *>4.
make                     *>5.
```

説明

1. 原始プログラムが，存在するディレクトリを変更します。
2. 登録集原文は，別ディレクトリであるため，cbl2kmf が検索できるように環境変数 CBLLIB を設定します。
3. 環境変数 CBLLIB を有効にします。
4. makefile を生成します。
5. 実行可能ファイル example1 を生成します。

(3) cbl2kmf コマンドに関する注意事項

cbl2kmf コマンドは，COBOL ソースファイルだけを対象として makefile を生成します。C プログラムに対しては，makefile が生成されないので注意してください。

(4) makefile のマクロ一覧

cbl2kmf コマンドが生成する makefile で定義するマクロ名の用途と値の設定可否を次に示します。

表 32-7 makefile のマクロ一覧

項番	マクロ名※	用途	値の設定可否	
			マクロ定義	環境変数
1	CBLMAIN	メインプログラムのソースファイル名	○	○
2	CFLAGS	C コンパイラのオプション	×	×
3	DEST	実行可能ファイルおよびライブラリのインストール先	○	×
4	EXTHDRS	登録集原文	×	×
5	HDRS	カレントディレクトリの登録集原文	×	×
6	INSTALL	インストールコマンド	○	×
7	LD	リンクに使用するコマンド	○	○
8	LDFLAGS	リンカに渡すオプション	○	×
9	LIBS	リンクするライブラリ	○	×
10	MAKEFILE	Makefile の名称 -f オプションでも指定可能	○	×
11	OBJS	オブジェクトファイル	×	×
12	PRINT	プリントコマンド	○	×
13	PROGRAM	実行可能ファイルの名称	○	×
14	SHELL	シェル	○	×
15	SRCS	ソースファイル	×	×
16	SYSHDRS	C 言語ソースファイルのシステムヘッダ	×	×
17	LIBRARY	ライブラリの名称	○	×
18	CBL	COBOL コンパイルコマンド	×	×
19	CBLFLAGS	COBOL2002 のコンパイラオプション	○	○

項番	マクロ名※	用途	値の設定可否	
			マクロ定義	環境変数
20	CBLMAINOPT	メインプログラムを指定するコンパイラオプション	×	×
21	CBLCMPLOPT	翻訳のみを指定するコンパイラオプション	×	×
22	SUFFIX	ファイル名の拡張子	×	×

（凡例）

○：cbl2kmf コマンド実行時に設定できる。

×：cbl2kmf コマンド実行時に設定できない。

注※

次のマクロは，cbl2kmf コマンドが COBOL ソースを解析してマクロの値を設定します。

・EXTHDRS
・HDRS
・OBJS
・SRCS
・SUFFIX

32.7.3　ヘルプ機能

ここではヘルプ機能（オプション表示）について説明します。

(1)　ヘルプ機能（オプション表示）

ヘルプ機能（オプション表示）とは，COBOL2002 のコンパイラを起動するとき，ccbl2002 コマンド行，および ccbl コマンド行に指定できるオプションを画面に表示する機能です。

オプションを何も指定しないで ccbl2002 コマンド，および ccbl コマンドを起動すると，すべてのコンパイラオプションが表示されます。

ccbl2002 コマンド，および ccbl コマンドの詳細については，「32.4.1　ccbl2002 コマンド」，および「32.4.2　ccbl コマンド（AIX で有効）」を参照してください。

32.7.4　初期化漏れチェック機能

ここでは，初期化漏れチェック機能の使用方法について説明します。

(1)　初期化漏れチェック機能の概要

初期化漏れチェック機能とは，データ項目を参照しているが，値の設定（初期化）がされていない，またはその可能性のあるデータ項目を「初期化漏れ」として検出する機能です。

実行時に検出される初期化漏れが原因の不良は，不良個所の特定のために複数のプログラムにわたった調査が必要になります。このため，不良個所の特定に時間が掛かることがあります。初期化漏れチェック機能を使用すると，コンパイル時に初期化漏れを検出でき，テスト工程前に対策できるため，COBOL プログラムの開発効率を向上できます。

初期化漏れチェック機能で「初期化漏れ」を検出する COBOL ソースの例を次の図に示します。

図 32-4　初期化漏れチェック機能で「初期化漏れ」を検出する COBOL ソースの例

```
                    :
000300 DATA DIVISION.
000400 WORKING-STORAGE SECTION.
000500   01 DAT-1   PIC X(2) VALUE SPACE.
000600   01 DAT-2   PIC X(2).              ← DAT-2は1度も値設定がない
000700   01 DAT-3   PIC X(3).
000800 LINKAGE SECTION.
000900   01 RESULT  PIC 9(3).
001000 PROCEDURE DIVISION USING RESULT.
001100   IF RESULT > 60 THEN
001200     MOVE "OK" TO DAT-3             ← DAT-3は値設定(初期化)されないパスがある
001300   END-IF.
001400
001500   DISPLAY DAT-1 DAT-2 DAT-3.       ← DAT-2，DAT-3を「初期化漏れ」として検出する
                    :
```

(2)　初期化漏れチェック機能を使用するときに指定するコンパイラオプション

初期化漏れチェック機能を使用する場合は，コンパイル時に次のオプションを同時に指定します。

- -Compile,CheckOnly オプション※

  -Compile,CheckOnly オプションについては，「32.5.10　リンクの設定」の「(1)　-Compile オプション」を参照してください。

- -CheckUninitData オプション

  -CheckUninitData オプションについては，「32.5.14　その他の設定」の「(24)　-CheckUninitData オプション」を参照してください。

注※

次の場合，-Compile,CheckOnly オプションが無効となるため，初期化漏れチェック機能も無効になります。

- 明示的に-Compile,NoLink オプションを指定した場合

注意事項

- 初期化漏れチェック機能は，-CBLVALUE オプションおよび環境変数 CBLVALUE，ならびに環境変数 CBLINITVALUE 指定時に設定される初期値を無視します。

- 初期化漏れチェック機能は，プログラムを実行したときに通る可能性のある経路（制御ブロック）をすべて走査し，手続き文で参照されているデータ項目の初期化漏れをチェックします。ただし，環境変数 CBL_UNINITDATA_BREAKOFF の指定が無効な場合に，走査した制御ブロックの数が上限を超えたときは，初期化漏れチェック機能の警告メッセージを出力して，初期化漏れチェック処理を打ち切ります。環境変数 CBL_UNINITDATA_BREAKOFF については，「32.6.3　コンパイラ環境変数の詳細」の「(2)　CBL_UNINITDATA_BREAKOFF」を参照してください。

(3)　初期化漏れチェック機能で出力するファイル

初期化漏れチェック機能では，この機能でチェックした初期化漏れの可能性があるデータ項目を，次に示す一覧でファイルに出力することもできます。

- 初期化漏れ確認結果一覧
- 初期化漏れ従属項目一覧

これらの一覧を出力する場合は，環境変数 CBLUNINITDATA_OUTRESULTLIST に ON を指定します。環境変数 CBLUNINITDATA_OUTRESULTLIST の詳細は，「32.6.3　コンパイラ環境変数の詳細」の「(17)　CBLUNINITDATA_OUTRESULTLIST」を参照してください。

また，これらの一覧の出力先を変更する場合は，環境変数 CBLUNINITDATA_OUTRESULTLISTDIR に出力先ディレクトリを指定します。環境変数 CBLUNINITDATA_OUTRESULTLISTDIR の詳細は，「32.6.3　コンパイラ環境変数の詳細」の「(18)　CBLUNINITDATA_OUTRESULTLISTDIR」を参照してください。

初期化漏れチェック機能で出力するファイルの共通規則を次に示します。

共通規則

1. ファイルは CSV 形式です。
2. ファイルの 1 行目は列名を表します。2 行目以降は列に対する内容の値を表します。
3. 各項目は引用符（"）で囲みます。ただし，項目の値がない場合は囲みません。
   "列名 1","列名 2","列名 3",…
   "値 1",,"値 3",…
4. ファイルの文字コードは COBOL ソースファイルの文字コードと同じです。
5. ファイルは，コンパイルの対象となる COBOL ソースファイルの単位に出力します。
6. ファイルは，コンパイルごとに上書きして出力します。
7. ファイル入出力でエラーが発生した場合，メッセージを出力し，処理を中断します。

(a)　初期化漏れ確認結果一覧

初期化漏れ確認結果一覧は，初期化漏れチェック機能でチェックした，初期化漏れの可能性があるデータ項目の情報を出力する一覧です。初期化漏れ確認結果一覧は，初期化漏れのメッセージごとに，次に示す形式で情報を出力します。

初期化漏れ確認結果一覧の形式

列名	内容
#	初期化漏れ確認結果一覧内の一意な通番です。
ディレクトリ名	COBOL ソースファイルがあるディレクトリ名です。
ファイル名	COBOL ソースファイル名です。
プログラム名	プログラム名です。
COPY 定義ディレクトリ名	メッセージが出力された個所が登録集原文内である場合，登録集原文があるディレクトリ名です。登録集原文内でない場合は何も出力されません。
COPY ファイル名	メッセージが出力された個所が登録集原文内である場合，登録集原文のファイル名。登録集原文内でない場合は何も出力されません。
一連番号	メッセージが出力された，コンパイルリストの原始プログラムリストに表示される行番号です。
ファイル内行番号	メッセージが出力された，COBOL ソースファイルまたは登録集原文ファイル内の相対行番号です。 ただし，埋め込み SQL 文の場合，メッセージは語「EXEC」の位置に出力されますが，ここにはデータ項目の位置が出力されます。
ファイル内カラム	メッセージが出力された，COBOL ソースファイルまたは登録集原文ファイル内のカラム番号です。 ただし，埋め込み SQL 文の場合，メッセージは語「EXEC」の位置に出力されますが，ここにはデータ項目の位置が出力されます。
データ項目名	データ項目の修飾付きデータ名です。 修飾付きデータ名の形式を次に示します。 `データ名 〔OF 集団項目名〕…〔IN ファイル名〕`
従属項目一覧有無	初期化漏れ従属項目一覧への情報の出力有無を 0 または 1 で示します。 0：初期化漏れ従属項目一覧へ情報は出力していません。 1：初期化漏れ従属項目一覧に情報を出力しています。

初期化漏れ確認結果一覧の規則

1. 初期化漏れ確認結果一覧のファイル名称は，「<COBOL ソースファイル名（拡張子を除く）>-UninitCheckResult.csv」です。

   （例）COBOL ソースファイル名が「Prog1.cbl」の場合

   Prog1-UninitCheckResult.csv

2. 初期化漏れ確認結果一覧のファイルは，初期化漏れの可能性があるデータ項目を検出しなかった場合でも出力します。

初期化漏れ確認結果一覧の出力例

COBOL ソースファイル

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. MAIN.
```

```
000300 DATA DIVISION.
000400 WORKING-STORAGE SECTION.
000500 01 A PIC X.
000600 01 B PIC X VALUE 'B'.
000700 01 C.
000800   03 C1 PIC X.
000900   03 C2 PIC X.
001000 PROCEDURE DIVISION.
001100     DISPLAY A.
001200     IF B = 'B' THEN
001300         MOVE SPACE TO C
001400         PERFORM SEC1
001500     ELSE
001600         PERFORM SEC1
001700     END-IF.
001800     STOP RUN.
001900
002000 SEC1.
002100     COPY SAMPLECOPY.
002200
002300 END PROGRAM MAIN.
```

登録集原文

```
           DISPLAY C1.
```

コンパイルリストの原始プログラムリスト

```
            A 000100 IDENTIFICATION DIVISION.
              000200 PROGRAM-ID. MAIN.                                  2300
              000300 DATA DIVISION.
              000400 WORKING-STORAGE SECTION.
              000500 01 A PIC X.                                       1100
              000600 01 B PIC X VALUE 'B'.                            Q1200
              000700 01 C.                                            *1300
              000800   03 C1 PIC X.                                   2101
              000900   03 C2 PIC X.
              001000 PROCEDURE DIVISION.
              001100     DISPLAY A.                                    500
                               ?
KCCC6951C-W データ名"A"は，初期化されていない可能性があります。
              001200     IF B = 'B' THEN                               600
            1 001300         MOVE SPACE TO C                           700
            1 001400         PERFORM SEC1                              2000
              001500     ELSE
            1 001600         PERFORM SEC1                              2000
              001700     END-IF.
              001800     STOP RUN.
              001900
              002000 SEC1.                                             P1400,P1600
              002100     COPY SAMPLECOPY.
   2101   C1               DISPLAY C1.                                  800
                               ?
KCCC6951C-W データ名"C1"は，初期化されていない可能性があります。
              002200
            A 002300 END PROGRAM MAIN.                                 200
```

初期化漏れ確認結果一覧

```
"#","ディレクトリ名","ファイル名","プログラム名","COPY定義ディレクトリ名","COPYファイル名","一連番号","ファイル内行番号","ファイル内カラム","データ項目名","従属項目一覧有無"
"1","/usr/user/WORK","Prog1.CBL","MAIN",,,"001100","11","20","A","0"
"2","/usr/user/WORK","Prog1.CBL","MAIN","/usr/user/WORK/COPY","SAMPLECOPY.CBL","2101","1","20","C1 OF C","0"
```

(b) 初期化漏れ従属項目一覧

初期化漏れ従属項目一覧は，初期化漏れチェック機能でチェックした，初期化漏れの可能性があるデータ項目が集団項目である場合に，その従属項目の情報を出力する一覧です。初期化漏れ従属項目一覧は，次に示す形式で情報を出力します。

初期化漏れ従属項目一覧の形式

列名	内容
#	初期化漏れ従属項目一覧内の一意な通番です。
確認結果一覧の#	初期化漏れ確認結果一覧での#（通番）です。
従属項目名	初期化漏れの可能性がある従属項目の修飾付きデータ名です。 修飾付きデータ名の形式を次に示します。 データ名 〔OF 集団項目名〕…〔IN ファイル名〕 初期化漏れの可能性がある従属項目が FILLER 項目の場合は，データ名を FILLER とみなした修飾付きデータ名です。なお，対象は，従属項目のうち基本項目だけとなります。

初期化漏れ従属項目一覧の規則

1. 初期化漏れ従属項目一覧のファイル名称は，「<COBOL ソースファイル名（拡張子を除く）>-UninitCheckSubordinate.csv」です。

   （例）COBOL ソースファイル名が Prog1.cbl の場合

   Prog1-UninitCheckSubordinate.csv

2. 初期化漏れ従属項目一覧のファイルは，初期化漏れ確認結果一覧での初期化漏れの可能性があるデータ項目に集団項目がない場合でも出力します。

初期化漏れ従属項目一覧の出力例

COBOL ソースファイル

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. MAIN.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 A.
        03 A1.
          05 A11 PIC X.
        03 A2.
          05 A21 PIC X.
       01 B.
        03 B1.
          05 B11 PIC X.
        03 B2.
          05 B21 PIC X.
       PROCEDURE DIVISION.
           DISPLAY A.

           MOVE SPACE TO B1.
           DISPLAY B.
```

```
        END PROGRAM MAIN.
```

コンパイルリストの原始プログラムリスト

```
     1      A        IDENTIFICATION DIVISION.
     2               PROGRAM-ID. MAIN.                                        21
     3               DATA DIVISION.
     4               WORKING-STORAGE SECTION.
     5               01 A.                                                    16
     6                03 A1.
     7                  05 A11 PIC X.
     8                03 A2.
     9                  05 A21 PIC X.
    10               01 B.                                                    19
    11                03 B1.                                                  *18
    12                  05 B11 PIC X.
    13                03 B2.
    14                  05 B21 PIC X.
    15               PROCEDURE DIVISION.
    16                   DISPLAY A.                                           5
                              ?
KCCC6951C-W データ名"A"は，初期化されていない可能性があります。
    17
    18                   MOVE SPACE TO B1.                                    11
    19                   DISPLAY B.                                           10
                              ?
KCCC6951C-W データ名"B"は，初期化されていない可能性があります。
    20
    21      A        END PROGRAM MAIN.                                        2
```

初期化漏れ確認結果一覧

```
"#","ディレクトリ名","ファイル名","プログラム名","COPY定義ディレクトリ名","COPYファイル名","一連番号","ファイル内行番号","ファイル内カラム","データ項目名","従属項目一覧有無"
"1","/usr/user/WORK","Prog1.CBL","MAIN",,,"16","16","20","A","1"
"2","/usr/user/WORK","Prog1.CBL","MAIN",,,"19","19","20","B","1"
```

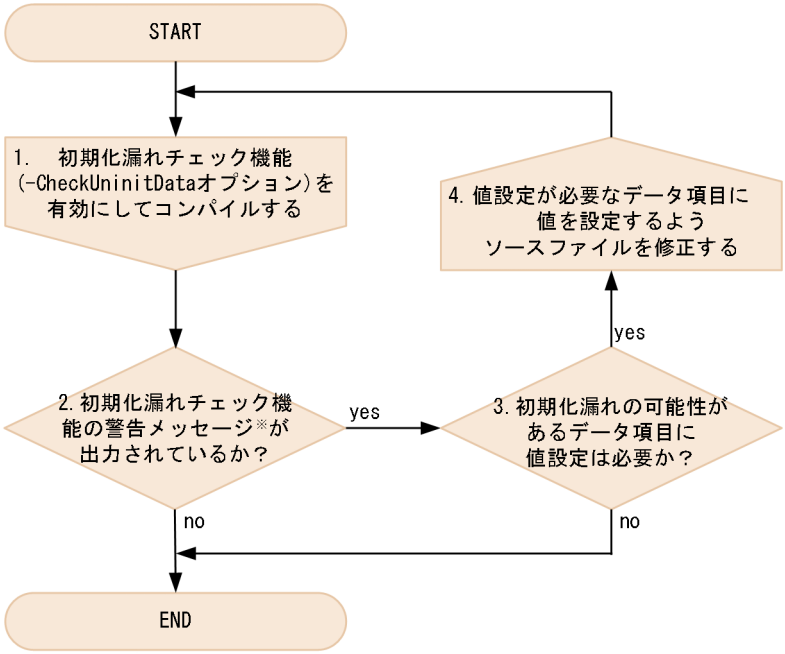
初期化漏れ従属項目一覧

```
"#","確認結果一覧の#","従属項目名"
"1","1","A11 OF A1 OF A"
"2","1","A21 OF A2 OF A"
"3","2","B21 OF B2 OF B"
```

(4) 初期化漏れチェック機能の使用方法

初期化漏れチェック機能を使用するときの作業の流れを次の図に示します。初期化漏れチェック機能を使用する前に，あらかじめ S レベル，U レベルのコンパイルエラーがないことを確認してください。

図 32-5　初期化漏れチェック機能を使用するときの作業の流れ



注※　KCCC6951C-W～KCCC6955C-Wのメッセージ

初期化漏れチェック機能は，次の方法で使用できます。

- コンパイルリストから調査する

(a)　コンパイルリストから調査する方法

コンパイルリストから初期化漏れを調査する手順を次に示します。

1. -CheckUninitData オプション，-Compile,CheckOnly オプションおよび-SrcList オプションを指定してコンパイルする。

   コンパイルリストに初期化漏れチェックの結果が出力されます。

2. コンパイルリストまたは標準エラー出力に，初期化漏れチェック機能の警告メッセージが出力されていないかを確認する。

3. 初期化漏れチェック機能の警告メッセージが出力されている場合は，メッセージの内容を確認して，初期化漏れの可能性があるデータ項目に，値設定が必要かどうかを確認する。

4. 値設定が必要な場合は，プログラムを修正する。

(5)　初期化漏れチェック機能使用時の注意事項

- 次の表に示す機能に該当するデータ項目は，初期化漏れチェック機能の対象外となり，常に初期化済みと判定されます。

項番	機能
1	オブジェクト指向機能

項番	機能
	• オブジェクトビュー • オブジェクトプロパティ • SELF および SUPER • オブジェクト参照 • INVOKE 文 など
2	利用者定義関数
3	連絡節で定義されているデータ項目
4	EXTERNAL 句または GLOBAL 句が指定されているデータ項目またはその従属項目
5	コンパイル対象とならない行 • コメント行 • デバッグ行（-DebugLine オプション未指定時） • 条件翻訳の無効行
6	制御が渡らない文，手続き
7	覚え書きとみなされた構文
8	S レベル，U レベルエラーが発生する COBOL 原始プログラムファイル
9	宣言部分の文，手続き（USE 文の宣言手続き）
10	XDM によるデータベース操作シミュレーション機能 • 構造型データベース操作シミュレーション機能（XDM/SD） • リレーショナルデータベース操作シミュレーション機能（XDM/RD）
11	ADDRESSED BY 句が指定されているデータ項目，またはその従属項目
12	可変部分に定義されたデータ項目
13	動的長基本項目
14	報告書作成機能
15	被再定義項目が初期化漏れチェック機能のチェック対象外である再定義項目およびその従属項目すべて※1※2
16	被再命名項目が初期化漏れチェック機能のチェック対象外である再命名項目※2
17	再定義項目が初期化漏れチェック機能のチェック対象外である被再定義項目およびその従属項目すべて※2
18	RENAMES 句に THRU 指定がある再命名項目※3
19	合成キー
20	すべての従属項目が初期化漏れチェック機能のチェック対象外であるデータ項目

注※1

チェック対象外であるデータ項目を再定義した例を次に示す。

（例）

```
022000 WORKING-STORAGE SECTION.
023000  01 DAT1 EXTERNAL        PIC X.
024000  01 DAT2 REDEFINES DAT1  PIC X.
          :
039000
043000     DISPLAY DAT2.           ← DAT2はチェック対象外
```

DAT2 の被再定義項目 DAT1 が，初期化漏れチェックの対象外（EXTERNAL 句指定のデータ項目）であるため，DAT2 もチェックされません。

注※2

チェック対象外のデータ項目で再定義した例を次に示します。

（例）

```
022000 WORKING-STORAGE SECTION.
023000  01 DAT1.
023100     03 DAT11             PIC X.
023200     66 DAT12 RENAMES DAT11.
024000  01 DAT2 REDEFINES DAT1  PIC 9.
025000  01 DAT3 REDEFINES DAT1  GLOBAL.
025100     03 DAT31             PIC X.
025200     66 DAT32 RENAMES DAT31.
          :
039000
040000     DISPLAY DAT31 DAT32.    ← DAT31，DAT32はチェック対象外
041000     DISPLAY DAT1 DAT2.      ← DAT1，DAT2はチェック対象外
042000     DISPLAY DAT11 DAT12.    ← DAT11，DAT12はチェック対象外
```

再定義項目 DAT3 が，初期化漏れチェックの対象外（GLOBAL 句指定のデータ項目）であるため，DAT3 と領域を共有する次の項目もチェックされません。

・DAT3 の従属項目 DAT31 と DAT31 の再命名項目 DAT32

・DAT3 の被再定義項目 DAT1，および DAT1 の従属項目 DAT11 と DAT11 の再命名項目 DAT12

・DAT1 の再定義項目 DAT2

注※3

THRU 指定がある再命名項目がチェック対象外となる例を次に示します。

（例）

```
022000 WORKING-STORAGE SECTION.
023000  01 DAT1.
024000    03 DAT2.
025000      05 DAT3  PIC X.
026000      05 DAT4  PIC 9.
027000      05 DAT5  PIC 9.
028000      05 DAT6  PIC X.
029000    66 R-DAT RENAMES DAT3 THRU DAT6.
          :
043000    DISPLAY R-DAT.          ← R-DATはチェック対象外
```

- ファイル節のレコードの初期化漏れチェックは，作業場所節と同じ扱いとなります。

- 手続き文の作用対象に添字付きデータ名が指定された場合，値参照または値設定の対象は，反復データ項目またはその従属項目のすべてとします。

  (例)

```
  01 DAT1 .
    03 DAT2 OCCURS 10.
      05 DAT3 PIC X(1).
      05 DAT4 PIC X(1).
          :
  MOVE SPACE TO DAT3(1).           ← DAT3のすべての要素を初期化済みとみなす
  DISPLAY DAT3(2).                 ← DAT3は初期化済みと判定される
  DISPLAY DAT4(3).                 ← DAT4は初期化漏れの可能性ありと検出される
  MOVE SPACE TO   DAT2(4).         ← DAT2の4番目の要素に半角空白を設定する
  DISPLAY DAT2(1) DAT3(2) DAT4(3). ← DAT2およびその従属項目は初期化済と判定される
```

- 手続き文の作用対象に部分参照付きデータ名が指定された場合，値参照または値設定の対象は，データ項目またはその従属項目すべてとします。

  (例)

```
  01 DAT1 .
    03 DAT2 PIC X(5).
    03 DAT3 PIC X(5).
          :
  MOVE SPACE TO DAT2(5:1). ← DAT2全体が初期化されたものとみなす
  DISPLAY DAT2(1:3).       ← DAT2は初期化済みと判定される
  MOVE SPACE TO DAT1(1:5). ← DAT1全体が初期化されたものとみなす
  DISPLAY DAT1  DAT2 DAT3. ← DAT1およびその従属項目すべて初期化済みと判定される
```

- 被再定義項目と再定義項目，同じ被再定義項目を持つ再定義項目など，再定義によって同じ記憶域に関連づけられた 2 つの集団項目で，一方の集団項目の従属項目に値が設定されたとき，もう一方の集団項目の従属項目は，値が設定された従属項目と同じ領域であったとしても，初期化済みとはみなしません。

  (例)

```
    01 A.
      03  A-1   PIC X.
      03  A-2   PIC X.
    01 B REDEFINES A.
      03  B-1   PIC X.
      03  B-2   PIC X.
          :
    MOVE '1'   TO A-1.        *> 被再定義項目の集団項目Aの従属項目A-1に値を設定
    DISPLAY B-1.              *> 再定義項目の集団項目Bの，A-1と同じ領域の従属項目
                              *> B-1を初期化漏れの可能性ありと判定する
           ?
KCCC6951C-W データ名"B-1"は，初期化されていない可能性があります。
```

- 被再定義項目と再定義項目，同じ被再定義項目を持つ再定義項目など，再定義によって同じ記憶域に関連づけられた 2 つのデータ項目がある場合に，一方が初期化済みであるときは，もう一方のデータ項目およびその従属項目はすべて初期化済みとみなします。このため，初期化済みのデータ項目の長さが，もう一方のデータ項目の長さより短かったとしても，もう一方のデータ項目およびその従属項目を初期化漏れの可能性ありとは判定しません。

  (例)

```
01 A.
  03  A-1   PIC X.
  03  A-2   PIC X.
01 B REDEFINES A PIC X.
       :
MOVE '0'   TO B.          *> 再定義項目のBに値を設定
DISPLAY A-2.              *> BよりもAの方がデータ項目の長さが長いが,
                          *> 被再定義項目の集団項目Aの従属項目A-2は
                          *> 初期化漏れの可能性ありとは判定しない
```

33 定義別のコンパイル方法とリポジトリファイル

COBOL2002 では，プログラム定義のほかに，関数定義，クラス定義，およびインタフェース定義など，さまざまな定義を指定できます。この章では，定義の情報を保管するリポジトリファイルについて説明します。また，定義別のコンパイル方法について説明します。

33.1　リポジトリファイルを使用する COBOL プログラム開発の概要

33.1.1　概要

COBOL2002 では，関数定義，クラス定義，およびインタフェース定義を使用できます。これらの翻訳単位が参照関係を持つ場合，参照先の翻訳単位と参照元の翻訳単位とで，それぞれ次のような処理が必要になります。

- 参照先の翻訳単位

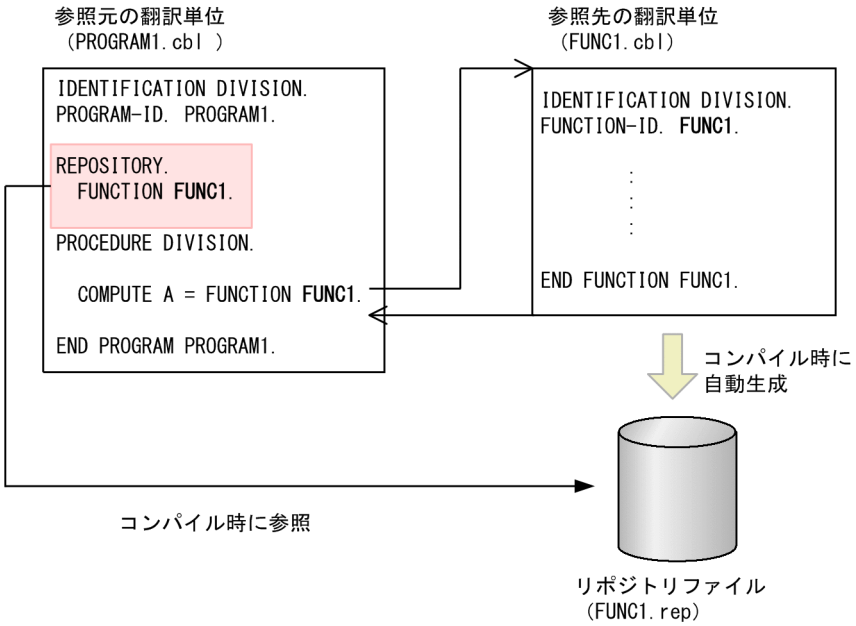
  参照元の翻訳単位から参照されるときに必要な定義情報を，参照元の翻訳単位のコンパイルより前に出力する必要があります。この定義情報を，外部リポジトリと呼びます。外部リポジトリの言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「4.12　外部リポジトリ」を参照してください。

  COBOL2002 では，関数定義，クラス定義，またはインタフェース定義をコンパイルすると，自動的に翻訳単位の外部リポジトリをリポジトリファイル（.rep）に出力します。

- 参照元の翻訳単位

  参照先の翻訳単位の定義情報を取り込む指定が必要です。

  COBOL2002 では，参照元の翻訳単位のリポジトリ段落に参照先の翻訳単位名を指定しておくことによって，参照元の翻訳単位のコンパイル時に参照先の翻訳単位に対応するリポジトリファイルを取り込めます。



規則

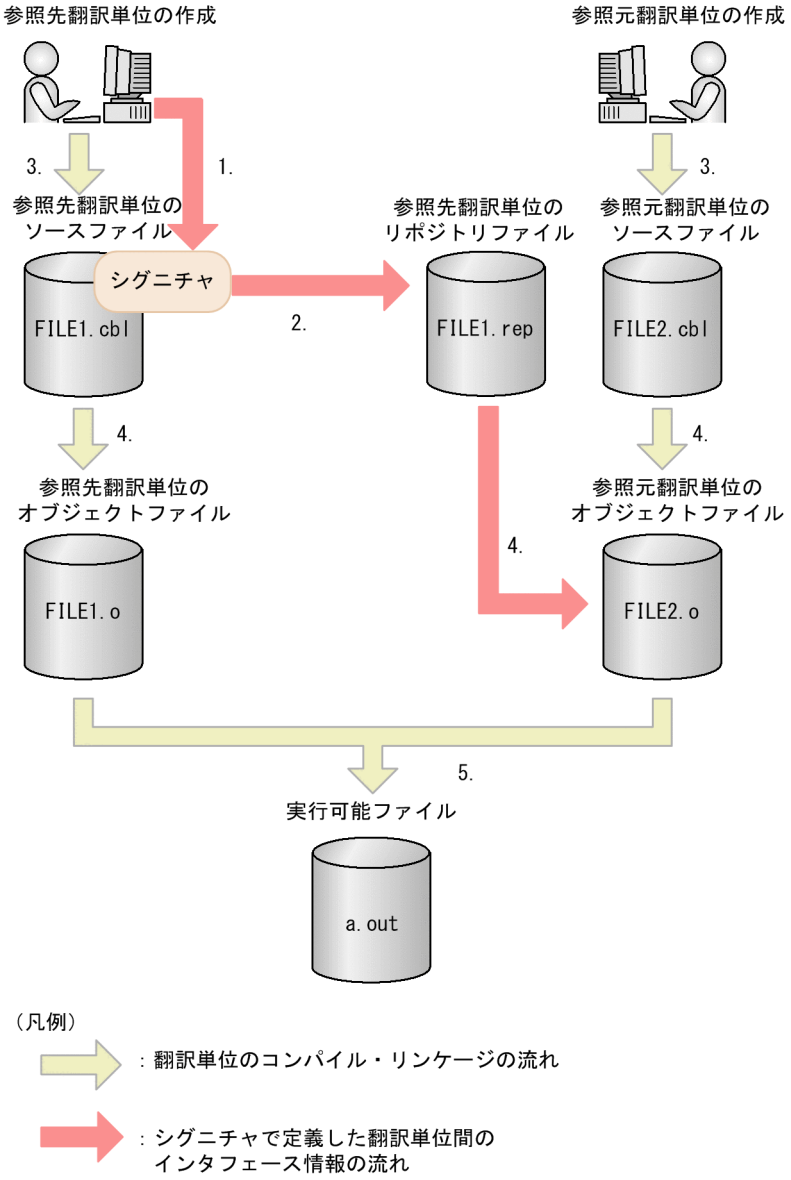
- 参照元の翻訳単位と参照先の翻訳単位が同じソースファイル中だけに存在する場合は，リポジトリファイルがなくても翻訳単位をコンパイルできます。

- プログラム定義だけで構成されているCOBOLプログラムの場合，リポジトリファイルは出力されません。また，リポジトリ段落に参照するプログラム名を指定する必要もありません。詳細は，「33.3.3　プログラム定義だけのコンパイル」を参照してください。
- 参照元の翻訳単位をコンパイルする時点で，参照先の翻訳単位のリポジトリファイルが出力されていない場合，参照関係が解決できないためコンパイルエラーとなります。そのため，異なるソースファイルに存在する翻訳単位同士が参照関係を持つ場合は，ソースファイルをコンパイルする順序を意識する必要があります。

33.1.2　リポジトリファイルを使用するCOBOLプログラムの作成手順

リポジトリファイルを使用するCOBOLプログラム作成の例として，各翻訳単位の定義と，それを参照する翻訳単位を別々のソースファイルで作成する場合の開発手順について説明します。この手順は，例えば複数の開発者が担当部分の原始プログラムを作成およびコンパイルし，最後にそれぞれのオブジェクトファイルをリンクして実行可能ファイルを作成するようなケースに適用できます。

開発手順を，次に示します。



1. 翻訳単位間のインタフェースの決定

   まず，参照先の翻訳単位と参照元の翻訳単位の間のインタフェースを決定する必要があります。COBOL2002 では，この翻訳単位間のインタフェース情報のことをシグニチャと呼びます。シグニチャの詳細については，マニュアル「COBOL2002　言語　標準仕様編」「4.12　外部リポジトリ」を参照してください。

   翻訳単位間のインタフェースを決定するためには，参照先の翻訳単位の原始プログラムで，シグニチャを作成しておく必要があります。

   ただし，オブジェクト指向機能の場合，クラス定義のシグニチャを作成する代わりに，インタフェース定義を使用する方法もあります。インタフェース定義は，実装するクラス定義とインタフェースの構成を変更したい場合などに使用します。インタフェースの詳細は，「20.　オブジェクト指向機能」を参照してください。

2. リポジトリファイルの生成

1.で作成したシグニチャを持つ，参照先の翻訳単位の原始プログラムをコンパイルして，リポジトリファイルを生成します。リポジトリファイルの生成方法については，「33.3.2　リポジトリファイルの単独生成」を参照してください。

3. 原始プログラムの作成

   原始プログラムを作成します。

   参照元の翻訳単位で，環境部構成節のリポジトリ段落に，参照先の翻訳単位名を指定します。また，1.でシグニチャだけを作成した参照先の翻訳単位の原始プログラムも完成させます。

4. リポジトリファイルの取り込みとコンパイル

   リポジトリファイルの格納ディレクトリに，2.で生成した，参照先の翻訳単位から生成したリポジトリファイルを格納します。リポジトリファイルの格納ディレクトリについては，「33.2.4　リポジトリファイルの参照方法」を参照してください。

   翻訳単位をコンパイルすると，コンパイラによってリポジトリファイルが読み込まれ，参照先の翻訳単位のシグニチャと，参照元の翻訳単位の INVOKE 文，SET 文，関数一意名などの参照情報との適合がチェックされます。適合チェックの詳細は，「33.1.3　リポジトリファイルに格納される情報と適合チェック」を参照してください。

5. 実行可能ファイルまたは共用ライブラリの生成

   4.で生成したオブジェクトファイルをリンカによって結合して，実行可能ファイルまたは共用ライブラリを生成します。

33.1.3　リポジトリファイルに格納される情報と適合チェック

リポジトリファイルには，関数定義，クラス定義，およびインタフェース定義のシグニチャ（定義情報）が格納されます。COBOL2002 コンパイラは，コンパイル時にリポジトリファイルを使用して，次の適合チェックをします。

関数定義の場合

参照元の引数と参照先の返却項目の適合がチェックされます。

クラス定義およびインタフェース定義の場合

オブジェクトの適合がチェックされます。このとき，メソッド定義を呼び出すときの引数と返却項目の適合についても，チェックされます。

適合していれば，実行時に関数定義やメソッド定義を呼び出せます。引数と返却項目の適合については，マニュアル「COBOL2002　言語　標準仕様編」「10.7　引数と返却項目の適合」を参照してください。また，オブジェクトの適合については，マニュアル「COBOL2002　言語　標準仕様編」「5.2.7　適合とインタフェース」および「20.　オブジェクト指向機能」を参照してください。

33.2　リポジトリファイル

COBOL2002 の場合，翻訳単位の原始プログラムをコンパイルしたときに出力される外部リポジトリは，リポジトリファイル（.rep）に格納されます。

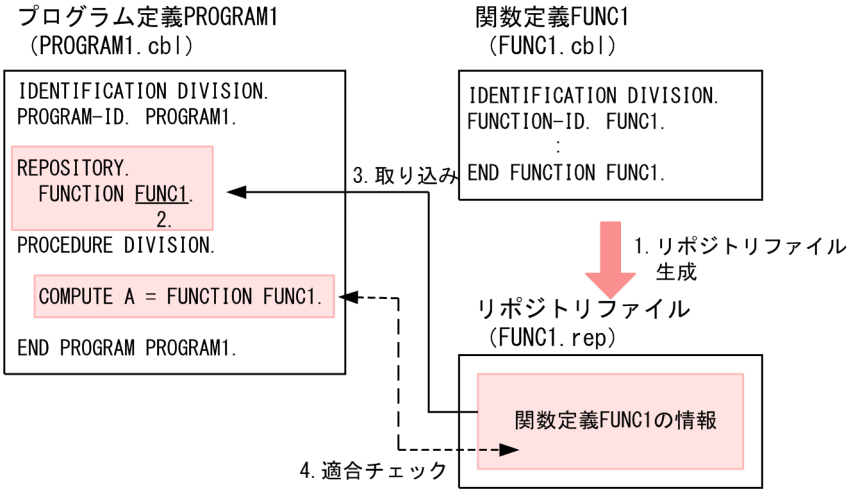
リポジトリファイルは，ソースファイルと 1 対 1 の関係となっています。そのため，ソースファイルをコンパイルした場合，一つのソースファイルに対して一つのリポジトリファイルが生成されます。生成されるリポジトリファイルの名称は，ソースファイルの名称と同じで，拡張子が.rep となります。

33.2.1　リポジトリファイルの生成とコンパイル時の利用

関数定義の場合とオブジェクト指向機能の場合について，それぞれリポジトリファイルの生成とコンパイル時のリポジトリファイルの利用について，例を示します。

(1)　関数定義の場合

関数定義を呼び出す COBOL プログラムの場合の例を，次に示します。なお，関数定義については，「2.5 利用者定義関数」を参照してください。

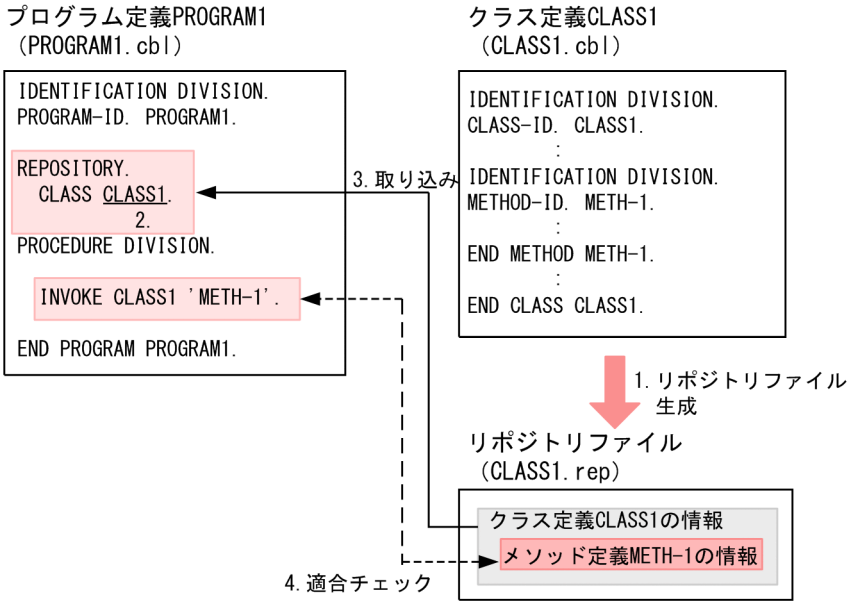


1. 関数定義 FUNC1 が含まれるソースファイル FUNC1.cbl をコンパイルして，リポジトリファイル FUNC1.rep を生成します。
2. 関数定義 FUNC1 を参照するプログラム定義 PROGRAM1 で，環境部構成節のリポジトリ段落に，参照する関数定義 FUNC1 を指定します。
3. プログラム定義 PROGRAM1 をコンパイルすると，リポジトリ段落の指定を基に，関数定義 FUNC1 が含まれるリポジトリファイル FUNC1.rep が検索して取り込まれます。リポジトリファイルの検索規則については，「33.2.4　リポジトリファイルの参照方法」の「(2)　リポジトリファイルの検索」を参照してください。
4. コンパイラは，リポジトリファイルから取り込んだ関数定義 FUNC1 の情報と，参照しているプログラム定義 PROGRAM1 の関数一意名 FUNCTION FUNC1 の情報の適合をチェックします。このと

き，情報が適合していなければエラーメッセージが出力され，適合していればコンパイルが正常終了します。

(2)　オブジェクト指向機能の場合

オブジェクト指向機能を呼び出す COBOL プログラムの場合の例を，次に示します。なお，オブジェクト指向機能については，「20.　オブジェクト指向機能」を参照してください。



1. クラス定義 CLASS1 が含まれるソースファイル CLASS1.cbl をコンパイルして，リポジトリファイル CLASS1.rep を生成します。
2. クラス定義 CLASS1 を参照するプログラム定義 PROGRAM1 で，環境部構成節のリポジトリ段落に，参照するクラス定義 CLASS1 を指定します。
3. プログラム定義 PROGRAM1 をコンパイルすると，リポジトリ段落の指定を基に，クラス定義 CLASS1 が含まれるリポジトリファイル CLASS1.rep が検索して取り込まれます。リポジトリファイルの検索規則については，「33.2.4　リポジトリファイルの参照方法」の「(2)　リポジトリファイルの検索」を参照してください。
4. コンパイラは，取り込んだクラス定義 CLASS1 の情報から，クラス定義 CLASS1 に含まれるメソッド定義 METH-1 の情報と，参照しているプログラム定義 PROGRAM1 の INVOKE 文に指定したメソッド METH-1 の情報の適合をチェックします。このとき，情報が適合していなければエラーメッセージが出力され，適合していればコンパイルが正常終了します。

33.2.2　ソースファイル，リポジトリファイル，およびリポジトリ段落の関係

リポジトリファイル（.rep）は，COBOL ソースファイルをコンパイルしたときに，そのソースファイルに格納されている翻訳単位ごとの定義情報から生成されます。

リポジトリファイルが生成されるディレクトリの規則について，次に示します。

- 環境変数 CBLREP を設定している場合は，環境変数に指定したディレクトリに生成されます。環境変数 CBLREP に複数のディレクトリを指定している場合は，最初に指定したディレクトリに生成されます。
- 環境変数 CBLREP を設定していない場合は，カレントディレクトリに生成されます。

ここでは，複数の翻訳単位を含む COBOL ソースファイルを例にして，ソースファイル，リポジトリファイル，およびリポジトリ段落の関連を説明します。

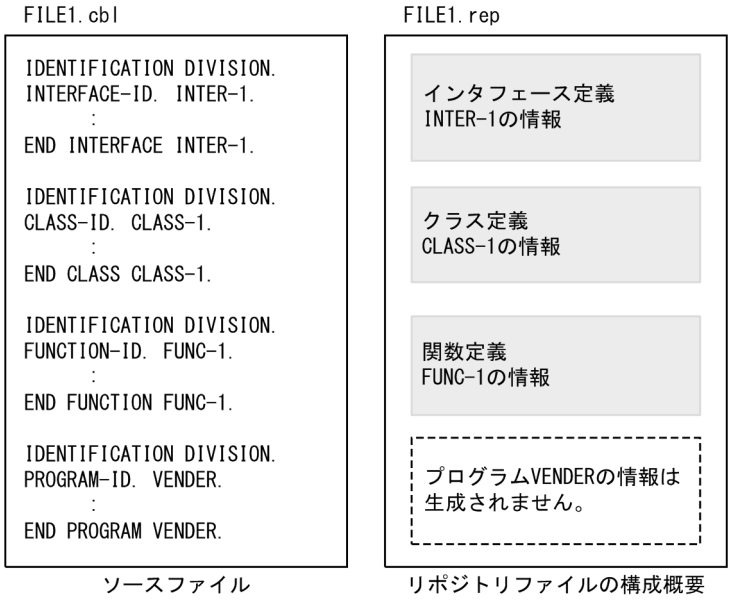
(1) ソースファイルとリポジトリファイルの関係

（例）

クラス定義，インタフェース定義，関数定義，およびプログラム定義を格納しているソースファイル（FILE1.cbl）をコンパイルすると，ソースファイルと同名で拡張子が（.rep）となったリポジトリファイル（FILE1.rep）が作成されます。リポジトリファイルには，それぞれの定義に対応する情報が格納されます。ただし，プログラム定義に対応する情報は，格納されません。

ソースファイルとリポジトリファイルの関係を，次に示します。

図 33-1 ソースファイルとリポジトリファイルの関係



(2) リポジトリファイルとリポジトリ段落の関係

リポジトリ段落に指定したクラス名，インタフェース名，利用者定義関数名，およびプロパティ名は，コンパイル時に同じソースファイル中に存在する翻訳単位，およびリポジトリファイル中から検索された情報と関連づけられます。

（例）

プログラム定義が関数定義を呼び出す場合の，リポジトリ段落の指定例について，次に示します。

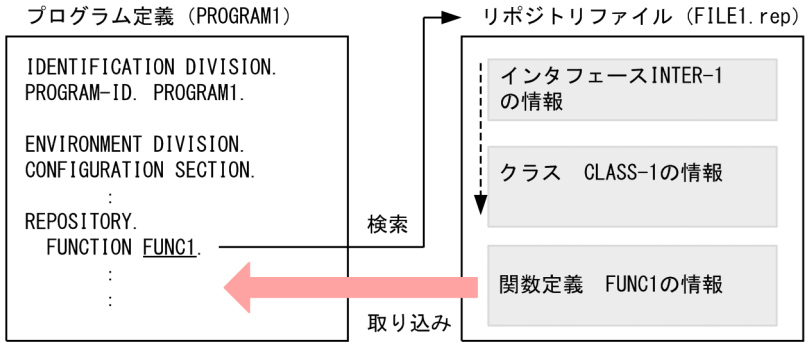
```
IDENTIFICATION DIVISION.
PROGRAM-ID.  PROGRAM1.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
      :
REPOSITORY.  FUNCTION FUNC1.
                1
```

1. プログラム定義のリポジトリ段落で，参照する利用者定義関数名（FUNC1）を宣言します。
2. プログラム定義（PROGRAM1）は，リポジトリ段落の宣言に従ってリポジトリファイル（FILE1.rep）の中の利用者定義関数名（FUNC1）の情報を参照できます。リポジトリファイル内の定義情報を検索する方法については，「33.2.4　リポジトリファイルの参照方法」の「(2)　リポジトリファイルの検索」を参照してください。

リポジトリファイルとリポジトリ段落の関係を，次に示します。

図 33-2　リポジトリファイルとリポジトリ段落の関係



注意事項

- リポジトリ段落で指定するクラス名，インタフェース名，または利用者定義関数名が，リポジトリ段落を指定した翻訳単位と異なるソースファイル中にある場合，参照先の翻訳単位をあらかじめコンパイルして，リポジトリファイルを生成しておく必要があります。
- リポジトリ段落に指定したクラス名，インタフェース名，利用者定義関数名，およびプロパティ名は，リポジトリ段落を指定した定義中だけで有効となります。

33.2.3　リポジトリファイルの生成方法

リポジトリファイルは，ソースファイルにクラス定義，インタフェース定義，または関数定義を含む場合，ソースファイルのコンパイル時に生成されます。このとき，オブジェクトファイルを生成しないでリポジトリファイルだけの生成もできます。リポジトリファイルの生成方法については，「33.3.2　リポジトリファイルの単独生成」を参照してください。

(1)　リポジトリファイルの生成規則

リポジトリファイルの生成規則を，次に示します。

環境変数 CBLREP を指定している場合

1. 環境変数 CBLREP に指定したディレクトリが指定順に検索され，ソースファイルと同じ名称のリポジトリファイルが最初に見つかった時点で，そのリポジトリファイルが更新されます。
2. ソースファイルと同じ名称のリポジトリファイルが見つからなければ，環境変数 CBLREP に指定した最初の有効なディレクトリに，リポジトリファイルが新規に生成されます。

環境変数 CBLREP を指定していない場合，または環境変数 CBLREP に指定したディレクトリが存在しない場合

1. カレントディレクトリ※にソースファイルと同じ名称のリポジトリファイルが存在する場合，そのリポジトリファイルが更新されます
2. カレントディレクトリ※に，ソースファイルと同じ名称のリポジトリファイルが存在しない場合，カレントディレクトリ※に，リポジトリファイルが新規に生成されます。

注※

この場合のカレントディレクトリは，次のようになります。

- ccbl2002 コマンドを使用している場合
  ccbl2002 コマンドを起動したディレクトリ

注意事項

- -Repository,Sup オプションを指定している場合，すでに存在するリポジトリファイルは，更新されません。詳細は，「33.5　リポジトリファイルの生成に関連するコンパイラオプション」を参照してください。
- リポジトリファイルの生成に関連するコンパイラオプションを指定しないでコンパイルした場合，インタフェースの情報（シグニチャ）に変更がないときは，リポジトリファイルが更新されません。詳細は，「33.5　リポジトリファイルの生成に関連するコンパイラオプション」を参照してください。

33.2.4　リポジトリファイルの参照方法

リポジトリ段落に指定したクラス名，インタフェース名，または利用者定義関数名の情報は，コンパイル時に次の順序で検索されます。

1. 同じソースファイル中に存在する翻訳単位のクラス名，インタフェース名，または利用者定義関数名
2. リポジトリファイル中のクラス名，インタフェース名，または利用者定義関数名

注意事項

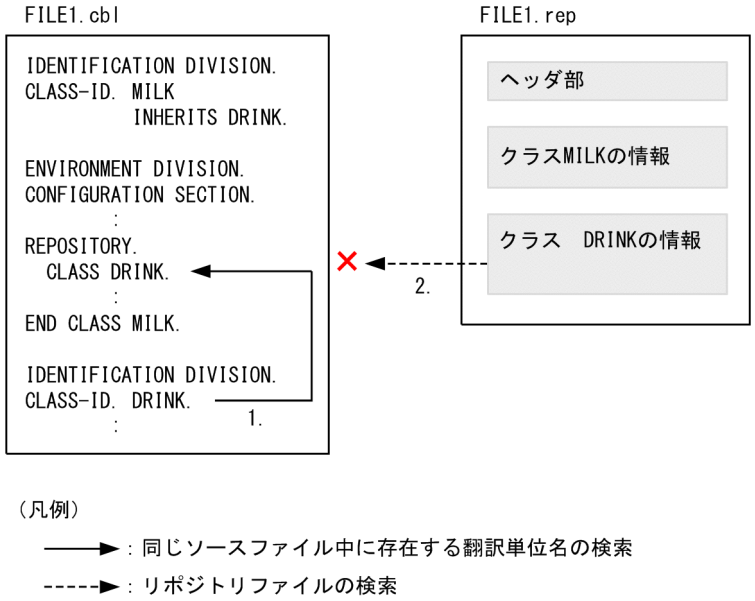
リポジトリ段落に指定したプロパティ名の定義情報は，同じリポジトリ段落に指定したクラス定義やインタフェース定義に含まれるため，個別の定義情報は検索されません。

(1)　同じソースファイル中に存在する翻訳単位名の検索

COBOL2002 コンパイラは，まず同じソースファイル中に存在する翻訳単位中で，リポジトリ段落に指定されたクラス名，インタフェース名，または利用者定義関数名を探します。

同じソースファイル中で，リポジトリ段落よりも後に参照するクラス名，インタフェース名，または利用者定義関数名を指定してもかまいません。また，リポジトリファイルに同じクラス名，インタフェース名，または利用者定義関数名が存在する場合でも，同じソースファイル中の翻訳単位名の情報が優先して使用されます。

図 33-3　同じソースファイル中の翻訳単位名の検索の例



1. クラス名を定義するクラス定義 DRINK が，リポジトリ段落でのクラス名 DRINK の宣言より後に出現してもかまいません。
2. 同じソースファイル中にクラス名 DRINK の情報があるため，リポジトリファイルは参照されません。

(2)　リポジトリファイルの検索

リポジトリファイルは，次の順序で検索されます。

1. 環境変数 CBLREP に指定したディレクトリ
2. カレントディレクトリ※
3. 環境変数 CBLSYSREP に指定したディレクトリ
4. インストールディレクトリ下のrep ディレクトリ

注※

この場合のカレントディレクトリは，次のようになります。

- ccbl2002 コマンドを使用している場合

ccbl2002 コマンドを起動したディレクトリ

環境変数 CBLREP には，リポジトリファイルの検索ディレクトリをカレントディレクトリ以外の場所に指定したい場合に設定します。環境変数 CBLSYSREP は，共用ライブラリとリポジトリファイルだけが提供されている場合など，生成元ソースファイルのないリポジトリファイルの検索ディレクトリを設定します。これらの環境変数の詳細については，「32.6　コンパイラ環境変数」を参照してください。

なお，標準クラスの BASE クラスは，インストールディレクトリ下の rep ディレクトリに格納されています。BASE クラスについては，「COBOL2002　言語　標準仕様編」「12.2　BASE クラス」を参照してください。

翻訳単位名の検索時，同じディレクトリに複数のリポジトリファイルが存在する場合，検索する翻訳単位名と同名のリポジトリファイルの中が検索され，見つからなければ，アルファベット順にリポジトリファイルが検索され，見つかった時点で検索が終了します。

検索したディレクトリ下にリポジトリファイルがない場合や，リポジトリファイルに読み込み権限がない場合，リポジトリファイルが不当な場合は，コンパイル時にエラーメッセージが出力され，コンパイルが終了します。

33.3　リポジトリ段落を指定したソースファイルのコンパイル方法

異なるソースファイルで定義されたクラス名，インタフェース名，利用者定義関数名をリポジトリ段落に指定したソースファイルをコンパイルする場合，それらの定義を含むソースファイルを先にコンパイルしておく必要があります。

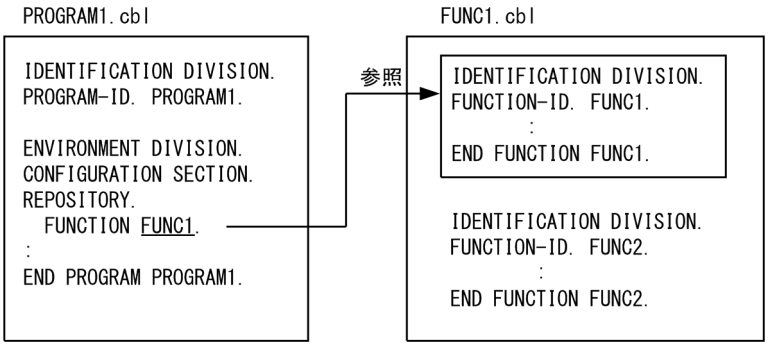
リポジトリ段落で参照するクラス定義，インタフェース定義，または関数定義のような翻訳単位を含むソースファイルを先にコンパイルしておく必要があります。

翻訳単位別にソースファイルをコンパイルする方法を，次に示します。

33.3.1　リポジトリ段落でほかの翻訳単位を参照する場合のコンパイル

リポジトリ段落でほかの翻訳単位を参照する場合，その翻訳単位を含む別のソースファイルを，先にコンパイルしておく必要があります。

例えば，次のように，ソースファイル PROGRAM1.cbl 内のプログラム定義 PROGRAM1 で，リポジトリ段落に関数定義 FUNC1 を指定して参照する場合，FUNC1 を含むソースファイル FUNC1.cbl を先にコンパイルし，その後でソースファイル PROGRAM1.cbl をコンパイルする必要があります。



この場合，次のようなコマンドを指定して，コンパイルします。

