HITACHI
Inspire the Next

COBOL2002　ユーザーズガイド

解説・手引書

3021-3-600-50

COBOL2002

前書き

■ 対象製品

P-2636-2344 COBOL2002 Net Developer 04-70（適用 OS：Windows 7，Windows 8.1，Windows 10，Windows 11，Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-2436-5344 COBOL2002 Net Server Runtime 04-70（適用 OS：Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-2636-3344 COBOL2002 Net Client Runtime 04-70（適用 OS：Windows 7，Windows 8.1，Windows 10，Windows 11）

P-2436-6344 COBOL2002 Net Server Suite 04-70（適用 OS：Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-2636-4344 COBOL2002 Net Client Suite 04-70（適用 OS：Windows 7，Windows 8.1，Windows 10，Windows 11）

P-2936-2344 COBOL2002 Net Developer(64) 04-70（適用 OS：Windows 7(x64)，Windows 8.1(x64)，Windows 10(x64)，Windows 11，Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-2936-5344 COBOL2002 Net Server Runtime(64) 04-70（適用 OS：Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-2936-6344 COBOL2002 Net Server Suite(64) 04-70（適用 OS：Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-2636-7344 COBOL2002 Developer Professional 04-70（適用 OS：Windows 7，Windows 8.1，Windows 10，Windows 11，Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-2936-7344 COBOL2002 Developer Professional(64) 04-70（適用 OS：Windows 7(x64)，Windows 8.1(x64)，Windows 10(x64)，Windows 11，Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

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

■ 発行

2024 年 1 月　3021-3-600-50

■ 著作権

変更内容

変更内容（3021-3-600-50） COBOL2002 Net Developer 04-70，COBOL2002 Net Server Runtime 04-70，COBOL2002 Net Client Runtime 04-70，COBOL2002 Net Server Suite 04-70，COBOL2002 Net Client Suite 04-70，COBOL2002 Net Developer(64) 04-70，COBOL2002 Net Server Runtime(64) 04-70，COBOL2002 Net Server Suite(64) 04-70，COBOL2002 Developer Professional 04-70，COBOL2002 Developer Professional(64) 04-70

追加・変更内容	変更個所
直前に実行された組み込み関数の 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 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。	21.3.3，21.8.1，28.5.6，30.1.4，30.7.3，33.5.4，33.5.14，付録 I
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。	29.4.3
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。	33.5.3，33.5.4，33.5.12，付録 I
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。	33.5.3，33.5.4，33.5.12，付録 I
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）	33.6.2，33.6.3，33.7.3
初期化漏れチェック機能での，再定義によって同じ記憶域に関連づけられたデータ項目の扱いについて注意事項を追加した。	33.7.3

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-2636-2344 COBOL2002 Net Developer
- P-2436-5344 COBOL2002 Net Server Runtime
- P-2636-3344 COBOL2002 Net Client Runtime
- P-2436-6344 COBOL2002 Net Server Suite
- P-2636-4344 COBOL2002 Net Client Suite
- P-2936-2344 COBOL2002 Net Developer(64)
- P-2936-5344 COBOL2002 Net Server Runtime(64)
- P-2936-6344 COBOL2002 Net Server Suite(64)
- P-2636-7344 COBOL2002 Developer Professional
- P-2936-7344 COBOL2002 Developer Professional(64)

■ 対象読者

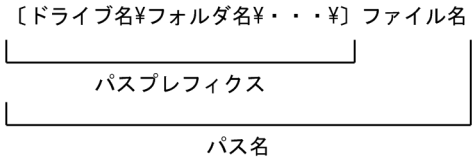
このマニュアルは，COBOL2002 の機能を知りたい方，または COBOL2002 の文法規則を機能から調べたい方を対象としています。また，COBOL の基本的な言語仕様と，Windows の操作方法について理解していることを前提としています。

■ 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

ドライブ名で始まるパス名。

ドライブ名¥フォルダ名¥ … ¥ファイル名

相対パス名

カレントフォルダからの相対のパス名。

・カレントフォルダの 1 階層上位のフォルダを経由する場合

..¥フォルダ名¥フォルダ名¥ … ¥ファイル名

・カレントフォルダ下のフォルダを経由する場合

フォルダ名¥フォルダ名¥ … ¥ファイル名

絶対パスプレフィクス

ドライブ名で始まるパスプレフィクス。

相対パスプレフィクス

カレントフォルダ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

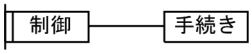
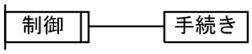
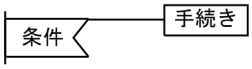
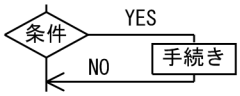
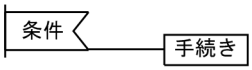
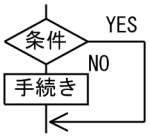
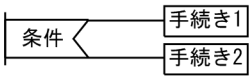
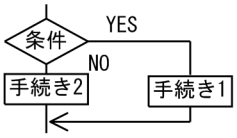
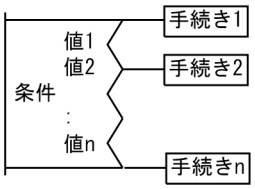
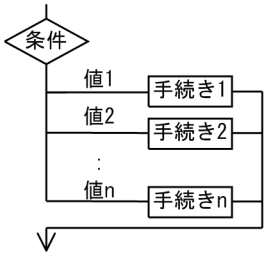
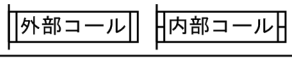
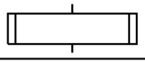
このマニュアルで使用する記号を次に示します。

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。

要素		PADでの表記	流れ図での表記 （JIS X 0121-1986による）
基本要素			
順次要素			
繰り返し要素	前判定 繰り返し (WHILE型)		
	後判定 繰り返し (UNTIL型)		
選択要素	単岐選択 (IF型)		
			
	双岐選択 (IF型)		
	多岐選択 (CASE型)		
定義済処理 （サブルーチン）			
端　　子			
結　合　子			

■ サポート機能一覧

規格

機能	製品種別	
	Windows(x86) COBOL2002	Windows(x64) COBOL2002
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	
		Windows(x86) COBOL2002	Windows(x64) COBOL2002
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	
	Windows(x86) COBOL2002	Windows(x64) COBOL2002
日本語	○	○
ブール演算	○	○
アドレス操作	○	○

機能		製品種別	
		Windows(x86) COBOL2002	Windows(x64) COBOL2002
1 バイト 2 進および COMP-X 項目		○	○
浮動小数点項目		○	○
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		○	○
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	
		Windows(x86) COBOL2002	Windows(x64) COBOL2002
	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	○	○
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		○	○
数字項目のけた拡張機能		×	○
動的長基本項目機能		○	○

機能		製品種別	
		Windows(x86) COBOL2002	Windows(x64) COBOL2002
定数長拡張機能	英数字定数長の拡張	○	○
Java プログラム呼び出し機能		○	○

(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

デバッグ機能

機能		製品種別	
		Windows(x86) COBOL2002	Windows(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	○	○
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

連携機能

機能	製品種別	
	Windows(x86) COBOL2002	Windows(x64) COBOL2002
XML 連携機能	○	○
Cosminexus 連携機能	○	○

(凡例)

○：サポートしている

開発／実行環境

機能	製品種別	
	Windows(x86) COBOL2002	Windows(x64) COBOL2002
開発マネージャ	○	○
実行支援	○	○
ODBC レコード定義生成	○	○
ファイル／レコード定義	○	○
画面定義	○	○
COBOL エディタ	○	○
コマンドプロンプト	○	○

(凡例)

○：サポートしている

使用を推奨しない機能について

次の機能は，旧製品との互換を維持するために提供しています。前提となるソフトウェアは販売を終了しています。COBOL2002 の新しい機能への移行を推奨します。

- Btrieve（Pervasive.SQL）による索引編成ファイル
- MIOS7 COBOL85 との互換機能

■ Windows の用語の表記

このマニュアルでは，次の用語について Windows 7，Windows Server 2008 R2 の表記を使用します。Windows 8.1，Windows 10，Windows 11，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022 をご使用になる場合は，表記を読み替えてください。

Windows 7，Windows Server 2008 R2 の用語	Windows 8.1，Windows 10，Windows 11，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022 の用語
ログオン	サインイン
ログオフ	サインアウト

■ ［COBOL2002］下のメニューと，ウィンドウのタイトルバーの表記

Windows(x64) COBOL2002 の場合，Windows のプログラム一覧にある［COBOL2002］下のメニューと，ウィンドウのタイトルバーには，末尾に「64bit」が付きます。このマニュアルでは，Windows(x86) COBOL2002 のメニューとタイトルバーの表記を使用します。Windows(x64) COBOL2002 をご使用になる場合は，表記を読み替えてください。

Windows(x86) COBOL2002 での表記	Windows(x64) COBOL2002 での表記
COBOL2002	COBOL2002 64bit
COBOL エディタ for COBOL2002	COBOL エディタ for COBOL2002 64bit
COBOL コマンドプロンプト for COBOL2002	COBOL コマンドプロンプト for COBOL2002 64bit
ODBC レコード定義生成 for COBOL2002	ODBC レコード定義生成 for COBOL2002 64bit
カバレージ for COBOL2002	カバレージ for COBOL2002 64bit
テストデバッガ for COBOL2002	テストデバッガ for COBOL2002 64bit
ファイル・レコード定義 for COBOL2002	ファイル・レコード定義 for COBOL2002 64bit
画面定義 for COBOL2002	画面定義 for COBOL2002 64bit
開発マネージャ for COBOL2002	開発マネージャ for COBOL2002 64bit
実行支援 for COBOL2002	実行支援 for COBOL2002 64bit

■ コンパイラオプション，および環境変数について

このマニュアルには，Windows(x64) COBOL2002 では使用できないコンパイラオプション，および環境変数についても記載しています。

使用できないコンパイラオプションについては「39.2.1　使用できないコンパイラオプション」を，実行時環境変数については「39.3.1　プログラムの実行環境の設定」の「(2)　実行時環境変数の一覧」をそれぞれ参照してください。

■ 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 Windows(x86) COBOL2002 用です。プログラム例を Windows(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。

また，このマニュアルの C 言語のプログラム例を Windows(x64) COBOL2002 でご使用になった場合，コンパイラによっては処理結果に相違が出ることがあります。

目次

前書き　2
変更内容　4
はじめに　5

第 1 編　COBOL2002 の概要

1　概要　39
1.1　COBOL2002 の概要　40
1.1.1　COBOL の概要　40
1.1.2　COBOL の特長　40
1.1.3　COBOL2002 の機能　40
1.1.4　COBOL2002 の製品体系　41
1.2　COBOL2002 の構成　44
1.3　COBOL2002 が提供するコンポーネントの種類と関連性　45
1.3.1　開発環境について　45
1.3.2　実行環境について　45
1.3.3　デバッグ環境について　46
1.3.4　マニュアルについて　46

2　COBOL2002 の主な新機能　47
2.1　オブジェクト指向機能　48
2.2　共通例外処理　49
2.3　翻訳指令　50
2.3.1　規格の互換性をチェックする翻訳指令　50
2.3.2　ソース原文の正書法を決定する翻訳指令　50
2.3.3　条件翻訳に関連する翻訳指令　50
2.3.4　コンパイルリストに関連する翻訳指令　51
2.3.5　例外処理に関連する翻訳指令　52
2.4　TYPEDEF 句と SAME AS 句　53
2.4.1　TYPEDEF 句　53
2.4.2　SAME AS 句　55
2.5　利用者定義関数　56
2.5.1　利用者定義関数の参照　56
2.5.2　利用者定義関数の引数と返却項目　57
2.6　再帰呼び出し　58

2.7 局所場所節 59
2.8 自由形式のソース原文や登録集原文 60

第 2 編 COBOL プログラムの書き方

3 翻訳グループを構成する定義の種類 61
3.1 翻訳グループの概要と考え方 62
3.2 定義の種類 63
3.2.1 プログラム定義 63
3.2.2 関数定義 64
3.2.3 クラス定義 65
3.2.4 インタフェース定義 67

4 COBOL プログラムのデータ領域 68
4.1 データ領域の種類 69
4.1.1 連絡節のデータ領域 69
4.1.2 作業場所節のデータ領域 70
4.1.3 局所場所節のデータ領域 71
4.1.4 その他の節のデータ領域 71
4.2 データ属性の種類 73
4.2.1 大域属性（GLOBAL 句） 73
4.2.2 外部属性（EXTERNAL 句） 74

第 3 編 手続き文

5 手続き文 78
5.1 概要 79
5.1.1 基本的な内部操作手続き文 79
5.1.2 条件式 81
5.1.3 中間結果の作成条件 84
5.1.4 CORRESPONDING 指定 85
5.2 算術演算機能 87
5.2.1 算術演算機能の特徴 87
5.2.2 算術文 88
5.2.3 有効けた数 91
5.2.4 演算の中間結果 92
5.3 文字列操作文 95
5.3.1 STRING 文 95
5.3.2 UNSTRING 文 97
5.3.3 INSPECT 文 99

5.4 条件分岐文 102
5.4.1 EVALUATE 文 102
5.4.2 IF 文 103
5.5 表操作 105
5.5.1 SEARCH 文 105
5.6 手続き分岐 108
5.6.1 GO TO 文 108
5.6.2 PERFORM 文 108
5.6.3 CONTINUE 文 116
5.7 データ転記文 118
5.7.1 INITIALIZE 文 118
5.7.2 MOVE 文 121
5.7.3 SET 文 122

第 4 編 入出力機能

6 ファイル入出力機能 127
6.1 ファイル入出力機能の種類と概要 128
6.1.1 使用できるファイル編成 128
6.1.2 使用できるファイル形式 129
6.2 ファイル割り当ての共通規則 130
6.2.1 定数指定 130
6.2.2 環境変数指定 132
6.2.3 データ名指定 135
6.2.4 実行時の動的割り当て（GUI モードの場合だけ） 136
6.3 入出力エラー処理 139
6.3.1 入出力エラー処理の概要 139
6.3.2 入出力状態の値 140
6.3.3 USE 手続き 140
6.3.4 ファイル入出力文でのエラー情報出力機能 141
6.3.5 ファイル属性の整合性チェック 142
6.4 順編成ファイル 144
6.4.1 ファイルの作成と割り当て方法 144
6.4.2 順編成ファイルの行制御 144
6.5 相対編成ファイル 146
6.5.1 ファイルの作成と割り当て方法 146
6.6 ISAM による索引編成ファイル 147
6.6.1 ファイルの作成と割り当て方法 147
6.6.2 ファイル編成とレコード形式 151

6.6.3 リモートファイルアクセス機能（Windows(x86) COBOL2002 で有効） 151
6.7 テキスト編成ファイル 153
6.7.1 ファイルの作成と割り当て方法 153
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 153
6.7.3 入出力手続き文と動作 154
6.7.4 規則 156
6.7.5 レコード末尾の空白文字を出力する機能 156
6.8 CSV 編成ファイル（表計算プログラムファイル） 160
6.8.1 ファイルの作成と割り当て方法 160
6.8.2 CSV 編成ファイルのファイル編成とレコード形式 160
6.8.3 入出力手続き文と動作 161
6.8.4 注意事項 163
6.8.5 セルデータを数値として入出力する機能 163
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 166
6.8.7 入力時の未使用項目の初期化機能 167
6.8.8 データの後の空白文字を出力する機能 168
6.8.9 セルデータをタブ文字区切りで入出力する機能 169
6.9 HiRDB による索引編成ファイル 170
6.9.1 プログラムの作成方法 170
6.9.2 ファイルの作成と割り当て方法 170
6.9.3 ファイル編成とレコード形式 171
6.9.4 HiRDB の定義と COBOL の定義の関連性 171
6.9.5 HiRDB による索引編成ファイル固有の機能と相違点 176
6.9.6 プログラムのコンパイルと実行 184
6.9.7 プログラム作成時の留意点 186
6.10 Btrieve（Pervasive.SQL）による索引編成ファイル（Windows(x86) COBOL2002 で有効） 188
6.10.1 前提条件とプログラムの作成方法 188
6.10.2 ファイルの作成と割り当て方法 188
6.10.3 ファイル編成とレコード形式 189
6.10.4 使用できる機能と制限事項 189
6.11 ラージファイル入出力機能 194
6.11.1 ラージファイル入出力機能の概要 194
6.11.2 ISAM による索引編成ファイルでのラージファイル入出力機能（Windows(x86) COBOL2002 で有効） 195
6.11.3 通常の入出力との互換性 196
6.11.4 ラージファイル入出力機能でのファイルの共用 196
6.11.5 ラージファイル入出力機能の制限事項 197

7 ファイル共用（ファイルシェア） 198
7.1 ファイル共用（ファイルシェア）の概要 199

7.2 ファイル共用の詳細 200
7.2.1 ファイルレベルのファイル共用 200
7.2.2 レコードレベルのファイル共用 201
7.2.3 ファイルの排他・共用の区別 203
7.2.4 各実行単位の施錠形式 204
7.2.5 ファイル共用を省略した場合の処理 206
7.2.6 Btrieve（Pervasive.SQL）による索引編成ファイル（Windows(x86) COBOL2002 で有効） 206
7.2.7 ラージファイル入出力機能でのファイルの共用 206
7.3 ファイル共用の注意事項 207

8 プリンタへのアクセス 208
8.1 プリンタアクセスの種類と概要 209
8.1.1 印刷モード 209
8.1.2 プリンタアクセスで使用できるファイル編成 209
8.2 プリンタアクセスの共通規則 211
8.2.1 ファイル割り当て 211
8.2.2 入出力手続き文と動作 212
8.2.3 行制御出力 212
8.2.4 実行時環境変数で行制御を操作する方法 213
8.2.5 印刷文書名称 213
8.2.6 外字の有効化 215
8.3 入出力エラー処理 217
8.4 GDI モード印刷 218
8.4.1 プリンタへの出力割り当て方法 218
8.4.2 出力形態とレコード形式 220
8.4.3 印刷書式の設定 225
8.4.4 外字の出力方法 225
8.4.5 注意事項 225
8.5 ESC/P モード印刷 226
8.5.1 プリンタへの出力と割り当て方法 226
8.5.2 出力形態とレコード形式 227
8.5.3 外字の出力方法 228
8.5.4 ESC/P モード印刷を利用したプリンタへの出力例 232
8.6 XMAP3 による印刷（Windows(x86) COBOL2002 で有効） 236
8.6.1 前提条件とプログラムの作成方法 236
8.6.2 プリンタへの出力と割り当て方法 236
8.6.3 出力形態とレコード形式 237
8.6.4 入出力手続き文と動作 238
8.6.5 書式オーバレイの出力方法 238

8.6.6 XMAP3 による印刷モードの注意事項 239

9 報告書作成機能 241
9.1 報告書作成機能の概要 242
9.2 ファイル割り当ての共通規則 243
9.3 入出力エラー処理 244
9.3.1 USE 手続き 244
9.4 ファイルの作成と割り当て方法 245
9.5 ファイル編成とレコード形式 246
9.6 報告書ファイルの出力 247
9.7 報告書ファイルの入力 249

10 ACCEPT／DISPLAY／STOP 文による入出力 250
10.1 ACCEPT／DISPLAY／STOP 文による入出力の種類と概要 251
10.2 少量入出力 252
10.2.1 入出力の対象とするファイルの割り当て方法 252
10.2.2 CUI モード，GUI モードとの関係 253
10.2.3 外部からのデータを入力する ACCEPT 文 254
10.2.4 日付や時刻を取得する ACCEPT 文 257
10.2.5 DISPLAY 文によるデータの出力 259
10.2.6 STOP 文 261
10.3 コマンド行へのアクセス 263
10.3.1 コマンド行へのアクセスの種類と概要 263
10.3.2 引数を個別に取得する方法 263
10.3.3 引数を一括して取得する方法 268
10.4 環境変数へのアクセス 269
10.4.1 環境変数の値の読み込み 269
10.4.2 環境変数への値の書き出し 269
10.4.3 環境変数へのアクセスに関する規則 270
10.4.4 使用例 271
10.5 イベントログファイル出力機能 274
10.5.1 イベントログファイル出力機能の概要 274
10.5.2 イベントの出力 274
10.5.3 イベントの出力内容 274
10.5.4 イベントの出力先 276
10.5.5 イベントログファイルへの出力がエラーになったときの動作 277
10.5.6 注意事項 278

11 整列併合機能 280
11.1 使用できるファイル 281

11.2 ファイルの割り当て 282
11.2.1 入出力用ファイル 282
11.2.2 整列作業用ファイル 282
11.2.3 注意事項 282
11.3 使用するメモリサイズ 283
11.3.1 整列処理のメモリサイズ 283
11.3.2 併合処理のメモリサイズ 283
11.4 使用できる特殊レジスタ 285
11.4.1 特殊レジスタの種類 285
11.4.2 SORT-RETURN 特殊レジスタ 285
11.4.3 SORT-CORE-SIZE 特殊レジスタ 285
11.5 注意事項 286
11.5.1 処理時間の短縮 286
11.5.2 その他の注意事項 287

12 画面入出力機能 288
12.1 通信節による画面機能 289
12.1.1 機能の概要 289
12.1.2 画面に対する入出力 289
12.1.3 仮想端末の共用 292
12.1.4 プリンタに対する帳票出力 293
12.2 画面節（SCREEN SECTION）による画面機能 296
12.2.1 画面の種類と構成 296
12.2.2 キーの機能 298
12.2.3 LINE／COLUMN 句を使用した画面の座標指定 299
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 300
12.2.5 注意事項 300
12.3 画面節（WINDOW SECTION）による画面機能 301
12.3.1 画面の種類と構成 301
12.3.2 データの表示形式 303
12.3.3 データの入力方式 304
12.3.4 キーの機能 306
12.3.5 ポップアップ画面入出力機能 307
12.3.6 ユーザポップアップ HELP 機能 308
12.3.7 注意事項 308

13 COBOL 入出力サービスルーチン 310
13.1 COBOL 入出力サービスルーチンの概要 311
13.1.1 概要 311

13.1.2 COBOL 入出力サービスルーチンが対応している機能 311
13.2 COBOL 入出力サービスルーチンの説明 313
13.3 COBOL 入出力サービスルーチンのインタフェース 315
13.3.1 サービスルーチンを呼び出す関数の形式 315
13.3.2 インタフェース領域の形式 315
13.4 リンクの指定 330
13.5 デバッグ情報の取得 331
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 331
13.5.2 インタフェース領域のダンプ出力 333
13.6 COBOL 入出力サービスルーチンでの複数レコード施錠 337
13.7 COBOL 入出力サービスルーチンでのディスク書き込み保証 338
13.7.1 ファイルごとに指定する方法 338
13.7.2 プロセス内のすべてのファイルに対して指定する方法 339
13.7.3 CBLWDISK サービスルーチンを呼び出して保証する方法 339
13.7.4 注意事項 339
13.8 COBOL 入出力サービスルーチンの使用例 340
13.9 注意事項 342

14 ファイルのディスク書き込み保証 343
14.1 ファイルのディスク書き込み保証 344
14.1.1 対象となるファイル編成 344
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 345
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 346
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 346
14.1.5 注意事項 347

15 バイトストリーム入出力サービスルーチン 348
15.1 バイトストリーム入出力サービスルーチンの概要 349
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 349
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 349
15.2 バイトストリーム入出力サービスルーチンの説明 352
15.2.1 CBLSTMCLOSE 352
15.2.2 CBLSTMCREATE 352
15.2.3 CBLSTMOPEN 353
15.2.4 CBLSTMREAD 354
15.2.5 CBLSTMWRITE 355
15.3 使用例 356

第 5 編　COBOL 実行単位と連絡

16　COBOL の実行単位　358

16.1　実行単位の構成　359
16.1.1　実行単位とは　359
16.1.2　実行可能ファイルや DLL とは　359
16.1.3　実行単位と実行モジュールの例　359
16.2　引数の受け取りと外部スイッチ　361
16.2.1　コマンド行に指定する引数の形式　361
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）　362
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）　365
16.2.4　外部スイッチ　366
16.3　COBOL プログラムのモード　369
16.3.1　CUI モード　369
16.3.2　GUI モード　370
16.3.3　モードの決定方法　372
16.4　COBOL 実行単位の終了　375
16.4.1　実行単位の終了方法　375
16.4.2　実行単位の終了コード　377

17　プログラム間の引数と返却項目　379

17.1　引数の受け渡し　380
17.1.1　引数の受け渡しの種類　380
17.1.2　使用例　380
17.1.3　引数の受け渡しの規則　381
17.2　復帰コードと返却項目　384
17.2.1　復帰コードと返却項目の使用方法　385

18　プログラムの呼び出し　388

18.1　プログラム呼び出しの種類と概要　389
18.1.1　定数指定の CALL 文　389
18.1.2　一意名指定の CALL 文　390
18.2　プログラムの取り消し　391
18.2.1　取り消し対象のプログラム　391
18.2.2　取り消しで解放される資源　393
18.2.3　取り消し後の呼び出し　393
18.3　COBOL 主プログラムと副プログラム　395
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　395
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　396
18.4　プログラム属性と呼び出し規約　398

18.4.1 プログラム属性 398
18.4.2 呼び出し規約 404
18.5 静的なリンクと動的なリンク 406
18.5.1 静的なリンク 406
18.5.2 動的なリンク 406
18.6 DLL に含まれるプログラムの呼び出し 408
18.6.1 DLL の概要 408
18.6.2 DLL に含まれるプログラムの呼び出し方法 408
18.6.3 動的なリンクのプレロード機能 411
18.6.4 動的なリンクのプログラム検索トレース機能 412
18.7 実行可能ファイルの呼び出し 417
18.7.1 実行可能ファイル呼び出しの概要 417
18.7.2 実行方式 417
18.7.3 実行可能ファイルの指定 418
18.7.4 引数の受け渡し 418
18.7.5 実行可能ファイルの終了コードの取得 419

19 他言語とのプログラム間連絡 420
19.1 C 言語との連携 421
19.1.1 概要 421
19.1.2 C プログラムから COBOL プログラムを呼び出す方法 422
19.1.3 COBOL プログラムから C プログラムを呼び出す方法 428
19.1.4 注意事項 434
19.1.5 外部属性を持つデータ項目の共用 435
19.1.6 COBOL プログラムと C プログラムのリンク方法 436
19.2 Visual Basic との連携（Windows(x86) COBOL2002 で有効） 437
19.2.1 Visual Basic から COBOL を呼び出すときの規則 437
19.2.2 COBOL プログラムのデータ型と Visual Basic での宣言 438
19.2.3 引数の受け渡し 439
19.3 Java との連携 441
19.3.1 概要 441
19.3.2 Java プログラムから COBOL プログラムを呼び出す方法 444
19.3.3 Java 言語と COBOL 言語のデータ型の対応 451
19.3.4 制限事項 454

第 6 編 オブジェクト指向機能

20 オブジェクト指向機能 455
20.1 オブジェクト指向の紹介 456
20.1.1 ソフトウェア開発の現状 456

20.1.2 オブジェクト指向 456
20.2 COBOL2002 でのオブジェクト指向機能 465
20.2.1 オブジェクト指向機能による定義 465
20.2.2 インスタンスオブジェクトの生成と消滅 466
20.2.3 メソッドの呼び起こし（メッセージパッシング） 468
20.2.4 オブジェクトプロパティ 470
20.2.5 オブジェクト指向による継承 472
20.2.6 オブジェクト指向でのインタフェース 474
20.2.7 オブジェクト指向による適合 477
20.2.8 オブジェクト指向によるポリモルフィズム 482
20.2.9 オブジェクト指向機能でのマルチスレッド対応 483

第 7 編 例外処理

21 共通例外処理 484
21.1 共通例外処理の概要 485
21.1.1 共通例外の仕組みと使用する用語 485
21.1.2 共通例外処理の機能 486
21.1.3 共通例外処理の使用例 486
21.1.4 共通例外処理に対応している機能 489
21.2 例外 492
21.2.1 例外名 492
21.2.2 例外オブジェクト 498
21.2.3 例外の致命度 499
21.2.4 最新例外状態 499
21.3 TURN 指令 501
21.3.1 TURN 指令によるチェック 501
21.3.2 TURN 指令の有効範囲 502
21.3.3 例外チェックが無効な場合の動作 503
21.4 共通例外の宣言手続き 514
21.4.1 実行される宣言手続き 514
21.4.2 宣言手続きからの復帰 517
21.5 例外の伝播 520
21.5.1 PROPAGATE 指令による例外の自動伝播 521
21.5.2 EXIT 文，GOBACK 文の RAISING 指定による例外の伝播 522
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 527
21.6 明示的な例外の引き起こし 530
21.7 例外情報の参照 532
21.7.1 組み込み関数を使用した例外情報の参照 532

21.7.2 EXCEPTION-OBJECT 537
21.7.3 最新例外状態のクリア 538
21.8 例外の検出条件 540
21.8.1 例外が検出される文の詳細 540
21.8.2 例外検出での注意事項 543
21.8.3 例外処理の動作 544
21.9 共通例外処理の注意事項 547
21.9.1 共通例外処理を使用した場合の性能について 547
21.9.2 従来形式の例外処理と共通例外処理の関係 547

第 8 編 DB／DC 連携

22 データコミュニケーション機能 550
22.1 データコミュニケーション機能の概要 551
22.2 DC シミュレーション 552
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 553

23 データベース操作機能 554
23.1 データベースアクセス機能 555
23.1.1 埋め込み SQL 文を使った COBOL プログラムの作成 555
23.1.2 プログラムの例 559
23.2 ODBC インタフェース機能の概要 565
23.2.1 ODBC インタフェース機能が動作する環境 565
23.2.2 データソースの管理について 566
23.2.3 コンパイル 568
23.2.4 SQL 文のエラー処理 568
23.2.5 埋め込み変数 571
23.2.6 トランザクション 573
23.2.7 コネクション 573
23.2.8 タイムアウト秒数の設定 574
23.2.9 カーソルオプションの設定 574
23.2.10 データベース固有の注意事項 577
23.2.11 動的 SQL の ODBC API 関数発行の変更 577

24 XDM によるデータベース操作シミュレーション機能 579
24.1 データベース操作シミュレーションの概要 580
24.2 構造型データベース（XDM/SD）操作シミュレーション 581
24.2.1 コンパイル方法 581
24.2.2 実行方法 582
24.2.3 内部的に展開される CALL 文の引数 582

24.2.4 XDM/SD プログラムのテスト方法の制限事項 589
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 591
24.3.1 コンパイル方法 591
24.3.2 テスト方法 591

第 9 編 多様な COBOL プログラムの作成

25 OLE2 オートメーション機能 593
25.1 OLE2 オートメーション機能の概要 594
25.2 OLE2 オートメーションクライアント機能 595
25.2.1 OLE オブジェクトの生成と取得 595
25.2.2 OLE メソッドと OLE プロパティの操作 596
25.2.3 OLE メソッドが返す OLE オブジェクトを利用した参照 597
25.2.4 OLE アプリケーションの終了と OLE オブジェクトの解放 598
25.2.5 VARIANT 値と COBOL データのやり取り 599
25.2.6 注意事項 603

26 マルチスレッド環境での実行 604
26.1 マルチスレッド対応 COBOL プログラムの概要 605
26.1.1 スレッドの制御 605
26.1.2 実行単位 605
26.1.3 データの共用 605
26.1.4 プログラムのコーディング 605
26.2 マルチスレッド対応 COBOL プログラムの生成 606
26.2.1 マルチスレッド対応 COBOL プログラムのコンパイル 606
26.3 マルチスレッド対応 COBOL プログラムが対応している機能 607
26.4 マルチスレッド対応 COBOL プログラムの開始と終了 610
26.4.1 COBOL 以外のプログラムからの呼び出しによる方法 610
26.4.2 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 611
26.4.3 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 612
26.4.4 戻り文に対する動作 612
26.5 実行時エラーが発生したときの動作 613
26.6 マルチスレッド対応 COBOL プログラムを GUI モードで使用する方法 614
26.6.1 COBOL プログラムが主プログラムの場合 614
26.6.2 COBOL プログラムが主プログラムでない場合 614
26.6.3 注意事項 614
26.7 環境変数の取り扱い 615
26.7.1 スレッドごとに固有の出力ファイル名称を付ける機能 615
26.7.2 スレッドごとに環境変数を設定する機能 616
26.8 マルチスレッド対応 COBOL プログラムのデバッグ 618

26.8.1 マルチスレッド対応 COBOL プログラムのデバッグ 618
26.9 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 619
26.9.1 EXTERNAL 句を用いたデータの共用 619
26.9.2 呼び出してはいけないサービスルーチン 619
26.9.3 スレッドを起動する関数 620

27 MSMQ アクセス機能 621
27.1 MSMQ の概要 622
27.1.1 キュー 622
27.1.2 メッセージ 623
27.2 MSMQ アクセスサービスルーチン 626
27.2.1 キューを作成する（CBLMQCREATE） 626
27.2.2 キューを削除する（CBLMQDELETE） 627
27.2.3 キューをオープンする（CBLMQOPEN） 628
27.2.4 キューをクローズする（CBLMQCLOSE） 629
27.2.5 メッセージを送信する（CBLMQSENDMSG） 630
27.2.6 メッセージを受信する（CBLMQRECEIVEMSG） 632
27.2.7 キューのパス名を検索する（CBLMQLOCATE） 634
27.3 MSMQ アクセスサービスルーチンのインタフェース 637
27.4 MSMQ アクセスサービスルーチンの実行順序 643
27.5 MSMQ アクセスサービスルーチンの使用例 644
27.5.1 キューを作成するコーディングの例 644
27.5.2 キューを削除するコーディングの例 644
27.5.3 メッセージを送信するコーディングの例 645
27.5.4 メッセージを受信するコーディングの例 646
27.5.5 キューのパス名を検索するコーディングの例 648

28 Unicode 機能 650
28.1 Unicode 機能の概要 651
28.1.1 コンパイルでの Unicode 機能 651
28.1.2 実行での Unicode 機能 652
28.1.3 デバッグでの Unicode 機能 652
28.2 Unicode 機能のサポート範囲 653
28.3 Unicode 機能の前提条件 654
28.3.1 コード変換ライブラリ 654
28.4 Unicode 機能の詳細 655
28.4.1 コンパイル 655
28.4.2 実行 656
28.5 Unicode に対応する機能 659

28.5.1 基本機能 665
28.5.2 入出力機能 672
28.5.3 CBLNCNV サービスルーチン 676
28.5.4 MSMQ アクセス機能 676
28.5.5 OLE2 オートメーション機能 677
28.5.6 組み込み関数 677
28.6 シフト JIS で入出力する情報 681
28.7 Unicode 機能での制限事項 683
28.7.1 コンパイル時の制限事項 683
28.7.2 実行時の制限事項 684

29 数字項目のけた拡張機能（Windows(x64) COBOL2002 で有効） 686
29.1 数字項目のけた拡張機能の概要 687
29.1.1 概要 687
29.1.2 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 688
29.2 数字項目のけた拡張機能の詳細 689
29.2.1 数字項目のけた拡張機能で対象となるデータ項目 689
29.2.2 数字項目のけた拡張機能で対象となる定数 689
29.2.3 数字項目のけた拡張機能での有効けた数 690
29.3 数字項目のけた拡張機能での演算の中間結果 691
29.3.1 演算の中間結果 691
29.3.2 10 進浮動小数点形式について 691
29.4 数字項目のけた拡張機能に対応する機能一覧 693
29.4.1 数字項目のけた拡張機能で対象となる機能 693
29.4.2 数字項目のけた拡張機能で対象となるソース単位 695
29.4.3 数字項目のけた拡張機能で対象となる節や文 696
29.5 数字項目のけた拡張機能の注意事項 700
29.5.1 数字項目のけた拡張機能を使用する場合の演算結果 700
29.5.2 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 701

第 10 編 サービスルーチン

30 サービスルーチン 702
30.1 サービスルーチンの概要 703
30.1.1 プログラム実行制御 703
30.1.2 ダイアログボックス／ウィンドウ 703
30.1.3 デバッグ機能 704
30.1.4 変換・転記・演算 704
30.1.5 画面節（SCREEN SECTION および WINDOW SECTION） 704
30.1.6 データベース操作機能 705

30.1.7 COBOL の入出力機能 705
30.1.8 XMAP3 を使用した画面・帳票関連 705
30.1.9 MSMQ アクセス機能 706
30.1.10 その他の機能 706
30.2 戻り値の使い方 707
30.3 サービスルーチン使用時の注意事項 708
30.4 プログラム実行制御 709
30.4.1 CBLGINT 709
30.4.2 CBLEND 710
30.4.3 CBLABN 712
30.4.4 CBLARGC 713
30.4.5 CBLARGV 713
30.4.6 CBLEXEC 715
30.4.7 CBLHANDLE 717
30.5 ダイアログボックス／ウィンドウ 719
30.5.1 CBLMESSAGE 719
30.5.2 CBLINPUTDLG 720
30.5.3 JCPOPUP 722
30.6 デバッグ機能 727
30.6.1 CBLDBGINF 727
30.6.2 CBLDATADUMP 728
30.7 変換・転記・演算 731
30.7.1 CBLNCNV 731
30.7.2 CBLUBIT 734
30.7.3 CBLCNVERRORINFO 735
30.8 画面節（SCREEN SECTION および WINDOW SECTION） 738
30.8.1 CBLSGET 738
30.8.2 CBLSETTITLE 739
30.9 データベース操作機能 741
30.9.1 CBLSQLERROR 741
30.9.2 CBLSQLSETOPT 744
30.10 XMAP3 を使用した画面・帳票関連 746
30.10.1 CBLXMAPERROR 746
30.11 その他の機能 749
30.11.1 CBLPUT 749
30.11.2 CBLGET 749
30.11.3 CBLADTRM 750
30.11.4 CBLDLTRM 754
30.11.5 CBLCNSL 755

30.11.6 CBLBELL 756
30.11.7 CBLCUR 757

第 11 編 プログラム作成上の留意点

31 プログラミング上の留意点 759
31.1 処理速度の速いプログラムの作成 760
31.1.1 チェック項目一覧 760
31.1.2 チェック項目の説明 760
31.2 移植性の良いプログラムの作成 766
31.2.1 チェック項目一覧 766
31.2.2 チェック項目の説明 766
31.3 COBOL プログラムが使用するスタック領域 770
31.3.1 スタック領域に配置されるデータ 770
31.3.2 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 771
31.3.3 スタック領域のサイズ変更方法 772

32 最適化機能 773
32.1 最適化のレベル 774
32.1.1 最適化オプションの種類 774
32.2 最適化の内容 776
32.2.1 そと PERFORM 文のインライン展開 776
32.2.2 10 進項目の 2 進項目化 782
32.2.3 不変式のループ外移動 786
32.2.4 コピー伝播 786
32.2.5 共通式の削除 787
32.2.6 定数の畳み込み 788
32.2.7 演算強さの軽減 788

第 12 編 コンパイルと実行

33 COBOL ソースの作成とコンパイル 789
33.1 コンパイル時の主な入出力ファイル 790
33.2 COBOL ソースの作成方法 792
33.2.1 ソースファイル名と拡張子 792
33.2.2 原始プログラムの作成規則 792
33.2.3 正書法 793
33.3 さまざまな形態の COBOL 原始プログラムのコンパイル 797
33.3.1 原始文操作機能 797
33.3.2 スタックコンパイル機能（連続コンパイル機能）の利用 799

33.3.3 条件翻訳の利用 802
33.3.4 条件翻訳結果のコンパイルリスト 804
33.4 コンパイラの起動方法 807
33.4.1 ccbl2002 コマンド 807
33.4.2 ccbl コマンド 809
33.4.3 cblbuild2k コマンド 810
33.5 コンパイラオプション 814
33.5.1 構文規則 814
33.5.2 一般規則 815
33.5.3 コンパイラオプションの優先順位 815
33.5.4 コンパイラオプションの一覧 821
33.5.5 最終生成物の種類（プロジェクトの種類）の設定 831
33.5.6 他製品との連携の設定 834
33.5.7 実行の設定 838
33.5.8 プログラムの最適化の設定 841
33.5.9 デバッグの設定 842
33.5.10 リンクの設定 853
33.5.11 規格の設定 860
33.5.12 他システムとの移行の設定 865
33.5.13 リスト出力の設定 892
33.5.14 その他の設定 894
33.6 コンパイラ環境変数 912
33.6.1 コンパイラ環境変数の設定方法 912
33.6.2 コンパイラ環境変数の一覧 912
33.6.3 コンパイラ環境変数の詳細 914
33.7 コンパイラ付属機能 927
33.7.1 TD コマンド生成機能 927
33.7.2 ヘルプ機能 927
33.7.3 初期化漏れチェック機能 927

34 定義別のコンパイル方法とリポジトリファイル 940
34.1 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 941
34.1.1 概要 941
34.1.2 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 942
34.1.3 リポジトリファイルに格納される情報と適合チェック 944
34.2 リポジトリファイル 945
34.2.1 リポジトリファイルの生成とコンパイル時の利用 945
34.2.2 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 946
34.2.3 リポジトリファイルの生成方法 948

34.2.4 リポジトリファイルの参照方法 949
34.3 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 952
34.3.1 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 952
34.3.2 リポジトリファイルの単独生成 953
34.3.3 プログラム定義だけのコンパイル 955
34.4 リポジトリファイルの管理 956
34.4.1 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 956
34.4.2 リポジトリ管理ツール 960
34.5 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 970

35 実行可能ファイルと DLL の作成 972
35.1 実行可能ファイルの作成方法 973
35.1.1 コンパイルとリンクを同時に実行する方法 973
35.1.2 コンパイルとリンクを別々に実行する方法 976
35.1.3 コンパイルとリンクを実行する場合の注意事項 980
35.2 DLL の作成方法 983
35.2.1 DLL の作成 983
35.3 DLL を呼び出す実行可能ファイルの作成方法 985
35.3.1 インポートライブラリの指定 985
35.3.2 -StdCall オプションと stdcall 呼び出し指示ファイル（Windows(x86) COBOL2002 で有効） 986
35.4 リンカパスの切り替え機能 990
35.4.1 機能概要 990
35.4.2 リンカパスの切り替え機能使用時の注意事項 991

36 プログラムの実行 992
36.1 実行可能ファイルの起動方法 993
36.2 プログラムの実行環境の設定 994
36.2.1 実行時環境変数の設定方法 994
36.2.2 実行時環境変数の一覧 995
36.2.3 一般 1000
36.2.4 少量データ 1003
36.2.5 ファイル 1004
36.2.6 画面 1011
36.2.7 画面（XMAP） 1021
36.2.8 整列併合 1022
36.2.9 拡張機能 1022
36.2.10 デバッグ 1025
36.2.11 イベントログ 1026
36.2.12 オブジェクト指向 1027

第 13 編　デバッグ

37　アプリケーションデバッグ機能　1029
37.1　デバッグ機能の種類と概要　1030
37.2　異常終了時要約情報リスト　1031
37.2.1　異常終了時要約情報リストの内容　1031
37.2.2　トレースバック表示　1033
37.2.3　環境変数情報表示　1034
37.2.4　異常終了時要約情報リストの出力先　1034
37.2.5　プログラム混在時のリストの内容　1035
37.3　データ領域ダンプリスト　1036
37.3.1　データ領域ダンプリストの内容　1036
37.3.2　データ領域ダンプリストの出力先　1039
37.4　プログラム間整合性チェック　1041
37.4.1　整合性チェックの内容　1041
37.4.2　整合性チェックの警告エラー出力　1042
37.5　添字，指標の繰り返し回数，制御変数チェック　1044
37.5.1　デバッグオプションとの関連性　1044
37.5.2　注意事項　1044
37.6　データ例外検出機能　1045
37.7　COBOL が検出するハードウェア例外およびソフトウェア例外　1046
37.7.1　ハードウェア例外およびソフトウェア例外とは　1046
37.7.2　アプリケーションエラー　1046
37.7.3　COBOL 実行時ライブラリが検出する例外　1047
37.7.4　注意事項　1049
37.8　DISPLAY 文による一意名の 16 進ダンプ表示　1050
37.9　テストデバッグ機能　1051
37.9.1　デバッグ機能　1051
37.9.2　テスト機能　1051
37.10　カバレージ機能　1052
37.10.1　カバレージ情報の表示　1052
37.10.2　カウント情報の表示　1052

第 14 編　リンカとリソースコマンド

38　リンカ，ライブラリ管理ツール，リソースコンパイラ　1053
38.1　リンカ　1054
38.1.1　機能の概要　1054
38.1.2　コマンドラインの形式　1054
38.1.3　オプション　1055

38.1.4 リソースファイル 1062
38.1.5 ダイナミックリンク機能 1062
38.1.6 ライブラリの検索 1063
38.1.7 戻り値 1064
38.2 ライブラリ管理ツール 1065
38.2.1 機能の概要 1065
38.2.2 コマンドラインの形式 1065
38.2.3 オプション 1066
38.2.4 各種機能の使い方 1068
38.2.5 戻り値 1070
38.3 リソースコンパイラ 1071
38.3.1 機能の概要 1071
38.3.2 コマンドラインの形式 1071
38.3.3 オプション 1072
38.3.4 戻り値 1073
38.3.5 リソース定義ファイルと COBOL で使用するリソース定義文 1074
38.4 モジュール定義ファイル 1075
38.4.1 モジュール定義ファイルの記述規則 1075
38.4.2 モジュール定義文 1075
38.4.3 モジュール定義文とリンカオプションとの関係 1078

第 15 編 64bit アプリケーションの作成

39 64bit アプリケーションの作成 1079
39.1 Windows(x64) COBOL2002 について 1080
39.1.1 使用できない機能 1080
39.1.2 Windows(x64) COBOL2002 固有の言語仕様 1081
39.1.3 Windows(x64) COBOL2002 各機能の固有仕様 1083
39.2 COBOL ソースの作成とコンパイル 1085
39.2.1 使用できないコンパイラオプション 1085
39.3 プログラムの実行 1087
39.3.1 プログラムの実行環境の設定 1087
39.3.2 プログラムの実行時の注意事項 1087
39.4 実行可能ファイルと DLL の作成 1089
39.4.1 実行可能ファイルと DLL の作成の注意事項 1089

付録 1090
付録 A 前バージョンからの変更点 1091
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 1092
付録 B.1 COBOL2002 V4 への移行性と互換性 1092

付録 B.2　OOCOBOL からの移行性　1098
付録 B.3　COBOL85 版 Cosminexus 連携の移行性と互換性　1098
付録 B.4　COBOL85 版 XML 連携の移行性と互換性　1099
付録 B.5　機能ごとの移行性と互換性に関する注意事項　1100
付録 C　Windows OS 固有の注意事項　1104
付録 C.1　管理者権限についての注意事項　1104
付録 C.2　ファイル格納先についての注意事項　1109
付録 C.3　文字コードについての注意事項　1109
付録 C.4　実行可能ファイルの実行権限についての注意事項　1110
付録 C.5　実行時の注意事項　1110
付録 D　日立 COBOL85 からの古い仕様　1112
付録 D.1　ACCEPT／DISPLAY 文を使用した CSV ファイルへのアクセス　1112
付録 D.2　サービスルーチンを使った OLE2 オートメーションクライアント機能　1113
付録 E　コンパイルリスト　1133
付録 E.1　リストの出力　1133
付録 E.2　リストの見方　1138
付録 F　COBOL で使用するファイル　1163
付録 F.1　COBOL2002 で使用するファイル　1163
付録 F.2　COBOL プログラムの実行時に必要なファイル　1168
付録 G　コンパイラの制限値　1170
付録 H　入出力状態の値　1178
付録 I　COBOL85 と COBOL2002 のコンパイラオプションの対応　1184
付録 J　環境変数の設定　1192
付録 J.1　コンポーネントおよびコマンドでの環境設定　1192
付録 J.2　環境設定のカスタマイズ　1194
付録 J.3　環境変数設定時およびカスタマイズ時の注意事項　1196
付録 K　COBOL コマンドプロンプト　1197
付録 K.1　COBOL コマンドプロンプトの起動と終了　1197
付録 K.2　COBOL コマンドプロンプトの環境変数の状態　1197
付録 L　トラブルシュートに有効な資料　1199
付録 L.1　資料採取の準備　1200
付録 L.2　障害発生時の対処　1204
付録 L.3　注意事項　1204
付録 L.4　障害事例　1204
付録 M　各バージョンの変更内容　1207
付録 N　このマニュアルの参考情報　1210
付録 N.1　関連マニュアル　1210
付録 N.2　このマニュアルでの表記　1211
付録 N.3　KB（キロバイト）などの単位表記について　1214

付録 O　　用語解説　1215

索引　1228

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

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※1
COBOL2002 Net Developer	○	○	−
COBOL2002 Developer Professional※2	○	○	−
COBOL2002 Net Server Runtime	−	○	○
COBOL2002 Net Client Runtime	−	○	○
COBOL2002 Net Server Suite	○	○	○
COBOL2002 Net Client Suite	○	○	○

（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※1

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。

注※2

COBOL2002 Net Developer と COBOL2002 Professional Tool Kit を同梱したプログラムプロダクトです。

(1) COBOL2002 Net Developer

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。コンパイラをはじめとするさまざまな開発ツールや，アプリケーションプログラムの実行テストに必要な実行時ライブラリや実行環境支援などのツールも使用できます。

ただし，COBOL2002 Net Developer はシステム開発だけを目的としたプログラムプロダクトとなっています。したがって，システム開発とシステム運用の両方を実行するためには，COBOL2002 Net Server Suite や COBOL2002 Net Client Suite を利用してください。

(2) COBOL2002 Developer Professional

プログラム開発環境である COBOL2002 Net Developer と，プログラム開発を支援する機能である COBOL2002 Professional Tool Kit とを同梱したプログラムプロダクトです。

(3) COBOL2002 Net Server Runtime

開発した COBOL プログラムをサーバ OS 上で運用するため，実行環境を使用するための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や実行環境支援などのツールが使用できます。

COBOL2002 Net Developer で開発したアプリケーションプログラムを，サーバ側のシステム上で運用する場合，このプログラムプロダクトが必要です。

(4) COBOL2002 Net Client Runtime（32bit 版クライアント OS 向けに提供）

開発した COBOL プログラムをクライアント OS 上で運用するため，実行環境を使用するための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や実行環境支援などのツールが使用できます。

COBOL2002 Net Developer で開発したアプリケーションプログラムを，クライアント側のシステム上で運用する場合，このプログラムプロダクトが必要です。

(5) COBOL2002 Net Server Suite

COBOL2002 Net Developer と COBOL2002 Net Server Runtime を統合したプログラムプロダクトです。サーバ OS 上での，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。

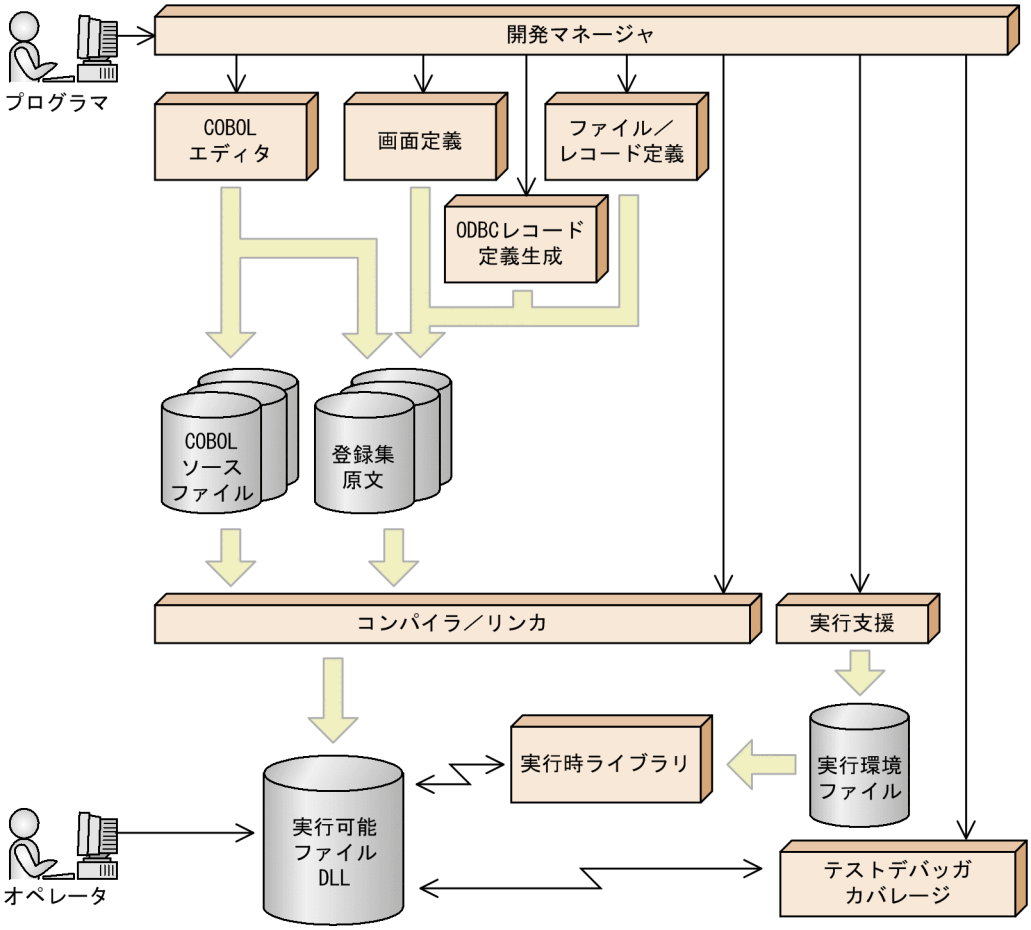
(6) COBOL2002 Net Client Suite（32bit 版クライアント OS 向けに提供）

COBOL2002 Net Developer と COBOL2002 Net Client Runtime を統合したプログラムプロダクトです。クライアント OS 上での，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
開発マネージャ	COBOL プログラムの開発支援
COBOL エディタ	COBOL プログラムの編集
ODBC レコード定義生成	データベースのデータソースから COBOL のレコード定義を生成
画面定義	COBOL プログラムで使用する画面の作成
ファイル／レコード定義	COBOL プログラムで使用するファイル仕様およびレコード仕様の設定

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
実行支援	COBOL プログラムの実行環境の設定
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

表 1-3　障害調査支援のコンポーネント

コンポーネント	機能概要
COBOL2002 情報抽出ツール	バイナリファイルから COBOL2002 の情報を抽出・表示

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 1-4　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 で使用できるマニュアルの概要を次に示します。

表 1-5　マニュアルの種類

マニュアル名	概要
COBOL2002　ユーザーズガイド	COBOL2002 の機能と使用方法について説明しています。
COBOL2002　操作ガイド	開発マネージャ，テストデバッガなど，COBOL2002 で使用できる開発ツール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，「33.2.3　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，「33.3.3　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，「33.3.3　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，「33.3.3　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 E　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 E.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 E.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。

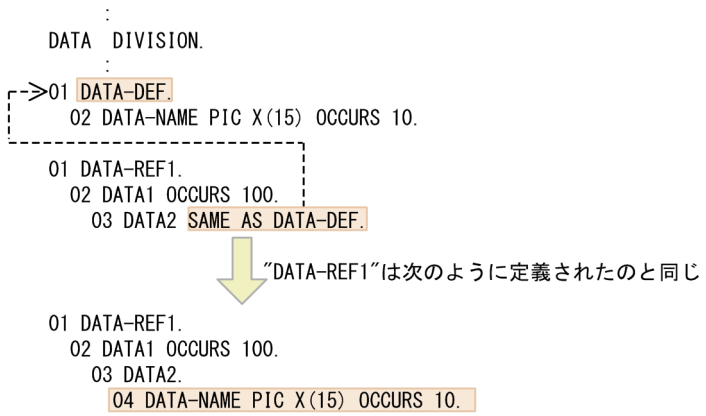
2.4.2　SAME AS 句

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。

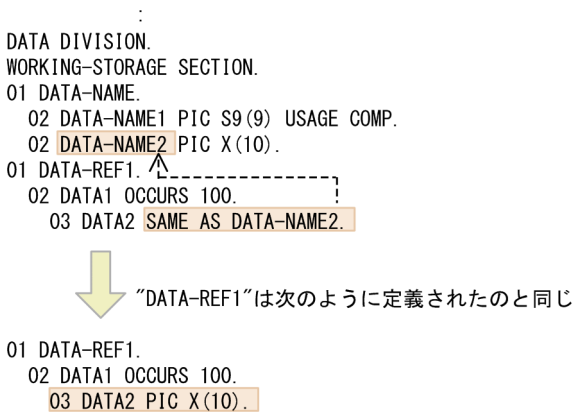
(例 1)

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

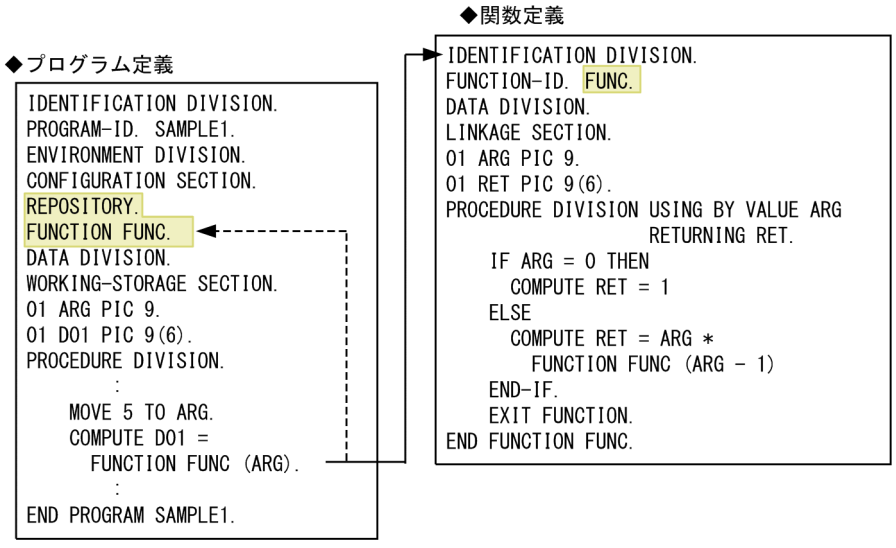
利用者定義関数は，送り出し側の作用対象として，関数一意名によって参照（呼び出し）されます。関数一意名で参照される利用者定義関数は，リポジトリ段落に指定された関数定義となります。

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　プログラム属性と呼び出し規約」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，リンカの-STACK オプションでスタック領域を拡張する必要があります。リンカオプションの詳細については，「38.1.3 オプション」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。

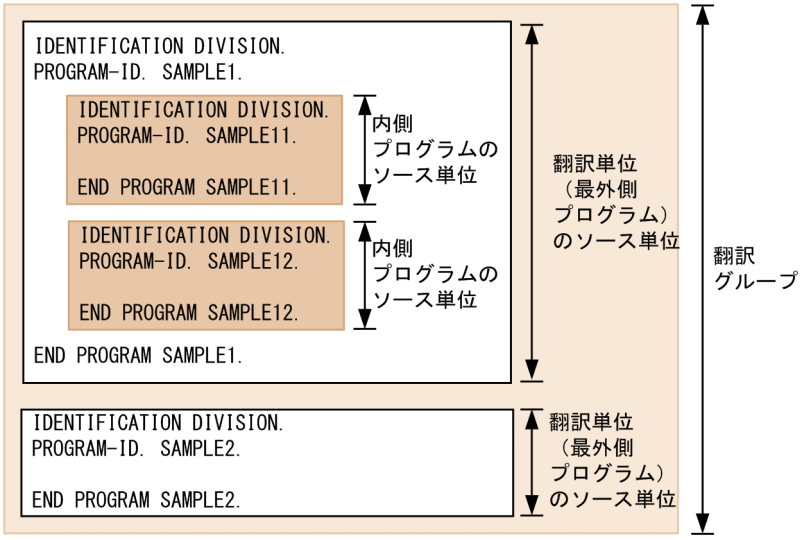
- 最外側のプログラム定義
- 関数定義
- クラス定義
- インタフェース定義

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（.obj）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 DLL 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，「33.　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

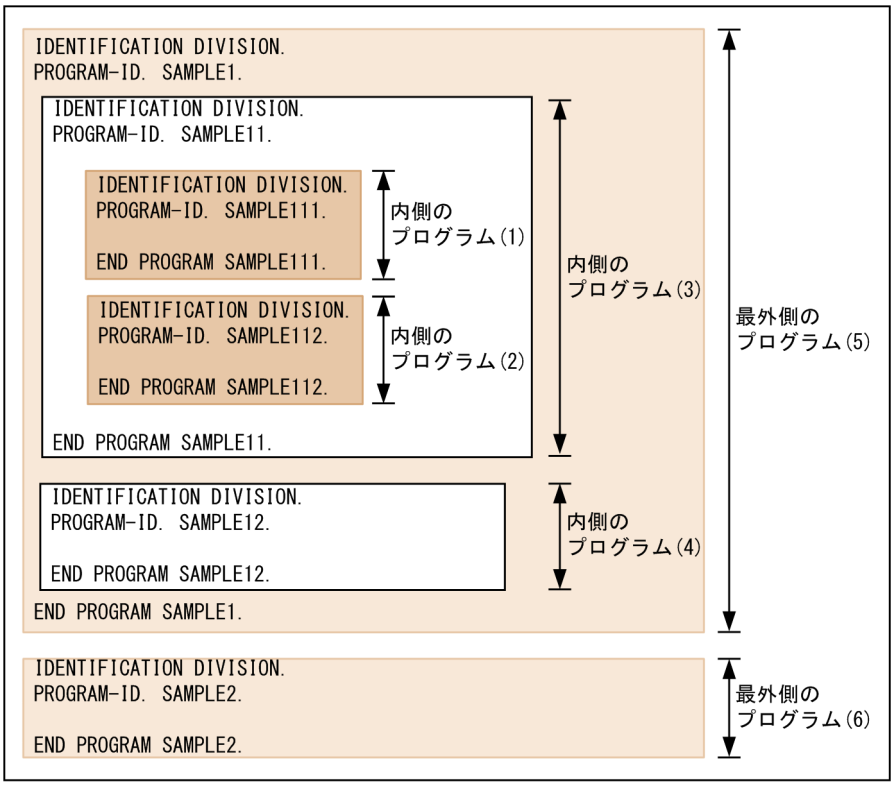
  ほかのプログラムに含まれるプログラムです。

  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。

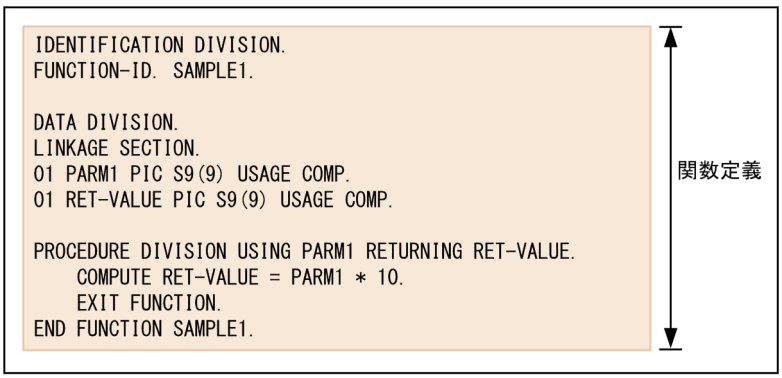
関数名段落の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「7.4　関数名段落(FUNCTION-ID)」を参照してください。

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。

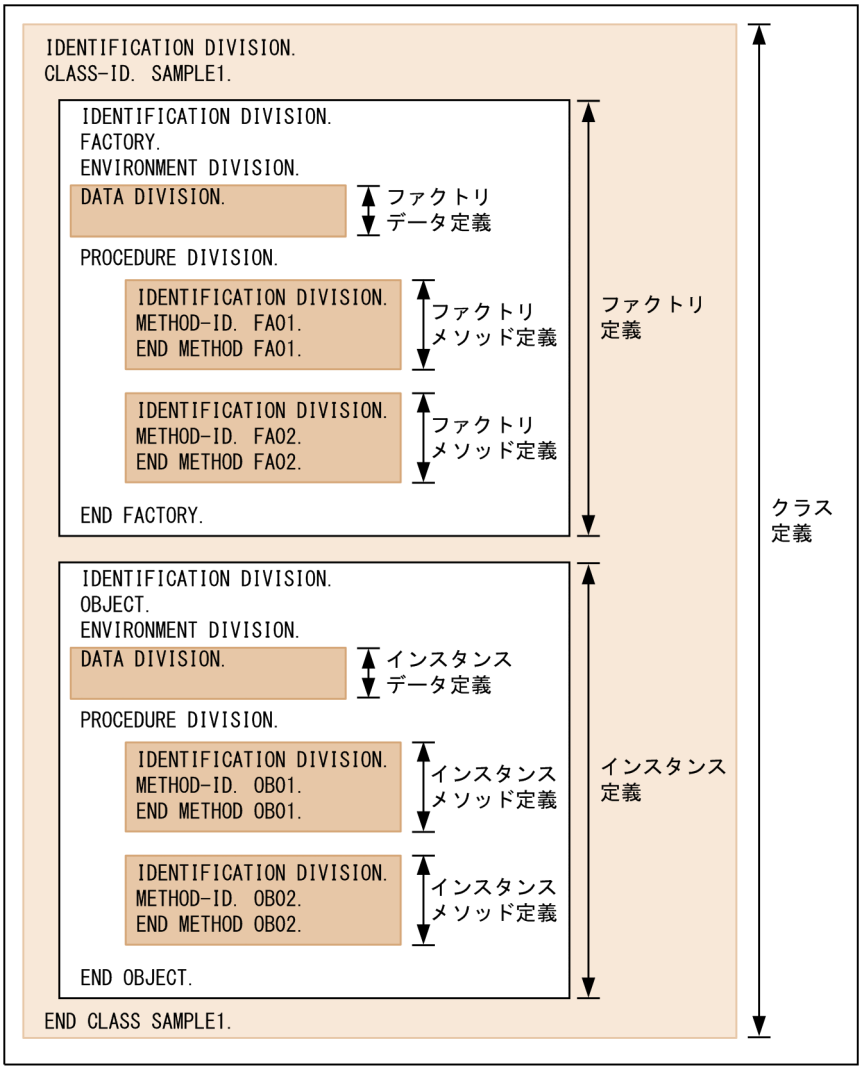
- メソッド定義

  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。

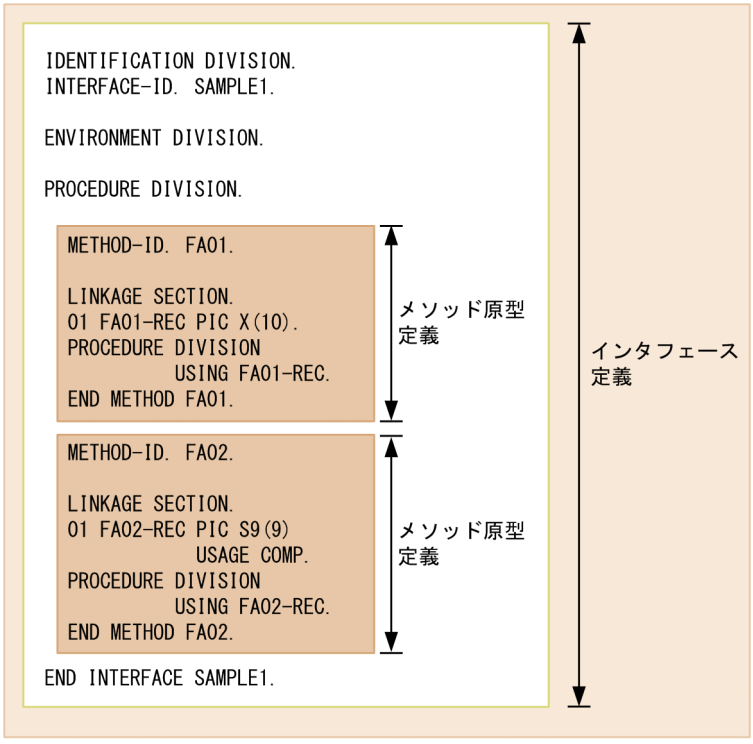
インタフェース名段落の言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「7.5　インタフェース名段落（INTERFACE-ID)」を参照してください。インタフェースについては，マニュアル「COBOL2002　言語　標準仕様編」「付録Ｉ　用語の定義（Terms and Definitions)」のインタフェースの説明を参照してください。

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。

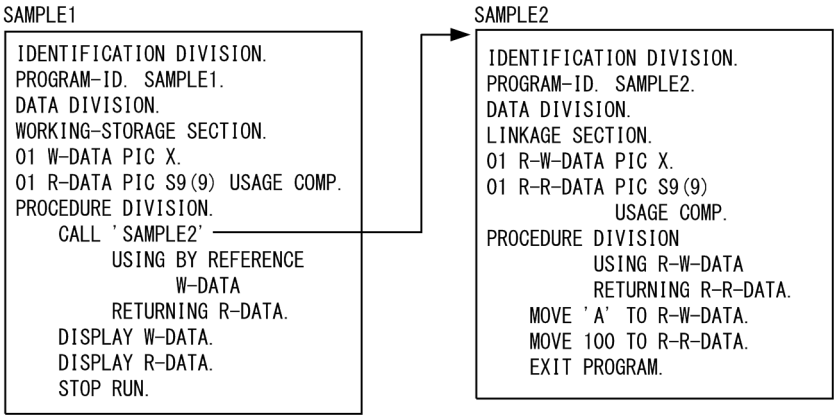
連絡節は，ソース要素が呼び出され，呼び出し先の要素の手続き部見出しに USING／RETURNING が指定されている場合だけ有効となります。

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。

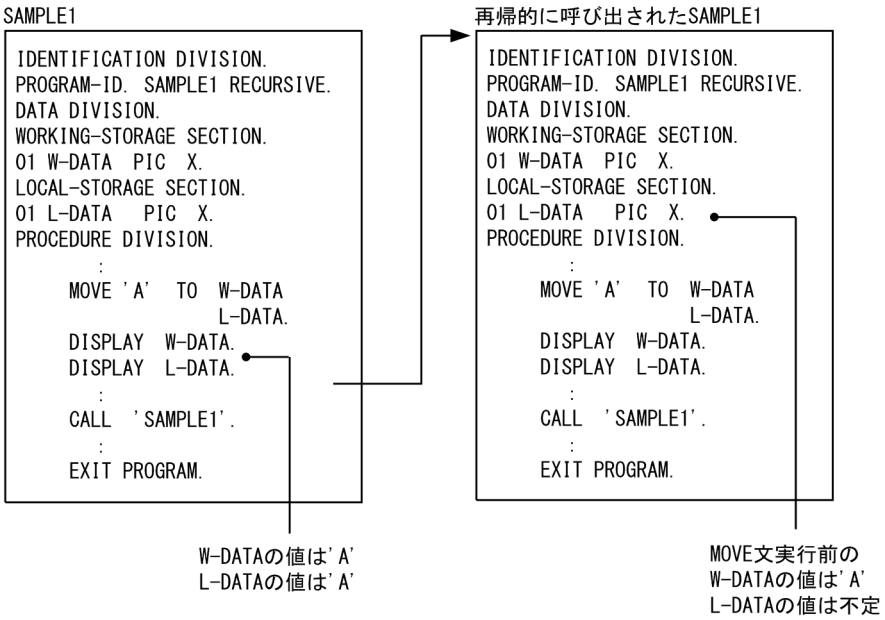
局所場所節は，プログラム定義，関数定義，メソッド定義などの再帰的に呼び出せるプログラムが呼び出されたときに，呼び出される単位ごとに確保されるデータを定義します。

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。

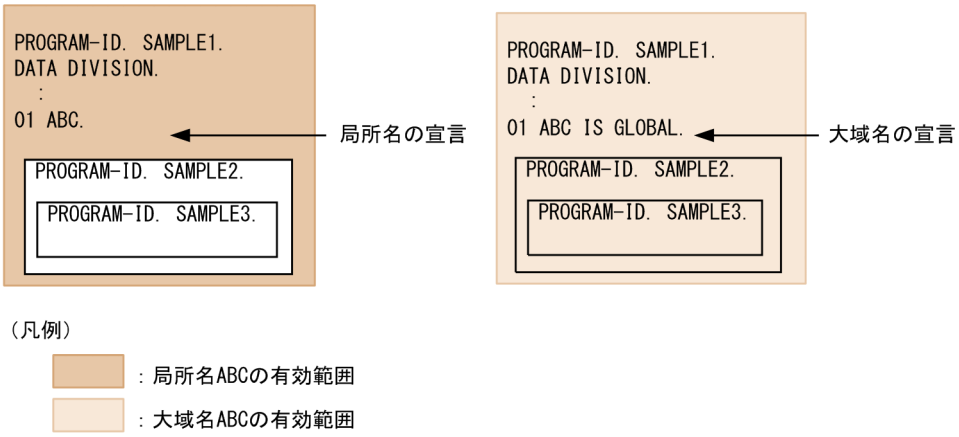
データ名やファイル名に GLOBAL 句を指定すると大域名となります。指定しない場合は，局所名となります。

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 領域に VARIANT データ項目を含む場合の注意事項

EXTERNAL 領域に VARIANT データ項目を含む場合，次の規則に従って初期値（X'00'）を設定しておく必要があります。

- 共有可能属性のとき

  EXTERNAL 領域に該当する外部変数は，他言語側で初期化なし，または X'00'を初期値として持つ外部変数として定義します。
- 共有不可能属性のとき

  環境変数 CBLEXVALUE に NULL を指定します。

(3) 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
- SYMBOLIC KEY 句の有無，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，-IsamExtend，-IsamExtend,Zone，-IgnoreLCC，-CompatiM7，-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「(3) 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
- APPLY FILE-SHARE 句の有無

  -IgnoreAPPLY,FILESHARE オプション指定時は，覚え書きとみなし，指定なしとして扱います。
- DATA FORMAT 句の指定の有無と指定内容
- 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
データベース操作	• 埋め込み SQL 文	23. データベース操作機能
画面	• ACCEPT※2	12. 画面入出力機能

分類	文	参照先
	• DISPLAY※2 • ERASE • REPLY • USE※2 • WAIT	
OLE2 オートメーション機能	• INVOKE※2 • SET※2	25.　OLE2 オートメーション機能
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

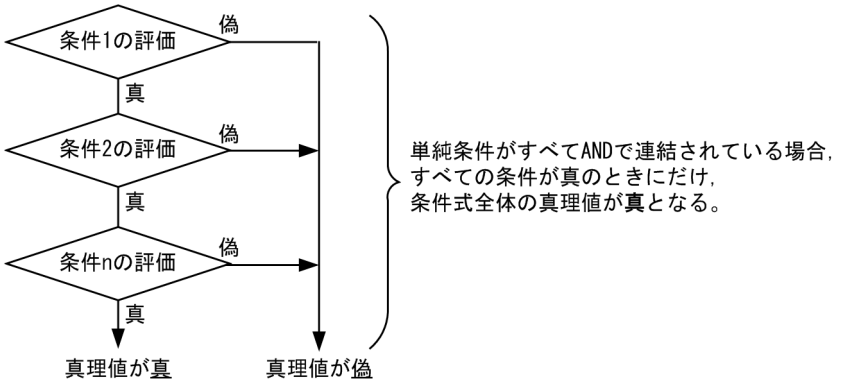
条件式の評価の流れを，幾つかの場合に分けて説明します。

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)

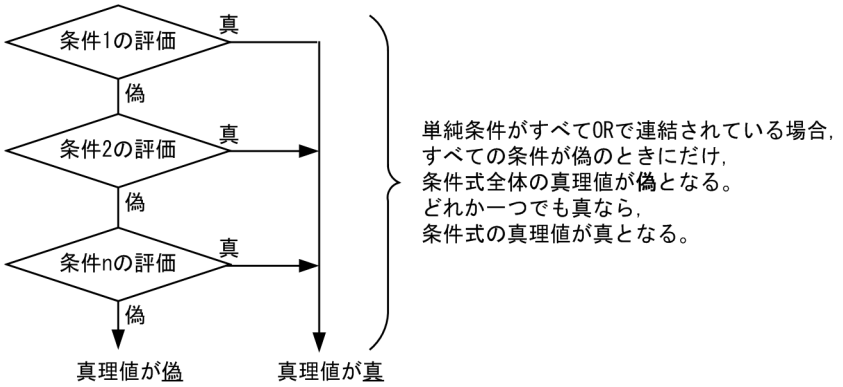
```
IF A = 10 AND B = 20 AND C = 30
  THEN
    DISPLAY '--- THEN ---'
  ELSE
    DISPLAY '--- ELSE ---'
END-IF.
```

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)

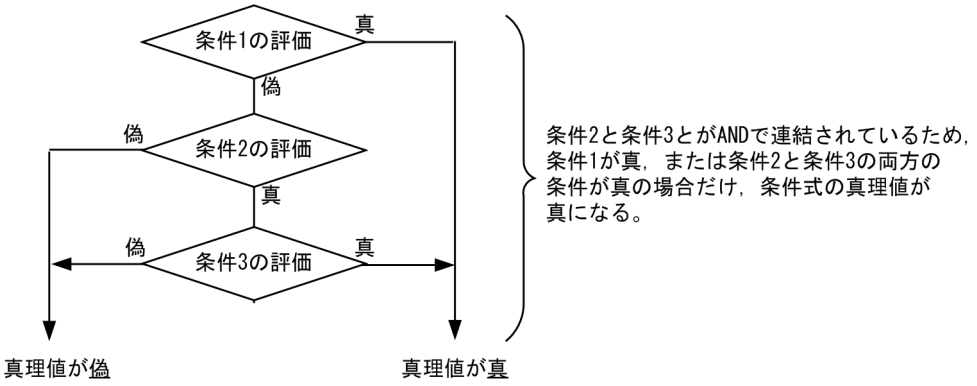
```
IF A = 10 OR B = 20 OR C = 30
  THEN
    DISPLAY '--- THEN ---'
  ELSE
    DISPLAY '--- ELSE ---'
END-IF.
```

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)

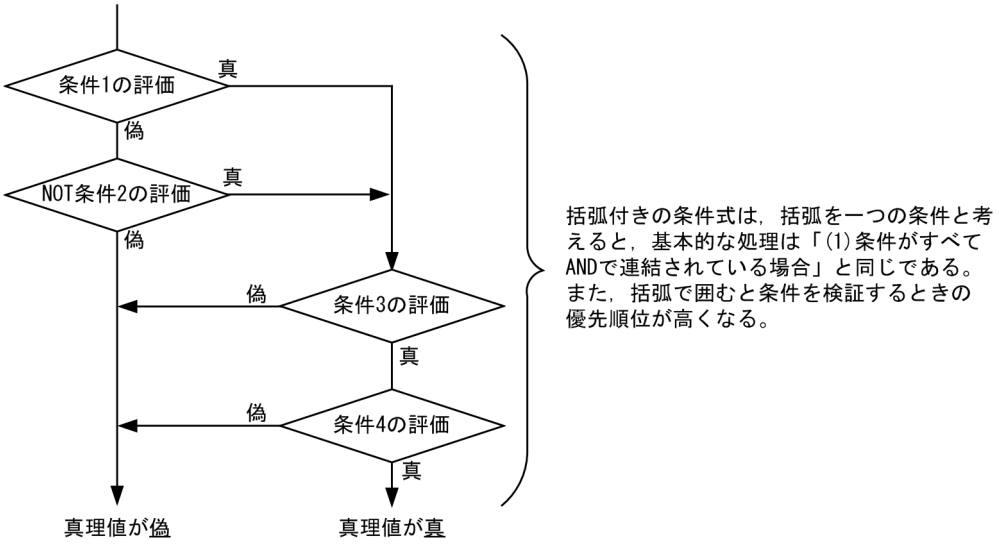
```
IF A = 10 OR B = 20 AND C = 30
  THEN
    DISPLAY '--- THEN ---'
  ELSE
    DISPLAY '--- ELSE ---'
END-IF.
```

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。

- 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。

- 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	$I_1 + I_2$	$D_1 + D_2$

演算の種類	整数部のけた数（I）	小数部のけた数（D）
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。

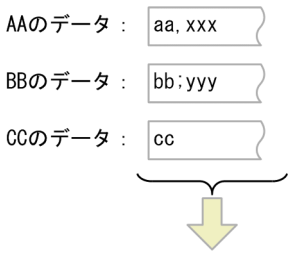
STRING 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.47　STRING 文」を参照してください。

STRING 文の例を次に示します。

（例 1）

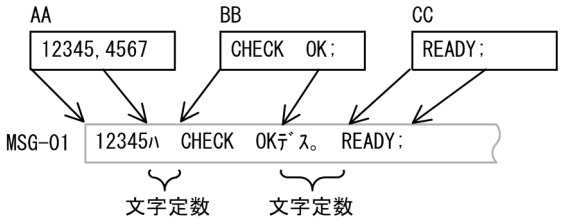
受け取り開始位置を指定しない（POINTER 指定のない）STRING 文

- 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。

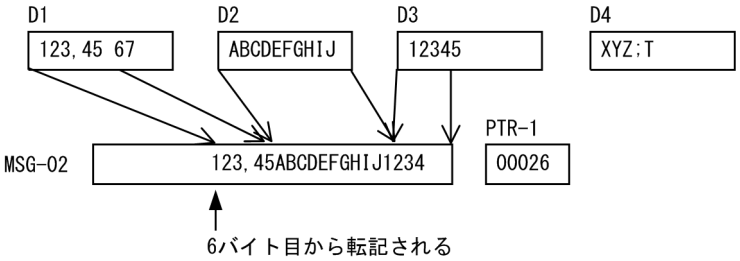
（例 2）

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

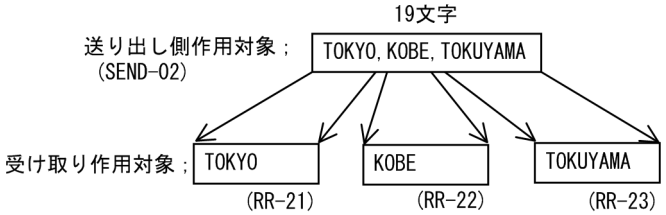
UNSTRING 文は，送り出し側作用対象の連続したデータを分解し，幾つかの受け取り側作用対象データ項目に移します。

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。

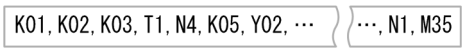


```
       77 SEND-02 PIC X(19).
       77 RR-21   PIC X(8).
       77 RR-22   PIC X(8).
       77 RR-23   PIC X(8).
             :
           UNSTRING SEND-02 DELIMITED BY ','
                    INTO RR-21 RR-22 RR-23.
```

(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。

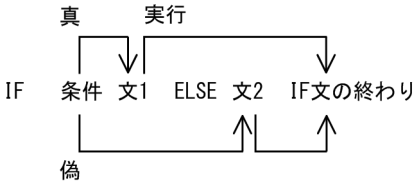
5.4.2　IF 文

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。

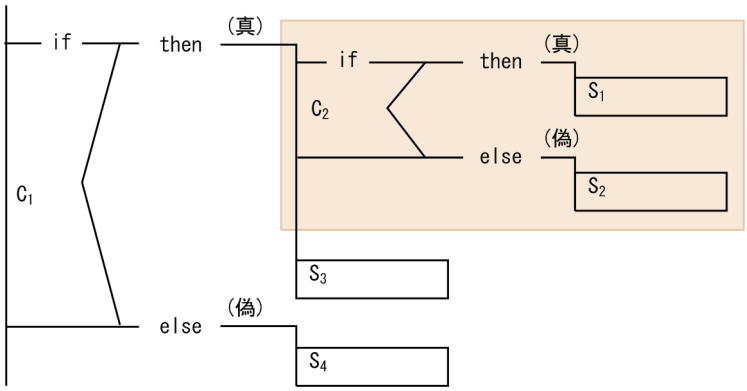
（例 1）

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
```

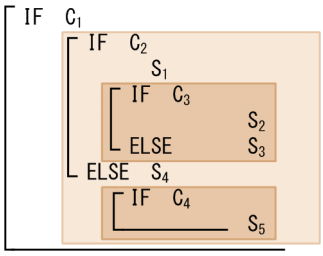
処理の流れ



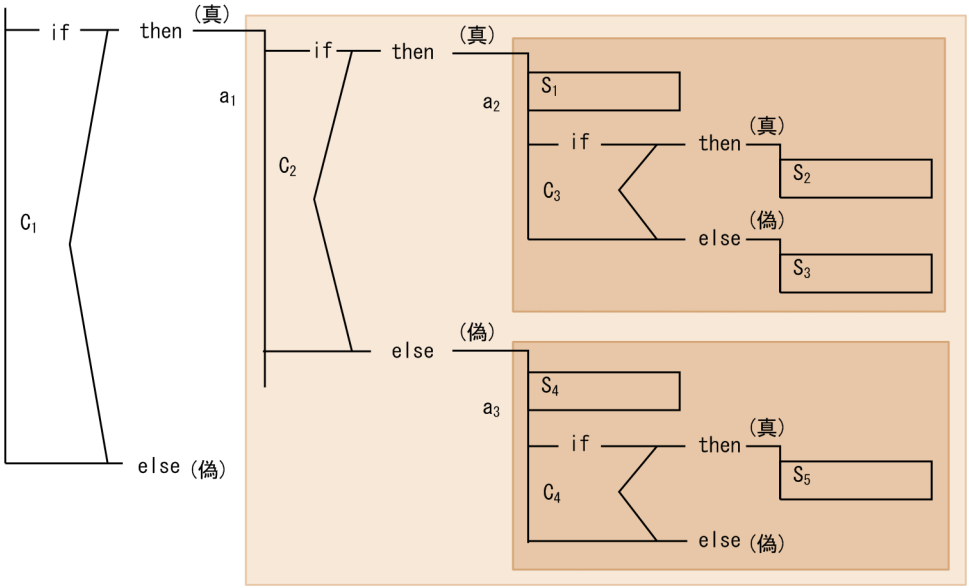
（例 2）

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。

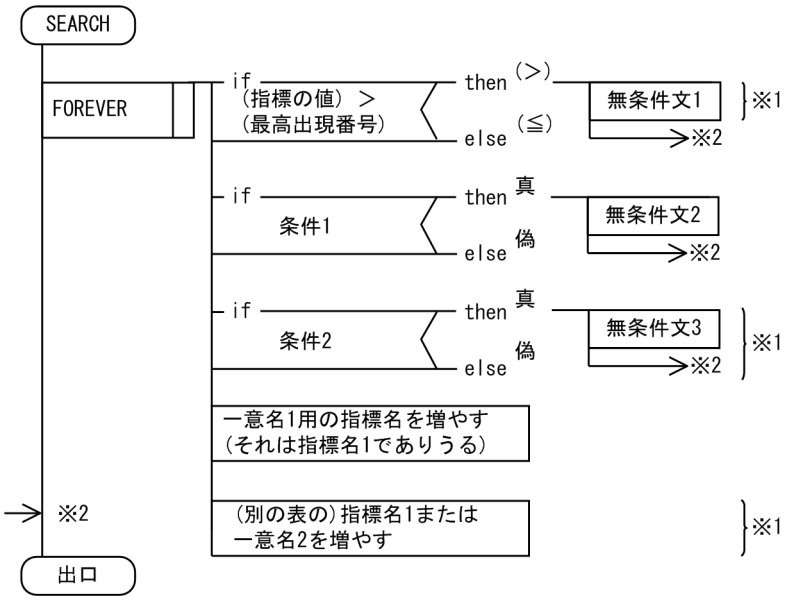
5.5.1　SEARCH 文

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。

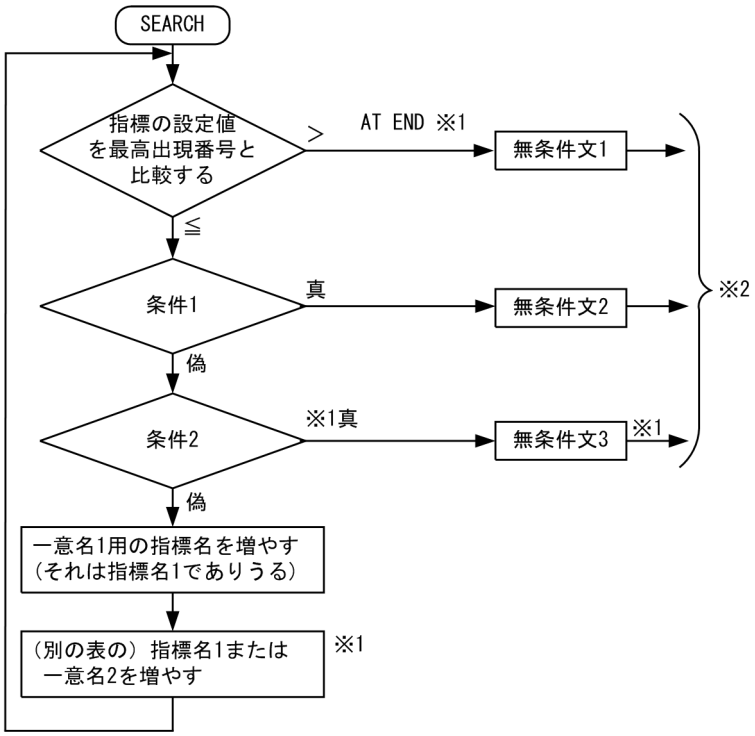
SEARCH 文については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.41　SEARCH 文」を参照してください。

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

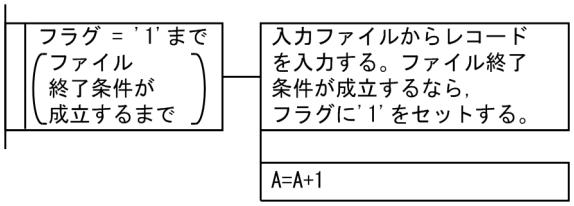
PERFORM（実行）文は，幾つかの手続きに明示的に制御を移し，指定した手続きの実行が終わると，暗黙的に制御を戻します。また，PERFORM 文は，その PERFORM 文の範囲に含まれる一つ以上の文の実行を制御するためにも使用されます。

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



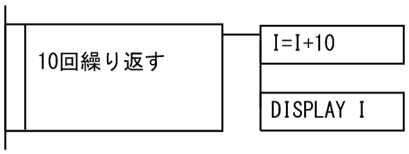
実行例

```
       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

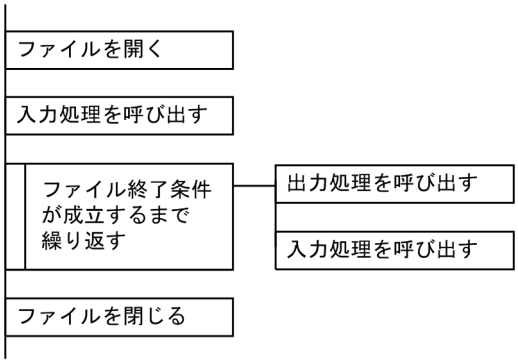
```
       PERFORM 10 TIMES      *>1.
         COMPUTE I = I + 10  *>2.
         DISPLAY I           *>3.
       END-PERFORM.
```

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
                :
```

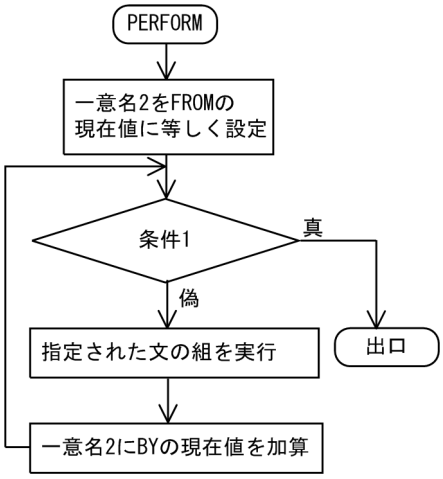
```
            IF A(J) = 0
              EXIT PERFORM  …1.
            END-IF
          END-PERFORM  …2.
            :
        END-PERFORM.
```

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。

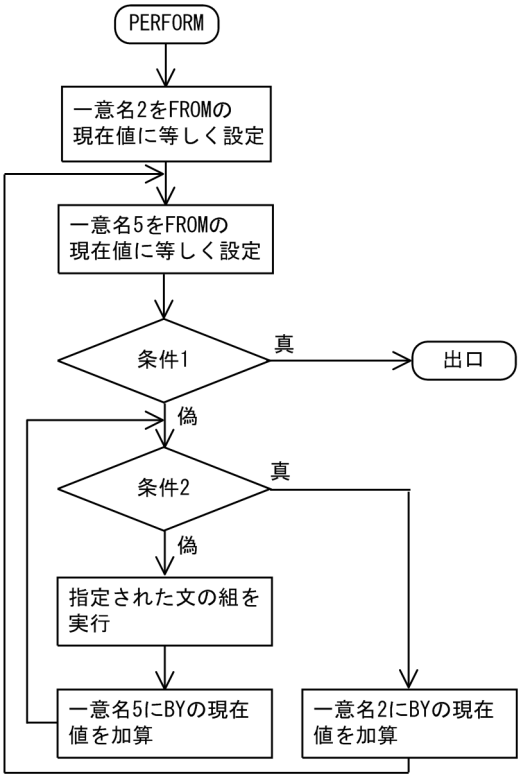
   条件 1 を満たしていれば，制御は PERFORM 文の終わりに移ります。

   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。

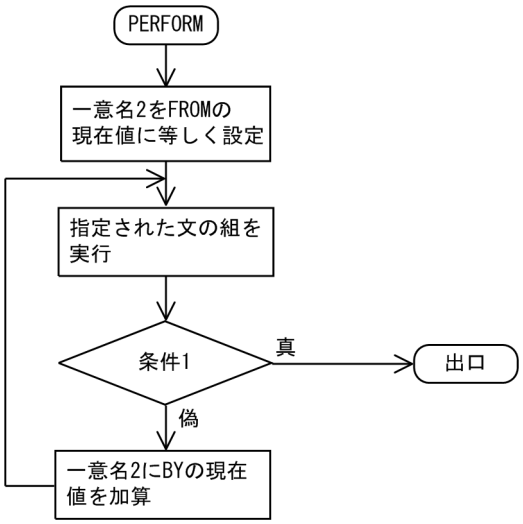
5. 指定された文の組（無条件文 1）を 1 回実行します。

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.
```

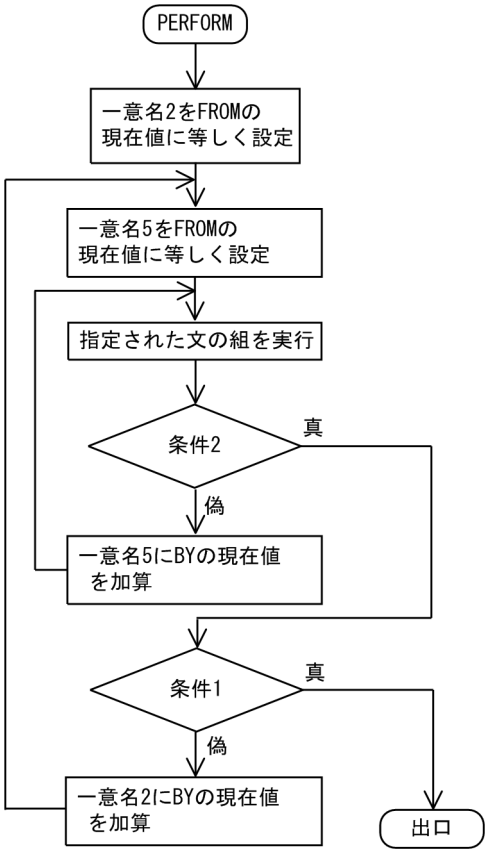
1. 一意名 2 のデータ項目の内容を，定数 1 または PERFORM 文開始時の一意名 3 のデータ項目の現在値に等しくします。
2. 指定された文の組（無条件文 1）を 1 回実行します。
3. 条件 1 を検査します。

   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

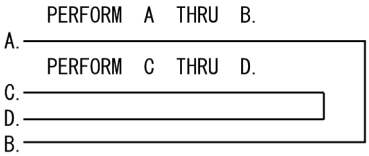
PERFORM 文の実行範囲にほかの PERFORM 文がある場合，含まれている PERFORM 文の実行範囲は，初めの PERFORM 文の論理的な実行範囲の内側に完全に含まれているか，または完全に外側である必要があります。PERFORM 文の実行範囲内で始められたほかの PERFORM 文の制御は，前者の PERFORM 文の出口を通ることはできません。また，実行中の 2 個以上の PERFORM 文は，共通の出口を持つこともできません。

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。

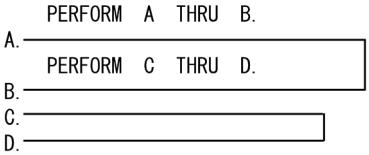
(例 1)

正しい例（完全に内側に含まれている）



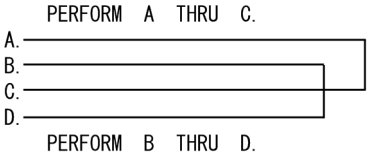
(例 2)

正しい例（完全に外側にある）



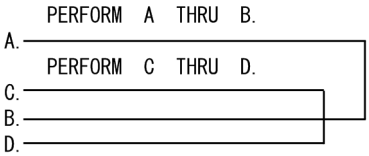
(例 3)

正しい例（同時に実行されていない）



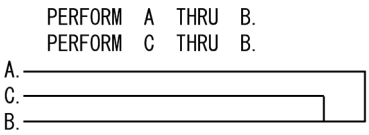
(例 4)

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
- OLE プロパティを取得・設定する
- OLE オブジェクトを参照したり，バリアントデータを設定したりする

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」，マニュアル「COBOL2002　言語　拡張仕様編」「13.5.8　SET 文（WINDOW SECTION)」，およびマニュアル

「COBOL2002　言語　拡張仕様編」「18.3.2　SET 文（OLE2 オートメーションインタフェース機能)」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)　OLE プロパティを取得・設定する SET 文

「25.2.2　OLE メソッドと OLE プロパティの操作」を参照してください。

(7) OLE オブジェクトを参照したり，バリアントデータを設定したりする SET 文

「25.2.3　OLE メソッドが返す OLE オブジェクトを利用した参照」および「25.2.5　VARIANT 値と COBOL データのやり取り」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
	Btrieve による索引編成ファイル※3		Btrieve または Pervasive.SQL	• COBOL85 または COBOL2002 で作成した Btrieve ファイル • Btrieve ファイルマネージャ，または Pervasive.SQL のユティリティで作成したファイル
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 • Excel などの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。

注※3

Windows(x86) COBOL2002 で有効です。

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。
- 物理ファイル名に次の名称は使用できません。
  - TAPE
  - TAPE:
- COBOL2002 が使用するファイル名やプログラム名に，COM1 や LPT1 などのシステムが予約しているデバイス名は使用できません。
- 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，「6.11　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
  - ISAM による索引順編成ファイル

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「C:¥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。

また，「環境変数指定」では，環境変数を指定しないで COBOL プログラムを実行した場合，実行中に物理ファイルを割り当てられます。これを「物理ファイルの動的割り当て」と呼びます。

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，ドライブ名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。
- フォルダ名，ファイル名に NULL 文字（X'00'）を含んではなりません。NULL 文字を含んだフォルダ名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，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フォルダ名に相当するフォルダがないと，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。
- ドライブ名を省略した場合，および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。
- CUI モードのときに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。
- CUI モードのときに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。
- GUI モードの場合で，定数に stdin，stdout，または stderr を指定したとき，これらは物理ファイル名とみなされます。
- 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。

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	FS=00

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
		再作成する※2	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，C ドライブの"DIR"フォルダに格納されている"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 'C:¥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，ドライブ名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。

- フォルダ名，ファイル名に NULL 文字（X'00'）を含んではなりません。NULL 文字を含んだフォルダ名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フォルダ名に相当するフォルダがないと，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。
- ドライブ名を省略した場合，および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。
- CUI モードのときに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。
- CUI モードのときに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。
- GUI モードの場合で，環境変数に stdin，stdout，または stderr を指定したとき，これらは物理ファイル名とみなされます。
- 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，次のようになります。

（CUI モードの場合）

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。

（GUI モードの場合）

動的割り当て画面が表示されます。詳細は，「6.2.4　実行時の動的割り当て（GUI モードの場合だけ）」を参照してください。

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
	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"C:¥DIR¥FILE1.FIL"を指定する例を，次に示します。

```
CBL_SYS_01=C:¥DIR¥FILE1.FIL
```

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，ドライブ名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。
- フォルダ名，ファイル名に NULL 文字（X'00'）を含んではなりません。NULL 文字を含んだフォルダ名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，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フォルダ名に相当するフォルダがないと，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。
- ドライブ名を省略した場合，および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。
- CUI モードのときに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。

- GUI モードの場合で，データ名に stdin，stdout，または stderr を指定したとき，これらは物理ファイル名とみなされます。
- 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。

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，C ドライブの"DIR"フォルダに格納されている"FILE1.FIL"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 'C:¥DIR¥FILE1.FIL' TO FILE-NAME.
      :
```

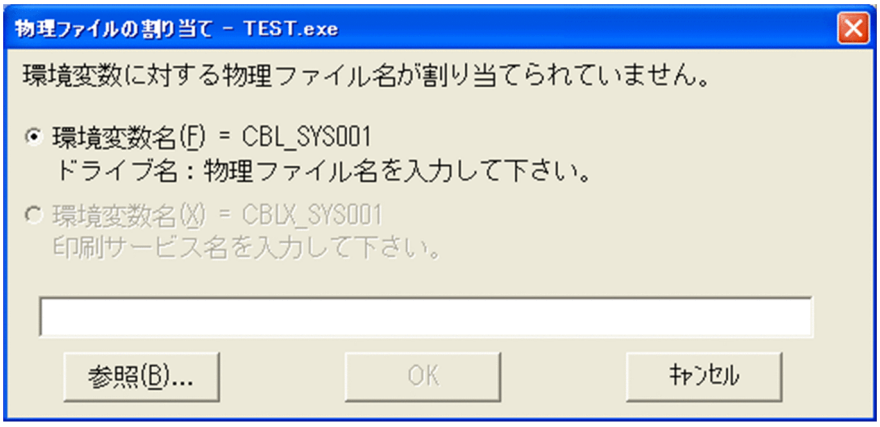
6.2.4　実行時の動的割り当て（GUI モードの場合だけ）

ASSIGN 句で処理系作成者語（外部装置名）を指定した場合，通常は環境変数で物理ファイルを割り当てますが，割り当てられていないファイルに対して OPEN 文実行時に動的に物理ファイルを割り当てることもできます。

また，書式印刷機能を使用する場合，同様に環境変数で割り当てられていない印刷サービス名称を OPEN 文実行時に割り当てることもできます。書式印刷機能の詳細は，「8.6　XMAP3 による印刷(Windows(x86) COBOL2002 で有効)」を参照してください。

(1)　物理ファイルの割り当て画面の説明

OPEN 文実行時，環境変数が設定されていない場合に出力される物理ファイルの割り当て画面を次に示します。



物理ファイルに出力する場合

ダイアログボックスの上段のボタンを選び，物理ファイル名をドライブ名からの絶対パス名で入力して［OK］ボタンを選びます。

書式印刷機能を使用してプリンタへ出力する場合

ダイアログボックスの下段のボタンを選び，印刷サービス名を入力して［OK］ボタンを選びます。

上記のどちらの場合も，［キャンセル］ボタンを選ぶと，外部装置への割り当てがないものとして OPEN 文が実行されます。

(2)　規則

COBOL プログラムでのファイル定義やコンパイラオプションの指定の有無によって，物理ファイル名または印刷サービス名のボタンが次のように無効になることがあります。

表 6-2　COBOL プログラムとボタンの有効・無効との関係

COBOL プログラムの指定			選択ボタンの有効／無効	
ファイル編成	-XMAP,LinePrint オプションの指定	APPLY FORMS-OVERLAY 句の指定	CBL_xxx (物理ファイル名)	CBLX_xxx (印刷サービス名)
順編成	指定なし	－	○	×
	指定あり	指定なし	○	○
		指定あり	×	○
HiRDB による索引編成ファイル	－	－	○	×
上記以外	－	－	○	×

（凡例）

○：有効

×：無効

－：指定とは無関係

また，物理ファイル名の選択ボタンが無効の場合，印刷サービス名の選択ボタンが選ばれている場合，またはファイル編成が HiRDB による索引編成ファイルの場合，［参照］ボタンも無効となります。

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 6-3　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

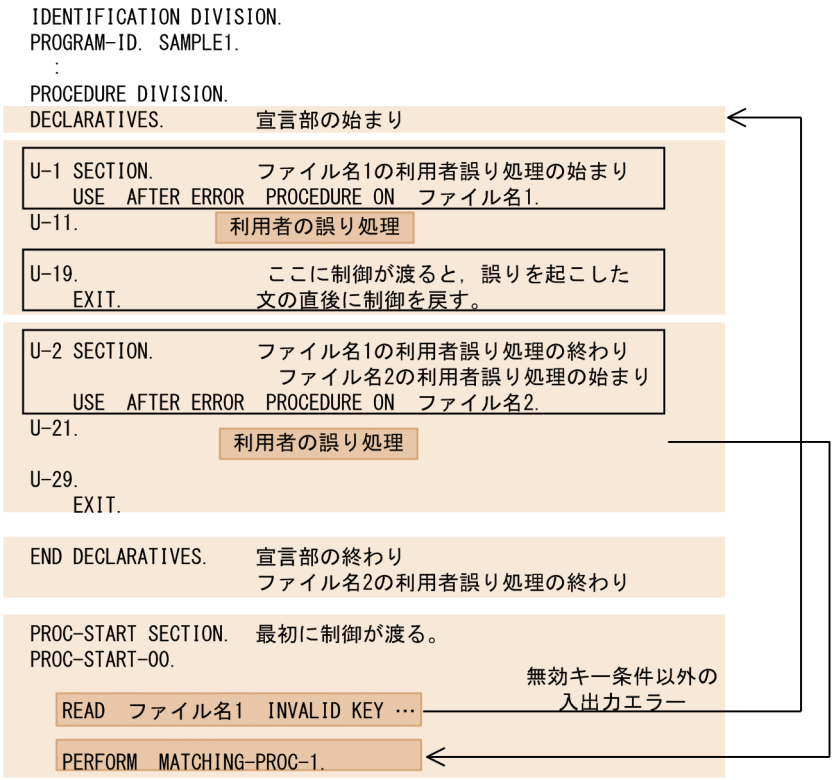
FILE STATUS 句を指定しておくと，入出力文でエラーが発生した場合，FILE STATUS 句で指定したデータ項目に入出力状態の値が設定されます。入出力状態の値と意味については，「付録 H　入出力状態の値」およびマニュアル「COBOL2002　言語　標準仕様編」「5.1.12　入出力状態」を参照してください。

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。

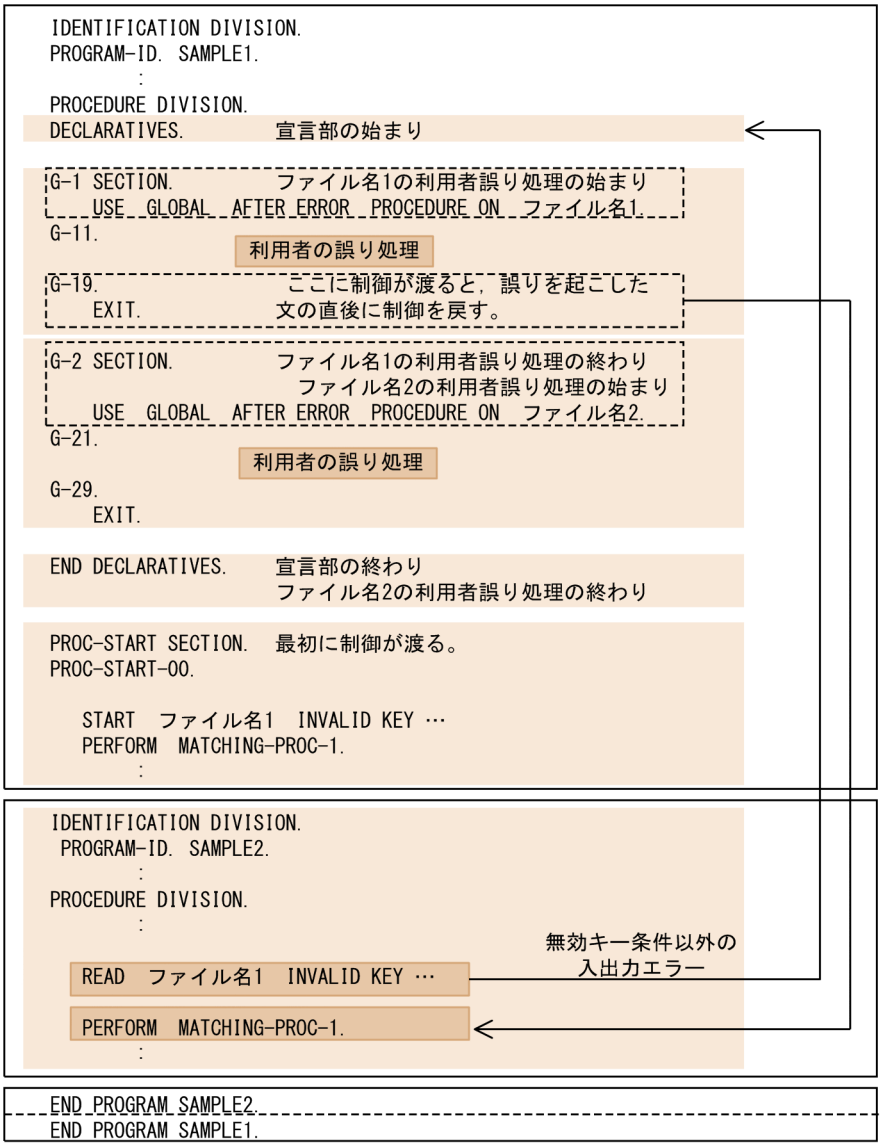
図 6-1　USE ERROR 手続き処理のコーディング例



(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，およびデータ形式（DATA FORMAT 句の指定）※をチェックします。

  注※

  データ形式（DATA FORMAT 句の指定）は，物理ファイル内に格納されないため，指定したシステムのエンディアン形式と矛盾していないかチェックします。そのため，ファイルを新規に作成する場合でも，矛盾があると判断された場合は，ファイル作成に失敗します。
- Btrieve 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，最小レコード長，キー個数，主レコードキー，および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
- Windows のメモ帳などのエディタ
- 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

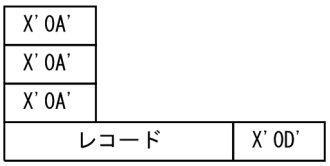
- 「6.2　ファイル割り当ての共通規則」に従って，物理ファイル名を割り当ててください。
  物理ファイル名には，OS のファイルシステム上で有効となるファイル名を指定してください。
- 順編成ファイルを使用する場合は，ORGANIZATION 句に SEQUENTIAL を指定します。

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。

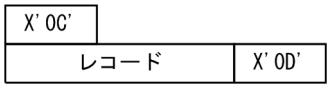
出力例を次に示します。出力例の X'0A'は改行，X'0D'は復帰，X'0C'は改ページを示します。

WRITE REC AFTER ADVANCING 3 を実行した場合

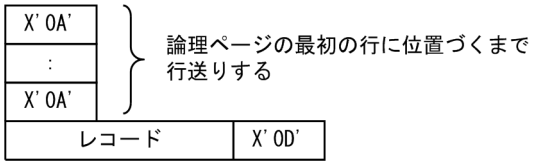


WRITE REC AFTER ADVANCING PAGE を実行した場合

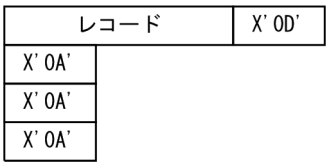
- LINAGE 句の指定がない場合



- LINAGE 句の指定がある場合



WRITE REC BEFORE ADVANCING 3 を実行した場合



6.5.1　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。
- ネットワークに接続してあるドライブからファイルを割り当てる場合，ISAM ユティリティのオプション情報設定で，ネットワークドライブ情報を「使用する」と設定する必要があります。ただし，この場合ファイルの整合性は保証しません。詳細はマニュアル「索引順編成ファイル管理 ISAM」を参照してください。

(4)　生成される物理ファイル

索引編成ファイルには，2GB 未満までアクセス可能なノーマルファイル形式と，2GB 以上のアクセスが可能なラージファイル形式の 2 種類があります。ただし，ラージファイル形式の索引編成ファイルは，Windows(x86) COBOL2002 だけで有効です。

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		.kdf （ノーマルファイル形式） .kdl （ラージファイル形式）	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 （ノーマルファイル形式） .l01 （ラージファイル形式）	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 （ノーマルファイル形式） .l02～.l99 （ラージファイル形式）	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		.drf （ノーマルファイル形式） .drl （ラージファイル形式）	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

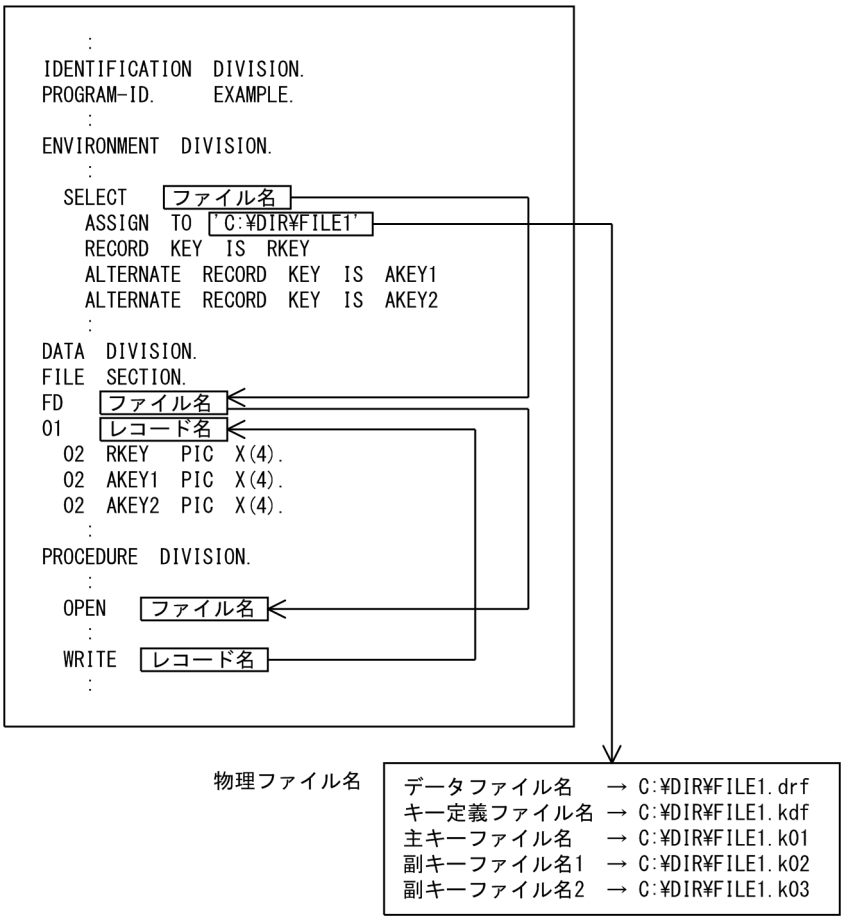
追加書き：存在するファイルに対して，追加で出力する

(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

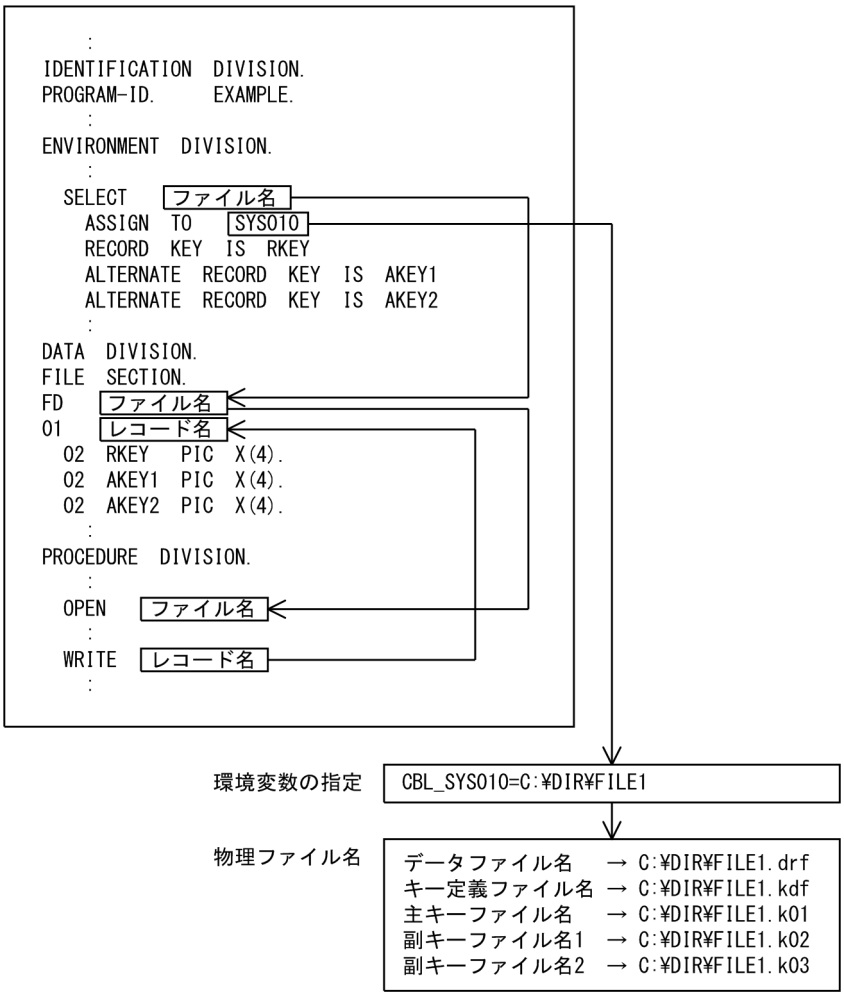
COBOLプログラム



物理ファイル名

(b)　環境変数指定の ASSIGN 句の場合

COBOLプログラム

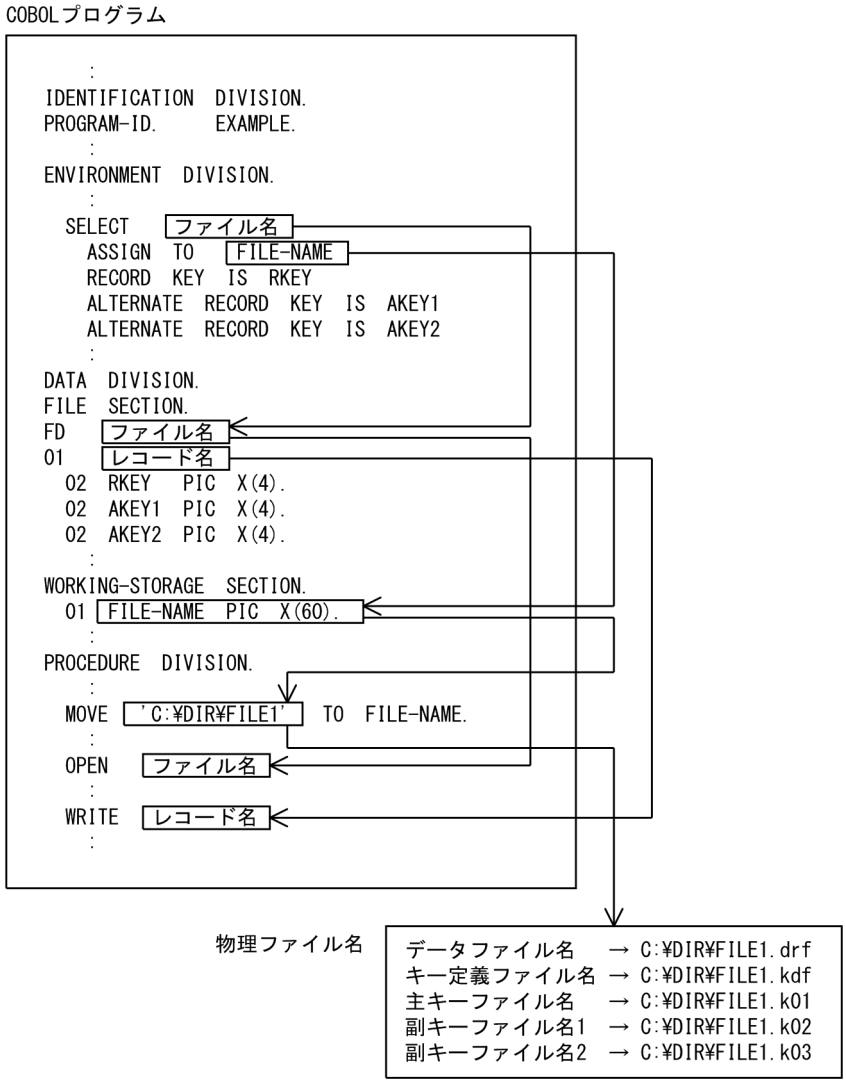


環境変数の指定　CBL_SYS010=C:¥DIR¥FILE1

物理ファイル名

データファイル名　→ C:¥DIR¥FILE1.drf
キー定義ファイル名 → C:¥DIR¥FILE1.kdf
主キーファイル名　→ C:¥DIR¥FILE1.k01
副キーファイル名1 → C:¥DIR¥FILE1.k02
副キーファイル名2 → C:¥DIR¥FILE1.k03

(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 リモートファイルアクセス機能（Windows(x86) COBOL2002 で有効）

接続されているほかの Windows や UNIX 上にある，ISAM を使用する索引編成ファイルにアクセスできます。これを，リモートファイルアクセス機能といいます。

(1) リモートファイルのアクセス方法

他マシン上の ISAM ファイルにアクセスする方法を説明します。

物理ファイルの割り当て時，次のように他ホスト名を指定すると，他マシン上の ISAM を使用する索引編成ファイルを割り当てられます。

形式

```
ホスト名:〔ドライブ名:〕ファイル名
```

構文規則

- ドライブ名は，Windows 上にある ISAM を使用する索引編成ファイルにアクセスする場合に指定が必要です。
- ホスト名を指定しなかった場合は，ローカルファイルとしてアクセスします。

一般規則

- Windows と UNIX の間で 2 進データ，内部浮動小数点データなどのバイナリデータをアクセスした場合，動作は保証しません。
- ラージファイル形式の索引編成ファイルは，リモートファイルアクセス機能を使用できません。

指定例

（例 1）UNIX 上の索引編成ファイルの場合