```
ccbl2002 FUNC1.cbl -Main,System PROGRAM1.cbl
```

ソースファイルでコンパイル順序を考慮したくない場合は，-Repository,Gen オプションを使用して，コンパイルを実行する前にリポジトリファイルだけを先に生成しておく必要があります。-Repository,Gen オプションの詳細については，「33.3.2　リポジトリファイルの単独生成」を参照してください。

33.3.2　リポジトリファイルの単独生成

ソースファイルのコンパイル時に-Repository,Gen オプションを指定すると，ソースファイルのコンパイルが実行されないで，リポジトリファイルだけを作成できます。-Repository,Gen オプションは，インタ

フェース部分だけを作成した翻訳単位を含むソースファイルをコンパイルしたり，互いに参照し合う翻訳単位をコンパイルしたりする場合に使用します。

ここでは，-Repository,Gen オプションの使用方法について説明します。-Repository,Gen オプションの指定方法や規則の詳細については，「32. COBOL ソースの作成とコンパイル」を参照してください。

(1) インタフェース部分だけを作成した翻訳単位のコンパイル

翻訳単位が未完成であっても，そのインタフェース部分だけが決まっていれば，-Repository,Gen オプションによってリポジトリファイルを作成できます。必要なリポジトリファイルを作成しておけば，リポジトリ段落に未完成の翻訳単位の名前を指定した原始プログラムでも，コンパイルできるようになります。この機能を利用すると，参照先の翻訳単位が未完成の状態でも，参照元の翻訳単位に-Compile,NoLink オプションを指定してコンパイルを実行し，構文レベルの誤りがないか確認できます。

例えば，インタフェース部分だけが決まっているクラス定義 CLASS1 およびそれをリポジトリ段落に指定したプログラム定義 PROGRAM1 があり，それぞれ CLASS1.cbl ファイルおよび PROGRAM1.cbl ファイルに格納されているとします。この場合は，次の手順で PROGRAM1.cbl をコンパイルします。

1. CLASS1 に対応するリポジトリファイルを生成する

   「ccbl2002 -Repository,Gen CLASS1.cbl」と指定すると，CLASS1 がインタフェース部分だけ作成されていても，リポジトリファイル CLASS1.rep が作成されます。

2. CLASS1 のリポジトリファイルの情報を取り込み，PROGRAM1 をコンパイルする

   「ccbl2002 PROGRAM1.cbl」と指定すると，1 で作成した CLASS1.rep が COBOL2002 コンパイラによって取り込まれ，PROGRAM1.cbl がコンパイルされます。

(2) 互いに参照し合う翻訳単位のコンパイル

プログラム単位を除く翻訳単位が，リポジトリ段落に互いの翻訳単位名を指定して，互いに定義情報を参照しあうような場合は，まず-Repository,Gen オプションによって必要なリポジトリファイルを作成しておき，次にそれぞれの翻訳単位をコンパイルしてください。

例えば，互いに参照しているクラス定義 CLASS1 および CLASS2 があり，それぞれ CLASS1.cbl ファイルおよび CLASS2.cbl ファイルに格納されているとします。CLASS1.cbl，CLASS2.cbl の順序でコンパイルしたい場合は，次の手順で互いのクラス定義をコンパイルします。

1. CLASS2 に対応するリポジトリファイルを生成する

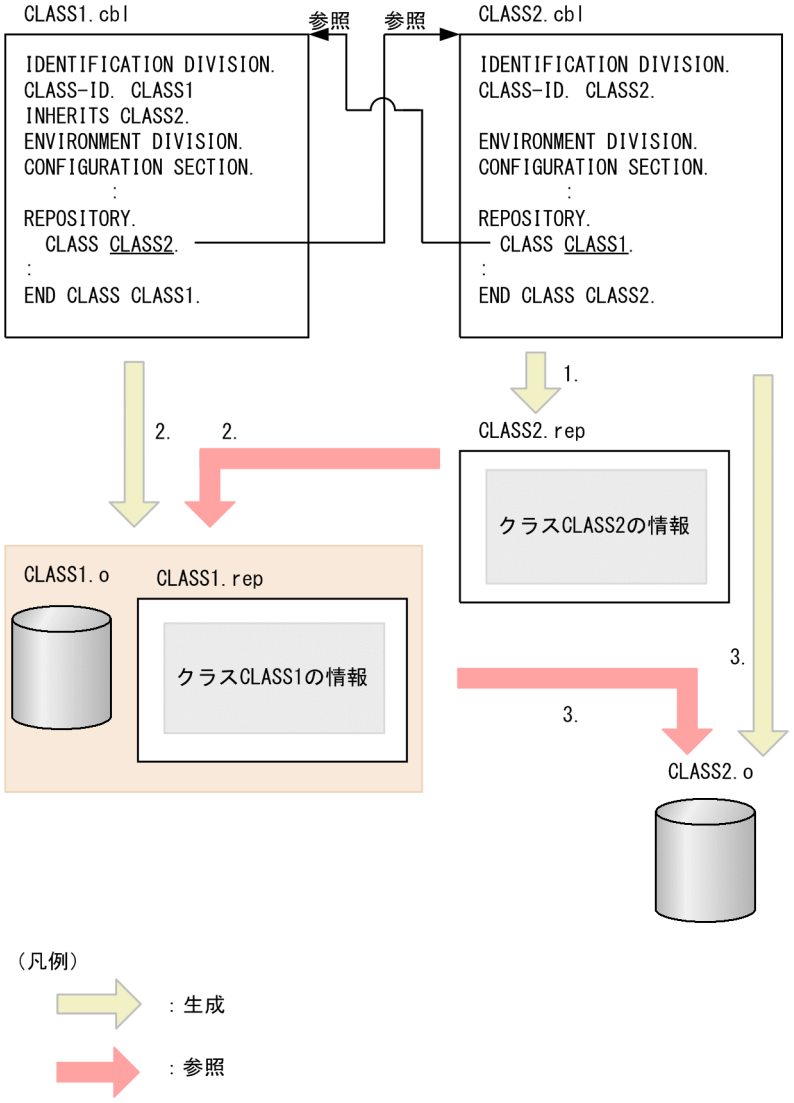
   「ccbl2002 -Repository,Gen CLASS2.cbl」と指定すると，リポジトリファイル CLASS2.rep が作成されます。

2. CLASS2 のリポジトリファイルの情報を取り込み，CLASS1 をコンパイルする

   「ccbl2002 CLASS1.cbl」と指定すると，1 で作成された CLASS2.rep が COBOL2002 コンパイラによって取り込まれ，CLASS1.cbl がコンパイルされます。このとき，CLASS1 のリポジトリファイル CLASS1.rep が作成されます。

3. CLASS1 のリポジトリファイルの情報を取り込み，CLASS2 をコンパイルする

「ccbl2002 CLASS2.cbl」と指定すると，2 で作成された CLASS1.rep が COBOL2002 コンパイラによって取り込まれ，CLASS2.cbl がコンパイルされます。



注意事項

- 一つのソースファイル中でスタックコンパイルとして翻訳単位を複数定義し，その中で互いに参照する場合は，リポジトリファイルを先に作成する必要はなく，一度にコンパイルできます。
- 互いに参照し合う翻訳単位を含むソースファイルから別々の共用ライブラリを作成できません。

33.3.3　プログラム定義だけのコンパイル

プログラム定義だけで構成されているソースファイルをコンパイルする場合，リポジトリ段落を使用してほかのプログラム定義を参照しないため，特に順番を考慮しないでコンパイルできます。

33.4.1　外部リポジトリに関連したコンパイルエラー発生時の対処方法

この項では，外部リポジトリを使用した COBOL プログラムをコンパイルしたときに出力されるコンパイルエラーの対処方法について，次のような場合に分けて説明します。

- 外部リポジトリに翻訳単位が見つからない場合
- メソッドまたは利用者定義関数の定義情報が適合しない場合
- インタフェースを実装，またはメソッドの上書きができない場合

(1)　外部リポジトリに翻訳単位が見つからない場合

リポジトリ段落に記述した参照先の翻訳単位の定義情報が外部リポジトリに見つからない場合，コンパイルエラーが出力されます。コンパイルリストを使った対処方法を次に説明します。

注意

外部リポジトリの検索手順は，「33.2.4　リポジトリファイルの参照方法」の「(2)　リポジトリファイルの検索」を参照してください。コンパイルリストについては，「付録 D　コンパイルリスト」を参照してください。

図 33–4　参照元の翻訳単位を含む COBOL プログラムのコンパイルリスト（外部リポジトリに翻訳単位が見つからない場合）

```
                        :
     3                  ENVIRONMENT DIVISION.
     4                  CONFIGURATION SECTION.
     5                  REPOSITORY.
     6                  CLASS CLS1.                                  }..... 1.
                             ?
KCCC8922C-S 外部リポジトリに翻訳単位"CLS1"が見つかりません。
     7                  DATA DIVISION.
                        :
```

1. コンパイルリストで，?が付いた個所（上記の図の 1.）の記述を参照し，翻訳単位の名称（CLS1），CLASS，INTERFACE，FUNCTION，および PROPERTY の指定が正しいかを確認してください。
2. 記述が誤っている場合は，COBOL プログラムの誤りを修正してください。
3. 記述が正しい場合は，記述した翻訳単位を定義した COBOL プログラムを先にコンパイルし，「33.2.4 リポジトリファイルの参照方法」の「(2)　リポジトリファイルの検索」に記載している検索対象のディレクトリにリポジトリファイルを作成してください。

(2)　メソッドまたは利用者定義関数の定義情報が適合しない場合

INVOKE 文や関数一意名で，呼び起こし対象のメソッドまたは利用者定義関数の定義情報が適合しない場合，コンパイルエラーが出力されます。コンパイルリストを使った対処方法を次に説明します。

呼び起こすメソッドを定義したクラス，または利用者定義関数が定義された COBOL プログラムを特定できる場合は，「(a)　COBOL プログラムを使った対処方法」に記載している手順でコンパイルエラーの要因を修正してください。COBOL プログラムが特定できない場合は，「(b)　リポジトリファイルを使った対処方法」に記載している手順でコンパイルエラーの要因を修正してください。

注意

適合チェックの詳細は，マニュアル「COBOL2002　言語　標準仕様編」「10.7　引数と返却項目の適合」を参照してください。コンパイルリストについては，「付録 D　コンパイルリスト」を参照してください。

(a)　COBOL プログラムを使った対処方法

図 33–5　参照元の翻訳単位を含む COBOL プログラムのコンパイルリスト（メソッドまたは利用者定義関数の定義情報が適合しない場合：その 1）

```
                                    :
     9                PROCEDURE DIVISION.
    10                  INVOKE CLS1 'MF1' RETURNING DATA1.        }..... 1.
                        ?                                         }
KCCC8947C-S
呼び起こされるメソッド又は利用者定義関数の実引数が対応する仮引数に適合していません。又
は，仮結果が実結果に適合していません。
                                    :
```

1. コンパイルリストで，?が付いた個所（上記の図の 1.）の記述を参照し，呼び起こすメソッドを定義したクラス，または利用者定義関数の名称（CLS1）を確認してください。
2. 手順 1.のクラスまたは利用者定義関数が定義された COBOL プログラムを参照し，定義情報を確認してください。
3. 確認した定義情報を使って COBOL プログラムの誤りを修正してください。

(b)　リポジトリファイルを使った対処方法

図 33–6　参照元の翻訳単位を含む COBOL プログラムのコンパイルリスト（メソッドまたは利用者定義関数の定義情報が適合しない場合：その 2）

```
                                    :
* リポジトリファイル名
   行番号    リポジトリファイル名
         6  /users/cobol2002/CLASS1.rep                          ..... 3.
                                    :
     3                ENVIRONMENT DIVISION.
     4                CONFIGURATION SECTION.
     5                REPOSITORY.
     6                CLASS CLS1.                                ..... 2.
     7                DATA DIVISION.
     8                 01 DATA1 PIC 99.
     9                PROCEDURE DIVISION.
    10                  INVOKE CLS1 'MF1' RETURNING DATA1.       }..... 1.
                         ?                                       }
KCCC8947C-S
呼び起こされるメソッド又は利用者定義関数の実引数が対応する仮引数に適合していません。又
は，仮結果が実結果に適合していません。
                                    :
```

1. コンパイルリストで，?が付いた個所（上記の図の 1.）の記述を参照し，呼び起こすメソッドを定義したクラス，または利用者定義関数の名称（CLS1）を確認してください。
2. 手順 1.のクラスまたは利用者定義関数の名称（CLS1）を，参照元の COBOL プログラムで最初に定義した，リポジトリ段落の行番号 6 を確認してください（上記の図の 2.）。
3. 手順 2.の行番号 6 を基に，参照しているリポジトリファイル名（/users/cobol2002/CLASS1.rep）をコンパイルリストで確認してください（上記の図の 3.）。
4. 手順 3.で確認したリポジトリファイル名（/users/cobol2002/CLASS1.rep）から，リポジトリ管理ツールを使用して定義情報を出力してください。リポジトリ管理ツールの仕様については，「33.4.2 リポジトリ管理ツール」を参照してください。
5. リポジトリ管理ツールが出力した定義情報（次の図の 1.）を使って COBOL プログラムの誤りを修正してください。リポジトリ管理ツールの出力例を次に示します。

図 33-7　リポジトリ管理ツールの出力例（メソッドまたは利用者定義関数の定義情報が適合しない場合：その 3）

```
                                :
- << クラス情報 >> ----------------------------------------------

    クラス名              : CLS1
    生成時刻              : YYYY-MM-DD HH:MM:SS

                                :
  ---  ファクトリオブジェクト  ---
                                :
    [ メソッド情報 ]

      メソッド名          引数／返却項目の情報
    ------------------------------------------------------------
     MF1                [RETURN]      PIC  X(2)         USAGE DISPLAY      …… 1.
                                :
```

(3)　インタフェースを実装，またはメソッドの上書きができない場合

インタフェースを実装するとき，または継承したクラスのメソッドを上書きするとき，実装または上書きするメソッド原型（またはメソッド）がない，または適合しない場合は，コンパイルエラーが出力されます。

インタフェースを実装する場合を例に，コンパイルリストを使った対処方法を次に説明します。

実装するインタフェースが定義された COBOL プログラムを特定できる場合は，「(a)　COBOL プログラムを使った対処方法」に記載している手順でコンパイルエラーの要因を修正してください。COBOL プログラムが特定できない場合は，「(b)　リポジトリファイルを使った対処方法」に記載している手順でコンパイルエラーの要因を修正してください。

注意

適合チェックの詳細は，マニュアル「COBOL2002　言語　標準仕様編」「5.2.7　適合とインタフェース」を参照してください。コンパイルリストについては，「付録 D　コンパイルリスト」を参照してください。

(a) COBOL プログラムを使った対処方法

図 33-8 参照元の翻訳単位を含む COBOL プログラムのコンパイルリスト（インタフェースを実装，またはメソッドの上書きができない場合：その 1）

```
                                :
      3                 ENVIRONMENT DIVISION.
      4                 CONFIGURATION SECTION.
      5                 REPOSITORY.
      6                 INTERFACE INT1.                                  }
                                  ?                                      }..... 1.
KCCC3231C-S IMPLEMENTS句に指定されたインタフェース名に含まれるメソッド原型を実装していません。
      7       B         ID DIVISION.
      8                  FACTORY. IMPLEMENTS INT1.
                                :
```

1. コンパイルリストで，?が付いた個所（上記の図の 1.）の記述を参照し，実装するインタフェース名（INT1）を確認してください。
2. 手順 1.のインタフェース（INT1）が定義された COBOL プログラムを参照し，定義しているメソッド原型を確認してください。
3. 確認したメソッド原型をすべて実装するように COBOL プログラムを修正してください。

(b) リポジトリファイルを使った対処方法

図 33-9 参照元の翻訳単位を含む COBOL プログラムのコンパイルリスト（インタフェースを実装，またはメソッドの上書きができない場合：その 2）

```
                                :
* リポジトリファイル名
   行番号    リポジトリファイル名
         6  /users/cobol2002/INTERFACE1.rep                          ..... 2.
                                :
      3                 ENVIRONMENT DIVISION.
      4                 CONFIGURATION SECTION.
      5                 REPOSITORY.
      6                 INTERFACE INT1.                                  }
                                  ?                                      }..... 1.
KCCC3231C-S IMPLEMENTS句に指定されたインタフェース名に含まれるメソッド原型を実装していません。
      7       B         ID DIVISION.
      8                  FACTORY. IMPLEMENTS INT1.
                                :
```

1. コンパイルリストで，?が付いた個所（上記の図の 1.）の記述を参照し，実装するインタフェースの名称（INT1）を確認してください。
2. 手順 1.のインタフェースの名称（INT1）を，参照元の COBOL プログラムで最初に定義したリポジトリ段落の行番号 6 を確認してください（上記の図の 1.）。
3. 手順 2.の行番号 6 を基に，参照しているリポジトリファイル名（/users/cobol2002/INTERFACE1.rep）をコンパイルリストで確認してください（上記の図の 2.）。

4. 手順 3.で確認したリポジトリファイル名（/users/cobol2002/INTERFACE1.rep）から，リポジトリ管理ツールを使用して定義情報を出力してください。リポジトリ管理ツールの仕様については，「33.4.2 リポジトリ管理ツール」を参照してください。

5. リポジトリ管理ツールが出力した定義情報（次の図の 1.）を使って，インタフェース（INT1）で定義されたすべてのメソッド原型を実装するように COBOL プログラムを修正してください。リポジトリ管理ツールの出力例を次に示します。

図 33-10　リポジトリ管理ツールの出力例（インタフェースを実装，またはメソッドの上書きができない場合：その 3）

```
                                :
- << インタフェース情報 >> ------------------------------------

     インタフェース名    : INT1
     生成時刻            :YYYY-MM-DD HH:MM:SS

     [ メソッド情報 ]

       メソッド名          引数／返却項目の情報
     ------------------------------------------------------------
      MF2                [BY REFERENCE] PIC  X(4)         USAGE DISPLAY.     …… 1.
                                :
```

33.4.2　リポジトリ管理ツール

外部リポジトリに関連したコンパイルエラーが発生したとき，リポジトリファイルに格納された情報を確認すると，コンパイルエラーの原因を容易に特定できることがあります。

リポジトリ管理ツールは，リポジトリファイルに格納された翻訳単位の定義情報を出力できます。この項では，リポジトリ管理ツール（cbl2krep コマンド）の機能について説明します。

コンパイルエラー発生時の対処方法については，「33.4.1　外部リポジトリに関連したコンパイルエラー発生時の対処方法」を参照してください。

(1)　リポジトリ管理ツールの概要

リポジトリ管理ツールには，次に示す機能があります。

- 翻訳単位の情報の表示
- オプション概略の一覧（ヘルプ機能）

(a)　形式

```
cbl2krep 〔リポジトリファイル名〕 … 〔オプション〕 …
```

リポジトリファイル名

リポジトリファイル名を指定します。ファイルの表示をするときは，指定した順序に従ってリポジトリファイルが処理されます。

オプション

リポジトリファイルをどのように処理するかを指定するオプションです。次のオプションを指定できます。

```
-list -help
```

オプションの詳細については，「(2)　リポジトリ管理ツールの機能詳細」を参照してください。

リポジトリファイル名およびオプションの指定を省略した場合，リポジトリ管理ツールの指定形式を出力します。

```
Usage : cbl2krep [repository-files] [options]

  -help
  -list
```

(b)　終了コード

リポジトリ管理ツールの終了コードを次の表に示します。

表 33-1　リポジトリ管理ツールの終了コード

終了コード	意味
0	正常終了
1	エラーが発生
2	回復不能エラーが発生

(c)　注意事項

リポジトリ管理ツールを使用するときの注意事項を次に示します。

1. リポジトリファイル名は，.rep を付けて指定する必要があります。
2. 複数のオペランドを指定する場合は，各オペランドを空白またはタブで区切ります。
3. 複数のリポジトリファイル名を指定できる個所には，ワイルドカード（*）が使用できます。
4. -help オプションを指定した場合，そのほかのオプションやリポジトリファイル名の指定は無視されます。
5. オプションで使用する文字は，英大文字と英小文字は等価とみなされます。
6. リポジトリファイル名，および翻訳単位名で使用する文字は，英大文字と英小文字が等価とみなされません。

(2)　リポジトリ管理ツールの機能詳細

(a)　翻訳単位情報の表示

形式

```
cbl2krep リポジトリファイル名1 … -list
```

機能

指定したリポジトリファイル中の翻訳単位の情報を標準出力に出力します。ただし，標準出力に出力されるのは，指定したリポジトリファイル内の定義情報だけです。継承したクラスやインタフェースのプロパティ情報，およびメソッド情報は出力されません。

オペランド

リポジトリファイル名 1

表示したいリポジトリファイルを指定します。

-list

リポジトリファイル中の翻訳単位の情報を標準出力に出力します。

-list オプションを指定したときに出力されるクラス定義の翻訳単位情報の内容を次の図に，関数定義の翻訳単位情報の表示を「図 33–12　-list オプションを指定したときに出力される関数定義の翻訳単位情報の内容」に示します。

図 33-11　-list オプションを指定したときに出力されるクラス定義の翻訳単位情報の内容

```
*****************************************************************

   Repostory File Information List

     REP File Name : ORANGE.rep
     Creation Date : YYYY-MM-DD HH:MM:SS                          }…… 1.
     Compiler Name : COBOL2002 (X) VV-RR                          }

*****************************************************************

- << CLASS Information >> ---------------------------------------

     CLASS NAME          : ORANGE
     CREATION DATE       : YYYY-MM-DD HH:MM:SS                    }…… 2.

     Inherited CLASS/INTERFACE Information :                      }
                                                                  }…… 3.
       DRINK (CLASS)                                              }

     Referred  CLASS/INTERFACE Information :                      }
                                                                  }…… 4.
       I-INT1 (INTERFACE)                                         }

  --- FACTORY OBJECT  ---

     [ PROPERTY Information ]                                     }
                                                                  }
       Property-Name       Kind      Attributes                   }
     ------------------------------------------------------       }…… 5.
      VITAMIN_C          (GET)   PIC  S9(9)        USAGE COMP    }
      VITAMIN_C          (SET)   PIC  S9(9)        USAGE COMP    }

     [ METHOD Information ]                                       }
                                                                  }…… 6.
       None                                                       }

  --- INSTANCE OBJECT ---

     [ PROPERTY Information ]                                     }
                                                                  }…… 5.
       None                                                       }

     [ METHOD Information ]                                       }
                                                                  }
       Method-Name        Parameters/Return                       }
     ------------------------------------------------------       }…… 6.
      INITIALIZEOBJ                                               }
      SUPPLEMENT         [BY REFERENCE] PIC  S9(9)        USAGE COMP }
```

1. リポジトリファイルヘッダ情報

   リポジトリファイル名，リポジトリファイルの生成時刻（YYYY-MM-DD 年-月-日，HH:MM:SS 時:分:秒），およびリポジトリファイルを生成したコンパイラ名（バージョン（VV-RR）を含む）が表示されます。コンパイラ名の（X）は，COBOL2002 の識別記号を示します。詳細は「付録 K.2　このマニュアルでの表記」を参照してください。リポジトリファイル名はコマンドラインに指定した名前で表示されます。

2. 生成時刻

   リポジトリファイル内の翻訳単位情報の生成時刻（YYYY-MM-DD 年-月-日，HH:MM:SS 時:分:秒）が表示されます。

3. 継承クラス・インタフェース情報

リポジトリ段落に指定した翻訳単位名のうち，INHERITS 句に指定したクラス名およびインタフェース名が出力されます。括弧内には，翻訳単位の種別が出力されます。

4. 参照翻訳単位情報

リポジトリ段落に指定した翻訳単位名のうち，INHERITS 句に指定していない翻訳単位名が出力されます。括弧内には，翻訳単位の種別が出力されます。

5. プロパティ情報

クラス定義に指定したオブジェクトプロパティの情報（プロパティ名，参照（GET）／更新（SET）種別，データの属性情報）が出力されます。プロパティ情報は，ファクトリオブジェクトとインスタンスオブジェクトとで別々に出力されます。出力される項目がない場合は，「None」が出力されます。出力される属性情報については，「属性情報の詳細」に示します。

6. メソッド情報

クラス定義に指定したメソッドの情報（メソッド名，引数／返却項目の情報）が出力されます。メソッド情報は，ファクトリオブジェクトとインスタンスオブジェクトとで別々に出力されます。出力される項目がない場合は，「None」が出力されます。

図 33-12　-list オプションを指定したときに出力される関数定義の翻訳単位情報の内容

```
*****************************************************************

   Repostory File Information List

     REP File Name : FUNC.rep                                          }····· 1.
     Creation Date : YYYY-MM-DD HH:MM:SS                               }
     Compiler Name : COBOL2002 (X) VV-RR                               }

*****************************************************************

- << FUNCTION Information >> ------------------------------------

     FUNCTION NAME      : FUNCTION2
     CREATION DATE      : YYYY-MM-DD HH:MM:SS

     Referred  CLASS/INTERFACE Information :                           }····· 2.
       FUNCTION1 (FUNCTION)                                            }

     [ FUNCTION Information ]                                          }
                                                                       }
       Function-Name      Parameters/Return                            }····· 3.
     ------------------------------------------------------------      }
      FUNCTION2          [RETURN]       PIC  X(10)        USAGE DISPLAY }
                         [BY REFERENCE] PIC  9(9)         USAGE COMP    }
```

1. リポジトリファイルヘッダ情報

リポジトリファイル名，リポジトリファイルの生成時刻（YYYY-MM-DD 年-月-日，HH:MM:SS 時:分:秒），およびリポジトリファイルを生成したコンパイラ名（バージョン（VV-RR）を含む）が表示されます。コンパイラ名の（X）は，COBOL2002 の識別記号を示します。詳細は「付録 K.2　このマニュアルでの表記」を参照してください。リポジトリファイル名はコマンドラインに指定した名前で表示されます。

2. 参照翻訳単位情報

リポジトリ段落に指定した翻訳単位名が出力されます。括弧内には，翻訳単位の種別が出力されます。

3. 関数情報

関数定義の情報（関数名，引数／返却項目の情報）が出力されます。

属性情報の詳細

データ項目の種類と出力される属性情報または引数／返却項目の情報の対応を次の表に示します。

表 33-2　データ項目の種類と属性情報または引数／返却項目の情報

データ項目の種類	出力される属性情報または引数／返却項目の情報
固定長集団項目	-
可変長集団項目	-
日本語集団項目	GROUP-USAGE NATIONAL
数字編集項目	-
英数字編集項目	-
日本語編集項目	-
外部 10 進項目	USAGE DISPLAY ただし，SIGN 句の属性情報がある場合，SIGN 句の属性情報に従って次のどれかが表示されます。 USAGE DISPLAY SIGN IS LEADING USAGE DISPLAY SIGN IS LEADING SEPARATE USAGE DISPLAY SIGN IS TRAILING SEPARATE
内部 10 進項目	USAGE COMP-3
外部浮動小数点項目	USAGE DISPLAY
内部浮動小数点項目	USAGE COMP-1 USAGE COMP-2
アドレスデータ項目／ポインタ項目	USAGE ADDRESS
指標データ項目	USAGE INDEX
2 進項目	USAGE COMP USAGE COMP-5 USAGE COMP-X
ブール項目	USAGE BIT USAGE DISPLAY
英字／英数字項目	USAGE DISPLAY
日本語項目	USAGE NATIONAL
オブジェクト参照データ項目	OBJECT REFERENCE

データ項目の種類	出力される属性情報または引数／返却項目の情報
	（インタフェース名／[FACTORY OF] ACTIVE CLASS／[FACTORY OF] クラス名 [ONLY] 指定ありの場合，その指定も出力されます）

（凡例）

－：属性情報なし

(b)　オプション概略の一覧

形式

```
cbl2krep -help
```

機能

オプションの概略を一覧表示します。

(3)　リポジトリ管理ツール使用時のエラーメッセージ

(a)　エラーメッセージの形式

リポジトリ管理ツールが出力するメッセージの形式を，次に示します。

形式

```
cbl2krep : error エラー番号: メッセージ内容
```

このマニュアルでは，リポジトリ管理ツールが出力するエラーメッセージを，次の形式で記載します。

エラー番号

メッセージ内容

(要因)

エラーが返される要因を示す。

(S)

システムの処置を示す。

(P)

プログラム作成者の処置を示す。

エラーメッセージの内容を次に示します。

(b)　エラーメッセージの内容

リポジトリ管理ツールが出力するメッセージのエラー番号およびメッセージ内容を，次に示します。

0001

Cannot open file -- '******'.

0002

Cannot read file -- '******'.

0003

annot write file -- '******'.

(要因)

リポジトリファイルの書き込みで I/O エラーが発生した。次の要因が考えられる。

1. カレントディレクトリがネットワークディレクトリのとき，ネットワーク障害が発生している。
2. ディスクの空きがない。または，ディスク障害が発生している。

(S)

cbl2krep の実行を中止する。

(P)

1. 何度かコンパイルを試みる。
2. コンピュータの管理者に問い合わせる。

0004

File -- '******' is not a repository file.

0005

Repository file -- '******' is not readable because of unsuitable target machine.

0006

Repository file -- '******' is corrupt.

0007

Cannot find compilation unit name -- '******' in repository file.

0008

Duplicate compilation unit name -- '******'.

0009

Logical error occurred. Error code : ******.

(要因)

cbl2krep の実行で，内部的に論理エラーが発生した。

(S)

cbl2krep の実行を中止する。

(P)

当社保守員に連絡する。

0010

Out of memory.

(要因)

cbl2krep が稼働するメモリが足りない。

(S)

cbl2krep の実行を中止する。

(P)

削除できる資源を削除して再実行する。

0011

Invalid option -- '******' is specified.

0012

Option -- '******' has no parameter.

0013

Option -- '******' is too long.

0014

Repository file -- '******' is not accessable by this version cbl2krep.

0017

Cannot find class name -- '******' in repository file.

0018

Cannot find interface name -- '******' in repository file.

0019

Unnecessary parameter specified for option -- '******'.

0020

No repository file specified.

0023

Cannot open a file because many files are opened by other programs.

(要因)

ほかのプログラムで，システムの制限までファイルを開いている。

(S)

cbl2krep の実行を中止する。

(P)

ほかのプログラムで使用しているファイルを閉じる。

0025

Cannot find file -- '******'.

0027（Linux(x64)で有効）

Unable to invoke command -- '******'.

(要因)

cbl2krep から呼び出すコマンドが存在しない，または壊れている。

(S)

cbl2krep の実行を中止する。

(P)

当社保守員に連絡する。

33.5　リポジトリファイルの生成に関連するコンパイラオプション

リポジトリファイルの生成に関連するコンパイラオプションについて説明します。

-Repository,Gen オプション

-Repository,Gen オプションを指定した場合，ソースファイルからオブジェクトファイルを生成しないで，リポジトリファイルだけを生成します。詳細は，「33.3.2　リポジトリファイルの単独生成」を参照してください。

-Repository,Sup オプション

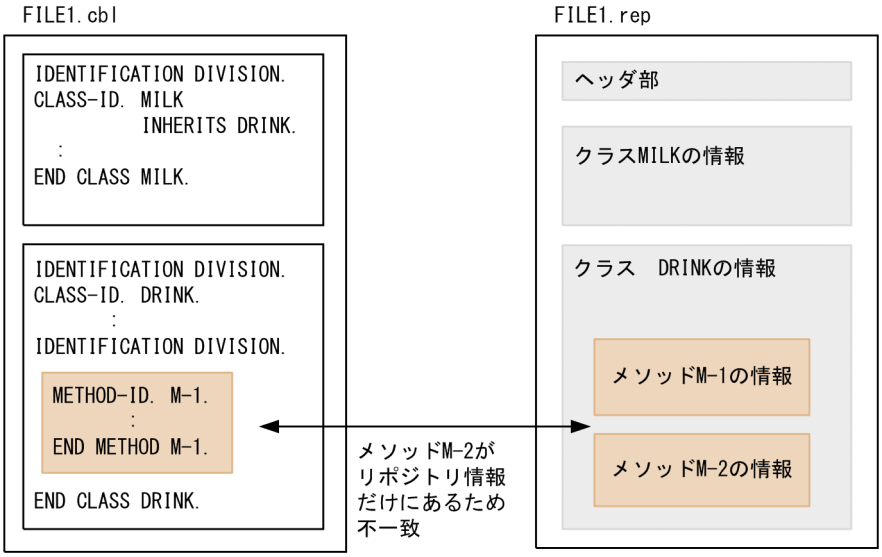
-Repository,Sup オプションを指定した場合，コンパイル時にリポジトリファイルが更新されません。ただし，リポジトリファイルが存在しない場合は，-Repository,Sup オプションの指定に関係なく，リポジトリファイルが新規に生成されます。

-Repository,Sup オプションを指定した場合は，コンパイルによってオブジェクトファイルが生成されます。

-RepositoryCheck オプション

-RepositoryCheck オプションを指定した場合，同じソースファイル中の翻訳単位の定義とリポジトリファイル中の情報に相違があるとき，コンパイル時に KCCC3301C-W の警告メッセージが出力されます。このとき，リポジトリファイルは更新されません。

-RepositoryCheck オプションによって警告メッセージが出力される例を，次に示します。



-RepositoryCheck オプションを指定した場合，FILE1.cbl のクラス DRINK の定義情報と，FILE1.rep のクラス DRINK の情報に相違があるため，警告メッセージが出力されます。

このとき，FILE1.rep は更新されません。

おのおののコンパイラオプションを指定した場合，コンパイル時にリポジトリファイルが更新されるかどうかを，次に示します。

指定するコンパイラオプション	インタフェースの情報（シグニチャ）に変更あり	インタフェースの情報（シグニチャ）に変更なし
オプションなし	○	×
-Repository,Gen オプションを指定	○	○
-Repository,Sup オプションを指定	×	×
-RepositoryCheck オプションを指定	×	×

（凡例）

○：リポジトリファイルが更新される

×：リポジトリファイルが更新されない

34 実行可能ファイルと共用ライブラリの作成

この章では，プログラムをコンパイルして作成したオブジェクトファイルをリンクして，実行可能ファイルや共用ライブラリを作成する方法について説明します。

34.1　実行可能ファイルの作成方法

実行可能ファイルとは，一つ以上のプログラムによって構成され，制御プログラムから直接呼び出して実行できるファイルのことです。

ここでは，ccbl2002 コマンドを実行して COBOL プログラムから実行可能ファイルを作成する方法と，C プログラムのオブジェクトファイルなどとリンクする方法について説明します。

34.1.1　コンパイルとリンクを同時に実行する方法

ccbl2002 コマンドを使用すると，COBOL プログラムのコンパイルとリンクを同時に実行して，実行可能ファイルを生成できます。

ccbl2002 コマンドの詳細については，「32.　COBOL ソースの作成とコンパイル」を参照してください。

ccbl2002 コマンドを使って実行可能ファイルを生成する例を，次に示します。

(1)　一つの COBOL プログラムから実行可能ファイルを生成する

一つの COBOL プログラムをコンパイル，リンクして実行可能ファイルを生成する例を，次に示します。

ソースファイル名称：test01.cbl

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. TEST01.
                :
       PROCEDURE DIVISION.
                :
           STOP RUN.
```

ccbl2002 コマンドの指定

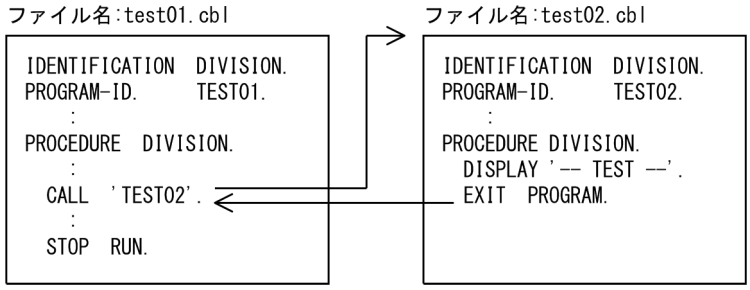
COBOL プログラム"test01.cbl"に-Main,System オプションを指定し，実行可能ファイル名称に"test02"を指定します。

```
ccbl2002 -Main,System test01.cbl -OutputFile test02
```

上記のコマンドを実行すると，実行可能ファイル"test02"が生成されます。

(2)　複数の COBOL プログラムから実行可能ファイルを生成する

複数の COBOL プログラムをコンパイル，リンクして実行可能ファイルを生成する例を，次に示します。



ccbl2002 コマンドの指定

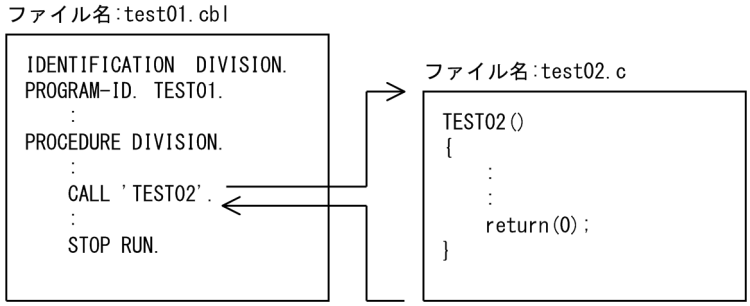
主プログラムである"test01.cbl"に-Main,System オプションを指定し，副プログラムである"test02.cbl"には何も指定しないで ccbl2002 コマンドを実行します。

```
ccbl2002 -Main,System test01.cbl test02.cbl -OutputFile test03
```

上記のコマンドを実行すると，実行可能ファイル"test03"が生成されます。

(3) COBOL プログラムと C プログラムの混在した実行可能ファイルを生成する

COBOL プログラムと C プログラムが混在する実行可能ファイルを生成する例を，次に示します。



ccbl2002 コマンドの指定

主プログラムである"test01.cbl"に-Main,System オプションを指定し，副プログラムである"test02.c"には何も指定しないで ccbl2002 コマンドを実行します。

```
ccbl2002 -Main,System test01.cbl test02.c -OutputFile test03
```

上記のコマンドを実行すると，実行可能ファイル"test03"が生成されます。

(4) 注意事項

(a) AIX システムで，オブジェクトファイルを指定して実行可能ファイルを作成する場合の注意事項

- AIX COBOL2002 V3 以前に作成した COBOL オブジェクトファイル，-OldStyleObject オプションを指定して作成した COBOL オブジェクトファイル，C プログラムのオブジェクトファイルを混在させるときは，-OldLinkOpt,GCBypass オプションを指定してください。-OldLinkOpt,GCBypass オ

プションの詳細は，「32.5.10　リンクの設定」の「(5)　-OldLinkOpt,GCBypass オプション（AIX で有効）」を参照してください。

(b)　AIX システムで，アーカイブファイルを指定して実行可能ファイルを作成する場合の注意事項

- AIX COBOL2002 V4 で作成したオブジェクトファイルを含むアーカイブファイルを指定して実行可能ファイルを作成すると，実行可能ファイルのサイズが，AIX COBOL2002 V3 以前に作成した実行可能ファイルのサイズと比べて，極端に肥大化することがあります。このサイズの肥大化は，-OldStyleObject オプションで回避できます。-OldStyleObject オプションの詳細，およびサイズの肥大化については，「32.5.10　リンクの設定」の「(6)　-OldStyleObject オプション（AIX で有効）」を参照してください。
- アーカイブファイル内の静的な参照関係がないオブジェクトファイルは，特にリンカオプションの指定がないと，リンク対象になりません。アーカイブファイル内の静的な参照関係がないオブジェクトファイルをリンク対象とするときは，次のどちらかのリンカオプションを指定してください。
  - アーカイブファイル内のすべてのオブジェクトファイルをリンク対象とするとき

    -bkeepfile:アーカイブファイル名
  - アーカイブファイル内の特定のプログラムのオブジェクトファイルだけをリンク対象とするとき

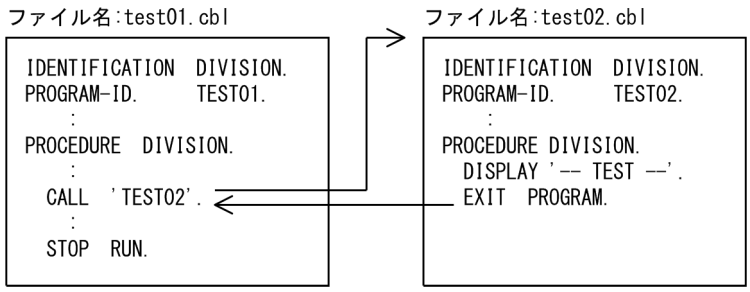
    -u プログラム名

34.1.2　コンパイルとリンクを別々に実行する方法

COBOL プログラムが複数ある場合，あらかじめ ccbl2002 コマンドでオブジェクトファイルを作成しておき，その後 ccbl2002 コマンドや cc コマンドでオブジェクトファイルをリンクして実行可能ファイルを生成できます。

(1)　ccbl2002 コマンドによるリンク方法

ccbl2002 コマンドでコンパイルとリンクを別々に実行する方法を，次に示します。



ccbl2002 コマンドの指定

まず，COBOL プログラム"test01.cbl"および"test02.cbl"をコンパイルし，オブジェクトファイルを生成します。コンパイルだけを実行し，リンクを実行しない場合は，ccbl2002 コマンドに-Compile,NoLink オプションを指定します。

```
ccbl2002 -Main,System test01.cbl -Compile,NoLink
ccbl2002 test02.cbl -Compile,NoLink
```

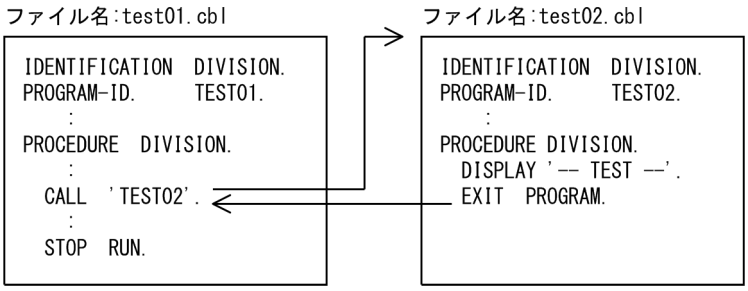
次に，生成された"test01.o"および"test02.o"を ccbl2002 コマンドを使用してリンクします。

```
ccbl2002 test01.o test02.o -OutputFile test03
```

上記のコマンドを実行すると，実行可能ファイル"test03"が生成されます。

(2)　cc コマンドによるリンク方法

cc コマンドでコンパイルとリンクを別々に実行する方法を，次に示します。



ccbl2002 コマンドの指定

まず，COBOL プログラム"test01.cbl"および"test02.cbl"をコンパイルし，オブジェクトファイルを生成します。コンパイルだけを実行し，リンクを実行しない場合は，ccbl2002 コマンドに-Compile,NoLink オプションを指定します。

```
ccbl2002 -Main,System test01.cbl test02.cbl -Compile,NoLink
```

次に，生成された"test01.o"および"test02.o"を cc コマンドを使用してリンクします。

```
cc test01.o test02.o -o test03 -L/opt/HILNGcbl2k/lib -lcbl2k
   -lcbl2kml -lm
```

上記のコマンドを実行すると，実行可能ファイル"test03"が生成されます。

注

-L オプションに指定する検索パスや-l オプションに指定するライブラリ名については，「34.1.5 cc コマンドおよび ld コマンドの-l オプション」の「(1)　COBOL2002 実行時ライブラリを使用する場合」を参照してください。

(3) 注意事項

(a) AIX システムで，オブジェクトファイルを指定して実行可能ファイルを作成する場合の注意事項

- ccbl2002 コマンドでリンクするとき

  AIX COBOL2002 V3 以前に作成した COBOL オブジェクトファイル，-OldStyleObject オプションを指定して作成した COBOL オブジェクトファイル，C プログラムのオブジェクトファイルを混在させるときは，-OldLinkOpt,GCBypass オプションを指定してください。-OldLinkOpt,GCBypass オプションの詳細は，「32.5.10　リンクの設定」の「(5)　-OldLinkOpt,GCBypass オプション（AIX で有効）」を参照してください。

- cc／ld コマンドでリンクするとき

  AIX システムでは，静的な参照関係がないファイルは，cc／ld コマンドで特にリンカオプションを指定しないと，リンク対象となりません。そのため，AIX COBOL2002 V3 以前に作成した COBOL オブジェクトファイル，-OldStyleObject オプションを指定して作成した COBOL オブジェクトファイル，C プログラムのオブジェクトファイルをリンクする場合，静的な参照関係がないファイルは，リンク対象とならないことがあります。

  静的な参照関係がないファイルもリンク対象とするときは，次のどれかのリンカオプションを指定してください。

  - -bgcbypass リンカオプション（-bgcbypass:入力ファイル数）
  - -bkeepfile リンカオプション（-bkeepfile:入力ファイル名）
  - -u リンカオプション（-u プログラム名）

(b) AIX システムで，アーカイブファイルを指定して実行可能ファイルを作成する場合の注意事項

- AIX COBOL2002 V4 で作成したオブジェクトファイルを含むアーカイブファイルを指定して実行可能ファイルを作成すると，実行可能ファイルのサイズが，AIX COBOL2002 V3 以前に作成した実行可能ファイルのサイズと比べて，極端に肥大化することがあります。このサイズの肥大化は，-OldStyleObject オプションで回避できます。-OldStyleObject オプションの詳細，およびサイズの肥大化については，「32.5.10　リンクの設定」の「(6)　-OldStyleObject オプション（AIX で有効）」を参照してください。
- アーカイブファイル内の静的な参照関係がないオブジェクトファイルは，特にリンカオプションの指定がないと，リンク対象となりません。アーカイブファイル内の静的な参照関係がないオブジェクトファイルをリンク対象とするときは，次のどちらかのリンカオプションを指定してください。
  - アーカイブファイル内のすべてのオブジェクトファイルをリンク対象とするとき

    -bkeepfile:アーカイブファイル名
  - アーカイブファイル内の特定のプログラムのオブジェクトファイルだけをリンク対象とするとき

    -u プログラム名

(c) Linux システムでの注意事項

- Linux(x86) COBOL2002 で生成したオブジェクトを，cc コマンドを使用してリンクする場合，32bit アプリケーションとしてリンクしなければなりません。OS が Linux(x64)の場合は，-m32 オプションの指定が必要です。なお，cc コマンドのオプションの詳細は，システムのマニュアルを参照してください。

34.1.3 ccbl2002 コマンドの-l オプション

ccbl2002 コマンドを使って実行可能ファイルを生成する場合，使用する機能によって，-l オプションの指定が必要になります。

注意事項

ccbl2002 コマンドは，システムのリンカ（cc コマンド）を呼び出して実行可能ファイルを作成します。リンカは，コマンド行に指定された順序どおりにファイルを処理します。このため，ccbl2002 コマンドでは，使用する機能のライブラリ（-lisam，-lrsort など）をその機能を使用する COBOL プログラムよりあとに指定してください。

(1) 整列併合機能を使用する場合

SORT ライブラリ（-lrsort，-lrsort64 オプションなど）を使用する場合は，ライブラリの検索パスに次の指定が必要です。

AIX(32)，Linux(x86)の場合

```
/opt/HISORTlib/lib
```

AIX(64)，Linux(x64)の場合

```
/opt/HISORTlib64/lib
```

(a) マルチスレッド機能を使用しない場合

AIX(32)，Linux(x86)の場合

-lrsort オプションの指定が必要です。

AIX(64)，Linux(x64)の場合

-lrsort64 オプションの指定が必要です。

(b) マルチスレッド機能を使用する場合

AIX(32)，Linux(x86)の場合

-lmsort オプションの指定が必要です。

AIX(64)，Linux(x64)の場合

-lmsort64 オプションの指定が必要です。

(2) 画面機能を使用する場合（AIX(32)で有効）

XMAP の入出力ドライバのためのオプションの指定が必要です。

詳細については，マニュアル「画面・帳票サポートシステム XMAP3 Server」を参照してください。

(3) 索引ファイル機能を使用する場合

ISAM ライブラリ（-lisam，-lisam64 オプションなど），SORT ライブラリ（-lrsort，-lrsort64 オプションなど）を使用する場合は，ライブラリの検索パスに次の指定が必要です。

AIX(32)，Linux(x86)の場合

```
/opt/HIISlib/lib，/opt/HISORTlib/lib
```

AIX(64)，Linux(x64)の場合

```
/opt/HIISlib64/lib，/opt/HISORTlib64/lib
```

(a) マルチスレッド機能を使用しない場合

AIX(32)，Linux(x86)の場合

-lisam，-lrsort オプションの指定が必要です。

AIX(64)，Linux(x64)の場合

-lisam64，-lrsort64 オプションの指定が必要です。

(b) マルチスレッド機能を使用する場合

AIX(32)，Linux(x86)の場合

-lmisam，-lmsort オプションの指定が必要です。

AIX(64)，Linux(x64)の場合

-lmisam64，-lmsort64 オプションの指定が必要です。

(4) データコミュニケーション機能をシミュレーションによってテストデバッグする場合

AIX(32)，Linux の場合

-lcbl2kdc オプションの指定が必要です。

AIX(64)の場合

-lcbl2kdc64 オプションの指定が必要です。

(5)　プリンタを使用する場合（AIX(32)で有効）

書式印刷機能（-XMAP,LinePrint オプション）で XMAP3 を使用する場合，-lxmovl または-lxpw オプションが必要です。

(6)　CGI プログラム作成支援機能を使用する場合（AIX(32)で有効）

-lcbl2kcgi オプションの指定が必要です。

(7)　マルチスレッド機能を使用する場合

Linux の場合

-lpthread オプションの指定が必要です。

AIX の場合

-lpthreads オプションの指定が必要です。

(8)　HiRDB による索引ファイル入出力機能を使用する場合（AIX で有効）

HiRDB のライブラリが必要です。

HiRDB のライブラリ指定については，マニュアル「HiRDB の UAP 開発ガイドマニュアル」を参照してください。

(9)　XML 対応 COBOL プログラムを使用する場合

ccbl2002 コマンドを使って，XML ドキュメントにアクセスする COBOL プログラムをリンクできません。

XML ドキュメントにアクセスする COBOL プログラムのコンパイルとリンクについては，マニュアル「COBOL2002　XML 連携機能ガイド」を参照してください。

(10)　ODBC インターフェース機能を使用する場合（Linux で有効）

ODBC ドライバマネージャのライブラリが必要です。ODBC ドライバマネージャのライブラリ指定については，unixODBC に関するリファレンスなどを参照してください。

34.1.4　ccbl2002 コマンド使用時の検索ライブラリの種別指定

ccbl2002 コマンドを使って，リンクするときに検索するライブラリの種別（アーカイブファイルまたは共用ライブラリ）を指定するには，次のオプションを指定します。

なお，ccbl2002 コマンドでは，指定したライブラリのあとに内部的にシステムライブラリが指定されます。システムライブラリにはアーカイブ提供がない場合があるため，オプションでアーカイブファイルを

指定した場合には，必ず，続けて共用ライブラリ（システムの標準）検索のオプションを指定する必要があります。

AIX の場合

-b オプションを指定します。

-b オプションは，AIX の cc コマンドまたは ld コマンドのオプションです。共用ライブラリ検索のオプションの指定は「-bdynamic」です。

（例）

```
ccbl2002 -Main,System a.cbl b.o -L. -bstatic -lxxx -bdynamic
```

Linux(x86)の場合

-Wl および-B オプションを指定します。

-Wl および-B オプションは，Linux(x86)の cc コマンドのオプションです。共用ライブラリ検索のオプションの指定は「-Bdynamic」です。

（例）

```
ccbl2002 -UniObjGen -Main,System a.cbl b.o -L. -Wl,-Bstatic -lxxx -Wl,-Bdynamic
```

Linux(x64)の場合

-Wl および-B オプションを指定します。

-Wl および-B オプションは，Linux(x64)の cc コマンドのオプションです。共用ライブラリ検索のオプションの指定は「-Bdynamic」です。

（例）

```
ccbl2002 -UniObjGen -Main,System a.cbl b.o -L. -Wl,-Bstatic -lxxx -Wl,-Bdynamic
```

34.1.5　cc コマンドおよび ld コマンドの-l オプション

cc コマンドおよび ld コマンドを使って実行可能ファイルを生成したいときは，-l オプションについて次のような指定が必要です。cc コマンドおよび ld コマンドの詳細については，システムのマニュアルを参照してください。-l オプションの詳細については，「34.1.3　ccbl2002 コマンドの-l オプション」を参照してください。

(1)　COBOL2002 実行時ライブラリを使用する場合

COBOL2002 実行時ライブラリには共用ライブラリが提供されています。次に共用ライブラリを示します。

システム名	共用ライブラリ	
	ファイル名	使用時に必要なオプション
AIX(32)	libcbl2k.a	-lcbl2k※1

システム名	共用ライブラリ	
	ファイル名	使用時に必要なオプション
AIX(64)	libcbl2k64.a	-lcbl2k64※2
Linux	libcbl2k.so	-lcbl2k※1

注※1

-lcbl2k オプション指定時は，常に共用ライブラリが参照されます。

また，-lcbl2k オプションに続いて-lcbl2kml オプションの指定が必要です。COBOL2002 でのライブラリ指定文字列（-lcbl2k など）を使用するとき，ライブラリの検索パスに AIX(32)，または Linux(x86)の場合は「/opt/HILNGcbl2k/lib」の指定が，Linux(x64)の場合は「/opt/HILNGcbl2k64/lib」の指定が必要です。

注※2

-lcbl2k64 オプション指定時は，常に共用ライブラリが参照されます。

また，-lcbl2k64 オプションに続いて-lcbl2kml64 オプションの指定が必要です。COBOL2002 でのライブラリ指定文字列（-lcbl2k64 など）を使用するとき，ライブラリの検索パスに「/opt/HILNGcbl2k64/lib」の指定が必要です。

注意事項

cc コマンドおよび ld コマンドは，コマンド行に指定された順序どおりにファイルを処理します。このため，cc コマンドおよび ld コマンドには，次の順序で引数を指定します。

- -lcbl2k，-lcbl2k64，-lcbl2kml，および-lcbl2kml64 オプションは，COBOL プログラムよりあとに指定する。
- COBOL プログラムで使用する機能のライブラリ（-lisam，-lisam64，-lrsort，-lrsort64 など）は，-lcbl2k，-lcbl2k64，-lcbl2kml，および-lcbl2kml64 オプションよりあとに指定する。

（例）

AIX(32)の場合

```
cc  xxx.o  -L/opt/HILNGcbl2k/lib -lcbl2k -lcbl2kml …
    -L/opt/HIISlib/lib -lisam -L/opt/HISORTlib/lib
    -lrsort … -lm
```

AIX(64)の場合

```
cc  xxx.o  -q64 -L/opt/HILNGcbl2k64/lib -lcbl2k64
    -lcbl2kml64 … -L/opt/HIISlib64/lib -lisam64
    -L/opt/HISORTlib64/lib -lrsort64 … -lm
```

Linux(x86)の場合

```
cc  xxx.o  -L/opt/HILNGcbl2k/lib -lcbl2k -lcbl2kml …
    -L/opt/HIISlib/lib -lisam -L/opt/HISORTlib/lib
    -lrsort … -lm
```

Linux(x64)の場合

```
cc  xxx.o  -L/opt/HILNGcbl2k64/lib -lcbl2k -lcbl2kml …
    -L/opt/HIISlib64/lib -lisam64 -L/opt/HISORTlib64/lib
    -lrsort64 … -lm
```

(2) cc コマンドを使用する場合

cc コマンドを使用して実行可能ファイルを作成するときは，ダイナミックローダライブラリオプション（-ldl オプション）の指定が必要です。

(3) DISPLAY 文および浮動小数点形式データを使用する場合

-lm オプションの指定が必要です。

(4) テストデバッガを使用する場合（-TDInf，-CVInf，-TestCmd,Full，および-TestCmd,Sim オプション指定時）

-TDInf，-CVInf，-TestCmd,Full，および-TestCmd,Sim オプションのどれかを指定してコンパイルした COBOL オブジェクトをリンクする場合，次の指定が必要です。

AIX(32)，Linux(x86)の場合

```
cc  xxx.o -L/opt/HILNGcbl2k/lib -lcbl2k -lcbl2kml … -ldl …
```

AIX(64)の場合

```
cc  xxx.o -q64 -L/opt/HILNGcbl2k64/lib -lcbl2k64
    -lcbl2kml64 … -ldl …
```

Linux(x64)の場合

```
cc  xxx.o -L/opt/HILNGcbl2k64/lib -lcbl2k -lcbl2kml … -ldl …
```

(5) 組み込み関数機能を使用する場合

次に示す組み込み関数を使用する場合，-lm オプションの指定が必要です。

ACOS 関数，ASIN 関数，ATAN 関数，COS 関数，SIN 関数，TAN 関数，LOG 関数，LOG10 関数，SQRT 関数

(6) マルチスレッド機能を使用する場合

AIX(32)の場合

-lcbl2kmp および-lpthreads オプションの指定が必要です。

AIX(64)の場合

-lcbl2kmp64 および-lpthreads オプションの指定が必要です。

Linux の場合

-lcbl2kmp および-lpthread オプションの指定が必要です。

(7) CGI プログラム作成支援機能を使用する場合（AIX(32)で有効）

-lcbl2kcgi オプションの指定が必要です。

(8) HiRDB による索引ファイル入出力機能を使用する場合（AIX(32)で有効）

AIX(32)の場合

-lcbl2krd オプションの指定および HiRDB のライブラリが必要です。

AIX(64)の場合

-lcbl2krd64 オプションの指定および HiRDB のライブラリが必要です。

なお，HiRDB ライブラリを指定する際は COBOL プログラムで次に示す機能を使用しないでください。

- OLTP(OpenTP1)環境下での実行
- 複数接続機能
- 64 ビットモード（AIX(32)の場合）
- 32 ビットモード（AIX(64)の場合）

HiRDB のライブラリ指定については，マニュアル「HiRDB の UAP 開発ガイドマニュアル」を参照してください。

(9) XML 対応 COBOL プログラムを使用する場合

XML アクセス用実行時ライブラリが必要です。ライブラリ指定については，マニュアル「COBOL2002 XML 連携機能ガイド」を参照してください。

(10) 整列併合機能を使用する場合

SORT ライブラリ（-lrsort，-lrsort64 オプションなど）を使用する場合は，ライブラリの検索パスに次の指定が必要です。

AIX(32)，Linux(x86)の場合

```
/opt/HISORTlib/lib
```

AIX(64)，Linux(x64)の場合

```
/opt/HISORTlib64/lib
```

(a) マルチスレッド機能を使用しない場合

AIX(32)，Linux(x86)の場合

-lrsort オプションの指定が必要です。

AIX(64)，Linux(x64)の場合

-lrsort64 オプションの指定が必要です。

(b) マルチスレッド機能を使用する場合

AIX(32)，Linux(x86)の場合

-lmsort オプションの指定が必要です。

AIX(64)，Linux(x64)の場合

-lmsort64 オプションの指定が必要です。

(11) 索引ファイル機能を使用する場合

ISAM ライブラリ（-lisam，-lisam64 オプションなど），SORT ライブラリ（-lrsort，-lrsort64 オプションなど）を使用する場合は，ライブラリの検索パスに次の指定が必要です。

AIX(32)，Linux(x86)の場合

```
/opt/HIISlib/lib，/opt/HISORTlib/lib
```

AIX(64)，Linux(x64)の場合

```
/opt/HIISlib64/lib，/opt/HISORTlib64/lib
```

(a) マルチスレッド機能を使用しない場合

AIX(32)，Linux(x86)の場合

-lisam，-lrsort オプションの指定が必要です。

AIX(64)，Linux(x64)の場合

-lisam64，-lrsort64 オプションの指定が必要です。

(b) マルチスレッド機能を使用する場合

AIX(32)，Linux(x86)の場合

-lmisam，-lmsort オプションの指定が必要です。

AIX(64)，Linux(x64)の場合

-lmisam64，-lmsort64 オプションの指定が必要です。

(12) ODBC インターフェース機能を使用する場合（Linux で有効）

-lcbl2kodbc オプションの指定および ODBC ドライバマネージャのライブラリが必要です。

ODBC ドライバマネージャのライブラリ指定については，unixODBC に関するリファレンスなどを参照してください。

34.2　共用ライブラリの作成方法

34.2.1　共用ライブラリの作成

共用ライブラリとは，副プログラムだけで構成され，実行可能ファイル中のプログラムから呼ばれることで実行できるファイルのことです。ここでは，次に示すファイルを共用ライブラリと呼びます。

システム名	共用ライブラリ
AIX	xxx.a xxx.so※
Linux	xxx.so

注※

xxx.so ライブラリを，リンク時に-l オプションで指定する場合は，-brtl オプションの指定が必要です。-brtl オプションの詳細については，システムのマニュアルを参照してください。

共用ライブラリを作成する方法は，実行可能ファイルの生成と同じです。ここでは，共用ライブラリの作成方法のうち，実行可能ファイルの生成と異なる部分について説明します。

(1)　共用ライブラリを作成する

ccbl2002 コマンドで COBOL プログラムから共用ライブラリを作成する場合は，-PIC,Std オプションを指定します。

共用ライブラリを作成する例を次に示します。

ソースファイル名称：test02.cbl

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. TEST02.
       PROCEDURE DIVISION.
           DISPLAY '--- TEST02  START ---'.
           DISPLAY '--- TEST02    END ---'.
```

ccbl2002 コマンドの指定

AIX(32)の場合

```
ccbl2002 -PIC,Std  test02.cbl
ld -o test02.a test02.o -bpT:0x10000000 -bpD:0x20000000
   -bnoentry -bM:SRE -bexpall
   -L/opt/HILNGcbl2k/lib -lcbl2k -lcbl2kml -lm -lc
```

上記のコマンドを実行すると，共用ライブラリ"test02.a"が生成されます。

AIX(64)の場合

```
ccbl2002 -PIC,Std  test02.cbl
ld -o test02.a test02.o -b64 -bpT:0x100000000 -bpD:0x110000000
```

```
   -bnoentry -bM:SRE -bexpall
   -L/opt/HILNGcbl2k64/lib -lcbl2k64 -lcbl2kml64 -lm -lc
```

上記のコマンドを実行すると，共用ライブラリ"test02.a"が生成されます。

Linux(x86)の場合

```
ccbl2002 -PIC,Std -UniObjGen test02.cbl
ld -shared -o test02.so test02.o -Bstatic
   -L/opt/HILNGcbl2k/lib -lcbl2kml -Bdynamic -lc
```

上記のコマンドを実行すると，共用ライブラリ"test02.so"が生成されます。

Linux(x64)の場合

```
ccbl2002 -PIC,Std -UniobjGen test02.cbl
ld -shared -o test02.so test02.o -Bstatic
   -L/opt/HILNGcbl2k64/lib -lcbl2kml -Bdynamic -lc
```

上記のコマンドを実行すると，共用ライブラリ"test02.so"が生成されます。

(2)　アーカイブ形式の共用ライブラリを作成する

AIX の場合，ar コマンドを使用してアーカイブ形式の共用ライブラリを作成できます。

アーカイブ形式の共用ライブラリにした場合，動的なリンクによって，アーカイブ形式の共用ライブラリ中のプログラムを呼び出せます。詳細については，「18.　プログラムの呼び出し」を参照してください。

また，ar コマンドについては，システムのマニュアルを参照してください。

(1)の共用ライブラリ"test02.a"と，別の共用ライブラリ"test03.a"をアーカイブ形式の共用ライブラリにまとめる場合の例を示します。

ソースファイル名称：test03.cbl

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. TEST03.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
         01 DAT1   PIC X(10).
       PROCEDURE DIVISION.
           MOVE ALL 'A' TO DAT1.
           DISPLAY DAT1.
```

ccbl2002 コマンドおよび ar コマンドの指定

```
ccbl2002 -PIC,Std  test03.cbl
ld -o test03.a test03.o -bpT:0x10000000 -bpD:0x20000000
   -bnoentry -bM:SRE -bexpall
   -L/opt/HILNGcbl2k/lib -lcbl2k -lcbl2kml -lm -lc
ar rv libSUB.a test02.a test03.a
```

(3) ほかのプログラムを呼ぶ副プログラムを作成する

共用ライブラリに含まれるプログラム（副プログラム）からほかのプログラムを呼び出すとき，次の条件に当てはまる場合は，コンパイル時に-DynamicLink,Call オプションを指定する必要があります。

- 動的なリンクによって，ほかのプログラムを呼び出す場合
- 一意名指定の CALL 文によって，ほかのプログラムを呼び出す場合

ほかのプログラムを呼ぶ副プログラムの作成例を，次に示します。

副プログラムが COBOL プログラムの場合

"test01.cbl"から"test02.cbl"を呼び出す場合の例を示します。

AIX(32)の場合

```
ccbl2002 -PIC,Std -DynamicLink,Call test01.cbl test02.cbl
ld -o test01.a test01.o -bpT:0x10000000
   -bpD:0x20000000 -bnoentry -bM:SRE
   -bexpall -L/opt/HILNGcbl2k/lib
   -lcbl2k -lcbl2kml -lm -lc
ld -o test02.a test02.o -bpT:0x10000000
   -bpD:0x20000000 -bnoentry -bM:SRE
   -bexpall -L/opt/HILNGcbl2k/lib
   -lcbl2k -lcbl2kml -lm -lc
```

AIX(64)の場合

```
ccbl2002 -PIC,Std -DynamicLink,Call test01.cbl test02.cbl
ld -o test01.a test01.o -b64 -bpT:0x100000000
   -bpD:0x110000000 -bnoentry -bM:SRE
   -bexpall -L/opt/HILNGcbl2k64/lib
   -lcbl2k64 -lcbl2kml64 -lm -lc
ld -o test02.a test02.o -b64 -bpT:0x100000000
   -bpD:0x110000000 -bnoentry -bM:SRE
   -bexpall -L/opt/HILNGcbl2k64/lib
   -lcbl2k64 -lcbl2kml64 -lm -lc
```

Linux(x86)の場合

```
ccbl2002 -PIC,Std -UniObjGen -DynamicLink,Call test01.cbl test02.cbl
ld -shared -o test01.so test01.o -Bstatic
   -L/opt/HILNGcbl2k/lib -lcbl2kml -Bdynamic -lc
ld -shared -o test02.so test02.o -Bstatic
   -L/opt/HILNGcbl2k/lib -lcbl2kml -Bdynamic -lc
```

Linux(x64)の場合

```
ccbl2002 -PIC,Std -UniObjGen -DynamicLink,Call test01.cbl test02.cbl
ld -shared -o test01.so test01.o -Bstatic
   -L/opt/HILNGcbl2k64/lib -lcbl2kml -Bdynamic -lc
ld -shared -o test02.so test02.o -Bstatic
   -L/opt/HILNGcbl2k64/lib -lcbl2kml -Bdynamic -lc
```

注意事項

- コンパイルするとき，-DynamicLink,Call オプションを指定しないで，別コンパイルすることもできます。
- オブジェクトファイルは，一つの共用ライブラリにまとめることもできます。

副プログラム"test01.c"から共用ライブラリを作成する場合

副プログラム"test01.c"を作成する場合の例を示します。

AIX(32)の場合

```
cc -c test01.c
cc -G -o test01.a test01.o
```

・C プログラムで作成した共用ライブラリ（test01.a）と COBOL プログラムで作成した共用ライブラリ（test02.a）をアーカイブ形式の共用ライブラリにまとめることもできます。

```
    ccbl2002 -PIC,Std test02.cbl
    ld -o test02.a test02.o -bpT:0x10000000
       -bpD:0x20000000 -bnoentry -bM:SRE
       -bexpall -L/opt/HILNGcbl2k/lib
       -lcbl2k -lcbl2kml -lm -lc

    ar rv libSUB2.a test01.a test02.a
```

アーカイブ形式の共用ライブラリにした場合，動的なリンクによって，アーカイブ形式の共用ライブラリ中のプログラムを呼び出せます。詳細については，「18.　プログラムの呼び出し」を参照してください。

また，ar コマンドについては，システムのマニュアルを参照してください。

AIX(64)の場合

```
cc -c test01.c -q64
cc -G -o test01.a test01.o -q64
```

・C プログラムで作成した共用ライブラリ（test01.a）と COBOL プログラムで作成した共用ライブラリ（test02.a）をアーカイブ形式の共用ライブラリにまとめることもできます。

```
    ccbl2002 -PIC,Std test02.cbl
    ld -o test02.a test02.o -b64 -bpT:0x100000000
       -bpD:0x110000000 -bnoentry -bM:SRE
       -bexpall -L/opt/HILNGcbl2k64/lib
       -lcbl2k64 -lcbl2kml64 -lm -lc

    ar -X64 rv libSUB2.a test01.a test02.a
```

アーカイブ形式の共用ライブラリにした場合，動的なリンクによって，アーカイブ形式の共用ライブラリ中のプログラムを呼び出せます。詳細については，「18.　プログラムの呼び出し」を参照してください。

また，ar コマンドについては，システムのマニュアルを参照してください。

Linux の場合

```
cc -fpic -c test01.c
ld -shared -o test01.so test01.o -lc
```

(4)　注意事項

(a)　AIX システムでの注意事項

AIX COBOL2002 V4 で作成したオブジェクトファイルを含むアーカイブファイルを指定して共用ライブラリを作成すると，共用ライブラリのサイズが，AIX COBOL2002 V3 以前に作成した共用ライブラリのサイズと比べて，極端に肥大化することがあります。このサイズの肥大化は，-OldStyleObject オプションで回避できます。-OldStyleObject オプションの詳細，およびサイズの肥大化については，「32.5.10 リンクの設定」の「(6)　-OldStyleObject オプション（AIX で有効）」を参照してください。

(b)　Linux システムでの注意事項

Linux(x86) COBOL2002 で生成したオブジェクトを，cc コマンドまたは ld コマンドを使用してリンクする場合，32bit アプリケーションとしてリンクしなければなりません。OS が Linux(x64)の場合は，それぞれ次のオプションの指定が必要です。なお，cc コマンドおよび ld コマンドのオプションの詳細は，システムのマニュアルを参照してください。

- cc コマンドの場合：-m32 オプション
- ld コマンドの場合：-melf_i386 オプション

35 プログラムの実行

この章では，コンパイル・リンクによって作成した COBOL プログラムを実行する方法について説明します。

35.1　実行可能ファイルの起動方法

COBOL2002 で作成したプログラムを実行する方法を，以下に示します。

実行可能ファイル名称を次のように入力します。

形式

```
実行可能ファイル名称　〔実行可能ファイルの引数〕
```

実行可能ファイル名称

実行可能ファイル名称は，原始プログラムをコンパイル時に-OutputFile オプションで指定したファイル名です。-OutputFile オプションを指定しなかったときのファイル名称は，a.out になります。

-OutputFile オプションの詳細は，「32.5.10　リンクの設定」の「(2)　-OutputFile オプション」を参照してください。

実行可能ファイルの引数

指定した引数は，原始プログラム中の PROCEDURE DIVISION の USING 指定の引数，または CBLARGV サービスルーチンなどによって取り込めます。実行可能ファイルの引数を複数指定するときは，それぞれを空白で区切ります。引数の受け取り方法の詳細は，「16.　COBOL の実行単位」を参照してください。

35.2　画面環境の設定（AIX で有効）

実行時に使用する画面環境の設定は，リソースファイルによって行います。

リソースファイルについては，「12.　画面入出力機能（AIX で有効）」，「29.5.1　JCPOPUP（AIX で有効）」を参照してください。

35.3　プログラムの実行環境の設定

35.3.1　実行時環境変数の設定方法

実行時の環境は実行時環境変数で設定します。

実行時環境変数は，次の方法で設定します。

- システムに従った環境変数の設定方法

(1)　システムに従った環境変数の設定方法

sh（Bシェル）の場合

形式

```
環境変数名=環境変数の値
export 環境変数名
```

注意事項

セミコロン（;）などの特殊文字を環境変数の値に含むときは，アポストロフィ（'）または引用符（"）で囲んでください。

(2)　実行時環境変数の規則

実行時環境変数の規則を，次に示します。

- 設定する環境変数名，および環境変数の値の大文字と小文字は等価とみなしません。

35.3.2　実行時環境変数の一覧

実行時環境変数の一覧を，次に示します。

なお，未サポートの実行時環境変数が指定された場合，指定された内容は無効となって実行されます。

(1)　一般

環境変数名	指定する内容	OS	
		AIX	Linux
CBLENVMAX	環境変数値の書き出しを行う DISPLAY 文の実行回数の設定	○	○
CBLEXVALUE	EXTERNAL 句の指定のあるデータ項目の初期値を指定する	○	○
CBLLANG	動作する言語環境（文字コード）を指定する	○	○

環境変数名	指定する内容	OS	
		AIX	Linux
CBLLPATH	動的なリンク時，共用ライブラリ検索ディレクトリを指定する場合，検索したい共用ライブラリのあるディレクトリの指定	○	○
CBLLSLIB	動的なリンク時，検索する共用ライブラリを限定したい場合，検索したい共用ライブラリ名の指定	○	○
CBLLTAG	動的なリンクの動作を指示するオプションを指定する	○	×
CBLPGMSEARCHTRC	動的なリンクによる共用ライブラリの検索情報とプレロード時の共用ライブラリのロード情報を出力するファイル名を指定する	○	○
CBLPGMSEARCHTRC_SIZE	プログラム検索トレースファイル名を切り替えるサイズを指定する	○	○
CBLPRELOAD	動的なリンクで検索対象となる共用ライブラリ名称を記述したプレロードリストのファイル名を指定する	○	○
CBLUNIENDIAN	用途が NATIONAL の項目に対する Unicode のバイトオーダを指定する	○	○
CBLUPSI	外部スイッチの状態	○	○
CBL_SYSERR	実行時エラーメッセージの出力先ファイル名	○	○

（凡例）

○：サポートしている

×：サポートしていない

(2) 少量データ

環境変数名	指定する内容	OS	
		AIX	Linux
CBLDATE	ACCEPT 文，CURRENT-DATE 関数，MOVE 文（日付と時刻用）でシステムから受け取る西暦年月日	○	○
CBLDAY	ACCEPT 文でシステムから受け取る通算日付	○	○
CBLSTDIOLVL	DISPLAY 文によってデータを標準出力，または標準エラー出力する場合，COBOL 実行時ライブラリが使用するシステム入出力関数のレベルの指定	○	○
CBL_STOPNOADV	STOP　定数文を実行したとき，メッセージ ID と定数 1 の直後の改行文字を出力するかどうか	○	○
CBL_SYSIN	FROM SYSIN 指定の ACCEPT 文での入力ファイル名	○	○
CBL_SYSOUT	UPON SYSOUT 指定の DISPLAY 文での出力ファイル名	○	○
CBL_SYSPUNCH	UPON SYSPUNCH 指定の DISPLAY 文での出力ファイル名	○	○
CBL_SYSSTD	FROM SYSSTD 指定の ACCEPT 文での入力ファイル名	○	○

（凡例）

○：サポートしている

(3) ファイル

環境変数名	指定する内容	OS		
		AIX(32)	AIX(64)	Linux
CBLCSVCHAR	-NumCsv オプションを指定して CSV 編成ファイルを数値として読み込むとき，無視する文字列	○	○	○
CBLCSVINIT	CSV 編成ファイルの READ 文実行時に，セルと対応しない未使用の基本項目を初期化するかどうか	○	○	○
CBLD_ファイル名	ファイル単位に入出力を指示するオプション	○	○	○
CBLFSYNC	ファイルクローズ時のディスク書き込み保証を適用するかどうか	○	○	○
CBLINBUFSIZE	OPEN 文のモードが INPUT 指定のバッファサイズ制御によるファイル入力時に使用するバッファサイズ	○	○	○
CBLIOMESSAGE	ファイル入出力文でのエラー情報出力機能を使用するかどうか	○	○	○
CBLISAMDL	既存の索引ファイルに対して OPEN OUTPUT を実行したとき，旧ファイルを削除後，新規に作成するかどうか	○	○	○
CBLISAMFSYNC	ISAM による索引編成ファイルに対して，ファイルクローズ時のディスク書き込み保証を適用するかどうか	○	○	○
CBLLARGEFILE	ラージファイル入出力機能の対象である実行単位中のファイルに対して，ラージファイル入出力機能を適用するかどうか	○	○	○
CBLOUTBUFSIZE	OPEN 文のモードが OUTPUT/EXTEND 指定のバッファサイズ制御によるファイル出力時に使用するバッファサイズ	○	○	○
CBLRDBDATAERR	HiRDB による索引編成ファイルで，レコード中の保証されないデータをエラーとして検出するかどうか	○	○	×
CBLRDBILWAIT	HiRDB による索引編成ファイルで，内部的に発行される SELECT に対して排他オプションを付けるかどうか	○	○	×
CBLRDBOPURGE	HiRDB による索引編成ファイルの全データを削除する際に，PURGE TABLE を使用するかどうか	○	○	×
CBLRDBROWVALCONSTRUCTOR	HiRDB による索引編成ファイルで，内部発行 SELECT で行値構成子を使用する「内部発行される SQL 文で行値構成子を使用する機能」を使用するかどうか	○	○	×
CBLSTMFSYNC	バイトストリーム入出力サービスルーチンで扱うファイルに対して，ファイルクローズ時のディスク書き込み保証を適用するかどうか	○	○	○

環境変数名	指定する内容	OS		
		AIX(32)	AIX(64)	Linux
CBLSTMLARGEFILE	バイトストリーム入出力サービスルーチンでのラージファイル入出力機能を適用するかどうか	○	○	○
CBLTEXTSUPPRESSBOM	テキスト編成ファイルでの Unicode シグニチャ出力を切り替える	○	○	○
CBLTEXTWRITESPACE	テキスト編成ファイルに対する WRITE 文または REWRITE 文で，レコード末尾からの連続する半角空白文字をファイルに書き出す	○	○	○
CBLX_外部装置名	書式印刷するときの印刷サービス名称	○	×	×
CBL_RDBCOMMIT	HiRDB による索引編成ファイルのトランザクション管理をするかどうか	○	○	×
CBL_外部装置名	ファイル入出力での入出力ファイル名	○	○	○

(凡例)

○：サポートしている

×：サポートしていない

(4) 画面（XMAP）(AIX(32)で有効)

環境変数名	指定する内容	OS		
		AIX(32)	AIX(64)	Linux
CBLPRNTID	画面機能での送信先プリンタに対する仮想端末名	○	×	×
CBLPRNT_xxx	画面機能で送信先がプリンタのときの仮想端末名	○	×	×
CBLTERMID	画面機能での送信先ディスプレイに対する仮想端末名	○	×	×
CBLTERMSHAR	複数プログラムでの仮想端末共有	○	×	×
CBLTERM_xxx	画面機能で送信先がディスプレイのときの仮想端末名	○	×	×

(凡例)

○：サポートしている

×：サポートしていない

(5) 整列併合

環境変数名	指定する内容	OS	
		AIX	Linux
CBLSORTSIZE	整列処理で使用するメモリサイズ	○	○
CBLSORTWORK	整列処理用の作業用ファイルのディレクトリ名	○	○

（凡例）

○：サポートしている

(6) 拡張機能

環境変数名	指定する内容	OS		
		AIX(32)	AIX(64)	Linux
CBLCGIERR	CGI プログラムで実行時エラーが発生した場合，エラー情報の先頭に HTTP ヘッダを付けるかどうか	○	×	×
CBLCGIINITSIZE	CBLCGIINIT サービスルーチンが受け取ったフォーム情報で，「名前」または「値」として受け取る文字列の最大長	○	×	×
CBLMTEND	マルチスレッド対応 COBOL プログラムで実行時エラーが発生したときに，abort 関数を発行して終了させたい場合，ABORT を指定する	○	○	○
CBLNO_LIBFREE	COBOL プログラムの終了処理で COBOL が動的なリンクでロードした COBOL 実行時ライブラリをアンロードするかどうか	○	○	○
CBLSQLCOMMOD	ODBC インタフェースを使用してデータベースに接続する場合のコミットモードを設定する	×	×	○
CBLSQLCURUSE	ODBC インタフェース機能を使用した場合にカーソルオプションの設定を変更する	×	×	○
CBLSQLDYNAMIC	ODBC インタフェース機能の動的 SQL で内部的に発行する ODBC API を変更する	×	×	○
CBLSQLLOGINTIMEOUT	ODBC インタフェースを使用した場合に ODBC オプションの SQL_LOGIN_TIMEOUT の値を指定する	×	×	○
CBLSQLQUERYTIMEOUT	ODBC インタフェースを使用した場合に ODBC オプションの SQL_QUERY_TIMEOUT の値を指定する	×	×	○
CBLSQLROWCOUNT	ODBC インタフェースを使用した場合に影響行数が 0 のときに SQLCODE に 100 を設定する	×	×	○
CBLSQLSUPPRESSMSG	ODBC インタフェースを使用した場合に実行時メッセージの出力を抑止するかどうか	×	×	○
CBLSQLWMSG	ODBC インタフェースを使用した場合に警告メッセージの出力を抑止するかどうか	×	×	○

（凡例）

○：サポートしている

×：サポートしていない

(7) デバッグ

環境変数名	指定する内容	OS	
		AIX	Linux
CBLABNLST	異常終了時要約情報リストの出力先	○	○
CBLCORE	シグナル発生時のコアダンプを出力するかどうか	○	○
CBLDDUMP	データ領域ダンプの出力先	○	○
CBLDMPLEVEL	データ領域ダンプリストの出力で，情報の出力レベルの指定	○	○
CBLDMPPGMN	データ領域ダンプリストを出力するとき，ダンプリストを出力するプログラム名の指定	○	○
CBLPRMCHKW	-DebugCompati オプション指定時，またはテストデバッグ時のプログラム間整合性チェックを緩和する	○	○
CBLTDEXEC※	プログラム実行時にテストデバッガを起動するかどうか	○	○
CBL_FLSRVDUMP	COBOL 入出力サービスルーチンのデバッグ情報を出力するファイル名	○	○
CBLTDDISPLAY	連動実行時，デバッガを表示するための端末の表示先の指定※	○	○
CBLEXCEPT	プログラムの実行で例外が発生したときの動作を指定する	○	○
CBLDATADUMPFILE	CBLDATADUMP サービスルーチンによるデータ領域ダンプの出力先	○	○

（凡例）

○：サポートしている

注※

詳細は，マニュアル「COBOL2002　使用の手引　操作編」を参照してください。

(8) オブジェクト指向

環境変数名	指定する内容	OS	
		AIX	Linux
CBLGCINTERVAL	前回のガーベジコレクションを終了してから，次のガーベジコレクションを開始するまでのメモリ使用量（ガーベジコレクションの実行間隔）	○	○
CBLGCSTART	ガーベジコレクタの開始条件となる，インスタンスオブジェクトの生成によるメモリ使用量の累積値	○	○

（凡例）

○：サポートしている

35.3.3　一般

(1)　CBLENVMAX

環境変数値の書き出しを行う DISPLAY 文の実行回数を 8 けた以内の符号なし整数で指定します。

詳細は，「10.4　環境変数へのアクセス」を参照してください。

(2)　CBLEXVALUE

EXTERNAL 句指定のあるデータ項目に NULL を設定して，領域を初期化します。この環境変数の指定は，作業場所節で定義しているデータ項目に対して有効となります。

詳細は，「4.2.2　外部属性（EXTERNAL 句）」を参照してください。

(3)　CBLLANG

動作する言語環境（文字コード）を指定します。

詳細は，「27.4.2　実行」を参照してください。

(4)　CBLLPATH

動的なリンク時，共用ライブラリの検索ディレクトリを指定します。

詳細は，「18.5　静的なリンクと動的なリンク」を参照してください。

(5)　CBLLSLIB

動的なリンク時，検索したい共用ライブラリ名を指定します。

詳細は，「18.5　静的なリンクと動的なリンク」を参照してください。

(6)　CBLLTAG（AIX で有効）

この環境変数は，動的なリンクの動作を指示するオプションを指定します。

複数のオプションを指定する場合は，各オプションをコロン(:)で区切って指定します。同じオプションを複数指定した場合は，後に指定した値が有効となります。

この環境変数に指定するオプション文字列の長さは，1,024 バイト以内にしてください。1,024 バイトを超える文字列を指定した場合，メッセージが出力され，この環境変数の指定は無効となります。

（例）

```
CBLLTAG=NOSO:NOARMBR
```

詳細は，「18.6.2　共用ライブラリに含まれるプログラムの呼び出し方法」の「(2)　動的なリンク」を参照してください。

次に指定できるオプションを示します。

SO／NOSO

動的なリンクで，検索対象とするライブラリの拡張子を選択できます。SO を指定した場合は，拡張子「.so」の共用ライブラリを検索対象とします。NOSO を指定した場合は，拡張子「.so」の共用ライブラリを検索対象としません。このオプションの標準値は，SO です。

ARMBR／NOARMBR／ARMBRLSYM

動的なリンクで，検索対象とするライブラリ種別を選択できます。ARMBR と ARMBRLSYM を指定した場合は，アーカイブ化された共用ライブラリを検索対象とします。NOARMBR を指定した場合は，アーカイブ化された共用ライブラリを検索対象としません。このオプションの標準値は，ARMBR です。

ARMBR と ARMBRLSYM の違いを次に示します。

- ARMBR：共用ライブラリ中のすべてのシンボル情報から検索します。
- ARMBRLSYM：共用ライブラリ中のエクスポートされたシンボル情報だけから検索します。

(7)　CBLPGMSEARCHTRC

次のトレース情報を出力するファイル名を指定します。

- 動的なリンクによるプログラム呼び出し時の共用ライブラリの検索情報
- プレロード時の共用ライブラリのロード情報

詳細は，「18.6.4　動的なリンクのプログラム検索トレース機能」を参照してください。

(8)　CBLPGMSEARCHTRC_SIZE

プログラム検索トレースファイル名を切り替えるサイズを指定します。

詳細は，「18.6.4　動的なリンクのプログラム検索トレース機能」を参照してください。

(9)　CBLPRELOAD

実行単位内で最初に実行される COBOL プログラムの起動時に，動的なリンクで検索対象となる共用ライブラリ名称を記述したプレロードリストのファイル名を指定します。

詳細は，「18.6.3　動的なリンクのプレロード機能」を参照してください。

(10)　CBLUNIENDIAN

用途が NATIONAL の項目に対する Unicode のバイトオーダを指定します。

詳細は，「27.4.2　実行」を参照してください。

(11)　CBLUPSI

外部スイッチの状態を，８個の０と１で指定します。

詳細は，「16.2.4　外部スイッチ」の「(2)　外部スイッチの設定方法」を参照してください。

(12)　CBL_SYSERR

実行時エラーメッセージの出力先ファイル名を指定します。ファイル名は絶対パス名で指定します。拡張子の種類や拡張子を付けるかどうかは任意です。CBL_SYSERR の指定形式は次のようになります。

形式

```
CBL_SYSERR=ファイル名〔+〕
```

＋を指定した場合：追加モードで出力されます。

CBL_SYSERR を指定した場合の実行時エラーメッセージの出力先を次に示します。

ファイル名の指定	データの出力先
CBL_SYSERR=stdin〔+〕	ファイル名「stdin」
CBL_SYSERR=stdout〔+〕	標準出力※
CBL_SYSERR=stderr〔+〕	標準エラー出力※
CBL_SYSERR= 上記以外のファイル名〔+〕	環境変数 CBL_SYSERR で指定した名称のファイル

注

標準出力（stdout），および標準エラー出力（stderr）を指定する場合は，英小文字で指定してください。「STDOUT」のように英大文字で指定した場合，物理ファイル名として扱われます。

注※

追加モードは無視されます。

35.3.4　少量データ

(1)　CBLDATE

システムから受け取る西暦年月日を，yyyymmdd（yyyy は西暦の年，mm は月，dd は日）の８けたの形式で指定します。

詳細は，「10.2.3　日付や時刻を取得する ACCEPT 文」の「(2)　ACCEPT 文で取得する日付の変更」を参照してください。

(2) CBLDAY

ACCEPT 文でシステムから受け取る西暦年，および通年日を yyyyddd（yyyy は西暦の年，ddd は通年日）の 7 けたの形式で設定します。

詳細は，「10.2.3 日付や時刻を取得する ACCEPT 文」の「(2) ACCEPT 文で取得する日付の変更」を参照してください。

(3) CBLSTDIOLVL

DISPLAY 文によってデータを標準出力（stdout），または標準エラー出力（stderr）に出力する場合，COBOL 実行時ライブラリが使用するシステム入出力関数のレベルを指定します。

形式

```
CBLSTDIOLVL=OUTLOW:ERRLOW
```

規則

- 複数のオプションを指定する場合は，各オプションをコロン（:）で区切って指定してください。また，同じオプションを複数指定した場合，後に設定した値が有効となります。
- 環境変数 CBLSTDIOLVL に指定する文字列の長さは，1,024 バイト以内でなければなりません。1,024 バイトを超える文字列を指定した場合は，メッセージが出力され，環境変数 CBLSTDIOLVL の指定は無効となります。

環境変数 CBLSTDIOLVL は，標準出力または標準エラー出力を高水準システム入出力関数の使用が制限される出力先に変更しているときに指定します。

環境変数 CBLSTDIOLVL に ERRLOW または OUTLOW を指定した場合，DISPLAY 文でエラーが発生してもプログラムの実行を継続します。また，エラー発生時の DISPLAY 文の出力データを COBOL ログファイルに出力します。

標準出力または標準エラー出力の出力先を変更しない場合は，環境変数 CBLSTDIOLVL を指定する必要はありません。環境変数 CBLSTDIOLVL の指定条件と機能の詳細については，「10.2.5 システム入出力関数レベルの指定」を参照してください。

設定できるオプションを次に示します。次に示すオプション以外は無効となります。

(a) OUTLOW／OUTHIGH

OUTLOW を指定した場合，出力先が標準出力（stdout）の DISPLAY 文実行時，低水準システム関数を使用してデータを出力します。OUTHIGH を指定した場合，高水準システム関数を使用します。標準値は OUTHIGH になります。

(b)　ERRLOW／ERRHIGH

ERRLOW を指定した場合，出力先が標準エラー出力（stderr）の DISPLAY 文実行時，低水準システム関数を使用してデータを出力します。ERRHIGH を指定した場合，高水準システム関数を使用します。標準値は ERRHIGH になります。

(4)　CBL_STOPNOADV

環境変数 CBL_STOPNOADV に YES を指定すると，STOP 定数文を実行したとき，実行時メッセージ ID と，定数の直後の改行文字が出力されません。

詳細は，「10.2.6　STOP 文」を参照してください。

(5)　CBL_SYSIN

FROM SYSIN 指定時の ACCEPT 文で入力ファイル名を指定します。

詳細は，「10.2.2　外部からのデータを入力する ACCEPT 文」の「(1)　標準転記による ACCEPT 文」を参照してください。

(6)　CBL_SYSOUT

UPON SYSOUT 指定時の DISPLAY 文で出力ファイル名を指定します。

詳細は，「10.2.4　DISPLAY 文によるデータの出力」を参照してください。

(7)　CBL_SYSPUNCH

UPON SYSPUNCH 指定時の DISPLAY 文で出力ファイル名を指定します。

詳細は，「10.2.4　DISPLAY 文によるデータの出力」を参照してください。

(8)　CBL_SYSSTD

FROM SYSSTD 指定時の ACCEPT 文で入力ファイル名を指定します。

詳細は，「10.2.2　外部からのデータを入力する ACCEPT 文」の「(1)　標準転記による ACCEPT 文」を参照してください。

35.3.5　ファイル

(1)　CBLCSVCHAR

-NumCsv オプションを指定して CSV 編成ファイルの入出力をする場合，無視したい文字列を指定します。

規則

- 環境変数 CBLCSVCHAR に複数の文字列を指定する場合は，各文字列をセミコロン（;）で区切ります。
- 環境変数 CBLCSVCHAR に指定する文字列の長さは，1,024 バイト以内でなければなりません。1,024 バイトを超える文字列を指定した場合は，メッセージが出力され，環境変数 CBLCSVCHAR の指定は無効となります。

詳細は，「6.8.5　セルデータを数値として入出力する機能」の「(3)　数値として入力するとき，不要な文字列を無視する機能」を参照してください。

(2)　CBLCSVINIT

CSV ファイルの READ 文実行時に，セルと対応しない未使用の基本項目を初期化する場合に指定します。

詳細は，「6.8.7　入力時の未使用項目の初期化機能」を参照してください。

(3)　CBLD_ファイル名

入出力動作を指示する下記のオプションをファイル単位に指定します。

規則

- 複数のオプションを指定するときは各オプションをコロン（：）で区切ります。
- 環境変数 CBLD_ファイル名は，OPEN 文を実行するごとに環境変数の値が参照されます。
- ここで指定するファイル名は SELECT 句で指定したファイル名に対応します。ただし，ファイル名内のハイフン（-）は下線（_）に置き換えて指定します。
- 背反するオプションを同時に指定した場合，あとから指定したオプションが有効となります。
- 環境変数 CBLD_ファイル名に指定する文字列の長さは，1,024 バイト以内でなければなりません。1,024 バイトを超える文字列を指定した場合，メッセージが出力され，これらの環境変数の指定は無効となります。
- 同じファイルに対して環境変数 CBLD_ファイル名と環境変数 CBL_ファイル名の両方を指定した場合，環境変数 CBLD_ファイル名に指定した内容だけが有効となり，環境変数 CBL_ファイル名に指定した値は無効となります。なお，環境変数 CBLD_ファイル名に NULL（指定値なし）を指定した場合でも，環境変数 CBL_ファイル名に指定した内容は無効となります。

設定例

（例 1）

（COBOL での記述例）

```
SELECT A-FILE ASSIGN TO SYS000.
```

（環境変数の設定例）

```
CBLD_A_FILE=ISAMDL:NOISAMPREV
```

（例 2）

背反するオプションを同時に指定した場合，あとから指定したオプションが有効となります。

```
CBLD_A_FILE=ISAMDL:NOISAMDL
```

上記の指定をした場合，最初の"ISAMDL"は無効となり，あとに設定した"NOISAMDL"が有効となります。

設定できるオプションを次に示します。次に示すオプション以外は無効となります。

(a) ISAMDL／NOISAMDL

ISAMDL は，索引編成ファイルで OPEN モードが OUTPUT の場合，既存のファイルを削除して再生成します。NOISAMDL は，ファイルの削除はしません。省略時は NOISAMDL が仮定されます。

(b) ISAMPREV／NOISAMPREV

ISAMPREV は，索引編成ファイルで START 文の指定が LESS，LESS THAN OR EQUAL，および LAST の場合，あとに続く NEXT 指定の READ 文でキーの降順に呼び出します。NOISAMPREV は，START 文の指定に関係なく，NEXT 指定の READ 文はキーの昇順に呼び出します。省略時は ISAMPREV が仮定されます。

(c) SAMAADV／NOSAMAADV

SAMAADV は，ADVANCING 指定を書かない WRITE 文に AFTER ADVANCING 1 LINE を仮定します。NOSAMAADV はこれを仮定しません。省略時は NOSAMAADV が仮定されます。

SAMAADV と SAMBADV を同時に指定した場合は，あとに設定した方が有効となります。

(d) CSVQUOTE／NOCSVQUOTE

CSVQUOTE は，CSV 編成ファイルの WRITE 文でセルの内容を出力するとき，データを引用符（ " ）で囲みます。NOCSVQUOTE は，引用符を付けません。省略時は CSVQUOTE が仮定されます。

NOCSVQUOTE を指定した場合の詳細は，「6.8　CSV 編成ファイル（表計算プログラムファイル）」を参照してください。

(e) CSVWRITESPACE／NOCSVWRITESPACE

CSVWRITESPACE は，CSV 編成ファイルの WRITE 文で，出力するレコードの基本項目のデータの最後の空白文字を出力します。NOCSVWRITESPACE は，データの最後の空白文字を出力しません。なお，省略時は NOCSVWRITESPACE が仮定されます。

CSVWRITESPACE を指定した場合の詳細は，「6.8.8　データの後の空白文字を出力する機能」を参照してください。

(f)　CSVTABSEPARATED／NOCSVTABSEPARATED

CSVTABSEPARATED は，CSV ファイル入出力機能でセルデータをタブ文字区切りで入出力します。NOCSVTABSEPARATED はセルデータにコンマ（,）区切りで入出力します。なお，省略時は NOCSVTABSEPARATED が仮定されます。

CSVTABSEPARATED を指定した場合の詳細は，「6.8.9　セルデータをタブ文字区切りで入出力する機能」を参照してください。

(g)　FSYNC／NOFSYNC／WDISK

強制的にディスク書き込みをするかしないかを指定します。

FSYNC を指定した場合，プログラムの終了時，または CLOSE 文の実行終了時に，COBOL プログラムの環境部のファイル記述項で指定したファイル名のファイルに対して，強制的なディスク書き込みを適用します。

WDISK を指定した場合，WRITE 文，REWRITE 文，または DELETE 文の実行終了時に，COBOL プログラムの環境部のファイル記述項で指定したファイル名のファイルに対して，強制的なディスク書き込みを適用します。

NOFSYNC を指定した場合は，ディスクへの書き込み保証が適用されません。

FSYNC，NOFSYNC，WDISK のどれかを同時に指定した場合は，あとに指定した方が有効となります。また，FSYNC，NOFSYNC，WDISK のどれも指定しない場合は，環境変数 CBLFSYNC，および環境変数 CBLISAMFSYNC の指定に従います。

ファイルのディスク書き込み保証機能の詳細については，「14.1　ファイルのディスク書き込み保証」を参照してください。

(h)　SAMENDIO／NOSAMENDIO

SAMENDIO は，固定長形式の順ファイルでファイルサイズがレコード長の整数倍ではなく，ファイルの最終レコードが定義レコード長よりも短い場合でも入出力できるようにします。

NOSAMENDIO では，ファイルの最終レコードが定義レコード長より短い場合，エラーになります。標準値は，NOSAMENDIO になります。

詳細は，「6.11.1　ファイルサイズがレコード長の整数倍でない固定長形式の順ファイルの入出力」を参照してください。

(i)　LARGEFILE／NOLARGEFILE

LARGEFILE は，ラージファイル（ファイルサイズが 2GB より大きいファイル）に対しても入出力ができるようにします。

NOLARGEFILE では，ラージファイルに対してアクセスした場合，エラーになります。標準値は，NOLARGEFILE になります。詳細は，「6.10　ラージファイル入出力機能」を参照してください。

(j)　NOINBUFSIZE／NOOUTBUFSIZE（Linux で有効）

NOINBUFSIZE は，指定されたファイルのバッファサイズ制御による入力時に，環境変数 CBLINBUFSIZE で設定された値を無効にします。NOOUTBUFSIZE は，指定されたファイルのバッファサイズ制御による出力時に，環境変数 CBLOUTBUFSIZE で設定された値を無効にします。詳細は「6.11.2　ファイルバッファサイズ指定機能」を参照してください。

(k)　TEXTWRITESPACE／NOTEXTWRITESPACE

テキスト編成ファイルに対する WRITE 文または REWRITE 文で，レコード末尾からの連続する半角空白文字をファイルに書き出したい場合，TEXTWRITESPACE を指定します。テキスト編成ファイルに対する WRITE 文または REWRITE 文で，レコード末尾からの連続する半角空白文字を削除したい場合は，NOTEXTWRITESPACE を指定します。詳細については，「6.7.5　レコード末尾の空白文字を出力する機能」を参照してください。

(l)　TEXTSUPPRESSBOM／NOTEXTSUPPRESSBOM

テキスト編成ファイルに Unicode シグニチャを出力しない場合，TEXTSUPPRESSBOM を指定します。Unicode シグニチャを出力する場合，NOTEXTSUPPRESSBOM を指定します。

詳細については，「27.5.2　入出力機能」の「(1)　テキスト編成ファイル」を参照してください。

(m)　RDBOPURGE／NORDBOPURGE（AIX で有効）

HiRDB による索引編成ファイルで，出力モードでファイルを開く場合，すでにファイル内に存在するレコードを高速で削除するとき，RDBOPURGE を指定してください。レコードを高速で削除しない場合，NORDBOPURGE を指定してください。

詳細については，「6.9.5　HiRDB による索引編成ファイル固有の機能と相違点」の「(5)　その他の拡張機能」を参照してください。

(4)　CBLFSYNC

プロセス内のすべてのファイルに対して，クローズ時のディスクへの書き込み保証を適用したいときに YES を指定します。YES 以外の値を指定したときは，無効となります。

詳細は，「14.1.2　ファイルクローズ時のディスク書き込み保証の指定方法」および「13.6　COBOL 入出力サービスルーチンでのディスク書き込み保証」を参照してください。

(5)　CBLINBUFSIZE

OPEN 文のモードが INPUT 指定のバッファサイズ制御によるファイル入力時に，使用するバッファサイズを 2,000,000,000（約 2GB）までの数値で指定します。

詳細は，「6.11　ファイル入出力拡張機能」を参照してください。

(6) CBLIOMESSAGE

ファイル入出力文でのエラー情報出力機能を使用したい場合に，この環境変数を指定します。指定可能な値を次に示します。

(a) STATUS90

ファイル管理記述項に FILE STATUS 句の指定がある場合に，入出力文の実行で入出力状態値が 90 となるエラーが発生したとき，該当する実行時メッセージを出力してプログラムの実行を継続します。ただし，出力するメッセージのレベルは I（お知らせ）となります。

この環境変数については，「6.3.4　ファイル入出力文でのエラー情報出力機能」を参照してください。

(7) CBLISAMDL

既存の索引ファイルに対して OPEN OUTPUT を実行した場合，旧ファイルを削除後，ファイルを新しく作成したいときに YES を指定します。指定しなかったときや，YES 以外の文字を指定したときは，NO が仮定されます。このとき，既存のファイルに対しての追加書きとなります。

(8) CBLISAMFSYNC

プロセス内のすべての ISAM による索引編成ファイルに対して，クローズ時のディスクへの書き込み保証を適用したいときに YES を指定します。YES 以外の値を指定したときは，無効となります。ファイルのディスク書き込み保証については，「14.　ファイルのディスク書き込み保証」を参照してください。

(9) CBLLARGEFILE

ラージファイル入出力機能の対象である実行単位中のすべてのファイルに対して，ラージファイル入出力機能を適用するときに YES を指定します。詳細は「6.10　ラージファイル入出力機能」を参照してください。

(10) CBLOUTBUFSIZE

OPEN 文のモードが OUTPUT／EXTEND 指定のバッファサイズ制御によるファイル出力時に，使用するバッファサイズを 2,000,000,000（約 2GB）までの数値で指定します。

詳細は，「6.11　ファイル入出力拡張機能」を参照してください。

(11) CBLRDBDATAERR（AIX で有効）

HiRDB による索引編成ファイルで，レコード中に保証されないデータが含まれていないかをチェックする場合に YES を指定します。

詳細は「6.9.5(5)(b)　データチェック機能」を参照してください。

(12) CBLRDBILWAIT（AIX で有効）

RDB ファイル入出力機能を使用する場合，INPUT モードで開いたファイルに対し，内部的に発行する SELECT に「WITHOUT LOCK WAIT」の排他オプションを付加するときに YES を指定します。

詳細は「6.9.5(4)　HiRDB による索引編成ファイル固有のファイル共用」を参照してください。

(13) CBLRDBOPURGE（AIX で有効）

HiRDB による索引編成ファイルに対して出力モードでファイルを開く場合，すでにファイル内に存在するレコードを高速で削除したいときに YES を指定します。

詳細は「6.9.5(5)(a)　出力ファイルの高速オープン機能」を参照してください。

(14) CBLRDBROWVALCONSTRUCTOR（AIX で有効）

HiRDB による索引編成ファイルで，内部発行 SELECT で行値構成子を使用する，内部発行される SQL 文で行値構成子を使用する機能を有効にするときに，YES を指定します。

詳細は，「6.9.5　HiRDB による索引編成ファイル固有の機能と相違点」の「(5)　その他の拡張機能」にある「(c)　内部発行される SQL 文で行値構成子を使用する機能」を参照してください。

(15) CBLSTMFSYNC

プロセス内のすべてのバイトストリーム入出力サービスルーチンで扱うファイルに対して，クローズ時のディスクへの書き込み保証を適用したいときに YES を指定します。YES 以外の値を指定したときは，無効となります。

詳細は，「15.5　バイトストリーム入出力サービスルーチンでのディスク書き込み保証機能」を参照してください。

(16) CBLSTMLARGEFILE

バイトストリーム入出力サービスルーチンでのラージファイル入出力機能を適用したいときに，YES を指定します。省略時および YES 以外の値を指定した場合，バイトストリーム入出力サービスルーチンでのラージファイル入出力機能は無効となります。

詳細は，「15.4　バイトストリーム入出力サービスルーチンでのラージファイル入出力機能」を参照してください。

(17) CBLTEXTSUPPRESSBOM

テキスト編成ファイルに Unicode シグニチャを出力しない場合に指定します。

詳細は，「27.5.2　入出力機能」の「(1)　テキスト編成ファイル」を参照してください。

(18)　CBLTEXTWRITESPACE

テキスト編成ファイルに対する WRITE 文または REWRITE 文で，レコード末尾からの連続する半角空白文字をファイルに書き出したい場合に指定します。

詳細は，「6.7.5　レコード末尾の空白文字を出力する機能」を参照してください。

(19)　CBLX_外部装置名（AIX(32)で有効）

書式印刷機能を使用したプリンタ出力するときの印刷サービス名称を指定します。

詳細は，「8.4　XMAP3 による印刷（AIX(32)で有効）」を参照してください。

(20)　CBL_RDBCOMMIT（AIX で有効）

HiRDB による索引編成ファイルに対して，COMMIT 文，ROLLBACK 文によって RDB アクセスのトランザクション管理を行う場合に YES，AUTO，または MANUAL を指定します。

詳細は「6.9.5(1)(a)　環境変数 CBL_RDBCOMMIT」を参照してください。

(21)　CBL_外部装置名

ファイル入出力での入出力ファイル名を指定します。

詳細は，「6.2.2　環境変数指定」を参照してください。

35.3.6　画面（XMAP）（AIX(32)で有効）

(1)　CBLPRNTID

通信節による画面機能で SYMBOLIC TERMINAL 句の指定を省略した場合，送信先プリンタの仮想端末名を指定します。

詳細は，「12.1.4　プリンタに対する帳票出力」の「(4)　送信先の設定方法」を参照してください。

(2)　CBLPRNT_xxx

通信節による画面機能で SYMBOLIC TERMINAL 句を指定した場合，送信先がプリンタのときの仮想端末名を指定します。

詳細は，「12.1.4　プリンタに対する帳票出力」の「(4)　送信先の設定方法」を参照してください。

(3)　CBLTERMID

通信節による画面機能で SYMBOLIC TERMINAL 句の指定を省略した場合，送受信先がディスプレイのときの仮想端末名を指定します。

詳細は，「12.1.2　画面に対する入出力」の「(7)　送受信先の設定方法」を参照してください。

(4)　CBLTERMSHAR

複数プログラム間で一つの仮想端末を共有する場合，YES を指定します。YES 以外の値を指定したときは，無効となります。

詳細は，「12.1.3　仮想端末の共用」を参照してください。

(5)　CBLTERM_xxx

通信節による画面機能で SYMBOLIC TERMINAL 句を指定した場合，送受信先がディスプレイのときの仮想端末名を指定します。

詳細は，「12.1.2　画面に対する入出力」の「(7)　送受信先の設定方法」を参照してください。

35.3.7　整列併合

(1)　CBLSORTSIZE

整列処理で外部ファイルを使用するとき，整列機能が確保するメモリサイズをキロバイト単位で指定します。8 けたの符号なし整数で設定します。

詳細には，「11.3.1　整列処理のメモリサイズ」を参照してください。

(2)　CBLSORTWORK

整列処理するときの作業用ファイルのパスプレフィクスを指定します。

詳細は，「11.2.2　整列作業用ファイル」を参照してください。

35.3.8　拡張機能

(1)　CBLCGIERR（AIX(32)で有効）

CGI プログラムで実行時エラーが発生したとき，メッセージの先頭に CGI ヘッダ（HTTP ヘッダの MIME フォーマットを表す"Content-type: text/plain¥n¥n"）を付けて出力したい場合に指定します。

詳細は，「25.9　実行時エラーメッセージの取得方法」を参照してください。

(2)　CBLCGIINITSIZE（AIX(32)で有効）

CBLCGIINIT サービスルーチンが受け取るフォーム情報で，「名前」「値」のデコードされていない状態での最大文字列長をキロバイト単位で指定します。指定できる値の範囲は，1～2,000,000 です。この範囲外の値を指定した場合，または環境変数 CBLCGIINITSIZE を指定しなかった場合は，64 が仮定されます。

詳細は，「25.7.2　サービスルーチンの説明」の「(7)　CBLCGIINIT」を参照してください。

(3)　CBLMTEND

マルチスレッド対応 COBOL プログラムで，実行時エラーが発生したときのスレッドの終了方法を変更できます。環境変数 CBLMTEND を指定しない場合，COBOL は pthread_exit 関数を発行してそのスレッドを終了します。

例えば，マルチスレッド対応 COBOL プログラムで実行時エラーが発生し，スレッドの終了ではなくプロセスが終了しなければならない製品と連携している場合は，環境変数 CBLMTEND = ABORT を指定することで，実行時エラーが発生したときのスレッドの終了方法を変更できます。

また，ABORT を指定した場合に，マルチスレッド対応 COBOL プログラムで COBOL 実行時エラーが発生したとき，または-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange オプションのどれかを指定したプログラムが異常終了したときに，abort 関数を発行します。

注意事項

- STOP RUN 文実行時は，環境変数 CBLMTEND の指定がないときと同様に，pthread_exit 関数を発行して，そのスレッドを終了させます。
- abort 関数が発行された場合，その abort シグナルを補足するプログラムがないかぎり，システムがプロセスを終了するため，ほかの実行中のスレッドも強制的に終了します。
- 環境変数 CBLMTEND は，スレッド単位での指定はできません。プログラム実行の前に指定してください。

(4)　CBLNO_LIBFREE

環境変数 CBLNO_LIBFREE に EXIT を指定した場合，COBOL プログラムの終了処理で COBOL が動的なリンクでロードした COBOL 実行時ライブラリをアンロードしません。環境変数 CBLNO_LIBFREE を省略した場合，または EXIT 以外が指定された場合は適用されません。詳細については，「18.6　共用ライブラリに含まれるプログラムの呼び出しと共用ライブラリのアンロード」を参照してください。

(5) CBLSQLCOMMOD（Linux で有効）

ODBC インタフェースを使って CONNECT 文でデータベースに接続するとき，コミットモードを設定するかどうかを設定します。CBLSQLCOMMOD に設定する値によって，CONNECT 文でデータベースに接続するときの動作は異なります。