```
UNIXHOST:/users/isamfile
```

（例 2）Windows 上の索引編成ファイルの場合

```
PCHOST:C:¥users¥isamfile
```

(2) データ形式とデータ操作

リモートファイルにアクセスする場合，ファイル定義のデータ形式と物理ファイルのデータ形式の組み合わせによって，データの操作が異なります。

データ形式の組み合わせと，リモートファイルアクセス機能でのデータ操作の関係を次に示します。

ファイル記述項 DATA FORMAT 句の指定	物理ファイル	
	標準数値形式	反復数値形式
STANDARD（標準数値形式）	数値データを変換する	属性不一致で実行時エラーとなる
REVERSE（反復数値形式）	属性不一致で実行時エラーとなる	数値データを変換しない
指定なし	数値データを変換しない	数値データを変換しない

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
- Windows のメモ帳などのエディタ
- 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。

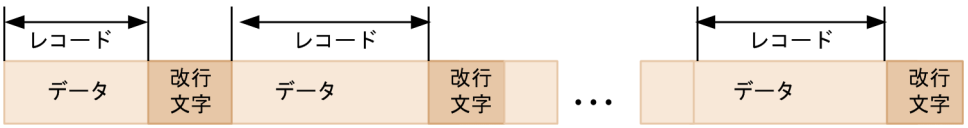
6.7.2　テキスト編成ファイルのファイル編成とレコード形式

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。

- 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'）がデータとして入力されます。また，NULL（X'00'）以降もデータとして入力されます。
- 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。
- テキスト編成ファイル中に EOF コード（X'1A'）が含まれる場合，その位置がファイルの終端とみなされます。
- 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，「28.　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 6-4　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 6-5　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。

なお，ACCEPT／DISPLAY 文を使用した CSV ファイルへのアクセス方法については，「付録 D.1 ACCEPT／DISPLAY 文を使用した CSV ファイルへのアクセス」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
- Windows のメモ帳などのエディタ
- 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

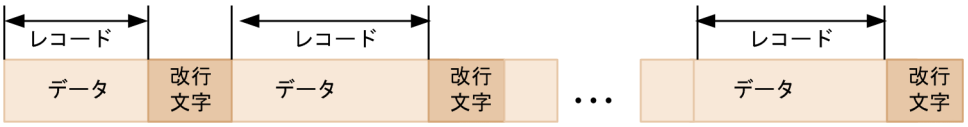
CSV 編成ファイルでは，テキスト編成ファイルと同じように，1 行（改行文字までの文字列）を 1 レコードとみなします。さらに，おのおののレコードは，コンマ（,）によって複数のデータ項目に区切られた形式になっています。コンマによって区切られたデータ項目をセルと呼びます。CSV 編成ファイルでは，このセル単位でデータを入出力できます。

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。



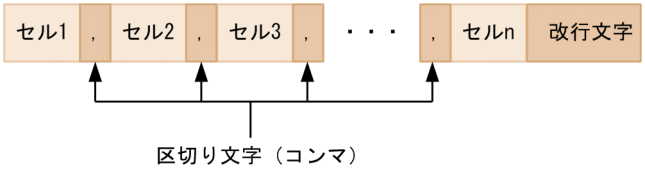
(2)　レコード形式

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，「28.　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 6-6　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 6-6　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。

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		
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

未使用基本項目の属性	初期値データ
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

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，「28. 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

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

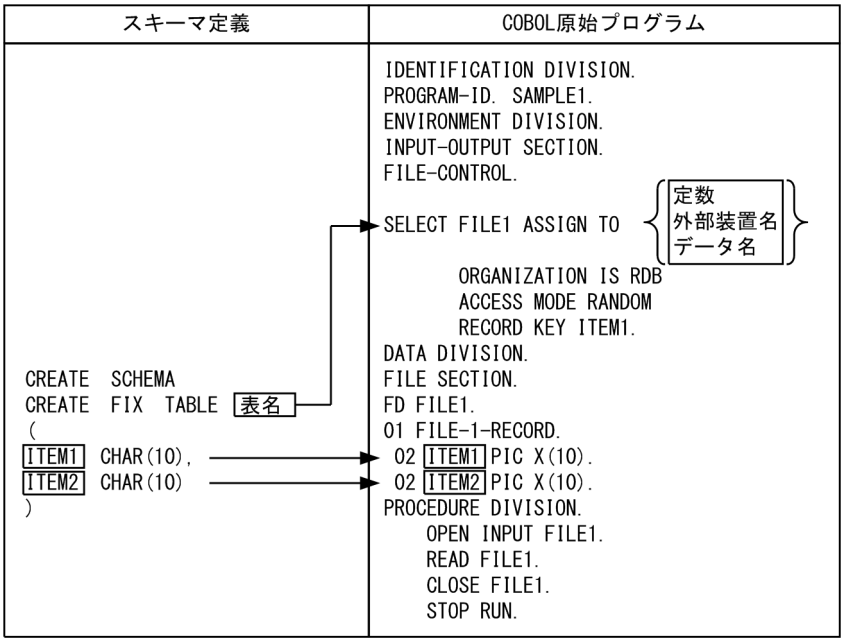
COBOL の実行単位で，HiRDB による索引編成ファイルに対するすべてのファイルを閉じたときに，HiRDB から切り離されます。

(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



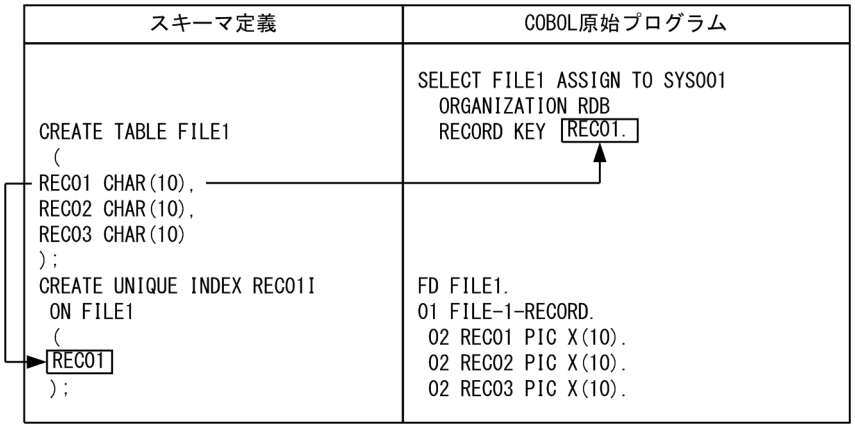
規則

- 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。

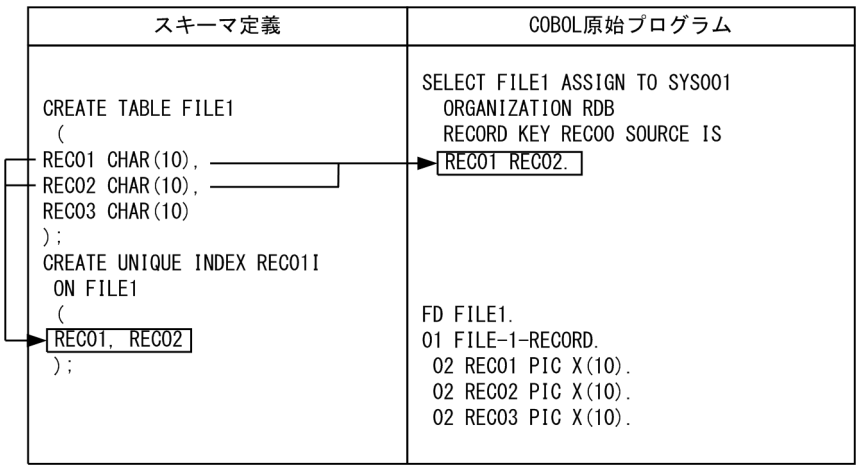
  1.単一列インデクスの場合（合成キー以外の場合）

  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 6-7 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, Windows(x64) COBOL2002 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。

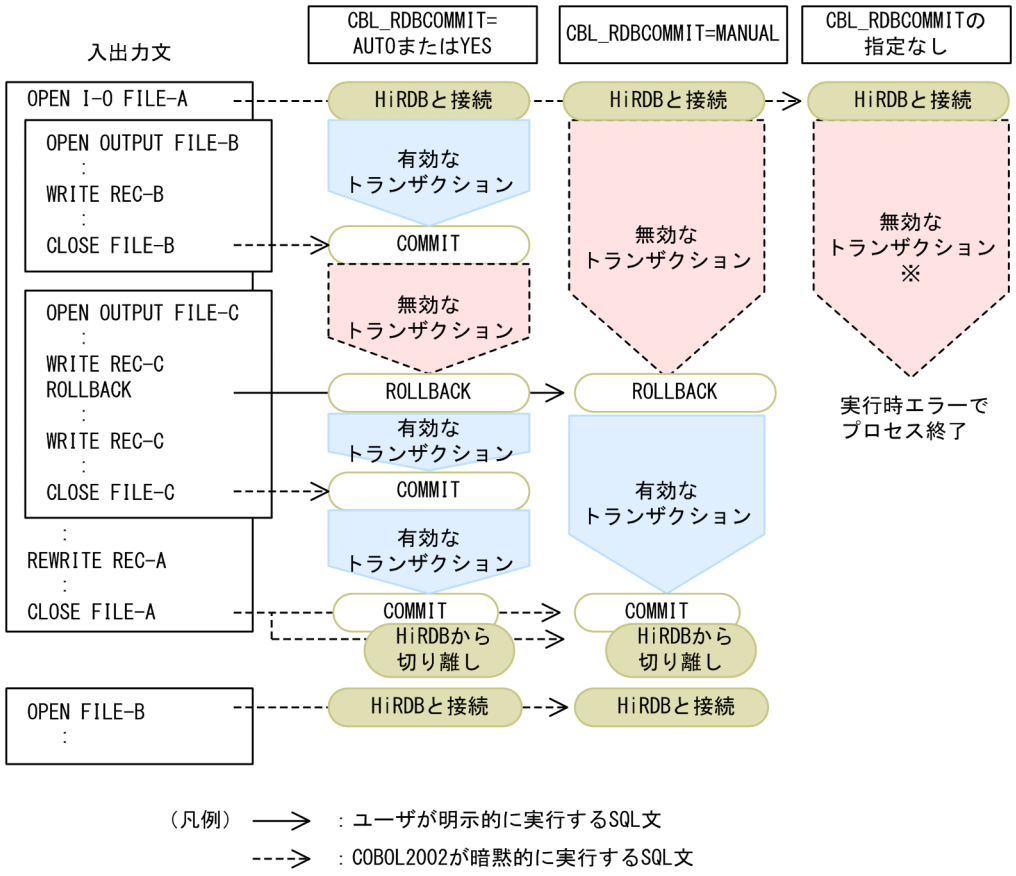
それ以外の値を指定した場合，または環境変数 CBL_RDBCOMMIT を指定しなかった場合

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 6-8　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 H　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 6-9　-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，または HiRDB.ini 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 Btrieve（Pervasive.SQL）による索引編成ファイル（Windows(x86) COBOL2002 で有効）

COBOL2002 では，Pervasive.SQL，または Btrieve で作成された Btrieve 形式のファイルを，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，Btrieve（Pervasive.SQL）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。

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」を参照してください。

6.10.1 前提条件とプログラムの作成方法

(1) 前提条件

Btrieve ファイルを使用するには，次の前提条件を満たしている必要があります。

- Pervasive.SQL，Btrieve のサーバ／クライアント版，またはスタンドアロン版のどれかが組み込まれていて，動作していること
- Pervasive.SQL，Btrieve アプリケーションがアクセスできる状態になっていること

(2) プログラムの作成方法

Btrieve（Pervasive.SQL）による索引編成ファイルを使用するには，-IsamExtend オプションまたは-IsamExtend,Zone オプションを指定してプログラムをコンパイルする必要があります。-IsamExtend オプションおよび-IsamExtend,Zone オプションを指定しない場合は，ISAM による索引編成ファイルを使用します。

6.10.2 ファイルの作成と割り当て方法

Btrieve（Pervasive.SQL）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

Btrieve（Pervasive.SQL）による索引編成ファイルは，次の方法で作成できます。

- COBOL の入出力機能
- Btrieve でアクセスする C プログラムなどによるファイル作成
- Btrieve ユティリティ

(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。

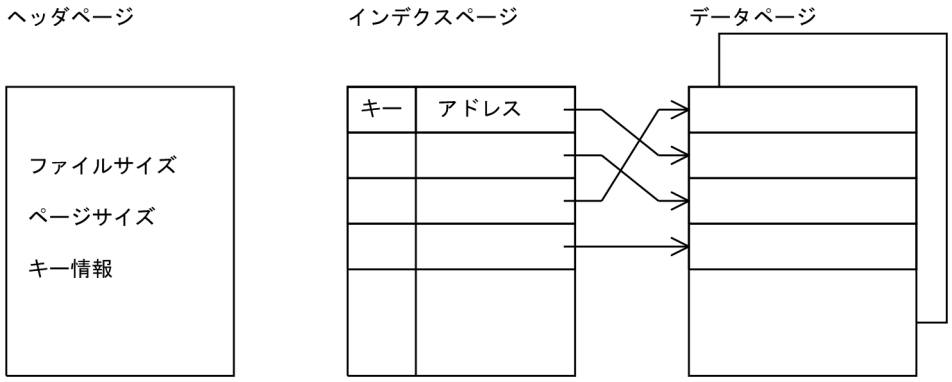
物理ファイル名には，OS のファイルシステム上で有効となるファイル名を指定してください。

6.10.3　ファイル編成とレコード形式

Btrieve（Pervasive.SQL）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。なお，ファイルは Pervasive.SQL，Btrieve が定めるファイル形式で構成されています。

通常，Btrieve（Pervasive.SQL）による索引編成ファイルはヘッダページ，インデクスページ，データページの 3 種類で構成されます。Btrieve（Pervasive.SQL）による索引編成ファイルの構成を次に示します。

図 6–5　Btrieve（Pervasive.SQL）による索引編成ファイルの構成



6.10.4　使用できる機能と制限事項

Btrieve（Pervasive.SQL）による索引編成ファイルで使用できる機能と，制限事項を次に示します。

(1)　使用できる機能

Btrieve（Pervasive.SQL）による索引編成ファイルでは，次の機能が使用できます。

(a)　入出力手続き文

OPEN 文，CLOSE 文，READ 文，WRITE 文，REWRITE 文，DELETE 文，START 文，UNLOCK 文が使用できます。

(b)　ファイル共用

ファイル排他，レコード施錠が使用できます。レコード施錠としては，自動単一レコード施錠，手動複数レコード施錠のどちらも使用できます。

(c) 合成キー

1～8 個の項目が指定できます。項目の合計は 255 バイト以下でなければなりません。

(d) 逆順読み

PREVIOUS 指定の READ 文が指定されたとき，Btrieve（Pervasive.SQL）による索引編成ファイルの逆順読みができます。

(e) キー属性

RECORD KEY 句，または ALTERNATE RECORD KEY 句のデータ名の項類が英数字項目以外のとき，次の規則に従って主レコードキーおよび副レコードキーの一致，不一致を判定します。

主レコードキーまたは副レコードキーの項類	キーの判定属性	
	-IsamExtend 指定（Zone サブオプションなし）	-IsamExtend,Zone 指定（Zone サブオプションあり）
固定長集団項目	文字	文字
外部 10 進項目	文字	外部 10 進項目※
内部 10 進項目	文字	文字
外部浮動小数点数字項目	文字	文字
数字編集項目	文字	文字
英字編集項目	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

注※

外部 10 進項目キーで Btrieve（Pervasive.SQL）による索引編成ファイルに対して入出力する場合，COBOL で定義された外部 10 進項目キーは Btrieve ファイルで定義された Numeric 属性キーに相当します。ただし，符号付き外部 10 進項目に関するコード体系が異なるため，入出力時には「図 6-6　COBOL 外部 10 進項目から Btrieve ファイルの Numeric 属性への変換テーブル」「図 6-7　Btrieve ファイルの Numeric 属性から COBOL 外部 10 進項目への変換テーブル」に示す変換テーブルに従って，自動的に最右端 1 バイトをコード変換します。

また，外部 10 進項目キーに SIGN LEADING 指定，SIGN SEPARATE 指定がある場合は，文字として扱われます。

図 6-6　COBOL 外部 10 進項目から Btrieve ファイルの Numeric 属性への変換テーブル

下位4ビット	上位4ビット															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	10	20	30	40	50	60	7D	80	90	A0	B0	C0	D0	E0	F0
1	01	11	21	31	41	51	61	4A	81	91	A1	B1	C1	D1	E1	F1
2	02	12	22	32	42	52	62	4B	82	92	A2	B2	C2	D2	E2	F2
3	03	13	23	33	43	53	63	4C	83	93	A3	B3	C3	D3	E3	F3
4	04	14	24	34	44	54	64	4D	84	94	A4	B4	C4	D4	E4	F4
5	05	15	25	35	45	55	65	4E	85	95	A5	B5	C5	D5	E5	F5
6	06	16	26	36	46	56	66	4F	86	96	A6	B6	C6	D6	E6	F6
7	07	17	27	37	47	57	67	50	87	97	A7	B7	C7	D7	E7	F7
8	08	18	28	38	48	58	68	51	88	98	A8	B8	C8	D8	E8	F8
9	09	19	29	39	49	59	69	52	89	99	A9	B9	C9	D9	E9	F9
A	0A	1A	2A	3A	4A	5A	6A	7A	8A	9A	AA	BA	CA	DA	EA	FA
B	0B	1B	2B	3B	4B	5B	6B	7B	8B	9B	AB	BB	CB	DB	EB	FB
C	0C	1C	2C	3C	4C	5C	6C	7C	8C	9C	AC	BC	CC	DC	EC	FC
D	0D	1D	2D	3D	4D	5D	6D	7D	8D	9D	AD	BD	CD	DD	ED	FD
E	0E	1E	2E	3E	4E	5E	6E	7E	8E	9E	AE	BE	CE	DE	EE	FE
F	0F	1F	2F	3F	4F	5F	6F	7F	8F	9F	AF	BF	CF	DF	EF	FF

（凡例）　：コード変換が行われることを示します。

図 6-7　Btrieve ファイルの Numeric 属性から COBOL 外部 10 進項目への変換テーブル

下位4ビット	上位4ビット															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	10	20	30	40	77	60	70	80	90	A0	B0	C0	D0	E0	F0
1	01	11	21	31	31	78	61	71	81	91	A1	B1	C1	D1	E1	F1
2	02	12	22	32	32	79	62	72	82	92	A2	B2	C2	D2	E2	F2
3	03	13	23	33	33	53	63	73	83	93	A3	B3	C3	D3	E3	F3
4	04	14	24	34	34	54	64	74	84	94	A4	B4	C4	D4	E4	F4
5	05	15	25	35	35	55	65	75	85	95	A5	B5	C5	D5	E5	F5
6	06	16	26	36	36	56	66	76	86	96	A6	B6	C6	D6	E6	F6
7	07	17	27	37	37	57	67	77	87	97	A7	B7	C7	D7	E7	F7
8	08	18	28	38	38	58	68	78	88	98	A8	B8	C8	D8	E8	F8
9	09	19	29	39	39	59	69	79	89	99	A9	B9	C9	D9	E9	F9
A	0A	1A	2A	3A	71	5A	6A	7A	8A	9A	AA	BA	CA	DA	EA	FA
B	0B	1B	2B	3B	72	5B	6B	30	8B	9B	AB	BB	CB	DB	EB	FB
C	0C	1C	2C	3C	73	5C	6C	7C	8C	9C	AC	BC	CC	DC	EC	FC
D	0D	1D	2D	3D	74	5D	6D	70	8D	9D	AD	BD	CD	DD	ED	FD
E	0E	1E	2E	3E	75	5E	6E	7E	8E	9E	AE	BE	CE	DE	EE	FE
F	0F	1F	2F	3F	76	5F	6F	7F	8F	9F	AF	BF	CF	DF	EF	FF

（凡例）　：コード変換が行われることを示します。

(2) 物理的制限事項

Btrieve（Pervasive.SQL）による索引編成ファイルの物理的制限事項を，次に示します。なお，詳細については，Pervasive.SQL，Btrieve のマニュアルを参照してください。

最大レコード長

- 固定長の場合：ページサイズ-8（ページ情報）
- 可変長の場合：65,535 バイト

  ただし，NetWare(R)サーバの場合，57,000 バイトです。

Btrieve（Pervasive.SQL）による索引編成ファイルで使用するページサイズは，環境変数 CBL_BTRPGSZ で指定できます。指定できる値は，512～4,096 までのうちで 512 の倍数です。512 の倍数でない場合は，512 の倍数に切り上げられます。

(3) 機能的制限事項

Btrieve（Pervasive.SQL）による索引編成ファイルの機能的制限事項を，次に示します。

(a) DATA FORMAT 句

Btrieve（Pervasive.SQL）による索引編成ファイルでは，UNIX のデータにアクセスできません。このため，DATA FORMAT 句を使用したプログラムに-IsamExtend オプションまたは-IsamExtend,Zone オプションを指定してコンパイルすると，エラーになります。

(b) セキュリティ

ファイル共用は使用できますが，Btrieve（Pervasive.SQL）のオーナ　ネームは使用できません。したがって，ファイルにアクセスする場合は，オーナと同等の権限が必要となります。

(c) OUTPUT モードの追加書き

OUTPUT モード指定の場合，ファイルの追加書きはしません。常にファイルを再作成します。

(d) 重複キーの検出

副レコードキーの重複を許すようにした場合，キーが重複していても検出できません。

(4) ISAM による索引編成ファイルとの相違点

Btrieve（Pervasive.SQL）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を，次に示します。

- 参照キーを更新した場合，後行の READ 文 NEXT 指定または READ 文 PREVIOUS 指定は，更新されたキーで次のレコードを入力します。
- 順アクセスで入力したレコードを削除した場合，ファイル位置指示子が不定となり，後行の READ 文 NEXT 指定または READ 文 PREVIOUS 指定が実行時エラーとなります。

- ほかで使用中のファイルを OUTPUT 指定の OPEN 文で実行した場合，ファイル排他が検知できません。この場合，オープンエラーとなります。
- ほかで施錠しているレコードに対して，施錠指定のない READ 文を実行した場合，施錠エラーが検知できません。
- START 文の KEY 指定には，レコードキーとして指定したデータ名を指定してください。レコードキーとして指定していないデータ名を指定した場合，コンパイルエラーとなります。
- START 文で位置づけられたレコードを削除したり，レコードの参照キーを変更した場合，あとに続く READ 文 NEXT 指定や READ 文 PREVIOUS 指定の動作は保証しません。

6.11　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，ファイルサイズが 2GB 以上のファイル（ラージファイル）に対する入出力ができます。

6.11.1　ラージファイル入出力機能の概要

次のファイルの場合，ファイルサイズが 2GB 以上のファイル（ラージファイル）であっても，COBOL プログラムから入出力できます。ラージファイルの入出力方法は，通常のファイル（ファイルサイズが 2GB 未満のファイル）と同じです。

ラージファイルで，入出力できるファイルサイズはファイルシステムに依存します。

(1)　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。

Windows(x86) COBOL2002 の場合

- 順編成ファイル
- テキスト編成ファイル
- CSV 編成ファイル
- ISAM による索引編成ファイル

なお，ISAM による索引編成ファイルについては実行時環境変数の指定が必要です。詳細は，「6.11.2　ISAM による索引編成ファイルでのラージファイル入出力機能（Windows(x86) COBOL2002 で有効）」を参照してください。

ファイルサイズの詳細は，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。

Windows(x64) COBOL2002 の場合

- 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。

6.11.2 ISAMによる索引編成ファイルでのラージファイル入出力機能（Windows(x86) COBOL2002で有効）

ISAMによる索引編成ファイルでラージファイル入出力機能を使用するためには，実行時環境変数を指定します。

(1) ファイル単位に指定する方法

形式

```
CBLD_ファイル名= { ISAMLARGE | NOISAMLARGE }
```

ISAMLARGEを指定した場合，ISAMによる索引編成ファイルでラージファイル入出力を可能にします。NOISAMLARGEを指定した場合は，ノーマルファイル形式のISAMによる索引編成ファイルへの入出力となります。省略時は，NOISAMLARGEが仮定されます。

ISAMLARGE

ISAMによる索引編成ファイルに対して，ラージファイル形式のファイル入出力機能が有効になります。

NOISAMLARGE

ISAMによる索引編成ファイルに対して，ノーマルファイル形式のファイル入出力を行います。

(2) 実行単位中のすべてのファイルに指定する方法

形式

```
CBLISAMLARGE=YES
```

入出力するISAMによる索引編成ファイルに対して一括でラージファイル形式を指定する場合，この実行時環境変数にYESを指定します。この実行時環境変数にYES以外の値を指定した場合，この実行時環境変数は無効となります。

(3) 注意事項

ISAMによる索引編成ファイルでラージファイル入出力機能使用時の注意事項を次に示します。

- すでに存在するISAMによる索引編成ファイルに対してINPUT指定，I-O指定，EXTEND指定でOPEN文を実行する場合，実行時環境変数CBLISAMLARGEまたは実行時環境変数CBLD_ファイル名のISAMLARGEオプションに指定したファイルの形式と異なるとき，ファイルが存在しないエラー（KCCC4618R-S）となります。
  - ノーマルファイル形式のファイルに対してラージファイル形式を指定したとき。
  - ラージファイル形式のファイルに対してノーマルファイル形式を指定したとき。
- 実行時環境変数CBLD_ファイル名にISAMLARGEとNOISAMLARGE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。

- 実行時環境変数 CBLD_ファイル名と実行時環境変数 CBLISAMLARGE を同時に指定した場合は，CBLD_ファイル名が優先されます。
- 実行時環境変数 CBLD_ファイル名および実行時環境変数 CBLISAMLARGE の指定時に適用されるファイル形式を表 6-10 に示します。

表 6-10　実行時環境変数 CBLD_ファイル名および実行時環境変数 CBLISAMLARGE の指定時に適用されるファイル形式

実行時環境変数 CBLISAMLARGE	実行時環境変数 CBLD_ファイル名		
	ISAMLARGE	NOISAMLARGE	指定なし
YES	ラージファイル形式	ノーマルファイル形式	ラージファイル形式
YES 以外，または 指定なし	ラージファイル形式	ノーマルファイル形式	ノーマルファイル形式

- ノーマルファイル形式の索引編成ファイルから，ラージファイル形式の索引編成ファイルに切り替えるには，マニュアル「索引順編成ファイル管理 ISAM」の「ノーマルファイル形式からラージファイル形式への移行手順」を参照してください。

6.11.3　通常の入出力との互換性

ラージファイル入出力機能では，既存のファイルをラージファイルとして入出力することができます。ただし，ISAM による索引編成ファイルのラージファイル入出力機能では，すでにノーマルファイル形式のファイルが存在する場合（拡張子が.kdf，.k01～.k99，または.drf のファイル）は OPEN 文でエラー（KCCC4618R-S）となります。

6.11.4　ラージファイル入出力機能でのファイルの共用

Windows(x86) COBOL2002 の場合

次のファイルの場合，ラージファイルでもファイルサイズが 2GB 未満のファイルと同様にファイル共用を使用できます。

- 順編成ファイル
- ISAM による索引編成ファイル

Windows(x64) COBOL2002 の場合

順編成ファイルの場合，ラージファイルでも通常のファイル（ファイルサイズが 2GB 未満のファイル）と同様にファイル共用を使用できます。

6.11.5　ラージファイル入出力機能の制限事項

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。

ISAM による索引編成ファイルでは，ラージファイル形式の索引編成ファイルをリモートアクセス機能で使用できません。

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。また，マルチスレッド環境でのファイル共用（ファイルシェア）機能は，マルチスレッド対応 COBOL プログラムに対応します。

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	○	○	順ファイルの場合，ラージファイルであってもファイル共用を使用できる。
相対ファイル	○	○	
ISAM による索引編成ファイル	○	○	
Btrieve による索引編成ファイル※	○	○	詳細は「7.2.6　Btrieve (Pervasive.SQL) による索引編成ファイル（Windows(x86) COBOL2002 で有効)」を参照。
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

注※

Windows(x86) COBOL2002 だけで使用できます。

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

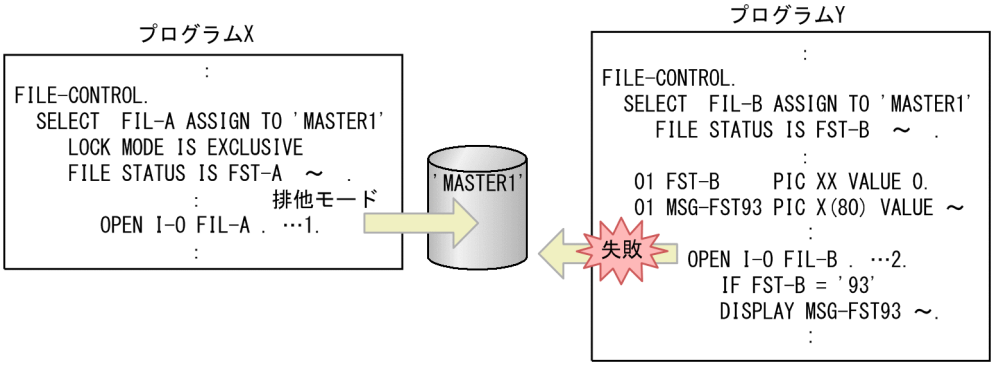
排他モードのファイルは，一つの実行単位に対してだけ開かれています。ほかの実行単位がこのファイルを使用しようとしても，ファイル施錠のエラーが返り，使用が拒否されます。

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。

(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

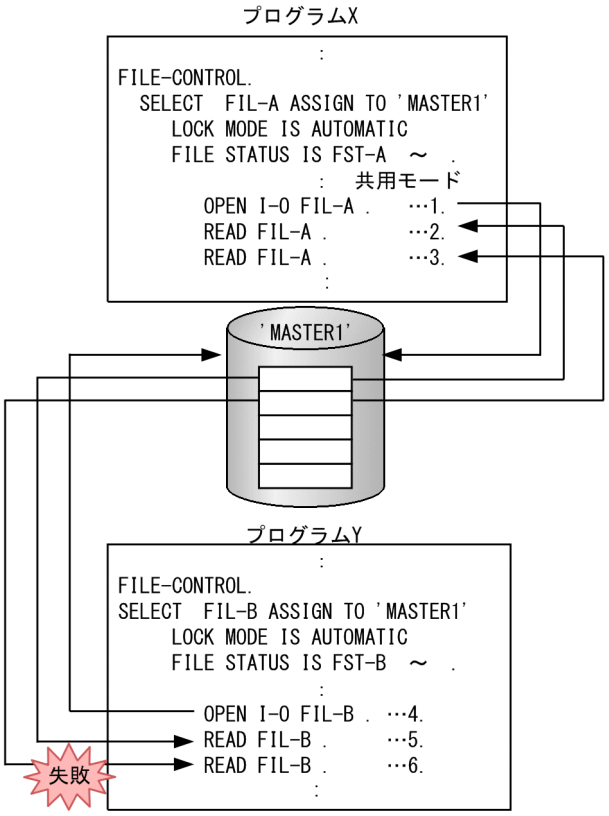
単一レコード施錠を指定した実行単位は，一つのファイル内に一つだけのレコード施錠を取得できます。

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。相対ファイルの場合，施錠できるレコード数に限りがあり，このレコード数は環境変数 CBL_RECLOCKMAX で指定します。

形式

```
CBL_RECLOCKMAX=レコード数
```

- 「レコード数」には施錠したいレコード数を指定します。指定できる値は 0～999,999,999 です。なお，指定した値に比例してメモリが消費されますので，実レコード数に見合った値を指定してください。
- この環境変数を指定していない場合，および「0」を指定した場合は，「50」が仮定されます。
- この環境変数に指定した値よりも多くのレコード施錠を要求した場合は実行時エラーとなります。

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

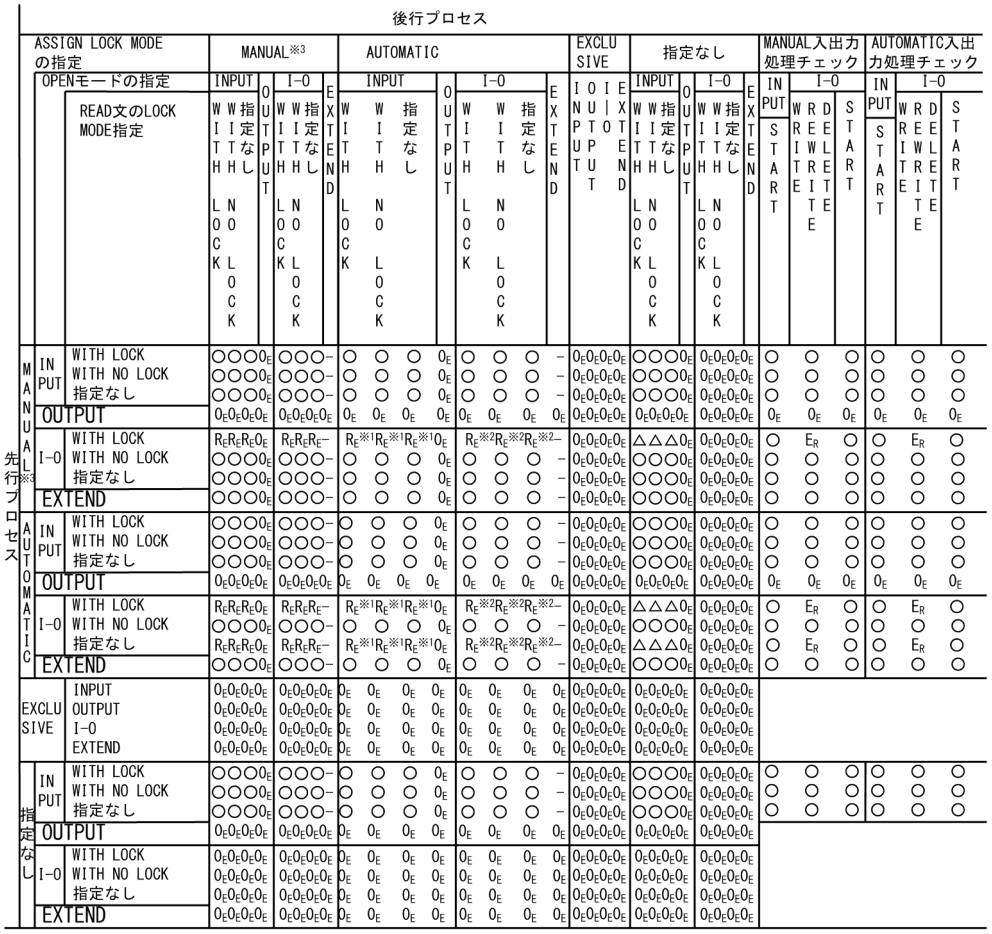
注※

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の場合の各実行単位の施錠形式を次に示します。



（凡例）

○：指定できる　　E_R：入出力エラー　　O_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※1

索引ファイルでは，ダーティリード機能，およびレコード占有解除待ち機能が使用できます。ダーティリード機能，およびレコード占有解除待ち機能については，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。

注※2

索引ファイルでは，レコード占有解除待ち機能が使用できます。レコード占有解除待ち機能については，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。

注※3

順ファイルでは，ASSIGN 句に LOCK MODE MANUAL の指定はできません。順ファイルを対象外とします。

(2)　Btrieve（Pervasive.SQL）による索引編成ファイルの場合

Btrieve（Pervasive.SQL）による索引編成ファイルの場合の各実行単位の施錠形式を次に示します。

先行プロセス			後行プロセス: 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			OUTPUT	I-O			INPUT			OUTPUT	I-O			INPUT			OUTPUT	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	NO LOCK	指定なし	EXTEND	WITH LOCK	NO LOCK	指定なし		WITH LOCK	NO LOCK	指定なし	EXTEND	IO INPUT EXTEND	WITH LOCK	NO LOCK	指定なし		WITH LOCK	NO LOCK	指定なし	EXTEND		START	WRITE	REWRITE	DELETE	START
MANUAL	INPUT	WITH LOCK		○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
		WITH NO LOCK		○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
		指定なし		○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
	OUTPUT			$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	0_E	0_E	0_E	0_E	0_E	0_E	0_E	0_E	0_E
	I-O	WITH LOCK		○	○	○	0_E	R_E	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	E_R	○	○	E_R	○	○	E_R	○
		WITH NO LOCK		○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
		指定なし		○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
	EXTEND			○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
AUTOMATIC	INPUT	WITH LOCK		○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
		WITH NO LOCK		○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
		指定なし		○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
	OUTPUT			$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	0_E	0_E	0_E	0_E	0_E	0_E	0_E	0_E	0_E
	I-O	WITH LOCK		○	○	○	0_E	R_E	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	E_R	○	○	E_R	○	○	E_R	○
		WITH NO LOCK		○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
		指定なし		○	○	○	0_E	R_E	○	−	0_E	R_E	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	E_R	○	○	E_R	○	○	E_R	○
	EXTEND			○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
EXCLUSIVE	INPUT			$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$									
	OUTPUT			$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$									
	I-O			$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$									
	EXTEND			$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$									
指定なし	INPUT	WITH LOCK		○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
		WITH NO LOCK		○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
		指定なし		○	○	○	0_E	○	○	−	0_E	○	○	−	0_E	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	○	○	○	○	○	○	○	○	○
	OUTPUT			$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$									
	I-O	WITH LOCK		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$									
		WITH NO LOCK		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$									
		指定なし		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$									
	EXTEND			$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$		$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$	$0_E0_E0_E0_E$									

（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）
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

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　Btrieve（Pervasive.SQL）による索引編成ファイル（Windows(x86) COBOL2002 で有効）

Btrieve（Pervasive.SQL）による索引編成ファイルでファイル共用を使用する場合の注意事項を，次に示します。

- 次の場合，ファイル排他またはレコード施錠の検知はしません。
  1. 先行プロセスで排他モードで開いているファイルを，後行プロセスで共用モードで開こうとした場合。
  2. 先行プロセスで施錠を掛けているレコードに，後行プロセスで施錠のない READ 文を実行した場合。
- 先行プロセスで排他モードで開かれているファイルに対し，後行プロセスでレコード施錠をした場合，ファイル施錠のエラーが返ります。この場合，ファイル位置指示子は保証します。
- 参照キーを更新した場合，あとに続く READ 文 NEXT 指定または READ 文 PREVIOUS 指定は，更新されたキーで次のレコードを入力します。
- 順アクセスで入力したレコードを削除した場合，ファイル位置指示子が不定となり，あとに続く READ 文 NEXT 指定または READ 文 PREVIOUS 指定は，実行時エラーとなります。
- ほかで使用中のファイルに対し，OUTPUT 指定の OPEN 文を実行した場合，ファイル排他は検知できず，オープンエラーとなります。

7.2.7　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，「6.11.4　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。

7.3　ファイル共用の注意事項

書き込み権限のないフォルダ下にある順編成ファイル，相対編成ファイルを複数の実行単位から INPUT モードで開く場合，ファイル共用機能を使用してもファイル施錠のエラーを検出できません。

ファイル施錠のエラーを検出したい場合は，フォルダに対して書き込み権限を付与するか，書き込み権限のあるフォルダにファイルを移動してください。

8 プリンタへのアクセス

COBOL プログラムでは，GDI モード印刷機能，ESC/P モード印刷機能，または 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 3 種類の方法があります。

GDI モード印刷

GDI モード印刷は，Windows 固有の印刷モードです。

GDI モード印刷では，出力先のプリンタ種別を意識しないでプリンタ出力できます。また，次のような印刷制御ができます。

- CHARACTER TYPE 句を指定して，行データの印刷を制御できます。これによって，行の途中で文字サイズや文字間隔を変えるなど，細かな印刷制御ができます。
- プリンタや印刷するファイル単位に印刷書式を作成できます。印刷書式ごとに 1 ページ内の行数や文字フォントサイズを指定できます。

ESC/P モード印刷

ESC/P モード印刷は，プリンタ固有の印刷モードでプリンタに出力するモードです。CHARACTER TYPE 句での行データの印刷制御はできませんが，エスケープシーケンス文字列という機能コードを使用することで，文字の打ち出し位置，文字間隔，行間隔などを細かく操作できます。このため，定型帳票への印刷などで，ほかの印刷モードで対処できない場合に有効です。

ただし，ESC/P モード印刷を使用した場合，エスケープシーケンス文字列を意識した出力をする必要があるため，プログラムの移植性が損なわれることがあります。

XMAP3 による印刷（Windows(x86) COBOL2002 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
GDI モード印刷	○	○
ESC/P モード印刷	○	○
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

すべての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　GDI モード印刷」，「8.5 ESC/P モード印刷」，および「8.6　XMAP3 による印刷（Windows(x86) COBOL2002 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	印刷モード		XMAP3 による印刷
	GDI モード印刷	ESC/P モード印刷	
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。	プリンタ出力処理の準備をする。OUTPUT モードで実行する。	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してプリンタに書き出す。	印刷データを一時ファイルに蓄積する。	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。	蓄積された一時ファイルの印刷データをプリンタに出力する。	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。

- SAMBADV／NOSAMBADV

  SAMBADV は，ADVANCING 指定を記述しない WRITE 文で自動的に BEFORE ADVANCING 1 LINE が仮定されます。NOSAMBADV は，仮定しません。標準値は NOSAMBADV です。この機能は，MIOS7 COBOL85 互換機能です。

- SAMPADV／NOSAMPADV

  SAMPADV は，OPEN 文の実行後，最初に実行される AFTER ADVANCING PAGE 指定の WRITE 文について行制御を有効にします。NOSAMPADV は，行制御を無効にします。省略時は SAMPADV が仮定されます。

  このオプション設定時には，次の注意が必要です。

  - SAMPADV および NOSAMPADV は，順編成ファイル，テキスト編成ファイルの印刷機能，書式印刷機能，および報告書作成機能※に対して有効となります。
  - LINAGE 句が指定されている場合，SAMPADV および NOSAMPADV の指定は無効となります。

  注※

  報告書作成機能の GENERATE 文は，内部的に WRITE 文を使用するため，このオプションが有効になります。

8.2.5　印刷文書名称

COBOL2002 の印刷機能では，印刷出力結果を一度スプールに登録します。スプールに登録する際の印刷文書名称は，COBOL2002 が独自に定めるデフォルトの印刷文書名称と，ユーザが指定した印刷文書名称の 2 種類があります。ここでは，それぞれの文書名称について説明します。

(1)　デフォルトの印刷文書名称

COBOL2002 の印刷機能では，デフォルトの印刷文書名称として，次の規則に基づいて印刷文書名称を作成します。

形式

```
実行ファイル名△△△FNAM-ファイル名
```

実行ファイル名

印刷をする実行可能ファイルの名称です。絶対パスでの名称指定の場合でもパス名は表示されず，拡張子.exe が削除された実行可能ファイル名称が表示されます。

ファイル名

COBOL プログラム中の SELECT 句で指定したファイル名称です。

△

1 バイトの空白を示します。

表示される印刷文書名称の例を，次に示します。

（例）実行可能ファイル名：SAMPLE.exe

```
      :
ENVIRONMENT DIVISION.
      :
SELECT SAM-FILE ASSIGN TO SYS001
      :
```

表示される印刷文書名称

SAMPLE△△△FNAM-SAM-FILE

(2) ユーザ指定の印刷文書名称

スプールに登録される印刷文書名称は，次のどちらかの方法で変更できます。

- COBOL プログラム中に印刷文書名称を指定する方法
- 実行時環境変数で印刷文書名称を指定する方法

(a) COBOL プログラム中に印刷文書名称を指定する方法（APPLY FORMS-NAME 句）

環境部の APPLY FORMS-NAME 句に，スプールに登録する印刷文書名称を設定します。

APPLY FORMS-NAME 句については，マニュアル「COBOL2002 言語 拡張仕様編」「25.2.3(2) APPLY FORMS-NAME 句」を参照してください。

（例）

```
      :
ENVIRONMENT DIVISION.
      :
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT A-FILE ASSIGN TO SYS000.
      :
I-O-CONTROL.
APPLY FORMS-NAME TO DOCNAME ON A-FILE.
      :
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DOCNAME PIC X(9) VALUE 'FORMNAME1'.
      :
```

表示される印刷文書名称

FORMNAME1

(3) 実行時環境変数で印刷文書名称を指定する方法（環境変数 CBLF_ファイル名）

実行時環境変数 CBLF_ファイル名を使用して，スプールに登録される印刷文書名称を指定できます。

形式

```
CBLF_ファイル名=印刷文書名称
```

ファイル名

SELECT 句で指定したファイル名と対応します。指定したファイル名の中にハイフン（-）があった場合は，下線（_）に置き換えた名称となります。

印刷文書名称

スプールに登録したい印刷文書名称を，任意の文字列で指定します。

(4) 注意事項

- 印刷文書名称の変更は，GDI モード印刷，および ESC/P モード印刷の場合だけ適用されます。
- システムが認識できない形式の印刷文書名称を指定した場合，印刷開始時にエラーとなります。
- APPLY FORMS-NAME 句と環境変数 CBLF_ファイル名の両方を指定した場合，CBLF_ファイル名に指定した印刷文書名称が有効となります。
- 印刷文書名称には，1,024 バイト以内の文字列を指定してください。また，印刷文書名称には英数字，または日本語の文字列を指定してください。ただし，印刷文書名称として指定できる長さと文字列は使用するプリンタによって異なります。

  なお，印刷文書名称に指定された文字列の途中に NULL（X'00'）が含まれている場合，NULL 以降の文字列は出力されません。

8.2.6 外字の有効化

Windows では，外字登録はログオンユーザごとに実行されます。しかし，Windows サービスプログラムを実行するセッション 0 のように，Windows へログオンしないでプログラムを実行する環境では，ユーザプロファイルが読み込まれないため外字が有効になりません。ただし，次のどちらかの場合，プログラムから Windows API の EnableEUDC 関数を発行することで外字を有効にできます。

- プログラムにユーザプロファイルを読み込む機能がある。
- プログラムを実行するユーザで，Windows に一度でもログオンしている。

COBOL2002 の GDI モード印刷および ESC/P モード印刷では，OPEN 文で内部的に EnableEUDC 関数を発行するため，上記の環境で COBOL プログラムを実行する場合，外字が有効になります。
EnableEUDC 関数の処理は，環境変数 CBLEUDCFUNC の指定で変更できます。

環境変数 CBLEUDCFUNC を指定すると，GDI モード印刷および ESC/P モード印刷の OPEN 文で内部的に発行する EnableEUDC 関数の処理を変更できます。

形式

```
CBLEUDCFUNC={ ERRORSTOP | NOFUNC }
```

ERRORSTOP

EnableEUDC 関数がエラーを返したときに重大エラーのメッセージを出力し，FILE STATUS 句や USE 手続きがなければプログラムは終了します。FILE STATUS 句の指定がある場合，入出力状態の値には 90 を設定します。

NOFUNC

EnableEUDC 関数は呼び出しません。

環境変数 CBLEUDCFUNC の指定がない場合，および指定値に誤りがある場合は，EnableEUDC 関数がエラーを返したときに警告メッセージを出力して，プログラムを継続します。このとき，FILE STATUS 句があってもメッセージを出力し，入出力状態の値は 00 になります。

注意事項

- 環境変数 CBLEUDCFUNC の指定がない環境で警告メッセージが出力された場合，外字が正しく印刷されないことがあります。この警告メッセージが出力された場合，印刷結果を確認し，外字が正しく印刷されていないときはプログラムを再実行してください。
- NOFUNC を指定する場合は，外字を使用していないこと，または OPEN 文で EnableEUDC 関数を呼び出さなくても外字が有効な環境であることを確認してください。

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　GDI モード印刷

GDI モードは，Windows 固有の印刷モードです。GDI モードを使用すると，出力先のデバイスを意識しないでプリンタ出力ができます。

GDI モード印刷では，順編成ファイル，テキスト編成ファイルで行制御出力する行データを，直接プリンタに出力できます。また，行データが印刷制御付きでも出力できます。

8.4.1　プリンタへの出力割り当て方法

GDI モード印刷を使用する場合は，環境部のファイル管理記述項でファイルを指定したあと，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'PRINTER〔：印刷書式番号〕'という名称の物理ファイルを割り当てます。PRINTER は小文字でもかまいません。パス付きで指定する場合でも，最右側の¥の次が PRINTER〔：印刷書式番号〕であればかまいません。

印刷書式番号の意味については，「(2)　複数の印刷書式を使用したプリンタ出力機能」を参照してください。

(1)　ファイルの割り当て

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。

定数指定

```
       ENVIRONMENT  DIVISION.
       INPUT-OUTPUT  SECTION.
       FILE-CONTROL.
       SELECT FILE-1 ASSIGN TO 'PRINTER'.
           :
```

環境変数指定

```
       ENVIRONMENT  DIVISION.
       INPUT-OUTPUT  SECTION.
       FILE-CONTROL.
       SELECT FILE-1 ASSIGN TO PRT.
           :
```

注

このとき，次の環境変数が指定されているものとします。

```
CBL_PRT=PRINTER
```

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 'PRINTER' TO FILE-NAME.
    :
```

(2)　複数の印刷書式を使用したプリンタ出力機能

物理ファイルにプリンタを割り当てた場合，帳票の印刷書式（出力先，出力形式）をファイルごとに指定できます。複数の印刷書式を設定しておくことで，プログラム中で使用するファイルごとに異なる印刷書式で帳票を印刷できます。設定した印刷書式は印刷書式番号（1～99）で識別します。

(a)　印刷書式の設定

各印刷書式番号に対応した印刷書式を設定するには，実行支援ウィンドウの［設定］メニューの［印刷書式］コマンドを使用します。詳細については，マニュアル「COBOL2002　操作ガイド」を参照してください。

(b)　物理ファイルの割り当て方法

定数指定，環境変数指定，データ名指定のどの場合でも，物理ファイル名に次の指定をすることで，印刷書式番号に対応した印刷書式で帳票を出力できます。

```
PRINTER 〔:印刷書式番号〕
```

また，印刷書式にはデフォルトの印刷書式を設定しておくこともできます。この場合，単に"PRINTER"だけを指定すると，デフォルトの印刷書式で印刷されます。

(c)　物理ファイルの動的割り当て

ASSIGN 句で処理系作成者語を指定した場合（環境変数指定の場合），GUI モードでの OPEN 文実行時に，印刷書式を動的に設定できます。

OPEN 文実行時に，該当するファイルに対して物理ファイルが割り当てられていない場合，物理ファイルの割り当て画面が表示されます（「6.2.4　実行時の動的割り当て（GUI モードの場合だけ）」を参照）。この画面に次のどれかの指定をして，［OK］ボタンを選びます。

PRINTER:印刷書式番号

指定した印刷書式番号に対応した印刷書式で印刷されます。

PRINTER:

印刷定義ダイアログボックスが表示されます。ここで，印刷書式を確認しながら印刷書式番号を指定できます。ダイアログボックスの形式などについては，マニュアル「COBOL2002　操作ガイド」の実行支援を使った実行環境の設定の説明を参照してください。

PRINTER

デフォルトの印刷書式で印刷されます。

(d) 使用例

物理ファイル名称の定数指定を使用して印刷書式番号を設定する例を，次に示します。

```
      :
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT FILE-1 ASSIGN TO 'PRINTER:1'.
      :
```

8.4.2 出力形態とレコード形式

GDI モード印刷では，出力するデータ項目に CHARACTER TYPE 句を指定することで，印刷制御付きの行データを指定できます。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。

WRITE 文と CHARACTER TYPE 句との関係を次に示します。

(1) WRITE 文とレコード名の関係

(a) WRITE レコード名の場合

レコード名に CHARACTER TYPE 句がある場合だけ，行データを印刷制御します。

記述例

```
FD A-FILE.
01 A-REC.
  02 A-REC-1 PIC N(10) CHARACTER TYPE IS POINT-7 WIDE.
  02 A-REC-2 PIC N(10) CHARACTER TYPE IS POINT-9 FORMAT-7.
    :
PROCEDURE DIVISION.
    :
    WRITE A-REC AFTER 1 LINE.
    :
```

(b) WRITE レコード名 FROM 一意名の場合

一意名に CHARACTER TYPE 句がある場合だけ，行データを印刷制御します。この場合，レコード名に CHARACTER TYPE 句があっても，一意名の CHARACTER TYPE 句の情報が有効となります。

記述例

```
FD A-FILE.
01 A-REC PIC N(80).
    :
WORKING-STORAGE SECTION.
01 DATA1.
  02 A-REC-1 PIC N(10) CHARACTER TYPE IS POINT-7 WIDE.
```

```
  02 A-REC-2 PIC N(10) CHARACTER TYPE IS POINT-9 FORMAT-7.
    :
PROCEDURE DIVISION.
    :
    WRITE A-REC FROM DATA1 AFTER 1 LINE.
    :
```

(2) CHARACTER TYPE 句指定での印刷

(a) POINT-*l* 指定

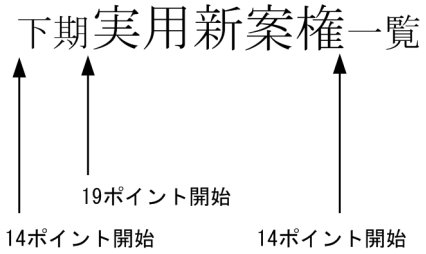
POINT-*l* を書くと，文字サイズ（ポイント）を指定できます。*l* の値はポイント数を示します。

なお，Windows COBOL2002 では，POINT-*l* を省略した場合のポイントは，実行支援の［字サイズ］で指定した値となります。

記述例

```
    :
01 A.
  02 A0 CHARACTER TYPE IS POINT-14.
    03 A1 PIC NN VALUE N'下期'.
    03 A2 PIC N(5) POINT-19
                VALUE N'実用新案権'.
    03 A3 PIC NN VALUE N'一覧'.
   :
PROCEDURE DIVISION.
      :
    WRITE A-REC FROM A AFTER 1 LINE.
      :
```

印刷結果



(b) FORMAT-*n* 指定

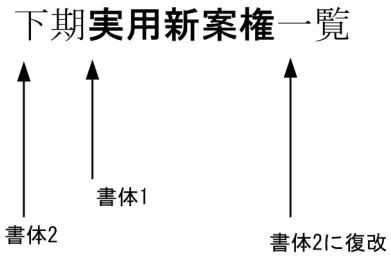
FORMAT-*n* を書くと，書体を指定できます。書体は，あらかじめ実行支援の［書体］で設定しておく必要があります。

なお，Windows COBOL2002 では，FORMAT-*n* を省略した場合の書体は，実行支援の［書体］のデフォルトで指定したフォントとなります。

記述例

```
      :
01 A.
  02 A0 CHARACTER TYPE IS FORMAT-2.
    03 A1 PIC NN VALUE N'下期'.
    03 A2 PIC N(5) FORMAT-1
                 VALUE N'実用新案権'.
    03 A3 PIC NN VALUE N'一覧'.
  :
PROCEDURE DIVISION.
      :
    WRITE A-REC FROM A AFTER 1 LINE.
      :
```

印刷結果



(c) INTERVAL-*i* 指定

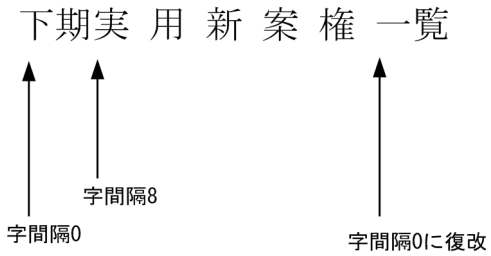
INTERVAL-*i* を書くと，各文字間隔を指定できます。INTERVAL-*i* を省略した場合の文字間隔は，0 ポイントとなります。

なお，Windows COBOL2002 では，実行支援の［半角文字と全角文字の間隔を調整する］チェックボックスをチェックしない場合は，全角文字と半角文字の字間隔は *i*/2 ポイントとなります。このチェックボックスをチェックした場合，全角文字の字間隔は *i* ポイント，半角文字の字間隔は *i*/2 ポイントとなります。

記述例

```
      :
01 A.
  02 A0 CHARACTER TYPE IS INTERVAL-0.
    03 A1 PIC NN VALUE N'下期'.
    03 A2 PIC N(5) INTERVAL-8
                 VALUE N'実用新案権'.
    03 A3 PIC NN VALUE N'一覧'.
  :
PROCEDURE DIVISION.
      :
    WRITE A-REC FROM A AFTER 1 LINE.
      :
```

印刷結果



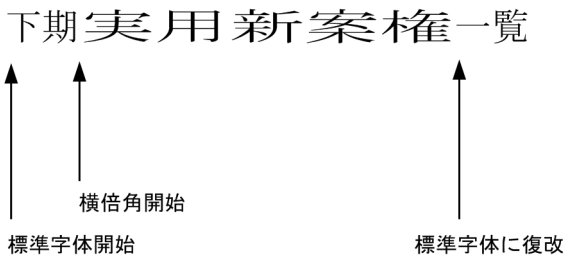
(d) WIDE 指定

WIDE を書くと，横倍角を指定できます。

記述例

```
      :
    01 A.
      02 A0.
        03 A1 PIC NN VALUE N'下期'.
        03 A2 PIC N(5) WIDE
                     VALUE N'実用新案権'.
        03 A3 PIC NN VALUE N'一覧'.
      :
    PROCEDURE DIVISION.
          :
        WRITE A-REC FROM A AFTER 1 LINE.
          :
```

印刷結果



(e) 環境変数 CBLGDIINTERVAL を使用した出力例

環境変数 CBLGDIINTERVAL に YES を指定すると，CHARACTER TYPE IS INTERVAL 句が出現しない間の項目に対して，実行支援の印刷書式に設定した字間隔を使用します。ただし，CHARACTER TYPE 句の指定がまったくないレコードに対しては，環境変数 CBLGDIINTERVAL の指定と関係なく，実行支援の印刷書式に従った字間隔が使用されます。

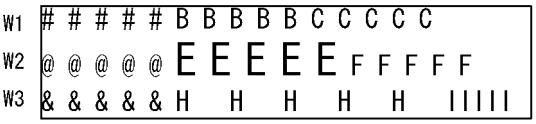
環境変数 CBLGDIINTERVAL を使用した出力例を，次に示します。

レコード定義

```
    01 W1.
      02 A PIC X(5) VALUE ALL '#'.
      02 B PIC X(5) VALUE ALL 'B'.
```

```
        02 C PIC X(5) VALUE ALL 'C'.
      01 W2.
        02 D PIC X(5) VALUE ALL '@'.
        02 E PIC X(5) VALUE ALL 'E'
                      CHARACTER TYPE IS POINT-16.
        02 F PIC X(5) VALUE ALL 'F'.
      01 W3.
        02 G PIC X(5) VALUE ALL '&'.
        02 H PIC X(5) VALUE ALL 'H'
                      CHARACTER TYPE IS INTERVAL-24.
        02 I PIC X(5) VALUE ALL 'I'.
```

印刷結果（CBLGDIINTERVAL に YES を指定した場合）



W1 および W2

実行支援の印刷書式の設定に従った字間隔※で出力される

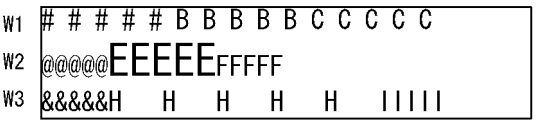
W3

- INTERVAL 指定が出現するまでは，実行支援の印刷書式の設定に従った字間隔※で出力される
- INTERVAL 指定があるデータ項目は，INTERVAL 指定で指定された字間隔で出力される
- INTERVAL 指定以降のデータ項目は，字間隔 0 で出力される

注※

実行支援の印刷書式の字間隔を 10cpi に設定した場合

印刷結果（CBLGDIINTERVAL に YES 以外の値を指定した場合，または CBLGDIINTERVAL に値を指定しなかった場合）



W1

実行支援の印刷書式に従った字間隔※で出力される

W2

CHARACTER TYPE 句の指定があり，かつ環境変数 CBLGDIINTERVAL に YES が指定されていないため，字間隔 0 で出力される

W3

- INTERVAL 指定がないデータ項目は，W2 と同様に出力される
- INTERVAL 指定があるデータ項目は，INTERVAL 指定で指定された字間隔で出力される

注※

実行支援の印刷書式の字間隔を 10cpi に設定した場合

8.4.3　印刷書式の設定

印刷書式の設定には，実行支援を使用します。

実行支援の詳細については，マニュアル「COBOL2002　操作ガイド」を参照してください。

8.4.4　外字の出力方法

GDI モード印刷を使用する場合，Windows 上の MS-IME などのかな漢字変換システムの辞書に外字を登録すると，外字を印刷できます。

8.4.5　注意事項

- プリンタを I-O モードでオープンし，WRITE 文を使用した場合の結果は保証しません。
- 行データの重ね打ちはできません。
- WRITE 文に ADVANCING／POSITIONING 指定がない場合，改行などの行制御はされません。
- WRITE 文に AFTER ADVANCING PAGE 指定があっても，プリンタをオープンして最初に実行する場合は，印刷データがない状態のため，改ページは無視して印刷されます。
- WRITE 文の ADVANCING で指定された行送りの結果，印刷行の位置が実行支援などで設定された「1 ページの印刷行数」を超えたとき，改ページをして，印刷行数を「1 ページの印刷行数」で割った余りの位置まで行送りをします。このとき，行送りの行数が 2 ページを超えても，改ページは 1 回だけです。

  例）

  次の条件では，36LINES による改ページは 1 ページです（3 ページとはなりません）。

  1.の印刷のあとで，その次のページの 7 行目に 2.を印刷します。

  [条件]

  - 実行支援で，1 ページの印刷行数を「10 行」に指定する。
  - 次に示す WRITE 文を実行する。

    WRITE PRINT-PAGE AFTER ADVANCING PAGE.　　…1.

    WRITE PRINT-AREA AFTER ADVANCING 36 LINES.　　…2.
- 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。

8.5　ESC/P モード印刷

ESC/P モードは，プリンタ固有の出力モードです。ESC/P モードを使用すると，高速にプリンタ出力ができます。ただし，ESC/P モード印刷を使用するには，プリンタが ESC/P モードをサポートしている必要があります。

ESC/P モード印刷では，順編成ファイル，テキスト編成ファイルで機能コード（エスケープシーケンス）が入ったレコードを，直接プリンタに出力できます。機能コードは，文字の拡大や書体といったプリンタの各機能を制御するコードです。ただし，X'81'～X'9F'，X'E0'～X'FC'のコードを含む機能コードや外字は使用できません。

ESC/P モード印刷では，CHARACTER TYPE 句は有効となりません。ESC/P モード印刷では，機能コードを設定して印刷制御を行ってください。

8.5.1　プリンタへの出力と割り当て方法

環境部の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，物理ファイル名に'SYSPRT〔:ネットワークプリンタ名〕'を指定します。SYSPRT は小文字でもかまいません。パス付きで指定する場合でも，最右側の¥の次が'SYSPRT〔:ネットワークプリンタ名〕'であればかまいません。ネットワークプリンタ名の意味については，「(2)　ネットワークプリンタへの出力」を参照してください。

(1)　ファイルの割り当て

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。

定数指定

```
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FILE-1 ASSIGN TO 'SYSPRT'.
           :
```

環境変数指定

```
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FILE-1 ASSIGN TO PRT.
           :
```

注

このとき，次の環境変数が指定されているものとします。

```
CBL_PRT=SYSPRT
```

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 'SYSPRT' TO FILE-NAME.
    :
```

(2) ネットワークプリンタへの出力

ESC/P モード印刷では，物理ファイル名に次の指定をすることで，ネットワークプリンタに出力できます。あらかじめ複数のネットワークプリンタを設定しておけば，一つのプログラム中で使用する複数のファイルを，別々のプリンタで印刷できます。

```
SYSPRT:ネットワークプリンタ名
```

（例）

SERVER1 に接続されている PRINTER1 というプリンタに出力する場合

```
SYSPRT:¥¥SERVER1¥PRINTER1
```

なお，この指定方法は，定数指定，環境変数指定，データ名指定のどれでも使用できます。

8.5.2 出力形態とレコード形式

ESC/P モード印刷の場合，必要な機能コードを行データに設定して，プリンタへ出力します。

横倍角データを出力する場合のコーディング例を，次に示します。

記述例

```
   :
FD A-FILE.
01 S-REC PIC X(50).
   :
01 A-WK.
  02 A0.
    03 A1       PIC NN   VALUE N'下期'.
    03 CNTL-CD1 PIC X(3) VALUE X'1B5701'. …1.
    03 A2       PIC N(5) VALUE N'実用新案権'.
    03 CNTL-CD2 PIC X(3) VALUE X'1B5700'. …2.
    03 A3       PIC NN   VALUE N'一覧'.
   :
PROCEDURE DIVISION.
       :
```

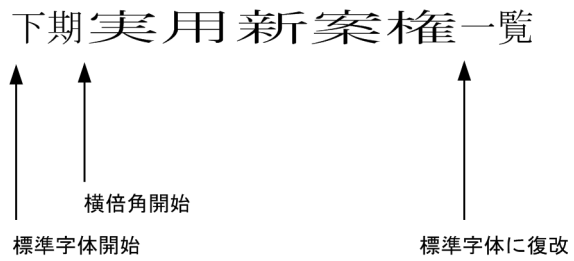
```
            WRITE S-REC FROM A-WK AFTER ADVANCING 1.
               :
```

1. 横倍角開始の機能コード※
2. 横倍角終了の機能コード※

注※

機能コードは，プリンタに依存します。ご利用のプリンタの機能コードは，プリンタのマニュアルでご確認ください。

印刷結果



・横倍角開始制御コード（プリンタに依存）
・横倍角終了制御コード（プリンタに依存）

(1) 注意事項

ESC/P モードでプリンタへ出力する場合の注意事項を次に示します。

- ESC/P モードでは，機能コードのサポート範囲がプリンタごとに異なります。サポートする機能コードの詳細については，ご利用のプリンタのマニュアルを参照してください。
- ESC/P モードプリンタへ直接出力する場合，COBOL プログラムから出力された印刷データは，内部的に一時ファイルに格納され，CLOSE 文の実行後に一時ファイルの内容がプリンタへ出力されます。このため，WRITE 文が実行されてから実際に印刷されるまで，時間がかかる場合があります。
- 外部装置名に「CBL_PRT=¥¥host¥printer」のような UNC 形式は使用できません。

8.5.3 外字の出力方法

- ESC/P モード印刷では，Windows 上で作成できるユーザ定義文字を外字として扱います。また，次に示すベンダ定義文字（ベンダが独自に追加した文字）も外字として印刷できます。
- Windows 特殊文字
- NEC 選定 IBM 拡張文字
- IBM 拡張文字

(1) ユーザ定義文字の外字出力

ここでは，Windows 上で作成した外字を ESC/P モード印刷機能でプリンタへ印刷する方法を説明します。

環境変数 CBLP_ファイル名を設定すると，あらかじめプリンタに登録しなくても，Windows 上で作成した外字を出力できます。なお，環境変数 CBLP_ファイル名は，CLOSE 文を実行するごとに環境変数の値が参照されます。

(a) 形式

```
CBLP_ファイル名=プリンタの外字コード〔,タイプフェイス名〕
```

ファイル名

SELECT 句で指定したファイル名と対応させます。指定したファイル名の中にハイフン（-）があった場合には，下線（_）に置き換えた名称となります。

プリンタの外字コード

実行時に一時的に使用する，プリンタの外字領域のコードを指定します。プリンタの外字コードは，4けたの 16 進数で指定します。

タイプフェイス名

出力したい外字が登録されているフォントの，タイプフェイス名を指定します。指定を省略した場合は，「ＭＳ 明朝」が仮定されます。

(b) 出力の手順と例

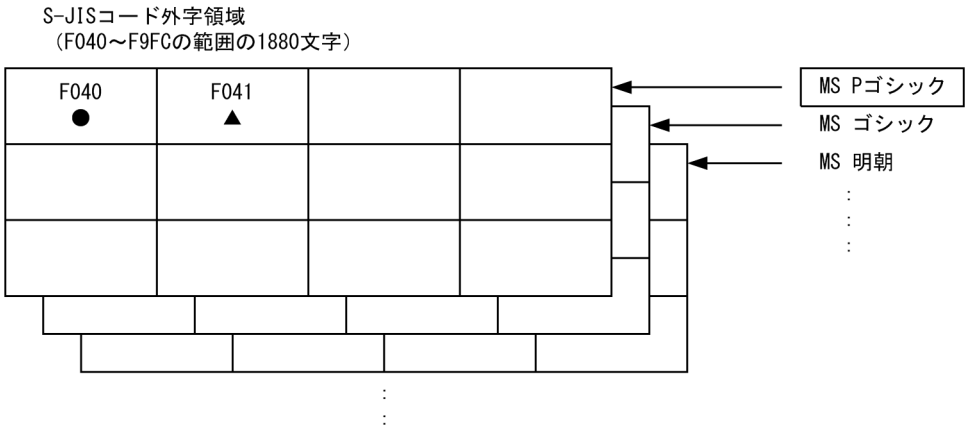
1. Windows 上での外字の作成・登録

Windows の外字エディタなどを使って，外字を作成します。Windows 上で作成した外字は，次のようなイメージで外字領域に登録されます。

（例）

タイプフェイス「ＭＳ Ｐゴシック」に関連づけて，コード X'F040'に「●」，コード X'F041'に「▲」という外字を登録した場合

図 8-1 Windows 上で作成した外字の登録イメージ



2. 作業領域にするプリンタの外字コードの決定

プリンタの外字領域の中で，一時的に外字を登録するコードを決めます。プリンタの外字領域と作業領域にする外字コードとの関係は，次のようになります。

（例）

プリンタの外字コード X'7721'を作業領域にする場合

図 8-2　プリンタの外字領域と作業領域にする外字コードとの関係

3. 環境変数の設定

環境変数 CBLP_ファイル名を使って，Windows の外字コード，プリンタの外字コード，および出力ファイルを関連づけます。

（例）上記(a)(b)の例の設定で，FILE01 を出力する場合

```
CBLP_FILE01=7721,ＭＳ△Ｐゴシック
```

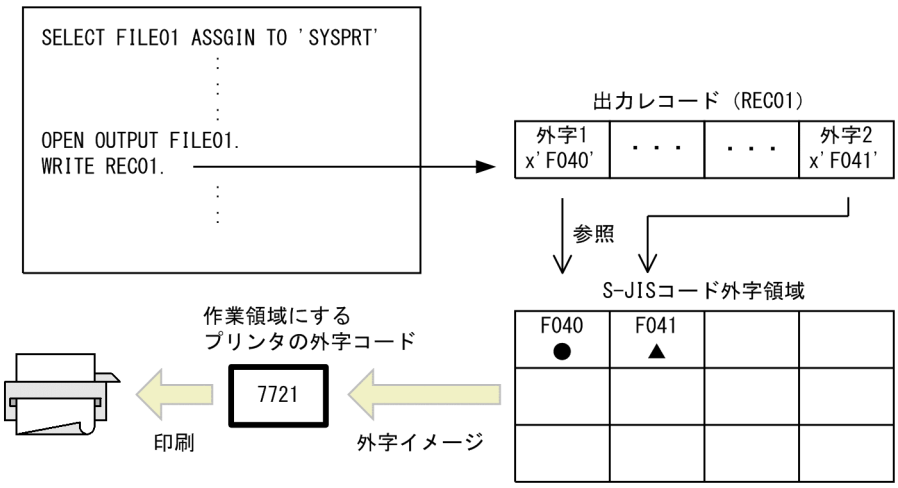
注

△は半角空白文字を表します。

4. プログラムの実行

プログラムを実行すると，次のようなイメージで，Windows 上で作成した外字がプリンタに出力されます。

図 8-3　作業用のプリンタ外字コードを使用した外字印刷の流れ



注意事項

ユーザ定義文字を外字として出力する場合の注意事項を，次に示します。

- プリンタの外字コードに，出力先となるプリンタの外字領域コード以外を指定した場合，結果は保証しません。
- プリンタの外字コードに 4 けたの 16 進数以外の値を指定した場合，エラーとなります。
- Windows で作成した外字は，関連づけられるタイプフェイスに登録してください。タイプフェイスに対して外字登録できないようなタイプフェイス名を指定した場合，外字の出力結果は保証しません。
- タイプフェイス名に指定する文字列は，半角や全角の区別も含めて正しく指定してください。指定したタイプフェイス名が認識できなかった場合，エラーとなります。
- タイプフェイス名を省略した場合，「ＭＳ 明朝」が仮定されます。ただし，「ＭＳ 明朝」のフォントがプログラムの実行する環境にない場合，エラーとなります。
- あらかじめ，プリンタの外字領域に外字が登録されていて，その領域をプリンタの外字コードとして指定した場合，その登録内容は保証しません。
- 印刷時に，プリンタの仕様に合わせて外字のサイズが変更されます。そのため，外字のビット情報が一部欠落する場合があります。

(2) ベンダ定義文字の外字出力

ここでは，ベンダ定義文字を外字として印刷する方法を説明します。

環境変数 CBLPRTEXCHR を設定すると，あらかじめプリンタにベンダ定義文字を登録しなくても，ベンダ定義文字を出力できます。

形式

```
CBLPRTEXCHR=ベンダ定義文字種別〔:ベンダ定義文字種別…〕
```

ベンダ定義文字種別

外字として印刷するベンダ定義文字の種別を指定します。複数のベンダ定義文字種別を指定する場合は，コロン（:）で区切ります。

環境変数 CBLPRTEXCHR に指定するベンダ定義文字の種別を次に示します。

表 8-2 環境変数 CBLPRTEXCHR に指定するベンダ定義文字の種別

環境変数に指定する文字列	種別	コード範囲	
		上位バイト	下位バイト
BND1	Windows 特殊文字	$(87)_{16}$	$(40)_{16}$～$(FC)_{16}$※
BND2	NEC 選定 IBM 拡張文字	$(ED)_{16}$～$(EE)_{16}$	$(40)_{16}$～$(FC)_{16}$※
BND3	IBM 拡張文字	$(FA)_{16}$～$(FC)_{16}$	$(40)_{16}$～$(FC)_{16}$※

注※ $(7F)_{16}$ を除きます。

注意事項

- 環境変数 CBLPRTEXCHR に上記以外の文字列を指定した場合，実行時エラーとなります。
- 環境変数 CBLPRTEXCHR は，環境変数 CBLP_ファイル名を指定しているときに有効となります。
- 環境変数 CBLPRTEXCHR で指定したベンダ定義文字に対して，コード範囲中にある文字コードはすべて外字として印刷します。未定義の文字コードを外字として印刷した場合，印刷結果は保証しません。
- 外字としてベンダ定義文字を印刷した場合の印刷結果は，同じベンダ定義文字を通常の 2 バイト文字として印刷した印刷結果に比べて，文字の一部が欠けたり，曲線が滑らかにならなくなったりすることがあります。

8.5.4　ESC/P モード印刷を利用したプリンタへの出力例

ESC/P モード印刷機能を利用したプリンタへの出力例を説明します。はじめに，ご利用のプリンタで ESC/P モード（AX モード）が使用できるか確認してください。

(1)　プリンタの設定

印刷前のプリンタの設定手順を示します。

1. プリンタを ESC/P モードで動作させる。
2. Windows のプリントマネージャで，ESC/P モードのプリンタを使えるように設定する。

また，ESC/P モードで詳細な帳票設計をする場合，プリンタの仕様をあらかじめ調べておく必要があります。詳細は，ご利用のプリンタのマニュアルを参照してください。

(2)　文字列の出力

文字列を出力する例題について説明します。

(a)　プリンタの仕様

この例題で想定するプリンタの仕様を次に示します。

表 8–3　プリンタの仕様

項目		値
フォント	ANK 文字 全角漢字文字 半角漢字文字	Roman 明朝 明朝
文字幅	ANK 文字 全角漢字文字	10CPI（Character Per Inch） 24 ドット

項目		値
	半角漢字文字	12 ドット
文字スペース量	ANK 文字 全角漢字文字 半角漢字文字	左：0 ドット，右：0 ドット 左：0 ドット，右：2 ドット 左：0 ドット，右：3 ドット

(b)　プログラミング例

文字列'HITACHI'を出力するプログラム例を次に示します。

文字列を出力するプログラミング例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
      ************************************************************
      *      文字列 'HITACHI'を出力する                          *
      ************************************************************
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       SELECT S-F1 ASSIGN TO 'SYSPRT'
                   ORGANIZATION IS LINE SEQUENTIAL.
       DATA DIVISION.
       FILE SECTION.
       FD S-F1.
       01 S-R1 PIC X(132).
       WORKING-STORAGE SECTION.
       01 T-HITACHI PIC X(7) VALUE 'HITACHI'.
       PROCEDURE DIVISION.
           OPEN OUTPUT S-F1.
      *
           MOVE T-HITACHI TO S-R1.
           WRITE S-R1 AFTER ADVANCING 1.
      *
           CLOSE S-F1.
           STOP RUN.
```

(3)　下線を設定した文字列の出力

(2)で説明したプログラム例に機能コードを設定することで，出力文字列に下線を設定できます。

(a)　プリンタの仕様

「表 8-3　プリンタの仕様」と同様の仕様です。

(b)　プログラミング例

文字列'HITACHI'に下線を設定して出力するプログラム例を次に示します。

印刷結果は'HITACHI'となります。

文字列に下線を設定して出力するプログラミング例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE2.
      ************************************************************
      *     文字列  'HITACHI'に下線を設定して出力する            *
      ************************************************************
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       SELECT S-F1 ASSIGN TO 'SYSPRT'
                   ORGANIZATION IS LINE SEQUENTIAL.
       DATA DIVISION.
       FILE SECTION.
       FD S-F1.
       01 S-R1          PIC X(132).
       WORKING-STORAGE SECTION.
       01 T-OUTDATA.
        02 T-SET-UNDER PIC X(3) VALUE X'1B2D01'.  …1.
        02 T-HITACHI   PIC X(7) VALUE 'HITACHI'.
        02 T-NON-UNDER PIC X(3) VALUE X'1B2D00'.  …2.
       PROCEDURE DIVISION.
           OPEN OUTPUT S-F1.
      *
           MOVE T-OUTDATA TO S-R1.
           WRITE S-R1 AFTER ADVANCING 1.
      *
           CLOSE S-F1.
           STOP RUN.
```

1. 下線設定の機能コード
2. 下線解除の機能コード

(4) 文字幅を整えた文字列の出力

ESC/P モードで，全角文字と半角文字が混在する帳票を出力する場合，全角文字と半角文字の文字幅が 1：1/2 で出力されないことがあります。例えば，同じバイト数の行でも，長さが異なる印刷結果となることがあります。これは，利用するプリンタの印刷文字の幅の仕様によるものです。

印刷文字の幅を整えるには，プログラム中で ESC/P の機能コードを出力レコードに埋め込みます。なお，文字幅と ESC/P の機能コードについては，ご利用のプリンタのマニュアルを参照してください。

(a) プリンタの仕様

この例題で想定するプリンタの仕様を次に示します。

項目		値
文字幅（180dpi）	ANK 文字	18 ドット（18/180 インチ）
	漢字全角文字	24 ドット（24/180 インチ）
	漢字半角文字	12 ドット（12/180 インチ）

(b) プログラミング例

漢字全角文字の文字幅と ANK 文字の文字幅の比率を 1：1/2 に調整します。データの出力後は文字幅を元に戻します。

プログラムで使用する引数を次に示します。

- PUT-DATA：出力データ名
- PUT-LEN：出力データ長

文字幅を調整するプログラム例を，次に示します。

文字幅を調整するプログラミング例

```
       WORKING-STORAGE SECTION.
       01 P          PIC 9(4).
       01 SI-CODE    PIC X(4) VALUE X'1C530606'. …1.
       01 EI-CODE    PIC X(4) VALUE X'1C530003'.
       01 ICODE-LEN  PIC 9    VALUE 4.
       LINKAGE SECTION.
       01 PUT-DATA   PIC X(80).
       01 PUT-LEN    PIC 9(04).
       PROCEDURE DIVISION USING PUT-DATA PUT-LEN.
           OPEN OUTPUT S-F1.
      *
       CODE-CONV SECTION.
      *
      * Pは，出力領域の転送先ポインタ
           MOVE 1         TO P.
           MOVE ALL SPACE TO S-R1.
      *
      * 漢字全角文字の文字幅を補正する
           MOVE SI-CODE TO S-R1(1:ICODE-LEN).
           ADD ICODE-LEN TO P.
      *
      * 出力データをそのまま転送
           MOVE PUT-DATA(1:PUT-LEN) TO S-R1(P:PUT-LEN).
           ADD PUT-LEN TO P.
      *
      * 漢字全角文字の文字幅を元に戻す
           MOVE EI-CODE TO S-R1(P:ICODE-LEN).
           WRITE S-R1 AFTER ADVANCING 1.

      *--- 終了処理 ---*
           CLOSE S-F1.
           EXIT PROGRAM.
```

1. 漢字の全角文字スペース量

   左スペース量：6 ドット

   右スペース量：6 ドット

8.6　XMAP3 による印刷（Windows(x86) COBOL2002 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。

8.6.1　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。

8.6.2　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

- GUI モードの場合，環境変数で印刷サービス名称が割り当てられていなければ，OPEN 文の実行時に表示される，物理ファイルの割り当て画面で印刷サービス名称を割り当てることもできます。物理ファイルの割り当て画面については，「6.2.4　実行時の動的割り当て（GUI モードの場合だけ）」を参照してください。
- 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 8-4　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 プログラムでの APPLY FORMS-OVERLAY 句の指定	外部装置名（環境変数）		ASSIGN 定数 または ASSIGN データ名
		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	○※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。

出力先がファイルの場合でも，物理ファイル名に'PRINTER'を指定すると，CHARACTER TYPE 句を有効としてプリンタに出力できます。この場合，XMAP3 を使用しないで，GDI モード印刷となります。「8.1　プリンタアクセスの種類と概要」を参照してください。

また，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 E　コンパイルリスト」を参照してください。

8.6.3　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。

8.6.4　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 プログラミングガイド 帳票編」またはマニュアル「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」を参照してください。

8.6.5　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 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」を参照してください。

8.6.6　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 プログラムは，XMAP3 の書式オーバレイ用のライブラリ（x3klib32.lib）をリンクする必要があります。ただし，ccbl2002 コマンドでコンパイルと同時にリンクまで実行する場合，XMAP3 ライブラリの指定は不要です。

- 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。

- 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。

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。

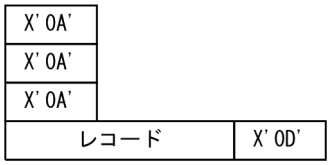
報告書ファイルの出力形式は，レコード形式と報告書記述項での CODE 句の有無によって異なります。報告書ファイルの出力形式と出力例を次に示します。出力例の X'0A'は改行，X'0D'は復帰，X'0C'は改ページを示します。

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。

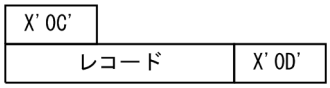
（例 1）

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

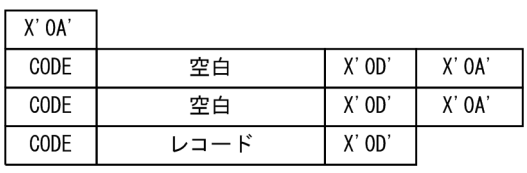
順編成ファイルに対する，AFTER ADVANCING 指定の WRITE 文の形式で出力されます。

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。

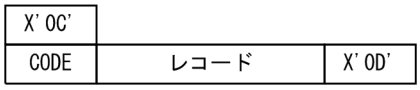
（例 1）

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



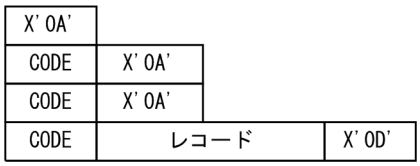
可変長で CODE 句の指定がある場合

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）や COBOL2002 の出力するコンソールウィンドウに対して，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，「10.2.4　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。

イベントログファイルへの出力

Windows のイベントログサービスで使えるイベントログファイルに対して，データをイベントとして出力できます。

詳細は，「10.5　イベント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や COBOL2002 の出力するコンソールウィンドウ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	
		GUI モードの場合	CUI モードの場合
SYSIN　または SYSIPT	CBL_SYSIN=stdin	ファイル名「stdin」	標準入力
	CBL_SYSIN=stdout	ファイル名「stdout」	（エラーとなる）
	CBL_SYSIN=stderr	ファイル名「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	COBOL2002 が出力するダイアログボックス	標準入力
CONSOLE		COBOL2002 が出力するダイアログボックス	標準入力
SYSSTD	CBL_SYSSTD=stdin	ファイル名「stdin」	標準入力
	CBL_SYSSTD=stdout	ファイル名「stdout」	（エラーとなる）
	CBL_SYSSTD=stderr	ファイル名「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	COBOL2002 が出力するダイアログボックス	標準入力

注

CUI モードのときに，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	
		GUI モードの場合	CUI モードの場合
SYSPUNCH または SYSPCH	CBL_SYSPUNCH=stdin〔+〕	ファイル名「stdin」	（エラーとなる）
	CBL_SYSPUNCH=stdout〔+〕	ファイル名「stdout」	標準出力※1
	CBL_SYSPUNCH=stderr〔+〕	ファイル名「stderr」	標準エラー出力※1
	CBL_SYSPUNCH=syslog〔+〕	ファイル名「syslog」※2	
	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	COBOL2002 が出力するウィンドウ	標準出力
SYSOUT または SYSLST	CBL_SYSOUT=stdin〔+〕	ファイル名「stdin」	（エラーとなる）
	CBL_SYSOUT=stdout〔+〕	ファイル名「stdout」	標準出力※1
	CBL_SYSOUT=stderr〔+〕	ファイル名「stderr」	標準エラー出力※1
	CBL_SYSOUT=syslog〔+〕	ファイル名「syslog」※2	
	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	COBOL2002 が出力するウィンドウ	標準出力
CONSOLE		COBOL2002 が出力するウィンドウ	標準出力

注

CUI モードのときに，標準出力（stdout），標準エラー出力（stderr），およびイベントログファイル出力（syslog）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※1

追加モードは無視されます。

注※2

イベントログファイル出力機能が有効な場合は，イベントログファイル出力になります。詳細は，「10.5　イベントログファイル出力機能」を参照してください。

10.2.2　CUI モード，GUI モードとの関係

ACCEPT 文，DISPLAY 文，および STOP 文の入出力先の指定が特にない場合，CUI，GUI モードでは次のような指定になります。

GUI モード

COBOL2002 が出力するダイアログボックス，またはコンソールウィンドウ

CUI モード

標準入出力（stdin，stdout）

10.2.3　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 に値が指定されていない場合，または CONSOLE 指定がある場合の入力先は，次のような指定になります。

モードの種類	データの入力元
GUI モード	COBOL2002 が出力するダイアログボックス
CUI モード	標準入力 標準エラー出力

- 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'）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

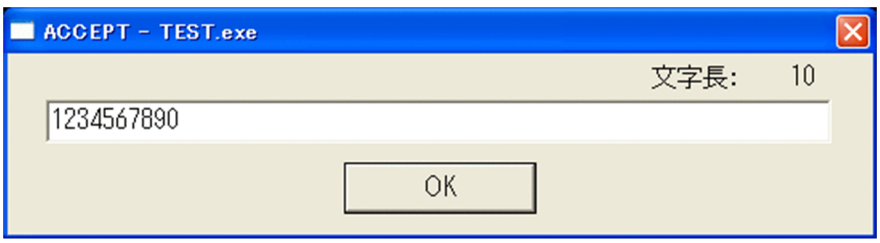
1 レコードは常に 80 バイトであり，改行文字がレコードの区切りとなります。1 レコードが 80 バイトを超える場合は，先頭から 80 バイトが転送され，残りのデータは改行文字まで切り捨てられます。1 レコードが 80 バイト未満の場合は，残った領域に空白文字が埋められたレコードとして扱われます。

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'）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。

(e) ACCEPT 文の表示形式（GUI モードの場合）

GUI モードで ACCEPT 文を実行した場合，COBOL2002 が出力するダイアログボックスを次に示します。



ここで［Esc］キーを押すと，ダイアログボックスが閉じ，データを入力しないで［OK］ボタンを選んだ場合と同じ動きとなります。

なお，ダイアログボックスに入力した文字はコンソールウィンドウに出力されます。

(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。

10.2.4　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）	符号のない 1 けたの数字項目

FROM 指定	内容	形式
	1：月曜日 2：火曜日 ： 7：日曜日	
TIME	hhmmsstt hh：00～23（時） mm：00～59（分） ss：00～59（秒） tt：00～99（1/100 秒）	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
```

10.2.5 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

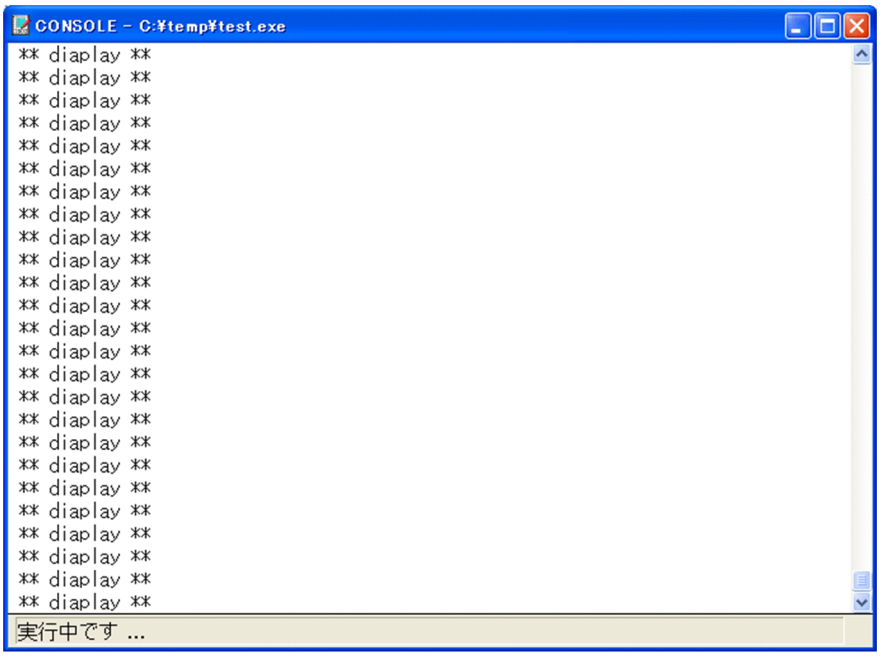
```
DISPLAY　{一意名｜定数}
　〔UPON　{SYSPUNCH｜SYSOUT｜SYSLST｜SYSPCH｜CONSOLE｜呼び名}〕
　〔WITH NO ADVANCING〕
　〔IN DATA DUMP〕
```

(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，「37.8　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。

(3)　DISPLAY 文の表示形式（GUI モードの場合）

GUI モードで DISPLAY を実行した場合，COBOL2002 が出力するウィンドウ（コンソールウィンドウ）を次に示します。



コンソールウィンドウの規則

- ウィンドウの表示サイズは，25 行×80 カラムで固定です。DISPLAY 文で出力したデータが 80 バイトを超える場合は，折り返して表示されます。
- コンソールウィンドウには，次のデータが出力されます。

コンソールウィンドウに出力されるデータの種類

- 出力先がファイルでない場合の DISPLAY 文
- COBOL2002 が出力する実行時メッセージ
- 入力先の割り当てがファイルでない場合の ACCEPT 文で入力されたデータ

- ウィンドウ内の文字のフォントは，実行支援ウィンドウの［設定］メニューの［画面環境］で表示される画面環境ダイアログボックスの［フォント］ボタンで変更できます。変更方法については，マニュアル「COBOL2002　操作ガイド」を参照してください。
- ウィンドウの文字色と背景色は，Windows のコントロールパネルで変更できます。

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

GUI モードの場合の規則

STOP 定数文が実行されると，実行が中断し，COBOL2002 が出力するダイアログボックスに定数 1 が表示されます。次に，例を示します。

（STOP 定数文を使用したプログラムの例）

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. TEST1.
       PROCEDURE DIVISION.
           STOP '＊＊　ＳＴＯＰ定数　＊＊'
           EXIT PROGRAM.
       END PROGRAM TEST1.
```

（GUI モードで出力されるダイアログボックス）

このダイアログボックスの［OK］ボタンを選ぶと，プログラムが再実行されます。

CUI モードの場合の規則

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

```
SAMPLE1.EXE 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，または指定した環境変数に値がないとき，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。
  - CBLACTWIN
  - CBLCOMCBR
  - CBLTDEXEC
- 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。
- DISPLAY 文の呼び名 4 指定で書き出す環境変数の値の長さは，32,766 バイト以内でなければなりません。32,766 バイトを超える値を指定した場合，32,766 バイトまでの値を環境変数に書き出します。
- ACCEPT 文の呼び名 4 指定で取得する環境変数の値の長さは，32,766 バイト以内でなければなりません。32,766 バイトを超える環境変数の値を指定した場合，実行時エラーとなります。

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　イベントログファイル出力機能

10.5.1　イベントログファイル出力機能の概要

イベントとイベントログファイル

Windows では，システムまたはアプリケーションで発生した，ユーザへの通知を必要とする重要な出来事をイベントと呼びます。

通常，多くのイベントは，Windows によってイベントログファイルに情報が出力されます。これをイベントログサービスといい，イベントログサービスは Windows を起動するたびに自動的に開始されます。

イベントログファイルに出力されたイベントは，イベントビューアで監視できます。イベントビューアを使用すると，システム，セキュリティ，およびアプリケーションのイベントログファイルを表示，管理できます。イベントビューアの詳細については，イベントビューアのヘルプを参照してください。

イベントログファイル出力機能とは

COBOL2002 のイベントログファイル出力機能は，COBOL プログラム実行中に出力されるエラーメッセージを，イベントログファイルに出力する機能です。また，DISPLAY 文の出力内容をイベントログファイルに出力することもできます。出力されたイベントは，イベントビューアで参照できます。DISPLAY 文の文法規則については，マニュアル「COBOL2002　言語　標準仕様編」「10.8.12 DISPLAY 文」を参照してください。

10.5.2　イベントの出力

イベントログファイル出力機能を使用するには，プログラムの実行時に次の指定が必要です。

```
set CBLSYSLOG=EVLOG
```

環境変数 CBLSYSLOG に EVLOG 以外の値を指定したり，値を指定しなかったりした場合は，イベントは出力されません。

また，DISPLAY 文を用いてイベントを出力する場合，上記の環境変数 CBLSYSLOG の指定に加えて次の指定が必要です。

```
set CBL_SYSPUNCH=syslog
  または
set CBL_SYSOUT=syslog
```

10.5.3　イベントの出力内容

イベントの出力内容およびイベントの種類の指定方法を次に示します。

(1) 出力内容

イベントログファイル出力機能で出力されるイベントの出力内容を，次に示します。

表 10–2 イベントの出力内容

項目名	出力内容
日付	イベントが出力された年月日
時刻	イベントが出力された時刻
ユーザ名	ログオンしているユーザ名
コンピュータ名	イベントが発生したコンピュータ名
イベント ID	• 実行時メッセージを出力する場合 COBOL メッセージ番号 • DISPLAY 文で出力する場合 「0」
ソース名	イベントが発生したアプリケーション名。 イベントログファイル出力機能使用時のソース名は， Windows(x86) COBOL2002 の場合は「COBOL2002」 Windows(x64) COBOL2002 の場合は「COBOL2002 64bit」
種類	「情報」，「警告」，「エラー」のどれか
分類	• 実行時メッセージを出力する場合 なし • DISPLAY 文で出力する場合 「情報」
説明	• 実行時メッセージを出力する場合 COBOL プログラムが出力する実行時エラーメッセージ※ （プリフィクス，メッセージ番号，メッセージレベル，およびメッセージテキスト） • DISPLAY 文で出力する場合 DISPLAY 文で指定された一意名または定数

注※

実行時エラーメッセージには，それぞれ対応するイベントの種類が設定されています。実行時エラーメッセージのレベルとイベントの種類の関係を，次に示します。

表 10–3 実行時エラーメッセージのレベルとイベントの種類

実行時エラーメッセージのレベル	イベントの種類
回復不能エラー（U） エラー（S）	エラー
警告（W）	警告
お知らせ（I）	情報

なお，DISPLAY 文を用いてイベントを出力する場合は，出力するイベントの種類を指定できます。

(2) イベントの種類の指定方法

DISPLAY 文を用いてイベントを出力する場合，出力するイベントの種類を指定できます（実行時エラーメッセージ出力時は指定できません）。出力するイベントの種類を指定するには，プログラムの実行時に次の指定が必要です。

```
set CBLSYSLOGLVL= {ERR | WAR | INF}
```

環境変数 CBLSYSLOGLVL に指定された値は，イベントログファイル出力用の DISPLAY 文が実行されるたびに参照されます。環境変数 CBLSYSLOGLVL に指定した値とイベントの種類の対応を，次に示します。

表 10–4 環境変数 CBLSYSLOGLVL の指定値とイベントの種類

環境変数 CBLSYSLOGLVL の指定値	イベントの種類
ERR	エラー
WAR	警告
INF	情報

注

環境変数 CBLSYSLOGLVL に値が指定されていない場合は，INF が仮定されます。

10.5.4 イベントの出力先

イベントの出力先および出力先の指定方法を次に示します。

(1) 出力先

イベントログファイルには次の 3 種類があります。

1. システムログファイル
2. セキュリティログファイル
3. アプリケーションログファイル

イベントログファイル出力機能では，3.のアプリケーションログファイルを使用します。

イベントログファイルは，システムによって用意されるファイルです。また，イベントログファイル出力機能では，ローカルコンピュータのイベントログファイルだけでなく，ネットワーク上のコンピュータのイベントログファイルに対してもイベントを出力できます。

(2) 出力先の指定方法

イベントの出力先コンピュータを指定するには，プログラムの実行時に次の指定が必要です。

```
set CBLSYSLOGSRV=コンピュータ名
```

環境変数 CBLSYSLOGSRV には，ネットワーク上で有効なコンピュータ名を指定してください。ネットワーク上で無効なコンピュータ名を指定したり，コンピュータ名を指定しなかったりした場合は，ローカルコンピュータのイベントログファイルへイベントが出力されます。

イベントログファイル出力機能でネットワーク上のコンピュータにイベントを出力し，出力先コンピュータ上でイベントビューアを使ってアプリケーションログを表示すると，COBOL プログラムで出力したメッセージの前に「イベント ID(nnnn) （ソース COBOL2002 内）に関する説明が見付かりませんでした。・・・」というシステムのメッセージが付けられることがあります。

この場合は，次のどちらかの方法でシステムのメッセージ付けを抑止できます。

- イベントを出力するネットワーク上のコンピュータに，COBOL2002 をインストールする。
- COBOL2002 をインストールしているマシンのイベントビューアの［操作］メニューから［別のコンピュータへ接続］を選択して，イベントを出力したコンピュータを選択し，イベントを表示する。

10.5.5 イベントログファイルへの出力がエラーになったときの動作

出力するイベントに不当な文字コードが含まれていたときは，文字コードが置き換えられてからイベントログファイルに書き込まれます。詳細は，「10.5.6 注意事項」の「(3) イベントログファイルに出力する文字列に不当な文字が含まれている場合の動作」を参照してください。

それ以外の理由でイベントの出力に失敗した場合，イベントの出力先の指定内容によって次のどちらかの処理が行われます。なお，イベントの出力先の指定方法については，「10.5.4 イベントの出力先」の「(2) 出力先の指定方法」を参照してください。

(1) イベントの出力先がローカルコンピュータの場合

次の順序でイベントログファイルの出力処理が実行されます。

1. イベントをイベントログファイルに書き込む
2. 1.に失敗したときは，書き込みに失敗したイベントを破棄し，GUI モードのときはコンソールウィンドウに，CUI モードのときは標準エラー出力（stderr）に，実行時エラーメッセージを出力する

(2) イベントの出力先がネットワーク上のコンピュータの場合

次の順序でイベントログファイルの出力処理が実行されます。

1. ネットワーク上のコンピュータのイベントログファイルにイベントを書き込む
2. 1.に失敗したときは，ローカルコンピュータのイベントログファイルにイベントを書き込む
3. 2.にも失敗したときは，書き込みに失敗したイベントを破棄し，GUI モードのときはコンソールウィンドウに，CUI モードのときは標準エラー出力（stderr）に，実行時エラーメッセージを出力する

10.5.6 注意事項

イベントログファイル出力機能を使用する際の注意事項を示します。

(1) DISPLAY 文の出力データ長

DISPLAY 文を用いてイベントログファイルに出力できるデータの長さは 1,024 バイトまでです。DISPLAY 文を用いて 1,024 バイトを超えるデータをイベントログファイルに出力した場合，1,025 バイト以降のデータは無視されます。

(2) テストデバッグ時の動作

テストデバッグ時に出力される実行時メッセージは，イベントログファイル出力機能の対象になりません。ただし，DISPLAY 文の出力内容は，イベントログファイルに出力できます。

(3) イベントログファイルに出力する文字列に不当な文字が含まれている場合の動作

(a) イベントの中に NULL 文字が含まれている場合

NULL 文字以降の文字列は，イベントログファイルに出力されません。

(b) イベントログファイルに出力する文字列に不当な文字が含まれている場合

イベントの出力エラーになることがあります。その場合は，次のように処理されます。

イベントの中に，次の文字コードの範囲以外の文字が含まれている場合

ASCII コード

X'0D'，X'0A'，X'20'～X'7E'，X'A1'～X'DF'

シフト JIS コード

第 1 バイト：X'81'～X'9F'，X'E0'～X'FC'

第 2 バイト：X'40'～X'7E'，X'80'～X'FC'

不当な文字が次のように変換され，イベントログファイルにテキスト出力されます。テキスト出力されたデータは，イベントビューアの説明欄に表示されます。

シフト JIS コードで，第 1 バイトが範囲内，第 2 バイトが範囲外のとき

げた記号「 = 」（X'81AC'）に変換されます。

それ以外の文字コードのとき

ピリオド「.」（X'2E'）に変換されます。

また，変換前の文字列は，イベントログファイルにバイナリ出力されます。バイナリ出力されたデータは，イベントビューアのデータ欄に 16 進表示されます。

(4) DISPLAY 文を用いたイベント出力での改行

DISPLAY 文を用いてイベントログファイルにイベントを出力する場合，改行コードを用いて改行しないでください。

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フォルダの名称は，環境変数 CBLSORTWORK で指定します。

形式

```
CBLSORTWORK=フォルダ名
```

フォルダ名は，ドライブ名からの絶対パス名で指定します。

（例）

```
CBLSORTWORK=C:¥TMP¥WORK
```

環境変数を指定しなかった場合，作業ファイルのフォルダは，実行可能ファイルがあるフォルダになります。

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
```

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

Windows(x86) COBOL2002 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

Windows(x64) COBOL2002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

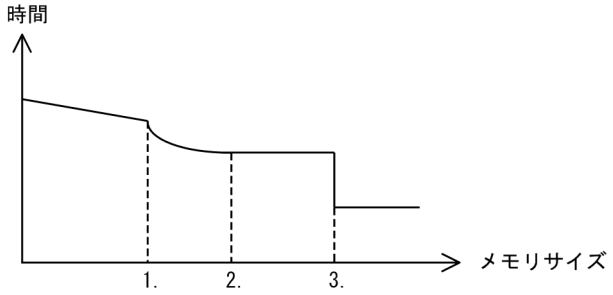
11.5.1　処理時間の短縮

(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

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

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 開発・実行ガイド」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 開発・実行ガイド」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。

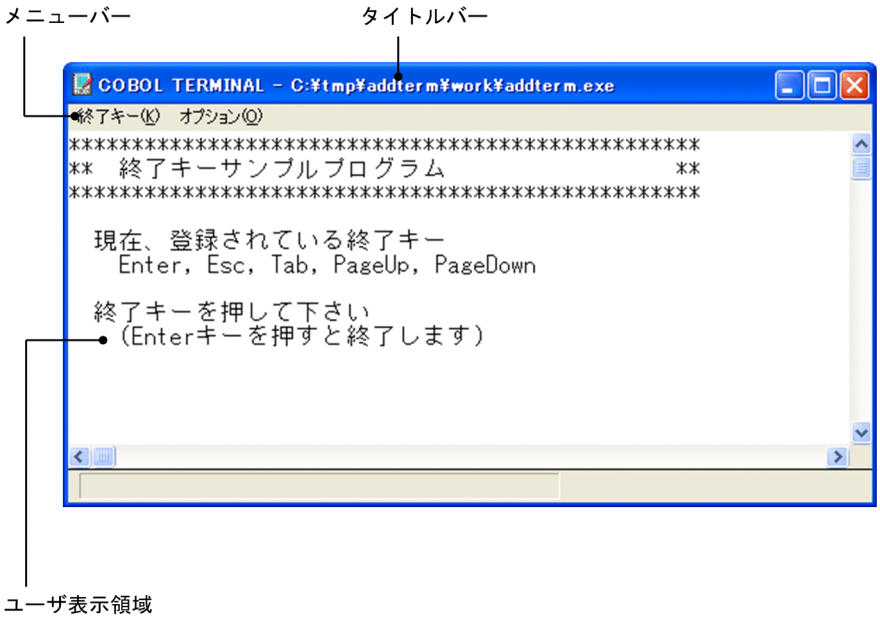
なお，SCREEN SECTION と WINDOW SECTION は，同時に使用できません。同時に使用した場合，エラーメッセージが出力されます。

12.2.1　画面の種類と構成

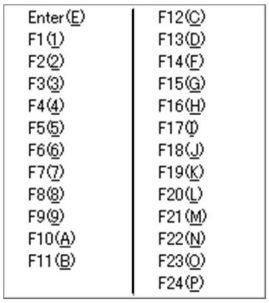
画面節（SCREEN SECTION）では，主画面とエラー表示画面の 2 種類の画面を使用します。

(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。



注 1

［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

［F10］キーは通常システムキーとして割り当てられています。これを COBOL プログラム中で使用するためには，ユーザキーとして割り当てる必要があります。割り当て方法については，マニュアル「COBOL2002　操作ガイド」の実行支援を使った画面環境の設定の説明を参照してください。

また，メニューバーの［オプション］をクリックすると，［フォントサイズ変更］コマンドが表示されます。これをクリックすると，フォントの指定ダイアログボックスが表示されます。

このダイアログボックスでフォントのサイズ，フォント名を指定して［OK］ボタンを選ぶと，ユーザ表示領域に表示される文字のサイズやフォントを変更できます。デフォルトでは，次のフォントが指定されています。

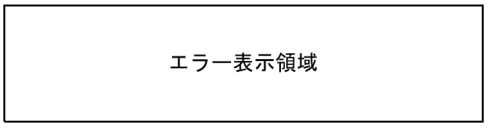
- フォント名：ＭＳ ゴシック
- サイズ：12

注意事項

- 日本語を表示できないフォントは，指定できません。
- 実行支援で，システムで使用できないフォントや日本語を表示できないフォントが指定されている場合，デフォルトのフォントが使用されます。

(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

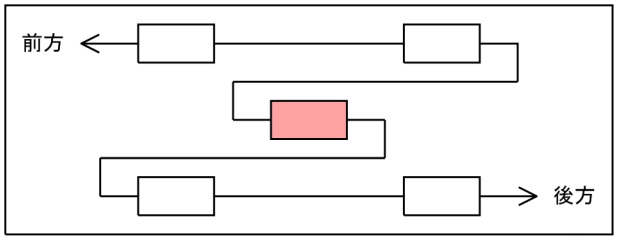
カーソルの移動などで使用するキーの意味を次に示します。

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。
[Insert]	現在カーソルが位置づけられているカラムに空白または 0 を挿入します。
[Delete]	現在カーソルが位置づけられているカラムの文字を削除します。
[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。
[Alt] ＋ [矢印]	矢印（[→]，[←]，[↑]，[↓]）の方向に画面がスクロールします。
[Enter]， [F1] ～ [F24]， [実行]	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。

カーソルの移動などの機能に対して割り当てられたキーは変更できます。キーの割り当て方法については，マニュアル「COBOL2002　操作ガイド」の実行支援を使った画面環境の設定の説明を参照してください。

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

- 実行支援の画面環境で割り当てたキーは，画面節（SCREEN SECTION および WINDOW SECTION）だけで有効です。

12.2.3　LINE／COLUMN 句を使用した画面の座標指定

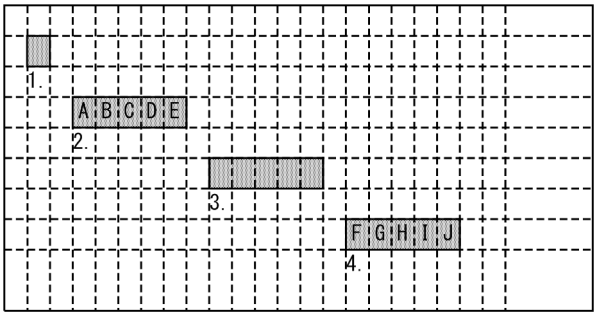
画面節（SCREEN SECTION）では，LINE／COLUMN 句を指定すると，画面項目に画面上での座標を与えられます。例を次に示します。

（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。

ファンクションキーとキー番号は，［F1］キー：1，［F2］キー：2，……［F24］キー：24 のように対応します。

注 1

［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

［F10］キーは通常システムキーとして割り当てられています。これを COBOL プログラム中で使用するためには，ユーザキーとして割り当てる必要があります。割り当て方法については，マニュアル「COBOL2002　操作ガイド」の実行支援を使った画面環境の設定の説明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

- 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。
  - 実行支援の画面設定で画面サイズを大きくする。

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。

Enter(E)	F12(C)
F1(1)	F13(D)
F2(2)	F14(F)
F3(3)	F15(G)
F4(4)	F16(H)
F5(5)	F17(I)
F6(6)	F18(J)
F7(7)	F19(K)
F8(8)	F20(L)
F9(9)	F21(M)
F10(A)	F22(N)
F11(B)	F23(O)
	F24(P)

注 1

［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

［F10］キーは通常システムキーとして割り当てられています。これを COBOL プログラム中で使用するためには，ユーザキーとして割り当てる必要があります。割り当て方法については，マニュアル「COBOL2002　操作ガイド」の実行支援を使った画面環境の設定の説明を参照してください。

また，メニューバーの［オプション］をクリックすると，［フォントサイズ変更］コマンドが表示されます。これをクリックすると，フォントの指定ダイアログボックスが表示されます。

このダイアログボックスでフォントのサイズ，フォント名を指定して［OK］ボタンを選ぶと，ユーザ表示領域に表示される文字のサイズやフォントを変更できます。デフォルトでは，次のフォントが指定されています。

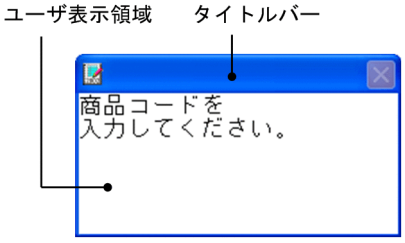
- フォント名：ＭＳ ゴシック
- サイズ：12

注意事項

- 日本語を表示できないフォントは，指定できません。
- 実行支援で，システムで使用できないフォントや日本語を表示できないフォントが指定されている場合，デフォルトのフォントが使用されます。

(2)　ポップアップ画面

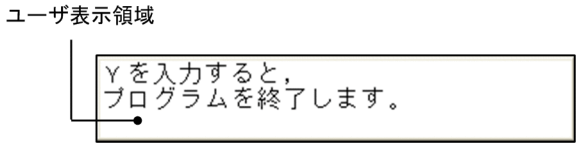
ポップアップ画面は，ポップアップ入出力機能で使用する画面であり，タイトルバーとユーザ表示領域から構成されます。ユーザ表示領域のサイズはプログラム中で指定します。ポップアップ画面の構成を次に示します。



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

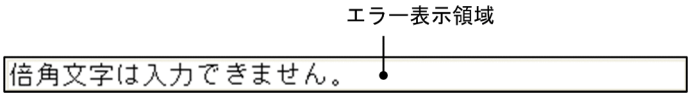
ユーザポップアップ HELP 画面は，ユーザポップアップ HELP 機能で使用する画面であり，ユーザ表示領域だけから構成されます。ユーザ表示領域のサイズはプログラム中で指定します。ユーザポップアップ HELP 画面の構成を次に示します。



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	1 2 3 . 4 0
9(6)V	123456	1 2 3 4 5 6
S9(3)V9(2)	-12.34	- 1 2 . 3 4 (左符号の場合)
S9(3)V9(2)	+12.34	1 2 . 3 4 + (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。

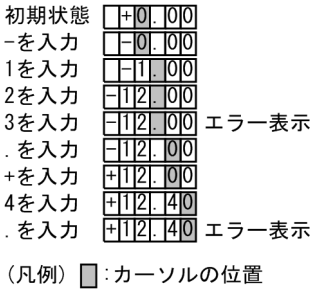
12.3.3　データの入力方式

(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。

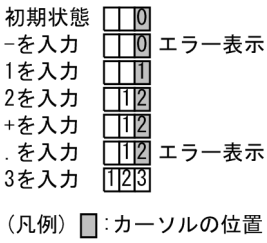
(例 1)

PIC S99V99 の場合



(例 2)

PIC 999 の場合

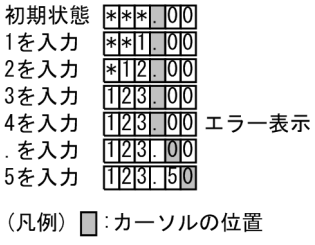


(2)　数字編集項目の入力方式

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。

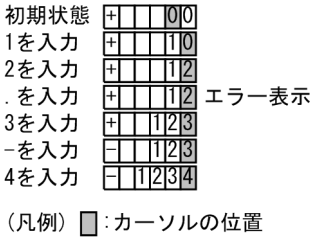
(例 1)

PIC ***.**の場合



(例 2)

PIC +ZZZ99 の場合

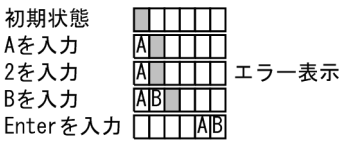


(3) 英字項目，英数字項目，日本語項目の入力方式

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

(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。

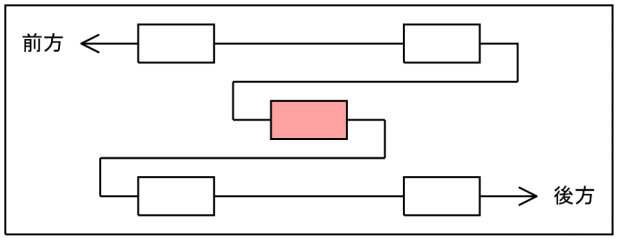
使用するキー	意味
[→]	カーソルが右に 1 カラム移動します。ただし，フィールドの右端のときは，後方のフィールドの先頭に移動します。
[←]	カーソルが左に 1 カラム移動します。ただし，フィールドの左端のときは，前方のフィールドの末尾に移動します。
[↑]	前方のフィールドの先頭にカーソルが移動します。
[↓]	後方のフィールドの先頭にカーソルが移動します。
[Tab]	後方のフィールドの先頭にカーソルが移動します。
[BackSpace]	現在カーソルが位置づけられているカラムの入力を取り消し，カーソルが左に 1 カラム移動します。
[Insert]	現在カーソルが位置づけられているカラムに空白または 0 を挿入します。
[Delete]	現在カーソルが位置づけられているカラムの文字を削除します。
[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。 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。
[Alt] ＋ [矢印]	矢印（[→]，[←]，[↑]，[↓]）の方向に画面がスクロールします。
[Enter]， [F1] ～ [F24]， [実行]	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。

カーソルの移動などの機能に対して割り当てられたキーは変更できます。キーの割り当て方法については，マニュアル「COBOL2002　操作ガイド」の実行支援を使った画面環境の設定の説明を参照してください。

また，フィールド間のカーソル移動の動作変更については，「36.2.6　画面」の「(14) CBLUPDOWNMOVE」を参照してください。

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。
- 実行支援の画面環境で割り当てたキーは，画面節（SCREEN SECTION および WINDOW SECTION）だけで有効です。

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，[実行] キー）を押すと，ポップアップ画面が閉じます。

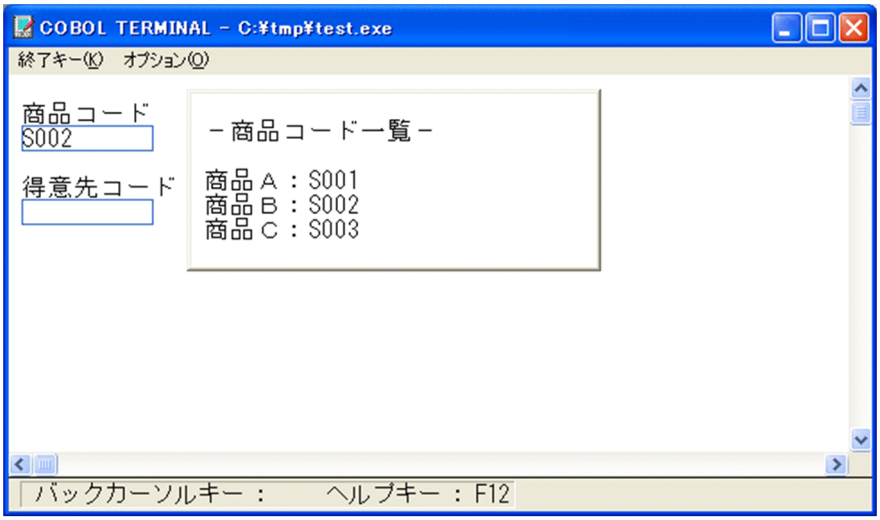
注意事項

- HELP LINE 指定，HELP COLUMN 指定で指定する表示位置は，ユーザ表示領域の位置を示したものであり，タイトルバーは含みません。
- 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（[F12] 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，［実行］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) 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。
- 実行支援の画面設定で画面サイズを大きくする。

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）
  - GUI モードでの動作
- 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 WINAPI 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
	Windows(x86) COBOL2002 の場合	Windows(x64) COBOL2002 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		Windows(x86) COBOL2002 の場合			Windows(x64) COBOL2002 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		Windows(x86) COBOL2002 の場合			Windows(x64) COBOL2002 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		Windows(x86) COBOL2002 の場合			Windows(x64) COBOL2002 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		Windows(x86) COBOL2002 の場合			Windows(x64) COBOL2002 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「13.7 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		Windows(x86) COBOL2002 の場合			Windows(x64) COBOL2002 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
	未使用	21	7	char[7]	29	11	char[11]	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，次のフォルダの下に提供されています。

```
COBOL2002インストールフォルダ¥include
```

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		Windows(x86) COBOL2002 の場合			Windows(x64) COBOL2002 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 H 入出力状態の値」を参照

データ項目名		Windows(x86) COBOL2002 の場合			Windows(x64) COBOL2002 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
	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	

データ項目名		Windows(x86) COBOL2002 の場合			Windows(x64) COBOL2002 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		
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，次のフォルダの下に提供されています。

```
COBOL2002インストールフォルダ¥include
```

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

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
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)
		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
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

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_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		Windows(x86) COBOL2002 の場合			Windows(x64) COBOL2002 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	－	－	－	－

データ項目名		Windows(x86) COBOL2002 の場合			Windows(x64) COBOL2002 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
	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） CBLPARM_ACCESS_KEY：乱アクセス（READ KEY）		－	－	○※4	－	－	－	－	－	－
	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		－	－	－	－	－	－	－	○	－	－

データ項目名		Windows(x86) COBOL2002 の場合			Windows(x64) COBOL2002 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
								(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 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，cbl85fl.lib の指定が必要です。指定形式を次に示します。

形式

```
cl  -Iyyy  xxx.c  cbl85fl.lib
```

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フォルダ

```
COBOL2002インストールフォルダ¥include
```

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

```
ccbl2002 -Main,System zzz.cbl xxx.obj
```

xxx.obj

　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

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。

注意事項

環境変数 CBL_FLSRVDUMP を実行支援で指定した場合，最初の COBOL プログラムが動作するまで，この環境変数の指定は無効となります。

(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	○	○	×	○

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
CBLREWRITE	○	○	×	○
CBLUNLOCK	○	×	×	×
CBLDELETE	○	○	×	×
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 = C:¥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 = C:¥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 = C:¥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複数レコード施錠

COBOL2002 の入出力機能と同様に，COBOL 入出力サービスルーチンでも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。複数レコード施錠の詳細については，「7.2.2　レコードレベルのファイル共用」を参照してください。

複数レコード施錠は，相対ファイルだけで使用できます。また，COBOL2002 の入出力機能と同様に，環境変数 CBL_RECLOCKMAX で同時に施錠するレコード数を指定できます。

注意事項

環境変数 CBL_RECLOCKMAX を実行支援で指定した場合，最初の COBOL プログラムが動作するまで，この環境変数の指定は無効となります。

13.7　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

13.7.1　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。

13.7.2　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。

13.7.3　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。

13.7.4　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。

13.8 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, "c:¥¥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	×	×
Btrieve（Pervasive.SQL）による索引編成ファイル※1	×	×
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※2

注※1

Windows(x86) COBOL2002 で有効です。

注※2

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。

Btrieve（Pervasive.SQL）による索引編成ファイルのディスク書き込み保証

Btrieve（Pervasive.SQL）による索引編成ファイルのディスク書き込み保証は，Pervasive.SQL，Btrieve の仕様に従います。詳細は，Pervasive.SQL，Btrieve のマニュアルを参照してください。

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，「13.7　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。

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

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。

(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 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 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 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	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

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。
- バイトストリーム入出力サービスルーチンは，ラージファイル（ファイルサイズが 2GB 以上のファイル）を取り扱えません。バイトストリーム入出力サービスルーチンで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。
- 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。 • 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。

値	内容
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

- Windows(x86) COBOL2002 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。
- Windows(x64) COBOL2002 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）の予備領域で，0 を指定します。
- Windows(x86) COBOL2002 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。
- Windows(x64) COBOL2002 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）の予備領域で，0 を指定します。
- Windows(x86) COBOL2002 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。

- Windows(x64) COBOL2002 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

- Windows(x86) COBOL2002 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。
- Windows(x64) COBOL2002 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

- Windows(x86) COBOL2002 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。
- Windows(x64) COBOL2002 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。

Windows(x86) COBOL2002 の場合

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SAMPLE1.
   :
 DATA  DIVISION.
 WORKING-STORAGE SECTION.
   01 FILE-NAME      PIC X(255)      VALUE 'C:¥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.
```

Windows(x64) COBOL2002 の場合

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SAMPLE1.
   :
 DATA  DIVISION.
```

```
 WORKING-STORAGE SECTION.
   01 FILE-NAME      PIC X(255)      VALUE 'C:¥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.
```

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 DLL 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 DLL（C プログラムなど）を含むことができます。

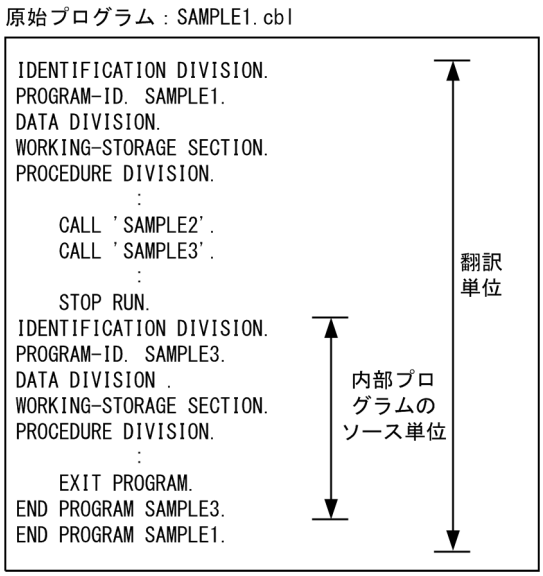
16.1.2　実行可能ファイルや DLL とは

実行可能ファイルや DLL 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 DLL 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。

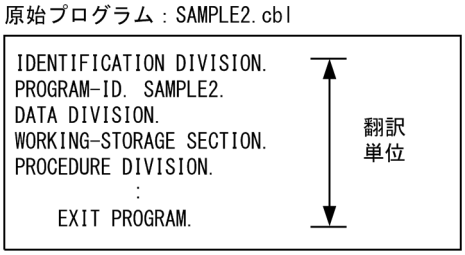
16.1.3　実行単位と実行モジュールの例

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）

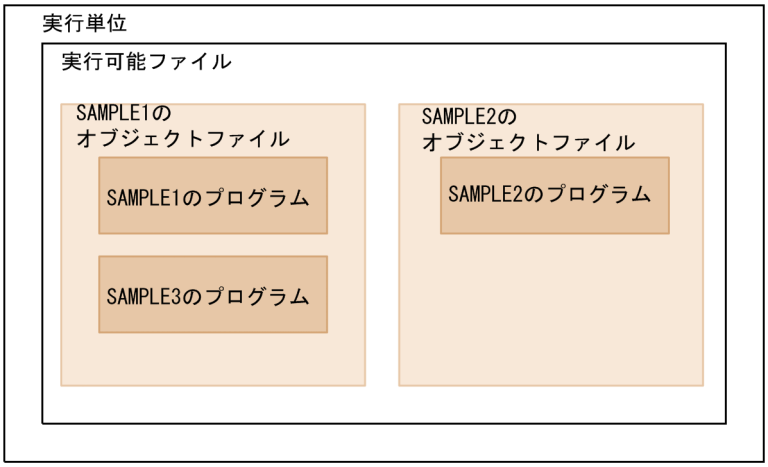


(原始プログラム：SAMPLE2.cbl)

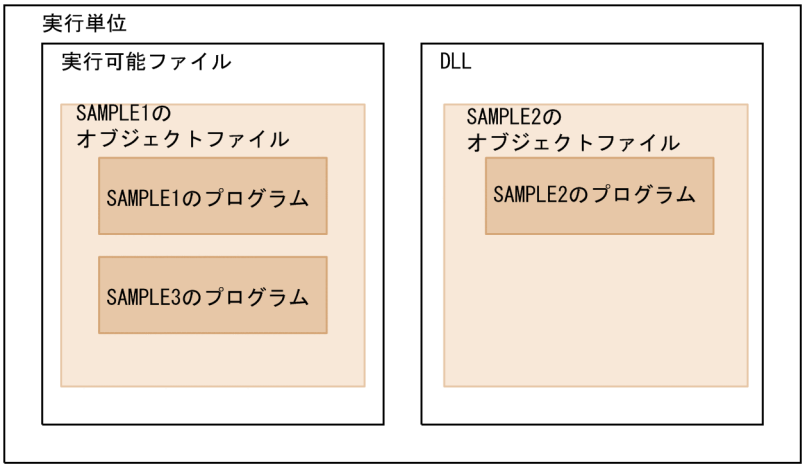


(生成されるプログラムの構成)

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 DLL 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）

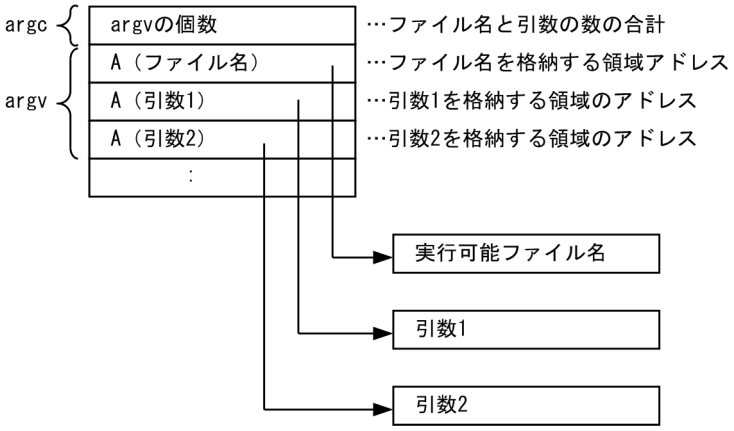
△：半角空白文字

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。

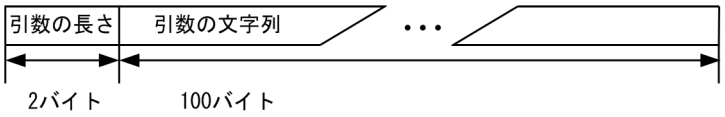
コマンド行

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，「30.4.4　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，「30.4.5　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 プログラムのモード

実行単位中の COBOL プログラムは，CUI モード／GUI モードのどちらかで実行されます。

16.3.1 CUI モード

CUI モードでは，ACCEPT／DISPLAY 文での入出力先や実行時メッセージの出力先が，OS の標準入出力（stdin，stdout，stderr）となります。CUI モードでの COBOL プログラムの実行例を，次に示します。

(実行するプログラム)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. TEST1.

       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       SELECT TXT-FILE ASSIGN TO FILE-NAME
                      ORGANIZATION IS LINE SEQUENTIAL.
       SELECT DMY-FILE ASSIGN TO SYS001
                      ORGANIZATION IS LINE SEQUENTIAL.
       DATA DIVISION.
       FILE SECTION.
       FD TXT-FILE.
       01 TXT-REC PIC X(40).
       FD DMY-FILE.
       01 DMY-REC PIC X(40).
       WORKING-STORAGE SECTION.
       01 FILE-NAME PIC X(40).

       PROCEDURE DIVISION.
           DISPLAY '--- プログラムのテストを開始します ---'. …1.
           DISPLAY '--- ファイル名を入力してください   ---'. …1.
           ACCEPT FILE-NAME.                                …2.

           OPEN  OUTPUT TXT-FILE.
           WRITE TXT-REC.
           CLOSE TXT-FILE.

           OPEN  INPUT DMY-FILE.                            …3.
           READ  DMY-FILE.
           CLOSE DMY-FILE.

           EXIT PROGRAM.

       END PROGRAM TEST1.
```

(実行画面)

```
D:¥>test1.exe
--- プログラムのテストを開始します ---                …1.
--- ファイル名を入力してください   ---                …1.
test1_outfile.txt                                    …2.
```

```
KCCC3517R-S                                                    …3.
CBL_外部装置名で示される環境変数が指定されていません。
プログラム名＝TEST1
行番号／欄＝000029/12
環境変数名＝CBL_SYS001


D:¥>
```

1. DISPLAY 文によるメッセージの出力です。標準出力に出力されます。
2. ACCEPT 文によるファイル名の入力です。標準入力から入力します。
3. ファイルに対応する環境変数未設定による実行時メッセージです。実行時メッセージは，標準エラー出力に出力されます。

16.3.2 GUI モード

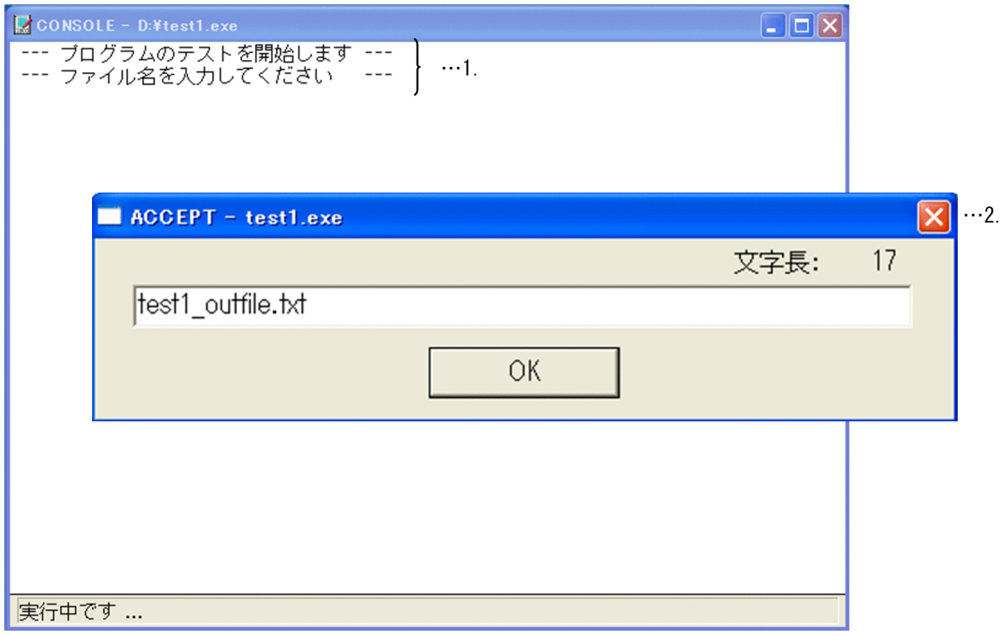
GUI モードでは，ACCEPT／DISPLAY 文での入出力先や実行時メッセージの出力先が，COBOL2002 が出力するコンソールウィンドウとなります。GUI モードでの COBOL プログラムの実行例を，次に示します。

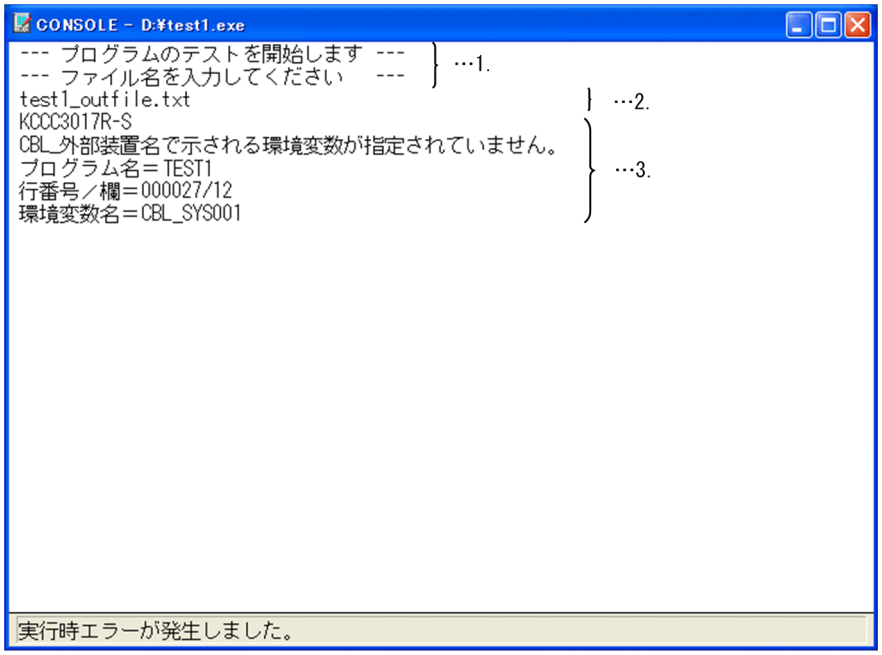
(実行するプログラム)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. TEST1.

       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       SELECT TXT-FILE ASSIGN TO FILE-NAME
                       ORGANIZATION IS LINE SEQUENTIAL.
       SELECT DMY-FILE ASSIGN TO SYS001
                       ORGANIZATION IS LINE SEQUENTIAL.
       DATA DIVISION.
       FILE SECTION.
       FD TXT-FILE.
       01 TXT-REC PIC X(40).
       FD DMY-FILE.
       01 DMY-REC PIC X(40).
       WORKING-STORAGE SECTION.
       01 FILE-NAME PIC X(40).

       PROCEDURE DIVISION.
           DISPLAY '--- プログラムのテストを開始します ---'. …1.
           DISPLAY '--- ファイル名を入力してください   ---'. …1.
           ACCEPT FILE-NAME.                                 …2.

           OPEN OUTPUT TXT-FILE.
           WRITE TXT-REC.
           CLOSE TXT-FILE.

           OPEN INPUT DMY-FILE.                              …3.
           READ DMY-FILE.
```

```
           CLOSE DMY-FILE.

           EXIT PROGRAM.

       END PROGRAM TEST1.
```

（実行画面）





1. DISPLAY 文によるメッセージの出力です。コンソールウィンドウに出力されます。
2. ACCEPT 文によるファイル名の入力です。入力用のダイアログボックスから入力します。ダイアログボックスに入力した値は，DISPLAY 文の出力結果と同様に，コンソールウィンドウに出力されます。

3. ファイルに対応する環境変数未設定による実行時メッセージです。実行時メッセージも，コンソールウィンドウに出力されます。

コンソールウィンドウ

GUI モードでプログラムを実行する場合，次のようなときに COBOL コンソールウィンドウが出力されます。

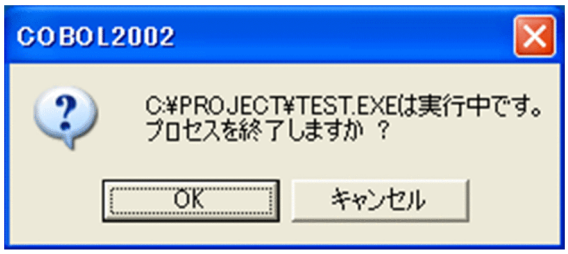
- 実行単位で最初の DISPLAY 文が実行された。このとき，環境変数 CBL_SYSOUT または CBL_SYSPUNCH でファイルが割り当てられていない。
- 実行単位で最初の ACCEPT 文が実行された。このとき，環境変数 CBL_SYSIN または CBL_SYSSTD でファイルが割り当てられていない。
- 実行時エラーが発生した。このとき，環境変数 CBL_SYSERR でファイルが割り当てられていない。

COBOL コンソールウィンドウのステータスバーには，プログラムの実行状態が次のように表示されます。

- COBOL プログラムが実行中の場合：「実行中です」
- STOP RUN 文が実行された場合：「実行が終了しました」
- 実行時エラーが発生した場合：「実行時エラーが発生しました」

コンソールウィンドウからプログラムを終了する方法を，次に示します。

- ステータスバーに「実行が終了しました」または「実行時エラーが発生しました」の表示があるとき
  コンソールウィンドウのシステムメニューから［閉じる］を選ぶとプロセスが終了します。
- ステータスバーに「実行中です」の表示があるとき
  コンソールウィンドウのシステムメニューから［閉じる］を選ぶと，次のメッセージボックスが出力されます。ここで［OK］ボタンを選ぶとプロセスが終了します。



16.3.3　モードの決定方法

COBOL プログラムを GUI モード，CUI モードのどちらで実行するかは，アプリケーションの主プログラムが COBOL プログラムの場合と，他言語のプログラムの場合とで，次のように指定方法が異なります。

(1)　アプリケーションの主プログラムが COBOL プログラムの場合

アプリケーションの主プログラムが COBOL プログラムの場合は，アプリケーションの主プログラムのコンパイル時にコンパイラオプションを指定することで，GUI モード，CUI モードのどちらで実行するかを選択できます。

形式

ccbl2002　{-Lib,GUI ｜-Lib,CUI}　ソースファイル名

-Lib,GUI

COBOL プログラムを，GUI モードで実行したい場合に指定します。

-Lib,CUI

COBOL プログラムを，CUI モードで実行したい場合に指定します。

-Lib,GUI，-Lib,CUI オプションのどちらも指定しなかった場合は，-Lib,GUI オプションが仮定されます。また，両方指定した場合は，後に指定したオプションが有効となります。

(2)　アプリケーションの主プログラムが他言語のプログラムの場合

他言語の主プログラムから COBOL で作成した副プログラムを呼び出すと，COBOL プログラムは，CUI モードで実行されます。

COBOL プログラムを GUI モードで実行したい場合は，COBOL プログラムを呼び出す前に，他言語の主プログラムから CBLGINT サービスルーチンを呼び出してください。

他言語のプログラムから，CBLGINT サービスルーチンを使って COBOL プログラムを GUI モードで実行する方法を，次に示します。なお，CBLGINT サービスルーチンの規則については，「30.4.1 CBLGINT」を参照してください。

(a)　C プログラムからの呼び出し例

```
#include <windows.h>

  extern int WINAPI CBLGINT();  /*CBLGINTの外部参照宣言*/

  CBLGINT();                    /*CBLGINTの呼び出し*/

   (COBOLプログラムの呼び出し)
```

(b)　Visual Basic からの呼び出し例

```
関数宣言
Private Declare Sub CBLGINT Lib "CBL85RT.DLL"
        Alias "_CBLGINT@0" ()

手続き
Private Sub Command1_Click()
  Call CBLGINT
```

```
        :
  (COBOLプログラムの呼び出し)
        :
End Sub
```

注

関数宣言でのエイリアスの名称については，次のとおり指定する必要があります。

```
Private Declare Sub CBLGINT Lib "CBL85RT.DLL"
                            Alias "_CBLGINT@0" ()
```

16.4　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。

- -Main オプションが指定されたプログラムに STOP RUN 文，GOBACK 文のどちらも書かれていない場合で，プログラムの処理が最外側のプログラムの末尾に到達したとき
- 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

GUI モードの場合の終了方法

COBOL プログラムを GUI モードで実行していた場合は，COBOL2002 が出力するコンソールウィンドウのステータスバーに，「実行が終了しました」というメッセージが表示されます。この状態で，コンソールウィンドウのシステムメニューから［閉じる］を選ぶと，コンソールウィンドウが閉じ，実行単位が終了します。

また，環境変数 CBL_BATCH を使用すると，自動的にコンソールウィンドウを閉じて実行単位を終了できます。詳細は，「36.2.3　一般」の「(12)　CBL_BATCH」を参照してください。

なお，コンソールウィンドウが表示されていない場合は，何も出力しないでそのまま実行単位が終了します。

CUI モードの場合，およびコンソールウィンドウが出力されていない場合の終了方法

COBOL プログラムを CUI モードで実行していた場合，およびコンソールウィンドウが出力されていない場合は，実行の終了を示すメッセージを何も出力しないで，そのまま実行単位が終了します。

16.4.1　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

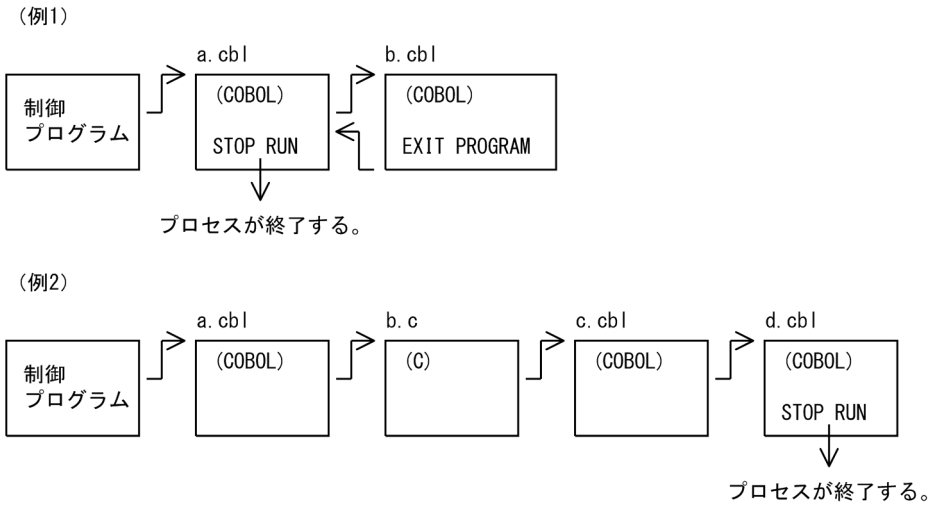
(1)　STOP RUN 文による終了

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



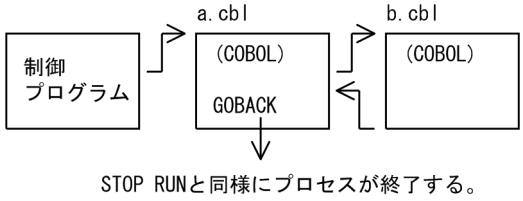
(2)　GOBACK 文による終了

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

```
#include <windows.h>

extern int WINAPI 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，「30.4.2　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 関数や ExitProcess 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。

16.4.2　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，VOS3 COBOL85 との動作と互換性を保つ必要がある場合は，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

終了コードには 1 が設定されます。

CBLABN サービスルーチンが実行された場合

環境変数 CBLABNCODE に YES が指定されている場合は，終了コードには CBLABN サービスルーチンの引数が設定されます。

環境変数 CBLABNCODE に YES 以外が指定されている場合は，終了コードには 1 が設定されます。

上記以外のエラーが発生した場合

- -DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，または-DebugRange オプション指定ありの COBOL プログラムのとき

  終了コードには，1 が設定されます。

  ただし，環境変数 CBLEXCEPT=THROW を指定したときは，システムが返す値が設定されます。
- 上記のコンパイラオプション指定なしの COBOL プログラムのとき

  終了コードにはシステム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例外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) cbltd2k -Execute sample.exe
  - Window 問題レポートでエラー情報を診断した場合

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，「37.4　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。
- Windows(x86) COBOL2002 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。
- Windows(x64) COBOL2002 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※

Windows(x86) COBOL2002 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

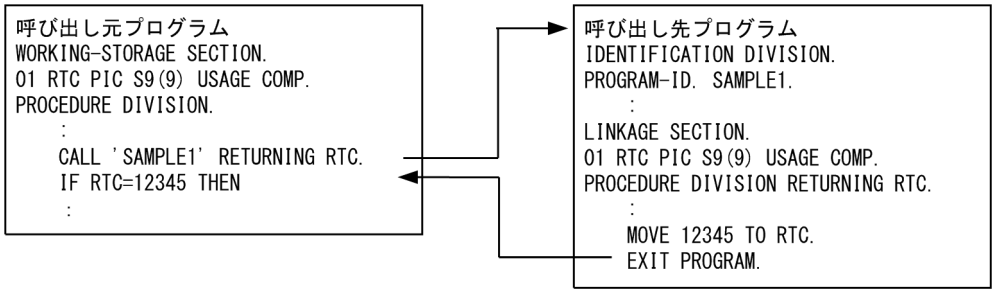
注※

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

- 復帰コードに設定できる値の範囲は，S9(9) COMP の項目について設定できる値の範囲です。ただし，VOS3 COBOL85 との動作と互換性を保つ必要がある場合は，0～4,095 の範囲で設定します。
- 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

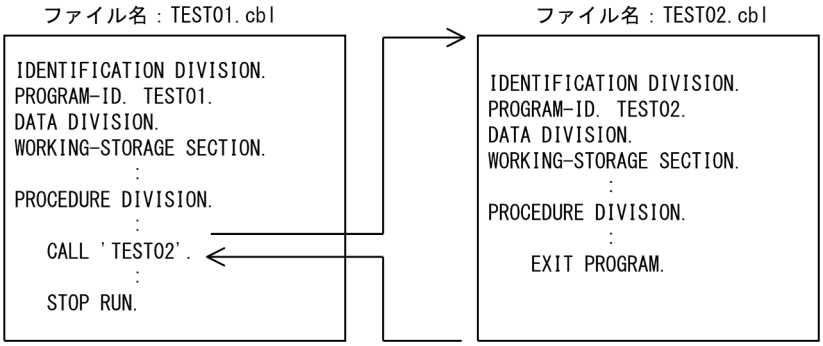
定数指定の CALL 文は，CALL 文に呼び出すプログラム名を定数で指定する方法です。

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。

なお，-DynamicLink,Call オプションを指定した場合は，呼び出すプログラムの解決は，各 CALL 文の最初の実行時に行われます。このため，-DynamicLink,Call を指定した場合，各 CALL 文の最初の実行の性能は，-DynamicLink,Call を指定しない場合と比べて劣化します。

ただし，各 CALL 文の 2 回目以降の実行では，最初の呼び出しで解決したプログラム情報を使用するので，最初の呼び出しと比べると処理時間は短縮されます。

（例）

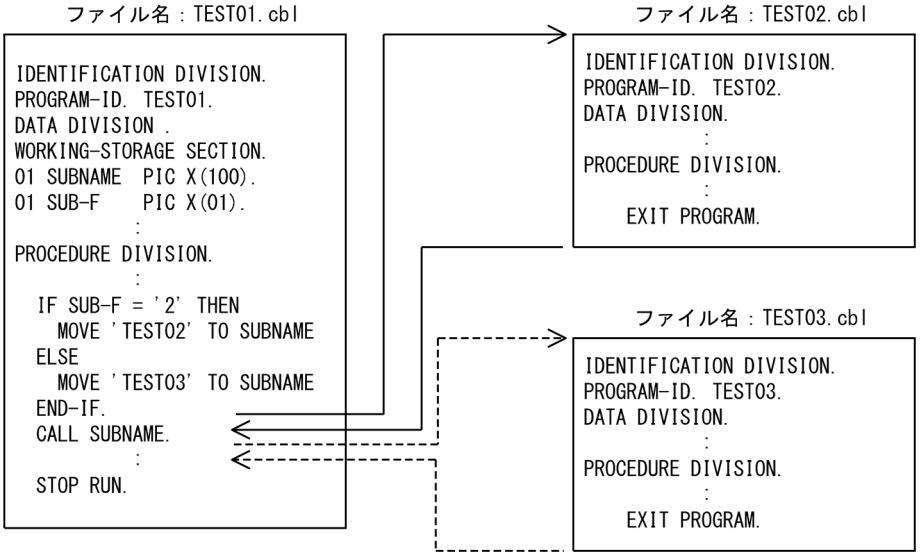


18.1.2 一意名指定の CALL 文

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)



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。

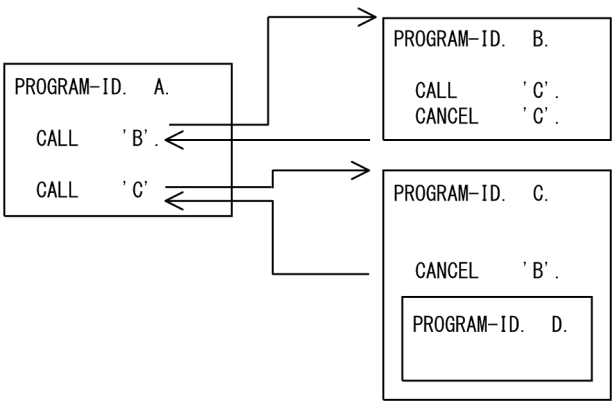
18.2.1　取り消し対象のプログラム

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）

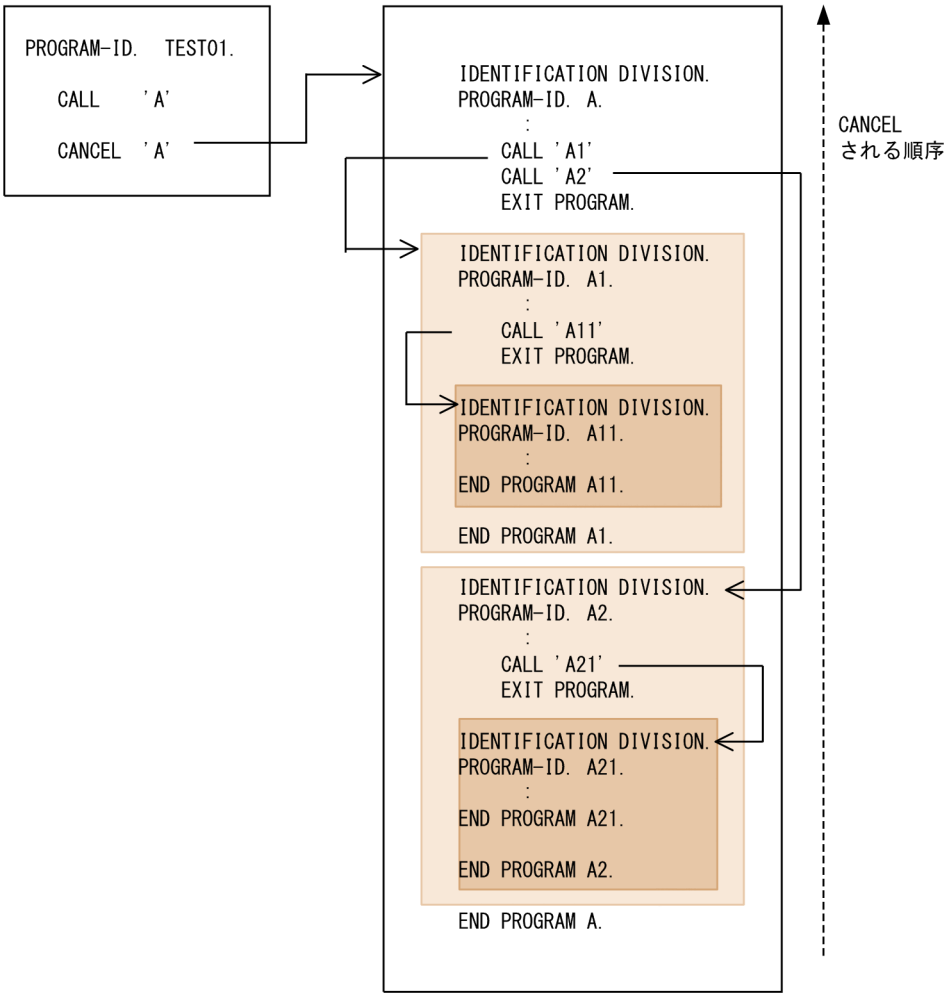


- プログラム A は，自身が呼び出せるプログラム B，およびプログラム C に対して CANCEL 文を実行できます。
- プログラム B，プログラム C がお互いを呼び出せる場合，直接呼び出したプログラムでなくても，プログラム B からプログラム C に対して，プログラム C からプログラム B に対して，それぞれ CANCEL 文を実行できます。
- プログラム D は，プログラム C の内側のプログラムなので，プログラム A からプログラム D に対して CANCEL 文を実行できません。

(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）

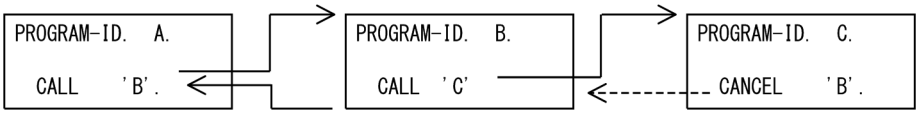


プログラム TEST01 から「CANCEL 'A'.」を実行すると，プログラム A に含まれるすべての内側のプログラムに対しても CANCEL 文が実行されたものとして扱われます。

(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
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)	(2)	不定
画面節（WINDOW SECTION）	(1)	(2)	不定
通信節（画面機能）	−	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

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
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

(説明)

- 作業場所節，報告書節，画面節（SCREEN SECTION），画面節（WINDOW SECTION）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），画面節（WINDOW SECTION），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。
- 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。

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）

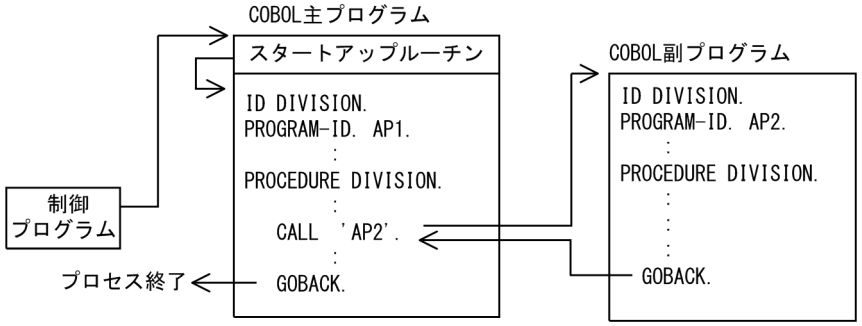
制御プログラムから直接呼ばれる COBOL 主プログラムを作成するには，-Main,System または-Main,V3 オプションを指定してプログラムをコンパイルします。

-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 関数または WinMain 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。

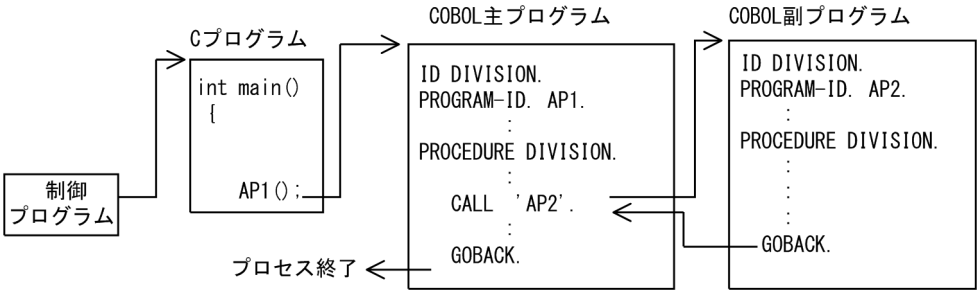


(2)　制御プログラムから間接的に呼ばれる場合

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。

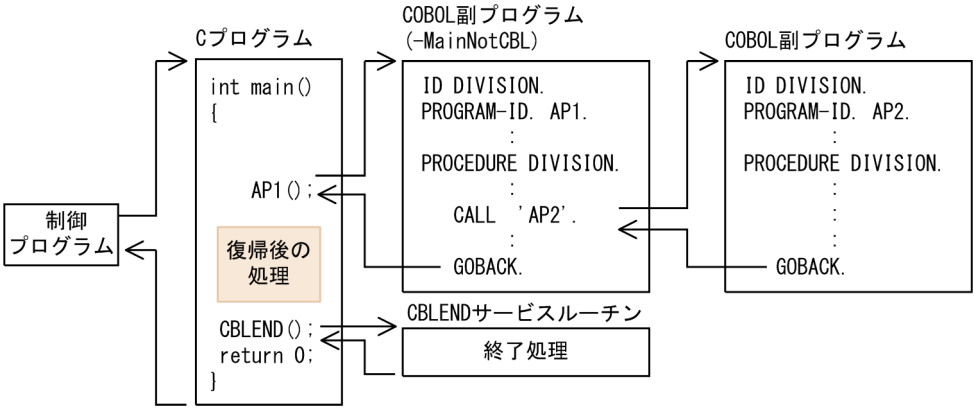
18.3.2　COBOL プログラムを副プログラムとして動作させる場合

「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，「30.4.2　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　プログラム属性と呼び出し規約

ここでは，COBOL プログラムのプログラム属性と，呼び出し規約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），画面節（WINDOW SECTION），通信節（画面機能），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），画面節（WINDOW SECTION），ファイル節，サブスキーマ節，通信節（画面機能），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

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)	不定	不定
画面節（WINDOW SECTION）	(1)	(2)	不定
通信節（画面機能）	−	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

(説明)

- 作業場所節，報告書節，画面節（SCREEN SECTION），画面節（WINDOW SECTION）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。

- 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。

なお，-DllInit オプションを指定して作成した DLL 中のプログラムは，INITIAL 句の指定がない場合でも初期化属性プログラムとなります。

(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)

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
報告書節	(1)	(3)	(4)
画面節（SCREEN SECTION）	(1)	(4)	(4)
画面節（WINDOW SECTION）	(1)	(3)	(4)
通信節（画面機能）	−	(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

（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），画面節（WINDOW SECTION）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），画面節（WINDOW SECTION），通信節（画面機能），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。

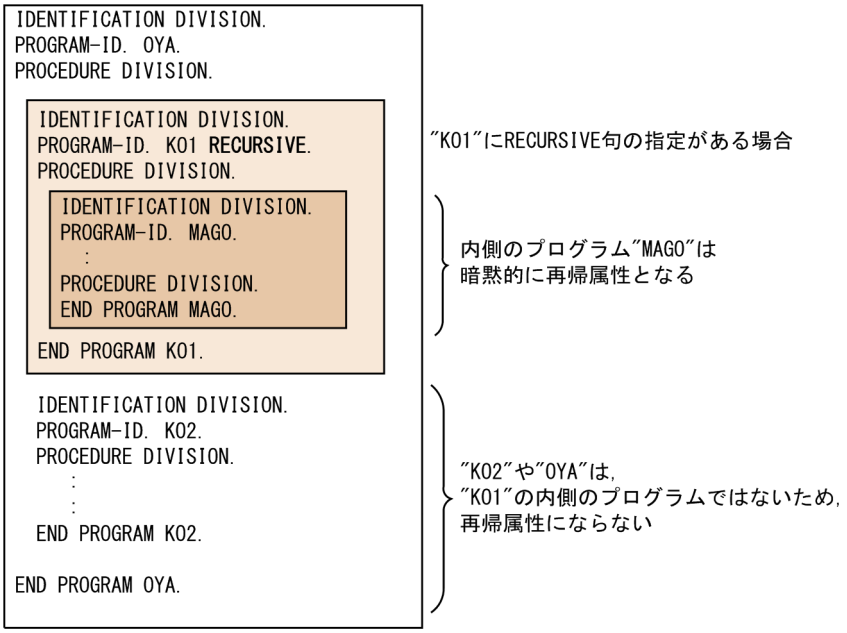
- 画面節（SCREEN SECTION），画面節（WINDOW SECTION），通信節（画面機能），通信節（データコミュニケーション機能），サブスキーマ節で使用された資源の状態は，そのままの状態で引き継がれます。

(a) RECURSIVE 句の効果

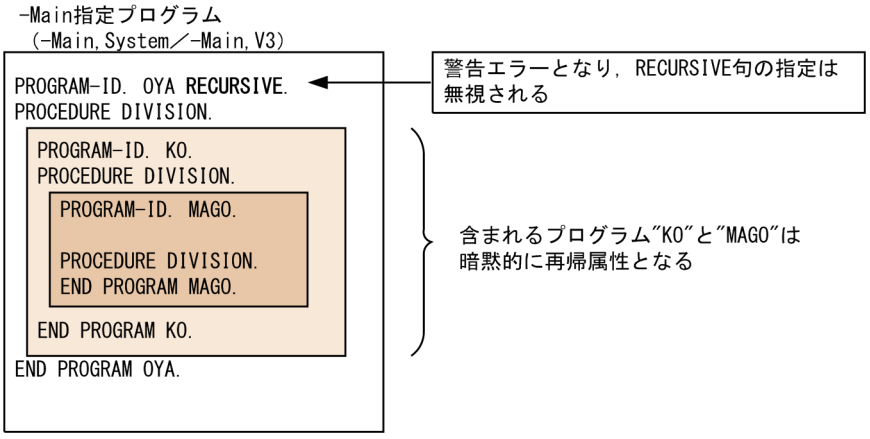
RECURSIVE 句を指定した場合の効果は，次のとおりです。

- 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。



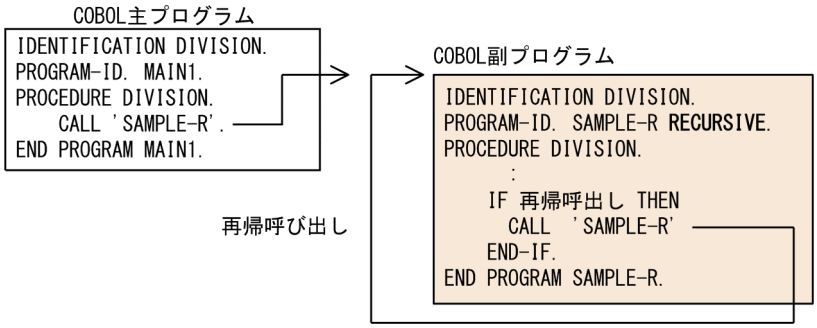
  上記の最外側のプログラム（"OYA"）を呼び出そうとした場合，次の結果となります。

- 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。

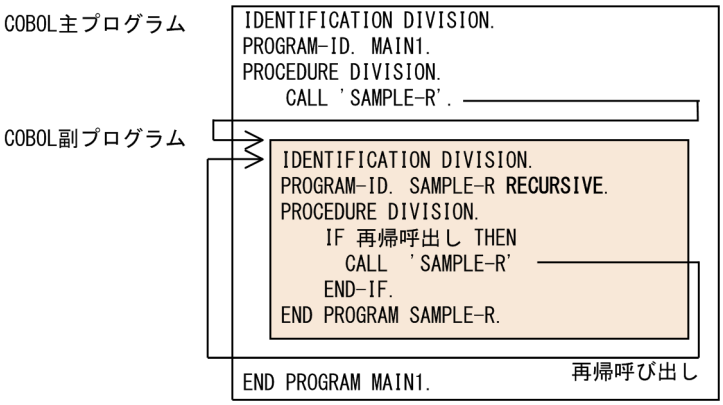
(b) 再帰呼び出しの例

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）

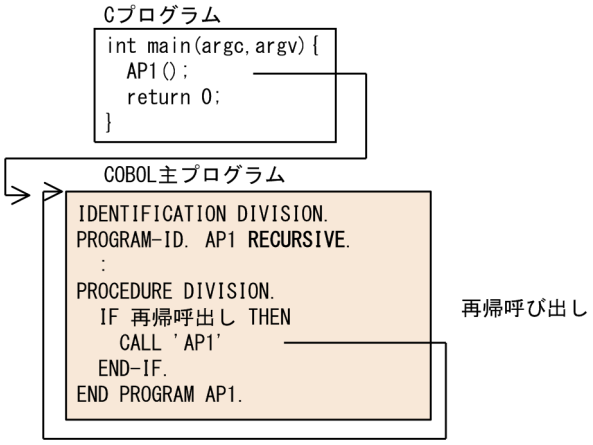


（内側のプログラムの再帰呼び出し）



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。

18.4.2　呼び出し規約

COBOL プログラムでは，以下の呼び出し規約のプログラムを作成できます。

Windows(x86) COBOL2002 の場合

- cdecl
- stdcall

Windows(x64) COBOL2002 の場合

- fastcall

呼び出し規約を指定することによって，C などの他言語で作成したプログラムから COBOL プログラムを呼び出したり，COBOL プログラムから他言語のプログラムを呼び出せます。

Windows(x86) COBOL2002 の場合，呼び出し規約は，呼び出すプログラムで，環境部の EXTERNAL-PROGRAM SECTION の CALL-CONVENTION 段落に指定します。呼び出し規約の指定方法については，マニュアル「COBOL2002　言語　拡張仕様編」「25.2.1　プログラム間連絡機能の環境部」を参照してください。

(1)　cdecl 呼び出し規約（Windows(x86) COBOL2002 で有効）

cdecl 呼び出し規約は，C プログラムの cdecl 呼び出し規約のインタフェースを持つプログラムを呼び出すときに使用します。DLL でないプログラムは，すべて cdecl 呼び出し規約となります。

cdecl 呼び出し規約のプログラムを呼ぶ場合は，呼び出す COBOL プログラムの CALL-CONVENTION 段落で，呼び出し先プログラムと CDECL を関連づけます。なお，CALL-CONVENTION 段落の指定がない場合は，CDECL が仮定されます。

(2)　stdcall 呼び出し規約（Windows(x86) COBOL2002 で有効）

stdcall 呼び出し規約は，C プログラムの stdcall 呼び出し規約と同じインタフェースを持つプログラムを呼び出すときに使用します。

stdcall 呼び出し規約のプログラムを呼ぶ場合は，呼び出す COBOL プログラムの CALL-CONVENTION 段落で，呼び出し先プログラムと STDCALL を関連づけます。

CALL-CONVENTION 段落の指定がない場合でも，-StdCall オプションを指定すると，呼び出すプログラムの呼び出し規約を stdcall にできます。詳細は，「35.　実行可能ファイルと DLL の作成」を参照してください。

(3)　fastcall 呼び出し規約（Windows(x64) COBOL2002 で有効）

fastcall 呼び出し規約は，C プログラムの fastcall 呼び出し規約と同じインタフェースを持つプログラムを呼び出すときに使用します。

Windows(x64) COBOL2002 では，fastcall 呼び出し規約のプログラムだけ作成できます。

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，「35.　実行可能ファイルと DLL の作成」を参照してください。また，DLL の静的なリンクと動的なリンクについては，「18.6.2　DLL 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
- DLL のインポートライブラリを使用して，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　DLL に含まれるプログラムの呼び出し

18.6.1　DLL 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 DLL（ダイナミックリンクライブラリ）といいます。

呼び出し先プログラムを DLL 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 DLL 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，DLL 中の関数を変更しても実行可能ファイルやほかの DLL の再コンパイルや再リンクは不要です。

18.6.2　DLL に含まれるプログラムの呼び出し方法

DLL 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インポート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，DLL もロードされます。

DLL 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 DLL を呼び出します。

ロード対象の DLL は次の DLL 検索フォルダ下から順に検索されます。

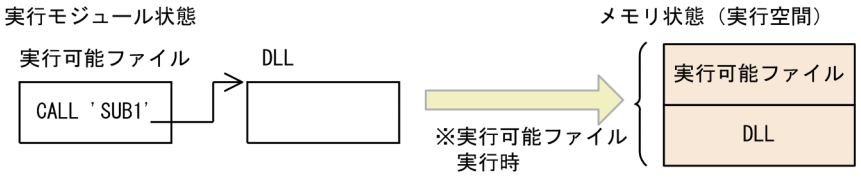
1. 現在のプロセスの実行可能ファイルがあるフォルダ
2. Windows のシステムフォルダ
3. Windows フォルダ
4. カレントフォルダ
5. 環境変数 PATH で登録されているフォルダ

プログラム実行時，これらの検索パスに DLL がないと，プロセス起動エラーとなります。

-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 DLL が CALL '定数'で呼ばれ，実行可能ファイルの作成時にインポートライブラリを指定して DLL をリンクした場合，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，DLL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。

DLL を動的にリンクするには，次のどれかの文を実行し DLL に含まれるプログラムを呼び出すか，またはそのアドレスを求めます。

- 一意名指定の CALL 文
- -DynamicLink,Call オプション指定時の定数指定の CALL 文
- -DynamicLink,Call オプション指定時の ADDR 関数

動的なリンクの場合，プログラム実行時に環境変数 CBLLDLL で呼び出し先プログラムが含まれる DLL を指定します。

形式

```
CBLLDLL=DLL名称［;DLL名称］…
```

DLL 名称

動的にリンクする DLL の名称を指定します。

規則

DLL 名称の指定方法には，DLL のファイル名称だけを指定する方法と，フォルダを含めたファイル名称を指定する方法があります。

ファイル名称だけを指定した場合，動的なリンクの対象となる DLL は，次の DLL 検索フォルダにある指定名称の DLL から順に検索されます。

1. 現在のプロセスの実行可能ファイルがあるフォルダ
2. Windows のシステムフォルダ
3. Windows フォルダ
4. カレントフォルダ
5. 環境変数 PATH で登録されているフォルダ

フォルダを含めたファイル名称を指定した場合，指定したフォルダの DLL だけが動的なリンクの対象となります。

DLL 自動ロード機能

DLL 自動ロード機能は，環境変数 CBLLDLL の指定でプログラムが見つからなかった場合に，プログラム名.dll という名称の DLL をロードして，その DLL 中のプログラムを実行できるようにする機能です。

DLL 自動ロード機能で一度ロードすれば，それ以降は DLL 中のほかのプログラムを CALL 文で呼び出せます。

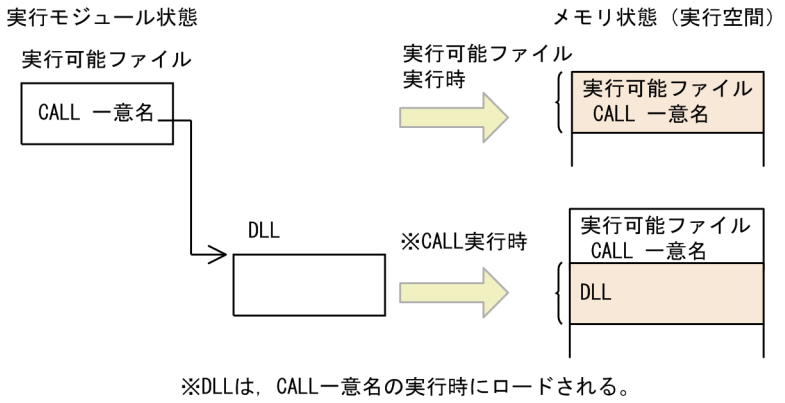
DLL 自動ロード機能を使用するためには，実行時環境変数 CBLLPROGDLL に YES を指定します。環境変数の設定方法については，「36.2.1　実行時環境変数の設定方法」を参照してください。

環境変数 CBLLDLL で指定した DLL に呼び出し先プログラムがない場合，CALL 文で実行時エラーとなります。また，DLL 自動ロード機能を使用する場合は，「呼び出し先プログラムの名称.dll」という名称の DLL がないとき，または「呼び出し先プログラムの名称.dll」に呼び出し先プログラムが含まれないときも，CALL 文で実行時エラーとなります。

なお，動的なリンクのプレロード機能を使用することで，プログラム起動時に検索対象となる DLL をロードできます。動的なリンクのプレロード機能の詳細は，「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。

（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。



※DLL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

- プログラムを呼び出す場合，呼び出し先プログラムの呼び出し規約に従って呼び出さなければなりません。呼び出し規約が異なっている場合，静的なリンクのときは，リンク時にエラーとなり，動的なリンクのときは，実行時エラーとなります。
- 動的にロードされた DLL 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 DLL に含まれる関数であってはなりません。

- Windows(x86) COBOL2002 では，呼び出し先プログラムが STDCALL 呼び出し規約の場合，CALL 文に指定されたプログラム名に装飾名を付けて呼び出されます。COBOL 以外のプログラムを STDCALL 呼び出し規約にする場合は，プログラム名に装飾名を付けてエクスポートしなければ動的なリンクでは呼び出されません。詳細は，「35.2.1　DLL の作成」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 DLL 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。

- 利用者定義関数を含めた DLL 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，「35.　実行可能ファイルと DLL 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

ダイナミックリンクでは，独自に DLL 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，DLL のロードやアンロードに関連するシステム関数（LoadLibrary，FreeLibrary）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 DLL 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 DLL 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 DLL 名称を記述します。

- 1 行に一つの DLL 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。
- 拡張子が「.DLL」でないファイル名は無視されます。
- 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 DLL 名称の先頭まで • DLL 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	DLL 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DLL
C:¥tmp¥test01.dll
```

規則

- プレロードする DLL 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 DLL 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 DLL 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 DLL 内のプログラムは，環境変数 CBLLDLL や CBLLPROGDLL 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 DLL 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 DLL の検索情報
- プレロード時の DLL 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,Call オプションの指定がない 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
	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）

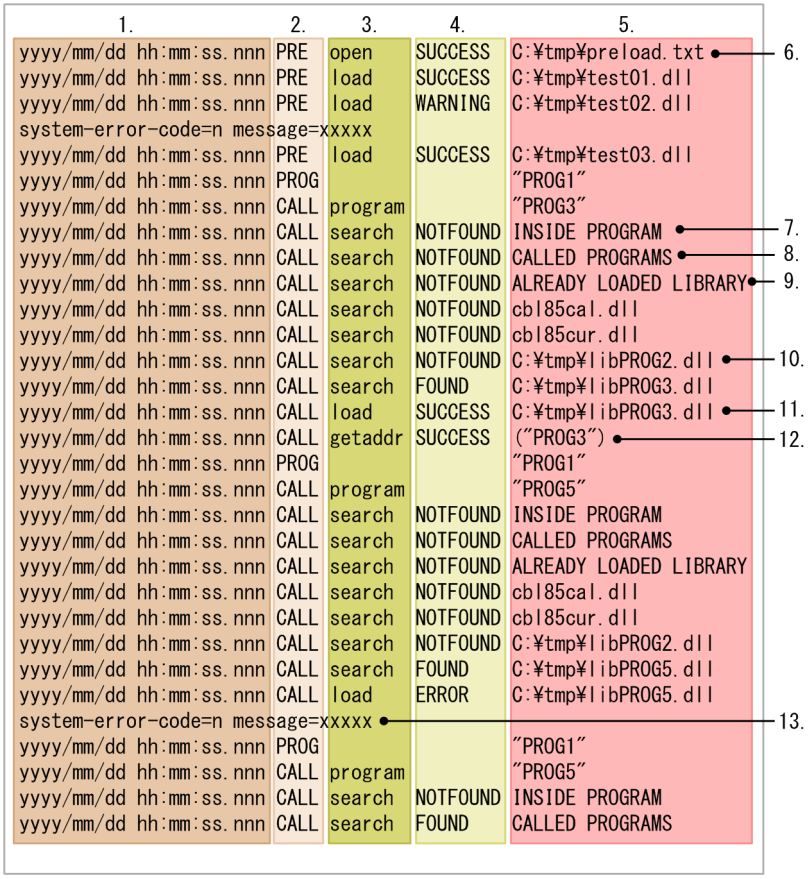
－：条件なし

(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。
- 環境変数 CBLLDLL にパスを含む DLL 名を指定している場合，プログラム検索トレースの DLL 名には指定した値がそのまま出力されます。
- 環境変数 CBLLDLL にパスがない DLL 名を指定している，または環境変数 CBLLPROGDLL を指定している場合は，プログラム検索トレースの DLL 名には検索対象となる DLL 名の絶対パスが出力されます。ただし，検索対象の DLL が見つからなかった場合は，次の情報が出力されます。

- 環境変数 CBLLDLL：パス指定のない DLL 名
- 環境変数 CBLLPROGDLL：呼び出し先プログラム名

(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 DLL 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	DLL 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 DLL 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 DLL 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。
   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. すでにロードされている DLL 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。
10. 未ロードの DLL から呼び出し先プログラムを検索した結果を出力する。
11. 呼び出し先プログラムを含む DLL をロードした結果を出力する。
12. 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。
13. DLL 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。

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（.exe）やバッチファイル（.bat）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.exe），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に空白が含まれている場合は，ファイルを呼び出すときにファイル名を引用符（ " ）で囲んでください。引用符で囲まないと，異なったファイルを呼び出してしまうことがあります。以下の例を参考にしてください。

（例）

「PROG.1 ZIKKOU.exe」と「PROG.1」という二つのファイルがあった場合

```
・CALL ’PROG.1 ZIKKOU.exe’
    →　「PROG.1」が呼び出される。

・CALL ’ ”PROG.1 ZIKKOU.exe”’
    →　「PROG.1 ZIKKOU.exe」が呼び出される。
```

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フォルダ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。

- フォルダを含めたパス名で指定する。
- フォルダ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。

(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，指定したフォルダ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，次の順序でファイルが検索され，最初に見つかったファイルが実行されます。

1. 呼び出し元のプログラムがあるフォルダ
2. カレントフォルダ
3. Windows のシステムフォルダ
4. Windows フォルダ
5. 環境変数 PATH 内に登録されているフォルダ

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.exe 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.exe' USING PARM.
```

この場合，「A.exe AAA」という形で A.exe 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渡しても，一つの一意名にすべてのパラメタを設定して渡してもかまいません。

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。

(2)　実行可能ファイル名に含めて渡す方法

実行可能ファイル名に利用者引数を含めたコマンド列として利用者引数を渡すこともできます。

（例）

```
        CALL 'A.exe AAA'.
```

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。

詳細は，「17.2　復帰コードと返却項目」を参照してください。

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) 構造化例外処理使用時の注意事項

構造化例外処理を使用して COBOL プログラム実行中の例外を処理する場合の注意事項を次に示します。

- システム例外発生後，__except 節のフィルタ式以外の個所で COBOL プログラムや COBOL のサービスルーチンを呼び出すことはできません。
- __except 節のフィルタ式中では，COBOL プログラムを呼び出せます。また，この延長で CBLABN サービスルーチンを呼び出して，プログラムを終了できます。

(4) C プログラムでの 64bit アプリケーションの作成（Windows(x64) COBOL2002 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，ポインタ型のサイズは 8 バイトになります。

(5) 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。

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 GUI モードの COBOL プログラムを呼び出す場合，最初の COBOL プログラムを呼び出す前に，CBLGINT サービスルーチンを呼び出す必要があります。詳細は，「16.3　COBOL プログラムのモード」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，「16.4 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。
- C プログラムから COBOL プログラムを呼び出す場合，呼び出し規則を合わせる必要があります。Windows(x86) COBOL2002 の場合，-Dll オプションで Cdecl サブオプションを指定した場合，COBOL プログラムに対する C プログラムのプロトタイプ宣言は，cdecl でなければなりません。また，-Dll オプションで Stdcall サブオプションを指定した場合，COBOL プログラムに対する C プログラムのプロトタイプ宣言は，stdcall でなければなりません。
- 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。

- C プログラムから COBOL プログラムを呼ぶ場合は，C 実行時ライブラリに，マルチスレッドに対応したダイナミックリンクライブラリを使用する必要があります。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。
- 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	_int64	unsigned _int64	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	○※4	○※5	○※5	○※6	○※6	○※11	○※11	×	×		
COMP-X	○※7	○※7	○※8	○※8	○※9	○※9	○※12	○※12	×	×		
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									×	×		

COBOL	C											
	char	unsigned char	short	unsigned short	int, long	unsigned int, unsigned long	_int64	unsigned _int64	float	double	ポインタ	構造体
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					○				×	×		
浮動小数点数字定数	×	×	×	×	×	×	×	×	×	○	×	×
ZERO					○				×	×		
NULL									×	×	○	
オブジェクト参照項目									×	×		
OLE オブジェクト参照データ項目									×	×		
バリアントデータ項目									×	×		
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。

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　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　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　10～18 けたの整数です。

注※12　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。

(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。
- COBOL プログラムから呼び出される C プログラムの呼び出し規則が stdcall の場合，次のどちらかをしておく必要があります。
  1. 呼び出す COBOL プログラムを-StdCall オプションを指定してコンパイルし，stdcall 呼び出し規約のプログラム名称を記述した.cbw ファイルを作成する。
  2. EXTERNAL-PROGRAM SECTION の CALL-CONVENTION 段落で，プログラムを stdcall 呼び出し規約にしておく。

- 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。
  - DLL 中の C プログラムを呼ぶ場合，DLL 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，DLL 作成時にモジュール定義ファイル（.def）を使用する場合は，EXPORTS 文で関数に対して NONAME 指定をしてはなりません。NONAME を指定したプログラムは CALL 一意名では呼び出せません。
  - 実行可能ファイル（.exe）と静的にリンクしたオブジェクト中の C プログラムを呼ぶ場合，実行可能ファイル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，C プログラムは，次のように記憶クラス属性として dllexport を指定しておく必要があります。dllexport を指定していない C プログラムを静的にリンクした場合，CALL 一意名では呼び出せません。

  （例）

  ```
  __declspec( dllexport ) void sub();
  ```

  - C プログラムの属性を__stdcall とする場合は，記憶クラス属性として dllexport を指定し，リンク時はモジュール定義ファイル（.def）を指定してはいけません。dllexport 指定がないか，またはモジュール定義ファイルの EXPORTS 文でプログラム名が指定されている場合，そのプログラムは CALL 一意名では呼び出せません。
  - C プログラムの属性を__cdecl とする場合，記憶クラスとして dllexport を指定するか，またはモジュール定義ファイルの EXPORTS 文でプログラム名を指定しなければなりません。

(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
	−	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

C プログラムのデータ型	対応する COBOL のデータ項目の型		
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）	9(5) COMP ～ 9(9) COMP 9(5) COMP-X ～ 9(9) COMP-X
long	−	符号付き 2 進項目（5～9 けた）	S9(5) COMP ～ S9(9) COMP
unsigned _int64	−	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
_int64	−	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※	−	集団項目	−

（凡例）

−：該当しない

注※

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。

(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.
```

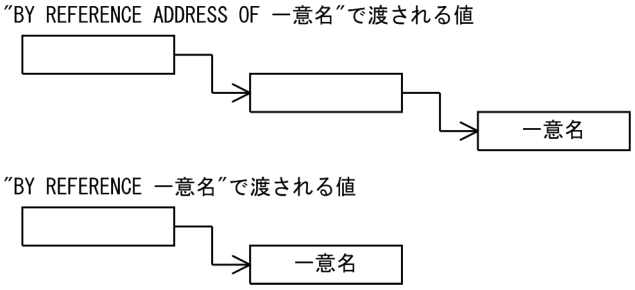
C 副プログラム

```
#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Windows(x86)の場合   */
                    /* Windows(x64)の場合は _int64 *      */
     )
 {
   printf("length(p)     =(%d)¥n",*p_ptr_cl);   … 3.
                    /* この書式指定はWindows(x86)の場合   */
```

```
                    /* Windows(x64)の場合は %I64d          */
    :
}
int SAMPLE5(
    int p_ptr_vl    /* この引数の型はWindows(x86)の場合    */
                    /* Windows(x64)の場合は _int64         */
    )
{
  printf("length(p)     =(%d)¥n",p_ptr_vl);    … 4.
                    /* この書式指定はWindows(x86)の場合    */
                    /* Windows(x64)の場合は %I64d          */
    :
}
```

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。

注※

Microsoft C／C++の関数

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，「35.　実行可能ファイルと DLL の作成」を参照してください。

19.2　Visual Basic との連携（Windows(x86) COBOL2002 で有効）

Visual Basic プログラムから，DLL として作成された COBOL プログラムを呼び出せます。

また，Visual Basic プログラムで作成した DLL（インプロセスサーバ）を COBOL プログラムから呼び出せます。この場合，COBOL2002 の OLE2 オートメーションクライアント機能を使用してください。詳細は，「25.　OLE2 オートメーション機能」を参照してください。

ここでは，COBOL プログラムと Visual Basic 6.0 プログラムの連携方法について説明します。

19.2.1　Visual Basic から COBOL を呼び出すときの規則

Visual Basic プログラムから COBOL プログラムを呼ぶときの規則を，次に示します。

- Visual Basic プログラムは，COBOL プログラムの RETURN-CODE 特殊レジスタで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.　プログラム間の引数と返却項目」を参照してください。
- Visual Basic プログラムから GUI モードの COBOL プログラムを呼び出す場合は，最初の COBOL プログラムを呼び出す前に，CBLGINT サービスルーチンを呼び出す必要があります。詳細は，「16.3 COBOL プログラムのモード」を参照してください。
- Visual Basic 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，「16.4　COBOL 実行単位の終了」を参照してください。
- Visual Basic プログラムから COBOL プログラムを呼び出す場合，DLL の関数宣言で呼び出し規則に合わせたエイリアスを指定する必要があります。

  また，呼び出される COBOL プログラムは，stdcall 呼び出し規約でなければなりません。

エイリアス名称

```
_プログラム名@n
```

n

次の計算式で求められる値を指定します。

n ＝（引数の数×4）＋

（値渡しで受け取る倍精度浮動小数点数字項目引数の数×4）

（例）

プログラム名が SAMPLE1 で，2 進項目と倍精度浮動小数点数字項目の引数を値渡しにする場合の関数宣言

```
Private Declare Sub SAMPLE1 Lib "COB1.DLL"
  Alias "_SAMPLE1@12" (ByVal A As Long, ByVal B As Double)
```

- 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 Visual Basic 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 long 型にしなければなりません。
- Visual Basic 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。
- Visual Basic から呼び出される COBOL プログラムは，stdcall 呼び出し規約でなければなりません。cdecl 呼び出し規約のプログラムは，呼び出せません。

19.2.2 COBOL プログラムのデータ型と Visual Basic での宣言

整数値，文字列，およびユーザ定義型について，Visual Basic と COBOL のデータ型の対応を，次に示します。

表 19-2 Visual Basic と COBOL のデータ型の対応

データ型	Visual Basic （送り出し側作用対象）	COBOL （受け取り側作用対象）
バイト型	`Dim a As Byte`	`01 a PIC 9(1) COMP-X`
整数型	`Dim a As Integer`	`01 a PIC S9(4) COMP`
長整数型	`Dim a As Long`	`01 a PIC S9(9) COMP`
可変長文字列	`Dim a As String`	`01 a PIC X(n)`
固定長文字列	`Dim a As String * 10`	`01 a PIC X(n)`

ユーザ定義型	`Type UserDef`
	`  x As Integer`
	`End Type`
	`Dim a As UserDef`
		`01 a`
		`  02 x PIC 9(1) COMP-X`

（凡例）

n：文字列の長さを示す

バイト型，整数型，長整数型の場合

引数を渡す方法として，値渡し（ByVal）と参照渡し（ByRef）を使用できます。

- 値渡しを使用する場合，COBOL プログラムで値を変更しても，Visual Basic のデータ項目の内容には影響を与えません。
- 参照渡しを使用する場合，COBOL プログラムで値を変更すると，Visual Basic のデータ項目の内容も変更されます。

可変長文字列，固定長文字列の場合

引数を渡す方法として，値渡し（ByVal）を使用できません。Visual Basic と COBOL での引数の属性は，次の組み合わせとなります。

Visual Basic 値渡し（ByVal） → COBOL 参照渡し（BY REFERENCE）

ユーザ定義型の場合

ユーザ定義型は，COBOL の集団項目に対応します。ただし，ユーザ定義型に可変長文字列が含まれる場合，アドレスデータ項目を使用して，ポインタ処理をしなければなりません。

19.2.3 引数の受け渡し

Visual Basic 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。

(1) 整数型，ユーザ定義型のデータ項目を渡す例

Visual Basic プログラム

```
Private Declare Sub COBSUB Lib "SAMPLE1" _
                    Alias "_SAMPLE1@12" _
                    (ByVal a As Integer, _
                     ByRef a As Integer, _
                     ByRef a As UserDef)
Private Declare Sub CBLGINT Lib "CBL85RT" Alias "_CBLGINT@0" ()
Private Declare Sub CBLEND Lib "CBL85RT" Alias "_CBLEND@0" ()

Dim PARM1 As Integer
Dim PARM2 As Integer

Private Type UserDef
  U_INT1 As Integer
  U_INT2 As Integer
End Type
Dim PARM3 As UserDef

Private Sub Command1_Click()
             :
    Call SAMPLE1(PARM1, PARM2, PARM3)
             :
End Sub

Private Sub Form_Load()
           :
   Call CBLGINT
           :
End Sub

Private Sub Form_Unload(Cancel As Integer)
           :
    Call CBLEND
           :
End Sub
```

COBOL プログラム

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       LINKAGE SECTION.
       01 L-PARM1  PIC S9(04) USAGE COMP.
       01 L-PARM2  PIC S9(04) USAGE COMP.
       01 L-PARM3.
         02 U-INT1 PIC S9(04) USAGE COMP.
         02 U-INT2 PIC S9(04) USAGE COMP.
       PROCEDURE DIVISION USING BY VALUE     L-PARM1
                                BY REFERENCE L-PARM2
                                BY REFERENCE L-PARM3.
           ADD 100 TO L-PARM1.
           ADD 100 TO L-PARM2.
           ADD 100 TO U-INT1.
           ADD 100 TO U-INT2.

           EXIT PROGRAM.
```

- Visual Basic プログラムから COBOL プログラムに引数が値渡しで渡された場合，COBOL プログラム内で受け取る引数には，BY VALUE を指定しなければなりません。また，参照渡しで渡される場合，COBOL プログラム内で受け取る引数には，BY REFERENCE を指定しなければなりません。
- COBOL プログラムが Visual Basic プログラムから数字項目を受け取る場合は，内部・外部 10 進項目，内部・外部浮動小数点項目などの数字属性を意識する必要があります。

19.3 Java との連携

19.3.1 概要

Java プログラムから COBOL プログラムを呼び出せます。ここでは，Java プログラムと COBOL プログラムの連携の手順について説明します。

Java アプリケーションサーバを利用して既存の COBOL プログラムを使った Web 環境を構築する場合は，Cosminexus 連携機能，および TP1 サーバ用 Cosminexus 連携機能を使用します。これらの機能を使用すると，Java 言語の他言語インタフェース機能（Java Native Interface）処理とデータ交換処理をラッピングする Java クラスを自動生成できるため，開発期間を短縮できます。また，COBOL プログラムで発生したエラー情報などを取得する例外メソッドを用意しているので，この API を使って例外処理を備えた実行環境を構築できます。

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

TP1 サーバ用 Cosminexus 連携機能を使用するには，次の製品が必要です。

Java から TP1 環境下の COBOL SPP（Service Providing Program）のサービスを利用する場合（TP1 サーバ用 Cosminexus 連携機能）に必要な製品

開発環境：

TP1/COBOL adapter for Cosminexus Version 2

サーバ実行環境：

TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2

TP1 サーバ用 Cosminexus 連携機能の詳細については，マニュアル「TP1/COBOL adapter for Cosminexus　ユーザーズガイド」を参照してください。

なお，COBOL プログラムから Java プログラムを直接呼び出せません。Java プログラム呼び出し機能を使用してください。

Java プログラム呼び出し機能の詳細については，マニュアル「COBOL2002　Java プログラム呼び出し機能ガイド」を参照してください。

(1) 概要

Java プログラムから COBOL プログラムを呼び出すには，Java 言語の他言語インタフェース機能（Java Native Interface）を利用します。したがって，Java 言語の他言語インタフェース機能の制限に基づいて，Java から呼び出される COBOL プログラムは stdcall 呼び出し規約の DLL 形式で作成する必要があります。

また，Java プログラムから COBOL プログラムを呼び出す場合には，COBOL プログラムの初期化，および終了処理するサービスルーチン（CBLGINT／CBLEND）を呼び出す必要があります。Java プログラムからは直接サービスルーチンを呼び出せないため，サービスルーチンを呼び出す C プログラムを作成し，Java プログラムからその C プログラムを呼び出すようにします。

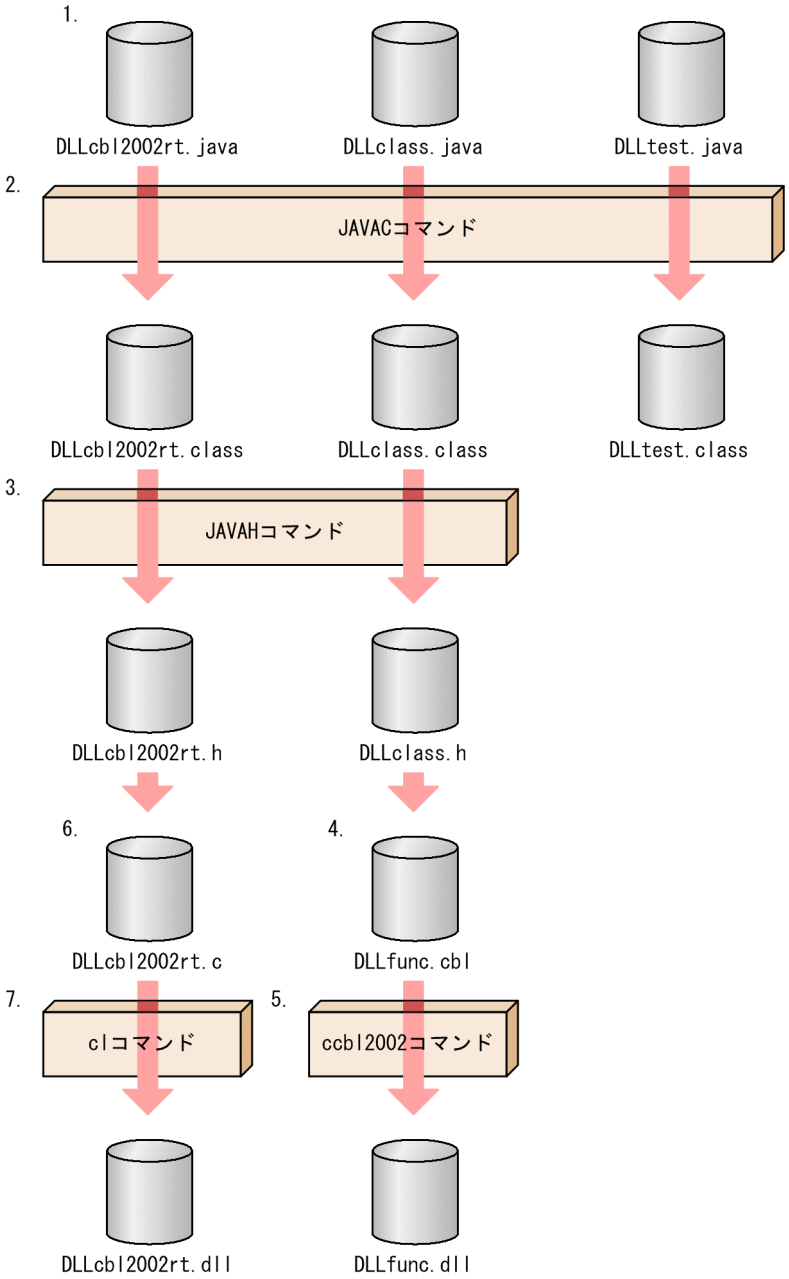
注意事項

Java プログラムとのインタフェースは，Sun Microsystems, Inc の Java Development Kit（JDK）のバージョン 1.1 以降を対象とします。

(2) Java プログラムとのインタフェースを使ったプログラムの作成手順

Java プログラムとのインタフェースを使ったプログラムの作成手順を次に示します。

図 19-1　Java プログラムとのインタフェースを使ったプログラムの作成手順



1. Java プログラムのソースファイルを作成する。

2. Java ソースファイルをコンパイルして Java バイトコードを生成する。

3. Java バイトコードから C 言語ヘッダファイルを生成する。

4. COBOL プログラムのソースファイルを作成する。

5. COBOL ソースファイルをコンパイルして COBOL プログラム（DLL 形式）を生成する。

6. サービスルーチン（CBLGINT／CBLEND）を呼び出す C プログラムのソースファイルを作成する。

7. C ソースファイルをコンパイルして DLL を生成する。

8. Java プログラムを実行する。

なおこの節では，次のような構成で Java プログラムとのインタフェースの利用手順を説明します。

Java 言語での作業

次の作業について説明します。

- Java 言語の main メソッドを含む Java プログラムの作成
- COBOL プログラムを呼び出す Java プログラムの作成
- C プログラムを呼び出す Java プログラムの作成

COBOL 言語での作業

Java プログラムから呼び出す COBOL プログラムの作成方法について説明します。

C 言語での作業

COBOL プログラムの初期化・終了処理を呼び出す C プログラムの作成方法について説明します。

Java プログラムの実行

作成した Java プログラムの実行方法について説明します。

制限事項

Java プログラムとのインタフェースでの制限事項について説明します。

19.3.2 Java プログラムから COBOL プログラムを呼び出す方法

(1) Java 言語での作業

(a) Java ソースプログラムの作成

COBOL プログラムを呼び出す Java プログラムでは，次の三つの Java ソースファイルを作成する必要があります。

- main メソッドを含む Java プログラム（Java クラス）
- COBOL プログラムに対応する Java プログラム（Java クラス）
- COBOL プログラムの初期化・終了処理をする C プログラムに対応する Java プログラム（Java クラス）

main メソッドを含む Java プログラム（Java クラス）

Java アプリケーションは，「static」属性と「public」属性を持つ「main」という名前のメソッドから実行が開始されます。そのため，この「main」メソッドを含む Java プログラム（Java クラス）が必要です。

COBOL プログラムの呼び出し，およびサービスルーチンの呼び出しを含む「main」メソッドを含むクラス「DLLtest」を記述した Java ソースファイル「DLLtest.java」の例を次に示します。

Java ソースファイルの例（DLLtest.java）

```
public class DLLtest
{
  public static void main (String args[])
  {
   // サービスルーチンを呼び出す
   // Cプログラムに対応するクラスのインスタンスを生成
   DLLcbl2002rt aDLLcbl2002rt = new DLLcbl2002rt ();
   // 呼び出すCOBOLプログラムに対応する
   //クラスのインスタンスを生成
   DLLclass aDLLclass = new DLLclass ();
   // COBOLプログラムの実行環境初期化処理をする
   //サービスルーチンを呼び出す
   aDLLcbl2002rt.cblgint ();
   // COBOLプログラムの呼び出し
   int result = aDLLclass.DLLcobol (10, 20);

   // COBOLプログラムの実行環境終了処理をする
   // サービスルーチンを呼び出す
    aDLLcbl2002rt.cblend ();
   // Javaプログラムの終了
   System.exit (0);
 }
}
```

COBOL プログラムに対応する Java プログラム（Java クラス）

Java プログラムから他言語プログラムを呼び出す場合には，他言語プログラムに対応する Java プログラム（Java クラス）を作成しなければなりません。そのため，呼び出される COBOL プログラムに対応する Java プログラム（Java クラス）が必要です。

Java プログラムから呼び出す，他言語プログラムに対応する Java プログラム（Java クラス）には，呼び出す他言語プログラムに対応する「native」属性を持つ Java メソッドを定義します。また，他言語プログラムを含む DLL をロードする static 命令も含んでいる必要があります。

COBOL プログラムに対応する Java プログラム（Java クラス）の例と，この例で仮定している名称を次に示します。

Java ソースファイル（DLLclass.java）

```
class DLLclass {
 // 呼び出されるCOBOLプログラムに対応する
 // メソッドプロトタイプを定義する
 public native int DLLcobol (int arg1, int arg2);
 // 呼び出されるCOBOLプログラムを含むDLLをロードする
 static {
   System.loadLibrary (”DLLfunc”);
 }
}
```

表 19-3　例で仮定している名称

名称	意味
DLLcobol	呼び出される COBOL プログラムに対応する Java メソッドのメソッド名

名称	意味
DLLfunc.dll	呼び出される COBOL プログラムを含む DLL のファイル名
DLLclass	呼び出される COBOL プログラムに対応する Java メソッドを含む Java プログラム（Java クラス）のクラス名
DLLclass.java	DLLclass を記述したソースファイルのファイル名

注意事項

- 他言語プログラムに対応する Java メソッドの宣言には，「native」を指定します。「native」指定は，Java 言語以外の言語で実装されたメソッドであることを表します。なお，メソッドの本体は記述しません。
- 「native」が指定された Java メソッドに対応する他言語プログラムのプログラム名は，Java メソッド名を変換した名前でなければなりません。詳細は，「19.3.2　Java プログラムから COBOL プログラムを呼び出す方法」の「(2)　COBOL 言語側の作業」を参照してください。
- 「loadLibrary」メソッドの引数には，呼び出す他言語プログラムを含む DLL のファイル名（拡張子は除く）を指定します。

COBOL プログラムの初期化・終了処理をする C プログラムに対応する Java プログラム（Java クラス）

COBOL プログラムに対応する Java プログラム（Java クラス）と同様に，COBOL プログラムの初期化・終了処理を実行するサービスルーチン（CBLGINT／CBLEND）を呼び出す C プログラムに対応する Java プログラム（Java クラス）が必要です。

C プログラムに対応する Java プログラム（Java クラス）の例と，この例で仮定している名称を次に示します。

Java ソースファイルの例（DLLcbl2002rt.java）

```
class DLLcbl2002rt {
  public native int cblgint ();
  public native int cblend ();
  static {
    System.loadLibrary ("DLLcbl2002rt");
  }
}
```

表 19-4　例で仮定している名称

名称	意味
DLLcbl2002rt	呼び出す C プログラムを含む DLL のファイル名
DLLcbl2002rt	「native」指定を持つ Java メソッドと DLL をロードする処理を含む Java クラスのクラス名
DLLcbl2002rt.java	Java クラス「DLLcbl2002rt」を定義した Java ソースファイルのファイル名

(b)　Java ソースファイルのコンパイル

作成した Java ソースファイルをコンパイルして，Java バイトコードファイルを生成します。

次に，(a)で作成した Java ソースファイルのコンパイル指定例を示します。

Javaソースファイルのコンパイル指定例

```
javac DLLtest.java
javac DLLclass.java
javac DLLcbl2002rt.java
```

上記の指定を実行すると，コンパイル後にJavaバイトコードファイル「DLLtest.class」「DLLclass.class」「DLLcbl2002rt.class」が生成されます。

(c) ヘッダファイルの生成

生成されたJavaバイトコードファイルの中の，他言語プログラムに対応するJavaプログラム（Javaクラス）のJavaバイトコードファイルから，C言語ヘッダファイルを生成します。このヘッダファイルには，「native」指定のあるJavaメソッドに対応するC言語の関数プロトタイプ宣言が含まれています。

サービスルーチンを呼び出すCプログラムは，このヘッダファイルをインクルードする必要があります。このヘッダファイルは，COBOLプログラム作成時にも必要です。

(b)で生成したJavaバイトコードファイル「DLLclass.class」と「DLLcbl2002rt.class」から，C言語ヘッダファイルを生成させる例を次に示します。

C言語ヘッダファイルの生成指定例

```
javah -jni DLLclass
javah -jni DLLcbl2002rt
```

上記の指定を実行すると，C言語ヘッダファイル「DLLclass.h」と「DLLcbl2002rt.h」が生成されます。次に，生成された「DLLclass.h」中のメソッド「DLLcobol」の関数プロトタイプ宣言の例を示します。

```
JNIEXPORT jint JNICALL Java_DLLclass_DLLcobol
                       (JNIEnv *, jobject, jint, jint);
```

この例では，Javaメソッドに対応して次のようなC言語の関数が作成されたことになります。

関数の要素	内容
関数名	Java_DLLclass_DLLcobol
引数	4個
関数の戻り値の型	jint

注意事項

- javahコマンドには，-jniオプションを指定してください。このオプションを指定すると，Java以外の言語によって実装されたメソッドとのインタフェースであるJava言語の他言語インタフェーススタイルのヘッダファイルが生成されます。
- javahコマンドの引数には，「native」指定のあるJavaメソッドを含む，Javaクラスのクラス名を指定します。

(2) COBOL 言語側の作業

(a) COBOL 原始プログラムの記述

Java プログラムから実際に呼び出される COBOL プログラムのソースファイルを作成します。ここで作成する COBOL プログラムは，次の条件を満たしている必要があります。

- COBOL プログラム名は，Java バイトコードファイルから生成されたヘッダファイル中の，対応するプロトタイプ宣言と同じ名称でなければならない。
- COBOL プログラム名は，引用符で囲まなければならない。
- COBOL プログラムの引数，戻り値は，Java バイトコードファイルから生成したヘッダファイル中の，対応するプロトタイプ宣言と同じ個数でなければならない。
- COBOL プログラムの引数，戻り値の型は，対応する COBOL の型でなければならない。

  データ型の対応関係については，「19.3.3 Java 言語と COBOL 言語のデータ型の対応」を参照してください。

Java プログラムから呼び出される COBOL プログラムの，ソースファイル例を次に示します。

COBOL プログラムのソースファイル例（DLLfunc.cbl）

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. 'Java_DLLclass_DLLcobol'.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       LINKAGE SECTION.
         01 JNIEnv             USAGE ADDRESS.   …1.
         01 jobject            USAGE ADDRESS.   …1.
         01 arg-1   PIC S9(9) USAGE COMP.   …2.
         01 arg-2   PIC S9(9) USAGE COMP.   …2.
         01 ret-1   PIC S9(9) USAGE COMP.   …3.
       PROCEDURE DIVISION USING BY VALUE
                  JNIEnv jobject arg-1 arg-2  …4.
                   RETURNING ret-1.
      *手続き部本体の記述
           DISPLAY 'arg-1: ' arg-1.
           DISPLAY 'arg-2: ' arg-2.
           COMPUTE ret-1 = arg-1 * arg-2.
           DISPLAY 'ret-1: ' ret-1.
       END PROGRAM 'Java_DLLclass_DLLcobol'.
```

1. COBOL プログラムの仮引数「JNIEnv」「jobject」は，必ず指定してください。これらの引数は，Java の実行時環境を保持するポインタで，Java 言語側とインタフェースの同期をとります。
2. COBOL プログラムの仮引数「arg-1」，「arg-2」は，対応する Java メソッドの引数「arg1」と「arg2」にそれぞれ対応します。
3. COBOL プログラムの戻り値「ret-1」は，対応する Java メソッドの戻り値と対応します。なお，対応する Java メソッドの型が void の場合は，RETURNING を指定しないでください。

4. Java から COBOL への引数は，「値渡し」で渡されます。したがって，COBOL 側では仮引数に「BY VALUE」を指定する必要があります。

(b) COBOL ソースファイルのコンパイル

作成した COBOL ソースファイルをコンパイルして stdcall 呼び出し規約（Windows(x86) COBOL2002 で有効），または fastcall 呼び出し規約（Windows(x64) COBOL2002 で有効）の DLL を生成します。

次に，(a)で例に挙げたソースファイル「DLLfunc.cbl」をコマンドプロンプト上からコンパイルする例を示します。

Windows(x86) COBOL2002 の場合

```
ccbl2002 -Dll,Stdcall -DllInit -MainNotCBL -Bin1Byte
         -OutputFile DLLfunc.dll DLLfunc.cbl
```

Windows(x64) COBOL2002 の場合

```
ccbl2002 -Dll※ -DllInit -MainNotCBL -Bin1Byte
         -OutputFile DLLfunc.dll DLLfunc.cbl
```

注※

Windows(x64) COBOL2002 では fastcall 呼び出し規約を使用するため，サブオプションは指定しません。

注意事項

- 生成する DLL のファイル名は，Java プログラムで指定した DLL ファイル名と同じでなければなりません。
- コンパイラオプションに，-Dll,Stdcall（Windows(x86) COBOL2002 で有効），-Dll（Windows(x64) COBOL2002 で有効），および-MainNotCBL オプションの指定が必要です。また，初期化属性を持つ DLL を作成したい場合は，-DllInit オプションを指定します。
- Java 言語のデータ型「Boolean」または「byte」を COBOL プログラムで引数として受け取る場合には，1 バイト 2 進項目を有効とするために-Bin1Byte オプションの指定が必要です。
- 開発マネージャを使用して DLL を生成する場合，最終生成物の種類に「ダイナミックリンクライブラリ」を選び，リンク処理するように指定してください。また，［プロジェクトの設定］メニューの［実行］から，-MainNotCBL オプションを指定してください。指定方法については，マニュアル「COBOL2002　操作ガイド」を参照してください。

(3) C 言語での操作

(a) C ソースファイルの作成

COBOL プログラムの初期化・終了処理をするサービスルーチン（CBLGINT／CBLEND）を呼び出す C プログラムを作成します。

作成するCプログラムの関数は，Javaバイトコードファイルから生成したC言語ヘッダファイル内の関数プロトタイプ宣言と同じ名称・型・引数である必要があります。

次に，C言語ヘッダファイル「DLLcbl2002rt.h」に対応するCソースファイルの例「DLLcbl2002rt.c」を示します。

Cソースファイルの例（DLLcbl2002rt.c）

```
#include <windows.h>
#include "DLLcbl2002rt.h"
extern int WINAPI CBLGINT();
extern int WINAPI CBLEND();
JNIEXPORT jint JNICALL
    Java_DLLcbl2002rt_cblgint(JNIEnv *env, jobject obj)
{
   /* サービスルーチンの初期化ルーチンを
      呼び出す */
   CBLGINT();
}

JNIEXPORT jint JNICALL
    Java_DLLcbl2002rt_cblend(JNIEnv *env, jobject obj)
{
   /* サービスルーチンの終了処理ルーチンを
      呼び出す */
 CBLEND();
}
```

ソース内で使用する関数は，実際にJavaバイトコードファイルから生成されたC言語ヘッダファイル（DLLcbl2002rt.h）中の対応する関数プロトタイプ宣言と同じである必要があります。

(b) Cソースファイルのコンパイル

作成したCソースファイルをコンパイルし，DLLを生成します。

次に，(a)で作成したCソースファイルをコンパイルする指定例を示します。

Cソースファイルのコンパイル指定例

```
cl /LD DLLcbl2002rt.c cbl2k_32.lib
```

注

「cbl2k_32.lib」は，必ず指定してください。これは，COBOL2002で使用できるDLLのインポートライブラリです。指定しない場合，リンクエラーとなります。

上記の例を実行すると，「DLLcbl2002rt.dll」が生成されます。

(4) プログラムの実行

(a) 環境変数の設定

Java プログラムを実行する前に，Java プログラム以外の他言語プログラムを含む DLL があるパス名を，環境変数 PATH に指定する必要があります。環境変数 PATH の指定例を次に示します。

環境変数 PATH の指定例

- 生成した COBOL プログラムの DLL「DLLfunc.dll」，および C プログラムの DLL「DLLcbl2002rt.dll」が，「d:¥users¥dll」にある場合

```
set PATH=d:¥users¥dll;%PATH%
```

(b) Java プログラムの実行

Java プログラムの実行は，JDK に付属している Java インタプリタを使用して実行する方法と，Java インタプリタを内蔵した Web ブラウザから実行する方法があります。

JDK 付属の Java インタプリタを使用して実行する例を次に示します。この場合，「public」属性と「static」属性を持ち，メソッド名が「main」であるメソッドを含むクラスのクラス名（DLLtest）を Java インタプリタの引数に指定します。

Java インタプリタで Java プログラムを実行する例

```
java DLLtest
```

19.3.3 Java 言語と COBOL 言語のデータ型の対応

(1) COBOL 言語で使用できる Java 言語のデータ型と対応

COBOL 言語で使用できる Java 言語のデータ型と，対応する COBOL のデータ項目の型，および C 言語でのデータ型を，次に示します。

Java の型	対応する COBOL のデータ項目の型			対応する C 言語のデータ型
jboolean	-Bin1Byte 指定あり	符号なし 2 進項目（1～2 けた）※	9(1) COMP ～ 9(2) COMP	Boolean
	-Bin1Byte 指定なし	符号なし 2 進項目（1～2 けた）※	9(1) COMP-X ～ 9(2) COMP-X	
jbyte	-Bin1Byte 指定あり	符号付き 2 進項目（1～2 けた）	S9(1) COMP ～ S9(2) COMP	byte
jchar	-Bin1Byte	符号なし 2 進項目	9(3) COMP ～ 9(4) COMP	char

Java の型	対応する COBOL のデータ項目の型			対応する C 言語のデータ型
	指定あり	(3～4 けた)		
	-Bin1Byte 指定なし	符号なし 2 進項目 (1～4 けた)	9(1) COMP ～ 9(4) COMP	
		符号なし 2 進項目 (3～4 けた)	9(3) COMP-X ～ 9(4) COMP-X	
jshort	-Bin1Byte 指定あり	符号付き 2 進項目 (3～4 けた)	S9(3) COMP ～ S9(4) COMP	short
	-Bin1Byte 指定なし	符号付き 2 進項目 (1～4 けた)	S9(1) COMP ～ S9(4) COMP	
jint	－	符号付き 2 進項目 (5～9 けた)	S9(5) COMP ～ S9(9) COMP	int
jlong	－	符号付き 2 進項目 (10～18 けた)	S9(10) COMP ～ S9(18) COMP	_int64
jfloat	－	単精度内部浮動小数点数字項目	COMP-1	float
jdouble	－	倍精度内部浮動小数点数字項目	COMP-2	double

注※

実際の値は以下の二つだけになります。

0：Java 言語での false を意味する

1：Java 言語での true を意味する

また，次に示す Java 言語でのデータ型は，COBOL 言語で使用できません。

- オブジェクト型
- クラス型
- 文字列型
- すべての型の配列

ただし文字列型は，C 言語のサポート関数を作成すると，COBOL で使用できます。詳細は，「19.3.3 Java 言語と COBOL 言語のデータ型の対応」の「(2)　Java の文字列型を COBOL で使用する方法」を参照してください。

(2)　Java の文字列型を COBOL で使用する方法

Java の言語仕様上，文字列型はすべて String クラスのオブジェクトとなっています。そのため，Java プログラムから COBOL プログラムを呼び出した時，引数に文字列型を指定しても，COBOL プログラムでは文字列の内容を参照できません。

しかし，Java 言語の他言語インタフェース機能（Java Native Interface）には，Java 言語の String オブジェクトを C 言語の文字列型に変換できる機能があります。String オブジェクトを C 言語の文字列型に変換する C プログラムを作成し，COBOL プログラムから呼び出すと，COBOL プログラムで Java 言語の文字列型を参照できます。

C プログラムのサポート関数「jstring2string」を記述した C ソースファイル「util.c」の例を次に示します。

サポート関数を定義する C ソースファイル（util.c）

```
#include <windows.h>
#include <jni.h>
            /* JDKで使用できるインクルードファイル */
/* Java言語のStringオブジェクトをC言語の
   文字列に変換する */
/* この関数はC言語の文字列での文字列長を返す */
int jstring2string
(char *str,     /* C言語の文字列を格納するバッファ */
 int max,       /* 格納するバッファのサイズ       */
 JNIEnv *env,   /* Java実行時環境のポインタ       */
 jstring jstr) /* Java言語のString文字列         */
{
  const jchar *wstr;
  /* JavaNativeInterfaceの文字列変換機能を利用する */
  /* ただし，この機能によって変換された文字列は
     Unicodeである */
  wstr = (*env)->GetStringChars (env, jstr, 0);
  /* Unicode文字列をシフトJIS文字列に変換する */
  WideCharToMultiByte
      (CP_ACP, 0, wstr, -1, str, max, NULL, NULL);
  /* 変換後の文字列の長さを返す */
  return strlen (str);
}
```

上記の C プログラム（util.c）のサポート関数を呼び出す COBOL プログラムの例を次に示します。このプログラムが，Java プログラムから呼び出されるとき，引数には文字列が代入されます。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. 'Java_DLL_func'.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
         01 len PIC S9(9) USAGE COMP.
         01 max PIC S9(9) USAGE COMP VALUE 1024.
         01 str.
           02 FILLER PIC X OCCURS 1024.
       LINKAGE SECTION.
         01 JNIEnv  ADDRESS.
         01 jobject ADDRESS.
         01 jstr    ADDRESS.
       PROCEDURE DIVISION USING BY VALUE
                             JNIEnv jobject jstr.
           CALL 'jstring2string' USING str
                          BY VALUE max JNIEnv jstr
                          RETURNING len.
           DISPLAY 'len: ' len.
           DISPLAY 'str: ' str(1:len).
```

```
        EXIT PROGRAM.
    END PROGRAM 'Java_DLL_func'.
```

19.3.4 制限事項

Java プログラムインタフェース機能での制限事項について説明します。

- Java バイトコードファイルから生成される C 言語ヘッダファイル中の，Java メソッドに対応する関数名の長さは，160 バイトまでです。関数名が 160 バイト以内になるよう，Java メソッド名を定義してください。
- Java プログラム呼び出し機能を使用して呼び出した Java プログラムから，同じスレッドで COBOL プログラムを呼び出せません。呼び出した場合の動作は保証しません。

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。

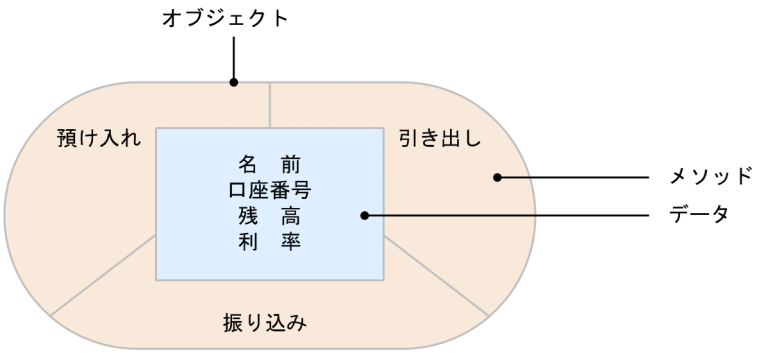
- 残高
- 利率

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。

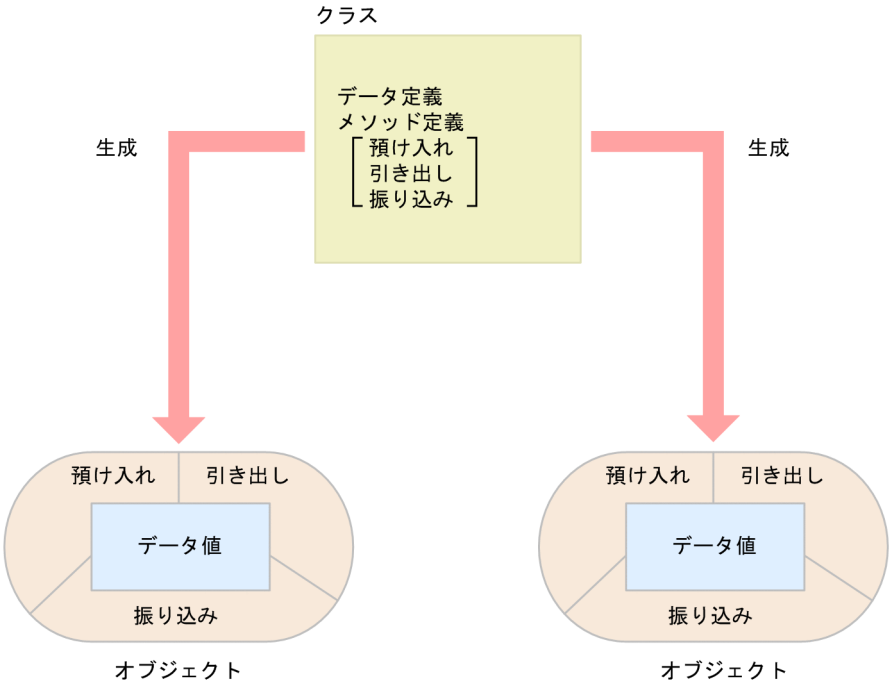
このカプセル化は，従来のプログラミング言語によるシステム開発で課題となっていた，データ間またはデータとプログラムの依存度の高さによって発生する変更時の障害を解消します。

(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。

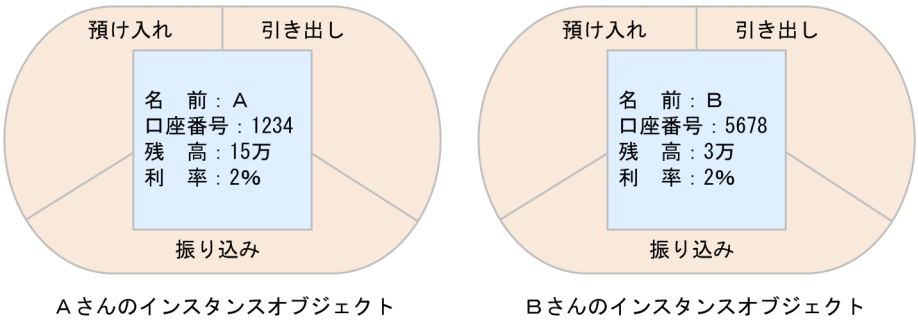
銀行口座に必要なデータ（名前，口座番号など）と，それらのデータを操作するメソッド（預け入れ，引き出しなど）は，銀行口座クラスとして定義します。こうして定義されたクラスは，オブジェクトを生成するための型であり，だれの口座でもありません。実際の口座は，銀行口座クラスを基にオブジェクトとして生成します。各人の口座のデータの定義は同じですが，名前，口座番号などのデータには，それぞれ異なる値が設定されます。



(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。



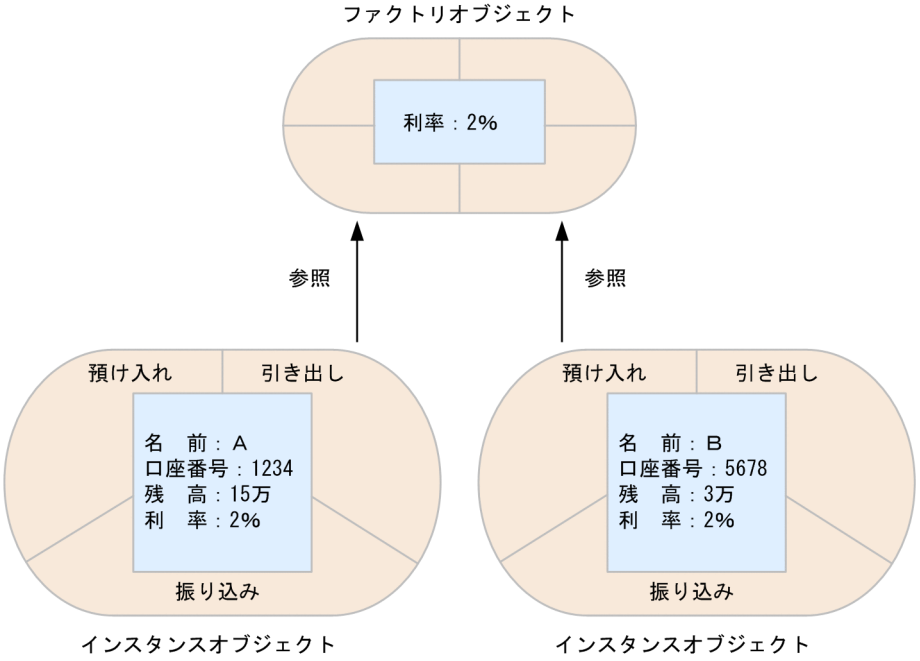
(b)　ファクトリオブジェクト

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。



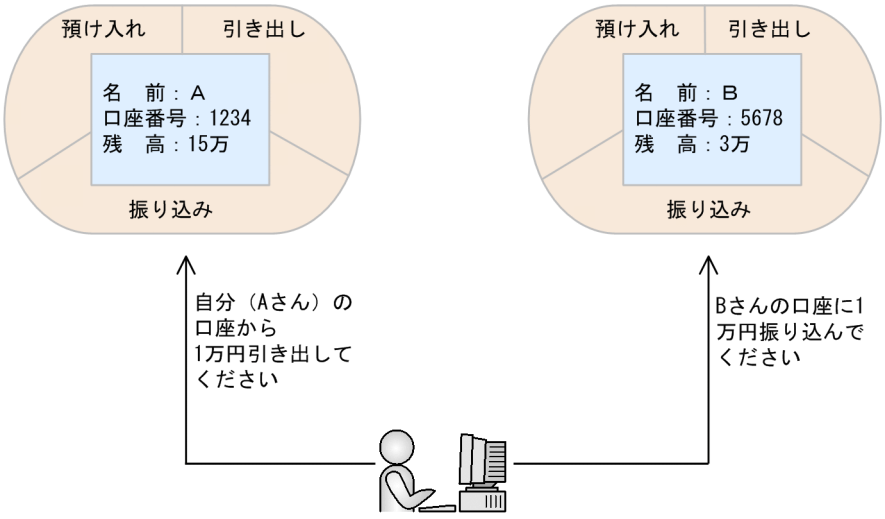
(5)　メソッドの呼び起こし（メッセージパッシング）

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

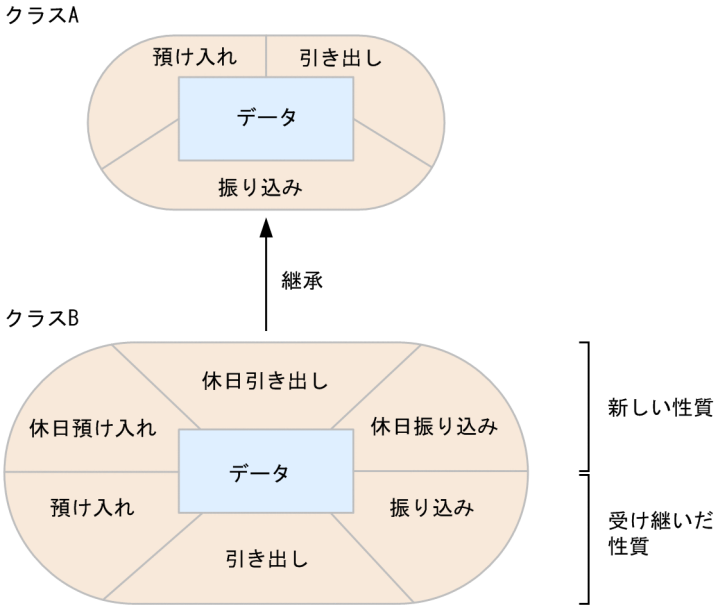
A さんが B さんの口座に 1 万円を振り込もうとしています。しかし，A さんの口座オブジェクトのメソッドは，B さんの口座オブジェクトのデータを操作できません。そこで，B さんの口座オブジェクトに次のようなメッセージを送ります。



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

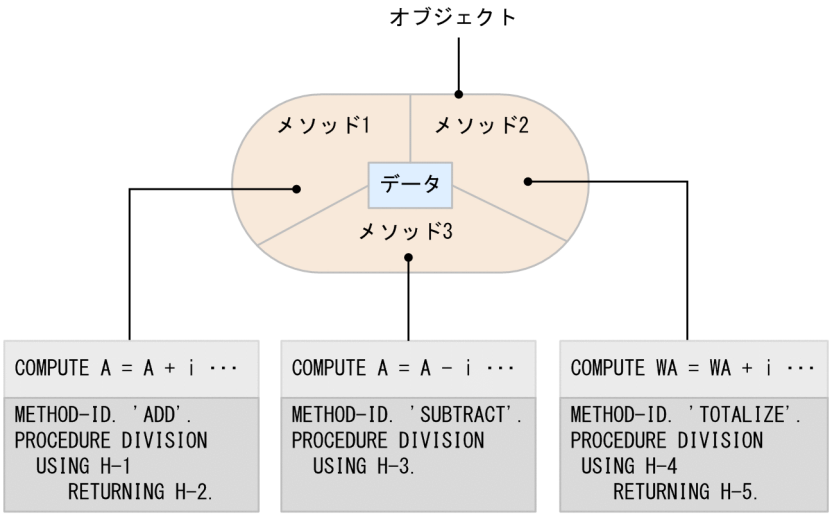
インタフェースは，このあとに説明する適合やポリモルフィズムに深くかかわっている重要な概念です。インタフェースは，オブジェクトがどのようなサービスを提供するのかを表します。具体的には，オブジェクトが持つメソッドの名前とパラメタの並び（これらをまとめて，メソッド原型といいます）の集まりから成ります。インタフェースは，オブジェクトが持つすべてのメソッド原型を表現することも，一部を表現することもできます。

(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。

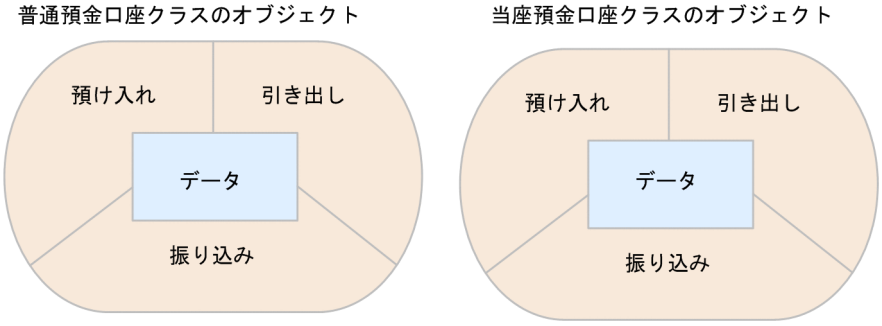
(8) ポリモルフィズム

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。

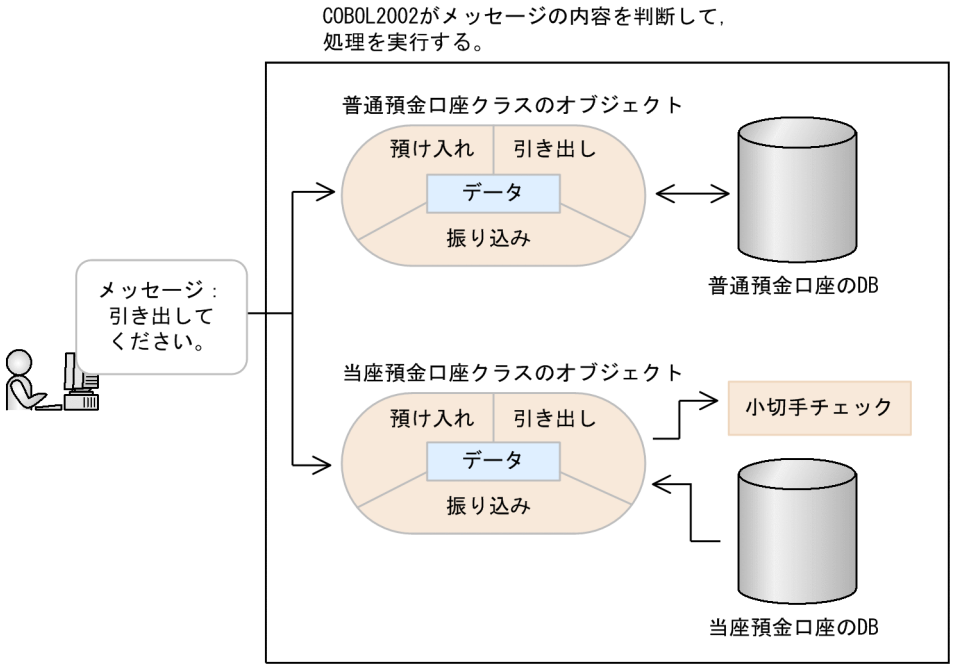
（例）普通預金口座および当座預金口座からの引き出し

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

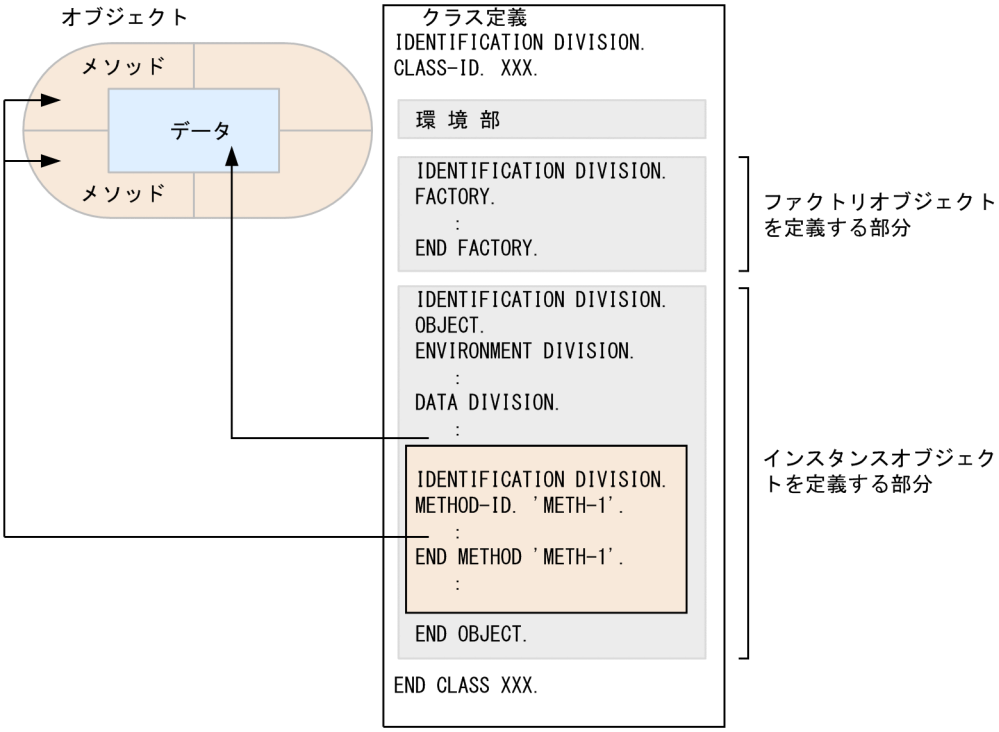
プログラム定義では，リポジトリ段落に使用するクラス名を指定します。また，インスタンスオブジェクトを生成する場合，オブジェクトを識別するためのオブジェクト参照を定義します。手続き部では，メソッドを呼び起こすための手続き文である INVOKE 文を記述します。

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，「36.2　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.
```

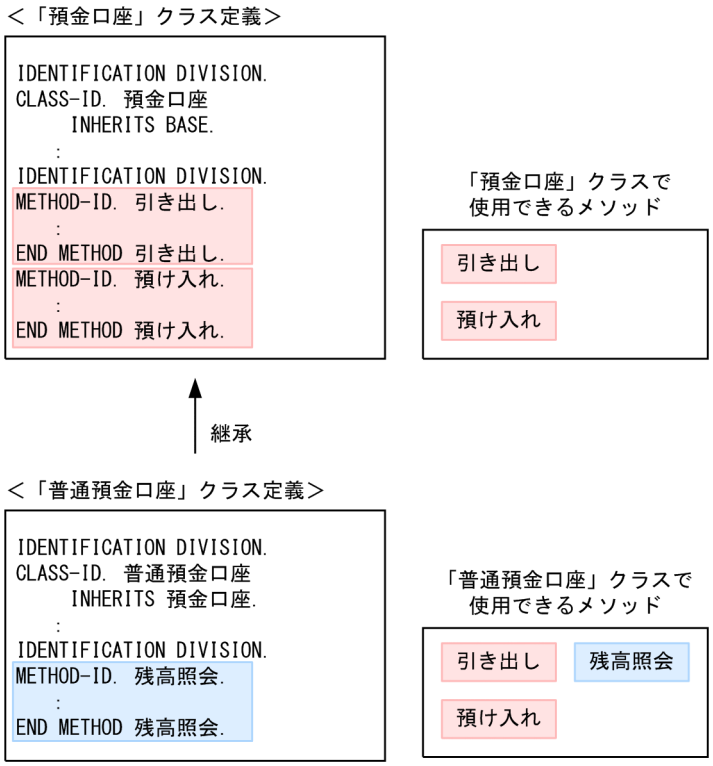
継承すると，スーパクラスの性質が，継承するクラスおよび継承するクラスのサブクラスにも定義されているかのように扱われます。継承の詳細については，マニュアル「COBOL2002　言語　標準仕様編」「5.2.9　クラス継承」を参照してください。

(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)

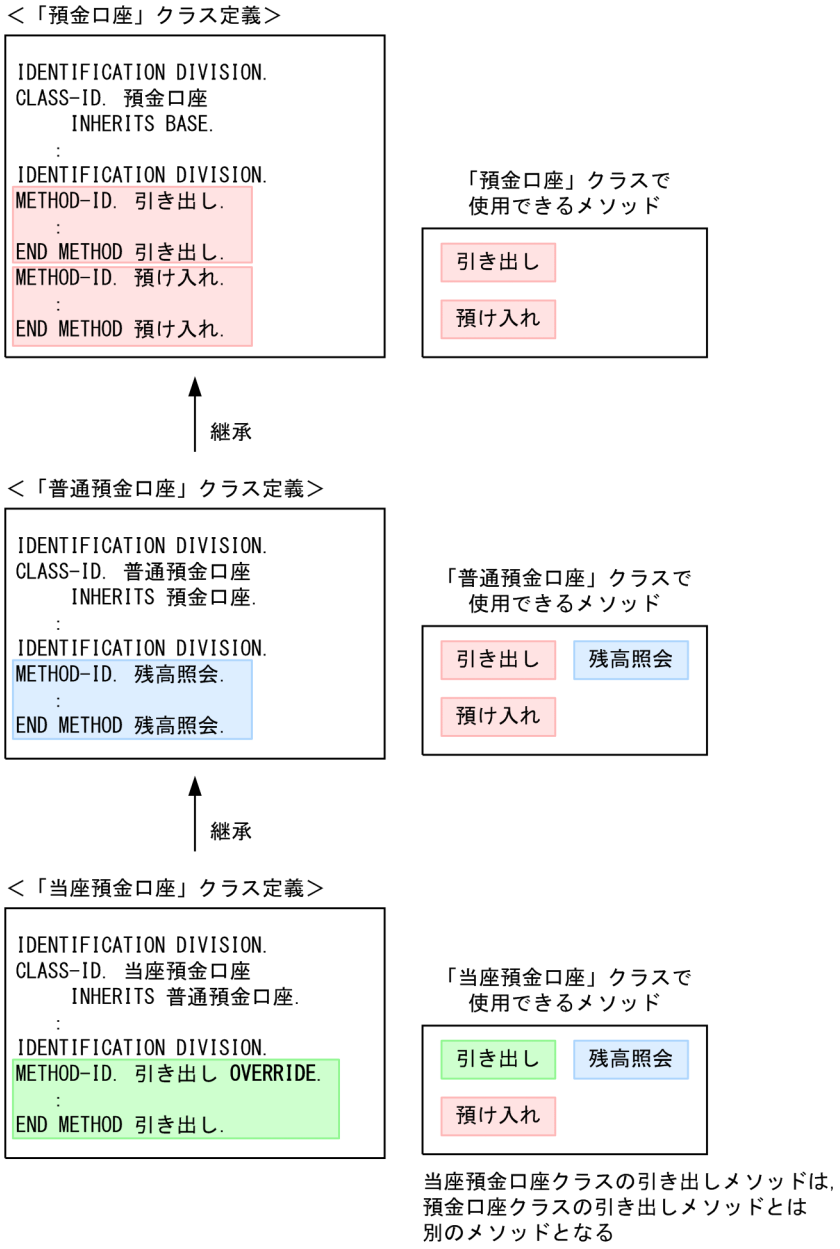
「預金口座」クラスの「預け入れ」メソッド，「引き出し」メソッドに加えて「残高照会」メソッドを持つ「普通預金口座」クラスを作成します。この場合，「普通預金口座」クラスは，「預金口座」クラスを継承し，「残高照会」メソッドだけを定義すれば，必要なメソッドを備えられます。



(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。

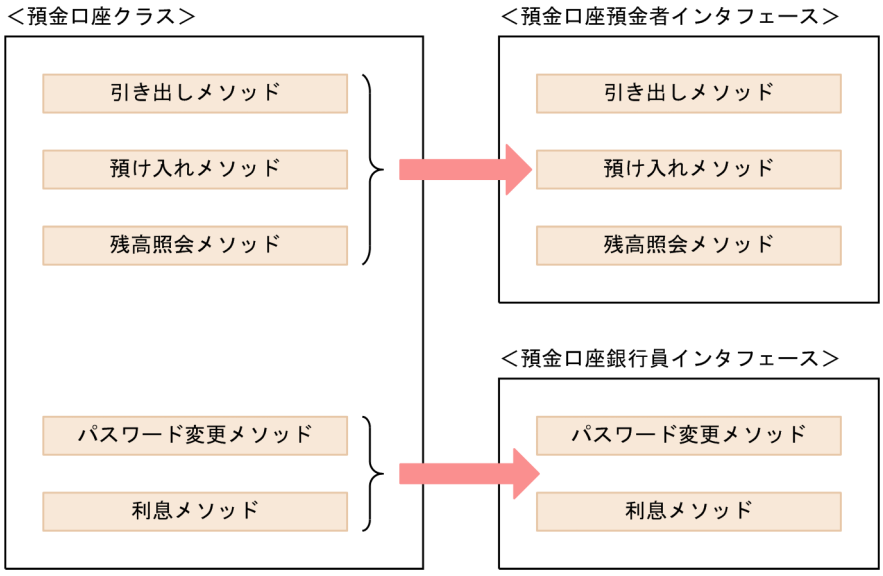
ファクトリオブジェクトとインスタンスオブジェクトは，それぞれインタフェースを持ちます。このインタフェースは，それぞれのオブジェクトが持つ，すべてのメソッド定義の集まりです。

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「34.3.2　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。

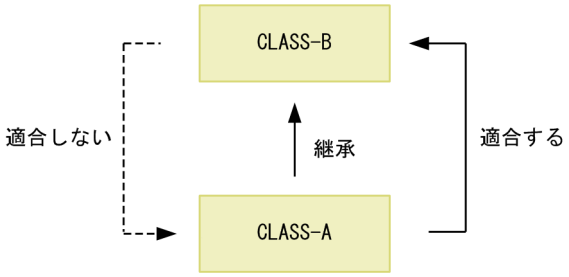
ここでは，適合チェックの概要について，説明します。

(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。

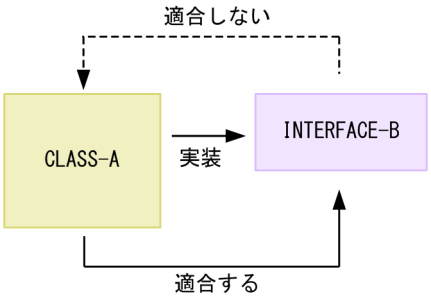
(b) インタフェースとクラスの適合

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

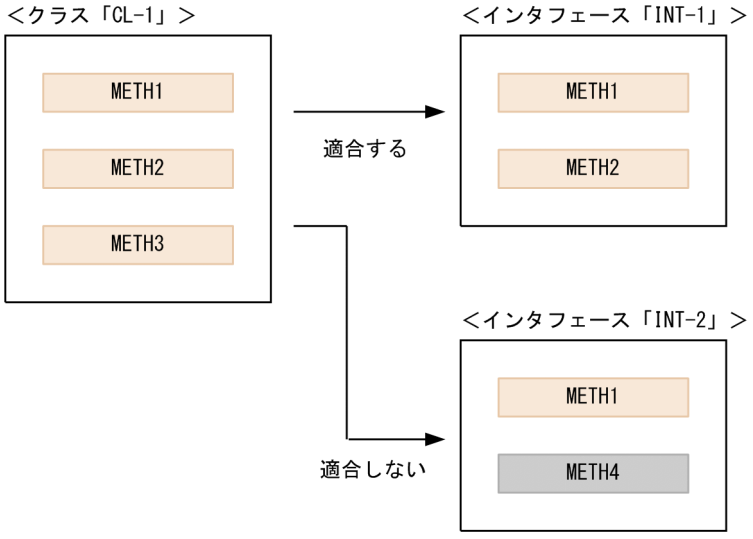
　METH1，METH2，METH3

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

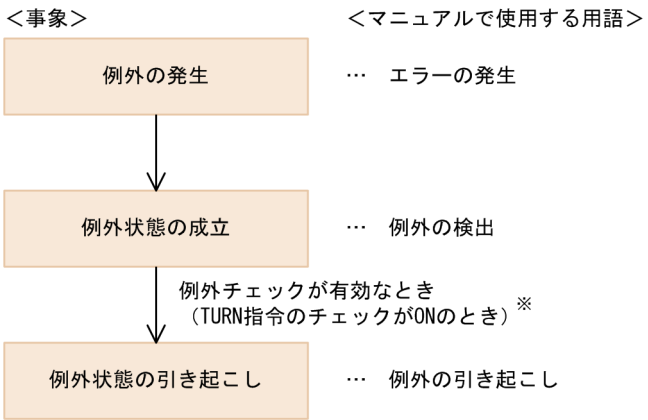
2. エラーの種類または入出力ファイル別に例外処理を定義する場合

   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	×	
	C 言語インタフェース	×	
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	○	
	Btrieve による索引編成ファイル※2	○	
	CSV 編成ファイル	○	
	ラージファイル入出力	○	
	プリンタへのアクセス（入出力による書式印刷機能）※2	○	
	プリンタへのアクセス（GDI モード印刷）	○	
	プリンタへのアクセス（ESC/P モード印刷）	○	
	ファイルのディスク書き込み保証	○	
	イベントログファイル出力機能	×	
	通信節による画面操作	×	
	画面節（WINDOW SECTION）による画面操作	×	
	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	×	
	OLE2 オートメーションインタフェース機能	×	
	マルチスレッド環境での実行	○	
	基本機能サービスルーチン	×	
	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	×	

種類	機能名	共通例外への対応	備考
	MSMQ アクセス機能	×	
	Unicode 機能	○	
	数字項目のけた拡張機能※3	△※4	
	動的長基本項目機能	○	
	定数長拡張機能	○	
	Java プログラム呼び出し機能	×	
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

Windows(x86) COBOL2002 で有効です。

注※3

Windows(x64) COBOL2002 で有効です。

注※4

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，「29.　数字項目のけた拡張機能（Windows(x64) COBOL2002 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。

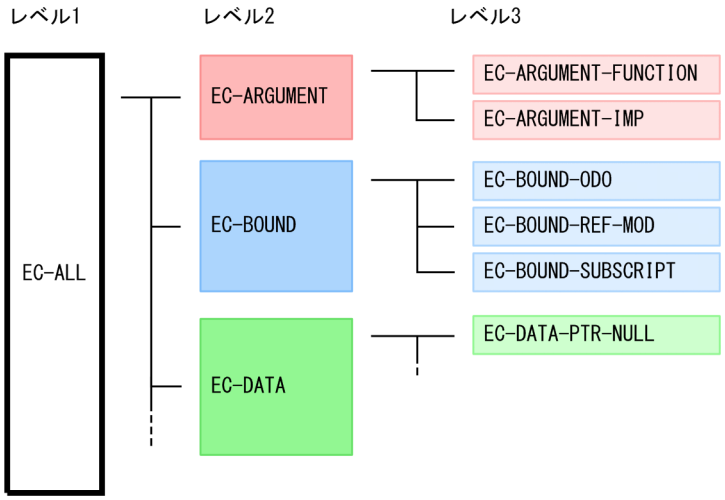
レベル 1

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 H　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	• DLL 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

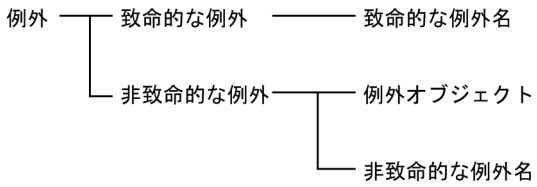
例外オブジェクトは，オブジェクト指向機能を使用した COBOL プログラムで例外を取得するときに，例外名に関連づけられた例外と同じように使用します。例外オブジェクトは，例外名に関連づけられた例外と比べて，多様な例外状況を取得できます。例外オブジェクトの詳細については，「21.7.2　EXCEPTION-OBJECT」を参照してください。また，オブジェクト指向機能については，「20.　オブジェクト指向機能」を参照してください。

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

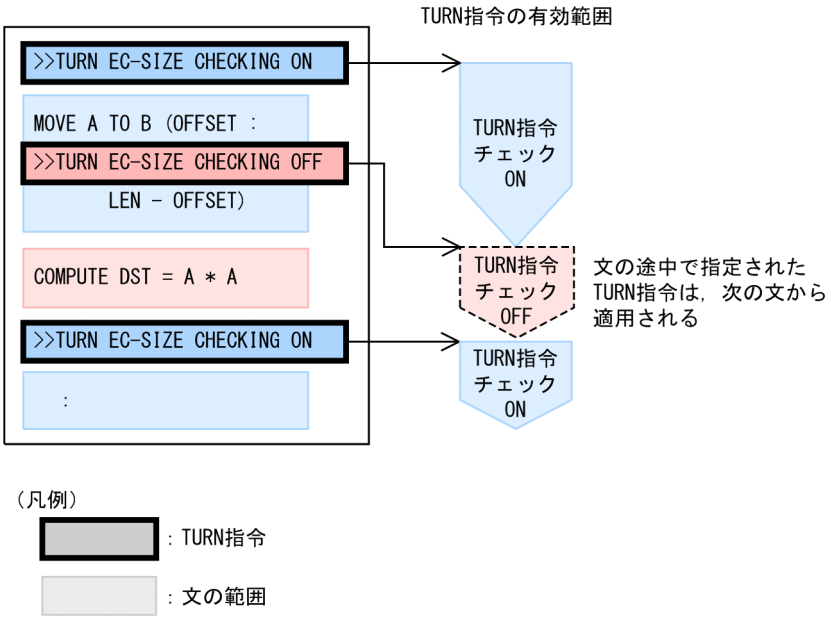
表 21–4　TURN 指令と対象の例外名（EC-I-O 例外名）

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。

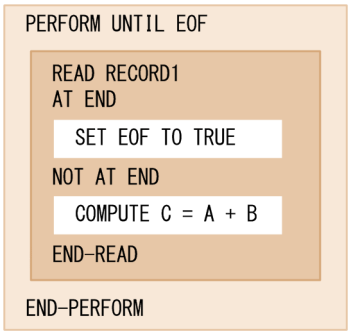
TURN 指令を文の途中に記述した場合の例を，次に示します。



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 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 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 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 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	アプリケーションエラー（ハードウェア例外）で異常終了。
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。
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。

例外名	例外チェックが無効な場合の動作
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。
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。

例外名	例外チェックが無効な場合の動作
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。

例外名	例外チェックが無効な場合の動作
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	• -TDInf オプションなしの場合 ON SIZE ERROR, NOT ON SIZE ERROR 指定がない場合はアプリケーションエラー（ハードウェア例外）で異常終了。 • -TDInf オプションありの場合 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，ハードウェア例外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，ハードウェア例外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，ハードウェア例外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，ハードウェア例外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 DLL 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※	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	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※	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
			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※

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。

(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

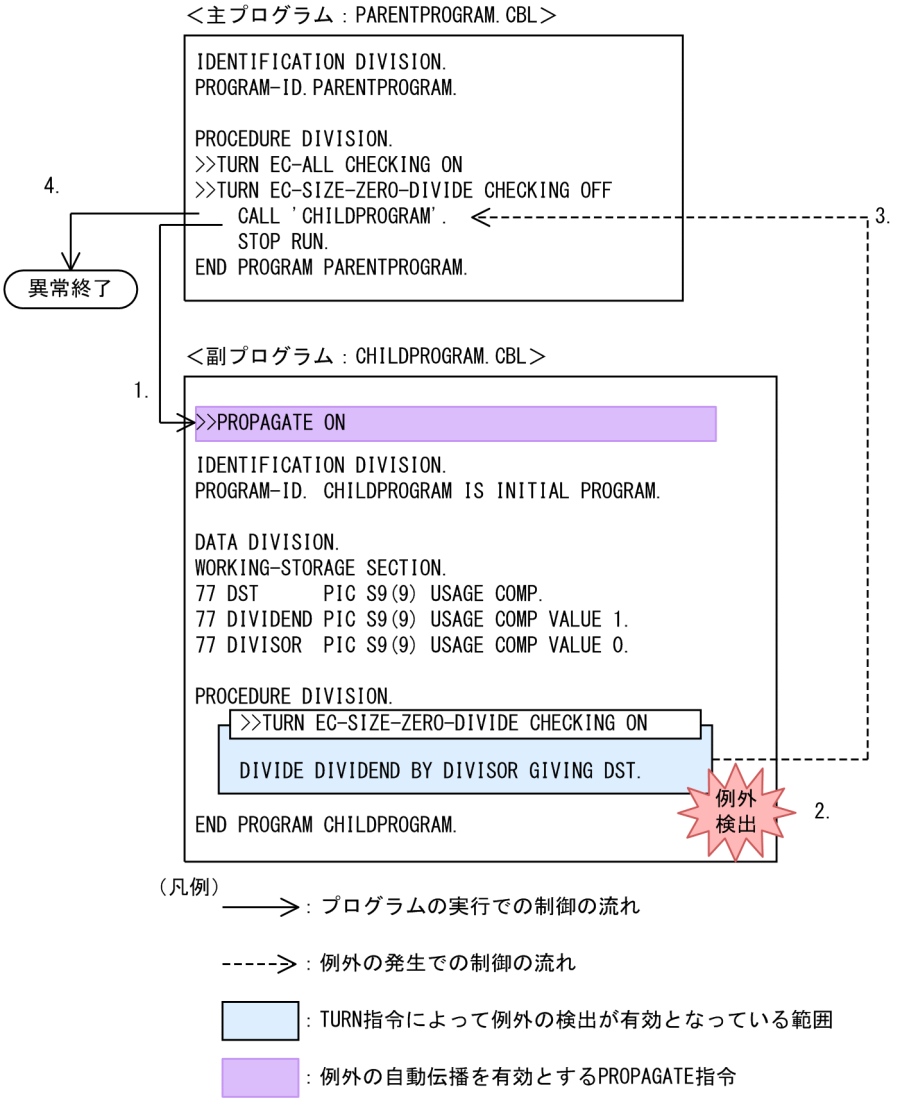
致命度	動作
致命的な例外	「図 21–5　プログラム「CHILDPROGRAM」から致命的な例外の伝播を実行するが，プログラム「PARENTPROGRAM」の例外チェックが無効な場合の動作例」のように，制御遷移した文（CALL 文，INVOKE 文，および利用者定義関数の呼び出し）で，実行時エラーメッセージ（KCCC0402R-S）を出力し，実行単位が異常終了となる。 なお，例外を受け取れないプログラム※の場合は，「図 21–6　プログラム「CHILDPROGRAM」から致命的な例外の伝播を実行しようとするが，プログラム「PARENTPROGRAM」が例外を受け取れない場合の動作例」のように，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。

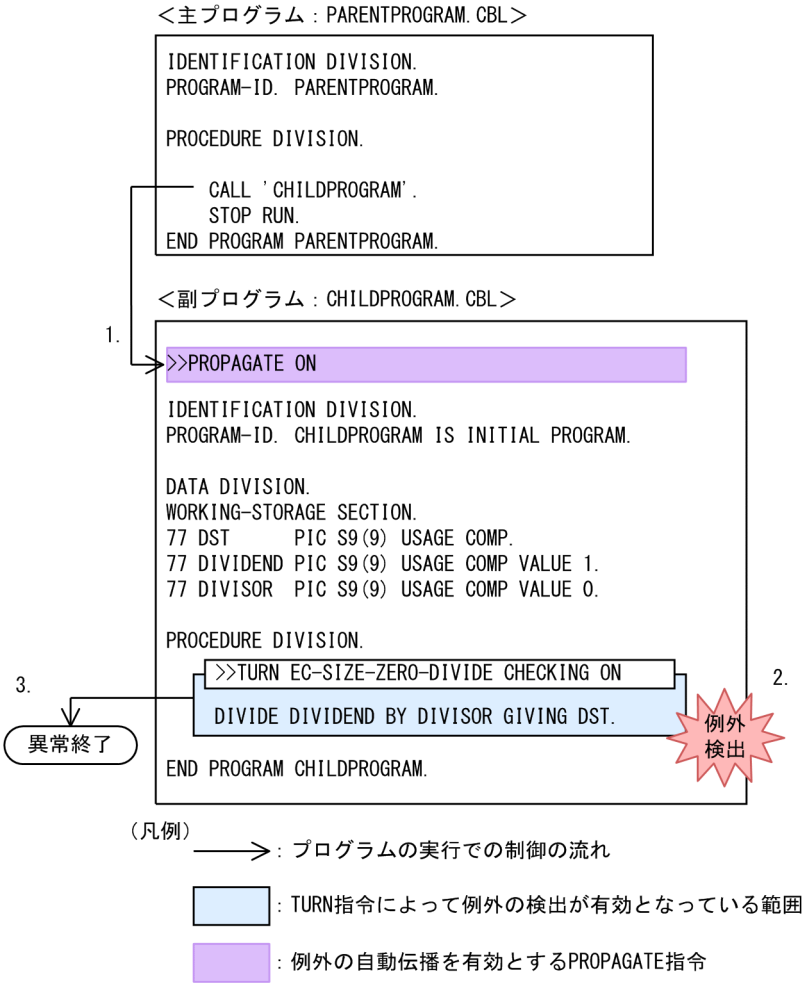
図 21-5　プログラム「CHILDPROGRAM」から致命的な例外の伝播を実行するが，プログラム「PARENTPROGRAM」の例外チェックが無効な場合の動作例



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※

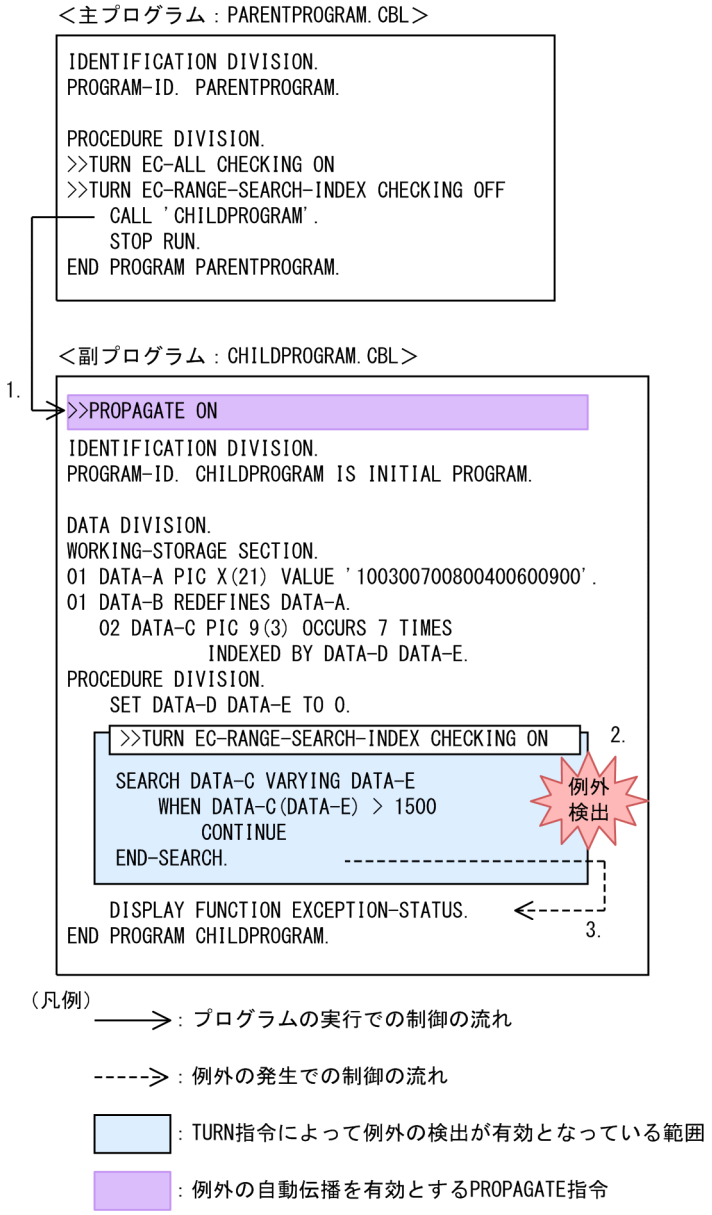
この場合の例外を受け取れないプログラムとは，次の条件がすべて重なった場合を指します。

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.
```

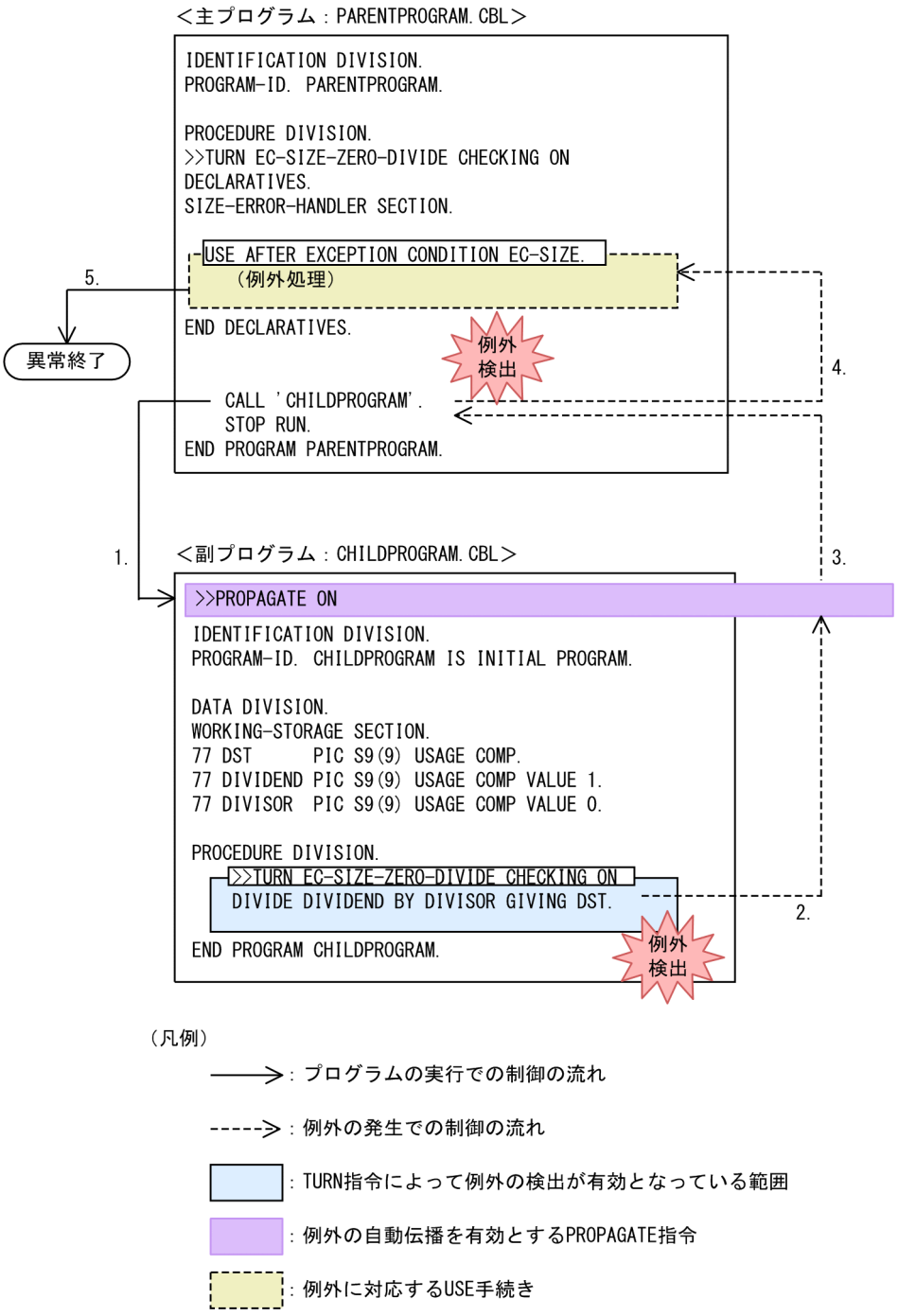
1. プログラム「SAMPLE1」中の CALL 文で，プログラム「SAMPLE2」を呼び出します。
2. プログラム「SAMPLE2」中の DIVIDE 文で，例外が検出されます。
3. 例外宣言手続きが実行されます。
4. GOBACK 文によって，宣言手続きから呼び出し元のプログラム「SAMPLE1」の CALL 文の次の行へ復帰します。

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※1	COBOL2002				COBOL 以外の言語
				-Compati85,NoPropagate 指定なし※2		-Compati85,NoPropagate 指定あり※2		
				伝播抑止条件※3	左記以外	伝播抑止条件※3	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※4	×	×	○	×	×	－
		無条件指定あり※4	×	○※5	○	×	×	－

（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

COBOL85 でコンパイルしたプログラムが動作可能なシステム

注※2

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，「33.5.12　他システムとの移行の設定」の「(3)　-Compati85 オプション」を参照してください。

注※3

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※4

ON OVERFLOW 指定，ON EXCEPTION 指定，NOT ON EXCEPTION 指定

注※5

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 ＞

```
       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 ＞

```
       >>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」「11.22 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。なお，-LiteralExtend,Alnum オプションを指定した場合は，1,167 バイトになります。

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

手続きの種別	関数の戻り値の内容
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」「11.23 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」「11.24　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」「11.25 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	○	−
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	○	−
MEDIAN	○	○
MOD	○	−

組み込み関数	EC-ARGUMENT	
	FUNCTION	IMP
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※1	○	○	○	−
ADD	○	−	○	−
CALL	○	−	○	−
CANCEL	−	−	○	−
COMPUTE	○	−	○	○

文	EC-BOUND			EC-PROGRAM（利用者定義関数を指定可能な文）
	ODO	REF-MOD	SUBSCRIPT	
DISPLAY	−	○	○	○
DIVIDE	○	−	○	○
EVALUATE	−	○	○	○
EXIT	−	−	−	−
GOBACK	−	−	−	−
IF	−	○	○	○
INITIALIZE	○	○	○	−
INSPECT	○	○	○	−
INVOKE※2	○	−	○	−
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※1

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。

注※2

OLE2 オートメーションインタフェース機能については，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
ダイナミックリンク機能の処理中にメモリが不足した。	EC-PROGRAM-RESOURCES
DLL 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
-MultiThread オプション指定あり／なしのプログラムが混在して動作した。	EC-PROGRAM-RESOURCES
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，呼び出し先プログ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を使う場合の指定