形式

```
CBLSQLCOMMOD= {DEFAULT | AUTO | MANUAL}
```

DEFAULT

コミットモードが設定されません。デフォルトモード（自動コミットモード）で接続します。

AUTO

自動コミットモードが設定されて接続します。

MANUAL

手動コミットモードが設定されて接続します。

なお，この環境変数の設定を指定しなかったときには，MANUAL が仮定されます。

(6) CBLSQLCURUSE（Linux で有効）

ODBC インタフェース機能を使用した場合にカーソルオプションの設定を変更します。

形式

```
CBLSQLCURUSE=DYNAMIC
```

DYNAMIC

ODBC ドライバのスクロール機能の動的カーソルを使用するように設定します。この環境変数の指定がない場合は，ODBC カーソルライブラリの静的カーソルを使用するように ODBC オプションを設定します。DYNAMIC 以外の値を指定した場合の結果は，保証しません。

詳細は，「23.2.8　カーソルオプションの設定」を参照してください。

(7) CBLSQLDYNAMIC（Linux で有効）

ODBC インタフェース機能の動的 SQL で内部的に発行する ODBC API を変更したい場合に指定します。

詳細は，「23.2.10　動的 SQL の ODBC API 関数発行の変更」を参照してください。

(8) CBLSQLLOGINTIMEOUT（Linux で有効）

ODBC インタフェースを使って ODBC オプションの SQL_LOGIN_TIMEOUT に設定するタイムアウト秒数を指定します。

詳細は，「23.2.7　タイムアウト秒数の設定」を参照してください。

(9) CBLSQLQUERYTIMEOUT（Linux で有効）

ODBC インタフェースを使って ODBC オプションの SQL_QUERY_TIMEOUT に設定するタイムアウト秒数を指定します。

詳細は，「23.2.7 タイムアウト秒数の設定」を参照してください。

(10) CBLSQLROWCOUNT（Linux で有効）

ODBC インタフェースを使って埋め込み SQL 文の DELETE 文，INSERT 文，または UPDATE 文実行によって影響を受けた行数が 0 行の場合，SQLCODE 変数に 100 を設定したいときに YES を指定します。

詳細は，「23.2.3 SQL 文のエラー処理」を参照してください。

(11) CBLSQLSUPPRESSMSG（Linux で有効）

ODBC インタフェースを使って埋め込み SQL 文の実行時，エラーが発生したときに出力される KCCC8002R-S のメッセージ出力を抑止したい場合，8002 を指定します。

詳細は，「23.2.3 SQL 文のエラー処理」を参照してください。

(12) CBLSQLWMSG（Linux で有効）

ODBC インタフェースを使用してデータベースにアクセスする場合，警告メッセージの出力を抑止したいときに YES を指定します。指定しなかったときや，YES 以外の値を指定したときは，警告メッセージを出力します。

詳細は，「23.2.3 SQL 文のエラー処理」を参照してください。

35.3.9 デバッグ

(1) CBLABNLST

異常終了時要約情報リストの出力先を指定します。

詳細は，「36.2 異常終了時要約情報リスト」を参照してください。

(2) CBLCORE

シグナル発生時にコアダンプを出力するかどうかを指定します。詳細については，「36.8 コアダンプの出力」を参照してください。

(3) CBLDDUMP

異常終了時のデータ領域ダンプの出力先を指定します。詳細については，「36.3　データ領域ダンプリスト」を参照してください。

(4) CBLDMPLEVEL

データ領域ダンプリスト出力で，情報の出力レベルを指定します。詳細については，「36.4　データ領域ダンプリスト出力情報の選択」を参照してください。

(5) CBLDMPPGMN

データ領域ダンプリストを出力するとき，ダンプリストを出力するプログラム名を指定します。詳細については，「36.4　データ領域ダンプリスト出力情報の選択」を参照してください。

(6) CBLDATADUMPFILE

CBLDATADUMP サービスルーチンによるデータ領域ダンプの出力先を指定します。詳細は「36.3.2 データ領域ダンプリストの出力先」を参照してください。

(7) CBLEXCEPT

この環境変数に NOSIGNAL を指定すると，-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange のどれかを指定したプログラムの実行中に，例外が発生した場合，COBOL での例外検出はしません。詳細は，「36.9　シグナル」を参照してください。

(8) CBLPRMCHKW

形式

```
CBLPRMCHKW= {YES | NOCHK}
```

YES

テストデバッグ中のプログラム間整合性エラーを警告化します。

NOCHK

-DebugCompati オプション指定時でもプログラム間整合性チェックはしません。

プログラム間の引数および返却項目に関するエラーがあることがわかっているが，異常終了させないでテストデバッグまたはプログラムを実行したいときに指定する環境変数です。

詳細は，「36.5.2　整合性チェックの警告エラー出力」を参照してください。

(9) CBLTDEXEC

プログラムの開始と同時に，次のテストデバッガの機能を連動させるときに指定します。この環境変数は，デバッグの対象となるプログラムを起動する前に指定しておく必要があります。

- ラインモードによるテストデバッガ
- カバレージ採取
- カウント

プログラムからの連動実行の詳細は，マニュアル「COBOL2002　使用の手引　操作編」を参照してください。

形式

```
CBLTDEXEC= {TL 引数 | CV 引数 | CN 引数}
```

TL 引数
　ラインモードによるテストデバッグを連動実行します。

CV 引数
　カバレージ採取を連動実行します。

CN 引数
　カウントを連動実行します。

注意事項

　引数に空白やタブを含むファイル名やディレクトリ名は指定してはなりません。

(10) CBL_FLSRVDUMP

COBOL 入出力サービスルーチンのデバッグ情報を出力するファイル名を指定します。詳細は，「13.5.2 インタフェース領域のダンプ出力」を参照してください。

(11) CBLTDDISPLAY

連動実行時，テストデバッガを表示するための端末の表示先を変更するときに指定します。詳細は，マニュアル「COBOL2002　使用の手引　操作編」を参照してください。

35.3.10 オブジェクト指向

(1) CBLGCINTERVAL

前回のガーベジコレクションの終了時から，インスタンスオブジェクトの生成によってメモリ使用量がどれだけ増加するとガーベジコレクションを開始するかを指定します。

規則

- 指定できるバイト数の値は，0～2,147,483,647 です。この値以外が指定されたときは，65,535 バイトが仮定されます。
- 環境変数 CBLGCINTERVAL を指定しなかった場合は，65,535 バイトが仮定されます。
- 指定できる値のけた数は，10 けた以内です。10 けたを超えているときは，65,535 バイトが仮定されます。

指定例

ガーベジコレクションの実行間隔を 32,768 バイトにする例を次に示します。

```
CBLGCINTERVAL=32768
export CBLGCINTERVAL
```

注意事項

この環境変数の指定値を大きく設定した場合，メモリ資源を多く使用しますが，実行性能は向上する傾向にあります。指定値を小さく設定した場合，実行性能は向上しませんがメモリ資源を節約できます。アプリケーションの性質や環境を考慮に入れて指定してください。

(2) CBLGCSTART

ガーベジコレクタの開始条件である，インスタンスオブジェクトの生成によるメモリ使用量の累積値を指定します。メモリ使用量の累積が，指定された値以上になったときガーベジコレクションが実行されます。

規則

- 指定できるバイト数の値は，0～2,147,483,647 です。この値以外が指定されているときは，524,288 バイトが仮定されます。
- 環境変数 CBLGCSTART を指定しなかった場合は，524,288 バイトが仮定されます。
- 指定できる値のけた数は，10 けた以内です。10 けたを超えているときは，524,288 バイトが仮定されます。

指定例

ガーベジコレクションの開始条件のメモリ使用量を 65,536 バイトにする例を次に示します。

```
CBLGCSTART=65536
export CBLGCSTART
```

注意事項

この環境変数の指定値を大きく設定した場合，メモリ資源を多く使用しますが，実行性能は向上する傾向にあります。指定値を小さく設定した場合，実行性能は向上しませんがメモリ資源を節約できます。アプリケーションの性質や環境を考慮に入れて指定してください。

36 アプリケーションデバッグ機能

この章では，プログラムの実行時にデバッグ用のリスト出力や各種のチェックをする機能について説明します。

36.1　デバッグ機能の種類と概要

アプリケーションデバッグ機能とは，プログラムの実行時に各種のエラーチェックをしたり，異常終了時やエラー発生時にデバッグに役立つリストを出力したりする機能です。

アプリケーションデバッグ機能を使うためには，デバッグ用のコンパイラオプションの指定が必要です。アプリケーションデバッグの各機能と指定するコンパイラオプションとの対応を次に示します。

表 36–1　アプリケーションデバッグ機能と指定するコンパイラオプション

機能	指定するコンパイラオプション
異常終了時要約情報リストの出力	-DebugInf，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange のどれか（ただし，リスト中にトレースバック情報もあわせて出力するためには-DebugInf,Trace オプションの指定が必要）
データ領域ダンプリストの出力	-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange のどれか
プログラム間整合性チェック	-DebugCompati
添字，部分参照範囲外チェック	-DebugCompati
データ例外検出機能	-DebugData
テストデバッグ機能	-TDInf
カバレージ機能	-CVInf
添字の繰り返し回数の範囲外チェック	-DebugRange
デバッグ行の利用	-DebugLine

注

-TDInf オプションを指定したプログラムをテストデバッガでデバッグするとき，-DebugCompati オプションの指定有無に関係なく，プログラム間の引数および返却項目に関するエラーをチェックします。

また，-TDInf オプションはテストデバッガのためにゼロによる除算チェックなど，最適化に影響を与えるオブジェクトコードを生成します。これによって，-TDInf オプションを指定することで，異常終了時要約情報リストなどの行番号／欄の情報が変化する場合があります。行番号／欄の情報を常に正しく保つためには，-Optimize,0 オプションを指定してください。

36.2　異常終了時要約情報リスト

COBOL2002 では，プログラムが異常終了した場合に原因を究明するための情報を，異常終了時要約情報リストとして出力できます。

異常終了時要約情報リストを出力するには，COBOL プログラムのコンパイル時に-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange のどれかのオプションを指定してください。

ただし，COBOL プログラム実行時の初期処理，または終了処理中にエラーが発生した場合は，異常終了時要約情報リストが出力されない場合があります。

36.2.1　異常終了時要約情報リストの内容

異常終了時要約情報リストに出力される情報を，次に示します。

なお，次の説明中の「行番号」とは，コンパイルリストに示された行番号のことを指します。

最終実行文情報

異常終了したプログラムの名称※，コンパイル日時，行番号，欄が出力されます。

注※

プログラムの名称には，プログラムの種類によって次のどれかが表示されます。

- 関数名
- クラス名／メソッド名
- プログラム名

呼び出し元プログラムトレース

CALL 文または INVOKE 文で呼び出されたプログラム（内側のプログラムと最外側のプログラムの呼び出しも含む）で異常終了した場合，呼び出し元のプログラムの名称※と CALL 文または INVOKE 文の行番号が出力されます。

注※

プログラムの名称には，プログラムの種類によって次のどれかが表示されます。

- 関数名
- クラス名／メソッド名
- プログラム名

シグナル種別

シグナルの種別とその意味が出力されます。シグナル種別については，「36.9　シグナル」を参照してください。

トレースバック情報（-DebugInf,Trace オプション指定時だけ出力）

異常終了するまでに実行の対象となったプログラムの名称※と行番号が出力されます。詳細は，「36.2.2 トレースバック表示」を参照してください。

注※

プログラムの名称には，プログラムの種類によって次のどれかが表示されます。

- 関数名
- メソッド名
- プログラム名

環境変数情報

異常終了時の実行時環境変数の名称と指定値が出力されます。環境変数情報については，「36.2.3　環境変数情報表示」を参照してください。

出力例

異常終了時要約情報リストの出力例を，次に示します。

```
0---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
                                                                              }1.
     COBOL2002 (X) VV-RR  ***異常終了時要約情報リスト*** YYYY-MM-DD HH:MM:SS
最終実行文情報       プログラム名 = SAMPLE2                                    }
                     コンパイル日付（時間） = YYYY-MM-DD (HH:MM:SS)            }
                     行番号 = 001300,欄 = 012                                  }
                                                                               }
シグナル種別 = SIGSEGV（セグメンテーション違反）                               }2.
                                                                               }
呼び出し元プログラム　トレース                                                 }
  プログラム名                    行番号                                       }
  MAINPRG                         001800                                       }
  SAMPLE2                         000800                                       }
                                                                               }
節名／段落／Ｃ１分岐トレースバック                                             }
  プログラム名                    行番                                         }
  SAMPLE2                         001200 001100 001000 000900 000800           }3.
  SAMPLE1                         000700                                       }
  MAINPRG                         001700 001600 001500 001400 001300 001200    }
                                  001100 001000 000900 000800 000700           }
                                                                               }
環境変数情報                                                                   }
CBLLANG=UNICODE                                                                }
    :                                                                          }
LANG=ja_JP.UTF-8                                                               }4.
LD_LIBRARY_PATH=.:/opt/HILNGcbl2k/lib:/opt/HIISlib/lib:/opt/HISORTlib/lib:/op  }
  t/HiRDB_S～                                                                  }
PATH=/usr/local/bin:/bin:/usr/bin:.:/opt/HILNGcbl2k/bin:/opt/HIISlib/bin:/opt  }
  /HISORTli～                                                                  }
```

1.異常終了時要約情報リストの先頭に出力されるリストヘッダの内容を次に示します。

COBOL2002

COBOL2002 を示します。

(X)

COBOL2002 の識別記号を示します。詳細は「付録 K.2　このマニュアルでの表記」を参照してください。

VV-RR

COBOL2002 のバージョン番号を示します。

YYYY-MM-DD

異常終了したプログラムの実行日付（年-月-日）を示します。

HH:MM:SS

異常終了したプログラムの実行時刻（時:分:秒）を示します。

2.-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange のどれかのオプションを指定した場合，最終実行文情報，および CALL 文または INVOKE 文のトレース情報が出力されます。

3.-DebugInf,Trace オプションを指定した場合，トレースバック情報が出力されます。

4.-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange のどれかのオプションを指定した場合，環境変数情報を出力します。

なお，1 行が 79 カラムを超える場合は，改行します。

注意事項

最適化によって，最終実行文情報の行番号に数行のずれが発生したり，0 になることがあります。正しい行番号／欄の情報を出力するには，コンパイル時に-Optimize,0 オプションを指定して最適化を抑止する必要があります。

36.2.2 トレースバック表示

-DebugInf,Trace オプションを指定してコンパイルしたプログラムでは，異常終了時要約情報リストにトレースバック情報が出力されます。

トレースバック情報には，異常終了するまでに実行の対象となったプログラムの名称と次の行番号※が出力されます。

- PROCEDURE DIVISION または ENTRY 文の行番号※
- 実行された手続きの名称が書かれた行の行番号※
- IF 文などで分岐したあと，最初に実行した文の行番号※

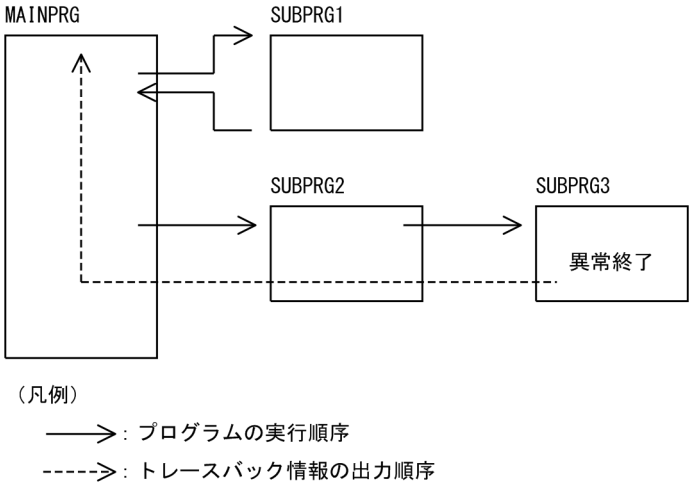
出力される行番号※は 1 プログラム当たり最大 240 個です。

注※

行番号は，コンパイルリストに示された番号です。

トレースバック情報は，次の例の点線で示した矢印のように追跡して出力されます。この例では，プログラム SUBPRG1 は実行済みであってもトレースバック情報の対象にはなりません。

（トレースバック情報出力範囲の例）



36.2.3　環境変数情報表示

COBOL プログラム異常終了時に有効な環境変数を出力します。異常終了の要因に環境変数の設定値が関係すると考えられる場合，環境変数情報からその値を容易に求められます。

出力する環境変数を次に示します。

- COBOL2002 実行時環境変数※
- システム環境変数 PATH，LANG，LD_LIBRARY_PATH（Linux）
- システム環境変数 PATH，LANG，LIBPATH（AIX）

注※

"CBL"，"COBOL"で始まる環境変数名が出力対象となります。このため，COBOL2002 実行時に内部的に設定されている環境変数も出力される場合があります。

36.2.4　異常終了時要約情報リストの出力先

異常終了時要約情報リストの出力先は，環境変数 CBLABNLST で指定します。

環境変数の指定例を次に示します。

形式

```
CBLABNLST=/users/abend.lst
export CBLABNLST
```

規則

- ファイルは，絶対パス名で指定します。

- 指定したファイルがすでにある場合は，ファイルの最後にリストが追加されます。ファイルがない場合は，ファイルが新規に作成されます。
- 複数のプロセスから同時に一つのファイルに対して異常終了時要約情報リストを出力した場合，動作は保証しません。このため，環境変数 CBLABNLST で指定する異常終了時要約情報リストの出力先は，プロセスごとに異なるファイルとなるようにしてください。
- 環境変数 CBLABNLST の指定がない場合，異常終了時要約情報リストは，標準エラー（stderr）に出力されます。
- また，環境変数に値を設定していない場合（"CBLABNLST="だけを指定した場合），ファイルは出力されません。

36.2.5　プログラム混在時のリストの内容

次のように種類の異なるプログラムが混在していると，出力される異常終了時要約情報リストの内容が異なります。

- デバッグ用オプション（-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，または-DebugRange）を指定したプログラムと指定しないプログラムが混在している場合
- COBOL 以外のプログラムが混在している場合

プログラムが混在している場合に出力される異常終了時要約情報リストの内容を次に示します。

表 36–2　プログラム混在時の異常終了時要約情報リストの内容

異常終了したプログラム	異常終了前の状態		
	-DebugInf,Trace 指定のプログラムがすでに動作している	デバッグ用オプション指定のプログラムがすでに動作している	デバッグ用オプション指定のプログラムがまだ動作していない
デバッグ用オプションのどれかを指定したプログラム	○※1	×	×
-DebugInf,Trace を指定したプログラム	○	○	○
デバッグ用オプション指定のないプログラム，またはＣなどの他言語プログラム	×※1	×※2	リストは出力されない。

（凡例）

○：異常終了時要約情報リスト中にトレースバック情報を含む

×：異常終了時要約情報リスト中にトレースバック情報を含まない

注※1

リストは，異常終了したプログラムよりも前に制御が渡っている-DebugInf,Trace 指定のプログラムから出力されます。

注※2

リストは，異常終了したプログラムよりも前に制御が渡っている-DebugInf，-DebugCompati，-DebugData，-TDInf，-CVInf，または-DebugRange 指定のプログラムから出力されます。

36.3　データ領域ダンプリスト

COBOL2002 では，プログラムが異常終了したとき，または CBLDATADUMP サービスルーチンを呼び出したときのデータ領域の状態を，データ領域ダンプリストとして出力できます。

データ領域ダンプリストを出力するには，COBOL プログラムのコンパイル時に-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange のどれかのオプションを指定してください。データ領域ダンプリストには，どれかのオプションが指定されているプログラムの情報が出力されます。

COBOL プログラムの翻訳時に-SrcList,xxxxx,DataLoc オプション※を指定して，コンパイルリスト（原始プログラムリスト）にデータ項目の相対位置を表示させた場合，データ領域中の各データ項目の相対位置がわかるため，データ領域を効率良く参照できます。

注※

xxxxx には，OutputAll，CopyAll，CopySup，NoCopy のどれかを指定します。

なお，相対位置の表示については，「32.5.13　リスト出力の設定」の「(1)　-SrcList オプション」および「付録 D.2　リストの見方」の「(4)　相対位置表示時の原始プログラムリスト」を参照してください。

36.3.1　データ領域ダンプリストの内容

(1)　データ領域ダンプの出力条件

データ領域ダンプの出力の対象となるプログラムを次に示します。

- 1 回以上実行され，かつ，異常終了の時点，または CBLDATADUMP サービスルーチン実行の時点で，CANCEL 文で取り消されていないプログラム
- 1 回以上実行された利用者定義関数
- 異常終了した時点，または CBLDATADUMP サービスルーチン実行の時点で動作中のメソッド
- 異常終了したメソッド，または CBLDATADUMP サービスルーチン実行時に動作中のメソッドを含むクラス

(2)　出力される内容

データ領域ダンプリストに出力される情報を次に示します。

- 異常終了したプログラム，または CBLDATADUMP サービスルーチンを呼び出すプログラムが実行されるまでに 1 度以上実行された COBOL プログラム名※1 と，定義されているデータの内容※2

注※1

プログラムの名称には，プログラムの種類によって次のどれかが表示されます。

- 関数名
- クラス名／メソッド名
- プログラム名

注※2

- 実行が終了したプログラムの局所場所節（LOCAL-STORAGE SECTION）は出力されない
- 実行が終了した INITIAL 属性プログラムの作業場所節（WORKING-STORAGE SECTION）は出力されない。

- EXTERNAL 句を指定したファイル名とレコード領域の内容
- EXTERNAL 句を指定したデータ名とデータの内容

  EXTERNAL 指定のファイル名およびデータ名は，外部属性を持ち，COBOL 実行単位内で一つの領域を共用するため，特定のプログラムには所属しません。このため，EXTERNAL 句を指定したファイル名とレコード領域および EXTERNAL 句を指定したデータ名とデータの内容は，定義されたプログラムのコンパイラオプション※指定の有無に関わらず表示されます。ただし，コンパイラオプション※を指定したプログラムが一つも実行されていない場合，EXTERNAL 句を指定したファイル名とレコード領域および EXTERNAL 句を指定したデータ名とデータの内容は表示されません。

注※

コンパイラオプションは次のどれかです。

```
-DebugInf, -DebugInf,Trace, -DebugCompati, -DebugData, -TDInf, -CVInf, -DebugRange
```

これらのリストは，それぞれの領域の先頭を 0 としたデータの相対的な位置と内容をダンプ形式で出力したものです。

(3)　出力例

(a)　プログラムで定義されたデータ領域の出力例

```
      COBOL2002 (c)  VV-RR    ***データ領域ダンプリスト***    YYYY-MM-DD  HH:MM:SS

**************************************************
* プログラムで定義されたデータ領域（EXTERNAL以外）*
**************************************************

<プログラム名 = SUBPRG1>

<FILE SECTION>
位置----   内容---- -------- -------- --------    ----------------
00000000   00000000 00000000 00000000 00000000    ................
00000010   53554250 52472030 30314242 42424242    SUBPRG 001BBBBBB
00000020   42424242 42424242 42424242 42424242    BBBBBBBBBBBBBBBB
           LINES 00000030-00000050 SAME AS ABOVE
00000060   42424242 42424242 42420000 00000000    BBBBBBBBBB......
00000070   00000000 00000000                      ........

<WORKING-STORAGE SECTION>
位置----   内容---- -------- -------- --------    ----------------
00000000   30303031 30303030 30303033 00000000    000100000003....
00000010   20202020 20202020 20202020 20202020
           LINES 00000020-00000060 SAME AS ABOVE
00000070   20202020 00000000                          ....

<LOCAL-STORAGE SECTION>
位置----   内容---- -------- -------- --------    ----------------
00000000   30303031 30303030 30303033 00000000    000100000003....
00000010   20202020 20202020 20202020 20202020
           LINES 00000020-00000060 SAME AS ABOVE
00000070   20202020 00000000                          ....
```

注

　同一内容の行が続く場合は，「LINE(S)位置　SAME AS ABOVE」が表示されます。

このリスト中のプログラム名は，PROGRAM-ID 段落に書かれたプログラム名です。

(b)　EXTERNAL 句を指定したファイルのレコード領域の出力例

```
      COBOL2002 (c) VV-RR   ***データ領域ダンプリスト***   YYYY-MM-DD  HH:MM:SS

**************************************
* EXTERNAL  指定ファイルのレコード領域 *
**************************************

<ファイル名 = FILE1>

位置----   内容---- -------- -------- --------    ----------------
00000000   45585445 524e414c 30313131 31313131    EXTERNAL01111111
00000010   31313131 31313131 31313131 31313131    1111111111111111
           LINES 00000020-00000040 SAME AS ABOVE
00000050   31313131 31313131 3131                 1111111111
```

このリスト中のファイル名は，SELECT の直後に書かれたファイル名です。

(c)　EXTERNAL 句を指定したデータ領域の出力例

```
      COBOL2002 (c) VV-RR   ***データ領域ダンプリスト***   YYYY-MM-DD  HH:MM:SS

******************************
* EXTERNAL  指定のデータ領域 *
******************************

<データ名 = ZZZZZ>

位置----   内容---- -------- -------- --------    ----------------
00000000   5a5a5a5a 5a5a5a5a 5a5a5a5a 5a5a5a5a    ZZZZZZZZZZZZZZZZ
00000010   5a5a5a5a                               ZZZZ

<データ名 = A>

位置----   内容---- -------- -------- --------    ----------------
00000000   45585445 524e414c 45585445 524e414c    EXTERNALEXTERNAL
           LINES 00000010-00000040 SAME AS ABOVE
```

このリスト中のデータ名は，EXTERNAL 句を指定した 01 レベルのデータ名を示します。位置と内容の部分は，データの先頭を 0 とした相対的な位置と内容をダンプ形式で示したものです。

36.3.2　データ領域ダンプリストの出力先

データ領域ダンプの出力先は，環境変数 CBLDDUMP または環境変数 CBLDATADUMPFILE が存在する場合に指定されたファイルに出力します。

- プログラム異常終了時
  環境変数 CBLDDUMP

- CBLDATADUMP サービスルーチン使用時
  環境変数 CBLDATADUMPFILE

環境変数の指定方法を次に示します。

(1) プログラム異常終了時

```
CBLDDUMP=/users/data.dmp
export CBLDDUMP
```

(2) CBLDATADUMP サービスルーチン使用時

```
CBLDATADUMPFILE=/users/data.dmp
export CBLDATADUMPFILE
```

ファイルは，絶対パス名で指定します。

指定したファイルがすでにある場合は，ファイルの最後にリストが追加されます。ファイルがない場合は，ファイルが新規に作成されます。

環境変数の指定がない場合，および環境変数に値を設定していない場合（"CBLDDUMP="または"CBLDATADUMPFILE="だけを指定した場合），リストは出力されません。

36.4　データ領域ダンプリスト出力情報の選択

データ領域ダンプリストを出力する場合，プログラムの構成によっては，多量のリストが出力されることがあります。

リストが多量に出力されると，データ領域を参照するときに，該当する領域を探すのが困難だったり，エディタで編集できなかったりする場合があります。

出力するリストの情報を選択するためには，環境変数 CBLDMPPGMN または環境変数 CBLDMPLEVEL を使用します。

なお，環境変数 CBLDMPPGMN と環境変数 CBLDMPLEVEL は，組み合わせても使用できます。

36.4.1　環境変数の設定

(1)　CBLDMPPGMN

形式

```
CBLDMPPGMN=プログラム名称〔:プログラム名称〕
```

規則

- プログラム名称には，データ領域ダンプリストのプログラムで定義されたデータ領域（EXTERNAL 句指定以外）の情報で，出力したいプログラムのプログラム名称を指定します。
- 複数のプログラム名称を指定する場合は，コロン（:）で区切ります。
- プログラム名称の直前に空白文字を入れてはいけません。

使用例

```
CBLDMPPGMN=SAMPLE1:SAMPLE5
export CBLDMPPGMN
```

プログラム名称が SAMPLE1 と SAMPLE5 の情報だけを出力します。

(2)　CBLDMPLEVEL

形式

```
CBLDMPLEVEL =  {1 | 2}
```

規則

- 1 を指定した場合は，プログラムで定義されたデータ領域（EXTERNAL 句指定以外）の情報だけを出力します。

- 2 を指定した場合は，EXTERNAL 句指定ファイルのレコード領域と EXTERNAL 句指定のデータ領域の情報だけを出力します。

注意事項

上記以外の値が指定された場合や，これらの環境変数の指定がない場合は，データ領域ダンプリストの出力情報の選択はできません。

36.4.2 環境変数の組み合わせによるデータ領域ダンプリストの出力情報の設定

環境変数 CBLDMPPGMN，CBLDMPLEVEL の組み合わせによる出力情報の内容を，次に示します。

表 36–3 環境変数の組み合わせによるデータ領域ダンプリストの出力内容

環境変数 CBLDMPPGMN	環境変数 CBLDMPLEVEL		
	指定なし	指定あり	
		1	2
指定なし	A，B	A	B
指定あり	C，B	C	B

出力情報の内容

A

プログラムで定義されたデータ領域（EXTERNAL 句指定以外）の情報をすべて出力します。

B

EXTERNAL 句指定ファイルのレコード領域，および EXTERNAL 句指定のデータ領域の情報を出力します。

C

プログラムで定義されたデータ領域（EXTERNAL 句指定以外）の情報を指定されたプログラム名についてだけ出力します。

なお，Bだけ出力する場合でも，プログラムで定義されたデータ領域（EXTERNAL 句指定以外）出力時のヘッダ情報は出力されます。ただし，A，Cだけ出力する場合は，EXTERNAL 句指定ファイルのレコード領域，または EXTERNAL 句指定のデータ領域出力時のヘッダ情報は出力されません。

リスト出力のプログラム名は，プログラムの種類によって，次のどれかとなります。

- 関数名
- クラス名／メソッド名
- プログラム名

36.4.3　出力例

次に，環境変数 CBLDMPPGMN または CBLDMPLEVEL を指定して出力したデータ領域ダンプリストの出力例を示します。

(1)　CBLDMPPGMN=SUBPRG1 かつ CBLDMPLEVEL=1 を指定した場合

```
COBOL2002 (c) VV-RR     ***データ領域ダンプリスト***    YYYY-MM-DD HH:MM:SS

**************************************************
* プログラムで定義されたデータ領域 (EXTERNAL 以外)  *
**************************************************

<プログラム名 = SUBPRG1>

<FILE SECTION>
位置----     内容---- -------- -------- --------      ----------------
00000000     00000000 00000000 00000000 00000000      ................
00000010     53554250 52472030 30314242 42424242      SUBPRG 001BBBBBB
00000020     42424242 42424242 42424242 42424242      BBBBBBBBBBBBBBBB
             LINES 00000030-00000050 SAME AS ABOVE
00000060     42424242 42424242 42420000 00000000      BBBBBBBBBB......
00000070     00000000 00000000 00000000 00000000      ................

<WORKING-STORAGE SECTION>
位置----     内容---- -------- -------- --------      ----------------
00000000     30303031 30303030 30303033 00000000      000100000003....
00000010     20202020 20202020 20202020 20202020
             LINES 00000020-00000060 SAME AS ABOVE
00000070     20202020 00000000                            ....
<LOCAL-STORAGE SECTION>
位置----
00000000      内容---- -------- -------- --------      ----------------
00000010     30303031 30303030 30303033 00000000      000100000003....
             20202020 20202020 20202020 20202020
             LINES 00000020-00000060 SAME AS ABOVE
00000070     20202020 00000000                            ....
```

(2) CBLDMPLEVEL=2 だけを指定した場合

```
COBOL2002 (c) VV-RR     ***データ領域ダンプリスト***    YYYY-MM-DD HH:MM:SS

**************************************************
* プログラムで定義されたデータ領域 (EXTERNAL 以外) *
**************************************************

COBOL2002 (c) VV-RR     ***データ領域ダンプリスト***    YYYY-MM-DD HH:MM:SS  ------1.

****************************************
* EXTERNAL 指定ファイルのレコード領域 *
****************************************

<ファイル名 = FILE1>

位置----    内容---- -------- -------- --------      ----------------
00000000    45585445 524e414c 30313131 31313131      EXTERNAL01111111
00000010    31313131 31313131 31313131 31313131      1111111111111111
            LINES 00000020-00000040 SAME AS ABOVE
00000050    31313131 31313131 3131                   1111111111
COBOL2002 (c) VV-RR     ***データ領域ダンプリスト***    YYYY-MM-DD HH:MM:SS  ------2.

****************************
* EXTERNAL 指定のデータ領域 *
****************************

<データ名 = ZZZZZ>

位置----    内容---- -------- -------- --------      ----------------
00000000    5a5a5a5a 5a5a5a5a 5a5a5a5a 5a5a5a5a      ZZZZZZZZZZZZZZZZ
00000010    5a5a5a5a                                 ZZZZ

<データ名 = A>

位置----    内容---- -------- -------- --------      ----------------
00000000    45585445 524e414c 45585445 524e414c      EXTERNALEXTERNAL
            LINES 00000010-00000040 SAME AS ABOVE
```

なお，上記のリストでは，印刷すると 1.および 2.で改ページされます。

36.5　プログラム間整合性チェック

引数および返却項目を利用してプログラム間の連絡をする場合，呼び出し元プログラムと呼び出し先プログラムの引数および返却項目の形式が異なると，プログラムが異常終了したり，不正な動作をしたりすることがあります。COBOL2002 では，-DebugCompati または-TDInf オプションを指定すれば，プログラム間の引数および返却項目の整合性をチェックできます。

なお，COBOL2002 では，プログラム間整合性チェックよりも，例外名 EC-PROGRAM-ARG-MISMATCH での引数と返却項目の適合チェックの使用を推奨します。引数と返却項目の適合チェックの詳細は，マニュアル「COBOL2002　言語　標準仕様編」「10.7　引数と返却項目の適合」を参照してください。

36.5.1　整合性チェックの内容

-DebugCompati または-TDInf オプションを指定すると，次の項目についてプログラム間の整合性をチェックできます。

- 引数の長さ
- 引数の個数
- 引数の属性（BY REFERENCE，BY CONTENT，BY VALUE）
- 返却項目の長さ

コンパイラオプションと整合性チェックの関係

呼び出し元プログラム，呼び出し先プログラムのコンパイル時に指定したオプションと，整合性チェックの関係を次に示します。

呼び出し先	呼び出し元		
	-DebugCompati	-TDInf	-DebugCompati，-TDInf なし
-DebugCompati	○	△	×
-TDInf	△	△	×
-DebugCompati，-TDInf なし	×	×	×

（凡例）

○：整合性チェックが行われる

△：テストデバッガを使用したデバッグ時だけ，整合性チェックが行われる

×：整合性チェックは行われない

規則

プログラム間の整合性チェックに関する規則を次に示します。

- 整合性チェックでプログラム間で受け渡す引数および返却項目に矛盾がある場合は，メッセージが出力され，プログラムが異常終了します。

ただし，-DebugCompati オプションの指定がないプログラムをテストデバッグする場合は，環境変数 CBLPRMCHKW に YES を指定すると処理を続行できます。また，環境変数 CBLPRMCHKW に NOCHK を指定すると，-DebugCompati オプション指定時およびテストデバッグ時に，プログラム間整合性チェックで不整合があってもエラーとしないで，処理を続行できます。詳細は，「36.5.2　整合性チェックの警告エラー出力」を参照してください。

- 整合性をチェックするのは，CALL 文で COBOL プログラムを呼び出すときで，実行時にチェックします。
- CALL 定数で内側のプログラムを呼び出す場合，-DebugCompati オプションまたは-TDInf オプションの指定がないときも，翻訳時に整合性をチェックします。
- COBOL プログラムと C プログラムとの間のチェックはしません。
- 引数が可変長項目のとき，引数の長さはチェックしません。
- -Main,System，-Main,V3 オプションを指定したプログラムに対しては，引数の長さ，引数の個数をチェックしません。
- -SimMain オプションを指定した実行可能ファイルで，最初に制御が渡るプログラムでの引数チェックはしません。
- BY VALUE 指定の浮動小数点項目は，送り出し側作用対象と受け取り側作用対象で同じ属性でなければなりません。属性が異なる場合，整合性チェックでエラーとなります。
- -DebugCompati オプションを指定すると，一部の例外名に対する TURN 指令が無効となります。詳細は「21.8.2　例外検出での注意事項」の「(4)　コンパイラオプションとの関連性」を参照してください。
- 引数が動的長基本項目の場合，送り出し側作用対象と受け取り側作用対象がともに動的長基本項目で，かつその字類が一致しなければなりません。また，LIMIT 指定が双方に指定されている場合は，その値が一致していなければなりません。

36.5.2　整合性チェックの警告エラー出力

プログラム間の引数および返却項目に矛盾があっても，整合性エラーとしないでテストデバッグを実行したい場合，環境変数 CBLPRMCHKW を指定します。

形式

```
CBLPRMCHKW = {NOCHK | YES}
export CBLPRMCHKW
```

規則

- 環境変数 CBLPRMCHKW に NOCHK を指定した場合

  プログラム間の引数および返却項目について，整合性がチェックされません。この場合，プログラム間の引数および返却項目に不整合があっても異常終了しないで，メッセージも出力されません。

-DebugCompati または-DebugRange オプションを指定したプログラムでも，プログラム間の引数および返却項目に不整合があっても異常終了しないで，メッセージも出力されません。ただし，プログラム間の引数および返却項目の整合性チェック以外のチェック（繰り返し回数の範囲チェックなど）は，実行されます。

- 環境変数 CBLPRMCHKW に YES を指定した場合

  -DebugCompati および-DebugRange オプションを指定しないプログラムのテストデバッグ中に，プログラム間の引数および返却項目に関するエラーが発生したときには，警告メッセージが出力され実行が継続されます。

  -DebugCompati または-DebugRange オプションを指定したプログラムでは，環境変数 CBLPRMCHKW に YES を指定しても無効となり，エラー発生時にはメッセージが出力され異常終了します。

- YES，または NOCHK 以外の文字を指定した場合は，この環境変数の指定は無効になります。

注意事項

- この環境変数を指定した場合，引数および返却項目の矛盾した領域に対して不当な値を設定したり，不当な領域を参照したりすると，異常終了やシステムダウンすることがあります。また，引数および返却項目に次のような矛盾があるプログラム間で，データを受け渡しする場合はプログラムの見直しが必要です。

  ・RETURNING 指定ありと，指定なしが混在している

  ・USING 指定のデータ項目数が異なる

  ・データ項目の長さが８バイトを超えるデータと８バイト以下のデータとで受け渡しする

- この環境変数の指定は，引数が不一致の場合の COBOL の動作を保証するものではありません。

36.6　添字，指標の繰り返し回数，制御変数チェック

36.6.1　デバッグオプションとの関連性

-DebugCompati または-DebugRange オプションを指定してコンパイルしたプログラムの実行時，次のエラーが検出されると，メッセージが出力され，実行が中止します。

なお，-DebugCompati オプションおよび-DebugRange オプションについては，「32.5.9　デバッグの設定」の「(3)　-DebugCompati オプション」および「(7)　-DebugRange オプション」を参照してください。

(1)　-DebugCompati オプション指定時

- 表操作で使用する添字または指標名が指す表要素が表の範囲外である。
- 部分参照の指定がデータ項目の範囲外である。
- プログラム間で整合性が取れていない。

(2)　-DebugRange オプション指定時

- 表操作で使用する添字または指標名の値が各次元の繰り返し回数の範囲外である。
- 部分参照の指定がデータ項目の範囲外である。

36.6.2　注意事項

- -DebugCompati または-DebugRange オプションを指定すると，一部の例外名に対する TURN 指令が無効となります。詳細は「21.8.2　例外検出での注意事項」の「(4)　コンパイラオプションとの関連性」を参照してください。
- COBOL2002 では，-DebugCompati または-DebugRange オプションよりも，例外名 EC-BOUND-REF-MOD，または EC-BOUND-SUBSCRIPT での，添字，指標の繰り返し回数，制御変数チェックの使用を推奨します。

36.7　データ例外検出機能

-DebugData オプションを指定してコンパイルしたプログラムを実行したとき，格納値とデータ項目の属性とがチェックされます。チェックされるのは，次の外部または内部 10 進項目です。格納値がデータ項目の属性と矛盾している場合，データ例外エラーとなります。

- 転記の送り出し側作用対象（受け取り側作用対象が 2 進項目の場合だけ）
- 算術式の作用対象
- 比較の作用対象
- 添字
- 部分参照の最左端位置と長さ

データ例外が検出されると，エラーメッセージが出力され，プログラムが異常終了します。

-DebugData,ValueHex オプションを指定してコンパイルしたプログラムを実行した場合，データ例外が検出されると，出力されるエラーメッセージにデータ項目の属性と矛盾している格納値が 16 進数で表示されます。

36.8　コアダンプの出力

-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange オプションのどれかを指定してコンパイルしたプログラムが異常終了したとき，通常はコアダンプは出力されません。コアダンプが出力されるためには，環境変数 CBLCORE に 1 を設定します。環境変数 CBLCORE の形式を次に示します。

形式

```
CBLCORE=1
```

規則

1 以外の値を設定したときは，コアダンプが出力されません。

注意事項

コアダンプの出力先は，実行環境のシステム設定に依存します。また，システムの core ファイルサイズの設定値が 0 になっている場合，core ファイルは出力されません。システムのマニュアルなどで必要な設定値をご確認ください。

36.9　シグナル

36.9.1　COBOL が登録するシグナル

シグナルとは，ハードウェア，およびソフトウェアで発生する割り込みのことです。COBOL では，-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange オプションのどれかを指定してコンパイルしたプログラムを実行する場合，次に示すシグナルを登録します。

- SIGILL（無効な命令）
- SIGIOT（abort による異常終了）
- SIGEMT（EMT 命令）(AIX で有効)
- SIGFPE（浮動小数点例外，算術演算エラー）
- SIGBUS（バスエラー）
- SIGSEGV（セグメンテーション違反）

なお，マルチスレッド対応 COBOL プログラムでは，さらに次の二つが追加されます。これらは，システムのシグナルではなく，異常終了時要約情報リストに表示されるシグナル種別です。

- RUNTIME ERROR（実行時エラーの発生）
- CBLABN（CBLABN サービスルーチンの呼び出し）

COBOL 以外のプログラムで上記シグナルを登録した場合，異常終了時の結果を保証できない場合があります。詳細は，「19.1　C 言語との連携」を参照してください。

上記シグナルが発生した原因として，代表的なものを次に示します。

- 添字，または指標の値が OCCURS 句で定義した範囲を超えた場合
- 部分参照で一意名の範囲を超えて参照した場合
- 呼び出すプログラムと呼び出されるプログラムとで引数の属性，長さ，数が不一致である場合
- COBOL 実行時エラーの発生，CBLABN サービスルーチンの呼び出し，または COBOL プログラム以外 abort システムコールを発行した場合

また，-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange オプションのどれかを指定してコンパイルされた COBOL プログラムに，一度以上制御が渡り，上記のシグナル発生によって異常終了した場合，COBOL の戻り値は 1 になります。

割り込み発生条件が，COBOL によって変更されることはありません。

36.9.2　シグナルの登録と回復

-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange オプションと-MultiThread オプション（マルチスレッド対応 COBOL プログラム）指定によるシグナルの登録と，COBOL プログラム終了時のシグナルの回復について，次に示します。

コンパイラオプション		COBOL のシグナル登録／回復	
-DebugInf -DebugInf,Trace -DebugCompati -DebugData -TDInf -CVInf -DebugRange	-MultiThread	シグナル登録	シグナル回復
あり	あり	登録あり シグナルは，すべてのスレッドで有効	回復なし
	なし※2	登録あり	回復あり
なし※1	－	登録なし	－
あり／なしが混在	あり	登録あり シグナルは，すべてのスレッドで有効	回復なし
	なし※2	登録あり※3	回復あり※3

（凡例）

－：該当しない

注※1

-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange オプションを指定しないプログラムだけが動作した場合です。

注※2

-MultiThread オプションを指定しないマルチスレッド非対応の COBOL プログラムだけが動作した場合です。

注※3

-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange オプション指定プログラムが初めて動作したときに COBOL がシグナルを登録し，COBOL プログラム終了時シグナルが回復します。

36.9.3　シグナルの登録と回復の注意事項

(1)　COBOL がシグナルを登録する場合

- プログラム実行中にシグナルが発生したとき，COBOL は異常終了を通知する COBOL のメッセージ，および異常終了時要約情報リストを出力して，マルチスレッド対応 COBOL プログラム動作時はスレッドを終了します。マルチスレッド対応 COBOL プログラム以外ではプロセスを終了します。
- COBOL プログラム以外でシグナルを登録し，その後，COBOL が同じシグナルを登録するとき，COBOL がシグナルを回復しないかぎり，先に登録していたシグナルのアクションには従いません。

(2)　COBOL がシグナルを登録しない場合

- プログラム実行中にシグナルが発生しても，異常終了を通知する COBOL のメッセージ，および異常終了時要約情報リストは出力されません。プログラムの終了方法はシステムに依存します。
- COBOL プログラム以外でシグナルを登録している場合は，登録時のアクションに従います。

(3)　COBOL がシグナルを回復する場合

- COBOL プログラムの終了時，COBOL は登録したシグナルを COBOL プログラム実行前の状態に回復します。
- CBLEND サービスルーチン呼び出し後，シグナルは COBOL プログラム実行前の状態に回復しています。サービスルーチンの詳細は，「29.　サービスルーチン」を参照してください。

(4)　COBOL がシグナルを回復しない場合

- COBOL プログラムの終了，または CBLEND サービスルーチン呼び出し後，COBOL はシグナルを回復しません。
- COBOL プログラム以外でシグナルを登録し，その後，COBOL が同じシグナルを登録するとき，先に登録していたシグナルの動作には従いません。

(5)　マルチスレッド対応 COBOL プログラムの場合

- 登録したシグナルは，すべてのスレッドで有効となります。
- COBOL プログラムの終了時，COBOL は登録したシグナルを回復しません。このため，CBLEND サービスルーチン呼び出し後でも，シグナルは COBOL が登録した状態です。
- シグナルの動作は，プロセス内のすべてのスレッドに影響があります。

  このため，次の順でシグナルが発生した場合，COBOL はシグナルをとらえ，異常終了を通知する COBOL のメッセージを出力してシグナルが発生したスレッドを終了します。

  1. スレッドで，-DebugInf オプション指定のプログラムを実行した。
  2. 1.以降，-DebugInf オプション指定のプログラムが動作していない別スレッドでシグナルが発生した。

(6) COBOL 以外のプログラムでのシグナル登録（ユーザの設定）

- COBOL プログラムの呼び出し終了後，シグナル発生時の動作を規定する必要があるときは，COBOL 以外のプログラムでシグナルの再登録を行ってください。
- -DebugInf オプションを指定したプログラムが実行されたときに COBOL が登録するシグナルが，COBOL 以外のプログラムで登録するシグナルに影響があるときは，プログラムの制御を変更するか，または-DebugInf オプションを指定しなければ回避できます。

36.9.4 プロセスの終了

異常終了時要約情報リストを出力後，プロセスは終了コード 1 で終了します。

36.9.5 プログラム実行時にシグナル登録しない環境変数 CBLEXCEPT

この環境変数に NOSIGNAL を指定すると，デバッグ用コンパイラオプション-DebugInf, -DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange のどれかを指定したプログラムの実行中にシグナル登録をしません。

このため，異常終了要因となるシグナルが発生しても，デバッグ情報は出力しません。終了方法は，制御プログラムの終了処理に依存します。

制御プログラムの終了処理を次に示します。

- NOSIGNAL を指定した場合，セグメンテーション違反など，シグナルの発生により異常終了したとき，メッセージ，および異常終了時要約情報リストなどは出力されません。このとき，core ファイルが出力されます※ので，core ファイル，実行可能ファイルからシステムデバッガ(xdb や dbx など)のトレース情報を採取してください。

  また，ユーザプログラムで例外処理（シグナル登録）があれば，ユーザプログラムに制御が移ります。

  なお，ミドルソフトウェアなどで，この例外処理を実装している場合もありますので，異常終了時の出力情報を確認してください。
- NOSIGNAL の指定をしても，COBOL 実行時エラーメッセージを出力して異常終了する場合は，従来と同じ動作となり，デバッグ用コンパイラオプションを指定したプログラムでは，メッセージ，および異常終了時要約情報リストが出力されます。
- この環境変数を指定しない場合は，従来と同様に，COBOL がシグナルを登録し，シグナル発生時はデバッグ情報を出力する動作となります。
- この環境変数を指定しても実行性能への影響はありません。

  注※

  システムの core ファイルサイズの設定値が 0 になっている場合，core ファイルが出力されません。システムのマニュアルなどで必要な設定値をご確認ください。

36.10　DISPLAY 文による一意名の 16 進ダンプ表示

DISPLAY 文での表示では，英数字項目に印字不可能文字が設定されている場合や，2 進項目に PICTURE 句のけた数を超えて値が格納されている場合には，内容を確認できないことがあります。このような場合に，一意名の格納値を 16 進ダンプ表示すると，一意名の内容を確認できます。

一意名の格納値をバイト単位で 16 進ダンプ表示するときは，DISPLAY 文に IN DATA DUMP を指定します。

DISPLAY 文の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」を参照してください。

各データ属性を 16 進ダンプ表示したときの表示例を次の表に示します。

表 36-4　データ属性を 16 進ダンプ表示したときの表示例

データ属性	16 進ダンプ表示の例	
	格納されている定数値	表示形式
集団項目，英字項目，英数字項目，英数字編集項目	'ABCD'	41424344
日本語項目，日本語編集項目	N'あいう'	82a082a282a4
外部 10 進項目	+123	313233
内部 10 進項目	+123	123c
数字編集項目	+12.3	2b31322e33
2 進項目，COMP-X 項目，アドレス名，アドレスデータ項目，ポインタ項目	値 10(H'0000000A')が 4 バイトの領域に格納されている場合	ビッグエンディアンの場合 0000000a リトルエンディアンの場合 0a000000
外部浮動小数点項目	+0.25E+0	2b322e35452d3031
内部浮動小数点項目	+0.25E+0	ビッグエンディアンの場合 3e800000 リトルエンディアンの場合 0000803e
外部ブール項目	B'010'	303130
内部ブール項目	B'010'	40

36.11　テストデバッグ機能

テストデバッガは，TD コマンドなどでプログラムの実行を制御しながらプログラムをテストしたり，デバッグしたりする機能です。テストデバッガで，プログラムの実行を実行文単位に中断し，データ項目に対して値を表示，代入，比較してデバッグできます。また，未完成のプログラムをテストできます。

テストデバッガには，ラインモードとバッチモードの二つの方法があります。

ラインモードでは，TD コマンドを入力することによって，対話的にテストしたり，デバッグしたりできます。ラインの入出力だけなので，応答の速いテストデバッグができます。

バッチモードでは，あらかじめ，実行したい TD コマンドをファイルにまとめて記述しておき，そのファイルを入力して起動することによって，一括してテストデバッグを実行する方法です。TD コマンドを記述したファイルを指定した起動コマンドを入力すると，ファイルに記述された TD コマンドを一括して実行します。一度実行を開始すれば，利用者の操作を必要としないので，効率良く大量のプログラムがテストできます。

テストデバッグ機能の詳細については，マニュアル「COBOL2002　使用の手引　操作編」を参照してください。

36.11.1　デバッグ機能

プログラムを実行しながら次の操作ができます。

- 実行プログラムの中断
  プログラムの実行を実行文単位に中断し，データの代入などができます。
- 実行の制御
  設定した中断点などに基づき実行できます。機能の概要についてはマニュアル「COBOL2002　使用の手引　操作編」を参照してください。
- データの操作
  COBOL プログラムのデータに対して，値の表示，値の代入，比較ができます。

36.11.2　テスト機能

未完成のプログラムを単体でテストできます。呼び出し元のプログラム，呼び出し先のプログラム，ファイル，または DC（データコミュニケーション）を，TD コマンドによってシミュレーションできます。

36.12　カバレージ機能

カバレージを使用して，テスト終了後，C0 メジャー，C1 メジャー，未実行文情報などのカバレージ情報を集計して表示できます。カバレージ情報を確認することでテストの進捗状況や性能を数値で把握できます。カバレージには，テストの進捗状況を定量的に表すカバレージ情報，文の実行回数をカウントするカウント情報の表示機能があります。

カバレージの方法には，バッチモードがあります。

バッチモードでは，起動コマンドの指示によって，カバレージの機能を操作できます。プログラムからカバレージを連動実行させ，一括してカバレージの蓄積およびカウント情報を表示することもできます。一度実行を開始すれば，利用者の操作を必要としないので，効率良く大量のプログラムのカバレージ情報の蓄積ができます。

カバレージ機能の詳細については，マニュアル「COBOL2002　使用の手引　操作編」を参照してください。

36.12.1　カバレージ情報の表示

カバレージ情報は，テストによって実行された手続き文の割合です。計画したテスト内容に従って，プログラムに記述した手続き文がテスト実行されたかどうかを把握でき，テスト工程の進捗度を判定できます。

実行されたテストプログラムの文はカバレージ情報としてプログラム情報ファイルに記憶されます。プログラム情報ファイルに蓄積されたカバレージ情報は，次の形式で表示できます。

- 翻訳単位ごとに実行された文の割合を，文・分岐・呼び出し文ごとに示す。
- ソース原文を表示して実行が済んだ文と実行されていない文を示す。
- 原始プログラムの修正によって変更された差分に対するカバレージ情報を示す。

さらに，蓄積したカバレージ情報をクリアする，別の環境で蓄積したカバレージ情報をマージするなどの機能があります。

36.12.2　カウント情報の表示

テスト実行させたプログラムの文の実行回数を表示します。繰り返し実行される文の実行回数や，呼び出すプログラムの実行回数も計測できるため，プログラムの実行性能を把握できます。このカウント情報を基にして，改善の施策を検討できます。

37 64bit アプリケーションの作成

この章では，AIX(64) COBOL2002 および Linux(x64) COBOL2002 について説明します。

37.1　AIX(64) COBOL2002 および Linux(x64) COBOL2002 について

AIX(64) COBOL2002 および Linux(x64) COBOL2002 では，ポインタサイズの 64bit 化に対応しています。

ここでは，AIX(64) COBOL2002 および Linux(x64) COBOL2002 の機能について説明します。

37.1.1　使用できない機能

AIX(64) COBOL2002 および Linux(x64) COBOL2002 で使用できない機能について説明します。

(1)　機能

AIX(64) COBOL2002 および Linux(x64) COBOL2002 では使用できない機能を次に示します。

表 37–1　AIX(64) COBOL2002 および Linux(x64) COBOL2002 で使用できない機能

機能名	OS		説明
	AIX (64)	Linux (x64)	
画面節（SCREEN SECTION）による画面操作	○	×	画面節（SCREEN SECTION）を使用して画面の入出力をします。
画面節（WINDOW SECTION）による画面操作	○	×	画面節（WINDOW SECTION）を使用して画面の入出力をします。
バイトストリーム入出力サービスルーチン	○	○	COBOL のレコード定義に依存しないで，C プログラムなどで作成したバイナリファイルの入出力をするサービスルーチンです。
XMAP3 を使用した書式印刷機能	×	×	書式と行データを重ね合わせる印刷（書式オーバレイ印刷）や，印刷制御付きの行データを印刷します。
リモートファイルアクセス機能	×	×	接続されているほかのワークステーションや PC 上にある，ISAM を使用する索引編成ファイルにアクセスします。
通信節による画面操作	×	×	ディスプレイとの間で画面データを送受信したり，プリンタに帳票データを送信したりします。
データコミュニケーション機能	○	○	オンラインコントロールプログラムを経由して，端末，ファイル，またはほかのプログラムとメッセージを受け渡します。
データベース操作機能（ODBC インタフェース）	×	○	SQL 埋め込み COBOL プログラムで，データベースにアクセスできます。