OpenTP1 を使用したデータコミュニケーション機能を使用する場合，-OpenTP1 オプションを指定してプログラムをコンパイルする必要があります。

なお，テストデバッガの DC シミュレーション機能で，データコミュニケーション機能を単体テストする場合は，このオプションを指定してはいけません。

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　操作ガイド」のテストデバッガの説明を参照してください。

注意事項

- 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

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。

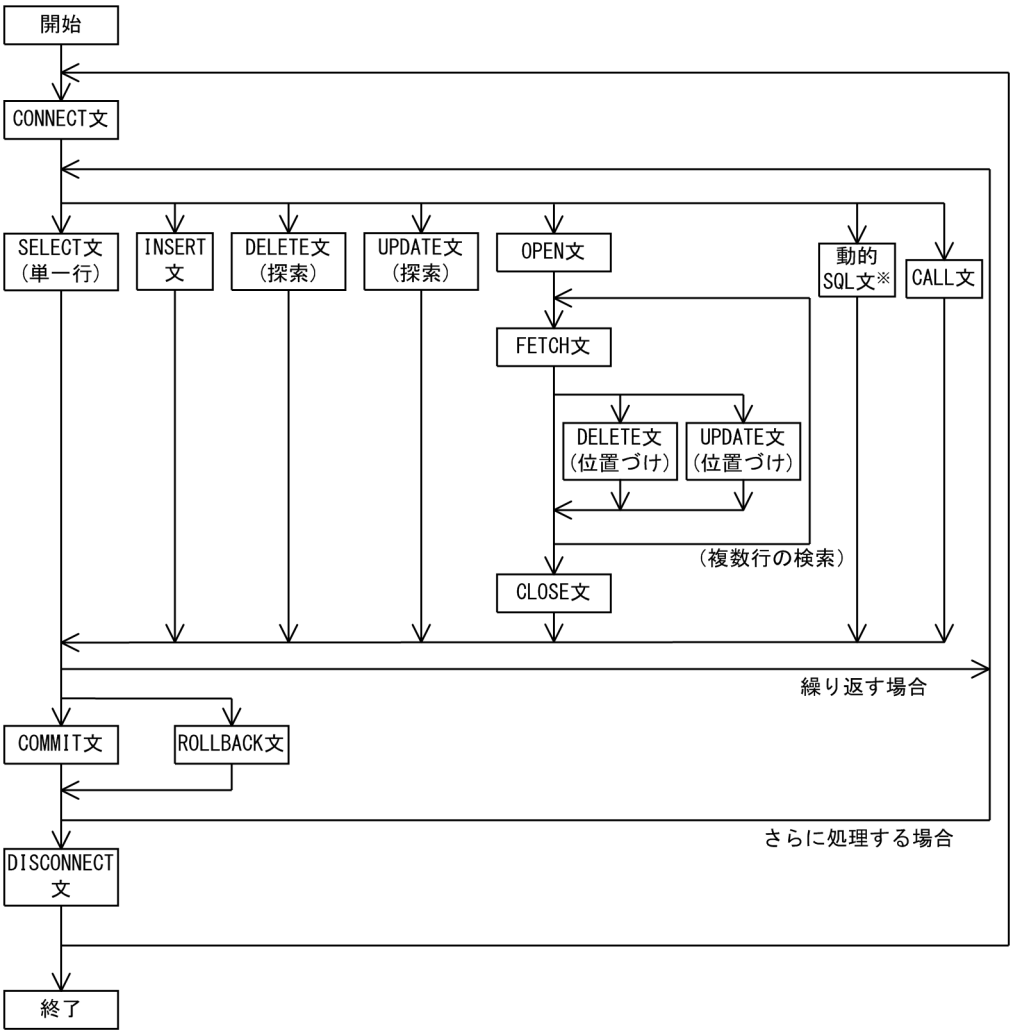
一つのデータベースに対して複数のプログラム（コネクション）からアクセスする場合，デッドロックが発生する可能性があります。デッドロックが発生する可能性を低くするために，プログラム作成時には次のことに注意してください。

- 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

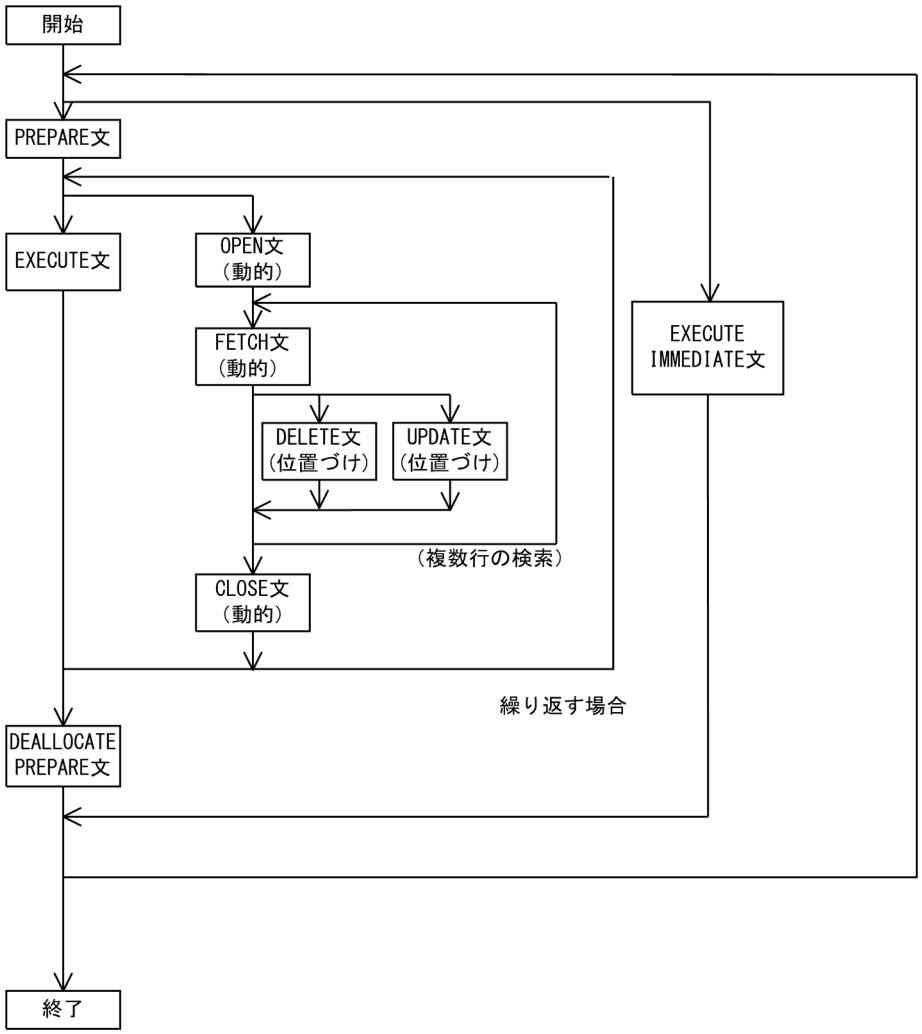
SQL 文は，次に示す順序で実行する必要があります。

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，SQL Server 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

氏名と住所の列を持つ「住所録」という表を定義します。

```
CREATE TABLE 住所録 ( 氏名 char(20),
                      住所 varchar(255))
```

(b) 表へのアクセス

「住所録」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 'サンプル'.
           01 ODBC-UID PIC X(10) VALUE 'sa'.
           01 ODBC-PWD PIC X(10) VALUE SPACE.
           01 行数     PIC S9(9) USAGE COMP VALUE ZERO.
           01 住所録.
            02 氏名    PIC  X(20).
            02 住所.
             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 氏名.
           MOVE '福岡市博多区１丁目' TO ODBC-char OF 住所.
           MOVE 18 TO ODBC-length OF 住所.
           EXEC SQL
             SELECT COUNT(*) INTO :行数 FROM 住所録
             WHERE 氏名 = :氏名
           END-EXEC.
      *<住所録の表に登録する>
           IF 行数 = 0 THEN
             EXEC SQL
               INSERT INTO 住所録 ( 氏名, 住所 )
               VALUES ( :氏名, :住所 )
             END-EXEC
      *<住所録の表の住所を更新する>
           ELSE
             EXEC SQL
               UPDATE 住所録 SET 住所 = :住所
                             WHERE 氏名 = :氏名
             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「住所録」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
             住所更新 ( @氏名 char(20),@住所 varchar(255))
             AS

           IF 0 = (SELECT COUNT(*) FROM 住所録
                                   WHERE 氏名 = @氏名)
             BEGIN
             INSERT INTO 住所録 ( 氏名,住所 )
                    VALUES ( @氏名,@住所 )
             END
           ELSE
             BEGIN
             UPDATE 住所録 SET 住所 = @住所
                           WHERE 氏名 = @氏名
             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 住所更新.
         02 氏名 PIC X(20).
         02 住所.
           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 氏名.
       MOVE '福岡市博多区２丁目' TO ODBC-char OF 住所.
       MOVE 18 TO ODBC-length OF 住所.
       EXEC SQL
         CALL 住所更新 ( :氏名, :住所 )
       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

氏名と住所の列を持つ「住所録」という表を定義します。

```
       CREATE TABLE 住所録 ( 氏名 char(20),
                            住所 varchar(255))
```

(b) 表へのアクセス

「住所録」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 住所録.
         02 氏名 PIC X(20).
         02 住所.
           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 住所録 (氏名, 住所) VALUES (?,?)'
```

```
          TO 動的SQL.
        EXEC SQL
          PREPARE DYNSQL1 FROM :動的SQL
        END-EXEC.
        MOVE '日立 太郎' TO 氏名.
        MOVE '福岡市博多区３丁目' TO ODBC-char OF 住所.
        MOVE 18 TO ODBC-length OF 住所.
        EXEC SQL
          EXECUTE DYNSQL1 USING :氏名, :住所
        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 氏名, 住所 FROM 住所録 ' 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 :氏名, :住所
        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「住所録」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「宛先」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 氏名.
        EXEC SQL
         SELECT 住所 INTO :住所 FROM 住所録
```

```
                    WHERE 氏名 = :氏名
      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 氏名.
      EXEC SQL
        UPDATE 宛先 SET 住所 = :住所 WHERE 氏名 = :氏名
      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

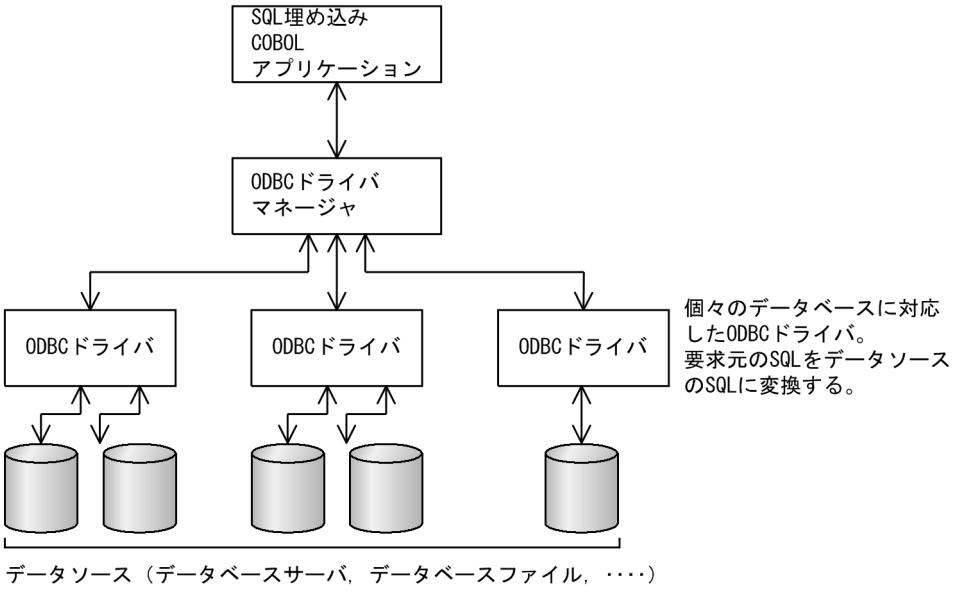
1. testdb1 への接続を確立。
   CONNDB1 が現行コネクションとなる。
2. testdb2 への接続を確立。
   CONNDB2 が現行コネクションとなる。
3. 現行コネクション CONNDB2 を解除。
   現行コネクションは不定となる。
4. 現行コネクションを CONNDB1 に変更。
   CONNDB1 が現行コネクションとなる。

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



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。

- SQLDataSources（ODBC レコード定義生成を使用する場合）
- 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　データソースの管理について

使用するデータソースは，ODBC データソースアドミニストレータを使用して追加，変更できます。ODBC データソースアドミニストレータは，32 ビット版 ODBC Administrator や，ODBC 管理ツール アプリケーションと呼ばれることもあります。

データソースやドライバの詳細については，ODBC ドライバ提供元のマニュアルを参照してください。

ここでは，ODBC データソースアドミニストレータを使用してデータソースを追加する方法，および ODBC 関数呼び出しをトレースする方法について説明します。

なお，次の説明内容は，ODBC3.5 を使用する場合の方法です。データソースとドライバの管理についてのその他の方法は，ODBC データソースアドミニストレータのヘルプ，および ODBC ドライバ提供元のマニュアルを参照してください。

(1) ODBC データソースアドミニストレータの起動

1. コントロールパネルの［管理ツール］からデータソース（ODBC）のアイコンを選ぶ。
   ODBC データソースアドミニストレータが起動し，データソースダイアログボックスが表示されます。

(2) データソースの管理

ODBC ドライバをインストールすれば，一つ以上のデータソースを定義できます。現在インストールされているすべてのドライバに定義されているデータソースは，次の場所に表示されます。

1. [ユーザー DSN] ページの「ユーザーデータソース」に表示されます。
   データソースは，特定の ODBC ドライバと，そのドライバからアクセスしたいデータを関連づけます。したがって，データソースを追加するときは，前提となるドライバがあらかじめ組み込まれている必要があります。

(a) データソースの追加

1. ODBC データソースアドミニストレータを起動し，[ユーザー DSN] ページを表示する。

2. [追加] ボタンを選ぶ。
   データソースの新規作成ウィザードが表示され，インストール済みのドライバの一覧が表示されます。

3. インストール済みのドライバの一覧から，使用するデータソース用のドライバを指定し，[完了] ボタンを選ぶ。
   使用するデータソース用のセットアップダイアログボックスが表示されます。ドライバ固有のセットアップダイアログボックスについては，ODBC ドライバ提供元のマニュアルを参照してください（ヘルプボタンがあればヘルプを参照できます)。

4. そのデータソースに関する名前やフォルダなどの，必要な情報を入力する。

5. データソースの設定が完了したら，セットアップダイアログボックスの［OK］ボタンを選ぶ。
   そのドライバ固有のセットアップダイアログボックスが閉じます。

6. 続けてほかのデータソースを追加したいときは，2.～5.の操作を繰り返す。

7. データソースの追加が終了したら，ODBC データソースアドミニストレータウィンドウの［OK］ボタンを選ぶ。
   指定したデータソースが追加されます。

(3) トレースオプション

ODBC データソースアドミニストレータの［トレース］ページで，ODBC 関数呼び出しをトレースする方法を設定できます。

［トレース］ページからトレースを実行すると，ドライバマネージャは，その後動作する ODBC 関数呼び出しすべてのログをログファイルに記録します。

(a) トレースを実行

1. ODBC データソースアドミニストレータを起動し，［トレース］ページを表示する。

2. [トレースの開始] ボタンを選ぶ。

3. 「ログファイルのパス」に ODBC 呼び出しを書き出すログファイル名を指定する。

   ログファイル名は，新規のファイル名を入力するか，新しい情報を追加する既存のログファイルを［参照］ボタンから選択します。

4. [OK] ボタンを選ぶ。

   ODBC データソースアドミニストレータが終了します。ODBC データソースアドミニストレータが終了した後に起動したすべての ODBC プログラムに対して，ODBC 関数呼び出しのログが記録されます。

注意事項

トレース機能を実行していると，ログファイルの容量が大きくなり，ODBC プログラムの実行速度が遅くなる可能性があります。

23.2.3　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。

23.2.4　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。

(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。

23.2.5　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。

使用する表の列に対応する COBOL データ定義は，ODBC レコード定義生成を使って生成できます。詳細は，マニュアル「COBOL2002　操作ガイド」を参照してください。

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.`
INTEGER	整数型符号付き 4 バイト 2 進項目	`Ln データ名　　PIC S9(9) USAGE COMP.`
	または，整数型符号なし 4 バイト 2 進項目	`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.`

ODBC SQL データ型	埋め込み変数の型	データ記述
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。

23.2.6　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。

23.2.7　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 'sa' TO ODBC-UID.
MOVE SPACE TO ODBC-PWD.
MOVE 'サンプル' 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=サンプル;UID=sa;PWD=;'
  TO ODBC-CONN.
EXEC SQL
  CONNECT TO :ODBC-CONN
END-EXEC.
```

23.2.8　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。

23.2.9　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。

しかし，この ODBC オプションの設定でのカーソル処理では，次の制限事項があります。

- SQL Server で，カーソル使用中にほかのクエリを実行した場合，次に示すエラーが返されることがあります。

  [Microsoft][ODBC SQL Server Driver] ほかの実行結果のために接続できません。

- ODBC カーソルライブラリで，ディスク容量不足などによりテンポラリファイル※が作成できないとき，次に示すエラーなどが返されます。

  [Microsoft][ODBC Cursor Library] 一般エラー：ファイル バッファーを作成できません。

- ODBC カーソルライブラリが使用するテンポラリファイル※のサイズが上限サイズ（2 ギガバイト）に達すると，プログラムがアクセス違反で異常終了することがあります。

  注※

  テンポラリファイルは，ODBC カーソルライブラリが内部的に作成，または使用するファイルです（COBOL2002 で管理するファイルではありません）。出力先やファイル名については，ODBC カーソルライブラリの仕様を確認してください。

このような問題を回避するため，SQL Server のデータベースに接続する場合は，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

この実行時環境変数を指定して，SQL Server 以外のデータベースに接続する場合の動作は保証しません。

SQL Server の ODBC ドライバを使用した ODBC カーソルライブラリの静的カーソルを使用する場合と，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場合では，動作が異なります。例を次に示します。

詳細およびその他の動作については，SQL Server の ODBC ドライバのリファレンスなどで確認してください。

表 23–6　使用するカーソルによる動作の差異

対象	ODBC カーソルライブラリの静的カーソル使用時	ODBC ドライバのスクロール機能の動的カーソル使用時
SQL Server のカーソル	クライアントカーソル	サーバカーソル
カーソル使用中の他クエリ（SELECT を含む SQL 文）の実行	実行できない。※1	実行できる。
副問い合わせなど複数の SELECT を含む SQL 文の実行	サポートされる。	サポートされない。※2
SQL 文 1 行のサイズ	8,060 バイトに制限されない。	8,060 バイトに制限される。
テンポラリファイル	作成される。※3	作成されない。

注※1

次のエラーが出力されます。

[Microsoft][ODBC SQL Server Driver]ほかの実行結果のために接続できません。

注※2

サーバカーソルではサポートしていませんが，SQL Server のドライバによって暗黙的にカーソルを変更し，該当の SQL 文を正常に実行します。

このとき，次の警告を出力します。また，この暗黙のカーソル変更は，該当の SQL 文だけに有効です。

[Microsoft][ODBC SQL Server Driver]カーソルの種類が変更されました。

[Microsoft][ODBC SQL Server Driver]カーソルの同時性が変更されました。

注※3

テンポラリファイルが作成できない場合，次のエラーが出力されます。

[Microsoft][ODBC Cursor Library] 一般エラー：ファイル バッファーを作成できません。

23.2.10　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

23.2.11　動的 SQL の ODBC API 関数発行の変更

SQL Server 2012 では，メタデータ検出機能が強化されています。この影響で，使用されている SQL Server をバージョンアップすると，SQL Server のデータベースに動的 SQL でアクセスする COBOL プログラムの実行性能が劣化する場合があります。この場合，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，Windows 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，Windows 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，Windows 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 Windows 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 IFIT を使用して Windows に転送します。このとき，Windows 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。

   Windows 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フォルダ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　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

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の埋め込み 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 コマンドを使用してテストができます。

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 Windows 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 OLE2 オートメーション機能

OLE2（Object Linking and Embedding 2）オートメーション機能を使うと，ほかのアプリケーションプログラムの OLE オブジェクトを COBOL プログラムから操作できます。

この章では，OLE2 オートメーション機能の概要と，この機能を使うために COBOL2002 がサポートしている OLE2 オートメーションクライアント機能について説明します。

25.1　OLE2 オートメーション機能の概要

OLE2（Object Linking and Embedding 2）は，データ，およびアプリケーションプログラムでのデータ操作を OLE オブジェクトとして管理する技術です。

OLE2 オートメーション機能は，複数の機能から構成されている OLE2 の一つの機能です。あるアプリケーションプログラム内の OLE オブジェクトを外部のアプリケーションプログラムに公開したり，公開されている OLE オブジェクトを外部のアプリケーションプログラムから操作したりするための手段です。

例えば，Excel のシートを外部の Visual Basic に公開したり，公開している Excel のシートを Visual Basic から操作したりするための手段のことです。前者を OLE2 オートメーションサーバ機能，後者を OLE2 オートメーションクライアント機能と呼びます。

COBOL2002 では，OLE2 オートメーションインタフェースと各種のサービスルーチンの 2 種類の方法によって，OLE2 オートメーションクライアント機能をサポートしています。OLE2 オートメーションクライアント機能については，「25.2　OLE2 オートメーションクライアント機能」を参照してください。

また，日立 COBOL85 からの古い仕様である，サービスルーチンを使った OLE2 オートメーションクライアント機能を使用したい場合は，「付録 D.2　サービスルーチンを使った OLE2 オートメーションクライアント機能」を参照してください。

なお，OLE オブジェクトを公開するアプリケーションプログラムを OLE2 オートメーションサーバアプリケーションといいます。このマニュアルではこのアプリケーションを単に OLE2 サーバと略称します。同様に，OLE2 サーバが公開している OLE オブジェクトを操作するためのアプリケーションプログラムを，OLE2 クライアントと略称します。

25.2　OLE2 オートメーションクライアント機能

COBOL2002 の OLE2 オートメーションクライアント機能は，外部のアプリケーション（OLE2 サーバ）が公開している OLE オブジェクトを，COBOL のプログラムから操作する機能です。

OLE2 オートメーションクライアント機能を使用するには，OLE2 サーバで使用できる OLE オブジェクトの機能をあらかじめ知っておく必要があります。

OLE2 オートメーションクライアント機能を使用するときの基本的な操作手順を次に示します。

1. 操作する OLE オブジェクトを決めて，その OLE オブジェクトを生成するか，または生成されている OLE オブジェクトを取得する。または，生成済みの OLE オブジェクトを直接参照する。
2. OLE メソッドと OLE プロパティを使って OLE オブジェクトを操作する。
3. 操作が終了したら，OLE オブジェクトを解放する。

この節では，Excel を対象としたオブジェクトの操作を例に挙げながら，OLE2 オートメーションクライアント機能の使用方法を説明します。

なお，OLE2 オートメーションクライアント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」「18.　OLE2 オートメーションインタフェース機能」を参照してください。

25.2.1　OLE オブジェクトの生成と取得

OLE2 サーバの OLE オブジェクトを操作するには，その OLE オブジェクトを生成するか，またはすでに生成されている OLE オブジェクトを取得します。

(1)　CREATEOBJ メソッド

INVOKE 文で CREATEOBJ メソッドを使用すると，OLE オブジェクトを生成できます。

Excel の Application オブジェクト※を操作するために OLE オブジェクトを生成する例を次に示します。

OLE オブジェクトの生成例

```
       WORKING-STORAGE SECTION.
       01 AP-OBJ USAGE OBJECT REFERENCE OLE.
       PROCEDURE DIVISION.
           INVOKE 'Excel.Application' 'CREATEOBJ'
                  RETURNING AP-OBJ.
```

注※

Excel では，OLE2 オートメーション機能で操作できる OLE オブジェクトの一つとして Application オブジェクトがあります。この OLE オブジェクトのクラス名は，Excel.Application としてシステム登録ファイルに登録されています。

(2) GETOBJ メソッド

INVOKE 文で GETOBJ メソッドを使用すると，すでに生成されている OLE オブジェクトを取得できます。

すでに生成されている Excel の Application オブジェクトを取得して操作する例を次に示します。

OLE オブジェクトの取得例

```
       WORKING-STORAGE SECTION.
       01 AP-OBJ USAGE OBJECT REFERENCE OLE.
       PROCEDURE DIVISION.
           INVOKE 'Excel.Application' 'GETOBJ'
                  RETURNING AP-OBJ.
```

また，INVOKE 文で GETOBJ メソッドを使用すると，作成済みのファイルから OLE オブジェクトのインスタンスを生成できます。

ハードディスク上にある"SAMPLE1.XLS"という作成済みのファイルを指定して，Excel の Sheet オブジェクト※を生成する例を次に示します。

インスタンスの生成例

```
       WORKING-STORAGE SECTION.
       01 SH-OBJ USAGE OBJECT REFERENCE OLE.
       PROCEDURE DIVISION.
           INVOKE 'Excel.Sheet' 'GETOBJ'
             USING VALUE 'C:¥SAMPLE1.XLS'
             RETURNING SH-OBJ.
```

注※

Sheet オブジェクトのクラス名は，Excel.Sheet としてシステム登録ファイルに登録されています。

25.2.2 OLE メソッドと OLE プロパティの操作

OLE メソッドや OLE プロパティを操作する場合は，生成または取得した OLE オブジェクトからのコンテナやコレクションを含んで定義する必要があります。このときコンテナやコレクションの参照に誤りがあると，実行時にエラーとなります。

コンテナおよびコレクションの概念については，マニュアル「COBOL2002　言語　拡張仕様編」「18.1.1(5)　コレクション」および「COBOL2002　言語　拡張仕様編」「18.1.1(6)　コンテナ」を参照してください。

Open メソッド，Visible プロパティ，Value プロパティ，PrintOut メソッドを使った操作の例を次に示します。

OLE メソッドと OLE プロパティの操作例

```
       WORKING-STORAGE SECTION.
       01 AP-OBJ USAGE OBJECT REFERENCE OLE.
```

```
PROCEDURE DIVISION.
    INVOKE 'Excel.Application' 'CREATEOBJ'
       RETURNING AP-OBJ.
    INVOKE AP-OBJ 'Workbooks.Open'
       USING VALUE 'C:¥SAMPLE1.XLS'. …1.
    SET 'Visible' WITH AP-OBJ TO 1. …2.
    SET 'Workbooks(1).Worksheets(1).Cells(1,1).Value'
          WITH AP-OBJ TO 'ABCD'. …3.
    SET 'Workbooks(1).Worksheets(1).Cells(1,2).Value'
          WITH AP-OBJ TO 'ABCD'.
    SET 'Workbooks(1).Worksheets(1).Cells(1,3).Value'
          WITH AP-OBJ TO 'ABCD'.
    SET 'Workbooks(1).Worksheets(1).Cells(1,4).Value'
          WITH AP-OBJ TO 'ABCD'.
    SET 'Workbooks(1).Worksheets(1).Cells(1,5).Value'
          WITH AP-OBJ TO 'ABCD'.
    INVOKE AP-OBJ
        'Workbooks(1).Worksheets(1). PrintOut'. …4.
```

例の説明

1. 作成済みのファイルをオープンします。
2. シートを表示します。
3. セルに値を設定します。
4. シートを印刷します。

25.2.3 OLE メソッドが返す OLE オブジェクトを利用した参照

OLE メソッドや OLE プロパティの参照には，コンテナやコレクションをすべて指定して参照する方法と，OLE メソッドが返す OLE オブジェクトを起点として参照する方法があります。

(1) コンテナやコレクションをすべて指定して参照する

セルの Value プロパティを使って値を設定するときに，生成した Application オブジェクトからのコンテナやコレクションをすべて指定して参照する例を次に示します。

参照例

```
WORKING-STORAGE SECTION.
01 AP-OBJ USAGE OBJECT REFERENCE OLE.
PROCEDURE DIVISION.
    :
    INVOKE 'Excel.Application' 'CREATEOBJ'
          RETURNING AP-OBJ.
    :
    SET 'Workbooks(1).Worksheets(1).Cells(1,1).Value'
          WITH AP-OBJ TO 'ABCD'.
    SET 'Workbooks(1).Worksheets(1).Cells(1,2).Value'
          WITH AP-OBJ TO 'ABCD'.
```

```
        SET 'Workbooks(1).Worksheets(1).Cells(1,3).Value'
              WITH AP-OBJ TO 'ABCD'.
```

例の説明

Workbooks(1).Worksheets(1).Cells(1,n)（n=1,2,3）がコンテナやコレクションの参照です。

(2) OLE メソッドが返す OLE オブジェクトを起点として参照する

OLE オブジェクトの起点を変えてコンテナやコレクションを参照する例を次に示します。

参照例

```
       WORKING-STORAGE SECTION.
       01 AP-OBJ USAGE OBJECT REFERENCE OLE.
       01 CEL-OBJ USAGE OBJECT REFERENCE OLE. …1.
       01 CEL-IDX PIC 9 VALUE ZERO.
       PROCEDURE DIVISION.
           INVOKE 'Excel.Application' 'CREATEOBJ'
                  RETURNING AP-OBJ.
           :
           PERFORM SET-CELL 3 TIMES.
           :
       SET-CELL SECTION.
           COMPUTE CEL-IDX = CEL-IDX + 1.
           INVOKE AP-OBJ 'Workbooks(1).Worksheets(1).Cells'
                  USING VALUE 1 CEL-IDX
                  RETURNING CEL-OBJ. …1.
           SET 'Value' WITH CEL-OBJ TO 'ABCD'. …2.
```

例の説明

1. Cells メソッドが返すオブジェクトを起点とします。
2. Cells オブジェクトを起点として参照し，値を設定します。

25.2.4 OLE アプリケーションの終了と OLE オブジェクトの解放

OLE2 オートメーションインタフェース機能では，OLE2 サーバで使用できる終了メソッドを使って，アプリケーションを終了します。不要となった OLE オブジェクトは SET 文で解放します。

Excel で，終了メソッドである Quit メソッドを使う例を次に示します。

OLE オブジェクトの終了と OLE オブジェクトの解放例

```
       WORKING-STORAGE SECTION.
       01 AP-OBJ USAGE OBJECT REFERENCE OLE.
       PROCEDURE DIVISION.
           INVOKE 'Excel.Application' 'CREATEOBJ'
               RETURNING AP-OBJ.
           :
           INVOKE AP-OBJ 'Quit'. …1.
           SET AP-OBJ TO NULL.   …2.
```

例の説明

1. アプリケーションプログラムを終了します。
2. OLE オブジェクトを解放します。

注意事項

OLE オブジェクト解放時の注意事項を次に示します。

- いったん解放した OLE オブジェクトを再度参照してはいけません。解放した OLE オブジェクトの OLE メソッドや OLE プロパティを参照すると，実行時にエラーとなります。
- 終了メソッドは，OLE2 サーバによって異なります。また，終了メソッドによっては，OLE オブジェクトの解放も行うものがあり，この場合は SET 文による OLE オブジェクトの解放は不要です。詳細は，各 OLE2 サーバのマニュアルを参照してください。

25.2.5 VARIANT 値と COBOL データのやり取り

(1) 暗黙的な変換

Excel のシートのセルに数値が設定されているときに，このセルから値を取得し，SET 文を使って COBOL の数値項目で値を受け取る例を次に示します。このとき，VARIANT 値から COBOL のデータに暗黙的なデータ変換が行われます。

VARIANT 値の概念については，マニュアル「COBOL2002　言語　拡張仕様編」「18.1.1　言語の概念(OLE2 オートメーションインタフェース機能)」を参照してください。

暗黙的な変換の例

```
       WORKING-STORAGE SECTION.
       01 AP-OBJ USAGE OBJECT REFERENCE OLE.
       01 CDATA-NUM PIC 9(9). …1.
       PROCEDURE DIVISION.
           INVOKE 'Excel.Application' 'CREATEOBJ'
               RETURNING AP-OBJ.
           :
           SET CDATA-NUM TO
               'Workbooks(1).Worksheets(1).Cells(1,1).Value'
                  WITH AP-OBJ. …2.
```

例の説明

1. 外部 10 進項目で項目を定義します。
2. VARIANT 値を受け取ります。

(2) 変換しない場合

VARIANT 値を COBOL データに変換する必要がない場合は，値をバリアントデータ項目で受け取ることで，受け渡した VARIANT 値のポインタを保持できます。

バリアント項目で取得した値を，別のセルにそのまま設定する例を次に示します。

変換しない例

```
       WORKING-STORAGE SECTION.
       01 AP-OBJ USAGE OBJECT REFERENCE OLE.
       01 CDATA-VAR USAGE VARIANT VALUE ZERO. …1.
       PROCEDURE DIVISION.
           INVOKE 'Excel.Application' 'CREATEOBJ'
               RETURNING AP-OBJ.
           :
           SET CDATA-VAR TO
               'Workbooks(1).Worksheets(1).Cells(1,1).Value'
                  WITH AP-OBJ. …2.
           SET 'Workbooks(1).Worksheets(1).Cells(1,2).Value'
                  WITH AP-OBJ TO CDATA-VAR. …3.
```

例の説明

1. バリアントデータで項目を定義します。
2. 値を取得します。
3. 取得した値をそのまま別のセルに設定します。

(3) VARIANT 値の解放

不要となった VARIANT 値は，バリアントデータ項目を初期化することで，解放できます。

不要となった VARIANT 値を解放する例を次に示します。

VARIANT 値の解放例

```
       WORKING-STORAGE SECTION.
       01 AP-OBJ USAGE OBJECT REFERENCE OLE.
       01 CDATA-VAR USAGE VARIANT VALUE ZERO. …1.
       PROCEDURE DIVISION.
           INVOKE 'Excel.Application' 'CREATEOBJ'
               RETURNING AP-OBJ.
           :
           SET CDATA-VAR TO
               'Workbooks(1).Worksheets(1).Cells(1,1).Value'
                  WITH AP-OBJ. …2.
           SET 'Workbooks(1).Worksheets(1).Cells(1,2).Value'
                  WITH AP-OBJ TO CDATA-VAR. …3.
           SET CDATA-VAR TO NULL. …4.
```

例の説明

1. バリアントデータで項目を定義します。
2. 値を設定します。
3. 取得した値をそのまま別のセルに設定します。
4. バリアントデータ項目を解放します。

(4) 明示的な変換

取得した VARIANT 値がどの型かがわからない場合は，いったん VARIANT 値の型をチェックしたあとで，その型に合った COBOL のデータ項目に設定します。VARIANT 値の型は，TYPE-OF-VARIANT 関数で取得できます。TYPE-OF-VARIANT 関数については，マニュアル「COBOL2002　言語　拡張仕様編」「18.5.3　TYPE-OF-VARIANT 関数（OLE2 オートメーションインタフェース機能)」を参照ください。

VARIANT 値の型をチェックしたあとで，該当する COBOL のデータ項目に値を設定する例を次に示します。

明示的な変換の例

```
       WORKING-STORAGE SECTION.
       01 AP-OBJ USAGE OBJECT REFERENCE OLE.
       01 CDATA-VAR USAGE VARIANT VALUE ZERO. …1.
       01 CDATA-COMP PIC S9(10) USAGE COMP.
       01 CDATA-COMP2 USAGE COMP-2.
       01 CDATA-ALP.
         02 FILLER OCCURS 255 DEPENDING ON C-LNG PIC X.
       01 C-LNG PIC 9(4) USAGE COMP VALUE ZERO.
       PROCEDURE DIVISION.
           INVOKE 'Excel.Application' 'CREATEOBJ'
               RETURNING AP-OBJ.
           :
           SET CDATA-VAR TO
               'Workbooks(1).Worksheets(1).Cells(1,1).Value'
                  WITH AP-OBJ. …2.
           EVALUATE FUNCTION TYPE-OF-VARIANT(CDATA-VAR) …3.
             WHEN 2
             WHEN 3 …4.
               COMPUTE CDATA-COMP =
                       FUNCTION VARIANT-TO-INTEGER(CDATA-VAR)
             WHEN 4
             WHEN 5 …4.
               COMPUTE CDATA-COMP2 =
                       FUNCTION VARIANT-TO-NUMERIC(CDATA-VAR)
             WHEN 6
             WHEN 7
             WHEN 8
             WHEN 9
               COMPUTE C-LNG =
                 FUNCTION LENGTH-OF-VARIANT(CDATA-VAR) …5.
               MOVE
                 FUNCTION VARIANT-TO-ALPHANUMERIC(CDATA-VAR)
                 TO CDATA-ALP …4.
           END-EVALUATE.
```

例の説明

1. バリアントデータで項目を定義します。
2. VARIANT 値を取得します。
3. 取得した VARIANT 値の型をチェックします。

4. 該当する COBOL データに振り分けて設定します。

5. 文字型へデータが設定された場合は，長さをチェックします。

(5) バリアントデータ項目の設定

SET 文を使って，CONVERT-TO-VARIANT 関数で作成した VARIANT

値のポインタを，バリアントデータ項目に保持したり，別のバリアントデータ項目に転記できます。

バリアントデータ項目に値を設定する例を次に示します。

バリアントデータ項目の設定例

```
       WORKING-STORAGE SECTION.
       01 CBL-DATA-DATE PIC X(8) VALUE '03/01/01'.
       01 CDATA-VAR1 USAGE VARIANT VALUE ZERO. …1.
       01 CDATA-VAR2 USAGE VARIANT VALUE ZERO. …1.'
       PROCEDURE DIVISION.
           INVOKE 'Excel.Application' 'CREATEOBJ'
                  RETURNING AP-OBJ.
                :
           SET CDATA-VAR1
                  TO FUNCTION
                  CONVERT-TO-VARIANT(7,CBL-DATA-DATE). …2.
           SET CDATA-VAR2 TO CDATA-VAR1. …3.
```

例の説明

1. バリアントデータで項目を定義します。(1.'も同様です)。

2. CONVERT-TO-VARIANT 関数で取得した VARIANT 値のポインタを，1.（CDATA-VAR1）のバリアントデータ項目に設定します。

   CONVERT-TO-VARIANT 関数で取得した値を転記する場合，関数で一時的に作成した VARIANT 値の設定領域は，転記が終了したあとに解放されます。

3. VARIANT 値をコピーした領域（CDATA-VAR1）へのポインタを，1.'（CDATA-VAR2）のバリアントデータ項目に転記します。

   なお，このとき受け取り側作用対象のバリアントデータ項目に VARIANT 値がすでにある場合，該当する VARIANT 領域は解放されます。

   また，このとき送り出し側作用対象のバリアントデータ項目のポインタが NULL の場合，受け取り側作用対象のバリアントデータ項目にも NULL が転記されます。

(6) バリアントデータ項目に EXTERNAL 句を指定する場合

バリアントデータ項目に EXTERNAL 句を指定する場合については，「4.2.2 外部属性（EXTERNAL 句）」の「(2) EXTERNAL 領域に VARIANT データ項目を含む場合の注意事項」を参照してください。

25.2.6　注意事項

OLE2 オートメーションクライアント機能全般に関する注意事項について説明します。

- 多くの OLE オブジェクトを連続して生成すると，リソース不足になることがあります。

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。

26.2　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，「26.3　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。

26.3　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		○※1	
	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		○※1	
	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	
	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		×※2	
	C 言語インタフェース		△	データの共用はできない。
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		○	

種類	機能名	マルチスレッドへの対応	備考
	報告書作成機能	×※1	
	HiRDB による索引編成ファイル	×※1	
	Btrieve（Pervasive.SQL）による索引編成ファイル※3	×※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。
	プリンタへのアクセス（入出力による書式印刷機能）※3	×※1	
	プリンタへのアクセス（GDI モード印刷）	×※1	
	プリンタへのアクセス（ESC/P モード印刷）	×※1	
	ファイルのディスク書き込み保証	○	
	イベントログファイル出力機能	×	
	通信節による画面操作	×※1	
	画面節（WINDOW SECTION）による画面操作	×※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	△※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	○	
	OLE2 オートメーションインタフェース機能	×※1	
	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「26.9.2　呼び出してはいけないサービスルーチン」を参照のこと。
	COBOL 入出力サービスルーチン	×	
	MSMQ アクセス機能	×	
	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※4	○	
	動的長基本項目機能	○	

種類	機能名	マルチスレッドへの対応	備考
	定数長拡張機能	○	
	Java プログラム呼び出し機能	○	
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）

○：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

Windows(x86) COBOL2002 で有効です。

注※4

Windows(x64) COBOL2002 で有効です。

26.4　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。

26.4.1　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。

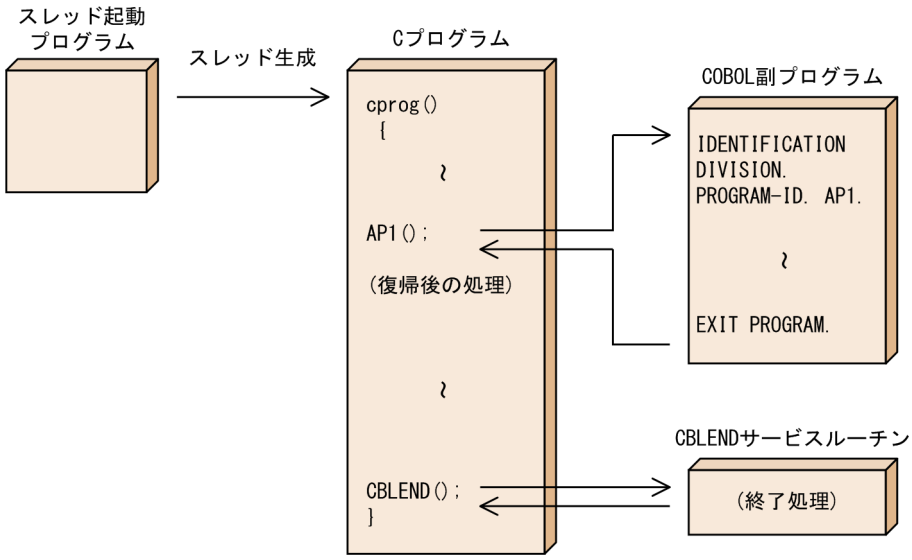
COBOL 主プログラムを終了させるには，STOP RUN 文を実行します。これによって，COBOL 実行環境が終了し，スレッドが終了されます。このとき，RETURN-CODE 特殊レジスタの値がスレッドの終了コードになります。

(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



なお，この方法でマルチスレッド対応 COBOL プログラムを呼び出した場合，呼び出し元のプログラムで例外ハンドラを登録して，例外処理できます。このとき，例外ハンドラブロックで CBLDBGINF サービスルーチンを呼び出せば，COBOL の異常終了時要約情報リストを出力できます。CBLDBGINF サービスルーチンの詳細は，「30.6.1　CBLDBGINF」を参照してください。

26.4.2　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

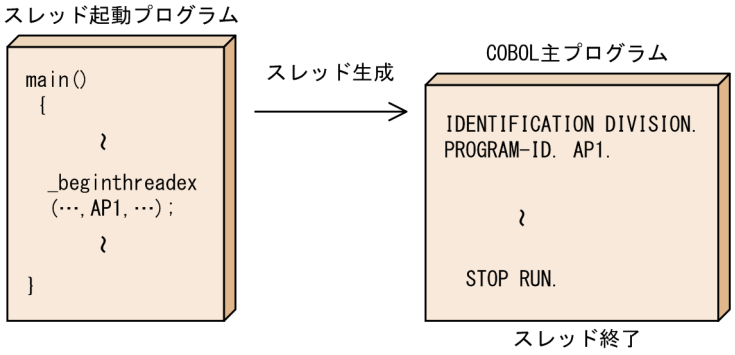
マルチスレッド対応 COBOL プログラムを入り口としてスレッドを開始した場合，このプログラムが終了したときに COBOL 実行環境を終了し，スレッドが終了します。このとき，RETURN-CODE 特殊レジスタに指定した値が，スレッドの終了コードになります。

この方法では，プログラム実行中にハードウェア例外が発生しても，プログラムでの例外処理はできません。ハードウェア例外が発生したときには，システムの例外処理が実行されます。なお，共通例外処理で検出できる例外については，シングルスレッドの COBOL プログラムと同じように，アプリケーションプログラムでの例外処理ができます。

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



26.4.3　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，その中のすべてのスレッドが終了していれば，プロセスが終了します。プログラムの実行が終了するためには，プログラムの実行によって発生したすべてのスレッドが終了する必要があります。

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。

26.4.4　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。

26.5　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，GetCurrentThreadId 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。

26.6　マルチスレッド対応 COBOL プログラムを GUI モードで使用する方法

マルチスレッド対応 COBOL プログラムを GUI モードで使用するときの設定について説明します。

GUI モードで使用するときの設定は，COBOL プログラムがアプリケーションの主プログラムかどうかによって異なります。

26.6.1　COBOL プログラムが主プログラムの場合

COBOL プログラムが主プログラムの場合は，主プログラムのコンパイル時に-Lib,GUI オプションを指定すれば，GUI モードを設定できます。この場合，スレッド上で動作している COBOL プログラムのすべてが，GUI モードで動作します。

26.6.2　COBOL プログラムが主プログラムでない場合

COBOL プログラムが主プログラムでない場合は，CBLGINT サービスルーチンを呼び出せば，GUI モードを設定できます。

CBLGINT サービスルーチンは，一つのスレッドで呼び出すと，すべてのスレッドが GUI モードで動作します。ただし，CBLGINT サービスルーチンを実行する前に COBOL プログラムが実行されているスレッドは，GUI モードになりません。確実にすべてのスレッドを GUI モードにしたいときは，ほかのスレッドを起動する前に，プライマリスレッドで CBLGINT サービスルーチンを呼び出してください。

CBLGINT サービスルーチンの詳細は，「30.4.1　CBLGINT」を参照してください。

26.6.3　注意事項

- GUI モードで表示される COBOL コンソール画面は，スレッドごとに表示されます。
- マルチスレッド対応 COBOL プログラムを実行中に，コンソール画面を閉じて実行を中止すると，そのプログラムが動作していたスレッドが終了されます。ただしこの場合，ファイルを閉じたり，実行時に確保した領域を解放したりする COBOL 実行環境終了処理はされません。
- マルチスレッド環境では，GUI モードで COBOL2002 が表示する画面のヘッダ部分に次の形式の識別子が付けられます。

形式

```
CONSOLE - 実行可能ファイル名(i)
```

i

　スレッド識別子の値

26.7　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

26.7.1　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，「36.2　プログラムの実行環境の設定」を参照してください。

26.7.2　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，「26.7.1　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，アプリケーションエラー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※

SetEnvironmentVariable 関数，または GetEnvironmentVariable 関数

26.8.1　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　操作ガイド」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　操作ガイド」を参照してください。

26.9　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

26.9.1　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。

26.9.2　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。
  - CBLADTRM サービスルーチン
  - CBLBELL サービスルーチン
  - CBLCUR サービスルーチン
  - CBLCNSL サービスルーチン
  - CBLDLTRM サービスルーチン
  - CBLGET サービスルーチン
  - CBLPUT サービスルーチン
  - CBLSETTITLE サービスルーチン
  - CBLSGET サービスルーチン
  - CBLSQLSETOPT サービスルーチン
  - JCPOPUP サービスルーチン
  - OLE2 サービスルーチン
  - MSMQ アクセスサービスルーチン

  OLE2 サービスルーチンについては，「付録 D.2　サービスルーチンを使った OLE2 オートメーションクライアント機能」を参照してください。MSMQ サービスルーチンについては，「27.2　MSMQ アクセスサービスルーチン」を参照してください。その他のサービスルーチンについては，「30.　サービスルーチン」を参照してください。
- 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。

26.9.3　スレッドを起動する関数

マルチスレッド対応 COBOL プログラムでは，STOP RUN 文やエラー発生によるスレッドの終了のときに，_endthreadex 関数を使用しています。そのため，スレッドの起動には_beginthreadex 関数を使用する必要があります。

27 MSMQ アクセス機能

MSMQ（Microsoft Message Queuing Server）アクセス機能を使うと，COBOL2002 から MSMQ を利用できます。

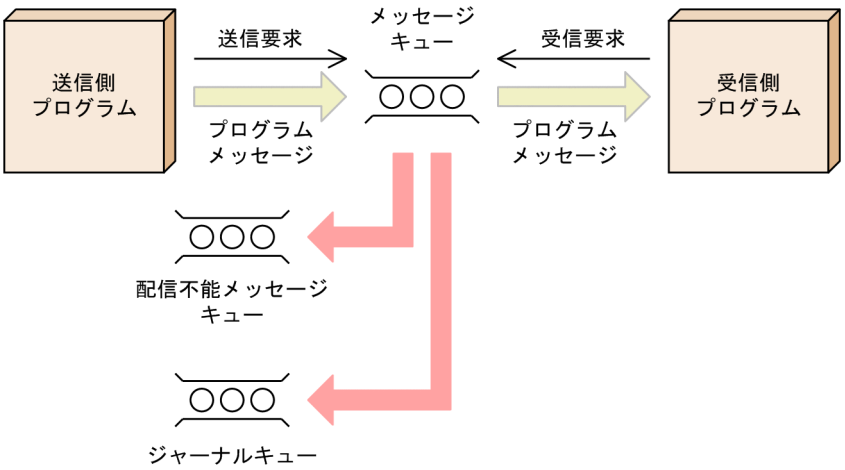
この章では，COBOL2002 から MSMQ を利用する方法について説明します。

27.1　MSMQ の概要

MSMQ（メッセージキュー）を使用すると，プログラム間で通信する際にメッセージを非同期で送受信できます。これをキューイング（非同期通信）といいます。キューイングを使用すると，受信側プログラムに要求した処理の終了を待たないで，送信側プログラムで別の処理を実行できます。

送信側プログラムが MSMQ に送信したメッセージは，いったんキューに格納された後，受信側プログラムに配信されます。

図 27-1　キューイングを使用したメッセージの送受信例



27.1.1　キュー

(1)　キューの種類

キューとは，さまざまな種類のメッセージの一時的な格納場所です。MSMQ には，次のキューがあります。

- パブリックキュー
- プライベートキュー
- ジャーナルキュー
- 配信不能メッセージキュー
- 管理キュー
- 応答キュー
- レポートキュー
- システムキュー

COBOL2002 の MSMQ アクセス機能では，パブリックキュー，およびプライベートキューだけにメッセージを送受信できます。

(2) キューのパス名

キューを参照する場合，キューのパス名を指定します。キューのパス名は，キューがあるコンピュータの名前とキューの名前を結合したものです。例えば，「User1」というコンピュータ上に「Myqueue」というキューを作成した場合，キューのパス名は「User1¥Myqueue」となります。なお，キューがローカルコンピュータ上にある場合は，「.¥Myqueue」のように記述できます。また，プライベートキューの場合は，「.¥PRIVATE$¥Myqueue」のようにキュー名の前に「¥PRIVATE$」を付けたパス名となります。

注意事項

キューのパス名は，大文字小文字が等価とみなされます。また，キューのパス名は，コンピュータの名前を含めた絶対パスで 124 文字以下になることを推奨します。

(3) キューのラベル

キューには，ラベルを付けられます。パブリックキューに付けられたラベルを使用して，パブリックキューのパス名を検索できます。

キューにラベルを付ける場合 CBLMQCREATE サービスルーチンの呼び出し時に，データパラメタ領域の「キューのラベル」データ項目，および「キューのラベルのデータ長」データ項目を設定しておきます。

キューに付けられたラベルからキューのバス名を検索する場合，データパラメタ領域の「キューのラベル」データ項目，および「キューのラベルのデータ長」データ項目を設定した後，CBLMQLOCATE サービスルーチンを呼び出します。CBLMQLOCATE サービスルーチンが正常に終了すると，ラベルが付けられたキューのパス名が，データパラメタ領域の「キューのパス名」データ項目に格納されます。

27.1.2 メッセージ

プログラム間の通信はメッセージの送受信によって行われます。また，コンピュータ間の情報およびデータの送受信も，メッセージの送受信によって行われます。メッセージには，テキストデータおよびバイナリデータを含めることができます。

トランザクションメッセージを使用すると，メッセージの送受信とほかの処理を一つにまとめることができます。トランザクションメッセージはトランザクションキューだけに送信できます。なお，COBOL2002 の MSMQ アクセス機能では，トランザクションメッセージ，およびトランザクションキューは使用できません。

(1) メッセージのラベル

メッセージには，ラベルを付けられます。付けられたラベルはプログラム中で使用できます。

メッセージにラベルを付ける場合，CBLMQSENDMSG サービスルーチンの呼び出し時に，データパラメタ領域の「メッセージのラベル」データ項目，および「メッセージのラベルのデータ長」データ項目を設定しておきます。

メッセージに付けられたラベルは，CBLMQRECEIVEMSG サービスルーチンを呼び出した後，データパラメタ領域の「メッセージのラベル」データ項目に格納されます。

(2) メッセージクラス

メッセージの種類をメッセージの機能または性質によって表したものをメッセージクラスといいます。代表的なメッセージクラスを次に示します。

- 通常メッセージ
- 受信確認メッセージ
- レポートメッセージ
- 応答メッセージ

COBOL2002 で生成できるメッセージは，すべて通常メッセージです。

メッセージクラスは，CBLMQRECEIVEMSG サービスルーチンを呼び出した後，インタフェース領域の「メッセージクラス」データ項目に格納されます。

(3) メッセージの優先順位

メッセージの優先順位によって，緊急なメッセージなど重要度の高いメッセージを，重要度の低いメッセージよりも先に送信できます。メッセージの優先順位は 0～7 の範囲で，0 が最も優先順位が低く，7 が最も優先順位が高くなります。

メッセージの優先順位を設定する場合，CBLMQSENDMSG サービスルーチンの呼び出し時に，インタフェース領域の「優先順位」データ項目を設定しておきます。

(4) メッセージの配信方法

MSMQ には高速と回復可能という 2 種類の配信方法があります。

- 高速メッセージ
  回復可能メッセージに比べてリソース消費量が少なく，配信速度も高速ですが，配信中にコンピュータに障害が発生するとメッセージを回復できません。
- 回復可能メッセージ
  高速メッセージに比べてリソース消費量が多く，配信速度も低速ですが，配信中にコンピュータに障害が発生してもメッセージを回復できます。

メッセージの配信方法を設定する場合，CBLMQSENDMSG サービスルーチンの呼び出し時に，インタフェース領域の「配信方法」データ項目を設定しておきます。

(5) ジャーナリング

メッセージのコピーを格納する処理をジャーナリングといいます。ジャーナリングされたメッセージは，ジャーナルキューに格納されます。

送信メッセージにジャーナリングを設定し，かつメッセージの送信が成功した場合，送信メッセージのコピーが送信側コンピュータのジャーナルキューに格納されます。

ジャーナリングする場合，CBLMQSENDMSG サービスルーチンの呼び出し時に，インタフェース領域の「ジャーナリング」データ項目を設定しておきます。

注意事項

メッセージングのパフォーマンスを向上させるために，すべてのジャーナルキューに格納されているメッセージを MSMQ エクスプローラなどの管理ツールを使って頻繁に削除する必要があります。

(6) 配信不能メッセージ

配信に失敗した，または配信できなくなったメッセージを配信不能メッセージといいます。配信不能メッセージは，配信不能メッセージキューに格納できます。ただし，送信元がメッセージを送信先のキューに送ることを許可されていない場合，配信不能メッセージは配信不能メッセージキューには格納されません。

配信不能メッセージを配信不能メッセージキューに格納する場合，CBLMQSENDMSG サービスルーチンの呼び出し時に，インタフェース領域の「配信不能メッセージ」データ項目を設定しておきます。

注意事項

メッセージングのパフォーマンスを向上させるために，すべての配信不能メッセージキューに格納されているメッセージを MSMQ エクスプローラなどの管理ツールを使って頻繁に削除する必要があります。

27.2　MSMQ アクセスサービスルーチン

COBOL2002 で MSMQ を使ったキューイングを処理するプログラムを作成するには，MSMQ アクセスサービスルーチンを使用します。

MSMQ アクセスサービスルーチンの一覧を次に示します。

表 27-1　MSMQ アクセスサービスルーチンの一覧

サービスルーチン名	機能
CBLMQCREATE	キューを作成する
CBLMQDELETE	キューを削除する
CBLMQOPEN	キューをオープンする
CBLMQCLOSE	キューをクローズする
CBLMQSENDMSG	メッセージを送信する
CBLMQRECEIVEMSG	メッセージを受信する
CBLMQLOCATE	キューのパス名を検索する

MSMQ アクセスサービスルーチンを呼び出すときには，インタフェース領域とデータパラメタ領域を引数に設定します。インタフェース領域は，COBOL プログラムと MSMQ システムとの間で情報をやり取りするための領域です。また，データパラメタ領域は，キューに対して送受信するデータやそのデータ長を格納するための領域です。

MSMQ アクセスサービスルーチンは，キューのパス名，キューのラベル，メッセージのラベル，およびデータ変換フラグに有効を指定して送受信するメッセージデータの文字列を，内部的にワイド文字列（Unicode）にデータ変換，またはワイド文字列（Unicode）からデータ変換して使用します。

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，「30.2　戻り値の使い方」を参照してください。

MSMQ アクセスサービスルーチンを使用するには，「メッセージキュー」が組み込まれている必要があります。

MSMQ アクセスサービスルーチンでは，必要な権限が与えられているプライベートキューとパブリックキューに対して，非トランザクションのメッセージの送受信，およびキューの作成と削除が実行できます。

27.2.1　キューを作成する（CBLMQCREATE）

キューを作成するには，CBLMQCREATE サービスルーチンを呼び出します。

形式

CALL ’CBLMQCREATE’ USING 引数 1 引数 2

引数

引数 1 には，「表 27–2　MSMQ アクセスサービスルーチンで使用するインタフェース領域」の名前を指定します。

引数 2 には，データパラメタ領域の名前を指定します。

インタフェース領域に設定する項目

インタフェース領域中に設定する項目はありません。

インタフェース領域の形式については，「表 27–2　MSMQ アクセスサービスルーチンで使用するインタフェース領域」を参照してください。

データパラメタ領域に設定する項目

作成するキューのパス名，およびキューのラベルを指定するデータ項目を，次の形式で指定します。

記述形式	内容
01 データ名 1.	CALL 文の USING で指定するデータパラメタ領域の名前
02 データ名 2 PIC S9(9) USAGE COMP.	キューのパス名のデータ長
02 データ名 3 PIC X(256).	キューのパス名
02 データ名 4 PIC S9(9) USAGE COMP.	キューのラベルのデータ長
02 データ名 5 PIC X(256).	キューのラベル

戻り値

0：サービスルーチンは正常に終了した。

-1：MSMQ アクセスでエラーを検出した。

-2：MSMQ アクセス以外でエラーを検出した。

また，インタフェース領域に次の項目が返されます。

- MSMQ のエラーコード（戻り値が-1 の場合）
- 表 27–3　詳細コードの内容（戻り値が-2 の場合）

規則

- データパラメタ領域のデータ名 2 には，データ名 3 で指定したキューのパス名のデータ長を，バイト数で指定します。
- データパラメタ領域のデータ名 4 には，データ名 5 で指定したキューのラベルのデータ長を，バイト数で指定します。

27.2.2　キューを削除する（CBLMQDELETE）

キューを削除するには，CBLMQDELETE サービスルーチンを呼び出します。

形式

```
CALL 'CBLMQDELETE' USING 引数1 引数2
```

引数

引数1には，「表 27-2　MSMQ アクセスサービスルーチンで使用するインタフェース領域」の名前を指定します。

引数2には，データパラメタ領域の名前を指定します。

インタフェース領域に設定する項目

インタフェース領域中に設定する項目はありません。

インタフェース領域の形式については，「表 27-2　MSMQ アクセスサービスルーチンで使用するインタフェース領域」を参照してください。

データパラメタ領域に設定する項目

削除するキューのパス名を，次の形式で指定します。

記述形式	内容
01 データ名1.	CALL 文の USING で指定するデータパラメタ領域の名前
02 データ名2 PIC S9(9) USAGE COMP.	キューのパス名のデータ長
02 データ名3 PIC X(256).	キューのパス名

戻り値

0：サービスルーチンは正常に終了した。

-1：MSMQ アクセスでエラーを検出した。

-2：MSMQ アクセス以外でエラーを検出した。

また，インタフェース領域に次の項目が返されます。

- MSMQ のエラーコード（戻り値が-1 の場合）
- 表 27-3　詳細コードの内容（戻り値が-2 の場合）

規則

- データパラメタ領域のデータ名2には，データ名3で指定したキューのパス名のデータ長を，バイト数で指定します。

27.2.3　キューをオープンする（CBLMQOPEN）

メッセージを送受信するためにキューをオープンするには，CBLMQOPEN サービスルーチンを呼び出します。

形式

```
CALL 'CBLMQOPEN' USING 引数1 引数2
```

引数

引数 1 には，「表 27-2　MSMQ アクセスサービスルーチンで使用するインタフェース領域」の名前を指定します。

引数 2 には，データパラメタ領域の名前を指定します。

インタフェース領域に設定する項目

- アクセスモード

  メッセージを送信する場合（SEND モード）は 2 を，メッセージを受信する場合（RECEIVE モード）は 1 を指定します。

インタフェース領域の形式については，「表 27-2　MSMQ アクセスサービスルーチンで使用するインタフェース領域」を参照してください。

データパラメタ領域に設定する項目

オープンするキューのパス名を，次の形式で指定します。

記述形式	内容
01 データ名 1.	CALL 文の USING で指定するデータパラメタ領域の名前
02 データ名 2 PIC S9(9) USAGE COMP.	キューのパス名のデータ長
02 データ名 3 PIC X(256).	キューのパス名

戻り値

0：サービスルーチンは正常に終了した。

-1：MSMQ アクセスでエラーを検出した。

-2：MSMQ アクセス以外でエラーを検出した。

また，インタフェース領域に次の項目が返されます。

- MSMQ のエラーコード（戻り値が-1 の場合）
- 表 27-3　詳細コードの内容（戻り値が-2 の場合）

規則

- データパラメタ領域のデータ名 2 には，データ名 3 で指定したキューのパス名のデータ長を，バイト数で指定します。

27.2.4　キューをクローズする（CBLMQCLOSE）

オープンしているキューをクローズするには，CBLMQCLOSE サービスルーチンを呼び出します。

形式

```
CALL ’CBLMQCLOSE’ USING 引数 1 引数 2
```

引数

引数 1 には，キューをオープンしたときに CBLMQOPEN サービスルーチンに指定した「表 27-2 MSMQ アクセスサービスルーチンで使用するインタフェース領域」の名前を指定します。

引数 2 には，データパラメタ領域の名前を指定します。

インタフェース領域に設定する項目

インタフェース領域中に設定する項目はありません。

インタフェース領域の形式については，「表 27-2　MSMQ アクセスサービスルーチンで使用するインタフェース領域」を参照してください。

データパラメタ領域に設定する項目

クローズするキューのパス名を，次の形式で指定します。

記述形式	内容
01 データ名 1.	CALL 文の USING で指定するデータパラメタ領域の名前
02 データ名 2 PIC S9(9) USAGE COMP.	キューのパス名のデータ長
02 データ名 3 PIC X(256).	キューのパス名

戻り値

0：サービスルーチンは正常に終了した。

-1：MSMQ アクセスでエラーを検出した。

-2：MSMQ アクセス以外でエラーを検出した。

また，インタフェース領域に次の項目が返されます。

- MSMQ のエラーコード（戻り値が-1 の場合）
- 表 27-3　詳細コードの内容（戻り値が-2 の場合）

規則

- データパラメタ領域のデータ名 2 には，データ名 3 で指定したキューのパス名のデータ長を，バイト数で指定します。

27.2.5　メッセージを送信する（CBLMQSENDMSG）

キューへメッセージを送信するには，CBLMQSENDMSG サービスルーチンを呼び出します。

形式

```
CALL 'CBLMQSENDMSG' USING 引数 1 引数 2
```

引数

引数 1 には，キューをオープンしたときに CBLMQOPEN サービスルーチンに指定した「表 27-2 MSMQ アクセスサービスルーチンで使用するインタフェース領域」の名前を指定します。

引数 2 には，データパラメタ領域の名前を指定します。

インタフェース領域に設定する項目

- 優先順位

  メッセージの優先順位を 0～7 の範囲で指定します。

- 配信方法

  メッセージの配信方法を指定します。

  高速メッセージとして配信する場合は 0 を，回復可能メッセージとして配信する場合は 1 を指定します。

- ジャーナリング

  メッセージの送信に成功した場合，ジャーナルキューにメッセージのコピーを格納するかどうかを指定します。

  メッセージのコピーを格納する場合は 1 を，格納しない場合は 0 を指定します。

- 配信不能メッセージ

  メッセージの送信に失敗した場合，配信不能メッセージキューにメッセージのコピーを格納するかどうかを指定します。

  メッセージのコピーを格納する場合は 1 を，格納しない場合は 0 を指定します。

- データ変換フラグ

  メッセージデータをデータ変換して，テキストデータとしてキューへ送信するかどうかを指定します。

  テキストデータに変換して送信する場合は 1 を，そのまま送信する場合は 0 を指定します。

インタフェース領域の形式については，「表 27-2　MSMQ アクセスサービスルーチンで使用するインタフェース領域」を参照してください。

データパラメタ領域に設定する項目

送信するメッセージデータ，およびメッセージのラベルを，次の形式で指定します。

記述形式	内容
01 データ名 1.	CALL 文の USING で指定するデータパラメタ領域の名前
02 データ名 2 PIC S9(9) USAGE COMP.	メッセージのラベルのデータ長
02 データ名 3 PIC X(512).	メッセージのラベルのパス名
02 データ名 4 PIC S9(9) USAGE COMP.	メッセージデータのデータ長
02 データ名 5 PIC X(n).	メッセージデータ

（凡例）

n：メッセージデータのデータ長

戻り値

0：サービスルーチンは正常に終了した。

-1：MSMQ アクセスでエラーを検出した。

-2：MSMQ アクセス以外でエラーを検出した。

また，インタフェース領域に次の項目が返されます。

- MSMQ のエラーコード（戻り値が-1 の場合）
- 表 27-3　詳細コードの内容（戻り値が-2 の場合）

規則

- データパラメタ領域のデータ名 2 には，データ名 3 で指定したメッセージのラベルのデータ長を，バイト数で指定します。
- データパラメタ領域のデータ名 5 の領域長には，メッセージデータのデータ長を指定します。指定できる最大データ長は，4,096,000 バイトです。
- データパラメタ領域のデータ名 4 には，データ名 5 で指定したメッセージデータのデータ長を，バイト数で指定します。

  ただし，インタフェース領域のデータ変換フラグに有効（テキストデータに変換）を指定した場合，CBLMQSENDMSG サービスルーチンはメッセージデータをワイド文字列（Unicode）に変換してキューに送信します。したがって，実際にキューに格納されるメッセージ本文のデータサイズは，データ名 4 で指定したメッセージデータのデータ長よりも大きくなり，最大で（メッセージデータのデータ長＋ 1）×2 バイトとなります。なお，実際にキューに格納されたメッセージ本文のデータサイズは，MSMQ エクスプローラなどで確認できます。
- このサービスルーチンを実行する場合，あらかじめ CBLMQOPEN サービスルーチンを使って，メッセージを送信するキューを SEND モードでオープンしておく必要があります。

27.2.6　メッセージを受信する（CBLMQRECEIVEMSG）

キューからメッセージを受信するには，CBLMQRECEIVEMSG サービスルーチンを呼び出します。

形式

```
CALL ’CBLMQRECEIVEMSG’ USING 引数 1 引数 2
```

引数

引数 1 には，キューをオープンしたときに CBLMQOPEN サービスルーチンに指定した「表 27-2 MSMQ アクセスサービスルーチンで使用するインタフェース領域」の名前を指定します。

引数 2 には，データパラメタ領域の名前を指定します。

インタフェース領域に設定する項目

- 受信待ち時間

  キューにメッセージが配信されるのを待つ時間を msec 単位で指定します。

  -1 を指定した場合，キューにメッセージが配信されるまで待ち続けます。
- データ変換フラグ

  キューから受信したメッセージ本文のデータをデータ変換して，テキストデータとして格納するかどうかを指定します。

  テキストデータに変換して格納する場合は 1 を，そのまま格納する場合は 0 を指定します。

インタフェース領域の形式については，「表 27-2　MSMQ アクセスサービスルーチンで使用するインタフェース領域」を参照してください。

データパラメタ領域に設定する項目

受信したメッセージデータ，およびメッセージのラベルが，次の形式で格納されます。

記述形式	内容
01 データ名 1.	CALL 文の USING で指定するデータパラメタ領域の名前
02 データ名 2 PIC S9(9) USAGE COMP.	メッセージのラベルのデータ長
02 データ名 3 PIC X(512).	メッセージのラベルのパス名
02 データ名 4 PIC S9(9) USAGE COMP.	メッセージデータのデータ長
02 データ名 5 PIC X(n).	メッセージデータ

（凡例）

n：メッセージデータのデータ長

戻り値

100：受信待ち時間の間にメッセージが配信されなかった。

2：受信メッセージを格納中に警告エラーを検出した。

1：メッセージを格納するデータ領域長が不足していた。

0：サービスルーチンは正常に終了した。

-1：MSMQ アクセスでエラーを検出した。

-2：MSMQ アクセス以外でエラーを検出した。

また，インタフェース領域に次の項目が返されます。

- MSMQ のエラーコード（戻り値が-1 の場合）
- 表 27-3　詳細コードの内容（戻り値が 2 または-2 の場合）
- メッセージクラス

  メッセージクラスが返されます。

  通常メッセージを受信した場合は，LOW-VALUE が返されます。

  それ以外のメッセージを受信した場合は，メッセージクラスのコードが返されます。詳細は，MSMQ のマニュアルを参照してください。
- 優先順位

  メッセージの優先順位が 0～7 の範囲で返されます。
- 配信方法

  メッセージの配信方法が返されます。

  高速メッセージとして配信された場合は 0 が，回復不能メッセージとして配信された場合は 1 が返されます。
- ジャーナリング

受信したメッセージのジャーナリングフラグ（ジャーナリングするかどうか）の設定が返されます。

ジャーナリングフラグが設定されていた場合は 1 が，設定されていなかった場合は 0 が返されます。

- 配信不能メッセージ

  受信したメッセージの配信不能メッセージフラグ（配信不能キューへ格納するかどうか）の設定が返されます。

  配信不能メッセージフラグが設定されていた場合は 1 が，設定されていなかった場合は 0 が返されます。

規則

- データパラメタ領域のデータ名 2 には受信したメッセージのラベルのデータ長が，バイト数で返されます。また，データ名 3 には受信したメッセージのラベルが返されます。
- データパラメタ領域のデータ名 4 には，データ名 5 のメッセージデータのデータ長を指定します。指定できる最大データ長は，4,096,000 バイトです。指定するデータ長には，実際にキューから受信するメッセージ本文のデータサイズ分が必要です。CBLMQSENDMSG サービスルーチンでメッセージを送信するとき，実際にキューに格納されるメッセージ本文のデータサイズについては，「27.2.5　メッセージを送信する（CBLMQSENDMSG）」の規則を参照してください。また，MSMQ エクスプローラなどでメッセージ本文のデータサイズを確認することもできます。
- データパラメタ領域のデータ名 5 のデータ長よりキューから受信したメッセージ本文のデータサイズが短かった場合，データ名 5 に受信したメッセージ本文のデータが，データ名 4 に受信したメッセージ本文のデータサイズが，それぞれ返されます。ただし，インタフェース領域のデータ変換フラグに有効（テキストデータに変換）を指定した場合，データ名 5 には受信したメッセージ本文のデータがテキストデータに変換されて格納され，データ名 4 にはデータ名 5 に格納されたテキストデータのデータ長が返されます。
- データパラメタ領域のデータ名 5 のデータ長よりキューから受信したメッセージ本文のデータサイズが長かった場合，データ名 4 に受信したメッセージ本文のデータサイズが返され，サービスルーチンの戻り値が 1 となります。このとき，データ名 2，データ名 3，データ名 5，およびインタフェース領域には何も返されません。
- キューから受信したメッセージのラベル，またはメッセージ本文のデータを格納中にサービスルーチンがデータエラーを検出した場合，サービスルーチンの戻り値が 2 となります。この場合，キューから受信したメッセージのラベル，またはメッセージ本文のデータは，テキストデータに変換されないで格納されます。
- このサービスルーチンを実行する場合，あらかじめ CBLMQOPEN サービスルーチンを使って，メッセージを受信するキューを RECEIVE モードでオープンしておく必要があります。

27.2.7　キューのパス名を検索する（CBLMQLOCATE）

パブリックキューのラベルからパス名を検索するには，CBLMQLOCATE サービスルーチンを呼び出します。

形式

```
CALL ’CBLMQLOCATE’ USING 引数 1 引数 2
```

引数

引数 1 には，「表 27-2　MSMQ アクセスサービスルーチンで使用するインタフェース領域」の名前を指定します。

引数 2 には，データパラメタ領域の名前を指定します。

インタフェース領域に設定する項目

- 検索強制終了フラグ

  すべてのパス名の検索が終了する前に検索を終了したい場合，この項目に 1 を指定して再度 CBLMQLOCATE サービスルーチンを呼び出します。

  それ以外の場合は 0 を指定します。

インタフェース領域の形式については，「表 27-2　MSMQ アクセスサービスルーチンで使用するインタフェース領域」を参照してください。

データパラメタ領域に設定する項目

検索するキューのラベル名，および検索結果のキューのパス名を格納するデータ項目を，次の形式で指定します。

記述形式	内容
01 データ名 1.	CALL 文の USING で指定するデータパラメタ領域の名前
02 データ名 2 PIC S9(9) USAGE COMP.	キューのパス名のデータ長
02 データ名 3 PIC X(256).	キューのパス名
02 データ名 4 PIC S9(9) USAGE COMP.	キューのラベルのデータ長
02 データ名 5 PIC X(256).	キューのラベル

戻り値

100：すべてのパス名の検索が終了した。

2：パス名を格納中にデータエラーを検出した。

1：パス名を検索中である。

0：サービスルーチンは正常に終了した。

-1：MSMQ アクセスでエラーを検出した。

-2：MSMQ アクセス以外でエラーを検出した。

また，インタフェース領域に次の項目が返されます。

- MSMQ のエラーコード（戻り値が-1 の場合）
- 表 27-3　詳細コードの内容（戻り値が 2 または-2 の場合）

規則

- データパラメタ領域のデータ名 4 には，データ名 5 で指定したキューのラベルのデータ長を，バイト数で指定します。

- キューのパス名の検索に成功した場合，データパラメタ領域のデータ名 2 およびデータ名 3 に，検索したキューのパス名のデータ長，およびパス名が返されます。また，CBLMQLOCATE サービスルーチンは，戻り値として 1 を返します。

  同じラベルを持つキューのパス名を引き続き検索するには，再度 CBLMQLOCATE サービスルーチンを呼び出してください。サービスルーチンを呼び出すたびに，順番にキューのパス名が検索され，データパラメタ領域のデータ名 2 およびデータ名 3 に上書きで格納されます。

  目的のパス名を取得できた場合など，パス名の検索を終了したい場合は，インタフェース領域の検索終了強制フラグに 1 を指定して，再度 CBLMQLOCATE サービスルーチンを呼び出してください。戻り値として 0 が返され，検索が終了します。この方法で正しく検索を終了させないと，ほかの MSMQ サービスルーチンを呼び出したときにエラーとなります。
- すべてのパス名の検索が終了すると，戻り値として 100 が返され，検索が終了します。この場合，検索終了強制フラグを使用して検索を終了させる必要はありません。
- パス名の格納中にデータエラーが検出された場合，戻り値として 2 が返されます。この場合，パス名はデータエラーとなったデータ形式で格納されます。
- 変換したパス名がデータパラメタ領域のデータ名 3 よりも長い場合，戻り値として 2 が返されます。この場合，変換したパス名はデータ名 3 の領域長分だけ格納されます。

27.3 MSMQ アクセスサービスルーチンのインタフェース

インタフェース領域は，COBOL プログラムと MSMQ システムとの間で情報をやり取りするための連絡領域です。インタフェース領域に必要な情報を設定してサービスルーチンを呼び出し，また，処理結果をインタフェース領域から取得します。

MSMQ アクセスサービスルーチンで使用するインタフェース領域の形式を次に示します。

表 27-2 MSMQ アクセスサービスルーチンで使用するインタフェース領域

記述形式	内容	CREATE	DELETE	OPEN	CLOSE	SEND	RECV	LOCATE
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							
02 データ名 02 PIC X(8).	MSMQ のエラーコード※1	△	△	△	△	△	△	△
02 データ名 03 PIC S9(9) USAGE COMP.	詳細コード※2	△	△	△	△	△	△	△
02 データ名 04 PIC X.	検索強制終了フラグ （0：無効，1：有効）							○
02 データ名 05 PIC X.	アクセスモード （1：RECEIVE，2：SEND）			○				
02 データ名 06 PIC X(2).	メッセージクラス						△	
02 データ名 07 PIC S9(9) USAGE COMP.	受信待ち時間						○	
02 データ名 08 PIC 9(4) USAGE COMP.	優先順位（0～7）					○	△	
02 データ名 09 PIC X.	配信方法 （0：高速，1：回復可能）					○	△	
02 データ名 10 PIC X.	ジャーナリング （0：無効，1：有効）					○	△	
02 データ名 11 PIC X.	配信不能メッセージ （0：無効，1：有効）					○	△	
02 データ名 12 PIC X.	データ変換フラグ （0：無効，1：有効）					○	○	
02 データ名 13 PIC X(102).	予約領域	×	×	×	×	×	×	×

（凡例）

○：サービスルーチンの呼び出し時，必ず設定する項目

△：サービスルーチンの完了時，リターン情報として設定される項目

×：初期化後に変更してはいけない項目

空欄：該当しない

CREATE：CBLMQCREATE サービスルーチン

DELETE：CBLMQDELETE サービスルーチン

OPEN：CBLMQOPEN サービスルーチン

CLOSE：CBLMQCLOSE サービスルーチン

SEND：CBLMQSENDMSG サービスルーチン

RECV：CBLMQRECEIVEMSG サービスルーチン

LOCATE：CBLMQLOCATE サービスルーチン

注※1

サービスルーチンの戻り値が-1 の場合に設定されます。エラーコードの内容については MSMQ のマニュアルを参照してください。

注※2

サービスルーチンの戻り値が 2 または-2 の場合に設定されます。

詳細コードの内容を，次に示します。

表 27-3　詳細コードの内容

戻り値	詳細コード	エラーの要因	該当するサービスルーチン	対処
2	1	受信メッセージのラベルが不正である。	CBLMQRECEIVEMSG	メッセージのラベルをテキストデータとして正しい値で再送信し，メッセージを再受信する。 (サービスルーチンの動作) サービスルーチンは，受信したメッセージのラベルをテキストデータに変換しないで格納した。
	2	バイナリデータとして送信されたメッセージを受信し，テキストデータへの変換に失敗した。	CBLMQRECEIVEMSG	• メッセージを再送信し，データ変換フラグを無効にして再受信する。 • メッセージをテキストデータとして正しい値で再送信し，再受信する。 (サービスルーチンの動作) サービスルーチンは，受信したメッセージデータをテキストデータに変換しないで格納した。
	3	テキストデータとして送信されたメッセージを受信したが，テキストデータへの変換に失敗した。	CBLMQRECEIVEMSG	メッセージをテキストデータとして正しい値で再送信し，再受信する。 (サービスルーチンの動作)

戻り値	詳細コード	エラーの要因	該当するサービスルーチン	対処
				サービスルーチンは，受信したメッセージデータをテキストデータに変換しないで格納した。
	4	受信メッセージのラベルが不正である。かつ，テキストデータとして送信されていないメッセージを受信し，テキストデータへの変換に失敗した。	CBLMQRECEIVEMSG	• メッセージのラベルをテキストデータとして正しい値で再送信し，データ変換フラグを無効にして再受信する。 • メッセージのラベル，およびメッセージデータをテキストデータとして正しい値で再送信し，再受信する。 (サービスルーチンの動作) サービスルーチンは，受信したメッセージのラベル，およびメッセージデータをテキストデータに変換しないで格納した。
	5	受信メッセージのラベルが不正である。かつ，テキストデータとして送信されたメッセージを受信したが，テキストデータへの変換に失敗した。	CBLMQRECEIVEMSG	メッセージのラベル，およびメッセージデータをテキストデータとして正しい値で再送信し，再受信する。 (サービスルーチンの動作) サービスルーチンは，受信したメッセージのラベル，およびメッセージデータをテキストデータに変換しないで格納した。
	6	検索したキューのパス名が不正である。	CBLMQLOCATE	検索したキューのパス名をテキストデータとして正しい値で再作成し，再度キューのパス名を検索する。 (サービスルーチンの動作) サービスルーチンは，検索したキューのパス名をテキストデータに変換しないで格納した。
	7	検索したキューのパス名が長過ぎる。	CBLMQLOCATE	検索したキューのパス名が，絶対パスで 256 バイト以下になるようにキューを再作成し，再度キューのパス名を検索する。 (サービスルーチンの動作) サービスルーチンは，検索したキューのパス名の先頭 256 バイトだけを格納した。
-2	40	キューをオープンしていない。	CBLMQCLOSE CBLMQSENDMSG	• キューをオープンしてから CBLMQCLOSE サービスルーチンを呼び出す。

戻り値	詳細コード	エラーの要因	該当するサービスルーチン	対処
			CBLMQRECEIVEMSG	• キューを SEND アクセスモードでオープンしてから CBLMQSENDMSG サービスルーチンを呼び出す。 • キューを RECEIVE アクセスモードでオープンしてから CBLMQRECEIVEMSG サービスルーチンを呼び出す。
	41	キューをクローズしていない。	CBLMQCREATE CBLMQDELETE CBLMQOPEN CBLMQLOCATE	キューをクローズしてからサービスルーチンを呼び出す。
	42	キューのパス名の検索が終了していない。	CBLMQCREATE CBLMQDELETE CBLMQOPEN CBLMQCLOSE CBLMQSENDMSG CBLMQRECEIVEMSG	CBLMQLOCATE サービスルーチンでの検索を終了してから，ほかの MSMQ アクセスサービスルーチンを呼び出す。
	50	インタフェース領域の必ず設定する項目に，規定の値が設定されていない。	CBLMQOPEN CBLMQSENDMSG CBLMQRECEIVEMSG CBLMQLOCATE	引数で指定したインタフェース領域に，規定の値を設定してサービスルーチンを呼び出す。
	51	データパラメタのデータ長が不正である。	CBLMQCREATE CBLMQDELETE CBLMQOPEN CBLMQSENDMSG CBLMQLOCATE	引数で指定したデータパラメタ領域のデータ長に正しい値を設定してサービスルーチンを呼び出す。
	52	データパラメタのデータが不正である。	CBLMQCREATE CBLMQDELETE CBLMQOPEN CBLMQSENDMSG CBLMQLOCATE	引数で指定したデータパラメタ領域のメッセージデータに正しい値を設定してサービスルーチンを呼び出す。
	60	コード変換のための環境が整っていない（Unicode 機能を使用する場合）。	CBLMQCREATE CBLMQDELETE CBLMQOPEN CBLMQCLOSE CBLMQSENDMSG CBLMQRECEIVEMSG CBLMQLOCATE	コード変換ライブラリが正しくインストールされているか，または使用するコード変換ライブラリが前提バージョンを満たしているかを確認する。

戻り値	詳細コード	エラーの要因	該当するサービスルーチン	対処
	61	コード変換中にエラーが発生した（Unicode 機能を使用する場合）。	CBLMQCREATE CBLMQDELETE CBLMQOPEN CBLMQCLOSE CBLMQSENDMSG CBLMQRECEIVEMSG CBLMQLOCATE	UCS-4 の範囲（UCS-2 の範囲を含む）の文字コードを使用しているか確認する。
	99	サービスルーチンを実行するためのメモリが不足している。	CBLMQOPEN CBLMQSENDMSG CBLMQRECEIVEMSG CBLMQLOCATE	削除できるプログラムなどを削除して再実行する。
	負数	上記以外の予期しないエラーを検出した。	CBLMQCREATE CBLMQDELETE CBLMQOPEN CBLMQCLOSE CBLMQSENDMSG CBLMQRECEIVEMSG CBLMQLOCATE	当社保守員に連絡する。このとき，詳細コードの値，ソースファイル，登録集原文，およびコンパイルリストファイルが資料として必要になる。

規則

- インタフェース領域は，最初の MSMQ アクセスサービスルーチンを呼び出す前に次の値で初期化しておく必要があります。
  - ・英数字項目：SPACE（ただし，予約領域は LOW-VALUE）
  - ・数字項目：ZERO

コーディング例

MSMQ アクセスサービスルーチンのインタフェース領域のコーディング例を次に示します。

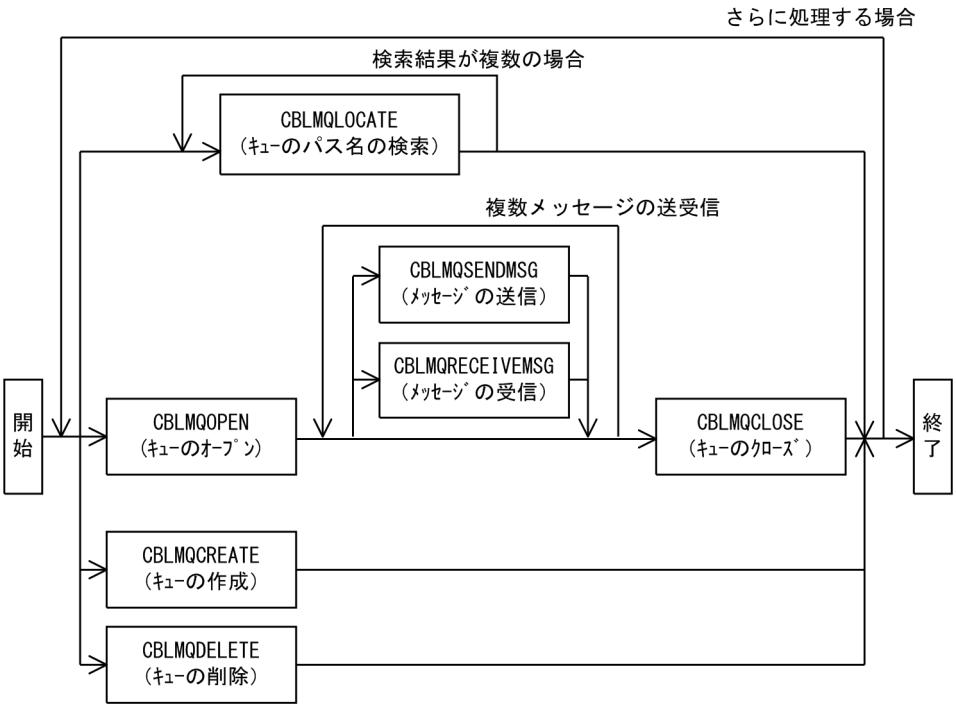
```
       01 CBLMQINF.
         02 MSMQ-ERRCD       PIC X(8)   VALUE SPACE.
         02 DETAIL-CD        PIC S9(9)  USAGE COMP
                                       VALUE ZERO.
         02 LOCATE-END       PIC X      VALUE SPACE.
         02 QUEUE-ACCESS     PIC X      VALUE SPACE.
         02 MQMSG-CLASS      PIC X(2)   VALUE SPACE.
         02 MQMSG-TIMEOUT    PIC S9(9)  USAGE COMP
                                       VALUE ZERO.
         02 MQMSG-PRIORITY   PIC 9(4)   USAGE COMP
                                       VALUE ZERO.
         02 MQMSG-DELIVERY   PIC X      VALUE SPACE.
         02 MQMSG-JOURNAL    PIC X      VALUE SPACE.
         02 MQMSG-DEADLETTER PIC X      VALUE SPACE.
```

```
        02 MSGDATA-CONV     PIC X      VALUE SPACE.
        02 CBLMQ-RESERVE    PIC X(102) VALUE LOW-VALUE.
```

27.4　MSMQ アクセスサービスルーチンの実行順序

MSMQ アクセスサービスルーチンの実行順序を，次に示します。

図 27-2　MSMQ アクセスサービスルーチンの実行順序



27.5　MSMQ アクセスサービスルーチンの使用例

27.5.1　キューを作成するコーディングの例

'.¥SampleQueue'というキューを作成するコーディングの例を次に示します。

```
      :
DATA DIVISION.
WORKING-STORAGE SECTION.
01 CBLMQINF.
  02 MSMQ-ERRCD       PIC X(8)   VALUE SPACE.
  02 DETAIL-CD        PIC S9(9)  USAGE COMP
                                  VALUE ZERO.
  02 LOCATE-END       PIC X      VALUE SPACE.
  02 QUEUE-ACCESS     PIC X      VALUE SPACE.
  02 MQMSG-CLASS      PIC X(2)   VALUE SPACE.
  02 MQMSG-TIMEOUT    PIC S9(9)  USAGE COMP
                                  VALUE ZERO.
  02 MQMSG-PRIORITY   PIC 9(4)   USAGE COMP
                                  VALUE ZERO.
  02 MQMSG-DELIVERY   PIC X      VALUE SPACE.
  02 MQMSG-JOURNAL    PIC X      VALUE SPACE.
  02 MQMSG-DEADLETTER PIC X      VALUE SPACE.
  02 MSGDATA-CONV     PIC X      VALUE SPACE.
  02 CBLMQ-RESERVE    PIC X(102) VALUE LOW-VALUE.
01 CREATE-PARM.
  02 PATH-LEN  PIC S9(9) USAGE COMP VALUE 13.
  02 PATH-NAME PIC X(256)     VALUE '.¥SampleQueue'.
  02 LABEL-LEN PIC S9(9) USAGE COMP VALUE 12.
  02 LABEL-DAT PIC X(256)     VALUE 'Sample Queue'.
  :
PROCEDURE DIVISION.
     :
    INITIALIZE CBLMQINF.
    MOVE LOW-VALUE TO CBLMQ-RESERVE OF CBLMQINF.

    CALL 'CBLMQCREATE' USING CBLMQINF CREATE-PARM.
    IF ZERO NOT = RETURN-CODE THEN
      DISPLAY 'QUEUE CREATE ERROR'
    END-IF.
    :
```

27.5.2　キューを削除するコーディングの例

'.¥SampleQueue'というキューを削除するコーディングの例を次に示します。

```
      :
DATA DIVISION.
WORKING-STORAGE SECTION.
01 CBLMQINF.
```

```
        02 MSMQ-ERRCD       PIC X(8)   VALUE SPACE.
        02 DETAIL-CD        PIC S9(9)  USAGE COMP
                                       VALUE ZERO.
        02 LOCATE-END       PIC X      VALUE SPACE.
        02 QUEUE-ACCESS     PIC X      VALUE SPACE.
        02 MQMSG-CLASS      PIC X(2)   VALUE SPACE.
        02 MQMSG-TIMEOUT    PIC S9(9)  USAGE COMP
                                       VALUE ZERO.
        02 MQMSG-PRIORITY   PIC 9(4)   USAGE COMP
                                       VALUE ZERO.
        02 MQMSG-DELIVERY   PIC X      VALUE SPACE.
        02 MQMSG-JOURNAL    PIC X      VALUE SPACE.
        02 MQMSG-DEADLETTER PIC X      VALUE SPACE.
        02 MSGDATA-CONV     PIC X      VALUE SPACE.
        02 CBLMQ-RESERVE    PIC X(102) VALUE LOW-VALUE.
      01 DELETE-PARM.
        02 PATH-LEN  PIC S9(9) USAGE COMP VALUE 13.
        02 PATH-NAME PIC X(256)     VALUE '.¥SampleQueue'.
        :
      PROCEDURE DIVISION.
          :
          INITIALIZE CBLMQINF.
          MOVE LOW-VALUE TO CBLMQ-RESERVE OF CBLMQINF.

          CALL 'CBLMQDELETE' USING CBLMQINF DELETE-PARM.
          IF ZERO NOT = RETURN-CODE THEN
            DISPLAY 'QUEUE DELETE ERROR'
          END-IF.
          :
```

27.5.3　メッセージを送信するコーディングの例

'.¥SampleQueue'というキューへメッセージを送信するコーディングの例を次に示します。

```
        :
      DATA DIVISION.
      WORKING-STORAGE SECTION.
      01 CBLMQINF.
        02 MSMQ-ERRCD       PIC X(8)   VALUE SPACE.
        02 DETAIL-CD        PIC S9(9)  USAGE COMP
                                       VALUE ZERO.
        02 LOCATE-END       PIC X      VALUE SPACE.
        02 QUEUE-ACCESS     PIC X      VALUE SPACE.
        02 MQMSG-CLASS      PIC X(2)   VALUE SPACE.
        02 MQMSG-TIMEOUT    PIC S9(9)  USAGE COMP
                                       VALUE ZERO.
        02 MQMSG-PRIORITY   PIC 9(4)   USAGE COMP
                                       VALUE ZERO.
        02 MQMSG-DELIVERY   PIC X      VALUE SPACE.
        02 MQMSG-JOURNAL    PIC X      VALUE SPACE.
        02 MQMSG-DEADLETTER PIC X      VALUE SPACE.
        02 MSGDATA-CONV     PIC X      VALUE SPACE.
        02 CBLMQ-RESERVE    PIC X(102) VALUE LOW-VALUE.
      01 QUEUE-PARM.
```

```
  02 PATH-LEN  PIC S9(9)  USAGE COMP VALUE 13.
  02 PATH-NAME PIC X(256) VALUE '.¥SampleQueue'.
01  MESSAGE-PARM.
  02 LABEL-LEN PIC S9(9)  USAGE COMP VALUE 14.
  02 LABEL-DAT PIC X(512) VALUE 'Sample Message'.
  02 BODY-LEN  PIC S9(9)  USAGE COMP VALUE 19.
  02 BODY-DAT  PIC X(512) VALUE 'Sample Message data'.
  :
PROCEDURE DIVISION.
    :
    INITIALIZE CBLMQINF.
    MOVE LOW-VALUE TO CBLMQ-RESERVE OF CBLMQINF.

    MOVE '2' TO QUEUE-ACCESS OF CBLMQINF.
    CALL 'CBLMQOPEN' USING CBLMQINF QUEUE-PARM.
    IF ZERO NOT = RETURN-CODE THEN
      DISPLAY 'QUEUE OPEN ERROR'
      STOP RUN
    END-IF.

    DISPLAY 'SEND MESSAGE LABEL:' LABEL-DAT(1:LABEL-LEN).
    DISPLAY 'SEND MESSAGE DATA:' BODY-DAT(1:BODY-LEN).
    MOVE 3   TO MQMSG-PRIORITY   OF CBLMQINF.
    MOVE '0' TO MQMSG-DELIVERY   OF CBLMQINF.
    MOVE '0' TO MQMSG-JOURNAL    OF CBLMQINF.
    MOVE '0' TO MQMSG-DEADLETTER OF CBLMQINF.
    MOVE '0' TO MSGDATA-CONV     OF CBLMQINF.
    CALL 'CBLMQSENDMSG' USING CBLMQINF MESSAGE-PARM.
    IF ZERO NOT = RETURN-CODE THEN
      DISPLAY 'QUEUE SEND MESSAGE ERROR'
    END-IF.

    CALL 'CBLMQCLOSE' USING CBLMQINF QUEUE-PARM.
    IF ZERO NOT = RETURN-CODE THEN
      DISPLAY 'QUEUE CLOSE ERROR'
      STOP RUN
    END-IF.
    :
```

27.5.4　メッセージを受信するコーディングの例

'.¥SampleQueue'というキューからメッセージを受信するコーディングの例を次に示します。

```
  :
DATA DIVISION.
WORKING-STORAGE SECTION.
01 CBLMQINF.
  02 MSMQ-ERRCD       PIC X(8)   VALUE SPACE.
  02 DETAIL-CD        PIC S9(9)  USAGE COMP
                                  VALUE ZERO.
  02 LOCATE-END       PIC X      VALUE SPACE.
  02 QUEUE-ACCESS     PIC X      VALUE SPACE.
  02 MQMSG-CLASS      PIC X(2)   VALUE SPACE.
  02 MQMSG-TIMEOUT    PIC S9(9)  USAGE COMP
```

```
                                    VALUE ZERO.
  02 MQMSG-PRIORITY   PIC 9(4)   USAGE COMP
                                    VALUE ZERO.
  02 MQMSG-DELIVERY   PIC X      VALUE SPACE.
  02 MQMSG-JOURNAL    PIC X      VALUE SPACE.
  02 MQMSG-DEADLETTER PIC X      VALUE SPACE.
  02 MSGDATA-CONV     PIC X      VALUE SPACE.
  02 CBLMQ-RESERVE    PIC X(102) VALUE LOW-VALUE.
01 QUEUE-PARM.
  02 PATH-LEN  PIC S9(9)  USAGE COMP VALUE 13.
  02 PATH-NAME PIC X(256) VALUE '.¥SampleQueue'.
01 MESSAGE-PARM.
  02 LABEL-LEN PIC S9(9)  USAGE COMP.
  02 LABEL-DAT PIC X(512).
  02 BODY-LEN  PIC S9(9)  USAGE COMP.
  02 BODY-DAT  PIC X(512).
  :
PROCEDURE DIVISION.
    :
    INITIALIZE CBLMQINF.
    MOVE LOW-VALUE TO CBLMQ-RESERVE OF CBLMQINF.

    MOVE '1' TO QUEUE-ACCESS OF CBLMQINF.
    CALL 'CBLMQOPEN' USING CBLMQINF QUEUE-PARM.
    IF ZERO NOT = RETURN-CODE THEN
      DISPLAY 'QUEUE OPEN ERROR'
      STOP RUN
    END-IF.

    MOVE 20000 TO MQMSG-TIMEOUT OF CBLMQINF.
    MOVE '0'   TO MSGDATA-CONV  OF CBLMQINF.
    COMPUTE BODY-LEN = FUNCTION LENGTH(BODY-DAT).
    CALL 'CBLMQRECEIVEMSG' USING CBLMQINF MESSAGE-PARM.
    EVALUATE RETURN-CODE
    WHEN ZERO
    WHEN 2
      IF 2 = RETURN-CODE THEN
        DISPLAY 'RECEIVE MESSAGE DATA ERROR'
      ELSE IF LOW-VALUE = MQMSG-CLASS OF CBLMQINF THEN
        DISPLAY 'NORMAL MESSAGE'
      ELSE
        DISPLAY 'NOT NORMAL MESSAGE'
      END-IF END-IF
      DISPLAY 'PRIORITY   :'
               MQMSG-PRIORITY   OF CBLMQINF
      DISPLAY 'DELIVERY   :'
               MQMSG-DELIVERY   OF CBLMQINF
      DISPLAY 'JOURNAL    :'
               MQMSG-JOURNAL    OF CBLMQINF
      DISPLAY 'DEAD-LETTER:'
               MQMSG-DEADLETTER OF CBLMQINF
      DISPLAY 'LABEL:' LABEL-DAT(1:LABEL-LEN)
      DISPLAY 'DATA :' BODY-DAT (1:BODY-LEN)
    WHEN 1
      DISPLAY 'RECEIVE MESSAGE BUFFER OVERFLOW:' BODY-LEN
    WHEN 100
      DISPLAY 'RECEIVE MESSAGE TIME-OUT'
    WHEN OTHER
```

```
      DISPLAY 'QUEUE RECEIVE MESSAGE ERROR'
    END-EVALUATE.

    CALL 'CBLMQCLOSE' USING CBLMQINF QUEUE-PARM.
    IF ZERO NOT = RETURN-CODE THEN
      DISPLAY 'QUEUE CLOSE ERROR'
      STOP RUN
    END-IF.
    :
```

27.5.5 キューのパス名を検索するコーディングの例

'Sample Queue'というラベルの付いた'user1¥samplequeue'というパス名のキューがあるかどうかを検索し，なければ'Sample Queue'というラベルの付いたパブリックキューのパス名を一覧表示するコーディング例を次に示します。

```
  :
DATA DIVISION.
WORKING-STORAGE SECTION.
01  CBLMQINF.
  02 MSMQ-ERRCD       PIC X(8)   VALUE SPACE.
  02 DETAIL-CD        PIC S9(9)  USAGE COMP
                                  VALUE ZERO.
  02 LOCATE-END       PIC X      VALUE SPACE.
  02 QUEUE-ACCESS     PIC X      VALUE SPACE.
  02 MQMSG-CLASS      PIC X(2)   VALUE SPACE.
  02 MQMSG-TIMEOUT    PIC S9(9)  USAGE COMP
                                  VALUE ZERO.
  02 MQMSG-PRIORITY   PIC 9(4)   USAGE COMP
                                  VALUE ZERO.
  02 MQMSG-DELIVERY   PIC X      VALUE SPACE.
  02 MQMSG-JOURNAL    PIC X      VALUE SPACE.
  02 MQMSG-DEADLETTER PIC X      VALUE SPACE.
  02 MSGDATA-CONV     PIC X      VALUE SPACE.
  02 CBLMQ-RESERVE    PIC X(102) VALUE LOW-VALUE.
01 LOCATE-PARM.
  02 PATH-LEN  PIC S9(9)  USAGE COMP.
  02 PATH-NAME PIC X(256).
  02 LABEL-LEN PIC S9(9)  USAGE COMP VALUE 12.
  02 LABEL-DAT PIC X(256) VALUE 'Sample Queue'.
  :
PROCEDURE DIVISION.
    :
    INITIALIZE CBLMQINF.
    MOVE LOW-VALUE TO CBLMQ-RESERVE OF CBLMQINF.

    MOVE '0' TO LOCATE-END OF CBLMQINF.
    PERFORM WITH TEST AFTER
      UNTIL ( 1 NOT = RETURN-CODE ) AND
            ( 2 NOT = RETURN-CODE )
      CALL 'CBLMQLOCATE' USING CBLMQINF LOCATE-PARM
      IF ( 1 = RETURN-CODE ) AND
         ( 'user1¥samplequeue' = PATH-NAME ) THEN
        MOVE '1' TO LOCATE-END OF CBLMQINF
```

```
      END-IF
    END-PERFORM.
    EVALUATE RETURN-CODE
    WHEN 0
      DISPLAY 'FOUND !:' PATH-NAME(1:PATH-LEN)
      STOP RUN
    WHEN 100
      DISPLAY 'NOT FOUND PATH-NAME'
    WHEN OTHER
      DISPLAY 'QUEUE LOCATE ERROR'
      STOP RUN
    END-EVALUATE.

    MOVE '0'TO LOCATE-END OF CBLMQINF.
    PERFORM WITH TEST AFTER
      UNTIL ( 1 NOT = RETURN-CODE ) AND
            ( 2 NOT = RETURN-CODE )
      CALL 'CBLMQLOCATE' USING CBLMQINF LOCATE-PARM
      EVALUATE RETURN-CODE
      WHEN 1
      WHEN 2
        IF 2 = RETURN-CODE THEN
          DISPLAY 'QUEUE PATH-NAME DATA ERROR'
        END-IF
        DISPLAY 'QUEUE PATH-NAME[' PATH-LEN ']:'
                PATH-NAME(1:FUNCTION MIN(PATH-LEN,256))
      END-EVALUATE
    END-PERFORM.
    IF ZERO > RETURN-CODE THEN
      DISPLAY 'QUEUE LOCATE ERROR'
    END-IF.
    :
```

28 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。

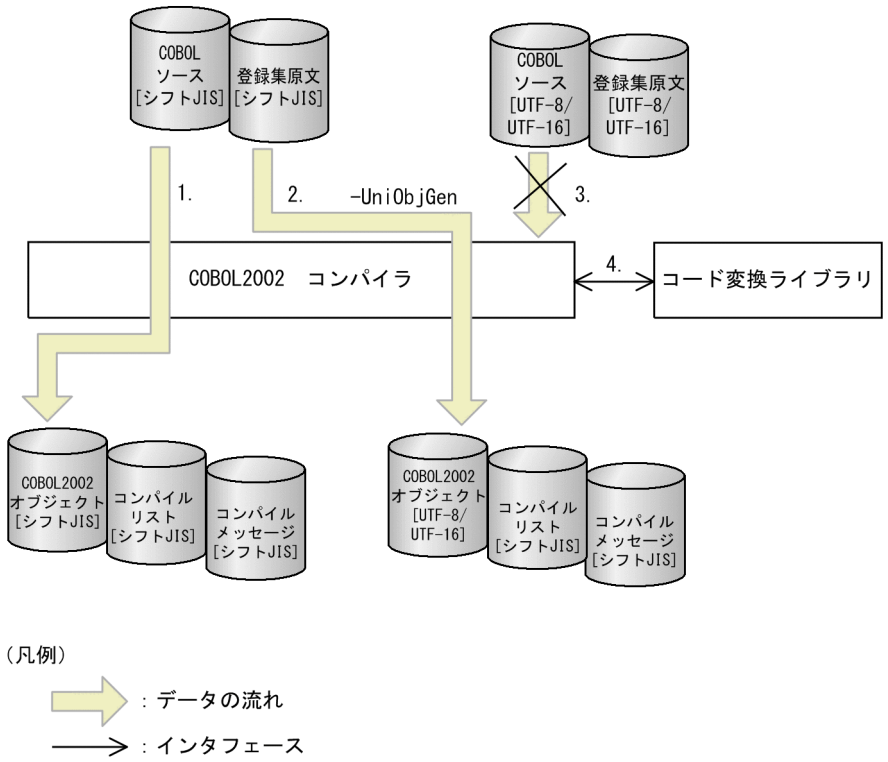
28.1　Unicode 機能の概要

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。

28.1.1　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。

図 28–1　コンパイルでの Unicode 機能

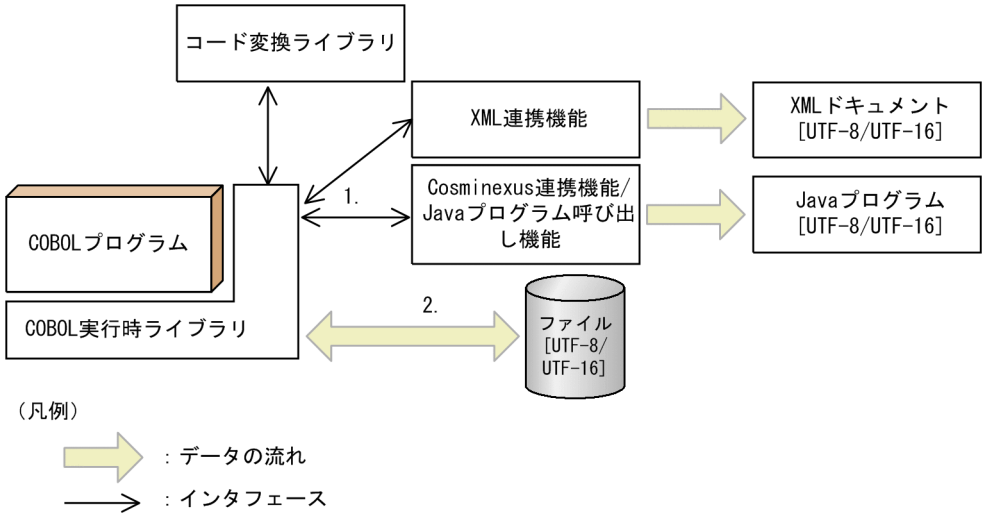


1. シフト JIS で記述された COBOL ソースプログラム，登録集原文を入力し，-UniObjGen オプションを指定しないでコンパイルすると，シフト JIS 環境で動作するオブジェクトが生成されます。
2. シフト JIS で記述された COBOL ソースプログラム，登録集原文を入力し，-UniObjGen オプションを指定してコンパイルすると，COBOL の Unicode 機能を使用する環境で動作するオブジェクトが生成されます。コンパイル時に出力されるコンパイルメッセージ，コンパイルリストはシフト JIS で出力されます。
3. Unicode で記述された COBOL ソースプログラムは，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。

28.1.2　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 28–2　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。
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。

28.1.3　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　操作ガイド」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

28.2 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。

28.3 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。

28.3.1 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。

- Windows(x86) COBOL2002 の場合
  - 日立コード変換 - Server Runtime
  - 日立コード変換 - Client Runtime
  - Hitachi Code Converter - Server Runtime for C/COBOL
  - Hitachi Code Converter - Client Runtime for C/COBOL
- Windows(x64) COBOL2002 の場合
  - 日立コード変換 - Server Runtime(64)
  - 日立コード変換 - Client Runtime(64)
  - Hitachi Code Converter - Client Runtime for C/COBOL(64)
  - Hitachi Code Converter - Server 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。

- Windows(x86) COBOL2002 の場合
  - 日立コード変換 - Development Kit
  - 日立コード変換 - Development Kit(64)
  - Hitachi Code Converter - Development Kit for C/COBOL
  - Hitachi Code Converter - Development Kit for C/COBOL(64)
- Windows(x64) COBOL2002 の場合
  - 日立コード変換 - Development Kit(64)
  - Hitachi Code Converter - Development Kit for C/COBOL(64)

28.4.1　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，「33.5.14　その他の設定」の「(16)　-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，「33.5.14　その他の設定」の「(17)　-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 28–1　-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）
指定あり	日本語文字定数	UTF-16LE 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		

(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。

- -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，「33.6.3　コンパイラ環境変数の詳細」の「(22)　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。

28.4.2　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，「36.2.3　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 28–2　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）
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-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）無変換		

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）
指定なし			

(2) -UniObjGen オプションと実行時環境変数 CBLLANG の関係

-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 28–3 -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 28–4 -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）
-UniEndian,Little	○	×	○
-UniEndian,Big	×	○	×
指定なし	○	×	○

（凡例）

○：動作を保証する

×：動作を保証しない

(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 28–5　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。
MSMQ アクセス機能	戻り値として-2 を返し，詳細コードにデータ変換失敗を示すコードを設定する。詳細については，「27.3　MSMQ アクセスサービスルーチンのインタフェース」の「表 27–3　詳細コードの内容」を参照のこと。
OLE2 オートメーションクライアント機能	実行時エラーとなり，詳細情報にコード変換失敗を示すコードを設定する。
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，「30.7.1 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)　開発マネージャからデバッガを起動する場合

開発マネージャからデバッガを起動する場合，デバッガに対しては実行支援による環境変数の指定が有効となりません。そのため，実行時環境変数 CBLLANG，CBLUNIENDIAN の指定については，開発マネージャの「プロジェクトの設定」ダイアログボックスのユーザ設定タブからも指定が必要となります。

28.5 Unicode に対応する機能

Unicode に対応する機能を，次に示します。

表 28–6 Unicode に対応する機能（規格）

機能名	Unicode 対応状況			備考
	用途 DISPLAY		用途 NATIONAL	
	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，「28.5.1 基本機能」，または「28.7 Unicode 機能での制限事項」を参照のこと。
順編成ファイル	○	○※2	○※2	
相対編成ファイル	○	○※2	○※2	
ISAM 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。
整列併合	○	○※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	
組み込み関数	○	○	○	
オブジェクト指向	○	○※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 • 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	○	○	○	

機能名	Unicode 対応状況			備考
	用途 DISPLAY		用途 NATIONAL	
	ASCII 範囲内※1	ASCII 範囲外※1		
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シフト JIS である。
区分化	○	○	○	

（凡例）

○：制限なく使用できる

△：制限付きで使用できる

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，「28.7.1　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 28–7　Unicode に対応する機能（X/Open）

機能名		Unicode 対応状況			備考
		用途 DISPLAY		用途 NATIONAL	
		ASCII 範囲内※1	ASCII 範囲外※1		
テキスト編成ファイル		○	△※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	○	○	○	シフト JIS として取得する。
	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。

機能名	Unicode 対応状況			備考
	用途 DISPLAY		用途 NATIONAL	
	ASCII 範囲内※1	ASCII 範囲外※1		
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，「28.5.2　入出力機能」の「(6)　画面入出力機能」，または「28.7　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	

（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，「28.7.1　コンパイル時の制限事項」を参照してください。

表 28–8　Unicode に対応する機能（拡張機能）

機能名	Unicode 対応状況			備考
	用途 DISPLAY		用途 NATIONAL	
	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

機能名		Unicode 対応状況			備考
		用途 DISPLAY		用途 NATIONAL	
		ASCII 範囲内※1	ASCII 範囲外※1		
					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。
Btrieve による索引編成ファイル※3		○	○※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	
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，「28.5.2　入出力機能」の「(6)　画面入出力機能」，または「28.7　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		○	×	×	
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	○	×	×	
	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		○	○	○	
イベントログファイル出力機能		○	×	×	実行時エラーメッセージはシフト JIS で出力されるが，埋め字として含まれる Unicode 文字は変換されない。

機能名		Unicode 対応状況			備考
		用途 DISPLAY		用途 NATIONAL	
		ASCII 範囲内※1	ASCII 範囲外※1		
データコミュニケーション機能		○	○※4	○※4	
データベース操作機能（ODBC インタフェース）※6		△	×	×	
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 オートメーションインタフェース機能	OLE2 オートメーションクライアント機能	○	○	×	
マルチスレッド環境での実行		○	○	○	
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。
	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。
	MSMQ アクセス機能	○	○	×	
数字項目のけた拡張機能※5		○	○	○	
動的長基本項目機能		○	○	○	
定数長拡張機能		○	○	○	
Java プログラム呼び出し機能※7		○	△	○	

（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

Windows(x64) COBOL2002 は未対応です。

注※4

UCS-2 の範囲を使用できます。

注※5

Windows(x86) COBOL2002 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

一部制限の詳細は，マニュアル「COBOL2002 Java プログラム呼び出し機能ガイド」を参照してください。

表 28–9　Unicode に対応する機能（連携機能）

機能名	Unicode 対応状況			備考
	用途 DISPLAY		用途 NATIONAL	
	ASCII 範囲内※1	ASCII 範囲外※1		
XML 連携機能	○	○※2	○※2	
Cosminexus 連携機能	○	○	○	

（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

UCS-2 の範囲を使用できます。

表 28–10　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
		用途 DISPLAY		用途 NATIONAL	
		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	GUI モード	○	○	○	
	バッチモード	○	○	○	
カバレージ機能	GUI モード	○	○	○	
	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。

28.5.1　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 28–11　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

説明		バイト数	けた数※1	文字数	見た目幅
	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 28–12　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

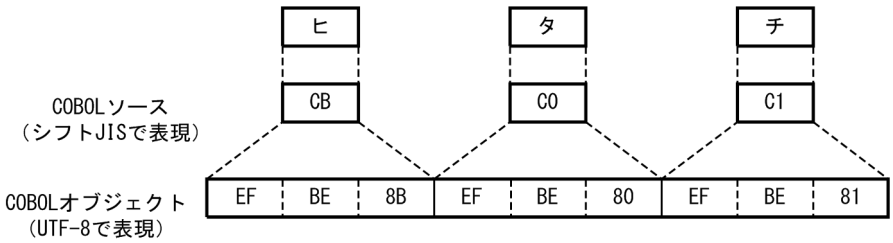
(2) 文字の長さ

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。

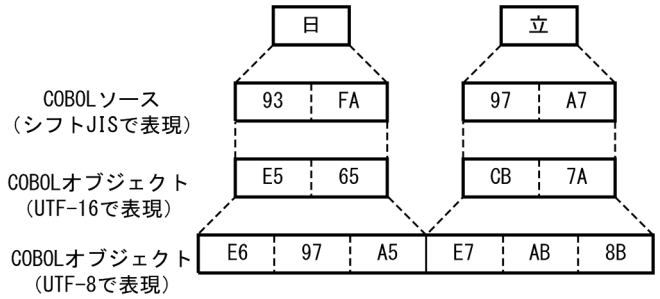
```
01 TEST-DATA1 PIC X(9) USAGE DISPLAY VALUE 'ﾋﾀﾁ'.
```



（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。

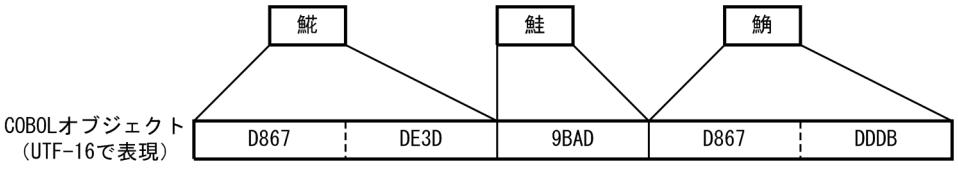
```
01 TEST-DATA2 PIC N(2) USAGE NATIONAL VALUE NC'日立'.
01 TEST-DATA3 PIC X(6) USAGE DISPLAY  VALUE   '日立'.
```



（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。

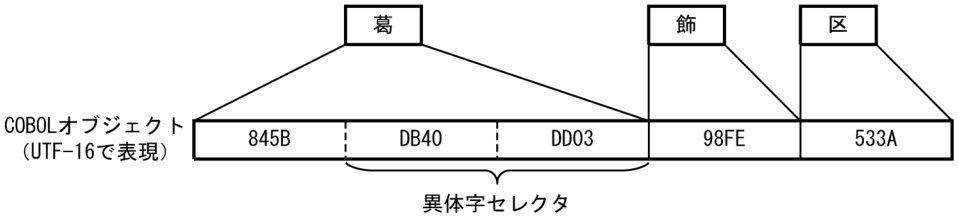
```
01 TEST-DATA2 PIC N(5) USAGE NATIONAL
   VALUE NX'D867DE3D9BADD867DDDB'. *> '鯱鮭鯣'
```



（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。

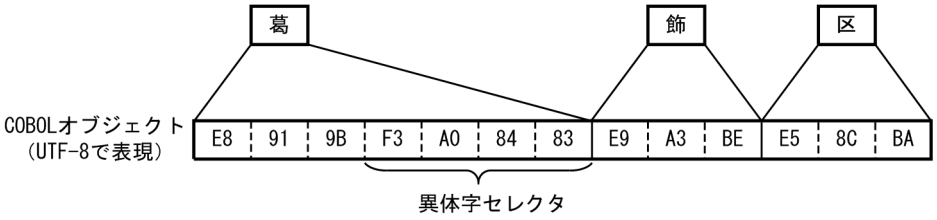
```
01 TEST-DATA3 PIC N(5) USAGE NATIONAL
   VALUE NX'845BDB40DD0398FE533A'.  *> '葛飾区'
```



（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

Windows(x86) COBOL2002 の場合

```
ccbl2002 -UniObjGen sample1.cbl codeconv.lib
```

Windows(x64) COBOL2002 の場合

```
ccbl2002 -UniObjGen sample1.cbl codeconv64.lib
```

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
- データベース操作機能
- 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。

28.5.2　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 28–13　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。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。

(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。

(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'0D0A'となります。

(6) 画面入出力機能

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

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。

28.5.3　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，「30.7.1　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。

28.5.4　MSMQ アクセス機能

MSMQ アクセス機能では，Unicode データのやり取りができます。MSMQ アクセス機能については，「27.　MSMQ アクセス機能」を参照してください。

Unicode 機能での MSMQ アクセスの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，詳細コードとして，60 を設定します。
- 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－2 を返し，詳細コードとして，61 を設定します。

28.5.5　OLE2 オートメーション機能

OLE2 オートメーション機能については「25.　OLE2 オートメーション機能」を参照してください。

(1)　OLE2 オートメーションクライアント機能

OLE2 オートメーションクライアント機能では，Unicode データのやり取りができます。

(a)　OLE2 オートメーション機能での制限

OLE2 オートメーションクライアント機能で，INVOKE 文の引数や戻り値，SET 文，または OLE2 インタフェース機能の組み込み関数の引数には，次の項目は指定できません。指定した場合はコンパイルエラーとなります。

- 用途が NATIONAL の項目
- 用途が NATIONAL の項目を従属項目として含む集団項目
- 日本語文字定数
- 16 進日本語文字定数

(b)　実行時の注意事項

実行時の注意事項を次に示します。

- 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，実行時エラーとなります。
- 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，実行時エラーとなります。

28.5.6　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 28–14　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。
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。

項番	用途	組み込み関数名	概要
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。

28.6　シフト JIS で入出力する情報

シフト JIS コードで入出力する情報を次に示します。

表 28–15　シフト JIS コードで入出力する情報

分類	入力／出力	入出力情報
翻訳時	入力	COBOL ソースプログラム（登録集原文を含む）
		コマンドラインの引数
		stdcall 呼び出し指示ファイル※
		モジュール定義ファイル
	出力	コンパイラメッセージ
		コンパイルリストファイル
		TD コマンド格納ファイル
実行時	入力	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
		ACCEPT 文で入力したデータ
		プレロードリストファイル
	出力	実行時エラーメッセージ
		異常終了時要約情報リスト
		データ領域ダンプリスト
		入出力サービスルーチンデバッグリスト
		プログラム検索トレースファイル
リポジトリ管理ツール	出力	エラーメッセージ
		リポジトリファイル情報リスト
テストデバッガ	入力	TD コマンド格納ファイル
	出力	モニタ表示出力ファイル（ログ出力ファイル）
		結果蓄積ファイル
		結果出力ファイル
		エラーメッセージ
カバレージ	出力	カウント情報リストファイル
		カバレージ情報リストファイル
		実行結果出力ファイル
		エラーメッセージ

注※

Windows(x86) COBOL2002 で有効です。

28.7　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。

28.7.1　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 28–16　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
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 • 外部プログラム節のプログラム名 1／定数 1（Windows(x86) COBOL2002 で有効） • 特殊名段落の定数 10※2
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 • ENTRY 文の定数 • DISPLAY 文の定数 1／2／3※1 • INVOKE 文の定数 1 • STOP 文の定数 2
組み込み関数	• ADDR 関数の引数 1
画面機能（SCREEN／WINDOW SECTION）	• 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

特殊名段落の定数 10 は報告書作成機能で，-CompatiV3 オプションを指定したときに有効になります。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。

28.7.2 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。
- 日本語，半角かたかなど，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
  - CBLMESSAGE サービスルーチン
  - CBLPUT サービスルーチン
  - CBLSETTITLE サービスルーチン
  - CBLINPUTDLG 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。
- イベントログファイル出力機能で，DISPLAY 文で出力する情報に多バイト文字を含む場合，出力結果は保証しません。
- 日本語，半角かたかなど，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。
- 外部で設定された環境変数の設定値およびコマンドラインの引数に 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，動作は保証しません。

29 数字項目のけた拡張機能（Windows(x64) COBOL2002 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。

29.1　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。

29.1.1　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 29-1　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。

29.1.2　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，「33.5.14　その他の設定」の「(18)　-MaxDigits38 オプション（Windows(x64) COBOL2002 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，「33.5.14　その他の設定」の「(19)　-IntResult,DecFloat40 オプション（Windows(x64) COBOL2002 で有効)」を参照してください。

29.2　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。

29.2.1　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 29–1　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。

29.2.2　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 29-2　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

29.2.3　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。

29.3　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。

29.3.1　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，「29.3.2　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

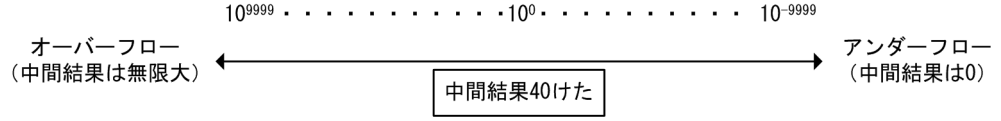
- 次のどれかに該当する演算の場合，内部浮動小数点演算となるため，中間結果に 40 けた 10 進浮動小数点形式は適用されません。
  - ・被べき数（底）またはべき数（指数）が浮動小数点項目または浮動小数点定数のべき乗演算
  - ・べき数（指数）が算術式のべき乗演算
  - ・べき数（指数）が整数でないべき乗演算
  - ・関数値のデータ属性が内部浮動小数点形式の組み込み関数
  - ・浮動小数点項目および浮動小数点定数に対するマイナス（-）単項演算子
  - ・浮動小数点項目および浮動小数点定数同士の演算

29.3.2　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 29-2 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 29-2　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 29-3 および図 29-4 に，数字項目の加算を行う場合の例を示します。

図 29-3　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。

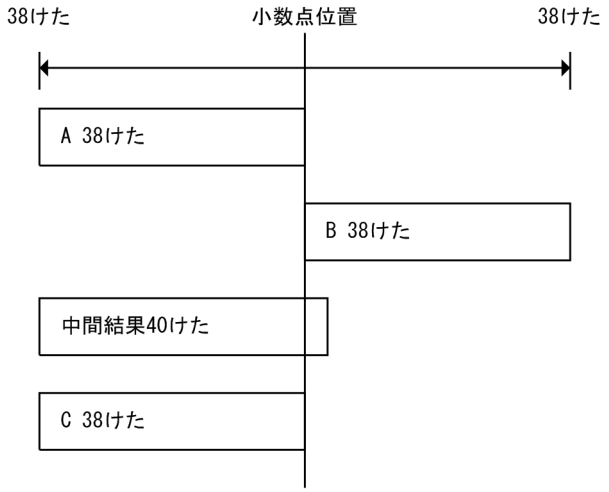
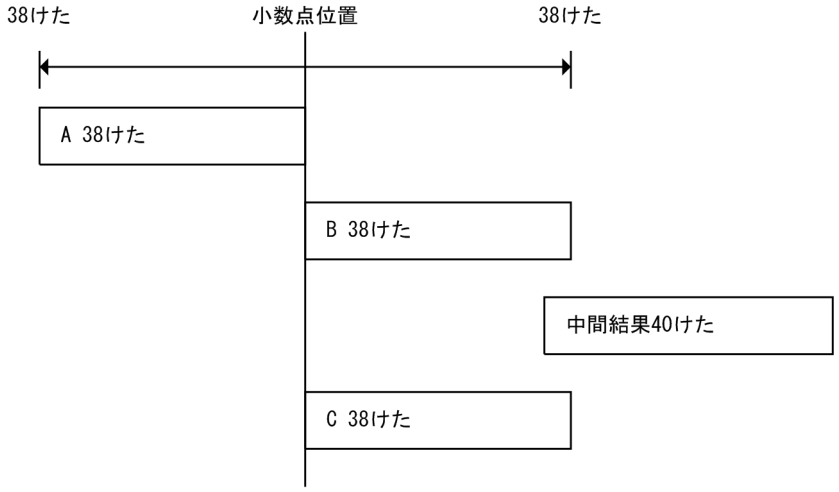


図 29-4　数字項目の加算を行う場合の中間結果（例 2）

```
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
「1.0000000000000000000000000000000000000000E-37」が保持されます。



29.4　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。

29.4.1　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 29–3　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		○
	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		×
	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		○
	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出力機能		○
	データコミュニケーション		×
	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		○

種類	機能名	けた拡張機能への対応
	定数長拡張機能	○
	Java プログラム呼び出し機能	○
デバッグ機能	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	○
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

29.4.2　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 29–4　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

29.4.3　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 29-5　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

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 29-6　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	×

ファイル編成	数字項目のけた拡張機能への対応
テキスト編成ファイル	○
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

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 29-7　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 29-8　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	○
INSPECT 文	×
INVOKE 文	×
MERGE 文	×
MOVE 文	○
MULTIPLY 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
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

29.5　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。

29.5.1　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 29-9　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.
```

29.5.2　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
```

30 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。

30.1　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。

30.1.1　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。

項番	サービスルーチン	機能
1	CBLGINT※	他言語のプログラムから呼び出した COBOL プログラムを，GUI モードで動作させる。
2	CBLEND※	COBOL の実行環境を終了させる。
3	CBLABN	プログラムを異常終了させる。
4	CBLARGC	コマンド行に指定した引数の個数を取得する。
5	CBLARGV	コマンド行に指定した引数の内容を取得する。
6	CBLEXEC	別のアプリケーションプログラムを起動する。
7	CBLHANDLE	コンソールウィンドウのハンドル，インスタンスハンドルを取得する。

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，「30.4　プログラム実行制御」を参照してください。

30.1.2　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。

項番	サービスルーチン	機能
1	CBLMESSAGE	指定した文字列を画面に表示する。
2	CBLINPUTDLG	指定した文字列を画面に表示し，入力された文字列を受け取る。
3	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。

サービスルーチンの詳細については，「30.5　ダイアログボックス／ウィンドウ」を参照してください。

30.1.3　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，異常終了時要約情報リスト，および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。

項番	サービスルーチン	機能
1	CBLDBGINF※	例外が発生したときに，異常終了時要約情報リストを出力する。
2	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。

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，「30.6　デバッグ機能」を参照してください。

30.1.4　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。

項番	サービスルーチン	機能
1	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。
2	CBLUBIT	ビットデータを論理演算する。
3	CBLCNVERRORINFO	組み込み関数の DISPLAY-OF 関数または NATIONAL-OF 関数のエラー情報を取得する。

サービスルーチンの詳細については，「30.7　変換・転記・演算」を参照してください。

30.1.5　画面節（SCREEN SECTION および WINDOW SECTION）

画面節での画面入出力機能で使用するサービスルーチンです。

画面節のサービスルーチンを，次に示します。

項番	サービスルーチン	機能
1	CBLSGET	キーボードからの入力を 1 文字受け取る。
2	CBLSETTITLE	画面機能のウィンドウタイトルバーに，指定された文字列を表示する。

サービスルーチンの詳細については，「30.8　画面節（SCREEN SECTION および WINDOW SECTION）」を参照してください。

30.1.6　データベース操作機能

データベース操作機能で使用するサービスルーチンです。

データベース操作機能のサービスルーチンを，次に示します。

項番	サービスルーチン	機能
1	CBLSQLERROR	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。
2	CBLSQLSETOPT	SQL 文による ODBC インタフェース使用時に，接続オプションを設定する。

サービスルーチンの詳細については，「30.9　データベース操作機能」を参照してください。

30.1.7　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。

30.1.8　XMAP3 を使用した画面・帳票関連

XMAP3 を使用した書式印刷機能（Windows(x86)COBOL2002 で有効），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，「30.10　XMAP3 を使用した画面・帳票関連」を参照してください。

30.1.9　MSMQ アクセス機能

COBOL プログラムから MSMQ にアクセスするための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，「27.　MSMQ アクセス機能」を参照してください。

30.1.10　その他の機能

その他の機能のためのサービスルーチンです。

その他の機能のサービスルーチンを，次に示します。

項番	サービスルーチン	機能
1	CBLPUT	画面に 1 文字を出力する。
2	CBLGET	キーボードから 1 文字を入力する。
3	CBLADTRM	終了キーを追加する。
4	CBLDLTRM	終了キーを削除する。
5	CBLCNSL	コンソールの状態を検査する。
6	CBLBELL	警告音を発信する。
7	CBLCUR	指定した位置へカーソルを移動する。

サービスルーチンの詳細については，「30.11　その他の機能」を参照してください。

30.2　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，「30.4.5　CBLARGV」の使用例を参照してください。

30.3　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。

- -BigEndian,Bin オプションを指定したプログラムから引数に 2 進項目を持つサービスルーチンの呼出しを行なう場合，引数となる 2 進項目は COMP-5 でなければなりません。
- 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 30–1　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

30.4　プログラム実行制御

30.4.1　CBLGINT

CBLGINT サービスルーチンは，COBOL プログラムを GUI モードで動作させるための環境を設定するものです。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

```
#include <windows.h>

extern int WINAPI CBLGINT();
                  /*CBLGINTの外部参照宣言*/

CBLGINT();        /*CBLGINTの呼び出し    */
```

引数

なし。

戻り値

常に 0 が返されます。

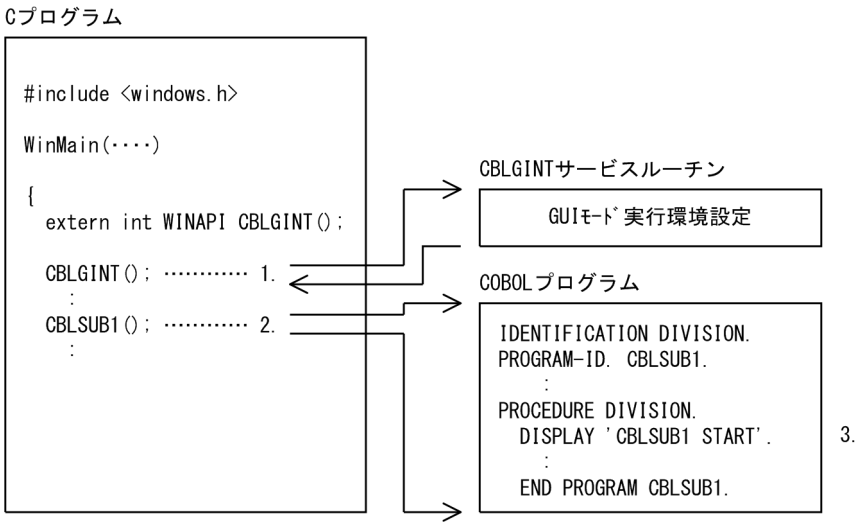
規則

- CBLGINT サービスルーチンは，COBOL プログラムの呼び出しに先立って 1 回だけ呼び出さなければなりません。
- COBOL プログラムを呼び出したあとに CBLGINT サービスルーチンを呼び出した場合，CBLGINT サービスルーチンは何も処理しないでリターンします。また，COBOL の実行環境は，最初に呼び出された COBOL プログラムのモードに従います。このため，CBLGINT サービスルーチンでは，設定済みの COBOL プログラムのモードを変更できません。
- CBLGINT サービスルーチンで GUI モードの COBOL プログラムの環境設定をした場合，GUI モードで動作する COBOL プログラムは，明示的または暗黙的な STOP RUN 文の実行，または CBLEND サービスルーチンの呼び出しまで有効です。
- CBLGINT サービスルーチンは，COBOL 以外の言語だけから呼び出せます。
- 主プログラムが COBOL 以外の言語で，CBLGINT サービスルーチンを呼び出さないで COBOL プログラムを呼び出した場合，COBOL プログラムは CUI モードで動作します。
- このサービスルーチンを使用するプログラムでは，リンク時に以下のインポートライブラリを指定してください。

  cbl2k_32.lib

使用例

CBLGINT サービスルーチンを使用して，C プログラムから呼び出す COBOL プログラムの GUI モードを設定する例を，次に示します。



1. C 言語で記述された WinMain 関数から，COBOL プログラムを GUI モードで実行させるための CBLGINT サービスルーチンを呼び出します。
2. COBOL プログラムを呼び出します。
3. 呼び出された COBOL プログラムで DISPLAY 文を実行すると，COBOL コンソールウィンドウに「CBLSUB1 START.」が表示されます。

30.4.2 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

```
#include <windows.h>

extern int WINAPI 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

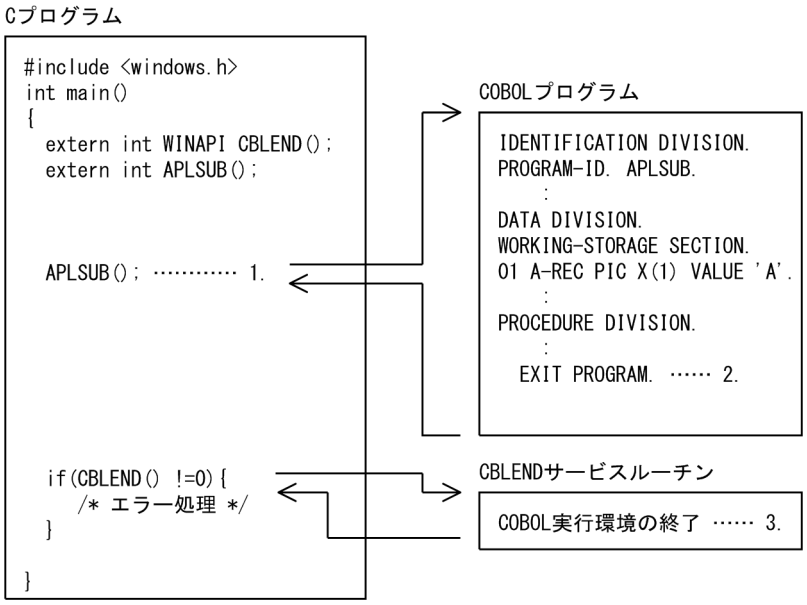
-1：COBOL プログラムが動作中であるため，CBLEND サービスルーチンが無視された場合

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，リンク時に次のインポートライブラリを指定してください。
  cbl2k_32.lib

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。

30.4.3 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。

環境変数

引数 1 に指定した値を異常終了時の終了コードにする場合，環境変数 CBLABNCODE に YES を指定してください。環境変数 CBLABNCODE に YES 以外を指定した場合，異常終了時の終了コードは 1 になります。

ただし，COBOL がプログラム（プロセスまたはスレッド）を終了しない環境（COBOL が副プログラムとして動作する場合など）では，環境変数 CBLABNCODE の指定が無効となります。

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。
- このサービスルーチンによってプログラムを終了させた場合，終了コードは 1 になります。

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。

30.4.4　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

「30.4.5　CBLARGV」の使用例を参照してください。

30.4.5　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。

30.4.6 CBLEXEC

CBLEXEC サービスルーチンは，COBOL プログラムから別のアプリケーションプログラムを起動するものです。

形式

```
CALL 'CBLEXEC' USING 引数1 引数2 引数3
```

引数

- 引数 1 には，プロセス起動するコマンド文字列の長さを，2 バイトの 2 進項目で指定します。指定できる長さは 1～255 バイトです。
- 引数 2 には，プロセス起動するコマンド文字列を指定します。この領域中にシステムが認識できないコードがある場合の結果は保証しません。
- 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 30-2 CBLEXEC 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 FILLER PIC X(01).	システムの使用する領域。値は X'00'でなければならない。
02 データ名2 PIC X(01).	起動したプロセスの終了を待つかどうかを指定する。 0：プロセスの終了を待つ。 1：プロセスの終了を待たない。
02 データ名3 PIC X(01).	起動したプロセスのウィンドウ表示方法を指定する。

記述形式	内容
	0：プロセスのデフォルトウィンドウ表示とする。 1：ウィンドウをアイコン化して表示する。
`02 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。

戻り値

0：正常に起動し，プロセスの終了を待たない場合

（正常に起動し，プロセスの終了を待つ場合は，呼び出したコマンドの戻り値がそのまま返ります）

-1：起動できなかった場合

規則

- 引数 2 の実際の長さより引数 1 中の長さが長い場合は，結果は保証しません。
- 引数 2 の実際の長さより引数 1 中の長さが短い場合は，引数 1 の長さまでがコマンド文字列とみなされます。
- CBLEXEC から COBOL プログラムを呼び出したとき，プログラムの実行終了（STOP RUN 文，実行時エラー）と同時にプロセスを終了させたい場合には，次の環境変数を設定します。

```
CBL_BATCH=1
```

- CBLEXEC を使って呼び出した COBOL 主プログラムで設定した RETURN-CODE 特殊レジスタの内容は制御が戻ってきたときに参照できます。参照できる範囲は 0～255 で，RETURN-CODE 特殊レジスタの下位 8 ビットの内容です。
- 引数 2 に空白が含まれた実行可能ファイル名を指定した場合，CALL 文のときと同様の注意が必要となります。詳細は，「18.7　実行可能ファイルの呼び出し」の注意事項を参照してください。

使用例

メモ帳を呼び出す例を次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
           :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 EXEC-NAME-LEN PIC 9(4) USAGE COMP VALUE 29.
       01 EXEC-NAME     PIC X(29)
                  VALUE 'NOTEPAD.EXE C:¥TMP¥SAMPLE.TXT'.
       01 EXEC-PARM.
         02 FILLER    PIC X(01) VALUE X'00'.
         02 PROC-IND  PIC X(01).
         02 SHOW-WIND PIC X(01) VALUE '0'.
         02 FILLER    PIC X(01) VALUE '0'.
           :
       PROCEDURE DIVISION.
           :
           MOVE '1' TO PROC-IND.
           CALL 'CBLEXEC'
                 USING EXEC-NAME-LEN EXEC-NAME EXEC-PARM.
           IF RETURN-CODE NOT = 0 THEN
```

```
          CBLEXECエラー処理
        END-IF.
        :
```

30.4.7 CBLHANDLE

CBLHANDLE サービスルーチンは，COBOL のウィンドウハンドルなどを取得するものです。

形式

```
CALL ’CBLHANDLE’ USING 引数1 引数2 引数3
```

引数

- 引数 1 には，コンソールウィンドウのハンドル情報を格納する領域を 4 バイトの 2 進項目※で指定します。
- 引数 2 には，インスタンスハンドルを格納する領域を 4 バイトの 2 進項目※で指定します。
- 引数 3 は予備項目であり，4 バイトの 2 進項目※で指定します。

注※

Windows(x64) COBOL2002 の場合は，8 バイトの 2 進項目です。

戻り値

0：正常終了した場合

-1：COBOL プログラムが主プログラムでない場合，または CUI モードで実行中の場合

規則

引数の指定，引数の属性が異なった場合の結果は保証しません。

使用例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
           :
       DATA DIVISION.
       WORKING-STORAGE  SECTION.
       01 HWND   PIC 9(9) USAGE COMP.※
       01 HINST  PIC 9(9) USAGE COMP.※
       01 HDUMMY PIC 9(9) USAGE COMP.※
           :
       PROCEDURE DIVISION.
           :
           CALL ’CBLHANDLE’ USING HWND HINST HDUMMY.
           IF RETURN-CODE NOT = 0 THEN
             CBLHANDLEエラー処理
           END-IF.
          :
```

注※

Windows(x64) COBOL2002 の場合は，「PIC 9(9)」は「PIC 9(18)」に読み替えてください。

30.5.1 CBLMESSAGE

CBLMESSAGE サービスルーチンは，引数で指定した文字列をメッセージボックスに表示するものです。

形式

```
CALL ’CBLMESSAGE’ USING 引数1 引数2
```

引数

- 引数 1 には，表示する文字列の長さを 4 バイトの 2 進項目で指定します。対応する長さは 1～1,024 バイトであり，この長さ以外を指定するとエラーになります。
- 引数 2 には，表示する文字列を指定します。システムが表示できないコードが文字列中にあった場合の結果は保証しません。

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 で指定した長さが引数 2 の実際の長さより長い場合，結果は保証しません。
- 引数 1 で指定した長さが引数 2 の実際の長さより短い場合，引数 1 の長さで出力されます。

使用例

「DISPLAY TESTING」という文字列をメッセージボックスに表示する場合の例を次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
                 :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 M-BOX-LEN PIC 9(9) USAGE COMP VALUE 15.
       01 M-BOX-DATA PIC X(15) VALUE ’DISPLAY TESTING’.
                 :
       PROCEDURE DIVISION.
           CALL ’CBLMESSAGE’ USING M-BOX-LEN
                                   M-BOX-DATA.
           IF RETURN-CODE NOT = 0 THEN
               CBLMESSAGEエラー処理
           END-IF.
                 :
```

30.5.2 CBLINPUTDLG

CBLINPUTDLG サービスルーチンは，引数で指定した文字列をダイアログボックスに表示したあと，そのダイアログボックスから入力された文字列を受け取るものです。なお，入力用の領域に入力された文字列は，左詰めで格納されます。

形式

```
CALL 'CBLINPUTDLG' 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 30-3 CBLINPUTDLG 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 PIC 9(09) 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。0～1,024 バイトまでの長さを指定できる。
02 データ名3 PIC 9(09) USAGE COMP.	入力する文字列の長さを 4 バイトの 2 進項目で指定する。0～1,024 バイトまでの長さを指定できる。

- 引数 2 には，出力する文字列を格納する領域を英数字項目で指定します。
- 引数 3 には，入力した文字列を格納する領域を英数字項目で指定します。

戻り値

1：正常終了した場合（キャンセルボタンが押された）

0：正常終了した場合（OK ボタンが押された）

-1：パラメタエラーが発生した場合

-2：その他のエラーが発生した場合

規則

出力文字列に関する注意

- 引数 2 に指定された文字列中に，システムが表示できないコードがあった場合の結果は保証しません。
- 引数 1 のデータ名 2 に指定した文字列長より，引数 2 に指定した出力領域長が長い場合，引数 1 のデータ名 2 に指定した文字列長で出力します。また，指定した文字列長が多バイト文字の途中で終わる場合，この文字は出力されません。
- 引数 1 のデータ名 2 に指定した文字列長より，引数 2 に指定した出力領域長が短い場合の結果は保証しません。
- 引数 2 に指定された文字列の途中に NULL（X'00'）がある場合，NULL 以降の文字列は出力されません。

入力文字列に関する注意

- 引数 1 のデータ名 3 に指定した文字列長より，引数 3 に指定した入力領域長が長い場合，引数 1 のデータ名 3 に指定した文字列長で入力できます。
- 引数 1 のデータ名 3 に指定した文字列長より，引数 3 に指定した入力領域長が短い場合の結果は保証しません。
- ダイアログボックスから入力された文字列長が引数 1 のデータ名 3 に指定した文字列長より短い場合，残りの入力領域は 1 バイトの空白文字で埋められます。

使用例

ダイアログボックスに「入力してください」と表示したあと，データをダイアログボックスから入力する例を次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
                :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ARG01.
         02 OUT-LEN PIC 9(09) USAGE COMP.
         02 IN-LEN  PIC 9(09) USAGE COMP.
       01 OUT-STR   PIC X(16).
       01 IN-AREA   PIC X(30).
                :
       PROCEDURE DIVISION.
      *** 引数情報をセット ***
            MOVE 16 TO OUT-LEN.
            MOVE '入力してください' TO OUT-STR.
            MOVE 30 TO IN-LEN.
      *** サービスルーチンを呼び出す ***
            CALL 'CBLINPUTDLG' USING ARG01 OUT-STR IN-AREA.
            IF RETURN-CODE < 0 THEN
                      CBLINPUTDLGエラー処理
            END-IF.
           :
```

作成したダイアログボックスの出力例を次に示します。

30.5.3 JCPOPUP

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 30-4　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
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

記述形式	内容
	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

モニタ画面の左上からの位置となります。また，1 行，1 列の幅は，表示する文字のフォントサイズに依存します。ただし，指定した値が不正な場合，および指定位置がモニタ画面外になる場合は，ポップアップウィンドウの出力位置は Windows 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

F1 キー：00　F9 キー：08　F17 キー：53

F2 キー：01　F10 キー：09　F18 キー：54

F3 キー：02　F11 キー：18　F19 キー：55

F4 キー：03　F12 キー：19　F20 キー：56

F5 キー：04　F13 キー：49　F21 キー：57

F6 キー：05　F14 キー：50　F22 キー：58

F7 キー：06　F15 キー：51　F23 キー：59

F8 キー：07　F16 キー：52　F24 キー：60

なお，F10 キーは通常システムキーとして割り当てられています。これを COBOL プログラム中で使用するためには，実行支援を使って，ユーザキーとして割り当てる必要があります。実行支援を使った画面環境の設定については，マニュアル「COBOL2002　操作ガイド」を参照してください。

- 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

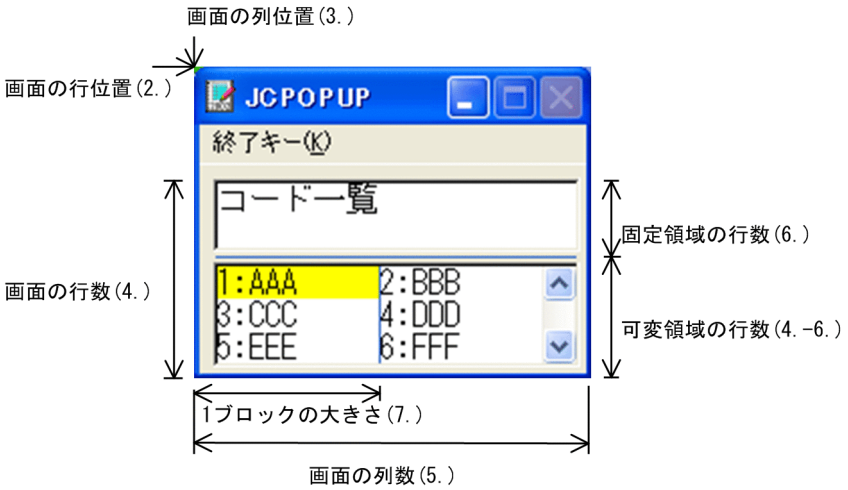
なし。

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。

CBLJCPOPENDKEY

JCPOPUP サービスルーチンの終了キーを拡張したい場合，環境変数 CBLJCPOPENDKEY に YES を指定します。拡張される終了キーと終了キーコードの対応を次に示します。なお，これらのキーはメニューに反映されず，キー操作だけ有効となります。

［Alt］＋［F1］キー：10　［PageDown］キー：24

［Alt］＋［F2］キー：11　［PageUp］キー：25

［Alt］＋［F3］キー：12　［Alt］＋［↑］キー：26

［Alt］＋［F4］キー：13　［Alt］＋［↓］キー：27

[Alt] ＋ [F5] キー：14　[Alt] ＋ [←] キー：28
[Alt] ＋ [F6] キー：15　[Alt] ＋ [→] キー：29
[Alt] ＋ [F7] キー：16　[Alt] ＋ [F8] キー：17
この環境変数の指定する値を指定しなかった場合や，YES 以外を指定した場合，終了キーは拡張されません。

30.6.1　CBLDBGINF

CBLDBGINF サービスルーチンは，COBOL 以外のプログラムから COBOL プログラムを呼ぶ場合，COBOL プログラムの例外発生時に異常終了時要約情報リストを出力するものです。

このサービスルーチンは，CALL 文で呼び出すのではなく，C 言語の構造化例外処理のフィルタ式から呼び出す必要があります。

記述例

```
#include <windows.h>
extern int WINAPI CBLDBGINF(DWORD);
                        /* CBLDBGINF 外部参照宣言 */
        :
  CBLDBGINF(GetExceptionCode())
        :
```

引数

発生した例外の種類を識別するコードを引数にします。GetExceptionCode()関数の戻り値をそのまま指定します。

戻り値

構造化例外のフィルタ式の評価結果の値を戻します。

- EXCEPTION_CONTINUE_SEARCH
  例外コードが EXCEPTION_BREAKPOINT，または EXCEPTION_SINGLE_STEP の場合
- EXCEPTION_EXECUTE_HANDLER
  例外コードが上記の二つ以外だった場合

なお，上記の EXCEPTION_CONTINUE_SEARCH および EXCEPTION_EXECUTE_HANDLER の詳細については，C または C++の各マニュアルの構造化例外処理に関する個所を参照してください。

規則

- このサービスルーチンは，C プログラムの構造化例外処理のフィルタ式から呼び出す必要があります。その他の方法によって呼び出した場合，実行時の動作は保証しません。
- 異常終了時要約情報リストには，-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，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情報が出力されます。リストの出力内容，および出力先については，「37.2　異常終了時要約情報リスト」を参照してください。
- 戻り値が EXCEPTION_EXECUTE_HANDLER のとき，CBLDBGINF は異常終了時要約情報リストを出力しません。

- このサービスルーチンを使用するプログラムでは，リンク時に以下のインポートライブラリを指定してください。

  cbl2k_32.lib

注意事項

COBOL2002 が動作していない環境で CBLDBGINF を呼び出した場合，エラーメッセージが表示されます。

使用例

CBLDBGINF サービスルーチンを呼び出す例を，次に示します。

```
#include <windows.h>
extern int WINAPI CBLDBGINF();
                        /* CBLDBGINF 外部参照宣言 */

_try{
 COBOL_SUB( );           /* COBOLプログラム実行 */
}

/* フィルタ式でのCBLDBGINF呼び出し */
_except (CBLDBGINF(GetExceptionCode())){
  /* 例外ハンドラブロック　例外発生時の処理をする */
          :
          :
}
```

30.6.2 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，「37.3　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

-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，「37.3.2 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。

30.7　変換・転記・演算

30.7.1　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 30-5　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（Unicode 機能を使用する場合）

Unicode 機能については，「28.　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　！　″　＃　＄　％　＆　’　（　）　＊　＋　，　－　．　／　：
  ；　＜　＝　＞　？　［　￥　］　＾　＿　＠　｛　｜　｝　。　「　」　、　・　゛　゜　ｰ
  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 "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 '.
       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.
                :
```

30.7.2 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.
           :
```

30.7.3 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 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 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 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 30-6　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

- 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。引数 1 が 2315 以外の場合，引数 2 には 0 が格納されます。
- 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。
- 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 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 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 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.
```

30.8　画面節（SCREEN SECTION および WINDOW SECTION）

30.8.1　CBLSGET

CBLSGET サービスルーチンは，キーボードからの入力を 1 文字受け取るものです。

形式

```
CALL ’CBLSGET’ USING 引数1
```

引数

引数 1 には，入力された文字を格納する領域を指定します。

この項目は，1 バイトの英数字項目でなければなりません。入力された文字は，対応するコードに変換されて，格納されます。入力文字とコードとの対応については，「図 30-1　入力文字とコードとの対応」を参照してください。

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を呼び出す前に，CBLCUR サービスルーチンでカーソルを位置づけておかないと，入力するときのカーソル位置が不定となります。
- このサービスルーチンは，画面節（WINDOW SECTION）の実行中には呼び出せません。呼び出した場合，実行時エラーになります。

  画面節（SCREEN SECTION）を実行中のときは，呼び出せます。
- 全角文字を入力した場合，引数 1 には NULL（X'00'）が格納されます。
- ［Alt］キー，［Ctrl］キー，および［Shift］キーなど，制御文字を入力した場合，入力は終了しません。

入力文字とコード

入力された文字とコードとの対応を次に示します。

図 30-1　入力文字とコードとの対応

下位4ビット	上位4ビット															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0			SP	0	@	P	゛(アクサングラーブ)	P				ー(長音)	タ	ミ		
1			!	1	A	Q	a	q	PF1	PF13		ア	チ	ム		
2			〃(引用符)	2	B	R	b	r	PF2	PF14		イ	ツ	メ		
3			#	3	C	S	c	s	PF3	PF15		ウ	テ	モ		
4			$	4	D	T	d	t	PF4	PF16		エ	ト	ヤ		
5			%	5	E	U	e	u	PF5	PF17		オ	ナ	ユ		
6			&	6	F	V	f	v	PF6	PF18	ヲ	カ	ニ	ヨ		
7			゛(アポストロフィー)	7	G	W	g	w	PF7	PF19	ァ	キ	ヌ	ラ		
8	後退		(	8	H	X	h	x	PF8	PF20	ィ	ク	ネ	リ		
9	⇤⇥		)	9	I	Y	i	y	PF9	PF21	ゥ	ケ	ノ	ル		
A			*	:	J	Z	j	z	PF10	PF22	ェ	コ	ハ	レ		
B	⇱		+	;	K	[	k	{	PF11	PF23	ォ	サ	ヒ	ロ		
C		→	(コンマ)，	<	L	¥	l	|	PF12	PF24	ャ	シ	フ	ワ		
D	「Return」	←	－(マイナス)	=	M	]	m	}	前頁	挿入	ュ	ス	ヘ	ン		
E	「Enter」	↑	(ピリオド)．	>	N	^	n	￣(オーバーライン)	次頁	削除	ョ	セ	ホ	゛(濁点)		
F		↓	/	?	O	_(アンダーライン)	o				ッ	ソ	マ	゜(半濁点)		

注　表中の「Return」(0D)は，文字キー側の［Enter］キーを，「Enter」(0E)は，テンキー側の［Enter］キーを表します。

30.8.2　CBLSETTITLE

CBLSETTITLE サービスルーチンは，画面節（SCREEN SECTION および WINDOW SECTION）の画面機能で，ウィンドウのタイトルバーに指定された文字列を表示するものです。

形式

```
CALL ’CBLSETTITLE’ USING 引数1 引数2
```

引数

- 引数 1 には，表示する文字列の長さを，バイト数で指定します。2 バイトの 2 進項目で指定します。指定できる文字列の長さは，1～1,024 バイトです。
- 引数 2 には，表示する文字列を指定します。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

規則

- 画面機能のウィンドウが表示されていないときにこのサービスルーチンを呼び出すと，ウィンドウが開いたときにタイトルバーに反映されます。
- このサービスルーチンを使って文字列を表示すると，すでにタイトルバーに表示されていた文字列は消去されます。
- 引数 1 で指定した長さが，引数 2 で指定した文字列の長さよりも長い場合，実行時の結果は保証しません。
- 引数 1 で指定した長さが，引数 2 で指定した文字列の長さよりも短い場合，引数 1 で指定した長さだけ出力されます。
- 引数 2 で指定した文字列の中に，システムが表示できないコードが含まれていた場合，実行時の動作は保証しません。
- 引数 2 で指定した文字列の中に，NULL（X'00'）が含まれていた場合，NULL 以降の文字列は出力されません。

30.9.1　CBLSQLERROR

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 30-7　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 30-8　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 30-7　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 'サンプル'.
 01 ODBC-UID            PIC X(10) VALUE 'sa'.
 01 ODBC-PWD            PIC X(10) VALUE SPACE.
 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。

30.9.2 CBLSQLSETOPT

CBLSQLSETOPT 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，SQLSetConnectOption 関数で接続関連を制御するオプションを設定します。

形式

```
CALL ’CBLSQLSETOPT’ USING 引数1 引数2
```

引数

- 引数 1 は，オプション種別を設定する 128 バイトの英数字項目です。
- 引数 2 は，引数 1 に関連づけられた値を 4 バイトの符号なし 2 進項目で指定します。引数 1 に指定するオプション種別に対して引数 2 に設定できる値を次に示します。

表 30-9 CBLSQLSETOPT サービスルーチンで設定できるオプション

引数 1 の設定値と機能	引数 2 の設定値と機能		注意事項
	設定値	機能	
SQL_COPT_SS_ENLIST_IN_XA SQL Server Version 6.5 以降の分散トランザクションの X/Open の XA トランザクションを行います。	1	XA トランザクションを ODBC 接続に関連づけます。	この指定によって XA トランザクションを ODBC 接続に関連づけた場合，COMMIT，ROLLBACK 文によるトランザクション管理はできません。それまでのトランザクションは，コミットされます。
	0	XA トランザクションを終了します。	

戻り値

0：正常終了した場合

-1：引数に誤りがある場合

-2：メモリ不足が発生した場合

規則

- 引数の属性，および引数に指定する値が上記と異なる場合の結果は保証しません。

SQLSetConnectOption 関数で ODBC の接続関連を制御するオプションを設定する場合

- CBLSQLSETOPT サービスルーチンで指定した接続オプションは，直後に実行される SQL 文で設定されます。SQLSetConnectOption 関数実行のタイミングを次に示します。
- CONNECT 文，または SET CONNECTION 文の場合
  CONNECT 文，または SET CONNECTION 文の処理後に関数を実行します。
- CONNECT 文，または SET CONNECTION 文以外の SQL 文
  各 SQL 文の処理に先立って関数を実行します。
- 指定した接続オプションの設定時に接続されている現行コネクションのデータソースが，該当するデータベースでない場合，その実行は保証しません。

使用例

SQLSetConnectOption 関数で ODBC の接続関連を制御するオプションを設定する場合

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
                :
       DATA DIVISION.
        WORKING-STORAGE SECTION.
                :
        01 OP-1 PIC X(128).
        01 OP-2 PIC 9(9) USAGE COMP.
                :
       PROCEDURE DIVISION.
                :
           *> XAトランザクションをODBC接続に関連づける
           MOVE 'SQL_COPT_SS_ENLIST_IN_XA' TO OP-1.
           MOVE 1 TO OP-2.

           CALL 'CBLSQLSETOPT' USING OP-1 OP-2.
           IF RETURN-CODE NOT = 0 THEN
             *> エラー処理
           END-IF.
           EXEC SQL … 1.
             SELECT
                :
```

1. CBLSQLSETOPT サービスルーチンを呼び出した直後に実行される埋め込み SQL 文（SELECT）で，SQL_COPT_SS_ENLIST_IN_XA の接続オプションが設定されます。

30.10　XMAP3 を使用した画面・帳票関連

XMAP3 を使用した書式印刷機能（Windows(x86) COBOL2002 で有効），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。

30.10.1　CBLXMAPERROR

CBLXMAPERROR サービスルーチンは，XMAP3 を使用した書式印刷機能（Windows(x86) COBOL2002 で有効），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 30-10　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 30-11　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。

30.11　その他の機能

30.11.1　CBLPUT

CBLPUT サービスルーチンは，画面に 1 文字を出力するものです。

形式

```
CALL ’CBLPUT’ USING 引数1
```

引数

引数 1 には，現在のカーソル位置に出力する文字を，1 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

規則

- 引数 1 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。
- このサービスルーチンを使用して文字を出力すると，カーソルは右へ 1 文字移動します。
- このサービスルーチンを呼ぶ前に，CBLCUR サービスルーチンでカーソルを位置づけておく必要があります。位置づけていないと，文字の表示される位置は不定となります。
- 画面節（WINDOW SECTION）を実行中にこのサービスルーチンを呼び出すと実行時エラーとなります。画面節（SCREEN SECTION）とは同時に実行できます。

使用例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
           :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 OP-1 PIC X(1) VALUE ’A’.
           :
       PROCEDURE DIVISION.
           :
           CALL ’CBLPUT’ USING OP-1.
           IF RETURN-CODE NOT = 0 THEN
             CBLPUTエラー処理
           END-IF.
           :
```

30.11.2　CBLGET

CBLGET サービスルーチンは，キーボードから文字を 1 文字入力するものです。

形式

```
CALL ’CBLGET’ USING 引数1
```

引数

引数 1 には，現在のカーソル位置から入力される文字を格納する領域を，1 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

規則

- このサービスルーチンを呼ぶ前に，CBLCUR サービスルーチンでカーソルを位置づけておく必要があります。位置づけていないと，入力時のカーソル位置は不定となります。
- 画面節（WINDOW SECTION）を実行中にこのサービスルーチンを呼び出すと実行時エラーとなります。画面節（SCREEN SECTION）とは同時に実行できます。

使用例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
           :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 OP-1 PIC X(1) VALUE SPACE.
           :
       PROCEDURE DIVISION.
           :
           CALL ’CBLGET’ USING OP-1.
           IF RETURN-CODE NOT = 0 THEN
             CBLGETエラー処理
           END-IF.
           :
```

30.11.3 CBLADTRM

CBLADTRM サービスルーチンは，ACCEPT 文の終了キーに任意のコードを追加するものです。

形式

```
CALL ’CBLADTRM’ USING 引数1 引数2
```

引数

- 引数 1 には，終了キーに指定するコードのバイト数を，2 バイトの 2 進項目で指定します。
- 引数 2 には，終了キーに指定するコードを指定します。この項目は，引数 1 で指定した長さの英数字項目（PIC X(n)：n=引数 1）でなければなりません。

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で指定した終了キーは，画面節（SCREEN SECTION および WINDOW SECTION）の ACCEPT 文に対して有効となります。ただし，ENTER キーと実行キーは本サービスルーチンで終了キーに追加できません。
- 画面節（WINDOW SECTION）の ACCEPT 文を実行した際に押された終了キーのコードは，WINDOW-KEYCODE 特殊レジスタに設定されます。
- 画面節（SCREEN SECTION）の ACCEPT 文を実行した際に押された終了キーのコードは，環境部の特殊名段落で CRT STATUS 句に指定したデータ名に設定されます。詳細については，「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」を参照してください。

使用例

終了キーに［Tab］キー，および［Esc］キーを追加する例を次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
           :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 OP-1 PIC 9(4) USAGE COMP VALUE 4.
       01 OP-2 PIC X(2) VALUE X'091B'.
           :
       PROCEDURE DIVISION.
           :
           CALL 'CBLADTRM' USING OP-1 OP-2.
           IF RETURN-CODE NOT = 0 THEN
             CBLADTRMエラー処理
           END-IF.
           :
```

終了キーとコード

終了キーとコードとの対応を「図 30-2　終了キーとコードとの対応（1 バイトコード）」，「図 30-3　終了キーとコードとの対応（2 バイトコード）」に示します。

終了キーは，日立パーソナルワークステーション 2020（以降，2020 と呼びます）のキーボードのキーで表しています。

2020 のキーボードにはあり，106 日本語キーボードにはないキーが幾つかあるため，106 日本語キーボードではこれらのキーを別のキーに置き換えることで対応しています。「図 30-4　2020 のキーボードと 106 日本語キーボードとの対応表」に 2020 のキーボードと 106 日本語キーボードの対応表を示します。

図 30-2　終了キーとコードとの対応（1 バイトコード）

下位4ビット	上位4ビット															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	テンキー		SP	0	@	P	゛（アクサングラーブ）	p				ー（長音）	タ	ミ		
1			!	1	A	Q	a	q			。（句点）	ア	チ	ム		
2			〃（引用符）	2	B	R	b	r			「	イ	ツ	メ		
3	ブレーク	中断	#	3	C	S	c	s			」	ウ	テ	モ		
4			$	4	D	T	d	t			、（読点）	エ	ト	ヤ		
5			%	5	E	U	e	u			・（中点）	オ	ナ	ユ		
6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7			゛（アポストロフィー）	7	G	W	g	w			ァ	キ	ヌ	ラ		
8	後退		(	8	H	X	h	x			ィ	ク	ネ	リ		
9			)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A		クリア	*	:	J	Z	j	z			ェ	コ	ハ	レ		
B		エスケープ	+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C		→	（コンマ），	<	L	¥	l	|			ャ	シ	フ	ワ		
D		←	－（マイナス）	=	M	]	m	}			ュ	ス	ヘ	ン		
E	End	↑	（ピリオド）．	>	N	^	n	￣（オーバーライン）			ョ	セ	ホ	゛（濁点）		
F	End	↓	/	?	O	（アンダーライン）＿	o	後退			ッ	ソ	マ	゜（半濁点）		

注　表中の  は，［Ctrl］キーを同時に押すことを表します。

図 30-3　終了キーとコードとの対応（2 バイトコード）

（80H+）

下位4ビット	上位4ビット															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0					PAI	PA1	変換 取消	無変 換								
1					SRQ	PA2	変換	変換					PF1	PF1		
2					テスト	PA3	ロー マ字	ロー マ字					PF2	PF2		
3						割込	半角	半角				挿入	PF3	PF3		
4					スクリーン	取消	ひら がな	ひら がな				削除	PF4	PF4		
5					FM	DUP	→	→				→	PF5	PF5		
6					データ	フィールド	←	←				←	PF6	PF6		
7					印刷 中止	印刷	↓	↓				↓	PF7	PF7		
8							↑	↑				↑	PF8	PF8		
9							文頭	前頁					PF9	PF9		
A							文末	次頁					PF10	PF10		
B							挿入	機能 拡張					PF11	PF11		
C							削除	000					PF12	PF12		
D							クリア						英数	英数		
E							実行 （送信）	実行 （送信）					カタ カナ	カタ カナ		
F							⇤→	⇤→								

注　表中の  は［Ctrl］キーを同時に押すことを，

 は［Shift］キーを同時に押すことを表します。

図 30-4　2020 のキーボードと 106 日本語キーボードとの対応表

2020 のキーボード	106 日本語キーボード
ブレーク	Break
後退	BackSpace
	Tab
	Home
	Enter
中断	Pause
クリア	なし
エスケープ	Esc
PAI	Ctrl + Alt + F1
SRQ	Ctrl + Alt + F2
テスト	Ctrl + Alt + F3
スクリーン	Ctrl + Alt + F9
FM	Ctrl + Alt + F10
データ	Ctrl + Alt + F11
印刷中止	Ctrl + Alt + F12
PA1	Alt + F1
PA2	Alt + F2
PA3	Alt + F3
割込	Alt + F8
取消	Alt + F9
DUP	Alt + F10
フィールド	Alt + F11
印刷	Alt + F12
変換取消	Ctrl + 無変換
文頭	Ctrl + PageUp
文末	Ctrl + PageDown
四角→	Alt + →
四角←	Alt + ←
四角↓	Alt + ↓
四角↑	Alt + ↑
前頁	PageUp
次頁	PageDown
機能拡張	Alt + F5
000	Alt + 0（テンキー）
実行	Alt + Enter
挿入	Insert
削除	Delete

30.11.4　CBLDLTRM

CBLDLTRM サービスルーチンは，ACCEPT 文の終了キーから任意のコードを削除するものです。

形式

```
CALL 'CBLDLTRM' USING 引数1 引数2
```

引数

- 引数 1 には，終了キーから削除するコードのバイト数を，2 バイトの 2 進項目で指定します。
- 引数 2 には，終了キーから削除するコードを指定します。この項目は，引数 1 で指定した長さの英数字項目（PIC X(n)：n=引数 1）でなければなりません。

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で呼び出した終了キーは，画面節（SCREEN SECTION および WINDOW SECTION）の ACCEPT 文に対して有効となります。
- 終了キーとコードとの対応については，「30.11.3　CBLADTRM」で示した「図 30-2　終了キーとコードとの対応（1 バイトコード）」，「図 30-3　終了キーとコードとの対応（2 バイトコード）」を参照してください。

使用例

終了キーから［Tab］キー，および［Esc］キーを削除する例を次に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
           :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 OP-1 PIC 9(4) USAGE COMP VALUE 4.
       01 OP-2 PIC X(2) VALUE X'091B'.
           :
       PROCEDURE DIVISION.
           :
           CALL 'CBLDLTRM' USING OP-1 OP-2.
           IF RETURN-CODE NOT = 0 THEN
             CBLDLTRMエラー処理
           END-IF.
           :
```

30.11.5　CBLCNSL

CBLCNSL サービスルーチンは，キーボードからの読み込み待ちの文字があるかどうかを検査するものです（ただし，このシステムでは，常に読み込み待ちの文字ありとなります）。

形式

```
CALL 'CBLCNSL' USING 引数1
```

引数

引数 1 には，キーボードからの読み込み待ちの文字があるかどうかの情報が次のように設定されます。この項目は，2 バイトの 2 進項目でなければなりません。

0：読み込み待ちの文字なし。

0 以外：読み込み待ちの文字あり。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

注意事項

このサービスルーチンは，常に 0 以外を引数に設定します。

使用例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
           :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 OP-1 PIC 9(4) USAGE COMP VALUE 0.
           :
       PROCEDURE DIVISION.
           :
           CALL 'CBLCNSL' USING OP-1.
           IF RETURN-CODE NOT = 0 THEN
             CBLCNSLエラー処理
           END-IF.
           :
```

30.11.6 CBLBELL

CBLBELL サービスルーチンは，警告音を鳴らす機能を提供するものです。

形式

```
CALL 'CBLBELL'
```

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

使用例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
           :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
           :
       PROCEDURE DIVISION.
```

```
        :
        CALL 'CBLBELL'.
        IF RETURN-CODE NOT = 0 THEN
          CBLBELLエラー処理
        END-IF.
        :
```

30.11.7 CBLCUR

CBLCUR サービスルーチンは，画面上の指定された位置にカーソルを移動するものです。

形式

```
CALL 'CBLCUR' USING 引数1 引数2
```

引数

- 引数 1 は使用しません。
- 引数 2 には，カーソルを移動する行とカラムを指定します。この項目は次の形式の集団項目とします。

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 PIC 9(04) USAGE COMP.`	カーソルを移動する行の番号を，2 バイトの 2 進項目で指定する。
`02 データ名3 PIC 9(04) USAGE COMP.`	カーソルを移動するカラム位置を，2 バイトの 2 進項目で指定する。

戻り値

0：正常終了した場合

-1：エラーが発生した場合

規則

- 行の番号は 1～24，カラム位置は 1～80 の値でなければなりません。
- 画面節（WINDOW SECTION）を実行中にこのサービスルーチンを呼び出すと実行時エラーとなります。画面節（SCREEN SECTION）とは同時に実行できます。

使用例

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SAMPLE1.
           :
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 OP-1 PIC X VALUE SPACE.
       01 OP-2.
         02 OP-2-1 PIC 9(4) USAGE COMP VALUE 10.
         02 OP-2-2 PIC 9(4) USAGE COMP VALUE 20.
```

```
        :
    PROCEDURE DIVISION.
        :
        CALL 'CBLCUR' USING OP-1 OP-2.
        IF RETURN-CODE NOT = 0 THEN
          CBLCURエラー処理
        END-IF.
        :
```

31 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。

31.1　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。

31.1.1　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。

31.1.2　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，「29.　数字項目のけた拡張機能（Windows(x64) COBOL2002 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。

31.2　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。

31.2.1　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。

31.2.2　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

このシステムのコンパイラのコード系はシフト JIS（SJIS）ですが，EUC（UJIS）を使っているコンパイラもあります。

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。

注※

Microsoft C／C++の関数

31.3　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。

なお，スタックオーバフロー発生後のプログラムの動作は保証しません。

31.3.1　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

Windows(x64) COBOL2002 の場合は 8 バイトとなります。

注※2

Windows(x64) COBOL2002 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※

Windows(x64) COBOL2002 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。

31.3.2　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，「31.3.1　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

31.3.3　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。

スタック領域のサイズ変更方法は，-STACK オプションによって使用できるスタック領域のサイズを変更します。-STACK オプションについては，「38.1.3　オプション」を参照してください。

32 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。

32.1　最適化のレベル

32.1.1　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。

32.2.1　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。

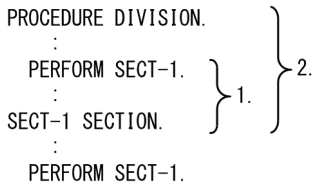
（例）

```
        :
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。

32.2.2　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

(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
- SYMBOLIC KEY データ名
- 埋め込み SQL 宣言節のデータ名
- 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
- SQL データ操作の文の一意名，データ名
- 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

(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

(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

32.2.3　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。

32.2.4　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。

（最適化前）

```
PROCEDURE DIVISION.
 MOVE 23 TO A.
 MOVE A  TO B.    ·····1.
 COMPUTE D = A + B + 4.   ···2.
 DISPLAY D.
 STOP RUN.
```

（最適化後）

```
PROCEDURE DIVISION.
 MOVE 23 TO A.  → 削除
 MOVE A  TO B.  → MOVE 23 TO B.  → 削除
 COMPUTE D = 50.
 DISPLAY D.
 STOP RUN.
```

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。

32.2.5　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。

32.2.6　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。

32.2.7　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。

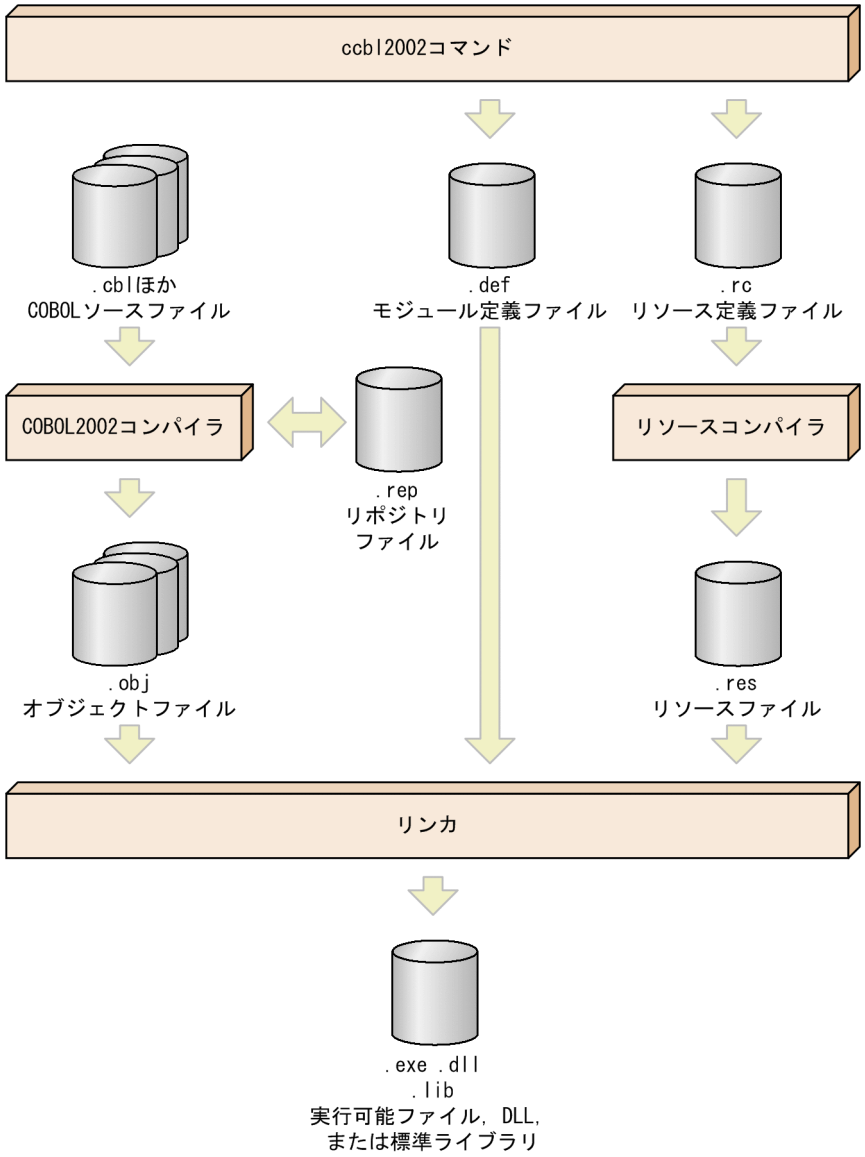
33 COBOL ソースの作成とコンパイル

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。

33.1　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 33–1　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（.obj）が生成されます。このとき，同時に次のファイルが生成されます。
   - モジュール定義ファイル（.def）
   - リソース定義ファイル（.rc）
2. 生成されたリソース定義ファイルを基に，リソースコンパイラによってリソースファイル（.res）が生成されます。
3. オブジェクトファイル，モジュール定義ファイル，リソースファイルがリンカに取り込まれ，実行可能ファイル（.exe），ダイナミックリンクライブラリ（.dll），または標準ライブラリ（.lib）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 F　COBOL で使用するファイル」を参照してください。

33.2　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。

33.2.1　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 Windows 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。

スラント（/）　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，.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

33.2.2　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。

33.2.3　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，.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，「33.6　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，.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，.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 33–1 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

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。

33.3　さまざまな形態の COBOL 原始プログラムのコンパイル

33.3.1　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。

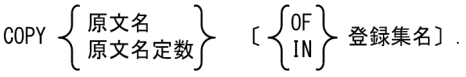
(2)　登録集原文のファイル形式

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，.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せん。
- 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，「33.5.12　他システムとの移行の設定」の「(12)　-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 SET コマンドで次のように設定しておきます。

形式

```
SET 登録集名=登録集検索フォルダ〔; …〕
```

規則

- 登録集検索フォルダには，原文名に対応するファイルを検索する任意のフォルダ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，フォルダ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，フォルダ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. .cob
5. .ocb
6. .cbf
7. .ocf

フォルダによる検索順序※

1. 登録集環境変数で設定したフォルダ
2. 環境変数 CBLLIB で指定したフォルダ

3. カレントフォルダ

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.の順にフォルダを検索し，目的のファイルがなければ，次に"ファイル名.cob"で同様に検索します。

注※

開発マネージャを使用する場合の検索順序については，「33.6.3　コンパイラ環境変数の詳細」の「(10)　CBLLIB」の説明を参照してください。

33.3.2　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 DLL 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。

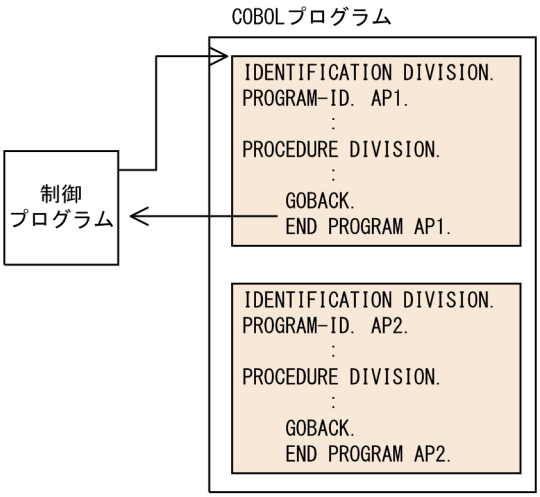
(1)　コンパイラオプションとの関係

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，「33.5.5　最終生成物の種類（プロジェクトの種類）の設定」を参照してください。

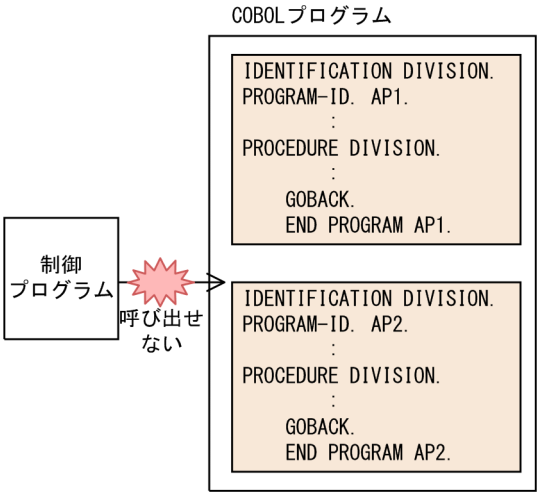
(2)　主プログラム指定

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，「34. 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 E 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

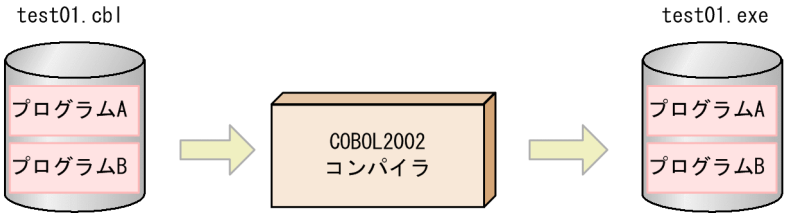
一つのコンパイル単位に最外側のプログラムが複数個含まれていても，これらの最外側のプログラムはすべて別のコンパイル単位の CALL 文で呼び出せます。

(6)　ソースファイルと各種ファイルの関係

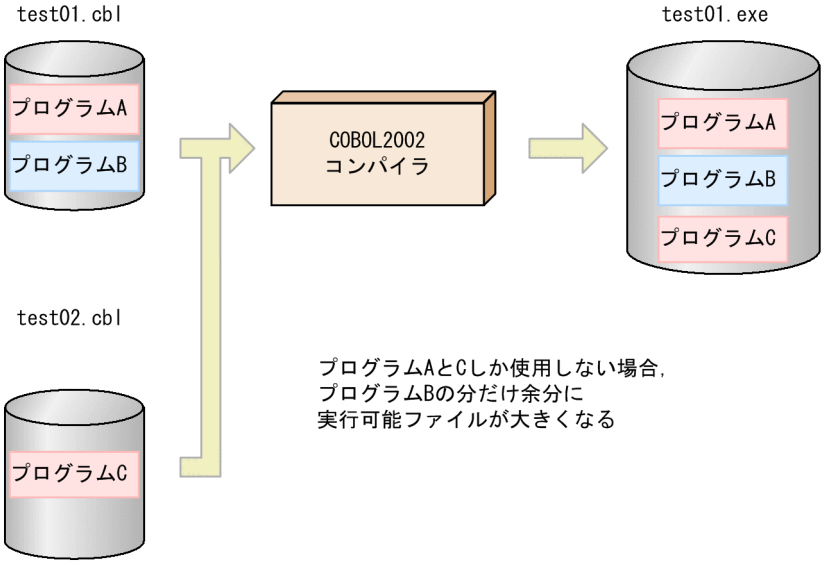
スタックコンパイル時にコンパイラが生成するファイルは，通常のコンパイル時に生成するファイルと同じです。

(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。



33.3.3　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。

33.3.4　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 E.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
```

33.4　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，DLL（ダイナミックリンクライブラリ）を生成できます。
- 開発マネージャを利用する方法
  開発マネージャを利用して，プログラムのコンパイル，リンクができます。
  開発マネージャを利用したコンパイルの方法については，マニュアル「COBOL2002　操作ガイド」を参照してください。
- cblbuild2k コマンドによる方法
  開発マネージャの［ビルド］［リビルド］をコマンドラインから起動する方法です。
- 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。

33.4.1　ccbl2002 コマンド

COBOL プログラムをコマンドプロンプトからコンパイルするには，ccbl2002 コマンドを使用します。

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，「33.5　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（.obj）

- ライブラリファイル（.lib）
- リソース定義ファイル（.rc）
- モジュール定義ファイル（.def）
- リソースファイル（.res）

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，ファイル名を引用符（ " ）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，「33.5　コンパイラオプション」を参照してください。

コマンドファイルを使った引数の指定方法

ccbl2002 コマンドにオプションやファイル名を指定する場合，コマンドラインに直接指定する代わりに，コマンドファイルを使っても指定できます。コマンドファイルを指定すると，ファイルの中身がコマンドライン上に展開され，コマンドの一部となります。

コマンドライン上には，コマンド名以降であればどこにでもコマンドファイルを指定できます。コマンドファイル名の先頭には，@を付けて指定します。

（コマンドファイルの使用例）

- cblsrc.txt の内容

```
-Main,System sample.cbl
```

- objfiles.txt の内容

```
prog1.obj prog2.obj
```

- 入力コマンド

```
ccbl2002 @cblsrc.txt @objfiles.txt -OutputFile test1.exe
```

- 展開後のコマンドライン

```
ccbl2002 -Main,System sample.cbl prog1.obj prog2.obj
         -OutputFile test1.exe
```

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

終了コード	意味	出力される エラーメッセージのレベル
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，または次のコマンドを入力された場合に ccbl2002 コマンドのヘルプを出力できます。

形式

```
ccbl2002 〔 {-Help | -?} 〕
```

33.4.2 ccbl コマンド

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（コンパイル用のバッチファイルなど）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，DLL，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 I　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，または次のコマンドを入力することで 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。

33.4.3 cblbuild2k コマンド

cblbuild2k コマンドは，プロジェクト単位に実行可能ファイル，DLL，または標準ライブラリを生成する方法です。このコマンドは，開発マネージャの［ビルド］［リビルド］メニューに対応するものです。

(1) プロジェクトの概念と定義方法

プロジェクトとは，実行単位を構成する資源を一括して管理するための概念です。プロジェクトには，実行単位を構成するソースファイル（登録集原文を含む）や，登録集原文を生成する基となった各種定義ファイル，およびコンパイル時の環境やオプションなどを登録・定義できます。定義したプロジェクトの内容は，プロジェクトマスタファイル（.hmf）に格納されます。プロジェクトマスタファイルには，複数の関連するプロジェクトをまとめて定義できます。なお，COBOL85 Version 5 以前のプロジェクトファイルを対象とするときは，元のファイルと同じフォルダに COBOL2002 のプロジェクトマスタファイルが作成されます。

プロジェクト，およびプロジェクトマスタは，開発マネージャ上で定義します。定義方法については，マニュアル「COBOL2002　操作ガイド」を参照してください。

プロジェクトマスタファイルの保存について

プロジェクトマスタファイルは，ビルド，またはリビルドによってプロジェクトマスタ内のプロジェクトの管理情報に変更が生じた場合，自動的に保存されます。ただし，プロジェクトマスタファイル読み込み時に無効なファイルやオプションなどの警告メッセージが表示されたときは，プロジェクトマスタファイルの自動保存は行われません。

(2) ビルドとリビルド

ビルド，リビルドは，プロジェクト内の資源の依存関係を基に，プロジェクト単位に実行可能ファイル，DLL，または標準ライブラリを生成する方法です。

ソースファイルからコンパイル，リンクをするだけでなく，例えば，必要な登録集原文が定義ファイルから生成されていなければ，コンパイルに先立って登録集原文を生成します。

ビルド，リビルドには次のような違いがあります。

(a) ビルド

実行可能ファイル，DLL，または標準ライブラリを生成したあとに定義ファイル，ソースファイル，オプションなどを変更した場合，必要な資源だけを再生成，再コンパイルして実行可能ファイル，DLL，または標準ライブラリを作り直す方法です。

(b) リビルド

変更の有無に関係なくすべての資源を再生成，再コンパイルして実行可能ファイル，DLL，または標準ライブラリを作り直す方法です。

再生成，再コンパイルが実行されるケースやその対象など，ビルド，リビルドの詳細については，マニュアル「COBOL2002　操作ガイド」を参照してください。

(3) コマンドの形式

cblbuild2k コマンドの指定形式を次に示します。

形式

```
cblbuild2k
   { 〔-a〕 〔-p〕 〔-y〕 〔-c〕 〔-m〕 入力ファイル名 … | -h | -? }
```

-a

このオプションを指定するとリビルドが実行されます。

指定しない場合はビルドが実行されます。

-p

このオプションを指定すると，プロジェクトマスタに定義されているすべてのプロジェクトに対してビルドが実行されます。

指定しない場合は，プロジェクトマスタで作業中プロジェクトに設定されたプロジェクトに対して，ビルドが実行されます。

-y

このオプションは，COBOL85 Version 5 以前のバージョンから COBOL2002 へ移行するための機能です。

このオプションを指定すると，プロジェクトマスタを作成するかどうかの確認メッセージが表示されません。

指定しない場合は，プロジェクトマスタを作成するかどうかの確認メッセージが表示されます。

-c

このオプションは，COBOL85 Version 5 以前のバージョンから COBOL2002 へ移行するための機能です。

このオプションを指定すると，プロジェクトマスタファイルの作成だけを行い，プロジェクトのビルドは実行しません。

指定しない場合は，プロジェクトマスタファイルの作成を行ったあと，プロジェクトのビルドが実行されます。

-m

このオプションは，このシステム以外で作成されたプロジェクトマスタファイルを COBOL2002 へ移行するための機能です。

このオプションを指定すると，入力ファイルがこのシステム以外の開発マネージャで保存されたプロジェクトマスタファイルの場合でも警告メッセージを表示しません。

指定しない場合は，警告メッセージを表示します。

入力ファイル名

処理の対象となるファイルを指定します。入力ファイルには，次のファイルが指定できます。

- プロジェクトマスタファイル（.hmf）

入力ファイル名の拡張子を省略した場合は，.hmf が仮定されます。また，入力ファイルには複数のファイルが指定できます。

また，COBOL85 で作成したプロジェクトファイルおよびプロジェクトマスタファイルも入力ファイルにできます。

「-」（ハイフン）で始まる名称を指定した場合はエラーになります。

-h または-?

コマンドの説明が表示されます。

規則

- オプションの記号には「-」の代わりに「/」も使用できます。
- 指定されたオプションは，すべての入力ファイルに対して有効となります。

(4) 割り込み

cblbuild2k コマンドの実行中に［Ctrl］＋［C］キーを押すと，処理を中断して終了します。ただし，各ソースのコンパイル単位で中断を受け付けるため，すぐには中断しない場合があります。

(5) 終了コード

cblbuild2k 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	cblbuild2k は正常終了した。
1	cblbuild2k でエラーが発生した。
2	cblbuild2k がユーザによって中断された。

終了コード 1 が返ってくる条件を，以下に示します。

- プログラム内で内部エラーが発生した
- cblbuild2k コマンドが呼び出すコンパイラ，およびリンカが，目的とする生成物を生成できなかった
- ユーザが「新しいファイルの種類」で登録した生成ツールが，目的とする生成物を生成できなかった
  ただし，これは生成ツールが以下の終了コードを返す仕様である必要があります。
  目的とする生成物を生成できた場合：0
  目的とする生成物を生成できなかった場合：0 以外

33.5　コンパイラオプション

ここでは，ccbl2002 コマンドおよび開発マネージャ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 I　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。

33.5.1　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。

33.5.2　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（-）またはスラント（/）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。

33.5.3　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，「33.6　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
-Dll	• -Main • -SimSub
-Compile	• -OutputFile • -ManifestFileExt
-Compile,CheckOnly	• -DebugInf • -DebugCompati • -DebugData • -TDInf • -CVInf • -DebugRange • -TestCmd
-StdMIA -Std85 -Std2002	• -V3Spec • -StdVersion • -CompatiM7

指定したオプション	無効となるオプション
	• -CompatiV3 • -H8Switch • -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 • -CompatiM7 • -NumAccept • -CmDol • -MaxDigits38 • -IntResult,DecFloat40 • -LiteralExtend • -V3RecEased
-Compati85,Syntax	• -LiteralExtend
-Repository,Gen	• -Compile
-UniObjGen	• -JPN • -CompatiV3※ • -V3Rec※ • -V3RecFCSpace※ • -V3RecEased※
-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	• -LiteralExtend
-SimMain -SimSub -SimIdent	• -LiteralExtend

注※

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。

(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 ※3 • -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
-StdMIA,14	• -StdMIA,13
-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)

指定したオプション	仮定されるオプション
-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。

注※3

-CompatiV3 オプションと-JPN,V3JPN オプションまたは-JPN,V3JPNSpace オプションを同時に指定した場合，-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
-DllInit	-Dll
-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
-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，「33.5.4 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）

-BigEndian ｛,Bin ｜,Float｝＋

に対して

-BigEndian,Bin -BigEndian,Float

と指定した場合，Bin サブオプションと Float サブオプションの両方が有効となります。これは，「-BigEndian,Bin,Float」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「-Lib,CUI」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「-Lib,GUI」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。
- 打ち消しのオプションは，コマンドラインや環境変数 CBLCOPT2002 に指定したオプションを打ち消し，オプション指定によって仮定されるオプションは打ち消しません。

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。

33.5.4　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。

なお，打ち消しのオプション（-noXXX）を開発マネージャで指定する場合は，プロジェクト設定ダイアログボックスの［ユーザ設定］タブを使用してください。［ユーザ設定］タブの詳細は，マニュアル「COBOL2002　操作ガイド」のオプションの設定方法の説明を参照してください。

(1)　最終生成物の種類（プロジェクトの種類）

最終生成物※の種類（開発マネージャでは，プロジェクトの種類）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，DLL，または標準ライブラリのことを示します。

表中の参照先の番号（(1)，(2)，…）は「33.5.5　最終生成物の種類（プロジェクトの種類）の設定」の番号（(1)，(2)，…）と対応しています。

項番	オプション	マネージャ	機能	参照先
1	-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。開発マネージャでは，次のオプションが該当する。 System 指定メインプログラム V3 指定メインプログラム	(1) -Main オプション
2	−	○	メインプログラムなし	(2) メインプログラムなし
3	-Dll，{Stdcall ｜ Cdecl}（Windows(x86) COBOL2002 で有効） -Dll（Windows(x64) COBOL2002 で有効） -noDll	○	DLL の形式を指定する。 開発マネージャでは，次のオプションが該当する。 最終生成物の種類がダイナミックリンクライブラリ StdCall：Dll の属性を stdcall にする Cdecl：Dll の属性を cdecl にする	(3) -Dll オプション（最終生成物の種類がダイナミックリンクライブラリ）
4	−	○	標準ライブラリを作成する	(4) 標準ライブラリ

（凡例）

○：開発マネージャから指定できる

−：コマンドライン（ccbl2002 コマンド）に対応するオプションがない

(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。

表中の参照先の番号（(1)，(2)，…）は「33.5.6 他製品との連携の設定」の番号（(1)，(2)，…）と対応しています。

項番	オプション	マネージャ	機能	参照先
1	-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。	(1) -SQL オプション
2	-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。	(2) -SQLDisp オプション
3	-RDBTran -noRDBTran	○	COMMIT 文／ROLLBACK 文を HiRDB による索引編成ファイルに対して適用する。	(3) -RDBTran オプション

項番	オプション	マネージャ	機能	参照先
4	-IsamExtend〔,Zone〕 -noIsamExtend	○	Btrieve（Pervasive.SQL）による索引編成ファイルを使用する。	(4)　-IsamExtend オプション（Windows(x86) COBOL2002 で有効）
5	-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。	(5)　-XMAP オプション（Windows(x86) COBOL2002 で有効）
6	-OpenTP1 -noOpenTP1	○	OpenTP1 を使用したデータコミュニケーション機能を使用できるようにする。	(6)　-OpenTP1 オプション

（凡例）

○：開発マネージャから指定できる

(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。

表中の参照先の番号（(1)，(2)，…）は「33.5.7　実行の設定」の番号（(1)，(2)，…）と対応しています。

項番	オプション	マネージャ	機能	参照先
1	-NumAccept -noNumAccept	○	ACCEPT 文に数字項目を指定できるようにする。	(1)　-NumAccept オプション
2	-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。	(2)　-NumCsv オプション
3	-MultiThread -noMultiThread	○	マルチスレッド対応 COBOL プログラムを作成する。	(3)　-MultiThread オプション
4	-MainNotCBL -noMainNotCBL	○	すべて副プログラムとして作成する。	(4)　-MainNotCBL オプション
5	-DllInit -noDllInit	○	DLL を初期化する。	(5)　-DllInit オプション
6	－	○	メインファイルを指定する（開発マネージャで，プロジェクトのメインプログラムにするソースファイルを指定する）	(6)　メインファイルを指定する

（凡例）

○：開発マネージャから指定できる

－：コマンドライン（ccbl2002 コマンド）に対応するオプションがない

(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。

表中の参照先の番号（(1)，(2)）は「33.5.8 プログラムの最適化の設定」の番号（(1)，(2)）と対応しています。

項番	オプション	マネージャ	機能	参照先
1	-Optimize，{0 | 1 | 2 | 3}	○	コンパイル時の最適化のレベルを指定する。	(1) -Optimize オプション
2	-ScreenSpeed -noScreenSpeed	○	画面の表示速度を重視する。	(2) -ScreenSpeed オプション

（凡例）

○：開発マネージャから指定できる

(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。

表中の参照先の番号（(1)，(2)，…）は「33.5.9 デバッグの設定」の番号（(1)，(2)，…）と対応しています。

項番	オプション	マネージャ	機能	参照先
1	-DebugLine -noDebugLine	○	デバッグ行を有効にする。	(1) -DebugLine オプション
2	-DebugInf〔,Trace〕 -noDebugInf	○	異常終了時，エラー要約情報を出力する。	(2) -DebugInf オプション
3	-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	(3) -DebugCompati オプション
4	-DebugData〔,ValueHex〕 -noDebugData	○	データ例外を検出する。	(4) -DebugData オプション
5	-TDInf -noTDInf	○	テストデバッグ情報を出力する。	(5) -TDInf オプション
6	-CVInf -noCVInf	○	カバレージ情報を出力する。	(6) -CVInf オプション
7	-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。	(7) -DebugRange オプション

項番	オプション	マネージャ	機能	参照先
8	-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。	(8)　-TestCmd オプション
9	-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。	(9)　-SimMain オプション
10	-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。	(10)　-SimSub オプション
11	-SimIdent -noSimIdent	○	副プログラム（一意名 CALL 文）をシミュレーションする。	(11)　-SimIdent オプション

（凡例）

○：開発マネージャから指定できる

(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。

表中の参照先の番号（(1)，(2)，…）は「33.5.10　リンクの設定」の番号（(1)，(2)，…）と対応しています。

項番	オプション	マネージャ	機能	参照先
1	-StdCall -noStdCall	○	stdcall 呼び出し指示ファイルを有効にする。	(1)　-StdCall オプション（Windows(x86) COBOL2002 で有効）
2	-StdCallFile .cbw ファイル名 -noStdCallFile	○	stdcall 呼び出し指示ファイル名を指定する。	(2)　-StdCallFile オプション（Windows(x86) COBOL2002 で有効）
3	-Lib, {GUI | CUI}	○	GUI モード／CUI モードのどちらの実行時ライブラリをリンクするかを指定する。	(3)　-Lib オプション
4	−	○	ライブラリの指定。	(4)　ライブラリの指定
5	−	○	リンケージを行わない。	(5)　リンケージを行わない
6	-Compile, {CheckOnly | NoLink} -noCompile	×	コンパイルの処理範囲を指定する。	(6)　-Compile オプション
7	-DefFile .def ファイル名 -noDefFile	×	生成する.def ファイル名を指定する。	(7)　-DefFile オプション
8	-IconFile アイコンファイル名 -noIconFile	×	アイコンファイル名を指定する。	(8)　-IconFile オプション

項番	オプション	マネージャ	機能	参照先
9	-ResrcFile ファイル名 -noResrcFile	×	生成するリソース定義ファイル名を指定する。	(9)　-ResrcFile オプション
10	-OutputFile ファイル名 -noOutputFile	×	生成する実行可能ファイル名，または DLL ファイル名を指定する。	(10)　-OutputFile オプション
11	-Link オプションの並び -noLink	U	リンカに渡すオプションを指定する。	(11)　-Link オプション
12	-DynamicLink,Call -noDynamicLink	○	定数指定の CALL 文実行に動的リンク機能を使用するときに指定する。	(12)　-DynamicLink オプション
13	-ManifestFileExt -noManifestFileExt	○	マニフェストファイルを実行可能ファイルや DLL に埋め込まず，外部マニフェストファイルとするときに指定する。	(13)　-ManifestFileExt オプション

（凡例）

○：開発マネージャから指定できる

U：開発マネージャで使用する場合，［ユーザ設定］タブに指定する

×：開発マネージャから指定できない

－：コマンドライン（ccbl2002 コマンド）に対応するオプションがない

(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。

表中の参照先の番号（(1)，(2)，…）は「33.5.11　規格の設定」の番号（(1)，(2)，…）と対応しています。

項番	オプション	マネージャ	機能	参照先
1	-StdMIA｛,13 ｜,14｝＋ -noStdMIA※	○	MIA 仕様の範囲外チェックをする。	(1)　-StdMIA オプション
2	-Std85｛,｛High ｜ Middle ｜ Low｝ ｜,Obso ｜,Report｝＋ -noStd85※	○	JIS 仕様をチェックする。	(2)　-Std85 オプション
3	-Std2002｛,OutRange ｜,Obso ｜,Archaic｝＋ -noStd2002	○	COBOL2002 規格仕様をチェックする。	(3)　-Std2002 オプション
4	-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。	(4)　-StdVersion オプション

（凡例）

○：開発マネージャから指定できる

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。

表中の参照先の番号（(1)，(2)，…）は「33.5.12 他システムとの移行の設定」の番号（(1)，(2)，…）と対応しています。

項番	オプション	マネージャ	機能	参照先
1	-CompatiM7 -noCompatiM7	○	MIOS7 COBOL85 との互換機能を有効にする。	(1) -CompatiM7 オプション (Windows(x86) COBOL2002 で有効)
2	-CompatiV3※ -noCompatiV3	○	VOS3 COBOL85 との互換機能を有効にする。	(2) -CompatiV3 オプション
3	-Compati85 {,IoStatus | ,Linage | ,Call | ,Power | ,Syntax | ,IDParag | ,RsvWord | ,NoPropagate | ,All} ＋ -noCompati85	○	COBOL85 互換機能を有効にする。	(3) -Compati85 オプション
4	-H8Switch -noH8Switch	○	HITAC8000 シリーズの仕様でコンパイルする。	(4) -H8Switch オプション
5	-Cblctr -noCblctr	○	CBL-CTR 特殊レジスタを使用できるようにする。	(5) -Cblctr オプション
6	-DigitsTrunc -noDigitsTrunc	○	転記文で上位けたを切り捨てる。	(6) -DigitsTrunc オプション
7	-IgnoreLCC -noIgnoreLCC	○	行送り制御文字を無視する。	(7) -IgnoreLCC オプション
8	-CmAster -noCmAster	○	1 カラム目が'*'の行を注記行とする。	(8) -CmAster オプション
9	-CmDol -noCmDol	○	7 カラム目が'$'の行を注記行とする。	(9) -CmDol オプション
10	-Comp5 -noComp5	○	COMP-5 を指定できるようにする。	(10) -Comp5 オプション

項番	オプション	マネージャ	機能	参照先
11	-V3Spec〔,CopyEased〕 -noV3Spec	○	VOS3 COBOL85 に対する COBOL2002 固有の構文をチェックする。	(11)　-V3Spec オプション
12	-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。	(12)　-V3ConvtName オプション
13	-Switch,｛EBCDIC ｜ EBCDIK｝〔,Unprintable〕〔,noApplyJpnItem〕 -noSwitch	○	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。	(13)　-Switch オプション
14	-V3Rec,｛Fixed ｜ Variable｝※ -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。	(14)　-V3Rec オプション
15	-V3RecFCSpace -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。	(15)　-V3RecFCSpace オプション
16	-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。	(16)　-V3RecEased オプション
17	-DoubleQuote -noDoubleQuote	○	引用符（ " ）を分離符とみなしてコンパイルする。	(17)　-DoubleQuote オプション
18	-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。	(18)　-BigEndian オプション
19	-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。	(19)　-VOSCBL オプション
20	-PortabilityCheck｛,Literal ｜,Numeric｝＋ -noPortabilityCheck	○	移行向けチェック機能を有効にする。	(20)　-PortabilityCheck オプション
21	-IgnoreAPPLY,FILESHARE -noIgnoreAPPLY	○	APPLY FILE-SHARE 句を覚え書きとみなす。	(21)　-IgnoreAPPLY,FILESHARE オプション

（凡例）

○：開発マネージャから指定できる

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。

(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。

表中の参照先の番号（(1)，(2)）は「33.5.13　リスト出力の設定」の番号（(1)，(2)）と対応しています。

項番	オプション	マネージャ	機能	参照先
1	-SrcList，{OutputAll | CopyAll | CopySup | NoCopy}〔,NoFalsePath〕〔,DataLoc〕 -noSrcList	○	コンパイルリストを出力する。	(1)　-SrcList オプション
2	-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。	(2)　-ErrSup オプション

（凡例）

○：開発マネージャから指定できる

(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。

表中の参照先の番号（(1)，(2)，…）は「33.5.14　その他の設定」の番号（(1)，(2)，…）と対応しています。

項番	オプション	マネージャ	機能	参照先
1	-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）。	(1)　-Bin1Byte オプション
2	-JPN，{Alnum | V3JPN | V3JPNSpace} -noJPN	○	日本語項目の扱いを指定する。	(2)　-JPN オプション
3	-EquivRule，{NotExtend | NotAny | StdCode} -noEquivRule	○	文字の等価規則をどう変更するか指定する。	(3)　-EquivRule オプション
4	-UscoreStart -noUscoreStart	○	先頭が下線の CALL 定数を指定できるようにする。	(4)　-UscoreStart オプション

項番	オプション	マネージャ	機能	参照先
5	-BinExtend -noBinExtend	○	2 進データ項目に指定できる初期値を拡張する。	(5)　-BinExtend オプション
6	-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。	(6)　-MinusZero オプション
7	-TruncCheck -noTruncCheck	○	転記でのデータ切り捨てをチェックする。	(7)　-TruncCheck オプション
8	-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。	(8)　-LowerAsUpper オプション
9	-CBLVALUE -noCBLVALUE	○	環境変数 CBLVALUE を有効にする。	(9)　-CBLVALUE オプション
10	-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。	(10)　-Repository オプション
11	-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。	(11)　-RepositoryCheck オプション
12	-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。	(12)　-Define オプション
13	-Details -noDetails	○	コンパイラオプションの詳細情報を出力する。	(13)　-Details オプション
14	-OldForm "旧オプションの並び"	U	PC COBOL85 のオプションを指定する。	(14)　-OldForm オプション
15	-Help ｜-?	×	ccbl2002 コマンドのヘルプを出力する。	(15)　-Help オプション
16	-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。	(16)　-UniObjGen オプション
17	-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。	(17)　-UniEndian オプション
18	-MaxDigits38	○	数字項目，数字編集項目，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。	(18)　-MaxDigits38 オプション（Windows(x64) COBOL2002 で有効）

項番	オプション	マネージャ	機能	参照先
19	-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。	(19)　-IntResult,DecFloat40 オプション（Windows(x64) COBOL2002 で有効）
20	-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。	(20)　-LiteralExtend オプション
21	-SpaceAsZero -noSpaceAsZero	○	外部 10 進項目中に空白文字があるとき，ゼロとみなして比較，演算，転記を実行する。	(21)　-SpaceAsZero オプション
22	-CheckUninitData -noCheckUninitData	×	データ項目の初期化漏れをチェックする。	(22)　-CheckUninitData オプション
23	-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。	(23)　-FunctionECSup,CodeConvErr オプション

（凡例）

○：開発マネージャから指定できる

U：開発マネージャで使用する場合，［ユーザ設定］タブに指定する

×：開発マネージャから指定できない

(11)　XML 連携（開発マネージャを使用する場合）

開発マネージャのツリービューで XML データ定義ファイルを選択したときに指定できるオプションです。

このページで指定するオプションの詳細については，マニュアル「COBOL2002　XML 連携機能ガイド」を参照してください。

33.5.5　最終生成物の種類（プロジェクトの種類）の設定

最終生成物※の種類（開発マネージャでは，プロジェクトの種類）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，DLL，または標準ライブラリのことを示します。

(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（System 指定メインプログラム）

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。

なお，開発マネージャでは，プロジェクトの種類で「System 指定メインプログラム」を選び，「メインファイルを指定する」オプションにメインファイルを指定すると，-Main,System オプションが指定されます。

-Main,V3 ファイル名（V3 指定メインプログラム）

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。

なお，開発マネージャでは，プロジェクトの種類で「V3 指定メインプログラム」を選び，「メインファイルを指定する」オプションにメインファイルを指定すると，-Main,V3 オプションが指定されます。

(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（.obj）の指定がない。
  2. -SimMain オプションの指定がない。

- -Main オプションと-MainNotCBL オプションまたは-Dll 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.obj
                      主ﾌﾟﾛｸﾞﾗﾑ   副ﾌﾟﾛｸﾞﾗﾑ
```

- -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，引用符（ " ）で囲みます。

(2) メインプログラムなし

開発マネージャ上でプロジェクトをビルドする場合に，メインの COBOL ソースファイルを指定しないとき，このオプションを指定します。「メインプログラムなし」を指定したプロジェクトは，メインプログラムのオブジェクトファイルまたはライブラリを，プロジェクトに登録する必要があります。

このコンパイル方式を選んだ場合で，作成しようとしている COBOL プログラムが次の二つの条件を満たすときは，-MainNotCBL オプションを指定する必要があります。

- 最初に呼び出される COBOL プログラムより前に，他言語で記述されたプログラムが実行される場合
- 最初に呼び出される COBOL プログラムから，呼び出し元プログラムに戻る場合

このプロジェクトの種類で指定できるオプションは，-Main,System オプション（System 指定メインプログラム）または-Main,V3 オプション（V3 指定メインプログラム）を指定して実行可能ファイルを生成する場合に準じますが，次のオプションは指定できません。

「メインファイルを指定する」

このオプションは，開発マネージャだけで指定できます。

(3) -Dll オプション（最終生成物の種類がダイナミックリンクライブラリ）

(a) 形式

```
-Dll, {Stdcall | Cdecl}
-Dll
-noDll
```

(b) 機能

DLL 形式のオブジェクトファイルを出力します。-Dll オプションを指定しない場合は，実行可能ファイル（.exe）形式のオブジェクトファイルを出力します。

-Dll オプションを指定した場合，同時に指定した COBOL 原始プログラムは，すべて DLL 形式のオブジェクトファイルとなります。

なお，開発マネージャでは，最終生成物の種類に「ダイナミックリンクライブラリ」を選ぶと，-Dll オプションが指定されます。

-Dll,Stdcall（Dll の属性を stdcall にする）（Windows(x86) COBOL2002 で有効）

DLL の属性を stdcall にします。

なお，開発マネージャでは，「Dll の属性を stdcall にする」を選ぶと，-Dll,Stdcall オプションが指定されます。

-Dll,Cdecl（Dll の属性を cdecl にする）（Windows(x86) COBOL2002 で有効）

DLL の属性を cdecl にします。

なお，開発マネージャでは，「Dll の属性を cdecl にする」を選ぶと，-Dll,Cdecl オプションが指定されます。

-Dll（ Dll の属性を fastcall にする）（Windows(x64) COBOL2002 で有効）

DLL の属性を fastcall にします。

開発マネージャでは，最終生成物の種類に「ダイナミックリンクライブラリ」を選びます。

-noDll

-Dll オプションの指定を打ち消します。

(c) 注意事項

- -Dll オプションを指定すると，-Main,System，-Main,V3，および-SimSub オプションが無効となります。
- Windows(x64) COBOL2002 の場合，-Dll,Stdcall オプション，または-Dll,Cdecl オプションを指定しても，-Dll オプションを指定したとみなされ，DLL の属性は fastcall になります。

(4) 標準ライブラリ

COBOL ソースから標準ライブラリを作成します。

プロジェクトの作成時に，最終生成物の種類で「標準ライブラリ」を選んだ場合，この項目が指定されます。

このプロジェクトによって生成された標準ライブラリをリンクする実行可能プログラムが次の条件を満たす場合，-MainNotCBL オプションを指定する必要があります。

- 最初に呼び出される COBOL プログラムより前に，他言語で記述されたプログラムが実行される場合
- 最初に呼び出される COBOL プログラムから，呼び出し元プログラムに戻る場合

このプロジェクトの種類を選んだ場合，プロジェクトに指定できるオプションは-Main,System または-Main,V3 オプションを指定した場合と同じです。ただし，次のオプションは指定できません。

-SimMain，-SimSub，-SimIdent，-Lib，「リンク処理をしない」，「メインファイルを指定する」

このオプションは，開発マネージャだけで指定できます。

33.5.6 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

埋め込み SQL 文を使用できるようにします。

-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

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

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

- -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

(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

(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。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) -IsamExtend オプション（Windows(x86) COBOL2002 で有効）

(a) 形式

```
-IsamExtend [,Zone]
-noIsamExtend
```

(b)　機能

-IsamExtend

Btrieve（Pervasive.SQL）による索引編成ファイルを使用します。ISAM による索引編成ファイルを使用するときは，このオプションを指定しません。Btrieve（Pervasive.SQL）による索引編成ファイルについては，「6.10　Btrieve（Pervasive.SQL）による索引編成ファイル（Windows(x86) COBOL2002 で有効）」を参照してください。

-IsamExtend,Zone

Btrieve（Pervasive.SQL）による索引編成ファイル使用する場合，レコードキーに外部 10 進項目が指定されたときに，外部 10 進属性として処理します。

-noIsamExtend

-IsamExtend オプションの指定を打ち消します。

(5)　-XMAP オプション（Windows(x86) COBOL2002 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。

注

XMAP3 を使用しないで CHARACTER TYPE 句を使用した印刷制御付きのプリンタ出力をする場合，GDI モード印刷機能を使用します。

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。

(6) -OpenTP1 オプション

(a) 形式

```
-OpenTP1
-noOpenTP1
```

(b) 機能

-OpenTP1

OpenTP1 を使用したデータコミュニケーション機能を使用する場合に指定します。ただし，テストデバッガの DC シミュレーション機能で，データコミュニケーション機能を単体テストする場合は，このオプションを指定しないでください。

-noOpenTP1

-OpenTP1 オプションの指定を打ち消します。

33.5.7 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。

(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。

(5)　-DllInit オプション

(a)　形式

```
-DllInit
-noDllInit
```

(b)　機能

-DllInit

呼び出し時に DLL を初期状態にします。

-noDllInit

-DllInit オプションの指定を打ち消します。

(c)　注意事項

- -DllInit オプションを指定して，-Dll オプションを指定しなかった場合，-DllInit オプションは無視されます。この場合，-DllInit オプションが無効であることを示す警告メッセージが出力されます。
- -DllInit オプションを指定した場合に初期化される情報については，「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。

(6)　メインファイルを指定する

プロジェクトのメインプログラムにするソースファイルを指定します。

メインファイルに指定していたファイルをプロジェクトから削除した場合などは，この項目でメインファイルを指定します。

このオプションは，開発マネージャだけで指定できます。

33.5.8　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，「32.　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，「32. 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。
- Windows(x64) COBOL2002 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。

(2) -ScreenSpeed オプション

(a) 形式

```
-ScreenSpeed
-noScreenSpeed
```

(b) 機能

-ScreenSpeed

画面節（WINDOW SECTION または SCREEN SECTION）による画面機能を使用するとき，表示の見栄えや機能よりも表示速度を重視するためのオプションです。このオプションを指定すると，表示速度は速くなりますが，次の不都合が生じます。

- 文字間隔が狭くなる，縦けい線と文字が一部重なるなど，表示の見栄えが悪くなる。
- データ項目が画面上で重なりあっている場合の表示は保証しない。
- 縦倍文字，横倍文字に対するけい線の表示は保証しない。
- 画面節（WINDOW SECTION）の THROUGH 指定の LINE NUMBER 句，COLUMN NUMBER 句を指定すると，ERASE（ATTRIBUTE）文が正しく実行されない場合がある。
- 表示する項目が 1 行に収まらない場合の表示は保証しない。

-noScreenSpeed

-noScreenSpeed オプションの指定を打ち消します。

(c) 注意事項

- -ScreenSpeed オプションを指定してコンパイルした画面機能を含むプログラムと，-ScreenSpeed オプションを指定しないでコンパイルしたプログラムを一つの実行可能ファイル，DLL，または標準ライブラリ中に混在させないでください。

33.5.9 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，「37.2 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，「37.4 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，「37.6 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，「37.6 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。また，除算例外については，除算実行前に検出します。

-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 で指定したフォルダまたはカレントフォルダに入れておく。

また，このオプションを指定し，-OutputFile オプションで実行可能ファイル名を指定しない場合は，生成される主プログラムの実行可能ファイル名は NONAME1.exe となります。

Windows(x86) COBOL2002 の場合

DLL を呼び出す擬似主プログラムを作成する場合は，次のオプションを同時に指定してコンパイルしてください。

- DLL の属性が stdcall 呼び出し規約の場合：-Dll,Stdcall
- DLL の属性が cdecl 呼び出し規約の場合：-Dll,Cdecl

Windows(x64) COBOL2002 の場合

DLL を呼び出す擬似主プログラムを作成する場合は，-Dll オプションを同時に指定してコンパイルしてください。

このとき，生成される主プログラムの実行可能ファイル名は NONAME1.exe 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。
- -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 で指定したフォルダまたはカレントフォルダに入れておく。

-noSimSub

-SimSub オプションの指定を打ち消します。

(c) 注意事項

- このオプションは，-Dll オプションと同時に指定できません。また，呼び出し属性が stdcall となる副プログラムシミュレーションはできません。
- 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。
- -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 で指定したフォルダまたはカレントフォルダに入れておく。

注意事項

- 呼び出し属性が stdcall となる副プログラムシミュレーションはできません。
- 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

-noSimIdent

-SimIdent オプションの指定を打ち消します。

(c) 注意事項

- 呼び出し属性が stdcall となる副プログラムシミュレーションはできません。
- 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。
- -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，カレントフォルダ，または環境変数 CBLPIDIR で指定したフォルダに出力されます。

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

33.5.10　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)　-StdCall オプション（Windows(x86) COBOL2002 で有効）

(a)　形式

```
-StdCall
-noStdCall
```

(b)　機能

-StdCall

stdcall 呼び出し指示ファイル（.cbw）を読み込むためのオプションです。stdcall 呼び出し指示ファイルは，CALL 文で最外側のプログラムを呼び出すときの属性が stdcall 呼び出し規約となるように指示するファイルです。したがって，cdecl 呼び出し規約で呼び出す最外側のプログラム名を，stdcall 呼び出し指示ファイルに指定できません。

このオプションを指定しない場合，stdcall 呼び出し指示ファイルは，読み込まれません。また，呼び出し属性は，CALL-CONVENTION 段落の指定に従います。

-noStdCall

-StdCall オプションの指定を打ち消します。

(2)　-StdCallFile オプション（Windows(x86) COBOL2002 で有効）

(a)　形式

```
-StdCallFile ファイル名
-noStdCallFile
```

(b)　機能

-StdCallFile ファイル名

-StdCall オプションで読み込む stdcall 呼び出し指示ファイル（.cbw）の名称を指定します。

このオプションを指定した場合，stdcall 呼び出し指示ファイルに指定したファイル名だけが参照され，「COBOL ソースファイル名.cbw」は参照されません。このオプションを指定しないときは，「COBOL ソースファイル名.cbw」だけが参照されます。

-noStdCallFile

-StdCallFile オプションの指定を打ち消します。

(3) -Lib オプション

(a) 形式

```
-Lib, {GUI | CUI}
```

(b) 機能

-Lib,GUI

GUI モードのユーザプログラムを作成します。

-Lib 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。

-Lib,CUI

CUI モードのユーザプログラムを作成します。

(4) ライブラリの指定

インポートライブラリ，標準ライブラリを指定します。

このオプションは，開発マネージャだけで指定できます。

(5) リンケージを行わない

リンク処理をしない場合に指定します。

リンク処理をしない場合は，コンパイル処理によってオブジェクトファイルが生成されますが，リンク処理は行われません。

擬似主プログラム生成，擬似副プログラム生成を指定する場合は，この項目を指定しないでください。

このオプションは，開発マネージャだけで指定できます。

(6) -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（.obj），実行可能ファイル（.exe），およびDLL（.dll）の生成を抑止します。

-Compile,CheckOnly

COBOLプログラムのコンパイル時にエラーチェックだけします。オブジェクトファイル，実行可能ファイル，およびDLLは出力しません。

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，DLL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，またはDLL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。
- 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。開発マネージャでは，指定できません。

(7) -DefFileオプション

(a) 形式

```
-DefFile ファイル名
-noDefFile
```

(b) 機能

-DefFile ファイル名

COBOL2002が自動生成するモジュール定義ファイル（.def）の名称を指定します。

モジュール定義ファイルは，カレントフォルダに出力されるため，パス名を付けて指定してはいけません。パス名を付けて指定した場合，動作は保証しません。

このオプションを指定しない場合，生成されるモジュール定義ファイル名は，生成されるDLLファイルの名称に拡張子.defを付けた名称になります。

-noDefFile

-DefFile 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，コマンドライン（ccbl2002 コマンド）で指定できます。開発マネージャでは，指定できません。

(8) -IconFile オプション

(a) 形式

```
-IconFile ファイル名
-noIconFile
```

(b) 機能

-IconFile ファイル名

生成する実行可能ファイルのアイコンを変更します。

このオプションを指定すると，COBOL2002 が自動生成するリソース定義ファイル（.rc）中に指定するアイコンを，オプションで指定したアイコンファイルに変更します。ファイル名は，パス名を付けて指定することもできます。

このオプションを指定しないときは，COBOL2002 が用意しているアイコンが設定されます。

-noIconFile

-IconFile 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，コマンドライン（ccbl2002 コマンド）で指定できます。開発マネージャでは，指定できません。

(9) -ResrcFile オプション

(a) 形式

```
-ResrcFile ファイル名
-noResrcFile
```

(b) 機能

-ResrcFile ファイル名

COBOL2002 が自動生成するリソース定義ファイル（.rc）の名称を指定します。

リソース定義ファイルは，カレントフォルダに出力されるため，パス名を付けて指定してはいけません。パス名を付けて指定した場合，動作は保証しません。

このオプションを指定しない場合，生成されるモジュール定義ファイル名は，生成される実行可能ファイルの名称に拡張子.rc を付けた名称になります。

-noResrcFile

-ResrcFile 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，コマンドライン（ccbl2002 コマンド）で指定できます。開発マネージャでは，指定できません。

(10) -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，または DLL 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，または DLL ファイルの名称を指定します。カレントフォルダに出力する場合は，ファイル名だけを指定します。また，任意のフォルダに出力する場合は，ドライブ名からの絶対パス名で指定します。

このオプションの指定がない場合，次の規則に従って実行可能ファイル，または DLL ファイル名が決定します。

（実行可能ファイルを生成する場合）

-Main オプションに指定したファイル名に拡張子（.exe）を付けたファイル名になります。-Main オプションの指定がない場合，ccbl2002 コマンド中で-Main が仮定されたソースファイル名（ccbl2002 コマンド中で最初にコンパイルされたソースファイル名）に拡張子.exe を付けた名称となります。このとき，出力先のフォルダも上記のソースファイルと同じになります。

（DLL ファイルを生成する場合）

ccbl2002 コマンド中で最初にコンパイルされたソースファイル名称に拡張子.dll を付けた名称となります。このとき，出力先のフォルダも上記のソースファイルと同じになります。

なお，-SimMain オプション，-SimSub オプション，または-SimIdent オプションを指定した場合については，-SimMain オプション，-SimSub オプション，-SimIdent オプションの説明をそれぞれ参照してください。

-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。

(c) 注意事項

- ccbl2002 コマンドは，-OutputFile に指定したファイル名の拡張子が.exe または.dll であるかどうかをチェックしません。.exe または.dll 以外の拡張子を指定した場合，動作は保証しません。
- 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。開発マネージャでは，指定できません。

(11) -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。
- コマンドファイル中に-Link オプションの指定がある場合は，その行の-Link オプション以降がリンカに渡され，次の行から別のオプションとして扱われます。
- -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。
- このオプションを開発マネージャで指定する場合，プロジェクト設定ダイアログボックスの［ユーザ設定］タブを使用してください。［ユーザ設定］タブの詳細は，マニュアル「COBOL2002　操作ガイド」のオプションの設定方法の説明を参照してください。
- オプションの並びに-OUT リンカオプションを指定して，生成する実行可能ファイル，または DLL ファイルの名称を指定しないでください。生成されたファイルの動作は保証しません。

(12) -DynamicLink オプション

(a) 形式

```
-DynamicLink,Call
-noDynamicLink
```

(b) 機能

-DynamicLink,Call

定数指定の CALL 文によって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ときに，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。

ccbl コマンドでコンパイルする場合に，-Bd オプションを指定できます。

-noDynamicLink

-DynamicLink オプションの指定を打ち消します。

(13) -ManifestFileExt オプション

(a) 形式

```
-ManifestFileExt
-noManifestFileExt
```

(b) 機能

-ManifestFileExt

マニフェストファイルを実行可能ファイル，および DLL に埋め込まずに，外部マニフェストファイルとします。

COBOL2002 コンパイラは，LINK コマンド実行時に生成されたマニフェスト情報※を，MT コマンドを使用して実行可能ファイルや DLL に埋め込みますが，このオプションを使用するとマニフェスト情報を外部マニフェストファイルとして出力できます。

外部マニフェストファイルの出力先フォルダは，生成する実行可能ファイル，または DLL ファイルと同じになります。

マニフェスト情報の追加，または変更をしたい場合，生成した外部マニフェストファイルを直接テキストエディタで編集するか，または追加情報のマニフェストファイルを作成して MT コマンドの機能によってマニフェスト情報をマージします。

なお，MT コマンドがマニフェスト情報の埋め込みに失敗して，COBOL2002 コンパイラがエラーとなることがありますが，このオプションを使用すると，MT コマンドの実行を抑止してエラーを回避できます。

注※

COBOL2002 が同梱する LINK コマンドは，使用する C 実行時ライブラリの情報や UAC 情報がマニフェスト情報として生成されます。

なお，マニフェスト情報の種類や詳細については，Microsoft のリファレンスを参照してください。

-noManifestFileExt

-ManifestFileExt オプションの指定を打ち消します。

(c) 注意事項

- このオプションと-Compile オプションを同時に指定した場合，このオプションは無効となります。
- このオプションは，開発マネージャ上で出力したコンパイルリストの情報リストのオプションには表示しません。
- このオプションを指定した場合，外部マニフェストファイル名は「実行可能ファイル名.manifest」，および「DLL ファイル名.manifest」となります。そのため，ファイルの出力先フォルダに同名のファイルがあると上書きされます。
- 実行可能ファイルに対応する外部マニフェストファイルは，必ず実行可能ファイルと同じフォルダに配置してください。実行可能ファイルと同じフォルダに配置されない場合，使用する C 実行時ライブラリのバージョンによっては実行可能ファイルが正しく動作しません。
- DLL ファイルを作成するとき，このオプションを指定して生成されるマニフェストファイルは DLL ファイルに埋め込んで使用してください。マニフェストの埋め込みについては，「35.1.2 コンパイルとリンクを別々に実行する方法」の「(2) LINK コマンドによるリンク方法」にある「(b) マニフェストの埋め込み方法」を参照してください。

  また，使用する C 実行時ライブラリのバージョンによってはマニフェストファイルは生成されないことがあります。

  生成されない場合は，マニフェストファイルがなくても動作に問題ありません。

33.5.11 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　-CompatiM7　-CompatiV3　-H8Switch
  -Cblctr　-IgnoreLCC　-JPN　-CmDol　-Comp5　-NumAccept
  -V3Rec　-EquivRule,NotAny　-SQL　-BinExtend　-MaxDigits38
  -IntResult,DecFloat40　-LiteralExtend　-V3RecFCSpace　-SQLDisp
  -V3RecEased
- -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　-CompatiM7　-CompatiV3　-H8Switch　-Cblctr
  -IgnoreLCC　-JPN　-CmDol　-Comp5　-NumAccept　-V3Rec
  -EquivRule,NotAny　-SQL　-BinExtend　-MaxDigits38
  -IntResult,DecFloat40　-LiteralExtend　-V3RecFCSpace　-SQLDisp
  -V3RecEased
- 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 -CompatiM7 -CompatiV3 -H8Switch -Cblctr
  -IgnoreLCC -JPN -CmDol -Comp5 -NumAccept -V3Rec
  -EquivRule,NotAny -SQL -BinExtend -MaxDigits38
  -IntResult,DecFloat40 -LiteralExtend -V3RecFCSpace -SQLDisp
  -V3RecEased

(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 のどちらかを仮定する。どちらを仮定するかは WRITE 文の指定の内容による。	左記の場合，常に AFTER ADVANCING 1 を仮定する。	
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。

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
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。

33.5.12 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) -CompatiM7 オプション（Windows(x86) COBOL2002 で有効）

(a) 形式

```
-CompatiM7
-noCompatiM7
```

(b) 機能

-CompatiM7

MIOS7 COBOL85 との互換機能を有効にします。

-noCompatiM7

-CompatiM7 オプションの指定を打ち消します。

(2) -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 Windows 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，PC 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。
- Windows(x64) COBOL2002 の場合，-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，「(19)　-IntResult,DecFloat40 オプション（Windows(x64) COBOL2002 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。

(3) -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 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
- 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 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。

(4)　-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。

(5) -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。

(6) -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。
- 例外名 EC-SIZE-TRUNCATION に対する TURN 指令が有効の場合で，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のけた数より大きいときは，このオプションの指定があっても，上位けたを切り捨てないで，EC-SIZE-TRUNCATION 例外を検出します。

(7) -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。

(8) -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。

(9) -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。

(10) -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。

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 33–2 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～$2^{63}-1$

USAGE 句に COMP-5 を指定した項目に VALUE 句がある場合，指定できる値については，マニュアル「COBOL2002 言語 拡張仕様編」「25.3.2(7) USAGE COMP-5 を指定した項目に設定できる初期値（VALUE 句の値）の拡張」を参照してください。

-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。

(11) -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，COBOL2002 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 COBOL2002 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

  -CompatiM7　-NumAccept　-CmDol　-MaxDigits38　-IntResult,DecFloat40

  -LiteralExtend　-V3RecEased

- -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。

(12) -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，「33.3.1　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。

(13) -Switch オプション

(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 33-3 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

条件		ASCII	EBCDIC	EBCDIK
	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 33-2　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 33-3　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。

(14) -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，「(15)　-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。

(15)　-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。

(16) -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。

(17) -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。

(18) -BigEndian オプション

(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。

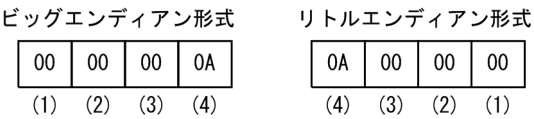
これは，マイクロプロセッサのアーキテクチャの違いによるもので，システムによって採用されるエンディアン形式が異なります。システムごとのエンディアン形式を次に示します。

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（OLE2 オートメーション機能で使用するサービスルーチンも含む）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。
- 開発マネージャでのビルド，リビルドや cblbuild2k コマンドを使用する場合，一つのプロジェクト中にこのオプションを指定したプログラムと指定しないプログラムを混在させないでください。
- 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。

(19) -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。

(20) -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。

(21) -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。

33.5.13　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 E　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 E.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。

33.5.14 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

(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
-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。

-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
- 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※2）

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

Windows(x64) COBOL2002 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（Windows(x64) COBOL2002 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※

Windows(x64) COBOL2002 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。

(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，「33.6.3 コンパイラ環境変数の詳細」の「(21) 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，「34.3.2 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，「34.2.3　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，「33.3.3　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，値全体をダブルコーテーション（"）で囲む必要があります。このとき，ダブルコーテーション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

(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出力します。

また，環境変数 CBLLINKER，環境変数 CBLMANIFESTTOOL，および環境変数 CBLRESOURCECOMPILER を指定した場合，環境変数に指定されたコマンドが起動するときに，コマンドのパス情報を出力します。

出力形式は，次のようになります。

形式

COBOL2002 コマンドの絶対パス名：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 インストールフォルダ¥bin¥ccbl2002: ccbl2002 -DebugInf -TDInf
-SrcList,CopyAll -Details -Optimize,1 -Lib,GUI -SimSub PROG1 -Main,System sample1.cbl
```

-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（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。

- -OldForm オプションに指定する旧オプションの並びの中に，さらにダブルコーテーション（"）がある場合，ダブルコーテーションを「¥"」に置き換えて指定してください。指定例を次に示します。

```
-OldForm ”-Fw ¥”c:¥temp¥sample.cbw¥” -Mw sample.cbl”
```

- このオプションを開発マネージャで指定する場合，プロジェクト設定ダイアログボックスの［ユーザ設定］タブを使用してください。［ユーザ設定］タブの詳細は，マニュアル「COBOL2002　操作ガイド」のオプションの設定方法の説明を参照してください。ただし，プロジェクト構成のコンパイル方法に影響する次のオプションを指定できません。指定した場合，動作は保証しません。

  -Mw　-Mh　-Cs　-Dl　-Vb　-Ad　-Ax　-Dc

(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。開発マネージャでは，指定できません。

(16)　-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　-V3RecEased

  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，-V3RecFCSpace，および-V3RecEased オプションは有効となります。

(17)　-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 UTF-16LE のコードに変換されます。

(18)　-MaxDigits38 オプション（Windows(x64) COBOL2002 で有効）

(a)　形式

```
-MaxDigits38
-noMaxDigits38
```

(b)　機能

-MaxDigits38

数字項目，数字編集項目，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，「29.　数字項目のけた拡張機能（Windows(x64)

COBOL2002 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，「29.　数字項目のけた拡張機能（Windows(x64) COBOL2002 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。

(19)　-IntResult,DecFloat40 オプション（Windows(x64) COBOL2002 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，「29.3　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。

- -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，「33.5.12　他システムとの移行の設定」の「(2)　-CompatiV3 オプション」を参照してください。

(20)　-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 33-4　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 33-5　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。

(21) -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。
- -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。

(22) -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，「33.7.3 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。

(23) -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 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 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，「30.7.3 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。

33.6　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。

33.6.1　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。環境変数はコンソール，または Windows システムの環境変数で，次の形式で設定します。

形式

```
環境変数=環境変数の値
```

注意事項

空白を含むフォルダ名やファイル名を環境変数の値に指定するときは，ダブルコーテーション（ " ）で囲まないでください。

(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，大文字と小文字は等価とみなします。

33.6.2　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 33–6　コンパイラ環境変数の一覧

項番	環境変数	マネージャ	設定内容
1	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
2	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
3	CBLCOPT	×	ccbl コマンドに指定する旧形式のオプション列
4	CBLCOPT2002	○	ccbl2002 コマンドに指定する新形式のオプション列
5	CBLERRMAX	○	コンパイルを打ち切る S レベルのエラーの数
6	CBLFIX	○	固定形式正書法の COBOL ソースファイルの拡張子

項番	環境変数	マネージャ	設定内容
7	CBLFIXEDFORMLINE	×	固定形式正書法の 1 行の長さの制限値
8	CBLFREE	○	自由形式正書法の COBOL ソースファイルの拡張子
9	CBLINITVALUE	○	初期化属性プログラムが呼ばれたときの VALUE 句の指定がないデータ項目の初期値（X'00'）
10	CBLLIB	○	登録集原文の検索フォルダ
11	CBLLINKER	○	コンパイラが使用するリンカの絶対パス名
12	CBLLINKINTERVAL	○	ccbl2002 コマンド，ccbl コマンドで実行可能ファイル，または DLL を作成する際に，対象ファイルへのアクセス処理の前に空ける間隔をミリ秒単位で指定する
13	CBLMANIFESTTOOL	○	コンパイラが使用するマニフェストツールの絶対パス名
14	CBLPIDIR	○	プログラム情報ファイル（.cbp）の生成先フォルダ
15	CBLREP	○	リポジトリファイルの出力先のフォルダ，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フォルダ
16	CBLRESOURCECOMPILER	○	コンパイラが使用するリソースコンパイラの絶対パス名
17	CBLSYSREP	○	リポジトリファイルの参照時に検索するフォルダ
18	CBLTAB	○	COBOL ソース中のタブ位置を設定する
19	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
20	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フォルダ
21	CBLVALUE	○	VALUE 句の指定のないデータ項目の初期値
22	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
23	登録集環境変数	○	登録集原文の検索フォルダ

（凡例）

○：開発マネージャから指定できる

×：開発マネージャから指定できない

33.6.3　コンパイラ環境変数の詳細

(1)　CBL_RDBSYS

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
```

(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
```

(3)　CBLCOPT

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。

各オプションは空白で区切って指定します。

（例）

```
CBLCOPT=-S1 -T4 -Ek
```

旧形式のオプションを開発マネージャで指定する場合，プロジェクト設定ダイアログボックスの［ユーザ設定］タブのコンパイラオプション欄で，-OldForm オプションを使用してください。［ユーザ設定］タブの詳細は，マニュアル「COBOL2002　操作ガイド」のオプションの設定方法の説明を参照してください。

(4) 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。

また，開発マネージャを使用する場合は，プロジェクト構成のコンパイル方法に影響する次のオプションを指定できません。指定した場合，動作は保証しません。

-Main　-MainNotCBL　-Dll

各オプションは空白で区切って指定します。

（例）

```
CBLCOPT2002=-StdVersion,1 -DebugData -Switch,EBCDIK
```

(5) 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
```

(6) 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，.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って指定します。

（例）

```
CBLFIX=.fix
```

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。

(7) 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
```

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。

(8) 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って指定します。

（例）

```
CBLFREE=.aaa .bbb .ccc
```

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

(9) 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
```

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，「(21)　CBLVALUE」を参照してください。

(10) CBLLIB

登録集原文を検索するフォルダを設定します。フォルダを複数指定する場合は，セミコロン（;）で区切って指定します。

（例）

```
CBLLIB=¥usr¥user¥copylib;c:¥source¥copy
```

なお，登録集環境変数が設定してある場合，登録集環境変数で設定したフォルダの方が環境変数 CBLLIB で指定したフォルダよりも優先します。フォルダによる検索順序を次に示します。

1. 登録集環境変数で設定したフォルダ
2. 環境変数 CBLLIB で設定したフォルダ
3. カレントフォルダ

開発マネージャを使用した場合の登録集原文の検索順序

開発マネージャからビルドを実行する場合，上記の 2.，3.については，開発マネージャによって次の検索順序が仮定されます。

1. プロジェクト設定ダイアログボックスで，ファイルに対して設定した環境変数 CBLLIB に指定されているフォルダ
2. プロジェクト設定ダイアログボックスで，プロジェクトに対して設定した環境変数 CBLLIB に指定されているフォルダ
3. プロジェクトの作業フォルダ
4. プロジェクトに登録されている登録集原文が格納されているすべてのフォルダ
5. プロジェクトに登録されている COBOL ソースファイルが格納されているすべてのフォルダ
6. システムに設定されている環境変数 CBLLIB に指定されているフォルダ

開発マネージャは上記 1.～6.のフォルダを仮定するため，1.の登録集環境変数で明示的に検索フォルダを設定してください。

(11) CBLLINKER

コンパイラが呼び出すリンカを変更したい場合に LINK コマンド（LINK.exe）の絶対パス名を指定します。

詳細は，「35.4 リンカパスの切り替え機能」を参照してください。

（例）

```
CBLLINKER=C:¥Program Files¥…¥BIN¥LINK.exe
```

注意事項

- LINK コマンドを相対パスで指定しないでください。指定した場合の動作は保証しません。
- この製品が同梱する LINK コマンドより古いバージョンの LINK コマンドを指定しないでください。古いバージョンの LINK.コマンドファイルのプロパティからファイルバージョンを参照してください。古い LINK コマンドを指定した場合の動作は保証しません。
- この環境変数を指定する場合，環境変数 PATH と LIB には，環境変数に指定した LINK コマンドが必要とするパスをこの製品のパスより優先して指定してください。指定しなかった場合の動作は保証しません。
- この環境変数を指定する場合，環境変数 CBLRESOURCECOMPILER および環境変数 CBLMANIFESTTOOL を同時に指定してください。同時に指定しなかった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 256 バイト以内でなければなりません。256 バイトを超える値を指定した場合，エラーメッセージを出力してコンパイルを中止します。このとき，生成対象のファイル（実行可能ファイルまたは DLL ファイル）が残っている場合は削除します。

- COBOL2002 コンパイラは LINK コマンド（LINK.exe）の起動に失敗した場合，エラーメッセージを出力し，コンパイルを中止します。このとき，生成対象のファイル（実行可能ファイルまたは DLL ファイル）が残っている場合は削除します。

(12) CBLLINKINTERVAL

ccbl2002 コマンドまたは ccbl コマンドで，実行可能ファイルまたは DLL を作成する際に，対象ファイルへのアクセス処理の前に空ける間隔をミリ秒単位で指定します。指定できる範囲は，1～60,000 であり，省略時および範囲外の値の場合は，この環境変数の指定を無効とします。

なお，環境変数 CBLLINKINTERVAL の指定が有効な場合は，対象ファイルが占有可能な状態か合わせてチェックします。

（例）対象ファイルへのアクセス間隔を 200 ミリ秒とする場合

```
CBLLINKINTERVAL=200
```

ccbl2002 コマンドまたは ccbl コマンドから呼び出す LINK コマンドや MT コマンドは，出力ファイルが OS やウィルスチェッカなどから，一時的に共有されることが原因でエラーになることがあります。

この場合，時間をおいて再コンパイルしてください。時間をおいても問題が解決しないときは，環境変数 CBLLINKINTERVAL で対象ファイルへのアクセス間隔を変更した状態で再コンパイルすることで対処してください。

注意事項

- 環境変数 CBLLINKINTERVAL で指定する値は，開発環境で使用されるディスク装置，マシン環境，およびシステムの仕様に依存するため，実際の試行（検証）によって最適な値を使用してください。
- 環境変数 CBLLINKINTERVAL の指定は，実行可能ファイルまたは DLL を作成する際のファイルアクセスごとに有効となります。また，ファイルアクセスごとに，ファイルの占有ができるかを最大で 5 回チェックします。このため，環境変数 CBLLINKINTERVAL を指定すると，指定した値以上にコンパイル時間が遅くなります。
- 環境変数 CBLLINKINTERVAL が有効な場合，開発マネージャでのビルド，クイックビルド，リビルドの際にもそれぞれの処理時間が遅くなります。また，それぞれの処理でのリンクの実行中は，リンクが完了するまでビルドの中止がすぐにできないため，環境変数 CBLLINKINTERVAL を指定する場合は注意が必要です。

(13) CBLMANIFESTTOOL

コンパイラが呼び出すマニフェストツールを変更したい場合に MT コマンド（MT.exe）の絶対パス名を指定します。

詳細は，「35.4　リンカパスの切り替え機能」を参照してください。

（例）

```
CBLMANIFESTTOOL=C:¥Program Files¥…¥bin¥MT.exe
```

注意事項

- MT コマンドを相対パスで指定しないでください。指定した場合の動作は保証しません。
- この環境変数を使用する場合，環境変数 PATH には環境変数に指定した MT コマンドが必要とするパスを Windows SDK の bin へのパスより優先して指定してください。指定しなかった場合の動作は保証しません。
- この環境変数を使用する場合，環境変数 CBLLINKER および環境変数 CBLRESOURCECOMPILER を同時に指定してください。同時に指定しなかった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 256 バイト以内でなければなりません。256 バイトを超える値を指定した場合，エラーメッセージを出力してコンパイルを中止します。このとき，生成対象のファイル（実行可能ファイルまたは DLL ファイル）が残っている場合は削除します。
- COBOL2002 コンパイラは MT コマンドの起動に失敗した場合，エラーメッセージを出力し，コンパイルを中止します。このとき，生成対象のファイル（実行可能ファイルまたは DLL ファイル）が残っている場合は削除します。

(14) CBLPIDIR

プログラム情報ファイル（.cbp）を任意のフォルダ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。設定できるフォルダは一つだけです。

（例）

```
CBLPIDIR=c:¥temp
```

(15) CBLREP

リポジトリファイル（.rep）を任意のフォルダに生成，または更新したい場合に，出力先のフォルダを指定します。指定されたフォルダは，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，「34. 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

- 指定するフォルダが複数ある場合は，各フォルダをセミコロン（;）で区切って指定します。
- 指定したフォルダ群に同じ名称のリポジトリファイルが複数ある場合，先に指定したフォルダ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，最初に指定されたフォルダ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，カレントフォルダが検索されます。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，カレントフォルダが対象となります。

（例）

フォルダとして¥usr¥user¥replib を指定します。

```
CBLREP=¥usr¥user¥replib
```

(16) CBLRESOURCECOMPILER

コンパイラが呼び出すリソースコンパイラを変更したい場合に RC コマンド（RC.exe）の絶対パス名を指定します。

詳細は，「35.4 リンカパスの切り替え機能」を参照してください。

（例）

```
CBLRESOURCECOMPILER=C:¥Program Files¥…¥bin¥RC.exe
```

注意事項

- RC コマンドを相対パスで指定しないでください。指定した場合の動作は保証しません。
- この環境変数を指定する場合，環境変数 PATH には環境変数に指定した RC コマンドが必要とするパスを Windows SDK の bin へのパスより優先して指定してください。指定しなかった場合の動作は保証しません。
- この環境変数を指定する場合，環境変数 CBLLINKER および環境変数 CBLMANIFESTTOOL を同時に指定してください。同時に指定しなかった場合の動作は保証しません。
- 環境変数に指定する値の長さは 256 バイト以内でなければなりません。256 バイトを超える値を指定した場合，エラーメッセージを出力してコンパイルを中止します。このとき，生成対象のファイル（リソースファイル）が残っている場合は削除します。
- COBOL2002 コンパイラは RC コマンドの起動に失敗した場合，エラーメッセージを出力し，コンパイルを中止します。このとき，生成対象のファイル（リソースファイル）が残っている場合は削除します。

(17) 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フォルダを指定します。

環境変数 CBLSYSREP に指定したフォルダ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（DLL 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フォルダを指定する場合に使用します。

詳細は，「34.　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

- 指定するフォルダが複数ある場合は，各フォルダをセミコロン（;）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，カレントフォルダが優先されます。

（例）

フォルダとして c:¥usr¥user¥sysreplib と d:¥users¥lib¥rep を指定します。

```
CBLSYSREP=c:¥usr¥user¥sysreplib:d:¥users¥lib¥rep
```

(18)　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
```

(19)　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，「33.7.3　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
```

(20)　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フォルダに出力したい場合に，その出力先フォルダを 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フォルダには，カレントフォルダが仮定されます。

注意事項

- 出力先フォルダ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=C:¥Work
```

(21) 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。

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
```

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，「(9)　CBLINITVALUE」を参照してください。

(22)　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
```

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。

(23)　登録集環境変数

登録集原文を検索するフォルダを設定します。フォルダを複数指定する場合は，セミコロン（;）で区切って指定します。

（例）

```
登録集名=¥usr¥user¥copylib
```

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。

ここで指定したフォルダは，環境変数 CBLLIB で指定したフォルダよりも優先します。詳細については，「33.3.1　原始文操作機能」を参照してください。

33.7　コンパイラ付属機能

33.7.1　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　操作ガイド」を参照してください。

33.7.2　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，「33.4.1　ccbl2002 コマンド」，および「33.4.2　ccbl コマンド」を参照してください。

33.7.3　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 33-4　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，「33.5.10　リンクの設定」の「(6)　-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，「33.5.14　その他の設定」の「(22)　-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。

- 開発マネージャでビルドを実行した場合
- 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，「33.6.3　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，「33.6.3　コンパイラ環境変数の詳細」の「(19)　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 に出力先フォルダを指定します。環境変数 CBLUNINITDATA_OUTRESULTLISTDIR の詳細は，「33.6.3　コンパイラ環境変数の詳細」の「(20)　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シフト JIS 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フォルダ名です。
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，登録集原文があるフォルダ名です。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","C:¥WORK","Prog1.CBL","MAIN",,,"001100","11","20","A","0"
"2","C:¥WORK","Prog1.CBL","MAIN","C:¥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","C:¥WORK","Prog1.CBL","MAIN",,,"16","16","20","A","1"
"2","C:¥WORK","Prog1.CBL","MAIN",,,"19","19","20","B","1"
```

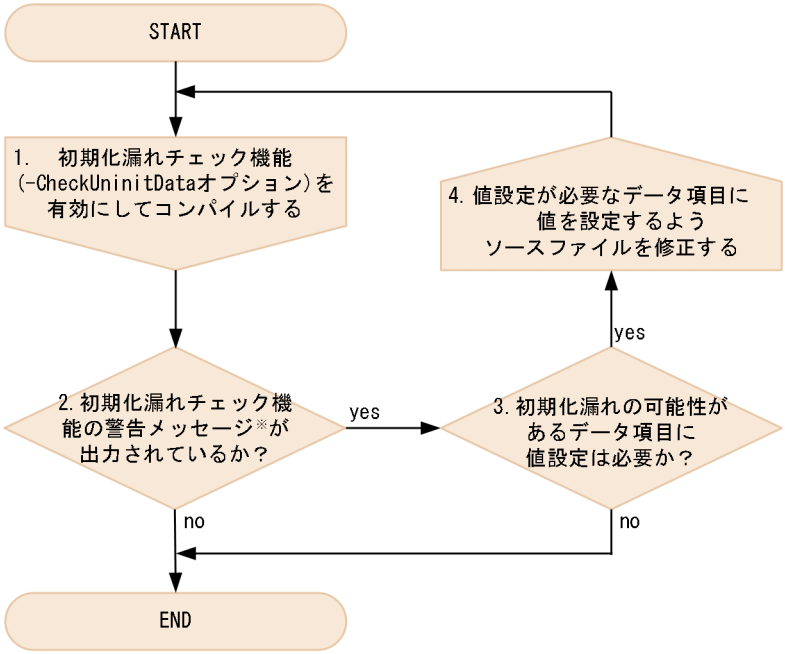
初期化漏れ従属項目一覧

```
"#","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 33-5　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
- COBOL エディタで構文チェックする
- COBOL ソース解析を使用する

(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。

   注意事項

   開発マネージャのビルドを使用する場合は，-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となります。開発マネージャで初期化漏れチェックをする場合は，COBOL2002 Developer Professional の COBOL ソース解析の機能を使用してください。

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。

(b) COBOL エディタで構文チェックする方法

COBOL エディタの構文チェック結果ウィンドウから初期化漏れを調査する手順を次に示します。COBOL エディタの操作方法については，マニュアル「COBOL2002　操作ガイド」を参照してください。

1. 構文チェック オプションダイアログボックスのコンパイラの設定のオプションに「-CheckUninitData」を指定して，構文チェックを実行する。
2. 構文チェック結果ウィンドウ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，メッセージをダブルクリックして該当行へジャンプし，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。

(c) COBOL ソース解析を使用する方法

COBOL ソース解析の解析ログから初期化漏れを調査する手順を次に示します。COBOL ソース解析の操作方法については，マニュアル「COBOL2002 Professional Tool Kit COBOL ソース解析ガイド」を参照してください。

1. -CheckUninitData コンパイラオプションを指定して COBOL ソース解析を実行する。
2. 解析ログを参照して，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，メッセージをダブルクリックして該当行へジャンプし，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 • 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

項番	機能
2	利用者定義関数
3	OLE2 オートメーション機能
4	連絡節で定義されているデータ項目
5	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
6	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
7	制御が渡らない文，手続き
8	覚え書きとみなされた構文
9	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
10	宣言部分の文，手続き（USE 文の宣言手続き）
11	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）
12	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
13	可変部分に定義されたデータ項目
14	動的長基本項目
15	報告書作成機能
16	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
17	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
18	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
19	RENAMES 句に THRU 指定がある再命名項目※3
20	合成キー
21	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
```

34 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。

34.1　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

34.1.1　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

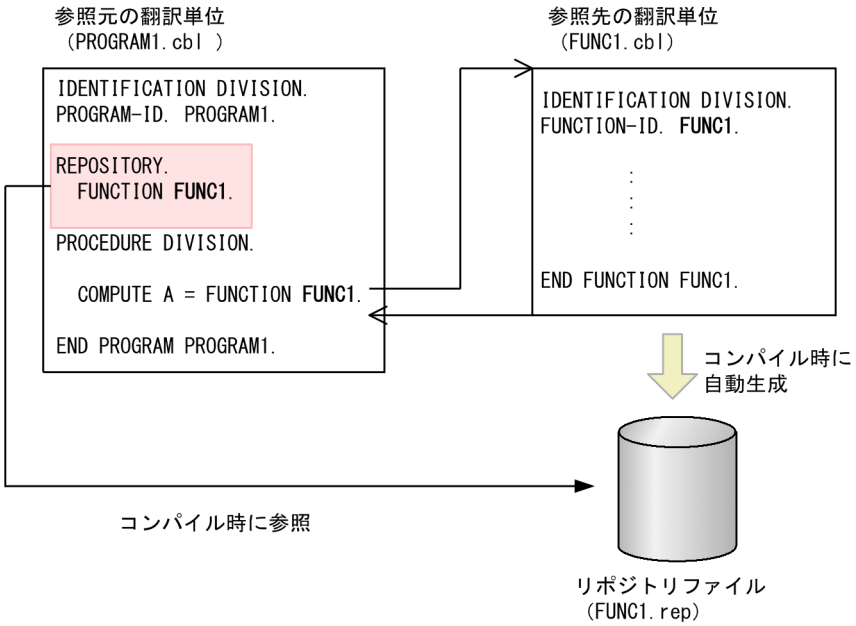
  参照元の翻訳単位から参照されるときに必要な定義情報を，参照元の翻訳単位のコンパイルより前に出力する必要があります。この定義情報を，外部リポジトリと呼びます。外部リポジトリの言語仕様については，マニュアル「COBOL2002　言語　標準仕様編」「4.12　外部リポジトリ」を参照してください。

  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。

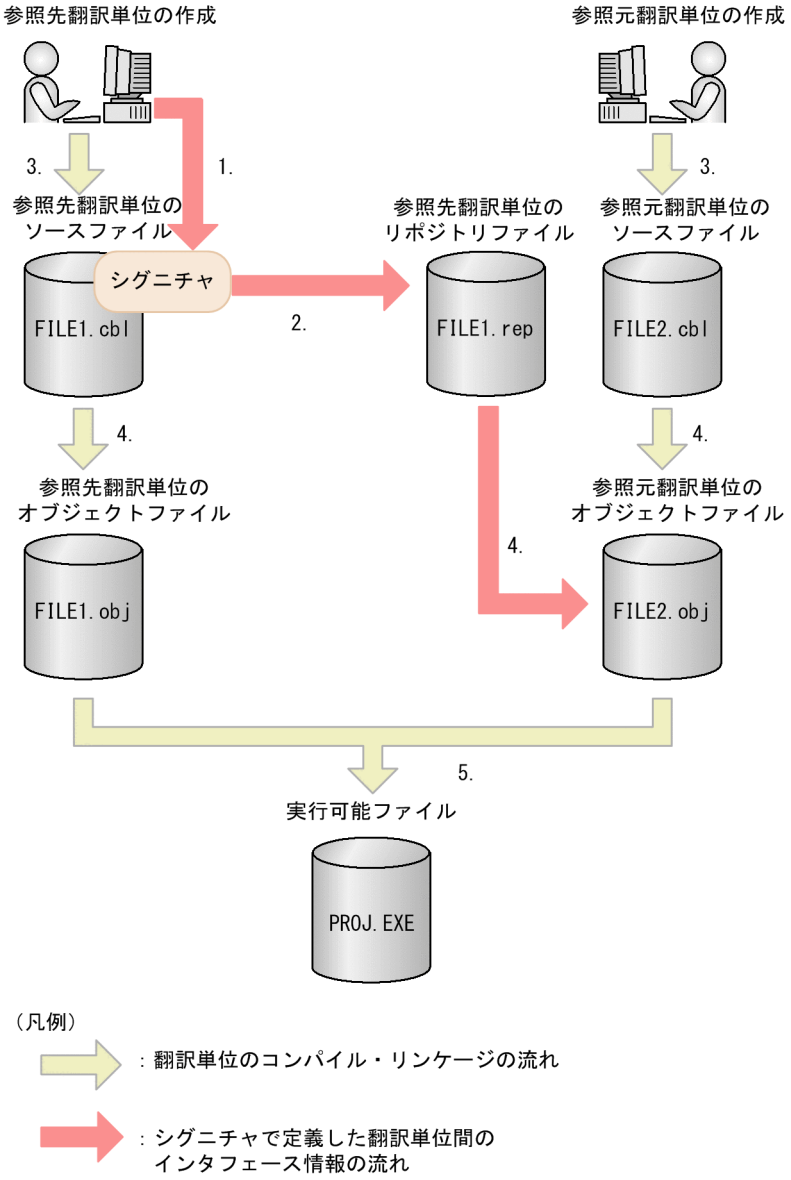
- プログラム定義だけで構成されているCOBOLプログラムの場合，リポジトリファイルは出力されません。また，リポジトリ段落に参照するプログラム名を指定する必要もありません。詳細は，「34.3.3　プログラム定義だけのコンパイル」を参照してください。
- 参照元の翻訳単位をコンパイルする時点で，参照先の翻訳単位のリポジトリファイルが出力されていない場合，参照関係が解決できないためコンパイルエラーとなります。そのため，異なるソースファイルに存在する翻訳単位同士が参照関係を持つ場合は，ソースファイルをコンパイルする順序を意識する必要があります。

  ただし，開発マネージャを使用しているときは，構成するソースファイルをすべてプロジェクトに指定しておけば，開発マネージャがコンパイルする順番を自動的に決定するため，翻訳単位の参照関係によってソースファイルのコンパイル順を意識する必要はありません。

34.1.2　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，「34.3.2　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フォルダに，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フォルダについては，「34.2.4　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，「34.1.3　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 DLL 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 DLL を生成します。

34.1.3　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。

34.2　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。

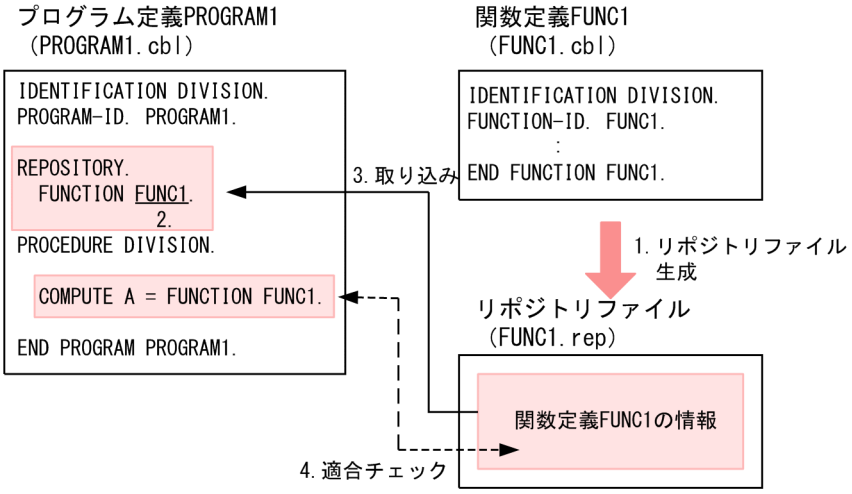
リポジトリファイルは，ソースファイルと 1 対 1 の関係となっています。そのため，ソースファイルをコンパイルした場合，一つのソースファイルに対して一つのリポジトリファイルが生成されます。生成されるリポジトリファイルの名称は，ソースファイルの名称と同じで，拡張子が.rep となります。