機能名	OS		説明
	AIX (64)	Linux (x64)	
CGI プログラム作成支援機能	×	×	COBOL2002 で作成したプログラムを CGI プログラムとして利用するための機能です。
Unicode 機能	○	○	COBOL が扱うデータを Unicode として扱うことで，プログラム間およびファイル入出力では，Unicode でやり取りをする機能です。
数字項目のけた拡張機能	○	○	数字項目（外部 10 進項目，内部 10 進項目），数字定数および数字編集項目で扱えるけた数の上限を 18 けたから 38 けたに拡張する機能です。
日本語集団項目	○	○	データ項目に GROUP-USAGE IS NATIONAL が指定された，用途が NATIONAL で字類および項類が日本語となる集団項目です。
動的長基本項目機能	○	○	DYNAMIC LENGTH 句を定義することで，実行時にデータ項目（英数字項目または日本語項目）の長さを変更できます。
定数長拡張機能	○	○	英数字定数の定数長を拡張します。
XML 連携機能	○	○	COBOL プログラムから，XML データを COBOL のレコードとして入出力します。

（凡例）

○：使用できる

×：使用できない

(2) サービスルーチン

AIX(64) COBOL2002 および Linux(x64) COBOL2002 では使用できないサービスルーチンを次に示します。

表 37–2 AIX(64) COBOL2002 および Linux(x64) COBOL2002 で使用できないサービスルーチン

サービスルーチン名	OS		説明
	AIX (64)	Linux (x64)	
CBLADDPAIR	×	×	CGI リストの最後に「名前」と「値」の対を追加します。
CBLCGIINIT	×	×	受け取ったフォーム情報から CGI リストを作成します。

サービスルーチン名	OS		説明
	AIX (64)	Linux (x64)	
CBLCGITRACE	×	×	CGI プログラムの作成を支援します。サービスルーチンのトレース情報をファイルに出力します。
CBLCONVERTTEXT	×	×	テキスト文字列を実体参照形式に変換し，出力します。
CBLCREATELIST	×	×	CGI リストを新たに作成します。
CBLDELETEPAIR	×	×	CGI リストの現在のポイント位置の「名前」と「値」の対を削除します。
CBLDESTROYLIST	×	×	CGI リストを削除し，領域を解放します。
CBLDISPLAYTEXT	×	×	テキスト文字列を出力します。
CBLENDREPEAT	×	×	CGI リストに，HTML 拡張言語の REPEAT で終端を認識する終端インジケータを追加します。
CBLFILLTEMPLATE	×	×	HTML テンプレートをインタプリットし，動的な Web ページを出力します。
CBLFINDNEXTPAIR	×	×	CGI リストの，次のポイント位置から「名前」をキーにして検索し，「値」を取得します。
CBLFINDPAIR	×	×	CGI リストの先頭ポイント位置から「名前」で検索し，「値」を取得します。
CBLGETENV	×	×	環境変数の値を取得します。
CBLGETPAIR	×	×	CGI リストの現在のポイント位置から「名前」と「値」の対を取得します。
CBLGETPAIRNEXT	×	×	CGI リストの現在のポイント位置から「名前」と「値」の対を取得し，ポイント位置を進めます。
CBLHTMLBEGIN	×	×	HTML の先頭部分を出力します。
CBLHTMLEND	×	×	HTML の終端部分を出力します。
CBLLISTCOUNT	×	×	CGI リストから「名前」と「値」の対の数を取得します。
CBLPRINTENV	×	×	CGI 環境変数の値を HTML 形式で出力します。
CBLPRINTLIST	×	×	CGI リストの内容を HTML 形式で出力します。
CBLSENDERROR	×	×	エラーメッセージを HTML 形式で出力します。
JCPOPUP	○	×	表形式のデータ項目を主画面とは別の画面に表示し，選ばれたブロック番号をインタフェース領域に格納します。

サービスルーチン名	OS		説明
	AIX (64)	Linux (x64)	
CBLDATADUMP	○	○	COBOL プログラム実行時の任意の時点でのデータ領域ダンプリストを出力します。

（凡例）

○：使用できる

×：使用できない

37.1.2 AIX(64) COBOL2002 および Linux(x64) COBOL2002 固有の言語仕様

AIX(64) COBOL2002 および Linux(x64) COBOL2002 では，ポインタサイズの 64bit 化に対応しました。そのため，AIX(64) COBOL2002 および Linux(x64) COBOL2002 に対応する COBOL2002 の言語仕様にも一部変更があります。

AIX(64) COBOL2002 および Linux(x64) COBOL2002 固有の言語仕様について説明します。

(1) アドレス系データを表現するデータ項目

長さが 8 バイトになるアドレス系データを表現するデータ項目を，次に示します。これらの項目の自然な境界は 8 バイトです。SYNCHRONIZED 句を指定してけた詰めする場合は，8 バイト境界になります。

- アドレス名
- アドレスデータ項目
- ポインタ項目
- 指標名
- 指標データ項目
- オブジェクト参照データ項目
- 既定義オブジェクト参照
- ADDRESS OF 指定で指定した一意名のアドレス
- 連絡節での BY REFERENCE 指定のデータ項目

SYNCHRONIZED 句については，マニュアル「COBOL2002　言語　標準仕様編」「9.16.80 SYNCHRONIZED 句」を参照してください。

けた詰めについては，マニュアル「COBOL2002　言語　標準仕様編」「4.4.1(7)　実行用コードの効率を高めるための項目のけた詰め」を参照してください。

(2)　長さを返す組み込み関数の戻り値

返却値のサイズが８バイト２進になる組み込み関数を，次に示します。

- LENGTH 関数
- LENGTH OF 指定で生成されるデータ領域
- COUNT-CHAR 関数
- LENGTH-OF-SUBSTRING 関数

LENGTH 関数については，マニュアル「COBOL2002　言語　標準仕様編」「11.32　LENGTH 関数」を参照してください。

COUNT-CHAR 関数については，マニュアル「COBOL2002　言語　拡張仕様編」「23.2.1　COUNT-CHAR 関数」を参照してください。

LENGTH-OF-SUBSTRING 関数については，マニュアル「COBOL2002　言語　拡張仕様編」「23.2.3　LENGTH-OF-SUBSTRING 関数」を参照してください。

37.1.3　AIX(64) COBOL2002，Linux(x64) COBOL2002 各機能の固有仕様

各機能の固有仕様について説明します。

(1)　サービスルーチンの引数でのハンドル項目

引数のハンドル項目サイズが８バイト２進になるサービスルーチンを，次に示します。

表 37–3　引数のハンドル項目サイズが８バイト２進になるサービスルーチン

サービスルーチン名	参照先
バイトストリーム入出力サービスルーチン	15.　バイトストリーム入出力サービスルーチン

(2)　インタフェース領域中のアドレス格納領域

インタフェース領域中のアドレス格納領域が８バイトになるサービスルーチンを，次に示します。

- COBOL 入出力サービスルーチン

なお，AIX(64) COBOL2002，および Linux(x64) COBOL2002 では，アドレス格納領域が自然な境界の８バイトになるように，アドレス格納領域の直前に明示的な境界の調整領域を追加しました。領域については，「13.3.2　インタフェース領域の形式」を参照してください。

(3)　併合処理のメモリサイズについて

併合処理のメモリサイズについては，アドレス格納領域として使うサイズが AIX(64) COBOL2002，および Linux(x64) COBOL2002 では 8 バイトになります。併合処理で使用するメモリサイズの計算式については，「11.3.2　併合処理のメモリサイズ」を参照してください。

37.2　COBOL ソースの作成とコンパイル

AIX(64) COBOL2002，および Linux(x64) COBOL2002 での COBOL ソースのコンパイル方法は，AIX(32) COBOL2002 および Linux(x86) COBOL2002 と同じです。しかし，AIX(64) COBOL2002，および Linux(x64) COBOL2002 と AIX(32) COBOL2002 および Linux(x86) COBOL2002 では使用できるコンパイラオプションの一部が違います。

AIX(64) COBOL2002 および Linux(x64) COBOL2002 が使用できるコンパイラオプションについては，「32.5.4　コンパイラオプションの一覧」を参照してください。

37.3　プログラムの実行

ここでは，プログラムの実行について説明します。

37.3.1　プログラムの実行環境の設定

プログラムの実行環境の設定について説明します。

(1)　実行時環境変数の設定方法

AIX(64) COBOL2002 および Linux(x64) COBOL2002 での実行時環境変数の設定方法は，AIX(32) COBOL2002 および Linux(x86) COBOL2002 と同じです。詳細については，「35.3.1　実行時環境変数の設定方法」を参照してください。

(2)　実行時環境変数の一覧

AIX(64) COBOL2002 および Linux(x64) COBOL2002 と AIX(32) COBOL2002 および Linux(x86) COBOL2002 では使用できる実行時環境変数の一部が違います。AIX(64) COBOL2002 および Linux(x64) COBOL2002 が使用できる実行時環境変数については，「35.3.2　実行時環境変数の一覧」を参照してください。

37.3.2　プログラムの実行時の注意事項

AIX(64) COBOL2002 および Linux(x64) COBOL2002 でのプログラム実行時の注意事項を次に示します。

Linux(x64)の場合

Linux(x64) COBOL2002 で作成したアプリケーションから CALL 文を使用して，Linux(x86) COBOL2002 で作成した実行可能ファイルを呼び出せます※。ただし，Linux(x64) COBOL2002 で作成したアプリケーションからは Linux(x86) COBOL2002 で作成した共用ライブラリに含まれるプログラムは呼び出せません。

注※

Linux(x64) COBOL2002 と Linux(x86) COBOL2002 がインストールされた環境で実行してください。

AIX(64)の場合

AIX(64) COBOL2002 で作成したアプリケーションから CALL 文を使用して，AIX(32) COBOL2002 で作成した実行可能ファイルを呼び出せます※。ただし，AIX(64) COBOL2002 で作成したアプリケーションからは AIX(32) COBOL2002 で作成した共用ライブラリに含まれるプログラムは呼び出せません。

注※

AIX(64) COBOL2002 と AIX(32) COBOL2002 がインストールされた環境で実行してください。

37.4　実行可能ファイルと共用ライブラリの作成

AIX(64) COBOL2002 および Linux(x64) COBOL2002 での実行可能ファイルと共用ライブラリの作成の詳細については，「34.　実行可能ファイルと共用ライブラリの作成」を参照してください。

37.4.1　実行可能ファイルと共用ライブラリの作成時の注意事項

AIX(64) COBOL2002 および Linux(x64) COBOL2002 での実行可能ファイルと共用ライブラリの作成時の注意事項を次に示します。

- 他システムの COBOL2002 で作成したファイル（オブジェクト，共用ライブラリ，アーカイブ）をリンクすることはできません。

38 Linux(x86) COBOL2002，Linux(x64) COBOL2002 での UTF-8 ロケールの対応

この章では，Linux(x86) COBOL2002，Linux(x64) COBOL2002 を使用する場合の注意事項について説明します。

38.1　Linux(x86) COBOL2002，Linux(x64) COBOL2002 について

ここでは，Linux(x86) COBOL2002，Linux(x64) COBOL2002 の動作環境，使用できる機能，使用できるサービスルーチンについて説明します。

38.1.1　動作環境

Linux(x86) COBOL2002，Linux(x64) COBOL2002 では，UTF-8 ロケールだけをサポートしています。

UTF-8 ロケールで動作する場合，Unicode 機能を前提機能としています。このため，使用できる機能，日本語項目，および多バイト文字の使用範囲に制限があります。

UTF-8 ロケールの詳細については，「付録 A.4　Unicode の場合」の「(1)　UTF-8 の動作環境（Linux で有効）」を参照してください。Unicode 機能の詳細については，「27.　Unicode 機能」を参照してください。

38.1.2　使用できる機能

Linux(x86) COBOL2002，Linux(x64) COBOL2002 で使用できる機能については，「27.5　Unicode に対応する機能」を参照してください。

38.1.3　使用できるサービスルーチン

使用できるサービスルーチンに制限があります。

使用できるサービスルーチンについては，次の章を参照してください。

サービスルーチン

「29.　サービスルーチン」を参照してください。

38.2　コンパイル時の注意事項

Linux(x86) COBOL2002，Linux(x64) COBOL2002 でコンパイルするときの注意事項を次に示します。

- シフト JIS で記述された COBOL ソースプログラムだけがコンパイルできます。UTF-8 で記述された COBOL ソースはコンパイルできません。
- コンパイル時には，-UniObjGen オプションおよび環境変数 CBLSRCENCODING に SJIS を指定しなければなりません。

  -UniObjGen オプションの詳細は，「32.5　コンパイラオプション」の「32.5.14　その他の設定」の「(17)　-UniObjGen オプション」を参照してください。コンパイラ環境変数 CBLSRCENCODING の詳細は，「32.6.3　コンパイラ環境変数の詳細」の「(14)　CBLSRCENCODING（Linux で有効）」を参照してください。
- ccbl コマンドは使用できません。ccbl2002 コマンドを使用してください。
- 日本語文字，半角かたかななど，UTF-8 で多バイトとなる文字は，言語仕様上，使用できる個所に制限があります。詳細は，「27.7.2　コンパイル時の制限事項」を参照してください。
- 環境変数名，環境変数に設定する値，コマンドライン引数，コンパイル時に使用するファイル名，ディレクトリの絶対パス名，およびコンパイルを実行するカレントディレクトリの絶対パス名には，UTF-8 で多バイトとなる文字は使用できません。詳細は，「27.7.2　コンパイル時の制限事項」を参照してください。
- Linux(x86) COBOL2002，Linux(x64) COBOL2002 で使用できるコンパイラオプションについては，「32.5.4　コンパイラオプションの一覧」を参照してください。また，-UniObjGen オプションと同時に指定した場合に無効となるオプションについては，「32.5.3　コンパイラオプションの優先順位」の「(2)　オプション間の優先順位」の「(b)　オプションを指定すると，ほかのオプションが無効となる場合」を参照してください。
- Linux(x86) COBOL2002，Linux(x64) COBOL2002 で使用できるコンパイラ環境変数については，「32.6.2　コンパイラ環境変数の一覧」を参照してください。
- コンパイルリストはシフト JIS で出力されます。詳細は，「27.6　入出力情報と取り扱う文字コード」を参照してください。
- コンソールに出力するエラーメッセージは UTF-8 で出力されます。詳細は，「27.6　入出力情報と取り扱う文字コード」を参照してください。

38.3　実行可能ファイルと共用ライブラリの作成時の注意事項

Linux(x86) COBOL2002，Linux(x64) COBOL2002 で，実行可能ファイルと共用ライブラリを作成するときの注意事項を次に示します。

- Linux(x86) COBOL85 で作成したファイル（オブジェクト，共用ライブラリ，アーカイブ）をリンクすることはできません。COBOL2002 環境下で再コンパイルが必要です。
- 実行可能ファイル名および共用ライブラリ名に UTF-8 で多バイトとなる文字を使用しないでください。使用した場合の動作は保証しません。

38.4　実行時の注意事項

Linux(x86) COBOL2002，Linux(x64) COBOL2002 の実行時の注意事項を次に示します。

- Linux(x86) COBOL85 で作成した実行可能ファイル，または共用ライブラリは，COBOL2002 環境下で実行および呼び出すことはできません。COBOL2002 環境下で，COBOL85 で作成したプログラムが実行または呼び出された場合，KCCC0117R-S のメッセージが出力されます。
- 実行時には，環境変数 CBLLANG に UNICODE を指定しなければなりません。環境変数 CBLLANG などの Unicode 機能で使用する環境変数については，「27.　Unicode 機能」の「27.4.2　実行」を参照してください。
- 日本語文字，半角かたかななど，UTF-8 で多バイトとなる文字は，ファイル名に使用できないなどの制限があります。詳細は，「27.7.3　実行時の制限事項」を参照してください。

39 Linux コンテナの構築（Linux(x64)で有効）

この章では，Linux(x64)で Linux コンテナイメージを作成する手順，および Linux コンテナ使用時の注意事項について説明します。

39.1 Linux コンテナについて

39.1.1 動作環境

- Docker 環境に対応しています。Docker イメージに COBOL2002 を組み込み，コンテナで動作させることができます。
- Podman で作成するコンテナに対応しています。ホスト OS に podman-docker パッケージをインストールし，Podman に実装された Docker コマンドラインを実行することで，コンテナイメージに COBOL2002 を組み込み，コンテナで動作させることができます。Podman に実装された Docker コマンドラインを実行して作成するコンテナイメージを，Docker イメージといいます。
- COBOL2002 とコンテナのベースイメージの組み合わせ，およびコンテナのベースイメージで必要なパッケージについては，「リリースノート」でご確認ください。

39.2 Linux コンテナイメージの作成手順

Linux コンテナイメージ作成時には Dockerfile を使用します。Dockerfile を作成するディレクトリを Dockerfile 格納ディレクトリと表記します。

39.2.1 COBOL2002 を Linux コンテナに取り込む方法

ここでは，COBOL2002 を Docker イメージに取り込む手順を例に説明します。COBOL2002 を Docker イメージに取り込むために Dockerfile を作成します。

1. Dockerfile 格納ディレクトリ下に作業ディレクトリを作成し，その作業ディレクトリに製品媒体の内容をコピーします。

   この例では，Dockerfile 格納ディレクトリの作業ディレクトリを「media」とします。

   ```
   [/]# mount /dev/cdrom <ホストのマウントディレクトリ>
   [/]# cp -a <ホストのマウントディレクトリ> <Dockerfile格納ディレクトリ>/media
   ```

   なお，コピーしたファイル群は，コンペアなどを実施してバイナリレベルで一致することを確認してください。

2. ローカル環境に yum レポジトリサーバを設定しているシステムの場合は，Dockerfile 格納ディレクトリにレポジトリファイルを作成します。

3. COBOL2002 を Docker イメージに取り込むための Dockerfile を作成します。

   この例では，Linux(x64) COBOL2002（形名：P-9W36-1241）を Docker イメージに取り込みます。

   Dockerfile の書式については，Dockerfile リファレンスまたは man dockerfile の記載を参照してください。

   ```
   # ----- Dockerfile記述例 -----
   FROM rhel7.3:latest …1.
   COPY local.repo /etc/yum.repos.d/ …2.
   RUN yum -y install cpio.x86_64 …3.
   RUN yum -y install cpp gcc glibc.i686 glibc-devel glibc-headers kernel-headers ¥
   libgcc.i686 mpfr nss-softokn-freebl ¥
   && yum clean all ¥
   && rm -rf /var/cache/yum/* …4.
   RUN mkdir /tmp/COBOL2002 …5.
   COPY media /tmp/COBOL2002 …6.
   WORKDIR /tmp/COBOL2002 …7.
   RUN chmod +x ./X64LIN/setup …8.
   RUN ./X64LIN/setup -f -k P-9W36-1241 ./ …9.
   ENV PATH $PATH:/opt/HILNGcbl2k64/bin:/usr/bin:/bin:/opt/HIISlib64/bin …10.
   ENV LD_LIBRARY_PATH $LD_LIBRARY_PATH:/opt/HILNGcbl2k64/lib:/opt/HIISlib64/lib:¥
   /opt/HISORTlib64/lib:/opt/HILNGcbl2k64/lib/cblxml …10.
   ENV NLSPATH $NLSPATH:/opt/HIISlib64/%L/%N.cat:/opt/HISORTlib64/%L/%N.cat:¥
   /opt/HILNGcbl2k64/lib/cblxml/cat/%L/%N …10.
   WORKDIR / …11.
   RUN rm -rf /tmp/COBOL2002 …12.
   CMD /bin/bash …13.
   ```

説明

1. ベースとなる Docker のイメージを指定します。
2. 事前に作成したレポジトリファイルをコンテナにコピーします。
3. コンテナのベースイメージに UBI を使用します（UBI をベースイメージとして使用しない場合，設定は不要です）。
4. 事前に，Docker イメージに取り込む COBOL2002 の前提パッケージをインストールします。前提パッケージについては，「リリースノート」でご確認ください。
5. コンテナに作業ディレクトリを作成します。この例では，コンテナの作業ディレクトリを「/tmp/COBOL2002」とします。
6. 手順 1.で Dockerfile 格納ディレクトリの作業ディレクトリにコピーした製品媒体の内容を，コンテナの作業ディレクトリにコピーします。
7. カレントディレクトリをコンテナの作業ディレクトリに位置づけます。
8. COBOL2002 セットアッププログラムに実行権限を付与します。
9. COBOL2002 のインストーラを使用して，コンテナに COBOL2002 をインストールします。
10. COBOL2002 を使用するために必要な環境変数を指定します。詳細は「リリースノート」でご確認ください。
11. カレントディレクトリをルートディレクトリに位置づけます。
12. コンテナの作業ディレクトリを削除します。
13. コンテナ起動時に実行するコマンドを指定します。

4. 手順 3.で作成した Dockerfile を指定して docker build コマンドを実行します。

この例では，Dockerfile 格納ディレクトリをカレントディレクトリに位置づけてからコマンドを実行しています。

```
[/]# cd <Dockerfile格納ディレクトリ>
[<Dockerfile格納ディレクトリ>]# ls
Dockerfile
local.repo
media
[<Dockerfile格納ディレクトリ>]# docker build -t <Dockerイメージ名> ./
```

Docker イメージが Dockerfile 格納ディレクトリに作成されます。

39.2.2　COBOL プログラムを Linux コンテナ起動時に実行する方法

ここでは，COBOL プログラムを Docker イメージに取り込んで，コンテナ起動時に実行する手順を例に説明します。

COBOL プログラムをコンテナ起動時に実行する場合は，「39.2.1　COBOL2002 を Linux コンテナに取り込む方法」で作成した Dockerfile に，COBOL プログラムを Docker イメージに取り込む記述を追加して，コンテナ起動時に COBOL プログラムを実行するように記述を変更する必要があります。

1. Dockerfile 格納ディレクトリ下に作業ディレクトリを作成し，その作業ディレクトリに COBOL プログラムを格納します。

   この例では，Dockerfile 格納ディレクトリの作業ディレクトリを「sample」，COBOL プログラムを「sample.out」とします。

```
[/]# cd <Dockerfile格納ディレクトリ>/sample
[<Dockerfile格納ディレクトリ>/sample]# ls
sample.out
```

2. COBOL プログラムを Docker イメージに取り込んで，コンテナ起動時に実行するための Dockerfile を作成します。

   この例では，「39.2.1　COBOL2002 を Linux コンテナに取り込む方法」の手順 3.で作成した Dockerfile を基に作成します。

   次に示す処理の記述個所は，**太字（色付き）**で示します。

   - COBOL プログラムを Docker イメージに取り込む処理
   - コンテナ起動時に COBOL プログラムを実行する処理

```
# ----- Dockerfile記述例 -----
FROM rhel7.3:latest
COPY local.repo /etc/yum.repos.d/
RUN yum -y install cpio.x86_64
RUN yum -y install cpp gcc glibc.i686 glibc-devel glibc-headers kernel-headers ¥
libgcc.i686 mpfr nss-softokn-freebl ¥
&& yum clean all ¥
&& rm -rf /var/cache/yum/*
RUN mkdir /tmp/COBOL2002
COPY media /tmp/COBOL2002
WORKDIR /tmp/COBOL2002
RUN chmod +x ./X64LIN/setup
RUN ./X64LIN/setup -f -k P-9W36-1241 ./ …1.
ENV PATH $PATH:/opt/HILNGcbl2k64/bin:/usr/bin:/bin:/opt/HIISlib64/bin
ENV LD_LIBRARY_PATH $LD_LIBRARY_PATH:/opt/HILNGcbl2k64/lib:/opt/HIISlib64/lib:¥
/opt/HISORTlib64/lib:/opt/HILNGcbl2k64/lib/cblxml
ENV NLSPATH $NLSPATH:/opt/HIISlib64/%L/%N.cat:/opt/HISORTlib64/%L/%N.cat:¥
/opt/HILNGcbl2k64/lib/cblxml/cat/%L/%N
WORKDIR /
RUN rm -rf /tmp/COBOL2002
RUN mkdir /tmp/sample …2.
COPY sample /tmp/sample …3.
WORKDIR /tmp/sample …4.
CMD ./sample.out …5.
```

   説明

   1. COBOL プログラムをコンテナ起動時に実行する場合は，コンテナにインストールする COBOL2002 として，実行環境とシステム運用環境を提供するプログラムプロダクト（COBOL2002 Net Server Runtime(64)）を指定できます。

   2. コンテナに作業ディレクトリを作成します。この例では，コンテナの作業ディレクトリを「/tmp/sample」とします。

3. 手順 1.で Dockerfile 格納ディレクトリの作業ディレクトリにコピーした COBOL プログラムを，コンテナの作業ディレクトリにコピーします。

4. カレントディレクトリをコンテナの作業ディレクトリに位置づけます。

5. コンテナ起動時に実行する COBOL プログラムを指定します。

3. 手順 2.で作成した Dockerfile を指定して docker build コマンドを実行します。

この例では，Dockerfile 格納ディレクトリをカレントディレクトリに位置づけてからコマンドを実行しています。

```
[/]# cd <Dockerfile格納ディレクトリ>
[<Dockerfile格納ディレクトリ>]# ls
Dockerfile
local.repo
media
sample
[<Dockerfile格納ディレクトリ>]# docker build -t <Dockerイメージ名> ./
```

Docker イメージが Dockerfile 格納ディレクトリに作成されます。

39.2.3 Linux コンテナの動作ロケールの設定例

ここでは，コンテナの動作ロケールを設定する Dockerfile を例に説明します。

ここで示す記述を「39.2.1 COBOL2002 を Linux コンテナに取り込む方法」や，「39.2.2 COBOL プログラムを Linux コンテナ起動時に実行する方法」で作成する Dockerfile に追加することで，コンテナの動作ロケールを設定できます。例では，動作ロケールに「UTF-8」を設定しています。

```
RUN localedef -f UTF-8 -i ja_JP /usr/lib/locale/ja_JP.utf8
ENV LANG UTF-8
```

39.2.4 Linux コンテナでの COBOL2002 の使用例

ここでは，ホスト上のファイルを入出力する COBOL プログラムを Docker イメージに取り込んで，コンテナ起動時に実行する手順を例に説明します。

この使用例では，コンテナを起動すると，COBOL プログラムが実行されて，ホスト上のファイルのデータが，ホスト上の別のファイルへ書き出されるという一連の処理を実現できます。

(1) COBOL プログラムの作成

COBOL プログラムの作成例を次に示します。

この COBOL プログラムでは，入力ファイルからデータを読み込み，読み込んだデータを出力ファイルへ書き出します。この例では，実行可能ファイルを「sample.out」とします。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT FILE-1 ASSIGN TO SYS010. …1.
SELECT FILE-2 ASSIGN TO SYS020. …2.
DATA DIVISION.
FILE SECTION.
FD FILE-1.
01 FILE-1-REC PIC X(30).
FD FILE-2.
01 FILE-2-REC PIC X(30).
PROCEDURE DIVISION.
OPEN INPUT FILE-1
OUTPUT FILE-2.
READ FILE-1.
MOVE FILE-1-REC TO FILE-2-REC.
WRITE FILE-2-REC.
CLOSE FILE-1
FILE-2.
STOP RUN.
END PROGRAM SAMPLE1.
```

説明

1. COBOL プログラムの入力ファイルを指定します。

   ここでは，COBOL プログラム実行時に，環境変数 CBL_SYS010 に指定された物理ファイル名を入力ファイルとします。

2. COBOL プログラムの出力ファイルを指定します。

   ここでは，COBOL プログラム実行時に，環境変数 CBL_SYS020 に指定された物理ファイル名を出力ファイルとします。

(2)　COBOL プログラムの格納

Dockerfile 格納ディレクトリ下に作業ディレクトリを作成し，「(1)　COBOL プログラムの作成」で作成した COBOL プログラム（sample.out）を格納します。この例では，Dockerfile 格納ディレクトリの作業ディレクトリを「sample」とします。

```
[/]# cd <Dockerfile格納ディレクトリ>/sample
[<Dockerfile格納ディレクトリ>/sample]# ls
sample.out
```

(3)　Docker イメージの作成

「39.2.1　COBOL2002 を Linux コンテナに取り込む方法」に従って，ベースとなる Docker イメージを作成します。この例では，Docker イメージ名を「cobol2002serversuite:0460」とします。

(4)　Dockerfile の作成

COBOL プログラムを Docker イメージに取り込むための Dockerfile を作成します。また，COBOL プログラムからファイルにアクセスするために，環境変数で COBOL プログラムのファイル名に対して，物理ファイル名を割り当てます。

記述例を次に示します。

```
FROM cobol2002serversuite:0460 …1.
RUN mkdir /tmp/sample …2.
COPY sample /tmp/sample …3.
WORKDIR /tmp/sample …4.
RUN mkdir /tmp/mntdir …5.
ENV CBL_SYS010 /tmp/mntdir/file01 …6.
ENV CBL_SYS020 /tmp/mntdir/file02 …7.
CMD ./sample.out …8.
```

説明

1. ベースとなる Docker のイメージを指定します。この例では，「(3)　Docker イメージの作成」で作成した「cobol2002serversuite:0460」を指定しています。
2. コンテナに作業ディレクトリを作成します。この例では，コンテナの作業ディレクトリを「/tmp/sample」とします。
3. 「(2)　COBOL プログラムの格納」で Dockerfile 格納ディレクトリの作業ディレクトリにコピーした COBOL プログラムを，コンテナの作業ディレクトリにコピーします。
4. カレントディレクトリをコンテナの作業ディレクトリに位置づけます。
5. ホスト側のディレクトリをマウントするコンテナ側のディレクトリを作成します。コンテナは，ホスト側のディレクトリをマウントして起動します。この例では，ホスト側のディレクトリを「/tmp/hostdir」，コンテナ側のディレクトリを「/tmp/mntdir」とします。
6. 環境変数 CBL_SYS010 に物理ファイル名（/tmp/mntdir/file01）を指定して，COBOL プログラムで使用する入力ファイルを割り当てます。コンテナを起動する前に，コンテナ側のディレクトリにマウントされるホスト側のディレクトリ（/tmp/hostdir）に，ファイル（file01）を作成しておいてください。
7. 環境変数 CBL_SYS020 に物理ファイル名（/tmp/mntdir/file02）を指定して，COBOL プログラムで使用する出力ファイルを割り当てます。
8. コンテナ起動時に実行する COBOL プログラムを指定します。

(5)　COBOL プログラムの Docker イメージへの取り込み

「(4)　Dockerfile の作成」で作成した Dockerfile を指定して，docker build コマンドを実行します。この例では，Docker イメージ名を「cobol2002_sample:0460」とし，Dockerfile 格納ディレクトリをカレントディレクトリに位置づけてからコマンドを実行しています。

```
[/]# cd <Dockerfile格納ディレクトリ>
[<Dockerfile格納ディレクトリ>]# ls
Dockerfile
local.repo
media
```

COBOL プログラムを実行する Docker イメージが，Dockerfile 格納ディレクトリに作成されます。

(6) コンテナの起動

docker run コマンドで，ホスト側のディレクトリをコンテナ側のディレクトリにマウントして，コンテナを起動します。この例では，ホスト側ディレクトリを「/tmp/hostdir」，コンテナ側ディレクトリを「/tmp/mntdir」とします。また，Docker イメージ名を「(5) COBOL プログラムの Docker イメージへの取り込み」で作成した「cobol2002_sample:0460」とします。各オプションについては，Docker のドキュメントを参照してください。

```
[/]#docker run --name <コンテナの名前> --mount type=bind,source=/tmp/hostdir,destination=/tmp/mntdir cobol2002_sample:0460
```

コンテナを起動すると，COBOL2002 プログラム（sample.out）が実行され，ホスト上のファイル（/tmp/hostdir/file01）のデータが，ホスト上の別のファイル（/tmp/hostdir/file02）へ書き出されます。

39.3 Linux コンテナ使用時の注意事項

COBOL2002 をコンテナとして起動して，テストデバッグ機能またはカバレージ機能（カバレージ情報とカウント情報を含む）を使用する場合は，コンテナ起動時，ケーパビリティに「SYS_PTRACE」を追加してください。

ケーパビリティに「SYS_PTRACE」を追加すると，動作に次のような影響があります。本番環境で使用する場合は，十分に影響を考慮した上で，ケーパビリティに「SYS_PTRACE」を追加してください。

- 関数をトレースするため，実行性能が劣化するおそれがあります。
- 関数をトレースするため，引数がトレースの結果として出力されることがあります。

付録

付録 A　COBOL で使用する文字集合

ここでは，COBOL で使用する文字集合について説明します。

付録 A.1　概要

このシステムでは，「計算機コード化文字集合の国別文字集合」および「COBOL 文字集合における拡張文字」の表現形式として次の文字を使用できます。

- シフト JIS
- 日本語 EUC
- Unicode（ただし，Unicode 機能を使用する場合。COBOL 文字集合には使用できない。）

COBOL で使用する文字集合の分類と定義に関する詳細については，マニュアル「COBOL2002　言語標準仕様編」「4.1　文字集合」を参照してください。

各システムで使用できる動作環境および文字コードについて，次に示します。

(1)　動作環境と文字コード

環境変数 LANG の設定値と，コンパイル，実行の動作環境，および COBOL で使用する文字コードを次に示します。

表 A-1　動作環境と文字コード

システム	環境変数 LANG の設定値	動作環境（ロケール）	文字コード
AIX	Ja_JP	シフト JIS	シフト JIS Unicode※
	ja_JP	日本語 EUC	日本語 EUC
Linux	ja_JP.UTF-8 ja_JP.utf8	UTF-8	Unicode※

注※

Unicode 機能を使用することで，Unicode データを扱うことができます。詳細については，「付録 A.4　Unicode の場合」を参照してください。

(2)　文字コードに関する注意事項

(a)　環境変数 LANG の扱い

環境変数 LANG の設定値について詳しくは，「付録 A.2　シフト JIS の場合」，「付録 A.3　EUC の場合」，および「付録 A.4　Unicode の場合」を参照してください。

環境変数 LANG に値を指定しなかった場合，または表中の環境変数 LANG の値以外を指定した場合，エラーメッセージは英語で表示されます。

日本語項目に対する表意定数の値

日本語項目に対する表意定数の値は，次のようになります。

シフト JIS コード，および PCK コードの場合

SPACE：$(8140)_{16}$

ZERO：$(824F)_{16}$

日本語 EUC コードの場合

SPACE：$(A1A1)_{16}$

ZERO：$(A3B0)_{16}$

Unicode の場合

Unicode 機能を使用した場合の表意定数については，「27.5　Unicode に対応する機能」の「27.5.1　基本機能」を参照してください。

付録 A.2　シフト JIS の場合

環境変数 LANG にシフト JIS に対応した設定値が設定された環境（シフト JIS 環境）では，コンパイル時および実行時の動作環境がシフト JIS となります。シフト JIS 環境では，シフト JIS コードで記述された COBOL ソースをコンパイルおよび実行できます。シフト JIS に対応した環境変数 LANG の設定値を次に示します。

表 A–2　動作環境がシフト JIS となる環境変数 LANG の設定値

OS	環境変数 LANG の設定値
AIX	Ja_JP
Linux	設定できません。

シフト JIS 環境で作成した COBOL プログラム（オブジェクト，アーカイブライブラリ，共用ライブラリ，または実行可能ファイル）は，シフト JIS 環境で実行する必要があります。また，ほかの環境で作成した COBOL プログラムと混在して使用することはできません。シフト JIS 環境以外で実行した場合，およびほかの環境で作成した COBOL プログラムと混在して使用した場合は，動作は保証しません。

付録 A.3　EUC の場合

(1)　EUC コードを使用した COBOL プログラム

(a)　使用方法

環境変数 LANG に，日本語 EUC に対応した設定値が設定された環境（日本語 EUC 環境）では，コンパイル時および実行時の動作環境が日本語 EUC となります。日本語 EUC 環境では，日本語 EUC コードで記述された COBOL ソースをコンパイルおよび実行できます。日本語 EUC に対応した環境変数 LANG の設定値を次に示します。

表 A-3　動作環境が日本語 EUC となる環境変数 LANG の設定値

OS	環境変数 LANG の設定値
AIX	ja_JP
Linux	設定できません。

日本語 EUC 環境で作成した COBOL プログラム（オブジェクト，アーカイブライブラリ，共用ライブラリ，または実行可能ファイル）は，日本語 EUC 環境で実行する必要があります。また，ほかの環境で作成した COBOL プログラムと混在して使用することはできません。日本語 EUC 環境以外で実行した場合，およびほかの環境で作成した COBOL プログラムと混在して使用した場合は，動作は保証しません。

(b)　注意事項

次の内容に反した場合の結果は保証しません。また，エラーチェックもその旨の記述がないときは行われません。

- 半角かたかな文字は，2 バイト文字となります。そのため，半角かたかな文字がある文字定数と関連する定義エリアは 2 バイト分必要となります。

  （例）

  ```
  01  TEST-DATA    PIC X(8) VALUE 'ｱｲｳAB'.
  ```

  - 文字定数に関する既存のエラーチェックも半角かたかな 1 文字を 2 バイトとして行います。
  - 半角かたかな文字は，日本語定数の中に含めることはできません。日本語定数の中に半角かたかな文字を指定した場合，コンパイルエラーとなります。
  - 半角かたかな文字は，2 バイトで 1 文字しか印刷されません。そのため，後ろのデータ項目の印刷位置が半角かたかな文字の文字数分だけずれます。
  - 固定形式正書法では 72 バイト目までが翻訳の対象とされ，73 バイト目以降の記述は無視されます。シフト JIS 環境で作成された半角かな文字を含む原始プログラムを EUC コードに変換して使用すると，半角かな文字の部分が右側にずれます。73 バイト目以降にずれた部分は，翻訳の対象とみなされません。
- 利用者語およびデータ項目として，3 バイトの外字は使用できません。

- 外字，機種依存文字およびシフト JIS コードの 2 バイト文字データ，シフト JIS コードの半角かたかな文字のデータ自体をプログラム中で使用する場合は，16 進文字定数または 16 進日本語文字定数で指定してください。
- 2 バイト文字，半角かたかな文字の COBOL ソースファイルファイル名，ディレクトリ名は使用できません。コード体系の相違から問題が発生する場合があります。
- -Switch,EBCDIK／-Switch,EBCDIC オプション指定時，または PROGRAM COLLATING SEQUENCE 句の指定時に，比較条件に半角かたかな文字を含む ALL 文字定数を指定した場合の動作は保証しません。

(2) -EucPosition オプション（AIX で有効）

EUC コード使用時に見かけ上の文字位置で，固定形式正書法の境界を決定するときに指定します。このオプションを指定した場合，半角かたかな文字は 2 バイトとして扱われます。しかし，固定形式正書法の境界を決定するときには，見かけ上の 1 バイトとして扱われます。

（例）

```
0         10        20        30        40        50        60        70
12345678901234567890123456789012345678901234567890123456789012345678901234567890
-----+-*--|----+----|----+----|----+----|----+----|----+----|----+----|----+----|
001200       03  TEST-DATA-001                PIC X(10)       VALUE 'ｱｲｳｴｵ'.
```

- この COBOL ソースファイルが EUC コードで書かれていた場合，コンパイルエラーとなります。
- -EucPosition オプションを指定すると，このままコンパイルできます。ただし，この場合でも PIC X(5)ではなく，PIC X(10)にする必要があります。

(a) 使用方法

コンパイル時に，-EucPosition オプションを指定します。

(b) -EucPosition オプション指定時の注意事項

- -EucPosition オプションの対象となる COBOL ソースファイルが EUC コードでない場，その結果は保証しません。
- 環境変数 LANG に，「表 A-3　動作環境が日本語 EUC となる環境変数 LANG の設定値」に示す設定値が指定されていない環境で-EucPosition オプションが指定されたとき，-EucPosition オプションは無視されます。
- 自由形式正書法の COBOL ソースファイルに対して，-EucPosition オプションは指定できません。自由形式正書法の COBOL ソースファイルに-EucPosition オプションを指定した場合は，コンパイルエラーとなります。自由形式正書法とオプションとの関係については，「32.2.3　正書法」の「(4)　自由形式正書法で指定できないコンパイラオプション」を参照してください。

(3)　組み込み関数を使用した文字列操作

(a)　日本語 EUC 環境に対応した組み込み関数

日本語 EUC の文字操作や文字数またはけた数の取得ができる組み込み関数を次に示します。組み込み関数については，マニュアル「COBOL2002　言語　拡張仕様編」「25.5　組み込み関数」を参照してください。

表 A-4　日本語 EUC 環境に対応した組み込み関数

用途	組み込み関数名	概要
文字列の取り出し	SUBSTRING	字類が英字，英数字または日本語のデータ項目から部分文字列を返します。
文字数の取得	COUNT-CHAR	字類が英字，英数字または日本語のデータ項目の文字数を返します。
けた数の取得	LENGTH-OF-SUBSTRING	字類が英字，英数字または日本語のデータ項目の部分文字列のけた数を返します。

(b)　用語説明

使用する用語の説明を次に示します。

表 A-5　日本語 EUC 環境に対応した組み込み関数で使用する用語説明

用語	説明	COBOL2002 での用途
見た目幅	半角文字を 1，全角文字を 2 とした見た目の幅	Unicode 機能や日本語 EUC 環境の組み込み関数で，文字を見た目幅で数える場合に使用します。

(c)　日本語 EUC 環境に対応した組み込み関数の注意事項

- SUBSTRING，COUNT-CHAR，LENGTH-OF-SUBSTRING 組み込み関数は，部分参照や LENGTH 関数と比較して，文字の判定が必要になるため，実行時間が増加します。そのため，次の場合，部分参照の使用を推奨します。
  - 半角かたかなや全角文字が格納されない英数字項目など，1 文字のサイズが可変となる可能性がないデータ項目を参照する場合，部分参照を使用する。
  - 文字列の取り出しで，けた数と文字数の両方がわかっている場合，けた数による部分参照を使用する。
- 日本語 EUC 環境でコンパイルした SUBSTRING，COUNT-CHAR，LENGTH-OF-SUBSTRING 組み込み関数を使用するプログラムを日本語 EUC 環境以外で実行した場合，実行時エラーとなり，プログラムの実行を中止します。
- 実行時に引数をチェックして，引数誤りがある場合は，EC-ARGUMENT-FUNCTION 例外が発生します。

  日本語 EUC 環境に対応した組み込み関数で，EC-ARGUMENT-FUNCTION 例外が発生する引数誤りの要因を次に示します。

表 A–6　日本語 EUC 環境に対応した組み込み関数の引数誤りの要因

組み込み関数名	引数誤りの要因
COUNT-CHAR	• 引数 1 の文字コードが不当な値である。※ • 引数 1 が日本語項目の場合に奇数バイトの文字が混在している。
LENGTH-OF-SUBSTRING	• 引数 1 の文字コードが不当な値である。※ • 引数 1 が日本語項目の場合に奇数バイトの文字が混在している。 • 引数 2 と引数 3 が一意名または算術式で，値が正の整数でない。 • 引数 2 で指定する開始位置，または引数 2 で指定する開始位置から引数 3 で指定する長さだけ進めた文字位置が引数 1 の終端を超えている。 • 引数 2 で指定する開始位置，または引数 2 で指定する開始位置から引数 3 で指定する長さだけ進めた文字位置が全角文字の途中である。
SUBSTRING	• 引数 1 の文字コードが不当な値である。※ • 引数 1 が日本語項目の場合に奇数バイトの文字が混在している。 • 引数 2 と引数 3 が一意名または算術式で，値が正の整数でない。 • 引数 2 で指定する開始位置，または引数 2 で指定する開始位置から引数 3 で指定する長さだけ進めた文字位置が引数 1 の終端を超えている。 • 引数 2 で指定する開始位置，または引数 2 で指定する開始位置から引数 3 で指定する長さだけ進めた文字位置が全角文字の途中である。

注※

文字コードが不当な値となっている記述例を次に示します。

```
（例）
01 DATA1 PIC X(20).
VALUE X'8EB18EB28EB38EB48EF58EB68EB78EB88EB98EBA'.
*> ｱｲｳｴｵｶｷｸｹｺと設定したいが，10ﾊﾞｲﾄ目が不正(B5 ではなく F5 になっている
*> この位置に F5 があるのは日本語 EUC として不正な文字と扱う)
MOVE FUNCTION SUBSTRING(DATA1, 2, 4) TO DATA2.
```

付録 A.4　Unicode の場合

Unicode については，「27.　Unicode 機能」を参照してください。

(1)　UTF-8 の動作環境（Linux で有効）

環境変数 LANG に UTF-8 に対応した設定値が設定された環境（UTF-8 環境）では，コンパイル時および実行時の動作環境が UTF-8 となります。UTF-8 環境では，シフト JIS コードで記述された COBOL ソースをコンパイルおよび実行できます。UTF-8 に対応した環境変数 LANG の設定値を次に示します。

表 A–7　動作環境が UTF-8 となる環境変数 LANG の設定値

OS	環境変数 LANG の設定値
Linux	ja_JP.UTF-8

OS	環境変数 LANG の設定値
	ja_JP.utf8

注意事項

- Unicode 機能を使用する必要があります。
- UTF-8 環境で作成した COBOL プログラム（オブジェクト，アーカイブライブラリ，共用ライブラリ，または実行可能ファイル）は，UTF-8 環境で実行する必要があります。また，ほかの環境で作成した COBOL プログラムと混在して使用することはできません。UTF-8 環境以外で実行した場合，およびほかの環境で作成した COBOL プログラムと混在して使用した場合は，動作は保証しません。
- UTF-8 環境では，Unicode で記述された COBOL ソースはコンパイルおよび実行できません。
- ccbl コマンドは使用できません。

(2)　Unicode データを扱う COBOL プログラム

COBOL2002 では，Unicode 機能を使用して Unicode のデータを扱う COBOL プログラムをコンパイルおよび実行できます。Unicode 機能の詳細は，「27.　Unicode 機能」を参照してください。

付録 B　COBOL85 および旧バージョンの COBOL2002 からの移行性と互換性

ここでは，COBOL85 製品および旧バージョンの COBOL2002 からの移行性と互換性について説明します。

この節での製品の表記を次の表に示します。

表 B-1　製品の表記

この節での表記	製品
COBOL85 製品	COBOL85 OOCOBOL COBOL85 版 Cosminexus 連携 COBOL85 版 XML 連携
COBOL85	COBOL85 COBOL85 Run Time System
COBOL2002 V3 以前	COBOL2002（バージョン 04-00 未満）
COBOL2002 V4	COBOL2002（バージョン 04-00 以降）

付録 B.1　COBOL2002 V4 への移行性と互換性

COBOL2002 V4 への移行性と互換性について説明します。

AIX(32) COBOL2002 または Linux(x86) COBOL2002 から AIX(64) COBOL2002 または Linux(x64) COBOL2002 への移行時には，移行前に 32bit アプリケーションと 64bit アプリケーションの相違について確認してください。64bit アプリケーションについては，「37.　64bit アプリケーションの作成」を参照してください。

(1)　COBOL85，COBOL2002 の旧バージョンで作成したファイルの移行性

COBOL85 および COBOL2002 の旧バージョンで作成したファイルの COBOL2002 V4 への移行性を次に示します。

(a)　AIX(64) COBOL2002 および AIX(32) COBOL2002 への移行性

AIX(64) COBOL2002 V4 への移行性を次の表に示します。

表 B-2　AIX(64) COBOL2002 V4 への移行性

対象ファイル		移行元システム		
ファイル種別	拡張子	AIX(64) COBOL2002	AIX(32) COBOL2002	COBOL85
COBOL ソースファイル	.cbl ほか	○	○	○
開発環境資産（makefile，コンパイル・リンケージするシェルプログラム）	拡張子規定なし	△	△	△
実行可能ファイル	拡張子規定なし	△	×	×
オブジェクトファイル	.o	△	×	×
アーカイブファイル	.a	△	×	×
共用ライブラリファイル（アーカイブ形式を含む）	.so，.a	△	×	×
プログラム情報ファイル	.cbp，.cbs，.pif	△	×	×
TD コマンド格納ファイル（テストコマンドファイル）	.tdi，.tds	○	○	△
Cosminexus 連携機能 Java 実行ファイル	.class	○	○※1	○
XML 連携機能関連ファイル	.xml，.cxd，.cxc	○	○	○
リポジトリファイル	.rep	○	×	−
画面機能リソースファイル※2	拡張子なし	○	×	×

（凡例）

○：移行できる

△：一部移行性あり（詳細は，「(2)　ファイルの移行性の注意事項」を参照してください）

×：移行できない

−：移行先または移行元で未サポートのファイル

注※1

AIX(32) COBOL2002 01-00 から 01-03 までの Cosminexus 連携機能でサポートしていたファイルです。

注※2

画面機能リソースファイルのファイル名は，それぞれ次のとおりです。

(a)アプリケーションリソースファイル名

・AIX(32) COBOL2002 の場合：Cbl2002term

・AIX(64) COBOL2002 の場合：Cbl2002term64

・COBOL85 の場合：Cbl85term

(b)JCPOPUP サービスルーチンリソースファイル名

・AIX(32) COBOL2002 の場合：Cbl2002popup

・AIX(64) COBOL2002 の場合：Cbl2002popup64

・COBOL85 の場合：Cbl85popup

AIX(32) COBOL2002 V4 への移行性を次の表に示します。

表 B-3　AIX(32) COBOL2002 V4 への移行性

対象ファイル		移行元システム	
ファイル種別	拡張子	AIX(32) COBOL2002	COBOL85
COBOL ソースファイル	.cbl ほか	○	○
開発環境資産（makefile，コンパイル・リンケージするシェルプログラム）	拡張子規定なし	△	△
実行可能ファイル	拡張子規定なし	△	×
オブジェクトファイル	.o	△	×
アーカイブファイル	.a	△	×
共用ライブラリファイル（アーカイブ形式を含む）	.so，.a	△	×
プログラム情報ファイル	.cbp，.cbs，.pif	△	×
TD コマンド格納ファイル（テストコマンドファイル）	.tdi，.tds	○	△
XML 連携機能関連ファイル	.xml，.cxd，.cxc	○	○
リポジトリファイル	.rep	○	−
画面機能リソースファイル※	拡張子なし	○	×

（凡例）

○：移行できる

△：一部移行性あり（詳細は，「(2)　ファイルの移行性の注意事項」を参照してください）

×：移行できない

−：移行先または移行元で未サポートのファイル

注※

画面機能リソースファイルのファイル名は，それぞれ次のとおりです。

(a)アプリケーションリソースファイル名

・AIX(32) COBOL2002 の場合：Cbl2002term

・COBOL85 の場合：Cbl85term

(b)JCPOPUP サービスルーチンリソースファイル名

・AIX(32) COBOL2002 の場合：Cbl2002popup

・COBOL85 の場合：Cbl85popup

(b)　Linux(x64) COBOL2002 および Linux(x86) COBOL2002 への移行性

Linux(x64) COBOL2002 V4 への移行性を次の表に示します。

表 B–4　Linux(x64) COBOL2002 V4 への移行性

対象ファイル		移行元システム		
ファイル種別	拡張子	Linux(x64) COBOL2002	Linux(x86) COBOL2002	COBOL85
COBOL ソースファイル	.cbl ほか	○	○	○
開発環境資産（makefile，コンパイル・リンケージするシェルプログラム）	拡張子規定なし	○	△	△
実行可能ファイル	拡張子規定なし	△	×	×
オブジェクトファイル	.o	○	×	×
アーカイブファイル	.a	○	×	×
共用ライブラリファイル	.so	△	×	×
プログラム情報ファイル	.cbp，.cbs，.pif	△	×	×
TD コマンド格納ファイル	.tdi，.tds	○	○	△
Cosminexus 連携機能 Java 実行ファイル	.class	○	○※	−
XML 連携機能関連ファイル	.xml，.cxd，.cxc	○	○	−
リポジトリファイル	.rep	○	×	−

（凡例）

○：移行できる

△：一部移行性あり（詳細は，「(2)　ファイルの移行性の注意事項」を参照してください）

×：移行できない

−：移行先または移行元で未サポートのファイル

注※

Linux(x86) COBOL2002 01-01 から 01-02 までの Cosminexus 連携機能でサポートしていたファイルです。

Linux(x86) COBOL2002 V4 への移行性を次の表に示します。

表 B–5　Linux(x86) COBOL2002 V4 への移行性

対象ファイル		移行元システム	
ファイル種別	拡張子	Linux(x86) COBOL2002	COBOL85
COBOL ソースファイル	.cbl ほか	○	○
開発環境資産（makefile，コンパイル・リンケージするシェルプログラム）	拡張子規定なし	△	△
実行可能ファイル	拡張子規定なし	△	×
オブジェクトファイル	.o	○	×

対象ファイル		移行元システム	
ファイル種別	拡張子	Linux(x86) COBOL2002	COBOL85
アーカイブファイル	.a	○	×
共用ライブラリファイル	.so	△	×
プログラム情報ファイル	.cbp，.cbs，.pif	△	×
TD コマンド格納ファイル	.tdi，.tds	○	△
XML 連携機能関連ファイル	.xml，.cxd，.cxc	○	−
リポジトリファイル	.rep	○	−

（凡例）

○：移行できる

△：一部移行性あり（詳細は，「(2)　ファイルの移行性の注意事項」を参照してください）

×：移行できない

−：移行先または移行元で未サポートのファイル

(2)　ファイルの移行性の注意事項

(a)　実行可能ファイルまたは共用ライブラリファイル（.so，.a）の移行での注意事項

- 実行可能ファイルまたは共用ライブラリファイルを作成時に，OS やほかのミドルウェアのオブジェクトやアーカイブライブラリをリンクしている場合，再リンクが必要になることがあります。OS やほかのミドルウェアの互換性を確認してください。
- AIX(32) COBOL2002 または Linux(x86) COBOL2002 からの移行で，DC シミュレーション機能を使用している（リンク時に-Bstatic(Linux の場合)，-bstatic(AIX の場合)に-lcbl2KDC オプションを指定して作成している）場合，再リンクする必要があります。

(b)　開発環境資産（makefile，コンパイル・リンケージするシェルプログラム）の移行での注意事項

- COBOL2002 のインストールディレクトリは COBOL85 と異なります。COBOL2002 実行時ライブラリをリンクするときの-L オプションの指定を変更してください。
- COBOL85，AIX(32) COBOL2002，または Linux(x86) COBOL2002 からの移行で DC シミュレーション機能を使用するプログラムのリンク時に-Bstatic(Linux の場合)，-bstatic(AIX の場合)に-lcbl2kdc オプションが指定されている場合，-Bdynamic(Linux の場合)，-bdynamic(AIX の場合)に指定されるように変更する必要があります。
- AIX COBOL2002 V4 への移行で，開発環境資産の ccbl2002 コマンド（または ccbl コマンド）の引数に，アーカイブファイルを-l オプションではなくファイル名で指定しているとき，AIX COBOL2002 V3 以前と同様にアーカイブファイルを取り込むためには，アーカイブファイルの指定をファイル名指定から-bkeepfile リンカオプションによる指定（-bkeepfile:アーカイブファイル名）に変更する必要があります。

- AIX COBOL2002 V4 では，COBOL プログラムのオブジェクト形式を，AIX COBOL2002 V3 以前の形式から変更しています。AIX COBOL2002 V4 で COBOL ソースファイルをリコンパイルして作成したオブジェクトファイルやアーカイブファイルを使用して，実行可能ファイルまたは共用ライブラリを作成する場合，COBOL プログラムのオブジェクト形式の違いによって，開発環境資産の修正が必要になることがあります。詳細は，「34.1　実行可能ファイルの作成方法」または「34.2　共用ライブラリの作成方法」を参照してください。

(c)　プログラム情報ファイル（.cbp，.cbs，.pif）の移行での注意事項

同じプラットフォームの旧バージョンで作成したプログラム情報ファイルのカバレージ情報は移行できます。ただし，カバレージ情報の蓄積をする場合，使用するカバレージと同じバージョンのコンパイラで再コンパイルしてください。

これまでに蓄積したカバレージ情報は，プログラム情報ファイルを消さないで再コンパイルすることで引き継がれます。

(d)　COBOL85 からのテストコマンドファイル（.tdi，.tds）の移行

サブコマンドの指定形式が変更されています。COBOL2002 の TD コマンドの形式で指定してください。

(e)　オブジェクトファイル(.o)またはアーカイブファイル(.a)を移行して実行可能ファイルを作成する場合の注意事項

AIX COBOL2002 V4 への移行で，AIX(32) COBOL2002 V3 や AIX(64) COBOL2002 V3 以前に作成した COBOL オブジェクト，C プログラムのオブジェクト，およびそれらを含むアーカイブファイルを混在させて実行可能ファイルを作成する場合，動的なリンクによるプログラム呼び出しを使用しているなどで，静的な参照関係がないファイルをリンク対象とするときは，次のオプションのどれかを指定してリンク対象としてください。詳細は，「34.1　実行可能ファイルの作成方法」を参照してください。

- -OldLinkOpt,GCBypass オプション（ccbl2002 コマンドでリンクする場合）
- -bgcbypass リンカオプション
- -bkeepfile リンカオプション
- -u リンカオプション

付録 B.2　機能ごとの移行性と互換性に関する注意事項

(1)　異なるプラットフォームから AIX(64) COBOL2002 への移行で引数の受け渡しに関する注意事項

次に示す条件がすべて重なったとき，引数が正しく受け渡しできません。この場合，受け渡す引数のデータ型を一致させる修正をしてください。

- 次に示すデータとそれ以外のデータの組み合わせで値渡しの引数が指定されている。
  - 集団項目(2 バイト以上)
  - 英字・英数字項目(2 文字以上)
  - 英数字編集項目
  - 外部 10 進項目(2 バイト以上)
  - 内部 10 進項目
  - 数字編集項目(2 バイト以上)
  - 外部浮動小数点項目
  - 日本語項目
  - 日本語編集項目
  - 外部ブール項目(2 バイト以上)
  - 内部ブール項目
- 仮引数と実引数のサイズがともに 4 バイトである。

(2) AIX(64) COBOL2002，Linux(x64) COBOL2002 での Cosminexus 連携機能の Java 実行ファイルに関する注意事項

Windows 版 COBOL2002 Cosminexus 連携機能で生成した COBOL アクセス用 Bean は，Windows 環境で Java コンパイルすると class 化できます。class 化したファイルは，UNIX 環境でもそのまま使用できます。

付録 C　日立 COBOL85 からの古い仕様

ここでは，COBOL85 で作成したプログラムを移行するために用意されている機能について説明します。

COBOL2002 で新規にプログラムを作成する場合は，-IdentCall オプション（COBOL85 形式オプション -i ）や環境変数 CBL_ファイル名を使用しないで，-DynamicLink,IdentCall オプション（COBOL85 形式オプション -Bs）や環境変数 CBLD_ファイル名を使用してください。

付録 C.1　COBOL85 移行用コンパイラオプション

(1)　-IdentCall オプション（COBOL85 形式オプション -i ）

形式

COBOL2002 形式オプション