34.2.1　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。

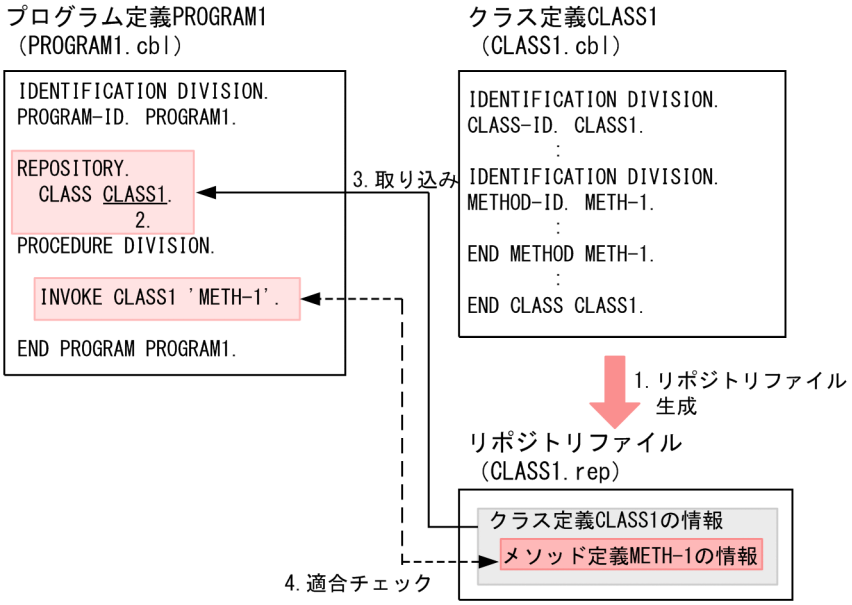


1. 関数定義 FUNC1 が含まれるソースファイル FUNC1.cbl をコンパイルして，リポジトリファイル FUNC1.rep を生成します。
2. 関数定義 FUNC1 を参照するプログラム定義 PROGRAM1 で，環境部構成節のリポジトリ段落に，参照する関数定義 FUNC1 を指定します。
3. プログラム定義 PROGRAM1 をコンパイルすると，リポジトリ段落の指定を基に，関数定義 FUNC1 が含まれるリポジトリファイル FUNC1.rep が検索して取り込まれます。リポジトリファイルの検索規則については，「34.2.4　リポジトリファイルの参照方法」の「(2)　リポジトリファイルの検索」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，「34.2.4　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。

34.2.2　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フォルダの規則について，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フォルダに生成されます。環境変数 CBLREP に複数のフォルダを指定している場合は，最初に指定したフォルダに生成されます。
- 環境変数 CBLREP を設定していない場合は，カレントフォルダに生成されます。

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。

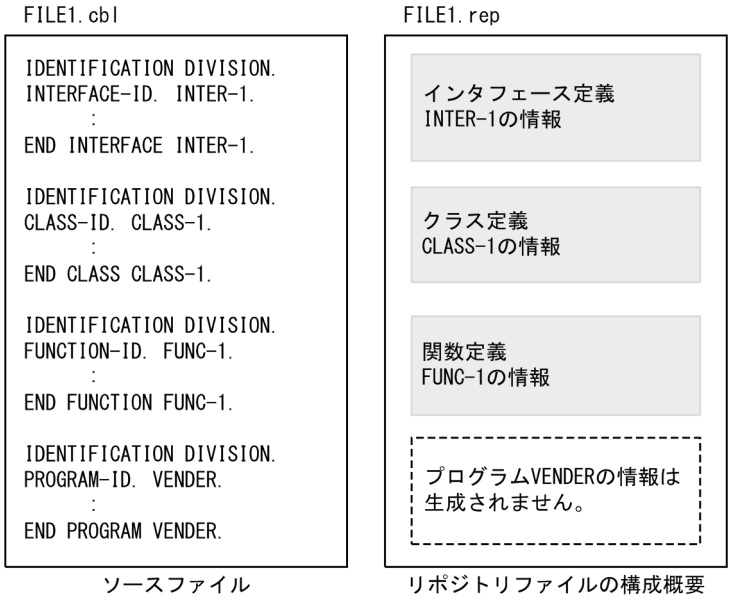
(1) ソースファイルとリポジトリファイルの関係

（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 34-1 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）

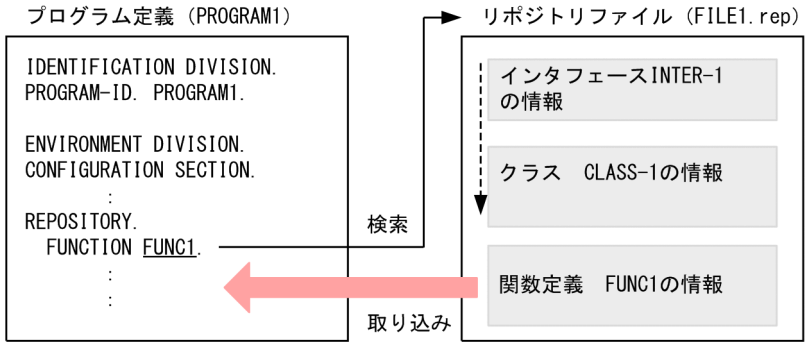
プログラム定義が関数定義を呼び出す場合の，リポジトリ段落の指定例について，次に示します。

```
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，「34.2.4　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 34-2　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。

34.2.3　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，「34.3.2　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 に指定したフォルダ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フォルダに，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 に指定したフォルダが存在しない場合

1. カレントフォルダ※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. カレントフォルダ※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，カレントフォルダ※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フォルダは，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フォルダ
- 開発マネージャを使用している場合
  プロジェクトの作業フォルダ

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，「34.5　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，「34.5　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。

34.2.4　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

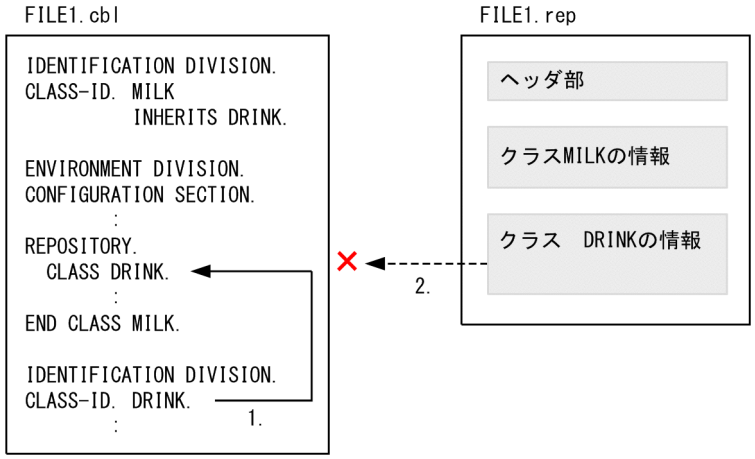
リポジトリ段落に指定したプロパティ名の定義情報は，同じリポジトリ段落に指定したクラス定義やインタフェース定義に含まれるため，個別の定義情報は検索されません。

(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 34-3　同じソースファイル中の翻訳単位名の検索の例



（凡例）

——►：同じソースファイル中に存在する翻訳単位名の検索

-----►：リポジトリファイルの検索

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 に指定したフォルダ
2. カレントフォルダ※
3. 環境変数 CBLSYSREP に指定したフォルダ
4. インストールフォルダ下の rep フォルダ

注※

この場合のカレントフォルダは，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フォルダ

- 開発マネージャを使用している場合
  プロジェクトの作業フォルダ

環境変数 CBLREP には，リポジトリファイルの検索フォルダをカレントフォルダ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，DLL 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フォルダを設定します。これらの環境変数の詳細については，「33.6　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，「インストールフォルダ¥rep」のフォルダ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，同じフォルダ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。

検索したフォルダ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。

34.3　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。

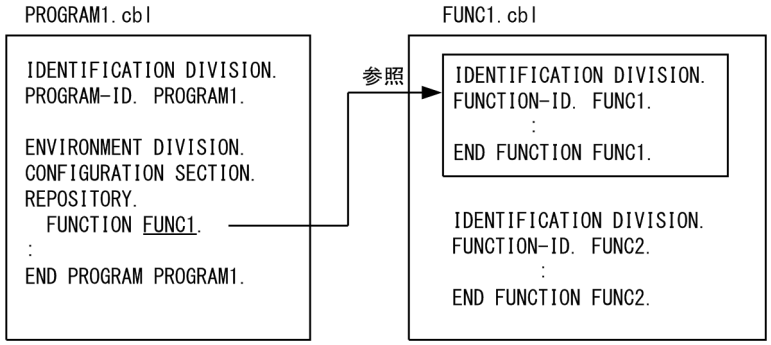
リポジトリ段落で参照するクラス定義，インタフェース定義，または関数定義のような翻訳単位を含むソースファイルを先にコンパイルしておく必要があります。ただし，開発マネージャを使用しているときは，構成するソースファイルをすべてプロジェクトに指定しておけば，開発マネージャがコンパイルする順番を自動的に決定するため，翻訳単位の参照関係によってソースファイルのコンパイル順を意識する必要はありません。

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。

34.3.1　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，「34.3.2　リポジトリファイルの単独生成」を参照してください。

34.3.2　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，「33.　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

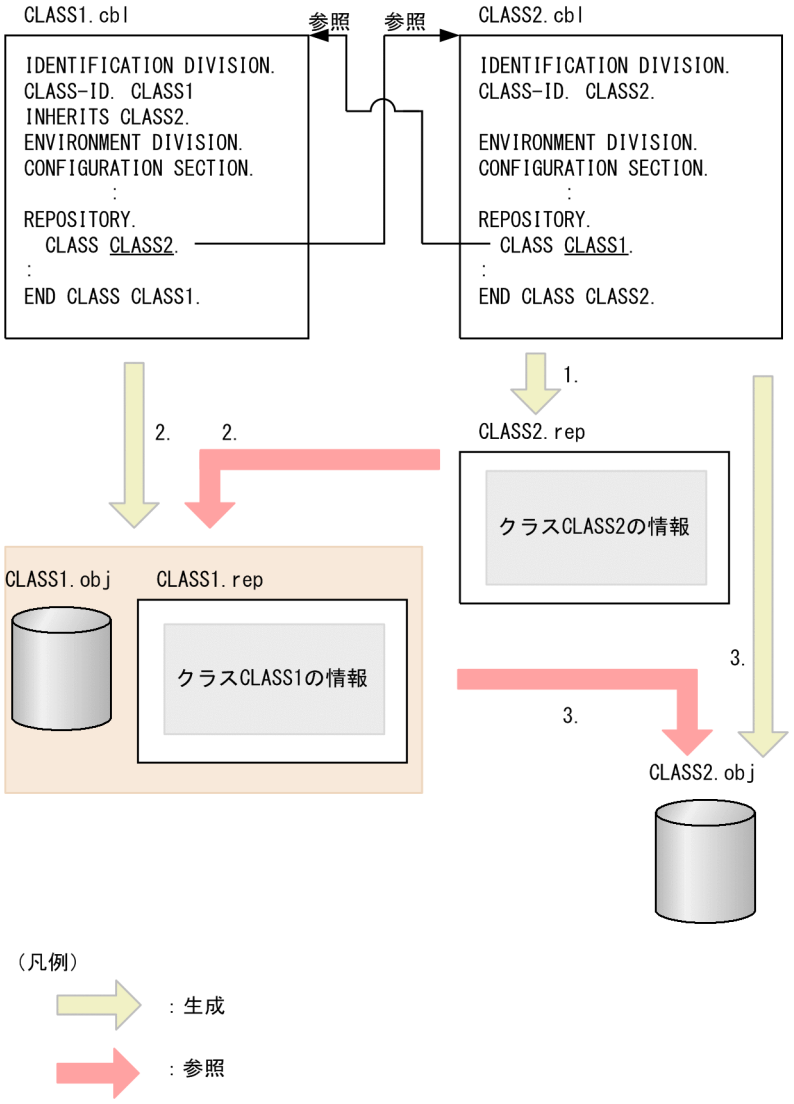
   「ccbl2002 -Repository,Gen CLASS2.cbl」と指定すると，リポジトリファイル CLASS2.rep が作成されます。

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 DLL を作成できません。

34.3.3　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。

34.4.1　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，「34.2.4　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 E　コンパイルリスト」を参照してください。

図 34–4　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，「34.2.4 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フォルダ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 E　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 34-5　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 34-6　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  C:¥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（C:¥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（C:¥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，「34.4.2 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 34-7　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 E　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 34-8 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 34-9 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  C:¥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（C:¥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（C:¥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，「34.4.2　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 34-10　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.
                                :
```

34.4.2　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，「34.4.1　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 34-1　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す。

(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 34–12　-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 34-11　-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

```
*****************************************************************

   リポジトリファイル情報リスト

       ファイル名   : orange.rep                                    }…… 1.
        生成時刻    : YYYY-MM-DD HH:MM:SS                          }
       コンパイラ   : COBOL2002 (X)  VV-RR                         }

*****************************************************************

- << クラス情報 >> ------------------------------------------

     クラス名             : ORANGE
     生成時刻             : YYYY-MM-DD HH:MM:SS                    }…… 2.

     継承クラス・インタフェース情報 :                              }
                                                                  }…… 3.
       DRINK (CLASS)                                              }

     参照翻訳単位情報  :                                          }
                                                                  }…… 4.
       CONTAINER ( INTERFACE )                                    }

  ---  ファクトリオブジェクト  ---

     [ プロパティ情報 ]                                           }
                                                                  }
       プロパティ名         種別       属性情報                  }…… 5.
     ------------------------------------------------------       }
       VITAMIN_C          (参照)  PIC  S9(9)  USAGE COMP          }
       VITAMIN_C          (更新)  PIC  S9(9)  USAGE COMP          }

     [ メソッド情報 ]                                             }
                                                                  }…… 6.
       None                                                       }

  --- インスタンスオブジェクト ---

     [ プロパティ情報 ]                                           }
                                                                  }…… 5.
       None                                                       }

     [ メソッド情報 ]                                             }
                                                                  }
       メソッド名          引数／返却項目の情報                    }
     ------------------------------------------------------       }…… 6.
      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 N.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 34-12　-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

```
*********************************************************************

   リポジトリファイル情報リスト

       ファイル名   : func.rep                                          }…… 1.
        生成時刻    : YYYY-MM-DD HH:MM:SS                               }
       コンパイラ   : COBOL2002 (X) VV-RR                               }

*********************************************************************

- << 関数情報 >> ------------------------------------------------

     関数名              : FUNCTION2
     生成時刻            : YYYY-MM-DD HH:MM:SS

     参照翻訳単位情報 :                                                }…… 2.
                                                                       }
       FUNCTION1 (FUNCTION)                                            }

     [ 関数情報 ]                                                      }
                                                                       }
       関　数　名          引数／返却項目の情報                        }…… 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 N.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 34-2　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

ファイル‘******’はオープンできません。

0002

ファイル‘******’の読み込みでI/Oエラーが発生しました。

0003

ファイル‘******’の書き込みでI/Oエラーが発生しました。

(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. カレントフォルダがネットワークフォルダのとき，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

ファイル‘******’はリポジトリファイルではありません。

0005

リポジトリファイル‘******’は，対象マシンが異なるので参照できません。

0006

リポジトリファイル‘******’は壊れています。

0007

‘******’という名前の翻訳単位はリポジトリに存在しません。

0008

‘******’という名前の翻訳単位が複数のリポジトリに存在します。

0009

論理番号 ‘******’ で論理エラーが発生しました。

(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

メモリ不足で続行できません。

(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

不正なオプション ‘******’ が指定されています。

0012

オプション ‘******’ にパラメタがありません。

0013

オプション ‘******’ が長すぎます。

0014

このバージョンの cbl2krep ではリポジトリファイル ‘******’ はアクセスできません。

0017

‘******’ という名前のクラスはリポジトリファイルに存在しません。

0018

‘******’ という名前のインタフェースはリポジトリファイルに存在しません。

0019

オプション ‘******’ にパラメタは指定できません。

0020

リポジトリファイルが指定されていません。

0023

他のプログラムでシステムの制限までファイルをオープンしているので，cbl2krep でファイルをオープンすることができません。

(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

ファイル ‘******’ は存在しません。

34.5　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，「34.3.2　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

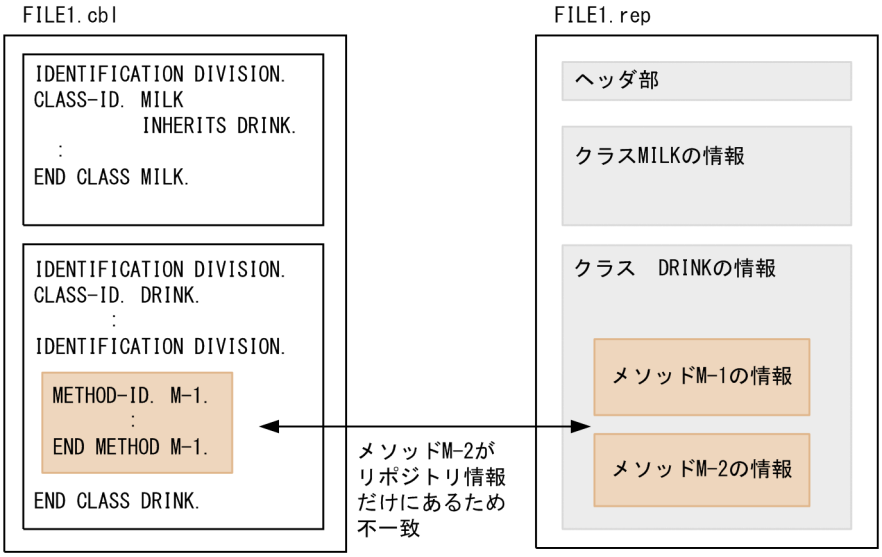
-Repository,Sup オプションを指定した場合，コンパイル時にリポジトリファイルが更新されません。ただし，リポジトリファイルが存在しない場合は，-Repository,Sup オプションの指定に関係なく，リポジトリファイルが新規に生成されます。

-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

35 実行可能ファイルと DLL 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 DLL を作成する方法について説明します。

35.1　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 コマンドを使って，COBOL プログラムから実行可能ファイルを作成したり，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。

35.1.1　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，「33.　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

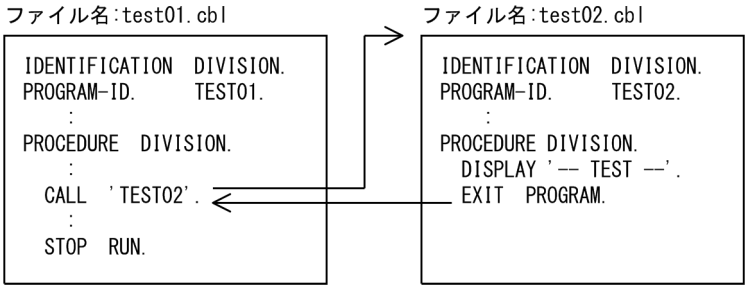
COBOL プログラム"test01.cbl"に-Main,System オプションを指定し，実行可能ファイル名称に"test02.exe"を指定します。

```
ccbl2002 -Main,System test01.cbl -OutputFile test02.exe
```

上記のコマンドを実行すると，実行可能ファイル"test02.exe"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

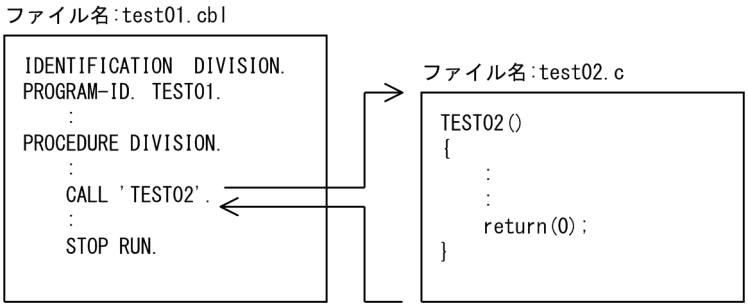
主プログラムである"test01.cbl"に-Main,System オプションを指定し，副プログラムである"test02.cbl"には何も指定しないで ccbl2002 コマンドを実行します。

```
ccbl2002 -Main,System test01.cbl test02.cbl -OutputFile test03.exe
```

上記のコマンドを実行すると，実行可能ファイル"test03.exe"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作成する場合，C プログラムをあらかじめ C コンパイラでコンパイルし，オブジェクトファイルを生成しておく必要があります。



ccbl2002 コマンドの指定

COBOL プログラム"test01.cbl"と，あらかじめ作成しておいた C プログラムのオブジェクトファイル"test02.obj"を指定します。

```
ccbl2002 -Main,System test01.cbl test02.obj -OutputFile test03.exe
```

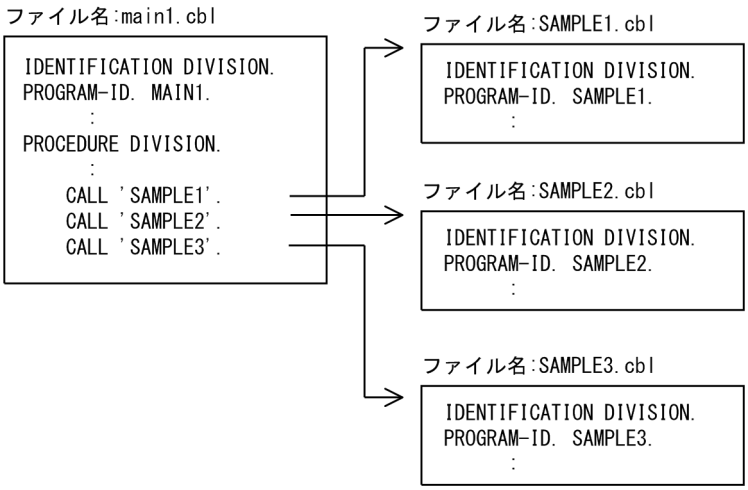
(4) COBOL プログラムと標準ライブラリをリンクして実行可能ファイルを生成する

標準ライブラリとは，オブジェクトファイルから構成され，ライブラリ管理ツール LIB によって作成されたライブラリです。標準ライブラリを使用すると，複数のオブジェクトファイルを一つにまとめて管理できます。

ライブラリ管理ツール LIB については「38.2　ライブラリ管理ツール」を参照してください。

ccbl2002 コマンドを使って，オブジェクトファイルのリンクと同様に標準ライブラリをリンクできます。

標準ライブラリを使用した実行可能プログラムを作成する例を次に示します。



この場合，ccbl2002 コマンドで標準ライブラリに取り込みたい COBOL プログラムのオブジェクトファイル（.obj）を生成しておき，lib コマンドで標準ライブラリとしてリンクします。そのあと，ccbl2002 コマンドで実行可能ファイル（main1.exe）を生成します。

```
ccbl2002 SAMPLE1.cbl -Compile,NoLink
ccbl2002 SAMPLE2.cbl -Compile,NoLink
ccbl2002 SAMPLE3.cbl -Compile,NoLink
lib SAMPLE1.obj SAMPLE2.obj SAMPLE3.obj /OUT:SAMPLE.lib
ccbl2002 -Main,System MAIN1.cbl SAMPLE.lib -OutputFile main1.exe
```

なお，開発マネージャを使用して標準ライブラリをリンクすることもできます。この場合は，次の手順で実行可能ファイルを作成してください。

1. 開発マネージャの標準ライブラリのプロジェクトを作成する。
2. 1.で作成したプロジェクトをビルドする。
3. 開発マネージャで実行可能ファイルのプロジェクトを作成する。
4. 2.で生成された標準ライブラリをオプションの「ライブラリの指定」または「ソースファイルの追加」でプロジェクトに追加する。
5. 4 で作成したプロジェクトをビルドする。
   標準ライブラリをリンクした実行可能ファイルが作成されます。

なお，開発マネージャの操作方法については，マニュアル「COBOL2002　操作ガイド」を参照してください。

使用する機能によって必要となるライブラリ

標準ライブラリを構成するソースが次の機能を使用している場合，標準ライブラリをリンクしている実行可能プログラムのリンク時に，関連するライブラリを指定する必要があります。ただし，開発マネージャで作成した標準ライブラリをリンクする場合は，これらのライブラリを指定する必要はありません。

機能	ライブラリ
OLE2 オートメーション機能を使用していて，-TDInf オプションを指定した場合	cbl2kgul.lib
整列併合機能を使用する場合	libmsort.lib（Windows(x86) COBOL2002 の場合） libnsort64.lib（Windows(x64) COBOL2002 の場合）
COBOL85 Version 4.0 以前の整列併合機能を使っているオブジェクトファイルがある場合	libnsort.lib※
通信節での画面データの送受信・帳票データの送信機能を使用する場合（XMAP3）	x3mwdr32.lib（Windows(x86) COBOL2002 の場合） x3mwdr64.lib（Windows(x64) COBOL2002 の場合）
-XMAP,LinePrint（XMAP3 での書式印刷機能）オプションを指定した場合	x3klib32.lib※
-CompatiM7（MIOS7 COBOL85 互換）オプションを指定した場合	jcm7lib.lib※
データコミュニケーション機能を使用していて，-OpenTP1 オプションを指定した場合	libmcf.lib

注※

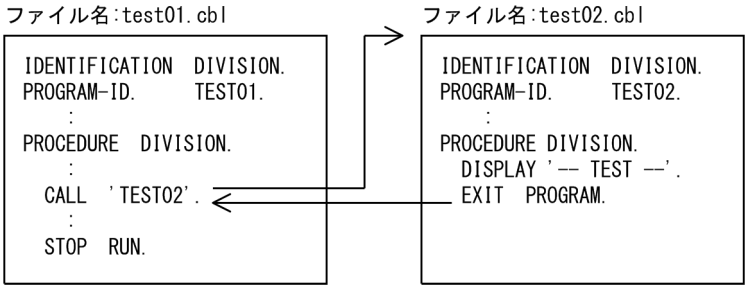
Windows(x86) COBOL2002 で有効です。

35.1.2　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，そのあと ccbl2002 コマンドや LINK 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.obj"および"test02.obj"を ccbl2002 コマンドを使用してリンクします。

```
ccbl2002 test01.obj test02.obj -OutputFile test03.exe
```

上記のコマンドを実行すると，実行可能ファイル"test03.exe"が生成されます。

(2) LINK コマンドによるリンク方法

ccbl2002 コマンドで COBOL プログラムから COBOL オブジェクトファイルを生成したあと，リンカを使用することで実行可能ファイルを作成できます。

リンクの方法を以下に示します。

1. LINK コマンドを実行する。
2. 次にマニフェストを埋め込む。

リンカについては，「38.1 リンカ」を参照してください。LINK コマンドの実行については，「35.1.2 コンパイルとリンクを別々に実行する方法」の「(2) LINK コマンドによるリンク方法」の「(a) LINK コマンドの指定」を参照してください。

マニフェストの埋め込みについては，「35.1.2 コンパイルとリンクを別々に実行する方法」の「(2) LINK コマンドによるリンク方法」の「(b) マニフェストの埋め込み方法」を参照してください。

(a) LINK コマンドの指定

リンカを使用して，COBOL プログラムを含む実行可能ファイルおよび DLL を作成するときは，リンカの引数に次のライブラリを指定する必要があります。

- 必須ライブラリ
  - GUI モードの実行可能ファイルを作成する場合
    cbl2k_32.lib

cbl2klg.lib
msvcrt.lib
kernel32.lib
vcruntime.lib
ucrt.lib

- CUI モードの実行可能ファイルを作成する場合
  cbl2k_32.lib
  cbl2klc.lib
  msvcrt.lib
  kernel32.lib
  vcruntime.lib
  ucrt.lib
- DLL を作成する場合
  cbl2k_32.lib
  cbl2kdl.lib
  msvcrt.lib
  kernel32.lib
  vcruntime.lib
  ucrt.lib

- 使用する機能によって必要となるライブラリ
  COBOL プログラムで使用する機能によって，リンク時に，関連するライブラリを指定する必要があります。
  使用する機能とライブラリについては，「35.1.1　コンパイルとリンクを同時に実行する方法」の「(4) COBOL プログラムと標準ライブラリをリンクして実行可能ファイルを生成する」を参照してください。
- LINK コマンドに指定する引数
  LINK コマンドに必要なパラメタを指定する場合，次のパラメタは必ず指定しなければなりません。
  - GUI モードの実行可能ファイルを作成の場合
    /ENTRY:WinMainCRTStartup
    /NODEFAULTLIB
  - CUI モードの実行可能ファイルを作成の場合
    /ENTRY:mainCRTStartup
    /NODEFAULTLIB
  - DLL を作成の場合（Windows(x86) COBOL2002 の場合）
    /ENTRY:_DllMainCRTStartup@12
    /NODEFAULTLIB
  - DLL を作成の場合（Windows(x64) COBOL2002 の場合）

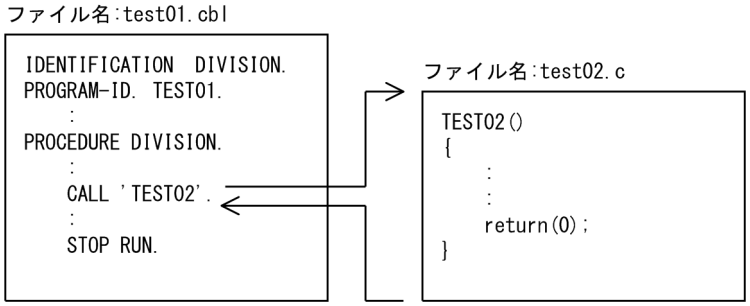
/ENTRY:_DllMainCRTStartup
/SUBSYSTEM:{CONSOLE|WINDOWS}
/NODEFAULTLIB

LINK コマンドの引数には，/NODEFAULTLIB を指定してください。
これによる，次に示す/DEFAULTLIB オプションの扱いについて説明します。

- /DEFAULTLIB:kernel32.lib
- /DEFAULTLIB:uuid.lib
- /DEFAULTLIB:MSVCRT
- /DEFAULTLIB:OLDNAMES
  kernel32.lib がライブラリの検索対象から除外されますが，kernel32.lib は明示的にリンクされるので，リンケージの結果に影響はありません。
  /NODEFAULTLIB:oldnames.lib を指定した場合，oldnames.lib の指定（/DEFAULTLIB:oldnames.lib）だけが無効になります。kernel32.lib，uuid.lib，および msvcrt.lib は，COBOL2002 の環境に含まれるため，リンケージは正常に終了します。

COBOL2002 では，LINK コマンド（LINK.exe）の実行時のパラメタを取得しません。/VERBOSE オプションを指定すると，リンカの進行状況を確認できます。

- 使用例



上記の場合，次の手順で実行可能ファイルを作成します。

1. ccbl2002 コマンドで，test01.cbl から test01.obj を生成します。

```
ccbl2002 -Compile,NoLink -Main,System test01.cbl
```

2. C コンパイラで，test02.c から test02.obj を生成します。
3. リンカで，GUI モードの実行可能ファイルを生成します。

```
LINK TEST01.obj TEST02.obj cbl2k_32.lib cbl2klg.lib
                           /out:TEST01.exe
                           /ENTRY:WinMainCRTStartup
                           msvcrt.lib kernel32.lib
                           vcruntime.lib ucrt.lib
```

(b) マニフェストの埋め込み方法

LINK コマンドを実行すると，使用する C 実行時ライブラリのバージョンやリンカオプションの指定によって，実行可能ファイル（.exe），または DLL ファイル（.dll）のほかにマニフェストファイル（.manifest）が生成されることがあります。マニフェストファイルには，使用する C 実行時ライブラリの情報や UAC 情報などが登録されていて，プログラムを実行する場合に必要です。

生成されたマニフェストファイルの情報（マニフェスト）は，MT コマンドを使用して，対応する実行可能ファイル，または DLL ファイル中に埋め込んで使用してください。埋め込まれていない場合，プログラム実行時に C 実行時ライブラリのローディングエラーになることがあります。

MT コマンドの使用例を以下に示します。

- MT コマンドに指定するパラメタ

  MT コマンドを使用してマニフェストを埋め込む場合，以下のパラメタを指定します。

  ```
  -manifest マニフェストファイル名
  -outputresource:出力ファイル名[;[#]リソースID]
  ```

  出力ファイル名

  実行可能ファイル名，または DLL ファイル名を指定します。

  リソース ID

  リソースが実行可能ファイルの場合：1

  リソースが DLL ファイルの場合：2

  なお，リソース ID の指定を省略した場合は 1 が仮定されます。

- MT コマンドの使用例
  - 実行可能ファイルの場合

    LINK コマンドを使用して実行可能ファイル**TEST01.exe** を生成した場合，マニフェストファイル **TEST01.exe.manifest** が生成されます。**TEST01.exe** 中にマニフェストを埋め込む場合のコマンド例を，以下に示します。

    ```
    MT -manifest TEST01.exe.manifest -outputresource:TEST01.exe;#1
    ```

  - DLL ファイルの場合

    LINK コマンドを使用して DLL ファイル**DLLSUB.dll** を生成した場合，マニフェストファイル **DLLSUB.dll.manifest** が生成されます。**DLLSUB.dll** 中にマニフェストを埋め込む場合のコマンド例を，以下に示します。

    ```
    MT -manifest DLLSUB.dll.manifest -outputresource:DLLSUB.dll;#2
    ```

35.1.3 コンパイルとリンクを実行する場合の注意事項

コンパイルとリンクを同時に実行する場合，および別々に実行する場合の注意事項について説明します。

(1)　COBOL プログラムと C プログラムをリンクする場合の注意事項

COBOL プログラムと C プログラムをリンクする場合は，C 実行時ライブラリに，マルチスレッドに対応したダイナミックリンクライブラリを使用する必要があります。

(2)　標準ライブラリを使用する場合の注意事項

- 次のオプションを使用する場合で，ccbl2002 コマンドで実行可能ファイルを作成する場合は，実行可能ファイルと標準ライブラリ作成時に，同じオプションを指定する必要があります。

  -XMAP,LinePrint（Windows(x86) COBOL2002 の場合），-CompatiM7（Windows(x86) COBOL2002 の場合），-OpenTP1（データコミュニケーション機能の使用と同時に使用する場合）
- テストデバッガ，カバレージを使用する場合は，実行可能ファイルと標準ライブラリのプログラム情報ファイルが共に参照できるようにする必要があります。プログラム情報ファイルの出力先の詳細については，マニュアル「COBOL2002　操作ガイド」のテストデバッガで使用するファイルの説明を参照してください。
- 実行可能プログラムを作成する場合，標準ライブラリに含まれているプログラムを，すべて動的リンクで呼び出している場合は，標準ライブラリを指定してもリンクできません。

(3)　オブジェクトファイルを別環境でリンクする場合の注意事項

- COBOL プログラムのオブジェクトファイルをコピーして，別環境でリンクする場合，オブジェクトファイルだけでなく，EXPORT 名称ファイル（.cbd）もコピーする必要があります。
- COBOL プログラムのオブジェクトファイルを用いて別環境でリンクする場合で，オブジェクトファイルの生成時に次のオプションを指定したときは，リンク時にも同じオプションを指定する必要があります。

  -XMAP,LinePrint（Windows(x86) COBOL2002 の場合），-CompatiM7（Windows(x86) COBOL2002 の場合），-OpenTP1（データコミュニケーション機能の使用と同時に使用する場合）

(4)　マニフェストの UAC 情報を変更する場合の注意事項

ccbl2002 コマンドを使用して実行可能ファイルを作成するとき，COBOL2002 コンパイラは埋め込むマニフェストに UAC 情報を指定するセクションを挿入します。既定のアクセス許可レベルは asInvoker です。このアクセス許可レベルを変更したいときは，-Link コンパイラオプションに-MANIFESTUAC リンカオプションを指定してアクセス許可レベルの値を設定してください。-MANIFESTUAC リンカオプションの詳細については，「38.1.3　オプション」の「(14)　-MANIFESTUAC［: {NO ｜ UAC フラグメント}］」を参照してください。

管理者のアクセス許可レベルで実行するように指定する例を次に示します。

```
ccbl2002 -Main,System TEST01.cbl -OutputFile TEST02.exe
-Link -MANIFESTUAC:level='requireAdministrator'
```

(5)　ccbl2002 コマンドによるリンク時にエラーが発生した場合

ccbl2002 コマンドまたは ccbl コマンドから呼び出す LINK コマンドや MT コマンドは，出力ファイルが OS やウィルスチェッカなどから一時的に共有されることが原因で，エラーになることがあります。この場合，時間をおいて再コンパイルしてください。時間をおいても問題が解決しないときは，次のどちらかの方法で対処してください。

- 環境変数 CBLLINKINTERVAL で，対象ファイルへのアクセス間隔を変更した状態にして再コンパイルしてください。なお，環境変数 CBLLINKINTERVAL については，「33.6.3　コンパイラ環境変数の詳細」の「(12)　CBLLINKINTERVAL」を参照してください。
- ccbl2002 コマンドに-ManifestFileExt オプションを指定して，マニフェストの埋め込みを抑止した状態で再コンパイルしてください。なお，-ManifestFileExt オプションについては，「33.5.10　リンクの設定」の「(13)　-ManifestFileExt オプション」を参照してください。

(6)　実行可能ファイル，DLL ファイルの名称の注意事項

実行可能ファイル，または DLL ファイルの名称に「;」および「@」は指定しないでください。作成する実行可能ファイル，または DLL ファイルの名称に「;」および「@」が含まれていると，MT コマンドでエラーとなりマニフェストの埋め込みに失敗します。

35.2 DLL の作成方法

35.2.1 DLL の作成

DLL（ダイナミックリンクライブラリ）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，拡張子が（.dll）のファイルを DLL と呼びます。

DLL 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，DLL 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。

なお，Windows(x64) COBOL2002 の場合は「-Dll,Stdcall」と「Dll,Cdecl」を「-Dll」に読み替えてください。

(1) DLL を作成する

ccbl2002 コマンドで COBOL プログラムから DLL を作成する場合は，-Dll,Stdcall または-Dll,Cdecl オプションを指定します。コンパイルとリンクを別々に ccbl2002 コマンドでする場合は，コンパイル時，リンク時共に-Dll,Stdcall または-Dll,Cdecl オプションを指定します。

次に，DLL を作成する例を示します。

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

COBOL プログラム"test02.cbl"と-Dll,Stdcall または-Dll,Cdecl オプションを指定し，-OutputFile オプションに DLL 名称"test02.dll"を指定します。

```
ccbl2002 -Dll,Stdcall test02.cbl -OutputFile test02.dll
```

上記のコマンドを実行すると，DLL ファイル"test02.dll"が生成されます。

注意事項

- 見出し部に INITIAL 句がないプログラムに対して-Dll,Stdcall または-Dll,Cdecl オプションだけを指定した場合，生成される DLL は，初期化属性を持ちません。このため，DLL が繰り返し呼び出されても，次の情報が初期状態に戻されないで，前の状態を保持します。

（情報が保持される項目）

  - プログラムに関連するファイルや報告書の状態・画面節（SCREEN SECTION），画面節（WINDOW SECTION），通信節，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），画面節（WINDOW SECTION），ファイル節，サブスキーマ節，通信節の内容
- PERFORM 文に対する制御機構
- ALTER 文で設定した GO TO 文

- -Dll,Stdcall または-Dll,Cdecl オプションを指定してコンパイルするプログラム中で EXTERNAL 句を使用する場合，DYNAMIC 指定の EXTERNAL 句だけが使用できます。

(2) 呼び出し時に DLL を初期化する

ccbl2002 コマンドに，-Dll,Stdcall または-Dll,Cdecl オプションと合わせて，-DllInit オプションを指定すると，生成された DLL は初期化属性プログラムとなり，その DLL が呼び出されるたびに，次の情報が初期状態に戻されます。

初期化属性のプログラムの詳細については，「18.4.1 プログラム属性」を参照してください。

(3) 注意事項

- LINK コマンドを使用して DLL ファイルを作成する場合，DLL ファイルに LINK コマンドが生成したマニフェストファイルを埋め込む必要があります。

  詳細は，「35.1.2 コンパイルとリンクを別々に実行する方法」の「(2) LINK コマンドによるリンク方法」を参照してください。

- Windows(x86) COBOL2002 の場合，STDCALL 呼び出し規約の C プログラムを動的なリンクで呼び出したいときは，プログラム名は装飾名を付けてエクスポートする必要があります。エクスポートには次の方法があります。
  - LINK コマンドの/DEF オプションにモジュール定義ファイル（def ファイル）を指定する
  - LINK コマンドの/EXPORT オプションを指定する
  - C ソースファイル内でエクスポートすることを明示する

  STDCALL 呼び出し規約のプログラム名の装飾やプログラムのエクスポートについては，MSDN（Microsoft Developer Network）などのドキュメントを参照してください。なお，COBOL プログラムの場合，COBOL コンパイラがプログラム名でエクスポートするため，リンク時に明示する必要はありません。

35.3　DLL を呼び出す実行可能ファイルの作成方法

DLL を呼び出す実行可能ファイルを作成するときに必要なインポートライブラリ，および-StdCall オプションと stdcall 呼び出し指示ファイル（.cbw）について説明します。

35.3.1　インポートライブラリの指定

DLL を呼び出すプログラムから実行可能ファイルを生成する場合，静的なリンクで DLL を呼び出すには，ccbl2002 コマンドにインポートライブラリを指定する必要があります。

なお，Windows(x64) COBOL2002 の場合は「-Dll,Stdcall」と「Dll,Cdecl」を「-Dll」に読み替えてください。

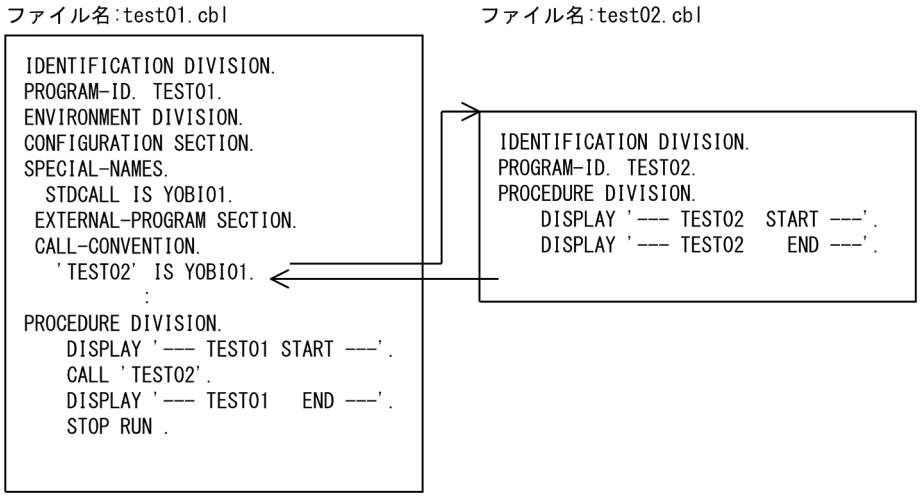
DLL の呼び出しについては，「18.6　DLL に含まれるプログラムの呼び出し」を参照してください。

(1)　静的なリンクの場合

静的にリンクする実行可能ファイルを作成する場合は，ccbl2002 コマンドにインポートライブラリを指定します。

ccbl2002 コマンドの指定例を，次に示します。

使用例



ccbl2002 コマンドの指定

ccbl2002 コマンドで-Dll,Stdcall または-Dll,Cdecl オプションを指定して，"test02.cbl"に対する DLL "test02.dll"を生成します。このとき，インポートライブラリ"test02.lib"も同時に生成されます。

```
ccbl2002 -Dll,Stdcall test02.cbl -OutputFile test02.dll
```

主プログラム"test01.cbl"に対し-Main,System オプションを指定し，"test02.dll"作成時に生成されたインポートライブラリ"test02.lib"を指定します。

```
ccbl2002 -Main,System test01.cbl test02.lib -OutputFile test03.exe
```

上記のコマンドを実行すると，DLL ファイル"test02.dll"に静的にリンクする実行可能ファイル"test03.exe"が生成されます。

(2)　動的なリンクの場合

動的なリンクで実行可能ファイルを作成する場合には，インポートライブラリは必要ありません。

(3)　DLL からほかの DLL を呼び出す場合

DLL からほかの DLL を呼び出す場合も，実行可能ファイルと同様に，静的リンクの場合は呼び出し元プログラムのインポートライブラリをすべて指定する必要があります。また，動的リンクの場合はインポートライブラリの指定は不要です。

35.3.2　-StdCall オプションと stdcall 呼び出し指示ファイル (Windows(x86) COBOL2002 で有効)

stdcall 呼び出し規約のプログラムを呼び出す場合，呼び出す側の呼び出し規約も stdcall にする必要があります。呼び出すプログラム側で呼び出し規約を stdcall にするには，COBOL ソース中に指定する方法（CALL-CONVENTION 句）と，stdcall 呼び出し指示ファイルを使用する方法の 2 種類があります。

呼び出し属性を COBOL ソース中に指定する方法

呼び出し規約を COBOL ソース中に指定する方法です。環境部の EXTERNAL-PROGRAM SECTION の CALL-CONVENTION 段落で指定します。

指定方法の詳細については，マニュアル「COBOL2002　言語　拡張仕様編」「25.2.1　プログラム間連絡機能の環境部」，および「18.4.2　呼び出し規約を参照してください。

呼び出し属性を stdcall 呼び出し指示ファイルで指定する方法

stdcall 呼び出し指示ファイル（.cbw）は，COBOL プログラムから stdcall 呼び出し規約で呼び出す DLL のプログラム名を登録しておくためのファイルです。このファイル中に記述したプログラムは，COBOL プログラムから stdcall 規約で呼び出されます。

この方法は，呼び出し規約を言語仕様でサポートする前に使用していた方法です。このため，特に理由がない場合は，呼び出し属性を COBOL ソース中に指定する方法を使用することを推奨します。

(1)　コンパイル方法

作成した stdcall 呼び出し指示ファイルを有効にするためには，コンパイル時に-StdCall オプションを指定する必要があります。

-StdCall オプションを指定すると，stdcall 呼び出し指示ファイルがあれば使用されます。また，-StdCall オプションの指定がなければ，stdcall 呼び出し指示ファイルがあっても無視されます。

(2) stdcall 呼び出し指示ファイルの規則

stdcall 呼び出し指示ファイルの作成規則

- stdcall 呼び出し指示ファイル名は，ソースファイル名.cbw とし，ソースファイルと同じフォルダに格納しておく必要があります。※

注※

-StdCallFile オプションを使用すれば，任意のフォルダに格納されている任意の名称の stdcall 呼び出し指示ファイルを指定できます。

stdcall 呼び出し指示ファイルの記述規則

stdcall 呼び出し指示ファイルには，呼び出す DLL のプログラム名をテキストエディタを使用して記述します。このときの規則を次に示します。

- プログラム名を複数記述する場合，プログラム名は改行して記述するか，または空白で区切って記述します。
- ファイルの中にはコメントを 1 行単位に記述できます。コメントは，//の横に続けて記述します。
- プログラム名の構成規則に反した文字を指定した場合，結果は保証しません。
- stdcall 呼び出し指示ファイル中にプログラム名が記述されていない場合，stdcall 呼び出し指示ファイルがないものとして扱われます。
- プログラム名を 160 バイト以上記述した場合，先頭から 160 バイトまでが有効となります。ただし，160 バイト目と 161 バイト目が日本語となるときは，先頭から 159 バイトまでが有効となります。
- -Compati85,Syntax オプションを指定して，プログラム名を 30 バイト以上記述した場合，先頭から 30 バイトまでが有効となります。
- プログラム名の変換規則に対する処理はされないため，プログラム名は変換規則適用後の文字で記述する必要があります。
- プログラム名中の文字に対しては，COBOL の語の等価規則は適用されません。
- .cbw ファイルの記述例を次に示します。

(例)

```
----+----1----+----2----+----3----+----
プログラム名
//コメント
        プログラム名
  //   コメント
プログラム名   プログラム名   プログラム名
     :
```

- -Compati85,Syntax オプションを指定している場合で，プログラム名を 30 バイト以上記述したときは，先頭から 30 バイトまでがプログラム名として有効となります。

(3) stdcall 呼び出し指示ファイルの使用例

stdcall 呼び出し指示ファイルを使用してプログラムをコンパイルする例を，次に示します。

使用例

(DLLMAIN.cbl)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DLLMAIN.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 DLLMODUL PIC X(7) VALUE 'DLLSUB2'.
       PROCEDURE DIVISION.
      *
      *** DLLモジュール CALL(定数呼び出し)
           CALL 'DLLSUB1'.
      *
      *** DLLモジュール CALL(一意名呼び出し)
           CALL DLLMODUL.
      *
           STOP RUN.
```

(DLLSUB1.cbl)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DLLSUB1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       PROCEDURE DIVISION.
      *
      * STOP 定数文の実行
           STOP 01.
      *
           EXIT PROGRAM.
```

(DLLSUB2.cbl)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DLLSUB2.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       PROCEDURE DIVISION.
      *
      * STOP 定数文の実行
           STOP 02.
      *
           EXIT PROGRAM.
```

1. COBOL プログラム"DLLSUB1.cbl"，"DLLSUB2.cbl"をそれぞれ-Dll,Stdcall オプションを指定してコンパイルし，DLL を作成します。

```
ccbl2002 DLLSUB1.cbl -Dll,Stdcall -OutputFile DLLSUB1.dll
ccbl2002 DLLSUB2.cbl -Dll,Stdcall -OutputFile DLLSUB2.dll
```

2. 主プログラム"DLLMAIN.cbl"と同じフォルダに stdcall 呼び出し指示ファイル"DLLMAIN.cbw"を作成し，-StdCall オプションを指定してコンパイルします。このとき，DLLSUB1 を静的リンクするために，インポートライブラリ DLLSUB1.lib を指定します。

```
ccbl2002 -Main,System DLLMAIN.cbl -StdCall DLLSUB1.lib
         -OutputFile DLLMAIN.exe
```

DLLMAIN.cbw の記述内容

```
// DLLSUB1.dllとDLLSUB2.dllを.cbwファイルに登録
  DLLSUB1 DLLSUB2
```

35.4　リンカパスの切り替え機能

リンカパスの切り替え機能の概要，および使用時の注意事項について説明します。

35.4.1　機能概要

リンカパスの切り替え機能は，COBOL2002 コンパイラから呼び出す LINK.exe，RC.exe，および MT.exe のパスを切り替える機能です。

この機能は，次の環境変数を使用して，環境変数に設定されたパス名でそれぞれのコマンドを COBOL2002 コンパイラから呼び出します。

- CBLLINKER=LINK コマンドのパス
- CBLRESOURCECOMPILER=RC コマンドのパス
- CBLMANIFESTTOOL=MT コマンドのパス

なお，各環境変数の詳細は，「33.6.3　コンパイラ環境変数の詳細」の「(11)　CBLLINKER」，「(13) CBLMANIFESTTOOL」，および「(16)　CBLRESOURCECOMPILER」を参照してください。

これらの環境変数の設定がない場合，COBOL2002 コンパイラは COBOL2002 のインストールフォルダ¥bin¥SDK，および Windows SDK の bin へのパスの下のコマンドを絶対パス名で呼び出します。

環境変数 CBLLINKER，環境変数 CBLRESOURCECOMPILER，および環境変数 CBLMANIFESTTOOL を設定した場合，-Details オプションを指定すると，COBOL2002 コンパイラでリンクするとき，実行するコマンド名を以下の形式で出力します。

形式

```
環境変数CBLRESOURCECOMPILERに設定されているパス名:
環境変数CBLLINKERに設定されているパス名:
環境変数CBLMANIFESTTOOLに設定されているパス名:
```

使用例

コンパイラから呼び出すリンカを Visual Studio のリンカに切り替えるときの例を次に示します。

1. Windows のプログラム一覧にある［COBOL2002］下のメニューから［COBOL コマンドプロンプト for COBOL2002］を選びます。

2. コマンドプロンプト上で，Visual Studio の環境設定バッチ（vcvarsall.bat）を起動して，Visual Studio のリンカを使用するために必要な環境変数を設定します。

   なお，Visual Studio の環境設定バッチの引数などの使用方法については，Visual Studio のリファレンスなどで確認してください。

3. 環境変数を設定します。

   環境変数 CBLLINKER，環境変数 CBLRESOURCECOMPILER，および環境変数 CBLMANIFESTTOOL を設定します。

設定例を次に記します。

（例）

```
set CBLLINKER=Microsoft Visual Studio 2015のbinへのパス¥link.exe
set CBLRESOURCECOMPILER=Windows SDKのbinへのパス¥rc.exe
set CBLMANIFESTTOOL=Windows SDKのbinへのパス¥mt.exe
```

4. ccbl2002 コマンドによってリンクします。

（例）

```
C:¥ examples>ccbl2002 -Main,System cmain.obj cobolsub.cbl -Details
Windows 10 SDKのbinへのパス¥rc.exe:                          } ユーザ指定の情報
Microsoft Visual Studio 2015のbinへのパス¥link.exe:           }
Microsoft (R) Incremental Linker Version x.xx.xxxxx.xxxxx
Copyright (C) Microsoft Corporation.  All rights reserved.

/ENTRY:WinMainCRTStartup
"/OUT:cmain.exe"
/NODEFAULTLIB
"cmain.obj"
"cobolsub.obj"                                                } リンケージリスト
"cmain.res"                                                   }
cbl2klg.lib
cbl2k_32.lib
msvcrt.lib
kernel32.lib
vcruntime.lib
ucrt.lib
cbldcsm2k.lib
   ライブラリ cmain.lib とオブジェクト cmain.exp を作成中
Windows 10 SDKのbinへのパス¥mt.exe:                          …ユーザ指定の情報
```

35.4.2　リンカパスの切り替え機能使用時の注意事項

開発マネージャでプログラムを開発するとき，この機能を使用する場合は，Visual Studio の環境設定バッチで設定される環境変数と値を，開発マネージャの起動前にシステム環境変数に設定するか，または Visual Studio のコマンドプロンプト上から開発マネージャを起動してください。

36 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。

36.1　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。

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，ソースファイル名に拡張子.exe を付けた名称になります。

-OutputFile オプションの詳細は，「33.5.10　リンクの設定」の「(10)　-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。

実行方法には次のような方法があります。

- エクスプローラで実行可能ファイル（.exe）のアイコンをダブルクリックする。
- コマンドプロンプト，または Windows の「ファイル名を指定して実行」ダイアログボックスに，形式に記載されているように実行可能ファイル（.exe）を入力する。
- 開発マネージャの［実行］コマンドを使用する。
- 実行支援を起動して実行する。

開発マネージャ，実行支援を利用した実行方法については，マニュアル「COBOL2002　操作ガイド」を参照してください。

なお，実行支援で生成するプログラム別実行環境ファイル（.cbr）は，実行ファイルと同じフォルダに，同じ名称（拡張子は除く）で保存する必要があります。

注意事項

プログラムの実行時に，プログラムが実行できる十分な仮想メモリがない場合，「指定したプログラムは実行されません。」というメッセージが出力されます。この場合，Windows の仮想メモリを大きくしてください。

36.2.1　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，次のどちらか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
- 実行支援を使用し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

コマンドプロンプトまたは Windows システムの環境変数から，次の形式で設定します。

形式

```
環境変数=環境変数の値
```

(a)　注意事項

- 空白を含むフォルダ名やファイル名を環境変数の値に指定するときは，引用符（ " ）で囲まないでください。
- 環境変数の値の長さは，次の表に示す範囲内でシステムとして有効な値を指定してください。

表 36–1　環境変数の値の長さの範囲

環境変数名	値の長さの範囲	範囲外の長さを指定した場合
• CBLTDEXEC • CBLLDLL	1～32,766 バイト	COBOL プログラム実行時に有効な環境変数とみなされません。
上記以外の環境変数	1～1,024 バイト	

- 実行環境ファイル（.cbr）からの環境変数の読み込み，および環境変数へのアクセス機能で，値にメタキャラクタ（"%変数名%"の形式）が含まれているときは，次の規則に従います。
  - メタキャラクタに環境変数名が指定されると，メタキャラクタの部分はその環境変数の現在の値と置き換えられます。置き換えの規則は，コマンドインタプリタがメタキャラクタを扱うときの規則と同じです。
  - メタキャラクタに指定された環境変数がなかった場合，メタキャラクタの記述は置き換えられずにそのまま残ります。
  - メタキャラクタを置き換えた結果，環境変数の値の長さが 32,766 バイトを超えた場合，または置き換えが正常に終了しなかった場合は，置き換え前の環境変数の値が登録されます。

（例）次の環境変数が指定されている場合

```
CBLL1=test1.dll
CBLL2=test2.dll
CBLLDLL=test3.dll;test4.dll
```

1.正常に置き換えられる例

```
CBLLDLL=%CBLLDLL%;%CBLL1%;%CBLL2%;test5.dll;test6.dll
↓
CBLLDLL=test3.dll;test4.dll;test1.dll;test2.dll;test5.dll;test6.dll
```

2.環境変数が見つからないので，置き換えられない例

```
CBLLDLL=%CBLLDLL%;%CBLL3%;test5.dll
↓
CBLLDLL=test3.dll;test4.dll;%CBLL3%;test5.dll
```

(2) 実行支援を使用した環境変数の設定方法

実行支援を使用して，実行時環境変数を設定できます。

詳細は，マニュアル「COBOL2002　操作ガイド」を参照してください。

(3) 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。

- 環境変数の値に YES を指定する場合，大文字と小文字は等価とみなされます。
- システム環境変数と実行支援での環境変数で，異なる値を設定した場合の優先順位を次に示します。
  1. プログラム別実行環境ファイル（実行支援で設定する.exe ファイルと同じ名称の.cbr ファイル）
  2. 共通実行環境ファイル（環境変数 CBLCOMCBR で指定するファイル）
  3. コマンドプロンプトやコントロールパネルで設定した環境変数

36.2.2 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。

(1) 一般

環境変数名	指定する内容
CBLABNCODE	CBLABN サービスルーチンの引数を COBOL2002 アプリケーションの終了コードにするかどうか
CBLCOMCBR	システム共通の実行環境でプログラムを実行するときの実行環境ファイル名
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
CBLLDLL	動的なリンクで実行時にダイナミックリンクする DLL の名称
CBLLPROGDLL	DLL 自動ロード機能を使うかどうか
CBLPGMSEARCHTRC	プログラム検索トレース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	プレロードリスト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_BATCH	COBOL2002 アプリケーションの終了と同時にプロセスを終了させるかどうか
CBL_SYSERR	実行時エラーメッセージの出力先ファイル名

(2) 少量データ

環境変数名	指定する内容
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
CBL_STOPNOADV	CUI モードで 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

(3) ファイル

環境変数名	指定する内容
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
CBLEUDCFUNC	外字を有効にするため内部的に発行している EnableEUDC 関数の扱いの指定
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

環境変数名	指定する内容
CBLF_ファイル名	印刷機能を利用したとき，スプールに登録される印刷文書名称の指定
CBLGDIINTERVAL	GDI モード印刷で INTERVAL 指定がないとき，実行支援の印刷書式に設定した字間隔を使用するかどうか
CBLGDIWMSG	GDI モード印刷機能を使用する場合に，警告メッセージの出力を抑止するかどうか
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
CBLISAMLARGE※	実行単位中のすべての ISAM による索引編成ファイルに対してラージファイル形式を適用するかどうか
CBLPRTEXCHR	CBLP_ファイル名を設定して ESC/P モード印刷機能を利用するプリンタへ外字を出力するとき，ベンダ定義文字を外字として出力する
CBLP_ファイル名	ESC/P モード印刷機能を利用するプリンタへ，Windows 上で作成した外字を登録する
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
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_BTRPGSZ※	Btrieve（Pervasive.SQL）による索引編成ファイルのページサイズ
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_RECLOCKMAX	相対編成ファイルで LOCK MODE 句に MANUAL を指定したとき同時に施錠できるレコード数
CBL_SYSCSVIN	FROM SYSCSV 指定の ACCEPT 文での入力ファイル名
CBL_SYSCSVOUT	UPON SYSCSV 指定の DISPLAY 文での出力ファイル名
CBL_外部装置名	ファイル入出力での入出力ファイル名

注※

Windows(x86) COBOL2002 で有効です。

(4) 画面

環境変数名	指定する内容
CBLACTWIN	COBOL アプリケーションの出力するコンソールおよび画面機能のウィンドウを DISPLAY 文のメッセージ出力のタイミングでアクティブにするかどうか
CBLATTRIBUTE	画面節（WINDOW SECTION）の ERASE ATTRIBUTE 文で指定したフィールドに重ねて表示してあるけい線，および属性を消去するかどうか
CBLAUTOCLEAR	画面機能でのデータ入力時の動作変更
CBLENTERCHK	ACCEPT 文や REPLY 文実行時，画面節（WINDOW SECTION）の ENTER-CHECK 句指定項目がデータ未入力の場合，WINDOW-STATUS 特殊レジスタへの値設定
CBLFEP	-JPN,Alnum，または-JPN,V3JPN オプション指定時に，カーソル移動によって自動的に日本語入力に切り替えるかどうか
CBLIMEPOS	画面機能で，日本語をフィールドに直接入力するかどうか
CBLJCPOPENDKEY	JCPOPUP サービスルーチンの終了キーの拡張
CBLM7ENDKEY	MIOS7 の終了キーを，［Alt］＋［F1］～［Alt］＋［F8］から［Shift］＋［F1］～［Shift］＋［F8］に変更する
CBLNOCLOSE	COBOL プログラム実行中の強制中断を抑止するかどうか
CBLONLYNUM	環境変数 CBLAUTOCLEAR での画面動作の変更を，画面節（WINDOW SECTION）の数字項目，および数字編集項目だけに限定するかどうか
CBLOVERFLOW	画面節（WINDOW SECTION）入力時に，自動カーソル位置づけを抑止するかどうか，およびオーバフローをチェックするかどうか
CBLSETFIELD	画面節（WINDOW SECTION）の SET 文でフィールドの属性を変更するとき，それまで SET 文で変更していた属性を初期化するかどうか
CBLUNDERDOT	画面機能で，入力および入出力フィールドにピリオド（.）を表示するかどうか
CBLUPDOWNMOVE	画面機能（画面節（WINDOW SECTION））での ACCEPT 文，または REPLY 文実行時，［↑］キー，［↓］キーの動作を変更するかどうか

(5) 画面（XMAP）

環境変数名	指定する内容
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

(6) 整列併合

環境変数名	指定する内容
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フォルダ名

(7) 拡張機能

環境変数名	指定する内容
CBLOPS	MIOS7 COBOL85 互換のために複数の環境変数を指定する
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

(8) デバッグ

環境変数名	指定する内容
CBLABNLST	異常終了時要約情報リストの出力先
CBLDDUMP	異常終了時のデータ領域ダンプの出力先
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
CBLEXCEPT	プログラムの実行で例外が発生したときの動作
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かどうか
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

(9)　イベントログ

環境変数名	指定する内容
CBLSYSLOG	イベントログファイル出力機能を使用するかどうか
CBLSYSLOGLVL	イベントログファイル出力機能での出力対象イベントの種類
CBLSYSLOGSRV	イベントログファイル出力機能でのイベント出力先のコンピュータ名

(10)　オブジェクト指向

環境変数名	指定する内容
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

36.2.3　一般

(1)　CBLABNCODE

CBLABN サービスルーチンの実行時，サービスルーチンの引数をプログラムの終了コードに反映させたい場合に指定します。

詳細は，「30.4.3　CBLABN」を参照してください。

(2)　CBLCOMCBR

システムで共通な実行環境でプログラムを実行したい場合，共通実行環境ファイル名を指定します。共通実行環境ファイル名には，実行支援で作成した.cbr ファイルを絶対パスで指定します。

この環境変数には，複数のファイル名を指定できません。また，この環境変数に指定した共通実行環境ファイル中の CBLCOMCBR は無効となります。

なお，共通実行環境ファイルに加えて，プログラム別に実行環境を設定したい場合は，実行支援で.exe ファイルと同じ名称の実行環境ファイルを作成します。詳細は，マニュアル「COBOL2002　操作ガイド」を参照してください。

(3)　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。

(4) CBLLANG

動作する言語環境（文字コード）を指定します。

詳細は，「28.4.2 実行」を参照してください。

(5) CBLLDLL

ダイナミックリンクする DLL の名称を指定します。

詳細は，「18.6.2 DLL 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。

(6) CBLLPROGDLL

DLL 自動ロード機能を使用する場合，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，「18.6.2 DLL 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。

(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 DLL の検索情報
- プレロード時の DLL 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 DLL 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，「28.4.2 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_BATCH

COBOL アプリケーションの終了（STOP RUN 文の実行または実行時エラーの発生）と同時にプロセスを終了させたい場合に 1 を指定します。1 以外を指定したかまたはこの環境変数を指定しない場合は，コンソールウィンドウのシステムメニューの［閉じる］を選んでプロセスを終了させなければなりません。

この環境変数は，CUI モードでの実行時，およびコンソールウィンドウが出力されていない場合には意味を持ちません。

なお，この環境変数の使い方については，「30.4.6 CBLEXEC」も参照してください。

(13) 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。ファイル名はドライブ名からの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	
	GUI モード	CUI モード
CBL_SYSERR=stdin〔+〕	ファイル名「stdin」	
CBL_SYSERR=stdout〔+〕	ファイル名「stdout」	標準出力※
CBL_SYSERR=stderr〔+〕	ファイル名「stderr」	標準エラー出力※
CBL_SYSERR=syslog〔+〕	ファイル名「syslog」	
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

CUI モードのときに，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。

36.2.4　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，「10.2.4　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，「10.2.4　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)　CBL_STOPNOADV

CUI モードの COBOL プログラムで STOP 定数文を実行した場合，実行時メッセージ ID と，定数の直後の改行文字の出力を抑止したい場合，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，「10.2.6　STOP 文」を参照してください。

(4)　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，「10.2.3　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。

(5)　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，「10.2.5　DISPLAY 文によるデータの出力」を参照してください。

(6)　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，「10.2.5　DISPLAY 文によるデータの出力」を参照してください。

(7)　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，「10.2.3　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。

36.2.5　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。

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。

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。

(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) SAMBADV／NOSAMBADV

SAMBADV は，ADVANCING 指定を書かない WRITE 文に BEFORE ADVANCING 1 LINE を仮定します。NOSAMBADV はこれを仮定しません。省略時は NOSAMBADV 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。

(e) BTMODIFY／NOBTMODIFY（Windows(x86) COBOL2002 で有効）

BTMODIFY は，Btrieve（Pervasive.SQL）による索引編成ファイルの主レコードキーに対し，変更可能モードが設定されている物理ファイルをオープンできるようにします。このオプション設定時には次の注意が必要です。

- BTMODIFY を指定した場合でも，実際に主レコードキーの変更はできない。
- BTMODIFY を指定した場合でも，COBOL で作成されるファイルの主レコードキーは変更不可モードとなる。

なお，省略時は NOBTMODIFY が仮定されます。

(f)　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。

(g)　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。

(h)　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。

(i)　SAMPADV／NOSAMPADV

SAMPADV は，OPEN 文の実行後，最初に実行される AFTER ADVANCING PAGE 指定の WRITE 文についての行制御を有効にします。NOSAMPADV は，行制御を無効にします。省略時は SAMPADV が仮定されます。このオプション設定時には次の注意が必要です。

- SAMPADV および NOSAMPADV は，順編成ファイル，テキスト編成ファイルの印刷機能，書式印刷機能，および報告書作成機能※に対して有効となります。
- LINAGE 句が指定されている場合，SAMPADV および NOSAMPADV の指定は無効となります。

注※

報告書作成機能の GENERATE 文は，内部的に WRITE 文を使用するため，このオプションが有効になります。

(j)　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 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。

(k)　ISAMLARGE／NOISAMLARGE（Windows(x86) COBOL2002 で有効）

ISAMLARGE は，ISAM による索引編成ファイルにラージファイル入出力機能を有効にします。NOISAMLARGE は，ISAM による索引編成ファイルに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とします。省略時は NOISAMLARGE が仮定されます。詳細は，「6.11　ラージファイル入出力機能」を参照してください。

(l)　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。

(m)　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，「28.5.2　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。

(n)　RDBOPURGE／NORDBOPURGE

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) CBLEUDCFUNC

GDI モード印刷および ESC/P モード印刷の OPEN 文で，外字を有効にするため内部的に発行している EnableEUDC 関数の扱いを変更したいときに指定します。

詳細は，「8.2.6 外字の有効化」を参照してください。

(5) 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「13.7 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。

(6) CBLF_ファイル名

COBOL の GDI 印刷機能および ESC/P 印刷機能を使用した場合に，スプールに登録される印刷文書名称を指定します。

詳細は，「8.2.5 印刷文書名称」の「(2) ユーザ指定の印刷文書名称」を参照してください。

(7) CBLGDIINTERVAL

GDI モード印刷機能を使用する場合に，CHARACTER TYPE IS INTERVAL 句が出現しない間の項目の字間隔に対して，実行支援の印刷書式に設定した字間隔を適用したいときに YES を指定します。ただし，CHARACTER TYPE 句の指定がまったくないレコードに対しては，環境変数 CBLGDIINTERVAL の指定に関係なく，実行支援の印刷書式に設定した字間隔が適用されます。なお，環境変数 CBLGDIINTERVAL に YES を指定した場合の詳細については，「8.4.2 出力形態とレコード形式」の「(2) CHARACTER TYPE 句指定での印刷」の「(e) 環境変数 CBLGDIINTERVAL を使用した出力例」を参照してください。

YES 以外の値を指定したときは，無効となります。

(8) CBLGDIWMSG

GDI モード印刷機能を利用してプリンタ出力する場合に，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。

(9) 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。

(10) 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。

(11) CBLISAMLARGE（Windows(x86) COBOL2002 で有効）

実行単位中のすべての ISAM による索引編成ファイルでラージファイル入出力を可能にするときに 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，「6.11　ラージファイル入出力機能」を参照してください。

(12) CBLPRTEXCHR

環境変数 CBLP_ファイル名を指定して ESC/P モードプリンタに外字を出力する場合で，ベンダ定義文字を外字として出力したい場合に指定します。

詳細は，「8.5.3　外字の出力方法」の「(2)　ベンダ定義文字の外字出力」を参照してください。

(13) CBLP_ファイル名

Windows 上で作成した外字を，そのままのコードで ESC/P モードプリンタに出力する場合に指定します。

詳細は，「8.5.3　外字の出力方法」の「(1)　ユーザ定義文字の外字出力」を参照してください。

(14) CBLRDBDATAERR

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。YES 以外の値を指定したときは，無効となります。

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」の「(b)　データチェック機能」を参照してください。

(15) CBLRDBILWAIT

HiRDB による索引編成ファイルで，ファイルを入力モードで開いたときに排他オプションを設定する場合に 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，「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「(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。

(16)　CBLRDBOPURGE

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あるレコードを高速で削除したいときに 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，「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」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。

(17)　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。

(18)　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，「28.5.2　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。

(19)　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。

(20)　CBLX_外部装置名（Windows(x86) COBOL2002 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，「8.6　XMAP3 による印刷（Windows(x86) COBOL2002 で有効）」を参照してください。

(21)　CBL_BTRPGSZ（Windows(x86) COBOL2002 で有効）

Btrieve（Pervasive.SQL）による索引編成ファイルのページサイズを指定します。

詳細は，「6.10.4　使用できる機能と制限事項」を参照してください。

(22)　CBL_RDBCOMMIT

HiRDB による索引編成ファイルで COMMIT 文，ROLLBACK 文によって RDB アクセスのトランザクションを管理する場合に指定します。

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「(1)　トランザクション管理機能」を参照してください。

(23)　CBL_RECLOCKMAX

相対編成ファイルで，LOCK MODE 句に MANUAL が指定されたとき，同時に施錠できるレコード数を指定します。

詳細は，「7.2.2　レコードレベルのファイル共用」を参照してください。

(24)　CBL_SYSCSVIN

FROM SYSCSV 指定の ACCEPT 文でアクセスする CSV ファイルを，絶対パス名で指定します。

詳細は，「付録 D.1　ACCEPT／DISPLAY 文を使用した CSV ファイルへのアクセス」を参照してください。

(25)　CBL_SYSCSVOUT

UPON SYSCSV 指定の DISPLAY 文でアクセスする CSV ファイルを，絶対パス名で指定します。

詳細は，「付録 D.1　ACCEPT／DISPLAY 文を使用した CSV ファイルへのアクセス」を参照してください。

(26)　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。

36.2.6　画面

(1)　CBLACTWIN

COBOL アプリケーションの出力するコンソールおよび画面機能のウィンドウを，DISPLAY 文がメッセージを出力するタイミングでアクティブにしたい場合に 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。

(2) CBLATTRIBUTE

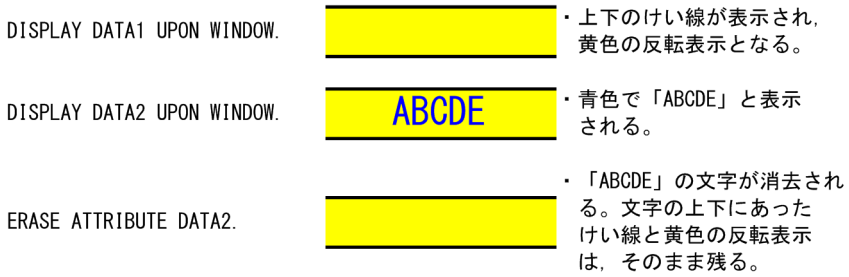
画面節（WINDOW SECTION）の ERASE ATTRIBUTE 文で指定したフィールドに重ねて表示されているけい線や属性を消去しない場合に YES を指定します。YES 以外の値を指定したときは，無効となります。

（例）

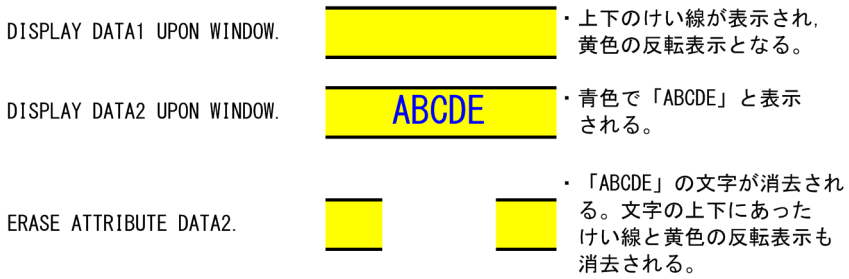
次のようなデータ項目を画面に表示する場合，環境変数 CBLATTRIBUTE を指定するときと指定しないときでは，次のように動作が異なります。

```
01 GAMEN1.
  02 DATA1 LINE 2 COLUMN 5 THRU 20
           RULE OVER UNDER
           COLOR YELLOW REVERSE.
  02 DATA2 PIC X(5) LINE 2 COLUMN 10
           COLOR BLUE
           VALUE 'ABCDE'.
```

CBLATTRIBUTE=YES の場合



CBLATTRIBUTE=YES 以外の場合



注意事項

けい線や属性は，フィールドの左端より左側から，フィールドの右端より右側の範囲に設定されている必要があります。これ以外の範囲にけい線や属性が設定されている場合，環境変数 CBLATTRIBUTE での動作は保証しません。

（例）

フィールドのデータ項目

```
PIC X(5) LINE 5 COLUMN 5 VALUE 'ABCDE'.
```

- 正しく表示されるけい線の例

```
LINE 5 COLUMN 4 THRU 11 RULE OVER UNDER.
```

- 表示が保証されないけい線の例

```
LINE 5 COLUMN 5 THRU 11 RULE OVER UNDER.
LINE 5 COLUMN 4 THRU 10 RULE OVER UNDER.
```

(3) CBLAUTOCLEAR

画面節（SCREEN SECTION および WINDOW SECTION）の入力文の動作を変更します。環境変数 CBLAUTOCLEAR に YES を指定すると，カーソルが移動したフィールドが選択状態となります。

YES 以外の値を指定したときは，無効となります。

次に，画面節（SCREEN SECTION および WINDOW SECTION）での環境変数 CBLAUTOCLEAR の機能を説明します。

(a) 画面節（SCREEN SECTION および WINDOW SECTION）での共通の動作

- 環境変数 CBLAUTOCLEAR に YES を指定した場合，フィールドに入力制御が渡ると，カーソルはフィールド全体を選択した状態になります。これを選択状態といいます。
- フィールドが選択状態のままでデータを入力すると，入力前のデータの内容がクリアされ，入力したデータが格納されます。なお，環境変数 CBLAUTOCLEAR に YES 以外を指定した場合，入力されたデータは，入力前のデータに上書きされて格納されます。

（例）

CBLAUTOCLEAR に YES を指定した場合

入力前のフィールド：”ABCDE” ←"F"を入力

入力後のフィールド：”F ”

CBLAUTOCLEAR に YES 以外を指定した場合

入力前のフィールド：”ABCDE” ←"F"を入力

入力後のフィールド：”FBCDE”

- フィールドが選択状態のときにデータを入力すると，選択状態は解除されます。
- フィールドが選択状態で［BackSpace］キー，［Insert］キー，または［Delete］キーを押した場合，［End］キーが押されたものとして動作します。このとき，カーソルの選択状態は解除されます。

(b) 画面節（WINDOW SECTION）固有の動作

- 環境変数 CBLAUTOCLEAR に YES を指定した場合，ACCEPT 文で，データの入力待ちになる前の処理（指定した項目の行，カラム位置に表示されているデータをクリアする）を抑止します。

  なお，環境変数 CBLAUTOCLEAR に YES を指定しないで ACCEPT 文を実行した場合，入力待ちになる前に，ACCEPT 文で指定した項目の行，カラム位置に表示されているデータを，ACCEPT 文で指定した項目の属性（例えば，英数字項目のときは空白文字）でクリアします。環境変数

CBLAUTOCLEAR に YES を指定すると，ACCEPT 文で指定した項目の行，カラム位置に表示されているデータはクリアされません。

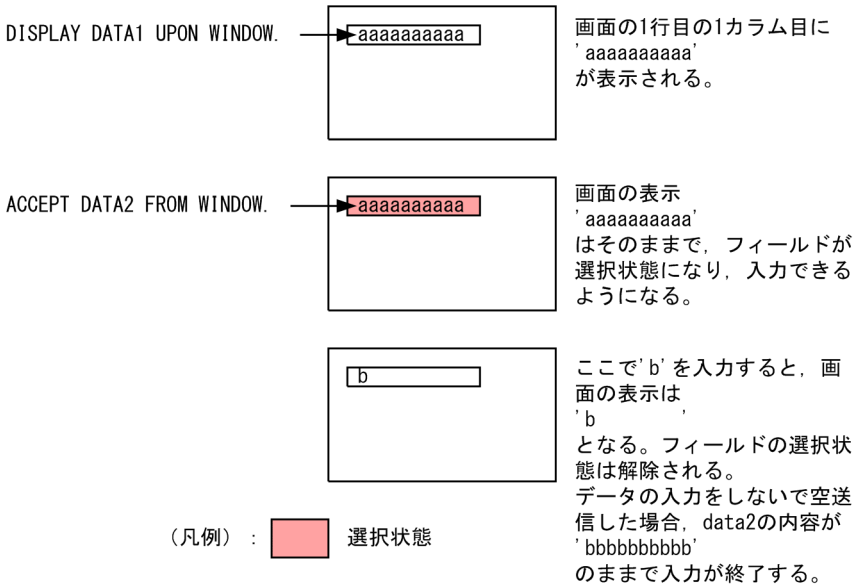
また，このとき，データの入力をしないで空送信しても，ACCEPT 文で指定した項目のデータの内容は変わりません。

- フィールドが選択状態で，[←] キーを押した場合，前方のフィールドにカーソルが移動します。
- フィールドが選択状態で，[→] キーを押した場合，後方のフィールドにカーソルが移動します。

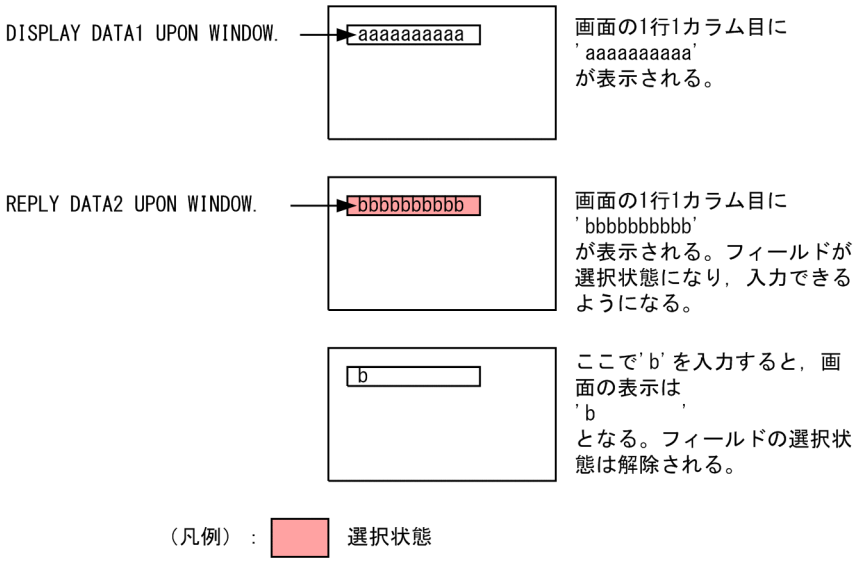
次に，画面節（WINDOW SECTION）で環境変数 CBLAUTOCLEAR に YES を指定した例を示します。

```
       WINDOW SECTION.
         01 GAMEN FIELD TYPE I-O.
           02 DATA1 PIC X(10) LINE 1 COLUMN 1
                             VALUE 'aaaaaaaaaa'.
           02 DATA2 PIC X(10) LINE 1 COLUMN 1
                             VALUE 'bbbbbbbbbb'.
           02 DATA3 PIC 9(10) LINE 1 COLUMN 1
                             VALUE 1234567890.
```

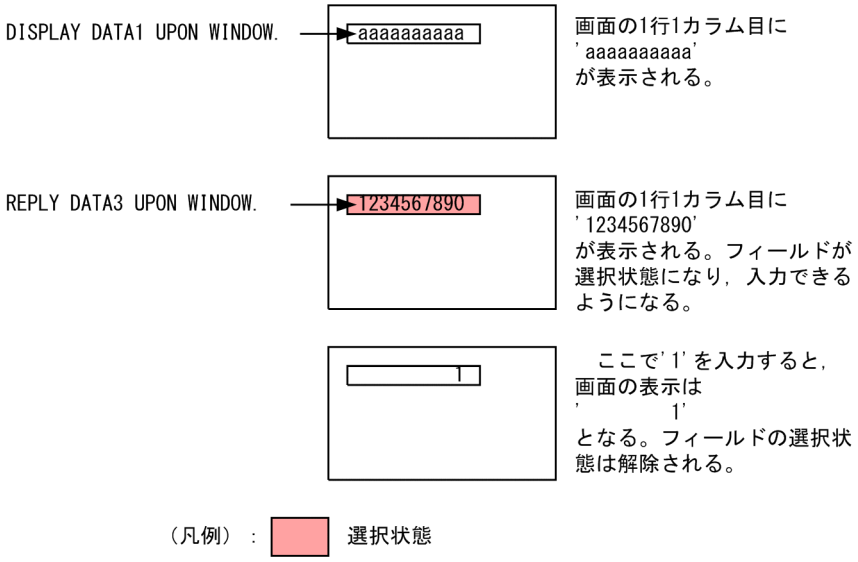
（例 1）



(例 2)



(例 3)



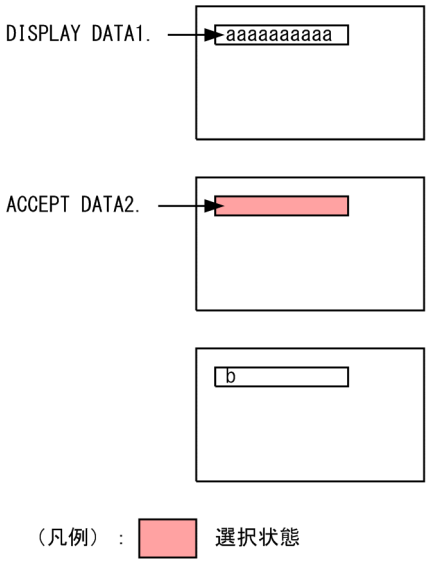
(c) 画面節（SCREEN SECTION）固有の動作

- フィールドが選択状態で，[←] キー，または [→] キーを押した場合，データはクリアされないで，選択状態が解除されます。

  次に，画面節（SCREEN SECTION）で環境変数 CBLAUTOCLEAR に YES を指定した例を示します。

```
       WORKING-STORAGE SECTION.
         01 AAA PIC X(10) VALUE 'aaaaaaaaaa'.
         01 BBB PIC X(10) VALUE 'bbbbbbbbbb'.
         01 CCC PIC 9(10) VALUE 1234567890.
       SCREEN SECTION.
         01 GAMEN.
           02 DATA1 PIC X(10) LINE 1 COLUMN 1 USING AAA.
           02 DATA2 PIC X(10) LINE 1 COLUMN 1 USING BBB.
           02 DATA3 PIC 9(10) LINE 1 COLUMN 1 USING CCC.
```

(例 1)

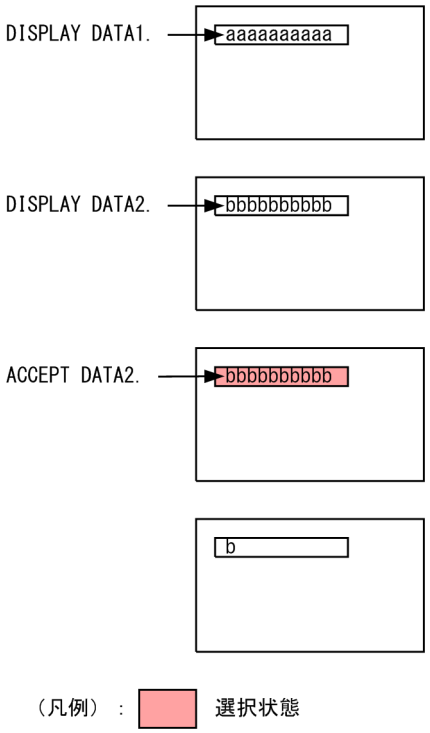


画面の1行1カラム目に
'aaaaaaaaaa'
が表示される。

画面上に
' '
が表示され，フィールドが選択状態になり，入力できるようになる。

ここで'b'を入力すると，画面の表示は
'b '
となる。フィールドの選択状態は解除される。
データの入力をしないで空送信した場合，data2の内容は
' '
となり，入力が終了する。

(例 2)



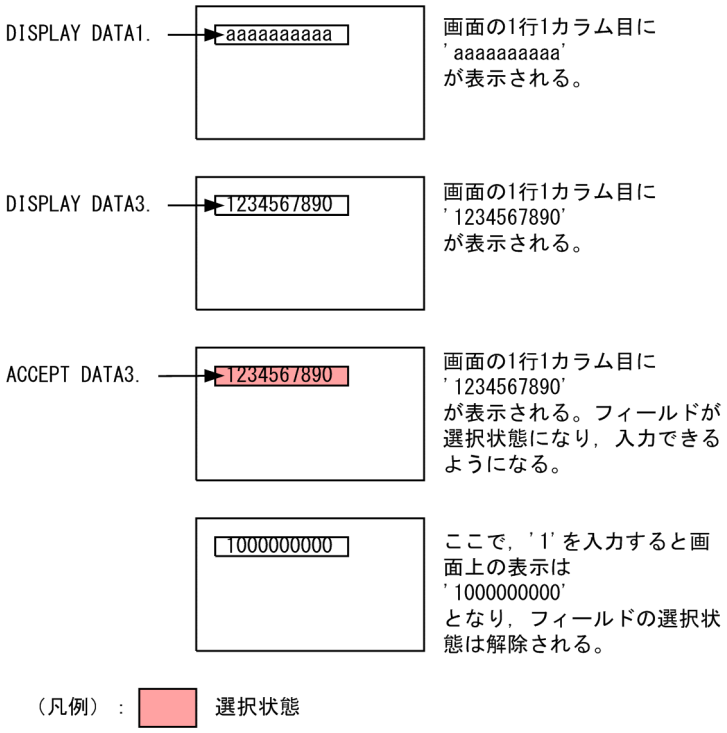
画面の1行1カラム目に
'aaaaaaaaaa'
が表示される。

画面の1行1カラム目に
'bbbbbbbbbb'
が表示される。

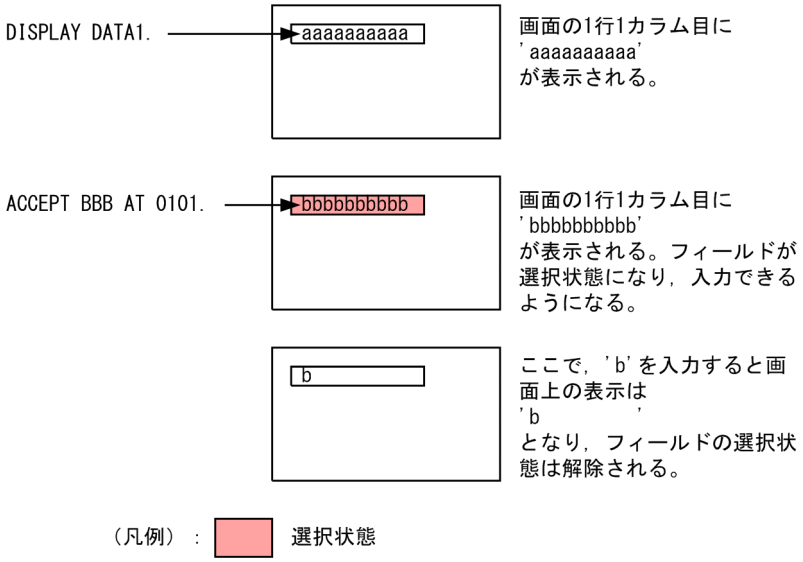
フィールドが選択状態になり，入力できるようになる。

ここで，'b'を入力すると
画面上の表示は
'b '
となり，フィールドの選択状態は解除される。

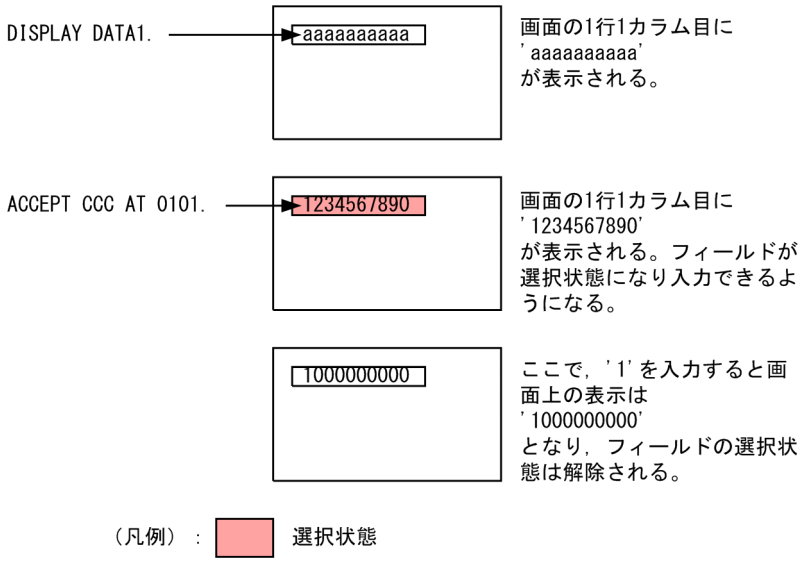
(例 3)


DISPLAY DATA1.
aaaaaaaaaa
画面の1行1カラム目に
'aaaaaaaaaa'
が表示される。
DISPLAY DATA3.
1234567890
画面の1行1カラム目に
'1234567890'
が表示される。
ACCEPT DATA3.
1234567890
画面の1行1カラム目に
'1234567890'
が表示される。フィールドが
選択状態になり，入力できる
ようになる。
1000000000
ここで，'1'を入力すると画
面上の表示は
'1000000000'
となり，フィールドの選択状
態は解除される。
（凡例）：
選択状態

(例 4)


DISPLAY DATA1.
aaaaaaaaaa
画面の1行1カラム目に
'aaaaaaaaaa'
が表示される。
ACCEPT BBB AT 0101.
bbbbbbbbbb
画面の1行1カラム目に
'bbbbbbbbbb'
が表示される。フィールドが
選択状態になり，入力できる
ようになる。
b
ここで，'b'を入力すると画
面上の表示は
'b '
となり，フィールドの選択状
態は解除される。
（凡例）：
選択状態

（例 5）



(4) CBLENTERCHK

ACCEPT 文，または REPLY 文の実行時，画面節（WINDOW SECTION）の ENTER-CHECK 句を指定した項目にデータの入力がなかったときに，WINDOW-STATUS 特殊レジスタへ値を設定したい場合に YES を指定します。YES を指定した場合，WINDOW-STATUS 特殊レジスタには「02」が設定されます。YES 以外の値を指定したときは，無効となります。

なお，ENTER-CHECK 句を指定した項目が複数ある場合，それらすべての項目に対してデータの入力がなかったときだけ，「02」が設定されます。

(5) CBLFEP

-JPN,Alnum オプションまたは-JPN,V3JPN オプション指定時，画面節（SCREEN SECTION および WINDOW SECTION）でカーソルを日本語項目に移動した場合，自動的に日本語入力に切り替えたいときに 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。

なお，この機能を有効にするには，実行支援の画面環境で「日本語入力のオン／オフを自動的に切り替える」という設定をしておく必要があります。実行支援の設定については，マニュアル「COBOL2002　操作ガイド」を参照してください。

(6) CBLIMEPOS

画面節（SCREEN SECTION および WINDOW SECTION）で，フィールドに日本語を直接入力したい場合に 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。

(7) CBLJCPOPENDKEY

JCPOPUP サービスルーチンの終了キーを追加したい場合に 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，「30.5.3　JCPOPUP」を参照してください。

(8)　CBLM7ENDKEY

MIOS7 で使用する終了キー（［I］～［VIII］キー）の互換キーを，変更したい場合に YES を指定します。YES を指定した場合，互換キーの組み合わせが［Alt］＋［F1］キー～［Alt］＋［F8］キーから，［Shift］＋［F1］キー～［Shift］＋［F8］キーに変更されます。

YES 以外の値を指定したときは，無効となります。

(9)　CBLNOCLOSE

ウィンドウのシステムメニュー，およびタイトルバーの［閉じる］ボタンを非活性 にしたいときに YES を指定します。これによって，COBOL プログラムの実行中にウィンドウを閉じて強制的にプログラムを中断できなくなります。

YES 以外の値を指定したときは，無効となります。

(10)　CBLONLYNUM

環境変数 CBLAUTOCLEAR に YES を指定した場合の入力動作の変更を，画面節（WINDOW SECTION）の数字項目，および数字編集項目だけに限定したい場合，YES を指定します。

環境変数 CBLAUTOCLEAR の機能については，「(3)　CBLAUTOCLEAR」を参照してください。

YES 以外の値を指定したときは，無効となります。

また，この環境変数は CBLAUTOCLEAR に YES が指定されていない場合，無効となります。

(11)　CBLOVERFLOW

画面節（WINDOW SECTION）でのデータ入力時，カーソルが自動的に移動する機能を抑止し，けたあふれチェックをしたい場合，YES を指定します。

CBLOVERFLOW に YES を指定すると，入力フィールドまたは更新フィールドの最後の文字が入力されても，カーソルが自動的に次のフィールドに移動しなくなります。また，最後の文字を入力した状態から続けて文字を入力すると，オーバフローのエラーメッセージが出力されます。

YES 以外の値を指定したときは，無効となります。

(12)　CBLSETFIELD

画面節（WINDOW SECTION）の SET 文でフィールドの属性を変更する場合，それまでに SET 文で変更した属性をクリアしたいときに 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。

（例）

CBLSETFIELD=YES の場合

SET FIELD AAA TO RED.　AAAAA　・AAAの表示が赤色になる。

SET FIELD AAA TO REVERSE.　AAAAA　・AAAの表示が元の色に戻り，反転表示となる。

CBLSETFIELD=YES 以外の場合

SET FIELD AAA TO RED.　AAAAA　・AAAの表示が赤色になる。

SET FIELD AAA TO REVERSE.　AAAAA　・AAAの表示が赤色のまま，反転表示となる。

(13)　CBLUNDERDOT

画面節（SCREEN SECTION および WINDOW SECTION）で，入力，および入出力フィールドにピリオド（.）を表示したい場合は，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。

(14)　CBLUPDOWNMOVE

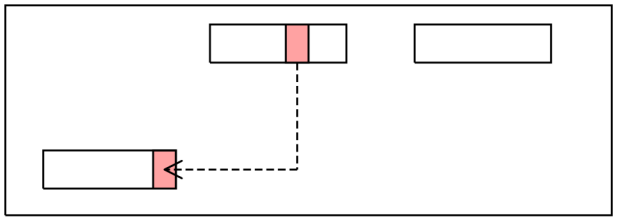
画面節（WINDOW SECTION）の ACCEPT 文，または REPLY 文実行時，［↑］キー，［↓］キーの動作を変更する環境変数です。

通常，この環境変数を指定していない場合，［↑］キー，［↓］キーを操作すると，現在のフィールドの前フィールド，または次フィールドにカーソルが位置づけられます。この環境変数を指定した場合，カーソルは下方向に位置する，入力，および入出力フィールドの先頭入力位置にカーソルが位置づけられます。YES 以外の値を指定したときは，無効となります。

環境変数 CBLUPDOWNMOVE を指定した場合，カーソルの動作例を次に示します。なお，ここで扱っているフィールドはすべて数字項目を扱う入力フィールドとします。

（例 1）

［↓］キーを押す

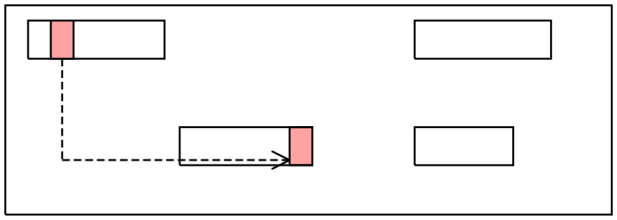


（凡例）　：カーソル

カーソルは下に向かって移動します。この場合，真下より左方向に入力フィールドがあるため，そのフィールドの先頭入力位置にカーソルが位置づけられます。

（例 2）

［↓］キーを押す

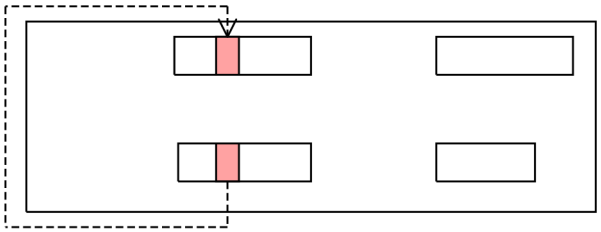


（凡例）　：カーソル

カーソルは下に向かって移動します。この場合，真下より左方向に入力フィールドがないため，左端から最初に位置する入力フィールドの先頭入力位置に，カーソルが位置づけられます。

（例 3）

［↓］キーを押す



（凡例）　：カーソル

現在，カーソルの位置より下方向に入力フィールドがないため，画面上部に位置する入力フィールドの位置に，カーソルが位置づけられます。

36.2.7　画面（XMAP）

(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。

36.2.8　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。

36.2.9　拡張機能

(1)　CBLOPS

MIOS7 COBOL85 互換のための環境変数です。CBLOPS に YES を指定すると，次に示す環境変数のすべてに YES が指定されたものとして動作します。

YES 以外の値を指定したときは，無効となります。

- CBLISAMDL
- CBLGDIWMSG
- CBLFEP
- CBLIMEPOS
- CBLENTERCHK
- CBLUNDERDOT
- CBLJCPOPENDKEY
- CBLUPDOWNMOVE
- CBLAUTOCLEAR

(2) CBLSQLCOMMOD

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。

(3) 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，「23.2.9　カーソルオプションの設定」を参照してください。

(4)　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 を変更したい場合に指定します。

詳細は，「23.2.11　動的 SQL の ODBC API 関数発行の変更」を参照してください。

(5)　CBLSQLLOGINTIMEOUT

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，「23.2.8　タイムアウト秒数の設定」を参照してください。

(6)　CBLSQLQUERYTIMEOUT

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，「23.2.8　タイムアウト秒数の設定」を参照してください。

(7)　CBLSQLROWCOUNT

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，「23.2.4　SQL 文のエラー処理」を参照してください。

(8)　CBLSQLSUPPRESSMSG

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，「23.2.4　SQL 文のエラー処理」を参照してください。

(9)　CBLSQLWMSG

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，「23.2.4　SQL 文のエラー処理」を参照してください。

36.2.10　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，「37.2　異常終了時要約情報リスト」を参照してください。

(2)　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，「37.3　データ領域ダンプリスト」を参照してください。

(3)　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「37.3.2　データ領域ダンプリストの出力先」を参照してください。

(4)　CBLEXCEPT

この環境変数に THROW 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 のデバッグ情報を出力して上位プログラムへ例外コードをスローします。詳細は，「37.7.3　COBOL 実行時ライブラリが検出する例外」を参照してください。

(5)　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，「37.4.2　整合性チェックの警告エラー出力」を参照してください。

(6) 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。

- GUI モードのテストデバッガ
- 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　操作ガイド」を参照してください。

なお，OpenTP1 から起動されるプログラムなど，プログラムが Windows のサービス機能から起動された場合，リモートデスクトップ接続などでリモートアクセスしたクライアントからではなく，サーバ上でデバッグ操作をしてください。

形式

```
CBLTDEXEC= {TD | CV 〔引数〕 | CN 〔引数〕 }
```

TD 引数

GUI モードの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。

(7) 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。

36.2.11 イベントログ

(1) CBLSYSLOG

イベントログファイル出力機能を有効にする場合，EVLOG を指定します。

詳細は，「10.5.2　イベントの出力」を参照してください。

(2) CBLSYSLOGLVL

イベントログファイル出力機能で，DISPLAY 文によるイベントの種類を指定します。

詳細は，「10.5.3　イベントの出力内容」の「(2)　イベントの種類の指定方法」を参照してください。

(3)　CBLSYSLOGSRV

イベントログファイル出力機能で，イベントログを出力するコンピュータ名を指定します。

詳細は，「10.5.4　イベントの出力先」の「(2)　出力先の指定方法」を参照してください。

36.2.12　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。

```
set CBLGCINTERVAL=32768
```

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。

```
set CBLGCSTART=65536
```

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。

37 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。

37.1　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 37–1　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。

37.2　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。

37.2.1　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

例外種別

例外の種別とその意味が出力されます。例外種別については，「37.7　COBOL が検出するハードウェア例外およびソフトウェア例外」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，「37.2.2 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，「37.2.3　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

     COBOL2002 (X) VV-RR     ***異常終了時要約情報リスト*** YYYY-MM-DD HH:MM:SS      } 1.
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                                       }
                                                                                     }
例外種別 = EXCEPTION_INT_DIVIDE_BY_ZERO                                              } 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                   行番号                                             }
  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                                                                    }
  CBL_BATCH=1                                                                        }
      :                                                                              }
  Lib=C:¥Program Files¥Hitachi¥COBOL2002¥lib;C:¥Program Files¥Hitachi¥COBOL2002      } 4.
    ¥lib¥SDK～                                                                       }
  Path=C:¥Program Files¥Hitachi¥COBOL2002¥bin;C:¥Program Files¥Hitachi¥COBOL200      }
    2¥bin¥SDK～                                                                      }
```

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 N.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。

37.2.2 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※

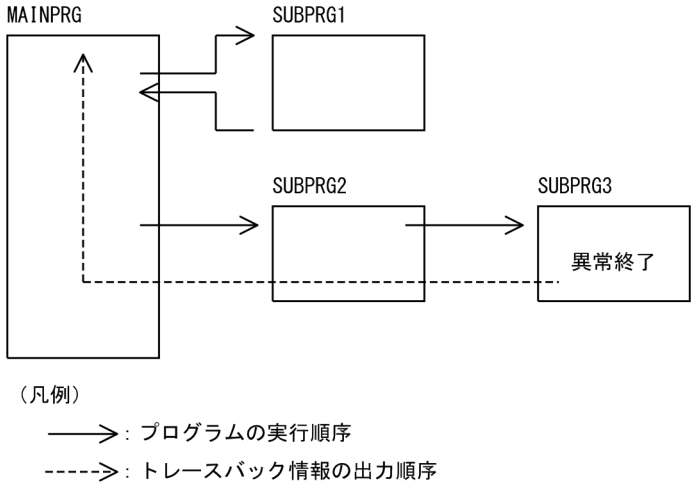
出力される行番号※は 1 プログラム当たり最大 240 個です。

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）



37.2.3　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。

- COBOL プログラム実行に関係する可能性のある実行時環境変数※
- システム環境変数 PATH，LIB

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。

37.2.4　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

```
set CBLABNLST=C:¥users¥abend.lst
```

規則

- ファイルは，ドライブ名からの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，GUI モードのときは COBOL2002 が出力するコンソール画面に，CUI モードのときは標準エラー（stderr）に，異常終了時要約情報リストが出力されます。
- 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。

37.2.5　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 37–2　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。

37.3　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，「33.5.13　リスト出力の設定」の「(1)　-SrcList オプション」および「付録 E.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。

37.3.1　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。

37.3.2　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

```
set CBLDDUMP=C:¥users¥data.dmp
```

(2) CBLDATADUMP サービスルーチン使用時

```
set CBLDATADUMPFILE=C:¥users¥data.dmp
```

ファイルは，ドライブ名からの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。

37.4　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。

37.4.1　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，「37.4.2　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。

37.4.2　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}
```

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。

- この環境変数の指定は，プログラムの呼び出し規約が stdcall 呼び出し規約プログラムの呼び出しでは使用できません。stdcall 呼び出し規約プログラムでは，引数を完全に一致させる必要があるため，引数が不一致の場合，Windows でリンクエラーとなります。

37.5　添字，指標の繰り返し回数，制御変数チェック

37.5.1　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，「33.5.9　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。

37.5.2　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。

37.6　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。

37.7　COBOL が検出するハードウェア例外およびソフトウェア例外

37.7.1　ハードウェア例外およびソフトウェア例外とは

例外は，プログラムの実行状態が異常になった場合に発生します。PC のアーキテクチャでは，命令実行中の異常事態を検知するために CPU 内部で発生させる割り込みのことを特に例外と呼び，CPU 外部の要因で引き起こされる割り込みとは区別します。この例外には CPU が検知するハードウェア例外とソフトウェアによって発行されるソフトウェア例外があります。

37.7.2　アプリケーションエラー

-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange オプションのどれも指定しないでコンパイルしたプログラムの実行でハードウェア例外やソフトウェア例外が発生した場合，OS がアプリケーションエラーのダイアログボックスを表示します。代表的なアプリケーションエラーと対処法を「表 37–3　代表的なアプリケーションエラー」に示します。

アプリケーションエラーが発生した場合は，次の手順で発生している例外種別および発生個所を調べてください。

1. -DebugInf または-DebugInf,Trace オプションを指定して再度コンパイルと実行をする
2. 異常終了時要約情報リストで発生している例外種別および発生個所を調べる

発生個所が CALL 文を指している場合は，その呼び出し先プログラムも調査の対象としてください。

表 37–3　代表的なアプリケーションエラー

例外名称（例外コード）	意味と対処方法
アクセス違反 （0xc0000005）	（原因）アクセスが許されていないメモリに読み書きしようとした。 （対処）このエラーは主に次の要因で発生する。これに該当しないかを確認する。 1. 呼び出し元プログラムと呼び出し先プログラムで，引数の属性，長さ，または数が一致していない 2. 表操作で使用する添字または指標の値が OCCURS で定義した範囲を超えた 3. 部分参照でデータ項目の領域を超えて参照した 4. アドレス名に不正なアドレスが入っている状況で，ADDRESSED 句付きで定義されたデータ名を参照した 5. 可変長項目の DEPENDING ON に指定されたデータ項目に不当な値が入っている状態で，その可変長項目，またはその従属項目を参照した 1.～3.の場合，-DebugRange を指定してコンパイル・実行することで，そのエラーであることおよび発生個所を特定できる。

例外名称（例外コード）	意味と対処方法
ゼロ除算 （0xc0000094）	（原因）ゼロで除算しようとした（被除数，除数ともに浮動小数点データでない場合）。 （対処）ゼロによる除算が発生しないようにプログラムを変更する。
スタックオーバフロー （0xc00000fd）	（原因）プログラムが使用するスタックサイズが，プログラムで許されたスタックの上限を超えた。再帰属性プログラム，メソッド，および利用者定義関数で，次の場合に発生しやすくなる。 • 局所場所節のデータ定義の合計サイズが大きい • 大量に再帰呼び出しがある • 連絡節のデータ定義の合計サイズが大きい • BY CONTENT，BY VALUE が仮定される関数の引数に大きなデータ項目がある （対処）プログラムのスタックの上限は，-STACK オプションで変更できる。このオプションで変更してから再リンクして，スタックの上限を変更する。-STACK オプションについては，「38.1.3　オプション」を参照のこと。 COBOL プログラムが使用するスタック領域については，「31.3　COBOL プログラムが使用するスタック領域」を参照のこと。

37.7.3　COBOL 実行時ライブラリが検出する例外

(1)　例外種別の出力

-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プログラムを実行した場合に例外が発生すると，COBOL がこれを検知し，実行時メッセージ，および異常終了時要約情報リストで例外種別を出力します。この情報によって，どのような例外が発生したか，および例外がどの実行文で発生しているかを把握できます。

実行時メッセージ，および異常終了時要約情報リストに表示される例外種別とその意味を次に示します。

表 37-4　実行時メッセージ，および異常終了時要約情報リストに表示される例外種別

例外種別	意味
RUNTIME ERROR	COBOL 実行時エラーが発生した。
CBLABN	CBLABN サービスルーチンが呼び出された。
EXCEPTION_ACCESS_VIOLATION	「表 37-3　代表的なアプリケーションエラー」の「アクセス違反」を参照
EXCEPTION_INT_DIVIDE_BY_ZERO	「表 37-3　代表的なアプリケーションエラー」の「ゼロ除算」を参照
EXCEPTION_INT_OVERFLOW	整数演算の結果，最上位ビットのキャリーアウトが発生した。
EXCEPTION_FLT_DENORMAL_OPERAND	浮動小数点演算で使用している作用対象に異常があり，その値が小さ過ぎるため，標準の浮動小数点値として表せない。

例外種別	意味
EXCEPTION_FLT_DIVIDE_BY_ZERO	浮動小数点値を浮動小数点値 0 で割ろうとした。
EXCEPTION_FLT_INEXACT_RESULT	浮動小数点演算の結果を 10 進小数として正確に表せない。
EXCEPTION_FLT_OVERFLOW	浮動小数点演算の指数の値が，対応する型の上限値を超えている。
EXCEPTION_FLT_UNDERFLOW	浮動小数点演算の指数の値が，対応する型の下限値を超えている。
EXCEPTION_FLT_STACK_CHECK	浮動小数点演算の結果，スタックのオーバフローまたはアンダーフローが発生した。
EXCEPTION_FLT_INVALID_OPERATION	この表にないすべての浮動小数点例外を表す。
EXCEPTION_PRIV_INSTRUCTION	現在のマシンモードでは実行できない演算命令を実行しようとした。
STATUS_NONCONTINUABLE_EXCEPTION	実行を続行できない例外の発生後，実行を続行しようとした。

(2) 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。

ただし，環境変数 CBLEXCEPT=THROW を指定したときはシステムが返す値が設定されます。

(3) 例外発生時の動作を指定する実行時環境変数 CBLEXCEPT

形式

```
CBLEXCEPT=THROW
```

規則

この環境変数に THROW 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例外が発生した場合，例外コードを上位プログラムにスローします。

このとき，異常終了時要約情報リストを出力し，終了方法は上位の制御プログラムの終了処理に依存します。

例えば，デバッグ環境を構築できない環境でシステム例外発生時のデバッグをする場合に指定します。

この環境変数を指定したときの動作を次に示します。

- COBOL プログラムがメインでないアプリケーションや GUI プログラムでは，この環境変数を指定しても，有効となりません。
- CBLDBGINF サービスルーチンを呼び出した場合は，例外コードを上位プログラムにスローしません。
- THROW を指定した場合，ハードウェア例外やソフトウェア例外の発生により異常終了したとき，Windows 問題レポートの設定などによって，クラッシュダンプが出力される場合があります。

  クラッシュダンプからシステム例外発生時点のスタックトレースなどを参照するために有用なコンテキスト情報のアドレスが実行時メッセージ中に出力されます。

また，上位プログラムのユーザープログラムに例外処理があれば，そちらに制御が移ります。

なお，ミドルソフト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。

- 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。

37.7.4　注意事項

- COBOL プログラムがメインでないアプリケーションでは，-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange を指定しても，ハードウェア例外およびソフトウェア例外の発生時にアプリケーションエラーになり，異常終了時要約情報リストは出力されません。
- COBOL 以外のプログラムで上記の例外が発生した場合，異常終了時の結果が保証されないときがあります。詳細については，「19.1　C 言語との連携」を参照してください。
- -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 主プログラムでスタックオーバフローが発生した場合は，-DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，または-DebugRange オプションの有無に関係なく，アプリケーションエラーとなります。
- -DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange を指定してコンパイルした COBOL プログラムに 1 回以上制御が渡ったあとでなければ，異常終了時要約情報リストは出力されません。

37.8　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 37–5　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

37.9　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，GUI モードとバッチモードの二つの方法があります。

GUI モードでは，ウィンドウに表示されたプログラムの動きを確認しながら，マウスやキーボードの操作によって，対話的にテストしたり，デバッグしたりできます。TD コマンドも入力できます。

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。コマンドプロンプトに，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　操作ガイド」を参照してください。

37.9.1　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　操作ガイド」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。

37.9.2　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。

37.10　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。

カバレージには，GUI モード，バッチモードの二つの方法があります。

GUI モードでは，ウィンドウのメニューからカバレージの機能を操作できます。

バッチモードでは，コマンドプロンプトから入力する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　操作ガイド」を参照してください。

37.10.1　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。

37.10.2　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。

38 リンカ，ライブラリ管理ツール，リソースコンパイラ

この章では，リンカ，ライブラリ管理ツール，リソースコンパイラの使い方とモジュール定義ファイルの作成方法について説明します。この章に記載していない各コマンドのオプションは，COBOL2002 のコンパイルおよび実行での動作を保証しません。

この章は，Microsoft が提供するソフトウェアの仕様記述を，日立製作所が翻訳し，修正または変更を行ったものです。それらの翻訳，修正および変更に関して，Microsoft は，レビューも承認も行っていませんし，変更内容に責任を負うものでもありません。また，翻訳，修正，変更に伴って誤りが含まれたとしても Microsoft は責任を負いません。

38.1　リンカ

38.1.1　機能の概要

リンカ（LINK コマンド）は，複数のオブジェクトファイル，ライブラリファイル，リソースファイルを連結して，Windows 上で動作する実行可能ファイル，DLL を作成するものです。

38.1.2　コマンドラインの形式

LINK コマンドの形式を次に示します。

形式

```
LINK OBJファイル名 〔LIBファイル名〕 〔オプション〕
```

OBJ ファイル名

オブジェクトファイル（.obj），およびリソースファイル（.res）の並びを指定します。ただし，リソースファイルの指定は任意です。

LIB ファイル名

ライブラリファイルの並びを指定します。

オプション

リンカオプションの並びを指定します。リンカオプションについては，「38.1.3　オプション」を参照してください。

指定規則

- コマンドライン上には，コマンド名以降であればどこにでもコマンドファイルを指定できます。ファイル名の先頭には@を付けて指定します。コマンドファイルを指定すると，ファイルの中身がコマンドライン上に展開され，コマンドの一部となります。

（例）

```
LINK @OBJECT.lst -MAP @LIBRARY.lst -OUT:TEST.exe
```

- OBJ ファイル名，LIB ファイル名，リンカオプションの指定順序は任意です。ただし，OBJ ファイルは指定した順番に結合されます。
- コマンドラインの区切り記号には，空白およびタブが使用できます。コマンドファイル内では，空白，タブ，セミコロン（;）および改行文字が使用できます。セミコロン（;）から改行文字までの指定は無効となり，コマンドファイル内にコメントを記述するために使用します。
- OBJ ファイル名，LIB ファイル名にはワイルドカード（*）が使用できます。
- ファイル名は，必ず拡張子を付けて指定します。

- リンカオプションは英大文字，英小文字のどちらで指定しても同じ扱いとなります。また，ハイフン（-）の代わりにスラント（/）で始めることもできます。
- 背反するオプションを指定した場合の動作は，各オプションの詳細を参照してください。

38.1.3 オプション

リンカ（LINK コマンド）で使用できるオプションを次に示します。

表 38–1 リンカオプション一覧

オプション名	機能	参照先	Windows(x86) COBOL2002	Windows(x64) COBOL2002
-ALLOWBIND	DLL をバインドできないことを指定します	(1) -ALLOWBIND:NO	○	○
-DEF	モジュール定義ファイル(.def)をリンカ（LINK コマンド）に渡します	(2) -DEF:モジュール定義ファイル名	○	○
-DEFAULTLIB	外部参照を解決するときに指定したライブラリを検索します	(3) -DEFAULTLIB:ライブラリ名	○	○
-DLL	DLL を作成します	(4) -DLL	○	○
-ENTRY	開始アドレスを設定します	(5) -ENTRY:シンボル名	○	○
-EXPORT	関数をエクスポートします	(6) -EXPORT:名前［,@順序番号［,NONAME]］	○	○
-FORCE	未解決または複数定義のシンボルがある場合でもリンクを強制的に終了させ，有効な実行可能ファイル DLL ファイルを生成します	(7) -FORCE［:｛MULTIPLE｜UNRESOLVED｝］	○	○
-HEAP	ヒープサイズをバイト単位で設定します	(8) -HEAP:仮想メモリサイズ［,実メモリサイズ］	○	○
-IMPLIB	デフォルトのインポートライブラリ名をオーバーライドします	(9) -IMPLIB:インポートライブラリ名	○	○
-INCLUDE	シンボルを明示的に参照します	(10) -INCLUDE:シンボル名	○	○
-LARGEADDRESSAWARE	アプリケーションが 2 ギガバイトを超えるアドレスをサポートしていることを指定します	(11) -LARGEADDRESSAWARE	○	○
-LIBPATH	環境変数 LIB に先行して検索したいライブラリパスを指定します	(12) -LIBPATH:ライブラリのパス	○	○
-MANIFEST	リンカがマニフェストファイルを生成することを指定します。	(13) -MANIFEST［:EMBED［,ID=リソース ID]］	○	○

オプション名	機能	参照先	Windows(x86) COBOL2002	Windows(x64) COBOL2002
-MANIFESTUAC	マニフェストファイルに組み込む UAC 情報を指定します。	(14)　-MANIFESTUAC〔:{NO ｜ UAC フラグメント}〕	○	○
-MAP	マップファイルを作成します	(15)　-MAP〔:ファイル名〕	○	○
-MAPINFO	指定した情報をマップファイルに格納します	(16)　-MAPINFO:EXPORTS	○	○
-NODEFAULTLIB	外部参照を解決するときにすべてのライブラリ，または，指定したデフォルトのライブラリを無視します	(17)　-NODEFAULTLIB〔:ライブラリ名〕	○	○
-ORDER	指定された順序で COMDAT をイメージに取り込みます	(18)　-ORDER:@ファイル名	○	○
-OUT	出力ファイル名を指定します	(19)　-OUT:出力ファイル名	○	○
-STACK	スタックサイズをバイト単位で設定します	(20)　-STACK:仮想メモリサイズ〔,実メモリサイズ〕	○	○
-SUBSYSTEM	OS に対して，実行可能ファイル(.exe)の実行方法を指定します	(21)　-SUBSYSTEM:{CONSOLE ｜ WINDOWS}	○	○
-VERBOSE	リンカ（LINK コマンド）の進行状況メッセージを出力します	(22)　-VERBOSE〔:{LIB ｜ UNUSEDLIBS}〕	○	○
-VERSION	バージョン番号を割り当てます	(23)　-VERSION:バージョン番号〔.リビジョン番号〕	○	○
-?	オプションの概略一覧を表示します	(24)　-?	○	○

（凡例）

○：オプションあり

(1)　-ALLOWBIND:NO

対象の DLL を BIND コマンドにバインドさせないための情報を，DLL ヘッダに設定するためのオプションです。

(2)　-DEF:モジュール定義ファイル名

モジュール定義ファイル（.def）を指定するためのオプションです。ファイル名は拡張子を付けて指定します。-DEF オプションは一つだけ指定できます。モジュール定義ファイルについては，「38.4　モジュール定義ファイル」を参照してください。

(3) -DEFAULTLIB:ライブラリ名

コマンドライン上のオブジェクトファイルやライブラリだけで解決できなかった外部参照の検索先ライブラリを指定するためのオプションです。同じライブラリ名が-NODEFAULTLIB オプションにも指定されている場合は，-NODEFAULTLIB オプションが優先されます。

(4) -DLL

ダイナミックリンクライブラリ（DLL）を作成するためのオプションです。

(5) -ENTRY:シンボル名

実行モジュールの入口点を指定するためのオプションです。シンボル名には OS が最初に制御を渡すプログラム（スタートアッププログラム）の名前を指定します。

OS が最初に呼び出すプログラムは，次の呼び出し規約でなければなりません。

Windows(x86) COBOL2002 の場合

- stdcall

Windows(x64) COBOL2002 の場合

- fastcall

例えば，アセンブラプログラムが主プログラムの場合などで，ユーザ自身がスタートアッププログラムを作成したときに，-ENTRY オプションを指定します。

注意事項

COBOL プログラムが主プログラムとなるアプリケーションを作成する場合，または COBOL の DLL アプリケーションを作成する場合は，次のように入口点を指定してください。

- コンソールアプリケーションの場合

  ```
  -ENTRY:mainCRTStartup
  ```

- WINDOWS アプリケーションの場合

  ```
  -ENTRY:WinMainCRTStartup
  ```

- DLL の場合

Windows(x86) COBOL2002 の場合

```
-ENTRY:_DllMainCRTStartup@12
```

Windows(x64) COBOL2002 の場合

```
-ENTRY:_DllMainCRTStartup
```

ただし，これらの指定は，LINK コマンドで直接 COBOL プログラムをリンクする場合だけ必要になります。ccbl2002 コマンド，および開発マネージャを使用する場合は，リンカオプションでの入口点の指定は必要ありません。

また，LINK コマンドで COBOL プログラムをリンクする場合は，-ENTRY オプションの代わりに，-SUBSYSTEM オプションでも入口点を指定できます。指定形式を次に示します。

- コンソールアプリケーションの場合

  -SUBSYSTEM:CONSOLE

- WINDOWS アプリケーションの場合

  -SUBSYSTEM:WINDOWS

これらのオプションを指定すると，指定に応じて適切な入口点が仮定されます。

(6) -EXPORT:名前〔,@順序番号〔,NONAME〕〕

名前をエクスポートするためのオプションです。このオプションはモジュール定義ファイルの EXPORTS 文と同じです。

@順序番号（1～65,535）を指定すると，名前ではなく番号で関数を呼び出せます。この呼び出しの方がより速く関数を呼び出せます。

順序番号のあとに NONAME を付けると，名前はエクスポートされないため，順序番号による呼び出しだけができるようになります。

(7) -FORCE〔:{MULTIPLE | UNRESOLVED}〕

外部シンボルの未解決エラーや外部シンボルの多重定義エラーが発生しても，強制的に実行可能ファイルを生成させるためのオプションです。

MULTIPLE を指定すると，外部シンボルの定義が重複していても出力ファイルを作成します。

UNRESOLVED を指定すると，未定義の外部シンボルがあっても出力ファイルを作成します。

引数を省略すると，MULTIPLE と UNRESOLVED の両方が仮定されます。

(8) -HEAP:仮想メモリサイズ〔,実メモリサイズ〕

ローカルヒープのサイズを変更するためのオプションです。仮想メモリ上の上限サイズと，1 回で割り当てられる実メモリの上限サイズをバイト単位で指定します。

仮想メモリサイズのデフォルト値は 1 メガバイトです。また，指定した値は 4 バイト単位に切り上げられます。このオプションは，モジュール定義ファイルの HEAPSIZE 文と等価です。

(9) -IMPLIB:インポートライブラリ名

リンカ（LINK コマンド）の出力するインポートライブラリの名称を指定するためのオプションです。省略時は，主出力ファイルである実行可能ファイル名の拡張子を.lib に変えて作成されます。

(10) -INCLUDE:シンボル名

指定したシンボルを含むオブジェクトファイルを強制的に取り込むためのオプションです。

(11) -LARGEADDRESSAWARE

生成される実行可能ファイルが，2 ギガバイトを超えるアドレスを取り扱えるようにするためのオプションです。Windows(x64) COBOL2002 の場合は，このオプションは既定で有効になります。

(12) -LIBPATH:ライブラリのパス

環境変数 LIB に先行して検索したいライブラリパスを指定するためのオプションです。入力する-LIBPATH オプションごとに一つのフォルダを指定します。複数のフォルダを指定する場合は，フォルダごとに-LIBPATH オプションを指定します。リンカ（LINK コマンド）は，指定された順にフォルダを検索します。

(13) -MANIFEST〔:EMBED〔,ID=リソース ID〕〕

リンカがマニフェストファイルを生成することを指定するためのオプションです。

EMBED を指定すると，リンク時に実行可能ファイルまたは DLL ファイルにマニフェストファイルを埋め込むことができます。リソース ID には，実行可能ファイルの場合は 1 を，DLL ファイルの場合は 2 を指定します。ID パラメタを省略すると，-DLL オプションが指定されている場合は 2 が，それ以外の場合は 1 が既定値として設定されます。

-MANIFEST オプションを指定して，-MANIFESTUAC オプションと-DLL オプションのどちらも指定していない場合，ユーザアカウント制御（UAC）レベルが asInvoker に設定されているマニフェストに既定の UAC フラグメントが挿入されます。ユーザアカウント制御（UAC）レベルの詳細については，「(14) -MANIFESTUAC〔:{NO | UAC フラグメント}〕」を参照してください。

-MANIFEST オプションを指定すると，マニフェストファイルの名前は，出力ファイルの名前と同じになり，ファイル名に.manifest が追加されます。例えば，出力ファイルの名前が MyFile.exe の場合，マニフェストファイル名は MyFile.exe.manifest になります。

(14) -MANIFESTUAC〔:{NO | UAC フラグメント}〕

ユーザアカウント制御（UAC）情報をプログラムマニフェストに組み込むかどうかを指定するオプションです。

-MANIFESTUAC

既定の UAC 情報をプログラムマニフェストに組み込みます。

既定値として，level 値は asInvoker が，uiAccess 値は false が設定されます。

-MANIFESTUAC:NO

UAC 情報をプログラムマニフェストに組み込みません。

-MANIFESTUAC:UAC フラグメント

UAC フラグメントを指定すると，指定した値の UAC 情報をプログラムマニフェストに組み込みます。

UAC フラグメントは次の形式で指定します。

```
-MANIFESTUAC:level= {'asInvoker'|'highestAvailable'|'requireAdministrator'}
```

値	意味
asInvoker［既定値］	アプリケーションを開始したプロセスと同じアクセス許可レベルで実行します。［管理者として実行］を選択すると，アプリケーションをより高いアクセス許可レベルに昇格させることができます。
highestAvailable	可能な限り高いアクセス許可レベルで実行します。使用できる最も高いアクセス許可レベルが，開始したプロセスのレベルより高い場合は，資格情報の入力が求められます。
requireAdministrator	管理者のアクセス許可レベルで実行します。アプリケーションを開始するユーザは管理者グループのメンバである必要があります。開始したプロセスが管理者のアクセス許可レベルで実行されない場合は，資格情報の入力が求められます。

```
-MANIFESTUAC:uiAccess= {'true'|'false'}
```

アプリケーションがユーザインタフェースの保護レベルをバイパスし，入力をデスクトップ上のアクセス許可レベルの高いウィンドウ（例えば，オンスクリーンキーボード）にアクセスできるようにする場合は true，それ以外の場合は false を指定します。既定値は false です。true の設定は，ユーザインタフェースのユーザ補助アプリケーションでだけで行います。

また，次の形式で指定すると，level 値および uiAccess 値の指定を 1 ステップで行うことができます。