```
-IdentCall プログラム名 [,プログラム名…]
```

COBOL85 形式オプション

```
-i プログラム名 [,プログラム名…]
```

一意名指定の CALL 文で呼び出す可能性のあるプログラム名称を指定して，動的なリンクで呼び出すシミュレーションができます。ただし，このオプションで指定するプログラム名称が格納されている COBOL ソースファイル，またはオブジェクトファイルをコンパイルしたときに同時に指定し，同一ロード上でリンクしてください。

-IdentCall オプションを指定して動的なリンクでの呼び出しのシミュレーションを行っている一意名指定の CALL 文は，-DynamicLink,IdentCall オプション（COBOL85 形式オプション -Bs）へ移行することを推奨します。

-DynamicLink,IdentCall オプションの詳細については，「32.5.4　コンパイラオプションの一覧」を参照してください。

(2)　CBL_ファイル名

環境変数 CBL_ファイル名は，UNIX COBOL85 の互換ため用意されている環境変数です。COBOL2002 では，環境変数 CBLD_ファイル名を使用してください。

規則

- 環境変数 CBL_ファイル名は，環境変数 CBLD_ファイル名の指定がある場合，無効となります。

そのほかの規則，および設定方法については，「35.3.5　ファイル」の「(3)　CBLD_ファイル名」を参照してください。

設定例

（COBOL での記述例）

```
SELECT A-FILE ASSIGN TO SYS000.
```

（環境変数の設定例）

```
CBL_A_FILE=ISAMDL:NOISAMPREV
```

付録 D　コンパイルリスト

付録 D.1　リストの出力

コンパイラが出力するリストの種類と出力方法について説明します。

(1)　コンパイルリストの種類

(a)　情報リスト

プログラム情報やエラー総数などのコンパイル時の情報を要約して出力したものです。

(b)　原始プログラムリスト

コンパイル時に入力した原始プログラムを出力したものです。相互参照情報やコンパイル時にエラーが検出されたときのエラーメッセージなども出力されます。

(c)　エラーリスト

コンパイルエラーのエラーレベルやエラーメッセージを出力したものです。

(2)　リストの出力方法

(a)　コンパイラオプションの指定

COBOL プログラムをコンパイルしてコンパイルリストを出力するオプションを次に示します。これらのコンパイラオプションを指定しない場合，コンパイルリストを出力しません。

-SrcList,NoCopy

COPY 文で複写した登録集原文の内容を原始プログラムリスト中に展開しません。

-SrcList,CopySup

SUPPRESS 指定のある COPY 文で複写した登録集原文の内容は原始プログラムリスト中に展開しません。SUPPRESS 指定のない COPY 文の場合はすべて展開します。

-SrcList,CopyAll

COPY 文で複写した登録集原文の内容をすべて原始プログラムリスト中に展開します。

-SrcList,OutputAll

COPY 文の指定や条件翻訳，LISTING 指令にかかわらず，強制的にすべてのソース原文をコンパイルリストに展開します。SUPPRESS 指定のある COPY 文の場合も展開します。

-SrcList,xxxxx,NoFalsePath

条件翻訳結果の無効行はコンパイルリストに出力しません。

xxxxx には，CopyAll，CopySup，NoCopy のどれかを指定します。

-SrcList,xxxxx,DataLoc

コンパイルリスト（原始プログラムリスト）にデータ項目の相対位置と長さ（バイト）を 16 進数で表示します。相対位置は，データ部のファイル節／作業場所節／局所場所節の各節の先頭からの位置を表示します。

xxxxx には，OutputAll，CopyAll，CopySup，NoCopy のどれかを指定します。

- NoCopy，CopySup，CopyAll，および OutputAll サブオプションは，同時には指定できません。同時に指定した場合，最後に指定したオプションが有効になります。
- NoFalsePath サブオプションは，その他のオプションと同時に指定する必要があります。ただし，OutputAll サブオプションと同時に指定した場合は，すべての行が出力されるため，NoFalsePath サブオプションは意味を持ちません。
- DataLoc サブオプション指定時に S レベル／U レベルのコンパイルエラーが発生した場合，DataLoc サブオプションの指定があっても相対位置は表示しません。
- DataLoc サブオプションを指定する場合，OutputAll，CopyAll，CopySup，NoCopy のどれかと同時に指定する必要があります。指定がない場合は，コンパイルエラーとなります。

(b) コンパイルリストの出力先

情報リストと原始プログラムリストはコンパイルリストファイル（.lst）に出力されます。

また，エラーリストは標準エラー出力（stderr）に出力されます。

(3) コンパイルリストの出力に関連する翻訳指令

コンパイルリストの出力に関連する翻訳指令について，説明します。

(a) LISTING 指令

LISTING 指令は，コンパイルリストにソースを出力するかどうかを指定します。

>>LISTING ON

この行以降のソースをコンパイルリストに出力します。

>>LISTING OFF

この行の次行以降のソースをコンパイルリストに出力しません。

なお，LISTING 指令の行そのものは，ON／OFF の状態に関係なく，常にコンパイルリストに出力されます。

LISTING 指令の例を次に示します。下線部分は，コンパイルリストに出力されません（出力されない行は詰められます）。

（例 1）

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
```

```
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       >>LISTING OFF
       01 A PIC X(10).
       >>LISTING ON
       01 B PIC X(10).
       PROCEDURE DIVISION.

           DISPLAY 'OK'.
```

上記のプログラムを-SrcList,CopyAll オプションを指定してコンパイルした場合，次のようなコンパイルリストが出力されます。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       >>LISTING OFF
       >>LISTING ON
       01 B PIC X(10).
       PROCEDURE DIVISION.

           DISPLAY 'OK'.
```

コンパイルリストへの出力が OFF の状態でも，LISTING OFF の行は出力されます。LISTING OFF が出力される例を次に示します。

(例 2)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       >>LISTING OFF
       01 A PIC X(10).
       >>LISTING OFF       ←この行は出力される
       01 B PIC X(10).
       >>LISTING ON
       PROCEDURE DIVISION.

           DISPLAY 'OK'.
```

上記のプログラムを-SrcList,CopyAll オプションを指定してコンパイルした場合，次のようなコンパイルリストが出力されます。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       >>LISTING OFF
       >>LISTING OFF
       >>LISTING ON
```

```
       PROCEDURE DIVISION.

           DISPLAY 'OK'.
```

LISTING 指令を含む COBOL プログラム（登録集原文）を COPY 文で複写した場合，LISTING 指令の効果は登録集原文の終わりで終了し，複写元のプログラムの COPY 文が持つ LISTING 指令の状態に戻ります。COPY 文の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「3.2.2　COPY 文」を参照してください。

原始プログラム（SAMPLE1.CBL）中の COPY 文で，LISTING 指令を含む COBOL プログラム（登録集原文　SAMPLE2.CBL）を複写した場合の例を次に示します。

（例 3）

原始プログラム SAMPLE1.CBL

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       COPY SAMPLE2.
       PROCEDURE DIVISION.

           DISPLAY 'OK'.
```

登録集原文 SAMPLE2.CBL

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       >>LISTING OFF
       01 A PIC X(10).
```

登録集原文複写後のプログラム

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       COPY SAMPLE2.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       >>LISTING OFF
       01 A PIC X(10).
       PROCEDURE DIVISION.   ←この行以降は出力される

           DISPLAY 'OK'.
```

上記のプログラムを-SrcList,CopyAll オプションを指定してコンパイルした場合，次のようなコンパイルリストが出力されます。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       COPY SAMPLE2.
C1     DATA DIVISION.
C1     WORKING-STORAGE SECTION.
C1     >>LISTING OFF
```

```
       PROCEDURE DIVISION.

           DISPLAY 'OK'.
```

(b)　PAGE 指令

PAGE 指令は，コンパイルリストの改ページを指定します。PAGE 指令のコンパイルリストでの効果は，固定形式正書法の改ページ標識「/」と同じです。

PAGE 指令の例を次に示します。

（例）

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       >>PAGE    ←この行から改ページする
       DATA DIVISION.
       WORKING-STORAGE SECTION.
```

コンパイルリスト

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.

（この位置で改ページ）

       >>PAGE
       DATA DIVISION.
       WORKING-STORAGE SECTION.
```

(c)　LISTING 指令および PAGE 指令の共通規則

このシステムでは，翻訳処理の最後に LISTING 指令および PAGE 指令が処理されます。そのため，条件翻訳の無効行（条件翻訳の結果，コンパイル対象とならなかった行）に LISTING 指令または PAGE 指令が現れた場合，これらの指令は無効となります。

LISTING 指令および PAGE 指令が条件翻訳の無効行にある場合の例を次に示します。

（例）

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       >>DEFINE ABC 123
       >>IF ABC  = 123
      *>  コメント
       >>ELSE
         >>PAGE          ←この>>PAGEは効果がない
         >>LISTING OFF   ←この>>LISTING OFFは効果がない
       >>END-IF
       DATA DIVISION.
```

```
       WORKING-STORAGE SECTION.
       PROCEDURE DIVISION.
```

付録 D.2　リストの見方

リストのヘッダと各リストの見方について説明します。

(1)　リストのヘッダ

リストの先頭に出力されるヘッダの出力形式を次に示します。

図 D-1　ヘッダの出力形式

```
COBOL2002 (c)  VV-RR  *** CCC･･･CCC *** YYYY-MM-DD HH:MM:SS
    1.     2.    3.          4.             5.        6.
```

1. COBOL2002：COBOL2002 であることの記述
2. (c)：COBOL2002 の識別記号
   識別記号については「付録 K.2　このマニュアルでの表記」を参照
3. VV-RR：COBOL2002 のバージョン番号
4. CCC・・・CCC：リストの名称（「情報リスト」または，「原始プログラムリスト」）
5. YYYY-MM-DD：コンパイル日付（年-月-日）
6. HH:MM:SS：コンパイル時刻（時:分:秒）

(2)　情報リスト

情報リストの出力形式を次に示します。リスト 1 行当たりのカラム数は 80 カラムです。

図 D-2　情報リストの出力形式

```
        COBOL2002 (c)     VV-RR     *** 情報リスト          ***                    YYYY-MM-DD  HH:MM:SS

* プログラム名                                  プログラム属性                                    1.
      100  SAMPLETP
    75200  AFTER_PROC                           INITIAL

* クラス名                                      クラス属性                                        2.
    41000  FRUIT                                FINAL
   105200  FRUIT_SHOP                           FINAL

* インタフェース名                                                                              3.
     7800  INTF_FRUIT
   103100  INTF_SHOP

* 関数名                                                                                        4.
   320100  UF_GPROC_SSP
   420200  UF_RPROC_S02
                                                                                               5.
* 入力ファイル名 : SAMPLE1.cbl                                                                  6.

* エラー総数  :     7個                                                                         7.
      (Ｉレベル :     2個  Ｗレベル :     5個  Ｓレベル :     0個)

* オプション  :  -SrcList,CopyAll
                 -CVInf
                 -DebugRange

* 入力ソース枚数  :    638行                                                                    8.
        (主入力 :    624行  登録集 :     14行)
  記述項         :     65個
  文             :    161個
```

```
* 登録集原文名
  行番号    登録集原文名                                                                        9.
       17  /cobol2002/work/test/COPYSRC1.cbl

* リポジトリファイル名
  行番号    リポジトリファイル名                                                                10.
      210  /cobol2002/work/test/FRUIT_SHOP.rep

* サブスキーマ情報
  行番号    サブスキーマ名                 スキーマ名                                           11.
  1000064  SUBSCM0X                       S001

* ファイル情報
  行番号    ファイル名                        編成 アクセス OPEN種別                            12.
  1000122  RDB-FILE                          RB   SEQ
  1000131  I-FILE                            S    SEQ      I O I-O E
  1000137  O-FILE                            S    SEQ
  1000138  I-FILE2                           I    SEQ      I O I-O

* 外部プログラム名
   ERRSHORI   EXT_PROC                                                                         13.

* 内部プログラム名                          プログラム属性                                     14.
   106900  ENDSHORI
   108300  ENDSHORI2                        COMMON

* ファクトリメソッド名                      メソッド属性                                       15.
    23900  Open                             FINAL
   100200  Close

* インスタンスメソッド名                    メソッド属性                                       16.
     6900  PrintFruit                       FINAL
     8300  PrintAmoun
    18300  SetName

* メソッド名                                メソッド属性                                       17.
   200100  I-METH1                          GET PROPERTY
   200500  I-METH2

* 関数名                                                                                      18.
   UFUNC001    UFUNC002   UFUNC003
```

1. プログラム名

```
* プログラム名                                  プログラム属性
      100  SAMPLETP
   75200   AFTER_PROC                          INITIAL
     a.        b.                                 c.
```

a.

IDENTIFICATION DIVISION の行番号

b.

最外側の原始プログラムの PROGRAM-ID 段落で指定した名称

c.

PROGRAM-ID 段落で指定されたプログラム属性

出力内容は，INITIAL／RECURSIVE のどちらかとなります。

PROGRAM-ID 段落でプログラム属性を指定していない場合は出力されません。

スタックコンパイル機能（連続コンパイル機能）を使用している場合，最外側のプログラム定義が複数ある場合があります。この場合，プログラム名は 1 行に一つずつ出力されます。

2. クラス名

```
＊ クラス名                                    クラス属性
    41000  FRUIT                               FINAL
   105200  FRUIT_SHOP                          FINAL
     a.        b.                                c.
```

a.

IDENTIFICATION DIVISION の行番号

b.

CLASS-ID 段落で指定した名称

c.

CLASS-ID 段落で指定した属性

出力内容は，FINAL だけとなります。

属性が指定されていない場合は，出力されません。

スタックコンパイル機能（連続コンパイル機能）を使用している場合，クラス定義が複数ある場合があります。この場合，クラス名は 1 行に一つずつ出力されます。

3. インタフェース名

```
＊ インタフェース名
     7800   INTF_FRUIT
   103100   INTF_SHOP
     a.        b.
```

a.

IDENTIFICATION DIVISION の行番号

b.

INTERFACE-ID 段落で指定した名称

スタックコンパイル機能（連続コンパイル機能）を使用している場合，インタフェース定義が複数ある場合があります。この場合，インタフェース名は 1 行に一つずつ出力されます。

4. 関数名

```
* 関数名
   320100  UF_GPROC_SSP
   420200  UF_RPROC_S02
     a.          b.
```

a.

IDENTIFICATION DIVISION の行番号

b.

FUNCTION-ID 段落で指定した名称

スタックコンパイル機能（連続コンパイル機能）を使用している場合，関数定義が複数ある場合があります。この場合，関数名は 1 行に一つずつ出力されます。

5. 入力ファイル名：コンパイラに入力したファイルの名称
6. エラー総数：発生したエラーの総数

   ただし，メッセージ番号 9000 番台以降を除きます。また，U レベル（回復不能）エラーの個数を含みます。

   エラー総数が 0 以外の場合，W レベル，S レベルが必ず表示されます。

   W レベル：警告エラーの個数

   S レベル：重大エラーの個数

   情報提示レベルのメッセージが発生した場合，I レベルが表示されます。

   I レベル：情報提示レベルのメッセージ
7. オプション：コンパイル時に有効となったコンパイラオプション

   オプションは，1 行に一つずつ出力されます。ただし，80 カラムを超えた場合は，次の行に出力されます。
8. 入力ソース枚数：入力した原始プログラムのレコード枚数

   主入力：主入力のレコード枚数

   登録集：登録集のレコード枚数

   記述項：データ名の個数

   文：手続き部の文の個数

   ただし，「文」には，ELSE，END 動詞は含みません。また，EXEC SQL ～ END-EXEC は 1 文として数えます。
9. 登録集原文名

   行番号：COPY 文が記述されている原始プログラム中の行番号

   登録集原文名：COPY 文で取り込む登録集原文が格納されているファイルの名称
10. リポジトリファイル名

    行番号：構成節中でクラス名／インタフェース名が指定された行番号

    リポジトリファイル名：取り込んだリポジトリファイル名（絶対パス）
11. サブスキーマ情報

    行番号：サブスキーマ名が記述されている原始プログラム中の行番号

サブスキーマ名：サブスキーマ名

スキーマ名：スキーマ名

12. ファイル情報

行番号：入出力処理で使用するファイルを定義している行番号

ファイル名：ファイル記述項で指定したファイル名

編成：ファイル編成

S：順編成ファイル

SX：XMAP3 のプリンタ（AIX(32)で有効）だけを使用する順編成ファイル

Sx：実行時に XMAP3 のプリンタ（AIX(32)で有効），または順編成ファイルの使用を指定できるファイル

I：索引編成ファイル

R：相対編成ファイル

T：テキスト編成ファイル

SR：整列用ファイル

CV：CSV 編成ファイル

RB：HiRDB による索引編成ファイル

アクセス：ファイルのアクセス法

SEQ：順アクセス

RAN：乱アクセス

DYN：動的アクセス

OPEN 種別：ファイルオープンの目的の種別

I：INPUT

O：OUTPUT

I-O：I-O

E：EXTEND

13. 外部プログラム名

CALL 文で呼び出す外部プログラムの名称

80 カラムを超えた場合は，次の行に出力されます。

コンパイラ内部で生成した CALL 文の外部プログラム名は出力されません。

また，スタックコンパイルの場合は，翻訳グループ内で解決済みの外部プログラム名は出力されません。

14. 内部プログラム名

```
* 内部プログラム名                              プログラム属性
   106900  ENDSHORI
   108300  ENDSHORI2                            COMMON
     a.        b.                                 c.
```

a.

内部プログラム（最外側のプログラム以外のプログラム）の IDENTIFICATION DIVISION の行番号

b.

内部プログラムの PROGRAM-ID 段落で指定したプログラム名

80 カラムを超えた場合は，次の行に出力されます。

c.

内部プログラムの PROGRAM-ID 段落で指定されたプログラム属性

PROGRAM-ID 段落でプログラム属性が指定されていない場合は出力されません。

内部プログラムが複数ある場合，内部プログラム名は 1 行に一つずつ出力されます。

ここで表示される内容および形式は，1.のプログラム名と同じです。

15. ファクトリメソッド名

```
* ファクトリメソッド名                    メソッド属性
    23900  Open                          FINAL
   100200  Close
     a.      b.                            c.
```

a.

ファクトリメソッドの IDENTIFICATION DIVISION の行番号

b.

ファクトリメソッドの METHOD-ID 段落で指定したメソッド名

c.

METHOD-ID 段落で指定した属性

出力内容は，FINAL／OVERRIDE／GET PROPERTY／SET PROPERTY となります。ただし，SET PROPERTY／GET PROPERTY は，ほかの属性と同時に指定できません。

また，メソッド名だけの場合は，何も出力されません。

クラスを継承している場合，スーパクラスのファクトリメソッド名は出力されません。また，インタフェースを継承している場合，継承されているインタフェースのメソッド名は出力されません。

16. インスタンスメソッド名

```
* インスタンスメソッド名                  メソッド属性
    6900  PrintFruit                     FINAL
    8300  PrintAmoun
   18300  SetName
     a.      b.                            c.
```

a.

インスタンスメソッドの IDENTIFICATION DIVISION の行番号

b.

インスタンスメソッドの METHOD-ID 段落で指定したメソッド名

c.

METHOD-ID 段落で指定した属性

出力内容は，FINAL／OVERRIDE／GET PROPERTY／SET PROPERTY となります。ただし，SET PROPERTY／GET PROPERTY は，ほかの属性と同時に指定できません。

また，メソッド名だけの場合は，何も出力されません。

クラスを継承している場合，スーパクラスのファクトリメソッド名およびインスタンスメソッド名は出力されません。また，インタフェースを継承している場合，継承されているインタフェースのメソッド名は出力されません。

17. インタフェース定義中のメソッド情報

a.

インタフェース定義中のメソッドの IDENTIFICATION DIVISION の行番号

b.

インタフェース定義中の METHOD-ID 段落で指定したメソッド名

c.

METHOD-ID 段落で指定された属性

出力内容は，GET PROPERTY／SET PROPERTY のどちらかとなります。

また，メソッド名だけの場合は，何も出力されません。

18. 関数名

呼び出している利用者定義関数の名称

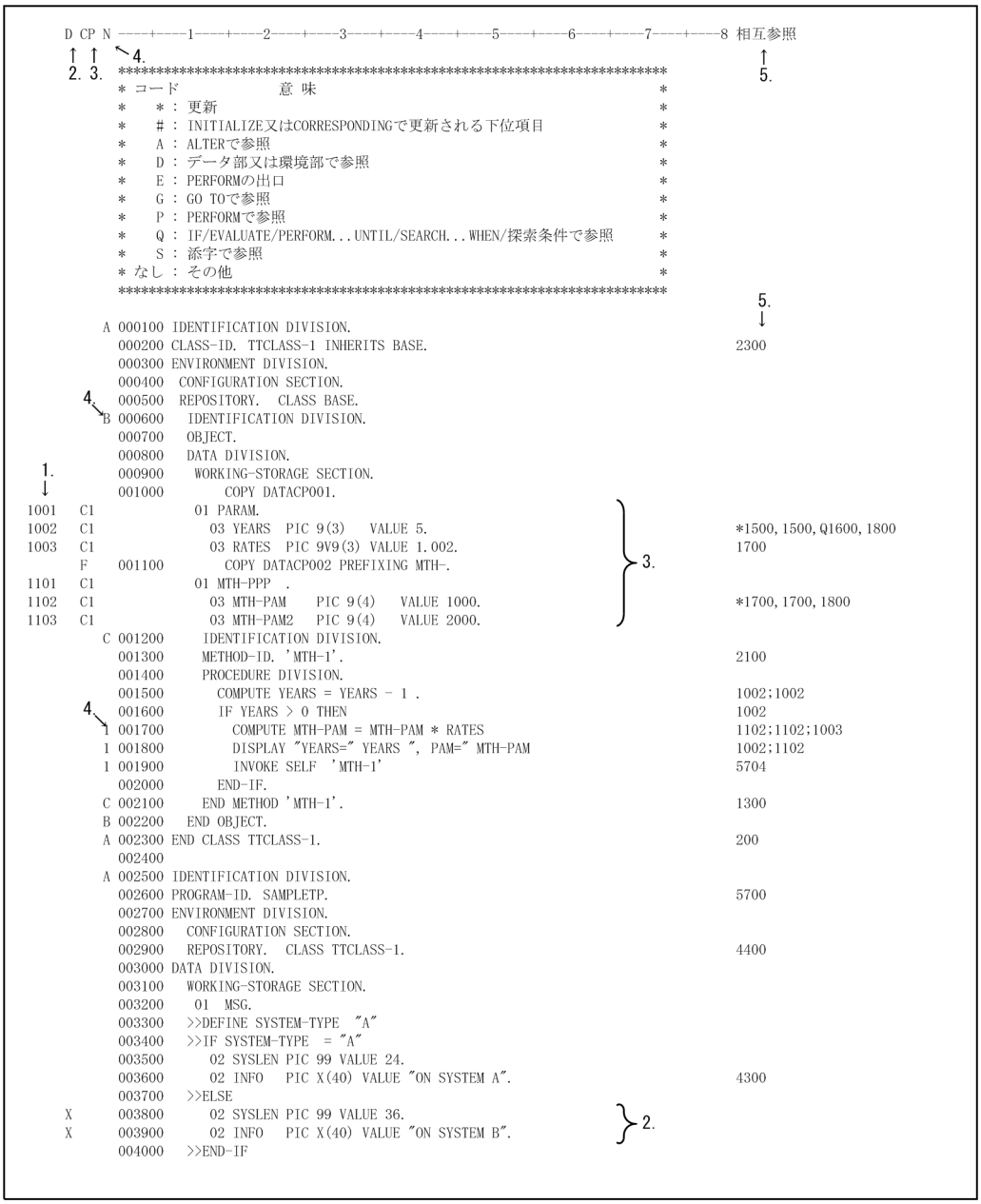
注意事項

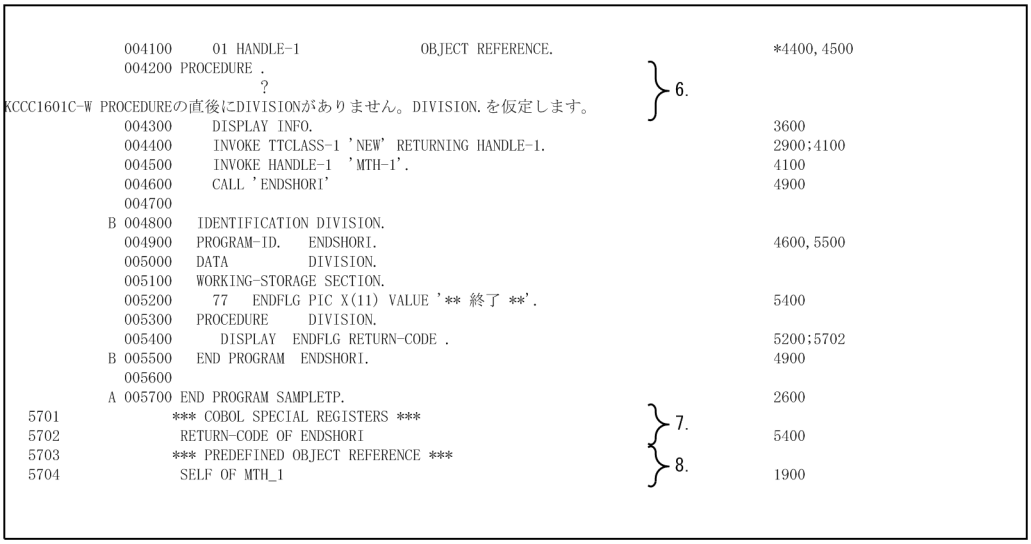
- 該当する定義がソース中にない場合，情報リストにその部分は出力されません。
- SQL 情報（SQL 構文内の実行文の数）は出力されません。
- プログラム名などが長くなった場合には，次の行に折り返して出力されます。折り返した場合には，前行の開始位置と同じ位置から開始します。また，行番号は，折り返した行には出力されません。
- 全角文字が折り返しの境界にわたった場合，その全角文字は，次の行に出力されます。

(3) 原始プログラムリスト

原始プログラムリストの出力形式を次に示します。リスト 1 行当たりのカラム数は 134 カラムです。ただし，自由形式正書法のソースの場合，-EucPosition オプション指定の場合，またはコンパイラ環境変数 CBLFIXEDFORMLINE=255 が有効な場合はソースの 1 行の長さによって，307 カラムまで拡張されます。

図 D-3　原始プログラムリストの出力形式





1. 行番号

   COPY 文で複写した原始プログラムの行番号が昇順に並ばない場合，この原始プログラムに対してコンパイラが生成した行番号が出力されます。EXEC SQL～END-EXEC の間もコンパイラが生成した行番号が出力されます。また，入力した原始プログラムの行番号が昇順になっていない部分についても，新しい行番号を生成します。これらの行番号は，該当する行の左にゼロサプレス右寄せで出力されます。

2. D

   翻訳指令（Compiler Directive）に関する情報

   X

   条件翻訳の結果，無効になる行に出力されます。

3. CP

   COPY 文で複写した原始プログラムの種別

   Cn

   COPY 文で複写した原始プログラムが展開された行に出力されます。

   「n」は 1～9 の整数で，COPY 文の入れ子レベルを示します。入れ子レベルが 10 以上のときは「C*」が出力されます。

   F

   PREFIXING 指定または SUFFIXING 指定の COPY 文の行に出力されます。

   COPY 文によって展開された行は，通常と COPY 文と同じように Cn を出力します。

   DB

   サブスキーマによって展開された原始プログラムの行に出力されます。

4. N

   プログラムと文の入れ子レベル

   プログラムの入れ子レベルは，IDENTIFICATION DIVISION と END PROGRAM の行に英字 A～Z の順番で出力されます。Z を超えたときは「#」が出力されます。また，プログラムだけでなく，

END METHOD，END FACTORY，END CLASS，END OBJECT，END FUNCTION のように，IDENTIFICATION DIVISION と対になるものには，すべて出力されます。

文の入れ子レベルは 1～9 の整数で出力されます。10 以上のときは「*」が出力されます。1 行中に複数の文があって入れ子のレベルが異なるときは，先頭の文の入れ子レベルが出力されます。

5. 相互参照

定義／参照している行の参照情報コード

参照情報コードとその意味については，リストの上段で説明しています。

行番号が「,」で区切ってあるときは行中の 1 データ名が複数行で参照されていることを示し，「;」で区切ってあるときは行中に複数個のデータ名が記述してあることを示します。

次に相互参照情報の見方の例を示します。

(a)1002 行（定義側）

YEARS というデータ名は，行番号 1500 中の 2 か所と行番号 1600 および行番号 1800 で参照されています。参照している行番号を','で区切って表示されます。

(b)1800 行（参照側）

YEARS と MTH-PAM は，それぞれ行番号 1002 と 1102 で定義されています。これら二つは別名称なので，';'で区切って表示されます。

6. エラーメッセージ

エラーが発生したとき，その直後にエラーメッセージとエラーのカラムの位置が出力されます。カラムの位置は「？」で表示されます。

7. COBOL 特殊レジスタ

使用した COBOL 特殊レジスタが原始プログラムの次に出力されます。特殊レジスタを使用しているファイル名とプログラム名も出力されます。

8. 既定義オブジェクト

既定義オブジェクトの SELF または EXCEPTION-OBJECT の相互参照情報が出力されます。

注意事項

- 9000 番台以降のメッセージは，原始プログラムリスト中に出力されません。
- SQL 文のエラー情報と相互参照情報は出力されません。
- U レベル（回復不能）エラー発生時は，リストが最後まで出力されない場合があります。
- スタックコンパイル機能（連続コンパイル機能）を使用している場合，すべてのプログラムが連続して出力されます。

(4) 相対位置表示時の原始プログラムリスト

(a) 原始プログラムリストの内容

-SrcList,xxxxx,DataLoc オプション※1 指定時，原始プログラムリストの原文の前に相対位置と長さを表示します。COBOL プログラムの実行で異常終了時に出力されるデータ領域ダンプリスト※2 と，原始プ

ログラムリストに表示される相対位置を付き合わせることで，プログラムの異常終了時のデータ項目内容を効率良く参照できます。

注※1

xxxxx は，OutputAll，CopyAll，CopySup，NoCopy のどれかを指定してください。

注※2

データ領域ダンプリストについては，「36.3　データ領域ダンプリスト」を参照してください。

データロケーションを表示した場合，リスト 1 行当たりのカラム数は 161 カラムです。ただし，自由形式正書法のソースの場合，またはコンパイラ環境変数 CBLFIXEDFORMLINE=255 が有効な場合はソースの 1 行の長さによって，334 カラムまで拡張されます。AIX の場合，-EucPosition オプションが有効なときも，ソースの 1 行の長さによって，334 カラムまで拡張されます。

相対位置表示時の原始プログラムリストの出力形式を次に示します。

図 D-4　相対位置表示時の原始プログラムリストの出力形式

```
D CP N 相対位置        長さ        ----+----1----+----2----+----3----+----4----+----5----+----6 ～(省略)
          1.↗          2.↗     :(省略)
                                 130000 DATA DIVISION.
                                 140000  FILE SECTION.
                                 150000      FD FILE1.
     (00000010)     (00000014) 160000         01 FDATA1.
     (00000010)     (0000000a) 170000           03 FDATA11       PIC X(10).
     (0000001a)     (0000000a) 180000           03 FDATA12       PIC X(10).
                                 200000      FD FILE2 EXTERNAL.
     (00000000)     (00000014) 210000         01 FDATA2.
     (00000000)     (0000000a) 220000           03 FDATA21       PIC x(10).
     (0000000a)     (0000000a) 230000           03 FDATA22       PIC X(10).
                                 240000
                                 300000  WORKING-STORAGE SECTION.
     (00000000)     (00000007) 310000      01 DATA1    PIC X(7) VALUE "DATA1".
     (00000008)     (00000007) 320000      01 DATA2    PIC X(7) VALUE "DATA2".
                                 330000
     (00000010)     (000000b4) 340000      01 DATA3.
3.  0(00000010)     (0000000f) 350000        02  DATA31  OCCURS 10 DEPENDING ON DEP1.
    0(00000000)     (00000005) 360000          03  DATA311    PIC X(05).
    0(00000005)     (00000005) 370000          03  DATA312    PIC X(05).
    0(0000000a)     (00000005) 380000          03  DATA313    PIC X(05).
4.  V(00000000)     (0000000a) 390000        02  DATA32       PIC X(10).
    V(0000000a)     (0000000a) 400000        02  DATA33       PIC X(10).
    V(00000014)     (0000000a) 410000        02  DATA34       PIC X(10).
                                 420000
     (000000c8)     (00000002) 430000      01 DEP1     PIC 9(02)      VALUE 10.
                                 440000
     (000000d0)     (00000001) 450000      01 DATA4.
     (000000d0)     (00000001) 460000        02  DATA41.
     (000000d0)[0]  (00000001) 470000          03  DATA411    PIC 1(1) BIT.
     (000000d0)[1]  (00000001) 480000          03  DATA412    PIC 1(1) BIT.
     (000000d0)[2]  (00000001) 490000          03  DATA413    PIC 1(1) BIT.
     (000000d0)[3]  (00000001) 500000          03  DATA414    PIC 1(1) BIT.
     (000000d0)[4]  (00000001) 510000          03  DATA415    PIC 1(1) BIT.
     (000000d0)[5]  (00000001) 520000          03  DATA416    PIC 1(1) BIT.
     (000000d0)[6]  (00000001) 530000          03  DATA417    PIC 1(1) BIT.
     (000000d0)[7]  (00000001) 540000          03  DATA418    PIC 1(1) BIT.
                                 550000
                                 560000  LOCAL-STORAGE SECTION.
5.   (00000000)     (0000000a) 570000      01 DATA5    PIC X(10) VALUE "DATA5".
     (00000010)     (0000000a) 580000      01 DATA6    PIC X(10) VALUE "DATA6".
                                 500000 PROCEDURE DIVISION.
```

出力される原始プログラムリストの内容を次に示します。なお，次に示す内容以外のものは，「(3)　原始プログラムリスト」を参照してください。

1. 相対位置

この行には，データ項目の相対位置（バイト）を 16 進数で表示します。相対位置は，データ部のファイル節／作業場所節／局所場所節に定義されたデータ項目に表示します。

2. 長さ

この行には，データ項目の長さ（バイト数）を 16 進数で表示します。

3. O

OCCURS 句を含むデータ名またはその従属項目を表します。

4. V

相対位置が可変となるデータ名を表します。

5. 0～7

内部ブール項目の場合にビット位置を表示します。

注意事項

- 条件名，定数名，指標名およびアドレス名に対しては，相対位置および長さは表示しません。
- コンパイル時に S レベルまたは U レベルのコンパイルエラーが発生した場合，-SrcList,xxxxx,DataLoc オプションの指定があっても相対位置は表示しません。

(b)　データ領域ダンプリストの出力対象

相対位置をコンパイルリストに表示するにあたり，翻訳単位ごとのデータ領域ダンプリストの出力対象を次の表に示します。なお，コンパイルリストに表示する相対位置は，データ領域ダンプリストに表示されるデータ項目に対して表示します。

表 D-1　翻訳単位ごとのデータ領域ダンプリスト出力対象

節	翻訳単位					
	プログラム定義	関数定義	インタフェース定義	クラス定義		
			メソッド定義	ファクトリ定義	オブジェクト定義	メソッド定義
ファイル節 FILE SECTION	○	○	−	○	○	−
作業場所節 WORKING-STORAGE SECTION	○	○	−	○	○	−
局所場所節 LOCAL-STORAGE SECTION	○	○	−	−	−	○
連絡節 LINKAGE SECTION	×	×	−	−	−	×
サブスキーマ節 SUBSCHEMA SECTION	×	×	−	−	−	−
通信節	×	×	−	×	×	−

節	翻訳単位					
	プログラム定義	関数定義	インタフェース定義	クラス定義		
			メソッド定義	ファクトリ定義	オブジェクト定義	メソッド定義
COMMUNICATION SECTION						
報告書節 REPORT SECTION	×	×	−	×	×	−
画面節 SCREEN SECTION※	×	×	−	×	×	−
画面節 WINDOW SECTION※	×	×	−	×	×	−

（凡例）

○：データ領域ダンプリストに表示される

×：データ領域ダンプリストに表示されない

−：節を指定できない

注※

AIX で有効です。

(c)　データ種別ごとの相対位置表示の可否

データ種別ごとの各節での相対位置の表示の可否を次の表に示します。

表 D-2　データ種別ごとの各節での相対位置の表示の可否

データ種別	指定された句やデータ名	ファイル節※	作業場所節※	局所場所節
データ記述項	データ名が FILLER	○	○	○
	ADDRESSED 句のアドレス名	−	×	×
	ADDRESSED 句を含むデータ名	−	○	○
	INDEXED 句の指標名	×	×	×
	EXTERNAL 句	○	○	−
	GLOBAL 句	○	○	○
	JUSTIFIED 句	○	○	○
	OCCURS 句	○	○	○
	OCCURS 句を含むデータ名の従属項目	○	○	○
	可変反復データ項目に続くデータ項目	○	○	○

データ種別	指定された句やデータ名	ファイル節※	作業場所節※	局所場所節
	PROPERTY 句	○	○	○
	REDEFINES 句	○	○	○
	RENAMES 句	○	○	○
	SAME AS 句を含むデータ名	○	○	○
	SAME AS 句を含むデータ名の従属項目	△	△	△
	SYNCHRONIZED 句	○	○	○
	TYPE 句を含むデータ名	○	○	○
	TYPE 句を含むデータ名の従属項目	△	△	△
	TYPEDEF 句	○	○	○
アドレスデータ項目	USAGE ADDRESS	○	○	○
指標データ項目	USAGE INDEX	○	○	○
オブジェクト参照データ項目	USAGE OBJECT REFERENCE	−	○	○
ポインタ項目	USAGE POINTER	○	○	○
外部ブール項目	USAGE DISPLAY	○	○	○
内部ブール項目	USAGE BIT	○	○	○
2 進項目など	その他の USAGE 句	○	○	○
独立データ項目	77 レベルのデータ名	−	○	○
定数名	78 レベルのデータ名	×	×	×
条件名	88 レベルのデータ名	×	×	×

（凡例）

○：相対位置を表示できる

△：従属項目はコンパイルリストに表示されないため，従属項目の相対位置を参照したい場合は，「(d)　データ項目ごとの相対位置の見方」の「TYPE 句／SAME AS 句を含むデータ名の従属項目」を参照のこと

×：相対位置を表示できない

−：節に指定できない

注※

クラス定義の場合は，データ部先頭からの相対位置となります。

(d)　データ項目ごとの相対位置の見方

■ OCCURS 句を含むデータ名の従属項目

OCCURS 句を含むデータ名の従属項目については，基本データ項目と同様に節の先頭からの相対位置を16 進数で表示します。また，OCCURS 句を含むデータ名およびその従属項目であることを示す「O」を相対位置の左側に表示します。

OCCURS 句を含むデータ名が集団項目の場合の相対位置の表示を次に示します。

なお，要素の相対位置については，次の計算式で求めてください。

計算式

表中のデータ名の相対位置=参照するデータ名の相対位置の値※＋（参照する表要素のその次元での出現番号－1）×表の 1 要素の長さ

注※

相対位置に「O」の表示があるデータ名

図 D-5　OCCURS 句を含むデータ名の従属項目の相対位置の表示

```
D CP N 相対位置        長さ        ----+----1----+----2----+----3----+----4----+----5----+----6 ~
                                 :（省略）
                                   300000  WORKING-STORAGE SECTION.
        (00000000)     (00000007) 310000       01 DATA1    PIC X(7)  VALUE "DATA1".
        (00000008)     (0000000a) 320000       01 DATA2    PIC X(10) VALUE "DATA2".
                                  330000
        (00000018)     (0000004b) 340000       01 DATA3.
      O (00000018)     (0000000f) 350000         02  DATA31  OCCURS 5.
1.    O (00000018)     (00000005) 360000           03  DATA311    PIC X(05).
      O (0000001d)     (00000005) 370000           03  DATA312    PIC X(05).
      O (00000022)     (00000005) 380000           03  DATA313    PIC X(05).
                                  400000
        (00000068)     (00000002) 420000       01 I        PIC 9(02) COMP VALUE 1.
        (00000070)     (00000002) 430000       01 J        PIC 9(02) COMP VALUE 0.
        (00000078)     (00000002) 440000       01 K        PIC 9(02) COMP VALUE 0.
                                  450000
                                  460000  LOCAL-STORAGE SECTION.
        (00000000)     (0000000a) 470000       01 DATA4    PIC X(10) VALUE "DATA4".
        (00000010)     (0000000a) 480000       01 DATA5    PIC X(10) VALUE "DATA5".
                                  490000
                                  500000 PROCEDURE DIVISION.
                                  510000       MOVE ALL SPACE TO DATA3.
                                  520000       MOVE "HHHHHIIIIIJJJJJ" TO DATA31(3).
                                  530000       COMPUTE K = I / J.
   A                              540000       END PROGRAM SAMPLE1.
```

1. O

   OCCURS 句を含むデータ名またはその従属項目を表します。

例：「データ名：DATA312」の 3 番目の要素の場合

表中のデータ名の相対位置=（0x0000001d）＋（3－1）×（0x0000000f）

=（0x0000003b）

=59 バイト

上記の図に対応したデータ領域ダンプリストの内容を次に示します。

```
<プログラム名 = SAMPLE1>

<WORKING-STORAGE SECTION>                                    「データ名：DATA312」の3番目の要素の内容
位置----   内容---- -------- -------- --------    ----------------
00000000   44415441 31202000 44415441 32202020    DATA1 .DATA2
00000010   20200000 00000000 20202020 20202020     .....
00000020   20202020 20202020 20202020 20202020
00000030   20202020 20204848 48484849 49494949          HHHHHIIIII
00000040   4a4a4a4a 4a202020 20202020 20202020    JJJJJ
00000050   20202020 20202020 20202020 20202020
00000060   20202000 00000000 01000000 00000000       ............
00000070   00000000 00000000 00000000 00000000    ................
```

データ領域ダンプリストの<WORKING-STORAGE SECTION>の先頭から（0x0000003b）（59バイト）目の内容を５バイト参照することで,「データ名：DATA312」の３番目の内容を確認できます。

OCCURS句を含むデータ名が基本項目の場合の相対位置の表示を次に示します。

図 D-6　OCCURS句を含むデータ名が基本項目の場合の相対位置の表示

```
      D CP N 相対位置        長さ        ----+----1----+----2----+----3----+----4----+----5----+----6 ～
                                       :(省略)
                                 300000  WORKING-STORAGE SECTION.
            (00000000)    (00000007) 310000       01 DATA1     PIC X(7)  VALUE "DATA1".
            (00000008)    (0000000a) 320000       01 DATA2     PIC X(10) VALUE "DATA2".
                                 330000
            (00000018)    (00000019) 340000       01 DATA3.
          O (00000018)    (00000005) 350000         02  DATA31  OCCURS 5 PIC X(05).
                                 400000
            (00000038)    (00000002) 420000       01 I         PIC 9(02) COMP VALUE 1.
            (00000040)    (00000002) 430000       01 J         PIC 9(02) COMP VALUE 0.
            (00000048)    (00000002) 440000       01 K         PIC 9(02) COMP VALUE 0.
                                 450000
                                 460000  LOCAL-STORAGE SECTION.
            (00000000)    (0000000a) 470000       01 DATA4     PIC X(10) VALUE "DATA4".
            (00000010)    (0000000a) 480000       01 DATA5     PIC X(10) VALUE "DATA5".
                                 490000
                                 500000 PROCEDURE DIVISION.
                                 510000       MOVE ALL SPACE TO DATA3.
                                 520000       MOVE "HHHHH" TO DATA31(3).
                                 530000       COMPUTE K = I / J.
          A                      540000       END PROGRAM SAMPLE1.
```

例：「データ名：DATA31」の３番目の要素の場合

表中のデータ名の相対位置＝（0x00000018）＋（3－1）×（0x00000005）

＝（0x00000022）

＝34バイト

上記の図に対応したデータ領域ダンプリストの内容を次に示します。

```
<プログラム名 = SAMPLE1>

<WORKING-STORAGE SECTION>                                    「データ名：DATA31」の3番目の要素の内容
位置----   内容---- -------- -------- --------    ----------------
00000000   44415441 31202000 44415441 32202020    DATA1 .DATA2
00000010   20200000 00000000 20202020 20202020     ......
00000020   20204848 48484849 20202020 20202020     HHHHH
00000030   20000000 00000000 01000000 00000000    ................
00000040   00000000 00000000 00000000 00000000    ................
```

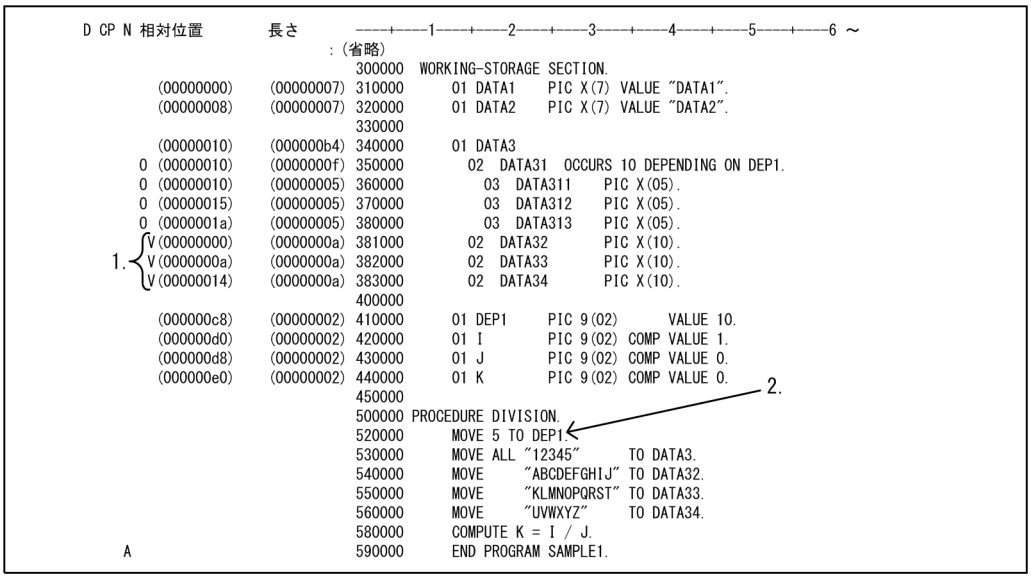
データ領域ダンプリストの<WORKING-STORAGE SECTION>の先頭から（0x00000022）（34バイト）目の内容を５バイト参照することで,「データ名：DATA31」の３番目の要素の内容を確認できます。

■ 可変反復データ項目に続くデータ項目

可変反復データ項目に続くデータ項目は，繰り返し回数によって長さが可変となります。そのため，可変反復データ項目に続くデータ項目の相対位置も可変となります。この可変反復データ項目に続くデータ項目の相対位置は，定義された節の先頭からの相対位置ではなく，最初に現れたデータ項目を 0 とした相対位置を 16 進数で表示します。また，相対位置が可変となるデータ名であることを示す「V」を，相対位置の左側に表示します。

相対位置が可変となるデータ名が基本項目の場合の相対位置の表示を次に示します。

図 D-7　相対位置が可変となるデータ名が基本項目の場合の相対位置の表示



```
D CP N 相対位置        長さ        ----+----1----+----2----+----3----+----4----+----5----+----6 ～
                                 :(省略)
                                  300000  WORKING-STORAGE SECTION.
          (00000000)    (00000007) 310000      01 DATA1    PIC X(7) VALUE "DATA1".
          (00000008)    (00000007) 320000      01 DATA2    PIC X(7) VALUE "DATA2".
                                   330000
          (00000010)    (000000b4) 340000      01 DATA3
        0 (00000010)    (0000000f) 350000        02  DATA31  OCCURS 10 DEPENDING ON DEP1.
        0 (00000010)    (00000005) 360000          03  DATA311    PIC X(05).
        0 (00000015)    (00000005) 370000          03  DATA312    PIC X(05).
        0 (0000001a)    (00000005) 380000          03  DATA313    PIC X(05).
        V (00000000)    (0000000a) 381000        02  DATA32       PIC X(10).
1.      V (0000000a)    (0000000a) 382000        02  DATA33       PIC X(10).
        V (00000014)    (0000000a) 383000        02  DATA34       PIC X(10).
                                   400000
          (000000c8)    (00000002) 410000      01 DEP1     PIC 9(02)      VALUE 10.
          (000000d0)    (00000002) 420000      01 I        PIC 9(02) COMP VALUE 1.
          (000000d8)    (00000002) 430000      01 J        PIC 9(02) COMP VALUE 0.
          (000000e0)    (00000002) 440000      01 K        PIC 9(02) COMP VALUE 0.
                                   450000                                          2.
                                   500000 PROCEDURE DIVISION.
                                   520000      MOVE 5 TO DEP1.
                                   530000      MOVE ALL "12345"      TO DATA3.
                                   540000      MOVE     "ABCDEFGHIJ" TO DATA32.
                                   550000      MOVE     "KLMNOPQRST" TO DATA33.
                                   560000      MOVE     "UVWXYZ"     TO DATA34.
                                   580000      COMPUTE K = I / J.
  A                                590000      END PROGRAM SAMPLE1.
```

1. V

   相対位置が可変となるデータ名には「V」を表示します。

2. 制御変数

   DEP1 に 5 を設定しているので，繰り返し回数は 5 回で計算します。

要素の相対位置については，次の計算式で求めてください。

計算式

相対位置が可変となるデータ名※1 の相対位置=DEPENDING ON 指定がある OCCURS 句を含むデータ名※2 の相対位置の値＋プログラムが異常終了した時点の表の繰り返し回数の値※3×表の 1 要素の長さ※4 ＋参照するデータ名の相対位置の値

注※1

相対位置に「V」の表示があるデータ名

注※2

相対位置に「O」の表示があるデータ名

注※3

OCCURS～DEPENDING ON に指定されたデータ名に設定した値

注※4

相対位置に「O」の表示があるデータ名の長さ

例：「データ名：DATA33」の場合

可変反復データ項目に続くデータ名の相対位置＝（0x00000010）＋5×（0x0000000f）＋（0x0000000a）

＝（0x00000065）

＝101 バイト

上記の図に対応したデータ領域ダンプリストの内容を次に示します。

```
<プログラム名 = SAMPLE1>

<WORKING-STORAGE SECTION>
位置----   内容---- -------- -------- --------   ----------------
00000000   44415441 31202000 44415441 32202000   DATA1  .DATA2  .
00000010   31323334 35313233 34353132 33343531   1234512345123451
00000020   32333435 31323334 35313233 34353132   2345123451234512
00000030   33343531 32333435 31323334 35313233   3451234512345123
00000040   34353132 33343531 32333435 31323334   4512345123451234
00000050   35313233 34353132 33343541 42434445   5123451234 5ABCDE
00000060   46474849 4a4b4c4d 4e4f5051 52535455   FGHIJKLMNOPQRSTU
00000070   56575859 5a202020 20000000 00000000   VWXYZ    .......
00000080   00000000 00000000 00000000 00000000   ................
           LINES 00000090-000000b0 SAME AS ABOVE
000000c0   00000000 00000000 05000000 00000000   ................
000000d0   01000000 00000000 00000000 00000000   ................
000000e0   00000000 00000000                     ........
```

「データ名：DATA33」の内容

データ領域ダンプリストの<WORKING-STORAGE SECTION>の先頭から（0x00000065）（101 バイト）目の内容を 10 バイト参照することで，「データ名：DATA33」の内容を確認できます。

相対位置が可変となるデータ名が OCCURS 句を含んでいる場合の相対位置の表示を次に示します。

図 D-8　相対位置が可変となるデータ名が OCCURS 句を含んでいる場合の相対位置の表示

```
     D CP N 相対位置        長さ        ----+----1----+----2----+----3----+----4----+----5----+----6 ~
                                    :(省略)
                                    300000  WORKING-STORAGE SECTION.
              (00000000)    (00000007) 310000       01 DATA1    PIC X(7) VALUE "DATA1".
              (00000008)    (00000007) 320000       01 DATA2    PIC X(7) VALUE "DATA2".
                                    330000
              (00000010)    (000000d2) 340000       01 DATA3.
            O (00000010)    (0000000f) 350000         02  DATA31  OCCURS 10 DEPENDING ON DEP1.
            O (00000010)    (00000005) 360000           03  DATA311    PIC X(05).
            O (00000015)    (00000005) 370000           03  DATA312    PIC X(05).
            O (0000001a)    (00000005) 380000           03  DATA313    PIC X(05).
              V(00000000)    (0000000a) 381000         02  DATA32       PIC X(10).
           1.{OV(0000000a)    (0000000a) 382000         02  DATA33 OCCURS 5.
           1.{OV(0000000a)    (00000005) 383000           03  DATA331    PIC X(05).
           1.{OV(0000000f)    (00000005) 384000           03  DATA332    PIC X(05).
                                    400000
              (000000e8)    (00000002) 410000       01 DEP1     PIC 9(02)      VALUE 10.
              (000000f0)    (00000002) 420000       01 I        PIC 9(02) COMP VALUE 1.
              (000000f8)    (00000002) 430000       01 J        PIC 9(02) COMP VALUE 0.
              (00000100)    (00000002) 440000       01 K        PIC 9(02) COMP VALUE 0.
                                    450000
                                    500000 PROCEDURE DIVISION.
                                    520000       MOVE 5 TO DEP1.
                                    540000       MOVE ALL "12345"      TO DATA3.
                                    550000       MOVE     "ABCDEFGHIJ" TO DATA32.
                                    560000       MOVE     "KLMNOPQRST" TO DATA33(3).
                                    590000       COMPUTE K = I / J.
                A                   610000       END PROGRAM SAMPLE1.
```

2.

1. O，V

相対位置が可変となるデータ名が OCCURS 句を含んでいる場合は，「O」と「V」を表示します。

2. 制御変数

DEP1 に 5 を設定しているので，繰り返し回数は 5 回で計算します。

要素の相対位置については，次の計算式で求めてください。

計算式

相対位置が可変となるデータ名※1 の相対位置=DEPENDING ON 指定がある OCCURS 句を含むデータ名※2 の相対位置の値＋プログラムが異常終了した時点の表の繰り返し回数の値※3×表の 1 要素の長さ※4 ＋参照するデータ名の相対位置の値＋（参照する表要素のその次元での出現番号－1）×表の 1 要素の長さ※5

注※1

相対位置に「O」と「V」の表示があるデータ名

注※2

相対位置に「O」の表示がある DEPENDING ON 指定がある OCCURS 句を含むデータ名

注※3

OCCURS～DEPENDING ON に指定されたデータ名に設定した値

注※4

相対位置に「O」の表示がある DEPENDING ON 指定がある OCCURS 句を含むデータ名の長さ

注※5

相対位置に「O」と「V」の表示がある OCCURS 句を含むデータ名の長さ

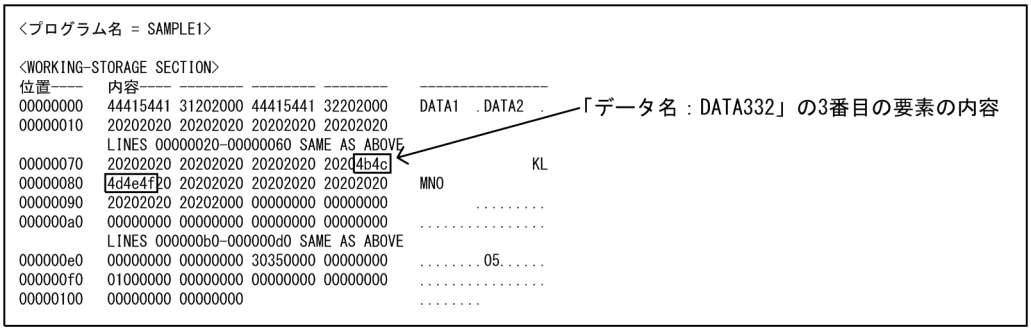
例：「データ名：DATA332」の場合

相対位置が可変となるデータ名の相対位置=（0x00000010）＋5×（0x0000000f）＋（0x0000000f）＋（3－1）×（0x0000000a）

=（0x0000007e）

=126 バイト

上記の図に対応したデータ領域ダンプリストの内容を次に示します。




```
<プログラム名 = SAMPLE1>

<WORKING-STORAGE SECTION>
位置----   内容---- -------- -------- --------    ----------------
00000000   44415441 31202000 44415441 32202000    DATA1 .DATA2 .
00000010   20202020 20202020 20202020 20202020
           LINES 00000020-00000060 SAME AS ABOVE
00000070   20202020 20202020 20202020 20204b4c                KL
00000080   4d4e4f20 20202020 20202020 20202020    MNO
00000090   20202020 20202000 00000000 00000000            .....
000000a0   00000000 00000000 00000000 00000000    ................
           LINES 000000b0-000000d0 SAME AS ABOVE
000000e0   00000000 00000000 30350000 00000000    ........05......
000000f0   01000000 00000000 00000000 00000000    ................
00000100   00000000 00000000                      ........
```

データ領域ダンプリストの<WORKING-STORAGE SECTION>の先頭から（0x0000007e）（126 バイト）目の内容を 5 バイト参照することで，「データ名：DATA332」の内容を確認できます。

■ 内部ブール項目

内部ブール項目については，相対位置としてビット位置も表示します。そのため，() 内に 16 進数でバイト位置を，[] 内にビット位置を 10 進数でそれぞれ表示します。また，長さはビット数を 16 進数で表示します。

内部ブール項目の相対位置の表示を次に示します。

図 D-9　内部ブール項目の相対位置の表示

```
D CP N 相対位置        長さ       ----+----1----+----2----+----3----+----4----+----5----+----6 ～
                         :(省略)
                                   300000  WORKING-STORAGE SECTION.
        (00000000)     (00000007) 310000       01 DATA1    PIC X(7)  VALUE "*DATA1*".
        (00000008)     (00000007) 320000       01 DATA2    PIC X(7)  VALUE "*DATA2*".
1.      (00000010)     (00000001) 340000       01 DATA3.
        (00000010)     (00000001) 350000         02  DATA31.
        (00000010)[0]  (00000001) 351000           03  DATA311    PIC 1(1) BIT.
        (00000010)[1]  (00000001) 352000           03  DATA312    PIC 1(1) BIT.
        (00000010)[2]  (00000001) 353000           03  DATA313    PIC 1(1) BIT.
        (00000010)[3]  (00000001) 354000           03  DATA314    PIC 1(1) BIT.
        (00000010)[4]  (00000001) 355000           03  DATA315    PIC 1(1) BIT.
        (00000010)[5]  (00000001) 356000           03  DATA316    PIC 1(1) BIT.
        (00000010)[6]  (00000001) 357000           03  DATA317    PIC 1(1) BIT.
        (00000010)[7]  (00000001) 358000           03  DATA318    PIC 1(1) BIT.
        (00000018)[0]  (00000003) 359000       01 DATA4    PIC  1(3) BIT.
        (00000020)[0]  (00000004) 359100       01 DATA5    PIC  1(4) BIT.
                                  400000
2.      (00000028)     (00000002) 420000       01 I        PIC 9(02) COMP VALUE 1.
        (00000030)     (00000002) 430000       01 J        PIC 9(02) COMP VALUE 0.
        (00000038)     (00000002) 440000       01 K        PIC 9(02) COMP VALUE 0.
                                  450000
                                  500000 PROCEDURE DIVISION.
                                  520000       MOVE B"1" TO DATA311.
                                  521000       MOVE B"1" TO DATA312.
                                  522000       MOVE B"1" TO DATA313.
                                  523000       MOVE B"1" TO DATA314.
                                  524000       MOVE B"0" TO DATA315.
                                  525000       MOVE B"0" TO DATA316.
                                  526000       MOVE B"0" TO DATA317.
                                  527000       MOVE B"0" TO DATA318.
                                  550000       COMPUTE K = I / J.
    A                             570000       END PROGRAM SAMPLE1.
```

1. 0～7

   [] 内にビット位置を 10 進数で表示します。

2. 長さ

   ビット数を 16 進数で表示します。

上記の図に対応したデータ領域ダンプリストの内容を次に示します。

```
<プログラム名 = SAMPLE1>

<WORKING-STORAGE SECTION>
位置----   内容---- -------- -------- -------- --------    ----------------
00000000   2a444154 41312a00 2a444154 41322a00    *DATA1*.*DATA2*.
00000010   f0000000 00000000 01000000 00000000    ................
00000020   00000000 00000000 00000000 00000000    ................
```

「データ名：DATA3」または
「データ名：DATA31」の内容

データ領域ダンプリストの<WORKING-STORAGE SECTION>の先頭から（0x00000010）（16 バイト）目の内容を 1 バイト参照することで，「データ名：DATA3」または「データ名：DATA31」のビット値を確認できます。

■ EXTERNAL 句を含むレコード記述項

EXTERNAL 句を含むレコード記述項は，レコードの先頭の相対位置を 0 として表示します。

EXTERNAL 句を含むレコード記述項の相対位置の表示を次に示します。

図 D-10　EXTERNAL 句を含むレコード記述項の相対位置の表示

```
      D CP N 相対位置        長さ       ----+----1----+----2----+----3----+----4----+----5----+----6 ～
                            :(省略)
1.                                    130000 DATA DIVISION.
                                      140000  FILE SECTION.
                                      150000      FD FILE1 EXTERNAL.
                (00000000)  (00000050) 160000         01 FDATA1.
                (00000000)  (00000005) 170000           03 FDATA11       PIC 9(05).
                (00000005)  (00000005) 180000           03 FDATA12       PIC 9(05).
                (0000000a)  (00000046) 190000           03 FDATA13       PIC X(70).
                                      290000
                                      300000  WORKING-STORAGE SECTION.
                (00000000)  (00000002) 420000      01 I         PIC 9(02) COMP VALUE 1.
                (00000008)  (00000002) 430000      01 J         PIC 9(02) COMP VALUE 0.
                (00000010)  (00000002) 440000      01 K         PIC 9(02) COMP VALUE 0
                                      490000
                                      500000 PROCEDURE DIVISION.
                                      510000      OPEN OUTPUT FILE1.
                                      520000      MOVE 10         TO FDATA11.
                                      530000      MOVE 20         TO FDATA12.
                                      540000      MOVE ALL "ABC" TO FDATA13.
                                      550000      COMPUTE K = I / J.
                                      560000      CLOSE FILE1.
          A                           570000      END PROGRAM SAMPLE1.
```

1. 相対位置

   レコード先頭の相対位置を 0 として表示します。

上記の図に対応したデータ領域ダンプリストの内容を次に示します。

```
****************************************
* EXTERNAL 指定ファイルのレコード領域 *
****************************************

<ファイル名 = FILE1>

位置----   内容----- -------- -------- --------    ----------------
00000000   30303031 30303030 32304142 43414243    0001000020ABCABC
00000010   41424341 42434142 43414243 41424341    ABCABCABCABCABCA
00000020   42434142 43414243 41424341 42434142    BCABCABCABCABCAB
00000030   43414243 41424341 42434142 43414243    CABCABCABCABCABC
00000040   41424341 42434142 43414243 41424341    ABCABCABCABCABCA
```

レコード記述項の先頭から内容を参照できる

EXTERNAL 句を含むレコード記述項は，レコードの先頭から内容を参照できます。

■ クラス定義のファイル節／作業場所節

クラス定義のファクトリ／オブジェクト定義中のファイル節／作業場所節の各データ項目の相対位置は，各節の先頭からの値ではなく，データ部先頭からの値になります。

クラス定義のファイル節／作業場所節の相対位置の表示を次に示します。

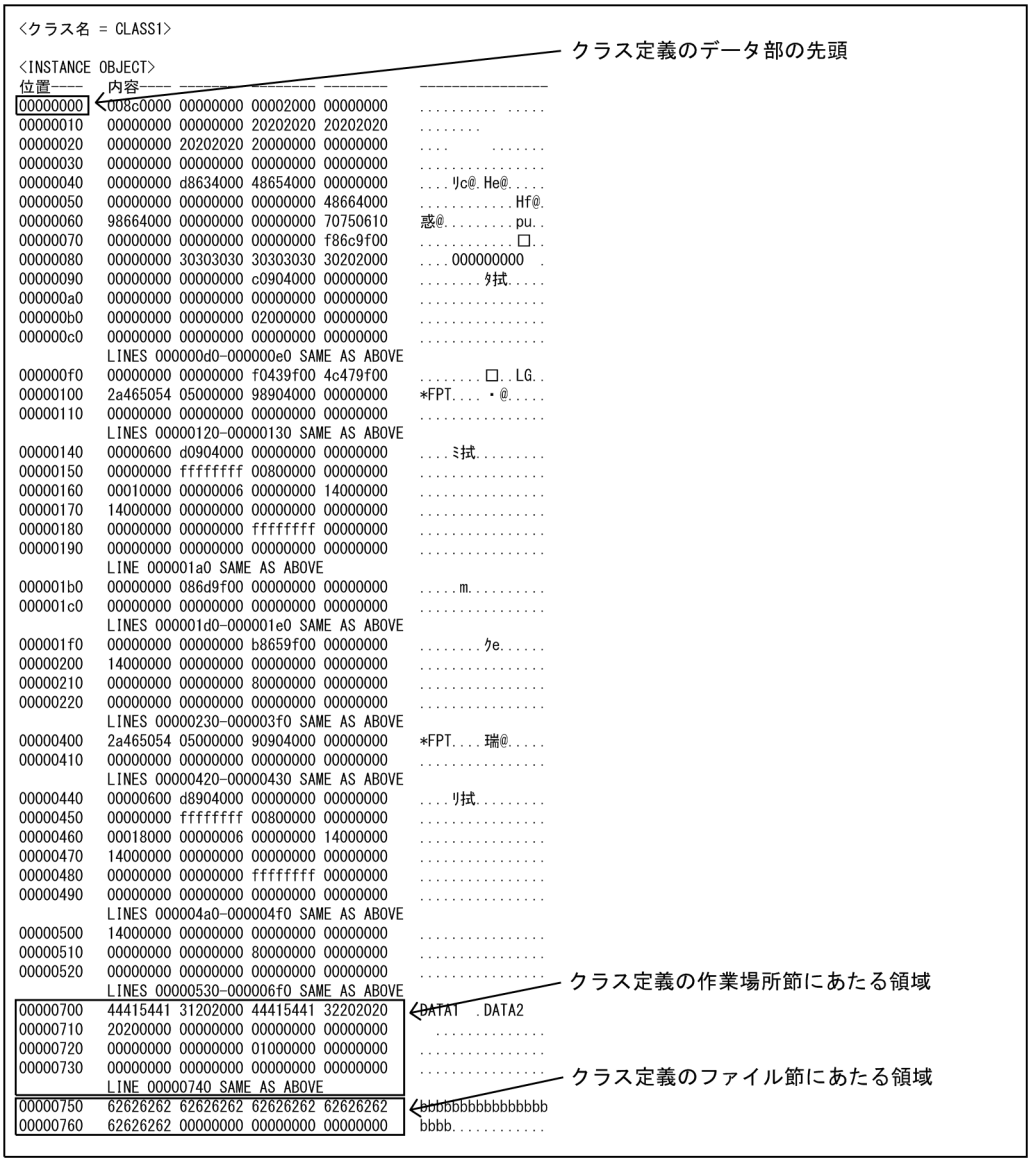
図 D-11　クラス定義のファイル節／作業場所節の相対位置の表示

```
D CP N 相対位置        長さ        ----+----1----+----2----+----3----+----4----+----5----+----6 ～
                                 :(省略)
     B                           010000 ID DIVISION.
                                 020000  OBJECT.
                                 030000 ENVIRONMENT DIVISION.
                                 040000  INPUT-OUTPUT SECTION.
                                 050000    FILE-CONTROL.
                                 060000       SELECT FILE1 ASSIGN TO SYS010
                                 070000              ORGANIZATION SEQUENTIAL.
                                 080000       SELECT FILE2 ASSIGN TO SYS020
                                 090000              ORGANIZATION SEQUENTIAL.
1.                               120000
                                 130000 DATA DIVISION.
                                 140000  FILE SECTION.
                                 150000       FD FILE1.
          (00000750)    (00000014) 160000          01 FDATA1.
          (00000750)    (0000000a) 170000            03 FDATA11       PIC X(10).
          (0000075a)    (0000000a) 180000            03 FDATA12       PIC X(10).
                                 190000
                                 200000       FD FILE2 EXTERNAL.
          (00000000)    (00000014) 210000          01 FDATA2.
          (00000000)    (0000000a) 220000            03 FDATA21       PIC X(10).
          (0000000a)    (0000000a) 230000            03 FDATA22       PIC X(10).
                                 231000
                                 300000  WORKING-STORAGE SECTION.
          (00000700)    (00000007) 310000       01 DATA1    PIC X(7)  VALUE "DATA1".
          (00000708)    (0000000a) 320000       01 DATA2    PIC X(10) VALUE "DATA2".
                                 330000
          (00000718)    (0000000f) 340000       01 DATA3.
          (00000718)    (0000000f) 350000         02  DATA31.
          (00000718)    (00000005) 360000           03  DATA311    PIC X(05).
          (0000071d)    (00000005) 370000           03  DATA312    PIC X(05).
          (00000722)    (00000005) 380000           03  DATA313    PIC X(05).
```

1. 相対位置

   データ部先頭からの値になります。

上記の図に対応したデータ領域ダンプリストの内容を次に示します。




```
<クラス名 = CLASS1>

<INSTANCE OBJECT>
位置----  内容---- -------- -------- --------  ----------------
00000000  008c0000 00000000 00002000 00000000  ..........  .....
00000010  00000000 00000000 20202020 20202020  ........
00000020  00000000 20202020 20000000 00000000  ....     ........
00000030  00000000 00000000 00000000 00000000  ................
00000040  00000000 d8634000 48654000 00000000  ....ﾘc@.He@.....
00000050  00000000 00000000 00000000 48664000  ............Hf@.
00000060  98664000 00000000 00000000 70750610  惑@.........pu..
00000070  00000000 00000000 00000000 f86c9f00  ............口..
00000080  00000000 30303030 30303030 30202000  ....000000000   .
00000090  00000000 00000000 c0904000 00000000  .........ﾀ拭.....
000000a0  00000000 00000000 00000000 00000000  ................
000000b0  00000000 00000000 02000000 00000000  ................
000000c0  00000000 00000000 00000000 00000000  ................
          LINES 000000d0-000000e0 SAME AS ABOVE
000000f0  00000000 00000000 f0439f00 4c479f00  ........口..LG..
00000100  2a465054 05000000 98904000 00000000  *FPT....・@.....
00000110  00000000 00000000 00000000 00000000  ................
          LINES 00000120-00000130 SAME AS ABOVE
00000140  00000600 d0904000 00000000 00000000  ....ﾐ拭.........
00000150  00000000 ffffffff 00800000 00000000  ................
00000160  00010000 00000006 00000000 14000000  ................
00000170  14000000 00000000 00000000 00000000  ................
00000180  00000000 00000000 ffffffff 00000000  ................
00000190  00000000 00000000 00000000 00000000  ................
          LINE 000001a0 SAME AS ABOVE
000001b0  00000000 086d9f00 00000000 00000000  .....m..........
000001c0  00000000 00000000 00000000 00000000  ................
          LINES 000001d0-000001e0 SAME AS ABOVE
000001f0  00000000 00000000 b8659f00 00000000  .........ｸe......
00000200  14000000 00000000 00000000 00000000  ................
00000210  00000000 00000000 80000000 00000000  ................
00000220  00000000 00000000 00000000 00000000  ................
          LINES 00000230-000003f0 SAME AS ABOVE
00000400  2a465054 05000000 90904000 00000000  *FPT....瑞@.....
00000410  00000000 00000000 00000000 00000000  ................
          LINES 00000420-00000430 SAME AS ABOVE
00000440  00000600 d8904000 00000000 00000000  ....ﾘ拭.........
00000450  00000000 ffffffff 00800000 00000000  ................
00000460  00018000 00000006 00000000 14000000  ................
00000470  14000000 00000000 00000000 00000000  ................
00000480  00000000 00000000 ffffffff 00000000  ................
00000490  00000000 00000000 00000000 00000000  ................
          LINES 000004a0-000004f0 SAME AS ABOVE
00000500  14000000 00000000 00000000 00000000  ................
00000510  00000000 00000000 80000000 00000000  ................
00000520  00000000 00000000 00000000 00000000  ................
          LINES 00000530-000006f0 SAME AS ABOVE
00000700  44415441 31202000 44415441 32202020  DATA1  .DATA2
00000710  20200000 00000000 00000000 00000000    ............
00000720  00000000 00000000 01000000 00000000  ................
00000730  00000000 00000000 00000000 00000000  ................
          LINE 00000740 SAME AS ABOVE
00000750  62626262 62626262 62626262 62626262  bbbbbbbbbbbbbbbb
00000760  62626262 00000000 00000000 00000000  bbbb............
```

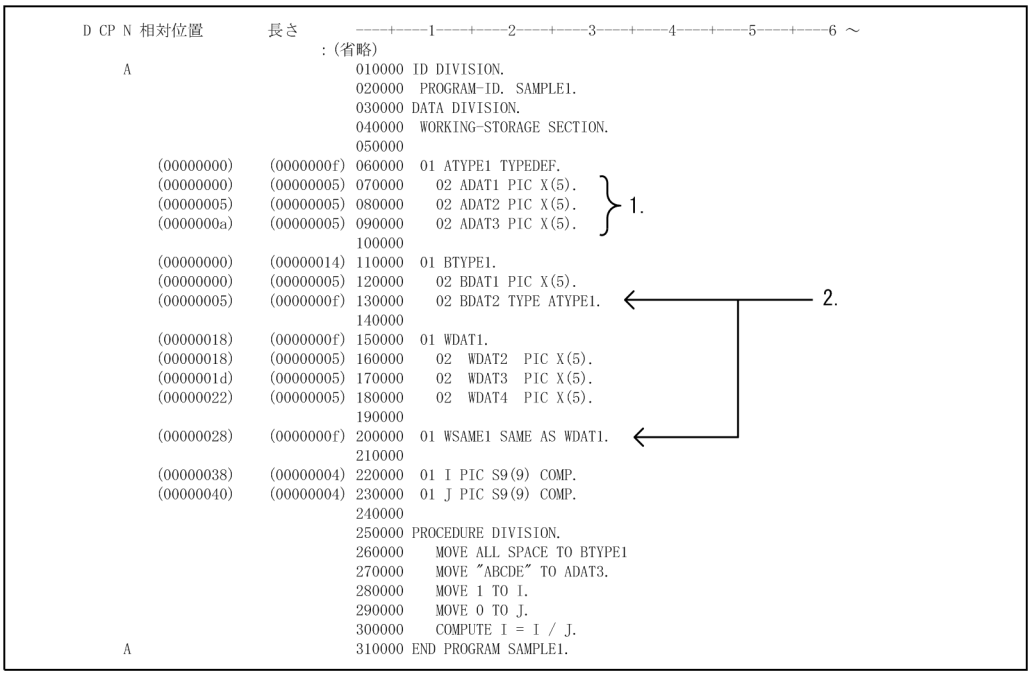
クラス定義の場合は，コンパイルリストでデータ部の先頭からの相対位置を表示しているため，データ領域ダンプリストも領域の先頭からの相対位置で参照できます。

■ TYPE 句／SAME AS 句を含むデータ名の従属項目

TYPE 句または SAME AS 句を含むデータ名の従属項目は，コンパイルリストに表示されません。そのため，従属項目の相対位置を参照したい場合は，ユーザ自身で TYPE 句または SAME AS 句を含むデータ名の従属項目の相対位置を求めてください。

TYPE 句／SAME AS 句の定義がある原始プログラムリストを次に示します。

図 D-12　TYPE 句／SAME AS 句の定義がある原始プログラムリスト



```
D CP N 相対位置        長さ        ----+----1----+----2----+----3----+----4----+----5----+----6 ～
                                 :(省略)
     A                            010000 ID DIVISION.
                                  020000  PROGRAM-ID. SAMPLE1.
                                  030000 DATA DIVISION.
                                  040000  WORKING-STORAGE SECTION.
                                  050000
          (00000000)     (0000000f) 060000  01 ATYPE1 TYPEDEF.
          (00000000)     (00000005) 070000    02 ADAT1 PIC X(5).   }
          (00000005)     (00000005) 080000    02 ADAT2 PIC X(5).   } 1.
          (0000000a)     (00000005) 090000    02 ADAT3 PIC X(5).   }
                                  100000
          (00000000)     (00000014) 110000  01 BTYPE1.
          (00000000)     (00000005) 120000    02 BDAT1 PIC X(5).
          (00000005)     (0000000f) 130000    02 BDAT2 TYPE ATYPE1.  <-- 2.
                                  140000
          (00000018)     (0000000f) 150000  01 WDAT1.
          (00000018)     (00000005) 160000    02  WDAT2  PIC X(5).
          (0000001d)     (00000005) 170000    02  WDAT3  PIC X(5).
          (00000022)     (00000005) 180000    02  WDAT4  PIC X(5).
                                  190000
          (00000028)     (0000000f) 200000  01 WSAME1 SAME AS WDAT1.
                                  210000
          (00000038)     (00000004) 220000  01 I PIC S9(9) COMP.
          (00000040)     (00000004) 230000  01 J PIC S9(9) COMP.
                                  240000
                                  250000 PROCEDURE DIVISION.
                                  260000    MOVE ALL SPACE TO BTYPE1
                                  270000    MOVE "ABCDE" TO ADAT3.
                                  280000    MOVE 1 TO I.
                                  290000    MOVE 0 TO J.
                                  300000    COMPUTE I = I / J.
     A                            310000 END PROGRAM SAMPLE1.
```

1. 従属項目

   TYPEDEF 句を含むデータ名の従属項目を示します。

2. TYPE 句を含むデータ名の従属項目

   暗黙的に TYPEDEF 句を含むデータ名の従属項目が展開されます。ただし，SAME AS 句も TYPE 句と同様にコンパイルリストには表示されません。

TYPE 句を含む「データ名：BDAT2」の従属項目は，TYPEDEF 句を含むデータ名の従属項目である三つのデータ名（"ADAT1"/"ADAT2"/"ADAT3"）になります。TYPEDEF 句で定義したときの相対位置は，TYPEDEF 句を含むデータ名の相対位置を 0 として表示します。そのため，TYPE 句を含むデータ名の従属項目となるデータ名の相対位置は，次の計算式で求めてください。

計算式

参照する TYPE 句を含むデータ名の従属項目であるデータ名の相対位置=TYPE 句を含むデータ名の相対位置＋参照する TYPEDEF 句を含むデータ名の従属項目の相対位置

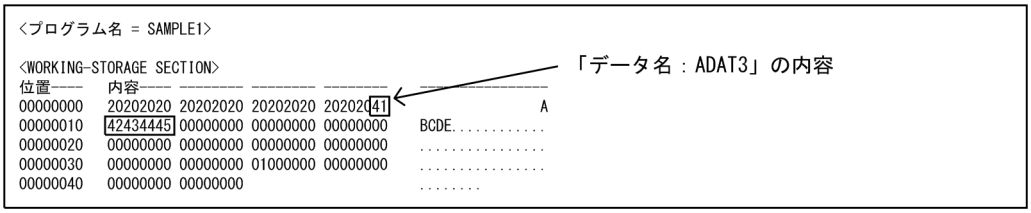
例：「データ名：ADAT3」の場合

「データ名：ADAT3」の相対位置=（0x0000005）＋（0x0000000a）

=（0x0000000f）

=15 バイト

上記の図に対応したデータ領域ダンプリストの内容を次に示します。




```
<プログラム名 = SAMPLE1>

<WORKING-STORAGE SECTION>
位置----   内容---- -------- -------- -------- --------  ----------------
00000000   20202020 20202020 20202020 20202041                       A
00000010   42434445 00000000 00000000 00000000  BCDE............
00000020   00000000 00000000 00000000 00000000  ................
00000030   00000000 00000000 01000000 00000000  ................
00000040   00000000 00000000                    ........
```

データ領域ダンプリストの<WORKING-STORAGE SECTION>の先頭から（0x0000000f）（15 バイト）目の内容を 5 バイト参照することで,「データ名：ADAT3」の内容を確認できます。

(5) エラーリスト

エラーリストの出力形式を次に示します。リスト 1 行当たりのカラム数は 80 カラムです。

図 D-13 エラーリストの出力形式

```
"../test/SAMPLE1.cbl", line 30: KCCC1601C-W PROCEDUREの直後にDIVISIONがありません。DIVISION.を仮定します。
        1.                2.       3.  4.                              5.
```

1. エラーがある原始プログラムのファイル名
   COPY 文で入力したプログラムにエラーがあるときは，そのファイル名が出力されます。
2. エラーがある原始プログラム内の行番号（原始プログラムの先頭行からの相対行数）
3. メッセージ番号
4. エラーレベルの種類
   I：情報提示レベルのメッセージ
   W：警告エラー
   S：重大エラー
   U：回復不能エラー
5. エラーメッセージの本文

付録 E　COBOL で使用するファイル

付録 E.1　COBOL2002 で使用するファイル

COBOL2002 で使用するファイルの一覧を，次に示します。

表 E-1　COBOL2002 で使用するファイル

拡張子	ファイル種別	内容	出力元	入力先
.a	ライブラリファイル	ライブラリを格納するファイル。	リンカ	−
.cbl ほか※1	COBOL ソースファイル	COBOL 原始プログラムを格納するファイル。	−	コンパイラ
.cbl※4	XML アクセス用データ定義	XML ドキュメントへのアクセスに必要な COBOL のデータ定義を格納するファイル。	XML アクセス用 COBOL ソース生成コマンド	コンパイラ
.cbl※4	XML アクセス用ステータス定義	XML アクセスルーチンのステータス名称を定義する登録集原文。	−	コンパイラ
.cbl※4	XML アクセスルーチン	XML ドキュメントにアクセスする COBOL 副プログラムを格納するファイル。	XML アクセス用 COBOL ソース生成コマンド	コンパイラ
.cbl※2※3	COBOL UAP 引数定義ファイル	Cosminexus 連携機能から呼び出したい COBOL UAP 引数を定義したファイル。	−	Cosminexus 連携機能
.cbp	プログラム情報ファイル	テストデバッグ情報やカバレージ情報を格納するファイル。-TDInf または-CVInf オプションを指定してコンパイルしたときに出力される。	コンパイラ	コンパイラ，テストデバッガ
.cbs	擬似プログラム用プログラム情報ファイル	-SimMain，-SimSub，-SimIdent のどれかのオプションを指定してコンパイルしたときに出力されるプログラム情報ファイル。	コンパイラ	テストデバッガ
.class※3	Cosminexus 上 Java 実行ファイル	Cosminexus 連携機能を呼び出す Java 実行ファイル。	−	Cosminexus 連携機能
.cll	カバレージ情報リストファイル	カバレージ情報をリスト形式にして格納するファイル。	カバレージ	−
.cnl	カウント情報リストファイル	カウント情報をリスト形式にして格納するファイル。	カバレージ	−
.cno	実行結果出力ファイル（カウント情報の表示）	プログラムからの連動実行によるカウント情報を表示するとき，実行結果とトラブルシュート情報を出力するファイル。	カバレージ	−

拡張子	ファイル種別	内容	出力元	入力先
.cvo	実行結果出力ファイル（カバレージ情報の蓄積）	プログラムからの連動実行によるカバレージ情報を蓄積するとき，実行結果とトラブルシュート情報を出力するファイル。	カバレージ	−
.cxc※4	カタログファイル	公開識別子とファイルの対応づけを定義するファイル。	−	XML アクセス用 COBOL ソース生成コマンド XML アクセス用実行時ライブラリ
.cxd※4	XML データ定義ファイル	XML ドキュメントの要素と COBOL のデータ構造とのマッピングを記述したファイル。	−	XML アクセス用 COBOL ソース生成コマンド
.DAT	データレコードファイル	索引順編成ファイルで，実際にレコードを格納しているファイル。	ISAM	ISAM
.DEF	キー定義ファイル	索引順編成ファイルで，データファイルとキーファイルとの対応を表すデータを格納するファイル。	ISAM	ISAM
.dtd※4 または.xml※4	XML 文書型定義ファイル	XML ドキュメントのマークづけ要素とその構造を定義したファイル。	−	XML アクセス用 COBOL ソース生成コマンド
.env	CGI 環境ファイル	CGI 用環境変数を格納するファイル。	ライブラリ	ライブラリ
拡張子規定なし	実行可能ファイル	コンパイル，リンクをして実行可能になったプログラムを格納するファイル。	リンカ	−
.htm または .html	HTML ファイル	Web ページに出力する内容を格納するファイル。	−	HTML トランスレータ
.java※3	COBOL アクセス用 Bean（Java ソースファイル）	Cosminexus 連携機能が出力する COBOL アクセス用 Bean を格納するファイル。	Cosminexus 連携機能	−
.java※3	EJB 関連 Java ソースファイル	Cosminexus 連携機能が出力する EJB 用ホームインタフェース，リモートインタフェース，Enterprise Bean を格納するファイル。	Cosminexus 連携機能	−
.K01	主キーファイル	索引順編成ファイルで，主レコードキーでレコードを検索するためのファイル。	ISAM	ISAM

拡張子	ファイル種別	内容	出力元	入力先
.K02～.K99	副キーファイル	索引順編成ファイルで，副レコードキーでレコードを検索するためのファイル。	ISAM	ISAM
.lst	コンパイルリストファイル	コンパイルリストを格納するファイル。-SrcList オプションを指定してコンパイルしたときに出力される。	コンパイラ	－
.o	オブジェクトファイル	コンパイルの結果であるオブジェクトプログラムを格納するファイル。	コンパイラ	リンカ
.rep	リポジトリファイル	プログラム定義，関数定義，クラス定数，およびインタフェース定義の中で規定された情報を格納するファイル。	コンパイラ	コンパイラ
.sl .so	共用ライブラリファイル	共用ライブラリファイル。	リンカ	－
.tdi	TD コマンド格納ファイル（中断点情報）	中断点情報の TD コマンドを格納するファイル。-TestCmd,Full または-TestCmd,Break オプションを指定してコンパイルしたときに出力される。	コンパイラ	テストデバッガ
.tdo	実行結果出力ファイル（テストデバッグ）	プログラムからの連動実行によるテストデバッグを実行するとき，実行結果とトラブルシュート情報を出力するファイル。	テストデバッガ	－
.tds	TD コマンド格納ファイル（シミュレーション情報）	使用するシミュレーション情報の TD コマンドを格納するファイル。-TestCmd,Sim オプションを指定してコンパイルしたときに出力される。	コンパイラ	テストデバッガ
.trc	トレース情報ファイル	CGI プログラムの作成を支援するサービスルーチンの，トレース情報を格納するファイル。	ライブラリ	－
.xml※3	デプロイ情報（DD ファイル）	Cosminexus 連携機能が出力する EJB 用デプロイ情報を格納するファイル。	Cosminexus 連携機能	－

（凡例）

－：該当しない

注

出力元，入力先欄の略称の意味は次のとおりです。

ライブラリ：実行時ライブラリ

ISAM：ISAM ユティリティ

注※1

目的に応じて次の拡張子を使用します。

- 固定形式正書法で書かれた原始プログラムをコンパイルする場合
  .cbl，.CBL，.cob，.ocb，または環境変数 CBLFIX で指定した拡張子
- 自由形式正書法で書かれた原始プログラムをコンパイルする場合
  .cbf，.ocf，または環境変数 CBLFREE で指定した拡張子

注※2

目的に応じて次の拡張子を使用します。

- 固定形式正書法で書かれた COBOL 引数定義ファイルの場合
  .cbf，または環境変数 CBLFIX で指定した拡張子を除く，すべての拡張子
- 自由形式正書法で書かれた COBOL 引数定義ファイルの場合
  .cbf，または環境変数 CBLFREE で指定した拡張子

注※3

Cosminexus 連携機能で使用します。Cosminexus 連携機能の詳細は，マニュアル「COBOL2002 Cosminexus 連携機能ガイド」を参照してください。

注※4

XML 連携機能で使用します。XML 連携機能の詳細は，マニュアル「COBOL2002　XML 連携機能ガイド」を参照してください。

付録 E.2　COBOL プログラムの実行時に必要なファイル

COBOL プログラムの実行時に必要なファイルの一覧を，次に示します。

表 E-2　COBOL プログラムの実行時に使用するファイル

拡張子	ファイル種別	ファイルが必要な条件
.sl，.so	共用ライブラリ	呼ばれるプログラムを共用ライブラリとして作成している場合
規定なし	実行可能ファイル	必須
.DRF	データレコードファイル	ISAM による索引編成ファイルを使用する場合
.KDF	キー定義ファイル	
.K01	主キーファイル	
.K02～.K99	副キーファイル	

付録 F　コンパイラの制限値

コンパイラの制限値，限界値を次に示します。

表 F-1　コンパイラの制限値，限界値

区分	項目	制限値，限界値
全般規定	語の長さ	31 文字
	英数字定数，日本語文字定数の長さ	1～160 文字※15
	固定小数点数字定数のけた数	1～18 けた※10
	浮動小数点数字定数のけた数	仮数部 1～16 けた 指数部 2 けた
	単精度浮動小数点数字定数の値の範囲（絶対値）	約 $10^{-37.9}$ 以上約 $10^{38.5}$ 以下，または 0
	倍精度浮動小数点数字定数の値の範囲（絶対値）	約 10^{-308} 以上約 10^{308} 以下，または 0
	連結式で連結できる定数の長さ	2～1,024 文字
基本機能	作業場所節のデータ項目の長さ	16,777,215 バイト（クラス定義） 1,073,741,823 バイト（上記以外）
	局所場所節のデータ項目の長さ	1,073,741,823 バイト
	ファイル節のデータ項目の長さ	16,777,191 バイト※1（クラス定義のファイル節 FD 記述項） 1,073,741,799 バイト※1（上記以外のファイル節 FD 記述項） 65,535 バイト（ファイル節 SD 記述項）
	連絡節のデータ項目の長さ	16,777,215 バイト（BY VALUE,RETURNING 項目） 1,073,741,823 バイト（上記以外）
	上記以外の節のデータ項目の長さ	16,777,215 バイト
	1 レコードに含まれる可変長項目の制御変数	100 個
	PICTURE 文字列の長さ	30 文字※14
	作業場所節の英字項目のけた数	16,777,215 けた（クラス定義） 1,073,741,823 けた（上記以外）
	局所場所節の英字項目のけた数	1,073,741,823 けた
	ファイル節の英字項目のけた数	16,777,191 バイト※1（クラス定義のファイル節 FD 記述項） 1,073,741,799 バイト※1（上記以外のファイル節 FD 記述項） 65,535 バイト（ファイル節 SD 記述項）

区分	項目	制限値，限界値
	連絡節の英字項目のけた数	16,777,215 けた（BY VALUE,RETURNING 項目） 1,073,741,823 けた（上記以外）
	上記以外の英字項目のけた数	16,777,215 けた
	外部 10 進，内部 10 進項目のけた数	18 けた※11
	2 進項目のけた数	18 けた
	外部浮動小数点数字項目のけた数	仮数部 1～16 けた 指数部 2 けた
	作業場所節の英数字，英数字編集項目のけた数	16,777,215 けた（クラス定義） 1,073,741,823 けた（上記以外）
	局所場所節の英数字，英数字編集項目のけた数	1,073,741,823 けた
	ファイル節の英数字，英数字編集項目のけた数	16,777,191 バイト※1（クラス定義のファイル節 FD 記述項） 1,073,741,799 バイト※1（上記以外のファイル節 FD 記述項） 65,535 バイト（ファイル節 SD 記述項）
	連絡節の英数字，英数字編集項目のけた数	16,777,215 けた（BY VALUE,RETURNING 項目） 1,073,741,823 けた（上記以外）
	上記以外の英数字，英数字編集項目のけた数	16,777,215 けた
	数字編集項目のけた数	249 けた (数字は 18 けた以内※13)
	ブール項目のけた数	2,034 けた
	日本語項目のけた数	16,383 けた
	動的長基本項目の LIMIT に指定できる値	英数字項目の場合 • 16,777,214 けた 日本語項目の場合 • 16,382 けた
	外部浮動小数点数字項目の値の範囲（絶対値）	約 $10^{-37.9}$ 以上約 $10^{38.5}$ 以下，または 0
	条件文の入れ子	1,000 重
	うち PERFORM 文の入れ子	100 重
	PERFORM 文の VARYING 指定で変化させられる一意名の数	7 個（AFTER 指定の数は 6 個以内）
	EVALUATE 文の入れ子	50 重
	GO TO DEPENDING 文の一意名のけた数	9 けた

区分	項目	制限値，限界値
	STOP 文の定数の長さ	160 文字※15
	ACCEPT 文で一度に転送できるデータのサイズ	114 バイト（FROM CONSOLE 指定時）
	COPY 文の入れ子	20 重
	原文名の長さ	31 文字※2
	仮原文の原文語	1～322 文字
	演算結果を保証する算術式の中間結果のけた数	30 けた※12
	算術式に含まれる算術演算子	100 個
	作業場所節※3 の節の合計サイズ	16,777,215 バイト（クラス定義） 1,073,741,823 バイト（上記以外）
	局所場所節の節の合計サイズ	1,073,741,823 バイト
	ファイル節※3※4 の節の合計サイズ	16,777,215 バイト（クラス定義） 1,073,741,823 バイト（上記以外）
	その他の節（連絡節を除く）の合計サイズ	16,777,215 バイト
	算術式中に含まれる算術演算子	100 個
表操作機能	作業場所節の OCCURS 句の整数 2 の値，添字の値，部分参照の値	16,777,215（クラス定義） 1,073,741,823（上記以外）
	局所場所節の OCCURS 句の整数 2 の値，添字の値，部分参照の値	1,073,741,823
	ファイル節の OCCURS 句の整数 2 の値，添字の値，部分参照の値	16,777,191（クラス定義のファイル節 FD 記述項） 1,073,741,799（上記以外のファイル節 FD 記述項） 65,535（ファイル節 SD 記述項）
	連絡節の OCCURS 句の整数 2 の値，添字の値，部分参照の値	16,777,215（BY VALUE,RETURNING 項目）※5 1,073,741,823（上記以外）
	上記外の節の OCCURS 句の整数 2 の値，添字の値，部分参照の値	16,777,215
	OCCURS 句に指定できる指標名の数	12 個
	OCCURS 句に定義した指標名に設定できる最大値	10 けた（クラス定義以外の作業場所節，クラス定義以外のファイル節 FD 記述項，局所場所節，BY VALUE／RETURNING 項目以外の連絡節） 9 けた（BY VALUE／RETURNING 項目の連絡節および上記以外の節）
	OCCURS 句の次元数	7 次元

区分	項目	制限値，限界値
	OCCURS KEY の個数	12 個
	SEARCH 文の入れ子	15 重
入出力機能	RECORD CONTAINS CHARACTERS 句で指定できる値	16,777,191※1 （クラス定義のファイル節 FD 記述項） 1,073,741,799※1 （上記以外のファイル節 FD 記述項） 65,535 （ファイル節 SD 記述項）
	BLOCK CONTAINS 句で指定できる値	16,777,191※1 （クラス定義のファイル節 FD 記述項） 1,073,741,799※1 （上記以外のファイル節 FD 記述項）
	WRITE ADVANCING で指定できる整数	0～99
	RECORD KEY 句で指定できるキーの最大長	255 バイト
	ALTERNATE RECORD KEY 句で指定できるキーの最大長	255 バイト
	ALTERNATE RECORD KEY 句で指定できるキーの個数	98 個
	LINAGE 句で指定できる整数	1～32,767
	LINAGE FOOTING で指定できる整数	0～32,767
	LINAGE TOP で指定できる整数	0～32,767
	LINAGE BOTTOM で指定できる整数	0～32,767
	RECORD IS VARYING 句の FROM に指定できる整数	1～16,777,191※1 （クラス定義のファイル節 FD 記述項） 1～1,073,741,799※1 （上記以外のファイル節 FD 記述項） 1～65,535 （ファイル節 SD 記述項）
	RECORD IS VARYING 句の TO に指定できる整数	1～16,777,191※1 （クラス定義のファイル節 FD 記述項） 1～1,073,741,799※1 （上記以外のファイル節 FD 記述項） 1～65,535 （ファイル節 SD 記述項）
	ファイルのブロック長に指定できる値	1～16,777,191※1 （クラス定義） 1～1,073,741,799

区分	項目	制限値，限界値
		（上記以外）
	CSV 編成ファイルの READ／WRITE 文で扱える基本項目の最大数	AIX(32)，Linux(x86)の場合 • 2,729 個（-NumCsv オプション指定ありの場合） • 5,459 個（-NumCsv オプション指定なしの場合） AIX(64)，Linux(x64)の場合 • 2,047 個（-NumCsv オプション指定ありの場合） • 4,094 個（-NumCsv オプション指定なしの場合）
	ASSIGN 句のデータ名 1 に指定できるデータ項目の最大長	32,767 バイト
プログラム間連絡機能	プログラム名，利用者定義関数名，クラス名，インタフェース名，メソッド名の長さ	31 文字（定数指定でない場合）
	プログラム名，メソッド名を定数指定する場合の長さ	160 バイト※16
	-Main,V3 オプションで引数を受け取るときに指定できる最大長	100 バイト
	FD EXTERNAL 句の数（プログラム単位，実行時単位）	255 個
	01 EXTERNAL 句の数（外部属性）	実行単位内で 32,767 個 プログラム単位で 32,767 個※6
	入れ子のプログラムの個数	65,535 個
	入れ子のプログラムのネストの深さ	255 レベル
	ENTRY 文の個数	32,767 個
	CALL 文，INVOKE 文の BY VALUE，BY CONTENT，RETURNING に指定されたデータ項目のサイズ	16,777,215 バイト
	利用者定義関数の引数として指定されたデータ項目のサイズ（ただし，対応する仮引数に BY VALUE が指定または仮定される場合，および返却値だけ）	16,777,215 バイト
データベースアクセス機能	SQL 文の長さ	65,528 バイト※8
	SQL 文で参照する埋め込み変数の長さ	COBOL2002 のデータ項目と同じ
	SQL 文で記述する語，定数の長さ	COBOL2002 の語，定数と同じ
整列併合機能	SORT 文，MERGE 文の入力ファイル数	12 個
	整列併合用キーの指定個数	64 個

区分	項目	制限値，限界値
	整列併合用キーのサイズの合計	4,080 バイト
	整列併合用レコードのサイズ	65,535 バイト
オブジェクト指向機能	クラス定義中に記述可能なメソッドの数	65,530※7
その他	コンパイル可能なソース行数	999,999 行
	手続き名の個数	1,048,575 個
	ファイルの数	255 個
	自由形式正書法での 1 行のバイト数	255 バイト
	COPY 文 PREFIXING 指定および SUFFIXING 指定に指定できる語の長さ	30 文字
	一つの実行単位が使用できるスタックサイズの上限	システムのスタックサイズの設定値に依存
	COBOL2002 で扱えるファイル名の長さの上限	パス名を含めて 255 バイト※9

注

ソース行数以外の制限値は独立していないため，すべての制限値が同時に適用されるとは限りません。

注※1

索引ファイルは，割り当てるファイルによって次のようになります。

- ISAM を使用する場合：65,503 バイト
- HiRDB による索引編成ファイルを使用する場合：1,073,741,799 バイト
  ただし，CREATE TABLE 定義の指定に依存する。

注※2

-V3Rec または-CompatiV3 オプション（VOS3 COBOL85 互換のオプション）を指定した場合，先頭の 8 文字だけが有効となります。

注※3

-MultiThread オプション指定時，ファイル節，作業場所節の合計の上限は，16,777,215 バイトとなります。

注※4

ファイル節の合計サイズは，次のように計算します。

Σ（各記述項のレコード長＋ 24+α）

α：レコード間の境界調整で生じる遊びバイト

レコード間は 8 バイト境界調整される。

```
FILE SECTION.
FD FILE1.
01 A1.                    }
  02 A2 PIC X(10).       } レコード長10
                         } 遊びバイト6
SD FILE2.
01 B1.                    }
  02 B2 PIC X(10).       } レコード長10
```

ファイル節の合計サイズ=（A1のレコード長+24+遊びバイト）+（B1のレコード長+24）
=40＋34
=74バイト

注※5

添字の値，および部分参照の値の場合，上限は 1,073,741,823 となります。

注※6

ただし，プログラム単位では，アドレスデータ項目の個数や連絡節の 01 レベルデータ項目の個数によって，上記の最大数まで指定できないことがあります。

注※7

クラス定義に含まれるファクトリ定義，オブジェクト定義のデータ記述項に PROPERTY 句が指定されている場合は次のようになります。

クラス定義中に記述可能なメソッドの数

＝ 65,530 -（SET,GET 指定がない PROPERTY 句の数）×2 -（SET,GET 指定がある PROPERTY 句の数）

注※8

SQL 文の長さは，次の項目の合計値になります。

- SQL 文を構成する語，定数の長さ（「EXEC SQL」と「END-EXEC」は含めない）
- 語と定数の間は，分離符の空白文字 1 バイトを加算する。
  （語と定数の間に，注記行や複数の空白文字があっても，常に空白文字 1 バイトとして扱う）

ただし，動的 SQL の長さは，コンパイラではチェックしません。

注※9

パス名を含めないファイル名の長さの上限も 255 バイトです。

使用できるファイル名の長さが機能により個別に規定されている場合は，その長さが上限となります。

また，指定されたファイル名は内部的に絶対パス名に変換することがありますが，このとき内部的に処理できる絶対パス名の長さの上限は，1,023 バイトです。

注※10

-MaxDigits38 オプションを指定した場合，けた数は 1～38 けたとなります。

注※11

-MaxDigits38 オプションを指定した場合，けた数の上限は 38 けたとなります。

注※12

-IntResult,DecFloat40 オプションを指定した場合，けた数は 40 けたとなります。

注※13

AIX(64)，Linux(x64)の場合，-MaxDigits38 オプションの指定がある場合，数字は 38 けた以内となります。

注※14

AIX(64)，Linux(x64)の場合，-MaxDigits38 オプションを指定したとき，PICTURE 文字列の長さは 60 文字となります。

注※15

-LiteralExtend,Alnum オプションを指定したとき，英数字定数の長さは 1～8,191 文字（バイト）となります。

注※16

-LiteralExtend,Alnum オプションを指定した場合，プログラム名およびメソッド名を定数指定するときの長さは，1～1,024 文字（バイト）となります。

付録 G　入出力状態の値

入出力状態の値が表す内容を，次に示します。

表 G-1　入出力状態の値

条件	入出力状態の値	順編成ファイル，相対編成ファイル，索引編成ファイルの場合	HiRDB による索引編成ファイルの場合	COBOL 入出力サービスルーチンの場合
成功	00	入出力文の実行が成功した。その入出力動作に関しては，これ以上情報がない。	○	○
	02	入出力文の実行は成功したが，重複キーが検出された。 1. READ 文で読み込んだレコードキーの参照キーの値が，その索引での次のレコードの参照キーと等しい。 2. REWRITE 文，または WRITE 文によって指定されたレコードの副レコードキーの値が，すでにファイルにある。ただし，この副レコードキーは重複した値が許されているので，誤りではない。	×	×
	04	READ 文の実行は成功した。しかし，処理したレコードの長さが，そのファイルのファイル固有属性に従っていない（テキスト編成ファイルだけ）。 または，CSV 編成ファイルから入力した一つのセルの文字列長が，入力領域より長い。	−	×
	05	OPEN 文の実行は成功した。しかし，OPEN 文を実行したとき，参照した不定ファイルはない。OPEN 文のモードが入出力両用，または拡張ならば，ファイルは生成される。	○ （入力モードの場合だけ）	×
	07	入出力文の実行は成功した。ただし，REEL／UNIT 指定の CLOSE 文で参照したファイルが非リール／ユニットの媒体上にある（順編成ファイルだけ）。	−	×
ファイル終了	10	順アクセスの READ 文を実行しようとしたが，次のレコードがない。次の場合のどちらかである。 1. ファイルの終わりに達した。 2. 存在しない不定入力ファイルに対して，順アクセスの READ 文を初めて実行しようとした。	○	○ （1.の場合だけ）
	14	相対編成ファイルに対して順アクセスの READ 文を実行しようとしたが，相対レコード番号の有効けた数がそのファイルの相対キーデータ項目のサイズよりも大きい（相対編成ファイルだけ）。	−	○ （相対編成ファイルの場合だけ）
無効キー	21	順アクセスで，順序誤りがあった。成功した READ 文と次の REWRITE 文の間で主レコードキーの値が変更されたか，または WRITE 文の実行時に主レコードキーの値が昇順になっていない。	○	×
	22	重複キーがあった。 1. すでにレコードがある位置に，重ねて WRITE 文でレコードを書き出そうとした。	○ （2.の場合だけ）	○

条件	入出力状態の値	順編成ファイル，相対編成ファイル，索引編成ファイルの場合	HiRDB による索引編成ファイルの場合	COBOL 入出力サービスルーチンの場合
		2. WRITE 文，または REWRITE 文で索引編成ファイルに書き出そうとしたレコードの主レコードキー，または副レコードキーの値が，すでにファイル中にある。ただし，この副レコードキーには，DUPLICATES が指定されていない。		（相対編成ファイルで 1.の場合だけ）
	23	次の場合のどちらかである。 1. 物理ファイル中に存在しないレコードをランダムに呼び出そうとした。 2. 存在しない不定入力ファイルに対して START 文，または乱アクセスの READ 文を，実行しようとした。	○	○ （相対編成ファイルの場合だけ）
	24	次の場合のどれかである。 1. COBOL の外部で定義されたファイルの区域外に書き出そうとした。区域の定め方は，作成者が決める。 2. 順アクセスの WRITE 文を実行しようとしたが，そのレコードの相対レコード番号の有効けた数がそのファイルの相対キーデータ項目より大きい。 3. 乱アクセスの WRITE 文を実行しようとしたが，その相対レコード番号が 0 以下である。	○ （1.の場合だけ）	○ （相対編成ファイルの場合だけ）
永続誤り	30	永続誤りがある。次のどれかである。 1. 相対可変長ファイルのレコード領域が破壊されている。 2. 相対編成ファイルのスロットが破壊されている。有効，無効レコードの識別ができない。 3. 特定のファイルを指定し，OPEN 文が成功したが，そのあと入出力矛盾が発生した（ファイル破壊を検出した場合など）。 4. 相対編成ファイルで，最大レコード長および最小レコード長の範囲を超えている。	ファイル位置指示子が不定となっている。例えば，COMMIT 文，ROLLBACK 文の実行によってファイル位置指示子が有効なレコードを指していない。	○ （相対編成ファイルで，1.，2.の場合だけ）
	34	区域外書き出しによる永続誤りがある。 COBOL の外部で定義されたファイルの区域外にレコードを書き出そうとした（順編成ファイルだけ）。	−	○
	35	不定ファイルでないファイルがないとき，INPUT 指定，I-O 指定，または EXTEND 指定を持つ OPEN 文を実行しようとしたことによる永続誤りがある。	○	○
	37	OPEN 文を実行しようとしたが，指定されたファイルが，その OPEN 文で指定されたモードを使えない。 次の場合のどれかである。 1. EXTEND 指定，または OUTPUT 指定が指定されていたが，そのファイルでは出力動作が使えない。 2. I-O 指定が指定されていたが，そのファイルでは，入出力両用モードで開かれている索引編成ファイルに対して許される入出力動作が使えない。	×	○ （1.，2.，および 3.の場合だけ）