```
-MANIFESTUAC:"level= {'asInvoker'|'highestAvailable'|'requireAdministrator'}  uiAccess=
 {'true'|'false'} "
```

-MANIFESTUAC オプションを複数指定した場合は，最後に入力したオプションが優先されます。

(15) -MAP〔:ファイル名〕

マップファイルを生成するためのオプションです。ファイル名を指定しなかった場合は，主出力ファイルである実行可能ファイル名の拡張子を.map に変えて作成されます。

マップファイルは，プログラムやデータが実行可能プログラムや DLL 中のどこに配置されたか，それらはどのライブラリのどのオブジェクトファイルから取り込まれたものかを示す情報を含んでいるリストファイルです。外部シンボル単位にそれらの情報が出力されます。さらに，-MAPINFO:EXPORTS オプションを指定することで，マップファイルの中にエクスポート情報を加えることもできます。

(16) -MAPINFO:EXPORTS

-MAP オプションで生成されるマップファイルの中にエクスポート情報を加えるためのオプションです。

(17) -NODEFAULTLIB〔:ライブラリ名〕

指定されたライブラリをデフォルトライブラリの検索対象から除くためのオプションです。ライブラリ名を指定しなかった場合は，すべてのデフォルトライブラリが検索対象から除かれます。

複数のデフォルトライブラリを検索対象から除外するためには，除外するライブラリごとに-NODEFAULTLIB オプションを繰り返し指定します。

(18) -ORDER:@ファイル名

実行可能ファイル内に配置する関数の順序を指定するためのオプションです。@ファイルには，1 行に一つの関数の割合で関数名を記述します。コマンドファイルと同様に，行内の有効カラムの終わりをセミコロン（;）で指定でき，その後ろにコメントを書けます。このオプションを使用するには，オブジェクトファイル内のすべての関数に，必要な COMDAT レコードが生成されていることが必要です。必要な COMDAT レコードがない場合は，オプションは無効になり，関数は参照された順番に配置されます。

(19) -OUT:出力ファイル名

主出力である実行可能ファイル，DLL ファイルの名称を指定するためのオプションです。このオプションの指定がなく，モジュール定義ファイルにも出力ファイルが指定されていなければ，最初の OBJ ファイルの拡張子を.exe，または.dll に変えた名前で出力ファイルが作成されます。

(20) -STACK:仮想メモリサイズ〔,実メモリサイズ〕

プログラムのスタックのサイズを変更するためのオプションです。仮想メモリ上の上限サイズと，1 回で割り当てられる実メモリの上限サイズをバイト単位で指定します。

仮想メモリサイズのデフォルト値は 1 メガバイト，実メモリサイズのデフォルト値は 4 キロバイトに設定されます。

このオプションは，モジュール定義ファイルの STACKSIZE 文と等価です。

(21) -SUBSYSTEM:｛CONSOLE ｜ WINDOWS｝

OS に，実行可能ファイル（.exe）の実行方法を指定するためのオプションです。

CONSOLE を指定すると，コンソールを必要とするプログラム（コンソールプログラム）を作成します。コンソールプログラムには，OS からコンソールが与えられます。

WINDOWS を指定すると，コンソールを必要としないプログラム（ウィンドウプログラム）を作成します。

-SUBSYSTEM オプションを指定すると，-ENTRY オプションを指定しなくても，実行可能プログラムの入口点に COBOL で使用できるスタートアッププログラムが設定されます。

なお，ccbl2002 コマンド，および開発マネージャを使用する場合は，-SUBSYSTEM オプションを指定しないでください。

(22) -VERBOSE〔:{LIB | UNUSEDLIBS}〕

リンクプロセスの情報を詳しく表示するためのオプションです。

リンクされるオブジェクトファイルの名称，使用するライブラリの名称，リンクの各フェーズなどが表示されます。

LIB を指定すると，検索されたライブラリだけの進行状況メッセージが表示されます。表示される情報は，ライブラリ検索の進行状況，各ライブラリおよびオブジェクト名(絶対パス)，そのライブラリで解決されたシンボル，ならびにそのシンボルを参照しているオブジェクトの一覧です。

UNUSEDLIBS を指定すると，イメージの作成時に使用されていないライブラリファイルに関する情報が表示されます。

(23) -VERSION:バージョン番号〔.リビジョン番号〕

プログラムのバージョンを主出力ファイルに設定するためのオプションです。バージョン番号，リビジョン番号とも，0～65,535 の整数を指定します。デフォルトは 0.0 です。

(24) -?

オプションの概略一覧を表示するオプションです。

38.1.4 リソースファイル

リンカ（LINK コマンド）は，リソースファイル（.res）をオブジェクトファイルの形式に変換し，オブジェクトファイルと同様に扱います。したがって，リソースコンパイラ（RC コマンド）の出力したリソースファイルをオブジェクトファイルと同様に指定すれば，リソースを実行可能ファイルにリンクできます。

38.1.5 ダイナミックリンク機能

ダイナミックリンク機能とは，未解決の名前を定義している実行可能ファイルを実行時に動的にロードして名前を解決する機能です。この動的にロードしてリンクできるファイルを DLL（ダイナミックリンクライブラリ）といいます。メモリ中にロードされた DLL は複数のプロセスで共用できるので，メモリを有効活用できます。

ここでは，ダイナミックリンクを実現する上で必要なエクスポート，インポートの機能について説明します。

(1) エクスポート

エクスポートとは，名前をほかの実行可能ファイルから参照できるように公開することです。リンカ（LINK コマンド）は，その名前の定義を含む実行可能ファイルのエクスポートテーブルにその名前を登録します。

名前のエクスポートは，モジュール定義ファイルの EXPORTS 文，またはリンカ（LINK コマンド）の-EXPORT オプションでできます。ただし，COBOL2002 では，DLL 関数に対してコンパイラがその名前を自動的にエクスポートするため，EXPORTS 文または-EXPORT オプションで明示的にエクスポートする必要はありません。

(2) インポート

インポートとは，プログラムで参照する名前が同一実行可能ファイル内で定義されているものではなく，ほかの実行可能ファイルでエクスポートされている名前であることを宣言することです。

名前のインポートは，リンカ（LINK コマンド）の-INCLUDE オプションで行います。

(3) ダイナミックリンク動作

インポートされた名前を参照するプログラムの実行時，その名前をエクスポートしている DLL がロードされます。このときローダで，エクスポート情報とインポート情報によって名前が解決されます。

(4) エクスポートファイル

エクスポートファイル（.exp）は，ライブラリ管理ツール（LIB コマンド）とリンカ（LINK コマンド）の間のインタフェースとして使用されるファイルです。COBOL2002 を使ってプログラムを作成する場合は，エクスポートファイルを直接使用することはありません。

(5) インポートライブラリ

インポートライブラリ（.lib）は，インポートの手間を軽減するためのものです。リンク時に，ほかのライブラリと同様にインポートライブラリを指定することで，外部名をインポートできます。

インポートライブラリは，DLL の作成時に，DLL ファイル名の拡張子を.lib に変更した名称で生成されます。

38.1.6 ライブラリの検索

インポート宣言された名前以外の外部参照が，指定したオブジェクトファイルだけでは解決できない場合，ライブラリが検索されます。検索の手順を次に示します。

1. コマンドライン上のライブラリを左から順に取り出し，それぞれ次の操作をします。

(a) ライブラリが絶対パスで指定されている場合

そのライブラリに該当する外部名があるかどうかをチェックします。あれば検索を終了します。

(b) ライブラリ名だけが指定されている場合

ライブラリのパス（-LIBPATH オプションで指定されたパス，および環境変数 LIB に指定されたパス）を順に検索して該当するライブラリを見つけます。そして，そのライブラリに外部名があるかどうかをチェックします。あれば検索を終了します。ライブラリのパスは，まず-LIBPATH オプションに指定されたパスをオプションの指定順に検索し，そのあとに環境変数 LIB に指定されたパスを順に検索します。

2. 1.の操作で見つからない場合，-DEFAULTLIB オプションに指定されたデフォルトライブラリと，オブジェクトファイル内で定義されているデフォルトライブラリに対して(a)(b)の操作をします。
3. 1.2.の操作で見つからない場合は，外部参照未解決のためエラーとします。

注意事項

リンカ（LINK コマンド）では，外部名はすべて英大文字と英小文字が区別されます。

38.1.7 戻り値

リンカ（LINK コマンド）の戻り値は次のとおりです。

戻り値	意味
0	正常終了した。
0 以外	エラーが発生した。リンク終了の原因となったエラーの番号が，戻り値として返される。

38.2　ライブラリ管理ツール

38.2.1　機能の概要

ライブラリ管理ツール（LIB コマンド）は，Windows のライブラリとそのライブラリ内のオブジェクトファイルを管理するためのツールで，次の機能を持ちます。

- ライブラリの新規作成
- ライブラリ内のオブジェクトファイルの追加・変更
- 複数のライブラリの結合
- ライブラリ内のオブジェクトファイルの削除
- ライブラリからのオブジェクトファイルの取り出し
- インポートライブラリの生成

38.2.2　コマンドラインの形式

LIB コマンドの形式を次に示します。

形式

```
LIB 〔OBJファイル名〕 〔LIBファイル名〕 〔オプション〕
```

OBJ ファイル名

オブジェクトファイル（.obj）の並びを指定します。

LIB ファイル名

ライブラリファイルの並びを指定します。

オプション

オプションの並びを指定します。オプションについては，「38.2.3　オプション」を参照してください。

指定規則

- コマンドライン上には，コマンド名以降であればどこにでもコマンドファイルを指定できます。コマンド名の先頭には@を付けて指定します。コマンドファイルを指定すると，ファイルの中身がコマンドライン上に展開され，コマンドの一部となります。

（例）

```
LIB @OBJECT.lst -OUT:TEST.lib
```

- コマンドラインの区切り記号には，空白およびタブが使用できます。コマンドファイル内では，空白，タブ，セミコロン（;）および改行文字が使用できます。セミコロン（;）から改行文字までの指定は無効となり，コマンドファイル内にコメントを記述するために使用します。
- OBJ ファイル名，LIB ファイル名にはワイルドカード（*）が使用できます。
- ファイル名は必ず拡張子を付けて指定します。
- オプションは英大文字，英小文字のどちらで指定しても同じ扱いとなります。また，ハイフン（-）の代わりにスラント（/）で始めることもできます。
- 背反するオプションを二つ以上指定した場合，最後に指定したオプションが有効となります。

38.2.3 オプション

ライブラリ管理ツール（LIB コマンド）で使用できるオプションを次に示します。

表 38–2 ライブラリ管理ツールオプション一覧

オプション名	機能	参照先	Windows(x86) COBOL2002	Windows(x64) COBOL2002
-DEF	インポートライブラリを作成するためのモジュール定義ファイル(.def)を指定します	(1) -DEF:モジュール定義ファイル	○	○
-EXPORT	エクスポートを指定します	(2) -EXPORT:入口点名［,@順序番号［,NONAME］］	○	○
-EXTRACT	ライブラリメンバの抽出を指定します	(3) -EXTRACT:オブジェクトファイル名	○	○
-INCLUDE	シンボルを追加します	(4) -INCLUDE:シンボル	○	○
-LIBPATH	環境変数 LIB に先行して検索したいライブラリパスを指定します	(5) -LIBPATH:ライブラリのパス	○	○
-LIST	ライブラリメンバのリストを表示します	(6) -LIST	○	○
-NODEFAULTLIB	デフォルトライブラリを無視します	(7) -NODEFAULTLIB［:ライブラリ名］	○	○
-OUT	出力ファイル名を指定します	(8) -OUT:出力ファイル名	○	○
-REMOVE	ライブラリからオブジェクトを削除します	(9) -REMOVE:オブジェクトファイル名	○	○
-VERBOSE	処理の進行状況を詳しく表示します	(10) -VERBOSE	○	○
-?	オプションの概略一覧を表示します	(11) -?	○	○

（凡例）

○：オプションあり

(1)　-DEF:モジュール定義ファイル

モジュール定義ファイル（.def）からインポートライブラリを生成するためのオプションです。ファイル名は拡張子を付けて指定します。モジュール定義ファイル（.def）は一つだけ指定できます。モジュール定義ファイルについては，「38.4　モジュール定義ファイル」を参照してください。

ライブラリ管理ツール（LIB コマンド）は，モジュール定義ファイル内の LIBRARY 文，EXPORTS 文だけを参照し，それ以外は無視します。

(2)　-EXPORT:入口点名〔,@順序番号〔,NONAME〕〕

入口点の名前をエクスポートするためのオプションです。これは，モジュール定義ファイルの EXPORTS 文と同じです。

@順序番号（1～65,535）を指定すると，名前ではなく番号で関数を呼び出せます。この呼び出しの方がより速く関数を呼び出せます。

順序番号のあとに NONAME を付けると，名前はエクスポートされないため，順序番号による呼び出しだけができるようになります。

(3)　-EXTRACT:オブジェクトファイル名

ライブラリ内からオブジェクトファイルを取り出すためのオプションです。

取り出されたオブジェクトファイルは，取り出し元のライブラリに残ります。

-REMOVE オプションを同時に指定すると，取り出されたオブジェクトファイルはライブラリ内から削除されます。この場合，必ず-EXTRACT オプションを先に指定しなければなりません。

(4)　-INCLUDE:シンボル

指定されたシンボルをシンボルテーブルに追加するためのオプションです。

-DEF オプション指定時に，通常は取り込まれないライブラリオブジェクトを追加する場合に使用します。

(5)　-LIBPATH:ライブラリのパス

環境変数 LIB に先行して検索したいライブラリパスを指定するためのオプションです。入力する-LIBPATH オプションごとに一つのフォルダを指定します。複数のフォルダを指定する場合は，フォルダごとに-LIBPATH オプションを指定します。ライブラリ管理ツール（LIB コマンド）は，指定された順にフォルダを検索します。

(6)　-LIST

ライブラリ内のオブジェクトファイルのリストを表示するためのオプションです。

(7)　-NODEFAULTLIB〔:ライブラリ名〕

外部参照を解決するときに，検索するライブラリリストからデフォルトライブラリを除くためのオプションです。

ライブラリ名を指定しなかった場合は，すべてのデフォルトライブラリが検索対象から除かれます。

複数のデフォルトライブラリを検索対象から除外するためには，除外するライブラリごとに-NODEFAULTLIB オプションを繰り返し指定します。

(8)　-OUT:出力ファイル名

出力ファイルの名称を指定するためのオプションです。-EXTRACT オプションを指定した場合，このオプションは必ず指定しなければなりません。

(9)　-REMOVE:オブジェクトファイル名

オブジェクトファイルをライブラリから削除するためのオプションです。

(10)　-VERBOSE

ライブラリの処理に関する追加情報を表示するためのオプションです。

(11)　-?

オプションの概略一覧を表示するためのオプションです。

38.2.4　各種機能の使い方

「38.2.1　機能の概要」で示した LIB の各機能を使うためのコマンドラインの指定方法について説明します。

(1)　ライブラリの新規作成

一つ以上のオブジェクトファイルから一つのライブラリを作成します。

```
LIB OBJファイル … OBJファイル　〔-OUT:LIBファイル〕
```

-OUT オプションを省略した場合は，最初に指定した OBJ ファイルの拡張子を.lib に変更したファイル名で作成されます。

(2)　ライブラリ内のオブジェクトファイルの追加・変更

ライブラリ内で一つ以上のオブジェクトファイルを追加・変更します。

```
LIB LIBファイル OBJファイル … OBJファイル
    〔-OUT:LIBファイル〕
```

指定したオブジェクトファイルがライブラリ内にない場合，そのオブジェクトファイルが新規に追加されます。

指定したオブジェクトファイルがライブラリ内にある場合，そのオブジェクトファイルが指定したオブジェクトファイルで更新されます。

元のライブラリをそのまま残したい場合は，新規に作成するライブラリの名称を-OUT オプションで指定します。

(3) 複数のライブラリの結合

複数のライブラリを結合して一つのライブラリを生成します。

```
LIB LIBファイル … LIBファイル 〔-OUT:LIBファイル〕
```

元のライブラリをそのまま残したい場合は，新規に作成するライブラリの名称を-OUT オプションで指定します。

(4) ライブラリ内のオブジェクトファイルの削除

ライブラリ内のオブジェクトファイルを削除します。

```
LIB LIBファイル -REMOVE:OBJファイル 〔-OUT:LIBファイル〕
```

元のライブラリをそのまま残したい場合は，新規に作成するライブラリの名称を-OUT オプションで指定します。

(5) ライブラリからのオブジェクトファイルの取り出し

ライブラリ内のオブジェクトファイルを取り出して，ディスク上にオブジェクトファイルを生成します。元のライブラリは変更されません。

```
LIB LIBファイル -EXTRACT:OBJファイル -OUT:OBJファイル
```

(6) インポートライブラリの生成

モジュール定義ファイル（.def）からインポートライブラリを生成します。

モジュール定義ファイル内の EXPORTS 文のエクスポート名と LIBRARY 文の DLL ファイル名からインポート情報を作成し，インポートライブラリを生成します。

```
LIB -DEF:DEFファイル 〔LIBファイル…〕〔OBJファイル…〕〔-OUT:LIBファイル〕
```

インポートライブラリの生成の際，エクスポートファイル（.exp）も同時に生成されます。このエクスポートファイルは，リンカ（LINK コマンド）のための中間ファイルであり，ユーザが使用することはありません。

38.2.5　戻り値

ライブラリ管理ツール（LIB コマンド）の戻り値は次のとおりです。

戻り値	意味
0	正常終了した。
0 以外	エラーが発生した。LIB の終了の原因となったエラーの番号が，戻り値として返される。

38.3　リソースコンパイラ

38.3.1　機能の概要

アプリケーションプログラム中でアイコンを使用する場合，次の手順でアイコンのリソースをアプリケーションプログラムに組み込む必要があります。

1. リソースエディタを使ってリソースデータファイル（アイコンデータファイル）を作成する。
2. アプリケーションで使用するリソースを記述したリソース定義ファイル（.rc）を作成する。
3. リソースコンパイラ（RC コマンド）を使ってリソース定義ファイルをコンパイルし，リソースファイルを作成する。
4. アプリケーションのリンク時に，このリソースファイルをオブジェクトファイルと同様に指定してリンクする。

リソースコンパイラ（RC コマンド）は，リソース定義ファイル（.rc）とその中で指定されたリソースデータファイルを入力し，そのオブジェクトファイルであるリソースファイル（.res）を生成するツールです。

生成されたリソースファイルは，ほかのオブジェクトファイルと同様にリンカのコマンドラインに指定することでアプリケーションプログラムに組み込めます。

38.3.2　コマンドラインの形式

RC コマンドの形式を次に示します。

形式

```
RC 〔オプション〕 RCファイル名
```

オプション

オプションの並びを指定します。オプションについては，「38.3.3　オプション」を参照してください。

RC ファイル名

入力となるリソース定義ファイルの名称を指定します。拡張子を付けて指定しなければなりません。

なお，リソース定義ファイルの内容については，「38.3.5　リソース定義ファイルと COBOL で使用するリソース定義文」を参照してください。

指定規則

- リソース定義ファイル名にはワイルドカード（*）を指定できません。

- オプションは英大文字，英小文字のどちらで指定しても同じ扱いとなります。また，ハイフン（-）の代わりにスラント（/）で始めることもできます。

38.3.3　オプション

リソースコンパイラ（RC コマンド）で使用できるオプションを次に示します。

表 38–3　リソースコンパイラオプション一覧

オプション名	機能	参照先	Windows(x86) COBOL2002	Windows(x64) COBOL2002
-d	シンボルを定義します	(1)　-d シンボル	○	○
-fo	出力リソースファイル名を指定します	(2)　-fo RES ファイル名	○	○
-i	環境変数 INCLUDE の検索パスを追加します	(3)　-i INCLUDE 検索パス	○	○
-l	デフォルトの言語 ID を 16 進数で指定します	(4)　-l コードページ番号	○	○
-n	文字列テーブルのすべての文字列に NULL を追加します	(5)　-n	○	○
-u	シンボルの定義を取り消します	(6)　-u シンボル	○	○
-v	進行状況のメッセージを表示します	(7)　-v	○	○
-x	環境変数 INCLUDE を無視します	(8)　-x	○	○
-?または-h	オプションの概略一覧を表示します	(9)　-?または-h	○	○

（凡例）
　○：オプションあり

(1)　-d シンボル

プリプロセサのシンボルを定義するためのオプションです。

(2)　-fo RES ファイル名

生成されるリソースファイル名称を指定するためのオプションです。このオプションを指定しなかった場合は，リソース定義ファイル名の拡張子を.res に変更した名称でリソースファイルが生成されます。

(3)　-i INCLUDE 検索パス

環境変数 INCLUDE の検索対象パスに先行して検索するパスを指定するオプションです。

(4) -l コードページ番号

翻訳のためのデフォルトの言語を指定するためのオプションです。言語は，コードページ番号で指定してください。

注意事項

コードページとは，Windows が多国語対応のアプリケーション開発を容易にするために用意した文字コードの体系です。各国の言語ごとにページという単位で区分されています。

(5) -n

文字列テーブルのすべての文字列を，NULL 文字（X'00'）で終了する文字列にするためのオプションです。

(6) -u シンボル

プリプロセサのシンボルの定義を取り消すためのオプションです。

(7) -v

処理過程のレポートを表示するためのオプションです。

(8) -x

環境変数 INCLUDE を無視するためのオプションです。

(9) -?または-h

オプションの概略一覧を表示するためのオプションです。

38.3.4 戻り値

リソースコンパイラ（RC コマンド）の戻り値は次のとおりです。

戻り値	意味
0	正常終了した。
1	エラーが発生した。

38.3.5　リソース定義ファイルと COBOL で使用するリソース定義文

リソース定義ファイルは，リソースコンパイラ（RC コマンド）の入力となるファイルであり，アプリケーションプログラムで使用するリソース（アイコン）の定義を記述したものです。リソースは，リソース定義文で定義します。COBOL アプリケーションの作成で使用されるリソース定義文を，次に示します。

(1)　ICON 文

ICON 文は，アイコンの形を定義するビットマップを指定する文です。

構文

```
リソース名 ICON ファイル名
```

リソース名

リソースを識別するための一意名または整数値（1～65,535）を指定します。

ファイル名

リソースデータファイル（アイコンデータファイル）を指定します。カレントフォルダにないときは，絶対パス名で指定します。絶対パス名は引用符（"）で囲んでも囲まなくてもかまいません。ただし，空白を含む場合は必ず引用符（"）で囲まなければなりません。

使用例

アイコンビットマップ test.ico で，アイコンリソース test を定義します。

```
test ICON ¥users¥test.ico
```

注意事項

アイコンリソースは複数のイメージを含むことができます。

38.4　モジュール定義ファイル

モジュール定義ファイル（.def）は，ファイル名称，属性，エクスポート，システムの条件など，実行可能ファイルや DLL の特性を記述するファイルです。

リンカ（LINK コマンド）は，リンク時にこのファイルを参照し，このファイルに従った実行可能ファイル，DLL を作成します。また，ライブラリ管理ツール（LIB コマンド）でインポートライブラリを生成する場合にもこのファイルを使用します。

COBOL2002 では，ユーザはモジュール定義ファイルを意識しなくてもアプリケーションプログラムを生成できます。このモジュール定義ファイルは，リンカオプションのデフォルト値を変更するときや，外部名のエクスポートをするときに使用します。

ここではモジュール定義ファイルを記述するときの規則について説明します。

38.4.1　モジュール定義ファイルの記述規則

モジュール定義ファイルを記述するときの規則を次に示します。

- キーワードは，すべて英大文字で記述します。
- キーワードおよび引数の間，ならびに引数および引数の間は，空白，タブ，または改行文字で区切ります。
- コメントを記述するときは，セミコロン（;）と改行文字の間に記述します。また，モジュール定義文と同じ行には記述できません。
- NAME 文または LIBRARY 文を記述するときは，ほかのどの文よりも前に記述しなければなりません。また，NAME 文と LIBRARY 文は同時には記述できません。
- 空白やセミコロン（;）を含むファイル名は，引用符（"）で囲まなければなりません。
- EXPORTS 文は，同じモジュール定義ファイル内で複数回使用できます。また，複数の引数を指定できます。複数の引数を指定する場合は，それぞれの引数を一つ以上の空白，タブ，または改行文字で区切らなければなりません。

38.4.2　モジュール定義文

記述できるモジュール定義文を次に示します。

NAME 文：実行可能ファイル名の定義

LIBRARY 文：DLL ファイル名の定義

STACKSIZE 文：実行時に使用するスタックサイズの上限の定義

HEAPSIZE 文：実行時に使用するヒープサイズの上限の定義

EXPORTS 文：外部定義名のエクスポート宣言

VERSION 文：実行可能モジュールのバージョンの定義

(1) NAME 文

```
NAME 〔実行可能ファイル名〕
```

NAME 文は，実行可能ファイルを作成することを指示する文であり，同時に実行可能ファイルの名称を定義します。

実行可能ファイル名を省略した場合は，出力ファイル名はリンカ（LINK コマンド）の-OUT オプションの指定に従います。拡張子を省略した場合は.exe が仮定されます。

（例）実行可能ファイル TEST.exe を作成します。

```
NAME TEST
```

この NAME 文は，ほかのどの文よりも前に記述しなければなりません。また，LIBRARY 文と同時には指定できません。

NAME 文は，-DLL オプションを同時に指定しないときのリンカ（LINK コマンド）の-OUT オプションと等価です。

(2) LIBRARY 文

```
LIBRARY 〔DLLファイル名〕
```

LIBRARY 文は，DLL ファイルの名称を定義します。

DLL ファイル名を省略した場合は，出力ファイル名はリンカ（LINK コマンド）の-OUT オプションの指定に従います。拡張子を省略した場合は.dll が仮定されます。

（例）DLL ファイル TEST.dll を作成します。

```
LIBRARY TEST
```

この LIBRARY 文は，ほかのどの文よりも前に記述しなければなりません。また，NAME 文と同時には指定できません。

LIBRARY 文は，リンカ（LINK コマンド）の-OUT オプションと等価です。

DLL ファイル名を作成するときは，リンカ（LINK コマンド）の-DLL オプションの設定が必要です。

(3) STACKSIZE 文

```
STACKSIZE 仮想メモリサイズ〔,実メモリサイズ〕
```

STACKSIZE 文は，プログラムの実行時に使用できるスタックサイズの上限を定義するための文です。この文で，仮想メモリサイズの上限と，1 回で割り当てられる実メモリサイズの上限を定義します。

（例）スタックサイズの上限を 160,000 バイトにします。

```
STACKSIZE 160000
```

STACKSIZE 文は，リンカ（LINK コマンド）の-STACK オプションと等価です。

(4) HEAPSIZE 文

```
HEAPSIZE 仮想メモリサイズ〔,実メモリサイズ〕
```

HEAPSIZE 文は，プログラムの実行時に使用できるヒープサイズの上限を定義するための文です。この文で，仮想メモリサイズの上限と，1 回で割り当てられる実メモリサイズの上限を定義します。

（例）ヒープサイズの上限を 80,000 バイトにします。

```
HEAPSIZE 80000
```

HEAPSIZE 文は，リンカ（LINK コマンド）の-HEAP オプションと等価です。

(5) EXPORTS 文

```
EXPORTS エクスポート名〔=実外部名〕〔@順序番号〔NONAME〕〕
       〔エクスポート名〔=実外部名〕〔@順序番号〔NONAME〕〕〕
       〔…〕
```

EXPORTS 文は，プログラム中の外部名をエクスポートするための文です。

実外部名を省略した場合，エクスポート名と実外部名は同じとみなされます。

@順序番号（1～65,535）を指定すると，名前ではなく番号で関数を呼び出せます。順序番号のあとに NONAME を付けると，名称はエクスポートされないため，順序番号による呼び出しだけができるようになります。

（例）外部名_COMSUB1 を CBLSUB1 としてエクスポートします。

```
EXPORTS CBLSUB1=_COMSUB1
```

EXPORTS 文はリンカ（LINK コマンド）の-EXPORT オプションと等価です。

(6) VERSION 文

```
VERSION バージョン番号〔.リビジョン番号〕
```

VERSION 文は，実行可能ファイル，DLL のヘッダにバージョン番号を設定するための文です。

VERSION 文はリンカ（LINK コマンド）の-VERSION オプションと等価です。

38.4.3 モジュール定義文とリンカオプションとの関係

次のモジュール定義文には，それぞれ等価のリンカオプションがあります。

NAME 文

HEAPSIZE 文

LIBRARY 文

EXPORTS 文

STACKSIZE 文

VERSION 文

これらのモジュール定義文と等価のリンカオプションを同時に指定した場合，リンカオプションの指定が優先します。また，モジュール定義文と等価のリンカオプションを共に省略した場合は，リンカオプションのデフォルト値が適用されます。

優先順位は次のような順番で適用されます。

1. リンカオプションでの指定内容
2. モジュール定義文での指定内容
3. リンカオプションのデフォルト値

39 64bit アプリケーションの作成

この章では，Windows(x64) COBOL2002 について説明します。

39.1 Windows(x64) COBOL2002 について

Windows(x64) COBOL2002 では，Windows(x86) COBOL2002 と比べて呼び出し規約の変更，およびポインタサイズの 64bit 化に対応しています。ここでは，Windows(x86) COBOL2002 と Windows(x64) COBOL2002 について，相違を説明します。

39.1.1 使用できない機能

Windows(x64) COBOL2002 と Windows(x86) COBOL2002 では，使用できる機能が異なります。

Windows(x86) COBOL2002 の機能のうち，Windows(x64) COBOL2002 で使用できない機能について説明します。

(1) 機能

Windows(x64) COBOL2002 では使用できない機能を次に示します。

表 39–1 Windows(x64) COBOL2002 で使用できない機能

機能名	説明
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 Windows や UNIX 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。
Visual Basic との連携機能	Visual Basic プログラムから，DLL として作成された COBOL プログラムを呼び出します。

(2) ファイル編成

Windows(x64) COBOL2002 では使用できないファイル編成を次に示します。

- Btrieve（Pervasive.SQL）による索引編成ファイル

(3) MIOS7 COBOL85 互換機能

Windows(x64) COBOL2002 では，MIOS7 COBOL85 との互換機能を有効にする-CompatiM7 オプションは使用できません。しかし，-CompatiM7 オプションを指定しなくても有効となる機能は使用できます。使用できる機能を次に示します。

(a) 言語仕様

-CompatiM7 オプションを指定しなくても使用できる言語仕様を，次に示します。

環境部の入出力管理記述項(I-O-CONTROL)

- APPLY FILE-SHARE 句※

環境部のファイル管理記述項

- 相対編成ファイルに対する SYMBOLIC KEY 句
- ISAM による索引編成ファイルに対する SYMBOLIC KEY 句

注※

-IgnoreAPPLY,FILESHARE オプション指定時は，覚え書きとみなします。

(b)　実行時環境変数

-CompatiM7 オプションを指定しなくても使用できる実行時環境変数を，次に示します。

- CBLM7ENDKEY
- CBLOPS

39.1.2　Windows(x64) COBOL2002 固有の言語仕様

Windows(x64) COBOL2002 では，Windows(x86) COBOL2002 と比べて呼び出し規約の変更，およびポインタサイズの 64bit 化に対応しました。そのため，Windows(x64) COBOL2002 に対応する COBOL2002 の言語仕様にも Windows(x86) COBOL2002 と比べて一部変更があります。

Windows(x64) COBOL2002 固有の言語仕様について説明します。

(1)　呼び出し規約の fastcall への限定

Windows(x64) COBOL2002 では，呼び出し規約が fastcall に限定されます。ほかの呼び出し規約を指定することはできません。fastcall 呼び出し規約については，「18.4.2　呼び出し規約」の「(3)　fastcall 呼び出し規約（Windows(x64) COBOL2002 で有効)」を参照してください。

(a)　言語仕様

Windows(x64) COBOL2002 では使用できない呼び出し規約の指定に関する言語仕様を，次に示します。

表 39-2　使用できない呼び出し規約の指定に関する言語仕様

言語仕様	指定
環境部の特殊名段落（SPECIAL-NAMES）	• SPECIAL-NAMES の機能名の CDECL／STDCALL／PASCAL 指定
環境部の外部プログラム節（EXTERNAL-PROGRAM SECTION）	• EXTERNAL-PROGRAM SECTION の指定 • CALL-CONVENTION 段落の指定

これらの言語仕様を指定した場合，コンパイラは fastcall 呼び出し規約で処理し，指定された言語仕様を覚え書きとする警告エラーを出力します。

(b) コンパイラオプション

Windows(x64) COBOL2002 では使用できない，呼び出し規約の指定に関するコンパイラオプションを次に示します。

もし，これらのコンパイラオプションを指定した場合は，コンパイラは指定を無視して処理を続けます。

- -StdCall オプション
- -StdCallFile オプション
- -Dll オプションのサブオプション{Stdcall|Cdecl}※

注※

-Dll オプションのサブオプションの指定だけが無効になります。

(2) 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
- OLE オブジェクト参照項目
- バリアントデータ項目
- 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。

(3)　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-VARIANT 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」「11.31　LENGTH 関数」を参照してください。

LENGTH-OF-VARIANT 関数については，マニュアル「COBOL2002　言語　標準仕様編」「11.33 LENGTH-OF-VARIANT 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。

39.1.3　Windows(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 39–3　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
CBLHANDLE サービスルーチン	30.4.7　CBLHANDLE
サービスルーチンを使った OLE2 オートメーションクライアント機能※	付録 D.2　サービスルーチンを使った OLE2 オートメーションクライアント機能
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

注※

この機能は日立 COBOL85 からの古い仕様です。OLE2 オートメーションクライアント機能を使用する場合は，「25.　OLE2 オートメーション機能」に書いてある方法を使用してください。

(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，Windows(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 Windows(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。

39.2　COBOL ソースの作成とコンパイル

Windows(x64) COBOL2002 での COBOL ソースのコンパイル方法は，Windows(x86) COBOL2002 と同じです。しかし，Windows(x64) COBOL2002 と Windows(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。

詳細については「33.　COBOL ソースの作成とコンパイル」を参照してください。

39.2.1　使用できないコンパイラオプション

Windows(x64) COBOL2002 では使用できないコンパイラオプションについて説明します。

(1)　最終生成物の種類（プロジェクトの種類）のコンパイラオプション

最終生成物※の種類（開発マネージャでは，プロジェクトの種類）を設定するコンパイラオプションのうち，Windows(x64) COBOL2002 では使用できない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，DLL，または標準ライブラリのことを示します。

表 39–4　Windows(x64) COBOL2002 で使用できない最終生成物の種類のコンパイラオプション

コンパイラオプション	説明
-Dll，{Stdcall ｜ Cdecl}	DLL 形式のオブジェクトファイルを出力します。DLL 属性は stdcall，または cdecl です。

(2)　製品連携のコンパイラオプション

他製品との連携を設定するコンパイラオプションのうち，Windows(x64) COBOL2002 では使用できないコンパイラオプションを次に示します。

表 39–5　Windows COBOL2002 で使用できない製品連携のコンパイラオプション

コンパイラオプション	説明
-IsamExtend［,Zone］	Btrieve（Pervasive.SQL）による索引編成ファイルを使用します。
-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します。

(3)　リンクのコンパイラオプション

リンクを設定するコンパイラオプションのうち，Windows(x64) COBOL2002 では使用できないコンパイラオプションを次に示します。

表 39–6　Windows(x64) COBOL2002 で使用できないリンクのコンパイラオプション

コンパイラオプション	説明
-StdCall	stdcall 呼び出し指示ファイルを有効にします。
-StdCallFile .cbw ファイル名	stdcall 呼び出し指示ファイル名を指定します。

(4)　移行のコンパイラオプション

他システムとの移行を設定するコンパイラオプションのうち，Windows(x64) COBOL2002 では使用できないコンパイラオプションを次に示します。

表 39–7　Windows(x64) COBOL2002 で使用できない移行のコンパイラオプション

コンパイラオプション	説明
-CompatiM7	MIOS7 COBOL85 との互換機能を有効にします。

39.3　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。

39.3.1　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

Windows(x64) COBOL2002 での実行時環境変数の設定方法は，Windows(x86) COBOL2002 と同じです。詳細については，「36.2.1　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

ここでは，Windows(x64) COBOL2002 では使用できない実行時環境変数について説明します。

なお，Windows(x64) COBOL2002 で使用できない実行時環境変数を指定した場合は，指定した実行時環境変数は無視されて，処理が続行されます。

(a)　ファイル

ファイルの実行時環境変数のうち，Windows(x64) COBOL2002 では使用できない実行時環境変数を次に示します。

表 39-8　Windows(x64) COBOL2002 で使用できないファイルの実行時環境変数

実行時環境変数	指定する内容
CBLX_外部装置名	書式印刷するときの印刷サービス名称
CBL_BTRPGSZ	Btrieve による索引編成ファイルのページサイズ

39.3.2　プログラムの実行時の注意事項

Windows(x64) COBOL2002 でのプログラム実行時の注意事項を次に示します。

- Windows(x64) COBOL2002 で作成したアプリケーションから CALL 文または CBLEXEC サービスルーチンを使用して，Windows(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，Windows(x64) COBOL2002 で作成したアプリケーションからは Windows(x86) COBOL2002 で作成した DLL に含まれるプログラムは呼び出せません。

注※

Windows(x64) COBOL2002 と Windows(x86) COBOL2002 がインストールされた環境で実行してください。

39.4　実行可能ファイルと DLL の作成

Windows(x64) COBOL2002 での実行可能ファイルと DLL の作成の詳細については，「35.　実行可能ファイルと DLL の作成」を参照してください。

39.4.1　実行可能ファイルと DLL の作成の注意事項

Windows(x64) COBOL2002 での実行可能ファイルと DLL の作成の注意事項を次に示します。

- Windows(x86) COBOL2002 のオブジェクトファイル（.obj），およびライブラリファイル（.lib）は，COBOL2002 で作成したオブジェクトとリンクできません。

付録

付録 A　前バージョンからの変更点

ここでは，COBOL2002 の適用 OS の追加，機能のサポート終了に関する変更について説明します。

Windows COBOL2002 03-05 から Windows COBOL2002 04-00 への変更一覧

Windows COBOL2002 03-05 から Windows COBOL2002 04-00 への変更内容を次に示します。

- 次に示す適用 OS が対象外となりました。
  - ・Windows XP
  - ・Windows Vista
  - ・Windows 8
  - ・Windows Server 2003
  - ・Windows Server 2003 R2
  - ・Windows Server 2008
- OLE2 オートメーション機能のサーバ機能は，使用できません。
- CGI プログラム作成支援機能は，使用できません。
- EUR を使用した通信節による帳票出力機能は，使用できません。

各バージョンの変更内容については，「付録 M　各バージョンの変更内容」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
OOCOBOL	OOCOBOL OOCOBOL Run Time System
COBOL85 版 Cosminexus 連携	COBOL adapter for Cosminexus COBOL 拡張 Server Run Time System for Cosminexus
COBOL85 版 XML 連携	COBOL adapter for XML COBOL 拡張 Run Time System for XML COBOL adapter for XML developer's kit
COBOL2002 V1	COBOL2002（バージョン 01-00 以降 02-00 未満）
COBOL2002 V2	COBOL2002（バージョン 02-00 以降 03-00 未満）
COBOL2002 V3	COBOL2002（バージョン 03-00 以降 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。

なお，次の機能を使用しているプログラムは，Windows COBOL2002 V4 への移行性はありません。

- OLE2 オートメーションサーバ機能
- CGI プログラム作成支援機能
- EUR を使用した通信節による帳票出力機能

Windows(x86) COBOL2002 から Windows(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，「39.　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 Windows(x86) COBOL2002 V1/V2/V3 で作成したファイルの Windows(x86) COBOL2002 V4 への移行性を次の表に示します。

表 B–2　Windows(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	Windows(x86) COBOL2002 V3	Windows(x86) COBOL2002 V2	Windows(x86) COBOL2002 V1	COBOL85
COBOL ソースファイル	.cbl ほか	○	○	○	○
プログラム情報ファイル	.cbp ほか	△	△	△	×
画面定義ファイル	.wdf	○	○	○	○
ファイル定義ファイル	.flf	○	○	○	○
レコード定義ファイル	.rdf	○	○	○	○
実行環境ファイル	.cbr	△	△	△	×
stdcall 呼び出し指示ファイル	.cbw	○	○	○	○
プログラムテンプレートファイル	.cet	○	○	○	○※1
COBOL エディタ設定ファイル	.cex	○	−	−	−
DLL ファイル	.dll	○	○	△	×
実行可能ファイル	.exe	○	○	△	×
ライブラリファイル	.lib	△	△	△	×
オブジェクトファイル	.obj	○	○	△	×
プロジェクトファイル	.pmi	△	△	△	△※2
プロジェクトマスタファイル	.hmf	△	△	△	△※3
印刷書式情報ファイル	.prt	△	△	△	×
ブラウズデータファイル	.brd	−	−	−	−
OLE 定義ファイル	.odf	−	−	−	−
CGI 環境ファイル	.env	−	−	−	−
パッケージ情報ファイル	.pkg	○	○	○	○※3

対象ファイル		移行元システム			
ファイル種別	拡張子	Windows(x86) COBOL2002 V3	Windows(x86) COBOL2002 V2	Windows(x86) COBOL2002 V1	COBOL85
プロジェクト情報ファイル	.hmp	△	△	△	△※3
COBOL サービスルーチンファイル	.svw	○	○	○	○※3
ユーザキーワードファイル	.usw	○	○	○	○※3
TD コマンド格納ファイル (サブコマンド格納ファイル)	.tdi ほか	○	○	○	×
リポジトリファイル	.rep	○	○	○	−
COBOL アクセス用 Bean	.java	○	○	○	○
EJB 関連 Java ソースファイル	.java	○	○	○	○
XML アクセス用データ定義ファイル	.cbl	○	○	○	−
XML アクセスルーチンファイル (COBOL 副プログラム)	.cbl	○	○	○	−
文書型定義(DTD)ファイル	.cbl	○	○	○	−
XML ドキュメントファイル	.xml	○	○	○	−
データ定義(DDF)ファイル	.cxd	○	○	○	−
カタログファイル	.cxc	○	○	○	−

(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

COBOL85 03-01 以降にサポートしたファイルです。

注※2

COBOL85 06-00 未満でサポートしていたファイルです。なお，COBOL85 06-00 以降は，拡張子".hmf"になります。

注※3

COBOL85 06-00 以降にサポートしたファイルです。

Windows(x86) COBOL2002 および Windows(x64) COBOL2002 V1/V2/V3 で作成したファイルの Windows(x64) COBOL2002 V4 への移行性を次の表に示します。

表 B-3　Windows(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	Windows(x64) COBOL2002 V3	Windows(x64) COBOL2002 V2	Windows(x64) COBOL2002 V1	Windows(x86) COBOL2002
COBOL ソースファイル	.cbl ほか	○	○	○	○
プログラム情報ファイル	.cbp ほか	△	△	△	×
画面定義ファイル	.wdf	○	−	−	○
ファイル定義ファイル	.flf	○	−	−	○
レコード定義ファイル	.rdf	○	−	−	○
実行環境ファイル	.cbr	△	△	△	△
stdcall 呼び出し指示ファイル	.cbw	−	−	−	−
プログラムテンプレートファイル	.cet	○	○	○	○
COBOL エディタ設定ファイル	.cex	○	−	−	○
DLL ファイル	.dll	○	○	△	×
実行可能ファイル	.exe	○	○	△	×
ライブラリファイル	.lib	△	△	△	×
オブジェクトファイル	.obj	○	○	△	×
プロジェクトファイル	.pmi	○	○	○	△
プロジェクトマスタファイル	.hmf	○	○	○	△
印刷書式情報ファイル	.prt	△	△	△	△
ブラウズデータファイル	.brd	−	−	−	−
OLE 定義ファイル	.odf	−	−	−	−
CGI 環境ファイル	.env	−	−	−	−
パッケージ情報ファイル	.pkg	○	○	○	○
プロジェクト情報ファイル	.hmp	○	○	○	△
COBOL サービスルーチンファイル	.svw	○	○	○	○
ユーザキーワードファイル	.usw	○	○	○	○

対象ファイル		移行元システム			
ファイル種別	拡張子	Windows(x64) COBOL2002 V3	Windows(x64) COBOL2002 V2	Windows(x64) COBOL2002 V1	Windows(x86) COBOL2002
TD コマンド格納ファイル (サブコマンド格納ファイル)	.tdi ほか	○	○	○	○
リポジトリファイル	.rep	○	○	○	×
COBOL アクセス用 Bean	.java	○	○	○	○
EJB 関連 Java ソースファイル	.java	○	○	○	○
XML アクセス用データ定義ファイル	.cbl	○	○	−	△
XML アクセスルーチンファイル (COBOL 副プログラム)	.cbl	○	○	−	△
文書型定義(DTD)ファイル	.cbl	○	○	−	○
XML ドキュメントファイル	.xml	○	○	−	○
データ定義(DDF)ファイル	.cxd	○	○	−	○
カタログファイル	.cxc	○	○	−	○

(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)　プログラム情報ファイル（.cbp ほか）の移行での注意事項

蓄積されているカバレージ情報は，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げます。再生成したプログラム情報ファイルにカバレージ情報を蓄積するようにしてください。

(b) 実行環境ファイル（.cbr），印刷書式情報ファイル（.prt）の移行での注意事項

実行環境ファイル（.cbr），印刷書式情報ファイル（.prt）を作成したときの OS やマシン環境※が異なる場合は，実行環境ファイルの印刷書式と印刷書式情報ファイルを再設定してください。印刷書式情報には，OS とマシン固有の情報が含まれます。

注※

JP1/ServerConductor/Deployment Manager や Hitachi Compute Systems Manager Deployment Manager Plug-in のディスク複製インストール機能，および仮想化プラットフォームが提供するイメージファイル化による複製機能を使用する場合も含みます。

(c) DLL ファイル（.dll），実行可能ファイル（.exe），ライブラリファイル（.lib），オブジェクトファイル（.obj）の移行での注意事項

- 複数プログラムで外部属性を持つ（EXTERNAL 句指定のある）同一のファイルを使用する場合，Windows COBOL2002 01-00 でコンパイルしたプログラムと，Windows COBOL2002 01-01 以降でコンパイルしたプログラムを混在させているときは，外部属性を持つ同一のファイルを使用するプログラムをすべて再コンパイルしてください。
- Windows(x86) COBOL2002 V1 で作成した DLL ファイル，実行可能ファイルは，プログラムを構成するすべての DLL ファイル，実行可能ファイルをそのまま使用する場合，再コンパイルしなくても移行できます。

  ただし，次の個所に日本語などの多バイト文字を使用している場合，プログラムを構成する DLL ファイル，実行可能ファイルを一つでも再コンパイルするときは，すべての DLL ファイル，実行可能ファイルを再コンパイルしてください。

  - プログラム名／クラス名／インタフェース名／関数名
  - XML 連携機能のインタフェース名
- Windows COBOL2002 V1 で作成したライブラリファイルやオブジェクトファイルで，ISAM による索引ファイル入出力機能や，次の個所に日本語などの多バイト文字を使用している場合は，再コンパイルしてください。
  - プログラム名／クラス名／インタフェース名／関数名
  - XML 連携機能のインタフェース名
- Windows COBOL2002 で作成した日本語などの多バイト文字を使用した名称の DLL ファイルは，再コンパイルしてライブラリファイル（.lib）を再生成してください。再生成したライブラリファイル（.lib）で再リンクしてください。

(d) XML アクセス用データ定義ファイル（.cbl），XML アクセスルーチンファイル（.cbl）の移行での注意事項

Windows(x86) COBOL2002 XML 連携機能で生成した XML アクセスルーチンおよび XML アクセス用データ定義を移行する場合に注意事項があります。詳細については，マニュアル「COBOL2002　XML 連携機能ガイド」の「64bit 版 COBOL2002　XML 連携機能の制限事項」を参照してください。

(e) プロジェクトファイル（.pmi），プロジェクトマスタファイル（.hmf），プロジェクト情報ファイル（.hmp）の移行での注意事項

COBOL85 または Windows(x86) COBOL2002 で作成した次のファイルで，SEWB+/CONSTRUCTION または SEWB+基本開発環境セットのプログラム定義ファイルをソースファイルとして登録したものは，Windows COBOL2002 V4 への移行はできません。

- プロジェクトファイル（.pmi）
- プロジェクトマスタファイル（.hmf）
- プロジェクト情報ファイル（.hmp）

付録 B.2　OOCOBOL からの移行性

OOCOBOL からの移行性はありません。

付録 B.3　COBOL85 版 Cosminexus 連携の移行性と互換性

COBOL85 版 Cosminexus 連携の移行性と互換性について説明します。

- COBOL2002 Cosminexus 連携機能で生成した COBOL アクセス用 Bean（.java）は，COBOL85 版 Cosminexus 連携で使用できません。
- COBOL85 版 Cosminexus 連携で生成した COBOL アクセス用 Bean を COBOL2002 Cosminexus 連携機能向けに再生成する場合，COBOL アクセス用 Bean を使用している Java プログラムも再コンパイルしてください。
- COBOL2002 Cosminexus 連携機能では，setter で不当データを設定した際に発生する例外の発生個所が変わります。例外の発生個所を次に示します。
  - COBOL85 版 Cosminexus 連携：call メソッド
  - COBOL2002 Cosminexus 連携機能：setter メソッド

  setter と callCOBOL を同じ try-catch でキャッチしている場合，COBOL85 版 Cosminexus 連携では，setter で発生したエラーか callCOBOL で発生したエラーか区別できませんが，COBOL2002 Cosminexus 連携機能では区別できます。

  Java プログラムの変更例を次に示します。

  （変更前）

```
    :
try {
  bean.setP_name("");
  bean.setP_address("");
  callCOBOL();
} catch ( J2CBException e ) {
```

```
    : /* setter,callCOBOLどちらかで発生したエラー */
  }
```

（変更後）

```
    :
  try {
    bean.setP_name("");
    bean.setP_address("");
  } catch ( J2CBException e ) {
    : /* setterで発生したエラー */
  }
  try {
    callCOBOL();
  } catch ( J2CBException e ) {
    : /* callCOBOLで発生したエラー */
  }
```

- setter の引数に COBOL 引数定義と対応しないオブジェクトを指定した場合※1，COBOL85 版 Cosminexus 連携と COBOL2002 Cosminexus 連携機能で動作が異なります。
  - COBOL85 版 Cosminexus 連携：callCOBOL メソッドで予期しないエラーとなるか，例外にならないで続行※2 することがあります。
  - COBOL2002 Cosminexus 連携機能：setter メソッドで例外となります。「J2CByyy2014」のメッセージが出力されます。メッセージの詳細については，マニュアル「COBOL2002　Cosminexus 連携機能ガイド」を参照してください。

  COBOL85 版 Cosminexus 連携で稼働済みの Java プログラムを COBOL2002 環境に移行する場合，このケースに該当しないか確認してください。該当する場合は，COBOL で定義した属性に対応するオブジェクトを，setter の引数に指定してください。

  注※1

  USAGE COMP-1（単精度浮動小数点項目）で定義した引数に，String オブジェクトで値を設定しようとした場合などです。

  COBOL で定義した属性に対応するオブジェクトの一覧については，COBOL アクセス用 Bean 生成ツールの「ステップ 2/3 画面」を参照してください。COBOL アクセス用 Bean 生成ツールについては，マニュアル「COBOL2002 Cosminexus 連携機能ガイド」を参照してください。

  注※2

  例えば，小数点のない 2 進 2 バイトのデータ項目に setter で値を設定する場合，setter の引数には通常 Short オブジェクトを指定します。しかし，setter の引数に Integer オブジェクトを指定すると，誤った値が設定されることがあります。
- 例外発生時の例外情報コードで，COBOL85 版 Cosminexus 連携と異なるコードがあります。

付録 B.4　COBOL85 版 XML 連携の移行性と互換性

COBOL85 版 XML 連携の移行性と互換性について説明します。

- COBOL85 版 XML 連携は COBOL2002 と連携して使用できません。
- COBOL85 版 XML 連携で作成したファイルの移行性を次の表に示します。

表 B–4　COBOL85 版 XML 連携に関係するファイルの COBOL2002 V4 への移行性

対象ファイル		移行先システム	
ファイル種別	拡張子	Windows(x86) COBOL2002 V4	Windows(x64) COBOL2002 V4
XML アクセス用データ定義ファイル	.cbl	○	×
XML アクセスルーチンファイル（COBOL 副プログラム）	.cbl	○	×
文書型定義（DTD）ファイル	.xml または.dtd	○	○
XML ドキュメントファイル	.xml	○	○
データ定義（DDF）ファイル	.cxd	○	○
カタログファイル	.cxc	○	○

（凡例）
○：移行できる
×：移行できない

付録 B.5　機能ごとの移行性と互換性に関する注意事項

(1)　浮動小数点演算の結果に関する注意事項

異なるプラットフォームへ移行する場合，浮動小数点演算の精度の違いによって，移行元システムと結果差異が発生することがあります。浮動小数点数として妥当な値でも，異なるプラットフォームへ移行する場合は浮動小数点演算の精度の違いによる誤差に注意してください。

浮動小数点演算の精度の違いによる誤差が発生する条件を次に示します。

- 被べき数（底）またはべき数（指数）が浮動小数点項目や浮動小数点数字定数である場合
- べき数（指数）が整数でない定数や整数項目でないべき乗演算を使用する場合
- 次のどれかの組み込み関数を使用する場合
  - ACOS 関数
  - ASIN 関数
  - ATAN 関数
  - COS 関数

- LOG 関数
- LOG10 関数
- RANDOM 関数
- SIN 関数
- SQRT 関数
- SUM 関数（引数に浮動小数点項目，除算またはべき乗が含まれる算術式を指定した場合）
- TAN 関数
- ANNUITY 関数
- PRESENT-VALUE 関数

(2) 実行可能ファイルまたは DLL ファイル作成時の注意事項

- インストール先の変更点

  COBOL2002 V4 では，COBOL85 または旧バージョンの COBOL2002 からインストール先が変更されています。

  バッチファイルなどで環境変数 PATH と LIB を個別に設定している場合や，対象ファイルを絶対パス名で参照している場合，設定値の修正が必要です。

- リンカやライブラリのインストール先

  COBOL2002 V3 では，同梱する Visual Studio のリンカなどのインストール先を変更しています。また，COBOL2002 V4 では，COBOL2002 のインストーラが Windows SDK に含まれるマニフェストツールおよび C/C++インポートライブラリをインストールしなくなりました。

  表 B-5　リンカやライブラリのインストール先

対象ファイル	対象システム	インストール先
リンカ，ライブラリ管理ツール，リソースコンパイラ	COBOL2002 V1/V2/V3※1	<COBOL2002 インストールフォルダ>¥BIN
	COBOL2002 V3※2/V4	<COBOL2002 インストールフォルダ>¥BIN¥SDK
マニフェストツール	COBOL2002 V1/V2	<COBOL2002 インストールフォルダ>¥BIN
	COBOL2002 V3	<COBOL2002 インストールフォルダ>¥BIN¥SDK
	COBOL2002 V4	Windows SDK インストール先の MT コマンドがある¥BIN ダ
C/C++インポートライブラリ	COBOL2002 V1/V2/V3※1	<COBOL2002 インストールフォルダ>¥BIN
	COBOL2002 V3※2	<COBOL2002 インストールフォルダ>¥LIB¥SDK

対象ファイル	対象システム	インストール先
	COBOL2002 V4	Windows SDK インストール先のインポートライブラリがあるフォルダ※3

注※1

Windows(x64) COBOL2002 03-00 の場合だけです。

注※2

Windows(x64) COBOL2002 03-00 を除きます。

注※3

詳細については，「付録 J　環境変数の設定」を参照してください。

- ISAM ライブラリのインストール先

  移行元の OS が Windows Vista 未満，または Windows Server 2008 未満の場合はインストール先が変更となります。

  COBOL2002 V4 で同梱する ISAM のインストール先は COBOL2002 インストールフォルダと同じ階層です。

  COBOL2002 のインストール先をデフォルトにしたときの，同梱する ISAM のインストール先を次に示します。

  - Windows(x86) COBOL2002 の場合：

    32bit OS の場合：%ProgramFiles%¥ISAM

    64bit OS の場合：%ProgramFiles(x86)%¥ISAM
  - Windows(x64) COBOL2002 の場合：%ProgramFiles%¥ISAM64

(3)　Windows(x86) COBOL2002 01-03 未満の COBOL2002 Cosminexus 連携機能を移行する場合の注意事項

(a)　COBOL アクセス用 Bean を javac でコンパイルする場合の注意事項

javac コマンドで，Windows(x86) COBOL2002 01-03 未満で生成した COBOL アクセス用 Bean に対し-Xlint オプションを指定すると，次に示す警告メッセージが出力されます。

警告: [serial] 直列化可能なクラス yyy には，serialVersionUID が定義されていません。

注

yyy は，COBOL アクセス用 Bean として生成したパッケージ名.クラス名を示します。

この警告メッセージは無視することができ，生成したクラスの実行動作を保証します。

(b)　ライブラリの常駐化に関する注意事項

デフォルトオプション指定では，COBOL UAP 呼び出し後，COBOL UAP ライブラリをアンロードしないように変更しています。メモリ資源の面から，使用頻度が少ないライブラリをアンロードしたい場合は，

loadingrule オプションおよび環境変数 CBLJ2CBSTAYLIB で，COBOL UAP ライブラリのロード／アンロードを制御してください。

付録 C　Windows OS 固有の注意事項

Windows で COBOL2002 を使用するときの注意事項を次に示します。

付録 C.1　管理者権限についての注意事項

UAC 機能が有効な場合，管理者ユーザとしてログオンしても，アプリケーションの実行権限は標準権限に下げられます。例えば，管理者権限を必要とするアプリケーションを起動しようとすると，システムは，昇格ダイアログボックスを表示して，ユーザの確認を求めます。この場合，ユーザが実行を許可すると，アプリケーションは管理者権限で実行されます。または，管理者権限で実行するためにプログラムやコマンドのプロパティを変更するか，「管理者として実行」で実行するなどの操作が必要です。

管理者権限についての注意事項を次に示します。

(1)　プログラムを管理者権限で実行する場合の注意事項

管理者権限でプログラムを実行する場合，起動時に Windows のユーザアカウント制御（UAC）の昇格ダイアログボックスが表示されます。また，ユーザが作成したバッチファイル内など，呼び出し元で設定した環境変数は，呼び出し先の管理者権限で動作するプロセスには引き継がれません。

これらを回避する場合は，呼び出し元のプログラムを管理者権限で実行してください。その際，次の点に注意してください。

- 呼び出し元から呼ばれるすべてのプログラムは管理者権限で動作することになります。セキュリティの面で問題がないかどうか確認してください。
- 標準権限のコマンドプロンプトから管理者権限のプログラムを起動した場合，管理者権限のプログラムから標準権限のコマンドプロンプトで設定した環境変数にアクセスできません。コマンドプロンプトも管理者権限で起動してください。バッチファイルを使用する場合は，特に注意してください。
- 開発マネージャや COBOL エディタなどを管理者権限で実行させている場合，標準権限で動作するエクスプローラからファイルをドラッグ＆ドロップできません。

(2)　テストデバッガおよびカバレージ使用時の注意事項

- テストデバッガおよびカバレージで，次に示す操作を標準権限（Administrators グループのユーザ含む）で実行する場合，注意が必要です。
  - 連動実行のテストデバッガでデバッグする場合
  - 管理者権限を設定したプログラムをデバッグ対象にする場合

  次に示す方法で実行してください。
  - 連動実行のテストデバッガでデバッグする場合

Administrators グループのユーザの場合は，テストデバッガおよびデバッグ対象プログラムを管理者権限で実行してください。

それ以外のユーザの場合は，テストデバッガおよびデバッグ対象プログラムを，管理者権限で実行するか，「グローバルオブジェクトの作成」権限を追加してから標準権限で実行してください。

「グローバルオブジェクトの作成」権限は，コントロールパネルの［管理ツール］から［ローカルセキュリティポリシー］を使用して，実行するユーザに追加してください。

- 管理者権限を設定したプログラムをデバッグ対象にする場合

  テストデバッガおよびカバレージを管理者権限で実行してください。

  テストデバッガおよびカバレージを開発マネージャから呼び出す場合は，開発マネージャを管理者権限で実行してください。

  テストデバッガ，カバレージまたは開発マネージャを標準権限（Administrators グループのユーザ含む）で実行すると，デバッグ対象プログラムの起動に失敗します。

- テストデバッガおよびカバレージを管理者権限で実行した場合は，デバッグ対象プログラムも管理者権限で動作します。標準権限で実行した場合は，デバッグ対象プログラムも標準権限で動作します。しかし，テストデバッガおよびカバレージを標準権限で実行した場合は，Windows リソース保護（WRP）の影響を受けるため，管理者権限で実行したときと実行結果が変わることがあります。そのため，運用時と同じ権限でテストデバッガおよびカバレージを実行してください。

(3) ISAM による索引編成ファイル使用時の注意事項

プログラムで ISAM による索引編成ファイルを使用する場合は，管理者権限で実行してください。管理者権限がない場合，ISAM による索引編成ファイルにアクセスすると実行時エラーになります。

(4) ISAM ユティリティに関する注意事項

使用する ISAM ユティリティによって，管理者権限での実行が必要かどうかが変わります。ISAM ユティリティと管理者権限について説明します。

(a) 管理者権限での実行が必要な ISAM ユティリティ

管理者権限での実行が必要な ISAM ユティリティを次に示します。

Windows(x86) COBOL2002 の場合

- オプション情報の設定ユティリティ
- islckext コマンド
- islckclear コマンド
- islckclearL コマンド

Windows(x64) COBOL2002 の場合

- オプション情報の設定ユティリティ
- islckext64 コマンド

- islckclear64 コマンド

(b) 原則，管理者権限での実行が必要な ISAM ユティリティ

原則，管理者権限での実行が必要な ISAM ユティリティを次に示します。

Windows(x86) COBOL2002 の場合

- 定義ファイルの作成ユティリティ
- レコード内容の表示ユティリティ
- iscpy コマンド
- isers コマンド
- isren コマンド
- isprt コマンド

Windows(x64) COBOL2002 の場合

- 定義ファイルの作成ユティリティ
- レコード内容の表示ユティリティ
- iscpy64 コマンド
- isers64 コマンド
- isren64 コマンド
- isprt64 コマンド

ただし，セキュリティ上の問題で，管理者権限で実行できない場合は，コントロールパネルの［管理ツール］から［ローカルセキュリティポリシー］を使用して，標準権限にグローバルオブジェクトの作成権限を追加したユーザで実行してください。

グローバルオブジェクトの作成権限がないユーザで，これらの ISAM ユティリティを標準権限で実行した場合，定義ファイルの作成ユティリティ以外の ISAM ユティリティはエラーとなります。表示されるエラーメッセージを次の表に示します。

項番	ISAM ユティリティ	エラーメッセージ
1	• レコード内容の表示ユティリティ • isprt コマンド	KAIU001-E ＩＳＡＭアクセス関数エラーが発生しました。(isopen)
2	• iscpy コマンド • isers コマンド	KAIU001-E ＩＳＡＭアクセス関数エラーが発生しました。(normal,5)
3	isren コマンド	KAIU001-E ＩＳＡＭアクセス関数エラーが発生しました。(isrename,1010)

注意

これらのメッセージは，ISAM ファイルにアクセス権がない場合にも表示されます。

また，管理者権限で起動したコマンドプロンプトでもエラーの原因を確認できます。次に示すコマンドを実行し，表示される関数エラー情報を確認してください。

Windows(x86) COBOL2002 の場合

```
ismdmpntコマンド（ラージファイル形式の場合ismdmpntLコマンド）
```

Windows(x64) COBOL2002 の場合

```
ismdmpnt64コマンド
```

該当する時間に iserrno=ESYSERR,issyserr=5,syserrknd=13 のエラー情報が表示されている場合，グローバルオブジェクトの作成権限がないユーザで，これらの ISAM ユティリティを標準権限で実行したことを示します。

ismdmpnt コマンドまたは ismdmpnt64 コマンドの表示例

```
[01]14/08/01; 12:00:00; procno = 2760; thrdno = 2392 (ISAMファイル名)
(ISAM関数): iserrno=ESYSERR,iserrcd = 00000113 ,issyserr = 00000005,syserrknd = 00000013
```

なお，定義ファイルの作成ユティリティは，ISAM ファイルの作成先にすでに ISAM ファイルが存在する場合，アクセス権などの属性情報が引き継がれます。

(c) 標準権限で実行できる ISAM ユティリティ

標準権限で実行できる ISAM ユティリティを次に示します。

Windows(x86) COBOL2002 の場合

- キーの追加／削除／再構築ユティリティ
- ファイルの検証ユティリティ
- ファイルの圧縮ユティリティ
- ファイルの変換ユティリティ
- ファイルの抽出ユティリティ
- ファイルの移行ユティリティ
- キー定義情報の表示ユティリティ
- iskeydef コマンド
- iskeymnt コマンド
- isconv コマンド
- ischk コマンド
- isext コマンド
- isinfo コマンド
- iscond コマンド

- isshift コマンド
- isgendef コマンド

Windows(x64) COBOL2002 の場合

- キーの追加／削除／再構築ユティリティ
- ファイルの検証ユティリティ
- ファイルの圧縮ユティリティ
- ファイルの変換ユティリティ
- ファイルの抽出ユティリティ
- ファイルの移行ユティリティ
- キー定義情報の表示ユティリティ
- iskeydef64 コマンド
- iskeymnt64 コマンド
- isconv64 コマンド
- ischk64 コマンド
- isext64 コマンド
- isinfo64 コマンド
- iscond64 コマンド
- isshift64 コマンド

なお，ファイル／レコード定義は，ISAM ファイルを生成する際に ISAM ユティリティを使用しますが，ISAM ファイルは標準権限で生成できるため，ファイル／レコード定義は，標準権限，管理者権限のどちらでも起動できます。

(d)　ISAM ユティリティを管理者権限だけで使用できるようにする方法

ISAM ユティリティを<ISAM のインストールフォルダ>※¥Bin¥Compat フォルダに格納されている ISAM ユティリティと置き換えることで，ISAM ユティリティを管理者権限だけで使用できます。なお，置き換え前の ISAM ユティリティに戻すことができるように，作業前に置き換え前の ISAM ユティリティをバックアップしておくことを推奨します。

ISAM ユティリティを置き換える手順を次に示します。

1. 管理者権限のあるユーザであることを確認する。
2. コマンドプロンプトを管理者権限に権限昇格して起動する。
3. <ISAM のインストールフォルダ>※¥Bin¥Compat フォルダに移動する。

```
cd <ISAMのインストールフォルダ>※¥Bin¥Compat
```

4. ISAM ユティリティを実行していないことを確認する。

5. xcopy コマンドで Compat フォルダ内の exe ファイルを直上の Bin フォルダへコピーする。

```
xcopy *.exe ..¥ /y
```

6. ログオフする。

注※

ISAM のインストールフォルダは，デフォルトでは次のとおりです。

32bit 版：

```
C:¥Program Files¥Hitachi¥ISAM
```

64bit 版：

```
C:¥Program Files¥Hitachi¥ISAM64
```

付録 C.2　ファイル格納先についての注意事項

(1)　ファイルダイアログボックスで格納する場合

開発マネージャ，COBOL エディタ，実行支援などで編集したファイルを格納する場合，インストール直後の状態では，ファイルダイアログボックスで表示される格納先は COBOL2002 インストールフォルダに位置付けられています。そのまま格納しようとすると Windows リソース保護（WRP）によって失敗※しますので，別の場所を指定してください。Windows リソース保護（WRP）については，各 OS のヘルプなどを参照してください。

注※

管理者権限で実行した場合を除きます。

付録 C.3　文字コードについての注意事項

COBOL2002 のコンポーネント（コンパイラ，実行時ライブラリ，開発ツール）や COBOL2002 または COBOL85 で作成したプログラムで使用するフォルダ名，ファイル名，プログラムへの入力文字列，および環境変数に指定できる文字は，シフト JIS の範囲だけです。JIS X0213 の第 3 水準漢字，および第 4 水準漢字を含む Unicode の文字は使用できません。

第 3 水準漢字，および第 4 水準漢字の文字を含んだフォルダやファイル名などを入力した場合，システムが第 3 水準漢字，および第 4 水準漢字の文字を”?”などに変換し，意図しない値になりますので，注意してください。

付録 C.4　実行可能ファイルの実行権限についての注意事項

Windows(x86) COBOL2002 02-00 以降または Windows(x64) COBOL2002 で作成したアプリケーションは，標準権限，管理者権限のどちらでも実行できます。

ただし，アプリケーションを実行する場合の実行権限について，次の注意事項があります。

- COBOL プログラムから CALL 文または CBLEXEC サービスルーチンで実行可能ファイルを実行する場合は，呼び出し元と同じ権限で実行します。例えば，標準権限のアプリケーションから管理者権限のアプリケーション（実行可能ファイル）を CALL 文で呼び出した場合，KCCC0137R-S のメッセージが出力され，システムから返されたエラー番号に 740（管理者権限のないユーザから呼び出された）が示されます。このとき，プログラム互換性アシスタント（PCA）に登録され，再度実行した場合はエラーにならない場合があります※。ただし，プログラム互換性アシスタント（PCA）に登録された標準権限のアプリケーションから管理者権限のアプリケーションの呼び出しを COBOL2002 では保証しません。
- ISAM による索引ファイル入出力機能を使用する場合，管理者権限で実行してください。セキュリティ上の問題で管理者権限で実行できない場合は，コントロールパネルの［管理ツール］から［ローカルセキュリティポリシー］を使用して，標準権限にグローバルオブジェクトの作成権限を追加したユーザで実行してください。

注※

プログラム互換性アシスタント（PCA）が自動的に回避策を適用する機能は無効となっていて，継続的にエラーとなることがあります。

アプリケーションの権限の設定方法，およびプログラム互換性アシスタント（PCA）については，各 OS のヘルプなどを参照してください。

管理者権限を設定する例

実行可能ファイルのリンク時，-MANIFESTUAC リンカオプションで実行可能ファイルのアクセス許可レベルの値を指定します。

```
ccbl2002 -Main,System test01.cbl -Link -MANIFESTUAC:level='requireAdministrator'
```

付録 C.5　実行時の注意事項

実行時の注意事項を次に示します。

- Windows(x86) COBOL2002 02-00 以降または Windows(x64) COBOL2002 で作成したアプリケーションは，デフォルトで内部マニフェストを埋め込んでいます。

  外部マニフェストファイル※よりも内部マニフェストの情報が優先されるため，内部マニフェストを埋め込んだアプリケーションに対して，外部マニフェストファイルの設定内容は有効となりません。

注※

実行可能ファイル名の後ろに「.manifest」を付加したファイルで，権限属性などを定義できます。

- Windows(x86) COBOL2002 02-00 以降または Windows(x64) COBOL2002 で作成したアプリケーションを，「プロパティ」ダイアログボックスで実行権限を設定したあと，別のフォルダに移動した場合は，再度実行可能ファイルの「プロパティ」ダイアログボックスで，管理者権限を設定する必要があります。詳細は，各 OS のヘルプなどを参照してください。
- アプリケーションでファイル入出力機能などを使用する場合，Windows リソース保護（WRP）によって，Windows リソース（OS ファイル，フォルダなど）が保護されるので，ファイルの出力先の指定に注意が必要です。Windows リソース保護（WRP）については，各 OS のヘルプなどを参照してください。
- リンカのインストール先フォルダについては，「付録 J　環境変数の設定」を参照してください。

付録 D　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，これらの機能を使用しないで，同じ機能を持つほかの機能を使用してください。

付録 D.1　ACCEPT／DISPLAY 文を使用した CSV ファイルへのアクセス

ACCEPT 文，DISPLAY 文を使用して，CSV 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。

なお，ACCEPT／DISPLAY 文を使用した CSV ファイルへのアクセス方法では，一つの実行単位で一つの CSV ファイルしか扱えません。そのため，通常は READ／WRITE 文を使用して CSV 編成ファイルとして入出力する方法を推奨します。CSV 編成ファイルの入出力方法については，「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。

(1)　CSV データの読み込み

環境変数 CBL_SYSCSVIN で割り当てた CSV ファイルのデータを，ACCEPT 文を実行することで 1 行ずつ（改行文字まで）読み込めます。読み込まれた 1 行は，コンマ（,）で区切られたデータ（セル）ごとにユーザが定義したデータ領域に格納されます。

ACCEPT 文の形式

```
ACCEPT 一意名 FROM 呼び名※
```

注※

呼び名は，環境部の特殊名段落で，SYSCSV に関連づけておく必要があります。詳細は，マニュアル「COBOL2002　言語　拡張仕様編」「17.3.1　環境部（SPECIAL-NAMES）(CSV ファイル入出力機能)」を参照してください。

CSV ファイルの割り当て

ACCEPT 文で入力する CSV ファイルは，環境変数 CBL_SYSCSVIN に絶対パス名で指定しておきます。環境変数の指定例を次に示します。

```
CBL_SYSCSVIN=C:¥dir¥file1.csv
```

規則

- 環境変数 CBL_SYSCSVIN にファイルが割り当てられていない場合，最初の ACCEPT 文実行時に実行時メッセージが出力され，プログラムの実行が中止されます。
- CSV ファイルから読み込まれた 1 行中の各セルは，一意名に従属する基本項目に先頭から 1 対 1 対応で格納されます。セルの個数が基本項目の個数より多い場合，対応しないセルは無視されます。セルの個数が基本項目の個数より少ない場合，対応しない基本項目の値は，ACCEPT 文の実行前の状態から変更されません。

- セルは，英数字項目の転記規則に従って一意名のデータ項目に格納されます。
- 入力データ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（X'0D0A'）を区切り文字としてデータが入力されます。
- 入力データ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。

  1. コンマの直後がダブルコーテーションの場合，次のダブルコーテーションの直前までをデータとして扱う。

  2. 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う。

(2) CSV データの書き込み

環境変数 CBL_SYSCSVOUT で割り当てたファイルに，DISPLAY 文を実行することで CSV 形式のデータを 1 行ずつ書き出せます。各行は，ユーザが定義したデータ領域の値にコンマを付けた形式で書き出されます。書き出されたファイルは，表計算プログラムの CSV ファイルとなります。

DISPLAY 文の形式

```
DISPLAY 一意名 UPON 呼び名※
```

注※

呼び名は，環境部の特殊名段落で，SYSCSV に関連づけておく必要があります。詳細は，マニュアル「COBOL2002　言語　拡張仕様編」「17.3.1　環境部（SPECIAL-NAMES）(CSV ファイル入出力機能)」を参照してください。

CSV ファイルの割り当て

DISPLAY 文で出力する CSV ファイルは，環境変数 CBL_SYSCSVOUT に絶対パス名で指定しておきます。環境変数の指定例を次に示します。

```
CBL_SYSCSVOUT=C:¥dir¥file2.csv
```

規則

- 環境変数 CBL_SYSCSVOUT にファイル名が割り当てられていない場合，最初の DISPLAY 文実行時に実行時メッセージが出力され，プログラムの実行が中止されます。
- データは，一意名または一意名に従属する基本項目のそれぞれの値にコンマを付けてデータを引用符で囲み，さらに終端に改行文字を付けて出力されます。
- セルデータの最後に空白文字がある場合，その空白は出力されません。

付録 D.2　サービスルーチンを使った OLE2 オートメーションクライアント機能

ここでは，サービスルーチンを使って OLE2 オートメーションクライアント機能を使用する方法について説明します。

なお，通常 OLE2 オートメーションクライアント機能を使用する場合は，「25.2　OLE2 オートメーションクライアント機能」に示す方法を使用してください。

(1)　サービスルーチンの一覧

OLE2 オートメーションクライアント機能を使用するために 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。なお，サービスルーチンは，表中の呼び出し順序に従って呼び出す必要があります。

表 D-1　サービスルーチンの種類

呼び出し順序	サービスルーチン	機能
1	CBLOLE2INIT	OLE2 を初期設定する。
2	CBLCREATEOBJ	操作したい OLE2 サーバの OLE オブジェクトを生成する。
	CBLGETOBJ	操作したい OLE2 サーバの OLE オブジェクトを取得する。
3	CBLGETPROPERTY	OLE2 サーバの OLE プロパティの値を取得する。
	CBLPUTPROPERTY	OLE2 サーバの OLE プロパティに値を設定する。
	CBLMETHODCALL	OLE2 サーバの OLE メソッドを操作する。
	CBLVARIANTINF	VARIANT 値の属性を調べる。
	CBLSETCBLITEM	VARIANT 値から COBOL データ項目に変換する。
	CBLSETVARIANT	COBOL データ項目から VARIANT 値に変換する。
4	CBLOLE2UNINIT	OLE2 を終了させる。

なお，各サービスルーチンの「戻り値」に記されているエラーコード値の内容については，「(3)　エラーコード値」を参照してください。

(2)　サービスルーチンの詳細

サービスルーチンの詳細を以下に示します。

なお，使用例のプログラムを Windows(x64) COBOL2002 で使用する場合は，読み替えが必要です。

使用例の記述（Windows(x86) COBOL2002 で使用）

```
01 ARG1 PIC 9(9) USAGE COMP VALUE ZERO.
```

読み替えの記述（Windows(x64) COBOL2002 で使用）

```
01 ARG1 PIC 9(18) USAGE COMP VALUE ZERO.
```

(a)　CBLOLE2INIT

OLE2 の初期値を設定するサービスルーチンです。

形式

```
CALL 'CBLOLE2INIT' USING BY VALUE 引数1
                   BY REFERENCE 引数2.
```

引数

- 引数 1 には，4 バイト※の 2 進項目で 0 を設定します。このサービスルーチンを呼び出したあとに，引数 1 の値を書き換えないでください。
- 引数 2 は，次の集団項目を要求します。

  ```
  01 引数2.
   02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.
  ```

  4 バイトの 2 進項目で 0 を設定します。

注※

Windows(x64) COBOL2002 の場合は 8 バイトに読み替えてください。

戻り値

正常：0

上記以外：エラーコード値

注意事項

CBLOLE2INIT サービスルーチンは，次のサービスルーチンより先に呼び出されている必要があります。

- CBLCREATEOBJ
- CBLVARIANTINF
- CBLGETOBJ
- CBLSETCBLITEM
- CBLGETPROPERTY
- CBLSETVARIANT
- CBLPUTPROPERTY
- CBLOLE2UNINIT
- CBLMETHODCALL

使用例

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ARG1 PIC 9(9) USAGE COMP VALUE ZERO.
01 PRMINIT.
 02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.
    :
PROCEDURE DIVISION.
    :
    CALL 'CBLOLE2INIT' USING BY VALUE ARG1
                       BY REFERENCE PRMINIT.
```

(b) CBLCREATEOBJ

OLE2 サーバの OLE オブジェクトを生成します。

形式

```
CALL ’CBLCREATEOBJ’ USING BY VALUE 引数1
                          BY REFERENCE 引数2 引数3.
```

引数

- 引数 1 には，CBLOLE2INIT で設定した引数 1 を指定します。
- 引数 2 は，このサービスルーチンが生成した OLE2 サーバの OLE オブジェクトを格納する領域です。この領域はアドレスデータ項目で設定します。サービスルーチンの発行時は 0 を設定します。

  ```
  01 引数2 USAGE ADDRESS.
  ```

  次のサービスルーチンを呼び出す場合，この領域と同じ領域を先に設定しておく必要があります。

  ・CBLGETPROPERTY

  ・CBLPUTPROPERTY

  ・CBLMETHODCALL

  設定しないでこれらのサービスルーチンを呼び出した場合の結果は保証しません。
- 引数 3 は，次の集団項目を要求します。

  ```
  01 引数3.
   02 FILLER PIC 9(9) USAGE COMP VALUE ZERO. … 1.
   02 データ名1 PIC 9(9) USAGE COMP.        … 2.
   02 データ名2 USAGE ADDRESS.              … 3.
  ```

項目の説明

1. 4 バイトの 2 進項目で 0 を設定します。
2. 3.で指定する領域がポイントする領域の長さを，4 バイトの 2 進項目で設定します。
3. OLE2 サーバのクラス名を指定します。この領域は，アドレスデータ項目で定義します。この領域がポイントする領域は，次のような基本項目です（xx：クラス名を格納するのに必要なバイト数）。

   ```
   01 データ名 PIC X(xx).
   ```

戻り値

正常：0

上記以外：エラーコード値

使用例

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ARG1 PIC 9(9) USAGE COMP VALUE ZERO.
01 PRMOBJ USAGE ADDRESS.
01 PRMCREATEOBJ.
```

```
        02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.
        02 CLSLENG PIC 9(9) COMP VALUE 17.
        02 CLSNAMEPTR USAGE ADDRESS.
       01 CLASSNAME PIC X(17) VALUE 'Excel.Application'.
           :
       PROCEDURE DIVISION.
           COMPUTE PRMOBJ = ZERO.
           COMPUTE CLSNAMEPTR = FUNCTION ADDR(CLASSNAME).
           CALL 'CBLCREATEOBJ'
                USING BY VALUE ARG1
                      BY REFERENCE PRMOBJ PRMCREATEOBJ.
```

(c) CBLGETOBJ

OLE2 サーバの OLE オブジェクトをファイルから取得する場合，またはすでに OLE2 サーバが起動している場合に，その OLE2 サーバの OLE オブジェクトを取得するサービスルーチンです。

形式

```
CALL 'CBLGETOBJ' USING BY VALUE 引数1
                 BY REFERENCE 引数2 引数3.
```

引数

- 引数 1 には，CBLOLE2INIT で設定した引数 1 を指定します。
- 引数 2 は，このサービスルーチンが取得した OLE2 サーバの OLE オブジェクトを格納する領域です。この領域はアドレスデータ項目で設定します。サービスルーチンの発行時は 0 を設定します。

  ```
        01 引数2 USAGE ADDRESS.
  ```

  次のサービスルーチンを呼び出す場合，この領域と同じ領域を設定して呼び出す必要があります。設定しないで呼び出した場合の結果は保証しません。

  ・CBLGETPROPERTY

  ・CBLPUTPROPERTY

  ・CBLMETHODCALL

- 引数 3 は，次の集団項目を要求します。

Windows(x86) COBOL2002 の場合

```
       01 引数3.
         02 FILLER PIC 9(9) USAGE COMP VALUE ZERO. … 1.
         02 データ名1 PIC 9(9) USAGE COMP.        … 2.
         02 データ名2 USAGE ADDRESS.              … 3.
         02 データ名3 PIC 9(9) USAGE COMP.        … 4.
         02 データ名4 USAGE ADDRESS.              … 5.
```

Windows(x64) COBOL2002 の場合

```
       01 引数3.
         02 FILLER PIC 9(9) USAGE COMP VALUE ZERO. … 1.
         02 データ名1 PIC 9(9) USAGE COMP.        … 2.
         02 データ名2 USAGE ADDRESS.              … 3.
         02 データ名3 PIC 9(9) USAGE COMP.        … 4.
```

```
        02 FILLER PIC 9(9) USAGE COMP VALUE ZERO. … 1.
        02 データ名4 USAGE ADDRESS.              … 5.
```

項目の説明

1. 4 バイトの 2 進項目で 0 を設定します。
2. 3.で指定する領域がポイントする領域の長さを 4 バイトの 2 進項目で設定します。
3. 取得したい OLE オブジェクトが含まれているファイルの絶対パス名，およびファイル名を格納した領域のポインタを指定します。この領域は，アドレスデータ項目で定義します。この領域がポイントする領域は次のような基本項目です（xx：ファイル名を格納するのに必要なバイト数)。

```
01 データ名 PIC X(xx).
```

4. 5.で指定する領域がポイントする領域の長さを 4 バイトの 2 進項目で設定します。
5. OLE2 サーバのクラス名を格納した領域のポインタを指定します。この領域はアドレスデータ項目で定義します。この領域がポイントする領域は，次のような基本項目です（xx：クラス名を格納するのに必要なバイト数)。

```
    01 データ名 PIC X(xx).
```

戻り値

正常：0

上記以外：エラーコード値

使用例

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ARG1 PIC 9(9) USAGE COMP VALUE ZERO.
01 PRMOBJ USAGE ADDRESS.
01 PRMGETOBJ.
 02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.
 02 FILELENG PIC 9(9) USAGE COMP VALUE 16.
 02 FILENAMEPTR USAGE ADDRESS.
 02 CLSLENG PIC 9(9) USAGE COMP VALUE 11.
 02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.※
 02 CLSNAMEPTR USAGE ADDRESS.
01 FILENAME PIC X(16) VALUE 'c:¥temp¥data.xls'.
01 CLASSNAME PIC X(11) VALUE 'Excel.Sheet'.
    :
PROCEDURE DIVISION.
    COMPUTE PRMOBJ = ZERO.
    COMPUTE FILENAMEPTR = FUNCTION ADDR(FILENAME).
    COMPUTE CLSNAMEPTR = FUNCTION ADDR(CLASSNAME).
    CALL 'CBLGETOBJ'
         USING BY VALUE ARG1
               BY REFERENCE PRMOBJ PRMGETOBJ.
```

注※

Windows(x64) COBOL2002 では追加してください。

(d) CBLGETPROPERTY

OLE2 サーバの OLE オブジェクトの OLE プロパティの値を取得するサービスルーチンです。

形式

```
CALL 'CBLGETPROPERTY' USING BY VALUE 引数1
                            BY REFERENCE 引数2 引数3 引数4.
```

引数

- 引数 1 には，CBLOLE2INIT で設定した引数 1 を指定します。
- 引数 2 には，CBLCREATEOBJ および CBLGETOBJ サービスルーチンが生成・取得した OLE2 サーバの OLE オブジェクトを格納している領域を指定します。

  ```
  01 引数2 USAGE ADDRESS.
  ```

  この領域は，アドレスデータ項目で必ず指定してください。
- 引数 3 は，このサービスルーチンが，OLE2 サーバの OLE プロパティから取得した VARIANT 値のアドレスを設定します。初期値としては 0 を設定します。

  ```
  01 引数3 USAGE ADDRESS.
  ```

  この値は，CBLSETCBLITEM サービスルーチンで COBOL のデータ項目に設定できます。
- 引数 4 は，次の集団項目を要求します。

  ```
  01 引数4.
   02 FILLER PIC 9(9) USAGE COMP VALUE ZERO. … 1.
   02 データ名1 PIC 9(9) USAGE COMP.        … 2.
   02 データ名2 USAGE ADDRESS.              … 3.
  ```

項目の説明

1. 4 バイトの 2 進項目で 0 を設定します。
2. 3.で指定する領域がポイントする領域の長さを 4 バイトの 2 進項目で設定します。
3. OLE2 サーバの OLE プロパティおよびそれまでのコンテナ，コレクションをピリオドで区切った文字列を格納した領域のポインタを指定します。また，この領域はアドレスデータ項目で定義します。この領域がポインタする領域は次のような基本項目です（xx：文字列を格納するのに必要なバイト数）。

   ```
   01 データ名 PIC X(xx).
   ```

戻り値

正常：0

上記以外：エラーコード値

注意事項

3.で指定した領域がポイントする文字列の中で，ピリオドで区切られているいちばん右側の文字列が OLE プロパティと判断されます。また，これ以外はすべてコレクションまたはコンテナと判断されます。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ARG1 PIC 9(9) USAGE COMP VALUE ZERO.
       01 PRMOBJ USAGE ADDRESS.
       01 PRMVARIANT USAGE ADDRESS.
       01 PRMGETPROPERTY.
        02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.
        02 STRINGLENG PIC 9(9) USAGE COMP VALUE 43.
        02 STRINGNAMEPTR USAGE ADDRESS.
       01 STRINGNAME PIC X(43)VALUE
          'Workbooks(1).Worksheets(1).Cells(1,1). Value'.
           :
       PROCEDURE DIVISION.
           CALL 'CBLCREATEOBJ' USING …
           COMPUTE PRMVARIANT = ZERO.
           COMPUTE STRINGNAMEPTR
                   = FUNCTION ADDR(STRINGNAME).
           CALL 'CBLGETPROPERTY'
                USING BY VALUE ARG1
                      BY REFERENCE PRMOBJ
                                   PRMVARIANT
                                   PRMGETPROPERTY.
```

(e) CBLPUTPROPERTY

OLE2 サーバの OLE オブジェクトの OLE プロパティに値を設定するサービスルーチンです。

形式

```
CALL 'CBLPUTPROPERTY' USING BY VALUE 引数1
                    BY REFERENCE 引数2 引数3 引数4.
```

引数

- 引数 1 には，CBLOLE2INIT で設定した引数 1 を指定します。
- 引数 2 には，CBLCREATEOBJ および CBLGETOBJ サービスルーチンが生成，取得した OLE2 サーバの OLE オブジェクトを格納した領域を指定します。

  ```
  01 引数2 USAGE ADDRESS.
  ```

  この領域は，アドレスデータ項目で必ず指定してください。
- 引数 3 は，このサービスルーチンが OLE2 サーバの OLE プロパティに設定する VARIANT 値のアドレスを設定します。初期値としては 0 を設定します。

  ```
  01 引数3 USAGE ADDRESS.
  ```

  この領域は，アドレスデータ項目で定義します。

  この値は，設定する COBOL のデータ項目をあらかじめ CBLSETVARIANT サービスルーチンによって VARIANT 値に変換しておきます。
- 引数 4 は，次の集団項目を要求します。

```
01 引数4.
 02 FILLER PIC 9(9) USAGE COMP VALUE ZERO. … 1.
 02 データ名1 PIC 9(9) USAGE COMP.        … 2.
 02 データ名2 USAGE ADDRESS.              … 3.
```

項目の説明

1. 4 バイトの 2 進項目で 0 を設定します。

2. 3.で指定する領域がポイントする領域の長さを 4 バイトの 2 進項目で設定します。

3. OLE2 サーバの OLE プロパティおよびそれまでのコンテナ，コレクションをピリオドで区切った文字列を格納した領域のポインタを指定します。また，この領域はアドレスデータ項目で定義します。この領域がポインタする領域は次のような基本項目です（xx：文字列を格納するのに必要なバイト数)。

```
01 データ名 PIC X(xx).
```

戻り値

正常：0

上記以外：エラーコード値

注意事項

3.で指定した領域がポイントする文字列の中で，ピリオドで区切られているいちばん右側の文字列が OLE プロパティと判断されます。また，これ以外はすべてコレクションまたはコンテナと判断されます。

使用例

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ARG1 PIC 9(9) USAGE COMP VALUE ZERO.
01 PRMOBJ USAGE ADDRESS.
01 PRMVARIANT USAGE ADDRESS.
01 PRMPUTPROPERTY.
 02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.
 02 STRINGLENG PIC 9(9) USAGE COMP VALUE 43.
 02 STRINGNAMEPTR USAGE ADDRESS.
01 STRINGNAME PIC X(43) VALUE
   'Workbooks(1).Worksheets(1).Cells(1,1).Value'.
    :
PROCEDURE DIVISION.
    CALL 'CBLCREATEOBJ' USING …
    COMPUTE PRMVARIANT =
          (CBLSETVARIANTサービスルーチンで
           変換したVARIANT値).
    COMPUTE STRINGNAMEPTR
            = FUNCTION ADDR(STRINGNAME).
    CALL 'CBLPUTPROPERTY' USING
         BY VALUE ARG1
         BY REFERENCE PRMOBJ
                      PRMVARIANT
                      PRMPUTPROPERTY.
```

(f) CBLMETHODCALL

OLE2 サーバの OLE オブジェクトの OLE メソッドを操作するサービスルーチンです。

形式

```
CALL 'CBLMETHODCALL' USING BY VALUE 引数1
         BY REFERENCE 引数2 引数3 引数4.
```

引数

- 引数 1 には，CBLOLE2INIT で設定した引数 1 を指定します。
- 引数 2 には，CBLCREATEOBJ および CBLGETOBJ サービスルーチンが生成，取得した OLE2 サーバの OLE オブジェクトを格納した領域を指定します。

  ```
  01 引数2 USAGE ADDRESS.
  ```

  この領域は，アドレスデータ項目で必ず指定してください。
- 引数 3 は，このサービスルーチンが OLE2 サーバの OLE メソッドから返される値を設定する VARIANT 値のアドレスを設定します。

  ```
  01 引数3 USAGE ADDRESS.
  ```

  この領域はアドレスデータ項目で定義します。OLE メソッドが値を返さない場合は 0 を設定します。
- 引数 4 は，次の集団項目を要求します。

Windows(x86) COBOL2002 の場合

```
01 引数4.
 02 FILLER PIC 9(9) USAGE COMP VALUE ZERO. … 1.
 02 データ名1 PIC 9(9) USAGE COMP.        … 2.
 02 データ名2 USAGE ADDRESS.              … 3.
 02 データ名3 PIC 9(9) USAGE COMP.        … 4.
 02 データ名4 USAGE ADDRESS.              … 5.
```

Windows(x64) COBOL2002 の場合

```
01 引数4.
 02 FILLER PIC 9(9) USAGE COMP VALUE ZERO. … 1.
 02 データ名1 PIC 9(9) USAGE COMP.        … 2.
 02 データ名2 USAGE ADDRESS.              … 3.
 02 データ名3 PIC 9(9) USAGE COMP.        … 4.
 02 FILLER PIC 9(9) USAGE COMP VALUE ZERO. … 1.
 02 データ名4 USAGE ADDRESS.              … 5.
```

項目の説明

1. 4 バイトの 2 進項目で 0 を設定します。
2. 3.で指定する領域がポイントする領域の長さを 4 バイトの 2 進項目で設定します。
3. パラメタを含む該当するメソッド，およびそれまでのコンテナ，コレクションをピリオドで区切った文字列を格納した領域のポインタを指定します。また，この領域はアドレスデータ項目で定義します。

この領域がポイントする領域は次のような基本項目です（xx：文字列を格納するのに必要なバイト数）。

```
01 データ名 PIC X(xx).
```

4. この領域は，OLE メソッドに引数を指定する場合に，3.で指定した領域中の OLE メソッドの文字列にパラメタを含めて指定するのではなく，VARIANT 値の配列として指定するとき，5.で指定する引数の配列テーブルのエントリ数を設定します。

   3.で指定する領域中のメソッドの文字列にパラメタを含める場合，または OLE メソッドにパラメタがない場合は 0 を設定します。4 バイトの 2 進項目で設定します。

5. この領域は，OLE メソッドに引数を指定する場合に，3.で指定する領域中の OLE メソッドの文字列にパラメタを含めて指定するのではなく，VARIANT 値の配列として指定するとき，引数の配列テーブルのアドレスを設定します。

   この領域がポイントする配列テーブルは次のような集団項目です。

```
01 任意の名称.
  02 任意の名称1 USAGE ADDRESS.  ┐
     :                           ├ ……6.
  02 任意の名称n USAGE ADDRESS.  ┘
```

   このとき，6.で指定する領域がポイントする VARIANT 値は，あらかじめ COBOL のデータ項目を CBLSETVARIANT サービスルーチンで変換しておいてください。3.で指定する領域中の OLE メソッドの文字列にパラメタを含める場合，または OLE メソッドにパラメタがない場合は 0 を設定します。

戻り値

正常：0

上記以外：エラーコード値

注意事項

(a)

引数 4 の 3.がポイントする文字列の中で，ピリオドで区切られているいちばん右側の文字列が OLE メソッドと判断されます。これ以外はすべてコレクションまたはコンテナと判断されます。

括弧で囲まれている場合は，その OLE メソッドにパラメタが含まれていると判断され，引数 4 の 5.で指定した配列テーブルは無視されます。

複数のパラメタを指定する場合は「,」で区切らなければなりません。

(b)

引数 4 の 3.がポイントする文字列の OLE メソッドにパラメタを含めた場合，このサービスルーチンは次のように仮定します。

- 「'」または「"」で囲まれているパラメタは文字列とする。
- 0～9 と小数点のピリオドから構成される数値は倍精度浮動小数点とする。

(c)

パラメタの属性が(b)の方法で設定する仮定値と異なる場合は，引数４の 5.に VARIANT 値の配列テーブルポインタを設定して OLE メソッドのパラメタを指定してください。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ARG1 PIC 9(9) USAGE COMP VALUE ZERO.
       01 PRMOBJ USAGE ADDRESS.
       01 PRMVARIANT USAGE ADDRESS.
       01 PRMMETHODCALL.
        02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.
        02 STRINGLENG PIC 9(9) USAGE COMP VALUE 48.
        02 STRINGNAMEPTR USAGE ADDRESS.
        02 VARTBLCNT PIC 9(9) USAGE COMP VALUE ZERO.
        02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.※
        02 VARTBLPTR USAGE ADDRESS.
       01 STRINGNAME PIC X(48) VALUE
          'InputBox(''Please Input Data'',
      -   ''')COBOL InputBox'')'.
      *     :
       PROCEDURE DIVISION.
           COMPUTE PRMVARIANT = ZERO.
           COMPUTE VARTBLPTR = ZERO.
           COMPUTE STRINGNAMEPTR
                   = FUNCTION ADDR(STRINGNAME).
           CALL 'CBLMETHODCALL'
                USING BY VALUE ARG1
                      BY REFERENCE PRMOBJ
                                   PRMVARIANT
                                   PRMMETHODCALL.
```

注※

Windows(x64) COBOL2002 では追加してください。

(g) CBLVARIANTINF

CBLGETPROPERTY，CBLMETHODCALL，および CBLSETVARIANT サービスルーチンが得る VARIANT 値の属性を取得するサービスルーチンです。CBLVARIANTINF は，取得した属性の値によって CBLSETCBLITEM サービスルーチンでの処理を選択するのに使用します。

形式

```
CALL 'CBLVARIANTINF' USING BY VALUE 引数1
                     BY REFERENCE 引数2 引数3 引数4.
```

引数

- 引数 1 には，CBLOLE2INIT で設定した引数 1 を指定します。
- 引数 2 には，属性を取得する VARIANT 値を設定します。

```
       01 引数2 USAGE ADDRESS.
```

- 引数 3 には，VARIANT 値から取得した属性値が設定されます。

```
       01 引数3 PIC 9(9) USAGE COMP.
```

4 バイトの 2 進項目で定義します。初期値としては 0 を設定します。

この領域に返される属性値を次に示します。

0：Empty 値を表します。数値としては 0，文字列としては長さが 0 であることを表します。COBOL2002 では対応するデータ項目はありません。

1：Null 値を表します。有効な数値や文字列を持たないことを示します。値が 0 や Empty 値とは異なります。COBOL2002 では対応するデータ項目はありません。

2：整数型（Integer）を表します。COBOL2002 では符号付き 2 バイト 2 進項目を表します。

3：長整数型（Long）を表します。COBOL2002 では符号付き 4 バイト 2 進項目を表します。

4：単精度浮動小数点数型（Single）を表します。COBOL2002 では単精度内部浮動小数点項目を表します。

5：倍精度浮動小数点数型（Double）を表します。COBOL2002 では倍精度内部浮動小数点項目を表します。

6：通貨型（Currency）を表します。COBOL2002 では対応するデータ項目はありません。CBLSETCBLITEM または CBLSETVARIANT で変換する場合は，英数字項目を指定します。

7：日付型（Date）を表します。COBOL2002 では対応するデータ項目はありません。CBLSETCBLITEM または CBLSETVARIANT で変換する場合は，21 バイトの英数字項目を指定します。

8：文字列型を表します。COBOL2002 では対応するデータ項目はありません。CBLSETCBLITEM または CBLSETVARIANT で変換する場合は，英数字項目を指定します。

- 引数 4 は，次の集団項目を要求します。

```
       01 引数4.
        02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.
```

4 バイトの 2 進項目で 0 を設定します。

戻り値

正常：0

上記以外：エラーコード値

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ARG1 PIC 9(9) USAGE COMP VALUE ZERO.
       01 PRMOBJ USAGE ADDRESS.
       01 PRMVARIANT USAGE ADDRESS.
       01 PRMVARTYPE PIC 9(9) USAGE COMP VALUE ZERO.
       01 PRMGETPROPERTY.
        02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.
        02 STRINGLENG PIC 9(9) USAGE COMP VALUE 43.
        02 STRINGNAMEPTR USAGE ADDRESS.
       01 STRINGNAME PIC X(43) VALUE
```

```
        'Workbooks(1).Worksheets(1).Cells(1,1).Value'.
      01 PRMVARIANTINF.
       02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.
          :
      PROCEDURE DIVISION.
          CALL 'CBLCREATEOBJ' USING …
          COMPUTE PRMVARIANT = ZERO.
          COMPUTE STRINGNAMEPTR
                  = FUNCTION ADDR(STRINGNAME).
          CALL 'CBLGETPROPERTY'
                USING BY VALUE ARG1
                      BY REFERENCE PRMOBJ
                                   PRMVARIANT
                                   PRMGETPROPERTY.
          CALL 'CBLVARIANTINF'
               USING BY VALUE ARG1
                     BY REFERENCE PRMVARIANT
                                  PRMVARTYPE
                                  PRMVARIANTINF.
```

(h) CBLSETCBLITEM

CBLGETPROPERTY または CBLMETHODCALL サービスルーチンで取得した VARIANT の値を指定した COBOL のデータ項目に変換し，設定するサービスルーチンです。

形式

```
CALL 'CBLSETCBLITEM' USING BY VALUE 引数1
                           BY REFERENCE 引数2 引数3.
```

引数

- 引数 1 には，CBLOLE2INIT で設定した引数 1 を指定します。
- 引数 2 は，変換前の VARIANT 値を設定します。

  ```
        01 引数2 USAGE ADDRESS.
  ```

- 引数 3 は，次の集団項目を要求します。

  ```
        01 引数3.
         02 FILLER PIC 9(9) USAGE COMP VALUE ZERO. … 1.
         02 データ名1 PIC 9(9) USAGE COMP.        … 2.
         02 データ名2 USAGE ADDRESS.              … 3.
  ```

項目の説明

1. 4 バイトの 2 進項目で 0 を設定します。
2. COBOL データ項目に設定する VARIANT 値の属性値を設定します。4 バイトの 2 進項目で設定します。

   この領域に設定する値を次に示します。

   2：整数型（Integer）を表します。3.の領域がポイントする領域は次のような基本項目です。

   ```
     01 データ名 PIC S9(4) USAGE COMP.
   ```

3：長整数型（Long）を表します。3.の領域がポイントする領域は次のような基本項目です。

```
01 データ名 PIC S9(9) USAGE COMP.
```

4：単精度浮動小数点数型（Single）を表します。3.の領域がポイントする領域は次のような基本項目です。

```
01 データ名 COMP-1.
```

5：倍精度浮動小数点数型（Double）を表します。3.の領域がポイントする領域は次のような基本項目です。

```
01 データ名 COMP-2.
```

6：通貨型（Currency）を表します。3.の領域がポイントする領域は次のような集団項目です（xx：文字列を格納するのに必要なバイト数）。

```
01  データ名1.
  02  データ名2 PIC 9(9) USAGE COMP. … 4.
  02  データ名3 PIC X(xx).     … 5.
```

4.には，5.の領域長を設定します。このサービスルーチンを正常に終了した場合，5.の領域に格納された文字列長が設定されます。5.の領域に文字列が格納できなかった場合は，5.の領域として必要な文字列長が設定され，エラーとなります。

7：日付型（Date）を表します。3.の領域がポイントする領域は"6：通貨型"と同様な集団項目です。

8：文字列型を表します。3.の領域がポイントする領域は"6：通貨型"と同様な集団項目です。

3. 変換後の値を設定する COBOL データ項目領域のポインタです。アドレスデータ項目で定義します。この領域がポイントする領域については 2.を参照してください。

戻り値

正常：0

上記以外：エラーコード値

変換規則

VARIANT 値の属性と COBOL データ項目の属性とが異なる場合，変換はされません。

注意事項

- 変換できない COBOL のデータ項目を指定した場合，エラーとなります。
- 事前に CBLVARIANTINF サービスルーチンで設定する VARIANT 値を把握してから使用してください。
- 通貨型，日付型の場合，COBOL2002 では文字列として設定します。

使用例

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ARG1 PIC 9(9) USAGE COMP VALUE ZERO.
       01 PRMOBJ USAGE ADDRESS.
```

```
       01 PRMVARIANT USAGE ADDRESS.
       01 PRMSETCBLITEM.
        02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.
        02 VARTYPE PIC 9(9) USAGE COMP VALUE 8.
        02 CBLITEMPTR USAGE ADDRESS.
       01 CBLITEM.
        02 ITEMLENG PIC 9(9) USAGE COMP VALUE 255.
        02 ITEMSTRING PIC X(255).
           :
       PROCEDURE DIVISION.
           COMPUTE PRMVARIANT = VARIANT値.
           COMPUTE CBLITEMPTR = FUNCTION ADDR(CBLITEM).
           CALL 'CBLSETCBLITEM'
                USING BY VALUE ARG1
                      BY REFERENCE PRMVARIANT
                                   PRMSETCBLITEM.
```

(i) CBLSETVARIANT

CBLPUTPROPERTY または CBLMETHODCALL サービスルーチンで COBOL のデータ項目を設定するため，VARIANT 値に変換するサービスルーチンです。

形式

```
CALL 'CBLSETVARIANT' USING BY VALUE 引数1
                           BY REFERENCE 引数2 引数3.
```

引数

- 引数 1 には，CBLOLE2INIT で設定した引数 1 を指定します。
- 引数 2 には，このサービスルーチンが生成した変換後の VARIANT 領域のアドレスを設定します。
  この領域はアドレスデータ項目で定義します。また，初期値として 0 を設定します。

  ```
       01 引数2 USAGE ADDRESS.
  ```

- 引数 3 は，次の集団項目を要求します。

  ```
       01 引数3.
        02 FILLER PIC 9(9) USAGE COMP VALUE ZERO. … 1.
        02 データ名1 PIC 9(9) USAGE COMP.        … 2.
        02 データ名2 USAGE ADDRESS.              … 3.
  ```

項目の説明

1. 4 バイトの 2 進項目で 0 を設定します。
2. VARIANT 値に変換する COBOL データ項目の領域の属性値を設定します。4 バイトの 2 進項目で設定します。
   この領域に設定する値を次に示します。
   0：Empty 値を設定します。3.の領域がポイントする領域には 0 を設定します。
   1：Null 値を設定します。3.の領域がポイントする領域には 0 を設定します。

2：符号付き 2 バイト 2 進項目を表します。3.の領域がポイントする領域は次のような基本項目です。

```
01 データ名 PIC S9(4) USAGE COMP.
```

3：符号付き 4 バイト 2 進項目を表します。3.の領域がポイントする領域は次のような基本項目です。

```
01 データ名 PIC S9(9) USAGE COMP.
```

4：単精度内部浮動小数点を表します。3.の領域がポイントする領域は次のような基本項目です。

```
01 データ名 USAGE COMP-1.
```

5：倍精度内部浮動小数点を表します。3.の領域がポイントする領域は次のような基本項目です。

```
01 データ名 USAGE COMP-2.
```

6：通貨型（Currency）を表します。3.の領域がポイントする領域は次のような集団項目です（xx：文字列を格納するのに必要なバイト数）。

```
01 データ名1.
 02 データ名2 PIC 9(9) USAGE COMP. … 4.
 02 データ名3 PIC X(xx).     … 5.
```

4.には，5.の文字列のバイト数を設定します。

5.は文字列を格納します。

7：日付および時間（Date／Time）を表します。3.の領域がポイントする領域は"6：通貨型"と同様な集団項目です。

```
01 データ名 PIC X(21).
```

8：文字列型（英数字項目）を表します。3.の領域がポイントする領域は"6：通貨型"と同様な集団項目です

3. 変換する COBOL データ項目の領域のポインタです。アドレスデータ項目で定義します。この領域がポイントする領域については 2.を参照してください。

戻り値

正常：0

上記以外：エラーコード値

注意事項

通貨型，日付型の場合，COBOL2002 では文字列として設定します。

例えば，サーバが Excel の場合，日付型の設定では99/2/26 10:30 という形式で設定してください。

使用例

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ARG1 PIC 9(9) USAGE COMP VALUE ZERO.
01 PRMVARIANT USAGE ADDRESS.
01 PRMSETVARIANT.
```

```
        02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.
        02 CBLTYPE PIC 9(9) USAGE COMP VALUE 8.
        02 CBLITEMPTR USAGE ADDRESS.
       01 CBLITEM.
        02 ITEMLENG PIC 9(9) USAGE COMP VALUE 15.
        02 ITEMSTRING PIC X(15) VALUE 'OLE2 Automation'.
           :
       PROCEDURE DIVISION.
           COMPUTE PRMVARIANT = ZERO.
           COMPUTE CBLITEMPTR = FUNCTION ADDR(CBLITEM).
           CALL 'CBLSETVARIANT'
                USING BY VALUE ARG1
                      BY REFERENCE PRMVARIANT
                                   PRMSETVARIANT.
```

(j) CBLOLE2UNINIT

OLE2 を終了させるサービスルーチンです。このサービスルーチンでは，CBLCREATEOBJ や CBLGETOBJ が生成した OLE オブジェクトや取得した OLE オブジェクトを終了できません。その場合は，このサービスルーチンを発行する前に，Close や Quit など OLE2 オートメーションサーバ機能の終了メソッドを呼び出しておいてください。

また，このサービスルーチンを発行したあと，次のサービスルーチンを呼び出してはなりません。

- CBLCREATEOBJ
- CBLVARIANTINF
- CBLGETOBJ
- CBLOLE2UNINIT
- CBLGETPROPERTY
- CBLSETCBLITEM
- CBLPUTPROPERTY
- CBLSETVARIANT
- CBLMETHODCALL

形式

```
CALL 'CBLOLE2UNINIT' USING BY VALUE 引数1
                           BY REFERENCE 引数2.
```

引数

- 引数 1 には，CBLOLE2INIT で設定した引数 1 を指定します。
- 引数 2 は，次の集団項目を要求します。

  ```
       01 引数2.
        02 FILLER PIC 9(9) USAGE COMP VALUE ZERO.
  ```

  4 バイトの 2 進項目で 0 を設定します。

注意事項

COBOL で作成した OLE2 サーバに対して，このサービスルーチンを呼び出さないでください。

(3) エラーコード値

OLE2 オートメーションクライアント機能で使用するサービスルーチンが正常終了した場合，RETURN-CODE 特殊レジスタに 0 が戻されます。

サービスルーチンが異常終了した場合は，次のエラーコード値のどれかが戻されます。

表 D-2 エラーコード値

エラーコード値	エラーの内容
10	引数の値が NULL である。
13	ファイル名の拡張子がない。
14	プログラム ID が異なっている。
15	ファイル名，クラス名の指定が共にない。
18	Dispatch ポインタが取得できなかった。
19	数値を構成する要素に誤りがある。
20	BSTR 文字列を確保できなかった。
21	未サポートの VARIANT タイプが設定されている。
22	文字列を格納する領域が，文字列長よりも長い。
23	不当な日付が指定されている。
24	クラス名の指定がない。
25	サーバオブジェクトポインタが指定されていない。
26	OLE プロパティの指定がない。
27	OLE メソッドの指定がない
28	COBOL データ項目の指定がない。
29	OLE プロパティに引数が指定されている。
30	引数 VARIANT テーブルの指定がない。
31	VARIANT 値と COBOL データ項目の型が不一致である。
32	設定できない VARIANT 値が指定されている。
33	指定されたキーがオープンできない。
34	指定されたキーの値が取得できない。
35	指定されたキーがクローズできない。

エラーコード値	エラーの内容
その他の番号	OLE のシステムエラーが発生した。10 進数のエラーコード値から，OLE 関連のマニュアルでエラー状態を調べて対策する。

(4) 注意事項

OLE2 オートメーションクライアント機能全般に関する注意事項について説明します。

- OLE2 オートメーションインタフェース機能と OLE2 オートメーションクライアント機能で使用するサービスルーチンを，同じプログラム中で同時に使用してはいけません。
- 多くの OLE オブジェクト参照データ項目を連続して初期化すると，リソース不足になることがあります。

付録 E　コンパイルリスト

付録 E.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 E.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 E-1　ヘッダの出力形式

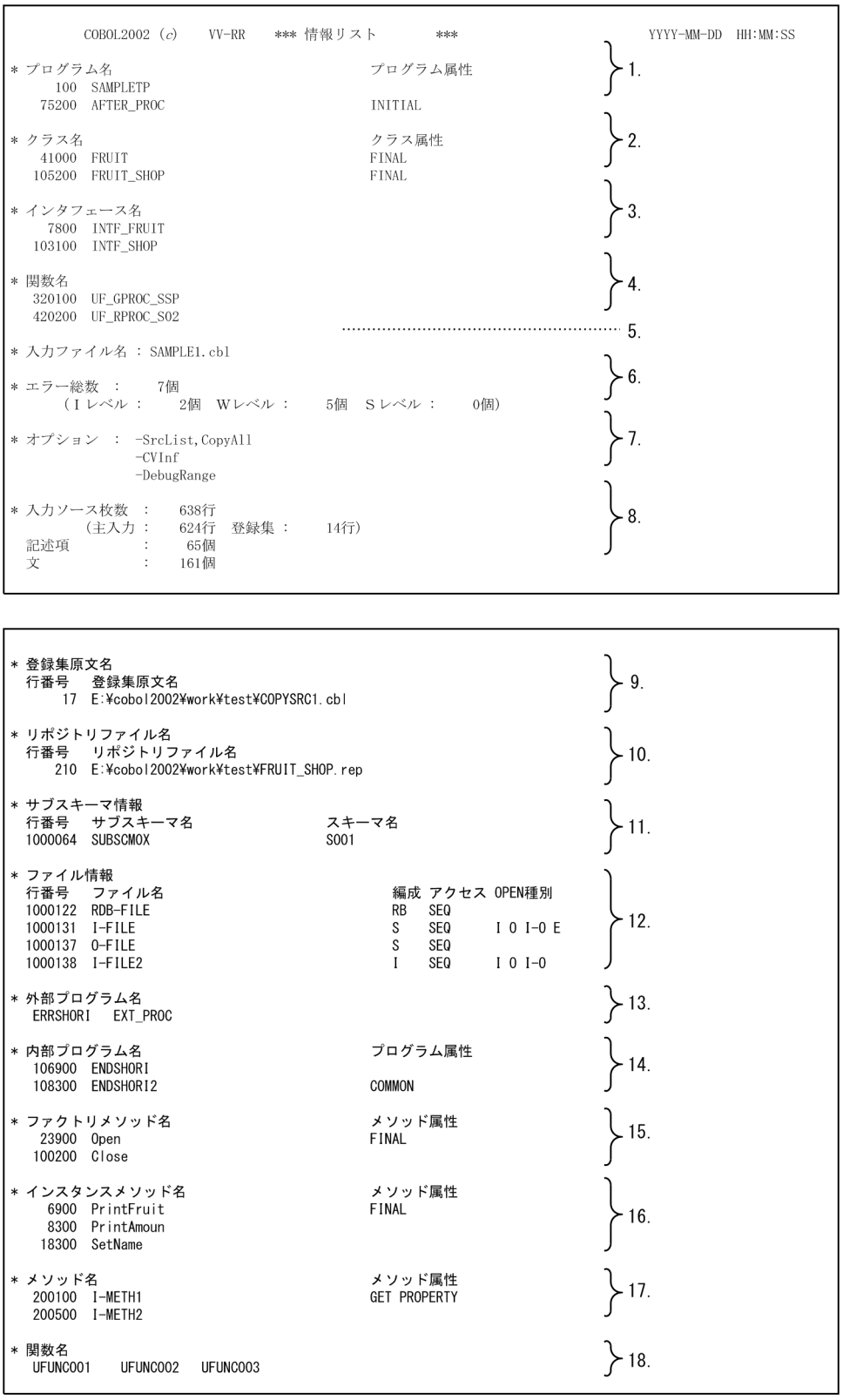
```
COBOL2002 (c)  VV-RR  *** CCC･･･CCC *** YYYY-MM-DD HH:MM:SS
    1.     2.    3.          4.             5.        6.
```

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 N.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 E-2　情報リストの出力形式



1. プログラム名

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

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

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，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

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

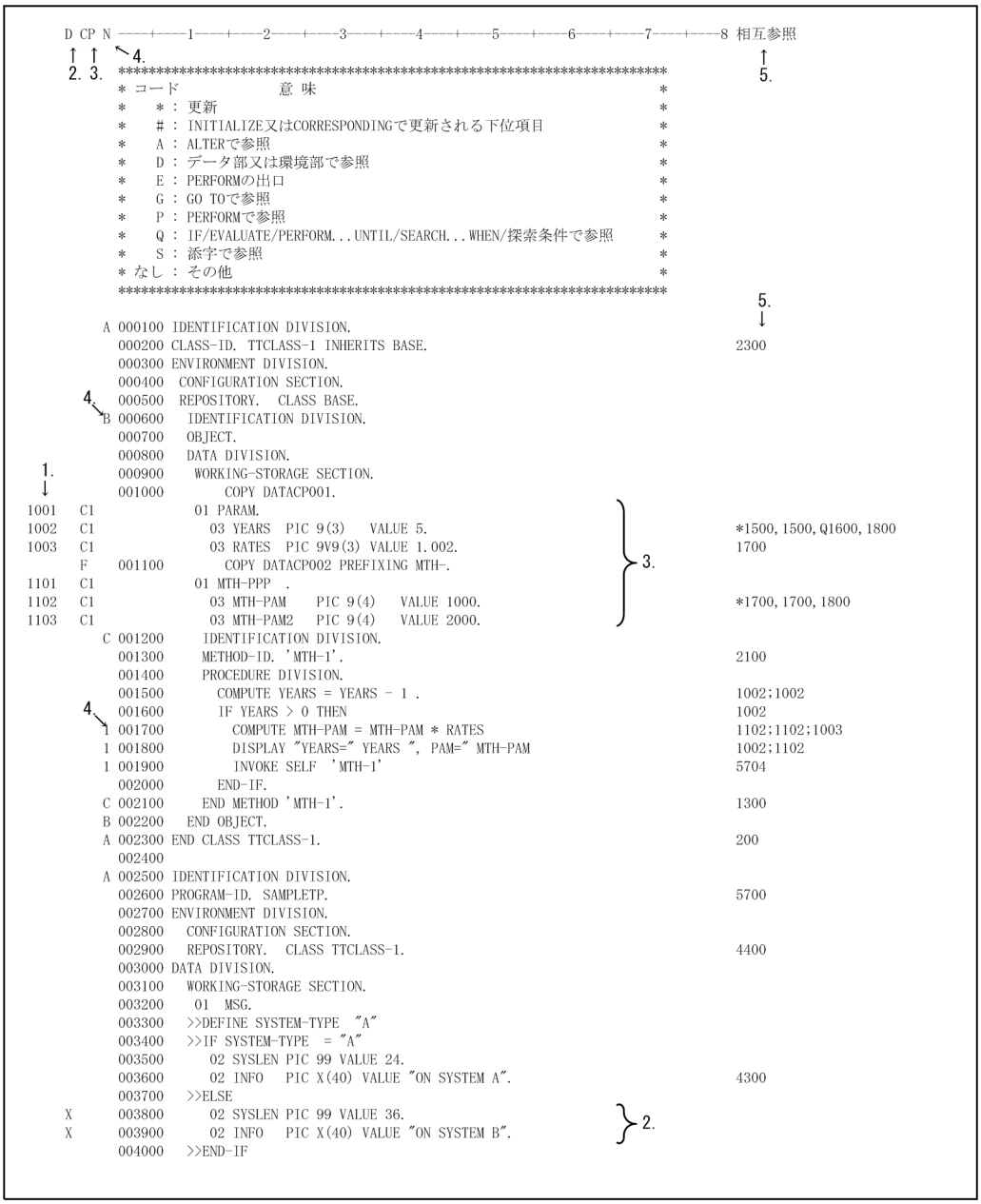
注意事項

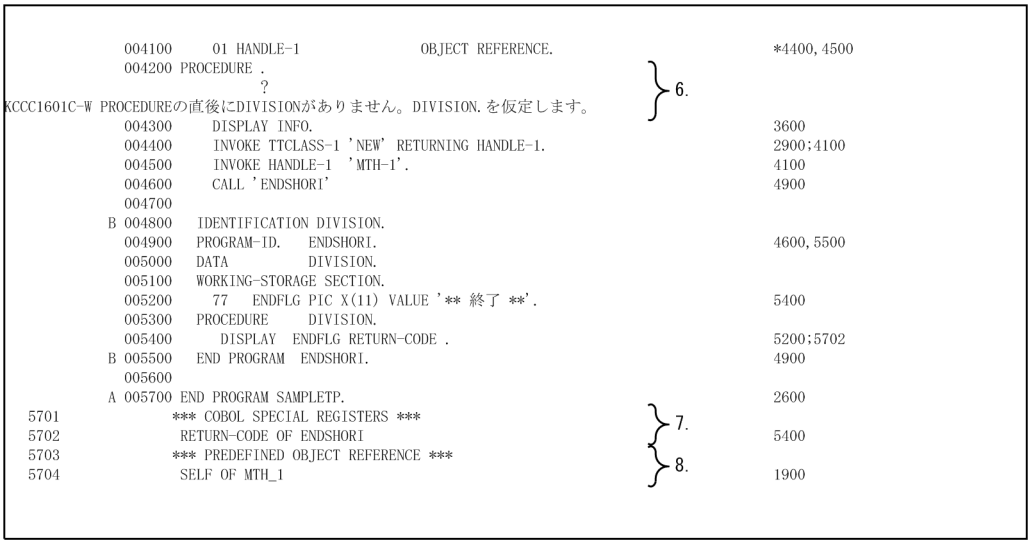
- 該当する定義がソース中にない場合，情報リストにその部分は出力されません。
- SQL 情報（SQL 構文内の実行文の数）は出力されません。
- ファイル／レコード定義の展開をする COPY 文の場合も，登録集原文の情報は出力されません。
- プログラム名などが長くなった場合には，次の行に折り返して出力されます。折り返した場合には，前行の開始位置と同じ位置から開始します。また，行番号は，折り返した行には出力されません。
- 全角文字が折り返しの境界にわたった場合，その全角文字は，次の行に出力されます。

(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，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 E-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。

   F

   ファイル／レコード定義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，「37.3　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。

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 E-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.
          0(00000010)     (0000000f) 350000        02  DATA31  OCCURS 10 DEPENDING ON DEP1.
3.       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).
          V(00000000)     (0000000a) 390000        02  DATA32       PIC X(10).
4.        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
5.                              560000  LOCAL-STORAGE SECTION.
           (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 E-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	○	○	－	－	－	○

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
連絡節 LINKAGE SECTION	×	×	−	−	−	×
サブスキーマ節 SUBSCHEMA SECTION	×	×	−	−	−	−
通信節 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

(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 E-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	○	○	○

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
	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	○	○	○
	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	−	○	○
OLE オブジェクト参照データ項目	USAGE OBJECT REFERENCE OLE	−	○	−
バリアントデータ項目	USAGE VARIANT	−	○	−
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 E-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 E-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    ................
```

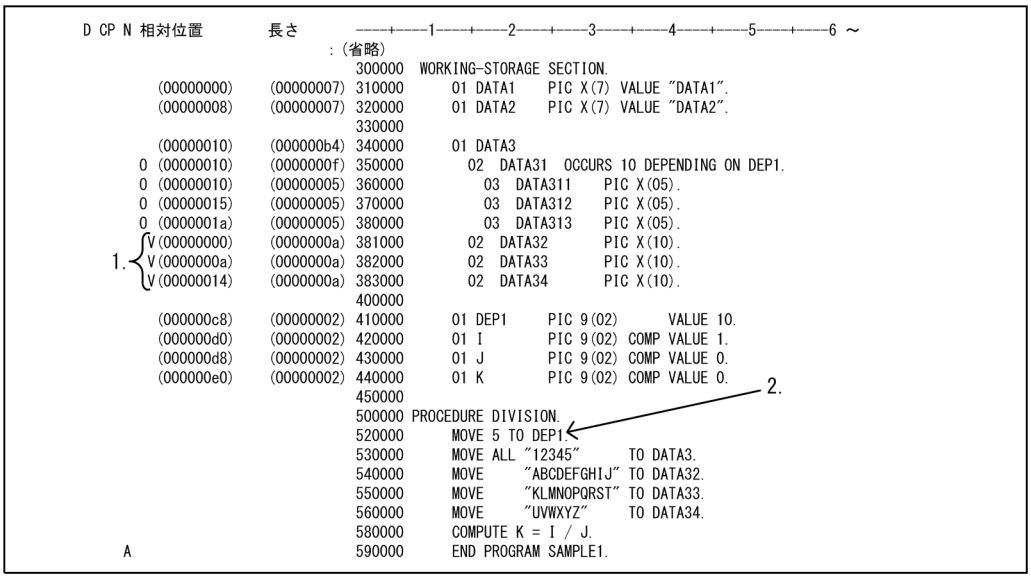
データ領域ダンプリストの<WORKING-STORAGE SECTION>の先頭から（0x00000022）（34バイト）目の内容を５バイト参照することで,「データ名：DATA31」の３番目の要素の内容を確認できます。

■ 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 E-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



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 E-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).
        {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                      ........
```

「データ名：DATA332」の3番目の要素の内容

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 E-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 E-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

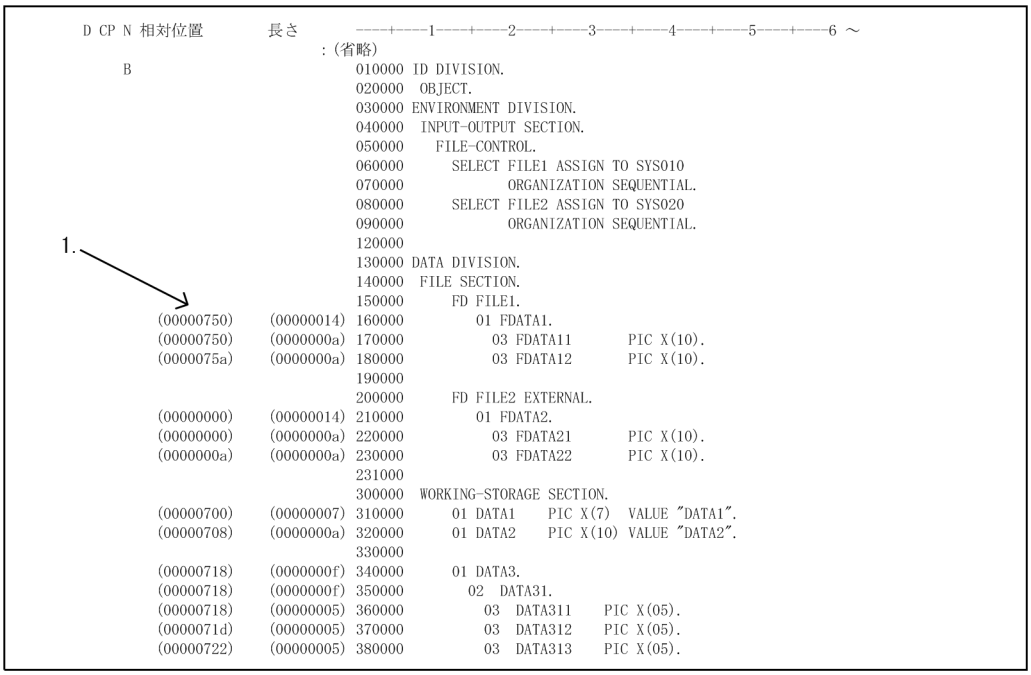
EXTERNAL 句を含むレコード記述項は，レコードの先頭から内容を参照できます。

■ 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 E-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

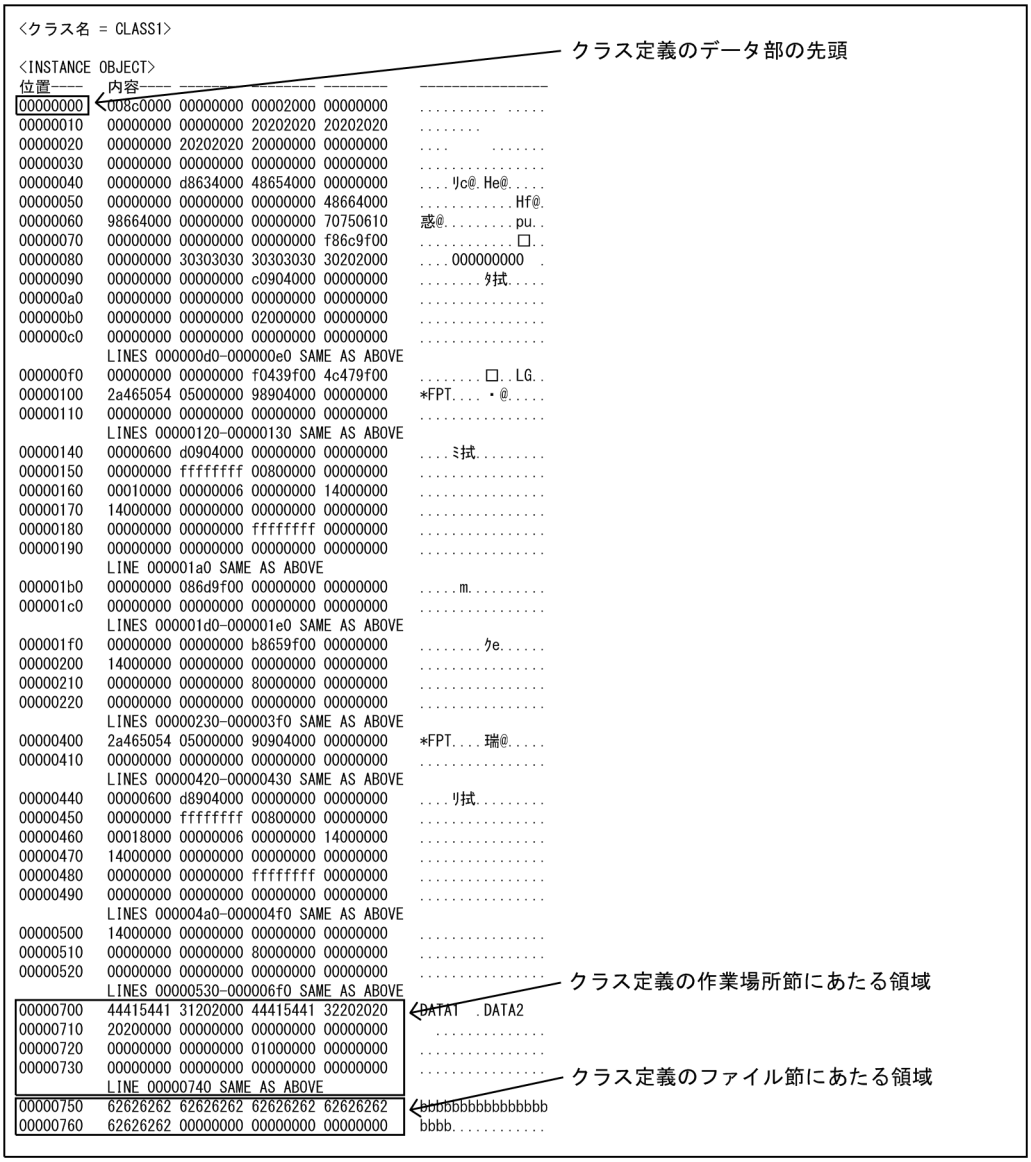


```
D CP N 相対位置        長さ        ----+----1----+----2----+----3----+----4----+----5----+----6 ～
                                 :(省略)
     B                           010000 ID DIVISION.
                                 020000  OBJECT.
                                 030000 ENVIRONMENT DIVISION.
                                 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............
```

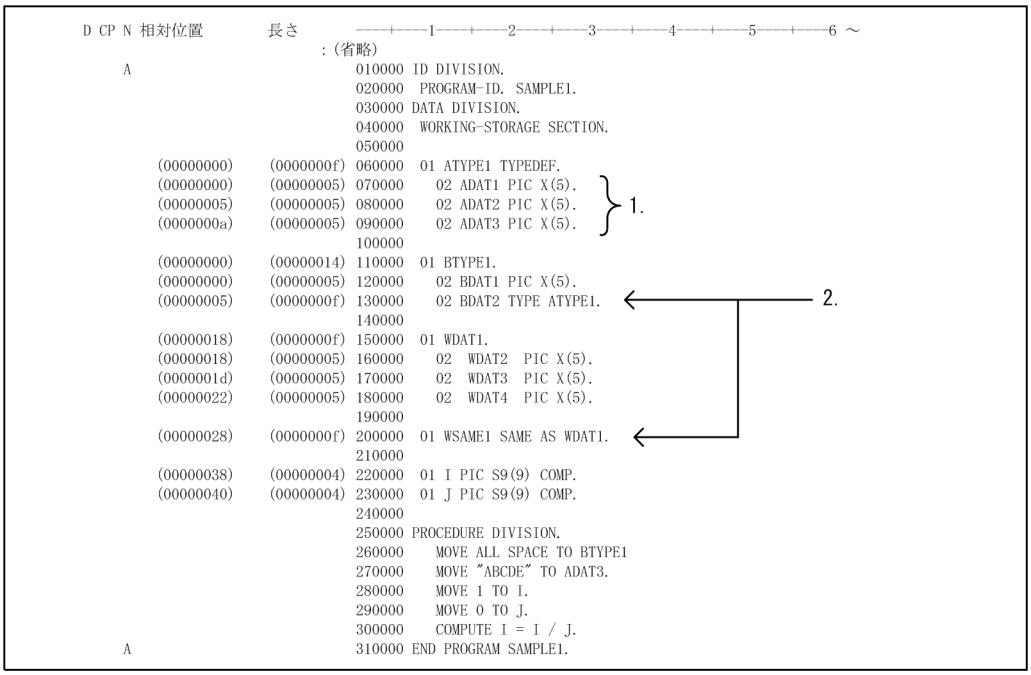
クラス定義の場合は，コンパイルリストでデータ部の先頭からの相対位置を表示しているため，データ領域ダンプリストも領域の先頭からの相対位置で参照できます。

■ 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 E-12　TYPE 句／SAME AS 句の定義がある原始プログラムリスト



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

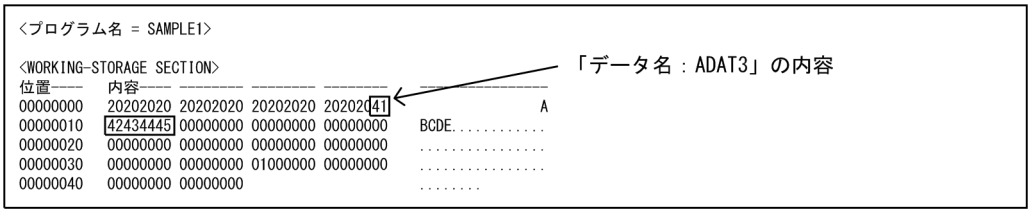
例：「データ名：ADAT3」の場合

「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 E-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 F　COBOL で使用するファイル

付録 F.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 F-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
.bkf	バックアップファイル	プロジェクトマスタファイルのバックアップ情報を格納するファイル。	マネージャ	マネージャ
.cbc	COPY 関連づけファイル	COPY プログラム変更時に再コンパイルするかどうかを認識するための情報を格納するファイル。	コンパイラ	マネージャ
.cbe	2 項関係情報ファイル	データの受け渡しに関係する 2 項関係情報を格納するファイル。	コンパイラ	COBOL2002 Professional Tool Kit（データ影響波及分析）
.cbd	EXPORTS 名称ファイル	モジュール定義ファイルを作成するための情報を格納するファイル。	コンパイラ	コンパイラ マネージャ
.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
.cbo	オブジェクト関連づけファイル	リンク時に必要なオブジェクトの情報を取得するためのファイル。	コンパイラ	マネージャ
.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

拡張子	ファイル種別	内容	出力元	入力先
.cbr	実行環境ファイル	設定した実行環境を記憶しておくファイル。	実行支援	実行支援，ライブラリ
.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
.cbw※5	stdcall 呼び出し指示ファイル	DLL ファイル中の stdcall 呼び出し規約のプログラムを呼び出すとき，呼び出すプログラムの名称を登録するためのファイル。	－	コンパイラ
.cet	プログラムテンプレートファイル	プログラムテンプレートを格納するファイル。	－	エディタ
.cex	COBOL エディタ設定ファイル	COBOL エディタのカスタマイズ情報を保存するファイル。	エディタ	エディタ
.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	－
.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
.def	モジュール定義ファイル	名前，属性，システムの条件など，実行可能ファイルや DLL の特性を記述したファイル。	コンパイラ	リンカ

拡張子	ファイル種別	内容	出力元	入力先
.dll※7	DLL ファイル	DLL（ダイナミックリンクライブラリ）を格納するファイル。	リンカ	COBOL2002 情報抽出ツール
.drf または .drl※8	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
.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
.exe※7	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	COBOL2002 情報抽出ツール
.exp	EXPORT ファイル	リンカが生成するエクスポートライブラリの出力ファイル。	リンカ	－
.flf	ファイル定義ファイル	ファイル定義の情報を格納するファイル。	F/R 定義	F/R 定義
.hmf	プロジェクトマスタファイル	プロジェクトマスタに関する情報を格納するファイル。	マネージャ	マネージャ
.hmp	プロジェクト情報ファイル	プロジェクトマスタから出力したプロジェクトの情報を格納するファイル。	マネージャ	マネージャ
.html または .csv	ソース解析情報ファイル	COBOL ソースを解析した結果を出力するファイル。	マネージャ	マネージャ
.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 または .l01※8	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
.k02～.k99 または .l02～.l99※8	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

拡張子	ファイル種別	内容	出力元	入力先
.kdf または .kdl※8	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， F/R 定義	ISAM
.lib	ライブラリファイル	ライブラリを格納するファイル。ライブラリには，オブジェクトプログラムのライブラリである標準ライブラリと，DLL の関数情報を保持するインポートライブラリがある。	リンカ， LIB コマンド	COBOL2002 情報抽出ツール
.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	－
.obj	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， COBOL2002 情報抽出ツール
.pkg	パッケージ情報ファイル	起動条件など，ツールの情報を格納するファイル。	マネージャ	マネージャ
.prt	印刷書式情報ファイル	印刷書式情報を格納するファイル。	実行支援	実行支援，ライブラリ
.rc	リソース定義ファイル	COBOL プログラムで使用するリソースの情報を記述したファイル。	コンパイラ	リンカ
.rdf	レコード定義ファイル	レコード定義の情報を格納するファイル。	F/R 定義	F/R 定義
.rep※6	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
.res	リソースファイル	リソース定義ファイルをリソースコンパイラでコンパイルした結果生成されるオブジェクトファイル。	リンカ	リンカ
.sai	ソース解析情報ファイル	COBOL ソースを解析した情報を出力するファイル。	コンパイラ	COBOL2002 Professional Tool Kit（COBOL ソース解析）
.svw	COBOL サービスルーチンファイル	COBOL エディタでキーワードとして有効にするサービスルーチンを登録するファイル。	エディタ	エディタ
.tag	タグファイル	タグジャンプ情報を格納するファイル。	エディタ	エディタ

拡張子	ファイル種別	内容	出力元	入力先
.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
.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
.usw	ユーザキーワードファイル	COBOL エディタで，予約語，サービスルーチン名以外でキーワードとする語句を登録するファイル。	エディタ	エディタ
.wdf	画面定義ファイル	画面定義の情報を格納するファイル。	画面定義	画面定義
.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。

マネージャ：開発マネージャ

エディタ：COBOL エディタ

ライブラリ：実行時ライブラリ

ISAM：ISAM ユティリティ

F/R 定義：ファイル／レコード定義

LIB コマンド：ライブラリ管理ツール LIB

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，.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。

注※5

Windows(x86) COBOL2002 で有効です。

注※6

Windows(x86) COBOL2002 で作成したファイルは，Windows(x64) COBOL2002 では使用できません。Windows(x64) COBOL2002 で再コンパイルしてから使用してください。

注※7

Windows(x86) COBOL2002 で作成した実行可能ファイルまたは DLL ファイルは，Windows(x64) COBOL2002 では使用できません。しかし，Windows(x64) COBOL2002 で作成した実行可能ファイルまたは DLL ファイルから CALL 文または CBLEXEC サービスルーチンを使用して，Windows(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，あらかじめ Windows(x86) COBOL2002 をインストールしてください。

注※8

Windows(x64) COBOL2002 では使用できません。

また，.kdl はファイル／レコード定義では出力されません。

付録 F.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 F-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
.cbr	実行環境ファイル	実行支援で実行環境を設定した場合
.dll※1	DLL ファイル	COBOL プログラムを実行可能ファイルとして作成した場合
.exe※1	実行可能ファイル	COBOL プログラムを DLL として作成した場合
.drf（.drl※2）	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 （　）内はラージファイル入出力機能を使用する場合
.kdf（.kdl※2）	キー定義ファイル	
.k01（.l01※2）	主キーファイル	
.k02～.k99 （.l02～.l99※2）	副キーファイル	

拡張子	ファイル種別	ファイルが必要な条件
.prt	印刷書式情報ファイル	GDI 印刷を使用する場合

注※1

Windows(x86) COBOL2002 で作成した実行可能ファイルまたは DLL ファイルは，Windows(x64) COBOL2002 では使用できません。しかし，Windows(x64) COBOL2002 で作成した実行可能ファイルまたは DLL ファイルから CALL 文または CBLEXEC サービスルーチンを使用して，Windows(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，あらかじめ，Windows(x86) COBOL2002 をインストールしてください。

注※2

Windows(x86) COBOL2002 で有効です。

付録 G　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 G-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※16
	固定小数点数字定数のけた数	1～18 けた※11
	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※15
	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※12
	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※14）
	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※16
	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※13
	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	Windows(x86) COBOL2002 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） Windows(x64) COBOL2002 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※17
	-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
	stdcall 呼び出し規約にするプログラムの個数（CALL 一意名がある場合）	1,700 個※7
	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
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
	整列併合用キーのサイズの合計	4,080 バイト

区分	項目	制限値，限界値
	整列併合用レコードのサイズ	65,535 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※9
	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
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※8
報告書作成機能	PAGE LIMIT 句に指定できる値	999 （-CompatiV3 指定時は，9,999）
	LINE NUMBER 句に指定できる値	999（-CompatiV3 指定時は，9,999）
	COLUMN 句に指定できる値	• RECORD 句がない時：133－印刷項目のサイズ+ 1 • RECORD 句がある時：RECORD 句に指定した最大レコード長－印刷項目のサイズ+ 1 注：CODE 句を指定したときは，指定した文字定数のサイズ分だけ印刷項目の配置位置が右端にずれるので，その分 COLUMN 句に指定できる制限値も少なくなります
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※10

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
- Btrieve（Pervasive.SQL）による索引編成ファイルを使用する場合：ページサイズに依存する。
  詳細は，「6.10.4　使用できる機能と制限事項」の「(2)　物理的制限事項」を参照してください。
- 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。
  詳細は，「6.9　HiRDB による索引編成ファイル」を参照してください。

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

次のプログラム数の合計値です。

- 外部プログラム節で，STDCALL または PASCAL に関連づけたプログラム
- stdcall 呼び出し指示ファイル(.cbw)に指定したプログラム
- stdcall 呼び出し規約の DLL 中の内側のプログラム

注※8

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※9

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※10

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，260 バイトです。

注※11

-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※12

-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※13

-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※14

-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※15

-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※16

-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※17

-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 H　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 H-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（索引編成ファイルだけ）。	○	×
	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（相対編成ファイルだけ）。	○ （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（索引編成ファイルだけ）。		（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（相対編成ファイル，および索引編成ファイル）。 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（相対編成ファイル，および索引編成ファイル）。 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（相対編成ファイルだけ）。 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（相対編成ファイルだけ）。	○ （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. 順固定長ファイルから入力した最後のレコードの長さが，レコード長より短い。 2. 順固定長ファイルに標準入力（stdin）を割り当てた場合，入力したレコードの長さがレコード長より短い（最大レコード長と最小レコード長の範囲外の値が設定されている）。 3. 相対可変長ファイルのレコード領域が破壊されている。 4. 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。 5. 入出力両用モードで開かれたテキスト編成ファイルに対して READ 文を実行したが，入力したテキスト行がプログラム中で定義したレコード領域より長い。 6. 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）。 7. 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，3.，4.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	次の場合のどちらかである。	○ （1 の場合だけ）	○ （1 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
		1. 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。 2. ISAM による索引編成ファイルで，指定した物理ファイルと実行時環境変数 CBLISAMLARGE，または CBLD_ファイル名で指定したファイルの形式が異なっている。		
	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順編成ファイル／相対編成ファイル／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。 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。	×	○ （1.，2.，および 3.の場合だけ）
	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）

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
	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	正しいオープンモードでは開かれていないファイル結合子に対して，WRITE 文を実行しようとした。次の場合のどちらかである。 1. 順アクセス法の場合，拡張モードや出力モードで開かれていないファイル結合子に WRITE 文を実行しようとした。 2. 動的アクセス法や乱アクセス法の場合，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結合子に 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. ファイルにフォルダが指定された。 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。 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. 同時に施錠できるレコード数の上限を超えた。 8. 索引編成ファイルに対して 2GB 以上のファイル入出力を実行した。※2 9. ノーマルファイル形式の索引編成ファイルに対して 2GB 以上のファイル入出力を実行した。※3 10. ラージファイルに対応した ISAM がインストールされていない。※3 11. ISAM のライブラリが検索パス上に存在しない。 12. ライブラリロード中にエラーが発生した。 13. 環境変数 CBLEUDCFUNC に ERRORSTOP を指定した環境で，GDI モード印刷または ESC/P モード印刷の OPEN 文を実行時に EnableEUDC 関数がエラーを返した。	○※4	○※5
	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）	○※6
	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。	×	○ （1.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. 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。		
	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

Windows(x64) COBOL2002 で有効です。

注※3

Windows(x86) COBOL2002 で有効です。

注※4

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。
- HiRDB システムがデッドロックやタイムアウトなどのエラーを検知した。

注※5

1.，2.，4.，5.，6.，7.，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※6

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 I　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	-Ad※1※5	−	OLE2 サーバの生成形態をインプロセスサーバに指定する。
	-Ax※5	−	OLE2 サーバの生成形態をアウトオブプロセスサーバに指定する。
	-Ag※2※5	−	OLE2 サーバの生成時，レジストリに登録する（xxxxx には，Dll または Exe を指定する）。
2	-Ai	-NumAccept	ACCEPT 文に数字項目を指定できるようにする。
3	-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）。
4	-Bb	-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	-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。
5	-Bt※5	-IsamExtend	Btrieve（Pervasive.SQL）による索引編成ファイルを使用する。
6	-Bz※5	-IsamExtend,Zone	Btrieve（Pervasive.SQL）による索引編成ファイルを使用する場合で，レコードキーに外部 10 進項目が指定されたときに，外部 10 進属性として処理する。
7	-C1	-Compile,CheckOnly	コンパイル時，オブジェクトファイル，実行可能ファイルおよび DLL を出力しない。
	-C2	-Compile,NoLink	コンパイル時，実行可能ファイルおよび DLL を出力しない。
8	-Ch	-V3Spec	VOS3 COBOL85 に対する COBOL2002 固有の構文をチェックする。
	-Cg	-V3Spec,CopyEased	VOS3 COBOL85 に対する 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。
9	-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。
10	-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。

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
	-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。
11	-Cp※5	-StdCall	stdcall 呼び出し指示ファイルを有効にする。
12	-Cs	-MainNotCBL	すべて副プログラムとして作成する。
13	-Ct	-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。
14	-d	-DebugLine	デバッグ行を有効にする。
15	-Dl※1※5	-Dll,Stdcall	DLL の形式を stdcall 呼び出し形式に指定する。
	-Dc※1※3※5	-Dll,Cdecl	DLL の形式を cdecl 呼び出し形式に指定する。
16	-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。
17	-Ec	-Switch,EBCDIC	字類条件を EBCDIC に指定する。
	-Ek	-Switch,EBCDIK	字類条件を EBCDIK に指定する。
	−	-Switch,xxxxx,Unprintable	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	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）。
18	-F8	-Cblctr	CBL-CTR 特殊レジスタを使用できるようにする。
19	-Fd	-DefFile	生成する.def ファイル名を指定する。
20	-Fi	-IconFile	アイコンファイル名を指定する。
21	-Fr	-ResrcFile	生成するリソース定義ファイル名を指定する。
22	-Fw※5	-StdCallFile	stdcall 呼び出し指示ファイル名を指定する。
23	-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。
24	-H8	-H8Switch	HITAC8000 シリーズの仕様でコンパイルする。
25	-Hd	-IgnoreLCC	行送り制御文字を無視する。
26	-Hf	-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。

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
	-Hv	-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。
27	−	-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。
28	−	-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）。
29	-I3	−	80386 用のマシン語を使用する。
	-I4	−	80486 用のマシン語を使用する。
30	-Ks※5	-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。
31	-Lg	-Lib,GUI	GUI モードの実行時ライブラリをリンクする。
	-Lc	-Lib,CUI	CUI モードの実行時ライブラリをリンクする。
32	-M1	-CmAster	1 カラム目が'*'の行を注記行とする。
33	-M7※5	-CompatiM7	MIOS7 COBOL85 との互換機能を有効にする。
34	-Mc	-UscoreStart	先頭が下線の CALL 定数を指定できるようにする。
35	-Md	-CmDol	7 カラム目が'$'の行を注記行とする。
36	-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。
	-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。
37	-Mt	-MultiThread	マルチスレッド対応 COBOL プログラムを作成する。
38	-Na	-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。
39	-Nb	-JPN,V3JPN	LENGTH 関数の引数，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	-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。

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
40	-o	-OutputFile	生成する実行可能ファイル名，または DLL ファイル名を指定する。
41	-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。
42	-Od※4	-SQLDisp	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。
43	-Oi	-SQL,ODBC	埋め込み SQL 文を ODBC で使用できるようにする。
	−	-SQL,ODBC,NoCont	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。
44	-Ot	-OpenTP1	OpenTP1 を使用したデータコミュニケーション機能を使用できるようにする。
45	-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。
	−	-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）。
46	-Pu	-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。
47	-Rd	-RDBTran	COMMIT 文／ROLLBACK 文を HiRDB による索引編成ファイルに対して適用する。
48	-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。

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
49	-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 進数で表示します。
	-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。
	-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。
50	-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。
	-Vv	-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。
51	-v	-Details	コンパイラオプションの詳細情報を出力する。
52	-V3	-CompatiV3	VOS3 COBOL85 との互換機能を有効にする。
53	-Va	-CBLVALUE	環境変数 CBLVALUE を有効にする。
54	-Vb 指定なし※1	-DllInit	DLL を初期化する。
55	-Vx	-BinExtend	2 進データ項目に指定できる初期値を拡張する。
56	-Wl	-Link	リンカに渡すオプションを指定する。
57	-Ws	-ScreenSpeed	画面の表示速度を重視する。
58	-X5	-Comp5	COMP-5 を指定できるようにする。

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
59	-Xb	-DigitsTrunc	転記文で上位けたを切り捨てる。
60	-Xc	-DoubleQuote	引用符（ " ）を分離符とみなしてコンパイルする。
61	-Xo	-TruncCheck	転記でのデータ切り捨てをチェックする。
62	-Xr	-SQL,XDM	埋め込み SQL 文を XDM/RD で使用できるようにする。
63	-?	-Help | -?	コマンドのヘルプを出力する。
64	−	-SimIdent	副プログラム（一意名 CALL 文）をシミュレーションする。
65	−	-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。
66	−	-Compati85,IoStatus	入出力状態 1x，2x に対する処理を PC COBOL85 と同様にする。
	−	-Compati85,Linage	LINAGE 値が不正なときの処理を PC 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 PC 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 PC COBOL85 と同様にする。
	−	-Compati85,Syntax	コンパイル時の解釈を PC 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	PC 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。
67	−	-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。
68	−	-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。
69	−	-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。
70	−	-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。

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
71	−	-OldForm	PC COBOL85 のオプションを指定する。
72	−	-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。
73	−	-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。
74	−	-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。
75	−	-DynamicLink,Call	定数指定の CALL 文で，静的にリンクされていないプログラムを動的にリンクして実行する。
76	−	-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。
77	−	-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。

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
78	－	-ManifestFileExt	マニフェストファイルを実行可能ファイル／DLL に埋め込まず，外部マニフェストファイルとする。
79	－	-IgnoreAPPLY,FILESHARE	APPLY FILE-SHARE 句を覚え書きとみなす。
80	－	-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。
	－	-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。
81	－	-SpaceAsZero	外部 10 進項目中に空白文字があるとき，ゼロとみなして比較，演算，転記を実行する。
82	－	-CheckUninitData	データ項目の初期化漏れをチェックする。
83	－	-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

ccbl2002 コマンドで-DllInit オプションと-Dll,Stdcall，または-Dll,Cdecl を同時に指定すると，ccbl コマンドで-Vb オプションを指定しないで-Dl，または-Dc オプションを指定した場合と同じ動作となります。

注※2

-OldForm オプションで-Ag オプションだけを指定した場合，-Ag オプションは無効となります。-Ag オプションを指定したい場合は，-Ad オプションまたは-Ax オプションを同時に指定してください。

注※3

-OldForm オプションで-Dc オプションだけを指定した場合，-Dc オプションはオプション指定エラーとなります。-Dc オプションを指定したい場合は，-Dl オプションを同時に指定してください。

また，-Dc オプションだけを指定して-Details オプションまたは-SrcList オプションでオプションを表示した場合，-Dll,Cdecl オプションが表示されます。

注※4

-OldForm オプションで-Od オプションだけを指定した場合，-Od オプションは無効となります。-Od オプションを指定したい場合は，-Oi オプションを同時に指定するか，-SQL,ODBC オプションを指定してください。

また，-Od オプションだけを指定して-Details オプションまたは-SrcList オプションでオプションを表示した場合，-SQLDisp オプションは表示されません。

注※5

Windows(x86) COBOL2002 で有効です。

注※6

Windows(x64) COBOL2002 で有効です。

付録 J　環境変数の設定

COBOL2002 を正常に動作させるためには，環境変数とフォルダを次のように設定する必要があります。

環境変数 PATH

実行可能ファイル（.exe）と DLL（.dll）を検索するフォルダの順序を指定します。環境変数 PATH に，次のフォルダを追加します。

- 日立共通 DLL のインストール先
- COBOL2002 のインストールフォルダ¥bin
- COBOL2002 のインストールフォルダ¥bin¥SDK
- Windows SDK の bin

環境変数 LIB

LINK コマンド，LIB コマンドで使用するライブラリのフォルダの順序を指定します。

環境変数 LIB に，次のフォルダを追加します。

- COBOL2002 のインストールフォルダ¥lib
- COBOL2002 のインストールフォルダ¥lib¥SDK
- Windows SDK の lib

環境変数 CBLLIB※

登録集原文を検索するフォルダを設定します。環境変数 CBLLIB に COBOL2002 のインストールフォルダ¥copy を追加します。

注※

XML 連携機能を使用する場合，環境変数 CBLLIB に COBOL2002 のインストールフォルダ¥copy を設定する必要があります。

ただし，COBOL2002 のコンポーネントおよびコマンドでは，その中で使用するコマンドやユーザプログラムを実行する前に，必要な環境変数が設定されます。したがって，上記の環境変数をユーザが設定しない場合でも，COBOL2002 のコンポーネントおよびコマンドで使用するコマンドやユーザプログラムは正しく動作します。

ここでは，COBOL2002 のコンポーネントおよびコマンドが行う環境設定やカスタマイズの方法について説明します。

付録 J.1　コンポーネントおよびコマンドでの環境設定

COBOL2002 のコンポーネントおよびコマンドは，その中で使用するコマンドやユーザプログラムを実行する前に必要な環境変数を設定します。これによって環境変数 PATH や環境変数 LIB などの環境変数が設定されていなくても，コンポーネントおよびコマンドで使用するコマンドや実行するユーザプログラムは正しく動作します。

表 J-1　コンポーネントおよびコマンドの環境設定

環境変数	設定内容
PATH	システムの環境変数 PATH の値の先頭に「日立共通 DLL のインストール先，COBOL2002 のインストールフォルダ¥bin，COBOL2002 のインストールフォルダ¥bin¥SDK，Windows SDK の bin へのパス」を付加した値に変更する。 `PATH=日立共通DLLのインストール先;COBOL2002のインストールフォルダ¥bin;COBOL2002のインストールフォルダ¥bin¥SDK;Windows SDKのbinへのパス;%PATH%`
LIB	システムの環境変数 LIB の値の先頭に「COBOL2002 のインストールフォルダ¥lib，COBOL2002 のインストールフォルダ¥lib¥SDK，Windows SDK の lib へのパス」を付加した値に変更する。 `LIB=COBOL2002のインストールフォルダ¥lib;COBOL2002のインストールフォルダ¥lib¥SDK;Windows SDKのlibへのパス;%LIB%`
CBLLIB	環境変数 CBLLIB の値の先頭に「COBOL2002 のインストールフォルダ¥copy」を付けた値を設定する。 `CBLLIB=COBOL2002のインストールフォルダ¥copy;%CBLLIB%`

環境設定を行うのはコマンドやユーザプログラムを実行するコンポーネントおよびコマンドです。

環境設定対象のコンポーネントおよびコマンドと設定する環境変数を次に示します。

表 J-2　環境設定対象のコンポーネント

コンポーネント	環境変数		
	PATH	LIB	CBLLIB
GUI モードのテストデバッガ	○	○	－
GUI モードのカバレージ	○	○	－
COBOL エディタ	○	○	○
開発マネージャ	○	○	○
ファイル／レコード定義	○	○	－
実行支援	○	○	－

（凡例）

○：変更する

－：変更しない

表 J-3　環境設定対象のコマンド

コマンド	環境変数		
	PATH	LIB	CBLLIB
cblbuild2k コマンド	○	○	○
cbltd2k コマンド	○	○	－
cblcn2k コマンド	△	△	－

コマンド	環境変数		
	PATH	LIB	CBLLIB
cblcv2k コマンド	△	△	－

（凡例）

○：変更する

△：連動実行を除き，変更する

－：変更しない

なお，「ANCHORID=KANKYOUSETTEITAISYOUKONPO【参照元】表 G-2【E】」と「ANCHORID=KANKYOUSETTEITAISYOUKOMANDO【参照元】表 G-3【E】」にないコンポーネントおよびコマンドでは，環境設定は行いません。

付録 J.2　環境設定のカスタマイズ

COBOL2002 のコンポーネントおよびコマンドでする環境設定の内容は，カスタマイズできます。環境設定をカスタマイズするための環境変数を次に示します。

表 J-4　カスタマイズ用の環境変数

環境変数	説明
CBL2KENV_PATH	環境変数 PATH に設定したフォルダとは異なるフォルダを設定する場合に使用する。
CBL2KENV_LIB	環境変数 LIB に設定したフォルダは異なるフォルダを設定する場合に使用する。
CBL2KENV_CBLLIB	環境変数 CBLLIB に設定したフォルダとは異なるフォルダを設定する場合に使用する。

(1)　環境変数 CBL2KENV_PATH

COBOL2002 のコンポーネントおよびコマンドを実行している間，環境変数 PATH の値を任意のフォルダに置き換えたい場合，環境変数 CBL2KENV_PATH を使います。この環境変数には，実行可能ファイルなどを検索するフォルダを設定します。フォルダを複数指定する場合は，セミコロン（;）で区切って指定します。ここで指定された値はチェックされません。

（例）

```
PATH=COBOL2002のインストールフォルダ¥bin;C:¥WINNT¥system32;C:¥WINNT
```

既存のパスを上記のパスに置き換えたい場合は，次のように指定します。

```
set CBL2KENV_PATH=C:¥WINNT¥system32;C:¥WINNT
```

実行支援や開発マネージャのプロジェクト設定ダイアログボックスでは，環境変数 CBL2KENV_PATH を指定しても有効になりません。

注意事項

- 環境変数 CBL2KENV_PATH を指定した場合，COBOL2002 のコンポーネントおよびコマンドでは，環境変数 PATH に設定された内容が無効となります。そのため，環境変数 CBL2KENV_PATH に，Windows のシステムフォルダを指定する必要があります。
- COBOL2002 のインストールフォルダ¥bin を指定する必要はありません。

(2) 環境変数 CBL2KENV_LIB

COBOL2002 のコンポーネントおよびコマンドを実行している間，環境変数 LIB の値を任意のフォルダに置き換えたい場合，環境変数 CBL2KENV_LIB を使います。環境変数 CBL2KENV_LIB には，ライブラリを検索するフォルダを設定します。フォルダを複数指定する場合は，セミコロン（;）で区切って指定します。検索するフォルダがない場合，%を指定します。ここで指定された値はチェックされません。

（例）

LIB=COBOL2002 のインストールフォルダ¥lib;C:¥Lib と置き換えたい場合，1.のように指定します。

LIB=COBOL2002 のインストールフォルダ¥lib と置き換えたい場合，2.のように指定します。

```
set CBL2KENV_LIB=C:¥Lib      … 1.
set CBL2KENV_LIB=%           … 2.
```

実行支援や開発マネージャのプロジェクト設定ダイアログボックスで環境変数 CBL2KENV_LIB を指定しても有効になりません。

注意事項

- 環境変数 CBL2KENV_LIB に，COBOL2002 のインストールフォルダ¥lib を指定する必要はありません。

(3) 環境変数 CBL2KENV_CBLLIB

COBOL2002 のコンポーネントおよびコマンドを実行している間，環境変数 CBLLIB の値を任意のフォルダに置き換えたい場合，環境変数 CBL2KENV_CBLLIB を使います。環境変数 CBL2KENV_CBLLIB には，登録集原文を検索するフォルダを設定します。フォルダを複数指定する場合は，セミコロン（;）で区切って指定します。検索するフォルダがない場合，%を指定します。ここで指定された値はチェックされません。

（例）

CBLLIB=COBOL2002 のインストールフォルダ¥copy;C:¥Include と設定したい場合，1.のように指定します。

CBLLIB=COBOL2002 のインストールフォルダ¥copy と設定したい場合，2.のように指定します。

```
set CBL2KENV_CBLLIB=C:¥Include      … 1.
set CBL2KENV_CBLLIB=%               … 2.
```

実行支援や開発マネージャのプロジェクト設定ダイアログボックスで環境変数 CBL2KENV_CBLLIB を指定しても有効になりません。

注意事項

- 環境変数 CBL2KENV_CBLLIB に，COBOL2002 のインストールフォルダ¥copy を指定する必要はありません。

付録 J.3　環境変数設定時およびカスタマイズ時の注意事項

コンポーネントおよびコマンドの環境設定やカスタマイズは，コンポーネントおよびコマンドの起動中だけ有効になります。エクスプローラなどからユーザプログラムを起動した場合は有効になりません。

付録 K　COBOL コマンドプロンプト

COBOL コマンドプロンプトは，COBOL2002 のコマンドやユーザプログラムを正しく実行するために必要な環境変数が設定されたコマンドプロンプトです。

付録 K.1　COBOL コマンドプロンプトの起動と終了

(1)　起動方法

Windows のプログラム一覧にある［COBOL2002］下のメニューから［COBOL コマンドプロンプト for COBOL2002］を選びます。

COBOL コマンドプロンプトの画面例を次に示します。

起動時のフォルダは，COBOL2002 のインストールフォルダとなります。

(2)　終了方法

COBOL コマンドプロンプトのシステムメニューの［閉じる］を選びます。

付録 K.2　COBOL コマンドプロンプトの環境変数の状態

COBOL コマンドプロンプト中の環境変数は，次のように設定されています。また，この COBOL コマンドプロンプトの環境変数はカスタマイズできます。環境設定のカスタマイズの詳細については，「付録 J 環境変数の設定」の「付録 J.2　環境設定のカスタマイズ」を参照してください。

表 K-1　コマンドプロンプトの環境設定

環境変数	設定内容
PATH	システムの環境変数 PATH の値の先頭に「日立共通 DLL のインストール先，COBOL2002 のインストールフォルダ¥bin，COBOL2002 のインストールフォルダ¥bin¥SDK」を付加した値に変更する。

環境変数	設定内容
	PATH=日立共通DLLのインストール先;COBOL2002のインストールフォルダ¥bin;COBOL2002のインストールフォルダ¥bin¥SDK;%PATH%
LIB	システムの環境変数 LIB の値の先頭に「COBOL2002 のインストールフォルダ¥lib，COBOL2002 のインストールフォルダ¥lib¥SDK」を付加した値に変更する。 LIB=COBOL2002のインストールフォルダ¥lib;COBOL2002のインストールフォルダ¥lib¥SDK;%LIB%
CBLLIB※	環境変数 CBLLIB の値の先頭に「COBOL2002 のインストールフォルダ¥copy」を付けた値を設定する。 CBLLIB=COBOL2002のインストールフォルダ¥copy;%CBLLIB%

注※

「COBOL2002 Net Client Runtime」，および「COBOL2002 Net Server Runtime」で使用できる COBOL コマンドプロンプトでは設定されません。

環境変数の値の設定が正しく行われていない場合，環境変数が追加されない状態でコマンドプロンプトが起動されます。起動されたコマンドプロンプトには，環境変数の設定に失敗した内容を示すメッセージが表示されます。環境変数に設定されている指定を減らすか，または環境設定のカスタマイズを行ってください。その後，再度 COBOL コマンドプロンプトを実行してください。

付録 L　トラブルシュートに有効な資料

COBOL2002 の COBOL プログラムで障害が発生した場合，調査に有効な資料について説明します。トラブルシュートに有効な資料を次の表に示します。

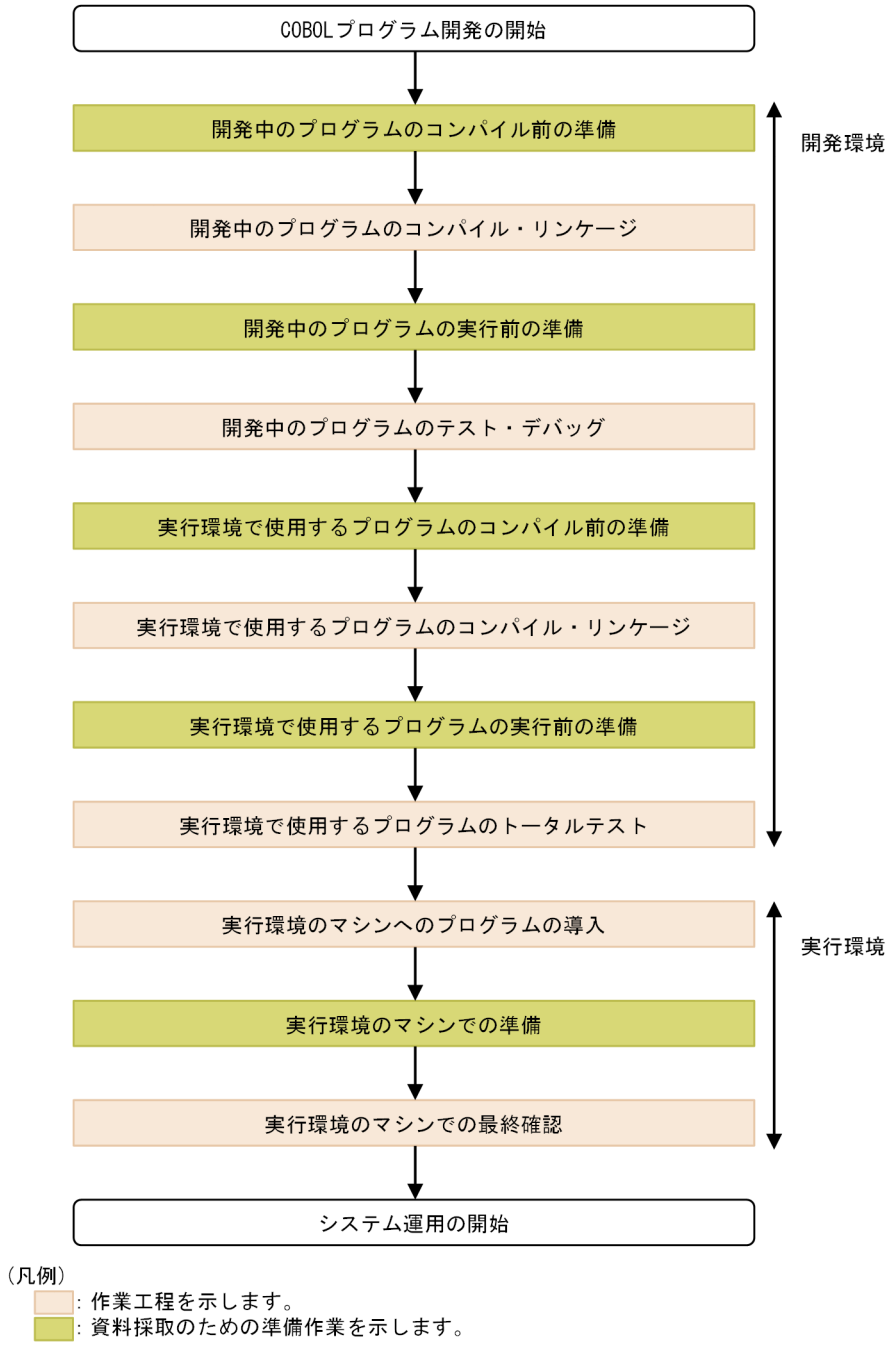
表 L-1　トラブルシュートに有効な資料一覧

資料名	概要説明	資料の参照先
コンパイラメッセージ	COBOL プログラムのコンパイル時に出力されるエラーメッセージです。	マニュアル「COBOL2002 メッセージ」
コンパイルリスト	コンパイラが出力するリストです。次の情報が含まれます。 • 情報リスト（コンパイル時に指定された COBOL2002 のコンパイラオプションを含む） • 原始プログラムリスト • エラーリスト（コンパイル時のエラーメッセージのリスト）	付録 E　コンパイルリスト
実行時メッセージ	COBOL プログラムの実行時に出力されるエラーメッセージです。	マニュアル「COBOL2002 メッセージ」
異常終了時要約情報リスト	COBOL プログラムが異常終了したときに出力されるリストです。 異常終了した COBOL プログラムの名称や行番号，異常終了するまでのトレース情報が出力されます。	37.2　異常終了時要約情報リスト
データ領域ダンプリスト	COBOL 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が出力されるリストです。 異常終了するまでに実行された COBOL プログラムのファイル節，作業場所節，局所場所節のダンプが出力されます。	37.3　データ領域ダンプリスト
プログラム検索トレースファイル	動的なリンクによる呼び出しで，呼び出し先プログラムの検索トレースが出力されるファイルです。	18.6.4　動的なリンクのプログラム検索トレース機能
情報抽出ツール出力結果リスト	COBOL2002 情報抽出ツールが出力するリストです。ツール実行時に指定する入力ファイルに対応した情報が出力されます。	マニュアル「COBOL2002 操作ガイド」
環境変数の設定情報	COBOL プログラムを実行するときに設定されている環境変数の情報です。	36.2　プログラムの実行環境の設定
メイクファイル	COBOL プログラムのコンパイルやリンケージの方法を記述したファイルです。バッチファイルなどで代用している場合もあります。	Microsoft のドキュメント
プロセスダンプ	ダンプが出力された時点でのプロセス情報が出力されます。プロセス情報には，実行可能ファイルや DLL ファイルの情報，スタックやヒープなどメモリの内容が含まれます。	Microsoft のドキュメント

付録 L.1　資料採取の準備

資料を採取するには，COBOL プログラム開発の開始からシステム運用の開始までの各作業工程のコンパイル前や，実行前に準備が必要です。COBOL プログラム開発の開始からシステム運用の開始までの作業の流れを次の図に示します。

図 L-1　COBOL プログラム開発の開始からシステム運用の開始までの作業の流れ



（凡例）
：作業工程を示します。
：資料採取のための準備作業を示します。

(1)　開発環境のマシンでの準備

(a)　開発中のプログラムのコンパイル前の準備

テスト用に，開発中の COBOL プログラムをコンパイル・リンケージする前に，コンパイル時に指定するコンパイラオプションを確認してください。コンパイラオプションについては，「33.5　コンパイラオプション」を参照してください。

- コンパイラオプションに-SrcList オプションを指定することで，コンパイルリストを出力できます。データ部のデータ項目の相対位置，および登録集原文を含んだコンパイルリストを出力するために，-SrcList,CopyAll,DataLoc オプションを指定することをお勧めします。
- コンパイラオプションに，次のどれかのデバッグオプションを指定することで，COBOL プログラムが異常終了したときに異常終了時要約情報リストやデータ領域ダンプリストを出力できます。

  -DebugInf，-DebugInf,Trace，-DebugCompati，-DebugData，-TDInf，-CVInf，-DebugRange

  開発中は，-DebugData オプションや，-DebugRange オプションを指定することをお勧めします。なお，テストデバッグ機能を使用する場合は-TDInf オプション，カバレージ機能を使用する場合は-CVInf オプションをコンパイラオプションに指定してください。
- メイクファイルやコンパイルリストは障害調査に有効な資料なので，開発後も保管してください。

(b)　開発中のプログラムの実行前の準備

コンパイル・リンケージで作成した COBOL プログラムをテスト・デバッグする前に，実行時環境変数の設定を確認してください。実行時環境変数については，「36.2　プログラムの実行環境の設定」を参照してください。

- 実行時エラーメッセージは，環境変数 CBL_SYSERR に出力先ファイルを指定して，ファイルに出力することをお勧めします。

  環境変数 CBL_SYSERR の指定がない場合，実行時エラーメッセージは，COBOL プログラムの動作モードが GUI モードのときは COBOL2002 が出力するコンソール画面に，CUI モードのときは標準エラー出力（stderr）に出力されます。
- 異常終了時要約情報リストは，環境変数 CBLABNLST に出力先ファイルを指定して，ファイルに出力することをお勧めします。

  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，COBOL プログラムの動作モードが GUI モードのときは COBOL2002 が出力するコンソール画面に，CUI モードのときは標準エラー出力（stderr）に出力されます。
- データ領域ダンプリストは，環境変数 CBLDDUMP に出力先ファイルを指定して，ファイルに出力することをお勧めします。

  環境変数 CBLDDUMP の指定がない場合，および環境変数に出力先ファイルを指定していない("CBLDDUMP="だけを指定した）場合，データ領域ダンプリストは出力されません。

- プログラム検索トレースファイルは，環境変数 CBLPGMSEARCHTRC に出力先ファイルを指定して，ファイルに出力することをお勧めします。プログラム検索トレースファイルは，動的なリンクによる呼び出しで，呼び出し先が見つからないなど期待する動作にならない場合の調査に有効です。

  環境変数 CBLPGMSEARCHTRC の指定がない場合，プログラム検索トレースファイルは出力されません。

- プロセスダンプは必須ではありませんが，領域破壊などによってプログラムが期待する動作にならない場合の調査に有効なので出力することお勧めします。プロセスダンプは，次の方法で出力できます。

出力方法	説明
Windows のタスクマネージャを使用する	この方法は，対話形式で操作できる場合に有効です。終了しないプログラムに対するプロセスダンプも出力できます。
レジストリにプロセスダンプを出力するための設定を追加する	この方法は，再現性の低い障害に対して，プロセスダンプを出力したい場合に有効です。事前に設定しておくことで，プログラムがシステム例外によって異常終了した場合にプロセスダンプが出力できます。
Microsoft などから提供されるツールを使用する	ツールの仕様に従います。

ここでは，レジストリにプロセスダンプを出力するための設定を追加する方法について説明します。レジストリを変更する場合の注意点を次に示します。

- レジストリに誤った設定をした場合，Windows システムに対して重大な問題を引き起こすおそれがあるため，誤った設定をしないように注意してください。
- レジストリにプロセスダンプを出力するための設定を追加した場合は，すべてのアプリケーションプログラムでの障害時にプロセスダンプが出力されるようになります。
- プロセスダンプを出力する場合，その分ディスク容量が圧迫されます。プロセスダンプの出力先には，十分なディスク領域が確保されている出力先フォルダを設定してください。

プロセスダンプを出力する場合の設定手順を次に示します。

1. レジストリエディタ（regedit.exe）を起動する
2. 次のレジストリキーを選択する

   HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Windows¥Windows Error Reporting¥LocalDumps
3. 手順 2.のレジストリキーに対して，次の値を追加する

   値名が作成済みの場合は，データだけ変更してください。

値名	レジストリキーのデータの種類	値名へ割り当てられるデータ
DumpCount	DWORD(32bit)値 （REG_DWORD）	プロセスダンプの出力回数を指定する。指定した回数を超えた場合，古いプロセスダンプを削除して出力する。
DumpFolder	展開可能な文字列値 （REG_EXPAND_SZ）	プロセスダンプの出力先フォルダを絶対パスで指定する。ファイル名は指定できない。
DumpType	DWORD（32bit）値	プロセスダンプの種類として「2」を指定する。次の値が指定できる。

値名	レジストリキーのデータの種類	値名へ割り当てられるデータ
	(REG_DWORD)	• 0：カスタムダンプ • 1：ミニダンプ • 2：フルダンプ

4. Windows サービスの「Windows Error Reporting Service」が起動されていない場合は，起動する
5. COBOL プログラム実行時は，環境変数 CBLEXCEPT に THROW を指定する

(c) 実行環境で使用するプログラムのコンパイル前の準備

実行環境で使用する COBOL プログラムをコンパイル・リンケージする前に，コンパイル時に指定するコンパイラオプションを確認してください。コンパイラオプションについては，「33.5 コンパイラオプション」を参照してください。

- テスト用に指定していたコンパイラオプションを解除してリコンパイルします。デバッグオプションを指定する場合は，-DebugInf オプションだけ指定することをお勧めします。

  -DebugInf オプション以外のデバッグオプションを指定した場合，実行性能に影響があります。デバッグオプションは，十分に性能評価した上で指定してください。
- -SrcList オプションは，生成されるオブジェクトに影響しません。コンパイルリストを出力するため，実行環境で使用する COBOL プログラムをコンパイルする場合も，-SrcList オプションは指定してください。
- メイクファイルやコンパイルリストは障害調査に有効な資料なので，システム稼働後も保管してください。

(d) 実行環境で使用するプログラムの実行前の準備

コンパイル・リンケージで作成した COBOL プログラムを実行環境に導入する前に，実行時環境変数の設定を確認してください。実行時環境変数については，「36.2 プログラムの実行環境の設定」を参照してください。

- プログラム検索トレースファイルの出力は実行性能に影響するため，実行環境では環境変数 CBLPGMSEARCHTRC の指定を解除してください。

(2) 実行環境のマシンでの準備

システムの運用を開始する前に，マシン環境を確認してください。

- 開発環境のマシンと，実行環境のマシンが同じフォルダ構成になっているか確認してください。フォルダ構成が異なる場合，パスを指定している環境変数は，実行環境のマシンのフォルダ構成に合わせて値を変更してください。
- 開発環境のマシンと，実行環境のマシンで OS が異なる場合，COBOL2002 のリリースノートやマニュアルを参照し，使用している機能が実行環境のマシンの OS でもサポートされていることを確認してください。

付録 L.2　障害発生時の対処

障害発生時は，採取した資料を基に原因を調査してください。各資料の説明は，COBOL2002 のマニュアルや Microsoft のサイトなどを参照してください。

付録 L.3　注意事項

実行時メッセージの行番号を基にコンパイルリストから調査する場合は，コンパイルリストと異常終了時要約情報リストのコンパイル日付が一致していることを確認してください。

付録 L.4　障害事例

過去に多く発生した障害事例を次に示します。

(1)　コンパイル時のトラブル

項番	現象	原因	対策
1	ODBC インタフェース機能で，埋め込み SQL に DB 固有の構文（WITH （ROWLOCK, UPDLOCK）指定や SCOPE_IDENTITY()関数など）を記述すると，コンパイル時に KCCC3800C-S エラーとなりました。	埋め込み SQL に DB 固有の構文は使用できません。	ODBC インタフェース機能では，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」に記載している構文だけが使用できます。次のどちらかの方法で対策してください。 • 動的 SQL を使用する。 • HiRDB や Oracle を使用している場合は，ODBC 経由ではなくプリコンパイラを使用する。
2	コンパイル時に，算術演算で KCCC4355C-I エラーとなりました。	一つの算術文に複数の算術式を記述したことで中間結果が 30 けたを超えました。	複数記述された算術式は複数の算術文に分割してください。 このとき，データ項目の整数部のけた数，小数部のけた数は，算術式の値を格納できるけた数で定義してください。

(2)　実行時のトラブル

項番	現象	原因	対策
1	CALL 文による動的なリンクでの呼び出しで KCCC0003R-S エラーとなりました。	次の原因が考えられます。 1. 環境変数 CBLLDLL に，呼び出し先プログラムを含む DLL が指定されていない。 2. 環境変数 CBLLPROGDLL 指定時に，呼び出し先プログラム名と DLL 名が一致していない。	原因ごとの対策を次に示します。 1. 環境変数 CBLLDLL の指定が正しいか確認する。 2. 呼び出し先プログラムのプログラム名と DLL 名が一致しているか確認する。呼び出し先プログラム名にハイフン（-）を含む場合，DLL

項番	現象	原因	対策
		3. 呼び出し先プログラムが，DLL からエクスポートされていないため，指定された DLL に呼び出し先プログラムが存在しない。	名もハイフン（-）のままの名称とする（下線（_）に置き換えない）。 3. 呼び出し先プログラムを含む DLL のエクスポート情報を参照し，呼び出し先プログラムがエクスポートされていることを確認する。呼び出し先プログラムがエクスポートされていない場合，エクスポートするようにプログラムまたはメイクファイルを修正する。DLL でエクスポートされているシンボル名は次の方法で確認できる。 ・COBOL2002 情報抽出ツールに DLL ファイルを指定して実行する。 ・/EXPORTS オプションを指定した dumpbin コマンド※に DLL ファイルを指定して実行する。
2	CALL 文による動的なリンクでの呼び出しで KCCC0131R-S エラーとなりました。	ロード対象 DLL にリンクされた，ほかの DLL がロードできません。	DLL をロードするには，ロード対象 DLL にリンクされたすべての DLL が存在し，かつ，環境変数 PATH にこれらの DLL へのパスが指定されていなければなりません。環境変数 PATH に，すべての必要なパスが指定されているか確認してください。 ロード対象 DLL にリンクされた DLL は次の方法で確認できます。 • COBOL2002 情報抽出ツールにロード対象 DLL を指定して実行する。 • /DEPENDENTS オプションを指定した dumpbin コマンド※にロード対象 DLL を指定して実行する。
3	COBOL プログラムが正常終了したあとの終了コードが 0 以外になりました。	戻り値を持たないプログラムを呼び出しています。	COBOL プログラムが正常終了した場合，RETURN-CODE 特殊レジスタの値がプログラムの終了コードになります。 プログラム実行中に C 言語などで作成された戻り値を持たない（void 型）プログラムを呼び出した場合，RETURN-CODE 特殊レジスタの値は不定となります。この状態でプログラムを終了すると，プログラムの終了コードは不定となります。 この場合，プログラムを終了する直前に，明示的に RETURN-CODE 特殊レジスタの値を設定してください。
4	COBOL プログラムが終わらないため，タスクマネージャから強制終了したが，原因がわかりませんでした。	次の原因が考えられます。 • PERFORM 文などの反復処理の終了条件が真にならない。 • ACCEPT 文や STOP 文で応答待ちになっている。	実行中の処理を特定し，原因を取り除いてください。 実行中の処理は次の手順で特定できます。 1. タスクマネージャで調査対象のプロセスを選択し，右クリックでダンプファイルの作成を選択する。

項番	現象	原因	対策
			2. COBOL2002 情報抽出ツールに，手順 1.で作成したダンプファイル（プロセスダンプ）を指定して実行する。 3. 手順 2.で出力した情報抽出ツール出力結果リストの実行中プログラムの情報から，実行中プログラムの処理を特定する。なお，コンパイル時にデバッグオプションの指定がない場合，実行中プログラムの行番号は特定できない。

注※

dumpbin コマンドは，Visual Studio，および COBOL2002 で提供しています。

付録 M　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（3021-3-600-40）　COBOL2002 Net Developer 04-50，COBOL2002 Net Server Runtime 04-50，COBOL2002 Net Client Runtime 04-50，COBOL2002 Net Server Suite 04-50，COBOL2002 Net Client Suite 04-50，COBOL2002 Net Developer(64) 04-50，COBOL2002 Net Server Runtime(64) 04-50，COBOL2002 Net Server Suite(64) 04-50，COBOL2002 Developer Professional 04-50，COBOL2002 Developer Professional(64) 04-50

追加・変更内容
次の製品の適用 OS に「Windows 11」を追加した。 • P-2636-2344 COBOL2002 Net Developer • P-2636-3344 COBOL2002 Net Client Runtime • P-2636-4344 COBOL2002 Net Client Suite • P-2936-2344 COBOL2002 Net Developer(64) • P-2636-7344 COBOL2002 Developer Professional • P-2936-7344 COBOL2002 Developer Professional(64)
次の製品の適用 OS に「Windows Server 2022」を追加した。 • P-2636-2344 COBOL2002 Net Developer • P-2436-5344 COBOL2002 Net Server Runtime • P-2436-6344 COBOL2002 Net Server Suite • P-2936-2344 COBOL2002 Net Developer(64) • P-2936-5344 COBOL2002 Net Server Runtime(64) • P-2936-6344 COBOL2002 Net Server Suite(64) • P-2636-7344 COBOL2002 Developer Professional • P-2936-7344 COBOL2002 Developer Professional(64)
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。
開発マネージャで-VOSCBL オプションの DataComm サブオプションを指定できるように変更した。

変更内容（3021-3-600-30）　COBOL2002 Net Developer 04-30，COBOL2002 Net Server Runtime 04-30，COBOL2002 Net Client Runtime 04-30，COBOL2002 Net Server Suite 04-30，COBOL2002 Net Client Suite 04-30，COBOL2002 Net Developer(64) 04-30，COBOL2002 Net Server Runtime(64) 04-30，COBOL2002 Net Server Suite(64) 04-30，COBOL2002 Developer Professional 04-30，COBOL2002 Developer Professional(64) 04-30

追加・変更内容
実行環境に，障害調査支援のコンポーネントとして COBOL2002 情報抽出ツールを追加した。 また，COBOL2002 で使用するファイル（拡張子が.dll，.exe，.lib および.obj のファイル）の入力先の説明に，COBOL2002 情報抽出ツールを追加した。
トラブルシュートに有効な資料の説明を追加した。

COBOL2002 Net Developer 04-21，COBOL2002 Net Server Runtime 04-21，COBOL2002 Net Client Runtime 04-21，COBOL2002 Net Server Suite 04-21，COBOL2002 Net Client Suite 04-21，COBOL2002 Net Developer(64) 04-21，COBOL2002 Net Server Runtime(64) 04-21，COBOL2002 Net Server Suite(64) 04-21，COBOL2002 Developer Professional 04-21，COBOL2002 Developer Professional(64) 04-21

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。
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（3021-3-600-20） COBOL2002 Net Developer 04-20，COBOL2002 Net Server Runtime 04-20，COBOL2002 Net Client Runtime 04-20，COBOL2002 Net Server Suite 04-20，COBOL2002 Net Client Suite 04-20，COBOL2002 Net Developer(64) 04-20，COBOL2002 Net Server Runtime(64) 04-20，COBOL2002 Net Server Suite(64) 04-20，COBOL2002 Developer Professional 04-20，COBOL2002 Developer Professional(64) 04-20

追加・変更内容
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（3021-3-600-10） COBOL2002 Net Developer 04-10，COBOL2002 Net Server Runtime 04-10，COBOL2002 Net Client Runtime 04-10，COBOL2002 Net Server Suite 04-10，COBOL2002 Net Client Suite 04-10，COBOL2002 Net Developer(64) 04-10，COBOL2002 Net Server Runtime(64) 04-10，COBOL2002 Net Server Suite(64) 04-10，COBOL2002 Developer Professional 04-10，COBOL2002 Developer Professional(64) 04-10

追加・変更内容
次の製品の適用 OS に「Windows Server 2019」を追加した。 • P-2636-2344 COBOL2002 Net Developer • P-2436-5344 COBOL2002 Net Server Runtime • P-2436-6344 COBOL2002 Net Server Suite • P-2936-2344 COBOL2002 Net Developer(64) • P-2936-5344 COBOL2002 Net Server Runtime(64) • P-2936-6344 COBOL2002 Net Server Suite(64) • P-2636-7344 COBOL2002 Developer Professional • P-2936-7344 COBOL2002 Developer Professional(64)
-VOSCBL オプションに DataComm サブオプションを追加した。これに伴い，DataComm サブオプションを開発マネージャで使用する場合の注意事項を追加した。

変更内容（3021-3-600） COBOL2002 Net Developer 04-00，COBOL2002 Net Server Runtime 04-00，COBOL2002 Net Client Runtime 04-00，COBOL2002 Net Server Suite 04-00，COBOL2002 Net Client Suite 04-00，COBOL2002 Net Developer(64) 04-00，COBOL2002 Net

Server Runtime(64) 04-00，COBOL2002 Net Server Suite(64) 04-00，COBOL2002 Developer Professional 04-00，COBOL2002 Developer Professional(64)

追加・変更内容
環境変数の値の長さを，1,024 バイト以内から 32,766 バイト以内に変更した。
次のリンカオプションの説明を追加した。 • リンク時に実行可能ファイルまたは DLL ファイルにマニフェストファイルを埋め込むためのリンカオプション（-MANIFEST:EMBED,ID=リソース ID） • イメージの作成時に使用されていないライブラリファイルに関する情報を表示するためのリンカオプション（-VERBOSE:UNUSEDLIBS）

付録 N　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 N.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　操作ガイド （3021-3-601）
- 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　Java プログラム呼び出し機能ガイド （3021-3-607）
- COBOL2002　XML 連携機能ガイド （3021-3-608）
- COBOL2002　メッセージ （3021-3-609）
- COBOL2002 Professional 製品 導入ガイド （3021-3-615）
- TP1/COBOL adapter for Cosminexus　ユーザーズガイド （3020-3-B08）
- 索引順編成ファイル管理 ISAM （3020-3-D88）
- ソートマージ （3020-3-N73）
- 日立コード変換ユーザーズガイド （3020-7-351）※1
- 日立コード変換ユーザーズガイド （3020-7-355）※1
- Hitachi Code Converter（Windows(R)編）（3020-7-359）※1
- Hitachi Code Converter for Java （3020-7-360）※1
- 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 コマンドリファレンス（Windows(R)用）（3020-6-455）※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 コマンドリファレンス（Windows(R)用）（3020-6-559）※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 N.2　このマニュアルでの表記

このマニュアルでは，マイクロソフト製品の名称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
Excel			Microsoft Excel
MSMQ			Microsoft Message Queuing Server
MS-DOS			Microsoft MS-DOS
SQL Server			Microsoft SQL Server
Visual Basic			Microsoft Visual Basic
Visual Studio			Microsoft Visual Studio
Windows	Windows 7	Windows 7(x86)	Microsoft Windows 7 Professional 日本語版(32 ビット版)
			Microsoft Windows 7 Enterprise 日本語版(32 ビット版)
			Microsoft Windows 7 Ultimate 日本語版(32 ビット版)
		Windows 7(x64)	Microsoft Windows 7 Professional 日本語版(64 ビット版)
			Microsoft Windows 7 Enterprise 日本語版(64 ビット版)
			Microsoft Windows 7 Ultimate 日本語版(64 ビット版)

マニュアルでの表記			製品名
	Windows 8.1	Windows 8.1(x86)	Windows 8.1 Pro 日本語版(32 ビット版)
			Windows 8.1 Enterprise 日本語版(32 ビット版)
		Windows 8.1(x64)	Windows 8.1 Pro 日本語版(64 ビット版)
			Windows 8.1 Enterprise 日本語版(64 ビット版)
	Windows 10	Windows 10(x86)	Windows 10 Pro 日本語版(32 ビット版)
			Windows 10 Enterprise 日本語版(32 ビット版)
		Windows 10(x64)	Windows 10 Pro 日本語版(64 ビット版)
			Windows 10 Enterprise 日本語版(64 ビット版)
	Windows 11		Windows 11 Pro 日本語版
			Windows 11 Enterprise 日本語版
	Windows Server 2008 R2		Microsoft Windows Server 2008 R2 Standard 日本語版
			Microsoft Windows Server 2008 R2 Enterprise 日本語版
			Microsoft Windows Server 2008 R2 Datacenter 日本語版
	Windows Server 2012		Microsoft Windows Server 2012 Standard 日本語版
			Microsoft Windows Server 2012 Datacenter 日本語版
	Windows Server 2012 R2		Microsoft Windows Server 2012 R2 Standard 日本語版
			Microsoft Windows Server 2012 R2 Datacenter 日本語版
	Windows Server 2016		Microsoft Windows Server 2016 Standard 日本語版
			Microsoft Windows Server 2016 Datacenter 日本語版
	Windows Server 2019		Windows Server 2019 Standard 日本語版
			Windows Server 2019 Datacenter 日本語版
	Windows Server 2022		Windows Server 2022 Standard 日本語版
			Windows Server 2022 Datacenter 日本語版

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
Windows または Windows COBOL2002	Windows(x86)または Windows(x86) COBOL2002	P-2636-2344 P-2436-5344 P-2636-3344

マニュアルでの表記		該当する製品の形名
		P-2436-6344 P-2636-4344 P-2636-7344
	Windows (x64)または Windows (x64) COBOL2002	P-2936-2344 P-2936-5344 P-2936-6344 P-2936-7344

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
Pervasive.SQL	Pervasive.SQL
	Pervasive.SQL 2000
XMAP3	XMAP3 Server
	XMAP3 Developer Version 5
	XMAP3 Server Runtime Version 5
	XMAP3 Client Runtime Version 5
Windows SDK	Windows 10 SDK
	Windows SDK for Windows 11

- 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。
- 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。
- Enterprise JavaBeans を EJB と表記しています。
- 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 E.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
X と表示された場合	Windows(x86) COBOL2002 であることを示します。

識別記号（c）	内容
E と表示された場合	Windows(x64) COBOL2002 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 N.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 O　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。

C0 メジャー情報

全体の実行文の数に対する実行済み文の数の割合を示す情報。

C1 メジャー情報

全体の分岐の数に対する実行済みの分岐の数の割合を示す情報。

COBOL2002 Developer Professional

COBOL2002 Net Developer と COBOL2002 Professional Tool Kit を一つにしたプログラムプロダクトのこと。Windows の稼働環境に応じて，次の 2 製品がある。

- COBOL2002 Developer Professional（32bit 版）
- COBOL2002 Developer Professional(64)（64bit 版）

このマニュアルでは，上記の製品を総称して，COBOL2002 Developer Professional と表記する。

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。

OLE オブジェクト

OLE2 オートメーション機能を使用するときの操作の対象。OLE オブジェクトは，OLE プロパティと OLE メソッドを持つ。

OLE オブジェクト参照データ項目

USAGE 句で用途は OLE オブジェクト参照（OBJECT REFERENCE OLE）と指定されたデータ項目。インスタンス化された OLE オブジェクトを指すポインタを格納する。

OLE プロパティ

OLE オブジェクトの状態または属性。

OLE メソッド

OLE オブジェクトに対する操作。

S1 メジャー情報

全体の呼び出し文（CALL 文，OLE メソッド操作を除く INVOKE 文）の数に対する実行済みの呼び出し文の数の割合を示す情報。

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。

UAC（User Account Control）

管理者ユーザとしてログオンしても，既定ではアプリケーションの実行権限は標準権限に下げられる。管理者権限を必要とするアプリケーションを起動しようとすると，システムは昇格ダイアログボックスを表示して，ユーザの確認を求める。ユーザが実行を許可した場合だけ，アプリケーションは管理者権限で実行されることになる。これによって，ユーザが知らない間に悪意のあるソフトウェアをインストールされたり，システム設定を変更されたりすることを防止できる。

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。

WRP（Windows Resource Protection）

Windows システムの安定性，および信頼性を向上させるための機能。特定の OS ファイル，フォルダ，レジストリ キーなど，Windows の読み取り専用リソースを保護する。

（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。

埋め込み型 SQL

COBOL ソースファイル中に直接埋め込まれた SQL 文のこと。埋め込み型 SQL を使うと，COBOL プログラムでデータベースへアクセスする機能が使用できる。

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。

管理者権限

管理者ユーザおよびビルトイン管理者（Administrator アカウント）に与えられる権限。

管理者ユーザ

Windows の Administrators グループに属するアカウント。ビルトイン管理者（Administrator アカウント）とは区別される。

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，または cblbuild2k 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。

差分カバレージ情報

原始プログラムの修正によって影響を受けた部分に対するカバレージ情報。

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。

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。

昇格ダイアログボックス

Windows の UAC で表示されるダイアログボックス。アプリケーションの実行権限を管理者権限に昇格しようとする場合，ユーザに権限昇格の確認を求めるダイアログボックス。

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。

ストアドプロシージャ

コンパイルしたSQL文を，コマンドとしてサーバに登録したもの。実行時にSQL文を解析する通常の方法に比べ，実行速度が高速で，ネットワークにかかる負荷も小さい。

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。

単体テスト

プログラム（コンパイル）単位のテスト。

強い型付け

STRONG指定のあるTYPEDEF句によってデータ型を宣言すること。

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。

ノーマルファイル形式

2GB 未満のファイルサイズまで操作できる形式の索引順編成ファイルのこと。使用できるファイルシステムは，FAT，FAT32 および NTFS である。

（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。

バリアントデータ項目

USAGE 句で用途はバリアント（VARIANT）と指定されたデータ項目。INVOKE 文や SET 文で使用する VARIANT 値を指すポインタを格納するためのデータ領域。

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。

標準権限

標準ユーザに与えられる権限。故意に，または誤ってシステムに変更を加える許可を持たない。

標準ユーザ

Windows の Users グループに属するアカウント。

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。

ビルド

プロジェクトに登録されたソースファイルや定義ファイルから実行可能ファイルを生成する方法。ビルドの対象となるファイルは，変更のあったファイルだけである。

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。

プロジェクト

実行可能ファイル，ダイナミックリンクライブラリ（DLL），または標準ライブラリを作成するための資源を一括して管理するための概念。一つのプロジェクトは，一つの実行可能ファイル，DLL，または標準ライブラリに対応している。

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。

弱い型付け

STRONG 指定のない TYPEDEF 句によってデータ型を宣言すること。

（ラ行）

ラージファイル形式

2GB を超えて操作できる形式の索引順編成ファイルのこと。使用できるファイルシステムは NTFS である。

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

_beginthreadex 620
_endthreadex 620
.bkf〔バックアップファイル〕 1163
.cbc〔COPY 関連づけファイル〕 1163
.cbd〔EXPORTS 名称ファイル〕 1163
.cbe〔2 項関係情報ファイル〕 1163
.cbf 798, 1167
.cbl 792, 798
.cbl〔COBOL UAP 引数定義ファイル〕 1163
.cbl〔COBOL ソースファイル〕 1163
.cbl〔XML アクセス用ステータス定義〕 1163
.cbl〔XML アクセス用データ定義〕 1163
.cbl〔XML アクセスルーチン〕 1163
.cbo〔オブジェクト関連づけファイル〕 1163
.cbp 846
.cbp〔プログラム情報ファイル〕 1163
.cbr〔実行環境ファイル〕 1164
.cbs 852
.cbs〔擬似プログラム用プログラム情報ファイル〕 1164
.cbw 853, 986
.cbw〔stdcall 呼び出し指示ファイル〕 1164
.cet〔プログラムテンプレートファイル〕 1164
.cex〔COBOL エディタ設定ファイル〕 1164
.class〔Cosminexus 上 Java 実行ファイル〕 1164
.cll〔カバレージ情報リストファイル〕 1164
.cnl〔カウント情報リストファイル〕 1164
.cno〔実行結果出力ファイル（カウント情報の表示）〕 1164
.cob 792, 798, 1167
.csv〔ソース解析情報ファイル〕 1165
.cvo〔実行結果出力ファイル（カバレージ情報の蓄積）〕 1164
.cxc〔カタログファイル〕 1164
.cxd〔XML データ定義ファイル〕 1164
.def 790, 855, 1075
.def〔モジュール定義ファイル〕 1164
.dll 790
.dll〔DLL ファイル〕 1165
.drf〔データレコードファイル〕 1165
.drl〔データレコードファイル〕 1165
.dtd〔XML 文書型定義ファイル〕 1165
.exe 790
.exe〔実行可能ファイル〕 1165
.exp〔エクスポートファイル〕 1165
.flf〔ファイル定義ファイル〕 1165
.hmf 810
.hmf〔プロジェクトマスタファイル〕 1165
.hmp〔プロジェクト情報ファイル〕 1165
.html〔ソース解析情報ファイル〕 1165
.java〔COBOL アクセス用 Bean（Java ソースファイル）〕 1165
.java〔EJB 関連 Java ソースファイル〕 1165
.k01〔主キーファイル〕 1165
.k02～.k99〔副キーファイル〕 1165
.kdf〔キー定義ファイル〕 1166
.kdl〔キー定義ファイル〕 1166
.l01〔主キーファイル〕 1165
.l02～.l99〔副キーファイル〕 1165
.lib 790
.lib〔ライブラリファイル〕 1166
.lst 1134
.lst〔コンパイルリストファイル〕 1166
.obj 790
.obj〔オブジェクトファイル〕 1166
.ocb 792, 798, 1167
.ocf 792, 798, 1167
.pkg〔パッケージ情報ファイル〕 1166
.prt〔印刷書式情報ファイル〕 1166
.rc 790, 856, 1071
.rc〔リソース定義ファイル〕 1166
.rdf〔レコード定義ファイル〕 1166
.rep 945

.rep〔リポジトリファイル〕 1166
.res 790, 1071
.res〔リソースファイル〕 1166
.sai〔ソース解析情報ファイル〕 1166
.svw〔COBOL サービスルーチンファイル〕 1166
.tag〔タグファイル〕 1166
.tdi〔TD コマンド格納ファイル（中断点情報）〕 1167
.tds〔TD コマンド格納ファイル（中断点情報）ファイル（シミュレーション情報）〕 1167
.usw〔ユーザキーワードファイル〕 1167
.wdf〔画面定義ファイル〕 1167
.xml〔XML 文書型定義ファイル〕 1165
.xml〔デプロイ情報（DD ファイル）〕 1167
%変数名% 994
-? 809, 906, 1062, 1068, 1073
-ALLOWBIND 1056
-BigEndian 885
-BigEndian,Bin 708
-Bin1Byte 894
-BinExtend 897
-Cblctr 871
-CBLVALUE 901
-CheckUninitData オプション 911
-CmAster 873
-CmDol 873
-Comp5 874
-Compati85 868
-CompatiM7 865
-CompatiV3 292, 865
-Compile 854
-CVInf 846, 1030
-d 1072
-DebugCompati 844, 1030
-DebugData 845, 1030
-DebugInf 843, 1030
-DebugInf,Trace 1030
-DebugLine 843, 1030
-DebugRange 847, 1030
-DEF 1056, 1067
-DEFAULTLIB 1057
-DefFile 855
-Define 903
-Define オプション 802
-Details 904
-DigitsTrunc 871
-Dll 833
-DLL 1057
-DllInit 840
-DoubleQuote 885
-DynamicLink 859
-ENTRY 1057
-EquivRule 896
-ErrSup 893
-EXPORT 1058, 1067
-EXTRACT 1067
-fo 1072
-FORCE 1058
-FunctionECSup,CodeConvErr オプション 911
-h 1073
-H8Switch 870
-HEAP 1058
-Help 809, 906
-i 1072
-IconFile 856
-IgnoreAPPLY,FILESHARE 891
-IgnoreLCC 872
-IMPLIB 1058
-INCLUDE 1059, 1067
-IntResult,DecFloat40 908
-IsamExtend 836
-JPN 895
-l 1073
-LARGEADDRESSAWARE 1059
-Lib 854
-Lib,CUI 373
-Lib,GUI 373, 614
-LIBPATH 1059, 1067
-Link 858

-LIST 1067
-LiteralExtend 909
-LowerAsUpper 901
-Main 831
-Main,System 395
-Main,V3 395
-MainNotCBL 396, 610, 710, 839
-Main オプションが指定されたプログラム 1215
-MANIFEST 1059
-ManifestFileExt 859
-MANIFESTUAC 1059
-MAP 1060
-MAPINFO 1060
-MaxDigits38 907
-MinusZero 898
-MultiThread 606, 839
-n 1073
-NODEFAULTLIB 1061, 1068
-NumAccept 256, 268, 838
-NumCsv 839
-OldForm 905
-OpenTP1 838
-Optimize 841
-Optimize,0 774
-Optimize,1 774
-Optimize,2 774
-Optimize,3 774
-ORDER 1061
-OUT 1061, 1068
-OutputFile 857
-PortabilityCheck 890
-RDBTran 836
-RDBTran オプションの指定と COMMIT／ROLLBACK 文の扱い 185
-REMOVE 1068
-Repository 902
-Repository,Gen 953, 970
-Repository,Sup 970
-RepositoryCheck 903, 970
-ResrcFile 856
-ScreenSpeed 842
-SimIdent 851
-SimMain 849
-SimSub 850
-SpaceAsZero 910
-SQL 835
-SQLDisp 835
-SrcList 892
-SrcList,CopyAll 1133
-SrcList,CopySup 1133
-SrcList,NoCopy 1133
-SrcList,OutputAll 1133
-SrcList,xxxxx,DataLoc 1134
-SrcList,xxxxx,NoFalsePath 1133
-STACK 1061
-Std2002 862
-Std85 861
-StdCall 853
-StdCallFile 853
-StdMIA 860
-StdVersion 863
-SUBSYSTEM 1061
-Switch 877
-TDInf 846, 1030
-TestCmd 848
-TruncCheck 899
-u 1073
-UniEndian 907
-UniEndian オプション指定時の注意事項 656
-UniEndian オプションと実行時環境変数 CBLUNIENDIAN の関係 657
-UniObjGen 906
-UniObjGen オプション指定時の注意事項 655
-UniObjGen オプションと実行時環境変数 CBLLANG の関係 657
-UniObjGen オプションと-UniEndian オプションの関係 655
-UscoreStart 897

-v 1073
-V3ConvtName 876
-V3ConvtName オプションを指定したときの原文名定数の扱い 798
-V3Rec 881
-V3RecEased 884
-V3RecFCSpace 882
-V3Spec 875
-VERBOSE 1062, 1068
-VERSION 1062
-VOSCBL 888
-x 1073
-XMAP 837
-XMAP,LinePrint 236

数字

10 進項目の 2 進項目化 782
16 進英数字定数 767
1 バイトの 2 進項目 894
1 文字入力する 749
1 文字を出力する 749
2020 のキーボードと 106 日本語キーボードとの対応表 754
2 項関係情報ファイル 1163
40 けた 10 進浮動小数点形式 691
64bit アプリケーションの作成 1079
9x の入出力状態 768

A

ADD 文 88
AND 81
ANSI（アメリカ規格） 40
API の合致レベル 566
APPLY FORMS-NAME 句 214
APPLY FORMS-OVERLAY 句 238, 837
argc 713
argv 713
AS 481
ASCII 877, 878
ASSIGN 句 130

B

BASE クラス 1215
BEGIN DECLARE SECTION 555
BOM 1215
BTMODIFY 1005
Btrieve（Pervasive.SQL）による索引編成ファイル 188, 206, 837, 1005
BY CONTENT 380
BY REFERENCE 380
BY VALUE 380

C

C0 メジャー情報 1215
C1 メジャー情報 1215
CALL 556
CALL 定数 897, 901
CALL 文 389, 390
CALL 文で直接システム関数を呼び出し 769
CALL-CONVENTION 段落 404
CANCEL 文 391
CANCEL 文実行後のプログラムの状態 393
CBL_BATCH 1002
CBL_BTRPGSZ 192, 1010
CBL_FLSRVDUMP 333, 1026
CBL_RDBCOMMIT 176, 1011
CBL_RDBSYS 914
CBL_RECLOCKMAX 203, 1011
CBL_STOPNOADV 262, 1003
CBL_SYSCSVIN 1011, 1112
CBL_SYSCSVOUT 1011, 1113
CBL_SYSERR 1002
CBL_SYSIN 252, 1003
CBL_SYSOUT 1003
CBL_SYSPUNCH 1003
CBL_SYSSTD 252, 1004
CBL_UNINITDATA_BREAKOFF 914
CBL_外部装置名 1011

CBL2KENV_CBLLIB 1194
CBL2KENV_LIB 1194
CBL2KENV_PATH 1194
cbl2krep 960
CBLABN 712
CBLABNCODE 1000
CBLABNLST 1025, 1034
CBLACTWIN 1011
CBLADTRM 750
CBLARGC 713
CBLARGV 713
CBLATTRIBUTE 1012
CBLAUTOCLEAR 1013
CBLBELL 756
cblbuild2k コマンド 807, 810
CBLCLOSE 313
CBLCNSL 755
CBLCNVERRORINFO 735
CBLCOMCBR 1000
CBLCOPT 914
CBLCOPT2002 807, 915
CBLCREATEOBJ 1116
CBLCSVCHAR 165, 1004
CBLCSVINIT 167, 1004
CBLCUR 757
CBLD_ファイル名 1004
CBLDATADUMP 728
CBLDATADUMPFILE 1025, 1039
CBLDATE 258, 1003
CBLDAY 259, 1003
CBLDBGINF 611, 727
CBLDDUMP 1025, 1039
CBLDELETE 313, 314
CBLDLTRM 754
CBLEND 710
CBLEND サービスルーチンによる終了 376
CBLENTERCHK 1018
CBLERRMAX 915
CBLEUDCFUNC 216, 1008
CBLEXCEPT 1025
CBLEXEC 715
CBLEXVALUE 75, 1000
CBLF_ファイル名 215, 1008
CBLFEP 1018
CBLFIX 792, 797, 915
CBLFIXEDFORMLINE 916
CBLFREE 792, 797, 916
CBLFSYNC 339, 345, 1008
CBLGCINTERVAL 467, 1027
CBLGCSTART 467, 1027
CBLGDIINTERVAL 1008
CBLGDIWMSG 1008
CBLGET 749
CBLGETOBJ 1117
CBLGETPROPERTY 1119
CBLGINT 373, 709
CBLHANDLE 717
CBLIMEPOS 1018
CBLINITVALUE 917
CBLINPUTDLG 720
CBLIOMESSAGE 1008
CBLISAMDL 1009
CBLISAMLARGE 1009
CBLJCPOPENDKEY 1018
CBLLANG 1001
CBLLDLL 1001
CBLLIB 798, 917, 1193, 1198
CBLLINKER 918
CBLLINKINTERVAL 919
CBLLPROGDLL 1001
CBLM7ENDKEY 1019
CBLMANIFESTTOOL 919
CBLMESSAGE 719
CBLMETHODCALL 1122
CBLMQCLOSE 629
CBLMQCREATE 626
CBLMQDELETE 627
CBLMQLOCATE 634

CBLMQOPEN 628
CBLMQRECEIVEMSG 632
CBLMQSENDMSG 630
CBLNCNV 731
CBLNCNV サービスルーチン〔Unicode 機能〕 676
CBLNOCLOSE 1019
CBLOLE2INIT 1114
CBLOLE2UNINIT 1130
CBLONLYNUM 1019
CBLOPEN 313
CBLOPS 1022
CBLOVERFLOW 1019
CBLP_ファイル名 229, 1009
CBLPGMSEARCHTRC 1001
CBLPGMSEARCHTRC_SIZE 1001
CBLPIDIR 920
CBLPRELOAD 1001
CBLPRMCHKW 1025
CBLPRNT_xxx 294, 1021
CBLPRNTID 294, 1021
CBLPRTEXCHR 231, 1009
CBLPUT 749
CBLPUTPROPERTY 1120
CBLRDBDATAERR 183, 1009
CBLRDBILWAIT 1009
CBLRDBOPURGE 1010
CBLRDBROWVALCONSTRUCTOR 1010
CBLREAD 313
CBLREP 920, 949
CBLRESOURCECOMPILER 921
CBLREWRITE 313, 314
CBLSETCBLITEM 1126
CBLSETFIELD 1019
CBLSETTITLE 739
CBLSETVARIANT 1128
CBLSGET 738
CBLSORTSIZE 283, 1022
CBLSORTWORK 282, 1022
CBLSQLCOMMOD 1023
CBLSQLCURUSE 1023
CBLSQLDYNAMIC 1024
CBLSQLERROR 741
CBLSQLLOGINTIMEOUT 1024
CBLSQLQUERYTIMEOUT 1024
CBLSQLROWCOUNT 1024
CBLSQLSETOPT 744
CBLSQLSUPPRESSMSG 1024
CBLSQLWMSG 1024
CBLSTART 313, 314
CBLSTMCLOSE 352
CBLSTMCREATE 352
CBLSTMOPEN 353
CBLSTMREAD 354
CBLSTMWRITE 355
CBLSYSLOG 1026
CBLSYSLOGLVL 1026
CBLSYSLOGSRV 1027
CBLSYSREP 921
CBLTAB 792, 922
CBLTDEXEC 1026
CBLTERM_xxx 291, 1022
CBLTERMID 291, 1022
CBLTERMSHAR 293, 1022
CBLTEXTSUPPRESSBOM 1010
CBLTEXTWRITESPACE 1010
CBLUBIT 734
CBLUNDERDOT 1020
CBLUNIENDIAN 1001
CBLUNINITDATA_OUTRESULTLIST 922
CBLUNINITDATA_OUTRESULTLISTDIR 922
CBLUNLOCK 313, 314
CBLUPDOWNMOVE 1020
CBLUPSI 366, 1002
CBLV3UNICODE 924
CBLVALUE 902, 923
CBLVARIANTINF 1124
CBLWDISK 313, 314
CBLWRITE 313, 314

CBLX_外部装置名　236, 1010
CBLXMAPERROR　746
CBL-CTR 特殊レジスタ　871
ccbl2002 コマンド　807
ccbl2002 コマンドに指定するオプション列　915
ccbl2002 コマンドのヘルプ　906
ccbl コマンド　807, 809
ccbl コマンドに指定するオプション列　914
cdecl 呼び出し規約　404
CHARACTER TYPE 句　220, 837
cl　450
CLOSE　556
COBOL2002 Developer Professional　42, 1215
COBOL2002 Net Client Runtime　42
COBOL2002 Net Client Suite　43
COBOL2002 Net Developer　42
COBOL2002 Net Server Runtime　42
COBOL2002 Net Server Suite　42
COBOL2002 V4 への移行性と互換性　1092
COBOL2002 仕様　863
COBOL2002 情報抽出ツール　45
COBOL2002 で追加された機能　40
COBOL2002 でのオブジェクト指向機能　465
COBOL2002 の概要　40
COBOL2002 の機能　40
COBOL2002 の構成　44
COBOL85 および旧バージョンの COBOL2002 からの移行性と互換性　1092
COBOL85 形式のコンパイラオプション　809
COBOL85 と COBOL2002 のコンパイラオプションの対応　1184
COBOL85 版 Cosminexus 連携の移行性と互換性　1098
COBOL85 版 XML 連携の移行性と互換性　1099
COBOL UAP 引数定義ファイル　1163
COBOL アクセス用 Bean　1165
COBOL エディタ　45
COBOL エディタ設定ファイル　1164
COBOL エラー詳細情報　324
COBOL から C を呼ぶときの規則　428
COBOL 言語で使用できる Java 言語のデータ型　451
COBOL 原始プログラムのコンパイル　797
COBOL 原始プログラムの正書法　794
COBOL コマンドプロンプト　1197
COBOL コマンドプロンプトの環境変数の状態　1197
COBOL コマンドプロンプトの起動と終了　1197
COBOL サービスルーチンファイル　1166
COBOL 実行環境の終了方法　397
COBOL 実行単位の終了　375
COBOL 主プログラム　395
COBOL 主プログラムと副プログラム　395
COBOL ソースの作成　789
COBOL ソースファイル　790, 1163
COBOL で使用するファイル　1163
COBOL 入出力サービスルーチン　310
COBOL 入出力サービスルーチンが対応している機能　311
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　331
COBOL 入出力サービスルーチンでのディスク書き込み保証　338
COBOL 入出力サービスルーチンでの複数レコード施錠　337
COBOL 入出力サービスルーチンのインタフェース　315
COBOL 入出力サービスルーチンの使用例　340
COBOL 入出力サービスルーチンの制限事項　312
COBOL 入出力サービスルーチンを使用するときの注意事項　342
COBOL の概要　40
COBOL の実行環境を終了させる　710
COBOL の実行単位　358
COBOL のデータ項目と C プログラムの型の対応　423
COBOL の特長　40
COBOL 副プログラム　395
COBOL プログラムが使用するスタック領域　770
COBOL プログラムから C プログラムを呼び出す方法　428
COBOL プログラム間で値を返す場合の規則　384

COBOL プログラムのデータ型と Visual Basic での宣言 438
COBOL プログラムのモード 369
COLUMN 句 299
COMMIT 555, 836
COMMON 404
COMPUTE 文 88
COMP-5 874
CONNECT 555
CONTINUE 文 116
COPY 関連づけファイル 1163
COPY 登録集 797
COPY 文による登録集原文の取り込み 797
COPY 文の REPLACING 指定および REPLACE 文の作用対象 769
COPY 文の指定形式 797
CORRESPONDING 指定 85
Cosminexus 上 Java 実行ファイル 1164
Cosminexus 連携機能 41, 441
CREATEOBJ メソッド 595
CRT STATUS 句 300
CSVQUOTE 1006
CSVTABSEPARATED 169, 1006
CSVWRITESPACE 168, 1006
CSV ファイルへのアクセス 1112
CSV 編成ファイル 160, 839, 1215
CSV 編成ファイルに数値データとして有効な文字の種類 164
CSV 編成ファイルのファイル編成とレコード形式 160
CUI モード 369
CUI モードのユーザプログラム 854
C から COBOL を呼ぶときの規則 422
C 言語との連携 421
C ソースファイルのコンパイル 450
C プログラムから COBOL プログラムを呼び出す方法 422
C プログラムでの 64bit アプリケーションの作成 421
C プログラムの型と COBOL での宣言 429

D

DBMS 565
DC シミュレーション 552
DD ファイル 1167
DEALLOCATE PREPARE 556
DECLARE CURSOR 555
DEFINED 条件 802
DEFINE 指令 50
DELETE 555
DISCONNECT 555
DISPLAY 文によるデータの出力 259
DIVIDE 文 90
DLL 359, 1062
DLL 検索フォルダ 408
DLL 自動ロード機能 410
DLL に含まれるプログラムの呼び出し 408
DLL に含まれるプログラムの呼び出し方法 408
DLL の概要 408
DLL の作成 983
Dll の属性を cdecl にする 833
Dll の属性を fastcall にする 834
Dll の属性を stdcall にする 833
DLL ファイル 1165
DML 情報エリアの詳細 586

E

EBCDIC 877, 878
EBCDIK 877, 878
EC-ALL 492, 493
EC-ARGUMENT 493
EC-ARGUMENT-FUNCTION 493
EC-ARGUMENT-IMP 493
EC-BOUND 493
EC-BOUND-ODO 493
EC-BOUND-REF-MOD 493
EC-BOUND-SUBSCRIPT 493
EC-DATA 493
EC-DATA-PTR-NULL 493
EC-FLOW 493

EC-FLOW-GLOBAL-EXIT 494
EC-FLOW-GLOBAL-GOBACK 494
EC-FLOW-IMP 494
EC-FLOW-RELEASE 494
EC-FLOW-RETURN 494
EC-FLOW-USE 494
EC-I-O 494
EC-I-O-AT-END 494
EC-I-O-EOP 494
EC-I-O-EOP-OVERFLOW 494
EC-I-O-IMP 494
EC-I-O-INVALID-KEY 494
EC-I-O-LINAGE 494
EC-I-O-LOGIC-ERROR 494
EC-I-O-PERMANENT-ERROR 495
EC-OO 495
EC-OO-CONFORMANCE 495
EC-OO-EXCEPTION 495
EC-OO-IMP 495
EC-OO-METHOD 495
EC-OO-NULL 495
EC-OO-RESOURCE 495
EC-OO-UNIVERSAL 495
EC-OVERFLOW 495
EC-OVERFLOW-STRING 495
EC-OVERFLOW-UNSTRING 495
EC-PROGRAM 495
EC-PROGRAM-ARG-MISMATCH 495
EC-PROGRAM-CANCEL-ACTIVE 495
EC-PROGRAM-IMP 495
EC-PROGRAM-NOT-FOUND 496
EC-PROGRAM-RECURSIVE-CALL 496
EC-PROGRAM-RESOURCES 496
EC-RAISING 496
EC-RAISING-NOT-SPECIFIED 496
EC-RANGE 496
EC-RANGE-INSPECT-SIZE 496
EC-RANGE-INVALID 496
EC-RANGE-PERFORM-VARYING 496
EC-RANGE-SEARCH-INDEX 496
EC-RANGE-SEARCH-NO-MATCH 496
EC-SIZE 496
EC-SIZE-EXPONENTIATION 496
EC-SIZE-OVERFLOW 496
EC-SIZE-TRUNCATION 496
EC-SIZE-UNDERFLOW 497
EC-SIZE-ZERO-DIVIDE 497
EC-SORT-MERGE 497
EC-SORT-MERGE-IMP 497
EC-SORT-MERGE-RELEASE 497
EC-SORT-MERGE-RETURN 497
EC-USER 497
EC-USER- 498
EC-USER-（ユーザ定義例外名） 497
EJB 関連 Java ソースファイル 1165
END DECLARE SECTION 555
ESC/P モード印刷 226
ESC/P モード印刷を利用したプリンタへの出力例 232
EVALUATE 指令 51
EVALUATE 指令の例 802
EVALUATE 文 102
EXCEPTION-FILE 関数 532
EXCEPTION-LOCATION 関数 532
EXCEPTION-OBJECT 537
EXCEPTION-STATEMENT 関数 534
EXCEPTION-STATUS 関数 535
EXECUTE 556
EXECUTE IMMEDIATE 556
EXIT 文 518
EXIT 文，GOBACK 文の RAISING 指定による例外の伝播 522
EXPORTS 文 1077
EXPORTS 名称ファイル 1163
EXPORT ファイル 1165
EXTERNAL 句 74, 436
EXTERNAL 句を用いたデータの共用 619
EXTERNAL 属性チェック 76

F

fastcall 呼び出し規約 405
FETCH 556
FILE STATUS 句 548
FLAG-85 指令 50
FSYNC 345, 1006

G

GDI モード印刷 218
GetCurrentThreadId 613
GETOBJ メソッド 596
GET PROPERTY 470
GET プロパティメソッド 470
GLOBAL 句 73
GOBACK 文 518
GOBACK 文による終了 376
GO TO 文 108
GUI モード 370, 709
GUI モードのユーザプログラム 854

H

HEAPSIZE 文 1077
HiRDB による索引編成ファイル 170
HTML 1216

I

ICON 文 1074
IF 指令 51
IF 指令の使用例 802
IF 文 103
IMPLEMENTS 句 476
INHERITS 句 473
INITIALIZE 文 118
INITIAL 句 398
INSERT 555
INSPECT 文 99
INTERFACE-ID 476
ISAM 45, 147
ISAMDL 1005
ISAMLARGE 1007
ISAMPREV 1005
ISAM による索引編成ファイルでのラージファイル入出力機能 195
ISAM ユティリティ 147
ISO（国際規格） 40
IVS 1216

J

java 451
javac 447
javah 447
Java アプリケーションサーバ 441
Java インタプリタ 451
Java クラス 444
Java 言語 441
Java 言語と COBOL 言語のデータ型の対応 451
Java ソースファイル 1165
Java ソースファイルのコンパイル 446
Java ソースプログラムの作成 444
Java との連携 441
Java の文字列型を COBOL で使用する方法 452
Java バイトコードファイル 446
Java プログラム 441
Java プログラムから COBOL プログラムを呼び出す方法 444
JCPOPUP 722
JIS（日本工業規格） 40
JIS 仕様 862
JXBSAID 581

L

LIB 1065, 1193, 1198
LIBRARY 文 1076
LINE 句 299
LISTING 指令 51, 1134

M

MIA 仕様 860
MIA バージョン 1.3 仕様を範囲外チェック 861

MIA バージョン 1.4 仕様の範囲外チェック 861
MOVE 文 121
MSMQ アクセス機能 621
MSMQ アクセス機能（Unicode 機能） 676
MSMQ アクセスサービスルーチン 626
MSMQ アクセスサービスルーチンの一覧 626
MSMQ アクセスサービスルーチンのインタフェース 637
MSMQ アクセスサービスルーチンの実行順序 643
MSMQ アクセスサービスルーチンの使用例 644
MSMQ の概要 622
MULTIPLY 文 90

N

NAME 文 1076
NEW メソッド 466
NOBTMODIFY 1005
NOCSVQUOTE 166, 1006
NOCSVTABSEPARATED 1006
NOCSVWRITESPACE 1006
NOFSYNC 345, 1006
NOISAMDL 1005
NOISAMLARGE 1007
NOISAMPREV 1005
NOSAMAADV 1005
NOSAMBADV 1005
NOSAMPADV 1006
NOTEXTSUPPRESSBOM 1007
NOTEXTWRITESPACE 1007

O

ODBC SQL データ型に対応した COBOL のデータ記述 571
ODBC インタフェース 1023, 1024
ODBC インタフェース機能 565, 741, 744, 835
ODBC インタフェース機能が動作する環境 565
ODBC ドライバ 565
ODBC の合致レベル 565
ODBC メッセージ 569
ODBC レコード定義生成 45
OLE2 オートメーション機能 593
OLE2 オートメーション機能（Unicode 機能） 677
OLE2 オートメーションクライアント機能 594, 595, 1113
OLE2 オートメーションサーバ機能 594
OLE アプリケーションの終了 598
OLE オブジェクト 1216
OLE オブジェクト参照データ項目 1216
OLE オブジェクトの解放 598
OLE オブジェクトの生成と取得 595
OLE プロパティ 1216
OLE メソッド 1216
OLE メソッドが返す OLE オブジェクトを利用した参照 597
OLE メソッドと OLE プロパティの操作 596
OOCOBOL からの移行性 1098
OPEN 556
OpenTP1 179, 551, 838
OPEN 文〔CSV 編成ファイル〕 161
OR 82

P

PAGE 指令 51, 1137
PATH 1193, 1197
PCA 1110
PDCANCEL 181
PDSWAITTIME 181
PERFORM 文 108
PERFORM 文の実行範囲 115
PREPARE 556
PRINTER 218
PROPAGATE 指令 52, 521
PROPERTY 句 472

R

RAISE 文 530
RAISING LAST 指定 523
RAISING 指定 522

RAISING 指定による例外オブジェクトの伝播 525
RDB の列のデータ型と COBOL のデータ記述 174
READ INTO 765
READ 文〔CSV 編成ファイル〕 162
READ 文〔テキスト編成ファイル〕 154
RECURSIVE 句 402
RELEASE 文 287
RESUME 文 517
RETURNING 385
RETURN 文 287
RETURN-CODE 特殊レジスタ 377, 386
REWRITE 文〔テキスト編成ファイル〕 155
ROLLBACK 555, 836

S

S1 メジャー情報 1216
SAMAADV 1005
SAMBADV 1005
SAME AS 句 55
SAMPADV 1006
SEARCH 文 105
SELECT 555
SELECT 句 130
SELF 1216
SET CONNECTION 555
SET LAST EXCEPTION TO OFF 499
SET PROPERTY 470
SET プロパティメソッド 470
SET 文 122, 367, 538
SORT 45
SORT-CORE-SIZE 285
SORT-FILE-SIZE 285
SORT-MESSAGE 285
SORT-MODE-SIZE 285
SORT-RETURN 285
SOURCE FORMAT 指令 50
SOURCE FORMAT 指令による正書法の切り替え 795
SQL 565
SQLCA 591
SQLCODE 変数 568, 569
SQLCODE 変数の値と意味 569
SQLSTATE 568
SQL エラーコード 569
SQL 宣言節 556
SQL の合致レベル 566
SQL 文 555, 556
SQL 文でエラーが発生した場合の対処方法 568
SQL 文のエラー処理 568
SQL 文の実行順序 557
SQL 連絡領域 SQLCA 591
STACKSIZE 文 1077
stdcall 986
stdcall 呼び出し規約 405, 853
stdcall 呼び出し指示ファイル 853, 986, 1164
stdcall 呼び出し指示ファイルの作成規則 987
stderr 131, 133, 135
stdin 131, 133, 135
stdout 131, 133, 135
STOP RUN 文による終了 375
STOP 文 261
STRING 文 95
STRONG 指定 54
SUBTRACT 文 89
SUPER 1216
SYMBOLIC TERMINAL 句の指定 292, 295
SYSPRT 226
System 指定メインプログラム 832

T

TD コマンド 1217
TD コマンド格納ファイル 848
TD コマンド格納ファイル（シミュレーション情報） 1167
TD コマンド格納ファイル（中断点情報） 1167
TEXTSUPPRESSBOM 1007
TEXTWRITESPACE 1007
TP1 サーバ用 Cosminexus 連携機能 441
TURN 指令 52, 501

TURN 指令と対象の例外名　501
TURN 指令と対象の例外名（EC-I-O 例外名）　501
TURN 指令によるチェック　501
TURN 指令のチェックが ON　486
TURN 指令の有効範囲　502
TYPEDEF 句　53

U

UAC　1217
UCS-2　1217
UCS-4　1217
Unicode　1217
Unicode 機能　650
Unicode 機能での制限事項　683
Unicode 機能の概要　651
Unicode 機能のサポート範囲　653
Unicode 機能の詳細　655
Unicode 機能の前提条件　654
Unicode シグニチャ　1217
Unicode に対応する機能　659
UNSTRING 文　97
UPDATE　555
UPSI-0　366
UPSI-1　366
UPSI-2　366
UPSI-3　366
UPSI-4　366
UPSI-5　366
UPSI-6　366
UPSI-7　366
USAGE 句項目の指定　762
USE 手続き　140
USE 文　514
UTF-16　1217
UTF-8　1218

V

V3 指定メインプログラム　832
VALUE 句のないデータ項目の初期値　923
VARIANT 値と COBOL データのやり取り　599
VARIANT データ項目　76
VERSION 文　1078
Visual Basic から COBOL を呼び出すときの規則437
Visual Basic との連携　437

W

WDISK　346, 1006
Web システム連携機能　41
WHENEVER　555
Windows(x64) COBOL2002 では使用できない機能　1080
Windows(x64) COBOL2002 では使用できないコンパイラオプション　1085
Windows(x64) COBOL2002 では使用できない実行時環境変数　1087
Windows(x64) COBOL2002 では使用できないファイル編成　1080
Windows OS 固有の注意事項　1104
Windows リソース保護　1109
WITH DUPLICATES 指定　287
WRITE FROM　765
WRITE 文〔CSV 編成ファイル〕　162
WRITE 文〔テキスト編成ファイル〕　155
WRP　1109, 1218

X

XDM/RD　591
XDM/SD　581
XDM/SD プログラムのテスト方法の制限事項　589
XDM によるデータベース操作シミュレーション機能　579
XMAP3　837
XML アクセス用ステータス定義　1163
XML アクセス用データ定義　1163
XML アクセスルーチン　1163
XML データ定義ファイル　1164
XML 文書型定義ファイル　1165
XML 連携機能　41

あ

アイコン　856
アイコンデータファイル　1071
アイコンのリソース　1071
値渡し　380
あとに指定した方のオプションが有効となる場合 816
アドレスデータ項目を使った引数の受け取り　363
アプリケーションデバッグ機能　1029
アプリケーションの主プログラム　395, 832, 1218
アプリケーションプログラムを起動する　715
暗黙的なプロパティメソッド　472

い

異常終了　712
異常終了時要約情報リスト　727, 843, 1031
移植性の良いプログラムの作成　766
一意名指定の CALL 文　390
一般規則　815
イベントログファイル出力機能　274
入れ子のプログラム　1218
印刷サービス名称　136
印刷サービス名称の指定　236
印刷書式情報ファイル　1166
印刷書式番号　218, 219
印刷制御付き　837
印刷文書名称　213
インスタンスオブジェクト　459, 1218
インスタンスオブジェクトの消滅　467
インスタンスオブジェクトの生成　466
インスタンス定義　65
インスタンスデータ　1218
インスタンスメソッド　1218
インスタンスメソッドの呼び起こし　468
インタフェース　462, 474, 1218
インタフェースエリアの詳細　584
インタフェース定義　67
インタフェースとクラスの適合　479
インタフェースの実装　476
インタフェースの定義　476
インタフェースの利用　475
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　953
インタフェース領域　637
インタフェース領域の形式　315
インタフェース領域のダンプ出力　333
インタフェースを使ったオブジェクトの使用　477
インポート　1063
インポートライブラリ　1063, 1166
インポートライブラリの生成　1069

う

ウィンドウハンドル　717
受け取り側作用対象　470
内側のプログラム　63, 1219
埋め込み SQL 宣言節　556
埋め込み SQL 文　741, 744, 835
埋め込み SQL 文を使った COBOL プログラムの作成　555
埋め込み型 SQL　1219
埋め込み変数　571
埋め込み例外宣言　556
上書き（WRITE）モード　290

え

英小文字　767
英字項目　305, 731
英数字項目　305, 731
英数字定数の分離符　885
英数字編集項目　305, 731
エイリアス　437
エクスポート　1063
エクスポートファイル　1063
エラーコード値　1131
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　503
エラー表示画面　297, 303
エラーメッセージの形式〔リポジトリ管理ツール〕　966

エラーメッセージの内容〔リポジトリ管理ツール〕 966
エラーリスト 1133, 1162
演算強さの軽減 788
演算の中間結果 92

お

送り出し側作用対象 470
同じオプションを重複して指定した場合の規則 820
オブジェクト 1219
オブジェクト関連づけファイル 1163
オブジェクト参照データ項目 1219
オブジェクト参照の適合チェック 477
オブジェクト指向 456, 1219
オブジェクト指向機能 48, 455
オブジェクト指向機能でのマルチスレッド対応 483
オブジェクト指向でのインタフェース 474
オブジェクト指向による継承 472
オブジェクト指向による適合 477
オブジェクト指向によるポリモルフィズム 482
オブジェクト指向の紹介 456
オブジェクトの構造 457
オブジェクトの消滅 1219
オブジェクトの生成 1219
オブジェクトビュー 481
オブジェクトファイル 790, 1054, 1065, 1166
オブジェクトファイルの追加・変更 1068
オブジェクトファイルの取り出し 1069
オブジェクトプロパティ 470, 1219
オプション 1220
オプション（ライブラリ管理ツール） 1066
オプション（リソースコンパイラ） 1072
オプション（リンカ） 1055
オプション概略の一覧 966
オプション指定項 814
オプションの打ち消し指定 821
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 818
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 816
オプション〔コンパイラ〕 814

か

カーソルオプション 574
カーソル宣言 556
カーソルを移動する 757
ガーベジコレクション 467, 1220
外字 229
開発環境のコンポーネント 45
開発マネージャ 45, 807
開発マネージャからデバッガを起動する場合 658
開発マネージャを使用した場合の登録集原文の検索順序 918
回復可能メッセージ 624
外部スイッチ 124, 366, 1220
外部スイッチの参照方法 367
外部スイッチの初期化 366
外部スイッチの変更 367
外部属性（EXTERNAL 句） 74
外部属性を持つデータ項目の共用 435
外部プログラム 1220
外部変数 1220
外部リポジトリ 941
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 956
カウント情報 1220
カウント情報リストファイル 1164
書き換え（ERASE）モード 290
各実行単位の施錠形式 204, 205
拡張コード文字 896
拡張子 792
拡張子による検索順序 798
拡張子による正書法の指定 793
仮想端末の共用 292
仮想端末名 291, 295
仮想端末名の仮定値を指定する環境変数 292, 295
仮想端末名を指定する環境変数 291, 295
型 1220
カタログファイル 1164

活性化されるプログラム　1220
活性化するプログラム　1220
活性状態　200
カバレージ　46
カバレージ情報　847, 1221
カバレージ情報リストファイル　1164
カプセル化　458, 1221
可変長項目のあとに固定長項目　764
画面節（SCREEN SECTION）による画面機能　296
画面節（SCREEN SECTION および WINDOW SECTION）　738
画面節（WINDOW SECTION）による画面機能 301
画面定義　45
画面定義ファイル　1167
画面データの送受信先　291
画面に対する入出力　289
画面入出力機能　288
画面の座標指定　299
画面の種類と構成　296, 301
画面の定義　289
画面の表示モード　290
環境設定のカスタマイズ　1194
環境変数　912, 1221
環境変数 CBL2KENV_CBLLIB　1195
環境変数 CBL2KENV_LIB　1195
環境変数 CBL2KENV_PATH　1194
環境変数 CBLLIB　1192
環境変数 LIB　1192
環境変数 PATH　1192
環境変数指定　132
環境変数設定時およびカスタマイズ時の注意事項1196
環境変数の設定　1192
環境変数の取り扱い　615
環境変数へのアクセス　269
関数定義　64
管理者権限　1221
管理者権限についての注意事項　1104
管理者ユーザ　1221
管理情報インタフェース領域　316

き

キー数　287
キー属性の指定　286
キー定義ファイル　1166
キーの機能　298, 306
キー比較条件　328
キーボードからの入力　738
規格の互換性をチェックする翻訳指令　50
擬似プログラム用プログラム情報ファイル 852, 1164
既定義オブジェクト参照　1221
既定義オブジェクトを使用したメソッドの呼び起こし　469
機能ごとの移行性と互換性に関する注意事項　1100
基本機能〔Unicode 機能〕　665
基本的な内部操作手続き文　79
キュー　622
キューイング　622
旧形式のオプション　905
キューの種類　622
キューのパス名　623
キューのパス名を検索する　634
キューのパス名を検索するコーディングの例　648
キューのラベル　623
キューをオープンする　628
キューをクローズする　629
キューを削除する　627
キューを削除するコーディングの例　644
キューを作成する　626
キューを作成するコーディングの例　644
境界　762
行制御出力　212
共通式の削除　787
共通実行環境ファイル　1000
共通属性プログラム　404
共通例外処理　49, 484, 547, 1221
共通例外処理に対応している機能　489
共通例外処理の概要　485
共通例外処理の機能　486
共通例外処理の使用例　486

共通例外処理の注意事項　547
共通例外の仕組み　485
共通例外の宣言手続き　514
共有可能属性　74
共有可能属性の初期値　75
共有不可能属性　74
共有不可能属性の初期値　75
共用モード　200
局所場所節　59, 765
局所場所節のデータ領域　71
局所名　73

く

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　669
組み込み関数　677
組み込み関数を使用した例外情報の参照　532
クラス　458, 1221
クラス定義　65, 465
クラス同士またはインタフェース同士の適合　478

け

警告音を鳴らす　756
継承　461, 472, 1221
継承の使い方　473
現行コネクションの変更の例　563
原始プログラムの作成規則　792
原始プログラムリスト　1133, 1144
原始文操作機能　797
原文名　797
原文名定数　797

こ

構造型データベース　581
構造型データベース（XDM/SD）操作シミュレーション　581
構造化例外処理使用時の注意事項　421
高速メッセージ　624
構文規則　814
コード変換失敗時の動作　658
コード変換ライブラリ　654
固定形式拡張子　798
固定形式正書法　792, 793, 915
古典的要素　863
異なった文字集合間の比較　767
コピー伝播　786
コマンド行に指定した引数の個数　713
コマンド行に指定した引数の内容　713
コマンド行に指定する引数の形式　361
コマンド行へのアクセス　263
コマンドファイルを使った引数の指定方法　808
固有情報エリアの詳細　587
コンソールウィンドウ　370
コンパイラ　45
コンパイラオプション　814, 970, 1030
コンパイラオプションの一覧　821
コンパイラオプションの一般規則　815
コンパイラオプションの詳細情報　904
コンパイラオプションの優先順位　815
コンパイラ環境変数　912, 914
コンパイラ環境変数の一覧　912
コンパイラの起動方法　807
コンパイラの制限値　1170
コンパイラ付属機能　927
コンパイル　568, 789, 1222
コンパイル時の主な入出力ファイル　790
コンパイル時の制限事項　683
コンパイル時の入出力の流れ　790
コンパイルリスト　1133
コンパイルリストに関連する翻訳指令　51
コンパイルリストの出力形式を指定する　892
コンパイルリストの出力先　1134
コンパイルリストの出力に関連する翻訳指令　1134
コンパイルリストファイル　1166
コンパイル〔Unicode 機能〕　655
コンポーネントおよびコマンドでの環境設定　1192
コンポーネントの種類　45

さ

サーバの最大待ち時間の設定　181
サービスルーチン　702
サービスルーチンの種類　1114
サービスルーチンを使った OLE2 オートメーションクライアント機能　1113
サービスルーチンを使った引数の受け取り　364
再帰属性プログラム　400
再帰呼び出し　58
再帰呼び出しの例　403
再コンパイル　811
最終実行文情報　1031
最終生成物の種類（プロジェクトの種類）の設定 831
最新例外状態　499
最新例外状態のクリア　538
再生成　811
最外側のプログラム　63, 1222
最適化機能　773
最適化のレベル　774, 841
作業場所節　765
作業場所節のデータ領域　70
索引編成ファイル　147
索引編成ファイルでのレコードの検索基準の相違 177
索引編成ファイルに対する OPEN モード　148
サブオプション　814
サブクラス　1222
差分カバレージ情報　1222
サロゲート　1222
算術演算機能　87
算術演算機能の特徴　87
算術文　88
参照渡し　380

し

識別子　556
シグニチャ　943
実行可能ファイル　359, 417, 790, 1165
実行可能ファイルの起動方法　993
実行可能ファイルの作成単位　801
実行可能ファイルの実行権限についての注意事項1110
実行可能ファイルの指定　418
実行可能ファイル名の指定方法　418
実行可能プログラム　1222
実行環境のコンポーネント　45
実行環境ファイル　1164
実行結果出力ファイル（カウント情報の表示）　1164
実行結果出力ファイル（カバレージ情報の蓄積）1164
実行される宣言手続き　514
実行時エラーメッセージの出力先　1002
実行支援　45
実行時環境変数　994
実行時環境変数 CBLLANG と CBLUNIENDIAN の関係　656
実行時環境変数で行制御を操作する方法　213
実行時環境変数の一覧　995
実行時環境変数の一覧〔64bit アプリケーションの作成〕　1087
実行時のカーソルの位置づけ　290
実行時の制限事項　684
実行時の注意事項　1110
実行時の動的割り当て　136
実行時要素　1222
実行状態を示すコード　291, 294
実行時ライブラリ　45
実行単位　359
実行単位の終了方法　375
実行方式　417
実行〔Unicode 機能〕　656
実装インタフェースの適合チェック　480
指標の繰り返し回数　1044
シフト JIS で入出力する情報　681
シフト JIS 範囲外の文字　671
シミュレーション情報　848
ジャーナリング　625
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　852
自由形式拡張子　798
自由形式正書法　60, 792, 793

自由形式正書法で書かれた COBOL 原始プログラム 916
自由形式正書法で指定できないコンパイラオプション 796
自由形式のソース原文や登録集原文 60
従来形式の例外処理 486, 547
終了キー 750, 754
終了キーとコードとの対応（1 バイトコード） 752
終了キーとコードとの対応（2 バイトコード） 753
終了コード 377, 419, 808, 812, 961
主画面 296, 301
主キーファイル 1165
主プログラムシミュレーション機能 849
主プログラムとして動作させる場合 395
順序の比較 767
順編成ファイル 144
障害調査支援のコンポーネント 45
昇格ダイアログボックス 1222
条件式 81
条件式の評価の流れ 81
条件分岐文 102
条件変数 125
条件翻訳結果のコンパイルリスト 804
条件翻訳に関連する翻訳指令 50
条件翻訳の利用 802
照合順序 877
小数点以下のけた数 762
使用する文 289, 293
使用するメモリサイズ 283
仕様チェックオプションを複数指定した場合 818
使用できるファイル 281
使用できるファイル形式 129
使用できるファイルの種類 194
使用できるファイル編成 128
情報リスト 1133, 1138
少量入出力 252
初期化属性プログラム 398
初期化漏れチェック機能 927
初期状態 393
書式オーバレイ 837
処理時間の短縮 286
処理速度の速いプログラムの作成 760
字類条件 877
字類条件の判断基準と文字の大小関係 877
新規作成 1068
新機能 47

す

数字項目のけた拡張機能 686
数字項目の定義や演算を工夫 760
数字項目の入力方式 304
数字項目の表示形式 303
数字編集項目 304, 731
数字編集項目の入力方式 304
数値の内部表現を意識したコーディング 767
スーパクラス 1222
スキーマ定義 172
スタックコンパイル機能 799
スタック領域に配置されるデータ 770
スタック領域のサイズ変更方法 772
ストアドプロシージャ 561, 1223
スレッドごとに環境変数を設定する 616
スレッドごとに固有の出力ファイル名称を付ける 615
スレッド識別子 614
スレッド識別子の値 613, 615
スレッドを起動する関数 620

せ

制御プログラム 1223
制御変数 1044
制御文字 792
整合性チェック 1041
整合性チェックの警告エラー出力 1042
正書法 50, 793
正書法の決定の優先順位 794
正書法の異なるプログラム間の複写 794
静的なリンク 406, 408
静的に行う方法によるプログラムの例 559

製品体系　41
西暦日付の変更　258
整列作業用ファイル　282
整列処理のメモリサイズ　283
整列併合機能　280, 346
セル　1112
セルデータを数値として入出力する　163
セルデータをタブ文字区切りで入出力する機能　169
宣言手続きからの復帰　517
前バージョンからの変更点　1091

そ

送受信間の物理マップ　292, 295
送受信先の設定方法　291
送信先の設定方法　294
相対位置表示時の原始プログラムリスト　1148
相対編成ファイル　146
添字　761, 1044
ソース解析情報ファイル　1165, 1166
ソース原文の正書法を決定する翻訳指令　50
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　69
ソースの作成方法　792
ソースファイルとリポジトリファイルの関係　947
ソースファイルの拡張子と正書法の種類の対応　793
ソースファイル名　792
ソースファイル名と拡張子　792
そと PERFORM　110
そと PERFORM 文のインライン展開　776
ソフトウェア例外　1046

た

第 1 次規格　40, 863
第 2 次規格　40, 863
第 3 次規格　40, 864
第 4 次規格　40, 864
ダイアログボックスから入力された文字列を受け取る　720
大域属性（GLOBAL 句）　73
大域名　73
大域名と局所名の有効範囲　73
タイトルバー　739
ダイナミックリンク機能　1062
ダイナミックリンク動作　1063
ダイナミックリンクライブラリ　790, 1062
タイムアウト秒数の設定　574
ダイヤモンド継承　1223
互いに参照し合う翻訳単位のコンパイル　953
タグファイル　1166
他言語とのプログラム間連絡　420
多重継承　1223
タブ文字　792
単一レコード施錠　201
単純継承　1223
単体テスト　1223
ダンプリスト　1036

ち

致命的例外　499
中間結果　84
中間結果のけた数　92
中断点情報　848
帳票データの送信先　294
帳票の定義　293
重複キー　179

つ

通算日付の変更　259
通常属性プログラム　398
通常の入出力との互換性（ラージファイル入出力機能）　196
通信節による画面機能　289
強い型付け　1223
強く型付けされた項目　54

て

定義の種類　63
定義別のコンパイル方法とリポジトリファイル　940

定数指定　130
定数指定の CALL 文　389
定数指定の引数　381
定数長拡張機能　1223
定数の畳み込み　788
ディスク書き込みオプション　338
ディスク書き込み保証　343
データ有無コード　290
データ型の定義と参照　53
データ項目の型の対応　381
データコミュニケーション機能　550
データコミュニケーション機能の概要　551
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　553
データソースの管理　566
データ属性の種類　73
データ転記文　118
データ転送　763
データの後の空白文字を出力する機能　168
データの入力方式　304
データの比較　671
データの表示形式　303
データベースアクセス機能　555
データベース管理システム　565
データベース操作機能　554
データベース操作シミュレーション　580
データベース操作シミュレーション機能　580
データベース操作文で内部的に展開される CALL 文　582
データベース操作文で内部的に展開される CALL 文の引数　583
データ名指定　135
データ領域　68
データ領域ダンプの出力条件　1036
データ領域ダンプリスト　1036
データ領域ダンプリストの出力先　1039
データ領域の種類　69
データ例外検出機能　1045
データレコードファイル　1165
適合　477, 1223
適合チェック　944
テキスト編成ファイル　153
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　672
テキスト編成ファイルのファイル編成とレコード形式　153
テストデバッガ　46
手続き部見出しの RAISING 指定　524
手続き文　78
手続き分岐　108
デバッグ環境のコンポーネント　46
デバッグ機能の種類と概要　1030
デバッグ行　843
デバッグ情報　846
デバッグ情報が出力されるタイミング　334
デバッグ情報の出力例　335
デバッグ情報の取得　331
デバッグ情報へ出力されるインタフェース領域の種類　334
デフォルトの印刷文書名称　213
デプロイ情報　1167
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　509

と

動的 SQL　556
動的長基本項目　1224
動的なリンク　406, 409
動的なリンクのプレロード機能　411
動的なリンクのプログラム検索トレース機能　412
動的に行う方法によるプログラムの例　562
登録集環境変数　798, 926
登録集検索フォルダ　798
登録集原文　797, 917, 926, 1224
登録集原文の検索順序　798
登録集原文の正書法　794
登録集原文のファイル形式　797
登録集原文ファイルの検索　794
登録集名　798, 926

特殊レジスタ　285
トラブルシュートに有効な資料　1199
トランザクション　557, 573
トランザクション管理機能　176
トランザクション分離レベル　573
取り消し後の呼び出し　393
取り消し対象のプログラム　391
取り消しで解放される資源　393
取り消しの実行順序　391
トレースオプション　567
トレースバック情報　843, 1032
ドロー　289, 293

な

内部プログラム　1224
内容渡し　380

に

日本語 EUC　1224
日本語項目　305, 731
日本語定数　767
日本語編集項目　305, 731
日本語文字データの転記，比較　767
日本語を含む原始プログラム　768
入出力エラー　287
入出力エラー処理　139
入出力機能〔Unicode 機能〕　672
入出力状態　768
入出力状態の値　140, 1178
入出力データの変換　671
入出力の流れ　790
入力時の未使用項目の初期化機能　167
入力文字とコード　738

の

ノーマルファイル形式　1224

は

ハードウェア例外　1046
配信不能メッセージ　625
排他モード　200
バイトオーダ　1224
バイトオーダマーク　1215
バイトストリーム入出力サービスルーチン　348
バイトストリーム入出力サービスルーチンの概要 349
バイトストリーム入出力サービスルーチンの説明 352
背反関係にあるサブオプション同士を指定した場合　820
廃要素　862, 863
バックアップファイル　1163
パッケージ情報ファイル　1166
パネル定義　289, 293
パラメタ　462
パラメタインタフェース領域　325
バリアントデータ項目　1224
範囲外チェック　860, 862, 863
半角かたかな　766

ひ

比較する項目の長さ　761
引数の受け取り　361
引数の受け取り方法　362, 365
引数の受け渡し　418, 439
引数の受け渡しの規則　381
引数の受け渡しの種類　380
引数の整合性チェック　381
引数の引き渡し方法　421
引数を値渡しで受け取る　426
引数を値渡しで引き渡す　431
引数を一括して取得する方法　268
引数を個別に取得する方法　263
引数をポインタ型で受け取る　425
引数をポインタ型で引き渡す　431
非致命的例外　499
ビッグエンディアン形式　886
日付や時刻を取得する ACCEPT 文　257
ビットデータ　734
表計算プログラムファイル　160

表示速度を重視する　842
表示モード　290
標準エラー出力　131, 133, 135
標準クラス　1225
標準権限　1225
標準コード文字　896
標準出力　131, 133, 135
標準転記による ACCEPT 文　254
標準入出力　369
標準入力　131, 133, 135
標準ユーザ　1225
標準ライブラリ　790, 834
表操作　105
表要素　1225
ビルド　811, 1225
ビルドとリビルド　810

ふ

ファイル／レコード定義　45
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　346
ファイル格納先についての注意事項　1109
ファイル共用　198
ファイル共用を省略した場合の処理　206
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　345
ファイルシェア　198
ファイル属性の整合性チェック　142
ファイル定義ファイル　1165
ファイル入出力機能　127
ファイル入出力文でのエラー情報出力機能　141
ファイルの移行性の注意事項　1096
ファイルの検索順序　418
ファイルのディスク書き込み保証　344
ファイルの排他・共用の区別　203
ファイルの割り当て　282
ファイル編成とレコード形式　151
ファイルレベルのファイル共用　200
ファイル割り当ての共通規則　130
ファクトリオブジェクト　459, 1225
ファクトリ定義　65
ファクトリデータ　1225
ファクトリメソッド　1225
ファクトリメソッドの呼び起こし　468
ファンクションキー入力結果の取得　300
フォルダによる検索順序　798
不活性状態　200
副キーファイル　1165
複数レコード施錠　203
副プログラムシミュレーション機能　850
副プログラムとして動作させる場合　396
復帰コードの値　386
復帰コードの規則　386
復帰コードの参照方法　386
復帰コードの設定方法　386
物理ファイル名　130
物理マップ名　292, 295
浮動継続指示子　835
負の符号を持つゼロ　898
不変式のループ外移動　786
不要な小数点　764
プライマリスレッド　612
プリンタ出力の識別　237
プリンタに対する帳票出力　293
プリンタへのアクセス　208
プリンタへの出力と割り当て方法　226
プログラミング上の留意点　759
プログラム　1225
プログラムが異常終了した場合の終了コード　378
プログラムからの連動実行　1026
プログラム間整合性チェック　1041
プログラム間の引数と返却項目　379
プログラム間連絡　801
プログラム原文領域　768
プログラム互換性アシスタント　1110
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　771
プログラム情報ファイル　846, 847, 920, 1163

プログラム属性　398
プログラム定義　63, 465
プログラム定義だけのコンパイル　955
プログラムテンプレートファイル　1164
プログラムとファイル割り当ての関係　148
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〕　184
プログラムの実行　992
プログラムの実行環境の設定　994
プログラムの実行環境の設定〔64bit アプリケーションの作成〕　1087
プログラムの実行状態　372
プログラムの実行〔64bit アプリケーションの作成〕　1087
プログラムの取り消し　391
プログラムの呼び出し　388, 421
プログラム名　767
プロジェクト　1225
プロジェクト情報ファイル　1165
プロジェクトの概念と定義方法　810
プロジェクトの種類　831
プロジェクトマスタファイル　810, 1165
ブロックカーソル　722

へ

併合処理のメモリサイズ　283
ヘッダの出力形式　1138
ヘッダファイル　447
ヘルプ機能　927
ヘルプ機能（オプション表示）　927
返却項目　385

ほ

報告書作成機能　241, 871
ポップアップ画面　302
ポップアップ画面入出力機能　307
ポップアップブロックカーソル画面　722
ポリモルフィズム　463, 482, 1225
翻訳グループ　62
翻訳グループを構成する定義の種類　61
翻訳指令　50
翻訳単位　62
翻訳単位情報の表示　962
翻訳単位の考え方　64
翻訳変数名　903

ま

マニフェストの埋め込み方法　980
マニュアルの種類　46
マルチスレッド　839, 1226
マルチスレッド環境下でのファクトリデータ　483
マルチスレッド環境下でのメソッドの実行　483
マルチスレッド環境での実行　604
マルチスレッド対応 COBOL プログラム　483, 605
マルチスレッド対応 COBOL プログラムが対応している機能　607
マルチスレッド対応 COBOL プログラムの開始と終了　610
マルチスレッド対応 COBOL プログラムの概要　605
マルチスレッド対応 COBOL プログラムのコンパイル　606
マルチスレッド対応 COBOL プログラムの生成　606
マルチスレッド対応 COBOL プログラムのデバッグ　618
マルチスレッド対応 COBOL プログラムを GUI モードで使用する方法　614
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　619

み

見た目幅　1226

め

明示的なプロパティメソッド　470
明示的な例外の引き起こし　530
メインファイルを指定する　840
メインフレーム　865, 1226
メインプログラムなし　833
メソッド　1226
メソッド原型　1226

メソッド原型定義　67
メソッド定義　66
メソッドに渡す引数　470
メソッドの検索　469
メソッドの呼び起こし　460, 468, 1226
メソッド名　462
メソッド呼び起こしの適合　479
メタキャラクタ　994
メッセージ　623, 1226
メッセージクラス　624
メッセージの配信方法　624
メッセージの優先順位　624
メッセージのラベル　623
メッセージパッシング　460, 468
メッセージを受信する　632
メッセージを受信するコーディングの例　646
メッセージを送信する　630
メッセージを送信するコーディングの例　645
メモリサイズ　283
メモリサイズと処理時間　286

も

モードの決定方法　372
文字コードについての注意事項　1109
文字コードを変換するプログラム例　669
文字の大小関係　877
文字幅を整えた文字列の出力　234
モジュール定義ファイル　790, 855, 1075, 1164
モジュール定義ファイルの記述規則　1075
モジュール定義文　1075
モジュール定義文とリンカオプションとの関係　1078
文字列操作文　95
文字列の出力　232
戻り値　1064, 1070, 1073
戻り値の受け渡し　426, 433
戻り値の使い方　707

ゆ

有効けた数　91
ユーザキーワードファイル　1167
ユーザ指定の印刷文書名称　214
ユーザポップアップ HELP 画面　303
ユーザポップアップ HELP 機能　308

よ

用途 DISPLAY と用途 NATIONAL との間の変換　671
呼び出し規約　404
呼び出し先プログラム　1226
呼び出してはいけないサービスルーチン　619
呼び出し元プログラム　1226
呼び出し元プログラムトレース　1031
弱い型付け　1227
弱く型付けされた項目　54

ら

ラージファイル形式　1227
ラージファイル入出力機能　194
ラージファイル入出力機能でのファイルの共用　196, 206
ラージファイル入出力機能の概要　194
ラージファイル入出力機能の制限事項　197
ライブラリ　977, 1065
ライブラリ管理ツール　1065, 1066
ライブラリ管理ツールオプション　1066
ライブラリの結合　1069
ライブラリの検索　1063
ライブラリの指定　854
ライブラリファイル　1065, 1166

り

リストの出力　1133
リストの見方　1138
リソースコンパイラ　1071, 1072
リソースコンパイラオプション　1072
リソース定義ファイル　790, 856, 1071, 1166
リソース定義文　1074
リソースデータファイル　1071
リソースファイル　790, 1062, 1071, 1166

リトルエンディアン形式　886
リビルド　811
リポジトリ管理ツール　960
リポジトリ管理ツール使用時のエラーメッセージ 966
リポジトリ管理ツールの終了コード　961
リポジトリ段落　1227
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　952
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　952
リポジトリファイル　945, 1166, 1227
リポジトリファイルとリポジトリ段落の関係　947
リポジトリファイルに格納される情報　944
リポジトリファイルの管理　956
リポジトリファイルの検索　950
リポジトリファイルの参照方法　949
リポジトリファイルの生成　945
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　970
リポジトリファイルの生成方法　948
リポジトリファイルの単独生成　953
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　941
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　942
リポジトリファイル〔CBLREP〕　920
リポジトリファイル〔CBLSYSREP〕　921
リポジトリファイル〔-Repository,Gen〕　902
リモートファイルアクセス機能　151
利用者定義関数　56, 65
利用者定義関数の参照　56
利用者定義関数の注意事項　57
利用者定義関数の引数と返却項目　57
利用者定義関数の引数の扱い　57
利用者定義例外名　498
リレーショナルデータベース　591
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　591
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　835
リンカ　858, 1054, 1055
リンカオプション　1054, 1055, 1078
リンクの指定　330
リンケージを行わない　854

る

ループの削除　788

れ

例外　486, 492
例外オブジェクト　498, 537
例外が検出される文　541
例外が検出される文の詳細　540
例外検出での注意事項　543
例外種別　1031
例外情報の参照　532
例外処理　485, 1227
例外処理に関連する翻訳指令　52
例外処理の動作　544
例外宣言　556
例外チェックが無効な場合の動作　503
例外の検出条件　540
例外の自動伝播　521
例外の致命度　499
例外の伝播　520
例外名　492
例外名の一覧　493
例外名のレベル　492
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　527
例外を検出する組み込み関数　540
レコード施錠　201
レコード単位アクセスモード　328
レコード定義ファイル　1166
レコードの検索基準　177
レコード末尾の空白文字を出力する機能　156
レコードレベルのファイル共用　201
連続コンパイル機能　799
連絡節のデータ領域　69

ろ

論理演算　734