条件	入出力状態の値	順編成ファイル，相対編成ファイル，索引編成ファイルの場合	HiRDB による索引編成ファイルの場合	COBOL 入出力サービスルーチンの場合
		3. INPUT 指定が指定されたが，そのファイルでは入力動作が使えない。 4. OUTPUT 指定以外が指定されたが，書式印刷機能でプリンタの出力動作が使えない。		
	38	施錠して閉じたファイルに対して，OPEN 文を実行しようとしたことによる永続誤りがある。	○	×
	39	ファイル固有属性とプログラム中で指定した属性との間で矛盾が検出されたため，OPEN 文が不成功になった。	×	○
論理誤り	41	開かれているファイルに OPEN 文を実行しようとした。	○	○
	42	開かれていないファイルに CLOSE 文を実行しようとした。	○	○
	43	順アクセスで，DELETE 文，または REWRITE 文を実行する前に，関連するファイルに対して実行された最後の入出力文が，成功した READ 文でない。	○	○
	44	次のどちらかの条件で，区域外書き出しが起こった。 1. 関連するファイル名の RECORD IS VARYING 句によって許される最大のレコードより大きい，または最小のレコードより小さいレコードを書き出そうとした，または書き換えようとした。 2. レコードを書き換えようとしたが，そのレコードは，書き換えられるレコードと同じサイズでない。	○ （1.の場合だけ）	○※1
	46	入力モードまたは入出力両用レコードで開かれたファイルに順アクセスの READ 文を実行しようとしたが，有効な次のレコードがない。次の場合のどれかである。 1. 先行する START 文が不成功であった。 2. 先行する READ 文が，ファイル終了条件によって不成功であった。 3. 先行する READ 文が，ファイル終了条件を引き起こした。	○	○ （1.は相対編成ファイルの場合だけ）
	47	入力モード，または入出力両用モードで開かれていないファイルに，READ 文，または START 文を実行しようとした。	○	○
	48	入出力両用モード，出力モード，または拡張モードで開かれていないファイルに，WRITE 文を実行しようとした。	○	○
	49	入出力両用モードで開かれていないファイルに，DELETE 文，または REWRITE 文を実行しようとした。	○	○
作成者規定	90	入出力文の実行時，実行できない状態になった。 次のどれかである。 1. 入出力に必要なメモリが不足した。 2. ファイルにディレクトリが指定された。 3. CBL_外部装置名で示される環境変数がない。 4. 1 プロセス内でオープンできるファイル数の上限を超えた。	○※2	○※3

条件	入出力状態の値	順編成ファイル，相対編成ファイル，索引編成ファイルの場合	HiRDB による索引編成ファイルの場合	COBOL 入出力サービスルーチンの場合
		5. CLOSE 文実行時，バッファに残っていた内容をファイルに書き出すとき，大容量の記憶ファイルの許容範囲を超えた。 6. ファイル名指定に誤りがある。 7. ラージファイル入出力機能の環境変数指定とファイル属性情報，またはファイルシステムの属性情報とで矛盾がある。 8. ラージファイル入出力機能を使用していない場合にファイルサイズが 2GB を超えて入出力を実行した。 9. ISAM による索引編成ファイルのクローズ時のディスク書き込み保証で使用する関数アドレスの取得に失敗した。		
	92	入出力文の妥当でない使い方によって実行が不成功になった。 1. レコード長が DEPENDING ON 指定で指定されている場合，指定されたデータ名中の値が数字以外である。 2. レコード番号が処理できる最大値を超えた（相対編成ファイルだけ）。 3. 開かれていないファイルに UNLOCK 文を実行しようとした。	○ （1.，および 3.の場合だけ）	○※4
	93	次の場合のどちらかである。 1. OPEN 文を実行しようとしたが，該当ファイルまたは該当ファイル中のレコードはすでに使用されている。 2. 書式印刷機能を使用するとき，プリンタはすでに使用されている。	×	○ （1.の場合だけ）
	98	該当せず。	出力モード指定，または不定ファイルに対応する入出力両用，拡張モード指定の OPEN 文を実行しようとしたが，ファイルがなかった。	×
	99	入出力文（OPEN 文，CLOSE 文を除く）を実行しようとしたが，該当レコードはすでに使用されている。	×	○

（凡例）

○：標準仕様と同じ

×：値は返されない

－：該当しない

注※1

2.，および次の場合に該当する。

- ファイルオープン時に指定した最大のレコードより大きい，または最小のレコードより小さいレコードを書き出そうとした，または書き換えようとした。

注※2

1.～6.，および次の場合に該当する。

- 副レコードキーの個数が，98 を超えている。
- データが不正である。

注※3

1.，2.，4.，5.，6.，7.，8.，および次の場合に該当する。

- サービスルーチンに指定したパラメタインタフェース領域に誤りがある。
- サービスルーチンで取り扱えないファイル名を指定した。
- COBOL 実行時ライブラリが組み込まれていない。
- 入出力のためのシステム関数でエラーが発生した。

注※4

2.，3.，および次の場合に該当する。

- 指定できないファイル編成，およびアクセス法で CBLSTART サービスルーチンを実行した。
- 指定できないファイル編成，およびアクセス法で NEXT 指定の CBLREAD サービスルーチンを実行した。
- 指定できないファイル編成，およびアクセス法で KEY 指定の CBLREAD サービスルーチンを実行した。
- 指定できないファイル編成で CBLDELETE サービスルーチンを実行した。
- 順ファイルに対して LOCK MODE MANUAL の指定をした。
- 開かれていないファイルに対して CBLWDISK サービスルーチンを実行した。

付録 H　COBOL85 と COBOL2002 のコンパイラオプションの対応

COBOL85 の ccbl コマンドと，COBOL2002 の ccbl2002 コマンドでの，コンパイラオプションの対応関係を，次に示します。

項番	ccbl コマンドのオプション名	ccbl2002 コマンドのオプション名	機能
1	-Ai	-NumAccept	ACCEPT 文に数字項目を指定できるようにする。
2	-B1	-Bin1Byte	1 バイトの 2 進項目を有効にする（PICTURE 句の指定で 2 けたまでは 1 バイトとして扱う）。
3	-Bb※1	-BigEndian,Bin	2 進データ項目をビッグエンディアン形式で処理する。
	-Fb※1	-BigEndian,Float	浮動小数点データ項目をビッグエンディアン形式で処理する。
4	-Bd	-DynamicLink,Call	プログラムの呼び出しで，動的なリンク（ダイナミックリンク）を行う。
	-Bs	-DynamicLink,IdentCall	一意名指定の CALL 文でプログラムを呼び出す場合，動的なリンク（ダイナミックリンク）を行う。定数指定の CALL 文は，静的なリンク（スタティックリンク）となる。
5	-C1	-Compile,CheckOnly	コンパイル時，オブジェクトファイル，実行可能ファイルを出力しない。
	-C2	-Compile,NoLink	コンパイル時，実行可能ファイルを出力しない。
6	-Ci	-NumCsv	CSV 編成ファイルで，セルデータを数値として入出力できるようにする。
7	-Cm	-StdMIA,13	MIA1.3 仕様の範囲外チェックをする。
	-Cy	-StdMIA,14	MIA1.4 仕様の範囲外チェックをする。
	-Cu	-Std85,High	JIS 仕様の範囲外チェックをする。
	-Cj	-Std85,High,Obso	JIS 仕様の範囲外と廃要素をチェックする。
	-Cv	-Std85,Middle	JIS 仕様の中位集合範囲外（上位集合，JIS 仕様の範囲外）チェックをする。
	-Cw	-Std85,Low	JIS 仕様の下位集合範囲外（中位集合，上位集合，JIS 仕様の範囲外）チェックをする。
	-Cr	-Std85,Report	JIS 仕様の報告書作成機能をチェックする。
8	-Cs	-MainNotCBL	すべて副プログラムとして作成する。
9	-d	-DebugLine	デバッグ行を有効にする。
10	-Dz	-MinusZero	10 進項目で負の符号を持つゼロを正の符号を持つゼロに変換する。

項番	ccbl コマンドの オプション名	ccbl2002 コマンドの オプション名	機能
11	-Ec※2	-Switch,EBCDIC	字類条件を EBCDIC に指定する。
	-Ek※2	-Switch,EBCDIK	字類条件を EBCDIK に指定する。
	−	-Switch,xxxxx,Unprintable※2	英数字の照合順序を指定したコードに切り替える。また，1 バイトで印字できないコードも，照合順序として固有の文字位置に割り付ける（xxxxx には，EBCDIC または EBCDIK を指定する）。
	−	-Switch,xxxxx,noApplyJpnItem※2	日本語項目を-Switch オプションの適用対象外とする（xxxxx には，EBCDIC または EBCDIK を指定する）。
12	-F8	-Cblctr	CBL-CTR 特殊レジスタを使用できるようにする。
13	-Gm	-SimMain	主プログラムをシミュレーションする。
	-Gs	-SimSub	副プログラムをシミュレーションする。
14	-H8	-H8Switch	HITAC8000 シリーズの仕様でコンパイルする。
15	-Hd	-IgnoreLCC	行送り制御文字を無視する。
16	-Hf※5	-V3Rec,Fixed	メインフレーム（VOS3）の固定長レコード形式のプログラムを，VOS3 の日本語文字の扱いに合わせてコンパイルする。
	-Hv※5	-V3Rec,Variable	メインフレーム（VOS3）の可変長レコード形式のプログラムを，VOS3 の日本語文字の扱いに合わせてコンパイルする。
17	−	-V3RecFCSpace	空白に関する機能キャラクタの扱いをメインフレームと同等にする。
18	−	-V3RecEased,QuoteCheck	-V3Rec オプション指定時の仕様チェックを緩和する（定数の分離符のチェック）。
	−	-V3RecEased,WordCheck	-V3Rec オプション指定時の仕様チェックを緩和する（語または PICTURE 文字列の制限値のチェック）。
19	-i	-IdentCall	動的なリンクでの呼び出しをシミュレーションする。
20	-K1※2	-EucPosition	EUC 環境下で，見かけ上の文字位置で固定形式正書法の境界を決定するときに指定する。
21	-Ks※3	-XMAP,LinePrint	書式印刷機能を使用して，順編成ファイルをプリンタに出力する。
22	-M1	-CmAster	1 カラム目が'*'の行を注記行とする。
23	-Md	-CmDol	7 カラム目が'$'の行を注記行とする。
24	-Mw	-Main,System	先頭の最外側プログラムを主プログラムとして作成する。引数の形式は，システム固有の argc，argv 形式とする。

項番	ccbl コマンドのオプション名	ccbl2002 コマンドのオプション名	機能
	-Mh	-Main,V3	先頭の最外側プログラムを主プログラムとして作成する。引数の形式は，メインフレーム（VOS3）形式とする。
25	-Mt	-MultiThread	マルチスレッド対応 COBOL プログラムを作成する。
26	-Na※2	-JPN,Alnum	日本語項目，日本語編集項目，および日本語文字定数をそれぞれ英数字項目，英数字編集項目，および英数字定数として扱う。
	−	-JPN,V3JPN※2	STRING 文，UNSTRING 文，INSPECT 文を除く日本語項目，日本語編集項目，および日本語文字定数をそれぞれ英数字項目，英数字編集項目，および英数字定数として扱う。
	−	-JPN,V3JPNSpace※2	-CompatiV3 オプション指定時に，日本語項目，日本語編集項目，および日本語文字定数をそのままの属性で扱う。
27	-Nl	-International	インターナショナリゼーション機能を使用するときに指定する。
28	-o	-OutputFile	生成する実行可能ファイル名を指定する。
29	-O0	-Optimize,0	コンパイル時，最適化をしない。
	-O1	-Optimize,1	コンパイル時，文の中で閉じた最適化をする。
	-O2	-Optimize,2	コンパイル時，広域的な最適化をする。
	-O3	-Optimize,3	コンパイル時，広域的な最適化をし，10 進項目を 2 進項目に変換する。
30	−	-SQLDisp※1	埋め込み SQL 文に用途（USAGE 句）が表示用（DISPLAY）のデータ項目を指定できるようにする。
31	−	-SQL,ODBC※1	埋め込み SQL 文を ODBC で使用できるようにする。
	−	-SQL,ODBC,NoCont※1	埋め込み SQL 文を ODBC で使用できるようにする。SQL 構文内の浮動継続指示子を有効としない。
32	-P1	-SrcList,NoCopy	コンパイルリストに情報を出力する。すべての COPY 文の展開を抑止する。
	-P2	-SrcList,CopySup	コンパイルリストに情報を出力する。プログラムに SUPPRESS 指定があるときだけ，COPY 文の展開を抑止する。
	-P3	-SrcList,CopyAll	コンパイルリストに情報を出力する。SUPPRESS 指定を無視して，すべての COPY 文を強制的に展開する。
	−	-SrcList,OutputAll	コンパイルリストに情報を出力する。 COPY 文の SUPPRESS 指定，翻訳指令を無視して，すべての情報を出力する。

項番	ccbl コマンドのオプション名	ccbl2002 コマンドのオプション名	機能
	−	-SrcList,xxxxx,NoFalsePath	コンパイルリストに情報を出力するとき，条件翻訳の無効行を出力しない（xxxxx には，CopyAll，CopySup，または NoCopy を指定する）。
	−	-SrcList,xxxxx,DataLoc	コンパイルリスト（原始プログラムリスト）にデータ項目の相対位置と長さを表示する（xxxxx には，OutputAll，CopyAll，CopySup，または NoCopy を指定する）。
33	-Rd※2	-RDBTran	HiRDB による索引ファイル入出力機能を使用する。
34	-RG	-Profile,Gprof	gprof でのプロファイル用オブジェクトファイルを作成する。
	-Rp※2	-Profile,Prof	prof でのプロファイル用オブジェクトファイルを作成する。
35	-S1	-StdVersion,1	第 1 次規格の解釈でコンパイルする。
	-S2	-StdVersion,2	第 2 次規格の解釈でコンパイルする。
36	-Sz	-PIC,Std	共用ライブラリに使う位置独立（PIC）コードを作成する場合に指定する。
	-SZ	-PIC,Expand	-PIC,Std オプションを指定してリンクエラーになった場合に使用する。
37	-T1	-DebugInf	異常終了時，エラー要約情報を出力する。
	-T2	-DebugInf,Trace	異常終了時，エラー要約情報およびトレースバック情報を出力する。
	-T3	-DebugCompati	実行時に次のチェックをする。 • 添字，指標名の繰り返し回数の範囲外チェック • プログラム間整合性チェック
	-T4	-DebugData	データ例外を検出する。
	−	-DebugData,ValueHex	データ例外検知機能でデータ例外が検出された場合，出力されるエラーメッセージにデータ項目の属性と矛盾している格納値を 16 進数で表示する。
	-T5	-TDInf	テストデバッグ情報を出力する。
	-T6	-CVInf	カバレージ情報を出力する。
	-T7	-DebugRange	添字，指標名の繰り返し回数について，次元ごとの範囲外チェックをする。
	-Tt	-TestCmd,Full	中断点情報，シミュレーション情報の TD コマンドを TD コマンド格納ファイルに出力する。
	-Tb	-TestCmd,Break	中断点情報の TD コマンドを TD コマンド格納ファイルに出力する。

項番	ccbl コマンドのオプション名	ccbl2002 コマンドのオプション名	機能
	-Ts	-TestCmd,Sim	シミュレーション情報の TD コマンドを TD コマンド格納ファイルに出力する。
38	-Un	-EquivRule,NotExtend	拡張コード文字と標準コード文字を等価とみなさない。
	-Ue	-EquivRule,NotAny	拡張コード文字と標準コード文字を等価とみなさない。さらに，標準コードの英大文字と標準コードの英小文字も等価とみなさない。
	−	-EquivRule,StdCode	拡張コード文字と標準コード文字を等価とみなさない。さらに，日本語文字定数中に標準コードの空白を書いてもエラーとしない。
39	-v	-Details	コンパイラオプションの詳細情報を出力する。また，一部の環境変数の内容を出力できる。
40	-V3※5	-CompatiV3	VOS3 COBOL85 との互換機能を有効にする。
41	-Va	-CBLVALUE	環境変数 CBLVALUE を有効にする。
42	-Vx	-BinExtend	2 進データ項目に指定できる初期値を拡張する。
43	-Wl	-Link	リンカに渡すオプションを指定する。
44	-X5	-Comp5	COMP-5 を指定できるようにする。
45	-Xb	-DigitsTrunc	転記文で上位けたを切り捨てる。
46	-Xc	-DoubleQuote	引用符（ " ）を分離符とみなしてコンパイルする。
47	-Xe	-TruncCheck,Binary	転記でのデータ切り捨てをチェックする。さらに，送り出し側作用対象が 2 進項目で，受け取り側作用対象が外部 10 進項目／内部 10 進項目のとき，送り出し側作用対象の 2 進項目は格納可能な最大けた数でチェックする。
	-Xo	-TruncCheck	転記でのデータ切り捨てをチェックする。
48	-Xr	-SQL,XDM	リレーショナルデータベース（XDM/RD）操作シミュレーション機能を使用する。
49	指定なし	-Help	コマンドのヘルプを出力する。
50	−	-SimIdent	副プログラム（一意名 CALL 文）をシミュレーションする。
51	−	-Std2002,OutRange	COBOL2002 規格仕様をチェックする。
	−	-Std2002,Obso	COBOL2002 規格仕様の廃要素をチェックする。
	−	-Std2002,Archaic	COBOL2002 仕様の古典的要素をチェックする。
52	−	-Compati85,IoStatus	入出力状態 1x，2x に対する処理を UNIX COBOL85 と同様にする。

項番	ccbl コマンドのオプション名	ccbl2002 コマンドのオプション名	機能
	−	-Compati85,Linage	LINAGE 値が不正なときの処理を UNIX COBOL85 と同様にする。
	−	-Compati85,Call	CALL 文の ON EXCEPTION，または ON OVERFLOW 指定の無条件文を実行するエラーの条件を UNIX COBOL85 と同様にする。
	−	-Compati85,Power	べき乗演算の精度，エラー時の処理を UNIX COBOL85 と同様にする。
	−	-Compati85,Syntax	コンパイル時の解釈を UNIX COBOL85 と同様にする。
	−	-Compati85,IDParag	見出し部の構文チェックを緩和する。
	−	-Compati85,RsvWord	UNIX COBOL85 の予約語と同じ範囲でコンパイルする。
	−	-Compati85,NoPropagate	プログラムへの伝播を抑止し，オブジェクトファイルサイズを縮小する。
	−	-Compati85,All	-Compati85 オプションのすべてのサブオプションを仮定する。
53	−	-ErrSup,I	I レベルエラーの出力を抑止する。
	−	-ErrSup,W	W レベルエラーの出力を抑止する。
54	−	-Repository,Gen	コンパイル時，リポジトリファイルだけを出力する。
	−	-Repository,Sup	リポジトリファイルがすでにある場合，更新しない。
55	−	-RepositoryCheck	翻訳グループ中の定義と外部リポジトリ中の情報に相違があるかどうかをチェックする。
56	−	-Define	コンパイル時に有効となる，翻訳変数名とその値を定義する。
57	−	-OldForm	UNIX COBOL85 のオプションを指定する。
58	−	-UniObjGen	シフト JIS の COBOL ソースから Unicode のオブジェクトを生成する。
	−	-UniEndian,Little	シフト JIS で記述された日本語文字定数を UTF-16LE に変換する。
	−	-UniEndian,Big	シフト JIS で記述された日本語文字定数を UTF-16BE に変換する。
59	−	-V3Spec	VOS3 COBOL85 に対する UNIX COBOL2002 固有の構文をチェックする。
	−	-V3Spec,CopyEased	VOS3 COBOL85 に対する UNIX COBOL2002 固有の構文をチェックする。ただし，COPY 文，REPLACE 文の一部のエラーチェックをしない。

項番	ccbl コマンドの オプション名	ccbl2002 コマンドの オプション名	機能
60	−	-V3ConvtName	VOS3 COBOL85 からのソースファイル互換のため，COPY 文の原文名定数中の'¥'と'@'を変換する。
61	−	-UscoreStart	先頭が下線の CALL 定数を指定できるようにする。
62	−	-LowerAsUpper	定数指定の CALL に指定された英小文字を英大文字に変換してプログラムを呼び出す。
63	−	-Lx64ConventionCheck※4	Linux(x64)で C 言語との連携を行うときに問題となる可能性のあるコードがあるかどうかをチェックする。
64	−	-MaxDigits38※6	数字項目および数字定数に指定できる最大けた数を 18 けたから 38 けたに拡張する。
65	−	-IntResult,DecFloat40※6	算術演算の中間結果の表現形式を 40 けたの 10 進浮動小数点形式にする。
66	−	-VOSCBL,OccursKey	メインフレーム互換機能を有効にする。OCCURS 句の KEY IS 指定のデータ名の名前の有効範囲を，その OCCURS 句のある記述項または，それに従属する記述項とする。
	−	-VOSCBL,ReportControl	メインフレーム互換機能を有効にする。制御脚書きの印刷中に改ページが起こったとき，ページ頭書きで SOURCE 句によって制御用データ項目を参照している場合，その制御用データ項目の値を元の値で参照する。
	−	-VOSCBL,DataComm	メインフレーム互換機能を有効にする。データコミュニケーション機能の通信記述項で，次のどちらかに該当するときの初期値を，メインフレーム（VOS3/VOS1）の XDM/DCCM データコミュニケーション機能を使用する場合の値とする。 • 通信記述項の句の指定を省略している。 • 通信文の実行前に，通信記述項の句に指定されたデータ名に初期値を設定していない。
	−	-VOSCBL,RedefinesData	メインフレーム互換機能を有効にする。REDEFINES 句の右辺のデータ名が未定義のとき，直前の同一レベル番号のデータ名を REDEFINES 句の右辺として仮定する。
	−	-VOSCBL,AssignDataToDevice	ASSIGN 句のデータ名を外部装置名とみなす。
	−	-VOSCBL,EvaluateWhenOther	EVALUATE 文で WHEN OTHER 指定の直前の無条件文を省略した際に CONTINUE 文を仮定する。
67	−	-LiteralExtend,Alnum	英数字定数と定数指定のプログラム名の長さを拡張する。
68	−	-IgnoreAPPLY,FILESHARE	APPLY FILE-SHARE 句を覚え書きとみなす。
69	−	-PortabilityCheck,Literal	メインフレームと動作が異なる可能性がある定数にお知らせメッセージを出力する。

項番	ccbl コマンドの オプション名	ccbl2002 コマンドの オプション名	機能
	－	-PortabilityCheck,Numeric	メインフレームと動作が異なる可能性がある数字項目，浮動小数点定数にお知らせメッセージを出力する。
70	－	-SpaceAsZero	外部 10 進項目中の空白に対して特殊処理を行う。
71	－	-OldStyleObject※2	オブジェクトの形式を COBOL2002 V3 以前と同様の形式に戻す。
72	－	-CheckUninitData	データ項目の初期化漏れをチェックする。
73	－	-FunctionECSup,CodeConvErr	変換エラーが発生しても例外条件を成立させない。

（凡例）

－：対応するコンパイラオプションがない

注※1

Linux で有効です。

注※2

AIX で有効です。

注※3

AIX(32)で有効です。

注※4

Linux(x64)で有効です。

注※5

コンパイラ環境変数 CBLV3UNICODE に YES を指定したときだけ有効です。

注※6

AIX(64)，Linux(x64)で有効です。

付録 I　サービスルーチンのリソース一覧

リソースの標準値が格納されているアプリケーションリソースファイル名称を，次に示します。

JCPOPUP サービスルーチン

リソースファイル名

AIX(32)の場合：Cbl2002popup

AIX(64)の場合：Cbl2002popup64

リソースファイルのクラス名

AIX(32)の場合：Cbl2002popup

AIX(64)の場合：Cbl2002popup64

リソースファイルの格納場所については，「12.4　リソース一覧」を参照してください。

リソース一覧を次に示します。

リソース名	内容	指定値，標準値
background	背景色	任意 標準値…white
foreground	前景色	任意 標準値…black
showPF1	終了キーメニュー中に［PF1］キーを表示	True…表示する False…表示しない 標準値…True
showPF2	終了キーメニュー中に［PF2］キーを表示	
showPF3	終了キーメニュー中に［PF3］キーを表示	
showPF4	終了キーメニュー中に［PF4］キーを表示	
showPF5	終了キーメニュー中に［PF5］キーを表示	
showPF6	終了キーメニュー中に［PF6］キーを表示	
showPF7	終了キーメニュー中に［PF7］キーを表示	
showPF8	終了キーメニュー中に［PF8］キーを表示	
showPF9	終了キーメニュー中に［PF9］キーを表示	
showPF10	終了キーメニュー中に［PF10］キーを表示	
showPF11	終了キーメニュー中に［PF11］キーを表示	
showPF12	終了キーメニュー中に［PF12］キーを表示	
showPF13	終了キーメニュー中に［PF13］キーを表示	
showPF14	終了キーメニュー中に［PF14］キーを表示	
showPF15	終了キーメニュー中に［PF15］キーを表示	

リソース名	内容	指定値，標準値
showPF16	終了キーメニュー中に［PF16］キーを表示	
showPF17	終了キーメニュー中に［PF17］キーを表示	
showPF18	終了キーメニュー中に［PF18］キーを表示	
showPF19	終了キーメニュー中に［PF19］キーを表示	
showPF20	終了キーメニュー中に［PF20］キーを表示	
showPF21	終了キーメニュー中に［PF21］キーを表示	
showPF22	終了キーメニュー中に［PF22］キーを表示	
showPF23	終了キーメニュー中に［PF23］キーを表示	
showPF24	終了キーメニュー中に［PF24］キーを表示	
PF1.labelString	［PF1］キーのラベル文字列	任意 標準値…PF1
PF2.labelString	［PF2］キーのラベル文字列	任意 標準値…PF2
PF3.labelString	［PF3］キーのラベル文字列	任意 標準値…PF3
PF4.labelString	［PF4］キーのラベル文字列	任意 標準値…PF4
PF5.labelString	［PF5］キーのラベル文字列	任意 標準値…PF5
PF6.labelString	［PF6］キーのラベル文字列	任意 標準値…PF6
PF7.labelString	［PF7］キーのラベル文字列	任意 標準値…PF7
PF8.labelString	［PF8］キーのラベル文字列	任意 標準値…PF8
PF9.labelString	［PF9］キーのラベル文字列	任意 標準値…PF9
PF10.labelString	［PF10］キーのラベル文字列	任意 標準値…PF10（a）
PF11.labelString	［PF11］キーのラベル文字列	任意 標準値…PF11（b）
PF12.labelString	［PF12］キーのラベル文字列	任意 標準値…PF12（c）
PF13.labelString	［PF13］キーのラベル文字列	任意

リソース名	内容	指定値，標準値
		標準値…PF13（d）
PF14.labelString	[PF14] キーのラベル文字列	任意 標準値…PF14（e）
PF15.labelString	[PF15] キーのラベル文字列	任意 標準値…PF15（f）
PF16.labelString	[PF16] キーのラベル文字列	任意 標準値…PF16（g）
PF17.labelString	[PF17] キーのラベル文字列	任意 標準値…PF17（h）
PF18.labelString	[PF18] キーのラベル文字列	任意 標準値…PF18（i）
PF19.labelString	[PF19] キーのラベル文字列	任意 標準値…PF19（j）
PF20.labelString	[PF20] キーのラベル文字列	任意 標準値…PF20（k）
PF21.labelString	[PF21] キーのラベル文字列	任意 標準値…PF21（l）
PF22.labelString	[PF22] キーのラベル文字列	任意 標準値…PF22（m）
PF23.labelString	[PF23] キーのラベル文字列	任意 標準値…PF23（n）
PF24.labelString	[PF24] キーのラベル文字列	任意 標準値…PF24（o）

。

付録 J　各バージョンの変更内容

各バージョンの変更内容を示します。

変更内容（3021-3-602-40）　COBOL2002 Net Server Suite(64) 04-60，COBOL2002 Net Server Runtime(64) 04-60

追加・変更内容
次の製品の適用 OS に「Red Hat Enterprise Linux Server 9 (64-bit x86_64)」を追加した。 • P-9W36-1241　COBOL2002 Net Server Suite(64) • P-9W36-2241　COBOL2002 Net Server Runtime(64)
Linux(x64) COBOL2002 で次に示す関数をサポートした。 • 文字コード変換組み込み関数（CONVERT-CODE 関数）
Linux コンテナに COBOL2002 を取り込む方法，Linux コンテナ起動時に COBOL プログラムを実行する方法を追加した。

変更内容（3021-3-602-30）　COBOL2002 Net Server Suite 04-41，COBOL2002 Net Server Runtime 04-41，COBOL2002 Net Server Suite(64) 04-41，COBOL2002 Net Server Runtime(64) 04-41

追加・変更内容
コンパイルリストに 80 カラムを超えるソースコードを表示する機能（環境変数 CBLFIXEDFORMLINE）をサポートした。
REDEFINES 句の右辺のデータ名が未定義のときに，直前の同一レベル番号のデータ名を仮定するオプション（-VOSCBL,RedefinesData オプション）をサポートした。
初期化漏れチェック機能の判定条件を改善した。

変更内容（3021-3-602-20）　COBOL2002 Net Server Suite 04-20，COBOL2002 Net Server Runtime 04-20，COBOL2002 Net Server Suite(64) 04-20，COBOL2002 Net Server Runtime(64) 04-20

追加・変更内容
次の製品の適用 OS に「Red Hat Enterprise Linux Server 8 (64-bit x86_64)」を追加した。 • P-9W36-1241　COBOL2002 Net Server Suite(64) • P-9W36-2241　COBOL2002 Net Server Runtime(64)
データ項目の初期化漏れを検出できる初期化漏れチェック機能（-CheckUninitData オプション）をサポートした。

変更内容（3021-3-602-10）　COBOL2002 Net Server Suite 04-10，COBOL2002 Net Server Runtime 04-10，COBOL2002 Net Server Suite(64) 04-10，COBOL2002 Net Server Runtime(64) 04-10

追加・変更内容
-Switch オプションで Unprintable サブオプションが無効な場合に，印字できないコードの照合順序の説明を変更した。

変更内容（3021-3-602） COBOL2002 Net Server Suite 04-00，COBOL2002 Net Server Runtime 04-00，COBOL2002 Net Server Suite(64) 04-00，COBOL2002 Net Server Runtime(64) 04-00

追加・変更内容
次に示す適用 OS がサポート対象外となった。 • AIX V6.1 • Red Hat Enterprise Linux 5 (AMD/Intel 64) • Red Hat Enterprise Linux 5 Advanced Platform (AMD/Intel 64) • Red Hat Enterprise Linux Server 6 (64-bit x86_64)
データコミュニケーション機能の通信記述項で，指定を省略した句の初期値を，メインフレーム(VOS3/VOS1)の XDM/DCCM データコミュニケーション機能を使用する場合の値とするオプションをサポートした（-VOSCBL,DataComm）。
コンパイルとリンクを同時に実行する場合，静的な参照関係のないオブジェクトをリンク対象とするために，ccbl2002 コマンドで指定する-bgcbypass リンカオプションを仮定するオプションをサポートした（-OldLinkOpt,GCBypass）。
makefile 生成機能で次のマクロ定義をサポートした。 • DEST • INSTALL • LD • MAKEFILE • PRINT • SHELL • LIBRARY
次の組み込み関数で日本語 EUC 環境に対応した。 • COUNT-CHAR 関数 • LENGTH-OF-SUBSTRING 関数 • SUBSTRING 関数

付録 K　このマニュアルの参考情報

このマニュアルを読むに当たっての参考情報を示します。

付録 K.1　関連マニュアル

このマニュアルの関連マニュアルを次に示します。必要に応じてお読みください。

- COBOL2002　使用の手引　操作編（3021-3-603）
- COBOL2002　言語　標準仕様編　（3021-3-604）
- COBOL2002　言語　拡張仕様編　（3021-3-605）
- COBOL2002　Cosminexus 連携機能ガイド　（3021-3-606）
- COBOL2002　XML 連携機能ガイド　（3021-3-608）
- COBOL2002　メッセージ　（3021-3-609）
- 索引順編成ファイル管理 ISAM　（3000-3-169）
- ソートマージ　（3000-3-151）
- ソートマージ　（3020-3-N73）
- 日立コード変換ユーザーズガイド　（3000-7-415）※1
- 日立コード変換ユーザーズガイド　（3020-7-351）※1
- Hitachi Code Converter（UNIX 編）（3020-7-358）※1
- 画面・帳票サポートシステム XMAP3 Server　（3000-7-508）
- XMAP3 Version 5 画面・帳票サポートシステム XMAP3 概説　（3020-7-511）
- XMAP3 Version 5 画面・帳票サポートシステム XMAP3 開発ガイド　（3020-7-512）
- XMAP3 Version 5 画面・帳票サポートシステム XMAP3 プログラミングガイド　（3020-7-513）
- XMAP3 Version 5 画面・帳票サポートシステム XMAP3 実行ガイド　（3020-7-514）
- HiRDB Version 9 解説（3020-6-450）※2
- HiRDB Version 9 システム導入・設計ガイド（UNIX(R)用）（3000-6-452）
- HiRDB Version 9 システム定義（UNIX(R)用）（3000-6-453）
- HiRDB Version 9 システム運用ガイド（UNIX(R)用）（3000-6-454）
- HiRDB Version 9 コマンドリファレンス（UNIX(R)用）（3000-6-455）※3
- HiRDB Version 9 UAP 開発ガイド（3020-6-456）※4
- HiRDB Version 9 SQL リファレンス（3020-6-457）※5
- HiRDB Version 9 メッセージ（3020-6-458）

- HiRDB Version 10 解説（3020-6-551）※2
- HiRDB Version 10 システム導入・設計ガイド（UNIX(R)用）（3020-6-552）
- HiRDB Version 10 システム定義（UNIX(R)用）（3020-6-554）
- HiRDB Version 10 システム運用ガイド（UNIX(R)用）（3020-6-556）
- HiRDB Version 10 コマンドリファレンス（UNIX(R)用）（3020-6-558）※3
- HiRDB Version 10 UAP 開発ガイド（3020-6-560）※4
- HiRDB Version 10 SQL リファレンス（3020-6-561）※5
- HiRDB Version 10 メッセージ（3020-6-562）

注※1

このマニュアルでは，「コード変換ライブラリのマニュアル」と表記しています。

注※2

このマニュアルでは，「HiRDB の解説マニュアル」と表記しています。

注※3

このマニュアルでは，「HiRDB のコマンドリファレンスマニュアル」と表記しています。

注※4

このマニュアルでは，「HiRDB の UAP 開発ガイドマニュアル」と表記しています。

注※5

このマニュアルでは，「HiRDB の SQL リファレンスマニュアル」と表記しています。

付録 K.2　このマニュアルでの表記

このマニュアル中では，「このシステム」と表現している場合，「COBOL2002」を示しています。

また，このマニュアルは，製品種別によって相違点があります。本文中での製品種別ごとの表記を次に示します。

マニュアルでの表記			該当する製品の形名
UNIX または UNIX COBOL2002	AIX	AIX(32)または AIX(32) COBOL2002	P-1M36-1141 P-1M36-2141
		AIX(64)または AIX(64) COBOL2002	P-1M36-1241 P-1M36-2241
	Linux	Linux(x86)または Linux(x86) COBOL2002	—※

マニュアルでの表記			該当する製品の形名
		Linux(x64)または Linux(x64) COBOL2002	P-9W36-1241 P-9W36-2241

注※

該当する製品の形名の詳細は，「リリースノート」でご確認ください。

また，このマニュアルでは各製品を次のように表記しています。

マニュアルでの表記			製品名
UBI			Red Hat Universal Base Image
UNIX	AIX		AIX V7.1
			AIX V7.2
			AIX 7.3
	Linux	Linux Server 7 (64-bit x86_64)	Red Hat Enterprise Linux Server 7 (64-bit x86_64)
		Linux Server 8 (64-bit x86_64)	Red Hat Enterprise Linux Server 8 (64-bit x86_64)
		Linux Server 9 (64-bit x86_64)	Red Hat Enterprise Linux Server 9 (64-bit x86_64)
XMAP3			XMAP3 Server
			XMAP3 Developer Version 5
			XMAP3 Server Runtime Version 5
			XMAP3 Client Runtime Version 5

- XMAP3 の製品を区別する必要がある場合は，それぞれの製品名称を表記しています。
- 日立 COBOL2002 のことを日立 COBOL2002，または単に COBOL2002 と表記しています。なお，COBOL2002 のバージョンを区別する必要がある場合は，次のように表記しています。

マニュアルでの表記	該当する製品
COBOL2002 V3 以前	COBOL2002（バージョン 04-00 未満）
COBOL2002 V4	COBOL2002（バージョン 04-00 以降）

- 日立 COBOL85 のことを日立 COBOL85，または単に COBOL85 と表記しています。また，プラットフォームを明確にする必要がある場合は，「PC COBOL85」「VOS3 COBOL85」のように表記しています。
- 特に断り書きがない場合，プログラム定義，関数定義，およびメソッド定義を総称して「プログラム」と表記しています。

- リストのヘッダ部に表示される COBOL2002 の識別記号を次のように表記しています。識別記号以外については，「付録 D.2　リストの見方」を参照してください。

```
COBOL2002 (c)  VV-RR  *** CCC･･･CCC *** YYYY-MM-DD HH:MM:SS
          |
      識別記号
```

識別記号（c）	内容
A と表示された場合	AIX(32)であることを示します。
B と表示された場合	AIX(64)であることを示します。
J と表示された場合	Linux(x86)であることを示します。
K と表示された場合	Linux(x64)であることを示します。

- コンパイラオプションの説明では，次の表記を使用します。

「XXX オプション」，または単に「XXX」とオプション名が表記されている場合

　XXX オプションについて，サブオプションの組み合わせを含む，すべての場合を意味します。

「XXX,YYY オプション」，または単に「XXX,YYY」とサブオプションを含めたオプション名が表記されている場合

　XXX,YYY オプションだけの場合を意味します。

「XXX コンパイラオプション」と表記されている場合

　リンカオプションなど，ほかのオプションと明確に区別する必要がある場合を意味します。

(例 1)

　「-Compile オプション」または「-Compile」と記載している場合，-Compile オプションのサブオプションの組み合わせすべて（-Compile,CheckOnly／-Compile,NoLink）を意味します。

(例 2)

　「-Compile,CheckOnly オプション」または「-Compile,CheckOnly」と記載している場合，-Compile,CheckOnly だけを意味します。

付録 K.3　KB（キロバイト）などの単位表記について

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

付録 L　用語解説

(記号)

-Main オプションが指定されたプログラム

アプリケーションの主プログラム（main 関数を持つプログラム）となる COBOL プログラムのこと。COBOL2002 では，-Main,System または-Main,V3 オプションを指定してコンパイルすると，-Main オプションが指定されたプログラムを作成できる。

(英字)

BASE クラス（BASE class)

COBOL2002 で使用できる標準クラス。BASE クラスには，オブジェクトの生成や初期化などのメソッドが定義されている。BASE クラスのサブクラスならば，どのクラスでも BASE クラスの機能が利用できる。

BOM（バイトオーダマーク)

ファイルの先頭に付加された，Unicode の表現形式を表す情報。

COBOL2002 では，テキスト編成ファイルに対してこの情報を付加する。本文中では，Unicode シグニチャと表記する。

CGI（Common Gateway Interface)

Web サーバの機能を拡張するための API のこと。CGI を使用すると，Web サーバとほかのプログラムを連携させて Web クライアントに複雑なサービスを使用できる。

CSV 編成ファイル

CSV（Comma Separated Value）形式でデータが記述されたファイルのこと。表計算プログラムファイルともいう。

CSV 形式とは，表計算プログラムやリレーショナルデータベースでデータを扱えるテキストデータの形式をいう。データの区切りをコンマ（,），レコードの区切りを改行で表す。レコードは可変長形式になる。コンマで区切られた個々のデータは，セルと呼ばれる。

HTML（Hyper Text Markup Language)

インターネット経由で閲覧できる Web ページを作成するための言語。テキストファイル中にタグと呼ばれる制御文字を埋め込むと，文字書式の設定や画像の埋め込み，ほかのページへのリンクなどを表現できる。

IVS（Ideographic Variation Sequence/Selector）

漢字を表す Unicode の直後に Variation Selector と呼ばれるコードを付加し，漢字の「異体字」を表現する方法のこと。

COBOL2002 では，用途が DISPLAY および NATIONAL の項目で使用する。

SELF

メッセージの受け手のオブジェクトを参照するための名前のこと。SELF が参照するオブジェクトを SELF オブジェクトという。メソッド中で使用される SELF は，そのメソッドの呼び起こし対象のオブジェクト自身のことを表す。

SUPER

SELF と同様に，メッセージの受け手のオブジェクトを参照するための名前のこと。SUPER は，INHERITS 句が指定されたクラスのメソッド実装定義中だけで使用できる。SUPER が参照するオブジェクトは，SELF が参照するオブジェクトだが，メソッドの検索方法が SELF の場合と異なる。メソッド実装定義中に SUPER を指定すると，そのメソッド実装定義が含まれるクラスのスーパクラスを起点に，メソッドが検索される。

TD コマンド

テストデバッグで使用するコマンド。

UCS-2（Universal multi-octet Character Set 2）

符号化文字集合の一つの形式。1 文字を 2 バイトで表現する。

COBOL2002 では，用途が DISPLAY および NATIONAL の項目でサポートする。

UCS-4（Universal multi-octet Character Set 4）

符号化文字集合の一つの形式。1 文字を 4 バイトで表現する。

COBOL2002 では，用途が DISPLAY および NATIONAL の項目で UCS-4 の範囲をサポートする。

Unicode

1 バイトでは表現できない文字セットを含めあらゆる文字セットをサポートした文字コード。代表的な符号化文字集合として UCS-2，UCS-4 がある。代表的なエンコーディングスキーマとして UTF-8，UTF-16 がある。

COBOL2002 では，UCS-4 の範囲（UCS-2 の範囲を含む）をサポートする。

本文中では，符号化文字集合，およびエンコーディングスキーマを文字コードと表記する。

Unicode シグニチャ

Unicode の表現形式（UTF-16LE／UTF-16BE／UTF-8），またはそれを識別するための情報。

UTF-16(16-bit UCS Transformation Format)

エンコーディングスキーマの一つの形式。一つのコード単位を 2 バイトとし，1 文字を 1 コード単位（2 バイト），または 2 コード単位（4 バイト）で表現する。UTF-16 では 2 バイトのコード単位の 1 バイト目を先に書くビッグエンディアン形式（UTF-16BE）と 1 バイト目を後に書くリトルエンディアン形式（UTF-16LE）がある。

COBOL2002 では，用途が NATIONAL の項目で UCS-4 の範囲（UCS-2 の範囲を含む）をサポートする。

UTF-8（8-bit UCS Transformation Format)

エンコーディングスキーマの一つの形式。ASCII 文字を 1 バイト，日本語文字を 3~8 バイト，半角かたかなを 3 バイトで表現する。

COBOL2002 では，用途が DISPLAY の項目で使用する。

（ア行）

アプリケーションの主プログラム

main 関数を持つプログラムのこと。

COBOL2002 では，-Main,System または-Main,V3 オプションを指定してコンパイルすると，アプリケーションの主プログラムとなる COBOL プログラムを作成できる。また，他言語で作成したアプリケーションの主プログラムから，COBOL プログラムを呼び出して利用することもできる。

入れ子のプログラム

原始プログラムが入れ子構造になっている場合の，内側のプログラムのこと。内側のプログラム，または内部プログラムともいう。

インスタンスオブジェクト（instance object)

あるクラスに属するオブジェクトのこと。「BASE クラス」のファクトリオブジェクトのメソッド「NEW クラス」の呼び起こしによって生成され，実行単位の終了か，ガーベジコレクションによって消滅させられる。

インスタンスデータ

インスタンスオブジェクトが独自に持つデータのこと。インスタンスメソッドによって操作される。

インスタンスメソッド

インスタンスデータを操作するためのメソッド。

インタフェース（interface）

メッセージの送り手から見えるオブジェクトの部分のこと。

例えば，現金支払い機を一つのオブジェクトとすると，「引き出し」「預け入れ」などのメニューがインタフェースに当たり，そのサービスを実現するための内部的な仕組みが実装に当たる。インタフェースは，メソッド原型の集合である。つまり，どのようなメソッド原型を持つかによって，オブジェクトが持つインタフェースの型が決まる。インタフェースは，クラスとは別に定義できる。

内側のプログラム

原始プログラムが入れ子構造になっている場合の，内側のプログラムのこと。入れ子のプログラム，または内部プログラムともいう。

オブジェクト（object）

データとそのデータを操作するメソッド（手続き）を一体化させたモジュール。オブジェクトは，ある特定の仕事を受け持ち，所定のメッセージに対してメソッドを動作させることで問題を処理する。オブジェクト指向のプログラムでは，オブジェクト同士がメッセージをやり取りすることによって，問題が処理される。

オブジェクト参照データ項目

オブジェクトを参照するためのデータ。オブジェクト参照は，データ項目に，USAGE IS OBJECT REFERENCE 句を指定したもの。

オブジェクト指向（Object Orientation）

現実にある「もの」（オブジェクト）同士のやり取りをモデル化し，それをそのままコンピュータの世界でも実現しようとする考え方のこと。

オブジェクトの消滅（object destruction）

ファクトリオブジェクト，インスタンスオブジェクトを使用できない状態にすること。ファクトリオブジェクトは実行単位の終了によって消滅させられる。また，インスタンスオブジェクトは，実行単位の終了か，ガーベジコレクションによって自動的に消滅させられる。

オブジェクトの生成（object creation）

オブジェクトを使用できる状態にすること。オブジェクトはプログラムの実行時に生成される。COBOL2002 のオブジェクト指向機能では，「BASE クラス」のファクトリオブジェクトのメソッド「NEW メソッド」を呼び起こすことでオブジェクトが生成される。

オブジェクトプロパティ（object property）

データ項目に PROPERTY 句を書くか，METHOD-ID に PROPERTY 指定を書くと，そのデータ項目は別のクラスやプログラムから直接参照できるようになる。このデータ項目は，オブジェクト中のほかの属性（INVOKE 文による呼び起こしによってだけ参照される）に対して，特性（オブジェクトプロパティ）であると宣言される。

オプション

コンパイラやコマンドへの動作を指示するためのもの。

（カ行）

ガーベジコレクション（garbage collection）

実行単位のどこからも参照されなくなったオブジェクトを，COBOL2002 のオブジェクト指向機能が検索し，自動的に消滅させる仕組みのこと。

外部スイッチ

ハードウェアまたはソフトウェアの装置であって，作成者によって定義，命名され，二者択一の状態のどちらかであることを示すために使用される。

外部プログラム

原始プログラムが入れ子構造になっている場合の，最も外側のプログラムのこと。このマニュアルでは，外部プログラムのことを最外側のプログラムと呼ぶ。

外部変数

C 言語で，関数外で宣言され，どの関数からでも参照できる変数（グローバル変数）のこと。外部変数は，プログラム実行中に常に同じ場所に配置され，値を保持し続ける。

カウント情報

プログラム中の文の実行回数をカウントして数値で表したもの。

型（type）

ある集合の中の個々を，何によって識別するかを明示するもの。例えば，プログラムを構成する要素をデータごとに識別する場合は，データ型となる。

活性化されるプログラム

CALL 文の対象となるプログラムで，実行時に呼び出し元プログラムと組み合わされて 1 個の実行単位となる。このマニュアルでは，活性化されるプログラムのことを呼び出し先のプログラムと呼ぶ。

活性化するプログラム

CALL 文を実行してほかのプログラムを呼ぶプログラム。このマニュアルでは，活性化するプログラムのことを呼び出し元のプログラムと呼ぶ。

カバレージ情報

テストの進捗状況を数値で表したもので，C0 メジャー情報，C1 メジャー情報，S1 メジャー情報，およびこれらの差分情報がある。

カプセル化（encapsulation）

データとそのデータを操作するメソッドから成るオブジェクトを，外部から隠ぺいすること。カプセル化によって個々のオブジェクトの独立性が高まり，データまたはプログラム間の相互依存性が減少するので，ほかのデータやモジュールへの影響を心配しないで，処理内容の変更や修正ができるようになる。

環境変数

コンパイラ環境へのオプションを変数として設定や実行可能ファイルおよびその実行環境へのオプションを変数として設定しておくもの。

既定義オブジェクト参照

COBOL2002 のオブジェクト指向機能で，言語仕様としてあらかじめ定義された一意名のことで，SELF，SUPER，NULL，および EXCEPTION-OBJECT がある。それぞれは決まったオブジェクトを参照する。

共通例外処理

COBOL2002 で新しく追加された新機能の例外処理のこと。

クラス（class）

ある共通の性質を持つオブジェクトを一つのグループにまとめて定義したもの。クラスでは，それに属するオブジェクトのデータフォーマットおよびメソッドを定義することで，そのクラスに属するオブジェクトの型を規定する。

継承（inheritance）

スーパクラスから，その性質（メソッド）をサブクラスが受け継ぐ仕組みのこと。COBOL2002 のオブジェクト指向機能では，INHERITS 句を指定することでスーパクラスから，その性質を継承できる。一つのクラスの性質を継承することを単純継承という。これに対して，複数のスーパクラスの性質を継承することを多重継承という。また，多重継承の結果，同じクラスを重複して継承することをダイヤモンド継承という。

コンパイル

原始プログラムを翻訳すること。

COBOL プログラムのコンパイルは，ccbl2002 コマンドで実行する。

（サ行）

最外側のプログラム

原始プログラムが入れ子構造になっている場合の，最も外側のプログラムのこと。外部プログラムともいう。

サブクラス（subclass）

クラスの階層関係の中で，継承する側のクラスのこと。

サロゲート

UTF-16 の拡張で，2 つのコード単位（4 バイト）で 1 文字を表す機能。この 2 つのコード単位の組み合わせのことをサロゲートペアと呼ぶ。

COBOL2002 では，用途が NATIONAL の項目で使用する。

シグナル

ハードウェア，およびソフトウェアで発生する割り込みのこと。

実行可能プログラム

呼び出し元プログラムと呼び出し先プログラムをコンパイル，リンクし，一つの実行できるプログラムにしたもの。

実行時要素

プログラム定義，メソッド定義，および関数定義の総称。このマニュアルでは，実行時要素のことをプログラムと呼ぶ。

スーパクラス（superclass）

クラスの階層関係の中で，継承される側のクラスのこと。

制御プログラム

このシステムの OS（オペレーティングシステム）のことを指す。

（タ行）

ダイヤモンド継承（repeated inheritance，diamond inheritance）

多重継承の結果，同じクラスを直接的または間接的に重複して継承すること。

多重継承（multiple inheritance）

複数のクラスをスーパクラスとして継承すること。

単純継承（simple inheritance）

一つのクラスをスーパクラスとして継承すること。

定数長拡張機能

英数字定数の定数長を拡張できるようにする機能。英数字定数の長さの限界値を 160 文字（バイト）から 8,191 文字（バイト）に拡張できる。

定数長を拡張するプログラムのコンパイル時には，-LiteralExtend,Alnum コンパイラオプションを指定する。

適合（conformance）

メッセージの送り先のオブジェクトが適切かどうかを調べるための概念。適合は，オブジェクトが持つインタフェースに基づいてチェックされる。あるインタフェース B を持つオブジェクトに対するメッセージが，別のインタフェース A を持つオブジェクトでも対応できる場合，インタフェース A はインタフェース B に適合しているという。インタフェース A がインタフェース B に適合する場合，インタフェース B に適合するインタフェース型のオブジェクトが使用できれば，インタフェース A に適合するインタフェース型のオブジェクトも使用できる。適合は，コンパイル時または実行時にチェックされる。これによって，オブジェクトの誤用が避けられるだけでなく，同じメッセージを異なるクラスのオブジェクトに送れるようになる。

動的長基本項目（dynamic-length elementary item）

実行時に長さを変更できるデータ項目（英数字項目または日本語項目）のこと。DYNAMIC LENGTH 句で定義する。動的長基本項目に新しい値が格納されると，項目の長さは自動的に調整される。

動的長基本項目は格納する値によって長さが変わる。想定以上にデータ項目の長さが長くなることを防ぐため，データ項目の最大長は LIMIT で指定できる。

登録集原文

COBOL プログラム中でよく利用される標準化した手続き，ファイル記述，レコード記述，または完全な一つのプログラムなどを，コンパイルするプログラムとは別のファイルに登録したもの。

（ナ行）

内部プログラム

原始プログラムが入れ子構造になっている場合の，内側のプログラムのこと。このマニュアルでは，内部プログラムのことを内側のプログラム，または入れ子のプログラムと呼ぶ。

日本語 EUC

UNIX 系 OS で標準的に使われる文字コードの一つ。ASCII 文字を 1 バイト，半角かたかなを 2 バイト，日本語文字を 2 バイトや 3 バイトで表現する。

日本語集団項目（national group item）

用途が NATIONAL で字類および項類が日本語となる集団項目。日本語集団項目となるデータ項目には，明示的にまたは暗黙的に GROUP-USAGE IS NATIONAL の指定が必要。

日本語集団項目の従属項目は，日本語項目，日本語編集項目および日本語集団項目だけとなる。

（ハ行）

バイトオーダ

2 バイト以上のデータの記録を行なう順序のこと。例えば，0x1234 のデータを 0x1234 のように最上位のバイトから順番に記録する方式をビッグエンディアン，0x3412 のように最下位のバイトから順番に記録する方式をリトルエンディアンという。2 バイトの UTF-16 は，バイトオーダを意識する。

標準クラス

COBOL2002 のオブジェクト指向機能で，言語仕様としてあらかじめ定義されたクラスのこと。BASE クラスがある。

表要素

表中の繰り返す項目の組に属する 1 個のデータ項目。

ファクトリオブジェクト

一つのクラス中のすべてのオブジェクトが共有するデータ（ファクトリデータ）とそれを操作するためのメソッド（ファクトリメソッド）から成るオブジェクト。ファクトリオブジェクトは，実行単位の開始によって生成され，実行単位が終了すると消滅させられる。

ファクトリデータ

一つのクラス中のすべてのオブジェクトが共有するデータのこと。ファクトリメソッドによって操作される。

ファクトリメソッド

ファクトリデータを操作するためのメソッド。

プログラム

プログラム定義，メソッド定義，および関数定義の総称。実行時要素ともいう。

ポリモルフィズム（polymorphism）

一般的には，ある一つの指示に対して異なるさまざまな動作をとれる特徴を指す。オブジェクト指向では，同じメッセージでも受け取るオブジェクトのクラスが異なれば，動作が異なる特徴のことをいう。

（マ行）

マルチスレッド

プログラム内の仕事を，スレッドという単位に分けて実行する方式。複数のプログラムを並列に実行できる。

見た目幅

Unicode 機能の組み込み関数で使用する，文字の見た目の幅。半角文字の幅は 1，全角文字の幅は 2 として扱う。

Unicode 機能の組み込み関数で，文字を見た目幅で数える場合に使用する。

メインフレーム

大規模な業務システムなどに用いられる汎用大型コンピュータ。

メソッド（method）

オブジェクトで使用できるインタフェース（メソッド原型という）とそのサービスの実装を定義した手続きのこと。オブジェクト間のやり取りは，すべてメソッドを通して行われる。メソッドには，ファクトリメソッドおよびインスタンスメソッドの 2 種類がある。

メソッド原型（method prototype）

メソッドを定義する際，メソッドで使用できるインタフェースを定義した部分。メソッド原型は，メソッド名と，パラメタおよび戻り値から成る。

メソッドの呼び起こし（method invocation）

オブジェクトに対してメソッドを呼び起こすこと。COBOL2002 では，INVOKE 文によってメソッドを呼び起こす。

メッセージ（message）

オブジェクトに対して送られる要求のこと。メッセージは，「受け手」「メソッド」「パラメタ」から成る。COBOL2002 のオブジェクト指向機能では，INVOKE 文によってメッセージをオブジェクトに送り，適切なメソッドを呼び起こす。

（ヤ行）

呼び出し先プログラム

CALL 文の対象となるプログラムで，実行時に呼び出し元プログラムと組み合わされて 1 個の実行単位となる。活性化されるプログラムともいう。

呼び出し元プログラム

CALL 文を実行してほかのプログラムを呼ぶプログラム。活性化するプログラムともいう。

（ラ行）

リポジトリ段落（repository）

構成節中に指定する。リポジトリ段落に指定したクラス名，インタフェース名，および利用者定義関数名は，そのリポジトリ段落を含んでいるプログラム定義，クラス定義，インタフェース定義，または関数定義中で有効となる。

リポジトリファイル（repository file）

クラス定義，インタフェース定義，および関数定義の定義情報を格納するファイル。コンパイル対象のソースファイルごとに生成する。

例外処理

手続き文の実行中に発生したエラーに対して処理する機能のこと。

索引

記号

数字

A

B

C

D

E

F

G

H

I

S

T

U

V

W

X

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

に

ね

は

ひ

ふ

へ

ほ

ま

み

め

も

ゆ

よ

ら

り

る

れ

ろ